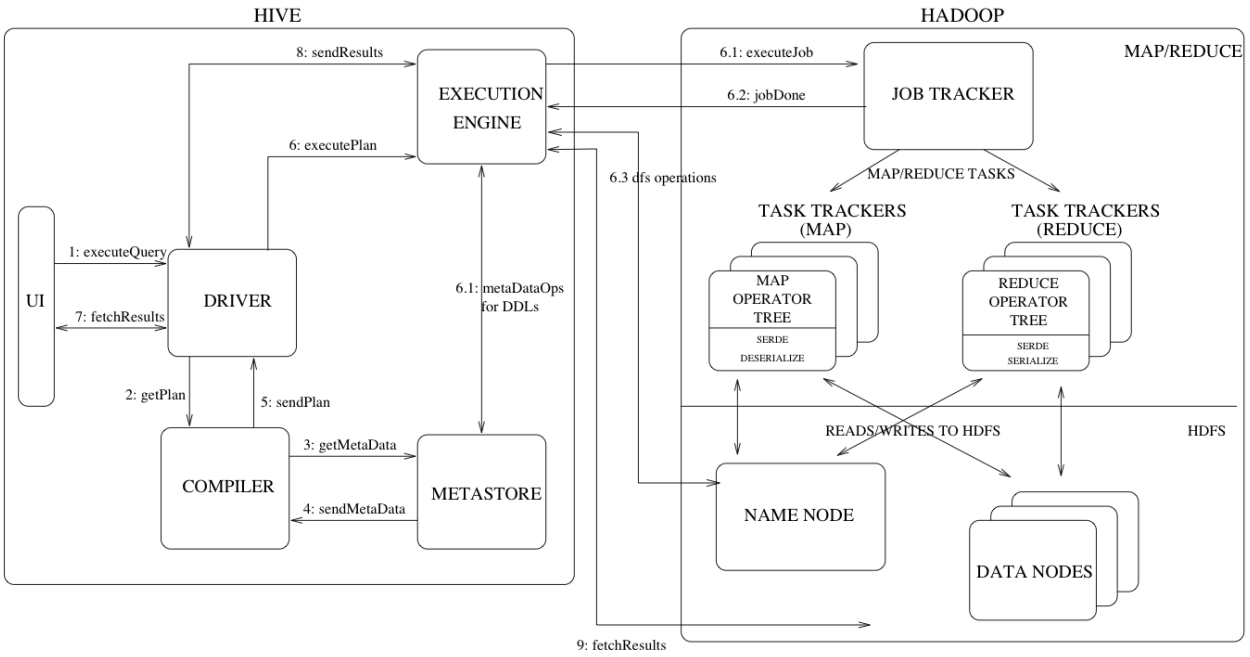
1. Hive原理

1.1 Hive架构



来自：https://cwiki.apache.org/confluence/display/Hive/Design

Hive包括UI、Driver、Compiler、Metastore和Execution Engine等主要组件：

- UI：用户接口，用于提交查询语句或系统操作语句，例如，Beeline或Hue。
- Driver：接收查询，处理会话、提供JDBC/ODBC上的execute和fetch接口。
- Compiler：解析查询并生成查询计划。
- Metastore：存储表（table）和分区（partition）的结构信息，包括列和列类型，序列化和反序列化类等信息。
- Execution Engine：执行Compiler生成的查询计划，一个查询计划是一个包含多个stage的有向无环图，每个stage是以下三种操作之一：map/reduce任务、元数据操作或HDFS操作。

