第 3 章 Hadoop 分布式存储企业实战

3.1 Hadoop 概念剖析

Hadoop 是由 Apache 基金会开发的分布式系统基础架构。开发人员可以在不了解分布式底层细节的情况下开发分布式应用，充分利用集群的优势进行高速运算和存储。

Hadoop 实现了一个分布式文件系统（Distributed File System），其中一个组件是 HDFS（Hadoop Distributed File System）。HDFS 有高容错的特点，用来部署在低廉的（low-cost）硬件上，适合那些有超大数据集（large data set）的应用程序。

HDFS 放宽了 POSIX 的要求，可以以流的形式访问文件系统中的数据。Hadoop 框架最核心的设计是 HDFS 和 MapReduce。HDFS 为海量的数据提供了存储，MapReduce 为海量的数据提供了计算。

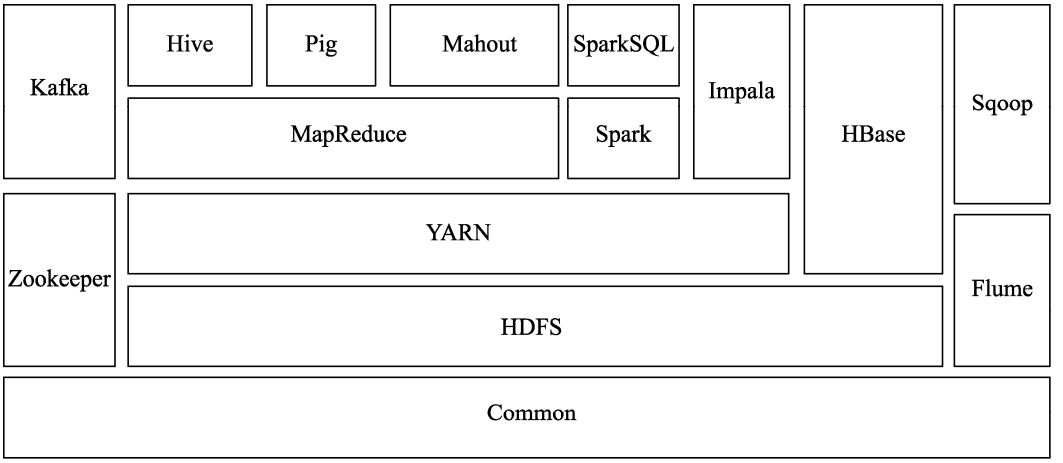
Hadoop 起源于 Apache Nutch 项目，始于 2002 年，是 Apache Lucene 的子项目之一。2004 年，Google 在“操作系统设计与实现”（Operating System Design and Implementation，OSDI）会议上公开发表了题为 *MapReduce：Simplified Data Processing on Large Clusters*（MapReduce：简化大规模集群上的数据处理）的论文之后，受到启发的 Doug Cutting 等人开始尝试实现 MapReduce 计算框架，并将它与 NDFS（Nutch Distributed File System）结合，用以支持 Nutch 引擎的主要算法。

由于 NDFS 和 MapReduce 在 Nutch 引擎中有着良好的应用，所以它们于 2006 年 2 月被分离出来，成为一套完整而独立的软件，并命名为 Hadoop。到了 2008 年年初，Hadoop 已成为 Apache 的顶级项目，包含众多子项目，被应用到包括阿里、百度、腾讯、雅虎、京东在内的很多互联

网公司。

3.2　Hadoop 服务组件

Hadoop 生态圈的服务组件繁多，常见的有 Common、HDFS、HBase、MapReduce 等，如图 3-1 所示。



Hadoop生态圈

图 3-1　Hadoop 生态圈

（1）Hadoop Common 是 Hadoop 体系最底层的一个模块，为 Hadoop 各个子模块提供各种工具，如系统配置工具 Configuration、远程调用 RPC、序列化机制和日志操作等，是其他模块的基础。

（2）HDFS 是 Hadoop 分布式文件系统缩写，是 Hadoop 的基石。HDFS 是一个具备高度容错性的文件系统，适合部署在廉价的机器上，能提供高吞吐量的数据访问，非常适合具有大规模数据集的应用。

（3）YARN 是统一资源管理和调度平台。它解决了上一代 Hadoop 资源利用率低和不能兼容异构计算框架等多种问题，实现了资源隔离方案和双调度器。

（4）MapReduce 是一种编程模型，利用函数式编程思想，将对数据集的过程分为 Map 和 Reduce 两个阶段。MapReduce 编程模型非常适合进行分布式计算。Hadoop 提供 MapReduce 的计算框架，实现了这种编程模型，开发人员可以通过 Java、C++、Python、PHP 等多种语言进

行编程。

（5）Spark 是加州伯克利大学 AMP 实验室开发的新一代计算框架，在迭代计算方面有很大优势，与 MapReduce 相比性能提升明显，可以与 YARN 集成，还提供了 SparkSQL 组件。

（6）HBase 源于 Google 的 Bigtable 论文，是一个分布式的开源数据库，采用了 Bigtable 的数据模型——列族。HBase 擅长对大规模数据进行随机、实时读写访问。

（7）Zookeeper 作为一个分布式服务框架，是基于 Fast Paxos 算法实现的，解决分布式系统中一致性的问题。提供了配置维护、名字服务、分布式同步、组服务等。

Hive 最早是 Facebook 开发并使用的，是基于 Hadoop 的一个数据仓库工具，可以将结构化的数据文件映射为一张表，提供简单的 SQL 查询功能，并将 SQL 转化为 MapReduce 作业运行。优点是学习成本低，降低了 Hadoop 的使用门槛。

（8）Pig 与 Hive 类似，也是对大规模数据集进行分析和评估的工具。与 Hive 不同的是，Pig 提供了一种高层的、面向领域的抽象语言 Pig Latin。Pig 可以将 Pig Latin 转化为 MapReduce 作业。与 SQL 相比，Pig Latin 更加灵活，但学习成本更高。

（9）Impala 是 Cloudera 公司开发的，可以对数据提供交互查询的 SQL 接口。Impala 和 Hive 使用相同的统一存储平台，相同的元数据，SQL 语法，ODBC 驱动程序和用户界面。Impala 还提供了一个熟悉的面向批量或者实时查询的统一平台。Impala 的特点是查询速度非常快，其性能大幅度领先于 Hive。Impala 并不是基于 MapReduce 的，它的定位是 OLAP，是 Google 的新三驾马车之一 Dremel 的开源实现。

（10）Mahout 是一个机器学习和数据挖掘的库，它利用 MapReduce 编程模型实现 *k*–means、Native、Bayes、Collaborative Filtering 等经典的机器学习算法，并使其具有良好的可扩展性。

（11）Flume 是 Cloudera 公司提供的一个高可用、高可靠、分布式的海量日志采集、聚合和传输系统，支持在日志系统中定制各类数据发送方，用于数据收集。Flume 拥有对数据进行简单处理并写到各个数据接收方的能力。

（12）Sqoop 是 SQL to Hadoop 的缩写，主要作用是在结构化的数据存储与 Hadoop 之间进行数据双向交换。也就是说，Sqoop 可以将关系型数据库的数据导入 HDFS、Hive，也可以将其从 HDFS、Hive 导出到关系型数据库中。Sqoop 利用了 Hadoop 的优点，整个导入导出都是由 MapReduce 计算框架实现并行化，非常高效。

（13）Kafka 是一种高吞吐量的分布式发布/订阅消息系统，具有分布式、高可用的特性，在大数据系统中得到了广泛应用。如果把大数据系统比作一台计算机，那么 Kafka 就是前端总线，

它连接了平台中的各个组件。

3.3　Hadoop 工作原理

HDFS 即 Hadoop 分布式文件系统，它的设计目标是把大规模数据集存储到集群中的多台普通商用计算机上，并提供高可靠性和高吞吐量的服务。

HDFS 是参考 Google 公司的 GFS 实现的，不管是 Google 公司的计算平台还是 Hadoop 计算平台，都是运行在大量普通商用计算机上的，这些计算机节点很容易出现硬件故障，而这两种计算平台都将硬件故障作为常态，通过软件设计保证系统的可靠性。

HDFS 的数据分块地存储在每个节点上，当某个节点出现故障时，HDFS 相关组件能快速检测故障，提供容错机制并完成数据的自动恢复。

HDFS 主要由 NameNode、SecondaryNameNode 和 DataNode 3 个组件构成，是以 Master/Slave 模式运行的。

其中，NameNode 和 SecondaryNameNode 运行在 Master 节点上，而 DataNode 运行在 Slave 节点上，所以 HDFS 集群一般由一个 NameNode、一个 SecondaryNameNode 和多个 DataNode 组成，其架构如图 3-2 所示。

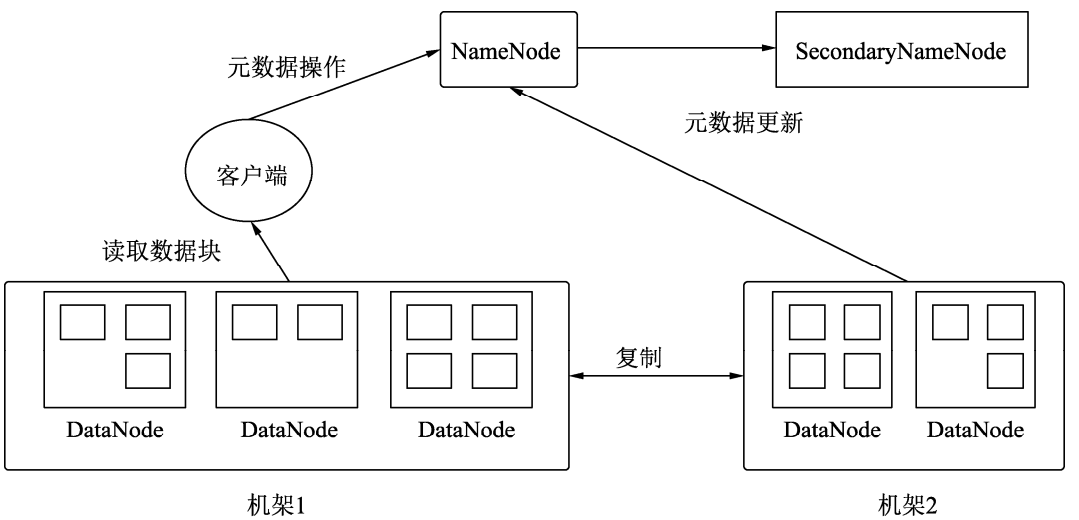


图 3-2　HDFS 集群架构

在 HDFS 中，文件是被分成块进行存储的，一个文件可以包含多个块，每个块存储在不同的 DataNode 中。从图 3-2 中可知，当一个客户端请求读取一个文件时，它需要先从 NameNode 中获取文件的元数据信息，然后从对应的数据节点上并行地读取数据块。

