# Sqoop1和Sqoop2区别

| **Feature** | **Sqoop 1** | **Sqoop 2** |
| --- | --- | --- |
| Connectors for all major RDBMS | Supported. | Not supported.  **Workaround**: Use the generic JDBC Connector which has been tested on the following databases: Microsoft SQL Server, PostgreSQL, MySQL and Oracle.  This connector should work on any other JDBC compliant database. However, performance might not be comparable to that of specialized connectors in Sqoop. |
| Kerberos Security Integration | Supported. | Supported. |
| Data transfer from RDBMS to Hive or HBase | Supported. | Not supported.  **Workaround:** Follow this two-step approach.   1. Import data from RDBMS into HDFS 2. Load data into Hive or HBase manually using appropriate tools and commands such as the LOAD DATA statement in Hive |
| Data transfer from Hive or HBase to RDBMS | Not supported.  **Workaround:** Follow this two-step approach.   1. Extract data from Hive or HBase into HDFS (either as a text or Avro file) 2. Use Sqoop to export output of previous step to RDBMS | Not supported.  Follow the same workaround as for Sqoop 1. |

参考：http://www.cloudera.com/documentation/enterprise/latest/topics/cdh\_ig\_sqoop\_vs\_sqoop2.html

# Sqoop1使用方法

使用前需要将驱动放入/opt/cloudera/parcels/CDH/lib/sqoop/lib/和/opt/cloudera/parcels/CDH/lib/sqoop2/lib/中，否则会报找不到驱动的错误

## Mysql导入到hdfs

1. **测试连接是否正常**

|  |
| --- |
| sqoop eval --connect jdbc:mysql://cdh:3306/dmp --username root --password hadoop --query 'select \* from dmp\_tag' |

会返回sql的执行结果

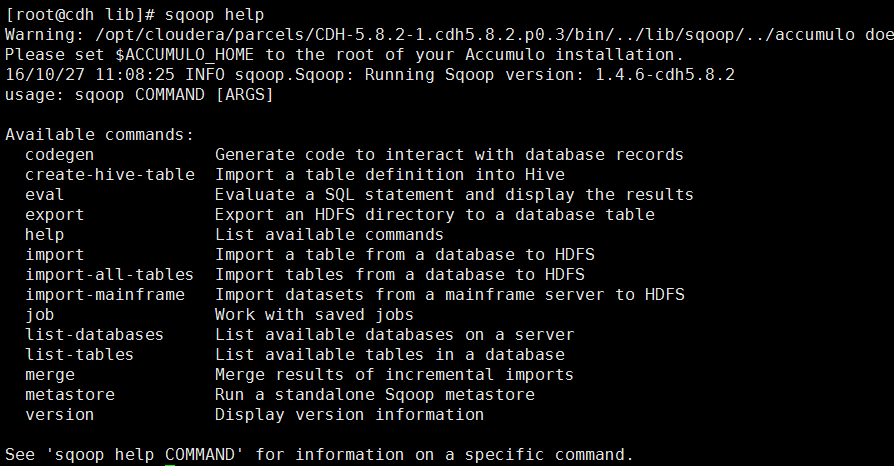
**2，列举所有mysql数据库**

可以通过sqoop-list-databases –help查看帮助文件

|  |
| --- |
| sqoop-list-databases --connect jdbc:mysql://cdh:3306/ --username root --password hadoop |

