**阶段一模块四（zookeeper）**

**需求：基于Zookeeper实现简易版配置中心**

要求实现以下功能：

创建一个Web项目，将数据库连接信息交给Zookeeper配置中心管理，即：当项目Web项目启动时，从Zookeeper进行MySQL配置参数的拉取

要求项目通过数据库连接池访问MySQL（连接池可以自由选择熟悉的）

当Zookeeper配置信息变化后Web项目自动感知，正确释放之前连接池，创建新的连接池

**思路分析：**

1.定义一个用于发布数据库连接信息到zookeeper的接口，用来修改数据库连接信息

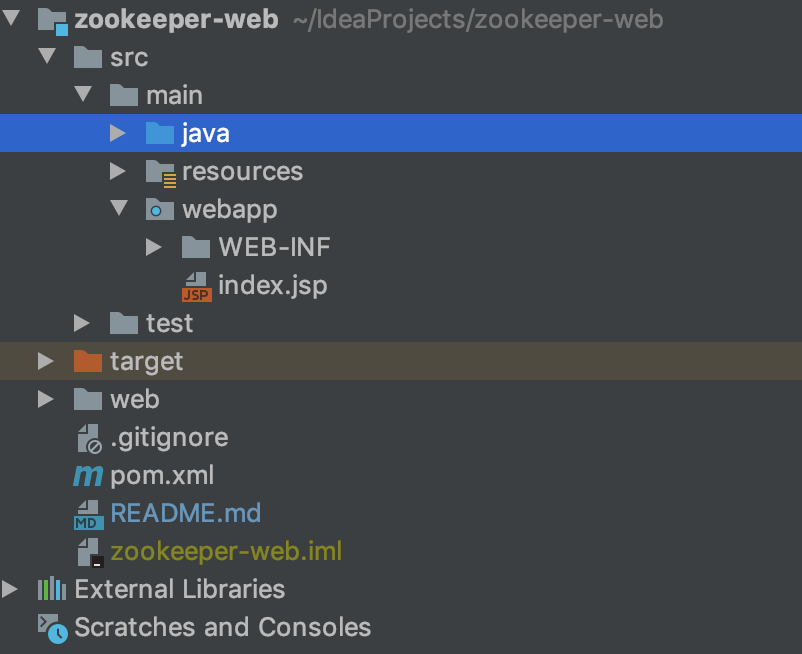
2.项目启动时从zookeeper获取数据库连接信息，创建数据库连接池

3.项目要时刻监听zookeeper中数据库连接信息的变化

4.当发布数据库连接信息到zookeeper中时，如果连接信息有变化，项目会重新从zookeeper中获取数据库连接信息，释放之前的连接池，并创建新的数据库连接池

**实现步骤：**

1.创建一个spring web项目，添加需要的依赖到pom文件。



<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.lagou</groupId>

<artifactId>zookeeper-web</artifactId>

<version>1.0</version>

<packaging>war</packaging>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.2.8.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>5.2.8.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.2.8.RELEASE</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>RELEASE</version>

</dependency>

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

<version>3.4.14</version>

</dependency>

<dependency>

<groupId>com.101tec</groupId>

<artifactId>zkclient</artifactId>

<version>0.2</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.46</version>

</dependency>

</dependencies>

<!-- JVM 运行环境 -->

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5.1</version>

<configuration>

<source>${java.version}</source>

<target>${java.version}</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

</plugins>

</build>

</project>

resources目录下新建日志配置文件：log4j.properties

log4j.rootLogger=INFO, Console

#Console

log4j.appender.Console=org.apache.log4j.ConsoleAppender

log4j.appender.Console.layout=org.apache.log4j.PatternLayout

log4j.appender.Console.layout.ConversionPattern=%-5p - %m%n

2.定义一个用于修改配置信息的接口：Publisher.java

package com.lagou.zk;

import org.I0Itec.zkclient.ZkClient;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class Publisher {

