# 课程大纲（辅助系统）

|  |  |  |
| --- | --- | --- |
| 离线辅助系统 | 数据接入 | Flume介绍 |
| Flume组件 |
| Flume实战案例 |
| 任务调度 | 调度器基础 |
| 市面上调度工具 |
| Oozie的使用 |
| Oozie的流程定义详解 |
| 数据导出 | sqoop基础知识 |
| sqoop实战及原理 |
| Sqoop数据导入实战 |
| Sqoop数据导出实战 |
| Sqoop作业操作 |
| Sqoop的原理 |

学习目标：

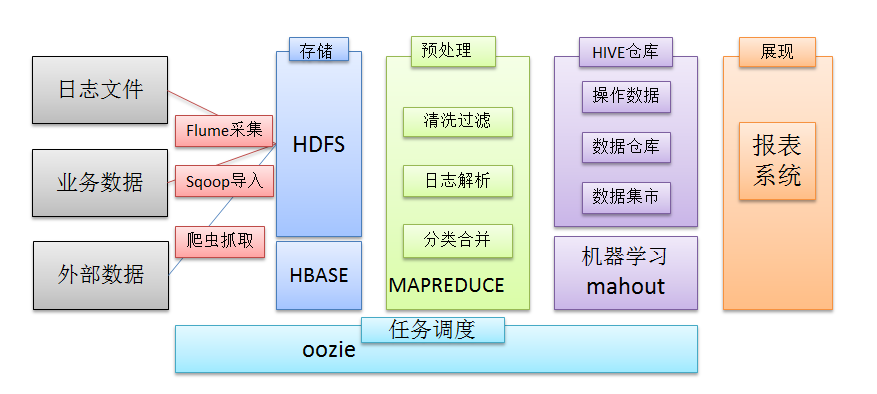
1、理解flume、sqoop、oozie的应用场景

2、理解flume、sqoop、oozie的基本原理

3、掌握flume、sqoop、oozie的使用方法

# 前言

在一个完整的大数据处理系统中，除了hdfs+mapreduce+hive组成分析系统的核心之外，还需要数据采集、结果数据导出、任务调度等不可或缺的辅助系统，而这些辅助工具在hadoop生态体系中都有便捷的开源框架，如图所示：



# 1. 日志采集框架Flume

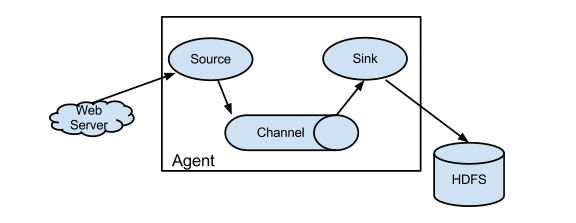
## 1.1 Flume介绍

### 1.1.1 概述

* Flume是一个分布式、可靠、和高可用的海量日志采集、聚合和传输的系统。
* Flume可以采集文件，socket数据包等各种形式源数据，又可以将采集到的数据输出到HDFS、hbase、hive、kafka等众多外部存储系统中
* 一般的采集需求，通过对flume的简单配置即可实现
* Flume针对特殊场景也具备良好的自定义扩展能力，因此，flume可以适用于大部分的日常数据采集场景

### 1.1.2 运行机制

1. Flume分布式系统中最**核心的角色是agent**，flume采集系统就是由一个个agent所连接起来形成
2. **每一个agent相当于一个数据传递员，内部有三个组件：**
   * 1. Source：采集源，用于跟数据源对接，以获取数据
     2. Sink：下沉地，采集数据的传送目的，用于往下一级agent传递数据或者往最终存储系统传递数据
     3. Channel：angent内部的数据传输通道，用于从source将数据传递到sink

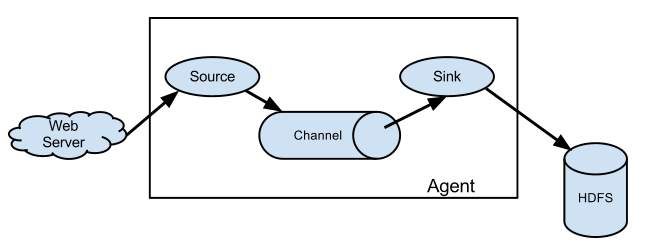


**官方封装了很多种Source，可以在官网去查看flume.apache.org**

### 1.1.4 Flume采集系统结构图

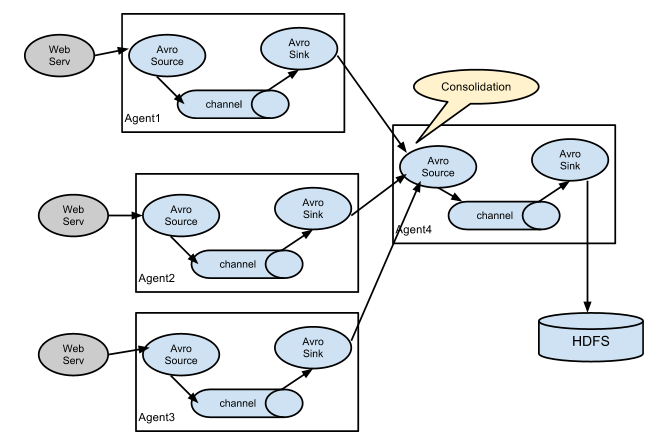
#### 1. 简单结构

单个agent采集数据



#### 2. 复杂结构

多级agent之间串联



## 1.2 Flume实战案例

#### 1.2.1 Flume的安装部署

1. Flume的安装非常简单，只需要解压即可，当然，前提是已有hadoop环境

上传安装包到数据源所在节点上

然后解压 tar -zxvf apache-flume-1.6.0-bin.tar.gz

然后进入flume的目录，修改conf下的flume-env.sh，在里面配置JAVA\_HOME

2、根据数据采集的需求**配置采集方案**，描述在配置文件中(文件名可任意自定义)

