

## 附录 A

# 编译安装 Hadoop

## A.1 编译 Hadoop

### A.1.1 搭建环境

#### 1. 安装并设置 maven

(1) 下载 maven 安装包。

建议安装 3.0 以上版本 (Spark 2.0 编译要求 Maven 3.3.9 及以上版本), 本次安装选择的是 maven 3.3.9 的二进制包, 下载地址如下:

`http://mirror.bit.edu.cn/apache/maven/maven-3/`

(2) 上传 git 并解压缩。

把下载的 maven 安装包上传到/home/spark/work 目录,使用如下命令解压缩并把文件夹移动到/app/soft 目录下:

```
$cd /home/spark/work
$tar -zxf apache-maven-3.3.9-bin.tar.gz
$mv maven-3.3.9 /app/soft
$ll /app/soft
```

(3) 编译安装。

在/etc/profile 配置文件中加入如下设置:

```
export PATH=/app/soft/maven-3.3.9/bin:$PATH
```

修改/etc/profile 配置文件并验证配置是否成功, 如图附录 A-1 所示。

```
$source /etc/profile
$mvn -version
```

## 2 | 图解 Spark: 核心技术与案例实战

```
master
[spark@master soft]$ source /etc/profile
[spark@master soft]$ mvn -version
Apache Maven 3.3.9 (r01de14724cdef164cd33c7c8c2fe155faf9602da; 201
28+0800)
Maven home: /app/soft/maven-3.3.9
Java version: 1.7.0_55, vendor: Oracle Corporation
Java home: /app/soft/jdk1.7.0_55/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "2.6.32-431.el6.x86_64", arch: "amd64",
"
[spark@master soft]$
```

图 附录 A-1 查看 Maven 是否安装成功

### 2. 使用 yum 安装必要软件

以 root 用户使用 yum 安装 svn、gcc 等编译所需要的软件:

```
#yum install svn
#yum install autoconf automake libtool cmake
#yum install ncurses-devel
#yum install openssl-devel
#yum install gcc*
```

### 3. 安装并设置 protobuf

注: 该程序包需要在 gcc 安装完毕后才能安装, 否则提示无法找到 gcc 编译器。

(1) 下载 protobuf 安装包, 如图附录 A-2 所示。

下载链接为 <https://code.google.com/p/protobuf/downloads/list>

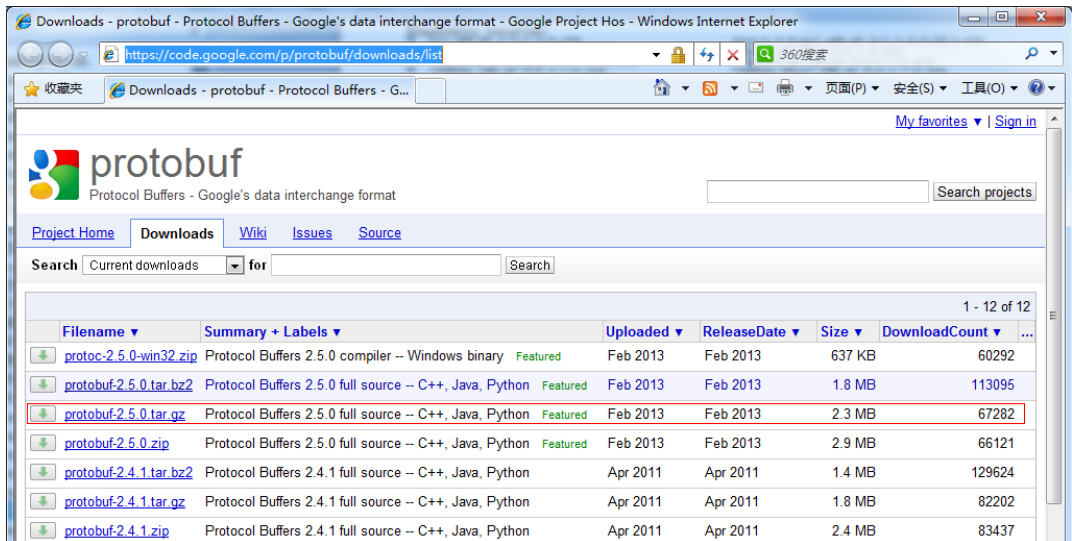


图 附录 A-2 Protobuf 下载页面

(2) 解压安装包并移动目录。

把 `protobuf-2.5.0.tar.gz` 安装包上传到 `/home/spark/work` 目录，通过如下命令把该安装包解压并移动到 `/app/soft` 目录中：

```
$tar -zxvf protobuf-2.5.0.tar.gz
$mv protobuf-2.5.0 /app/soft
$ll /app/soft
```

(3) 编译安装。

进入目录以 `root` 用户运行如下命令对 `protobuf` 进行编译安装，该过程比较慢，需要花费十几分钟的时间：

```
#cd /app/soft/protobuf-2.5.0
#./configure
#make
#make check
#make install
```

(4) 验证是否安装成功。

编译安装成功之后，通过如下方式来验证是否安装成功，如图附录 A-3 所示。

```
#protoc
```

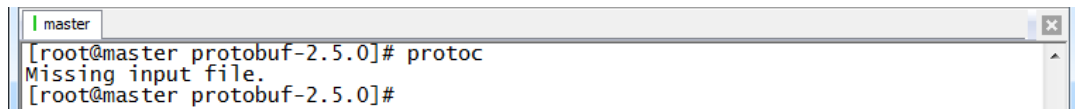


图 附录 A-3 确认 Protobuf 是否安装成功

## A.1.2 编译 Hadoop

### 1. 下载 Hadoop 源代码并解压

用户可以在 `apache` 官网或者镜像站点下载 `hadoop` 源代码包，比如在下面地址中选择下载 `hadoop-2.7.2-src.tar.gz` 源代码包：

<http://apache.fayea.com/hadoop/common/hadoop-2.7.2/>

下载后把源代码包上传到 `/home/spark/work` 目录中解压，然后移动到 `/app/compile` 目录：

```
$cd /home/spark/work
$tar -zxvf hadoop-2.7.2-src.tar.gz
$mv hadoop-2.7.2-src /app/compile
$ll /app/compile
```

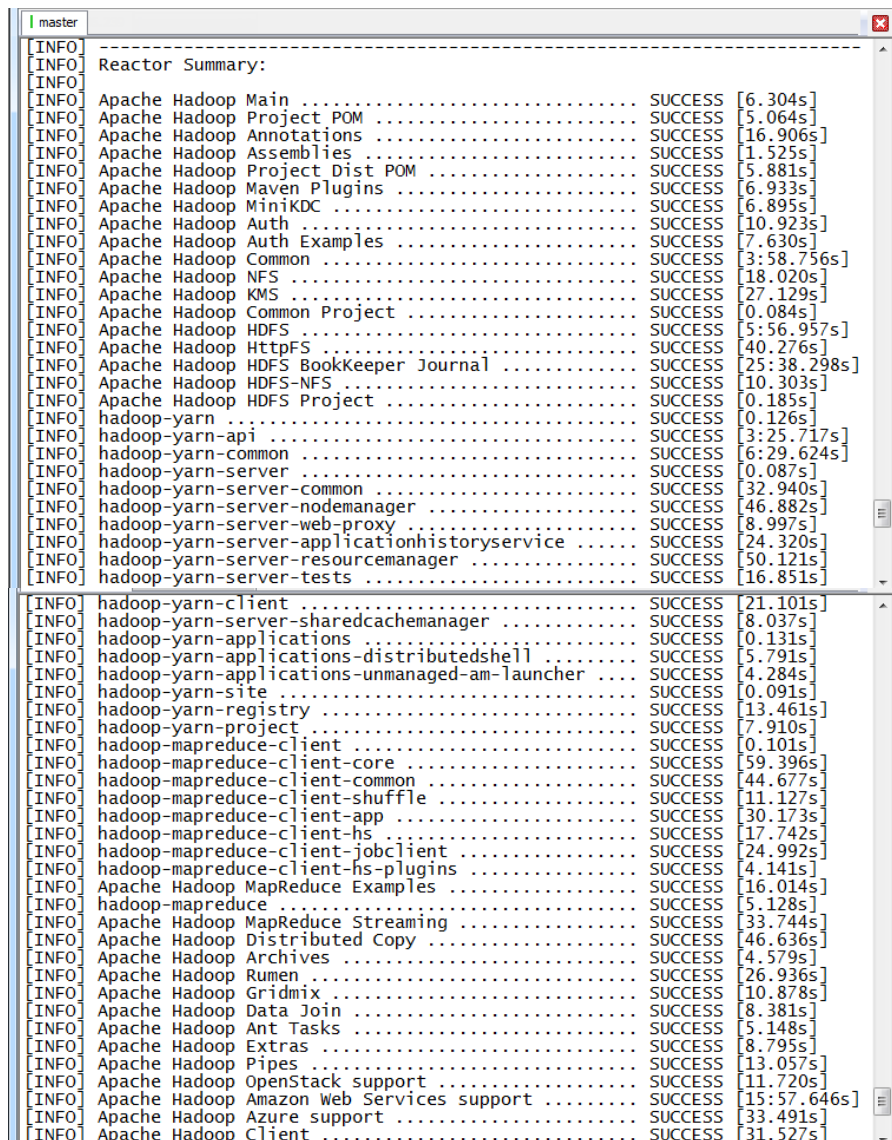
### 2. 编译 Hadoop 源代码

在 `Hadoop` 源代码的根目录执行如下命令：

#### 4 | 图解 Spark: 核心技术与案例实战

```
$cd /app/compile/hadoop-2.7.2-src
$mvn package -Pdist,native -DskipTests -Dtar
```

该过程需要 64 个任务进行编译，耗费的时间较长，在编译过程需要联网，从网络中下载所需要依赖包。由于依赖包速度较慢，可以打开新的命令终端使用 `$du -sh` 查看整个目录或 `$du -sh *` 子目录大小变化，该过程经常卡死或出现异常。这种情况下可以中断编译过程，重新执行命令进行编译，编译完成后截图如图附录 A-4 所示。



