目　录

宿迁高铁站

空白文档

激活码激活

空白文档

版本记录

空白文档

起 `echarts-convert` 服务

空白文档

版本记录

启动redis

docker-compose方式安装

学习账号

安装`Prometheus(普罗米修斯)`

数据库缺少字段

空白文档

10.8.26.62 部署完成，算法已升级，清理程序已更新【修改会删除算法图片】，已修改删除策略，已优化AI数据类型

一键部署目录结构

应用服务 WEB 访问端口

盒子错误解决方案

北京盒子hub部署

chatgpt 镜像站汇总

chatgpt 镜像站汇总

个人能力有限，搜集到的不多，求大家多多贡献啊！众人拾柴火焰高！

介绍

汇总所有 chatgpt 镜像站，帮助国内在正常网络下的同学访问gpt。

欢迎大家一起贡献，请直接提交 issue 或者 pr （只能提交普通网络下就可以访问成功的）

无需登录直接可用的：

- https://chat.jinshutuan.com/ - 免费由开发者提供
- https://itedus.cn - 免费但需要关注公众号
- https://openai.run/ - 非会员每日免费额度 10,000 Tokens
- https://chat.darkhorseone.cn/ - 代码一号
- https://chat.aichatos.top/ - 免费由开发者提供(可以开启chatgpt联网功能)
- https://chat.ttext.cn/ - 免费时限制每小时6次提问
- https://chat.wuguokai.cn - 免费由开发者提供
- https://chatgpt.nicelinks.site/ - 免费使用频次有限制
- http://ai.yscxy.net/ - 免费由开发者提供
- https://chat.binjie.site:7777 - 自动跳转 https://chat.jinshutuan.com/
- https://chat.jinshutuan.com/ - 免费由开发者提供
- https://chatgpt.0voice.com/#/chat - 免费由零声教育内部提供使用
- https://aiweb.douguguo.com/ - 默认同一ip只提供一次免费查询机会
- https://www.ebo168.com/ - 免费由开发者提供(有广告维持服务)
- https://chat.extkj.cn/ - 访问404 免费由开发者提供(有广告维持服务)
- https://beezy.cool/ - 官网下载插件使用
- https://chat.aidutu.cn - 免费需要关注微信公众号
- https://chatgpt.qdymys.cn/ - 免费有调用频率限制
- https://huggingface.co/chat/ - 国外开源可访问模型(默认用英文回复,当前时间无法理解上下文)
- https://h2o.ai/ - 国外开源可访问模型(可以使用中文回复,当前时间无法理解上下文)
- https://chatgptmirror.com/ - 前三次无需登录免费(之后需要登录)
- https://bestai.litongxue.icu/ - 免费体验有次数限制
- https://chat.08qt.com - 免费可用
- https://www.pmmaster.cc - 免费需要关注微信公众号,也可以选择使用自己的API-KEY / (导航栏有访问路径)
- https://gpt.368ai.cn/ - 免费试用三次提问
- https://chatanywhere.top/ - 免费试用
- http://ai.wikll.com - 暂时无法访问(2023/05/22)

- http://chat.iisai.cn - 暂时无法访问(2023/05/22)
- http://mychatos.top/#/chat - 暂时无法访问(2023/05/22)
- https://chatbot.theb.ai/ - 暂时无法访问(2023/05/22)
- https://openai.nm.cn/ - 暂时无法访问(2023/05/22)
- https://chat-shared.zhile.io/shared.html - 暂时无法访问(2023/05/22)
- https://chat.gpthink.top/#/chat - 暂时无法访问(2023/05/22)
- http://duckling.buzz/ - 暂时无法访问(2023/05/22)
- https://chat.openai.run/ - 暂时无法访问(2023/05/22)
- https://chatmindai.cn - 暂时无法访问(2023/05/22)
- https://chat.getgpt.com.cn/ - 暂时无法访问(2023/05/22)
- https://studyai.life/ - 暂时无法访问(2023/05/22)
- https://chat1.aichatos.com/ - 暂时无法访问(2023/05/22)
- https://chat.wikll.com/ - 暂时无法访问(2023/05/22)
- https://gpt.mqgggg.top/ - 暂时无法访问(2023/05/22)
- https://openai.tdp.icu - 暂时无法访问(2023/05/22)
- https://infiniteai.chat - 暂时无法访问(2023/05/22)
- http://s.wikll.com - 暂时无法访问(2023/05/22)
- http://huangkz.chat - 暂时无法访问(2023/05/22)
- https://ai.zhangsan.cloud/ - 暂时无法访问(2023/05/22)
- https://chat.acheckling.com.cn/ - 免费由开发者提供
- 这个真的不建议使用，有点膈应人 - WuchenChina提供

登录后可以免费使用的：

- https://ai.gcchen.cn/ - 星云 AI
- http://chat.apeto.cn/ - Genius
- https://ai.yunweikc.com/ - 昀为AI
- https://chatbot.js.cn - AI小助理
- http://chat.forwardminded.xyz/
- https://gezhe.com/ - 歌者AI
- https://app.haitunchat.com/login/
- http://chatgpt.bainaark.top/index.php
- https://chat.mac89.com/
- https://chatgpt.ctfcode.com/#/register
- http://chat.niitcxl.cn
- https://chat.douresources.com - 需要微信登录、聊天免费、支持上下文关联、有AI绘画
- http://www.xckfsq.com/index/chatgpt - 信创开放平台
- https://zhihuibao.pro/ - 智慧宝注册赠送36000免费token, 之后需要付费. 可以使用gpt-3.5-turbo和gpt-4
- https://joychat.cceven.cc/ - JoyChat 悦聊助手, 支持上下文关联
- https://itgpt.owley.co/ - gpt知识库
- https://hugai.vip/ - chatgpt3.5(为防止被封使用文心一言的名称)
- http://chatsite.chat/ - 暂时无法访问(2023/05/22)
- http://www.gpt-smart.com/ - 暂时无法访问(2023/05/22)
- https://front2.stargpt.top/#/chat - 暂时无法访问(2023/05/22)
- https://dittin.com/ - 暂时无法访问(2023/05/22)
- https://chat.yokonsan.com/ - 暂时无法访问(2023/05/22)
- https://gpthink.xyz/#/chat - 暂时无法访问(2023/05/22)
- https://gpt.fly2you.cn/web/ - 暂时无法访问(2023/05/22)

- https://gpt.opengpt88.com/ - 暂时无法访问(2023/05/22)
- http://ai.apemangpt.com/ - 暂时无法访问(2023/05/22)
- https://ai.shenhuo.net/ - 暂时无法访问(2023/05/24)

需要付费的：

- https://chatgpt-cn.co - 微信注册，微信/支付宝付款，按Token收费
- https://ai4u.cloud - 自研的联网AI，支持联网上下文，根据实时联网数据清洗算法打造
- https://www.openai-asia.com - 国内邮箱注册支持支付宝付款
- https://static-mp-1bc4f84b-cb57-4859-b32d-1ff3a3b3a3f1.next.bspapp.com/ - 建议手机访问
- https://tryai1.github.io/
- https://god.supperjoy.online
- https://yanghanwen.xyz/ - 暂时无法访问(2023/05/22)

需要自己提供 **key** 的：

- https://chat-shared.zhile.io/ - 暂时无法访问(2023/05/22)
- http://www.aihu003.me/ - 暂时无法访问(2023/05/22)
- https://gpt.dizent.cn/ - 暂时无法访问(2023/05/22)
- https://chat.okis.dev/zh-CN?mode=chat - 暂时无法访问(2023/05/22)
- https://ai.pmaigc.com/ - 暂时无法访问(2023/05/22)

开源的，可以自己部署到本地的：

- https://gitee.com/lboot/lucy-chat
- https://github.com/Chanzhaoyu/chatgpt-web
- https://gitee.com/aniu-666/chat-gpt-website
- https://github.com/hncboy/chatgpt-web-java
- http://be.apeto.cn/archives/shang-ye-ban-chatgpt - 商业可部署chatgpt项目(演示地址:http://chat.apeto.cn)

插件：

- https://www.wetab.link/zh/ - Edge浏览器插件（邮箱登录后免费使用）
- https://chatgpt.cn.obiscr.com/ - 在 idea 上使用 chatgpt
- https://monica.im/ - Chrome或Edge浏览器安装插件

国内自研大模型汇总：

- https://yiyan.baidu.com/ - 文心一言，百度出品
- https://tongyi.aliyun.com/ - 通义千问，阿里出品
- https://techday.sensetime.com/ - SenseTime，商汤科技出品
- https://tiangong.kunlun.com/ - 天工AI助手，昆仑万维集团
- https://xinghuo.xfyun.cn/ - 星火认知大模型，讯飞
- https://moss.fastnlp.top/ - Moss，复旦团队出品
- https://www.so.com/zt/invite.html - 360智脑，360出品
- https://github.com/THUDM/ChatGLM-6B - ChatGLM-6B，清华大学唐杰团队开发

国外其它大模型汇总：

- https://talk.truthgpt.one/ - 马斯克出品，无需魔法，无需注册账号，直接使用。
- https://bard.google.com/ - 谷歌出品，需魔法。
- https://slack.com - 使用流程参考

其它多模态 **AI** 技术：

- https://github.com/CompVis/stable-diffusion - Stable Diffusion，AI 画图 [开源]
- https://github.com/getcursor/cursor - Cursor，使用Ai来辅助编程 [开源]
- https://writesonic.com/ - AI 帮你文案策划
- https://yige.baidu.com/ - 文心一格，AI 作图，百度出品
- https://www.midjourney.com/ - Midjourney，AI 画图
- https://gamma.app/ - Gamma，AI 帮你生成 PPT
- https://fliki.ai/ - 暂时无法访问(2023/05/22)

AI工具集导航网站**:**

- https://ai-bot.cn/ - 第一次加载较慢
- https://www.aiyjs.com/ - AI工具集导航网站
- https://www.aigc.cn/ - AIGC 导航网站
- https://www.ainav.net/ - AI工具集导航网站
- https://c.runoob.com/ai/ - 菜鸟工具

空白文档

安装

安装

```
#版本安装
sudo curl -L https://github.com/docker/compose/releases/download/1.29.0/doc
chmod +x /usr/local/bin/docker-compose

#bash 补全命令
curl -L https://raw.githubusercontent.com/docker/compose/1.27.4/contrib/com
```

```
root@ipc-H410M-S2-V2:/aibox/docker_compose# ./build.sh
## ------ 一键化部署【开始】 ------ ##
## ------ load 镜像包 ------ ##
## ------ 停止docker-compose服务 ------ ##
## ------ 构建docker-compose服务 ------ ##
## ------ 启用docker-compose服务 ------ ##
## ------ 一键化部署【结束】 ------ ##
```

```
version: '3'    --首行,描述执行docker-composer的规则版本号，现在用我的是3版本
services:       --例出我要编排的所有服务，即把每个容器看作一个服务，取一个别名
  nginx:        --我给前端服务取名nginx
    restart: always                 --这里表示如果容器意外退出，会让它自动重启
    image: 192.168.1.14/material-management/nginx:latest    --来自我们基础镜
    container_name: mynginx          --取一个容器名
    network_mode:  "service:nacos"   --加入nacos这个服务的网络，它们用同一IP，这
    privileged: true             --让容器用特权，这样挂载的目录或文件进行创建，执
    cap_drop:                    --下面也是权限相关，用于介绍 关闭所有权限并赋予特
      - ALL
    cap_add:
      - CHOWN
      - DAC_OVERRIDE
      - SETGID
      - SETUID
    volumes:                        --挂载特定目录或文件
      - /data/nginx/etc/nginx.conf:/etc/nginx/nginx.conf    --如果是文件，记录
      - /data/nginx/log/:/var/log/nginx
      - /data/nginx/www/:/var/www
  nacos:
    image: 192.168.1.14/common/nacos:fengshun
    container_name: mynacos
    restart: always
    networks:
      - mynetwork
    ports:                      --开放这些端口，因为我让下面这些容器都加入这个网络
      - 80:8080
      - 8080:8080
      - 8081:8081
      - 8088:8088
      - 8848:8848
      - 6379:6379
      - 9876:9876
      - 10909:10909
```

```
      - 10911:10911
rocketmq-server:             --rocketmq的命名服务
   image: 192.168.1.14/common/rocketmq:namesvr
   container_name: rocketmq-server
   restart: always
   volumes:
      - /data/rocketmq-server/:/etc/rocketmq/
   network_mode:  "service:nacos"
rocketmq-broker:
   image: foxiswho/rocketmq:broker
   container_name: rocketmq-broker
   restart: always
   environment:
      NAMESRV_ADDR: "localhost:9876"
      JAVA_OPTS: " -Duser.home=/opt"
      JAVA_OPT_EXT: "-server -Xms128m -Xmx128m -Xmn128m"
   command: mqbroker -c /etc/rocketmq/broker.conf
   volumes:
      - /data/rocketmq-broker/broker.conf:/etc/rocketmq/broker.conf
   network_mode:  "service:nacos"
   depends_on:
      - rocketmq-server     --等rocketmq的命名服务启动后再创建
minio:
   image: 192.168.1.14/common/minio:fengshun
   container_name: myminio
   restart: always
   cap_add:
      - ALL
   volumes:
      - /data/miniodata:/data
   network_mode:  "service:nacos"
redis:
   image: 192.168.1.14/common/redis:latest
   container_name: myredis
   restart: always
   network_mode:  "service:nacos"
   volumes:
      - /data/redisdata:/data
file-server:
   image: 192.168.1.14/material-management/file-server:pre-1.2.6-SNAPSHOT
   container_name: file-server
   restart: always
   volumes:
      - /data/logs/filelog/:/tmp/mclog/
   network_mode:  "service:nacos"
   privileged: true
   cap_add:
      - ALL
   depends_on:
      - minio     --等minio服务启动后再创建
      - nacos    --等nacos的注册中心启动后再创建
msg-server:
   image: 192.168.1.14/material-management/message-server:pre-1.2.9-SNAPS
   container_name: msg-server
   restart: always
   volumes:
      - /data/logs/msglog:/tmp/mclog/
   network_mode:  "service:nacos"
   privileged: true
   cap_add:
```

```
        - ALL
    depends_on:
        - redis
        - nacos
  log-server:
    image: 192.168.1.14/material-management/log-server:pre-1.2.9-SNAPSHOT
    container_name: log-server
    restart: always
    privileged: true
    cap_add:
        - ALL
    volumes:
        - /data/logs/loglog/:/tmp/mclog/
    network_mode:  "service:nacos"
    depends_on:
        - nacos
  gateway-server:
    image: 192.168.1.14/material-management/gateway-server:pre-1.2.9-SNAPS
    container_name: gateway-server
    restart: always
    privileged: true
    volumes:
        - /data/logs/gatewaylog/:/tmp/mclog/
    network_mode:  "service:nacos"
    cap_add:
        - ALL
    depends_on:
        - nacos
networks:     --创建一个docker 网络
  mynetwork:
    name:  'mynetwork'     --命名网络
    external: false
    driver: bridge          --采取桥接，还有HOST，就是跟随宿主机的IP
```

Docker常用命令

Docker常用命令

`docker version` 查看docker版本信息

`docker info` 查看docker系统信息 `system` 级别

`docker help` 查看帮助信息 命令

`docker` CLI文档 & 帮助文档

镜像命令

`docker images`

```
Jeffrey@Jeffrey  ~  docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
mysql        latest    3218b38490ce   4 weeks ago     516MB
centos       latest    e6a0117ec169   4 months ago    272MB
```

解释

- `REPOSITORY` 镜像的仓库源
- `TAG` 镜像的标签
- `IMAGE ID` 镜像的ID
- `CREATED` 镜像的创建时间
- ` `SIZE` 镜像的大小

**可选项 **

```
  -a, --all             #列出所有的镜像
  -q, --quiet           #只显示镜像的id
```

搜索镜像 **`docker search`**

```
Jeffrey@Jeffrey  /  docker search mysql
NAME                 DESCRIPTION
mysql   MySQL is a widely used, open-source relation…   11997     [OK]
mariadb MariaDB Server is a high performing open sou…   4595      [OK]
......
```

**可选项 **

```
  -f, --filter filter  #过滤   docker search --filter=STARS=3000   过滤STARS
```

下载镜像 **`docker pull`**

> `docker pull 镜像名 [:tag]` 指定版本，不写TAG 默认最新的

```
Jeffrey@Jeffrey  /  docker pull mysql
Using default tag: latest
latest: Pulling from library/mysql
latest: Pulling from library/mysql
no matching manifest for linux/arm64/v8 in the manifest list entries
#由于Mac M1 是M1芯片，官方包支持，所以过滤搜索支持的
#==========================================================================
Jeffrey@Jeffrey  /  docker pull --platform linux/x86_64 mysql
Using default tag: latest
latest: Pulling from library/mysql
72a69066d2fe: Pull complete                    #docker image的核心 =》 分层
93619dbc5b36: Pull complete
99da31dd6142: Pull complete
626033c43d70: Pull complete
37d5d7efb64e: Pull complete
ac563158d721: Pull complete
d2ba16033dad: Pull complete
688ba7d5c01a: Pull complete
00e060b6d11d: Pull complete
1c04857f594f: Pull complete
4d7cfa90e6ea: Pull complete
e0431212d27d: Pull complete
Digest: sha256:e9027fe4d91c0153429607251656806cc784e914937271037f7738bd5b8e
Status: Downloaded newer image for mysql:latest
docker.io/library/mysql:latest #docker的真实地址

# docker pull --platform linux/x86_64 mysql ===等价于=== docker.io/library/m
```

删除镜像 **`docker rmi -f`**

```
docker rmi -f 18e5af790473  # 根据容器id进行删除
#docker rmi -f 容器id 容器id2 容器id3 容器di4
docker rmi -f $(docker images -aq) #查询出所有的镜像id   进行递归批量**容器命令**
```

容器命令

> 说明：有了镜像才可以创建容器，linux，下载一个centos镜像测试学习*

创建/运行容器

> `docker pull centos`
>
> `docker run [可选参数] image`

```
#创建容器
docker pull centos
#运行容器
docker run [可选参数] image
#常用参数
--name="Name"    容器名称  tomcat01
```

```
-d                                                          后台方式运行
-i -t                                              交互方式运行，进入容器查看内容
-p                                                          指定容器的端口：8080:
                  -p       IP:主机端口：映射容器端口
                  -p       主机端口：映射容器端口（常用）
                  -p       容器端口
                  容器端口
-P                                                          随机指定端口

#例子
Jeffrey@Jeffrey > docker run -it centos /bin/bash
[root@b370ddb8872d /]# ls
```

查看容器

`docker ps`

```
#列出所有运行的容器
 Jeffrey@Jeffrey  /  docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
# docker ps
-a                                                          列出当前运行的容器+历史
-n=Num                                             列出最进运行过的容器
-q                                                          只显示容器的编号
-aq                                                         只显示（当前运行+历史运
```

退出容器

```
#直接退出容器，容器停止
[root@b370ddb8872d /]# exit
#退出容器，容器不停止
Ctrl + P + Q
```

删除容器

`docker rm -f $(docker ps -aq)` || `docker ps -a -q|xargs docker rm`

```
                                                    #删除指定的容器，不能删除正在运
docker rm -f $(docker ps -aq)                      #删除所有容器
docker ps -a -q|xargs docker rm           #删除所有容器
```

启动/停止容器

```
docker start 容器ID                                                 启动容器
docker restart 容器ID                                       重启当前正在运行的容器
docker stop 容器ID                                                  停止当前正在运
docker kill 容器ID                                                  杀掉当前正在运
```

常用其他命令

后台启动容器

`docker run -d centos`

```
#docker run -d 镜像名称
Jeffrey@Jeffrey  ~  docker run -d centos
bf8c16435567538a9f2052803ac3f2735af1071c5e062a962b875d6ddfa00b3f
Jeffrey@Jeffrey  ~  docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

#注意：docker ps 发现容器停止了
#常见的坑：docker容器使用后台运行，就必须有一个前台进程，docker发现没有应用，就会自动停
#以上的centos 在容器启动后，发现自己没有提供服务，就会立刻停止
```

查看日志

`docker logs -t -f --tail [:number] 容器id`

```
docker logs -t -f --tail [:number] 容器id

## 显示日志
-tf                                          #显示所有日志
--tail nums               #显示指定数量的日志信息

###测试 | 编写一段脚本进行测试

Jeffrey@Jeffrey  ~  docker run -d centos /bin/sh -c "while true;do echo Jef
177dd25b70615257abab44a241797fa7e226a22fd1b1e9c2909c6c4fdfd42535
Jeffrey@Jeffrey  ~  docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS
177dd25b7061   centos    "/bin/sh -c 'while t…"   4 seconds ago   Up 3 secc
Jeffrey@Jeffrey  ~  docker logs -tf --tail 10 177dd25b7061
2022-01-21T07:22:50.346652796Z Jeffrey
2022-01-21T07:22:51.353023921Z Jeffrey
2022-01-21T07:22:52.361788380Z Jeffrey
2022-01-21T07:22:53.371457547Z Jeffrey
2022-01-21T07:22:54.376483631Z Jeffrey
2022-01-21T07:22:55.380710007Z Jeffrey
2022-01-21T07:22:56.384901757Z Jeffrey
2022-01-21T07:22:57.395018841Z Jeffrey
2022-01-21T07:22:58.405004341Z Jeffrey
2022-01-21T07:22:59.413037967Z Jeffrey
2022-01-21T07:23:00.420680509Z Jeffrey
.......
```

查看容器中的进程信息

`docker top 容器ID`

```
docker top 容器ID

Jeffrey@Jeffrey  ~  docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS
177dd25b7061   centos    "/bin/sh -c 'while t…"   6 minutes ago   Up 6 minu
 Jeffrey@Jeffrey  ~  docker top 177dd25b7061
UID    PID   PPID   C  STIME  TTY   TIME           CMD
root   4381  4356   0  07:22  ?   00:00:00  /bin/sh *********  sleep 1;done
root   4832  4381   0  07:29  ?   00:00:00  /usr/bin ********* leep /usr/bi
```

查看容器元数据信息

> `docker inspect 容器ID`

```
Jeffrey@Jeffrey  ~  docker inspect 177dd25b7061
[
    {
        "Id": "177dd25b70615257abab44a241797fa7e226a22fd1b1e9c2909c6c4fdfd4
        "Created": "2022-01-21T07:22:35.987721595Z",
        "Path": "/bin/sh",
        "Args": [
            "-c",
            "while true;do echo Jeffrey;sleep 1;done"
        ],
        "State": {
           `*******`
        },
        "Image": "sha256:e6a0117ec169eda93dc5ca978c6ac87580e36765a66097a6bf

        `********`

        }
    }
]
```

进入当前运行的容器

> `docker exec -it 容器ID baseShell`
>
> `docker attach 容器ID`
>
> *区别： `exec` 进入容器后，打开一个新的终端（常用)*
>
> *`attach`* 进入容器正在运行的终端，不会启动新的线程

```
Jeffrey@Jeffrey  ~  docker ps
CONTAINER ID   IMAGE     COMMAND    CREATED          STATUS          PORTS
177dd25b7061   centos    "/bin/…"   20 minutes ago   Up 20 minutes
Jeffrey@Jeffrey  ~  docker exec -it 177dd25b7061 /bin/bash
[root@177dd25b7061 /]# ls
bin  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc  root  r
```

从容器内拷贝文件到主机

> `docker cp 容器ID:容器内路径 目标主机路径`

```
#容器内创建文件
Jeffrey@Jeffrey  ~  docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS          POR
c5faef731d6e   centos    "/bin/bash"   25 seconds ago   Up 24 seconds
#进入docker容器内部
Jeffrey@Jeffrey  ~  docker attach c5faef731d6e
[root@c5faef731d6e /]# cd home/
[root@c5faef731d6e home]# ls
[root@c5faef731d6e home]# touch test.php
[root@c5faef731d6e home]# exit    #退出不影响
```

```
#进行文件Copy
Jeffrey@Jeffrey  ~  docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED              STATUS
c5faef731d6e   centos    "/bin/bash"   About a minute ago   Exited (127) 37
bf8c16435567   centos    "/bin/bash"   46 minutes ago       Exited (0) 46 m
2c924d3240f0   centos    "/bin/bash"   58 minutes ago       Exited (0) 57 m
Jeffrey@Jeffrey  ~  docker cp c5faef731d6e:/home/test.php /Users/Jeffrey/Ph
Jeffrey@Jeffrey  ~  cd /Users/Jeffrey/PhpWwwRoot/dingoApi
Jeffrey@Jeffrey  ~/PhpWwwRoot/dingoApi  ll
total 0
-rw-r--r--  1 Jeffrey  staff     0B Jan 21 16:03 test.php

## 拷贝是一个手动的过程，未来我们使用-v数据卷的技术，实现自动同步
```

Docker安装Nginx

```
#搜索镜像
docker search nginx
#下载镜像
docker pull nginx
#运行nginx容器  -d 后台运行  --name 命名 nginx01   -p 宿主机:容器内部端口
docker run -d --name nginx01 -p 3344:80 nginx
#==========================实例==============================
Jeffrey@Jeffrey  ~  docker run -d --name nginx01 -p 3344:80 nginx
940119dd19390a463a44c8037f8cccc9dc8ea3d90df94d92d82dfecb46e45be1
Jeffrey@Jeffrey  ~  docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS
940119dd1939   nginx     "/docker-entrypoint.…"   4 seconds ago   Up 3 seco
Jeffrey@Jeffrey  ~  curl 127.0.0.1:3344
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
 Jeffrey@Jeffrey  ~
```

> 图解：端口暴露的概念：相当于端口号的映射

image-20220125155836829

Docker安装Tomcat

> 官方使用案例 `docker run -it --rm tomcat:9.0`
>
> **–rm** 一般用来测试，用完即删除 不加–rm 在容器停止之后，容器还可以查到 除了测试不建议使用

```
#下载最新版
docker pull tomcat
#启动运行
docker run -d -p 3345:8080 --name tomcat01 tomcat
#http://127.0.0.1:3345/可以访问，但是报错404

#进入容器
docker exec -it tomcat01 /bin/bash

#发现问题
# 1、linux命令不全
# 2、webapps目录下年是空的  【镜像的原因：默认最小镜像，所有必须要的内容都剔除掉了】
# 保证最小可运行的环境
# 发现webapps.dist目录
root@a95b956bc409:/usr/local/tomcat# ls -al
total 176
drwxr-xr-x 1 root root  4096 Dec 21 20:39 .
drwxr-xr-x 1 root root  4096 Dec 21 20:31 ..
-rw-r--r-- 1 root root 18994 Dec  2 22:01 BUILDING.txt
-rw-r--r-- 1 root root  6210 Dec  2 22:01 CONTRIBUTING.md
-rw-r--r-- 1 root root 60269 Dec  2 22:01 LICENSE
-rw-r--r-- 1 root root  2333 Dec  2 22:01 NOTICE
-rw-r--r-- 1 root root  3378 Dec  2 22:01 README.md
-rw-r--r-- 1 root root  6905 Dec  2 22:01 RELEASE-NOTES
-rw-r--r-- 1 root root 16517 Dec  2 22:01 RUNNING.txt
drwxr-xr-x 2 root root  4096 Dec 21 20:39 bin
drwxr-xr-x 1 root root  4096 Jan 25 09:05 conf
drwxr-xr-x 2 root root  4096 Dec 21 20:39 lib
drwxrwxrwx 1 root root  4096 Jan 25 09:05 logs
drwxr-xr-x 2 root root  4096 Dec 21 20:39 native-jni-lib
drwxrwxrwx 2 root root  4096 Dec 21 20:39 temp
drwxr-xr-x 2 root root  4096 Dec 21 20:39 webapps
drwxr-xr-x 7 root root  4096 Dec  2 22:01 webapps.dist
drwxrwxrwx 2 root root  4096 Dec  2 22:01 work

#将webapps.dist目录下的内容  全部拷贝到webapps目录下
root@a95b956bc409:/usr/local/tomcat# cp -r webapps.dist/* webapps

#访问 http://127.0.0.1:3345/ 正常访问  Tomcat
```

Docker 安装 ES + Kibana

```
# es 暴露的端口很多
# es 十分的耗内存
# es 的数据一般需要放在安全目录，需要用到挂载技术
# --net somenetwork  网络配置
$ docker run -d --name elasticsearch --net somenetwork -p 9200:9200 -p 9300
# 不限制内存的情况下，特别占内存，服务器会特别卡，下面启动是限制了内存的操作
```

```
#启动es
$ docker run -d --name elasticsearch  -p 9200:9200 -p 9300:9300 -e "discove
```

```
docker run -d \
--restart unless-stopped --name btpanel \
-p 8888:8888 \
-p 22:22 \
-p 443:443 \
-p 80:80 \
-p 888:888 \
-v /Users/Jeffrey/DockerMapping/lnmp/wwwroot:/www/wwwroot \
-v /Users/Jeffrey/DockerMapping/lnmp/mysql/data:/www/server/data \
-v /Users/Jeffrey/DockerMapping/lnmp/nginx/vhost:/www/server/panel/vhost \
--privileged=true
btpanel/btpanel:lnmp

docker run -d \
--restart unless-stopped --name btpanel \
-p 8888:8888 \
-p 22:22 \
-p 443:443 \
-p 80:80 \
-p 888:888 \
-v /Users/Jeffrey/DockerMapping/lnmp/wwwroot:/www/wwwroot \
-v /Users/Jeffrey/DockerMapping/lnmp/mysql/data:/www/server/data \
-v /Users/Jeffrey/DockerMapping/lnmp/nginx/vhost:/www/server/panel/vhost \
--privileged=true
btpanel/btpanel:lnmp
```

```
docker run -dit \
-p 80:80 \
-p 443:443 \
-p 3306:3306 \
-p 9000:9000 \
-v /Users/Jeffrey/DockerMapping/lnmp/wwwroot:/www \
-v /Users/Jeffrey/DockerMapping/lnmp/mysql/data:/data/mysql \
-v /Users/Jeffrey/DockerMapping/lnmp/mysql:/etc \
-v /Users/Jeffrey/DockerMapping/lnmp/nginx:/usr/local/nginx \
-v /Users/Jeffrey/DockerMapping/lnmp/php7:/usr/local/php7 \
--privileged=true \
--name=lnmp \
2233466866/lnmp

/Users/Jeffrey/DockerMapping/lnmp/wwwroot:/www
/Users/Jeffrey/DockerMapping/lnmp/mysql:/etc/
/Users/Jeffrey/DockerMapping/lnmp/nginx:/usr/local/nginx
/Users/Jeffrey/DockerMapping/lnmp/php7:/usr/local/php7

docker run -d \
--restart unless-stopped --name btpanel \
-p 8888:8888 \
-p 22:22 \
-p 443:443 \
-p 80:80 \
-p 888:888 \
-v /Users/Jeffrey/DockerMapping/lnmp/wwwroot:/www/wwwroot \
-v /Users/Jeffrey/DockerMapping/lnmp/mysql/data:/www/server/data \
-v /Users/Jeffrey/DockerMapping/lnmp/nginx/vhost:/www/server/panel/vhost \
btpanel/btpanel:lnmp
```

音柱节点

```
docker run -idt --name ccc -v /etc/localtime:/etc/localtime:ro -v /var/run/
```

```
docker run -idt --name yinzhu --net=host yinzhu:v3.1
```

WorkFlow {#workflow}

Git 分支规范

> - 活跃项目：近 2 个月内 GitLab 有代码提交
> - 活跃分支：2022-04-01 后有 push 或 merge 的分支

WorkFlow

首先，强烈建议学习一下阮一峰的 **Git** 工作流程，以及文中提到的相关文章。

常见的 WorkFlow 有三种：

- Git Flow - 适用于单个人维护的项目
- GitHub Flow - 适用于比较简单的项目，迭代不频繁、发布环境/方式单一
- GitLab Flow - 能适用于各种场景

根据实际情况，每个团队/项目可以选择上面三种 WorkFlow 之一，不能过于随意。

无论选择上面哪种 WorkFlow，都要注意 保持 **`master`** 到 **`beta`** 、 **`release`** 等分支的同步，尽量使用 merge 而不是 cherry-pick，因为后者一但有遗漏会导致两份代码存在差异。

一些原则：

1. 线上发生 **bug** 时，从发生 **bug** 的 **tag** 开出 **`bugfix`** 分支
   修复后 merge 回原分支以及受影响的其它 `master` 、 `release` 以及 `beta` 等分支
2. **`feature`** 分支一般从 **`release`** 开出
   开发完成之后 merge 回 `release` 分支，上线后通过 `release` 分支 merge 回 `master`
   上线前 `feature` 分支的 bugfix 视为开发过程的一部分，在 `feature` 分支完成
3. **`bugfix`** 、 **`feature`** 等分支一般不再开出子分支
4. 除 **`master`** 以外， **`develop`** 、 **`test`** 、 **`beta`** 、 **`release`** 等其他任何分支不要长期使用
   例如长期持续使用 `test/pro` 、 `beta/pro` 、 `release/pro` 做 pro 环境的测试和发布，会导致它们可能与 `master` 分支存在差异、并且差异日积月累地增多，从而引发不必要的问题，正确的做法是：
5. 每个版本开始时，从 `master` 开出 `test/pro-22.9.1` 、 `beta/pro-22.9.1` 、 `release/pro-22.9.1` 等分支
6. 个人及 feature 分支 merge 到 test

7. 测试通过后从 feature merge 到 beta —— 因为 test 分支可能比较混乱
8. 上线时从 beta merge 到 release —— 确保 beta 环境和线上代码一样
9. 上线之后在 release 分支打 tag（例如 `release/pro-22.9.1` ）、merge 回 `master` 、然后删掉

{width=680px}

分支管理

为了配合 代码检查 以及适应各种 Git WorkFlow，Git 分支需要符合以下规范。

1. 开发分支

- 功能开发、bugfix、代码优化等开发分支应符合以下格式，开发完成后通过 merge request 合并到测试/发布分支
- 要求每个分支只能单一，不要把多个需求、以及与需求不相关的 bugfix 混在一个分支里
- 一般情况下禁止合并自己的 merge request

人员维度：

- 个人分支：`<nickname>/<featrue-name | patch-name>` ，例如 `ming/group-managers`

类型维度：

- 需求分支：`[featrue | fea]/<featrue-name>`，例如 `featrue/group-managers`
- 修复分支：`[patch | hotfix | fix | bugfix | bug | ...]/<patch-name>`，例如 `fix/my-page-columns`

> 这些分支上的 push 可以不触发代码检查，但 无法作为 **merge request** 目标分支。

2. 公共分支

用于测试、发布的保护分支要设置为保护分支：

- root 分支：不带 `/` 的分支，建议使用规范命名，数量不超过 **5** 个，例如 `master` / `main`、`develop`、`test`、`alpha`、`beta`、`release` 等
- 测试/发布分支：带有前缀的分支 `[test | release | stable] / [<version> | <environment> | <customer-name>]`，例如 `release/1.2.0`、`release/latest`、`release/PuXin`、`test/beta`、`test/HuaTu`

> 这些分支上的 **push/merge** 以及 **merge request** 需要触发代码检查，`test/*` 可选。

3. 保护分支设置

> 设置：Project > Settings > Repository > Protected Branches

规范/安全模式（推荐）： > push 操作仅应急时使用 | | Allowed to merge | Allowed to push | | ------- | :---------------------- | :-- | | master | Developers + Maintainers | Maintainers （仅用于紧急情况） ~~(ci msg + " - IKNOWWHATIAMDOING")~~ | | develop | Developers + Maintainers | Maintainers （仅用于紧急情况） ~~(ci msg + " - IKNOWWHATIAMDOING")~~ | 快速/粗放模式（**不推荐**）： | | Allowed to merge | Allowed to push | | ------- | :---------------------- | :---------------------- | | master | Developers + Maintainers | Developers + Maintainers | | develop | Developers + Maintainers | Developers + Maintainers |

4. 定制分支

少量的可套用上面分支格式，发布分支需要强制检查，但 定制客户较多时建议 **Fork** 到新的仓库（代码可以 push/merge 会原仓库）

5. 历史分支

长期不用的分支建议打 **tag** 之后删除（从旧分支上创建名字符合规范的新分支，然后删除旧分支）

Code Review

在代码合并到 `master`、`release` 或 `beta` 分支时，需要通过 **GitLab** 的 **merge request** 流程进行 **code review**。

- 为了使分支的 graph 保持简洁，发起 merge request 之前最好执行 `git reset --soft` + `git commit` 或者 `git rebase --autosquash`，如果确实需要保留多条提交记录需要在 merge request 的备注里说明
- maintainer 在 merge 代码时，如果发现提交次数不是唯一，需要考虑勾选 `squash commits` 合并提交，并且无论如何都要勾选 `delete source branch` (必选)
- 小功能、bugfix 由 maintainer 独自或者叫相关 developer 一起 review，大块功能需要叫相关同学一起 reivew

> SourceTree 里 Remote 设置 Optional extended integration —— Host Type 选 `GitLab CE`、Host 填 `https://git.baijiashilian.com/`，然后在 push 过的分支上右键即可 Create Pull Request；

版本规范

版本管理规范

空白文档

启动 mysql, 并设置为开机启动

> Mysql

```
# 启动 mysql, 并设置为开机启动
brew services start mysql
# 关闭 mysql
brew services stop mysql
# 重启 mysql
brew services restart mysql
```

> Redis

```
# 启动 mysql, 并设置为开机启动
brew services start redis
# 关闭 mysql
brew services stop redis
# 重启 mysql
brew services restart redis
```

空白文档

IO流介绍

IO流介绍

> 输入流

- `InputStream`
  - `FIleInputStream`
  - `BufferedInputStream`
  - `ObjectInputStream`
- `Reader`
  - `FileReader`
  - `BufferedReader`
  - `InputStreamReader`

> 输出流

- `OutputStream`
  - `FileOutputStream`
  - `BufferedOutputStream`
  - `ObjectOutputStream`
- `Writer`
  - `FileWriter`
  - `BufferedWriter`
  - `OutputStreamWriter`

> 文件流：文件在程序中是以 `流` 的形式进行操作的
>
> `流：` 数据在数据源（文件）和程序（内存）之间经历的路径
>
> `输入流：` 数据从数据源（人间）到程序（内存）的路径
>
> `输出流：` 数据从程序（内存）到数据源（文件）的路径

常用文件操作

创建文件

```
new File(String pathname);
new File(File parent, String child);              #根据父目录文件 + 子路径构建
new File(String parent, String child);   #根据父目录 + 子路径构建
file.creatNewFile  #创建新文件
```

案例

```
private static final String path = "/Users/Jeffrey/Public/";
@Test
public void createdFileTypeTest1(){
  String fileName = "a.txt";

  try {
    File file = new File(path + fileName);
    boolean newFile = file.createNewFile();
    if (newFile){
      System.out.println("success");
    } else {
      System.out.println("error");
    }
  }catch (Exception e){
    System.out.println(e.getMessage());
    e.printStackTrace();
  }
}

@Test
public void createdFileTypeTest3(){
  String fileName = "a4.txt";

  File file = new File(new File(path),fileName);
  try {

    boolean newFile = file.createNewFile();
    if (newFile){
      System.out.println("success");
    } else {
      System.out.println("error");
    }
  }catch (Exception e){
    System.out.println(e.getMessage());
    e.printStackTrace();
  }

@Test
public void createdFileTypeTest2(){
  String fileName = "a1.txt";

  try {
    File file = new File(path,fileName);
    boolean newFile = file.createNewFile();
    if (newFile){
      System.out.println("success");
    } else {
      System.out.println("error");
    }
  }catch (Exception e){
    System.out.println(e.getMessage());
    e.printStackTrace();
  }
}
}
```

读取文件

```
gatName
getAbsolutePath  #绝对路径
getParent
length
exists
isFile
isDirectory
```

案例

```
private static final String path = "/Users/Jeffrey/Public/";
@Test
public void getInfo(){
  String fileName = path + "a.txt";
  File file = new File(fileName);
  System.out.println("文件名 : " + file.getName());
  System.out.println("文件绝对路径 : " + file.getAbsolutePath());
  System.out.println("文件上级目录 : " + file.getParent());
  System.out.println("文件大小(字节) : " + file.length());
  System.out.println("文件存不存在 : " + file.exists());
  System.out.println("文件是不是文件 : " + file.isFile());
  System.out.println("文件是不是文件夹 : " + file.isDirectory());
}
#=============打印==============
> 文件名 : a.txt
> 文件绝对路径 : /Users/Jeffrey/Public/a.txt
> 文件上级目录 : /Users/Jeffrey/Public
> 文件大小(字节) : 0
> 文件存不存在 : true
> 文件是不是文件 : true
> 文件是不是文件夹 : false
```

IO流原理及分类

流的分类

- 按操作数据单位不同分为： 字节流（8 bit）、字符流（按字符）
- 按数据流的流向不同分为：输入流、输出流
- 按流的角色的不同分为：字节流、处理流/包装流

| （抽象基类） | 字符流 | 字节流 | | :———-: | :———-: | :———-: | | 输入流 | Reader | InputStream | | 输出流 | Writer | OutputStream |

IO流常用的类

`InputStrem` 字节输入流

> `InputStream` 抽象类是所有类字节输入流的超类
>
> **`InputStream`** 常用的子类：
>
> - `FileInputStream` ： 文件输入流
> - `BufferedInputStream` ： 缓存字节输入流

- `ObjectInputStream` ：对象字节输入流

```
private static final String path = "/Users/Jeffrey/Public/a.txt";

/**
 * 这个方法效率较低
 * ->优化：  使用  read(byte[] b)
 */
@Test
public void readFile1(){
  int readChar;
  try (FileInputStream fileInputStream = new FileInputStream(path)){
    while ((readChar = fileInputStream.read()) != -1){
      System.out.print((char) readChar);
    }
  }catch (IOException e){
    e.printStackTrace();
  }
}

/**
 * 使用  read(byte[] b)  增强效率
 */
@Test
public void readFile2(){
  byte[] buf = new byte[8];
  int readLen;
  try (FileInputStream fileInputStream = new FileInputStream(path)){
    while ((readLen = fileInputStream.read(buf)) != -1){
      System.out.println(new String(buf,0,readLen));
    }
  }catch (IOException e){
    e.printStackTrace();
  }
}
```

`OutputStrem` 字节输入流

`OutputStream` 抽象类是所有类字节输出流的超类

`OutputStream` 常用的子类：

- `FileOutputStream` ：文件输出流
- `BufferedOutputStream` ：缓存字节输出流

java基础知识

java基础知识

StringBuilder 操作字符串

```
/*
  public StringBuilder append(任意类型): 添加数据，并返回对象本身
  public StringBuilder reberse() : 返回相反的字符序列
  public int length(): 返回长度（字符出现的个数）
  publis String toString(): 通过toString() 可以实现把StringBuilder转换为String
*/
StringBuilder res = new StringBuilder();
res.append("xx");//添加数据
res.reverse();//翻转字符串
res.length();//返回长度（字符出现的个数）
res.toString();//转换为String类型
```

常用**API**

```
Random  //随机数
Math    //数学运算
System  //System 类包含一些有用的类字段和方法。它不能被实例化。
Object  //所有类的父类
  toString
  equals  //也是使用的== 进行比较    比较的是内存地址   重写可以比较内容，自动生成
BigDecimal // 精确计算    使用字符形式
  //特殊：divide(另一个BigDecimal对象，精确几位，舍入模式)
```

自动装箱，自动拆箱

- 装箱：把基本数据类型转换为对应的包装类型
- 拆箱：把包装类型转换为基本数据类型

数组的高级操作

- 二分查找 （数组顺序按照大小顺序排列）每次去掉一半的查找范围

```
  int max = arr.length - 1;
          int min = 0;
          while (min <= max){
              int mid = (min + max) >> 1;
              if (arr[mid] > num){
                  max = mid -1;
              }else if (arr[mid] < num){
                  min = mid + 1;
              }else{
                  return mid;
              }
          }
          return -1;
```

- 冒泡排序

```
  for (int i = 0; i < arr.length - 1 ; i++) {
            for (int j = 0;j < arr.length - 1 - i;j++){
                if (arr[j] > arr[j + 1]){
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
        return arr;
```

- 快速排序

时间

```
Date()  //当前时间
Date(毫秒) //根据毫秒获取日期    long格式
SimpleDateFormate //格式化Date日期   与   解析日期格式的日期
```

异常

- 首先会看程序中是否有异常处理 代码，如果没有，交给本方法的调用者，也就是虚拟机，虚拟机针对于异常做了那些事情？
  - 将异常信息打印出来
  - 停止程序运行，哪里出现了异常，就在哪里停止。

异常处理

`throws`

- 编译时异常，因为在编译时就会进行检查，一定要在方法后面进行显示声明
- 运行时异常，因为是在运行时才会发生，可以不在方法名后面进行声明

集合 **`Collection`**

集合

- Collection 单列几个
  - List

```
List集合特点：
        - 有序：存储和取出的顺序一直
        - 有索引：可以通过索引取出元素
        - 可重复：存储的元素可以重复
List集合特有的方法：
        - add    在指定位置插入指定的元素
        - remove          移除指定索引处的元素
        - set    修改指定索引处的元素
        - get    获取指定索引的元素
```

  - ArrayList
  - LinkedList

    - Set 不可重复
    - HashSet
    - TreeSet
- Map 双列结合
    - HashMap
    - TreeMap

遍历集合：`Iterator`

- For 如果操作需要用到索引，使用for
- 迭代器 Iterator 如果需要删除元素，使用迭代器
- 增强for 如果仅仅只是遍历，使用增强for

File类 表示文件的位置，对文件以及文件夹进行操作

```
File path = new File("C:\\test\\a.txt");
boolean res = path.createNewFile();   //只能创建文件
//如果文件存在，不创建，返回false
//如果文件不存在，创建成功后，返回true
```

IO 对硬盘中的文件进行读写

Maven学习

- 安装自行看文档

命令

```
mvn compile //编译
mvn clean   //删除编译
mvn test    //做测试使用的
mvn package //打包
mvn install //安装到本地仓库
```

UTF-8问题

- 解决 `Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform depend`

```
<project.build.sourceEncoding>
    UTF-8
</project.build.sourceEncoding>
```

`Assert` 弃用

- `import junit.framework.Assert;` 改为 `import org.junit.Assert;`

POMxml配置项先关信息

- xml头信息

- project 根元素
- modelVersion 对象模型版本号
- groupId 项目
- artifactId 项目名称
- version 项目版本
- packaging 项目最终是什么 jar包 就写jar
- dependencies 项目运行依赖的资源

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3
  <!--指定pom的模型版本-->
  <modelVersion>4.0.0</modelVersion>
  <!--打包方式，web为war包，java程序为jar包-->
  <packaging>war</packaging>

  <name>web01</name><!--可以删掉-->
<!--  组织id-->
  <groupId>cn.zhangfayuan</groupId>
<!--  项目id-->
  <artifactId>web01</artifactId>
<!--  版本号   release （完成版）   snapshot（开发板）-->
  <version>1.0-SNAPSHOT</version>
<!--设置当前工程的所有依赖-->
  <dependencies>
<!--    具体的依赖-->
<!--    <dependency>-->
<!--      <groupId></groupId>-->
<!--      <artifactId></artifactId>-->
<!--    </dependency>-->
  </dependencies>

  <!--构建插件-->
  <build>
    <plugins>
      <plugin>
          <groupId>org.apache.tomcat.maven</groupId>
          <artifactId>tomcat7-maven-plugin</artifactId>
          <version>2.1</version>
        <configuration>
          <port>8088</port>
          <path>/</path>
        </configuration>
      </plugin>
    </plugins>
  </build>

</project>
```

依赖管理

```
<dependencies>
```

```
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.12</version>
  </dependency>
</dependencies>
```

依赖传递

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.12</version>
  </dependency>
  <!-- 依赖传递，同样使用依赖管理的格式，在project02中的依赖，当前引入的程序中也可以进
  <dependency>
    <groupId>cn.zhangfayuan</groupId>
    <artifactId>project02</artifactId>
    <version>1.0-SNAPSHOT</version>
  </dependency>
</dependencies>
```

可选依赖（不透明，隐藏依赖）

```
 <!-- 依赖传递，同样使用依赖管理的格式，在project02中的依赖，当前引入的程序中也可以进行
  <dependency>
    <groupId>logkit</groupId>
    <artifactId>logkit</artifactId>
    <version>1.0.1</version>
    <optional>true</optional><!-- 可选依赖 -->
</dependency>
```

排除依赖

```
 <dependency>
   <groupId>cn.zhangfayuan</groupId>
   <artifactId>project02</artifactId>
   <version>1.0-SNAPSHOT</version>
   <exclusions>
     <exclusion><!-- 排除依赖 -->
       <groupId>logkit</groupId>
       <artifactId>logkit</artifactId>
     </exclusion>
   </exclusions>
```

```
</dependency>
```

依赖的范围

简介

配置

`scope`

依赖传递的范围

Maven项目构建生命周期

- Maven对项目构建的生命周期划分为：
  - clean：清理工作
  - pre-clean：执行一些需要在clean之前完成的工作
  - Clean：移除所有上一次构建生成的文件
  - post-clean：执行一些需要在clean之后立刻完成的工作
  - default：核心工作，例如：编译、测试、打包、部署等
  - site：产生报告、发布站点等

IOC 入门案例

步骤

- 加载spring
- 创建资源
- 配置资源
- 使用资源

Ioc配置

bean

MyBatis学习

设置

1、pom文件中引入相关坐标

或者是以内包 `libs` 目录下，并加入包

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.25</version>
</dependency>

<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>3.5.7</version>
</dependency>
```

2、新增配置文件

`UsersMapper.xml` [src目录下 或 resources资源目录下]

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="UsersMapper">
    <select id="getAll" resultType="users">  <!-- 使用了类别名，全名 cn.pithy.
        SELECT * FROM users
    </select>

    <select id="findById" resultType="cn.pithy.bean.Users" parameterType="j
        SELECT * from users where id = #{id}
    </select>

    <insert id="insert" parameterType="cn.pithy.bean.Users" useGeneratedKey
        INSERT into users (username, password) values (#{username},#{passwo
    </insert>

    <update id="update" parameterType="cn.pithy.bean.Users">
        UPDATE users set username = #{username},password=#{password} where
    </update>

    <delete id="delete" parameterType="java.lang.Integer">
        DELETE from users where id = #{id}
    </delete>
</mapper>
```

3、新增核心配置文件

`MybatisConfig.xml` [src目录下 或 resources资源目录下]

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>
  <!-- 类名起别名 -->
   <typeAliases>
        <typeAlias type="cn.pithy.bean.Users" alias="users" />
    </typeAliases>
```

```
    <environments default="mysql">
        <environment id="mysql">
            <transactionManager type="jdbc"/>
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql://127.0.0.1:8889/spr
                <property name="username" value="root"/>
                <property name="password" value="root"/>
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <mapper resource="UsersMapper.xml" />
    </mappers>
</configuration>
```

4、编写对象类

5、编写程序

```
public class UsersDao {

    public List<Users> getAll() throws IOException {
        //加载核心配置文件
        InputStream resourceAsStream = Resources.getResourceAsStream("MyBat
        //实例化Sql session工厂对象
        SqlSessionFactory build = new SqlSessionFactoryBuilder().build(reso
        //通过工厂对象获取sql session 对象
        //        SqlSession sqlSession = build.openSession(true); 自动提交事
        SqlSession sqlSession = build.openSession();
        //执行xml中的sql语句
        List<Users> users = sqlSession.selectList("UsersMapper.getAll");
        //释放资源
        sqlSession.close();
        resourceAsStream.close();
        return users;
    }

    public Users getById(int id) throws IOException{
        InputStream resourceAsStream = Resources.getResourceAsStream("MyBat
        SqlSessionFactory build = new SqlSessionFactoryBuilder().build(reso
        //        SqlSession sqlSession = build.openSession(true); 自动提交事
        SqlSession sqlSession = build.openSession();
        Users user = sqlSession.selectOne("UsersMapper.findById", id);
        sqlSession.close();
        resourceAsStream.close();
        return user;
    }

    public void insert() throws IOException{
        InputStream resourceAsStream = Resources.getResourceAsStream("Mybat
        SqlSessionFactory build = new SqlSessionFactoryBuilder().build(reso
//        SqlSession sqlSession = build.openSession(true); 自动提交事务
        SqlSession sqlSession = build.openSession();
        Users user = new Users();
        user.setUsername("woCao");
        user.setPassword("99999999");

        int insert = sqlSession.insert("UsersMapper.insert", user);
```

```
        if (insert > 0){
            sqlSession.commit();
            System.out.println("Success" + insert + user.getId());
        }else {
            System.out.println("Error" + insert);
        }
        sqlSession.close();
        resourceAsStream.close();
    }

    public void update() throws IOException{
        InputStream resourceAsStream = Resources.getResourceAsStream("MyBat
        SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder(
        SqlSession sqlSession = sqlSessionFactory.openSession(true);
        Users user = sqlSession.selectOne("UsersMapper.findById", 6);
        user.setUsername("woCaoLei哈哈哈");
        int update = sqlSession.update("UsersMapper.update",user);
        if (update > 0){
            System.out.println("Success" + update + user.getId());
        }else {
            System.out.println("Error" + update);
        }
        sqlSession.close();
        resourceAsStream.close();
    }

    public void delete(int id) throws IOException{
        InputStream resourceAsStream = Resources.getResourceAsStream("MyBat
        SqlSessionFactory build = new SqlSessionFactoryBuilder().build(reso
        SqlSession sqlSession = build.openSession(true);
        int delete = sqlSession.delete("UsersMapper.delete", id);
        if (delete > 0){
            System.out.println("Success" + delete );
        }else {
            System.out.println("Error" + delete);
        }
        sqlSession.close();
        resourceAsStream.close();
    }
}
```

相关配置项

传统方式实现**DAO**层

控制层=》业务层=》接口层

package

—-**dao**

—-**controller** 控制层

|—-UsersController.java

—-**service** 业务接口层

|—-**impl**

|—-UsersServiceImp.java 业务接口实现

|—-UsersService.java 业务接口类

—- **mapper** 持久层接口、

|—-**impl** *** 接口代理方式实现的话 这个目录不要***

|—- UsersMapperImpl.java 接口的实现类

```
resourceAsStream = Resources.getResourceAsStream("MyBatisConfig.xml");
SqlSessionFactory build = new SqlSessionFactoryBuilder().build(resourceAsSt
sqlSession = build.openSession(true);
UsersMapper mapper = sqlSession.getMapper(UsersMapper.class); //注意这个部分
list = mapper.getAll();
```

|—-UsersMapper.java 接口类 *** 定义接口类以及实现方法 ***

LOG4J 打印**sql**信息

设置步骤

1. 导入jar包【pom文件中加入LOG4J坐标】
2. 配置核心配置文件

```
  <!--  集成LOG4J日志  -->
  <settings>
    <setting name="logImpl" value="log4j"/>
  </settings>
```

1. src目录下配LOG4J配置文件【maven项目配置在resources资源目录下】

MyBatis 进阶

接口代理的方式实现**DAO**持久层

分页（使用查看文档）

一对一查询

封装映射对象关系

一对多查询

多对多查询

MyBatis高级操作

MyBatis注解开发

常用单标操作注解

- @Select(”sql语句“)
  - bean下创建类文件

```
package cn.pithy.bean;


public class Users {
    private int id;


    private String username;


    private String password;


    ....
}
```

  - mapper下创建接口文件

```
package cn.pithy.mapper;


import cn.pithy.bean.Users;
import org.apache.ibatis.annotations.Select;


import java.util.List;


/**
 * @author zhangfayuan
 */
public interface UsersMapper {
    /**
```

```
     * 查询所有
     * @return List
     */
    @Select("select * from users")
    public abstract List<Users> selectAll();
}
```

- 核心配置文件中配置映射关系

```
<!--  配置映射关系  -->
<mappers>
<package name="cn.pithy.mapper"/><!-- mapper所在的包地址 -->
</mappers>
```

- 然后直接使用

```
@Test
public void selectAll() throws IOException {
  InputStream resourceAsStream = Resources.getResourceAsStream("MyBatis
  SqlSessionFactory build = new SqlSessionFactoryBuilder().build(resour
  SqlSession sqlSession = build.openSession(true);
  UsersMapper mapper = sqlSession.getMapper(UsersMapper.class);
  List<Users> users = mapper.selectAll();
  for (Users user : users) {
    System.out.println(user);
  }
  sqlSession.close();
  resourceAsStream.close();


}
```

- @Insert(”sql语句“)
- @Update(”sql语句“)
- Delete(”sql语句“)

注解实现多表操作

一对一操作

MyBatis-plus+SpringBoot

1、安装SpringBoot

2、安装依赖

- 安装 `mybatis-plus-boot-starter`

```
<!--        引入mybatis依赖-->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
```

```
            <version>3.4.3.4</version>
        </dependency>
```

- 安装 `druid`

```
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.2.8</version>
        </dependency>
```

3、入口文件扫描Dao

```
@SpringBootApplication
@MapperScan("cn.pithy.dao")
public class SpringTest1Application {

    public static void main(String[] args) {
        SpringApplication.run(SpringTest1Application.class, args);
    }

}
```

4、编写配置文件

```
spring.datasource.type=com.alibaba.druid.pool.DruidDataSource
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/springboot_data?character
spring.datasource.username=root
spring.datasource.password=zha
```

5、编写类

```
package cn.pithy.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;
import lombok.experimental.Accessors;

@Data
@AllArgsConstructor  //有参构造
@NoArgsConstructor      //无参构造
@ToString //tostring
@Accessors(chain = true)  //开启链式操作
public class Users {
    private int id;
    private String name;
    private int age;
    private String address;
    private String phone;
}
```

6、创建dao接口，继承BaseMapper

7、可以使用了

SpringBoot配置

读取配置内容

@Value

```
@Value("${message1}")
private String message1;
@RequestMapping("/getValue")
public String getValue()
{
  String envGetName =  env.getProperty("name");
  return envGetName;
}
```

Environment

```
@Autowired
private Environment env;

@RequestMapping("/getValue")
public String getValue()
{
  String envGetName =  env.getProperty("name");
  return envGetName;
}
```

@ConfigurationProperties

- 先定义一个类

```
  package cn.pithy.springbootinit;

  import org.springframework.boot.context.properties.ConfigurationPropertie
  import org.springframework.stereotype.Component;

  /**
   * @author zhangfayuan
   */
  @Component
  @ConfigurationProperties(prefix = "persion")
  public class Persion {

      private String name;
      private int age;

      public String getName() {
          return name;
      }

      public void setName(String name) {
          this.name = name;
```

```
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Persion{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

- 注入

```
@Autowired
private Persion persion;
@RequestMapping("/getValue")
public String getValue()
{
  return persion.toString();
}
```

profile不同环境切换

1、配置方式

多profile配置方式

```
## -- application.properties
spring.profiles.active=pro

## -- application-dev.properties
server.port=8081

## -- application-test.properties
server.port=8082

## -- application-pro.properties
server.port=8083
```

yml多文档方式

```
---
server:
  port: 8081
spring:
  profiles: dev
---
```

```
server:
  port: 8082
spring:
  profiles: test
---
server:
  port: 8083

spring:
  profiles: pro
---
spring:
  profiles:
    active: pro
```

2、profile激活方式

配置文件

上面的方式

虚拟机参数

`-Dspring.profiles.active=dev`

命令行参数

`--spring.profiles.active=dev`

注解大全

```
url: jdbc:mysql://47.92.25.174:3336/ai_algo_aat?useUnicode=true&characterEr
username: algo
password: algo123QAZ
```

目录结构

- 数据实体类 `domain` ：com.bajins.entity
- 数据接口访问层 `Dao` ： com.bajins.mapper
- 数据服务接口层 `Service` ：com.bajins.service
- 数据服务接口实现层: com.bajins.service.impl
- 前端控制器层：com.bajins.controller
- 工具类库：com.bajins.utils
- 配置类：com.bajins.config
- 数据传输对象：com.bajins.dto

JAVA常用函数

JAVA常用函数

字符串转列表

```
String ids = "1,2,3,4,5";
List<Integer> idsList = Arrays.stream(ids.split(",")).map(s -> Integer.pars
```

时间戳转日期格式

```
Long createdEnd = createTime;/毫秒时间戳
SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:
System.out.println(simpleDateFormat.format(createTime));
```

递归删除文件夹

```
public static void main(String[] args){
  File path = new File("/Users/Jeffrey/Desktop/willdelete");
  /递归删除
  boolean res = deletePath(path);
  if(res) System.out.println("success");
}

 private static boolean deletePath(File path) {
        if (path.exists()){
            File[] files = path.listFiles();
            assert files != null;
            for (File file : files) {
                if (file.isFile()){
                    /直接删除
                    file.delete();
                }else{
                    /如果是文件夹  继续打开文件夹，判断有没有文件
                    deletePath(file);
                }
            }
            return path.delete();
        }
        return true;
    }
```

文件写入

```
fileOutputStream(byte,boolean append)
fileOutputStream(byre[],boolean append)
/换行  "\r\n".getBytes()
FileOutputStream fileOutputStream = null;
try {
  fileOutputStream = new FileOutputStream("/Users/Jeffrey/Desktop/TTT.txt")
  fileOutputStream.write(99);
```

```
  fileOutputStream.close();
}catch (Exception e){
  e.printStackTrace();
}finally {
  if (fileOutputStream != null){
    try {
      fileOutputStream.close();
    }catch (Exception e){
      e.printStackTrace();
    }
  }
}
```

并行和并发、进程和线程

第一种开启线程

```
public class MyThread extends Thread{
    private String title;
    public MyThread(String title) {
        this.title = title;
    }

    @Override
    public void run(){
        for (int i = 0; i < 100; i++) {
            System.out.println(this.title + "开启了线程" + i);
        }
    }
}
/开启线程
MyThread myThread = new MyThread("Test1");
MyThread myThread1 = new MyThread("Test2");
myThread.start();
myThread1.start();
```

第二种开启线程【**Runnable**】

```
public class MyRunnable implements Runnable{
    @Override
    public void run() {
        for (int i = 0; i < 100; i++) {
            System.out.println("第二种方式开启线程 " + i);
        }
    }
}
/开启线程
MyRunnable myRunnable = new MyRunnable();
Thread thread = new Thread(myRunnable);
thread.start();
```

第三种看起线程【**Callable**和**Future**】

- 优点

    - 扩展性强，实现该接口的同时可以继承其他的类
- 缺点
    - 编程相对复杂，不能直接使用Thread类

```
public class MyCallable implements Callable<String> {
    @Override
    public String call() throws Exception {
        //返回值表示线程执行结束之后的返回结果
        for (int i = 0; i < 100; i++) {
            System.out.println("第三种开启下线程 " + (i + 1));
        }
        return "执行结束";
    }
}
//开启线程
MyCallable myCallable = new MyCallable();
FutureTask<String> stringFutureTask = new FutureTask<>(myCallable);
Thread thread1 = new Thread(stringFutureTask);
thread1.start();
String s = stringFutureTask.get();
System.out.println(s);
```

生产者、消费者

消费者

```
package cn.zhangfayuan.collection;

public class Eater extends Thread{
    @Override
    public void run() {
        while (true){
            synchronized (TableDesk.lock){
                if (TableDesk.count == 0){
                    break;
                }else{
                    if (TableDesk.flag){
                        System.out.println("消费者消费");
                        TableDesk.flag = false;
                        TableDesk.lock.notifyAll();
                        TableDesk.count--;
                    }else{
                        try {
                            TableDesk.lock.wait();
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                }
            }
        }
    }
}
```

生产者

```
package cn.zhangfayuan.collection;
```

```
public class Cooker extends Thread{
    @Override
    public void run() {
        while (true){
            synchronized (TableDesk.lock){
                if (TableDesk.count == 0){
                    break;
                } else {
                   if (TableDesk.flag){
                       //有数据 就等待
                       try {
                           TableDesk.lock.wait();
                       } catch (InterruptedException e) {
                           e.printStackTrace();
                       }
                   }else{
                       //生产
                       System.out.println("生产者生产");
                       TableDesk.flag = true;
                       TableDesk.lock.notifyAll();
                   }
                }
            }
        }
    }
}
```

中间作用类

```
package cn.zhangfayuan.collection;

public class TableDesk {

    public static int count = 10;
    public static boolean flag = false;
    public static final Object lock  = new Object();


}
```

调用

```
Eater eater = new Eater();
Cooker cooker = new Cooker();
eater.start();
cooker.start();

//======================  处理结果   ==================
//生产者生产
//消费者消费
//生产者生产
//消费者消费
//生产者生产
//消费者消费
//生产者生产
//消费者消费
//生产者生产
//消费者消费
//生产者生产
```

```
//消费者消费
//生产者生产
//消费者消费
//生产者生产
//消费者消费
//生产者生产
//消费者消费
//生产者生产
//消费者消费
```

redis-cli -h redis-k268b7ibip3s-proxy-nlb.jvessel-open-hb.jdcloud.com -p 6379 -a 4jq1QdugammK

zip

持枪

1.png

2.png

3.png

持棍

1.png

4.png

5.png

持刀

1.png

6.png

7.png

第一步： /pc/project/task/upload/103 参数 【 id 文件】

第二步 ：/pc/project/task/getProjectTaskAllot 参数 【{“projectId”:103,“taskIds”:[270]}】

/pc/project/task/page

不做分类 其他 其他-背包 其他-行为 持刀 持改锥 持枪 持棍 未背包 空手 背包

Sharding-jdbc分片策略

策略：

> 第一种：none

对应NoneSHardingStragey,不分片策略，SQL会被发送给所有节点进行还行，这个规则没有项目可以配置

> 第二种：inline行表达时分片策略（核心）

对应InlineShardingStragey，使用Groovy的表达，提供对sql

语句种的=和in的分片操作支持，只支持单片键。对于简单的分片算法，可以通过简单的配置使用，从而避免复杂的java代码开发

> 第三种：

验证用户名与MD5 (+ salt)密码

验证用户名与**MD5 (+ salt)**密码

> 模拟数据： md5加密：e10adc3949ba59abbe56e057f20f883e MD5 + salt：b42619cf9e4a1f5dad2aaa8a7a70b31e salt="!@#$" MD5 + salt + hash：f57cf2d66c506567c49ad04bb58fa566 salt="!@#$" hasd散列轮数=1024

realm设置

```
package com.catchadmin.shiro;

import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.authc.AuthenticationInfo;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authc.SimpleAuthenticationInfo;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;
import org.apache.shiro.util.ByteSource;

public class Md5AuthRealm extends AuthorizingRealm {
    /**
     * 授权
     * @param principalCollection
     * @return
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection
        return null;
    }

    /**
     * 认证
     * @param authenticationToken
     * @retur
     * @throws AuthenticationException
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToke
        String username = authenticationToken.getPrincipal().toString();
        System.out.println("authenticationToken:" + username);
        if ("zhangfayuan".equals(username)){
            return new SimpleAuthenticationInfo(username,"e10adc3949ba59abb
        }
        //加盐验证
       if ("zhangfayuan".equals(username)){
            return new SimpleAuthenticationInfo(username,"b42619cf9e4a1f5da
        }
        return null;
    }
}
```

请求验证设置

```
public String md5Checking() {
    //创建安全管理器
    DefaultSecurityManager defaultSecurityManager = new DefaultSecurityMana
    //注入realm
    Md5AuthRealm md5AuthRealm = new Md5AuthRealm();
    //设置加密方式
    HashedCredentialsMatcher hashedCredentialsMatcher = new HashedCredentia

    hashedCredentialsMatcher.setHashAlgorithmName("md5");//设置加密方式
    hashedCredentialsMatcher.setHashIterations(1024); //设置hash散列轮数
    md5AuthRealm.setCredentialsMatcher(hashedCredentialsMatcher);
    defaultSecurityManager.setRealm(md5AuthRealm);
    // 将安全管理器注入安全工具
    SecurityUtils.setSecurityManager(defaultSecurityManager);
    //获取 subject 主体
    Subject subject = SecurityUtils.getSubject();
    UsernamePasswordToken token = new UsernamePasswordToken("zhangfayuan",
    try {
        subject.login(token);
        System.out.println("登录成功");
    }catch (UnknownAccountException e){
        System.out.println("用户名不存在");
    }catch (IncorrectCredentialsException e){
        System.out.println("密码错误");
    }
    return "Login Success!";
}
```

验证用户名与**MD5 + salt**密码

```
//设置加密方式
HashedCredentialsMatcher hashedCredentialsMatcher = new HashedCredentialsMa
hashedCredentialsMatcher.setHashAlgorithmName("md5");//设置加密方式
hashedCredentialsMatcher.setHashIterations(1024); //设置hash散列轮数
md5AuthRealm.setCredentialsMatcher(hashedCredentialsMatcher);

//Md5 + salt + hash
if ("zhangfayuan".equals(username)){
    return new SimpleAuthenticationInfo(username,"b42619cf9e4a1f5dad2aaa8a7
}
```

springboot具体的工作流程

- 创建DO文件
- 数据库操作
  - infrastructure下dao文件夹下创建对用的DO文件
  - Mapper创建对应的自定义sql操作
  - client=>dto目录下创建CMD文件【验证规则】
  - client=>api目录下创建service接口类
  - 自定义sql在in
  - app=>executor目录下创建对应的xxxCmdExe(增、删、改)/xxxQryExe(查)实现操作【增删改查】
  - app=>service下创建xxxServiceLmlp 继承client=>api 的service接口类，并实现方法，调用xxxExe的具体实现方法
  - controller调用xxService接口中的方法

```
@GetMapping("/video/setting/get")
@ApiOperation(value = "获已开启的算法与版本", httpMethod = "GET")
```

