大数据数仓基石-Hive

Hive3.1全解析

=== 楼兰 ===

一、关于Hive

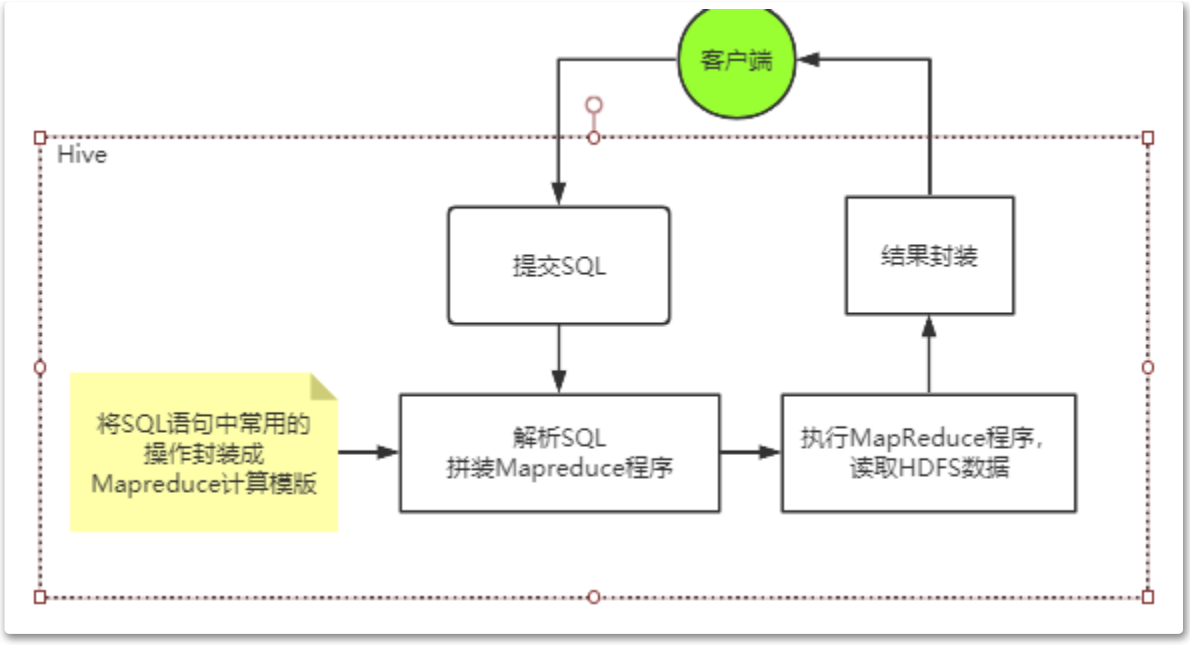
1、Hive是什么

Hive是由Facbook开源的一个解决海量结构化日志的数据统计工具，是Apache的一个顶级项目。官网地址： http://hive.apache.org/ 。我们要了解一个组件，官网的介绍是最重要的：

The Apache Hive ™ data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage using SQL. Structure can be projected onto data already in storage. A command line tool and JDBC driver are provided to connect users to Hive.

Hive提供了一种使用SQL语句来读、写、管理基于分布式系统的大型数据的功能。可以基于已有的数据进行部署。Hive给用户提供了一个命令行工具以及JDBC驱动用来连接Hive。

通常情况下， Hive是基于Hadoop的一个数据仓库工具，他可以将hdfs上的结构化数据文件映射成一张表，并提供以类SQL语句(HQL语句)进行查询统计的功能。而他的本质，其实就是将SQL语句转化成为模版化了的Mapreduce程序。整体的一个流程是这样。



所以Hive不存储数据，自己也没有任何计算功能，只是相当于类SQL语句与Hadoop文件之间的一个解释器。他本质上只是一个对HDFS上的文件进行索引与计算的工具。他需要依赖Hadoop的Yarn来进行资源分配，也需要Hadoop的MapReduce来提供计算支持。后面我们会知道，hive在进行数据计算时，不仅可以用MapReduce来支持，也可以集合其他更灵活，更高效的大数据计算框架。

2、Hive的适用场景

从上面的介绍可以简单掌握Hive的特点：

优点：
1、使用SQL语法查询，不用再去写复杂的MapReduce程序，减少开发成本，上手快。
2、基于MapReduce算法，所以在处理大数据时，优势非常明显。
3、可以支持自定义函数，计算能力强。

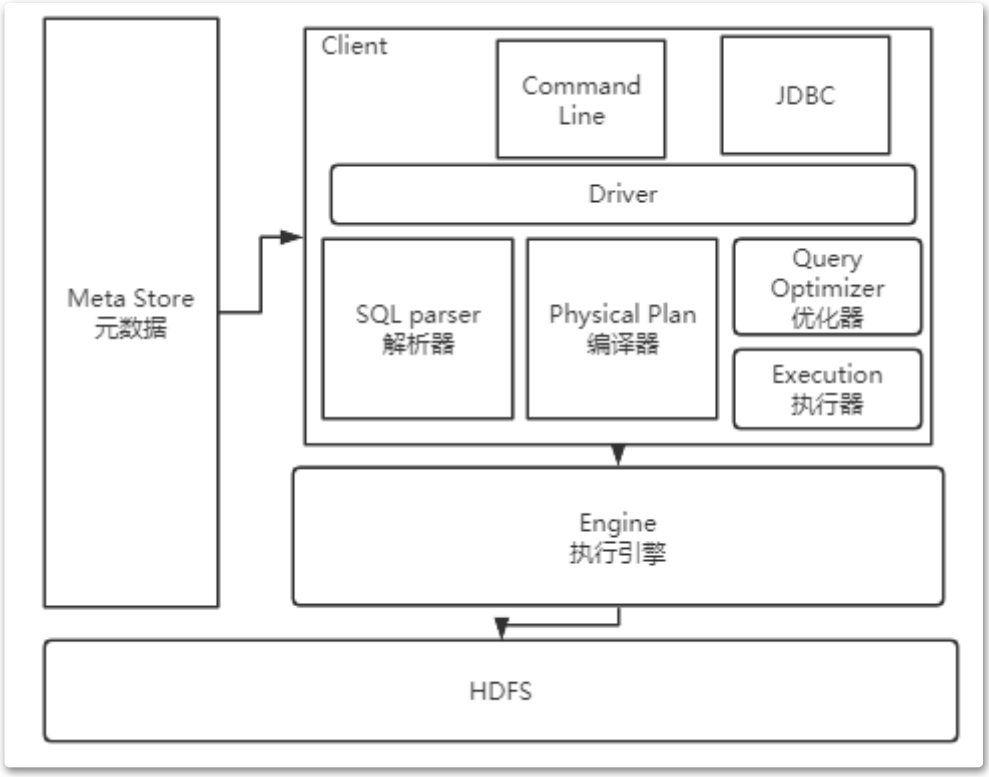
缺点：
1、Hive的执行延迟比较高。这是因为启动并运行一个MapReduce程序本身需要消耗非常多的资源。
2、Hive的HQL语句表达能力有限，并且他是基于模版实现的，所以通常不够智能化。很多复杂的大数据计算无法支持。比如迭代式计算。
3、Hive处理大数据量非常擅长，但是处理小数据量就没有优势。

所以，Hive通常适用于大数据的OLAP场景，做一些面向分析，允许有延迟的数据挖掘工作。并且结合其他组件也可以用来做一些数据清洗之类的简单数据处理工作。

Hive是针对数据仓库来进行设计的，这种场景下，通常是读多写少。并且数据都是来自于外部的HDFS，所以Hive中不建议做数据的修改操作，所有的数据最好是在加载的时候就已经确定好了。

3、Hive的整体架构

Hive的整体架构大概是这样：



1、用户接口：Hive提供了Command Line命令行工具，JDBC接口的方式来管理数据，并且也提供了WebUI，在浏览器访问hive。

2、元数据： 用来记录Hive中数据组织方式的数据。比如库名、表名、数据所在目录等。元数据默认是存储在自带的derby数据库中。derby是一个本地单机数据库，不利于元数据共享。通常情况下，会使用MqSQL来存储元数据。

3、驱动器Driver：通过一系列的组件来执行SQL。

首先通过SQL parser解析器将客户端的SQL语句解析成抽象语法树AST(Abstract Syntax Tree)。这一步通常会用一些第三方工具来完成，比如antlr。

接下来 Physical Plan 编译器将ASTA树编译生成具体的逻辑执行计划

然后 Query Optimizer 优化器对逻辑执行计划进行优化。

最后 把逻辑执行计划转换成为可以运行的物理计划。对于Hive，物理计划就是一个MapReduce计算程序。

4、Execution Engine 执行引擎：Hive将生成的MapReduce计算程序提交给外部的计算工具执行。默认Hive是使用的Hadoop的Mapreduce来执行，另外，hive也支持tez和spark。而计算的所有数据来源最终全都来自于Hdfs。

4、为什么要用Hive而不用关系型数据库?

这是面试中最喜欢问的问题。Hive是基于Hadoop的，所以Hive的可扩展性与Hadoop的可扩展性是一致的。而Hadoop相比于传统数据库：

一方面他的扩展能力相当强，Hadoop支持将不同配置的机器一起组成庞大的集群。目前世界上最大的Hadoop集群在Yahoo，有超过5000台节点。而关系型数据库相比就差很远了。例如，Oracle理论上的扩展能力也只有100台左右，这还是基于高成本服务器组成的理论环境。

另一方面他的数据处理能力也相当强。当数据量非常大时，关系型数据库虽然也可以用分库分表等方式做一些扩展，但是数据承载能力始终相当有限，并且数据量大了之后，性能下降明显。而基于Hadoop的大数据体系承载海量数据就轻松很多。数据量增大对整体性能的影响非常有限。

二、Hive安装

1、下载地址与版本

Hive可以从官网快速下载最新版本。

下载地址： http://hive.apache.org/downloads.html 。

官网上有一个指导手册： https://cwiki.apache.org/confluence/display/Hive/LanguageManual。

github官方仓库： https://github.com/apache/hive

Hive有两个大版本，2.x的版本是针对Hadoop 2.x.y开发的。目前最新版本2.3.8.而3.x的版本是针对Hadoop3.x.y版本开发的，目前最新的是3.1.2。我们这次就会基于3.1.2来学习Hive。

2、Hive安装部署

Hive是基于Hadoop使用的，所以安装Hive之前，Hadoop和JDK是必须要安装的。另外，mysql和spark也是Hive通常会使用到的组件。所以mysql 和spark最好也安装上。这些组件的安装就不多说了。

实验环境

三台机器，/etc/hosts文件中的机器名分别配置为hadoop01,hadoop02,hadoop03(集群都是通过机器名来配置的， 只要网络没问题， IP不重要)。上面已经安装了hadoop集群和spark集群。另外，在hadoop01上安装了mysql数据库。