[INFO]	Reactor Summary:		
[INFO]	Apache Hadoop Main	SUCCESS	[6.304s]
[INFO]	Apache Hadoop Project POM	SUCCESS	[5.064s]
[INFO]	Apache Hadoop Annotations	SUCCESS	[16.906s]
[INFO]	Apache Hadoop Assemblies	SUCCESS	[1.525s]
[INFO]	Apache Hadoop Project Dist POM	SUCCESS	[5.881s]
[INFO]	Apache Hadoop Maven Plugins	SUCCESS	[6.933s]
[INFO]	Apache Hadoop MiniKDC	SUCCESS	[6.893s]
[INFO]	Apache Hadoop Auth	SUCCESS	[10.923s]
[INFO]	Apache Hadoop Auth Examples	SUCCESS	[7.630s]
[INFO]	Apache Hadoop Common	SUCCESS	[3:58.756s]
[INFO]	Apache Hadoop NFS	SUCCESS	[18.020s]
[INFO]	Apache Hadoop KMS	SUCCESS	[27.129s]
[INFO]	Apache Hadoop Common Project	SUCCESS	[0.084s]
[INFO]	Apache Hadoop HDFS	SUCCESS	[5:56.957s]
[INFO]	Apache Hadoop HTTPFS	SUCCESS	[40.276s]
[INFO]	Apache Hadoop HDFS BookKeeper Journal	SUCCESS	[25:38.298s]
[INFO]	Apache Hadoop HDFS-NFS	SUCCESS	[10.303s]
[INFO]	Apache Hadoop HDFS Project	SUCCESS	[0.185s]
[INFO]	hadoop-yarn	SUCCESS	[0.126s]
[INFO]	hadoop-yarn-api	SUCCESS	[3:25.717s]
[INFO]	hadoop-yarn-common	SUCCESS	[6:29.624s]
[INFO]	hadoop-yarn-server	SUCCESS	[0.087s]
[INFO]	hadoop-yarn-server-common	SUCCESS	[32.940s]
[INFO]	hadoop-yarn-server-nodemanager	SUCCESS	[46.882s]
[INFO]	hadoop-yarn-server-web-proxy	SUCCESS	[8.997s]
[INFO]	hadoop-yarn-server-applicationhistoryservice	SUCCESS	[24.320s]
[INFO]	hadoop-yarn-server-resourcemanager	SUCCESS	[50.121s]
[INFO]	hadoop-yarn-server-tests	SUCCESS	[16.851s]
[INFO]	hadoop-yarn-client	SUCCESS	[21.101s]
[INFO]	hadoop-yarn-server-sharedcachemanager	SUCCESS	[8.037s]
[INFO]	hadoop-yarn-applications	SUCCESS	[0.131s]
[INFO]	hadoop-yarn-applications-distributedshell	SUCCESS	[5.791s]
[INFO]	hadoop-yarn-applications-unmanaged-am-launcher	SUCCESS	[4.284s]
[INFO]	hadoop-yarn-site	SUCCESS	[0.091s]
[INFO]	hadoop-yarn-registry	SUCCESS	[13.461s]
[INFO]	hadoop-yarn-project	SUCCESS	[7.910s]
[INFO]	hadoop-mapreduce-client	SUCCESS	[0.101s]
[INFO]	hadoop-mapreduce-client-core	SUCCESS	[59.396s]
[INFO]	hadoop-mapreduce-client-common	SUCCESS	[44.677s]
[INFO]	hadoop-mapreduce-client-shuffle	SUCCESS	[11.127s]
[INFO]	hadoop-mapreduce-client-app	SUCCESS	[30.173s]
[INFO]	hadoop-mapreduce-client-hs	SUCCESS	[17.742s]
[INFO]	hadoop-mapreduce-client-jobclient	SUCCESS	[24.992s]
[INFO]	hadoop-mapreduce-client-hs-plugins	SUCCESS	[4.141s]
[INFO]	Apache Hadoop MapReduce Examples	SUCCESS	[16.014s]
[INFO]	hadoop-mapreduce	SUCCESS	[5.128s]
[INFO]	Apache Hadoop MapReduce Streaming	SUCCESS	[33.744s]
[INFO]	Apache Hadoop Distributed Copy	SUCCESS	[46.636s]
[INFO]	Apache Hadoop Archives	SUCCESS	[4.579s]
[INFO]	Apache Hadoop Rumen	SUCCESS	[26.936s]
[INFO]	Apache Hadoop Gridmix	SUCCESS	[10.878s]
[INFO]	Apache Hadoop Data Join	SUCCESS	[8.381s]
[INFO]	Apache Hadoop Ant Tasks	SUCCESS	[5.148s]
[INFO]	Apache Hadoop Extras	SUCCESS	[8.795s]
[INFO]	Apache Hadoop Pipes	SUCCESS	[13.057s]
[INFO]	Apache Hadoop OpenStack support	SUCCESS	[11.720s]
[INFO]	Apache Hadoop Amazon Web Services support	SUCCESS	[15:57.646s]
[INFO]	Apache Hadoop Azure support	SUCCESS	[33.491s]
[INFO]	Apache Hadoop Client	SUCCESS	[31.527s]

[INFO]	Apache Hadoop Client .....	SUCCESS	[31.527s]
[INFO]	Apache Hadoop Mini-Cluster .....	SUCCESS	[0.546s]
[INFO]	Apache Hadoop Scheduler Load Simulator .....	SUCCESS	[25.224s]
[INFO]	Apache Hadoop Tools Dist .....	SUCCESS	[1:23.672s]
[INFO]	Apache Hadoop Tools .....	SUCCESS	[0.053s]
[INFO]	Apache Hadoop Distribution .....	SUCCESS	[1:38.845s]
[INFO]	-----		
[INFO]	BUILD SUCCESS		
[INFO]	-----		
[INFO]	Total time: 1:19:15.883s		
[INFO]	Finished at: Thu Mar 24 16:59:35 CST 2016		
[INFO]	Final Memory: 114M/697M		
[INFO]	-----		

图 附录 A-4 Hadoop 编译结果

### 3. 验证编译是否成功

到 `hadoop-dist/target/hadoop-2.7.2/lib/native` 目录中, 使用 `$file ./libhadoop.so.1.0.0` 命令查看 `libhadoop.so.1.0.0` 属性, 该文件为 ELF 64-bit LSB, 则表示文件成功编译为 64 位, 如图附录 A-5 所示。其中打包好的 `hadoop-2.7.2.tar.gz` 文件存在 `hadoop-dist/target` 目录中, 作为后续部署的安装包。

```

[spark@compiler ~]$ cd /app/compile/hadoop-2.7.2-src/
[spark@compiler hadoop-2.7.2-src]$ cd hadoop-dist/target/hadoop-2.7.2/lib/native
[spark@compiler native]$ file *
libhadoop.a:          current ar archive
libhadooppipes.a:     current ar archive
libhadoop.so:         symbolic link to 'libhadoop.so.1.0.0'
libhadoop.so.1.0.0:   ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, not stripped
libhadooputils.a:     current ar archive
libhdfs.a:            current ar archive
libhdfs.so:           symbolic link to 'libhdfs.so.0.0.0'
libhdfs.so.0.0.0:     ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, not stripped
[spark@compiler native]$

```

图 附录 A-5 验证 Hadoop 编译是否成功

## A.2 安装 Hadoop

由于在实战过程中, 需要使用 HDFS 文件系统, 以及在介绍运行架构需要使用 YARN 调度框架需要安装 Hadoop, 这里使用的是 Hadoop 2.7.2 版本。

### A.2.1 修改配置文件

#### 1. 上传并解压 Hadoop 安装包

使用前面编译好的 `hadoop-2.7.2` 安装包, 或者从 `apache` 网站上下载, 上传到 `master` 节点的 `/home/spark/work` 目录下, 解压缩并移动到 `/app/spark` 目录下:

## 6 | 图解 Spark: 核心技术与案例实战

```
$cd /home/spark/work
$tar -zxvf hadoop-2.7.2.tar.gz
$mv hadoop-2.7.2 /app/spark
$ll /app/spark
```

### 2. 在 Hadoop 目录下创建子目录

以 hadoop 用户登录在/app/spark/hadoop-2.7.2 目录下创建 tmp、name 和 data 目录:

```
$cd /app/spark/hadoop-2.7.2
$mkdir tmp
$mkdir name
$mkdir data
$ll
```

### 3. 配置 hadoop-env.sh

使用如下命令打开配置文件 hadoop-env.sh:

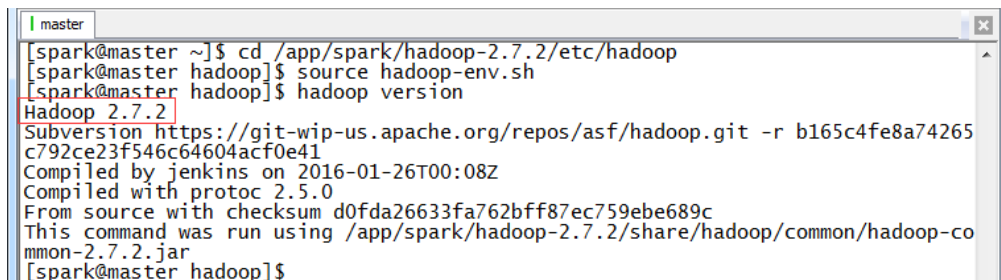
```
$cd /app/spark/hadoop-2.7.2/etc/hadoop
$sudo vi hadoop-env.sh
```

加入如下配置内容, 设置 JAVA\_HOME 和 PATH 路径:

```
export JAVA_HOME=/app/soft/jdk1.7.0_55
export PATH=$PATH:/app/spark/hadoop-2.7.2/bin
export HADOOP_CONF_DIR=/app/spark/hadoop-2.7.2/etc/hadoop
```

编译配置文件 hadoop-env.sh, 并确认生效, 如图附录 A-6 所示:

```
$source hadoop-env.sh
$hadoop version
```

