　　本文以[Sqoop User Guide (v1.4.5)](https://sqoop.apache.org/docs/1.4.5/SqoopUserGuide.html" \t "http://blog.csdn.net/dabokele/article/details/_blank)为主，对Sqoop-1.4.5的用户手册进行翻译，同时会结合一些实际操作中的注意事项一并写入。

# 1、Introduction

　　Sqoop是一个用于在**[Hadoop](http://lib.csdn.net/base/hadoop" \o "Hadoop知识库" \t "http://blog.csdn.net/dabokele/article/details/_blank)**和关系型**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/dabokele/article/details/_blank)**之间流转数据的一个工具。可以使用Sqoop将数据从关系型数据库系统(RDBMS)比如**[MySQL](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/dabokele/article/details/_blank)**或者**[Oracle](http://lib.csdn.net/base/oracle" \o "Oracle知识库" \t "http://blog.csdn.net/dabokele/article/details/_blank)**导入到**[hadoop](http://lib.csdn.net/base/hadoop" \o "Hadoop知识库" \t "http://blog.csdn.net/dabokele/article/details/_blank)**分布式文件系统(HDFS)上，然后数据在Hadoop MapReduce上转换，以及将数据导出到RDBMS中。   
　　Sqoop自动实现了上面提到的很多过程，Sqoop使用MapReduce来导入和导出数据，这样既可以提供并行化操作又可以提高容错能力。   
　　这篇文档介绍了如何使用Sqoop在database和Hadoop之间流转数据，以及介绍了Sqoop命令行工具的使用方法。

# 2、Supported Releases

　　这篇文档适用于Sqoop v1.4.5。

# 3、Sqoop Releases

　　Sqoop是Apache软件基金会的一个开源项目。可以访问[http://Sqoop.apache.org](http://sqoop.apache.org/" \t "http://blog.csdn.net/dabokele/article/details/_blank)获取   
- 最新版本的Sqoop以及其源码   
- bug的追踪   
- 包含Sqoop文档的Wiki

# 4、Prerequisites

　　如果想要使用Sqoop，需要已经准备好以下条件：   
- 了解一些基本的计算机知识和术语   
- 熟悉比如bash等的命令行操作   
- 了解关系型数据管理系统(RDBMS)   
- 熟悉一些Hadoop的相关操作   
　　   
　　在使用Sqoop之前，Hadoop必须安装并配置好，目前Sqoop支持hadoop的4个主要版本-0.20,0.23,1.0以及2.0。   
　　这篇文档基于**[Linux](http://lib.csdn.net/base/linux" \o "Linux知识库" \t "http://blog.csdn.net/dabokele/article/details/_blank)**或类**[linux](http://lib.csdn.net/base/linux" \o "Linux知识库" \t "http://blog.csdn.net/dabokele/article/details/_blank)**环境，如果使用的是Windows系统的话，最好使用cygwin来操作以下的内容。如果使用Mac OS X，可能会遇到一些报错。Sqoop最好的运行环境还是Linux。

# 5、Basic Usage

　　使用Sqoop可以从关系型数据库中导入数据到HDFS上，在这个过程中import操作的输入是一个数据库表，Sqoop会逐行读取记录到HDFS中。import操作的输出是包含读入表的一系列HDFS文件，import操作是并行的也就是说可以启动多个map同时读取数据到HDFS，每一个map对应一个输出文件。这些文件可以是TextFile类型，也可以是Avro类型或者SequenceFile类型。   
　　在import过程中还会生成一个**[Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "http://blog.csdn.net/dabokele/article/details/_blank)**类，这个类与输入的表对应，类名即表名，类变量即表字段。import过程会使用到这个Java类。   
　　在import操作之后，就可以使用这些数据来实验export过程了。export是将数据从HDFS或者**[Hive](http://lib.csdn.net/base/hive" \o "Hive知识库" \t "http://blog.csdn.net/dabokele/article/details/_blank)**导出到关系型数据库中。export过程并行的读取HDFS上的文件，将每一条内容转化成一条记录，然后作为一个新行insert到关系型数据库表中。   
　　除了import和export，Sqoop还包含一些其他的操作。比如可以使用sqoop-list-databases工具查询数据库结构，可以使用sqoop-list-tables工具查询表信息。还可以使用sqoop-eval工具执行SQL查询。 

# 6、Sqoop Tools

　　Sqoop提供了一系列的操作工具，使用Sqoop需要指定你想要使用的具体工具，以及提供对应的一些参数，使用方式如下。

$ sqoop tool-name [tool-arguments]

　　可以使用sqoop help命令查看帮助信息，从

$ sqoop help

usage: sqoop COMMAND [ARGS]

Available commands:

codegen 生成Java代码

create-hive-table 根据表结构生成hive表

eval 执行SQL语句并返回结果

export 导出HDFS文件到数据库表

help 帮助

import 从数据库导入数据到HDFS

import-all-tables 导入数据库所有表到HDFS

list-databases 列举所有的database

list-tables 列举数据库中的所有表

version 查看版本信息

See 'sqoop help COMMAND' for information on a specific command.