安装Hive

1、把apache-hive-3.1.2-bin.tar.gz上传到hadoop01上。并解压到/app/hive/目录。

然后，在hive中会有多个log4j的日志jar包冲突，可以选择删除(可选操作)

```
mv /app/hive/lib/log4j-slf4j-impl-2.10.0.jar /app/hive/lib/log4j-slf4j-impl-2.10.0.jar.bak
```

2、将hive 下的conf目录下的hive-env.sh.template文件复制为hive-env.sh

```
mv /app/hive/conf/hive-env.sh.template /app/hive/conf/hive-env.sh
```

3、配置hive-env.sh 主要是配置 Hadoop的安装地址和Hive的安装地址。

```
vi hive-env.sh
    在文件最下面添加指定两个变量
export HADOOP_HOME=/app/hadoop/hadoop-3.2.2/
export HIVE_CONF_DIR=/app/hive
```

4、配置环境变量HIVE_HOME

在~/.bash_profile中配置HIVE_HOME的环境变量

```
export HIVE_HOME=/app/hive
PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$JAVA_HOME/bin:$HIVE_HOME/bin:$HOME/bin
```

5、重建schema

```
cd /app/hive
mv metastore_db/ metastore_db_bak
bin/schematool -dbType derby -initSchema
```

> 这里使用默认的derby本地数据库来存放hive的元数据。这样配置也看不到配了些什么。后面会将元数据调整到mysql。

这样，Hive的安装基本就完成了。通常Hive不像Hadoop、Spark那样需要搭建集群，而Hive只是相当于Hadoop的一个客户端，所以他并不需要部署集群。通常单机就足够了。

6、启动hive

接下来可以启动hive，简单使用一下。 启动Hive之前一定要注意Hadoop的hdfs和yarn，必须是启动的。

然后，需要在hdfs上创建两个hive默认使用的目录。

```
hadoop fs -mkdir /tmp
hadoop fs -chmod g+w /tmp
hadoop fs -mkdir -p /user/hive/warehouse
hadoop fs -chmod g+w /user/hive/warehouse
```

这样就可以启动Hive了

```
[root@192-168-65-174 hive]# pwd
/app/hive
[root@192-168-65-174 hive]# bin/hive
which: no hbase in
(:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/usr/local/es/node-v8.1.0-linux-x64/bin:/app/hadoop/hadoop-3.2.2//bin:/app/hadoop/hadoop-3.2.2//sbin:/app/java/jdk1.8.0_212//bin:/root/bin)
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/app/hive/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/app/hadoop/hadoop-3.2.2/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive Session ID = e6dc942e-9a79-4683-a681-ce697e9a5887

Logging initialized using configuration in jar:file:/app/hive/lib/hive-common-3.1.2.jar!/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
```

```
hive>
```

> 1、从这段文字中能看到一些提示信息，例如hive一开始会去找有没有hbase，因为hive可以与hbase互通数据，这个我们后面会介绍。另外当前hive版本已经不建议使用Hive-On-MR，建议使用tez或者spark了。
>
> 2、启动时有可能会报错Exception in thread "main" java.lang.NoSuchMethodError: com.google.common.base.Preconditions.checkArgument(ZLjava/lang/String;Ljava/lang/Object;)V
>
> 这是因为guava的jar包版本太低。这时，可以从hadoop中复制一个高版本的guava的jar包替换过来就行了。
>
> cp /app/hadoop/hadoop-3.2.2/share/hadoop/common/lib/guava-27.0-jre.jar /app/hive/lib/
> rm -f /app/hive/lib/guava-19.0.jar

基本操作

对数据库的简单操作

```
hive> show databases;
hive> show tables;
hive> create table test(id int);
hive> insert into test values(1);
hive> select * from test;
```

这个时候，如果另外启动一个窗口，也在/app/hive目录下执行bin/hive指令连接hive，会报错。

这是因为hive使用的derby作为元数据库，开启hive之后就会占用元数据库，并且不能与其他客户端共享。所以接下来我们还是将hive的元数据地址改为mysql。

使用mysql作为hive的元数据

实验环境已经在hadoop01机器上安装好了mysql。mysql的安装过程就略过了。

> 如果安装mysql有麻烦，推荐使用宝塔面板，可以简化mysql数据库的安装。

首先需要获取mysql的jdbc驱动包mysql-connector-java-8.0.23.jar(与mysql版本对应)，放到hive的lib目录下。 hive默认是不带jdbc驱动包的。

然后进入$HIVE_HOME/conf目录，新建一个配置文件hive-site.xml。添加以下内容：

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <!-- jdbc 连接的 URL -->
    <property>
        <name>javax.jdo.option.ConnectionURL</name>
        <value>jdbc:mysql://hadoop01:3306/metastore?useSSL=false</value>
    </property>
    <!-- jdbc 连接的 Driver-->
    <property>
        <name>javax.jdo.option.ConnectionDriverName</name>
        <value>com.mysql.cj.jdbc.Driver</value>
    </property>
    <!-- jdbc 连接的 username-->
    <property>
        <name>javax.jdo.option.ConnectionUserName</name>
        <value>root</value>
    </property>
    <!-- jdbc 连接的 password -->
    <property>
        <name>javax.jdo.option.ConnectionPassword</name>
        <value>root</value>
    </property>
    <!-- Hive 元数据存储版本的验证 -->
    <property>
        <name>hive.metastore.schema.verification</name>
        <value>false</value>
    </property>
    <!--元数据存储授权-->
    <property>
        <name>hive.metastore.event.db.notification.api.auth</name>
        <value>false</value>
    </property>
    <!-- Hive 默认在 HDFS 的工作目录 -->
    <property>
        <name>hive.metastore.warehouse.dir</name>
        <value>/user/hive/warehouse</value>
    </property>
</configuration>
```

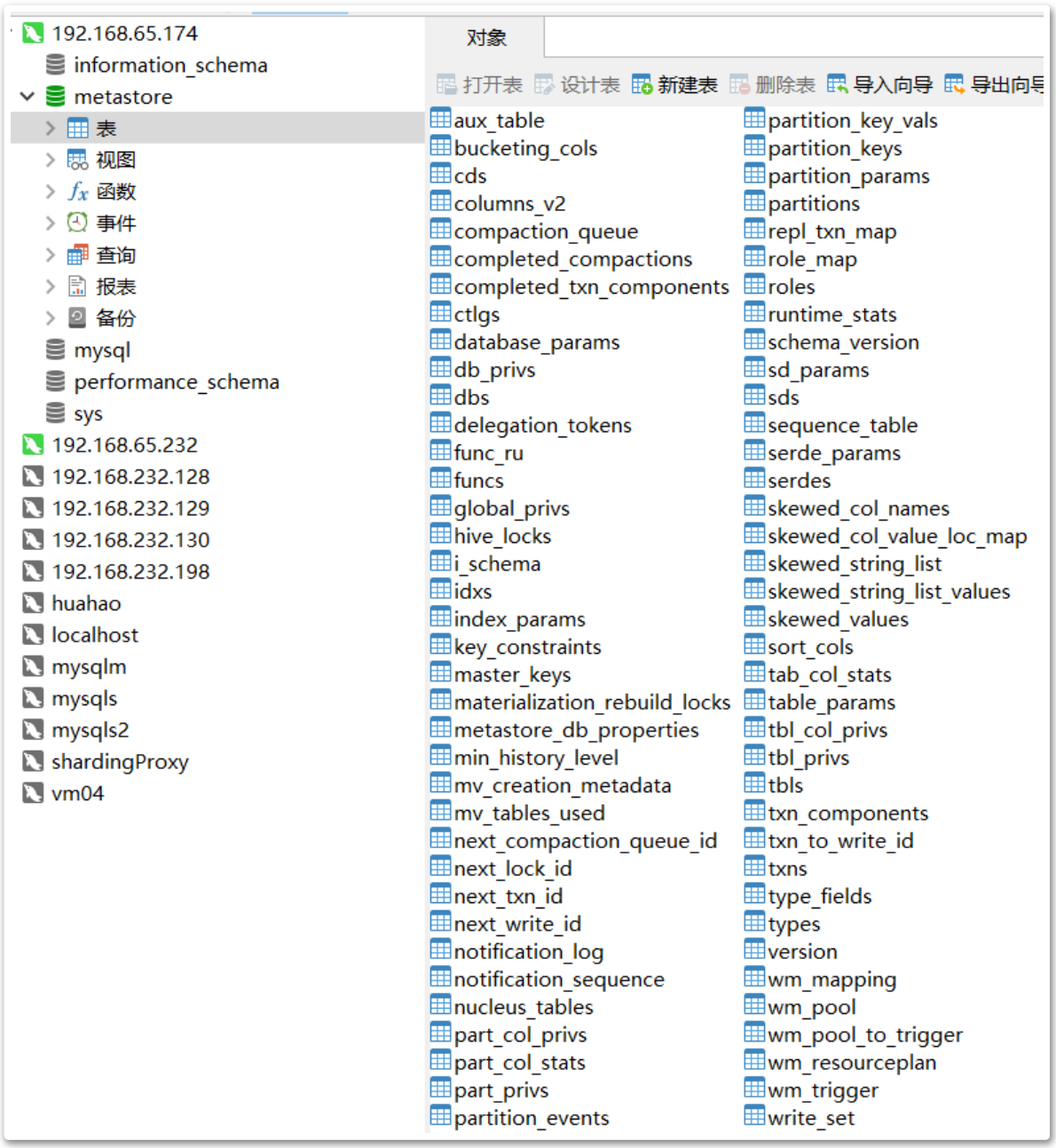
> 注意相关MySQL的配置需要符合你的实际情况。

接下来登入mysql，创建一个metastore数据库。

然后初始化hive元数据库：

```
[root@192-168-65-174 hive]# bin/schematool -initSchema -dbType mysql --verbose
```

执行完成后，可以在数据库中看到创建了一堆表，这些表就是Hive的元数据信息。



> 现在先不用管这些表是什么意思。后面会带大家进来看一部分。

这样我们就将Hive的元数据从Derby转移到了MySQL。在SecureCRT中打开多个窗口也不会报错了。

使用JDBC连接远程Hive服务

到目前为止，Hive还只是一个单机的服务，只能在服务器上进行本地连接，其他服务器无法访问。这时可以通过hive提供的元数据服务，将Hive提升为一个对外服务。

启动服务

在Hive的conf/hive-site.xml中添加配置信息：

```
    <!-- 指定存储元数据要连接的地址 -->
    <property>
        <name>hive.metastore.uris</name>
        <value>thrift://hadoop01:9083</value>
    </property>
    <!-- 指定 hiveserver2 连接的 host -->
    <property>
        <name>hive.server2.thrift.bind.host</name>
        <value>hadoop01</value>
    </property>
    <!-- 指定 hiveserver2 连接的端口号 -->
    <property>
        <name>hive.server2.thrift.port</name>
        <value>10000</value>
    </property>
```

然后启动Hive的元数据服务