A terminal window titled 'master' showing a series of commands and their outputs. The user navigates to the Hadoop configuration directory, sources the environment file, and runs the 'hadoop version' command. The output displays 'Hadoop 2.7.2' and detailed information about the distribution, including the source URL, commit hash, compilation date, and the JAR file used.

```
[spark@master ~]$ cd /app/spark/hadoop-2.7.2/etc/hadoop
[spark@master hadoop]$ source hadoop-env.sh
[spark@master hadoop]$ hadoop version
Hadoop 2.7.2
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r b165c4fe8a74265c792ce23f546c64604acf0e41
Compiled by jenkins on 2016-01-26T00:08Z
Compiled with protoc 2.5.0
From source with checksum d0fda26633fa762bff87ec759ebe689c
This command was run using /app/spark/hadoop-2.7.2/share/hadoop/common/hadoop-common-2.7.2.jar
[spark@master hadoop]$
```

图 附录 A-6 验证 Hadoop 部署是否正确

### 4. 配置 yarn-env.sh

在/app/spark/hadoop-2.7.2/etc/hadoop 打开配置文件 yarn-env.sh:

```
$cd /app/spark/hadoop-2.7.2/etc/hadoop
$sudo vi yarn-env.sh
```

加入配置内容, 设置 JAVA\_HOME 路径:

```
export JAVA_HOME=/app/soft/jdk1.7.0_55
```

使用如下命令编译配置文件 `yarn-env.sh`，使其生效：

```
$source yarn-env.sh
```

## 5. 配置 core-site.xml

使用如下命令打开 `core-site.xml` 配置文件：

```
$cd /app/spark/hadoop-2.7.2/etc/hadoop
```

```
$sudo vi core-site.xml
```

在配置文件中，按照如下内容进行配置：

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://master:9000</value>
  </property>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://master:9000</value>
  </property>
  <property>
    <name>io.file.buffer.size</name>
    <value>131072</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>file:/app/spark/hadoop-2.7.2/tmp</value>
    <description>Abase for other temporary directories.</description>
  </property>
  <property>
    <name>hadoop.proxyuser.hduser.hosts</name>
    <value>*</value>
  </property>
  <property>
    <name>hadoop.proxyuser.hduser.groups</name>
    <value>*</value>
  </property>
</configuration>
```

## 6. 配置 hdfs-site.xml

使用如下命令打开 `hdfs-site.xml` 配置文件：

```
$cd /app/spark/hadoop-2.7.2/etc/hadoop
```

```
$sudo vi hdfs-site.xml
```

在配置文件中，按照如下内容进行配置：

```
<configuration>
  <property>
    <name>dfs.namenode.secondary.http-address</name>
    <value>master:9001</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/app/spark/hadoop-2.7.2/name</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/app/spark/hadoop-2.7.2/data</value>
  </property>
  <property>
    <name>dfs.replication</name>
    <value>2</value>
  </property>
  <property>
    <name>dfs.webhdfs.enabled</name>
    <value>true</value>
  </property>
</configuration>
```

## 7. 配置 mapred-site.xml

默认情况下不存在 mapred-site.xml 文件，可以从模板复制一份，并打开该配置文件：

```
$cd /app/spark/hadoop-2.7.2/etc/hadoop
$cp mapred-site.xml.template mapred-site.xml
$sudo vi mapred-site.xml
```

在配置文件中，按照如下内容进行配置：

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>mapreduce.jobhistory.address</name>
    <value>master:10020</value>
  </property>
  <property>
    <name>mapreduce.jobhistory.webapp.address</name>
    <value>master:19888</value>
  </property>
</configuration>
```



```

    </property>
</configuration>

```

## 8. 配置 yarn-site.xml

使用如下命令打开 `yarn-site.xml` 配置文件:

```

$cd /app/spark/hadoop-2.7.2/etc/hadoop
$sudo vi yarn-site.xml

```

在配置文件中, 按照如下内容进行配置:

```

<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>
  <property>
    <name>yarn.resourcemanager.address</name>
    <value>master:8032</value>
  </property>
  <property>
    <name>yarn.resourcemanager.scheduler.address</name>
    <value>master:8030</value>
  </property>
  <property>
    <name>yarn.resourcemanager.resource-tracker.address</name>
    <value>master:8031</value>
  </property>
  <property>
    <name>yarn.resourcemanager.admin.address</name>
    <value>master:8033</value>
  </property>
  <property>
    <name>yarn.resourcemanager.webapp.address</name>
    <value>master:8088</value>
  </property>
</configuration>

```

## 9. 配置 Slaves 文件

使用 `$sudo vi slaves` 打开从节点配置文件, 在文件中加入 `master`、`slave1` 和 `slave2` 节点作为数据节点 (DataNode):

```

master
slave1
slave2

```

## 10. 向各节点分发 Hadoop 程序

确认 slave1 和 slave2 节点/app/spark 所属组 and 用户均为 spark, 然后进入 mater 节点/app/spark 目录, 使用如下命令把 hadoop-2.7.2 文件夹复制到 slave1 和 slave2 节点:

```

$cd /app/spark
$scp -r hadoop-2.7.2 spark@slave1:/app/spark/
$scp -r hadoop-2.7.2 spark@slave2:/app/spark/

```

## A.2.2 启动并验证部署

### 1. 格式化 NameNode

格式化 NameNode, 如图附录 A-7 所示。

```

$cd /app/spark/hadoop-2.7.2/
$./bin/hdfs namenode -format
16/04/09 00:10:34 INFO util.GSet: capacity = 2^18 = 262144 entries
16/04/09 00:10:34 INFO namenode.FSNamesystem: dfs.namenode.safemode.threshold-pct = 0.9990000128746033
16/04/09 00:10:34 INFO namenode.FSNamesystem: dfs.namenode.safemode.min.datanodes = 0
16/04/09 00:10:34 INFO namenode.FSNamesystem: dfs.namenode.safemode.extension = 30000
16/04/09 00:10:34 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.window.num.buckets = 10
16/04/09 00:10:34 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.num.users = 10
16/04/09 00:10:34 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.windows.minutes = 1,5,25
16/04/09 00:10:34 INFO namenode.FSNamesystem: Retry cache on namenode is enabled
16/04/09 00:10:34 INFO namenode.FSNamesystem: Retry cache will use 0.03 of total heap and retry cache entry expiry time
16/04/09 00:10:34 INFO util.GSet: Computing capacity for map NameNodeRetryCache
16/04/09 00:10:34 INFO util.GSet: VM type = 64-bit
16/04/09 00:10:34 INFO util.GSet: 0.0299999999329447746% max memory 966.7 MB = 297.0 KB
16/04/09 00:10:34 INFO util.GSet: capacity = 2^15 = 32768 entries
16/04/09 00:10:34 INFO namenode.FSImage: Allocated new BlockPoolId: BP-2092510663-192.168.1.10-1460131834632
16/04/09 00:10:34 INFO common.Storage: Storage directory /app/spark/hadoop-2.7.2/name has been successfully formatted.
16/04/09 00:10:35 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
16/04/09 00:10:35 INFO util.ExitUtil: Exiting with status 0
16/04/09 00:10:35 INFO namenode.NameNode: SHUTDOWN_MSG:
/*****
SHUTDOWN_MSG: Shutting down NameNode at master/192.168.1.10
*****/

```

图 附录 A-7 格式化 NameNode

### 2. 启动并验证 HDFS

使用如下命令启动 HDFS:

```

$cd /app/spark/hadoop-2.7.2/sbin
$./start-dfs.sh

```

此时在 master 上面运行的进程有 NameNode、SecondaryNameNode 和 DataNode, 而 slave1 和 slave2 上面运行的进程有 NameNode 和 DataNode

### 3. 启动并验证 YARN

使用如下命令启动 YARN:

```
$cd /app/spark/hadoop-2.7.2/sbin  
$./start-yarn.sh
```

此时在 master 上运行的进程有 NameNode、SecondaryNameNode、DataNode、NodeManager 和 ResourceManager, 而 slave1 和 slave2 上面运行的进程有 NameNode、DataNode 和 NodeManager。

## 附录 B

# 安装 MySQL 数据库

### B.1 卸载旧的 MySQL 程序

#### 1. 查找以前是否安装有 mysql

使用命令查看是否已经安装过 mysql:

```
#rpm -qa | grep -i mysql
```

如果没有结果，则可以进行 mysql 数据库安装。

#### 2. 如果有，则先停止 mysql 服务并删除之前安装的 mysql

如果之前安装过 MySQL，则先停止 mysql 服务，然后删除之前安装的 mysql。

```
#rpm -ev MySQL-server-5.6.21-1.el6.x86_64
```

```
#rpm -ev MySQL-devel-5.6.21-1.el6.x86_64
```

```
#rpm -ev MySQL-client-5.6.21-1.el6.x86_64
```

如果未安装 mysql，但是存在 CentOS 自带 mysql-libs-5.1.71-1.el6.x86\_64，使用下面的命令卸载即可。

```
#rpm -ev --nodeps mysql-libs-5.1.71-1.el6.x86_64
```

#### 3. 查找之前老版本 mysql 的目录并且删除老版本 mysql 的文件和库

```
#find / -name mysql
```

如果存在目录，则删除对应的 mysql 目录，按如下操作删除已经存在的目录：

```
#rm -rf /usr/lib64/mysql
```

```
#rm -rf /var/lib/mysql
```

#### 4. 再次查找机器是否安装 mysql

```
#rpm -qa | grep -i mysql
```

无结果，说明已经卸载彻底、接下来直接安装 mysql 即可。

## B.2 下载并安装 MySQL

### B.2.1 下载 MySQL 安装包

从 MySQL 网站下载地址 <http://dev.mysql.com/downloads/mysql/#downloads>，使用系统为 CentOS 选择 Red Hat Enterprise Linux/Oracle 系列，如图附录 B-1 所示。



