# 初始Hive

**第一部分：Hive简介**

什么是Hive

•Hive是基于Hadoop的一个数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供类SQL查询功能。

•本质是将SQL转换为MapReduce程序

**第二部分：为什么使用Hive**

面临的问题

  人员学习成本太高

  项目周期要求太短

  我只是需要一个简单的环境

  MapReduce  如何搞定

  复杂查询好难

  Join如何实现

为什么要使用Hive

•操作接口采用类SQL语法，提供快速开发的能力

•避免了去写MapReduce，减少开发人员的学习成本

•扩展功能很方便

**Hive的特点**

•可扩展

Hive可以自由的扩展集群的规模，一般情况下不需要重启服务

•延展性

Hive支持用户自定义函数，用户可以根据自己的需求来实现自己的函数

•容错

良好的容错性，节点出现问题SQL仍可完成执行

**第三部分：Hive与Hadoop的关系**

**第四部分：Hive与传统数据库对比**

|  |  |  |
| --- | --- | --- |
|  | **Hive** | **RDBMS** |
| 查询语言 | HQL | SQL |
| 数据存储 | HDFS | Raw Device or Local FS |
| 执行 | MapReduce | Excutor |
| 执行延迟 | 高 | 低 |
| 处理数据规模 | 大 | 小 |
| 索引 | 0.8版本后加入位图索引 | 有复杂的索引 |

**第五部分：Hive的历史**

•由FaceBook 实现并开源

•2011年3月，0.7.0版本 发布，此版本为重大升级版本，增加了简单索引，HAING等众多高级特性

•2011年06月，0.7.1 版本发布，修复了一些BUG，如在Windows上使用JDBC的的问题

• 2011年12月，0.8.0版本发布，此版本为重大升级版本，增加了insert into 、HA等众多高级特性

•2012年2月5日，0.8.1版本发布，修复了一些BUG，如 使 Hive 可以同时运行在 Hadoop0.20.x 与 0.23.0

•2012年4月30日，0.9.0版本发布，重大改进版本，增加了对Hadoop 1.0.0的支持、实现BETWEEN等特性

**第六部分：Hive的未来发展**

•增加更多类似传统数据库的功能，如存储过程

•提高转换成的MapReduce性能

•拥有真正的数据仓库的能力

•UI部分加强

转载请注明出处【 <http://sishuok.com/forum/blogPost/list/6220.html>】

# Hive安装与配置

**第一部分：软件准备与环境规划**

**Hadoop环境介绍**

•Hadoop安装路径

•/home/test/Desktop/hadoop-1.0.0/

•Hadoop 元数据存放目录

•/home/test/data/core/namenode

•Hadoop 数据存放路径

•/home/test/data/core/datanode

Hive环境规划

•Hive安装路径

•/home/test/Desktop/

•Hive数据存放路径

•hdfs

•/user/hive/warehouse

•Hive元数据

•第三方数据库

•derby mysql

**软件准备**

•OS

•ubuntu

•JDK

•java 1.6.0\_27

•Hadoop

•hadoop-1.0.0.tar

•Hive

•hive-0.8.1.tar

**第二部分：Hive项目介绍**

**项目结构**

**Hive配置文件介绍**

•hive-site.xml      hive的配置文件

•hive-env.sh        hive的运行环境文件

•hive-default.xml.template  默认模板

•hive-env.sh.template     hive-env.sh默认配置

•hive-exec-log4j.properties.template   exec默认配置

• hive-log4j.properties.template log默认配置

**hive-site.xml**

< property>

  <name>javax.jdo.option.ConnectionURL</name> <value>jdbc:mysql://localhost:3306/hive?createData baseIfNotExist=true</value>

  <description>JDBC connect string for a JDBC metastore</description>

</property>

<property>

<name>javax.jdo.option.ConnectionDriverName</name>

    <value>com.mysql.jdbc.Driver</value>

  <description>Driver class name for a JDBC metastore</description>

</property>

<property>

<name>javax.jdo.option.ConnectionUserName</name>

    <value>root</value>

   <description>username to use against metastore database</description>

</property>

<property>

<name>javax.jdo.option.ConnectionPassword</name>

   <value>test</value>

   <description>password to use against metastore database</description>

</property>

**hive-env.sh**

•配置Hive的配置文件路径

•export HIVE\_CONF\_DIR= your path

•配置Hadoop的安装路径

•HADOOP\_HOME=your hadoop home

**第三部分：使用Derby数据库的安装方式**

**什么是Derby安装方式**

•Apache Derby是一个完全用java编写的数据库，所以可以跨平台，但需要在JVM中运行

•Derby是一个Open source的产品，基于Apache License 2.0分发

•即将元数据存储在Derby数据库中，也是Hive默认的安装方式

**安装Hive**

•解压Hive

•tar zxvf  hive-0.8.1.tar  /home/test/Desktop

•建立软连接

•ln –s hive-0.8.1 hive

•添加环境变量

•export HIVE\_HOME=/home/test/Desktop/hive

•export PATH=….HIVE\_HOME/bin:$PATH:.

**配置Hive**

•进入hive/conf目录

•依据hive-env.sh.template，创建hive-env.sh文件

•cp  hive-env.sh.template hive-env.sh

•修改hive-env.sh

•指定hive配置文件的路径

•export HIVE\_CONF\_DIR=/home/test/Desktop/hive/conf

•指定Hadoop路径

• HADOOP\_HOME=/home/test/Desktop/hadoop

**hive-site.xml**

<property>

  <name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:derby:;databaseName=metastore\_db;create=true</value>

  <description>JDBC connect string for a JDBC metastore</description>

</property>

<property>

  <name>javax.jdo.option.ConnectionDriverName</name>

  <value>org.apache.derby.jdbc.EmbeddedDriver</value>

  <description>Driver class name for a JDBC metastore</description>

</property>

<property>

  <name>javax.jdo.option.ConnectionUserName</name>

  <value>APP</value>

  <description>username to use against metastore database</description>

</property>

<property>

  <name>javax.jdo.option.ConnectionPassword</name>

  <value>mine</value>

  <description>password to use against metastore database</description>

</property>

**启动Hive**

•命令行键入

•Hive

•显示

WARNING: org.apache.hadoop.metrics.jvm.EventCounter is deprecated. Please use org.apache.hadoop.log.metrics.EventCounter in all the log4j.properties files.

Logging initialized using configuration in jar:file:/home/test/Desktop/hive-0.8.1/lib/hive-common-0.8.1.jar!/hive-log4j.properties

Hive history file=/tmp/test/hive\_job\_log\_test\_201208260529\_167273830.txt

hive>

**测试语句**

•建立测试表test

•create table test  (key string);

•show tables;

**第四部分：使用MySQL数据库的安装方式**

**安装MySQL**

•Ubuntu 采用apt-get安装

•sudo apt-get install mysql-server

•建立数据库hive

•create database hive

•创建hive用户,并授权

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

•flush privileges;

**安装Hive**

•解压Hive

•tar zxvf  hive-0.8.1.tar  /home/test/Desktop

•建立软连接

•ln –s hive-0.8.1 hive

•添加环境变量

•export HIVE\_HOME=/home/test/Desktop/hive

•export PATH=….HIVE\_HOME/bin:$PATH:.

**修改hive-site.xml**

**<property>**

**<name>**javax.jdo.option.ConnectionURL **</name>**

**<value>**jdbc:mysql://localhost:3306/hive **</value>**

**</property>**

**<property>**

**<name>**javax.jdo.option.ConnectionDriverName **</name>**

**<value>**com.mysql.jdbc.Driver **</value>**

**</property>**

**<property>**

**<name>**javax.jdo.option.ConnectionPassword **</name>**

**<value>**hive **</value>**

**</property>**

**<property>**

**<name>**hive.hwi.listen.port **</name>**

**<value>**9999 **</value>**

**<description>**This is the port the Hive Web Interface will listen on **</descript** **ion>**

**</property>**

**<property>**

**<name>**datanucleus.autoCreateSchema **</name>**

**<value>**false **</value>**

**</property>**

**<property>**

**<name>**datanucleus.fixedDatastore **</name>**

**<value>**true **</value>**

**</property>**

**<property>**

**<name>**hive.metastore.local **</name>**

**<value>**true **</value>**

**<description>**controls whether to connect to remove metastore server or open a new metastore server in Hive Client JVM **</description>**

**</property>**

**启动Hive**

•命令行键入

•Hive

•显示

WARNING: org.apache.hadoop.metrics.jvm.EventCounter is deprecated. Please use org.apache.hadoop.log.metrics.EventCounter in all the log4j.properties files.

Logging initialized using configuration in jar:file:/home/test/Desktop/hive-0.8.1/lib/hive-common-0.8.1.jar!/hive-log4j.properties

Hive history file=/tmp/test/hive\_job\_log\_test\_201208260529\_167273830.txt

hive>

**测试语句**

•建立测试表test

•create table test (key string);

•show tables;

# Hive 内建操作符与函数开发

## 第一部分：关系运算

**Hive支持的关系运算符**

•常见的关系运算符

•等值比较: =

•不等值比较: <>

•小于比较: <

•小于等于比较: <=

•大于比较: >

•大于等于比较: >=

•空值判断: IS NULL

•非空判断: IS NOT NULL

•LIKE比较: LIKE

•JAVA的LIKE操作: RLIKE

•REGEXP操作: REGEXP

•等值比较: =

   语法：A=B

操作类型：所有基本类型

描述: 如果表达式A与表达式B相等，则为TRUE；否则为FALSE

举例：

hive> select 1 from dual where 1=1;

•不等值比较: <>

语法: A <> B

操作类型: 所有基本类型

描述: 如果表达式A为NULL，或者表达式B为NULL，返回NULL；如果表达式A与表达式B不相等，则为TRUE；否则为FALSE

举例：

hive> select 1 from dual where 1 <> 2;

•小于比较: <

语法: A < B

操作类型: 所有基本类型

描述: 如果表达式A为NULL，或者表达式B为NULL，返回NULL；如果表达式A小于表达式B，则为TRUE；否则为FALSE

举例：

hive> select 1 from dual where 1 < 2;

•小于等于比较: <=

语法: A <= B

操作类型: 所有基本类型

描述: 如果表达式A为NULL，或者表达式B为NULL，返回NULL；如果表达式A小于或者等于表达式B，则为TRUE；否则为FALSE

举例：

hive> select 1 from dual where 1 <= 1;

•大于等于比较: >=

语法: A >= B

操作类型: 所有基本类型

描述: 如果表达式A为NULL，或者表达式B为NULL，返回NULL；如果表达式A大于或者等于表达式B，则为TRUE；否则为FALSE

举例：

hive> select 1 from dual where 1 >= 1;

•空值判断: IS NULL

语法: A IS NULL

操作类型: 所有类型

描述: 如果表达式A的值为NULL，则为TRUE；否则为FALSE

举例：

hive> select 1 from dual where null is null;

•非空判断: IS NOT NULL

语法: A IS NOT NULL

操作类型: 所有类型

描述: 如果表达式A的值为NULL，则为FALSE；否则为TRUE

举例：

hive> select 1 from dual where 1 is not null;

•LIKE比较: LIKE

语法: A LIKE B

操作类型: strings

描述: 如果字符串A或者字符串B为NULL，则返回NULL；如果字符串A符合表达式B   的正则语法，则为TRUE；否则为FALSE。B中字符”\_”表示任意单个字符，而字符”%”表示任意数量的字符。

举例：

hive> select 1 from dual where ‘key' like 'foot%';

1

hive> select 1 from dual where ‘key ' like 'foot\_\_\_\_';

1

**注意：否定比较时候用** **NOT A LIKE B**

hive> select 1 from dual where NOT ‘key ' like 'fff%';

•JAVA的LIKE操作: RLIKE

语法: A RLIKE B

操作类型: strings

描述: 如果字符串A或者字符串B为NULL，则返回NULL；如果字符串A符合JAVA正则表达式B的正则语法，则为TRUE；否则为FALSE。

举例：

hive> select 1 from dual where 'footbar’ rlike '^f.\*r$’;

