

Hadoop 集群（第 14 期）

——Hive 应用开发

1、Hive 的服务

Hive 不仅仅是一个 **shell**，通过配置，它可以提供诸如 **Thrift 服务器**、**Web 接口**、**元数据**和 **JDBC/ODBC 服务**，具有**强大的功能**和**良好的可扩展性**。

1.1 Hive Shell

Hive Shell 是**默认**的服务，提供**命令行接口**，可以在此命令行上**直接编写** HiveQL 语句执行，**每条语句**以**分号**结束，也可以在 Hive Shell 上**执行 Hive 自带的管理命令**，例如导入 jar 包，加入**临时环境变量**等操作。

1) 执行 Hive 语句

- 执行查询语句

```
hive>select name from xp;
```

```
hive> select name from xp;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201203240454_0001, Tracking URL = http://Master.Hadoop:50030/jobdetails.jsp?jobid=job_201203240454_0001
Kill Command = /usr/hadoop/libexec/./bin/hadoop job -Dmapred.job.tracker=http://192.168.1.2:9001 -kill job_201203240454_0001
Hadoop job information for Stage-1: number of mappers: 0; number of reducers: 0
2012-03-25 17:37:45,996 Stage-1 map = 0%, reduce = 0%
2012-03-25 17:37:52,010 Stage-1 map = 100%, reduce = 100%
Ended Job = job_201203240454_0001
MapReduce Jobs Launched:
Job 0: HDFS Read: 0 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Time taken: 22.176 seconds
hive>
```

从上面的得知，在执行“select * from xp;”时，可以看到启动了一个 MapReduce job，建好表导入数据后，就可以浏览与 Hive 相关的目录。

- 执行 HDFS 文件操作

在 Hive 的 shell 上使用 dfs 命令可以查看 HDFS 上的文件。

```
hive> dfs -ls /user/hive/warehouse/;
Found 1 items
drwxr-xr-x - hadoop supergroup          0 2012-03-25 03:53 /user/hive/warehouse/xp
hive>
```

HDFS 上的“**/user/hive/warehouse**”目录是 **Hive** 的**数据仓库目录**，每个表对应一个以表明命名的目录，目录下存放导入的文件、分区目录、桶目录等数据文件。**Hive** 的查询日志默认保存在本地文件系统的“**/tmp/<user.name>**”目录下，**Hive** 的 MapReduce 执行计划保

存在本地的“/tmp/<user.name>/hive”中。这三个目录可以分别通过属性：

- hive.metastore.metadb.dir: (HDFS 上的) 数据目录
- hive.querylog.location: 查询日志存放目录
- hive.exec.scratchdir: (HDFS 上的) 临时文件目录

2) 设置和查看临时变量

备注: 设置只在当前会话有效, 方便切换 Hive 的执行环境。

```
hive>set fs.default.name=hdfs://192.168.1.2:9000
hive>set fs.default.name;
fs.default.name
```

```
hive> set fs.default.name=hdfs://192.168.1.2:9000;
hive> set fs.default.name;
fs.default.name=hdfs://192.168.1.2:9000
hive>
```

3) 导入 jar 包 (jar 包存在)

```
hive>add jar hivejar.jar
Added hivejar.jar to class path
Added resource:hivejar.jar
```

4) 创建函数 (类存在)

```
hive>create temporary function udfTest as 'com.cstore.udfExample';
```

当然也可以在本地系统的命令行运行 Hive 的 shell。

```
$ hive -e 'select * from xp' (执行 HiveQL 语句)
$ hive -config /hive-0.8.1/conf (重新载入新的配置文件)
$ hive -service hiveserver (启动服务)
```

```
[hadoop@Master ~]$ hive -e 'select * from xp'
Logging initialized using configuration in file:/usr/hive/conf/hive-log4j.properties
Hive history file=/tmp/hadoop/hive_job_log_hadoop_201203250557_1916499547.txt
OK
Time taken: 2.542 seconds
[hadoop@Master ~]$
```

1.2 JDBC/ODBC支持

Hive 提供了对 **JDBC** 和 **ODBC** 的支持, 这一特性使基于 JDBC/ODBC 的应用能力以很小的代价平滑过渡到 Hive 平台。

1) JDBC

在 Hive 的 jar 包中，“**org.apache.hadoop.hive.jdbc.HiveDriver**”负责提供 **JDBC** 接口，客户端程序有了这个包，就可以把 Hive 当成一个数据库来使用，**大部分的操作与对传统数据库的操作相同**，Hive 允许支持 JDBC 协议的应用程序连接到 Hive。当 Hive 在指定端口启动 hiveserver 服务后，客户端通过 Java 的 Thrift 和 Hive 服务器进行通信，连接过程如下：

- 开启 hiveserver 服务

```
$ hive -service hiveserver 50000 (50000)
```

- 建立与 Hive 的连接

```
Class.forName("org.apache.hadoop.hive.jdbc.HiveDriver");  
Connection con= DriverManager.getConnection("jdbc:hive://ip:50000/default,"hive","hadoop")
```

默认只能连接到 default，通过上面的两行代码建立连接后，其他的操作与传统数据库无太大差别。

Hive 的 JDBC 驱动目前还不太成熟，并不支持所有的 JDBC API。

1.3 Thrift服务

Hive 的 Thrift 是一种跨语言服务的可伸缩软件框架。它结合了功能强大的软件堆栈的代码生成引擎，可以无缝地与 C++、C#、Java、Python、PHP 和 Ruby 结合。Thrift 允许用户简单地定义文件中的数据类型和服务接口，编译器生成代码来实现 RPC 客户端和服务器的通信。

Hive 内部集成了 Thrift 服务，支持在多种编程语言中运行 Hive 命令，使客户端可以跨平台连接 Hive。

1.4 Web接口

Hive Web Interface，简称 **hwi**，是 Hive 提供的 Web 接口。通过这个 Web 接口，可以方便地执行与 Hive shell 的命令**具有功能相同**的操作。

下面看看网络接口具有的特征：

- 分离查询的执行

在 CLI 下，如果我们要执行多个查询就要打开多个终端，而通过网络接口，可以同时执行多个查询，网络接口可以在网络服务器上管理会话（session）。

- 不用本地安装 Hive

一个用户不需要本地安装 Hive 就可以通过网络浏览器访问 Hive 进行操作。一个用户如果想通过 Web 跟 Hadoop 和 Hive 交互，那么需要访问多个端口。而一个远程或 VPN 用户则只需要访问 Hive 网络接口所使用的“0.0.0.0 tcp/9999”。

使用 Hive 的网络接口需要修改配置文件。配置文件所在的位置为“usr/hive/conf/”目录下面，可以参考下面代码进行，添加到用户定义的配置文件“hive-site.xml”中。

```

<property>
  <name>hive.hwi.war.file</name>
  <value>lib/hive-hwi-0.8.1.war</value>
  <description>This sets the path to the HWI war file, relative to ${HIVE_HOME}.
</description>
</property>
<property>
  <name>hive.hwi.listen.host</name>
  <value>0.0.0.0</value>
  <description>This is the host address the Hive Web Interface will listen on</description>
</property>
<property>
  <name>hive.hwi.listen.port</name>
  <value>9999</value>
  <description>This is the port the Hive Web Interface will listen on</description>
</property>

```

```

<property>
  <name>javax.jdo.option.ConnectionPassword</name>
  <value>hadoop</value>
</property>
<property>
  <name>hive.hwi.war.file</name>
  <value>lib/hive-hwi-0.8.1.war</value>
</property>
<property>
  <name>hive.hwi.listen.host</name>
  <value>0.0.0.0</value>
</property>
<property>
  <name>hive.hwi.listen.port</name>
  <value>9999</value>
</property>
</configuration>
hive-site.xml 37L, 950C 已写入
[hadoop@Master conf]$