3、**指定采集方案配置文件**，在相应的节点上启动flume agent

先用一个最简单的例子来测试一下程序环境是否正常

1. 先在flume的conf目录下新建一个文件

vi netcat-logger.conf

|  |
| --- |
| # 定义这个agent中各组件的名字  a1.sources = r1  a1.sinks = k1  a1.channels = c1  # 描述和配置source组件：r1  a1.sources.r1.type = netcat  a1.sources.r1.bind = localhost  a1.sources.r1.port = 44444  # 描述和配置sink组件：k1  a1.sinks.k1.type = logger  # 描述和配置channel组件，此处使用是内存缓存的方式、1000条数据（event）、  a1.channels.c1.type = memory  a1.channels.c1.capacity = 1000  a1.channels.c1.transactionCapacity = 100  # 描述和配置source channel sink之间的连接关系  a1.sources.r1.channels = c1  a1.sinks.k1.channel = c1 |

1. 启动agent去采集数据

|  |
| --- |
| bin/flume-ng agent -c conf -f conf/netcat-logger.conf -n a1 -Dflume.root.logger=INFO,console |

-c conf 指定flume自身的配置文件所在目录

-f conf/netcat-logger.con 指定我们所描述的采集方案

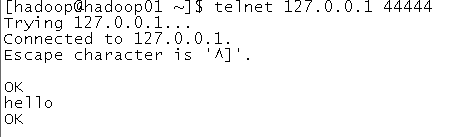
-n a1 指定我们这个agent的名字

1. 测试

先要往agent采集监听的端口上发送数据，让agent有数据可采

随便在一个能跟agent节点联网的机器上

telnet anget-hostname port （telnet localhost 44444）



采集文件夹

一行一个event，采集后重命名，加上complete

# gent中各组件的名字

a1.sources = r1

a1.sinks = k1

a1.channels = c1

# 描述和配置source组件：r1

a1.sources.r1.type = spooldir

a1.sources.r1.spoolDir = /home/hadoop/apps/testData //监控的目录

a1.sources.r1.fileHeader = true

# 描述和配置sink组件：k1

a1.sinks.k1.type = logger

# 描述和配置channel组件，此处使用是内存缓存的方式

a1.channels.c1.type = memory

a1.channels.c1.capacity = 1000

a1.channels.c1.transactionCapacity = 100

# 描述和配置source channel sink之间的连接关系

a1.sources.r1.channels = c1

a1.sinks.k1.channel = c1

#### 1.2.2 采集案例

##### 1、采集目录到HDFS

采集需求：某服务器的某特定目录下，会不断产生新的文件，每当有新文件出现，就需要把文件采集到HDFS中去

根据需求，首先定义以下3大要素

* 采集源，即source——监控文件目录 : spooldir
* 下沉目标，即sink——HDFS文件系统 : hdfs sink
* source和sink之间的传递通道——channel，可用file channel 也可以用内存channel

配置文件编写：

|  |
| --- |
| #定义三大组件的名称  agent1.sources = source1  agent1.sinks = sink1  agent1.channels = channel1  # 配置source组件  agent1.sources.source1.type = spooldir  agent1.sources.source1.spoolDir = /home/hadoop/logs/  agent1.sources.source1.fileHeader = false  #配置拦截器  agent1.sources.source1.interceptors = i1  agent1.sources.source1.interceptors.i1.type = host  agent1.sources.source1.interceptors.i1.hostHeader = hostname  # 配置sink组件  agent1.sinks.sink1.type = hdfs  agent1.sinks.sink1.hdfs.path =hdfs://hdp-node-01:9000/weblog/flume-collection/%y-%m-%d/%H-%M  agent1.sinks.sink1.hdfs.filePrefix = access\_log  agent1.sinks.sink1.hdfs.maxOpenFiles = 5000  agent1.sinks.sink1.hdfs.batchSize= 100  agent1.sinks.sink1.hdfs.fileType = DataStream  agent1.sinks.sink1.hdfs.writeFormat =Text  agent1.sinks.sink1.hdfs.rollSize = 102400  agent1.sinks.sink1.hdfs.rollCount = 1000000  agent1.sinks.sink1.hdfs.rollInterval = 60  #agent1.sinks.sink1.hdfs.round = true  #agent1.sinks.sink1.hdfs.roundValue = 10  #agent1.sinks.sink1.hdfs.roundUnit = minute  agent1.sinks.sink1.hdfs.useLocalTimeStamp = true  # Use a channel which buffers events in memory  agent1.channels.channel1.type = memory  agent1.channels.channel1.keep-alive = 120  agent1.channels.channel1.capacity = 500000  agent1.channels.channel1.transactionCapacity = 600  # Bind the source and sink to the channel  agent1.sources.source1.channels = channel1  agent1.sinks.sink1.channel = channel1 |

Channel参数解释：

capacity：默认该通道中最大的可以存储的event数量

trasactionCapacity：每次最大可以从source中拿到或者送到sink中的event数量

keep-alive：event添加到通道中或者移出的允许时间

##### 2、采集文件到HDFS

采集需求：比如业务系统使用log4j生成的日志，日志内容不断增加，需要把追加到日志文件中的数据实时采集到hdfs

根据需求，首先定义以下3大要素

* 采集源，即source——监控文件内容更新 : exec ‘tail -F file’
* 下沉目标，即sink——HDFS文件系统 : hdfs sink
* Source和sink之间的传递通道——channel，可用file channel 也可以用内存channel