图 附录 B-1 下载 MySQL 选择界面

以 MySQL5.6.21 版本为例，操作系统为 64 位，选择如下安装包进行下载：

- MySQL-client-5.6.21-1.el6.x86\_64
- MySQL-server-5.6.21-1.el6.x86\_64
- MySQL-devel-5.6.21-1.el6.x86\_64

下载在本地如图附录 B-2 所示。




 MySQL-client-5.6.21-1.el6.x86_64.rpm	2014/11/21 11:11	360压缩	18,130 KB
 MySQL-devel-5.6.21-1.el6.x86_64.rpm	2014/11/21 11:04	360压缩	3,323 KB
 MySQL-server-5.6.21-1.el6.x86_64.rpm	2014/11/21 11:14	360压缩	54,350 KB

图 附录 B-2 下载 MySQL 结果列表

### B.2.2 安装 MySQL 程序

把下载的文件上传到/home/spark/work 目录，进入该目录，安装 mysql 服务端：

```
$cd /home/spark/work
#rpm -ivh MySQL-server-5.6.21-1.el6.x86_64.rpm
```

安装 mysql 客户端和 mysql-devel：

```
#rpm -ivh MySQL-client-5.6.21-1.el6.x86_64.rpm
#rpm -ivh MySQL-devel-5.6.21-1.el6.x86_64.rpm
```

## B.3 设置 MySQL

### B.3.1 设置 root 密码

#### 1. 停止 mysql 服务

使用如下命令停止 mysql 服务:

```
#service mysql stop
#service mysql status
```

#### 2. 跳过验证启动 mysql

在 CentOS6.5 下安装 mysql 设置 root 密码时, 出现如下错误:

```
/usr/bin/mysqladmin: connect to server at 'localhost' failed
error: 'Access denied for user 'root'@'localhost' (using password: NO) '
```

可以进入安全模式进行设置 root 密码, 可以使用如下步骤进行: 使用如下命令验证启动 mysql, 由于 & 结尾是后台运行进程, 运行该命令可以再打开命令窗口或者 Ctrl+C 组合键继续进行下一步操作。由于 mysql 启动时间会长点, 需要等待几分钟再查看启动状态, 如图附录 B-3 所示。

```
#service mysql stop
#mysqld_safe --skip-grant-tables &
#service mysql status

[root@master work]# mysqld_safe --skip-grant-tables &
[1] 9834
[root@master work]# 160413 16:05:46 mysqld_safe Logging to '/var/lib/mysql/master.err'.
160413 16:05:46 mysqld_safe Starting mysqld daemon with databases from /var/lib/mysql

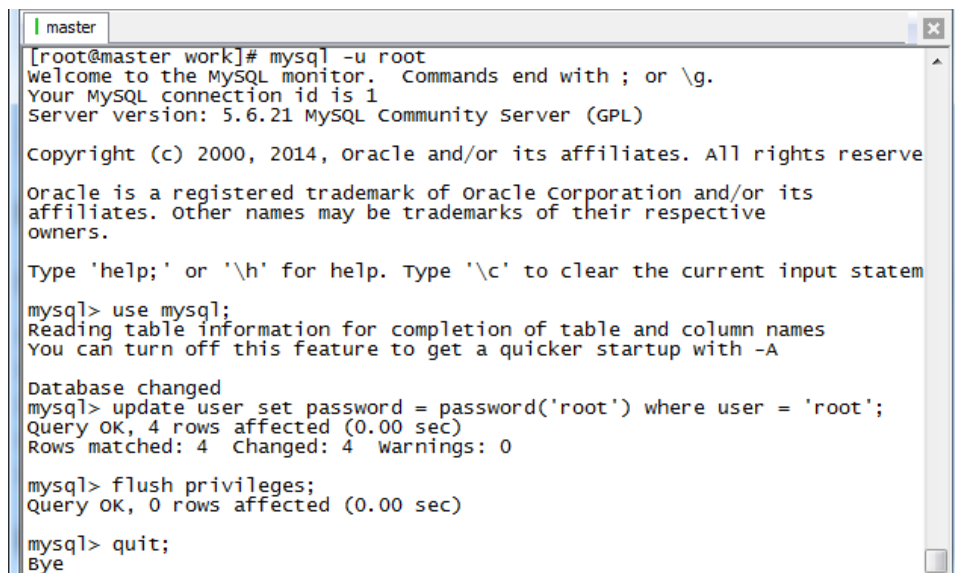
[root@master work]#
[root@master work]#
[root@master work]# service mysql status
MySQL running (9927) [ OK ]
[root@master work]# █
```

图 附录 B-3 设置安全模式登录

#### 3. 跳过验证启动 MySQL

验证 mysql 服务已经在后台运行后, 执行如下语句, 如图附录 B-4 所示, 其中后面 3 条命令是在 mysql 语句中:

```
mysql -u root
mysql> use mysql;
mysql> update user set password = password('root') where user = 'root';
mysql> flush privileges;
mysql> quit;
```



```

[root@master work]# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.21 MySQL Community Server (GPL)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement

mysql> use mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> update user set password = password('root') where user = 'root';
Query OK, 4 rows affected (0.00 sec)
Rows matched: 4  Changed: 4  Warnings: 0

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)

mysql> quit;
Bye

```

图 附录 B-4 以安全模式设置 root 密码

#### 4. 跳过验证启动 MySQL

重启 mysql 服务并查看状态：

```

#service mysql stop
#service mysql start
#service mysql status

```

### B.3.2 设置 Hive 用户

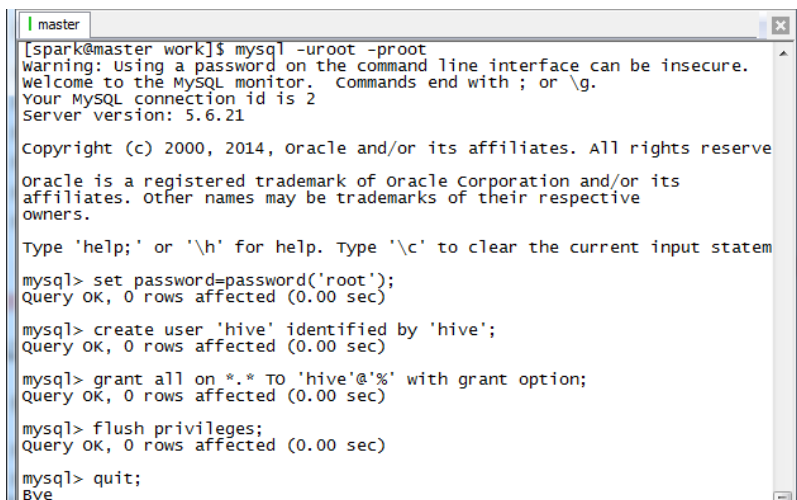
进入 mysql 命令行，创建 Hive 用户并赋予所有权限，如图附录 B-5 所示：

```

mysql -uroot -proot
mysql> set password=password('root');
mysql> create user 'hive' identified by 'hive';
mysql> grant all on *.* TO 'hive'@'%' with grant option;
mysql> flush privileges;
mysql> quit;

```

**【注意】**如果是 root 第一次登录数据库，需要重新设置一下密码，所报异常信息如下：ERROR 1820 (HY000): You must SET PASSWORD before executing this statement.



```

[spark@master work]$ mysql -uroot -proot
Warning: Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.6.21

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> set password=password('root');
Query OK, 0 rows affected (0.00 sec)

mysql> create user 'hive' identified by 'hive';
Query OK, 0 rows affected (0.00 sec)

mysql> grant all on *.* to 'hive'@'%' with grant option;
Query OK, 0 rows affected (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)

mysql> quit;
Bye

```

图 附录 B-5 在 MySQL 中创建 hive 用户

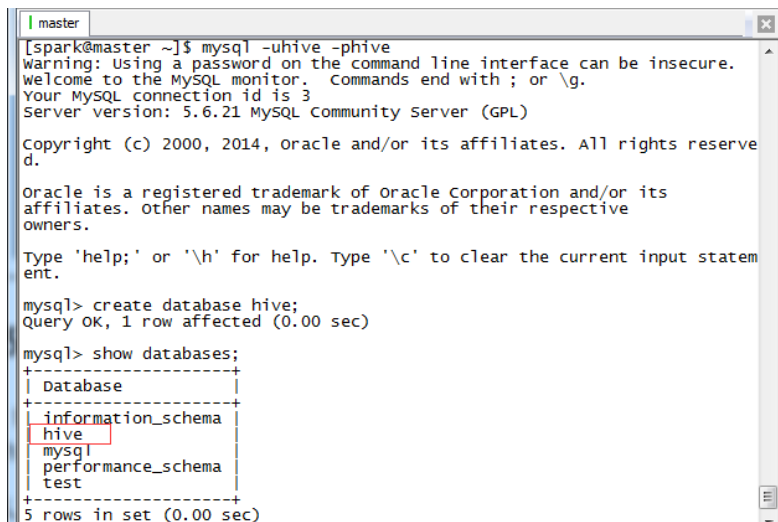
### B.3.3 创建 Hive 数据库

使用 hive 用户登录，创建 Hive 数据库，如图 附录 B-6 所示：

```

mysql -uhive -phive
mysql> create database hive;
mysql> show databases;

```



```

[spark@master ~]$ mysql -uhive -phive
Warning: Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.6.21 MySQL Community Server (GPL)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database hive;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| hive      |
| mysql    |
| performance_schema |
| test     |
+-----+
5 rows in set (0.00 sec)

```

图 附录 B-6 在 MySQL 中创建 Hive 数据库



## 附录 C

# 编译安装 Hive

如果需要直接安装 Hive，可以跳过编译步骤，从 Hive 的官网下载编译好的安装包，下载地址为 <http://hive.apache.org/downloads.html>。