可以看到，sqoop提供的操作工具有10个。具体工具的使用帮助可以sqoop help (tool-name)或者sqoop tool-name --help进行查看。 

## 6.1. Using Command Aliases

　　在sqoop的bin目录下，提供了比如sqoop-merge之类的课执行脚本，其实这个脚本中封装的命令是sqoop merge。类似的命令还有sqoop-import, sqoop-export等。

## 6.2. Controlling the Hadoop Installation

　　sqoop命令最终执行了hadoop命令脚本。如果在服务器上安装了多个hadoop，可以使用HADOOPCOMMONHOME和HADOOP\_MAPRED\_HOME环境变量来指定使用哪个hadoop，比如

$ HADOOP\_COMMON\_HOME=/path/to/some/hadoop \

HADOOP\_MAPRED\_HOME=/path/to/some/hadoop-mapreduce \

sqoop import --arguments...

或者

$ export HADOOP\_COMMON\_HOME=/some/path/to/hadoop$ export HADOOP\_MAPRED\_HOME=/some/path/to/hadoop-mapreduce$ sqoop import --arguments...

## 6.3. Using Generic and Specific Arguments

　　Sqoop的每一个工具都需要接收一些参数，这些参数有hadoop相关参数，也有用户指定的sqoop相关参数。   
　　比如，sqoop import工具的使用

$ sqoop help import

usage: sqoop import [GENERIC-ARGS] [TOOL-ARGS]

Common arguments:

--connect <jdbc-uri> 指定JDBC连接串 --connect-manager <jdbc-uri> 指定连接类 --driver <class-name> 指定使用的JDFC驱动类 --hadoop-mapred-home <dir>+ 可以覆盖$HADOOP\_MAPRED\_HOME参数 --help 帮助 --password-file 指定密码文件 -P 从命令行读取密码 --password <password> 指定密码 --username <username> 指定用户名 --verbose 显示import过程更多信息 --hadoop-home <dir>+ Deprecated. 可以覆盖$HADOOP\_HOME参数

[...]

hadoop命令行参数:

(must preceed any tool-specific arguments)

Generic options supported are

-conf <configuration file> 指定配置文件-D <property=value> 指定特定参数的值-fs <local|namenode:port> 指定namenode-jt <local|jobtracker:port> 字段jobtracker-files <comma separated list of files> 逗号分隔的文件列表，这些文件会上传到hdfs-libjars <comma separated list of jars> 逗号分隔的jar包，这些jar包会上传到hdfs-archives <comma separated list of archives>

The general command line syntax is

bin/hadoop command [genericOptions] [commandOptions]

　　hadoop相关的参数只能写在sqoop工具之后，但是在任何sqoop参数(比如 –connect)之前。   
　-conf, -D, -fs以及-jt参数用于配置Hadoop相关参数，比如-D mapred.job.name=<job\_name>可以指定Sqoop提交的MR任务名称，如果不指定的话，默认名称为表名。   
　-files, -libjars, -archives参数基本不会在Sqoop中用到。 

## 6.4. Using Options Files to Pass Arguments

　　对于不经常发生变化的参数，可以写在一个文件中，这个文件中每行写一个参数，如果某个参数特别长导致一行写不下的话，在结尾写一个反斜杠”\”，然后可以换行继续写。在这个配置文件中可以类似于shell脚本那样写注释，文件中空行不会起作用，引号外的空格也会忽略。配置参数的书写顺序是：首先写hadoop相关参数(即以“-”开头的参数)，然后写sqoop相关参数(即“–”开头的那些参数)，最后写需要传递到子程序中的参数。   
　　使用配置文件的话，用参数--options-file来指定，如果有个配置文件 /users/homer/work/import.txt中的内容如下

import

--connectjdbc:mysql://localhost/db

--usernamefoo

那么，下面两个命令的作用相同

$ sqoop import --connect jdbc:mysql://localhost/db --username foo --table TEST$ sqoop --options-file /users/homer/work/import.txt --table TEST

import.txt文件中包含注释和空行，空格的形式如下：

## Options file for Sqoop import#

# Specifies the tool being invoked

import

# Connect parameter and value

--connect

jdbc:mysql://localhost/db

# Username parameter and value

--username

foo

## Remaining options should be specified in the command line.#

## 6.5. Using Tools

　　文档接下来部分会详细描述每个Sqoop工具的作用，以及使用方法

# 7. sqoop-import

## 7.1. Purpose

　　import工具可以用于从RDBMS中导入一张表到HDFS。表中的每一条记录对应生成HDFS文件中的每一行。这些记录可以以text files或者Avro或者SequenceFiles格式进行存储。

## 7.2. Syntax

　　使用方法如下

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

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

参数列表1-import基本参数

| **参数** | **描述** |
| --- | --- |
| –connect < jdbc-uri > | JDBC连接串 |
| –connection-manager < class-name > | 连接管理类 |
| –driver < class-name > | 手动指定JDBC驱动类 |
| –hadoop-mapred-home < dir > | 可以覆盖$HADOOP\_MAPRED\_HOME |
| –help | 使用帮助 |
| –password-file | 指定包含密码的文件 |
| -P | 执行import时会暂停，等待用户手动输入密码 |
| –password < password > | 直接将密码写在命令行中 |
| –username < username > | 指定用户名 |
| –verbose | 显示Sqoop任务更多执行信息 |
| –connection-param-file < filename > | 可选的参数，用于提供连接参数 |
| –relaxed-isolation | 设置每个mapmer的连接事务隔离 |

**7.2.1. Connecting to a Database Server**   
　　这里介绍Sqoop如何连接到一个数据库，主要使用的是–connect参数，比如

$ sqoop import --connect jdbc:mysql://database.example.com/employees

这个连接串会连接database.example.com服务器上的employees数据库。这个连接串会被各个map节点用于连接数据库，所以如果写成localhost的话就会报错，**–connect参数后面不能写localhost来表示服务器地址**   
　　指定数据库连接后，接下来需要提供数据库访问的用户名和密码。上面提到了密码有三种设置方式，其中–password是最不安全的。推荐将密码写入一个文件，然后将该文件的权限设置成400，通过–password-file指定到该文件，这个文件可以位于本地也可位于HDFS上。

