



Apache Impala



一、 课程计划

目录

一、 课程计划.....	2
二、 Apache Impala.....	5
1. Impala 基本介绍.....	5
2. Impala 与 Hive 关系	6
3. Impala 与 Hive 异同	7
3.1. Impala 使用的优化技术.....	8
3.2. 执行计划.....	8
3.3. 数据流.....	8
3.4. 内存使用.....	9
3.5. 调度.....	9
3.6. 容错.....	9
3.7. 适用面.....	9
4. Impala 架构.....	10
4.1. Impalad	10
4.2. Impala State Store	10
4.3. CLI.....	11
4.4. Catalogd	11
5. Impala 查询处理过程.....	12
三、 Impala 安装部署.....	13
1. 安装前提.....	13
2. 下载安装包、依赖包.....	13
3. 虚拟机新增磁盘（可选）	14
3.1. 关机新增磁盘.....	14
3.2. 开机挂载磁盘.....	17
4. 配置本地 yum 源.....	19
4.1. 上传安装包解压.....	19
4.2. 配置本地 yum 源信息.....	19
5. 安装 Impala.....	21
5.1. 集群规划.....	21
5.2. 主节点安装.....	21
5.3. 从节点安装.....	21
6. 修改 Hadoop、Hive 配置.....	22
6.1. 修改 hive 配置.....	22
6.2. 修改 hadoop 配置	24



6.3. 重启 hadoop、hive	25
6.4. 复制 hadoop、hive 配置文件.....	25
7. 修改 impala 配置.....	26
7.1. 修改 impala 默认配置.....	26
7.2. 添加 mysql 驱动	26
7.3. 修改 bigtop 配置	26
8. 启动 impala 服务.....	27
8.1. impala web ui.....	27
四、Impala-shell 命令参数.....	28
1. impala-shell 外部命令	28
2. impala-shell 内部命令	29
五、Impala sql 语法	31
1. 数据库特定语句.....	31
1.1. 创建数据库.....	31
1.2. 删除数据库.....	32
2. 表特定语句.....	33
2.1. create table 语句	33
2.2. insert 语句	34
2.3. select 语句	35
2.4. describe 语句.....	35
2.5. alter table.....	36
2.6. delete、truncate table.....	37
2.7. view 视图	38
2.8. order by 子句	39
2.9. group by 子句	40
2.10. having 子句	40
2.11. limit、offset	40
2.12. with 子句	41
2.13. distinct.....	41
六、Impala 数据导入方式.....	42
1. load data	42
2. insert into values.....	43
3. insert into select.....	43
4. create as select.....	43
七、Impala 的 java 开发	44
1. 下载 impala jdbc 依赖.....	44
2. 创建 java 工程	44
3. java api	46



黑马程序员
www.itheima.com

传智播客旗下
高端IT教育品牌

改变中国IT教育，我们正在行动



二、 Apache Impala

1. Impala 基本介绍

impala 是 cloudera 提供的一款高效率的 **sql 查询工具**，提供**实时的查询**效果，官方测试性能比 hive 快 10 到 100 倍，其 sql 查询比 sparkSQL 还要更加快速，号称是当前大数据领域最快的查询 sql 工具，

impala 是参照谷歌的新三篇论文（Caffeine--网络搜索引擎、Pregel--分布式图计算、Dremel--交互式分析工具）当中的 Dremel 实现而来，其中旧三篇论文分别是（BigTable, GFS, MapReduce）分别对应我们即将学的 HBase 和已经学过的 HDFS 以及 MapReduce。

impala 是**基于 hive** 并使用内存进行计算，兼顾数据仓库，具有实时，批处理，多并发等优点。



Apache Impala is the open source, native analytic database
for Apache Hadoop.

Follow us on Twitter at [@ApacheImpala](https://twitter.com/ApacheImpala)!

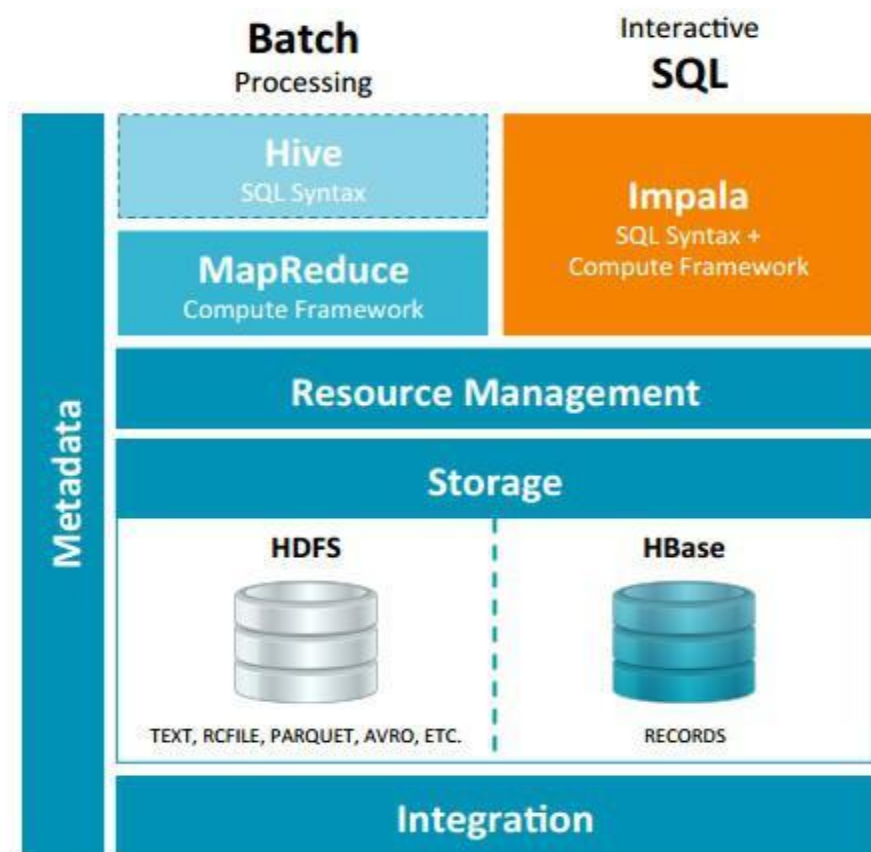
2. Impala 与 Hive 关系

impala 是基于 hive 的大数据分析查询引擎，直接使用 hive 的元数据库 metadata，意味着 impala 元数据都存储在 hive 的 metastore 当中，并且 impala 兼容 hive 的绝大多数 sql 语法。所以需要安装 impala 的话，必须先安装 hive，保证 hive 安装成功，并且还需要启动 hive 的 metastore 服务。

Hive 元数据包含用 Hive 创建的 database、table 等元信息。元数据存储在关系型数据库中，如 Derby、MySQL 等。

客户端连接 metastore 服务，metastore 再去连接 MySQL 数据库来存取元数据。有了 metastore 服务，就可以有多个客户端同时连接，而且这些客户端不需要知道 MySQL 数据库的用户名和密码，只需要连接 metastore 服务即可。

```
nohup hive --service metastore >> ~/metastore.log 2>&1 &
```



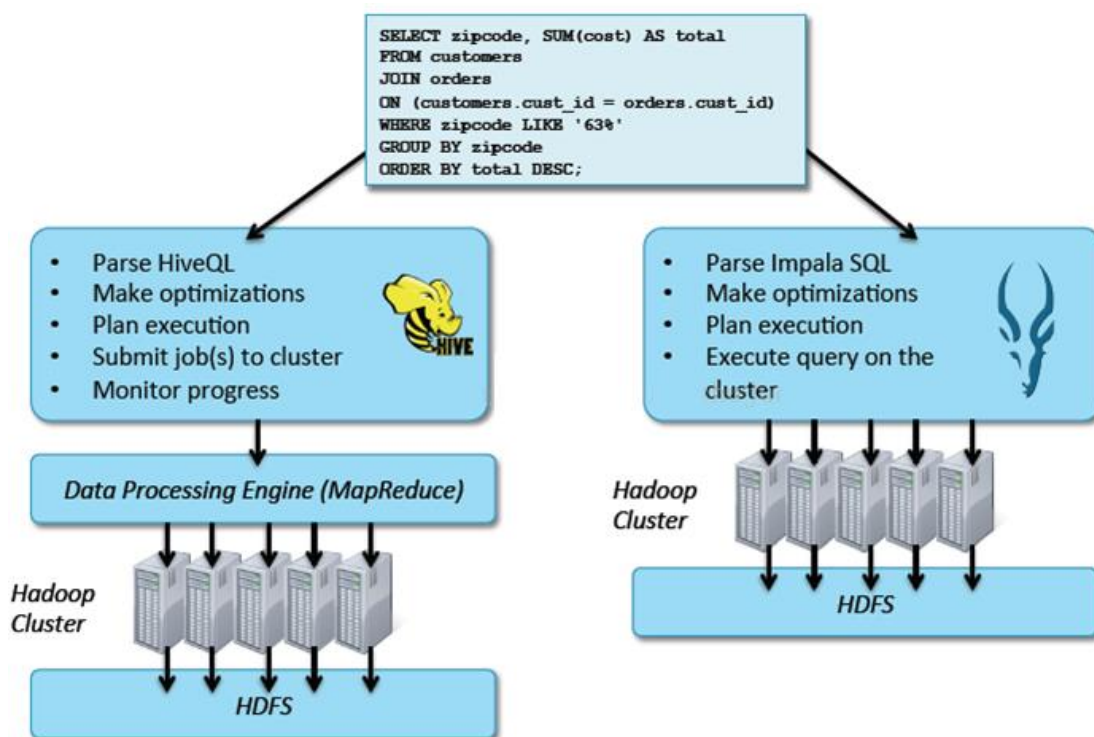
Hive 适合于长时间的批处理查询分析，而 Impala 适合于实时交互式 SQL 查询。可以先使用 hive 进行数据转换处理，之后使用 Impala 在 Hive 处理后的结果数据集上进行快速的数据分析。



