OUTAC

分布式开发基础教程

Document Number:	<Feature Identifier>
Document Status:	Draft
Document Issue Number:	0.9.0
Issue Date:	2013-12-20
Security Status:	OUTAC.CN Confidential
Author:	杨立峰（311155@qq.com）

Intended left spaces

Table of Contents

Table of Tables

Table of Figures

IDENTIFICATION OF DOCUMENT

Storage Location

https://www.outac.cn/documents/86b3edeacec54522a603a8c828c0af97/uploads/4abb8d4883f04a3abff11759b3e65ed8/

List of Contributors

Identify all of the key contributors to this document.

Name	Role
杨立峰（311155@qq.com）	Software Platform Architect
	Base Software Designer
	Base Software Designer
	Base Software Designer
	Base Software Platform SDA

Approver(s)

Identify the person or people who must approve this document.

Name	Role	Area of Responsibility
杨立峰	System Design Authority	Converged Platform

Publication History

Identify the changes that are made to the document. Please update the Issue Number appropriately. The Issue Number must conform to the Document and Record Control Process. A generic Publication History table with example text is provided below.

Issue	Change Summary	Author(s)	Date
Preliminary	Initial creation	杨立峰	2015-03-08
Draft 0.50	Done document architecture	杨立峰	2015-05-17
Draft 0.90	Changes based on internal informal reviews	杨立峰	2015-12-20
1.00	Changes from formal review		NA

Intended left spaces

1 服务器基础环境搭建

本文以 Poplar Master 服务器基础环境配置为例分别演示用户配置、sudo 权限配置、网路配置、安装 JDK 工具等。用户需参照以下步骤完成 POPSlave1~POPSlave3 服务器的基础环境配置。

1.1 安装环境

硬件环境：Ubuntu 14.04 64bit 服务器 4 台（一台为 Master 节点，三台为 Slave 节点）

软件环境：Java 1.8.0_66、hadoop-2.7.1、zookeeper-3.4.7、hbase-1.1.2

1.2 用户配置

1) 添加一个用户

```
[hadoop@Poplar:~]$ adduser hadoop            #新建 hadoop 用户
[hadoop@Poplar:~]$ passwd hadoop             #hadoop 用户设置密码
```

2) 建工作组

```
[hadoop@Poplar:~]$ groupadd hadoop           #新建 hadoop 工作组
```

3) 给已有的用户增加工作组

```
[hadoop@Poplar:~]$ usermod -G hadoop hadoop
```

4) 给 hadoop 用户创建目录

```
[hadoop@Poplar:~]$ sudo mkdir /home/hadoop  #如果已有不用添加
[hadoop@Poplar:~]$ sudo chown hadoop /home/hadoop
```

1.3 sudo 权限配置

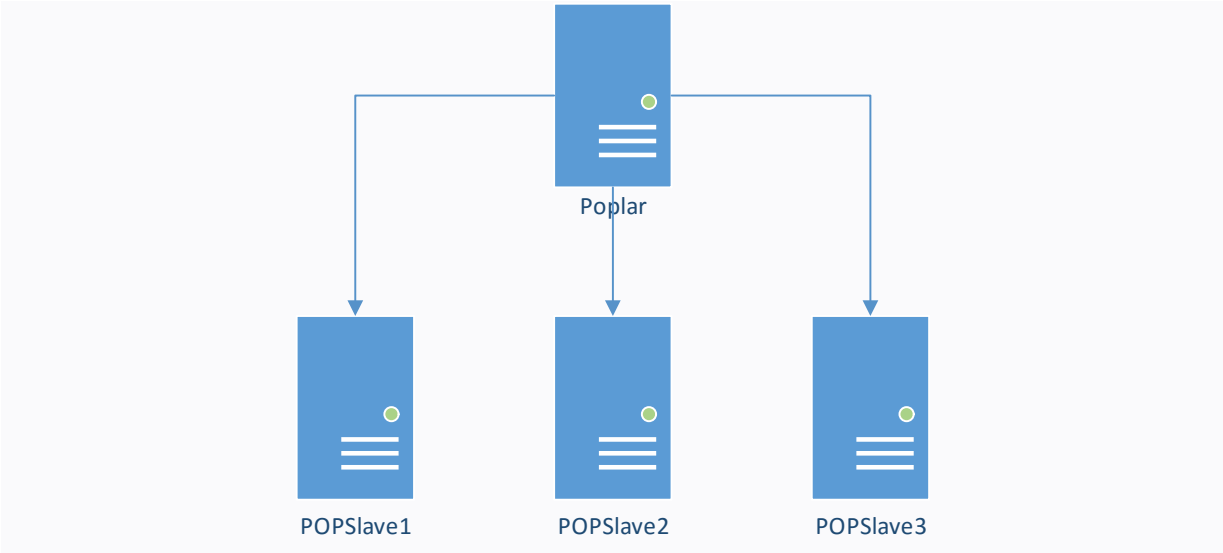
可考虑为 hadoop 用户增加管理员权限，方便部署，避免一些权限不足的问题：

```
[hadoop@Poplar:~]$ sudo adduser hadoop sudo
```

1.4 网络配置

我们使用 4 台机器来搭建 Hadoop 完全分布式环境，也可以采用四台 VMWare 虚拟机模拟，但是这样需要运行虚拟机的电脑比较好的处理能力。4 台机器的拓扑图如下图所示：

Figure 1-1 网络拓扑图



Hadoop 集群中每个节点的角色如下表所示：

Table 1-1 节点角色配置表

主机名	角色	IP 地址	jps 命令结果	Hadoop 用户属组	安装目录
Poplar	namenode	192.168.42.121	NameNode ResourceManager SecondaryNameNode	hadoop:hadoop	/opt/hadoop
POPSlave1	datanode	192.168.42.122	DataNode NodeManage		
POPSlave2	datanode	192.168.42.123	DataNode NodeManager		
POPSlave3	datanode	192.168.42.124	DataNode NodeManager		

1) 配置 IP 地址

修改网口 IP 地址配置：

```
[hadoop@Poplar:~]$ sudo vi /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
address 192.168.42.121
gateway 192.168.42.1
```

```
netmask 255.255.255.0
network 192.168.42.0
broadcast 192.168.42.255
```

修改 hosts 配置方式如下：

```
[hadoop@Poplar:~]$ sudo vi /etc/hosts
127.0.0.1               localhost
192.168.42.121          Poplar
192.168.42.122          POPSlave1
192.168.42.123          POPSlave2
192.168.42.124          POPSlave3
```

完成后，如下图所示(/etc/hosts 中只能有一个 127.0.0.1，对应为 localhost，否则会出错)。

2) 修改 Host 主机名

```
[hadoop@Poplar:~]$ sudo vi /etc/hostname
Poplar
```

3) 重启主机使得主机名生效

```
[hadoop@Poplar:~]$ sudo reboot
```

注意，该网络配置需要在所有主机上进行

重启一下，在终端中才会看到机器名的变化。配置好后可以在各个主机上执行 ping Master 和 ping Slave1 测试一下，看是否相互 ping 得通。

1.5 安装配置 SSH

1.5.1 配置 Poplar 本机无密码登录

最后注销当前用户，使用 hadoop 用户进行登陆。

Ubuntu 默认安装了 SSH client，还需要安装 SSH server。

```
[hadoop@Poplar:~]$ sudo apt-get install openssh-server
```

集群、单节点模式都需要用到 SSH 无密码登陆，首先设置 SSH 无密码登陆本机。输入命令：

```
[hadoop@Poplar:~]$ ssh localhost
```

会有如下提示(SSH 首次登陆提示)，输入 yes。

```
[hadoop@Poplar:~]$ ssh localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is 8d:fd:62:dc:75:0f:6b:e8:08:36:23:49:be:6d:5d:df.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
hadoop@localhost's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)
```

然后按提示输入密码 hadoop，这样就登陆到本机了。但这样的登陆是需要密码的，需要配置成无密码登陆。

先退出刚才的 ssh，然后生成 ssh 证书：

```
[hadoop@Poplar:~]$ exit              # 退出 ssh localhost
[hadoop@Poplar:~]$ cd ~/.ssh         # 如果没有该目录，先执行一次 ssh localhost
[hadoop@Poplar:~]$ ssh-keygen -t rsa # 一直按回车就可以
[hadoop@Poplar:~]$ cp id_rsa.pub authorized_keys
```

此时再用 ssh localhost 命令，就可以直接登陆了，如下所示。

```
hadoop@Poplar:~/.ssh$ ssh localhost
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Tue Dec 15 10:27:03 CST 2015

  System load:  0.89               Processes:           441
  Usage of /:   0.7% of 983.20GB   Users logged in:     1
  Memory usage: 46%                IP address for eth0: 192.168.42.121
  Swap usage:   0%

  Graph this data and manage this system at:
    https://landscape.canonical.com/

606 packages can be updated.
252 updates are security updates.

Last login: Tue Dec 15 10:27:05 2015 from localhost
```

1.5.2 配置 Poplar 本机无密码登录 POPSlave1~POPSlave3

完成后可以使用 ssh Poplar 验证一下。接着将公匙传输到 POPSlave1 节点：

```
scp ~/.ssh/id_rsa.pub hadoop@POPSlave1:/home/hadoop/
```

scp 时会要求输入 POPSlave1 上 hadoop 用户的密码(hadoop)，输入完成后会提示传输完毕。

接着在 POPSlave1 节点 上将 ssh 公匙保存到相应位置，执行：

```
cat ~/id_rsa.pub >> ~/.ssh/authorized_keys
```

其他 POPSlave 节点，一样进行操作，也要将公匙传输到其他的 POPSlave 节点，在其他 POPSlave 节点上加入授权这两步。

最后在 Master 节点上就可以无密码 SSH 到 POPSlave 节点了。

```
ssh POPSlave1
```

1.5.3 配置 POPSlave1~POPSlave3 本机无密码登录 Poplar

下面以 POPSlave1 无密码登录 Poplar 为例进行讲解，用户需参照下面步骤完成 POPSlave2~POPSlave3 无密码登录 Poplar。

1) 以 hadoop 用户远程登录 Poplar，复制 POPSlave1 服务器的公钥"id_rsa.pub"到 POPSlave1 服务器的"/home/hadoop/"目录下。

```
[hadoop@Poplar hadoop]$ scp hadoop@POPSlave1:/home/hadoop/.ssh/id_rsa.pub
/home/hadoop
```

2) 将 POPSlave1 的公钥（/home/hadoop/id_rsa.pub）追加到 Poplar 的 authorized_keys 中。

```
[hadoop@Poplar hadoop]$ cd /home/hadoop
[hadoop@Poplar hadoop]$ cat id_rsa.pub >> .ssh/authorized_keys
[hadoop@Poplar hadoop]$ rm -r /home/hadoop/id_rsa.pub
```

3) 以 hadoop 用户远程登录 POPSlave1 服务器，在 POPSlave1 服务器测试通过 SSH 无密码登录 Poplar。

```
[hadoop@POPSlave1 hadoop]$ ssh Poplar
```

1.6 安装 JDK 工具

建议安装 Oracle 的 JDK，不建议使用 OpenJDK，不过按 http://wiki.apache.org/hadoop/HadoopJavaVersions 中说的，新版本在 OpenJDK 1.7 下是没问题的。这里分别安装这两种 JDK。

- Oracle JDK 安装

将下载的 JDK 解压到/opt 下。需要配置一下 JAVA_HOME 环境变量，这个环境变量很多地方都会用到，在 /etc/environment 中配置：

```
[hadoop@Poplar:~]$ sudo vim /etc/environment
```

文件如下所示：

```
PATH="/opt/zookeeper/bin:/opt/hadoop/bin:/opt/hadoop/sbin:/opt/hbase/bin:/opt/go/bin:
/opt/jdk/bin/:/opt/liteide/bin/:/opt/vscode:/usr/local/sbin:/usr/local/bin:/usr/sbin:
/usr/bin:/sbin:/bin:/usr/games:/usr/local/games"

# Java
JDK_HOME=/opt/jdk
JAVA_HOME=/opt/jdk
JRE_HOME=/opt/jdk/jre
CLASSPATH=".:/opt/jdk/lib:/opt/jdk/jre/lib"

# HADOOP
HADOOP_HOME=/opt/hadoop
HADOOP_HOME_WARN_SUPPRESS=1

# ZooKeeper
ZOOKEEPER=/opt/zookeeper
ZOO_LOG_DIR=/home/hadoop/hadoop/zookeeper/log
```

也可以编辑/etc/profile 文件。在后面添加 Java 的 JRE_HOME、CLASSPATH 以及 PATH 内容。

```
[hadoop@Poplar ~]$ sudo vim /etc/profile
# JAVA
export JRE_HOME=$JAVA_HOME/jre
export CLASSPATH=.:$CLASSPATH:$JAVA_HOME/lib:$JRE_HOME/lib
export PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin
# HADOOP
export HADOOP_HOME=/opt/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
```

```
export HADOOP_HOME_WARN_SUPPRESS=1
```

- OpenJDK 7 安装

```
[hadoop@Poplar:~]$ sudo apt-get install openjdk-7-jre openjdk-7-jdk
```

默认的安装位置为: /usr/lib/jvm/java-7-openjdk-amd64 (可以通过命令 dpkg -L openjdk-7-jdk 看到)。安装完后就可以使用了，可以用 java -version 检查一下。

需要配置一下 JAVA_HOME 环境变量，这个环境变量很多地方都会用到，在 /etc/environment 中配置：

```
[hadoop@Poplar:~]$ sudo vim /etc/environment
```

在文件末尾添加一行：

```
JAVA_HOME="/usr/lib/jvm/java-7-openjdk-amd64"
```

编辑"/etc/profile"文件。在后面添加 Java 的" JRE_HOME"、"CLASSPATH"以及"PATH"内容。

```
[hadoop@Poplar ~]$ sudo vim /etc/profile
# JAVA
export JRE_HOME=$JAVA_HOME/jre
export CLASSPATH=.:$CLASSPATH:$JAVA_HOME/lib:$JRE_HOME/lib
export PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin
# HADOOP
export HADOOP_HOME=/usr/hadoop-1.2.1
export PATH=$PATH:$HADOOP_HOME/bin
export HADOOP_HOME_WARN_SUPPRESS=1
```

保存，最后需要注销然后再次登陆，或重启一下，才能保证 JAVA_HOME 在新打开的终端窗口中都能使用（注销、重启后，新打开一个终端窗口，输入 echo $JAVA_HOME 检验）。

2 HADOOP 安装

2.1 安装 Hadoop

建议删除/opt/hadoop/安装环境，从零开始配置 Hadoop 完全分布式环境。

1) 以 hadoop 用户远程登录 Poplar 服务器，下载 hadoop-2.7.1.tar.gz ，并将其拷贝到 Poplar 服务器的/home/hadoop/目录下。
2) 解压 Hadoop 源文件

```
sudo tar -zxvf ~/Downloads/hadoop-2.7.1.tar.gz -C /opt    # 解压到/opt 中
```

3) 重命名 hadoop

```
sudo mv /opt/hadoop-2.7.1/ /opt/hadoop                    # 将文件名改为 hadoop
```

4) 设置 hadoop 文件夹的用户属组和用户组

很关键到一步，便于 hadoop 用户对该文件夹的文件拥有读写权限，不然后续 hadoop 启动后，无法在该文件夹创建文件和写入日志信息。

```
sudo chown -R hadoop:hadoop /usr/hadoop
```

5) 删除安装包

```
rm -rf  /home/hadoop/hadoop-2.7.1.tar.gz
```

6) 配置环境变量

前面我们已经配置了环境变量，这里可以不用配置。Hadoop 解压后即可使用。输入如下命令 Hadoop 检查是否可用，成功则会显示命令行的用法：

```
[hadoop@Poplar:~]$ $HADOOP_HOME/bin/hadoop
Usage: hadoop [--config confdir] [COMMAND | CLASSNAME]
  CLASSNAME            run the class named CLASSNAME
 or
  where COMMAND is one of:
  fs                   run a generic filesystem user client
  version              print the version
  jar <jar>            run a jar file
                       note: please use "yarn jar" to launch
                             YARN applications, not this command.
  checknative [-a|-h]  check native hadoop and compression libraries availability
  distcp <srcurl> <desturl> copy file or directories recursively
  archive -archiveName NAME -p <parent path> <src>* <dest> create a hadoop archive
  classpath            prints the class path needed to get the
  credential           interact with credential providers
                       Hadoop jar and the required libraries
  daemonlog            get/set the log level for each daemon
  trace                view and modify Hadoop tracing settings
```

```
Most commands print help when invoked w/o parameters.
```

2.2 Hadoop 单机配置

Hadoop 默认配置是以非分布式模式运行，即单 Java 进程，方便进行调试。可以执行附带的例子 WordCount 来感受下 Hadoop 的运行。例子将 Hadoop 的配置文件作为输入文件，统计符合正则表达式 dfs[a-z.]+的单词的出现次数。

```
[hadoop@Poplar:~]$ mkdir -p ~/hadoop/tmp/input
[hadoop@Poplar:~]$ cd hadoop/tmp
hadoop@Poplar:~/hadoop/tmp$ cp $HADOOP_HOME/etc/hadoop/*.xml input
hadoop@Poplar:~/hadoop/tmp$ $HADOOP_HOME/bin/hadoop jar
$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.1.jar grep input
output 'dfs[a-z.]+'
```

执行成功后如下所示，输出了作业的相关信息，输出的结果是符合正则的单词 dfsadmin 出现了 1 次。Hadoop 单机 WordCount 输出结果：

```
15/12/15 11:42:31 INFO Configuration.deprecation: session.id is deprecated. Instead,
use dfs.metrics.session-id
……
……
15/12/15 11:42:34 INFO mapreduce.Job:  map 100% reduce 0%
……
……
15/12/15 11:42:37 INFO mapreduce.Job:  map 100% reduce 100%
15/12/15 11:42:37 INFO mapreduce.Job: Job job_local1782558076_0001 completed
successfully
15/12/15 11:42:37 INFO mapreduce.Job: Counters: 30
   File System Counters
          ……
   Map-Reduce Framework
          ……
   Shuffle Errors
          ……
   File Input Format Counters
          Bytes Read=26007
   File Output Format Counters
          Bytes Written=123

……
hadoop@Poplar:~/hadoop/tmp$ cat ./output/*
1  dfsadmin
```

再次运行会提示出错，需要将./output 删除。

```
rm -R ./output
```

2.3 Hadoop 伪分布式配置

伪分布式模式也叫单节点集群模式， NameNode、SecondaryNameNode、DataNode、ResourceManager、NodeManager 所有的守护进程全部运行在 Poplar 节点之上。

1) 修改配置文件

Hadoop 可以在单节点上以伪分布式的方式运行，Hadoop 进程以分离的 Java 进程来运行，节点即是 NameNode 也是 DataNode。需要修改 2 个配置文件$HADOOP_HOME/etc/hadoop/core-site.xml

和$HADOOP_HOME/etc/hadoop/hdfs-site.xml。Hadoop 的配置文件是 xml 格式，声明 property 的 name 和 value。

修改配置文件$HADOOP_HOME/etc/hadoop/core-site.xml，将：

```
<configuration>
</configuration>
```

修改为下面配置：

```
<configuration>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>file:/home/hadoop/hadoop/tmp</value>
        <description>A base for other temporary directories.</description>
    </property>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://Poplar:9000</value>
    </property>
</configuration>
```

