**Hive**

## **一、基础知识**

**1 Hive是一个数据仓库基础工具在Hadoop中用来处理结构化数据。**

它架构在Hadoop之上，总归为大数据，并使得查询和分析方便。并提供简单的sql查询功能，可以将sql语句转换为MapReduce任务进行运行。术语“大数据”是大型数据集，其中包括体积庞大，高速，以及各种由与日俱增的数据的集合。使用传统的数据管理系统，它是难以加工大型数据。因此，Apache软件基金会推出了一款名为Hadoop的解决大数据管理和处理难题的框架。

**Hive是一个数据仓库基础工具在Hadoop中用来处理结构化数据。**它架最初，Hive是由Facebook开发，后来由Apache软件基金会开发，并作为进一步将它作为名义下Apache Hive为一个开源项目。它用在好多不同的公司。例如，亚马逊使用它在 Amazon Elastic MapReduce。

**2 Hadoop是一个开源框架来存储和处理大型数据在分布式环境中。**

它包含两个模块，一个是MapReduce，另外一个是Hadoop分布式文件系统（HDFS）。

（1）MapReduce：它是一种并行编程模型在大型集群普通硬件可用于处理大型结构化，半结构化和非结构化数据。

（2）HDFS：Hadoop分布式文件系统是Hadoop的框架的一部分，用于存储和处理数据集。它提供了一个容错文件系统在普通硬件上运行。

**3 Hadoop生态系统包含了用于协助Hadoop的不同的子项目（工具）模块**

Sqoop: 它是用来在HDFS和RDBMS之间来回导入和导出数据。

Pig: 它是用于开发MapReduce操作的脚本程序语言的平台。

Hive: 它是用来开发SQL类型脚本用于做MapReduce操作的平台。

**4 有多种方法来执行MapReduce作业：**

传统的方法是使用Java MapReduce程序结构化，半结构化和非结构化数据。

针对MapReduce的脚本的方式，使用Pig来处理结构化和半结构化数据。

Hive查询语言（HiveQL或HQL）采用Hive为MapReduce的处理结构化数据。

**5 Hive 不是一个关系数据库**

不是一个设计用于联机事务处理（OLTP）

不是实时查询和行级更新的语言

**6 Hiver特点**

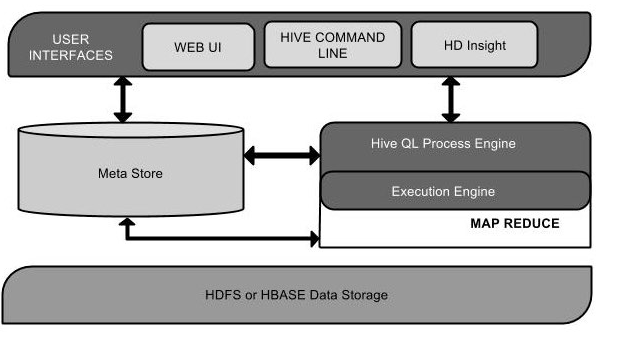
它存储架构在一个数据库中并处理数据到HDFS。

它是专为OLAP设计。

它提供SQL类型语言查询叫HiveQL或HQL。

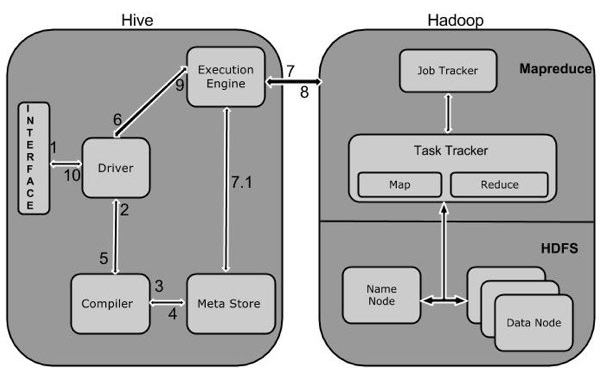
它是熟知，快速，可扩展和可扩展的。

**7 Hive架构**



|  |  |
| --- | --- |
| **单元名称** | **操作** |
| 用户接口/界面 | Hive是一个数据仓库基础工具软件，可以创建用户和HDFS之间互动。用户界面，Hive支持是Hive的Web UI，Hive命令行，HiveHD洞察（在Windows服务器）。 |
| 元存储 | Hive选择各自的数据库服务器，用以储存表，数据库，列模式或元数据表，它们的数据类型和HDFS映射。 |
| HiveQL处理引擎 | HiveQL类似于SQL的查询上Metastore模式信息。这是传统的方式进行MapReduce程序的替代品之一。相反，使用Java编写的MapReduce程序，可以编写为MapReduce工作，并处理它的查询。 |
| 执行引擎 | HiveQL处理引擎和MapReduce的结合部分是由Hive执行引擎。执行引擎处理查询并产生结果和MapReduce的结果一样。它采用MapReduce方法。 |
| HDFS 或 HBASE | Hadoop的分布式文件系统或者HBASE数据存储技术是用于将数据存储到文件系统。 |

**8 Hive工作原理**



|  |  |
| --- | --- |
| **Step No.** | **操作** |
| 1 | Execute Query  Hive接口，如命令行或Web UI发送查询驱动程序（任何数据库驱动程序，如JDBC，ODBC等）来执行。 |
| 2 | Get Plan  在驱动程序帮助下查询编译器，分析查询检查语法和查询计划或查询的要求。 |
| 3 | Get Metadata  编译器发送元数据请求到Metastore（任何数据库）。 |
| 4 | Send Metadata  Metastore发送元数据，以编译器的响应。 |
| 5 | Send Plan  编译器检查要求，并重新发送计划给驱动程序。到此为止，查询解析和编译完成。 |
| 6 | Execute Plan  驱动程序发送的执行计划到执行引擎。 |
| 7 | Execute Job  在内部，执行作业的过程是一个MapReduce工作。执行引擎发送作业给JobTracker，在名称节点并把它分配作业到TaskTracker，这是在数据节点。在这里，查询执行MapReduce工作。 |
| 7.1 | Metadata Ops  与此同时，在执行时，执行引擎可以通过Metastore执行元数据操作。 |
| 8 | Fetch Result  执行引擎接收来自数据节点的结果。 |
| 9 | Send Results  执行引擎发送这些结果值给驱动程序。 |
| 10 | Send Results  驱动程序将结果发送给Hive接口。 |

**9 hbase和hive的对比**

1）、hbase与hive都是架构在hadoop之上的，都是用hadoop作为底层存储。

2）、**Hive是建立在Hadoop之上为了减少MapReduce jobs编写工作的批处理系统，HBase是为了支持弥补Hadoop对实时操作的缺陷的项目 。**