$ sqoop import --connect jdbc:mysql://database.example.com/employees \

--username venkatesh --password-file ${user.home}/.password

　　不推荐使用–password参数传递密码，因为使用ps命令就可以直接查看到该参数。   
　　Sqoop中内置了**[mysql](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/dabokele/article/details/_blank)**的驱动，如果需要指定其他的基于JDBC的数据库连接，需要将对应jar包放至lib目录下，然后指定驱动类，比如，

$ sqoop import --driver com.microsoft.jdbc.sqlserver.SQLServerDriver \

--connect <connect-string> ...

表2-验证相关的参数

| **参数** | **描述** |
| --- | --- |
| –valid | 启动数据验证功能，只对单表copy有效 |
| –validator < class-name > | 指定验证类 |
| –validation-threshold < class-name > | 指定使用的阈值验证类 |
| –validation-failurehandler < class-name > | 指定验证失败的处理类 |

表3-import控制参数

| **参数** | **描述** |
| --- | --- |
| –append | 导入的数据追加到数据文件中 |
| –as-avrodatafile | 导入数据格式为avro |
| –as-sequencefile | 导入数据格式为sqeuqncefile |
| –as-textfile | 导入数据格式为textfile |
| –boundary-query < statement > | 代替min(split-by),max(split-by)值指定的边界，下面会有详细介绍 |
| –columns < col,col,col… > | 指定导入字段 |
| –delete-target-dir | 如果导入的target路径存在，则删除 |
| –direct | 使用direct模式 |
| –fetch-size < n > | 从数据库一次性读入的记录数 |
| -inline-lob-limit < n > | 设定大对象数据类型的最大值 |
| -m, –num-mappers < n > | 指定并行导入数据的map个数，默认为4个 |
| -e, –query < statement > | 导入查询语句得到的数据 |
| –split-by < column-name > | 一般与-m参数一起使用，指定分割split的字段 |
| –table < table-name > | 指定database中的表名 |
| –target-dir < dir > | 指定目标HDFS路径 |
| –warehouse-dir < dir > | 指定表目标路径所在路径 |
| –where < where clause > | 即sql语句中的where条件 |
| -z, –compress | 打开压缩功能 |
| –compression-codec < c > | 使用Hadoop的压缩，默认为gzip压缩 |
| –null-string < null-string > | 源表中为null的记录导入为string类型时显示为null-string，默认显示为”null” |
| –null-non-string < null-string > | 源表中为null的记录导入为非string类型时显示为null-string，默认显示为”null” |

**7.2.2. Selecting the Data to Import**   
　　使用–table参数可以指定导入的表或者视图，默认情况下，该表或视图的所有字段都会被按顺序导入进来。但是如果只需要导入该表的部分字段，或者调整导入字段的顺序时，可以使用–columns参数。例如--columns "name, employee\_id, jobtitle"。   
　　也可以使用–where参数符合筛选条件的记录进行导入，例如--where "id > 400",则源表中只有id大于400的记录才会被导入进来。   
　　默认情况下，Sqoop会根据指定的–split-by字段的min，max值对记录进行分片并导入。但是在有些情况下，如果不需要导入从min(split-by)到max(split-by)的记录时，可以使用--boundary-query <statement>参数传入一个返回两个数字类型的sql语句来指定导入范围，如果传入非数字类型的字段，比如varchar型，会报Invalid value for getLong()之类的错误信息。   
　　   
**7.2.3. Free-form Query Imports**   
　　除了像上面那样指定–table，–columns, –where等参数的导入方法之外，Sqoop还支持传入一个SQL语句，并把该SQL语句的结果导入到HDFS中。这里需要使用–query或者-e参数。**但是使用–query参数时，同时要指定–target-dir参数**，如果需要并行的导入数据，即多个map并行的用相同的query语句查询不同的数据并最终汇总到一起。那么**需要在query语句中增加一个$CONDITIONS字符串，并且指定–split-by字段**。比如如下两个：

$ sqoop import \

--query 'SELECT a.\*, b.\* FROM a JOIN b on (a.id == b.id) WHERE $CONDITIONS' \ --split-by a.id --target-dir /user/foo/joinresults

$ sqoop import \

--query 'SELECT a.\*, b.\* FROM a JOIN b on (a.id == b.id) WHERE $CONDITIONS' \ -m 1 --target-dir /user/foo/joinresults