配置文件编写：

|  |
| --- |
| agent1.sources = source1  agent1.sinks = sink1  agent1.channels = channel1  # Describe/configure tail -F source1  agent1.sources.source1.type = exec  agent1.sources.source1.command = tail -F /home/hadoop/logs/access\_log  agent1.sources.source1.channels = channel1  #configure host for source  agent1.sources.source1.interceptors = i1  agent1.sources.source1.interceptors.i1.type = host  agent1.sources.source1.interceptors.i1.hostHeader = hostname  # Describe sink1  agent1.sinks.sink1.type = hdfs  #a1.sinks.k1.channel = c1  agent1.sinks.sink1.hdfs.path =hdfs://hdp-node-01:9000/weblog/flume-collection/%y-%m-%d/%H-%M  agent1.sinks.sink1.hdfs.filePrefix = access\_log  agent1.sinks.sink1.hdfs.maxOpenFiles = 5000  agent1.sinks.sink1.hdfs.batchSize= 100  agent1.sinks.sink1.hdfs.fileType = DataStream  agent1.sinks.sink1.hdfs.writeFormat =Text  agent1.sinks.sink1.hdfs.rollSize = 102400  agent1.sinks.sink1.hdfs.rollCount = 1000000  agent1.sinks.sink1.hdfs.rollInterval = 60  agent1.sinks.sink1.hdfs.round = true  agent1.sinks.sink1.hdfs.roundValue = 10  agent1.sinks.sink1.hdfs.roundUnit = minute  agent1.sinks.sink1.hdfs.useLocalTimeStamp = true  # Use a channel which buffers events in memory  agent1.channels.channel1.type = memory  agent1.channels.channel1.keep-alive = 120  agent1.channels.channel1.capacity = 500000  agent1.channels.channel1.transactionCapacity = 600  # Bind the source and sink to the channel  agent1.sources.source1.channels = channel1  agent1.sinks.sink1.channel = channel1 |

tail-hdfs.conf

用tail命令获取数据，下沉到hdfs

mkdir /home/hadoop/log

while true

do

echo 111111 >> /home/hadoop/log/test.log

sleep 0.5

done

tail -F test.log

采集到hdfs中, 文件中的目录不用自己建的

检查下hdfs式否是salf模式:

hdfs dfsadmin -report

bin/flume-ng agent -c conf -f conf/tail-hdfs.conf -n a1

前端页面查看下, master:50070, 文件目录: /flum/events/16-04-20/

启动命令：

bin/flume-ng agent -c conf -f conf/tail-hdfs.conf -n a1

################################################################

# Name the components on this agent

a1.sources = r1

a1.sinks = k1

a1.channels = c1

#exec 指的是命令

# Describe/configure the source

a1.sources.r1.type = exec

#F根据文件名追中, f根据文件的nodeid追中

a1.sources.r1.command = tail -F /home/hadoop/log/test.log

a1.sources.r1.channels = c1

# Describe the sink

#下沉目标

a1.sinks.k1.type = hdfs

a1.sinks.k1.channel = c1

#指定目录, flum帮做目的替换

a1.sinks.k1.hdfs.path = /flume/events/%y-%m-%d/%H%M/

#文件的命名, 前缀

a1.sinks.k1.hdfs.filePrefix = events-

#10 分钟就改目录

a1.sinks.k1.hdfs.round = true

a1.sinks.k1.hdfs.roundValue = 10

a1.sinks.k1.hdfs.roundUnit = minute

#文件滚动之前的等待时间(秒)

a1.sinks.k1.hdfs.rollInterval = 3

#文件滚动的大小限制(bytes)

a1.sinks.k1.hdfs.rollSize = 500

#写入多少个event数据后滚动文件(事件个数)

a1.sinks.k1.hdfs.rollCount = 20

#5个事件就往里面写入

a1.sinks.k1.hdfs.batchSize = 5

#用本地时间格式化目录

a1.sinks.k1.hdfs.useLocalTimeStamp = true

#下沉后, 生成的文件类型，默认是Sequencefile，可用DataStream，则为普通文本

a1.sinks.k1.hdfs.fileType = DataStream

# Use a channel which buffers events in memory

a1.channels.c1.type = memory

a1.channels.c1.capacity = 1000

a1.channels.c1.transactionCapacity = 100

# Bind the source and sink to the channel

a1.sources.r1.channels = c1

a1.sinks.k1.channel = c1

一个串联的Flume

从tail命令获取数据发送到avro端口

另一个节点可配置一个avro源来中继数据，发送外部存储

##################

# Name the components on this agent

a1.sources = r1

a1.sinks = k1

a1.channels = c1

# Describe/configure the source

a1.sources.r1.type = exec

a1.sources.r1.command = tail -F /home/hadoop/log/test.log

a1.sources.r1.channels = c1

# Describe the sink

#绑定的不是本机, 是另外一台机器的服务地址, sink端的avro是一个发送端, avro的客户端, 往hadoop01这个机器上发

a1.sinks = k1

a1.sinks.k1.type = avro

a1.sinks.k1.channel = c1

a1.sinks.k1.hostname = master

a1.sinks.k1.port = 4141

a1.sinks.k1.batch-size = 2

# Use a channel which buffers events in memory

a1.channels.c1.type = memory

a1.channels.c1.capacity = 1000

a1.channels.c1.transactionCapacity = 100

# Bind the source and sink to the channel

a1.sources.r1.channels = c1

a1.sinks.k1.channel = c1

下一端的agent

从avro端口接收数据，下沉到logger

bin/flume-ng agent -c conf -f conf/avro-logger.conf -n al -Dflume.root.logger=INFO,console

#########

采集配置文件，avro-hdfs.conf

# Name the components on this agent

a1.sources = r1

a1.sinks = k1

a1.channels = c1

# Describe/configure the source

#source中的avro组件是接收者服务, 绑定本机

a1.sources.r1.type = avro

a1.sources.r1.channels = c1

a1.sources.r1.bind = 0.0.0.0

a1.sources.r1.port = 4141

# Describe the sink

a1.sinks.k1.type = logger

# Use a channel which buffers events in memory

a1.channels.c1.type = memory

a1.channels.c1.capacity = 1000

a1.channels.c1.transactionCapacity = 100

# Bind the source and sink to the channel

a1.sources.r1.channels = c1

a1.sinks.k1.channel = c1

发送数据：

$ bin/flume-ng avro-client -H localhost -p 4141 -F /usr/logs/log.10

## 1.3 更多source和sink组件

Flume支持众多的source和sink类型，详细手册可参考官方文档

<http://flume.apache.org/FlumeUserGuide.html>

# 2. 工作流调度器azkaban

## 2.1 概述

### 2.1.1为什么需要工作流调度系统

* 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

* 各任务单元之间存在时间先后及前后依赖关系
* 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先将原始数据同步到HDFS上；
2. 借助MapReduce计算框架对原始数据进行转换，生成的数据以分区表的形式存储到多张Hive表中；
3. 需要对Hive中多个表的数据进行JOIN处理，得到一个明细数据Hive大表；
4. 将明细数据进行复杂的统计分析，得到结果报表信息；
5. 需要将统计分析得到的结果数据同步到业务系统中，供业务调用使用。

