尚硅谷大数据技术之 Azkaban

官网：www.atguigu.com

一 概述

1.1 为什么需要工作流调度系统

1）一个完整的数据分析系统通常都是由大量任务单元组成：

shell 脚本程序，java 程序，mapreduce 程序、hive 脚本等

2）各任务单元之间存在时间先后及前后依赖关系

3）为了很好地组织起这样的复杂执行计划，需要一个工作流调度系统来调度执行；

例如，我们可能有这样一个需求，某个业务系统每天产生 20G 原始数据，我们每天都要对其进行处理，处理步骤如下所示：

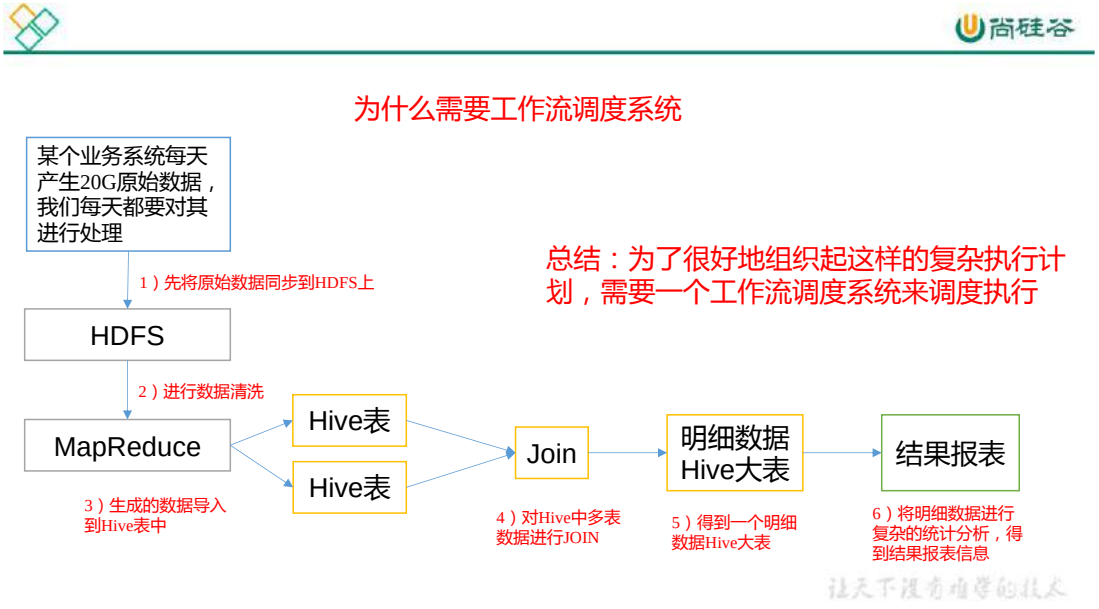
1) 通过 Hadoop 先将原始数据上传到 HDFS 上（HDFS 的操作）；

2) 使用 MapReduce 对原始数据进行清洗（MapReduce 的操作）；

3) 将清洗后的数据导入到 hive 表中（hive 的导入操作）；

4) 对 Hive 中多个表的数据进行 JOIN 处理，得到一张 hive 的明细表（创建中间表）；

5) 通过对明细表的统计和分析，得到结果报表信息（hive 的查询操作）；



1.2 Azkaban 的适用场景

根据以上业务场景： （2）任务依赖（1）任务的结果，（3）任务依赖（2）任务的结

果，（4）任务依赖（3）任务的结果，（5）任务依赖（4）任务的结果。一般的做法是，先执行完（1）再执行（2），再一次执行（3）（4）（5）。

这样的话，整个的执行过程都需要人工参加，并且得盯着各任务的进度。但是我们的很多任务都是在深更半夜执行的，通过写脚本设置 crontab 执行。其实，整个过程类似于一个有向无环图（DAG）。每个子任务相当于大任务中的一个节点，也就是，我们需要的就是一个工作流的调度器，而 Azkaban 就是能解决上述问题的一个调度器。

1.3 什么是 azkaban

Azkaban 是由 Linkedin 公司推出的一个批量工作流任务调度器，主要用于在一个工作流内以一个特定的顺序运行一组工作和流程，它的配置是通过简单的 key:value 对的方式，通过配置中的 dependencies 来设置依赖关系。Azkaban 使用 job 配置文件建立任务之间的依赖关系，并提供一个易于使用的 web 用户界面维护和跟踪你的工作流。

1.4 Azkaban 特点

1) 兼容任何版本的 hadoop
2) 易于使用的 Web 用户界面
3) 简单的工作流的上传
4) 方便设置任务之间的关系
5) 调度工作流
6) 模块化和可插拔的插件机制
7) 认证/授权(权限的工作)
8) 能够杀死并重新启动工作流
9) 有关失败和成功的电子邮件提醒

1.5 常见工作流调度系统

1）简单的任务调度：直接使用 crontab 实现；
2）复杂的任务调度：开发调度平台或使用现成的开源调度系统，比如 ooize、azkaban 等

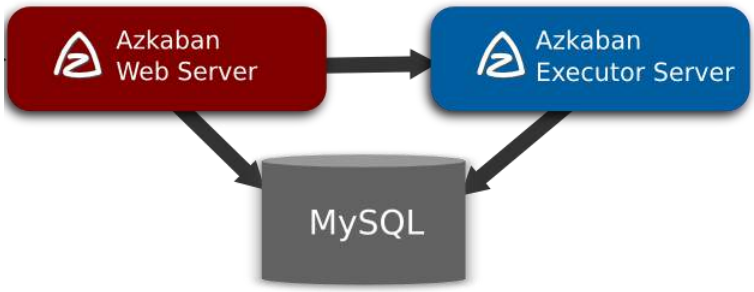
1.6 ooize 和 azkaban 特性对比

下面的表格对上述四种 hadoop 工作流调度器的关键特性进行了比较，尽管这些工作流调度器能够解决的需求场景基本一致，但在设计理念，目标用户，应用场景等方面还是存在显著的区别，在做技术选型的时候，可以提供参考

特性	Oozie	Azkaban
工作流描述语言	XML	text file with key/value pairs
是否要 web 容器	Yes	Yes
进度跟踪	web page	web page
Hadoop job 调度支持	yes	yes
运行模式	daemon	daemon
事件通知	no	Yes
需要安装	yes	yes
支持的 hadoop 版本	0.20+	currently unknown
重试支持	workflownode evel	yes
运行任意命令	yes	yes

1.7 Azkaban 的架构

Azkaban 由三个关键组件构成：



1) AzkabanWebServer：AzkabanWebServer 是整个 Azkaban 工作流系统的主要管理者，它用户登录认证、负责 project 管理、定时执行工作流、跟踪工作流执行进度等一系列任务。
2) AzkabanExecutorServer：负责具体的工作流的提交、执行，它们通过 mysql 数据库来协调任务的执行。
3) 关系型数据库（MySQL）：存储大部分执行流状态，AzkabanWebServer 和 AzkabanExecutorServer 都需要访问数据库。

1.8 Azkaban 下载地址

下载地址:http://azkaban.github.io/downloads.html

二 Azkaban 安装部署

2.1 安装前准备

1) 将 Azkaban Web 服务器、Azkaban 执行服务器、Azkaban 的 sql 执行脚本及 MySQL 安装包拷贝到 hadoop102 虚拟机/opt/software 目录下

a) azkaban-web-server-2.5.0.tar.gz

b) azkaban-executor-server-2.5.0.tar.gz

c) azkaban-sql-script-2.5.0.tar.gz

d) mysql-libs.zip

2) 选择 **Mysql** 作为 Azkaban 数据库，因为 Azkaban 建立了一些 Mysql 连接增强功能，以方便 Azkaban 设置，并增强服务可靠性。**（参见 hive 文档 2.4）**

2.2 安装 Azkaban

1) 在/opt/module/目录下创建 azkaban 目录

```
[atguigu@hadoop102 module]$ mkdir azkaban
```

2) 解压 azkaban-web-server-2.5.0.tar.gz、azkaban-executor-server-2.5.0.tar.gz、azkaban-sql-script-2.5.0.tar.gz 到/opt/module/azkaban 目录下

```
[atguigu@hadoop102 software]$ tar -zxvf azkaban-web-server-2.5.0.tar.gz -C /opt/module/azkaban/
[atguigu@hadoop102 software]$ tar -zxvf azkaban-executor-server-2.5.0.tar.gz -C /opt/module/azkaban/
[atguigu@hadoop102 software]$ tar -zxvf azkaban-sql-script-2.5.0.tar.gz -C /opt/module/azkaban/
```

3) 对解压后的文件重新命名

```
[atguigu@hadoop102 azkaban]$ mv azkaban-web-2.5.0/ server
[atguigu@hadoop102 azkaban]$ mv azkaban-executor-2.5.0/ executor
```

4) azkaban 脚本导入

进入 mysql，创建 azkaban 数据库，并将解压的脚本导入到 azkaban 数据库。

```
[atguigu@hadoop102 azkaban]$ mysql -uroot -p000000
mysql> create database azkaban;
mysql> use azkaban;
mysql> source /opt/module/azkaban/azkaban-2.5.0/create-all-sql-2.5.0.sql
```

注：source 后跟.sql 文件，用于批量处理.sql 文件中的 sql 语句。

2.3 生成密钥库

Keytool 是 java 数据证书的管理工具，使用户能够管理自己的公/私钥对及相关证书。

-keystore 指定密钥库的名称及位置(产生的各类信息将不在.keystore 文件中)

-genkey 在用户主目录中创建一个默认文件".keystore"

-alias 对我们生成的.keystore 进行指认别名；如果没有默认是 mykey

-keyalg 指定密钥的算法 RSA/DSA 默认是 DSA

1）生成 keystore 的密码及相应信息的密钥库

```
[atguigu@hadoop102 azkaban]$ keytool -keystore keystore -alias jetty -genkey -keyalg RSA
```

```
输入密钥库口令:
再次输入新口令:
您的名字与姓氏是什么?
  [Unknown]:
您的组织单位名称是什么?
  [Unknown]:
您的组织名称是什么?
  [Unknown]:
您所在的城市或区域名称是什么?
  [Unknown]:
您所在的省/市/自治区名称是什么?
  [Unknown]:
该单位的双字母国家/地区代码是什么?
  [Unknown]:
CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown 是否
正确?
  [否]:  y

输入 <jetty> 的密钥口令
        (如果和密钥库口令相同, 按回车):
再次输入新口令:
```