修改配置文件 etc/hadoop/hdfs-site.xml 为

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:/home/hadoop/hadoop/dfs/name</value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:/home/hadoop/hadoop/dfs/data</value>
    </property>
</configuration>
```

关于配置的一点说明：上面只要配置 fs.default.name 和 dfs.replication 就可以运行，不过有个说法是如没有配置 hadoop.tmp.dir 参数，此时 Hadoop 默认的使用的临时目录为 /tmp/hadoo-hadoop，而这个目录在每次重启后都会被干掉，必须重新执行 format 才行（未验证），所以伪分布式配置中最好还是设置一下。此外也需要显式指定 dfs.namenode.name.dir 和 dfs.datanode.data.dir，否则下一步可能会出错。

修改 etc/hadoop/mapred-site.xml 文件配置为：

```
<configuration>
    <property>
        <name>mapred.job.tracker</name>
        <value>localhost:9001</value>
    </property>
</configuration>
```

修改 etc/hadoop/yarn-site.xml 文件配置为：

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
```

```
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
</configuration>
```

2) 初始化 HDFS 文件系统

配置完成后，首先初始化文件系统 HDFS：

```
[hadoop@Poplar:~]$ $HADOOP_HOME/bin/hdfs namenode -format
```

成功的话，最后的提示如下，Exitting with status 0 表示成功，Exitting with status 1: 则是出错。若出错，可试着加上 sudo, 既 sudo bin/hdfs namenode -format 试试看。

```
[hadoop@Poplar:~]$ $HADOOP_HOME/bin/hdfs namenode -format
15/12/15 12:11:02 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = Poplar/192.168.42.121
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 2.7.1
STARTUP_MSG:   classpath =
……
……
15/12/15 12:11:05 INFO util.ExitUtil: Exiting with status 0
15/12/15 12:11:05 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at Poplar/192.168.42.121
************************************************************/
```

3) 开启 NameNode 和 DataNode 守护进程

```
[hadoop@Poplar:~]$ $HADOOP_HOME/sbin/start-dfs.sh
```

若出现下面 SSH 的提示，输入 yes 即可。

```
[hadoop@Poplar:~]$ $HADOOP_HOME/sbin/start-dfs.sh
Starting namenodes on [localhost]
localhost: starting namenode, logging to /opt/hadoop/logs/hadoop-hadoop-namenode-
Poplar.out
localhost: starting datanode, logging to /opt/hadoop/logs/hadoop-hadoop-datanode-
Poplar.out
Starting secondary namenodes [0.0.0.0]
The authenticity of host '0.0.0.0 (0.0.0.0)' can't be established.
ECDSA key fingerprint is 8d:fd:62:dc:75:0f:6b:e8:08:36:23:49:be:6d:5d:df.
Are you sure you want to continue connecting (yes/no)? yes
0.0.0.0: Warning: Permanently added '0.0.0.0' (ECDSA) to the list of known hosts.
0.0.0.0: starting secondarynamenode, logging to /opt/hadoop/logs/hadoop-hadoop-
secondarynamenode-Poplar.out
```

有可能会出现如下很多的 warn 提示，下面的步骤中也会出现，特别是 native-hadoop library 这个提示，可以忽略，并不会影响 hadoop 的功能。想解决这些提示可以看后面的附加教程(最好还是解决下，不困难，省得看这么多无用提示)。

成功启动后，可以通过命令 jps 看到启动了如下进程 NameNode、DataNode 和 SecondaryNameNode。

```
hadoop@Poplar:~$ jps
```

```
12624 NameNode
12757 DataNode
13101 Jps
12974 SecondaryNameNode
```

4) 通过查看启动日志分析启动失败原因

有时 Hadoop 无法正确启动，如 NameNode 进程没有顺利启动，这时可以查看启动日志来排查原因，不过新手可能需要注意几点：

启动时会提示形如 “Master: starting namenode, logging to /opt/hadoop/logs/hadoop-hadoop-namenode-Poplar.out”，其中 Poplar 对应你的机器名，但其实启动日志信息是记录在 /opt/hadoop/logs/hadoop-hadoop-namenode-Poplar.log 中，所以应该查看这个 .log 的文件。

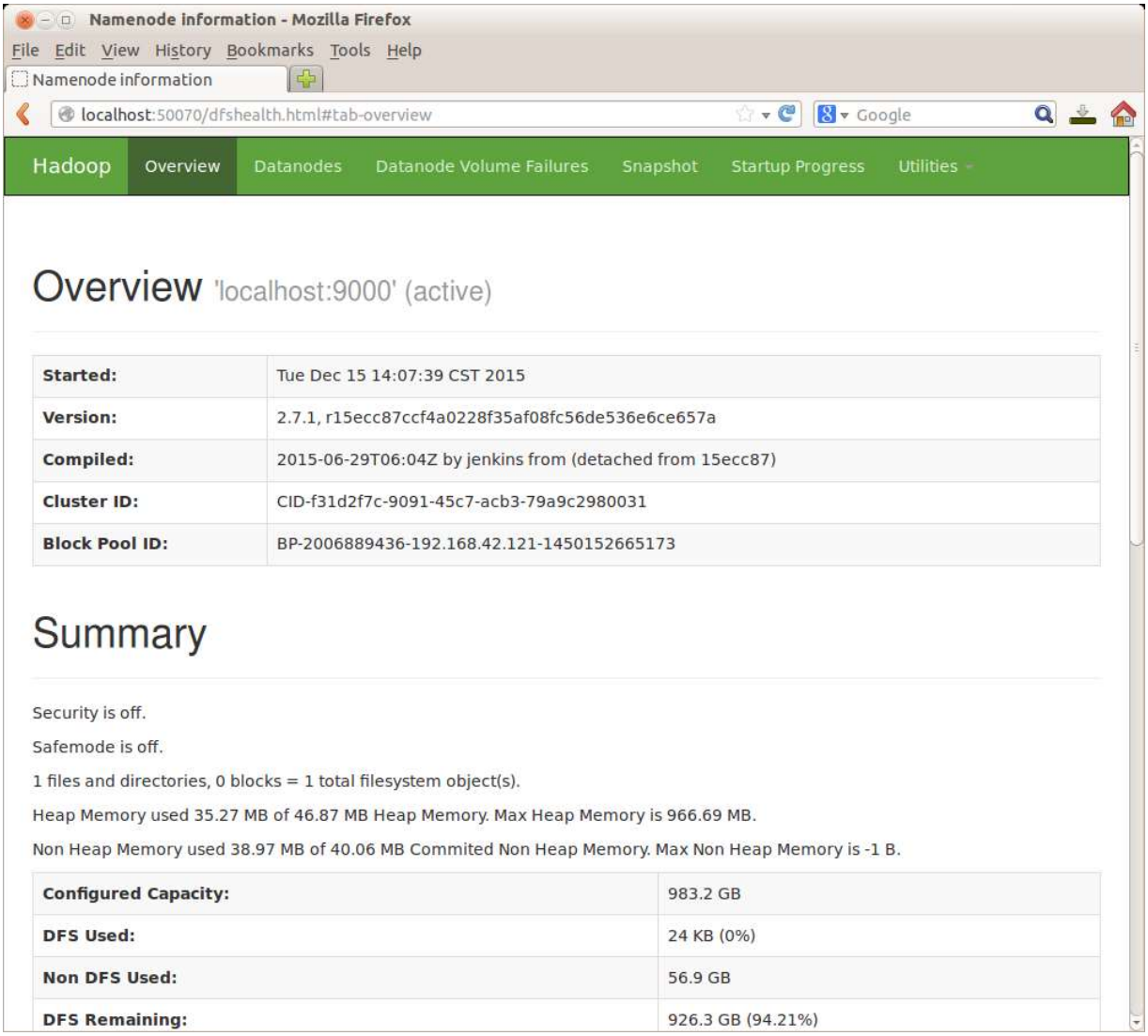
每一次的启动日志都是追加在日志文件之后，所以得拉到最后面看，这个看下记录的时间就知道了。

一般出错的提示在最后面，也就是写着 Fatal、Error 或者 Java Exception 的地方。

5) Hadoop Web 界面

此时可以访问 Web 界面 http://localhost:50070 来查看 Hadoop 的信息。

Figure 2-1 Hadoop 的 Web 界面



2.4 Hadoop 伪分布式实例-WordCount

首先创建所需的几个目录。

```
[hadoop@Poplar:~]$ $HADOOP_HOME/bin/hdfs dfs -mkdir /user
[hadoop@Poplar:~]$ $HADOOP_HOME/bin/hdfs dfs -mkdir /user/hadoop
[hadoop@Poplar:~]$ $HADOOP_HOME/bin/hdfs dfs -mkdir /user/hadoop/input
```

上一步创建的 /user/hadoop/input 是我们在 HDFS 中创建的一个目录，用来存储我们要分析的文件。接着将本地磁盘文件系统目录$HADOOP_HOME/etc/hadoop 中的文件作为输入文件复制到分布式文件系统中，即将/opt/hadoop/etc/hadoop 复制到分布式文件系统中的/user/hadoop/input 中。下面的命令的目标路径就是 /user/hadoop/input：

```
$HADOOP_HOME/bin/hdfs dfs -put $HADOOP_HOME/etc/hadoop/*.xml /user/hadoop/input
```

运行 MapReduce 作业，执行成功的话跟单机模式相同，输出作业信息。

```
[hadoop@Poplar:~]$ $HADOOP_HOME/bin/hadoop jar
$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.1.jar grep input
output 'dfs[a-z.]+'
```

以下为命令输出的运行 log。

```
15/12/15 15:00:56 INFO Configuration.deprecation: session.id is deprecated. Instead,
use dfs.metrics.session-id
15/12/15 15:00:56 INFO jvm.JvmMetrics: Initializing JVM Metrics with
processName=ResourceManager, sessionId=
……
……
15/12/15 15:02:01 INFO mapreduce.Job:  map 100% reduce 0%
……
15/12/15 15:02:02 INFO mapreduce.Job:  map 100% reduce 100%
15/12/15 15:02:02 INFO mapreduce.Job: Job job_local1436447405_0002 completed
successfully
……
```

查看运行结果。注意到跟单机模式中用的不是相同的数据，所以运行结果不同（换成原来的数据，结果是一致的）。

```
[hadoop@Poplar:~]$ $HADOOP_HOME/bin/hdfs dfs -cat /user/hadoop/output/*
1  dfsadmin
1  dfs.replication
1  dfs.namenode.name.dir
1  dfs.datanode.data.dir
```

也可以将运行结果取回到本地。

```
[hadoop@Poplar:~]$ $HADOOP_HOME/bin/hdfs dfs -get /user/hadoop/output output
```

可以看到，使用 bin/hdfs dfs -命令可操作分布式文件系统， 如：

```
$HADOOP_HOME/bin/hdfs dfs -ls /user/hadoop         # 查看`/user/hadoop`中的文件
$HADOOP_HOME/bin/hdfs dfs -rm -R /user/hadoop/input/*    # 删除 input 中的文件
$HADOOP_HOME/bin/hdfs dfs -rm -R /user/hadoop/output     # 删除 output 中的文件
```

注意：运行程序时，输出目录需不存在

运行 Hadoop 程序时，结果的输出目录（如 output）不能存在，否则会提示错误，因此运行前需要先删除输出目录。建议在程序中加上如下代码进行删除，避免繁琐的命令行操作：

```
Configuration conf = new Configuration();
Job job = new Job(conf);
...
/* 删除输出目录 */
Path outputPath = new Path(args[1]);
outputPath.getFileSystem(conf).delete(outputPath, true);
...
```

结束 Hadoop 进程，则运行：

```
$HADOOP_HOME/sbin/stop-dfs.sh
```

注意：下次再启动 hadoop，无需进行 HDFS 的初始化，只需要运行 sbin/stop-dfs.sh 就可以。

3 HADOOP 集群安装

完成单机模式配置或者伪分布模式配置，我就可以进行 HADOOP 的开发和测试。但是生产环境要运行在完全分布式。

在本章我们将搭建完全分布式环境，基于 Hadoop 2.7.1，但应该适用于所有 2.x 版本。NameNode、SecondaryNameNode、ResourceManager 守护进程在主节点上，DataNode、NodeManager 在从节点上。本教程只是基础的安装配置，更多功能、配置、技巧就需要各位自行探索了。

3.1 环境

参考第 1.1 节安装环境节的说明。

3.2 准备工作

先按照教程第 2 章 HADOOP 安装，在**所有机器上**配置 hadoop 用户、安装 SSH server、安装 Java 环境，在 **Master 主机**上安装 Hadoop。

Hadoop 的安装配置只需要在 Master 节点主机上进行，配置好后再复制到各个节点。

建议先按照上面的教程在 Master 主机上安装一次单机环境的 Hadoop，如果直接上手集群，在 Master 主机上安装 Hadoop 时，要记得修改 hadoop 文件的权限。

3.3 网络配置

参考 1.4 网络配置的说明。

3.4 SSH 无密码登陆节点

参考 1.5 安装配置 SSH 的说明。

3.5 配置集群/分布式环境

集群/分布式模式需要修改 etc/hadoop 中的 5 个配置文件，后四个文件可点击查看官方默认设置值，这里仅设置了正常启动所必须的设置项： slaves、core-site.xml、hdfs-site.xml、mapred-site.xml、yarn-site.xml 。

1) 文件 slaves

```
cd /opt/hadoop/etc/hadoop
vi slaves
```

将原来 localhost 删除，把所有 Slave 的主机名写上，每行一个。例如我只有 3 个 Slave 节点，那么该文件中就有 3 行内容：

```
POPSlave1
POPSlave2
POPSlave3
```

2) core-site.xml

将原本的如下内容：

```
<property>
</property>
```

改为下面的配置。后面的配置文件的修改类似。

```
<property>
    <name>fs.default.name</name>
    <value>hdfs://Poplar:9000</value>
</property>
<property>
    <name>hadoop.tmp.dir</name>
    <value>file:/home/hadoop/hadoop/tmp</value>
    <description>A base for other temporary directories.</description>
</property>
```

3) hdfs-site.xml

因为只有 3 个 Slave，所以 dfs.replication 的值设为 3。

```
<property>
    <name>dfs.namenode.secondary.http-address</name>
    <value>Poplar:50090</value>
</property>
<property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/home/hadoop/hadoop/dfs/name</value>
</property>
<property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/home/hadoop/hadoop/dfs/data</value>
</property>
<property>
    <name>dfs.replication</name>
    <value>3</value>
</property>
```

4) mapred-site.xml

这个文件不存在，首先需要从模板中复制一份：

```
cp mapred-site.xml.template mapred-site.xml
```

然后配置修改如下：

```
<property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
</property>
```

5) yarn-site.xml

```
<property>
    <name>yarn.resourcemanager.hostname</name>
    <value>Poplar</value>
</property>
<property>
    <name>yarn.nodemanager.aux-services</name>
```

```
    <value>mapreduce_shuffle</value>
</property>
```

配置好后，将 Poplar 上的 Hadoop 文件复制到各个节点上(虽然直接采用 scp 复制也可以正确运行，但会有所不同，如符号链接 scp 过去后就有点不一样了。所以先打包再复制比较稳妥)。

```
cd /opt
sudo tar -zcf ./hadoop.tar.gz ./hadoop
scp ./hadoop.tar.gz POPSlave1:/home/hadoop
```

在 POPSlave1 上执行：

```
sudo tar -zxf ~/hadoop.tar.gz -C /opt
sudo chown -R hadoop:hadoop /opt/hadoop
```

如果之前有跑过伪分布式模式，建议切换到集群模式前先删除之前的临时文件：

```
rm -r /home/hadoop/hadoop/tmp
```

切换 Hadoop 模式应删除之前的临时文件

切换 Hadoop 的模式，不管是从集群切换到伪分布式，还是从伪分布式切换到集群，如果遇到无法正常启动的情况，可以删除所涉及节点的临时文件夹，这样虽然之前的数据会被删掉，但能保证集群正确启动。或者可以为集群模式和伪分布式模式设置不同的临时文件夹（未验证）。所以如果集群以前能启动，但后来启动不了，特别是 DataNode 无法启动，不妨试着删除所有节点（包括 POPSlave 节点）上的 tmp 文件夹，重新执行一次 bin/hdfs namenode -format，再次启动试试。

然后在 Master 节点上就可以启动 hadoop 了。

```
cd /opt/hadoop/
bin/hdfs namenode -format       # 首次运行需要执行初始化，后面不再需要
sbin/start-dfs.sh
sbin/start-yarn.sh
```

通过 jps 查看 Master 的 Hadoop 进程。可以看到 Poplar 节点启动了 NameNode、SecondrryNameNode、ResourceManager 进程。

通过 jps 查看 Slave 的 Hadoop 进程。POPSlave 节点则启动了 DataNode 和 NodeManager 进程。

另外也可以在 Poplar 节点上通过命令 bin/hdfs dfsadmin -report 查看 DataNode 是否正常启动。例如我这边一共有 3 个 Datanodes。

通过 dfsadmin 查看 DataNode 的状态。

通过查看启动日志分析启动失败原因。

有时 Hadoop 集群无法正确启动，如 Master 上的 NameNode 进程没有顺利启动，这时可以查看启动日志来排查原因，不过新手可能需要注意几点：

启动时会提示 “Master: starting namenode, logging to /opt/hadoop/logs/hadoop-hadoop-namenode-Poplar.out”，但其实启动日志信息是记录在 /opt/hadoop/logs/hadoop-hadoop-namenode-Poplar.log 中；

每一次的启动日志都是追加在日志文件之后，所以得拉到最后面看，这个看下记录的时间就知道了。

一般出错的提示在最后面，也就是写着 Error 或者 Java 异常的地方。

也可以通过 Web 页面看到查看 DataNode 和 NameNode 的状态，http://Poplar:50070/

关闭 Hadoop 集群也是在 Poplar 节点上执行：

```
sbin/stop-dfs.sh
sbin/stop-yarn.sh
```

4 ZOOKEEPER 安装配置

4.1 ZooKeeper 安装

4.1.1 下载安装包

通过 Apache 网站下载 ZooKeeper 安装包。

4.1.2 解压安装包

解压到/opt 下：

```
tar -zxvf ~/Downloads/zookeeper-3.4.7.tar.gz -C /opt      # 解压到/opt
mv /opt/zookeeper-3.4.7 /opt/zookeeper                    # 重命名
```

4.1.3 配置环境变量

建个文件加方 zookeeper 的数据，所有节点都需要。

```
mkdir -p /home/hadoop/hadoop/zookeeper/data
mkdir -p /home/hadoop/hadoop/zookeeper/log          # log 目录
```

编辑/etc/environment 文件添加：

```
ZOOKEEPER=/opt/zookeeper
```

添加 PATH 设置：

```
PATH="/opt/zookeeper/bin:/opt/hadoop/bin:/opt/hadoop/sbin:/opt/hbase/bin:/opt/go/bin:
/opt/jdk/bin/:/opt/liteide/bin/:/opt/vscode:/usr/local/sbin:/usr/local/bin:/usr/sbin:
/usr/bin:/sbin:/bin:/usr/games:/usr/local/games"
```

设置 LOG 路径：

```
ZOO_LOG_DIR=/home/hadoop/hadoop/zookeeper/log/
```

4.2 单机模式

ZooKeeper 可以配置成在单机模式：

```
[hadoop@Poplar:~]$ cd /opt/zookeeper/conf
[hadoop@Poplar:/opt/zookeeper/conf]$ cp zoo_sample.cfg zoo.cfg
[hadoop@Poplar:/opt/zookeeper/conf]$ vi zoo.cfg
tickTime=2000
initLimit=10
syncLimit=5
dataDir=/home/hadoop/hadoop/zookeeper/data
clientPort=2181
```