**3，将mysql数据库中的数据导入到hdfs**

sqoop help查看帮助



这里导入hdfs用到import开头的几个，当然sqoop也提供了一个工具sqoop-import，和sqoop import是一样的

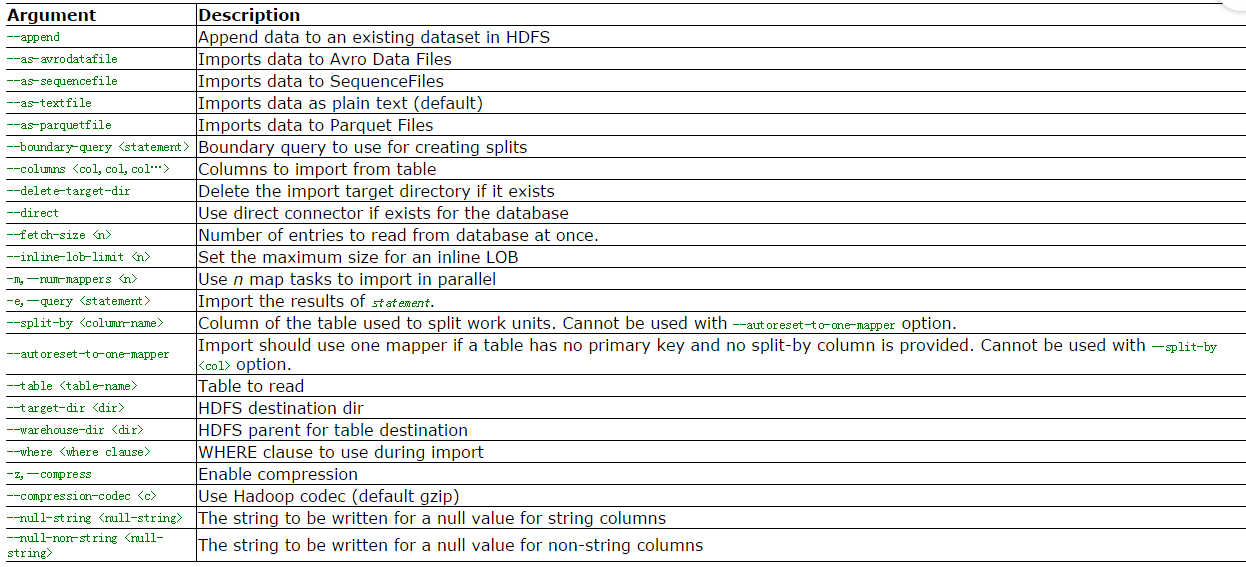
|  |
| --- |
| sqoop-import --connect jdbc:mysql://cdh:3306/dmp --username root --password hadoop --table dmp\_tag -m 2 --target-dir /user/root/result |

这里注意权限问题，给的目录当前执行用户需要有写入权限

也可以启用压缩（默认是gzip压缩），也可以通过--compression-codec指定压缩格式，以及追加到上一个目录后面

|  |
| --- |
| sqoop-import --connect jdbc:mysql://cdh:3306/dmp --username root --password hadoop --table dmp\_tag -m 2 --target-dir /user/root/result -z --append |

详细使用说明如下：



这里还可以做很多过滤，比如指定导入哪几列，执行一个sql查询等

通过sql语句导入

|  |
| --- |
| sqoop-import --connect jdbc:mysql://cdh:3306/dmp --username root --password hadoop --query 'select \* from dmp\_tag where $CONDITIONS and id>23814' --split-by id -m 4 --target-dir /user/root/result1 |

指定列导入

|  |
| --- |
| sqoop-import --connect jdbc:mysql://cdh:3306/dmp --username root --password hadoop --table dmp\_tag --columns ID,CODE,NAME --where "id>23815" -m 4 --target-dir /user/root/result3 |

Sqoop还支持将任意的查询结果集导入，不使用--table、--columns和--where，使用SQL语句--query参数执行自由查询导入，但是必须指定--target-dir目录，必须指定--split-by 分隔列，同时必须使用where且在其后加个$CONDITIONS（必须是where后面，不能是条件后面），使Sqoop进程替代为一个唯一的条件表达式达到条件查询效果

如果SQL语句中使用双引号（“”），则必须使用\$CONDITIONS代替$CONDITIONS，使你的shell不将其识别为shell自身的变量

Sqoop从大部分的数据源并行的导入数据，我们可以使用-m参数控制Map tasks的数目，默认是4个

这里导入还有很多选项可以选择，比如存储格式，可以进行列式存储，或者是avro存储等

**4，其他可选参数**

附：可选的文件参数如下表。

| **参数** | **说明** |
| --- | --- |
| --enclosed-by <char> | 给字段值前后加上指定的字符，比如双引号，示例：--enclosed-by '\"'，显示例子："3","jimsss","dd@dd.com" |
| --escaped-by <char> | 给双引号作转义处理，如字段值为"测试"，经过 --escaped-by "\\" 处理后，在hdfs中的显示值为：\"测试\"，对单引号无效 |
| --fields-terminated-by <char> | 设定每个字段是以什么符号作为结束的，默认是逗号，也可以改为其它符号，如句号.，示例如：--fields-terminated-by |
| --lines-terminated-by <char> | 设定每条记录行之间的分隔符，默认是换行串，但也可以设定自己所需要的字符串，示例如：--lines-terminated-by "#" 以#号分隔 |
| --mysql-delimiters | Mysql默认的分隔符设置，字段之间以,隔开，行之间以换行\n隔开，默认转义符号是\，字段值以单引号'包含起来。 |
| --optionally-enclosed-by <char> | enclosed-by是强制给每个字段值前后都加上指定的符号，而--optionally-enclosed-by只是给带有双引号或单引号的字段值加上指定的符号，故叫可选的 |