注意:

密钥库的密码至少必须 6 个字符，可以是纯数字或者字母或者数字和字母的组合等等

密钥库的密码最好和<jetty> 的密钥相同，方便记忆

2）将 keystore 拷贝到 azkaban web 服务器根目录中

```
[atguigu@hadoop102 azkaban]$ mv keystore /opt/module/azkaban/server/
```

2.4 时间同步配置

先配置好服务器节点上的时区

1） 如果在/usr/share/zoneinfo/这个目录下不存在时区配置文件 Asia/Shanghai，就要用 tzselect 生成。

```
[atguigu@hadoop102 azkaban]$ tzselect
Please identify a location so that time zone rules can be set correctly.
Please select a continent or ocean.
 1) Africa
 2) Americas
 3) Antarctica
 4) Arctic Ocean
 5) Asia
 6) Atlantic Ocean
 7) Australia
 8) Europe
 9) Indian Ocean
10) Pacific Ocean
11) none - I want to specify the time zone using the Posix TZ format.
#? 5
Please select a country.
 1) Afghanistan          18) Israel              35) Palestine
 2) Armenia              19) Japan               36) Philippines
 3) Azerbaijan           20) Jordan              37) Qatar
 4) Bahrain              21) Kazakhstan          38) Russia
 5) Bangladesh           22) Korea (North)       39) Saudi Arabia
 6) Bhutan               23) Korea (South)       40) Singapore
```

```
 7) Brunei              24) Kuwait              41) Sri Lanka
 8) Cambodia            25) Kyrgyzstan          42) Syria
 9) China               26) Laos                43) Taiwan
10) Cyprus              27) Lebanon             44) Tajikistan
11) East Timor          28) Macau               45) Thailand
12) Georgia             29) Malaysia            46) Turkmenistan
13) Hong Kong           30) Mongolia            47) United Arab Emirates
14) India               31) Myanmar (Burma)     48) Uzbekistan
15) Indonesia           32) Nepal               49) Vietnam
16) Iran                33) Oman                50) Yemen
17) Iraq                34) Pakistan
#? 9
Please select one of the following time zone regions.
1) Beijing Time
2) Xinjiang Time
#? 1

The following information has been given:

       China
       Beijing Time

Therefore TZ='Asia/Shanghai' will be used.
Local time is now:      Thu Oct 18 16:24:23 CST 2018.
Universal Time is now:  Thu Oct 18 08:24:23 UTC 2018.
Is the above information OK?
1) Yes
2) No
#? 1

You can make this change permanent for yourself by appending the line
       TZ='Asia/Shanghai'; export TZ
to the file '.profile' in your home directory; then log out and log in
again.

Here is that TZ value again, this time on standard output so that you
can use the /usr/bin/tzselect command in shell scripts:
Asia/Shanghai
```

2）拷贝该时区文件，覆盖系统本地时区配置

```
[atguigu@hadoop102   azkaban]$   cp   /usr/share/zoneinfo/Asia/Shanghai
/etc/localtime
```

3）集群时间同步（同时发给三个窗口）

```
[atguigu@hadoop102 azkaban]$ sudo date -s '2018-10-18 16:39:30'
```

2.5 配置文件

2.5.1 Web 服务器配置

1）进入 azkaban web 服务器安装目录 conf 目录，打开 azkaban.properties 文件

```
[atguigu@hadoop102 conf]$ pwd
/opt/module/azkaban/server/conf
[atguigu@hadoop102 conf]$ vim azkaban.properties
```

2）按照如下配置修改 azkaban.properties 文件。

```
#Azkaban Personalization Settings
#服务器 UI 名称,用于服务器上方显示的名字
azkaban.name=Test
#描述
```

```
azkaban.label=My Local Azkaban
#UI 颜色
azkaban.color=#FF3601
azkaban.default.servlet.path=/index
#默认 web server 存放 web 文件的目录
web.resource.dir=/opt/module/azkaban/server/web/
#默认时区,已改为亚洲/上海 默认为美国
default.timezone.id=Asia/Shanghai

#Azkaban UserManager class
user.manager.class=azkaban.user.XmlUserManager
#用户权限管理默认类（绝对路径）
user.manager.xml.file=/opt/module/azkaban/server/conf/azkaban-users.xml

#Loader for projects
#global 配置文件所在位置（绝对路径）
executor.global.properties=/opt/module/azkaban/executor/conf/global.pro
perties
azkaban.project.dir=projects

#数据库类型
database.type=mysql
#端口号
mysql.port=3306
#数据库连接 IP
mysql.host=hadoop102
#数据库实例名
mysql.database=azkaban
#数据库用户名
mysql.user=root
#数据库密码
mysql.password=000000
#最大连接数
mysql.numconnections=100

# Velocity dev mode
velocity.dev.mode=false

# Azkaban Jetty server properties.
# Jetty 服务器属性.
#最大线程数
jetty.maxThreads=25
#Jetty SSL 端口
jetty.ssl.port=8443
#Jetty 端口
jetty.port=8081
#SSL 文件名（绝对路径）
jetty.keystore=/opt/module/azkaban/server/keystore
#SSL 文件密码
jetty.password=000000
#Jetty 主密码与 keystore 文件相同
jetty.keypassword=000000
#SSL 文件名（绝对路径）
jetty.truststore=/opt/module/azkaban/server/keystore
#SSL 文件密码
jetty.trustpassword=000000

# Azkaban Executor settings
executor.port=12321

# mail settings
mail.sender=
```

```
mail.host=
job.failure.email=
job.success.email=

lockdown.create.projects=false

cache.directory=cache
```

3）web 服务器用户配置

在 azkaban web 服务器安装目录 conf 目录，按照如下配置修改 azkaban-users.xml 文件，增加管理员用户。

```
[atguigu@hadoop102 conf]$ vim azkaban-users.xml
<azkaban-users>
  <user      username="azkaban"      password="azkaban"      roles="admin"
groups="azkaban" />
  <user username="metrics" password="metrics" roles="metrics"/>
  <user username="admin" password="admin" roles="admin,metrics" />
  <role name="admin" permissions="ADMIN" />
  <role name="metrics" permissions="METRICS"/>
</azkaban-users>
```

2.5.2 执行服务器配置

1）进入执行服务器安装目录 conf，打开 azkaban.properties

```
[atguigu@hadoop102 conf]$ pwd
/opt/module/azkaban/executor/conf
[atguigu@hadoop102 conf]$ vim azkaban.properties
```

2） 按照如下配置修改 azkaban.properties 文件。

```
#Azkaban
#时区
default.timezone.id=Asia/Shanghai

# Azkaban JobTypes Plugins
#jobtype 插件所在位置
azkaban.jobtype.plugin.dir=plugins/jobtypes

#Loader for projects
executor.global.properties=/opt/module/azkaban/executor/conf/global.pro
perties
azkaban.project.dir=projects

database.type=mysql
mysql.port=3306
mysql.host=hadoop102
mysql.database=azkaban
mysql.user=root
mysql.password=000000
mysql.numconnections=100

# Azkaban Executor settings
#最大线程数
executor.maxThreads=50
#端口号(如修改,请与 web 服务中一致)
executor.port=12321
#线程数
executor.flow.threads=30
```

2.6 启动 executor 服务器

在 executor 服务器目录下执行启动命令

```
[atguigu@hadoop102 executor]$ pwd
/opt/module/azkaban/executor
[atguigu@hadoop102 executor]$ bin/azkaban-executor-start.sh
```

2.7 启动 web 服务器

在 azkaban web 服务器目录下执行启动命令

```
[atguigu@hadoop102 server]$ pwd
/opt/module/azkaban/server
[atguigu@hadoop102 server]$ bin/azkaban-web-start.sh
```

注意：

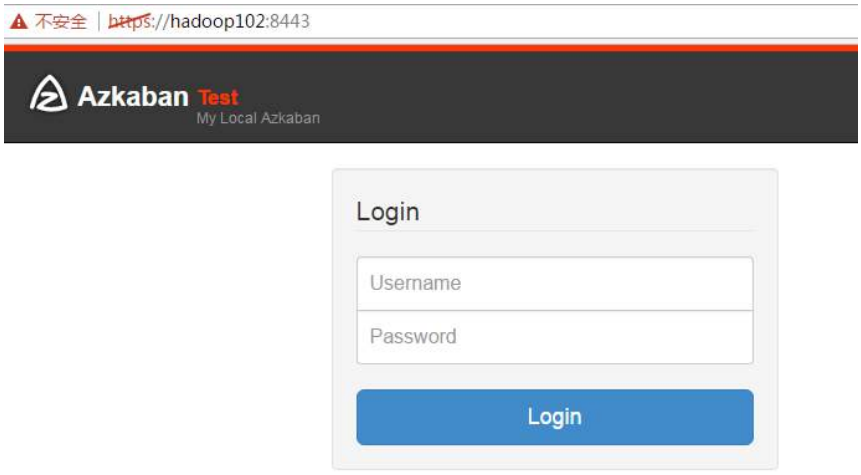
先执行 executor，再执行 web，避免 Web Server 会因为找不到执行器启动失败。

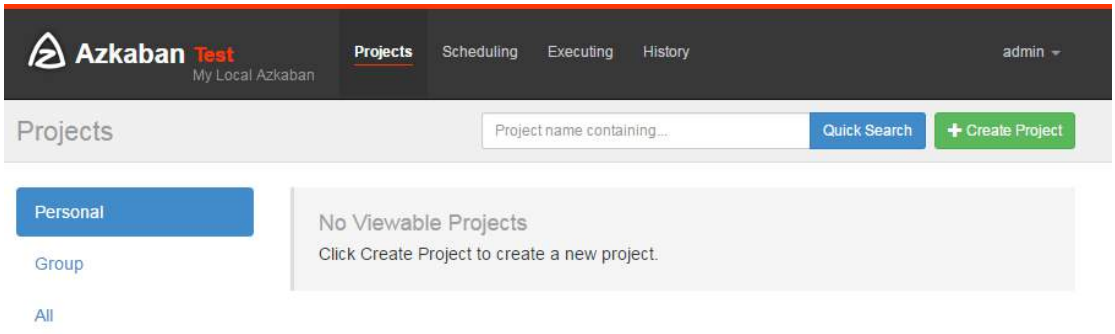
jps 查看进程

```
[atguigu@hadoop102 server]$ jps
3601 AzkabanExecutorServer
5880 Jps
3661 AzkabanWebServer
```