## C.1 编译 Hive

### C.1.1 下载 Hive 源代码包

在 Hive 的官网下载页面上下载，为加快下载速度选择中国境内的镜像，并下载 apache-hive-1.2.1-

### C.1.2 编译 Hive

编译 Hive 源代码的时候，需要从网上下载依赖包，所以整个编译过程机器必须保证在联网状态。编译执行如下脚本：

```
$cd /app/compile/hive-1.2.1-src
$export MAVEN_OPTS="-Xmx2g -XX:MaxPermSize=512M -XX:ReservedCodeCacheSize= 512m"
$mvn -Phadoop-2-Pdist-DskipTests -Dmaven.javadoc.skip=true clean package
```

在编译过程中可能出现速度慢或者中断，可以再次启动编译，编译程序会在上次的编译中断处继续进行编译，整个编译过程耗时与网速紧密相关，网速较快的情况需要 1h 左右（图附录 C-1 的时间是重复多次下载依赖包，然后编译成功的界面），最终编译打包的文件为 \$HIVE\_HOME/packaging/target/apache-hive-1.2.1-bin.tar.gz。

```

[INFO]
[INFO] Hive ..... SUCCESS [ 7.980 s]
[INFO] Hive Shims Common ..... SUCCESS [ 8.234 s]
[INFO] Hive Shims 0.205 ..... SUCCESS [ 3.078 s]
[INFO] Hive Shims 0.23 ..... SUCCESS [13.257 s]
[INFO] Hive Shims Scheduler ..... SUCCESS [ 3.017 s]
[INFO] Hive Shims ..... SUCCESS [ 2.045 s]
[INFO] Hive Common ..... SUCCESS [ 9.857 s]
[INFO] Hive Serde ..... SUCCESS [11.644 s]
[INFO] Hive Metastore ..... SUCCESS [29.653 s]
[INFO] Hive Ant Utilities ..... SUCCESS [ 1.588 s]
[INFO] Spark Remote Client ..... SUCCESS [19.378 s]
[INFO] Hive Query Language ..... SUCCESS [01:50 min]
[INFO] Hive Service ..... SUCCESS [ 7.670 s]
[INFO] Hive Accumulo Handler ..... SUCCESS [ 8.445 s]
[INFO] Hive JDBC ..... SUCCESS [13.503 s]
[INFO] Hive Beeline ..... SUCCESS [ 3.762 s]
[INFO] Hive CLI ..... SUCCESS [ 4.056 s]
[INFO] Hive Contrib ..... SUCCESS [ 3.009 s]
[INFO] Hive HBase Handler ..... SUCCESS [ 8.032 s]
[INFO] Hive HCatalog ..... SUCCESS [ 1.329 s]
[INFO] Hive HCatalog Core ..... SUCCESS [ 5.503 s]
[INFO] Hive HCatalog Pig Adapter ..... SUCCESS [ 3.829 s]
[INFO] Hive HCatalog Server Extensions ..... SUCCESS [ 3.493 s]
[INFO] Hive HCatalog Webhcat Java Client ..... SUCCESS [ 3.095 s]
[INFO] Hive HCatalog Webhcat ..... SUCCESS [16.216 s]
[INFO] Hive HCatalog Streaming ..... SUCCESS [ 2.867 s]
[INFO] Hive HWI ..... SUCCESS [ 2.411 s]
[INFO] Hive ODBC ..... SUCCESS [ 1.311 s]
[INFO] Hive Shims Aggregator ..... SUCCESS [ 0.185 s]
[INFO] Hive TestUtils ..... SUCCESS [ 0.605 s]
[INFO] Hive Packaging ..... SUCCESS [ 3.106 s]
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 05:16 min
[INFO] Finished at: 2016-05-03T14:44:20+08:00
[INFO] Final Memory: 183M/823M
[INFO]

```

图 附录 C-1 编译 Hive 成功界面

通过如下命令查看最终编译完成整个目录大小，可以看到大小为 350MB 左右：

```
$du -s /app/compile/hive-1.2.1-src
```

## C.2 安装 Hive

### C.2.1 解压缩并迁移

使用上一步骤编译好的 Hive 编译包移动到安装目录上，用如下命令解压缩 hive 安装文件：

```

$cd /app/compile/hive-1.2.1-src/packaging/target/
$mv apache-hive-1.2.1-bin.tar.gz /home/spark/work/
$cd /home/spark/work/
$tar -zxvf hive-1.2.1-bin.tar.gz

```

改名并迁移到/app/soft 目录下：

```
$cd /app/spark
$mv apache-hive-1.2.1-bin /app/spark/hive-1.2.1
$ll /app/soft
```

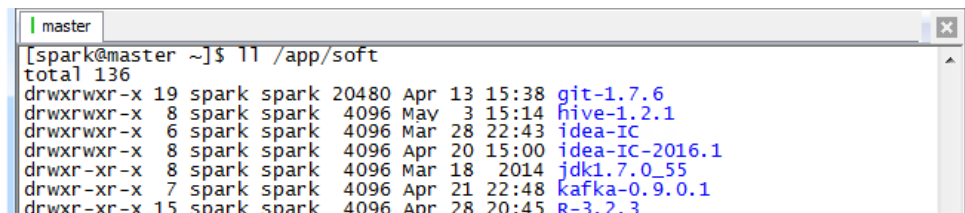


图 附录 C-2 Hive 移动到/app/soft 目录下

## C.2.2 下载 MySQL 驱动并放到 Hive 的 lib 目录下

登录 MySQL 官网进入下载页面 <http://dev.mysql.com/downloads/connector/j/>，默认情况下是 Windows 安装包，这里需要选择 Platform Independent 版本下载 zip 格式的文件，如图附录 C-3 所示。



图 附录 C-3 MySQL 驱动下载界面

把下载的 hive 安装包和 mysql 驱动包，使用如下命令放到 Hive 的 lib 目录下：

```
$cd /home/spark/work
$mv mysql-connector-java-5.1.34-bin.jar /app/soft/hive-1.2.1/lib
```

## C.2.3 配置/etc/profile 环境变量

使用如下命令打开/etc/profile 文件，设置如下参数：

```
export HIVE_HOME=/app/soft/hive-1.2.1
export PATH=$PATH:$HIVE_HOME/bin
export CLASSPATH=$CLASSPATH:$HIVE_HOME/bin
```

配置完毕后, 需要编译该配置文件或重新登录以生效该配置:

```
$source /etc/profile
```

## C.2.4 设置 hive-env.sh 配置文件

进入 hive-1.2.1/conf 目录, 复制 hive-env.sh.template 为 hive-env.sh 并进行配置:

```
$cd /app/soft/hive-1.2.1/conf
$cp hive-env.sh.template hive-env.sh
$sudo vi hive-env.sh
```

分别设置 HADOOP\_HOME 和 HIVE\_CONF\_DIR 两个值:

```
# Set HADOOP_HOME to point to a specific hadoop install directory
export HADOOP_HOME=/app/spark/hadoop-2.7.2
# Hive Configuration Directory can be controlled by:
export HIVE_CONF_DIR=/app/soft/hive-1.2.1/conf
```

## C.2.5 设置 hive-site.xml 配置文件

创建 hive-site.xml 配置文件, 在该配置文件中加入以下配置内容:

```
$touch hive-site.xml
$sudo vi hive-site.xml
```

hive 默认为 derby 数据库, derby 数据只运行单个用户进行连接, 这里需要调整为 Mysql 数据库, 以下为修改配置内容:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>javax.jdo.option.ConnectionURL</name>
    <value>jdbc:mysql://master:3306/hive?createDatabaseIfNotExists=true;characterEncoding=UTF-8</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionDriverName</name>
    <value>com.mysql.jdbc.Driver</value>
  </property>
  <property>
    <name>javax.jdo.option.ConnectionUserName</name>
    <value>hive</value>
```

```

</property>
<property>
  <name>javax.jdo.option.ConnectionPassword</name>
  <value>hive</value>
</property>
<property>
  <name>datanucleus.readOnlyDatastore</name>
  <value>>false</value>
</property>
<property>
  <name>datanucleus.fixedDatastore</name>
  <value>>false</value>
</property>
<property>
  <name>datanucleus.autoCreateSchema</name>
  <value>>true</value>
</property>
<property>
  <name>datanucleus.autoCreateTables</name>
  <value>>true</value>
</property>
<property>
  <name>datanucleus.autoCreateColumns</name>
  <value>>true</value>
</property>
</configuration>

```

## C.3 启动 Hive 并验证

### C.3.1 启动 Hive

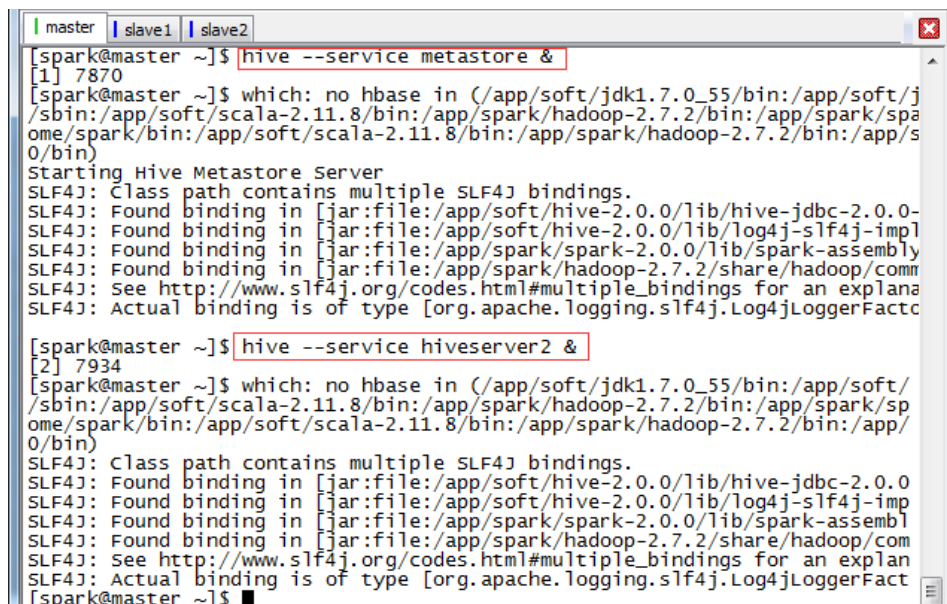
实际使用时，一般通过后台启动 metastore 和 hiveserver 实现服务，命令如下：