```

开启 Hive 的 Web 服务:

```
hive --service hwi
```

```

[hadoop@Master ~]$ hive --service hwi
12/03/25 18:43:02 INFO hwi.HWIServer: HWI is starting up
12/03/25 18:43:02 WARN conf.HiveConf: DEPRECATED: Ignoring hive-default.xml found on the CLASSPATH at /usr/hive/conf/hive-default.xml
12/03/25 18:43:02 INFO mortbay.log: Logging to org.slf4j.impl.Log4jLoggerAdapter(org.mortbay.log) via org.mortbay.log.Slf4jLog
12/03/25 18:43:02 INFO mortbay.log: jetty-6.1.26
12/03/25 18:43:02 INFO mortbay.log: Extract /usr/hive/lib/hive-hwi-0.8.1.war to /tmp/Jetty_0_0_0_9999_hive.hwi.0.8.1.war__m9wzki/webapp
12/03/25 18:43:03 INFO mortbay.log: Started SocketConnector@0.0.0.0:9999

```

完成上述工作之后，在浏览器键入地址http://host_name:9999/hwi，即可进入Hive的Web页面，如图 1.4-1 所示。



图 1.4-1 Hive 的 Web 界面

点击“Create Session”创建会话，在如图 1.4-2 所示的界面中可以执行查询操作。

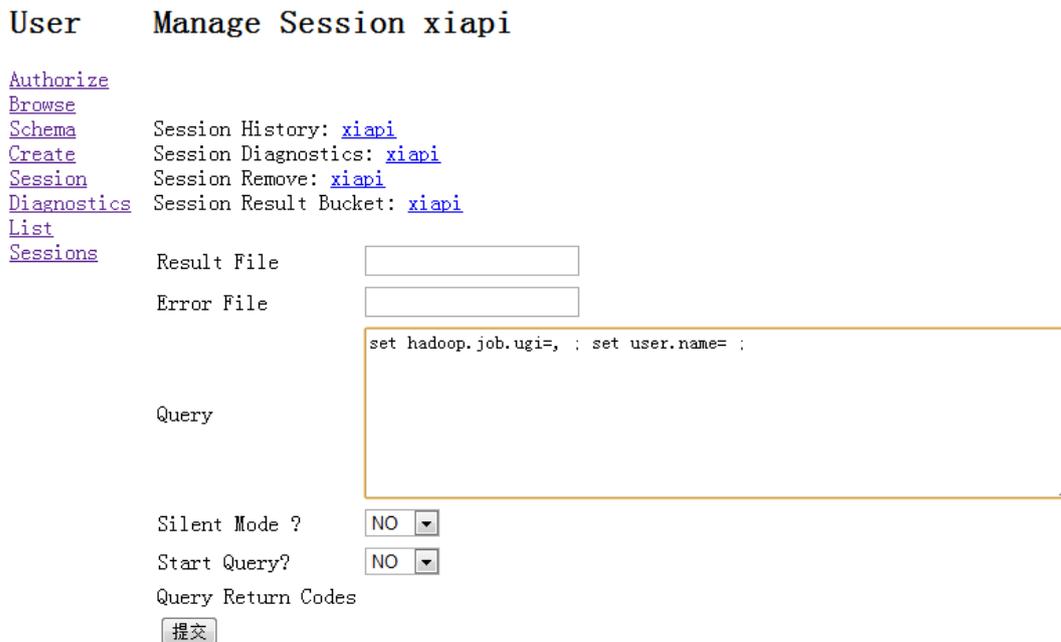


图 1.4-2 查询页面

可以看到 Hive 的网络接口拉近了用户和系统的距离，我们可以通过网络接口之间创建会话并进行查询。用户界面和功能展示非常直观，适合于刚接触到 Hive 的用户。

2、HiveQL详解

HiveQL 是一种类似 SQL 的语言，它与大部分的 SQL 语法兼容，但是并不完全支持 SQL 标准，如 **HiveQL 不支持更新操作**，也**不支持索引和事务**，它的**子查询**和**join 操作**也很**局限**，这是因其底层依赖于 Hadoop 云平台这一特性决定的，但其有些特点是 SQL 所无法企及的。例如多表查询、支持 create table as select 和集成 MapReduce 脚本等，本节主要介绍 Hive 的数据类型和常用的 HiveQL 操作。

2.1 HiveQL的数据类型

Hive 支持**基本数据类型**和**复杂类型**，**基本数据类型**主要有**数值类型**、**布尔型**和**字符串**，**复杂类型**有三种：**ARRAY**、**MAP**和**STRUCT**，如表 2.1-1 所示。

表 2.1-1 数据类型

基本类型	大小	描述
TINYINT	1 字节	有符号整数
SMALLINT	2 字节	有符号整数
INT	4 字节	有符号整数
BIGINT	8 字节	有符号整数
FLOAT	4 字节	单精度浮点数
DOUBLE	8 字节	双精度浮点数
BOOLEAN	~	去 true/false
STRING	最大 2GB	字符串，类似于 sql 中的 varchar 类型
复杂类型	大小	描述
ARRAY	不限	一组有序字段，字段类型 必须相同
MAP	不限	无序键值对，键值内部字段类型 必须相同
STRUCT	不限	一组字段，字段类型 可以不同

基本类型可以隐式向上转换，特别需要主要是的 STRING 类型可以转换为 BOUBLE。

2.2 HIveQL的常用操作

常用的 HiveQL 操作有如下几种。

1) 创建表

首先建立三张供测试用的表：**userinfo** 表中有**两列**，以 **tab 键分割**，分别存储**用户**的 **id** 和名字 **name**；**choice** 表中有**两列**，以 **tab 键分割**，分别存储用户的 **userid** 和选课名称 **classname**；**classinfo** 表中有**两列**，以 **tab 键分割**，分别存储课程老师 **teacher** 和课程名 **classname**。

```
create table userinfo(id int,name string)
row format delimited fields terminated by '\t';
```

```
hive> create table userinfo(id int,name string)
> row format delimited fields terminated by '\t';
OK
Time taken: 2.835 seconds
hive>
```

```
create table choice(userid int,classname string)
row format delimited fields terminated by '\t';
```

```
hive> create table choice(userid int,classname string)
> row format delimited fields terminated by '\t';
OK
Time taken: 0.155 seconds
hive>
```

```
create table classinfo(teacher string,classname string)
row format delimited fields terminated by '\t';
```

```
hive> create table classinfo(teacher string,classname string)
> row format delimited fields terminated by '\t';
OK
Time taken: 0.15 seconds
hive>
```

显示刚才创建的数据表：

```
hive> show tables;
OK
choice
classinfo
userinfo
xp
Time taken: 2.2 seconds
```

“**row format delimited fields terminated by**”是 HiveQL 特有的，用来指定**数据的分割方式**，如果不人为指定，则默认的格式如下。

```
row format delimited fields terminated by '\001' collection items terminated by '\002' map keys
terminated by '\003' lines terminated by '\n' stored as textfile.
```

上述“**collection items terminated by '\002'**”用来**指定集合类型中数据的分割方式**，针对 **ARRAY**、**STRUCT** 和 **MAP** 的 **key/value** 之间的分割；“**map keys terminated by '\003'**”针对 **MAP** 的 **key** 内的分割方式；“**lines terminated by '\n'**”制定了**行之间以回车分割**；“**stored as textfile**”指定以**文本存储**。**分割方式和存储方式**可以显示指定。

Hive 中的**表**可以分为**托管表**和**外部表**，**托管表**的**数据移动到数据仓库目录**下，由 **Hive** 管理，**外部表**的**数据在指定位置**，不在 **Hive** 的**数据仓库**中，只是在 **Hive 元数据库**中**注册**。上面创建的表都是**托管表**，创建**外部表**采用“**create external tablename**”方式创建，并在**创建表的同时指定表的位置**。

2) 导入数据

建表后，可以从本地文件系统或 HDFS 中导入数据文件，导入数据样例如下：

- **userinfo.txt**

```
1 xiapi
2 xiaoxue
3 qingqing
```

- **choice.txt**

```
1 math
1 china
1 english
2 china
2 english
3 english
```

- **classinfo.txt**

```
jack math
sam china
lucy english
```

首先在 Master.Hadoop 的 “/home/hadoop” 下面按照上面建立三个文件，并添加如上的内容信息。

```
[hadoop@Master ~]$ ll | grep txt
-rw-r--r--. 1 hadoop hadoop 53 3月 25 23:08 choice.txt
-rw-r--r--. 1 hadoop hadoop 33 3月 25 23:08 classinfo.txt
-rw-r--r--. 1 hadoop hadoop 29 3月 25 23:08 userinfo.txt
[hadoop@Master ~]$
```