2. Hive安装配置

安装主机：

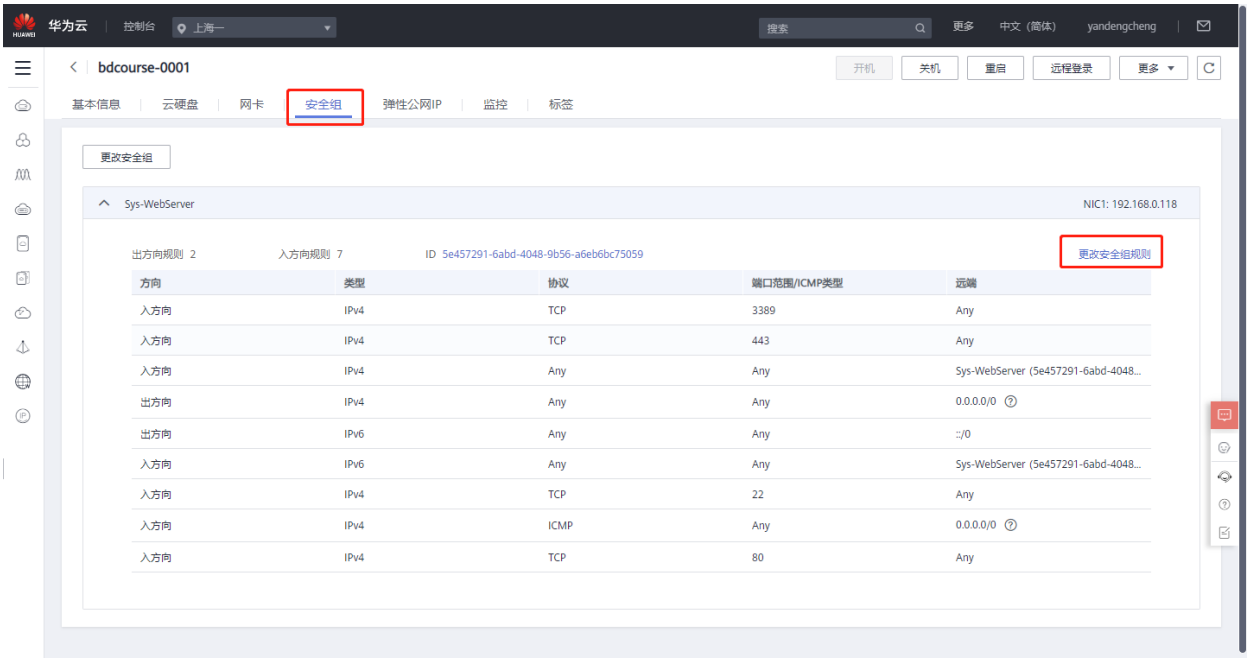
- bdcourse-0001

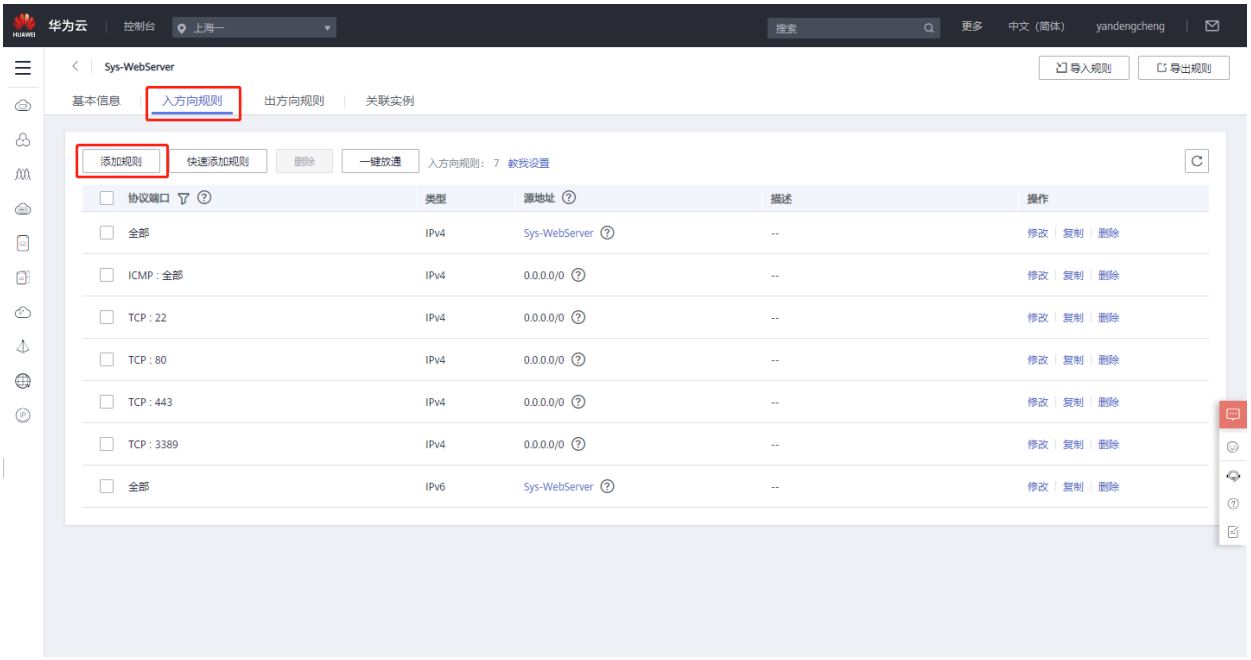
2.0 准备 - 安装MySQL

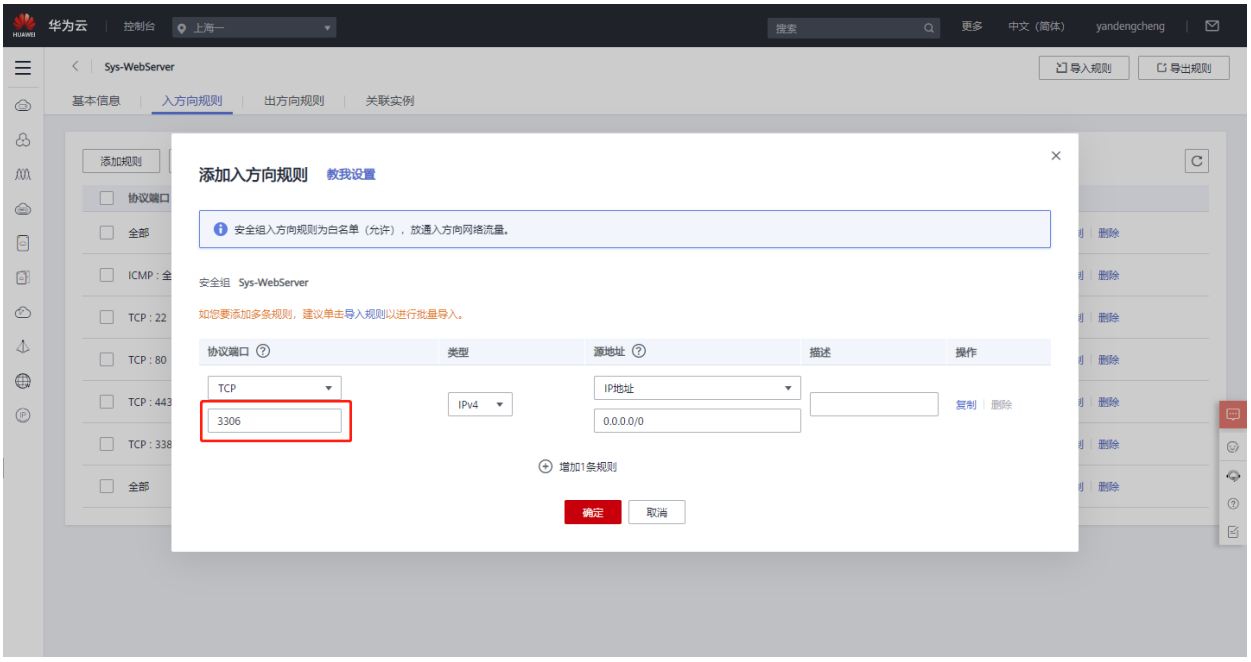
```
$ apt update
$ apt install mysql-server
$ mysql_secure_installation
# root密码Bd2020
# 其中VALIDATE PASSWORD PLUGIN选择n
```

```
# 其中 Disallow root login remotely? (Press y|Y for Yes, any other key for No) : n

# 允许root远程访问
$ mysql -uroot -p
mysql> GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY 'Bd2020';
mysql> FLUSH PRIVILEGES;

# 修改mysql配置
$ vi /etc/mysql/mysql.conf.d/mysqld.cnf
# 注释掉bind-address           = 127.0.0.1
$ systemctl restart mysql
```