3）、在操作RMDB数据库，如果是全表扫描，就用Hive+Hadoop，如果是索引访问，就用HBase+Hadoop 。

4）、Hive query就是MapReduce jobs可以从5分钟到数小时不止，HBase是非常高效的，肯定比Hive高效的多。

5）、Hive本身不存储和计算数据，它完全依赖于HDFS和MapReduce，Hive中的表纯逻辑。

6）、hive借用hadoop的MapReduce来完成一些hive中的命令的执行。

7）、hbase是物理表，不是逻辑表，提供一个超大的内存hash表，搜索引擎通过它来存储索引，方便查询操作。

8）、hbase是列存储。

9）、hdfs作为底层存储，hdfs是存放文件的系统，而Hbase负责组织文件。

10）、hive需要用到hdfs存储文件，需要用到MapReduce计算框架。

**10 Hadoop生态上几个技术的关系与区别：hive、pig、hbase 关系与区别**

**（1）Pig**

一种操作hadoop的轻量级脚本语言，最初又雅虎公司推出，不过现在正在走下坡路了。当初雅虎自己慢慢退出pig的维护之后将它开源贡献到开源社区由所有爱好者来维护。不过现在还是有些公司在用，不过我认为与其使用pig不如使用hive。：）

Pig是一种数据流语言，用来快速轻松的处理巨大的数据。

Pig包含两个部分：Pig Interface,Pig Latin。

Pig可以非常方便的处理HDFS和HBase的数据，和Hive一样,Pig可以非常高效的处理其需要做的，通过直接操作Pig查询可以节省大量的劳动和时间。当你想在你的数据上做一些转换，并且不想编写MapReduce jobs就可以用Pig.

**（2）Hive**

不想用程序语言开发MapReduce的朋友比如DB们，熟悉SQL的朋友可以使用Hive开离线的进行数据处理与分析工作。注意Hive现在适合在离线下进行数据的操作，就是说不适合在挂在真实的生产环境中进行实时的在线查询或操作，因为一个字“慢”。相反起源于FaceBook,Hive在Hadoop中扮演数据仓库的角色。建立在Hadoop集群的最顶层，对存储在Hadoop群上的数据提供类SQL的接口进行操作。你可以用 HiveQL进行select,join,等等操作。如果你有数据仓库的需求并且你擅长写SQL并且不想写MapReduce jobs就可以用Hive代替。

**（3）HBase**

HBase作为面向列的数据库运行在HDFS之上，HDFS缺乏随即读写操作，HBase正是为此而出现。HBase以Google BigTable为蓝本，以键值对的形式存储。项目的目标就是快速在主机内数十亿行数据中定位所需的数据并访问它。

HBase是一个数据库，一个NoSql的数据库，像其他数据库一样提供随即读写功能，Hadoop不能满足实时需要，HBase正可以满足。如果你需要实时访问一些数据，就把它存入HBase。

你可以用Hadoop作为静态数据仓库，HBase作为数据存储，放那些进行一些操作会改变的数据。

**（4）Pig VS Hive**

Hive更适合于数据仓库的任务，Hive主要用于静态的结构以及需要经常分析的工作。Hive与SQL相似促使 其成为Hadoop与其他BI工具结合的理想交集。Pig赋予开发人员在大数据集领域更多的灵活性，并允许开发简洁的脚本用于转换数据流以便嵌入到较大的 应用程序。Pig相比Hive相对轻量，它主要的优势是相比于直接使用Hadoop Java APIs可大幅削减代码量。正因为如此，Pig仍然是吸引大量的软件开发人员。Hive和Pig都可以与HBase组合使用，Hive和Pig还为HBase提供了高层语言支持，使得在HBase上进行数据统计处理变的非常简单

**（5）Hive VS HBase**

Hive是建立在Hadoop之上为了减少MapReduce jobs编写工作的批处理系统，HBase是为了支持弥补Hadoop对实时操作的缺陷的项目 。想象你在操作RMDB数据库，如果是全表扫描，就用Hive+Hadoop,如果是索引访问，就用HBase+Hadoop 。Hive query就是MapReduce jobs可以从5分钟到数小时不止，HBase是非常高效的，肯定比Hive高效的多。

## 二、数据类型

**1 整型**

整型数据可以指定使用整型数据类型，INT。当数据范围超过INT的范围，需要使用BIGINT，如果数据范围比INT小，使用SMALLINT。 TINYINT比SMALLINT小。

下表描述了各种INT数据类型：



**2 字符串类型**

字符串类型的数据类型可以使用单引号('')或双引号(“”)来指定。它包含两个数据类型：VARCHAR和CHAR。Hive遵循C-类型的转义字符。

下表描述了各种CHAR数据类型：

数据类型 长度

VARCHAR 1 to 65355

CHAR 255

**3 时间戳**

它支持传统的UNIX时间戳可选纳秒的精度。它支持的java.sql.Timestamp格式“YYYY-MM-DD HH:MM:SS.fffffffff”和格式“YYYY-MM-DD HH:MM:ss.ffffffffff”。

日期

DATE值在年/月/日的格式形式描述 {{YYYY-MM-DD}}.

**4 小数点**

在Hive 小数类型与Java大十进制格式相同。它是用于表示不可改变任意精度。语法和示例如下：

DECIMAL(precision, scale)

decimal(10,0)

**5 联合类型**

联合是异类的数据类型的集合。可以使用联合创建的一个实例。语法和示例如下：

UNIONTYPE<int, double, array<string>, struct<a:int,b:string>>

{0:1}

{1:2.0}

{2:["three","four"]}

{3:{"a":5,"b":"five"}}

{2:["six","seven"]}

{3:{"a":8,"b":"eight"}}

{0:9}

{1:10.0}

**6 浮点类型**

浮点类型是只不过有小数点的数字。通常，这种类型的数据组成DOUBLE数据类型。

**7 十进制类型**

十进制数据类型是只不过浮点值范围比DOUBLE数据类型更大。十进制类型的范围大约是 -10-308 到 10308.