按照下面导入数据。

```
load data local inpath '/home/hadoop/userinfo.txt' overwrite into table userinfo;
load data local inpath '/home/hadoop/choice.txt' overwrite into table choice;
load data local inpath '/home/hadoop/classinfo' overwrite into table classinfo;
```

如果导入的数据在 HDFS 上，则不需要 **local** 关键字。**托管表** 导入的数据文件可在数据仓库目录 “**user/hive/warehouse/<tablename>**” 中看到。

Hive 的数据导入只是复制或移动文件，并不对数据的模式进行检查，对数据模式的检查要等到查询时才进行，这就是 Hive 采用的 “schema on read” 加载方式。这种方式可以大大提高加载数据的效率。

执行上面语句结果如下：

```
hive> load data local inpath '/home/hadoop/userinfo.txt' overwrite into table userinfo;
Copying data from file:/home/hadoop/userinfo.txt
Copying file: file:/home/hadoop/userinfo.txt
Loading data to table default.userinfo
Deleted hdfs://192.168.1.2:9000/user/hive/warehouse/userinfo
OK
Time taken: 2.671 seconds
hive> load data local inpath '/home/hadoop/choice.txt' overwrite into table choice;
Copying data from file:/home/hadoop/choice.txt
Copying file: file:/home/hadoop/choice.txt
Loading data to table default.choice
Deleted hdfs://192.168.1.2:9000/user/hive/warehouse/choice
OK
Time taken: 0.307 seconds
hive> load data local inpath '/home/hadoop/classinfo.txt' overwrite into table classinfo;
Copying data from file:/home/hadoop/classinfo.txt
Copying file: file:/home/hadoop/classinfo.txt
Loading data to table default.classinfo
Deleted hdfs://192.168.1.2:9000/user/hive/warehouse/classinfo
OK
Time taken: 0.361 seconds
hive>
```

3) 分区

分区是表的**部分列的集合**，可以为**频繁使用的数据建立分区**，这样查找分区中的数据时就不需要扫描全表，这对于**提高查找效率**很有帮助，建立分区的语句为：

```
//table 中的列不能和 partition 中的列重合了
create table ptest(userid int) partitioned by (name string)
row format delimited fields terminated by '\t';
load data local inpath '/home/hadoop/xiapi.txt' overwrite into table ptest partition (name='xiapi');
```

备注：文本文件“xiapi.txt”的内容只有用户名为“xiapi”的id。

```
[hadoop@Master ~]$ pwd
/home/hadoop
[hadoop@Master ~]$ ll | grep txt
-rw-r--r--. 1 hadoop hadoop    53  3月 25 23:08 choice.txt
-rw-r--r--. 1 hadoop hadoop    33  3月 25 23:08 classinfo.txt
-rw-r--r--. 1 hadoop hadoop    29  3月 25 23:08 userinfo.txt
-rw-r--r--. 1 hadoop hadoop     2  3月 26 00:54 xiapi.txt
[hadoop@Master ~]$ more xiapi.txt
1
[hadoop@Master ~]$
```

首先创建分区：

```
hive> create table ptest(userid int) partitioned by (name string)
> row format delimited fields terminated by '\t';
OK
Time taken: 2.916 seconds
```

然后倒入分区数据：

```
hive> load data local inpath '/home/hadoop/xiapi.txt' overwrite into table ptest partition(name='xiapi');
Copying data from file:/home/hadoop/xiapi.txt
Copying file: file:/home/hadoop/xiapi.txt
Loading data to table default.ptest partition (name=xiapi)
OK
Time taken: 0.642 seconds
hive>
```

建立分区后，会在相应的表目录下建立以分区名命名的目录，目录下是分区的数据，如下所示：

```
hive>dfs -ls /user/hive/warehouse/ptest/name=xiapi/;
```

```
hive> dfs -ls /user/hive/warehouse/ptest/name=xiapi/;
Found 1 items
-rw-r--r-- 1 hadoop supergroup 2 2012-03-26 00:58 /user/hive/warehouse/ptest/name=xiapi/xiapi.txt
hive>
```

对分区进行查询：

```
hive>select userid from ptest where name='xiapi';
```

```
hive> select userid from ptest where name='xiapi';
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201203240454_0003, Tracking URL = http://Master.Hadoop:50030/jobdetails.jsp?jobid=job_201203240454_0003
Kill Command = /usr/hadoop/libexec/./bin/hadoop job -Dmapred.job.tracker=http://192.168.1.2:9001 -kill job_201203240454_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2012-03-26 01:03:06,664 Stage-1 map = 0%, reduce = 0%
2012-03-26 01:03:12,689 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.53 sec
2012-03-26 01:03:13,695 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.53 sec
2012-03-26 01:03:14,700 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.53 sec
2012-03-26 01:03:15,705 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.53 sec
2012-03-26 01:03:16,709 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.53 sec
2012-03-26 01:03:17,714 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.53 sec
2012-03-26 01:03:18,718 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 0.53 sec
MapReduce Total cumulative CPU time: 530 msec
Ended Job = job_201203240454_0003
MapReduce Jobs Launched:
Job 0: Map: 1 Accumulative CPU: 0.53 sec HDFS Read: 226 HDFS Write: 2 SUCCESS
Total MapReduce CPU Time Spent: 530 msec
OK
1
Time taken: 21.371 seconds
hive>
```

显示分区：

```
hive>show partitions ptest;
```

```
hive> show partitions ptest;
OK
name=xiapi
Time taken: 0.081 seconds
hive>
```

对分区插入数据：

```
insert overwrite table ptest partition(name='xiapi') select id from userinfo where name='xiapi';
```

```

hive> insert overwrite table ptest partition(name='xiapi')
> select id from userinfo where name='xiapi';
Total MapReduce jobs = 2
Launching Job 1 out of 2
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201203240454_0004, Tracking URL = http://Master.Hadoop:50030/jobdetails.jsp?jobid=job_201203240454_0004
Kill Command = /usr/hadoop/libexec/./bin/hadoop job -Dmapred.job.tracker=http://192.168.1.2:9001 -kill job_201203240454_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2012-03-26 01:09:30,088 Stage-1 map = 0%, reduce = 0%
2012-03-26 01:09:36,107 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.61 sec
2012-03-26 01:09:37,111 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.61 sec
2012-03-26 01:09:38,117 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.61 sec
2012-03-26 01:09:39,121 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.61 sec
2012-03-26 01:09:40,125 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.61 sec
2012-03-26 01:09:41,129 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.61 sec
2012-03-26 01:09:42,135 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 0.61 sec
MapReduce Total cumulative CPU time: 610 msec
Ended Job = job_201203240454_0004
Ended Job = 576313603, job is filtered out (removed at runtime).
Moving data to: hdfs://192.168.1.2:9000/tmp/hive-hadoop/hive_2012-03-26_01-09-12_442_2752936120819056122/-ext=10000
Loading data to table default.ptest partition (name=xiapi)
Deleted hdfs://192.168.1.2:9000/user/hive/warehouse/ptest/name=xiapi
Partition default.ptest[name=xiapi] stats: [num_files: 1, num_rows: 0, total_size: 2, raw_data_size: 0]
Table default.ptest stats: [num_partitions: 1, num_files: 1, num_rows: 0, total_size: 2, raw_data_size: 0]
1 Rows loaded to ptest
MapReduce Jobs Launched:
Job 0: Map: 1 Accumulative CPU: 0.61 sec HDFS Read: 248 HDFS Write: 2 SUCCESS
Total MapReduce CPU Time Spent: 610 msec
OK
Time taken: 30.791 seconds
hive>