配置华为云安全组，允许MySQL数据库3306端口入站







配置Hive元数据库

创建数据库`hivemetastore`。

2.1 下载、安装Hive

注意：文档中关于`PATH`的修改都是在原有`PATH`基础之上增加，**切勿按照示例里面直接替换。**

```
$ wget http://mirrors.ustc.edu.cn/apache/hive/hive-2.3.7/apache-hive-2.3.7-bin.tar.gz
$ tar -xzvf apache-hive-2.3.7-bin.tar.gz

$ mv apache-hive-2.3.7-bin /opt/
$ ln -s /opt/apache-hive-2.3.7-bin /opt/hive

$ vi /etc/profile

# 添加、修改Hive相关环境变量
export HIVE_HOME=/opt/hive
# 注意：这里需要在原有PATH变量的前面添加$HIVE_HOME/bin，而不是全部改成下面的情况
export PATH=$HIVE_HOME/bin:$PATH

# 使环境变量生效
$ source /etc/profile
```

2.2 下载MySQL Connector/J

```
$ wget https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-8.0.19.tar.gz
$ tar -xzvf mysql-connector-java-8.0.19.tar.gz
$ mv mysql-connector-java-8.0.19/mysql-connector-java-8.0.19.jar $HIVE_HOME/lib/
```

2.3 在HDFS中为Hive创建相关目录

```
$ $HADOOP_HOME/bin/hadoop fs -mkdir /tmp
$ $HADOOP_HOME/bin/hadoop fs -mkdir /user
$ $HADOOP_HOME/bin/hadoop fs -mkdir /user/hive
$ $HADOOP_HOME/bin/hadoop fs -mkdir /user/hive/warehouse
```

2.4 配置Hadoop代理用户

Hadoop允许配置以代理用户的身份提交任务或访问HDFS，用户代理按照主机（host）、用户组（group）和用户（user）三个方面进行配置，表示允许在指定主机上模拟指定用户组和指定用户。

```
vi $HADOOP_HOME/etc/hadoop/core-site.xml

# 添加如下配置
    <property>
        <name>hadoop.proxyuser.root.hosts</name>
        <value>*</value>
    </property>
    <property>
        <name>hadoop.proxyuser.root.groups</name>
        <value>*</value>
    </property>
    <property>
        <name>hadoop.proxyuser.root.users</name>
        <value>root</value>
    </property>

# 重启Hadoop
${HADOOP_HOME}/sbin/stop-dfs.sh
${HADOOP_HOME}/sbin/stop-yarn.sh
${HADOOP_HOME}/sbin/mr-jobhistory-daemon.sh stop historyserver

${HADOOP_HOME}/sbin/start-dfs.sh
${HADOOP_HOME}/sbin/start-yarn.sh
${HADOOP_HOME}/sbin/mr-jobhistory-daemon.sh start historyserver
```

参考：

- https://hadoop.apache.org/docs/r2.7.7/hadoop-project-dist/hadoop-common/Superusers.html
- https://my.oschina.net/OttoWu/blog/806814

2.5 配置Hive

```
# 创建相关文件夹
mkdir /opt/hive/tmp

# 编辑配置文件
$ cp $HIVE_HOME/conf/hive-default.xml.template $HIVE_HOME/conf/hive-site.xml

$ vi $HIVE_HOME/conf/hive-site.xml
```

```
# 修改或新增如下配置信息，其中前两个配置为新增，且放在最前面
  <property>
    <name>system:java.io.tmpdir</name>
    <value>/opt/hive/tmp</value>
  </property>
  <property>
    <name>system:user.name</name>
    <value>${user.name}</value>
  </property>

  <property>
    <name>hive.server2.transport.mode</name>
    <value>binary</value>
  </property>

  <property>
    <name>hive.server2.thrift.bind.host</name>
    <value>bdcourse-0001</value>
  </property>
  <property>
    <name>hive.server2.thrift.port</name>
    <value>10000</value>
  </property>
  <property>
    <name>hive.server2.webui.host</name>
    <value>bdcourse-0001</value>
  </property>
  <property>
    <name>hive.server2.webui.port</name>
    <value>10002</value>
  </property>

  <property>
    <name>hive.metastore.port</name>
    <value>9083</value>
  </property>
  <property>
    <name>hive.metastore.warehouse.dir</name>
    <value>/user/hive/warehouse</value>
  </property>
  <property>
    <name>hive.metastore.uris</name>
    <value>thrift://bdcourse-0001:9083</value>
  </property>

  <property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://bdcourse-0001:3306/hivemetastore</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.jdbc.Driver</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>root</value>
  </property>
  <property>
```

```
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>Bd2020</value>
  </property>

  <property>
    <name>datanucleus.schema.autoCreateAll</name>
    <value>false</value>
  </property>
  <property>
    <name>hive.metastore.schema.verification</name>
    <value>true</value>
  </property>
```

2.5 初始化数据表

```
$ $HIVE_HOME/bin/schematool -dbType mysql -initSchema
```

2.6 启动Hive MetaStore

手动启动metastore或通过systemd启动，**二选一。**

手动启动

```
$ mkdir $HIVE_HOME/logs
$ nohup $HIVE_HOME/bin/hive --service metastore > $HIVE_HOME/logs/metastore.log &
```

通过systemd启动

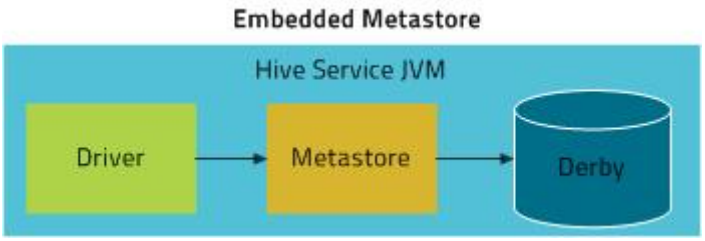
仅需要在master节点启动

```
$ vi /opt/services/hive-metastore.run.sh

#!/bin/bash

source /etc/profile

start() {
  /opt/hive/bin/hive --service metastore
}

case "$1" in
  start)
    start
    ;;
  *)
esac

$ vi /opt/services/hive-metastore.service

[Unit]
Description=Hive metastore
```