3. Impala 与 Hive 异同

Impala 与 Hive 都是构建在 Hadoop 之上的数据查询工具各有不同的侧重适应面,但从客户端使用来看 Impala 与 Hive 有很多的共同之处,如数据表元数据、ODBC/JDBC 驱动、SQL 语法、灵活的文件格式、存储资源池等。

但是 Impala 跟 Hive 最大的优化区别在于：**没有使用 MapReduce 进行并行计算**,虽然 MapReduce 是非常好的并行计算框架,但它更多的面向批处理模式,而不是面向交互式的 SQL 执行。与 MapReduce 相比,Impala 把整个查询分成一个**执行计划树**,而不是一连串的 MapReduce 任务,在分发执行计划后,Impala 使用**拉式获取数据**的方式获取结果,把结果数据组成按执行树流式传递汇集,减少的了把中间结果写入磁盘的步骤,再从磁盘读取数据的开销。Impala 使用服务的方式避免每次执行查询都需要启动的开销,即相比 Hive 没了 MapReduce 启动时间。





3.1. Impala 使用的优化技术

使用 LLVM 产生运行代码，针对特定查询生成特定代码，同时使用 Inline 的方式减少函数调用的开销，加快执行效率。(C++特性)

充分利用可用的硬件指令（SSE4.2）。

更好的 IO 调度，Impala 知道数据块所在的磁盘位置能够更好的利用多磁盘的优势，同时 Impala 支持直接数据块读取和本地代码计算 checksum。

通过选择合适数据存储格式可以得到最好性能（Impala 支持多种存储格式）。

最大使用内存，中间结果不写磁盘，及时通过网络以 stream 的方式传递。

3.2. 执行计划

Hive: 依赖于 MapReduce 执行框架，执行计划分成 map->shuffle->reduce->map->shuffle->reduce... 的模型。如果一个 Query 会被编译成多轮 MapReduce，则会有更多的写中间结果。由于 MapReduce 执行框架本身的特点，过多的中间过程会增加整个 Query 的执行时间。

Impala: 把执行计划表现为一棵完整的执行计划树，可以更自然地分发执行计划到各个 Impalad 执行查询，而不用像 Hive 那样把它组合成管道型的 map->reduce 模式，以此保证 Impala 有更好的并发性和避免不必要的中间 sort 与 shuffle。

3.3. 数据流

Hive: 采用推的方式，每一个计算节点计算完成后将数据主动推给后续节点。

Impala: 采用拉的方式，后续节点通过 getNext 主动向前面节点要数据，以此方式数据可以流式的返回给客户端，且只要有 1 条数据被处理完，就可以立即展现出来，而不用等到全部处理完成，更符合 SQL 交互式查询使用。



3.4. 内存使用

Hive: 在执行过程中如果内存放不下所有数据，则会使用外存，以保证 Query 能顺序执行完。每一轮 MapReduce 结束，中间结果也会写入 HDFS 中，同样由于 MapReduce 执行架构的特性，shuffle 过程也会有写本地磁盘的操作。

Impala: 在遇到内存放不下数据时，版本 1.0.1 是直接返回错误，而不会利用外存，以后版本应该会进行改进。这使用得 Impala 目前处理 Query 会受到一定的限制，最好还是与 Hive 配合使用。

3.5. 调度

Hive: 任务调度依赖于 Hadoop 的调度策略。

Impala: 调度由自己完成，目前只有一种调度器 simple-schedule，它会尽量满足数据的局部性，扫描数据的进程尽量靠近数据本身所在的物理机器。调度器目前还比较简单，在 SimpleScheduler::GetBackend 中可以看到，现在还没有考虑负载，网络 IO 状况等因素进行调度。但目前 Impala 已经有对执行过程的性能统计分析，应该以后版本会利用这些统计信息进行调度吧。

3.6. 容错

Hive: 依赖于 Hadoop 的容错能力。

Impala: 在查询过程中，没有容错逻辑，如果在执行过程中发生故障，则直接返回错误（这与 Impala 的设计有关，因为 Impala 定位于实时查询，一次查询失败，再查一次就好了，再查一次的成本很低）。

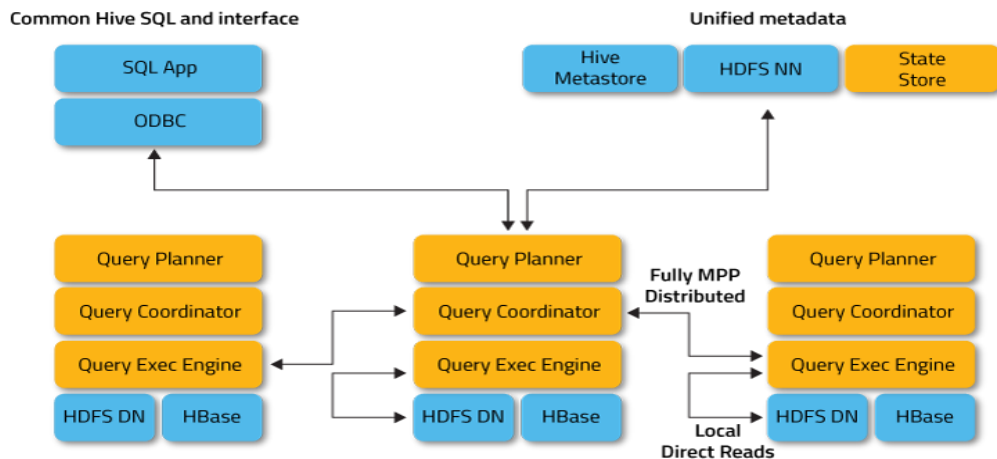
3.7. 适用面

Hive: 复杂的批处理查询任务，数据转换任务。

Impala: 实时数据分析，因为不支持 UDF，能处理的问题域有一定的限制，与 Hive 配合使用，对 Hive 的结果数据集进行实时分析。

4. Impala 架构

Impala 主要由 **Impalad**、**State Store**、**Catalogd** 和 **CLI** 组成。



4.1. Impalad

Impalad: 与 DataNode 运行在同一节点上，由 **Impalad** 进程表示，它接收客户端的查询请求（接收查询请求的 **Impalad** 为 **Coordinator**，**Coordinator** 通过 **JNI** 调用 **java** 前端解释 **SQL** 查询语句，生成查询计划树，再通过调度器把执行计划分发给具有相应数据的其它 **Impalad** 进行执行），读写数据，并行执行查询，并把结果通过网络流式的传送回给 **Coordinator**，由 **Coordinator** 返回给客户端。同时 **Impalad** 也与 **State Store** 保持连接，用于确定哪个 **Impalad** 是健康和可以接受新的工作。

在 **Impalad** 中启动三个 **ThriftServer**: **beeswax_server**(连接客户端)，**hs2_server**（借用 **Hive** 元数据），**be_server**(**Impalad** 内部使用)和一个 **ImpalaServer** 服务。

4.2. Impala State Store

Impala State Store: 跟踪集群中的 **Impalad** 的健康状态及位置信息，由 **statestored** 进程表示，它通过创建多个线程来处理 **Impalad** 的注册订阅和与各 **Impalad** 保持心跳连接，各 **Impalad** 都会缓存一份 **State Store** 中的信息，当 **State Store** 离线后（**Impalad** 发现 **State Store** 处于离线时，会进入 **recovery** 模式，反复注册，当 **State Store** 重新加入集群后，自动恢复正常，更新缓存数据）因为 **Impalad** 有 **State Store** 的缓存仍然可以工作，但会因为有些 **Impalad** 失效了，而已缓存数



据无法更新，导致把执行计划分配给了失效的 Impalad，导致查询失败。

4.3. CLI

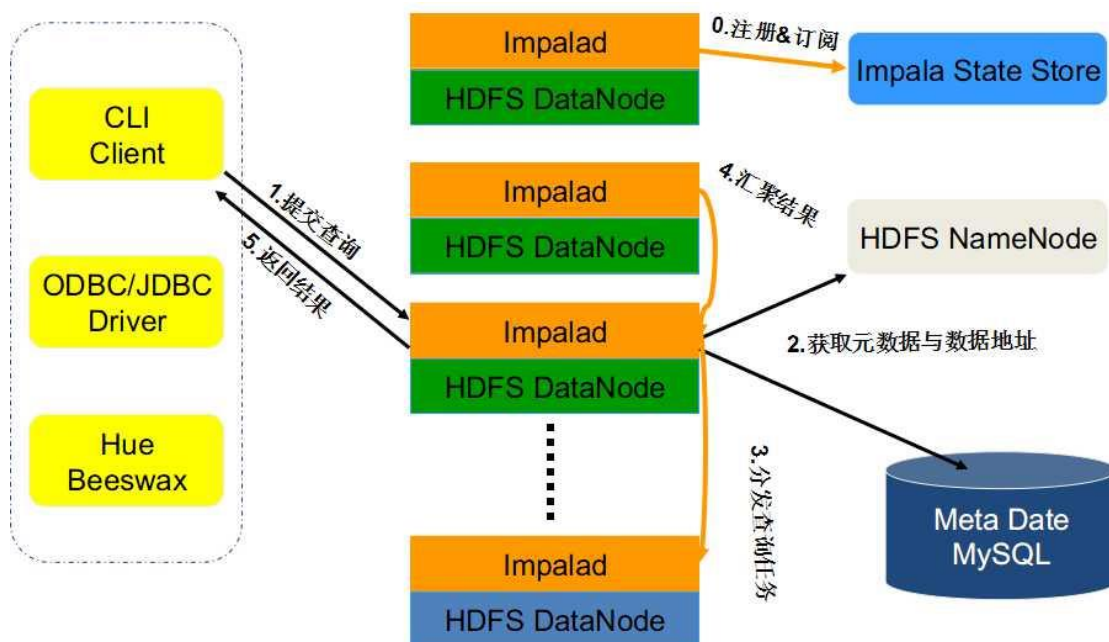
CLI: 提供给用户查询使用的命令行工具（Impala Shell 使用 python 实现），同时 Impala 还提供了 Hue，JDBC， ODBC 使用接口。

