# 1 配置hive

## 1.1 安装hive

下载

从apache官网下载新版本hive，要注意和hadoop版本的匹配。

安装

需要提前安装好jdk和hadoop，并且配置过JAVA\_HOME和HADOOP\_HOME

解压即可安装

可以进入bin下执行hive脚本，如果能够进入hive命令行说明hive安装成功

配置环境变量：

HIVE\_INSTALL=/opt/apache-hive-1.2.1-bin

PATH=$JAVA\_HOME/bin:$HIVE\_INSTALL/bin:$PATH

## 1.2 安装mysql

安装Percona分支版本的MySQL。性能超越标准的MySQL。

### 上传解压

首先安装 cmake

yum –y install cmake //也需要安装gcc-c++,openssl openssl-devel。前面已经安装。

cd /usr/local/src/

mkdir mysql

cd mysql /

rz 上传安装包

tar -xvf Percona-Server-5.6.24-72.2-r8d0f85b-el6-x86\_64-bundle.tar

rpm -ivh Percona-Server-56-debuginfo-5.6.24-rel72.2.el6.x86\_64.rpm

rpm -ivh Percona-Server-shared-56-5.6.24-rel72.2.el6.x86\_64.rpm

rpm -ivh Percona-Server-client-56-5.6.24-rel72.2.el6.x86\_64.rpm

rpm –ivh Percona-Server-server-56-5.6.24-rel72.2.el6.x86\_64.rpm

安装顺序：rpm包很多，只需安装debuginfo 、shared、client、server

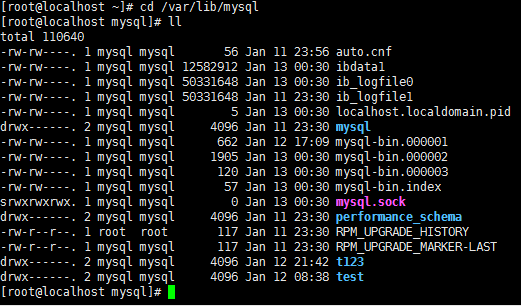
### 安装目录

配置文件路径 /etc/my.cnf

注意：安装完成后，会将配置文件放在etc目录下

vi /etc/my.cnf

数据文件路径：/var/lib/mysql



#修改my.cnf,默认在/usr/my.cnf

vim /usr/my.cnf

[client]

default-character-set=utf8

[mysql]

default-character-set=utf8

[mysqld]

character\_set\_server=utf8

### 启动服务

service mysql start //自动安装到它自己制定的目录下，注册服务为mysql

service mysql status

service mysql stop

配置开机启动：

chkconfig --list //展现开机的服务

### 修改root密码

默认没有密码不安全：

mysqladmin –u root password "root"



Warning警告提示，忽略即可。

登录：

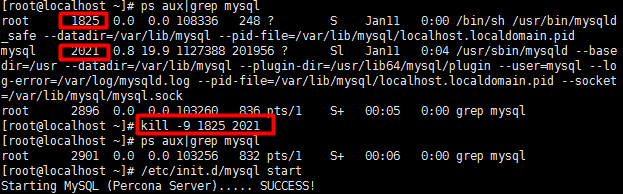
mysql –uroot –proot

### 问题：PID file could not be found

mysql无法启动ERROR! MySQL is running but PID file could not be found ?

Can't connect to local MySQL server through socket '/var/lib/mysql/mysql.sock'

解决办法：



ps aux | grep mysql

kill -9 pid1 pid2 #pid1,pid2为具体的查询出来的端口

service mysql start 或者 /etc/init.d/mysql start

### 错误：提示uuid重复

由于data拷贝是全目录拷贝，将/var/lib/mysql/auto.cnf也拷贝，它里面记录了对数据库的一个uuid标识，随便产生个新的uuid，替换掉新目录中的auto.cnf中的uuid串即可。

可以用select uuid()来产生新值，手工黏贴到auto.cnf文件中。

### 验证是否成功安装

use mysql //打开mysql数据库

show tables //查看mysql数据库下的表

注意：Percona安装和mysql正式版的安装，在依赖上有区别。

### 打开3306端口

/sbin/iptables -I INPUT -p tcp --dport 3306 -j ACCEPT

/etc/rc.d/init.d/iptables save #修改生效

/etc/init.d/iptables status #查看配置

### 执行数据库的脚本

创建库和数据库表及数据

mysql>use jtdb;

mysql>set names utf8; #防止乱码

mysql>source jtdb.sql;