```
hive --service metastore
```

接下来还要启动一个hiveserver2服务。

```
hive --service hiveserver2
```

> 这两个服务都是前台服务，会占据当前命令行窗口。当然也可以通过 nohup hive --service metastore & 将他转到后台。

然后就可以使用Hive的Beanline客户端以JDBC的方式来接Hive了。但是这时要注意，如果配置了元数据服务的端口，那在hive指令运行时，就会去连接元数据服务。

连接Hive服务

使用Hive的beeline连接Hive服务：

```
[root@192-168-65-174 hive]# beeline -u jdbc:hive2://hadoop01:10000 -n root
Connecting to jdbc:hive2://hadoop01:10000
Connected to: Apache Hive (version 3.1.2)
Driver: Hive JDBC (version 3.1.2)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 3.1.2 by Apache Hive
0: jdbc:hive2://hadoop01:10000>
```

出现这个提示符，就表示beeline连接hive成功了。

> 可能出现的错误:
>
> 1、User: root is not allowed to impersonate root (state=08S01,code=0) 出现这个错误就需要在hadoop的core-site.xml中添加以下配置：
>
> ```
> <property>
>  <name>hadoop.proxyuser.root.hosts</name>
>  <value>*</value>
> </property>
> <property>
>  <name>hadoop.proxyuser.root.groups</name>
>  <value>*</value>
> </property>
> ```
>
> 然后重启Hadoop。
>
> 2、命令行最后的 -n root 是需要指定hadoop的用户。如果不带这个参数，会报错：
>
> > Error: Could not open client transport with JDBC Uri: jdbc:hive2://hadoop01:10000: Failed to open new session: java.lang.RuntimeException: org.apache.hadoop.security.AccessControlException: Permission denied: user=anonymous, access=EXECUTE, inode="/tmp":root:supergroup:drwxrwx---
>
> 这是因为Hadoop的权限问题。需要通过-n指定hdfs文件所属用户。我的Hadoop是使用root用户直接安装的。

> 3、另外要注意，一旦配置了Hive的服务，那在使用hive之前就必须要启动服务才行。即使是使用本地的hive指令连接，也需要服务支持。

Hive配置总结

到这里，我们就已经完成了Hive的安装以及服务部署。现在回过头来对安装部署的过程中的一些问题进行梳理：

1、hive指令的其他用法

我们多次用到了hive指令，这里就需要梳理一下hive指令到底有哪些功能。

```
[root@192-168-65-174 ~]# hive -help
which: no hbase in
(:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/usr/local/es/node-v8.1.0-linux-x64/bin:/app/hadoop/hadoop-3.2.2//bin:/app/hadoop/hadoop-3.2.2//sbin:/app/java/jdk1.8.0_212//bin:/app/hive/bin:/root/bin)
Hive Session ID = 9c5ee6f7-fb92-4a84-bcd8-5bd54f191940
usage: hive
 -d,--define <key=value>          Variable substitution to apply to Hive
                                  commands. e.g. -d A=B or --define A=B
    --database <databasename>     Specify the database to use
 -e <quoted-query-string>         SQL from command line
 -f <filename>                    SQL from files
 -H,--help                        Print help information
    --hiveconf <property=value>   Use value for given property
    --hivevar <key=value>         Variable substitution to apply to Hive
                                  commands. e.g. --hivevar A=B
 -i <filename>                    Initialization SQL file
 -S,--silent                      Silent mode in interactive shell
 -v,--verbose                     Verbose mode (echo executed SQL to the
                                  console)
```

这里面，用得比较多的是 -e 和 -f 两个指令。 -e 指令可以直接执行一个SQL，例如 hive -e "select * from student;"。而 -f 可以指定一个SQL文件，并且通常我们还会使用Linux指令将输出结果重定向到一个文件。 例如 hive -f /root/hivetest.sql > /root/hivetestResult。

然后，我们还用过了hive --service 指令来启动hive的元数据服务。我们可以这样来看一下hive中内置了多少服务：

```
[root@192-168-65-174 ~]# hive --service -help
which: no hbase in
(:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/usr/local/es/node-
v8.1.0-linux-x64/bin:/app/hadoop/hadoop-3.2.2//bin:/app/hadoop/hadoop-
3.2.2//sbin:/app/java/jdk1.8.0_212//bin:/app/hive/bin:/root/bin)
Service -help not found
Available Services: beeline cleardanglingscratchdir cli fixacidkeyindex help
hiveburninclient hiveserver2 hplsql jar lineage llapdump llap llapstatus
metastore metatool orcfiledump rcfilecat schemaTool strictmanagedmigration
tokentool version
```

例如，我们就可以使用version服务查看hive的版本信息：

```
[root@192-168-65-174 ~]# hive --service version
Hive 3.1.2
Git git://HW13934/Users/gates/tmp/hive-branch-3.1/hive -r
8190d2be7b7165effa62bd21b7d60ef81fb0e4af
Compiled by gates on Thu Aug 22 15:01:18 PDT 2019
From source with checksum 0492c08f784b188c349f6afb1d8d9847
```

2、hive的日志配置

hive的log日志默认存放在/tmp/root/hive.log文件中(跟用户名有关)。

如果要修改hive跟日志相关的配置，可以将$HIVE_HOME/conf下的hive-log4j2.properties.template复制成为hive-log4j2.properties，然后对里面的配置进行定制。例如hive日志文件的地址配置在property.hive.log.dir属性，而日志文件名配置在property.hive.log.file属性。

3、hive的关键配置

hive相比hadoop、spark这些组件来说还是简单很多，所以他的核心配置也少很多。他的所有核心配置属性，都在$HIVE_HOME/conf/hive-default.xml.template文件中，这里面有所有的常用配置信息以及描述，是很重要的优化Hive的参考。比如最重要的这个配置：

```
 <property>
    <name>hive.execution.engine</name>
    <value>mr</value>
    <description>
      Expects one of [mr, tez, spark].
      Chooses execution engine. Options are: mr (Map reduce, default), tez, spark. While MR
      remains the default engine for historical reasons, it is itself a historical engine
      and is deprecated in Hive 2 line. It may be removed without further warning.
    </description>
  </property>
```

这个配置就是hive的执行引擎，默认是使用Hadoop的MapReduce。同时也支持Tez和Spark。后面我们会有课程讲解Hive与Spark、Hbase这些组件的合作，这里就是很重要的一个配置。

hive相比于Hadoop、Spark这些大数据组件，部署和使用都相对简单很多。但是Hive对环境的依赖相当多，因此，部署过程还是很容易出问题的。有时间还是需要多动手熟悉一下。

三、Hive的基础使用

Hive支持以类SQL的HQL语句来对数据进行管理。理论上，Hive支持对数据通用的增删改查操作，但是在实际使用过程中，Hive更多的是作为一个数据索引工具，对已有的数据进行索引。所以在Hive的使用过程中，更多的是使用DDL语句建表来对数据进行映射，然后使用查询语句对数据进行查询。而对数据的修改、删除等管理功能，虽然Hive也支持，但是一般很少用。

1、Hive的数据结构

SQL语句我们都熟悉，要使用SQL需要先建表。Hive的基础数据类型如下表：

Hive数据类型	JAVA数据类型	长度
TINYINT	byte	1byte有符号整数
SMALINT	short	2byte有符号整数
INT	int	4byte有符号整数
BIGINT	long	8byte有符号整数
BOOLEAN	boolean	true或者false

Hive数据类型	JAVA数据类型	长度
FLOAT	float	单精度浮点数
DOUBLE	double	双精度浮点数
STRING	string	字符串。可以指定字符集
TIMESTAMP		时间类型
BINARY		字节数组

其中要注意的是加粗的那两个。

首先对于整数，hive提供了好几种类型，这几个类型的本质区别其实就在整数的范围。而Hive的这几个整数类型，其实也不是强制的。在Hive中提供了一种隐式转换的功能，所有整数类型都可以隐式的转换成一个范围更广的类型。比如TINYINT可以转换成INT， INT可以转换成BIGINT。另外，整数类型、FLOAT和STRING类型也都可以隐式转换成为DOUBLE。并且，在这些数据类型中，hive也提供了cast函数来进行强转。 例如 select ('1' as int) +3。 这里就会将字符串1转换成数字1，这样就能进行数学计算了。但是，如果强转失败就会返回null，比如select ('x' as int)。

然后对于字符串类型，他不像MySQL的varchar，不需要指定长度。Hive中的SRING类型是一个可变的字符串，理论上他可以存储2GB的字符数。

接下来，我们就可以使用beeline工具来一个简单的建表语句试试。

```
create table userinfo(
    name string,
    age int,
    salary bigint
);
insert into userinfo(name,age,salary) values('roy',18,1000);
select * from userinfo;
```

> 注意，如果复制粘贴到命令行，最好不要用tab做缩进。直接用空格。

依次执行这些语句，会发现插入语句执行的非常慢，并且日志中看到，他会启动一个MapReduce计算，效率是非常低的。至于查询，后面会讲到，hive对简单查询可以通过文件索引直接做，但是复杂查询也还是要启动MapReduce计算的，那样查询速度就会慢很多了。

执行完这些语句后，我们来看看Hive是怎么保存数据的。

首先，到hive的Mysql元数据中去看看，里面有个tbls表，记录了建表的情况。而在tab_col_stats表中，记录了数据列的信息。元数据的表结构我们不需要过多了解，这里我们只需要大概知道，hive的元数据中保存了数据库的完整表结构。但是这里面并没有保存具体的数据。

然后，到hdfs中去看下，其实这个建表语句就是在hdfs上创建了一个/user/hive/warehouse/userinfo目录。而插入数据后，在这个目录下就会出现一个000000_0的文件，里面的内容就是我们插入的数据。

所以，从这个示例中，我们可以知道hive大概的工作机制；就是在元数据中保存数据结构，而数据全部保存在hdfs上。当然，Hive具体如何工作的，后面会有更详细的讲解。

2、Hive的复杂数据结构

在几种基础类型之外，Hive还提供了三种复杂类型，用于映射更复杂的文本数据结构。

数据类型	描述	示例
STRUCT	类似于JAVA中的POJO对象，以结构化的多个属性组成一个共有的数据。	struct< street:string,city:string >
MAP	类似于JAVA中的MAP，表示键值对组成的元组集合。	map<string,int>
ARRAY	类似于JAVA中的数组。表示一组具有相同类型和名称的变量集合。数组中的对象可以通过一个从0开始的index直接访问。	array ["A","B","C"]

下面来一个例子理解下HIVE的复杂类型：

例如，有一组数据，结构用JSON表示是这样的：

```
{
    "name": "roy",
    "friends": ["bob" , "john"] , //列表 Array,
    "children": { //键值 Map,
        "yula": 6 ,
        "sophia": 17
    }
    "address": { //结构 Struct,
        "street": "lugu",
        "city": "changsha"
    }
}
```

我们需要以一个文本的形式来保留这样的数据，就会创建这样的一个测试文件test.txt。