```

$hive --service metastore &
$hive --service hiveserver2&

```

启动后用通过 jps 命令可以看到两个服务在后台运行，如图附录 C-4 所示。



```

[spark@master ~]$ hive --service metastore &
[1] 7870
[spark@master ~]$ which: no hbase in (/app/soft/jdk1.7.0_55/bin:/app/soft/j
/bin:/app/soft/scala-2.11.8/bin:/app/spark/hadoop-2.7.2/bin:/app/spark/spa
ome/spark/bin:/app/soft/scala-2.11.8/bin:/app/spark/hadoop-2.7.2/bin:/app/s
0/bin)
Starting Hive Metastore Server
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/app/soft/hive-2.0.0/lib/hive-jdbc-2.0.0-
SLF4J: Found binding in [jar:file:/app/soft/hive-2.0.0/lib/log4j-slf4j-impl
SLF4J: Found binding in [jar:file:/app/spark/spark-2.0.0/lib/spark-assembly
SLF4J: Found binding in [jar:file:/app/spark/hadoop-2.7.2/share/hadoop/comm
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explana
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFacto

[spark@master ~]$ hive --service hiveserver2 &
[2] 7934
[spark@master ~]$ which: no hbase in (/app/soft/jdk1.7.0_55/bin:/app/soft/
/bin:/app/soft/scala-2.11.8/bin:/app/spark/hadoop-2.7.2/bin:/app/spark/sp
ome/spark/bin:/app/soft/scala-2.11.8/bin:/app/spark/hadoop-2.7.2/bin:/app/
0/bin)
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/app/soft/hive-2.0.0/lib/hive-jdbc-2.0.0
SLF4J: Found binding in [jar:file:/app/soft/hive-2.0.0/lib/log4j-slf4j-imp
SLF4J: Found binding in [jar:file:/app/spark/spark-2.0.0/lib/spark-assembl
SLF4J: Found binding in [jar:file:/app/spark/hadoop-2.7.2/share/hadoop/com
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explan
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFact
[spark@master ~]$ █

```

图 附录 C-4 Hive 启动后台服务

### C.3.2 验证安装

登录 hive, 在 hive 创建表并查看该表, 如图附录 C-5 所示, 命令如下:

```

$hive
hive> create table test(a string, b int);
hive> show tables;
hive> desc test;

```

登录 Mysql, 在 TBLS 表中查看新增 test 表:

```

$mysql -uhive -phive
mysql> use hive;
mysql> select TBL_ID, CREATE_TIME, DB_ID, OWNER, TBL_NAME,TBL_TYPE from TBLS;

```

图附录 C-6 为在 Hive 元数据表查询到创建表。

```

[spark@master spark-2.0.0]$ hive
which: no hbase in (/app/soft/jdk1.7.0_55/bin:/app/soft/jdk1.7.0_55/bin:/u
ala-2.11.8/bin:/app/spark/hadoop-2.7.2/bin:/app/spark/spark-2.0.0/bin:/app
/soft/scala-2.11.8/bin:/app/spark/hadoop-2.7.2/bin:/app/spark/spark-2.0.0/
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/app/soft/hive-2.0.0/lib/hive-jdbc-2.0.0
SLF4J: Found binding in [jar:file:/app/soft/hive-2.0.0/lib/log4j-slf4j-imp
SLF4J: Found binding in [jar:file:/app/spark/spark-2.0.0/lib/spark-assembly
SLF4J: Found binding in [jar:file:/app/spark/hadoop-2.7.2/share/hadoop/com
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explan
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFact

Logging initialized using configuration in jar:file:/app/soft/hive-2.0.0/l
Hive-on-MR is deprecated in Hive 2 and may not be available in the future
eases.
hive> create table test(a string, b int);
OK
Time taken: 3.782 seconds
hive> show tables;
OK
test
Time taken: 0.326 seconds, Fetched: 1 row(s)
hive> desc test;
OK
a                string
b                int
Time taken: 0.262 seconds, Fetched: 2 row(s)
hive> █

```

图 附录 C-5 Hive 中创建测试表

```

[spark@master ~]$ mysql -uhive -phive
warning: Using a password on the command line interface can be insecure.
welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 96
Server version: 5.6.21 MySQL Community Server (GPL)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserve
d.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statem
ent.

mysql> use hive;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select TBL_ID, CREATE_TIME, DB_ID, OWNER, TBL_NAME, TBL_TYPE from TB
LS;
+-----+-----+-----+-----+-----+-----+
| TBL_ID | CREATE_TIME | DB_ID | OWNER | TBL_NAME | TBL_TYPE |
+-----+-----+-----+-----+-----+-----+
|      1 | 1460557944 |      1 | spark | test     | MANAGED_TABLE |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> █

```

图 附录 C-6 在 Hive 元数据表查询到创建表

## C.4 Hive 实例演示

### C.4.1 准备数据

#### 1. 上传数据

交易数据存放在该系列配套资源的/saledata 目录下，把这些数据文件上传到 master 节点的 /home/spark/word 目录下。

#### 2. 启动 Hive 并创建数据库

启动 HDFS、YARN 和 Hive，启动完毕后创建 Hive 数据库：

```
$hive --service metastore &
$hive
hive> create database hive;
hive> show databases;
hive> use hive;
```

#### 3. 在 Hive 创建表

启动 Hadoop 集群，进入 Hive 命令行操作界面，使用如下命令创建 3 张数据表。

- **tbDate**: 定义了日期的分类，将每天分别赋予所属的月份、星期、季度等属性，字段分别为日期、年月、年、月、日、周几、第几周、季度、旬、半月。
- **tbStock**: 定义了订单表头，字段分别为订单号、交易位置、交易日期。
- **tbStockDetail**: 定义了订单明细，该表和 tbStock 以交易号进行关联，字段分别为订单号、行号、货品、数量、金额。

```
hive> CREATE TABLE tbDate(dateID string,theyearmonth string,theyear string,
    themonth string,thedata string,theweek string,theweeks string,thequot string,
    thetenday string,thehalfmonth string) ROW FORMAT DELIMITED FIELDS TERMINATED
    BY ',' LINES TERMINATED BY '\n' ;
```

Time taken: 1.121 seconds

```
hive> CREATE TABLE tbStock(ordernumber STRING,locationid string,dateID string)
    ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' ;
```

Time taken: 0.166 seconds

```
hive> CREATE TABLE tbStockDetail(ordernumber STRING,rownum int,itemid string,qty
    int,price int ,amount int) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES
    TERMINATED BY '\n' ;
```

Time taken: 0.267 seconds



```
hive> show tables;
OK
tbdate
tbstock
tbstockdetail
Time taken: 0.089 seconds, Fetched: 3 row(s)
```

#### 4. 导入数据

从本地操作系统分别加载日期、交易信息和交易详细信息表数据：

```
hive> LOAD DATA LOCAL INPATH '/home/spark/work/saledata/tbDate.txt' INTO TABLE
    tbDate;
Loading data to table hive.tbdate
OK
Time taken: 2.784 seconds
```

```
hive> LOAD DATA LOCAL INPATH '/home/spark/work/saledata/tbStock.txt' INTO TABLE
    tbStock;
Loading data to table hive.tbstock
OK
Time taken: 0.648 seconds
```

```
hive> LOAD DATA LOCAL INPATH '/home/spark/work/saledata/tbStockDetail.txt' INTO
    TABLE tbStockDetail;
Loading data to table hive.tbstockdetail
OK
Time taken: 1.44 seconds
```

查看 HDFS 中相关 SALEDATA 数据库中增加了 3 个文件夹，分别对应 3 张表：

```
[spark@master ~]$ hadoop fs -ls /user/hive/warehouse/hive.db
Found 3 items
drwxr-xr-x- spark 2016-04-14 15:18 /user/hive/warehouse/hive.db/ tbdate
drwxr-xr-x- spark 2016-04-14 15:18 /user/hive/warehouse/hive.db/ tbstock
drwxr-xr-x- spark 2016-04-14 15:18 /user/hive/warehouse/hive.db/ tbstockdetail
```

## C.4.2 计算所有订单每年的总金额

### 1. 算法分析

要计算所有订单每年的总金额，首先需要获取所有订单的订单号、订单日期和订单金信息，然后把这些信息和日期表进行关联，获取年份信息，最后根据这 4 个列按年份归组统计获取所有订单每年的总金额。

2. 执行 HSQL 语句