参数说明：

- tickTime：zookeeper 中使用的基本时间单位，毫秒值
- dataDir：数据目录，可以是任意目录

- dataLogDir：log 目录，同样可以是任意目录。如果没有设置该参数，将使用和 dataDir 相同的设置
- clientPort：监听 client 连接的端口号

至此, zookeeper 的单机模式已经配置好了. 启动 server 只需运行脚本：

```
[hadoop@Poplar:/opt/zookeeper]$ bin/zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /opt/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

Server 启动之后，就可以启动 client 连接 server 了，执行脚本：

```
[hadoop@Poplar:/opt/zookeeper]$ bin/zkCli.sh -server Poplar:2181
Connecting to Poplar:2181
……
……
WatchedEvent state:SyncConnected type:None path:null
```

敲回车可以进入到 ZooKeeper 的命令行，输入命令 help、status 等命令。

有问题查看/home/hadoop/hadoop/zookeeper/log/zookeeper.out。

4.3 伪集群模式

所谓伪集群，是指在单台机器中启动多个 zookeeper 进程，模拟一个集群。因为是在一台机器上模拟集群，所以端口不能重复，这里用 2181~2183，2287~2289，以及 3387~3389 相互错开。另外每个 ZooKeeper 的 instance，都需要设置独立的数据存储目录、日志存储目录，所以 dataDir 这个节点对应的目录，需要手动先创建好。

我们先复制三份 ZooKeeper 程序，并建立 3 个 dataDir 目录：

```
[hadoop@Poplar:/opt]$ cp -a zookeeper zookeeper0
[hadoop@Poplar:/opt]$ cp -a zookeeper zookeeper1
[hadoop@Poplar:/opt]$ cp -a zookeeper zookeeper2
[hadoop@Poplar:/opt]$ mkdir -p /home/hadoop/hadoop/zookeeper/data0
[hadoop@Poplar:/opt]$ mkdir -p /home/hadoop/hadoop/zookeeper/data1
[hadoop@Poplar:/opt]$ mkdir -p /home/hadoop/hadoop/zookeeper/data2
```

然后对应修改其 zoo.cfg 配置文件，/opt/zookeeper0/conf/zoo.cfg 内容如下：

```
tickTime=2000
initLimit=10
syncLimit=5
dataDir=/home/hadoop/hadoop/zookeeper/data0
clientPort=2181
server.0=Poplar:2287:3387
server.1=Poplar:2288:3388
server.2=Poplar:2289:3389
```

/opt/zookeeper1/conf/zoo.cfg 内容如下：

```
tickTime=2000
initLimit=10
syncLimit=5
dataDir=/home/hadoop/hadoop/zookeeper/data1
clientPort=2182
server.0=Poplar:2287:3387
server.1=Poplar:2288:3388
server.2=Poplar:2289:3389
```

/opt/zookeeper2/conf/zoo.cfg 内容如下：

```
tickTime=2000
initLimit=10
syncLimit=5
dataDir=/home/hadoop/hadoop/zookeeper/data2
clientPort=2183
server.0=Poplar:2287:3387
server.1=Poplar:2288:3388
server.2=Poplar:2289:3389
```

新增了几个参数，其含义如下：

- initLimit：zookeeper 集群中的包含多台 server，其中一台为 leader，集群中其余的 server 为 follower。initLimit 参数配置初始化连接时，follower 和 leader 之间的最长心跳时间。此时该参数设置为 10，说明时间限制为 10 倍 tickTime，即 10*2000=20000ms=20s。
- syncLimit：该参数配置 leader 和 follower 之间发送消息，请求和应答的最大时间长度。此时该参数设置为 5，说明时间限制为 5 倍 tickTime，即 10000ms。
- server.X=A:B:C：其中 X 是一个数字，表示这是第几号 server；A 是该 server 所在的 IP 地址；B 配置该 server 和集群中的 leader 交换消息所使用的端口；C 配置选举 leader 时所使用的端口。由于配置的是伪集群模式，所以各个 server 的 B，C 参数必须不同。

在之前设置的 dataDir 中新建 myid 文件，写入一个数字，该数字表示这是第几号 server。该数字必须和 zoo.cfg 文件中的 server.X 中的 X 一一对应。/home/hadoop/hadoop/zookeeper/data0/myid 文件中写入 0，/home/hadoop/hadoop/zookeeper/data1/myid 文件中写入 1，/home/hadoop/hadoop/zookeeper/data2/data/myid 文件中写入 2。

```
[hadoop@Poplar:/opt]$ echo 0 > ~/hadoop/zookeeper/data0/myid
[hadoop@Poplar:/opt]$ echo 1 > ~/hadoop/zookeeper/data1/myid
[hadoop@Poplar:/opt]$ echo 2 > ~/hadoop/zookeeper/data2/myid
```

分别启动各个实例的 ZooKeeper：

```
[hadoop@Poplar:/opt]$ zookeeper0/bin/zkServer.sh start
[hadoop@Poplar:/opt]$ zookeeper1/bin/zkServer.sh start
[hadoop@Poplar:/opt]$ zookeeper2/bin/zkServer.sh start
```

按照前面单机模式下的方式启动 ZooKeeper shell 测试安装情况。

```
[hadoop@Poplar:/opt]$ zookeeper2/bin/zkCli.sh -server Poplar:2181
Connecting to Poplar:2181
……
……
WATCHER::

WatchedEvent state:SyncConnected type:None path:null
[zk: Poplar:2181(CONNECTED) 0] ls /
[zookeeper]
[zk: Poplar:2181(CONNECTED) 1] help
ZooKeeper -server host:port cmd args
   stat path [watch]
   set path data [version]
   ls path [watch]
   delquota [-n|-b] path
   ls2 path [watch]
   setAcl path acl
   setquota -n|-b val path
   history
```

```
   redo cmdno
   printwatches on|off
   delete path [version]
   sync path
   listquota path
   rmr path
   get path [watch]
   create [-s] [-e] path data acl
   addauth scheme auth
   quit
   getAcl path
   close
   connect host:port
[zk: Poplar:2181(CONNECTED) 2] quit
Quitting...
[hadoop@Poplar:/opt]$
```

4.4 集群模式

集群模式的配置和伪集群基本一致。由于集群模式下，各 server 部署在不同的机器上，因此各 server 的 conf/zoo.cfg 文件可以完全一样。下面是一个示例：

```
tickTime=2000
initLimit=10
syncLimit=5
dataDir=/home/hadoop/hadoop/zookeeper/data
clientPort=2181
server.0=Poplar:2287:3387
server.1=POPSlave1:2287:3387
server.2=POPSlave2:2287:3387
```

示例中部署了 3 台 zookeeper server，分别部署在 Poplar、POPSlave1、POPSlave2 上。需要注意的是，各 server 的 dataDir 目录下的 myid 文件中的数字必须不同。Poplar server 的 myid 为 0，POPSlave1 server 的 myid 为 1，POPSlave2 server 的 myid 为 2。

将配置好的 ZooKeeper 打包，分发到别的机器。

```
scp -r /opt/zookeeper hadoop@POPSlave1:/opt/zookeeper
scp -r /opt/zookeeper hadoop@POPSlave2:/opt/zookeeper
```

将每台设备的 ZooKeeper 启动。然后可以通过 ZooKeeper shell 测试启动情况。

5 HBASE 安装

5.1 下载安装包

hbase-1.1.2.tar.gz 版本与 hadoop-2.7.1 良好兼容，从官网下载 hbase-1.1.2.tar.gz 安装包，并将下载的 hbase-1.1.2.tar.gz 拷贝到 /home/hadoop 目录下。HBase 官网下载地址：http://archive.apache.org/dist/hbase/

5.2 解压安装包

```
[hadoop@Poplar:~]$ cd /opt
[hadoop@Poplar:/opt]$ sudo tar -xvf /home/hadoop/hbase-1.1.2.tar.gz  #解压安装源码包
[hadoop@Poplar:/opt]$ mv hbase-1.1.2 hbase                          #重命名
[hadoop@Poplar:/opt]$ cd hbase
[hadoop@Poplar:/opt]$ sudo chown -R hadoop:hadoop hbase             #赋予 HBase 安装目录下所有文件 hadoop 权限
```

5.3 配置安装路径

如果已经按照第 1 章服务器基础环境搭建的说明进行安装，已经将 HBase 路径加入，则不需要进行下面的操作。

```
# 将 HBase 下的 bin 目录添加到系统的 path 中，在/etc/profile 文件尾行添加如下的内容
[hadoop@Poplar:/opt]$ sudo vim /etc/profile
Export PATH=$PATH:/opt/hbase/bin
#执行 source 命令使上述配置在当前终端立即生效
[hadoop@Poplar:/opt]$ source /etc/profile
```

5.4 验证是否安装成功

```
[hadoop@Poplar:opt]$ hbase version
2015-12-16 11:57:09,973 INFO  [main] util.VersionInfo: HBase 1.1.2
2015-12-16 11:57:09,976 INFO  [main] util.VersionInfo: Source code repository
git://hw11397.local/Volumes/hbase-1.1.2RC2/HBase
revision=cc2b70cf03e3378800661ec5cab11eb43fafe0fc
2015-12-16 11:57:09,976 INFO  [main] util.VersionInfo: Compiled by ndimiduk on Wed
Aug 26 20:11:27 PDT 2015
2015-12-16 11:57:09,976 INFO  [main] util.VersionInfo: From source with checksum
73da41f3d1b867b7aba6166c77fafc17
```

看到以上打印消息表示 HBase 已经安装成功，接下来将分别进行 HBase 单机模式和伪分布式模式的配置。

6 HBASE 单机模式

6.1 配置 hbase-env.sh

将 JAVA_HOME 变量设置为 Java 安装的根目录，配置如下所示：

```
[hadoop@Poplar:/opt/hbase]$ vim conf/hbase-env.sh
```

对 hbase-env.sh 文件做如下修改：

```
export JAVA_HOME=/opt/jdk    # 配置本机的 java 安装根目录
export HBASE_MANAGES_ZK=true # 配置由 HBase 自己管理 zookeeper,不需要单独的 zookeeper。
```

6.2 配置 hbase-site.xml

在启动 HBase 前需要设置属性 hbase.rootdir，用于指定 HBase 数据的存储位置，此处设置为 HBase 安装目录下的 hbase-tmp 文件夹即（file:///home/hadoop/hbase/hbase-tmp），配置如下：

```
[hadoop@Poplar:/opt/hbase]$ vim conf/hbase-site.sh
<configuration>
    <property>
        <name>hbase.rootdir</name>
        <value>file:///home/hadoop/hbase/hbase-tmp</value>
    </property>
</configuration>
```

特别注意：hbase.rootdir 默认为/tmp/hbase-${user.name},这意味着每次重启系统都会丢失数据。

6.3 启动 HBase

```
[hadoop@Poplar:/opt/hbase]$ start-hbase.sh
starting master, logging to /opt/hbase/bin/../logs/hbase-hadoop-master-Poplar.out
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option PermSize=128m; support was
removed in 8.0
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=128m; support
was removed in 8.0
```

6.4 进入 shell 模式

进入 shell 模式之后，通过 status 命令查看 HBase 的运行状态，通过 exit 命令退出 shell。

```
[hadoop@Poplar:/opt/hbase]$ hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hbase/lib/slf4j-log4j12-
1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop/share/hadoop/common/lib/slf4j-log4j12-
1.7.10.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.slf4j.impl.Log4jLoggerFactory]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.1.2, rcc2b70cf03e3378800661ec5cab11eb43fafe0fc, Wed Aug 26 20:11:27 PDT
2015

hbase(main):001:0> status
1 servers, 0 dead, 2.0000 average load
```

```
hbase(main):002:0> exit
[hadoop@Poplar:/opt/hbase]$
```

6.5 停止 HBase

```
[hadoop@Poplar:/opt/hbase]$ stop-hbase.sh
stopping hbase.....................
```

特别注意：如果在操作 HBase 的过程中发生错误，可以通过{HBASE_HOME}目录（/opt/hbase）下的 logs 子目录中的日志文件查看错误原因。

7 HBASE 伪分布式模式

7.1 配置 hbase-env.sh

将 JAVA_HOME 变量设置为 Java 安装的根目录，配置如下所示：

```
[hadoop@Poplar:/opt/hbase]$ vim conf/hbase-env.sh
```

对 hbase-env.sh 文件做如下修改：

```
export JAVA_HOME=/opt/jdk    # 配置本机的 java 安装根目录
export HBASE_MANAGES_ZK=true # 配置由 HBase 自己管理 zookeeper,不需要单独的 zookeeper。
```

7.2 配置 hbase-site.xml

修改 hbase.rootdir，将其指向 Poplar(与 hdfs 的端口保持一致)，并指定 HBase 在 HDFS 上的存储路径。将属性 hbase.cluter.distributed 设置为 true。假设当前 Hadoop 集群运行在伪分布式模式下，且 NameNode 运行在 9000 端口；

```
[hadoop@Poplar:/opt/hbase]$ vim conf/hbase-site.xml
<configuration>
    <property>
        <name>hbase.rootdir</name>
        <value>hdfs://Poplar:9000/hbase</value>
    </property>
    <property>
        <name>hbase.cluster.distributed</name>
        <value>true</value>
    </property>
    <property>
        <name>hbase.master</name>
        <value>Poplar:60000</value>
    </property>
</configuration>
```

7.3 启动 HBase

完成以上操作后启动 HBase，启动顺序：先启动 Hadoop→再启动 HBase，关闭顺序：先关闭 Hbase→再关闭 Hadoop。

7.3.1 启动 hadoop 集群

```
[hadoop@Poplar:/opt/hbase]$ start-dfs.sh    # 启动 hadoop dfs
Starting namenodes on [localhost]
localhost: starting namenode, logging to /opt/hadoop/logs/hadoop-hadoop-namenode-
Poplar.out
localhost: starting datanode, logging to /opt/hadoop/logs/hadoop-hadoop-datanode-
Poplar.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /opt/hadoop/logs/hadoop-hadoop-
secondarynamenode-Poplar.out
[hadoop@Poplar:/opt/hbase]$ start-yarn.sh   # 启动 yarn
starting yarn daemons
```

```
starting resourcemanager, logging to /opt/hadoop/logs/yarn-hadoop-resourcemanager-
Poplar.out
localhost: starting nodemanager, logging to /opt/hadoop/logs/yarn-hadoop-nodemanager-
Poplar.out
[hadoop@Poplar:/opt/hbase]$ jps      # 查看进程
27011 DataNode
27239 SecondaryNameNode
26872 NameNode
27561 NodeManager
27419 ResourceManager
27598 Jps
```

特别注意：读者可先通过 jps 命令查看 Hadoop 集群是否启动，如果 Hadoop 集群已经启动，则不需要执行 Hadoop 集群启动操作。

7.3.2 启动 HBase

```
[hadoop@Poplar:/opt/hbase]$ start-hbase.sh
localhost: starting zookeeper, logging to /opt/hbase/bin/../logs/hbase-hadoop-
zookeeper-Poplar.out
starting master, logging to /opt/hbase/bin/../logs/hbase-hadoop-master-Poplar.out
starting regionserver, logging to /opt/hbase/bin/../logs/hbase-hadoop-1-regionserver-
Poplar.out
[hadoop@Poplar:/opt/hbase]$ jps
28289 HMaster
28194 HQuorumPeer
27011 DataNode
27239 SecondaryNameNode
26872 NameNode
27561 NodeManager
28521 Jps
27419 ResourceManager
28396 HRegionServer
```

7.4 进入 shell 模式

进入 shell 模式之后，通过 list 命令查看当前数据库所有表信息，通过 create 命令创建一个 member 表，其拥有 member_id, address, info 三个列族，通过 describe 命令查看 member 表结构，通过 exit 命令退出 HBase shell 模式。

```
[hadoop@Poplar hadoop]$ hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.94.20, r09c60d770f2869ca315910ba0f9a5ee9797b1edc, Fri May 23 22:00:41 PDT
2014

[hadoop@Poplar:/opt/hbase/conf]$ hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hbase/lib/slf4j-log4j12-
1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop/share/hadoop/common/lib/slf4j-log4j12-
1.7.10.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.slf4j.impl.Log4jLoggerFactory]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.1.2, rcc2b70cf03e3378800661ec5cab11eb43fafe0fc, Wed Aug 26 20:11:27 PDT
2015
```

```
hbase(main):001:0> create 'member','member_id','address','info'
0 row(s) in 8.7100 seconds

=> Hbase::Table - member
hbase(main):002:0> list
TABLE
member
1 row(s) in 0.1360 seconds

=> ["member"]
hbase(main):003:0> describe 'member'
Table member is ENABLED
member
COLUMN FAMILIES DESCRIPTION
{NAME => 'address', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false',
 KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER',
 COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE =>
'65536', REPLICATION_SCOPE => '0'}
{NAME => 'info', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KE
EP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', CO
MPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65
536', REPLICATION_SCOPE => '0'}
{NAME => 'member_id', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false
', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER
', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE =
> '65536', REPLICATION_SCOPE => '0'}
3 row(s) in 0.5920 seconds

hbase(main):004:0> exit
[hadoop@Poplar:/opt/hbase/conf]$
```

7.5 查看 HDFS 的 HBase 数据库文件

通过 hadoop fs -ls /hbase 命令查看 HBase 分布式数据库在 HDFS 上是否成功创建，/hbase/member 文件夹即为上一步我们所建立的 member 数据库在 HDFS 上的存储位置。

```
[hadoop@Poplar:/opt/hbase/conf]$ hadoop fs -ls /hbase
Found 8 items
drwxr-xr-x   - hadoop supergroup          0 2015-12-18 14:50 /hbase/.tmp
drwxr-xr-x   - hadoop supergroup          0 2015-12-18 14:49 /hbase/MasterProcWALs
drwxr-xr-x   - hadoop supergroup          0 2015-12-18 14:49 /hbase/WALs
drwxr-xr-x   - hadoop supergroup          0 2015-12-18 14:49 /hbase/corrupt
drwxr-xr-x   - hadoop supergroup          0 2015-12-18 11:54 /hbase/data
-rw-r--r--   1 hadoop supergroup         42 2015-12-18 11:53 /hbase/hbase.id
-rw-r--r--   1 hadoop supergroup          7 2015-12-18 11:53 /hbase/hbase.version
drwxr-xr-x   - hadoop supergroup          0 2015-12-18 14:49 /hbase/oldWALs
```

7.6 停止 HBase

完成上述操作后，执行关闭 HBase 操作，关闭顺序：先关闭 HBase→再关闭 Hadoop。

```
[hadoop@Poplar hadoop]$ stop-hbase.sh    #停止 HBase
stopping hbase..................
Poplar: stopping zookeeper.