　　第二个语句是将map个数设置成1。这里需要注意一点上面两个示例中使用的是单引号，如果非要使用双引号，那么应该写**“$CONDITIONS**，即在前面加一个反斜杠。   
　　在当前Sqoop版本中，也不能传入特别复杂的sql语句，比如在where条件中不能有or操作，并且对有子查询的语句，可能会出现一些未知的错误。   
　　   
**7.2.4. Controlling Parallelism**   
　　Sqoop支持并发的从多种数据源中导入数据。可以使用-m或者–num-mappers参数指定map的个数，每一个map处理数据源中的一部分数据，默认情况下Sqoop的并发为4，对某些数据库来说，将并发调整到8或16会有显著的性能提升。   
　　但是并发调整的太高也不好，比如如果map个数太多，超出整个集群的最大并行数时，其他的map任务就需要等前面的map任务执行完毕才能继续执行，这样有可能反而增加了导入总时间。同时，如果并发太高，会增加数据库的访问压力，所以，应该根据实际情况调整一个最优并发参数。   
　　当并发的导入数据时，Sqoop需要有一个明确的划分数据集的指标，告诉Sqoop如何为每个map分配处理的数据，使得既无遗漏也无重复。默认情况下如果指定表有主键，Sqoop会根据主键的max, min值除以map个数来划分数据集。比如一张表有1000条记录，主键为id，并发为4，那么每个map上执行是SQL语句是SELECT \* FROM sometable WHERE id >= lo AND id < hi，对每个map来说，(lo, hi)分别是(0, 250), (250, 500), (500, 750), 以及(750, 1001)。   
　　但是这样会有一个数据倾斜的问题，比如1000条记录大部分分布在0~250区间内，那么会有一个map处理大量数据，而其他map处理的数据特别少。这时候就需要使用--split-by参数手动指定一个分割字段了。目前Sqoop不支持联合主键，那么如果导入表没有主键，或者有联合主键时，就需要手动指定分割字段。   
　　   
**7.2.5. Controlling Distributed Cache**   
　　每一个Sqoop任务执行时都会上传一次ib目录下的jar包。但是如果使用Oozie来启动Sqoop任务时，就没有必要这样做了，因为Oozie会使用它自己的Sqoop共享lib文件。使用--skip-dist-cache参数来指定Oozie调度起的Sqoop任务跳过上传jar包这一过程。   
　　   
**7.2.6. Controlling the Import Process**   
　　默认情况下，Imporg过程会使用到JDBC提供的稳定服务去访问和连接数据库。但是有些数据库提供了一些其他的数据库访问工具来为import过程提供更高效的服务。比如，MySQL提供了一个mysqldump工具可以使得从MySQL导出数据到其他系统更加的快速。通过指定--direct参数就可以指定Sqoop使用mysqldump从数据库导入数据。这个方式会比JDBC提供更加高效。   
　　对每一种关系型数据库如何使用direct模式import数据，需要如何安装配置，如何配置，以及有哪些局限性，都在第24节，24. Notes for specific connectors中有进一步的描述。   
　　默认情况下，Sqoop会将一个表名为foo的表，导入到HDFS上一个名称为foo的路径下。比如，如果使用用户someuser来执行Sqoop import，那么会在将foo表导入到HDFS上的/user/someuser/foo/(files)路径下。当然了，可以使用--warehouse-dir参数来指定导入文件的父路径。

**7.2.7. Controlling transaction isolation**   
**7.2.8. Controlling type mapping**   
**7.2.9. Incremental Imports**   
**7.2.10. File Formats**   
**7.2.11. Large Objects**   
**7.2.12. Importing Data Into** **[hive](http://lib.csdn.net/base/hive" \o "Hive知识库" \t "http://blog.csdn.net/dabokele/article/details/_blank)**   
**7.2.13. Importing Data Into** **[Hbase](http://lib.csdn.net/base/hbase" \o "Hbase知识库" \t "http://blog.csdn.net/dabokele/article/details/_blank)**   
**7.2.14. Importing Data Into Accumulo**   
**7.2.15. Additional Import Configuration Properties**

## 7.3. Example Invocations

# 8. sqoop-import-all-tables

## 8.1. Purpose

## 8.2. Syntax

## 8.3. Example Invocations

# 9. sqoop-export

## 9.1. Purpose

　　Sqoop的export工具可以从HDFS同步一系列文件数据到RDBMS中。使用这个工具的前提是导出目标表在数据库必须存在。导出文件根据用户指定的分隔符转化成一系列的输出记录。   
　　默认的导出操作会将这些记录转化成一系列的INSERT语句，根据这些语句将记录插入到关系型数据库中。而在update模式下，Sqoop会生成一系列的UPDATE语句，将数据库中已经存在的记录进行更新。在call模式下，Sqoop会为每一条记录调用一个存储过程来处理。 

## 9.2. Syntax

　　Sqoop的export工具使用语法如下：

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

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

　　前面的generic-args是Hadoop相关参数，如果有这些参数的话，必须写在export参数之前，export参数之间的顺序可以随意调整。

　　表19-基本参数

| **参数** | **描述** |
| --- | --- |
| –connect < jdbc-uri > | JDBC连接串 |
| –connection-manager < class-name > | 连接管理类 |
| –driver < class-name > | 手动指定JDBC驱动类 |
| –hadoop-mapred-home < dir > | 可以覆盖$HADOOP\_MAPRED\_HOME |
| –help | 使用帮助 |
| –password-file | 指定包含密码的文件 |
| -P | 执行import时会暂停，等待用户手动输入密码 |
| –password < password > | 直接将密码写在命令行中 |
| –username < username > | 指定用户名 |
| –verbose | 显示Sqoop任务更多执行信息 |
| –connection-param-file < filename > | 可选的参数，用于提供连接参数 |
| –relaxed-isolation | 设置每个mapmer的连接事务隔离 |

　　表20-验证相关的参数

| **参数** | **描述** |
| --- | --- |
| –valid | 启动数据验证功能，只对单表copy有效 |
| –validator < class-name > | 指定验证类 |
| –validation-threshold < class-name > | 指定使用的阈值验证类 |
| –validation-failurehandler < class-name > | 指定验证失败的处理类 |