生活篇

生活篇

\1. 逛一次宜家

\2. 喂一次鸽子

\3. 做一次大扫除

\4. 做一桌子菜

\5. 吃一次泡面

\6. 养一只动物

\7. 做一个蛋糕

\8. 手洗一桶衣服

\9. 做一次全身检查

\10. 给对方掏一次耳朵

\11. 给对方剪一次指甲

\12. 给对方生日来一个惊喜

\13. 一起理头发

\14. 街头被画一次画像

时光篇

\15. 看一次日落

\16. 看一次日出

\17. 一起晨跑

\18. 晒一下午太阳

\19. 半夜开车兜一次风

\20. 春天去一次野餐

\21. 夏天抓一次萤火虫

\22. 秋天扫一次落叶

\23. 冬天堆一个大雪人

\24. 为人生祈祷一分钟

\25. 接吻一小时

\26. 赖24小时的床

兴趣篇

\27. 游一次泳

\28. 打一次羽毛球

\29. 钓一次鱼

\30. 画一幅油画

\31. 学一首歌并合唱

\32. 玩一次高尔夫球

\33. 玩一款桌游

\34. 玩一次蹦极

\35. 玩一次潜水

\36. 玩一次漂流

\37. 滑一次雪

\38. 一起办健身卡

欣赏篇

\39. 看一次流星雨

\40. 去一次自然博物馆

\41. 逛一次美术展或设计展

\42. 逛一次植物园

\43. 看一场露天电影

\44. 看一场话剧

\45. 看一场演唱会

\46. 看一场音乐会

\47. 泡一天图书馆

\48. 玩一次迪士尼乐园

\49. 看一场球赛

\50. 去天文台观一次天象

享受篇

\51. 住一次精品酒店

\52. 看同一部剧

\53. 看同一部动漫

\54. 玩同一款游戏

\55. 做一次SPA

\56. 泡一次温泉

\57. 买一箱零食

\58. 吃一顿米其林三星

\59. 看一场party

\60. 买一件奢饰品

\61. 坐一次摩天轮

旅行篇

\62. 去一次海边

\63. 去一次草原

\64. 去一次沙漠

\65. 去一次热带雨林

\66. 去一次高原

\67. 骑一次双人自行车

\68. 骑一次马

\69. 坐一趟10小时以上的火车

\70. 在机场过夜

\71. 去一次大洋洲的一个国家

\72. 去一次欧洲的一个国家

\73. 去一次东南亚的一个国家

\74. 去一次非洲的一个国家

\75. 去一次美洲的一个国家

挑战篇

\76. 去街头卖一次东西

\77. 疯狂呐喊一次

\78. 酒吧跳一次舞

\79. 吃一次虫子做的东西

\80. 穿一次外族民族服饰

\81. cos一个角色

\82. 和一个名人合影

\83. 玩一次vr游戏

\84. 通宵完成一件事

\85. 完成一幅大拼图

\86. 收集一样东西

\87. 实践自己一个创意

\88. 玩一次密室逃脱

\89. 喝醉一次

分享篇

\90. 写一篇书评或影评

\91. 献一份爱心给灾区

\92. 给对方写一封情书

\93. 回一次母校

\94. 穿情侣装

95.翻译一篇外文

\96. 写一篇旅行游记

\97. 种一棵树

\98. 录一段教学视频

\99. 给共同好友买一件礼物

SOP充装作业

```
pcPointStep.setCreateTime(DateUtils.getNowDate());
pcPointStep.setUpdateTime(DateUtils.getNowDate());
pcPointStepMapper.insertPcPointStep(pcPointStep);

List<Long> deviceIds = pcPointStep.getDeviceIds();
List<Long> deviceSetting = pcPointStep.getDeviceSetting();
Long pcProcessDetailId = pcPointStep.getId();

for (int i = 0; i < deviceIds.size(); i++) {
    PcPointStepDevice pcPointStepDevice = new PcPointStepDevice();
    pcPointStepDevice.setStepId(pcProcessDetailId);
    pcPointStepDevice.setGroupId(pcPointStep.getGroupId());
    pcPointStepDevice.setDeviceId(deviceIds.get(i));
    Long coordinateId = deviceSetting.get(i);
    String coordinate = "[]";
    if (coordinateId > 0) {
        coordinate = pcCoordinateStagingMapper.selectPcCoordinateStagingByI
    }
    pcPointStepDevice.setCoordinate(coordinate);
    pcPointStepDeviceMapper.insertPcPointStepDevice(pcPointStepDevice);
}
```

```
pcPointStep.setUpdateTime(DateUtils.getNowDate());
pcPointStepMapper.updatePcPointStep(pcPointStep);
//先查询出所有的记录步骤关联通道信息
PcPointStepDevice pcPointStepDevice1 = new PcPointStepDevice();
pcPointStepDevice1.setGroupId(pcPointStep.getGroupId());
List<PcPointStepDevice> pcPointStepDevices = pcPointStepDeviceMapper.select
//删除之前关联的通道信息
pcPointStepDeviceMapper.deletePcPointStepDeviceByPcPointStepId(pcPointStep.

List<Long> deviceIds = pcPointStep.getDeviceIds();
List<Long> deviceSetting = pcPointStep.getDeviceSetting();
List<Long> pcPointStepDeviceIds = pcPointStep.getPcPointStepDeviceIds();
Long pcProcessDetailId = pcPointStep.getId();

for (int i = 0; i < deviceIds.size(); i++) {
    PcPointStepDevice pcPointStepDevice = new PcPointStepDevice();
    if (deviceSetting.get(i) == -1) {
        Long deviceId = deviceIds.get(i);
        Long stepDeviceId = pcPointStepDeviceIds.get(i);
        PcPointStepDevice pcPointStepDevice2;
        if (stepDeviceId != 0) {
            pcPointStepDevice2 = pcPointStepDevices.stream().filter(o -> o.
            pcPointStepDevice2.setDeviceId(deviceId);
        } else {
            pcPointStepDevice2 = pcPointStepDevices.stream().filter(o -> Ob
        }
        pcPointStepDeviceMapper.insertPcPointStepDevice(pcPointStepDevice2)
        continue;
    }

    pcPointStepDevice.setStepId(pcProcessDetailId);
    pcPointStepDevice.setGroupId(pcPointStep.getGroupId());
    pcPointStepDevice.setDeviceId(deviceIds.get(i));
```

```
    Long coordinateId = deviceSetting.get(i);
    String coordinate = "[]";
    if (coordinateId > 0) {
        coordinate = pcCoordinateStagingMapper.selectPcCoordinateStagingBy1
    }
    pcPointStepDevice.setCoordinate(coordinate);
    pcPointStepDeviceMapper.insertPcPointStepDevice(pcPointStepDevice);
}
```

```
int seconds = 0;
PcMonitoringTask pcMonitoringTask = pcMonitoringTaskService.selectPcMonitor
while (redisUtil.hHasKey(PC_POINT_STEP_HANDLE_KEY + "_mark" ,callbackResult
    log.info("=++++++++=======++++++++=======++++++++=======++++++++=======
    seconds++;
    if (seconds > pcMonitoringTask.getTimeInterval()){
        log.info("=++++++++=======++++++++=======++++++++=======++++++++===
        return;
    }
}

redisUtil.hset(PC_POINT_STEP_HANDLE_KEY + "_mark",callbackResult.getPointId
List<PcPointStep> pcPointSteps = pcPointStepMapper.selectPcPointStepByGroup
System.out.println(JSONObject.toJSONString(pcPointSteps));
int baffleNum = 0;
PcPoint pcPoint = pcPointMapper.selectPcPointById(callbackResult.getPointId

  //************************
  //        步骤处理
  //************************
```

```
public void truckHeadResultHandle(PcPointRecord pcPointRecord, PcPointStep
  List<ModelResult> result = callbackResult.getResult();
  List<ModelResult> collect = result.stream().filter(o -> type.equals(o.get
  PcPointStepDevice pcPointStepDevice = new PcPointStepDevice();
  pcPointStepDevice.setDeviceId(callbackResult.getVideoId());
  pcPointStepDevice.setGroupId(callbackResult.getPointId());
  pcPointStepDevice.setStepId(pcPointStep.getId());
  List<PcPointStepDevice> pcPointStepDevices = pcPointStepDeviceMapper.sele
  if (pcPointStepDevices.isEmpty()) {
      return;
  }
  PcPointStepDevice pcPointStepDeviceInfo = pcPointStepDevices.get(0);

  Map<String, List<ModelResult>> coincideModelResultsType = getCoincideMode
  if (!coincideModelResultsType.isEmpty()) {
      String imagePath = callbackResult.getImagePath();
      String s = scpDownFile(sftpConfig, imagePath);
      String frameImgPath = imageResultHandle(coincideModelResultsType, s,
      boolean isDone = checkStepIsDone(pcPointRecord, pcPointStep, callback
      if (pcPointStep.getStartFlag() == 1 && !isDone) {
          Date date = DateUtils.parseDate(callbackResult.getTime());
          PcPointRecord pcPointRecordInsertParams = PcPointRecord.builder()
          String processIsRunningKey =  callbackResult.getTaskId() + "_" +
          boolean addProcessIsRunning = redisUtil.hsetnx(PC_POINT_STEP_HAND
          if (addProcessIsRunning){
              pcPointRecordMapper.insertPcPointRecord(pcPointRecordInsertPa
```

```
                pcPointRecord = pcPointRecordInsertParams;
                redisUtil.hset(PC_POINT_STEP_HANDLE_KEY + URLConstant.PROCESS
                PcPointRecordDetail pcPointRecordDetail = PcPointRecordDetail
                log.info(">>>>>>>>>>>>>pcPointRecord: {}",JSONObject.toJSONSt
                log.info("》》》》》》》》》》》》》》》》》》》》》》》》》:{}",PC_POI
                boolean hSetNx = redisUtil.hsetnx(PC_POINT_STEP_HANDLE_KEY +
                if (hSetNx) {
                    pcPointRecordDetailMapper.insertPcPointRecordDetail(pcPoi
                }
            }
        } else if (pcPointStep.getEndFlag() == 1) {
            redisUtil.hset(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.get
        } else {
            if (!checkStepIsDone(pcPointRecord, pcPointStep, callbackResult.g
                PcPointRecordDetail pcPointRecordDetail = PcPointRecordDetail
                log.info(">>>>>>>>>>>>>pcPointRecord: {}",JSONObject.toJSONSt
                log.info("》》》》》》》》》》》》》》》》》》》》》》》》》:{}",PC_POI
                boolean hSetNx = redisUtil.hsetnx(PC_POINT_STEP_HANDLE_KEY +
                if (hSetNx){
                    pcPointRecordDetailMapper.insertPcPointRecordDetail(pcPoi
                }
            }
        }
        if (pcPointStep.getStartFlag() == 1) {
            redisUtil.hset(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.get
            redisUtil.hset(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.get
        }
        if (pcPointStep.getEndFlag() != 1) {
            redisUtil.hsetnx(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.g
        }

        File file = new File(s);
        if (file.exists()) {
            file.delete();
        }
    } else {
        if (pcPointStep.getEndFlag() == 1) {
            if (redisUtil.hHasKey(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRec
                Object hget = redisUtil.hget(PC_POINT_STEP_HANDLE_KEY + "_" +
                long i = DateUtils.getDifferenceSeconds(DateUtils.parseDate(c
                if (i > pcPointStep.getMonitorPrefixSecond()) {
                    if (!redisUtil.hHasKey(PC_POINT_STEP_HANDLE_KEY + "_" + p
                        return;
                    }
                    String imagePath = callbackResult.getImagePath();
                    String s = scpDownFile(sftpConfig, imagePath);
                    String frameImgPath = imageResultHandle(coincideModelResu
                    for (PcPointStep pointStep : pcPointSteps) {
                        if (!redisUtil.hHasKey(PC_POINT_STEP_HANDLE_KEY + "_'
                            Long alarmState = URLConstant.FALSE_STATUS;
                            Long dealState = URLConstant.DEAL_STATE_NOT_DEAL;
                            if (URLConstant.TEUCK_HEAD.equals(pointStep.getLa
                                alarmState = URLConstant.TRUE_STATUS;
                                dealState = URLConstant.DEAL_STATE_NOT_NEED_D
                            }
                            if (URLConstant.GUARDIAN_PERSION_DUTY.equals(poir
                                if (redisUtil.hHasKey(PC_POINT_STEP_HANDLE_KE
                                    alarmState = URLConstant.TRUE_STATUS;
                                    dealState = URLConstant.DEAL_STATE_NOT_NE
                                }
```

```
                            }
                            if (URLConstant.REAR_CAR_PATROL_PERSION.equals(pc
                                if (redisUtil.hHasKey(PC_POINT_STEP_HANDLE_KE
                                    alarmState = URLConstant.TRUE_STATUS;
                                    dealState = URLConstant.DEAL_STATE_NOT_NE
                                }
                            }
                            if (URLConstant.SAFETY_SLING.equals(pointStep.get
                                alarmState = URLConstant.TRUE_STATUS;
                                dealState = URLConstant.DEAL_STATE_NOT_NEED_D
                                if (redisUtil.hHasKey(PC_POINT_STEP_HANDLE_KE
                                    frameImgPath = redisUtil.hget(PC_POINT_ST
                                }
                            }

                            if (alarmState.equals(URLConstant.FALSE_STATUS))
                                pcPointRecord.setAlarmState(URLConstant.FALSE
                            }
                            PcPointRecordDetail pcPointRecordDetail = PcPoint

                            boolean hSetNx = redisUtil.hsetnx(PC_POINT_STEP_H
                            if (hSetNx){
                                pcPointRecordDetailMapper.insertPcPointRecord
                                if (pcPointRecordDetail.getAlarmState() == 2)
                                    PcMonitoringDeviceResult build = PcMonitc
                                    pcMonitoringDeviceResultService.insertPcM

                                    PcMonitoringDevice pcMonitoringDevice = p
                                    aiTaskService.publishAlarmInformation(bui
                                }
                            }
                        }
                    }
                    String processIsRunningKey =  callbackResult.getTaskId()
                    redisUtil.hdel(PC_POINT_STEP_HANDLE_KEY + URLConstant.PRO
                    redisUtil.del(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRec

                    pcPointRecord.setEndTime(DateUtils.parseDate(callbackResu
                    pcPointRecord.setPointState(3L);
                    pcPointRecordMapper.updatePcPointRecord(pcPointRecord);
                }
            }
        }
    }
}

public void baffleResultHandle(PcPointRecord pcPointRecord, PcPointStep pcP
    String imagePath = callbackResult.getImagePath();
    List<ModelResult> result = callbackResult.getResult();
    List<ModelResult> collect = result.stream().filter(o -> type.equals(o.get
    PcPointStepDevice pcPointStepDevice = new PcPointStepDevice();
    pcPointStepDevice.setDeviceId(callbackResult.getVideoId());
    pcPointStepDevice.setGroupId(callbackResult.getPointId());
    pcPointStepDevice.setStepId(pcPointStep.getId());
    List<PcPointStepDevice> pcPointStepDevices = pcPointStepDeviceMapper.sele
    if (pcPointStepDevices.isEmpty()) {
        return;
    }
    PcPointStepDevice pcPointStepDeviceInfo = pcPointStepDevices.get(0);
    Map<String, List<ModelResult>> coincideModelResultsType = getCoincideMode
```

```
PcPointRecordDetail pcPointRecordDetail = null;
String keys = PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.getId() + UR
log.info("#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#*#:baffleType: {},keys:{}", baff

boolean isAlarm = false;
if (baffleNum <= 1) {
    if (!coincideModelResultsType.isEmpty()) {
        String s = scpDownFile(sftpConfig, imagePath);
        String frameImgPath = imageResultHandle(coincideModelResultsType,
        pcPointRecordDetail = PcPointRecordDetail.builder().build();

        File file = new File(s);
        if (file.exists()) {
            file.delete();
        }
    } else {
        if (redisUtil.hHasKey(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRec
            Object hget = redisUtil.hget(PC_POINT_STEP_HANDLE_KEY + "_" +
            long i = DateUtils.getDifferenceSeconds(DateUtils.parseDate(c
            if (i > pcPointStep.getMonitorPrefixSecond()) {
                String s = scpDownFile(sftpConfig, imagePath);
                String frameImgPath = imageResultHandle(coincideModelResu
                pcPointRecordDetail = PcPointRecordDetail.builder().build
                File file = new File(s);
                if (file.exists()) {
                    file.delete();
                }
            }
        }
    }
    boolean baffleIsDone = checkStepIsDone(pcPointRecord, pcPointStep, ca
    if (!baffleIsDone && pcPointRecordDetail != null) {
        if (redisUtil.hsetnx(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointReco
            isAlarm = pcPointRecordDetail.getAlarmState() == 2;
            pcPointRecordDetailMapper.insertPcPointRecordDetail(pcPointRe
            boolean bafflePlaceAlready = !isAlarm;
            redisUtil.hset(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord
                    URLConstant.BAFFLE_PLACE_ALREADY + callbackResult.get
        }
    }

} else {
    if (!coincideModelResultsType.isEmpty()) {
        String s = scpDownFile(sftpConfig, imagePath);
        String frameImgPath = imageResultHandle(coincideModelResultsType,
        pcPointRecordDetail = PcPointRecordDetail.builder().build();
        File file = new File(s);
        if (file.exists()) {
            file.delete();
        }
    }
    boolean bafflePlaceAlready = false;
    if (redisUtil.hHasKey(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.
        bafflePlaceAlready = Boolean.parseBoolean(redisUtil.hget(PC_POINT
    }

    boolean pipesReset = redisUtil.hHasKey(PC_POINT_STEP_HANDLE_KEY + "_'

    if (pipesReset && pcPointRecordDetail != null){
        if (!bafflePlaceAlready){
```

```
                pcPointRecordDetail.setAlarmState(URLConstant.FALSE_STATUS);
                pcPointRecordDetail.setDealState(URLConstant.DEAL_STATE_NOT_D
            }

            boolean baffleIsDone = checkStepIsDone(pcPointRecord, pcPointStep
            if (!baffleIsDone && redisUtil.hsetnx(PC_POINT_STEP_HANDLE_KEY +
                isAlarm = pcPointRecordDetail.getAlarmState() == 2;
                pcPointRecordDetailMapper.insertPcPointRecordDetail(pcPointRe
            }
        }
    }
    if (pcPointRecordDetail != null && isAlarm) {
        if (Objects.equals(pcPointRecordDetail.getAlarmState(), URLConstant.F
            if (Objects.equals(pcPointRecord.getAlarmState(), URLConstant.TRU
                pcPointRecord.setAlarmState(URLConstant.FALSE_STATUS);
                pcPointRecord.setDealState(URLConstant.DEAL_STATE_NOT_DEAL);
                pcPointRecordMapper.updatePcPointRecord(pcPointRecord);
            }
            PcMonitoringDeviceResult build = PcMonitoringDeviceResult.builder
            pcMonitoringDeviceResultService.insertPcMonitoringDeviceResult(bu
            PcMonitoringDevice pcMonitoringDevice = pcMonitoringDeviceService
            aiTaskService.publishAlarmInformation(build, pcPoint.getDeptName(
        }
    }
}

public void guardianPersonDutyResultHandle(PcPointRecord pcPointRecord, PcP
    String imagePath = callbackResult.getImagePath();
    List<ModelResult> result = callbackResult.getResult();
    List<ModelResult> collect = result.stream().filter(o -> type.equals(o.get

    PcPointStepDevice pcPointStepDevice = new PcPointStepDevice();
    pcPointStepDevice.setDeviceId(callbackResult.getVideoId());
    pcPointStepDevice.setGroupId(callbackResult.getPointId());
    pcPointStepDevice.setStepId(pcPointStep.getId());
    List<PcPointStepDevice> pcPointStepDevices = pcPointStepDeviceMapper.sele
    if (pcPointStepDevices.isEmpty()) {
        return;
    }
    PcPointStepDevice pcPointStepDeviceInfo = pcPointStepDevices.get(0);
    Map<String, List<ModelResult>> coincideModelResultsType = getCoincideMode
    PcPointRecordDetail pcPointRecordDetail = null;
    boolean isAlarm = false;
    if (coincideModelResultsType.isEmpty()) {
        String s = scpDownFile(sftpConfig, imagePath);
        String frameImgPath = imageResultHandle(coincideModelResultsType, s,

        if (redisUtil.hHasKey(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.
            Object hget = redisUtil.hget(PC_POINT_STEP_HANDLE_KEY + "_" + pcP
            long i = DateUtils.getDifferenceSeconds(DateUtils.parseDate(callb
            if (i > pcPointStep.getMonitorPrefixSecond()) {
                pcPointRecordDetail = PcPointRecordDetail.builder().build();
                isAlarm = true;
            }
        }
        File file = new File(s);
        if (file.exists()) {
            file.delete();
        }
    } else {
```

```
        redisUtil.hset(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.getId()
        redisUtil.hset(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.getId()
    }
    if (isAlarm && !checkStepIsDone(pcPointRecord, pcPointStep, callbackResul
        pcPointRecordDetailMapper.insertPcPointRecordDetail(pcPointRecordDeta
        if (Objects.equals(pcPointRecordDetail.getAlarmState(), URLConstant.F
            if (Objects.equals(pcPointRecord.getAlarmState(), URLConstant.TRU
                pcPointRecord.setAlarmState(URLConstant.FALSE_STATUS);
                pcPointRecord.setDealState(URLConstant.DEAL_STATE_NOT_DEAL);
                pcPointRecordMapper.updatePcPointRecord(pcPointRecord);
            }
            PcMonitoringDeviceResult build = PcMonitoringDeviceResult.builder
            pcMonitoringDeviceResultService.insertPcMonitoringDeviceResult(bu
            PcMonitoringDevice pcMonitoringDevice = pcMonitoringDeviceService
            aiTaskService.publishAlarmInformation(build, pcPoint.getDeptName(
        }
        redisUtil.hdel(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.getId()
        redisUtil.hdel(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.getId()
    }
}


public void connectPipesCorrectResultHandle(PcPointRecord pcPointRecord, Pc
    String imagePath = callbackResult.getImagePath();
    List<ModelResult> result = callbackResult.getResult();
    List<ModelResult> collect = result.stream().filter(o -> type.equals(o.get
    List<ModelResult> tsCollect = result.stream().filter(o -> "ts_" + type.eq

    collect.addAll(tsCollect);

    if (collect.isEmpty()) {
        return;
    }
    PcPointStepDevice pcPointStepDevice = new PcPointStepDevice();
    pcPointStepDevice.setDeviceId(callbackResult.getVideoId());
    pcPointStepDevice.setGroupId(callbackResult.getPointId());
    pcPointStepDevice.setStepId(pcPointStep.getId());
    List<PcPointStepDevice> pcPointStepDevices = pcPointStepDeviceMapper.sele
    if (pcPointStepDevices.isEmpty()) {
        return;
    }
    PcPointStepDevice pcPointStepDeviceInfo = pcPointStepDevices.get(0);

    Map<String, List<ModelResult>> coincideModelResultsType = getCoincideMode
    PcPointRecordDetail pcPointRecordDetail = null;

    if (!coincideModelResultsType.isEmpty()) {
        String s = scpDownFile(sftpConfig, imagePath);
        String frameImgPath = imageResultHandle(coincideModelResultsType, s,
        pcPointRecordDetail = PcPointRecordDetail.builder().build();
        File file = new File(s);
        if (file.exists()) {
            file.delete();
        }
    }

    if (pcPointRecordDetail != null && !checkStepIsDone(pcPointRecord, pcPoin
        boolean hSetNx = redisUtil.hsetnx(PC_POINT_STEP_HANDLE_KEY + "_" + pc
                URLConstant.RESULT_STEP + pcPointStep.getId() + "_" + callbac
                URLConstant.END);
```

```
      if (hSetNx){
          pcPointRecordDetailMapper.insertPcPointRecordDetail(pcPointRecord
          if (Objects.equals(pcPointRecordDetail.getAlarmState(), URLConsta
              if (Objects.equals(pcPointRecord.getAlarmState(), URLConstant
                  pcPointRecord.setAlarmState(URLConstant.FALSE_STATUS);
                  pcPointRecord.setDealState(URLConstant.DEAL_STATE_NOT_DEA
                  pcPointRecordMapper.updatePcPointRecord(pcPointRecord);
              }
              PcMonitoringDeviceResult build = PcMonitoringDeviceResult.bui
              pcMonitoringDeviceResultService.insertPcMonitoringDeviceResul

              PcMonitoringDevice pcMonitoringDevice = pcMonitoringDeviceSer
              aiTaskService.publishAlarmInformation(build, pcPoint.getDeptN
          }
          redisUtil.hset(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.get
      }
  }
}

public void safetySlingResultHandle(PcPointRecord pcPointRecord, PcPointSte
  String imagePath = callbackResult.getImagePath();
  List<ModelResult> result = callbackResult.getResult();
  List<ModelResult> collect = result.stream().filter(o -> type.equals(o.get
  List<ModelResult> peopleCollect = result.stream().filter(o -> targetType.
  PcPointStepDevice pcPointStepDevice = new PcPointStepDevice();
  pcPointStepDevice.setDeviceId(callbackResult.getVideoId());
  pcPointStepDevice.setGroupId(callbackResult.getPointId());
  pcPointStepDevice.setStepId(pcPointStep.getId());
  List<PcPointStepDevice> pcPointStepDevices = pcPointStepDeviceMapper.sele
  if (pcPointStepDevices.isEmpty()) {
      return;
  }
  PcPointStepDevice pcPointStepDeviceInfo = pcPointStepDevices.get(0);

  Map<String, List<ModelResult>> coincideModelResultsType = getCoincideMode
  Map<String, List<ModelResult>> peopleCoincideModelResultsType = getCoinci

  Map<String, List<ModelResult>> allResult = coincideModelResultsType.entry
          .collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue,


  int safetyLineNum = 0;
  int peopleNum = 0;

  for (int i = 0; i < coincideModelResultsType.size(); i++) {
      List<ModelResult> modelResults = coincideModelResultsType.get(i + "")
      safetyLineNum += modelResults.size();
  }


  for (int i = 0; i < peopleCoincideModelResultsType.size(); i++) {
      List<ModelResult> modelResults = peopleCoincideModelResultsType.get(i
      peopleNum += modelResults.size();
  }


  if (peopleCoincideModelResultsType.isEmpty()) {
      redisUtil.hset(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.getId()
              URLConstant.SAFETY_SLING_RESULT_RULE_CHECKING + pcPointStep.g
              DateUtils.getTime());
```

```
    return;
}

PcPointRecordDetail pcPointRecordDetail;
boolean isHasSafetySlingResultRuleChecking = redisUtil.hHasKey(PC_POINT_S
        URLConstant.SAFETY_SLING_RESULT_RULE_CHECKING + pcPointStep.getId

if (safetyLineNum < peopleNum) {
    String frameImgPath;
    if (!redisUtil.hHasKey(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord
        String s = scpDownFile(sftpConfig, imagePath);
        frameImgPath = imageResultHandle(allResult, s, pcPointStep, pcPoi
        redisUtil.hset(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.get
    } else {
        frameImgPath = redisUtil.hget(PC_POINT_STEP_HANDLE_KEY + "_" + pc
    }
    Long monitorPrefixSecond = pcPointStep.getMonitorPrefixSecond();
    if (isHasSafetySlingResultRuleChecking) {
        Object safetySlingResultRuleChecking = redisUtil.hget(PC_POINT_ST
        long times = DateUtils.getDifferenceSeconds(DateUtils.parseDate(c
        if (times > monitorPrefixSecond && !checkStepIsDone(pcPointRecord
            pcPointRecordDetail = PcPointRecordDetail.builder().pointId(c
                    .deviceId(callbackResult.getVideoId())
                    .modelId(pcPointStep.getModelId())
                    .stepId(pcPointStep.getId())
                    .stepName(pcPointStep.getStepName())
                    .recordId(pcPointRecord.getId())
                    .alarmState(URLConstant.FALSE_STATUS)
                    .dealState(URLConstant.DEAL_STATE_NOT_DEAL)
                    .imgUrl(frameImgPath)
                    .startFlag(pcPointStep.getStartFlag())
                    .endFlag(pcPointStep.getEndFlag())
                    .detectTime(DateUtils.parseDate(callbackResult.getTim


            boolean hSetNx = redisUtil.hsetnx(PC_POINT_STEP_HANDLE_KEY +
            if (hSetNx){
                pcPointRecordDetailMapper.insertPcPointRecordDetail(pcPoi

                if (Objects.equals(pcPointRecord.getAlarmState(), URLCons
                    pcPointRecord.setAlarmState(URLConstant.FALSE_STATUS)
                    pcPointRecord.setDealState(URLConstant.DEAL_STATE_NOT
                    pcPointRecordMapper.updatePcPointRecord(pcPointRecord
                }
                PcMonitoringDeviceResult build = PcMonitoringDeviceResult
                pcMonitoringDeviceResultService.insertPcMonitoringDeviceR
                PcMonitoringDevice pcMonitoringDevice = pcMonitoringDevic
                aiTaskService.publishAlarmInformation(build, pcPoint.getD

            }
        }
    } else {
        redisUtil.hset(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.get
                URLConstant.SAFETY_SLING_RESULT_RULE_CHECKING + pcPointSt
                DateUtils.getTime());
    }
} else {
    if (!redisUtil.hHasKey(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord
        String s = scpDownFile(sftpConfig, imagePath);
        String frameImgPath = imageResultHandle(coincideModelResultsType,
```

```
            redisUtil.hset(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.get
                    URLConstant.NO_ALARM_IMAGE_PATH + pcPointStep.getId() + '
                    frameImgPath);
        }
        redisUtil.hset(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.getId()
                URLConstant.SAFETY_SLING_RESULT_RULE_CHECKING + pcPointStep.g
                DateUtils.getTime());
    }
}

public void rearCarPatrolPersonResultHandle(PcPointRecord pcPointRecord, Pc

    String imagePath = callbackResult.getImagePath();
    List<ModelResult> result = callbackResult.getResult();
    List<ModelResult> collect = result.stream().filter(o -> type.equals(o.get

    PcPointStepDevice pcPointStepDevice = new PcPointStepDevice();
    pcPointStepDevice.setDeviceId(callbackResult.getVideoId());
    pcPointStepDevice.setGroupId(callbackResult.getPointId());
    pcPointStepDevice.setStepId(pcPointStep.getId());
    List<PcPointStepDevice> pcPointStepDevices = pcPointStepDeviceMapper.sele
    if (pcPointStepDevices.isEmpty()) {
        return;
    }

    PcPointStepDevice pcPointStepDeviceInfo = pcPointStepDevices.get(0);

    Map<String, List<ModelResult>> coincideModelResultsType = getCoincideMode
    PcPointRecordDetail pcPointRecordDetail = null;
    boolean isAlarm = false;
    boolean connectPipesCorrect = redisUtil.hHasKey(PC_POINT_STEP_HANDLE_KEY
    String state = "continue";
    if (connectPipesCorrect) {
        Object connectPipesCorrectTime = redisUtil.hget(PC_POINT_STEP_HANDLE_
        long i = DateUtils.getDifferenceSeconds(DateUtils.parseDate(callbackR
        if (coincideModelResultsType.isEmpty()) {
            if (i > pcPointStep.getMonitorPrefixSecond()) {

                state = "alarm";
            }
        } else {
            if (i > pcPointStep.getMonitorPrefixSecond()) {

                state = "alarm";
            } else {
                state = "normal";
            }
        }
    }
    if ("continue".equals(state)) {
        return;
    }

    Long alarmState = URLConstant.TRUE_STATUS;
    Long dealState = URLConstant.DEAL_STATE_NOT_NEED_DEAL;
    if ("alarm".equals(state)) {

        alarmState = URLConstant.FALSE_STATUS;
        dealState = URLConstant.DEAL_STATE_NOT_DEAL;
        isAlarm = true;
```

```
  }

  String s = scpDownFile(sftpConfig, imagePath);
  String frameImgPath = imageResultHandle(coincideModelResultsType, s, pcPc
  boolean isDone = checkStepIsDone(pcPointRecord, pcPointStep, callbackResu
  if (!isDone) {
      pcPointRecordDetail = PcPointRecordDetail.builder().build();

      boolean hSetNx = redisUtil.hsetnx(PC_POINT_STEP_HANDLE_KEY + "_" + pc
      if (hSetNx){
          pcPointRecordDetailMapper.insertPcPointRecordDetail(pcPointRecord

          if (!isAlarm) {
              return;
          }

          if (Objects.equals(pcPointRecordDetail.getAlarmState(), URLConsta
              if (Objects.equals(pcPointRecord.getAlarmState(), URLConstant

                  pcPointRecord.setAlarmState(URLConstant.FALSE_STATUS);
                  pcPointRecord.setDealState(URLConstant.DEAL_STATE_NOT_DEA
                  pcPointRecordMapper.updatePcPointRecord(pcPointRecord);
              }


              PcMonitoringDeviceResult build = PcMonitoringDeviceResult.bui
              pcMonitoringDeviceResultService.insertPcMonitoringDeviceResul

              PcMonitoringDevice pcMonitoringDevice = pcMonitoringDeviceSer
              aiTaskService.publishAlarmInformation(build, pcPoint.getDeptN
          }
      }
  }

  File file = new File(s);
  if (file.exists()) {
      file.delete();
  }
}


private void manholeCoverCloseResultHandle(PcPointRecord pcPointRecord, PcP

  String imagePath = callbackResult.getImagePath();
  List<ModelResult> result = callbackResult.getResult();
  List<ModelResult> collect = result.stream().filter(o -> type.equals(o.get
  List<ModelResult> tsCollect = result.stream().filter(o -> "ts_" + type.eq
  collect.addAll(tsCollect);

  PcPointStepDevice pcPointStepDevice = new PcPointStepDevice();
  pcPointStepDevice.setDeviceId(callbackResult.getVideoId());
  pcPointStepDevice.setGroupId(callbackResult.getPointId());
  pcPointStepDevice.setStepId(pcPointStep.getId());
  List<PcPointStepDevice> pcPointStepDevices = pcPointStepDeviceMapper.sele
  if (pcPointStepDevices.isEmpty()) {
      return;
  }

  PcPointStepDevice pcPointStepDeviceInfo = pcPointStepDevices.get(0);
```

```
  Map<String, List<ModelResult>> coincideModelResultsType = getCoincideMode

  PcPointRecordDetail pcPointRecordDetail = null;
  if (redisUtil.hHasKey(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.getI
      String s = scpDownFile(sftpConfig, imagePath);
      String frameImgPath = imageResultHandle(coincideModelResultsType, s,
      if (!coincideModelResultsType.isEmpty()) {
          pcPointRecordDetail = PcPointRecordDetail.builder().build();
      } else {
          pcPointRecordDetail = PcPointRecordDetail.builder().build();
      }

      File file = new File(s);
      if (file.exists()) {
          file.delete();
      }

      boolean isDone = checkStepIsDone(pcPointRecord, pcPointStep, callback
      if (pcPointRecordDetail != null && !isDone) {
          boolean hSetNx = redisUtil.hsetnx(PC_POINT_STEP_HANDLE_KEY + "_"
          if (hSetNx){
              pcPointRecordDetailMapper.insertPcPointRecordDetail(pcPointRe
              if (Objects.equals(pcPointRecordDetail.getAlarmState(), URLCo
                  if (Objects.equals(pcPointRecord.getAlarmState(), URLCons
                      pcPointRecord.setAlarmState(URLConstant.FALSE_STATUS)
                      pcPointRecord.setDealState(URLConstant.DEAL_STATE_NOT
                      pcPointRecordMapper.updatePcPointRecord(pcPointRecord
                  }
                  PcMonitoringDeviceResult build = PcMonitoringDeviceResult
                  pcMonitoringDeviceResultService.insertPcMonitoringDeviceR
                  PcMonitoringDevice pcMonitoringDevice = pcMonitoringDevic
                  aiTaskService.publishAlarmInformation(build, pcPoint.getD
              }
          }
      }
  }
}
private void operateResetResultResetHandle(PcPointRecord pcPointRecord, PcP

  String imagePath = callbackResult.getImagePath();
  if (!redisUtil.hHasKey(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.get
      return;
  }

  List<ModelResult> result = callbackResult.getResult();
  List<ModelResult> collect = result.stream().filter(o -> type.equals(o.get


  PcPointStepDevice pcPointStepDevice = new PcPointStepDevice();
  pcPointStepDevice.setDeviceId(callbackResult.getVideoId());
  pcPointStepDevice.setGroupId(callbackResult.getPointId());
  pcPointStepDevice.setStepId(pcPointStep.getId());
  List<PcPointStepDevice> pcPointStepDevices = pcPointStepDeviceMapper.sele
  if (pcPointStepDevices.isEmpty()) {
      return;
  }
  PcPointStepDevice pcPointStepDeviceInfo = pcPointStepDevices.get(0);
  PcPointRecordDetail pcPointRecordDetail = null;
  if ("pipes".equals(type)){
      String coordinate = pcPointStepDeviceInfo.getCoordinate();
```

```
    com.alibaba.fastjson.JSONArray coordinateArray = JSONObject.parseArra

    boolean pipesReset = true;
    Map<String, List<ModelResult>> coincideModelResultsTypes = new HashMa


    List<ModelResult> collect1 = new ArrayList<>();
    for (Object o : coordinateArray) {
        pcPointStepDeviceInfo.setCoordinate(Collections.singletonList(o).

        Map<String, List<ModelResult>> coincideModelResultsType = getCoin

        if (coincideModelResultsType.isEmpty()) {
            pipesReset = false;
            break;
        }
        List<ModelResult> collect2 = coincideModelResultsType.values().st
        collect1.addAll(collect2);
    }


    if (pipesReset){
        Map<String, List<ModelResult>> coincideModelResultsType = new Has
        coincideModelResultsType.put("0",collect1);
        String s = scpDownFile(sftpConfig, imagePath);
        String frameImgPath = imageResultHandle(coincideModelResultsType,
        pcPointRecordDetail = PcPointRecordDetail.builder().build();

        File file = new File(s);
        if (file.exists()) {
            file.delete();
        }
    }
}else {
    Map<String, List<ModelResult>> coincideModelResultsType = getCoincide

    if (!coincideModelResultsType.isEmpty()) {
        String s = scpDownFile(sftpConfig, imagePath);
        String frameImgPath = imageResultHandle(coincideModelResultsType,
        pcPointRecordDetail = PcPointRecordDetail.builder().build();

        File file = new File(s);
        if (file.exists()) {
            file.delete();
        }
    }
}

boolean isDone = checkStepIsDone(pcPointRecord, pcPointStep, callbackResu
if (pcPointRecordDetail != null && !isDone) {
    boolean hSetNx = redisUtil.hsetnx(PC_POINT_STEP_HANDLE_KEY + "_" + pc
    if (hSetNx){
        pcPointRecordDetailMapper.insertPcPointRecordDetail(pcPointRecord

        if (Objects.equals(pcPointRecordDetail.getAlarmState(), URLConsta
            if (Objects.equals(pcPointRecord.getAlarmState(), URLConstant

                pcPointRecord.setAlarmState(URLConstant.FALSE_STATUS);
                pcPointRecord.setDealState(URLConstant.DEAL_STATE_NOT_DEA
                pcPointRecordMapper.updatePcPointRecord(pcPointRecord);
```

```
            }


            PcMonitoringDeviceResult build = PcMonitoringDeviceResult.bui
            pcMonitoringDeviceResultService.insertPcMonitoringDeviceResul
            PcMonitoringDevice pcMonitoringDevice = pcMonitoringDeviceSer
            aiTaskService.publishAlarmInformation(build, pcPoint.getDeptN
        }
    }

    if ("pipes".equals(type)) {
        redisUtil.hsetnx(PC_POINT_STEP_HANDLE_KEY + "_" + pcPointRecord.g
    }
  }
}
```

```
String screen_url = configService.selectConfigByKey(URLConstant.SCREEN_URL)
if (null == screen_url || screen_url.trim().length() < 1 || screen_url.equa
    return;
ConcurrentHashMap<String, ConcurrentHashMap<String, ConcurrentHashMap<Strin
if (CollectionUtil.isEmpty(webSocketSet))
    return;
List<Long> parentDeptIdList = sysDeptService.getParentDeptIdList(monitoring

ConcurrentHashMap<String, ConcurrentHashMap<String, WebSocketServer>> apiSc
if (CollectionUtil.isNotEmpty(apiScreen)) {
    Map<String, ConcurrentHashMap<String, WebSocketServer>> sendUserMap = a
    sendUserMap.entrySet().stream().forEach(entry -> {
        try {
            String is_or_not_important_alarm = configService.selectConfigBy
            List<String> isOrNotImportantAlarm = StringUtils.isNotBlank(is_
            Map<String, Object> hashMap = new HashMap<>();
            hashMap.put("importantAlarm", !Optional.ofNullable(isOrNotImpor

            entry.getValue().entrySet().stream().forEach(entry_1 -> publish
        } catch (Exception e) {
            logger.error("失败" + e);
        }
    });
}
```

```
public Map<String, Object> getAlarmStatistics(Long userId, String url) {
  Map<String, Object> mapList = new HashMap<>();
  PcMonitoringDeviceResult pcMonitoringDeviceResult = new PcMonitoringDevic
  SysUser sysUser = iSysUserService.selectUserById(userId);
  if (userId != 1L) {
      List<Long> deptIdList = sysDeptService.getSubDeptIdList(sysUser.getDe
      pcMonitoringDeviceResult.setDeptIds(deptIdList);
  }
  Long allCountResult = pcMonitoringDeviceResultMapper.selectPcMonitoringDe
  mapList.put("all", allCountResult);
  Date weekStartTime = LocalDateTimeUtils.convert(LocalDateTimeUtils.weekSt
  Date last7DaysEndTime = LocalDateTimeUtils.convert(LocalDateTimeUtils.las
  Map<String, Object> params = new HashMap<>();
  params.put("beginCreateDate", weekStartTime);
```

```
  params.put("endCreateDate", last7DaysEndTime);
  pcMonitoringDeviceResult.setParams(params);

  Long weekCountResult = pcMonitoringDeviceResultMapper.selectPcMonitoringD
  mapList.put("week", weekCountResult);
  Date last7DaysStartTime = LocalDateTimeUtils.convert(LocalDateTimeUtils.l
  List<String> betweenDays = DateUtils.getBetweenDays(last7DaysStartTime, l
  params.put("beginCreateDate", last7DaysStartTime);
  pcMonitoringDeviceResult.setParams(params);
  Map<String, Object> listMap1 = pcMonitoringDeviceResultMapper.selectPcMon
  Map<String, Object> weeks = new LinkedHashMap<>();
  for (String betweenDay : betweenDays) {
      String value = "0";
      if (Optional.ofNullable(listMap1.get(betweenDay)).isPresent())
          value = new Gson().toJson(new Gson().fromJson(String.valueOf(list
      weeks.put(betweenDay.substring(5).replace("-", "."), (int) Double.par
  }
  mapList.put("detail", weeks);
  return mapList;
}
```

```
public static void scpFile(SftpConfig sftpConfig ,String remote, String loc
  try {
      long startTime = System.currentTimeMillis();
      SshClient client = SshClient.setUpDefaultClient();
      client.start();
      ClientSession session = client.connect(sftpConfig.getUserName(), sftp
      session.addPasswordIdentity(sftpConfig.getPassword());
      boolean isSuccess = session.auth().verify().isSuccess();
      // 认证成功
      if (isSuccess) {
          long middleTime = System.currentTimeMillis();
          ScpClientCreator creator = ScpClientCreator.instance();
          ScpClient scpClient = creator.createScpClient(session);
          scpClient.download(remote,local);
          if (scpClient != null) {
              scpClient = null;
          }
          if (session != null && session.isOpen()) {
              session.close();
          }
          if (client != null && client.isOpen()) {
              client.stop();
              client.close();
          }
      }
      long endTime = System.currentTimeMillis();
  }catch (Exception e){
      log.error(e.getMessage());
  }
  }

  private static long countNum = 0;
  public static ChannelSftp connect(SftpConfig sftpConfig) {
  ChannelSftp sftp = null;
  try {
      JSch jsch = new JSch();
      jsch.getSession(sftpConfig.getUserName(), sftpConfig.getHost(), sftpC
      Session sshSession = jsch.getSession(sftpConfig.getUserName(), sftpCo
```

```
    sshSession.setPassword(sftpConfig.getPassword());
    Properties sshConfig = new Properties();
    sshConfig.put("StrictHostKeyChecking", "no");
    sshSession.setConfig(sshConfig);
    sshSession.connect();
    Channel channel = sshSession.openChannel("sftp");
    channel.connect();
    sftp = (ChannelSftp) channel;
} catch (Exception e) {
    try {
        countNum += 1;
        if (sftpConfig.getCount() == countNum) {
            throw new RuntimeException(e);
        }
        Thread.sleep(sftpConfig.getSleepTime());
        connect(sftpConfig);
    } catch (InterruptedException e1) {
        throw new RuntimeException(e1);
    }
}
return sftp;
}
public static void upload(String directory, String uploadFile, SftpConfig
ChannelSftp sftp = connect(sftpConfig);
try {
    sftp.cd(directory);
} catch (Exception e) {
    try {
        sftp.mkdir(directory);
        sftp.cd(directory);
    } catch (Exception e1) {
        throw new RuntimeException("ftp创建文件路径失败" + directory);
    }
}
File file = new File(uploadFile);
InputStream inputStream=null;
try {
    inputStream = Files.newInputStream(file.toPath());
    sftp.put(inputStream, file.getName());
} catch (Exception e) {
    throw new RuntimeException("sftp异常" + e);
} finally {
    disConnect(sftp);
    closeStream(inputStream,null);
}
}

public static void download(String directory, String downloadFile, String
OutputStream output = null;
try {
    File localDirFile = new File(saveFile);
    if (!localDirFile.exists()) {
        localDirFile.mkdirs();
    }
    ChannelSftp sftp = connect(sftpConfig);
    sftp.cd(directory);
    output = Files.newOutputStream(new File(saveFile.concat(File.separato
    sftp.get(downloadFile, output);
    disConnect(sftp);
} catch (Exception e) {
```

```
    throw new RuntimeException("文件下载出现异常，[{}]", e);
} finally {
    closeStream(null,output);
}
}
public static void getFileDir(String remoteFilePath, String localDirPath,
File localDirFile = new File(localDirPath);
if (!localDirFile.exists()) {
    localDirFile.mkdirs();
}
ChannelSftp channelSftp = connect(sftpConfig);
Vector<ChannelSftp.LsEntry> lsEntries = channelSftp.ls(remoteFilePath);
for (ChannelSftp.LsEntry entry : lsEntries) {
    String fileName = entry.getFilename();
    if (checkFileName(fileName)) {
        continue;
    }
    String remoteFileName = getRemoteFilePath(remoteFilePath, fileName);
    channelSftp.get(remoteFileName, localDirPath);
}
disConnect(channelSftp);
}
private static void closeStream(InputStream inputStream, OutputStream out
if (outputStream != null) {
    try {
        outputStream.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
if(inputStream != null){
    try {
        inputStream.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}

private static boolean checkFileName(String fileName) {
if (".".equals(fileName) || "..".equals(fileName)) {
    return true;
}
return false;
}

private static String getRemoteFilePath(String remoteFilePath, String fil
if (remoteFilePath.endsWith("/")) {
    return remoteFilePath.concat(fileName);
} else {
    return remoteFilePath.concat("/").concat(fileName);
}
}

public static void delete(String directory, String deleteFile, ChannelSft
try {
    sftp.cd(directory);
    sftp.rm(deleteFile);
} catch (Exception e) {
    throw new RuntimeException(e);
```

```
  }
  }
  public static List<String> listFiles(String directory, SftpConfig sftpCon
  ChannelSftp sftp = connect(sftpConfig);
  List fileNameList = new ArrayList();
  try {
      sftp.cd(directory);
  } catch (SftpException e) {
      return fileNameList;
  }
  Vector vector = sftp.ls(directory);
  for (int i = 0; i < vector.size(); i++) {
      if (vector.get(i) instanceof ChannelSftp.LsEntry) {
          ChannelSftp.LsEntry lsEntry = (ChannelSftp.LsEntry) vector.get(i)
          String fileName = lsEntry.getFilename();
          if (".".equals(fileName) || "..".equals(fileName)) {
              continue;
          }
          fileNameList.add(fileName);
      }
  }
  disConnect(sftp);
  return fileNameList;
  }
  public static void disConnect(ChannelSftp sftp) {
  try {
      sftp.disconnect();
      sftp.getSession().disconnect();
  } catch (Exception e) {
      e.printStackTrace();
  }
  }


public static String downloadFromSftp(String remoteFile,String localFile,Sf
    ChannelSftp connect = connect(sftpConfig);
    try {
        connect.get(remoteFile,localFile);
    }catch (Exception e){
        log.error(e.getMessage());
    }finally {
        connect.disconnect();
    }
    if (new File(localFile).exists()){
        return localFile;
    }
    return null;
}

public LinkedHashMap<String, Object> workOrderStatistics(PcWoTask pcWoTask)
Date last7DaysStartTime = LocalDateTimeUtils.convert(LocalDateTimeUtils.las
Date last7DaysEndTime = LocalDateTimeUtils.convert(LocalDateTimeUtils.last7

List<String> betweenDays = DateUtils.getBetweenDays(last7DaysStartTime, las
Map<String,Object> params = new HashMap<>();
params.put("beginDate",last7DaysStartTime);
params.put("endDate",last7DaysEndTime);
pcWoTask.setParams(params);

List<PcWoTask> pcWoTasks = pcWoTaskMapper.selectPcWoTaskListForFormatDate(p
```

```
Map<String, List<PcWoTask>> collectForAiModel = pcWoTasks.stream().collect(

LinkedHashMap<String,Object> echartsRes = new LinkedHashMap<>();
echartsRes.put("all",pcWoTasks.size());
echartsRes.put("unprocessed",pcWoTasks.stream().filter(a -> a.getWoTaskStat

//周天数数据
List<String> weeks = new ArrayList<>();
for (String betweenDay : betweenDays) {
    weeks.add(DateUtils.dateToWeek(DateUtils.parseDate(betweenDay)));
}
echartsRes.put("weeks",weeks);

//组装每天的每个模型的工单数量
Map<String,List<Integer>> numsDetail = new HashMap<>();

collectForAiModel.forEach((modelName, pcWoTasksForModels) -> {
    List<Integer> workOrderNums = new ArrayList<>();
    for (String betweenDay : betweenDays) {
        int nums = 0;
        for (PcWoTask pcWoTasksForModel : pcWoTasksForModels) {
            if (betweenDay.equals(pcWoTasksForModel.getCreateDate())){
                nums +=1;
            }
        }
        workOrderNums.add(nums);
    }
    numsDetail.put(modelName,workOrderNums);
});
echartsRes.put("detail",numsDetail);
return echartsRes;
}
```

SpringBatch

一、课程目标

一、课程目标

课程目标

- 系统了解Spring Batch批处理
- 项目中能熟练使用Spring Batch批处理

课程内容

前置知识

- Java基础
- Maven
- Spring SpringMVC SpringBoot
- MyBatis

适合人群

- 想学习的所有人

二、Spring Batch简介

2.1 何为批处理？

何为批处理，大白话：就是将数据分批次进行处理的过程。比如：银行对账逻辑，跨系统数据同步等。

常规的批处理操作步骤：系统**A**从数据库中导出数据到文件，系统**B**读取文件数据并写入到数据库

典型批处理特点：

- 自动执行，根据系统设定的工作步骤自动完成
- 数据量大，少则百万，多则上千万甚至上亿。(如果是10亿，100亿那只能上大数据了)
- 定时执行，比如：每天，每周，每月执行。

2.2 Spring Batch了解

官网介绍：https://docs.spring.io/spring-batch/docs/current/reference/html/spring-batch-intro.html#spring-batch-intro

这里挑重点讲下：

- Sping Batch 是一个轻量级的、完善的的批处理框架，旨在帮助企业建立健壮、高效的批处理应用。
- Spring Batch 是Spring的一个子项目，基于Spring框架为基础的开发的框架
- Spring Batch 提供大量可重用的组件，比如：日志，追踪，事务，任务作业统计，任务重启，跳过，重复，资源管理等
- Spring Batch 是一个批处理应用框架，不提供调度框架，如果需要定时处理需要额外引入-调度框架，比如： Quartz

2.3 Spring Batch 优势

Spring Batch 框架通过提供丰富的开箱即用的组件和高可靠性、高扩展性的能力，使得开发批处理应用的人员专注于业务处理，提高处理应用的开发能力。下面就是使用Spring Batch后能获取到优势：

- 丰富的开箱即用组件
- 面向Chunk的处理

- 事务管理能力
- 元数据管理
- 易监控的批处理应用
- 丰富的流程定义
- 健壮的批处理应用
- 易扩展的批处理应用
- 复用企业现有的IT代码

2.4 Spring Batch 架构

Spring Batch 核心架构分三层：应用层，核心层，基础架构层。

Application：应用层，包含所有的批处理作业，程序员自定义代码实现逻辑。

Batch Core：核心层，包含Spring Batch启动和控制所需要的核心类，比如：JobLauncher， Job，Step等。

Batch Infrastructure：基础架构层，提供通用的读，写与服务处理。

三层体系使得Spring Batch 架构可以在不同层面进行扩展，避免影响，实现高内聚低耦合设计。

三、入门案例

3.1 批量处理流程

前面对Spring Batch 有大体了解之后，那么开始写个案例玩一下。

开始前，先了解一下Spring Batch程序运行大纲：

JobLauncher：作业调度器，作业启动主要入口。

Job：作业，需要执行的任务逻辑，

Step：作业步骤，一个Job作业由1个或者多个Step组成，完成所有Step操作，一个完整Job才算执行结束。

ItemReader：Step步骤执行过程中数据输入。可以从数据源(文件系统，数据库，队列等)中读取Item(数据记录)。

ItemWriter：Step步骤执行过程中数据输出，将Item(数据记录)写入数据源(文件系统，数据库，队列等)。

ItemProcessor：Item数据加工逻辑(输入)，比如：数据清洗，数据转换，数据过滤，数据校验等

JobRepository： 保存Job或者检索Job的信息。SpringBatch需要持久化Job(可以选择数据库/内存)，JobRepository就是持久化的接口

3.2 入门案例-H2版(内存)

需求：打印一个**hello spring batch**！不带读/写/处理

步骤**1**：导入依赖

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.3</version>
    <relativePath/>
</parent>
<dependencies>
```

```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-batch</artifactId>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
    </dependency>

        <!--内存版-->
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
    </dependency>

</dependencies>
```

其中的h2是一个嵌入式内存数据库，后续可以使用MySQL替换

步骤**2**：创建测试方法

```
package com.langfeiyes.batch._01_hello;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
@EnableBatchProcessing
public class HelloJob {
    //job调度器
    @Autowired
    private JobLauncher jobLauncher;
    //job构造器工厂
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    //step构造器工厂
    @Autowired
    private StepBuilderFactory stepBuilderFactory;
    //任务-step执行逻辑由tasklet完成
    @Bean
    public Tasklet tasklet(){
        return new Tasklet() {
```

```
            @Override
            public RepeatStatus execute(StepContribution contribution, Chur
                System.out.println("Hello SpringBatch....");
                return RepeatStatus.FINISHED;
            }
        };
    }
    //作业步骤-不带读/写/处理
    @Bean
    public Step step1(){
        return stepBuilderFactory.get("step1")
                .tasklet(tasklet())
                .build();
    }
    //定义作业
    @Bean
    public Job job(){
        return jobBuilderFactory.get("hello-job")
                .start(step1())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(HelloJob.class, args);
    }

}
```

步骤**3**：分析

例子是一个简单的SpringBatch 入门案例，使用了最简单的一种步骤处理模型：Tasklet模型，step1中没有带上读/写/处理逻辑，只有简单打印操作，后续随学习深入，我们再讲解更复杂化模型。

3.3 入门案例-MySQL版

MySQL跟上面的h2一样，区别在连接数据库不一致。

步骤**1**：在**H2**版本基础上导入**MySQL**依赖

```
<!-- <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency> -->

<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.12</version>
</dependency>
```

步骤**2**：配置数据库四要素与初始化**SQL**脚本

```
spring:
  datasource:
```

```
    username: root
    password: admin
    url: jdbc:mysql://127.0.0.1:3306/springbatch?serverTimezone=GMT%2B8&use
    driver-class-name: com.mysql.cj.jdbc.Driver
    # 初始化数据库，文件在依赖jar包中
  sql:
    init:
      schema-locations: classpath:org/springframework/batch/core/schema-mys
      mode: always
      #mode: never
```

这里要注意， sql.init.model 第一次启动为always， 后面启动需要改为never，否则每次执行SQL都会异常。

第一次启动会自动执行指定的脚本，后续不需要再初始化

步骤**3**：测试

跟H2版一样。

四、入门案例解析

1>@EnableBatchProcessing

批处理启动注解，要求贴配置类或者启动类上

```
@SpringBootApplication
@EnableBatchProcessing
public class HelloJob {
    ...
}
```

贴上@EnableBatchProcessing注解后，SpringBoot会自动加载JobLauncher JobBuilderFactory StepBuilderFactory 类并创建对象交给容器管理，要使用时，直接@Autowired即可

```
//job调度器
@Autowired
private JobLauncher jobLauncher;
//job构造器工厂
@Autowired
private JobBuilderFactory jobBuilderFactory;
//step构造器工厂
@Autowired
```

```
private StepBuilderFactory stepBuilderFactory;
```

2>配置数据库四要素

批处理允许重复执行，异常重试，此时需要保存批处理状态与数据，Spring Batch 将数据缓存在H2内存中或者缓存在指定数据库中。入门案例如果要保存在MySQL中，所以需要配置数据库四要素。

3>创建**Tasklet**对象

```
//任务-step执行逻辑由tasklet完成
@Bean
public Tasklet tasklet(){
    return new Tasklet() {
        @Override
        public RepeatStatus execute(StepContribution contribution, ChunkCon
            System.out.println("Hello SpringBatch....");
            return RepeatStatus.FINISHED;
        }
    };
}
```

Tasklet负责批处理step步骤中具体业务执行，它是一个接口，有且只有一个execute方法，用于定制step执行逻辑。

```
public interface Tasklet {
        RepeatStatus execute(StepContribution contribution, ChunkContext ch
}
```

execute方法返回值是一个状态枚举类：RepeatStatus，里面有可继续执行态与已经完成态

```
public enum RepeatStatus {
        /**
         * 可继续执行的-tasklet返回这个状态会进入死循环
         */
        CONTINUABLE(true),
        /**
         * 已经完成态
         */
        FINISHED(false);
    ....
}
```

4>创建**Step**对象

```
//作业步骤-不带读/写/处理
@Bean
public Step step1(){
    return stepBuilderFactory.get("step1")
        .tasklet(tasklet())
        .build();
}
```

Job作业执行靠Step步骤执行，入门案例选用最简单的Tasklet模式，后续再讲Chunk块处

理模式。

5>创建**Job**并执行**Job**

```
//定义作业
@Bean
public Job job(){
    return jobBuilderFactory.get("hello-job")
        .start(step1())
        .build();
}
```

创建Job对象交给容器管理，当springboot启动之后，会自动去从容器中加载Job对象，并将Job对象交给JobLauncherApplicationRunner类，再借助JobLauncher类实现job执行。

验证过程；

打断点，debug模式启动

SpringApplication类**run**方法

JobLauncherApplicationRunner类

JobLauncher接口–实现类：SimpleJobLauncher

五、作业对象 Job

5.1 作业介绍

5.1.1 作业定义

Job作业可以简单理解为一段业务流程的实现，可以根据业务逻辑拆分一个或者多个逻辑块(step)，然后业务逻辑顺序，逐一执行。

所以作业可以定义为：能从头到尾独立执行的有序的步骤**(Step)**列表。

- 有序的步骤列表

一次作业由不同的步骤组成，这些步骤顺序是有意义的，如果不按照顺序执行，会引起逻辑混乱，比如购物结算，先点结算，再支付，最后物流，如果反过来那就乱套了，作业也是这么一回事。

- 从头到尾

一次作业步骤固定了，在没有外部交互情况下，会从头到尾执行，前一个步骤做完才会到后一个步骤执行，不允许随意跳转，但是可以按照一定逻辑跳转。

- 独立

每一个批处理作业都应该不受外部依赖影响情况下执行。

看回这幅图，批处理作业Job是由一组步骤Step对象组成，每一个作业都有自己名称，可以定义Step执行顺序。

5.1.2 作业代码设计

前面定义讲了作业执行是相互独立的，代码该怎么设计才能保证每次作业独立的性呢？

答案是：**Job instance**(作业实例) 与 **Job Execution**(作业执行对象)

Job instance(作业实例)

当作业运行时，会创建一个Job Instance(作业实例)，它代表作业的一次逻辑运行，可通过作业名称与作业标识参数进行区分。

比如一个业务需求： 每天定期数据同步，作业名称**-daily-sync-job** 作业标记参数**-**当天时间

Job Execution(作业执行对象)

当作业运行时，也会创建一个Job Execution(作业执行器)，负责记录Job执行情况(比如：开始执行时间，结束时间，处理状态等)。

那为啥会出现上面架构设计呢？原因：批处理执行过程中可能出现两种情况：

- 一种是一次成功

仅一次就成从头到尾正常执行完毕，在数据库中会记录一条Job Instance 信息， 跟一条Job Execution 信息

- 另外一种异常执行

在执行过程因异常导致作业结束，在数据库中会记录一条Job Instance 信息， 跟一条Job Execution 信息。如果此时使用相同识别参数再次启动作业，那么数据库中不会多一条Job Instance 信息， 但是会多了一条Job Execution 信息，这就意味中任务重复执行了。刚刚说每天批处理任务案例，如果当天执行出异常，那么人工干预修复之后，可以再次执

行。

最后来个总结：

Job Instance = Job名称 **+** 识别参数

Job Instance 一次执行创建一个 **Job Execution**对象

完整的一次**Job Instance** 执行可能创建一个**Job Execution**对象，也可能创建多个**Job Execution**对象

5.2 作业配置

再看回入门案例

```
package com.langfeiyes.batch._01_hello;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
@EnableBatchProcessing
public class HelloJob {
    //job构造器工厂
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    //step构造器工厂
    @Autowired
    private StepBuilderFactory stepBuilderFactory;
    //任务-step执行逻辑由tasklet完成
    @Bean
    public Tasklet tasklet(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("Hello SpringBatch....");
                return RepeatStatus.FINISHED;
            }
        };
    }
    //作业步骤-不带读/写/处理
    @Bean
    public Step step1(){
        return stepBuilderFactory.get("step1")
                .tasklet(tasklet())
```

```
                .build();
    }
    //定义作业
    @Bean
    public Job job(){
        return jobBuilderFactory.get("hello-job")
                .start(step1())
                .build();
    }

    public static void main(String[] args) {
        SpringApplication.run(HelloJob.class, args);
    }

}
```

在启动类中贴上@EnableBatchProcessing注解，SpringBoot会自动听JobLauncher JobBuilderFactory StepBuilderFactory 对象，分别用于执行Jog，创建Job，创建Step逻辑。有了这些逻辑，Job批处理就剩下组装了。

5.3 作业参数

5.3.1 JobParameters

前面提到，作业的启动条件是作业名称 + 识别参数，Spring Batch使用**JobParameters**类来封装了所有传给作业参数。

我们看下JobParameters 源码

```
public class JobParameters implements Serializable {

        private final Map<String,JobParameter> parameters;

        public JobParameters() {
                this.parameters = new LinkedHashMap<>();
        }

        public JobParameters(Map<String,JobParameter> parameters) {
                this.parameters = new LinkedHashMap<>(parameters);
        }
    .....
}
```

从上面代码/截图来看，JobParameters 类底层维护了Map，是一个Map集合的封装器，提供了不同类型的get操作。

5.3.2 作业参数设置

还记得Spring Batch 入门案例吗，当初debug时候看到Job作业最终是调用时 **JobLauncher **(job启动器)接口run方法启动。

看下源码：JobLauncher

```
public interface JobLauncher {
        public JobExecution run(Job job, JobParameters jobParameters) throw
                        JobRestartException, JobInstanceAlreadyCompleteExce

}
```

在JobLauncher 启动器执行run方法时，直接传入即可。

```
jobLauncher.run(job, params);
```

那我们使用SpringBoot 方式启动Spring Batch该怎么传值呢？

1>定义**ParamJob**类，准备好要执行的**job**

```
package com.langfeiyes.batch._02_params;

import org.springframework.batch.core.*;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.configuration.annotation.StepScope;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;


@SpringBootApplication
@EnableBatchProcessing
public class ParamJob {
    //job构造器工厂
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    //step构造器工厂
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public Tasklet tasklet(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("param SpringBatch....");
                return RepeatStatus.FINISHED;
            }
        };
    }
    @Bean
    public Step  step1(){
        return  stepBuilderFactory.get("step1")
                .tasklet(tasklet())
                .build();
    }
    @Bean
    public Job job(){
        return jobBuilderFactory.get("param-job")
                .start(step1())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(HelloJob.class, args);
    }
}
```

2>使用**idea**的命令传值的方式设置**job**作业参数

> 注意：如果不想这么麻烦，其实也可以，先空参数执行一次，然后指定参数后再执行。

点击绿色按钮，启动SpringBoot程序，作业运行之后，会在batch_job_execution_params增加一条记录，用于区分唯一的Job Instance实例

注意：如果不改动**JobParameters** 参数内容，再执行一次批处理，会直接报错。

原因：**Spring Batch** 相同**Job**名与相同标识参数只能成功执行一次。

5.3.3 作业参数获取

当将作业参数传入到作业流程，该如何获取呢？

Spring Batch 提供了2种方案：

方案**1**：使用**ChunkContext**类

ParamJob类中tasklet写法

```
@Bean
public Tasklet tasklet(){
    return new Tasklet() {
        @Override
        public RepeatStatus execute(StepContribution contribution, ChunkCor
            Map<String, Object> parameters = chunkContext.getStepContext().
            System.out.println("params---name:" + parameters.get("name"));
            return RepeatStatus.FINISHED;
        }
    };
}
```

注意：**job**名：**param-job job**参数：**name=dafei** 已经执行了，再执行会报错

所以要么改名字，要么改参数，这里选择改**job**名字（拷贝一份**job**实例方法，然后注释掉，修改**Job**名称）

```
//    @Bean
//    public Job job(){
//        return jobBuilderFactory.get("param-job")
//                .start(step1())
//                .build();
//    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("param-chunk-job")
                .start(step1())
                .build();
    }
```

方案**2**：使用**@Value** 延时获取

```
@StepScope
@Bean
public Tasklet tasklet(@Value("#{jobParameters['name']}")String name){
    return new Tasklet() {
        @Override
        public RepeatStatus execute(StepContribution contribution, ChunkCor
            System.out.println("params---name:" + name);
            return RepeatStatus.FINISHED;
        }
    };
}

@Bean
public Step  step1(){
    return  stepBuilderFactory.get("step1")
        .tasklet(tasklet(null))
        .build();
}
```

step1调用tasklet实例方法时不需要传任何参数，Spring Boot 在加载Tasklet Bean实例时会自动注入。

```
//    @Bean
//    public Job job(){
```

```
//        return jobBuilderFactory.get("param-chunk-job")
//                .start(step1())
//                .build();
//    }

@Bean
public Job job(){
    return jobBuilderFactory.get("param-value-job")
        .start(step1())
        .build();
}
```

这里要注意，必须贴上**@StepScope** ，表示在启动项目的时候，不加载该Step步骤bean，等step1()被调用时才加载。这就是所谓延时获取。

5.3.4 作业参数校验

当外部传入的参数进入作业时，如何确保参数符合期望呢？使用Spring Batch 的参数校验器：**JobParametersValidator** 接口。

先来看下JobParametersValidator 接口源码：

```
public interface JobParametersValidator {
        void validate(@Nullable JobParameters parameters) throws JobParamet
}
```

JobParametersValidator 接口有且仅有唯一的validate方法，参数为JobParameters，没有返回值。这就意味着不符合参数要求，需要抛出异常来结束步骤。

定制参数校验器

Spring Batch 提供JobParametersValidator参数校验接口，其目的就是让我们通过实现接口方式定制参数校验逻辑。

需求：如果传入作业的参数**name**值 为**null** 或者 **“”** 时报错

```
public class NameParamValidator  implements JobParametersValidator {
    @Override
    public void validate(JobParameters parameters) throws JobParametersInva
        String name = parameters.getString("name");

        if(!StringUtils.hasText(name)){
            throw new JobParametersInvalidException("name 参数不能为空");
        }
    }
}
```

其中的JobParametersInvalidException 异常是Spring Batch 专门提供参数校验失败异常，当然我们也可以自定义或使用其他异常。

```
package com.langfeiyes.batch._03_param_validator;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
```

```
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.configuration.annotation.StepScope;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

import java.util.Map;

@SpringBootApplication
@EnableBatchProcessing
public class ParamValidatorJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public Tasklet tasklet(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                Map<String, Object> parameters = chunkContext.getStepContex
                System.out.println("params---name:" + parameters.get("name'
                return RepeatStatus.FINISHED;
            }
        };
    }

    @Bean
    public Step  step1(){
        return  stepBuilderFactory.get("step1")
                .tasklet(tasklet())
                .build();
    }

    //配置name参数校验器
    @Bean
    public NameParamValidator validator(){
        return new NameParamValidator();
    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("name-param-validator-job")
                .start(step1())
                .validator(validator())  //参数校验器
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(ParamValidatorJob.class, args);
    }
}
```

```
```

新定义**validator()**实例方法，将定制的参数解析器加到Spring容器中，修改job()实例方法，加上**.validator(validator())** 校验逻辑。

第一次启动时，没有传任何参数

```
String name = parameters.getString("name");
```

name为null，直接报错

加上name=dafei参数之后，正常执行

默认参数校验器

除去上面的定制参数校验器外，Spring Batch 也提供2个默认参数校验器：DefaultJobParametersValidator(默认参数校验器) 跟 CompositeJobParametersValidator(组合参数校验器)。

DefaultJobParametersValidator参数校验器

```
public class DefaultJobParametersValidator implements JobParametersValidato
        private Collection<String> requiredKeys;
        private Collection<String> optionalKeys;
    ....
}
```

默认的参数校验器它功能相对简单，维护2个key集合requiredKeys 跟 optionalKeys

- requiredKeys 是一个集合，表示作业参数jobParameters中必须包含集合中指定的keys
- optionalKeys 也是一个集合，该集合中的key 是可选参数

需求：如果作业参数没有**name**参数报错，**age**参数可有可无

```
package com.langfeiyes.batch._03_param_validator;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
```

```
import org.springframework.batch.core.configuration.annotation.StepScope;
import org.springframework.batch.core.job.DefaultJobParametersValidator;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

import java.util.Map;

@SpringBootApplication
@EnableBatchProcessing
public class ParamValidatorJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public Tasklet tasklet(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                Map<String, Object> parameters = chunkContext.getStepContex
                System.out.println("params---name:" + parameters.get("name'
                System.out.println("params---age:" + parameters.get("age"))
                return RepeatStatus.FINISHED;
            }
        };
    }

    @Bean
    public Step  step1(){
        return  stepBuilderFactory.get("step1")
                .tasklet(tasklet())
                .build();
    }

    //配置name参数校验器
    @Bean
    public NameParamValidator validator(){
        return new NameParamValidator();
    }

     //配置默认参数校验器
    @Bean
    public DefaultJobParametersValidator defaultValidator(){
        DefaultJobParametersValidator defaultValidator = new DefaultJobPara
        defaultValidator.setRequiredKeys(new String[]{"name"});  //必填
        defaultValidator.setOptionalKeys(new String[]{"age"});   //可选
        return defaultValidator;
    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("default-param-validator-job")
                .start(step1())
```

```
                //.validator(validator())  //参数校验器
                .validator(defaultValidator())  //默认参数校验器
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(ParamValidatorJob.class, args);
    }
}
```

新定义defaultValidator() 实例方法，将默认参数解析器加到Spring容器中，修改job实例方法，加上**.validator(defaultValidator())**。

右键启动，不填name 跟 age 参数，直接报错

如果填上name参数，即使不填age参数，可以通过，原因是age是可选的。

组合参数校验器

CompositeJobParametersValidator 组合参数校验器，顾名思义就是将多个参数校验器组合在一起。

看源码，大体能看出该校验器逻辑

```
public class CompositeJobParametersValidator implements JobParametersValida

        private List<JobParametersValidator> validators;

        @Override
        public void validate(@Nullable JobParameters parameters) throws Job
                for (JobParametersValidator validator : validators) {
                        validator.validate(parameters);
                }
        }

        public void setValidators(List<JobParametersValidator> validators)
                this.validators = validators;
        }

        @Override
        public void afterPropertiesSet() throws Exception {
                Assert.notNull(validators, "The 'validators' may not be nul
                Assert.notEmpty(validators, "The 'validators' may not be em
        }
}
```

底层维护一个validators 集合，校验时调用validate 方法，依次执行校验器集合中校验器方法。另外，多了一个afterPropertiesSet方法，用于校验validators 集合中的校验器是否

为null。

需求：要求步骤中必须有**name**属性，并且不能为空

分析：必须有，使用DefaultJobParametersValidator 参数校验器， 不能为null，使用指定定义的NameParamValidator参数校验器

```
package com.langfeiyes.batch._03_param_validator;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.configuration.annotation.StepScope;
import org.springframework.batch.core.job.CompositeJobParametersValidator;
import org.springframework.batch.core.job.DefaultJobParametersValidator;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

import java.util.Arrays;
import java.util.Map;

@SpringBootApplication
@EnableBatchProcessing
public class ParamValidatorJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public Tasklet tasklet(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                Map<String, Object> parameters = chunkContext.getStepContex
                System.out.println("params---name:" + parameters.get("name"
                System.out.println("params---age:" + parameters.get("age"))
                return RepeatStatus.FINISHED;
            }
        };
    }

    @Bean
    public Step  step1(){
        return  stepBuilderFactory.get("step1")
                .tasklet(tasklet())
                .build();
    }

    //配置name参数校验器
```

```
    @Bean
    public NameParamValidator validator(){
        return new NameParamValidator();
    }

     //配置默认参数校验器
    @Bean
    public DefaultJobParametersValidator defaultValidator(){
        DefaultJobParametersValidator defaultValidator = new DefaultJobPara
        defaultValidator.setRequiredKeys(new String[]{"name"});  //必填
        defaultValidator.setOptionalKeys(new String[]{"age"});   //可选
        return defaultValidator;
    }

    //配置组合参数校验器
    @Bean
    public CompositeJobParametersValidator compositeValidator(){

        DefaultJobParametersValidator defaultValidator = new DefaultJobPara
        defaultValidator.setRequiredKeys(new String[]{"name"});  //name必填
        defaultValidator.setOptionalKeys(new String[]{"age"});   //age可选

        NameParamValidator nameParamValidator = new NameParamValidator();

        CompositeJobParametersValidator compositeValidator = new CompositeJ
        //按照传入的顺序，先执行defaultValidator 后执行nameParamValidator
        compositeValidator.setValidators(Arrays.asList(defaultValidator, na

        try {
            compositeValidator.afterPropertiesSet();  //判断校验器是否为null
        } catch (Exception e) {
            e.printStackTrace();
        }

        return compositeValidator;
    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("composite-param-validator-job")
                .start(step1())
                //.validator(validator())  //参数校验器
                //.validator(defaultValidator())  //默认参数校验器
                .validator(compositeValidator())  //组合参数校验器
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(ParamValidatorJob.class, args);
    }
}
```

新定义compositeValidator() 实例方法，将组合参数解析器加到spring容器中，修改job()实例方法，加上**.validator(compositeValidator())**。

右键启动，不填name参数，测试报错。如果放开name参数，传null值，一样报错。

5.3.5 作业增量参数

不知道大家发现了没有，每次运行作业时，都改动作业名字，或者改动作业的参数，原因是作业启动有限制：相同标识参数与相同作业名的作业，只能成功运行一次。那如果想每次启动，又不想改动标识参数跟作业名怎么办呢？答案是：使用 **JobParametersIncrementer (**作业参数增量器**)**

看下源码，了解一下原理

```
public interface JobParametersIncrementer {
        JobParameters getNext(@Nullable JobParameters parameters);

}
```

JobParametersIncrementer 增量器是一个接口，里面只有getNext方法，参数是 JobParameters 返回值也是JobParameters。通过这个getNext方法，在作业启动时我们可以给JobParameters 添加或者修改参数。简单理解就是让标识参数每次都变动

作业递增**run.id**参数

Spring Batch 提供一个run.id自增参数增量器：**RunIdIncrementer**，每次启动时，里面维护名为**run.id** 标识参数，每次启动让其自增 1。

看下源码：

```
public class RunIdIncrementer implements JobParametersIncrementer {

        private static String RUN_ID_KEY = "run.id";

        private String key = RUN_ID_KEY;

        public void setKey(String key) {
                this.key = key;
        }

        @Override
        public JobParameters getNext(@Nullable JobParameters parameters) {

                JobParameters params = (parameters == null) ? new JobParame
                JobParameter runIdParameter = params.getParameters().get(th
                long id = 1;
                if (runIdParameter != null) {
                        try {
                                id = Long.parseLong(runIdParameter.getValue
                        }
                        catch (NumberFormatException exception) {
                                throw new IllegalArgumentException("Invalid
                                                + this.key, exception);
```

```
                        }
                }
                return new JobParametersBuilder(params).addLong(this.key, i
        }

}
```

核心getNext方法，在JobParameters 对象维护一个**run.id**，每次作业启动时，都调用getNext方法获取JobParameters，保证其 **run.id** 参数能自增1

具体用法：

```
package com.langfeiyes.batch._04_param_incr;

import com.langfeiyes.batch._03_param_validator.NameParamValidator;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.configuration.annotation.StepScope;
import org.springframework.batch.core.job.CompositeJobParametersValidator;
import org.springframework.batch.core.job.DefaultJobParametersValidator;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

import java.util.Arrays;
import java.util.Map;

@SpringBootApplication
@EnableBatchProcessing
public class IncrementParamJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public Tasklet tasklet(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                Map<String, Object> parameters = chunkContext.getStepContex
                System.out.println("params---run.id:" + parameters.get("run
                return RepeatStatus.FINISHED;
            }
        };
    }

    @Bean
    public Step  step1(){
```

```
        return  stepBuilderFactory.get("step1")
                .tasklet(tasklet())
                .build();
    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("incr-params-job")
                .start(step1())
                .incrementer(new RunIdIncrementer())  //参数增量器(run.id自增
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(IncrementParamJob.class, args);
    }
}
```

修改tasklet()方法，获取**run.id**参数，修改job实例方法，加上**.incrementer(new RunIdIncrementer())**，保证参数能自增。

连续执行3次，观察：**batch_job_execution_params** 表

其中的run.id参数值一直增加，其中再多遍也没啥问题。

作业时间戳参数

run.id 作为标识参数貌似没有具体业务意义，如果将时间戳作为标识参数那就不一样了，比如这种运用场景：每日任务批处理，这时就需要记录每天的执行时间了。那该怎么实现呢？

Spring Batch 中没有现成时间戳增量器，需要自己定义

```
//时间戳作业参数增量器
public class DailyTimestampParamIncrementer implements JobParametersIncreme
    @Override
    public JobParameters getNext(JobParameters parameters) {
        return new JobParametersBuilder(parameters)
                .addLong("daily", new Date().getTime())  //添加时间戳
                .toJobParameters();
    }
}
```

定义一个标识参数：daily，记录当前时间戳

```
package com.langfeiyes.batch._04_param_incr;

import com.langfeiyes.batch._03_param_validator.NameParamValidator;
import org.springframework.batch.core.Job;
```

```
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.configuration.annotation.StepScope;
import org.springframework.batch.core.job.CompositeJobParametersValidator;
import org.springframework.batch.core.job.DefaultJobParametersValidator;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

import java.util.Arrays;
import java.util.Map;

@SpringBootApplication
@EnableBatchProcessing
public class IncrementParamJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public Tasklet tasklet(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                Map<String, Object> parameters = chunkContext.getStepContex
                System.out.println("params---daily:" + parameters.get("dail
                return RepeatStatus.FINISHED;
            }
        };
    }

    //时间戳增量器
    @Bean
    public DailyTimestampParamIncrementer dailyTimestampParamIncrementer(){
        return new DailyTimestampParamIncrementer();
    }


    @Bean
    public Step  step1(){
        return  stepBuilderFactory.get("step1")
                .tasklet(tasklet())
                .build();
    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("incr-params-job")
                .start(step1())
                //.incrementer(new RunIdIncrementer())  //参数增量器(run.id自
                .incrementer(dailyTimestampParamIncrementer())  //时间戳增量
```

```
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(IncrementParamJob.class, args);
    }
}
```

定义实例方法**dailyTimestampParamIncrementer()**将自定义时间戳增量器添加Spring容器中，修改job()实例方法，添加**.incrementer(dailyTimestampParamIncrementer())** 增量器，修改tasklet() 方法，获取 **daily**参数。

连续执行3次，查看**batch_job_execution_params** 表

很明显可以看出daily在变化，而run.id 没有动，是3，为啥？因为**.incrementer(new RunIdIncrementer())** 被注释掉了。

5.4 作业监听器

作业监听器：用于监听作业的执行过程逻辑。在作业执行前，执行后2个时间点嵌入业务逻辑。

- 执行前：一般用于初始化操作， 作业执行前需要着手准备工作，比如：各种连接建立，线程池初始化等。
- 执行后：业务执行完后，需要做各种清理动作，比如释放资源等。

Spring Batch 使用**JobExecutionListener** 接口 实现作业监听。