参考：<http://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html>

<https://segmentfault.com/a/1190000002532293>

## Oracle导入到hdfs

sqoop list-tables --connect jdbc:oracle:thin:@cdh:1521:xe --username yunchen --password=root

## 增量导入

参数：

**--check-column (col)**： 检查指定的列，根据此列判断哪些记录是新数据且需要导入的，列不能是字符相关类型（CHAR/NCHAR/VARCHAR/VARNCHAR/ LONGVARCHAR/LONGNVARCHAR）

**--incremental (mode)**： 指定增量模式，mode包含两种方式：append和lastmodified

append： 当表中的记录是以id持续增加导入新的记录的时候，可以使用append模式，--check-column id 用于检查id

lastmodified： 表有时候也会执行更新操作，此时可以使用lastmodified导入

**--last-value (value)**： 指定某个值，将大于该值的检查列记录导入，以确定仅将新的或者更新后的记录导入新的文件系统

一般情况下，--check-column是数据库中的关键字，以此来判断哪些列是新增数据。--incremental 增量导入模式分为两种：append和lastmodified。--last-value 后面接的值一般是上一次导入操作后指定列的最大值，便于Sqoop将此最大值后新增的值导入。

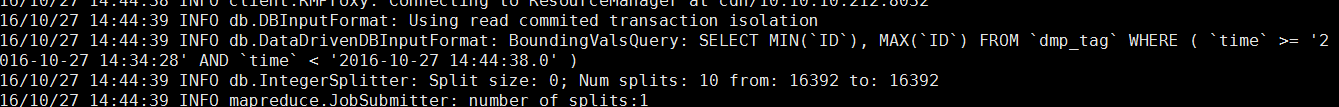
首次全量导入

|  |
| --- |
| sqoop-import --connect jdbc:mysql://cdh:3306/dmp --username root --password hadoop --table dmp\_tag -m 10 --target-dir /user/root/time --append |

再根据时间戳增量导入

|  |
| --- |
| sqoop-import --connect jdbc:mysql://cdh:3306/dmp --username root --password hadoop --table dmp\_tag --check-column time --incremental lastmodified --last-value '2016-10-27 14:34:28' -m 10 --target-dir /user/root/time --append |

这里注意：命令生成的sql语句是>=--last-value的值，<最大值



使用append增量导入，指定列--check-column后面需要接的是一个值，然后再使用--last-value后接最近的一个值，以此实现新加操作的导入。此处不详细介绍。

注意：不管是使用哪种模式导入，--check-column、--incremental和--last-value三个参数在命令中都必须同时使用。

## 合并HDFS文件

| **参数** | **描述** |
| --- | --- |
| –class-name | 表对应的class文件 |
| –jar-file | 表对应的jar包 |
| –merge-key | 进行merge的字段，根据该字段判断新旧记录，字段名需要大写 |
| –new-data | 增量数据对应的HDFS文件路径 |
| –onto | 全量数据对应的HDFS文件路径 |
| –target-dir | 合并后的HDFS文件路径 |

首先生成表结构的jar包：

|  |
| --- |
| sqoop codegen --connect jdbc:mysql://cdh:3306/dmp --username root --password hadoop --table dmp\_tag |

然后指定

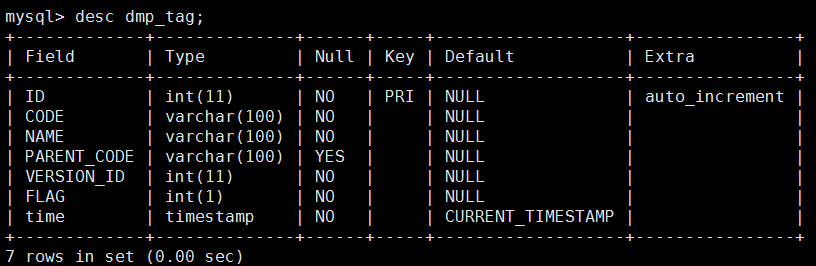
|  |
| --- |
| sqoop merge --new-data /user/root/inc1 --onto /user/root/inc --target-dir /user/root/inca1 --jar-file /tmp/sqoop-root/compile/0d586a5feda3f0dbb190defaa8ca0856/dmp\_tag.jar --class-name dmp\_tag --merge-key ID |