1

**注意：判断一个字符串是否全为数字：**

hive>select 1 from dual where '123456' rlike '^\\d+$';

1

hive> select 1 from dual where '123456aa' rlike '^\\d+$';

1

•REGEXP操作: REGEXP

语法: A REGEXP B

操作类型: strings

描述: 功能与RLIKE相同

举例：

hive> select 1 from dual where ‘key' REGEXP '^f.\*r$';

1

## 第二部分：逻辑运算与数学运算

**Hive数学运算**

•加法操作: +

•减法操作: -

•乘法操作: \*

•除法操作: /

•取余操作: %

•位与操作: &

•位或操作: |

•位异或操作: ^

•位取反操作: ~

•加法操作: +

语法: A + B

操作类型：所有数值类型

说明：返回A与B相加的结果。结果的数值类型等于A的类型和B的类型的最小父类型（详见数据类型的继承关系）。比如，int + int 一般结果为int类型，而int + double 一般结果为double类型

举例：

hive> select 1 + 9 from dual;

10

•减法操作: -

语法: A – B

操作类型：所有数值类型

说明：返回A与B相减的结果。结果的数值类型等于A的类型和B的类型的最小父类型（详见数据类型的继承关系）。比如，int – int 一般结果为int类型，而int – double 一般结果为double类型

举例：

hive> select 10 – 5 from dual;

5

• **乘法操作** **: \***

语法: A \* B

操作类型：所有数值类型

说明：返回A与B相乘的结果。结果的数值类型等于A的类型和B的类型的最小父类型（详见数据类型的继承关系）。注意，如果A乘以B的结果超过默认结果类型的数值范围，则需要通过cast将结果转换成范围更大的数值类型

举例：

hive> select 40 \* 5 from dual;

200

• **除法操作** **: /**

语法: A / B

操作类型：所有数值类型

说明：返回A除以B的结果。结果的数值类型为double

举例：

hive> select 40 / 5 from dual;

8.0

**注意：** **hive** **中最高精度的数据类型是** **double,** **只精确到小数点后** **16** **位，在做除法运算的时候要** **特别注意**

hive>select ceil(28.0/6.999999999999999999999) from dual limit 1;

结果为4

hive>select ceil(28.0/6.99999999999999) from dual limit 1;

结果为5

• **取余操作** **: %**

语法: A % B

操作类型：所有数值类型

说明：返回A除以B的余数。结果的数值类型等于A的类型和B的类型的最小父类型（详见数据类型的继承关系）。

举例：

hive> select 41 % 5 from dual;

1

hive> select 8.4 % 4 from dual;

0.40000000000000036

**注意：精度在** **hive** **中是个很大的问题，类似这样的操作最好通过** **round** **指定精度**

hive> select round(8.4 % 4 , 2) from dual;

0.4

• **位与操作** **: &**

语法: A & B

操作类型：所有数值类型

说明：返回A和B按位进行与操作的结果。结果的数值类型等于A的类型和B的类型的最小父类型（详见数据类型的继承关系）。

举例：

hive> select 4 & 8 from dual;

0

hive> select 6 & 4 from dual;

4

• **位或操作** **: |**

语法: A | B

操作类型：所有数值类型

说明：返回A和B按位进行或操作的结果。结果的数值类型等于A的类型和B的类型的最小父类型（详见数据类型的继承关系）。

举例：

hive> select 4 | 8 from dual;

12

hive> select 6 | 8 from dual;

14

• **位异或操作** **: ^**

语法: A ^ B

操作类型：所有数值类型

说明：返回A和B按位进行异或操作的结果。结果的数值类型等于A的类型和B的类型的最小父类型（详见数据类型的继承关系）。

举例：

hive> select 4 ^ 8 from dual;

12

hive> select 6 ^ 4 from dual;

2

• **位取反操作** **: ~**

语法: ~A

操作类型：所有数值类型

说明：返回A按位取反操作的结果。结果的数值类型等于A的类型。

举例：

hive> select ~6 from dual;

-7

hive> select ~4 from dual;

-5

**Hive逻辑运算**

•逻辑与操作: AND

•逻辑或操作: OR

•逻辑非操作: NOT

• **逻辑与操作** **: AND**

语法: A AND B

操作类型：boolean

说明：如果A和B均为TRUE，则为TRUE；否则为FALSE。如果A为NULL或B为NULL，则为NULL

举例：

hive> select 1 from dual where 1=1 and 2=2;

1

• **逻辑或操作** **: OR**

语法: A OR B

操作类型：boolean

说明：如果A为TRUE，或者B为TRUE，或者A和B均为TRUE，则为TRUE；否则为FALSE

举例：

hive> select 1 from dual where 1=2 or 2=2;

1

• **逻辑非操作** **: NOT**

语法: NOT A

操作类型：boolean

说明：如果A为FALSE，或者A为NULL，则为TRUE；否则为FALSE

举例：

hive> select 1 from dual where not 1=2;

• **逻辑非操作** **: NOT**

语法: NOT A

操作类型：boolean

说明：如果A为FALSE，或者A为NULL，则为TRUE；否则为FALSE

举例：

hive> select 1 from dual where  not 1=2 ;

## 第三部分：数值运算

•取整函数: round

•指定精度取整函数: round

•向下取整函数: floor

•向上取整函数: ceil

•向上取整函数: ceiling

•取随机数函数: rand

•自然指数函数: exp

•以10为底对数函数: log10

•以2为底对数函数: log2

• 对数函数: log

•幂运算函数: pow

•幂运算函数: power

•开平方函数: sqrt

•二进制函数: bin

•十六进制函数: hex

•反转十六进制函数: unhex

•进制转换函数: conv

•绝对值函数: abs

•正取余函数: pmod

•正弦函数: sin

•反正弦函数: asin

•余弦函数: cos

•反余弦函数: acos

•positive函数: positive

•negative函数: negative

• **取整函数** **: round**

语法: round(double a)

返回值: BIGINT

说明: 返回double类型的整数值部分 （遵循四舍五入）

举例：

hive> select round(3.1415926) from dual;

3

hive> select round(3.5) from dual;

4

hive> create table dual as select round(9542.158) from dual;

hive> describe dual;

\_c0     bigint

• **指定精度取整函数** **: round**

语法: round(double a, int d)

返回值: DOUBLE

说明: 返回指定精度d的double类型

举例：

hive> select round(3.1415926,4) from dual;

3.1416

• **向下取整函数** **: floor**

语法: floor(double a)

返回值: BIGINT

说明: 返回等于或者小于该double变量的最大的整数

举例：

hive> select floor(3.1415926) from dual;

3

hive> select floor(25) from dual;

25

• **向上取整函数** **: ceil**

语法: ceil(double a)

返回值: BIGINT

说明: 返回等于或者大于该double变量的最小的整数

举例：

hive> select ceil(3.1415926) from dual;

4

hive> select ceil(46) from dual;

46

• **向上取整函数** **: ceiling**

语法: ceiling(double a)

返回值: BIGINT

说明: 与ceil功能相同

举例：

hive> select ceiling(3.1415926) from dual;

4

hive> select ceiling(46) from dual;

46

• **取随机数函数** **: rand**

语法: rand(),rand(int seed)

返回值: double

说明: 返回一个0到1范围内的随机数。如果指定种子seed，则会等到一个稳定的随机数序列

举例：

hive> select rand() from dual;

0.5577432776034763

• **自然指数函数** **: exp**

语法: exp(double a)

返回值: double

说明: 返回自然对数e的a次方

举例：

hive> select exp(2) from dual;

7.38905609893065

自然对数函数: ln

语法: ln(double a)

返回值: double

说明: 返回a的自然对数

• **以** **10** **为底对数函数** **: log10**

语法: log10(double a)

返回值: double

说明: 返回以10为底的a的对数

举例：

hive> select log10(100) from dual;

2.0

• **以** **2** **为底对数函数** **: log2**

语法: log2(double a)

返回值: double

说明: 返回以2为底的a的对数

举例：

hive> select log2(8) from dual;

3.0

• **对数函数** **: log**

语法: log(double base, double a)

返回值: double

说明: 返回以base为底的a的对数

举例：

hive> select log(4,256) from dual;

4.0

• **幂运算函数** **: pow**

语法: pow(double a, double p)

返回值: double

说明: 返回a的p次幂

举例：

hive> select pow(2,4) from dual;

16.0

• **幂运算函数** **: power**

语法: power(double a, double p)

返回值: double

说明: 返回a的p次幂,与pow功能相同

举例：

hive> select power(2,4) from dual;

16.0

• **开平方函数** **: sqrt**

语法: sqrt(double a)

返回值: double

说明: 返回a的平方根

举例：

hive> select sqrt(16) from dual;

4.0

• **二进制函数** **: bin**

语法: bin(BIGINT a)

返回值: string

说明: 返回a的二进制代码表示

举例：

hive> select bin(7) from dual;

111

• **十六进制函数** **: hex**

语法: hex(BIGINT a)

返回值: string

说明: 如果变量是int类型，那么返回a的十六进制表示；如果变量是string类型，则返回该字符串的十六进制表示

举例：

hive> select hex(17) from dual;

11

hive> select hex(‘abc’) from dual;

616263

• **反转十六进制函数** **: unhex**

语法: unhex(string a)

返回值: string

说明: 返回该十六进制字符串所代码的字符串

举例：

hive> select unhex(‘616263’) from dual;

abc

hive> select unhex(‘11’) from dual;

-

hive> select unhex(616263) from dual;

abc

• **进制转换函数** **: conv**

语法: conv(BIGINT num, int from\_base, int to\_base)

返回值: string

说明: 将数值num从from\_base进制转化到to\_base进制

举例：

hive> select conv(17,10,16) from dual;

11

hive> select conv(17,10,2) from dual;

10001

• **绝对值函数** **: abs**

语法: abs(double a)   abs(int a)

返回值: double        int

说明: 返回数值a的绝对值

举例：

hive> select abs(-3.9) from dual;

3.9

hive> select abs(10.9) from dual;

10.9

• **正取余函数** **: pmod**

语法: pmod(int a, int b),pmod(double a, double b)

返回值: int double

说明: 返回正的a除以b的余数

举例：

hive> select pmod(9,4) from dual;

1

hive> select pmod(-9,4) from dual;

3

• **正弦函数** **: sin**

语法: sin(double a)

返回值: double

说明: 返回a的正弦值

举例：

hive> select sin(0.8) from dual;

0.7173560908995228

• **反正弦函数** **: asin**

语法: asin(double a)

返回值: double

说明: 返回a的反正弦值

举例：

hive> select asin(0.7173560908995228) from dual;

0.8

• **余弦函数** **: cos**

语法: cos(double a)

返回值: double

说明: 返回a的余弦值

举例：

hive> select cos(0.9) from dual;

0.6216099682706644

• **反余弦函数** **: acos**

语法: acos(double a)

返回值: double

说明: 返回a的反余弦值

举例：

hive> select acos(0.6216099682706644) from dual;

0.9

• **positive** **函数** **: positive**

语法: positive(int a), positive(double a)

返回值: int double

说明: 返回a

举例：

hive> select positive(-10) from dual;

-10

hive> select positive(12) from dual;

12

• **negative** **函数** **: negative**

语法: negative(int a), negative(double a)

返回值: int double

说明: 返回-a

举例：

hive> select negative(-5) from dual;

5

hive> select negative(8) from dual;

-8

## 第四部分：日期函数

•UNIX时间戳转日期函数: from\_unixtime

• 获取当前UNIX时间戳函数: unix\_timestamp

•日期转UNIX时间戳函数: unix\_timestamp

• 指定格式日期转UNIX时间戳函数: unix\_timestamp

•日期时间转日期函数: to\_date

•日期转年函数: year

• 日期转月函数: month

• 日期转天函数: day

• 日期转小时函数: hour

• 日期转分钟函数: minute

• 日期转秒函数: second

• 日期转周函数: weekofyear

• 日期比较函数: datediff

• 日期增加函数: date\_add

• 日期减少函数: date\_sub

• **UNIX** **时间戳转日期函数** **: from\_unixtime**

语法: from\_unixtime(bigint unixtime[, string format])

