Hive 简明教程

大数据技术系列

作者：淳月宾

QQ：522383832

欢迎点击这里的链接进入精彩的Linux公社 网站

Linux公社（www.Linuxidc.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

Linux公社是专业的Linux系统门户网站，实时发布最新Linux资讯，包括Linux、Ubuntu、Fedora、RedHat、红旗Linux、Linux教程、Linux认证、SUSE Linux、Android、Oracle、Hadoop、CentOS、MySQL、Apache、Nginx、Tomcat、Python、Java、C语言、OpenStack、集群等技术。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

Linux公社 主站网址：www.linuxidc.com 旗下网站：www.linuxidc.net

包括：**Ubuntu 专题** **Fedora 专题** **Android 专题** **Oracle 专题** **Hadoop 专题** **RedHat 专题** **SUSE 专题** **红旗 Linux 专题** **CentOS 专题**

Linux 公社微信公众号：linuxidc_com

目录

前言

Hive 是对于数据仓库进行管理和分析的工具。但是不要被“数据仓库”这个词所吓倒，数据仓库是很复杂的东西，但是如果你会 SQL，就会发现 Hive 是那么的简单，简单到甚至不用学就可以使用 Hive 做出业务需求所需要的东西。

但是 Hive 和 SQL 毕竟不同，执行原理、优化方法，底层架构都完全不相同。

大数据离线分析使用 Hive 已经成为主流，但是目前市面上 Hive 相关的中文书籍只有一本《Hive 编程指南》，对于不懂技术的数据分析人员来说，这本书有些繁琐和深奥；对于 Hive 技术人员来说，这本书对于原理和细节描述的又显得浅显和不足。

基于工作中的 Hive 使用情况，我整理了这个实用性的教程，这个教程分为三个部分：Hive 基本使用、Hive 执行原理与优化、Hive 高级知识。由浅入深地简单介绍 Hive 技术。

第一部分：完全以日常使用为目标，整理了常用的 Hive 语法，而抛弃了不常用的部分，用来满足不懂技术的分析人员来快速使用 Hive 进行常见的日常数据分析。

第二部分：如果想能写出高效的 Hive 语句，必须要先了解 Hive 执行原理，然后掌握一系列的优化方法。所以第二部分主要内容是 Hive 原理与优化。

第三部分：讲解 Hive 的一系列技术细节，以满足技术人员想了解技术细节的要求，为能更加高效和灵活地使用 Hive 提供技术基础。

三个部分的详细说明：

模块	主要内容	预期目标	面对人群
Hive 基本使用	Hive 概念、常用语法、内置函数、自定义函数	可以使用 Hive 做常见的大数据分析工作	能统计出数即可的分析人员
Hive 执行原理与优化	Hive 技术架构、执行原理、优化方法	了解 Hive 执行原理，可以写出性能比较好的 Hive 程序	对性能有要求的分析人员
Hive 高级知识	Hive 技术细节	了解 Hive 技术细节	对 Hive 原理感兴趣的人员

参考资料：

1、Hive 编程指南》 Eduard Capriolo、Dean Wampler、Jason Rutberglen 著　曹坤 译

2、Hive 官方文档：https://cwiki.apache.org/confluence/display/Hive/GettingStarted

3、互联网上其他资源

第一部分：Hive 基本使用

一、 Hive 简介

1、 定义

facebook 为了解决海量日志数据的分析而开发了 Hive,后来开源给了 Apache 基金会组织。

官网上的定义：

The Apache Hive ™ data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage using SQL.

Hive 是一种用 SQL 语句来协助读写、管理存储在分布式存储系统上的大数据集的数据仓库软件。

2、 Hive 的几个特点

(1) Hive 最大的特点是 Hive 通过类 SQL 来分析大数据，而避免了写 MapReduce Java 程序来分析数据，这样使得分析数据更容易。

(2) 数据是存储在 HDFS 上的，Hive 本身并不提供数据的存储功能

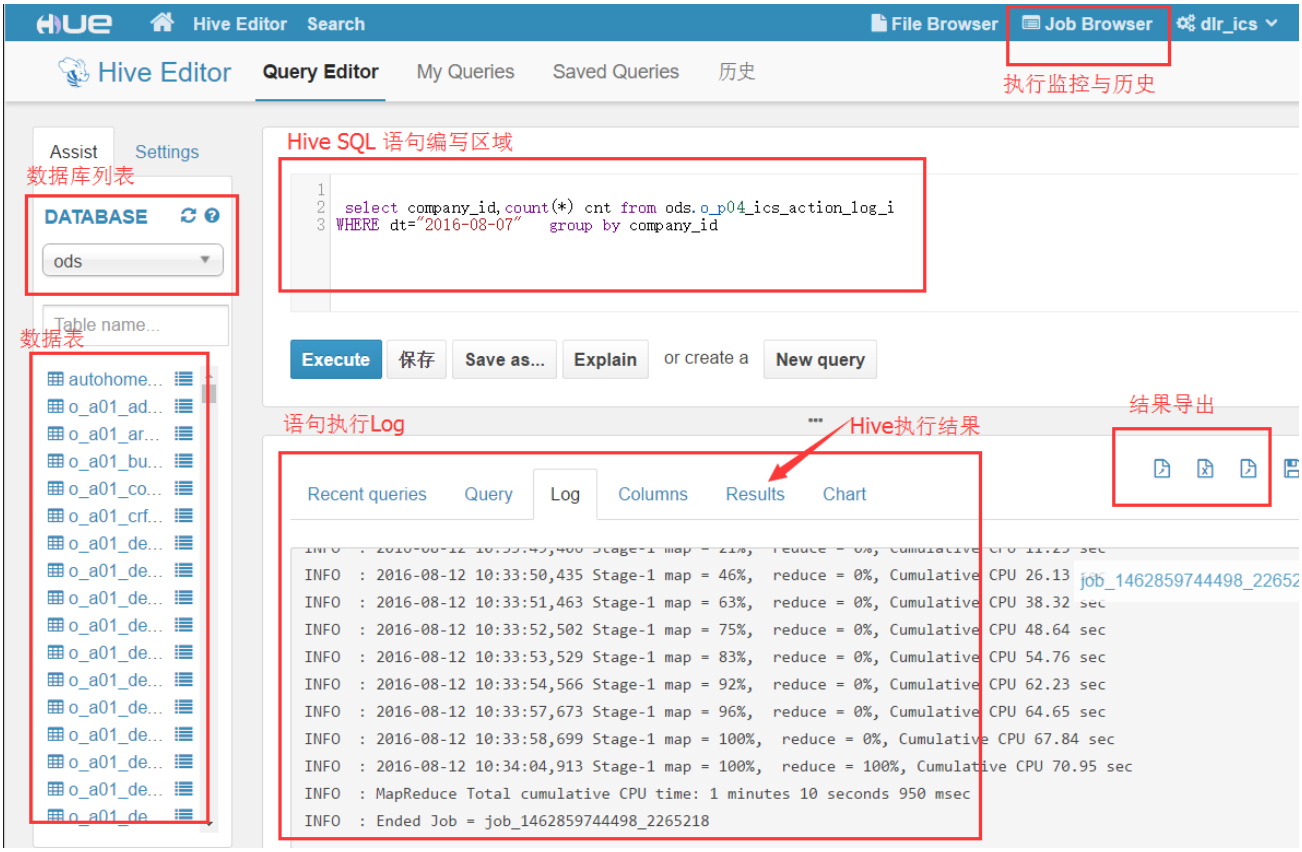
(3) Hive 是将数据映射成数据库和一张张的表，库和表的元数据信息一般存在关系型数据库上（比如 MySQL）。

(4) 数据存储方面：他能够存储很大的数据集，并且对数据完整性、格式要求并不严格。

(5) 数据处理方面：不适用于实时计算和响应，使用于离线分析。

3、 Hive 使用

公司搭建的 HUE 平台地址：http://192.168.201.66:8888/



二、 Hive 中的基本数据类型

1、基本数据类型

Hive 支持关系型数据中大多数基本数据类型，同时 Hive 中也有特有的三种复杂类型。这一段文章只介绍 Hive 的基本数据类型，负载类型在第三部分高级知识中有讲解。

下面的表列出了 Hive 中的常用基本数据类型：

数据类型	长度	备注
Tinyint	1 字节的有符号整数	-128~127
SmallInt	1 个字节的有符号整数	-32768~32767
Int	4 个字节的有符号整数	-2147483648 ~ 2147483647
BigInt	8 个字节的有符号整数	9223372036854775808 ~ 9223372036854775807
Boolean	布尔类型，true 或者 false	true、false
Float	单精度浮点数	
Double	双精度浮点数	
String	字符串	
TimeStamp	整数	支持 Unix timestamp，可以达到纳秒精度

Binary	字节数组	
Date	日期	0000-01-01 ~ 9999-12-31，常用 String 代替

2、隐形类型转换

	void	boolean	tinyint	smallint	int	bigint	float
void to	true	true	true	true	true	true	true
boolean to	false	true	false	false	false	false	false
tinyint to	false	false	true	true	true	true	true
smallint to	false	false	false	true	true	true	true
int to	false	false	false	false	true	true	true
bigint to	false	false	false	false	false	true	true
float to	false	false	false	false	false	false	true
double to	false	false	false	false	false	false	false
decimal to	false	false	false	false	false	false	false
string to	false	false	false	false	false	false	false
varchar to	false	false	false	false	false	false	false
timestamp to	false	false	false	false	false	false	false
date to	false	false	false	false	false	false	false
binary to	false	false	false	false	false	false	false

	double	decimal	string	varchar	timestamp	date	binary
void to	true	true	true	true	true	true	true
boolean to	false	false	false	false	false	false	false

欢迎点击这里的链接进入精彩的Linux公社 网站

Linux公社（www.Linuxidc.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

Linux公社是专业的Linux系统门户网站，实时发布最新Linux资讯，包括Linux、Ubuntu、Fedora、RedHat、红旗Linux、Linux教程、Linux认证、SUSE Linux、Android、Oracle、Hadoop、CentOS、MySQL、Apache、Nginx、Tomcat、Python、Java、C语言、OpenStack、集群等技术。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

Linux公社 主站网址：www.linuxidc.com 旗下网站：www.linuxidc.net

包括：**Ubuntu 专题** **Fedora 专题** **Android 专题** **Oracle 专题** **Hadoop 专题** **RedHat 专题** **SUSE 专题** **红旗 Linux 专题** **CentOS 专题**

Linux 公社微信公众号：linuxidc_com

	double	decimal	string	varchar	timestamp	date	binary
tinyint to	true	true	true	true	false	false	false
smallint to	true	true	true	true	false	false	false
int to	true	true	true	true	false	false	false
bigint to	true	true	true	true	false	false	false
float to	true	true	true	true	false	false	false
double to	true	true	true	true	false	false	false
decimal to	false	true	true	true	false	false	false
string to	true	true	true	true	false	false	false
varchar to	true	true	true	true	false	false	false
timestamp to	false	false	true	true	true	false	false
date to	false	false	true	true	false	true	false
binary to	false	false	false	false	false	false	true

三、 Hive DDL 数据定义语法

1、 创建数据库

创建一个数据库就在 HDFS 上创建一个目录，数据库类似命名空间来组织表，在大量 Hive 的情况下，避免表名冲突。默认的数据库是 default。

创建数据库

```
create database if not exists dealer_db;
```

2、 查看数据库定义

Describe 命令来查看数据库定义，包括：数据库名称、数据库在 HDFS 目录、HDFS

```
describe database dealer_db;
```

OK
dealer_db hdfs://bigdata-51cdh.chybinmy.com:8020/user/hive/warehouse/dealer_db.db
hadoop USER

dealer_db 是数据库名称

hdfs://bigdata-51cdh.chybinmy.com:8020/user/hive/warehouse/dealer_db.db 是 dealer_db 库对应的存储数据的 HDFS 上的目录

3、 查看数据库列表

show databases;

4、 删除数据库

删除数据库时，如果库中存在数据表，是不能删除的，要先删除所有表，再删除数据库。

添加上 cascade 后，就可以先自动删除所有表后，再删除数据库。

删除数据库后，HDFS 上数据库对应的目录就被删除掉了。

drop database if exists testdb cascade;

5、 切换当前数据库

use dealer_db;

6、 创建普通表

```
create table if not exists dealerinfo
(
  dealerid int,
```

```
    dealername string,
    cityid int,
    createtime date
)
row format delimited fields terminated by '\t'
stored as textfile;
```

以上例子是创建表的一种方式，如果表不存在，就创建表 dealerinfo。row format delimited fields terminated by ‘\t’ 是指定列之间的分隔符；stored as textfile 是指定文件存储格式为 textfile。（Hive 的文件格式会在第三部分详细介绍）

创建表一般有几种方式：

(1) create table 方式：以上例子中的方式。

(2) create table as select 方式：根据查询的结果自动创建表，并将查询结果数据插入新建的表中。

(3) create table like tablename1 方式：是克隆表，只复制 tablename1 表的结构

复制表和克隆表会在下面的 Hive 数据管理部分详细讲解。

7、 创建分区表

Hive 查询一般是扫描整个目录，但是有时候我们关心的数据只是集中在某一部分数据上，比如我们一个 Hive 查询，往往是只是查询某一天的数据，这样的情况下，可以使用分区表来优化，一天是一个分区，查询时候，Hive 只扫描指定天分区的数据。