1. NameNode

NameNode 是主服务器，负责管理文件系统的命名空间及客户端对文件的访问。当客户端请求数据时，仅从 NameNode 中获取文件的元数据信息，具体的数据传输不经过 NameNode，而是直接与具体的 DataNode 进行交互。

文件的元数据信息记录了文件系统中的文件名和目录名，以及它们之间的层级关系，每个文件目录的所有者及其权限，还记录了每个文件由哪些块组成，这些元数据信息记录在文件 fsimage 中，当系统初次启动时，NameNode 将读取 fsimage 中的信息并保存到内存中。

这些块的位置信息是由 NameNode 启动后从每个 DataNode 获取并保存在内存当中的，这样既减少了 NameNode 的启动时间，又减少了读取数据的查询时间，提高了整个系统的效率。

2. SecondaryNameNode

从字面意义上来看，SecondaryNameNode 很容易被当作 NameNode 的备份节点，其实不然。可以通过图 3-3 看 HDFS 中 SecondaryNameNode 的作用。

NameNode 管理着元数据信息，元数据信息会定期保存到 edits 和 fsimage 文件中。其中，edits 文件保存操作日志信息，在 HDFS 运行期间，新的操作日志不会立即与 fsimage 文件进行合并，也不会保存到 NameNode 的内存中，而是会先写入 edits 文件。

edits 文件达到一定阈值后，或间隔一段时间，会触发 SecondaryNameNode，使其进行工作，这个时间点称为 checkpoint。

SecondaryNameNode 的角色就是定期地合并 edits 和 fsimage 文件，其合并步骤如下：

在进行合并之前，SecondaryNameNode 会通知 NameNode 停用当前的 editlog 文件，NameNode 会将新记录写入新的 editlog.new 文件中。

SecondaryNameNode 向 NameNode 请求复制 fsimage 和 edits 文件，然后把 fsimag 和 edits 文件合并成新的 fsimage 文件，并命名为 fsimage.ckpt。

NameNode 从 SecondaryNameNode 获取 fsimage.ckpt，并替换掉 fsimage，同时用 edits.new 文件替换旧的 edits 文件，更新 checkpoint 的时间。

最终 fsimage 保存的是上一个 checkpoint 的元数据信息，而 edits 保存的是从上个 checkpoint 开始发生的 HDFS 元数据改变的信息。

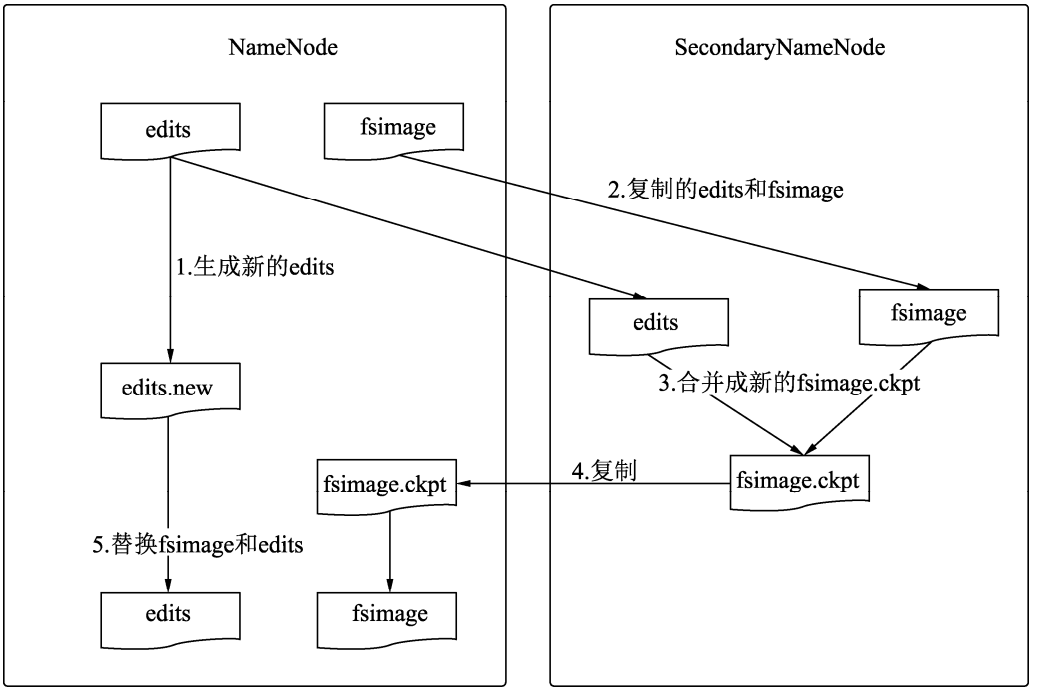


图 3-3　SecondaryNameNode 数据管理

3. DataNode

DataNode 是 HDFS 中的工作节点，也是从服务器，负责存储数据块、为客户端提供数据块的读写服务，同时响应 NameNode 的相关指令，如完成数据块的复制、删除等。

DataNode 会定期发送心跳信息给 NameNode，告知 NameNode 当前节点存储的文件块信息。当客户端给 NameNode 发送读写请求时，NameNode 告知客户端每个数据块所在的 DataNode 信息，然后客户端直接与 DataNode 进行通信，减少了 NameNode 的系统开销。

当 DataNode 在执行块存储操作时，DataNode 还会与其他 DataNode 通信，复制这些块到其他 DataNode 上进行冗余实现。

3.4　HDFS 分块与副本机制

在 HDFS 中，文件最终是以数据块的形式存储的，而副本机制极大程度上避免了宕机所造成的数据丢失，可以在数据读取时进行数据校验。

1. 分块机制

HDFS 中数据块大小默认为 64MB，而一般磁盘块的大小为 512B，HDFS 中的数据块之所以这么大，是为了最小化寻址开销。

如果数据块足够大，从磁盘传输数据的时间会明显大于寻找数据块的地址的时间，因此，传输一个由多个数据块组成的大文件的时间取决于磁盘传输速率。

随着新一代磁盘驱动器传输速率的提升，寻址的开销会更少，在大多数情况下 HDFS 使用更大的块。当然数据块不是越大越好，因为 Hadoop 中一个 map 任务一次通常只处理一个数据块中的数据，如果数据块过大，会导致整体任务数量过小，降低作业处理的速度。HDFS 按块存储有以下好处。

（1）文件可以任意大，不会受到单个节点的磁盘容量的限制。理论上讲，HDFS 的存储容量是无限的。

（2）简化文件子系统的设计。将系统的处理对象设置为数据块，可以简化存储管理，因为数据块大小固定，所以每个文件分成多少个数据块，每个 DataNode 能存多少个数据块，都很容易计算。同时系统中 NameNode 只负责管理文件的元数据，DataNode 只负责数据存储，分工明确，提高了系统的效率。

（3）有利于提高系统的可用性。HDFS 通过数据备份来提供数据的容错能力和高可用性，而按照数据块的存储方式非常适合数据备份。同时，数据块以副本形式存在于多个 DataNode 中，有利于负载均衡，即当某个节点处于繁忙状态时，客户端还可以从其他节点获取这个数据块的副本。

2. 副本机制

HDFS 中数据块的副本数默认为 3，当然数据块的副本数可以设置，这些副本分散存储在集群中。副本的分布位置直接影响 HDFS 的可靠性和性能。

大型的分布式文件系统会跨多个机架，图 3-4 所示为 HDFS 涉及 2 个机架的架构。

把所有副本都存放在不同的机架上，可以防止出现由机架故障导致数据块不可用的情况，同时在多个客户端访问文件系统时很容易实现负载均衡。如果是写数据，则各个数据块需要同步到不同机架上，会影响写数据的效率。

在默认 3 个副本情况下，HDFS 会把第 1 个副本放到机架的一个节点上，第 2 个副本放在同一个机架的另一个节点上，第 3 个副本放在不同的机架上。这种策略减少了跨机架副本的个数，提高了数据块的写性能，也可以保证当一个机架出现故障时仍然能正常运转。

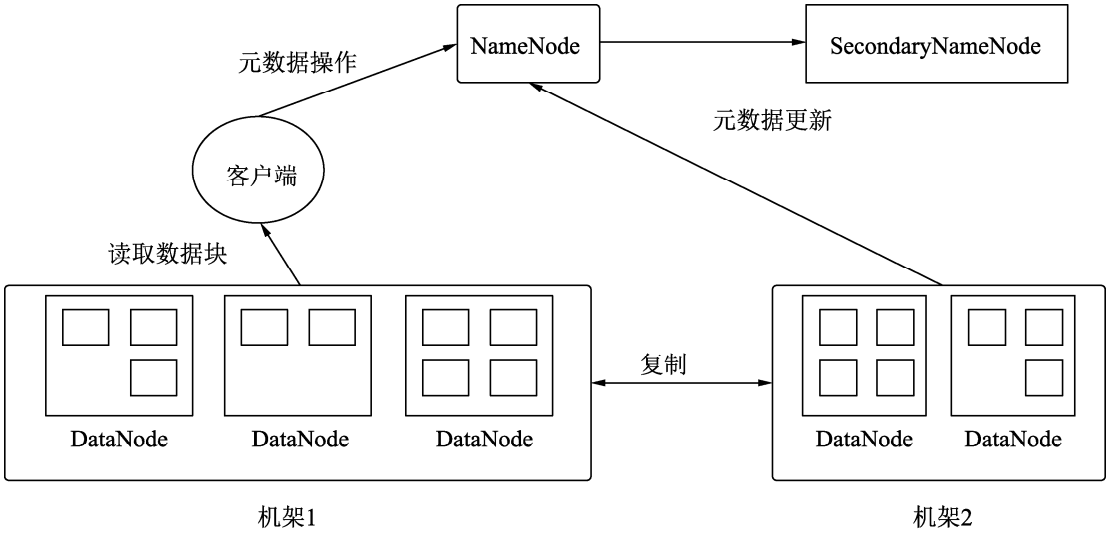


图 3-4　HDFS 架构图（2 个机架）

3.5　HDFS 读写机制剖析

前面讲到客户端读写文件是要与 NameNode 和 DataNode 通信的，下面详细介绍 HDFS 中读写文件的过程。

3.5.1　读文件

HDFS 通过 RPC 调用 NameNode 获取文件块的位置信息，并且返回每个数据块所在的 DataNode 的地址信息，然后再从 DataNode 获取数据块的副本。HDFS 读文件的过程如图 3-5 所示。