### 开启MYSQL远程访问权限

语法：

grant [权限] on [数据库名].[表名] to ['用户名']@['web服务器的ip地址'] identified by ['密码'];

grant all on \*.\* to 'root'@'%' identified by 'root';

或者指定IP地址

grant all on \*.\* to 'root'@'192.168.1.103' identified by 'root';

或者启动权限：grant all privileges on \*.\* to 'root'@'%' identified by 'root' with grant option;

Hive默认使用Derby作为元数据库，但是这种方式只能用于测试不应用于生产环境中，生产环境中应该使用其他数据库作为元数据储存的数据库使用

替换默认的derby数据库为mysql

Hive目前支持Derby和MySql作为元数据库使用。

------------------

mysql在linux中的安装过程，参照文档：linux下mysql5.6.x安装.txt

------------------

删除hdfs中的/user/hive

hadoop fs -rmr /user/hive

复制hive/conf/hive-default.xml.template为hive-site.xml

cp hive-default.xml.template hive-site.xml

在<configuration>中进行配置

|  |
| --- |
| <configuration>  <propperty>  <name>javax.jdo.option.ConnectionURL</name>  <value>jdbc:mysql://hadoop01:3306/hive?createDatabaseIfNotExist=true</value>  <description>JDBC connect string for a JDBC metastore</description> </property>  <property>  <name>javax.jdo.option.ConnectionDriverName</name>  <value>com.mysql.jdbc.Driver</value>  <description>Driver class name for a JDBC metastore</description>  </property>  <property> <name>javax.jdo.option.ConnectionUserName</name>  <value>root</value> <description>username to use against metastore database</description> </property><property>  <name>javax.jdo.option.ConnectionPassword</name>  <value>root</value>  <description>password to use against metastore database</description> </property>  </configuration> |

## 1.3 元数据

配置过mysql元数据库后可能遇到的问题：

hive无法启动，提示缺少驱动包：将mysql的数据库驱动包拷贝到hive/lib下

hive无法启动，提示连接被拒绝：在mysql中启动权限

启动权限：grant all privileges on \*.\* to 'root'@'%' identified by 'root' with grant option;

hive能够启动，但是在执行一些操作时报奇怪的错误：原因很可能是mysql中hive库不是latin1编码，删除元数据库手动创建hive库，指定编码集为latin1

连接mysql，发现多了一个hive库。其中保存有hive的元数据。

DBS-数据库的元数据信息

DB\_ID 数据库编号

DB\_LOCATION\_URI 数据库在hdfs中的位置

TBLS-表信息。

TBL\_ID 表的编号

DB\_ID 属于哪个数据库

TBL\_NAME 表的名称

TBL\_TYPE 表的类型 内部表MANAGED\_TABLE/外部表

COLUMNS\_V2表中字段信息

CD\_ID 所属表的编号

COLUMN\_NAME 列明

TYPE\_NAME 列的类型名

SDS-表对应hdfs目录

CD\_ID 表的编号

LOCATION 表对应到hdfs中的位置

## 1.4 配置hive

Hive --config /usr/local/src/conf/hive-conf 重新定义hive查找hive-site.xml文件。

指向配置文件的目录，而不是配置文件本身。

也可也用HIVE\_CONF\_DIR环境变量来指定文件目录。

Hive-site.xml最适合存放详细的集群链接信息，可以使用hadoop属性fs.defalut.name和maperd.job.tracker指定文件系统和jobtarcker

### 1.4.1 多个hive用户共享hadoop

Hadoop fs -mkdir /tmp

Hadoop fs -chmod a+w /tmp

Hadoop fs -mkdir /user/hive/warehouse

Hadoop fs -chmod a+w /user/hive/warehouse

如果是同组用户吧仓库权限设置为g+w

## 1.5 HWI

启动HWI(只有在/opt/ant/lib中没有找到ant库时才需要设定ANT\_LIB变量)

Export ANT\_LIB=/path/to/ant/lib

Hive --service hwi

在浏览器中：http：//localhost:9999/hwi

### 1.4.2 set命令

确保表的定义都使用桶：

Hive>set hive .enforce.bucketing=true;