返回值: string

说明: 转化UNIX时间戳（从1970-01-01 00:00:00 UTC到指定时间的秒数）到当前时区的时间格式

举例：

hive> select from\_unixtime(1323308943,'yyyyMMdd') from dual;

20111208

• **获取当前** **UNIX** **时间戳函数** **: unix\_timestamp**

语法: unix\_timestamp()

返回值: bigint

说明: 获得当前时区的UNIX时间戳

举例：

hive> select unix\_timestamp() from dual;

1323309615

• **日期转** **UNIX** **时间戳函数** **: unix\_timestamp**

语法: unix\_timestamp(string date)

返回值: bigint

说明: 转换格式为"yyyy-MM-dd HH:mm:ss"的日期到UNIX时间戳。如果转化失败，则返回0。

举例：

hive> select unix\_timestamp('2011-12-07 13:01:03') from dual;

1323234063

• **指定格式日期转** **UNIX** **时间戳函数** **: unix\_timestamp**

语法: unix\_timestamp(string date, string pattern)

返回值: bigint

说明: 转换pattern格式的日期到UNIX时间戳。如果转化失败，则返回0。

举例：

hive> select unix\_timestamp('20111207 13:01:03','yyyyMMdd HH:mm:ss') from dual;

1323234063

• **日期时间转日期函数** **: to\_date**

语法: to\_date(string timestamp)

返回值: string

说明: 返回日期时间字段中的日期部分。

举例：

hive> select to\_date('2011-12-08 10:03:01') from dual;

2011-12-08

• **日期转年函数** **: year**

语法: year(string date)

返回值: int

说明: 返回日期中的年。

举例：

hive> select year('2011-12-08 10:03:01') from dual;

2011

hive> select year('2012-12-08') from dual;

2012

• **日期转月函数** **: month**

语法: month (string date)

返回值: int

说明: 返回日期中的月份。

举例：

hive> select month('2011-12-08 10:03:01') from dual;

12

hive> select month('2011-08-08') from dual;

8

• **日期转天函数** **: day**

语法: day (string date)

返回值: int

说明: 返回日期中的天。

举例：

hive> select day('2011-12-08 10:03:01') from dual;

8

hive> select day('2011-12-24') from dual;

24

• **日期转小时函数** **: hour**

语法: hour (string date)

返回值: int

说明: 返回日期中的小时。

举例：

hive> select hour('2011-12-08 10:03:01') from dual;

10

• **日期转分钟函数** **: minute**

语法: minute (string date)

返回值: int

说明: 返回日期中的分钟。

举例：

hive> select minute('2011-12-08 10:03:01') from dual;

3

• **日期转秒函数** **: second**

语法: second (string date)

返回值: int

说明: 返回日期中的秒。

举例：

hive> select second('2011-12-08 10:03:01') from dual;

1

• **日期转周函数** **: weekofyear**

语法: weekofyear (string date)

返回值: int

说明: 返回日期在当前的周数。

举例：

hive> select weekofyear('2011-12-08 10:03:01') from dual;

49

• **日期比较函数** **: datediff**

语法: datediff(string enddate, string startdate)

返回值: int

说明: 返回结束日期减去开始日期的天数。

举例：

hive> select datediff('2012-12-08','2012-05-09') from dual;

213

• **日期增加函数** **: date\_add**

语法: date\_add(string startdate, int days)

返回值: string

说明: 返回开始日期startdate增加days天后的日期。

举例：

hive> select date\_add('2012-12-08',10) from dual;

2012-12-18

• **日期减少函数** **: date\_sub**

语法: date\_sub (string startdate, int days)

返回值: string

说明: 返回开始日期startdate减少days天后的日期。

举例：

hive> select date\_sub('2012-12-08',10) from dual;

2012-11-28

## 第五部分：条件函数

•If函数: if

•非空查找函数: COALESCE

•条件判断函数：CASE

• **If** **函数** **: if**

语法: if(boolean testCondition, T valueTrue, T valueFalseOrNull)

返回值: T

说明:  当条件testCondition为TRUE时，返回valueTrue；否则返回valueFalseOrNull

举例：

hive> select if(1=2,100,200) from dual;

200

hive> select if(1=1,100,200) from dual;

100

• **非空查找函数** **: COALESCE**

语法: COALESCE(T v1, T v2, …)

返回值: T

说明:  返回参数中的第一个非空值；如果所有值都为NULL，那么返回NULL

举例：

hive> select COALESCE(null,'100','50′) from dual;

100

**条件判断函数：** **CASE**

语法 : CASE a WHEN b THEN c [WHEN d THEN e]\* [ELSE f] END

返回值 : T

说明：如果 a 等于 b ，那么返回 c ；如果 a 等于 d ，那么返回 e ；否则返回 f

举例：

hive> Select case 100 when 50 then 'tom' when 100 then 'mary' else 'tim' end from dual;

mary

## 第六部分：字符串函数

•字符串长度函数：length

•字符串反转函数：reverse

•字符串连接函数：concat

• 带分隔符字符串连接函数：concat\_ws

• 字符串截取函数：substr,substring

• 字符串截取函数：substr,substring

• 字符串转大写函数：upper,ucase

• 字符串转小写函数：lower,lcase

• 去空格函数：trim

• 左边去空格函数：ltrim

• 右边去空格函数：rtrim

•正则表达式替换函数：regexp\_replace

•正则表达式解析函数：regexp\_extract

•URL解析函数：parse\_url

•json解析函数：get\_json\_object

•空格字符串函数：space

•重复字符串函数：repeat

•首字符ascii函数：ascii

•左补足函数：lpad

•右补足函数：rpad

•分割字符串函数: split

•集合查找函数: find\_in\_set

• **字符串长度函数：** **length**

语法: length(string A)

返回值: int

说明：返回字符串A的长度

举例：

hive> select length('abcedfg') from dual;

7

• **字符串反转函数：** **reverse**

语法: reverse(string A)

返回值: string

说明：返回字符串A的反转结果

举例：

hive> select reverse(abcedfg’) from dual;

gfdecba

• **字符串连接函数：** **concat**

语法: concat(string A, string B…)

返回值: string

说明：返回输入字符串连接后的结果，支持任意个输入字符串

举例：

hive> select concat(‘abc’,'def’,'gh’) from dual;

abcdefgh

• **带分隔符字符串连接函数：** **concat\_ws**

语法: concat\_ws(string SEP, string A, string B…)

返回值: string

说明：返回输入字符串连接后的结果，SEP表示各个字符串间的分隔符

举例：

hive> select concat\_ws(',','abc','def','gh') from dual;

abc,def,gh

• **字符串截取函数：** **substr,substring**

语法: substr(string A, int start),substring(string A, int start)

返回值: string

说明：返回字符串A从start位置到结尾的字符串

举例：

hive> select substr('abcde',3) from dual;

cde

hive> select substring('abcde',3) from dual;

cde

hive>  select substr('abcde',-1) from dual;  （和ORACLE相同）

e

• **字符串截取函数：** **substr,substring**

语法: substr(string A, int start, int len),substring(string A, int start, int len)

返回值: string

说明：返回字符串A从start位置开始，长度为len的字符串

举例：

hive> select substr('abcde',3,2) from dual;

cd

hive> select substring('abcde',3,2) from dual;

cd

hive>select substring('abcde',-2,2) from dual;

de

• **字符串转大写函数：** **upper,ucase**

语法: upper(string A) ucase(string A)

返回值: string

说明：返回字符串A的大写格式

举例：

hive> select upper('abSEd') from dual;

ABSED

hive> select ucase('abSEd') from dual;

ABSED

• **字符串转小写函数：** **lower,lcase**

语法: lower(string A) lcase(string A)

返回值: string

说明：返回字符串A的小写格式

举例：

hive> select lower('abSEd') from dual;

absed

hive> select lcase('abSEd') from dual;

absed

• **去空格函数：** **trim**

语法: trim(string A)

返回值: string

说明：去除字符串两边的空格

举例：

hive> select trim(' abc ') from dual;

abc

• **左边去空格函数：** **ltrim**

语法: ltrim(string A)

返回值: string

说明：去除字符串左边的空格

举例：

hive> select ltrim(' abc ') from dual;

abc

• **右边去空格函数：** **rtrim**

语法: rtrim(string A)

返回值: string

说明：去除字符串右边的空格

举例：

hive> select rtrim(' abc ') from dual;

abc

• **正则表达式替换函数：** **regexp\_replace**

语法: regexp\_replace(string A, string B, string C)

返回值: string

说明：将字符串A中的符合java正则表达式B的部分替换为C。注意，在有些情况下要使用转义字符,类似oracle中的regexp\_replace函数。

举例：

hive> select regexp\_replace('foobar', 'oo|ar', '') from dual;

fb

• **正则表达式解析函数：** **regexp\_extract**

语法: regexp\_extract(string subject, string pattern, int index)

返回值: string

说明：将字符串subject按照pattern正则表达式的规则拆分，返回index指定的字符。

举例：

hive> select regexp\_extract('foothebar', 'foo(.\*?)(bar)', 1) from dual;

the

hive> select regexp\_extract('foothebar', 'foo(.\*?)(bar)', 2) from dual;

bar

hive> select regexp\_extract('foothebar', 'foo(.\*?)(bar)', 0) from dual;

foothebar

**注意，在有些情况下要使用转义字符，下面的等号要用双竖线转** **义，这是** **java** **正则表达式的规则。**

select data\_field,

regexp\_extract(data\_field,'.\*?bgStart\\=([^&]+)',1) as aaa,

regexp\_extract(data\_field,'.\*?contentLoaded\_headStart\\=([^&]+)',1) as bbb,

regexp\_extract(data\_field,'.\*?AppLoad2Req\\=([^&]+)',1) as ccc

from pt\_nginx\_loginlog\_st

where pt = '2012-03-26' limit 2;

• **URL** **解析函数：** **parse\_url**

语法: parse\_url(string urlString, string partToExtract [, string keyToExtract])

返回值: string

说明：返回URL中指定的部分。partToExtract的有效值为：HOST, PATH, QUERY, REF, PROTOCOL, AUTHORITY, FILE, and USERINFO.

举例：

hive> select parse\_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'HOST') from dual;

facebook.com

hive> select parse\_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'QUERY', 'k1') from dual;

v1

• **json** **解析函数：** **get\_json\_object**

语法: get\_json\_object(string json\_string, string path)

返回值: string

说明：解析json的字符串json\_string,返回path指定的内容。如果输入的json字符串无效，那么返回NULL。

举例：

hive> select  get\_json\_object('{"store":

>   {"fruit":\[{"weight":8,"type":"apple"},{"weight":9,"type":"pear"}],

>    "bicycle":{"price":19.95,"color":"red"}

>   },

>  "email":"amy@only\_for\_json\_udf\_test.net",

>  "owner":"amy"

> }

> ','$.owner') from dual;

amy

• **空格字符串函数：** **space**

语法: space(int n)

返回值: string

说明：返回长度为n的字符串

举例：

hive> select space(10) from dual;

hive> select length(space(10)) from dual;

10

• **重复字符串函数：** **repeat**

语法: repeat(string str, int n)

返回值: string

说明：返回重复n次后的str字符串

举例：

hive> select repeat('abc',5) from dual;

abcabcabcabcabc

• **首字符** **ascii** **函数：** **ascii**

语法: ascii(string str)

返回值: int

说明：返回字符串str第一个字符的ascii码

举例：

hive> select ascii('abcde') from dual;

97

• **左补足函数：** **lpad**

语法: lpad(string str, int len, string pad)

返回值: string

说明：将str进行用pad进行左补足到len位

举例：

hive> select lpad('abc',10,'td') from dual;

tdtdtdtabc

**注意：与** **GP** **，** **ORACLE** **不同，** **pad 不能默认**

• **右补足函数：** **rpad**

语法: rpad(string str, int len, string pad)

返回值: string

说明：将str进行用pad进行右补足到len位

举例：

hive> select rpad('abc',10,'td') from dual;

abctdtdtdt

• **分割字符串函数** **: split**

语法:  split(string str, string pat)

返回值:  array

说明: 按照pat字符串分割str，会返回分割后的字符串数组

举例：

hive> select split('abtcdtef','t') from dual;

["ab","cd","ef"]

• **集合查找函数** **: find\_in\_set**

语法: find\_in\_set(string str, string strList)

返回值: int

说明: 返回str在strlist第一次出现的位置，strlist是用逗号分割的字符串。如果没有找该str字符，则返回0

举例：

hive> select find\_in\_set('ab','ef,ab,de') from dual;

2

hive> select find\_in\_set('at','ef,ab,de') from dual;

0

## 第七部分：集合统计函数

•个数统计函数: count

•总和统计函数: sum

•平均值统计函数: avg

• 最小值统计函数: min

•最大值统计函数: max

• **个数统计函数** **: count**

语法: count(\*), count(expr), count(DISTINCT expr[, expr\_.])