Logger logger = LoggerFactory.getLogger(Publisher.class);

ZkClient zkClient = null;

public static final String serverstring = "linux121:2181,linux122:2181,linux123:2181";

//

public static final String path = "/webapp/dblinkcfg";

/\*\*

\* 获取zk连接对象

\*/

private void connectZk() {

zkClient = new ZkClient(serverstring);

if (!zkClient.exists(path)) {

// 创建保存数据库连接信息的节点

zkClient.createPersistent(path, true);

}

}

/\*\*

\* 发布数据库配置信息

\*

\* @param cfgInfo 连接信息

\*/

public void publish(String cfgInfo) {

connectZk();

zkClient.writeData(path, cfgInfo);

logger.info("发布数据库连接信息成功：" + cfgInfo);

}

}

3.定义一个监听器，用于监听zk中保存配置信息的节点（webapp/dblinkcfg）

Listener.java

package com.lagou.zk;

import org.I0Itec.zkclient.IZkDataListener;

import org.I0Itec.zkclient.ZkClient;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.io.IOException;

import java.util.Properties;

public class Listener {

private static Logger logger = LoggerFactory.getLogger(Listener.class);

// zk服务器地址信息

public static final String serverstring = "linux121:2181,linux122:2181,linux123:2181";

// 获取ZkClient对象

private static ZkClient zkClient = new ZkClient(serverstring);

// 保存数据库配置信息的节点路径

private static String path = "/webapp/dblinkcfg";

public static void monitor() throws IOException {

zkClient.subscribeDataChanges(path, new IZkDataListener() {

public void handleDataChange(String dataPath, Object data) throws Exception {

logger.info("zk中的数据库配置信息发生修改！尝试重新获取数据库连接池...");

// 重新获取配置信息

String cfg = zkClient.readData(dataPath, true);

Properties pro = new Properties();

Utils.loadData(pro, cfg);

// 释放旧的连接池

ConnectionManager.clearPool();

// 创建新的连接池

Utils.createDbPool(pro);

}

public void handleDataDeleted(String dataPath) throws Exception {

logger.error("zk中的数据库配置信息已被删除！");

}

});

}

}

4.数据库连接管理器：ConnectionManager.java

package com.lagou.zk;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.util.LinkedList;

import java.util.List;

/\*\*

\* 数据库连接管理器

\*/

public class ConnectionManager {

private static List<Connection> pool = new LinkedList<Connection>();

public static String Url = "";

public static String USERNAME = "";

public static String PASSWORD = "";

public static String DRIVER = "";

public static int initCount;

public static int maxCount;

public static int currentCount;

private static volatile ConnectionManager instance = null;

private ConnectionManager() {

init();

}

public static ConnectionManager getInstance() {

if (null == instance) {

synchronized (ConnectionManager.class) {

if (null == instance) {

return new ConnectionManager();

}

}

}

return instance;

}

public static void init() {

addConnection();

}

public static void addConnection() {

for (int i = 0; i < initCount; i++) {

try {

pool.add(createConnection());

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

}

}

public static Connection createConnection() throws ClassNotFoundException {

Connection conn = null;

try {

Class.forName(DRIVER);

conn = DriverManager.getConnection(Url, USERNAME, PASSWORD);

} catch (SQLException e) {

e.printStackTrace();

}

return conn;

}

public static Connection getConnection() throws SQLException, ClassNotFoundException {

synchronized (pool) {

if (pool.size() > 0) {

System.out.println("Current Connection size is:" + pool.size());

return pool.get(0);

} else if (currentCount < maxCount) {

Class.forName(DRIVER);

Connection conn = createConnection();

pool.add(conn);

currentCount++;

return conn;

} else {

throw new SQLException("Current Connection is Zero");

}

}

}

/\*\*

\* 清空连接池,释放连接

\*/

public static void clearPool() {

for (Connection connection : pool) {

try {

connection.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

pool = new LinkedList<Connection>();

