**Sqoop-1.4.4工具import和export使用详解**

Sqoop可以在HDFS/Hive和关系型数据库之间进行数据的导入导出，其中主要使用了import和export这两个工具。这两个工具非常强大，提供了很多选项帮助我们完成数据的迁移和同步。比如，下面两个潜在的需求：

1. 业务数据存放在关系数据库中，如果数据量达到一定规模后需要对其进行分析或同统计，单纯使用关系数据库可能会成为瓶颈，这时可以将数据从业务数据库数据导入（import）到Hadoop平台进行离线分析。
2. 对大规模的数据在Hadoop平台上进行分析以后，可能需要将结果同步到关系数据库中作为业务的辅助数据，这时候需要将Hadoop平台分析后的数据导出（export）到关系数据库。

这里，我们介绍Sqoop完成上述基本应用场景所使用的import和export工具，通过一些简单的例子来说明这两个工具是如何做到的。

工具通用选项

import和export工具有些通用的选项，如下表所示：

|  |  |
| --- | --- |
| 选项 | 含义说明 |
| --connect <jdbc-uri> | 指定JDBC连接字符串 |
| --connection-manager <class-name> | 指定要使用的连接管理器类 |
| --driver <class-name> | 指定要使用的JDBC驱动类 |
| --hadoop-mapred-home <dir> | 指定$HADOOP\_MAPRED\_HOME路径 |
| --help | 打印用法帮助信息 |
| --password-file | 设置用于存放认证的密码信息文件的路径 |
| -P | 从控制台读取输入的密码 |
| --password <password> | 设置认证密码 |
| --username <username> | 设置认证用户名 |
| --verbose | 打印详细的运行信息 |
| --connection-param-file <filename> | 可选，指定存储数据库连接参数的属性文件 |

数据导入工具import

import工具，是将HDFS平台外部的结构化存储系统中的数据导入到Hadoop平台，便于后续分析。我们先看一下import工具的基本选项及其含义，如下表所示：

|  |  |
| --- | --- |
| 选项 | 含义说明 |
| --append | 将数据追加到HDFS上一个已存在的数据集上 |
| --as-avrodatafile | 将数据导入到Avro数据文件 |
| --as-sequencefile | 将数据导入到SequenceFile |
| --as-textfile | 将数据导入到普通文本文件（默认） |
| --boundary-query <statement> | 边界查询，用于创建分片（InputSplit） |
| --columns <col,col,col…> | 从表中导出指定的一组列的数据 |
| --delete-target-dir | 如果指定目录存在，则先删除掉 |
| --direct | 使用直接导入模式（优化导入速度） |
| --direct-split-size <n> | 分割输入stream的字节大小（在直接导入模式下） |
| --fetch-size <n> | 从数据库中批量读取记录数 |
| --inline-lob-limit <n> | 设置内联的LOB对象的大小 |
| -m,--num-mappers <n> | 使用n个map任务并行导入数据 |
| -e,--query <statement> | 导入的查询语句 |
| --split-by <column-name> | 指定按照哪个列去分割数据 |
| --table <table-name> | 导入的源表表名 |
| --target-dir <dir> | 导入HDFS的目标路径 |
| --warehouse-dir <dir> | HDFS存放表的根路径 |
| --where <where clause> | 指定导出时所使用的查询条件 |
| -z,--compress | 启用压缩 |
| --compression-codec <c> | 指定Hadoop的codec方式（默认gzip） |
| --null-string <null-string> | 果指定列为字符串类型，使用指定字符串替换值为null的该类列的值 |
| --null-non-string <null-string> | 如果指定列为非字符串类型，使用指定字符串替换值为null的该类列的值 |

下面，我们通过实例来说明，在实际中如何使用这些选项。

* 将MySQL数据库中整个表数据导入到Hive表

|  |  |
| --- | --- |
| 1 | bin/sqoop import --connect jdbc:mysql://10.95.3.49:3306/workflow --table project --username shirdrn -P --hive-import -- --default-character-set=utf-8 |

* 将MySQL数据库workflow中project表的数据导入到Hive表中。
* 将MySQL数据库中多表JION后的数据导入到HDFS

|  |  |
| --- | --- |
| 1 | bin/sqoop import --connect jdbc:mysql://10.95.3.49:3306/workflow --username shirdrn -P --query 'SELECT users.\*, tags.tag FROM users JOIN tags ON (users.id = tags.user\_id) WHERE $CONDITIONS' --split-byusers.id --target-dir /hive/tag\_db/user\_tags -- --default-character-set=utf-8 |