　　表21-export控制参数

| **参数** | **描述** |
| --- | --- |
| –columns < col,col,col… > | 指定导出字段 |
| –direct | 使用direct模式 |
| –export-dir < dir > | 导出HDFS文件路径 |
| -m, –num-mappers < n > | 指定并行导出数据的map个数，默认为4个 |
| –table < table-name > | 指定database中的表名 |
| –call < stored-proc-name > | 调用的存储过程名称 |
| –update-key < col-name > | UPDATE模式下的更新字段，多个字段可以用逗号分隔 |
| –update-mode < mode > | 当某记录在数据库中找不到对应记录时的操作。默认为updateonly，可以设置成allowinsert，这个模式遇到新增记录会插入导出表中 |
| –input-null-string < null-string > | string类型字段表示null值的字符 |
| –input-null-non-string < null-string > | 非string类型字段表示null值的字符 |
| –staging-table < staging-table-name > | 在最终导出到目标表之前临时存储这些记录的表 |
| –clear-staging-table | 表示可以清除上面的staging-table表中的记录 |
| –batch | 对底层数据库的插入操作使用batch模式 |

　　--export-dir参数，以及--table或者--call是必须指定的参数。因为这三个参数分别用来指定导出数据集，以及数据导出后的去处，既可以是导出到某张表，也可以是对每一条导出记录调用存储过程。   
　　默认情况下，会导出所有字段。可以使用--columns参数指定部分字段进行导出。多个字段使用逗号分隔，比如--columns "col1,col2,col3"。需要注意的是，对那些没有指定的导出字段，在数据库中要么有默认值，要么运行为NULL。否则导出过程在插入这些字段的值时会报错。   
　　在导出时可以指定并行的Map个数，导出过程的速度由并发数来决定。默认情况下，export过程会使用4个Map并行。具体的并发数由实际的数据量大小，数据库的性能等共同确定。由参数--num-mappers或者-m来设置。   
　　有些数据库提供一中direct模式，比如MySQL。使用--direct参数可以指定特定的数据库连接方式。具体可以参考[24. Notes for specific connectors](http://blog.csdn.net/dabokele/article/details/51868594" \l "t93" \t "http://blog.csdn.net/dabokele/article/details/_blank)。   
　　对于数据库中的NULL值定义，可以由参数--input-null-string和--input-null-none-string两个参数来指定。如果不指定--input-null-string字段的话，那么导出数据中的null字符串会当做NULL值插入数据库中。如果不指定--input-null-non-string的话，那么导出数据中的null字符串以及空字符串都会当做NULL值插入数据库中。这里需要注意对非string类型字段，会将空字符串直接导出为NULL值。   
　　由于Sqoop将导出过程切分成了多个transaction，那么就会有可能某个导出job失败而导致只有部分数据提交到了导出数据库中。当失败job重试时，就有可能会出现数据重复，或者导出数据冲突等情况发生。这时可以指定一个--staging-table参数来避免这种情况的发生。导出的数据首先会缓存在该表中，最后等job执行成功后会将该表中的数据移动到最终目标表中。   
　　使用--staging-table需要在job执行前提前创建该表，这张表在结构上需要与目标表保持一致。并且这张表在任务执行前需要为空，或者指定--clear-staging-table参数，如果在导出前该表中有数据，那么这个参数会提前将该表清空。

--staging-table参数不能在--direct模式下工作。并且如果指定了--update-key或者指定的存储过程会涉及到数据的插入时，也不能使用--staging-table参数。

## 9.3. Inserts vs. Updates

　　默认情况下，export操作会将记录插入到导出表中。每一条导出记录都会转化成一条INSERT语句。如果导出表有字段约束(比如主键约束)时，需要注意避免导出记录违反这些约束条件的限制。这种模式只适合将数据导出到一张新的，或者空表中。   
　　如果指定--update-key参数，Sqoop会将每一条导出记录转化成一条UPDATE语句，这样可以更新已存在的记录。比如，看一下下面的表定义语句：

CREATE TABLE foo(

id INT NOT NULL PRIMARY KEY,

msg VARCHAR(32),

bar INT);

　　如果有一个HDFS文件内容如下：

0,this is a test,421,some more data,100...

　　运行sqoop-export --table foo --update-key id --export-dir /path/to/data --connect …，会执行一个export任务，这个任务执行的SQL语句如下，

UPDATE foo SET msg='this is a test', bar=42 WHERE id=0;

UPDATE foo SET msg='some more data', bar=100 WHERE id=1;...

　　如果一条UPDATE语句匹配不到对应的记录或者匹配到多条记录时，不会发生任何的错误。导出过程会正常继续执行后面的SQL语句。不过如果在使用--update-key参数的同时指定参数--update-mode为allowinsert的话，UPDATE语句匹配不到的记录就会以INSERT的形式插入到目标表中。   
　　--update-key参数也可以设置多个字段，多个字段之间用逗号隔开。

　　表22-输入记录格式化参数

| **参数** | **描述** |
| --- | --- |
| –input-enclosed-by < char> | 指定必须的字段外包字符 |
| –input-escaped-by < char > | 指定输入字段忽略字符 |
| –input-fields-terminated-by < char > | 输入文件字段分隔符 |
| –input-lines-terminated-by < char > | 输入文件行分隔符 |
| –input-optionally-enclosed-by < char > | 设置可选的字段外包字符 |