```
roy,bob_john,yula:6_sophia:17,lugu_changsha
mike,lea_jacky,rita:7_king:20,chao yang_beijing
```

这个表述的形式就是一行代表一条数据，属性之间用逗号分隔，然后数组之间用下划线分隔，键值对之间用冒号分隔。

然后，就需要在hive中建表，来映射这样的文本结构：

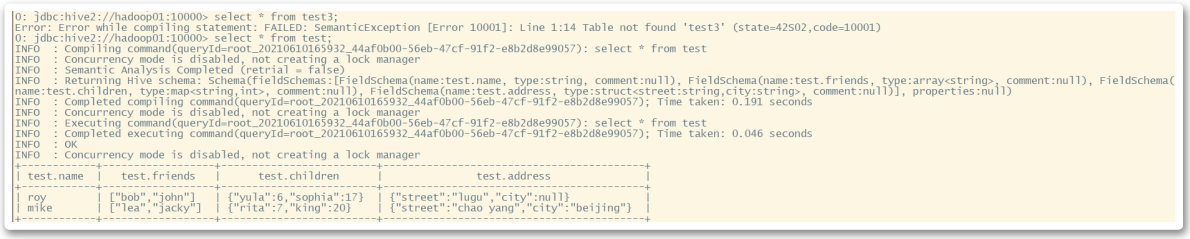
```
create table test3(
name string,
friends array<string>,
children map<string, int>,
address struct<street:string, city:string>
)
row format delimited fields terminated by ','
collection items terminated by '_'
map keys terminated by ':'
lines terminated by '\n';
```

> 这个建表语句就比较复杂了，但是对照我们的数据结构，还是不难理解。
>
> row format delimited fields terminated by ','表示列分隔符
> collection items terminated by '_' 表示 Map、Struct和Array的分隔符
> map keys terminated by ':' Map中key和value的分隔符
> lines terminated by '\n'; 一行数据的分隔符。

然后，可以把test.txt文本直接上传到hdfs中test表所在的目录

```
hadoop fs -put test.txt /user/hive/warehouse/test
```

接下来，直接在hive中执行select * from test3，查看所有的数据。



```
0: jdbc:hive2://hadoop01:10000> select * from test3;
Error: Error while compiling statement: FAILED: SemanticException [Error 10001]: Line 1:14 Table not found 'test3' (state=42S02,code=10001)
0: jdbc:hive2://hadoop01:10000> select * from test;
INFO  : Compiling command(queryId=root_20210610165932_44af0b00-56eb-47cf-91f2-e8b2d8e99057): select * from test
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Returning Hive schema: Schema(fieldSchemas:[FieldSchema(name:test.name, type:string, comment:null), FieldSchema(name:test.friends, type:array<string>, comment:null), FieldSchema(
name:test.children, type:map<string,int>, comment:null), FieldSchema(name:test.address, type:struct<street:string,city:string>, comment:null)], properties:null)
INFO  : Completed compiling command(queryId=root_20210610165932_44af0b00-56eb-47cf-91f2-e8b2d8e99057); Time taken: 0.191 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=root_20210610165932_44af0b00-56eb-47cf-91f2-e8b2d8e99057): select * from test
INFO  : Completed executing command(queryId=root_20210610165932_44af0b00-56eb-47cf-91f2-e8b2d8e99057); Time taken: 0.046 seconds
INFO  : OK
INFO  : Concurrency mode is disabled, not creating a lock manager
+------------+--------------+-----------------------+-----------------------------------------+
| test.name  | test.friends |    test.children      |              test.address               |
+------------+--------------+-----------------------+-----------------------------------------+
| roy        | ["bob","john"]  | {"yula":6,"sophia":17}  | {"street":"lugu","city":null}           |
| mike       | ["lea","jacky"] | {"rita":7,"king":20}    | {"street":"chao yang","city":"beijing"} |
+------------+--------------+-----------------------+-----------------------------------------+
```

这时就可以直接在hive中查到数据了。到这里，我们一方面可以了解hive如何去映射复杂的数据结构，另一方面也能更进一步了解hive的工作机制。刚才的过程中，hive中的建表语句只是在mysql元数据中维护了test表的结构信息，而具体的数据，hive并没有自己进行维护，只是去hdfs的对应目录去读取(而且，就算要写insert语句，这样的复杂结构似乎也不好写)。其实，这也是hive最常用的工作方式，即数据已经采集到了hdfs上，然后再用hive去针对文本结构进行表映射。

> 当然，要完整这样的映射，我们这个简单的例子还是不够的，比如文件地址、数据来源等，hive都提供了更复杂的映射方式，后面也会有更具体的讲解，在这里，我们只需要理解hive的工作机制。

接下来，可以尝试下访问这些复杂结构中的各个数据。

```
select name,friends[1],children['yula'],address.city  from test3;
```

这些复杂结构的访问方式也不用强记，大都跟java中对应类型的访问方式类似。

小结：

从这个简单的示例中，我们了解了hive的数据类型，能够在hive中正确维护表结构。另外，也基本上理解了hive的工作方式。

hive的元数据中维护表结构，而表结构本质上就是对文件的一种映射关系。具体的数据以文件的形式放在hdfs对应的目录里。hive中的一个表实际上就对应hdfs中的一个目录，hive中的hql语句操作的就是这个目录中的文本文件。

而在hive的所有功能当中，其实最重要的就是通过DDL建表语句来保留对数据文件的映射关系，然后通过查询语句来对数据文件进行统计。而对数据的修改则不是hive的强项，通常都是由其他大数据计算框架，比如spark、hadoop、flink等这些框架来计算。

那接下来，我们就从DDL和查询这两个方向，继续深入学习下hive的使用方式。

四、Hive-DDL

1、维护数据库

跟mysql类似，hive中也有库的概念。默认会创建一个default数据库。

Hive中的完整建库语句如下

```
CREATE DATABASE [IF NOT EXISTS] database_name
[COMMENT database_comment]
[LOCATION hdfs_path]
[WITH DBPROPERTIES (property_name=property_value, ...)];
```

这其中，中括号的部分都是可选指令。比如 if not exists表示当前库不存在的时候才创建，存在时就不去覆盖已有的数据库。

我们执行这样的语句：

```
create database db_hive;      --创建库
show databases;     --查询所有库
desc database db_hive;     -- 描述库 这里可以看到对应的hdfs路径。
```

从这里可以看到，hive中创建的库，和表相似，也是对应一个hdfs上的路径。唯一的不同，只是这个目录名字是在数据库名后加了个.db后缀。 后面使用use db_hive，就可以切换到这个数据库，在这个库里的表就会创建在自己对应的目录里。

然后，删除库的语句：

```
drop database db_hive; --删除库
drop database db_hive cascade; --强制删除
```

这些基本的语句都跟mysql很类似。而hive中的库与mysql中的库也有点区别，hive中的库可以添加一些属性。

```
alter database db_hive set dbproperties('creatime'='20210610');   --添加属性
desc database extended db_hive;   --查看数据库信息，extended就可以查到设置的这些属性。
```

2、维护表

表可以认为是Hive中最为重要的一个组件，因为他实际上维护了hive中的表与hdfs上的数据文件映射关系，数据文件的形式是很多样的，所以hive中的表也需要设计得非常复杂，来映射各种不同的数据文件。

简单的建表语句之前已经接触过了，这里就先列出一下hive中完整的可选建表语句：

```
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]
[TBLPROPERTIES (property_name=property_value, ...)]
[AS select_statement]
```

同样，其中中括号的是可选项，其他就是必选的了。从中括号的数量也能看出hive中的表比库复杂多了。这些中括号中，有很多是直接一看就能明白的。比如

- [IF NOT EXISTS] 表示表不存在时才建表。
- [(col_name data_type)] 表示表的字段
- [COMMENT table_comment] 表的描述信息
- [ROW FORMAT row_format] 表的行格式，还包含了复杂数据结构的分隔符，这个之前已经接触过了。
- [LOCATION hdfs_path] 表对应的hdfs路径
- [TBLPROPERTIES (property_name=property_value, ...)] 表的额外属性
- [AS select_statement] AS后跟查询语句，根据查询语句的结果建表
- [STORED AS file_format] 表的文件存储格式。有三种格式可选：TESTFILE(文本)，SEQUENCEFILE（二进制序列文件）、RCFILE（列式存储格式文件）。最

常用的就是文本类型，这样可以在hdfs上直接看到数据内容。而如果需要压缩，可以使用SEQUENCEFILE。

而复杂点的是下面这几个选项。

- [EXTERNAL] 表示是不是外部表。
- [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)] 维护分区表
- [CLUSTERED BY (col_name, col_name, ...)] ,[SORTED BY (col_name [ASC|DESC], ...)] INTO num_buckets BUCKETS]。这两个可选项是维护分桶表

接下来简单介绍下hive中对表的维护。关于hive的外部表、分区表和分桶表，会在后面一一讲解。

重命名表：

```
ALTER TABLE table_name RENAME TO new_table_name;
```

更新列

```
ALTER TABLE table_name CHANGE [COLUMN] col_old_name col_new_name column_type
[COMMENT col_comment] [FIRST|AFTER column_name]
```

增加或替换列

```
ALTER TABLE table_name ADD|REPLACE COLUMNS (col_name data_type [COMMENT
col_comment], ...)
```

> add表示是增加列。replace columns 表示替换表的所有列，相当于对表结构进行重建。
>
> 并且对表结构的修改都只是修改hive的元数据，hdfs上的数据文件不会改动。

3、外部表

什么是外部表?

通常我们在hive中创建的表，包括我们之前创建的表，默认都是内部表或者称为管理表。hive在维护这些表时同样会去维护表的数据文件。**如果在hive中删除一个内部表，那表对应的hdfs文件也会删除。**而我们知道，hive最擅长的是对数据进行查询，而不是管理。他的数据通常需要与别的工具共享。而这种内部表，是不适合与其他工具共享数据的。

而外部表就表示hive并不真正拥有这些表中的数据。在hive中删除这张表，只会删除hive自己的元数据信息，而并不会真正删除hdfs上的文件。当然，如果在hive中删除数据，那hdfs文件中的数据还是会一起删除掉的。

例如使用hive做日志数据搜索，通常需要其他工具将应用的日志集中收集到HDFS文本文件中，然后再用hive来分析。这个时候，显然还是使用外部表比内部表更合适。其实在Hive的大部分应用场景中，外部表用得比内部表更多。

内部表与外部表的相互转换

1、查询表的类型：

```
0: jdbc:hive2://hadoop01:10000> desc formatted userinfo;
...
| Table Type:                   | MANAGED_TABLE
```

2、内部表修改为外部表：