# 2 Hive的语法

## 2.1.数据类型

TINYINT - byte

SMALLINT - short

INT - int

BIGINT - long

BOOLEAN - boolean

FLOAT - float

DOUBLE - double

STRING - String

TIMESTAMP - TimeStamp

BINARY - byte[]

Array 一组有序字段，字段的类型必须相同

Map 一组无序键值对

Struct 一组命名的字段，字段类型可以不同。

## 2.2 create table

create table book (id bigint, name string) row format delimited fields terminated by '\t';

create external table book (id bigint, name string) row format delimited fields terminated by '\t' location 'xxxxx';

create table book (id bigint, name string) partitioned by (category string) row format delimited fields terminated by '\t';

2.3 修改表

增加分区:ALTER TABLE book add PARTITION (category = 'zazhi') location '/user/hive/warehouse/datax.db/book/category=zazhi';

删除分区:ALTER TABLE table\_name DROP partition\_spec, partition\_spec,...

重命名表:ALTER TABLE table\_name RENAME TO new\_table\_name

修改列:ALTER TABLE table\_name CHANGE [COLUMN] col\_old\_name col\_new\_name column\_type [COMMENT col\_comment] [FIRST|AFTER column\_name]

增加/替换列:ALTER TABLE table\_name ADD|REPLACE COLUMNS (col\_name data\_type [COMMENT col\_comment], ...)

修改表属性：

ALTER TABLE table\_name SET TBLPROPERTIES (property\_name = property\_value, property\_name = property\_value, ... );

案例：修改表book1的comment属性为this is a external table;

alter table book1 set tblproperties('comment'='this is a external table');

通过desc formatted book1;可查看到修改成功

## 2.4 .Show

查看数据库

SHOW DATABASES;

查看表名

SHOW TABLES;

查看表名，部分匹配

SHOW TABLES 'page.\*';

SHOW TABLES '.\*view';

查看某表的所有Partition，如果没有就报错：

SHOW PARTITIONS page\_view;

查看某表结构：

DESCRIBE invites;

查看分区内容

SELECT a.foo FROM invites a WHERE a.ds='2008-08-15';

查看有限行内容，同Greenplum，用limit关键词

SELECT a.foo FROM invites a limit 3;

查看表分区定义

DESCRIBE EXTENDED page\_view PARTITION (ds='2008-08-08');

## 2.5 .Load

LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)]

Load 操作只是单纯的复制/移动操作，将数据文件移动到 Hive 表对应的位置。

Load并不检查表目录中的文件是否为生命的模式，只有字查询时才知道，如果返回为NULL则可能为不匹配。

## 2.6.Insert

### 2.6.1 插入指定表

将行记录插入到表中：

INSERT INTO TABLE tablename [PARTITION (partcol1[=val1], partcol2[=val2] ...)] VALUES values\_row [, values\_row ...]

(1)将查询的结果插入指定表中

INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)] select\_statement1 FROM from\_statement

(2)将查询结果写入文件系统中

INSERT OVERWRITE [LOCAL] DIRECTORY directory1 SELECT ... FROM ...

### 2.6.2 多表插入

这种多表插入比单单表更搞笑，因为只需要扫描一边源表

From source

Insert overwrite table target1

Select col1，col2，count（distnct station）

Group by col1

Insert overwrite table target2

Select col1，col2；

## 2.7 .Drop

删除一个内部表的同时会同时删除表的元数据和数据。删除一个外部表，只删除元数据而保留数据。

## 2.8 .Limit

Limit 可以限制查询的记录数。查询的结果是随机选择的

## 2.9 .Select

**SELECT** [**ALL** | **DISTINCT**] select\_expr, select\_expr, ...  
 **FROM** table\_reference  
 [**WHERE** where\_condition]   
 [**GROUP BY** col\_list]  
 [ CLUSTER **BY** col\_list| [DISTRIBUTE **BY** col\_list]

[SORT **BY** col\_list]  
 ]  
 [LIMIT **number**]

## 2.10 .JOIN

join\_table:

table\_reference JOIN table\_factor [join\_condition]

| table\_reference {LEFT|RIGHT|FULL} [OUTER] JOIN table\_reference join\_condition

| table\_reference LEFT SEMI JOIN table\_reference join\_condition

table\_reference:

table\_factor

| join\_table

table\_factor:

tbl\_name [alias]

| table\_subquery alias

| ( table\_references )

join\_condition:

ON equality\_expression ( AND equality\_expression )\*

equality\_expression:

expression = expression

练习：

1、创建表student:

create table student(sid int,sname string,sage int,sdept string) partitioned by(sex string) row format delimited fields terminated by '\t';

2、创建表course:

create table course(cid int,cname string) row format delimited fields terminated by '\t';

3、创建表score:

create table score(sid int,cid int,score int) clustered by(sid) into 2 buckets row format delimited fields terminated by '\t';

4、加载数据：

load data local inpath '/root/student\_male.txt' overwrite into table student partition(sex='male');

load data local inpath '/root/student\_female.txt' overwrite into table student partition(sex='female');

load data local inpath '/root/score.txt' overwrite into table score;

load data local inpath '/root/course.txt' overwrite into table course;

5、查询所有学生的id和姓名,并根据学生id排序:

select sid,sname from student sort by sid;

6、统计女学生的人数:

select count(\*) from student where sex='female';

7、查询选修1号课程的成绩排名前3的学生id和成绩:

Order by 只用一个reduce排序，sort by 为每个reduce排序

select sid,score from score where cid=1 sort by score desc limit 3;

或者：

select sid,score from score where cid=1 order by score desc limit 3;

8、查询成绩有不及格（<60）的学生姓名:

select sname from student where sid in (select sid from score where score<60);

9、 求各个课程号及相应的选课人数:

select cid,count(1) from score group by cid;

10、计算每个学生的总成绩:

select sid,sum(score) from score group by sid;

11、查询选修了3门以上课程的学生学号:

select sid from (select sid,count(cid) num from score group by sid) a where a.num>3;

或者：

select sid from score group by sid having count(cid)>3;

12、查询有选修课程的学生姓名:

select distinct sname from student join score on student.sid=score.sid;

此处应注意join的优化：尽量把数据量大的表放在前边，数据量大时可以明显的看出性能的提升。

13、查询选修2号课程且成绩在90分以上的所有学生:

select student.sname,score.score from student join score on student.sid=score.sid where score.cid=2 and score.score>90;

14、查询与liu在同一个sdept学习的学生:

select s1.sname from student s1 join student s2 on s1.sdept=s2.sdept and s2.sname='liu';

## 2.11 desc

1、查看数据库描述：

**DESCRIBE** DATABASE [EXTENDED] db\_name;

2、查看表/列描述：

**DESCRIBE** [EXTENDED|FORMATTED] **table\_name**;

案例：查看表book2的详细描述；

describe formatted book2;

3、查看分区描述：

**DESCRIBE** [EXTENDED|FORMATTED]**table\_name**[.column\_name] PARTITION

partition\_spec;

案例：查看表book2的分区category=cartoon

describe formatted book2 partition(category='cartoon');

## 2.12特性

Hive支持表级和分区级别的锁，事物，可以防止一个进程删除正在被另一个进程读取的表

支持索引（紧凑索引和位图索引）

From子句中只能有一个表或者视图，支持偏序的sort by

子查询只能出现在from子句中

# 3 内部表 外部表

内部表：

如果表是通过hive来创建，其中的数据也是通过hive来导入，则先有表后有表中的数据，则称这种表为内部表。

外部表：

如果是先有的数据文件，再创建hive表来管理，则这样的表称为外部表。

创建语法：

create external table ext\_student(id int ,name string) row format delimited fields terminated by '\t' location '/datax';

\*\*内部表被Drop的时候，表关联的hdfs中的文件夹和其中的文件都会被删除

\*\*外部表被Drop的时候，表关联的hdfs中的文件夹和其中的文件都不会被删除

内部表/外部表可以在表文件夹下直接上传符合格式的文件，从而增加表中的记录

# 4 分区表

## 4.1 分区

hive也支持分区表

对数据进行分区可以提高查询时的效率

普通表和分区表区别：有大量数据增加的需要建分区表

create table book (id bigint, name string) partitioned by (category string) row format delimited fields terminated by '\t';

create table book (id bigint, name string) partitioned by (category string,gender string) row format delimited fields terminated by '\t';

