# 正式班阶段一模块四作业

提交人：田龙

## ZooKeeper作业：

### 需求

1. 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
2. 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）
3. 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

### 实现方法

#### 实现思路

1、通过Druid连接连接数据库，连接数库信息通过Zk获取

2、ZK中Data信息存储JSON数据，JSON解析为Map后，传入DruidDataSourceFactory.createDataSource(Map)

3、根据ZK监听DataChange，删除或修改后更新Map，再更新DataSource

4、测试实现结果则修改所用database，由azkaban改为hivemetadata

#### 具体实现

ZK准备：

set /databaseConfig {"driverClassName":"com.mysql.jdbc.Driver","url":"jdbc:mysql://192.168.80.123:3306/azkaban?useSSL=false&useUnicode=true&characterEncoding=utf8","username":"root","password":"12345678"}

```java
private ZkClient zkClient;
private DataSource dataSource;

public static void main(String[] args) {
    try {
        StartZkDataSync startZkDataSync = new StartZkDataSync();
        // 初始化ZK
        startZkDataSync.initZK();
        // 获取ZK中databaseConfig并创建DataSource
        startZkDataSync.DBConfig();
        // 测试修改
        startZkDataSync.testChange();
        // 将任务持久化
        Thread.sleep(Integer.MAX_VALUE);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

/**
 * 初始化ZK
 */
private void initZK() {
    // 获取ZK
    zkClient = new ZkClient( serverstring: "192.168.80.121:2181");
    // 自定义序列化类
    zkClient.setZkSerializer(new ZkDatabaseSerializer());
    // 若没有文件则创建
    boolean exists = zkClient.exists( path: "/databaseConfig");
    if (!exists){
        zkClient.createPersistent( path: "/databaseConfig", data: "{\"driverClassName\":\"com.mysql.jdbc.Driver\",\"url\":\
    }
    // 声明一个DataChange监听
    zkClient.subscribeDataChanges( path: "/databaseConfig", new IZkDataListener() {
        // 修改事件监听
        @Override
        public void handleDataChange(String path, Object data) throws Exception {
            System.out.println(path + " has changed....");
            System.out.println("New data = " + data);
            DBConfig();
        }
        // 删除事件监听
        @Override
        public void handleDataDeleted(String path) throws Exception {
            System.out.println(path + " has deleted!!!");
        }
    });
}

/**
 * 获取ZK中databaseConfig并创建DataSource
 */
private void DBConfig() throws Exception {
    Map configMap = getConfigMap();
    updateDataSource(configMap);
    testSQL();
}

/**
* 更新DataSource
 */
private void updateDataSource(Map configMap) throws Exception {
    dataSource = DruidDataSourceFactory.createDataSource(configMap);
}
```

```java
/**
 *  获取databaseConfig中的data,返回Map格式
 */
private Map getConfigMap() throws Exception {
    // 获取databaseConfig中的data
    Object data = zkClient.readData( path: "/databaseConfig");
    // 转为JSON
    JSONObject propertiesJSON = JSON.parseObject(data.toString());
    // 返回Map
    return propertiesJSON.toJavaObject(Map.class);
}

/**
* 测试修改
 */
private void testChange() throws InterruptedException, SQLException {
    // 更新节点的数据 删除节点 验证监听器是否正常运行
    Object readData = zkClient.readData( path: "/databaseConfig");
    System.out.println("readData = " + readData);
    zkClient.writeData( path: "/databaseConfig",  object: "{\"driverClassName\": \"com.mysql.jdbc.Driver\",\"url\":\"jdbc:mys
    Thread.sleep( millis: 1000);
    zkClient.delete( path: "/databaseConfig");
}

/**
 * 测试执行的SQL
 */
private void testSQL() throws SQLException {
    Connection connection = dataSource.getConnection();
    PreparedStatement preparedStatement = connection.prepareStatement( sql: "show tables ");
    ResultSet resultSet = preparedStatement.executeQuery();
    while (resultSet.next()){
        String databasesStr = resultSet.getString( columnIndex: 1);
        System.out.println(databasesStr);
    }
}
```