　　表23-输出记录格式化参数

| **参数** | **描述** |
| --- | --- |
| –enclosed-by < char> | 指定必须的字段外包字符 |
| –escaped-by < char > | 指定字段忽略字符 |
| –fields-terminated-by < char > | 字段分隔符 |
| –lines-terminated-by < char > | 行分隔符 |
| –mysql-delimiters | 使用MySQL默认分隔符，字段用,分隔，行用\n分隔，可选外包字符为'，忽略的字符为\ |
| –optionally-enclosed-by < char > | 设置可选的字段外包字符 |

　　Sqoop会根据上面设置的各种分隔符自动生成代码来解析输入文件。在非默认的情况下，需要设置这些参数来帮助Sqoop正确的解析输入文件中的记录。如果分隔符指定错误的话，可能会出现对一行记录解析不到足够的字段数，从而导致报出ParseExceptions的错误。

　　表24-代码生成参数

| **参数** | **描述** |
| --- | --- |
| –bindir < dir > | 指定编译后的class文件路径 |
| –class-name < name > | 指定生成代码的类名，这个会覆盖--package-name参数的设置。当与--jar-file配合使用时，指定使用的类名 |
| –jar-file < file > | 关闭代码生成，使用指定的jar文件 |
| –outdir < dir > | 设置生成java文件路径 |
| –package-name < name > | 设置生成代码的package名 |
| –map-column-java < m > | 设置生成代码中指定字段的Java类型 |

　　如果需要导出的数据在前面已经由import导入过，那么可以使用import时生成的代码来进行export的过程。这时可以通过--jar-file以及--class-name来指定使用的jar文件以及需要使用的类名。   
　　使用已有的代码，会与--update-key参数相冲突。UPDATE模式需要生成新的代码来完成这一过程，如果使用UPDATE模式，就不能使用--jar-file参数，并且必须指定所有没有默认值的分隔符。 

## 9.4. Exports and Transactions

　　数据导出过程由多个writers并行的执行，每一个writer使用一个独立的数据库连接，这些writer的transaction也是各自独立的。Sqoop使用多个INSERT语句来插入记录，每一个stattement最多能插入100条记录。每个writer在100个statements时，执行一次提交操作，所以，每一次提交可以插入10000条记录。因此导出操作不是原子性的，在导出过程结束前可以看到部分导出数据。

## 9.5. Failed Exports

　　数据导出过程可能会由于以下原因导致导出失败：

* Hadoop集群失去与数据库的连接(可能由于硬件原因，或者网络等原因造成)
* 在INSERT模式下向有约束的表插入数据
* HDFS文件中记录本身有异常，数据无法正常解析
* 分隔符指定错误，导致解析发生错误

硬件原因，比如磁盘空间不足或内存不足等

　　如果一个export的数据导出的map任务发生失败，将会导致整个export任务失败。并且失败的export过程的结果是未知的。每个map任务在独立的transaction中并发的导出数据，并且，每一个map任务定期的提交当前transaction中的记录。如果一个map任务失败，当前transaction将会回滚。而之前正常提交的transaction中的记录已经导出到外部表中了。

## 9.6. Example Invocations

　　下面的命令将数据导出到一个表bar中：

$ sqoop export --connect jdbc:mysql://db.example.com/foo --table bar \

--export-dir /results/bar\_data

　　上面命令从/results/bar\_data文件中读取记录，通过指定的jdbc连接串连接到数据库，将该数据导出到mysql中的表bar中。在任务执行前bar表必须已经存在。Sqoop会根据上面的命令生成一系列的INSERT INTO操作。   
　　如果想要导出到指定字段，可以使用--columns "col1,col2,col3"。没有指定在--columns参数中的字段，要么在定义时有默认值，要么允许为NULL，否则导出会发生错误。

　　另外一个加建议条件的导出过程：

$ sqoop export --connect jdbc:mysql://db.example.com/foo --table bar \

--export-dir /results/bar\_data --validate

　　再看一个不导出到表而是调用存储过程的示例：

$ sqoop export --connect jdbc:mysql://db.example.com/foo --call barproc \

--export-dir /results/bar\_data

# 10. validation

## 10.1. Purpose

　　校验操作会对不管是import过程还是export过程，都进行源数据和目标数据的比较。

## 10.2. Introduction

　　有关校验的过程，有以下三个基本概念：

* 校验阈值：确定源数据和目标数据之间允许出现的记录不一致情况的程度。默认为AbsoluteValidationThreshold，表示需要源数据和目标数据记录数保持一致。
* 校验失败的处理方法：当出现校验失败时如何进行处理，默认情况下使用的是LogOnFailureHandler，会记录一个提示信息到日志文件中。
* 校验器：校验的主要逻辑，默认使用的是RowCountValidator，用于比较源数据和目标数据的记录数。

## 10.3. Syntax

　　语法格式：

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

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

　　从前面的参数表中可以看到校验参数是import或者export参数的一部分。

## 10.4. Configuration

　　校验框架是灵活可拔插的。它提供了一些默认的实现方法，用户也可以基于接口的定义来实现自定义的类和参数。

* Validator，默认实现类是org.apache.sqoop.validation.RowCountValidator
* Validation Threshold，默认实现类是org.apache.sqoop.validation.AbsoluteValidationThreshold
* Validation Failure Handler，默认实现类是org.apache.sqoop.validation.AbortOnFailureHandler

## 10.5. Limitations

　　目前校验过程只能校验单表导入到HDFS这一种场景，下面列举了校验过程的一些不足之处：

* 多表模式不支持
* 自定义sql查询语句不支持
* 数据导入到Hive, Hbase, Accumulo不支持
* 带--where参数的不支持
* 增量导入的不支持