4.4. Catalogd

Catalogd: 作为 metadata 访问网关，从 Hive Metastore 等外部 catalog 中获取元数据信息，放到 impala 自己的 catalog 结构中。impalad 执行 ddl 命令时通过 catalogd 由其代为执行，该更新则由 statestored 广播。

5. Impala 查询处理过程

Impalad 分为 Java 前端与 C++处理后端，接受客户端连接的 Impalad 即作为这次查询的 Coordinator，Coordinator 通过 JNI 调用 Java 前端对用户的查询 SQL 进行分析生成执行计划树。



Java 前端产生的执行计划树以 Thrift 数据格式返回给 C++后端(Coordinator)（执行计划分为多个阶段，每一个阶段叫做一个 *PlanFragment*，每一个 *PlanFragment* 在执行时可以由多个 *Impalad* 实例并行执行(有些 *PlanFragment* 只能由一个 *Impalad* 实例执行,如聚合操作)，整个执行计划为一执行计划树)。

Coordinator 根据执行计划，数据存储信息（*Impala* 通过 *libhdfs* 与 *HDFS* 进行交互。通过 *hdfsGetHosts* 方法获得文件数据块所在节点的位置信息），通过调度器（现在只有 *simple-scheduler*，使用 *round-robin* 算法）Coordinator::Exec 对生成的执行计划树分配给相应的后端执行器 *Impalad* 执行（查询会使用 *LLVM* 进行代码生成，编译，执行），通过调用 *GetNext()*方法获取计算结果。

如果是 *insert* 语句，则将计算结果通过 *libhdfs* 写回 *HDFS* 当所有输入数据被消耗光，执行结束，之后注销此次查询服务。



三、 Impala 安装部署

1. 安装前提

集群提前安装好 hadoop, hive。

hive 安装包 scp 在所有需要安装 impala 的节点上, 因为 impala 需要引用 hive 的依赖包。

hadoop 框架需要支持 C 程序访问接口, 查看下图, 如果有该路径下有这么文件, 就证明支持 C 接口。

```
[root@node-1 native]# pwd
/export/servers/hadoop-2.7.5/lib/native
[root@node-1 native]# ll
total 4372
-rw-r--r--. 1 root root 1123342 May  8 15:39 libhadoop.a
-rw-r--r--. 1 root root 1487268 May  8 15:39 libhadooppipes.a
lrwxrwxrwx. 1 root root    18 May  8 22:23 libhadoop.so -> libhadoop.so.1.0.0
-rwxr-xr-x. 1 root root  673484 May  8 15:39 libhadoop.so.1.0.0
-rw-r--r--. 1 root root  582040 May  8 15:39 libhadooputils.a
-rw-r--r--. 1 root root  364844 May  8 15:39 libhdfs.a
lrwxrwxrwx. 1 root root    16 May  8 22:23 libhdfs.so -> libhdfs.so.0.0.0
-rwxr-xr-x. 1 root root  229161 May  8 15:39 libhdfs.so.0.0.0
[root@node-1 native]#
```

2. 下载安装包、依赖包

由于 impala 没有提供 tar 包进行安装, 只提供了 rpm 包。因此在安装 impala 的时候, 需要使用 rpm 包来进行安装。rpm 包只有 cloudera 公司提供了, 所以去 cloudera 公司网站进行下载 rpm 包即可。

但是另外一个问题, impala 的 rpm 包依赖非常多的其他的 rpm 包, 可以一个个的将依赖找出来, 也可以将所有的 rpm 包下载下来, 制作成我们本地 yum 源来进行安装。这里就选择制作本地的 yum 源来进行安装。

所以首先需要下载到所有的 rpm 包, 下载地址如下

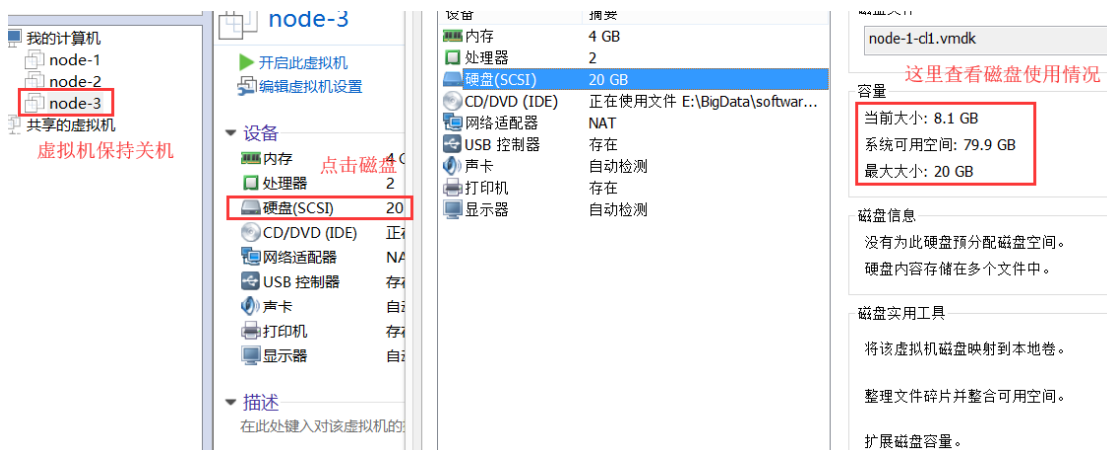
<http://archive.cloudera.com/cdh5/repo-as-tarball/5.14.0/cdh5.14.0-centos6.tar.gz>

3. 虚拟机新增磁盘（可选）

由于下载的 cdh5.14.0-centos6.tar.gz 包非常大，大概 5 个 G，解压之后也最少需要 5 个 G 的空间。而我们的虚拟机磁盘有限，可能会不够用了，所以可以为虚拟机挂载一块新的磁盘，专门用于存储的 cdh5.14.0-centos6.tar.gz 包。

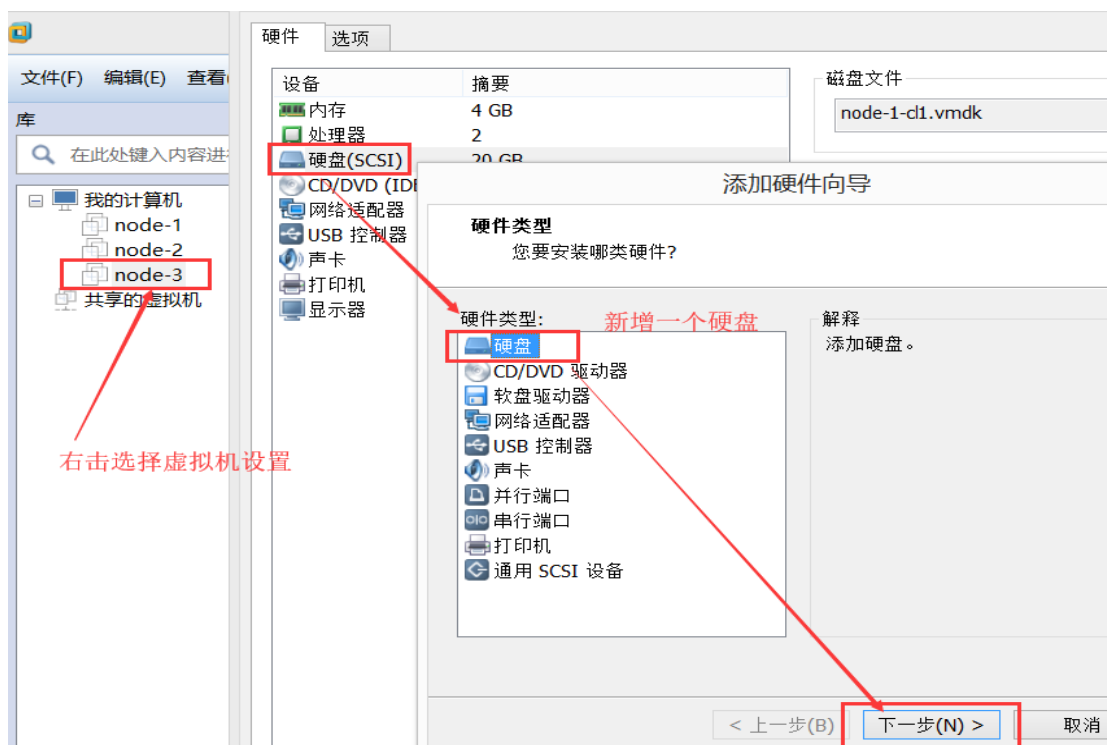
注意事项：新增挂载磁盘需要**虚拟机保持在关机状态**。

如果磁盘空间有余，那么本步骤可以省略不进行。



3.1. 关机新增磁盘

虚拟机关机的状态下，在 VMware 当中新增一块磁盘。







添加硬件向导

指定磁盘容量

磁盘大小为多少?