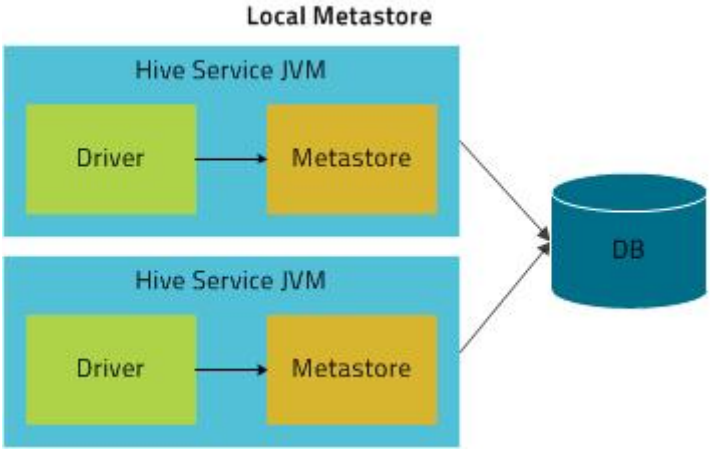
```
After=network.target

[Service]
User=root
Group=root
Type=simple
WorkingDirectory=/opt/hive
ExecStart=/opt/services/hive-metastore.run.sh start
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target

# 修改文件属性，使其可执行
$ chmod u+x /opt/services/hive-metastore.run.sh
# 建立软连接，Systemd 默认从目录/etc/systemd/system/读取配置文件
$ ln -s /opt/services/hive-metastore.service /etc/systemd/system/hive-metastore.service
# 重新加载服务的配置文件
$ systemctl daemon-reload
# 设置开机启动（通过在/etc/systemd/system/multi-user.target.wants/创建软连接实现）
$ systemctl enable hive-metastore
# 启动服务
$ systemctl start hive-metastore
# 查看服务状态
$ systemctl status hive-metastore
# 停止服务
$ systemctl stop hive-metastore
```

Hive Metastore有三种部署模式：Embedded Mode，Local Mode和Remote Mode。

Embedded Mode：Derby数据库和metastore service嵌入在HiveServer进程中。



Local Mode：metastore service嵌入在HiveServer进程中，metastore数据库单独运行，metastore service通过JDBC和metastore数据库通讯。



Remote Mode：metastore service运行在独立的JVM进程，HiveServer2，HCatalog，Impala等其他进程通过Thrift接口与metastore service通讯（`hive.metastore.uris`配置），metastore service通过JDBC和metastore数据库通讯（`javax.jdo.option.ConnectionURL`配置）

Remote mode还可以**使用Zookeeper配置动态服务发现**

参数：

- `hive.metastore.service.discovery.mode`：配置为`zookeeper`，当一个metastore启动的时候加入到zookeeper，关机的时候从zookeeper移除，client和server都需要有该配置
- `hive.metastore.uris`：逗号分割的`<zookeeper host>:<port>`
- `hive.metastore.zookeeper.client.port`：如果所有的ZooKeeper都具有相同的端口时使用该配置
- `hive.metastore.zookeeper.namespace`
- `hive.metastore.zookeeper.session.timeout`
- `hive.metastore.zookeeper.connection.timeout`
- `hive.metastore.zookeeper.connection.max.retries`
- `hive.metastore.zookeeper.connection.basesleeptime`

参考：

- https://cwiki.apache.org/confluence/display/Hive/AdminManual+Metastore+Administration
- https://docs.cloudera.com/documentation/enterprise/5-9-x/topics/cdh_ig_hive_metastore_configure.html

2.7 启动HiveServer2

手动启动hiveserver2或通过systemd启动，**二选一。**

手动启动

```
$ nohup $HIVE_HOME/bin/hive --service hiveserver2 > $HIVE_HOME/logs/hiveserver2.log &
```

通过systemd启动

仅需要在master节点启动

```
$ vi /opt/services/hive-hiveserver2.run.sh

#!/bin/bash

source /etc/profile

start() {
  /opt/hive/bin/hive --service hiveserver2
}

case "$1" in
  start)
    start
    ;;
  *)
esac

$ vi /opt/services/hive-hiveserver2.service

[Unit]
Description=Hive hiveserver2
After=network.target

[Service]
User=root
Group=root
Type=simple
WorkingDirectory=/opt/hive
ExecStart=/opt/services/hive-hiveserver2.run.sh start
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target

# 修改文件属性，使其可执行
$ chmod u+x /opt/services/hive-hiveserver2.run.sh
# 建立软连接，Systemd 默认从目录/etc/systemd/system/读取配置文件
$ ln -s /opt/services/hive-hiveserver2.service /etc/systemd/system/hive-
hiveserver2.service
# 重新加载服务的配置文件
$ systemctl daemon-reload
# 设置开机启动（通过在/etc/systemd/system/multi-user.target.wants/创建软连接实现）
$ systemctl enable hive-hiveserver2
# 启动服务
$ systemctl start hive-hiveserver2
# 查看服务状态
$ systemctl status hive-hiveserver2
# 停止服务
$ systemctl stop hive-hiveserver2
```

HiveServer2（HS2）提供远程客户端执行Hive查询并获取结果的服务接口（server interface），基于Thrift RPC，支持多客户端并发连接和认证。HS2 is a single process running as a composite

service, which includes the Thrift-based Hive service (TCP or HTTP) and a Jetty web server for web UI.