### 2.1.2 工作流调度实现方式

简单的任务调度：直接使用linux的crontab来定义；

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

### 2.1.3 常见工作流调度系统

市面上目前有许多工作流调度器

在hadoop领域，常见的工作流调度器有Oozie, Azkaban,Cascading,Hamake等

### 2.1.4 各种调度工具特性对比

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 | Hamake | Oozie | Azkaban | Cascading |
| 工作流描述语言 | XML | XML (xPDL based) | text file with key/value pairs | Java API |
| 依赖机制 | data-driven | explicit | explicit | explicit |
| 是否要web容器 | No | Yes | Yes | No |
| 进度跟踪 | console/log messages | web page | web page | Java API |
| Hadoop job调度支持 | no | yes | yes | yes |
| 运行模式 | command line utility | daemon | daemon | API |
| Pig支持 | yes | yes | yes | yes |
| 事件通知 | no | no | no | yes |
| 需要安装 | no | yes | yes | no |
| 支持的hadoop版本 | 0.18+ | 0.20+ | currently unknown | 0.18+ |
| 重试支持 | no | workflownode evel | yes | yes |
| 运行任意命令 | yes | yes | yes | yes |
| Amazon EMR支持 | yes | no | currently unknown | yes |

### 2.1.5 Azkaban与Oozie对比

对市面上最流行的两种调度器，给出以下详细对比，以供技术选型参考。总体来说，ooize相比azkaban是一个重量级的任务调度系统，功能全面，但配置使用也更复杂。如果可以不在意某些功能的缺失，轻量级调度器azkaban是很不错的候选对象。

详情如下：

* 功能

两者均可以调度mapreduce,pig,java,脚本工作流任务

两者均可以定时执行工作流任务

* 工作流定义

Azkaban使用Properties文件定义工作流

Oozie使用XML文件定义工作流

* 工作流传参

Azkaban支持直接传参，例如${input}

Oozie支持参数和EL表达式，例如${fs:dirSize(myInputDir)}

* 定时执行

Azkaban的定时执行任务是基于时间的

Oozie的定时执行任务基于时间和输入数据

* 资源管理

Azkaban有较严格的权限控制，如用户对工作流进行读/写/执行等操作

Oozie暂无严格的权限控制

* 工作流执行

Azkaban有两种运行模式，分别是solo server mode(executor server和web server部署在同一台节点)和multi server mode(executor server和web server可以部署在不同节点)

Oozie作为工作流服务器运行，支持多用户和多工作流

* 工作流管理

Azkaban支持浏览器以及ajax方式操作工作流

Oozie支持命令行、HTTP REST、Java API、浏览器操作工作流

## 2.2 Azkaban介绍

Azkaban是由Linkedin开源的一个批量工作流任务调度器。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。Azkaban定义了一种KV文件格式来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。

它有如下功能特点：

* Web用户界面
* 方便上传工作流
* 方便设置任务之间的关系
* 调度工作流
* 认证/授权(权限的工作)
* 能够杀死并重新启动工作流
* 模块化和可插拔的插件机制
* 项目工作区
* 工作流和任务的日志记录和审计

## 2. 3Azkaban安装部署

### 准备工作

Azkaban Web服务器

azkaban-web-server-2.5.0.tar.gz

Azkaban执行服务器

azkaban-executor-server-2.5.0.tar.gz

MySQL

目前azkaban只支持 mysql,需安装mysql服务器,本文档中默认已安装好mysql服务器,并建立了 root用户,密码 root.

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

### 安装

将安装文件上传到集群,最好上传到安装 hive、sqoop的机器上,方便命令的执行

在当前用户目录下新建 azkabantools目录,用于存放源安装文件.新建azkaban目录,用于存放azkaban运行程序

### azkaban web服务器安装

解压azkaban-web-server-2.5.0.tar.gz

命令: tar –zxvf azkaban-web-server-2.5.0.tar.gz

将解压后的azkaban-web-server-2.5.0 移动到 azkaban目录中,并重新命名 webserver

命令: mv azkaban-web-server-2.5.0 ../azkaban

        cd ../azkaban

        mv azkaban-web-server-2.5.0 server

### azkaban 执行服器安装

解压azkaban-executor-server-2.5.0.tar.gz

命令:tar –zxvf azkaban-executor-server-2.5.0.tar.gz

将解压后的azkaban-executor-server-2.5.0 移动到 azkaban目录中,并重新命名 executor

命令:mv azkaban-executor-server-2.5.0  ../azkaban

cd ../azkaban

mv azkaban-executor-server-2.5.0  executor

azkaban脚本导入

解压: azkaban-sql-script-2.5.0.tar.gz

命令:tar –zxvf azkaban-sql-script-2.5.0.tar.gz

将解压后的mysql 脚本,导入到mysql中:

进入mysql

mysql> create database azkaban;

mysql> use azkaban;

Database changed

mysql> source /home/hadoop/azkaban-2.5.0/create-all-sql-2.5.0.sql;

### 创建SSL配置

参考地址: http://docs.codehaus.org/display/JETTY/How+to+configure+SSL

命令: keytool -keystore keystore -alias jetty -genkey -keyalg RSA

运行此命令后,会提示输入当前生成 keystor的密码及相应信息,输入的密码请劳记,信息如下:

输入keystore密码：

再次输入新密码:

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]：

您所在的州或省份名称是什么？

  [Unknown]：

该单位的两字母国家代码是什么

  [Unknown]：  CN

CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=CN 正确吗？

  [否]：  y

输入<jetty>的主密码

        （如果和 keystore 密码相同，按回车）：

再次输入新密码:

完成上述工作后,将在当前目录生成 keystore 证书文件,将keystore 考贝到 azkaban web服务器根目录中.如:cp keystore azkaban/webserver

