HiveClient

This page describes the different clients supported by Hive. The command line client currently only supports an embedded server. The JDBC and Thrift-Java clients support both embedded and standalone servers. Clients in other languages only support standalone servers.

For details about the standalone server see Hive Server or HiveServer2.

Command Line

Operates in embedded mode only, that is, it needs to have access to the Hive libraries. For more details see Getting Started and Hive CLI.

JDBC

This document describes the JDBC client for the original Hive Server (sometimes called *Thrift server* or *HiveServer1*). For information about the HiveServer2 JDBC client, see JDBC in the HiveServer2 Clients document. HiveServer2 use is recommended; the original HiveServer has several concurrency issues and lacks several features available in HiveServer2.

> **Version information**
>
> The original Hive Server was removed from Hive releases starting in version 1.0.0. See HIVE-6977.

For embedded mode, uri is just "jdbc:hive://". For standalone server, uri is "jdbc:hive://host:port/dbname" where host and port are determined by where the Hive server is run. For example, "jdbc:hive://localhost:10000/default". Currently, the only dbname supported is "default".

JDBC Client Sample Code

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

public class HiveJdbcClient {
  private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";

  public static void main(String[] args) throws SQLException {
    try {
      Class.forName(driverName);
    } catch (ClassNotFoundException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
      System.exit(1);
    }
    Connection con = DriverManager.getConnection("jdbc:hive://localhost:10000/default", "", "");
    Statement stmt = con.createStatement();
    String tableName = "testHiveDriverTable";
    stmt.executeQuery("drop table " + tableName);
    ResultSet res = stmt.executeQuery("create table " + tableName + " (key int, value string)");
    // show tables
    String sql = "show tables '" + tableName + "'";
    System.out.println("Running: " + sql);
    res = stmt.executeQuery(sql);
    if (res.next()) {
      System.out.println(res.getString(1));
    }
    // describe table
    sql = "describe " + tableName;
    System.out.println("Running: " + sql);
    res = stmt.executeQuery(sql);
    while (res.next()) {
      System.out.println(res.getString(1) + "\t" + res.getString(2));
    }

    // load data into table
    // NOTE: filepath has to be local to the hive server
    // NOTE: /tmp/a.txt is a ctrl-A separated file with two fields per line
    String filepath = "/tmp/a.txt";
    sql = "load data local inpath '" + filepath + "' into table " + tableName;
    System.out.println("Running: " + sql);
    res = stmt.executeQuery(sql);

    // select * query
    sql = "select * from " + tableName;
    System.out.println("Running: " + sql);
    res = stmt.executeQuery(sql);
    while (res.next()) {
      System.out.println(String.valueOf(res.getInt(1)) + "\t" + res.getString(2));
    }

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

Running the JDBC Sample Code

```
# Then on the command-line
$ javac HiveJdbcClient.java

# 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

$ java -cp $CLASSPATH HiveJdbcClient

# Alternatively, you can run the following bash script, which will seed the data file
# and build your classpath before invoking the client.

#!/bin/bash
HADOOP_HOME=/your/path/to/hadoop
HIVE_HOME=/your/path/to/hive

echo -e '1\x01foo' > /tmp/a.txt
echo -e '2\x01bar' >> /tmp/a.txt

HADOOP_CORE={{ls $HADOOP_HOME/hadoop-*-core.jar}}
CLASSPATH=.:$HADOOP_CORE:$HIVE_HOME/conf

for i in ${HIVE_HOME}/lib/*.jar ; do
    CLASSPATH=$CLASSPATH:$i
done

java -cp $CLASSPATH HiveJdbcClient
```

JDBC Client Setup for a Secure Cluster

To configure Hive on a secure cluster, add the directory containing hive-site.xml to the CLASSPATH of the JDBC client.

Python

Operates only on a standalone server. Set (and export) PYTHONPATH to build/dist/lib/py.

The python modules imported in the code below are generated by building hive.

Please note that the generated python module names have changed in hive trunk.

```
#!/usr/bin/env python

import sys

from hive import ThriftHive
from hive.ttypes import HiveServerException
from thrift import Thrift
from thrift.transport import TSocket
from thrift.transport import TTransport
from thrift.protocol import TBinaryProtocol

try:
    transport = TSocket.TSocket('localhost', 10000)
    transport = TTransport.TBufferedTransport(transport)
    protocol = TBinaryProtocol.TBinaryProtocol(transport)

    client = ThriftHive.Client(protocol)
    transport.open()

    client.execute("CREATE TABLE r(a STRING, b INT, c DOUBLE)")
    client.execute("LOAD TABLE LOCAL INPATH '/path' INTO TABLE r")
    client.execute("SELECT * FROM r")
    while (1):
      row = client.fetchOne()
      if (row == None):
        break
      print row
    client.execute("SELECT * FROM r")
    print client.fetchAll()

    transport.close()

except Thrift.TException, tx:
    print '%s' % (tx.message)
```

PHP

Operates only on a standalone server.

```
<?php
// set THRIFT_ROOT to php directory of the hive distribution
$GLOBALS['THRIFT_ROOT'] = '/lib/php/';
// load the required files for connecting to Hive
require_once $GLOBALS['THRIFT_ROOT'] . 'packages/hive_service/ThriftHive.php';
require_once $GLOBALS['THRIFT_ROOT'] . 'transport/TSocket.php';
require_once $GLOBALS['THRIFT_ROOT'] . 'protocol/TBinaryProtocol.php';
// Set up the transport/protocol/client
$transport = new TSocket('localhost', 10000);
$protocol = new TBinaryProtocol($transport);
$client = new ThriftHiveClient($protocol);
$transport->open();

// run queries, metadata calls etc
$client->execute('SELECT * from src');
var_dump($client->fetchAll());
$transport->close();
```

ODBC

Operates only on a standalone server. The Hive ODBC client provides a set of C-compatible library functions to interact with Hive Server in a pattern similar to those dictated by the ODBC specification. See Hive ODBC Driver.

Thrift

Thrift Java Client

Operates both in embedded mode and on standalone server.

Thrift C++ Client

Operates only on a standalone server. In the works.

Thrift Node Clients

Thrift Node clients are available on github at https://github.com/wdavidw/node-thrift-hive and https://github.com/forward/node-hive.

Thrift Ruby Client

A Thrift Ruby client is available on github at https://github.com/forward3d/rbhive.