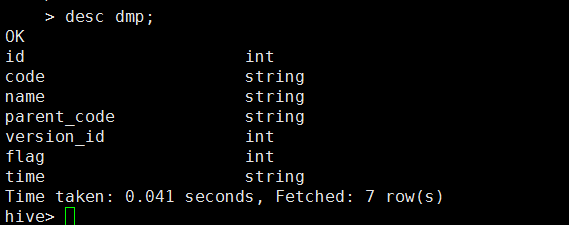
从--jar-file参数对应的代码中可以看到，指定表的所有字段在代码中都转换成了大写，所以在指定merge-key时，记得大写字段名，否则会报错。在执行merge操作时，必须指定--jar-file参数和--class-name参数。jar文件可以在对这个表的sqoop import过程中生成，也可以用后面的codegen工具生成。

## Mysql导入到hive

1. **同步表到hive中**

|  |
| --- |
| sudo -u hive sqoop create-hive-table --connect jdbc:mysql://cdh:3306/dmp --username root --password hadoop --table dmp\_tag --hive-table dmp |





这里导入后字段数据类型变了

**2，调整Hive数据类型**

上面任务执行成功后，经过检测，发现Hive表结构中的数据类型与MySQL对应列有如下关系：

MySQL(bigint) --> Hive(bigint)

MySQL(tinyint) --> Hive(tinyint)

MySQL(int) --> Hive(int)

MySQL(double) --> Hive(double)

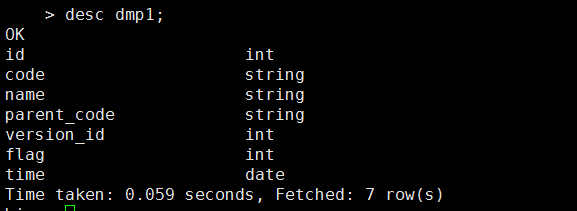
MySQL(bit) --> Hive(boolean)

MySQL(varchar) --> Hive(string)

MySQL(decimal) --> Hive(double)

MySQL(date/timestamp) --> Hive(string)

|  |
| --- |
| sudo -u hive sqoop create-hive-table --connect jdbc:mysql://cdh:3306/dmp --username root --password hadoop --table dmp\_tag --hive-table dmp1 --map-column-hive time="DATE" |



这里就改过来了,不过不建议先建表再导入数据到hive中，我这的测试是这样做后有时候会导致所有数值都为空null，很奇怪

**3，导入数据**

直接执行导入，

|  |
| --- |
| sudo -u hive sqoop import --connect jdbc:mysql://cdh:3306/dmp --username root --password hadoop --table dmp\_tag --fields-terminated-by "\t" --lines-terminated-by "\n" --hive-import --hive-overwrite --hive-table dmp6 --delete-target-dir |

参考：<http://www.tuicool.com/articles/fQrAJby>

## Mysql导入HBase

|  |
| --- |
| sqoop import --connect jdbc:mysql://cdh:3306/dmp --username root --password hadoop --table dmp\_tag --hbase-table ttt --hbase-create-table --hbase-row-key ID --hbase-bulkload -m 7 --column-family info |

以上指定了HBase的Rowkey后，再次执行从MySQL导入数据到HBase的Sqoop语句，基于相同的Rowkey值，HBase内相应的行会进行更新替换。

**Hive使用HBase数据**

|  |
| --- |
| CREATE EXTERNAL TABLE default.angel(  ID BIGINT,  CODE BIGINT,  FLAG BIGINT,  NAME STRING,  VERSION\_ID BIGINT,  time STRING)  ROW FORMAT SERDE 'org.apache.hadoop.hive.hbase.HBaseSerDe'  STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'  WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key, info:CODE, info:FLAG, info:NAME, info:VERSION\_ID, info:time")  TBLPROPERTIES("hbase.table.name" = "ttt"); |