**8 Null 值**

缺少值通过特殊值 - NULL表示。

**9 复杂类型**

Hive复杂数据类型如下：

**10 数组**

在Hive 数组与在Java中使用的方法相同。

Syntax: ARRAY<data\_type>

**11 映射**

映射在Hive类似于Java的映射。

Syntax: MAP<primitive\_type, data\_type>

**12 结构体**

在Hive结构体类似于使用复杂的数据。

Syntax: STRUCT<col\_name : data\_type [COMMENT col\_comment], ...>

## 三、Hive操作

**1 创建数据库**

（1）创建数据库是用来创建数据库在Hive中语句。在Hive数据库是一个命名空间或表的集合。此语法声明如下：

CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>

在这里，IF NOT EXISTS是一个可选子句，通知用户已经存在相同名称的数据库。可以使用SCHEMA 在DATABASE的这个命令。下面的查询执行创建一个名为userdb数据库：

hive> CREATE DATABASE [IF NOT EXISTS] userdb;

或

hive> CREATE SCHEMA userdb;

（2）下面的查询用于验证数据库列表：

hive> SHOW DATABASES;

default

userdb

（3）JDBC 程序

在JDBC程序来创建数据库如下。

import java.sql.SQLException;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.Statement;

import java.sql.DriverManager;

public class HiveCreateDb {

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

public static void main(String[] args) throws SQLException {

// Register driver and create driver instance

Class.forName(driverName);

// get connection

Connection con = DriverManager.getConnection("jdbc:hive://localhost:10000/default", "", "");

Statement stmt = con.createStatement();

stmt.executeQuery("CREATE DATABASE userdb");

System.out.println(“Database userdb created successfully.”);

con.close();

}

}

保存程序在一个名为HiveCreateDb.java文件。下面的命令用于编译和执行这个程序。

$ javac HiveCreateDb.java

$ java HiveCreateDb

输出：

Database userdb created successfully.

**2 删除数据库**

（1）下面的查询用于删除数据库。假设要删除的数据库名称为userdb。

hive> DROP DATABASE IF EXISTS userdb;

（2）以下是使用CASCADE查询删除数据库。这意味着要全部删除相应的表在删除数据库之前。

hive> DROP DATABASE IF EXISTS userdb CASCADE;

（3）以下使用SCHEMA查询删除数据库。

hive> DROP SCHEMA userdb;

**3 创建表**

1. 下面的查询创建使用上述数据的表名为 employee。



hive> CREATE TABLE IF NOT EXISTS employee ( eid int, name String,

> salary String, destination String)

> COMMENT ‘Employee details’

> ROW FORMAT DELIMITED

> FIELDS TERMINATED BY ‘\t’

> LINES TERMINATED BY ‘\n’

> STORED AS TEXTFILE;

（2）LOAD DATA 语句

我们将插入下列数据到表中。在/home/user目录中名为sample.txt的文件。

1201 Gopal 45000 Technical manager

1202 Manisha 45000 Proof reader

1203 Masthanvali 40000 Technical writer

1204 Kiran 40000 Hr Admin

1205 Kranthi 30000 Op Admin

下面的查询加载给定文本插入表中。

hive> LOAD DATA LOCAL INPATH '/home/user/sample.txt'

> OVERWRITE INTO TABLE employee;

下载成功完成，能看到以下回应：

OK

Time taken: 15.905 seconds

hive>

**4 修改表**

ALTER TABLE name RENAME TO new\_name

ALTER TABLE name ADD COLUMNS (col\_spec[, col\_spec ...])

ALTER TABLE name DROP [COLUMN] column\_name

ALTER TABLE name CHANGE column\_name new\_name new\_type

ALTER TABLE name REPLACE COLUMNS (col\_spec[, col\_spec ...])

（1）下面的查询增加了一个列名dept在employee表。

hive> ALTER TABLE employee ADD COLUMNS (

> dept STRING COMMENT 'Department name');

（2）以下从employee表中查询删除的所有列，并使用emp替换列

hive> ALTER TABLE employee REPLACE COLUMNS (

> eid INT empid Int,

> ename STRING name String);

（3）下面的查询增加了一个列名dept在employee表。

hive> ALTER TABLE employee ADD COLUMNS (

> dept STRING COMMENT 'Department name');

**5 删除表**

hive> DROP TABLE IF EXISTS employee;

**6 Hive分区**

Hive组织表到分区。它是将一个表到基于分区列，如日期，城市和部门的值相关方式。使用分区，很容易对数据进行部分查询。

表或分区是细分成桶，以提供额外的结构，可以使用更高效的查询的数据。桶的工作是基于表的一些列的散列函数值。

例如，一个名为Tab1表包含雇员数据，如 id, name, dept 和yoj (即加盟年份)。假设需要检索所有在2012年加入，查询搜索整个表所需的信息员工的详细信息。但是，如果用年份分区雇员数据并将其存储在一个单独的文件，它减少了查询处理时间。下面的示例演示如何分区的文件和数据：

（1）下面文件包含employee 数据表。

/tab1/employeedata/file1

id, name, dept, yoj

1, gopal, TP, 2012

2, kiran, HR, 2012

3, kaleel,SC, 2013

4, Prasanth, SC, 2013

（2）上面的数据被划分成使用年两个文件。

/tab1/employeedata/2012/file2

1, gopal, TP, 2012

2, kiran, HR, 2012

/tab1/employeedata/2013/file3

3, kaleel,SC, 2013

4, Prasanth, SC, 2013

（3）添加分区

可以通过添加分区表改变所述表。假设我们有一个表叫employee ，拥有如 Id, Name, Salary, Designation, Dept, 和 yoj等字段。

语法：

ALTER TABLE table\_name ADD [IF NOT EXISTS] PARTITION partition\_spec

[LOCATION 'location1'] partition\_spec [LOCATION 'location2'] ...;

partition\_spec:

: (p\_column = p\_col\_value, p\_column = p\_col\_value, ...)

（4）以下查询用于将分区添加到employee表。

hive> ALTER TABLE employee

> ADD PARTITION (year=’2013’)

> location '/2012/part2012';

（5）重命名分区

此命令的语法如下。

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

以下查询用来命名一个分区：

hive> ALTER TABLE employee PARTITION (year=’1203’)

> RENAME TO PARTITION (Yoj=’1203’);

（6）删除分区

下面语法用于删除分区：

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

以下查询是用来删除分区：

hive> ALTER TABLE employee DROP [IF EXISTS]

> PARTITION (year=’1203’);

**7 创建一个视图**

可以创建一个视图，在执行SELECT语句的时候。语法如下：

CREATE VIEW [IF NOT EXISTS] view\_name [(column\_name [COMMENT column\_comment], ...) ]

[COMMENT table\_comment]

AS SELECT ...

示例

举个例子来看。假设employee表拥有如下字段：Id, Name, Salary, Designation 和 Dept。生成一个查询检索工资超过30000卢比的员工详细信息，我们把结果存储在一个名为视图 emp\_30000.

+------+--------------+-------------+-------------------+--------+

| ID | Name | Salary | Designation | Dept |

+------+--------------+-------------+-------------------+--------+

|1201 | Gopal | 45000 | Technical manager | TP |

|1202 | Manisha | 45000 | Proofreader | PR |

|1203 | Masthanvali | 40000 | Technical writer | TP |

|1204 | Krian | 40000 | Hr Admin | HR |

|1205 | Kranthi | 30000 | Op Admin | Admin |

+------+--------------+-------------+-------------------+--------+

下面使上述业务情景查询检索员的工详细信息：

hive> CREATE VIEW emp\_30000 AS

> SELECT \* FROM employee

> WHERE salary>30000;

**8 删除一个视图**

使用下面的语法来删除视图：

DROP VIEW view\_name

下面的查询删除一个名为emp\_30000的视图：

hive> DROP VIEW emp\_30000;

**9 创建索引**

索引也不过是一个表上的一个特定列的指针。创建索引意味着创建一个表上的一个特定列的指针。它的语法如下：

CREATE INDEX index\_name

ON TABLE base\_table\_name (col\_name, ...)