[hadoop@Poplar hadoop]$ stop-all.sh  #停止 Hadoop
stopping ResourceManager
```

```
Poplar: stopping NodeManager
stopping namenode
Poplar: stopping datanode
Poplar: stopping secondarynamenode
```

8 HBASE 用户界面

8.1 HBase 用户界面

通过下面的链接可以访问 HBase 的一些相关信息，链接说明如下表格所示：

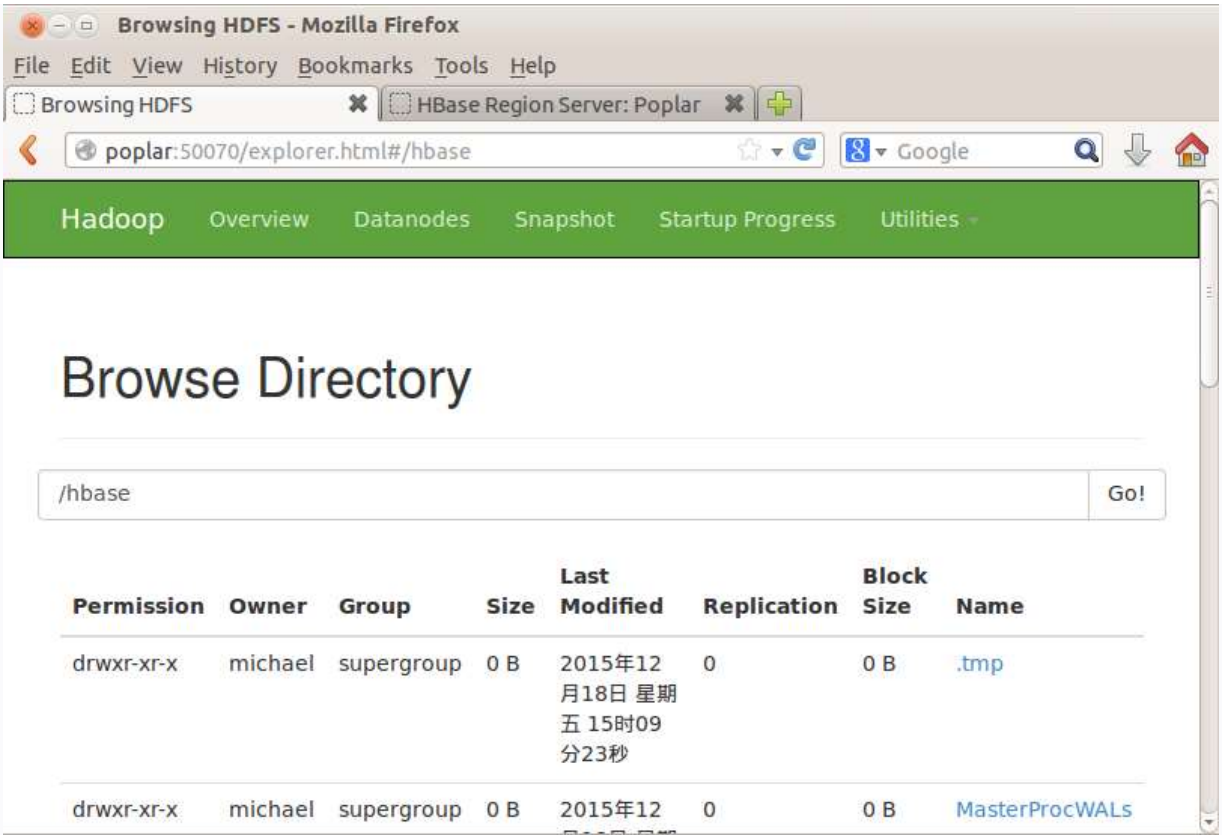
Table 8-1 HBase Web 页面地址

链接	说明
http://poplar:50070/explorer.html#/hbase	HBase 在 HDFS 上生成的/HBase 目录，用于存放数据
http://poplar:16010/master-status	Master 主页面
http://poplar:16010/zk.jsp	ZooKeeper 页面
http://poplar:16010/table.jsp?name=member	查看 member 表
http://poplar:16030/rs-status	Region 服务器页面

8.2 HDFS 主页

输入 http://poplar:50070/explorer.html#/hbase 进入 HDFS 主页，在该主页点击 Utilities 下的 Browse the filesystem 超链接，选择 hbase 目录，可以查看 HBase 在 HDFS 上生成的/hbase 目录结构，该目录用于存放 HBase 数据，如下图所示：

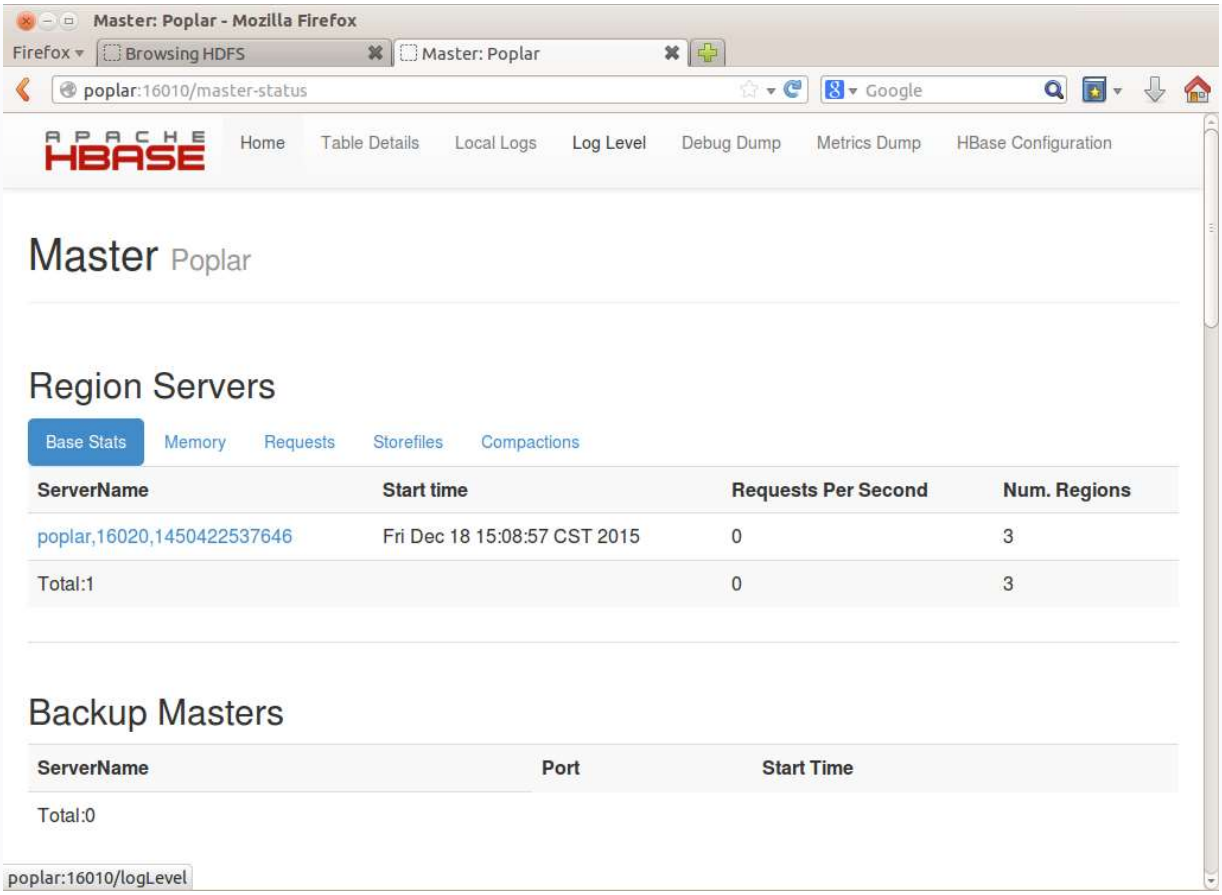
Figure 8-1 HBase 在 HDFS 上的信息



8.3 Master 页面

通过地址 http://poplar:16010/master-status 可以查看 HBase 的相关信皂，如下图所示。

Figure 8-2 HBase Master 页面

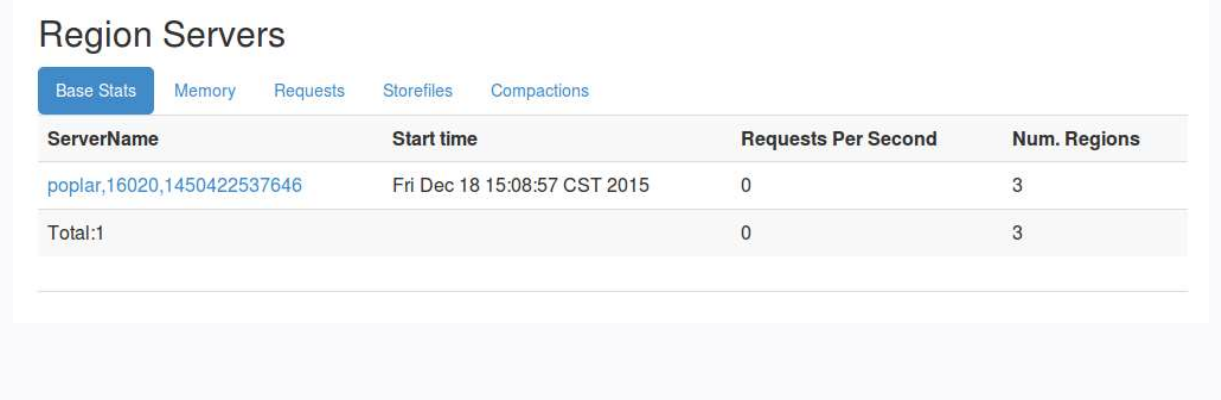


主要包含的信息如下：

- **Region Servers 信息**

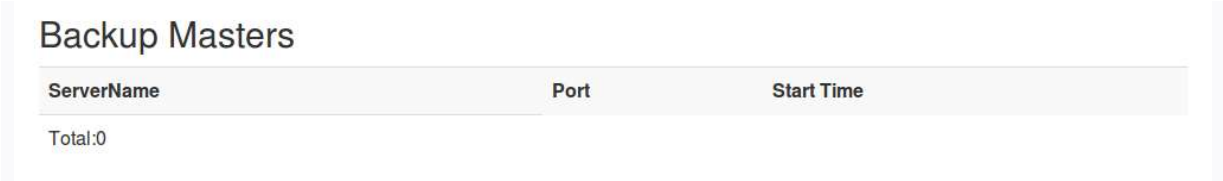
Region 服务器信息给出了所有 Region 服务器的地址，如下图所示；

Figure 8-3 Region Servers



■ **Backup Masters 信息**

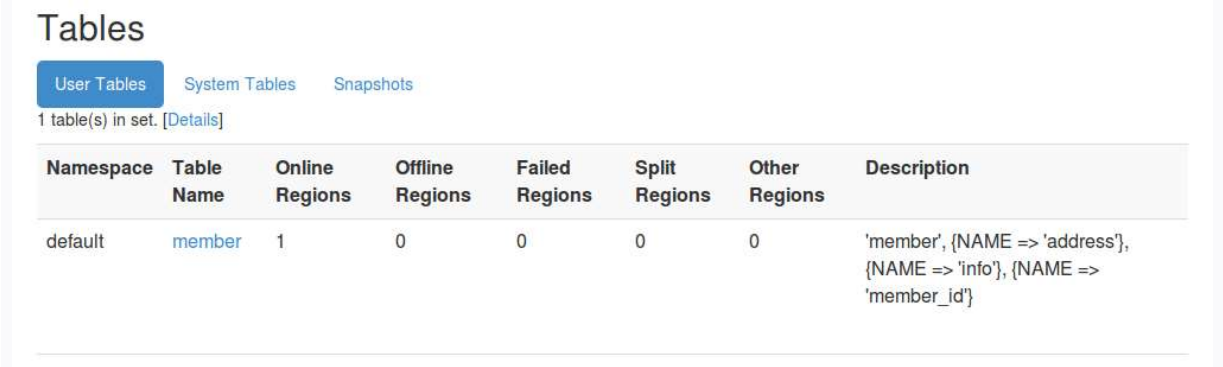
Figure 8-4 Backup Masters



■ **Tables 信息**

用户表信息给出了 HBase 中的表信息及相关属性，目录表信息包含系统表信息，可以点开查看。

Figure 8-5 Tables 信息



点击上图[Details]链接，跳转到 Tables Details 界面，如下图所示：

Figure 8-6 Details Tables

■ **Tasks 信息**

Tasks 包含 HBase 运行的任务信息。

■ **Software Attributes 信息**

Software Attributes 包含了当前集群的详细信息，从上往下依次为 HBase 的版本及编译信息、Hadoop 的版本及编译信息、HBase 根目录的路径、Region 服务器的平均负载以及 ZooKeeper Quorums 的地址。

Figure 8-7 Software Attributes

Software Attributes

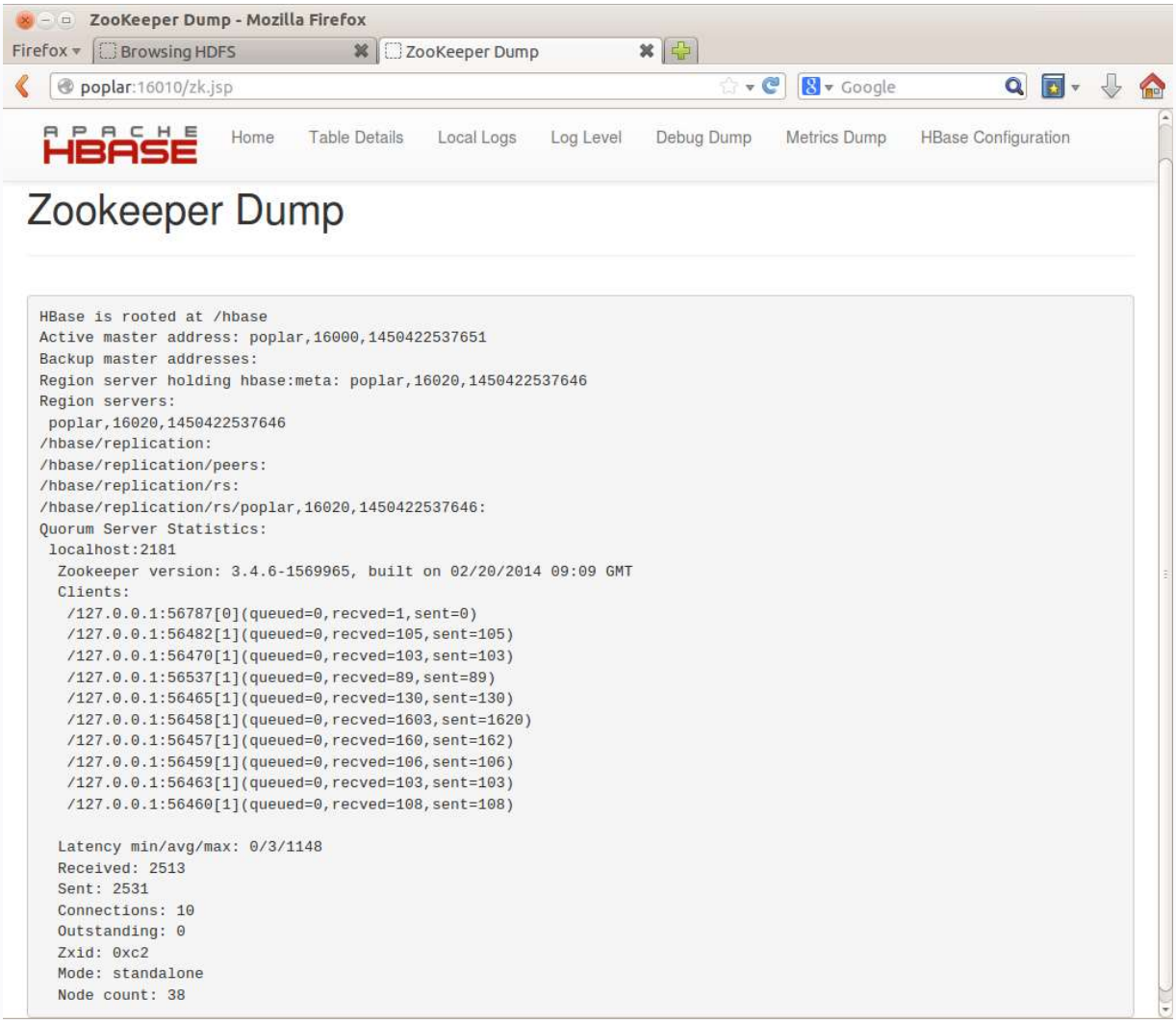
Attribute Name	Value	Description
HBase Version	1.1.2, revision=cc2b70cf03e3378800661ec5cab11eb43fafe0fc	HBase version and revision
HBase Compiled	Wed Aug 26 20:11:27 PDT 2015, ndimiduk	When HBase version was compiled and by whom
HBase Source Checksum	73da41f3d1b867b7aba6166c77fafc17	HBase source MD5 checksum
Hadoop Version	2.5.1, revision=2e18d179e4a8065b6a9f29cf2de9451891265cce	Hadoop version and revision
Hadoop Compiled	2014-09-05T23:05Z, kasha	When Hadoop version was compiled and by whom
Hadoop Source Checksum	6424fcab95bfff8337780a181ad7c78	Hadoop source MD5 checksum
ZooKeeper Client Version	3.4.6, revision=1569965	ZooKeeper client version and revision
ZooKeeper Client Compiled	02/20/2014 09:09 GMT	When ZooKeeper client version was compiled
Zookeeper Quorum	localhost:2181	Addresses of all registered ZK servers. For more, see zk dump.
Zookeeper Base Path	/hbase	Root node of this cluster in ZK.
HBase Root Directory	hdfs://Poplar:9000/hbase	Location of HBase home directory

HMaster Start Time	Fri Dec 18 15:08:57 CST 2015	Date stamp of when this HMaster was started
HMaster Active Time	Fri Dec 18 15:09:17 CST 2015	Date stamp of when this HMaster became active
HBase Cluster ID	21268884-4ebe-4349-be3e-843790fd611f	Unique identifier generated for each HBase cluster
Load average	3.00	Average number of regions per regionserver. Naive computation.
Coprocessors	[]	Coprocessors currently loaded by the master
LoadBalancer	org.apache.hadoop.hbase.master.balancer.StochasticLoadBalancer	LoadBalancer to be used in the Master

8.4 ZooKeeper 页面

通过 Software Attributes 信息中 Zookeeper Quorum 项提供的链接，可以进入 ZooKeeper 页面，该页面显示了 HBase 的根目录、当前的主 Master 地址、保存-ROOT-表的 Region 服务器的地址、其他 Region 服务器的地址及 ZooKeeper 的一些内部信息，如下图所示。

Figure 8-8 Zookeeper Dump



8.5 用户表页面

通过 Master 页面中用户表信息提供的链接 http://poplar:16010/table.jsp?name=member，可以进入用户表页面，如下图所示。该页面给出了表当前是否可用以及表在 Region 服务器上的信息。同时提供了根据行键合并及拆分表的操作。

Figure 8-9 Table 信息

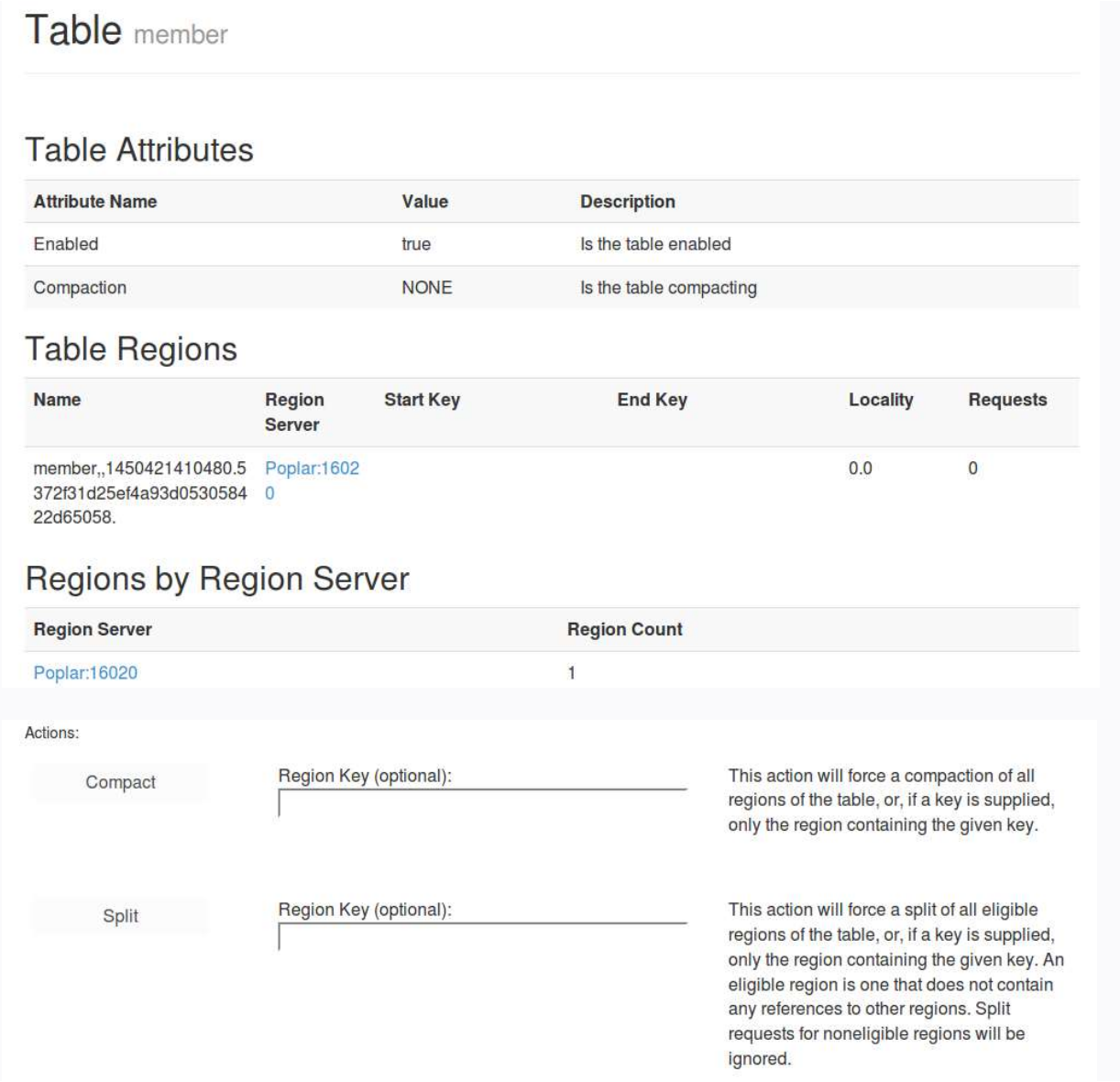


Table member

Table Attributes

Attribute Name	Value	Description
Enabled	true	Is the table enabled
Compaction	NONE	Is the table compacting

Table Regions

Name	Region Server	Start Key	End Key	Locality	Requests
member,,1450421410480.5372f31d25ef4a93d053058422d65058.	Poplar:16020			0.0	0

Regions by Region Server

Region Server	Region Count
Poplar:16020	1

Actions:

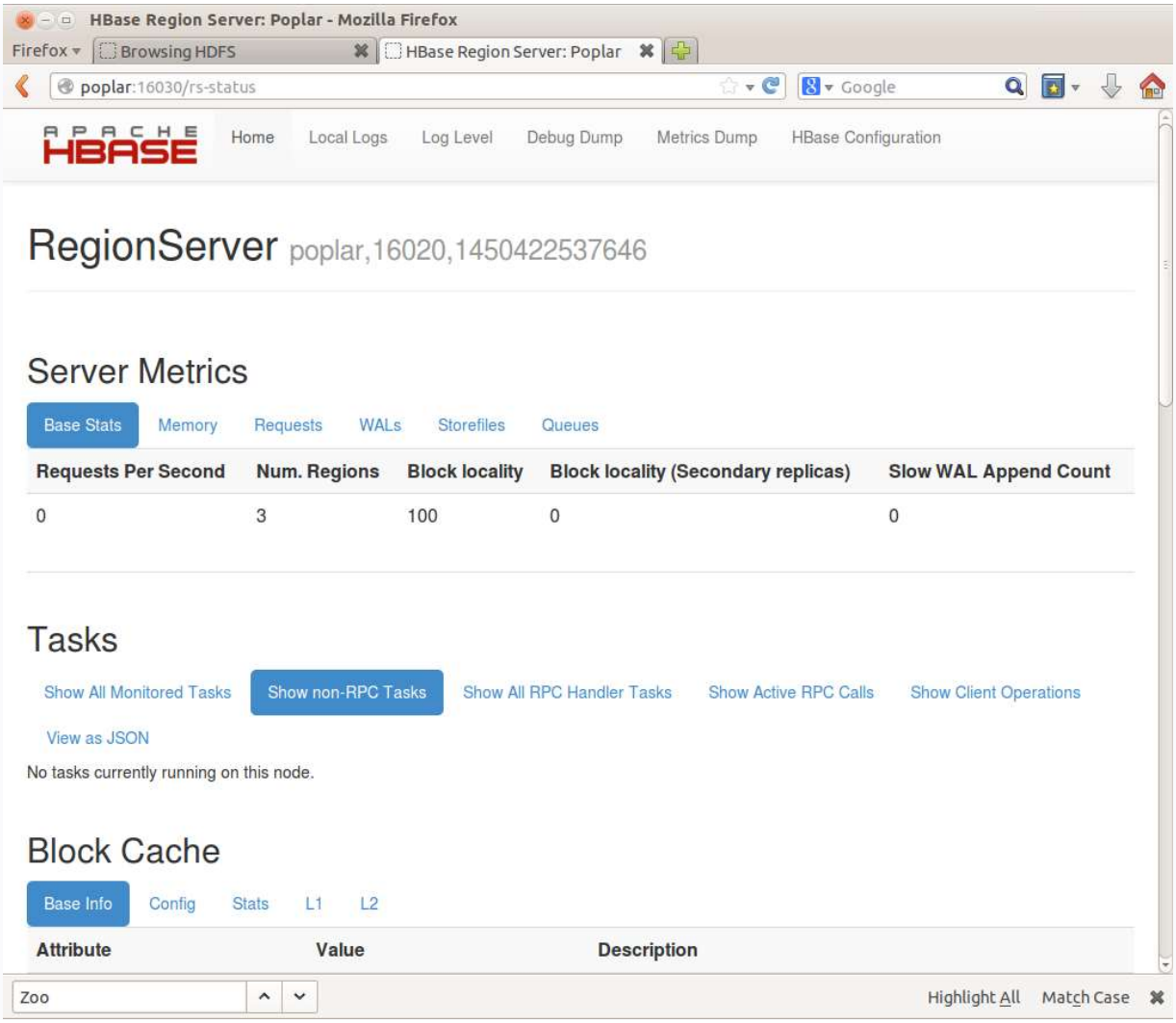
Compact — Region Key (optional): — This action will force a compaction of all regions of the table, or, if a key is supplied, only the region containing the given key.

Split — Region Key (optional): — This action will force a split of all eligible regions of the table, or, if a key is supplied, only the region containing the given key. An eligible region is one that does not contain any references to other regions. Split requests for noneligible regions will be ignored.

8.6 Region 服务器页面

通过 Master 页面中 Region 服务器信息提供的链接 http://poplar:16030/rs-status，可以进入 Region 服务器页面，该页面显示了 Region 服务器的基本属性和其上所有 Regions 的信息，如下图所示。

Figure 8-10 RegionServer 信息



9 HBASE 之完全分布式模式安装

本篇我们将详细介绍如何搭建 HBase 完全分布式环境，搭建 HBase 完全分布式环境的前提是我们已经搭建好了 Hadoop 完全分布式环境，搭建 Hadoop 完全分布式环境请参考：HADOOP 集群安装。

9.1 HBase 集群分布表

Hadoop 完全分布式环境和 HBase 完全分布式集群分别搭建成功后，Hadoop 集群中每个节点的角色如下表所示：

Table 9-1 Hadoop 集群节点角色

主机名	角色	IP 地址	jps 命令结果	HBase 用户属组	安装目录
Poplar	namenode	192.168.42.121	NameNode ResourceManager SecondaryNameNode **HMaster**	hadoop:hadoop	/opt/hbase
POPSlave1	datanode	192.168.42.122	DataNode NodeManager **HRegionServer** **HQuorumPeer**		
POPSlave2	datanode	192.168.42.123	DataNode NodeManager **HRegionServer** **HQuorumPeer**		
POPSlave3	datanode	192.168.42.124	DataNode NodeManager **HRegionServer** **HQuorumPeer**		

9.2 HBase 集群安装

参照 HBase 单机模式和 HBase 伪分布式模式完成集群中所有机器 HBase 的安装。

9.3 配置 hbase-env.sh

编辑集群中所有机器的 conf/hbase-env.sh，如下：