Thrift配置项（`hive-site.xml`）：

- `hive.server2.thrift.bind.host`：绑定的TCP地址
- `hive.server2.thrift.port`：TCP端口，默认10000
- `hive.server2.thrift.min.worker.threads`： 默认5
- `hive.server2.thrift.max.worker.threads`： 默认500

HiveServer2支持通过HTTP发送Thrift RPC消息，适用于需要代理的情况，如负载均衡。
HiveServer2可以运行于TCP模式或HTTP模式二者之一。

HiveServer2 HTTP模式设置：

- `hive.server2.transport.mode`： 默认`binary`，即TCP模式，设置为`http`则使用HTTP模式
- `hive.server2.thrift.http.port`： 默认10001
- `hive.server2.thrift.http.max.worker.threads`： 默认500
- `hive.server2.thrift.http.min.worker.threads`： 默认5
- `hive.server2.thrift.http.path`： 默认cliservice，The service endpoint

HiveServer2默认以提交查询的用户身份执行查询，`hive.server2.enable.doAs`设置为`false`则使用运行hiveserver2进程的用户身份运行。

2.8 启动Beeline

```
$ $HIVE_HOME/bin/beeline
# 建立Hive连接，不需要输入密码
beeline> !connect jdbc:hive2://bdcourse-0001:10000 root
```

Beeline是一个JDBC客户端，基于SQLLine CLI，支持嵌入模式（embedded mode）和远程模式（remote mode）。嵌入模式下，启动Beeline会运行一个嵌入的Hive。远程模式下Beeline通过Thrift连接到独立的HiveServer2。

Beeline命令

- `!<SQLLine command>`：SQLLine命令，如`!quit`退出，`!list`（所有当前连接列表），`!close`（关闭当前连接），`!columns {table name}`（显示表的字段信息），`!connect {url} {username} {password} [driver class]`（连接到数据库），`!dbinfo`，`!describe [table name]`，`!history`（列出历史命令），`!indexes {table name}`，`!reconnect`

Beeline Hive命令

- `reset [<key>]`：重置配置（所有或<key>）为默认值
- `set <key>=<value>`：设置配置值
- `set`：列出配置项
- `set -v`：列出所有Hive和Hadoop配置项

- `dfs <dfs command>`：执行dfs命令
- `<query string>`：执行查询
- `add FILE[S] <filepath> <filepath>* add JAR[S] <filepath> <filepath>* add ARCHIVE[S] <filepath> <filepath>*`
- `add FILE[S] <ivyurl> <ivyurl>* add JAR[S] <ivyurl> <ivyurl>* add ARCHIVE[S] <ivyurl> <ivyurl>*`
- `list FILE[S] list JAR[S] list ARCHIVE[S]`
- `list FILE[S] <filepath>* list JAR[S] <filepath>* list ARCHIVE[S] <filepath>*`
- `delete FILE[S] <filepath>* delete JAR[S] <filepath>* delete ARCHIVE[S] <filepath>*`
- `reload`

Beeline命令行参数

- `-u <database URL>`：要连接的JDBC URL，例如 `beeline -u db_URL`
- `-r`：重连接
- `-n <username>`
- `-p <password>`
- `-d <driver class>`
- `-e <query>`
- `-f <file>`
- `--hiveconf property=value`
- `--color=[true/false]`
- `--hivevar name=valu`
- `--verbose=[true/false]`
- `--outputformat=[table/vertical/csv/tsv/dsv/csv2/tsv2]`

3. Hive语句示例

注意：这里是示例！！！

创建Hive表

```
CREATE TABLE pokes (foo INT, bar STRING);
# 包含foo，bar，ds三列，分区列是虚拟列
CREATE TABLE invites (foo INT, bar STRING) PARTITIONED BY (ds STRING);
```

查看表信息

```
SHOW TABLES;
SHOW TABLES '.*s';
DESCRIBE invites;
```

修改、删除表

```
ALTER TABLE events RENAME TO 3koobecaf;
ALTER TABLE pokes ADD COLUMNS (new_col INT);
ALTER TABLE invites ADD COLUMNS (new_col2 INT COMMENT 'a comment');
ALTER TABLE invites REPLACE COLUMNS (foo INT, bar STRING, baz INT COMMENT 'baz replaces
new_col2');
ALTER TABLE invites REPLACE COLUMNS (foo INT COMMENT 'only keep the first column');

DROP TABLE pokes;
```

从文件加载数据

```
# 如果不加LOCAL，则表明数据在HDFS上，下面的命令将把数据移动到Hive路径下
# hive.metastore.warehouse.dir配置项指定Hive的根目录
LOAD DATA LOCAL INPATH './examples/files/kv1.txt' OVERWRITE INTO TABLE pokes;

# 加载数据到指定的分区
LOAD DATA LOCAL INPATH './examples/files/kv2.txt' OVERWRITE INTO TABLE invites PARTITION
(ds='2008-08-15');
LOAD DATA LOCAL INPATH './examples/files/kv3.txt' OVERWRITE INTO TABLE invites PARTITION
(ds='2008-08-08');
```