Hadoop 读文件操作步骤如下所述。

（1）客户端发起文件读取的请求。

（2）NameNode 将文件对应的数据块信息及每个数据块的位置信息，包括每个数据块的所有副本的位置信息（即每个副本所在的 DataNode 的地址信息）都传送给客户端。

（3）客户端收到数据块信息后，直接和数据块所在的 DataNode 通信，并行地读取数据块。

在客户端获得 NameNode 关于每个数据块的信息后，客户端会根据网络拓扑选择与它最近的 DataNode 来读取每个数据块。当与 DataNode 通信失败时，它会选取另一个较近的 DataNode，同时会对出故障的 DataNode 做标记，避免与其重复通信，并发送 NameNode 故障节点的信息。

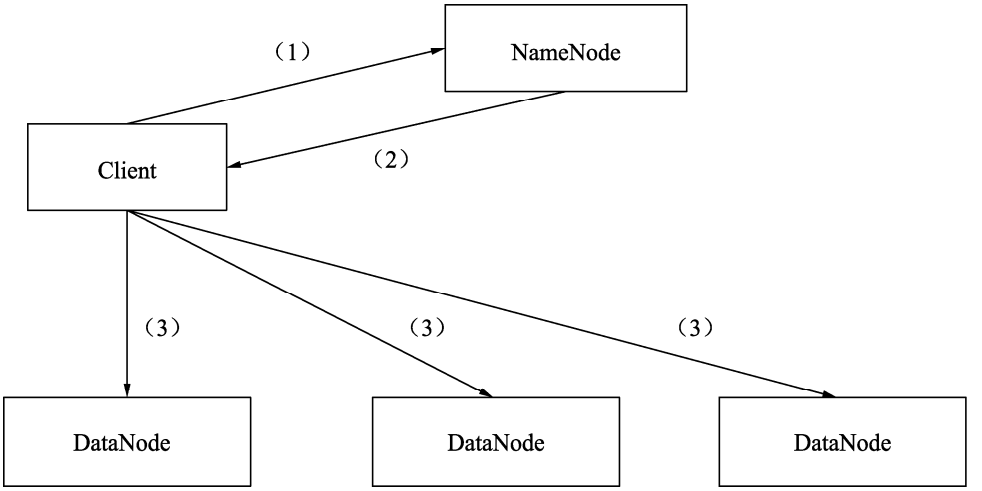


图 3-5　HDFS 读文件的过程

3.5.2　写文件

当客户端发送写文件请求时，NameNode 负责通知 DataNode 创建文件，在创建之前会检查客户端是否有允许写入数据的权限。通过检测后，NameNode 会向 edits 文件写入一条创建文件的操作记录。

HDFS 写文件的过程如图 3-6 所示。

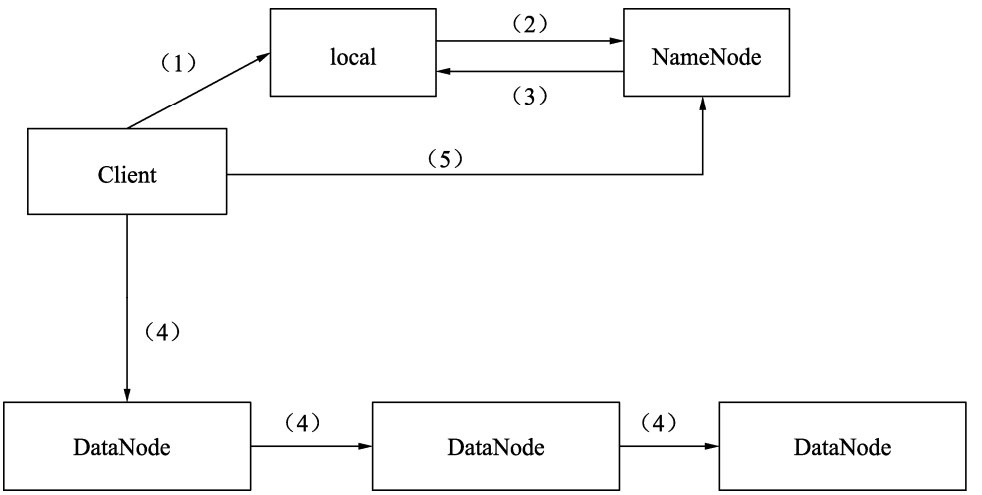


图 3-6　Hadoop 写文件的过程

Hadoop 写文件的操作步骤如下。

（1）客户端在向 NameNode 发送写请求之前，先将数据写入本地的临时文件中。

（2）待临时文件块达到系统设置的块大小时，开始向 NameNode 请求写文件。

（3）NameNode 检查集群中每个 DataNode 的状态信息，获取空闲的节点，并在检查客户端权限后创建文件，返回客户端一个数据块及其对应 DataNode 的地址列表。列表中包含副本存放的地址。

（4）客户端在获取 DataNode 相关信息后，将临时文件中的数据块写入列表中的第 1 个 DataNode，同时第 1 个 DataNode 会将数据以副本的形式传送至第 2 个 DataNode，第 2 个节点也会将数据传送至第 3 个 DataNode。DataNode 以数据包的形式从客户端接收数据，并以流水线的形式将数据写入和备份到所有的 DataNode 中，每个 DataNode 收到数据后会向前一个节点发送确认信息。数据传输完毕，第 1 个 DataNode 会向客户端发送确认信息。

（5）当客户端收到每个 DataNode 的确认信息时，表示数据块已经持久化地存储在所有 DataNode 中，接着客户端会向 NameNode 发送确认信息。如果在第 4 步中任意一个 DataNode 失败，客户端则会告知 NameNode，将数据备份到新的 DataNode 中。

3.6　Hadoop 环境要求

从 0 开始构建一套 Hadoop 大数据平台，实战环境为目前主流服务器操作系统 CentOS 7.x。Hadoop 的安装部署属于 Java 进程，就是启动了 JVM 进程运行服务。

（1）HDFS：存储数据，提供分析的数据。

```
NameNode/DataNode
```

（2）YARN：提供程序运行的资源。

```
ResourceManager/NodeManager
```

```
系统版本：CentOS 7.x x86_64
Java 版本：JDK-1.8.0_131
Hadoop 版本：hadoop-3.2.1
192.168.1.145   namenode、secondary namenode、resource manager
192.168.1.146   datanode、nodemanager
192.168.1.147   datanode、nodemanager
```

3.7 hosts 及防火墙设置

部署 Hadoop 服务之前，需要对 node1、node2、node3 节点进行添加 hosts、同步时间、关闭防火墙、SELinux、修改主机名等操作，操作方法和命令如下：

```
cat >/etc/hosts<<EOF
127.0.0.1  localhost localhost.localdomain
192.168.1.145 node1
192.168.1.146 node2
192.168.1.147 node3
EOF
sed -i '/SELINUX/s/enforcing/disabled/g'  /etc/sysconfig/selinux
setenforce  0
systemctl   stop    firewalld.service
systemctl   disable   firewalld.service
yum install ntpdate rsync lrzsz -y
ntpdate  pool.ntp.org
hostname 'cat /etc/hosts|grep $(ifconfig|grep broadcast|awk '{print $2}')|
awk '{print $2}'';su
```

3.8 配置节点免密钥登录

node1 节点作为 Master 控制节点，执行以下指令创建公钥和私钥，然后将公钥复制至其余节点即可。

```
ssh-keygen -t rsa -N '' -f /root/.ssh/id_rsa -q
ssh-copy-id -i /root/.ssh/id_rsa.pub root@node1
ssh-copy-id -i /root/.ssh/id_rsa.pub root@node2
ssh-copy-id -i /root/.ssh/id_rsa.pub root@node3
```

3.9 配置节点 Java 环境

为所有的节点配置 Java JDK 环境，操作方法和命令如下：

```
#解压 JDK 软件包
tar  -xvzf  jdk1.8.0_131.tar.gz
#创建 JDK 部署目录
```

```
mkdir -p  /usr/java/
\mv jdk1.8.0_131 /usr/java/
#设置环境变量
cat>>/etc/profile<<EOF
export JAVA_HOME=/usr/java/jdk1.8.0_131/
export HADOOP_HOME=/data/hadoop/
export JAVA_LIBRARY_PATH=/data/hadoop/lib/native/
export PATH=\$PATH:\$HADOOP_HOME/bin/:\$JAVA_HOME/bin
EOF
#使环境变量生效
source /etc/profile
java -version
```

3.10　Hadoop 部署实战

在 node1 节点部署 Hadoop 并且进行相应的配置，操作方法和命令如下：

```
#下载 Hadoop 软件包
yum install wget -y
wget -c https://mirrors.tuna.tsinghua.edu.cn/apache/hadoop/common/hadoop-
3.2.1/hadoop-3.2.1.tar.gz
#解压 Hadoop 软件包
tar  -xzvf  hadoop-3.2.1.tar.gz
#创建 Hadoop 程序&数据目录
mkdir -p /data/
#将 Hadoop 程序部署至/data/hadoop 目录下
\mv hadoop-3.2.1/ /data/hadoop/
#查看 Hadoop 是否部署成功
ls -l /data/hadoop/
```

3.11　node1 Hadoop 配置

（1）修改 node1 节点 core-site.xml 文件代码（操作命令：vi/data/hadoop/etc/hadoop/core-site.xml），代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
```

```
  <name>fs.default.name</name>
  <value>hdfs://node1:9000</value>
 </property>
<property>
  <name>hadoop.tmp.dir</name>
  <value>/tmp/hadoop-${user.name}</value>
  <description>A base for other temporary directories.</description>
</property>
</configuration>
```

（2）修改 node1 节点 mapred-site.xml 文件代码（操作命令：vi /data/hadoop/etc/hadoop/mapred-site.xml），代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
    <property>
     <name>mapred.job.tracker</name>
     <value>node1:9001</value>
    </property>
</configuration>
```

（3）修改 node1 节点 hdfs-site.xml 文件代码（操作命令：vi /data/hadoop/etc/hadoop/hdfs-site.xml），代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
<name>dfs.name.dir</name>
<value>/data/hadoop/data_name1,/data/hadoop/data_name2</value>
</property>
<property>
<name>dfs.data.dir</name>
<value>/data/hadoop/data_1,/data/hadoop/data_2</value>
</property>
<property>
<name>dfs.replication</name>
<value>2</value>
</property>
</configuration>
```

（4）修改 hadoop-env.sh 文件（操作命令：vi /data/hadoop/etc/hadoop/hadoop-env.sh），在末尾追加 JAVA_HOME 变量，操作命令如下：