```

删除分区:

```
hive> alter table ptest drop partition (name='xiapi')
```

备注: 通常情况下需要先预先创建好分区, 然后才能使用该分区。还有分区列的值要转化为文件夹的存储路径, 所以如果分区列的值中包含特殊值, 如 '%', ':', '/', '#', 它将会被使用 % 加上 2 字节的 ASCII 码进行转义。

4) 桶

可以把表或分区组织成桶, 桶是按行分开组织特定字段, 每个桶对应一个 reduce 操作。在建立桶之前, 需要设置 “hive.enforce.bucketing” 属性为 true, 使 Hive 能够识别桶。在表中分桶的操作如下:

```

hive> set hive.enforce.bucketing=true;
hive> set hive.enforce.bucketing;
hive.enforce.bucketing=true;
hive> create table btest2(id int,name string) clustered by(id) into 3 buckets
row format delimited fields terminated by '\t';

```

```

hive> set hive.enforce.bucketing;
hive.enforce.bucketing=false
hive> set hive.enforce.bucketing=true;
hive> set hive.enforce.bucketing;
hive.enforce.bucketing=true
hive> create table btest2(id int,name string) clustered by(id) into 3 buckets
> row format delimited fields terminated by '\t';
OK
Time taken: 0.178 seconds
hive>

```

向桶中插入数据, 这里按照用户 id 分了三个桶, 在插入数据时对应三个 reduce 操作,

输出三个文件。

```
hive>insert overwrite table btest2 select * from userinfo;
```

```
hive> insert overwrite table btest2 select * from userinfo;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 3
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201203240454_0010, Tracking URL = http://Master.Hadoop:50030/jobdetails.jsp?jobid=job_201203240454_0010
Kill Command = /usr/hadoop/libexec/./bin/hadoop job -Dmapred.job.tracker=http://192.168.1.2:9001 -kill job_201203240454_0010
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 3
2012-03-26 02:59:54, 712 Stage-1 map = 0%, reduce = 0%
2012-03-26 03:00:00, 727 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.89 sec
2012-03-26 03:00:01, 731 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.89 sec
2012-03-26 03:00:02, 734 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.89 sec
2012-03-26 03:00:03, 738 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.89 sec
2012-03-26 03:00:04, 741 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.89 sec
2012-03-26 03:00:05, 744 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.89 sec
2012-03-26 03:00:06, 748 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.89 sec
```

查看数据仓库下的桶目录，三个桶对应三个目录。

```
hive>dfs -ls /user/hive/warehouse/btest2;
```

```
hive> dfs -ls /user/hive/warehouse/btest2;
Found 3 items
-rw-r--r--  1 hadoop supergroup    11 2012-03-26 03:00 /user/hive/warehouse/btest2/000000_0
-rw-r--r--  1 hadoop supergroup     8 2012-03-26 03:00 /user/hive/warehouse/btest2/000001_0
-rw-r--r--  1 hadoop supergroup    10 2012-03-26 03:00 /user/hive/warehouse/btest2/000002_0
hive>
```

Hive 使用对分桶所用的值进行 hash，并用 hash 结果除以桶的个数做取余运算的方式来分桶，保证了每个桶中都有数据，但每个桶中的数据条数不一定相等，如下所示。

```
hive>dfs -cat /user/hive/warehouse/btest2/*0_0;
hive>dfs -cat /user/hive/warehouse/btest2/*1_0;
hive>dfs -cat /user/hive/warehouse/btest2/*2_0;
```

```
hive> dfs -cat /user/hive/warehouse/btest2/*0_0;
3 qingqing
hive> dfs -cat /user/hive/warehouse/btest2/*1_0;
1 xiapi
hive> dfs -cat /user/hive/warehouse/btest2/*2_0;
2 xiaoxue
hive>
```

分桶可以获得比分区更高的查询效率，同时分桶也便于对全部数据进行采样处理。下面是分桶取样的操作。

```
hive>select * from btest2 tablesample(bucket 1 out of 3 on id);
```

```
hive> select * from btest2 tablesample(bucket 1 out of 3 on id);
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201203240454_0011, Tracking URL = http://Master.Hadoop:50030/jobdetails.jsp?jobid=job_201203240454_0011
Kill Command = /usr/hadoop/libexec/./bin/hadoop job -Dmapred.job.tracker=http://192.168.1.2:9001 -kill job_201203240454_0011
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2012-03-26 03:20:13,110 Stage-1 map = 0%, reduce = 0%
2012-03-26 03:20:19,123 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.8 sec
2012-03-26 03:20:20,127 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.8 sec
2012-03-26 03:20:21,130 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.8 sec
2012-03-26 03:20:22,134 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.8 sec
2012-03-26 03:20:23,137 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.8 sec
2012-03-26 03:20:24,140 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.8 sec
2012-03-26 03:20:25,144 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 0.8 sec
MapReduce Total cumulative CPU time: 800 msec
Ended Job = job_201203240454_0011
MapReduce Jobs Launched:
Job 0: Map: 1 Accumulative CPU: 0.8 sec HDFS Read: 224 HDFS Write: 11 SUCCESS
Total MapReduce CPU Time Spent: 800 msec
OK
3 qingqing
Time taken: 20.939 seconds
hive>
```

5) 多表插入

多表插入指的是在同一条语句中，把读取的同一份元数据插入到不同的表中。只需要扫描一遍元数据即可完成所有表的插入操作，效率很高。多表操作示例如下。

```
hive>create table mutill as select id,name from userinfo;
```

```
hive> create table mutill as select id,name from userinfo;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201203240454_0012, Tracking URL = http://Master.Hadoop:50030/jobdetails.jsp?jobid=job_201203240454_0012
Kill Command = /usr/hadoop/libexec/./bin/hadoop job -Dmapred.job.tracker=http://192.168.1.2:9001 -kill job_201203240454_0012
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2012-03-26 03:28:25,511 Stage-1 map = 0%, reduce = 0%
2012-03-26 03:28:31,525 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.57 sec
2012-03-26 03:28:32,528 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.57 sec
2012-03-26 03:28:33,532 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.57 sec
2012-03-26 03:28:34,535 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.57 sec
2012-03-26 03:28:35,539 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.57 sec
2012-03-26 03:28:36,542 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.57 sec
2012-03-26 03:28:37,546 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 0.57 sec
MapReduce Total cumulative CPU time: 570 msec
Ended Job = job_201203240454_0012
Moving data to: hdfs://192.168.1.2:9000/user/hive/warehouse/mutill
Table default.mutill stats: [num_partitions: 0, num_files: 1, num_rows: 0, total_size: 29, raw_data_size: 0]
3 Rows loaded to hdfs://192.168.1.2:9000/tmp/hive-hadoop/hive_2012-03-26_03-28-16_450_1227424995179687891/-ext=10000
MapReduce Jobs Launched:
Job 0: Map: 1 Accumulative CPU: 0.57 sec HDFS Read: 248 HDFS Write: 29 SUCCESS
Total MapReduce CPU Time Spent: 570 msec
OK
Time taken: 21.722 seconds
hive>
```

```
hive>create table mutill2 like mutill;
```

```
hive> create table mutill2 like mutill;
OK
Time taken: 0.141 seconds
hive>
```

```
hive>from userinfo insert overwrite table mutill
select id,name insert overwrite table mutill2
select count(distinct id),name group by name;
```

```
hive> from userinfo insert overwrite table mutill
> select id,name insert overwrite table mutill2
> select count(distinct id),name group by name;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201203240454_0013, Tracking URL = http://Master.Hadoop:50030/jobdetails.jsp?jobid=job_201203240454_0013
Kill Command = /usr/hadoop/libexec/./bin/hadoop job -Dmapred.job.tracker=http://192.168.1.2:9001 -kill job_201203240454_0013
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2012-03-26 03:37:07,613 Stage-2 map = 0%, reduce = 0%
2012-03-26 03:37:13,628 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 1.1 sec
2012-03-26 03:37:14,632 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 1.1 sec
2012-03-26 03:37:15,636 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 1.1 sec
2012-03-26 03:37:16,640 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 1.1 sec
2012-03-26 03:37:17,643 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 1.1 sec
2012-03-26 03:37:18,647 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 1.1 sec
2012-03-26 03:37:19,651 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 1.1 sec
2012-03-26 03:37:20,655 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 1.1 sec
2012-03-26 03:37:21,659 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 1.1 sec
2012-03-26 03:37:22,663 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 1.1 sec
2012-03-26 03:37:23,667 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 1.1 sec
2012-03-26 03:37:24,670 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 1.1 sec
2012-03-26 03:37:25,674 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 3.12 sec
2012-03-26 03:37:26,679 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 3.12 sec
```