```
public interface JobExecutionListener {
    //作业执行前
        void beforeJob(JobExecution jobExecution);
    //作业执行后
        void afterJob(JobExecution jobExecution);
}
```

需求：记录作业执行前，执行中，与执行后的状态

方式一：接口方式

```
//作业状态--接口方式
public class JobStateListener  implements JobExecutionListener {
    //作业执行前
    @Override
    public void beforeJob(JobExecution jobExecution) {
        System.err.println("执行前-status：" + jobExecution.getStatus());
    }
    //作业执行后
    @Override
```

```
    public void afterJob(JobExecution jobExecution) {
        System.err.println("执行后-status：" + jobExecution.getStatus());
    }
}
```

定义JobStateListener 实现JobExecutionListener 接口，重写beforeJob，afterJob 2个方法。

```
import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
@EnableBatchProcessing
public class StatusListenerJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public Tasklet tasklet(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                JobExecution jobExecution = contribution.getStepExecution()
                System.err.println("执行中-status：" + jobExecution.getStatu
                return RepeatStatus.FINISHED;
            }
        };
    }

    //状态监听器
    @Bean
    public JobStateListener jobStateListener(){
        return new JobStateListener();
    }

    @Bean
    public Step  step1(){
        return  stepBuilderFactory.get("step1")
                .tasklet(tasklet())
                .build();
    }
    @Bean
    public Job job(){
        return jobBuilderFactory.get("status-listener-job")
                .start(step1())
```

```
                .listener(jobStateListener())  //设置状态监听器
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(StatusListenerJob.class, args);
    }
}
```

新加**jobStateListener()**实例方法创建对象交个Spring容器管理，修改job()方法，添加**.listener(jobStateListener())** 状态监听器，直接执行，观察结果

方式二：注解方式

除去上面通过实现接口方式实现监听之外，也可以使用**@BeforeJob @AfterJob** 2个注解实现

```
//作业状态--注解方式
public class JobStateAnnoListener  {
    @BeforeJob
    public void beforeJob(JobExecution jobExecution) {
        System.err.println("执行前-anno-status：" + jobExecution.getStatus()
    }

    @AfterJob
    public void afterJob(JobExecution jobExecution) {
        System.err.println("执行后-anno-status：" + jobExecution.getStatus()
    }
}
```

```

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.core.listener.JobListenerFactoryBean;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
import org.springframework.context.annotation.Bean;

@SpringBootApplication
@EnableBatchProcessing
public class StatusListenerJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public Tasklet tasklet(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chur
                JobExecution jobExecution = contribution.getStepExecution()
                System.err.println("执行中-anno-status：" + jobExecution.get
                return RepeatStatus.FINISHED;
            }
        };
    }

    //状态监听器
/*    @Bean
    public JobStateListener jobStateListener(){
        return new JobStateListener();
    }*/

    @Bean
    public Step  step1(){
        return  stepBuilderFactory.get("step1")
                .tasklet(tasklet())
                .build();
    }
    @Bean
    public Job job(){
        return jobBuilderFactory.get("status-listener-job1")
                .start(step1())
                .incrementer(new RunIdIncrementer())
                //.listener(jobStateListener())  //设置状态监听器
                .listener(JobListenerFactoryBean.getListener(new JobStateAr
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(StatusListenerJob.class, args);
    }
}
```

修改job()方法，添加**.listener(JobListenerFactoryBean.getListener(new JobStateAnnoListener()))**状态监听器，直接执行，观察结果

不需要纠结那一长串方法是啥逻辑，只需要知道它能将指定监听器对象加载到spring容器中。

5.5 执行上下文

5.5.1 作业与步骤上下文

语文中有个词叫上下文，比如：联系上下文解读一下作者所有表达意思。从这看上下文有环境，语境，氛围的意思。类比到编程，业内也喜欢使用Context表示上下文。比如Spring容器： SpringApplicationContext 。有上下文这个铺垫之后，我们来看下Spring Batch的上下文。

Spring Batch 有2个比较重要的上下文：

- **JobContext**

JobContext 绑定 JobExecution 执行对象为Job作业执行提供执行环境(上下文)。

作用：维护**JobExecution** 对象，实现作业收尾工作，与处理各种作业回调逻辑

- **StepContext**

StepContext 绑定 StepExecution 执行对象为Step步骤执行提供执行环境(上下文)。

作用：维护**StepExecution** 对象，实现步骤收尾工作，与处理各种步骤回调逻辑

5.5.2 执行上下文

除了上面讲的**JobContext** 作业上下文， **StepContext** 步骤上线下文外，还有Spring Batch还维护另外一个上下文：**ExecutionContext** 执行上下文，作用是：数据共享

Spring Batch 中 ExecutionContext 分2大类

- **Job ExecutionContext**

作用域：一次作业运行，所有Step步骤间数据共享。

- **Step ExecutionContext**：

作用域：一次步骤运行，单个Step步骤间(ItemReader/ItemProcessor/ItemWrite组件间)

数据共享。

5.5.3 作业与步骤执行链

5.5.4 作业与步骤引用链

- 作业线

Job—JobInstance—JobContext—JobExecution–ExecutionContext

- 步骤线

Step–StepContext –StepExecution–ExecutionContext

5.5.5 作业上下文API

```
JobContext context = JobSynchronizationManager.getContext();
JobExecution jobExecution = context.getJobExecution();
Map<String, Object> jobParameters = context.getJobParameters();
Map<String, Object> jobExecutionContext = context.getJobExecutionContext();
```

5.5.6 步骤上下文API

```
ChunkContext chunkContext = xxx;
StepContext stepContext = chunkContext.getStepContext();
StepExecution stepExecution = stepContext.getStepExecution();
Map<String, Object> stepExecutionContext = stepContext.getStepExecutionCont
Map<String, Object> jobExecutionContext = stepContext.getJobExecutionContex
```

5.5.7 执行上下文API

```
ChunkContext chunkContext = xxx;
//步骤
StepContext stepContext = chunkContext.getStepContext();
StepExecution stepExecution = stepContext.getStepExecution();
ExecutionContext executionContext = stepExecution.getExecutionContext();
executionContext.put("key", "value");
//--------------------------------------------------------------------
//作业
JobExecution jobExecution = stepExecution.getJobExecution();
ExecutionContext executionContext = jobExecution.getExecutionContext();
executionContext.put("key", "value");
```

5.5.8 API综合小案例

需求：观察作业**ExecutionContext**与 步骤**ExecutionContext**数据共享

分析：

1>定义step1 与step2 2个步骤

2>在step1中设置数据

作业-ExecutionContext 添加： key-step1-job value-step1-job

步骤-ExecutionContext 添加： key-step1-step value-step1-step

3>在step2中打印观察

作业-ExecutionContext 步骤-ExecutionContext

```
package com.langfeiyes.batch._06_context;

import com.langfeiyes.batch._04_param_incr.DailyTimestampParamIncrementer;
import org.springframework.batch.core.*;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.core.listener.JobListenerFactoryBean;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.scope.context.JobContext;
import org.springframework.batch.core.scope.context.JobSynchronizationManag
import org.springframework.batch.core.scope.context.StepContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.item.ExecutionContext;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
@SpringBootApplication
@EnableBatchProcessing
public class ExecutionContextJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public Tasklet tasklet1(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun

                //步骤
                ExecutionContext stepEC = chunkContext.getStepContext().get
                stepEC.put("key-step1-step","value-step1-step");
                System.out.println("-----------------1-------------------
                //作业
                ExecutionContext jobEC = chunkContext.getStepContext().getS
                jobEC.put("key-step1-job","value-step1-job");
```

```
                return RepeatStatus.FINISHED;
            }
        };
    }
    @Bean
    public Tasklet tasklet2(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun

                //步骤
                ExecutionContext stepEC = chunkContext.getStepContext().get
                System.err.println(stepEC.get("key-step1-step"));
                System.out.println("-----------------2-------------------
                //作业
                ExecutionContext jobEC = chunkContext.getStepContext().getS
                System.err.println(jobEC.get("key-step1-job"));


                return RepeatStatus.FINISHED;
            }
        };
    }

    @Bean
    public Step  step1(){
        return  stepBuilderFactory.get("step1")
                .tasklet(tasklet1())
                .build();
    }
    @Bean
    public Step  step2(){
        return  stepBuilderFactory.get("step2")
                .tasklet(tasklet2())
                .build();
    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("execution-context-job")
                .start(step1())
                .next(step2())
                .incrementer(new RunIdIncrementer())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(ExecutionContextJob.class, args);
    }
}
```

运行结果：

可以看出，在**stepContext** 设置的参数作用域仅在**StepExecution** 执行范围有效，而**JobContext** 设置参数作用与在所有**StepExcution** 有效，有点局部与全局 的意思。

打开数据库观察表：batch_job_execution_context 跟 batch_step_execution_context 表

JobContext数据保存到：**batch_job_execution_context**

StepContext数据保存到：**batch_step_execution_context**

总结：

步骤数据保存在**Step ExecutionContext**，只能在**Step**中使用，作业数据保存在**Job ExecutionContext**，可以在所有**Step**中共享

六、步骤对象 Step

前面一章节讲完了作业的相关介绍，本章节重点讲解步骤。

6.1 步骤介绍

一般认为步骤是一个独立功能组件，因为它包含了一个工作单元需要的所有内容，比如：输入模块，输出模块，数据处理模块等。这种设计好处在哪？给开发者带来更自由的操作空间。

目前Spring Batch 支持2种步骤处理模式：

- 简单具于Tasklet 处理模式

这种模式相对简单，前面讲的都是居于这个模式批处理

```
@Bean
public Tasklet tasklet(){
    return new Tasklet() {
        @Override
        public RepeatStatus execute(StepContribution contribution, ChunkC
            System.out.println("Hello SpringBatch....");
            return RepeatStatus.FINISHED;
        }
    };
}
```

只需要实现Tasklet接口，就可以构建一个step代码块。循环执行step逻辑，直到tasklet.execute方法返回RepeatStatus.FINISHED

- 居于块(chunk)的处理模式

居于块的步骤一般包含2个或者3个组件：1>ItemReader 2>ItemProcessor(可选) 3>ItemWriter 。 当用上这些组件之后，Spring Batch 会按块处理数据。

6.2 简单Tasklet

学到这，我们写过很多简单Tasklet模式步骤，但是都没有深入了解过，这节就细致分析一下具有Tasklet 步骤使用。

先看下Tasklet源码

```
public interface Tasklet {
       @Nullable
       RepeatStatus execute(StepContribution contribution, ChunkContext ch
}
```

Tasklet 接口有且仅有一个方法：execute，

参数有2个：

StepContribution：步骤信息对象，用于保存当前步骤执行情况信息，核心用法：设置步骤结果状态**contribution.setExitStatus(ExitStatus status)**

```
contribution.setExitStatus(ExitStatus.COMPLETED);
```

ChunkContext：chuck上下文，跟之前学的StepContext JobContext一样，区别是它用于记录chunk块执行场景。通过它可以获取前面2个对象。

返回值1个：

RepeatStatus：当前步骤状态， 它是枚举类，有2个值，一个表示execute方法可以循环

执行，一个表示已经执行结束。

```
public enum RepeatStatus {

        /**
         * 当前步骤依然可以执行，如果步骤返回该值，会一直循环执行
         */
        CONTINUABLE(true),
        /**
         * 当前步骤结束，可以为成功也可以表示不成，仅代表当前step执行结束了
         */
        FINISHED(false);
}
```

需求：练习上面**RepeatStatus**状态

```
@SpringBootApplication
@EnableBatchProcessing
public class SimpleTaskletJob {
    @Autowired
    private JobLauncher jobLauncher;
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;
    @Bean
    public Tasklet tasklet(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("------>" + System.currentTimeMillis());
                //return RepeatStatus.CONTINUABLE;  //循环执行
                return RepeatStatus.FINISHED;
            }
        };
    }
    @Bean
    public Step step1(){
        return stepBuilderFactory.get("step1")
                .tasklet(tasklet())
                .build();
    }
    //定义作业
    @Bean
    public Job job(){
        return jobBuilderFactory.get("step-simple-tasklet-job")
                .start(step1())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(SimpleTaskletJob.class, args);
    }

}
```

6.3 居于块**Tasklet**

居于块的Tasklet相对简单Tasklet来说，多了3个模块：ItemReader(读模块)，ItemProcessor(处理模块)，ItemWriter(写模块)， 跟它们名字一样， 一个负责数据读，一个负责数据加工，一个负责数据写。

结构图：

时序图：

需求：简单演示**chunk Tasklet**使用

ItemReader ItemProcessor ItemWriter 都接口，直接使用匿名内部类方式方便创建

```
package com.langfeiyes.batch._08_step_chunk_tasklet;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.item.*;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
```

```
import java.util.Arrays;
import java.util.List;

@SpringBootApplication
@EnableBatchProcessing
public class ChunkTaskletJob {
    @Autowired
    private JobLauncher jobLauncher;
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;


    @Bean
    public ItemReader itemReader(){
        return new ItemReader() {
            @Override
            public Object read() throws Exception, UnexpectedInputExceptior
                System.out.println("-------------read------------");
                return "read-ret";
            }
        };
    }

    @Bean
    public ItemProcessor itemProcessor(){
        return new ItemProcessor() {
            @Override
            public Object process(Object item) throws Exception {
                System.out.println("-------------process------------>" + it
                return "process-ret->" + item;
            }
        };
    }
    @Bean
    public ItemWriter itemWriter(){
        return new ItemWriter() {
            @Override
            public void write(List items) throws Exception {
                System.out.println(items);
            }
        };
    }
    @Bean
    public Step step1(){
        return stepBuilderFactory.get("step1")
                .chunk(3)  //设置块的size为3次
                .reader(itemReader())
                .processor(itemProcessor())
                .writer(itemWriter())
                .build();
    }
    //定义作业
    @Bean
    public Job job(){
        return jobBuilderFactory.get("step-chunk-tasklet-job")
                .start(step1())
                .incrementer(new RunIdIncrementer())
```

```
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(ChunkTaskletJob.class, args);
    }
}
```

执行完了之后结果

```
-------------read------------
-------------read------------
-------------read------------
-------------process------------>read-ret
-------------process------------>read-ret
-------------process------------>read-ret
[process-ret->read-ret, process-ret->read-ret, process-ret->read-ret]
-------------read------------
-------------read------------
-------------read------------
-------------process------------>read-ret
-------------process------------>read-ret
-------------process------------>read-ret
[process-ret->read-ret, process-ret->read-ret, process-ret->read-ret]
-------------read------------
-------------read------------
-------------read------------
-------------process------------>read-ret
-------------process------------>read-ret
-------------process------------>read-ret
[process-ret->read-ret, process-ret->read-ret, process-ret->read-ret]
....
```

观察上面打印结果，得出2个得出。

1>程序一直在循环打印，先循环打印3次reader， 再循环打印3次processor，最后一次性输出3个值。

2>死循环重复上面步骤

问题来了，为啥会出现这种效果，该怎么改进？

其实这个是ChunkTasklet 执行特点，**ItemReader**会一直循环读，直到返回**null**，才停止。而processor也是一样，itemReader读多少次，它处理多少次， itemWriter 一次性输出当前次输入的所有数据。

我们改进一下上面案例，要求只读3次， 只需要改动itemReader方法就行

```
int timer = 3;
@Bean
public ItemReader itemReader(){
    return new ItemReader() {
        @Override
        public Object read() throws Exception, UnexpectedInputException, Pa

            if(timer > 0){
                System.out.println("-------------read------------");
                return  "read-ret-" + timer--;
```

```
            }else{
                return null;
            }

        }
    };
}
```

结果不在死循环了

```
-------------read------------
-------------read------------
-------------read------------
-------------process------------>read-ret-3
-------------process------------>read-ret-2
-------------process------------>read-ret-1
[process-ret->read-ret-3, process-ret->read-ret-2, process-ret->read-ret-1]
```

思考一个问题， 如果将timer改为 10，而 **.chunk(3)** 不变结果会怎样？

```
-------------read------------
-------------read------------
-------------read------------
-------------process------------>read-ret-10
-------------process------------>read-ret-9
-------------process------------>read-ret-8
[process-ret->read-ret-10, process-ret->read-ret-9, process-ret->read-ret-8
-------------read------------
-------------read------------
-------------read------------
-------------process------------>read-ret-7
-------------process------------>read-ret-6
-------------process------------>read-ret-5
[process-ret->read-ret-7, process-ret->read-ret-6, process-ret->read-ret-5]
-------------read------------
-------------read------------
-------------read------------
-------------process------------>read-ret-4
-------------process------------>read-ret-3
-------------process------------>read-ret-2
[process-ret->read-ret-4, process-ret->read-ret-3, process-ret->read-ret-2]
-------------read------------
-------------process------------>read-ret-1
[process-ret->read-ret-1]
```

找出规律了嘛？

当chunkSize = 3 表示 reader 先读3次，提交给processor处理3次，最后由writer输出3个值

timer =10， 表示数据有10条，一个批次(趟)只能处理3条数据，需要4个批次(趟)来处理。

是不是有批处理味道出来

结论：**chunkSize** 表示： 一趟需要**ItemReader**读多少次，**ItemProcessor**要处理多少

次。

ChunkTasklet 泛型

上面案例默认的是使用Object类型读、写、处理数据，如果明确了Item的数据类型，可以明确指定具体操作泛型。

```
import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.item.*;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

import java.util.List;

//开启 spring batch 注解--可以让spring容器创建springbatch操作相关类对象
@EnableBatchProcessing
//springboot 项目，启动注解， 保证当前为为启动类
@SpringBootApplication
public class ChunkTaskletJob {

    //作业启动器
    @Autowired
    private JobLauncher jobLauncher;

    //job构造工厂---用于构建job对象
    @Autowired
    private JobBuilderFactory jobBuilderFactory;

    //step 构造工厂--用于构造step对象
    @Autowired
    private StepBuilderFactory stepBuilderFactory;


    int timer = 10;
    //读操作
    @Bean
    public ItemReader<String> itemReader(){
        return new ItemReader<String>() {
            @Override
            public String read() throws Exception, UnexpectedInputExceptior
                if(timer > 0){
                    System.out.println("-------------read------------");
                    return "read-ret-->" + timer--;
                }else{
                    return null;
                }
            }
```

```
        };
    }
    //处理操作
    @Bean
    public ItemProcessor<String, String> itemProcessor(){
        return new ItemProcessor<String, String>() {
            @Override
            public String process(String item) throws Exception {
                System.out.println("-------------process------------>" + it
                return "process-ret->" + item;
            }
        };
    }

    //写操作
    @Bean
    public ItemWriter<String> itemWriter(){
        return new ItemWriter<String>() {
            @Override
            public void write(List<? extends String> items) throws Exceptio
                System.out.println(items);
            }
        };
    }

    //构造一个step对象--chunk
    @Bean
    public Step step1(){
        //tasklet 执行step逻辑， 类似 Thread()--->可以执行runable接口
        return stepBuilderFactory.get("step1")
                .<String, String>chunk(3)  //暂时为3
                .reader(itemReader())
                .processor(itemProcessor())
                .writer(itemWriter())
                .build();
    }

    @Bean
    public  Job job(){
        return jobBuilderFactory.get("chunk-tasklet-job")
                .start(step1())
                .incrementer(new RunIdIncrementer())
                .build();
    }

    public static void main(String[] args) {
        SpringApplication.run(ChunkTaskletJob.class, args);
    }

}
```

6.4 步骤监听器

前面我们讲了作业的监听器，步骤也有监听器，也是执行步骤执行前监听，步骤执行后监听。

步骤监听器有2个分别是：StepExecutionListener ChunkListener 意义很明显，就是step前后，chunk块执行前后监听。

先看下StepExecutionListener接口

```
public interface StepExecutionListener extends StepListener {
        void beforeStep(StepExecution stepExecution);
        @Nullable
        ExitStatus afterStep(StepExecution stepExecution);
}
```

需求：演示**StepExecutionListener** 用法

自定义监听接口

```
public class MyStepListener implements StepExecutionListener {
    @Override
    public void beforeStep(StepExecution stepExecution) {
        System.out.println("-----------beforeStep--------->");
    }

    @Override
    public ExitStatus afterStep(StepExecution stepExecution) {
        System.out.println("-----------afterStep--------->");
        return stepExecution.getExitStatus();  //不改动返回状态
    }
}
```

```
package com.langfeiyes.batch._09_step_listener;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
@EnableBatchProcessing
public class StepListenerJob {
    @Autowired
    private JobLauncher jobLauncher;
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;
    @Bean
    public Tasklet tasklet(){
        return new Tasklet() {
            @Override
```

```
            public RepeatStatus execute(StepContribution contribution, Chur
                System.out.println("------>" + System.currentTimeMillis());
                return RepeatStatus.FINISHED;
            }
        };
    }


    @Bean
    public MyStepListener stepListener(){
        return new MyStepListener();
    }
    @Bean
    public Step step1(){
        return stepBuilderFactory.get("step1")
                .tasklet(tasklet())
                .listener(stepListener())
                .build();
    }
    //定义作业
    @Bean
    public Job job(){
        return jobBuilderFactory.get("step-listener-job1")
                .start(step1())
                .incrementer(new RunIdIncrementer())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(StepListenerJob.class, args);
    }

}
```

在step1方法中，加入：**.listener(stepListener())** 即可

同理ChunkListener 操作跟上面一样

```
public interface ChunkListener extends StepListener {
        static final String ROLLBACK_EXCEPTION_KEY = "sb_rollback_exceptior
        void beforeChunk(ChunkContext context);
        void afterChunk(ChunkContext context);
        void afterChunkError(ChunkContext context);
}
```

唯一的区别是多了一个afterChunkError 方法，表示当chunk执行失败后回调。

6.5 多步骤执行

到目前为止，我们演示的案例基本上都是一个作业， 一个步骤，那如果有多个步骤会怎样？Spring Batch 支持多步骤执行，以应对复杂业务需要多步骤配合执行的场景。

需求：定义**2**个步骤，然后依次执行

```
package com.langfeiyes.batch._10_step_multi;

import com.langfeiyes.batch._09_step_listener.MyChunkListener;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
@EnableBatchProcessing
public class MultiStepJob {
    @Autowired
    private JobLauncher jobLauncher;
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;
    @Bean
    public Tasklet tasklet1(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("--------------tasklet1---------------")
                return RepeatStatus.FINISHED;
            }
        };
    }
    @Bean
    public Tasklet tasklet2(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("--------------tasklet2---------------")
                return RepeatStatus.FINISHED;
            }
        };
    }

    @Bean
    public Step step1(){
        return stepBuilderFactory.get("step1")
                .tasklet(tasklet1())
                .build();
    }

    @Bean
    public Step step2(){
        return stepBuilderFactory.get("step2")
                .tasklet(tasklet2())
```

```
                .build();
    }

    //定义作业
    @Bean
    public Job job(){
        return jobBuilderFactory.get("step-multi-job1")
                .start(step1())
                .next(step2()) //job 使用next 执行下一步骤
                .incrementer(new RunIdIncrementer())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(MultiStepJob.class, args);
    }
}
```

定义2个tasklet： tasklet1 tasklet2， 定义2个step： step1 step2 修改 job方法，从**.start(step1())** 然后执行到 **.next(step2())**

Spring Batch 使用next 执行下一步步骤，如果还有第三个step，再加一个next(step3)即可

6.6 步骤控制

上面多个步骤操作，先执行step1 然后是step2，如果有step3， step4，那执行顺序也是从step1到step4。此时爱思考的小伙伴肯定会想，步骤的执行能不能进行条件控制呢？比如：step1执行结束根据业务条件选择执行step2或者执行step3，亦或者直接结束呢？答案是**yes**：设置步骤执行条件即可

Spring Batch 使用 **start next on from to end** 不同的api 改变步骤执行顺序。

6.6.1 条件分支控制-使用默认返回状态

需求：作业执行**firstStep**步骤，如果处理成功执行**sucessStep**，如果处理失败执行**failStep**

```
package com.langfeiyes.batch._11_step_condition;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
```

```
@SpringBootApplication
@EnableBatchProcessing
public class ConditionStepJob {
    @Autowired
    private JobLauncher jobLauncher;
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;
    @Bean
    public Tasklet firstTasklet(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("--------------firstTasklet------------
                return RepeatStatus.FINISHED;
                //throw new RuntimeException("测试fail结果");
            }
        };
    }
    @Bean
    public Tasklet successTasklet(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("--------------successTasklet----------
                return RepeatStatus.FINISHED;
            }
        };
    }
    @Bean
    public Tasklet failTasklet(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("--------------failTasklet--------------
                return RepeatStatus.FINISHED;
            }
        };
    }
    @Bean
    public Step firstStep(){
        return stepBuilderFactory.get("step1")
                .tasklet(firstTasklet())
                .build();
    }
    @Bean
    public Step successStep(){
        return stepBuilderFactory.get("successStep")
                .tasklet(successTasklet())
                .build();
    }
    @Bean
    public Step failStep(){
        return stepBuilderFactory.get("failStep")
                .tasklet(failTasklet())
                .build();
    }
```

```
    //定义作业
    @Bean
    public Job job(){
        return jobBuilderFactory.get("condition-multi-job")
                .start(firstStep())
                .on("FAILED").to(failStep())
                .from(firstStep()).on("*").to(successStep())
                .end()
                .incrementer(new RunIdIncrementer())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(ConditionStepJob.class, args);
    }
}
```

观察给出的案例，job方法以 **.start(firstStep())** 开始作业，执行完成之后， 使用**on** 与**from** 2个方法实现流程转向。

.on(“FAILED”).to(failStep()) 表示当**firstStep()**返回FAILED时执行。

.from(firstStep()).on(“*”).to(successStep()) 另外一个分支，表示当**firstStep()**返回 * 时执行。

上面逻辑有点像 if / else 语法

```
if("FAILED".equals(firstStep())){
    failStep();
}else{
    successStep();
}
```

几个注意点：

1> on 方法表示条件， 上一个步骤返回值，匹配指定的字符串，满足后执行后续 to 步骤

2> * 为通配符，表示能匹配任意返回值

3> from 表示从某个步骤开始进行条件判断

4> 分支判断结束，流程以end方法结束，表示if/else逻辑结束

5> on 方法中字符串取值于 ExitStatus 类常量，当然也可以自定义。

6.6.2 条件分支控制-使用自定义状态值

前面也说了，on条件的值取值于ExitStatus 类常量，具体值有：UNKNOWN，EXECUTING，COMPLETED，NOOP，FAILED，STOPPED等，如果此时我想自定义返回值呢，是否可行？答案还是yes：Spring Batch 提供JobExecutionDecider 接口实现状态值定制。

需求：先执行**firstStep**，如果返回值为**A**，执行**stepA**， 返回值为**B**，执行**stepB**， 其他执行**defaultStep**

分析：先定义一个决策器，随机决定返回A / B / C

```
public class MyStatusDecider implements JobExecutionDecider {
    @Override
    public FlowExecutionStatus decide(JobExecution jobExecution, StepExecut
        long ret = new Random().nextInt(3);
        if(ret == 0){
            return new FlowExecutionStatus("A");
        }else if(ret == 1){
            return new FlowExecutionStatus("B");
        }else{
            return new FlowExecutionStatus("C");
        }
    }
}
```

```
package com.langfeiyes.batch._11_step_condition_decider;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
@EnableBatchProcessing
public class CustomizeStatusStepJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;
    @Bean
    public Tasklet taskletFirst(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("--------------taskletFirst-------------
                return RepeatStatus.FINISHED;
            }
        };
    }
    @Bean
    public Tasklet taskletA(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("--------------taskletA---------------")
                return RepeatStatus.FINISHED;
            }
```

```
        };
    }
    @Bean
    public Tasklet taskletB(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("-------------taskletB--------------")
                return RepeatStatus.FINISHED;
            }
        };
    }
    @Bean
    public Tasklet taskletDefault(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("-------------taskletDefault----------
                return RepeatStatus.FINISHED;
            }
        };
    }


    @Bean
    public Step firstStep(){
        return stepBuilderFactory.get("firstStep")
                .tasklet(taskletFirst())
                .build();
    }

    @Bean
    public Step stepA(){
        return stepBuilderFactory.get("stepA")
                .tasklet(taskletA())
                .build();
    }

    @Bean
    public Step stepB(){
        return stepBuilderFactory.get("stepB")
                .tasklet(taskletB())
                .build();
    }

    @Bean
    public Step defaultStep(){
        return stepBuilderFactory.get("defaultStep")
                .tasklet(taskletDefault())
                .build();
    }


    //决策器
    @Bean
    public MyStatusDecider statusDecider(){
        return new MyStatusDecider();
    }
```

```
    //定义作业
    @Bean
    public Job job(){
        return jobBuilderFactory.get("customize-step-job")
                .start(firstStep())
                .next(statusDecider())
                .from(statusDecider()).on("A").to(stepA())
                .from(statusDecider()).on("B").to(stepB())
                .from(statusDecider()).on("*").to(defaultStep())
                .end()
                .incrementer(new RunIdIncrementer())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(CustomizeStepJob.class, args);
    }

}
```

反复执行，会返回打印的值有

```
--------------taskletA---------------
--------------taskletB---------------
--------------taskletDefault---------------
```

它们随机切换，为啥能做到这样？注意，并不是**firstStep()** 执行返回值为A/B/C控制流程跳转，而是由后面**.next(statusDecider())** 决策器。

6.7 步骤状态

Spring Batch 使用ExitStatus 类表示步骤、块、作业执行状态，大体上有以下几种：

```
public class ExitStatus implements Serializable, Comparable<ExitStatus> {

    //未知状态
        public static final ExitStatus UNKNOWN = new ExitStatus("UNKNOWN");

    //执行中
        public static final ExitStatus EXECUTING = new ExitStatus("EXECUTIN

    //执行完成
        public static final ExitStatus COMPLETED = new ExitStatus("COMPLETE

    //无效执行
        public static final ExitStatus NOOP = new ExitStatus("NOOP");

    //执行失败
        public static final ExitStatus FAILED = new ExitStatus("FAILED");

    //执行中断
        public static final ExitStatus STOPPED = new ExitStatus("STOPPED");
    ...
}
```

一般来说，作业启动之后，这些状态皆为流程自行控制。顺利结束返

回：**COMPLETED**， 异常结束返回：**FAILED**，无效执行返回：**NOOP**， 这是肯定有小伙伴说，能不能编程控制呢？答案是可以的。

Spring Batch 提供 3个方法决定作业流程走向：

end()：作业流程直接成功结束，返回状态为：**COMPLETED**

fail()：作业流程直接失败结束，返回状态为：**FAILED**

stopAndRestart(step) ：作业流程中断结束，返回状态：**STOPPED** 再次启动时，从step位置开始执行 (注意：前提是参数与Job Name一样)

**需求：当步骤firstStep执行抛出异常时，通过end， fail，stopAndRestart改变步骤执行状态 **

```
package com.langfeiyes.batch._12_step_status;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

//开启 spring batch 注解--可以让spring容器创建springbatch操作相关类对象
@EnableBatchProcessing
//springboot 项目，启动注解， 保证当前为为启动类
@SpringBootApplication
public class StatusStepJob {

    //作业启动器
    @Autowired
    private JobLauncher jobLauncher;

    //job构造工厂---用于构建job对象
    @Autowired
    private JobBuilderFactory jobBuilderFactory;

    //step 构造工厂--用于构造step对象
    @Autowired
    private StepBuilderFactory stepBuilderFactory;


    //构造一个step对象执行的任务（逻辑对象）
    @Bean
    public Tasklet firstTasklet(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chur
```

```
            System.out.println("----------------firstTasklet----------

            throw new RuntimeException("假装失败了");
            //return RepeatStatus.FINISHED;  //执行完了
        }
    };
}

@Bean
public Tasklet successTasklet(){
    return new Tasklet() {
        @Override
        public RepeatStatus execute(StepContribution contribution, Chun

            System.out.println("----------------successTasklet---------

            return RepeatStatus.FINISHED;  //执行完了
        }
    };
}
@Bean
public Tasklet failTasklet(){
    return new Tasklet() {
        @Override
        public RepeatStatus execute(StepContribution contribution, Chun

            System.out.println("----------------failTasklet------------

            return RepeatStatus.FINISHED;  //执行完了
        }
    };
}



//构造一个step对象
@Bean
public Step firstStep(){
    //tasklet 执行step逻辑， 类似 Thread()--->可以执行runable接口
    return stepBuilderFactory.get("firstStep")
            .tasklet(firstTasklet())
            .build();
}
//构造一个step对象
@Bean
public Step successStep(){
    //tasklet 执行step逻辑， 类似 Thread()--->可以执行runable接口
    return stepBuilderFactory.get("successStep")
            .tasklet(successTasklet())
            .build();
}

//构造一个step对象
@Bean
public Step failStep(){
    //tasklet 执行step逻辑， 类似 Thread()--->可以执行runable接口
    return stepBuilderFactory.get("failStep")
            .tasklet(failTasklet())
            .build();
```

```
    }

    //如果firstStep 执行成功：下一步执行successStep 否则是failStep
    @Bean
    public  Job job(){
        return jobBuilderFactory.get("status-step-job")
                .start(firstStep())
                //表示将当前本应该是失败结束的步骤直接转成正常结束--COMPLETED
                //.on("FAILED").end()
                //表示将当前本应该是失败结束的步骤直接转成失败结束：FAILED
                //.on("FAILED").fail()
                //表示将当前本应该是失败结束的步骤直接转成停止结束：STOPPED    里面参
                .on("FAILED").stopAndRestart(successStep())
                .from(firstStep()).on("*").to(successStep())
                .end()
                .incrementer(new RunIdIncrementer())
                .build();
    }

    public static void main(String[] args) {
        SpringApplication.run(StatusStepJob.class, args);
    }

}
```

6.8 流式步骤

FlowStep 流式步骤，也可以理解为步骤集合，由多个子步骤组成。作业执行时，将它当做一个普通步骤执行。一般用于较为复杂的业务，比如：一个业务逻辑需要拆分成按顺序执行的子步骤。

需求：先后执行**stepA**，**stepB**，**stepC**， 其中**stepB**中包含**stepB1**， **stepB2**，**stepB3**。

```
package com.langfeiyes.batch._13_flow_step;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.job.builder.FlowBuilder;
import org.springframework.batch.core.job.builder.JobBuilder;
import org.springframework.batch.core.job.builder.SimpleJobBuilder;
import org.springframework.batch.core.job.flow.Flow;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
```

```
@SpringBootApplication
@EnableBatchProcessing
public class FlowStepJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public Tasklet taskletA(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("------------stepA--taskletA-----------
                return RepeatStatus.FINISHED;
            }
        };
    }
    @Bean
    public Tasklet taskletB1(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("------------stepB--taskletB1----------
                return RepeatStatus.FINISHED;
            }
        };
    }
    @Bean
    public Tasklet taskletB2(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("------------stepB--taskletB2----------
                return RepeatStatus.FINISHED;
            }
        };
    }
    @Bean
    public Tasklet taskletB3(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("------------stepB--taskletB3----------
                return RepeatStatus.FINISHED;
            }
        };
    }
    @Bean
    public Tasklet taskletC(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("------------stepC--taskletC-----------
                return RepeatStatus.FINISHED;
            }
        };
    }
```

```
    @Bean
    public Step stepA(){
        return stepBuilderFactory.get("stepA")
                .tasklet(taskletA())
                .build();
    }


    @Bean
    public Step stepB1(){
        return stepBuilderFactory.get("stepB1")
                .tasklet(taskletB1())
                .build();
    }
    @Bean
    public Step stepB2(){
        return stepBuilderFactory.get("stepB2")
                .tasklet(taskletB2())
                .build();
    }
    @Bean
    public Step stepB3(){
        return stepBuilderFactory.get("stepB3")
                .tasklet(taskletB3())
                .build();
    }
    @Bean
    public Flow flowB(){
        return new FlowBuilder<Flow>("flowB")
                .start(stepB1())
                .next(stepB2())
                .next(stepB3())
                .build();
    }
    @Bean
    public Step stepB(){
        return stepBuilderFactory.get("stepB")
                .flow(flowB())
                .build();
    }

    @Bean
    public Step stepC(){
        return stepBuilderFactory.get("stepC")
                .tasklet(taskletC())
                .build();
    }

    //定义作业
    @Bean
    public Job job(){
        return jobBuilderFactory.get("flow-step-job")
                .start(stepA())
                .next(stepB())
                .next(stepC())
                .incrementer(new RunIdIncrementer())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(FlowStepJob.class, args);
```

```
    }

}
```

此时的flowB()就是一个FlowStep，包含了stepB1, stepB2, stepB3 3个子step，他们全部执行完后， stepB才能算执行完成。下面执行结果也验证了这点。

```
2022-12-03 14:54:16.644  INFO 19116 --- [           main] o.s.batch.core.jo
------------stepA--taskletA---------------
2022-12-03 14:54:16.699  INFO 19116 --- [           main] o.s.batch.core.st
2022-12-03 14:54:16.738  INFO 19116 --- [           main] o.s.batch.core.jo
2022-12-03 14:54:16.788  INFO 19116 --- [           main] o.s.batch.core.jo
------------stepB--taskletB1---------------
2022-12-03 14:54:16.844  INFO 19116 --- [           main] o.s.batch.core.st
2022-12-03 14:54:16.922  INFO 19116 --- [           main] o.s.batch.core.jo
------------stepB--taskletB2---------------
2022-12-03 14:54:16.952  INFO 19116 --- [           main] o.s.batch.core.st
2022-12-03 14:54:16.996  INFO 19116 --- [           main] o.s.batch.core.jo
------------stepB--taskletB3---------------
2022-12-03 14:54:17.032  INFO 19116 --- [           main] o.s.batch.core.st
2022-12-03 14:54:17.057  INFO 19116 --- [           main] o.s.batch.core.st
2022-12-03 14:54:17.165  INFO 19116 --- [           main] o.s.batch.core.jo
------------stepC--taskletC---------------
2022-12-03 14:54:17.215  INFO 19116 --- [           main] o.s.batch.core.st
```

使用FlowStep的好处在于，在处理复杂额批处理逻辑中，flowStep可以单独实现一个子步骤流程，为批处理提供更高的灵活性。

七、批处理数据表

如果选择数据库方式存储批处理数据，Spring Batch 在启动时会自动创建9张表，分别存储： JobExecution、JobContext、JobParameters、JobInstance、JobExecution id序列、Job id序列、StepExecution、StepContext/ChunkContext、StepExecution id序列 等对象。Spring Batch 提供 JobRepository 组件来实现这些表的CRUD操作，并且这些操作基本上封装在步骤，块，作业api操作中，并不需要我们太多干预，所以这章内容了解即可。

7.1 batch_job_instance表

当作业第一次执行时，会根据作业名，标识参数生成一个唯一JobInstance对象，batch_job_instance表会记录一条信息代表这个作业实例。

字段	描述

JOB_INSTANCE_ID	作业实例主键
VERSION	乐观锁控制的版本号
JOB_NAME	作业名称
JOB_KEY	作业名与标识性参数的哈希值，能唯一标识一个job实例

7.2 batch_job_execution表

每次启动作业时，都会创建一个JobExecution对象，代表一次作业执行，该对象记录存放于batch_job_execution 表。

字段	描述
JOB_EXECUTION_ID	job执行对象主键
VERSION	乐观锁控制的版本号
JOB_INSTANCE_ID	JobInstanceId(归属于哪个JobInstance)
CREATE_TIME	记录创建时间
START_TIME	作业执行开始时间
END_TIME	作业执行结束时间
STATUS	作业执行的批处理状态
EXIT_CODE	作业执行的退出码
EXIT_MESSAGE	作业执行的退出信息
LAST_UPDATED	最后一次更新记录的时间

7.3 batch_job_execution_context表

batch_job_execution_context用于保存JobContext对应的ExecutionContext对象数据。

字段	描述
JOB_EXECUTION_ID	job执行对象主键
SHORT_CONTEXT	ExecutionContext系列化后字符串缩减版
SERIALIZED_CONTEXT	ExecutionContext系列化后字符串

7.4 batch_job_execution_params表

作业启动时使用标识性参数保存的位置：batch_job_execution_params， 一个参数一个记录

字段	描述
JOB_EXECUTION_ID	job执行对象主键
TYPE_CODE	标记参数类型
KEY_NAME	参数名
STRING_VALUE	当参数类型为String时有值
DATE_VALUE	当参数类型为Date时有值
LONG_VAL	当参数类型为LONG时有值
DOUBLE_VAL	当参数类型为DOUBLE时有值
IDENTIFYING	用于标记该参数是否为标识性参数

7.5 btch_step_execution表

作业启动，执行步骤，每个步骤执行信息保存在tch_step_execution表中

字段	描述
STEP_EXECUTION_ID	步骤执行对象id
VERSION	乐观锁控制版本号

STEP_NAME	步骤名称
JOB_EXECUTION_ID	作业执行对象id
START_TIME	步骤执行的开始时间
END_TIME	步骤执行的结束时间
STATUS	步骤批处理状态
COMMIT_COUNT	在步骤执行中提交的事务次数
READ_COUNT	读入的条目数量
FILTER_COUNT	由于ItemProcessor返回null而过滤掉的条目数
WRITE_COUNT	写入条目数量
READ_SKIP_COUNT	由于ItemReader中抛出异常而跳过的条目数量
PROCESS_SKIP_COUNT	由于ItemProcessor中抛出异常而跳过的条目数量
WRITE_SKIP_COUNT	由于ItemWriter中抛出异常而跳过的条目数量
ROLLBACK_COUNT	在步骤执行中被回滚的事务数量
EXIT_CODE	步骤的退出码
EXT_MESSAGE	步骤执行返回的信息
LAST_UPDATE	最后一次更新记录时间

7.6 batch_step_execution_context表

StepContext对象对应的ExecutionContext 保存的数据表：batch_step_execution_context

字段	描述
STEP_EXECUTION_ID	步骤执行对象id
SHORT_CONTEXT	ExecutionContext系列化后字符串缩减版
SERIALIZED_CONTEXT	ExecutionContext系列化后字符串

7.7 H2内存数据库

除了关系型数据库保存的数据外，Spring Batch 也执行内存数据库，比如H2，HSQLDB，这些数据库将数据缓存在内存中，当批处理结束后，数据会被清除，一般用于进行单元测试，不建议在生产环境中使用。

八、作业控制

作业的运行指的是对作业的控制，包括作业启动，作业停止，作业异常处理，作业重启处理等。

8.1 作业启动

8.1.1 SpringBoot 启动

目前为止，上面所有的案例都是使用Spring Boot 原生功能来启动作业的，其核心类：**JobLauncherApplicationRunner**， Spring Boot启动之后，马上调用该类run方法，然后将操作委托给SimpleJobLauncher类run方法执行。默认情况下，Spring Boot一启动马上执行作业。

如果不想Spring Boot启动就执行，可以通过配置进行修改

```
spring:
  batch:
    job:
      enabled: false    #false表示不启动
```

8.1.2 Spring 单元测试启动

开发中如果想简单验证批处理逻辑是否能运行，可以使用单元测试方式启动作业

先引入spring-test测试依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
</dependency>
```

建立启动类

```
@SpringBootApplication
@EnableBatchProcessing
public class App {
    public static void main(String[] args) {
        SpringApplication.run(App.class, args);
    }
}
```

建立测试类

```
package com.langfeiyes.batch._14_job_start_test;

import org.junit.jupiter.api.Test;
import org.springframework.batch.core.*;
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.core.step.tasklet.TaskletStep;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest(classes = App.class)
public class StartJobTest {
    //job调度器
    @Autowired
    private JobLauncher jobLauncher;
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;
    public Tasklet tasklet(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.out.println("Hello SpringBatch....");
                return RepeatStatus.FINISHED;
            }
        };
    }
    public Step  step1(){
        TaskletStep step1 = stepBuilderFactory.get("step1")
                .tasklet(tasklet())
                .build();
        return step1;
    }
    //定义作业
    public Job job(){
        Job job = jobBuilderFactory.get("start-test-job")
                .start(step1())
                .build();
        return job;
    }

    @Test
    public void testStart() throws Exception{
        //job作业启动
        //参数1：作业实例，参数2：作业运行携带参数
        jobLauncher.run(job(), new JobParameters());
    }
}
```

跟之前的SpringBoot启动区别在于多了JobLauncher 对象的获取，再由这个对象调用run方法启动。

8.1.3 RESTful API 启动

如果批处理不是SpringBoot启动就启动，而是通过web请求控制，那该怎么办呢？不难，引入web环境即可

1>首先限制，不随SpringBoot启动而启动

```
spring:
  batch:
    job:
      enabled: false   #false表示不启动
```

2>引入web 环境

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

3>编写启动类

```
@SpringBootApplication
public class App {
    public static void main(String[] args) {
        SpringApplication.run(App.class, args);
    }
}
```

4>编写配置类

```
package com.langfeiyes.batch._15_job_start_restful;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.core.step.tasklet.TaskletStep;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@EnableBatchProcessing
@Configuration
public class BatchConfig {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public Tasklet tasklet(){
        return new Tasklet() {
            @Override
```

```
            public RepeatStatus execute(StepContribution contribution, Chur
                System.out.println("Hello SpringBatch....");
                return RepeatStatus.FINISHED;
            }
        };
    }

    @Bean
    public Step step1(){
        TaskletStep step1 = stepBuilderFactory.get("step1")
                .tasklet(tasklet())
                .build();
        return step1;
    }
    //定义作业
    @Bean
    public Job job(){
        Job job = jobBuilderFactory.get("hello-restful-job")
                .start(step1())
                .build();
        return job;
    }
}
```

5>编写Controller类

```
package com.langfeiyes.batch._15_job_start_restful;

import org.springframework.batch.core.*;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.batch.core.repository.JobExecutionAlreadyRunning
import org.springframework.batch.core.repository.JobInstanceAlreadyComplete
import org.springframework.batch.core.repository.JobRestartException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.Map;
import java.util.Properties;

@RestController
public class HelloController {
    @Autowired
    private JobLauncher jobLauncher;
    @Autowired
    private Job job;
    @GetMapping("/job/start")
    public ExitStatus start() throws Exception {
         //启动job作业
        JobExecution jobExet = launcher.run(job, jp);
        return jobExet.getExitStatus();
    }
}
```

6>测试

注意：如果需要接收参数

1>作业使用run.id自增

```
//构造一个job对象
@Bean
public Job job(){
    return jobBuilderFactory.get("hello-restful-job")
        .start(step1())
        .incrementer(new RunIdIncrementer())
        .build();
}
```

2>改动HelloController接口方法

```
@RestController
public class HelloController {
    @Autowired
    private JobLauncher launcher;
    @Autowired
    private Job job;
    @Autowired
    private JobExplorer jobExplorer;  //job 展示对象
    @GetMapping("/job/start")
    public ExitStatus startJob(String name) throws Exception {
        //启动job作业
        JobParameters jp = new JobParametersBuilder(jobExplorer)
                .getNextJobParameters(job)
                .addString("name", name)
                .toJobParameters();
        JobExecution jobExet = launcher.run(job, jp);
        return jobExet.getExitStatus();
    }
}
```

8.2 作业停止

作业的停止，存在有3种情况：

- 一种自然结束

作业成功执行，正常停止，此时作业返回状态为：**COMPLETED**

- 一种异常结束 作业执行过程因为各种意外导致作业中断而停止，大多数作业返回状态为：**FAILED**
- 一种编程结束

某个步骤处理数据结果不满足下一步骤执行前提，手动让其停止，一般设置返回状态为：**STOPED**

上面1,2种情况相对简单，我们重点说下第三种：以编程方式让作业停止。

模拟一个操作场景

1>有一个资源类，里面有2个属性：总数：totalCount = 100， 读取数：readCount = 0

2>设计2个步骤，step1 用于叠加readCount 模拟从数据库中读取资源， step2 用于执行逻辑

3>当totalCount == readCount 时，为正常情况，正常结束。如果不等时，为异常状态。此时不执行step2，直接停止作业。

4>修复数据，在从step1开始执行，并完成作业

```
public class ResouceCount {
    public static int totalCount = 100;  //总数
    public static int  readCount = 0;    //读取数
}
```

要实现上面需求，有2种方式可以实现

方案**1**：**Step** 步骤监听器方式

监听器

```
public class StopStepListener implements StepExecutionListener {
    @Override
    public void beforeStep(StepExecution stepExecution) {
    }

    @Override
    public ExitStatus afterStep(StepExecution stepExecution) {

       //不满足
        if(ResouceCount.totalCount != ResouceCount.readCount){
            return ExitStatus.STOPPED;  //手动停止，后续可以重启
        }
        return stepExecution.getExitStatus();
    }
}
```

代码

```
package com.langfeiyes.batch._16_job_stop;

import com.langfeiyes.batch._01_hello.HelloJob;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
```

```
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
@EnableBatchProcessing
public class ListenerJobStopJob {

    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    private int readCount = 50; //模拟只读取50个
    @Bean
    public Tasklet tasklet1(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                for (int i = 1; i <= readCount; i++) {
                    System.out.println("---------------step1执行-"+i+"-----
                    ResouceCount.readCount ++;
                }
                return RepeatStatus.FINISHED;
            }
        };
    }

    @Bean
    public Tasklet tasklet2(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.err.println("step2执行了.......");
                System.err.println("readCount:" + ResouceCount.readCount +
                return RepeatStatus.FINISHED;
            }
        };
    }

    @Bean
    public StopStepListener stopStepListener(){
        return new StopStepListener();
    }
    @Bean
    public Step step1(){
        return stepBuilderFactory.get("step1")
                .tasklet(tasklet1())
                .listener(stopStepListener())
                .allowStartIfComplete(true)  //执行完后，运行重启
                .build();
    }

    @Bean
    public Step step2(){
        return stepBuilderFactory.get("step2")
                .tasklet(tasklet2())
                .build();
```

```
    }

    //定义作业
    @Bean
    public Job job(){
        return jobBuilderFactory.get("job-stop-job")
                .start(step1())
                .on("STOPPED").stopAndRestart(step1())
                .from(step1()).on("*").to(step2()).end()
                .build();

    }
    public static void main(String[] args) {
        SpringApplication.run(ListenerJobStopJob.class, args);
    }
}
```

第一次执行：tasklet1 中readCount 默认执行50次，不满足条件， stopStepListener() afterStep 返回STOPPED, job进行条件控制
走**.on(“STOPPED”).stopAndRestart(step1())** 分支，停止并允许重启–下次重启，从step1步骤开始执行

第二次执行， 修改readCount = 100， 再次启动作业，task1遍历100次，满足条件，stopStepListener() afterStep 正常返回，job条件控制
走**.from(step1()).on(“*”).to(step2()).end()**分支，正常结束。

注意：step1() 方法中**.allowStartIfComplete(true)** 代码必须添加，因为第一次执行step1步骤，虽然不满足条件，但是它仍属于正常结束(正常执行完tasklet1的流程)，状态码：COMPLETED， 第二次重启，默认情况下正常结束的step1步骤是不允许再执行的，所以必须设置：**.allowStartIfComplete(true)** 允许step1即使完成也可以重启。

方案**2**：**StepExecution**停止标记

```
package com.langfeiyes.batch._17_job_stop_sign;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
@EnableBatchProcessing
public class SignJobStopJob {

    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
```

```
    private StepBuilderFactory stepBuilderFactory;

    private int readCount = 50; //模拟只读取50个
    @Bean
    public Tasklet tasklet1(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                for (int i = 1; i <= readCount; i++) {
                    System.out.println("---------------step1执行-"+i+"-----
                    ResouceCount.readCount ++;
                }

                if(ResouceCount.readCount != ResouceCount.totalCount){
                    chunkContext.getStepContext().getStepExecution().setTer
                }

                return RepeatStatus.FINISHED;
            }
        };
    }

    @Bean
    public Tasklet tasklet2(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.err.println("step2执行了.......");
                System.err.println("readCount:" + ResouceCount.readCount +
                return RepeatStatus.FINISHED;
            }
        };
    }


    @Bean
    public Step step1(){
        return stepBuilderFactory.get("step1")
                .tasklet(tasklet1())
                .allowStartIfComplete(true)
                .build();
    }

    @Bean
    public Step step2(){
        return stepBuilderFactory.get("step2")
                .tasklet(tasklet2())
                .build();
    }

    //定义作业
    @Bean
    public Job job(){
         return jobBuilderFactory.get("job-stop-job")
                .start(step1())
                //.on("STOPPED").stopAndRestart(step1())
                //.from(step1()).on("*").to(step2()).end()
                .next(step2())
                .build();
```

```
    }
    public static void main(String[] args) {
        SpringApplication.run(SignJobStopJob.class, args);
    }
}
```

变动的代码有2处

tasket1(), 多了下面判断

```
if(ResouceCount.readCount != ResouceCount.totalCount){
    chunkContext.getStepContext().getStepExecution().setTerminateOnly();
}
```

其中的StepExecution#setTerminateOnly() 给运行中的stepExecution设置停止标记，Spring Batch 识别后直接停止步骤，进而停止流程

job() 改动

```
return jobBuilderFactory.get("job-stop-job")
    .start(step1())
    .next(step2())
    .build();
```

正常设置步骤流程。

8.3 作业重启

作业重启，表示允许作业步骤重新执行，默认情况下，只允许异常或终止状态的步骤重启，但有时存在特殊场景，要求需要其他状态步骤重启，为应付各种复杂的情形，Spring Batch 提供3种重启控制操作。

8.3.1 禁止重启

这种适用一次性执行场景，如果执行失败，就不允许再次执行。可以使用作业的禁止重启逻辑

```
package com.langfeiyes.batch._18_job_restart_forbid;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
```

```
@SpringBootApplication
@EnableBatchProcessing
public class JobForBidRestartJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public Tasklet tasklet1(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.err.println("-------------tasklet1-------------");

                chunkContext.getStepContext().getStepExecution().setTermina
                return RepeatStatus.FINISHED;

            }
        };
    }

    @Bean
    public Tasklet tasklet2(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.err.println("-------------tasklet2-------------");
                return RepeatStatus.FINISHED;
            }
        };
    }


    @Bean
    public Step step1(){
        return stepBuilderFactory.get("step1")
                .tasklet(tasklet1())
                .build();
    }

    @Bean
    public Step step2(){
        return stepBuilderFactory.get("step2")
                .tasklet(tasklet2())
                .build();
    }

    //定义作业
    @Bean
    public Job job(){
        return jobBuilderFactory.get("job-forbid-restart-job")
                .preventRestart()  //禁止重启
                .start(step1())
                .next(step2())
                .build();

    }
    public static void main(String[] args) {
        SpringApplication.run(JobForBidRestartJob.class, args);
```

```
    }
}
```

观察上面代码，比较特别之处：

tasklet1() 加了setTerminateOnly 设置，表示让步骤退出

```
chunkContext.getStepContext().getStepExecution().setTerminateOnly();
```

job() 多了**.preventRestart()** 逻辑，表示步骤不允许重启

第一次按上面的代码执行一次， step1() 状态为 **STOPPED**

第二次去掉setTerminateOnly逻辑，重新启动步骤，观察结果，直接报错

8.3.2 限制重启次数

适用于重启次数有限的场景，比如下载/读取操作，可能因为网络原因导致下载/读取失败，运行重试几次，但是不能无限重试。这时可以对步骤执行进行重启次数限制。

```
package com.langfeiyes.batch._19_job_restart_limit;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
@EnableBatchProcessing
public class JobLimitRestartJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public Tasklet tasklet1(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.err.println("-------------tasklet1-------------");

                chunkContext.getStepContext().getStepExecution().setTermina
                return RepeatStatus.FINISHED;
```

```
            }
        };
    }

    @Bean
    public Tasklet tasklet2(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.err.println("-------------tasklet2-------------");
                return RepeatStatus.FINISHED;
            }
        };
    }


    @Bean
    public Step step1(){
        return stepBuilderFactory.get("step1")
                .startLimit(2)
                .tasklet(tasklet1())
                .build();
    }

    @Bean
    public Step step2(){
        return stepBuilderFactory.get("step2")
                .tasklet(tasklet2())
                .build();
    }

    //定义作业
    @Bean
    public Job job(){
        return jobBuilderFactory.get("job-restart-limit-job")
                .start(step1())
                .next(step2())
                .build();

    }
    public static void main(String[] args) {
        SpringApplication.run(JobLimitRestartJob.class, args);
    }
}
```

变动：

step1() 添加了**.startLimit(2)** 表示运行重启2次，注意，第一次启动也算一次

tasklet1() 设置setTerminateOnly 第一次先让step1 状态为**STOPPED**

第一次执行， step1 为 **STOPPED** 状态

第二次执行，不做任何操作，第二次执行，step1 还是STOPPED状态

第三次执行， 注释掉tasklet1() 中setTerminateOnly ， 查询结果

8.3.3 无限重启

Spring Batch 限制同job名跟同标识参数作业只能成功执行一次，这是Spring Batch 定理，无法改变的。但是，对于步骤不一定适用，可以通过步骤的 allowStartIfComplete(true) 实现步骤的无限重启。

```
package com.langfeiyes.batch._20_job_restart_allow;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.StepContribution;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.scope.context.ChunkContext;
import org.springframework.batch.core.step.tasklet.Tasklet;
import org.springframework.batch.repeat.RepeatStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
@EnableBatchProcessing
public class JobAllowRestartJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public Tasklet tasklet1(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.err.println("-------------tasklet1-------------");
                return RepeatStatus.FINISHED;

            }
        };
    }

    @Bean
    public Tasklet tasklet2(){
        return new Tasklet() {
            @Override
            public RepeatStatus execute(StepContribution contribution, Chun
                System.err.println("-------------tasklet2-------------");
                return RepeatStatus.FINISHED;
            }
        };
    }


    @Bean
```

```
    public Step step1(){
        return stepBuilderFactory.get("step1")
                .tasklet(tasklet1())
                .build();
    }

    @Bean
    public Step step2(){
        return stepBuilderFactory.get("step2")
                .tasklet(tasklet2())
                .build();
    }

    //定义作业
    @Bean
    public Job job(){
        return jobBuilderFactory.get("job-allow-restart-job")
                .start(step1())
                .next(step2())
                .build();

    }
    public static void main(String[] args) {
        SpringApplication.run(JobAllowRestartJob.class, args);
    }
}
```

观察上面代码，很正常逻辑

第一次启动：step1 step2正常执行，整个Job 成功执行完成

第二次启动：不做任何改动时，再次启动job，没有报错，但是观察数据库表batch_job_execution 状态为 **NOOP** 无效执行，step1 step2 不会执行。

第三次启动：给step1 step2 添加上**.allowStartIfComplete(true)**， 再次启动，一切正常，并且可以无限启动

九、ItemReader

居于块操作的步骤由一个ItemReader，一个ItemProcessor和一个ItemWriter组成，一个负责读取数据，一个负责处理数据，一个负责输出数据，上一章节讲完步骤，接下来就重点讲解Spring Batch 输入组件：**ItemReader**

ItemReader 是Spring Batch 提供的输入组件，规范接口是ItemReader, 里面有个read()方法，我们可以实现该接口去定制输入逻辑。

```
public interface ItemReader<T> {
        @Nullable
        T read() throws Exception, UnexpectedInputException, ParseExceptior
}
```

Spring Batch 根据常用的输入类型，提供许多默认的实现，包括：平面文件、数据库、

JMS资源和其他输入源等，接下来一起操作一下比较场景的输入场景。

9.1 读平面文件

平面文件一般指的都是简单行/多行结构的纯文本文件，比如记事本记录文件。与xml这种区别在于没有结构，没有标签的限制。Spring Batch默认使用 FlatFileItemReader 实现平面文件的输入。

9.1.1 方式1：delimited–字符串截取

需求：读取**user.txt**文件，解析出所有用户信息

user.txt

```
1#dafei#18
2#xiaofei#16
3#laofei#20
4#zhongfei#19
5#feifei#15
```

实体类

```
@Getter
@Setter
@ToString
public class User {
    private Long id;
    private String name;
    private int age;
}
```

实现作业

```
package com.langfeiyes.batch._21_itemreader_flat;

import com.langfeiyes.batch._20_job_restart_allow.JobAllowRestartJob;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.item.ItemWriter;
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.builder.FlatFileItemReaderBuilde
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

import java.util.List;

@SpringBootApplication
@EnableBatchProcessing
public class FlatReaderJob {
```

```
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public FlatFileItemReader<User> userItemReader(){
        return new FlatFileItemReaderBuilder<User>()
                .name("userItemReader")
                .resource(new ClassPathResource("users.txt"))
                .delimited().delimiter("#")
                .names("id", "name", "age")
                .targetType(User.class)

                .build();
    }

    @Bean
    public ItemWriter<User> itemWriter(){
        return new ItemWriter<User>() {
            @Override
            public void write(List<? extends User> items) throws Exception
                items.forEach(System.err::println);
            }
        };
    }

    @Bean
    public Step step(){
        return stepBuilderFactory.get("step1")
                .<User, User>chunk(1)
                .reader(userItemReader())
                .writer(itemWriter())
                .build();

    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("flat-reader-job")
                .start(step())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(FlatReaderJob.class, args);
    }
}
```

核心在userItemReader() 实例方法

```
//FlatFileItemReader spring batch 平面文件读入类
//这个类操作特点：一行一行的读数据
@Bean
public FlatFileItemReader<User> userItemReader(){
    return new FlatFileItemReaderBuilder<User>()
        .name("userItemReader")
        .resource(new ClassPathResource("users.txt"))  //指定读取的文件
        .delimited().delimiter("#")  //读取出一行数据，该如何分割数据，默认以,分割
```

```
        .targetType(User.class)       //读取出一行数据封装成什么对象
        //给分割后数据打name标记，后续跟User对象属性进行映射
        .names("id", "name", "age")
        .build();
}
```

除了上面讲到的核心方法，FlatFileItemReaderBuilder还提供.**fieldSetMapper** **.lineTokenizer** 2个方法，用于定制文件解析与数据映射。

9.1.2 方式2：FieldSetMapper–字段映射

FlatFileItemReaderBuilder 提供的方法，用于字段映射，方法参数是一个FieldSetMapper 接口对象

```
public interface FieldSetMapper<T> {
        T mapFieldSet(FieldSet fieldSet) throws BindException;
}
```

FieldSet 字段集合，FlatFileItemReader 解析出一行数据，会将这行数据封装到FieldSet 对象中。

我们用一个案例来解释一下FieldSetMapper 用法

编写users2.txt文件

```
1#dafei#18#广东#广州#天河区
2#xiaofei#16#四川#成都#武侯区
3#laofei#20#广西#桂林#雁山区
4#zhongfei#19#广东#广州#白云区
5#feifei#15#广东#广州#越秀区
```

用户对象

```
@Getter
@Setter
@ToString
public class User {
    private Long id;
    private String name;
    private int age;
    private String address;
}
```

观察，user2.txt文件中有 id name age province city area 按理用户对象属性应该一一对应，但是此时User只有address，也就是说，后续要将 province ， city ， area 合并成 address 地址值。此时怎么办？这是就需要自定义FieldSetMapper 啦。

```
public class UserFieldMapper implements FieldSetMapper<User> {
    @Override
    public User mapFieldSet(FieldSet fieldSet) throws BindException {

        //自己定义映射逻辑
        User User = new User();
        User.setId(fieldSet.readLong("id"));
        User.setAge(fieldSet.readInt("age"));
        User.setName(fieldSet.readString("name"));
```

```
        String addr = fieldSet.readString("province") + " "
                + fieldSet.readString("city") + " " + fieldSet.readString('
        User.setAddress(addr);
        return User;
    }
}
```

上面代码实现FieldSet与User对象映射，将province city area 合并成一个属性address。
另外readXxx 是FieldSet 独有的方法，Xxx是java基本类型。

```
package com.langfeiyes.batch._22_itemreader_flat_mapper;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.item.ItemWriter;
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.builder.FlatFileItemReaderBuilde
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ClassPathResource;

import java.util.List;

@SpringBootApplication
@EnableBatchProcessing
public class MapperFlatReaderJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public UserFieldMapper userFieldMapper(){
        return new UserFieldMapper();
    }


    @Bean
    public FlatFileItemReader<User> userItemReader(){
        return new FlatFileItemReaderBuilder<User>()
                .name("userMapperItemReader")
                .resource(new ClassPathResource("users2.txt"))
                .delimited().delimiter("#")
                .names("id", "name", "age", "province", "city", "area")
                .fieldSetMapper(userFieldMapper())
                .build();
    }

    @Bean
    public ItemWriter<User> itemWriter(){
        return new ItemWriter<User>() {
            @Override
            public void write(List<? extends User> items) throws Exception
                items.forEach(System.err::println);
```

```
            }
        };
    }

    @Bean
    public Step step(){
        return stepBuilderFactory.get("step1")
                .<User, User>chunk(1)
                .reader(userItemReader())
                .writer(itemWriter())
                .build();

    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("mapper-flat-reader-job")
                .start(step())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(MapperFlatReaderJob.class, args);
    }
}
```

上面代码核心在userItemReader实例方法

.fieldSetMapper(userFieldMapper())：用上自定义的字段映射器

.names(“id”, “name”, “age”, “province”, “city”, “area”)： users2.txt 每一行使用#分割出现6列，给每一列取名字，然后将其封装到FieldSet对象中

.targetType(User.class)： 注意，使用了fieldSetMapper 之后，不需要在加上这行

9.2 读JSON文件

Spring Batch 也提供专门操作Json文档的API： JsonItemReader，具体使用且看案例

需求：读取下面**json**格式文档

```
[
  {"id":1, "name":"dafei", "age":18},
  {"id":2, "name":"xiaofei", "age":17},
  {"id":3, "name":"zhongfei", "age":16},
  {"id":4, "name":"laofei", "age":15},
  {"id":5, "name":"feifei", "age":14}
]
```

封装成User对象

```
@Getter
@Setter
@ToString
public class User {
    private Long id;
```

```
    private String name;
    private int age;
}
```

```
package com.langfeiyes.batch._23_itemreader_flat_json;

import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.item.ItemWriter;
import org.springframework.batch.item.json.JacksonJsonObjectReader;
import org.springframework.batch.item.json.JsonItemReader;
import org.springframework.batch.item.json.builder.JsonItemReaderBuilder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ClassPathResource;

import java.util.List;

@SpringBootApplication
@EnableBatchProcessing
public class JsonFlatReaderJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;


    @Bean
    public JsonItemReader<User> userItemReader(){

        ObjectMapper objectMapper = new ObjectMapper();
        JacksonJsonObjectReader<User> jsonObjectReader = new JacksonJsonObj
        jsonObjectReader.setMapper(objectMapper);

        return new JsonItemReaderBuilder<User>()
                .name("userJsonItemReader")
                .jsonObjectReader(jsonObjectReader)
                .resource(new ClassPathResource("users.json"))
            .build();
        }

    @Bean
    public ItemWriter<User> itemWriter(){
        return new ItemWriter<User>() {
            @Override
            public void write(List<? extends User> items) throws Exception
                items.forEach(System.err::println);
            }
        };
    }

    @Bean
    public Step step(){
        return stepBuilderFactory.get("step1")
```

```
                .<User, User>chunk(1)
                .reader(userItemReader())
                .writer(itemWriter())
                .build();

    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("json-flat-reader-job")
                .start(step())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(JsonFlatReaderJob.class, args);
    }
}
```

上面代码核心在：userItemReader() 实例方法，明确指定转换成json格式需要使用转换器，本次使用的Jackson

9.3 读数据库

下面是一张用户表user， 如果数据是存放在数据库中，那么又该怎么读取？

```
CREATE TABLE `user` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT '主键',
  `name` varchar(255) DEFAULT NULL COMMENT '用户名',
  `age` int DEFAULT NULL COMMENT '年龄',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb3;
```

```
INSERT INTO `user` VALUES (1, 'dafei', 18);
INSERT INTO `user` VALUES (2, 'xiaofei', 17);
INSERT INTO `user` VALUES (3, 'zhongfei', 16);
INSERT INTO `user` VALUES (4, 'laofei', 15);
INSERT INTO `user` VALUES (5, 'feifei', 14);
```

Spring Batch 提供2种从数据库中读取数据的方式：

9.3.1 居于游标方式

游标是数据库中概念，可以简单理解为一个指针

游标遍历时，获取数据表中某一条数据，如果使用JDBC操作，游标指向的那条数据会被封装到ResultSet中，如果想将数据从ResultSet读取出来，需要借助Spring Batch 提供 RowMapper 实现表数据与实体对象的映射。

```
user表数据---->User对象
```

Spring Batch JDBC 实现数据表读取需要做几个准备

1>实体对象User

```
@Getter
@Setter
@ToString
public class User {
    private Long id;
    private String name;
    private int age;
}
```

2>RowMapper 表与实体对象映射实现类

```
public class UserRowMapper implements RowMapper<User> {
    @Override
    public User mapRow(ResultSet rs, int rowNum) throws SQLException {
        User user = new User();
        user.setId(rs.getLong("id"));
        user.setName(rs.getString("name"));
        user.setAge(rs.getInt("age"));
        return user;
    }
}
```

3>JdbcCursorItemReader编写

```
package com.langfeiyes.batch._24_itemreader_db_cursor;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.item.ItemWriter;
import org.springframework.batch.item.database.JdbcCursorItemReader;
import org.springframework.batch.item.database.builder.JdbcCursorItemReader
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
```

```
import javax.sql.DataSource;
import java.util.List;

@SpringBootApplication
@EnableBatchProcessing
public class CursorDBReaderJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Autowired
    private DataSource dataSource;

    @Bean
    public UserRowMapper userRowMapper(){
        return new UserRowMapper();
    }

    @Bean
    public JdbcCursorItemReader<User> userItemReader(){

        return new JdbcCursorItemReaderBuilder<User>()
                .name("userCursorItemReader")
                .dataSource(dataSource)
                .sql("select * from user")
                .rowMapper(userRowMapper())
                .build();
        }

    @Bean
    public ItemWriter<User> itemWriter(){
        return new ItemWriter<User>() {
            @Override
            public void write(List<? extends User> items) throws Exception
                items.forEach(System.err::println);
            }
        };
    }

    @Bean
    public Step step(){
        return stepBuilderFactory.get("step1")
                .<User, User>chunk(1)
                .reader(userItemReader())
                .writer(itemWriter())
                .build();

    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("cursor-db-reader-job")
                .start(step())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(CursorDBReaderJob.class, args);
    }
```

```
}
```

解析：

1>操作数据库，需要引入DataSource

2>留意userItemReader() 方法，需要明确指定操作数据库sql

3>留意userItemReader() 方法，需要明确指定游标回来之后，数据映射规则：rowMapper

这里要注意，如果sql需要where 条件，需要额外定义

比如： 查询 age > 16的用户

```
@Bean
public JdbcCursorItemReader<User> userItemReader(){

    return new JdbcCursorItemReaderBuilder<User>()
        .name("userCursorItemReader")
        .dataSource(dataSource)
        .sql("select * from user where age > ?")
        .rowMapper(userRowMapper())
        //拼接参数
        .preparedStatementSetter(new ArgumentPreparedStatementSetter(new Ob
        .build();
}
```

9.3.2 居于分页方式

游标的方式是查询出所有满足条件的数据，然后一条一条读取，而分页是按照指定设置的pageSize数，一次性读取pageSize条。

分页查询方式需要几个要素

1>实体对象，跟游标方式一样

2>RowMapper映射对象，跟游标方式一样

3>数据源，跟游标方式一样

4>PagingQueryProvider 分页逻辑提供者

```
package com.langfeiyes.batch._25_itemreader_db_page;


import org.springframework.batch.core.Job;
```

```
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.item.ItemWriter;
import org.springframework.batch.item.database.JdbcCursorItemReader;
import org.springframework.batch.item.database.JdbcPagingItemReader;
import org.springframework.batch.item.database.PagingQueryProvider;
import org.springframework.batch.item.database.builder.JdbcCursorItemReader
import org.springframework.batch.item.database.builder.JdbcPagingItemReader
import org.springframework.batch.item.database.support.SqlPagingQueryProvid
import org.springframework.batch.item.database.support.SqlitePagingQueryPro
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.jdbc.core.ArgumentPreparedStatementSetter;

import javax.sql.DataSource;
import java.util.HashMap;
import java.util.List;

@SpringBootApplication
@EnableBatchProcessing
public class PageDBReaderJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Autowired
    private DataSource dataSource;

    @Bean
    public UserRowMapper userRowMapper(){
        return new UserRowMapper();
    }


    @Bean
    public PagingQueryProvider pagingQueryProvider() throws Exception {
        SqlPagingQueryProviderFactoryBean factoryBean = new SqlPagingQueryP
        factoryBean.setDataSource(dataSource);
        factoryBean.setSelectClause("select *");   //查询列
        factoryBean.setFromClause("from user");    //查询的表
        factoryBean.setWhereClause("where age > :age"); //where 条件
        factoryBean.setSortKey("id");   //结果排序
        return factoryBean.getObject();
    }

    @Bean
    public JdbcPagingItemReader<User> userItemReader() throws Exception {
        HashMap<String, Object> param = new HashMap<>();
        param.put("age", 16);
        return new JdbcPagingItemReaderBuilder<User>()
                .name("userPagingItemReader")
                .dataSource(dataSource)  //数据源
                .queryProvider(pagingQueryProvider())  //分页逻辑
                .parameterValues(param)   //条件
                .pageSize(10) //每页显示条数
```

```
                .rowMapper(userRowMapper())  //映射规则
                .build();
    }

    @Bean
    public ItemWriter<User> itemWriter(){
        return new ItemWriter<User>() {
            @Override
            public void write(List<? extends User> items) throws Exception
                items.forEach(System.err::println);
            }
        };
    }

    @Bean
    public Step step() throws Exception {
        return stepBuilderFactory.get("step1")
                .<User, User>chunk(1)
                .reader(userItemReader())
                .writer(itemWriter())
                .build();

    }

    @Bean
    public Job job() throws Exception {
        return jobBuilderFactory.get("page-db-reader-job1")
                .start(step())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(PageDBReaderJob.class, args);
    }
}
```

解析：

1>需要提供pagingQueryProvider 用于拼接分页SQL

2>userItemReader() 组装分页查询逻辑。

9.4 读取异常

任何输入都有可能存在异常情况，那Spring Batch 如何应对输入异常情况呢？ 3种操作逻辑：

1>跳过异常记录

这里逻辑是当Spring Batch 在读取数据时，根据各种意外情况抛出不同异常，ItemReader 可以按照约定跳过指定的异常，同时也可以限制跳过次数。

```
@Bean
public Step step() throws Exception {
    return stepBuilderFactory.get("step1")
        .<User, User>chunk(1)
```

```
        .reader(userItemReader())
        .writer(itemWriter())
        .faultTolerant() //容错
        .skip(Exception.class)  //跳过啥异常
        .noSkip(RuntimeException.class)  //不能跳过啥异常
        .skipLimit(10)  //跳过异常次数
        .skipPolicy(new SkipPolicy() {
            @Override
            public boolean shouldSkip(Throwable t, int skipCount) throws Sk
                //定制跳过异常与异常次数
                return false;
            }
        })
        .build();

}
```

如果出错直接跳过去，这操作有点自欺欺人，并不是优雅的解决方案。开发可选下面这种。

2>异常记录记日志

所谓记录日志，就是当ItemReader 读取数据抛出异常时，将具体数据信息记录下来，方便后续人工接入。

具体实现使用ItemReader监听器。

```
public class ErrorItemReaderListener implements ItemReadListener {
    @Override
    public void beforeRead() {

    }

    @Override
    public void afterRead(Object item) {

    }

    @Override
    public void onReadError(Exception ex) {
        System.out.println("记录读数据相关信息...");
    }
}
```