```
echo "export JAVA_HOME=/usr/java/jdk1.8.0_131/" >> /data/hadoop/etc/
hadoop/hadoop-env.sh
```

（5）修改 workers 文件（操作命令：vi /data/hadoop/etc/hadoop/workers），操作命令如下：

```
cat>/data/hadoop/etc/hadoop/workers<<EOF
node1
node2
node3
EOF
```

（6）修改 Hadoop 默认启动、关闭脚本，添加 root 执行权限，操作命令如下：

```
cd /data/hadoop/sbin/
for i in 'ls start*.sh stop*.sh';do sed -i "1a\HDFS_DATANODE_USER=root\nHDFS_
DATANODE_SECURE_USER=root\nHDFS_NAMENODE_USER=root\nHDFS_SECONDARYNAMENODE_
USER=root\nYARN_RESOURCEMANAGER_USER=root\n\YARN_NODEMANAGER_USER=root"
$i ;done
```

（7）将 node1 部署完成的与 Hadoop 有关的所有文件、目录同步至 node2 和 node3 节点，操作命令如下：

```
for i in 'seq 2 3';do ssh -l root node$i -a "mkdir -p /data/hadoop/" ;done
for i in 'seq 2 3';do rsync -aP --delete /data/hadoop/ root@node$i:/data/
hadoop/ ;done
```

3.12　启动 Hadoop 服务

在启动 Hadoop 服务之前，需要执行一个非常关键的操作，即需要在 NameNode 上执行初始化命令，初始化 name 目录和数据目录。操作命令如下，结果如图 3-7 所示。

```
#初始化集群
hadoop namenode -format
#停止所有服务
/data/hadoop/sbin/stop-all.sh
#使用 kill 停止服务
ps -ef|grep hadoop|grep java |grep -v grep |awk '{print $2}'|xargs kill -9
sleep 2
#启动所有服务
/data/hadoop/sbin/start-all.sh
```

```
[root@node1 ~]#
[root@node1 ~]# hadoop namenode -format
WARNING: Use of this script to execute namenode is deprecated.
WARNING: Attempting to execute replacement "hdfs namenode" instead.

WARNING: /data/hadoop//logs does not exist. Creating.
2021-04-12 18:01:01,692 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = node1/192.168.1.145
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 3.3.0
STARTUP_MSG:   classpath = /data/hadoop//etc/hadoop:/data/hadoop//share/hadoop/comm
```

（a）

```
2021-04-12 18:01:03,660 INFO common.Storage: Storage directory /data/hadoop/data
2021-04-12 18:01:03,662 INFO common.Storage: Storage directory /data/hadoop/data
2021-04-12 18:01:03,725 INFO namenode.FSImageFormatProtobuf: Saving image file /
ression
2021-04-12 18:01:03,731 INFO namenode.FSImageFormatProtobuf: Saving image file /
ression
2021-04-12 18:01:03,958 INFO namenode.FSImageFormatProtobuf: Image file /data/ha
ved in 0 seconds .
2021-04-12 18:01:03,958 INFO namenode.FSImageFormatProtobuf: Image file /data/ha
ved in 0 seconds .
2021-04-12 18:01:04,004 INFO namenode.NNStorageRetentionManager: Going to retain
2021-04-12 18:01:04,017 INFO namenode.FSImage: FSImageSaver clean checkpoint: tx
2021-04-12 18:01:04,018 INFO namenode.FSImage: FSImageSaver clean checkpoint: tx
2021-04-12 18:01:04,020 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
```

（b）

```
[root@node1 ~]# /data/hadoop/sbin/start-all.sh
Starting namenodes on [node1]
Last login: Mon Apr 12 17:48:48 CST 2021 on pts/0
node1: Warning: Permanently added 'node1,192.168.1.145' (ECDSA) to the list of kno
node1: Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).
Starting datanodes
Last login: Mon Apr 12 18:04:59 CST 2021 on pts/0
node1: Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).
node3: WARNING: /data/hadoop/logs does not exist. Creating.
node2: WARNING: /data/hadoop/logs does not exist. Creating.
Starting secondary namenodes [node1]
Last login: Mon Apr 12 18:04:59 CST 2021 on pts/0
node1: Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).
```

（c）

图 3-7　初始化 name 目录和数据目录

3.13　Hadoop 集群验证

查看 3 个节点的 Hadoop 服务进程和服务端口信息，操作命令如下，结果如图 3-8 所示。

```
#查看服务进程
ps -ef|grep -aiE hadoop
```

```
#查看服务端口
netstat -tnlp
#查看 Java 进程
jps
#查看 Hadoop 日志内容
tail -fn 100 /data/hadoop/logs/*
```