}

public static void release(Connection conn) {

pool.remove(conn);

}

}

Utils.java

package com.lagou.zk;

import java.io.IOException;

import java.io.StringReader;

import java.util.Properties;

public class Utils {

public static void loadData(Properties pro, String cfg) throws IOException {

pro.load(new StringReader(cfg));

}

public static void createDbPool(Properties pro) {

// 解析配置信息，创建数据库连接池

ConnectionManager.DRIVER = pro.getProperty("driverClassName");

ConnectionManager.Url = pro.getProperty("url");

ConnectionManager.USERNAME = pro.getProperty("username");

ConnectionManager.PASSWORD = pro.getProperty("password");

ConnectionManager.initCount = Integer.parseInt(pro.getProperty("initCount"));

ConnectionManager.maxCount = Integer.parseInt(pro.getProperty("maxCount"));

ConnectionManager.currentCount = Integer.parseInt(pro.getProperty("currentCount"));

ConnectionManager.init();

}

}

5.启动时初始化数据库连接池并接听

package com.lagou.zk;

import org.I0Itec.zkclient.ZkClient;

import org.springframework.stereotype.Component;

import javax.annotation.PostConstruct;

import java.io.IOException;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.Properties;

@Component

public class Startup {

@PostConstruct

public void init() throws IOException, SQLException, ClassNotFoundException {

ZkClient zkClient = new ZkClient("linux121:2181,linux122:2181,linux123:2181");

String cfg = zkClient.readData("/webapp/dblinkcfg", true);

Properties pro = new Properties();

Utils.loadData(pro, cfg);

// 创建数据库连接池

Utils.createDbPool(pro);

// 监听节点数据的变化

Listener.monitor();

// 使用连接池测试

Connection conn = ConnectionManager.getConnection();

PreparedStatement preparedStatement = conn.prepareStatement("select \* from execution\_jobs");

ResultSet resultSet = preparedStatement.executeQuery();

if (resultSet.next()) {

System.out.println(resultSet.getString("flow\_id"));

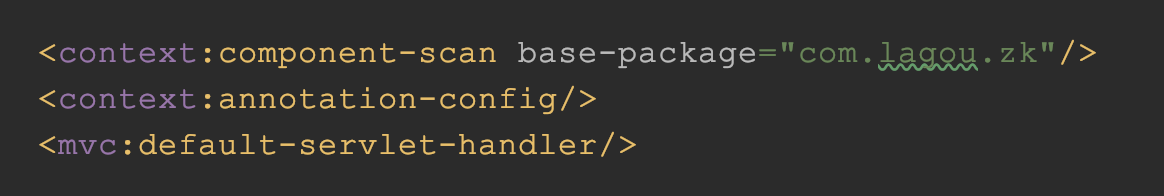
System.out.println(resultSet.getString("job\_id"));

}

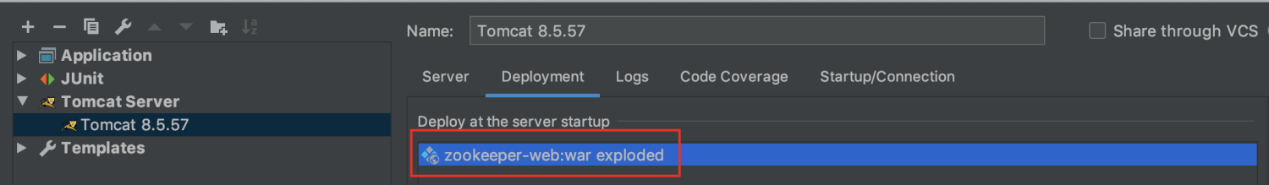
}

}

需要在applicationContext.xml中配置自动扫描和注解驱动



6.将应用部署到Tomcat



7.定义测试类，往zk中/webapp/dblinkcfg节点的发布数据库配置信息

package com.lagou.zk;

import org.junit.Test;

public class PublisherTest {

Publisher publisher = new Publisher();