最大磁盘大小(GB)(S): 20.0

针对 CentOS 64 位 的建议大小: 20 GB

☐ 立即分配所有磁盘空间(A)。

分配所有容量可以提高性能，但要求所有物理磁盘空间立即可用。如果不立即分配所有空间，虚拟磁盘的空间最初很小，会随着您向其中添加数据而不断变大。

☐ 将虚拟磁盘存储为单个文件(O)

☒ 将虚拟磁盘拆分成多个文件(M)

拆分磁盘后，可以更轻松地在计算机之间移动虚拟机，但可能会降低大容量磁盘的性能。

< 上一步(B)

下一步(N) >

取消

添加硬件向导

指定磁盘文件

您要在何处存储磁盘文件?

磁盘文件(F)

将为每 2 GB 容量的虚拟磁盘创建一个磁盘文件。除第一个文件之外，每个文件的文件名均将根据此处提供的文件名自动生成。

文件名: node-3-0.vmdk

浏览(R)...

< 上一步(B)

完成

取消

根据自己笔记本情况确定
新增的磁盘大小



3.2. 开机挂载磁盘

开启虚拟机，对新增的磁盘进行分区，格式化，并且挂载新磁盘到指定目录。

```
[root@node-3 ~]# df -lh 查看磁盘使用情况 只有/ 目录下有个18G磁盘
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/vg_node1-lv_root 18G  7.6G  8.7G  47% /
tmpfs            1.9G   0  1.9G   0% /dev/shm
/dev/sda1        477M  41M  411M   9% /boot
```

```
[root@node-3 ~]# fdisk -l 查看磁盘挂载情况

Disk /dev/sda: 21.5 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000688f9

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1 *          1           64       51200    83  Linux
Partition 1 does not end on cylinder boundary.
/dev/sda2            64        2611     20458496    8e  Linux LVM

Disk /dev/sdb: 21.5 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000
```

原先的磁盘/dev/sda 挂载到/dev/sda1下

我们新增的磁盘还没有指定挂载到哪个目录

```
[root@node-3 ~]# fdisk /dev/sdb 开始格式化磁盘，创建分区，写入分区
Device contains neither a valid DOS partition table, nor Sun, SGI or
Building a new DOS disklabel with disk identifier 0xd66100d1.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by fdisk.

WARNING: DOS-compatible mode is deprecated. It's strongly recommended to
switch off the mode (command 'c') and change display units to
sectors (command 'u').

Command (m for help): n 新建分区
Command action
   e   extended
   p   primary partition (1-4)
p 输入p表示分区为主分区
Partition number (1-4): 1 指定分区个数为1，表示只使用1个分区
First cylinder (1-2610, default 1): 分区起始截止磁盘数，回车默认
Using default value 1
Last cylinder, +cylinders or +size{K,M,G} (1-2610, default 2610):
Using default value 2610

Command (m for help): w w表示开始写入分区
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```



```
Disk /dev/sdb: 21.5 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xd66100d1
```

此时再次查看磁盘情况，发现新增的磁盘也有了一个分区

Device	Boot	Start	End	Blocks	Id	System
/dev/sdb1		1	2610	20964793+	83	Linux

下面对分区进行格式化操作：

```
mkfs -t ext4 -c /dev/sdb1
```

```
[root@node-3 ~]# mkfs -t ext4 -c /dev/sdb1
mke2fs 1.41.12 (17-May-2010)
Filesystem label=
os type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
1310720 inodes, 5241198 blocks
262059 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=4294967296
160 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632,
4096000
checking for bad blocks (read-only test): done
writing inode tables: done
Creating journal (32768 blocks): done
writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 35 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
```

分区进行格式化操作

创建挂载目录：`mount -t ext4 /dev/sdb1 /cloudera_data/`

```
[root@node-3 ~]# mkdir /cloudera_data
[root@node-3 ~]# mount -t ext4 /dev/sdb1 /cloudera_data/
[root@node-3 ~]# df -lh
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/mapper/vg_node1-lv_root	18G	7.6G	8.7G	47%	/
tmpfs	1.9G	0	1.9G	0%	/dev/shm
/dev/sda1	477M	41M	411M	9%	/boot
/dev/sdb1	20G	44M	19G	1%	/cloudera_data

创建挂载目录

查看磁盘大小

执行挂载操作

已经成功挂载

添加至开机自动挂载：

```
vim /etc/fstab
```

```
/dev/sdb1 /cloudera_data ext4 defaults 0 0
```

```
[root@node-3 html]# vim /etc/fstab

#
# /etc/fstab
# Created by anaconda on Mon Apr 8 19:26:07 2019
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
/dev/mapper/vg_node1-lv_root / ext4 defaults 1 1
UUID=cfb5b5c8-3487-4d7e-ad64-22826b782f9a /boot ext4 defaults 1 2
/dev/mapper/vg_node1-lv_swap swap swap defaults 0 0
tmpfs /dev/shm tmpfs defaults 0 0
devpts /dev/pts devpts gid=5,mode=620 0 0
sysfs /sys sysfs defaults 0 0
proc /proc proc defaults 0 0
/dev/sdb1 /cloudera_data ext4 defaults 0 0
```



4. 配置本地 yum 源

4.1. 上传安装包解压

使用 sftp 的方式把安装包大文件上传到服务器/cloudera_data 目录下。

```
sftp> !pwd
e:/
sftp> cd /cloudera_data/
sftp> put cdh5.14.0-centos6.tar.gz
Uploading cdh5.14.0-centos6.tar.gz to /cloudera_data/cdh5.14.0-centos6.tar.gz
100% 5294884kB 57553kB/s 00:01:32
e:/cdh5.14.0-centos6.tar.gz: 5421961451 bytes transferred in 92 seconds (57553 kB/s)
```

```
cd /cloudera_data
```

```
tar -zxvf cdh5.14.0-centos6.tar.gz
```

4.2. 配置本地 yum 源信息

安装 Apache Server 服务器

```
yum -y install httpd
```

```
service httpd start
```

```
chkconfig httpd on
```

配置本地 yum 源的文件

```
cd /etc/yum.repos.d
```

```
vim localimp.repo
```

```
[localimp]
name=localimp
baseurl=http://node-3/cdh5.14.0/
gpgcheck=0
enabled=1
```

创建 apache httpd 的读取链接

```
ln -s /cloudera_data/cdh/5.14.0 /var/www/html/cdh5.14.0
```



确保 linux 的 Selinux 关闭

临时关闭:

```
[root@localhost ~]# getenforce
```

Enforcing

```
[root@localhost ~]# setenforce 0
```

```
[root@localhost ~]# getenforce
```

Permissive

永久关闭:

```
[root@localhost ~]# vim /etc/sysconfig/selinux
```

SELINUX=enforcing 改为 SELINUX=disabled

重启服务 reboot

通过浏览器访问本地 yum 源，如果出现下述页面则成功。

<http://192.168.227.153/cdh5.14.0/>

Index of /cdh5.14.0

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
Parent Directory		-	
RPMS/	07-Jan-2018 08:44	-	
SRPMS/	07-Jan-2018 08:44	-	
generated_index.html	07-Jan-2018 08:44	450	
mirrors	22-Jan-2018 15:37	79	
repodata/	07-Jan-2018 08:44	-	

Apache/2.2.15 (CentOS) Server at 192.168.227.153 Port 80

将本地 yum 源配置文件 localimp.repo 发放到所有需要安装 impala 的节点。

```
cd /etc/yum.repos.d/
```

```
scp localimp.repo node-2:$PWD
```

```
scp localimp.repo node-3:$PWD
```

5. 安装 Impala

5.1. 集群规划

服务名称	从节点	从节点	主节点
impala-catalog			Node-3
impala-state-store			Node-3
impala-server(impalad)	Node-1	Node-2	Node-3

5.2. 主节点安装

在规划的**主节点 node-3** 执行以下命令进行安装：

```
yum install -y impala impala-server impala-state-store impala-catalog impala-shell
```

5.3. 从节点安装

在规划的**从节点 node-1、node-2** 执行以下命令进行安装：

```
yum install -y impala-server
```



6. 修改 Hadoop、Hive 配置

需要在 3 台机器**整个集群上进行操作**，都需要修改。**hadoop、hive 是否正常服务并且配置好**，是决定 impala 是否启动成功并使用的前提。

6.1. 修改 hive 配置

可在 node-1 机器上进行配置，然后 scp 给其他 2 台机器。

```
vim /export/servers/hive/conf/hive-site.xml
```

```
<configuration>
  <property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://node-
1:3306/hive?createDatabaseIfNotExist=true</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.jdbc.Driver</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>root</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>hadoop</value>
  </property>
  <property>
    <name>hive.cli.print.current.db</name>
    <value>true</value>
```



```
</property>
<property>
  <name>hive.cli.print.header</name>
  <value>true</value>
</property>
<!-- 绑定运行 hiveServer2 的主机 host,默认 localhost -->
<property>
  <name>hive.server2.thrift.bind.host</name>
  <value>node-1</value>
</property>
<!-- 指定 hive metastore 服务请求的 uri 地址 -->
<property>
  <name>hive.metastore.uris</name>
  <value>thrift://node-1:9083</value>
</property>
<property>
  <name>hive.metastore.client.socket.timeout</name>
  <value>3600</value>
</property>
</configuration>
```