```
[root@node2 ~]# ps -ef|grep -aiE hadoop
root       6680      1  6 18:06 ?        00:00:08 /usr/java/jdk1.8.0_131//bin/j
OR,RFAS -Dyarn.log.dir=/data/hadoop/logs -Dyarn.log.file=hadoop-root-datanode
ibrary.path=/data/hadoop/lib/native -Dhadoop.log.dir=/data/hadoop/logs -Dhado
op.id.str=root -Dhadoop.root.logger=INFO,RFA -Dhadoop.policy.file=hadoop-poli
root       6800      1  9 18:06 ?        00:00:10 /usr/java/jdk1.8.0_131//bin/j
op/logs -Dyarn.log.file=hadoop-root-nodemanager-node2.log -Dyarn.home.dir=/da
ative -Dhadoop.log.dir=/data/hadoop/logs -Dhadoop.log.file=hadoop-root-nodema
t.logger=INFO,RFA -Dhadoop.policy.file=hadoop-policy.xml -Dhadoop.security.lo
root       6912  6169  0 18:08 pts/0    00:00:00 grep --color=auto -aiE hadoop
[root@node2 ~]#
```

（a）

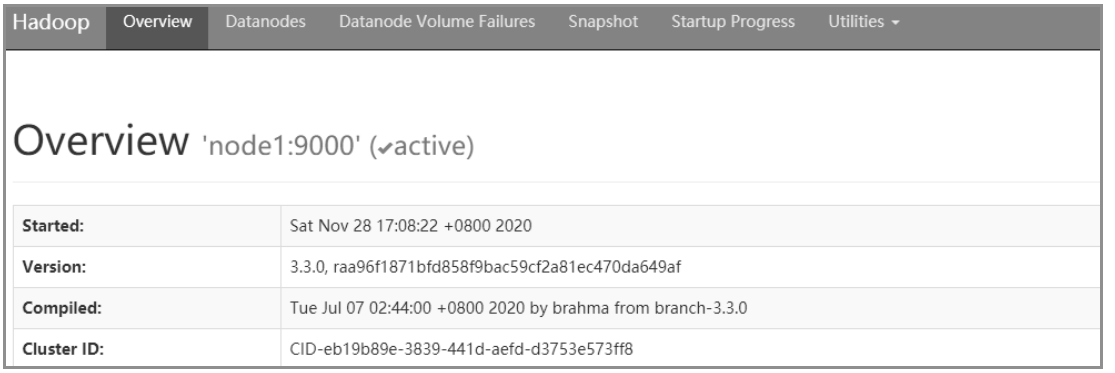
```
[root@node1 ~]# netstat -tnlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:8040            0.0.0.0:*               LISTEN      9146/java
tcp        0      0 0.0.0.0:9864            0.0.0.0:*               LISTEN      8528/java
tcp        0      0 192.168.1.145:9000      0.0.0.0:*               LISTEN      8394/java
tcp        0      0 0.0.0.0:8042            0.0.0.0:*               LISTEN      9146/java
tcp        0      0 127.0.0.1:45258         0.0.0.0:*               LISTEN      8528/java
tcp        0      0 0.0.0.0:9866            0.0.0.0:*               LISTEN      8528/java
tcp        0      0 0.0.0.0:9867            0.0.0.0:*               LISTEN      8528/java
tcp        0      0 0.0.0.0:9868            0.0.0.0:*               LISTEN      8735/java
tcp        0      0 0.0.0.0:9870            0.0.0.0:*               LISTEN      8394/java
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      5536/sshd
tcp        0      0 0.0.0.0:8088            0.0.0.0:*               LISTEN      9016/java
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      5740/master
tcp        0      0 0.0.0.0:42235           0.0.0.0:*               LISTEN      9146/java
tcp        0      0 0.0.0.0:8030            0.0.0.0:*               LISTEN      9016/java
tcp        0      0 0.0.0.0:8031            0.0.0.0:*               LISTEN      9016/java
tcp        0      0 0.0.0.0:8032            0.0.0.0:*               LISTEN      9016/java
tcp        0      0 0.0.0.0:8033            0.0.0.0:*               LISTEN      9016/java
tcp6       0      0 :::22                   :::*                    LISTEN      5536/sshd
tcp6       0      0 ::1:25                  :::*                    LISTEN      5740/master
```

（b）

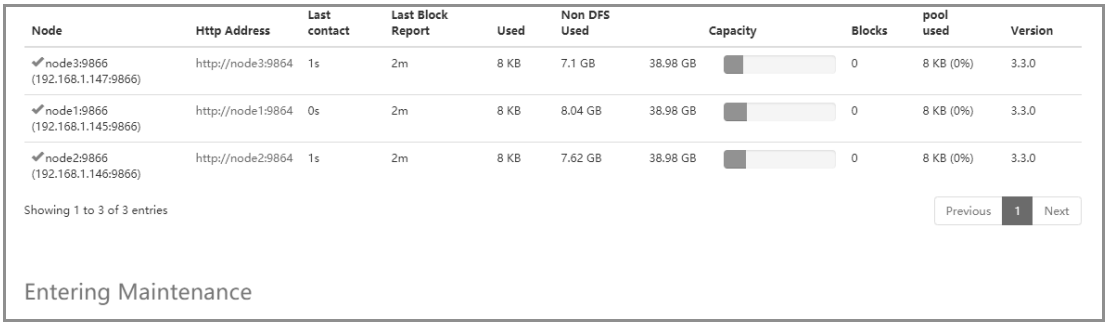
图 3-8　查看服务端口和服务端口信息

3.14　Hadoop Web 测试

根据以上配置，可以成功部署 Hadoop 大数据平台，访问 node1 9870 端口（URL：http://192.168.1.145:9870/），如图 3-9 所示。



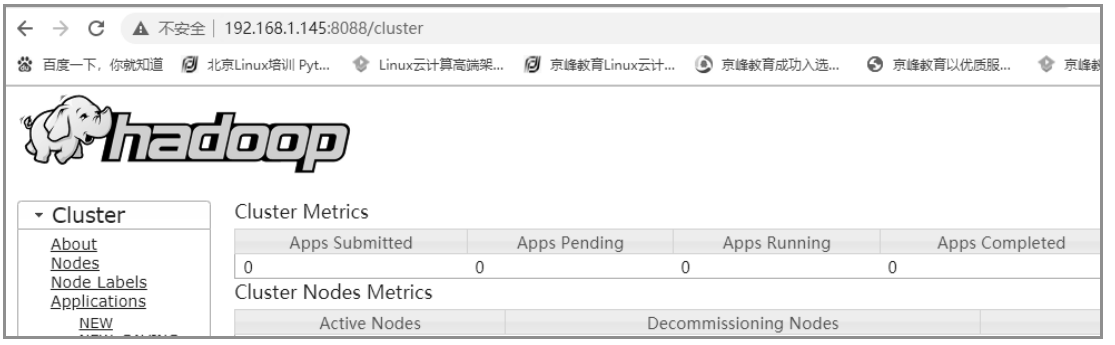
（a）



（b）

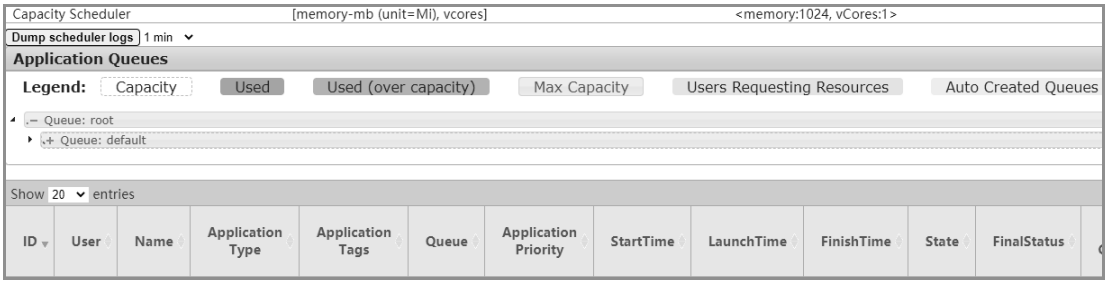
图 3-9　Hadoop Web 平台效果

访问 Hadoop 集群 Web 地址：http://192.168.1.145:8088/，如图 3-10 所示。



（a）

图 3-10　Hadoop 集群 Web 界面



（b）

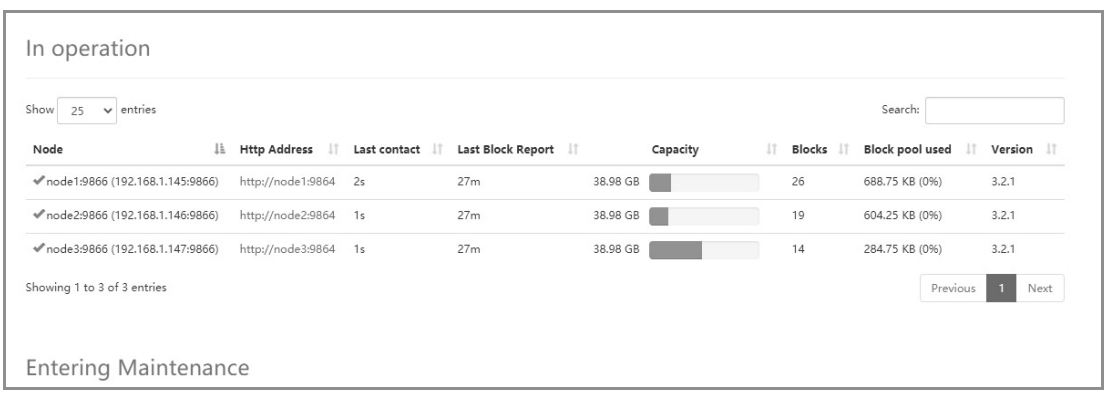
图 3-10　（续）

至此，Hadoop 大数据集群搭建完毕。

3.15　Hadoop 命令实战

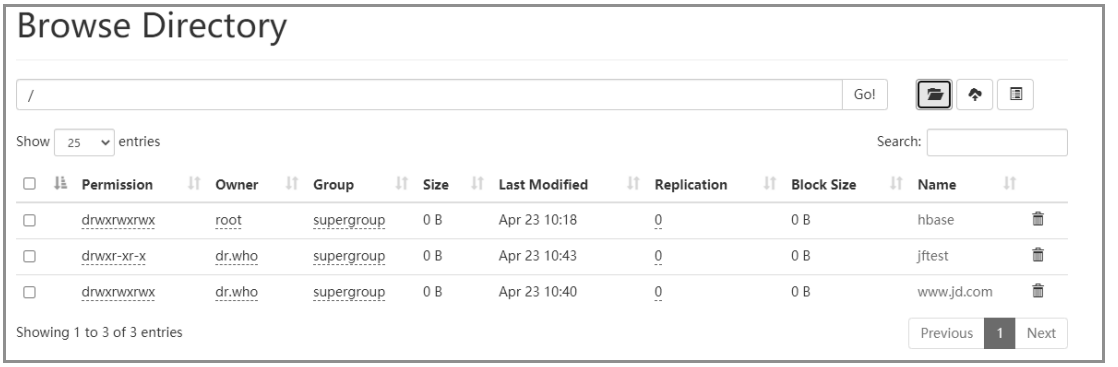
Hadoop 在生产环境中使用时，业务系统或者网站可以通过 Web 接口访问 HDFS 分布式文件系统。NameNode 和 DataNode 各自启动了一个内置的 Web 服务器，显示了集群当前的基本状态和信息。

默认配置下，NameNode 的首页地址是 http://namenode-name:8088/。这个页面列出了集群里的所有 DataNode 和集群的基本状态。这个 Web 接口也可以用来浏览整个文件系统，如图 3-11 所示。



（a）

图 3-11　浏览节点状态与文件系统



（b）

图 3-11 （续）

除了使用 Web 接口，还可以使用 Shell 命令行访问，Hadoop 包括一系列的类 Shell 命令，可直接与 HDFS 及其他 Hadoop 支持的文件系统交互。bin/hadoop fs–help 命令可以列出所有 Hadoop Shell 支持的命令。

这些命令支持大多数普通文件系统的操作，如复制文件、改变文件权限等。它还支持一些 HDFS 特有的操作，如改变文件副本数目。调用文件系统（FS）Shell 命令应使用 bin/hadoop fs <args>的形式。所有的 FS Shell 命令使用 URI 路径作为参数。

URI 格式是 scheme://authority/path。对于 HDFS 文件系统，scheme 是 hdfs；对于本地文件系统，scheme 是 file。其中，scheme 和 authority 参数都是可选的，如果未指定，就会使用配置中的默认 scheme。

一个 HDFS 文件或目录如/parent/child 可以表示为 hdfs://namenode:namenodeport/parent/child，或者更简单的/parent/child（假设配置文件中的默认值是 namenode:namenodeport）。大多数 FS Shell 命令与 UNIX Shell 命令类似，不同之处会在下面介绍各命令使用详情时指出。出错信息会输出到 stderr，其他信息输出到 stdout，如图 3-12 所示。

```
[root@node1 ~]# hadoop fs -mkdir hdfs://node1:9000/jftest/
[root@node1 ~]#
[root@node1 ~]# hadoop fs -ls hdfs://node1:9000/
Found 2 items
drwxr-xr-x   - root supergroup          0 2021-04-23 10:18 hdfs://node1:9000/hbase
drwxr-xr-x   - root supergroup          0 2021-04-23 10:35 hdfs://node1:9000/jftest
[root@node1 ~]#
[root@node1 ~]# hadoop fs -chmod 777 -R hdfs://node1:9000/
chmod: `-R': No such file or directory
[root@node1 ~]# hadoop fs -chmod -R 777 hdfs://node1:9000/
[root@node1 ~]# hadoop fs -chmod -R 777 hdfs://node1:9000/
[root@node1 ~]#
[root@node1 ~]# hadoop fs -chmod -R 777 hdfs://node1:9000/www.jd.com/
[root@node1 ~]#
```

图 3-12 Hadoop 的 Shell 命令

① cat

使用方法：hadoop fs –cat URI [URI …]

含义：将路径指定文件的内容输出到 stdout。

示例如下：

```
hadoop fs -cat hdfs://host1:port1/file1 hdfs://host2:port2/file2
hadoop fs -cat file:///file3 /user/hadoop/file4
```

返回值：成功返回 0，失败返回–1。

② chgrp

使用方法：hadoop fs –chgrp [–R] GROUP URI [URI …]

含义：改变文件所属的组。使用–R 将使改变在目录结构下递归进行。命令的使用者必须是文件的所有者或者超级用户，更多的信息请参见 HDFS 权限用户指南。

③ chmod

使用方法：hadoop fs –chmod [–R] <MODE[,MODE]… | OCTALMODE> URI [URI …]

含义：改变文件的权限。使用–R 将使改变在目录结构下递归进行，命令的使用者必须是文件的所有者或者超级用户。更多的信息请参见 HDFS 权限用户指南。

④ chown

使用方法：hadoop fs –chown [–R] [OWNER][:[GROUP]] URI [URI]

含义：改变文件的拥有者。使用–R 将使改变在目录结构下递归进行，命令的使用者必须是超级用户。更多的信息请参见 HDFS 权限用户指南。

⑤ copyFromLocal

使用方法：hadoop fs –copyFromLocal <localsrc> URI

含义：除了限定源路径是一个本地文件外，与 put 命令相似。

⑥ copyToLocal

使用方法：hadoop fs –copyToLocal [–ignorecrc] [–crc] URI <localdst>

含义：除了限定目标路径是一个本地文件外，与 get 命令类似。

⑦ cp

使用方法：hadoop fs –cp URI [URI …] <dest>

含义：将文件从源路径复制到目标路径。这个命令允许有多个源路径，此时目标路径必须是一个目录。

示例如下：

```
hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2
hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2 /user/hadoop/dir
```

返回值：成功返回 0，失败返回-1。

⑧ du

使用方法：hadoop fs -du URI [URI …]

含义：显示目录中所有文件的大小，或者当只指定一个文件时，显示此文件的大小。

示例如下：