## 执行结果

```
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
QRTZ_BLOB_TRIGGERS
QRTZ_CALENDARS
QRTZ_CRON_TRIGGERS
QRTZ_FIRED_TRIGGERS
QRTZ_JOB_DETAILS
QRTZ_LOCKS
QRTZ_PAUSED_TRIGGER_GRPS
QRTZ_SCHEDULER_STATE
QRTZ_SIMPLE_TRIGGERS
QRTZ_SIMPROP_TRIGGERS
QRTZ_TRIGGERS
active_executing_flows
active_sla
execution_dependencies
execution_flows
execution_jobs
execution_logs
executor_events
executors
project_events
project_files
project_flow_files
project_flows
project_permissions
project_properties
project_versions
projects
properties
triggers
readData = {"driverClassName":"com.mysql.jdbc.Driver","url":"jdbc:mysql://192.168.80.123:3306/azkaban?useSSL=false&amp;useUnicode=true&amp;characterEncoding=utf8","username":"
/databaseConfig has changed
New data = {"driverClassName": "com.mysql.jdbc.Driver","url":"jdbc:mysql://192.168.80.123:3306/hivemetadata?useSSL=false&amp;useUnicode=true&amp;characterEncoding=utf8","usern
AUX_TABLE
BUCKETING_COLS
CDS
COLUMNS_V2
COMPACTION_QUEUE
COMPLETED_COMPACTIONS
COMPLETED_TXN_COMPONENTS
DATABASE_PARAMS
DBS
DB_PRIVS
DELEGATION_TOKENS
FUNCS
FUNC_RU
GLOBAL_PRIVS
HIVE_LOCKS
IDXS
INDEX_PARAMS
KEY_CONSTRAINTS
MASTER_KEYS
NEXT_COMPACTION_QUEUE_ID
NEXT_LOCK_ID
NEXT_TXN_ID
NOTIFICATION_LOG
NOTIFICATION_SEQUENCE
NUCLEUS_TABLES
PARTITIONS
Key Promoter X
Command Run missed 419 time(s)
'Alt+4'  (Don't show again)
```

## HBase作业：

### 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

### 实现方法

#### 实现思路

1、通过API获得连接，创建表及内容

2、通过重写BaseRegionObserver中postDelete方法实现类似Oracle中Trigger的AFTER DELETE Trigger，在删除后继续执行逻辑