普通表和分区表的区别在于：一个 Hive 表在 HDFS 上是有一个对应的目录来存储数据，普通表的数据直接存储在这个目录下，而分区表数据存储时，是再划分子目录来存储的。一个分区一个子目录。主要作用是来优化查询性能。

```
--创建经销商操作日志表
create table dealer_action_log
(
companyId INT comment    '公司 ID',
userid INT comment    '销售 ID',
originalstring STRING comment    'url',
host STRING comment    'host',
absolutepath STRING comment    '绝对路径',
```

```
query STRING comment    '参数串',
refurl STRING comment    '来源 url',
clientip STRING comment    '客户端 Ip',
cookiemd5 STRING comment    'cookiemd5',
timestamp STRING comment    '访问时间戳'
)
partitioned by (dt string)
row format delimited fields terminated by ','
stored as textfile;
```

这个例子中，这个日志表以 dt 字段分区，dt 是个虚拟的字段，dt 下并不存储数据，而是用来分区的，实际数据存储时，dt 字段值一样的数据存入同一个子目录中，插入数据或者导入数据时，同一天的数据 dt 字段赋值一样，这样就实现了数据按 dt 日期分区存储。

当 Hive 查询数据时，如果指定了 dt 筛选条件，那么只需要到对应的分区下去检索数据即可，大大提高了效率。**所以对于分区表查询时，尽量添加上分区字段的筛选条件。**

8、 创建桶表

(1) 桶表也是一种用于优化查询而设计的表类型。创建通表时，指定桶的个数、分桶的依据字段，hive 就可以自动将数据分桶存储。查询时只需要遍历一个桶里的数据，或者遍历部分桶，这样就提高了查询效率。

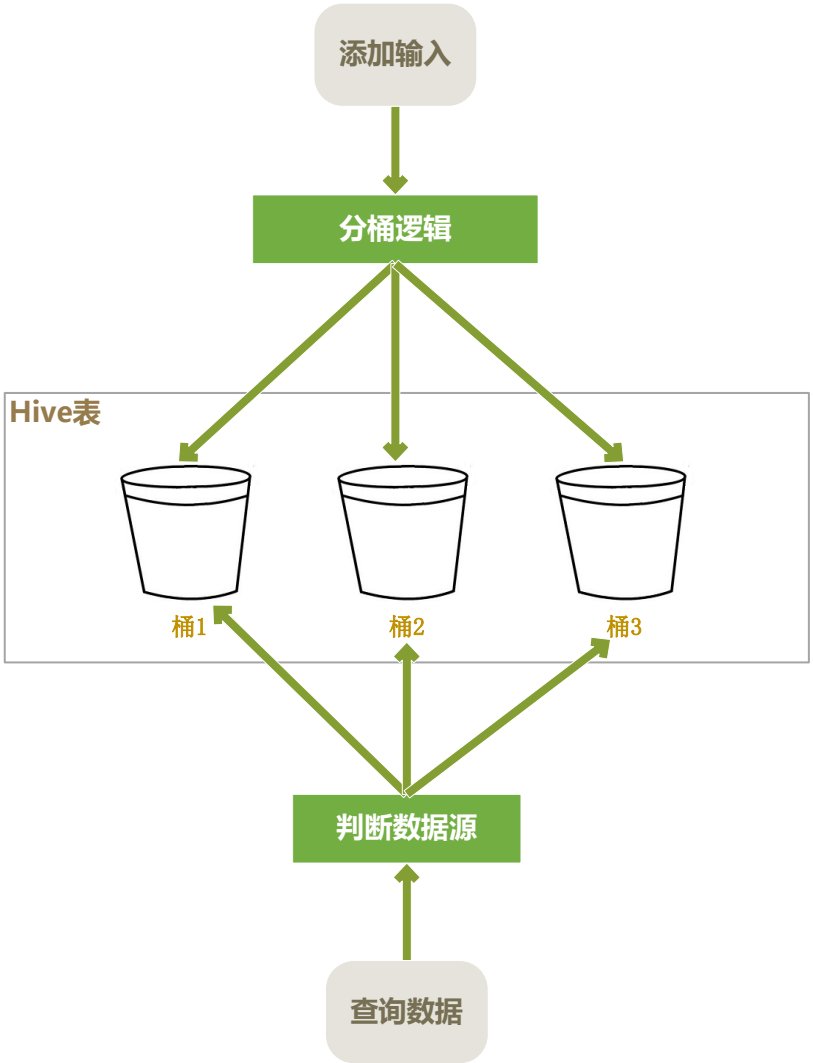
(2) 举例

```
--创建线索表
create table dealer_leads
(
leads_id string,
dealer_id string,
user_id string,
user_phone string,
user_name string,
create_time string
)
clustered by (dealer_id) sorted by(leads_id) into 10 buckets
row format delimited fields terminated by '\t'
stored as textfile;
```

(3) 说明

- clustered by 是指根据 dealer_id 的值进行哈希后模除分桶个数，根据得到的结果，确定这行数据分入哪个桶中，这样的分法，可以确保相同 dealer_id 的数据放入同一个桶中。而经销商的线索数据，大部分是根据 dealer_id 进行查询的。这样大部分情况下是只需要查询一个桶中的数据就可以了。
- sorted by 是指定桶中的数据以哪个字段进行排序，排序的好处是，在 join 操作时能获得很高的效率。
- into 10 buckets 是指定一共分多少个桶。
- 在 HDFS 上存储时，一个桶存入一个文件中，这样根据 dealer_id 进行查询时，可以快速确定数据存在于哪个桶中，而只遍历一个桶可以提供查询效率。

(4) 分桶表读写过程



9、 查看有哪些表

show tables;

Show TABLES '*info'; --可以用正则表达式筛选要列出的表

10、 查看表定义

(1) 查看简单定义

describe dealerinfo;

	col_name	data_type	comment
0	dealerid	int	
1	dealername	string	
2	cityid	int	
3	createtime	date	

(2) 查看表详细信息

describe formatted dealerinfo;

表的详细表定义信息：

备注	**col_name**	**data_type**	**comment**
列信息	# col_name	data_type	comment
		NULL	NULL
	dealerid	int	
	dealername	string	
	cityid	int	
	createtime	date	
		NULL	NULL
	# Detailed Table Information	NULL	NULL
所在库	Database:	dealer_db	NULL
所属 HUE 用户	Owner:	admin	NULL
表创建时间	CreateTime:	Tue Aug 16 06:05:14 PDT 2016	NULL
最后访问时间	LastAccessTime:	UNKNOWN	NULL
	Protect Mode:	None	NULL
	Retention:	0	NULL
表数据文件在 HDFS 上路径	Location:	hdfs://bigdata-51cdh.chybinmy.com:8020/user/hive/warehouse/dealer_db.db/dealerinfo	NULL
表类型(内部表或者外部表)	Table Type:	MANAGED_TABLE	NULL

表分区信息	Table Parameters:	NULL	NULL
		transient_lastDdlTime	1471352714
		NULL	NULL
	# Storage Information	NULL	NULL
序列化反序列化类	SerDe Library:	org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe	NULL
mapreduce 中的输入格式	InputFormat:	org.apache.hadoop.mapred.TextInputFormat	NULL
mapreduce 中的输出格式	OutputFormat:	org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat	NULL
压缩	Compressed:	No	NULL
总占用数据块个数	Num Buckets:	-1	NULL
	Bucket Columns:	[]	NULL
	Sort Columns:	[]	NULL
	Storage Desc Params:	NULL	NULL
		field.delim	\t
		serialization.format	\t

11、 修改表

对表的修改操作有：修改表名、添加字段、修改字段。

(1) 修改表名

```
--将表名从 dealerinfo 改为 dealer_info
alter table dealerinfo rename to dealer_info;
```

(2) 添加字段

```
--在 dealer_info 表添加一个字段 provinceid，int 类型
alter table dealer_info add columns (provinceid int );
```

(3) 修改字段

```
alter table dealer_info replace columns (dealerid int,dealername string,cityid int,joindate date,provinceid int);
```

修改字段，只是修改了 Hive 表的元数据信息（元数据信息一般是存储在 MySql 中），并不对存在于 HDFS 中的表数据做修改。

并不是所有的 Hive 表都可以修改字段，只有使用了 native SerDe (序列化反序列化类型)的表才能修改字段，可以修改的字段的 SerDe 有：DynamicSerDe,

MetadataTypedColumnsetSerDe, LazySimpleSerDe and ColumnarSerDe，关于 SerDe 会在下面的 第三部分 Hive 高级知识的 Hive 文件格式中详细介绍。

12、 删除表

```
--如果表存在，就删除。
drop table if exists dealer_info;
```

四、 Hive DML 数据管理语法

1、 向 Hive 中加载数据

(1) 加载到普通表

可以将本地文本文件内容批量加载到 Hive 表中，要求文本文件中的格式和 Hive 表的定义一致，包括：字段个数、字段顺序、列分隔符都要一致。

这里的 dealer_info 表的表定义是以\t 作为列分隔符，所以准备好数据后，将文本文件拷贝到 hive 客户端机器上后，执行加载命令。

load data local inpath '/home/hadoop/dealerinfodata.txt' overwrite into table dealer_info;

(1) local 关键字表示源数据文件在本地，源文件可以在 HDFS 上，如果在 HDFS 上，则去掉 local，inpath 后面的路径是类似”hdfs://namenode:9000/user/datapath”这样的 HDFS 上文件的路径。

(2) overwrite 关键字表示如果 hive 表中存在数据，就会覆盖掉原有的数据。如果省略 overwrite，则默认是追加数据。

加载完成数据后，在 HDFS 上就会看到加载的数据文件。

(2) 加载到分区表

```
load data local inpath '/home/hadoop/actionlog.txt' overwrite into table dealer_action_log
PARTITION (dt='2016-08-19');
```

partition 是指定这批数据放入分区 2016-08-19 中。

(3) 加载到分桶表

```
--先创建普通临时表
create table dealer_leads_tmp
(
```

```
leads_id string,
dealer_id string,
user_id string,
user_phone string,
user_name string,
create_time string
)
row format delimited fields terminated by ','
stored as textfile;

--数据载入临时表
load data local inpath '/home/hadoop/lead.txt' overwrite into table dealer_leads_tmp;

--导入分桶表
set hive.enforce.bucketing = true;
insert overwrite table dealer_leads select * from  dealer_leads_tmp;
```

set hive.enforce.bucketing = true; 这个配置非常关键，为 true 就是设置为启用分桶。

2、 导出数据

```
--导出数据，是将 hive 表中的数据导出到本地文件中。
insert overwrite local directory '/home/hadoop/dealer_info.bak2016-08-22 '
select * from dealer_info;
```

去掉 local 关键字，也可以到处到 HDFS 上。

3、 插入数据

(1) insert select 语句

上一节分桶表数据导入，用到从 dealer_leads_tmp 表向 dealer_leads 表中导入数据，用到了 insert 数据。

```
insert overwrite table dealer_leads select * from  dealer_leads_tmp;
```

这里是将查询结果导入到表中，overwrite 关键字是覆盖目标表中的原来数据。如果缺省，就是追加数据。

如果是插入数据的表是分区表，那么就如下所示：

```
insert overwrite table dealer_leads PARTITION (dt='2016-08-31') select * from dealer_leads_tmp;
```

(2) 一次遍历多次插入

```
from dealer_action_log
  insert overwrite table log1 select companyid,originalstring   where companyid='100006'
  insert overwrite table log2 select companyid,originalstring   where companyid='10002'
```

每次 hive 查询，都会将数据集整个遍历一遍。当查询结果会插入多个表中时，可以采用以上语法，将一次遍历写入多个表，以达到提高效率的目的。

4、 复制表

复制表是将源表的结构和数据复制并创建为一个新表，复制过程中，可以对数据进行筛选，列可以进行删减。

```
create table dealer_leads_bak
row format delimited fields terminated by '\t'
stored as textfile
as
select leads_id,dealer_id,'2016-08-22' as bakdate
from dealer_leads
where create_time<'2016-08-22';
```

上面这个例子是对 dealer_leads 表进行复制备份，复制时筛选了 2016-08-22 以前的数据，减少几个列，并添加了一个 bakdate 列。

5、 克隆表

```
--克隆表 dealer_leads，创建新表 dealer_leads_like
create table dealer_leads_like like   dealer_leads;
```

克隆表，会克隆源表的所有元数据，但是不会复制源表的数据。

6、 备份表

```
export table dealer_action_log partition (dt='2016-08-19')
to '/user/hive/action_log.export'
```

这个例子是将 dealer_action_log 表中的一个分区，备份到 HDFS 上，to 后面的路径是 HDFS 上的路径。

备份是将表的元数据和数据都导出到 HDFS 上。

7、 还原表

```
import table dealer_action_log_like from '/user/hive/action_log.export';
```

将备份在 HDFS 上的文件，还原到 dealer_action_log_like 表中。

五、 Hive QL 数据查询语法

1、 Select 查询

(1) 指定列表

```
select * from dealer_leads;
select leads_id,dealer_id,create_time from dealer_leads;
select e.leads_id from dealer_leads e;
```

(2) 函数列

```
select companyid,upper(host),UUID(32) from dealer_action_log;
```

可以使用 hive 自带的函数，也可以是使用用户自定义函数。

上面这个例子 upper()就是 hive 自带函数，UUID()就是用户自定义函数。

关于函数详细介绍，可以参考后面的章节。

(3) 算数运算列

```
select companyid,userid, (companyid + userid) as sumint from dealer_action_log;
```

可以进行各种算数运算，运算结果做为结果列。

运算符	描述	运算符	描述
A+B	数字相加	A-B	数字相减
A*B	相乘	A/B	相除
A%B	模除		

(4) 限制返回条数

```
select * from dealer_action_log limit 100;
```

类似于 sql server 里的 top N。

(5) Case When Then 语句

```
--case when 两种写法
```

```
select case companyid when 0 then '未登录' else companyid end from dealer_action_log;
select case  when companyid=0 then '未登录' else companyid end from dealer_action_log;
```