```
export JAVA_HOME=/opt/jdk
export HBASE_CLASSPATH=/opt/hbase/conf
export HBASE_MANAGES_ZK=false
```

9.4 配置 hbase-site.xml

编辑所有机器上的 hbase-site.xml 文件，命令如下：

```
[hadoop@Poplar:/opt/hbase]$ vi /opt/hbase/conf/hbase-site.xml
<configuration>
    <property>
        <name>hbase.rootdir</name>
        <value>hdfs://Poplar:9000/hbase</value>
    </property>
    <property>
        <name>hbase.cluster.distributed</name>
        <value>true</value>
    </property>
    <property>
        <name>hbase.zookeeper.quorum</name>
        <value>POPSlave1,POPSlave2,POPSlave3</value>
    </property>
    <property>
        <name>hbase.tmp.dir</name>
        <value>file:/home/hadoop/hadoop/hbase/tmp</value>
    </property>
    <property>
        <name>hbase.master</name>
        <value>hdfs://Poplar:60000</value>
    </property>
    <property>
        <name>hbase.zookeeper.property.dataDir</name>
        <value>file:/home/hadoop/hadoop/zookeeper</value>
    </property>
</configuration>
```

hbase-site.xml 配置文件中属性详细说明如下表所示：

Table 9-2 完全分布式配置属性说明

属性名	说明
hbase.rootdir	指定 Hbase 数据存储目录
hbase.cluster.distributed	true，指定为完全分布式模式
hbase.master	指定 Master 位置
hbase.zookeeper.quorum	指定 zookeeper 集群，多台机器用逗号分隔

特别注意：

- hbase.rootdir 属性值 HDFS 路径必须与你的 Hadoop 集群的 core-site.xml 文件配置保持完全一致；
- hbase.zookeeper.quorum 的个数必须是奇数。
- hbase.rootdir 默认为/tmp/hbase-${user.name},这意味着每次重启系统都会丢失数据。

9.5 配置 regionservers

编辑所有 HRegionServers 节点的 regionservers 文件。修改/opt/hbase/conf 文件夹下的 regionservers 文件，添加 DataNode 节点的 hostname，命令如下：

```
[hadoop@Poplar:/opt/hbase]$ vi /opt/hbase/conf/regionservers
POPSlave1
```

```
POPSlave2
POPSlave3
```

9.6 分发到其它的机器

将配置的 Hbase 安装文件分发到其他机器，只要将文件拷贝到对应目录即可：

```
scp -r /opt/hadoop/hbase hadoop@POPSlave1:/opt/hadoop/
scp -r /opt/hadoop/hbase hadoop@POPSlave2:/opt/hadoop/
scp -r /opt/hadoop/hbase hadoop@POPSlave3:/opt/hadoop/
```

9.7 启动 HBase

集群中所有节点完成上述 HBase 部署之后，即可启动 HBase 集群。启动顺序：hadoop→HBase，如果使用自己安装的 zookeeper 启动顺序是：hadoop→zookeeper→HBase 停止顺序：hbase→zookeeper→hadoop。

```
[hadoop@Poplar lib]$ start-hbase.sh          #启动 HBase
#查看 Poplar 机器运行进程
[hadoop@Poplar ~]$ jps
24330 HMaster
4726 NameNode
4880 SecondaryNameNode
4998 ResourceManager
9628 RunJar
24476 Jps
#查看 POPSlave1 机器运行进程
[hadoop@POPSlave1 usr]$ jps
10712 Jps
1429 DataNode
1506 NodeManager
10573 HQuorumPeer
10642 HRegionServer
#查看 POPSlave2 机器运行进程
[hadoop@POPSlave2 usr]$ jps
9955 HRegionServer
1409 DataNode
9888 HQuorumPeer
1484 NodeManager
10018 Jps
#查看 POPSlave3 机器运行进程
[hadoop@POPSlave3 usr]$ jps
11790 HRegionServer
1411 DataNode
1487 NodeManager
11873 Jps
11723 HQuorumPeer
```

10 HBASE SHELL DDL 操作

DDL(Data Definition Language)是数据库模式定义语言，是用于描述数据库中要存储的现实世界实体的语言，本节内容将执行关于 HBase 的 DDL 操作，包括：数据库表的建立、查看所有表、查表结构、删除列族、删除表等操作。

10.1 一般操作

本小节的所有操作均是在 HBase 伪分布式配置模式下运行的，故需先运行 Hadoop 集群（如果已启动则不需再启动），再运行 HBase，最后进入 HBase Shell 模式。

1) 准备工作

```
#启动 Hadoop 集群
[hadoop@Poplar:/opt/hbase]$start-all.sh        #启动 hadoop
[hadoop@Poplar:/opt/hbase]$jps                 #查看进程
#启动 HBase
[hadoop@Poplar:/opt/hbase]$ start-hbase.sh     #启动 HBase
[hadoop@Poplar:/opt/hbase]$jps                 #查看进程
#进入 shell 模式
[hadoop@Poplar:/opt/hbase]$ hbase shell
```

2) 查看 HBase 服务器状态信息

```
HBase(main):002:0> status
1 servers, 0 dead, 3.0000 average load
```

3) 查看 HBase 版本信息

```
HBase(main):002:0> version
0.94.20, r09c60d770f2869ca315910ba0f9a5ee9797b1edc, Fri May 23 22:00:41 PDT 2014
```

10.2 DDL 操作

使用个人信息为列演示 HBase 的用法。创建一个 user 表，其结构如表所示。

Table 10-1 User 表结构

Row Key	address			info		
	contry	province	city	age	birthday	company
andy	China	Guangdong	Guangzhou	37	1979-08-14	Poplar
…						
…						

这里 address 和 info 对于表来说是一个有三个列的列族：address 列族由三个列 contry、province 和 city 组成；info 列族由三个列 age、birthday 和 company 组成。当然可以根据需要在 address 和 info 中建立更多的列族，如 name、telephone 等相应的列族加入 info 列族。

1) 创建一个表 member

user 是表的名字，'user_id','address','info'分别为 user 表的三个列族。

```
HBase(main):002:0> create 'user','user_id','address','info'
0 row(s) in 1.4270 seconds
```

2) 查看所有表

```
HBase(main):002:0> list
TABLE
test
user
wordcount
3 row(s) in 0.0950 seconds
```

3) 查看表结构

```
HBase(main):002:0> describe 'user'
DESCRIPTION  ENABLED
 'user', {NAME => 'address', DATA_BLOCK_ENCODING =>  true
 'NONE', BLOOMFILTER => 'NONE', REPLICATION_SCOPE =>
  '0', VERSIONS => '3', COMPRESSION => 'NONE', MIN_V
 ERSIONS => '0', TTL => '2147483647', KEEP_DELETED_C
 ELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY =>
  'false', ENCODE_ON_DISK => 'true', BLOCKCACHE => '
 true'}, {NAME => 'info', DATA_BLOCK_ENCODING => 'NO
 NE', BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0
 ', VERSIONS => '3', COMPRESSION => 'NONE', MIN_VERS
 IONS => '0', TTL => '2147483647', KEEP_DELETED_CELL
 S => 'false', BLOCKSIZE => '65536', IN_MEMORY => 'f
 alse', ENCODE_ON_DISK => 'true', BLOCKCACHE => 'tru
 e'}, {NAME => 'user_id', DATA_BLOCK_ENCODING => 'NO
 NE', BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0
 ', VERSIONS => '3', COMPRESSION => 'NONE', MIN_VERS
 IONS => '0', TTL => '2147483647', KEEP_DELETED_CELL
 S => 'false', BLOCKSIZE => '65536', IN_MEMORY => 'f
 alse', ENCODE_ON_DISK => 'true', BLOCKCACHE => 'tru
 e'}
1 row(s) in 0.0950 seconds
```

4) 删除一个列族。

删除一个列族分为三步，第一步 disable 表，第二步 alter 表，第三步 enable 表；在第一步创建 user 表时创建了三个列族，但是发现 user_id 这个列族是多余的，现在需要将其删除，操作如下：

第一步：disable 表

```
HBase(main):008:0> disable 'user'
row(s) in 1.3790 seconds
```

第二步：alter 表

```
HBase(main):010:0> alter 'user',{NAME=>'user_id',METHOD=>'delete'}
Updating all regions with the new schema...
1/1 regions updated.
Done.
0 row(s) in 1.3660 seconds
HBase(main):010:0> describe 'user'
DESCRIPTION  ENABLED
 'user', {NAME => 'address', DATA_BLOCK_ENCODING =>  false
 'NONE', BLOOMFILTER => 'NONE', REPLICATION_SCOPE =>
  '0', VERSIONS => '3', COMPRESSION => 'NONE', MIN_V
 ERSIONS => '0', TTL => '2147483647', KEEP_DELETED_C
 ELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY =>
  'false', ENCODE_ON_DISK => 'true', BLOCKCACHE => '
 true'}, {NAME => 'info', DATA_BLOCK_ENCODING => 'NO
 NE', BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0
 ', VERSIONS => '3', COMPRESSION => 'NONE', MIN_VERS
 IONS => '0', TTL => '2147483647', KEEP_DELETED_CELL
 S => 'false', BLOCKSIZE => '65536', IN_MEMORY => 'f
 alse', ENCODE_ON_DISK => 'true', BLOCKCACHE => 'tru
 e'}
1 row(s) in 0.1050 seconds
```

第三步：enable 表

```
HBase(main):010:0> enable 'user'
0 row(s) in 1.3040 seconds
```

5) 删除表

删除一个列族分为两步，第一步 disable 表，第二步 drop 表；

第一步：disable 表

```
HBase(main):010:0> list
TABLE
test
user
wordcount
HBase(main):015:0> disable 'test'
0 row(s) in 1.3530 seconds
```

第二步：drop 表

```
HBase(main):016:0> drop 'test'
0 row(s) in 1.5380 seconds
```

6) 查询表是否存在

```
HBase(main):017:0> exists 'user'
Table user does exist
0 row(s) in 0.3530 seconds
```

7) 判断表是否 enable