```
hadoop fs -du /user/hadoop/dir1 /user/hadoop/file1 hdfs://host:port/
user/hadoop/dir1
```

返回值：成功返回 0，失败返回-1。

⑨ dus

使用方法：hadoop fs -dus <args>

含义：显示文件的大小。

⑩ expunge

使用方法：hadoop fs -expunge

含义：清空回收站。请参考 HDFS 设计文档以获取更多关于回收站特性的信息。

⑪ get

使用方法：hadoop fs -get [-ignorecrc] [-crc] <src> <localdst>

含义：复制文件到本地文件系统。可用-ignorecrc 选项复制 CRC 校验失败的文件。使用-crc 选项复制文件以及 CRC 信息。

示例如下：

```
hadoop fs -get /user/hadoop/file localfile
hadoop fs -get hdfs://host:port/user/hadoop/file localfile
```

返回值：成功返回 0，失败返回-1。

⑫ getmerge

使用方法：hadoop fs -getmerge <src> <localdst> [addnl]

含义：接收一个源目录和一个目标文件作为输入，并且将源目录中所有的文件连接成本地目标文件。addnl 是可选的，用于指定在每个文件结尾添加一个换行符。

⑬ ls

使用方法：hadoop fs –ls <args>

含义：如果是文件，则按照以下格式返回文件信息。

```
文件名 <副本数> 文件大小 修改日期 修改时间 权限 用户 ID 组 ID
```

如果是目录，则返回它直接子文件的一个列表，就像在 UNIX 中一样。目录返回列表的信息如下：

```
目录名 <dir> 修改日期 修改时间 权限 用户 ID 组 ID
```

示例如下：

```
hadoop fs -ls /user/hadoop/file1 /user/hadoop/file2 hdfs://host:port/user/
hadoop/dir1 /nonexistentfile
```

返回值：成功返回 0，失败返回–1。

⑭ lsr

使用方法：hadoop fs –lsr <args>

含义：ls 命令的递归版本，类似于 UNIX 中的 ls –R。

⑮ mkdir

使用方法：hadoop fs –mkdir <paths>

含义：接收路径指定的 URI 作为参数，创建这些目录。其行为类似于 UNIX 的 mkdir –p，它会创建路径中的各级父目录。

示例如下：

```
hadoop fs -mkdir /user/hadoop/dir1 /user/hadoop/dir2
hadoop fs -mkdir hdfs://host1:port1/user/hadoop/dir hdfs://host2:port2/
user/hadoop/dir
```

返回值：成功返回 0，失败返回–1。

⑯ movefromLocal

使用方法：dfs –moveFromLocal <src> <dst>

含义：输出一个"not implemented"信息。

⑰ mv

使用方法：hadoop fs –mv URI [URI …] <dest>

含义：将文件从源路径移动到目标路径。这个命令允许有多个源路径，此时目标路径必须是一个目录。不允许在不同的文件系统间移动文件。

示例如下：

```
hadoop fs -mv /user/hadoop/file1 /user/hadoop/file2
hadoop fs -mv hdfs://host:port/file1 hdfs://host:port/file2 hdfs://host:
port/file3 hdfs://host:port/dir1
```

返回值：成功返回 0，失败返回-1。

⑱ put

使用方法：hadoop fs -put <localsrc> … <dst>

含义：从本地文件系统中复制单个或多个源路径到目标文件系统。也支持从标准输入中读取输入写入目标文件系统。

示例如下：

```
hadoop fs -put localfile /user/hadoop/hadoopfile
hadoop fs -put localfile1 localfile2 /user/hadoop/hadoopdir
hadoop fs -put localfile hdfs://host:port/hadoop/hadoopfile
hadoop fs -put - hdfs://host:port/hadoop/hadoopfile
```

返回值：成功返回 0，失败返回-1。

⑲ rm

使用方法：hadoop fs -rm URI [URI …]

含义：删除指定的文件。只删除非空目录和文件，请参考 rmr 命令了解递归删除。

示例如下：

```
hadoop fs -rm hdfs://host:port/file /user/hadoop/emptydir
```

返回值：成功返回 0，失败返回-1。

⑳ rmr

使用方法：hadoop fs -rmr URI [URI …]

含义：delete 的递归版本。

示例如下：

```
hadoop fs -rmr /user/hadoop/dir
hadoop fs -rmr hdfs://host:port/user/hadoop/dir
```

返回值：成功返回 0，失败返回-1。

㉑ setrep

使用方法：hadoop fs -setrep [-R] <path>

含义：改变一个文件的副本系数。-R 选项用于递归改变目录下所有文件的副本系数。

示例如下：

```
hadoop fs -setrep -w 3 -R /user/hadoop/dir1
```

返回值：成功返回 0，失败返回–1。

㉒ stat

使用方法：hadoop fs –stat URI [URI …]

含义：返回指定路径的统计信息。

示例如下：

```
hadoop fs -stat path
```

返回值：成功返回 0，失败返回–1。

㉓ tail

使用方法：hadoop fs –tail [–f] URI

含义：将文件尾部 1KB 的内容输出到 stdout。支持–f 选项，作用和 UNIX 中的一致。

示例如下：

```
hadoop fs -tail pathname
```

返回值：成功返回 0，失败返回–1。

㉔ test

使用方法：hadoop fs –test –[ezd] URI

选项：

–e 检查文件是否存在，如果存在则返回 0。

–z 检查文件是否是 0B，如果是则返回 0。

–d 如果路径是个目录，则返回 1，否则返回 0。

示例如下：

```
hadoop fs -test -e filename
```

㉕ text

使用方法：hadoop fs –text <src>

含义：将源文件输出为文本格式。允许的格式是 zip 和 TextRecordInputStream。

㉖ touchz

使用方法：hadoop fs –touchz URI [URI …]

含义：创建一个 0B 的空文件。

示例如下：

```
hadoop -touchz pathname
```

返回值：成功返回 0，失败返回-1。

3.16 Hadoop 节点扩容

随着公司业务不断发展，数据量也越来越大，此时需要对 Hadoop 集群规模进行扩容，在现有 Hadoop 3 台集群的基础上动态增加 node4 节点上的 DataNode 与 NodeManager。操作方法和步骤如下。

3.16.1 hosts 及防火墙设置

扩容 node4 节点时，需要对 node1、node2、node3、node4 节点进行添加 hosts、同步时间、关闭防火墙、SELinux、修改主机名等操作，操作命令如下：

```
cat >/etc/hosts<<EOF
127.0.0.1  localhost localhost.localdomain
192.168.1.145 node1
192.168.1.146 node2
192.168.1.147 node3
192.168.1.148 node4
EOF
sed -i '/SELINUX/s/enforcing/disabled/g'  /etc/sysconfig/selinux
setenforce  0
systemctl   stop    firewalld.service
systemctl   disable   firewalld.service
yum install ntpdate rsync lrzsz -y
ntpdate  pool.ntp.org
hostname 'cat /etc/hosts|grep $(ifconfig|grep broadcast|awk '{print $2}')
|awk '{print $2}'';su
```

3.16.2 配置节点免密钥登录

node1 节点作为 Master 控制节点，执行以下命令创建公钥和私钥，然后将公钥复制至其余节点。

```
ssh-copy-id -i /root/.ssh/id_rsa.pub root@node4
```

3.16.3　配置节点 Java 环境

```
#解压 JDK 软件包
tar  -xvzf  jdk1.8.0_131.tar.gz
#创建 JDK 部署目录
mkdir -p  /usr/java/
\mv jdk1.8.0_131 /usr/java/
#设置环境变量
cat>>/etc/profile<<EOF
export JAVA_HOME=/usr/java/jdk1.8.0_131/
export HADOOP_HOME=/data/hadoop/
export JAVA_LIBRARY_PATH=/data/hadoop/lib/native/
export PATH=\$PATH:\$HADOOP_HOME/bin/:\$JAVA_HOME/bin
EOF
#使其环境变量生效
source /etc/profile
java -version
```

3.16.4　Hadoop 服务部署

将 node1 部署完成的 Hadoop 所有文件、目录同步至 node2 和 node3 节点，操作命令如下：

```
for i in node4;do ssh -l root $i -a "mkdir -p /data/hadoop/" ;done
for i in node4;do rsync -aP --delete /data/hadoop/ root@$i:/data/
hadoop/ ;done
for i in node4;do ssh -l root $i -a "rm -rf /data/hadoop/data* ";done
```

3.16.5　添加 Hadoop 新节点

（1）动态添加 DataNode 和 NodeManager 节点，查看现有 HDFS 各节点状态，操作命令如下，如图 3-13 所示。

```
hdfs dfsadmin -report
```

可以看到添加 DataNode 节点之前，DataNode 节点总共有 3 个，分别在 node1、node2 和 node3 服务器上。

（2）查看 YARN 各节点状态，操作命令如下：

```
yarn node -list
```

添加 NodeManager 之前，NodeManager 进程运行在 node1、node2 和 node3 服务器上。

```
[root@node1 ~]# hdfs dfsadmin -report
Configured Capacity: 125558587392 (116.94 GB)
Present Capacity: 85934138440 (80.03 GB)
DFS Remaining: 85906719716 (80.01 GB)
DFS Used: 27418724 (26.15 MB)
DFS Used%: 0.03%
Replicated Blocks:
        Under replicated blocks: 0
        Blocks with corrupt replicas: 0
        Missing blocks: 0
        Missing blocks (with replication factor 1): 0
        Low redundancy blocks with highest priority to recover: 0
        Pending deletion blocks: 0
```

图 3-13　Hadoop HDFS 查看报告信息

（3）添加 DataNode 和 NodeManager 节点，在所有服务器的 Hadoop workers 文件中添加 node4 节点，操作命令如下，结果如图 3-14 所示。

```
echo node4 >>/data/hadoop/etc/hadoop/workers ;cat /data/hadoop/etc/hadoop/
workers
```

```
[root@node1 ~]# ls /data/hadoop/etc/hadoop/workers
/data/hadoop/etc/hadoop/workers
[root@node1 ~]# cat /data/hadoop/etc/hadoop/workers
node1
node2
node3
[root@node1 ~]# echo node4 >>/data/hadoop/etc/hadoop/workers ;cat /data/ha
node1
node2
node3
node4
[root@node1 ~]#
[root@node1 ~]#
[root@node1 ~]#
```

图 3-14　Hadoop 新增节点实战

（4）在 node4 新增节点服务器上启动 DataNode 和 NodeManager 服务，操作命令如下，如图 3-15 所示。

```
hdfs --daemon start datanode
yarn --daemon start nodemanager
```

（5）在 node1 服务器上执行以下命令，刷新 Hadoop 集群节点，操作命令如下：

```
hdfs dfsadmin -refreshNodes
/data/hadoop/sbin/start-balancer.sh
```

（6）再次查看集群的状态，查看 HDFS 各节点状态，操作命令如下：