这个例子，判断 companyid 值，如果为 0 则显示为未登录，如果不为 0，则返回 companyid 的值。

2、 Where 筛选

操作符	说明	操作符	说明
A=B	A 等于 B 就返回 true,适用于各种基本类型	**A<=>B**	**都为 Null 则返回 True，其他和=一样**
A<>B	不等于	A!=B	不等于
A<B	小于	A<=B	小于等于
A>B	大于	A>=B	大于等于
A Between B And C	筛选 A 的值处于 B 和 C 之间	A Not Between B And C	筛选 A 的值不处于 B 和 C 之间
A Is NULL	筛选 A 是 NULL 的	A Is Not NULL	筛选 A 值不是 NULL 的
A Link B	%一个或者多个字符 _一个字符	A Not Like B	%一个或者多个字符 _一个字符
A RLike B	**正则匹配**		

3、 Group By 分组

4、 子查询

Hive 对子查询的支持有限，只允许在 select from 后面出现。比如：

```
--只支持如下形式的子查询
select * from (
  select dealerid,dealername from dealer_info i where i.dealerid='10595'
  ) a;

--不支持如下的子查询
select
(select dealername from dealer_info i where i.dealerid=d.dealer_id)
from dealer_leads d where d.dealer_id='10595';
```

六、 Join

1、 Hive Join 的限制

(1) 只支持等值连接

Hive 支持类似 SQL Server 的大部分 Join 操作，但是注意只支持等值连接，并不支持不等连接。原因是 Hive 语句最终是要转换为 MapReduce 程序来执行的，但是 MapReduce 程序很难实现这种不等判断的连接方式。

```
--等值连接
select lead.* from dealer_leads lead
left join dealer_info info
on lead.dealer_id=info.dealerid;

--不等连接（不支持）
select lead.* from dealer_leads lead
left join dealer_info info
on lead.dealer_id!=info.dealerid;
```

(2) 连接谓词中不支持 or

```
--on 后面的表达式不支持 or
select lead.* from dealer_leads lead
left join dealer_info info
on lead.dealer_id=info.dealerid or lead.leads_id=0;
```

2、 Inner join

内连接同 SQL Sever 中的一样，连接的两个表中，只有同时满足连接条件的记录才会放入结果表中。

3、 Left join

同 SQL Server 中一样，两个表左连接时，符合 Where 条件的左侧表的记录都会被保留下来，而符合 On 条件的右侧的表的记录才会被保留下来。

4、 Right join

同 Left Join 相反，两个表左连接时，符合 Where 条件的右侧表的记录都会被保留下来，而符合 On 条件的左侧的表的记录才会被保留下来。

5、 Full join

Full Join 会将连接的两个表中的记录都保留下来。

6、 Left Semi-Join（ exists 语句）

SQL Server 中有 exists 语句，类似下面的语句，但是 Hive 中不支持 Exists 语句。

```
--SQL Sever 中的exists 语句，但是hive 中支持
SELECT i.* FROM  DealerInfo i WHERE EXISTS (SELECT 1 FROM DealerScopeRelation s WHERE
s.DealerInfoId=i.DealerInfoID AND s.CompanyID=i.CompanyID)
```

对于这种需求，Hive 使用 Left Semi-Join（左半开连接）来解决。

```
SELECT i.* from DealerInfo i left semi-join DealerScopeRelation s
on i.DealerInfoId=s.DealerInfoId and i.CompanyID=s.CompanyID
```

但是这里注意，select 后面的列，不能有 left semi-join 右边表的字段，只能是左边表的字段。

七、 排序

1、 Order By

```
select * from dealer_leads order by dealer_id
```

Hive 中的 Order By 达到的效果和 SQL Server 中是一样的，会对查询结果进行全局排序，但是 Hive 语句最终要转换为 MapReduce 程序放到 Hadoop 分布式集群上去执行，Order By 这样的操作，肯定要在 Map 后汇集到一个 Reduce 上执行，如果结果数据量大，那就会造成 Reduce 执行相当漫长。

所以，Hive 中尽量不要用 Order By，除非非常确定结果集很小。

但是排序的需求总是有的，Hive 中使用下面的几种排序来满足需求。

2、 Sort By

```
select * from dealer_leads sort by dealer_id
```

这个例子中，Sort By 是在每个 reduce 中进行排序，是一个局部排序，可以保证每个 Reduce 中是按照 dealer_id 进行排好序的，但是全局上来说，相同的 dealer_id 可以被分配到不同的 Reduce 上，虽然在各个 Reduce 上是排好序的，但是全局上不一定是排好序的。

3、 Distribute By 和 Sort By

```
--Distribute By 和 Sort By 实例
select * from dealer_leads where dealer_id!='0' Distribute By cast(dealer_id as int) Sort by
cast(dealer_id as int);
```

Distribute By 指定 map 输出结果怎么样划分后分配到各个 Reduce 上去，比如 Distribute By dealer_id，就可以保证 dealer_id 字段相同的结果被分配到同一个 reduce 上去执行。然后再指定 Sort By dealer_id，则在 Reduce 上进行按照 dealer_id 进行排序。

但是这种还是不能做到全局排序，只能保证排序字段值相同的放在一起，并且在 reduce 上局部是排好序的。

需要注意的是 Distribute By 必须写在 Sort By 前面。

4、 Cluster By

如果 Distribute By 和 Sort By 的字段是同一个，可以简写为 Cluster By

```
select * from dealer_leads where dealer_id!='0' Cluster By cast(dealer_id as int) ;
```

5、 常见全局排序需求

常见的排序需求有两种：要求最终结果是有序的、按某个字段排序后取出前 N 条数据。

(1) 最终结果是有序的

最终分析结果往往是比较小的，因为客户不太可能最终要的是一个超级大数据集。所以在得到最终的小结果集后，使用 order by 进行排序。

```
select * from (
    select dealer_id,count(leads_id) cnt from dealer_leads   where dealer_id!='0' group by dealer_id
  ) a order by a.cnt;
```

这个语句让程序首先执行 group by 语句获取到一个小结果集，group by 过程中是不指定排序的，然后再对小结果集进行排序，这样得到的最终结果是全局排序的。

(2) 取前 N 条

```
select a.leads_id,a.user_name from (
  select leads_id,user_name   from dealer_leads
  distribute by length(user_name)   sort by length(user_name) desc limit 10
) a order by length(a.user_name) desc limit 10;
```

八、 Hive 内置函数

1、 参考资料

Hive 中自带了大量的内置函数，详细可参看如下资源：

(1) 官方文档：

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-Built-inFunctions

(2) 网友整理的中文文档

http://blog.csdn.net/wisgood/article/details/17376393

2、 explode 函数

3、 collect_set 函数

4、 collect_list 函数

九、 Hive 自定义函数

同 SQL Server 一样，Hive 也允许用户自定义函数，这大大扩展了 Hive 的功能，Hive 是用 Java 语言写的，所以自定义函数也需要用 Java 来写。

编写一个 Hive 的自定义函数，需要新建一个 Java 类来继承 UDF 类并实现 evaluate()函数，evaluate()函数中编写自定义函数的实现逻辑，返回值给 Hive 使用，需要注意的是，evaluate()函数的输入输出都必须是 Hadoop 的数据类型，以便可以被 MapReduce 程序来进行序列化反序列化。编写完成后将 Java 程序打成 Jar 包，在 Hive 会话中载入 Jar 包来使用自定义函数。

在执行 Hive 语句时，遇到一个自定义函数就会实例化一个类，并执行对应的 evaluate()函数，每行输入都会调用一次 evaluate()函数，所以在编写自定义函数时，一定要注意大数据量时的效率问题。

Hive 中的自定义函数依据输入输出数据的个数，分为以下几类：

1、 UDF 用户自定义函数（一进一出）

这种是最普通最常见的自定义函数，类似内置函数 length()、yaer()等函数，输入为一个值，输出也为一个值。下面是一个获取唯一 ID 的自定义函数例子：

```
package com.autohome.ics.bigdata.common.Date;

import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.hive.ql.udf.UDFType;
```

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import java.util.Date;
/*
 生成一个指定长度的随机字符串(最长为36 位)
 */
@org.apache.hadoop.hive.ql.exec.Description(name = "UUID",
        extended = "示例：select UUID(32) from src;",
        value = "_FUNC_(leng)-生成一个指定长度的随机字符串(最长为 36 位)")
@UDFType(deterministic = false)
public class UUID extends UDF {
    public Text evaluate(IntWritable leng) {
        String uuid = java.util.UUID.randomUUID().toString();
        int le = leng.get();
        le = le > uuid.length() ? uuid.length() : le;
        return new Text(uuid.substring(0, le));
    }
    /*
     生成一个随机字符串
    */
    public Text evaluate() {
        String uuid = java.util.UUID.randomUUID().toString();
        return new Text(uuid);
    }
}
```

(1) 这个实例是获取一个指定长度的随机字符串自定义函数，这个自定义函数创建了一个类 UUID，继承于 UDF 父类。

(2) UUID 类要实现 evaluate 函数，获取一个指定长度的随机字符串。

(3) evaluate 函数是可以有多个重载的。

(4) Description 是自定义函数的描述信息。

(5) 这里有一个参数 deterministic，是标识这个自定义函数是否是那种输入确定时输出就确定的函数，默认是 true，比如 length 函数就是如果输入同一个值，那么输出肯定是一致的，但是我们这里的 UUID 就算输入确定，但是输出也是不确定的，所以要将 deterministic 设置为 false。

2、 UDAF 用户自定义聚合函数（多进一出）

UDAF 是自定义聚合函数，类似于 sum()、avg()，这一类函数输入是多个值，输出是一个值。

UDAF 是需要 hive sql 语句和 group by 联合使用的。

聚合函数常常需要对大量数组进行操作，所以在编写程序时，一定要注意内存溢出问题。

UDAF 分为两种：简单 UDAF 和通用 UDAF。简单 UDAF 写起来比较简单，但是因为使用了 JAVA 的反射机制导致性能有所损失，另外有些特性不能使用，如可变参数列表，通用 UDAF 可以使用所有功能，但是写起来比较复杂。

(1) 简单 UDAF 实例

```
package com.autohome.ics.bigdata.common.number;
import org.apache.hadoop.hive.ql.exec.UDAF;
import org.apache.hadoop.hive.ql.exec.UDAFEvaluator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import java.util.regex.Pattern;

/**
* Created by 鸣宇淳 on 2016/8/30.
*对传入的字符串列表，按照数字大小进行排序后，找出最大的值
*/
public class MaxIntWithString extends UDAF {
    public static  class MaxIntWithStringUDAFEvaluator implements  UDAFEvaluator{
        //最终结果最大的值
        private IntWritable MaxResult;

        @Override
        public void init() {
            MaxResult=null;
        }
        //每次对一个新值进行聚集计算都会调用 iterate 方法
        //这里将 String 转换为 int 后，进行比较大小
        public boolean iterate(Text value)
        {
            if(value==null)
                return false;
            if(!isInteger(value.toString()))
            {
                return false;
            }
            int number=Integer.parseInt(value.toString());

            if(MaxResult==null)
```

```
                MaxResult=new IntWritable(number);
            else
                MaxResult.set(Math.max(MaxResult.get(), number));
            return true;
        }
        //Hive 需要部分聚集结果的时候会调用该方法
        //会返回一个封装了聚集计算当前状态的对象
        public IntWritable terminatePartial()
        {
            return MaxResult;
        }
        //合并两个部分聚集值会调用这个方法
        public boolean merge(Text other)
        {
            return iterate(other);
        }
        //Hive 需要最终聚集结果时候会调用该方法
        public IntWritable terminate()
        {
            return MaxResult;
        }
        private static boolean isInteger(String str) {
            Pattern pattern = Pattern.compile("^[-\\+]?[\\d]*$");
            return pattern.matcher(str).matches();
        }
    }
}
```

- UDAF 要继承于 UDAF 父类 org.apache.hadoop.hive.ql.exec.UDAF。
- 内部类要实现 org.apache.hadoop.hive.ql.exec.UDAFEvaluator 接口。
- MaxIntWithStringUDAFEvaluator 类里需要实现 init、iterate、terminatePartial、merge、terminate 这几个函数，是必不可少的
- init()方法用来进行全局初始化的。
- iterate()中实现比较逻辑。
- terminatePartial 是 Hive 部分聚集时调用的，类似于 MapReduce 里的 Combiner，这里能保证能得到各个部分的最大值。
- merge 是多个部分合并时调用的，得到了参与合并的最大值。

✧ terminate 是最终 Reduce 合并时调用的，得到最大值。

这里参考了：

http://computerdragon.blog.51cto.com/6235984/1288567

http://blog.csdn.net/xch_w/article/details/16886179

(2) 通用 UDAF 实例

开发通用 UDAF 有两个步骤，第一个是编写 resolver 类，第二个是编写 evaluator 类。resolver 负责类型检查，操作符重载。evaluator 真正实现 UDAF 的逻辑。通常来说，顶层 UDAF 类继承 org.apache.hadoop.hive.ql.udf.GenericUDAFResolver2，里面编写嵌套类 evaluator 实现 UDAF 的逻辑。

通用 UDAF 使用场景较少，详情可以参看内置函数的源码，或者官方文档。

3、 UDTF 自定义表生成函数（一进多出）

UDTF 是将一个输入值转变为一个数组。

下面这个例子是从 nginx 日志中的 agent 信息中提取浏览器名称和版本号的自定义函数，输入参数类似于：*”Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML like Gecko) Chrome/31.0.1650.63 Safari/537.36”*，输出为：*Chrome 31.0.1650.63*。