```
0: jdbc:hive2://hadoop01:10000> desc formatted userinfo; alter table
userinfo set tblproperties('EXTERNAL'='TRUE');

0: jdbc:hive2://hadoop01:10000> desc formatted userinfo;
...
| Table Type:                   | EXTERNAL_TABLE
```

3、外部表修改为内部表：

```
0: jdbc:hive2://hadoop01:10000> alter table userinfo set
tblproperties('EXTERNAL'='FALSE');

0: jdbc:hive2://hadoop01:10000> desc formatted userinfo;
...
| Table Type:                   | MANAGED_TABLE
```

注意：('EXTERNAL'='TRUE')和('EXTERNAL'='FALSE')为固定写法，区分大小写！

4、分区表

之前我们已经知道，Hive中的一个表实际对应HDFS文件系统上的一个独立的文件夹，文件夹下的所有文本就是Hive中的表数据。而分区表同样也是对应HDFS上的一个文件夹，不过，在分区表中增加了分区的概念，每个分区对应文件夹中的一个子目录。利用分区表，可以把一个大的数据集根据业务需要分割成效的数据集。然后在查询时，也可以在where子语句中选择执行的分区。

建表：下面我们来建立一个分区表：部门信息以及他们的办公室。

```
create table deptinfo(
  deptno int, dname string, officeroom string
)
partitioned by (year string)
row format delimited fields terminated by ',';
```

> 注意，分区字段不能是表中已经存在的列。

建完表之后，可以到HDFS上看一下，此时并没有创建deptinfo表的目录。

查数据：然后我们在服务器本地准备几个数据文件

dept_1.txt

```
10,ACCOUNTING,700
15,RESEARCH,800
20,SALES,900
25,MANAGER,700
30,DEV,600
35,TEST,600
```

dept_2.txt

```
10,ACCOUNTING,1700
15,RESEARCH,1800
20,SALES,1900
25,MANAGER,1700
30,DEV,1600
35,TEST,1600
```

dept_3.txt

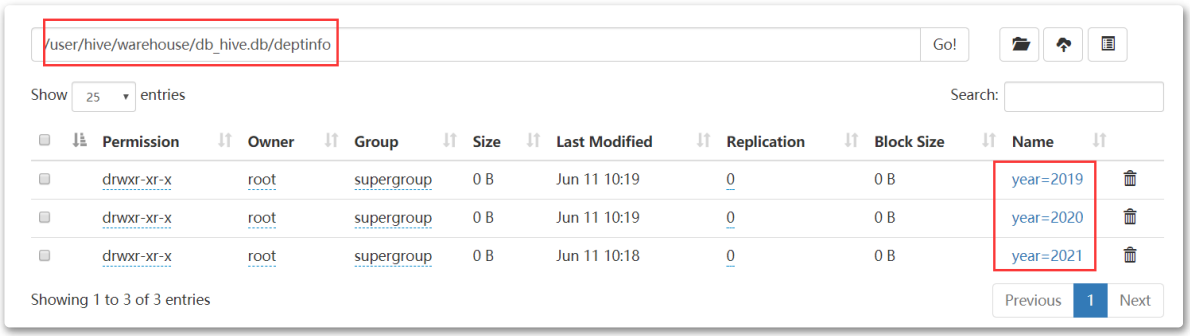
```
10,ACCOUNTING,2700
15,RESEARCH,2800
20,SALES,2900
25,MANAGER,2700
30,DEV,2600
35,TEST,2600
```

然后将文件导入分区表：

```
load data local inpath '/root/dept_1.txt' into table deptinfo partition (year='2021');
load data local inpath '/root/dept_2.txt' into table deptinfo partition (year='2020');
load data local inpath '/root/dept_3.txt' into table deptinfo partition (year='2019');
```

> 注意：分区表加载数据时，必须要指定分区

导入成功后，先到hdfs上看一下他是如何组织的hdfs数据文件：



他是以分区字段作为目录，然后将数据文件上传到了对应的目录。

可以先查一下deptinfo表的数据

```
select * from deptinfo;
select * from deptinfo where year='2021';
select * from deptinfo where year='2021' or year='2020';
```

可以看到，分区列可以像数据列一样进行查询。

管理分区：接下来可以用以下这些语句来对分区进行管理

```
#创建分区
hive>alter table deptinfo add partition(year='2022');
hive>alter table deptinfo add partition(year='2017')
partition(year='2018');
#删除分区
hive>alter table deptinfo drop
partition(year='2017'),partition(year='2018');
#查看分区表的分区情况
hive>show partitions deptinfo;
#查看表结构
hive>desc formatted deptinfo;
# Partition Information
# col_name              data_type               comment
year                    string
```

另外，在分区表的基础上，还可以对分区字段再继续做二级分区。例如我们这个示例中，已经按年进行了分区，在分区后，还可以添加一个按月的二级分区。

动态分区： 我们之前的分区表，都是以静态方式分配好分区。但是，实际使用时，我们往往希望就按照数据的某一个字段值来分区，并且事先也并不知道要分多少个区。比如之前的部门信息表，就希望按照officeroom字段来分区。这种就称为动态分区。

我们来创建另外一张分区表，对部门信息按照deptno进行分区。

```
create table deptinfo_dep(
  dname string, officeroom string
)
partitioned by (deptno int)
row format delimited fields terminated by ',';
```

然后我们尝试从deptinfo表把数据全部转移过来。

那如果可以指定deptno的分区键，就可以这样转移。这样Hive会给deptinfo_dep表创建一个deptno=70的分区。

```
insert into table deptinfo_dep partition(deptno='70') select
dname,officeroom from deptinfo; --执行mapreduce转移数据。
```

但是现在如果想要把deptinfo表的数据全部迁移过来，不知道要创建多少分区。这时要怎么写呢？可不可以这样写：

```
insert into table deptinfo_dep partition(deptno) select
dname,officeroom,deptno from deptinfo;
```

不指定分区键的值，然后在后面的select语句中，会默认以最后一个字段作为分区键去创建对应的分区。但是这个语句执行后，会报错：

```
hive> insert into table deptinfo_dep partition(deptno) select
dname,officeroom,deptno from deptinfo;
FAILED: SemanticException [Error 10096]: Dynamic partition strict mode
requires at least one static partition column. To turn this off set
hive.exec.dynamic.partition.mode=nonstrict
```

从字面上就能看到，这是因为Hive的动态分区默认是strict严格模式，这种模式下必须指定分区键的值。可以执行后面的set语句将hive.exec.dynamic.partition.mode属性的值调整为nonstrict。

那接下来在hive中执行这个语句 set hive.exec.dynamic.partition.mode=nonstrict 后，就可以执行上面的动态分区语句了。

然后在执行这个语句的过程中，会启动一个MapReduce程序来做计算，但是在日志的开头，还会有几个提示：

```
hive> insert into table deptinfo_dep partition(deptno) select
dname,officeroom,deptno from deptinfo;
.....
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
```

很显然，这几个参数是跟MapReduce任务相关的几个属性。比如我们这个场景，数据量非常小，那reduce其实就启动一个效率能高一点，其多个多个reduce反而降低了整个处理效率。那就可以通过这些参数来对MapReduce计算过程进行微调。

另外还有几个与动态分区有关的参数，也总结一下：

```
#是否开启动态分区功能，默认true
hive> set hive.exec.dynamic.partition;
hive.exec.dynamic.partition=true
```

```
#在所有执行MR的节点上，最大一共可以创建多少个动态分区。默认1000
hive> set hive.exec.max.dynamic.partitions;
hive.exec.max.dynamic.partitions=1000
#在每个执行MR的节点上，最大可以创建多少个动态分区。
hive> set hive.exec.max.dynamic.partitions.pernode;
hive.exec.max.dynamic.partitions.pernode=100
#整个MR Job中，最大可以创建多少个HDFS文件。
hive> set hive.exec.max.created.files;
hive.exec.max.created.files=100000
#当有空分区生成时，是否抛出异常
hive> set hive.error.on.empty.partition;
hive.error.on.empty.partition=false
```

最后，关于分区表，应该要注意到，采用静态分区时，大部分情况下都不会产生MR的计算过程，也就是说通过hive自己的元数据就能够处理简单的分区表。而采用动态分区后，大部分情况下都会产生MR的计算。这对资源和性能都是有损耗的，所以在使用过程中应该要酌情考虑。

5、分桶表

分区表提供了一个隔离数据和优化查询的遍历方式。不过，并非所有的数据集合都可以形成合理的分区。在分区的基础上，Hive可以进一步组织成桶，对数据文件进行更细粒度的数据范围划分。

分区针对的是数据的存储路径，而分桶针对的是数据文件。

创建分桶表： 按照ID分桶，分成4个桶。

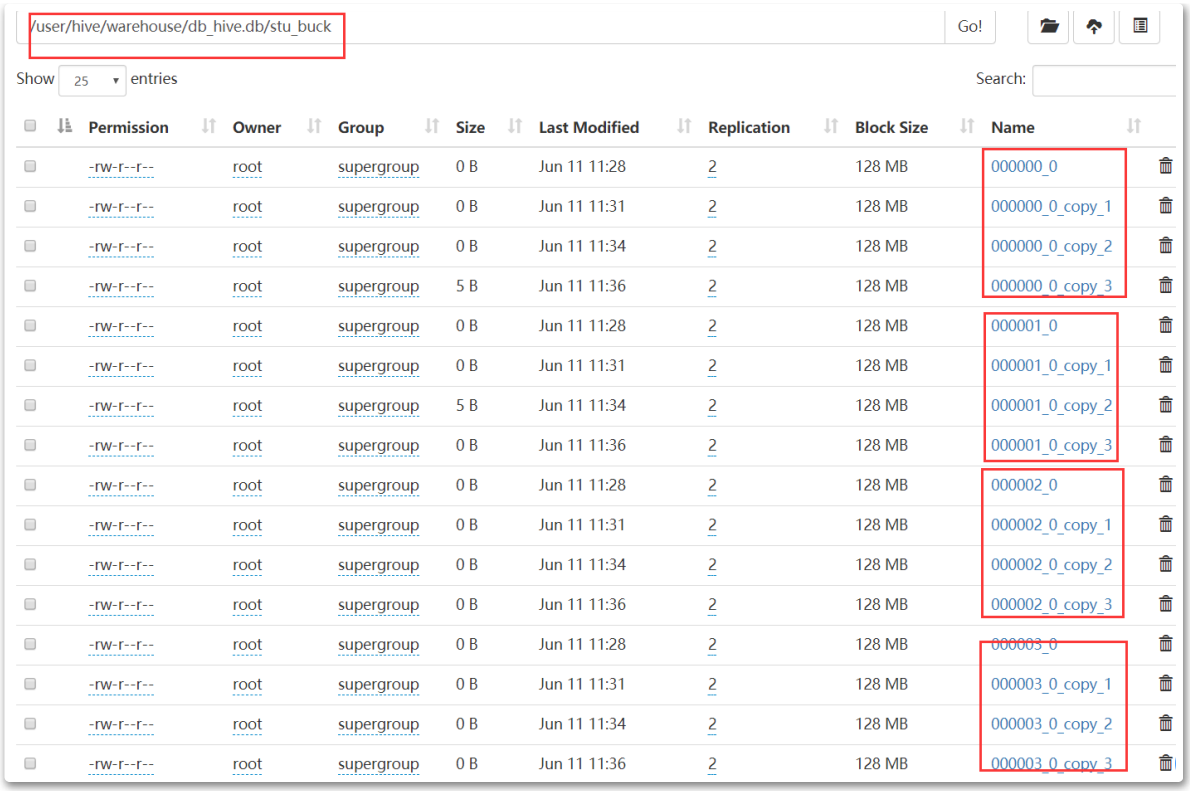
```
create table stu_buck(id int, name string)
clustered by(id)
into 4 buckets
row format delimited fields terminated by ',';
```

插入数据： 这次我们直接从hdfs把数据导入到hive。