## 10.6. Example Invocations

　　从corp库导入一张EMPLOYEES表，校验记录数是否一致：

$ sqoop import --connect jdbc:mysql://db.foo.com/corp \

--table EMPLOYEES --validate

　　导出数据到表bar中，并启动校验功能：

$ sqoop export --connect jdbc:mysql://db.example.com/foo --table bar \

--export-dir /results/bar\_data --validate

　　不使用默认校验参数的示例：

$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table EMPLOYEES \

--validate --validator org.apache.sqoop.validation.RowCountValidator \

--validation-threshold \

org.apache.sqoop.validation.AbsoluteValidationThreshold \

--validation-failurehandler \

org.apache.sqoop.validation.AbortOnFailureHandler

# 11. Saved Jobs

# 12. sqoop-job

## 12.1. Purpose

## 12.2. Syntax

## 12.3. Saved jobs and passwords

## 12.4. Saved jobs and incremental imports

# 13. sqoop-metastore

## 13.1. Purpose

## 13.2. Syntax

# 14. sqoop-merge

## 14.1. Purpose

　　在Sqoop中提供了一个merge工具，用于将两个dataset中的记录进行合并操作。这个经常用于增量同步中，比如有一个全量的dataset和一个增量的dataset，执行merge时希望如果增量和全量中都有的话优先取增量dataset中的记录，如果只有增量中有，则取增量中的记录，如果只有全量中有，则取全量中的记录。 

## 14.2. Syntax

　　在$SQOOP\_HOME/bin路径下有一个sqoop-merge命令，其中的主要逻辑是执行sqoop merge操作。所以，merge有两种使用方法

$ sqoop merge (generic-args) (merge-args)

$ sqoop-merge (generic-args) (merge-args)

　　参数介绍

| **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 |

　　注意这里的三个路径都是HDFS上的路径，并且要求--target-dir对应的路径不存在，存在的话就会报错。可以理解成，将--new-data中的数据合并到--onto文件中。当Sqoop合并两个dataset时，会假设每条记录有一个唯一主键，该主键由参数--merge-key指定。比如下面两个dataset   
dataset-onto:

| **id** | **comment** |
| --- | --- |
| 1 | xxx |
| 3 | zzz |

dataset-new-data:

| **id** | **comment** |
| --- | --- |
| 1 | aaa |
| 2 | yyy |

如果指定--merge-key id，那么就会根据id去对两个dataset进行join，最终结果   
dataset-target-dir

| **id** | **comment** |
| --- | --- |
| 1 | aaa |
| 2 | yyy |
| 3 | zzz |

　　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。   
　　merge tool一般用于增量import之后。假设有两个增量import执行，前面一个任务的HDFS路径名为older，后面那个任务的HDFS路径为newer，那么这两个文件的merge操作可以使用以下命令

$ sqoop merge --new-data newer --onto older --target-dir merged \

--jar-file datatypes.jar --class-name Foo --merge-key id

这个命令会运行一个MapReduce任务，根据id字段order和newner的每一行记录进行join。newner中的记录优先于older中的记录使用。   
　　merge tool可以用于SequenceFile, Avro, 以及text类型的import操作。但是需要注意的是older和newner的文件类型必须保持一致。

# 15. sqoop-codegen

## 15.1. Purpose

## 15.2. Syntax

## 15.3. Example Invocations

# 16. sqoop-create-hive-table

## 16.1. Purpose

## 16.2. Syntax

## 16.3. Example Invocations

# 17. sqoop-eval

## 17.1. Purpose

## 17.2. Syntax

## 17.3. Example Invocations

# 18. sqoop-list-databases

## 18.1. Purpose

## 18.2. Syntax

## 18.3. Example Invocations

# 19. sqoop-list-tables

## 19.1. Purpose

## 19.2. Syntax

## 19.3. Example Invocations

# 20. sqoop-help

## 20.1. Purpose

## 20.2. Syntax

## 20.3. Example Invocations

# 21. sqoop-version

## 21.1. Purpose

## 21.2. Syntax

## 21.3. Example Invocations

# 22. Sqoop-HCatalog Integration

## 22.1. HCatalog Background

## 22.2. Exposing HCatalog Tables to Sqoop

**22.2.1. New Command Line Options**   
**22.2.2. Supported Sqoop Hive Options**   
**22.2.3. Direct Mode support**   
**22.2.4. Unsupported Sqoop Options**   
22.2.4.1. Unsupported Sqoop Hive Import Options   
22.2.4.2. Unsupported Sqoop Export and Import Options   
**22.2.5. Ignored Sqoop Options**

## 22.3. Automatic Table Creation

## 22.4. Delimited Text Formats and Field and Line Delimiter Characters

## 22.5. HCatalog Table Requirements

## 22.6. Support for Partitioning

## 22.7. Schema Mapping

## 22.8. Support for HCatalog Data Types

## 22.9. Providing Hive and HCatalog Libraries for the Sqoop Job

## 22.10. Examples

## 22.11. Import

## 22.12. Export

# 23. Compatibility Notes

## 23.1. Supported Databases

## 23.2. MySQL

**23.2.1. zeroDateTimeBehavior**   
**23.2.2. UNSIGNED columns**   
**23.2.3. BLOB and CLOB columns**   
**23.2.4. Importing views in direct mode**