```
HBase(main):018:0> is_enabled 'user'
true
```

```
0 row(s) in 0.0750 seconds
```

8) 判断表是否 disable

```
HBase(main):019:0> is_disabled 'user'
false
0 row(s) in 0.0600 seconds
```

11 HBASE SHELL DML 操作

DML（Data Manipulation Language）是数据操纵语言，用户通过它可以实现对数据库的基本操作。例如，对表中数据的查询、插入、删除和修改。 在 DML 中，应用程序可以对数据库作插，删，改，排，检等五种操作。本节将针对 HBase 数据库执行如下 DML 操作，包括：添加记录、查看记录、查看表中的记录总数，删除记录、删除一张表、查看某个列族的所有记录等。HBase Shell 基本操作命令如表所示：

Table 11-1 HBase Shell 基本操作命令

命令	命令表达式
创建表	create ‘表名称’,’列名称 1’,’列名称 2’,’列名称 N’
添加记录	put ‘表名称’,’行名称’,’列名称’,’值’
查看记录	get ‘表名称’,’行名称’
查看表中的记录总数	count ‘表名称’
删除记录	delete ‘表名称’,’行名称’,’列名称’
删除一张表	先要屏蔽该表，才能对该表删除，第一步： disable ‘表名称’ 第二步： drop ‘表名称’
查看所有记录	scan ‘表名称’
查看某个表某个列中所有数据	scan ‘表名称’, [‘列名称’]
更新记录	重写一遍进行覆盖

11.1 向表 user 插入记录

1) 向 user 表的行键 andieguo 的 info 列族成员：age、birthday、compay 分别添加数据

语法：put <table>,<rowkey>,<family:column>,<value>,<timestamp>

例如：给表 user 的添加一行记录：<rowkey>是'andieguo'，<family:column>是'info:age'，value 是'27'，timestamp：系统默认

```
HBase(main):021:0> put 'user','andieguo','info:age','27'
HBase(main):022:0> put 'user','andieguo','info:birthday','1989-09-01'
HBase(main):026:0> put 'user','andieguo','info:company','zonesion'
```

2) 向 user 表的行键 andieguo 的 address 列族成员：contry、province、city 分别添加数据

语法：put <table>,<rowkey>,<family:column>,<value>,<timestamp>

例如：给表 user 的添加一行记录：<rowkey>是'andieguo'，<family:column>是 'address:contry',value 是'china'，timestamp：系统默认

```
HBase(main):028:0> put 'user','andieguo','address:contry','china'
HBase(main):029:0> put 'user','andieguo','address:province','wuhan'
HBase(main):030:0> put 'user','andieguo','address:city','wuhan'
```

11.2 获取一条记录

1) 获取一个 ID 的所有记录

语法：get <table>,<rowkey>,[<family:column>,....]

例如：查询<table>为'user'，<rowkey>为'andieguo'下的所有记录

```
HBase(main):031:0> get 'user','andieguo'
COLUMNCELL
 address:city timestamp=1409303693005, value=wuhan
 address:contry   timestamp=1409303656326, value=china
 address:province timestamp=1409303678219, value=wuhan
 info:age timestamp=1409303518077, value=27
 info:birthdaytimestamp=1409303557859, value=1989-09-01
 info:company timestamp=1409303628168, value=zonesion
6 row(s) in 0.0350 seconds
```

2) 获取一个 ID 的一个列族的所有数据

语法：get <table>,<rowkey>,[<family:column>,....]

例如：查询<table>为'user'，<rowkey>为'andieguo', <family>为'info'下的所有记录

```
HBase(main):032:0> get 'user','andieguo','info'
COLUMNCELL
 info:age timestamp=1409303518077, value=27
 info:birthdaytimestamp=1409303557859, value=1989-09-01
 info:company timestamp=1409303628168, value=zonesion
3 row(s) in 0.0200 seconds
```

3) 获取一个 ID 的一个列族中的一个列的所有数据

语法：get <table>,<rowkey>,[<family:column>,....]

例如：查询<table>为'user'，<rowkey>为'andieguo', <family:column >为'info:age'下的所有记录

```
HBase(main):034:0> get 'user','andieguo','info:age'
COLUMNCELL
 info:age timestamp=1409303518077, value=27
1 row(s) in 0.0240 seconds
```

11.3 更新一条记录

将 andieguo 的年龄修改为 28，命令如下：

```
HBase(main):035:0> put 'user','andieguo','info:age','28'
0 row(s) in 0.0090 seconds

HBase(main):036:0> get 'user','andieguo','info:age'
COLUMNCELL
 info:age timestamp=1409304167955, value=28
1 row(s) in 0.0160 seconds
```

11.4 获取指定版本的数据

```
HBase(main)::037:0> get 'user','andieguo',{COLUMN=>'info:age',TIMESTAMP=>1409304}
COLUMNCELL
 info:age timestamp=1409304167955, value=28
1 row(s) in 0.0090 seconds
```

11.5 全表扫描

```
HBase(main):042:0> scan 'user'
ROWCOLUMN+CELL
 andieguo  column=address:city, timestamp=1409303693005,value=wuhan
 andieguo  column=address:contry, timestamp=1409303656326,value=china
 andieguo  column=address:province, timestamp=1409303678219,value=wuhan
 andieguo  column=info:age, timestamp=1409304167955,value=28
 andieguo  column=info:birthday, timestamp=1409303557859,value=1989-09-01
 andieguo  column=info:company, timestamp=1409303628168,value=zonesion
1 row(s) in 0.0340 seconds
```

11.6 删除 ID 为”andieguo”的列为’info:age’字段

```
HBase(main):043:0> delete 'user','andieguo','info:age'
0 row(s) in 0.0200 seconds

HBase(main):044:0> get 'user','andieguo'
COLUMN CELL
 address:city  timestamp=1409303693005,value=wuhan
 address:contrytimestamp=1409303656326,value=china
 address:province  timestamp=1409303678219,value=wuhan
 info:birthday timestamp=1409303557859,value=1989-09-01
 info:company  timestamp=1409303628168,value=zonesion
5 row(s) in 0.0180 seconds
```

11.7 查询表中有多少行

```
HBase(main):045:0> count 'user'
1 row(s) in 0.0770 seconds
```

11.8 向 ID 为”andieguo”添加’info:age’字段

1) 第一次添加(默认使用 counter 实现递增）

```
HBase(main):048:0> incr 'user','andieguo','info:age'
COUNTER VALUE = 1

HBase(main):052:0> get 'user','andieguo','info:age'
COLUMN CELL
 info:age  timestamp=1409304832249, value=\x00\x00\x00\x00\x00\x00\x00\x01
1 row(s) in 0.0150 seconds
```

2) 第二次添加(默认使用 counter 实现递增）

```
HBase(main):050:0> incr 'user','andieguo','info:age'
COUNTER VALUE = 2

HBase(main):052:0> get 'user','andieguo','info:age'
```

```
COLUMN CELL
 info:age  timestamp=1409304832249, value=\x00\x00\x00\x00\x00\x00\x00\x02
1 row(s) in 0.0150 seconds
```

3) 获取当前的 COUNTER 值

```
HBase(main):053:0> get_counter 'user','andieguo','info:age'
COUNTER VALUE = 2
```

11.9 将表数据清空

```
HBase(main):054:0> truncate 'user'
Truncating 'user' table (it may take a while):
 - Disabling table...
 - Dropping table...
 - Creating table...
0 row(s) in 3.5320 seconds
```

可以看出，HBase 在执行 truncate 命令时，通过先对表执行 disable，再执行 drop 操作，最后执行重新建表来实现数据清空。

12 HBASE API 访问

12.1 HBase API 介绍

12.1.1 几个相关类与 HBase 数据模型之间的对应关系

HBaseAdmin 用于数据库的创建与删除，HBaseConfiguration 用于数据库的配置，Htable 用数据库表的相关操作，HtableDescriptor 用于数据库表列族的相关操作，Put 用于数据库表记录的添加，Get 用户数据库表记录的获取，Scanner 用于数据库表全表查询。

Table 12-1 类与 Hbase 数据模型关系

java 类	HBase 数据模型
HbaseAdmin	数据库（DataBase）
HbaseConfiguration	
Htable	表（Table）
HtableDescriptor	列族（Column Family）
Put	列修饰符（Column Qualifier）
Get	
Scanner	

12.1.2 HBaseConfiguration

类名：org.apache.hadoop.hbase.HBaseConfiguration

作用：对 HBase 进行配置

常用方法：void set(String name, String value)，通过属性名来设置值。

用法示例：

```
Configuration conf = HBaseConfiguration.create();
conf.set("hbase.zookeeper.quorum", "Master");
```

12.1.3 HBaseAdmin

类名：org.apache.hadoop.hbase.client.HBaseAdmin

作用：提供了一个接口来管理 HBase 数据库的表信息。它提供的方法包括：创建表，删除表，列出表项，使表有效或无效，以及添加或删除表列族成员等。

Table 12-2 HBaseAdmin 类方法

返回值	函数	描述
void	addColumn(String tableName, HColumnDescriptor column)	向一个已经存在的表添加列
	createTable(HTableDescriptor desc)	创建一个表，同步操作
	deleteTable(byte[] tableName)	删除一个已经存在的表
	enableTable(byte[] tableName)	使表处于有效状态
	disableTable(byte[] tableName)	使表处于无效状态
HTableDescriptor[]	listTables()	列出所有用户控件表项
boolean	tableExists(String tableName)	检查表是否存在

12.1.4 HTableDescriptor

类名：org.apache.hadoop.hbase.HTableDescriptor

作用：包含了表的名字极其对应表的列族。

常用方法：void addFamily(HcolumnDescriptor family) 添加一个列族。其详细用法如下所示，向 tb_user 表中添加了一个 content 列族。

```
HTableDescriptor tableDescriptor = new HTableDescriptor("tb_user");
HColumnDescriptor col = new HColumnDescriptor("content:");
tableDescriptor.addFamily(col);
```

12.1.5 HColumnDescriptor

类名：org.apache.hadoop.hbase.HColumnDescriptor

作用：维护着关于列族的信息，例如版本号，压缩设置等。它通常在创建表或者为表添加列族的时候使用。列族被创建后不能直接修改，只能通过删除然后重新创建的方式。列族被删除的时候，列族里面的数据也会同时被删除。

12.1.6 HTable

类名：org.apache.hadoop.hbase.client.HTable

作用：可以用来和 HBase 表直接通信。此方法对于更新操作来说是非线程安全的。

用法示例：

```
HTable table = null;
ResultScanner rs = null;
try {
    Scan scan = new Scan();
    table = new HTable(conf, tableName);
    rs = table.getScanner(scan);
} catch (IOException e) {
    e.printStackTrace();
```

```
}
```

Table 12-3 HTable 类方法

返回值	函数	描述
void	close()	释放所有的资源或挂起内部缓冲区中的更新
Boolean	exists(Get get)	检查 Get 实例所指定的值是否存在于 HTable 的列中
Result	get(Get get)	获取指定行的某些单元格所对应的值
ResultScanner	getScanner(byte[] family)	获取当前给定列族的 scanner 实例
HTableDescriptor	getTableDescriptor()	获取当前表的 HTableDescriptor 实例
byte[]	getTableName()	获取表名
static boolean	isTableEnabled(HBaseConfiguration conf, String tableName)	检查表是否有效
void	put(Put put)	向表中添加值

12.1.7 Put

类名：org.apache.hadoop.hbase.client.Put

作用：用来对单个行执行添加操作

用法示例：

```
HTable table = null;
try {
    table = new HTable(conf, tabelName);
    Put putRow1 = new Put(rowKey.getBytes());
    putRow1.add(family.getBytes(), qualifier.getBytes(),value.getBytes());
    table.put(putRow1);
    table.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

Table 12-4 Put 类方法

返回值	函数	描述
Put	add(byte[] family, byte[] qualifier, byte[] value)	将指定的列和对应的值添加到 Put 实例中
Put	add(byte[] family, byte[] qualifier, long ts, byte[] value)	将指定的列和对应的值及时间戳添加到 Put 实例中

12.1.8 Get

类名：org.apache.hadoop.hbase.client.Get

作用：用来获取单个行的相关信息

用法示例：

```
Get query = new Get(rowKey.getBytes());
query.addColumn(Bytes.toBytes(family), Bytes.toBytes(qualifier));
if(table.exists(query)){
    Result result = table.get(query);
    System.out.format("ROW\t%s\n",new String(result.getRow()));
    for(KeyValue kv : result.raw()){
        System.out.format("COLUMN\t %S:%s\t%s\n",new String(kv.getFamily()),new
String(kv.getQualifier()),new String(kv.getValue()));
    }
}
```

Table 12-5 Get 类方法

返回值	函数	描述
Get	addColumn(byte[] family, byte[] qualifier)	获取指定列族和列修饰符对应的列
Get	addFamily(byte[] family)	通过指定的列族获取其对应列的所有列
Get	setTimeRange(long minStamp,long maxStamp)	获取指定取件的列的版本号
Get	setFilter(Filter filter)	当执行 Get 操作时设置服务器端的过滤器

12.2 HBase API 实战

12.2.1 创建表

create(String tableName,String… families)实现了创建名为 tablename 的表，同时添加若干 families 列族，列族个数不定，详细源码请参考：HBaseAPI/src/com/zonesion/hbase/CreateTable.java。

```
public static void create(String tableName,String... families) {
    Configuration conf = HBaseConfiguration.create();
    conf.set("hbase.zookeeper.quorum", "Master");
    HBaseAdmin admin = null;
    try {
        admin = new HBaseAdmin(conf);
        HTableDescriptor tableDescriptor = new HTableDescriptor(
                tableName.getBytes());
        if (admin.tableExists(tableName)) {// 表存在
            System.out.println(tableName + "已经存在！");
        } else {
            for(String family :families){
                tableDescriptor.addFamily(new HColumnDescriptor(family));
            }
            admin.createTable(tableDescriptor);
```

```
        }
    } catch (MasterNotRunningException e) {
        e.printStackTrace();
    }
}
```

12.2.2 添加记录

put(String tabelName,String rowKey,String family,String qualifier,String value) 实现了向 tablename 表主键为 rowkey、列族为 family、列为 qualifier 添加值为 value 的记录，详细源码请参考：HBaseAPI/src/com/zonesion/hbase/PutRow.java。

```
public static void put(String tabelName,String rowKey,String family,String
qualifier,String value) {
    Configuration conf = HBaseConfiguration.create();
    conf.set("hbase.zookeeper.quorum", "Master");
    HTable table = null;
    try {
        table = new HTable(conf, tabelName);
        Put putRow1 = new Put(rowKey.getBytes());
        putRow1.add(family.getBytes(), qualifier.getBytes(),value.getBytes());
        table.put(putRow1);
        table.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

12.2.3 获取记录

getRow(String tableName,String rowKey) 实现了获取 tablename 表主键为 rowkey 的所有记录，详细源码请参考：HBaseAPI/src/com/zonesion/hbase/GetRow.java。

```
public static void getRow(String tableName,String rowKey) {
    Configuration conf = HBaseConfiguration.create();
    conf.set("hbase.zookeeper.quorum", "Master");
    HTable table = null;
    try {
        table = new HTable(conf, tableName);
        Get query = new Get(rowKey.getBytes());
        if(table.exists(query)){
            Result result = table.get(query);
            System.out.format("ROW\t%s\n",new String(result.getRow()));
            for(KeyValue kv : result.raw()){
                System.out.format("COLUMN\t %S:%s\t%s\n",new
String(kv.getFamily()),new String(kv.getQualifier()),new String(kv.getValue()));
            }
        }
        table.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

12.2.4 遍历表

list(String tableName)实现了遍历表 tableName，并打印所有遍历的记录，详细源码请参考：HBaseAPI/src/com/zonesion/hbase/GetScanner.java。

```
public static void list(String tableName) {
    Configuration conf = HBaseConfiguration.create();
    conf.set("hbase.zookeeper.quorum", "Master");
    HTable table = null;
    ResultScanner rs = null;
    try {
        Scan scan = new Scan();
        table = new HTable(conf, tableName);
        rs = table.getScanner(scan);
        for(Result row : rs){
            System.out.format("ROW\t%s\n",new String(row.getRow()));
            for(Map.Entry<byte[], byte[]> entry :
row.getFamilyMap("info".getBytes()).entrySet()){
                String column = new String(entry.getKey());
                String value = new String(entry.getValue());
                System.out.format("COLUMN\t info:%s\t%s\n",column,value);
            }
            for(Map.Entry<byte[], byte[]> entry :
row.getFamilyMap("address".getBytes()).entrySet()){
                String column = new String(entry.getKey());
                String value = new String(entry.getValue());
                System.out.format("COLUMN\t address:%s\t%s\n",column,value);
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

更多详细的用法请参考：HBaseAPI/src/com/zonesion/HBase 目录，其中 CreateTable.java 用于创建表，DeleteTable.java 用于删除表，FilterQuery.java 用于过滤查询，GetColumn.java 用于获取指定列记录，GetFamily.java 用于获取指定列族的所有记录，GetRow.java 用于获取指定 rowkey 的所有记录，GetScanner.java 用于遍历表，PutRow.java 用于添加记录，HBaseAPI.java 整合了上述所有资源以命令的形式进行调用，在实验过程中我们使用的 HBaseAPI.java 命令调用方式，用户也可单独执行以上所有资源。

12.3 部署运行

12.3.1 启动 Hadoop 集群和 HBase 服务

```
[hadoop@Poplar ~]$ start-dfs.sh      #启动 hadoop HDFS 文件管理系统
[hadoop@Poplar ~]$ start-mapred.sh       #启动 hadoop MapReduce 分布式计算服务
[hadoop@Poplar ~]$ start-hbase.sh        #启动 HBase
[hadoop@Poplar ~]$ jps               #查看进程
22003 HMaster
10611 SecondaryNameNode
22226 Jps
21938 HQuorumPeer
10709 ResourceManager
```

```
22154 HRegionServer
20277 Main
10432 NameNode
```

特别注意：用户可先通过 jps 命令查看 Hadoop 集群和 HBase 服务是否启动，如果 Hadoop 集群和 HBase 服务已经启动，则不需要执行此操作。

12.3.2 部署源码

```
#设置工作环境
[hadoop@Poplar ~]$ mkdir -p /usr/hadoop/workspace/HBase
#部署源码
将 HBaseAPI 文件夹拷贝到/usr/hadoop/workspace/hbase/ 路径下；
```

12.3.3 修改配置文件

1) 查看 HBase 核心配置文件 hbase-site.xml 的 hbase.zookeeper.quorum 属性

使用如下命令查看 hadoop 核心配置文件 hbase-site.xml 的 hbase.zookeeper.quorum 属性值，当前 Poplar 服务器的 hbase.zookeeper.quorum 属性值为 Poplar；

```
[hadoop@Poplar ~]$ cat /usr/hbase/conf/hbase-site.xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<!-- Put site-specific property overrides in this file. -->

<configuration>
   <!--zookeeper 运行的机器，要为奇数个，使得投票更公平-->
<property>
<name>hbase.zookeeper.quorum</name>
<value>Poplar</value>
</property>
</configuration>
```

2) 修改项目 HBaseAPI/src/config.properties 属性文件

将项目 HBaseAPI/src/config.properties 属性文件的 hbase.zookeeper.quorum 属性值修改为上一步查询到的属性值，保持 config.properties 文件的 hbase.zookeeper.quorum 属性值与 hbase-site.xml 文件的 hbase.zookeeper.quorum 属性值一致；