启动完成后，在浏览器(建议使用谷歌浏览器)中输入 **https://服务器 IP 地址:8443**，即可访问 azkaban 服务了。

在登录中输入刚才在 azkaban-users.xml 文件中新添加的户用名及密码，点击 login。





三 Azkaban 实战

Azkaba 内置的任务类型支持 command、java

3.1 单一 job 案例

1）创建 job 描述文件

```
[atguigu@hadoop102 jobs]$ vim first.job
#first.job
type=command
command=echo 'this is my first job'
```

2) 将 job 资源文件打包成 zip 文件

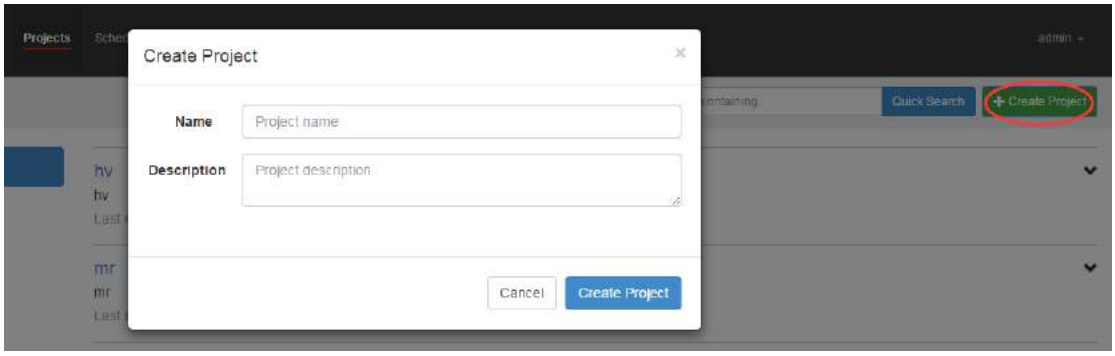
```
[atguigu@hadoop102 jobs]$ zip first.zip first.job
  adding: first.job (deflated 15%)
[atguigu@hadoop102 jobs]$ ll
总用量 8
-rw-rw-r--. 1 atguigu atguigu  60 10 月 18 17:42 first.job
-rw-rw-r--. 1 atguigu atguigu 219 10 月 18 17:43 first.zip
```

注意：

目前，Azkaban 上传的工作流文件只支持 xxx.zip 文件。zip 应包含 xxx.job 运行作业所需的文件和任何文件（文件名后缀必须以.job 结尾，否则无法识别）。作业名称在项目中必须是唯一的。

3）通过 azkaban 的 web 管理平台创建 project 并上传 job 的 zip 包

首先创建 project

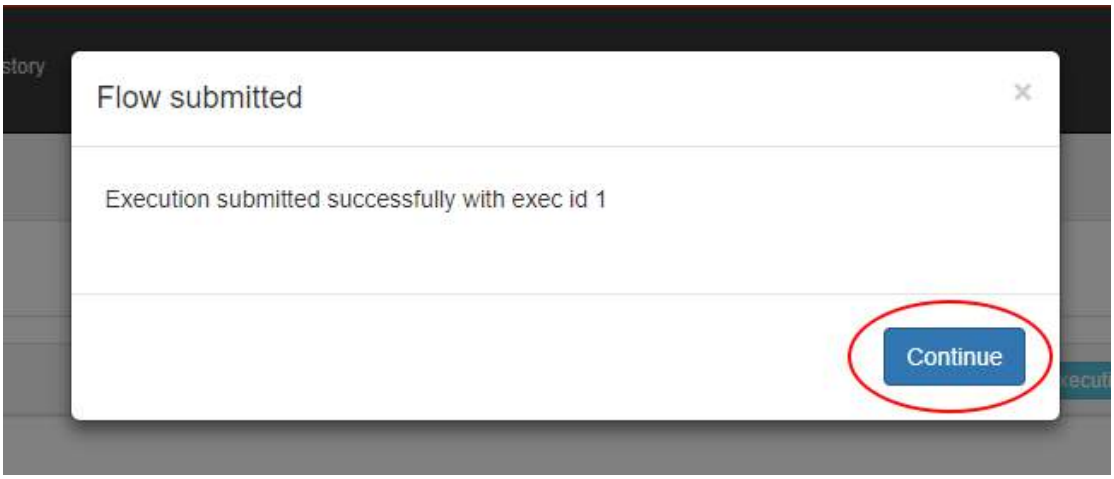


上传 zip 包

4）启动执行该 job

点击执行工作流

点击继续



5）Job 执行成功

6）点击查看 job 日志

3.2 多 job 工作流案例

1）创建有依赖关系的多个 job 描述

第一个 job：start.job

```
[atguigu@hadoop102 jobs]$ vim start.job
#start.job
type=command
command=touch /opt/module/kangkang.txt
```

第二个 job：step1.job 依赖 start.job

```
[atguigu@hadoop102 jobs]$ vim step1.job
#step1.job
type=command
dependencies=start
command=echo "this is step1 job"
```

第三个 job：step2.job 依赖 start.job

```
[atguigu@hadoop102 jobs]$ vim step2.job
#step2.job
type=command
dependencies=start
command=echo "this is step2 job"
```

第四个 job：finish.job 依赖 step1.job 和 step2.job

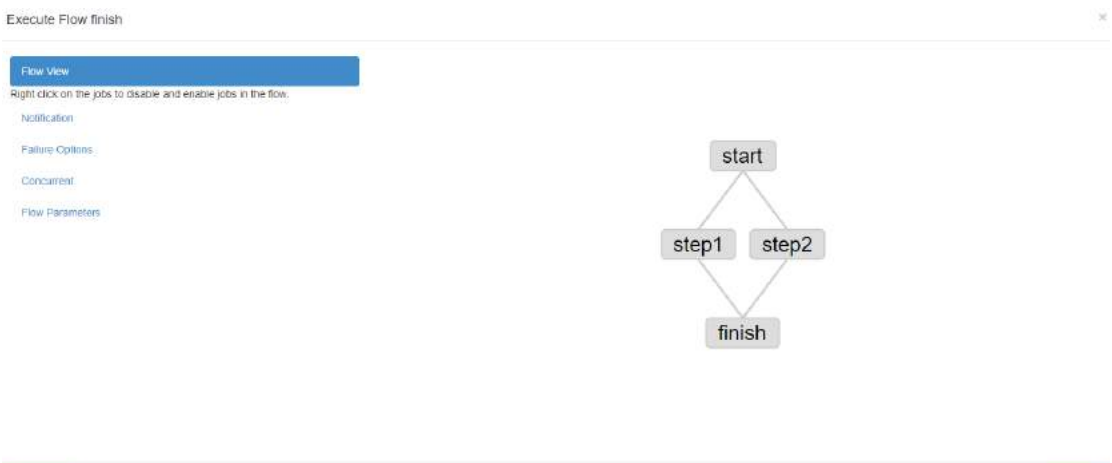
```
[atguigu@hadoop102 jobs]$ vim finish.job
#finish.job
type=command
dependencies=step1,step2
command=echo "this is finish job"
```

2）将所有 job 资源文件打到一个 zip 包中

```
[atguigu@hadoop102 jobs]$ zip jobs.zip start.job step1.job step2.job
finish.job
updating: start.job (deflated 16%)
  adding: step1.job (deflated 12%)
  adding: step2.job (deflated 12%)
  adding: finish.job (deflated 14%)
```

3）在 azkaban 的 web 管理界面创建工程并上传 zip 包

5）启动工作流 flow



6）查看结果

思考：

将 student.txt 文件上传到 hdfs，根据所传文件创建外部表，再将表中查询到的结果写入到本地文件

3.3 java 操作任务

使用 Azkaban 调度 java 程序

1）编写 java 程序

```
import java.io.IOException;

public class AzkabanTest {
  public void run() throws IOException {
      // 根据需求编写具体代码
      FileOutputStream fos = new
FileOutputStream("/opt/module/azkaban/output.txt");
      fos.write("this is a java progress".getBytes());
      fos.close();
    }

  public static void main(String[] args) throws IOException {
      AzkabanTest azkabanTest = new AzkabanTest();
      azkabanTest.run();
  }
}
```

2）将 java 程序打成 jar 包，创建 lib 目录，将 jar 放入 lib 内

```
[atguigu@hadoop102 azkaban]$ mkdir lib
[atguigu@hadoop102 azkaban]$ cd lib/
[atguigu@hadoop102 lib]$ ll
总用量 4
-rw-rw-r--. 1 atguigu atguigu 3355 10 月 18 20:55 azkaban-0.0.1-
SNAPSHOT.jar
```

3）编写 job 文件

```
[atguigu@hadoop102 jobs]$ vim azkabanJava.job
#azkabanJava.job
type=javaprocess
java.class=com.atguigu.azkaban.AzkabanTest
classpath=/opt/module/azkaban/lib/*
```

4）将 job 文件打成 zip 包

```
[atguigu@hadoop102 jobs]$ zip azkabanJava.zip azkabanJava.job
  adding: azkabanJava.job (deflated 19%)
```

5）通过 azkaban 的 web 管理平台创建 project 并上传 job 压缩包，启动执行该 job

```
[atguigu@hadoop102 azkaban]$ pwd
/opt/module/azkaban
[atguigu@hadoop102 azkaban]$ ll
总用量 24
drwxrwxr-x.  2 atguigu atguigu 4096 10 月 17 17:14 azkaban-2.5.0
drwxrwxr-x. 10 atguigu atguigu 4096 10 月 18 17:17 executor
drwxrwxr-x.  2 atguigu atguigu 4096 10 月 18 20:35 jobs
drwxrwxr-x.  2 atguigu atguigu 4096 10 月 18 20:54 lib
-rw-rw-r--.  1 atguigu atguigu   23 10 月 18 20:55 output
drwxrwxr-x.  9 atguigu atguigu 4096 10 月 18 17:17 server
[atguigu@hadoop102 azkaban]$ cat output
this is a java progress
```

3.3 HDFS 操作任务

1）创建 job 描述文件

```
[atguigu@hadoop102 jobs]$ vim fs.job
#hdfs job
type=command
command=/opt/module/hadoop-2.7.2/bin/hadoop fs -mkdir /azkaban
```