```
hive> insert into stu_buck values (1,'s1');
hive> insert into stu_buck values (2,'s2');
```

关于分桶表，可以看到，简单的insert语句，也需要启动一个MapReduce任务来计算。这就是因为Hive的元数据已经不足以支撑分桶逻辑的判断了。

另外，在Hdfs上可以看到，分桶表对应了四个分桶文件，然后每次计算时，都会重新产生新的分桶文件。



Hive在分桶时， 会对分桶字段的值进行hash计算。然后对桶的个数取余，以此来决定记录存放在哪个桶里。在计算过程中，建议将reduce的个数设置为-1，这样可以让job自行决定需要用多少个reduce。

随机抽样： 对于非常大的数据集，Hive还支持按通进行随意抽样，以满足某些只需要少量有代表性数据的场景。

```
hive>select * from stu_buck tablesample(bucket 1 out of 4 on id);
```

这个tablesample表示在对id的四个分桶中抽取第一个分桶里的随机数据。

最后，补充一个指令，可以查看hive中的建表情况：

```
show create table stu_buck;
```

关于Hive的表结构管理，还有另外很重要的一块功能就是Hive与其他数据平台的数据互通，比如HBase。这会在后续的课程中讲解。

五、Hive-数据管理

通常来说，对数据的增删改查都是属于数据管理。但是对于Hive来说，对数据的增、删、改，往往都不是他的重点。而对于Hive来说，管理数据的方式重要的就是导入导出 和 查询两个方面。这一章节就来介绍下hive如何对数据进行导入导出操作。

向表中装载数据(Load)

这种方式之前已经接触过。

```
hive> load data [local] inpath 'path' [overwrite] into table
tablename [partition (partcol1=val1,…)];
```

这个指令中 load data表示加载数据。
local表示从本地加载数据，默认是从Hdfs加载数据
inpath表示加载数据的路径
override表示覆盖表中已有的数据，否则就表示追加数据。
partition表示上传到指定的分区。

在指定位置创建表

这种方式通常是针对Hdfs上已经有数据的场景。大部分情况下都是建为外部表，hive只做数据索引，不做数据管理。

```
create external table if not exists student5(
id int ,name string
)
row format delimited fields terminated by ','
location '/student';
```

数据导出

使用insert直接将查询结果导出到本地

```
#导出到服务器本地
insert overrite local directory '/app/hive/data/export/student' select * from student;
#格式化后导出到本地
insert overrite local directory '/app/hive/data/export/student1' ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' select * from student;
#导出到HDFS
insert overrite directory '/hive/data/export/student' ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' select * from student;
#在Hive中直接操作hdfs
dfs -get /usr/hive/warehouse/student/student.txt /app/hive/data/export/student.txt
```

然后还有一组导入导出的指令 export import ，多用于不同平台集群之间迁移Hive表。

```
export table db_hive.student to '/user/hive/warehouse/export/student';
import table student2 from '/user/hive/warehouse/export/student';
```

另外，对于Hive的数据导入导出，更多的场景不是在Hive之间转移数据，而是在不同数据平台之间转移数据，例如hive与mysql数据、与redis数据之间进行互相导入导出。这时候一般会用其他一些专门的数据转移工具，比如Sqoop。会在后续的其他课程中介绍。

六、Hive-查询

查询是Hive的重点功能。官方也专门有网页详细介绍Hive的查询功能。具体可参见官方的wiki

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select

完整的查询语法：

```
[WITH CommonTableExpression (, CommonTableExpression)*]    (Note: Only
available starting with Hive 0.13.0)
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
  FROM table_reference
  [WHERE where_condition]
  [GROUP BY col_list]
  [ORDER BY col_list]
  [CLUSTER BY col_list
    | [DISTRIBUTE BY col_list] [SORT BY col_list]
  ]
 [LIMIT [offset,] rows]
```

基本的SQL操作，比如增删改查、join连接、sort排序等功能基本跟标准SQL没什么区别，这里就不一一介绍了，下面找一些Hive比较有特色的查询功能介绍一下。

1、构建数据

我们准备两张测试的表：

```
--部门表
create table if not exists dept(
deptno int,
dname string,
loc int
)
row format delimited fields terminated by ',';
--员工表
create table if not exists emp(
empno int,
ename string,
job string,
mgr int,
hiredate string,
sal double,
comm double,
deptno int)
row format delimited fields terminated by ',';
```

然后准备两个数据文件：

dept

```
10,ACCOUNTING,1700
20,RESEARCH,1800
30,SALES,1900
40,OPERATIONS,1700
```

emp

```
7369,SMITH,CLERK,7902,1980-12-17,800.00,,20
7499,ALLEN,SALESMAN,7698,1981-2-20,1600.00,300.00,30
7521,WARD,SALESMAN,7698,1981-2-22,1250.00,500.00,30
7566,JONES,MANAGER,7839,1981-4-2,2975.00,,20
7654,MARTIN,SALESMAN,7698,1981-9-28,1250.00,1400.00,30
7698,BLAKE,MANAGER,7839,1981-5-1,2850.00,,30
7782,CLARK,MANAGER,7839,1981-6-9,2450.00,,10
7788,SCOTT,ANALYST,7566,1987-4-19,3000.00,,20
7839,KING,PRESIDENT,,1981-11-17,5000.00,,10
7844,TURNER,SALESMAN,7698,1981-9-8,1500.00,0.00,30
7876,ADAMS,CLERK,7788,1987-5-23,1100.00,,20
7900,JAMES,CLERK,7698,1981-12-3,950.00,,30
7902,FORD,ANALYST,7566,1981-12-3,3000.00,,20
7934,MILLER,CLERK,7782,1982-1-23,1300.00,,10
```

然后将数据文件导入到这两个表中，作为示例数据。

```
hive> load data local inpath '/root/dept' into table dept;
hive> load data local inpath '/root/emp' into table emp;
```

接下来可以对这两个表进行基础的增删改查操作，基本跟MySQL差不多。稍微有点小区别：

1、hive语句不区分大小写

2、select语句中支持指定列，也支持给列取别名。支持常用的聚合函数如 count、max、min、sum、avg。支持limit 条件限制返回的行数。支持Group by +having的分组统计语句。支持join进行数据连接。

3、尝试执行update、delete操作时，会报错

```
FAILED: SemanticException [Error 10294]: Attempt to do update or delete
using transaction manager that does not support these operations.
```

这是因为频繁的行级数据更改已经违背了Hive的设计初衷，所以hive默认不支持。如果非要支持，需要调整hive的配置文件hive-site.xml

```
    <name>hive.support.concurrency</name>
    <value>true</value>

    <name>hive.enforce.bucketing</name>
    <value>true</value>

    <name>hive.exec.dynamic.partition.mode</name>
    <value>nonstrict</value>

    <name>hive.txn.manager</name>
    <value>org.apache.hadoop.hive.ql.lockmgr.DbTxnManager</value>

    <name>hive.compactor.initiator.on</name>
    <value>true</value>

    <name>hive.compactor.worker.threads</name>
    <value>1</value>

    <name>hive.in.test</name>
    <value>true</value>
```

然后重启。

2、常用的算数运算符

运算符	描述
A%B	A对B取余
A&B	A和B按位与
A|B	A和B按位或
A^B	A和B按位异或
~A	A按位取反
A<=>B	如果A和B都为NULL返回true，如果一边为NULL返回false
A [NOT] LIKE B	B是一个SQL下的简单正则表达式。'x%'表示A必须以'X'开头。
A RLIKE B, A REGEXP B	B是基于java的正则表达式。匹配使用的是JKD中的正则表达式。官方给了一个测试页面https://www.regexplanet.com/advanced/java/index.html

例如：如何判断一个数字是不是2的指数?

```
select 4&-4==4;  //true
select 5&-5==5;  //false
```

正则查询

```
--查找名字以A开头的员工
select * from emp where ename like 'A%';
--查找名字中第二个字母为A的员工
select * from emp where ename like '_A%';
--查找名字中带A的员工
select * from emp where ename rlike '[A]';
```

3、分组排序 SORT/ORDER/CLUSTER/DISTRIBUTE BY

order by：hive中支持以order by关键字对结果进行排序

```
hive>select * from emp order by deptno desc;

7521    WARD    SALESMAN        7698    1981-2-22       1250.0  500.0   30
7499    ALLEN   SALESMAN        7698    1981-2-20       1600.0  300.0   30
7900    JAMES   CLERK   7698    1981-12-3       950.0   NULL    30
7844    TURNER  SALESMAN        7698    1981-9-8        1500.0  0.0     30
7698    BLAKE   MANAGER 7839    1981-5-1        2850.0  NULL    30
7654    MARTIN  SALESMAN        7698    1981-9-28       1250.0  1400.0  30
7876    ADAMS   CLERK   7788    1987-5-23       1100.0  NULL    20
7902    FORD    ANALYST 7566    1981-12-3       3000.0  NULL    20
7788    SCOTT   ANALYST 7566    1987-4-19       3000.0  NULL    20
7369    SMITH   CLERK   7902    1980-12-17      800.0   NULL    20
7566    JONES   MANAGER 7839    1981-4-2        2975.0  NULL    20
7934    MILLER  CLERK   7782    1982-1-23       1300.0  NULL    10
7839    KING    PRESIDENT       NULL    1981-11-17      5000.0  NULL    10
7782    CLARK   MANAGER 7839    1981-6-9        2450.0  NULL    10
```

这时可以看到hive会对结果做整体排序。但是如果对于大规模的数据集，order by的效率就会非常低。在很多情况下，并不需要全局排序，只是需要分区有序，就可以用sort by。

sort by：sort by会为每个reducer产生一个排序文件。每个Reducer内部进行排序，而对全局结果集来说并不进行排序。