## 通过java实现mysql导入hdfs

|  |
| --- |
| package sqoop;  import org.apache.hadoop.conf.Configuration; import org.apache.sqoop.Sqoop; import org.apache.sqoop.tool.SqoopTool; import org.apache.sqoop.util.OptionsFileUtil;  */\*\*  \* Created by jimmy on 2016/10/28.  \* 1，sqoop jar包  \* 2，hadoop-common jar包  \* 3，hadoop-hdfs jar包  \* 4，hadoop-common lib jar包  \* 5，hadoop mapreduce jar包  \* 6，mysql jar包  \*  \* 完成后打jar包，不需要吧依赖的jar包打进去  \*  \* 运行命令：java -Djava.ext.dirs=/opt/cloudera/parcels/CDH/jars:/usr/java/latest/jre/lib/ext/ -jar sqoop1.jar  \*  \* 需求：  \* 1，源数据库可自定义 dmp  \* 2，源表可自定义 dmp\_tag  \* 3，源数据库类型可自定义 mysql or oracle  \* 4，hdfs集群地址可自定义  \* 5，输出路径可自定义  \* 6，增量全量自定义、增量字段、增量  \* 7，压缩格式、存储格式自定义  \*  \*/* public class importDataFromMysqlToHdfs\_v2 {   public static void main(String[] args) {  String[] sqoopargs = new String[]{  "--connect", "jdbc:mysql://cdh:3306/dmp",  "--username", "root",  "--password", "hadoop",  "--table", "dmp\_tag", /\* "--columns", "LOGIN\_NAME,PASSWORD",  "--split-by", "USER\_ID",\*/  "--target-dir", "/user/root/java"  };  try {  *importDataFromMysql*(sqoopargs);  } catch (Exception e){  e.printStackTrace();  }  }   public static int importDataFromMysql(String[] args) {   String[] expandedArgs = null;  try {  expandedArgs = OptionsFileUtil.*expandArguments*(args);  } catch (Exception ex) {  System.*err*.println(ex.getMessage());  System.*err*.println("Try 'sqoop help' for usage.");  }   com.cloudera.sqoop.tool.SqoopTool tool = (com.cloudera.sqoop.tool.SqoopTool) SqoopTool.*getTool*("import");  //com.cloudera.sqoop.tool.SqoopTool tool = new ImportTool();   Configuration conf = new Configuration() ;  conf.set("fs.default.name", "hdfs://cdh:8020/");//设置hadoop服务地址  Configuration pluginConf = tool.*loadPlugins*(conf);   Sqoop sqoop = new Sqoop(tool, pluginConf);  return Sqoop.*runSqoop*(sqoop, expandedArgs);  }  } |

代码需要进一步优化

## Kerberos认证

如果启用了kerberos认证，只需先kinit初始化凭据就行

# Sqoop2使用方法

## 介绍

在架构上， sqoop2 引入了 sqoop server （具体服务器为 tomcat ），对 connector 实现了集中的管理。其访问方式也变得多样化了，其可以通过 REST API 、 JAVA API 、 WEB UI 以及 CLI 控制台方式进行访问。另外，其在安全性能方面也有一定的改善，在 sqoop1 中我们经常用脚本的方式将 HDFS 中的数据导入到 mysql 中，或者反过来将 mysql 数据导入到 HDFS 中，其中在脚本里边都要显示指定 mysql 数据库的用户名和密码的，安全性做的不是太完善。在 sqoop2 中，如果是通过 CLI 方式访问的话，会有一个交互过程界面，你输入的密码信息不被看到。

## Mysql导入到hdfs

1. 常见命令

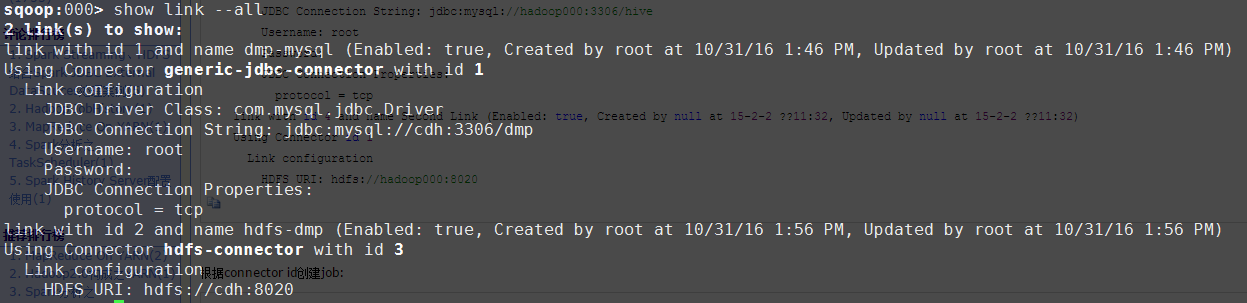
|  |
| --- |
| 启动客户端：  **sqoop**  **set server --host cdh --port 12000 --webapp sqoop**  查看所有connector：  **show connector --all**  查询所有link:  **show link**  删除指定link:  **delete link --lid x**  查询所有job:  **show job**  删除指定job:  **delete job --jid 1** |