#### 具体实现

```java
public class HBaseInsert {

    Configuration conf = null;
    Connection connection = null;

    public static void main(String[] args) {
        try {
            HBaseInsert hBaseInsert = new HBaseInsert();
            hBaseInsert.init();
//            hBaseInsert.createTable();
            hBaseInsert.putData();
            hBaseInsert.release();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }


    /**
     * 初始化
     */
    private void init() throws IOException {
        // 初始化配置文件 获取配置文件对象
        conf = HBaseConfiguration.create();
        conf.set("hbase.zookeeper.quorum", "192.168.80.121,192.168.80.122,192.168.80.123");
        conf.set("hbase.zookeeper.property.clientPort", "2181");
        // 通过conf获取到HBase集群的连接
        connection = ConnectionFactory.createConnection(conf);
    }

    /**
     * 创建表
     */
    private void createTable() throws IOException {
        // 获取admin对象
        HBaseAdmin admin = (HBaseAdmin) connection.getAdmin();
        // 创建HTableDescriptor，并命名表名
        HTableDescriptor relationship = new HTableDescriptor(TableName.valueOf("relationship"));
        // 添加列族
        relationship.addFamily(new HColumnDescriptor( familyName: "friends"));
        // 创建表
        admin.createTable(relationship);
    }

    /**
     * 插入数据
     */
    private void putData() throws IOException {
        // 获取表
        Table relationship = connection.getTable(TableName.valueOf("relationship"));
        List<Put> putList = new ArrayList<>();
        // 创建两个 Put 对象，声明列族，值
        Put put1 = new Put(Bytes.toBytes( s: "uid1"));
        put1.addColumn(Bytes.toBytes( s: "friends"),Bytes.toBytes( s: "uid2"), Bytes.toBytes( s: "uid2"));
        put1.addColumn(Bytes.toBytes( s: "friends"),Bytes.toBytes( s: "uid5"), Bytes.toBytes( s: "uid7"));
        put1.addColumn(Bytes.toBytes( s: "friends"),Bytes.toBytes( s: "uid5"), Bytes.toBytes( s: "uid7"));
        Put put2 = new Put(Bytes.toBytes( s: "uid2"));
        put2.addColumn(Bytes.toBytes( s: "friends"),Bytes.toBytes( s: "uid1"), Bytes.toBytes( s: "uid1"));
        put2.addColumn(Bytes.toBytes( s: "friends"),Bytes.toBytes( s: "uid3"), Bytes.toBytes( s: "uid3"));
        put2.addColumn(Bytes.toBytes( s: "friends"),Bytes.toBytes( s: "uid6"), Bytes.toBytes( s: "uid6"));
        put2.addColumn(Bytes.toBytes( s: "friends"),Bytes.toBytes( s: "uid10"), Bytes.toBytes( s: "uid10"));
        // 将Put对象插入到List中
        putList.add(put1);
        putList.add(put2);
        // 向表中插入Put对象
        relationship.put(putList);
    }

    /**
     * 释放资源
     */
    private void release() throws IOException {
        connection.close();
    }
```

```java
 * @author Tianlong
 */
public class DeleteCoprocessor extends BaseRegionObserver {

    @Override
    public void postDelete(ObserverContext<RegionCoprocessorEnvironment> e, Delete delete, WALEdit edit, Durability durability) throws IOException {
        // 获取表
        Table relationship = e.getEnvironment().getTable(TableName.valueOf("relationship"));
        // 通过delete获取uid1中的好友uid2的值
        List<Cell> friends = delete.getFamilyCellMap().get(Bytes.toBytes( s: "friends"));
        if (CollectionUtils.isEmpty(friends)) {
            relationship.close();
            return;
        }
        Cell cell = friends.get(0);
        // uid2的列族中删除uid1
        // 经查源码 该方法过期 使用CellUtil中方法
        // Delete delete1 = new Delete(cell.getRow());
        Delete delete1 = new Delete(CellUtil.cloneQualifier(cell));
        delete1.addColumn(Bytes.toBytes( s: "friends"), CellUtil.cloneRow(cell));
        relationship.delete(delete1);
        // 关闭资源
        relationship.close();
    }
}
```

**执行结果**

```
hbase(main):016:0> scan 'relationship'
ROW                                     COLUMN+CELL
 uid1                                   column=friends:uid2, timestamp=1608266608113, value=uid2
 uid1                                   column=friends:uid5, timestamp=1608266608113, value=uid7
 uid2                                   column=friends:uid1, timestamp=1608266608113, value=uid1
 uid2                                   column=friends:uid10, timestamp=1608266608113, value=uid10
 uid2                                   column=friends:uid3, timestamp=1608266608113, value=uid3
 uid2                                   column=friends:uid6, timestamp=1608266608113, value=uid6
2 row(s) in 0.0190 seconds

hbase(main):017:0> delete 'relationship','uid1','friends:uid2'
```

```
hbase(main):001:0> scan 'relationship'
ROW                                     COLUMN+CELL
 uid1                                   column=friends:uid5, timestamp=1608271069245, value=uid7
 uid2                                   column=friends:uid10, timestamp=1608271069245, value=uid10
 uid2                                   column=friends:uid3, timestamp=1608271069245, value=uid3
 uid2                                   column=friends:uid6, timestamp=1608271069245, value=uid6
2 row(s) in 0.2260 seconds

hbase(main):002:0>
```