* 这里，使用了--query选项，不能同时与--table选项使用。而且，变量$CONDITIONS必须在WHERE语句之后，供Sqoop进程运行命令过程中使用。上面的--target-dir指向的其实就是Hive表存储的数据目录。
* 将MySQL数据库中某个表的数据增量同步到Hive表

|  |  |
| --- | --- |
| 1 | bin/sqoop job --create your-sync-job -- import --connect jdbc:mysql://10.95.3.49:3306/workflow --table project --username shirdrn -P --hive-import --incremental append --check-column id --last-value 1 -- --default-character-set=utf-8 |

* 这里，每次运行增量导入到Hive表之前，都要修改--last-value的值，否则Hive表中会出现重复记录。
* 将MySQL数据库中某个表的几个字段的数据导入到Hive表

|  |  |
| --- | --- |
| 1 | bin/sqoop import --connect jdbc:mysql://10.95.3.49:3306/workflow --username shirdrn --P --table tags --columns 'id,tag' --create-hive-table -target-dir /hive/tag\_db/tags -m 1 --hive-table tags --hive-import -- --default-character-set=utf-8 |

* 我们这里将MySQL数据库workflow中tags表的id和tag字段的值导入到Hive表tag\_db.tags。其中--create-hive-table选项会自动创建Hive表，--hive-import选项会将选择的指定列的数据导入到Hive表。如果在Hive中通过SHOW TABLES无法看到导入的表，可以在conf/hive-site.xml中显式修改如下配置选项：

|  |  |
| --- | --- |
| 1 | <property> |
| 2 | <name>javax.jdo.option.ConnectionURL</name> |

|  |  |
| --- | --- |
| 3 | <value>jdbc:derby:;databaseName=hive\_metastore\_db;create=true</value> |
| 4 | </property> |

* 然后再重新运行，就能看到了。
* 使用验证配置选项

|  |  |
| --- | --- |
| 1 | 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 |

* 上面这个是官方用户手册上给出的用法，我们在实际中还没用过这个，有感兴趣的可以验证尝试一下。

数据导出工具export

export工具，是将HDFS平台的数据，导出到外部的结构化存储系统中，可能会为一些应用系统提供数据支持。我们看一下export工具的基本选项及其含义，如下表所示：

|  |  |
| --- | --- |
| 选项 | 含义说明 |
| --validate <class-name> | 启用数据副本验证功能，仅支持单表拷贝，可以指定验证使用的实现类 |
| --validation-threshold <class-name> | 指定验证门限所使用的类 |
| --direct | 使用直接导出模式（优化速度） |
| --export-dir <dir> | 导出过程中HDFS源路径 |
| -m,--num-mappers <n> | 使用n个map任务并行导出 |
| --table <table-name> | 导出的目的表名称 |
| --call <stored-proc-name> | 导出数据调用的指定存储过程名 |
| --update-key <col-name> | 更新参考的列名称，多个列名使用逗号分隔 |
| --update-mode <mode> | 指定更新策略，包括：updateonly（默认）、allowinsert |
| --input-null-string <null-string> | 使用指定字符串，替换字符串类型值为null的列 |
| --input-null-non-string <null-string> | 使用指定字符串，替换非字符串类型值为null的列 |
| --staging-table <staging-table-name> | 在数据导出到数据库之前，数据临时存放的表名称 |
| --clear-staging-table | 清除工作区中临时存放的数据 |
| --batch | 使用批量模式导出 |

下面，我们通过实例来说明，在实际中如何使用这些选项。这里，我们主要结合一个实例，讲解如何将Hive中的数据导入到MySQL数据库。  
首先，我们准备几个表，MySQL数据库为tag\_db，里面有两个表，定义如下所示：

CREATE TABLE tag\_db.users (

id INT(11) NOT NULL AUTO\_INCREMENT,name VARCHAR(100) NOT NULL,PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE tag\_db.tags (

id INT(11) NOT NULL AUTO\_INCREMENT,

user\_id INT NOT NULL,

tag VARCHAR(100) NOT NULL,

PRIMARY KEY (`id`)) ENGINE=InnoDB DEFAULT CHARSET=utf8;

这两个表中存储的是基础数据，同时对应着Hive中如下两个表：

CREATE TABLE users (id INT,name STRING);

CREATE TABLE tags (id INT,user\_id INT,tag STRING);