```
package com.autohome.ics.bigdata.common.String;

import cz.mallat.uasparser.OnlineUpdater;
import cz.mallat.uasparser.UASparser;
import cz.mallat.uasparser.UserAgentInfo;
import org.apache.hadoop.hive.ql.exec.UDFArgumentException;
import org.apache.hadoop.hive.ql.metadata.HiveException;
import org.apache.hadoop.hive.ql.udf.generic.GenericUDTF;
import org.apache.hadoop.hive.serde2.objectinspector.*;
import
org.apache.hadoop.hive.serde2.objectinspector.primitive.PrimitiveObjectInspectorFactory
;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

/**
```

```
 * 解析浏览器信息UDTF
 *
 * 将日志中的http_user_agent，得到 browser_name,browser_version 两个字段
 *  Created by ad on 2016/7/29.
 */
public class ParseUserAgentUDTF  extends GenericUDTF{
    private static UASparser uaSparser;
    static{
        try {
            uaSparser = new UASparser(OnlineUpdater.getVendoredInputStream());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    private final ListObjectInspector listIO = null;
    private final Object[] forwardObj = new Object[2];
    /**
     * 声明解析出来的字段名称和类型
     * @param argOIs
     * @return
     * @throws UDFArgumentException
     */
    @Override
    public StructObjectInspector initialize(StructObjectInspector argOIs) throws
UDFArgumentException {
        if(argOIs.getAllStructFieldRefs().size() != 1){
            throw new UDFArgumentException("args error!");
        }
        ArrayList<String> fieldNames = new ArrayList<String>();
        ArrayList<ObjectInspector> fieldOIs = new ArrayList<>();
        fieldNames.add("browser_name");
        fieldNames.add("browser_version");
        fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);
        fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);
        return
ObjectInspectorFactory.getStandardStructObjectInspector(fieldNames,fieldOIs);
    }
    @Override
    public void process(Object[] args) throws HiveException {
        // 真正解析的地方
      /*
       输入字符串："Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/31.0.1650.63 Safari/537.36"
       解析为：
```

```
        */
        String userAgent = args[0].toString();

        try {
            UserAgentInfo userAgentInfo = uaSparser.parse(userAgent);
            List<String> bws = new ArrayList<>();
            bws.add(userAgentInfo.getUaFamily());
            bws.add(userAgentInfo.getBrowserVersionInfo());
            super.forward(bws.toArray(new String[0]));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void close() throws HiveException {
    }
}
```

使用：

(1) 打包为 jar 包（包名为：common.jar），因为这个程序引用了依赖 uasparser，所以打包时注意应该将依赖 uasparser 打进去。

(2) hive 中添加 jar 包

```
add jar lib/common.jar;
```

(3) 生命函数

```
create temporary function ParseUserAgent as
'com.autohome.ics.bigdata.common.String.ParseUserAgentUDTF';
```

(4) 查询