## Azkaban作业:

**需求**

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。

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。

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。

**实现方法**

**实现思路**

1、创建两个job，其中analysis.job依赖import.job

2、因import.job中需要时间参数，故编写shell脚本，脚本中使用hive -e命令执行带变量的hive语句

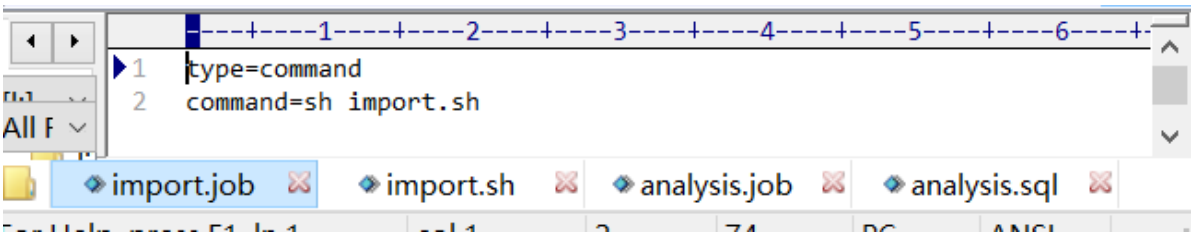
3、analysis.job可以调用analysis.sql中的逻辑，统计日活

4、编写analysis.sql

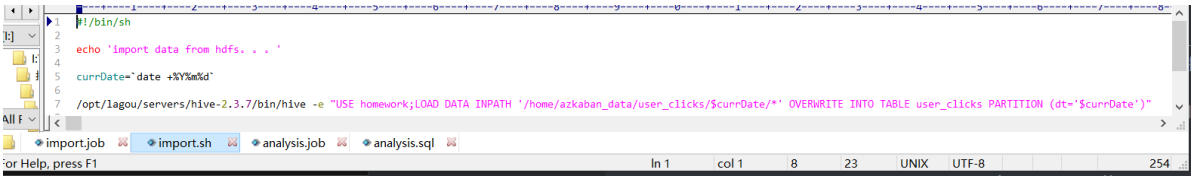
5、上传Azkaban，创建project后，通过cron表达式设置每天0200执行