6) 修改表

重命名表名、增加数据列的操作如下：

- 重命名表名

```
hive>alter table mutill rename to mutill1;
```

```
hive> alter table mutill rename to mutill1;
OK
Time taken: 0.227 seconds
```

- 增加数据列

```
hive>alter table mutill1 add columns(grade string);
```

```
hive> alter table mutill1 add columns(grade string);
OK
Time taken: 0.201 seconds
```

- 显示数据表结构

```
hive>describe mutill1;
```

```
hive> describe mutill1;
OK
id      int
name    string
grade   string
Time taken: 0.058 seconds
```

7) 删除表

```
hive>drop table mutill1;
```

```
hive> drop table mutill1;
OK
Time taken: 0.578 seconds
hive> show tables;
OK
btest2
choice
classinfo
mutil2
ptest
userinfo
xp
Time taken: 0.044 seconds
```

对于**托管表**，**drop**操作会把**元数据**和**数据文件**删除掉，对于**外部表**，只是**删除元数据**。如果只要删除表中的数据，保留表名可以在 HDFS 上删除数据文件：

```
hive>dfs -rmr /user/hive/warehouse/mutill1/*
```

8) 连接

连接是将**两个表**中在**共同数据项**上**相互匹配**的那些**行合并起来**，**HiveQL**的**连接**分为**内连接**、**左向外连接**、**右向外连接**、**全外连接**和**半连接**5种。

- **内连接**

内连接使用**比较运算符**根据**每个表**共有的列的**值匹配**两个表中的行。例如，检索 userinfo 和 choice 表中标识号相同的所有行。

```
hive>select userinfo.*,choice.* from userinfo join choice on (userinfo.id=choice.userid);
```

```
MapReduce Total cumulative CPU time: 3 seconds 830 msec
Ended Job = job_201203240454_0014
MapReduce Jobs Launched:
Job 0: Map: 2 Reduce: 1 Accumulative CPU: 3.83 sec HDFS Read: 516 HDFS Write: 108 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 830 msec
OK
1 xiapi 1 math
1 xiapi 1 china
1 xiapi 1 english
2 xiaoxue 2 china
2 xiaoxue 2 english
3 qingqing 3 english
Time taken: 31.985 seconds
```

- **左向外连接**

左向外连接的结果集包括“**LEFT OUTER**”子句中指定的**左表**的**所有行**，而**不仅仅是**连接列所匹配的行。如果**左表**的**某行**在**右表**中**没有匹配行**，则在**相关联的结果集**中**右表**的所有**选择列均为空值**。

```
hive>select userinfo.*,choice.* from userinfo left outer join choice on (userinfo.id=choice.userid)
```

- **右向外连接**

右向外连接是**左向外连接**的**反向连接**，将**返回右表的所有行**。如果**右表的某行在左表中没有匹配行**，则将为**左表返回空值**。

```
hive>select userinfo.*,choice.* from userinfo right outer join choice on (userinfo.id=choice.userid)
```

- **全外连接**

全外连接返回**左表和右表中的所有行**。当某行在另一表中没有匹配行时，则另一个表的选择列表包含空值。如果表之间有匹配行，则**整个结果集**包含基表的数据值。

```
hive>select userinfo.*,choice.* from userinfo full outer join choice on (userinfo.id=choice.userid)
```

- **半连接**

半连接是**Hive 所特有的**，**Hive 不支持 IN 操作**，但是**拥有替代的方案**：**left semi join**，称为**半连接**，**需要注意的是连接的表不能在查询的列中**，**只能出现在 on 子句中**。

```
hive>select userinfo.* from userinfo left semi join choice on (userinfo.id=choice.userid);
```

9) 子查询

标准 SQL 的子查询支持嵌套的 select 子句，**HiveQL 对子查询的支持很有限**，**只能在 from 引导的子句中**出现子查询。如下语句在 from 子句中嵌套了一个子查询（实现了对教课最多的老师的查询）。

```
hive>select teacher,MAX(class_num)
from (select teacher,count(classname) as class_num from classinfo group by teacher) subq
group by teacher;
```

10) 视图操作

目前，只有 Hive0.6 之后的版本才支持视图。

Hive 只支持逻辑视图，并不支持物理视图，建立视图可以在 **MySQL 元数据库**中看到创建的**视图表**，但是在 **Hive 的数据仓库目录**下**没有相应的视图表目录**。

当一个查询引用一个视图时，可以评估视图的定义并为下一步查询提供记录集合。这是一种概念的描述，实际上，作为查询优化的一部分，Hive 可以将视图的定义与查询的定义结合起来，例如从查询到视图所使用的过滤器。

在视图创建的同时确定视图的架构，如果随后再改变基本表（如添加一列）将不会在视图的架构中体现。如果基本表被删除或以不兼容的方式被修改，则该视图的查询将被无效。

视图是只读的，**不能用于 LOAD/INSERT/ALTER**。

视图可能包含 ORDER BY 和 LIMIT 子句，如果一个引用了视图的查询也包含这些子句，那么在执行这些子句时首先要查看视图语句，然后返回结果按照视图中的语句执行。

以下是创建视图的例子：

```
hive>create view teacher_classsum as
select teacher,count(classname) from classinfo group by teacher;
```

```
hive> create view teacher_classsum as
> select teacher,count(classname) from classinfo group by teacher;
OK
Time taken: 0.426 seconds
hive> select * from teacher_classsum;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201203240454_0015, Tracking URL = http://Master.Hadoop:50030/jobdetails.jsp?jobid=job_201203240454_0015
Kill Command = /usr/hadoop/libexec/./bin/hadoop job -Dmapred.job.tracker=http://192.168.1.2:9001 -kill job_201203240454_0015
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2012-03-26 05:20:12,770 Stage-1 map = 0%, reduce = 0%
2012-03-26 05:20:18,783 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.92 sec
2012-03-26 05:20:19,786 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.92 sec

MapReduce Total cumulative CPU time: 2 seconds 720 msec
Ended Job = job_201203240454_0015
MapReduce Jobs Launched:
Job 0: Map: 1 Reduce: 1 Accumulative CPU: 2.72 sec HDFS Read: 254 HDFS Write: 20 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 720 msec
OK
jack 1
lucy 1
sam 1
Time taken: 31.985 seconds
```

以下是删除视图的例子，命令如下：

```
hive>drop view teacher_classnum
```

```
Time taken: 31.985 seconds
hive> drop view teacher_classnum;
OK
Time taken: 0.186 seconds
hive>
```

备注： drop view 为删除指定视图的元数据，在视图中使用 drop table 是错误的。

11) 函数操作

- 创建函数

```
create temporary function function_name as class_name
```

该语句创建一个由类名实现的函数。在 Hive 中用户可以使用 Hive 类路径中的任何类，用户通过执行 add files 语句将函数类添加到类路径，并且可持续使用该函数进行操作。

- 删除函数

注销用户定义函数的格式如下：

```
drop temporary function function_name
```

3、Hive开发配置

3.1 开发环境

Java 版本: jdk-**6u31**-windows-i586.exe

Win 系统: Windows 7 旗舰版

Eclipse 软件: eclipse-jee-**indigo**-SR1-win32.zip | eclipse-jee-**helios**-SR2-win32.zip