返回值: int

说明: count(\*)统计检索出的行的个数，包括NULL值的行；count(expr)返回指定字段的非空值的个数；count(DISTINCT expr[, expr\_.])返回指定字段的不同的非空值的个数

举例：

hive> select count(\*) from dual;

20

hive> select count(distinct t) from dual;

10

• **总和统计函数** **: sum**

语法: sum(col), sum(DISTINCT col)

返回值: double

说明: sum(col)统计结果集中col的相加的结果；sum(DISTINCT col)统计结果中col不同值相加的结果

举例：

hive> select sum(t) from dual;

100

hive> select sum(distinct t) from dual;

70

• **平均值统计函数** **: avg**

语法: avg(col), avg(DISTINCT col)

返回值: double

说明: avg(col)统计结果集中col的平均值；avg(DISTINCT col)统计结果中col不同值相加的平均值

举例：

hive> select avg(t) from dual;

50

hive> select avg (distinct t) from dual;

30

• **最小值统计函数** **: min**

语法: min(col)

返回值: double

说明: 统计结果集中col字段的最小值

举例：

hive> select min(t) from dual;

20

• **最大值统计函数** **: max**

语法: maxcol)

返回值: double

说明: 统计结果集中col字段的最大值

举例：

hive> select max(t) from dual;

120

## 第八部分：符合类型构建操作

•Map类型构建: map

•Struct类型构建: struct

•array类型构建: array

• **Map** **类型构建** **: map**

语法: map (key1, value1, key2, value2, …)

说明：根据输入的key和value对构建map类型

举例：

hive> Create table alex\_test as select map('100','tom','200','mary') as t from dual;

hive> describe alex\_test;

t       map<string,string>

hive> select t from alex\_test;

{"100":"tom","200":"mary"}

• **Struct** **类型构建** **: struct**

语法: struct(val1, val2, val3, …)

说明：根据输入的参数构建结构体struct类型

举例：

hive> create table alex\_test as select struct('tom','mary','tim') as t from dual;

hive> describe alex\_test;

t       struct<col1:string,col2:string,col3:string>

hive> select t from alex\_test;

{"col1":"tom","col2":"mary","col3":"tim"}

• **array** **类型构建** **: array**

语法: array(val1, val2, …)

说明：根据输入的参数构建数组array类型

举例：

hive> create table alex\_test as select array("tom","mary","tim") as t from dual;

hive> describe alex\_test;

t       array<string>

hive> select t from alex\_test;

["tom","mary","tim"]

## 第九部分：复杂类型访问操作

•array类型访问: A[n]

•map类型访问: M[key]

•struct类型访问: S.x

• **array** **类型访问** **: A[n]**

语法: A[n]

操作类型: A为array类型，n为int类型

说明：返回数组A中的第n个变量值。数组的起始下标为0。比如，A是个值为['foo', 'bar']的数组类型，那么A[0]将返回'foo',而A[1]将返回'bar'

举例：

hive> create table alex\_test as select array("tom","mary","tim") as t from dual;

hive> select t[0],t[1],t[2] from alex\_test;

tom     mary    tim

• **map** **类型访问** **: M[key]**

语法: M[key]

操作类型: M为map类型，key为map中的key值

说明：返回map类型M中，key值为指定值的value值。比如，M是值为{'f' -> 'foo', 'b' -> 'bar', 'all' -> 'foobar'}的map类型，那么M['all']将会返回'foobar'

举例：

hive> Create table alex\_test as select map('100','tom','200','mary') as t from dual;

hive> select t['200'],t['100'] from alex\_test;

mary    tom

• **struct** **类型访问** **: S.x**

语法: S.x

操作类型: S为struct类型

说明：返回结构体S中的x字段。比如，对于结构体struct foobar {int foo, int bar}，foobar.foo返回结构体中的foo字段

举例：

hive> create table alex\_test as select struct('tom','mary','tim') as t from dual;

hive> describe alex\_test;

t       struct<col1:string,col2:string,col3:string>

hive> select t.col1,t.col3 from alex\_test;

tom     tim

## 第十部分：复杂类型长度统计函数

•Map类型长度函数: size(Map<K.V>)

•array类型长度函数: size(Array<T>)

•类型转换函数

• **Map** **类型长度函数** **: size(Map<K.V>)**

语法: size(Map<K.V>)

返回值: int

说明: 返回map类型的长度

举例：

hive> select size(map('100','tom','101','mary')) from dual;

2

• **array** **类型长度函数** **: size(Array<T>)**

语法: size(Array<T>)

返回值: int

说明: 返回array类型的长度

举例：

hive> select size(array('100','101','102','103')) from dual;

4

• **类型转换函数**

类型转换函数: cast

语法: cast(expr as <type>)

返回值: Expected "=" to follow "type"

说明: 返回array类型的长度

举例：

hive> select cast(1 as bigint) from dual;

1

## 类型转换

|  |  |
| --- | --- |
| string --> int | cast('1323308943' as int) |
|  |  |
|  |  |

# Hive JDBC

## 第一部分：搭建Hive JDBC开发环境

**搭建：Steps**

•新建工程hiveTest

•导入Hive依赖的包

•Hive  命令行启动Thrift服务

•hive --service hiveserver &

## 第二部分：基本操作对象的介绍

**Connection**

•说明：与Hive连接的Connection对象

•Hive 的连接

•jdbc:hive://IP:10000/default“

•获取Connection的方法

•DriverManager.getConnection("jdbc:hive://IP:10000/default", "", "");

**Statement**

•说明: 用于执行语句

•创建方法

•Statement stmt = con.createStatement();

•主要方法

•executeQuery

•execute

**ResultSet**

•说明：用来存储结果集

•创建方法

•stmt.executeQuery 方法得到

•主要方法

•getString()

**特殊类型的处理**

•Array

•Map

•Struct

## 第三部分：简单示例

**背景**

     随着 3G 时代的到来，选择各种移动终端来上网的人越 来越多，如手机、平板等，这里面蕴含着巨大的商机 ，因为终端可以直接精确到人。所以对用户上网日志 的分析，也成为一个新热点，这些在数据库时代是不 可想象的，因为数据规模超大，云计算解决了这个问 题

**需求描述**

•分析用户的手机上网日志，得到 **手机流量统计** **信息与根据终端来获得流量的统计信息**

**表规划**

• **日志表** **log**

• **字段**

       用户手机号码 ，终端类型 ，访问的URL ，访问开始时间 ，访问结束时间 ，访问流量

    phone string ，client\_type  string ,url  string,visit\_begintime timestamp,visit\_endtime timestamp,traffic  bigint

• **数据文件**

•数据间以\t分隔，行之间以\n分隔

•数据示例：

      13154587453     andorid www.baidu.com   2011-09-01 17:30:08     2011-09-01 17:31:09     10

•数据文件位置：

    /home/test/log.txt

• **手机号归属地** **phone\_location**

• **字段**

用户手机号码 ，归属地

phone string ，location stirng

• **数据文件**

数据间以\t分隔，行之间以\n分隔

数据示例：1334567890 chaoyang

数据文件位置：

/home/test/phone\_local.txt

• **手机流量统计信息** **total\_traffic\_by\_phone**

• **字段**

手机号码 ,总流量(k) , 归属地, 访问总时长(秒)

               phone string,totol\_traffic bigint,area string,visit\_total\_time timestamp

• **数据文件**

**数据间以** **\t** **分隔，行之间以** **\n** **分隔**

     数据示例：

           13154587453 102 chaoyang  123

• **终端类型统计信息** **total\_traffic\_by\_client\_type**

• **字段**

终端类型 ,访问总流量(k)

       client\_type string,totoal\_traffic bigint

• **数据文件**

**数据间以** **\t** **分隔，行之间以** **\n** **分隔**

     数据示例：

 andorid 1034

**实现步骤**

•初始化数据，表

•示例：create external table phone\_location (phone string,area string) row format delimited fields terminated by '\t' lines terminated by '\n' stored as textfile

•编写处理逻辑

**Java 代码示例**

try {

Class.forName(driverName);

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

Statement stmt = con.createStatement();

String sql = "show tables ";

// show tables

System.out.println("Running: " + sql);

ResultSet res = stmt.executeQuery(sql);

if (res.next()) {

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

}

## 第四部分：基于Hive的数据库连接池

**基于Hive的数据库连接池的实现**

•使用DataSource作为数据源的实现

•DBConnectionManager 采用单例模式

•提供获得连接，关闭连接的方法

•setupDataSource()

•DBConnectionManager getInstance()

•close(Connection conn)

•synchronized Connection getConnection()

