dsp-root

每个项目的公用模块，规定每个项目的maven依赖所能引用的范围，每个jar包的版本

会读取C:\Users\zhangms\.m2\repository\com\chinanetcenter\dsp\dsp-depends\1.1.3

这个目录下的dsp-depends-1.1.3.pom下的配置信息如果项目有需要引用额外的jar且这个jar不在dsp-depends配置中，则需要想华明申请看是否有必要

dsp-board

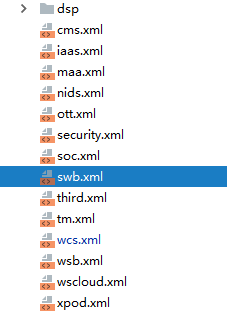
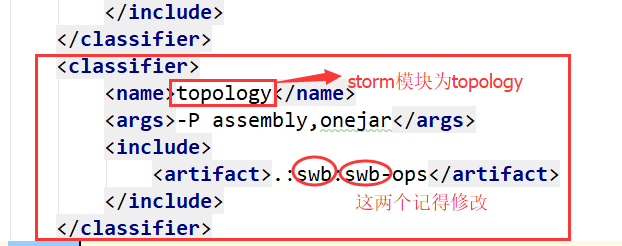
各个项目的模块信息，比如wcs这个项目包含wcs-camps,wcs-hadoop,wcs-ingest,wcs-storm这几个模块，其中wcs-common和wcs-parser这两个属于依赖模块，其他都属于业务模块

依赖模块在打包时需要用mvn clean install -P -repo

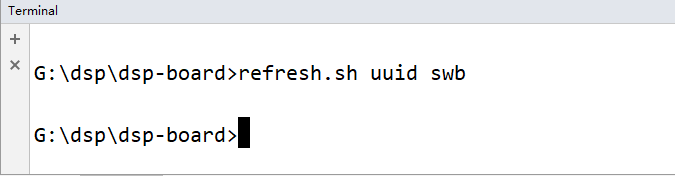
业务模块打包mvn clean install 就行了

项目启动时首先会根据项目底下的dsp-board.version来读取相应的maven的module版本，如果没有找到则会去读取dsp-board这个项目底下的current文件来确定要读取的module版本（这个版本配置在dsp-board的project底下的stable中）

工程新增模块需要在对应的xml文件中配置

以swb工程为例，新增一个storm模块

然后git提交，提交完后记得执行这个命令



修改后直接提交到远程仓库的master上 提交信息格式为:feat(ci/project/uuid):xxx

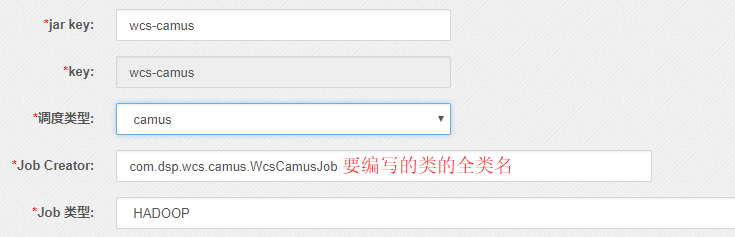
dsp-wcs

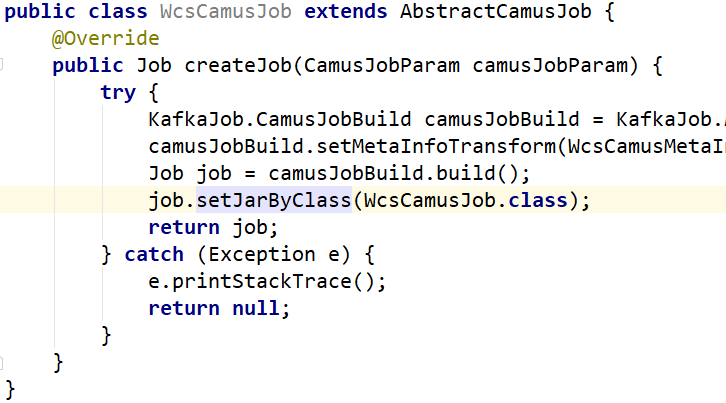
wcs-camus 日志相关

测试环境旧版portal

作业管理

Jar包管理

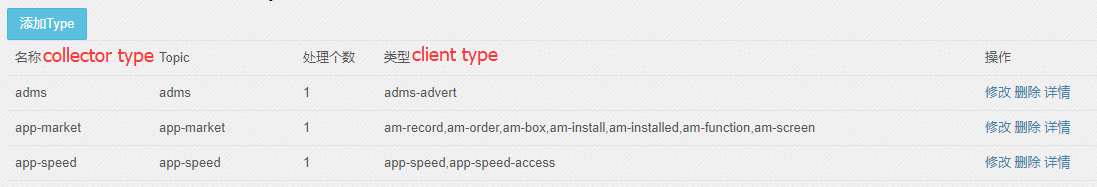




运维管理

Collector

Type列表



FieldSchema：

optional 指这个字段的值可能为空

比如一个日志，新的版本新增了两个字段，这时候就需要兼容新老版本的日志，所以最后两个字段需要使用optional的属性来兼容新老版本的日志解析

Index 是否是索引的意思，不大有用

新增字段要求都是往最后的地方加，如果中间的某部分的字段变了，比如字段名称或者字段类型，这时候就需要新建一个schema，把版本加上去，否则只是往后面加字段的话，不需要新建一个schema文件，版本不需要变化

数据不会直接写入到hbase或者mongo，有专门的写入程序去写入,massinsertmanager只是把数据放到每台supervisor的磁盘中，每台supervisor有个storm-daemon程序会把这些文件批量写入到数据库里面

FieldType里面的类型，有个record，默认是record直接取日志解析出来的值，设置成attach的就是从RecordAttach读值，相当于是在做数据中转，额外添加的属性，可以这么理解，还有一个是meta的，是从MetaInfo取值

Schema定义的是日志的结构，日志不包括meta的信息，不要加进去

Token：

Token(Matcher matcher)

Token create(String beginDelim, String endDelim)

都要实现match(String msg, MessageParser.Index from)接口，这个接口才是具体的解析工作

with 把上一串处理的结构在transform中继续做处理（string s 即是上一串return的结果）

StormTopology：

在build的时候就已经把spout设置为KafkaSpout了，只需要我们传入kafka的topic即可，同时也设置了ParseDispatchBolt，用于解析数据格式的bolt以及AsyncPersistBolt用于持久化的bolt

allgrouping, 一般用于全局的数据同步和共享才需要, 比如全局的配置更新等

cluster：

Asset asset = WcsZkconfig.getAsset(metaInfo);

Long clusterId = asset.getClusterId();

recordAttach.attach(CLUSTER, String.valueOf(clusterId));存进去的是clusterId

RedisKeyBytes：

to(RedisKey redisKey)方法

byte[3+][] bytes 前三个存prefix、time、parts经过MsgPackUtil处理后的值

接着再存parts中一个一个经过MsgPackUtil处理后的值

同时原来rediskey的3个属性值(prefix,time,parts)也有经过处理再赋值过去

IndicatorResource.getIndicator(RGW\_STATUS\_INDICATOR)中的名字是在哪里设置进去的?

indicatorBuilder.setAsyncProcessBolt貌似是在这里设置processbolt的时候设置的?