@Test

public void publish() {

String cfg = "driverClassName=com.mysql.jdbc.Driver\n" +

"url=jdbc:mysql://linux123:3306/azkaban\n" +

"username=hive\n" +

"password=12345678\n" +

"initCount=5\n" +

"maxCount=10\n" +

"currentCount=5";

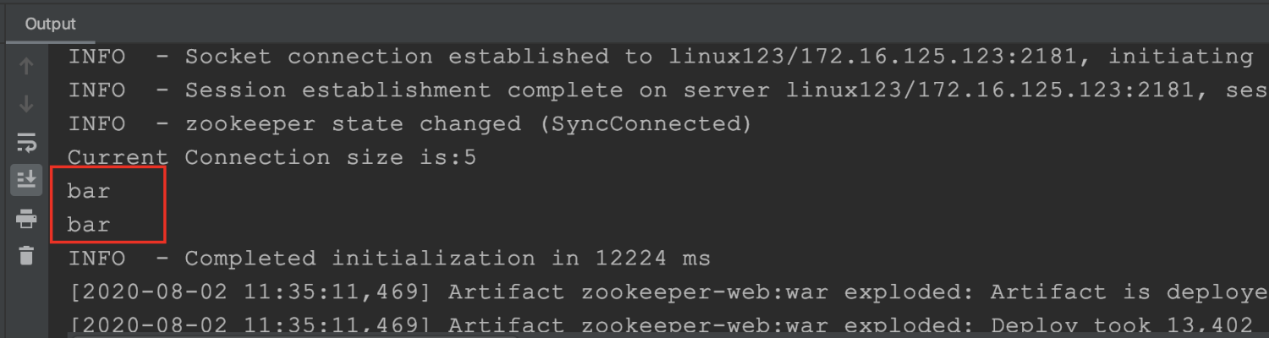
publisher.publish(cfg);

System.out.println(publisher.zkClient.readData("/webapp/dblinkcfg", true).toString());

}

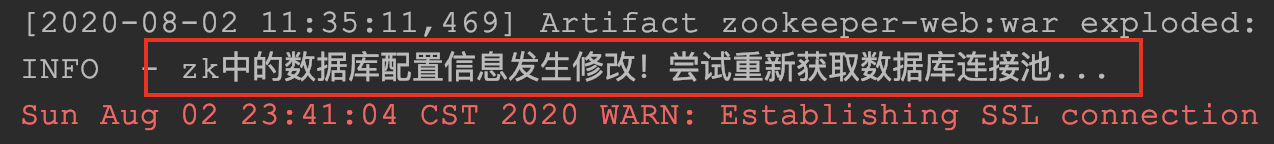
}

7.启动应用，查看状态

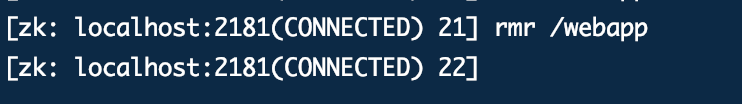


可以看到，成功获取到了数据库连接并查询到了数据库表中的信息。

8.修改zk中/webapp/dblinkcfg节点的数据：username=hive改为username=root，可以看到日志信息



9.删除节点/webapp/dblinkcfg





完成！！！

# 阶段一模块四（HBase）

## Hbase作业

在社交网站，社交APP上会存储有大量的用户数据以及用户之间的关系数据，比如A用户的好友列表会展示出他所有的好友，现有一张Hbase表，存储就是当前注册用户的好友关系数据，如下



需求

1. 使用Hbase相关API创建一张结构如上的表
2. 删除好友操作实现（好友关系双向，一方删除好友，另一方也会被迫删除好友）  
   例如：uid1用户执行删除uid2这个好友，则uid2的好友列表中也必须删除uid1