# hive参数

**第一部分：Hive 参数**

**hive.exec.max.created.files**

•说明：所有hive运行的map与reduce任务可以产生的文件的和

•默认值:100000

**hive.exec.dynamic.partition**

•说明：是否为自动分区

•默认值：false

**hive.mapred.reduce.tasks.speculative.execution**

•说明：是否打开推测执行

•默认值：true

**hive.input.format**

•说明：Hive默认的input format

•默认值： org.apache.hadoop.hive.ql.io.CombineHiveInputFormat

•如果有问题可以使用org.apache.hadoop.hive.ql.io.HiveInputFormat

**hive.exec.counters.pull.interval**

•说明：Hive与JobTracker拉取counter信息的时间

•默认值：1000ms

**hive.script.recordreader**

•说明：使用脚本时默认的读取类

•默认值： org.apache.hadoop.hive.ql.exec.TextRecordReader

**hive.script.recordwriter**

•说明：使用脚本时默认的数据写入类

•默认值： org.apache.hadoop.hive.ql.exec.TextRecordWriter

**hive.mapjoin.check.memory.rows**

•说明： 内存里可以存储数据的行数

•默认值： 100000

**hive.mapjoin.smalltable.filesize**

•说明：输入小表的文件大小的阀值，如果小于该值，就采用普通的join

•默认值： 25000000

**hive.auto.convert.join**

•说明：是不是依据输入文件的大小，将Join转成普通的Map Join

•默认值： false

**hive.mapjoin.followby.gby.localtask.max.memory.usage**

•说明：map join做group by 操作时，可以使用多大的内存来存储数据，如果数据太大，则不会保存在内存里

•默认值：0.55

**hive.mapjoin.localtask.max.memory.usage**

•说明：本地任务可以使用内存的百分比

•默认值： 0.90

**hive.heartbeat.interval**

•说明：在进行MapJoin与过滤操作时，发送心跳的时间

•默认值1000

**hive.merge.size.per.task**

•说明： 合并后文件的大小

•默认值： 256000000

**hive.mergejob.maponly**

•说明： 在只有Map任务的时候 合并输出结果

•默认值： true

**hive.merge.mapredfiles**

•默认值： 在作业结束的时候是否合并小文件

•说明： false

**hive.merge.mapfiles**

•说明：Map-Only Job是否合并小文件

•默认值：true

**hive.hwi.listen.host**

•说明：Hive UI 默认的host

•默认值：0.0.0.0

**hive.hwi.listen.port**

•说明：Ui监听端口

•默认值：9999

**hive.exec.parallel.thread.number**

•说明：hive可以并行处理Job的线程数

•默认值：8

**hive.exec.parallel**

•说明：是否并行提交任务

•默认值：false

**hive.exec.compress.output**

•说明：输出使用压缩

•默认值： false

**hive.mapred.mode**

•说明： MapReduce的操作的限制模式，操作的运行在该模式下没有什么限制

•默认值： nonstrict

**hive.join.cache.size**

•说明： join操作时，可以存在内存里的条数

•默认值： 25000

**hive.mapjoin.cache.numrows**

•说明： mapjoin 存在内存里的数据量

•默认值：25000

**hive.join.emit.interval**

•说明： 有连接时Hive在输出前，缓存的时间

•默认值： 1000

**hive.optimize.groupby**

•说明：在做分组统计时，是否使用bucket table

•默认值： true

**hive.fileformat.check**

•说明：是否检测文件输入格式

•默认值：true

**hive.metastore.client.connect.retry.delay**

•说明： client 连接失败时,retry的时间间隔

•默认值：1秒

**hive.metastore.client.socket.timeout**

•说明:  Client socket 的超时时间

•默认值：20秒

**mapred.reduce.tasks**

•默认值：-1

•说明：每个任务reduce的默认值

 -1 代表自动根据作业的情况来设置reduce的值

**hive.exec.reducers.bytes.per.reducer**

•默认值： 1000000000 （1G）

•说明：每个reduce的接受的数据量

    如果送到reduce的数据为10G,那么将生成10个reduce任务

**hive.exec.reducers.max**

•默认值：999

•说明： reduce的最大个数

**hive.exec.reducers.max**

•默认值：999

•说明： reduce的最大个数

**hive.metastore.warehouse.dir**

•默认值：/user/hive/warehouse

•说明： 默认的数据库存放位置

**hive.default.fileformat**

•默认值：TextFile

•说明： 默认的fileformat

**hive.map.aggr**

•默认值：true

•说明： Map端聚合，相当于combiner

**hive.exec.max.dynamic.partitions.pernode**

•默认值：100

•说明：每个任务节点可以产生的最大的分区数

**hive.exec.max.dynamic.partitions**

•默认值：1000

•说明： 默认的可以创建的分区数

**hive.metastore.server.max.threads**

•默认值：100000

•说明： metastore默认的最大的处理线程数

**hive.metastore.server.min.threads**

•默认值：200

•说明： metastore默认的最小的处理线程数

# Hive 高级编程

## 第一部分：产生背景

**产生背景**

•为了满足客户个性化的需求，Hive被设计成一个很开放的系统，很多内容都支持用户定制，包括：

•文件格式：Text File，Sequence File

•内存中的数据格式： Java Integer/String, Hadoop  IntWritable/Text

•用户提供的 map/reduce 脚本：不管什么语言，利用 stdin/stdout 传输数据

•用户自定义函数

**自定义函数**

•虽然Hive提供了很多函数，但是有些还是难以满足我们的需求。因此Hive提供了自定义函数开发

•自定义函数包括三种UDF、UADF、UDTF

•UDF(User-Defined-Function)

•UDAF（User- Defined Aggregation Funcation）

•UDTF(User-Defined Table-Generating Functions)  用来解决 输入一行输出多行(On-to-many maping) 的需求。

**HIVE中使用定义的函数的三种方式**

•在HIVE会话中add 自定义函数的jar文件，然后创建function，继而使用函数

•在进入HIVE会话之前先自动执行创建function，不用用户手工创建

•把自定义的函数写到系统函数中，使之成为HIVE的一个默认函数，这样就不需要create temporary function

## 第二部分：UDF

**UDF用法**

•UDF(User-Defined-Function)

•UDF函数可以直接应用于select语句，对查询结构做格式化处理后，再输出内容

•编写UDF函数的时候需要注意一下几点

•自定义UDF需要继承org.apache.hadoop.hive.ql.UDF

•需要实现evaluate函数

•evaluate函数支持重载

•UDF只能实现一进一出的操作，如果需要实现多进一出，则需要实现UDAF

**UDF用法代码示例**

**import** org.apache.Hadoop.hive.ql.exec.UDF

**public**  **class** Helloword  **extends** UDF{

**public** String evaluate(){

**return** "hello world!";

     }

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

**return** "hello world: " + str;

     }

}

**开发步骤**

•开发代码

•把程序打包放到目标机器上去

•进入hive客户端

•添加jar包：hive>add jar /run/jar/udf\_test.jar;

•创建临时函数：hive>CREATE TEMPORARY FUNCTION my\_add AS 'com.hive.udf.Add ‘

•查询HQL语句：

•SELECT my\_add (8, 9) FROM scores;

•SELECT my\_add (scores.math, scores.art) FROM scores;

•销毁临时函数：hive> DROP TEMPORARY FUNCTION my\_add ;

•细节

•在使用UDF的时候，会自动进行类型转换，例如：   
SELECT my\_add (8,9.1) FROM scores;

•结果是17.1，UDF将类型为Int的参数转化成double。类型的饮食转换是通过UDFResolver来进行控制的

## 第三部分：UDAF

**UDAF**

•Hive查询数据时，有些聚类函数在HQL没有自带，需要用户自定义实现

•用户自定义聚合函数: Sum, Average…… n – 1

•UDAF（User- Defined Aggregation Funcation）

**用法**

•一下两个包是必须的import org.apache.hadoop.hive.ql.exec.UDAF和 org.apache.hadoop.hive.ql.exec.UDAFEvaluator

**开发步骤**

•函数类需要继承UDAF类，内部类Evaluator实UDAFEvaluator接口

•Evaluator需要实现 init、iterate、terminatePartial、merge、terminate这几个函数

a）init函数实现接口UDAFEvaluator的init函数。

b）iterate接收传入的参数，并进行内部的轮转。其返回类型为boolean。

c）terminatePartial无参数，其为iterate函数轮转结束后，返回轮转数据，terminatePartial类似于hadoop的Combiner。

d）merge接收terminatePartial的返回结果，进行数据merge操作，其返回类型为boolean。

e）terminate返回最终的聚集函数结果。

**执行步骤**

•执行求平均数函数的步骤

a）将java文件编译成Avg\_test.jar。

b）进入hive客户端添加jar包：

hive>add jar /run/jar/Avg\_test.jar。

c）创建临时函数：

hive>create temporary function avg\_test 'hive.udaf.Avg';

d）查询语句：

hive>select avg\_test(scores.math) from scores;

e）销毁临时函数：

hive>drop temporary function avg\_test;

**UDAF代码示例**

public class MyAvg extends UDAF {

public static class AvgEvaluator implements UDAFEvaluator {

}

public void init() {}

public boolean iterate(Double o) {}

public AvgState terminatePartial() {}

public boolean terminatePartial(Double o) { }

public Double terminate() {}

}

## 第四部分：UDTF

**UDTF**

•UDTF(User-Defined Table-Generating Functions)  用来解决 输入一行输出多行(On-to-many maping) 的需求。

**开发步骤**

•UDTF步骤：

•必须继承org.apache.Hadoop.hive.ql.udf.generic.GenericUDTF

•实现initialize, process, close三个方法

•UDTF首先会

•调用initialize方法，此方法返回UDTF的返回行的信息（返回个数，类型）   
初始化完成后，会调用process方法，对传入的参数进行处理，可以通过forword()方法把结果返回

•最后close()方法调用，对需要清理的方法进行清理

**使用方法**

•UDTF有两种使用方法，一种直接放到select后面，一种和lateral view一起使用

•直接select中使用：select explode\_map(properties) as (col1,col2) from src;

•不可以添加其他字段使用：select a, explode\_map(properties) as (col1,col2) from src

•不可以嵌套调用：select explode\_map(explode\_map(properties)) from src

•不可以和group by/cluster by/distribute by/sort by一起使用：select explode\_map(properties) as (col1,col2) from src group by col1, col2

•和lateral view一起使用：select src.id, mytable.col1, mytable.col2 from src lateral view explode\_map(properties) mytable as col1, col2;

此方法更为方便日常使用。执行过程相当于单独执行了两次抽取，然后union到一个表里。

**lateral view**

• Lateral View语法

•lateralView: LATERAL VIEW udtf(expression) tableAlias AS columnAlias (',' columnAlias)\* fromClause: FROM baseTable (lateralView)\*

•Lateral View用于UDTF(user-defined table generating functions)中将行转成列，例如explode().

•目前Lateral View不支持有上而下的优化。如果使用Where子句，查询可能将不被编译。解决方法见：

     此时，在查询之前执行set hive.optimize.ppd=false;

•  例子

•pageAds。它有两个列

|  |  |
| --- | --- |
| string pageid | Array<int> adid\_list |
| " front\_page" | [1, 2, 3] |
| "contact\_page " | [ 3, 4, 5] |

•SELECT pageid, adid FROM pageAds LATERAL VIEW explode(adid\_list) adTable AS adid;

•将输出如下结果

string pageid int adid

"front\_page" 1

…….

“contact\_page" 3

**代码示例**

public class MyUDTF extends GenericUDTF{

public StructObjectInspector initialize(ObjectInspector[] args) {}

public void process(Object[] args) throws HiveException { }

}

实现：切分 ” key:value;key:value;” 这种字符串， 返回结果为 key, value 两个字段

# Hive QL

## 第一部分：ＤＤＬ

**DDL**

•建表

•删除表

•修改表结构

•创建／删除视图

•创建数据库

•显示命令

### 表

CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table\_name   
  [(col\_name data\_type [COMMENT col\_comment], ...)]   
  [COMMENT table\_comment]   
  [PARTITIONED BY (col\_name data\_type [COMMENT col\_comment], ...)]   
  [CLUSTERED BY (col\_name, col\_name, ...)   
  [SORTED BY (col\_name [ASC|DESC], ...)] INTO num\_buckets BUCKETS]   
  [ROW FORMAT row\_format]   
  [STORED AS file\_format]   
  [LOCATION hdfs\_path]

•CREATE TABLE 创建一个指定名字的表。如果相同名字的表已经存在，则抛出异常；用户可以用 IF NOT EXIST 选项来忽略这个异常

•EXTERNAL 关键字可以让用户创建一个外部表，在建表的同时指定一个指向实际数据的路径（LOCATION）

•LIKE 允许用户复制现有的表结构，但是不复制数据

•COMMENT可以为表与字段增加描述

•ROW FORMAT

    DELIMITED [FIELDS TERMINATED BY char] [COLLECTION ITEMS TERMINATED BY char]

        [MAP KEYS TERMINATED BY char] [LINES TERMINATED BY char]

   | SERDE serde\_name [WITH SERDEPROPERTIES (property\_name=property\_value, property\_name=property\_value, ...)]