1. 创建mysql link

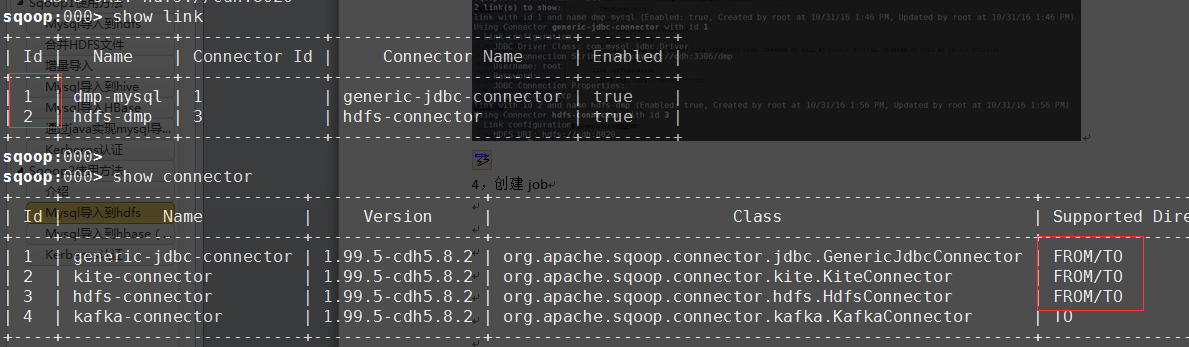
|  |
| --- |
|  |

1. 创建hdfs link

|  |
| --- |
|  |

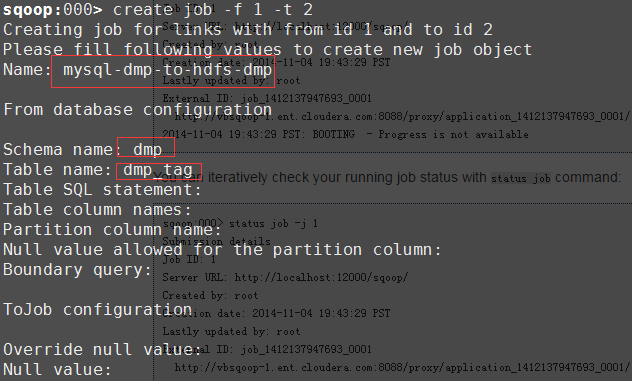


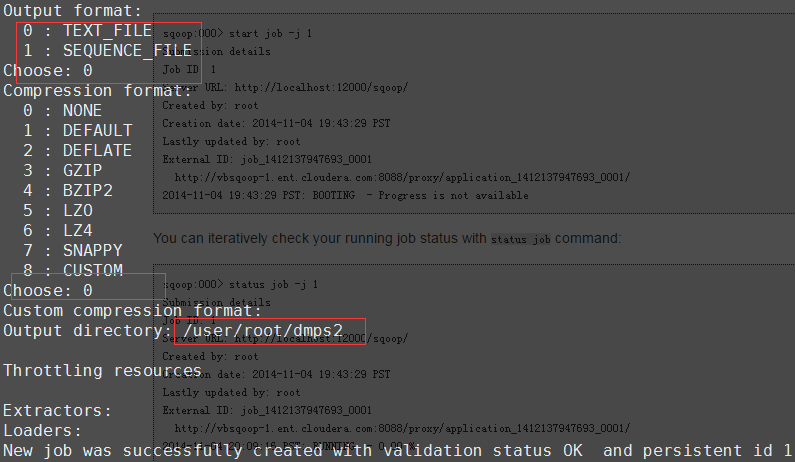
4，创建job



Connectors 的 From 用于读取数据， To 用于写入数据。使用上面的 show connector -all 命令可以显示出 Generic JDBC Connector 对 From 和 To 都是支持的。也就是说我们既可以从数据库中读取数据，也可以往数据库中写入数据。为了创建一个 Job ，我们需要指定 Job 的 From 和 To 部分， From 和 To 部分可以使用 link Id 来表示。

最后，我们可以使用这两个 link Id 来关联 job 的 From 和 To 部分。说的通俗一点，就是我们需要从哪里 (From) 读取数据 , 把这些数据导入 (To) 到哪里。





## Mysql导入到hbase（代码实现）

## Kerberos认证