```
#切换工作目录
[hadoop@Poplar ~]$ cd /usr/hadoop/workspace/hbase/HBaseAPI/
#修改属性值
[hadoop@Poplar HBaseAPI]$ vim src/config.properties
hbase.zookeeper.quorum=Poplar
#拷贝 src/config.properties 到 bin 文件夹
[hadoop@Poplar HBaseAPI]$ cp src/config.properties bin/
```

12.3.4 编译文件

```
#执行编译
[hadoop@Poplar HBaseAPI]$ javac -classpath /usr/hadoop/hadoop-core-
1.2.1.jar:/usr/hadoop/lib/commons-cli-1.2.jar:lib/zookeeper-3.4.5.jar:lib/hbase-
0.94.20.jar -d bin/ src/com/zonesion/hbase/*.java
#查看编译是否成功
[hadoop@Poplar HBaseAPI]$ ls bin/com/zonesion/hbase/ -la
total 52
drwxrwxr-x 2 hadoop hadoop 4096 Dec 30 16:18 .
drwxrwxr-x 3 hadoop hadoop 4096 Dec 30 16:18 ..
-rw-rw-r-- 1 hadoop hadoop 3283 Dec 30 16:18 CreateTable.class
-rw-rw-r-- 1 hadoop hadoop 2702 Dec 30 16:18 DeleteTable.class
-rw-rw-r-- 1 hadoop hadoop 3781 Dec 30 16:18 FilterQuery.class
-rw-rw-r-- 1 hadoop hadoop 2870 Dec 30 16:18 GetColumn.class
-rw-rw-r-- 1 hadoop hadoop 2789 Dec 30 16:18 GetFamily.class
-rw-rw-r-- 1 hadoop hadoop 2511 Dec 30 16:18 GetRow.class
-rw-rw-r-- 1 hadoop hadoop 3519 Dec 30 16:18 GetScanner.class
-rw-rw-r-- 1 hadoop hadoop 3085 Dec 30 16:18 HBaseAPI.class
-rw-rw-r-- 1 hadoop hadoop 4480 Dec 30 16:18 PropertiesHelper.class
-rw-rw-r-- 1 hadoop hadoop 2148 Dec 30 16:18 PutRow.class
```

12.3.5 打包 Jar 文件

```
#拷贝 lib 文件夹到 bin 文件夹
[hadoop@Poplar HBaseAPI]$ cp -r lib/ bin/
#打包 Jar 文件
[hadoop@Poplar HBaseAPI]$ jar -cvf HBaseAPI.jar -C  bin/ .
added manifest
adding: lib/(in = 0) (out= 0)(stored 0%)
adding: lib/zookeeper-3.4.5.jar(in = 779974) (out= 721150)(deflated 7%)
adding: lib/protobuf-java-2.4.0a.jar(in = 449818) (out= 420864)(deflated 6%)
adding: lib/hbase-0.94.20.jar(in = 5475284) (out= 5038635)(deflated 7%)
adding: com/(in = 0) (out= 0)(stored 0%)
adding: com/zonesion/(in = 0) (out= 0)(stored 0%)
adding: com/zonesion/hbase/(in = 0) (out= 0)(stored 0%)
adding: com/zonesion/hbase/GetScanner.class(in = 2681) (out= 1422)(deflated 46%)
adding: com/zonesion/hbase/GetColumn.class(in = 2229) (out= 1127)(deflated 49%)
adding: com/zonesion/hbase/DeleteTable.class(in = 2058) (out= 1162)(deflated 43%)
adding: com/zonesion/hbase/CreateTable.class(in = 2534) (out= 1395)(deflated 44%)
adding: com/zonesion/hbase/HBaseAPI.class(in = 2947) (out= 1319)(deflated 55%)
adding: com/zonesion/hbase/FilterQuery.class(in = 3114) (out= 1540)(deflated 50%)
adding: com/zonesion/hbase/GetRow.class(in = 1914) (out= 1017)(deflated 46%)
adding: com/zonesion/hbase/PutRow.class(in = 1600) (out= 888)(deflated 44%)
adding: com/zonesion/hbase/GetFamily.class(in = 2170) (out= 1106)(deflated 49%)
```

12.3.6 运行实例

1) help 命令

```
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI
HBaseAPI action ...
    create <tableName> [family...]
    delete <tableName>
```

```
    put <tableName> <rowKey> <family> <column> <value>
    scan <tableName>
    get <tableName> <rowKey>
    get <tableName> <rowKey> <family>
    get <tableName> <rowKey> <family> <column>
```

2) create 命令

```
#查看 create 帮助命令
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI create
    create <tableName> [family...]
#创建 tb_admin 表，其中该表包含 info、address 两个列族
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI create
tb_admin info address
```

3) put 命令

```
#查看 put 帮助命令
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI put
    put <tableName> <rowKey> <family> <column> <value>
#使用 put 命令向表为 tb_admin、<rowkey>是'andieguo'添加一系列记录
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI put
tb_admin andieguo info age 25
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI put
tb_admin andieguo info birthday 1990-09-08
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI put
tb_admin andieguo info company zonesion
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI put
tb_admin andieguo address country China
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI put
tb_admin andieguo address province hubei
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI put
tb_admin andieguo address city wuhan
```

4) scan 命令

```
#查看 scan 帮助命令
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI scan
    scan <tableName>
#使用 scan 命令查看 tb_admin 表里的所有记录
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI scan
tb_admin
ROW andieguo
COLUMN   info:age   25
COLUMN   info:birthday  1990-09-08
COLUMN   info:company   zonesion
COLUMN   address:city   wuhan
COLUMN   address:country    China
COLUMN   address:province   hubei
```

5) get 命令

```
#查看 get 帮助命令
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI get
    get <tableName> <rowKey>
    get <tableName> <rowKey> <family>
    get <tableName> <rowKey> <family> <column>
#使用 get 命令查看表 tb_admin 中 rowkey 为 andieguo 的所有记录
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI get
tb_admin andieguo
ROW andieguo
COLUMN   ADDRESS:city   wuhan
COLUMN   ADDRESS:country    China
COLUMN   ADDRESS:province   hubei
COLUMN   INFO:age   25
COLUMN   INFO:birthday  1990-09-08
COLUMN   INFO:company   zonesion
#使用 get 命令查看表 tb_admin 中 rowkey 为 andieguo、列族为 info 的所有记录
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI get
tb_admin andieguo info
ROW andieguo
COLUMN   INFO:age   25
COLUMN   INFO:birthday  1990-09-08
COLUMN   INFO:company   zonesion
#使用 get 命令查看表 tb_admin 中 rowkey 为 andieguo、info 为 age 的所有记录
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI get
tb_admin andieguo info age
ROW andieguo
COLUMN   INFO:age   25
```

6) delete 命令

```
#查看 delete 帮助命令
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI delete
    delete <tableName>
#使用 delete 命令删除 tb_admin 表
[hadoop@Poplar HBaseAPI]$ hadoop jar HBaseAPI.jar com.zonesion.hbase.HBaseAPI delete
tb_admin
14/12/19 19:46:37 INFO client.HBaseAdmin: Disabled tb_admin
14/12/19 19:46:38 INFO client.HBaseAdmin: Deleted tb_admin
```

关闭 HBaseAdmin

13 HBASE 读取 MAPREDUCE 数据写入 HBASE

本文将介绍利用 MapReduce 操作 HBase，借助最熟悉的单词计数案例 WordCount，将 WordCount 的统计结果存储到 HBase，而不是 HDFS。

13.1 输入与输出

1) 输入文件

```
file0.txt ( WordCountHBaseWriter\input\file0.txt )
Hello World Bye World
file1.txt ( WordCountHBaseWriter\input\file1.txt )
Hello Hadoop Goodbye Hadoop
```

2) 输出 HBase 数据库

以下为输出数据库 wordcount 的数据库结构，以及预期的输出结果，如下图所示：

Table 13-1 WordCount 数据结构

Rowkey	时间戳	列族 content
Bye	T1	content:count=1
GoodBye	T1	content:count=1
Hadoop	T1	content:count=2
Hello	T1	content:count=2

13.2 Mapper 函数实现

WordCountHBaseMapper 程序和 WordCount 的 Map 程序一样，Map 输入为每一行数据，例如"Hello World Bye World"，通过 StringTokenizer 类按空格分割成一个个单词，通过 context.write(word, one);输出为一系列< key,value>键值对：

```
<"Hello",1><"World",1><"Bye",1><"World",1>。
```

源码请参考：

```
public static class WordCountHBaseMapper extends
        Mapper<Object, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);// 输出<key,value>为<word,one>
        }
```

```
    }
}
```

13.3 Reducer 函数实现

WordCountHBaseReducer 继承的是 TableReducer 类，在 Hadoop 中 TableReducer 继承 Reducer 类，它的原型为 TableReducer< KeyIn,Values,KeyOut>，前两个参数必须对应 Map 过程的输出类型 key/value 类型，第三个参数为 ImmutableBytesWritable，即为不可变类型。reduce(Text key, Iterable< IntWritable> values,Context context)具体处理过程分析如下表所示。

Table 13-2 reduce 过程

代码	分析说明

for (IntWritable val : values) {
sum += val.get();
}
	这三行实现了对各个单词累计求和；

Put put = new Put(key.getBytes()); put.add(Bytes.toBytes("content"), Bytes.toBytes("count"),, B ytes.toBytes(String.valueOf(sum)));	表示 put 实例化，每一个词存一行；
Bytes.toBytes(String.valueOf(sum)));	表示添加的内容为：列族为 content,列修饰符为 co unt，列值为数目。
context.write(new ImmutableBytesWritable(key.getBytes()), put)	表示将 put 记录写入到 Hbase 表中 rowkey 为 key 的主键中。

源码请参考：

```
public static class WordCountHBaseReducer extends
            TableReducer<Text, IntWritable, ImmutableBytesWritable> {

        public void reduce(Text key, Iterable<IntWritable> values,
                Context context) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {// 遍历求和
                sum += val.get();
            }
            Put put = new Put(key.getBytes());//put 实例化，每一个词存一行
            //列族为 content,列修饰符为 count，列值为数目
            put.add(Bytes.toBytes("content"), Bytes.toBytes("count"),
Bytes.toBytes(String.valueOf(sum)));
            context.write(new ImmutableBytesWritable(key.getBytes()), put);// 输出求
和后的<key,value>
        }
    }
```

13.4 驱动函数实现

与 WordCount 的驱动类不同，在 Job 配置的时候没有配置 job.setReduceClass()，而是用以下方法执行 Reduce 类：

```
TableMapReduceUtil.initTableReducerJob(tablename, WordCountHBaseReducer.class, job);
```

该方法指明了在执行 job 的 reduce 过程时，执行 WordCountHBaseReducer，并将 reduce 的结果写入到表明为 tablename 的表中。

特别注意：此处的 TableMapReduceUtil 是 hadoop.hbase.mapreduce 包中的，而不是 hadoop.hbase.mapred 包中的，否则会报错。

源码请参考：

```
public static void main(String[] args) throws Exception {
    String tablename = "wordcount";
    Configuration conf = HBaseConfiguration.create();
    conf.set("hbase.zookeeper.quorum", "Master");
    HBaseAdmin admin = new HBaseAdmin(conf);
    if(admin.tableExists(tablename)){
        System.out.println("table exists!recreating.......");
        admin.disableTable(tablename);
        admin.deleteTable(tablename);
    }
    HTableDescriptor htd = new HTableDescriptor(tablename);
    HColumnDescriptor tcd = new HColumnDescriptor("content");
    htd.addFamily(tcd);//创建列族
    admin.createTable(htd);//创建表
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 1) {
      System.err.println("Usage: WordCountHBase <in>");
      System.exit(2);
    }
    Job job = new Job(conf, "WordCountHBase");
    job.setJarByClass(WordCounthbase.class);
    //使用 WordCountHBaseMapper 类完成 Map 过程；
    job.setMapperClass(WordCountHBaseMapper.class);
    TableMapReduceUtil.initTableReducerJob(tablename, WordCountHBaseReducer.class,
job);
    //设置任务数据的输入路径；
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    //设置了 Map 过程和 Reduce 过程的输出类型，其中设置 key 的输出类型为 Text；
    job.setOutputKeyClass(Text.class);
    //设置了 Map 过程和 Reduce 过程的输出类型，其中设置 value 的输出类型为 IntWritable；
    job.setOutputValueClass(IntWritable.class);
    //调用 job.waitForCompletion(true) 执行任务，执行成功后退出；
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

13.5 部署运行

13.5.1 启动 Hadoop 集群和 HBase 服务

```
[hadoop@Poplar ~]$ start-dfs.sh      #启动 hadoop HDFS 文件管理系统
[hadoop@Poplar ~]$ start-mapred.sh   #启动 hadoop MapReduce 分布式计算服务
[hadoop@Poplar ~]$ start-hbase.sh    #启动 HBase
[hadoop@Poplar ~]$ jps               #查看进程
22003 HMaster
10611 SecondaryNameNode
22226 Jps
21938 HQuorumPeer
10709 ResourceManager
22154 HRegionServer
20277 Main
10432 NameNode
```

特别注意：用户可先通过 jps 命令查看 Hadoop 集群和 HBase 服务是否启动，如果 Hadoop 集群和 HBase 服务已经启动，则不需要执行此操作。

13.5.2 部署源码

```
#设置工作环境
[hadoop@Poplar ~]$ mkdir -p /usr/hadoop/workspace/HBase
#部署源码
#将 WordCountHBaseWriter 文件夹拷贝到/usr/hadoop/workspace/hbase/ 路径下；
```

13.5.3 修改配置文件

1) 查看 HBase 核心配置文件 hbase-site.xml 的 hbase.zookeeper.quorum 属性

参考 HBase API 访问章的部署运行节的修改配置文件小节的"查看 HBase 核心配置文件 hbase-site.xml 的 hbase.zookeeper.quorum 属性"。

2) 修改项目 WordCountHBaseWriter/src/config.properties 属性文件

将项目 WordCountHBaseWriter/src/config.properties 属性文件的 hbase.zookeeper.quorum 属性值修改为上一步查询到的属性值，保持 config.properties 文件的 hbase.zookeeper.quorum 属性值与 hbase-site.xml 文件的 hbase.zookeeper.quorum 属性值一致；

```
#切换工作目录
[hadoop@Poplar ~]$ cd /usr/hadoop/workspace/hbase/WordCountHBaseWriter
#修改属性值
[hadoop@Poplar WordCountHBaseWriter]$ vim src/config.properties
hbase.zookeeper.quorum=Poplar
#拷贝 src/config.properties 文件到 bin/文件夹
[hadoop@Poplar WordCountHBaseWriter]$ cp src/config.properties bin/
```

13.5.4 上传输入文件

```
#创建输入文件夹
[hadoop@Poplar WordCountHBaseWriter]$ hadoop fs -mkdir HBaseWriter/input/
#上传文件到输入文件夹
[hadoop@Poplar WordCountHBaseWriter]$ hadoop fs -put input/file* HBaseWriter/input/
#查看上传文件是否成功
[hadoop@Poplar WordCountHBaseWriter]$ hadoop fs -ls HBaseWriter/input/
Found 2 items
-rw-r--r--   3 hadoop supergroup 22 2014-12-30 17:39
/user/hadoop/HBaseWriter/input/file0.txt
-rw-r--r--   3 hadoop supergroup 28 2014-12-30 17:39
/user/hadoop/HBaseWriter/input/file1.txt
```

13.5.5 编译文件

```
#执行编译
[hadoop@Poplar WordCountHBaseWriter]$ javac -classpath /usr/hadoop/hadoop-core-
1.2.1.jar:/usr/hadoop/lib/commons-cli-1.2.jar:lib/zookeeper-3.4.5.jar:lib/hbase-
0.94.20.jar -d bin/ src/com/zonesion/hbase/*.java
#查看编译是否成功
[hadoop@Poplar WordCountHBaseWriter]$ ls bin/com/zonesion/hbase/ -la
total 24
drwxrwxr-x 2 hadoop hadoop 4096 Dec 30 17:20 .
drwxrwxr-x 3 hadoop hadoop 4096 Dec 30 17:20 ..
-rw-rw-r-- 1 hadoop hadoop 3446 Dec 30 17:29 PropertiesHelper.class
-rw-rw-r-- 1 hadoop hadoop 3346 Dec 30 17:29 WordCountHBaseWriter.class
-rw-rw-r-- 1 hadoop hadoop 1817 Dec 30 17:29
WordCountHBaseWriter$WordCountHBaseMapper.class
-rw-rw-r-- 1 hadoop hadoop 2217 Dec 30 17:29
WordCountHBaseWriter$WordCountHBaseReducer.class
```

13.5.6 打包 Jar 文件

```
#拷贝 lib 文件夹到 bin 文件夹
[hadoop@Poplar WordCountHBaseWriter]$ cp -r lib/ bin/
#打包 Jar 文件
[hadoop@Poplar WordCountHBaseWriter]$ jar -cvf WordCountHBaseWriter.jar -C bin/ .
added manifest
adding: lib/(in = 0) (out= 0)(stored 0%)
adding: lib/zookeeper-3.4.5.jar(in = 779974) (out= 721150)(deflated 7%)
adding: lib/guava-11.0.2.jar(in = 1648200) (out= 1465342)(deflated 11%)
adding: lib/protobuf-java-2.4.0a.jar(in = 449818) (out= 420864)(deflated 6%)
adding: lib/hbase-0.94.20.jar(in = 5475284) (out= 5038635)(deflated 7%)
adding: com/(in = 0) (out= 0)(stored 0%)
adding: com/zonesion/(in = 0) (out= 0)(stored 0%)
adding: com/zonesion/hbase/(in = 0) (out= 0)(stored 0%)
adding: com/zonesion/hbase/WordCountHBaseWriter.class(in = 3136) (out= 1583)(deflated
49%)
adding: com/zonesion/hbase/WordCountHBaseWriter$WordCountHBaseMapper.class(in = 1817)
(out= 772)(deflated 57%)
adding: com/zonesion/hbase/WordCountHBaseWriter$WordCountHBaseReducer.class(in =
2217) (out= 929)(deflated 58%)
```