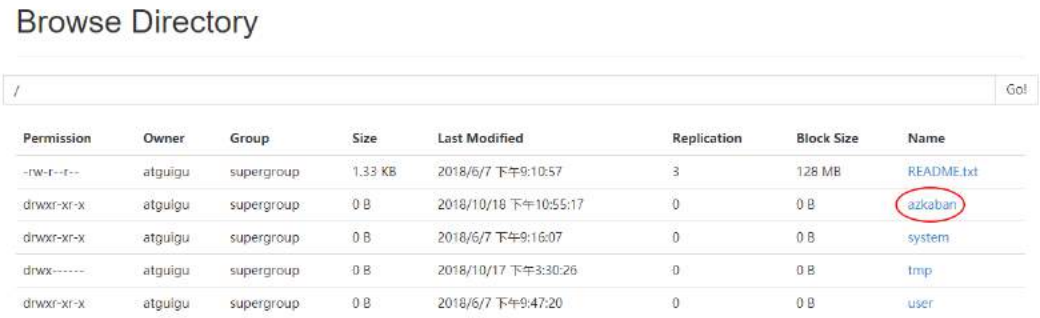
2）将 job 资源文件打包成 zip 文件

```
[atguigu@hadoop102 jobs]$ zip fs.zip fs.job
  adding: fs.job (deflated 12%)
```

3）通过 azkaban 的 web 管理平台创建 project 并上传 job 压缩包

4）启动执行该 job

5）查看结果



3.4 mapreduce 任务

mapreduce 任务依然可以使用 azkaban 进行调度

1） 创建 job 描述文件，及 mr 程序 jar 包

```
[atguigu@hadoop102 jobs]$ vim mapreduce.job
#mapreduce job
type=command
command=/opt/module/hadoop-2.7.2/bin/hadoop jar /opt/module/hadoop-
2.7.2/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar
wordcount /wordcount/input /wordcount/output
```

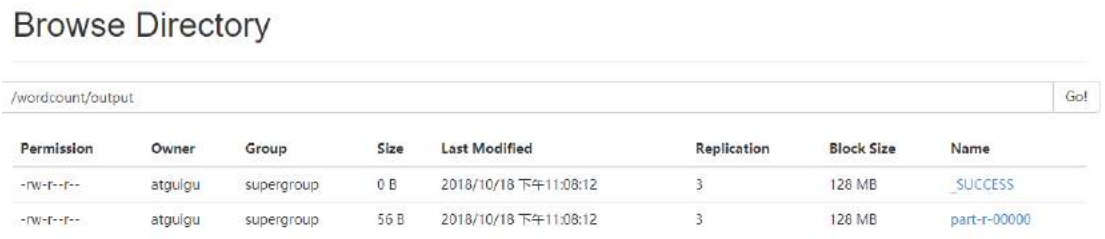
2） 将所有 job 资源文件打到一个 zip 包中

```
[atguigu@hadoop102 jobs]$ zip mapreduce.zip mapreduce.job
  adding: mapreduce.job (deflated 43%)
```

3）在 azkaban 的 web 管理界面创建工程并上传 zip 包

4）启动 job

5）查看结果



3.5 Hive 脚本任务

1）创建 job 描述文件和 hive 脚本

（1）Hive 脚本：student.sql

```
[atguigu@hadoop102 jobs]$ vim student.sql
use default;
drop table student;
create table student(id int, name string)
row format delimited fields terminated by '\t';
```

```
load data local inpath '/opt/module/datas/student.txt' into table
student;
insert overwrite local directory '/opt/module/datas/student'
row format delimited fields terminated by '\t'
select * from student;
```

（2）Job 描述文件：hive.job

```
[atguigu@hadoop102 jobs]$ vim hive.job
#hive job
type=command
command=/opt/module/hive/bin/hive                                    -f
/opt/module/azkaban/jobs/student.sql
```

2） 将所有 job 资源文件打到一个 zip 包中

```
[atguigu@hadoop102 jobs]$ zip hive.zip hive.job
  adding: hive.job (deflated 21%)
```

3）在 azkaban 的 web 管理界面创建工程并上传 zip 包

4）启动 job

5）查看结果

```
[atguigu@hadoop102 student]$ cat /opt/module/datas/student/000000_0
1001    yangyang
1002    bobo
1003    banzhang
1004    pengpeng
```

尚硅谷大数据技术之 Oozie

(作者：尚硅谷大数据研发部)

第 1 章 Oozie 简介

Oozie 英文翻译为：驯象人。一个基于工作流引擎的开源框架，由 Cloudera 公司贡献给 Apache，提供对 Hadoop MapReduce、Pig Jobs 的任务调度与协调。Oozie 需要部署到 Java Servlet 容器中运行。主要用于定时调度任务，多任务可以按照执行的逻辑顺序调度。

第 2 章 Oozie 的功能模块介绍

2.1 模块

1) Workflow

顺序执行流程节点，支持 fork（分支多个节点），join（合并多个节点为一个）

2) Coordinator

定时触发 workflow

3) Bundle Job

绑定多个 Coordinator

2.2 常用节点

1) 控制流节点（Control Flow Nodes）

控制流节点一般都是定义在工作流开始或者结束的位置，比如 start,end,kill 等。以及提供工作流的执行路径机制，如 decision，fork，join 等。

2) 动作节点（Action Nodes）

负责执行具体动作的节点，比如：拷贝文件，执行某个 Shell 脚本等等。

第 3 章 Oozie 的部署

3.1 部署 Hadoop（CDH 版本的）

3.1.2 修改 Hadoop 配置

core-site.xml

```
<!-- Oozie Server 的 Hostname -->
<property>
    <name>hadoop.proxyuser.atguigu.hosts</name>
    <value>*</value>
</property>

<!-- 允许被 Oozie 代理的用户组 -->
<property>
    <name>hadoop.proxyuser.atguigu.groups</name>
    <value>*</value>
</property>
```

mapred-site.xml

```
<!-- 配置 MapReduce JobHistory Server 地址 ，默认端口 10020 -->
<property>
    <name>mapreduce.jobhistory.address</name>
    <value>hadoop102:10020</value>
</property>

<!-- 配置 MapReduce JobHistory Server web ui 地址， 默认端口 19888 -->
<property>
    <name>mapreduce.jobhistory.webapp.address</name>
    <value>hadoop102:19888</value>
</property>
```

yarn-site.xml

```
<!-- 任务历史服务 -->
<property>
    <name>yarn.log.server.url</name>
    <value>http://hadoop102:19888/jobhistory/logs/</value>
</property>
```

完成后：记得 scp 同步到其他机器节点

3.1.3 重启 Hadoop 集群

[atguigu@hadoop102 hadoop-2.7.2]$ sbin/start-dfs.sh

[atguigu@hadoop103 hadoop-2.7.2]$ sbin/start-yarn.sh

[atguigu@hadoop102 hadoop-2.7.2]$ sbin/mr-jobhistory-daemon.sh start historyserver

注意：需要开启 JobHistoryServer，最好执行一个 MR 任务进行测试。

3.2 部署 Oozie

3.2.1 解压 Oozie

[atguigu@hadoop102 software]$ tar -zxvf /opt/software/cdh/oozie-4.0.0-cdh5.3.6.tar.gz -C ./

3.2.2 在 oozie 根目录下解压 oozie-hadooplibs-4.0.0-cdh5.3.6.tar.gz

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ tar -zxvf oozie-hadooplibs-4.0.0-cdh5.3.6.tar.gz -C ../

完成后 Oozie 目录下会出现 hadooplibs 目录。

3.2.3 在 Oozie 目录下创建 libext 目录

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ mkdir libext/

3.2.4 拷贝依赖的 Jar 包

1）将 hadooplibs 里面的 jar 包，拷贝到 libext 目录下：

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ cp -ra hadooplibs/hadooplib-2.5.0-cdh5.3.6.oozie-4.0.0-cdh5.3.6/* libext/

2）拷贝 Mysql 驱动包到 libext 目录下：

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ cp -a /opt/software/mysql-connector-java-5.1.27/mysql-connector-java-5.1.27-bin.jar ./libext/

3.2.5 将 ext-2.2.zip 拷贝到 libext/目录下

ext 是一个 js 框架，用于展示 oozie 前端页面：

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ cp -a /opt/software/cdh/ext-2.2.zip libext/

3.2.6 修改 Oozie 配置文件

oozie-site.xml

属性：oozie.service.JPAService.jdbc.driver

属性值：com.mysql.jdbc.Driver

解释：JDBC 的驱动

属性：oozie.service.JPAService.jdbc.url

属性值：jdbc:mysql://hadoop102:3306/oozie

解释：oozie 所需的数据库地址

属性：oozie.service.JPAService.jdbc.username

属性值：root

解释：数据库用户名

属性：oozie.service.JPAService.jdbc.password

属性值：000000

解释：数据库密码

属性：oozie.service.HadoopAccessorService.hadoop.configurations

属性值：*=/opt/module/cdh/hadoop-2.5.0-cdh5.3.6/etc/hadoop

解释：让 Oozie 引用 Hadoop 的配置文件

3.2.7 在 Mysql 中创建 Oozie 的数据库

进入 Mysql 并创建 oozie 数据库：

```
$ mysql -uroot -p000000
mysql> create database oozie;
```

3.2.8 初始化 Oozie

1) 上传 Oozie 目录下的 yarn.tar.gz 文件到 HDFS：

提示：yarn.tar.gz 文件会自行解压

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ bin/oozie-setup.sh sharelib create -fs hdfs://hadoop102:8020 -locallib oozie-sharelib-4.0.0-cdh5.3.6-yarn.tar.gz

执行成功之后，去 50070 检查对应目录有没有文件生成。

2) 创建 oozie.sql 文件

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ bin/ooziedb.sh create -sqlfile oozie.sql -run

3) 打包项目，生成 war 包

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ bin/oozie-setup.sh prepare-war

3.2.9 Oozie 的启动与关闭

启动命令如下：

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ bin/oozied.sh start

关闭命令如下：

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ bin/oozied.sh stop

3.2.10 访问 Oozie 的 Web 页面

http://hadoop102:11000/oozie

第 4 章 Oozie 的使用

4.1 案例一：Oozie 调度 shell 脚本

目标：使用 Oozie 调度 Shell 脚本

分步实现：

1）解压官方案例模板

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ tar -zxvf oozie-examples.tar.gz

2）创建工作目录

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ mkdir oozie-apps/

3）拷贝任务模板到 oozie-apps/目录

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ cp -r examples/apps/shell/ oozie-apps