### 配置文件

注：先配置好服务器节点上的时区

1. 先生成时区配置文件Asia/Shanghai，用交互式命令tzselect即可
2. 拷贝该时区文件，覆盖系统本地时区配置

cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime

azkaban web服务器配置

进入azkaban web服务器安装目录 conf目录

* 修改azkaban.properties文件

命令vi azkaban.properties

内容说明如下:

|  |
| --- |
| #Azkaban Personalization Settings  azkaban.name=Test                           #服务器UI名称,用于服务器上方显示的名字  azkaban.label=My Local Azkaban                               #描述  azkaban.color=#FF3601                                                 #UI颜色  azkaban.default.servlet.path=/index                         #  web.resource.dir=web/                                                 #默认根web目录  default.timezone.id=Asia/Shanghai                           #默认时区,已改为亚洲/上海 默认为美国    #Azkaban UserManager class  user.manager.class=azkaban.user.XmlUserManager   #用户权限管理默认类  user.manager.xml.file=conf/azkaban-users.xml              #用户配置,具体配置参加下文    #Loader for projects  executor.global.properties=conf/global.properties    # global配置文件所在位置  azkaban.project.dir=projects                                                #    database.type=mysql                                                              #数据库类型  mysql.port=3306                                                                       #端口号  mysql.host=hadoop03                                                   #数据库连接IP  mysql.database=azkaban                                                       #数据库实例名  mysql.user=root                                                                 #数据库用户名  mysql.password=root                                                          #数据库密码  mysql.numconnections=100                                                  #最大连接数    # Velocity dev mode  velocity.dev.mode=false  # Jetty服务器属性.  jetty.maxThreads=25                                                               #最大线程数  jetty.ssl.port=8443                                                                   #Jetty SSL端口  jetty.port=8081                                                                         #Jetty端口  jetty.keystore=keystore                                                          #SSL文件名  jetty.password=123456                                                             #SSL文件密码  jetty.keypassword=123456                                                      #Jetty主密码 与 keystore文件相同  jetty.truststore=keystore                                                                #SSL文件名  jetty.trustpassword=123456                                                   # SSL文件密码    # 执行服务器属性  executor.port=12321                                                               #执行服务器端口    # 邮件设置  mail.sender=xxxxxxxx@163.com                                       #发送邮箱  mail.host=smtp.163.com                                                       #发送邮箱smtp地址  mail.user=xxxxxxxx                                       #发送邮件时显示的名称  mail.password=\*\*\*\*\*\*\*\*\*\*                                                 #邮箱密码  job.failure.email=xxxxxxxx@163.com                              #任务失败时发送邮件的地址  job.success.email=xxxxxxxx@163.com                            #任务成功时发送邮件的地址  lockdown.create.projects=false                                           #  cache.directory=cache                                                            #缓存目录 |

* azkaban 执行服务器配置

进入执行服务器安装目录conf,修改azkaban.properties

vi azkaban.properties

|  |
| --- |
| #Azkaban  default.timezone.id=Asia/Shanghai                                              #时区    # Azkaban JobTypes 插件配置  azkaban.jobtype.plugin.dir=plugins/jobtypes                   #jobtype 插件所在位置    #Loader for projects  executor.global.properties=conf/global.properties  azkaban.project.dir=projects    #数据库设置  database.type=mysql                                                                       #数据库类型(目前只支持mysql)  mysql.port=3306                                                                                #数据库端口号  mysql.host=192.168.20.200                                                           #数据库IP地址  mysql.database=azkaban                                                                #数据库实例名  mysql.user=azkaban                                                                         #数据库用户名  mysql.password=oracle                                                                   #数据库密码  mysql.numconnections=100                                                           #最大连接数    # 执行服务器配置  executor.maxThreads=50                                                                #最大线程数  executor.port=12321                                                               #端口号(如修改,请与web服务中一致)  executor.flow.threads=30                                                                #线程数 |

* 用户配置

进入azkaban web服务器conf目录,修改azkaban-users.xml

vi azkaban-users.xml 增加 管理员用户

|  |
| --- |
| <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> |

### 启动

#### web服务器

在azkaban web服务器目录下执行启动命令

bin/azkaban-web-start.sh

注:在web服务器根目录运行

或者启动到后台

nohup bin/azkaban-web-start.sh 1>/tmp/azstd.out 2>/tmp/azerr.out &

#### 执行服务器

在执行服务器目录下执行启动命令

bin/azkaban-executor-start.sh ./

注:只能要执行服务器根目录运行

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.在登录中输入刚才新的户用名及密码,点击 login.

## 2.4Azkaban实战

Azkaba内置的任务类型支持command、java

### Command类型单一job示例

1. 创建job描述文件，注意要utf-8

vi command.job

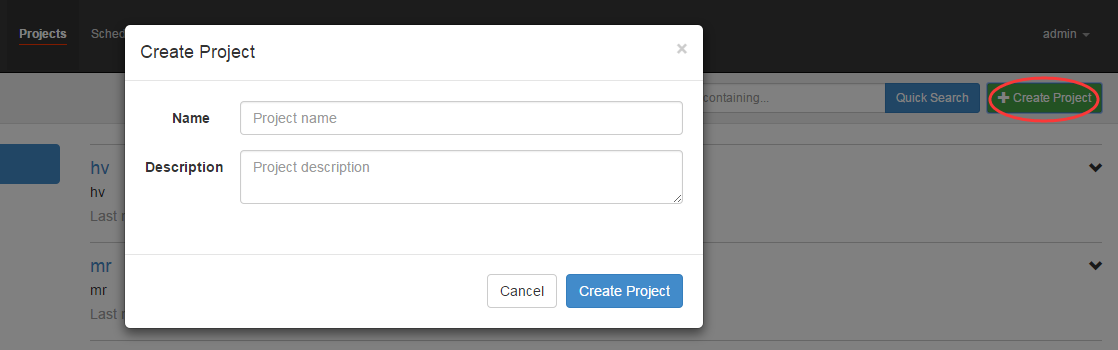
|  |
| --- |
| #command.job  type=command  command=echo 'hello' |