3>放弃处理

这种异常在处理不是很重要数据时候使用。

十、ItemProcessor

前面我们多次讲过，居于块的读与写，中间还夹着一个ItemProcessor 条目处理。当我们通过ItemReader 将数据读取出来之后，你面临2个选择：

1>直接将数据转向输出

2>对读入的数据进行再加工。

如果选择第一种，那ItemProcessor 可以不用出现，如果选择第二种，就需要引入ItemProcessor 条目处理组件啦。

Spring Batch 为Processor 提供默认的处理器与自定义处理器2种模式以满足各种需求。

10.1 默认ItemProcessor

Spring Batch 提供现成的ItemProcessor 组件有4种：

10.1.1 ValidatingItemProcessor：校验处理器

这个好理解，很多时候ItemReader读出来的数据是相对原始的数据，并没有做过多的校验

数据文件users-validate.txt

```
1##18
2##16
3#laofei#20
4#zhongfei#19
5#feifei#15
```

比如上面文本数据，第一条，第二条name数值没有指定，在ItemReader 读取之后，必定将 “” 空串封装到User name属性中，语法上没有错，但逻辑上可以做文章，比如：用户名不为空。

解决上述问题，可以使用Spring Batch 提供ValidatingItemProcessor 校验器处理。

接下来我们看下ValidatingItemProcessor 怎么实现

1>导入校验依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

2>定义实体对象

```
@Getter
@Setter
@ToString
public class User {
    private Long id;
    @NotBlank(message = "用户名不能为null或空串")
    private String name;
    private int age;
}
```

3>实现

```
package com.langfeiyes.batch._26_itemprocessor_validate;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.item.ItemWriter;
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.builder.FlatFileItemReaderBuilde
import org.springframework.batch.item.validator.BeanValidatingItemProcessor
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ClassPathResource;
import org.springframework.util.StringUtils;

import java.util.List;

@SpringBootApplication
@EnableBatchProcessing
public class ValidationProcessorJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;


    @Bean
    public FlatFileItemReader<User> userItemReader(){
        return new FlatFileItemReaderBuilder<User>()
                .name("userItemReader")
                .resource(new ClassPathResource("users-validate.txt"))
                .delimited().delimiter("#")
                .names("id", "name", "age")
                .targetType(User.class)
                .build();
    }

    @Bean
    public ItemWriter<User> itemWriter(){
        return new ItemWriter<User>() {
            @Override
            public void write(List<? extends User> items) throws Exception
                items.forEach(System.err::println);
            }
        };
    }


    @Bean
    public BeanValidatingItemProcessor<User> beanValidatingItemProcessor(){
        BeanValidatingItemProcessor<User> beanValidatingItemProcessor = new
        beanValidatingItemProcessor.setFilter(true);  //不满足条件丢弃数据

        return beanValidatingItemProcessor;
    }
```

```
    @Bean
    public Step step(){
        return stepBuilderFactory.get("step1")
                .<User, User>chunk(1)
                .reader(userItemReader())
                .processor(beanValidatingItemProcessor())
                .writer(itemWriter())
                .build();

    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("validate-processor-job4")
                .start(step())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(ValidationProcessorJob.class, args);
    }
}
```

解析：

1>核心是beanValidatingItemProcessor() 实例方法，核心BeanValidatingItemProcessor类是Spring Batch 提供现成的Validator校验类，这里直接使用即可。BeanValidatingItemProcessor 是 ValidatingItemProcessor 子类

2> step()实例方法，多了**.processor(beanValidatingItemProcessor())** 操作，引入ItemProcessor 组件。

10.1.2 ItemProcessorAdapter：适配器处理器

开发中，很多的校验逻辑已经有现成的啦，那做ItemProcessor处理时候，是否能使用现成逻辑呢？答案 是：yes

比如：现有处理逻辑：将User对象中name转换成大写

```
public class UserServiceImpl{
    public User toUppeCase(User user){
        user.setName(user.getName().toUpperCase());
        return user;
    }
}
```

新建users-adapter.txt 文件，用于测试

```
1#dafei#18
2#xiaofei#16
3#laofei#20
4#zhongfei#19
5#feifei#15
```

完整的逻辑

```
package com.langfeiyes.batch._27_itemprocessor_adapter;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.item.ItemWriter;
import org.springframework.batch.item.adapter.ItemProcessorAdapter;
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.builder.FlatFileItemReaderBuilde
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ClassPathResource;

import java.util.List;

@SpringBootApplication
@EnableBatchProcessing
public class AdapterProcessorJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;


    @Bean
    public FlatFileItemReader<User> userItemReader(){
        return new FlatFileItemReaderBuilder<User>()
                .name("userItemReader")
                .resource(new ClassPathResource("users-adapter.txt"))
                .delimited().delimiter("#")
                .names("id", "name", "age")
                .targetType(User.class)
                .build();
    }

    @Bean
    public ItemWriter<User> itemWriter(){
        return new ItemWriter<User>() {
            @Override
            public void write(List<? extends User> items) throws Exception
                items.forEach(System.err::println);
            }
        };
    }
    @Bean
    public UserServiceImpl userService(){
        return new UserServiceImpl();
    }
    @Bean
    public ItemProcessorAdapter<User, User> itemProcessorAdapter(){
        ItemProcessorAdapter<User, User> adapter = new ItemProcessorAdapter
        adapter.setTargetObject(userService());
        adapter.setTargetMethod("toUppeCase");

        return adapter;
    }
```

```
    @Bean
    public Step step(){
        return stepBuilderFactory.get("step1")
                .<User, User>chunk(1)
                .reader(userItemReader())
                .processor(itemProcessorAdapter())
                .writer(itemWriter())
                .build();

    }
    @Bean
    public Job job(){
        return jobBuilderFactory.get("adapter-processor-job")
                .start(step())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(AdapterProcessorJob.class, args);
    }
}
```

解析：

观察itemProcessorAdapter()实例方法，引入ItemProcessorAdapter 适配器类，绑定自定义的UserServiceImpl 类与toUppeCase方法，当ItemReader 读完之后，马上调用UserServiceImpl 类的toUppeCase 方法处理逻辑。方法传参数会被忽略，ItemProcessor会自动处理。

10.1.3 ScriptItemProcessor：脚本处理器

前面要实现User name 变大写，需要大费周折，又定义类，又是定义方法，能不能简化一点。答案也是yes， Spring Batch 提供js脚本的形式，将上面逻辑写到js文件中，加载这文件，就可以实现，省去定义类，定义方法的麻烦。

需求：使用**js**脚本方式实现用户名大写处理

userScript.js

```
item.setName(item.getName().toUpperCase());
item;
```

这里注意：

1>item是约定的单词，表示ItemReader读除来每个条目

2>userScript.js文件放置到resource资源文件中

完整代码

```
package com.langfeiyes.batch._28_itemprocessor_script;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
```

```
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.item.ItemWriter;
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.builder.FlatFileItemReaderBuilde
import org.springframework.batch.item.support.ScriptItemProcessor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ClassPathResource;

import java.util.List;

@SpringBootApplication
@EnableBatchProcessing
public class ScriptProcessorJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;


    @Bean
    public FlatFileItemReader<User> userItemReader(){
        return new FlatFileItemReaderBuilder<User>()
                .name("userItemReader")
                .resource(new ClassPathResource("users-adapter.txt"))
                .delimited().delimiter("#")
                .names("id", "name", "age")
                .targetType(User.class)
                .build();
    }

    @Bean
    public ItemWriter<User> itemWriter(){
        return new ItemWriter<User>() {
            @Override
            public void write(List<? extends User> items) throws Exception
                items.forEach(System.err::println);
            }
        };
    }


    @Bean
    public ScriptItemProcessor<User, User> scriptItemProcessor(){
        ScriptItemProcessor<User, User> scriptItemProcessor = new ScriptIte
        scriptItemProcessor.setScript(new ClassPathResource("userScript.js'
        return scriptItemProcessor;
    }

    @Bean
    public Step step(){
        return stepBuilderFactory.get("step1")
                .<User, User>chunk(1)
                .reader(userItemReader())
                .processor(scriptItemProcessor())
                .writer(itemWriter())
```

```
                .build();

    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("script-processor-job")
                .start(step())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(ScriptProcessorJob.class, args);
    }
}
```

解析：

核心还是scriptItemProcessor() 实例方法，ScriptItemProcessor 类用于加载js 脚本并处理js脚本。

10.1.4 CompositeItemProcessor：组合处理器

CompositeItemProcessor是一个ItemProcessor处理组合，类似于过滤器链，数据先经过第一个处理器，然后再经过第二个处理器，直到最后。前一个处理器处理的结果，是后一个处理器的输出。

需求：将解析出来用户**name**进行判空处理，并将**name**属性转换成大写

1>读取文件：users-validate.txt

```
1##18
2##16
3#laofei#20
4#zhongfei#19
5#feifei#15
```

2>封装的实体对象

```
@Getter
@Setter
@ToString
public class User {
    private Long id;
    @NotBlank(message = "用户名不能为null或空串")
```

```
    private String name;
    private int age;
}
```

3>用于转换大写工具类

```
public class UserServiceImpl {
    public User toUppeCase(User user){
        user.setName(user.getName().toUpperCase());
        return user;
    }
}
```

4>完整代码

```
package com.langfeiyes.batch._29_itemprocessor_composite;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.item.ItemWriter;
import org.springframework.batch.item.adapter.ItemProcessorAdapter;
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.builder.FlatFileItemReaderBuilde
import org.springframework.batch.item.support.CompositeItemProcessor;
import org.springframework.batch.item.validator.BeanValidatingItemProcessor
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ClassPathResource;

import java.util.Arrays;
import java.util.List;

@SpringBootApplication
@EnableBatchProcessing
public class CompositeProcessorJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;


    @Bean
    public FlatFileItemReader<User> userItemReader(){
        return new FlatFileItemReaderBuilder<User>()
                .name("userItemReader")
                .resource(new ClassPathResource("users-validate.txt"))
                .delimited().delimiter("#")
                .names("id", "name", "age")
                .targetType(User.class)
                .build();
    }

    @Bean
    public ItemWriter<User> itemWriter(){
        return new ItemWriter<User>() {
```

```
            @Override
            public void write(List<? extends User> items) throws Exception
                items.forEach(System.err::println);
            }
        };
    }

    @Bean
    public UserServiceImpl userService(){
        return new UserServiceImpl();
    }
    @Bean
    public BeanValidatingItemProcessor<User> beanValidatingItemProcessor(){
        BeanValidatingItemProcessor<User> beanValidatingItemProcessor = new
        beanValidatingItemProcessor.setFilter(true);  //不满足条件丢弃数据
        return beanValidatingItemProcessor;
    }

    @Bean
    public ItemProcessorAdapter<User, User> itemProcessorAdapter(){
        ItemProcessorAdapter<User, User> adapter = new ItemProcessorAdapter
        adapter.setTargetObject(userService());
        adapter.setTargetMethod("toUppeCase");

        return adapter;
    }

    @Bean
    public CompositeItemProcessor<User, User>  compositeItemProcessor(){
        CompositeItemProcessor<User, User> compositeItemProcessor = new Com
        compositeItemProcessor.setDelegates(Arrays.asList(
                beanValidatingItemProcessor(), itemProcessorAdapter()
        ));
        return  compositeItemProcessor;
    }

    @Bean
    public Step step(){
        return stepBuilderFactory.get("step1")
                .<User, User>chunk(1)
                .reader(userItemReader())
                .processor(compositeItemProcessor())
                .writer(itemWriter())
                .build();

    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("composite-processor-job")
                .start(step())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(CompositeProcessorJob.class, args);
    }
}
```

解析：

核心代码：compositeItemProcessor() 实例方法，使用setDelegates 操作将其他 ItemProcessor 处理合并成一个。

10.2 自定义ItemProcessor处理器

除去上面默认的几种处理器外，Spring Batch 也允许我们自定义，具体做法只需要实现 ItemProcessor接口即可

需求：自定义处理器，筛选出**id**为偶数的用户

1>定义读取文件user.txt

```
1#dafei#18
2#xiaofei#16
3#laofei#20
4#zhongfei#19
5#feifei#15
```

2>定义实体对象

```
@Getter
@Setter
@ToString
public class User {
    private Long id;
    private String name;
    private int age;
}
```

3>自定义处理器

```
//自定义
public class CustomizeItemProcessor implements ItemProcessor<User,User> {
    @Override
    public User process(User item) throws Exception {
        //id 为偶数的用户放弃
        //返回null时候 读入的item会被放弃，不会进入itemwriter
        return item.getId() % 2 != 0 ? item : null;
    }
}
```

4>完整代码

```
package com.langfeiyes.batch._30_itemprocessor_customize;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.item.ItemWriter;
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.builder.FlatFileItemReaderBuilde
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ClassPathResource;

import java.util.List;

@SpringBootApplication
@EnableBatchProcessing
public class CustomizeProcessorJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;


    @Bean
    public FlatFileItemReader<User> userItemReader(){
        return new FlatFileItemReaderBuilder<User>()
                .name("userItemReader")
                .resource(new ClassPathResource("users.txt"))
                .delimited().delimiter("#")
                .names("id", "name", "age")
                .targetType(User.class)
                .build();
    }

    @Bean
    public ItemWriter<User> itemWriter(){
        return new ItemWriter<User>() {
            @Override
            public void write(List<? extends User> items) throws Exception
                items.forEach(System.err::println);
            }
        };
    }
    @Bean
    public CustomizeItemProcessor customizeItemProcessor(){
        return new CustomizeItemProcessor();
    }

    @Bean
    public Step step(){
        return stepBuilderFactory.get("step1")
                .<User, User>chunk(1)
                .reader(userItemReader())
                .processor(customizeItemProcessor())
                .writer(itemWriter())
                .build();

    }
    @Bean
    public Job job(){
        return jobBuilderFactory.get("customize-processor-job")
                .start(step())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(CustomizeProcessorJob.class, args);
    }
```

```
}
```

十一、ItemWriter

有输入那肯定有输出，前面讲了输入ItemReader，接下来就看本篇的输出器：ItemWriter， Spring Batch提供的数据输出组件与数据输入组件是成对，也就是说有啥样子的输入组件，就有啥样子的输出组件。

11.1 输出平面文件

当将读入的数据输出到纯文本文件时，可以通过FlatFileItemWriter 输出器实现。

需求：将**user.txt**中数据读取出来，输出到**outUser.txt**文件中

1>定义user.txt文件

```
1#dafei#18
2#xiaofei#16
3#laofei#20
4#zhongfei#19
5#feifei#15
```

2>定义实体对象

```
@Getter
@Setter
@ToString
public class User {
    private Long id;
    private String name;
    private int age;
}
```

3>实现代码

```
package com.langfeiyes.batch._31_itemwriter_flat;


import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.FlatFileItemWriter;
import org.springframework.batch.item.file.builder.FlatFileItemReaderBuilde
import org.springframework.batch.item.file.builder.FlatFileItemWriterBuilde
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.PathResource;
```

```
import java.util.List;

@SpringBootApplication
@EnableBatchProcessing
public class FlatWriteJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public FlatFileItemReader<User> userItemReader(){
        return new FlatFileItemReaderBuilder<User>()
                .name("userItemReader")
                .resource(new ClassPathResource("users.txt"))
                .delimited().delimiter("#")
                .names("id", "name", "age")
                .targetType(User.class)
                .build();
    }

    @Bean
    public FlatFileItemWriter<User> itemWriter(){
        return new FlatFileItemWriterBuilder<User>()
                .name("userItemWriter")
                .resource(new PathResource("c:/outUser.txt"))  //输出的文件
                .formatted()  //数据格式指定
                .format("id: %s,姓名：%s,年龄：%s")  //输出数据格式
                .names("id", "name", "age")  //需要输出属性
                .build();
    }

    @Bean
    public Step step(){
        return stepBuilderFactory.get("step1")
                .<User, User>chunk(1)
                .reader(userItemReader())
                .writer(itemWriter())
                .build();

    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("flat-writer-job")
                .start(step())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(FlatWriteJob.class, args);
    }
}
```

解析：

上面代码核心是itemWriter() 方法，设置到itemWrite读取器配置与输出

```
id: 1,姓名：dafei,年龄：18
id: 2,姓名：xiaofei,年龄：16
id: 3,姓名：laofei,年龄：20
id: 4,姓名：zhongfei,年龄：19
id: 5,姓名：feifei,年龄：15
```

一些拓展

```
@Bean
public FlatFileItemWriter<User> itemWriter(){
    return new FlatFileItemWriterBuilder<User>()
        .name("userItemWriter")
        .resource(new PathResource("c:/outUser.txt"))  //输出的文件
        .formatted()  //数据格式指定
        .format("id: %s,姓名：%s,年龄：%s")  //输出数据格式
        .names("id", "name", "age")  //需要输出属性
        .shouldDeleteIfEmpty(true)   //如果读入数据为空，输出时创建文件直接删除
        .shouldDeleteIfExists(true) //如果输出文件已经存在，则删除
        .append(true)  //如果输出文件已经存在， 不删除，直接追加到现有文件中
        .build();
}
```

11.2 输出Json文件

当将读入的数据输出到Json文件时，可以通过JsonFileItemWriter输出器实现。

需求：将**user.txt**中数据读取出来，输出到**outUser.json**文件中

沿用上面的user.txt， user对象将数据输出到outUser.json

```
package com.langfeiyes.batch._32_itemwriter_json;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.FlatFileItemWriter;
import org.springframework.batch.item.file.builder.FlatFileItemReaderBuilde
import org.springframework.batch.item.file.builder.FlatFileItemWriterBuilde
import org.springframework.batch.item.json.JacksonJsonObjectMarshaller;
import org.springframework.batch.item.json.JsonFileItemWriter;
import org.springframework.batch.item.json.builder.JsonFileItemWriterBuilde
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.PathResource;

@SpringBootApplication
@EnableBatchProcessing
public class JsonWriteJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
```

```
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public FlatFileItemReader<User> userItemReader(){
        return new FlatFileItemReaderBuilder<User>()
                .name("userItemReader")
                .resource(new ClassPathResource("users.txt"))
                .delimited().delimiter("#")
                .names("id", "name", "age")
                .targetType(User.class)
                .build();
    }


    @Bean
    public JacksonJsonObjectMarshaller<User> objectMarshaller(){
        JacksonJsonObjectMarshaller marshaller = new JacksonJsonObjectMarsh
        return marshaller;
    }

    @Bean
    public JsonFileItemWriter<User> itemWriter(){
        return new JsonFileItemWriterBuilder<User>()
                .name("jsonUserItemWriter")
                .resource(new PathResource("c:/outUser.json"))
                .jsonObjectMarshaller(objectMarshaller())
                .build();
    }


    @Bean
    public Step step(){
        return stepBuilderFactory.get("step1")
                .<User, User>chunk(1)
                .reader(userItemReader())
                .writer(itemWriter())
                .build();

    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("json-writer-job")
                .start(step())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(JsonWriteJob.class, args);
    }
}
```

结果：

```
[
 {"id":1,"name":"dafei","age":18},
 {"id":2,"name":"xiaofei","age":16},
 {"id":3,"name":"laofei","age":20},
 {"id":4,"name":"zhongfei","age":19},
 {"id":5,"name":"feifei","age":15}
```

```
]
```

解析：

1>itemWriter() 实例方法构建JsonFileItemWriter 实例，需要明确指定Json格式装配器

2>Spring Batch默认提供装配器有2个：JacksonJsonObjectMarshaller GsonJsonObjectMarshaller 分别对应Jackson 跟 Gson 2种json格式解析逻辑，本案例用的是Jackson

11.3 输出数据库

当将读入的数据需要输出到数据库时，可以通过JdbcBatchItemWriter输出器实现。

需求：将**user.txt**中数据读取出来，输出到数据库**user**表中

沿用上面的user.txt， user对象将数据输出到user表中

1>定义操作数据库预编译类

```
//写入数据库需要操作insert sql， 使用预编译就需要明确指定参数值
public class UserPreStatementSetter implements ItemPreparedStatementSetter<
    @Override
    public void setValues(User item, PreparedStatement ps) throws SQLExcept
        ps.setLong(1, item.getId());
        ps.setString(2, item.getName());
        ps.setInt(3, item.getAge());
    }
}
```

2>完整代码

```
package com.langfeiyes.batch._33_itemwriter_db;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.item.database.JdbcBatchItemWriter;
import org.springframework.batch.item.database.builder.JdbcBatchItemWriterB
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.builder.FlatFileItemReaderBuilde
import org.springframework.batch.item.json.JacksonJsonObjectMarshaller;
import org.springframework.batch.item.json.JsonFileItemWriter;
import org.springframework.batch.item.json.builder.JsonFileItemWriterBuilde
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.PathResource;

import javax.sql.DataSource;
```

```
@SpringBootApplication
@EnableBatchProcessing
public class JdbcWriteJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Autowired
    private DataSource dataSource;

    @Bean
    public FlatFileItemReader<User> userItemReader(){
        return new FlatFileItemReaderBuilder<User>()
                .name("userItemReader")
                .resource(new ClassPathResource("users.txt"))
                .delimited().delimiter("#")
                .names("id", "name", "age")
                .targetType(User.class)
                .build();
    }
    @Bean
    public UserPreStatementSetter preStatementSetter(){
        return new UserPreStatementSetter();
    }
    @Bean
    public JdbcBatchItemWriter<User>  itemWriter(){
        return new JdbcBatchItemWriterBuilder<User>()
                .dataSource(dataSource)
                .sql("insert into user(id, name, age) values(?,?,?)")
                .itemPreparedStatementSetter(preStatementSetter())
                .build();
    }
    @Bean
    public Step step(){
        return stepBuilderFactory.get("step1")
                .<User, User>chunk(1)
                .reader(userItemReader())
                .writer(itemWriter())
                .build();
    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("jdbc-writer-job")
                .start(step())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(JdbcWriteJob.class, args);
    }
}
```

解析：

核心代码在itemWriter() 实例方法中， 需要1>准备构建JdbcBatchItemWriter实例 2>提前准备数据， 3>准备sql语句 4>准备参数绑定器

11.4 输出多终端

上面几种输出方法都是一对一，真实开发可能没那么简单了，可能存在一对多，多个终端输出，此时怎么办？答案是使用Spring Batch 提供的CompositeItemWriter 组合输出器。

需求：将**user.txt**中数据读取出来，输出到**outUser.txt/outUser.json/**数据库**user**表中

沿用上面的user.txt， user对象将数据输出到outUser.txt/outUser.json/user表中

```
package com.langfeiyes.batch._34_itemwriter_composite;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.item.database.JdbcBatchItemWriter;
import org.springframework.batch.item.database.builder.JdbcBatchItemWriterB
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.FlatFileItemWriter;
import org.springframework.batch.item.file.builder.FlatFileItemReaderBuilde
import org.springframework.batch.item.file.builder.FlatFileItemWriterBuilde
import org.springframework.batch.item.json.JacksonJsonObjectMarshaller;
import org.springframework.batch.item.json.JsonFileItemWriter;
import org.springframework.batch.item.json.builder.JsonFileItemWriterBuilde
import org.springframework.batch.item.support.CompositeItemWriter;
import org.springframework.batch.item.support.builder.CompositeItemWriterBu
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.PathResource;

import javax.sql.DataSource;
import java.util.Arrays;

@SpringBootApplication
@EnableBatchProcessing
public class CompositeWriteJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Autowired
    public DataSource dataSource;

    @Bean
    public FlatFileItemReader<User> userItemReader(){
        return new FlatFileItemReaderBuilder<User>()
                .name("userItemReader")
                .resource(new ClassPathResource("users.txt"))
                .delimited().delimiter("#")
                .names("id", "name", "age")
                .targetType(User.class)
                .build();
    }
```

```
@Bean
public FlatFileItemWriter<User> flatFileItemWriter(){
    return new FlatFileItemWriterBuilder<User>()
            .name("userItemWriter")
            .resource(new PathResource("c:/outUser.txt"))
            .formatted()  //数据格式指定
            .format("id: %s,姓名：%s,年龄：%s")  //输出数据格式
            .names("id", "name", "age")  //需要输出属性
            .build();
}

@Bean
public JacksonJsonObjectMarshaller<User> objectMarshaller(){
    JacksonJsonObjectMarshaller marshaller = new JacksonJsonObjectMarsh
    return marshaller;
}

@Bean
public JsonFileItemWriter<User> jsonFileItemWriter(){
    return new JsonFileItemWriterBuilder<User>()
            .name("jsonUserItemWriter")
            .resource(new PathResource("c:/outUser.json"))
            .jsonObjectMarshaller(objectMarshaller())
            .build();
}

@Bean
public UserPreStatementSetter preStatementSetter(){
    return new UserPreStatementSetter();
}


@Bean
public JdbcBatchItemWriter<User> jdbcBatchItemWriter(){
    return new JdbcBatchItemWriterBuilder<User>()
            .dataSource(dataSource)
            .sql("insert into user(id, name, age) values(?,?,?)")
            .itemPreparedStatementSetter(preStatementSetter())
            .build();
}

@Bean
public CompositeItemWriter<User> compositeItemWriter(){
    return new CompositeItemWriterBuilder<User>()
            .delegates(Arrays.asList(flatFileItemWriter(), jsonFileItem
            .build();
}


@Bean
public Step step(){
    return stepBuilderFactory.get("step1")
            .<User, User>chunk(1)
            .reader(userItemReader())
            .writer(compositeItemWriter())
            .build();

}
```

```
    @Bean
    public Job job(){
        return jobBuilderFactory.get("composite-writer-job")
                .start(step())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(CompositeWriteJob.class, args);
    }
}
```

解析：

代码没有啥技术难度，都是将前面的几种方式通过CompositeItemWriter 类整合在一起

```
@Bean
public CompositeItemWriter<User> compositeItemWriter(){
    return new CompositeItemWriterBuilder<User>()
        .delegates(Arrays.asList(flatFileItemWriter(), jsonFileItemWriter()
        .build();
}
```

十二、Spring Batch 高级

前面讲的Spring Batch 基本上能满足日常批处理了，下面则是Spring Batch 高级部分内容，大家可以自己选择了解。

12.1 多线程步骤

默认的情况下，步骤基本上在单线程中执行，那能不能在多线程环境执行呢？答案肯定是yes，但是也要注意，多线程环境步骤执行一定要慎重。原因：多线程环境下，步骤是要设置不可重启。

Spring Batch 的多线程步骤是使用Spring 的 TaskExecutor(任务执行器)实现的。约定每一个块开启一个线程独立执行。

需求：分**5**个块处理**user-thread.txt**文件

1>编写user-thread.txt文件

```
1#dafei#18
2#xiaofei#16
3#laofei#20
4#zhongfei#19
5#feifei#15
6#zhangsan#14
7#lisi#13
8#wangwu#12
9#zhaoliu#11
10#qianqi#10
```

2>定义实体对象

```
@Getter
@Setter
@ToString
public class User {
    private Long id;
    private String name;
    private int age;
}
```

3>完整代码

```
package com.langfeiyes.batch._35_step_thread;


import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.item.ItemWriter;
import org.springframework.batch.item.file.FlatFileItemReader;
```

```
import org.springframework.batch.item.file.builder.FlatFileItemReaderBuilde
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.task.SimpleAsyncTaskExecutor;

import java.util.List;

@SpringBootApplication
@EnableBatchProcessing
public class ThreadStepJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public FlatFileItemReader<User> userItemReader(){

        System.out.println(Thread.currentThread());

        FlatFileItemReader<User> reader = new FlatFileItemReaderBuilder<Use
                .name("userItemReader")
                .saveState(false) //防止状态被覆盖
                .resource(new ClassPathResource("user-thread.txt"))
                .delimited().delimiter("#")
                .names("id", "name", "age")
                .targetType(User.class)
                .build();

        return reader;
    }

    @Bean
    public ItemWriter<User> itemWriter(){
        return new ItemWriter<User>() {
            @Override
            public void write(List<? extends User> items) throws Exception
                items.forEach(System.err::println);
            }
        };
    }

    @Bean
    public Step step(){
        return stepBuilderFactory.get("step1")
                .<User, User>chunk(2)
                .reader(userItemReader())
                .writer(itemWriter())
                .taskExecutor(new SimpleAsyncTaskExecutor())
                .build();

    }

    @Bean
    public Job job(){
        return jobBuilderFactory.get("thread-step-job")
                .start(step())
```

```
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(ThreadStepJob.class, args);
    }
}
```

4>结果

```
User(id=2, name=xiaofei, age=16)
User(id=5, name=feifei, age=15)
User(id=4, name=zhongfei, age=19)
User(id=7, name=lisi, age=13)
User(id=1, name=dafei, age=18)
User(id=6, name=zhangsan, age=14)
User(id=3, name=laofei, age=20)
User(id=8, name=wangwu, age=12)
User(id=9, name=zhaoliu, age=11)
User(id=10, name=qianqi, age=10)
```

解析

1：**userItemReader()** 加上**saveState(false)** Spring Batch 提供大部分的ItemReader是有状态的，作业重启基本通过状态来确定作业停止位置，而在多线程环境中，如果对象维护状态被多个线程访问，可能存在线程间状态相互覆盖问题。所以设置为false表示关闭状态，但这也意味着作业不能重启了。

2：**step()** 方法加上**.taskExecutor(new SimpleAsyncTaskExecutor())** 为作业步骤添加了多线程处理能力，以块为单位，一个块一个线程，观察上面的结果，很明显能看出输出的顺序是乱序的。改变 job 的名字再执行，会发现输出数据每次都不一样。

12.2 并行步骤

并行步骤，指的是某2个或者多个步骤同时执行。比如下图

图中，流程从步骤1执行，然后执行步骤2， 步骤3，当步骤2/3执行结束之后，在执行步骤4.

设想一种场景，当读取2个或者多个互不关联的文件时，可以多个文件同时读取，这个就

是并行步骤。

需求：现有**user-parallel.txt, user-parallel.json 2**个文件将它们中数据读入内存

1>编写user-parallel.txt, user-parallel.json

```
6#zhangsan#14
7#lisi#13
8#wangwu#12
9#zhaoliu#11
10#qianqi#10
```

```
[
  {"id":1, "name":"dafei", "age":18},
  {"id":2, "name":"xiaofei", "age":17},
  {"id":3, "name":"zhongfei", "age":16},
  {"id":4, "name":"laofei", "age":15},
  {"id":5, "name":"feifei", "age":14}
]
```

2>编写实体对象

```
@Getter
@Setter
@ToString
public class User {
    private Long id;
    private String name;
    private int age;
}
```

3>代码实现

```
package com.langfeiyes.batch._36_step_parallel;


import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.job.builder.FlowBuilder;
import org.springframework.batch.core.job.flow.Flow;
import org.springframework.batch.item.ItemWriter;
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.builder.FlatFileItemReaderBuilde
import org.springframework.batch.item.json.JacksonJsonObjectReader;
import org.springframework.batch.item.json.JsonItemReader;
import org.springframework.batch.item.json.builder.JsonItemReaderBuilder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.task.SimpleAsyncTaskExecutor;

import java.util.List;
```

```
@SpringBootApplication
@EnableBatchProcessing
public class ParallelStepJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;


    @Bean
    public JsonItemReader<User> jsonItemReader(){
        ObjectMapper objectMapper = new ObjectMapper();
        JacksonJsonObjectReader<User> jsonObjectReader = new JacksonJsonObj
        jsonObjectReader.setMapper(objectMapper);

        return new JsonItemReaderBuilder<User>()
                .name("userJsonItemReader")
                .jsonObjectReader(jsonObjectReader)
                .resource(new ClassPathResource("user-parallel.json"))
                .build();
    }

    @Bean
    public FlatFileItemReader<User> flatItemReader(){
        return new FlatFileItemReaderBuilder<User>()
                .name("userItemReader")
                .resource(new ClassPathResource("user-parallel.txt"))
                .delimited().delimiter("#")
                .names("id", "name", "age")
                .targetType(User.class)
                .build();
    }
    @Bean
    public ItemWriter<User> itemWriter(){
        return new ItemWriter<User>() {
            @Override
            public void write(List<? extends User> items) throws Exception
                items.forEach(System.err::println);
            }
        };
    }

    @Bean
    public Step jsonStep(){
        return stepBuilderFactory.get("jsonStep")
                .<User, User>chunk(2)
                .reader(jsonItemReader())
                .writer(itemWriter())
                .build();
    }

    @Bean
    public Step flatStep(){
        return stepBuilderFactory.get("step2")
                .<User, User>chunk(2)
                .reader(flatItemReader())
                .writer(itemWriter())
                .build();
    }
```

```
    @Bean
    public Job parallelJob(){

        //线程1-读user-parallel.txt
        Flow parallelFlow1 = new FlowBuilder<Flow>("parallelFlow1")
                .start(flatStep())
                .build();

        //线程2-读user-parallel.json
        Flow parallelFlow2 = new FlowBuilder<Flow>("parallelFlow2")
                .start(jsonStep())
                .split(new SimpleAsyncTaskExecutor())
                .add(parallelFlow1)
                .build();


        return jobBuilderFactory.get("parallel-step-job")
                .start(parallelFlow2)
                .end()
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(ParallelStepJob.class, args);
    }
}
```

结果

```
User(id=6, name=zhangsan, age=14)
User(id=7, name=lisi, age=13)
User(id=8, name=wangwu, age=12)
User(id=9, name=zhaoliu, age=11)
User(id=1, name=dafei, age=18)
User(id=2, name=xiaofei, age=17)
User(id=10, name=qianqi, age=10)
User(id=3, name=zhongfei, age=16)
User(id=4, name=laofei, age=15)
User(id=5, name=feifei, age=14)
```

解析：

1：jsonItemReader() flatItemReader() 定义2个读入操作，分别读json格式跟普通文本格式

2：parallelJob() 配置job，需要指定并行的flow步骤，先是parallelFlow1然后是parallelFlow2 ， 2个步骤间使用**.split(new SimpleAsyncTaskExecutor())** 隔开，表示线程池开启2个线程，分别处理parallelFlow1， parallelFlow2 2个步骤。

12.3 分区步骤

分区：有划分，区分意思，在SpringBatch 分区步骤讲的是给执行步骤区分上下级。

上级： 主步骤(Master Step)

下级： 从步骤–工作步骤(Work Step)

主步骤是领导，不用干活，负责管理从步骤，从步骤是下属，必须干活。

一个主步骤下辖管理多个从步骤。

注意： 从步骤，不管多小，它也一个完整的Spring Batch 步骤，负责各自的读入、处理、写入等。

分区步骤结构图

分区步骤一般用于海量数据的处理上，其采用是分治思想。主步骤将大的数据划分多个小的数据集，然后开启多个从步骤，要求每个从步骤负责一个数据集。当所有从步骤处理结束，整作业流程才算结束。

分区器

主步骤核心组件，负责数据分区，将完整的数据拆解成多个数据集，然后指派给从步骤，让其执行。

拆分规则由Partitioner分区器接口定制，默认的实现类：**MultiResourcePartitioner**

```
public interface Partitioner {
        Map<String, ExecutionContext> partition(int gridSize);
}
```

Partitioner 接口只有唯一的方法：partition 参数gridSize表示要分区的大小，可以理解为要开启多个worker步骤，返回值是一个Map， 其中key：worker步骤名称， value：worker步骤启动需要参数值，一般包含分区元数据，比如起始位置，数据量等。

分区处理器

主步骤核心组件，统一管理work 步骤， 并给work步骤指派任务。

管理规则由PartitionHandler 接口定义，默认的实现类：**TaskExecutorPartitionHandler**

需求：下面几个文件将数据读入内存

分析：

步骤1：准备数据

user1-10.txt

```
1#dafei#18
2#dafei#18
3#dafei#18
4#dafei#18
5#dafei#18
6#dafei#18
7#dafei#18
8#dafei#18
9#dafei#18
10#dafei#18
```

user11-20.txt

```
11#dafei#18
12#dafei#18
13#dafei#18
14#dafei#18
15#dafei#18
16#dafei#18
17#dafei#18
18#dafei#18
19#dafei#18
20#dafei#18
```

user21-30.txt

```
21#dafei#18
22#dafei#18
23#dafei#18
24#dafei#18
25#dafei#18
```

```
26#dafei#18
27#dafei#18
28#dafei#18
29#dafei#18
30#dafei#18
```

user31-40.txt

```
31#dafei#18
32#dafei#18
33#dafei#18
34#dafei#18
35#dafei#18
36#dafei#18
37#dafei#18
38#dafei#18
39#dafei#18
40#dafei#18
```

user41-50.txt

```
41#dafei#18
42#dafei#18
43#dafei#18
44#dafei#18
45#dafei#18
46#dafei#18
47#dafei#18
48#dafei#18
49#dafei#18
50#dafei#18
```

步骤2：准备实体类

```
@Getter
@Setter
@ToString
public class User {
    private Long id;
    private String name;
    private int age;
}
```

步骤3：配置分区逻辑

```
public class UserPartitioner  implements Partitioner {
    @Override
    public Map<String, ExecutionContext> partition(int gridSize) {
        Map<String, ExecutionContext> result = new HashMap<>(gridSize);

        int range = 10; //文件间隔
        int start = 1; //开始位置
        int end = 10;  //结束位置
        String text = "user%s-%s.txt";

        for (int i = 0; i < gridSize; i++) {
            ExecutionContext value = new ExecutionContext();
            Resource resource = new ClassPathResource(String.format(text, s
            try {
```

```
                value.putString("file", resource.getURL().toExternalForm())
            } catch (IOException e) {
                e.printStackTrace();
            }
            start += range;
            end += range;

            result.put("user_partition_" + i, value);
        }
        return result;
    }
}
```

步骤4：全部代码

```
package com.langfeiyes.batch._37_step_part;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.EnableBatchP
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.configuration.annotation.StepScope;
import org.springframework.batch.core.partition.PartitionHandler;
import org.springframework.batch.core.partition.support.MultiResourcePartit
import org.springframework.batch.core.partition.support.TaskExecutorPartiti
import org.springframework.batch.item.ExecutionContext;
import org.springframework.batch.item.ItemWriter;
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.builder.FlatFileItemReaderBuilde
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;
import org.springframework.core.task.SimpleAsyncTaskExecutor;

import java.util.List;

@SpringBootApplication
@EnableBatchProcessing
public class PartStepJob {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;



    //每个分区文件读取
    @Bean
    @StepScope
    public FlatFileItemReader<User> flatItemReader(@Value("#{stepExecutionC
        return new FlatFileItemReaderBuilder<User>()
                .name("userItemReader")
                .resource(resource)
                .delimited().delimiter("#")
```

```
                .names("id", "name", "age")
                .targetType(User.class)
                .build();
    }

    @Bean
    public ItemWriter<User> itemWriter(){
        return new ItemWriter<User>() {
            @Override
            public void write(List<? extends User> items) throws Exception
                items.forEach(System.err::println);
            }
        };
    }


    //文件分区器-设置分区规则
    @Bean
    public UserPartitioner userPartitioner(){
        return new UserPartitioner();
    }

    //文件分区处理器-处理分区
    @Bean
    public PartitionHandler userPartitionHandler() {
        TaskExecutorPartitionHandler handler = new TaskExecutorPartitionHar
        handler.setGridSize(5);
        handler.setTaskExecutor(new SimpleAsyncTaskExecutor());
        handler.setStep(workStep());
        try {
            handler.afterPropertiesSet();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return handler;
    }

    //每个从分区操作步骤
    @Bean
    public Step workStep() {
        return stepBuilderFactory.get("workStep")
                .<User, User>chunk(10)
                .reader(flatItemReader(null))
                .writer(itemWriter())
                .build();
    }

    //主分区操作步骤
    @Bean
    public Step masterStep() {
        return stepBuilderFactory.get("masterStep")
                .partitioner(workStep().getName(),userPartitioner())
                .partitionHandler(userPartitionHandler())
                .build();
    }


    @Bean
    public Job partJob(){
        return jobBuilderFactory.get("part-step-job")
```

```
                .start(masterStep())
                .build();
    }
    public static void main(String[] args) {
        SpringApplication.run(PartStepJob.class, args);
    }
}
```

结果：

```
User(id=31, name=dafei, age=18)
User(id=32, name=dafei, age=18)
User(id=33, name=dafei, age=18)
User(id=34, name=dafei, age=18)
User(id=35, name=dafei, age=18)
User(id=36, name=dafei, age=18)
User(id=37, name=dafei, age=18)
User(id=38, name=dafei, age=18)
User(id=39, name=dafei, age=18)
User(id=40, name=dafei, age=18)
User(id=41, name=dafei, age=18)
User(id=42, name=dafei, age=18)
User(id=43, name=dafei, age=18)
User(id=44, name=dafei, age=18)
User(id=45, name=dafei, age=18)
User(id=46, name=dafei, age=18)
User(id=47, name=dafei, age=18)
User(id=48, name=dafei, age=18)
User(id=49, name=dafei, age=18)
User(id=50, name=dafei, age=18)
User(id=21, name=dafei, age=18)
User(id=22, name=dafei, age=18)
User(id=23, name=dafei, age=18)
User(id=24, name=dafei, age=18)
User(id=25, name=dafei, age=18)
User(id=26, name=dafei, age=18)
User(id=27, name=dafei, age=18)
User(id=28, name=dafei, age=18)
User(id=29, name=dafei, age=18)
User(id=30, name=dafei, age=18)
User(id=1, name=dafei, age=18)
User(id=2, name=dafei, age=18)
User(id=3, name=dafei, age=18)
User(id=4, name=dafei, age=18)
User(id=5, name=dafei, age=18)
User(id=6, name=dafei, age=18)
User(id=7, name=dafei, age=18)
User(id=8, name=dafei, age=18)
User(id=9, name=dafei, age=18)
User(id=10, name=dafei, age=18)
User(id=11, name=dafei, age=18)
User(id=12, name=dafei, age=18)
User(id=13, name=dafei, age=18)
User(id=14, name=dafei, age=18)
User(id=15, name=dafei, age=18)
User(id=16, name=dafei, age=18)
User(id=17, name=dafei, age=18)
User(id=18, name=dafei, age=18)
```

```
User(id=19, name=dafei, age=18)
User(id=20, name=dafei, age=18)
```

解析：核心点

1>文件分区器：userPartitioner()， 分别加载5个文件进入到程序

2>文件分区处理器：userPartitionHandler() ，指定要分几个区，由谁来处理

3>分区从步骤：workStep() 指定读逻辑与写逻辑

4>分区文件读取：flatItemReader()，需要传入Resource对象，这个对象在userPartitioner()已经标记为file

5>分区主步骤：masterStep() ，指定分区名称与分区器，指定分区处理器

十三、综合案例

到这，整个Spring Batch 教程知识点就全部讲完了，接下来就使用一个综合案例将讲过核心知识串联起来，再来回顾一遍。

13.1 案例需求

1>先动态生成50w条员工数据，存放在employee.csv文件中

2>启动作业异步读取employee.csv文件，将读到数据写入到employee_temp表，要求记录读与写消耗时间

3>使用分区的方式将employee_temp表的数据读取并写入到employee表

13.2 分析

上面需求存在一定连贯性，为了操作简单，使用springMVC项目， 每一个需求对应一个接口：

1：发起 **/dataInit** 初始化50w数据进入employee.csv文件

使用技术点：SpringMVC IO

2：发起**/csvToDB** 启动作业，将employee.csv 数据写入employee_temp表, 记录读与写消耗时间

使用技术点：SpringMVC ItemReader JobExecutionListener

ItemWriter (如果使用Mybatis框架 MyBatisBatchItemWriter/MyBatisPagingItemReaderReader)

3：发起**/dbToDB** 启动作业，将employee_temp数据写入employee表

使用技术点：SpringMVC ItemReader partitioner

ItemWriter(如果使用Mybatis框架：
MyBatisBatchItemWriter/MyBatisPagingItemReaderReader)

13.3 项目准备

步骤**1**：新开**spring-batch-example**

步骤**2**：导入依赖

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.3</version>
    <relativePath/>
</parent>
<properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-batch</artifactId>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.12</version>
    </dependency>


    <dependency>
        <groupId>org.mybatis.spring.boot</groupId>
        <artifactId>mybatis-spring-boot-starter</artifactId>
        <version>1.3.2</version>
    </dependency>
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>druid-spring-boot-starter</artifactId>
        <version>1.1.14</version>
    </dependency>

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
</dependencies>
```

步骤**3**：配置文件

```
spring:
  datasource:
    username: root
    password: admin
    url: jdbc:mysql://127.0.0.1:3306/springbatch?serverTimezone=GMT%2B8&use
    driver-class-name: com.mysql.cj.jdbc.Driver
    # 初始化数据库，文件在依赖jar包中
  sql:
    init:
      schema-locations: classpath:org/springframework/batch/core/schema-mys
      #mode: always
      mode: never
  batch:
    job:
      enabled: false

  druid:
    # 连接池配置
    #初始化连接池的连接数量 大小，最小，最大
    initial-size: 10
    min-idle: 10
    max-active: 20
    #配置获取连接等待超时的时间
    max-wait: 60000
    #配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒
    time-between-eviction-runs-millis: 60000
    # 配置一个连接在池中最小生存的时间，单位是毫秒
    min-evictable-idle-time-millis: 30000
    validation-query: SELECT 1 FROM DUAL
    test-while-idle: true
    test-on-borrow: true
    test-on-return: false
    # 是否缓存preparedStatement，也就是PSCache  官方建议MySQL下建议关闭    个人建议
    pool-prepared-statements: false
    max-pool-prepared-statement-per-connection-size: 20

mybatis:
  configuration:
    default-executor-type: batch


job:
  data:
    path: D:/spring-batch-example/
```

步骤**4**：建立**employee**表与**employe_temp**表

```
CREATE TABLE `employee` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL,
  `age` int DEFAULT NULL,
  `sex` int DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
```

```
CREATE TABLE `employee_temp` (
  `id` int NOT NULL AUTO_INCREMENT,
```

```
  `name` varchar(255) DEFAULT NULL,
  `age` int DEFAULT NULL,
  `sex` int DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
```

步骤**5**：建立基本代码体系**-domain-mapper-service-controller-mapper.xml**

domain

```
@Setter
@Getter
@ToString
public class Employee {
    private Long id;
    private String name;
    private int age;
    private int sex;
}
```

mapper.java

```
public interface EmployeeMapper {

    /**
     * 添加
     */
    int save(Employee employee);
}
```

service接口

```
public interface IEmployeeService {
    /**
     * 保存
     */
    void save(Employee employee);
}
```

service接口实现类

```
@Service
public class EmployeeServiceImpl implements IEmployeeService {
    @Autowired
    private EmployeeMapper employeeMapper;
    @Override
    public void save(Employee employee) {
        employeeMapper.save(employee);
    }
}
```

启动类

```
@SpringBootApplication
@EnableBatchProcessing
@MapperScan("com.langfeiyes.exp.mapper")
public class App {

    public static void main(String[] args) {
        SpringApplication.run(App.class, args);
    }
}
```

Mapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybati
<mapper namespace="com.langfeiyes.exp.mapper.EmployeeMapper">

    <resultMap id="BaseResultMap" type="com.langfeiyes.exp.domain.Employee'
        <result column="id" jdbcType="INTEGER" property="id" />
        <result column="name" jdbcType="VARCHAR" property="name" />
        <result column="age" jdbcType="VARCHAR" property="age" />
        <result column="sex" jdbcType="VARCHAR" property="sex" />
    </resultMap>


    <insert id="save" keyColumn="id" useGeneratedKeys="true" keyProperty="i
        insert into employee(id, name, age, sex) values(#{id},#{name},#{age
    </insert>
</mapper>
```

13.4 需求一

需求：先动态生成**50w**条员工数据，存放再**employee.csv**文件中

步骤1：定义：DataInitController

```
@RestController
public class DataInitController {

    @Autowired
    private IEmployeeService employeeService;

    @GetMapping("/dataInit")
    public String dataInit() throws IOException {
        employeeService.dataInit();
        return "ok";
    }
}
```

步骤2：在IEmployeeService 添加dataInit 方法

```
public interface IEmployeeService {
    /**
     * 保存
     */
    void save(Employee employee);

    /**
     * 初始化数据：生成50w数据
     */
    void dataInit() throws IOException;
}
```

步骤3：在EmployeeServiceImpl 实现方法

```
@Service
public class EmployeeServiceImpl implements IEmployeeService {
    @Autowired
    private EmployeeMapper employeeMapper;
    @Override
    public void save(Employee employee) {
        employeeMapper.save(employee);
    }

    @Value("${job.data.path}")
    public String path;

    @Override
    public void dataInit() throws IOException {
        File file = new File(path, "employee.csv");
        if (file.exists()) {
            file.delete();
        }
        file.createNewFile();
        FileOutputStream out = new FileOutputStream(file);
        String txt = "";

        Random ageR = new Random();
        Random boolR = new Random();

        // 给文件中生产50万条数据
        long beginTime = System.currentTimeMillis();
        System.out.println("开始时间：【 " + beginTime + " 】");
```

```
        for (int i = 1; i <= 500000; i++) {
            if(i == 500000){
                txt = i+",dafei_"+ i +"," + ageR.nextInt(100) + "," + (bool
            }else{
                txt = i+",dafei_"+ i +"," + ageR.nextInt(100) + "," + (bool
            }

            out.write(txt.getBytes());
            out.flush();
        }
        out.close();
        System.out.println("总共耗时：【 " + (System.currentTimeMillis() - be
    }
}
```

步骤4：访问http://localhost:8080/dataInit 生成数据。

13.5 需求二

需求：启动作业异步读取**employee.csv**文件，将读到数据写入到**employee_temp**表，要求记录读与写消耗时间

步骤1：修改IEmployeeService 接口

```
public interface IEmployeeService {
    /**
     * 保存
     */
    void save(Employee employee);

    /**
     * 初始化数据：生成50w数据
     */
    void dataInit() throws IOException;

    /**
     * 清空数据
     */
    void truncateAll();

    /**
     * 清空employee_temp数据
```

```
     */
    void truncateTemp();
}
```

步骤2：修改EmployeeServiceImpl

```
@Override
public void truncateAll() {
    employeeMapper.truncateAll();
}

@Override
public void truncateTemp() {
    employeeMapper.truncateTemp();
}
```

步骤3：修改IEmployeeMapper.java

```
public interface EmployeeMapper {

    /**
     * 添加
     */
    int save(Employee employee);

    /**
     * 添加临时表
     * @param employee
     * @return
     */
    int saveTemp(Employee employee);

    /**
     * 清空数据
     */
    void truncateAll();

    /**
     * 清空临时表数据
     */
    void truncateTemp();
}
```

步骤4：修改EmployeeMapper.xml

```
<insert id="saveTemp" keyColumn="id" useGeneratedKeys="true" keyProperty="i
    insert into employee_temp(id, name, age, sex) values(#{id},#{name},#{ag
</insert>

<delete id="truncateAll">
    truncate employee
</delete>

<delete id="truncateTemp">
    truncate employee_temp
</delete>
```

步骤5：在com.langfeiyes.exp.job.listener 包新建监听器，用于计算开始结束时间

```
package com.langfeiyes.exp.job.listener;

import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobExecutionListener;


public class CsvToDBJobListener implements JobExecutionListener {

    @Override
    public void beforeJob(JobExecution jobExecution) {
            long begin = System.currentTimeMillis();
            jobExecution.getExecutionContext().putLong("begin", begin);
            System.err.println("-----------------------【CsvToDBJob开始时间
        }

    @Override
    public void afterJob(JobExecution jobExecution) {
                long begin = jobExecution.getExecutionContext().getLong("be
                long end = System.currentTimeMillis();
                System.err.println("-----------------------【CsvToDBJob结
                System.err.println("-----------------------【CsvToDBJob总
            }
}
```

步骤6：在com.langfeiyes.exp.job.config包定义CsvToDBJobConfig配置类

```
package com.langfeiyes.exp.job.config;


import com.langfeiyes.exp.domain.Employee;
import com.langfeiyes.exp.job.listener.CsvToDBJobListener;
import org.apache.ibatis.session.SqlSessionFactory;
import org.mybatis.spring.batch.MyBatisBatchItemWriter;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.builder.FlatFileItemReaderBuilde
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.PathResource;
import org.springframework.core.task.SimpleAsyncTaskExecutor;

import java.io.File;

/**
 * 将数据从csv文件中读取，并写入数据库
 */
@Configuration
public class CsvToDBJobConfig {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;
```

```
    @Autowired
    private SqlSessionFactory sqlSessionFactory;

    @Value("${job.data.path}")
    private String path;

    //多线程读-读文件，使用FlatFileItemReader
    @Bean
    public FlatFileItemReader<Employee> cvsToDBItemReader(){
        FlatFileItemReader<Employee> reader = new FlatFileItemReaderBuilder
                .name("employeeCSVItemReader")
                .saveState(false) //防止状态被覆盖
                .resource(new PathResource(new File(path, "employee.csv").g
                .delimited()
                .names("id", "name", "age", "sex")
                .targetType(Employee.class)
                .build();

        return reader;
    }

    //数据库写-使用mybatis提供批处理读入
    @Bean
    public MyBatisBatchItemWriter<Employee> cvsToDBItemWriter(){
        MyBatisBatchItemWriter<Employee> itemWriter = new MyBatisBatchItemW
        itemWriter.setSqlSessionFactory(sqlSessionFactory); //需要指定sqlses
        //指定要操作sql语句，路径id为：EmployeeMapper.xml定义的sql语句id
        itemWriter.setStatementId("com.langfeiyes.exp.mapper.EmployeeMapper
        return itemWriter;
    }

    @Bean
    public Step csvToDBStep(){
        return stepBuilderFactory.get("csvToDBStep")
                .<Employee, Employee>chunk(10000)  //每个块10000个 共50个
                .reader(cvsToDBItemReader())
                .writer(cvsToDBItemWriter())
                .taskExecutor(new SimpleAsyncTaskExecutor())  //多线程读写
                .build();

    }

    //job监听器
    @Bean
    public CsvToDBJobListener csvToDBJobListener(){
        return new CsvToDBJobListener();
    }

    @Bean
    public Job csvToDBJob(){
        return jobBuilderFactory.get("csvToDB-step-job")
                .start(csvToDBStep())
                .incrementer(new RunIdIncrementer()) //保证可以多次执行
                .listener(csvToDBJobListener())
                .build();

    }
}
```

步骤7：在com.langfeiyes.exp.controller 添加JobController

```
package com.langfeiyes.exp.controller;

import com.langfeiyes.exp.service.IEmployeeService;
import org.springframework.batch.core.*;
import org.springframework.batch.core.explore.JobExplorer;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.Date;

@RestController
public class JobController {

    @Autowired
    private IEmployeeService employeeService;

    @Autowired
    private JobLauncher jobLauncher;

    @Autowired
    private JobExplorer jobExplorer;

    @Autowired
    @Qualifier("csvToDBJob")
    private Job csvToDBJob;

    @GetMapping("/csvToDB")
    public String csvToDB() throws Exception {
        employeeService.truncateTemp(); //清空数据运行多次执行

        //需要多次执行，run.id 必须重写之前，再重构一个新的参数对象
        JobParameters jobParameters = new JobParametersBuilder(new JobParam
                .addLong("time", new Date().getTime())
                .getNextJobParameters(csvToDBJob).toJobParameters();
        JobExecution run = jobLauncher.run(csvToDBJob, jobParameters);
        return run.getId().toString();
    }
}
```

步骤8：访问测试：http://localhost:8080/csvToDB

```
-------------------------【CsvToDBJob开始时间：】---->1670575356773<---------
-------------------------【CsvToDBJob结束时间：】---->1670575510967<---------
-------------------------【CsvToDBJob总耗时：】---->154194<------------------
```

13.6 需求三

需求：使用分区的方式将**employee_temp**表的数据读取并写入到**employee**表

步骤1：在com.langfeiyes.exp.job.config 包添加DBToDBJobConfig， 配置从数据库到数据库的作业

```
package com.langfeiyes.exp.job.config;


import com.langfeiyes.exp.domain.Employee;
import com.langfeiyes.exp.job.partitioner.DBToDBPartitioner;
import org.apache.ibatis.session.SqlSessionFactory;
import org.mybatis.spring.batch.MyBatisBatchItemWriter;
import org.mybatis.spring.batch.MyBatisPagingItemReader;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.configuration.annotation.JobBuilderFa
import org.springframework.batch.core.configuration.annotation.StepBuilderF
import org.springframework.batch.core.configuration.annotation.StepScope;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.core.partition.PartitionHandler;
import org.springframework.batch.core.partition.support.TaskExecutorPartiti
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.task.SimpleAsyncTaskExecutor;

import java.io.File;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * 将数据从employee_temp中读取，并写入employe 表
 */
@Configuration
public class DBToDBJobConfig {
    @Autowired
    private JobBuilderFactory jobBuilderFactory;
    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Autowired
    private SqlSessionFactory sqlSessionFactory;

    public static int PAGESIZE = 1000;   //mybatis分页读取数据，跟chunkSize 一
    public static int RANGE = 10000;  //每个分区读取数据范围(理解为个数)
    public static int GRIDSIZE = 50;  //分区个数


    //读数据-从employee_temp 表读 -- mybatis
    @Bean
    @StepScope
    public MyBatisPagingItemReader<Employee> dBToDBJobItemReader(
            @Value("#{stepExecutionContext[from]}") final Integer from,
            @Value("#{stepExecutionContext[to]}") final Integer to,
            @Value("#{stepExecutionContext[range]}") final Integer range){

        System.out.println("----------MyBatisPagingItemReader开始-----from:
        MyBatisPagingItemReader<Employee> itemReader = new MyBatisPagingIte
        itemReader.setSqlSessionFactory(sqlSessionFactory);
        itemReader.setQueryId("com.langfeiyes.exp.mapper.EmployeeMapper.sel
```

```
        itemReader.setPageSize(DBToDBJobConfig.PAGESIZE);
        Map<String, Object> map = new HashMap<>();
        map.put("from", from);
        map.put("to", to);
        itemReader.setParameterValues(map);

        return itemReader;
    }


    //数据库写- 写入到employee 表中
    @Bean
    public MyBatisBatchItemWriter<Employee> dbToDBItemWriter(){
        MyBatisBatchItemWriter<Employee> itemWriter = new MyBatisBatchItemW
        itemWriter.setSqlSessionFactory(sqlSessionFactory);
        itemWriter.setStatementId("com.langfeiyes.exp.mapper.EmployeeMapper
        return itemWriter;
    }

    //文件分区处理器-处理分区
    @Bean
    public PartitionHandler dbToDBPartitionHandler() {
        TaskExecutorPartitionHandler handler = new TaskExecutorPartitionHan
        handler.setGridSize(DBToDBJobConfig.GRIDSIZE);
        handler.setTaskExecutor(new SimpleAsyncTaskExecutor());
        handler.setStep(workStep());
        try {
            handler.afterPropertiesSet();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return handler;
    }

    //每个从分区操作步骤
    @Bean
    public Step workStep() {
        return stepBuilderFactory.get("workStep")
                .<Employee, Employee>chunk(DBToDBJobConfig.PAGESIZE)
                .reader(dBToDBJobItemReader(null, null, null))
                .writer(dbToDBItemWriter())
                .build();
    }

    @Bean
    public DBToDBPartitioner dbToDBPartitioner(){
        return new DBToDBPartitioner();
    }

    //主分区操作步骤
    @Bean
    public Step masterStep() {
        return stepBuilderFactory.get("masterStep")
                .partitioner(workStep().getName(),dbToDBPartitioner())
                .partitionHandler(dbToDBPartitionHandler())
                .build();
    }

    @Bean
    public Job dbToDBJob(){
```

```
        return jobBuilderFactory.get("dbToDB-step-job")
                .start(masterStep())
                .incrementer(new RunIdIncrementer())
                .build();
    }
}
```

步骤2：修改EmployeeMapper.xml

```
<select id="selectTempForList" resultMap="BaseResultMap">
    select * from employee_temp where id between #{from} and #{to}  limit #
</select>
```

步骤3：在com.langfeiyes.exp.job.partitioner 创建DBToDBPartitioner， 用于分区

```
package com.langfeiyes.exp.job.partitioner;

import com.langfeiyes.exp.job.config.DBToDBJobConfig;
import org.springframework.batch.core.partition.support.Partitioner;
import org.springframework.batch.item.ExecutionContext;

import java.util.HashMap;
import java.util.Map;

public class DBToDBPartitioner implements Partitioner {
    //约定分50个区， 每个区10000个数据
    @Override
    public Map<String, ExecutionContext> partition(int gridSize) {
        String text = "----DBToDBPartitioner---第%s分区-----开始：%s---结束：%

        Map<String, ExecutionContext> map = new HashMap<>();
        int from = 1;
        int to = DBToDBJobConfig.RANGE;
        int range = DBToDBJobConfig.RANGE;

        for (int i = 0; i < gridSize; i++) {
            System.out.println(String.format(text, i, from, to, (to - from
            ExecutionContext context = new ExecutionContext();
            context.putInt("from", from);
            context.putInt("to", to);
            context.putInt("range", range);

            from += range;
            to += range;

            map.put("partition_" + i, context);
        }
        return map;
    }
}
```

步骤4：修改JobController 类

```
@GetMapping("/dbToDB")
public String dbToDB() throws Exception {
    employeeService.truncateAll();
```

```
    JobParameters jobParameters = new JobParametersBuilder(new JobParameter
        .addLong("time", new Date().getTime())
        .getNextJobParameters(dbToDBJob).toJobParameters();
    JobExecution run = jobLauncher.run(dbToDBJob, jobParameters);
    return run.getId().toString();
}
```

步骤8：访问：http://localhost:8080/dbToDB

```
----DBToDBPartitioner---第0分区-----开始：1---结束：10000---数据量：10000------
----DBToDBPartitioner---第1分区-----开始：10001---结束：20000---数据量：10000--
----DBToDBPartitioner---第2分区-----开始：20001---结束：30000---数据量：10000--
----DBToDBPartitioner---第3分区-----开始：30001---结束：40000---数据量：10000--
----DBToDBPartitioner---第4分区-----开始：40001---结束：50000---数据量：10000--
----DBToDBPartitioner---第5分区-----开始：50001---结束：60000---数据量：10000--
----DBToDBPartitioner---第6分区-----开始：60001---结束：70000---数据量：10000--
----DBToDBPartitioner---第7分区-----开始：70001---结束：80000---数据量：10000--
----DBToDBPartitioner---第8分区-----开始：80001---结束：90000---数据量：10000--
----DBToDBPartitioner---第9分区-----开始：90001---结束：100000---数据量：10000-
----DBToDBPartitioner---第10分区-----开始：100001---结束：110000---数据量：1000
----DBToDBPartitioner---第11分区-----开始：110001---结束：120000---数据量：1000
----DBToDBPartitioner---第12分区-----开始：120001---结束：130000---数据量：1000
----DBToDBPartitioner---第13分区-----开始：130001---结束：140000---数据量：1000
----DBToDBPartitioner---第14分区-----开始：140001---结束：150000---数据量：1000
----DBToDBPartitioner---第15分区-----开始：150001---结束：160000---数据量：1000
----DBToDBPartitioner---第16分区-----开始：160001---结束：170000---数据量：1000
----DBToDBPartitioner---第17分区-----开始：170001---结束：180000---数据量：1000
----DBToDBPartitioner---第18分区-----开始：180001---结束：190000---数据量：1000
----DBToDBPartitioner---第19分区-----开始：190001---结束：200000---数据量：1000
----DBToDBPartitioner---第20分区-----开始：200001---结束：210000---数据量：1000
----DBToDBPartitioner---第21分区-----开始：210001---结束：220000---数据量：1000
----DBToDBPartitioner---第22分区-----开始：220001---结束：230000---数据量：1000
----DBToDBPartitioner---第23分区-----开始：230001---结束：240000---数据量：1000
----DBToDBPartitioner---第24分区-----开始：240001---结束：250000---数据量：1000
----DBToDBPartitioner---第25分区-----开始：250001---结束：260000---数据量：1000
----DBToDBPartitioner---第26分区-----开始：260001---结束：270000---数据量：1000
----DBToDBPartitioner---第27分区-----开始：270001---结束：280000---数据量：1000
----DBToDBPartitioner---第28分区-----开始：280001---结束：290000---数据量：1000
----DBToDBPartitioner---第29分区-----开始：290001---结束：300000---数据量：1000
----DBToDBPartitioner---第30分区-----开始：300001---结束：310000---数据量：1000
----DBToDBPartitioner---第31分区-----开始：310001---结束：320000---数据量：1000
----DBToDBPartitioner---第32分区-----开始：320001---结束：330000---数据量：1000
----DBToDBPartitioner---第33分区-----开始：330001---结束：340000---数据量：1000
----DBToDBPartitioner---第34分区-----开始：340001---结束：350000---数据量：1000
----DBToDBPartitioner---第35分区-----开始：350001---结束：360000---数据量：1000
----DBToDBPartitioner---第36分区-----开始：360001---结束：370000---数据量：1000
----DBToDBPartitioner---第37分区-----开始：370001---结束：380000---数据量：1000
----DBToDBPartitioner---第38分区-----开始：380001---结束：390000---数据量：1000
----DBToDBPartitioner---第39分区-----开始：390001---结束：400000---数据量：1000
----DBToDBPartitioner---第40分区-----开始：400001---结束：410000---数据量：1000
----DBToDBPartitioner---第41分区-----开始：410001---结束：420000---数据量：1000
----DBToDBPartitioner---第42分区-----开始：420001---结束：430000---数据量：1000
----DBToDBPartitioner---第43分区-----开始：430001---结束：440000---数据量：1000
----DBToDBPartitioner---第44分区-----开始：440001---结束：450000---数据量：1000
----DBToDBPartitioner---第45分区-----开始：450001---结束：460000---数据量：1000
----DBToDBPartitioner---第46分区-----开始：460001---结束：470000---数据量：1000
----DBToDBPartitioner---第47分区-----开始：470001---结束：480000---数据量：1000
----DBToDBPartitioner---第48分区-----开始：480001---结束：490000---数据量：1000
----DBToDBPartitioner---第49分区-----开始：490001---结束：500000---数据量：1000
```

```
----------MyBatisPagingItemReader开始-----from: 250001  -----to:260000  ---
----------MyBatisPagingItemReader开始-----from: 290001  -----to:300000  ---
----------MyBatisPagingItemReader开始-----from: 80001  -----to:90000  -----
----------MyBatisPagingItemReader开始-----from: 410001  -----to:420000  ---
----------MyBatisPagingItemReader开始-----from: 360001  -----to:370000  ---
----------MyBatisPagingItemReader开始-----from: 230001  -----to:240000  ---
----------MyBatisPagingItemReader开始-----from: 40001  -----to:50000  -----
----------MyBatisPagingItemReader开始-----from: 340001  -----to:350000  ---
----------MyBatisPagingItemReader开始-----from: 450001  -----to:460000  ---
----------MyBatisPagingItemReader开始-----from: 110001  -----to:120000  ---
----------MyBatisPagingItemReader开始-----from: 350001  -----to:360000  ---
----------MyBatisPagingItemReader开始-----from: 50001  -----to:60000  -----
----------MyBatisPagingItemReader开始-----from: 430001  -----to:440000  ---
----------MyBatisPagingItemReader开始-----from: 20001  -----to:30000  -----
----------MyBatisPagingItemReader开始-----from: 120001  -----to:130000  ---
----------MyBatisPagingItemReader开始-----from: 190001  -----to:200000  ---
----------MyBatisPagingItemReader开始-----from: 100001  -----to:110000  ---
----------MyBatisPagingItemReader开始-----from: 470001  -----to:480000  ---
----------MyBatisPagingItemReader开始-----from: 60001  -----to:70000  -----
----------MyBatisPagingItemReader开始-----from: 200001  -----to:210000  ---
```

到这，案例就全部结束了。

Vue学习

1、模板语法

1、模板语法

> 1、插值法 `{{}}` => 作用于标签体
>
> 2、指令语法 `v-bind` 、 `v-if` 、 `v-model` .... => 作用于标签属性

Vue模板语法有两大类：

1、差值语法

2、指令语法

2、el和data的两种写法

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
<div id="app">
    <p>初始化vue的两种写法：{{msg}}</p>
</div>
<script src="../js/vue.js"></script>
<script>
    /*Vue第一种写法*/
    new Vue({
        el:"#app",
        data:{
            msg : "Hello World"
        }
    })
    /*Vue第二种写法*/
    const v = new Vue({
        data:{
            msg: "Hello World"
        }
    })
    v.$mount("#app");


   /*data第一种写法*/
    new Vue({
        el:"#app",
        data:{
```

```
            msg : "Hello World"
        }
    })
    /*data第二种写法*/
    const v = new Vue({
       data(){
           return {
               msg:"Hello World"
           }
       }
    })
    v.$mount("#app");
</script>
</body>
</html>
```

3、MVVM模型

1、M：模型（Model）：对应data中的数据

2、V：视图（View）：模板

3、VM：视图模型（ViewModel）：Vue实例模型

4、Object.defineProperty

```
<script>
    let person = {name:"张赞",address:"南京"}
    Object.defineProperty(person,'age',{
        value:28
    })
    console.log(person)
</script>
/*以上方式的age不能参与遍历    Object.defineProperty */


<script>
                let person = {name:"张赞",address:"南京"}
    /*如果想使用age进行办理操作，可以使用 Object.defineProperty 的参数 enumerable
    Object.defineProperty(person,'age',{
        value:28,
        enumerable:true, //控制属性是否可以枚举，默认值是false
        writable:true, //控制属性是否可以进行修改，默认值是false
        configurable:true //控制属性是否可以进行删除，默认值是false
    })
    for (let personKey in person) {
        console.log("###",person[personKey])
    }
    console.log(person)
</script>


<script>
    let number = 18;
    let person = {
```

```
        name:"zhang",
        address:"nanjing"
    }
    Object.defineProperty(person,"age",{
        //当有人读取persion的age属性时，get函数（getter）就会被调用，且返回值就是ag
        get() {
            console.log("have person get this value：",number)
            return number;
        },
        //当有人设置persion的age属性时，set函数（setter）就会被调用，且会收到修改的
        set(v) {
            console.log("have person set this value：",v)
            number = v
        }
    })
</script>
```

5、数据代理

1、 `vue` 中的数据代理：

通过 `vm` 对象来代理 `data` 对象中属性的操作（读/写）

2、 `vue` 中数据代理的好处:

更加方便的操作 `data` 中的数据

3、基本原理：

通过 `Object.defineProperty()` 把 `data` 对象中所有属性添加到 `vm` 上，

为每一个添加到 `vm` 上的 属性，都执行一个 `getter/setter` .

在 `getter/setter` 内部去操作(读/写) `data` 中对应的属性

```
<script>
let obj = {name:"aaa",age:18}
let obj2 = {name:"bbb",address:"nanjing"}

Object.defineProperty(obj2,'age',{
  get(){
    return obj.age
  },
  set(v) {
    obj.age = v
  }
})
</script>
```

6、事件处理

> 事件的基本使用：

- 使用 `v-on:xxx` 或 `@xxx` 绑定事件，其中 `xxx` 是事件名

- 事件的回调需要配置在methods对象中，最终会在 `vm` 上
- `methods` 中配置的函数，不要使用箭头函数，否则 `this` 就不是 `vm` 了
- `methods` 中配置的函数，都是被 `Vue` 所管理的函数， `this` 的指向是 `vm` 或组件实例对象
- `@click="demo"` 和 `@click="demo($event)"` 效果一致，但是后者可以传参

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<div id="app">
    <button v-on:click="showInfo">点击我进行操作-v-on</button>
    <br>
    <button @click="showInfo">点击我进行操作-@（无参）</button>
    <br>
    <button @click="showInfo1(123)">点击我进行操作-@（有参）</button>
</div>
<script src="../js/vue.js"></script>
<script>
    new Vue({
        el: "#app",
        methods: {
            showInfo() {
                alert("Hello World")
            },
            showInfo1(num) {
                alert(num)
            }
        }
    })
</script>
</body>
</html>
```

7、事件修饰符

> Vue中的事件修饰符：

- `prevent` ：阻止默认事件（常用）
- `stop` ：阻止事件冒泡（常用）
- `once` ：事件只触发一次（常用）
- `capture` ：使用事件的捕获模式
- `self` ：是有 `event.target` 是当前操作的元素时才触发事件
- `passive` ：事件的默认行为立即执行，无需等待时间回调执行完毕

8、键盘事件

1、Vue中常用的案件别名

- 回车 => `enter`

- 删除 => `delete` （捕获删除和退格）
- 退出 => `esc`
- 空格 => `space`
- 换行 => `tab` 特殊，必须配合 `keydown` 使用
- 上 => `up`
- 下 => `down`
- 左 => `left`
- 右 = > `right`

2、 `Vue` 未提供别名的按键

可以使用按键原始额 `key` 值去绑定，单注意要转为 `kebab-case` （短横线命名）

3、系统修饰符（用法特殊）

`crtl` 、 `alt` 、 `shift` 、 `meta`

- 配合 `keyup` 使用：按下修饰键的同时，再按下其他键，随后释放其他键，事件才触发
- 配合 `keydown` 使用：正常触发事件

4、使用 `keyCode` 去指定具体的按键（不推荐）

5、定制按键别名

- `Vue.config.keyCodes.自定义键名` = 键码

9、计算属性

- 定义：要用的属性不存在，要通过已有属性计算得来
- 原理：底层借助了 `Object.defineProperty` 方法提供的 `getter` 和 `setter`
- get函数什么时候执行：
  - 当初次读取时会执行一次
  - 当依赖的数据发生改变时被再次调用
- 优势：与 `methods` 实现相比，内部有缓存机制（复用），效率更高，调试方便
- 备注：
  - 计算属性最终会出在 `vm` 上，直接读取使用即可
  - 如果计算属性要被修改，必须写 `set` 函数去相响应修改，且 `set` 中要引起计算时依赖的数据发生改变。

10、监视属性

> 绑定事件的时候：@xxx="yyy" yyy可以写一些简单的语句

```
>  <button @click="info = !info">切换天气 </button>
> ```

```html
<div id="app">
    <h2>今天天气很{{info}}</h2>
```
```

```
    <button @click="change">切换天气 </button>
    <!--<button @click="info = !info">切换天气 </button>   效果同上-->
</div>
<script src="../js/vue.js"></script>
<script type="text/javascript">
    new Vue({
        el:"#app",
        data:{
            isHot:true
        },
        //计算属性
        computed:{
            info(){
                return this.isHot ? "炎热" : "凉爽"
            }
        },
        // 方法
        methods: {
            change(){
                this.isHot = !this.isHot
            }
        },
    })
</script>
```

空白文档

空白文档

README

WorkFlow

Git 分支规范

> - 活跃项目：近 2 个月内 GitLab 有代码提交
> - 活跃分支：2022-04-01 后有 push 或 merge 的分支

WorkFlow

首先，强烈建议学习一下阮一峰的 **Git** 工作流程，以及文中提到的相关文章。

常见的 WorkFlow 有三种：

- Git Flow - 适用于单个人维护的项目
- GitHub Flow - 适用于比较简单的项目，迭代不频繁、发布环境/方式单一
- GitLab Flow - 能适用于各种场景

根据实际情况，每个团队/项目可以选择上面三种 WorkFlow 之一，不能过于随意。

无论选择上面哪种 WorkFlow，都要注意 保持 **`master`** 到 **`beta`** 、 **`release`** 等分支的同步，尽量使用 merge 而不是 cherry-pick，因为后者一但有遗漏会导致两份代码存在差异。