## 23.3. PostgreSQL

**23.3.1. Importing views in direct mode**

## 23.4. Oracle

**23.4.1. Dates and Times**

## 23.5. Schema Definition in Hive

# 24. Notes for specific connectors

## 24.1. MySQL JDBC Connector

**24.1.1. Upsert functionality**

## 24.2. MySQL Direct Connector

**24.2.1. Requirements**   
**24.2.2. Limitations**   
**24.2.3. Direct-mode Transactions**

## 24.3. Microsoft SQL Connector

**24.3.1. Extra arguments**   
**24.3.2. Schema support**   
**24.3.3. Table hints**

## 24.4. PostgreSQL Connector

**24.4.1. Extra arguments**   
**24.4.2. Schema support**

## 24.5. PostgreSQL Direct Connector

**24.5.1. Requirements**   
**24.5.2. Limitations**

## 24.6. pg\_bulkload connector

**24.6.1. Purpose**   
**24.6.2. Requirements**   
**24.6.3. Syntax**   
**24.6.4. Data Staging**

## 24.7. Netezza Connector

**24.7.1. Extra arguments**   
**24.7.2. Direct Mode**   
**24.7.3. Null string handling**

## 24.8. Data Connector for Oracle and Hadoop

**24.8.1. About**   
24.8.1.1. Jobs   
24.8.1.2. How The Standard **[oracle](http://lib.csdn.net/base/oracle" \o "Oracle知识库" \t "http://blog.csdn.net/dabokele/article/details/_blank)** Manager Works for Imports   
24.8.1.3. How The Data Connector for Oracle and Hadoop Works for Imports   
24.8.1.4. Data Connector for Oracle and Hadoop Exports   
**24.8.2. Requirements**   
24.8.2.1. Ensure The Oracle Database JDBC Driver Is Setup Correctly   
24.8.2.2. Oracle Roles and Privileges   
24.8.2.3. Additional Oracle Roles And Privileges Required for Export   
24.8.2.4. Supported Data Types   
**24.8.3. Execute Sqoop With Data Connector for Oracle and Hadoop**   
24.8.3.1. Connect to Oracle / Oracle RAC   
24.8.3.2. Connect to An Oracle Database Instance   
24.8.3.3. Connect to An Oracle RAC   
24.8.3.4. Login to The Oracle Instance   
24.8.3.5. Kill Data Connector for Oracle and Hadoop Jobs   
**24.8.4. Import Data from Oracle**   
24.8.4.1. Match Hadoop Files to Oracle Table Partitions   
24.8.4.2. Specify The Partitions To Import   
24.8.4.3. Consistent Read: All Mappers Read From The Same Point In Time   
**24.8.5. Export Data into Oracle**   
24.8.5.1. Insert-Export   
24.8.5.2. Update-Export   
24.8.5.3. Merge-Export   
24.8.5.4. Create Oracle Tables   
24.8.5.5. NOLOGGING   
24.8.5.6. Partitioning   
24.8.5.7. Match Rows Via Multiple Columns   
24.8.5.8. Storage Clauses   
**24.8.6. Manage Date And Timestamp Data Types**   
24.8.6.1. Import Date And Timestamp Data Types from Oracle   
24.8.6.2. The Data Connector for Oracle and Hadoop Does Not Apply A Time Zone to DATE / TIMESTAMP Data Types   
24.8.6.3. The Data Connector for Oracle and Hadoop Retains Time Zone Information in TIMEZONE Data Types   
24.8.6.4. Data Connector for Oracle and Hadoop Explicitly States Time Zone for LOCAL TIMEZONE Data Types   
24.8.6.5. java.sql.Timestamp   
24.8.6.6. Export Date And Timestamp Data Types into Oracle   
**24.8.7. Configure The Data Connector for Oracle and Hadoop**   
24.8.7.1. oraoop-site-template.xml   
24.8.7.2. oraoop.oracle.session.initialization.statements   
24.8.7.3. oraoop.table.import.where.clause.location   
24.8.7.4. oracle.row.fetch.size   
24.8.7.5. oraoop.import.hint   
24.8.7.6. oraoop.oracle.append.values.hint.usage   
24.8.7.7. mapred.map.tasks.speculative.execution   
24.8.7.8. oraoop.block.allocation   
24.8.7.9. oraoop.import.omit.lobs.and.long   
24.8.7.10. oraoop.locations   
24.8.7.11. sqoop.connection.factories   
24.8.7.12. Expressions in oraoop-site.xml   
**24.8.8. Troubleshooting The Data Connector for Oracle and Hadoop**   
24.8.8.1. Quote Oracle Owners And Tables   
24.8.8.2. Quote Oracle Columns   
24.8.8.3. Confirm The Data Connector for Oracle and Hadoop Can Initialize The Oracle Session   
24.8.8.4. Check The Sqoop Debug Logs for Error Messages   
24.8.8.5. Export: Check Tables Are Compatible   
24.8.8.6. Export: Parallelization   
24.8.8.7. Export: Check oraoop.oracle.append.values.hint.usage   
24.8.8.8. Turn On Verbose

# 25. Getting Support

# 26. Troubleshooting

## 26.1. General Troubleshooting Process

## 26.2. Specific Troubleshooting Tips

**26.2.1. Oracle: Connection Reset Errors**   
**26.2.2. Oracle: Case-Sensitive Catalog Query Errors**   
**26.2.3. MySQL: Connection Failure**   
**26.2.4. Oracle: ORA-00933 error (SQL command not properly ended)**   
**26.2.5. MySQL: Import of TINYINT(1) from MySQL behaves strangely**