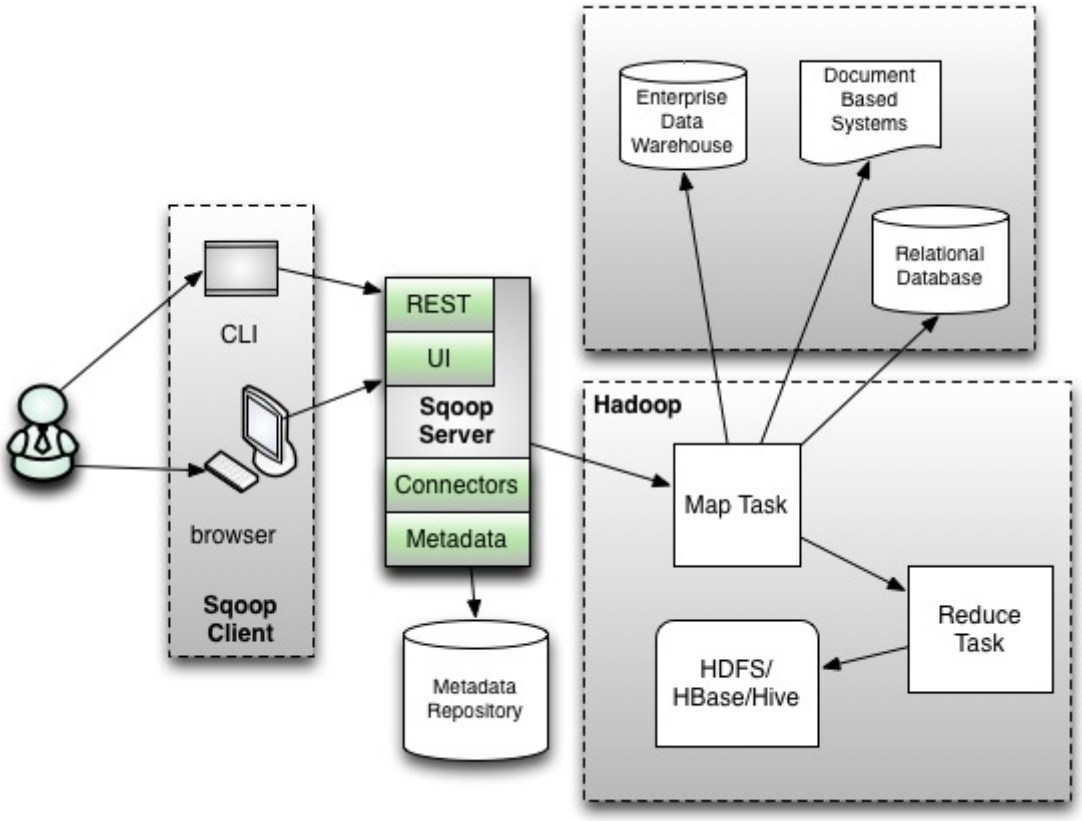
示例

```
SELECT a.foo FROM invites a WHERE a.ds='2008-08-15';
# 查询结果导入到HDFS或本地文件系统
INSERT OVERWRITE DIRECTORY '/tmp/hdfs_out' SELECT a.* FROM invites a WHERE a.ds='2008-08-
15';
INSERT OVERWRITE LOCAL DIRECTORY '/tmp/local_out' SELECT a.* FROM pokes a;

INSERT OVERWRITE TABLE events SELECT a.* FROM profiles a;
INSERT OVERWRITE TABLE events SELECT a.* FROM profiles a WHERE a.key < 100;
INSERT OVERWRITE LOCAL DIRECTORY '/tmp/reg_3' SELECT a.* FROM events a;
INSERT OVERWRITE DIRECTORY '/tmp/reg_4' SELECT a.invites, a.pokes FROM profiles a;
INSERT OVERWRITE DIRECTORY '/tmp/reg_5' SELECT COUNT(*) FROM invites a WHERE a.ds='2008-
08-15';
INSERT OVERWRITE DIRECTORY '/tmp/reg_5' SELECT a.foo, a.bar FROM invites a;
INSERT OVERWRITE LOCAL DIRECTORY '/tmp/sum' SELECT SUM(a.pc) FROM pc1 a;

# 聚合操作
FROM invites a INSERT OVERWRITE TABLE events SELECT a.bar, count(*) WHERE a.foo > 0 GROUP
BY a.bar;
INSERT OVERWRITE TABLE events SELECT a.bar, count(*) FROM invites a WHERE a.foo > 0 GROUP
BY a.bar;
# 连接
FROM pokes t1 JOIN invites t2 ON (t1.bar = t2.bar) INSERT OVERWRITE TABLE events SELECT
t1.bar, t1.foo, t2.foo;
# 多条插入语句
FROM src
INSERT OVERWRITE TABLE dest1 SELECT src.* WHERE src.key < 100
INSERT OVERWRITE TABLE dest2 SELECT src.key, src.value WHERE src.key >= 100 and src.key <
200
INSERT OVERWRITE TABLE dest3 PARTITION(ds='2008-04-08', hr='12') SELECT src.key WHERE
```

```
src.key >= 200 and src.key < 300
INSERT OVERWRITE LOCAL DIRECTORY '/tmp/dest4.out' SELECT src.value WHERE src.key >= 300;
```

4. Sqoop2安装配置

4.1 Sqoop2架构



参考：https://blogs.apache.org/sqoop/entry/apache_sqoop_highlights_of_sqoop

4.2 安装配置

Sqoop2的安装配置包含 `Server` 和 `Client` 两部分，这两部分包含在同一个二进制预编译文件 `sqoop-1.99.7-bin-hadoop200.tar.gz` 中。 `Server` 安装在集群中的**一个节点**上，作为Hadoop的客户端，因此Hadoop相关jar文件和配置文件必须在安装节点上，**但是该节点上不一定非得运行任何Hadoop相关的服务，Sqoop2可以安装在gateway节点上**； `Client` 可以安装在任意多的节点上。

```
$ wget http://mirrors.ustc.edu.cn/apache/sqoop/1.99.7/sqoop-1.99.7-bin-hadoop200.tar.gz
$ tar -xzvf sqoop-1.99.7-bin-hadoop200.tar.gz
$ mv sqoop-1.99.7-bin-hadoop200 /opt/
$ ln -s /opt/sqoop-1.99.7-bin-hadoop200 /opt/sqoop
```

Sqoop Server需要 `$HADOOP_COMMON_HOME` 、 `$HADOOP_HDFS_HOME` 、 `$HADOOP_MAPRED_HOME` 和 `$HADOOP_YARN_HOME` 等环境变量来查找Hadoop库文件，但是如果**只**设置了 `$HADOOP_HOME` 环境变量，Sqoop会自动从 `$HADOOP_HOME/share/hadoop/common` 、 `$HADOOP_HOME/share/hadoop/hdfs` 、 `$HADOOP_HOME/share/hadoop/mapreduce` 和 `$HADOOP_HOME/share/hadoop/yarn` 查找相应的jar文件。

```
$ sudo vi /etc/profile

export SQOOP_HOME=/opt/sqoop
# 在原PATH基础上添加$SQOOP_HOME/bin
export PATH=$SQOOP_HOME/bin:$PATH

$ source /etc/profile
```

Sqoop2预编译包里不包含JDBC Driver，需要下载并通过环境变量`SQOOP_SERVER_EXTRA_LIB`配置到Sqoop Server的classpath，或直接放到`$SQOOP_HOME/server/lib`中。

```
# 使用Hive安装时候下载的mysql-connector-java-8.0.19.jar
$ cp $HIVE_HOME/lib/mysql-connector-java-8.0.19.jar $SQOOP_HOME/server/lib/
```