一些原则：

1. 线上发生 **bug** 时，从发生 **bug** 的 **tag** 开出 **`bugfix`** 分支
   修复后 merge 回原分支以及受影响的其它 `master` 、 `release` 以及 `beta` 等分支
2. **`feature`** 分支一般从 **`release`** 开出
   开发完成之后 merge 回 `release` 分支，上线后通过 `release` 分支 merge 回 `master`
   上线前 `feature` 分支的 bugfix 视为开发过程的一部分，在 `feature` 分支完成
3. **`bugfix`** 、 **`feature`** 等分支一般不再开出子分支
4. 除 **`master`** 以外， **`develop`** 、 **`test`** 、 **`beta`** 、 **`release`** 等其他任何分支不要长期使用
   例如长期持续使用 `test/pro` 、 `beta/pro` 、 `release/pro` 做 pro 环境的测试和发布，会导致它们可能与 `master` 分支存在差异、并且差异日积月累地增多，从而引发不必要的问题，正确的做法是：
5. 每个版本开始时，从 `master` 开出 `test/pro-22.9.1` 、 `beta/pro-22.9.1` 、 `release/pro-22.9.1` 等分支
6. 个人及 feature 分支 merge 到 test

7. 测试通过后从 feature merge 到 beta —— 因为 test 分支可能比较混乱
8. 上线时从 beta merge 到 release —— 确保 beta 环境和线上代码一样
9. 上线之后在 release 分支打 tag（例如 `release/pro-22.9.1` ）、merge 回 `master` 、然后删掉

{width=680px}

分支管理

为了配合 代码检查 以及适应各种 Git WorkFlow，Git 分支需要符合以下规范。

1. 开发分支

- 功能开发、bugfix、代码优化等开发分支应符合以下格式，开发完成后通过 merge request 合并到测试/发布分支
- 要求每个分支只能单一，不要把多个需求、以及与需求不相关的 bugfix 混在一个分支里
- 一般情况下禁止合并自己的 merge request

人员维度：

- 个人分支：`<nickname>/<featrue-name | patch-name>` ，例如 `ming/group-managers`

类型维度：

- 需求分支：`[featrue | fea]/<featrue-name>`，例如 `featrue/group-managers`
- 修复分支：`[patch | hotfix | fix | bugfix | bug | ...]/<patch-name>`，例如 `fix/my-page-columns`

> 这些分支上的 push 可以不触发代码检查，但 无法作为 **merge request** 目标分支。

2. 公共分支

用于测试、发布的保护分支要设置为保护分支：

- root 分支：不带 `/` 的分支，建议使用规范命名，数量不超过 **5** 个，例如 `master` / `main`、`develop`、`test`、`alpha`、`beta`、`release` 等
- 测试/发布分支：带有前缀的分支 `[test | release | stable] / [<version> | <environment> | <customer-name>]`，例如 `release/1.2.0`、`release/latest`、`release/PuXin`、`test/beta`、`test/HuaTu`

> 这些分支上的 **push/merge** 以及 **merge request** 需要触发代码检查，`test/*` 可选。

3. 保护分支设置

> 设置：Project > Settings > Repository > Protected Branches

规范/安全模式（推荐）： > push 操作仅应急时使用 | | Allowed to merge | Allowed to push | | ------- | :---------- | :---------- | | master | Developers + Maintainers | Maintainers （仅用于紧急情况） ~~(ci msg + " - IKNOWWHATIAMDOING")~~ | | develop | Developers + Maintainers | Maintainers （仅用于紧急情况） ~~(ci msg + " - IKNOWWHATIAMDOING")~~ | 快速/粗放模式（**不推荐**）： | | Allowed to merge | Allowed to push | | ------- | :---------- | :---------- | | master | Developers + Maintainers | Developers + Maintainers | | develop | Developers + Maintainers | Developers + Maintainers |

4. 定制分支

少量的可套用上面分支格式，发布分支需要强制检查，但 定制客户较多时建议 **Fork** 到新的仓库（代码可以 push/merge 会原仓库）

5. 历史分支

长期不用的分支建议打 **tag** 之后删除（从旧分支上创建名字符合规范的新分支，然后删除旧分支）

Code Review

在代码合并到 `master`、`release` 或 `beta` 分支时，需要通过 **GitLab** 的 **merge request** 流程进行 **code review**。

- 为了使分支的 graph 保持简洁，发起 merge request 之前最好执行 `git reset --soft` + `git commit` 或者 `git rebase --autosquash`，如果确实需要保留多条提交记录需要在 merge request 的备注里说明
- maintainer 在 merge 代码时，如果发现提交次数不是唯一，需要考虑勾选 `squash commits` 合并提交，并且无论如何都要勾选 `delete source branch` (必选)
- 小功能、bugfix 由 maintainer 独自或者叫相关 developer 一起 review，大块功能需要叫相关同学一起 reivew

> SourceTree 里 Remote 设置 Optional extended integration —— Host Type 选 `GitLab CE`、Host 填 `https://git.baijiashilian.com/`，然后在 push 过的分支上右键即可 Create Pull Request；

版本规范

版本管理规范

README

测试

空白文档

优化

```
ALTER TABLE `pc_monitoring_device_result`
ADD INDEX `del_flag`(`del_flag`) USING BTREE;

ALTER TABLE `pc_monitoring_device_result`
ADD INDEX `ai_model_id`(`ai_model_id`) USING BTREE;

ALTER TABLE `pc_monitoring_device_result`
ADD INDEX `monitoring_device_id`(`monitoring_device_id`) USING BTREE;

ALTER TABLE `pc_monitoring_device_result`
ADD INDEX `alarm_event`(`alarm_event`) USING BTREE;

ALTER TABLE `pc_monitoring_device`
ADD INDEX `del_flag`(`del_flag`) USING BTREE;

ALTER TABLE `pc_ai_model`
ADD INDEX `del_flag`(`del_flag`) USING BTREE;

ALTER TABLE `sys_dept`
ADD INDEX `del_flag`(`del_flag`) USING BTREE;
```

```
alter table `pc_monitoring_device_result`  drop index `del_flag`;
alter table `pc_monitoring_device_result`  drop index `ai_model_id`;
alter table `pc_monitoring_device_result`  drop index `monitoring_device_id
alter table `pc_monitoring_device_result`  drop index `alarm_event`;
alter table `pc_monitoring_device`  drop index `del_flag`;
alter table `pc_ai_model`  drop index `del_flag`;
alter table `sys_dept`  drop index `del_flag`;
```

> 加索引之前

> 加索引之后

空白文档

flyway数据迁移

> 引入 `Flyway`
>
> 目前支持mysql5.7的社区版为7.15.0，支持mysql8.0的版本是8.2.0，8.2.1移除了mysql支持，如文档原 文：
>
> Extract MySQL code to plugin. This will need to be added as a new dependency.
>
> flyway的8.2.1版本移除mysql的解决方案，增加依赖：

```
> <dependency>
>       <groupId>org.flywaydb</groupId>
>       <artifactId>flyway-mysql</artifactId>
> </dependency>
> ```

```xml
<!--数据库迁移工具-->
<dependency>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-mysql</artifactId>  <!--flyway-core 在高版本之后,
    <version>9.21.0</version>
</dependency>
 <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<!-- Flyway插件信息配置 -->
<build>
    <plugins>
        <plugin>
             <!--其他plugin插件-->
        </plugin>

       <!--migration 插件信息    可以在maven的插件中直接点击migrate进行数据填充
          <plugin>
              <groupId>org.flywaydb</groupId>
              <artifactId>flyway-maven-plugin</artifactId>
              <version>5.2.4</version>
              <configuration>
                  <url>jdbc:mysql://localhost:3306/catch_admin?useUnic
                  <user>root</user>
                  <password>admin123</password>
                  <driver>com.mysql.cj.jdbc.Driver</driver>
              </configuration>
          </plugin>
    </plugins>
</build>
```

> 配置信息

```
## flyway数据迁移
spring:
  # 数据源，配置自己的数据库连接配置就好了
```
```

```
  datasource:
    url: jdbc:mysql://127.0.0.1:3306/catch_admin?useUnicode=true&characterE
    username: root
    password: admin123
    driver-class-name: com.mysql.cj.jdbc.Driver
    minPoolSize: 3
    maxPoolSize: 10
    maxLifetime: 20000
    borrowConnectionTimeout: 30
    loginTimeout: 30
    mainteanceInterval: 60
    maxIdleTime: 60
  flyway:
    #是否开启
    enabled: true
    encoding: utf-8
    locations: classpath:db/migration #迁移脚本的位置，默认db/migration
#    baseline-on-migrate: false
#    table: flyway_schema_history # 记录历史记录的表名称
#    out-of-order: false
    # migrate是否校验
#    validate-on-migrate: true

#    baseline-description: #执行基线时标记已有Schema的描述。
#    baseline-on-migrate: false # 在没有元数据表的情况下，针对非空Schema执行迁移时
#    baseline-version: 1 #执行基线时用来标记已有Schema的版本。 （默认值：1。）
#    check-location: false #检查迁移脚本所在的位置是否存在。 （默认值： false 。 ）
#    clean-on-validation-error: false #在验证错误时，是否自动执行清理。 （默认值：
#    enabled: true #开启Flyway。 （默认值： true 。 ）
#    encoding: UTF-8 #设置SQL迁移文件的编码。 （默认值： UTF-8 。 ）
#    ignore-failed-future-migration: false #在读元数据表时，是否忽略失败的后续迁移
#    init-sqls: #获取连接后立即执行初始化的SQL语句。
#    locations: classpath:db/migration #迁移脚本的位置。 （默认值： db/migratio
#    out-of-order: false #是否允许乱序（out of order）迁移。 （默认值： false 。
#    password: #待迁移数据库的登录密码
#    placeholder-prefix: #设置每个占位符的前缀。 （默认值： ${ 。 ）
#    placeholder-replacement: #是否要替换占位符。 （默认值： true 。 ）
#    flyway.placeholder-suffix: #设置占位符的后缀。 （默认值： } 。 ）
#    placeholders: [placeholder name] #设置占位符的值。
#    schemas: #Flyway管理的Schema列表，区分大小写。默认连接对应的默认Schema。
#    sql-migration-prefix: V #SQL迁移的文件名前缀。 （默认值： V 。 ）
#    sql-migration-separator: __ #SQL迁移的文件名分隔符。 （默认值： __ 。 ）
#    sql-migration-suffix: .sql #SQL迁移的文件名后缀。 （默认值： .sql 。 ）
#    table: #Flyway使用的Schema元数据表名称。 （默认值： schema_version 。 ）
#    target: #Flyway要迁移到的目标版本号。 （默认最新版本。 ）
#    url: #待迁移的数据库的JDBC URL。如果没有设置，就使用配置的主数据源。
#    user: #待迁移数据库的登录用户。
#    validate-on-migrate: true #在运行迁移时是否要自动验证。 （默认值： true 。 ）
```

> 放置脚本

- classpath:/db/migration 目录为flyway默认的迁移脚本存放目录，可以通过flyway.locations进行配置
- 迁移脚本文件名称：flyway的实现就是根据判断迁移记录表和迁移脚本文件名称来实现的。对于版本升级需要执行的sql，文件名称必须以“V”开头，后面跟版本号，再跟两个下划线，再跟自定义名称，最后跟“.sql”
- V1.0.1__migration.sql：sql的内容就是在版本1.0.1中新增了一条数据

格式详解

`V{version}_{date}_{num}__{type}_{description}_{Author}.sql`

例如：`V5_1_0_20180914_1__DDL_alter_table_medicinenames_qiaotiansheng`

详细说明：

version : 需求版本号 （2.3.3、2.4.0）

date: 提交日期20180507

num: 开发人员自由命名，格式必须为数字

type: sql文件类型 DML 数据更新(插入、更新、删除); DDL 结构更新; DCL 权限控制;

description: 文件描述

Author: 开发人员姓名

> 注意
>
> 我们在定义脚本的时候，除了 V 字开头的脚本之外，还有一种 R 字开头的脚本。
>
> V 字开头的脚本只会执行一次（脚本文件格式：V1.0.0**init.sql**）。 **R** 字开头的脚本，只要脚本内容发生了变化，启动时候就会执行（脚本文件格式：**R**init.sql 没有版本的概念）。 所有的脚本，一旦执行了，就会在 flyway_schema_history 表中有记录，如果你不小心搞错了，可以手动从 flyway_schema_history 表中删除记录，然后修改 SQL 脚本后再重新启动（生产环境不建议）。

Flyway将每一个数据库脚本称之为：**migrations**，**flyway**支持三种类型的**migration**：

- Versioned migrations：最常用的migration，可以简单的理解为数据库升级脚本
- Undo migrations：数据库版本回退脚本，需要Pro版本，忽略，而且使用过程存在较大风险，undo操作目前只能通过plugin或者command-line来执行
- Repeatable migrations：可重复执行的migration，例如create or replace脚本，当脚本checksums改变时会重新执行

例：brew upgrade git

```
brew -v   #查看brew的版本
brew update   #更新homebrew自己，把所有的Formula目录更新，并且会对本机已经安装并有更
brew -help  #查看命令帮助：
brew outdated  #查看那些已安装的程序需要更新
brew upgrade [包名]  #更新单个软件：
#例：brew upgrade git
brew upgrade  #更新所有软件：
brew install [包名]@版本  #安装软件
#例：brew install git
brew uninstall [包名]  #卸载
#例：brew uninstall git
brew cleanup   #清理所有包的旧版本 （安装包缓存）
#例：brew cleanup -n  #显示要删除的内容，但不要实际删除任何内容
#例：brew cleanup -s  #清理缓存，包括下载即使是最新的版本
#例：brew cleanup --prune=1     #删除所有早于指定时间的缓存文件（天）
brew cleanup [包名] #清理单个软件旧版本
#例：brew cleanup git
brew outdated #查看需要更新的包
brew cleanup -n  #查看可清理的旧版本包，不执行实际操作
brew pin $FORMULA  #锁定某个包
brew unpin $FORMULA  #取消锁定
brew info [包名]  #查看包信息
#例:brew info git
brew list  #查看安装列表
brew search [包名] #查询可用包
#例：brew search git
brew deps [包名] #显示包依赖
#例: brew deps git
```

`brew` 安装 `redis`

```
  jeffrey@jeffreydeMacBook-Pro  /  brew install redis
==> Fetching redis
==> Downloading https:#ghcr.io/v2/homebrew/core/redis/manifests/7.0.11
######################################################################
==> Downloading https:#ghcr.io/v2/homebrew/core/redis/blobs/sha256:d2972b71
==> Downloading from https:#pkg-containers.githubusercontent.com/ghcr1/blob
######################################################################
==> Pouring redis--7.0.11.arm64_monterey.bottle.tar.gz
==> Caveats
To restart redis after an upgrade:
  brew services restart redis
Or, if you don't want/need a background service you can just run:
  /opt/homebrew/opt/redis/bin/redis-server /opt/homebrew/etc/redis.conf
==> Summary
  /opt/homebrew/Cellar/redis/7.0.11: 14 files, 2.7MB
==> Running `brew cleanup redis`...
Disable this behaviour by setting HOMEBREW_NO_INSTALL_CLEANUP.
Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).

######################################################################
# 启动
brew services start redis
```

```
# 重启
brew services restart redis
# 暂停
brew services stop redis
```

`brew` 安装 `mysql`

```
jeffrey@jeffreydeMacBook-Pro  /opt/homebrew/etc   stable  brew install mysq
==> Downloading https://formulae.brew.sh/api/formula.jws.json
#=O#-     #        #
==> Downloading https://formulae.brew.sh/api/cask.jws.json
#=#=-  #       #
==> Fetching dependencies for mysql: icu4c, libevent, libcbor, libfido2, lz
==> Fetching icu4c
==> Downloading https://ghcr.io/v2/homebrew/core/icu4c/manifests/72.1
..................
..................
..................
=> Summary
  /opt/homebrew/Cellar/python@3.11/3.11.3: 3,178 files, 62.1MB
==> Running `brew cleanup python@3.11`...
Removing: /opt/homebrew/Cellar/python@3.11/3.11.2_1... (3,178 files, 62.2MB
==> Checking for dependents of upgraded formulae...
==> No broken dependents found!
==> Caveats
==> mysql
We've installed your MySQL database without a root password. To secure it r
    mysql_secure_installation

MySQL is configured to only allow connections from localhost by default

To connect run:
    mysql -u root

To restart mysql after an upgrade:
  brew services restart mysql
Or, if you don't want/need a background service you can just run:
  /opt/homebrew/opt/mysql/bin/mysqld_safe --datadir=/opt/homebrew/var/mysql
==> python@3.11
Python has been installed as
  /opt/homebrew/bin/python3

Unversioned symlinks `python`, `python-config`, `pip` etc. pointing to
`python3`, `python3-config`, `pip3` etc., respectively, have been installed
  /opt/homebrew/opt/python@3.11/libexec/bin

You can install Python packages with
  pip3 install <package>
They will install into the site-package directory
  /opt/homebrew/lib/python3.11/site-packages

tkinter is no longer included with this formula, but it is available separa
  brew install python-tk@3.11

gdbm (`dbm.gnu`) is no longer included in this formula, but it is available
  brew install python-gdbm@3.11
`dbm.ndbm` changed database backends in Homebrew Python 3.11.
If you need to read a database from a previous Homebrew Python created via
```

```
you'll need to read your database using the older version of Homebrew Pytho
`dbm` still defaults to `dbm.gnu` when it is installed.

For more information about Homebrew and Python, see: https://docs.brew.sh/H
 jeffrey@jeffreydeMacBook-Pro  /opt/homebrew/etc   stable
```

springboot多环境切换

1、创建配置文件

- application.yml
- application-dev.yml
- application-prod.yml

```
spring:
  profiles:
    active: @profileActive@
```

2、配置

pom.xml

```
<build>
    <resources>
             <resource>
                <directory>src/main/resources</directory>
                <filtering>true</filtering>
                <excludes>
                    <exclude>application*.yml</exclude>
                </excludes>
            </resource>
            <resource>
                <directory>src/main/resources</directory>
                <!-- 是否替换@xx@表示的maven properties属性值 -->
                <filtering>true</filtering>
                <includes>
                    <include>application.yml</include>
                    <include>application-${profileActive}.yml</include>
                </includes>
            </resource>
    </resources>
</build>


<profiles>
    <profile>
        <id>dev</id>
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
        <properties>
            <profileActive>dev</profileActive>
            <java_opts>-server -Xms256m -Xmx256m -XX:NewSize=128m -XX:MaxNe
        </properties>
    </profile>
    <!--生产环境-->
    <profile>
        <id>prod</id>
        <properties>
            <profileActive>prod</profileActive>
            <java_opts>
                <!--                    -server -server -Xmx12g -Xms12g -Xs
                <!--                    -XX:SurvivorRatio=8 -XX:+UseG1GC -X
```

```
                <!--                    -XX:+UnlockExperimentalVMOptions -X
                <!--                    -XX:+ParallelRefProcEnabled -Xloggc
                <!--                    -XX:+PrintGCTimeStamps -XX:NumberOf
                <!--                    -XX:ParallelGCThreads=4 -XX:ConcGCT
            </java_opts>
        </properties>
    </profile>
</profiles>
```

临时Git修改记录

AiTaskService

```
package com.ruoyi.pc.service;

import com.ruoyi.pc.domain.*;

import java.util.List;

public interface AiTaskService {

    /**
     * 处理消息
     *
     * @param message
     */
    public void processMessage(String message);

    public void saveMonitoringDeviceResult(String createDate, List<ModelRes

    public void publishAlarmInformation(PcMonitoringDeviceResult monitoring

    CheckResultAlarmRule checkResultRule(PcMonitoringDeviceModel pcMonitori
}
```

public class AiTaskServiceImpl implements AiTaskService

```
package com.ruoyi.pc.service.impl;


import cn.hutool.core.collection.CollectionUtil;
import cn.hutool.json.JSONUtil;
import com.google.gson.Gson;
import com.google.gson.JsonSyntaxException;
import com.ruoyi.common.config.RuoYiConfig;
import com.ruoyi.common.config.ServerConfig;
import com.ruoyi.common.constant.Constants;
import com.ruoyi.common.core.domain.entity.SysDept;
import com.ruoyi.common.utils.StringUtils;
import com.ruoyi.common.websocket.WebSocketServer;
import com.ruoyi.pc.config.DeviceMqtt;
import com.ruoyi.pc.config.redis.RedisUtil;
import com.ruoyi.pc.domain.*;
import com.ruoyi.pc.nodeserver.XbsNodeService;
import com.ruoyi.pc.service.*;
import com.ruoyi.pc.task.util.Location;
import com.ruoyi.pc.task.util.clustering.CentralPoint;
import com.ruoyi.pc.task.util.clustering.ClusteringUtils;
import com.ruoyi.pc.task.util.clustering.RectangleUtil;
import com.ruoyi.pc.util.*;
import com.ruoyi.system.service.ISysConfigService;
import com.ruoyi.system.service.ISysDeptService;
import com.ruoyi.system.service.ISysUserService;
import com.vividsolutions.jts.geom.Geometry;
import com.vividsolutions.jts.geom.GeometryFactory;
```

```
import net.sf.json.JSONArray;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;
import org.springframework.stereotype.Service;

import javax.annotation.Resource;
import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.math.BigDecimal;
import java.util.*;
import java.util.List;
import java.util.concurrent.ConcurrentHashMap;
import java.util.stream.Collectors;

/**
 * agent控制服务
 */
@Service("aiTaskService")
public class AiTaskServiceImpl implements AiTaskService {
    public static Logger logger = LoggerFactory.getLogger(AiTaskServiceImpl

    Gson gson = new Gson();

    @Resource
    ThreadPoolTaskExecutor pool;
    @Autowired
    RedisUtil redisUtil;
    @Autowired
    ISysConfigService configService;
    @Autowired
    IPcMonitoringDeviceResultService iPcMonitoringDeviceResultService;
    @Autowired
    IPcAlarmConfigService pcAlarmConfigService;
    @Autowired
    IPcMonitoringDeviceVolumeService iPcMonitoringDeviceVolumeService;
    @Autowired
    AgentService agentService;
    @Autowired
    XbsNodeService xbsNodeService;
    @Autowired
    IPcPlayRecordService playRecordService;
    @Autowired
    IPcMonitoringTaskService taskService;

    @Resource
    private ISysDeptService sysDeptService;

    @Autowired
    DeRecognition deRecognition;

    @Resource
    private IPcDataReportConfigService pcDataReportConfigService;

    @Resource
```

```
    private ISysUserService userService;

    @Resource
    private WebSocketServer webSocketServer;

    @Autowired
    private ServerConfig serverConfig;

    /**
     * 处理消息
     *
     * @param message
     */
    public void processMessage(String message) {
        // 1、获取创建时间、图片地址、通道信息、关联模型信息、格式化后的识别结果
        AiMessage aiMessage = null;
        try {
            aiMessage = gson.fromJson(message, AiMessage.class);
        } catch (JsonSyntaxException e) {
            logger.error("格式转换错误" + e);
            return;
        }
        Long taskId = aiMessage.getTaskId();
        String createDate = aiMessage.getCreateDate();
        logger.info("任务ID--------" + taskId + ",发布时间--------" + createD
        String imgPath = aiMessage.getImgPath();
        PcMonitoringDevice pcMonitoringDevice = aiMessage.getPcMonitoringDe
        PcMonitoringDeviceModel pcMonitoringDeviceModel = aiMessage.getPcMo
        List<ModelResult> modelResults = aiMessage.getModelResults();
        List<ModelResult> coincideModelResults = new ArrayList<>();
        CheckResultAlarmRule checkResultAlarmRule = checkResultRule(pcMonit
        //3、判断当前通道当前模型是否可以报警
        if (checkResultAlarmRule.isAlarm()) {
            ArrayList<Location> locations = checkResultAlarmRule.getLocatio
            boolean flag = true;
            //获取当前模型的监控任务时长
            //判断是否开启去重
            if (pcMonitoringDeviceModel.getEnableDeduplication() != null &&
                    && pcMonitoringDeviceModel.getDeduplicationMode() != nu
                    && pcMonitoringDeviceModel.getDeduplicationDuration() !
                if (Objects.equals(pcMonitoringDeviceModel.getEnableDedupli
                    if (!(pcMonitoringDeviceModel.getType().equals(URLConst
                        //进行告警去重，如果告警信息一致则直接过滤，不一致则进行监测
                        if (Objects.equals(pcMonitoringDeviceModel.getDedup
                            //过滤数量
                            flag = deRecognition.quantityDetection(taskId,
                        } else if (Objects.equals(pcMonitoringDeviceModel.g
                            //过滤坐标
                            flag = deRecognition.coordinateDetection(taskId
                        } else if (Objects.equals(pcMonitoringDeviceModel.g
                            //去重识别
                            flag = deRecognition.deduplicationIdentificatio
                        }
                    }
                }
            if (!flag) {
                return;
            }
            //4、判断是否设置监测时长以及是否可以报警
            TimeThresholdInfo timeThresholdInfo = isTimeThreshold(createDat
```

```
            if (timeThresholdInfo.isFlag()) {
                //结果坐标
                ArrayList<Location> newLocations = timeThresholdInfo.getLoc
                //模型框图坐标
                List<ModelResult> newModelResults = timeThresholdInfo.getMo
                // 5、保存告警信息
                pool.execute(() -> {
                    SysDept sysDept = sysDeptService.selectDeptById(pcMonit
                    //生成水印内容
                    List<String> markContents = new ArrayList<String>();
                    markContents.add("模型名称：" + pcMonitoringDeviceModel.
                    markContents.add("监控设备名称：" + pcMonitoringDevice.ge
                    markContents.add("告警信息：" + pcMonitoringDeviceModel.
                    markContents.add("告警时间：" + createDate);
                    markContents.add("部门名称：" + sysDept.getDeptName());
                    //5.1、根据坐标截图保存，并返回框图地址
                    String frameImgPath = transformImg(newLocations, timeTh
                    if (null == frameImgPath || frameImgPath.trim().length(
                        return;
                    }
                    //5.2、保存告警信息,包括框图访问地址
                    saveMonitoringDeviceResult(createDate, newModelResults,
                });
                //判断当前通道模型的识别结果是否进行播报：0：播报，1：不播报
                if (pcMonitoringDeviceModel.getSoundColumnAlarmStatus() !=
                    return;
                }
                // 6、音箱播控
                pool.execute(new Runnable() {
                    @Override
                    public void run() {
                        //6.1、根据通道ID，获取关联音柱设备列表
                        List<PcDevice> pcList = iPcMonitoringDeviceVolumeSe
                        //6.2、判断是否播放音箱设备，播放模型对应的音频文件
                        List<PcDevice> xbsPcList = new ArrayList<>(); //有线
                        List<PcDevice> xxyPcList = new ArrayList<>(); //无线
                        for (PcDevice pcDevice : pcList) {
                            if (null != pcDevice && pcDevice.getType().equa
                                xbsPcList.add(pcDevice);
                            }
                            if (null != pcDevice && pcDevice.getType().equa
                                xxyPcList.add(pcDevice);
                            }
                        }
                        //6.3.1、有线音柱将设备全部播放
                        if (xbsPcList.size() > 0) {
                            try {
                                //8.12.1、有线音柱将设备全部停止
                                xbsNodeService.stopPcDeviceByAep(xbsPcList)
                                //8.12.2、有线音柱开始播放设备
                                xbsNodeService.palyPcDeviceByAep(xbsPcList,
                            } catch (Exception e) {
                                logger.error("有线音柱播放失败" + e);
                            }
                        }
                        //6.3.2、无线音柱将设备全部播放
                        if (xxyPcList.size() > 0) {
                            try {
                                // 加入无线音柱操作列表
                                ArrayList<DeviceMqtt> devices = new ArrayLi
```

```
                                for (int i = 0; i < xxyPcList.size(); i++)
                                    PcDevice pcDevice = xxyPcList.get(i);
                                    DeviceMqtt deviceMqtt = new DeviceMqtt(
                                    deviceMqtt.setTask_id("2021010112003712
                                    deviceMqtt.setVol(Integer.valueOf(pcDev
                                    devices.add(deviceMqtt);
                                }
                                //8.12、开始播放无线音柱
                                agentService.setPlayService(pcMonitoringDev
                            } catch (Exception e) {
                                logger.error("无线音柱播放失败" + e);
                            }
                        }
                        //6.4、记录音箱设备播放记录
                        playRecordService.insertPcPlayRecord(pcMonitoringDe
                    }
                });
            }
        }
    }

    /**
     * 检测结果告警规则
     * @param pcMonitoringDeviceModel pcMonitoringDeviceModel
     * @param modelResults List<ModelResult>
     * @return CheckResultAlarmRule
     */
    public CheckResultAlarmRule checkResultRule(PcMonitoringDeviceModel pcM
        CheckResultAlarmRule checkResultAlarmRule = new CheckResultAlarmRul
        List<ModelResult> coincideModelResults = new ArrayList<>();
        ArrayList<Location> locations = null;
        boolean isAlarm = false; //默认不保存告警信息
        // 2、 当模型比较类型是大于等于阈值时
        if (pcMonitoringDeviceModel.getType().equals(URLConstant.MODEL_TYPE
            // 2.1、获取模型识别结果与通道框选区域重合的识别结果列表
            coincideModelResults = getCoincideModelResults(modelResults, pc
            // 2.2、判断重合的模型识别结果列表长度是否大于等于告警目标数阈值
            if (coincideModelResults.size() >= Integer.parseInt(pcMonitorin
                // 2.3、若是，继续格式化模型识别结果列表为框图Location列表
                locations = getLocations(coincideModelResults);
                // 2.4、定制算法判断（聚集算法、其他算法）并更新框图Location列表
                // 2.4.1、聚集算法告警判断，默认告警
                //2.4.2、其他算法
                isAlarm = isGather(pcMonitoringDeviceModel, locations);
            }
        }
        if (pcMonitoringDeviceModel.getType().equals(URLConstant.MODEL_TYPE
            // 2.1、获取模型识别结果与通道框选区域重合的识别结果列表
            coincideModelResults = getCoincideModelResults(modelResults, pc
            // 2.2、判断重合的模型识别结果列表长度是否大于等于告警目标数阈值
            if (coincideModelResults.size() >= Integer.parseInt(pcMonitorin
                // 2.3、若是，继续格式化模型识别结果列表为框图Location列表
                locations = getLocations(coincideModelResults);
                // 2.4、定制算法判断（聚集算法、其他算法）并更新框图Location列表
                // 2.4.1、聚集算法告警判断，默认告警
                //2.4.2、其他算法
                isAlarm = isGather(pcMonitoringDeviceModel, locations);
            }
        }
        // 2、 当模型比较类型是小于等于阈值（包含识别没有结果）时
```

```
        if (pcMonitoringDeviceModel.getType().equals(URLConstant.MODEL_TYPE
            logger.info("----------------------start------------------{}"
            List<ModelResult> modelResultList = new ArrayList<>();
            for (ModelResult modelResult : modelResults) {
                if (modelResult.getBbox().size() > 0 && modelResult.getScor
                    modelResultList.add(modelResult);
                }
            }
            if (modelResultList.size() > 0) {
                Map<String, List<ModelResult>> coincideModelResultsType = n
                JSONArray coordinates = JSONArray.fromObject(pcMonitoringDe
                if (coordinates.size() == 0) {
                    coincideModelResultsType.put("0", modelResultList);
                } else {
                    coincideModelResultsType = getCoincideModelResultsType(
                }
                logger.error("--------------coincideModelResultsType======
                boolean flag = false;
                String departure_model_type = configService.selectConfigByK
                if (CollectionUtil.isNotEmpty(coincideModelResultsType)) {
                    if (Objects.equals(departure_model_type, "0")) {
                        flag = coincideModelResultsType.entrySet().stream()
                        logger.error("--------------coincideModelResultsTy
                    } else if (Objects.equals(departure_model_type, "1")) {
                        flag = coincideModelResultsType.entrySet().stream()
                        logger.error("--------------coincideModelResultsTy
                    } else if (Objects.equals(departure_model_type, "2")) {
                        flag = coincideModelResultsType.values().stream().m
                        logger.error("--------------coincideModelResultsTy
                    }
                }
                isAlarm = !flag;
            } else {
                isAlarm = true;
            }
            locations = new ArrayList<>();
            logger.info("----------------------end------------------");
        }
        checkResultAlarmRule.setAlarm(isAlarm);
        checkResultAlarmRule.setLocations(locations);
        checkResultAlarmRule.setModelResults(coincideModelResults);
        return checkResultAlarmRule;
    }

    /**
     * 获取模型识别结果与通道框选区域重合的识别结果列表
     *
     * @param modelResults
     * @param monitoringDeviceModel
     * @return
     */
    private List<ModelResult> getCoincideModelResults(List<ModelResult> mod

        List<ModelResult> coincideModelResults = new ArrayList<ModelResult>
        // 1、判断是否标记了识别范围，若是，筛选识别结果和通道框选区域重合的识别结果数据
        // 2、获取识别区域框选范围坐标集合
        // coordinates: [{"oX":151,"oY":108},{"oX":417,"oY":130},{"oX":376,
        JSONArray coordinates = JSONArray.fromObject(monitoringDeviceModel.
//        logger.info("识别区域框选范围"+coordinates.toString());
        if (coordinates.size() > 0) {
```

```
            List<PolygonUtil.Polygon> polygonList = new ArrayList<>();
            for (Object coordinateArrays : coordinates) {
                JSONArray coordinateList = JSONArray.fromObject(coordinateA
                if (coordinateList.size() > 0) {
                    ArrayList<PolygonUtil.Point> coordinatePoints = new Arr
                    for (int i = 0; i < coordinateList.size(); i++) {
                        Coordinate coordinate = gson.fromJson(coordinateLis
                        coordinatePoints.add(new PolygonUtil.Point(coordina
                    }
                    PolygonUtil.Polygon coordinatePolygon = new PolygonUtil
                    polygonList.add(coordinatePolygon);
                }
            }
            // 3、 遍历识别结果集合，判断识别结果是否和通道框选区域重合，若是，添加jso
            // array: [{ "category": "smoke", "bbox": [182.2, 218.1, 211.2,
            for (ModelResult modelResult : modelResults) {
                List<Double> bbox = modelResult.getBbox(); // 获取框图位置信息
                PolygonUtil.Polygon locationPolygon = new PolygonUtil.Polyg
                        new PolygonUtil.Point(bbox.get(0), bbox.get(1)),
                        new PolygonUtil.Point(bbox.get(2), bbox.get(1)),
                        new PolygonUtil.Point(bbox.get(2), bbox.get(3)),
                        new PolygonUtil.Point(bbox.get(0), bbox.get(3))
                ));
                //判断多边形是否相交
                for (PolygonUtil.Polygon polygonTest : polygonList) {
                    //判断多边形是否相交
//                    boolean coinCide = PolygonUtil.intersectionJudgment(l
                    String area = getAreaProportion(locationPolygon, polygo
//                    String area = "";
                    logger.info("框图坐标->{}，识别结果坐标->{},面积占比大小：{}"
                    //判断设置是在框外还是框内 : 0:框内，1：框外
                    if (monitoringDeviceModel.getBoxStatus() == null || mon
                        //判断当前面积是否大于设置阈值
                        if (Double.parseDouble(area) > 0 && new BigDecimal(
                            coincideModelResults.add(modelResult);
                            break;
                        }
                    } else {
                        if (new BigDecimal(area).compareTo(new BigDecimal(S
                            coincideModelResults.add(modelResult);
                            break;
                        }
                    }
                }
            }
        } else {
            coincideModelResults.addAll(modelResults);
        }
        return coincideModelResults;
    }

    @SuppressWarnings("all")
    private Map<String, List<ModelResult>> getCoincideModelResultsType(List
        Map<String, List<ModelResult>> coincideModelResults = new HashMap<>
        // 1、判断是否标记了识别范围，若是，筛选识别结果和通道框选区域重合的识别结果数据
        // 2、获取识别区域框选范围坐标集合
        // coordinates: [[{"oX":151,"oY":108},{"oX":417,"oY":130},{"oX":376
        JSONArray coordinates = JSONArray.fromObject(monitoringDeviceModel.
        List<PolygonUtil.Polygon> polygonList = new ArrayList<>();
        for (Object coordinateArrays : coordinates) {
```

```
            JSONArray coordinateList = JSONArray.fromObject(coordinateArray
            if (coordinateList.size() > 0) {
                ArrayList<PolygonUtil.Point> coordinatePoints = new ArrayLi
                for (Object o : coordinateList) {
                    Coordinate coordinate = gson.fromJson(o.toString(), Coo
                    coordinatePoints.add(new PolygonUtil.Point(coordinate.g
                }
                PolygonUtil.Polygon coordinatePolygon = new PolygonUtil.Pol
                polygonList.add(coordinatePolygon);
            }
        }
        logger.info("----------------------start-----------------+++++++++
        for (ModelResult modelResult : modelResults) {
            List<Double> bbox = modelResult.getBbox(); // 获取框图位置信息
            PolygonUtil.Polygon locationPolygon = new PolygonUtil.Polygon(
                    new PolygonUtil.Point(bbox.get(0), bbox.get(1)),
                    new PolygonUtil.Point(bbox.get(2), bbox.get(1)),
                    new PolygonUtil.Point(bbox.get(2), bbox.get(3)),
                    new PolygonUtil.Point(bbox.get(0), bbox.get(3))
            ));
            //判断多边形是否相交
            logger.info("-------polygonList.size()------{}------------", po
            for (int i = 0; i < polygonList.size(); i++) {
                List<ModelResult> modelResultList = coincideModelResults.ge
                if (CollectionUtil.isEmpty(modelResultList))
                    modelResultList = new ArrayList<>();
                //判断多边形是否相交
                String area = getAreaProportion(locationPolygon, polygonLis
                logger.info("************{}**********area-----{}*****", i,
                //判断设置是在框外还是框内 : 0:框内，1：框外
                if (monitoringDeviceModel.getBoxStatus() == null || monitor
                    //判断当前面积是否大于设置阈值
                    if (Double.parseDouble(area) > 0 && new BigDecimal(area
                        logger.error("面积占比大小{}>面积占比阈值{}", area, mon
                        modelResultList.add(modelResult);
                        logger.info("----------modelResult---------{}", gso
                    }
                    coincideModelResults.put(i + "", modelResultList);
                } else {
                    if (new BigDecimal(area).compareTo(new BigDecimal(Strin
                        modelResultList.add(modelResult);
                    }
                    coincideModelResults.put(i + "", modelResultList);
                }
            }
        }
        logger.info("----------------------end-----------------++++++++++
        return coincideModelResults;
    }

    public static String getAreaProportion(PolygonUtil.Polygon locationPoly
        //框图坐标
        List<PolygonUtil.Point> polygonTestPoints = polygonTest.getPoints()
        //结果坐标
        List<PolygonUtil.Point> locationPolygonPoints = locationPolygon.get
        // 创建第一个多边形
        GeometryFactory factory = new GeometryFactory();

        com.vividsolutions.jts.geom.Coordinate[] coords1 = new com.vividsol
        for (int i = 0; i < polygonTestPoints.size(); i++) {
```

```
            coords1[i] = new com.vividsolutions.jts.geom.Coordinate(polygon
            if (polygonTestPoints.size() - 1 == i) {
                coords1[i + 1] = new com.vividsolutions.jts.geom.Coordinate
            }
        }
        com.vividsolutions.jts.geom.Polygon polygon1 = factory.createPolygo

        com.vividsolutions.jts.geom.Coordinate[] coords2 = new com.vividsol
        for (int i = 0; i < locationPolygonPoints.size(); i++) {
            coords2[i] = new com.vividsolutions.jts.geom.Coordinate(locatio
            if (locationPolygonPoints.size() - 1 == i) {
                coords2[i + 1] = new com.vividsolutions.jts.geom.Coordinate
            }
        }
        com.vividsolutions.jts.geom.Polygon polygon2 = factory.createPolygo

        // 对第一个多边形进行自交处理
        Geometry polygon1Buffer = polygon1.buffer(0.01);
        Geometry polygon1BufferUnion = polygon1Buffer.union();

        // 计算两个多边形的相交面积
        Geometry intersection = polygon2.intersection(polygon1BufferUnion);

        // 计算相交面积占polygon1面积的百分比
        double percentage = intersection.getArea() / polygon2.getArea() * 1

        return new Gson().toJson(percentage);
    }

    /**
     * 格式化模型识别结果列表为框图Location列表
     *
     * @param coincideModelResults
     */
    private ArrayList<Location> getLocations(List<ModelResult> coincideMode
        ArrayList<Location> locations = new ArrayList<>();
        for (ModelResult modelResult : coincideModelResults) {
            Location location = new Location();
            List<Double> bbox = modelResult.getBbox();
            location.setX(bbox.get(0).intValue());
            location.setY(bbox.get(1).intValue());
            Double width = bbox.get(2) - bbox.get(0);
            location.setWidth(width.intValue());
            Double height = bbox.get(3) - bbox.get(1);
            location.setHeight(height.intValue());
            locations.add(location);
        }
        return locations;
    }

    /**
     * 判断是否有聚集
     * K-Means算法计算聚集情况，true 聚集 false 没有聚集
     *
     * @param monitoringDeviceModel
     * @param locations
     * @return
     */
    private boolean isGather(PcMonitoringDeviceModel monitoringDeviceModel,
        boolean isGather = true; //是否可以直接保存告警结果，默认可以直接告警
```

```
        String model_id_gather = configService.selectConfigByKey(URLConstar
        if (null != model_id_gather && model_id_gather.trim().length() > 0
            //若需要聚集算法判断的模型名称中包含当前模型名称
            boolean flag = false; // 默认当前模型不是聚集算法
            String[] split = model_id_gather.split(",");
            for (int i = 0; i < split.length; i++) {
                if (Long.parseLong(split[i]) == monitoringDeviceModel.getAi
                    flag = true; // 是聚集算法
                    break;
                }
            }
            //当前是聚集算法
            if (flag) {
                //获取矩形框的中心点数组
                ArrayList<CentralPoint> centralPoints = RectangleUtil.calcu
                // 将中心点坐标用double数组存储，方便聚类计算使用
                ArrayList<double[]> dataSet = new ArrayList<>();
                for (CentralPoint centralPoint : centralPoints) {
                    double[] onePoint = new double[2];
                    onePoint[0] = centralPoint.getX();
                    onePoint[1] = centralPoint.getY();
                    dataSet.add(onePoint);
                }
                //K-Means算法计算聚集情况 true 聚集 false 没有聚集
                isGather = ClusteringUtils.kMeansGather(dataSet, Integer.pa
            }

        }
        return isGather;
    }

    /**
     * 判断当前通道是否可以告警
     *
     * @param createDate
     * @param taskId
     * @param locations
     * @param imgPath
     * @param pcMonitoringDeviceModel
     * @return
     */
    private TimeThresholdInfo isTimeThreshold(String createDate, Long taskI
        TimeThresholdInfo timeThresholdInfo = null;
        String timeThreshold = pcMonitoringDeviceModel.getTimeThreshold();
        Integer monitoringStatus = pcMonitoringDeviceModel.getMonitoringSta
        if ((monitoringStatus == null || monitoringStatus == 0) && null !=
            //若不为空且不等于0
            //判断redis中当前通道、当前模型是否存在监测时长相关信息，若不存在，保存第
            String keyTimeThreshold = pcMonitoringDeviceModel.getMonitoring
            if (!redisUtil.hasKey(keyTimeThreshold)) {
                // 若不存在，保存第一次监测的信息
                //第一次 number默认为1
                timeThresholdInfo = new TimeThresholdInfo(createDate, 1, lo
                redisUtil.set(keyTimeThreshold, gson.toJson(timeThresholdIn
                logger.info("第一次监测信息键值：" + keyTimeThreshold);
                timeThresholdInfo.setFlag(false);
            } else {
                //  若存在，获取上一次监控信息
                timeThresholdInfo = gson.fromJson(String.valueOf(redisUtil.
                timeThresholdInfo.setNumber(timeThresholdInfo.getNumber() +
```

```
                logger.info("第" + (timeThresholdInfo.getNumber() + 1) + "次
                List<ModelResult> results = timeThresholdInfo.getModelResul
                // 大于等于阈值时，比较获取最大得分的监控信息
                if (pcMonitoringDeviceModel.getType().equals(URLConstant.MO
//                    logger.info("大于等于阈值时，本次监测覆盖时间---------"+cr
                    //获取本次得分的最大值
                    double maxLocal = 0.0;
                    for (ModelResult result : modelResults) {
                        maxLocal = (result.getScore() >= maxLocal) ? result
                    }
//                    logger.info("获取本次得分的最大值---------"+maxLocal);
                    //获取上次得分的最大值
                    double maxLast = 0.0;
                    for (ModelResult result : results) {
                        maxLast = (result.getScore() >= maxLast) ? result.g
                    }
//                    logger.info("获取上次得分的最大值---------"+maxLast);
                    //若本次得分大于等于上次得分
                    if (maxLocal >= maxLast) {
//                        logger.info("本次监测覆盖时间---------"+createDate)
                        timeThresholdInfo.setLocations(locations);
                        timeThresholdInfo.setImgPath(imgPath);
                        timeThresholdInfo.setModelResults(modelResults);
                    }
                    //
                }
                // 大于零且小于等于阈值，比较获取最大得分的监控信息
                if (pcMonitoringDeviceModel.getType().equals(URLConstant.MO
//                    logger.info("大于零且小于等于阈值时，本次监测覆盖时间------
                    //获取本次得分的最大值
                    double maxLocal = 0.0;
                    for (ModelResult result : modelResults) {
                        maxLocal = (result.getScore() >= maxLocal) ? result
                    }
//                    logger.info("获取本次得分的最大值---------"+maxLocal);
                    //获取上次得分的最大值
                    double maxLast = 0.0;
                    for (ModelResult result : results) {
                        maxLast = (result.getScore() >= maxLast) ? result.g
                    }
//                    logger.info("获取上次得分的最大值---------"+maxLast);
                    //若本次得分大于等于上次得分
                    if (maxLocal >= maxLast) {
                        timeThresholdInfo.setLocations(locations);
                        timeThresholdInfo.setImgPath(imgPath);
                        timeThresholdInfo.setModelResults(modelResults);
                    }
                }
                //小于等于阈值（包含识别没有结果），比较获取最小得分的监控信息
                if (pcMonitoringDeviceModel.getType().equals(URLConstant.MO
//                    logger.info("小于等于阈值（包含识别没有结果）时，本次监测覆盖
                    // 获取本次得分的最小值
                    double minLocal = 0.0;
                    for (ModelResult result : modelResults) {
                        minLocal = (result.getScore() <= minLocal) ? result
                    }
//                    logger.info("获取本次得分的最小值---------"+minLocal);
                    // 获取上次得分的最小值
                    double minLast = 0.0;
                    for (ModelResult result : results) {
```

```
                        minLast = (result.getScore() <= minLast) ? result.g
                    }
//                    logger.info("获取上次得分的最小值---------"+minLast);
                    //若本次得分小于等于上次得分
                    if (minLocal <= minLast) {
                        timeThresholdInfo.setLocations(locations);
                        timeThresholdInfo.setImgPath(imgPath);
                        timeThresholdInfo.setModelResults(modelResults);
                    }
                }
                // 本次监测时间和第一次监控时间之差是否大于等于监测时长 若大于等于返回
                boolean b = DayUtils.compareTimeMinute(timeThresholdInfo.ge
                if (b) {
                    logger.info("超过监测时间---------" + timeThreshold);
                    // 本次监测时间和第一次监控时间之差大于等于监测时长
                    // 获取当前任务的轮询时间
//                    PcMonitoringTask pcMonitoringTask = taskService.selec
                    String timeThresholdProportion = pcMonitoringDeviceMode
//                    logger.info("当前比例为---------"+time_threshold_propo
//                    if ((timeThresholdInfo.getNumber() + 1) >= ((Integer.
                    if ((timeThresholdInfo.getNumber() + 1) >= Integer.pars
                        // 若当前时间大于等于第一次报警后的2倍监测时长后，就不报警
                        if (DayUtils.compareTimeMinute(timeThresholdInfo.ge
                            timeThresholdInfo.setFlag(false);
                        } else {
                            timeThresholdInfo.setFlag(true);
                        }
                    } else {
                        //若不大于
                        timeThresholdInfo.setFlag(false);
                    }
                    //清除当前redis信息
                    redisUtil.del(keyTimeThreshold);
                } else {
                    logger.info("未超过监测时间---------" + timeThreshold);
                    // 本次监测时间和第一次监控时间之差小于监测时长，记录当前最好的监
                    redisUtil.set(keyTimeThreshold, gson.toJson(timeThresho
                    timeThresholdInfo.setFlag(false);
                }
            }
        } else {
            timeThresholdInfo = new TimeThresholdInfo();
            timeThresholdInfo.setFlag(true);
            timeThresholdInfo.setImgPath(imgPath);
            timeThresholdInfo.setLocations(locations);
            timeThresholdInfo.setModelResults(modelResults);
        }
        return timeThresholdInfo;
    }


    /**
     * 根据坐标截图保存
     *
     * @param locations              坐标地址数组
     * @param imgPath                原图地址
     * @param imei                   通道号
     * @param monitoringDeviceModel 模型
     * @return 返回框图保存的地址
     */
```

```
    public String transformImg(ArrayList<Location> locations, String imgPat
        String frameImgPath = imei + "/" + "frameImg" + "/" + DayUtils.getT
        //新建文件夹
        FileUtil.createFile(RuoYiConfig.getFfmpefPath() + "/" + frameImgPat
        FileUtil.createFile(RuoYiConfig.getOriginalFfmpefPath() + "/" + fra
        String frameImgName = DayUtils.getToday() + "_" + DayUtils.getToday
        String frameOriginalImgPath = RuoYiConfig.getOriginalFfmpefPath() +
        frameImgPath = RuoYiConfig.getFfmpefPath() + "/" + frameImgPath + "
        //保存原图
        try {
            FileUtil.copy(imgPath, frameOriginalImgPath);
        } catch (IOException e) {
            e.printStackTrace();
            logger.error("原图复制失败! -----{}", e.getMessage());
        }
        Graphics g = null;
        FileOutputStream out = null;
        try {
            BufferedImage image = ImageIO.read(new File(imgPath));
            g = image.getGraphics();
            BasicStroke stokeLine = new BasicStroke(2.0f);
            Graphics2D g2 = (Graphics2D) g;
            g2.setStroke(stokeLine);
            g.setColor(Color.RED);//画笔颜色
            if (locations.size() > 0) {
                for (Location location : locations) {
                    //矩形框(原点x坐标，原点y坐标，矩形的长，矩形的宽)
                    g.drawRect(location.getX(), location.getY(), location.g
                }
            }
            // coordinates: [[{"oX":151,"oY":108},{"oX":417,"oY":130},{"oX"
            JSONArray coordinates = JSONArray.fromObject(monitoringDeviceMo
            if (coordinates.size() > 0) {
                Gson gson = new Gson();
                for (Object coordinateArrays : coordinates) {
                    JSONArray coordinateList = JSONArray.fromObject(coordin
                    if (coordinateList.size() > 0) {
                        Polygon polygon = new Polygon();
                        for (int i = 0; i < coordinateList.size(); i++) {
                            Coordinate coordinate = gson.fromJson(coordinat
                            polygon.addPoint(coordinate.getoX(), coordinate
                        }
                        //画多边形
                        g.setColor(Color.BLUE);//画笔颜色
                        g.drawPolygon(polygon);
                    }
                }
            }
            //生成水印
            int srcImgWidth = image.getWidth(null);
            int srcImgHeight = image.getHeight(null);
            // 加水印
//            g.drawImage(image, 0, 0, srcImgWidth, srcImgHeight, null);
            // Font font = new Font("Courier New", Font.PLAIN, 12);
            Font font = new Font("宋体", Font.PLAIN, srcImgWidth / 300 * 8)
            g.setColor(new Color(30, 144, 255)); // 根据图片的背景设置水印颜色
            g.setFont(font);
            for (int i = 0; i < markContents.size(); i++) {
                int x = 10;
                int y = srcImgHeight - srcImgHeight / 5 + srcImgWidth / 300
```

```
                g.drawString(markContents.get(i), x, y);
            }
            out = new FileOutputStream(frameImgPath);//输出图片的地址
            ImageIO.write(image, "jpeg", out);
            out.flush();
        } catch (IOException e) {
            logger.error("根据坐标截图保存失败" + e);
            return null;
        } finally {
            g.dispose();
            try {
                if (out != null) {
                    out.close();
                }
            } catch (IOException e) {
                logger.error("输出流关闭失败");
            }
        }
        return frameImgPath;
    }

    /**
     * 保存告警信息
     *
     * @param createDate             创建时间
     * @param modelResults           模型调用返回结果
     * @param frameImgPath           框图保存地址
     * @param pcMonitoringDevice     通道信息
     * @param monitoringDeviceModel 模型信息
     */
    public void saveMonitoringDeviceResult(String createDate, List<ModelRes
        PcMonitoringDeviceResult monitoringDeviceResult = new PcMonitoringD
        monitoringDeviceResult.setDelFlag(URLConstant.DELFLAG_0); // 未删除标
        monitoringDeviceResult.setCreateId(pcMonitoringDevice.getCreateId()
        monitoringDeviceResult.setCreateDate(createDate); // 创建时间
        monitoringDeviceResult.setDeptId(pcMonitoringDevice.getDeptId()); /
        monitoringDeviceResult.setMonitoringDeviceId(monitoringDeviceModel.
        monitoringDeviceResult.setAiModelId(monitoringDeviceModel.getAiMode
        monitoringDeviceResult.setSimilarity(monitoringDeviceModel.getSimil
        monitoringDeviceResult.setImgUrl(Constants.RESOURCE_PREFIX + frameI
        monitoringDeviceResult.setResult(modelResults.toString()); //接口调用
        monitoringDeviceResult.setAlarmEvent(0L);//设置事件类型
        monitoringDeviceResult.setSynStatus(5);
        int i = iPcMonitoringDeviceResultService.insertPcMonitoringDeviceRe
        if (i > 0) {
            PcDataReportConfig pcDataReportConfig = pcDataReportConfigServi
            //数据上报顶层平台
            if (iPcMonitoringDeviceResultService.synDataReport(monitoringDe
                logger.info("告警信息（id->{}）信息上报成功！", monitoringDevic
            } else {
                logger.info("告警信息（id->{}）信息上报失败！", monitoringDevic
            }
        }
        //判断当前通道、当前模型是否允许告警推送
        boolean isTimeInterval = isTimeInterval(createDate, monitoringDevic
        if (isTimeInterval) {
            pool.execute(() -> {
                //通道告警通知 短信、钉钉、企业微信、邮箱、第三方接口
                pcAlarmConfigService.sendMonitoringMsg(frameImgPath, monito
            });
```

```
        }
        publishAlarmInformation(monitoringDeviceResult, pcMonitoringDevice.
    }

    @SuppressWarnings("all")
    public void publishAlarmInformation(PcMonitoringDeviceResult monitoring
        String screen_url = configService.selectConfigByKey(URLConstant.SCR
        if (null == screen_url || screen_url.trim().length() < 1 || screen_
            return;
        //判断连接状态用户是否为null
        ConcurrentHashMap<String, ConcurrentHashMap<String, ConcurrentHashM
        if (CollectionUtil.isEmpty(webSocketSet))
            return;

        //获取该部门及其父部门id
        List<Long> parentDeptIdList = sysDeptService.getParentDeptIdList(mo

        ConcurrentHashMap<String, ConcurrentHashMap<String, WebSocketServer
        if (CollectionUtil.isNotEmpty(apiScreen)) {
            //判断推送视频流
            //判断当前部门下的用户及其父部门
            Map<String, ConcurrentHashMap<String, WebSocketServer>> sendUse
            sendUserMap.entrySet().stream().forEach(entry -> {
                try {
                    //创建发布实体类
                    String is_or_not_important_alarm = configService.select
                    List<String> isOrNotImportantAlarm = StringUtils.isNotE
                    Map<String, Object> hashMap = new HashMap<>();
                    hashMap.put("id", monitoringDeviceResult.getId());
                    hashMap.put("createDate", monitoringDeviceResult.getCre
                    hashMap.put("deptName", deptName);
                    hashMap.put("aiModelName", monitoringDeviceResult.getAi
                    hashMap.put("alarmEvent", monitoringDeviceResult.getAla
                    hashMap.put("imgUrl", screen_url + monitoringDeviceResu
                    hashMap.put("monitoringDeviceName", monitoringDeviceRes
                    hashMap.put("importantAlarm", !Optional.ofNullable(isOr

                    entry.getValue().entrySet().stream().forEach(entry_1 ->
                } catch (Exception e) {
                    logger.error("推送告警画面失败----->告警信息id{}-------->用
                    logger.error("推送告警画面失败" + e);
                }
            });
        }
    }

    @SuppressWarnings("all")
    public void publishAlarmInformation(WebSocketServer webSocketServer, St
            moduleName, String methodName) {
        try {
            HashMap<String, Object> map = new HashMap<>();
            map.put("code", 200);
            map.put("errMessage", "");
            map.put("data", hashMap);
            Map<String, Object> data = new HashMap<>();
            data.put("moduleName", moduleName);
            data.put("methodName", methodName);
            data.put("params", map);
            webSocketServer.sendMessage(gson.toJson(data));
        } catch (Exception e) {
```

```
            e.printStackTrace();
            logger.error("Error sending json map!");
        }

    }

    /**
     * 判断当前通道、当前模型是否允许告警推送
     *
     * @param createDate
     * @param monitoringDeviceModel
     * @return
     */
    private boolean isTimeInterval(String createDate, PcMonitoringDeviceMod
        boolean isTimeInterval = false;
        if (monitoringDeviceModel.getAlarmStatus().equals(URLConstant.PC_MO
            // 若允许推送，判断告警推送间隔时间是否等于0
            String timeInterval = monitoringDeviceModel.getTimeInterval();
            if (timeInterval.equals("0")) {
                isTimeInterval = true;
            } else {
                //若不等于0，查询当前通道当前模型上一次告警推送时间（从redis中取）
                String key_time_interval = monitoringDeviceModel.getMonitor
                if (!redisUtil.hasKey(key_time_interval)) {
                    // 若不存在，立即推送，并记录当前创建时间到redis
                    logger.info("第一次告警推送时间：" + key_time_interval);
                    redisUtil.set(key_time_interval, createDate);
                    isTimeInterval = true;
                } else {
                    // 若存在，比较上一次告警推送时间和创建时间
                    //获取上一次告警推送时间
                    String redisTime = String.valueOf(redisUtil.get(key_tim
                    // 创建时间和上一次告警推送时间之差大于等于告警间隔时间 若大于等
                    boolean b = DayUtils.compareTimeMinute(redisTime, creat
                    if (b) {
                        // 若创建时间和上一次告警时间之差大于等于告警间隔时间，立即推
                        redisUtil.set(key_time_interval, createDate);
                        isTimeInterval = true;
                    } else {
                        // 若创建时间和上一次告警时间之差小于告警间隔时间，跳过推送
                        isTimeInterval = false;
                    }
                }
            }
        }
        return isTimeInterval;
    }

}
```

CheckResultAlarmRule

```
package com.ruoyi.pc.domain;

import com.fasterxml.jackson.annotation.JsonFormat;
import com.ruoyi.common.annotation.Excel;
import com.ruoyi.common.core.domain.BaseEntity;
import com.ruoyi.pc.task.util.Location;
```

```
import lombok.Data;
import lombok.EqualsAndHashCode;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

/**
 * 监控通道对象 pc_monitoring_device
 *
 * @author sc
 * @date 2021-03-09
 */
@Data
@EqualsAndHashCode(callSuper = true)
public class CheckResultAlarmRule extends BaseEntity {
    private boolean alarm;

    private ArrayList<Location> locations;

    private List<ModelResult> modelResults;
}
```

```
public class PcVideoServiceImpl implements IPcVideoService
```

```
package com.ruoyi.pc.service.impl;

import com.alibaba.fastjson.JSONObject;
import com.google.gson.Gson;
import com.ruoyi.common.config.RuoYiConfig;
import com.ruoyi.common.utils.StringUtils;
import com.ruoyi.pc.config.redis.RedisUtil;
import com.ruoyi.pc.domain.*;
import com.ruoyi.pc.mapper.PcMonitoringDeviceMapper;
import com.ruoyi.pc.mapper.PcMonitoringTaskDeviceMapper;
import com.ruoyi.pc.mapper.PcMonitoringTaskMapper;
import com.ruoyi.pc.service.*;
import com.ruoyi.pc.task.VideoStreamTask;
import com.ruoyi.pc.util.*;
import com.ruoyi.quartz.service.ISysJobService;
import com.ruoyi.system.service.ISysConfigService;
import lombok.extern.log4j.Log4j2;
import net.sf.json.JSONArray;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;

import javax.annotation.Resource;
import java.io.File;
import java.io.IOException;
import java.math.BigDecimal;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.*;
import java.util.concurrent.CountDownLatch;
```

```
@Service
@Log4j2
public class PcVideoServiceImpl implements IPcVideoService {

    public static Logger logger = LoggerFactory.getLogger(VideoStreamTask.c

    Gson gson = new Gson();

    @Value("${server.port}")
    private int serverPort;

    @Resource
    ThreadPoolTaskExecutor pool;

    @Autowired
    IPcMonitoringTaskService pcMonitoringTaskService;
    @Autowired
    ISysConfigService configService;
    @Autowired
    private IPcMonitoringDeviceService pcMonitoringDeviceService;
    @Autowired
    FasterRcnnHttpService httpService;

    @Autowired
    IPcMonitoringDeviceModelService iPcMonitoringDeviceModelService;

    @Autowired
    RedisUtil redisUtil;

    @Autowired
    IPcMonitoringTaskDeviceService iPcMonitoringTaskDeviceService;

    @Autowired
    private ISysJobService sysJobService;

    @Resource
    private PcMonitoringTaskDeviceMapper pcMonitoringTaskDeviceMapper;

    @Resource
    private PcMonitoringDeviceMapper pcMonitoringDeviceMapper;

    @Resource
    private PcMonitoringTaskMapper pcMonitoringTaskMapper;

    @Resource
    AiTaskService aiTaskService;

    @Resource
    TaskService taskService;


    @Override
    public synchronized void startVideoStreamByTaskId(Long taskId, boolean
        //获取当前监控任务
        PcMonitoringTask pcMonitoringTask = pcMonitoringTaskMapper.selectPc
        //查询监控任务下开启的监控通道
        List<PcMonitoringDevice> monitoringDeviceIdList = pcMonitoringTaskD
        if (null != pcMonitoringTask && null != monitoringDeviceIdList && m
            CountDownLatch countDownLatch = new CountDownLatch(monitoringDe
```

```
            // 更新监控任务状态
            PcMonitoringTask pcMonitoringTaskStatus = new PcMonitoringTask(
            pcMonitoringTaskStatus.setId(taskId);
            pcMonitoringTaskStatus.setStatus(URLConstant.PC_TASK_STATUS_CAR
            pcMonitoringTaskMapper.updatePcMonitoringTask(pcMonitoringTaskS
            redisUtil.hset("taskThreadPoolExecutor", taskId + "", 0, 300);
            for (PcMonitoringDevice pcMonitoringDevice : monitoringDeviceId
                taskService.taskAsync(pcMonitoringTask, taskId, pcMonitorin
            }
        }
    }

    @Override
    public synchronized void startVideoStreamByMonitorDeviceId(Long taskId,
        //获取当前监控任务
        PcMonitoringTask pcMonitoringTask = pcMonitoringTaskMapper.selectPc
        PcMonitoringDevice pcMonitoringDevice = pcMonitoringDeviceMapper.se
        if (null != pcMonitoringTask && null != pcMonitoringDevice) {
            CountDownLatch countDownLatch = new CountDownLatch(1);
            // 更新监控任务状态
            PcMonitoringTask pcMonitoringTaskStatus = new PcMonitoringTask(
            pcMonitoringTaskStatus.setId(taskId);
            pcMonitoringTaskStatus.setStatus(URLConstant.PC_TASK_STATUS_CAR
            pcMonitoringTaskMapper.updatePcMonitoringTask(pcMonitoringTaskS
            redisUtil.hset("taskThreadPoolExecutor", taskId + "", 0, 300);
            taskService.taskAsync(pcMonitoringTask, taskId, pcMonitoringDev
        }
    }

    @Override
    public synchronized void stopVideoStreamByTaskId(Long taskId, boolean i
        //获取当前监控任务下的监控通道
        List<PcMonitoringDevice> monitoringDeviceIdList = pcMonitoringTaskD
        CountDownLatch countDownLatch = new CountDownLatch(monitoringDevice
        // 更新监控任务状态
        PcMonitoringTask pcMonitoringTaskStatus = new PcMonitoringTask();
        pcMonitoringTaskStatus.setId(taskId);
        pcMonitoringTaskStatus.setStatus(URLConstant.PC_TASK_STATUS_CARRY_C
        pcMonitoringTaskMapper.updatePcMonitoringTask(pcMonitoringTaskStatu
        redisUtil.hset("taskThreadPoolExecutor", taskId + "", 1, 300);
        for (PcMonitoringDevice pcMonitoringDevice : monitoringDeviceIdList
            taskService.taskAsync(null, taskId, pcMonitoringDevice, countDo
        }
    }

    @Override
    public synchronized void stopVideoStreamByMonitorDeviceId(Long taskId,
        PcMonitoringDevice pcMonitoringDevice = pcMonitoringDeviceMapper.se
        if (pcMonitoringDevice != null) {
            CountDownLatch countDownLatch = new CountDownLatch(1);
            // 更新监控任务状态
            PcMonitoringTask pcMonitoringTaskStatus = new PcMonitoringTask(
            pcMonitoringTaskStatus.setId(taskId);
            pcMonitoringTaskStatus.setStatus(URLConstant.PC_TASK_STATUS_CAR
            pcMonitoringTaskMapper.updatePcMonitoringTask(pcMonitoringTaskS
            redisUtil.hset("taskThreadPoolExecutor", taskId + "", 1, 300);
            taskService.taskAsync(null, taskId, pcMonitoringDevice, countDo
        }
    }
```

```
    /**
     * 处理视频流回调返回结果
     *
     * @param videoResult videoResult
     */
    @Override
    public void videoResult(VideoResult videoResult) {
        pool.execute(() -> {
            try {
                if (videoResult.getMessage() == null || Objects.equals(vide
                    String videoId = videoResult.getVideoId();//视频流获取监
                    if (null == videoId || "".equals(videoId) || !redisUtil
                        logger.error("当前结果taskId未保存到redis里，请查询redi
                        return;
                    }
                    String className = videoResult.getModelName();
                    if (null == className || "".equals(className)) {
                        logger.error("当前结果返回的模型标识为空");
                        return;
                    }
                    Long taskId = Long.valueOf(String.valueOf(redisUtil.get
                    if (null == taskId) {
                        logger.error("从redis中获取当前通道的任务Id为空");
                        return;
                    }
                    //查询唯一启用通道
                    PcMonitoringDevice pcMonitoringDeviceQuery = new PcMoni
                    pcMonitoringDeviceQuery.setUniqueCode(videoId);
                    pcMonitoringDeviceQuery.setAppStatus(URLConstant.STATUS
                    List<PcMonitoringDevice> pcMonitoringDevices = pcMonito
                    if (null == pcMonitoringDevices || pcMonitoringDevices.
                        logger.error("当前已启用的监控通道不存在");
                        return;
                    }
                    PcMonitoringDevice pcMonitoringDevice = pcMonitoringDev
                    //3、根据通道ID，获取关联的模型列表  (模型未删除、已启用、且模型
                    List<PcMonitoringDeviceModel> pcMonitoringDeviceModels
                    if (null == pcMonitoringDeviceModels || pcMonitoringDev
                        logger.error("当前通道关联的模型列表不存在");
                        return;
                    }
                    //4、根据通道信息，截取图片,返回截图地址
                    String imgPath = videoResult.getImagePath();
                    //4.1、判断截图地址是否为空
                    if (null == imgPath) {
                        logger.error("通道截取图片失败，通道号为----------------
                        return; // 图片地址为空，跳过当前通道，继续下个通道
                    }
                    //4.2、判断截图是否成功
                    File file = new File(imgPath);
                    if (file.exists()) {
                        String createDate = DayUtils.getNow();
                        //判断当前是否存在识别类别是2的模型
                        List<ModelResult> result = videoResult.getResult();
                        // 7、遍历当前通道关联模型列表
                        String key = "";
                        for (PcMonitoringDeviceModel pcMonitoringDeviceMode
                            boolean isOrNotSaveFlag = false;
                            if (pcMonitoringDeviceModel.getEnableDeduplicat
                                    && pcMonitoringDeviceModel.getDeduplica
```

```
                                && pcMonitoringDeviceModel.getDeduplica
                            if (Objects.equals(pcMonitoringDeviceModel.
                                key = "quantity_detection" + taskId + p
                            } else if (Objects.equals(pcMonitoringDevic
                                key = "coordinate_detection" + taskId +
                            } else if (Objects.equals(pcMonitoringDevic
                                key = "deduplication_identification" +
                            }
                        }
                        if (URLConstant.MODEL_TYPE_2.equals(pcMonitorin
                            String pc_http_url = configService.selectCo
                            if (null != pc_http_url && pc_http_url.trim
                                Map<String, Object> stringObjectMap = n
                                stringObjectMap.put("taskId", taskId);
                                stringObjectMap.put("pcMonitoringDevice
                                stringObjectMap.put("monitoringDeviceId
                                stringObjectMap.put("createDate", creat
                                stringObjectMap.put("frontAlgoResult",
                                getModelResultHttp(imgPath, pc_http_url
                                continue;
                            }
                        }
                        // 8.2、判断识别结果是否为空
                        if (null != result && result.size() > 0) {
                            // 8.3 当模型比较类型是大于等于相似度阈值时
                            if (pcMonitoringDeviceModel.getType().equal
                                //8.3.1、筛选大于等于相似度阈值的识别结果
                                List<ModelResult> modelResults = new Ar
                                for (int i = 0; i < result.size(); i++)
                                    // 识别结果格式化成ModelResult
                                    ModelResult modelResult = result.ge
                                    // 获取结果category等于当前模型label,
                                    if (modelResult.getCategory().equal
                                            && Double.valueOf(modelResu
                                        isOrNotSaveFlag = true;
                                        modelResults.add(modelResult);
                                    }
                                }

                                List<ModelResult> personFrontAlgo = get
                                // 8.3.2、筛选后识别结果数 大于等于 告警目标
                                if (null != personFrontAlgo && personFr
                                    // 8.3.3、发布到redis通道
                                    publishAiMessage(taskId, personFron
                                    logger.info("任务ID--------" + task
                                }
                            }
                            // 8.4 当模型比较类型是大于零且小于等于相似度阈值时
                            if (pcMonitoringDeviceModel.getType().equal
                                //8.4.1、筛选大于零且小于等于相似度阈值的识别
                                List<ModelResult> modelResults = new Ar
                                for (int i = 0; i < result.size(); i++)
                                    // 识别结果格式化成ModelResult
                                    ModelResult modelResult = result.ge
                                    // 获取结果category等于当前模型label,
                                    if (modelResult.getCategory().equal
                                            && Double.valueOf(modelResu
                                            && Double.valueOf(modelResu
                                        isOrNotSaveFlag = true;
                                        modelResults.add(modelResult);
```

```
                                    }
                                }
                                List<ModelResult> personFrontAlgo = get
                                // 8.4.2、筛选后识别结果数 大于等于 告警目标
                                if (null != personFrontAlgo && personFr
                                    // 8.4.3、发布到redis通道
                                    publishAiMessage(taskId, personFron
                                    logger.info("任务ID--------" + task
                                }
                            }
                        } else {
                            if (StringUtils.isNotBlank(key) && redisUti
                                if (key.startsWith("quantity_detection"
                                    Map<Object, Object> hmget = redisUt
                                    long expire = redisUtil.getExpire(k
                                    if (hmget.size() == 1) {
                                        redisUtil.hset(key, "2", DayUti
                                    } else if (hmget.size() == 2) {
                                        redisUtil.hset(key, "3", DayUti
                                    } else if (hmget.size() == 3) {
                                        Object second = redisUtil.hget(
                                        Object three = redisUtil.hget(k
                                        redisUtil.del(key);
                                        redisUtil.hset(key, "1", second
                                        redisUtil.hset(key, "2", three,
                                        redisUtil.hset(key, "3", DayUti
                                    }
                                } else {
                                    logger.error("--------------------
                                    redisUtil.del(key);
                                }
                            }
                        }
                        // 8.3、若模型设置保留截图，保存原图
                        if (pcMonitoringDeviceModel.getIsSave().equals(
                            pool.execute(() -> {
                                try {
                                    //根据模型保存图片
                                    //保存实体路径为 ffmpeg图片保存地址+通道
                                    String baseImgPath = RuoYiConfig.ge
                                    //新建文件夹
                                    FileUtil.createFile(baseImgPath);
                                    baseImgPath = baseImgPath + "/" + D
                                    FileUtil.copy(imgPath, baseImgPath)
                                } catch (IOException e) {
                                    logger.error("保存原图失败" + e);
                                }
                            });
                        }

                    }
                } else {
                    logger.error("{} 文件不存在++++++++++++++++++++", im
                }
            } else {
                //结果失败
                logger.error("当前处理是视频流处理出现错误时返回的信息，请根据
            }
        } catch (Exception e) {
            logger.error("视频流回调处理结果报错" + e);
```

```
            }
        });
    }

    public void getOffDutyDetection(String createDate, Long taskId, String
        if (pcMonitoringDeviceModel.getType().equals(URLConstant.MODEL_TYPE
            //8.5.1、筛选大于零且小于等于相似度阈值的识别结果
            List<ModelResult> modelResults = new ArrayList<ModelResult>();
            if (result.size() > 0) {
                for (int i = 0; i < result.size(); i++) {
                    // 识别结果格式化成ModelResult
                    ModelResult modelResult = result.get(i);
                    // 获取结果category等于当前模型label
                    if (modelResult.getCategory().equals(pcMonitoringDevice
                        modelResults.add(modelResult);
                    }
                }
            }
            // 8.5.2、筛选后识别结果数等于0
            if (modelResults.size() == 0) {
                ModelResult modelResult = new ModelResult();
                modelResult.setCategory(pcMonitoringDeviceModel.getLabel())
                modelResult.setBbox(new ArrayList<>());
                modelResult.setScore(0);
                modelResults.add(modelResult);
            }
            // 8.5.3、发布到redis通道
            publishAiMessage(taskId, modelResults, createDate, imgPath, pcM
            logger.info("小于等于阈值(包含识别没有结果)--------任务ID--------" +
        }
    }

    /**
     * 处理视频流前置算法回调返回结果
     *
     * @param videoResult
     */
    @Override
    public void videoFrontAlgoResult(VideoResult videoResult) {
        pool.execute(new Runnable() {
            @Override
            public void run() {
                try {
                    Map<String, Object> inferParameterMap = videoResult.get
                    Object pcMonitoringDeviceModelId = inferParameterMap.ge
                    if (null == pcMonitoringDeviceModelId || "".equals(pcMo
                        logger.error("前置算法缺少监控通道关联模型Id【**********
                        return;
                    }
                    PcMonitoringDeviceModel pcMonitoringDeviceModel = iPcMo
                    if (null == pcMonitoringDeviceModel) {
                        logger.error("前置算法数据库缺少监控通道关联模型【******
                        return;
                    }

                    Object pcMonitoringDeviceId = inferParameterMap.get("mo
                    if (null == pcMonitoringDeviceId || "".equals(pcMonitor
                        logger.error("前置算法缺少监控通道Id【***************
                        return;
                    }
```

```
                PcMonitoringDevice pcMonitoringDeviceQuery = new PcMoni
                pcMonitoringDeviceQuery.setId(Long.valueOf(String.value
                List<PcMonitoringDevice> pcMonitoringDevices = pcMonito
                if (null == pcMonitoringDevices || pcMonitoringDevices.
                    logger.error("前置算法数据库缺少监控通道【*********
                    return;
                }

                String imagePath = videoResult.getImagePath();
                if (null == imagePath || "".equals(imagePath)) {
                    logger.error("前置算法缺少图片路径【**************
                    return;
                }
                File file = new File(imagePath);
                if (!file.exists()) {
                    logger.error("前置算法缺少图片文件【**************
                    return;
                }
                Object taskId = inferParameterMap.get("taskId");
                if (null == taskId || "".equals(taskId)) {
                    logger.error("前置算法缺少任务Id【****************
                    return;
                }
                Object createDate = inferParameterMap.get("createDate")
                if (null == createDate || "".equals(createDate)) {
                    logger.error("前置算法缺少创建时间【**************
                    return;
                }
                if (pcMonitoringDeviceModel.getType().equals(URLConstan
                    getOffDutyDetection(String.valueOf(createDate), Lon
                    return;
                }
                Object resultOld = inferParameterMap.get("frontAlgoResu
                if (null == resultOld || "".equals(resultOld)) {
                    logger.error("前置算法缺少算法识别结果【************
                    return;
                }
                //人体前置算法识别结果
                List<ModelResult> result = videoResult.getResult();
                if (null == result || result.size() == 0) {
                    logger.error("前置算法缺少算法识别结果【************
                    return;
                }
                //算法识别结果
                JSONArray resultOldJSONArray = JSONArray.fromObject(res

                //如果是非需要两次结果的交集的    就是按照第二次结果与ROI进行比较
                boolean needCheckForFirstResult = (boolean) inferParame
                List<ModelResult> filterResult;
                if (!needCheckForFirstResult){
                    //不需要与第一次结果进行比较，则直接进行第二次结果的过滤
                    String personBodyPreModelIds = configService.select
                    String[] split = personBodyPreModelIds.split(",");
                    String resVal = null;
                    for (String s : split) {
                        String[] split1 = new String[0];
                        if (s.contains("&")){
                            split1 = s.split("&");
                            s = split1[0];
                        }
```

```
                            if (pcMonitoringDeviceModel.getAiModelId() == L
                                resVal = split1.length > 1 ? split1[1] : pc
                            }
                        }
                        pcMonitoringDeviceModel.setSimilarity(resVal == nul

                        CheckResultAlarmRule checkResultAlarmRule = aiTaskS
                        filterResult = checkResultAlarmRule.getModelResults
                    }else {
                        filterResult = getFilterResult(result, resultOldJSO
                    }


                    if (null != filterResult && filterResult.size() >= Inte
                        // 8.3.3、发布到redis通道
                        publishAiMessage(Long.valueOf(String.valueOf(taskId
                        logger.info("任务ID--------" + taskId + ", 发布时间--
                    }

                } catch (Exception e) {
                    logger.error("前置算法处理结果报错" + e);
                }
            }
        });
    }

    /**
     * 自启动视频流
     *
     * @return
     */
    public void selfStartVideoStream() {
        logger.info("=====================自动开启视频流====================
        try {
            PcMonitoringTask pcMonitoringTask = new PcMonitoringTask();
            List<PcMonitoringTask> pcMonitoringTaskList = pcMonitoringTaskS
            for (PcMonitoringTask pcMonitoringTask1 : pcMonitoringTaskList)
                if (pcMonitoringTask1.getStatus().equals(URLConstant.MONITO
                    //判断当前任务是视频分析还是抽帧检测
                    if (pcMonitoringTask1.getRecognitionMode().equals(URLCo
                        //是否在任务执行期间
                        boolean rNotTriggerStartVideoStream = sysJobService
                        if (rNotTriggerStartVideoStream) {
                            startVideoStreamByTaskId(pcMonitoringTask1.getI
                            Thread.sleep(5000);
                        }
                    }
                }
            }
        } catch (Exception e) {
            logger.error("视频流自启动失败", e.getMessage());
        }
    }

    /**
     * 获取当前设备的主机号和端口号
     *
     * @return
     */
    public String getUrl() {
```

```
        InetAddress address = null;
        try {
            address = InetAddress.getLocalHost();
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }
        return "http://" + address.getHostAddress() + ":" + this.serverPort
    }

    /**
     * 发布消息到redis中
     *
     * @param modelResults
     * @param createDate
     * @param imgPath
     * @param pcMonitoringDevice
     * @param pcMonitoringDeviceModel
     */
    @Async
    public void publishAiMessage(Long taskId, List<ModelResult> modelResult
        AiMessage aiMessage = new AiMessage();
        aiMessage.setTaskId(taskId);
        aiMessage.setCreateDate(createDate);
        aiMessage.setImgPath(imgPath);
        aiMessage.setPcMonitoringDevice(pcMonitoringDevice);
        aiMessage.setPcMonitoringDeviceModel(pcMonitoringDeviceModel);
        aiMessage.setModelResults(modelResults);
        aiTaskService.processMessage(gson.toJson(aiMessage));
    }

    public List<ModelResult> getPersonFrontAlgo(String createDate, Long tas
        // 人体识别算法模型标识
        String person_body_model_name = configService.selectConfigByKey(URL
        // 人体识别算法模型标签
        String person_body_label = configService.selectConfigByKey(URLConst

        String pc_http_url = configService.selectConfigByKey(URLConstant.PC
        // 需要人体前置检测的算法模型
        String person_body_pre_model_ids = configService.selectConfigByKey(
        // 配置模型
        boolean flag = false;
        if (null != person_body_model_name && person_body_model_name.trim()
                && null != person_body_label && person_body_label.trim().le
                && null != person_body_pre_model_ids && person_body_pre_mod
            // 判断需要人体前置检测的算法模型中的是否包含当前模型名称，若包含继续执行，
            // 默认当前模型不在人体检测前置算法模型中
            boolean checkROIResult = false;
            boolean needCheckForFirstResult = true;
            String[] pre_model_ids = person_body_pre_model_ids.split(",");
            for (String pre_model : pre_model_ids) {
                if (pre_model.contains("&")){
                    checkROIResult = true;
                    needCheckForFirstResult = false;
                    String[] split = pre_model.split("&");
                    pre_model = split[0];
                }
                if (null != pcMonitoringDeviceModel.getAiModelId() && Long.
                    flag = true; // 包含在人体检测前置算法模型中
                    break;
                }
```

```
            }
            if (!flag) {
                // 若不包含在，直接返回
                return result;
            } else {
                if (checkROIResult){
                    //检查结果是否符合条件
                    CheckResultAlarmRule checkResultAlarmRule = aiTaskServi
                    if (!checkResultAlarmRule.isAlarm()){
                        return null;
                    }
                }
                if (null != result && result.size() >= Integer.parseInt(pcM
                    Long monitoringDeviceId = pcMonitoringDeviceModel.getMo
                    Map<String, Object> stringObjectMap = new HashMap<>();
                    stringObjectMap.put("taskId", taskId);
                    stringObjectMap.put("pcMonitoringDeviceModelId", pcMoni
                    stringObjectMap.put("monitoringDeviceId", monitoringDev
                    stringObjectMap.put("createDate", createDate);
                    stringObjectMap.put("frontAlgoResult", result);
                    stringObjectMap.put("needCheckForFirstResult",needCheck
                    getModelResultHttp(imgPath, pc_http_url, person_body_mo
                }
                return null;
            }
        } else {
            return result;
        }
    }


    public List<ModelResult> getFilterResult(List<ModelResult> personBodyIn
        logger.info("测试筛选前的结果长度{}**********************************
        // 人体识别算法模型标识
//        String person_body_model_name = configService.selectConfigByKey(U
        // 人体识别算法模型标签
        String person_body_label = configService.selectConfigByKey(URLConst
        // 人体识别算法模型得分
        String person_body_score = configService.selectConfigByKey(URLConst
        if (null == person_body_score || person_body_score.trim().length()
            person_body_score = "0";
        }
        List<ModelResult> personBodyInfos = new ArrayList<>();
        for (ModelResult o : personBodyInfo) {
//            ModelResult personBodyResult = gson.fromJson(o.toString(), Mo
            if (person_body_label.equals(o.getCategory())
                    && o.getScore() >= Double.parseDouble(person_body_score
                personBodyInfos.add(o);
            }
        }
        if (personBodyInfos.size() == 0) {
            return new ArrayList<>();
        }
        //判断当前返回结果是否和框图存在交叉
        List<ModelResult> modelResult = new ArrayList<>();
        for (ModelResult personBodyResult : personBodyInfos) {
            // 人体识别坐标
            List<Double> personBodyBbox = personBodyResult.getBbox();
            // 人体的矩形
            PolygonUtil.Polygon personBodyPolygon = new PolygonUtil.Polygon
```

```
                    new PolygonUtil.Point(personBodyBbox.get(0), personBody
                    new PolygonUtil.Point(personBodyBbox.get(2), personBody
                    new PolygonUtil.Point(personBodyBbox.get(2), personBody
                    new PolygonUtil.Point(personBodyBbox.get(0), personBody
            ));
            for (Object classNameResult1 : result) {
                ModelResult classNameResult = gson.fromJson(classNameResult
                // 调用模型的循环
                if (!"no".equals(classNameResult.getCategory())) {
                    // 调用模型得到的结果坐标
                    List<Double> classNameBbox = classNameResult.getBbox();
                    // 调用模型得到的结果矩形
                    PolygonUtil.Polygon classNamePolygon = new PolygonUtil.
                            new PolygonUtil.Point(classNameBbox.get(0), cla
                            new PolygonUtil.Point(classNameBbox.get(2), cla
                            new PolygonUtil.Point(classNameBbox.get(2), cla
                            new PolygonUtil.Point(classNameBbox.get(0), cla
                    ));
                    if (PolygonUtil.intersectionJudgment(classNamePolygon,
//                                  logger.info("模型识别矩形与人体识别矩形相
                        modelResult.add(classNameResult);
                    }
                }
            }
        }
        // 去重
        if (modelResult.size() != 0 && modelResult.size() != 1) {
            for (int i = 0; i < modelResult.size(); i++) {
                for (int j = i + 1; j < modelResult.size(); j++) {
                    BigDecimal score1 = BigDecimal.valueOf(modelResult.get(
                    BigDecimal score2 = BigDecimal.valueOf(modelResult.get(
                    boolean isEqual = modelResult.get(i).getCategory().equa
                            score1.compareTo(score2) == 0;
                    if (isEqual) {
                        modelResult.remove(j);
                        j--;
                    }
                }
            }
        }
        logger.info("测试筛选后的结果长度{}**********************************
        return modelResult;
    }

    public void getModelResultHttp(String imgPath, String pc_http_url, Stri
        JSONObject param = new JSONObject();
        param.put("imagePath", imgPath);
        param.put("imageName", imgPath.substring(imgPath.lastIndexOf("/") +
        param.put("inferParameter", inferParameter);
        List<ModelNames> modelNamesList = new ArrayList<>();
        ArrayList<String> stringArrayList = new ArrayList<>(Arrays.asList(p
        ModelNames modelNames = new ModelNames(person_body_model_name, stri
        modelNamesList.add(modelNames);
        param.put("modelNames", modelNamesList);
        try {
//            logger.info("前置算法参数**************************************
            String resultPersonAlgo = OkHttpUtil.post(pc_http_url, param.to
//            logger.info("调用底层接口返回值**********************************
        } catch (Exception e) {
            logger.error("前置算法调用错误->{}", e.getMessage());
```

```
        }
    }
}
```