**具体实现**

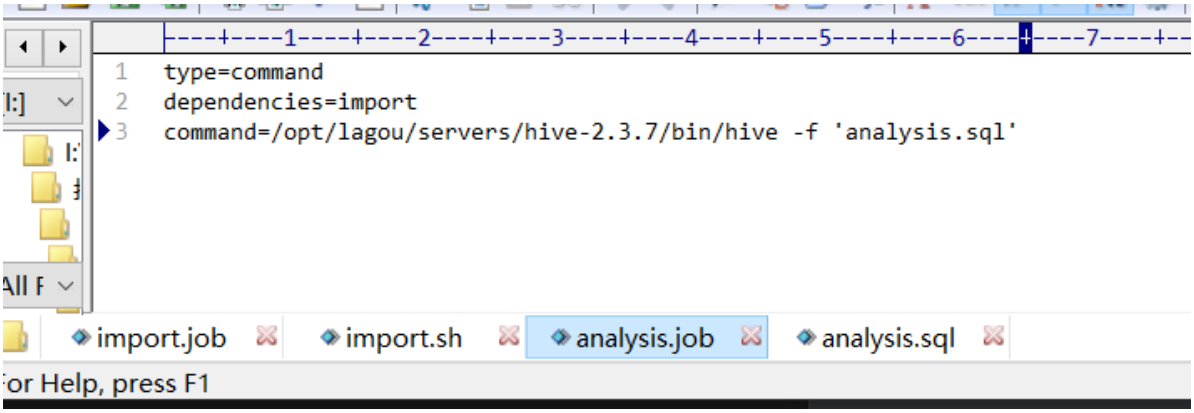
import.job



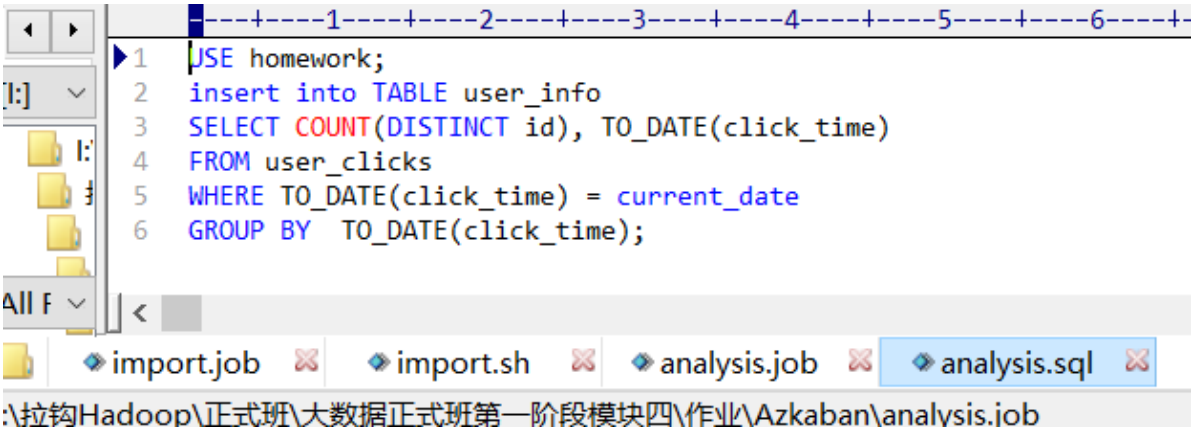

import.sh



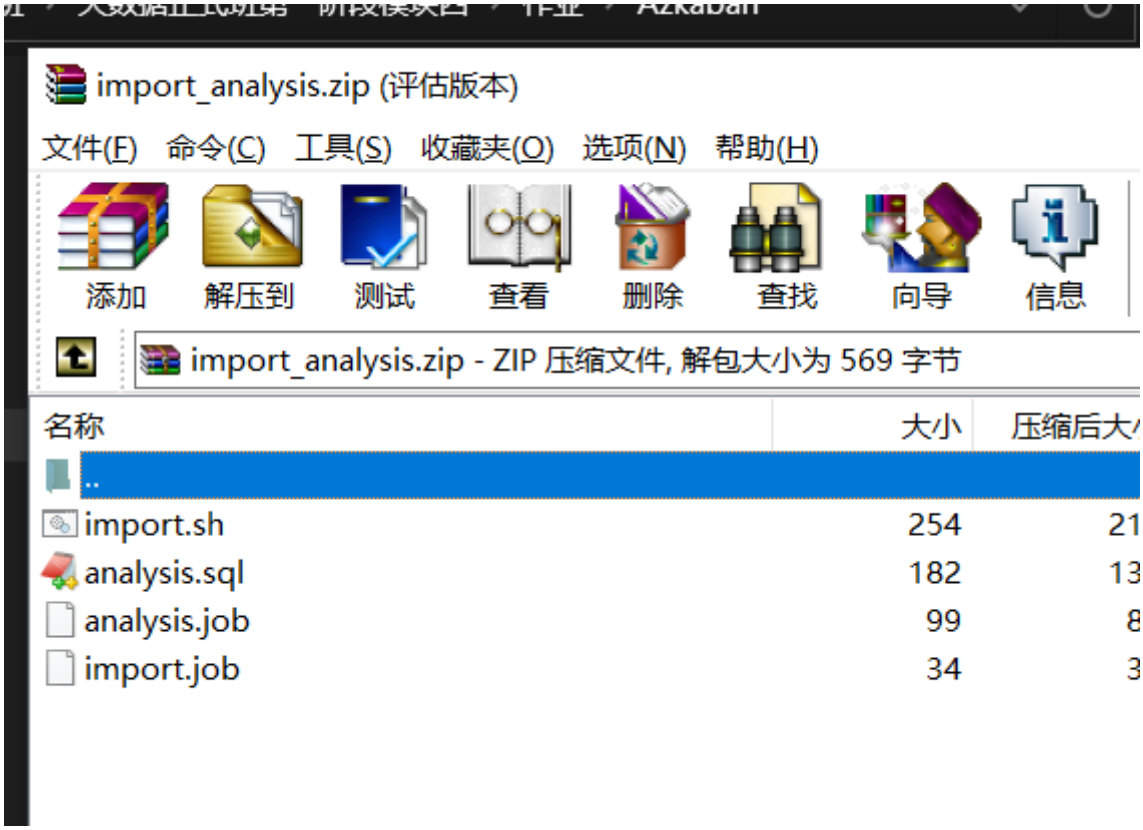

analysis.job



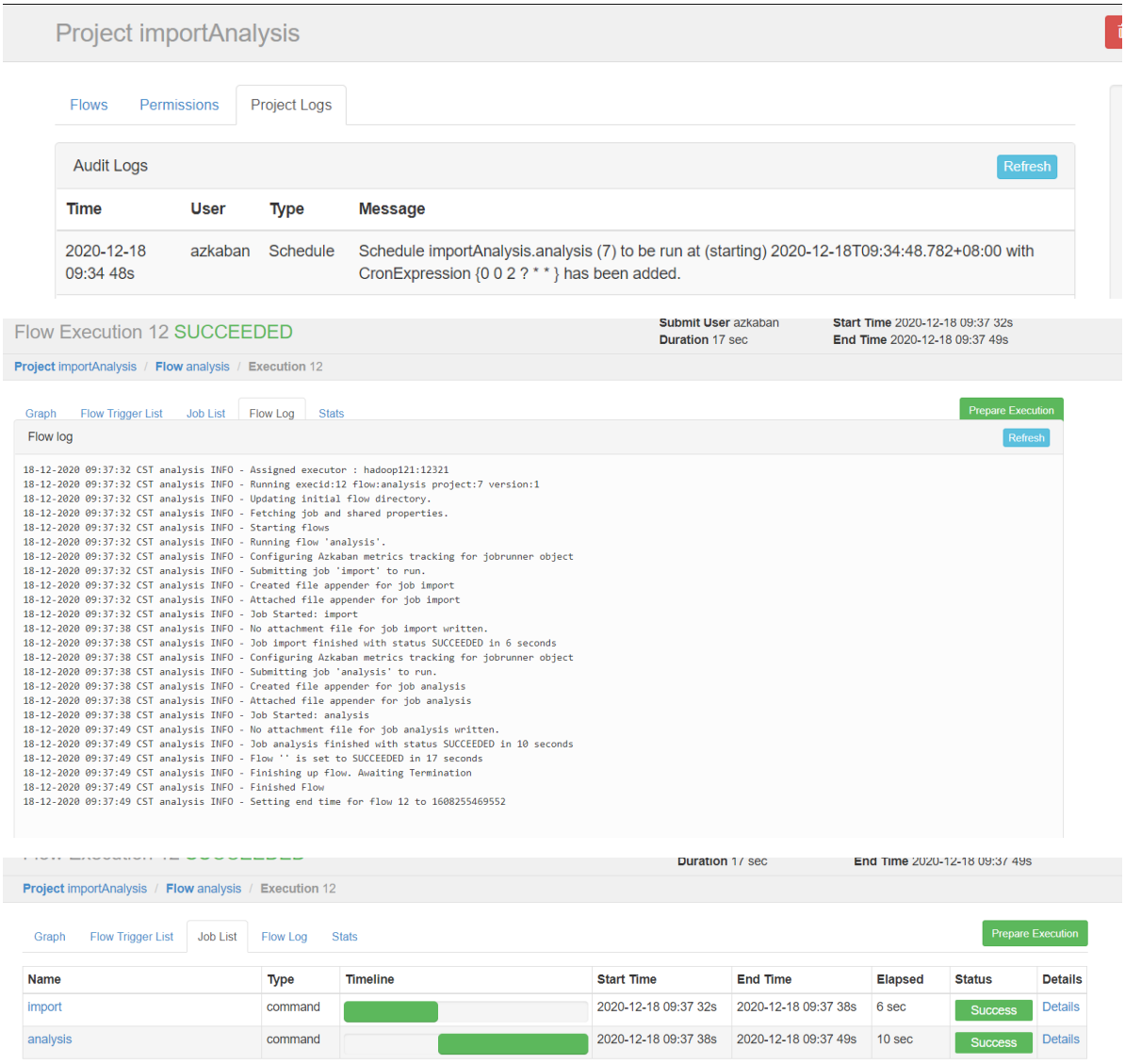

analysis.sql




打包


import_analysis.zip (评估版本)
文件(F) 命令(C) 工具(S) 收藏夹(O) 选项(N) 帮助(H)
添加 解压到 测试 查看 删除 查找 向导 信息
import_analysis.zip - ZIP 压缩文件, 解包大小为 569 字节
名称 大小 压缩后大小
..
import.sh 254 21
analysis.sql 182 13
analysis.job 99 8
import.job 34 3