Sqoop Server配置文件在`$SQOOP_HOME/conf/sqoop.properties`。

```
$ vi $SQOOP_HOME/conf/sqoop.properties
# 修改以下配置，其他保持默认
org.apache.sqoop.submission.engine.mapreduce.configuration.directory=/opt/hadoop/etc/hado
op
org.apache.sqoop.security.authentication.type=SIMPLE
org.apache.sqoop.security.authentication.handler=org.apache.sqoop.security.authentication
.SimpleAuthenticationHandler
org.apache.sqoop.security.authentication.anonymous=true
org.apache.sqoop.security.authentication.proxyuser.root.users=*
org.apache.sqoop.security.authentication.proxyuser.root.groups=*
org.apache.sqoop.security.authentication.proxyuser.root.hosts=*
org.apache.sqoop.security.authentication.default.user=root
```

4.3 初始化元数据库

```
# 初始化元数据库
$ sqoop2-tool upgrade
# 验证
$ sqoop2-tool verify
```

4.4 启动Sqoop Server

```
# 启动
$ sqoop2-server start
# 停止
# $ sqoop2-server stop
```

5. SegmentFault数据导入示例

5.0 创建HDFS目录

```
$ hadoop fs -mkdir /segmentfault
$ hadoop fs -mkdir /segmentfault/users
```

5.1 启动Sqoop Client

```
$ sqoop2-shell

sqoop:000> set server --host bdcourse-0001 --port 12000 --webapp sqoop
# 验证
sqoop:000> show version --all
```

5.2 创建Sqoop Link Object

```
# 列出已注册的connector
sqoop:000> show connector

# 创建JDBC Link，用于连接MySQL数据库，读取存储在其中的SegmentFault数据
sqoop:000> create link -connector generic-jdbc-connector
Creating link for connector with name generic-jdbc-connector
Please fill following values to create new link object
Name: sf-mysql

Database connection

Driver class: com.mysql.jdbc.Driver
Connection String: jdbc:mysql://bdcourse-0001:3306/segmentfault
Username: root
Password: ************
Fetch Size:
Connection Properties:
There are currently 0 values in the map:
entry#

SQL Dialect

Identifier enclose: `
New link was successfully created with validation status OK and name sf-mysql

# 查看创建的Link
sqoop:000> show link

# 创建HDFS Link，将MySQL中的数据抽取到HDFS中
sqoop:000> create link -connector hdfs-connector
Creating link for connector with name hdfs-connector
Please fill following values to create new link object
Name: sf-hdfs

HDFS cluster

URI: hdfs://bdcourse-0001:9000
```

```
Conf directory:
Additional configs::
There are currently 0 values in the map:
entry#
New link was successfully created with validation status OK and name sf-hdfs
```

5.3 创建Sqoop任务

```
# 查看已创建的Link对象
sqoop:000> show link
# 创建job对象
sqoop:000> create job -f "sf-mysql" -t "sf-hdfs"
Creating job for links with from name sf-mysql and to name sf-hdfs
Please fill following values to create new job object
Name: sf

Database source

Schema name: segmentfault
Table name: users
SQL statement:
Column names:
There are currently 0 values in the list:
element#
Partition column:
Partition column nullable:
Boundary query:

Incremental read

Check column:
Last value:

Target configuration

Override null value:
Null value:
File format:
  0 : TEXT_FILE
  1 : SEQUENCE_FILE
  2 : PARQUET_FILE
Choose: 0
Compression codec:
  0 : NONE
  1 : DEFAULT
  2 : DEFLATE
  3 : GZIP
  4 : BZIP2
  5 : LZO
  6 : LZ4
  7 : SNAPPY
  8 : CUSTOM
Choose: 0
Custom codec:
Output directory: /segmentfault/users
```

```
Append mode:

Throttling resources

Extractors:
Loaders:

Classpath configuration

Extra mapper jars:
There are currently 0 values in the list:
element#
New job was successfully created with validation status OK  and name sf
```

5.4 启动Sqoop任务

```
# 启动任务，加-s参数输出任务过程
sqoop:000> start job -name sf -s
# 以下为命令输出
Submission details
Job Name: sf
Server URL: http://192.168.1.31:12000/sqoop/
Created by: bdcourse
Creation date: 2020-04-27 12:37:20 UTC
Lastly updated by: bdcourse
External ID: job_1587947908605_0001
        http://18068-01:8088/proxy/application_1587947908605_0001/
2020-04-27 12:37:20 UTC: BOOTING  - Progress is not available
2020-04-27 12:37:47 UTC: RUNNING  - 0.00 %
2020-04-27 12:37:58 UTC: RUNNING  - 0.00 %
2020-04-27 12:38:08 UTC: RUNNING  - 0.00 %
2020-04-27 12:38:19 UTC: RUNNING  - 5.00 %
2020-04-27 12:38:30 UTC: RUNNING  - 15.00 %
2020-04-27 12:38:40 UTC: SUCCEEDED
Counters:
        org.apache.hadoop.mapreduce.FileSystemCounter
                FILE_LARGE_READ_OPS: 0
                FILE_WRITE_OPS: 0
                HDFS_READ_OPS: 10
                HDFS_BYTES_READ: 1535
                HDFS_LARGE_READ_OPS: 0
                FILE_READ_OPS: 0
                FILE_BYTES_WRITTEN: 2868990
                FILE_BYTES_READ: 0
                HDFS_WRITE_OPS: 10
                HDFS_BYTES_WRITTEN: 1181
        org.apache.hadoop.mapreduce.lib.output.FileOutputFormatCounter
                BYTES_WRITTEN: 0
        org.apache.hadoop.mapreduce.lib.input.FileInputFormatCounter
                BYTES_READ: 0
        org.apache.hadoop.mapreduce.JobCounter
                TOTAL_LAUNCHED_MAPS: 10
                MB_MILLIS_MAPS: 375837696
                SLOTS_MILLIS_REDUCES: 0
                VCORES_MILLIS_MAPS: 367029
```

```
                SLOTS_MILLIS_MAPS: 367029
                OTHER_LOCAL_MAPS: 10
                MILLIS_MAPS: 367029
        org.apache.sqoop.submission.counter.SqoopCounters
                ROWS_READ: 45
                ROWS_WRITTEN: 45
        org.apache.hadoop.mapreduce.TaskCounter
                SPILLED_RECORDS: 0
                MERGED_MAP_OUTPUTS: 0
                VIRTUAL_MEMORY_BYTES: 18913775616
                MAP_INPUT_RECORDS: 0
                SPLIT_RAW_BYTES: 1535
                MAP_OUTPUT_RECORDS: 45
                FAILED_SHUFFLE: 0
                PHYSICAL_MEMORY_BYTES: 2483531776
                GC_TIME_MILLIS: 17067
                CPU_MILLISECONDS: 44440
                COMMITTED_HEAP_BYTES: 1776848896
Job executed successfully

# 查看任务状态
sqoop:000> status job -n sf

# 退出Sqoop
sqoop:000> :exit
```