所谓的分区表是又在表文件夹下创建了子文件夹（[分区的名]=分区名字的值 ）分别存放同一张表中不同分区的数据。

从而将数据分区存放，提高按照分区查询数据时的效率。

所以当数据量比较大，而且数据经常要按照某个字段作为条件进行查询时，最好按照该条件进行分区存放。

通过观察元数据库，发现，分区表也会当作元数据存放在SDS表中。

\*\*如果手动的创建分区目录，是无法被表识别到的，因为在元数据库中并没有该分区的信息，如果想让手动创建的分区能够被表识别需要在元数据库SDS表中增加分区的元数据信息。

ALTER TABLE book add PARTITION (category = 'jp',gender='male') location '/user/hive/warehouse/testx.db/book/category=jp/gender=male';

把数据加载到分区表：

Load data local inpath ‘input/hive/file1’ into table logs partition (category=’jp’,gender=’male’)

查看分区： show Partitions tbalename;

2.2重命名分区

ALTER TABLE table\_name PARTITION partition\_spec RENAME TO PARTITION partition\_spec;

案例：

alter table book2 partition(category='zazhi') rename to partition(category='magazine');

在hdfs中可查看到book2目录下category=zazhi被修改为category=magazine;

//2.3恢复分区：直接在hdfs中创建相应的分区目录，但hive并没有该分区的元数据，此时可通过恢复分区加载到该分区；

// MSCK REPAIR TABLE table\_name;

// 案例：在hdfs book2中创建目录category=cartoon，并将该分区加入到表book2中；

// MSCK REPAIR TABLE book2;

// 报错，没有成功；

2.4删除分区:

ALTER TABLE table\_name DROP [IF EXISTS] PARTITION partition\_spec[, PARTITION partition\_spec, ...]

案例：删除book2表的categ ory=magazine分区；

alter table book2 drop partition(category='magazine');

3、alter either table or partition:

3.1修改表/分区的文件位置

ALTER TABLE table\_name [PARTITION partition\_spec] SET LOCATION "new location";

案例：

alter table book set location "/hqldata1";

## 4.2 动态分区

因为按照上面的方法向分区表中插入数据，如果源数据量很大，那么针对一个分区就要写一个insert，非常麻烦。况且在之前的版本中，必须先手动创建好所有的分区后才能插入，这就更麻烦了，你必须先要知道源数据中都有什么样的数据才能创建分区。  
使用动态分区可以很好的解决上述问题。动态分区可以根据查询得到的数据自动匹配到相应的分区中去。   
使用动态分区要先设置hive.exec.dynamic.partition参数值为true，默认值为false，即不允许使用.

Set hive.exec.dynamic.partition=true;

 insert overwrite table partition\_test partition(stat\_date='20110728',province)  
 select member\_id,name,province from partition\_test\_input where stat\_date='20110728';

stat\_date叫做静态分区列，province叫做动态分区列。select子句中需要把动态分区列按照分区的顺序写出来，静态分区列不用写出来。这样stat\_date='20110728'的所有数据，会根据province的不同分别插入到/user/hive/warehouse/partition\_test/stat\_date=20110728/下面的不同的子文件夹下，如果源数据对应的province子分区不存在，则会自动创建，非常方便，而且避免了人工控制插入数据与分区的映射关系存在的潜在风险。

注意，动态分区不允许主分区采用动态列而副分区采用静态列，这样将导致所有的主分区都要创建副分区静态列所定义的分区

动态分区可以允许所有的分区列都是动态分区列，但是要首先设置一个参数hive.exec.dynamic.partition.mode=nostrick;(慎用)

# 5 分桶表

hive也支持分桶表

分桶操作是更细粒度的分配方式

一张表可以同时分区和分桶

分桶的原理是根据指定的列的计算hash值模余分桶数量后将数据分开存放

分桶的主要作用是：对于一个庞大的数据集我们经常需要拿出来一小部分作为样例，然后在样例上验证我们的查询，优化我们的程序

第二个作用：处理查询时，链接两个在相同列上划分了桶的表，可以使map端链接高效实现。

创建带桶的 table ：

create table teacher(id int,name string) clustered by (id) into 4 buckets row format delimited fields terminated by '\t';

创建排序桶：

这样在map端链接是，对桶的链接成了高效的并归排序，进一步提高效率

Create table tercher(id int,name string) clustered by (id) sorted by (id asc) into 4 buckets row format delimited fields terminated by ‘\t’;

强制多个 reduce 进行输出：

set hive.enforce.bucketing=true;