**执行结果**


Project importAnalysis
Flows Permissions Project Logs
Audit Logs
Refresh
Time User Type Message
2020-12-18 09:34 48s azkaban Schedule Schedule importAnalysis.analysis (7) to be run at (starting) 2020-12-18T09:34:48.782+08:00 with CronExpression {0 0 2 ? * *} has been added.
Flow Execution 12 SUCCEEDED
Submit User azkaban
Start Time 2020-12-18 09:37 32s
Duration 17 sec
End Time 2020-12-18 09:37 49s
Project importAnalysis / Flow analysis / Execution 12
Graph Flow Trigger List Job List Flow Log Stats
Prepare Execution
Flow log
Refresh
18-12-2020 09:37:32 CST analysis INFO - Assigned executor : hadoop121:12321
18-12-2020 09:37:32 CST analysis INFO - Running execid:12 flow:analysis project:7 version:1
18-12-2020 09:37:32 CST analysis INFO - Updating initial flow directory.
18-12-2020 09:37:32 CST analysis INFO - Fetching job and shared properties.
18-12-2020 09:37:32 CST analysis INFO - Starting flows
18-12-2020 09:37:32 CST analysis INFO - Running flow 'analysis'.
18-12-2020 09:37:32 CST analysis INFO - Configuring Azkaban metrics tracking for jobrunner object
18-12-2020 09:37:32 CST analysis INFO - Submitting job 'import' to run.
18-12-2020 09:37:32 CST analysis INFO - Created file appender for job import
18-12-2020 09:37:32 CST analysis INFO - Attached file appender for job import
18-12-2020 09:37:32 CST analysis INFO - Job Started: import
18-12-2020 09:37:38 CST analysis INFO - No attachment file for job import written.
18-12-2020 09:37:38 CST analysis INFO - Job import finished with status SUCCEEDED in 6 seconds
18-12-2020 09:37:38 CST analysis INFO - Configuring Azkaban metrics tracking for jobrunner object
18-12-2020 09:37:38 CST analysis INFO - Submitting job 'analysis' to run.
18-12-2020 09:37:38 CST analysis INFO - Created file appender for job analysis
18-12-2020 09:37:38 CST analysis INFO - Attached file appender for job analysis
18-12-2020 09:37:38 CST analysis INFO - Job Started: analysis
18-12-2020 09:37:49 CST analysis INFO - No attachment file for job analysis written.
18-12-2020 09:37:49 CST analysis INFO - Job analysis finished with status SUCCEEDED in 10 seconds
18-12-2020 09:37:49 CST analysis INFO - Flow '' is set to SUCCEEDED in 17 seconds
18-12-2020 09:37:49 CST analysis INFO - Finishing up flow. Awaiting Termination
18-12-2020 09:37:49 CST analysis INFO - Finished Flow
18-12-2020 09:37:49 CST analysis INFO - Setting end time for flow 12 to 1608255469552
Duration 17 sec
End Time 2020-12-18 09:37 49s
Project importAnalysis / Flow analysis / Execution 12
Graph Flow Trigger List Job List Flow Log Stats
Prepare Execution
Name Type Timeline Start Time End Time Elapsed Status Details
import command 2020-12-18 09:37 32s 2020-12-18 09:37 38s 6 sec Success Details
analysis command 2020-12-18 09:37 38s 2020-12-18 09:37 49s 10 sec Success Details

```
hive (homework)> select * from user_clicks;
OK
user_clicks.id  user_clicks.click_time  user_clicks.index       user_clicks.dt
uid1    2020-12-18 12:10:10     a.html  20201218
uid2    2020-12-18 12:15:10     b.html  20201218
uid1    2020-12-18 13:10:10     c.html  20201218
uid1    2020-12-18 15:10:10     d.html  20201218
uid2    2020-12-18 18:10:10     e.html  20201218
Time taken: 0.097 seconds, Fetched: 5 row(s)
```

```
hive (homework)> select * from user_info;
OK
user_info.active_num    user_info.date_time
2       2020-12-18
Time taken: 0.098 seconds, Fetched: 1 row(s)
```