```
hive> use hive;
hive> select c.theyear, sum(b.amount) from tbStock a,tbStockDetail b,tbDate c
      where a.ordernumber=b.ordernumber and a.dateid=c.dateid group by c.theyear
      order by c.theyear;
```

运行过程中创建两个 Job，分别为 application\_1460617800545\_0001 和 application\_1460617800545\_000，在 YARN 的资源管理器界面中（默认 <http://master:8088/>），可以看到如图附录 C-7 所示的界面。

3. 查看结果

整个计算过程使用了 175.25s，结果如图附录 C-8 所示。

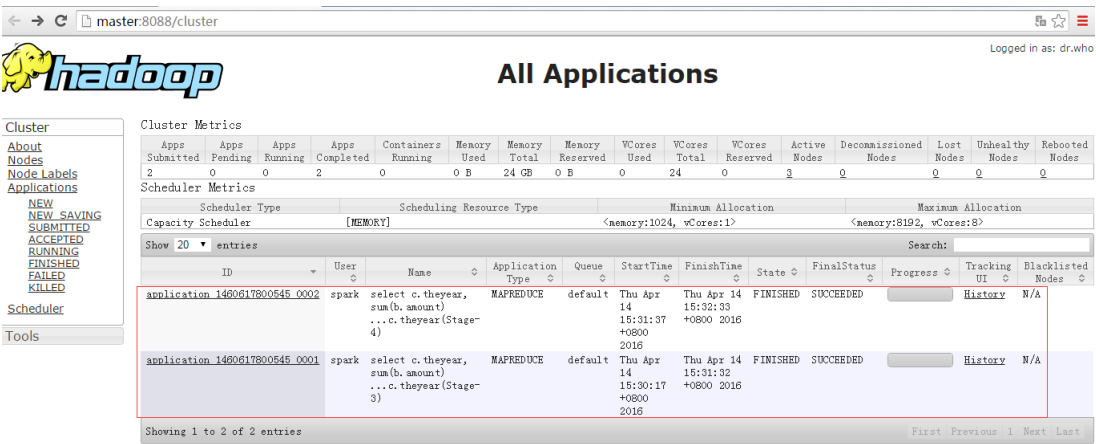


图 附录 C-7 在 YARN 监控界面作业运行状态

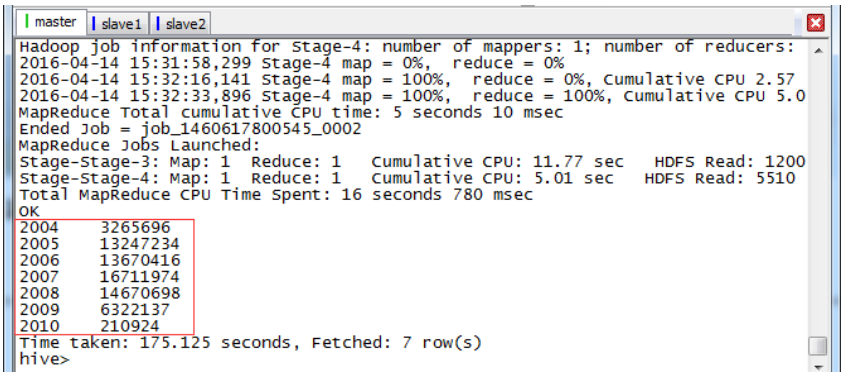


图 附录 C-8 计算所有订单每年的总金额结果

## C.4.3 计算所有订单每年最大金额订单的销售额

### 1. 算法分析

该算法分为两步：

- (1) 按照日期和订单号进行归组计算，获取所有订单每天的销售数据。
- (2) 把第一步获取的数据和日期表进行关联获取的年份信息，然后按照年份进行归组，使用 Max 函数，获取所有订单每年最大金额订单的销售额。

### 2. 执行 HSQL 语句

// 第一步：按照日期和订单号进行归组计算，获取所有订单每天的销售数据

```
hive> select a.dateid,a.ordernumber,sum(b.amount) as sumofamount from tbStock
a, tbStockDetail b where a.ordernumber=b.ordernumber group by a.dateid,
a.ordernumber;
```

// 第二步：按照年份进行归组，使用 Max 函数，获取所有订单每年最大金额订单的销售额

```
hive> select c.theyear,max(d.sumofamount) from tbDate c,(select a.dateid,
a.ordernumber,sum(b.amount) as sumofamount from tbStock a,tbStockDetail b
where a.ordernumber=b.ordernumber group by a.dateid,a.ordernumber) d where
c.dateid=d.dateid group by c.theyear sort by c.theyear;
```

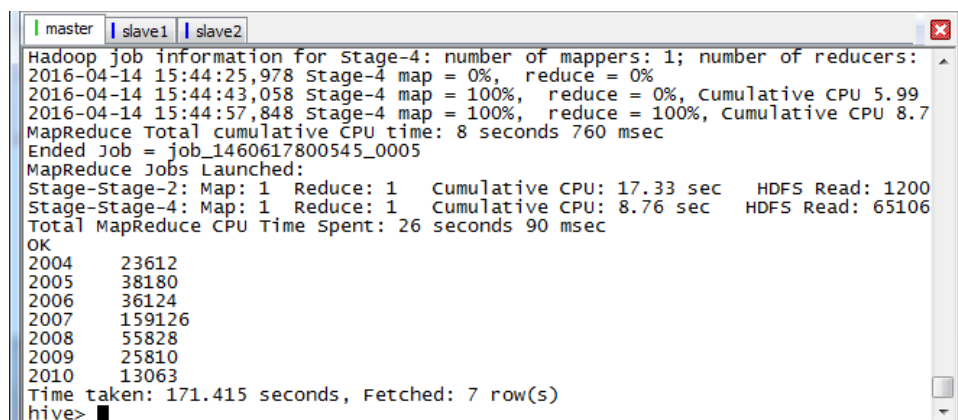
运行过程中创建两个 Job，分别为 job\_1437659442092\_0004 和 job\_1437659442092\_0005，在 YARN 的监控界面中可以看到如图附录 C-9 所示的界面。

Cluster Metrics															
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
5	0	0	5	0	0 B	24 GB	0 B	0	24	0	3	0	0	0	0
Scheduler Metrics															
Scheduler Type		Scheduling Resource Type				Minimum Allocation				Maximum Allocation					
Capacity Scheduler		[MEMORY]				<memory:1024, vCores:1>				<memory:8192, vCores:8>					
Show 20 ▾ entries												Search: <input type="text"/>			
ID ▾	User ▾	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes				
application_1460617800545_0005	spark	select c.theyear,max(d.sumofamou...c.theyear(Stage-4)	MAPREDUCE	default	Thu Apr 14 15:44:08 +0800 2016	Thu Apr 14 15:44:57 +0800 2016	FINISHED	SUCCEEDED	<div></div>	History	N/A				
application_1460617800545_0004	spark	select c.theyear,max(d.sumofamou...c.theyear(Stage-2)	MAPREDUCE	default	Thu Apr 14 15:42:37 +0800 2016	Thu Apr 14 15:43:35 +0800 2016	FINISHED	SUCCEEDED	<div></div>	History	N/A				

图 附录 C-9 在 YARN 监控界面作业运行状态

### 3. 查看结果

整个计算过程使用了 171.41s，结果如图附录 C-10 所示。



```

master  slave1  slave2
Hadoop job information for stage-4: number of mappers: 1; number of reducers:
2016-04-14 15:44:25,978 Stage-4 map = 0%, reduce = 0%
2016-04-14 15:44:43,058 Stage-4 map = 100%, reduce = 0%, Cumulative CPU 5.99
2016-04-14 15:44:57,848 Stage-4 map = 100%, reduce = 100%, Cumulative CPU 8.7
MapReduce Total cumulative CPU time: 8 seconds 760 msec
Ended Job = job_1460617800545_0005
MapReduce Jobs Launched:
Stage-Stage-2: Map: 1 Reduce: 1 Cumulative CPU: 17.33 sec HDFS Read: 1200
Stage-Stage-4: Map: 1 Reduce: 1 Cumulative CPU: 8.76 sec HDFS Read: 65106
Total MapReduce CPU Time Spent: 26 seconds 90 msec
OK
2004      23612
2005      38180
2006      36124
2007      159126
2008      55828
2009      25810
2010      13063
Time taken: 171.415 seconds, Fetched: 7 row(s)
hive>

```

图 附录 C-10 查看所有订单每年最大金额订单的销售结果

## 附录 D

# 安装 ZooKeeper

## D.1 安装 ZooKeeper

### D.1.1 下载 ZooKeeper

ZooKeeper 是 Apache 基金会的一个开源、分布式应用程序协调服务，是 Google 的 Chubby 一个开源的实现。它是一个为分布式应用提供一致性服务的软件，提供的功能包括配置维护、域名服务、分布式同步、组服务等。它的目标就是封装好复杂易出错的关键服务，将简单易用的接口和性能高效、功能稳定的系统提供给用户。

ZooKeeper 安装包可以在其官网的下载页面下载，下载地址为 <http://zookeeper.apache.org/releases.html#download>。为加快下载速度可以选择中国境内的镜像，选择稳定版本 `zookeeper-3.4.8.tar.gz` 安装包。

### D.1.2 解压并配置环境变量

下载后把安装包方放在目录 `/home/spark/work` 目录下，用下面的命令解压缩 ZooKeeper 安装包，并把解压后的目录移动到 `/app/soft` 目录下：

```
$cd /home/spark/work/  
$tar -zxvf zookeeper-3.4.8.tar.gz  
$mv zookeeper-3.4.8 /app/soft  
$ll /app/soft
```

为了方便运行 `zkServer.sh` 脚本，在集群中的节点中，需要将 ZooKeeper 的 `bin` 路径加入到 `/etc/profile` 中，设置如下内容（分发到各节点后，在各节点上做同样设置）：

```
export ZOOKEEPER_HOME=/app/soft/zookeeper-3.4.8
```

```
export PATH=$PATH:$ZOOKEEPER_HOME/bin
```

设置完毕后使用如下命令使配置生效:

```
$source /etc/profile
```

### D.1.3 修改 ZooKeeper 的配置文件

在 ZooKeeper 的根目录下建立 data 和 log 目录用于存放工作数据和日志文件:

```
$mkdir /app/soft/zookeeper-3.4.8/data/
```

```
$mkdir /app/soft/zookeeper-3.4.8/log/
```

在 ZooKeeper 配置目录下默认情况下, 不存在 zoo.cfg 文件, 需要复制一份, 然后进行修改, 命令如下:

```
$cd /app/soft/zookeeper-3.4.8/conf/
```

```
$cp zoo_sample.cfg zoo.cfg
```

```
$sudo vi zoo.cfg
```

修改 zoo.cfg 配置文件内容 (仅列出重要配置):

#用于存放 ZooKeeper 的数据和日志

```
dataDir=/app/soft/zookeeper-3.4.8/data
```

```
dataLogDir=/app/soft/zookeeper-3.4.8/log
```

//外部客户端连接端口号, 在 Kafka 中将使用该端口号

```
clientPort=2181
```

//ZooKeeper 集群相关配置信息

```
server.1=master:2888:3888
```

```
server.2=slave1:2888:3888
```

```
server.3=slave2:2888:3888
```

配置中 server.A=B: C: D 含义如下:

- A 为数字, 表示这个第几号服务器。
- B 表示该服务器的 IP 地址。
- C 表示该服务器与集群中的 Leader 服务器交换信息的端口。
- D 表示的是万一集群中的 Leader 服务器挂了, 需要一个端口来重新进行选举, 选出一个新的 Leader, 而这个端口就是用来执行选举时服务器相互通信的端口。

如果是伪集群的配置方式, 由于 B 都是一样, 所以不同的 ZooKeeper 实例通信端口号不能一样, 所以要给它们分配不同的端口号。

## D.1.4 分发 ZooKeeper 到各节点

使用 `scp` 命令到 ZooKeeper 分发到 `slave1` 和 `slave2` 节点上:

```
$cd /app/soft/
$scp -r zookeeper-3.4.8 spark@slave1:/app/soft
$scp -r zookeeper-3.4.8 spark@slave2:/app/soft
```

在 `dataDir` 目录下创建一个 `myid` 文件,然后分别在 `myid` 文件中按照 `zoo.cfg` 文件的 `server.A` 中 `A` 的数值,在不同机器上的该文件中填写相应的值,如 `master` 节点该值为 1、`slave1` 节点该值为 2、`slave2` 节点该值为 3。

```
$cd /app/soft/zookeeper-3.4.8/data
$vi myid
```

## D.2 启动并验证

执行命令“`zkServer.sh start`”将会启动 ZooKeeper。在此大家需要注意的是,不同节点上的 ZooKeeper 需要单独启动。而执行命令“`zkServer.sh stop`”将会停止 ZooKeeper。用户可以使用命令“`JPS`”查看 ZooKeeper 是否成功启动,或执行命令“`zkServer.sh status`”查看 ZooKeeper 集群状态:

```
$zkServer.sh start
$zkServer.sh status
```

图附录 D-1 为启动 Zookeeper 并查看状态的显示页面。

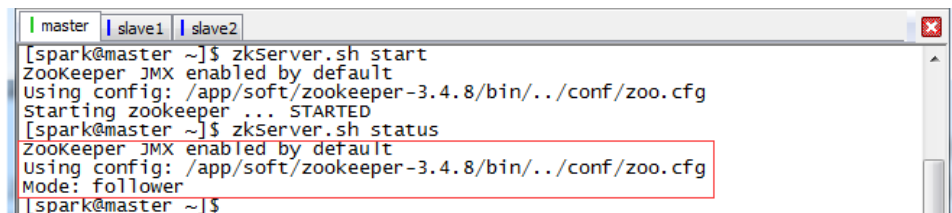


图 附录 D-1 启动 Zookeeper 并查看状态

当第一个节点启动 ZooKeeper 时由于集群的其他节点未启动 ZooKeeper,使用 `zkServer.sh status` 命令查看当前状态时会提示错误。但是随着后续节点的 ZooKeeper 的陆续启动,使用 `status` 查看状态时会显示当前节点的状态,本次 `master` 作为了 `follower`。

## 附录 E

# 安装 Kafka

## E.1 安装 Kafka

### E.1.1 下载 Kafka

Kafka 是由 LinkedIn 设计的一个高吞吐量、分布式、基于发布订阅模式的消息系统，使用 Scala 编写，它以可水平扩展、可靠性、异步通信和高吞吐率等特性而被广泛使用。目前越来越多的开源分布式处理系统都支持与 Kafka 集成，其中 Spark Streaming 作为后端流引擎配合 Kafka 作为前端消息系统正成为当前流处理系统的主流架构之一。

Kafka 安装包可以在其官网下载页面下载，下载地址为 <http://kafka.apache.org/downloads.html>。为加快下载速度可以选择中国境内的镜像，选择稳定版本 `kafka_2.11-0.9.0.1.tgz` 安装包。

### E.1.2 解压并配置环境变量

下载后把安装包方放在目录 `/home/spark/work` 目录下，用下面命令解压缩 Kafka 安装包，并把解压后的目录移动到 `/app/soft` 目录下：

```
$cd /home/spark/work/  
$tar -zxvf kafka_2.11-0.9.0.1.tgz  
$mv kafka-0.9.0.1 /app/soft  
$ll /app/soft
```

为了方便运行 Kafka 相关脚本，将 Kafka 的 bin 路径加入到 `/etc/profile` 中，设置如下内容（分发到各节点后，在各节点上做同样设置）：

```
export KAFKA_HOME=/app/soft/kafka-0.9.0.1  
export PATH=$PATH:$KAFKA_HOME/bin
```



设置完毕后使用如下命令使配置生效：

```
$source /etc/profile
```

### E.1.3 修改 Kafka 的配置文件

在 Kafka 的根目录下建立 log 目录用于存放日志文件：

```
$mkdir /app/soft/kafka-0.9.0.1/logs/
```

修改\$KAFKA\_HOME/config/server.properties 配置文件内容（仅列出重要配置）：

```
##### Server Basics #####
#建议根据 IP 区分,这里使用 ZooKeeper 中的 ID 来设置,如 master 节点设置为 0,slave1 节点设置
为 1, slave2 节点设置为 2
broker.id=1

##### Socket Server Settings #####
#broker 用于接收 producer 消息的端口
port=9092

#broker 的 hostname
host.name=master

#配置 PRODUCER/CONSUMER 连上来的时候使用的地址
advertised.host.name=master

##### Log Basics #####
#kafka 存放消息文件的路径
log.dirs=/app/soft/kafka-0.9.0.1/logs/

#topic 的默认分区数
num.partitions=2

##### ZooKeeper #####
#ZooKeeper 集群连接地址信息
zookeeper.connect=master:2181,slave1:2181,slave2:2181

#连接 ZooKeeper 超时时间,单位为毫秒
zookeeper.connection.timeout.ms=6000
```

### E.1.4 分发 Kafka 到各节点

使用 scp 命令到 Kafka 分发到 slave1 和 slave2 节点上：

```
$cd /app/soft/kafka-0.9.0.1
$scp -r kafka-0.9.0.1 spark@slave1:/app/soft
$scp -r kafka-0.9.0.1 spark@slave2:/app/soft
```

分发完毕后, 修改 server.properties 配置文件中 broker.id、host.name、advertised.host.name 和 zookeeper.connect 等配置项。

## E.2 启动并验证

### 1. 启动 ZooKeeper

分别在 master、slave1 和 slave2 节点上启动 ZooKeeper 服务:

```
$zkServer.sh start
```

### 2. 启动 Kafka

分别在 master、slave1 和 slave2 节点上启动 Kafka 服务, 如图附录 E-1 所示:

```
$kafka-server-start.sh $KAFKA_HOME/config/server.properties
```

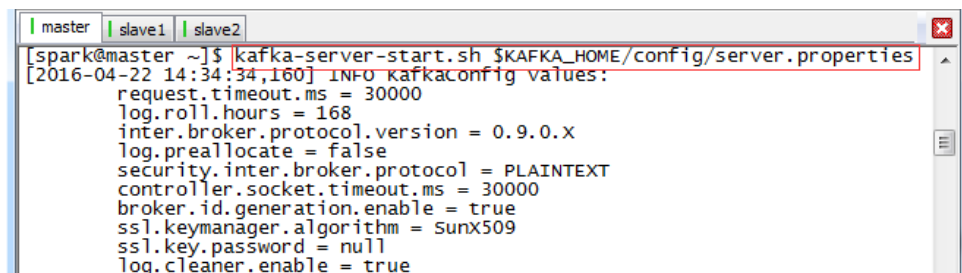


图 附录 E-1 启动 Kafka

### 3. 在 master 节点上新建主题 Topic

在 Kafka 中创建主题, 如图附录 E-2 所示。

```
$kafka-topics.sh --create --topic kafkaTopic --replication-factor 3 --partitions
2 --zookeeper master:2181
```

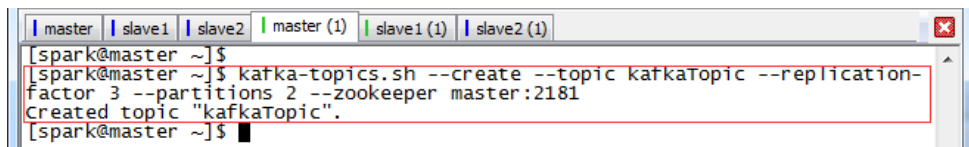


图 附录 E-2 在 Kafka 中创建主题

#### 4. 在 slave1 模拟消息生产者，发送消息至 Kafka

```
$ kafka-console-producer.sh --broker-list master:9092 --sync --topic kafkaTopic
```

当消费者连接后，在发送消息的终端输入：hello kafka/who are you?，如图附录 E-3 所示。

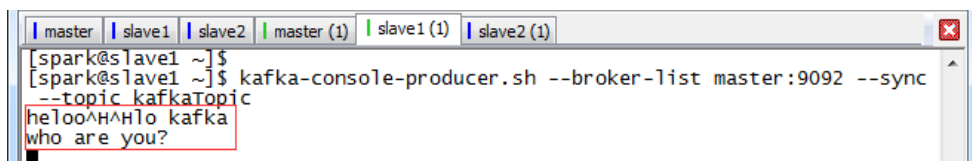


图 附录 E-3 slave1 模拟消息生产者发送消息

#### 5. 在 slave2 模拟消息消费者，显示消息的消费

```
$ kafka-console-consumer.sh --zookeeper master:2181 --topic kafkaTopic  
--from-beginning
```

由于设置接收从开始到现在的消息，以前发送的消息也显示在 slave2 终端上，如图附录 E-4 所示。

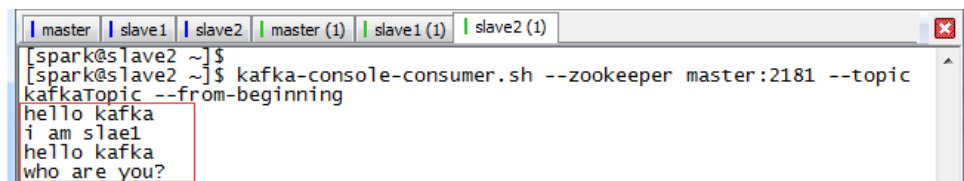


图 附录 E-4 slave2 模拟消息消费者收到消息