乱七八糟

chewei1_1:

0:0~0:18： 【剪掉】

2:38~2:49 :车后巡视人员

3:21~3:28：车后巡视人员

5:51~7:46 ： 安全绳

8:07~8:55 ：安全绳

chewei5_1:

28:25~29:02：车后巡视人员

32:25~到最后：安全绳

35:06到最后 ：孔盖重新处理 【减掉】

> `ApiController.java`
>
> `@GetMapping("/monitorList")` //获取已开启的视频流在大屏展示 方法：`pcMonitoringDeviceTransferToHttpFlv` 针对添加flv格式的视频流格式有效，如果是 `rtsp/rtmp` 格式的视频流的话，也只能针对推流格式的方式有效，要是非推流的方式，会导致 `rtsp/rtmp` 的流地址不进行推流到SRS专六，会导致大屏端查询出来的视频流也是 `rtsp/rtmp` 的格式，导致播放失败.

```
@Override
public List<MonitorInfo> pcMonitoringDeviceTransferToHttpFlv(List<PcMonitor
  List<MonitorInfo> list1 = new ArrayList<>();
  for (PcMonitoringDevice pcMonitoringDevice : list) {
      //判断当前是否在任务期间，任务开启，且是视频分析-推流
      MonitorInfo monitorInfo = new MonitorInfo();
      monitorInfo.setName(pcMonitoringDevice.getName());
      monitorInfo.setUpdateDate(pcMonitoringDevice.getCreateDate());
      String screen_stream_type = configService.selectConfigByKey(URLConsta
      if (null != screen_stream_type && !screen_stream_type.trim().isEmpty(
          monitorInfo.setStreamUrl(pcMonitoringDevice.getStreamingAddress()
      } else {
-------------------------------------------------------------------------
|         String srsIp = configService.selectConfigByKey(URLConstant.SRS_IP
|         String srs_http_flv_port = configService.selectConfigByKey(URLCon
|         String uniqueTag = configService.selectConfigByKey(URLConstant.IN
|         String uniqueCode = pcMonitoringDevice.getUniqueCode(); //通道唯一
|         String flvStreamUrl =  "http://" + srsIp + ":" + srs_http_flv_por
|         if (!Objects.equals(pcMonitoringDevice.getIframeUrl(), "") && pcM
|!pcMonitoringDevice.getIframeUrl().trim().isEmpty()) {
|             flvStreamUrl = pcMonitoringDevice.getIframeUrl();
|         }
|         monitorInfo.setStreamUrl(flvStreamUrl);
-------------------------------------------------------------------------
      }
```

```
        logger.info("==================当前monitorInfo->{}", monitorInfo);
        list1.add(monitorInfo);
    }
    return list1;
}
```

头

> guardian_person_duty,valve_inspection_action,rear_car_patrol_person,truck_head,baffle,people

安全监护人：guardian_person_duty 装卸前阀门检查动作:valve_inspection_action 车后侧巡视人员：rear_car_patrol_person 车头：truck_head 挡车牌：baffle 人(除红色和蓝色衣服的人员）：people

车尾

> guardian_person_duty,connecte_pipes_correct,pipes,ladder,safety_sling,pipes_discharge_action,rear_car_patrol_person,manhole_cover_close,manhole_cover_open,ts_manhole_cover_open,people

安全监护人：guardian_person_duty 鹤管连接正确：connecte_pipes_correct 鹤管：pipes 悬梯：ladder 安全绳：safety_sling 鹤管排放动作：pipes_discharge_action 车后侧巡视人员：rear_car_patrol_person 人孔盖关闭：manhole_cover_close 人孔盖开启：manhole_cover_open 人孔盖开启（特殊）：ts_manhole_cover_open 人(除红色和蓝色衣服的人员）：people

345 0 2023-09-24 01:07:18 1 2023-09-24 02:02:37 1 103 中化车头检测 zh_chetou_1 0 3 $^{199}/_{199}$ /usr/local/server/aipass/uploadPath/model/zh_chetou_1 guardian_person_duty,valve_inspection_action,rear_car_patrol_person,truck_head,baffle,people 50 16 200 0 1 0 640 yolov5m6 344 0 2023-09-24 00:56:40 1 2023-09-24 02:01:51 1 103 中化车尾检测 zh_chewei_1 4 $^{183}/_{199}$ /usr/local/server/aipass/uploadPath/model/zh_chewei_1 guardian_person_duty,connecte_pipes_correct,pipes,ladder,safety_sling,pipes_discharge_action,rear_car_patrol_person,manhole_cover_close,manhole_cover_open,ts_manhole_cover_open,people 50 16 200 1 1 0 640 yolov5m6

ts_manhole_cover_open,ladder,connecte_pipes_correct,rear_car_patrol_person

空白文档

AI盒子 一键部署操作文档

AI盒子 一键部署操作文档

第一步

环境检测

> 使用 `docker -v` 命令进行查看是否具有 `docker` 环境

```
root@nvidia-desktop:~# docker -v
Docker version 20.10.7, build 20.10.7-0ubuntu5~18.04.3
```

如果没有docker环境，可自行进行安装。

第二步

拷贝镜像**tar**文件

```
cd /
#创建文件夹
mkdir aibox
#移动文件到指定目录
cp /media/nvidia/XXXX硬盘名/aibox-deploy.tar /aibox
```

第三步

将**tar**包导成镜像

```
cd aibox
#将tar包导成镜像
docker load -i xxxxxx.tar
#查看镜像
root@nvidia-desktop:/aibox# docker images
REPOSITORY     TAG       IMAGE ID       CREATED              SIZE
aibox-deploy   1.0.3     a049726fa4eb   About a minute ago   14.5GB
root@nvidia-desktop:/aibox#
```

第四步：

移动模型文件

> 该模型名称应该与数据库中的模型名称对应上，如果对应不上，待项目启动后，进入系统进行更改，并重启平台

```
cd aibox
mkdir uploadPath
cd uploadPath
mkdir model_store
cd model_store
```

```
#将模型移到该目录小
#注意：模型的最终文件形式为  /aibox/uploadPath/xxxx/best.pt

aibox
  └── uploadPath
      └── model_store
          ├── common                                          #模型名
          │       └── best.pt                                #模型文件
          ├── mask                                                    #模
          │       └── best.pt                                #模型文件
          └── safety_hat                                        #模型名
                  └── best.pt                                 #模型文件
```

第五步

启动镜像

启动平台镜像 `docker run -idt --name [自定义容器名称] --runtime nvidia --net=host [镜像名称]:[镜像版本号] bash`

查看容器启动日志： `docker logs -f [容器名]` 查看日志 例子： `docker run -idt --name aibox1.0.3 --runtime nvidia --net=host aibox-deploy:1.0.3 bash`

具体操作如下：

```
root@nvidia-desktop:/aibox# docker run -idt -v /aibox/uploadPath:/usr/local
2cee2b57982796c9854830558b1361f42e3560dc173e0423b52209806cfece70    #出现这个
root@nvidia-desktop:/aibox# docker logs -f aibox
正在清理异常退出所占用的PID文件...
清理完成...
正在清理异常退出导致的MYSQL LOCK占用的文件...
清理完成...
启动redis服务...
Starting Redis server...
Redis is running...
修改Mysql目录权限....
修改mysql用户信息....
usermod: no changes
启动mysql服务...
 * Stopping MySQL database server mysqld                              [
 * Starting MySQL database server mysqld                              [
启动SRS服务...
启动大屏服务...
Using CATALINA_BASE:   /usr/local/server/apache-tomcat
Using CATALINA_HOME:   /usr/local/server/apache-tomcat
Using CATALINA_TMPDIR: /usr/local/server/apache-tomcat/temp
Using JRE_HOME:        /usr
Using CLASSPATH:       /usr/local/server/apache-tomcat/bin/bootstrap.jar:/u
Tomcat started.
启动AI平台....
Start /usr/local/server/aicsp/aicsp-ipc.1.0.0.jar success...

#出现以上日志内容便表示该容器中的服务已经处于启动的状态
```

第六步：

修改IP信息

- 获取本机的IP地址 ： `ifconfig`
- 登录到平台修改参数：
    - 系统管理 –> 参数管理
- 修改以下参数的IP为获取到的本机的IP
    - `big_screen_url_admin` #管理员大屏访问地址
    - `big_screen_url` #大屏访问地址
    - `screen_url` #大屏缓存数据图片链接
    - `srs_ip` #SRS推流服务IP

修改模型信息：

这一步只是针对于在上传模型的时候，名称与平台中模型标识不一致的信息，如果一直，省略该步骤

- AI能力管理 -> AI模型管理
- 新增或者编辑 【该步骤的信息，与第四步中的模型信息相匹配】

> 注意：这个时候其实并完成项目的部署
>
> - 项目的视频流并没有进行推流
> - 项目的模型并没有一定启动

第六步

设置开机启动

```
vim /etc/rc.local
#新增以下内容
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#!/bin/bash
sleep 1
sudo docker start aibox1 >/aibox/rc.log 2>&1 &
exit 0
```

> 注意：
>
> 如果系统为Ubuntu18,该方法将不生效。操作方法如下：

ubuntu 18.04 不再使用 inited 管理系统，改用 systemd

第一步：

```
ln -fs /lib/systemd/system/rc-local.service /etc/systemd/system/rc-local.se
```

第二步：

```
sudo echo "
[Install]
WantedBy=multi-user.target
Alias=rc-local.service
" >> /etc/systemd/system/rc-local.service
```

第三步：

```
chmod +x /etc/rc.local
```

第七步

重启机器测试

```
#直接命令行输入 以下命令进行重启机器
reboot
```

机器重启后，所有的设置才会真正生效，在配置正确的情况下，服务会正常运行。

其他命令

> 重启项目：
>
> 命令行输入： `docker restart aibox`

设置**docker**服务检测

```
#!/bin/sh
#Filename: check-docker-service.sh


PROC_NAME=dockerd
ProcNumber=`ps -ef |grep -w $PROC_NAME|grep -v grep|wc -l`
time1=$(date)
if [ $ProcNumber -le 0 ];then
   echo "$time1 :Docker service is not run" >> /root/check-docker-service.l
   systemctl start docker
   echo "$time1 :command:{ systemctl start docker } is execute" >> /root/ch

else
   echo "$time1 :Docker service is running.."  >> /root/check-docker-servic
fi
```

设置定时任务执行上面的脚本

```
sudo su #切换root用户添加crontab    否则添加的任务会没有权限

crontab -e
#添加下面的命令    每5分钟执行一次
*/5  * * * * sh /ai_server/dockerPackage/check-docker-service.sh
```

1、安装Mysql

> 先更新apt源

```
> vim /etc/apt/source.list
>
> deb http://mirrors.aliyun.com/ubuntu bionic main multiverse restrict
> deb http://mirrors.aliyun.com/ubuntu bionic-updates main multiverse
> deb http://mirrors.aliyun.com/ubuntu bionic-security main multiverse
> deb http://mirrors.aliyun.com/ubuntu bionic-proposed main multiverse
> deb http://mirrors.aliyun.com/ubuntu bionic-backports main multivers
> ```

1、安装Mysql

1.1 安装

```shell
apt update
apt install mysql-server
service mysql start
```

1.2 设置连接

1. 进入mysql容器：docker exec -it 容器id /bin/bash （退出：extc）
2. `mysql -uroot -p` ,回车输入密码进入mysql操作
3. 看当前所有数据库： `show databases;`
4. 进入mysql数据库： `use mysql;`
5. 查看mysql数据库中所有的表： `show tables;`
6. 查看user表中的数据： `select Host, User from user;`
7. 修改user表中的Host: `update user set Host='%' where User='root';`
   说明： % 代表任意的客户端,可替换成具体IP地址。
8. 注意：一定要记得在写sql的时候要在语句完成后加上”;“
9. 修改密码: `ALTER USER 'root'@'%' IDENTIFIED WITH mysql_native_password BY 'PcAiPass@2021!';`

注意： 如果遇到报错信息： **`mysql> alter user 'root'@'%' identified with mysql_native_password by 'your password'; ERROR 1396 (HY000): Operation ALTER USER failed for 'root'@'%'`**

请执行： **`flush privileges;`**

1. 最后刷新一下： `flush privileges;`
2. 修改针对于orderby的报错问题
```

```
vim /etc/mysql/mysql.conf.d/mysqld.cnf


sql_mode=STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIV
```

3. 设置大小写

```
vim /etc/mysql/mysql.conf.d/mysqld.cnf
#在【mysqld】下面增加一行：
lower_case_table_names = 1
```

4. 重启mysql

```
service mysql restart
```

> ### mysql启动报错【No directory, logging in with HOME=/】

```
 usermod -d /var/lib/mysql/ mysql#第一步
ln -s /var/lib/mysql/mysql.sock /tmp/mysql.sock#第二步
chown -R mysql:mysql /var/lib/mysql#第三步
#之后重启mysql即可
```

2、安装Redis（5.0版本以上）

帮助连接

2.1 安装

```
cd /usr/local/server
wget https://download.redis.io/releases/redis-7.0.3.tar.gz
tar -zxvf redis-7.0.3.tar.gz
cd redis-7.0.3
make && install
```

2.2 修改配置

```
cd /usr/local/server/redis-7.0.3
vim redis.conf

################################################
# daemonize  由 no 修改为 yes    设置为后台执行
################################################
daemonize yes

################################################
#bind  由 bind 127.0.0.1 -::1 修改为 bind 0.0.0.0
################################################
bind 0.0.0.0
```

2.3 设置远程连接

```
cd /usr/local/server/redis-7.0.3
vim redis.conf
#########################################################
#protected-mode yes 修改为 no
```

```
#这个对Ubuntu有影响，会在远程访问的时候要求必须有密码，redis默认不用密码，所以记得先设置
#######################################################
protected-mode no
```

> 解决linux下redis数据库overcommit_memory问题 以及

```
vim /etc/sysctl.conf

####
#新增以下内容
####
#内核参数overcommit_memory
#可选值：0、1、2。
#0， 表示内核将检查是否有足够的可用内存供应用进程使用；如果有足够的可用内存，内存申请允许
#1， 表示内核允许分配所有的物理内存，而不管当前的内存状态如何。
#2， 表示内核允许分配超过所有物理内存和交换空间总和的内存
vm.overcommit_memory = 1

#Deal with : WARNING: The TCP backlog setting of 511 cannot be enforced bec
net.core.somaxconn= 1024
```

> 解决： WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command ‘echo never > /sys/kernel/mm/transparent_hugepage/enabled’ as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled

```
vim /etc/rc.local
#关闭 redis THP
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

3、部署SRS服务

1、下载源码

```
git clone -b 4.0release https://gitee.com/ossrs/srs.git
```

2、编译

> 注意需要切换到 `srs/trunk` 目录

```
cd srs/trunk
./configure
make
```

3、修改端口号

3.1 修改配置文件

```
vim /usr/local/server/srs/trunk/conf/srs.conf
```

修改 `http_server` 配置项的 `listen` 端口号为： `9090`

```
#main config for srs.
#@see full.conf for detail config.

listen              1935;
max_connections     1000;
#srs_log_tank        file;
#srs_log_file        ./objs/srs.log;
daemon              on;
http_api {
    enabled         on;
    listen          1985;
}
http_server {
    enabled         on;
    listen          9090;
    dir             ./objs/nginx/html;
}
rtc_server {
    enabled on;
    listen 8000; # UDP port
    # @see https://github.com/ossrs/srs/wiki/v4_CN_WebRTC#config-candidate
    candidate $CANDIDATE;
}
vhost __defaultVhost__ {
    hls {
        enabled         on;
    }
    http_remux {
        enabled     on;
        mount       [vhost]/[app]/[stream].flv;
    }
    rtc {
        enabled     on;
        # @see https://github.com/ossrs/srs/wiki/v4_CN_WebRTC#rtmp-to-rtc
        rtmp_to_rtc off;
        # @see https://github.com/ossrs/srs/wiki/v4_CN_WebRTC#rtc-to-rtmp
        rtc_to_rtmp off;
    }
}
```

3.2 修改页面端口号

修改 `http` 的端口号为 `9090`

```
vim /usr/local/server/srs/trunk/objs/nginx/html/console/js/srs.console.js
 ################################################################################
 614 // the sc server is the server we connected to.
 615 scApp.provider("$sc_server", [function(){
 616     this.$get = function(){
 617         var self = {
 618             schema: "http",
 619             host: null,
 620             port: 1985,
 621             rtmp: [1935],
 622             http: [8081],
 623             baseurl: function(){
```

```
624                 return self.schema + "://" + self.host + (self.port ==
625             },
626             jsonp: function(url){
```

3.3 修改主页跳转js

```
vim /usr/local/server/srs/trunk/objs/nginx/html/index.html
####################
#将 parseInt(window.location.port)
####################
50         // Build console url.
51         if (true) {
52             // The prefix for default website.
53             const prefix = `${window.location.protocol}//${window.locat
54             // If not 8080, user should proxy to the default port.
55             const query = parseInt(window.location.port) === 8081 ? `?p
56             const enUrl = `${prefix}/console/en_index.html#/summaries${
57             const cnUrl = `${prefix}/console/ng_index.html#/summaries${
58             document.getElementById("enConsole").setAttribute('href', e
59             document.getElementById("cnConsole").setAttribute('href', c
60         }
61
62         // The player url.
63         if (true) {
64             const prefix = `players/?schema=${window.location.protocol.
65             const httpPort = window.location.port || (window.location.p
66             // If not 8080, user should proxy both stream and API to th
67             const query = parseInt(window.location.port) === 8081 ? ''
68             document.getElementById("enPlayer").setAttribute('href', `$
69             document.getElementById("cnPlayer").setAttribute('href', `$
70         }
```

4、启动服务器：

```
nohup ./objs/srs -c conf/srs.conf >/dev/null 2>&1 &
```

5、检查SRS启动情况

http://localhost:9090/，或者执行命令

```
#查看SRS的状态
./etc/init.d/srs status
MB0:trunk $ ./etc/init.d/srs status
SRS(pid 90408) is running.                                  [  OK  ]
#如果异常    查看日志排查

#或者看SRS的日志
tail -n 30 -f ./objs/srs.log




查看状态

./etc/init.d/srs status
停止
```

```
./etc/init.d/srs stop
查看srs版本

./objs -v
```

4、安装JAVA

```
apt-get install openjdk-8-jre-headless git wget curl -y
```

5、部署大屏项目

1、上传安装包到服务器到**/usr/local/server**目录下

apache-tomcat-8.5.43.tar.gz

或者从备份服务器下载安装包

```
cd /usr/local/server
wget http://58.220.24.92:8091/aicsp/profile/file/apache-tomcat-8.5.43.tar.
```

2、解压并更改**tomcat**名称

```
tar -zxvf apache-tomcat-8.5.43.tar.gz
mv apache-tomcat-8.5.43 apache-tomcat
```

3、修改端口为**8888**和修改配置文件**setclasspath.sh**

```
vim /usr/local/server/apache-tomcat/conf/server.xml
修改端口8080 -> 8888
cd /usr/local/server/apache-tomcat/bin
vim setclasspath.sh
增加JAVA_HOME和JRE_HOME两个环境变量，环境变量路径即为jdk安装路径
#-----------------------------------------------------------------------
#Set JAVA_HOME or JRE_HOME if not already set, ensure any provided settings
#are valid and consistent with the selected start-up options and set up the
#endorsed directory.
export JAVA_HOME=/usr/local/java
export JAVA_HOME=/usr/local/java/jre
#-----------------------------------------------------------------------
```

4、清除**tomcat**下的其他部署包

```
cd /usr/local/server/apache-tomcat/webapps/
rm -rf /usr/local/server/apache-tomcat/webapps/*   （谨慎删除，判断是否在webap
```

5、上传安装包到服务器到**/usr/local/server/apache-tomcat/webapps/**目录

dist.zip

或者从备份服务器下载安装包

```
cd /usr/local/server/apache-tomcat/webapps/
wget http://58.220.24.92:8091/aicsp/profile/file/dist.zip
```

6、解压大屏项目并删除**dist.zip**

```
unzip dist.zip
rm -rf dist.zip
```

7、启动大屏项目

```
/usr/local/server/apache-tomcat/bin/startup.sh
```

6、部署AI音视频融合平台

6.1 新增字体

```
cd /usr/share/fonts/
mkdir chinese
cd chinese
wget http://58.220.24.92:8091/aicsp/profile/file/SIMSUN.TTC

#安装字体
#如没有这个命令  请执行下面的步骤
mkfontscale
mkfontdir
#刷新系统字体缓存即刻生效
fc-cache -fv
#检查字体是否添加成功
fc-list | grep 宋体
```

> **bash: mkfontscale:** 未找到命令

```
#使mkfontscale和mkfontdir命令正常运行
sudo apt-get install ttf-mscorefonts-installer
#使fc-cache命令正常运行
sudo apt-get install fontconfig
```

7、迁移算法

8、打包docker镜像

8.1 编写一键启动脚本

```
cd /
mkdir aibox
vim /aibox/service_start.sh
##############################service_start.sh#################################
#!/bin/bash
sleep 1

FILEPATH=/var/run/redis_6379.pid
MYSQLFILEPATH=/var/run/mysqld/mysqld.sock.lock

```

```
if [[ -f "$FILEPATH" ]]; then
        echo "正在清理异常退出所占用的PID文件...";
        rm -f "$FILEPATH";
        echo "清理完成...";
fi;

#启动redis服务
echo "启动redis服务..."
service redis start

#启动mysql服务
#清理mysql程序异常退出导致的lock文件占用
if [[ -f "$MYSQLFILEPATH" ]]; then
        echo "存在mysql.sock.lock文件";
        echo "正在清理mysql.sock.lock文件"
        rm -f "$MYSQLFILEPATH";
        echo "清理完成...";
fi;

echo "修改Mysql目录权限...."
chown -R mysql:mysql /var/lib/mysql /var/run/mysqld

echo "修改mysql用户信息...."
#No directory, logging in with HOME=/
usermod -d /var/lib/mysql/ mysql
echo "启动mysql服务..."
service mysql start
#启动SRS服务
echo "启动SRS服务..."
cd /usr/local/server/srs/trunk/


nohup ./objs/srs -c ./conf/srs.conf >/dev/null 2>&1 &
#启动大屏服务
echo "启动大屏服务..."
/usr/local/server/apache-tomcat/bin/startup.sh
#启动AI平台
echo "启动AI平台...."
/usr/local/server/aicsp/ai-ipc.sh start

sleep 1


#如何关闭透明巨页（THP）?
echo never > /sys/kernel/mm/transparent_hugepage/enabled

exit 0
```

8.2 打包镜像

```
#停掉镜像
docker stop
#根据镜像打包
docker commit  -m '' 616fcc88d716 aibix:1.0.0
#根据Dockrfile打包镜像

################################################
#编写Dockerfile
cd aibox
```

```
vim Dockerfile

############################Dockerfile############################

FROM aibox:1.0.0

#定义时区参数
ENV TZ=Asia/Shanghai
#设置时区
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo '$TZ' > /etc/tim
#设置编码
ENV LANG C.UTF-8

COPY ./code/data_process1.0 /ai_server/code/data_process1.0   # 复制的是算法的
RUN chmod +x /ai_server/code/data_process1.0/video_app.py

#拷贝执行脚本
COPY ./service_start.sh /usr/local/server/service_start.sh
RUN chmod +x /usr/local/server/service_start.sh

#启动脚本
RUN chmod +x /usr/local/server/aicsp/ai-ipc.sh
ENTRYPOINT /usr/local/server/service_start.sh && tail -f /usr/local/server/


###################################执行打包命令#################################
docker build -t aibox:1.0.1 .
```

9、编辑监听docker服务的操作

9.1 设置docker服务检测

```
cd /ai_server/dockerPackage/  #目录可以随意设置
vim check-docker-service.sh
########################################check-docker-service.sh############
#!/bin/sh
#Filename: check-docker-service.sh

PROC_NAME=dockerd
ProcNumber=`ps -ef |grep -w $PROC_NAME|grep -v grep|wc -l`
time1=$(date)
if [ $ProcNumber -le 0 ];then
   echo "$time1 :Docker service is not run" >> /root/check-docker-service.l
   systemctl start docker
   echo "$time1 :command:{ systemctl start docker } is execute" >> /root/ch

else
   echo "$time1 :Docker service is running.."  >> /root/check-docker-servic
fi
```

9.2 设置定时任务执行上面的脚本

```
sudo su #切换root用户添加crontab    否则添加的任务会没有权限

crontab -e
```

```
#添加下面的命令    每5分钟执行一次
*/5  * * * * sh /ai_server/dockerPackage/check-docker-service.sh
```

9.3 每天重启docker容器

```
30 2 * * * docker restart aibox
```

```
#关闭docker服务
systemctl stop docker
systemctl stop docker.socket
#启动dockerfuwu
systemctl start docker
```

下方为案例：

redis启动脚本

```
wget https://download.redis.io/releases/redis-7.0.3.tar.gz
```

redis启动脚本

↓↓↓↓↓↓教程地址↓↓↓↓↓↓

> 解决linux下redis数据库overcommit_memory问题 以及

```
vim /etc/sysctl.conf

####
# 新增以下内容
####
#内核参数overcommit_memory
#可选值：0、1、2。
#0， 表示内核将检查是否有足够的可用内存供应用进程使用；如果有足够的可用内存，内存申请允许
#1， 表示内核允许分配所有的物理内存，而不管当前的内存状态如何。
#2， 表示内核允许分配超过所有物理内存和交换空间总和的内存
vm.overcommit_memory = 1

#Deal with : WARNING: The TCP backlog setting of 511 cannot be enforced bec
net.core.somaxconn= 1024
```

> 解决： WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command ‘echo never > /sys/kernel/mm/transparent_hugepage/enabled’ as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled

```
vim /etc/rc.local

#关闭 redis THP
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

1、下载源码

1、下载源码

```
git clone -b 4.0release https://gitee.com/ossrs/srs.git
```

2、编译

> 注意需要切换到 `srs/trunk` 目录

```
cd srs/trunk
./configure
make
```

3、修改端口号

3.1 修改配置文件

```
vim /usr/local/server/srs/trunk/conf/srs.conf
```

修改 `http_server` 配置项的 `listen` 端口号为： `9090`

```
# main config for srs.
# @see full.conf for detail config.

listen              1935;
max_connections     1000;
#srs_log_tank        file;
#srs_log_file        ./objs/srs.log;
daemon              on;
http_api {
    enabled         on;
    listen          1985;
}
http_server {
    enabled         on;
    listen          9090;
    dir             ./objs/nginx/html;
}
rtc_server {
    enabled on;
    listen 8000; # UDP port
    # @see https://github.com/ossrs/srs/wiki/v4_CN_WebRTC#config-candidate
    candidate $CANDIDATE;
}
vhost __defaultVhost__ {
    hls {
        enabled         on;
    }
    http_remux {
        enabled     on;
        mount       [vhost]/[app]/[stream].flv;
    }
    rtc {
        enabled     on;
        # @see https://github.com/ossrs/srs/wiki/v4_CN_WebRTC#rtmp-to-rtc
```

```
        rtmp_to_rtc off;
        # @see https://github.com/ossrs/srs/wiki/v4_CN_WebRTC#rtc-to-rtmp
        rtc_to_rtmp off;
    }
}
```

3.2 修改页面端口号

修改 `http` 的端口号为 `9090`

```
vim /usr/local/server/srs/trunk/objs/nginx/html/console/js/srs.console.js
 ##############################################################################
 614 // the sc server is the server we connected to.
 615 scApp.provider("$sc_server", [function(){
 616     this.$get = function(){
 617         var self = {
 618             schema: "http",
 619             host: null,
 620             port: 1985,
 621             rtmp: [1935],
 622             http: [8081],
 623             baseurl: function(){
 624                 return self.schema + "://" + self.host + (self.port ==
 625             },
 626             jsonp: function(url){
```

3.3 修改主页跳转**js**

```
 vim /usr/local/server/srs/trunk/objs/nginx/html/index.html
####################
# 将 parseInt(window.location.port)
####################
 50         // Build console url.
 51         if (true) {
 52             // The prefix for default website.
 53             const prefix = `${window.location.protocol}//${window.locat
 54             // If not 8080, user should proxy to the default port.
 55             const query = parseInt(window.location.port) === 8081 ? `?p
 56             const enUrl = `${prefix}/console/en_index.html#/summaries${
 57             const cnUrl = `${prefix}/console/ng_index.html#/summaries${
 58             document.getElementById("enConsole").setAttribute('href', e
 59             document.getElementById("cnConsole").setAttribute('href', c
 60         }
 61
 62         // The player url.
 63         if (true) {
 64             const prefix = `players/?schema=${window.location.protocol.
 65             const httpPort = window.location.port || (window.location.p
 66             // If not 8080, user should proxy both stream and API to th
 67             const query = parseInt(window.location.port) === 8081 ? ''
 68             document.getElementById("enPlayer").setAttribute('href', `$
 69             document.getElementById("cnPlayer").setAttribute('href', `$
 70         }
```

4、启动服务器：

```
nohup ./objs/srs -c conf/srs.conf >/dev/null 2> &1 &
```

5、检查SRS启动情况

http://localhost:9090/，或者执行命令

```
# 查看SRS的状态
./etc/init.d/srs status
MB0:trunk $ ./etc/init.d/srs status
SRS(pid 90408) is running.                                  [  OK  ]
#如果异常    查看日志排查

# 或者看SRS的日志
tail -n 30 -f ./objs/srs.log
```

开启**webrtc**

apt阿里云源

```
vim /etc/apt/source.list
deb http://mirrors.aliyun.com/ubuntu bionic main multiverse restricted univ
deb http://mirrors.aliyun.com/ubuntu bionic-updates main multiverse restric
deb http://mirrors.aliyun.com/ubuntu bionic-security main multiverse restri
deb http://mirrors.aliyun.com/ubuntu bionic-proposed main multiverse restri
deb http://mirrors.aliyun.com/ubuntu bionic-backports main multiverse restr
```

git文件忽略

git文件忽略

```
### IntelliJ IDEA ###
.idea/*
*.iws
*.iml
*.ipr
*.log
*.class
*/target/*
```

空白文档

菜单

> 综合安防平台数据接入平台功能
>
> - 添加 `综合安防平台对接表` 表
> - 添加 `综合安防平台设备表` 表
> - 添加菜单
> - 修改 `pc_monitoring_device` 表 `imei` 字段长度，添加 `security_id` 关联字段
>
> 添加参数配置
>
> - 添加 `data_report_state` 参数: `true` ：数据上报， `false` ：数据不上报
> - 添加 `data_report_to_intranet_url` ：上报地址（内网）
> - 添加 `data_report_to_extranet_url` ：上报地址（外网）
> - 添加 `data_report_type` ： `intranet` ：推送内网， `extranet` ：推送外网
> - 添加外网推送方式的用户名密码： `data_report_for_username` ：用户名， `data_report_for_password` ：密码
> - 添加 `data_report_for_auth_url` 获取token的地址

```
CREATE TABLE `pc_monitoring_device_security` (
  `id` int NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `ip` varchar(20) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT 'IP',
  `port` varchar(8) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '端口号',
  `app_key` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT 'appKe
  `app_secret` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT 'ap
  `descript` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
  `del_flag` int DEFAULT '0' COMMENT '删除标记',
  `create_id` int DEFAULT NULL COMMENT '创建人ID',
  `create_date` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_c
  `update_id` int DEFAULT NULL COMMENT '更新人ID',
  `update_date` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_c
  `dept_id` int DEFAULT NULL COMMENT '部门ID',
  `job_id` varchar(100) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci DE
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=12 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_u

CREATE TABLE `pc_monitoring_security_info` (
  `id` int NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `ip` varchar(20) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMENT '设备IP'
  `port` varchar(5) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMENT '端口号'
  `device_type` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMEN
  `dev_serial_num` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL CON
  `manufacturer` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMME
  `camera_index_code` varchar(100) COLLATE utf8mb4_unicode_ci NOT NULL COMM
```

```
  `camera_name` varchar(100) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMEN
  `camera_type_name` varchar(100) COLLATE utf8mb4_unicode_ci DEFAULT NULL C
  `channel_no` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMENT
  `channel_type_name` varchar(100) COLLATE utf8mb4_unicode_ci DEFAULT NULL
  `encode_dev_index_code` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT N
  `encode_dev_resource_type_name` varchar(100) COLLATE utf8mb4_unicode_ci D
  `region_index_code` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL
  `security_id` int NOT NULL COMMENT '综合安防对接ID',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci COMMENT=

##菜单

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('综合安防平台对接', '3000', '1', '/pc/monitoring_device_security', 'C'

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('综合安防平台对接查询', @parentId, '1',  '#',  'F', '0', 'pc:monitoring

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('综合安防平台对接新增', @parentId, '2',  '#',  'F', '0', 'pc:monitoring

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('综合安防平台对接修改', @parentId, '3',  '#',  'F', '0', 'pc:monitoring

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('综合安防平台对接删除', @parentId, '4',  '#',  'F', '0', 'pc:monitoring

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('综合安防平台对接导出', @parentId, '5',  '#',  'F', '0', 'pc:monitoring

#修改通道号长度
ALTER TABLE `pc_monitoring_device`
MODIFY COLUMN `imei` varchar(50) CHARACTER SET utf8mb3 COLLATE utf8mb3_gene

#添加security_id
ALTER TABLE `pc_monitoring_device`
ADD COLUMN `security_id` int DEFAULT '0' COMMENT 'securityId'  AFTER `addre

#添加参数
INSERT INTO `sys_config` (`config_name`, `config_key`, `config_value`, `con
INSERT INTO `sys_config` (`config_name`, `config_key`, `config_value`, `con
INSERT INTO `sys_config` (`config_name`, `config_key`, `config_value`, `con
INSERT INTO `sys_config` (`config_name`, `config_key`, `config_value`, `con
INSERT INTO `sys_config` (`config_name`, `config_key`, `config_value`, `con
INSERT INTO `sys_config` (`config_name`, `config_key`, `config_value`, `con
INSERT INTO `sys_config` (`config_name`, `config_key`, `config_value`, `con
```

项目更新

> 基于 `V2.0.3.1` 更新

```
docker pull registry.cn-hangzhou.aliyuncs.com/jeffrey_ai/aibox:v2.1.0.1
```

综合安防平台接口优化

```
package com.ruoyi.common.utils.security;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
import com.hikvision.artemis.sdk.ArtemisHttpUtil;
import com.hikvision.artemis.sdk.config.ArtemisConfig;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.*;

public class ArtemisPost {
    private static final Logger log = LoggerFactory.getLogger(ArtemisPost.c
    public static String host;
    public static String appKey;
    public static String appSecret;
    public static String type;
    public static ArtemisConfig artemisConfig;
    /**
     * API网关的后端服务上下文为：/artemis
     */
    private static final String ARTEMIS_PATH = "/artemis";

    public ArtemisPost(String host, String appKey, String appSecret){
        ArtemisConfig config = new ArtemisConfig();
        config.setHost(host); // 代理API网关nginx服务器ip端口
        config.setAppKey(appKey);  // 密钥appkey
        config.setAppSecret(appSecret);// 密钥appSecret
        artemisConfig = config;
    }

    /**
     * 获取所有的监控列表
     * @param pageNo 页数
     * @param pageSize 每页条数
     * @return String
     */

    public String  getAllCameras(Integer pageNo, Integer pageSize) {
        final String  getCamsApi = ARTEMIS_PATH + "/api/resource/v1/cameras
        Map<String, String> paramMap = new HashMap<>();// post请求参数
        paramMap.put("pageNo", pageNo.toString());
        paramMap.put("pageSize", pageSize.toString());
        String body = JSON.toJSON(paramMap).toString();
        return executeArtemisPost(getCamsApi, body);
    }

    /**
     * 根据区域获取下级编码设备列表
     *
     * @param regionIndexCode 区域编号
     * @return String
     */

    public String getRegionDeviceList(Integer pageNo, Integer pageSize, Str
        final String getCamsApi = ARTEMIS_PATH + "/api/resource/v1/encodeDe
```

```
        Map<String, Object> paramMap = new HashMap<>();// post请求参数
        List<String> list = new ArrayList<>();
        paramMap.put("regionIndexCode", regionIndexCode);
        paramMap.put("pageNo", pageNo.toString());
        paramMap.put("pageSize", pageSize.toString());
        list.add("view");
        paramMap.put("authCodes", list);
        String body = JSON.toJSON(paramMap).toString();
        return executeArtemisPost(getCamsApi, body);
    }

    /**
     * 名称查询监控列表
     *
     * @param pageNo   页数
     * @param pageSize 每页条数
     * @return String
     */

    @SuppressWarnings("all")
    public String getAllCamerasByName(Integer pageNo, Integer pageSize, Str
        final String getCamsApi = ARTEMIS_PATH + "/api/resource/v2/camera/s
        Map<String, String> paramMap = new HashMap<>();// post请求参数
        paramMap.put("pageNo", pageNo.toString());
        paramMap.put("pageSize", pageSize.toString());
        paramMap.put("name", cameraName.toString());
        paramMap.put("orderBy", "name");
        paramMap.put("orderType", "desc");
        String body = JSON.toJSON(paramMap).toString();
        return executeArtemisPost(getCamsApi, body);
    }

    /**
     * 获取设备流地址信息
     *
     * @param cameraIndexCode 摄像头编号
     * @param protocol        rtsp
     * @return String
     */
    public String getCamerasRtsp(String cameraIndexCode, String protocol) {
        String  getCamsApi = ARTEMIS_PATH + "/api/vnsc/mls/v1/preview/openA
        Map<String, Object> paramMap = new HashMap<>();// post请求参数
        paramMap.put("indexCode", cameraIndexCode);
        paramMap.put("protocol", protocol);
        paramMap.put("expireTime", -1);
        if (protocol.equals("rtsp")){
            paramMap.put("expand","streamform=rtp&transcode=1");
        }
        String body = JSON.toJSON(paramMap).toString();
        return executeArtemisPost(getCamsApi,body);
    }
//paramMap.put("expand","streamform=rtp&transcode=1");
    /**
     * 该接口用于手动触发设备抓图，返回图片的地址，抓图前请确保平台上已配置图片存储信息。
     * @param cameraIndexCode 摄像头唯一标识
     * @return 图片路径
     */
    public String manualCapture(String cameraIndexCode){
        final String  getCamsApi = ARTEMIS_PATH + "/api/video/v1/manualCapt
        Map<String, String> paramMap = new HashMap<>();// post请求参数
```

```
        paramMap.put("cameraIndexCode", cameraIndexCode);
        String body = JSON.toJSON(paramMap).toString();
        return executeArtemisPost(getCamsApi, body);
    }

    /**
     * 获取设备信息
     * @param cameraIndexCode 资源编号
     * @return String
     */
    public String getCameraInfoByIndexCode(String cameraIndexCode){
        final String  getCamsApi = ARTEMIS_PATH + "/api/resource/v1/cameras
        Map<String, String> paramMap = new HashMap<>();// post请求参数
        paramMap.put("cameraIndexCode", cameraIndexCode);
        String body = JSON.toJSON(paramMap).toString();
        return executeArtemisPost(getCamsApi,body);
    }


    /**
     * 执行Artemis的POST请求
     */
    private String executeArtemisPost(String apiPath, String body) {
        Map<String, String> path = Collections.singletonMap("https://", api
        try {
            return ArtemisHttpUtil.doPostStringArtemis(artemisConfig, path,
        } catch (Exception e) {
            log.error("Artemis POST请求异常: {}", e.getMessage());
            return null;
        }
    }
}
```

运管中心配置

运管中心配置

视频能力步骤

1、开发前准备

链接直达：环境准备

2、获取IP和端口

3、获取AppKey 和 appSecret

4、参考

分页获取监控点资源

获取监控点预览取流URLv2

> 1.平台正常运行；平台已经添加过设备和监控点信息。 2.平台需要安装mgc取流服务。 3.三方平台通过openAPI获取到监控点数据，依据自身业务开发监控点导航界面。 4.调用本接口获取预览取流URL，协议类型包括：hik、rtsp、rtmp、hls。 5.通过开放平台的开发包进行实时预览或者使用标准的GUI播放工具进行实时预览。
>
> |**6.**为保证数据的安全性，取流**URL**设有有效时间，有效时间为**5**分钟。 | 特别注意：获取到的视频流地址，5分钟之内需要进行访问，否则流地址会失效

海康威视OpenAPI对接系列3-使用JAVA demo

空白文档

2.0.3.1升级

```
ALTER TABLE `pc_ai_model`
ADD COLUMN `syn_status` tinyint(1) DEFAULT '4' COMMENT '同步状态 1 同步成功 2
ADD COLUMN   `syn_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP COMMENT '
ADD COLUMN   `resource_type` tinyint(1) DEFAULT '3' COMMENT '模型来源 1父平台
ADD COLUMN   `model_version` varchar(100) DEFAULT '' COMMENT '模型版本' AFTE
```

空白文档

sql

```
-- 车牌白名单
CREATE TABLE `pc_ocr_whitelist` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `number_plate` varchar(10) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '车
  `dept_name` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMENT
  `contact_person` varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMM
  `contact_phone_num` varchar(11) COLLATE utf8mb4_unicode_ci DEFAULT NULL C
  `del_flag` tinyint(1) NOT NULL DEFAULT '0' COMMENT '0 未删除  1 删除',
  `create_id` int(11) DEFAULT NULL COMMENT '创建人ID',
  `create_date` varchar(20) CHARACTER SET utf8 DEFAULT NULL COMMENT '创建时
  `update_id` int(11) DEFAULT NULL COMMENT '更新人ID',
  `update_date` varchar(20) CHARACTER SET utf8 DEFAULT NULL COMMENT '更新时
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci COMMENT=


-- 修改ocr结果信息表
ALTER TABLE `pc_ocr_lpr_result`
ADD COLUMN `type` tinyint(1) NOT NULL DEFAULT 1 COMMENT '1 非白名单  2：白名单
ADD COLUMN `belong_dept_name` varchar(255) CHARACTER SET utf8mb4 COLLATE ut
ADD COLUMN `contact_person` varchar(50) CHARACTER SET utf8mb4 COLLATE utf8m
ADD COLUMN `contact_phone_num` varchar(11) CHARACTER SET utf8mb4 COLLATE ut
DROP PRIMARY KEY,
ADD PRIMARY KEY (`id`) USING BTREE;

-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('Ocr车牌识别白名单', '3200', '1', '/pc/pc_ocr_whitelist', 'C', '0', 'p

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('Ocr车牌识别白名单查询', @parentId, '1',  '#',  'F', '0', 'pc:pc_ocr_wh

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('Ocr车牌识别白名单新增', @parentId, '2',  '#',  'F', '0', 'pc:pc_ocr_wh

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('Ocr车牌识别白名单修改', @parentId, '3',  '#',  'F', '0', 'pc:pc_ocr_wh

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('Ocr车牌识别白名单删除', @parentId, '4',  '#',  'F', '0', 'pc:pc_ocr_wh

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('Ocr车牌识别白名单导出', @parentId, '5',  '#',  'F', '0', 'pc:pc_ocr_wh
```

```
docker run  --env WVP_IP="192.168.1.14" -it -p 18080:18080 -p 30000-30500:3
```

sql

readme

```
-- app_model_main:模型主信息表
CREATE TABLE `app_model_main` (
  `id` int NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `name` varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL DEFAULT '' COMMENT
  `mark` varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL DEFAULT '' COMMENT
  `del_flag` int NOT NULL DEFAULT '0' COMMENT '删除标记',
  `create_time` datetime NOT NULL ON UPDATE CURRENT_TIMESTAMP COMMENT '创建
  `create_id` int NOT NULL COMMENT '创建人ID',
  `update_time` datetime DEFAULT NULL COMMENT '更新时间',
  `update_id` int DEFAULT NULL COMMENT '更新人ID',
  `dept_id` int DEFAULT NULL COMMENT '部门ID，用于数据权限',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci COMMENT=


-- 添加version_num字段
ALTER TABLE`app_model_main`
ADD COLUMN `version_num` int NOT NULL COMMENT '版本数量' AFTER `name`;


-- app_model: 新增字段   version basedOnVersion modelMainId

ALTER TABLE `app_model`
ADD COLUMN `version` varchar(255) NULL COMMENT '版本' AFTER `remark`,
ADD COLUMN `based_on_version` varchar(255) NULL COMMENT '所基于版本' AFTER `
ADD COLUMN `model_main_id` int NULL COMMENT '主模型ID' AFTER `based_on_versi


ALTER TABLE `app_model`
ADD COLUMN `version_create_rule` integer NULL DEFAULT 1 COMMENT '规则生成规则

ALTER TABLE `app_model`
ADD COLUMN `training_set_type` integer NULL DEFAULT 1 COMMENT '训练素材：1：主
```

> 火焰：/usr/local/server/aicsp/uploadPath/ffmpeg/video/2023/08/23/630c661e-7dec-4917-b186-3aaae88cea44.mp4

nohup ffmpeg -re -stream_loop -1 -i /usr/local/server/aicsp/uploadPath/ffmpeg/video/2023/08/23/630c661e-7dec-4917-b186-3aaae88cea44.mp4 -c copy -f flv rtmp://192.168.1.20:1985/online/fire &

ffmpeg -f dshow -i video="USB2.0 VGA UVC WebCam" -vcodec libx264 -acodec copy -preset:v ultrafast -tune:v zerolatency -f flv rtmp://39.97.214.170/live/livestream

```
<!DOCTYPE html>
<html lang="zh" xmlns:th="http://www.thymeleaf.org">
<head>
    <th:block th:include="include :: header('确认参数，开始训练')"/>
    <th:block th:include="include :: bootstrap-fileinput-css"/>
    <th:block th:include="include :: select2-css"/>
</head>
<body class="white-bg">
<div class="wrapper wrapper-content animated fadeInRight ibox-content">
```

```
    <form class="form-horizontal m" id="form-app_model-edit">
        <div class="form-group">
            <label class="col-sm-3 control-label"><h2>内存情况：</h2></label
        </div>
        <div class="form-group">
            <label class="col-sm-3 control-label">内存使用情况：</label>
            <div class="col-sm-9">
                <textarea style="height: 100px;" name="label" th:text="${in
                          readonly></textarea>
            </div>
        </div>
        <div class="form-group">
            <label class="col-sm-3 control-label"><h2>高级配置：</h2></label
        </div>
        <div class="form-group">
            <label class="col-sm-3 control-label">聚类训练迭代次数（30 ~ 50）：
            <div class="col-sm-3">
                <select name="clauculateK"
                        th:with="type=${@dict.getType('app_model_clauculate
                    <option th:each="dict : ${type}" th:text="${dict.dictLa
                            th:value="${dict.dictValue}"></option>
                </select>
            </div>
            <!--            <div class="col-sm-1">-->
            <!--                <input name="clauculateK" th:field="*{clauc
            <!--            </div>-->
            <label class="col-sm-3 control-label">迭代数据大小（1 ~ 128）：</l
            <div class="col-sm-3">
                <select name="batchSize" th:with="type=${@dict.getType('app
                        class="form-control">
                    <option th:each="dict : ${type}" th:text="${dict.dictLa
                            th:value="${dict.dictValue}"></option>
                </select>
                <!--                <input name="batchSize" th:field="*{bat
            </div>
        </div>
        <div class="form-group">
            <label class="col-sm-3 control-label">模型训练轮数：</label>
            <div class="col-sm-3">
                <select name="epochs" th:with="type=${@dict.getType('app_mo
                        class="form-control">
                    <option th:each="dict : ${type}" th:text="${dict.dictLa
                            th:value="${dict.dictValue}"></option>
                </select>
                <!--                <input name="epochs" th:field="*{epochs
            </div>
            <label class="col-sm-3 control-label">模型训练设备：</label>
            <div class="col-sm-3">
                <select id="trainDevice" name="trainDevice" class="form-con
                    <option th:each="gpuInfo : ${gpuInfos}" th:text="${gpuI
                            th:value="${gpuInfo.id}"></option>
                </select>
                <!--                <input name="trainDevice" th:field="*{t
            </div>
        </div>
        <div class="form-group">
            <label class="col-sm-3 control-label">模型训练图片大小：</label>
            <div class="col-sm-3">
                <select name="imageSize" th:with="type=${@dict.getType('app
                        class="form-control">
```

```
                    <option th:each="dict : ${type}" th:text="${dict.dictLa
                            th:value="${dict.dictValue}"></option>
                </select>
                <!--                <input name="imageSize" th:field="*{ima
            </div>
            <div id="trainCreateVersion">
                <label class="col-sm-3 control-label">模型训练生成版本：</labe
                <div class="col-sm-3">
                    <select name="modelVersion" th:with="type=${@dict.getTy
                        <option value="0" selected >默认空版本</option>
                        <option th:each="dict : ${type}" th:text="${dict.di
                                th:value="${dict.dictValue}"></option>
                    </select>
                    <!--                <input name="modelVersion" th:field
                </div>
            </div>

        </div>
        <div class="form-group">
            <label class="col-sm-3 control-label">模型部署进程数：</label>
            <div class="col-sm-3">
                <select name="workers" th:with="type=${@dict.getType('app_m
                        class="form-control">
                    <option th:each="dict : ${type}" th:text="${dict.dictLa
                            th:value="${dict.dictValue}"></option>
                </select>
                <!--                <input name="workers" th:field="*{worke
            </div>
            <label class="col-sm-3 control-label">模型部署设备：</label>
            <div class="col-sm-3">
                <select id="deployDevice" name="deployDevice" class="form-c
                    <option th:each="gpuInfo : ${gpuInfos}" th:text="${gpuI
                            th:value="${gpuInfo.id}"></option>
                </select>
                <!--                <input name="deployDevice" th:field="*{
            </div>
        </div>
        <div class="form-group">
            <label class="col-sm-3 control-label">训练基于版本：</label>
            <div class="col-sm-3">
                <select name="based_on_version" class="form-control" id="ba
                    <option selected="selected" value="-1">生成新版本</optio
                    <option th:each="appModelVersion : ${appModelVersions}"
                            th:value="${appModelVersion.id}"></option>
                </select>
            </div>
            <label class="col-sm-3 control-label">版本号生成规则：</label>
            <div class="col-sm-3">
                <div class="radio check-box">
                    <label><input type="radio" value="1" name="version_crea
                </div>
                <div class="radio check-box">
                    <label><input type="radio" value="2" name="version_crea
                </div>
                <!--                <input name="deployDevice" th:field="*{
            </div>
        </div>

        <div class="form-group">
            <label class="col-sm-3 control-label">训练集：</label>
```

```
            <div class="col-sm-3">
                <div class="radio check-box" id="currentDataSet">
                    <label><input type="radio" value="1" name="training_set
                </div>
                <div class="radio check-box" id="baseDataAdd">
                    <label><input type="radio" value="2" name="training_set
                </div>
                <div class="radio check-box" id="addData">
                    <label><input type="radio" value="3" name="training_set
                </div>
            </div>
        </div>
        <div class="form-group" id="trainData">
            <label class="col-sm-3 control-label">上传数据集：</label>
            <div class="col-sm-3">
                <input name="path" class="form-control" type="hidden">
                <div class="file-loading">
                    <input class="form-control file-upload trainData" accep
                </div>
            </div>
        </div>


    </form>
</div>
<th:block th:include="include :: footer"/>
<th:block th:include="include :: bootstrap-fileinput-js"/>
<th:block th:include="include :: select2-js"/>
<script th:inline="javascript">
    var prefix = ctx + "pc/app_model";
    $("#form-app_model-edit").validate({
        focusCleanup: true
    });

    function submitHandler() {
        if ($.validate.form()) {
            $.operate.save(prefix + "/edit", $('#form-app_model-edit').seri
        }
    }
    $(".file-upload").fileinput({
        uploadUrl: prefix + '/uploadImage',
        showClose: false, //是否显示关闭按钮
        dropZoneEnabled: false,//是否显示拖拽区域
        showPreview: false, //是否显示预览区域
        maxFileCount: 1,
        allowedFileExtensions: ['zip'],
        autoReplace: true
    }).on('fileuploaded', function (event, data, previewId, index) {
        $("input[name='" + event.currentTarget.id + "']").val(data.response
    }).on('fileremoved', function (event, id, index) {
        $("input[name='" + event.currentTarget.id + "']").val('')
    })

    $(function () {
        //查看当前基于训练版本
        const basedOnVersion = $("#basedOnVersion option:selected").val();
        if (basedOnVersion === "-1") {
            $("#currentDataSet").hide();
            $("#baseDataAdd").hide();
            $("#addData").show();
```

```
            $("#addData").children().find("input").attr("checked", true);
            $("#addData").children().find("input").parent().addClass('check
        }

        $('select[name="based_on_version"]').on('change', function (event)
            if ($(event.target).val() === "-1") {
                $("#currentDataSet").hide();
                $("#baseDataAdd").hide();
                $("#trainData").show();
                $("#addData").show();
                $("#addData").children().find("input").attr("checked", true
                $("#addData").children().find("input").parent().addClass('c
                $("#trainCreateVersion").show()
            } else {
                $("#currentDataSet").show();
                $("#trainData").hide();
                $("#currentDataSet").children().find("input").attr("checked
                $("#currentDataSet").children().find("input").parent().addC
                $("#baseDataAdd").show();
                $("#addData").hide();
                $("#trainCreateVersion").hide()
            }
        })


        $('input[name="training_set_type"]').on('ifChecked', function (even
            const trainingSetType = $(event.target).val();
            console.log(trainingSetType)
            if (trainingSetType === "1") {
                console.log("hidden")
                $("#trainData").hide();
            } else {
                console.log("show")
                $("#trainData").show();
            }
        });
    });

</script>
</body>
</html>
```


```
{"taskId":"17","videoId":"4c974d444f454b23afcf46b07b3a0f9a","imagePath":"/u
```



```
{"taskId":"17","videoId":"4c974d444f454b23afcf46b07b3a0f9a","imagePath":"/u
```



```
{
    "taskId":"17",
    "videoId":"4c974d444f454b23afcf46b07b3a0f9a",
    "imagePath":"/usr/local/server/aicsp/uploadPath/streamrecord/17/4c974d4
    "modelName":"common",
    "result":[
        {
            "category":"manhole_cover_open",
            "bbox":[
                603,
                266,
```

```
                697,
                321
            ],
            "score":0.9340547919273376
        },
        {
            "category":"person",
            "bbox":[
                946,
                0,
                1120,
                337
            ],
            "score":0.7641172409057617
        }
    ]
}
```


ffmpeg -rtsp_transport tcp -i “rtsp://admin:08021718.Tian@192.168.1.15:554/h264/ch1/main/av_stream_1” -vcodec libx264 -acodec aac -f flv -y “rtmp://192.168.1.11:1935/live/iiiiiiiii”

空白文档

sql

```
ALTER TABLE `pc_ai_model`
ADD COLUMN `model_mode` int(1) NULL DEFAULT 2 COMMENT '模式一二三' AFTER `mo
DROP PRIMARY KEY,
ADD PRIMARY KEY (`id`) USING BTREE
```

```
-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('设备组', '3320', '1', '/pc/pc_point', 'C', '0', 'pc:pc_point:view',

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('设备组查询', @parentId, '1',  '#',  'F', '0', 'pc:pc_point:list',

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('设备组新增', @parentId, '2',  '#',  'F', '0', 'pc:pc_point:add',

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('设备组修改', @parentId, '3',  '#',  'F', '0', 'pc:pc_point:edit',

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('设备组删除', @parentId, '4',  '#',  'F', '0', 'pc:pc_point:remove',

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('设备组导出', @parentId, '5',  '#',  'F', '0', 'pc:pc_point:export',



-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程记录', '3320', '1', '/pc/pc_point_record', 'C', '0', 'pc:pc_poin

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程记录查询', @parentId, '1',  '#',  'F', '0', 'pc:pc_point_record:l

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程记录新增', @parentId, '2',  '#',  'F', '0', 'pc:pc_point_record:a

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程记录修改', @parentId, '3',  '#',  'F', '0', 'pc:pc_point_record:e

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程记录删除', @parentId, '4',  '#',  'F', '0', 'pc:pc_point_record:r

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程记录导出', @parentId, '5',  '#',  'F', '0', 'pc:pc_point_record:e
```

```
select t.* from pc_monitoring_task_device d left join pc_monitoring_task t
on d.task_id = t.id
  <where>
      t.del_flag = 0
      and d.monitoring_device_id  = #{monitoringDeviceId}
  </where>
```

```
<!--        <div class="form-group">-->
<!--            <label class="col-sm-3 control-label is-required"> 流程关联通
<!--                : </label>-->
<!--            <div class="col-sm-5">-->
<!--                <input type="hidden" name="deviceSetting[]" th:attr="de
<!--                       class="process-step-device" value="-1">-->
<!--                <select name="deviceIds[]" id="process-step-device" cla
<!--                    <option th:each="pcMonitoringDevice : ${pcMonitorin
<!--                            th:value="${pcMonitoringDevice.id}"-->
<!--                            th:selected="${PcPointStepDevice1.deviceId}
<!--                </select>-->
<!--            </div>-->
<!--            <label class="device-opt control-label">-->
<!--                <a class="btn-sm btn-success" onclick="setROI('process-
<!--                    <i class="fa fa-cog"></i> 设置标记范围-->
<!--                </a>-->
<!--            </label>-->
<!--        </div>-->
<!--        <div class="add-new-device">-->
<!--            <div th:class="'form-group ' + ${PcPointStepDevice.id} + '-
<!--                 th:each="PcPointStepDevice : ${pcPointStepDevices}">-->
<!--                <label class="col-sm-3 control-label"> </label>-->
<!--                <div class="col-sm-5">-->
<!--                    <input type="hidden" name="deviceSetting[]" th:attr
<!--                           th:class="${PcPointStepDevice.id}" value="-1

<!--                    <select name="deviceIds[]" class="form-control devi
<!--                            th:id="${PcPointStepDevice.id}">-->
<!--                        <option th:each="pcMonitoringDevice : ${pcMonit
<!--                                th:text="${pcMonitoringDevice.name}" th
<!--                                th:selected="${PcPointStepDevice.device
<!--                        ></option>-->
<!--                    </select>-->

<!--                </div>-->
<!--                <label class="device-opt control-label">-->
<!--                    <a class="btn-sm btn-success" th:onclick="setROI([[
<!--                        <i class="fa fa-cog"></i> 设置标记范围-->
<!--                    </a>-->
<!--                    &lt;!&ndash;                    <a href="">设置标记范
<!--                </label>-->
<!--                <label class="device-opt control-label">-->
<!--                    <a class="btn-sm btn-danger" th:onclick="delDevice(
<!--                        <i class="fa fa-trash-o"></i> 删除-->
<!--                    </a>-->
<!--                </label>-->
<!--            </div>-->
<!--        </div>-->

<!--        <div>-->
<!--            <div class="form-group">-->
```

```
<!--                <label class="col-sm-3 control-label"> </label>-->
<!--                <div class="col-sm-8">-->
<!--                    <a class="btn dim btn-block" href="javascript:addNe
<!--                        <i class="fa fa-plus"></i>添加通道-->
<!--                    </a>-->
<!--                </div>-->
<!--            </div>-->
<!--        </div>-->


<!--        <div class="form-group">-->
<!--            <label class="col-sm-3 control-label is-required"> 是否必做-
<!--                : </label>-->
<!--            <div class="col-sm-8">-->
<!--                <div class="switch" style="display: flex">-->
<!--                    <div class="onoffswitch" style="float: left">-->
<!--                        <input type="hidden" name="mustFlag" th:value="
<!--                        <input type="checkbox" class="onoffswitch-check
<!--                        <label class="onoffswitch-label" for="mustFlag"
<!--                            <span class="onoffswitch-inner"></span>-->
<!--                            <span class="onoffswitch-switch"></span>-->
<!--                        </label>-->
<!--                    </div>-->
<!--                </div>-->
<!--            </div>-->
<!--        </div>-->

<!--        <div class="form-group">-->
<!--            <label class="col-sm-3 control-label is-required">结果是否存
<!--                : </label>-->
<!--            <div class="col-sm-8">-->
<!--                <div class="switch" style="display: flex">-->
<!--                    <div class="onoffswitch" style="float: left">-->
<!--                        <input type="hidden" name="targetExists" th:val
<!--                        <input type="checkbox" class="onoffswitch-check
<!--                        <label class="onoffswitch-label" for="targetExi
<!--                            <span class="onoffswitch-inner"></span>-->
<!--                            <span class="onoffswitch-switch"></span>-->
<!--                        </label>-->
<!--                    </div>-->
<!--                </div>-->
<!--            </div>-->
<!--        </div>-->
<!--        <div class="form-group">-->
<!--            <label class="col-sm-3 control-label is-required">是否全程监
<!--                : </label>-->
<!--            <div class="col-sm-8">-->
<!--                <div class="switch" style="display: flex">-->
<!--                    <div class="onoffswitch" style="float: left">-->
<!--                        <input type="hidden" name="fullMonitor" th:valu
<!--                        <input type="checkbox" class="onoffswitch-check
<!--                        <label class="onoffswitch-label" for="fullMonit
<!--                            <span class="onoffswitch-inner"></span>-->
<!--                            <span class="onoffswitch-switch"></span>-->
<!--                        </label>-->
<!--                    </div>-->
<!--                </div>-->
<!--            </div>-->
<!--        </div>-->
```

```
<!--        <div class="form-group">-->
<!--            <label class="col-sm-3 control-label">检测前置秒数-->
<!--                : </label>-->
<!--            <div class="col-sm-8">-->
<!--                <input name="monitorPrefixSecond" th:field="*{monitorPr
<!--                       type="text">-->
<!--            </div>-->
<!--        </div>-->
```

```
select  d.* from pc_monitoring_task t left join pc_monitoring_task_device d
where t.fill_flag = 1 and t.del_flag = 0 and d.monitoring_device_id in
<foreach item="id" collection="array" open="(" separator="," close=")">
    #{id}
</foreach>
group by t.id
```

安全监护人：guardian_person_duty 车后侧巡视人员：rear_car_patrol_person 鹤管排放动作：pipes_discharge_action 鹤管连接正确：connecte_pipes_correct 特殊鹤管连接正确：ts_connecte_pipes_correct 鹤管：pipes 悬梯：ladder 安全绳：safety_sling 人孔盖关闭：manhole_cover_close 人孔盖开启：manhole_cover_open 人孔盖开启（特殊）：ts_manhole_cover_open 人：people

车尾2-安全监护人,车尾2-车后侧巡视人员,车尾2-鹤管排放动作,车尾2-鹤管连接正确,车尾2-特殊鹤管连接正确,车尾2-鹤管,车尾2-悬梯,车尾2-安全绳,车尾2-人孔盖关闭,车尾2-人孔盖开启,车尾2-人

guardian_person_duty,rear_car_patrol_person,pipes_discharge_action,connecte_pipes_correct,ts_connecte_pipes_correct,pipes,ladder,safety_sling,manhole_cover_close,manhole_cover_open,people

安全监护人：guardian_person_duty 装卸前阀门检查动作：valve_inspection_action 车后侧巡视人员：rear_car_patrol_person 车头：truck_head 挡车牌：baffle 人：people

安全监护人,装卸前阀门检查动作,车后侧巡视人员,车头,挡车牌,人

guardian_person_duty,valve_inspection_action,rear_car_patrol_person,truck_head,baffle,people

需求

总流程

1、告警是实时告警

√1 车辆驶入检测 –车wei和车头两个摄像头判断，前后四个点位各一个摄像头 车头和车尾独立模型，车中按照点位相似度独立模型 √ 2车辆停放位置检测 —位置判断,ROI判断，ROI跟点位一一对应 √ 3车前挡车牌检测 — 车停止多长时间没放置挡车牌告警（时间可配） × 装卸前阀门检查动作检测 –（待确认-阀门检查动作最好附图，告警有没有条件） √ 4安全监护人员在岗检测 –时间段没有出现告警（时间可配） √ 5鹤管管连接状态检测 – 鹤管连接 √ 6安全带检测 –车顶人员和安全带做对比 √ 7车后侧装卸车过程人员安全巡视检测—时间段没有出现告警（时间可配） √ 8人孔盖关闭检测 – 人离开一个时间段未关闭，告警 √ 9鹤管排放动作检测 – 鹤管归为，检测到未排放动作 √ 10鹤管归位检测 – 在鹤管连接前提条件下检测，车走了，未归为，告警 √ 11悬梯归位检测 – 在鹤管连接前提条件下检测，车走了，未归为，告警 √ 12挡车牌归位检测 – 在车牌检测前提条件下检测告警，车走了，未放到指定位置，告警 √ 13车辆离开检测 – 车进入条件下检测