Hadoop 软件: hadoop-1.0.0.tar.gz

Hive 软件: hive-0.8.1.tar.gz

3.2 配置Eclipse

1) 官方配置

```
# To run the program in standalone mode, we need the following jars in the classpath
# from hive/build/dist/lib
#   hive_exec.jar
#   hive_jdbc.jar
#   hive_metastore.jar
#   hive_service.jar
#   libfb303.jar
#   log4j-1.2.15.jar
#
# from hadoop/build
#   hadoop-*-core.jar
#
# To run the program in embedded mode, we need the following additional jars in the classpath
# from hive/build/dist/lib
#   antlr-runtime-3.0.1.jar
#   derby.jar
#   jdo2-api-2.1.jar
#   jpox-core-1.2.2.jar
#   jpox-rdbms-1.2.2.jar
#
# as well as hive/build/dist/conf
```

2) 书上配置

导入 Hive 的 lib 目录下的所有包, 导入 Hadoop 的核心包, 导入 MySQL 的 JDBC 驱动包。

3) 最终配置

通过 Eclipse 创建一个新 Java 工程, 右击项目根目录, 选择“Properties→Java Build Path→Library→Add External JARs”, 将 HBase 安装文件解压后 lib 子目录下所有的 jar 包、Hadoop

安装文件解压后根目录下的 **hadoop-core-1.0.0.jar** 以及 MySQL 的 JDBC 驱动包，添加到本工程的 **Classpath** 下。

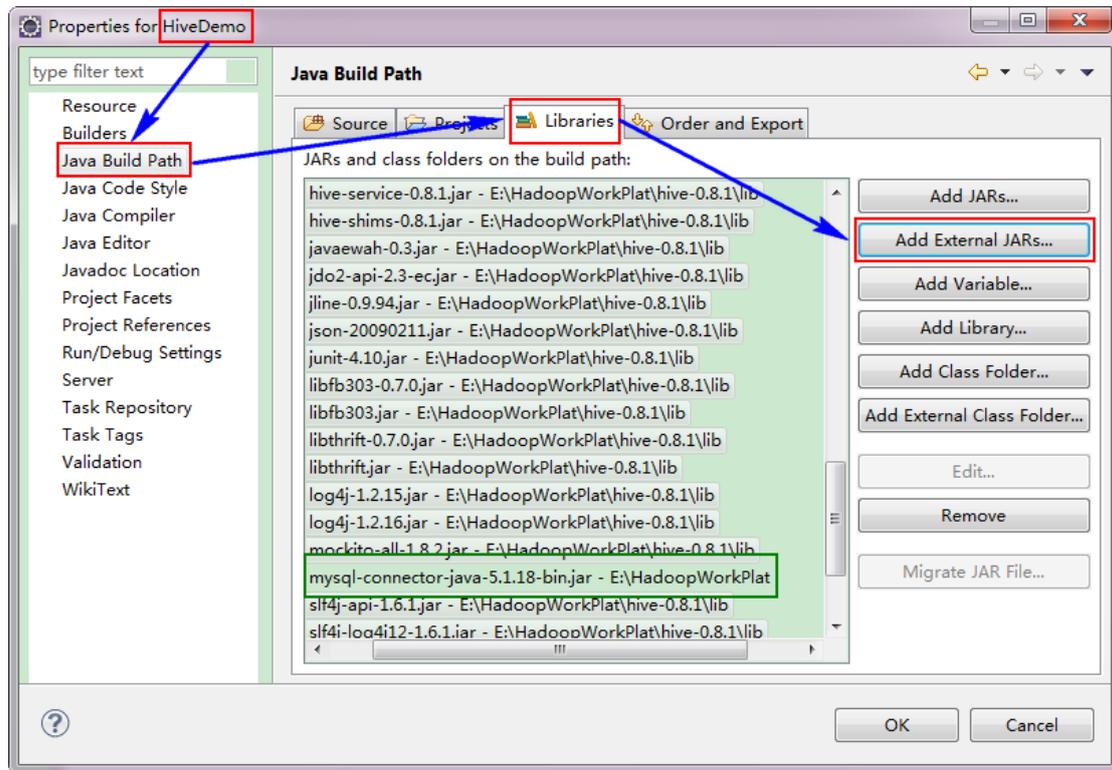


图 3.2-1 Java 工程添加额外 Jar 包

4、Hive初级案例

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：

```
nohup hive -service hiveserver &
```

4.1 实战 1——Hive与JDBC示例

1) 测试数据

test.txt 文件内容：

```
1 xiapi
2 xiaoxue
3 qingqing
```

2) 程序代码

```
package com.hebut.hive;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class HiveJdbcClient {

    private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";
    private static String sql = "";
    private static ResultSet res;

    public static void main(String[] args) throws SQLException {
        try {
            Class.forName(driverName);
        } catch (ClassNotFoundException err) {
            err.printStackTrace();
            System.exit(1);
        }

        Connection conn = DriverManager.getConnection(
            "jdbc:hive://192.168.1.2:10000/default", "hive", "hadoop");
        Statement stmt = conn.createStatement();

        // 创建的表名
        String tableName = "testHiveDriverTable";

        /** 第一步：存在就先删除 */
        sql = "drop table " + tableName;
        stmt.executeQuery(sql);

        /** 第二步：不存在就创建 */
        sql = "create table " + tableName + " (key int,value string) ";
        sql += " row format delimited fields terminated by '\t'";
        stmt.executeQuery(sql);

        // 执行“show tables”操作
        sql = "show tables '" + tableName + "'";
        System.out.println("Running:" + sql);
        res = stmt.executeQuery(sql);
        System.out.println("执行“show tables”运行结果: ");
        if (res.next()) {
```

```
        System.out.println(res.getString(1));
    }

    // 执行“describe table”操作
    sql = "describe " + tableName;
    System.out.println("Running:" + sql);
    res = stmt.executeQuery(sql);
    System.out.println("执行“describe table”运行结果: ");
    while (res.next()) {
        System.out.println(res.getString(1) + "\t" + res.getString(2));
    }

    // 执行“load data into table”操作
    String filepath = "/home/hadoop/test.txt";
    sql = "load data local inpath '" + filepath + "' into table "
        + tableName;
    System.out.println("Running:" + sql);
    res = stmt.executeQuery(sql);

    // 执行“select * query”操作
    sql = "select * from " + tableName;
    System.out.println("Running:" + sql);
    res = stmt.executeQuery(sql);
    System.out.println("执行“select * query”运行结果: ");
    while (res.next()) {
        System.out.println(res.getInt(1) + "\t" + res.getString(2));
    }

    // 执行“regular hive query”操作
    sql = "select count(1) from " + tableName;
    System.out.println("Running:" + sql);
    res = stmt.executeQuery(sql);
    System.out.println("执行“regular hive query”运行结果: ");
    while (res.next()) {
        System.out.println(res.getString(1));
    }
}
}
```

3) 运行结果

- Eclipse 输出端

```
Running:show tables 'testHiveDriverTable'
```

```

执行“show tables”运行结果:
testhivedrivertable
Running:describe testHiveDriverTable
执行“describe table”运行结果:
key int
value string
Running:load data local inpath '/home/hadoop/test.txt' into table
testHiveDriverTable
Running:select * from testHiveDriverTable
执行“select * query”运行结果:
1 xiapi
2 xiaoxue
3 qingqing
Running:select count(1) from testHiveDriverTable
执行“regular hive query”运行结果:
3

```

● Hadoop 集群端

```

[hadoop@Master ~]$ Hive history file=/tmp/hadoop/hive_job_log_hadoop_201203261926_716914750.txt
OK
OK
OK
OK
OK
Copying data from file:/home/hadoop/test.txt
Copying file: file:/home/hadoop/test.txt
Loading data to table default.testhivedrivertable
OK
OK
OK
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201203240454_0018, Tracking URL = http://Master.Hadoop:50030/jobdetails.jsp?jobid=job_201203240454_0018
Kill Command = /usr/hadoop/libexec/./bin/hadoop job -Dmapred.job.tracker=http://192.168.1.2:9001 -kill job_201203240454_0018
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2012-03-26 21:00:51,895 Stage-1 map = 0%, reduce = 0%
2012-03-26 21:00:57,915 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.87 sec
2012-03-26 21:00:58,920 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.87 sec
2012-03-26 21:00:59,924 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.87 sec
2012-03-26 21:01:00,927 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.87 sec
2012-03-26 21:01:01,931 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.87 sec
2012-03-26 21:01:02,934 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.87 sec
2012-03-26 21:01:03,938 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.87 sec
2012-03-26 21:01:04,943 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 0.87 sec