```
hdfs dfsadmin -report
```

```
[root@node4 ~]# ps -ef|grep hadoop
root     13947     1  5 11:47 pts/0    00:00:09 /usr/java/jdk1.8.0_131//bin/jav
adoop.security.logger=ERROR,RFAS -Dyarn.log.dir=/data/hadoop//logs -Dyarn.log.f
ata/hadoop/ -Dyarn.root.logger=INFO,console -Djava.library.path=/data/hadoop/li
data/hadoop//logs -Dhadoop.log.file=hadoop-root-datanode-node4.log -Dhadoop.hom
t.logger=INFO,RFA -Dhadoop.policy.file=hadoop-policy.xml org.apache.hadoop.hdfs
root     14046     1  6 11:47 pts/0    00:00:11 /usr/java/jdk1.8.0_131//bin/jav
-Dyarn.log.dir=/data/hadoop//logs -Dyarn.log.file=hadoop-root-nodemanager-node4
=INFO,console -Djava.library.path=/data/hadoop/lib/native/:/data/hadoop//lib/na
.file=hadoop-root-nodemanager-node4.log -Dhadoop.home.dir=/data/hadoop/ -Dhadoo
policy.file=hadoop-policy.xml -Dhadoop.security.logger=INFO,NullAppender org.ap
root     14160 13772  0 11:50 pts/0    00:00:00 grep --color=auto hadoop
[root@node4 ~]# jps
14161 Jps
13947 DataNode
14046 NodeManager
```

图 3-15　启动 DataNode 和 NodeManager 服务

可以看到，添加 DataNode 节点后，输出的结果中存在 node4 服务器上的 DataNode 节点，说明 DataNode 节点添加成功，如图 3-16 所示。

```
Name: 192.168.1.148:9866 (node4)
Hostname: node4
Decommission Status : Normal
Configured Capacity: 41852862464 (38.98 GB)
DFS Used: 8192 (8 KB)
Non DFS Used: 14248042496 (13.27 GB)
DFS Remaining: 27604811776 (25.71 GB)
DFS Used%: 0.00%
DFS Remaining%: 65.96%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
```

图 3-16　添加 node4 节点后的输出结果

3.16.6　删除 Hadoop 节点

随着公司业务的不断发展，Hadoop 服务器使用年限增长，难免要淘汰一些服务器，此时需要对 Hadoop 集群规模进行缩容，在现有 Hadoop 集群（4 台）的基础上动态删除 node4 服务器上的 DataNode 与 NodeManager 节点。操作方法和步骤如下。

（1）要想删除 node4 上 DataNode 与 NodeManager 节点，需要先停止 DataNode 和 NodeManager 进程，操作命令如下，结果如图 3-17 所示。

```
hdfs --daemon stop datanode
yarn --daemon stop nodemanager
```

```
ps -ef|grep hadoop|grep -v grep|awk '{print $2}'|xargs kill -9
```

```
[root@node4 ~]# hdfs --daemon stop datanode
[root@node4 ~]# yarn --daemon stop nodemanager
WARNING: nodemanager did not stop gracefully after 5 seconds: Trying to kill with kil
[root@node4 ~]#
[root@node4 ~]# ps -ef|grep hadoop|grep -v grep|awk '{print $2}'|xargs kill -9

Usage:
 kill [options] <pid|name> [...]

Options:
 -a, --all              do not restrict the name-to-pid conversion to processes
                        with the same uid as the present process
 -s, --signal <sig>     send specified signal
 -q, --queue <sig>      use sigqueue(2) rather than kill(2)
```

图 3-17　停止 DataNode 和 NodeManager 进程

（2）删除每台服务器上 Hadoop 的 workers 文件中的 node4，删除后的文件内容如下，如图 3-18 所示。

```
sed -i '/^node4$/d' /data/hadoop/etc/hadoop/workers ;cat /data/hadoop/etc/
hadoop/workers
```

```
=INFO,console -Djava.library.path=/data/hadoop/lib/native/:/data/hadoop//li
.file=hadoop-root-nodemanager-node4.log -Dhadoop.home.dir=/data/hadoop/ -Dh
policy.file=hadoop-policy.xml -Dhadoop.security.logger=INFO,NullAppender or
root     14160 13772  0 11:50 pts/0    00:00:00 grep --color=auto hadoop
[root@node4 ~]# jps
14161 Jps
13947 DataNode
14046 NodeManager
[root@node4 ~]# sed -i '/^node4$/d' /data/hadoop/etc/hadoop/workers ;cat /d
node1
node2
node3
[root@node4 ~]#
```

图 3-18　删除 workers 文件中的 node4

（3）在 node1 服务器上执行以下命令，刷新 Hadoop 集群节点。

```
hdfs dfsadmin -refreshNodes/data/hadoop/sbin/start-balancer.sh
```

（4）查看 HDFS 各节点状态，操作命令如下：

```
hdfs dfsadmin -report
```

在输出的信息中没有 node4 服务器上的 DataNode 节点，说明 node4 服务器上的 DataNode 节点删除成功。

（5）查看 YARN 各节点状态，操作命令如下：

```
yarn node -list
```

在输出的信息中没有 node4 服务器上的 NodeManager 节点，说明 node4 服务器上的 NodeManager 节点删除成功。

还可以动态删除 DataNode 节点与 NodeManager 节点，这种方式不需要删除 workers 文件中现有的 node4 配置。

（1）在 node1 节点上修改 hdfs-site.xml 文件，适当减小 dfs.replication 副本数，增加 dfs.hosts.exclude 配置如下：

```
<property>
    <name>dfs.hosts.exclude</name>
<value>/data/hadoop/etc/hadoop/excludes</value>
</property>
```

（2）在 node1 服务器上的/data/hadoop/etc/hadoop/目录下创建 excludes 文件，将要删除的 node4 服务器节点的主机名或 IP 地址配置到这个文件中，具体如下：

```
vim /data/hadoop/etc/hadoop/excludes
node4
```

（3）刷新节点，在 node1 服务器上执行以下命令，刷新 Hadoop 集群节点。