将 hive 安装包 cp 给其他两个机器。

```
cd /export/servers/
scp -r hive/ node-2:$PWD
scp -r hive/ node-3:$PWD
```



6.2. 修改 hadoop 配置

所有节点创建下述文件夹

```
mkdir -p /var/run/hdfs-sockets
```

修改所有节点的 hdfs-site.xml 添加以下配置，修改完之后重启 hdfs 集群生效

```
vim etc/hadoop/hdfs-site.xml
```

```
<property>
    <name>dfs.client.read.shortcircuit</name>
    <value>true</value>
</property>
<property>
    <name>dfs.domain.socket.path</name>
    <value>/var/run/hdfs-sockets/dn</value>
</property>
<property>
    <name>dfs.client.file-block-storage-
locations.timeout.millis</name>
    <value>10000</value>
</property>
<property>
    <name>dfs.datanode.hdfs-blocks-
metadata.enabled</name>
    <value>true</value>
</property>
```

`dfs.client.read.shortcircuit` 打开 DFSClient 本地读取数据的控制，

`dfs.domain.socket.path` 是 Datanode 和 DFSClient 之间沟通的 Socket 的本地路径。



把更新 **hadoop** 的配置文件，**scp** 给其他机器。

```
cd /export/servers/hadoop-2.7.5/etc/hadoop
```

```
scp -r hdfs-site.xml node-2:$PWD
```

```
scp -r hdfs-site.xml node-3:$PWD
```

注意：**root** 用户不需要下面操作，普通用户需要这一步操作。

给这个文件夹赋予权限，如果用的是普通用户 **hadoop**，那就直接赋予普通用户的权限，例如：

```
chown -R hadoop:hadoop /var/run/hdfs-sockets/
```

因为这里直接用的 **root** 用户，所以不需要赋权限了。

6.3. 重启 **hadoop**、**hive**

在 **node-1** 上执行下述命令分别启动 **hive metastore** 服务和 **hadoop**。

```
cd /export/servers/hive
```

```
nohup bin/hive --service metastore &
```

```
nohup bin/hive --service hiveserver2 &
```

```
cd /export/servers/hadoop-2.7.5/
```

```
sbin/stop-dfs.sh | sbin/start-dfs.sh
```

6.4. 复制 **hadoop**、**hive** 配置文件

impala 的配置目录为 **/etc/impala/conf**，这个路径下面需要把 **core-site.xml**，**hdfs-site.xml** 以及 **hive-site.xml**。

所有节点执行以下命令

```
cp -r /export/servers/hadoop-2.7.5/etc/hadoop/core-site.xml  
/etc/impala/conf/core-site.xml
```

```
cp -r /export/servers/hadoop-2.7.5/etc/hadoop/hdfs-site.xml  
/etc/impala/conf/hdfs-site.xml
```

```
cp -r /export/servers/hive/conf/hive-site.xml  
/etc/impala/conf/hive-site.xml
```



7. 修改 impala 配置

7.1. 修改 impala 默认配置

所有节点更改 impala 默认配置文件

```
vim /etc/default/impala
```

```
IMPALA_CATALOG_SERVICE_HOST=node-3
```

```
IMPALA_STATE_STORE_HOST=node-3
```

7.2. 添加 mysql 驱动

通过配置/etc/default/impala 中可以发现已经指定了 mysql 驱动的位置名字。

```
[root@node-3 ~]# vim /etc/default/impala
IMPALA_CATALOG_SERVICE_HOST=node-3
IMPALA_STATE_STORE_HOST=node-3
IMPALA_STATE_STORE_PORT=24000
IMPALA_BACKEND_PORT=22000
IMPALA_LOG_DIR=/var/log/impala

IMPALA_CATALOG_ARGS=" -log_dir=${IMPALA_LOG_DIR} "
IMPALA_STATE_STORE_ARGS=" -log_dir=${IMPALA_LOG_DIR} -state_store_port=${IMPALA_ST
IMPALA_SERVER_ARGS=" \
    -log_dir=${IMPALA_LOG_DIR} \
    -catalog_service_host=${IMPALA_CATALOG_SERVICE_HOST} \
    -state_store_port=${IMPALA_STATE_STORE_PORT} \
    -use_statestore \
    -state_store_host=${IMPALA_STATE_STORE_HOST} \
    -be_port=${IMPALA_BACKEND_PORT}"

ENABLE_CORE_DUMPS=false
# 注释别忘了
# LTBDHDFS_OPTS=-Djava.library.path=/usr/lib/impala/lib
# MYSQL_CONNECTOR_JAR=/usr/share/java/mysql-connector-java.jar
# IMPALA_BIN=/usr/lib/impala/sbin
# IMPALA_HOME=/usr/lib/impala
# HIVE_HOME=/usr/lib/hive
"/etc/default/impala" 29L, 980c written
[root@node-3 ~]#
```

使用软链接指向该路径即可（3 台机器都需要执行）

```
ln -s /export/servers/hive/lib/mysql-connector-java-
5.1.32.jar /usr/share/java/mysql-connector-java.jar
```

7.3. 修改 bigtop 配置

修改 bigtop 的 java_home 路径（3 台机器）

```
vim /etc/default/bigtop-utils
```

```
export JAVA_HOME=/export/servers/jdk1.8.0_65
```



8. 启动、关闭 impala 服务

主节点 node-3 启动以下三个服务进程

```
service impala-state-store start
```

```
service impala-catalog start
```

```
service impala-server start
```

从节点启动 node-1 与 node-2 启动 impala-server

```
service impala-server start
```

查看 impala 进程是否存在

```
ps -ef | grep impala
```

```
[root@node-3 ~]# service impala-state-store start
Started Impala State Store Server (statestored): [ OK ]
[root@node-3 ~]# service impala-catalog start
Started Impala Catalog Server (catalogd): [ OK ]
[root@node-3 ~]# service impala-server start
Started Impala Server (impalad): [ OK ]
[root@node-3 ~]# ps -ef | grep impala
impala 3661 1 0 20:04 ? 00:00:00 /usr/lib/impala/sbin/statestored -log_dir=/var/log/impala -state_store_port=24000
impala 3731 1 26 20:04 ? 00:00:04 /usr/lib/impala/sbin/catalogd -log_dir=/var/log/impala
impala 3823 1 41 20:04 ? 00:00:04 /usr/lib/impala/sbin/impalad -log_dir=/var/log/impala -catalogd_log_dir=/var/log/impala
root 3967 2166 0 20:04 pts/0 00:00:00 grep impala
[root@node-3 ~]#
```

启动之后所有关于 impala 的日志默认都在 **/var/log/impala**

如果需要关闭 impala 服务 把命令中的 start 该成 stop 即可。注意如果关闭之后进程依然驻留，可以采取下述方式删除。正常情况下是随着关闭消失的。

解决方式：

```
[root@node-2 ~]# jps
2915 SecondaryNameNode
3140 Jps
2804 DataNode
2650 -- process information unavailable
3004 NodeManager
[root@node-2 ~]# rm -rf /tmp/hsperfdata_impala/2650
```

8.1. impala web ui

访问 impalad 的管理界面 <http://node-3:25000/>

访问 statestored 的管理界面 <http://node-3:25010/>



四、 Impala-shell 命令参数

1. impala-shell 外部命令

所谓的外部命令指的是不需要进入到 impala-shell 交互命令行当中即可执行的命令参数。impala-shell 后面执行的时候可以带很多参数。你可以在启动 impala-shell 时设置，用于修改命令执行环境。

`impala-shell -h` 可以帮助我们查看帮助手册。也可以参考课程附件资料。

比如几个常见的：

`impala-shell -r` 刷新 impala 元数据，与建立连接后执行 REFRESH 语句效果相同

`impala-shell -f 文件路径` 执行指定的 sql 查询文件。

`impala-shell -i` 指定连接运行 impalad 守护进程的主机。默认端口是 21000。你可以连接到集群中运行 impalad 的任意主机。

`impala-shell -o` 保存执行结果到文件当中去。

```
[root@node-3 impaladata]# pwd
/root/impaladata
[root@node-3 impaladata]# impala-shell -f impala_test.sql
Starting Impala Shell without Kerberos authentication
Connected to node-3:21000
Server version: impalad version 2.11.0-cdh5.14.0 RELEASE (build d682
Query: select * from test
Query submitted at: 2019-06-03 20:42:19 (Coordinator: http://node-3:
Query progress can be monitored at: http://node-3:25000/query_plan?c
+----+
| a   |
+----+
| 10  |
| 10  |
| 30  |
| 20  |
| 30  |
| 20  |
+----+
```