13.5.7 运行实例

```
[hadoop@Poplar WordCountHBaseWriter]$ hadoop jar WordCountHBaseWriter.jar
com.zonesion.hbase.WordCountHBaseWriter /user/hadoop/HBaseWriter/input/
.....................省略.............
14/12/30 11:23:59 INFO input.FileInputFormat: Total input paths to process : 2
14/12/30 11:23:59 INFO util.NativeCodeLoader: Loaded the native-hadoop library
14/12/30 11:23:59 WARN snappy.LoadSnappy: Snappy native library not loaded
14/12/30 11:24:05 INFO mapred.JobClient: Running job: job_201412161748_0020
14/12/30 11:24:06 INFO mapred.JobClient:  map 0% reduce 0%
14/12/30 11:24:27 INFO mapred.JobClient:  map 50% reduce 0%
14/12/30 11:24:30 INFO mapred.JobClient:  map 100% reduce 0%
14/12/30 11:24:39 INFO mapred.JobClient:  map 100% reduce 100%
14/12/30 11:24:41 INFO mapred.JobClient: Job complete: job_201412161748_0020
14/12/30 11:24:41 INFO mapred.JobClient: Counters: 28
14/12/30 11:24:41 INFO mapred.JobClient:   Job Counters
14/12/30 11:24:41 INFO mapred.JobClient: Launched reduce tasks=1
14/12/30 11:24:41 INFO mapred.JobClient: SLOTS_MILLIS_MAPS=20955
14/12/30 11:24:41 INFO mapred.JobClient: Total time spent by all reduces waiting
after reserving slots (ms)=0
14/12/30 11:24:41 INFO mapred.JobClient: Total time spent by all maps waiting after
reserving slots (ms)=0
14/12/30 11:24:41 INFO mapred.JobClient: Launched map tasks=2
14/12/30 11:24:41 INFO mapred.JobClient: Data-local map tasks=2
14/12/30 11:24:41 INFO mapred.JobClient: SLOTS_MILLIS_REDUCES=11527
14/12/30 11:24:41 INFO mapred.JobClient:   File Output Format Counters
14/12/30 11:24:41 INFO mapred.JobClient: Bytes Written=0
14/12/30 11:24:41 INFO mapred.JobClient:   FileSystemCounters
14/12/30 11:24:41 INFO mapred.JobClient: FILE_BYTES_READ=104
14/12/30 11:24:41 INFO mapred.JobClient: HDFS_BYTES_READ=296
14/12/30 11:24:41 INFO mapred.JobClient: FILE_BYTES_WRITTEN=239816
14/12/30 11:24:41 INFO mapred.JobClient:   File Input Format Counters
14/12/30 11:24:41 INFO mapred.JobClient: Bytes Read=50
14/12/30 11:24:41 INFO mapred.JobClient:   Map-Reduce Framework
14/12/30 11:24:41 INFO mapred.JobClient: Map output materialized bytes=110
14/12/30 11:24:41 INFO mapred.JobClient: Map input records=2
14/12/30 11:24:41 INFO mapred.JobClient: Reduce shuffle bytes=110
14/12/30 11:24:41 INFO mapred.JobClient: Spilled Records=16
14/12/30 11:24:41 INFO mapred.JobClient: Map output bytes=82
14/12/30 11:24:41 INFO mapred.JobClient: Total committed heap usage (bytes)=417546240
14/12/30 11:24:41 INFO mapred.JobClient: CPU time spent (ms)=1110
14/12/30 11:24:41 INFO mapred.JobClient: Combine input records=0
14/12/30 11:24:41 INFO mapred.JobClient: SPLIT_RAW_BYTES=246
14/12/30 11:24:41 INFO mapred.JobClient: Reduce input records=8
14/12/30 11:24:41 INFO mapred.JobClient: Reduce input groups=5
14/12/30 11:24:41 INFO mapred.JobClient: Combine output records=0
14/12/30 11:24:41 INFO mapred.JobClient: Physical memory (bytes) snapshot=434167808
14/12/30 11:24:41 INFO mapred.JobClient: Reduce output records=5
14/12/30 11:24:41 INFO mapred.JobClient: Virtual memory (bytes) snapshot=2192027648
14/12/30 11:24:41 INFO mapred.JobClient: Map output records=8
```

13.5.8 查看输出结果

```
#另外开启一个终端，输入 HBase shell 命令进入 HBase shell 命令行
[hadoop@Poplar ~]$ hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
```

```
Version 0.94.20, r09c60d770f2869ca315910ba0f9a5ee9797b1edc, Fri May 23 22:00:41 PDT
2014

HBase(main):002:0> scan 'wordcount'
ROW   COLUMN+CELL
 Bye  column=content:count, timestamp=1419932527321, value=1
 Goodbye  column=content:count, timestamp=1419932527321, value=1
 Hadoope  column=content:count, timestamp=1419932527321, value=2
 Hellope  column=content:count, timestamp=1419932527321, value=2
 Worldpe  column=content:count, timestamp=1419932527321, value=2
5 row(s) in 0.6370 seconds
```

14 HBASE 读取 HBASE 数据写入 HDFS

介绍如何读取 HBase 中的数据并写入到 HDFS 分布式文件系统中。读取数据比较简单，我们借用上一篇 **HBase 读取 MapReduce 数据写入 HBase** 的 HBase 数据输出 wordcount 表作为本篇数据源的输入，编写 Mapper 函数，读取 wordcount 表中的数据填充到< key,value>，通过 Reduce 函数直接输出得到的结果即可。

14.1 输入与输出

■ 输入数据源

上一篇 HBase 读取 MapReduce 数据写入 HBase 实现了读取 MapReduce 数据写入到 HBase 表 wordcount 中，在本篇中，我们将 wordcount 表作为输入数据源。

■ 输出目标

HDFS 分布式文件系统中的文件。

14.2 Mapper 函数实现

WordCountHBaseReaderMapper 类继承了 TableMapper< Text,Text>抽象类，TableMapper 类专门用于完成 MapReduce 中 Map 过程与 HBase 表之间的操作。此时的 map(ImmutableBytesWritable key,Result value,Context context)方法，第一个参数 key 为 HBase 表的 rowkey 主键，第二个参数 value 为 key 主键对应的记录集合，此处的 map 核心实现是遍历 key 主键对应的记录集合 value，将其组合成一条记录通过 contentx.write(key,value)填充到< key,value>键值对中。

源码请参考：

```
public static class WordCountHBaseReaderMapper extends
    TableMapper<Text,Text>{

    @Override
    protected void map(ImmutableBytesWritable key,Result value,Context context)
            throws IOException, InterruptedException {
        StringBuffer sb = new StringBuffer("");
        for(Entry<byte[],byte[]>
entry:value.getFamilyMap("content".getBytes()).entrySet()){
            String str =  new String(entry.getValue());
            //将字节数组转换为 String 类型
            if(str != null){
                sb.append(new String(entry.getKey()));
                sb.append(":");
                sb.append(str);
            }
            context.write(new Text(key.get()), new Text(new String(sb)));
        }
    }
}
```

14.3 Reducer 函数实现

此处的 WordCountHBaseReaderReduce 实现了直接输出 Map 输出的< key,value>键值对，没有对其做任何处理。

源码请参考：

```
public static class WordCountHBaseReaderReduce extends Reducer<Text,Text,Text,Text>{
    private Text result = new Text();
    @Override
    protected void reduce(Text key, Iterable<Text> values,Context context)
            throws IOException, InterruptedException {
        for(Text val:values){
            result.set(val);
            context.write(key, result);
        }
    }
}
```

14.4 驱动函数实现

与 WordCount 的驱动类不同，在 Job 配置的时候没有配置 job.setMapperClass()，而是用以下方法执行 Mapper 类：

```
TableMapReduceUtil.initTableMapperJob(tablename,scan,WordCountHBaseReaderMapper.class
, Text.class, Text.class, job);
```

该方法指明了在执行 job 的 Map 过程时，数据输入源是 HBase 的 tablename 表，通过扫描读入对象 scan 对表进行全表扫描，为 Map 过程提供数据源输入，通过 WordCountHBaseReaderMapper.class 执行 Map 过程，Map 过程的输出 key/value 类型是 Text.class 与 Text.class，最后一个参数是作业对象。特别注意：这里声明的是一个最简单的扫描读入对象 scan，进行表扫描读取数据，其中 scan 可以配置参数，这里为了例子简单不再详述，用户可自行尝试。

源码请参考：

```
public static void main(String[] args) throws Exception {
    String tablename = "wordcount";
    Configuration conf = HBaseConfiguration.create();
    conf.set("hbase.zookeeper.quorum", "Master");
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 1) {
      System.err.println("Usage: WordCountHBaseReader <out>");
      System.exit(2);
    }
    Job job = new Job(conf, "WordCountHBaseReader");
    job.setJarByClass(WordCountHBaseReader.class);
    //设置任务数据的输出路径；
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[0]));
    job.setReducerClass(WordCountHBaseReaderReduce.class);
    Scan scan = new Scan();

TableMapReduceUtil.initTableMapperJob(tablename,scan,WordCountHBaseReaderMapper.class
, Text.class, Text.class, job);
    //调用 job.waitForCompletion(true) 执行任务，执行成功后退出；
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

14.5 部署运行

14.5.1 启动 Hadoop 集群和 HBase 服务

```
[hadoop@Poplar ~]$ start-dfs.sh      #启动 hadoop HDFS 文件管理系统
[hadoop@Poplar ~]$ start-mapred.sh       #启动 hadoop MapReduce 分布式计算服务
[hadoop@Poplar ~]$ start-hbase.sh        #启动 HBase
[hadoop@Poplar ~]$ jps               #查看进程
22003 HMaster
10611 SecondaryNameNode
22226 Jps
21938 HQuorumPeer
10709 ResourceManager
22154 HRegionServer
20277 Main
10432 NameNode
```

14.5.2 部署源码

```
#设置工作环境
[hadoop@Poplar ~]$ mkdir -p /home/hadoop/workspace/HBase
#部署源码
#将 WordCountHBaseReader 文件夹拷贝到/usr/hadoop/workspace/hbase/ 路径下；
```

14.5.3 修改配置文件

1) 查看 HBase 核心配置文件 hbase-site.xml 的 hbase.zookeeper.quorum 属性

参考 HBase API 访问章的部署运行节的修改配置文件小节的"查看 HBase 核心配置文件 hbase-site.xml 的 hbase.zookeeper.quorum 属性"。

2) 修改项目 WordCountHBaseWriter/src/config.properties 属性文件

将项目 WordCountHBaseWriter/src/config.properties 属性文件的 hbase.zookeeper.quorum 属性值修改为上一步查询到的属性值，保持 config.properties 文件的 hbase.zookeeper.quorum 属性值与 hbase-site.xml 文件的 hbase.zookeeper.quorum 属性值一致；

```
#切换工作目录
[hadoop@Poplar ~]$ cd /home/hadoop/workspace/hbase/ WordCountHBaseReader
#修改属性值
[hadoop@Poplar WordCountHBaseReader]$ vim src/config.properties
hbase.zookeeper.quorum=Poplar
#拷贝 src/config.properties 文件到 bin/文件夹
[hadoop@Poplar WordCountHBaseReader]$ cp src/config.properties bin/
```

14.5.4 编译文件

```
#切换工作目录
[hadoop@Poplar ~]$ cd /home/hadoop/workspace/hbase/WordCountHBaseReader
#执行编译
[hadoop@Poplar WordCountHBaseReader]$ javac -classpath /usr/hadoop/hadoop-core-
1.2.1.jar:/usr/hadoop/lib/commons-cli-1.2.jar:lib/zookeeper-3.4.5.jar:lib/hbase-
0.94.20.jar -d bin/ src/com/zonesion/hbase/WordCountHBaseReader.java
#查看编译文件
[hadoop@Poplar WordCountHBaseReader]$ ls bin/com/zonesion/hbase/ -la
total 20
drwxrwxr-x 2 hadoop hadoop 4096 Dec 29 10:36 .
drwxrwxr-x 3 hadoop hadoop 4096 Dec 29 10:36 ..
-rw-rw-r-- 1 hadoop hadoop 2166 Dec 29 14:31 WordCountHBaseReader.class
-rw-rw-r-- 1 hadoop hadoop 2460 Dec 29 14:31
WordCountHBaseReader$WordCountHBaseReaderMapper.class
-rw-rw-r-- 1 hadoop hadoop 1738 Dec 29 14:31
WordCountHBaseReader$WordCountHBaseReaderReduce.class
```

14.5.5 打包 Jar 文件

```
#拷贝 lib 文件夹到 bin 文件夹
[hadoop@Poplar WordCountHBaseReader]$ cp -r lib/ bin/
#打包 Jar 文件
[hadoop@Poplar WordCountHBaseReader]$ jar -cvf WordCountHBaseReader.jar -C bin/ .
added manifest
adding: lib/(in = 0) (out= 0)(stored 0%)
adding: lib/zookeeper-3.4.5.jar(in = 779974) (out= 721150)(deflated 7%)
adding: lib/guava-11.0.2.jar(in = 1648200) (out= 1465342)(deflated 11%)
adding: lib/protobuf-java-2.4.0a.jar(in = 449818) (out= 420864)(deflated 6%)
adding: lib/hbase-0.94.20.jar(in = 5475284) (out= 5038635)(deflated 7%)
adding: com/(in = 0) (out= 0)(stored 0%)
adding: com/zonesion/(in = 0) (out= 0)(stored 0%)
adding: com/zonesion/hbase/(in = 0) (out= 0)(stored 0%)
adding: com/zonesion/hbase/PropertiesHelper.class(in = 4480) (out= 1926)(deflated
57%)
adding: com/zonesion/hbase/WordCountHBaseReader.class(in = 2702) (out= 1226)(deflated
54%)
adding: com/zonesion/hbase/WordCountHBaseReader$WordCountHBaseReaderMapper.class(in =
3250) (out= 1275)(deflated 60%)
adding: com/zonesion/hbase/WordCountHBaseReader$WordCountHBaseReaderReduce.class(in =
2308) (out= 872)(deflated 62%)
adding: config.properties(in = 32) (out= 34)(deflated -6%)
```

14.5.6 运行实例

```
[hadoop@Poplar WordCountHBase]$ hadoop jar WordCountHBaseReader.jar
WordCountHBaseReader /user/hadoop/WordCountHBaseReader/output/
.....................省略.............
14/12/30 17:51:58 INFO mapred.JobClient: Running job: job_201412161748_0035
14/12/30 17:51:59 INFO mapred.JobClient:  map 0% reduce 0%
14/12/30 17:52:13 INFO mapred.JobClient:  map 100% reduce 0%
14/12/30 17:52:26 INFO mapred.JobClient:  map 100% reduce 100%
```

```
14/12/30 17:52:27 INFO mapred.JobClient: Job complete: job_201412161748_0035
14/12/30 17:52:27 INFO mapred.JobClient: Counters: 39
14/12/30 17:52:27 INFO mapred.JobClient:   Job Counters
14/12/30 17:52:27 INFO mapred.JobClient:     Launched reduce tasks=1
14/12/30 17:52:27 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=4913
14/12/30 17:52:27 INFO mapred.JobClient:     Total time spent by all reduces waiting
after reserving slots (ms)=0
14/12/30 17:52:27 INFO mapred.JobClient:     Total time spent by all maps waiting
after reserving slots (ms)=0
14/12/30 17:52:27 INFO mapred.JobClient:     Rack-local map tasks=1
14/12/30 17:52:27 INFO mapred.JobClient:     Launched map tasks=1
14/12/30 17:52:27 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=13035
14/12/30 17:52:27 INFO mapred.JobClient:   HBase Counters
14/12/30 17:52:27 INFO mapred.JobClient:     REMOTE_RPC_CALLS=8
14/12/30 17:52:27 INFO mapred.JobClient:     RPC_CALLS=8
14/12/30 17:52:27 INFO mapred.JobClient:     RPC_RETRIES=0
14/12/30 17:52:27 INFO mapred.JobClient:     NOT_SERVING_REGION_EXCEPTION=0
14/12/30 17:52:27 INFO mapred.JobClient:     NUM_SCANNER_RESTARTS=0
14/12/30 17:52:27 INFO mapred.JobClient:     MILLIS_BETWEEN_NEXTS=9
14/12/30 17:52:27 INFO mapred.JobClient:     BYTES_IN_RESULTS=216
14/12/30 17:52:27 INFO mapred.JobClient:     BYTES_IN_REMOTE_RESULTS=216
14/12/30 17:52:27 INFO mapred.JobClient:     REGIONS_SCANNED=1
14/12/30 17:52:27 INFO mapred.JobClient:     REMOTE_RPC_RETRIES=0
14/12/30 17:52:27 INFO mapred.JobClient:   File Output Format Counters
14/12/30 17:52:27 INFO mapred.JobClient:     Bytes Written=76
14/12/30 17:52:27 INFO mapred.JobClient:   FileSystemCounters
14/12/30 17:52:27 INFO mapred.JobClient:     FILE_BYTES_READ=92
14/12/30 17:52:27 INFO mapred.JobClient:     HDFS_BYTES_READ=68
14/12/30 17:52:27 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=159978
14/12/30 17:52:27 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=76
14/12/30 17:52:27 INFO mapred.JobClient:   File Input Format Counters
14/12/30 17:52:27 INFO mapred.JobClient:     Bytes Read=0
14/12/30 17:52:27 INFO mapred.JobClient:   Map-Reduce Framework
14/12/30 17:52:27 INFO mapred.JobClient:     Map output materialized bytes=92
14/12/30 17:52:27 INFO mapred.JobClient:     Map input records=5
14/12/30 17:52:27 INFO mapred.JobClient:     Reduce shuffle bytes=92
14/12/30 17:52:27 INFO mapred.JobClient:     Spilled Records=10
14/12/30 17:52:27 INFO mapred.JobClient:     Map output bytes=76
14/12/30 17:52:27 INFO mapred.JobClient:     Total committed heap usage
(bytes)=211025920
14/12/30 17:52:27 INFO mapred.JobClient:     CPU time spent (ms)=2160
14/12/30 17:52:27 INFO mapred.JobClient:     Combine input records=0
14/12/30 17:52:27 INFO mapred.JobClient:     SPLIT_RAW_BYTES=68
14/12/30 17:52:27 INFO mapred.JobClient:     Reduce input records=5
14/12/30 17:52:27 INFO mapred.JobClient:     Reduce input groups=5
14/12/30 17:52:27 INFO mapred.JobClient:     Combine output records=0
14/12/30 17:52:27 INFO mapred.JobClient:     Physical memory (bytes)
snapshot=263798784
14/12/30 17:52:27 INFO mapred.JobClient:     Reduce output records=5
14/12/30 17:52:27 INFO mapred.JobClient:     Virtual memory (bytes)
snapshot=1491795968
14/12/30 17:52:27 INFO mapred.JobClient:     Map output records=5
```

14.5.7 查看运行结果

```
[hadoop@Poplar WordCountHBaseReader]$ hadoop fs  -ls
/user/hadoop/WordCountHBaseReader/output/
Found 3 items
```

```
-rw-r--r--   1 hadoop supergroup          0 2014-07-28 18:04
/user/hadoop/WordCountHBaseReader/output/_SUCCESS
drwxr-xr-x   - hadoop supergroup          0 2014-07-28 18:04
/user/hadoop/WordCountHBaseReader/output/_logs
-rw-r--r--   1 hadoop supergroup         76 2014-07-28 18:04
/user/hadoop/WordCountHBaseReader/output/part-r-00000
[hadoop@Poplar WordCountHBaseReader]$ hadoop fs -cat
/user/hadoop/WordCountHBaseReader/output/part-r-00000
Bye count:1
Goodbye count:1
Hadoope count:2
Hellope count:2
Worldpe count:2
```

15 附录

15.1 编译 Hadoop 源码

若想自己编译源码，需安装 g++ 和 Protocol Buffers：

```
sudo tar -zxvf ~/下载/protobuf-2.5.0.tar.gz -C /opt/
sudo apt-get install g++
cd /opt/protobuf-2.5.0
sudo ./configure --PREFIX=/opt/protobuf
sudo make
sudo make check
sudo make install
protoc --version
```

最后一行代码应输出 protoc 的版本信息，若不能运行，则执行：

```
sudo ldconfig
protoc --version
```

若提示出错错误 “protoc: error while loading shared libraries: libprotoc.so.8: cannot open shared object file: No such file or directory”。则执行：

```
export LD_LIBRARY_PATH=/opt/protobuf/lib/
```

安装后切换到 Hadoop 源码目录下，执行：

```
cd ~/hadoop-2.7.1-src
mvn package -Pdist,native -DskipTests -Dtar
```

编译完成后，在 hadoop-2.7.1-src/hadoop-dist/target 可看到。