MultiBulkReply multiBulkReply = dataFuture.get();

Reply[] replies = multiBulkReply.data();

1. redis 127.0.0.1:6379> HGETALL myhash
2. 1) "field1"
3. 2) "Hello"
4. 3) "field2"
5. 4) "World"

replies按顺序存放k，v，k，v，k，v…………

//设置创建记录的类

StormConfig.put(RecordCreator.RECORD\_CREATOR\_CLASS, WcsRecordCreator.class.getName());

//解析日志用的线程数

StormConfig.put(AsyncParseAndDispatchBolt.PARSE\_DISPATCH\_THREAD, 5);

//累计多大的时候数据才写入文件

StormConfig.put(AsyncPersistBoltExecutor.PERSIST\_FLUSH\_SIZE, 5 \* FileUtils.ONE\_MB);

//累计多少条之后再写入文件

StormConfig.put(AsyncPersistBoltExecutor.PERSIST\_FLUSH\_LENGTH, 10000);

//一次写入多少条数据到文件

StormConfig.put(MassInsertManager.Config.MASSINSERT\_BATCH\_SIZE, 100000);

//数据刷入的间隔

StormConfig.put(MassInsertManager.Config.MASSINSERT\_FLUSH\_PERIOD, 50000L);

//gather里面处理的队列大小，超过这个队列大小就会停止从kafka获取数据，等到减少之后再启动

StormConfig.put(StormConfigKeys.GATHER\_QUEUE\_SIZE\_LIMIT, 1000000);

StormConfig.put(StormConfigKeys.MASSINSERT\_STACKING\_COUNT\_LIMIT, 100);

//设置topology的IndicatorBuilder

StormTopology.IndicatorBuilder indicatorBuilder = topology.setIndicatorBuilder(Lists.newArrayList(

new StormRedisConfig.StormRedis(NAME)));

//将key转成字节再存入redis中

RedisKeyBytes keyBytes = RedisKeyBytes.to(redisKey);

//获取分区信息

int partition = IndicatorResource.getPartition(keyBytes);

//???

int indicatorId = IndicatorResource.getIndicatorId(IndicatorResource.getIndicator(indicator));

//在redis中做累加操作，并把redis中执行的命令也存起来？

output.addCommand(partition, RedisCommands.hincrby(keyBytes, field.getBytes(), count));

//一个indicatorId对应多条记录?

output.addKey(indicatorId, new HashKey(partition, keyBytes, null));

HBase表创建：

rowKey：

主键id，时间(Long.MAX\_VALUE - TIME)，可选项(时间后面是可选的，按照这个顺序来设置rowKey)

字段名尽量简短，后面在接口文档里补充相关定义

**topology代码流程：**

XXXSchema：

CLIENT\_TYPE

VERSION(记得重写的方法返回值要写VERSION)