4）编写脚本 p1.sh

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ vi oozie-apps/shell/p1.sh

内容如下：

```
#!/bin/bash
/sbin/ifconfig > /opt/module/p1.log
```

5）修改 job.properties 和 workflow.xml 文件

job.properties

```
#HDFS 地址
nameNode=hdfs://hadoop102:8020
#ResourceManager 地址
jobTracker=hadoop103:8032
#队列名称
queueName=default
examplesRoot=oozie-apps
oozie.wf.application.path=${nameNode}/user/${user.name}/${examplesRoot}/shell
EXEC=p1.sh
```

workflow.xml

```
<workflow-app xmlns="uri:oozie:workflow:0.4" name="shell-wf">
<start to="shell-node"/>
<action name="shell-node">
    <shell xmlns="uri:oozie:shell-action:0.2">
        <job-tracker>${jobTracker}</job-tracker>
        <name-node>${nameNode}</name-node>
        <configuration>
            <property>
                <name>mapred.job.queue.name</name>
                <value>${queueName}</value>
            </property>
        </configuration>
        <exec>${EXEC}</exec>
        <!-- <argument>my_output=Hello Oozie</argument> -->
        <file>/user/atguigu/oozie-apps/shell/${EXEC}#${EXEC}</file>
```

```
            <capture-output/>
        </shell>
        <ok to="end"/>
        <error to="fail"/>
</action>
<decision name="check-output">
        <switch>
            <case to="end">
                ${wf:actionData('shell-node')['my_output'] eq 'Hello Oozie'}
            </case>
            <default to="fail-output"/>
        </switch>
</decision>
<kill name="fail">
        <message>Shell action failed, error
message[${wf:errorMessage(wf:lastErrorNode())}]</message>
</kill>
<kill name="fail-output">
        <message>Incorrect output, expected [Hello Oozie] but was
[${wf:actionData('shell-node')['my_output']}]</message>
</kill>
<end name="end"/>
</workflow-app>
```

6）上传任务配置

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ /opt/module/cdh/hadoop-2.5.0-cdh5.3.6/bin/hadoop fs -put oozie-apps/ /user/atguigu

7）执行任务

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ bin/oozie job -oozie http://hadoop101:11000/oozie

-config oozie-apps/shell/job.properties -run

8）杀死某个任务

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ bin/oozie job -oozie http://hadoop101:11000/oozie -kill 0000004-170425105153692-oozie-z-W

4.2 案例二：Oozie 逻辑调度执行多个 Job

目标：使用 Oozie 执行多个 Job 调度

分步执行：

1） 解压官方案例模板

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ tar -zxf oozie-examples.tar.gz

2） 编写脚本

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ vi oozie-apps/shell/p2.sh

内容如下：

```
#!/bin/bash
/bin/date > /opp2.log
```

3）修改 job.properties 和 workflow.xml 文件

job.properties

```
nameNode=hdfs://hadoop102:8020
jobTracker=hadoop103:8032
queueName=default
examplesRoot=oozie-apps

oozie.wf.application.path=${nameNode}/user/${user.name}/${examplesRoot}/shell
EXEC1=p1.sh
EXEC2=p2.sh
```

workflow.xml

```
<workflow-app xmlns="uri:oozie:workflow:0.4" name="shell-wf">
    <start to="p1-shell-node"/>
    <action name="p1-shell-node">
```

```
        <shell xmlns="uri:oozie:shell-action:0.2">
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <configuration>
                <property>
                    <name>mapred.job.queue.name</name>
                    <value>${queueName}</value>
                </property>
            </configuration>
            <exec>${EXEC1}</exec>
            <file>/user/atguigu/oozie-apps/shell/${EXEC1}#${EXEC1}</file>
            <!-- <argument>my_output=Hello Oozie</argument>-->
            <capture-output/>
        </shell>
        <ok to="p2-shell-node"/>
        <error to="fail"/>
    </action>

    <action name="p2-shell-node">
        <shell xmlns="uri:oozie:shell-action:0.2">
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <configuration>
                <property>
                    <name>mapred.job.queue.name</name>
                    <value>${queueName}</value>
                </property>
            </configuration>
            <exec>${EXEC2}</exec>
```

```
            <file>/user/admin/oozie-apps/shell/${EXEC2}#${EXEC2}</file>
            <!-- <argument>my_output=Hello Oozie</argument>-->
            <capture-output/>
        </shell>
        <ok to="end"/>
        <error to="fail"/>
    </action>
    <decision name="check-output">
        <switch>
            <case to="end">
                ${wf:actionData('shell-node')['my_output'] eq 'Hello Oozie'}
            </case>
            <default to="fail-output"/>
        </switch>
    </decision>
    <kill name="fail">
        <message>Shell action failed, error
message[${wf:errorMessage(wf:lastErrorNode())}]</message>
    </kill>
    <kill name="fail-output">
        <message>Incorrect output, expected [Hello Oozie] but was
[${wf:actionData('shell-node')['my_output']}]</message>
    </kill>
    <end name="end"/>
</workflow-app>
```

3） 上传任务配置

```
$ bin/hadoop fs -rmr /user/atguigu/oozie-apps/
$ bin/hadoop fs -put oozie-apps/ /user/atguigu/
```

4） 执行任务

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ bin/oozie job -oozie http://hadoop101:11000/oozie -config oozie-apps/shell/job.properties -run

4.3 案例三：Oozie 调度 MapReduce 任务

目标：使用 Oozie 调度 MapReduce 任务

分步执行：

1）找到一个可以运行的 mapreduce 任务的 jar 包（可以用官方的，也可以是自己写的）

2）拷贝官方模板到 oozie-apps

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ cp -r /opt/module/cdh/oozie-4.0.0-cdh5.3.6/examples/apps/map-reduce/ oozie-apps/

1) 测试一下 wordcount 在 yarn 中的运行

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ /opt/module/cdh/hadoop-2.5.0-cdh5.3.6/bin/yarn jar /opt/module/cdh/hadoop-2.5.0-cdh5.3.6/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.5.0-cdh5.3.6.jar wordcount /input/ /output/

4) 配置 map-reduce 任务的 job.properties 以及 workflow.xml

job.properties

```
nameNode=hdfs://hadoop102:8020
jobTracker=hadoop103:8032
queueName=default
examplesRoot=oozie-apps
#hdfs://hadoop102:8020/user/admin/oozie-apps/map-reduce/workflow.xml
oozie.wf.application.path=${nameNode}/user/${user.name}/${examplesRoot}/map-reduce/workflow.xml
outputDir=map-reduce
```

workflow.xml

```
<workflow-app xmlns="uri:oozie:workflow:0.2" name="map-reduce-wf">
    <start to="mr-node"/>
    <action name="mr-node">
        <map-reduce>
```

```
<job-tracker>${jobTracker}</job-tracker>
<name-node>${nameNode}</name-node>
<prepare>
    <delete path="${nameNode}/output/"/>
</prepare>
<configuration>
    <property>
        <name>mapred.job.queue.name</name>
        <value>${queueName}</value>
    </property>
    <!-- 配置调度 MR 任务时，使用新的 API -->
    <property>
        <name>mapred.mapper.new-api</name>
        <value>true</value>
    </property>

    <property>
        <name>mapred.reducer.new-api</name>
        <value>true</value>
    </property>

    <!-- 指定 Job Key 输出类型 -->
    <property>
        <name>mapreduce.job.output.key.class</name>
        <value>org.apache.hadoop.io.Text</value>
    </property>

    <!-- 指定 Job Value 输出类型 -->
    <property>
```

```
                <name>mapreduce.job.output.value.class</name>
                <value>org.apache.hadoop.io.IntWritable</value>
            </property>

            <!-- 指定输入路径 -->
            <property>
                <name>mapred.input.dir</name>
                <value>/input/</value>
            </property>

            <!-- 指定输出路径 -->
            <property>
                <name>mapred.output.dir</name>
                <value>/output/</value>
            </property>

            <!-- 指定 Map 类 -->
            <property>
                <name>mapreduce.job.map.class</name>
<value>org.apache.hadoop.examples.WordCount$TokenizerMapper</value>
            </property>

            <!-- 指定 Reduce 类 -->
            <property>
                <name>mapreduce.job.reduce.class</name>
<value>org.apache.hadoop.examples.WordCount$IntSumReducer</value>
            </property>
```

```
                    <property>
                        <name>mapred.map.tasks</name>
                        <value>1</value>
                    </property>
                </configuration>
            </map-reduce>
            <ok to="end"/>
            <error to="fail"/>
        </action>
        <kill name="fail">
            <message>Map/Reduce failed, error
message[${wf:errorMessage(wf:lastErrorNode())}]</message>
        </kill>
        <end name="end"/>
</workflow-app>
```

5）拷贝待执行的 jar 包到 map-reduce 的 lib 目录下

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ cp -a /opt/module/cdh/hadoop-2.5.0-cdh5.3.6/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.5.0-cdh5.3.6.jar oozie-apps/map-reduce/lib

6）上传配置好的 app 文件夹到 HDFS

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ /opt/module/cdh/hadoop-2.5.0-cdh5.3.6/bin/hdfs dfs -put oozie-apps/map-reduce/ /user/admin/oozie-apps

7）执行任务

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ bin/oozie job -oozie http://hadoop102:11000/oozie -config oozie-apps/map-reduce/job.properties -run

4.4 案例四：Oozie 定时任务/循环任务

目标：Coordinator 周期性调度任务

分步实现：

1） 配置 Linux 时区以及时间服务器

2） 检查系统当前时区：

date -R

注意：如果显示的时区不是+0800，删除 localtime 文件夹后，再关联一个正确时区的链接过去，命令如下：

```
# rm -rf /etc/localtime
# ln -s /usr/share/zoneinfo/Asia/Shanghai /etc/localtime
```

同步时间：

```
# ntpdate pool.ntp.org
```

修改 NTP 配置文件：

```
# vi /etc/ntp.conf
去掉下面这行前面的# ,并把网段修改成自己的网段：
restrict 192.168.122.0 mask 255.255.255.0 nomodify notrap
注释掉以下几行：
#server 0.centos.pool.ntp.org
#server 1.centos.pool.ntp.org
#server 2.centos.pool.ntp.org
把下面两行前面的#号去掉,如果没有这两行内容,需要手动添加
server    127.127.1.0       # local clock
fudge    127.127.1.0 stratum 10
```

重启 NTP 服务：

```
# systemctl start ntpd.service，
注意，如果是 centOS7 以下的版本，使用命令：service ntpd start
# systemctl enable ntpd.service，
注意，如果是 centOS7 以下的版本，使用命令：chkconfig ntpd on
```