2. impala-shell 内部命令

所谓内部命令是指，进入 impala-shell 命令行之后可以执行的语法。

```
[node-3:21000] > help;

Documented commands (type help <topic>):
=====
compute  describe  explain  profile  rerun  set  show  unset  values  with
connect  exit      history  quit     select shell tip  use   version

Undocumented commands:
=====
alter  delete  drop  insert  source  summary  upsert
create desc  help  load  src     update
```

connect hostname 连接到指定的机器 impalad 上去执行。

```
[node-3:21000] > connect node-2;
Connected to node-2:21000
Server version: impalad version 2.11.0-cdh5.14.0 RELEASE (build d68206561bc
[node-2:21000] > select * from test;
Query: select * from test
Query submitted at: 2019-06-03 20:45:19 (Coordinator: http://node-2:25000)
Query progress can be monitored at: http://node-2:25000/query_plan?query_id=
```

refresh dbname.tablename 增量刷新，刷新某一张表的元数据，主要用于刷新 hive 当中数据表里面的数据改变的情况。

```
[node-3:21000] > refresh default.test;
Query: refresh default.test
Query submitted at: 2019-06-03 20:47:44 (Coordinator: http://node-3:25000)
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=
Fetched 0 row(s) in 0.09s
[node-3:21000] >
```

invalidate metadata 全量刷新，性能消耗较大，主要用于 hive 当中新建数据库或者数据库表的时候来进行刷新。

quit/exit 命令 从 Impala shell 中弹出

explain 命令 用于查看 sql 语句的执行计划。

```
[node-3:21000] > explain select * from test;
Query: explain select * from test
+-----+
| Explain String |
+-----+
| Max Per-Host Resource Reservation: Memory=0B
| Per-Host Resource Estimates: Memory=32.00MB
| WARNING: The following tables are missing relevant table and/or column statistics.
| default.test
|
| PLAN-ROOT SINK
|
| 01:EXCHANGE [UNPARTITIONED]
|
| 00:SCAN HDFS [default.test]
|   partitions=1/1 files=6 size=18B
+-----+
```



explain 的值可以设置成 0,1,2,3 等几个值，其中 3 级别是最高的，可以打印出最全的信息

```
set explain_level=3;
```

profile 命令执行 sql 语句之后执行，可以

打印出更加详细的执行步骤，主要用于查询结果的查看，集群的调优等。

```
[node-3:21000] > profile;
Query Runtime Profile:
Query (id=7342841544ee0494:bed4e39c00000000):
Summary:
  Session ID: c4b22c0efd5b4e4:875c73b57b3797b9
  Session Type: BEESWAX
  Start Time: 2019-06-03 20:52:33.024657000
  End Time: 2019-06-03 20:52:33.159011000
  Query Type: QUERY
  Query State: FINISHED
  Query Status: OK
  Impala Version: impalad version 2.11.0-cdh5.14.0 RELEASE
  User: root
  Connected User: root
```

注意:如果在 hive 窗口中插入数据或者新建的数据库或者数据库表，那么在 impala 当中是不可直接查询，需要执行 invalidate metadata 以通知元数据的更新；

在 impala-shell 当中插入的数据，在 impala 当中是可以直接查询到的，不需要刷新数据库，其中使用的就是 catalog 这个服务的功能实现的，catalog 是 impala1.2 版本之后增加的模块功能，主要作用就是同步 impala 之间的元数据。

更新操作通知 Catalog，Catalog 通过广播的方式通知其它的 Impalad 进程。默认情况下 Catalog 是异步加载元数据的，因此查询可能需要等待元数据加载完成之后才能进行（第一次加载）。



五、 Impala sql 语法

1. 数据库特定语句

1.1. 创建数据库

CREATE DATABASE 语句用于在 Impala 中创建新数据库。

CREATE DATABASE IF NOT EXISTS database_name;

这里，IF NOT EXISTS 是一个可选的子句。如果我们使用此子句，则只有在没有具有相同名称的现有数据库时，才会创建具有给定名称的数据库。

```
[node-3:21000] > create database tttest;
Query: create database tttest
ERROR: ImpalaRuntimeException: Error making 'createDatabase' RPC to Hive Metastore:
CAUSED BY: MetaException: Got exception: org.apache.hadoop.security.AccessControlException Per
mission denied: user=impala, access=WRITE, inode="/user/hive/warehouse":root:supergroup:drwxr-
xr-x
    at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.check(FSPermissionChecke
r.java:307)
    at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.checkPermission(FSPermis
```

impala 默认使用 impala 用户执行操作，会报权限不足问题，解决办法：

一：给 HDFS 指定文件夹授予权限

```
hadoop fs -chmod -R 777 hdfs://node-1:9000/user/hive
```

二：hadoop 配置文件中 **hdfs-site.xml** 中设置权限为 false

```
<property>
  <name>dfs.permissions.enabled</name>
  <value>>false</value>
</property>
```

上述两种方式都可以。



```
[node-3:21000] > create database ittest;
Query: create database ittest
Fetched 0 row(s) in 0.17s
[node-3:21000] > show databases;
Query: show databases
+-----+-----+
| name          | comment                                     |
+-----+-----+
| _impala_builtins | System database for Impala builtin functions |
| default        | Default Hive database                       |
| itheima         |                                              |
| ittest         |                                              |
| test           |                                              |
+-----+-----+
Fetched 5 row(s) in 0.01s
```

默认就会在 hive 的数仓路径下创建新的数据库名文件夹

`/user/hive/warehouse/ittest.db`

也可以在创建数据库的时候指定 hdfs 路径。需要注意该路径的权限。

`hadoop fs -mkdir -p /input/impala`

`hadoop fs -chmod -R 777 /input/impala`

```
create external table t3(id int ,name string ,age int )
row format delimited fields terminated by '\t' location
'/input/impala/external';
```

```
[node-3:21000] > create external table t3(id int ,name string ,age int ) row format delimited fields terminated by '\t' location '/input/impala/external';
Query: create external table t3(id int ,name string ,age int ) row format delimited fields terminated by '\t' location '/input/impala/external';
Fetched 0 row(s) in 0.08s
```

1.2. 删除数据库

Impala 的 **DROP DATABASE** 语句用于从 Impala 中删除数据库。在删除数据库之前，建议从中删除所有表。

如果使用级联删除，Impala 会在删除指定数据库中的表之前删除它。

`DROP database sample cascade;`

```
[node-3:21000] > drop database ittest;
Query: drop database ittest
ERROR: ImpalaRuntimeException: Error making 'dropDatabase' RPC to Hive Metastore:
CAUSED BY: InvalidOperationException: Database ittest is not empty. One or more tables exist.
[node-3:21000] >

[node-3:21000] > drop database ittest cascade;
Query: drop database ittest cascade
[node-3:21000] >
```


2. 表特定语句

2.1. create table 语句

CREATE TABLE 语句用于在 Impala 中的所需数据库中创建新表。需要指定表名字并定义其列和每列的数据类型。

impala 支持的数据类型和 hive 类似，除了 sql 类型外，还支持 java 类型。

```
create table IF NOT EXISTS database_name.table_name (  
    column1 data_type,  
    column2 data_type,  
    column3 data_type,  
    .....  
    columnN data_type  
);
```

```
CREATE TABLE IF NOT EXISTS my_db.student(name STRING, age  
INT, contact INT );
```

```
[node-3:21000] > CREATE TABLE IF NOT EXISTS my_db.student(name STRING, age INT, contact INT );  
Query: create TABLE IF NOT EXISTS my_db.student(name STRING, age INT, contact INT )  
Fetched 0 row(s) in 0.06s
```

默认建表的数据存储路径跟 hive 一致。也可以在建表的时候通过 location 指定具体路径，需要注意 hdfs 权限问题。

Browse Directory

/user/hive/warehouse/my_db.db/student

Permission	Owner	Group
------------	-------	-------

2.2. insert 语句

Impala 的 INSERT 语句有两个子句: **into** 和 **overwrite**。into 用于插入新记录数据，overwrite 用于覆盖已有的记录。

```
insert into table_name (column1, column2, column3,...columnN)
values (value1, value2, value3,...valueN);
Insert into table_name values (value1, value2, value2);
```

这里，column1, column2, ... columnN 是要插入数据的表中的列的名称。还可以添加值而不指定列名，但是，需要确保值的顺序与表中的列的顺序相同。

举个例子：

```
create table employee (Id INT, name STRING, age INT,address STRING, salary
BIGINT);
```

```
insert into employee VALUES (1, 'Ramesh', 32, 'Ahmedabad', 20000 );
```

```
insert into employee values (2, 'Khilan', 25, 'Delhi', 15000 );
```

```
Insert into employee values (3, 'kaushik', 23, 'Kota', 30000 );
```

```
Insert into employee values (4, 'Chaitali', 25, 'Mumbai', 35000 );
```

```
Insert into employee values (5, 'Hardik', 27, 'Bhopal', 40000 );
```

```
Insert into employee values (6, 'Komal', 22, 'MP', 32000 );
```

Query progress can be monitored at: <http://node>

id	name	age	address	salary
6	Komal	22	MP	32000
1	Ramesh	32	Ahmedabad	20000
3	kaushik	23	Kota	30000
5	Hardik	27	Bhopal	40000
4	Chaitali	25	Mumbai	35000
2	Khilan	25	Delhi	15000

overwrite 覆盖子句覆盖表当中**全部记录**。覆盖的记录将从表中永久删除。

```
Insert overwrite employee values (1, 'Ram', 26, 'Vishakhapatnam', 37000 );
```

id	name	age	address	salary
1	Ram	26	Vishakhapatnam	37000



2.5. alter table

Impala 中的 **Alter table** 语句用于对给定表执行更改。使用此语句，我们可以添加，删除或修改现有表中的列，也可以重命名它们。

表重命名：

```
ALTER TABLE [old_db_name.]old_table_name RENAME TO  
[new_db_name.]new_table_name
```

向表中添加列：

```
ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])
```

从表中删除列：

```
ALTER TABLE name DROP [COLUMN] column_name
```

更改列的名称和类型：

```
ALTER TABLE name CHANGE column_name new_name new_type
```

```
[node-3:21000] > create table t_1(id int,name string);  
Query: create table t_1(id int,name string)  
Fetched 0 row(s) in 0.16s  
[node-3:21000] > alter table t_1 rename to t_2;  
Query: alter table t_1 rename to t_2  
Fetched 0 row(s) in 5.21s  
[node-3:21000] > show tables;  
Query: show tables  
+-----+  
| name |  
+-----+  
| employee |  
| student |  
| t_2 |  
+-----+  
Fetched 3 row(s) in 0.10s  
[node-3:21000] > alter table t_2 add columns(age int);  
Query: alter table t_2 add columns(age int)  
Fetched 0 row(s) in 5.11s  
[node-3:21000] > alter table t_2 drop age;  
Query: alter table t_2 drop age  
Fetched 0 row(s) in 0.19s  
[node-3:21000] > alter table t_2 change name mingzi string;  
Query: alter table t_2 change name mingzi string  
Fetched 0 row(s) in 0.18s  
[node-3:21000] > █
```