```
select ParseUserAgent(agent) from dealer_action_log_like;
```

(5) 结果

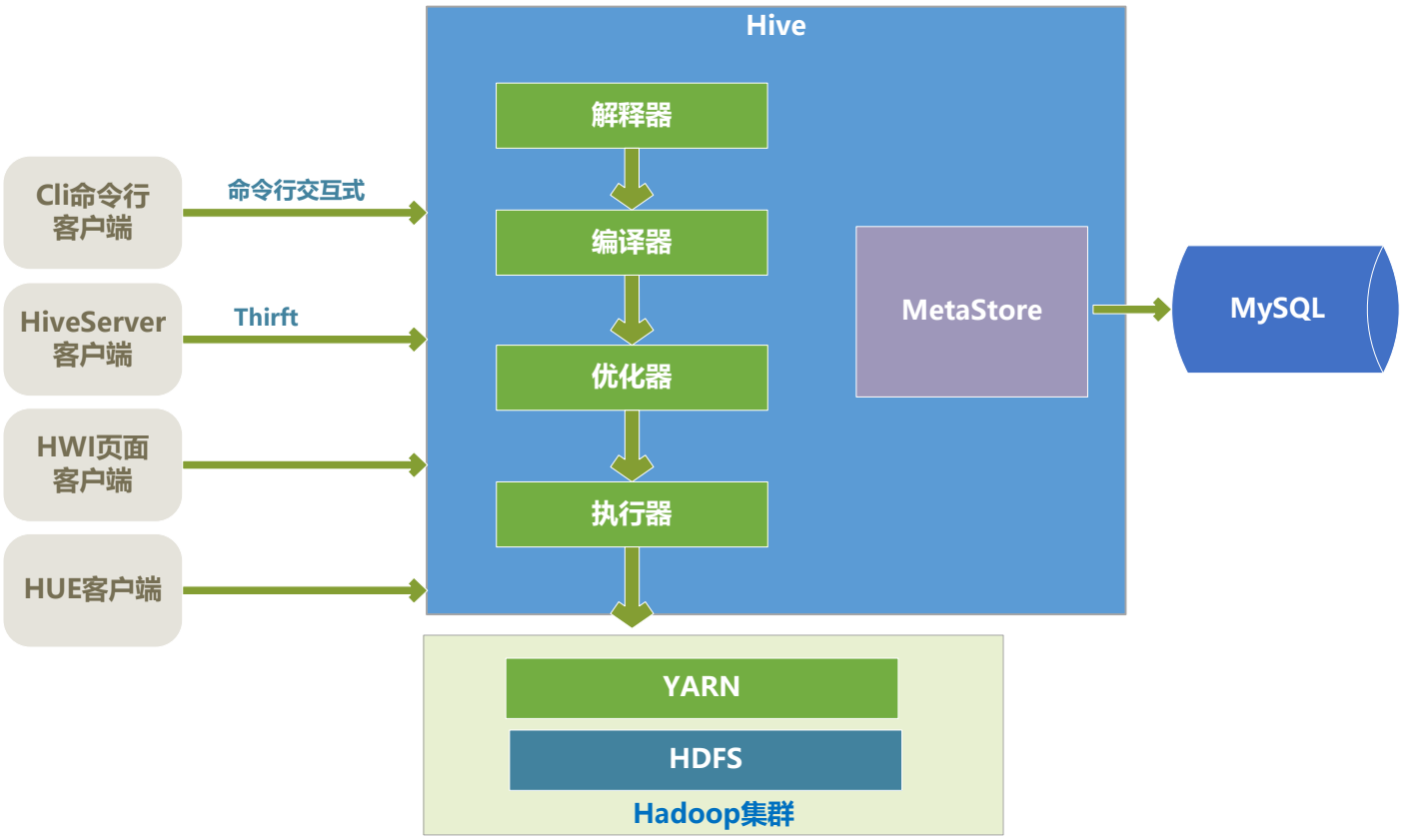
```
Chrome  31.0.1650.63
Chrome  31.0.1650.63
Chrome  31.0.1650.63
Chrome  31.0.1650.63
Chrome  31.0.1650.63
```

参考：https://cwiki.apache.org/confluence/display/Hive/DeveloperGuide+UDTF

第二部分：Hive 执行原理与优化

十、 Hive 技术架构

1、 架构图



2、 Hive 的核心

Hive 的核心是驱动引擎，驱动引擎由四部分组成：

(1) 解释器：解释器的作用是将 HiveSQL 语句转换为语法树（AST）。

(2) 编译器：编译器是将语法树编译为逻辑执行计划。

(3) 优化器：优化器是对逻辑执行计划进行优化。

(4) 执行器：执行器是调用底层的运行框架执行逻辑执行计划。

3、 Hive 的底层存储

Hive 的数据是存储在 HDFS 上的。Hive 中的库和表可以看做是对 HDFS 上数据做的一个映射。所以 Hive 必须是运行在一个 Hadoop 集群上的。

4、 Hive 程序的执行过程

Hive 中的执行器，是将最终要执行的 MapReduce 程序放到 YARN 上以一系列 Job 的方式去执行。

5、 Hive 的元数据存储

Hive 的元数据是一般是存储在 MySQL 这种关系型数据库上的，Hive 和 MySQL 之间通过 MetaStore 服务交互。

Owner	库、表的所属者	CreateTime	创建时间
LastAccessTime	最后修改时间	Location	存储位置
Table Type	表类型（内部表、外部表）		表的字段信息

6、 Hive 客户端

Hive 有很多种客户端。

(1) cli 命令行客户端：采用交互窗口，用 hive 命令行和 Hive 进行通信。

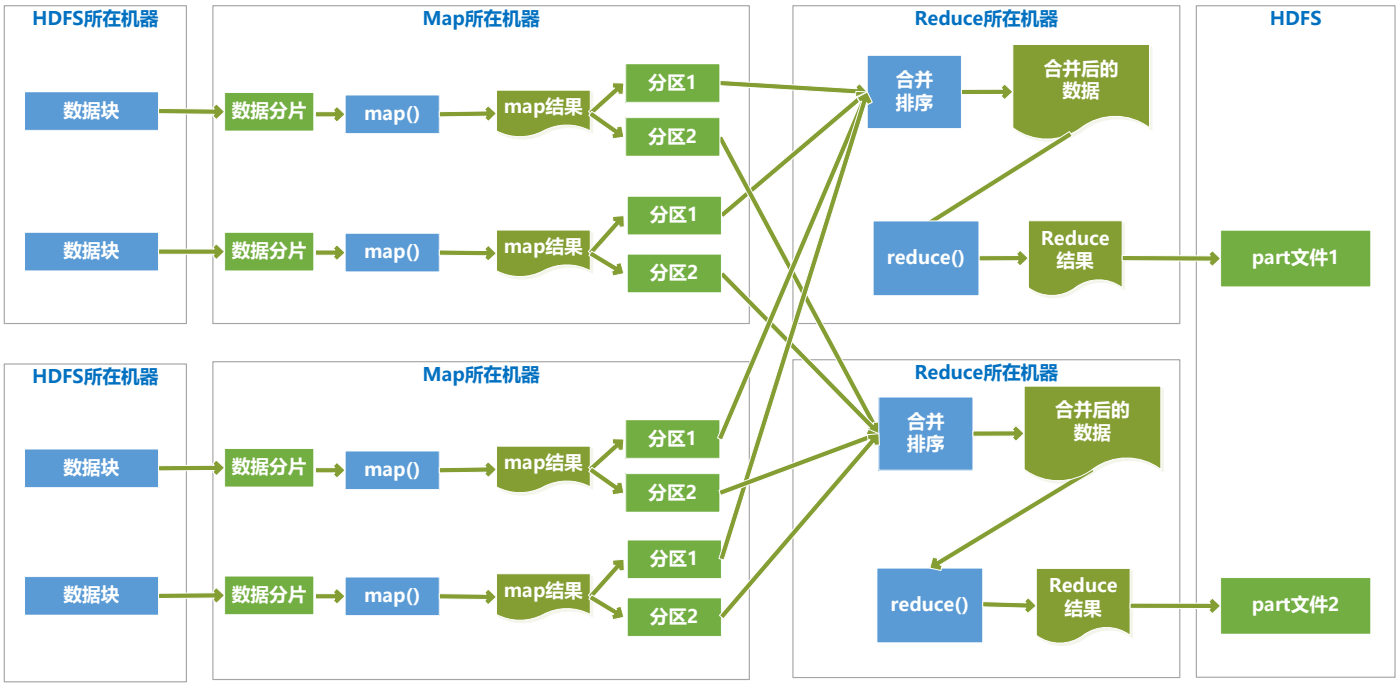
(2) HiveServer2 客户端：用 Thrift 协议进行通信，Thrift 是不同语言之间的转换器，是连接不同语言程序间的协议，通过 JDBC 或者 ODBC 去访问 Hive。

(3) HWI 客户端：hive 自带的一个客户端，但是比较粗糙，一般不用。

(4) HUE 客户端：通过 Web 页面来和 Hive 进行交互，使用的比较多。

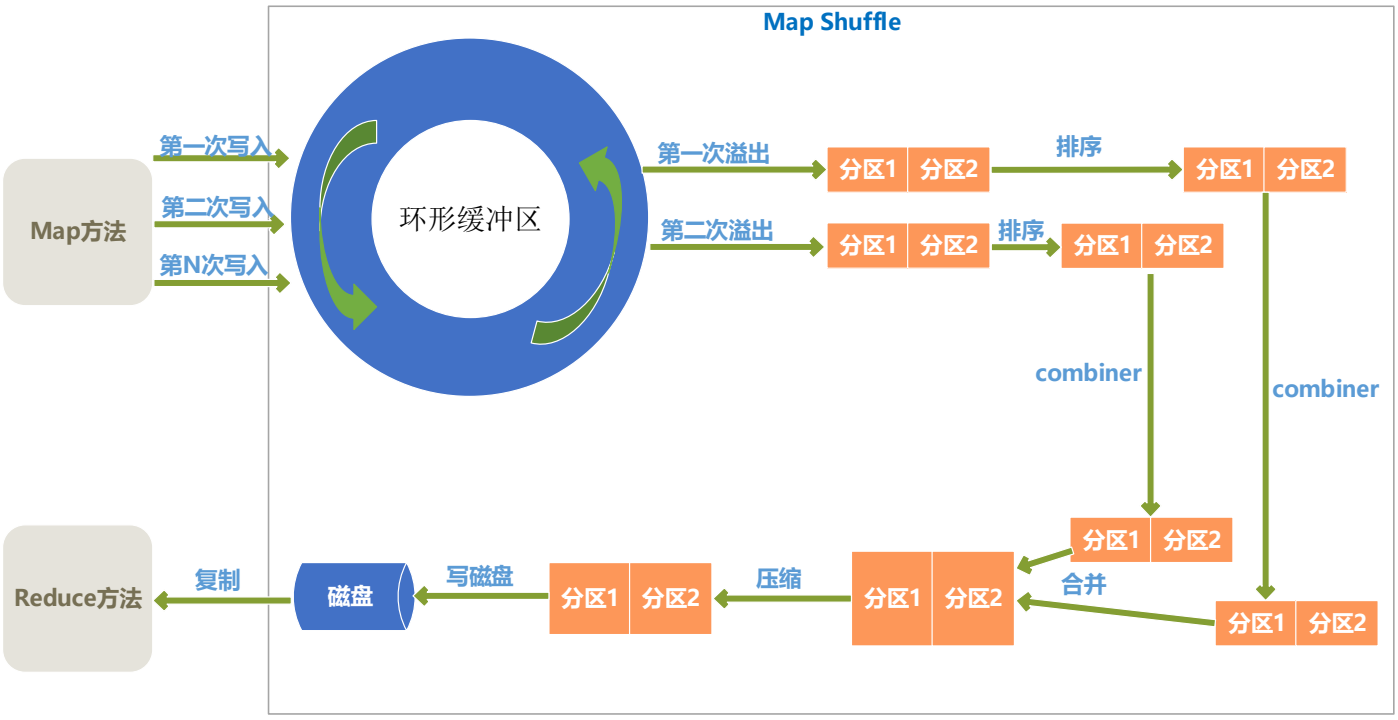
十一、 MapReduce 执行过程

1、 MapReduce 执行示意图



十二、 Shuffle 原理

1、 Map Shuffle 过程



(1) 环形缓冲区

Map 输出结果是先放入内存中的一个环形缓冲区，这个环形缓冲区默认大小为 100M(这个大小可以在 io.sort.mb 属性中设置)，当环形缓冲区里的数据量达到阀值时（这个值可以在 io.sort.spill.percent 属性中设置）就会溢出写入到磁盘，环形缓冲区是遵循先进先出原则，Map 输出一直不停地写入，一个后台进程不时地读取后写入磁盘，如果写入速度快于读取速度导致环形缓冲区里满了时，map 输出会被阻塞直到写磁盘过程结束。

(2) 分区

从环形缓冲区溢出到磁盘过程，是将数据写入 mapred.local.dir 属性指定目录下的特定子目录的过程。

但是在真正写入磁盘之前，要进行一系列的操作，首先就是对于每个键，根据规则计算出来将来要输出到哪个 reduce，根据 reduce 不同分不同的区，分区是在内存里分的，分区的个数和将来的 reduce 个数是一致的。

(3) 排序

在每个分区上，会根据键进行排序。

(4) combiner

combiner 方法是对于 map 输出的结果按照业务逻辑预先进行处理，目的是对数据进行合并，减少 map 输出的数据量。

排序后，如果指定了 conmbiner 方法，就运行 combiner 方法使得 map 的结果更紧凑，从而减少写入磁盘和将来网络传输的数据量。

(5) 合并溢出文件

环形缓冲区每次溢出，都会生成一个文件，所以在 map 任务全部完成之前，会进行合并成为一个溢出文件，每次溢出的各个文件都是按照分区进行排好序的，所以在合并文件过程中，也要进行分区和排序，最终形成一个已经分区和排好序的 map 输出文件。

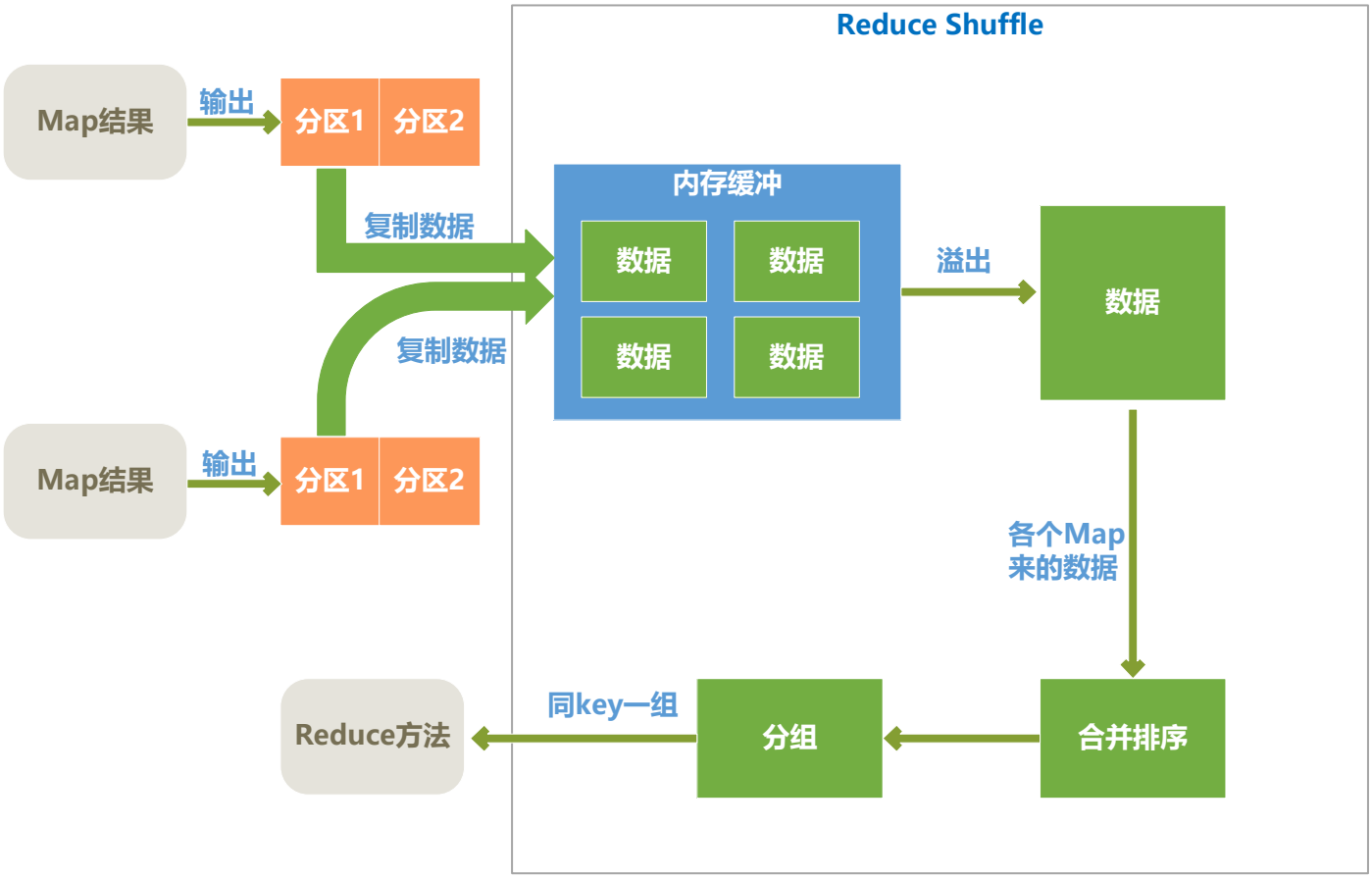
在合并文件时，如果文件个数大于某个指定的数量（可以在 min.num.spills.for.combine 属性设置），就会进再次 combiner 操作，如果文件太少，效果和效率上，就不值得花时间再去执行 combiner 来减少数据量了。

(6) 压缩

Map 输出结果在进行了一系列的分区、排序、combiner 合并、合并溢出文件后，得到一个 map 最终的结果后，就应该真正存储这个结果了，在存储之前，可以对最终结果数据进行压缩，一是可以节约磁盘空间，而是可以减少传递给 reduce 时的网络传输数据量。

默认是不进行压缩的，可以在 mapred.compress.map.output 属性设置为 true 就启用了压缩，而压缩的算法有很多，可以在 mapred.map.output.compression.codec 属性中指定采用的压缩算法，具体压缩详情，可以看本文的后面部分的介绍。

2、 Reduce Shuffle 过程



(1) 复制数据

各个 map 完成时间肯定是不同的，只要有一个 map 执行完成，reduce 就开始去从已完成的 map 节点上复制输出文件中属于它的分区中的数据，reduce 端是多线程并行来复制各个 map 节点的输出文件的，线程数可以在 mapred.reduce.parallel.copies 属性中设置。

reduce 将复制来的数据放入内存缓冲区（缓冲区大小可以在 mapred.job.shuffle.input.buffer.percent 属性中设置）。当内存缓冲区中数据达到阀值大小或者达到 map 输出阀值，就会溢写到磁盘。

写入磁盘之前，会对各个 map 节点来的数据进行合并排序，合并时如果指定了 combiner，则会再次执行 combiner 以尽量减少写入磁盘的数据量。为了合并，如果 map 输出是压缩过的，要在内存中先解压缩后合并。

(2) 合并排序

合并排序其实是和复制文件同时并行执行的，最终目的是将来自各个 map 节点的数据合并并排序后，形成一个文件。

(3) 分组

分组是将相同 key 的键值对分为一组，一组是一个列表，列表中每一组在一次 reduce 方法中处理。

(4) 执行 Reduce 方法

Reduce 端的 Shuffle 完成后，就交由 reduce 方法来进行处理了。

十三、 性能瓶颈和优化

Hadoop 就像吞吐量巨大的轮船，启动开销大，如果每次只做小数量的输入输出，利用率将会很低。所以用好 Hadoop 的首要任务是增大每次任务所搭载的数据量。Hadoop 的核心能力是 parition 和 sort，因而这也是优化的根本。

Hive 优化时，把 hive Sql 当做 mapreduce 程序来读,而不是当做 SQL 来读。

十四、 HiveQL 层面优化

Hive 的优化需要从 HiveQL 语句、架构、底层 MapReduce 三个层面入手。

1、 利用分区表优化

分区表是在某一个或者某几个维度上对数据进行分类存储，一个分区对应于一个目录。在这中的存储方式，当查询时，如果筛选条件里有分区字段，那么 Hive 只需要遍历对应分区目录下的文件即可，不用全局遍历数据，使得处理的数据量大大减少，提高查询效率。

当一个 Hive 表的查询大多数情况下，会根据某一个字段进行筛选时，那么非常适合创建为分区表。

2、 利用桶表优化

桶表的概念在本教程第一步有详细介绍，就是指定桶的个数后，存储数据时，根据某一个字段进行哈希后，确定存储再哪个桶里，这样做的目的和分区表类似，也是使得筛选时不用全局遍历所有的数据，只需要遍历所在桶就可以了。

(1) hive.optimize.bucketmapJOIN 为 true

(2) sort-merge JOIN

hive.input.format=org.apache.hadoop.hive.ql.io.bucketizedHiveInputFormat;

hive.optimize.bucketmapjoin=true;

hive.optimize.bucketmapjoin.sortedmerge=true;

3、 join 优化

(1) 优先过滤后再 join,最大限度地减少参与 Join 的数据量。

(2) 小表 join 大表原则

应该遵守小表 join 大表原则，原因是 Join 操作的 reduce 阶段，位于 join 左边的表内容会被加载进内存，将条目少的表放在左边，可以有效减少发生内存溢出的几率。join 中执行顺序是从做到右生成 Job，应该保证连续查询中的表的大小从左到右是依次增加的。

(3) join on 条件相同的放入一个 job

hive 中，当多个表进行 join 时，如果 join on 的条件相同，那么他们会合并为一个 MapReduce Job，所以利用这个特性，可以将相同的 join on 的放入一个 job 来节省执行时间。

```
select pt.page_id,count(t.url) PV
```

```
from rpt_page_type pt
join
(
    select url_page_id,url from trackinfo where ds='2016-10-11'
) t   on pt.page_id=t.url_page_id
join
(
    select page_id from rpt_page_kpi_new where ds='2016-10-11'
) r   on t.url_page_id=r.page_id
group by pt.page_id;
```

4、 启用 mapjoin

mapjoin 是将 join 双方比较小的表直接分发到各个 map 进程的内存中，在 map 进程中进行 join 操作，这样就省掉了 reduce 步骤，提高了速度。

mapjoin 相关参数如下：

```
<property>
    <name>hive.auto.convert.join</name>
    <value>true</value>
    <description>Whether Hive enables the optimization about converting common join into mapjoin based on the input file size</description>
  </property>
```

```
<property>
    <name>hive.auto.convert.join.noconditionaltask</name>
    <value>true</value>
    <description>
      Whether Hive enables the optimization about converting common join into mapjoin based on the input file size.
      If this parameter is on, and the sum of size for n-1 of the tables/partitions for a n-way join is smaller than the
      specified size, the join is directly converted to a mapjoin (there is no conditional task).
    </description>
  </property>
```

```
<property>
    <name>hive.auto.convert.join.noconditionaltask.size</name>
    <value>10000000</value>
    <description>
      If hive.auto.convert.join.noconditionaltask is off, this parameter does not take affect.
```

```
        However, if it is on, and the sum of size for n-1 of the tables/partitions for a n-way join is smaller than this size,
        the join is directly converted to a mapjoin(there is no conditional task). The default is 10MB
      </description>
    </property>
```