集群其他节点去同步这台时间服务器时间：

```
首先需要关闭这两台计算机的 ntp 服务
```

```
# systemctl stop ntpd.service，
centOS7 以下，则：service ntpd stop
# systemctl disable ntpd.service，
centOS7 以下，则：chkconfig ntpd off
# systemctl status ntpd，查看 ntp 服务状态
# pgrep ntpd，查看 ntp 服务进程 id
同步第一台服务器 linux01 的时间：
# ntpdate linux01
```

使用 root 用户制定计划任务,周期性同步时间：

```
# crontab -e
*/10 * * * * /usr/sbin/ntpdate hadoop102
```

重启定时任务：

```
# systemctl restart crond.service，
centOS7 以下使用：service crond restart，
```

其他台机器的配置同理。

3）配置 oozie-site.xml 文件

```
属性：oozie.processing.timezone
属性值：GMT+0800
解释：修改时区为东八区区时
```

注：该属性去 oozie-default.xml 中找到即可

4）修改 js 框架中的关于时间设置的代码

```
$ vi /opt/module/cdh/oozie-4.0.0-cdh5.3.6/oozie-server/webapps/oozie/oozie-console.js
修改如下：
function getTimeZone() {
    Ext.state.Manager.setProvider(new Ext.state.CookieProvider());
    return Ext.state.Manager.get("TimezoneId","GMT+0800");
}
```

5）重启 oozie 服务，并重启浏览器（一定要注意清除缓存）

```
[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ bin/oozied.sh stop
```

[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ bin/oozied.sh start

6）拷贝官方模板配置定时任务\

$ cp -r examples/apps/cron/ oozie-apps/

7）修改模板 job.properties 和 coordinator.xml 以及 workflow.xml

job.properties

```
nameNode=hdfs://hadoop102:8020
jobTracker=hadoop103:8032
queueName=default
examplesRoot=oozie-apps

oozie.coord.application.path=${nameNode}/user/${user.name}/${examplesRoot}/cron
#start：必须设置为未来时间，否则任务失败
start=2017-07-29T17:00+0800
end=2017-07-30T17:00+0800
workflowAppUri=${nameNode}/user/${user.name}/${examplesRoot}/cron

EXEC3=p3.sh
```

coordinator.xml

```
<coordinator-app name="cron-coord" frequency="${coord:minutes(5)}" start="${start}"
end="${end}" timezone="GMT+0800" xmlns="uri:oozie:coordinator:0.2">
<action>
    <workflow>
        <app-path>${workflowAppUri}</app-path>
        <configuration>
            <property>
                <name>jobTracker</name>
                <value>${jobTracker}</value>
            </property>
            <property>
```

```
                    <name>nameNode</name>
                    <value>${nameNode}</value>
                </property>
                <property>
                    <name>queueName</name>
                    <value>${queueName}</value>
                </property>
            </configuration>
        </workflow>
</action>
</coordinator-app>
```

workflow.xml

```
<workflow-app xmlns="uri:oozie:workflow:0.5" name="one-op-wf">
<start to="p3-shell-node"/>
   <action name="p3-shell-node">
        <shell xmlns="uri:oozie:shell-action:0.2">
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <configuration>
                <property>
                    <name>mapred.job.queue.name</name>
                    <value>${queueName}</value>
                </property>
            </configuration>
            <exec>${EXEC3}</exec>
            <file>/user/atguigu/oozie-apps/cron/${EXEC3}#${EXEC3}</file>
            <!-- <argument>my_output=Hello Oozie</argument>-->
            <capture-output/>
        </shell>
```

```
        <ok to="end"/>
        <error to="fail"/>
    </action>
<kill name="fail">
      <message>Shell action failed, error
message[${wf:errorMessage(wf:lastErrorNode())}]</message>
</kill>
<kill name="fail-output">
      <message>Incorrect output, expected [Hello Oozie] but was
[${wf:actionData('shell-node')['my_output']}]</message>
</kill>
<end name="end"/>
</workflow-app>
```

8）上传配置

```
[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ /opt/module/cdh/hadoop-2.5.0-cdh5.3.6/bin/hdfs dfs -put oozie-apps/cron/ /user/admin/oozie-apps
```

9）启动任务

```
[atguigu@hadoop102 oozie-4.0.0-cdh5.3.6]$ bin/oozie job -oozie http://hadoop102:11000/oozie -config oozie-apps/cron/job.properties -run
```

注意：oozie 允许的最小执行任务的频率是 5 分钟

第 5 章 常见问题总结

1）Mysql 权限配置

授权所有主机可以使用 root 用户操作所有数据库和数据表

```
mysql> grant all on *.* to root@'%' identified by '000000';
mysql> flush privileges;
mysql> exit;
```

2）workflow.xml 配置的时候不要忽略 file 属性

3）jps 查看进程时，注意有没有 bootstrap

4）关闭 oozie

如果 bin/oozied.sh stop 无法关闭，则可以使用 kill -9 [pid]，之后 oozie-server/temp/xxx.pid 文件一定要删除。

5）Oozie 重新打包时，一定要注意先关闭进程，删除对应文件夹下面的 pid 文件。（可以参考第 4 条目）

6）配置文件一定要生效

起始标签和结束标签无对应则不生效，配置文件的属性写错了，那么则执行默认的属性。

7）libext 下边的 jar 存放于某个文件夹中，导致 share/lib 创建不成功。

8）调度任务时，找不到指定的脚本，可能是 oozie-site.xml 里面的 Hadoop 配置文件没有关联上。

9）修改 Hadoop 配置文件，需要重启集群。一定要记得 scp 到其他节点。

10）JobHistoryServer 必须开启，集群要重启的。

11）Mysql 配置如果没有生效的话，默认使用 derby 数据库。

12）在本地修改完成的 job 配置，必须重新上传到 HDFS。

13）将 HDFS 中上传的 oozie 配置文件下载下来查看是否有错误。

14）Linux 用户名和 Hadoop 的用户名不一致。

尚硅谷大数据技术之 Sqoop

(作者：尚硅谷大数据研发部)

版本：V1.1

第 1 章 Sqoop 简介

Sqoop 是一款开源的工具，主要用于在 Hadoop(Hive)与传统的数据库(mysql、postgresql...)间进行数据的传递，可以将一个关系型数据库（例如 ： MySQL ,Oracle ,Postgres 等）中的数据导进到 Hadoop 的 HDFS 中，也可以将 HDFS 的数据导进到关系型数据库中。

Sqoop 项目开始于 2009 年，最早是作为 Hadoop 的一个第三方模块存在，后来为了让使用者能够快速部署，也为了让开发人员能够更快速的迭代开发，Sqoop 独立成为一个 Apache 项目。

Sqoop2 的最新版本是 1.99.7。请注意，2 与 1 不兼容，且特征不完整，它并不打算用于生产部署。

第 2 章 Sqoop 原理

将导入或导出命令翻译成 mapreduce 程序来实现。

在翻译出的 mapreduce 中主要是对 inputformat 和 outputformat 进行定制。

第 3 章 Sqoop 安装

安装 Sqoop 的前提是已经具备 Java 和 Hadoop 的环境。

3.1 下载并解压

1) 下载地址：http://mirrors.hust.edu.cn/apache/sqoop/1.4.6/

2) 上传安装包 sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz 到虚拟机中

3) 解压 sqoop 安装包到指定目录，如：

$ tar -zxf sqoop-1.4.6.bin__hadoop-2.0.4-alpha.tar.gz -C /opt/module/

3.2 修改配置文件

Sqoop 的配置文件与大多数大数据框架类似，在 sqoop 根目录下的 conf 目录中。

1) 重命名配置文件

```
$ mv sqoop-env-template.sh sqoop-env.sh
```

2) 修改配置文件

sqoop-env.sh

```
export HADOOP_COMMON_HOME=/opt/module/hadoop-2.7.2
export HADOOP_MAPRED_HOME=/opt/module/hadoop-2.7.2
export HIVE_HOME=/opt/module/hive
export ZOOKEEPER_HOME=/opt/module/zookeeper-3.4.10
export ZOOCFGDIR=/opt/module/zookeeper-3.4.10
export HBASE_HOME=/opt/module/hbase
```

3.3 拷贝 JDBC 驱动

拷贝 jdbc 驱动到 sqoop 的 lib 目录下，如：

```
$ cp mysql-connector-java-5.1.27-bin.jar
/opt/module/sqoop-1.4.6.bin__hadoop-2.0.4-alpha/lib/
```

3.4 验证 Sqoop

我们可以通过某一个 command 来验证 sqoop 配置是否正确：

```
$ bin/sqoop help
```

出现一些 Warning 警告（警告信息已省略），并伴随着帮助命令的输出：

```
Available commands:
  codegen            Generate code to interact with database records
  create-hive-table     Import a table definition into Hive
  eval              Evaluate a SQL statement and display the results
  export            Export an HDFS directory to a database table
  help              List available commands
  import            Import a table from a database to HDFS
  import-all-tables     Import tables from a database to HDFS
  import-mainframe    Import datasets from a mainframe server to HDFS
  job               Work with saved jobs
  list-databases        List available databases on a server
  list-tables          List available tables in a database
  merge             Merge results of incremental imports
  metastore           Run a standalone Sqoop metastore
  version           Display version information
```

3.5 测试 Sqoop 是否能够成功连接数据库

```
$ bin/sqoop list-databases --connect jdbc:mysql://hadoop102:3306/
--username root --password 000000
```

出现如下输出：

```
information_schema
metastore
mysql
oozie
performance_schema
```

第 4 章 Sqoop 的简单使用案例

4.1 导入数据

在 Sqoop 中，“导入”概念指：从非大数据集群（RDBMS）向大数据集群（HDFS，HIVE，HBASE）中传输数据，叫做：导入，即使用 import 关键字。

4.1.1 RDBMS 到 HDFS

1) 确定 Mysql 服务开启正常

2) 在 Mysql 中新建一张表并插入一些数据

```
$ mysql -uroot -p000000
mysql> create database company;
mysql> create table company.staff(id int(4) primary key not null auto_increment, name varchar(255), sex varchar(255));
mysql> insert into company.staff(name, sex) values('Thomas', 'Male');
mysql> insert into company.staff(name, sex) values('Catalina', 'FeMale');
```

3) 导入数据

（1）全部导入

```
$ bin/sqoop import \
--connect jdbc:mysql://hadoop102:3306/company \
--username root \
--password 000000 \
--table staff \
--target-dir /user/company \
--delete-target-dir \
--num-mappers 1 \
--fields-terminated-by "\t"
```

（2）查询导入

```
$ bin/sqoop import \
--connect jdbc:mysql://hadoop102:3306/company \
--username root \
--password 000000 \
--target-dir /user/company \
--delete-target-dir \
--num-mappers 1 \
--fields-terminated-by "\t" \
--query 'select name,sex from staff where id <=1 and $CONDITIONS;'
```

提示：must contain '$CONDITIONS' in WHERE clause.