AS 'index.handler.class.name'

[WITH DEFERRED REBUILD]

[IDXPROPERTIES (property\_name=property\_value, ...)]

[IN TABLE index\_table\_name]

[PARTITIONED BY (col\_name, ...)]

[

[ ROW FORMAT ...] STORED AS ...

| STORED BY ...

]

[LOCATION hdfs\_path]

[TBLPROPERTIES (...)]

例子

让我们举个索引例子。使用之前的字段 Id, Name, Salary, Designation, 和 Dept创建一个名为index\_salary的索引，对employee 表的salary列索引。

下面的查询创建一个索引：

hive> CREATE INDEX inedx\_salary ON TABLE employee(salary)

> AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler';

这是一个指向salary列。如果列被修改，变更使用的索引值存储。

**10 删除索引**

下面的语法用来删除索引：

DROP INDEX <index\_name> ON <table\_name>

下面的查询删除名为index\_salary索引：

hive> DROP INDEX index\_salary ON employee;

## 四、内置函数

|  |  |  |
| --- | --- | --- |
| **返回类型** | **签名** | **描述** |
| BIGINT | round(double a) | 返回BIGINT最近的double值。 |
| BIGINT | floor(double a) | 返回最大BIGINT值等于或小于double。 |
| BIGINT | ceil(double a) | 它返回最小BIGINT值等于或大于double。 |
| double | rand(), rand(int seed) | 它返回一个随机数，从行改变到行。 |
| string | concat(string A, string B,...) | 它返回从A后串联B产生的字符串 |
| string | substr(string A, int start) | 它返回一个起始，从起始位置的子字符串，直到A.结束 |
| string | substr(string A, int start, int length) | 返回从给定长度的起始start位置开始的字符串。 |
| string | upper(string A) | 它返回从转换的所有字符为大写产生的字符串。 |
|  |  |  |
| string | ucase(string A) | 和上面的一样 |
| string | lower(string A) | 它返回转换B的所有字符为小写产生的字符串。 |
| string | lcase(string A) | 和上面的一样 |
| string | trim(string A) | 它返回字符串从A.两端修剪空格的结果 |
| string | ltrim(string A) | 它返回A从一开始修整空格产生的字符串(左手侧) |
| string | rtrim(string A) | rtrim(string A)，它返回A从结束修整空格产生的字符串(右侧) |
| string | regexp\_replace(string A, string B, string C) | 它返回从替换所有子在B结果配合C.在Java正则表达式语法的字符串 |
| int | size(Map<K.V>) | 它返回在映射类型的元素的数量。 |
| int | size(Array<T>) | 它返回在数组类型元素的数量。 |
| value of <type> | cast(<expr> as <type>) | 它把表达式的结果expr<类型>如cast('1'作为BIGINT)代表整体转换为字符串'1'。如果转换不成功，返回的是NULL。 |
| string | from\_unixtime(int unixtime) | 转换的秒数从Unix纪元(1970-01-0100:00:00 UTC)代表那一刻，在当前系统时区的时间戳字符的串格式："1970-01-01 00:00:00" |
| string | to\_date(string timestamp) | 返回一个字符串时间戳的日期部分：to\_date("1970-01-01 00:00:00") = "1970-01-01" |
| int | year(string date) | 返回年份部分的日期或时间戳字符串：year("1970-01-01 00:00:00") = 1970, year("1970-01-01") = 1970 |
| int | month(string date) | 返回日期或时间戳记字符串月份部分：month("1970-11-01 00:00:00") = 11, month("1970-11-01") = 11 |
| int | day(string date) | 返回日期或时间戳记字符串当天部分：day("1970-11-01 00:00:00") = 1, day("1970-11-01") = 1 |
| string | get\_json\_object(string json\_string, string path) | 提取从基于指定的JSON路径的JSON字符串JSON对象，并返回提取的JSON字符串的JSON对象。如果输入的JSON字符串无效，返回NULL。 |

## 五 HiveQL 查询

**1 SELECT ... FROM 语句**

hive> SELECT name,salary FROM employees; --普通查询

hive>SELECT e.name, e.salary FROM employees e; --也支持别名查询

hive> SELECT name,subordinates FROM employees; 显示 John Doe ["Mary Smith","Todd Jones"] 数组类型的显示

hive>SELECT name,deductions FROM employees; 显示 John Doe {"Federal Taxes":0.2,"State Taxes":0.05} MAP 输出

hive>SELECT name,adress FROM employees; 显示 John Doe {"street":"1 Michigan Ave.","city":"Chicago","state":"IL"} address 列是一个 STRUCT

hive> SELECT name,subordinates[0] FROM employees; 查看数组中的第1个元素,如果不存在元素将返回 NULL

hive>SELECT name,deductions["State Taxes"] FROM employees; 查询MAP 元素

hive>SELECT name,adress.city FROM employees; 查询STRUCT 中的一个元素，可以用 . 符号

以上三种查询 在 where 子句中同样可以使用这些方式；

hive>SELECT symbol, `price.\*` FROM stocks; 用正则表达戒严 选择我们想要的列，本句是查 symbol 列和所有列名以 price 作为前缀的列；

hive>SELECT upper(name), salary, deductions["Federal Taxes"], round(salary \* (1 - deductions["Federal Taxes"])) FROM employees; 使用 round() 方法返回一个 Double 类型的最近整数。

Hive 中所支持的运算符：

A + B 、A - B、A \* B、A / B、A % B [求余]、A & B [按位与]、A | B [按位或]、A ^ B [按位取异或]、~A [按位取反]

hive> SET hive.map.aggr=true; --设置这个属性为 true 来提高聚合的性能

hive>SELECT count(\*), avg(salary) FROM employees; --这个设置会触发在 map 阶段进行的“顶级”聚合过程，（非顶级聚合过程将会在执行一个 GROUP BY 后进行），不过这个设置将需要更多的内存。

hive>SELECT count(DISTINCT symbol) FROM stocks; 多个函数还可以接受像 DISTINCT 这个表达式，来进行排重；v如果 symbol 是分区列时会返回 0.。。。是个bug；

hive>SELECT count(DISTINCT ymd), count(DISTINCT volume) FROM stocks; 官方不允许这样查，但实际可以这样查；