XXXSchemaFactory：

recordSchemaMap.put() 不要忘记

XXXTokenFactory：

GROUPS.put() 不要忘记

XXXRecordCreator：

getRecordSchema() 不要忘记

**查询条件就是rowKey**

**查询结果就是那些统计的数据(可能有多个统计结果)**

git：

git要先clone

要进行数据保存的bolt需要实现PersistEmitter接口

新增模块swb-ops的topology部署流程：

如果是新增的模块参考前面新增模块的操作；

# Redis配置：







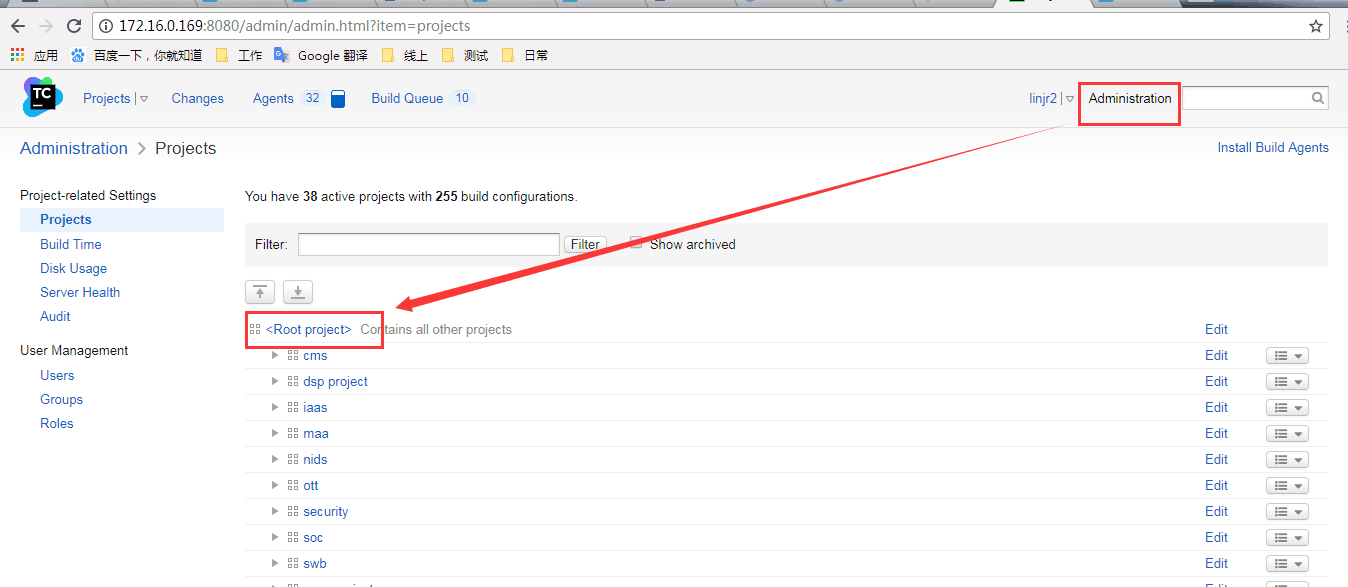


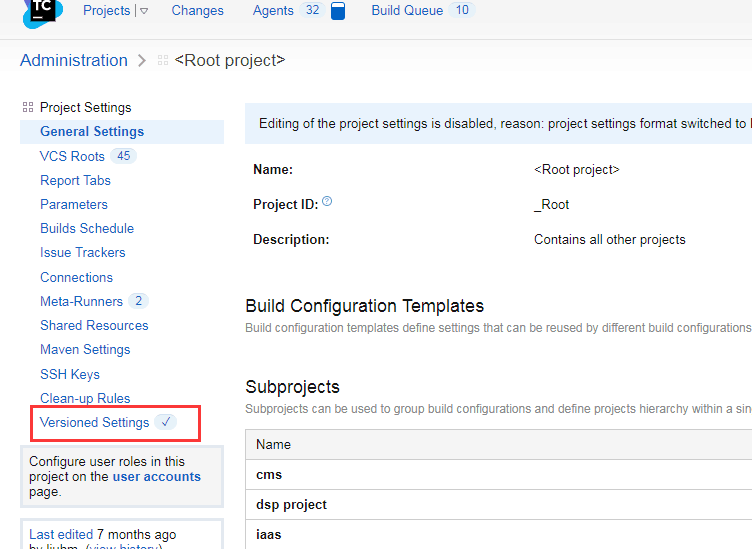
redis clean job 配置：