```
<property>
    <name>hive.auto.convert.join.use.nonstaged</name>
    <value>false</value>
    <description>
        For conditional joins, if input stream from a small alias can be directly applied to join operator without
        filtering or projection, the alias need not to be pre-staged in distributed cache via mapred local task.
        Currently, this is not working with vectorization or tez execution engine.
    </description>
  </property>
```

(1) hive.auto.convert.join 为 true 时，join 方数据量小的表会整体分发到各个 map 进程的内存中，在 map 进程本地进行 join 操作，这样能大大提高运算效率，牺牲的是内存容量，所以数据量小于某一个值的才允许用 mapjoin 分发到各个 map 节点里，而这个值用以下参数来配置。

(2) hive.auto.convert.join.noconditionaltask 设置为 true，hive 才基于输入文件大小进行自动转换为 mapjoin。

(3) hive.auto.convert.join.noconditionaltask.size 指定小于多少的表数据放入 map 内存，使用 mapjoin,默认是 10M。

(4) 这个优化只对 join 有效，对 left join、right join 无效。

5、 桶表 mapjoin

当两个分桶表 join 时，如果 join on 的是分桶字段，小表的分桶数时大表的倍数时，可以启用 map join 来提高效率。启用桶表 mapjoin 要启用 hive.optimize.bucketmapjoin 参数。

```
<property>
    <name>hive.optimize.bucketmapjoin</name>
    <value>true</value>
    <description>Whether to try bucket mapjoin</description>
  </property>
```

6、 Group By 数据倾斜优化

Group By 很容易导致数据倾斜问题，因为实际业务中，通常是数据集中在某些点上，这也符合常见的 2/8 原则，这样会造成对数据分组后，某一些分组上数据量非常大，而其他的分组上数据量很小，而在 mapreduce 程序中，同一个分组的数据会分配到同一个 reduce 操作上去，导致某一些 reduce 压力很大，其他的 reduce 压力很小，这就是数据倾斜，整个 job 执行时间取决于那个执行最慢的那个 reduce。

解决这个问题的方法是配置一个参数：set hive.groupby.skewindata=true。

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 GroupBy Key 被分布到同一个 Reduce 中），最后完成最终的聚合操作。

7、 Order By 优化

因为 order by 只能是在一个 reduce 进程中进行的，所以如果对一个大数据集进行 order by,会导致一个 reduce 进程中处理的数据相当大，造成查询执行超级缓慢。在要有进行 order by 全局排序的需求时，用以下几个措施优化：

(1) 在最终结果上进行 order by，不要在中间的大数据集上进行排序。如果最终结果较少，可以在一个 reduce 上进行排序时，那么就在最后的结果集上进行 order by。

(2) 如果需求是取排序后前 N 条数据，那么可以使用 distribute by 和 sort by 在各个 reduce 上进行排序后取前 N 条，然后再对各个 reduce 的结果集合并后在一个 reduce 中全局排序，再取前 N 条，因为参与全局排序的 Order By 的数据量最多有 reduce 个数*N，所以速度很快。

例子：

```
select a.leads_id,a.user_name from
(
  select leads_id,user_name   from dealer_leads
  distribute by length(user_name)   sort by length(user_name) desc limit 10
 ) a order by length(a.user_name) desc limit 10;
```

8、 Group By Map 端聚合

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。

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

9、 一次读取多次插入

有些场景是从一个表读取数据后，要多次利用，这时候就可以使用 multi insert 语法：

```
from dealer_action_log
 insert overwrite table log1 select companyid,originalstring   where companyid='100006'
 insert overwrite table log2 select companyid,originalstring   where companyid='10002'
```

每次 hive 查询，都会将数据集整个遍历一遍。当查询结果会插入多个表中时，可以采用以上语法，将一次遍历写入多个表，以达到提高效率的目的

10、 Join 字段显示类型转换

当参与 join 的字段类型不一致时，Hive 会自动进行类型转换，但是自动转换有时候效率并不高，可以根据实际情况通过显示类型转换来避免 HIVE 的自动转换。

举例说明：

11、 使用 orc、parquet 等列式存储格式

创建表时，尽量使用 orc、parquet 这些列式存储格式，因为列式存储的表，每一列的数据在物理上是存储在一起的，Hive 查询时会只遍历需要列数据，大大减少处理的数据量。

十五、 Hive 架构层面优化

1、 不执行 MapReduce

hive 中有个参数：hive.fetch.task.conversion，定义如下：

```
<property>
  <name>hive.fetch.task.conversion</name>
  <value>minimal</value>
  <description>
    Some select queries can be converted to single FETCH task minimizing latency.
    Currently the query should be single sourced not having any subquery and should not have
```

```
    any aggregations or distincts (which incurs RS), lateral views and joins.
    1. minimal : SELECT STAR, FILTER on partition columns, LIMIT only
    2. more    : SELECT, FILTER, LIMIT only (TABLESAMPLE, virtual columns)
  </description>
</property>
```

Hive 从 HDFS 读取数据，有两种方式：启用 MapReduce 读取、直接抓取。

很显然直接抓取数据比 MapReduce 读取数据要快的多，但是只有少数操作可以直接抓取数据，hive.fetch.task.conversion 参数就是设置什么情况下采用直接抓取方法，它的值有两个：

(1) minimal:只有 select * 、在分区字段上 where 过滤、有 limit 这三种场景下才启用直接抓取方式。

(2) more:在 select、where 筛选、limit 时，都启用直接抓取方式。

启用 fetch more 模式: *set hive.fetch.task.conversion=more;*

实例：

```
set hive.fetch.task.conversion=more;
select dealerid,dealername from dealer_info where cityid is not null;
```

这个例子中，如果 set hive.fetch.task.conversion=minimal，那么下面的查询语句会以 MapReduce 方法执行，运行时间比较长，但是改为 more 后，发现查询速度非常快。

2、 本地模式执行 MapReduce

Hive 在集群上查询时，默认是在集群上 N 台机器上运行，需要多个机器进行协调运行，这个方式很好地解决了大数据量的查询问题。但是当 Hive 查询处理的数据量比较小时，其实没有必要启动分布式模式去执行，因为以分布式方式执行就涉及到跨网络传输、多节点协调等，并且消耗资源。这个时间可以只使用本地模式来执行 mapreduce job，只在一台机器上执行，速度会很快。

启动本地模式涉及到三个参数：

参数名	默认值	备注
hive.exec.mode.local.auto	false	让 hive 决定是否在本地模式自动运行
hive.exec.mode.local.auto.input.files.max	4	不启用本地模式的 task 最大个数
hive.exec.mode.local.auto.inputbytes.max	128M	不启动本地模式的最大输入文件大小

各个参数定义如下：

```
<property>
  <name>hive.exec.mode.local.auto</name>
  <value>false</value>
  <description> Let Hive determine whether to run in local mode automatically </description>
</property>
```

```
<property>
    <name>hive.exec.mode.local.auto.input.files.max</name>
    <value>4</value>
    <description>When hive.exec.mode.local.auto is true, the number of tasks should less than this for local mode.</description>
  </property>
```

```
<property>
    <name>hive.exec.mode.local.auto.inputbytes.max</name>
    <value>134217728</value>
    <description>When hive.exec.mode.local.auto is true, input bytes should less than this for local mode.</description>
  </property>
```

set hive.exec.mode.local.auto=true 是打开 hive 自动判断是否启动本地模式的开关，但是只是打开这个参数并不能保证启动本地模式，要当 map 任务数不超过 hive.exec.mode.local.auto.input.files.max 的个数并且 map 输入文件大小不超过 hive.exec.mode.local.auto.inputbytes.max 所指定的大小时，才能启动本地模式。

3、 JVM 重用

因为 Hive 语句最终要转换为一系列的 MapReduce Job 的，而每一个 MapReduce Job 是由一系列的 Map Task 和 Reduce Task 组成的，默认情况下，MapReduce 中一个 Map Task 或者一个 Reduce Task 就会启动一个 JVM 进程，一个 Task 执行完毕后，JVM 进程就退出。这样如果任务花费时间很短，又要多次启动 JVM 的情况下，JVM 的启动时间会变成一个比较大的消耗，这个时候，就可以通过重用 JVM 来解决。

```
set mapred.job.reuse.jvm.num.tasks=5
```

这个设置就是制定一个 jvm 进程在运行多次任务之后再退出，这样一来，节约了很多的 JVM 的启动时间。

4、 并行化

一个 hive sql 语句可能会转为多个 mapreduce Job，每一个 job 就是一个 stage，这些 job 顺序执行，这个在 hue 的运行日志中也可以看到。但是有时候这些任务之间并不是是相互依赖的，如果集群资源允许的话，可以让多个并不相互依赖 stage 并发执行，这样就节约了时间，提高了执行速度，但是如果集群资源匮乏时，启用并行化反倒是会导致各个 job 相互抢占资源而导致整体执行性能的下降。

启用并行化：

```
set hive.exec.parallel=true;
```

十六、 Hive 底层 MapReduce 优化

1、 合理设置 Map 数

上面的 mapreduce 执行过程部分介绍了，在执行 map 函数之前会现将 HDFS 上文件进行分片，得到的分片做为 map 函数的输入，所以 map 数量取决于 Map 的输入分片(inputsplit)，一个输入分片对应于一个 Map Task，而输入分片由三个参数决定的：

参数名	默认值	备注
dfs.block.size	128M	HDFS 上数据块的大小
mapreduce.min.split.size	0	最小分片数
mapreduce.max.split.size	256M	最大分片大小

公式：分片大小=max(mapreduce.min.split.size,min(dfs.block.size, mapreduce.max.split.size))

默认情况下分片大小和 dfs.block.size 是一致的，即一个 HDFS 数据块对应一个输入分片，对应一个 Map Task。这时候一个 map task 中只处理一台机器上的一个数据块，不需要将数据跨网络传输，提高了数据处理速度。

2、 合理设置 reduce 数

决定 reduce 数量的相关参数有：

参数名	默认值	备注
hive.exec.reducers.bytes.per.reducer	1G	一个 reduce 数据量的大小
hive.exec.reducers.max	999	hive 最大的个数
mapred.reduce.tasks	-1	reduce task 的个数,-1 是根据 hive.exec.reducers.bytes.per.reducer 自动调整

所以可以用 set mapred.reduce.tasks 手动调整 reduce task 个数。

(1) 调整原则

每个 Map Task 和 Reduce Task 都是一个进程，启动、初始化和停止进程都需要耗费一定时间，但是如果每一个 Task 处理的数据量过多，会造成单个 Task 负载过大，执行时间也会很长。

调整的原则就是在单个 Task 负载和 Task 进程消耗之间找到一个平衡点。

(2) 一个实践

当发现 Reduce 阶段比较慢时，可以尝试着增加 reduce 个数，如果发现增加 reduce 个数后 Time taken 值减少，说明优化有效，可以继续增加 reduce 个数，当发现随着 reduce 个数增加，Time Taken 值增大了，说明已经过了最优的临界点。

因为 Time Taken 是 MapReduce 总耗自然时间，Reduce 个数增加后是增加了分布计算的进程数，更多进程参与计算会减少总耗时，但是太多又会增加总耗时。

当优化过程中，如果发现 MapReduce 的 CPU 总时间(MapReduce Total cumulative CPU time)减少了，但是 Time taken 值并没有减少，说明

第三部分：Hive 高级知识

十七、 Hive 文件格式

1、 常见文件格式

HDFS 上文件一般都是文本格式存储的，因为源数据本身就是文本格式的，但是在 Hive 执行过程中，可以采用其他优化的文本格式。

Hive 中的文件格式常见的有：textfile 文件格式、sequencefile 二进制序列化文件格式、rcfile、orc、parquet。hive 表的文件格式一般是在创建表时用 stored as 语句声明，如：