         用户在建表的时候可以自定义 SerDe 或者使用自带的 SerDe。如果没有指定 ROW FORMAT 或者 ROW FORMAT DELIMITED，将会使用自带的 SerDe。在建表的时候，用户还需要为表指定列，用户在指定表的列的同时也会指定自定义的 SerDe，Hive 通过 SerDe 确定表的具体的列的数据。

•STORED AS

            SEQUENCEFILE

            | TEXTFILE

            | RCFILE

            | INPUTFORMAT input\_format\_classname OUTPUTFORMAT             output\_format\_classname

       如果文件数据是纯文本，可以使用 STORED AS TEXTFILE。如果数据需要压缩，使用 STORED AS SEQUENCE 。

**建立外部表**

CREATE EXTERNAL TABLE page\_view(viewTime INT, userid BIGINT,

     page\_url STRING, referrer\_url STRING,

     ip STRING COMMENT 'IP Address of the User',

     country STRING COMMENT 'country of origination')

 COMMENT 'This is the staging page view table'

 ROW FORMAT DELIMITED FIELDS TERMINATED BY '\054'

 STORED AS TEXTFILE

 LOCATION '<hdfs\_location>';

**建分区表**

CREATE TABLE par\_table(viewTime INT, userid BIGINT,

     page\_url STRING, referrer\_url STRING,

     ip STRING COMMENT 'IP Address of the User')

 COMMENT 'This is the page view table'

 PARTITIONED BY(date STRING, pos STRING)

ROW FORMAT DELIMITED ‘\t’

   FIELDS TERMINATED BY '\n'

STORED AS SEQUENCEFILE;

**建Bucket表**

CREATE TABLE par\_table(viewTime INT, userid BIGINT,

     page\_url STRING, referrer\_url STRING,

     ip STRING COMMENT 'IP Address of the User')

 COMMENT 'This is the page view table'

 PARTITIONED BY(date STRING, pos STRING)

 CLUSTERED BY(userid) SORTED BY(viewTime) INTO 32 BUCKETS

 ROW FORMAT DELIMITED ‘\t’

   FIELDS TERMINATED BY '\n'

STORED AS SEQUENCEFILE;

**复制一个空表**

CREATE TABLE empty\_key\_value\_store

LIKE key\_value\_store;

### 删除表

DROP TABLE table\_name

### 修改表结构

•增加分区、删除分区

•重命名表

•修改列的名字、类型、位置、注释

•增加/更新列

•增加表的元数据信息

**增加、删除分区**

•增加

ALTER TABLE table\_name ADD [IF NOT EXISTS] partition\_spec [ LOCATION 'location1' ] partition\_spec [ LOCATION 'location2' ] ...

      partition\_spec:

  : PARTITION (partition\_col = partition\_col\_value, partition\_col = partiton\_col\_value, ...)

•删除

ALTER TABLE table\_name DROP partition\_spec, partition\_spec,...

**重命名表**

•ALTER TABLE table\_name RENAME TO new\_table\_name

**修改列的名字、类型、位置、注释**

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

•这个命令可以允许改变列名、数据类型、注释、列位置或者它们的任意组合

**增加/更新列**

•ALTER TABLE table\_name ADD|REPLACE COLUMNS (col\_name data\_type [COMMENT col\_comment], ...)

• ADD是代表新增一字段，字段位置在所有列后面(partition列前)

     REPLACE则是表示替换表中所有字段。

**增加表的元数据信息**

•ALTER TABLE table\_name SET TBLPROPERTIES table\_properties table\_properties:

         :[property\_name = property\_value…..]

•用户可以用这个命令向表中增加metadata

**改变表文件格式与组织**

•ALTER TABLE table\_name SET FILEFORMAT file\_format

•ALTER TABLE table\_name CLUSTERED BY(userid) SORTED BY(viewTime) INTO num\_buckets BUCKETS

•这个命令修改了表的物理存储属性

### 创建／删除视图

•CREATE VIEW [IF NOT EXISTS] view\_name [ (column\_name [COMMENT column\_comment], ...) ][COMMENT view\_comment][TBLPROPERTIES (property\_name = property\_value, ...)] AS SELECT

•增加视图

•如果没有提供表名，视图列的名字将由定义的SELECT表达式自动生成

•如果修改基本表的属性，视图中不会体现，无效查询将会失败

•视图是只读的，不能用LOAD/INSERT/ALTER

•DROP VIEW view\_name

•删除视图

### 创建数据库

•CREATE DATABASE name

### 显示命令

•show tables;

•show databases;

•show partitions ;

•show functions

•describe extended table\_name dot col\_name

## 第二部分：ＤＭＬ

**DML**

•向数据表内加载文件

•将查询结果插入到Hive表中

•0.8新特性 insert into

**向数据表内加载文件**

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

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

•filepath

•相对路径，例如：project/data1

•绝对路径，例如： /user/hive/project/data1

•包含模式的完整 URI，例如：hdfs://namenode:9000/user/hive/project/data1

**向数据表内加载文件**

•加载的目标可以是一个表或者分区。如果表包含分区，必须指定每一个分区的分区名

•filepath 可以引用一个文件（这种情况下，Hive 会将文件移动到表所对应的目录中）或者是一个目录（在这种情况下，Hive 会将目录中的所有文件移动至表所对应的目录中）

**LOCAL关键字**

•指定了LOCAL

•load 命令会去查找本地文件系统中的 filepath。如果发现是相对路径，则路径会被解释为相对于当前用户的当前路径。用户也可以为本地文件指定一个完整的 URI，比如：file:///user/hive/project/data1.

•load 命令会将 filepath 中的文件复制到目标文件系统中。目标文件系统由表的位置属性决定。被复制的数据文件移动到表的数据对应的位置

• 没有指定LOCAL

         如果 filepath 指向的是一个完整的 URI，hive 会直接使用这个 URI。 否则

•如果没有指定 schema 或者 authority，Hive 会使用在 hadoop 配置文件中定义的 schema 和 authority，fs.default.name 指定了 Namenode 的 URI

•如果路径不是绝对的，Hive 相对于 /user/ 进行解释。 Hive 会将 filepath 中指定的文件内容移动到 table （或者 partition）所指定的路径中

**OVERWRITE**

•指定了OVERWRITE

•目标表（或者分区）中的内容（如果有）会被删除，然后再将 filepath 指向的文件/目录中的内容添加到表/分区中。

•如果目标表（分区）已经有一个文件，并且文件名和 filepath 中的文件名冲突，那么现有的文件会被新文件所替代。

**将查询结果插入Hive表**

•将查询结果插入Hive表

•将查询结果写入HDFS文件系统

•基本模式

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

•多插入模式

 FROM from\_statement

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

[INSERT OVERWRITE TABLE tablename2 [PARTITION ...] select\_statement2] ...

•自动分区模式

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

**将查询结果写入HDFS文件系统**

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

        FROM from\_statement

        INSERT OVERWRITE [LOCAL] DIRECTORY directory1 select\_statement1

     [INSERT OVERWRITE [LOCAL] DIRECTORY directory2 select\_statement2]

•

•数据写入文件系统时进行文本序列化，且每列用^A 来区分，\n换行

**INSERT INTO**

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

## 第三部分：HiveQL 查询操作

**SQL操作**

•基本的Select 操作

•基于Partition的查询

•Join

**基本的Select 操作**

SELECT [ALL | DISTINCT] select\_expr, select\_expr, ...

FROM table\_reference

[WHERE where\_condition]

[GROUP BY col\_list [HAVING condition]]

[   CLUSTER BY col\_list

  | [DISTRIBUTE BY col\_list] [SORT BY| ORDER BY col\_list]

]

[LIMIT number]

•使用ALL和DISTINCT选项区分对重复记录的处理。默认是ALL，表示查询所有记录。DISTINCT表示去掉重复的记录

•

•Where 条件

•类似我们传统SQL的where 条件

•目前支持 AND,OR ,0.9版本支持between

•IN, NOT IN

•不支持EXIST ,NOT EXIST

**ORDER BY与SORT BY的不同**

•ORDER BY 全局排序，只有一个Reduce任务

•SORT BY 只在本机做排序

**Limit**

•Limit 可以限制查询的记录数

SELECT \* FROM t1 LIMIT 5

•实现Top k 查询

•下面的查询语句查询销售记录最大的 5 个销售代表。

SET mapred.reduce.tasks = 1   
  SELECT \* FROM test SORT BY amount DESC LIMIT 5

•REGEX Column Specification

SELECT 语句可以使用正则表达式做列选择，下面的语句查询除了 ds 和 hr 之外的所有列：

SELECT `(ds|hr)?+.+` FROM test

**基于Partition的查询**

•一般 SELECT 查询会扫描整个表，使用 PARTITIONED BY 子句建表，查询就可以利用分区剪枝（input pruning）的特性

•Hive 当前的实现是，只有分区断言出现在离 FROM 子句最近的那个WHERE 子句中，才会启用分区剪枝

**Join**

Syntax

join\_table:   
   table\_reference JOIN table\_factor [join\_condition]   
  | table\_reference {LEFT|RIGHT|FULL} [OUTER] JOIN table\_reference join\_condition   
  | table\_reference LEFT SEMI JOIN table\_reference join\_condition   
  
table\_reference:   
    table\_factor   
  | join\_table   
  
table\_factor:   
    tbl\_name [alias]   
  | table\_subquery alias   
  | ( table\_references )   
  
join\_condition:   
    ON equality\_expression ( AND equality\_expression )\*   
  
equality\_expression:   
    expression = expression

•Hive 只支持等值连接（equality joins）、外连接（outer joins）和（left semi joins）。Hive 不支持所有非等值的连接，因为非等值连接非常难转化到 map/reduce 任务

•LEFT，RIGHT和FULL OUTER关键字用于处理join中空记录的情况

•LEFT SEMI JOIN 是 IN/EXISTS 子查询的一种更高效的实现

•join 时，每次 map/reduce 任务的逻辑是这样的：reducer 会缓存 join 序列中除了最后一个表的所有表的记录，再通过最后一个表将结果序列化到文件系统

•实践中，应该把最大的那个表写在最后

**join 查询时，需要注意几个关键点**

•只支持等值join

•SELECT a.\* FROM a JOIN b ON (a.id = b.id)

•SELECT a.\* FROM a JOIN b   
    ON (a.id = b.id AND a.department = b.department)

•可以 join 多于 2 个表，例如

  SELECT a.val, b.val, c.val FROM a JOIN b   
    ON (a.key = b.key1) JOIN c ON (c.key = b.key2)

•如果join中多个表的 join key 是同一个，则 join 会被转化为单个 map/reduce 任务

**LEFT，RIGHT和FULL OUTER**

•例子

•SELECT a.val, b.val FROM a LEFT OUTER JOIN b ON (a.key=b.key)

•如果你想限制 join 的输出，应该在 WHERE 子句中写过滤条件——或是在 join 子句中写

•

•容易混淆的问题是表分区的情况

• SELECT c.val, d.val FROM c LEFT OUTER JOIN d ON (c.key=d.key)   
  WHERE a.ds='2010-07-07' AND b.ds='2010-07-07‘

•如果 d 表中找不到对应 c 表的记录，d 表的所有列都会列出 NULL，包括 ds 列。也就是说，join 会过滤 d 表中不能找到匹配 c 表 join key 的所有记录。这样的话，LEFT OUTER 就使得查询结果与 WHERE 子句无关

•解决办法

•SELECT c.val, d.val FROM c LEFT OUTER JOIN d   
  ON (c.key=d.key AND d.ds='2009-07-07' AND c.ds='2009-07-07')

**LEFT SEMI JOIN**

•LEFT SEMI JOIN 的限制是， JOIN 子句中右边的表只能在 ON 子句中设置过滤条件，在 WHERE 子句、SELECT 子句或其他地方过滤都不行

•

•SELECT a.key, a.value   
  FROM a   
  WHERE a.key in   
   (SELECT b.key   
    FROM B);

       可以被重写为：

      SELECT a.key, a.val   
   FROM a LEFT SEMI JOIN b on (a.key = b.key)

**UNION ALL**

•用来合并多个select的查询结果，需要保证select中字段须一致

•select\_statement UNION ALL select\_statement UNION ALL select\_statement ...

## 第四部分：从SQL到HiveQL应该转变的几个习惯

**Hive不支持等值连接**

•SQL中对两表内联可以写成：

•select \* from dual a,dual b where a.key = b.key;

•Hive中应为

•select \* from dual a join dual b on a.key = b.key;

**分号字符**

•分号是SQL语句结束标记，在HiveQL中也是，但是在HiveQL中，对分号的识别没有那么智慧，例如：

•select concat(key,concat(';',key)) from dual;

•但HiveQL在解析语句时提示：

        FAILED: Parse Error: line 0:-1 mismatched input '<EOF>' expecting ) in function specification

•解决的办法是，使用分号的八进制的ASCII码进行转义，那么上述语句应写成：

•select concat(key,concat('\073',key)) from dual;

**IS [NOT] NULL**

•SQL中null代表空值, 值得警惕的是, 在HiveQL中String类型的字段若是空(empty)字符串, 即长度为0, 那么对它进行IS NULL的判断结果是False.