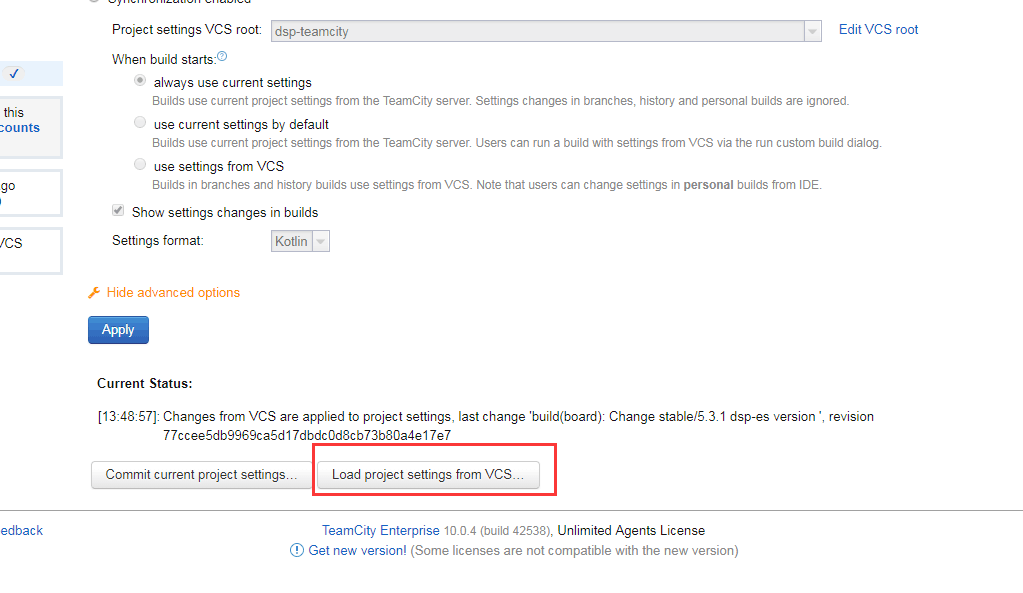


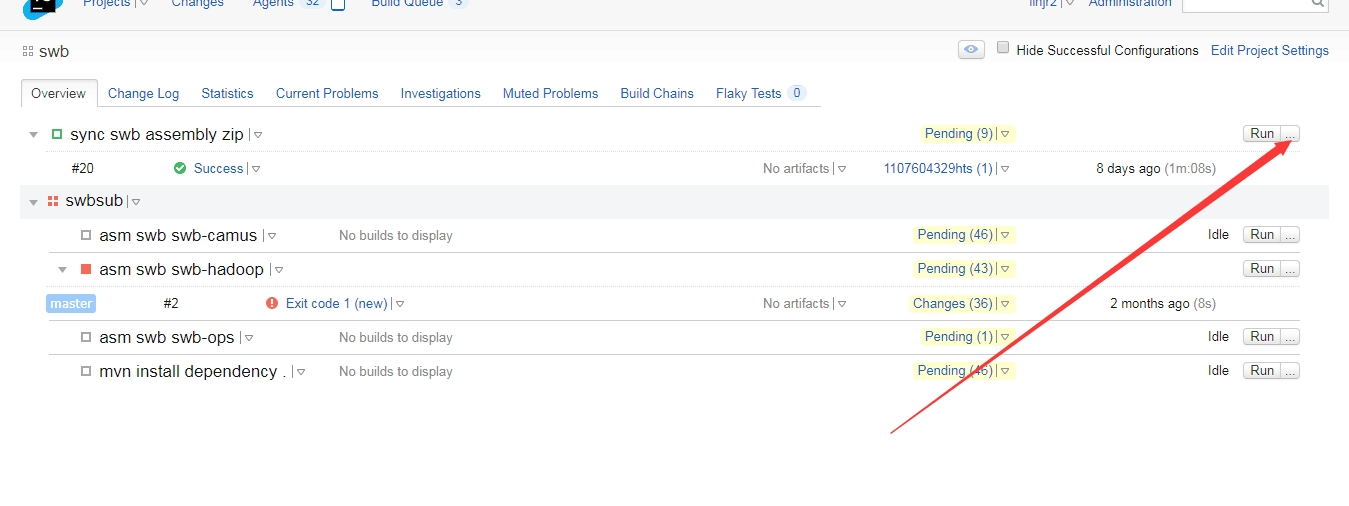
topology启动起来后再发布

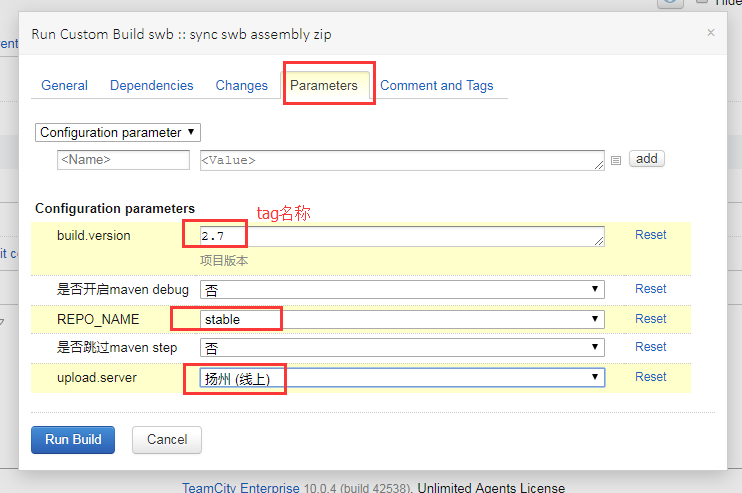
**teamcity打包:**



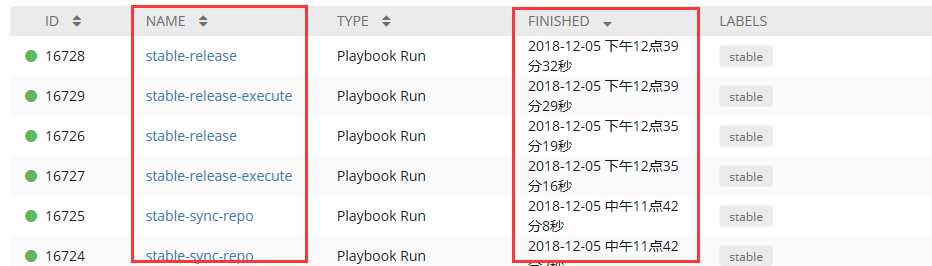








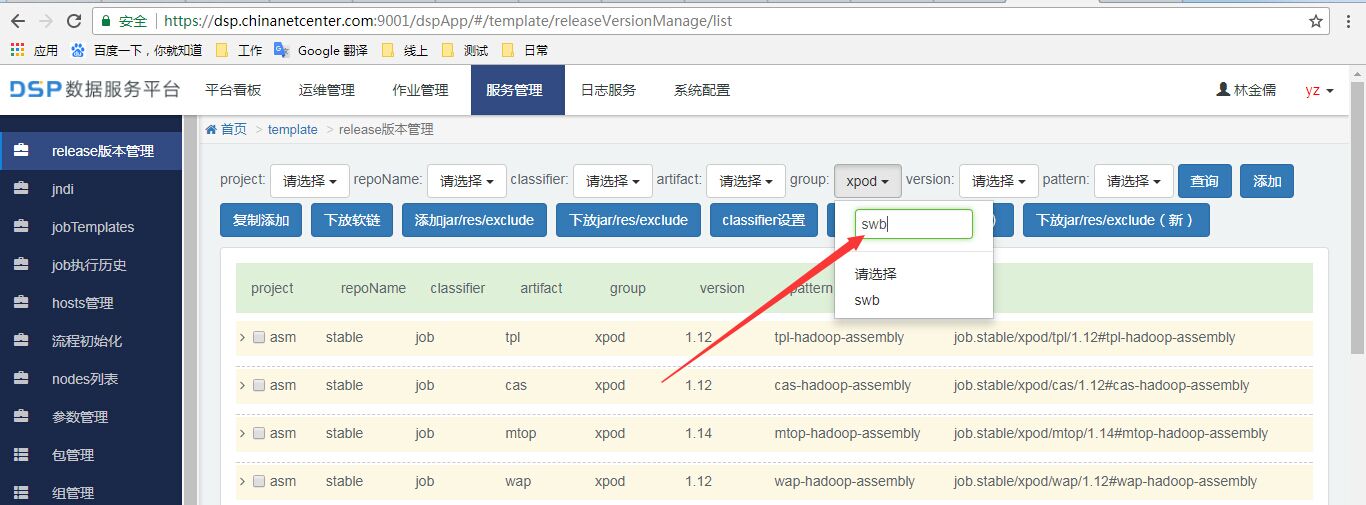
同步有没有成功可以上ansible看



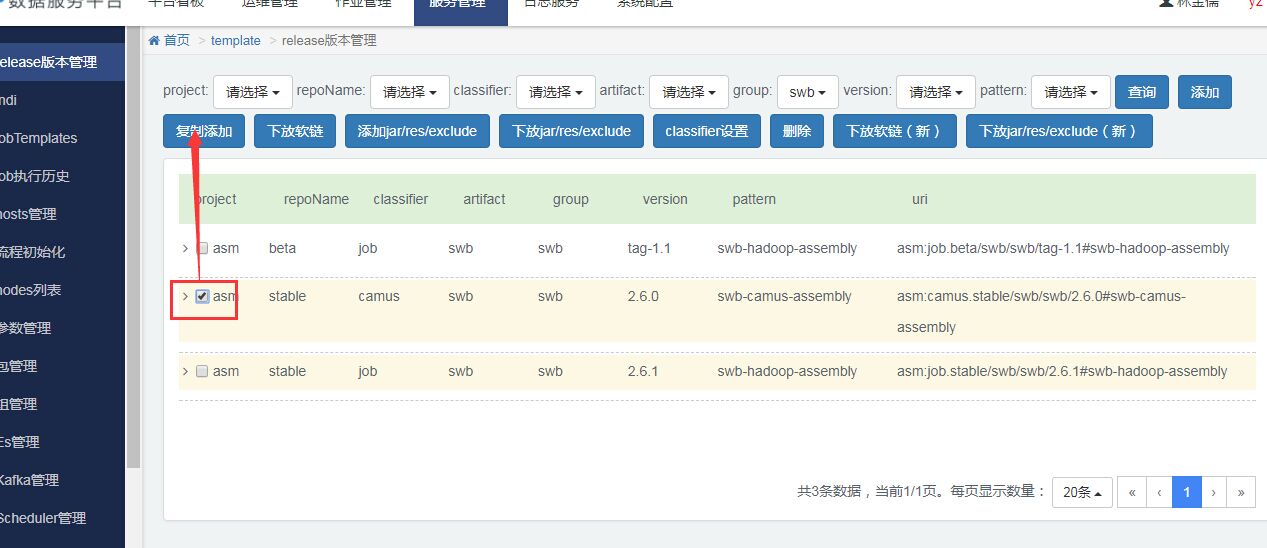
可通过名称、时间来判断具体是哪一个

**升级步骤：**

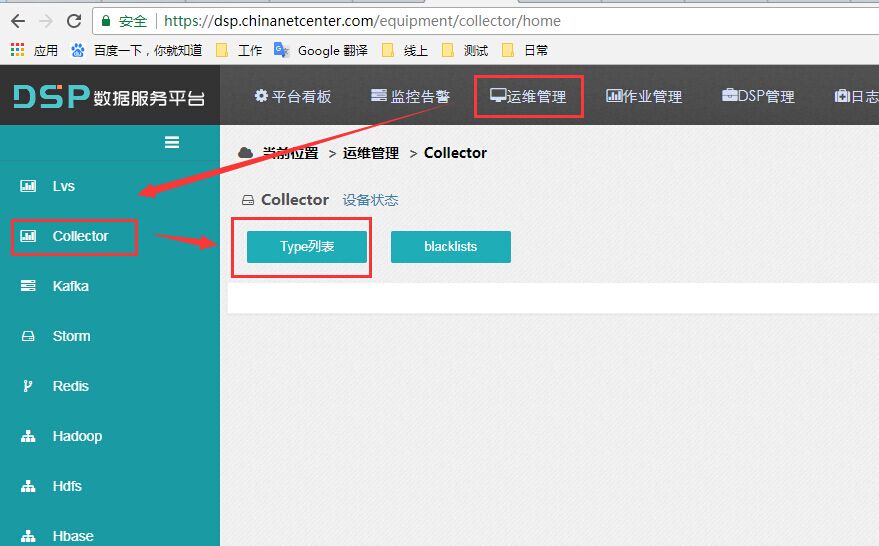
新版正式:



复制camus的版本?