往表中插入数据：

insert overwrite table teacher select \* from temp;//需要提前准备好temp，从temp查询数据写入到teacher

查看表的结构，会发现当前表下有四个文件：

dfs -ls /user/hive/warehouse/teacher;

读取数据，看一个文件的数据：

dfs -cat /user/hive/warehouse/teacher/000000\_0;

//桶使用 hash 来实现，所以每个文件拥有的数据的个数都有可能不相等。

对桶中的数据进行采样：

select \* from teacher tablesample(bucket 1 out of 4 on id);

//桶的个数从1开始计数，前面的查询从4个桶中的第一个桶获取数据。其实就是四分之一。

查询一半返回的桶数：

select \* from bucketed\_user tablesample(bucket 1 out of 2 on id);

//桶的个数从1开始计数，前2个桶中的第一个桶获取数据。其实就是二分之一。

# 6 HIVE的UDF

如果hive的内置函数不够用，我们也可以自己定义函数来使用，这样的函数称为hive的用户自定义函数，简称UDF。

新建java工程，导入hive相关包，导入hive相关的lib。

创建类继承UDF

自己编写一个evaluate方法，返回值和参数任意。

**public** **class** Strip **extends** UDF{

**public** String evaluate(String str){

**return** StringUtils.*strip*(str);

}

}

为了能让mapreduce处理，String要用Text处理。

将写好的类打成jar包，上传到linux中

在hive命令行下，向hive注册UDF：add jar /xxxx/xxxx.jar

为当前udf起一个名字：create temporary function fname as '类的全路径名';

之后就可以在hql中使用该自定义函数了： select strip(' bee e ');

Udf的方法名是大小写不敏感的。

# 7 HIVE的java api操作

hive也支持利用java api操作，hive实现了jdbc标准，所以可以使用标准的jdbc程序来处理对hive的操作。

1、hive首先要起动远程服务接口，命令：

hive –service hiveserver &

2、java工程中导入相应的需求jar包，列表如下:

antlr-runtime-3.0.1.jar

hive-exec-0.7.1.jar

hive-jdbc-0.7.1.jar

hive-metastore-0.7.1.jar

hive-service-0.7.1.jar

jdo2-api-2.3-ec.jar

libfb303.jar

或者：

直接导入hadoop。Common包

Hive启动java包

3、JDBC方式操作hive

略

1 配置hive

|  |
| --- |
| <property>    <name>hive.metastore.warehouse.dir</name>    <value>/usr/hive/warehouse</value>  //（hive中的数据库和表在HDFS中存放的文件夹的位置）  </property>  <property>    <name>hive.server2.thrift.port</name>    <value>10000</value>                   //（HiveServer2远程连接的端口，默认为10000）  </property>    <property>    <name>hive.server2.thrift.bind.host</name>    <value>\*\*.\*\*.\*\*.\*\*</value>                          //（hive所在集群的IP地址）  </property>  <property>    <name>hive.server2.long.polling.timeout</name>    <value>5000</value>                                // (默认为5000L,此处修改为5000，不然程序会报错)  </property>  <property>    <name>javax.jdo.option.ConnectionURL</name>    <value>jdbc:mysql://localhost:3306/hive?createDatabaseIfNotExist=true</value>  //（Hive的元数据库，我采用的是本地Mysql作为元数据库）  </property>    <property>    <name>javax.jdo.option.ConnectionDriverName</name>          //（连接元数据的驱动名）    <value>com.mysql.jdbc.Driver</value>  </property>  <property>    <name>javax.jdo.option.ConnectionUserName</name>             //（连接元数据库用户名）    <value>root</value>  </property>    <property>    <name>javax.jdo.option.ConnectionPassword</name>             // （连接元数据库密码）    <value>root</value>  </property> |