# Hive Shell 基本操作

**第一部分：Hive bin下脚本介绍**

**Hive bin下的脚本介绍**

Ext

hive

hive-config

**第二部分：Hive Shell 基本操作**

**Hive 命令行**

•hive [-hiveconf x=y]\* [<-i filename>]\* [<-f filename>|<-e query-string>] [-S]

•-i   从文件初始化HQL

•-e   从命令行执行指定的HQL

• -f   执行HQL脚本

•-v   输出执行的HQL语句到控制台

•-p <port> connect to Hive Server on port number -hiveconf x=y Use this to set hive/hadoop configuration variables.

**Hive 命令行示例**

•从命令行执行指定的sql语句

•$HIVE\_HOME/bin/hive -e 'select a.col from tab1 a'

•以指定的hive环境变量执行指定的sql语句

•$HIVE\_HOME/bin/hive -e 'select a.col from tab1 a' -hiveconf hive.exec.scratchdir=/home/my/hive\_scratch -hiveconf mapred.reduce.tasks=32

•以沉默模式执行指定的sql语句，并将执行结果导出到指定文件 HIVE\_HOME/bin/hive  -e 'select a.col from tab1 a' > a.txt

•以非交互式模式执行sql文件

•HIVE\_HOME/bin/hive -f /home/my/hive-script.sql

•在进入交互模式之前，执行初始化sql文件

•HIVE\_HOME/bin/hive -i /home/my/hive-init.sql

**Hive 交互式Shell命令**

• 当命令 $HIVE\_HOME/bin/hive以不带 -e/-f 选项的方式运行时， hive将进入到交互模式

•以（;）冒号结束命令行

|  |  |  |  |
| --- | --- | --- | --- |
| **\*Command \*** | | **Description** | |
| quit | | 退出命令行 | |
| set <key>=<value> | | 设置参数 | |
| set -v | | 打印出所有Hive支持的命令 | |
| **\*Command \*** | **Description** | |
| add FILE <value> <value>\* | 增加一个文件到资源列表. | |
| list FILE | 列出所有已经添加的资源 | |
| list FILE <value>\* | 根据value来查看添加的资源 | |
| ! <cmd> | execute a shell command from hive shell | |
| dfs <dfs command> | 执行DFS的命令 | |
| <query string> | 执行查询并输出到标准输出 | |

**第三部分：日志**

**日志**

•Hive使用Log4J来处理日志

•我们可以通过下面的命令设计Hive的日志级别

•$HIVE\_HOME/bin/hive -hiveconf hive.root.logger=INFO,console

•hive.root.logger的有INFO,DEBUG, 等

**第四部分：资源**

**Hive添加资源**

•Hive可以动态的添加资源，如文件

•一般情况下，我们是在与Hive进行交互时添加文件

•实际上是使用Hadoop的 Distributed Cache来控制的

**例子**

•ADD { FILE[S] | JAR[S] | ARCHIVE[S] } <filepath1> [<filepath2>]\*

•LIST { FILE[S] | JAR[S] | ARCHIVE[S] } [<filepath1> <filepath2> ..]

• DELETE { FILE[S] | JAR[S] | ARCHIVE[S] } [<filepath1> <filepath2> ..]

# Hive优化

**第一部分：Hadoop 计算框架的特性**

**什么是数据倾斜**

•由于数据的不均衡原因，导致数据分布不均匀，造成数据大量的集中到一点，造成数据热点

**Hadoop框架的特性**

•不怕数据大，怕数据倾斜

•jobs数比较多的作业运行效率相对比较低，比如即使有几百行的表，如果多次关联多次汇总，产生十几个jobs，耗时很长。原因是map reduce作业初始化的时间是比较长的

•sum,count,max,min等UDAF，不怕数据倾斜问题,hadoop在map端的汇总合并优化，使数据倾斜不成问题

•count(distinct ),在数据量大的情况下，效率较低，因为count(distinct)是按group by 字段分组，按distinct字段排序，一般这种分布方式是很倾斜的

**第二部分：优化的常用手段**

**优化的常用手段**

•解决数据倾斜问题

•减少job数

•设置合理的map reduce的task数，能有效提升性能。

•了解数据分布，自己动手解决数据倾斜问题是个不错的选择

•数据量较大的情况下，慎用count(distinct)。

•对小文件进行合并，是行至有效的提高调度效率的方法。

•优化时把握整体，单个作业最优不如整体最优。

**第三部分：Hive的数据类型方面的优化**

**优化原则**

•按照一定规则分区（例如根据日期）。通过分区，查询的时候指定分区，会大大减少在无用数据上的扫描, 同时也非常方便数据清理。

•合理的设置Buckets。在一些大数据join的情况下，map join有时候会内存不够。如果使用Bucket Map Join的话，可以只把其中的一个bucket放到内存中，内存中原来放不下的内存表就变得可以放下。这需要使用buckets的键进行join的条件连结，并且需要如下设置

     set hive.optimize.bucketmapjoin = true

**第四部分：Hive的操作方面的优化**

•全排序

•怎样做笛卡尔积

•怎样决定map个数

•怎样决定reducer个数

•合并MapReduce操作

•Bucket 与 sampling

•Partition

•JOIN

•Group By

•合并小文件

**全排序**

•Hive的排序关键字是SORT BY，它有意区别于传统数据库的ORDER BY也是为了强调两者的区别–SORT BY只能在单机范围内排序

**怎样做笛卡尔积**

•当Hive设定为严格模式（hive.mapred.mode=strict）时，不允许在HQL语句中出现笛卡尔积

•MapJoin是的解决办法

•MapJoin，顾名思义，会在Map端完成Join操作。这需要将Join操作的一个或多个表完全读入内存

•MapJoin的用法是在查询/子查询的SELECT关键字后面添加/\*+ MAPJOIN(tablelist) \*/提示优化器转化为MapJoin（目前Hive的优化器不能自动优化MapJoin）

•其中tablelist可以是一个表，或以逗号连接的表的列表。tablelist中的表将会读入内存，应该将小表写在这里

•在大表和小表做笛卡尔积时，规避笛卡尔积的方法是，给Join添加一个Join key，原理很简单：将小表扩充一列join key，并将小表的条目复制数倍，join key各不相同；将大表扩充一列join key为随机数

**控制Hive的Map数**

•通常情况下，作业会通过input的目录产生一个或者多个map任务

•主要的决定因素有： input的文件总个数，input的文件大小，集群设置的文件块大小(目前为128M, 可在hive中通过set dfs.block.size;命令查看到，该参数不能自定义修改)

•是不是map数越多越好

答案是否定的。如果一个任务有很多小文件（远远小于块大小128m）,则每个小文件也会被当做一个块，用一个map任务来完成，   
而一个map任务启动和初始化的时间远远大于逻辑处理的时间，就会造成很大的资源浪费。   
而且，同时可执行的map数是受限的

•是不是保证每个map处理接近128m的文件块，就高枕无忧了？

     答案也是不一定。比如有一个127m的文件，正常会用一个map去完成，但这个文件只有一个或者两个小字段，却有几千万的记录，   
如果map处理的逻辑比较复杂，用一个map任务去做，肯定也比较耗时。   
  
针对上面的问题3和4，我们需要采取两种方式来解决：即减少map数和增加map数；

•是不是保证每个map处理接近128m的文件块，就高枕无忧了？

     答案也是不一定。比如有一个127m的文件，正常会用一个map去完成，但这个文件只有一个或者两个小字段，却有几千万的记录，   
如果map处理的逻辑比较复杂，用一个map任务去做，肯定也比较耗时。   
  
针对上面的问题3和4，我们需要采取两种方式来解决：即减少map数和增加map数；

•举例

    a) 假设input目录下有1个文件a,大小为780M,那么hadoop会将该文件a分隔成7个块（6个128m的块和1个12m的块），从而产生7个map数   
b)    假设input目录下有3个文件a,b,c,大小分别为10m，20m，130m，那么hadoop会分隔成4个块（10m,20m,128m,2m）,从而产生4个map数   
即，如果文件大于块大小(128m),那么会拆分，如果小于块大小，则把该文件当成一个块

**怎样决定reducer个数**

•Hadoop MapReduce程序中，reducer个数的设定极大影响执行效率

•不指定reducer个数的情况下，Hive会猜测确定一个reducer个数，基于以下两个设定：

     参数1：hive.exec.reducers.bytes.per.reducer（默认为1G)

    参数2 ：hive.exec.reducers.max（默认为999）

•计算reducer数的公式

•N=min(参数2，总输入数据量/参数1)

•依据Hadoop的经验，可以将参数2设定为0.95\*(集群中TaskTracker个数)

•reduce个数并不是越多越好

同map一样，启动和初始化reduce也会消耗时间和资源；   
另外，有多少个reduce,就会有多少个输出文件，如果生成了很多个小文件，那么如果这些小文件作为下一个任务的输入，则也会出现小文件过多的问题

•什么情况下只有一个reduce

           很多时候你会发现任务中不管数据量多大，不管你有没有设置调整reduce个数的参数，任务中一直都只有一个reduce任务；   
其实只有一个reduce任务的情况，除了数据量小于

      hive.exec.reducers.bytes.per.reducer参数值的情况外，还有以下原因：   
a)    没有group by的汇总   
b)    用了Order by

**合并** **MapReduce** **操作**

• **Multi-group by**

•Multi-group by是Hive的一个非常好的特性，它使得Hive中利用中间结果变得非常方便

•FROM log

•  insert overwrite table test1 select log.id group by log.id

•   insert  overwrite table test2 select log.name group by log.name

• 上述查询语句使用了Multi-group by特性连续group by了2次数据，使用不同的group by key。这一特性可以减少一次MapReduce操作。

**Bucket 与 Sampling**

•Bucket是指将数据以指定列的值为key进行hash，hash到指定数目的桶中。这样就可以支持高效采样了

•Sampling可以在全体数据上进行采样，这样效率自然就低，它还是要去访问所有数据。而如果一个表已经对某一列制作了bucket，就可以采样所有桶中指定序号的某个桶，这就减少了访问量。

•如下例所示就是采样了test中32个桶中的第三个桶。

•SELECT \* FROM test 、、、TABLESAMPLE(BUCKET 3 OUT OF 32);

**JOIN** **原则**

•在使用写有 Join 操作的查询语句时有一条原则：应该将条目少的表/子查询放在 Join 操作符的左边

•原因是在 Join 操作的 Reduce 阶段，位于 Join 操作符左边的表的内容会被加载进内存，将条目少的表放在左边，可以有效减少发生 OOM 错误的几率

**Map Join**

•Join 操作在 Map 阶段完成，不再需要Reduce，前提条件是需要的数据在 Map 的过程中可以访问到

•例如：

•INSERT OVERWRITE TABLE phone\_traffic

SELECT /\*+ MAPJOIN(phone\_location) \*/  l.phone,p.location,l.traffic from phone\_location p join log l on (p.phone=l.phone)

•相关的参数为：

**hive.join.emit.interval = 1000** How many rows in the right-most join operand Hive should buffer before emitting the join result.