1. 将job资源文件打包成zip文件

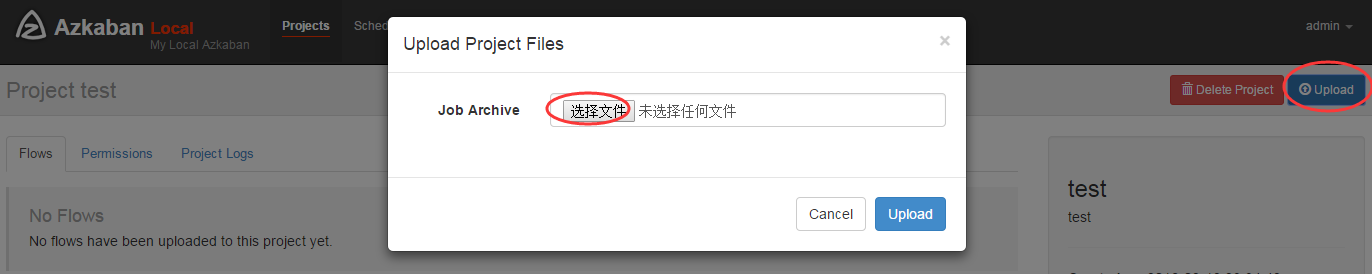
zip command.job

1. 通过azkaban的web管理平台创建project并上传job压缩包

首先创建project



上传zip包



4、启动执行该job



### Command类型多job工作流flow

1. 创建有依赖关系的多个job描述

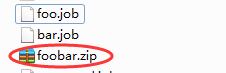
第一个job：foo.job

|  |
| --- |
| # foo.job  type=command  command=echo foo |

第二个job：bar.job依赖foo.job

|  |
| --- |
| # bar.job  type=command  dependencies=foo  command=echo bar |

1. 将所有job资源文件打到一个zip包中



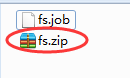
1. 在azkaban的web管理界面创建工程并上传zip包
2. 启动工作流flow

### HDFS操作任务

1. 创建job描述文件

|  |
| --- |
| # fs.job  type=command  command=/home/hadoop/apps/hadoop-2.6.1/bin/hadoop fs -mkdir /azaz |

1. 将job资源文件打包成zip文件



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

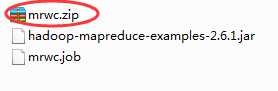
### MAPREDUCE任务

Mr任务依然可以使用command的job类型来执行

1. 创建job描述文件，及mr程序jar包（示例中直接使用hadoop自带的example jar）

|  |
| --- |
| # mrwc.job  type=command  command=/home/hadoop/apps/hadoop-2.6.1/bin/hadoop jar hadoop-mapreduce-examples-2.6.1.jar wordcount /wordcount/input /wordcount/azout |

1. 将所有job资源文件打到一个zip包中



3、在azkaban的web管理界面创建工程并上传zip包

4、启动job

### HIVE脚本任务

* 创建job描述文件和hive脚本

Hive脚本： test.sql

|  |
| --- |
| use default;  drop table aztest;  create table aztest(id int,name string) row format delimited fields terminated by ',';  load data inpath '/aztest/hiveinput' into table aztest;  create table azres as select \* from aztest;  insert overwrite directory '/aztest/hiveoutput' select count(1) from aztest; |

Job描述文件：hivef.job

|  |
| --- |
| # hivef.job  type=command  command=/home/hadoop/apps/hive/bin/hive -f 'test.sql' |

2、将所有job资源文件打到一个zip包中

3、在azkaban的web管理界面创建工程并上传zip包

4、启动job

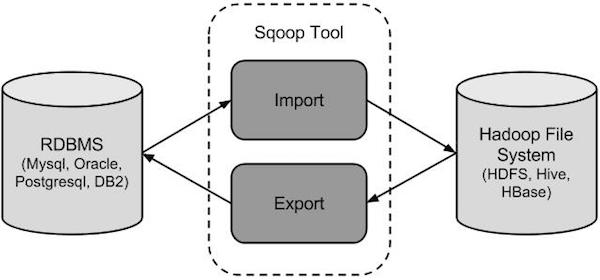
# 3. sqoop数据迁移

## 3.1 概述

sqoop是apache旗下一款**“Hadoop和关系数据库服务器之间传送数据”**的工具。

**导入数据**：MySQL，Oracle导入数据到Hadoop的HDFS、HIVE、HBASE等数据存储系统；

**导出数据：**从Hadoop的文件系统中导出数据到关系数据库



## 3.2 工作机制

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

## 3.3 sqoop实战及原理

### 3.3.1 sqoop安装

安装sqoop的前提是已经具备java和hadoop的环境

#### 1、下载并解压

最新版下载地址http://ftp.wayne.edu/apache/sqoop/1.4.6/

#### 2、修改配置文件

$ cd $SQOOP\_HOME/conf

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

打开sqoop-env.sh并编辑下面几行：

export HADOOP\_COMMON\_HOME=/home/hadoop/apps/hadoop-2.6.1/

export HADOOP\_MAPRED\_HOME=/home/hadoop/apps/hadoop-2.6.1/

export HIVE\_HOME=/home/hadoop/apps/hive-1.2.1

#### 加入mysql的jdbc驱动包

cp ~/app/hive/lib/mysql-conn ector-java-5.1.28.jar $SQOOP\_HOME/lib/

#### 4、验证启动

$ cd $SQOOP\_HOME/bin

$ sqoop-version

预期的输出：

15/12/17 14:52:32 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6

Sqoop 1.4.6 git commit id 5b34accaca7de251fc91161733f906af2eddbe83

Compiled by abe on Fri Aug 1 11:19:26 PDT 2015

到这里，整个Sqoop安装工作完成。

## 3.4 Sqoop的数据导入

“导入工具”导入单个表从RDBMS到HDFS。表中的每一行被视为HDFS的记录。所有记录都存储为文本文件的文本数据（或者Avro、sequence文件等二进制数据）