表生成函数：与聚合函数相反的一类函数不是所谓的表生成函数，基可以将单列扩展成多列或者多行;

hive> SELECT explode (subordinates) AS sub FROM employees; 本语句将 employees 表中每行记录中 subordinates 字段内容转换成 0 个或者多个新的记录行,如果 subordinates 字段内容为空的话，那么将不会产生新的记录，如果不为空的话，那么这个数组的每个元素都将产生一行新记录；AS sub 子句定义了列别名 sub。当全用表生成函数时，Hive 要求使用列别名。【具体在13章中会详细介绍】

hive> SELECT upper(name), salary, deductions["Federal Taxes"], round(salary \*(1 - deductions["Federal Taxes"])) FROM employees LIMIT 2;

LIMIT 子句用于限制返回的行数；

hive> SELECT upper(name), salary, deductions["Federal Taxes"] as fed\_taxes, round(salary \*(1 - deductions["Federal Taxes"])) as salary\_minus\_fed FROM employees LIMIT 2; fed\_taxes、 salary\_minus\_fed 给新查出的结果的两个列起个别名；

hive> FROM (upper(name), salary, deductions["Federal Taxes"] as fed\_taxes, round(salary \*(1 - deductions["Federal Taxes"])) as salary\_minus\_fed FROM employees) e SELECT e.name, e.salary\_minus\_fed\_taxes WHERE e.salary\_minus\_fed\_taxes >7000; SELECT 的嵌套查询

CASE ... WHEN ... THEN 句式例子：

hive>SELECT name,salary, CASE

WHEN salary < 5000 THEN 'low'

WHEN salary >= 5000 AND salary <7000 THEN 'middle'

WHEN salary>=7000 AND salary < 100000 THEN 'high'

ELSE 'very high' END AS bracket FROM employees;

Hive 大多数情况下查询都会触发一个 MapReduce 任务，Hive 中本模式的查询可以不必使用 MP；如： select \* from employees;

SELECT \* from employees WHERE country='us' and state='ca' limit 100; 对于 WHERE 语句中过滤条件只是分区字段这种情况（无论是否使用了 LIMT语句限制输出记录条数）也是无需MapReduce 过程的；

hive.exec.mode.local.auto=true; 如果这个值为 true Hive 还会尝试使用本地模式执行其他的操作,否则 Hive 会用 MP 来执行其他所有的查询；最好将它增加到 $HOME/.hiverc 文件中；

**2 where 语句用于过滤查询条件 用法与 普通 SQL 一样；**

谓词操作：这些词的操作同样可以用于 JION ... ON 和 HAVING 语句中

【图片取自《hive编程指南》 88-92页】

LIke 和 RLIKE ：LIKE是一个标准的 SQL 操作，可以让我们通过字符串的开头或结尾，以及指定特定的子字符串，或者当子字符串出现在字符串内的任何位置时进行匹配；【RLIKE 子句是 Hive中这个功能的一个扩展，其可以通过 JAVA 的正则表达式这个更强大的语言来指定匹配条件。】

hive>SELECT name, address.street FROM employees WHERE address.street LIKE '%Ave.' --查找以Ave 开头的 雇员姓名；

hive>SELECT name, address.stree FROM employees WHERE address.street RLIKE '.\*(Chicago|Ontario).\*'; --RLIKE 后面字符串含义：字符串中的 . 表示和任意的字符匹配，星号 \* 表示重复“左边的字符串”零次到无数次，表达式（x|y）表示 和 x 或者 y 匹配；【PS：不会正则的可以百度一下去学学】

**3 GROUP BY 语句，它通常会用 聚合函数一起使用，按照一个或者多个列对结果进行分组，然后对每个组执行聚合操作【用法与SQL差不多】**

hive>SELECT year(ymd),avg(price\_close) FROM stocks WHERE exchange='NASDAQ' AND symbol ='APPLE' GROUP BY year(ymd); --示例

HAVING 语句：

hive>SELECT year(ymd),avg(price\_close) FROM stocks

>WHERE exchange='NASDAQ' AND symbol ='APPLE' GROUP BY year(ymd)

>HAVING avg(price\_close)>50.0; --示例

**4 JOIN 语句**

INNER JOIN 内链接，只有进行链接的两个表中都存在与连接标准相关匹配的数据才会显示【用法与SQL差不多】

hive>SELECT a.ymd, a.price\_close, b.price\_close FROM stocks a JOIN STOCKS b ON a.ymd =b.ymd

>WHERE a.symbol ='appl' AND b.symbol='ibm';

注意：hive 中不支持的查询如下【同进也不支持在ON的子句中的谓词间使用 OR，可以支持 AND 】：

hive>SELECT a.ymd, a.price\_close, b.price\_close FROM stocks a JOIN STOCKS b ON a.ymd<=b.ymd >WHERE a.symbol ='appl' AND b.symbol='ibm';

多张表的链接：

hive>SELECT a.ymd, a.price\_close, b.price\_close,c.price\_close

> FROM stocks a JOIN stocks b ON a.ymd=b.ymd

> JOIN stocks c ON a.ymd = c.ymd

>WHERE a.symbol ='appl' AND b.symbol='ibm' AND c.symbol='ge';

大多数情况下，Hive 会对每个 JOIN 链接对象启动一个 MapReduce 任务，上面例子中会首先启动一个 MapReduce job 对表 a 和表 b 进行连接操作，然后会再启动一个 MapReduce job 将第一个 MapReduce job 的输出和和 c 进行连接操作；【hive 都是从左向右运顺序执行的】

【提示】

对于3个或者更多表进行 JOIN 链接时，如果每个 ON 子句都使用相同的链接键的话，那么只会产生一个 MapReduce；Hive 同时假定查询中最后一个表是最大的那个表，在对每行记录进行链接操时，它会尝将其他表缓存起来，然后扫描最后那个大表进计算，因此我们需要保证连续查询中的表的大小从左到右是依次增加的；

LEFT OUTER JOIN 左外链接【与SQL用法类似】

hive> SELECT s.ymd, s.symbol, s.price\_close, d.dividend FROM stocks s LEFT OUTER JION dividends d ON

>s.ydm AND s.symbol=d.symbol WHERE s.symbol='aapl' ;

在左外链接操作中， JOIN 操作符左边表中符合 WHERE 子句的所有记录将会被返回，右边表中没有符合 ON 后面的链接记录时，会返回 null;

【提示】WHERE 语句在连接操作执行后才会执行，因此 WHERE 语句应该只用于过滤那些非null值的列，同时，ON 语句中的分区过滤条件外链接（OUTER JOIN）中是无效的，不过在内链接中是有效的；

RIGHT OUTER JOIN 右外链接：会返回右边表所有符合 WHERE 语句的记录，左表中没有匹配的字段值用 NULL 代替；

FULL OUTER JOIN 完全链接：将会返回所有表中符合 where 语句条件的所有记录；

LEFT SEMI JOIN 左半开链接：会返回左边表的记录，前提是其他记录对于右边表满足ON语句中的判断条件;【不支持右关开链接】

hive>SELECT s.ymd, s.symbol, s.price\_close FROM stocks s LEFT SEMI JOIN dividends d ON s.ymd=d.ymd AND s.symbol=d.symbol;