```

4.2 实战 2——Hadoop 的日志分析

1) 日志格式分析

首先分析 Hadoop 的日志格式，日志是一行一条，日志格式可以依次描述为：日期、时间、级别、相关类和提示信息。如下所示：

```

2012-03-16 01:31:00,730 INFO org.apache.hadoop.hdfs.server.namenode.NameNode:
STARTUP_MSG:

```

```

/*****
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = Master.Hadoop/192.168.1.2
STARTUP_MSG:   args = []
STARTUP_MSG:   version = 1.0.0
STARTUP_MSG:                                     build           =
https://svn.apache.org/repos/asf/hadoop/common/branches/branch-1.0 -r 1214675; compiled by
'hortonfo' on Thu Dec 15 16:36:35 UTC 2011
*****/

2012-03-16 01:31:00,863 INFO org.apache.hadoop.metrics2.impl.MetricsConfig: loaded
properties from hadoop-metrics2.properties
2012-03-16 01:31:00,874 INFO org.apache.hadoop.metrics2.impl.MetricsSourceAdapter: MBean
for source MetricsSystem,sub=Stats registered.
2012-03-16 01:31:00,874 INFO org.apache.hadoop.metrics2.impl.MetricsSystemImpl: Scheduled
snapshot period at 10 second(s).
2012-03-16 01:31:00,875 INFO org.apache.hadoop.metrics2.impl.MetricsSystemImpl:
NameNode metrics system started
2012-03-16 01:31:00,976 INFO org.apache.hadoop.metrics2.impl.MetricsSourceAdapter: MBean
for source ugi registered.
2012-03-16 01:31:00,984 INFO org.apache.hadoop.metrics2.impl.MetricsSourceAdapter: MBean
for source jvm registered.
2012-03-16 01:31:00,984 INFO org.apache.hadoop.metrics2.impl.MetricsSourceAdapter: MBean
for source NameNode registered.
2012-03-16 01:31:01,008 INFO org.apache.hadoop.hdfs.util.GSet: VM type           = 32-bit
2012-03-16 01:31:01,008 INFO org.apache.hadoop.hdfs.util.GSet: 2% max memory = 17.77875
MB
2012-03-16 01:31:01,008 INFO org.apache.hadoop.hdfs.util.GSet: capacity           = 2^22 =
4194304 entries
.....

```

可以采用空格来对行内分隔，但是提高信息中可能也有空格，可以把提示信息组织成多个列存放，最后查询的时候把这个几列进行合并即可，在这里提示信息用三列来组织。另外日期分两部分，可以用复杂了类型 ARRAY 来组织，以逗号分隔。

表 4.2-1 hadoop 日志存储表描述

rdate	time	type	relateclass	information1	information2	information3
string	array<string>	string	string	string	string	string

表的定义如下：

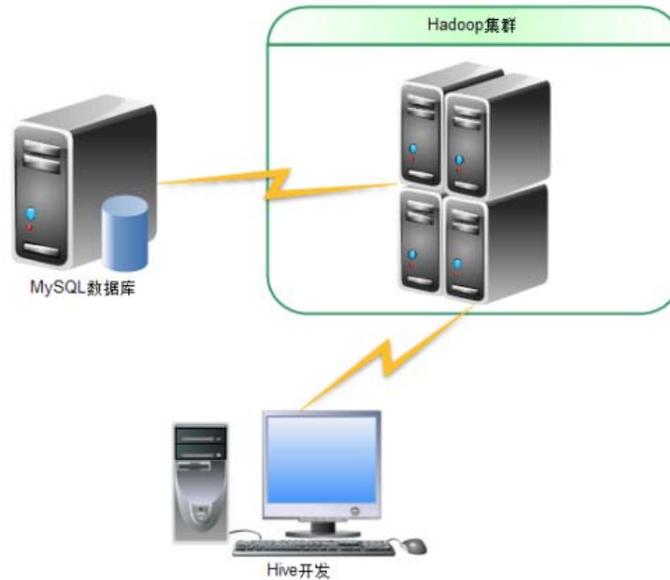
```

create table if not exists loginfo(rdate string,time array<string>,type string,relateclass
string,information1 string,information2 string,information3 string) row format delimited fields
terminated by ' ' collection items terminated by ',' map keys terminated by ':'

```

2) 程序设计

本程序是在个人机器用 Eclipse 开发，该程序连接 Hadoop 集群，处理完的结果存储在 MySQL 服务器上。下面是程序开发示例图。



MySQL 数据库的存储信息的表 “hadooplog” 的 SQL 语句如下：

```
DROP TABLE IF EXISTS `hive`.`hadooplog`;
CREATE TABLE `hive`.`hadooplog` (
  `id` int(11) NOT NULL auto_increment,
  `rdate` varchar(50) default NULL,
  `time` varchar(50) default NULL,
  `type` varchar(50) default NULL,
  `relateclass` TINYTEXT default NULL,
  `information` LONGTEXT default NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

创建数据库 “hive”：

```
mysql> create database hive;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| hive |
| mysql |
| performance_schema |
| school |
| test |
+-----+
6 rows in set (0.00 sec)
```

导入 SQL 语句创建表 “hadooplog”:

```
mysql> source /home/hadoop/hadooplog.sql
Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 0 rows affected (0.17 sec)

mysql> use hive;
Database changed
mysql> show tables;
+-----+
| Tables_in_hive |
+-----+
| hadooplog      |
+-----+
1 row in set (0.00 sec)

mysql> describe hadooplog;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id         | int(11)       | NO   | PRI | NULL    | auto_increment |
| rdate     | varchar(50)   | YES  |     | NULL    |                |
| time      | varchar(50)   | YES  |     | NULL    |                |
| type      | varchar(50)   | YES  |     | NULL    |                |
| relateclass | tinytext     | YES  |     | NULL    |                |
| information | longtext     | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