## 实现步骤

1. 创建maven工程，pom中引入相关jar包。
2. 编写Java代码，创建HBase表user\_rel

package com.lagou.hbase;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hbase.HBaseConfiguration;

import org.apache.hadoop.hbase.HColumnDescriptor;

import org.apache.hadoop.hbase.HTableDescriptor;

import org.apache.hadoop.hbase.TableName;

import org.apache.hadoop.hbase.client.Connection;

import org.apache.hadoop.hbase.client.ConnectionFactory;

import org.apache.hadoop.hbase.client.HBaseAdmin;

import java.io.IOException;

public class HomeworkCreateTab {

public static void main(String[] args) {

Configuration conf = HBaseConfiguration.create();

conf.set("hbase.zookeeper.quorum", "linux121,linux122,linux123");

conf.set("hbase.zookeeper.property.clientPort", "2181");

Connection conn = null;

try {

conn = ConnectionFactory.createConnection(conf);

HBaseAdmin admin = (HBaseAdmin) conn.getAdmin();

//创建Htabledesc描述器，表描述器

HTableDescriptor tableDescriptor = new HTableDescriptor(TableName.valueOf("user\_rel"));

//指定列族

tableDescriptor.addFamily(new HColumnDescriptor("friends"));

admin.createTable(tableDescriptor);

System.out.println("创建表user\_rel成功！");

admin.close();

conn.close;

} catch (IOException e) {

System.out.println("获取hbase连接失败！");

}

}

}

运行main方法，hbase shell中验证表是否创建成功

1. 创建Observer 协处理器，实现当uid1删除uid2时，触发uid2删除uid1的操作

package com.lagou.hbase.coprocessor;

import com.sun.org.apache.xpath.internal.operations.String;

import org.apache.hadoop.hbase.Cell;

import org.apache.hadoop.hbase.CellUtil;

import org.apache.hadoop.hbase.TableName;

import org.apache.hadoop.hbase.client.Delete;

import org.apache.hadoop.hbase.client.Durability;

import org.apache.hadoop.hbase.client.HTableWrapper;

import org.apache.hadoop.hbase.coprocessor.BaseRegionObserver;

import org.apache.hadoop.hbase.coprocessor.ObserverContext;

import org.apache.hadoop.hbase.coprocessor.RegionCoprocessorEnvironment;

import org.apache.hadoop.hbase.regionserver.wal.WALEdit;

import org.apache.hadoop.hbase.util.Bytes;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import java.io.IOException;

import java.util.List;

public class Deletecoprocessor extends BaseRegionObserver {

Logger logger = LoggerFactory.getLogger(Deletecoprocessor.class);

@Override

public void postDelete(ObserverContext<RegionCoprocessorEnvironment> e, final Delete delete, WALEdit edit, Durability durability) throws IOException {

HTableWrapper userRel = (HTableWrapper) e.getEnvironment().getTable(TableName.valueOf("user\_rel"));

//获取删除的行rowkey

byte[] row = delete.getRow();

//获取到firends下所有的cell

List<Cell> friends = delete.getFamilyCellMap().get(Bytes.toBytes("friends"));

for (Cell friend : friends) {

byte[] bytes = CellUtil.cloneQualifier(friend);

Delete delete1 = new Delete(bytes);

delete1.addColumn(Bytes.toBytes("friends"), row);

userRel.delete(delete1);

}

userRel.flushCommits();

userRel.close();

}

}

1. 将项目打成jar包并上传到HDFS
2. 挂载协处理器

alter 'user\_rel',METHOD => 'table\_att','Coprocessor'=>'hdfs://linux121:9000/processor/deleteprocessor.jar|com.lagou.hbase.coprocessor.Deletecoprocessor|1001|'