Hive 不支持的查询：

hive>SELECT s.ymd, s.symbol, s.price\_close FROM stocks s WHERE s.ymd,s.symbol IN(SELECT \* FROM dividends d);

JOIN 笛卡尔积：左边链接的行数乘以右边表的行数等于返回结果集的大小；

hive> SELECT \* FROM stocks JOIN dividends;

如果使用此方法查询，MapReduce 任何方式都无法法进行优化；

set hive.mapred.mode=strict 会禁止 笛卡尔积的查询；

map-side JOIN：如果所有表中只有一张表是小表，那么可以在最大的表通过 mapper 的时候将小表完全放到内存中，Hive可以在 map 端执行链接过程（称为 map-side JOIN）; 因为 Hive 可以和内存中的小表进行逐一匹配，从而省略掉常规连接操作所需要的 reduce 过程 ，即使对于很小的数据庥，这个优化也明显要快于常规的连接操作，不仅减少了 reduce 过程 ，而且有时还可以同时减少 map 过程的执行步骤；

hive>set hive.auto.conver.JOIN=true; 从0.7版本开始需要设置此属性才可以生效

hive>set hive.mapjoin.smalltable.filesize=25000000; 配置能够使用这个优化的小表的大小；（单位：字节）

右外链接、全外链接不支持上面的优化；

分桶表，对于大表，在特定的情况下也可以使用这个优化，但表中的数据必须是按照ON 语句中的键进行分桶的，而且其中一张表的分桶的个数必须是另一张表分桶个数的若干倍，这样才可以按照分桶数据进行链接；hive>set hive.optimize.bucketmapJOIN=true; 也需要设置，默认是关闭的；

**5 ORDER BY 和 SORT BY**

Hive 的 ORDER BY 语句和其他的 SQL 语言的定义都是一样。会对查询结果一个全局排序。也就是说会有一个所有的数据都通过一个 reducer 进行处理的过程，如果有大数据集，过程可能会消耗太过漫长的时间来执行；

Hive 还有一种排序 SORT BY ，其会在每个 reducer 中对数据进行排序，也就是会执行一个局部排序过程 ，这可以保证每个 reducer 的输出数据都是有序的（但不是全局有序）这样可以提高全局排序的效率；

SELECT s.ymd, s.symbol, s.price\_close FROM stocks s ORDER BY s.ymd ASC , s.symbol DESC;

SELECT s.ymd, s.symbol, s.price\_close FROM stocks s SORT BY s.ymd ASC , s.symbol DESC;

注意：因为 ORDER BY 操作可能会导致运行时间过长，如果属性 hvie.mapred.mode=strict 的话，那么 hive 要求这样的语句必须加 LIMIT 语句进行限制，默认情况下 属性是 nonstrict；

**6 SORT BY 与 DISTRIBUTE BY**

distribute by 控制 map 的输出在 reducer 中是如何划分的。

假设我们希望具有相同股票交易码的数据在一起处理，那么我们可以使用 distribute by 来保证具有相同股票交易码的记录会分发到同一个 reducer 中进行处理，然后使用 SOTR BY 来按照我们的期望对数据进行排序：

hive>SELECT s.ymd, s.symbol, s.price\_close FROM stocks s DISTRIBUTE BY s.symbol SORT BY s.symbol ASC, s.ymd ASC;

DISTRIBUTE BY 和 GROUP BY 在其控制着 reducer 是如何接受一行行数据进行处理这方面类似类的，而 SORT BY 则控制着 reducer 内的数据是如何进行排序的。需要注意的是 DISTRIBUTE BY 语句一定要写在 SOTR BY 语句之前；

**7 CLUSTER BY**

在上面的例子中 s.symbol 列被用在 DISTRIBUTE BY 语句中，而 s.symbol 和 s.ymd 位用 SOTR BY 语句中，如果这两个语句中涉及到的完全相同的列，而且采用的是升序的排序方式（也就是默认的排序方式）在这种情况下 cluster by 就等于前面的2个语句 ，相当于简写：

hive> SELECT s.ymd, s.symbol, s.price\_close FROM stock s CLUSTER BY s.symbol;

使用 DISTRIBUTE BY ... SOTR BY 语句或其简化版的 CLUSTER BY 语句会剥夺 SORT BY 的并行性，然而这样可以实现输出文件的数据是全局排序的；

**8 类型转换**

SELECT name, salary FROM employees WHERE cast(salary AS FLOAT )< 10000.0 ; 类型转换的语法： cast(value AS TYPE) 如果不成功则返回 NULL；

需要注意的是 将浮点数转换成整数的推荐方式是使用 round() 或者 floor() 函数，而不是类型转换操作符 cast；

binary 类型只支持 BINARY 类型转换为 STRING 类型；

**9 抽样查询：**

Hive 可以通过对表进行分桶抽样来满足抽样查询

hive>SELECT \* FOM numbers TABLESAMPLE (BUCKET 3 OUT OF 10 ON rand()) s; 分桶语句中的分母表示的是数据将会被散列的桶的个数，而分子表示将会选择的桶的个数；

hive>SELECT \* FROM numberflat TABLESAMPLE(0.1 PERCENT) s; 按照百分比方式进行抽样，这种是基于行数的；

注意：这种抽样试不一定适用于所遥 文件格式，另外这种抽样方式的最小抽样单元是一个 HDFS 数据块，如果表的数据大小小于普通的块大小的 128M 那么会返回 所有行；

基于百分比的抽样提供了一个变量用于控制基于数据块的调优的种子信息：

<property>

<name>hive.sample.seednumber</name>

<value>0</value>

</property>

**10 UNION ALL**

可以将2 个或多个表进行合并，每个 union 子查询必须都需要有相同的列，而且对应的每个字段的字段类型必须是一致的；这个功能便于将一个长的复杂的 WHERE 语句分割成 2 个或多个 union 子查询，除非源表建立了索，否则 这个查询将会对同一份源数据进行多次拷贝分发；