3) 程序代码

```
package com.hebut.hive;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

/**
 * 该类的主要功能是负责建立与 Hive 和 MySQL 的连接,
 * 由于每个连接的开销比较大, 所以此类的设计采用
 * 设计模式中的单件模式。
 */
class DBHelper {
    private static Connection connToHive = null;
    private static Connection connToMySQL = null;

    private DBHelper() {
```

```
}

// 获得与 Hive 连接，如果连接已经初始化，则直接返回
public static Connection getHiveConn() throws SQLException {
    if (connToHive == null) {
        try {
            Class.forName("org.apache.hadoop.hive.jdbc.HiveDriver");
        } catch (ClassNotFoundException err) {
            err.printStackTrace();
            System.exit(1);
        }

        connToHive = DriverManager.getConnection(
            "jdbc:hive://192.168.1.2:10000/default", "hive", "hadoop");
    }
    return connToHive;
}

// 获得与 MySQL 连接
public static Connection getMySQLConn() throws SQLException {
    if (connToMySQL == null) {
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException err) {
            err.printStackTrace();
            System.exit(1);
        }

        connToMySQL = DriverManager.getConnection(
            "jdbc:mysql://192.168.1.10:3306/hive?useUnicode=true
            &characterEncoding=UTF-8", "root", "hadoop");
    }
    return connToMySQL;
}

public static void closeHiveConn() throws SQLException {
    if (connToHive != null) {
        connToHive.close();
    }
}

public static void closeMySQLConn() throws SQLException {
    if (connToMySQL != null) {
        connToMySQL.close();
    }
}
```

```
    }  
  }  
}  
  
/**  
 *  
 * 针对 Hive 的工具类  
 */  
class HiveUtil {  
    // 创建表  
    public static void createTable(String sql) throws SQLException {  
        Connection conn = DBHelper.getHiveConn();  
        Statement stmt = conn.createStatement();  
        ResultSet res = stmt.executeQuery(sql);  
    }  
  
    // 依据条件查询数据  
    public static ResultSet queryData(String sql) throws SQLException {  
        Connection conn = DBHelper.getHiveConn();  
        Statement stmt = conn.createStatement();  
        ResultSet res = stmt.executeQuery(sql);  
        return res;  
    }  
  
    // 加载数据  
    public static void loadData(String sql) throws SQLException {  
        Connection conn = DBHelper.getHiveConn();  
        Statement stmt = conn.createStatement();  
        ResultSet res = stmt.executeQuery(sql);  
    }  
  
    // 把数据存储到 MySQL 中  
    public static void hiveToMySQL(ResultSet res) throws SQLException {  
        Connection conn = DBHelper.getMySQLConn();  
        Statement stmt = conn.createStatement();  
        while (res.next()) {  
            String rdate = res.getString(1);  
            String time = res.getString(2);  
            String type = res.getString(3);  
            String relateclass = res.getString(4);  
            String information = res.getString(5) + res.getString(6)  
                + res.getString(7);  
        }  
    }  
}
```

```
StringBuffer sql = new StringBuffer();
sql.append("insert into hadooplog values(0,");
sql.append(rdate + "','");
sql.append(time + "','");
sql.append(type + "','");
sql.append(relatedclass + "','");
sql.append(information + "'");

    int i = stmt.executeUpdate(sql.toString());
}
}
}

public class AnalyzeHadoopLog {

    public static void main(String[] args) throws SQLException {
        StringBuffer sql = new StringBuffer();

        // 第一步：在 Hive 中创建表
        sql.append("create table if not exists loginfo(rdate string,");
        sql.append("time array<string>,type string,relatedclass string,");
        sql.append("information1 string,information2 string,information3 ");
        sql.append("string) row format delimited fields terminated by ' '");
        sql.append(" collection items terminated by ','");
        sql.append(" map keys terminated by ':'");

        System.out.println(sql);
        HiveUtil.createTable(sql.toString());

        // 第二步：加载 Hadoop 日志文件
        sql.delete(0, sql.length());
        sql.append("load data local inpath ");
        sql.append("' /home/hadoop/hadoop.log'");
        sql.append(" overwrite into table loginfo");
        System.out.println(sql);
        HiveUtil.loadData(sql.toString());

        // 第三步：查询有用信息
        sql.delete(0, sql.length());
        sql.append("select rdate,time[0],type,relatedclass,");
        sql.append("information1,information2,information3 ");
        sql.append("from loginfo where type='ERROR'");
        System.out.println(sql);
        ResultSet res = HiveUtil.queryData(sql.toString());
    }
}
```

```

// 第四步：查出的信息经过变换后保存到 MySQL 中
HiveUtil.hiveToMySQL(res);

// 第五步：关闭 Hive 连接
DBHelper.closeHiveConn();

// 第六步：关闭 MySQL 连接
DBHelper.closeMySQLConn();
}
}

```

4) 运行结果

程序执行完之后进入 **MySQL** 的控制台，查看 **hadooplog** 表中的结果信息如下。

```

mysql> select * from hadooplog;
+-----+-----+-----+-----+-----+-----+
| id | rdate | time | type | relateclass | information |
+-----+-----+-----+-----+-----+-----+
| 1 | 2012-03-16 | 01:31:05 | ERROR | org.apache.hadoop.security.UserGroupInformation: | PrivilegedActionExceptions:hadoopcause:org.apache |
| 2 | 2012-03-16 | 01:31:15 | ERROR | org.apache.hadoop.security.UserGroupInformation: | PrivilegedActionExceptions:hadoopcause:org.apache |
| 3 | 2012-03-16 | 01:31:25 | ERROR | org.apache.hadoop.security.UserGroupInformation: | PrivilegedActionExceptions:hadoopcause:org.apache |
| 4 | 2012-03-16 | 01:31:35 | ERROR | org.apache.hadoop.security.UserGroupInformation: | PrivilegedActionExceptions:hadoopcause:org.apache |
| 5 | 2012-03-16 | 03:36:40 | ERROR | org.apache.hadoop.security.UserGroupInformation: | PrivilegedActionExceptions:hadoopcause:org.apache |
| 6 | 2012-03-16 | 03:36:50 | ERROR | org.apache.hadoop.security.UserGroupInformation: | PrivilegedActionExceptions:hadoopcause:org.apache |
| 7 | 2012-03-16 | 03:37:00 | ERROR | org.apache.hadoop.security.UserGroupInformation: | PrivilegedActionExceptions:hadoopcause:org.apache |
| 8 | 2012-03-16 | 03:37:04 | ERROR | org.apache.hadoop.security.UserGroupInformation: | PrivilegedActionExceptions:hadoopcause:org.apache |
| 9 | 2012-03-16 | 03:37:10 | ERROR | org.apache.hadoop.security.UserGroupInformation: | PrivilegedActionExceptions:hadoopcause:org.apache |
| 10 | 2012-03-16 | 03:37:15 | ERROR | org.apache.hadoop.security.UserGroupInformation: | PrivilegedActionExceptions:hadoopcause:org.apache |
| 11 | 2012-03-16 | 03:37:20 | ERROR | org.apache.hadoop.security.UserGroupInformation: | PrivilegedActionExceptions:hadoopcause:org.apache |
| 12 | 2012-03-16 | 04:36:47 | ERROR | org.apache.hadoop.security.UserGroupInformation: | PrivilegedActionExceptions:hadoopcause:org.apache |

```

5) 经验总结

在示例中同时对 **Hive** 的数据仓库和 **MySQL** 数据库进行操作，虽然都是使用了 **JDBC** 接口，但是一些地方还是有差异的，这个**实战示例**能比较好地体现 **Hive** 与关系型数据库的异同。

如果我们**直接**采用 **MapReduce** 来做，**效率**会比使用 **Hive** 高，因为 **Hive** 的**底层**就是调用了 **MapReduce**，但是程序的**复杂度**和**编码量**都会**大大增加**，特别是对于**不熟悉** **MapReduce** 编程的开发人员，这是一个棘手问题。**Hive** 在这两种方案中找到了**平衡**，**不仅**处理效率较高，而且**实现**起来也相对简单，给传统关系型数据库编码人员带来了**便利**，这就是目前 **Hive** 被许多商业组织所采用的**原因**。

编者简介

基本信息

姓名：解耀伟 性别：男
笔名：虾皮 民族：汉
学历：研究生 专业：计算机应用技术
电子信箱：xieyaowei1986@163.com
学校：河北工业大学（211 工程）



求职意向

希望在 IT 行业从事软件研发等工作。

编程语言

Java、C#、C、ExtJS、Flex、汇编、PHP、VB，熟练程度由左到右逐级减弱。

个人经历

大学期间

- 1) 担任职务：学生会生活部部长、生活委员、团支书
- 2) 获得奖项：二等奖学金（2 次）、三好学生（1 次）

研究生期间

- 1) 担任职务：班长
- 2) 获得奖项：优秀班干部（1 次）

工作经历

实验室项目：国家 863 计划项目 1 项；国家技术基础专项 2 项；河北省技术专项 1 项。

研究生课题：基于 Hadoop 分布式搜索引擎研究

个人评价

性格开朗，善于与人沟通，上进心强，品德优秀，吃苦耐劳，喜欢团队合作，能积极服从上级的安排。

寄 言

相信您的信任与我的能力将为我们带来共同的成功。

参考文献

感谢以下文章的编写者，没有你们的铺路，我或许会走得很艰难，参考不分先后，贡献同等珍贵。

【1】Hadoop 实战——陆嘉恒——机械工业出版社

【2】实战 Hadoop——刘鹏——电子工业出版社

【3】通过 JDBC 驱动连接 Hive 操作实例

地址：http://blog.csdn.net/kunshan_shenbin/article/details/7214491

【4】java 通过 jdbc 驱动连接 hive 操作实例

地址：<http://blog.csdn.net/a221133/article/details/6734762>

【5】Hive 的 JDBC 连接

地址：<http://cloud.csdn.net/a/20101128/282618.html>

【6】基于 Hive 的日志数据统计实战

地址：<http://cloud.csdn.net/a/20101128/282620.html>

【7】HiveClient

地址：<https://cwiki.apache.org/confluence/display/Hive/HiveClient>