旧版正式：





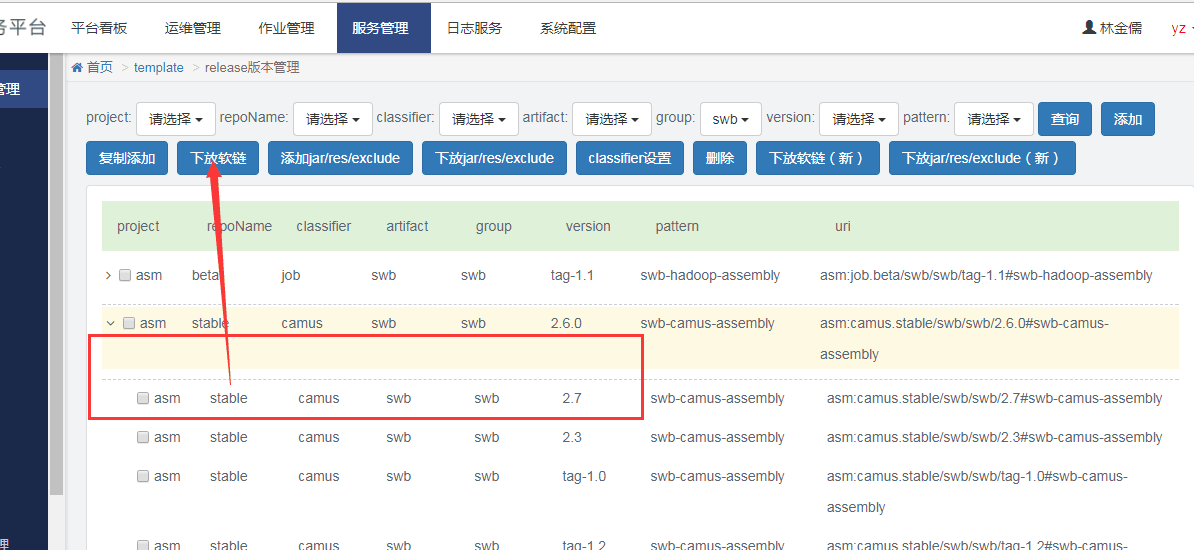
名称和topic要一样

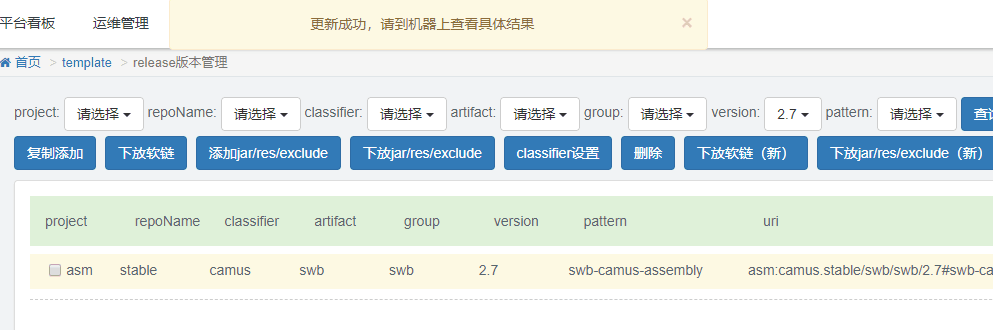


这里是类型填的是client\_type



新版正式：

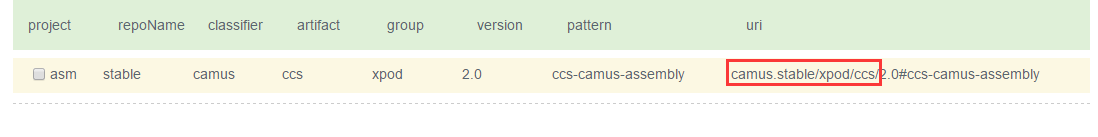


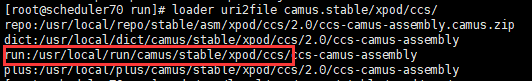


到scheduler70机器上查看是否成功：

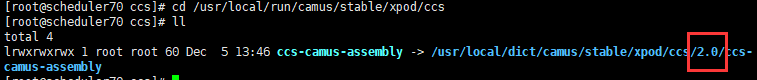
在跳板机ssh.snstack.cn上ssh [root@192.168.2.70](mailto:root@192.168.2.70)

loader uri2file camus.stable/xpod/ccs





cd到run的这个路径下



查看是否已更新到指定的版本即可

**打包到测试环境进行测试：**

本地先切到项目目录比如swb目录

C:\Users\zhangms\Documents\Tencent Files\3004174311\Image\C2C\S@8IWVX3[C{YI9O[)KCB3~2.png

1.安装依赖的项目:swb-parser,swb-common

mvn clean install **-P -repo**

2.然后在swb目录底下执行mvn\_publish.sh -d swb-ops -m "mvn clean package -P assembly,onejar" -v (版本，如20181128) -u (yz(线上)/sz(测试)) -s

例子：mvn\_publish.sh -d swb-ops -m "mvn clean package -P assembly,onejar" -v 20181128 -u sz -s

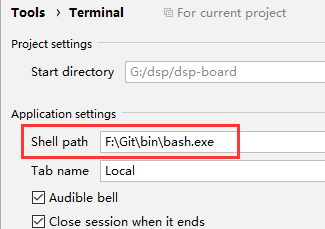
mvn\_publish.sh -d iaas-camus -m "mvn clean package -P assembly,onejar" -v 20181128 -u sz -s

注：topology只能通过命令行部署到测试环境，即-u只能设置成"sz"