如果 query 后使用的是双引号，则$CONDITIONS 前必须加转移符，防止 shell 识别为自己的变量。

（3）导入指定列

```
$ bin/sqoop import \
--connect jdbc:mysql://hadoop102:3306/company \
```

```
--username root \
--password 000000 \
--target-dir /user/company \
--delete-target-dir \
--num-mappers 1 \
--fields-terminated-by "\t" \
--columns id,sex \
--table staff
```

提示：columns 中如果涉及到多列，用逗号分隔，分隔时不要添加空格

（4）使用 sqoop 关键字筛选查询导入数据

```
$ bin/sqoop import \
--connect jdbc:mysql://hadoop102:3306/company \
--username root \
--password 000000 \
--target-dir /user/company \
--delete-target-dir \
--num-mappers 1 \
--fields-terminated-by "\t" \
--table staff \
--where "id=1"
```

4.1.2 RDBMS 到 Hive

```
$ bin/sqoop import \
--connect jdbc:mysql://hadoop102:3306/company \
--username root \
--password 000000 \
--table staff \
--num-mappers 1 \
--hive-import \
--fields-terminated-by "\t" \
--hive-overwrite \
--hive-table staff_hive
```

提示：该过程分为两步，第一步将数据导入到 HDFS，第二步将导入到 HDFS 的数据迁移到 Hive 仓库，第一步默认的临时目录是/user/atguigu/表名

4.1.3 RDBMS 到 Hbase

```
$ bin/sqoop import \
--connect jdbc:mysql://hadoop102:3306/company \
--username root \
--password 000000 \
--table company \
--columns "id,name,sex" \
--column-family "info" \
--hbase-create-table \
--hbase-row-key "id" \
--hbase-table "hbase_company" \
--num-mappers 1 \
--split-by id
```

提示：sqoop1.4.6 只支持 HBase1.0.1 之前的版本的自动创建 HBase 表的功能

解决方案：手动创建 HBase 表

```
hbase> create 'hbase_company,'info'
```

(5) 在 HBase 中 scan 这张表得到如下内容

```
hbase> scan ‘hbase_company’
```

4.2、导出数据

在 Sqoop 中，“导出”概念指：从大数据集群（HDFS，HIVE，HBASE）向非大数据集群（RDBMS）中传输数据，叫做：导出，即使用 export 关键字。

4.2.1 HIVE/HDFS 到 RDBMS

```
$ bin/sqoop export \
--connect jdbc:mysql://hadoop102:3306/company \
--username root \
--password 000000 \
--table staff \
--num-mappers 1 \
--export-dir /user/hive/warehouse/staff_hive \
--input-fields-terminated-by "\t"
```

提示：Mysql 中如果表不存在，不会自动创建

4.3 脚本打包

使用 opt 格式的文件打包 sqoop 命令，然后执行

1) 创建一个.opt 文件

```
$ mkdir opt
$ touch opt/job_HDFS2RDBMS.opt
```

2) 编写 sqoop 脚本

```
$ vi opt/job_HDFS2RDBMS.opt

export
--connect
jdbc:mysql://hadoop102:3306/company
--username
root
--password
000000
--table
staff
--num-mappers
1
--export-dir
/user/hive/warehouse/staff_hive
--input-fields-terminated-by
"\t"
```

3) 执行该脚本

```
$ bin/sqoop --options-file opt/job_HDFS2RDBMS.opt
```

第 5 章 Sqoop 一些常用命令及参数

5.1 常用命令列举

这里给大家列出来了一部分 Sqoop 操作时的常用参数，以供参考，需要深入学习的可以参看对应类的源代码。

序号	命令	类	说明
1	import	ImportTool	将数据导入到集群
2	export	ExportTool	将集群数据导出
3	codegen	CodeGenTool	获取数据库中某张表数据生成 Java 并打包 Jar
4	create-hive-table	CreateHiveTableTool	创建 Hive 表
5	eval	EvalSqlTool	查看 SQL 执行结果
6	import-all-tables	ImportAllTablesTool	导入某个数据库下所有表到 HDFS 中
7	job	JobTool	用来生成一个 sqoop 的任务，生成后，该任务并不执行，除非使用命令执行该任务。
8	list-databases	ListDatabasesTool	列出所有数据库名
9	list-tables	ListTablesTool	列出某个数据库下所有表
10	merge	MergeTool	将 HDFS 中不同目录下面的数据合在一起，并存放在指定的目录中
11	metastore	MetastoreTool	记录 sqoop job 的元

			数据信息，如果不启动 metastore 实例，则默认的元数据存储目录为：~/.sqoop，如果要更改存储目录，可以在配置文件 sqoop-site.xml 中进行更改。
12	help	HelpTool	打印 sqoop 帮助信息
13	version	VersionTool	打印 sqoop 版本信息

5.2 命令&参数详解

刚才列举了一些 Sqoop 的常用命令，对于不同的命令，有不同的参数，让我们来一一列举说明。

首先来我们来介绍一下公用的参数，所谓公用参数，就是大多数命令都支持的参数。

5.2.1 公用参数：数据库连接

序号	参数	说明
1	--connect	连接关系型数据库的 URL
2	--connection-manager	指定要使用的连接管理类
3	--driver	Hadoop 根目录
4	--help	打印帮助信息
5	--password	连接数据库的密码
6	--username	连接数据库的用户名
7	--verbose	在控制台打印出详细信息

5.2.2 公用参数：import

序号	参数	说明
1	--enclosed-by <char>	给字段值前加上指定的字符
2	--escaped-by <char>	对字段中的双引号加转义符

3	--fields-terminated-by <char>	设定每个字段是以什么符号作为结束，默认为逗号
4	--lines-terminated-by <char>	设定每行记录之间的分隔符，默认是\n
5	--mysql-delimiters	Mysql 默认的分隔符设置，字段之间以逗号分隔，行之间以\n 分隔，默认转义符是\，字段值以单引号包裹。
6	--optionally-enclosed-by <char>	给带有双引号或单引号的字段值前后加上指定字符。

5.2.3 公用参数：export

序号	参数	说明
1	--input-enclosed-by <char>	对字段值前后加上指定字符
2	--input-escaped-by <char>	对含有转移符的字段做转义处理
3	--input-fields-terminated-by <char>	字段之间的分隔符
4	--input-lines-terminated-by <char>	行之间的分隔符
5	--input-optionally-enclosed-by <char>	给带有双引号或单引号的字段前后加上指定字符

5.2.4 公用参数：hive

序号	参数	说明
1	--hive-delims-replacement <arg>	用自定义的字符串替换掉数据中的\r\n 和\013 \010 等字符
2	--hive-drop-import-delims	在导入数据到 hive 时，去掉数据中的\r\n\013\010 这样的字符

3	--map-column-hive <arg>	生成 hive 表时，可以更改生成字段的数据类型
4	--hive-partition-key	创建分区，后面直接跟分区名，分区字段的默认类型为 string
5	--hive-partition-value <v>	导入数据时，指定某个分区的值
6	--hive-home <dir>	hive 的安装目录，可以通过该参数覆盖之前默认配置的目录
7	--hive-import	将数据从关系数据库中导入到 hive 表中
8	--hive-overwrite	覆盖掉在 hive 表中已经存在的数据
9	--create-hive-table	默认是 false，即，如果目标表已经存在了，那么创建任务失败。
10	--hive-table	后面接要创建的 hive 表,默认使用 MySQL 的表名
11	--table	指定关系数据库的表名

公用参数介绍完之后，我们来按照命令介绍命令对应的特有参数。

5.2.5 命令&参数：import

将关系型数据库中的数据导入到 HDFS（包括 Hive，HBase）中，如果导入的是 Hive，那么当 Hive 中没有对应表时，则自动创建。

1) 命令：

如：导入数据到 hive 中

```
$ bin/sqoop import \
```

```
--connect jdbc:mysql://hadoop102:3306/company \
--username root \
--password 000000 \
--table staff \
--hive-import
```

如：增量导入数据到 hive 中，mode=append

```
append 导入：
$ bin/sqoop import \
--connect jdbc:mysql://hadoop102:3306/company \
--username root \
--password 000000 \
--table staff \
--num-mappers 1 \
--fields-terminated-by "\t" \
--target-dir /user/hive/warehouse/staff_hive \
--check-column id \
--incremental append \
--last-value 3
```

尖叫提示：append 不能与--hive-等参数同时使用（Append mode for hive imports is not yet supported. Please remove the parameter --append-mode）

如：增量导入数据到 hdfs 中，mode=lastmodified

```
先在 mysql 中建表并插入几条数据：
mysql> create table company.staff_timestamp(id int(4), name varchar(255), sex varchar(255),
last_modified timestamp DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP);
mysql> insert into company.staff_timestamp (id, name, sex) values(1, 'AAA', 'female');
mysql> insert into company.staff_timestamp (id, name, sex) values(2, 'BBB', 'female');
```

先导入一部分数据：

```
$ bin/sqoop import \
--connect jdbc:mysql://hadoop102:3306/company \
--username root \
--password 000000 \
--table staff_timestamp \
--delete-target-dir \
--m 1
```

再增量导入一部分数据：

```
mysql> insert into company.staff_timestamp (id, name, sex) values(3, 'CCC', 'female');
$ bin/sqoop import \
--connect jdbc:mysql://hadoop102:3306/company \
--username root \
--password 000000 \
--table staff_timestamp \
--check-column last_modified \
--incremental lastmodified \
--last-value "2017-09-28 22:20:38" \
--m 1 \
--append
```

尖叫提示：使用 lastmodified 方式导入数据要指定增量数据是要--append（追加）还是要--merge-key（合并）

尖叫提示：last-value 指定的值是会包含于增量导入的数据中

2) 参数：

序号	参数	说明
1	--append	将数据追加到 HDFS 中已经存在的 DataSet 中，如果使用该参数，sqoop 会把数据先导入到临时文件目录，再合并。