FROM (FROM src SELECT src.key, src,value WHERE src.key <100

UNION ALL

FROM src SELECT src.key, src,value WHERE src.key >110

) unioninput

INSERT OVERWRITE DIRECTORY '/tmp/union.out' SELECT unionimput.\*

## Hive原理

**1 Hive的工作原理简单来说就是一个查询引擎**

**2 接收到一个sql,后面做的事情包括**

**1）.词法分析/语法分析**

使用antlr将SQL语句解析成抽象语法树-AST

**2）.语义分析**

从Megastore获取模式信息，验证SQL语句中队表名,列名，以及数据类型的检查和隐式转换，以及Hive提供的函数和用户自定义的函数（UDF/UAF）

**3）.逻辑计划生产**

生成逻辑计划-算子树

**4）.逻辑计划优化**

对算子树进行优化，包括列剪枝，分区剪枝，谓词下推等

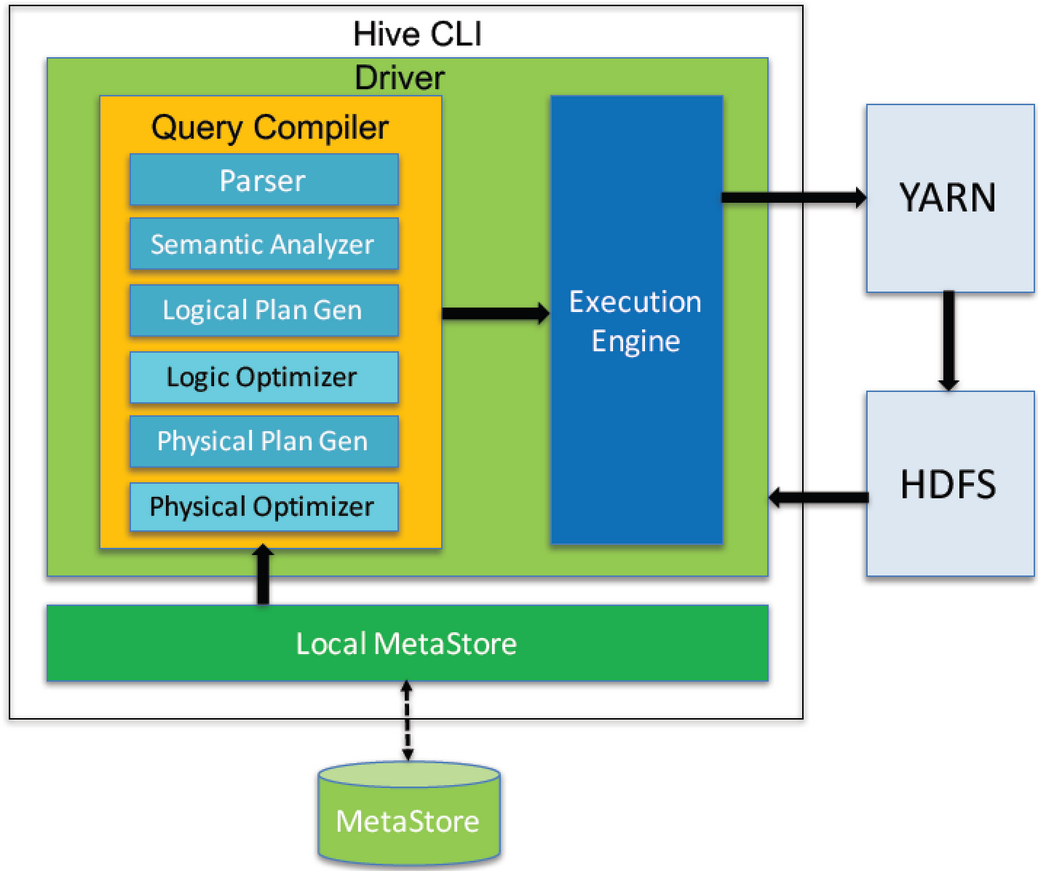
**5）.物理计划生成**

将逻辑计划生产包含由MapReduce任务组成的DAG的物理计划

**6）.物理计划执行**

将DAG发送到Hadoop集群进行执行

**7）.将查询结果返回**



# **3 使用了hive为什么要加mysql**

metastore是hive元数据的集中存放地，默认使用内嵌的derby数据库作为储引擎。

Derby引擎的缺点：**一次只能打开一个会话，Mysql作为外置存储引擎多用户同时访问。**

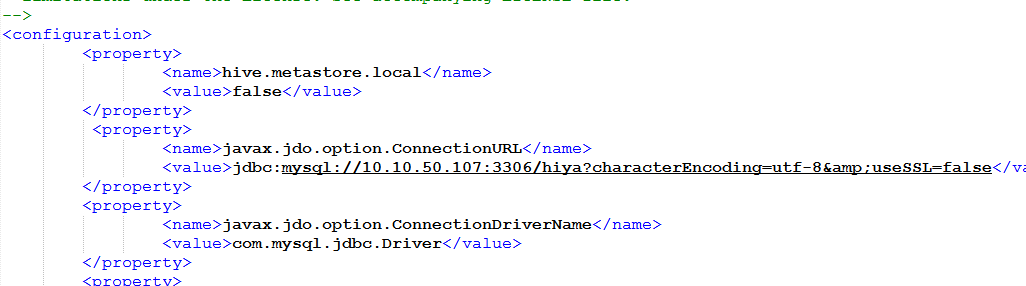
hive只是个工具，包括它的数据分析，依赖于mapreduce，它的数据管理，依赖于外部系统。这一步其实不是必须的，因为Hive默认的metadata（元数据）是存储在Derby里面的，但是有一个弊端就是同一时间只能有一个Hive实例访问，这适合做开发程序时做本地测试。为此，需公用的，mysql。这也是为什么，在安装hive时，也需要配置mysql了。

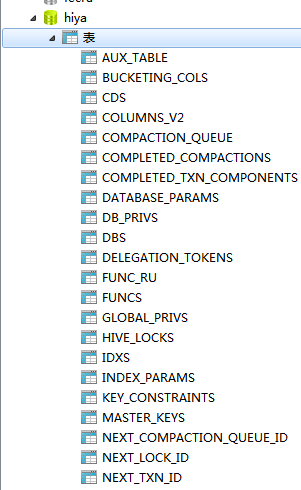
**4 Mysql作为Hive metaStore的存储数据库。里面大约有20张库表。**

其中主要涉及到的表如下

|  |  |  |
| --- | --- | --- |
| **表名** | **说明** | **关联键** |
| **TBLS** | 所有hive表的基本信息(表名，创建时间，所属者等) | TBL\_ID,SD\_ID |
| **TABLE\_PARAM** | 表级属性，（如是否外部表，表注释，最后修改时间等） | TBL\_ID |
| **COLUMNS** | Hive表字段信息(字段注释，字段名，字段类型，字段序号) | SD\_ID |
| **SDS** | 所有hive表、表分区所对应的hdfs数据目录和数据格式 | SD\_ID,SERDE\_ID |
| **SERDE\_PARAM** | 序列化反序列化信息，如行分隔符、列分隔符、NULL的表示字符等 | SERDE\_ID |
| **PARTITIONS** | Hive表分区信息（所属表，分区值） | PART\_ID,SD\_ID,TBL\_ID |
| **PARTITION\_KEYS** | Hive分区表分区键（即分区字段） | TBL\_ID |
| **PARTITION\_KEY\_VALS** | Hive表分区名(键值) | PART\_ID |

select table\_name , CREATE\_TIME from information\_schema.`TABLES` where table\_schema = 'erp' order by CREATE\_TIME desc