1. 验证

添加测试数据

hbase(main):021:0> put 'user\_rel','1000','friends:1001','1001'

hbase(main):022:0> put 'user\_rel','1000','friends:1002','1002'

hbase(main):024:0> put 'user\_rel','1001','friends:1000','1000'

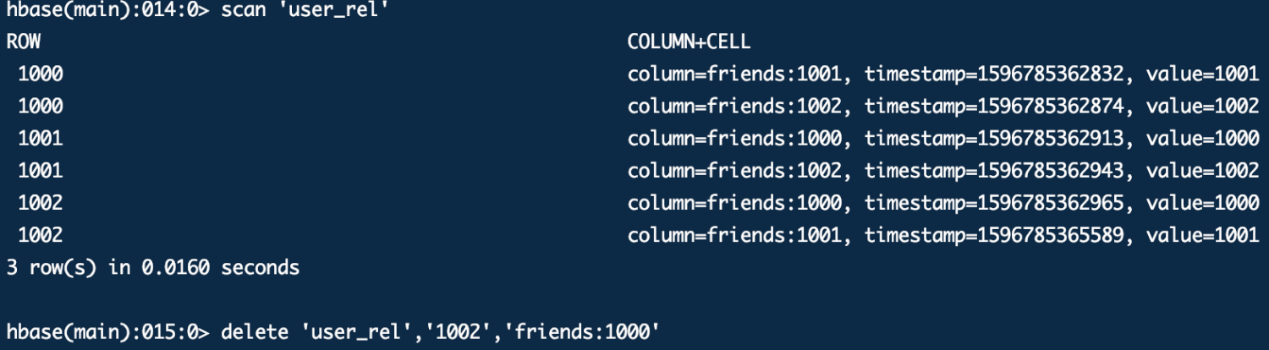
hbase(main):025:0> put 'user\_rel','1001','friends:1002','1002'

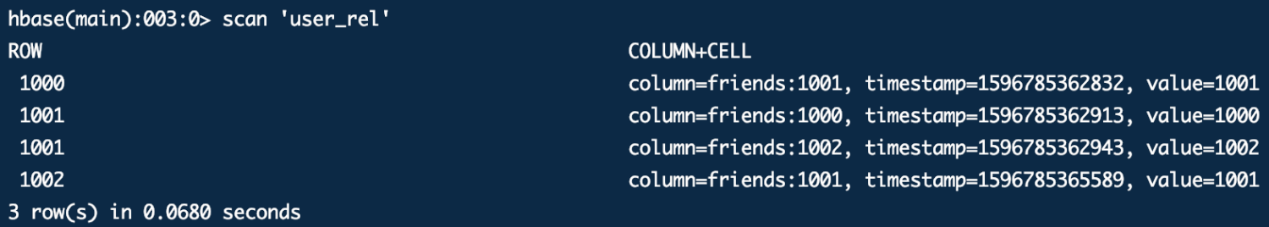
hbase(main):027:0> put 'user\_rel','1002','friends:1000','1000'

hbase(main):028:0> put 'user\_rel','1002','friends:1001','1001'

删除数据，并查看表数据

delete 'user\_rel','1002','friends:1000'





三、Azkaban

现有用户点击行为数据文件，每天产生会上传到hdfs目录，按天区分目录，现在我们需要每天凌晨两点定时导入Hive表指定分区中，并统计出今日活跃用户数插入指标表中。

日志文件

clicklog

userId   click\_time             index  
uid1 2020-06-21 12:10:10 a.html  
uid2 2020-06-21 12:15:10 b.html  
uid1 2020-06-21 13:10:10 c.html  
uid1 2020-06-21 15:10:10 d.html  
uid2 2020-06-21 18:10:10 e.html

用户点击行为数据，三个字段是用户id,点击时间，访问页面

hdfs目录会以日期划分文件，例如：

/user\_clicks/20200621/clicklog.dat  
/user\_clicks/20200622/clicklog.dat  
/user\_clicks/20200623/clicklog.dat  
...