2	--as-avrodatafile	将数据导入到一个 Avro 数据文件中
3	--as-sequencefile	将数据导入到一个 sequence 文件中
4	--as-textfile	将数据导入到一个普通文本文件中
5	--boundary-query <statement>	边界查询，导入的数据为该参数的值（一条 sql 语句）所执行的结果区间内的数据。
6	--columns <col1, col2, col3>	指定要导入的字段
7	--direct	直接导入模式，使用的是关系数据库自带的导入导出工具，以便加快导入导出过程。
8	--direct-split-size	在使用上面 direct 直接导入的基础上，对导入的流按字节分块，即达到该阈值就产生一个新的文件
9	--inline-lob-limit	设定大对象数据类型的最大值
10	--m 或–num-mappers	启动 N 个 map 来并行导入数据，默认 4 个。
11	--query 或--e <statement>	将查询结果的数据导入，使用时必须伴随参--target-dir，--hive-table，如果查询中有 where 条件，则条件后必须加上$CONDITIONS 关键字
12	--split-by <column-name>	按照某一列来切分表的工作单元，不能与

		--autoreset-to-one-mapper 连用（请参考官方文档）
13	--table <table-name>	关系数据库的表名
14	--target-dir <dir>	指定 HDFS 路径
15	--warehouse-dir <dir>	与 14 参数不能同时使用，导入数据到 HDFS 时指定的目录
16	--where	从关系数据库导入数据时的查询条件
17	--z 或--compress	允许压缩
18	--compression-codec	指定 hadoop 压缩编码类，默认为 gzip(Use Hadoop codec default gzip)
19	--null-string <null-string>	string 类型的列如果 null，替换为指定字符串
20	--null-non-string <null-string>	非 string 类型的列如果 null，替换为指定字符串
21	--check-column <col>	作为增量导入判断的列名
22	--incremental <mode>	mode：append 或 lastmodified
23	--last-value <value>	指定某一个值，用于标记增量导入的位置

5.2.6 命令&参数：export

从 HDFS（包括 Hive 和 HBase）中奖数据导出到关系型数据库中。

1) 命令：

如：

```
$ bin/sqoop export \
--connect jdbc:mysql://hadoop102:3306/company \
--username root \
```

```
--password 000000 \
--table staff \
--export-dir /user/company \
--input-fields-terminated-by "\t" \
--num-mappers 1
```

2) 参数：

序号	参数	说明
1	--direct	利用数据库自带的导入导出工具，以便于提高效率
2	--export-dir <dir>	存放数据的 HDFS 的源目录
3	-m 或--num-mappers <n>	启动 N 个 map 来并行导入数据，默认 4 个
4	--table <table-name>	指定导出到哪个 RDBMS 中的表
5	--update-key <col-name>	对某一列的字段进行更新操作
6	--update-mode <mode>	updateonly allowinsert(默认)
7	--input-null-string <null-string>	请参考 import 该类似参数说明
8	--input-null-non-string <null-string>	请参考 import 该类似参数说明
9	--staging-table <staging-table-name>	创建一张临时表，用于存放所有事务的结果，然后将所有事务结果一次性导入到目标表中，防止错误。
10	--clear-staging-table	如果第 9 个参数非空，则可以

		在导出操作执行前，清空临时事务结果表

5.2.7 命令&参数：codegen

将关系型数据库中的表映射为一个 Java 类，在该类中有各列对应的各个字段。

如：

```
$ bin/sqoop codegen \
--connect jdbc:mysql://hadoop102:3306/company \
--username root \
--password 000000 \
--table staff \
--bindir /home/admin/Desktop/staff \
--class-name Staff \
--fields-terminated-by "\t"
```

序号	参数	说明
1	--bindir <dir>	指定生成的 Java 文件、编译成的 class 文件及将生成文件打包为 jar 的文件输出路径
2	--class-name <name>	设定生成的 Java 文件指定的名称
3	--outdir <dir>	生成 Java 文件存放的路径
4	--package-name <name>	包名，如 com.z，就会生成 com 和 z 两级目录
5	--input-null-non-string <null-str>	在生成的 Java 文件中，可以将 null 字符串或者不存在的字符串设置为想要设定的值（例如空字符串）

6	--input-null-string <null-str>	将 null 字符串替换成想要替换的值（一般与 5 同时使用）
7	--map-column-java <arg>	数据库字段在生成的 Java 文件中会映射成各种属性，且默认的数据类型与数据库类型保持对应关系。该参数可以改变默认类型，例如：--map-column-java id=long, name=String
8	--null-non-string <null-str>	在生成 Java 文件时，可以将不存在或者 null 的字符串设置为其他值
9	--null-string <null-str>	在生成 Java 文件时，将 null 字符串设置为其他值（一般与 8 同时使用）
10	--table <table-name>	对应关系数据库中的表名，生成的 Java 文件中的各个属性与该表的各个字段一一对应

5.2.8 命令&参数：create-hive-table

生成与关系数据库表结构对应的 hive 表结构。

命令：

如：

```
$ bin/sqoop create-hive-table \
--connect jdbc:mysql://hadoop102:3306/company \
--username root \
--password 000000 \
--table staff \
--hive-table hive_staff
```

参数：

序号	参数	说明
1	--hive-home <dir>	Hive 的安装目录，可以通过该参数覆盖掉默认的 Hive 目录
2	--hive-overwrite	覆盖掉在 Hive 表中已经存在的数据
3	--create-hive-table	默认是 false，如果目标表已经存在了，那么创建任务会失败
4	--hive-table	后面接要创建的 hive 表
5	--table	指定关系数据库的表名

5.2.9 命令&参数：eval

可以快速的使用 SQL 语句对关系型数据库进行操作，经常用于在 import 数据之前，了解一下 SQL 语句是否正确，数据是否正常，并可以将结果显示在控制台。

命令：

如：

```
$ bin/sqoop eval \
--connect jdbc:mysql://hadoop102:3306/company \
--username root \
--password 000000 \
--query "SELECT * FROM staff"
```

参数：

序号	参数	说明
1	--query 或--e	后跟查询的 SQL 语句

5.2.10 命令&参数：import-all-tables

可以将 RDBMS 中的所有表导入到 HDFS 中，每一个表都对应一个 HDFS 目录

命令：

如：

```
$ bin/sqoop import-all-tables \
--connect jdbc:mysql://hadoop102:3306/company \
--username root \
--password 000000 \
--warehouse-dir /all_tables
```

参数：

序号	参数	说明
1	--as-avrodatafile	这些参数的含义均和 import 对应的含义一致
2	--as-sequencefile	
3	--as-textfile	
4	--direct	
5	--direct-split-size <n>	
6	--inline-lob-limit <n>	
7	--m 或—num-mappers <n>	
8	--warehouse-dir <dir>	
9	-z 或--compress	
10	--compression-codec	

5.2.11 命令&参数：job

用来生成一个 sqoop 任务，生成后不会立即执行，需要手动执行。

命令：

如：

```
$ bin/sqoop job \
 --create myjob -- import-all-tables \
 --connect jdbc:mysql://hadoop102:3306/company \
```

```
 --username root \
 --password 000000
$ bin/sqoop job \
--list
$ bin/sqoop job \
--exec myjob
```

尖叫提示：注意 import-all-tables 和它左边的--之间有一个空格

尖叫提示：如果需要连接 metastore，则--meta-connect jdbc:hsqldb:hsql://linux01:16000/sqoop

参数：

序号	参数	说明
1	--create <job-id>	创建 job 参数
2	--delete <job-id>	删除一个 job
3	--exec <job-id>	执行一个 job
4	--help	显示 job 帮助
5	--list	显示 job 列表
6	--meta-connect <jdbc-uri>	用来连接 metastore 服务
7	--show <job-id>	显示一个 job 的信息
8	--verbose	打印命令运行时的详细信息

尖叫提示：在执行一个 job 时，如果需要手动输入数据库密码，可以做如下优化

```
<property>
    <name>sqoop.metastore.client.record.password</name>
    <value>true</value>
    <description>If true, allow saved passwords in the metastore.</description>
</property>
```

5.2.12 命令&参数：list-databases

命令：

如：

```
$ bin/sqoop list-databases \
--connect jdbc:mysql://hadoop102:3306/ \
--username root \
--password 000000
```

参数：与公用参数一样

5.2.13 命令&参数：list-tables

命令：

如：

```
$ bin/sqoop list-tables \
--connect jdbc:mysql://hadoop102:3306/company \
--username root \
--password 000000
```

参数：与公用参数一样

5.2.14 命令&参数：merge

将 HDFS 中不同目录下面的数据合并在一起并放入指定目录中

数据环境：

```
new_staff
1        AAA      male
2        BBB      male
3        CCC      male
4        DDD      male
old_staff
1        AAA      female
2        CCC      female
3        BBB      female
6        DDD      female
```

尖叫提示：上边数据的列之间的分隔符应该为\t，行与行之间的分割符为\n，如果直接复制，

请检查之。

命令：

如：

创建 JavaBean：

```
$ bin/sqoop codegen \
--connect jdbc:mysql://hadoop102:3306/company \
--username root \
--password 000000 \
--table staff \
--bindir /home/admin/Desktop/staff \
--class-name Staff \
--fields-terminated-by "\t"
```

开始合并：

```
$ bin/sqoop merge \
--new-data /test/new/ \
--onto /test/old/ \
--target-dir /test/merged \
--jar-file /home/admin/Desktop/staff/Staff.jar \
--class-name Staff \
--merge-key id
```

结果：

```
1    AAA    MALE
2    BBB    MALE
3    CCC    MALE
4    DDD    MALE
6    DDD    FEMALE
```

参数：

序号	参数	说明
1	--new-data <path>	HDFS 待合并的数据目录，合并后在新的数据集中保留
2	--onto <path>	HDFS 合并后，重复的部分在新的数据集中被覆盖
3	--merge-key <col>	合并键，一般是主键 ID
4	--jar-file <file>	合并时引入的 jar 包，该 jar 包是通过 Codegen 工具生成的 jar 包
5	--class-name <class>	对应的表名或对象名，该 class 类是包含在 jar 包中的
6	--target-dir <path>	合并后的数据在 HDFS 里存放的目录

5.2.15 命令&参数：metastore

记录了 Sqoop job 的元数据信息，如果不启动该服务，那么默认 job 元数据的存储目录为~/.sqoop，可在 sqoop-site.xml 中修改。

命令：

如：启动 sqoop 的 metastore 服务

```
$ bin/sqoop metastore
```

参数：

序号	参数	说明
1	--shutdown	关闭 metastore