5.5 创建Hive表

```
# 打开Beeline
$ $HIVE_HOME/bin/beeline
# 建立Hive连接，不需要输入密码
beeline> !connect jdbc:hive2://bdcourse-0001:10000 root

# 创建数据表，外部表
CREATE EXTERNAL TABLE sfusers(
    `username`    STRING,
    `nickname`    STRING
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LOCATION '/segmentfault/users';

SELECT * FROM sfusers;

# 查看已创建的表
DESCRIBE sfusers;

# 退出Beeline
!quit
```

6. 开源软件社区数据分析

6.1 数据导入HDFS

```
$ hadoop fs -mkdir /ghtorrentsmall
$ hadoop fs -mkdir /ghtorrentsmall/users
$ hadoop fs -mkdir /ghtorrentsmall/projects
$ hadoop fs -mkdir /ghtorrentsmall/commits
$ hadoop fs -mkdir /ghtorrentsmall/issues
$ hadoop fs -mkdir /ghtorrentsmall/watchers
$ hadoop fs -mkdir /ghtorrentsmall/followers

$ cd ~/ghtorrentsmall
$ hadoop fs -put users.csv /ghtorrentsmall/users
$ hadoop fs -put projects.csv /ghtorrentsmall/projects
$ hadoop fs -put commits.csv /ghtorrentsmall/commits
$ hadoop fs -put issues.csv /ghtorrentsmall/issues
$ hadoop fs -put watchers.csv /ghtorrentsmall/watchers
$ hadoop fs -put followers.csv /ghtorrentsmall/followers
```

6.2 创建Hive表

```
# 打开Beeline
$ $HIVE_HOME/bin/beeline
# 建立Hive连接，不需要输入密码
beeline> !connect jdbc:hive2://bdcourse-0001:10000 root

CREATE EXTERNAL TABLE users(
    id           INT,
    login        STRING,
    company      STRING,
    created_at   STRING,
    type         STRING,
    fake         INT,
    deleted      INT,
    long         FLOAT,
    lat          FLOAT,
    country_code STRING,
    state        STRING,
    city         STRING,
    location     STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES
(
    "separatorChar" = ",",
    "quoteChar"     = "\""
)
LOCATION '/ghtorrentsmall/users';


CREATE EXTERNAL TABLE projects(
    id          INT,
    url         STRING,
    owner_id    INT,
    name        STRING,
    description STRING,
    language    STRING,
```

```
    created_at  STRING,
    forked_from STRING,
    deleted     INT,
    updated_at  STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES
(
    "separatorChar" = ",",
    "quoteChar"     = "\""
)
LOCATION '/ghtorrentsmall/projects';


CREATE EXTERNAL TABLE commits(
    id         INT,
    project_id INT,
    created_at STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES
(
    "separatorChar" = ",",
    "quoteChar"     = "\""
)
LOCATION '/ghtorrentsmall/commits';


CREATE EXTERNAL TABLE issues(
    id          INT,
    reporter_id INT,
    repo_id     INT,
    created_at  STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES
(
    "separatorChar" = ",",
    "quoteChar"     = "\""
)
LOCATION '/ghtorrentsmall/issues';


CREATE EXTERNAL TABLE watchers(
    user_id INT,
    repo_id INT
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES
(
    "separatorChar" = ",",
    "quoteChar"     = "\""
)
LOCATION '/ghtorrentsmall/watchers';

CREATE EXTERNAL TABLE followers(
    follower_id INT,
    user_id     INT,
```

```
    created_at  STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES
(
    "separatorChar" = ",",
    "quoteChar"     = "\""
)
LOCATION '/ghtorrentsmall/followers';
```

6.3 数据分析

示例1：融合GitHub和SegmentFault用户数据（**现有数据匹配不到**）

```
CREATE TABLE gs_users(
    g_id           INT,
    g_login        STRING,
    g_country_code STRING,
    s_username     STRING,
    s_nickname     STRING,
    s_rank         INT,
    s_following    INT,
    s_followed     INT,
    s_answers      INT,
    s_questions    INT,
    s_articles     INT,
    s_up           INT,
    s_certificate  INT
);

INSERT OVERWRITE TABLE gs_users SELECT `id`, `login`, `country_code`,
`username`, `nickname`, `rank`, `following`, `followed`, `answers`,
`questions`, `articles`, `up`, `certificate`
FROM users LEFT JOIN sfusers ON users.login=replace(sfusers.github, "https://github.com",
"");
```

示例2：统计开发者地区分布

```
CREATE TABLE userarea_distributions(
    country_code STRING,
    cnt INT
);

INSERT OVERWRITE TABLE userarea_distributions
SELECT country_code, COUNT(id) AS cnt FROM users GROUP BY country_code;
```

示例3：统计用户项目数

```
SELECT owner_id, COUNT(id) AS project_cnt
FROM projects
```

```
GROUP BY owner_id
ORDER BY project_cnt DESC;
```