Hive表

原始数据分区表

create table user\_clicks(id string,click\_time string ,index string) partitioned by(dt string) row format delimited fields terminated by '\t' ;

需要开发一个import.job每日从hdfs对应日期目录下同步数据到该表指定分区。（日期格式同上或者自定义）

指标表

create table user\_info(active\_num string,dateStr string) row format delimited fields terminated by '\t';

需要开发一个analysis.job依赖import.job执行，统计出每日活跃用户(一个用户出现多次算作一次)数并插入user\_inof表中。

开发以上提到的两个job，job文件内容和sql内容需分开展示，并能使用azkaban调度执行。

把clicklog日志数据上传到HDFS对应日期目录

hdfs dfs -put clicklog /user\_clicks/20210401

import.sh脚本把clicklog日志数据读入Hive数据库

#!/bin/sh

echo 'import data from hdfs。。。'

currDate=`date +%Y%m%d`

echo "现在时间：'$currDate'"

/opt/lagou/servers/hive-2.3.7/bin/hive -e "USE default;LOAD DATA INPATH '/user\_clicks/$currDate/\*' OVERWRITE INTO TABLE user\_clicks PARTITION (dt='$currDate');"

Import.job文件内容：

type=command

command=sh import.sh

analysis.sh脚本内容，分析user\_clicks表，然后数据插入user\_info表

#!/bin/sh

echo 'analysis user click。。。'

currDate=`date +%Y-%m-%d`

echo "现在时间：'$currDate'"

/opt/lagou/servers/hive-2.3.7/bin/hive -e "USE default;INSERT INTO TABLE user\_info SELECT COUNT(DISTINCT id), TO\_DATE(click\_time) FROM user\_clicks WHERE TO\_DATE(click\_time) = '$currDate' GROUP BY TO\_DATE(click\_time);"