### 3.4.1 语法

下面的语法用于将数据导入HDFS。

|  |
| --- |
| $ sqoop import (generic-args) (import-args) |

### 3.4.2 示例

#### 表数据

在mysql中有一个库userdb中三个表：emp, emp\_add和emp\_contact

表emp:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **id** | **name** | **deg** | **salary** | **dept** |
| 1201 | gopal | manager | 50,000 | TP |
| 1202 | manisha | Proof reader | 50,000 | TP |
| 1203 | khalil | php dev | 30,000 | AC |
| 1204 | prasanth | php dev | 30,000 | AC |
| 1205 | kranthi | admin | 20,000 | TP |

表emp\_add:

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | **hno** | **street** | **city** |
| 1201 | 288A | vgiri | jublee |
| 1202 | 108I | aoc | sec-bad |
| 1203 | 144Z | pgutta | hyd |
| 1204 | 78B | old city | sec-bad |
| 1205 | 720X | hitec | sec-bad |

表emp\_conn:

|  |  |  |
| --- | --- | --- |
| **id** | **phno** | **email** |
| 1201 | 2356742 | gopal@tp.com |
| 1202 | 1661663 | manisha@tp.com |
| 1203 | 8887776 | khalil@ac.com |
| 1204 | 9988774 | prasanth@ac.com |
| 1205 | 1231231 | kranthi@tp.com |

#### 导入表表数据到HDFS

下面的命令用于从MySQL数据库服务器中的emp表导入HDFS。

|  |
| --- |
| $bin/sqoop import \  --connect jdbc:mysql://hdp-node-01:3306/test \  --username root \  --password root \  --table emp --m 1 |

如果成功执行，那么会得到下面的输出。

|  |
| --- |
| 14/12/22 15:24:54 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5  14/12/22 15:24:56 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.  INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-hadoop/compile/cebe706d23ebb1fd99c1f063ad51ebd7/emp.jar  -----------------------------------------------------  O mapreduce.Job: map 0% reduce 0%  14/12/22 15:28:08 INFO mapreduce.Job: map 100% reduce 0%  14/12/22 15:28:16 INFO mapreduce.Job: Job job\_1419242001831\_0001 completed successfully  -----------------------------------------------------  -----------------------------------------------------  14/12/22 15:28:17 INFO mapreduce.ImportJobBase: Transferred 145 bytes in 177.5849 seconds (0.8165 bytes/sec)  14/12/22 15:28:17 INFO mapreduce.ImportJobBase: Retrieved 5 records. |

为了验证在HDFS导入的数据，请使用以下命令查看导入的数据

|  |
| --- |
| $ $HADOOP\_HOME/bin/hadoop fs -cat /user/hadoop/emp/part-m-00000 |

emp表的数据和字段之间用逗号(,)表示。

|  |
| --- |
| 1201, gopal, manager, 50000, TP  1202, manisha, preader, 50000, TP  1203, kalil, php dev, 30000, AC  1204, prasanth, php dev, 30000, AC  1205, kranthi, admin, 20000, TP |

#### 导入关系表到HIVE

|  |
| --- |
| bin/sqoop import --connect jdbc:mysql://hdp-node-01:3306/test --username root --password root --table emp --hive-import --m 1 |

#### 导入到HDFS指定目录

在导入表数据到HDFS使用Sqoop导入工具，我们可以指定目标目录。

以下是指定目标目录选项的Sqoop导入命令的语法。

|  |
| --- |
| --target-dir <new or exist directory in HDFS> |

下面的命令是用来导入emp\_add表数据到'/queryresult'目录。

|  |
| --- |
| bin/sqoop import \  --connect jdbc:mysql://hdp-node-01:3306/test \  --username root \  --password root \  --target-dir /queryresult \  --table emp --m 1 |

下面的命令是用来验证 /queryresult 目录中 emp\_add表导入的数据形式。

|  |
| --- |
| $HADOOP\_HOME/bin/hadoop fs -cat /queryresult/part-m-\* |

它会用逗号（，）分隔emp\_add表的数据和字段。

|  |
| --- |
| 1201, 288A, vgiri, jublee  1202, 108I, aoc, sec-bad  1203, 144Z, pgutta, hyd  1204, 78B, oldcity, sec-bad  1205, 720C, hitech, sec-bad |

#### 导入表数据子集

我们可以导入表的使用Sqoop导入工具，"where"子句的一个子集。它执行在各自的数据库服务器相应的SQL查询，并将结果存储在HDFS的目标目录。

where子句的语法如下。

|  |
| --- |
| --where <condition> |

下面的命令用来导入emp\_add表数据的子集。子集查询检索员工ID和地址，居住城市为：Secunderabad

|  |
| --- |
| bin/sqoop import \  --connect jdbc:mysql://hdp-node-01:3306/test \  --username root \  --password root \  --where "city ='sec-ba d'" \  --target-dir /wherequery \  --table emp\_add --m 1 |

下面的命令用来验证数据从emp\_add表导入/wherequery目录

|  |
| --- |
| $HADOOP\_HOME/bin/hadoop fs -cat /wherequery/part-m-\* |

它用逗号（，）分隔 emp\_add表数据和字段。

|  |
| --- |
| 1202, 108I, aoc, sec-bad  1204, 78B, oldcity, sec-bad  1205, 720C, hitech, sec-bad |

#### 增量导入

增量导入是仅导入新添加的表中的行的技术。

它需要添加‘incremental’, ‘check-column’, 和 ‘last-value’选项来执行增量导入。

下面的语法用于Sqoop导入命令增量选项。

|  |
| --- |
| --incremental <mode>  --check-column <column name>  --last value <last check column value> |

假设新添加的数据转换成emp表如下：

1206, satish p, grp des, 20000, GR

下面的命令用于在EMP表执行增量导入。

|  |
| --- |
| bin/sqoop import \  --connect jdbc:mysql://hdp-node-01:3306/test \  --username root \  --password root \  --table emp --m 1 \  --incremental append \  --check-column id \  --last-value 1205 |

以下命令用于从emp表导入HDFS emp/ 目录的数据验证。

|  |
| --- |
| $ $HADOOP\_HOME/bin/hadoop fs -cat /user/hadoop/emp/part-m-\*  它用逗号（，）分隔 emp\_add表数据和字段。  1201, gopal, manager, 50000, TP  1202, manisha, preader, 50000, TP  1203, kalil, php dev, 30000, AC  1204, prasanth, php dev, 30000, AC  1205, kranthi, admin, 20000, TP  1206, satish p, grp des, 20000, GR |