如果是Job的话根据实际情况来决定

**这一步需要安装tower-cli 3.1.7版本的**

**在idea的Terminal设置中**



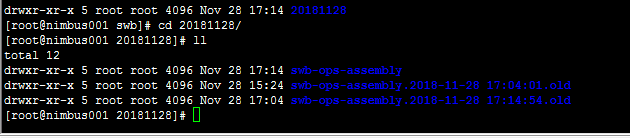
3.打包完成后

1.跳板机连接到测试环境的monitor

2. ssh nimbus001.sz

◆ 自动部署的路径：

/usr/local/dict/topology/beta/swb/swb



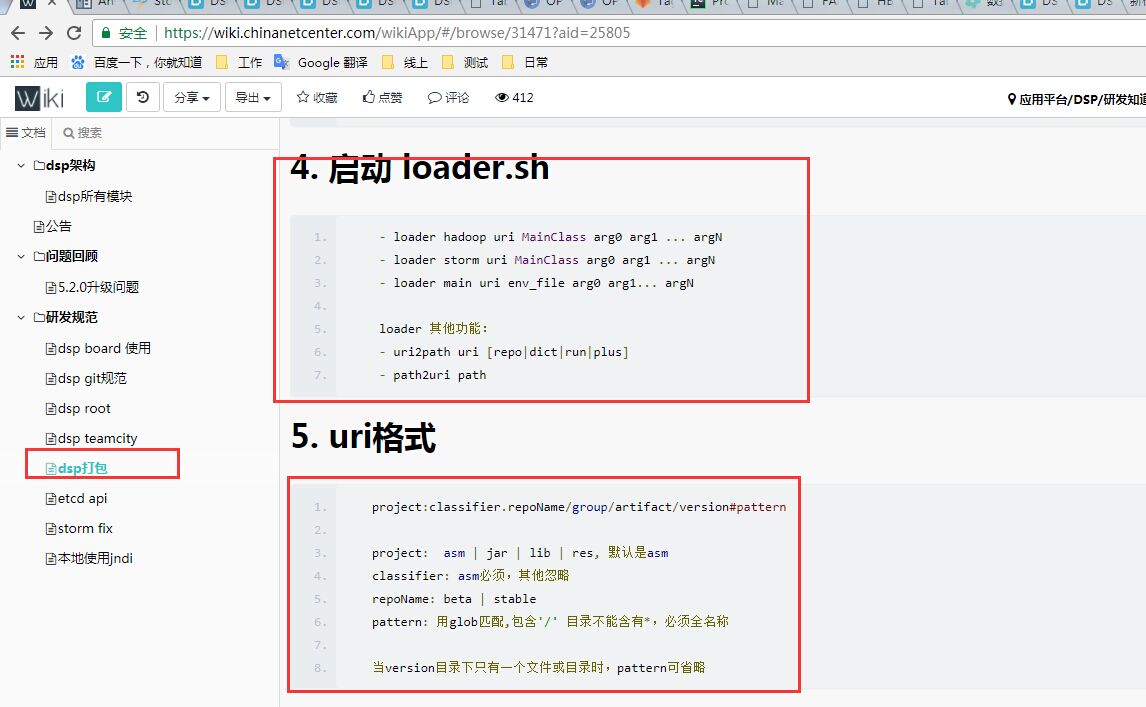
多次打包相同版本号会在这边体现

* 这个路径，底下会有classpath文件夹，因为页面没有配置，所以不会生成这个路径

/usr/local/plus/topology/beta/swb/swb/(版本号)/swb-ops-assembly/

* 执行

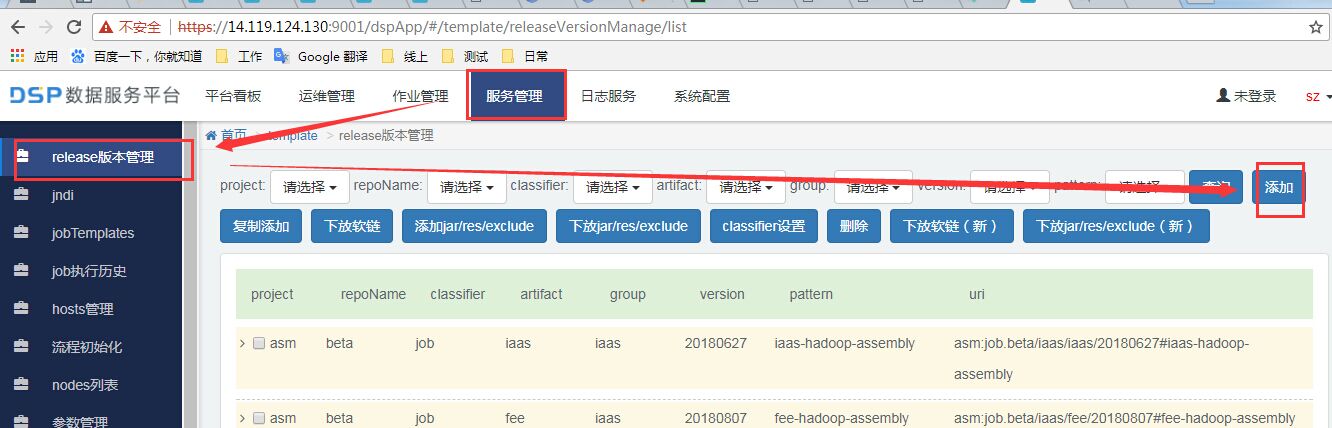
loader path2uri /usr/local/dict/topology/beta/swb/swb/20181128/swb-ops-assembly



C:\Users\zhangms\Documents\Tencent Files\3004174311\Image\C2C\]PXBSFRM`6NS0PI9Y(Y1(3P.png

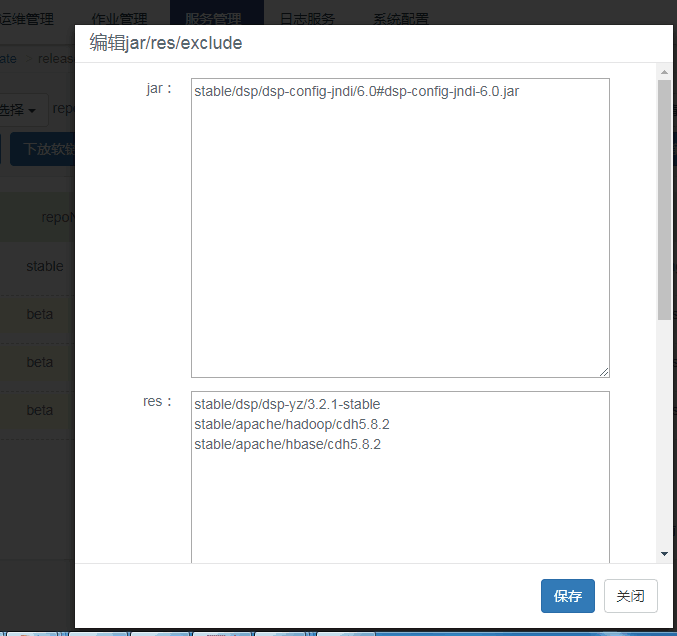
把生成的这串拷贝下来

* 新版测试



* 同一个页面group搜索swb，即你新增加的



里面有三栏，你再打开一个页面也是切到这个页面，group搜索wcs，找到topology类型

把这三栏复制到swb对应的那三栏并保存

* 完成后，点击[下放软链]下放swb的topology的软链

下放完后就能看到刚才说的classpath文件夹了

当前正在执行的topology的版本，这里面放着的是一个软链，就是你刚才页面配置的软链：

/usr/local/run/topology/beta/swb/swb/

camus日志：

有新日志进来的时候，或者日志格式有变化的时候才需要部署camus

ssh scheduler002.sh

vim /etc/hadoop-scheduler/ jar2uri.properties



1. 添加新配置
2. 表名，topic，schema为版本号，maxMapNum、matainfos不用变

# storm的启动：

loader storm uri(就是刚才loader path2uri 生成的路径) mainclass(就是topology的完成包名+类名) 参数（可选）

比如swb: loader storm asm:topology.beta/swb/swb/20181128#swb-ops-assembly com.dsp.swb.ops.topology.SinglePingTopology

在nimbus机器上启动

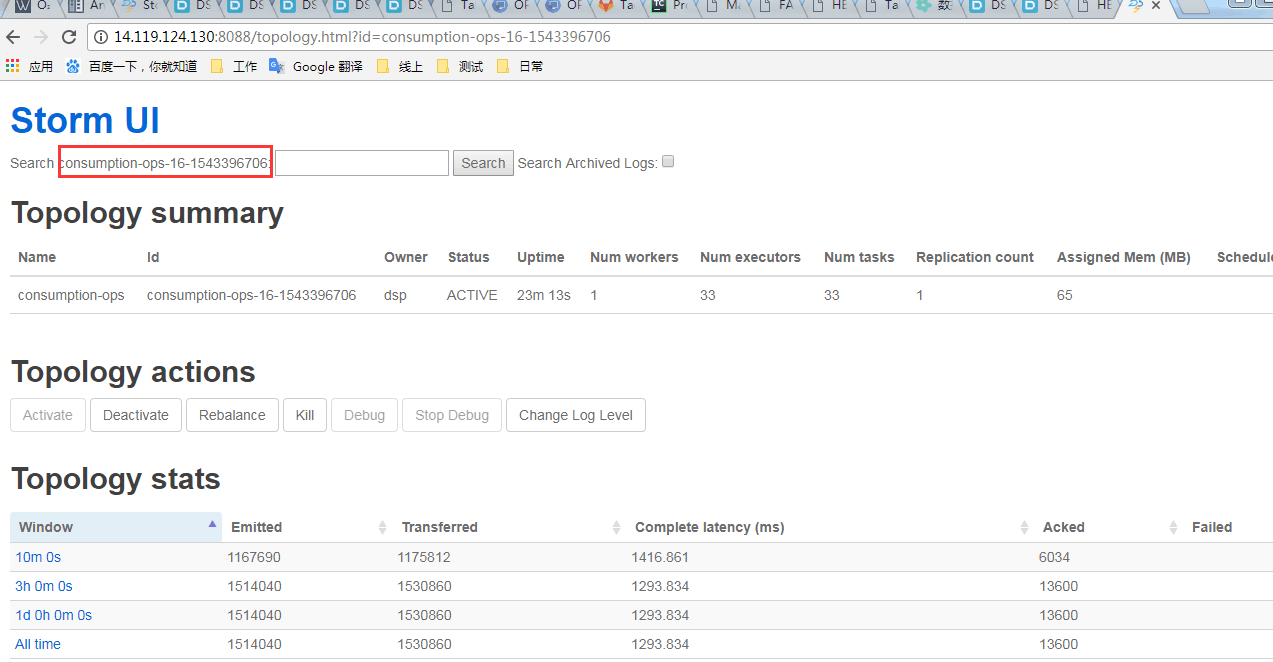
**storm 查看:**

storm list

**storm停止：**

storm kill 名称(上面查看时的名称)