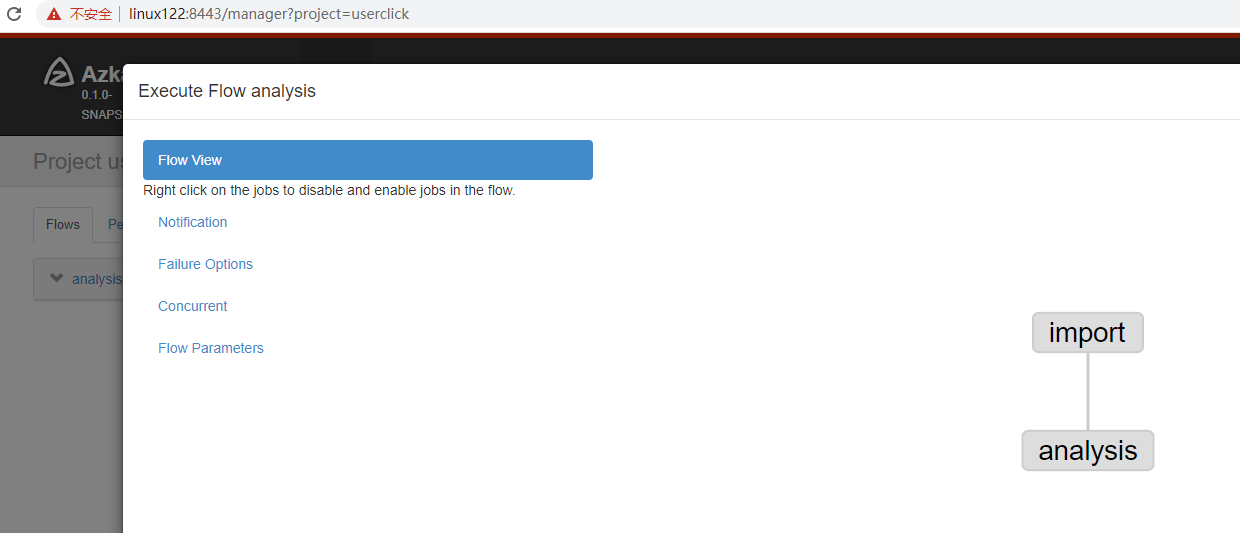
analysis.job内容

type=command

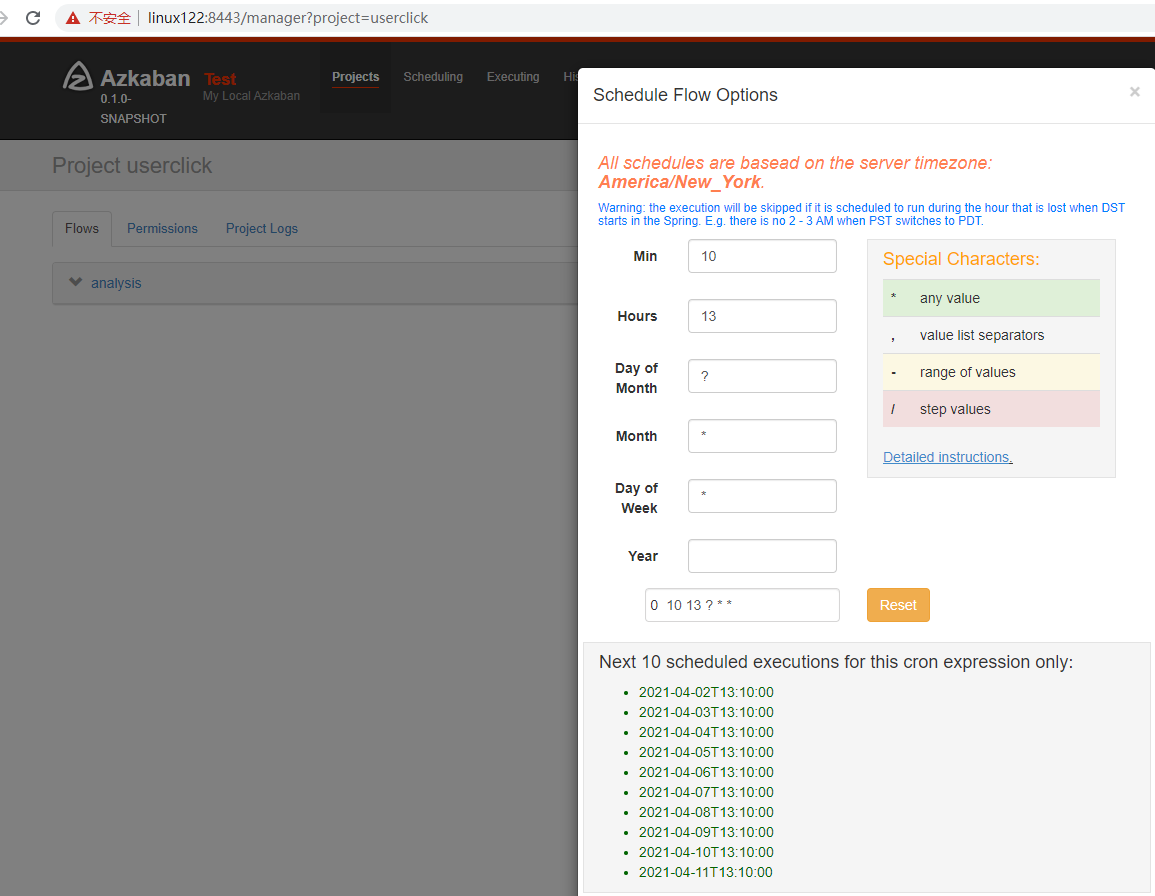
dependencies=import

command=sh analysis.sh

四个文件打包后上传至azkaban项目中：



设置项目每天下午13:10执行



观察定时执行的结果:执行成功！数据也插入了Hive表格中：