下面的命令是从表emp 用来查看修改或新添加的行

|  |
| --- |
| $ $HADOOP\_HOME/bin/hadoop fs -cat /emp/part-m-\*1  这表示新添加的行用逗号（，）分隔emp表的字段。  1206, satish p, grp des, 20000, GR |

## 3.5 Sqoop的数据导出

将数据从HDFS导出到RDBMS数据库

导出前，目标表必须存在于目标数据库中。

* 默认操作是从将文件中的数据使用INSERT语句插入到表中
* 更新模式下，是生成UPDATE语句更新表数据

### 语法

以下是export命令语法。

|  |
| --- |
| $ sqoop export (generic-args) (export-args) |

### 示例

数据是在HDFS中“EMP/”目录的emp\_data文件中。所述emp\_data如下：

|  |
| --- |
| 1201, gopal, manager, 50000, TP  1202, manisha, preader, 50000, TP  1203, kalil, php dev, 30000, AC  1204, prasanth, php dev, 30000, AC  1205, kranthi, admin, 20000, TP  1206, satish p, grp des, 20000, GR |

1、首先需要手动创建mysql中的目标表

|  |
| --- |
| $ mysql  mysql> USE db;  mysql> CREATE TABLE employee (  id INT NOT NULL PRIMARY KEY,  name VARCHAR(20),  deg VARCHAR(20),  salary INT,  dept VARCHAR(10)); |

2、然后执行导出命令

|  |
| --- |
| bin/sqoop export \  --connect jdbc:mysql://hdp-node-01:3306/test \  --username root \  --password root \  --table emp2 \  --export-dir /user/hadoop/emp/ |

3、验证表mysql命令行。

|  |
| --- |
| mysql>select \* from employee;  如果给定的数据存储成功，那么可以找到数据在如下的employee表。  +------+--------------+-------------+-------------------+--------+  | Id | Name | Designation | Salary | Dept |  +------+--------------+-------------+-------------------+--------+  | 1201 | gopal | manager | 50000 | TP |  | 1202 | manisha | preader | 50000 | TP |  | 1203 | kalil | php dev | 30000 | AC |  | 1204 | prasanth | php dev | 30000 | AC |  | 1205 | kranthi | admin | 20000 | TP |  | 1206 | satish p | grp des | 20000 | GR |  +------+--------------+-------------+-------------------+--------+ |

## 3.6 Sqoop作业

*注：Sqoop作业——将事先定义好的数据导入导出任务按照指定流程运行*

#### 语法

以下是创建Sqoop作业的语法。

|  |
| --- |
| $ sqoop job (generic-args) (job-args)  [-- [subtool-name] (subtool-args)]  $ sqoop-job (generic-args) (job-args)  [-- [subtool-name] (subtool-args)] |

#### 创建作业(--create)

在这里，我们创建一个名为myjob，这可以从RDBMS表的数据导入到HDFS作业。

|  |
| --- |
| bin/sqoop job --create myimportjob -- import --connect jdbc:mysql://hdp-node-01:3306/test --username root --password root --table emp --m 1 |

该命令创建了一个从db库的employee表导入到HDFS文件的作业。

#### 验证作业 (--list)

**‘--list’**参数是用来验证保存的作业。下面的命令用来验证保存Sqoop作业的列表。

$ sqoop job --list

它显示了保存作业列表。

Available jobs:

myjob

检查作业(--show)

**‘--show’**参数用于检查或验证特定的工作，及其详细信息。以下命令和样本输出用来验证一个名为myjob的作业。

$ sqoop job --show myjob

它显示了工具和它们的选择，这是使用在myjob中作业情况。

|  |
| --- |
| Job: myjob  Tool: import Options:  ----------------------------  direct.import = true  codegen.input.delimiters.record = 0  hdfs.append.dir = false  db.table = employee  ...  incremental.last.value = 1206  ... |

#### 执行作业 (--exec)

**‘--exec’**选项用于执行保存的作业。下面的命令用于执行保存的作业称为myjob。

|  |
| --- |
| $ sqoop job --exec myjob  它会显示下面的输出。  10/08/19 13:08:45 INFO tool.CodeGenTool: Beginning code generation  ... |

## 3.7 Sqoop的原理

### 概述

Sqoop的原理其实就是将导入导出命令转化为mapreduce程序来执行，sqoop在接收到命令后，都要生成mapreduce程序

使用sqoop的代码生成工具可以方便查看到sqoop所生成的java代码，并可在此基础之上进行深入定制开发

### 代码定制

以下是Sqoop代码生成命令的语法：

|  |
| --- |
| $ sqoop-codegen (generic-args) (codegen-args)  $ sqoop-codegen (generic-args) (codegen-args) |

示例：以USERDB数据库中的表emp来生成Java代码为例。

下面的命令用来生成导入

|  |
| --- |
| $ sqoop-codegen \  --import  --connect jdbc:mysql://localhost/userdb \  --username root \  --table emp |

如果命令成功执行，那么它就会产生如下的输出。

|  |
| --- |
| 14/12/23 02:34:40 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5  14/12/23 02:34:41 INFO tool.CodeGenTool: Beginning code generation  ……………….  14/12/23 02:34:42 INFO orm.CompilationManager: HADOOP\_MAPRED\_HOME is /usr/local/hadoop  Note: /tmp/sqoop-hadoop/compile/9a300a1f94899df4a9b10f9935ed9f91/emp.java uses or overrides a deprecated API.  Note: Recompile with -Xlint:deprecation for details.  14/12/23 02:34:47 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-hadoop/compile/9a300a1f94899df4a9b10f9935ed9f91/emp.jar |

验证: 查看输出目录下的文件

|  |
| --- |
| $ cd /tmp/sqoop-hadoop/compile/9a300a1f94899df4a9b10f9935ed9f91/  $ ls  emp.class  emp.jar  emp.java |

如果想做深入定制导出，则可修改上述代码文件