```
create table demo_textfile
(
  id int,
  name string
 )
 stored as textfile;
```

其中 textfile 和 sequencefile 是以行存储数据的，rcfile、orc、parquet 是列式存储的。

存储方式	文件格式		输入输出格式
行式存储	textfile	InputFormat	org.apache.hadoop.mapred.TextInputFormat
		OutputFormat	org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
	sequencefile	InputFormat	org.apache.hadoop.mapred.SequenceFileInputFormat
		OutputFormat	org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat
列式存储	rcfile	InputFormat	org.apache.hadoop.hive.ql.io.RCFileInputFormat
		OutputFormat	org.apache.hadoop.hive.ql.io.RCFileOutputFormat
	orc	InputFormat	org.apache.hadoop.hive.ql.io.orc.OrcInputFormat
		OutputFormat	org.apache.hadoop.hive.ql.io.orc.OrcInputFormat
	parquet	InputFormat	org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat
		OutputFormat	org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat

(1) TextFile 格式

TextFile 是 Hive 的默认文件格式，数据不做压缩，磁盘开销比较大，数据解析时开销也比较大。从本地文件向 Hive load 数据只能用 textfile 文件格式。

(2) SequenceFile 格式

SequenceFile 是 Hadoop API 提供的一种二进制文件支持，其具有使用方便、可分割、可压缩的特点。

(3) Rcfile 格式

Rcfile 是一种行列存储结合的存储方式，首先将数据按行分块，保证同一个记录在一个块上，避免读取一行记录需要读取多个块的情况，然后块数据列式存储，这样有利于数据压缩和列存取。

(4) Orc 格式

Orc 格式是 Rcfile 格式的升级版，性能有很大的提升，并且数据可以压缩存储，比 textfile 文件压缩比可以达到 70%，同时读取性能也非常高，推荐使用 orc 文件格式创建表。

- 单个 Hive Task 输出单个文件，减小文件系统负载。

- 支持 datetime、decimal 和其他复杂类型（struct、list、map 和 union）。
- 文件内含轻量级索引。减少不必要的扫描，高效定位记录。
- 基于数据类型的块模式压缩。例如 String 和 Integer 可以采用不同的压缩方式。
- 同一文件可以利用多个 RecordReader 并发读取。
- 支持免扫描进行文件分块。
- 读写文件时，绑定 I/O 所需的最大内存空间。
- 文件的 metadata 采取 Protocol Buffers 格式，允许灵活的属性增删。

(5) Parquet 格式

Parquet 是一种适合多种计算框架的文件格式，Parquet 是语言无关的，并且不与任何一种数据处理框架绑定在一起，适配多种语言和组件，能够与 parquet 配合的组件有：

查询引擎：Hive、Impala、Pig、Presto、Drill、Tajo、HAWQ、IBM Big SQL。

计算框架：MapReduce、Spark、Cascading、Crunch、Scalding、Kite

所以如果一套数据要多种引擎使用，Parquet 是最好的选择。

2、 列式存储

(1) 数据存储方式

目前大数据存储有两种方案可以供选择：行存储、列存储。

物理存储上，行存储是一行中各列顺序存储，列存储是一列中各行的值顺序存储。

行存储和列存储各有优缺点：

- 行存储的写入是一次性完成的，消耗的时间比列存储少，并且能够保证数据的完整性，缺点是数据读取过程中会产生冗余数据，如果数据量大会影响到数据的处理效率。
- 列存储在写入效率、保证数据完整性方面不如行存储，但是他的优势在于读取过程，不会产生冗余数据

(2) 列式存储在大数据存储中的优势

- 可以跳过不符合条件的数据，只读取需要的数据，降低 IO 数据量。

- 由于同一列的数据类型是一样的，所以可以使用更高效的压缩编码方式，最大限度地节约存储空间。
- 只读取需要列，能够获取更好的扫描性能。

十八、 Hive 压缩方法

1、 压缩的原因

Hive 最终是转为 MapReduce 程序来执行的，而 MapReduce 的性能瓶颈在于网络 IO 和磁盘 IO，要解决性能瓶颈，最主要的是减少数据量，对数据进行压缩是个好的方式。

压缩虽然是减少了数据量，但是压缩过程要消耗 CPU 的，但是在 Hadoop 中，往往性能瓶颈不在于 CPU，CPU 压力并不大，所以压缩充分利用了比较空闲的 CPU。

2、 Hadoop 常用压缩方法

压缩格式	是否可拆分	是否自带	压缩率	速度	是否 hadoop 自带
gzip	否	是	很高	比较快	是
lzo	是	是	比较高	很快	否，要安装
snappy	否	是	比较高	很快	否，要安装
bzip2	是	否	最高	慢	是

各个压缩格式对应的类：

压缩格式	类
Zlib	org.apache.hadoop.io.compress.DefaultCodec
Gzip	org.apache.hadoop.io.compress.GzipCodec
Bzip2	org.apache.hadoop.io.compress.BZip2Codec
Lzo	org.apache.hadoop.io.compress.lzo.LzoCodec
Lz4	org.apache.hadoop.io.compress.Lz4Codec
Snappy	org.apache.hadoop.io.compress.SnappyCodec

(1) 压缩方式选择原则：

- 压缩比率
- 压缩解压速度
- 是否支持 split

(2) map 的输入压缩

最好选择一种支持 split 的压缩方式，如果选择不支持 split 的压缩方式，大文件将会由一个 map 进程进行处理。如果要选择不支持的 split 压缩方式，那么就先将大文件进行分割成大小接近 128M 的小文件，然后对这些小文件进行单独压缩。

(3) map 的输出压缩

map 的输出压缩，要注重考虑压缩解压速度，常用的用 snappy 压缩，

(4) reduce 端的输出

很少对 reduce 端的输出进行压缩，但是一下两个场景会对使用压缩

- reduce 输出结果后面甚少使用，一般要用压缩以提高性能。一般使用压缩比率比较高的压缩格式。
- 迭代计算时，reduce 输出结果要给下一个 job 做为输入使用，着重使用压缩解压速度比较快的方式。

3、 配置 Hadoop 压缩解压

在 Hadoop 的 mapred-site.xml 配置文件中的配置。

参数	备注
mapreduce.map.output.compress	map 输出是否启用压缩
mapreduce.map.output.compress.codec	map 输出采用压缩方式
mapreduce.output.fileoutputformat.compress	是否启用 reduce 输出压缩
mapreduce.output.fileoutputformat.compress.codec	reduce 压缩方式
mapreduce.output.fileoutputformat.compress.type	压缩级别：NONE, RECORD(行级别), BLOCK(块级别)

4、 Hive 中的压缩

在 Hive 中，只有当属性 hive.exec.compress.intermediate 设置为 true,以上 hadoop 设置的压缩才生效。如果 Hive SQL 被翻译成多个 MapReduce 时，这个属性不单单控制 MapReduce 中 map 的输出结果压缩，也控制着 job 之间的输出输入的压缩。

```
<property>
    <name>hive.exec.compress.intermediate</name>
    <value>true</value>
```

```
    <description>
      This controls whether intermediate files produced by Hive between multiple map-reduce jobs are compressed.
      The compression codec and other options are determined from Hadoop config variables mapred.output.compress*
    </description>
  </property>
```

十九、 复杂类型

Hive 中的列支持使用 array、map、struct 三种机和数据类型。

类型	描述	语法
array	array 是一组具有相同类型和名称的变量的集合，这些变量称为数据的元素，每个数组元素编号都从 0 开始。	
map	map 是一组键值对元素集合。可通过 字段名['key']来访问值。	
struct	值类似于对象，有属性和值，可以用.来访问值。	

大多数关系型数据库并不支持这类集合数据类型，因为他们会破坏二维表的标准格式，造成数据不一致，然而在大数据系统中，不遵守标准格式的好吃就是可以提供更高的数据吞吐量。

1、 举例

```
create table employees (
 name string,
 salary float,
 subname array<string>,
 deductions map<string,float>,
 address struct<provice:string,city:string,zip:int>
);
```

这个例子是创建一个雇员表，subname 是下属雇员的姓名，是一个字符串数组。deductions 是一个由键值对构成的列表，几率了每一项扣款项，键是扣款项名称，值是扣款额。

address 是家庭住址，有 provice、city、zip 等属性。

2、 array 类型

数据类型相同的数组，可以用 array 类型。

3、 map 类型

一系列的键值对，可以用 map 类型。

4、 struct 类型

值为对象类型，有属性和值，并且有子对象。这时候用 struct 类型。

5、 union 类型

union 类型更加复杂，是 array、map、struct 三种类型的联合使用。

6、 字段分隔符

默认的字段分隔符：

分隔符	描述
\n	对于文本文件来说，每行是一条记录，所以\n 来分割记录
^A (Ctrl+A)	分割字段，也可以用\001 来表示
^B (Ctrl+B)	用于分割 Arrary 或者 Struct 中的元素，或者用于 map 中键值之间的分割，也可以用\002 分割。
^C	用于 map 中键和值自己分割，也可以用\003 表示。

以上是默认的分隔符，在创建表是可以自定义分隔符，实例如下：

```
create table employees (
 name string,
 salary float,
 subname array<string>,
 deductions map<string,float>,
 address struct<provice:string,city:string,zip:int>
)
row format delimited
fields terminated by '\001'
collection items terminated by '\002'
map keys terminated by '\003'
lines terminated by '\n'
stored as TextFile
;
```

二十、 Hive SQL 转换为 MapReduce 过程

Hive 是将 SQL 转化为 MapReduce 任务要经过解释、编译，整译过程分为六个阶段：

(1) Antlr 定义 SQL 的语法规则，完成 SQL 词法，语法解析，将 SQL 转化为抽象语法树 AST Tree

(2) 遍历 AST Tree，抽象出查询的基本组成单元 QueryBlock

(3) 遍历 QueryBlock，翻译为执行操作树 OperatorTree(逻辑执行计划)

(4) 逻辑层优化器进行 OperatorTree 变换，合并不必要的 ReduceSinkOperator，减少 shuffle 数据量

(5) 遍历 OperatorTree，翻译为 MapReduce 任务

(6) 物理层优化器进行 MapReduce 任务的变换，生成最终的执行计划

二十一、 Hive 解释器

Hive 解释器是将 Hive SQL 语句转换为抽象语法树 AST，过程 分为解析和生成语法树两个步骤。

1、 词法语法解析

Antlr 是一种语言识别工具，可以用来构造领域语言，使用 Antlr 构造特定的语言需要编写一个语法文件，定义词法和语法替换规则。

Hive 使用 Antlr 实现 SQL 的词法和语法解析，完成词法分析、语法分析、语义定义、中间代码生成的过程。Hive 中有五个文件记录着词法规则和语法规则：

词法规则文件：HiveLexer.g。

语法规则文件：SelectClauseParser.g、FormClauseParser.g、IdentifiersParser.g、HiveParser.g。

2、 生成抽象语法树

在词法和语法解析的同时，Antlr 生成一个抽象语法树。

二十二、 Hive 编译器

Hive 编译器是将抽象语法树编译为逻辑执行计划。

抽象语法树是比较复杂的，不够结构化，不方便直接翻译为 MapReduce 程序，所以 Hive 编译器会将抽象语法树再进一步抽象和结构化为逻辑执行计划。

二十三、 Hive 优化器

Hive 优化器是对逻辑执行计划进行优化。

大部分逻辑层优化器通过变换 OperatorTree，合并操作符，达到减少 MapReduce Job，减少 Shuffle 数据量的目的。

名称	作用
② SimpleFetchOptimizer	优化没有 GroupBy 表达式的聚合查询
② MapJoinProcessor	MapJoin，需要 SQL 中提供 hint，0.11 版本已不用
② BucketMapJoinOptimizer	BucketMapJoin
② GroupByOptimizer	Map 端聚合
① ReduceSinkDeDuplication	合并线性的 OperatorTree 中 partition/sort key 相同的 reduce
① PredicatePushDown	谓词前置
① CorrelationOptimizer	利用查询中的相关性，合并有相关性的 Job，HIVE-2206
ColumnPruner	字段剪枝

上表中带①符号的，优化目的都是尽量将任务合并到一个 Job 中，以减少 Job 数量，带②的优化目的是尽量减少 shuffle 数据量。

二十四、 Hive 执行器

Hive 执行器是调用底层的框架，执行优化好的逻辑执行计划。

编译器将操作树切分为一个 Task 链(DAG)，执行器会顺序执行其中的所有 Task，如果 Task 链不存在依赖关系时，可以采用并发执行的方式进行 Job 执行。

附录 A ： HIVE 安装

1、安装 Hive

(1)选择 CDH 版本的

选择版本 hive-0.13.1-cdh5.3.6.tar.gz

(2)解压文件

[hadoop@bigdata-senior03 sofeware]$ tar -zxf hive-0.13.1-cdh5.3.6.tar.gz -C /opt/modules/

```
[hadoop@bigdata-senior03 hive-0.13.1-cdh5.3.6]$ ll
total 312
drwxr-xr-x 3 hadoop hadoop   4096 Jul 29  2015 bin
drwxr-xr-x 2 hadoop hadoop   4096 Jul 29  2015 conf
drwxr-xr-x 3 hadoop hadoop   4096 Jul 29  2015 data
drwxr-xr-x 6 hadoop hadoop   4096 Jul 29  2015 docs
drwxr-xr-x 4 hadoop hadoop   4096 Jul 29  2015 examples
drwxr-xr-x 7 hadoop hadoop   4096 Jul 29  2015 hcatalog
drwxr-xr-x 4 hadoop hadoop   4096 Jul 29  2015 lib
-rw-r--r-- 1 hadoop hadoop  23828 Jul 29  2015 LICENSE
-rw-r--r-- 1 hadoop hadoop    277 Jul 29  2015 NOTICE
-rw-r--r-- 1 hadoop hadoop   3838 Jul 29  2015 README.txt
-rw-r--r-- 1 hadoop hadoop 253839 Jul 29  2015 RELEASE_NOTES.txt
drwxr-xr-x 3 hadoop hadoop   4096 Jul 29  2015 scripts
```

2、配置 Hive

(1)配置环境变量

设置 HADOOP_HOME 变量：

[hadoop@bigdata-senior03 hive-0.13.1-cdh5.3.6]$ vim /etc/profile

```
export HADOOP_HOME="/opt/modules/hadoop-2.5.0"
export PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH
```

(2)复制配置文件

从模板复制文件：

```
[hadoop@bigdata-senior03 hive-0.13.1-cdh5.3.6]$ cp conf/hive-default.xml.template conf/hive-site.xml
```

```
[hadoop@bigdata-senior03 hive-0.13.1-cdh5.3.6]$ cp conf/hive-env.sh.template conf/hive-env.sh

[hadoop@bigdata-senior03 hive-0.13.1-cdh5.3.6]$ cp conf/hive-exec-log4j.properties.template conf/hive-exec-log4j.properties

[hadoop@bigdata-senior03 hive-0.13.1-cdh5.3.6]$ cp conf/hive-log4j.properties.template conf/hive-log4j.properties
```

(3)修改 hive-env.sh

[hadoop@bigdata-senior03 hive-0.13.1-cdh5.3.6]$ vim conf/hive-env.sh

添加 JAVA_HOME 参数

export JAVA_HOME=/opt/modules/jdk1.7.0_67

```
export JAVA_HOME=/opt/modules/jdk1.7.0_67


# Set HADOOP_HOME to point to a specific hado
# HADOOP_HOME=${bin}/../../hadoop
```

(4)修改 hive-site.xml

hive-site.xml 文件有一个 bug，在第 2783 行后面缺少一个<property>标签。需要补全 property 标签。