**5 Hive:用Java代码通过JDBC连接Hiveserver**

**（1）hive依赖hadoop**，将hdfs当作文件存储介质，那是否意味着hive需要知道namenode的地址？

实际上在hive的hive-env.sh 中配置了 HADOOP\_HOME=/home/install/hadoop-2.5.1

**（2）hive的本地模式和远程模式有什么区别？**

hive本质上是将sql语法解析为mapreduce的过程，既然如此它就必须提交mapreduce任务到resoucemanager，那么它如何提交？就是通过hadoop提供的命令hadoop jar命令来提交。

本地模式：简单的理解，hive客户端仅供本地使用，直接使用hive命令，不需要指定IP 端口

远程模式：简单的理解，将hive发布成一个服务进程，通过hive --service hiveserver命令，那么其他hive客户端就可以连接hive的服务进程

其他客户端可以是jdbc方式、hive提供的beeline命令等，既然要连接远端的hive服务进程，那自然需要指定 IP 端口，这里的IP指的是hive服务进程所在的IP，端口自然也是，也自然与hadoop无关。所以不要混淆

HiveServer2提供了一个新的命令行工具Beeline，它是基于SQLLine CLI的JDBC客户端。

Beeline工作模式有两种，即本地嵌入模式和远程模式。嵌入模式情况下，它返回一个嵌入式的Hive（类似于Hive CLI）。而远程模式则是通过Thrift协议与某个单独的HiveServer2进程进行连接通信。

**（3）hive的三种连接方式**

@hive 命令行模式，直接输入/hive/bin/hive的执行程序，或者输入 hive --service cli

用于linux平台命令行查询，查询语句基本跟mysql查询语句类似

@hive web界面的 (端口号9999) 启动方式

hive –service hwi &

用于通过浏览器来访问hive，感觉没多大用途

@hive 远程服务 (端口号10000) 启动方式

hive --service hiveserver &

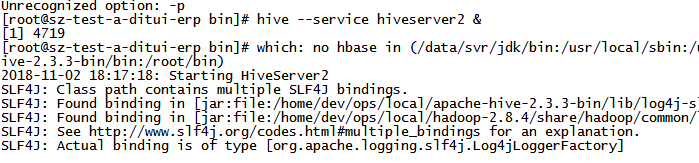
**（&表示后台运行）**

java client方式连接

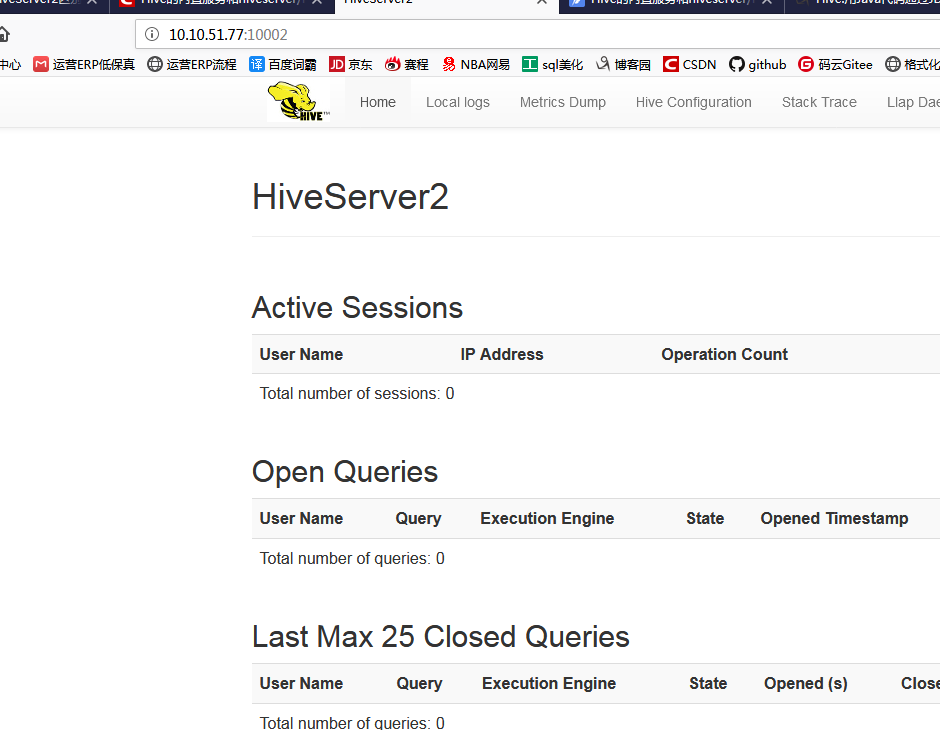
备注：连接Hive JDBC URL：jdbc:hive://192.168.6.116:10000/default （Hive默认端口：10000 默认数据库名：default）

**（4）开启hive 远程服务**

bin**/ hive --service hiveserver2 &**



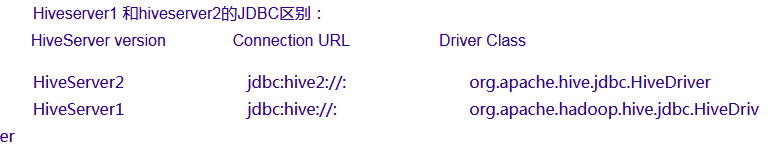
[http://10.10.51.77:10002/](http://192.168.48.130:10002/)



**（5）添加依赖**

**（6）代码**

**6 hiveServer/HiveServer2区别**

****

两者都允许远程客户端使用多种编程语言，通过HiveServer或者HiveServer2，客户端可以在不启动CLI的情况下对Hive中的数据进行操作，连这个和都允许远程客户端使用多种编程语言如java，[python](http://lib.csdn.net/base/python" \o "Python知识库" \t "https://blog.csdn.net/u013061459/article/details/_blank)等向hive提交请求，取回结果（从hive0.15起就不再支持hiveserver了），但是在这里我们还是要说一下hiveserver。

HiveServer或者HiveServer2都是基于Thrift的，但HiveSever有时被称为Thrift server，而HiveServer2却不会。既然已经存在HiveServer，为什么还需要HiveServer2呢？**这是因为HiveServer不能处理多于一个客户端的并发请求，这是由于HiveServer使用的Thrift接口所导致的限制，不能通过修改HiveServer的代码修正。因此在Hive-0.11.0版本中重写了HiveServer代码得到了HiveServer2，进而解决了该问题。HiveServer2支持多客户端的并发和认证，为开放API客户端如JDBC、ODBC提供更好的支持。**