```
hive> set mapreduce.job.reduces=4;
hive> select * from emp sort by deptno desc;
---分四个reducer。每个reducer内部有序，整体无序。
7900    JAMES   CLERK   7698    1981-12-3       950.0   NULL    30
7566    JONES   MANAGER 7839    1981-4-2        2975.0  NULL    20
7902    FORD    ANALYST 7566    1981-12-3       3000.0  NULL    20
```

```
7788    SCOTT   ANALYST 7566    1987-4-19       3000.0  NULL    20
7499    ALLEN   SALESMAN        7698    1981-2-20       1600.0  300.0   30
7698    BLAKE   MANAGER 7839    1981-5-1        2850.0  NULL    30
7934    MILLER  CLERK   7782    1982-1-23       1300.0  NULL    10
7521    WARD    SALESMAN        7698    1981-2-22       1250.0  500.0   30
7844    TURNER  SALESMAN        7698    1981-9-8        1500.0  0.0     30
7654    MARTIN  SALESMAN        7698    1981-9-28       1250.0  1400.0  30
7369    SMITH   CLERK   7902    1980-12-17      800.0   NULL    20
7876    ADAMS   CLERK   7788    1987-5-23       1100.0  NULL    20
7839    KING    PRESIDENT       NULL    1981-11-17      5000.0  NULL    10
7782    CLARK   MANAGER 7839    1981-6-9        2450.0  NULL    10
hive> insert overwrite local directory '/root/hive/data/emp' select * from
emp sort by deptno desc;
--在本地生成四个reduce结果文件。每个文件局部有序。
```

districbute by：在sort by的基础上，可以通过districbute by指定sql语句的分区键，即按哪个字段字段分区。类似于MapReduce中的pattition。

```
hive> select * from emp distribute by deptno sort by empno desc;
--按deptno进行分区
7902    FORD    ANALYST 7566    1981-12-3       3000.0  NULL    20
7876    ADAMS   CLERK   7788    1987-5-23       1100.0  NULL    20
7788    SCOTT   ANALYST 7566    1987-4-19       3000.0  NULL    20
7566    JONES   MANAGER 7839    1981-4-2        2975.0  NULL    20
7369    SMITH   CLERK   7902    1980-12-17      800.0   NULL    20
7934    MILLER  CLERK   7782    1982-1-23       1300.0  NULL    10
7900    JAMES   CLERK   7698    1981-12-3       950.0   NULL    30
7844    TURNER  SALESMAN        7698    1981-9-8        1500.0  0.0     30
7839    KING    PRESIDENT       NULL    1981-11-17      5000.0  NULL    10
7782    CLARK   MANAGER 7839    1981-6-9        2450.0  NULL    10
7698    BLAKE   MANAGER 7839    1981-5-1        2850.0  NULL    30
7654    MARTIN  SALESMAN        7698    1981-9-28       1250.0  1400.0  30
7521    WARD    SALESMAN        7698    1981-2-22       1250.0  500.0   30
7499    ALLEN   SALESMAN        7698    1981-2-20       1600.0  300.0   30
hive> insert overwrite inpath '/root/hive/data/emp2' select * from emp
distribute by deptno sort by empno desc;
--会在对应目录生成四个reduce文件。
```

districbute by的分区规则是按分区阻断的hash码与reduce的个数取模进行的，结果相同的就分到同一个区。

clueter by: 如果distribute by 和 sort by 的字段相同，可以使用cluster by。cluster by 兼具了distribute by和sort by 的功能，但是排序时，只能升序排序，不能指定排序规则。

```
hive> select * from emp cluster by deptno;
等价于
hive> select * from emp distribute by deptno sort by deptno;
```

4、With关键字 Common Table Expression（CTE）

Hive可以在执行SELECT、INSERT、CREATE TABLE AS SELECT 或者CREATE VIEW AS SELECT 指令时，可以使用with关键字创建一个临时的结果。就相当于提前创建了视图，只不过这个视图只在当前语句中有效。

```
hive> with cte_dept_20 as (select deptno,dname from dept where deptno='20')
select * from cte_dept_20; --查看20部门的部门名字
hive> with cte_dept as (select deptno,dname from dept),
    cte_emp as (select deptno,empno,ename from emp)
    select e.empno,e.ename,e.deptno,d.dname from cte_emp e left join
cte_dept d on e.deptno=d.deptno;  --查看每个人所属的部门
hive> create table dept_member as
    with cte_dept as (select deptno,dname from dept),
    cte_emp as (select deptno,empno,ename from emp)
    select e.empno,e.ename,e.deptno,d.dname from cte_emp e left join
cte_dept d on e.deptno=d.deptno; --根据结果创建表。
```

七、Hive-函数

1、系统自带的函数：

参见官方文档： https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF#LanguageManualUDF-CreatingCustomUDFs

```
--查看系统自带的函数
hive> show functions;
--显示自带的函数用法
hive> desc function nvl;
OK
nvl(value,default_value) - Returns default value if value is null else
returns value
Time taken: 0.088 seconds, Fetched: 1 row(s)
--详细显示自带函数的用法
hive> desc function extended nvl;
OK
```

```
nvl(value,default_value) - Returns default value if value is null else
returns value
Example:
  > SELECT nvl(null,'bla') FROM src LIMIT 1;
  bla
Function class:org.apache.hadoop.hive.ql.udf.generic.GenericUDFNvl
Function type:BUILTIN
Time taken: 0.089 seconds, Fetched: 6 row(s)
```

这样可以看到hive所有内置函数的解释以及使用示例。

2、开窗函数 Windowing,OVER,and Analytics

over 开窗函数： 指定每个分析函数工作的数据窗口大小。每个窗口的大小可能会随着每一行数据的不同而不同。

```
hive> select sum(sal) over() from emp; --查看所有薪水。
hive> select deptno ,sum(sal) over(partition by deptno) as depSal from emp;
-- 查看每个部门的总薪水
hive> select empno,sal,avg(comm) over(partition by deptno order by sal rows
between 1 preceding and 1 following) from emp;  --rows统计自己以及与自己的薪水最
接近的两个人，avg统计这三个人的平均奖金
```

rows between 必须跟在order by子句之后，对排序的结果进行限制。在rows between后可以接以下条件：1 PRECEDING： 前一行， 1 FOLLOWING:后一行，CURRENT ROW 当前行。 另外还有UNBOUNDED 表示不限制前面或后面多少行。

另外，针对over()开窗函数，还可以进行LAG()统计和LEAD()统计。其中，LAG(col,n,default_val)表示往前第n行的数据。LEAD(col,n,defalut_val)表示往后第n行数据。

```
hive> select empno ,
    rank() over (partition by job order by sal desc) salrk ,
    row_number() over (partition by job order by sal desc) salrow from emp;
--查询每个人，按工作分组，按薪水排序 之后的排名名次。 以及按工作分组，按薪水排序后每个人所
在的行数。
```

3、自定义函数

参见官方文档： https://cwiki.apache.org/confluence/display/Hive/HivePlugins

Hive中的用户自定义函数分为三类：

- UDF User-Defined-Function 一行入参出一结果，例如nvl函数，pi函数
- UDAF User-Defined-Aggreget Function 多行入参出一个结果，类似于 max、min、avg这些。
- UDTF User-Defined-Table-Generating Function 一行入参出多个结果。例如lateral view explode()

开发自定义函数

开发hive的自定义函数，需要创建一个maven工程，引入依赖：

```
<dependencies>
    <dependency>
        <groupId>org.apache.hive</groupId>
        <artifactId>hive-exec</artifactId>
        <version>3.1.2</version>
    </dependency>
</dependencies>
```

然后创建一个函数类：

```
package com.example.hive.udf;

import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;

public final class Lower extends UDF {
  public Text evaluate(final Text s) {
    if (s == null) { return null; }
    return new Text(s.toString().toLowerCase());
  }
}
```

使用maven打成jar包，my_jar.jar，然后将这个jar包上传到服务器上。

引入自定义函数

接下来启动hive后，需要引入这个jar包。

```
--引入jar包。
hive> add jar /root/my_jar.jar;
--查看引入的jar包。
hive> list jars;
```

接下来就可以使用这个自定义函数了

```
hive> create function my_lower as 'com.example.hive.udf.Lower';
hive> select my_low(ename) from emp;
```

九、Hive-调优

1、了解Hive的执行计划

之前实验阶段已经发现，有些简单的SQL可以不走MapperReduce直接执行，这样效率很高。而很多复杂的SQL都需要走MapperReduce。到执行阶段再去判断SQL如何运行，效率太低了。就需要提前了解SQL的执行计划，作为对SQL优化的依据。其实在MySQL中也可以查看执行计划，Hive中的执行计划也差不多。

基础语法：

```
explain [extended]|[dependency]|[authorizetion] query;
```

其中query就是查询语句。

例如：

```
#不走MapperReduce
hive> explain select * from emp;
OK
STAGE DEPENDENCIES:
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-0
    Fetch Operator
      limit: -1
      Processor Tree:
        TableScan
          alias: emp
```

```
          Statistics: Num rows: 1 Data size: 6570 Basic stats: COMPLETE
Column stats: NONE
          Select Operator
            expressions: empno (type: int), ename (type: string), job
(type: string), mgr (type: int), hiredate (type: string), sal (type:
double), comm (type: double), deptno (type: int)
            outputColumnNames: _col0, _col1, _col2, _col3, _col4, _col5,
_col6, _col7
            Statistics: Num rows: 1 Data size: 6570 Basic stats: COMPLETE
Column stats: NONE
            ListSink

Time taken: 0.26 seconds, Fetched: 17 row(s)
```

```
# 走MapperReduce
hive> explain select deptno,avg(sal) from emp group by deptno;
OK
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 depends on stages: Stage-1

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Map Operator Tree:
          TableScan
            alias: emp
            Statistics: Num rows: 1 Data size: 6570 Basic stats: COMPLETE
Column stats: NONE
            Select Operator
              expressions: sal (type: double), deptno (type: int)
              outputColumnNames: sal, deptno
              Statistics: Num rows: 1 Data size: 6570 Basic stats: COMPLETE
Column stats: NONE
              Group By Operator
                aggregations: sum(sal), count(sal)
                keys: deptno (type: int)
                mode: hash
                outputColumnNames: _col0, _col1, _col2
                Statistics: Num rows: 1 Data size: 6570 Basic stats:
COMPLETE Column stats: NONE
                Reduce Output Operator
                  key expressions: _col0 (type: int)
                  sort order: +
                  Map-reduce partition columns: _col0 (type: int)
                  Statistics: Num rows: 1 Data size: 6570 Basic stats:
COMPLETE Column stats: NONE
```