```
2774 <property>
2775   <name>hive.lazysimple.extended_boolean_liter
2776   <value>false</value>
2777   <description>
2778     LazySiimpleSerde uses this properties to c
2779     '1', and '0' as extened, legal boolean lit
2780     The default is false, which means only 'TR
2781     boolean literal.
2782   </description>
2783 </property>
2784   <name>hive.mapjoin.optimized.hashtable</name
2785   <value>true</value>
2786   <description>Whether Hive should use memory-
     hashtable cannot be serialized.</description>
2787 </property>
2788
```

3、安装 MySQL 数据库

Hive 对 mysql 要去是最低版本为 5.6.17 版本的。用 yum 安装 mysql 可能需要更新 yum 源的版本。

(1)更新 yum mysql 版本

- 下载 mysql 的 rpm 文件：mysql57-community-release-el6-8.noarch.rpm
- 安装 rpm，更新 yum 源里的 mysql 版本。

  [hadoop@bigdata-senior03 hive-0.13.1-cdh5.3.6]$ sudo rpm -Uvh /opt/sofeware/mysql57-community-release-el6-8.noarch.rpm
- 查看 yum 源里的 mysql

  [hadoop@bigdata-senior03 hive-0.13.1-cdh5.3.6]$ cd /etc/yum.repos.d/

```
[hadoop@bigdata-senior03 hive-0.13.1-cdh5.3.6]$ cd /etc/yum.repos.d/
[hadoop@bigdata-senior03 yum.repos.d]$ ll
total 32
-rw-r--r--. 1 root root 1991 Oct 23  2014 CentOS-Base.repo
-rw-r--r--. 1 root root  647 Oct 23  2014 CentOS-Debuginfo.repo
-rw-r--r--. 1 root root  289 Oct 23  2014 CentOS-fasttrack.repo
-rw-r--r--. 1 root root  630 Oct 23  2014 CentOS-Media.repo
-rw-r--r--. 1 root root 5394 Oct 23  2014 CentOS-Vault.repo
-rw-r--r--  1 root root 1221 Mar 22 19:25 mysql-community.repo
-rw-r--r--  1 root root 1236 Mar 22 19:25 mysql-community-source.repo
```

- 修改 mysql-community.repo 和 mysql-community-source.repo 文件

  在两个文件中，将 mysql5.7 中的 enabled 改为 0，将 MySQL5.6 中的 enabled 改为 1。这样 yum 安装时就安装 mysql5.6 版本了。

```
# Enable to use MySQL 5.6
[mysql56-community]
name=MySQL 5.6 Community Server
baseurl=http://repo.mysql.com/yum/mysql-5.6-community/el/6/$basearch/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql

[mysql57-community]
name=MySQL 5.7 Community Server
baseurl=http://repo.mysql.com/yum/mysql-5.7-community/el/6/$basearch/
enabled=0
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql
```

(2)yum 安装 mysql

```
[hadoop@bigdata-senior03 yum.repos.d]$ sudo yum -y install mysql-
community-server
```

(3)启动 mysql

```
[hadoop@bigdata-senior03 yum.repos.d]$ sudo service mysqld start
```

(4)mysql 安全性设置

- 安装完成后，默认 root 用户是没有密码的，并且有一些匿名用户，查看用户可以以下命令。

  用 root 用户登录 mysql ： [hadoop@bigdata-senior03 yum.repos.d]$ mysql -uroot

  切换到 mysql 库：mysql> user mysql;

  查看 user 表：

```
mysql> select host,user from user;
+-------------------------------+------+
| host                          | user |
+-------------------------------+------+
| 127.0.0.1                     | root |
| ::1                           | root |
| bigdata-senior03.chybinmy.com |      |
| bigdata-senior03.chybinmy.com | root |
| localhost                     |      |
| localhost                     | root |
+-------------------------------+------+
6 rows in set (0.04 sec)
```

- 用脚本初始化权限

  [hadoop@bigdata-senior03 yum.repos.d]$ sudo mysql_secure_installation

  这种初始化方式只适合于 MySql 5.6 版本。

提示输入当前 root 密码，当前密码为空：Enter current password for root (enter for none):

提示是否设置 root 密码： Set root password? [Y/n] y

提示是否移除匿名用户：Remove anonymous users? [Y/n] y

提示是否禁用 root 远程登录：Disallow root login remotely? [Y/n] n

提示是否移除 test 数据库：Remove test database and access to it? [Y/n] n

提示是否重新加载权限：Reload privilege tables now? [Y/n] y

- 再次查看用户信息

用 root 和密码登录： [hadoop@bigdata-senior03 yum.repos.d]$ mysql -uroot -p123456

切换到 mysql 库： mysql> use mysql;
查看用户信息，已经有了密码，并且匿名用户已经被删除了。

```
mysql> select host,user,password from user;

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

| host                          | user | password                                  |

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

| localhost                     | root |
*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |

| bigdata-senior03.chybinmy.com | root |
*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |

| 127.0.0.1                     | root |
*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |

| ::1                           | root |
*6BB4837EB74329105EE4568DDA7DC67ED2CA2AD9 |
```

```
+----------------------------+------+-----------------------------------------+

4 rows in set (0.00 sec)
```

- 给 root 设置所有权限

  mysql> grant all privileges on *.* to 'root'@'%' identified by '123456'with grant option;

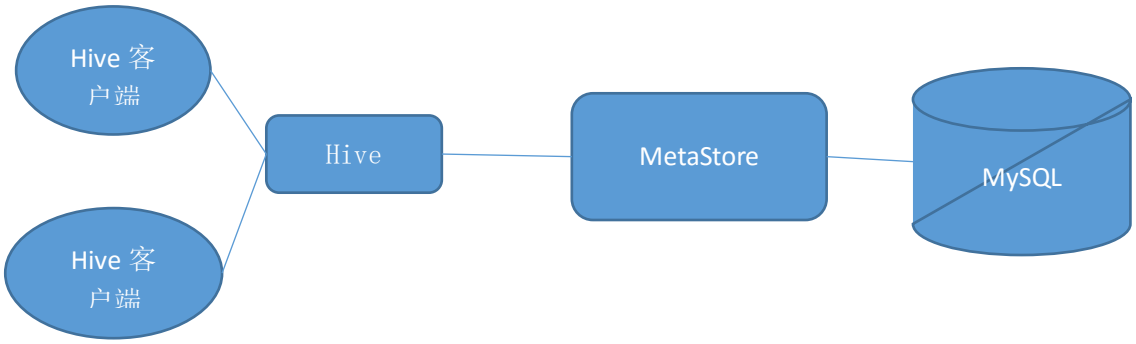
  刷新权限。

  mysql> flush privileges;

  Query OK, 0 rows affected (0.00 sec)

4、在远程 MySQL 存储模式配置

(1)部署架构图



hive 元数据存储在远程 mysql 数据库上，是通过 MetaStore 来进行访问的。所以要配置一个 metastore 服务。

(2)配置 hive-env.sh

配置好 JAVA_HOME 和 HADOOP_HOME

```
export JAVA_HOME=/opt/modules/jdk1.7.0_67


# Set HADOOP_HOME to point to a specific hadoop install directory
HADOOP_HOME=/opt/modules/hadoop-2.5.0
```

配置了 HADOOP_HOME 参数后，hive 就可以知道 Hadoop 在哪里，数据存储在哪里了。

(3)修改 hive-site.xml 文件

- 将 hive.metastore.uris 属性设置为 thrift://bigdata-senior03.chybinmy.com:9083

  这个是设置 metastore 服务在哪个机器上，端口是什么。thrift 是一种协议，用于不同语言程序间，利用 RPC（远程过程调用）方式进行通讯。

```
<property>
  <name>hive.metastore.uris</name>
  <value>thrift://bigdata-senior03.chybinmy.com:9083</value>
  <description>Thrift URI for the remote metastore. Used by me
on>
</property>
```

- 修改 javax.jdo.option.ConnectionURL 属性为：jdbc:mysql://bigdata-senior03.chybinmy.com:3306/remote_db?createDatabaseIfNotExist=true

  javax.jdo.option.ConnectionURL 属性的默认值为：

```
<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:derby:;databaseName=metastore_db;create=true</value>
  <description>JDBC connect string for a JDBC metastore</descriptio
</property>
```

  derby 是 hive 默认的存储元数据的数据库，一般不用这个数据库，用 mysql 数据库。

  这个是配置 mysql 的连接字符串，bigdata-senior03.chybinmy.com 是 mysql 所在的主机名，remote_db 是存储元数据的数据块名，createDatabaseIfNotExist 是指如果数据库不存在就自动创建。

- 修改 javax.jdo.option.ConnectionDriverName

```
<property>
  <name>javax.jdo.option.ConnectionDriverName</name>
  <value>com.mysql.jdbc.Driver</value>
  <description>Driver class name for a JDBC metastore</description>
</property>
```

- 修改连接 Mysql 的用户名和密码

```
<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>123456</value>
  <description>password to use against metastore database</description>
</property>
```

(4)拷贝 mysql 连接驱动包

metastore 连接 mysql 需要有一个 mysql 连接驱动包，这里选择驱动包 mysql-connector-java-5.1.26-bin.jar，将这个驱动包拷贝到 hive 根目录下的 lib 目录下。

[hadoop@bigdata-senior03 hive-0.13.1-cdh5.3.6]$ cp /opt/sofeware/mysql-connector-java-5.1.26-bin.jar /opt/modules/hive-0.13.1-cdh5.3.6/lib/

(5)启动 metastore 服务

必须先启动 metastore 服务，再启动 hive

[hadoop@bigdata-senior03 hive-0.13.1-cdh5.3.6]$ hive --service metastore

将 metastore 服务设置为守护进程

[hadoop@bigdata-senior03 hive-0.13.1-cdh5.3.6]$ nohup hive --service metastore > hive_metastore.run.log 2>&1 &

5、启动 Hadoop

start-dfs.sh

6、启动 Hive

此时，执行 hive 客户端，访问的元数据就是通过 metastore 来访问 mysql 上存储的元数据了。

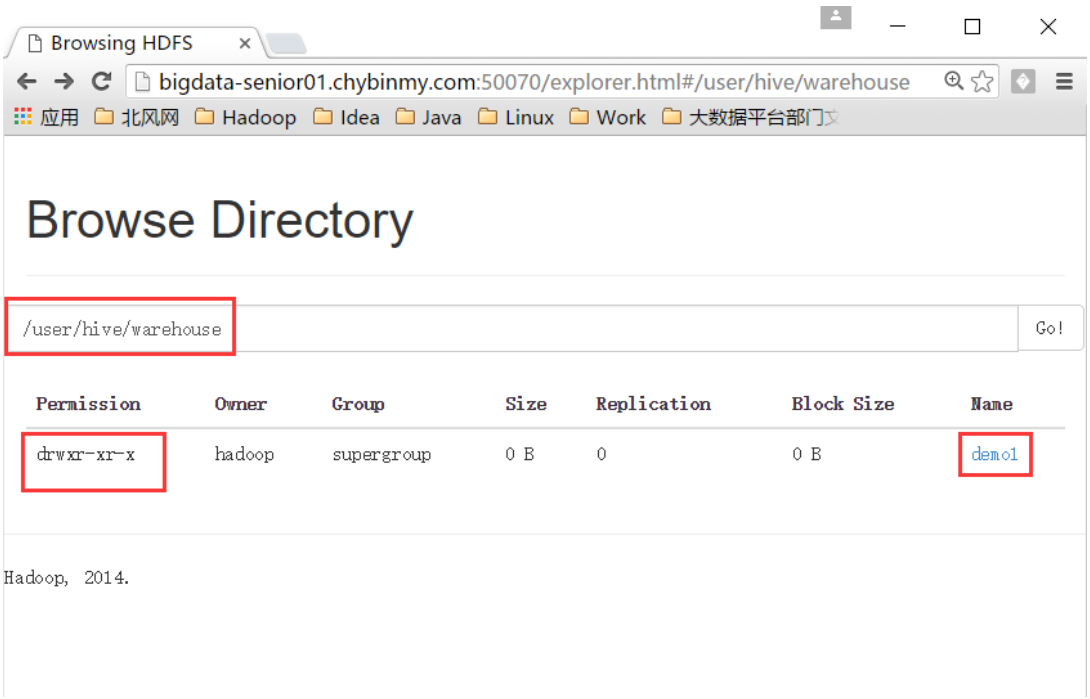
7、测试 hive

(1)创建 hive 表

```
hive> use default;
hive> create table demo1 (id int,name string);
hive> show tables;
demo1
```

(2)HDFS 查看数据

hive 表的数据实际是存储在 HDFS 的指定目录下的一个文件。



欢迎点击这里的链接进入精彩的Linux公社 网站

Linux公社（www.Linuxidc.com）于2006年9月25日注册并开通网站，Linux现在已经成为一种广受关注和支持的一种操作系统，IDC是互联网数据中心，LinuxIDC就是关于Linux的数据中心。

Linux公社是专业的Linux系统门户网站，实时发布最新Linux资讯，包括Linux、Ubuntu、Fedora、RedHat、红旗Linux、Linux教程、Linux认证、SUSE Linux、Android、Oracle、Hadoop、CentOS、MySQL、Apache、Nginx、Tomcat、Python、Java、C语言、OpenStack、集群等技术。

Linux公社（LinuxIDC.com）设置了有一定影响力的Linux专题栏目。

Linux公社 主站网址：www.linuxidc.com 旗下网站：www.linuxidc.net

包括：**Ubuntu 专题** **Fedora 专题** **Android 专题** **Oracle 专题** **Hadoop 专题** **RedHat 专题** **SUSE 专题** **红旗 Linux 专题** **CentOS 专题**

Linux 公社微信公众号：linuxidc_com