2 、（可不执行）先启动元数据库，在命令行中键入：hive --service metastore &

3 、在使用 JDBC 开发 Hive 程序时,  必须首先开启 Hive 的远程服务接口。使用下面命令进行开启:

hive --service hiveserver 2>/dev/**null** 2>/dev/**null**  - p 10000 &

查看hiveserver2是否已经开启： netstat -nl |grep 10000

|  |
| --- |
| **public class** HiveJdbcCli {  **private static** String *driverName*=**"org.apache.hive.jdbc.HiveDriver"**;  *//HiveServer远程连接的端口，默认为10000* **private static** String *url*=**"jdbc:hive2://192.168.8.107:10000/default"**;  **private static** String *username*=**"root"**;  **private static** String *password*=**"root"**;  **private static** String *sql*;  **private static** ResultSet *resultSet*;  **private static final** Logger ***log***= Logger.*getLogger*(HiveJdbcCli.**class**);   **public static void** main(String[] args) {  Connection conn=**null**;  Statement stat=**null**;  **try** {  conn=*getConn*();  stat=conn.createStatement();   *//1、存在就先删除* String tableName=*dropTable*(stat);  *//2 、不存在就创建  createTable*(stat,tableName);  *//3、查看创建的表  showTables*(stat,tableName);  *//4、describe table表  descTables*(stat,tableName);  *//5、load data into table  loadData*(stat,tableName);  *//6、select \* query  selectData*(stat,tableName);  *//7、regular hive query  countData*(stat,tableName);  } **catch** (ClassNotFoundException e) {  e.printStackTrace();  ***log***.error(*driverName*+**"is not found!"**);  System.*exit*(1);  } **catch** (SQLException e) {  e.printStackTrace();  ***log***.error(**"Connection error"**,e);  System.*exit*(1);  }**finally** {   **try** {  **if**(*resultSet*!=**null**){  *resultSet*.close();  *resultSet*=**null**;  }  **if**(stat!=**null**){  stat.close();  stat=**null**;  }  **if**(conn!=**null**){  conn.close();  conn=**null**;  }  } **catch** (SQLException e) {  e.printStackTrace();  }  }  }   **private static void** countData(Statement stat, String tableName) **throws** SQLException {  *sql* = **"select** *count***(1) from "**+tableName;  System.***out***.println(**"running sql:"**+*sql*);  *resultSet*=stat.executeQuery(*sql*);  **while** (*resultSet*.next()){  System.***out***.println(**"count---->"**+*resultSet*.getString(1));  }  }   **private static void** selectData(Statement stat, String tableName) **throws** SQLException {  *sql*=**"select** *\** **from "**+ tableName;  System.***out***.println(**"running sql:"**+*sql*);  *resultSet*=stat.executeQuery(*sql*);  **while**(*resultSet*.next()){  System.***out***.println(*resultSet*.getString(1)+**"\t"**+*resultSet*.getString(2));  }  }   **private static void** loadData(Statement stat, String tableName) **throws** SQLException {  String filePath=**"/opt/mydata/english.txt"**;  *sql*=**"load data local inpath '"**+filePath+**"' into table "**+tableName;  System.***out***.println(**"running sql:"**+*sql*);  stat.execute(*sql*);  }   **private static void** descTables(Statement stat, String tableName) **throws** SQLException {  *sql* = **"describe "** + tableName;  System.***out***.println(**"Running:"** + *sql*);  System.***out***.println(**"running sql:"**+*sql*);  *resultSet* = stat.executeQuery(*sql*);  System.***out***.println(**"执行 describe table 运行结果:"**);  **while** (*resultSet*.next()) {  System.***out***.println(*resultSet*.getString(1) + **"\t"** + *resultSet*.getString(2));  }  }   **private static void** showTables(Statement stat, String tableName) **throws** SQLException {  *sql* = **"show tables '"**+tableName+**"'"**;  System.***out***.println(**"running sql:"**+*sql*);  *resultSet*=stat.executeQuery(*sql*);  **while** (*resultSet*.next()){  System.***out***.println(*resultSet*.getString(1));  }  }   *//根据表名创建表* **private static void** createTable(Statement stat, String tableName) **throws** SQLException {  *sql*=**"create table "**+tableName+**" (key int,value string) row format delimited fields terminated by '\t'"**;  System.***out***.println(**"running sql:"**+*sql*);  stat.execute(*sql*);  }   *//* **private static** String dropTable(Statement stat) **throws** SQLException {  *//创建表名* String tableName=**"testHIve"**;  *sql*=**"drop table "**+tableName;  System.***out***.println(**"running sql:"**+*sql*);  stat.execute(*sql*);  **return** tableName;  }  *//获取链接* **private static** Connection getConn() **throws** ClassNotFoundException,  SQLException {  Class.*forName*(*driverName*);  Connection conn = DriverManager.*getConnection*(*url*, *username*, *password*);  **return** conn;  } } |