**storm日志的查看：**



拿着这一串(topology\_id)到具体的机器上看日志

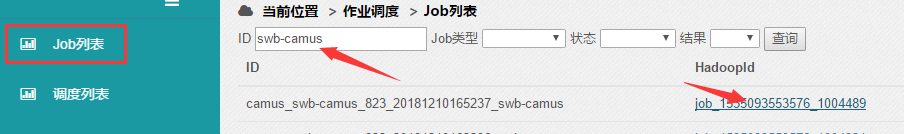
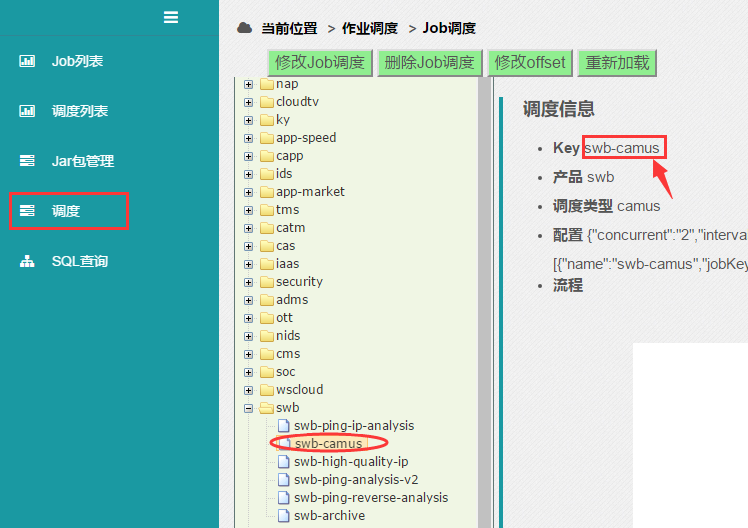
日志路径：

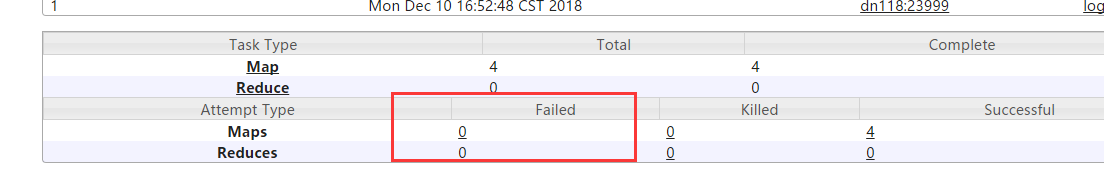
/var/log/storm/(topology\_id+端口和其他信息).log

规范：

表名要用下划线

# camus日志





在机器上查看：

hadoop fs -ls /hidalog/default/data/swb(topic)/2018-12-10(日期)/\* | grep swb-ops-ip\_v1(schema版本)

camus、receptor、

# 告警

**测试：**

ssh alert001.sz

/usr/local/alert\_2/data/monitor/iaas/

日志：/var/log/alert\_2/alerter.log

**正式：**

ssh alert17

/usr/local/alert\_2/data/monitor/

# collector

查看日志有没有推送

ssh collector001.sz

/var/log/collector/http.log

正式

collector312

/var/log/receptor/http.log

# storm 升级

首先要先看看旧版本的/usr/local/dict/topology/stable/wsb/wsb/1.4.0/wsb-storm-assembly/main路径下的main.jar有没有额外的资源文件

如果有就在项目的顶层pom文件里加上

<build>

<resources>

<resource>

<directory>src/main/resources</directory>

</resource>

</resources>

</build>

然后通过teamcity重新打包

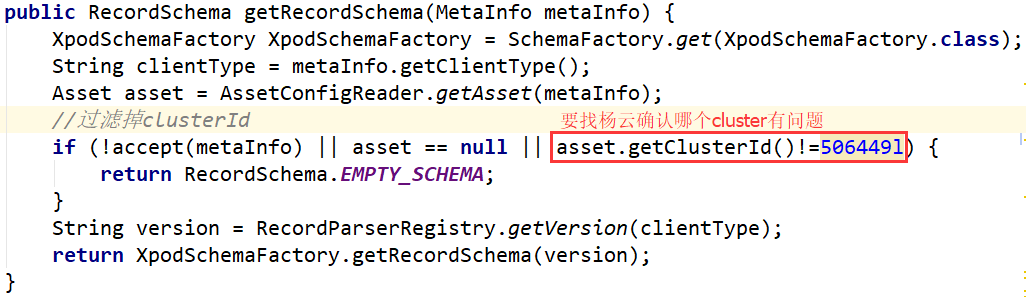
重启storm

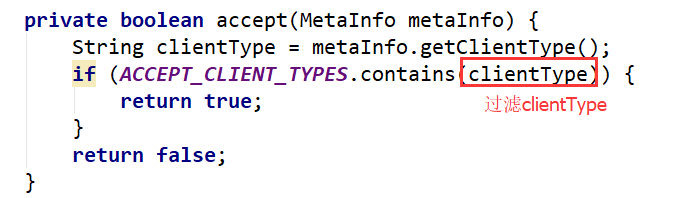
exeLoader topologyName restart

启动后stormUI页面查找Executor数>=worker数的bolt，查看其日志有没有问题，然后再看hubspout日志并搜索”cur “关键字，这个关键字是看storm从kafka读取日志的时间，如果小于当前时间5分钟左右那就正常，否则就是有延迟需要排查（如果之前停过storm的话那么有延迟是正常的，等一段时间后再看是不是恢复正常）

# storm fix

找到要修复的topology，找到对应的XXXRecordCreator，修改getRecordSchema()和accept（）方法，accept可能不需要，要问清楚

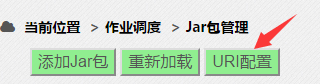




# job配置

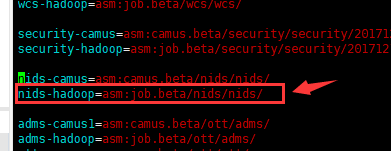
/usr/local/repo，和dict,run,plus四个目录，然后run的目录是dict的软链。这里说明一下run目录的作用，run目录的唯一一个作用就是在需要配置uri的地方可以缺省版本号，不用每次版本升级都修改配置(配置包括storm的启动脚本exeLoader，调度配置的uri)，所以要在这两个地方配置缺省版本的uri，必须要有stable包，并且软链下放成功。

正式：loader path2uri 取run目录 不用带版本号



测试：loader path2uri 取dict目录 要带版本号 portal已经失效

测试环境ssh scheduler002.sz vim /etc/hadoop-scheduler/jar2uri.properties



# topology问题排查:

topology跑的时候 会实时的在zookeeper上记录当前跑到哪个offset，如果间隔太久（超过kafka日志保存的时间），当前记录的offset对应的日志会被删除，所以会出现kafka offset out of range

如果是测试环境 那无关紧要 因为测试环境的storm就测试的时候会开一下

如果是正式环境 那就是有问题的 要及时提出来

测试环境的操作 可以自己到kafka001.sz的机器

1.zk命令是 /usr/local/src/zookeeper-3.4.9/bin/zkCli.sh启动客户端

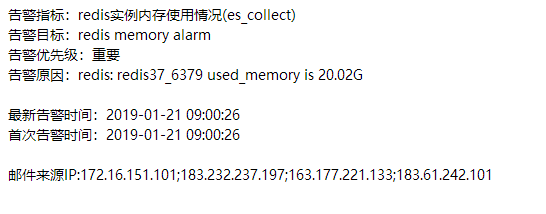
2.然后get /topologies/${topologyName}/，这个目录下的transactional