2.6. delete、truncate table

Impala **drop** table 语句用于删除 Impala 中的现有表。此语句还会删除内部表的底层 HDFS 文件。

注意：使用此命令时必须小心，因为删除表后，表中可用的所有信息也将永远丢失。

```
DROP table database_name.table_name;
```

```
[node-3:21000] > drop table student;  
Query: drop table student  
[node-3:21000]
```

Impala 的 **Truncate** Table 语句用于从现有表中删除所有记录。保留表结构。

您也可以使用 DROP TABLE 命令删除一个完整的表，但它会从数据库中删除完整的表结构，如果您希望存储一些数据，您将需要重新创建此表。

```
truncate table_name;
```

```
[node-3:21000] > select * from employee;  
Query: select * from employee  
Query submitted at: 2019-06-09 20:22:53 (Coordinator: http://node-3:25000)  
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=ca443  
+-----+-----+-----+-----+-----+  
| id | name | age | address | salary |  
+-----+-----+-----+-----+-----+  
| 1 | Ram | 26 | Vishakhapatnam | 37000 |  
+-----+-----+-----+-----+-----+  
Fetched 1 row(s) in 0.23s  
[node-3:21000] > truncate table employee;  
Query: truncate table employee  
Query submitted at: 2019-06-09 20:23:08 (Coordinator: http://node-3:25000)  
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=c3460  
Fetched 0 row(s) in 0.06s  
[node-3:21000] > select * from employee;  
Query: select * from employee  
Query submitted at: 2019-06-09 20:23:10 (Coordinator: http://node-3:25000)  
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=83461  
Fetched 0 row(s) in 0.13s  
[node-3:21000]
```



2.7. view 视图

视图仅仅是存储在数据库中具有关联名称的 Impala 查询语言的语句。它是
以预定义的 SQL 查询形式的表的组合。

视图可以包含表的所有行或选定的行。

Create View IF NOT EXISTS view_name as Select statement

```
[node-3:21000] > select * from employee;
Query: select * from employee
Query submitted at: 2019-06-09 20:27:17 (Coordinator: http://node-3:25000)
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=3a4
+-----+-----+-----+-----+-----+
| id | name | age | address | salary |
+-----+-----+-----+-----+-----+
| 2 | Khilan | 25 | Delhi | 15000 |
| 1 | Ramesh | 32 | Ahmedabad | 20000 |
| 4 | Chaitali | 25 | Mumbai | 35000 |
+-----+-----+-----+-----+-----+
```

创建视图 view、查询视图 view

CREATE VIEW IF NOT EXISTS employee_view AS select name, age from employee;

```
[node-3:21000] > CREATE VIEW IF NOT EXISTS employee_view AS select name, age from employee;
Query: create VIEW IF NOT EXISTS employee_view AS select name, age from employee
Query submitted at: 2019-06-09 20:29:36 (Coordinator: http://node-3:25000)
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=dd4b51393eb8712d:615392ca00000000
Fetched 0 row(s) in 0.03s
[node-3:21000] > select * from employee_view;
Query: select * from employee_view
Query submitted at: 2019-06-09 20:29:48 (Coordinator: http://node-3:25000)
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=1c4da8e10e75ef5f:2ee5156700000000
+-----+-----+
| name | age |
+-----+-----+
| Khilan | 25 |
| Ramesh | 32 |
| Chaitali | 25 |
+-----+-----+
```

修改视图

ALTER VIEW database_name.view_name 为 Select 语句

删除视图

DROP VIEW database_name.view_name;

```
[node-3:21000] > drop view employee_view;
Query: drop view employee_view
[node-3:21000] >
[node-3:21000] >
```

2.8. order by 子句

Impala **ORDER BY** 子句用于根据一个或多个列以升序或降序对数据进行排序。

默认情况下，一些数据库按升序对查询结果进行排序。

```
select * from table_name ORDER BY col_name
```

```
[ASC|DESC] [NULLS FIRST|NULLS LAST]
```

可以使用关键字 **ASC** 或 **DESC** 分别按升序或降序排列表中的数据。

如果我们使用 **NULLS FIRST**，表中的所有空值都排列在顶行；如果我们使用 **NULLS LAST**，包含空值的行将最后排列。

```
Fetchd 3 row(s) in 0.14s
[node-3:21000] > select * from employee order by salary;
Query: select * from employee order by salary
Query submitted at: 2019-06-09 20:36:22 (Coordinator: http://node-3:25000)
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=a74c81895e7

+----+-----+-----+-----+-----+
| id | name  | age  | address | salary |
+----+-----+-----+-----+-----+
| 2  | Khilan | 25   | Delhi   | 15000  |
| 1  | Ramesh | 32   | Ahmedabad | 20000  |
| 4  | Chaitali | 25   | Mumbai  | 35000  |
+----+-----+-----+-----+-----+

Fetchd 3 row(s) in 0.27s
[node-3:21000] > select * from employee order by salary desc;
Query: select * from employee order by salary desc
Query submitted at: 2019-06-09 20:36:29 (Coordinator: http://node-3:25000)
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=de458127225

+----+-----+-----+-----+-----+
| id | name  | age  | address | salary |
+----+-----+-----+-----+-----+
| 4  | Chaitali | 25   | Mumbai  | 35000  |
| 1  | Ramesh | 32   | Ahmedabad | 20000  |
| 2  | Khilan  | 25   | Delhi   | 15000  |
+----+-----+-----+-----+-----+
```

2.9. group by 子句

Impala **GROUP BY** 子句与 SELECT 语句协作使用，以将相同的数据排列到组中。

```
select data from table_name Group BY col_name;
```

2.10. having 子句

Impala 中的 **Having** 子句允许您指定过滤哪些组结果显示在最终结果中的条件。

一般来说，Having 子句与 group by 子句一起使用；它将条件放置在由 GROUP BY 子句创建的组上。

2.11. limit、offset

Impala 中的 **limit** 子句用于将结果集的行数限制为所需的数，即查询的结果集不包含超过指定限制的记录。

一般来说，select 查询的 resultset 中的行从 0 开始。使用 **offset 子句**，我们可以决定从哪里考虑输出。

```
[node-3:21000] > select * from employee limit 2 offset 1;
Query: select * from employee limit 2 offset 1
Query submitted at: 2019-06-09 20:41:04 (Coordinator: http://node-3:25000)
ERROR: AnalysisException: OFFSET requires an ORDER BY clause: LIMIT 2 OFFSET 1

[node-3:21000] > select * from employee order by salary limit 2 offset 1;
Query: select * from employee order by salary limit 2 offset 1
Query submitted at: 2019-06-09 20:41:33 (Coordinator: http://node-3:25000)
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=2f4bbba8475bfaae:f9fd1b6100000000
+----+-----+-----+-----+-----+
| id | name  | age  | address | salary |
+----+-----+-----+-----+-----+
| 1  | Ramesh | 32   | Ahmedabad | 20000 |
| 4  | Chaitali | 25   | Mumbai | 35000 |
+----+-----+-----+-----+-----+
Fetched 2 row(s) in 0.25s
[node-3:21000] > select * from employee order by id limit 2 offset 1;
Query: select * from employee order by id limit 2 offset 1
Query submitted at: 2019-06-09 20:41:44 (Coordinator: http://node-3:25000)
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=65451b2e5036c661:56ce03ad00000000
+----+-----+-----+-----+-----+
| id | name  | age  | address | salary |
+----+-----+-----+-----+-----+
| 2  | Khilan | 25   | Delhi | 15000 |
| 4  | Chaitali | 25   | Mumbai | 35000 |
+----+-----+-----+-----+-----+
```




2.12. with 子句

如果查询太复杂，我们可以为复杂部分定义别名，并使用 Impala 的 with 子句将它们包含在查询中。

```
with x as (select 1), y as (select 2) (select * from x union y);
```

例如：使用 with 子句显示年龄大于 25 的员工和客户的记录。

```
with t1 as (select * from customers where age>25),
```

```
    t2 as (select * from employee where age>25)
```

```
(select * from t1 union select * from t2);
```

2.13. distinct

Impala 中的 distinct 运算符用于通过删除重复值来获取唯一值。

```
select distinct columns... from table_name;
```



六、 Impala 数据导入方式

1. load data

首先创建一个表：

```
create table user(id int ,name string,age int ) row format delimited fields
```

terminated by "\\t";

```
[node-3:21000] > create table user(id int ,name string,age int ) row format delimited fields terminated by "\\t";
Query: create table user(id int ,name string,age int ) row format delimited fields terminated by "\\t"
Fetched 0 row(s) in 0.05s
[node-3:21000] >
```

准备数据 user.txt 并上传到 hdfs 的 /user/impala 路径下去

```
[root@node-2 ~]# vim user.txt
1      allen    18
2      kobe     19

hadoop fs -mkdir /user/impala/
hadoop fs -put user.txt /user/impala/
hadoop fs -chmod -R 777 /user/impala/
```