**hive.mapjoin.size.key = 10000**

**hive.mapjoin.cache.numrows = 10000**

**Group By**

•Map 端部分聚合

•并不是所有的聚合操作都需要在 Reduce 端完成，很多聚合操作都可以先在 Map 端进行部分聚合，最后在 Reduce 端得出最终结果

• 基于 Hash

• 参数包括：

•hive.map.aggr = true 是否在 Map 端进行聚合，默认为 True

•hive.groupby.mapaggr.checkinterval = 100000 在 Map 端进行聚合操作的条目数目

•有数据倾斜的时候进行负载均衡

•hive.groupby.skewindata = false

•当选项设定为 true，生成的查询计划会有两个 MR Job。第一个 MR Job 中，Map 的输出结果集合会随机分布到 Reduce 中，每个 Reduce 做部分聚合操作，并输出结果，这样处理的结果是相同的 Group By Key 有可能被分发到不同的 Reduce 中，从而达到负载均衡的目的；第二个 MR Job 再根据预处理的数据结果按照 Group By Key 分布到 Reduce 中（这个过程可以保证相同的 Group By Key 被分布到同一个 Reduce 中），最后完成最终的聚合操作。

**合并小文件**

•文件数目过多，会给 HDFS 带来压力，并且会影响处理效率，可以通过合并 Map 和 Reduce 的结果文件来消除这样的影响：

•hive.merge.mapfiles = true 是否和并 Map 输出文件，默认为 True

•hive.merge.mapredfiles = false 是否合并 Reduce 输出文件，默认为 False

•hive.merge.size.per.task = 256\*1000\*1000 合并文件的大小

# Hive体系结构

**第一部分：概念**

**概念**

•用户接口：用户访问Hive的入口

•元数据：Hive的用户信息与表的MetaData

•解释器：分析翻译HQL的组件

•编译器：编译HQL的组件

•优化器：优化HQL的组件

**第二部分：Hive架构与基本组成**

**架构图**

**基本组成**

•用户接口，包括 CLI，JDBC/ODBC，WebUI

•元数据存储，通常是存储在关系数据库如 mysql, derby 中

•解释器、编译器、优化器、执行器

•Hadoop：用 HDFS 进行存储，利用 MapReduce 进行计算

**各组件的基本功能**

•用户接口主要有三个：CLI，JDBC/ODBC和 WebUI

•CLI，即Shell命令行

•JDBC/ODBC 是 Hive 的JAVA，与使用传统数据库JDBC的方式类似

•WebGUI是通过浏览器访问 Hive

•Hive 将元数据存储在数据库中，目前只支持 mysql、derby，下一版本会支持更多的数据库。Hive 中的元数据包括表的名字，表的列和分区及其属性，表的属性（是否为外部表等），表的数据所在目录等

•解释器、编译器、优化器完成 HQL 查询语句从词法分析、语法分析、编译、优化以及查询计划的生成。生成的查询计划存储在 HDFS 中，并在随后有 MapReduce 调用执行

•Hive 的数据存储在 HDFS 中，大部分的查询由 MapReduce 完成（包含 \* 的查询，比如 select \* from table 不会生成 MapRedcue 任务）

**Metastore**

•Metastore是系统目录(catalog)用于保存Hive中所存储的表的元数据（metadata）信息

•Metastore是Hive被用作传统数据库解决方案（如oracle和db2）时区别其它类似系统的一个特征

•Metastore包含如下的部分：

•Database 是表（table）的名字空间。默认的数据库（database）名为‘default’

•Table 表（table）的原数据包含信息有：列（list of columns）和它们的类型（types），拥有者（owner），存储空间（storage）和SerDei信息

•Partition 每个分区（partition）都有自己的列（columns），SerDe和存储空间（storage）。这一特征将被用来支持Hive中的模式演变（schema evolution）

**Compiler**

•Driver调用编译器（compiler）处理HiveQL字串，这些字串可能是一条DDL、DML或查询语句

•编译器将字符串转化为策略（plan）

•策略仅由元数据操作和HDFS操作组成，元数据操作只包含DDL语句，HDFS操作只包含LOAD语句

•对插入和查询而言，策略由map-reduce任务中的具有方向的非循环图（directedacyclic graph，DAG）组成

**第三部分：Hive运行模式**

**Hive运行模式**

•Hive的运行模式即任务的执行环境

•分为本地与集群两种

•我们可以通过mapred.job.tracker 来指明

•设置方式

•hive > SET  mapred.job.tracker=local

**第四部分：数据类型**

**原始数据类型**

•Integers

TINYINT - 1 byte

SMALLINT - 2 byte

INT - 4 byte

BIGINT - 8 byte

•Boolean type

BOOLEAN - TRUE/FALSE

•Floating point numbers

FLOAT –单精度

DOUBLE – 双精度

•String type

STRING - sequence of characters in a specified character set

**复杂数据类型**

•Structs: 例子  {c INT; d INT}

•Maps (key-value tuples):. 例子'group' -> gid  M['group']

•Arrays (indexable lists):  例子[‘1', ‘2', ‘3']

•TIMESTAMP  0.8版本新加属性

**第五部分：Hive的元数据存储**

**存储方式与模式**

•Hive 将元数据存储在 数据库中

•连接到数据库模式有三种

•单用户模式

•多用户模式

•远程服务器模式

**单用户模式**

此模式连接到一个 In-memory 的数据库 Derby ，一般用于 Unit Test

**多用户模式**

通过网络连接到一个数据库中，是最经常使用到的模式

**远程服务器模式**

•用于非 Java 客户端访问元数据库，在服务器端启动MetaStoreServer，客户端利用 Thrift 协议通过MetaStoreServer 访问元数据库

**第六部分：Hive的数据存储**

**Hive数据存储的基本概念**

•Hive的数据存储是建立在Hadoop HDFS之上的

•Hive没有专门的数据存储格式

•存储结构主要包括：数据库、文件、表、视图

•Hive默认可以直接加载文本文件，还支持sequence file 、RCFile

•创建表时，我们直接告诉Hive数据的列分隔符与行分隔符，Hive即可解析数据

**Hive的数据模型-数据库**

•类似传统数据库的DataBase

•在第三方数据库里实际是一张表

•简单示例

•命令行hive > create database test\_database;

**Hive的数据模型-表**

•Table 内部表

•Partition  分区表

•External Table 外部表

•Bucket  Table

**内部表**

•与数据库中的 Table 在概念上是类似

•每一个 Table 在 Hive 中都有一个相应的目录存储数据

•例如，一个表 test，它在 HDFS 中的路径为：/ warehouse /test

• warehouse是在 hive-site.xml 中由 ${hive.metastore.warehouse.dir} 指定的数据仓库的目录

•所有的 Table 数据（不包括 External Table）都保存在这个目录中。

•删除表时，元数据与数据都会被删除

**内部表简单示例**

•创建数据文件test\_inner\_table.txt

•创建表

•create table test\_inner\_table (key string)

•加载数据

•LOAD DATA LOCAL INPATH ‘filepath’ INTO TABLE test\_inner\_table

•查看数据

•select \* from test\_inner\_table

•select count(\*) from test\_inner\_table

•删除表 drop table test\_inner\_table

**分区表**

•Partition 对应于数据库中的 Partition 列的密集索引

•在 Hive 中，表中的一个 Partition 对应于表下的一个目录，所有的 Partition 的数据都存储在对应的目录中

•例如：test表中包含 date 和 position 两个 Partition，则对应于 date = 20120801, position = zh 的 HDFS 子目录为：/ warehouse /test/date=20120801/ position =zh

•对应于  = 20100801, position = US 的HDFS 子目录为；/ warehouse /xiaojun/date=20120801/ position =US

**分区表简单示例**

•创建数据文件test\_partition\_table.txt

•创建表

•create table test\_partition\_table (key string) partitioned by (dt string)

•加载数据

•LOAD DATA INPATH ‘filepath’ INTO TABLE test\_partition\_table partition (dt=‘2006’)

•查看数据

•select \* from test\_partition\_table

•select count(\*) from test\_partition\_table

•删除表 drop table test\_partition\_table

**外部表**

•指向已经在 HDFS 中存在的数据，可以创建 Partition

•它和 内部表 在元数据的组织上是相同的，而实际数据的存储则有较大的差异

•内部表 的创建过程和数据加载过程（这两个过程可以在同一个语句中完成），在加载数据的过程中，实际数据会被移动到数据仓库目录中；之后对数据对访问将会直接在数据仓库目录中完成。删除表时，表中的数据和元数据将会被同时删除

• 外部表 只有一个过程，加载数据和创建表同时完成，并不会移动到数据仓库目录中，只是与外部数据建立一个链接。当删除一个 外部表 时，仅删除该链接

**外部表简单示例**

•创建数据文件test\_external\_table.txt

•创建表

•create external table test\_external\_table (key string)

•加载数据

•LOAD DATA INPATH ‘filepath’ INTO TABLE test\_inner\_table

•查看数据

•select \* from test\_external\_table

•select count(\*) from test\_external\_table

•删除表 drop table test\_external\_table

**Bucket Table**

•可以将表的列通过Hash算法进一步分解成不同的文件存储

•例如：将age列分散成20个文件，首先要对AGE进行Hash计算，对应为0的写入/warehouse/test/date=20120801/postion=zh/part-00000,对应为1的写入/warehouse/test/date=20120801/postion=zh/part-00001

•如果想应用很多的Map任务这样是不错的选择

**Bucket Table简单示例**

•创建数据文件test\_bucket\_table.txt

•创建表

•create table test\_bucket\_table (key string)

     clustered by (key) into 20 buckets

•加载数据

•LOAD DATA INPATH ‘filepath’ INTO TABLE test\_bucket\_table

•查看数据

•select \* from test\_bucket\_table

•set hive.enforce.bucketing = true;

**Hive的数据模型-视图**

•视图与传统数据库的视图类似

•视图是只读的

•视图基于的基本表，如果改变，指增加不会影响视图的呈现；如果删除，会出现问题

•如果不指定视图的列，会根据select语句后的生成

•示例

•create view test\_view as select \* from test

**第七部分：HiveUI介绍**

**启动UI**

•配置

•hive-site.xml 添加

  <property>

       <name>hive.hwi.war.file</name>

       <value>lib/hive-hwi-0.8.1.war</value>

  </property>

•

•启动Hive的UI sh $HIVE\_HOME/bin/hive --service hwi

# Hive的原理

**第一部分：Hive原理**

**为什么要学习Hive的原理**

•一条Hive HQL将转换为多少道MR作业

•怎么样加快Hive的执行速度

•编写Hive HQL的时候我们可以做什么

•Hive 怎么将HQL转换为MR作业

•Hive会采用什么样的优化方式

**Hive架构&执行流程**

**Hive执行流程**

•编译器将一个Hive QL转换操作符

•操作符是Hive的最小的处理单元

•每个操作符代表HDFS的一个操作或者一道MapReduce作业

**Operator**

•Operator都是hive定义的一个处理过程

•Operator都定义有:

•protected List **<Operator<?**  **extends** Serializable **>>** childOperators;

•protected List **<Operator<?**  **extends** Serializable **>>** parentOperators;

•protected boolean done; // 初始化值为false

•所有的操作构成了 Operator图，hive正是基于这些图关系来处理诸如limit, group by, join等操作

**Hive执行流程**

|  |  |
| --- | --- |
| 操作符 | 描述 |
| TableScanOperator | 扫描hive表数据 |
| ReduceSinkOperator | 创建将发送到Reducer端的<Key,Value>对 |
| JoinOperator | Join两份数据 |
| SelectOperator | 选择输出列 |
| FileSinkOperator | 建立结果数据,输出至文件 |
| FilterOperator | 过滤输入数据 |
| GroupByOperator | GroupBy语句 |
| MapJoinOperator | /\*+mapjoin(t) \*/ |
| LimitOperator | Limit语句 |
| UnionOperator | Union语句 |

•Hive通过ExecMapper和ExecReducer执行MapReduce任务

•在执行MapReduce时有两种模式

•本地模式

•分布式模式

**ANTLR词法语法分析工具**

•ANTLR—Another Tool for Language Recognition

•ANTLR 是开源的

•为包括Java，C++，C#在内的语言提供了一个通过语法描述来自动构造自定义语言的识别器（recognizer），编译器（parser）和解释器（translator）的框架

•Hibernate就是使用了该分析工具

**Hive编译器**

**编译流程**

**第二部分：一条HQL引发的思考**

**案例HQL**

•select key from test\_limit **limit** 1

•Stage-1

•TableScan Operator>Select Operator-> Limit->File Output Operator

•Stage-0

•Fetch Operator

•读取文件

**Mapper与InputFormat**

•该hive MR作业中指定的mapper是:

•mapred.mapper.class = org.apache.hadoop.hive.ql.exec.ExecMapper

•input format是:

•hive.input.format  = org.apache.hadoop.hive.ql.io.CombineHiveInputFormat

•该hive MR作业中指定的mapper是:

•mapred.mapper.class = org.apache.hadoop.hive.ql.exec.ExecMapper

•input format是:

•hive.input.format  = org.apache.hadoop.hive.ql.io.CombineHiveInputFormat