16、泄漏监测 —有泄露就告警

流程

1、车辆驶入检测 2、车辆停放位置检测 3、车前挡车牌检测 4、装卸前阀门检查动作检测 5、鹤管管连接检测 6、人孔盖关闭检测 7、鹤管排放动作检测 8、鹤管归位检测 9、悬梯归位检测 10、挡车牌归位检测 11、车辆离开检测

全程

1、安全监护人员在岗检测 –时间段没有出现告警（时间可配） 2、安全带检测 –车顶人员和安全带做对比 3、车后侧装卸车过程人员安全巡视检测—时间段没有出现告警（时间可配） 4、泄漏监测 —有泄露就告警

后续增加功能

1、总结 2、短视频

车辆停放位置检测 车前挡车牌检测 安全监护人员在岗检测 车后侧装卸车过程人员安全巡视检测 鹤管排放动作检测 鹤管管连接状态检测 鹤管归位检测 悬梯归位检测 安全带检测 人孔盖关闭检测 车辆驶入

46 truck_head 47 baffle 48 guardian_person_duty 49 rear_car_patrol_person 50 pipes_discharge_action 51 connecte_pipes_correct 52 pipes 53 ladder 54 safety_sling 56 manhole_cover_open 59 truck_head

车尾

安全监护人：guardian_person_duty 车后侧巡视人员：rear_car_patrol_person 鹤管排放动作：pipes_discharge_action 鹤管连接正确：connecte_pipes_correct 鹤管：pipes 悬梯：ladder 安全绳：safety_sling 人孔盖关闭：manhole_cover_close 人孔盖开启：manhole_cover_open 人孔盖开启（特殊）：ts_manhole_cover_open 人：people 车头：truck_head 挡车牌：baffle

车头,挡车牌,安全监护人,车后侧巡视人员,鹤管排放动作,鹤管连接正确,鹤管,悬梯,安全绳,人孔盖关闭,人孔盖开启,人孔盖开启（特殊）,人

truck_head,baffle,guardian_person_duty,rear_car_patrol_person,pipes_discharge_action,connecte_pipes_correct,pipes,ladder,safety_sling,manhole_cover_close,manhole_cover_open,ts_manhole_cover_open,people

```
return "/Users/jeffrey/Desktop/upload/ffmpeg/coordinate/1758FBE546B45E2F653
```

```
baffleRuleCheck_   //挡车牌规则判断用
guardianPersonDutyRuleCheck_    //安全监护人检测动作用
bafflePlaceAlready_     //挡车牌已经放置标识
connectPipesCorrect_ //鹤管连接标识
rearCarPatrolPersonRuleCheck_ //车后巡视人员检测规则用
manholeCoverOpenRuleCheck_ //人孔盖开启标识



safetySlingResultRuleChecking_ //安全绳算法检测判断条件
alarmImagePath_ //安全绳检测告警图片
noAlarmImagePath_ //安全绳检测正常图片
```

```
boolean isSpecialHandle = true;
int specialHandleCount = 0;
int specialHandleSeconds = 0;
if (modelIdsList.size() != 2){
    isSpecialHandle = false;
}else {
    List<String> isSpecialHandleRules = Arrays.asList("people", "safety_hoo
    for (PcAiModel pcAiModel : pcAiModels) {
        boolean contains = isSpecialHandleRules.contains(pcAiModel.getLabel
        if (!contains) {
            isSpecialHandle = false;
            break;
        }
    }
}

if (isSpecialHandle){
    PcAiModel pcAiModel = pcAiModels.stream().filter(o -> "safety_hook".equ
    Long id = pcAiModel.getId();
    PcMonitoringDeviceModel pcMonitoringDeviceModel = new PcMonitoringDevic
    pcMonitoringDeviceModel.setAiModelId(id);
    pcMonitoringDeviceModel.setMonitoringDeviceId(callback.getVideoId());
    List<PcMonitoringDeviceModel> pcMonitoringDeviceModels = pcMonitoringDe
    if (pcMonitoringDeviceModels.isEmpty()){
        isSpecialHandle = false;
    }else {
        PcMonitoringDeviceModel pcMonitoringDeviceModel1 = pcMonitoringDevi
        if (pcMonitoringDeviceModel1.getMonitoringStatus()  == 0){
            //开启了过滤检测
            specialHandleCount = Integer.parseInt(pcMonitoringDeviceModel1.
            specialHandleSeconds = Integer.parseInt(pcMonitoringDeviceModel
```

```
        }
    }
}
```

```
safety_hook_special_handle
```

```
{"taskId": 15, "videoId": "6", "pointId": 0, "imageName": "6_498_2023-11-28
```

/Users/jeffrey/Desktop/upload/SopFile/2023-11-28/1729439066870657024.jpg

```
redisUtil.del("PcPointStepHandle_mark" , callbackResult.getPointId() + "_"
```

```
{
            "category": "rear_car_patrol_person",
            "bbox": [
                400.0,
                400.0,
                1200.0,
                1200.0
            ],
            "score": 0.8769387006759644
        }
```

安全绳检测 按照时间进行配置

安全巡视人员 鹤管链接后15分总内出现一次即可

人孔盖开启之后 放到悬梯归位的时候判断

鹤管排放动作

```
String imagePath = callbackResult.getImagePath();
String s = scpDownFile(sftpConfig, imagePath);
```

基础算法

```
{
        "taskId": 15,
        "videoId": "6",
        "pointId": 0,
        "imageName": "6_498_2023-11-28-17-22-11",
        "node": "node_2",
        "imagePath": "/mnt/data/6/6_498_2023-11-28-17-22-11.jpg",
        "result": [
                {
                        "label": "chair",
                        "bbox": [
                                515,
                                723,
                                717,
                                1018
                        ],
                        "score": 0.83
                },
                {
```

```
                        "label": "chair",
                        "bbox": [
                                201,
                                839,
                                442,
                                1057
                        ],
                        "score": 0.701
                },
                {
                        "label": "person",
                        "bbox": [
                                541,
                                105,
                                633,
                                334
                        ],
                        "score": 0.697
                }
        ],
        "ip": "192.168.1.11",
        "datetime": "2023-11-28 17:22:11"
}
```

md5加密：e10adc3949ba59abbe56e057f20f883e MD5 + salt：b42619cf9e4a1f5dad2aaa8a7a70b31e MD5 + salt + hash：f57cf2d66c506567c49ad04bb58fa566

```
{
    "imageName": "12_20_12690_2023-11-20-11-25-41",
    "imagePath": "/usr/local/server/aicsp/uploadPath/ffmpeg/1.jpeg",
    "ip": "192.168.1.14",
    "node": "node_1",
    "params": {},
    "pointId": 1,
    "taskId": 14,
    "videoId": 1,
    "time": "@now('yyyy-MM-dd HH:mm:ss')",
    "result": [

       {
            "bbox": [
                1135.0,
                388.0,
                1287.0,
                553.0
            ],
            "label": "truck_head",
            "score": 0.9549359679222107
            // 车辆驶离/驶入
        }

        // {
        //     "bbox": [
        //         1390.0,
        //         476.0,
        //         1410.0,
        //         541.0
        //     ],
```

```
        //     "label": "baffle",
        //     "score": 0.3130616843700409
        // },
        //  {
        //     "bbox": [
        //         1500.0,
        //         722.0,
        //         2000.0,
        //         2100.0
        //     ],
        //     "label": "truck_head",
        //     "score": 0.8130616843700409
        //     // 车辆驶离
        // },

        // {
        //     "bbox": [
        //         1135.0,
        //         388.0,
        //         1287.0,
        //         553.0
        //     ],
        //     "label": "truck_head",
        //     "score": 0.9549359679222107
        //     // 车辆驶离/驶入
        // },

        // {
        //     "bbox": [
        //         689.0,
        //         500.0,
        //         1387.0,
        //         1387.0
        //     ],
        //     "label": "pipes_discharge_action",
        //     "score": 0.9549359679222107
        // },
        // {
        //     "bbox": [
        //         689.0,
        //         500.0,
        //         1387.0,
        //         1387.0
        //     ],
        //     "label": "guardian_person_duty",
        //     "score": 0.9549359679222107
        //     // 安全监护人在岗
        // },
        // {
        //     "bbox": [
        //         1000.0,
        //         388.0,
        //         1100.0,
        //         400.0
        //     ],
        //     "label": "baffle",
        //     "score": 0.9549359679222107
        // },

        // {
```

```
        //     "bbox": [
        //         540.0,
        //         318.0,
        //         1387.0,
        //         1387.0
        //     ],
        //     "label": "safety_sling",
        //     "score": 0.9549359679222107
        // },

        // {
        //     "bbox": [
        //         540.0,
        //         318.0,
        //         1387.0,
        //         1387.0
        //     ],
        //     "label": "people",
        //     "score": 0.9549359679222107
        // }
        // ,

        // {
        //     "bbox": [
        //         540.0,
        //         318.0,
        //         1387.0,
        //         1387.0
        //     ],
        //     "label": "people",
        //     "score": 0.9549359679222107
        // },
        // {
        //     "bbox": [
        //         540.0,
        //         318.0,
        //         1387.0,
        //         1387.0
        //     ],
        //     "label": "connecte_pipes_correct",
        //     "score": 0.9549359679222107
        //     // 鹤管链接
        // }
        // ,
        // {
        //     "bbox": [
        //         540.0,
        //         318.0,
        //         1387.0,
        //         1387.0
        //     ],
        //     "label": "rear_car_patrol_person",
        //     "score": 0.9549359679222107
        //     // 车后巡视人员
        // }
        // ,
        // {
        //     "bbox": [
        //         700.0,
        //         320.0,
```

```
        //         1387.0,
        //         1387.0
        //     ],
        //     "label": "pipes",
        //     "score": 0.9549359679222107
        //     // 鹤管归位
        // },

        // {
        //     "bbox": [
        //         1042.0,
        //         169.0,
        //         1787.0,
        //         1787.0
        //     ],
        //     "label": "ladder",
        //     "score": 0.9549359679222107
        //     // 悬梯归位
        // },

        // {
        //     "bbox": [
        //         720.0,
        //         40.0,
        //         1787.0,
        //         1787.0
        //     ],
        //     "label": "manhole_cover_open",
        //     "score": 0.9549359679222107
        //     // 悬梯归位
        // }
    ]
}
```

```
,
          {
            "bbox": [
                720.0,
                40.0,
                1787.0,
                1787.0
            ],
            "label": "manhole_cover_open",
            "score": 0.9549359679222107
        }



,
        {
            "bbox": [
                540.0,
                318.0,
                1387.0,
                1387.0
            ],
            "label": "pipes",
            "score": 0.9549359679222107
        }
```

```
[
    [
        {
            "oX":664,
            "oY":30
        },
        {
            "oX":672,
            "oY":1062
        },
        {
            "oX":1172,
            "oY":1062
        },
        {
            "oX":1116,
            "oY":40
        },
        {
            "oX":1116,
            "oY":40
        }
    ],
    [
        {
            "oX":1200,
            "oY":48
        },
        {
            "oX":1238,
            "oY":1060
        },
        {
            "oX":1708,
            "oY":1068
        },
        {
            "oX":1632,
            "oY":54
        },
        {
            "oX":1632,
            "oY":54
        }
    ]
]
```



```
[
    [
        {
            "oX":664,
            "oY":30
        },
        {
            "oX":672,
            "oY":1062
        },
        {
            "oX":1172,
            "oY":1062
```

```
        },
        {
            "oX":1116,
            "oY":40
        },
        {
            "oX":1116,
            "oY":40
        }
    ],
    [
        {
            "oX":1200,
            "oY":48
        },
        {
            "oX":1238,
            "oY":1060
        },
        {
            "oX":1708,
            "oY":1068
        },
        {
            "oX":1632,
            "oY":54
        },
        {
            "oX":1632,
            "oY":54
        }
    ],
    [
        {
            "oX":562,
            "oY":82
        },
        {
            "oX":576,
            "oY":1026
        },
        {
            "oX":1062,
            "oY":1018
        },
        {
            "oX":998,
            "oY":120
        },
        {
            "oX":998,
            "oY":120
        },
        {
            "oX":998,
            "oY":120
        }
    ],
    [
        {
```

```
            "oX":1364,
            "oY":132
        },
        {
            "oX":1398,
            "oY":1006
        },
        {
            "oX":1848,
            "oY":1016
        },
        {
            "oX":1816,
            "oY":104
        },
        {
            "oX":1816,
            "oY":104
        }
    ]
]
```

[[{“oX”:664,“oY”:30},{“oX”:672,“oY”:1062},{“oX”:1172,“oY”:1062},{“oX”:1116,“oY”:40},{“oX”:1116,“oY”:40}],[{“oX”:1200,“oY”:48},{“oX”:1238,“oY”:1060},{“oX”:1708,“oY”:1068},{“oX”:1632,“oY”:54},{“oX”:1632,“oY”:54}],[{“oX”:562,“oY”:82},{“oX”:576,“oY”:1026},{“oX”:1062,“oY”:1018},{“oX”:998,“oY”:120},{“oX”:998,“oY”:120},{“oX”:998,“oY”:120}],[{“oX”:1364,“oY”:132},{“oX”:1398,“oY”:1006},{“oX”:1848,“oY”:1016},{“oX”:1816,“oY”:104},{“oX”:1816,“oY”:104}]]

[[{“oX”:664,“oY”:30},{“oX”:672,“oY”:1062},{“oX”:1172,“oY”:1062},{“oX”:1116,“oY”:40},{“oX”:1116,“oY”:40}],[{“oX”:1200,“oY”:48},{“oX”:1238,“oY”:1060},{“oX”:1708,“oY”:1068},{“oX”:1632,“oY”:54},{“oX”:1632,“oY”:54}]]

```
server {
         listen 80;
                          server_name a.caaa.cn;

        include enable-php.conf;

        #charset koi8-r;

        location / {
             root /opt/homebrew/var/www;
            index  index.html index.htm index.php;
        }

        #error_page  404              /404.html;

        # redirect server error pages to the static page /50x.html
        #
        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   html;
        }

        #proxy the PHP scripts to Apache listening on 127.0.0.1:80

        location ~ \.php$ {
```

```
            proxy_pass   http://127.0.0.1;
        }

        #pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000

        location ~ \.php$ {
            root           html;
            fastcgi_pass   127.0.0.1:9000;
            fastcgi_index  index.php;
            fastcgi_param  SCRIPT_FILENAME  /scripts$fastcgi_script_name;
            include        fastcgi_params;
        }

        # deny access to .htaccess files, if Apache's document root
        # concurs with nginx's one
        #
        #location ~ /\.ht {
        #    deny  all;
        #}
    }




server
{
    listen 80;
    server_name a.caaa.cn;
    index index.php index.html index.htm default.php default.htm default.ht
    root "/opt/homebrew/var/www";

    #PHP-INFO-START
    include enable-php-83.conf;
    #PHP-INFO-END

    #REWRITE-START URL重写规则引用,修改后将导致面板设置的伪静态规则失效
    include /Users/jeffrey/Library/PhpWebStudy/server/vhost/rewrite/a.caaa.
    #REWRITE-END

    #禁止访问的文件或目录
    location ~ ^/(\.user.ini|\.htaccess|\.git|\.svn|\.project|LICENSE|READM
    {
        return 404;
    }

    #一键申请SSL证书验证目录相关设置
    location ~ \.well-known{
        allow all;
    }

    location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$
    {
        expires      30d;
        error_log off;
        access_log /dev/null;
    }

    location ~ .*\.(js|css)?$
    {
        expires      12h;
```

```
        error_log off;
        access_log /dev/null;
    }
    access_log  /Users/jeffrey/Library/PhpWebStudy/server/vhost/logs/a.caaa
    error_log  /Users/jeffrey/Library/PhpWebStudy/server/vhost/logs/a.caaa.
}
```

```
public void res(){
        String faceUrl = "http://127.0.0.1:10008/image_infer";
        Map<String, Object> params = new HashMap<>();
        params.put("imagePath", frameImgPath);
        params.put("faceDBId", 1);
        String response = HttpUtil.post(faceUrl, JSONObject.toJSONString(pa
        logger.info("*************************人脸识别结果===response:{}", r
        FaceModelResult faceModelResult = JSON.parseObject(response, FaceMo
        logger.info("*************************人脸识别结果===faceModelResult
        int code = Math.toIntExact(faceModelResult.getCode());

        if (code == 200) {
            logger.info("*************************人脸识别结果回调成功");
            //人脸识别成功并保存结果
            List<ModelResult> result = new ArrayList<>();
            try {
                result = faceModelResult.getResult();
            }catch (Exception e){
                logger.error("人脸识别结果获取异常：{}",  JSONObject.toJSONStr
            }

            logger.info("*************************人脸识别结果===result:{}",
            String face_distinguish_score = configService.selectConfigByKey
            if (!result.isEmpty()) {
                List<ModelResult> modelResults1 = new ArrayList<>();
                for (ModelResult modelResult : result) {
                    if (modelResult.getFaceId() == -1) {
                        continue;
                    }
                    ModelResult modelResult1 = new ModelResult();
                    modelResult1.setBbox(modelResult.getBbox());
                    modelResults1.add(modelResult1);
                    if (modelResult.getScore() > Double.parseDouble(face_di
                        PcFaceRecognitionLibrary faceRecognitionLibrary = p
                        remark.append(faceRecognitionLibrary.getUserName())
                    }
                }
                ArrayList<Location> locations = aiTaskService.getLocations(
                logger.info("*************************人脸识别结果===locatio
                faceImgPath = aiTaskService.transformImg(locations, frameIm
                logger.info("*************************人脸识别结果===faceImg
            }
        }

        remarkRes = remark.toString();
        logger.info("*************************人脸识别结果===remarkRes:{}",
        if ("".equals(remarkRes)) {
            remarkRes = "XXXX-XXXXXX";
        }
//                if ("".equals(faceImgPath)) {
        monitoringDeviceResult.setImgUrl(Constants.RESOURCE_PREFIX + faceIm
```

```
//                }
        monitoringDeviceResult.setRemark(remarkRes);
    }
```

```
@Data
public class FaceModelResult {
    private Long code;
    private String msg;
    public List<ModelResult> result;
}
```

```
@Data
public class ModelResult {

    /**
     * 模型标识
     */
    private String category;

    /**
     * 坐标
     */
    private List<Double> bbox;

    /**
     * 得分
     */
    private Double score;

    /**
     * 人脸ID
     */
    private Long faceId;

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }

    public List<Double> getBbox() {
        return bbox;
    }

    public void setBbox(List<Double> bbox) {
        this.bbox = bbox;
    }
```

```
    public double getScore() {
        return score;
    }

    public void setScore(double score) {
        this.score = score;
    }

}
```

空白文档

2023-08-21 更新告警周期

2023-08-21 更新告警周期

```
ALTER TABLE `pc_monitoring_task_rule`
ADD COLUMN `alarm_cycle` int(11) NULL DEFAULT 0 COMMENT '告警周期  单位秒' AF
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++
```

丰益氯碱新需求数据库设计

```
-- 添加复合检测字段

ALTER TABLE `pc_monitoring_task`
ADD COLUMN `complex_detection` tinyint(1) NULL DEFAULT 0 COMMENT '是否进行复

ALTER TABLE `pc_monitoring_task_rule_model`
ADD COLUMN `task_id` int(11) DEFAULT NULL COMMENT '任务ID'  AFTER `model_id

CREATE TABLE `pc_monitoring_task_rule` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `task_id` int(11) DEFAULT NULL COMMENT '任务ID',
  `rule` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMENT '检测
  `rule_type` tinyint(1) DEFAULT NULL COMMENT '规则类型 1：正常  2：告警',
  `compare_type` tinyint(1) DEFAULT '1' COMMENT '比较方式  1：数量比较  2：检测
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_ur

CREATE TABLE `pc_monitoring_task_rule_model` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `task_rule_id` int(11) DEFAULT NULL COMMENT '任务规则ID',
  `model_id` int(11) DEFAULT NULL COMMENT '规则关联模型',
  `task_id` int(11) DEFAULT NULL COMMENT '任务ID',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=41 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_u


-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('任务检测规则', '3000', '1', '/pc/pc_monitoring_task_rule', 'C', '1',

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();
```

```
-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('任务检测规则查询', @parentId, '1',  '#',  'F', '0', 'pc:pc_monitoring

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('任务检测规则新增', @parentId, '2',  '#',  'F', '0', 'pc:pc_monitoring

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('任务检测规则修改', @parentId, '3',  '#',  'F', '0', 'pc:pc_monitoring

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('任务检测规则删除', @parentId, '4',  '#',  'F', '0', 'pc:pc_monitoring

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('任务检测规则导出', @parentId, '5',  '#',  'F', '0', 'pc:pc_monitoring


-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('任务规则关联模型', '3000', '1', '/pc/pc_monitoring_task_rule_model',

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('任务规则关联模型查询', @parentId, '1',  '#',  'F', '0', 'pc:pc_monitor

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('任务规则关联模型新增', @parentId, '2',  '#',  'F', '0', 'pc:pc_monitor

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('任务规则关联模型修改', @parentId, '3',  '#',  'F', '0', 'pc:pc_monitor

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('任务规则关联模型删除', @parentId, '4',  '#',  'F', '0', 'pc:pc_monitor

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('任务规则关联模型导出', @parentId, '5',  '#',  'F', '0', 'pc:pc_monitor
```

SOP流程数据库

数据库SQL

```
-- 流程表
CREATE TABLE `pc_process`  (
  `id` int NOT NULL AUTO_INCREMENT COMMENT 'id',
  `name` varchar(255) NOT NULL COMMENT '流程名称',
  `time_interval` int NOT NULL COMMENT '检测频率  秒/次',
  `state` tinyint NULL COMMENT '状态   1：开启   非 1：关闭',
  `create_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
```

```
  `update_time` timestamp NULL COMMENT '更新时间',
  `del_flag` tinyint(1) NULL DEFAULT 0 COMMENT '删除标记',
  PRIMARY KEY (`id`)
) COMMENT = '流程表'

-- 流程详情步骤
CREATE TABLE `pc_process_detail` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `process_id` int(11) NOT NULL COMMENT '流程 ID',
  `name` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT '' COMMENT '流程步骤
  `model_id` int(11) NOT NULL COMMENT '模型ID',
  `step_num` int(11) NOT NULL COMMENT '第几步骤',
  `create_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  `update_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_
  `del_flag` tinyint(4) DEFAULT '0' COMMENT '删除标记',
  `must_do` tinyint(1) DEFAULT '2' COMMENT '是否必做：1：必做   2：非必做',
  `result_rule` tinyint(1) DEFAULT NULL COMMENT '检测结果规则：  1：有结果   0：
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=15 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_u

-- 流程步骤关联通道
CREATE TABLE `pc_process_detail_device` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `process_detail_id` int(11) NOT NULL COMMENT '流程详情步骤 ID',
  `device_id` int(11) NOT NULL COMMENT '设备 ID',
  `coordinate` text COLLATE utf8mb4_unicode_ci COMMENT 'ROI',
  `create_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  `update_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_
  `del_flag` tinyint(1) DEFAULT '0' COMMENT '删除标记',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_ur

-- 流程步骤检测结果
CREATE TABLE `pc_process_result`  (
  `id` int NOT NULL COMMENT 'ID',
  `process_id` int NOT NULL COMMENT '流程 ID',
  `deal_state` tinyint(1) NOT NULL DEFAULT 1 COMMENT '处理状态 1：无需处理   2
  `remark` varchar(255) NULL COMMENT '备注',
  `state` tinyint(1) NOT NULL DEFAULT 1 COMMENT '结果状态  1：正常   2：异常',
  `process_type` tinyint(1) NOT NULL DEFAULT 1 COMMENT '此次流程检测状态   1：
  `create_time` timestamp NULL DEFAULT NULL COMMENT '创建时间',
  `update_time` timestamp NULL COMMENT '更新时间',
  `del_flag` tinyint(1) NULL DEFAULT 0 COMMENT '删除标记',
  PRIMARY KEY (`id`)
) COMMENT = '流程步骤检测结果'

-- 步骤任务检测结果详情
CREATE TABLE `pc_process_result_detail`  (
  `id` int NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `process_result_id` int NOT NULL COMMENT '步骤任务检测结果ID',
  `state` tinyint(1) NOT NULL DEFAULT 1 COMMENT '状态：1：正常   2：异常',
  `img_path` varchar(255) NOT NULL COMMENT '检测结果图片地址',
  `step_num` int NOT NULL COMMENT '第几步骤',
  `step_name` varchar(50) NOT NULL COMMENT '步骤名称',
  `create_time` timestamp NULL COMMENT '创建时间',
  `update_time` timestamp NULL COMMENT '更新时间',
  `del_flag` tinyint(1) NULL DEFAULT 0 COMMENT '删除标记',
  PRIMARY KEY (`id`)
) COMMENT = '步骤任务检测结果详情'
```

```
-- 抓拍配置表
CREATE TABLE `pc_capture_setting`  (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `process_id` int NOT NULL COMMENT '流程 ID',
  `name` varchar(255) NOT NULL COMMENT '抓拍配置名称',
  `begin_step_id` int NOT NULL COMMENT '开始步骤 ID',
  `end_step_id` int NOT NULL COMMENT '结束步骤 ID',
  `state` tinyint(1) NOT NULL COMMENT '抓拍任务状态  1：开启  非 1：关闭',
  `capture_rule` integer NOT NULL COMMENT '抓拍时间间隔',
  `device_id` int NOT NULL COMMENT '设置抓拍设备',
  `create_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  `update_time` timestamp NULL COMMENT '更新时间',
  `del_flag` tinyint(1) NULL COMMENT '删除标记',
  PRIMARY KEY (`id`)
) COMMENT = '抓拍配置表'

-- 抓拍结果
CREATE TABLE `pc_capture_setting_result`  (
  `id` int NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `capture_setting_id` int NOT NULL COMMENT '抓拍配置 ID',
  `capture_setting_type` tinyint(1) NOT NULL DEFAULT 1 COMMENT '此次抓拍状态
  `create_time` timestamp NULL COMMENT '创建时间',
  `update_time` datetime NULL COMMENT '更新时间',
  PRIMARY KEY (`id`)
) COMMENT = '抓拍结果'

-- 抓拍结果详情
CREATE TABLE `pc_capture_setting_result_detail`  (
  `id` int NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `capture_setting_result_id` int NOT NULL COMMENT '步骤任务检测结果详情ID',
  `img_path` varchar(255) NOT NULL COMMENT '抓拍图片地址',
  `create_time` timestamp NULL COMMENT '创建时间',
  `update_time` timestamp NULL COMMENT '更新时间',
  PRIMARY KEY (`id`)
) COMMENT = '抓拍结果详情'

----------------------
-- 菜单sql
----------------------

-- 流程列表
INSERT INTO `sys_menu` (`menu_id`, `menu_name`, `parent_id`, `order_num`, `

SELECT @topPparentId := LAST_INSERT_ID();

-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程列表管理', @topPparentId, '1', '/pc/pc_process', 'C', '0', 'pc:pc

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程查询', @parentId, '1',  '#',  'F', '0', 'pc:pc_process:list','#'

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程新增', @parentId, '2',  '#',  'F', '0', 'pc:pc_process:add','#',

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
```

```
values('流程修改', @parentId, '3',  '#',  'F', '0', 'pc:pc_process:edit','#'

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程删除', @parentId, '4',  '#',  'F', '0', 'pc:pc_process:remove','

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程导出', @parentId, '5',  '#',  'F', '0', 'pc:pc_process:export','




-- 流程步骤管理
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程步骤管理',  @topPparentId, '1', '/pc/process_detail', 'C', '0', '

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程步骤管理查询', @parentId, '1',  '#',  'F', '0', 'pc:process_detai

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程步骤管理新增', @parentId, '2',  '#',  'F', '0', 'pc:process_detai

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程步骤管理修改', @parentId, '3',  '#',  'F', '0', 'pc:process_detai

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程步骤管理删除', @parentId, '4',  '#',  'F', '0', 'pc:process_detai

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程步骤管理导出', @parentId, '5',  '#',  'F', '0', 'pc:process_detai

-- 流程步骤检测结果

-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程步骤检测结果', @topPparentId, '1', '/pc/process_result', 'C', '0'

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程步骤检测结果查询', @parentId, '1',  '#',  'F', '0', 'pc:process_re

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程步骤检测结果新增', @parentId, '2',  '#',  'F', '0', 'pc:process_re

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程步骤检测结果修改', @parentId, '3',  '#',  'F', '0', 'pc:process_re

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程步骤检测结果删除', @parentId, '4',  '#',  'F', '0', 'pc:process_re

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('流程步骤检测结果导出', @parentId, '5',  '#',  'F', '0', 'pc:process_re
```

```
-- 抓拍配置
-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('抓拍配置管理', '3259', '1', '/pc/capture_setting', 'C', '0', 'pc:capt

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('抓拍配置管理查询', @parentId, '1',  '#',  'F', '0', 'pc:capture_setti

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('抓拍配置管理新增', @parentId, '2',  '#',  'F', '0', 'pc:capture_setti

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('抓拍配置管理修改', @parentId, '3',  '#',  'F', '0', 'pc:capture_setti

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('抓拍配置管理删除', @parentId, '4',  '#',  'F', '0', 'pc:capture_setti

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('抓拍配置管理导出', @parentId, '5',  '#',  'F', '0', 'pc:capture_setti

-- 抓拍记录
-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('抓拍记录', '3259', '1', '/pc/capture_setting_result', 'C', '0', 'pc:

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('抓拍记录查询', @parentId, '1',  '#',  'F', '0', 'pc:capture_setting_r

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('抓拍记录新增', @parentId, '2',  '#',  'F', '0', 'pc:capture_setting_r

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('抓拍记录修改', @parentId, '3',  '#',  'F', '0', 'pc:capture_setting_r

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('抓拍记录删除', @parentId, '4',  '#',  'F', '0', 'pc:capture_setting_r

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('抓拍记录导出', @parentId, '5',  '#',  'F', '0', 'pc:capture_setting_r


CREATE TABLE `pc_coordinate_staging` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `coordinate` text COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '坐标信息',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci COMMENT=
```

数据库设计

> 流程表 `process`

`id` ID

`name` 流程名称

`time_interval` 检测频率 秒/次

`state` 状态 1：开启 非 1：关闭

`create_time` 创建时间

`update_time` 更新时间

`del_flag` 删除标记

> 流程详情 `process_detail`

`id` ID

`process_id` 流程 ID `process` 表 ID

`name` 流程名称

`model_id` 模型 ID

`step_num` 步骤 第几步

`create_time` 创建时间

`update_time` 更新时间

`del_flag` 删除标记

> 步骤关联通道 `process_device`

`id` ID

`process_detail_id` 流程详情 ID

`device_id` 设备 ID

`coordinate` ROI

`create_time` 创建时间

`update_time` 更新时间

`del_flag` 删除标记

> 步骤任务检测结果 `process_result`

`id`

`process_id` 流程 ID `process` 表 ID

`deal_state` 1：无需处理 2：未处理 3：已处理

`remark` 备注

`state` 1：正常 2：异常

`process_type` 此次流程检测状态 1：正在进行中 2：完成

`create_time` 创建时间

`update_time` 更新时间

`del_flag` 删除标记

> 步骤任务检测结果详情 `process_result_detail`

`id`

`process_result_id` 步骤任务检测结果ID

`state` 状态：1：正常 2：异常

`img_path` 图片路径

`step_num` 第几步

`step_name` 步骤名称

`create_time` 创建时间

`update_time` 更新时间

> 抓拍配置表 `capture_setting`

`id` ID

`process_id` 流程 ID `process` 表 ID

`name` 抓拍任务名称

`begin_step` 开始步骤

`end_step` 结束步骤

`state` 抓拍任务状态 1：开启 非 1：关闭

`capture_rules` 抓拍规则（设置保存时间间隔）

`device_id` 抓拍摄像头

`del_flag` 删除标记

`create_time` 创建时间

`update_time` 更新时间

> 步骤任务检测结果详情 `capture_setting_result`

`id`

`capture_setting_id` 抓拍配置ID

`capture_setting_type` 此次抓拍状态 1：正在进行中 2：完成

`create_time` 创建时间

`update_time` 更新时间

> 步骤任务检测结果详情 `capture_setting_result_detail`

`id`

`capture_setting_result_id` 步骤任务检测结果详情ID

`img_path` 图片路径

`create_time` 创建时间

`update_time` 更新时间

```
package com.ruoyi.asynchronous;

import com.ruoyi.common.utils.DateUtils;
import com.ruoyi.common.utils.StringUtils;
import com.ruoyi.pc.domain.*;
import com.ruoyi.pc.domain.dto.InferResultDetailDTO;
import com.ruoyi.pc.domain.dto.PcProcessResultCallbackDTO;
import com.ruoyi.pc.domain.dto.ResultDetailDTO;
import com.ruoyi.pc.mapper.*;
import com.ruoyi.pc.service.IPcProcessDetailDeviceService;
import com.ruoyi.pc.util.FileUtil;
import com.ruoyi.pc.util.ImageConcatenationUtils;
import com.ruoyi.system.service.ISysConfigService;
import lombok.extern.log4j.Log4j2;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Service;

import java.io.File;
import java.sql.Time;
import java.util.*;
import java.util.stream.Collectors;

@Service
@Component
@Log4j2
public class PcProcessCallbackAsync {

    @Autowired
    private PcProcessMapper pcProcessMapper;
```

```
    @Autowired
    private PcProcessDetailMapper pcProcessDetailMapper;
    @Autowired
    private PcProcessDetailDeviceMapper pcProcessDetailDeviceMapper;
    @Autowired
    private PcProcessResultMapper pcProcessResultMapper;
    @Autowired
    private PcProcessResultDetailMapper pcProcessResultDetailMapper;

    @Autowired
    private ISysConfigService configService;

    @Autowired
    private IPcProcessDetailDeviceService pcProcessDetailDeviceService;

//    @Async
    public void callback(PcProcessResultCallbackDTO callback) {
        Long processId = callback.getTaskId();
        List<ResultDetailDTO> result = callback.getResult();
        //这次请求的所有图片
        List<String> resultImages = result.stream().map(ResultDetailDTO::ge

        String concatenationImgPath = "/Users/jeffrey/Desktop/ffmpeg/" + Sy
        boolean concatenation = ImageConcatenationUtils.concatenation(resul
        boolean alarm = true;
        if (concatenation) {
            //获取所有的流程步骤
            PcProcessDetail pcProcessDetail = new PcProcessDetail();
            pcProcessDetail.setProcessId(processId);

            //获取当前流程的所有步骤
            List<PcProcessDetail> pcProcessDetails = pcProcessDetailMapper.
            Map<Long, List<PcProcessDetail>> allSteps = pcProcessDetails.st

            //先查询有没有记录
            PcProcessResult pcProcessResult = new PcProcessResult();
            pcProcessResult.setProcessId(processId);
            pcProcessResult.setProcessType(1);
            PcProcessResult pcProcessResultInfo = pcProcessResultMapper.sel

            outer:
            for (int i = 0; i < pcProcessDetails.size(); i++) { //遍历步骤
                //当前步骤是否是必做项目
                boolean mustDo = pcProcessDetails.get(i).getMustDo() == 1;
                //当前步骤检测结果的规则
                boolean resultRule = pcProcessDetails.get(i).getResultRule(
                //查询当前步骤的通道信息
                PcProcessDetailDevice pcProcessDetailDevice = new PcProcess
                pcProcessDetailDevice.setProcessDetailId(pcProcessDetails.g
                List<PcProcessDetailDevice> pcProcessDetailDevices = pcProc
                log.info(pcProcessDetailDevices);

                //先看当前步骤有没有结果
                for (PcProcessDetailDevice processDetailDevice : pcProcessD
                    List<ResultDetailDTO> collect = result.stream().filter(
                    //判断有没有当前通道的结果返回，如果没有直接跳过，不告警【数据异常
                    if (collect.isEmpty()) {
                        alarm = false;
                        if (mustDo){
                            break outer;
```

```
                    }
                    continue ;
                }

                //判断没有结果的情况
                ResultDetailDTO resultDetailDTO = collect.get(0);
                //判断告警条件
                if (resultRule){
                    // 有结果
                    //有结果才算是正常
                    if (resultDetailDTO.getInferResult().isEmpty()){
                        alarm = false;
                        if (mustDo){
                            break outer;
                        }
                        continue;
                    }

                }else {
                    //没结果
                    //没有结果才算是正常
                    if (!resultDetailDTO.getInferResult().isEmpty()) {
                        alarm = false;
                        if (mustDo){
                            break outer;
                        }
                        continue;
                    }
                }

                PcProcessDetail pcProcessDetail2 = pcProcessDetails.get
                //有结果
                List<InferResultDetailDTO> inferResult = resultDetailDT
                List<InferResultDetailDTO> InferResultDetailDTOs = infe
                if (InferResultDetailDTOs.isEmpty()) {
                    alarm = false;
                    if (mustDo){
                        break outer;
                    }
                    continue;
                }

                //第一步
                if (pcProcessResultInfo == null && pcProcessDetails.get
                    //流程信息
                    PcProcess pcProcess = pcProcessMapper.selectPcProce
                    pcProcessResultInfo = new PcProcessResult();
                    pcProcessResultInfo.setProcessId(processId);
                    pcProcessResultInfo.setProcessType(1);
                    pcProcessResultInfo.setProcessName(pcProcess.getNam
                    pcProcessResultInfo.setDealState(1);
                    pcProcessResultInfo.setState(1);
                    pcProcessResultInfo.setCreateTime(DateUtils.getNowD
                    pcProcessResultInfo.setDelFlag(0);
                    pcProcessResultMapper.insertPcProcessResult(pcProce
                }

                //查询当钱检测到第几步骤
                PcProcessResultDetail pcProcessResultDetailParam = new
                pcProcessResultDetailParam.setProcessResultId(pcProcess
```

```
                    List<PcProcessResultDetail> pcProcessResultDetails = pc

                    //已经检查过的步骤
                    int size = pcProcessResultDetails.size();
                    //当前步骤已经检测过了
                    if (size > pcProcessDetails.get(i).getStepNum()){
                        break ;
                    }

                    //当前步骤检测到结果，查看之前步骤，存在没有告警的步骤，直接告警
                    //已经存在的步骤 ID
                    List<Long> alarmStepList = pcProcessResultDetails.strea
                    Long processStepId = pcProcessDetails.get(i).getId();


                    //查询出所有的通道信息
                    List<PcProcessDetailDevice> pcProcessDetailDevicesList
                    String deviceName = pcProcessDetailDevicesList.stream()
                    String deviceImei = pcProcessDetailDevicesList.stream()

                    List<Long> deviceIds = pcProcessDetailDevicesList.strea
                    String deviceIdStr = StringUtils.join(deviceIds, ",");

                    for (int j = 1; j <= pcProcessDetails.get(i).getStepNum
                        if (!alarmStepList.contains(j)){
                            //j 步骤需要告警
                            PcProcessResultDetail pcProcessResultDetail = n
                            pcProcessResultDetail.setProcessResultId(pcProc
                            pcProcessResultDetail.setState(2);
                            pcProcessResultDetail.setImgPath(concatenationI
                            pcProcessResultDetail.setStepNum((long) j);
                            pcProcessResultDetail.setStepName(allSteps.get(
                            pcProcessResultDetail.setCreateTime(DateUtils.g
                            pcProcessResultDetail.setUpdateTime(DateUtils.g
                            pcProcessResultDetail.setDelFlag(0);
                            pcProcessResultDetail.setModelId(Math.toIntExac
                            pcProcessResultDetail.setModelName(allSteps.get
                            pcProcessResultDetail.setDeviceId(deviceIdStr);
                            pcProcessResultDetail.setDeviceImei(deviceImei)
                            pcProcessResultDetail.setDeviceName(deviceName)
                            pcProcessResultDetail.setDealState(2);
                            pcProcessResultDetailMapper.insertPcProcessResu
                        }
                    }
                    //当前步骤大于已有的保存结果，证明该步骤未保存，进行结果保存
//                    PcProcessResultDetail pcProcessResultDetail = new PcP
//                    pcProcessResultDetail.setProcessResultId(pcProcessRes
//                    pcProcessResultDetail.setState(1);


                }
                //有结果的情况
            }
            if (!alarm){
                //删除保存的结果 + 拼接的图片
                resultImages.add(concatenationImgPath);
                for (String resultImage : resultImages) {
                    File file = new File(resultImage);
                    boolean delete = file.delete();
                }
```

```
            }

            log.info(pcProcessDetails);
            try {
                log.info("正在进行流程开关请求");
                Thread.sleep(1000 * 5);
                log.info("异步请求操作执行完成");
            } catch (Exception e) {
                log.info("123");
            }
        }
    }
}
```

```
for (PcProcessDetailDevice processDetailDevice : pcProcessDetailDevices) {
                    //result   =》   回调结果中的所有通道结果信息
                    //根据步骤的通道信息   获取回调结果中的通道结果信息
                    List<ResultDetailDTO> collect = result.stream().filter(
                            o -> o.getId().equals(processDetailDevice.getDe
                    ).collect(Collectors.toList());

                    //判断有没有当前通道的结果返回，如果没有直接跳过，不告警【数据异常
                    if (collect.isEmpty()) {
                        alarm = false;
                        if (mustDo){
                            break outer;
                        }
                        continue ;
                    }

                    //判断没有结果的情况
                    ResultDetailDTO resultDetailDTO = collect.get(0);
                    //判断告警条件
                    if (resultRule){
                        // 有结果
                        //有结果才算是正常
                        if (resultDetailDTO.getInferResult().isEmpty()){
                            alarm = false;
                            if (mustDo){
                                break outer;
                            }
                            continue;
                        }

                    }else {
                        //没结果
                        //没有结果才算是正常
                        if (!resultDetailDTO.getInferResult().isEmpty()) {
                            alarm = false;
                            if (mustDo){
                                break outer;
                            }
                            continue;
                        }
                    }

                    PcProcessDetail pcProcessDetail2 = pcProcessDetails.get
                    //有结果
```

```
                List<InferResultDetailDTO> inferResult = resultDetailDT
                List<InferResultDetailDTO> InferResultDetailDTOs = infe
                if (InferResultDetailDTOs.isEmpty()) {
                    alarm = false;
                    if (mustDo){
                        break outer;
                    }
                    continue;
                }

                //第一步
                if (pcProcessResultInfo == null && pcProcessDetails.get
                    //流程信息
                    PcProcess pcProcess = pcProcessMapper.selectPcProce
                    pcProcessResultInfo = new PcProcessResult();
                    pcProcessResultInfo.setProcessId(processId);
                    pcProcessResultInfo.setProcessType(1);
                    pcProcessResultInfo.setProcessName(pcProcess.getNam
                    pcProcessResultInfo.setDealState(1);
                    pcProcessResultInfo.setState(1);
                    pcProcessResultInfo.setCreateTime(DateUtils.getNowD
                    pcProcessResultInfo.setDelFlag(0);
                    pcProcessResultMapper.insertPcProcessResult(pcProce
                }

                //查询当钱检测到第几步骤
                PcProcessResultDetail pcProcessResultDetailParam = new
                pcProcessResultDetailParam.setProcessResultId(pcProcess
                List<PcProcessResultDetail> pcProcessResultDetails = pc

                //已经检查过的步骤
                int size = pcProcessResultDetails.size();
                //当前步骤已经检测过了
                if (size > pcProcessDetails.get(i).getStepNum()){
                    break ;
                }

                //当前步骤检测到结果，查看之前步骤，存在没有告警的步骤，直接告警
                //已经存在的步骤 ID
                List<Long> alarmStepList = pcProcessResultDetails.strea
                Long processStepId = pcProcessDetails.get(i).getId();


                //查询出所有的通道信息
                List<PcProcessDetailDevice> pcProcessDetailDevicesList
                String deviceName = pcProcessDetailDevicesList.stream()
                String deviceImei = pcProcessDetailDevicesList.stream()

                List<Long> deviceIds = pcProcessDetailDevicesList.strea
                String deviceIdStr = StringUtils.join(deviceIds, ",");

                for (int j = 1; j <= pcProcessDetails.get(i).getStepNum
                    if (!alarmStepList.contains(j)){
                        //j 步骤需要告警
                        PcProcessResultDetail pcProcessResultDetail = n
                        pcProcessResultDetail.setProcessResultId(pcProc
                        pcProcessResultDetail.setState(2);
                        pcProcessResultDetail.setImgPath(concatenationI
                        pcProcessResultDetail.setStepNum((long) j);
                        pcProcessResultDetail.setStepName(allSteps.get(
```

```
                            pcProcessResultDetail.setCreateTime(DateUtils.g
                            pcProcessResultDetail.setUpdateTime(DateUtils.g
                            pcProcessResultDetail.setDelFlag(0);
                            pcProcessResultDetail.setModelId(Math.toIntExac
                            pcProcessResultDetail.setModelName(allSteps.get
                            pcProcessResultDetail.setDeviceId(deviceIdStr);
                            pcProcessResultDetail.setDeviceImei(deviceImei)
                            pcProcessResultDetail.setDeviceName(deviceName)
                            pcProcessResultDetail.setDealState(2);
                            pcProcessResultDetailMapper.insertPcProcessResu
                        }
                    }
                    //当前步骤大于已有的保存结果，证明该步骤未保存，进行结果保存
//                    PcProcessResultDetail pcProcessResultDetail = new PcP
//                    pcProcessResultDetail.setProcessResultId(pcProcessRes
//                    pcProcessResultDetail.setState(1);


                }
```

空白文档

使用人脸识别离岗监测条件

使用人脸识别离岗监测条件

- 需要修改redis配置，将 `notify-keyspace-events` 的值修改为 `Ehx` 或者是 `Ex` **(区分大小写)**
- 重启 `sys_menu`

redis配置

```
# It is possible to select the events that Redis will notify among a set
# of classes. Every class is identified by a single character:
#
#  K     Keyspace events, published with __keyspace@<db>__ prefix.
#  E     Keyevent events, published with __keyevent@<db>__ prefix.
#  g     Generic commands (non-type specific) like DEL, EXPIRE, RENAME, .
#  $     String commands
#  l     List commands
#  s     Set commands
#  h     Hash commands
#  z     Sorted set commands
#  x     Expired events (events generated every time a key expires)
#  e     Evicted events (events generated when a key is evicted for maxme
#  A     Alias for g$lshzxe, so that the "AKE" string means all the event
#
#  The "notify-keyspace-events" takes as argument a string that is compos
#  of zero or multiple characters. The empty string means that notificati
#  are disabled.
#
#  Example: to enable list and generic events, from the point of view of
#           event name, use:
#
#  notify-keyspace-events Elg
#
#  Example 2: to get the stream of the expired keys subscribing to channe
#             name __keyevent@0__:expired use:
#
#  notify-keyspace-events Ex
#
#  By default all notifications are disabled because most users don't nee
#  this feature and the feature has some overhead. Note that if you don't
#  specify at least one of K or E, no events will be delivered.
notify-keyspace-events Ehx
```

数据库设计

```
CREATE TABLE `pc_face_recognition_library` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `library_id` int(11) NOT NULL COMMENT '人脸识别库ID',
  `user_name` varchar(50) NOT NULL COMMENT '用户名',
  `job_number` varchar(50) DEFAULT '' COMMENT '工号',
  `user_name_code` varchar(255) NOT NULL COMMENT '姓名拼音',
```

```
  `gender` tinyint(1) NOT NULL DEFAULT '0' COMMENT '性别  0：男 1：女 2：未知'
  `profile_path` varchar(255) NOT NULL COMMENT '文件路径',
  `create_date` varchar(50) NOT NULL COMMENT '创建时间',
  `del_flag` int(11) DEFAULT '0' COMMENT '删除标记',
  `create_id` bigint(20) NOT NULL COMMENT '用户Id',
  `dept_id` bigint(20) DEFAULT NULL COMMENT '部门Id',
  `report_result` tinyint(1) NOT NULL COMMENT '人脸信息库上报结果  -1: 失败 0:
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=152 DEFAULT CHARSET=utf8 COMMENT='人脸识别人员

######################################################################
### pc_face_recognition_library 人脸库新增工号字段
######################################################################
ALTER TABLE `pc_face_recognition_library`
MODIFY COLUMN `job_number` varchar(50) CHARACTER SET utf8 COLLATE utf8_gene

###################################
### pc_face_task 新增媒体播报信息
###################################
ALTER TABLE `pc_face_task`
ADD COLUMN `audio_type` int(2) NULL DEFAULT NULL COMMENT '模型媒体类型 0 MP3
ADD COLUMN `audio_url` varchar(255) CHARACTER SET utf8 COLLATE utf8_general
ADD COLUMN `audio_text` varchar(1000) CHARACTER SET utf8 COLLATE utf8_gener
MODIFY COLUMN `del_flag` int(2) NOT NULL DEFAULT 0 COMMENT '删除标记' AFTER


#################################################
### pc_face_task_volume_device
#################################################
CREATE TABLE `pc_face_task_volume_device` (
  `id` int(11) NOT NULL COMMENT 'ID',
  `task_id` bigint(20) NOT NULL COMMENT '人脸识别任务ID',
  `device_id` int(11) NOT NULL COMMENT '音柱设备ID',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='人脸识别任务关联音柱';


#################################################
## sys_dict_data 离岗检测通道播放
#################################################
INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict

#################################################
### sys_menu 新增音柱播报记录菜单
#################################################
INSERT INTO `sys_menu` (`menu_name`, `parent_id`, `order_num`, `url`, `targ


ALTER TABLE `pc_face_recognition_library`
ADD COLUMN `report_result` tinyint(1) NOT NULL COMMENT '人脸信息库上报结果  -1

#################################
########### 注意这条SQL  先查看该表现有数据的最大ID是多少，再进行添加，并且将下列所有的
#################################

INSERT INTO `sys_menu` (`menu_id`, `menu_name`, `parent_id`, `order_num`, `

-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人员库', '3149', '1', '/pc/library', 'C', '0', 'pc:library:view', '#
```

```
-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人员库查询', @parentId, '1',  '#',  'F', '0', 'pc:library:list',

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人员库新增', @parentId, '2',  '#',  'F', '0', 'pc:library:add',

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人员库修改', @parentId, '3',  '#',  'F', '0', 'pc:library:edit',

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人员库删除', @parentId, '4',  '#',  'F', '0', 'pc:library:remove',

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人员库导出', @parentId, '5',  '#',  'F', '0', 'pc:library:export',


CREATE TABLE `pc_device_has_library` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `device_id` int(11) NOT NULL COMMENT '设备ID',
  `library_id` int(11) NOT NULL COMMENT '库ID',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8 COMMENT='人脸识别分类库


CREATE TABLE `pc_face_classify_library` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `unique_code` varchar(255) NOT NULL COMMENT '唯一编码',
  `library_name` varchar(255) NOT NULL COMMENT '库名称',
  `description` varchar(255) NOT NULL COMMENT '描述',
  `del_flag` int(2) DEFAULT '0' COMMENT '删除标记',
  `create_id` int(11) DEFAULT NULL COMMENT '创建人ID',
  `create_date` varchar(20) DEFAULT NULL COMMENT '创建时间',
  `update_id` int(11) DEFAULT NULL COMMENT '更新人ID',
  `update_date` varchar(20) DEFAULT NULL COMMENT '更新时间',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8 COMMENT='人脸识别分类库


-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别分类库', '3149', '1', '/pc/face_classify_library', 'C', '0',

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别分类库查询', @parentId, '1',  '#',  'F', '0', 'pc:face_classif

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别分类库新增', @parentId, '2',  '#',  'F', '0', 'pc:face_classif

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别分类库修改', @parentId, '3',  '#',  'F', '0', 'pc:face_classif
```

```
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别分类库删除', @parentId, '4',  '#',  'F', '0', 'pc:face_classif

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别分类库导出', @parentId, '5',  '#',  'F', '0', 'pc:face_classif



CREATE TABLE `pc_face_task` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `dept_id` bigint(20) NOT NULL COMMENT '部门ID',
  `library_id` bigint(11) DEFAULT NULL COMMENT '人脸分类库ID',
  `name` varchar(200) NOT NULL COMMENT '监控任务名称',
  `state` tinyint(1) NOT NULL COMMENT '监控任务状态: 0启用中，1禁用中',
  `date_type` tinyint(1) NOT NULL COMMENT '类型: 0指定周期，1指定日期  人脸识别黑
  `date_week` varchar(20) NOT NULL COMMENT '周期: 2,3,4,5,6,7,1 表示周一到周日
  `start_time` varchar(20) NOT NULL COMMENT '开始时间',
  `end_time` varchar(20) NOT NULL COMMENT '结束时间',
  `alarm_duration` int(11) NOT NULL DEFAULT '0' COMMENT '报警时长  单位：秒',
  `user_id` bigint(20) NOT NULL COMMENT '创建人ID',
  `create_date` varchar(20) NOT NULL COMMENT '创建时间',
  `audio_type` int(2) DEFAULT NULL COMMENT '模型媒体类型 0 MP3 1 文字转语音',
  `audio_url` varchar(255) DEFAULT NULL COMMENT 'mp3地址',
  `audio_text` varchar(1000) DEFAULT NULL COMMENT '文字转语音',
  `job_id` varchar(100) DEFAULT NULL COMMENT '任务id',
  `del_flag` int(2) NOT NULL DEFAULT '0' COMMENT '删除标记',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=29 DEFAULT CHARSET=utf8 COMMENT='人脸识别任务'

-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别离岗任务', '3149', '1', '/pc/face_task', 'C', '0', 'pc:face_ta

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别离岗任务查询', @parentId, '1',  '#',  'F', '0', 'pc:face_task:

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别离岗任务新增', @parentId, '2',  '#',  'F', '0', 'pc:face_task:

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别离岗任务修改', @parentId, '3',  '#',  'F', '0', 'pc:face_task:

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别离岗任务删除', @parentId, '4',  '#',  'F', '0', 'pc:face_task:

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别离岗任务导出', @parentId, '5',  '#',  'F', '0', 'pc:face_task:

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别离岗任务导出', @parentId, '6',  '#',  'F', '0', 'pc:face_task:

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别离岗任务导出', @parentId, '7',  '#',  'F', '0', 'pc:face_task:


CREATE TABLE `pc_face_task_device` (
```

```
  `id` bigint(11) NOT NULL AUTO_INCREMENT,
  `device_id` bigint(11) NOT NULL COMMENT '通道设备ID',
  `type` tinyint(1) NOT NULL COMMENT '类型 1：入口通道  2：出口通道',
  `face_task_id` int(11) NOT NULL COMMENT '任务ID',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=36 DEFAULT CHARSET=utf8 COMMENT='人脸识别任务关

CREATE TABLE `pc_face_recognition_device` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT '主键ID',
  `del_flag` int(2) DEFAULT NULL COMMENT '删除标记',
  `create_id` int(11) DEFAULT NULL COMMENT '创建人ID',
  `create_date` varchar(20) DEFAULT NULL COMMENT '创建时间',
  `update_id` int(11) DEFAULT NULL COMMENT '更新人ID',
  `update_date` varchar(20) DEFAULT NULL COMMENT '更新时间',
  `dept_id` int(11) NOT NULL COMMENT '部门ID',
  `name` varchar(30) NOT NULL COMMENT '通道名称',
  `ip` varchar(20) NOT NULL COMMENT '通道流IP',
  `port` int(11) NOT NULL COMMENT '通道流端口号',
  `user_name` varchar(20) NOT NULL COMMENT '流通道用户名',
  `password` varchar(50) NOT NULL COMMENT '通道流密码',
  `unique_code` varchar(32) NOT NULL COMMENT '通道唯一编码',
  `imei` varchar(30) DEFAULT NULL COMMENT '通道号',
  `app_status` int(2) DEFAULT NULL COMMENT '通道应用状态 0 激活 1 禁用',
  `online_status` int(2) DEFAULT NULL COMMENT '通道在线状态 0 在线 1 离线',
  `address` varchar(255) DEFAULT NULL COMMENT '通道详细地址',
  PRIMARY KEY (`id`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8 ROW_FORMAT=COMPACT CO

-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别设备管理', '3149', '1', '/pc/face_recognition_device', 'C', '

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别设备管理查询', @parentId, '1',  '#',  'F', '0', 'pc:face_recog

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别设备管理新增', @parentId, '2',  '#',  'F', '0', 'pc:face_recog

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别设备管理修改', @parentId, '3',  '#',  'F', '0', 'pc:face_recog

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别设备管理删除', @parentId, '4',  '#',  'F', '0', 'pc:face_recog

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别设备管理导出', @parentId, '5',  '#',  'F', '0', 'pc:face_recog

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别设备管理导出', @parentId, '5',  '#',  'F', '0', 'pc:face_recog

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别设备管理导出', @parentId, '5',  '#',  'F', '0', 'pc:face_recog

CREATE TABLE `pc_face_recognition_result` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `del_flag` int(2) DEFAULT '0' COMMENT '删除标记',
```

```
  `create_id` int(11) DEFAULT NULL COMMENT '创建人ID',
  `create_date` varchar(20) DEFAULT NULL COMMENT '创建时间',
  `update_id` int(11) DEFAULT NULL COMMENT '更新人ID',
  `update_date` varchar(20) DEFAULT NULL COMMENT '更新时间',
  `dept_id` int(11) DEFAULT NULL COMMENT '部门ID',
  `device_id` int(11) DEFAULT NULL COMMENT '人脸识别设备ID',
  `score` double(5,2) NOT NULL COMMENT '置信度',
  `exit_image_path` varchar(255) NOT NULL COMMENT '结果图片地址',
  `exit_result` varchar(255) DEFAULT NULL COMMENT '出口结果',
  `exit_time` varchar(20) DEFAULT NULL COMMENT '出口检测时间',
  `entrance_image_path` varchar(255) DEFAULT NULL COMMENT '入口图片',
  `entrance_result` varchar(255) DEFAULT NULL COMMENT '入口结果',
  `entrance_time` varchar(20) DEFAULT NULL COMMENT '入口检测时间',
  `result_type` tinyint(1) DEFAULT NULL COMMENT '是否离岗  1 离岗 0 未离岗',
  `task_id` bigint(20) NOT NULL COMMENT '任务ID',
  `face_id` int(11) DEFAULT NULL COMMENT '人脸ID',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8 COMMENT='人脸识别结果表

-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别结果', '3149', '1', '/pc/face_recognition_result', 'C', '0',

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别结果查询', @parentId, '1',  '#',  'F', '0', 'pc:face_recognit

insert into sys_menu (menu_name, parent_id, order_num, url, menu_taptype, v
values('人脸识别结果新增', @parentId, '2',  '#',  'F', '0', 'pc:face_recognit

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别结果修改', @parentId, '3',  '#',  'F', '0', 'pc:face_recognit

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别结果删除', @parentId, '4',  '#',  'F', '0', 'pc:face_recognit

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('人脸识别结果导出', @parentId, '5',  '#',  'F', '0', 'pc:face_recognit

################################
########### 字典属性
################################

#是否离岗
INSERT INTO `sys_dict_type` (`dict_name`, `dict_type`, `status`, `create_by

INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict

#人脸信息上报回调
INSERT INTO `sys_dict_type` (`dict_name`, `dict_type`, `status`, `create_by

INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
INSERT INTO `sys_dict_data` ( `dict_sort`, `dict_label`, `dict_value`, `dic
INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
```

```
##########################################################################
###  pc_face_recognition_device
##########################################################################
ALTER TABLE `pc_face_recognition_device`
ADD COLUMN `coordinate` text CHARACTER SET utf8mb3 COLLATE utf8mb3_general_

ALTER TABLE `pc_face_task`
ADD COLUMN `threshold` double NOT NULL DEFAULT 0 COMMENT '离岗检测阈值' AFTER

ALTER TABLE `pc_face_recognition_result`
ADD COLUMN `entrance_score` double(5, 2) NULL COMMENT '入口置信度' AFTER `sc
```

初始化基础环境

```
#启用模型
python /AiBox/face_function/face_app.py  --port 10007  --works 2
```

```
docker run -idt --name  face2 --net=host -v /usr/local/server/aicsp/uploadP

#aicsp
docker run -idt --name aicsp -p 8091:8091 -v /usr/local/server/aicsp/:/usr/

#redis
docker run -itd --net=host  --name redis -p 6379:6379 -v /usr/local/docker/
#mysql
docker run -idt --net=host --name mysql -p 3306:3306 -e MYSQL_ROOT_PASSWORD

docker run --net=host  --name mysql -p 3306:3306  -v /usr/local/docker/face
```

```
    public boolean checkTaskIsRunning(PcFaceTask pcFaceTask, String resultD
        try {
            String[] baseWeek = pcFaceTask.getDateWeek().split(",");
            Calendar calendar = Calendar.getInstance();
            String[] dateToWeek = {String.valueOf(calendar.get(Calendar.DAY
            String todayDate = DayUtils.getTodayDate();
            if (!DayUtils.isWeekOver(baseWeek, dateToWeek)) {
                return false;
            }

            //当前时间减去15分钟
            String startDateStr = todayDate + " " + pcFaceTask.getStartTime
            String endDateStr = todayDate + " " + pcFaceTask.getEndTime();
            SimpleDateFormat sdf= new SimpleDateFormat("yyyy-MM-dd HH:mm:ss
            Date startDate =sdf.parse(startDateStr);
            Date endDate =sdf.parse(endDateStr);
            endDate.setTime(endDate.getTime() + pcFaceTask.getAlarmDuration
            String newEndDateStr = new SimpleDateFormat("yyyy-MM-dd HH:mm:s
            //从当前日期减去15分钟
            calendar.add(Calendar.MINUTE, -15);
            return DayUtils.isOverlapping(resultDate, startDateStr, newEndD
        }catch (Exception e){

        }
    }
```

ALTER TABLE `pc_face_recognition_library` MODIFY COLUMN `iframe_url` text COMMENT ‘iframe_url’, AFTER `user_name` ;

空白文档

版本管理

```
INSERT INTO `sys_dict_type` (`dict_name`, `dict_type`, `status`, `create_by

INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict

ALTER TABLE `pc_ai_model`
ADD COLUMN `model_type` varchar(50) NULL COMMENT '模型类型' AFTER `model_pat


update pc_ai_model set model_type= "detection";
```

LiveCMS-linux-3.3.3-23031716/ -rw-rw-r– 1 ubuntu ubuntu 16786626 Mar 17 17:04
LiveCMS-linux-3.3.3-23031716.tar.gz drwxr-xr-x 5 root root 4096 Mar 17 17:09
LiveSMS-linux-3.3.3-23031716/ -rw-rw-r– 1 ubuntu ubuntu 26483056 Mar 17 17:04
LiveSMS-linux-3.3.3-23031716.tar.gz

空白文档

顶层平台

顶层平台

```
INSERT INTO `sys_menu` (`menu_name`, `parent_id`, `order_num`, `url`, `targ

-- 按钮父菜单ID
SELECT @topParentId := LAST_INSERT_ID();

-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('子平台管理', @topParentId, '1', '/pc/platform', 'C', '0', 'system:pla

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('子平台管理查询', @parentId, '1',  '#',  'F', '0', 'pc:platform:list',

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('子平台管理新增', @parentId, '2',  '#',  'F', '0', 'pc:platform:add',

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('子平台管理修改', @parentId, '3',  '#',  'F', '0', 'pc:platform:edit',

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('子平台管理删除', @parentId, '4',  '#',  'F', '0', 'pc:platform:remove

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('子平台管理导出', @parentId, '5',  '#',  'F', '0', 'pc:platform:export


-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('监控通道管理', '2026', '1', '/pc/monitoring_device', 'C', '0', 'pc:mo

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('监控通道管理查询', @parentId, '1',  '#',  'F', '0', 'pc:monitoring_de

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('监控通道管理新增', @parentId, '2',  '#',  'F', '0', 'pc:monitoring_de

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('监控通道管理修改', @parentId, '3',  '#',  'F', '0', 'pc:monitoring_de

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('监控通道管理删除', @parentId, '4',  '#',  'F', '0', 'pc:monitoring_de

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('监控通道管理导出', @parentId, '5',  '#',  'F', '0', 'pc:monitoring_de
-- 菜单 SQL
```

```
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('配置模型列表', '2026', '1', '/pc/monitoring_device_model', 'C', '0',

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('配置模型列表查询', @parentId, '1',  '#',  'F', '0', 'pc:monitoring_de

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('配置模型列表新增', @parentId, '2',  '#',  'F', '0', 'pc:monitoring_de

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('配置模型列表修改', @parentId, '3',  '#',  'F', '0', 'pc:monitoring_de

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('配置模型列表删除', @parentId, '4',  '#',  'F', '0', 'pc:monitoring_de

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('配置模型列表导出', @parentId, '5',  '#',  'F', '0', 'pc:monitoring_de
```

> 添加心跳任务

- 任务名称：子平台心跳检测
- 任务组：默认
- 调用目标字符串：heartbeatCheckTask.check(10) # 参数为将新心跳停留在 (??) 分钟之前的子平台设置为心跳异常
- cron表达式： 0 */1 * * * ? #每一分钟执行一次

子平台

```
-- 平台类型
INSERT INTO `sys_dict_type` (`dict_name`, `dict_type`, `status`, `create_by
INSERT INTO `sys_dict_data` ( `dict_sort`, `dict_label`, `dict_value`, `dic
INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict

-- 子平台是否上报
INSERT INTO `sys_dict_type` (`dict_id`, `dict_name`, `dict_type`, `status`,
INSERT INTO `sys_dict_data` ( `dict_sort`, `dict_label`, `dict_value`, `dic
INSERT INTO `sys_dict_data` ( `dict_sort`, `dict_label`, `dict_value`, `dic


-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('数据上报配置', '1', '1', '/pc/data_report_config', 'C', '0', 'pc:data

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('数据上报配置查询', @parentId, '1',  '#',  'F', '0', 'pc:data_report_c

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('数据上报配置新增', @parentId, '2',  '#',  'F', '0', 'pc:data_report_c

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('数据上报配置修改', @parentId, '3',  '#',  'F', '0', 'pc:data_report_c

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('数据上报配置删除', @parentId, '4',  '#',  'F', '0', 'pc:data_report_c

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('数据上报配置导出', @parentId, '5',  '#',  'F', '0', 'pc:data_report_c



CREATE TABLE `pc_data_report_config` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `platform_name` varchar(50) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '
  `type` tinyint(4) NOT NULL COMMENT '平台类型',
  `parent_platform_ip` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL COM
  `data_report` tinyint(4) NOT NULL COMMENT '是否推送  1：推送  非1 ：不推送',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci COMMENT=




INSERT INTO `sys_dict_type` (`dict_name`, `dict_type`, `status`, `create_by

INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
```

空白文档

使用坐标点进行对比检测

> 数据库变更 `pc_monitoring_device_model` 表添加字字段
>
> - `save_video_detail` 是否保留前后视频详情
> - `save_video_time` 保留前后视频时间

```
change_video_base
```

```
ALTER TABLE `pc_monitoring_device_model`
ADD COLUMN `save_video_detail` tinyint NULL DEFAULT 0 COMMENT '是否保存前后视
ADD COLUMN `save_video_time` int NULL COMMENT '保留前后视频时间 单位秒' AFTER


UPDATE `pc_monitoring_device_model` SET `save_video_detail` = 0 WHERE save_


-- ALTER TABLE `aicsp`.`pc_monitoring_device_model`
-- MODIFY COLUMN `save_video_detail` tinyint NULL DEFAULT 0 COMMENT '是否保存

ALTER TABLE `pc_monitoring_device_result`
ADD COLUMN `video_status` tinyint NULL DEFAULT 0 COMMENT '保存视频状态 0:没有

ALTER TABLE `pc_monitoring_device_result`
ADD COLUMN `video_path` varchar(255) NULL COMMENT '视频地址' AFTER `video_st

INSERT INTO `sys_config` (`config_id`, `config_name`, `config_key`, `config
```

使用坐标点进行对比检测

```
INSERT INTO `sys_config` (`config_name`, `config_key`, `config_value`, `con
```

```
// 6,345,309,574
//[
// [6,345] 左下
// [309,345]  右下
// [309,574] 右上
// [6,574] 左上
// ]
// 118,278,354,507
//[
// [118,278]  左上
// [354,278] 右上
// [354,507] 右下
// [118,507] 左下
// ]


[[{"oX":729,"oY":150},{"oX":30,"oY":741},{"oX":252,"oY":1065},{"oX":525,"oY
[{"oX":600,"oY":567},{"oX":1092,"oY":630}]]
```

```
[[{"oX":729,"oY":150},{"oX":30,"oY":741},{"oX":252,"oY":1065},{"oX":525,"oY
[{"oX":600,"oY":567},{"oX":1092,"oY":630}],
[{"oX":714,"oY":741}],
[{"oX":675,"oY":825},{"oX":876,"oY":639}]]


[[{"oX":729,"oY":150},{"oX":30,"oY":741},{"oX":252,"oY":1065},{"oX":525,"oY
,[{"oX":12,"oY":1047}]]


675,825
876,


1000,567,1202,766
[
[1000,567],
[1002,567],
[1002,766],
[800,766]
]
```

ALTER TABLE `pc_ai_model` ADD COLUMN `syn_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP COMMENT '同步时间';

大全2.0.3.1升级流程

```
INSERT INTO `sys_config` (`config_name`, `config_key`, `config_value`, `cor

CREATE TABLE `pc_activation` (
  `system_code` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '
  `activation_code` text CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci C
  `activation_status` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL
  PRIMARY KEY (`system_code`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

INSERT INTO `sys_menu` (`menu_name`, `parent_id`, `order_num`, `url`, `targ




INSERT INTO `sys_config` ( `config_name`, `config_key`, `config_value`, `co
VALUES
        ( 'MAC地址参数', 'mac_address_type_name', 'enp3s0f0', 'Y', 'admin',
```