加载数据

```
load data inpath '/user/impala/' into table user;
```

查询加载的数据

```
select * from user;
```

```
[node-3:21000] > load data inpath '/user/impala/' into table user;
Query: load data inpath '/user/impala/' into table user
+-----+
| summary |
+-----+
| Loaded 1 file(s). Total files in destination location: 1 |
+-----+
Fetched 1 row(s) in 0.13s
[node-3:21000] > select * from user;
Query: select * from user
Query submitted at: 2019-06-09 20:52:46 (Coordinator: http://node-3:25000)
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=1b43d1e0508fb
+-----+
| id | name | age |
+-----+
| 1 | allen | 18 |
| 2 | kobe | 19 |
+-----+
Fetched 2 row(s) in 0.13s
```

如果查询不拿到数据，那么需要刷新一遍数据表。

```
refresh user;
```



2. insert into values

这种方式非常类似于 RDBMS 的数据插入方式。

```
create table t_test2(id int,name string);
```

```
insert into table t_test2 values(1,"zhangsan");
```

```
[node-3:21000] > create table t_test2(id int,name string);
Query: create table t_test2(id int,name string)
Fetches 0 row(s) in 0.04s
[node-3:21000] > insert into table t_test2 values(1,"zhangsan");
Query: insert into table t_test2 values(1,"zhangsan")
Query submitted at: 2019-06-09 20:58:53 (Coordinator: http://node-3:25000)
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=...
Modified 1 row(s) in 4.46s
[node-3:21000] > select * from t_test2;
Query: select * from t_test2
Query submitted at: 2019-06-09 20:59:12 (Coordinator: http://node-3:25000)
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=...
+-----+
| id | name |
+-----+
| 1 | zhangsan |
+-----+
Fetches 1 row(s) in 0.13s
```

3. insert into select

插入一张表的数据来自于后面的 select 查询语句返回的结果。

```
[node-3:21000] > create table t_test3(id int);
Query: create table t_test3(id int)
Fetches 0 row(s) in 0.05s
[node-3:21000] > insert into table t_test3 select id from t_test2;
Query: insert into table t_test3 select id from t_test2
Query submitted at: 2019-06-09 21:00:38 (Coordinator: http://node-3:25000)
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=e84eea41cc1722df:ab6df2e300000000
Modified 1 row(s) in 5.66s
[node-3:21000] > select * from t_test3;
Query: select * from t_test3
Query submitted at: 2019-06-09 21:00:48 (Coordinator: http://node-3:25000)
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=d94df6a1c83fcf7a:b8b3bb8100000000
+-----+
| id |
+-----+
| 1 |
+-----+
Fetches 1 row(s) in 0.12s
[node-3:21000] >
```

4. create as select

建表的字段个数、类型、数据来自于后续的 select 查询语句。

```
[node-3:21000] > create table t_test4 as select name from t_test2;
Query: create table t_test4 as select name from t_test2
Query submitted at: 2019-06-09 21:02:10 (Coordinator: http://node-3:25000)
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=84f87a7574e319b:40fd9ab...
+-----+
| summary |
+-----+
| Inserted 1 row(s) |
+-----+
Fetches 1 row(s) in 0.42s
[node-3:21000] > select * from t_test4;
Query: select * from t_test4
Query submitted at: 2019-06-09 21:02:16 (Coordinator: http://node-3:25000)
Query progress can be monitored at: http://node-3:25000/query_plan?query_id=524dc098f9dae926:a50ad8...
+-----+
| name |
+-----+
| zhangsan |
+-----+
Fetches 1 row(s) in 0.12s
```

七、 Impala 的 java 开发


















在实际工作当中，因为 impala 的查询比较快，所以可能会有使用到 impala 来做数据库查询的情况，可以通过 java 代码来进行操作 impala 的查询。

1. 下载 impala jdbc 依赖

下载路径：

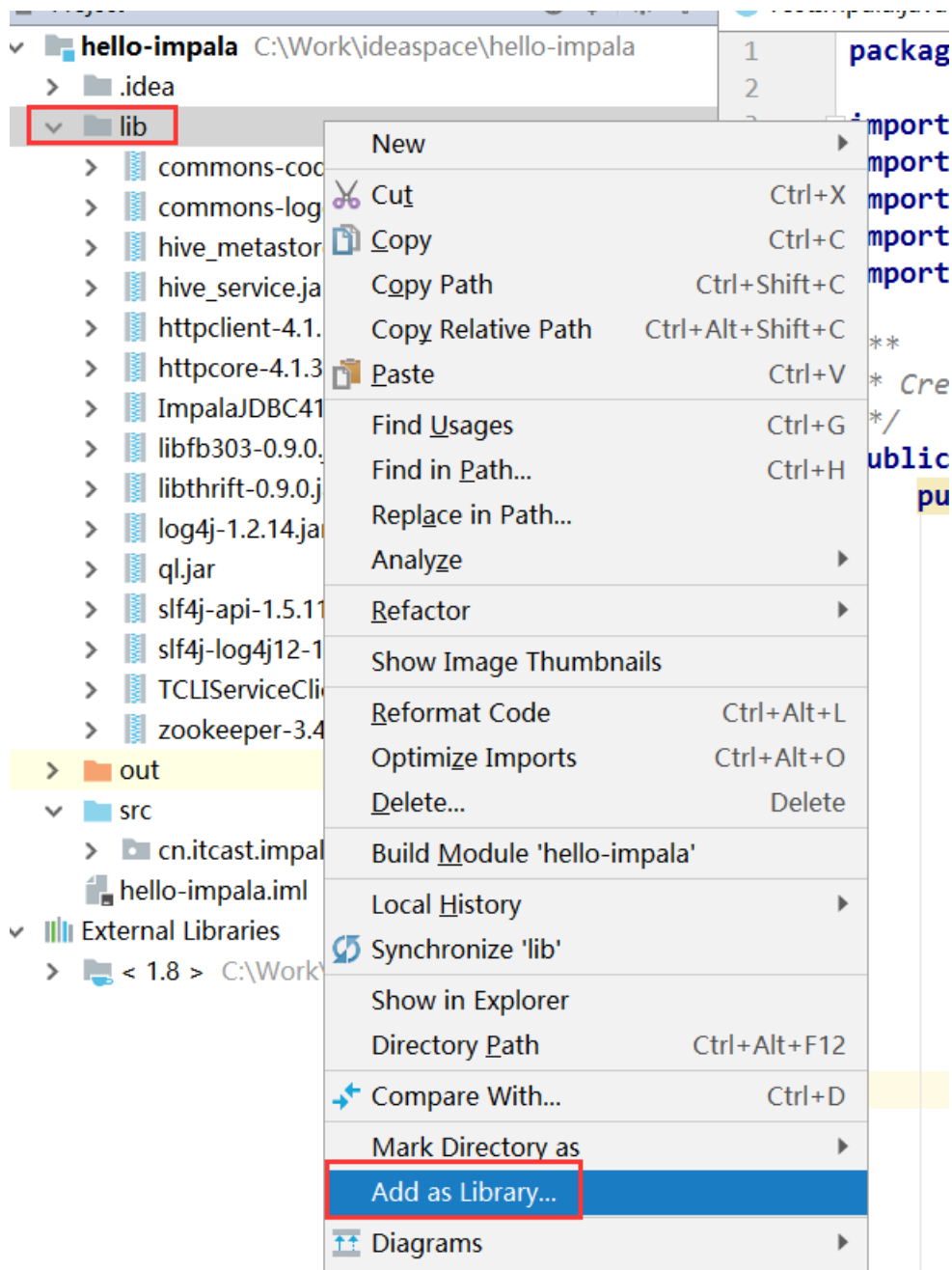
<https://www.cloudera.com/downloads/connectors/impala/jdbc/2-5-28.html>

因为 cloudera 属于商业公司性质，其提供的 jar 并不会出现在开源的 maven 仓库中，如果在企业中需要使用，请添加到企业 maven 私服。

 zookeeper-3.4.6.jar	79k
 TCLIServiceClient.jar	1,30k
 slf4j-log4j12-1.5.11.jar	2k
 slf4j-api-1.5.11.jar	2k
 ql.jar	29k
 log4j-1.2.14.jar	36k
 libthrift-0.9.0.jar	34k
 libfb303-0.9.0.jar	27k
 ImpalaJDBC41.jar	1,60k
 httpcore-4.1.3.jar	18k
 httpclient-4.1.3.jar	35k
 hive_service.jar	59k
 hive_metastore.jar	7,67k
 commons-logging-1.1.1.jar	6k
 commons-codec-1.3.jar	4k
 Cloudera-JDBC-Driver-for-Impala-Release-Notes.pdf	5k
 Cloudera-JDBC-Driver-for-Impala-Install-Guide.pdf	45k

2. 创建 java 工程

创建普通 java 工程，把依赖添加工程。





3. java api

```
public static void test(){
    Connection con = null;
    ResultSet rs = null;
    PreparedStatement ps = null;
    String JDBC_DRIVER = "com.cloudera.impala.jdbc41.Driver";
    String CONNECTION_URL = "jdbc:impala://node-3:21050";
    try
    {
        Class.forName(JDBC_DRIVER);
        con = (Connection) DriverManager.getConnection(CONNECTION_URL);
        ps = con.prepareStatement("select * from my_db.employee;");
        rs = ps.executeQuery();
        while (rs.next())
        {
            System.out.println(rs.getString(1));
            System.out.println(rs.getString(2));
            System.out.println(rs.getString(3));
        }
    } catch (Exception e)
    {
        e.printStackTrace();
    } finally
    {
        try {
            rs.close();
            ps.close();
            con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

public static void main(String[] args) {
    test();
}
```