```
                  value expressions: _col1 (type: double), _col2 (type: bigint)
      Execution mode: vectorized
      Reduce Operator Tree:
        Group By Operator
          aggregations: sum(VALUE._col0), count(VALUE._col1)
          keys: KEY._col0 (type: int)
          mode: mergepartial
          outputColumnNames: _col0, _col1, _col2
          Statistics: Num rows: 1 Data size: 6570 Basic stats: COMPLETE Column stats: NONE
          Select Operator
            expressions: _col0 (type: int), (_col1 / _col2) (type: double)
            outputColumnNames: _col0, _col1
            Statistics: Num rows: 1 Data size: 6570 Basic stats: COMPLETE Column stats: NONE
            File Output Operator
              compressed: false
              Statistics: Num rows: 1 Data size: 6570 Basic stats: COMPLETE Column stats: NONE
              table:
                  input format: org.apache.hadoop.mapred.SequenceFileInputFormat
                  output format: org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat
                  serde: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

  Stage: Stage-0
    Fetch Operator
      limit: -1
      Processor Tree:
        ListSink

Time taken: 0.656 seconds, Fetched: 53 row(s)
```

可以看到， 在 STAGE DEPNEDENCIES部分会列出整体的执行步骤。 STAGE PLANS 部分会列出每个步骤的详细执行计划。比如对于第一个不走MapperReduce的SQL语句，就只有一个Fetch Operator阶段。这个Fetch操作，Hive就可以直接读取对应的HDFS目录下的文件，而不需要经过复杂的MapperReduce。而对第二个需要走MapperReduce的SQL语句，就分为了两个阶段，第一个阶段是一个MapperReduce计算。第二个阶段依然是Fetch操作。在MapperReduce的执行计划中，列出了每个阶段的大致计算过程。例如对这个分组后求平均值的SQL语句，就会统计sum(sal)和count(sal)，然后在reduce阶段求平均值。

另外，关于Hive在哪些情况下走Fetch，哪些情况下走MapperReduce，在hive-default.xml.template中有一个属性

```
  <property>
    <name>hive.fetch.task.conversion</name>
    <value>more</value>
    <description>
      Expects one of [none, minimal, more].
      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.
      0. none : disable hive.fetch.task.conversion
      1. minimal : SELECT STAR, FILTER on partition columns, LIMIT only
      2. more    : SELECT, FILTER, LIMIT only (support TABLESAMPLE and virtual columns)
    </description>
  </property>
```

none就表示所有SQL都走MapperReduce，而默认的more，列出了大部分可以走Fetch的操作。

当然，对于Hive来说，能够仅仅通过一个Fetch操作就解决的，毕竟只是一小部分简单的SQL。大部分的SQL还是需要MapperReduce进行计算的，所以，对Hive来说，大部分的调优工作就是针对MapperReduce计算过程定制更适合的参数。

2、定制Mapper和Reducer数量

接下来对Hive的调优，就都是针对MapperReduce计算过程的调优了。

针对MapperReduce计算，默认情况下， Map阶段会将同一个key的所有数据分发给一个Reduce，这时，当其中一个key的数据量非常大时，就会造成一个Reduce非常繁忙，而其他的Reduce并不会帮忙分担计算任务的情况。这时候整个集群的计算能力实际上被最繁忙的Reduce给拖累了，而集群的整体计算能力又没有充分发挥出来，这就是经常提到的数据倾斜的问题。而对于我们之前第二个groupby的SQL语句，在Hive当中，可以通过启用map端聚合的方式，来减轻数据倾斜问题的影响。涉及到几个重要的配置参数

```
<!--对于Group By操作，是否启用map端聚合。 默认是启用 -->
<property>
  <name>hive.map.aggr</name>
```

```
    <value>true</value>
    <description>Whether to use map-side aggregation in Hive Group By queries</description>
  </property>
  <!-- 对于Group by操作，当有数据倾斜时，进行负载均衡优化。默认是关闭的 -->
   </property>
  <property>
    <name>hive.groupby.skewindata</name>
    <value>false</value>
    <description>Whether there is skew in data to optimize group by queries</description>
  </property>
  <!-- 在Map端进行聚合操作的条目数 -->
  <property>
    <name>hive.groupby.mapaggr.checkinterval</name>
    <value>100000</value>
    <description>Number of rows after which size of the grouping keys/aggregation classes is performed</description>
  </property>
```

而针对MapperReduce计算过程，另一个很重要的调优手段就是手动调整Map和Reduce的个数。

首先在Map阶段，需要尽量调整输入文件的个数，让输入的文件尽量接近Hadoop的文件块大小(默认128M)，而不要是很多小的文件。大量小而碎的文件不仅加大了NameNode的负担，也会造成更多的Map，从而浪费Map在任务启动和初始化过程中的资源。例如通过hadoop fs -getmerge指令，将HDFS上一个目录里的文件整理合并成服务器本地的一个大文件，再上传到HDFS上。另外，在Hive中，也有两个参数配置了MapperReduce计算过程中是否需要进行小文件合并。

```
  <property>
    <name>hive.merge.mapfiles</name>
    <value>true</value>
    <description>Merge small files at the end of a map-only job</description>
  </property>
  <property>
    <name>hive.merge.mapredfiles</name>
    <value>false</value>
    <description>Merge small files at the end of a map-reduce job</description>
  </property>
```

然后在Reduce阶段，同样需要调整Reduce计算的个数，过多的Reduce也同样会消耗更多的启动和初始化的时间，并没每个Reduce会产生一个输出文件，同样也会带来文件碎片过多的问题。而过少的Reduce又会降低整体的计算速度。所以，需要综合定制Reduce的个数。而定制Reduce个数的参数，其实在之前每次执行SQL时，已经有所体现。

```
  <!-- 每个Reduce处理的数据量 -->
  <property>
    <name>hive.exec.reducers.bytes.per.reducer</name>
    <value>256000000</value>
    <description>size per reducer.The default is 256Mb, i.e if the input size is 1G, it will use 4 reducers.</description>
  </property>
  <!-- 定制最大reduce数 -->
  <property>
    <name>hive.exec.reducers.max</name>
    <value>1009</value>
    <description>
      max number of reducers will be used. If the one specified in the configuration parameter mapred.reduce.tasks is
      negative, Hive will use this one as the max number of reducers when automatically determine number of reducers.
    </description>
  </property>
```

通过这两个参数可以定制如何计算reduce的个数。另外，还有一个参数可以强行指定reduce的个数，针对某些特定的SQL还是挺有用的。

```
hive> set mapreduce.job.reduces;
mapreduce.job.reduces=-1
```

3、Hive与其他大数据组件融合

在整个大数据技术生态中，Hive的定位通常是一个数据仓库，即他的强项在于元数据管理以及对Hdfs文件的映射索引。而管理、查询数据的功能其实是比较弱的。通常都会与其他组件融合，才能发挥Hive的强项。这里介绍下Hive与其他大数据组件集成的几种大致思路：

1、Hive on Tez： 这个之前提到过，就是将Hive的计算引擎由Hadoop的mapreduce转为Tez。而Tez是一个构建DAG有序无环图的计算引擎。构建于Hadoop的YARN之上。

2、Hive on Spark： 与Hive on Tez的思想类似，就是将Hive的计算引擎转为Spark。利用Spark内存计算高效的特点，加快MapperReduce计算的速度。但是这种方式配置起来比较麻烦，并且如果作为计算入口，其实Spark比Hive使用得更多。所以在实际项目中，这种方式用得比较少，而用得更多的是Spark on Hive。

3、Spark on Hive： 这种集成方式是让Spark主动来利用Hive的元数据来加快对Hdfs文件的索引。然后利用Spark的Spark SQL 来提供对Hdfs的文件操作。这种方式配置起来相对比较简单，使用场景也更多。

4、Hive & HBase： Hbase同样是一个构建于Hadoop的Hdfs之上的一个大数据引擎。但是，Hbase是基于列式存储的，他管理数据、过滤数据的效率是非常高的，但是其列式存储的特性，造成他的数据检索是非常自由，也不太规范的。而Hive与Hbase集成后，可以用Hive来映射Hbase中的数据，从而给HBase数据增加SQL管理的功能。这样能够更好的集合两个大数据组件各自的长处。这也是实际项目中使用的比较多的一种方式。

下面简单演示一下第3种，Spark on Hive的效果。这种配置方式比较简单，Spark采用的是基于hadoop3.2编译的spark-3.1.1-bin-hadoop3.2.tgz。在这个版本的部署包中已经完成了与Hive的集成。只需要将Hive下的hive-site.xml拷贝到spark的conf目录下，然后就可以使用SparkSQL来访问Hive中的数据了，并且，使用spark后，可以明显的感觉到sql的性能提升。并且，执行计划项目Hive on MR也简化了不少。

> 唯一需要注意下的是hive中的hive.execution.engine这个组件不要配置成Tez。

```
[root@192-168-65-174 bin]# ./spark-shell
2021-06-15 14:56:28,711 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://hadoop01:4040
Spark context available as 'sc' (master = local[*], app id = local-1623740196804).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.1.1
```

```
      /_/

Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_212)
Type in expressions to have them evaluated.
Type :help for more information.

scala> import org.apache.spark.sql.hive.HiveContext;
import org.apache.spark.sql.hive.HiveContext

scala> val hc = new HiveContext(sc);
warning: there was one deprecation warning (since 2.0.0); for details, enable `:setting -deprecation' or `:replay -deprecation'
hc: org.apache.spark.sql.hive.HiveContext = org.apache.spark.sql.hive.HiveContext@5b610c90

scala> hc.sql("show databases").show
2021-06-15 14:57:24,263 WARN conf.HiveConf: HiveConf of name hive.metastore.event.db.notification.api.auth does not exist
+---------+
|namespace|
+---------+
|  db_hive|
|  default|
+---------+

scala> hc.sql("select deptno,avg(sal) from emp group by deptno").show
+------+------------------+
|deptno|          avg(sal)|
+------+------------------+
|    20|            2175.0|
|    10|2916.6666666666665|
|    30|1566.6666666666667|
+------+------------------+

scala> hc.sql("explain select deptno,avg(sal) from emp group by deptno").show(truncate=false)
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
```

```
|plan                                                                                                                                                                                                                                                                                                                                                                                  |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
|== Physical Plan ==
*(2) HashAggregate(keys=[deptno#72], functions=[avg(sal#70)])
+- Exchange hashpartitioning(deptno#72, 200), ENSURE_REQUIREMENTS, [id=#73]
   +- *(1) HashAggregate(keys=[deptno#72], functions=[partial_avg(sal#70)])
      +- Scan hive default.emp [sal#70, deptno#72], HiveTableRelation [`default`.`emp`, org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, Data Cols: [empno#65, ename#66, job#67, mgr#68, hiredate#69, sal#70, comm#71, deptno#72], Partition Cols: []]

|
+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
```

关于大数据组件融合，是在大数据基础环境搭建时非常重要的一个环节，这一部分内容，涉及到很多其他大数据组件的内容，这里就不再一一介绍了。如果有时间的话，在这些常用的大数据组件介绍完成之后，再准备专门的内容来分享如何将这些大数据组件有效融合。这次Hive的专题就到这里了。