空白文档

宿迁高铁站

宿迁高铁站

OcrSql

```
-- led设备管理
CREATE TABLE `pc_led_system` (
  `id` int NOT NULL AUTO_INCREMENT COMMENT 'id',
  `dept_id` int NOT NULL COMMENT '部门ID',
  `ip` varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NUL
  `port` varchar(5) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NU
  `model` varchar(50) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT
  `name` varchar(30) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci NOT N
  `status` int DEFAULT '0' COMMENT '设备应用状态 0 激活 1 禁用',
  `del_flag` int DEFAULT '0' COMMENT '删除标记',
  `create_id` int DEFAULT NULL COMMENT '创建人ID',
  `create_date` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_c
  `update_id` int DEFAULT NULL COMMENT '更新人ID',
  `update_date` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_c
  `content` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMENT 'L
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_ur

-- 通道关联led设备
CREATE TABLE `pc_led_system_device` (
  `id` int NOT NULL AUTO_INCREMENT,
  `monitoring_device_id` int DEFAULT NULL COMMENT '监控通道Id',
  `led_system_id` int DEFAULT NULL COMMENT 'led设备Id',
  PRIMARY KEY (`id`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COMMENT='通道关联led设备';

-- LED设备管理播控记录表 sql
CREATE TABLE `pc_led_play_record` (
  `id` int NOT NULL AUTO_INCREMENT,
  `create_date` varchar(50) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_c
  `del_flag` int DEFAULT NULL COMMENT '删除标记',
  `dept_id` bigint DEFAULT NULL COMMENT '部门Id',
  `monitoring_device_id` int NOT NULL COMMENT '通道ID',
  `led_system_id` int NOT NULL COMMENT 'led设备ID',
  `content` text CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci COMMENT '
  PRIMARY KEY (`id`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COMMENT='LED设备管理播控记录';

-- ocr通道和音柱设备关联表
CREATE TABLE `pc_ocr_device_volume` (
  `id` int NOT NULL AUTO_INCREMENT,
  `ocr_device_id` int DEFAULT NULL COMMENT '监控通道Id',
  `device_id` int DEFAULT NULL COMMENT '音柱设备Id',
  PRIMARY KEY (`id`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=13 DEFAULT CHARSET=utf8mb3 COMMENT='ocr通道和

-- ocr监控通道表
CREATE TABLE `pc_ocr_device` (
```

```
  `id` int NOT NULL AUTO_INCREMENT COMMENT '主键ID',
  `del_flag` int DEFAULT '0' COMMENT '删除标记',
  `create_id` int DEFAULT NULL COMMENT '创建人ID',
  `create_date` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_c
  `update_id` int DEFAULT NULL COMMENT '更新人ID',
  `update_date` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_c
  `dept_id` int DEFAULT NULL COMMENT '部门ID',
  `name` varchar(30) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci DEFAU
  `imei` varchar(50) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci DEFAU
  `app_status` int DEFAULT '0' COMMENT '通道应用状态 1 激活 0 禁用',
  `online_status` int DEFAULT '1' COMMENT '通道在线状态 0 在线 1 离线',
  `streaming_address` varchar(255) CHARACTER SET utf8mb3 COLLATE utf8mb3_ge
  `address` varchar(255) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci D
  `background` varchar(255) DEFAULT NULL COMMENT '背景图',
  `coordinate` varchar(255) DEFAULT NULL COMMENT 'ROI坐标',
  PRIMARY KEY (`id`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb3 COMMENT='ocr监控通道

-- OCR监控任务
CREATE TABLE `pc_ocr_lpr_task` (
  `id` int NOT NULL AUTO_INCREMENT COMMENT '播控任务ID',
  `create_date` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_c
  `del_flag` int DEFAULT '0' COMMENT '删除标记',
  `user_id` bigint DEFAULT NULL COMMENT '创建人ID',
  `dept_id` bigint DEFAULT NULL COMMENT '部门ID',
  `name` varchar(200) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci DEFA
  `audio_type` tinyint(1) DEFAULT '1' COMMENT '音频方式  1：文字转语音 2：音频文
  `audio_url` varchar(200) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci
  `audio_text` varchar(50) DEFAULT NULL COMMENT '文字转语音的文字',
  `status` int DEFAULT NULL COMMENT '监控任务状态: 0禁用，1启用',
  `recognition_mode` int DEFAULT NULL COMMENT '识别模式：0：抽帧识别，1：视频分析
  `date_type` int DEFAULT NULL COMMENT '播放日期类型: 0指定周期，1指定日期',
  `date_week` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci
  `date_content` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_
  `play_form` int DEFAULT NULL COMMENT '播放形式：0 一次播放 ，1 循环播放',
  `start_time` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci
  `end_time` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci D
  `time_interval` int DEFAULT NULL COMMENT '循环播放时间间隔',
  `job_id` varchar(64) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci DEF
  `time_threshold` int DEFAULT NULL COMMENT '周期',
  PRIMARY KEY (`id`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb3 COMMENT='监控任务表

-- ocr监控任务和监控通道关联表
CREATE TABLE `pc_ocr_lpr_task_device` (
  `id` int NOT NULL AUTO_INCREMENT,
  `task_id` int DEFAULT NULL COMMENT '监控任务Id',
  `ocr_device_id` int DEFAULT NULL COMMENT 'ocr通道Id',
  PRIMARY KEY (`id`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=17 DEFAULT CHARSET=utf8mb3 COMMENT='ocr监控任

-- 监控设备监测结果表
CREATE TABLE `pc_ocr_lpr_result` (
  `id` int NOT NULL AUTO_INCREMENT COMMENT '主键ID',
  `del_flag` int DEFAULT '0' COMMENT '删除标记',
  `create_id` int DEFAULT NULL COMMENT '创建人ID',
  `create_date` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_c
  `update_id` int DEFAULT NULL COMMENT '更新人ID',
  `update_date` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_c
  `dept_id` int DEFAULT NULL COMMENT '部门ID',
```

```
  `ocr_device_id` int DEFAULT NULL COMMENT '监控设备ID',
  `score` double DEFAULT NULL COMMENT '相似度',
  `content` varchar(255) DEFAULT NULL COMMENT '车牌检测',
  `img_url` varchar(255) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_ci D
  `result` text CHARACTER SET utf32 COLLATE utf32_general_ci COMMENT '测算结
  `result_date` varchar(20) DEFAULT NULL COMMENT '结果时间',
  PRIMARY KEY (`id`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=32 DEFAULT CHARSET=utf8mb3 COMMENT='监控设备监

-- ----------------------------------------------------------------------------
-- open_or_close_statue 字典
INSERT INTO `sys_dict_type` (`dict_name`, `dict_type`, `status`, `create_by

INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict


-- 添加 LED设备控制管理 菜单

INSERT INTO `sys_menu` (`menu_name`, `parent_id`, `order_num`, `url`, `targ
-- 按钮父菜单ID
SELECT @topParentId := LAST_INSERT_ID();

-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('led设备管理',  @topParentId, '1', '/pc/led_system', 'C', '0', 'pc:le

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('led设备管理查询', @parentId, '1',  '#',  'F', '0', 'pc:led_system:lis

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('led设备管理新增', @parentId, '2',  '#',  'F', '0', 'pc:led_system:add

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('led设备管理修改', @parentId, '3',  '#',  'F', '0', 'pc:led_system:edi

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('led设备管理删除', @parentId, '4',  '#',  'F', '0', 'pc:led_system:rem

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('led设备管理导出', @parentId, '5',  '#',  'F', '0', 'pc:led_system:exp

-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('通道关联led设备', @topParentId, '1', '/pc/led_system_device', 'C', '0

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('通道关联led设备查询', @parentId, '1',  '#',  'F', '0', 'pc:led_system

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('通道关联led设备新增', @parentId, '2',  '#',  'F', '0', 'pc:led_system
```

```
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('通道关联led设备修改', @parentId, '3',  '#',  'F', '0', 'pc:led_system

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('通道关联led设备删除', @parentId, '4',  '#',  'F', '0', 'pc:led_system

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('通道关联led设备导出', @parentId, '5',  '#',  'F', '0', 'pc:led_system

-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('LED设备管理播控记录', @topParentId, '1', '/pc/led_system_play_record'

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('LED设备管理播控记录查询', @parentId, '1',  '#',  'F', '0', 'pc:led_sys

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('LED设备管理播控记录新增', @parentId, '2',  '#',  'F', '0', 'pc:led_sys

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('LED设备管理播控记录修改', @parentId, '3',  '#',  'F', '0', 'pc:led_sys

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('LED设备管理播控记录删除', @parentId, '4',  '#',  'F', '0', 'pc:led_sys

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('LED设备管理播控记录导出', @parentId, '5',  '#',  'F', '0', 'pc:led_sys

-- 车牌识别
INSERT INTO `sys_menu` (`menu_name`, `parent_id`, `order_num`, `url`, `targ

-- 按钮父菜单ID
SELECT @topParentIdForOcr := LAST_INSERT_ID();
-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr监控通道', @topParentIdForOcr, '1', '/pc/ocr_device', 'C', '0', '

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr监控通道查询', @parentId, '1',  '#',  'F', '0', 'pc:ocr_device:lis

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr监控通道新增', @parentId, '2',  '#',  'F', '0', 'pc:ocr_device:add

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr监控通道修改', @parentId, '3',  '#',  'F', '0', 'pc:ocr_device:edi

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr监控通道删除', @parentId, '4',  '#',  'F', '0', 'pc:ocr_device:rem

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr监控通道导出', @parentId, '5',  '#',  'F', '0', 'pc:ocr_device:exp
```

```
-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr车牌监控任务', @topParentIdForOcr, '1', '/pc/ocr_lpr_task', 'C', '

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr车牌监控任务查询', @parentId, '1',  '#',  'F', '0', 'pc:ocr_lpr_ta

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr车牌监控任务新增', @parentId, '2',  '#',  'F', '0', 'pc:ocr_lpr_ta

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr车牌监控任务修改', @parentId, '3',  '#',  'F', '0', 'pc:ocr_lpr_ta

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr车牌监控任务删除', @parentId, '4',  '#',  'F', '0', 'pc:ocr_lpr_ta

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr车牌监控任务导出', @parentId, '5',  '#',  'F', '0', 'pc:ocr_lpr_ta


-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr车牌监控结果', @topParentIdForOcr, '1', '/pc/ocr_lpr_result', 'C',

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr车牌监控结果查询', @parentId, '1',  '#',  'F', '0', 'pc:ocr_lpr_re

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr车牌监控结果新增', @parentId, '2',  '#',  'F', '0', 'pc:ocr_lpr_re

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr车牌监控结果修改', @parentId, '3',  '#',  'F', '0', 'pc:ocr_lpr_re

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr车牌监控结果删除', @parentId, '4',  '#',  'F', '0', 'pc:ocr_lpr_re

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr车牌监控结果导出', @parentId, '5',  '#',  'F', '0', 'pc:ocr_lpr_re

-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr通道和音柱设备关联', @topParentIdForOcr, '1', '/pc/ocr_device_volum

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr通道和音柱设备关联查询', @parentId, '1',  '#',  'F', '0', 'pc:ocr_d

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
```

```
values('ocr通道和音柱设备关联新增', @parentId, '2',  '#',  'F', '0', 'pc:ocr_d

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr通道和音柱设备关联修改', @parentId, '3',  '#',  'F', '0', 'pc:ocr_d

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr通道和音柱设备关联删除', @parentId, '4',  '#',  'F', '0', 'pc:ocr_d

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('ocr通道和音柱设备关联导出', @parentId, '5',  '#',  'F', '0', 'pc:ocr_d

INSERT INTO `sys_dict_type` (`dict_name`, `dict_type`, `status`, `create_by

INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
```

添加摄像头类型

```
ALTER TABLE `pc_monitoring_device`
ADD COLUMN `device_type` varchar(50) CHARACTER SET utf8 COLLATE utf8_genera

ALTER TABLE `pc_ocr_device`
ADD COLUMN `device_type` varchar(50) CHARACTER SET utf8 COLLATE utf8_genera

INSERT INTO `sys_dict_type` (`dict_name`, `dict_type`, `status`, `create_by
INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
INSERT INTO `sys_dict_data` ( `dict_sort`, `dict_label`, `dict_value`, `dic
INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
```

空白文档

激活码激活

> 新添加激活码激活平台功能
>
> 设计数据变更：
>
> - 添加 `pc_activation` 激活表
> - 添加 `sys_menu` 菜单信息

```
INSERT INTO sys_config` (`config_name`, `config_key`, `config_value`, `conf


CREATE TABLE `pc_activation` (
  `system_code` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '
  `activation_code` text CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci C
  `activation_status` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL
  PRIMARY KEY (`system_code`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

INSERT INTO `sys_menu` (`menu_name`, `parent_id`, `order_num`, `url`, `targ




INSERT INTO `sys_config` ( `config_name`, `config_key`, `config_value`, `co
VALUES
        ( 'MAC地址参数', 'mac_address_type_name', 'wlan4', 'Y', 'admin', '20
```

`公钥RSAPublicKey`：

```
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDNMGBEQTyKC+2zh2CUxEDI8M3j
SpSFIwDjW1JLCzJCQDTclGqX/A3DWsS+PMrFfRxd0w9rIoLx1DjF2KpJmAzUKpFHt
ptm8tWPAaPgcl58x7zJvnYUHIMcejXhUZfqE8qHVVQeNaUGxq2Nn9S2s+dFVJ9zSc
327mS9Ym0xt2wTYQIDAQAB
```

`私钥RSAPrivateKey`：

```
MIICdwIBADANBgkqhkiG9w0BAQEFAASCAmEwggJdAgEAAoGBAM0wYERBPIoL7bOH
YJTEQMjwzeNKlIUjAONbUksLMkJANNyUapf8DcNaxL48ysV9HF3TD2sigvHUOMXYq
kmYDNQqkUe2m2by1Y8Bo+ByXnzHvMm+dhQcgxx6NeFRl+oTyodVVB41pQbGrY2f1L
az50VUn3NJzfbuZL1ibTG3bBNhAgMBAAECgYAnewFq4KqHXVq1TP0WYtScVUkoTLd
bDqsjE/U3n0XiXXxOXPhNmiAD0Lk+aXASo8oLe4rh8FgFs/Hgj5nkYy9EY2B+e4nz
dGkaXpLCYJQL8gYFnh+TlcJnhcKBHkQuBGORVghDWOky0bMyKxdPlO20B8AySThHa
Q6k/CWjJFkMKQJBAOdlQbp//cCRImTI1GIW3ghQuqGrOtrQHnLO4My/FsW5elaqKh
U3sfAAa+Tk7jWm0cdsu5OYFjvsjzmrea0Y5wsCQQDjAcJCQn275itpTtOjqLDGH0l
f2AKFv+DARBDdvZNDULirzUpiMT6xHlWyIu3NXD6oRofvKiF40ADKOJPSsgLDAkEA
hLM9Dod4LocNAl9p7RbQHlStsylPh0zzKUQbB1BO7vV/g1Py7lgSY2Daf1oKgho5E
/T3XZ1KovL28Eu2a95sIQJAdgITD0jrfgOAa8UrpjdqdgQLq8RqQ0dlOe3oaq1DYc
oTxlP5KCR7d63SVMFtp8UdWXMY9U6MifZvgrB4wB4D6wJBAILQrxAtbzMG9qD9jLB
jBy6dVpMTGS2NqSUlIHz5CFdGT5d9gLMjLJ5cCovKgdzrX+pyAFmUIy7n3BFkGHtW
```

cTs=

```
      try {
    //      InetAddress inetAddress = InetAddress.getLocalHost();
    //      String macAddress = SystemUtil.getMacAddress(inetAddress);
    //String systemCode = Md5Utils.hash(macAddress).toUpperCase();
          String systemCode = "EA60BF27E27CE853EEBF4E8BACF69CCA";
          String RSAPublicKey = "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDNMC
          String RSAPrivateKey = "MIICdwIBADANBgkqhkiG9w0BAQEFAASCAmEwggJdA

          JsonObject jsonContainer =new JsonObject();
    //为当前的json对象添加键值对
          jsonContainer.addProperty("systemCode", systemCode);
          jsonContainer.addProperty("startDate",  "2022-12-01 00:00:00");
          jsonContainer.addProperty("finishedDate", "2022-12-31 00:00:00");

    //加密
          String encrypt = RsaUtil.encrypt(String.valueOf(jsonContainer), R
    //解密
          String decrypt = RsaUtil.decrypt(encrypt, RSAPrivateKey);
          log.info(123);
      }catch (Exception e){

      }




// 第一次使用时获取，获取后保存公私钥，不要重复获取
//生成秘钥
Map<String, String> stringStringMap = RsaUtils.generateKeyPair();
String RSAPublicKey = stringStringMap.get(PUBLIC_KEY);
String RSAPrivateKey = stringStringMap.get(PRIVATE_KEY);
try {
    InetAddress inetAddress = InetAddress.getLocalHost();
    String macAddress = SystemUtil.getMacAddress(inetAddress);
    //String systemCode = Md5Utils.hash(macAddress).toUpperCase();
    String systemCode = "机器码";
    String RSAPublicKey = "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDNMGBEQTyK
    String RSAPrivateKey = "MIICdwIBADANBgkqhkiG9w0BAQEFAASCAmEwggJdAgEAAoG

    JsonObject jsonContainer =new JsonObject();
    //为当前的json对象添加键值对
    jsonContainer.addProperty("systemCode", systemCode);
    jsonContainer.addProperty("finishedDate", "2022-12-31 00:00:00");

    //加密
    String encrypt = RsaUtil.encrypt(String.valueOf(jsonContainer), RSAPubl
    //解密
    String decrypt = RsaUtil.decrypt(encrypt1, RSAPrivateKey);
}catch (Exception e){

}
```

```
package com.ruoyi.common.utils;

import cn.hutool.crypto.SecureUtil;
```

```
import cn.hutool.crypto.asymmetric.KeyType;
import cn.hutool.crypto.asymmetric.RSA;
import org.apache.commons.codec.binary.Base64;

import java.security.KeyPair;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.util.HashMap;
import java.util.Map;

public class RsaUtil {

    /**
     * 类型
     */
    public static final String ENCRYPT_TYPE = "RSA";

    /**
     * 获取公钥的key
     */
    public static final String PUBLIC_KEY = "RSAPublicKey";

    /**
     * 获取私钥的key
     */
    public static final String PRIVATE_KEY = "RSAPrivateKey";

    /**
     * 公钥加密
     * @param content 要加密的内容
     * @param publicKey 公钥
     */
    public static String encrypt(String content, PublicKey publicKey) {
        try{
            RSA rsa = new RSA(null,publicKey);
            return rsa.encryptBase64(content, KeyType.PublicKey);
        }catch (Exception e){
            e.printStackTrace();
        }
        return null;
    }

    /**
     * 公钥加密
     * @param content 要加密的内容
     * @param publicKey 公钥
     */
    public static String encrypt(String content, String publicKey) {
        try{
            RSA rsa = new RSA(null,publicKey);
            return rsa.encryptBase64(content, KeyType.PublicKey);
        }catch (Exception e){
            e.printStackTrace();
        }
        return null;
    }


    /**
     * 私钥解密
```

```
     * @param content 要解密的内容
     * @param privateKey 私钥
     */
    public static String decrypt(String content, PrivateKey privateKey) {
        try {
            RSA rsa = new RSA(privateKey,null);
            return rsa.decryptStr(content, KeyType.PrivateKey);
        } catch (Exception e){
            e.printStackTrace();
        }
        return null;
    }

    /**
     * 私钥解密
     * @param content 要解密的内容
     * @param privateKey 私钥
     */
    public static String decrypt(String content, String privateKey) {
        try {
            RSA rsa = new RSA(privateKey,null);
            return rsa.decryptStr(content, KeyType.PrivateKey);
        } catch (Exception e){
            e.printStackTrace();
        }
        return null;
    }

    /**
     * 获取公私钥-请获取一次后保存公私钥使用
     * @return
     */
    public static Map<String,String> generateKeyPair() {
        try {
            KeyPair pair = SecureUtil.generateKeyPair(ENCRYPT_TYPE);
            PrivateKey privateKey = pair.getPrivate();
            PublicKey publicKey = pair.getPublic();
            // 获取 公钥和私钥 的 编码格式（通过该 编码格式 可以反过来 生成公钥和私钥
            byte[] pubEncBytes = publicKey.getEncoded();
            byte[] priEncBytes = privateKey.getEncoded();

            // 把 公钥和私钥 的 编码格式 转换为 Base64文本 方便保存
            String pubEncBase64 = Base64.encodeBase64String(pubEncBytes);
            String priEncBase64 = Base64.encodeBase64String(priEncBytes);

            Map<String, String> map = new HashMap<String, String>(2);
            map.put(PUBLIC_KEY,pubEncBase64);
            map.put(PRIVATE_KEY,priEncBase64);

            return map;
        } catch (Exception e){
            e.printStackTrace();
        }
        return null;
    }
}
```

```
log.info("##Time Checking Service Running: {}",LocalDateTime.now());
  String encryptMacAddress = SystemUtil.getMacAddressMd5();
```

```
       if (Math.abs(datePoorDay - i1) >1){
```

```
public class SystemUtil {
    public static String getMacAddressMd5() {
        String encryptMacAddress = null;
        String macAddress= "";
        try {
            RedisUtil redisUtil = SpringUtils.getBean(RedisUtil.class);
            if (redisUtil.hasKey("pcMacAddress")){
                macAddress = redisUtil.get("pcMacAddress").toString();
            }else{
                Map<String, Object> localInetMac = NetworkUtil.getLocalInet
                if (localInetMac == null){
                    throw new Exception("获取mac地址异常:null");
                }
                macAddress = localInetMac.get("mac").toString();
                redisUtil.set("pcMacAddress",macAddress);
            }
            log.info("MAC:{}",macAddress);
            encryptMacAddress = Md5Utils.hash(macAddress).toUpperCase();
        } catch (Exception e) {
            log.error("获取mac地址异常:{}",e.toString());
        }
        return encryptMacAddress;
    }


    ____________________________________________________________________
         String systemMacName = ni.getName();
            log.info("address:{},canonicalHostName:{},hostAddress:{},hostNa
                    address,
                    canonicalHostName,
                    hostAddress,
                    hostName,
                    displayName,
                    hardwareAddress,systemMacName);
            String macAddressName = "enp2s0";




--------------------------------------------------------------------------
   public class PcActivationCodeService implements ApplicationRunner {

    private final PcActivationMapper pcActivationMapper;

    private RedisUtil redisUtil;
    @Override
    public void run(ApplicationArguments args) {
        try {
            redisUtil.del(URLConstant.MAC_ADDRESS_TYPE_NAME);
            redisUtil.del("pcMacAddress");
            redisUtil.del("pcActivation");
            String encryptMacAddress = SystemUtil.getMacAddressMd5();
            //查询是否有激活码
            PcActivation pcActivation = pcActivationMapper.selectPcActivati
            if (pcActivation != null) {
```

空白文档

版本记录

> 数据库变更

```
ALTER TABLE `pc_ai_model`
ADD COLUMN `model_type` varchar(50) NULL COMMENT '模型类型' AFTER `model_pat

ALTER TABLE `pc_monitoring_device`
ADD COLUMN `need_cut_image` tinyint(1) NULL COMMENT '是否需要截图： 1 是  0
ADD COLUMN `background_image` varchar(255) NULL COMMENT '背景图' AFTER `need


INSERT INTO `sys_dict_type` (`dict_name`, `dict_type`, `status`, `create_by

INSERT INTO `sys_dict_data` ( `dict_sort`, `dict_label`, `dict_value`, `dic

INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict



-- 避免截图大小不一致
INSERT INTO `aicsp_ipc_flask`.`sys_config` (`config_name`, `config_key`, `c
```

ALTER TABLE `pc_monitoring_device` ADD COLUMN `need_cut_image` tinyint(1) NULL COMMENT '是否需要截图： 1 是 0 否' AFTER `address`, ADD COLUMN `background_image` varchar(255) NULL COMMENT '背景图' AFTER `need_cut_image`

空白文档

起 `echarts-convert` 服务

phantomjs D:\Development\Phantomjs\echartsconvert\echarts-convert.js -s -p 6666

```
chmod 755 -R /usr/local/server/aicsp/uploadPath/statistics/

##起 `echarts-convert` 服务
/usr/local/server/aicsp/uploadPath/statistics/phantomjs/bin/phantomjs /usr/
```

statistics-dev.zip

```
INSERT INTO `sys_menu` (`menu_name`, `parent_id`, `order_num`, `url`, `targ

SELECT @topParentId := LAST_INSERT_ID();

-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('周统计报表列表', @topParentId, '1', '/pc/pc_statistics_week', 'C', '0

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('周统计报表列表查询', @parentId, '1',  '#',  'F', '0', 'pc:pc_statistic

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('周统计报表列表新增', @parentId, '2',  '#',  'F', '0', 'pc:pc_statistic

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('周统计报表列表修改', @parentId, '3',  '#',  'F', '0', 'pc:pc_statistic

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('周统计报表列表删除', @parentId, '4',  '#',  'F', '0', 'pc:pc_statistic

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('周统计报表列表导出', @parentId, '5',  '#',  'F', '0', 'pc:pc_statistic


-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('月统计报表列表', @topParentId, '1', '/pc/pc_statistics_month', 'C', '

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('月统计报表列表查询', @parentId, '1',  '#',  'F', '0', 'pc:pc_statistic

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('月统计报表列表新增', @parentId, '2',  '#',  'F', '0', 'pc:pc_statistic

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('月统计报表列表修改', @parentId, '3',  '#',  'F', '0', 'pc:pc_statistic

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
```

```
values('月统计报表列表删除', @parentId, '4',  '#',  'F', '0', 'pc:pc_statistic

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('月统计报表列表导出', @parentId, '5',  '#',  'F', '0', 'pc:pc_statistic



INSERT INTO `sys_menu` (`menu_name`, `parent_id`, `order_num`, `url`, `targ

SELECT @parentId2 := LAST_INSERT_ID();

INSERT INTO `sys_menu` (`menu_name`, `parent_id`, `order_num`, `url`, `targ
INSERT INTO `sys_menu` (`menu_name`, `parent_id`, `order_num`, `url`, `targ
INSERT INTO `sys_menu` (`menu_name`, `parent_id`, `order_num`, `url`, `targ


-- 告警工单
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe

SELECT @menu_id1 := LAST_INSERT_ID();

INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
SELECT @menu_id2 := LAST_INSERT_ID();
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
SELECT @menu_id3 := LAST_INSERT_ID();
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
SELECT @menu_id4 := LAST_INSERT_ID();
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe
INSERT INTO `sys_menu`(`menu_name`, `parent_id`, `order_num`, `url`, `targe

INSERT INTO `sys_dict_type`(`dict_name`, `dict_type`, `status`, `create_by`
INSERT INTO `sys_dict_type`(`dict_name`, `dict_type`, `status`, `create_by`
INSERT INTO `sys_dict_type`(`dict_name`, `dict_type`, `status`, `create_by`

INSERT INTO `sys_dict_data`(`dict_sort`, `dict_label`, `dict_value`, `dict_
INSERT INTO `sys_dict_data`(`dict_sort`, `dict_label`, `dict_value`, `dict_
INSERT INTO `sys_dict_data`(`dict_sort`, `dict_label`, `dict_value`, `dict_
INSERT INTO `sys_dict_data`(`dict_sort`, `dict_label`, `dict_value`, `dict_
INSERT INTO `sys_dict_data`(`dict_sort`, `dict_label`, `dict_value`, `dict_
INSERT INTO `sys_dict_data`(`dict_sort`, `dict_label`, `dict_value`, `dict_
INSERT INTO `sys_dict_data`(`dict_sort`, `dict_label`, `dict_value`, `dict_
INSERT INTO `sys_dict_data`(`dict_sort`, `dict_label`, `dict_value`, `dict_
INSERT INTO `sys_dict_data`(`dict_sort`, `dict_label`, `dict_value`, `dict_
```

> 数据报表 `alarm_result`

```
id                          #id
descript                    #统计描述
type                        #报表类型    1 周报表   2   月报表
begin_date                  #统计的开始时间
end_date                    #统计的结束时间
created_date     #创建时间
```

> 告警报表总览 `alarm_result_daily_statistics`

- 每天的告警总数

```
id                                  #id
date                        #统计的时间   每日的时间
nums                        #统计的数量
create_time           #创建时间
```

> 告警统计数量 `alarm_result_total_statistics`

```
-- alarm_result_model_total
id                                  #ID
result_id                   #结果ID
model_id                    #模型ID
nums                        #数量
proportion       #占比
sort                        #排名

-- alarm_result_model_total_device
id
model_total_id
device_id
nums
proportion
begin_date                  #统计的开始时间
end_date                    #统计的结束时间
```

```
-- alarm_result_device_total
id                                  #ID
result_id                   #结果ID
device_id                   #设备ID
nums                        #数量
proportion       #占比
sort                        #排名

-- alarm_result_device_total_model
id
device_total_id
model_id
nums
proportion
begin_date                  #统计的开始时间
end_date                    #统计的结束时间
```

每天的模型告警统计

```
id                                      #ID
daily_statistics_id #daily_statistics_id
model_id                        #模型id
nums                            #告警数量
proportion              #占比
```

每天的设备告警统计

```
id                                      #ID
daily_statistics_id #daily_statistics_id
device_id                       #设备id
nums                            #告警数量
proportion              #占比
```

工单统计

```
id                                      #ID
model_id                        #模型ID
nums                            #告警数量
proportion              #占比
finishd                         #已完成
sourt                           #排序
```

> 告警报表详情 （算法-设备排名）
>
> - 每个设备每天的告警数量

```
id
daily_statistics_id #daily_statistics_id_id
device_id                       #设备ID
nums                            #告警数量
proportion              #占比
```

告警报表详情（设备） `alarm_result_device_detail`

- 每个设备的告警数量

```
id                              #ID
total_id                #alarm_result_total_id
device_id               #设备id
nums                    #告警数量
proportion      #占比
sort                    #排名
```

告警报表详情 （设备-算法排名）

```
id
device_detail_id        #alarm_result_detail_id
model_id                        #模型ID
device_id                       #设备ID
nums                            #告警数量
proportion              #占比
```

```
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_1.
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_2.
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_3.
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_4.
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_5.
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_6.
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_7.
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_8.
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_9.
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_10
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_11
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_12
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_13
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_14
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_15
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_16
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_17
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_18
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_19
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_20
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_21
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_22
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/week/周数据报表_23
1
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/month/月数据报表_8
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/month/月数据报表_8
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/month/月数据报表_8
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/month/月数据报表_8
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/month/月数据报表_8
/usr/local/server/aicsp/uploadPath/statistics/echarts/pdf/month/月数据报表_8
```

空白文档

版本记录

```
CREATE TABLE `pc_law_enforcement_video` (
  `id` int NOT NULL AUTO_INCREMENT COMMENT '主键ID',
  `del_flag` int DEFAULT '0' COMMENT '删除标记',
  `create_id` int DEFAULT NULL COMMENT '创建人ID',
  `create_date` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_c
  `update_id` int DEFAULT NULL COMMENT '更新人ID',
  `update_date` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_c
  `name` varchar(50) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMENT '视频名
  `video_path` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMENT
  `video_type` int DEFAULT NULL COMMENT '视频分类',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=60 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_u


CREATE TABLE `pc_law_enforcement_video_type` (
  `id` int NOT NULL AUTO_INCREMENT COMMENT '主键ID',
  `del_flag` int DEFAULT '0' COMMENT '删除标记',
  `create_id` int DEFAULT NULL COMMENT '创建人ID',
  `create_date` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_c
  `update_id` int DEFAULT NULL COMMENT '更新人ID',
  `update_date` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_general_c
  `name` varchar(50) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci DEFAU
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_ur

CREATE TABLE `pc_download_task` (
  `id` int NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `content` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL COMMENT '压缩包
  `status` tinyint(1) NOT NULL COMMENT '任务状态  0：失败  1：正在压缩 2：压缩成
  `path` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci DEFA
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=31 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_u


INSERT INTO `aicsp`.`sys_menu` (`menu_name`, `parent_id`, `order_num`, `url
-- 按钮父菜单ID
SELECT @topParentId := LAST_INSERT_ID();

-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('执法视频管理', @topParentId, '1', '/pc/law_enforcement_video', 'C', '

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('执法视频管理查询', @parentId, '1',  '#',  'F', '0', 'pc:law_enforceme

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('执法视频管理新增', @parentId, '2',  '#',  'F', '0', 'pc:law_enforceme

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('执法视频管理修改', @parentId, '3',  '#',  'F', '0', 'pc:law_enforceme

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
```

```
values('执法视频管理删除', @parentId, '4',  '#',  'F', '0', 'pc:law_enforceme

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('执法视频管理导出', @parentId, '5',  '#',  'F', '0', 'pc:law_enforceme


-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('执法视频类别', @topParentId, '1', '/pc/law_enforcement_video_type', '

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('执法视频类别查询', @parentId, '1',  '#',  'F', '0', 'pc:law_enforceme

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('执法视频类别新增', @parentId, '2',  '#',  'F', '0', 'pc:law_enforceme

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('执法视频类别修改', @parentId, '3',  '#',  'F', '0', 'pc:law_enforceme

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('执法视频类别删除', @parentId, '4',  '#',  'F', '0', 'pc:law_enforceme

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('执法视频类别导出', @parentId, '5',  '#',  'F', '0', 'pc:law_enforceme



-- 菜单 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('下载任务', @topParentId, '1', '/pc/download_task', 'C', '0', 'pc:dow

-- 按钮父菜单ID
SELECT @parentId := LAST_INSERT_ID();

-- 按钮 SQL
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('下载任务查询', @parentId, '1',  '#',  'F', '0', 'pc:download_task:lis

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('下载任务新增', @parentId, '2',  '#',  'F', '0', 'pc:download_task:add

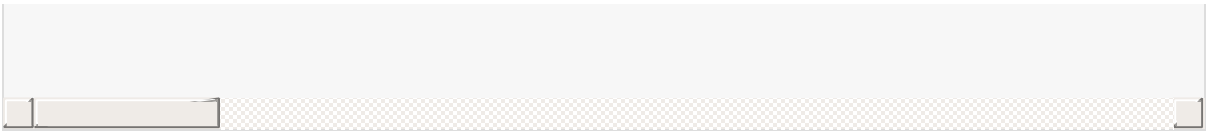
insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('下载任务修改', @parentId, '3',  '#',  'F', '0', 'pc:download_task:edi

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('下载任务删除', @parentId, '4',  '#',  'F', '0', 'pc:download_task:rem

insert into sys_menu (menu_name, parent_id, order_num, url, menu_type, visi
values('下载任务导出', @parentId, '5',  '#',  'F', '0', 'pc:download_task:exp


INSERT INTO `sys_dict_type` (`dict_name`, `dict_type`, `status`, `create_by

INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
INSERT INTO `sys_dict_data` (`dict_sort`, `dict_label`, `dict_value`, `dict
```



启动redis

启动**redis**

```
service redis start/stop/restart
```

启动**mysql**

```
service mysql start
```

启动**SRS**

```
nohup /usr/local/server/srs/trunk/objs/srs -c /usr/local/server/srs/trunk/c
```

启动**Tomcat**

```
/usr/local/server/apache-tomcat/bin/startup.sh
```

启动**saas**平台

```
/usr/local/server/aicsp/ai-ipc.sh start
```

Redis 服务设置 **service**启动

```
#!/bin/sh
# description: Start and Stop redis,Redis is a persistent key-value databas
# chkconfig: 2345 90 10
# chkconfig: 2345 10 90
# description: Start and Stop redis

REDISPORT=6379 #实际环境而定
EXEC=/usr/local/bin/redis-server #实际环境而定
REDIS_CLI=/usr/local/bin/redis-cli #实际环境而定

PIDFILE=/var/run/redis_6379.pid
CONF="/usr/local/server/redis-7.0.3/redis.conf" #实际环境而定

case "$1" in
        start)
                if [ -f $PIDFILE ]
                then
                        echo "$PIDFILE exists, process is already running c
                else
                        echo "Starting Redis server..."
                        $EXEC $CONF
                fi
                if [ "$?"="0" ]
                then
                        echo "Redis is running..."
                fi
                ;;
```

```
        stop)
                if [ ! -f $PIDFILE ]
                then
                        echo "$PIDFILE exists, process is not running."
                else
                        PID=$(cat $PIDFILE)
                        echo "Stopping..."
                        $REDIS_CLI -p $REDISPORT SHUTDOWN
                        while [ -x $PIDFILE ]
                        do
                                echo "Waiting for Redis to shutdown..."
                                sleep 1
                        done
                        echo "Redis stopped"
                fi
                ;;
        restart|force-reload)
                ${0} stop
                ${0} start
                ;;
        *)
                echo "Usage: /etc/init.d/redis {start|stop|restart|force-re
                exit 1
esac
```

docker run
-p 9002:80
–name nginx
-v /home/nginx/conf/nginx.conf:/etc/nginx/nginx.conf
-v /home/nginx/conf/conf.d:/etc/nginx/conf.d
-v /home/nginx/log:/var/log/nginx
-v /home/nginx/html:/usr/share/nginx/html
-d nginx:latest

docker-compose方式安装

docker-compose方式安装

安装 `docker` 服务

> docker服务安装 - 切换管理员

```
sudo su
#输入密码
```

- 查询 `docker` 版本 `shell docker -v Docker version 20.10.21, build 20.10.21-0ubuntu1~20.04.1` 出现上面的情况，证明 `docker` 服务已经安装，跳过这个步骤 “`shell docker -v
- Command ‘docker’ not found, but can be installed with:
- snap install docker # version 20.10.14, or
- apt install docker.io # version 20.10.12-0ubuntu2~20.04.1
- See ‘snap info docker’ for additional versions. 出现上面的情况，代表`docker`服务没有安装，先进行安装操作 shell apt install docker.io “ 安装完成后，再进行 docker -v`验证一下。出现版本号则证明安装成功。

> 安装 `nvidia-docker2` 服务

```
#安装curl
apt  install curl

#逐条执行以下命令
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | apt-key add -

distribution=$(. /etc/os-release;echo $ID$VERSION_ID)

curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-dock

apt-get update

#安装nvidia-docker2软件包并重新加载docker守护程序配置
apt-get install -y nvidia-docker2
pkill -SIGHUP dockerd
```

> docker配置参数修改

```
vim /etc/docker/daemon.json
{
"runtimes": {
        "nvidia": {
            "path": "nvidia-container-runtime",
            "runtimeArgs": []
        }
    },
  "data-root": "/usr/local/server/aicsp/uploadPath/docker",
```

```
  "log-driver":"json-file",
  "log-opts": {"max-size":"100m", "max-file": "3"}
}

systemctl restart docker # 重启docker

docker info | grep "Docker Root Dir"  #查看目录修改是否生效
```

安装 `docker-compose` 服务

先 `docker-compose -version` 验证，如果有版本信息则跳过此步骤，否则进行安装

> 方式1

```
sudo curl -L https://github.com/docker/compose/releases/download/1.29.0/doc

chmod +x /usr/local/bin/docker-compose
```

> 方式2

```
apt update
apt install docker-compose
#检验安装是否成功命令：docker-compose -version
```

安装项目

> 创建基础文件夹
>
> ```
> #进入根目录
> cd /
> #创建文件夹
> mkdir aibox
> #设置权限
> chown ipc ./aibox
> ```
>
> 上传项目包

将 `aicsp.zip` 上传至 `/aibox` 目录中。

```
#解压
unzip /aibox/aicsp.zip
cd /aibox/aicsp
chmod 755 ./build.sh ./restart.sh ./stop.sh
./build.sh
```

> 到这一步，部署完成。
>
> 注意点：
>
> 在安装的过程中，需要进行联网进行安装字体，否则会导致平台缺失字体导致验证码不显示问题

docker方式安装

在线安装

第一步：检查是否安装了**docker**

```
sudo su #输入密码

root@mypc-desktop:/home/mypc# docker -version
Command 'docker' not found, but can be installed with:
snap install docker     # version 20.10.14, or
apt  install docker.io  # version 20.10.12-0ubuntu2~20.04.1
See 'snap info docker' for additional versions.
################################
```

第二步：安装**docker**

> 注意 ： 已安装**docker**的跳过此步骤

```
root@mypc-desktop:/home/mypc# apt  install docker.io
-----------------------------------------------------
root@mypc-desktop:/home/mypc# docker -v
Docker version 20.10.12, build 20.10.12-0ubuntu2~20.04.1  #  出现版本号，证明
```

第三步：拉取镜像

```
#登录dockerHub账号
root@mypc-desktop:/home/mypc# docker login -u zhangfayuan
Password: zhang01110010
WARNING! Your password will be stored unencrypted in /root/.docker/config.j
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-sto

Login Succeeded
#拉取镜像
root@mypc-desktop:/home/mypc# docker pull zhangfayuan/aibox:v1.0.1
v1.0.1: Pulling from zhangfayuan/aibox
40dd5be53814: Pull complete
0645c48e225b: Pull complete
8889d023c18e: Pull complete
74115353f57d: Pull complete
9e0ea72fe76c: Pull complete
13f51f3f80fd: Pull complete
c4ba9103d17d: Pull complete
8d9b8d71f868: Pull complete
97badaa6a776: Pull complete
63041bc7286a: Pull complete
5f237510596a: Pull complete
e84ba40a3ba2: Pull complete
3514d699b3b1: Pull complete
d6379c113197: Pull complete
383de3c5d7cf: Pull complete
32bc74c8408d: Pull complete
08d2c3256245: Pull complete
bebbd0bac501: Pull complete
0a726c81dddb: Pull complete
7f13c73ec742: Pull complete
f94bde7b7670: Pull complete
```

```
68a87c8baf64: Pull complete
Digest: sha256:b564fc1be38e3682d6aaafded7f7d22ab46b7080495b56f4815d4d932acf
Status: Downloaded newer image for zhangfayuan/aibox:v1.0.1
docker.io/zhangfayuan/aibox:v1.0.1
root@mypc-desktop:/home/mypc# docker images
REPOSITORY          TAG       IMAGE ID       CREATED        SIZE
zhangfayuan/aibox   v1.0.1    b977035f7cd9   17 hours ago   62.8GB
```

第四步：启动镜像

https://blog.csdn.net/m0_51309197/article/details/123261224

```
docker run -itd --net=host --restart=always   --name aibox --runtime=nvidia


#docker run -itd --net=host --restart=always -v /usr/local/server/aicsp/upl
#如果出现  docker: Error response from daemon: Unknown runtime specified nvi

docker logs -f aibox # 查看启动日志信息
```

第五步：设置**docker**服务检测

```
#创建目录库
mkdir aibox
cd aibox

#创建docker服务检测服务脚本
vim check-docker-service.sh
```

写入以下内筒：

```
#!/bin/sh
#Filename: check-docker-service.sh


PROC_NAME=dockerd
ProcNumber=`ps -ef |grep -w $PROC_NAME|grep -v grep|wc -l`
time1=$(date)
if [ $ProcNumber -le 0 ];then
   echo "$time1 :Docker service is not run" >> /root/check-docker-service.l
   systemctl start docker
   echo "$time1 :command:{ systemctl start docker } is execute" >> /root/ch

else
   echo "$time1 :Docker service is running.."  >> /root/check-docker-servic
fi
```

设置定时任务执行上面的脚本

```
crontab -e
#每5分钟执行一次
*/5  * * * * sh /aibox/check-docker-service.sh
#设置服务每天凌晨2点半进行重启
30 2 * * * docker restart aibox
```

第六步：修改平台IP配置

tar包部署

第一步：检查是否安装了docker

```
sudo su #输入密码

root@mypc-desktop:/home/mypc# docker -version
Command 'docker' not found, but can be installed with:
snap install docker     # version 20.10.14, or
apt  install docker.io  # version 20.10.12-0ubuntu2~20.04.1
See 'snap info docker' for additional versions.
################################```
```

第二步：安装docker

> 注意：已安装docker的跳过此步骤

```
root@mypc-desktop:/home/mypc# apt  install docker.io
------------------------------------------------------
root@mypc-desktop:/home/mypc# docker -v
Docker version 20.10.12, build 20.10.12-0ubuntu2~20.04.1  #  出现版本号，证明
```

第三步：安装nvidia-smi

```
1、执行以下命令
1.1 curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | apt-key add
1.2 distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
1.3 curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-
1.4 apt-get update

2、安装nvidia-docker2软件包并重新加载docker守护程序配置
2.1 apt-get install -y nvidia-docker2
2.2 pkill -SIGHUP dockerd
```

第四步：拷贝tar包

从硬盘中将tar包拷贝到服务器任意目录

第五步：tar包转成镜像

- 进入tar包的目录
- 执行命令

```
  docker load -i ./aicsp_v2.0.1.tar #大概需要25分钟左右
```

第六步：启动容器

```
#查看镜像
root@ipc-NUC9i7QNX:/usr/local/DockerImages# docker images
REPOSITORY      TAG       IMAGE ID       CREATED        SIZE
hjkj123/aicsp   v2.0.1    47e9e76c41f7   23 hours ago   67.5GB
#地洞容器
root@ipc-NUC9i7QNX:/usr/local/server# docker run -itd --net=host --restart=
415e19d4e5bf90a5489e3277dd487d4dffa188e4a6ff5d2e187ed0ed41e3a4fb
root@ipc-NUC9i7QNX:/usr/local/server#
#查看启动结果
root@ipc-NUC9i7QNX:/usr/local/server/aicsp/upload/ffmpeg# docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED
415e19d4e5bf   hjkj123/aicsp:v2.0.1   "/bin/sh -c '/usr/lo…"   About a minu
#查看日志
root@ipc-NUC9i7QNX:/usr/local/server/aicsp/upload/ffmpeg# docker logs -f ai
正在清理异常退出所占用的PID文件...
清理完成...
启动redis服务...
Starting Redis server...
存在mysql.sock.lock文件
正在清理mysql.sock.lock文件
清理完成...
修改Mysql目录权限....
修改mysql用户信息....
usermod: no changes
启动mysql服务...
 * Starting MySQL database server mysqld                                [
启动SRS服务...
启动大屏服务...
Using CATALINA_BASE:   /usr/local/server/apache-tomcat
Using CATALINA_HOME:   /usr/local/server/apache-tomcat
Using CATALINA_TMPDIR: /usr/local/server/apache-tomcat/temp
Using JRE_HOME:        /usr
Using CLASSPATH:       /usr/local/server/apache-tomcat/bin/bootstrap.jar:/u
Tomcat started.
启动AI平台....
。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。 。
          _____ _____  _____ _____     _____ ______ _______      _______ _
     /\   |_   _/ ____|/ ____|  __ \   / ____|  ____|  __ \ \    / /_   _/
    /  \    | || |     | (___ | |__) | | (___ | |__  | |__) \ \  / /  | || |
   / /\ \   | || |      \___ \|  ___/   \___ \|  __| |  _  / \ \/ /   | || |
  / ____ \ _| || |____ ____) | |       ____) | |____| | \ \  \  /   _| || |
 /_/    \_\_____\_____|_____/|_|      |_____/|______|_|  \_\  \/   |_____\_
```

第七步：登录平台

> 登录 http://xx.xx.xx.xx:8091/aicsp

第八步：修改IP

- 获取本机的IP地址： `ifconfig`
- 登录到平台修改参数：
  - 系统管理 –> 参数管理
- 修改以下参数的IP为获取到的本机的IP

    - `big_screen_url_admin` #管理员大屏访问地址
    - `big_screen_url` #大屏访问地址
    - `screen_url` #大屏缓存数据图片链接
    - `url_ip` #大喇叭PHP包装接口访问地址
    - `srs_ip` #SRS推流服务IP

- 修改大屏IP
    - 系统管理 -》 监控大屏配置 -》 修改为本机IP

- 然后退出，刷新缓存，重新登陆
- 访问大屏测试 测试大屏是否正常

第十步：创建任务

- 配置视频通道模型
    - 视频能力管理 -> 配置模型列表
    - 新建配置【按需配置】

- 创建任务 视频能力管理 -》 监控任务 -》创建[按需创建] 【注意：识别模式 选择视频分许-不推流】

- 创建完成后 看告警列表是否有告警数据

> 可以的话 就成功了 不可以的话，咨询海波。。。

学习账号

极客时间：18115178069 051612

安装`Prometheus(普罗米修斯)`

安装 `Prometheus(普罗米修斯)`

> 领取镜像

```
docker pull prom/node-exporter
docker pull prom/prometheus
docker pull grafana/grafana
```

> 启动 node-exporter

```
docker run -d -p 9100:9100 \
  -v "/proc:/host/proc:ro" \
  -v "/sys:/host/sys:ro" \
  -v "/:/rootfs:ro" \
  --net="host" \
  prom/node-exporter
```

> 启动 prometheus

```
docker run  -d \
  -p 9090:9090 \
  -v /opt/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml  \
  prom/prometheus
```

> 启动grafana

```
#创建文件夹
mkdir /opt/grafana-storage
#设置权限
chmod 755 -R /opt/grafana-storage

docker run -d \
  -p 3000:3000 \
  --name=grafana \
  -v /opt/grafana-storage:/var/lib/grafana \
  grafana/grafana
```

8083 8085

面板使用 ： 8919

数据库缺少字段

```
ALTER TABLE `aicsp_bottom_server`.`pc_monitoring_device`
ADD COLUMN `sort` int(11) NULL DEFAULT NULL AFTER `syn_time`,
ADD COLUMN `security_id` int(11) NULL DEFAULT 0 COMMENT 'securityId' AFTER
DROP PRIMARY KEY,
ADD PRIMARY KEY (`id`) USING BTREE
```


```
{
  "taskId":1,
  "pcMonitoringDeviceModelId":18,
  "monitoringDeviceId":89,
  "createDate":"2023-08-16 15:43:56",
  "frontAlgoResult":[
    {
      "bbox":[
        900.0,500.0,1000.0,700.0
      ],
      "category":"chacar",
      "socre":0.7640645503997803
    }
  ],
  "needCheckForFirstResult":"false"
}
```

空白文档

10.8.26.62 部署完成，算法已升级，清理程序已更新【修改会删除算法图片】，已修改删除策略，已优化AI数据类型

10.8.26.62 部署完成，算法已升级，清理程序已更新【修改会删除算法图片】，已修改删除策略，已优化**AI**数据类型

172.31.100.70 部署完成，算法已升级，清理程序已更新【修改会删除算法图片】，已修改删除策略

172.31.100.76 部署完成，算法已升级，清理程序已更新【修改会删除算法图片】，已修改删除策略

172.31.100.201 部署完成，算法已升级，【清理程序未更新】

172.31.100.203 部署完成，算法已升级，清理程序已更新【修改会删除算法图片】，已修改删除策略

172.31.100.103 无法访问

172.31.100.202 无法访问

其他

Ubuntu里docker的卸载与安装

一键部署目录结构

一键部署目录结构

```
├── ai_deploy.sh
├── conf
│   ├── boliving
│   │   └── config.toml
│   ├── mysql
│   │   └── my.cnf
│   ├── nginx
│   │   └── conf.d
│   │       └── default.conf
│   └── redis
│       └── redis.conf
├── data
│   ├── mysql
│   ├── nginx
│   │   └── html
│   └── redis
├── docker-compose.yml
├── dockerFile
│   ├── ai
│   │   ├── Dockerfile
│   │   ├── ai_platform.tar
│   │   └── entrypoint.sh
│   ├── jar
│   │   ├── .DS_Store
│   │   ├── Dockerfile
│   │   └── start.jar
│   └── mysql
│       ├── .DS_Store
│       ├── Dockerfile
│       └── ai_algo_aat.sql
├── logs
│   ├── boliving
│   ├── java
│   ├── mysql
│   └── nginx
├── other
│   ├── comm
│   ├── etcdocker
│   │   ├── daemon.json
│   │   └── key.json
│   ├── license
│   ├── models
│   └── sh
│       ├── delLogs.sh
│       ├── remain.sh
│       └── remain.txt
└── tar
    ├── ai_platform_v1.tar
    ├── boliving_1.0.0.03121.tar
    ├── mysql_8.0-oracle.tar
    ├── nginx_1.18-alpine.tar
    ├── openjdk_8-jre.tar
```

```
    └── redis_6.0.16.tar
```

第一步：安装 **docker-compose(需联网)**

```
#docker-compose -v  查看是否有docker-compose
#如果未安装，执行下面命令进行安装
apt-get install docker-compose
```

第二步：上传项目到服务器

上传 ai_deploy.zip 压缩包到 `/home/nvidia/` 目录下【统一目录放到该目录下，理论上可以放在任意目录下】

第三步：解压项目压缩包

```
cd /home/nvidia/
unzip ./ai_deploy.zip
```

第四步：添加默认使用**GPU**相关配置【如果不先进行此步骤，会导致模型开启失败报错】

> 注意：这一步很关键，如果不进行设置，将导致算法无法开启
>
> Tip:如果这一步忘记了，按顺序执行以下命令进行重新部署：
>
> `docker rm -f $(docker ps -aq)` #删除所有容器 `docker rmi -f $(docker images -aq)` #查询出所有的镜像id 进行递归批量容器命令
>
> `sh /home/nvidia/ai_deploy/ai_deploy.sh`

第一步：

添加 `"default-runtime":"nvidia",` 到 `/etc/docker/daemon.json` ，操作如下：

```
vim /etc/docker/daemon.json

{
        "default-runtime":"nvidia",#新增
        "runtimes": {
        "nvidia": {
            "path": "nvidia-container-runtime",
            "runtimeArgs": []
        }
    }
}
```

第二步：

重启docker服务

```
sudo systemctl restart docker.service
```

第五步：上传授权文件

上传授权文件[`license.lic` , `publicCerts.keystore`]到 `/home/nvida/ai_deploy/other/license` 目录下

生成license的项目地址：生成license授权文件项目

第六步：执行安装脚本

> 注意：
>
> 如果盒子是连的 `wifi` ，请先进行盒子的IP固定，否则会导致授权验证不通过

```
cd /home/nvidia/ai_deploy
sh ./ai_deploy.sh
```

脚本内容如下：

```
#!/bin/bash

echo  "## ------ 一键化部署【开始】 ------ ##"

echo  "## ------ 删除历史容器 ------ ##"
docker rm -f mysql
docker rm -f redis
docker rm -f boliving
docker rm -f nginx
docker rm -f ai_platform
docker rm -f ai_algo_aat

echo  "## ------ load 镜像包 ------ ##"
cd ./tar
docker load -i redis_6.0.16.tar
docker load -i boliving_1.0.0.03121.tar
docker load -i mysql_8.0-oracle.tar
docker load -i nginx_1.18-alpine.tar
docker load -i openjdk_8-jre.tar
docker load -i ai_platform_v1.tar
echo  "## ------ load 镜像包 完成 ------ ##"
```

查看运行情况

```
root@nx:~# docker ps
CONTAINER ID        IMAGE                   COMMAND                  CREATED
7a24cc83fce7        aideploy_ai             "/bin/sh -c /ai/temp…"   About a
9ebab568ba1c        redis:6.0.16            "redis-server /etc/r…"   About a
6115243348e9        aideploy_web            "java -Djava.securit…"   About a
927e6adfcc99        boliving:1.0.0.03121    "/boliving/bin/boliv…"   About a
7e9f6be32b37        nginx:1.18-alpine       "/docker-entrypoint.…"   About a
96690d458d3f        mysql:8.0-oracle        "docker-entrypoint.s…"   About a
root@nx:~#
```

查看数据库迁移情况

```
root@nx:/home/nvidia/ai_deploy# docker logs -f mysql
#********
2022-04-28T10:08:35.632409Z 0 [System] [MY-011323] [Server] X Plugin ready
2022-04-28T10:08:35.632479Z 0 [System] [MY-010931] [Server] /usr/sbin/mysql
```

出现 `Bind-address: '::' port: 33060`，`ready for connections. Version: '8.0.28'` 就表明数据库数据迁移成功。

第六步：进行静态资源迁移

根据算法的不同，进行算法图片、算法模型指定目录的迁移

算法图片：`/data2/ai-algo-aat/upload/comm/`

算法模型：`/data2/ai-algo-aat/cache/{versionId}/models`

第七步：新增开机自启动脚本

```
vim /etc/rc.local
#新增以下内容
#!/bin/bash
sleep 1
cd /home/nvidia/ai_deploy/
sudo sh ai_deploy.sh
exit 0
```

> 注意：
>
> 如果系统为Ubuntu18,该方法将不生效。操作方法如下：

ubuntu 18.04 不再使用 inited 管理系统，改用 systemd

第一步：

```
ln -fs /lib/systemd/system/rc-local.service /etc/systemd/system/rc-local.se
```

第二步：

```
sudo echo "
[Install]
WantedBy=multi-user.target
Alias=rc-local.service
" >> /etc/systemd/system/rc-local.service
```

第三步：

```
chmod +x /etc/rc.local
```

第八步：新增定时任务

```
root@nx:~# crontab -e
#添加图片清理脚本 一小时一次
0 */1 * * * sh /home/nvidia/ai_deploy/other/sh/remain.sh >> /home/nvidia/ai
#添加日志清理脚本  每天23:30
30 23 * * * sh /home/nvidia/ai_deploy/other/sh/delLogs.sh >> /home/nvidia/a
```

更新项目

更新平台、算法或者是前端代码，只需要根据一键部署目录结构将对应的部署包放到指定的目录中，执行 `sh /home/nvidia/ai_deploy/ai_deploy.sh` 即可完成更新

其他：

清理**docker**容器

```
docker rm -f $(docker ps -aq) #删除所有容器
docker rmi -f $(docker images -aq) #查询出所有的镜像id    进行递归批量**容器命令**
```

算法迁移

```
#算法
algo
#算法数据库
algo_db
#算法数据库子表表
algo_db_sub
#算法模型
algo_version
#算法模型训练结果表
algo_version_result
#算法版本子表
algo_version_sub
algo_version_sub_result
```

卸载**docker-compose**

```
#apt方式安装
apt remove docker-compose
#二进制文件安装
sudo rm /usr/local/bin/docker-compose
```

应用服务 WEB 访问端口

```
project:
    name: start
# 应用服务 WEB 访问端口
server:
    port: 6060
# 是否开启debug模式
debug: false
# 解决一直打印日志
logging:
    level:
        com.baijiayun.blazers.ai.infrastructure.gatewayimpl: DEBUG
        com.baijiayun.blazers.ai.infrastructure.controller: TRACE
    config: classpath:log4j2-prod.xml
# 应用名称
spring:
    aop:
        proxy-target-class: true
    resource:
        static-locations: classpath:/static/,classpath:/public/
    # application:
    #     name: ai-aat-service
    main:
        # 允许覆盖Bean
        allow-bean-definition-overriding: true

    servlet:
        multipart:
            # 文件上传大小限制
            max-file-size: 1024MB
            max-request-size: 1024MB
        #resources:
        # spring 静态资源扫描路径
        #static_locations: classpath:/static/
    datasource:
        driver-class-name: com.mysql.cj.jdbc.Driver
        url: jdbc:mysql://127.0.0.1:3306/ai_algo_aat?useUnicode=true&charac
        # 本地链接MySQL，用公网地址： mysql-internet-cn-north-1-5f171159190246
        username: root
        password: 123456
        # 配置数据源相关:使用 HikariCP 数据源
        hikari:
            # 连接池名字
            pool-name: ai-aat-service-hikariCP
            # 等待连接池分配连接的最大时长（毫秒），超过这个时长还没可用的连接则发生SQ
            connection-timeout: 30000
            # 最小空闲连接数量，默认值10
            minimum-idle: 5
            # 最大连接数（包括空闲和正在使用的连接），默认值是10
            maximum-pool-size: 15
            # 是否自动提交池中返回的连接。默认值为true。
            auto-commit: true
            # 空闲时间。仅在minimum-idle小于maximum-poop-size的时候才会起作用。默
            idle-timeout: 600000
            # 一个连接的生命时长（毫秒），超时而且没被使用则被释放（retired），默认:3
            max-lifetime: 28740000
            # 在连接从池中获得连接以确认与数据库的连接仍然存在之前将要执行的查询。
```

```
            connection-test-query: SELECT 1
    redis:
        database: 0
        host: 127.0.0.1
        lettuce:
            pool:
                max-active: 8   #最大连接数据库连接数,设 -1 为没有限制
                max-idle: 8     #最大等待连接中的数量,设 0 为没有限制
                max-wait: -1  #最大建立连接等待时间。如果超过此时间将接到异常。设为
                min-idle: 0     #最小等待连接中的数量,设 0 为没有限制
            shutdown-timeout: 100ms
        password: 123456
        port: 6379
        redisson:
            config:
                singleServerConfig:
                    idleConnectionTimeout: 10000
                    connectTimeout: 10000
                    timeout: 3000
                    retryAttempts: 3
                    retryInterval: 1500
                    password: "123456"
                    subscriptionsPerConnection: 5
                    clientName: "redis"
                    address: "redis://127.0.0.1:6379"
                    subscriptionConnectionMinimumIdleSize: 1
                    subscriptionConnectionPoolSize: 50
                    connectionMinimumIdleSize: 32
                    connectionPoolSize: 64
                    database: 0
                    dnsMonitoringInterval: 30000
                threads: 0
                nettyThreads: 0
                codec: "!<org.redisson.codec.JsonJacksonCodec> {}"
                transportMode: "NIO"

# mybatis+mybatis-plus
mybatis:
    mapper-locations: classpath:mybatis/mapper/**/*.xml,classpath*:org/bjy/
    type-aliases-package: com.baijiayun.blazers.ai.*.*,;org.bjy.modules.sys
mybatis-plus:
    mapper-locations: classpath:mybatis/mapper/**/*.xml,classpath*:org/bjy/
    type-aliases-package: com.baijiayun.blazers.ai.*.*,;org.bjy.modules.sys
# 系统线程池设置
async:
    executor:
        # 默认线程池
        default:
            prefix: async-executor-default-     # 配置线程池中的线程的名称前缀
            core_pool_size: 4           # 配置核心线程数
            max_pool_size: 8            # 配置最大线程数
            keep_alive_seconds: 300     # 配置空闲时间
            queue_capacity: 100         # 配置队列大小

redis:
    cache:
        # redisson spring cache
        expires:
            # 数据字典
            dataDict: 3000
```

```
            # 项目任务
            projectTask: 3000

upload:
    prefix: ''
    comm:
        # 服务器上传的文件器访问域名地址前缀
        site: '/upload/comm'
        # 服务器本地上传文件的根目录
        dir_path: '/data2/ai-algo-aat/upload/comm'
        nas_path: '/data2/ai-algo-aat/upload/comm'
    project_task:
        site: '/upload/img'
        dir_path: '/data2/ai-algo-aat/upload/img'
        nas_path: '/data2/ai-algo-aat/upload/img'
    collection:
        site: '/collect/img'
        dir_path: '/data2/ai-algo-aat/collect/img'
        nas_path: '/data2/ai-algo-aat/collect/img'
    stream_video:
        site: '/collect/stream_video'
        dir_path: '/data2/ai-algo-aat/collect/stream_video'
        nas_path: '/data2/ai-algo-aat/collect/stream_video'
    deploy_package:
        site: '/upload/basePackage/packageZip'
        dir_path: '/data2/ai-algo-aat/upload/basePackage/packageZip'
        nas_path: '/data2/ai-algo-aat/upload/basePackage/packageZip'
api:
    algo:
        train:
            # 算法训练的api地址
            url: 'http://0.0.0.0:10004'
            imgModel: '/inferTask'
            qrCodeModel: '/inferBarcode'
    collect:
        task:
            # 发起采集任务
            url: 'http://0.0.0.0:8999'

    three:
        class:
            # 三个课堂
            url: 'https://test-threeclassroom.baijiayun.com'
            key: '0267784512fcbbc8120f66b74b261e9f'

# swagger
swagger2:
    enable: true

# 配置GPU服务器的监听地址和端口
gpu:
    host: gpu.metrics
    port: 9400

# 视频流分类
stream:
    prefix:
        webrtc: 'webrtc://0.0.0.0:8081/main/'
        rtsp: 'rtsp://0.0.0.0:8085/main/'
        rtmp: 'rtmp://0.0.0.0:8083/main/'
```

```
bjy:
    management:
        auth:
            bean:
                shiroConfig: true
                ShiroRealm: true
                RedisUtil: true
                RedisConfig: true
                GlobalExceptionHandler: false
                MybatisPlusSaasConfig: true
                MybatisInterceptor: true
        #  shiroConfig: true
        #  ShiroRealm: true
        #  RedisUtil: true
        #  RedisConfig: true
        #  GlobalExceptionHandler: false
    # 签名密钥串(前后端要一致，正式发布请自行修改)
    signatureSecret: dd05f1c54d63749eda95f9fa6d49v442a
    # 本地：local\Minio：minio\阿里云：alioss
    uploadType: minio
    path :
        #文件上传根目录 设置
        upload: /opt/upFiles
        #webapp文件路径
        webapp: /opt/webapp
    shiro:
        excludeUrls: /test/jeecgDemo/demo3,/test/jeecgDemo/redisDemo/**,/ca

# 共有8个级别，按照从低到高为：ALL < TRACE < DEBUG < INFO < WARN < ERROR < FATA
# intLevel值依次为0,100,200,300,400,500,600,700
# intLevel 值越小，级别越高
Configuration:
    # 日志框架本身的输出日志级别
    status: WARN
    # 自动加载配置文件的间隔时间，不低于5秒
    monitorInterval: 5
    packages: com.smartadmin.config.log.plugin

    Properties: # 定义全局变量
        Property: # 缺省配置（用于开发环境）。其他环境需要在VM参数中指定，如下：
            - name: LOG_PATH    # 输出文件路径
              value: ./logs
            - name: DEFAULT_FILENAME    # 默认文件名
              value: spring
            - name: DEFAULT_PATTERN    # 默认日志输出格式
              value: "%d{HH:mm:ss.SSS} %5p ${sys:PID} --- [%15.15t] %-40.40

    Appenders:
        Console: #控制台输出
            name: CONSOLE
            target: SYSTEM_OUT
            PatternLayout: # 日志消息格式
                pattern: "%clr{%d{HH:mm:ss.SSS}}{faint} %clr{%5p} %clr{${sy


        RollingFile: # 输出到文件，超过128MB归档
            - name: FILE
              fileName: ${LOG_PATH}/app/app.log    #输出文件的地址
              ignoreExceptions: false
```

```
            filePattern: "${LOG_PATH}/app/%d{yyyy-MM-dd}-%i.zip"    # 文件
            ThresholdFilter: # 日志级别过滤器
                level: DEBUG    # 日志级别
                onMatch: ACCEPT    # 高于INFO级别放行
                onMismatch: DENY    # 低于INFO级别拦截
            PatternLayout:
                pattern: "${DEFAULT_PATTERN}"
            Policies: # 日志拆分规则
                SizeBasedTriggeringPolicy: # 日志拆分规则
                    size: "128 MB"
                TimeBasedTriggeringPolicy: # 按天分类
                    modulate: true
                    interval: 1
            DefaultRolloverStrategy: # 单目录下，文件最多20个，超过会删除最早之
                max: 20

          - name: ERROR_FILE
            fileName: ${LOG_PATH}/err/err.log    #输出文件的地址
            ignoreExceptions: false
            filePattern: "${LOG_PATH}/err/%d{yyyy-MM-dd}-%i.zip"    # 文件
              #        ThresholdFilter: # 日志级别过滤器
              #          level: WARN    # 日志级别
              #          onMatch: ACCEPT    # 高于INFO级别放行
              #          onMismatch: DENY    # 低于INFO级别拦截
            PatternLayout:
                pattern: "${DEFAULT_PATTERN}"
            Policies: # 日志拆分规则
                SizeBasedTriggeringPolicy: # 日志拆分规则
                    size: "128 MB"
                TimeBasedTriggeringPolicy: # 按天分类
                    modulate: true
                    interval: 1
            DefaultRolloverStrategy: # 单目录下，文件最多20个，超过会删除最早之
                max: 20

    # 自定义Appender
    #    OperatorLogEsAppender:
    #      - name: OPERATOR_LOG_FILE
    #        ignoreExceptions: false
    #        PatternLayout:
    #          pattern: "%m%n%xwEx"

    Loggers:
        Root:
            level: INFO #root的级别为info，如果为debug的话，输出的内容太多
            AppenderRef:
                - ref: CONSOLE

        Logger:
            # 操作日志
            #      - name: com.smartadmin.config.log.OperatorLogAspect
            #        level: INFO
            #        AppenderRef:
            #          - ref: OPERATOR_LOG_FILE
            #      - name: com.smartadmin.config.exception.CatchException
            #        level: INFO
            #        AppenderRef:
            #          - ref: OPERATOR_LOG_FILE
            #      # 登录日志
            #      - name: com.smartadmin.controller.common.LoginController
```

```
            #        level: TRACE
            #        AppenderRef:
            #          - ref: LOGIN_LOG_FILE
            - name: com.mysql
              level: INFO
            - name: bboss
              level: INFO
            - name: org.springframework
              level: WARN
            - name: org.spring.springboot.dao
              level: DEBUG
            - name: org.springframework.web.servlet.mvc.method.annotation.E
              level: ERROR
            - name: com.smartadmin
              level: DEBUG
              AppenderRef:
                  - ref: FILE
```

```
AlgoVersionImport
```

盒子错误解决方案

盒子错误解决方案

> 访问、登录报 `404`

错误原因**1**：可能由于机器重启，nginx服务还没有启动完成

解决方案：等待片刻再进行访问重试即可。

错误原因**2**：如果方案一还是不能进行访问，那就是有可能是服务启动过程中出现问题。

解决方案： `sudo docker ps` 查看服务是否都已经启动，含

`mysql` 、 `nginx` 、 `redis` 、 `boliving` 、 `ai_platform_test`

如有未启动，可执行 `sudo sh /home/nvidia/step.sh` 进行重启

> 算法管理算法列表图片展示异常

原因**1**： 登录盒子数据库，查看 `algo` 找到对应的算法，查看 `icon_url` ，盒子数据为 `http://116.198.37.170:80/upload/comm/1647496478066_Baidu_217__2022-03-04_0001.jpeg` 格式访问不到图片的话，那么盒子无法进行连接外网，

解决方案： 删除 `icon_url` 中的前缀IP地址,修改为 `/upload/comm/1647496478066_Baidu_217__2022-03-04_0001.jpeg` 格式，并经 `1647496478066_Baidu_217__2022-03-04_0001.jpeg` 图片迁移到服务器 `/data2/ai-algo-aat/upload/comm` 目录下。

原因**2**：若当前 `url_url` 已经是不含IP的格式，只需检查服务器对应的目录下是否有该张图片资源。

解决方案：查看 `/data2/ai-algo-aat/upload/comm` 目录下是否有对应的图片资源，没有的话进行迁移即可。

> 报错： `"nested exception is org.apache.ibatis.builder.BuilderException: The expression 'ids' evaluated to a null value."`

原因：由于系统长时间为进行页面操作导致平台 `token` 过期，使用该凭证进行接口请求的时候会报这个错误。

解决方案：重新刷新页面登录即可。（新版本中已经修复该报错）

> 报错： `"No such file or directory:'/data/cache/328/models/weights/best.pt'`

原因：服务器中不含有模型文件或者是含有模型文件，路径地址与数据库中对应的地址不一致。

解决方案：查看对应的服务器目录下是否韩友谊对应的模型文件：`/data2/ai-algo-aat/cache/328/models/weights` 目录下是否含有 `best.pt` 、 `last.pt` 文件，

如果没有该文件，联系算法相关人员进行算法模型的迁移，

如果有这两个文件，检测目录是否为 `/data2/ai-algo-aat/cache/328/models/weights` ，如果不一致，讲文件迁移到该目录下即可。

北京盒子hub部署

北京盒子hub部署

Nginx

```
docker pull harbor-albj.baijiayun.com/ai_architecture/nginx:v1
docker run -m 200M  --restart=always --name=nginx --net=host -d -v /home/ng
```

Mysql

```
docker pull harbor-albj.baijiayun.com/ai_architecture/mysql:v1
docker run  -m 200M  --restart=always --name mysql --net=host -e MYSQL_ROOT
```

Redis

```
docker pull harbor-albj.baijiayun.com/ai_architecture/redis:v1
docker run --name redis  -m 200M  --restart=always --net=host -d harbor-alb
```

Boliving

```
docker pull harbor-albj.baijiayun.com/ai_architecture/boliving-arm:v1
docker run -itd --name boliving harbor-albj.baijiayun.com/ai_architecture/b

#copy一份配置文件到本地
mkdir -p /home/nvidia/docker_compose/boliving
docker cp boliving:/boliving/bin/ /home/nvidia/docker_compose/boliving/bin
#停止镜像
docker stop boliving
docker rm boliving
#修改配置信息
cd  /home/nvidia/docker_compose/boliving/bin
vim config.toml

#1 [Nacos]
#2 Enable=true
#3 ServerIP="127.0.0.1"
#4 ServerPort=8848
...
#26 [RTMP]
#27 ListenAddr = ":8083"
#28 [GateWay]
#29 ListenAddr = ":8081"
新加
TraceApi = "/api/boom/getBySerial"
RTCSAddr = "http://127.0.0.1:6060"
...
#46 RTSP]
```

```
#47 Timeout = 20
#48 AutoPull = false
#49 RemoteAddr = "rtsp://localhost/test"
#50 ListenAddr = ":8085"

#启动镜像，进行目录挂在
docker run -itd --net=host --name boliving  -v /home/nvidia/docker_compose/
#进入映射目录，修改配置文件.
docker exec -it boliving /bin/bash
cd /boliving/bin
./boliving -c ./config.toml&
```

JDK

```
sudo apt-get install openjdk-8-jdk
java -version
#上传平台的jar包
nohup java -jar start.jar  &
#便可登录平台
```

自动启动脚本 **step.sh**

```
#!/bin/bash
# 启动 算法 服务
docker restart ai_platform_test
# 启动 redis 服务
docker restart redis
# 启动 nginx 服务
docker restart nginx
# 启动 mysql 服务
docker restart mysql
# 启动 java 服务
ps -ef | grep "java"| grep -v grep | awk '{print $2}'| xargs kill -9

cd /home/nvidia/docker_compose

nohup java -jar start.jar &
# 启动 boliving 服务
docker restart boliving
docker exec -d boliving bash -c 'cd /boliving/bin/ && ./boliving -c ./confi
```

静态资源迁移

```
mkdir -p /data2/ai-algo-aat/upload/comm
chown nvidia /data2/ai-algo-aat/upload -R
#迁移文件 1647496478066_Baidu_217__2022-03-04_0001.jpeg
mkdir -p /home/nvidia/docker_compose/license/
chown nvidia /home/nvidia/ -R
#迁移license文件
```

docker-compose down docker-compose up –build -d

盒子部署算法

> Mysql

```
docker pull mysql:8.0-oracle
docker run -d --name -m 200M mysql -e MYSQL_ROOT_PASSWORD=123456 -p 3306:33
#然后链接mysql数据库进行数据导入
```

> Redis

```
docker run --name redis -m 200M -v /home/nvidia/data/redis/:/etc -p 6379:63

docker run --name redis -p 6379:6379 -d redis:6.0.16
```

> Nginx

```
docker run --name nginx -m 200M -d -p 80:80 -v /home/nginx/content:/usr/sha
```

> Nginx配置信息：
>
> 注意修改： proxy_pass http://宿主机IP:6060;

```
cd /home/nginx/conf.d

server {
    listen       80;
    listen  [::]:80;
    server_name  localhost;

    #charset koi8-r;
    access_log  /var/log/nginx/host.access.log  main;

    location / {
        root   /usr/share/nginx/html/dist;
        index  index.html index.htm;
    }
        location /dist/ {
         root /usr/share/nginx/html/;
            index  index.html index.htm;
            proxy_redirect off;
        }

        location /img/ {
            root /data/ai-algo-aat/upload/;
            autoindex on;
        }
        location /upload/ {
            root /nas/ai-algo-aat/;
            autoindex on;
        }
        location /collect/ {
            root /nas/ai-algo-aat/;
            autoindex on;
        }


        location /comm/ {
            root /mnt/ip230_data2/ai-algo-aat/upload/;
```

```
            autoindex on;
        }

        location /api/ {
            if ($request_method = 'OPTIONS') {
                add_header 'Access-Control-Allow-Origin' '*' always;
                add_header 'Access-Control-Allow-Credentials' 'true';
                add_header 'Access-Control-Allow-Methods' 'GET, POST, PATCH
                add_header 'Access-Control-Allow-Headers' 'DNT,X-Mx-ReqToke
                add_header 'Access-Control-Max-Age' 1728000;
                #add_header 'Content-Type' 'text/plain charset=UTF-8';
                add_header 'Content-Length' 0;
                return 200;
            }

            add_header Access-Control-Allow-Headers 'Content-Type,Token,Aut
            add_header Access-Control-Allow-Origin *;
            add_header Access-Control-Allow-Credentials true;

            root   html;
            index  index.html index.htm;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header Host $host:$server_port;
            proxy_pass http://10.8.26.61:6060;
            proxy_redirect off;
            client_max_body_size 1024M;
        }

    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }

    # proxy the PHP scripts to Apache listening on 127.0.0.1:80
    #
    #location ~ \.php$ {
    #    proxy_pass   http://127.0.0.1;
    #}

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    #location ~ \.php$ {
    #    root           html;
    #    fastcgi_pass   127.0.0.1:9000;
    #    fastcgi_index  index.php;
    #    fastcgi_param  SCRIPT_FILENAME  /scripts$fastcgi_script_name;
    #    include        fastcgi_params;
    #}

    # deny access to .htaccess files, if Apache's document root
    # concurs with nginx's one
    #
    #location ~ /\.ht {
    #    deny  all;
    #}
```

```
}
```

> 前端项目：
>
> 直接dist目录放到挂在目录中： `/home/nginx/content` 。访问IP即可访问！
>
> Boliving

```
#先将tar包导入到服务器，并生成镜像
docker load --input boliving-arm:bing.1.0.0.03121.tar

#运行镜像
docker run -itd --name boliving registry.cn-beijing.aliyuncs.com/webrtc-boo
#copy一份配置文件到本地
mkdir /home/nvidia/boliving
docker cp boliving:/boliving/bin/ /home/nvidia/boliving/bin

#停止镜像
docker stop [boliving容器id]
docker rm [boliving容器id]

#启动镜像，进行目录挂在
docker run --net=host -v /home/nvidia/boliving/bin:/boliving/bin registry.c

docker run -itd --net=host -v /home/nvidia/boliving/bin:/boliving/bin regis
#进入映射目录，修改配置文件.
cd  /home/nvidia/docker_compose/boliving/bin
vim config.toml
# ListenAddr 修改为 ":8083"  ListenAddr 修改为 ":8085"  RTCSAddr 修改为 "http:
#26 [RTMP]
#27 ListenAddr = ":8083"
#28 [GateWay]
#29 ListenAddr = ":8081"
#新加
#30 TraceApi = "/api/boom/getBySerial"
#31 RTCSAddr = "http://127.0.0.1:6060"
...
#46 RTSP]
#47 Timeout = 20
#48 AutoPull = false
#49 RemoteAddr = "rtsp://localhost/test"
#50 ListenAddr = ":8085"

#重启容器
docker restart [boliving容器id]
#进入容器
docker attach [boliving容器id]



cd /boliving/bin
./boliving -c ./config.toml&


#访问ip:8081,能够访问的话，证明安装成功
```

> JDK

```
sudo apt-get install openjdk-8-jdk
java -version
#上传平台的jar包
nohup java -jar start.jar > runtimelogs.log 2>&1 &
#便可登录平台
```

> 推流

```
ffmpeg -i 'rtsp://admin:admin123@172.31.100.59:554/cam/realmonitor?channel=
```

> ai平台添加 `rtsp://172.31.2.124:8085/ai/rtsp` 即可在线预览与视频截帧分析

```
docker run --name mysql --net=host -e MYSQL_ROOT_PASSWORD=123456 -v /home/r

docker run --name redis --net=host -d redis:6.0.16 --requirepass "654321"

docker run --name=nginx --net=host -d -v /home/nginx/content:/usr/share/ngi
```

```
/pc/algo/version/trainCallback    算法训练的结果回调通知
/pc/algo/version/modelCallback     模型的结果回调通知
/pc/algoApp/algoImgResultCallback    算法应用-图片应用结果回调
/collectionTask/collectionCallback       采集回调通知
/deployAuthorizeUsers/callback 部署包结果回调接口
/onlineVideoStream/onOrOffCallBack                  视频流开启成功/失败后的回调
/onlineVideoStream/modelCallback                    分析回调
```

通过培训，我们深刻体会到，如果想要成为一个优秀的员工，就要完全的表现出自己的优势，自己的个人价值，包括提升自己的能力、性格、态度、知识面等。

我们员工的知识面以及个人价值，能够体现出我们在用户心目中的定位，整体代表的公司在用户心目中的价值。更好的价值体现能够为公司提供一个良好的信誉与荣耀。

通过了藏药只是体系的讲解与培训，让我们对藏药的起源与发展有了很大的了解，作为即将为产品服务的我们来说，提高对药物信息的掌握，有助于我们更好的去引导用户，更有利的宣传公司产品的价值。也能够通过我们的专业知识，给用户带来更好的信任。