我们首先在上述MySQL的两个表中插入一些测试数据：

|  |  |
| --- | --- |
| 1 | INSERT INTO tag\_db.users(name) VALUES('jeffery'); |
| 2 | INSERT INTO tag\_db.users(name) VALUES('shirdrn'); |

|  |  |
| --- | --- |
| 3 | INSERT INTO tag\_db.users(name) VALUES('sulee'); |
| 4 |  |

|  |  |
| --- | --- |
| 5 | INSERT INTO tag\_db.tags(user\_id, tag) VALUES(1, 'Music'); |
| 6 | INSERT INTO tag\_db.tags(user\_id, tag) VALUES(1, 'Programming'); |

|  |  |
| --- | --- |
| 7 | INSERT INTO tag\_db.tags(user\_id, tag) VALUES(2, 'Travel'); |
| 8 | INSERT INTO tag\_db.tags(user\_id, tag) VALUES(3, 'Sport'); |

然后，使用Sqoop的import工具，将MySQL两个表中的数据导入到Hive表，执行如下命令行：

|  |  |
| --- | --- |
| 1 | bin/sqoop import --connect jdbc:mysql://10.95.3.49:3306/tag\_db --table users --username shirdrn -P --hive-import -- --default-character-set=utf-8 |
| 2 | bin/sqoop import --connect jdbc:mysql://10.95.3.49:3306/tag\_db --table tags --username shirdrn -P --hive-import -- --default-character-set=utf-8 |

导入成功以后，再在Hive中创建一个用来存储users和tags关联后数据的表：

|  |  |
| --- | --- |
| 1 | CREATE TABLE user\_tags ( |
| 2 | id STRING, |

|  |  |
| --- | --- |
| 3 | name STRING, |
| 4 | tag STRING |

|  |  |
| --- | --- |
| 5 | ); |

执行如下HQL语句，将关联数据插入user\_tags表：

|  |  |
| --- | --- |
| 1 | FROM users u JOIN tags t ON u.id=t.user\_id INSERT INTO TABLE user\_tags SELECT CONCAT(CAST(u.id AS STRING),CAST(t.id AS STRING)), u.name, t.tag; |

将users.id与tags.id拼接的字符串，作为新表的唯一字段id，name是用户名，tag是标签名称。  
再在MySQL中创建一个对应的user\_tags表，如下所示：

|  |  |
| --- | --- |
| 1 | CREATE TABLE tag\_db.user\_tags ( |
| 2 | id varchar(200) NOT NULL, |

|  |  |
| --- | --- |
| 3 | name varchar(100) NOT NULL, |
| 4 | tag varchar(100) NOT NULL |

|  |  |
| --- | --- |
| 5 | ); |

使用Sqoop的export工具，将Hive表user\_tags的数据同步到MySQL表tag\_db.user\_tags中，执行如下命令行：

|  |  |
| --- | --- |
| 1 | bin/sqoop export --connect jdbc:mysql://10.95.3.49:3306/tag\_db --username shirdrn --P --table user\_tags --export-dir /hive/user\_tags --input-fields-terminated-by '\001' -- --default-character-set=utf-8 |

执行导出成功后，可以在MySQL的tag\_db.user\_tags表中看到对应的数据。  
如果在导出的时候出现类似如下的错误：

14/02/27 17:59:06 INFO mapred.JobClient: Task Id : attempt\_201402260008\_0057\_m\_000001\_0, Status : FAILED

java.io.IOException: Can't export data, please check task tracker logs

    at org.apache.sqoop.mapreduce.TextExportMapper.map(TextExportMapper.java:112)

     at org.apache.sqoop.mapreduce.TextExportMapper.map(TextExportMapper.java:39)

     at org.apache.hadoop.mapreduce.Mapper.run(Mapper.java:145)

     at org.apache.sqoop.mapreduce.AutoProgressMapper.run(AutoProgressMapper.java:64)

     at org.apache.hadoop.mapred.MapTask.runNewMapper(MapTask.java:764)

     at org.apache.hadoop.mapred.MapTask.run(MapTask.java:364)

     at org.apache.hadoop.mapred.Child$4.run(Child.java:255)

     at java.security.AccessController.doPrivileged(Native Method)

     at javax.security.auth.Subject.doAs(Subject.java:396)

     at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1190)

     at org.apache.hadoop.mapred.Child.main(Child.java:249)

Caused by: java.util.NoSuchElementException

     at java.util.AbstractList$Itr.next(AbstractList.java:350)

     at user\_tags.\_\_loadFromFields(user\_tags.java:225)

     at user\_tags.parse(user\_tags.java:174)

     at org.apache.sqoop.mapreduce.TextExportMapper.map(TextExportMapper.java:83)

     ... 10 more

通过指定字段分隔符选项--input-fields-terminated-by，指定Hive中表字段之间使用的分隔符，供Sqoop读取解析，就不会报错了。