3.确定是自己的topology的目录,然后直接rmr /topologies/${topologyName}/transactional

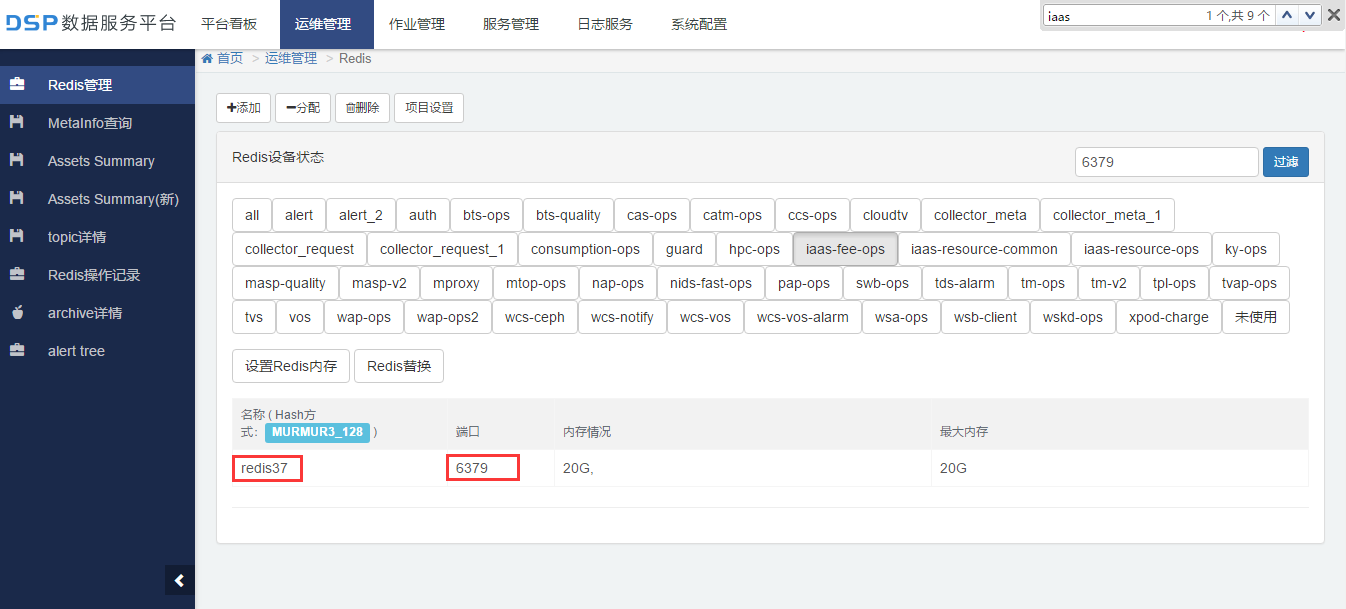
删掉这个目录 数据会从最新的offset取出来处理

txId一直告警，如果日志量很小，导致txId超过5分钟才更新一次回告警；如果是数据量太大，堆积也会告警；堆积的时候一般就会出现Topology is busy的异常 这个时候就要去ui上找一下status\_bolt，到对应的机器上搜一下status的关键字，查看一下哪个worker有堆积，然后到对应的机器上看日志，搜一下关键字QueueSize \d{4}，看一下4,5,6的数字是否有 如果有的话 就是堆积，(用\d{6}搜出来)，看这个日志对应的GatherBolt是哪一个，然后看看这个gather对应的client type的日志是否突增。如果没有 可以考虑把并发数加大，也就是topology的代码里面setGatherBolt的parallelism的参数加大

# redis阈值告警问题：

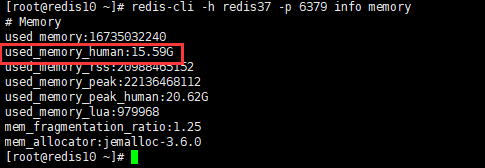


首先根据告警信息 **redis37\_6379**

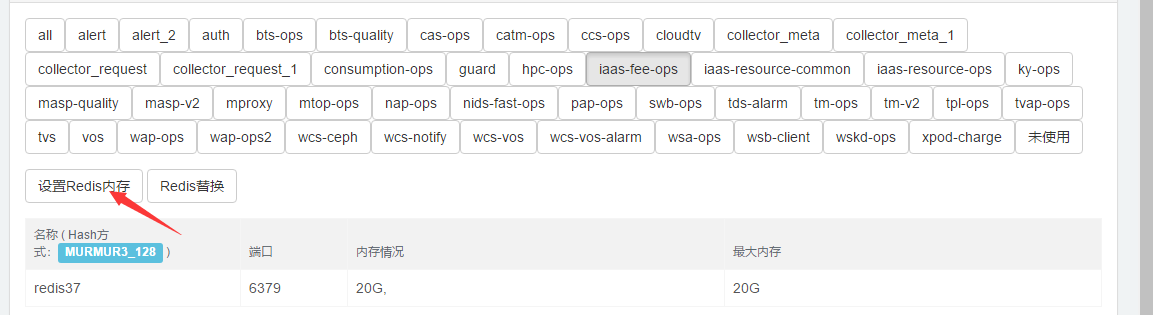
到portal找到对应的项目

然后上redis机器(随便一台) 输入命令：redis-cli -h redis37 -p 6379 **info memory**

查看当前项目使用的redis情况



询问业务方平时高峰期的时间段，在这个时间段去机器上观察这个值。



然后在这里设置redis内存，一般来说比高峰期高1~2G即可

# git操作

git项目同步所有分支：

git branch -r | grep -v '\->' | while read remote; do git branch --track "${remote#origin/}" "$remote"; done

stable/dsp/dsp-config-jndi/6.0#dsp-config-jndi-6.0.jar

stable/apache/hadoop/cdh5.8.2

stable/apache/hbase/cdh5.8.2

norm: -g org.apache.logging.log4j

norm: -g log4j

norm: -g org.slf4j

norm: -g org.apache.hadoop

hubspout

12号数据跑完

stable

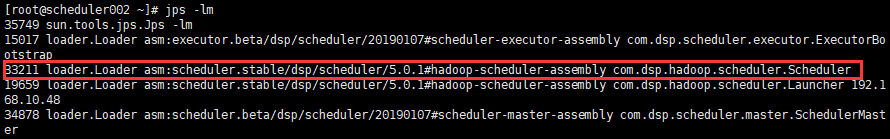
cp main.jar /usr/local/plus/topology/stable/xpod/bts/1.12/bts-ops-assembly/external/

# job调度问题：

重启调度:

ssh scheduler002.sz

先kill掉下图调度的进程



待kill成功后

再重启

loader main asm:scheduler.stable/dsp/scheduler/5.0.1#hadoop-scheduler-assembly （空格）/etc/hadoop-scheduler/hadoop-scheduler\_env.sh

调度日志查看:

/var/log/hadoop-scheduler/hadoop-scheduler.log

测试环境要记得ssh scheduler002.sz

vim /etc/hadoop-scheduler/jar2uri.properties

看相应camus或job是否指向要跑的版本

# kafka相关操作：

进入kafka目录

cd /usr/local/src/kafka\_2.11-0.10.1.0/bin/

查看相关topic下的kafka消费情况:

./kafka-console-consumer.sh --topic=wscloud-nova --bootstrap-server=kafka001.sz:9092 --partition=0 --offset=latest

最新

./kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list=kafka001.sz:9092 --topic=nids-fast --time=-1

最旧

./kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list=kafka001.sz:9092 --topic=nids-fast --time=-2