```
hdfs dfsadmin -refreshNodes
/data/hadoop/sbin/start-balancer.sh
```

这种方式也可以实现动态删除 DataNode 和 NodeManager 节点，如图 3-19 所示。

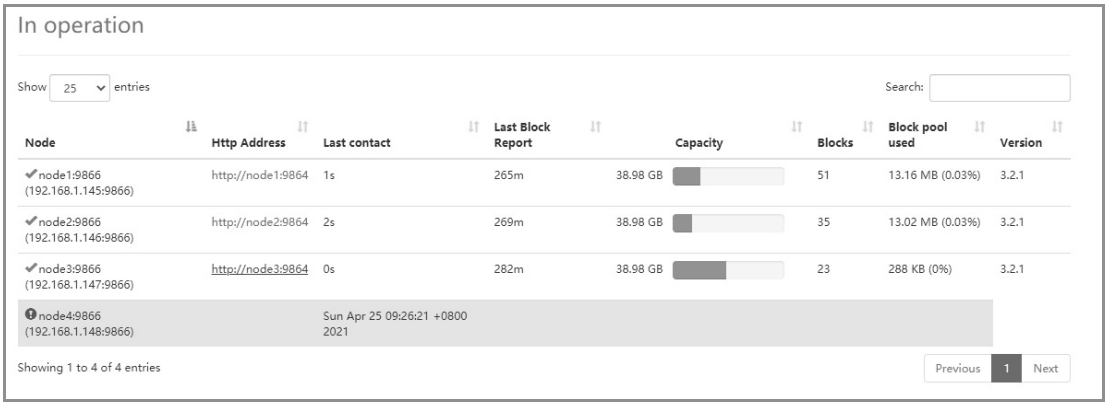


图 3-19　Hadoop 动态删除节点

3.17 HBase 概念剖析

HBase 是 Hadoop Database 的简称，本质上来说就是 Hadoop 系统的数据库，为 Hadoop 框架中的结构化数据提供存储服务，是面向列的分布式数据库。HBase 与 HDFS 不同，HDFS 是分布式文件系统，管理的是存放在多个硬盘上的数据文件，而 HBase 管理的是类似于 Key-Value 映射的表，如图 3-20 所示。

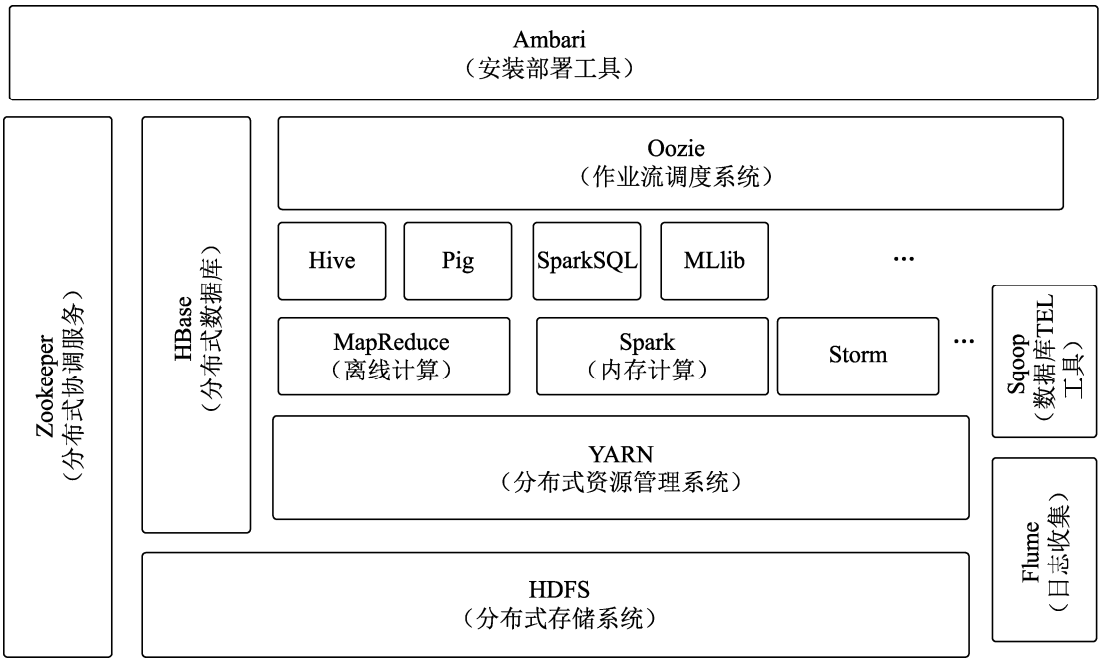


图 3-20　大数据服务组件图

Hadoop 分布式集群主要有 3 个核心部分：FS（Hadoop 分布式文件系统）、计算框架（MapReduce）和管理桥（Yet Another Resource Negotiator）。

HDFS 允许在分布式（提供更快的读/写访问）和冗余方式（提供更好的可用性）中存储大量数据。

MapReduce 允许以分布式和并行方式处理这些大规模数据，不局限于 HDFS。作为文件系统，HDFS 缺乏随机读/写的能力，它适用于顺序数据访问，这就是 HBase 出现的原因。它是 NoSQL 数据库，在 Hadoop 集群上运行，并为开发人员提供对数据的随机实时读/写访问。

开发人员可以将结构化和非结构化数据分别存储在 Hadoop 和 HBase 中，它们都提供了多种

访问数据的机制，如 Shell 和其他 API。HBase 以列的形式将数据存储为 Key-Value 对，HDFS 则将数据存储为二维表。

HBase 底层仍然依赖 HDFS 作为其物理存储，还需要 Zookeeper 协助提供部分配置服务，包括维护元数据和命名空间等。

传统的关系型数据库中，数据按照行进行存储，在进行查询时，需要使用大量的 I/O，数据库必须快速扩展才能满足性能要求。但当面对大规模数据处理任务时，计算性能会受到极大的影响，完全不能满足大数据处理的需求。

而 HBase 的数据存储，数据是按列存储的，查询时只访问所涉及的列，大量降低系统 I/O，数据类型一致，可以高效压缩存储。

HBase 中有个概念 Column Family，简称 CF，一般用于将相关的列组合起来。上述例子中，“姓”和“名”组合成为 info，组合的形式是类似于字典的 Key-Value 形式。在物理上，HBase 其实是按 CF 存储的，只是按照 Row-key 将相关 CF 中的列关联起来。

总体来说，HBase 对数据的存储方式和数据结构进行的修改和规整，使其更加善于去处理大数据的场景，因此在 Hadoop MapReduce 运行计算时能够提供更好的底层支持。

当然，HBase 并非完美，限于这样的数据结构和存储方式，HBase 只能做简单的 Key-value 查询，不支持复杂的统计 SQL。

相信看完以上的内容，关于 Hadoop 和 HBase 的关系，大家都能基本了解了。Hadoop 框架基于分布式文件系统 HDFS 和分布式数据库 HBase，保证了稳定可靠的底层支持，为后续的数据处理提供了保障。

Hadoop 与 HBase 的一些显著特征如下。

1. Hadoop

（1）对大型文件的流式访问方面进行了优化。

（2）遵循一次性写-读的意识形态。

（3）不支持随机读/写。

2. HBase

（1）以列的方式将数据存储为 Key-Value 对（列作为列族聚集在一起）。

（2）对大型数据集中的少量数据访问会延迟。

（3）提供灵活的数据模型。

Hadoop 最适合批量处理离线数据，而 HBase 适合处理实时数据流。

3.18 HBase 应用场景

HBase 解决不了所有的问题，但是对于具有某些特点的数据可以使用 HBase 高效处理，如以下的应用场景。

（1）数据模式是动态的或者可变的，且支持半结构化和非结构化的数据。

（2）数据库中的很多列都包含了较多空字段，在 HBase 中，空字段不会像在关系型数据库中一样占用空间。

（3）需要很高的吞吐量，瞬间写入量很大。

（4）数据要维护很多版本，可以使用 HBase，HBase 利用时间戳来区分不同版本的数据。

（5）具有高可扩展性，能动态地扩展整个存储系统。

在实际应用中，有很多公司使用 HBase，如 Facebook 公司的 Social Inbox 系统使用 HBase 作为消息服务的基础存储设施，每月可处理几千亿条消息；Yahoo 公司使用 HBase 存储检查近似重复的指纹信息的文档，它的集群中分别运行着 Hadoop 和 HBase，表中存了上百万行数据；Adobe 公司使用 Hadoop+HBase 的生产集群，将数据直接持续地存储在 HBase 中，并将 HBase 作为数据源进行 MapReduce 的作业处理；Apache 公司使用 HBase 来维护 Wiki 的相关信息。

下面通过几个案例来介绍 HBase 的实际应用。

3.18.1 搜索引擎应用

HBase 是 Google Bigtable 的开源实现，而 Google 公司开发 Bigtable 是为了它的搜索引擎应用。Google 等搜索引擎是基于索引来完成快速搜索服务的，该索引提供了特定词语，包含该词语的所有文档的映射。

搜索引擎的文档库是整个互联网，搜索的特定词语就是用户搜索框里输入的任何信息，Bigtable 和开源的 HBase 为这种文档库提供存储功能并按行访问。下面简单地分析 HBase 在搜索引擎中的应用逻辑。

首先，网络爬虫持续不断地从网络上抓取新页面，并将页面内容存储到 HBase 中，爬虫可以更新 HBase 中的数据表；然后，用户可以利用 MapReduce 在整张表上计算并生成索引，为网络搜索做准备；接着，用户发起搜索请求；最后，搜索引擎查询建立好的索引列表，获取文档索引后，再从 HBase 中获取所需的文档内容，最后将搜索结果呈现给用户。

3.18.2　捕获增量数据

数据通常是动态增加的，随着时间的推移，数据量会越来越大，如网站的日志、邮箱的邮件等。通常通过采集工具捕获来自各种数据源的增量数据，再使用 HBase 进行存储。

例如，这种采集工具可能是网页爬虫，采集的数据源可能是记录用户点击的广告信息、驻留的时间长度及对应的广告效果数据，也可能是记录服务器运行的各种参数数据。

下面介绍一些与该使用场景有关的成功案例。

3.18.3　存储监控参数

大型的、基于 Web 的产品后台一般都拥有成百上千台服务器，这些服务器不仅需要为前端的大量用户提供服务，还需要实现日志采集、数据存储、数据处理等功能。

为了保证产品的正常运行，监控服务器和服务器上运行的软件的健康状态是至关重要的。大规模监控整个环境需要能够采集和存储来自不同数据源的各种参数的监控系统。OpenTSDB（Open Time Series Database，开放时间序列数据库）正是这种监控系统，它可以从大规模集群中获取相应的参数并进行存储、索引和服务。

OpenTSDB 框架使用 HBase 作为核心平台来存储和检索所收集的参数，可以灵活地支持增加参数，也可以支持采集上万台机器和上亿个数据点，具有高可扩展性。

OpenTSDB 作为数据收集和监控系统，一方面能够存储和检索参数数据并将其保存很长时间，另一方面如果增加功能也可以添加各种新参数。最终使用 OpenSTDB 对 HBase 中存储的数据进行分析，并以图形化方式展示集群中的网络设备、操作系统及应用程序的状态。

3.18.4　存储用户交互数据

基于 Web 的应用还有一种很重要的数据，即用户交互数据。这一类数据包含了用户访问网站的行为习惯。通过分析用户交互数据，就可以获取用户在网站上的活动信息。例如，用户看了什么？某个按钮被用户单击了多少次？用户最近搜索了什么？从这些信息就可以了解用户的需求，从而针对不同的用户提供不同的应用。

例如，Facebook 里的 Like 按钮，每次用户单击该按钮“喜欢”一个特定主题，计数器就增加一次。Facebook 使用 HBase 的计数器来计量人们喜欢特定网页的次数。内容原创人或网页主人可以得到近乎实时的用户喜欢他们网页的数据。他们可以据此更敏捷地判断应该提供什

么内容。

Facebook 为此创建了一个名为 Facebook Insight 的系统，该系统需要一个可扩展的存储系统。公司考虑了很多种可能，包括关系型数据库、内存数据库和 Cassandra 数据库，最后决定使用 HBase。基于 HBase，Facebook 可以很方便地横向扩展服务规模，提供给数百万用户，也可以继续使用他们已有的运行大规模 HBase 机群的经验。该系统每天处理数百亿条事件，记录数百个参数。

3.18.5 存储遥测数据

软件在运行时经常会出现异常，这时大部分软件都会生成一个软件崩溃报告，该报告会返回给软件开发者，便于软件开发者评测软件质量，以及规划软件开发路线图。

例如，FireFox 网络浏览器是 Mozilla 基金会旗下的产品，支持各种操作系统，全世界数百万台计算机上都有它的身影。当 FireFox 浏览器崩溃时，会返回一个软件崩溃报告给 Mozilla。

Mozilla 使用一个叫作 Socorro 的系统收集这些报告，用来指导研发部门研制更稳定的产品。Socorro 系统的数据存储和分析构建在 HBase 上，采用 HBase 使得基本分析可以用到比以前多得多的数据。用这些分析数据指导 Mozilla 的开发人员，使其更有针对性地研制出 Bug 更少的版本。

趋势科技（TrendMicro）为企业客户提供互联网安全解决方案，来应对网络上千变万化的安全威胁。安全的重要环节是感知，日志的收集和分析对于这种感知能力是至关重要的。

趋势科技使用 HBase 来收集和分析日志活动，每天可收集数十亿条记录。HBase 中灵活的模式支持可变的数据结构，当分析流程重新调整时，可以增加新属性。

3.18.6 广告效果和点击流

在线广告是互联网产品的一项主要收入来源。互联网企业给用户提供免费服务，在用户使用服务时投放广告给目标用户。这种精准投放需要针对用户的交互数据做详细的捕获和分析，以理解用户的特征；再基于这种特征，选择并投放广告。企业可使用精细的用户交互数据建立更优的模型，进而获得更好的广告投放效果和更多的收入。

但这类数据以连续流的形式出现，很容易按用户划分。在理想情况下，这种数据一旦产生就能够马上使用。

HBase 非常适合收集这种用户交互数据，并已经成功地应用在相关领域。它可以增量捕获

第一手点击流和用户交互数据，然后用不同处理方式来处理数据，电商和广告监控行业都已经非常熟练地使用此类技术。

例如，淘宝的实时个性化推荐服务将中间推荐结果存储在 HBase 中，与广告相关的用户建模数据也存储在 HBase 中，用户模型多种多样，可以用于多种不同场景。例如，针对特定用户投放什么广告，用户在电商门户网站上购物时是否实时报价等。

HBase 已成熟地应用于国内外的很多大型公司，总之，HBase 适合用来存储各种类型的大规模数据，既支持实时的在线查询，也支持离线的应用服务。但对于需要连接和其他一些关系型数据特性要求时，HBase 就不适用了，因此，还是要根据应用场景选择是否用 HBase，发挥 HBase 的优势。

3.19　HBase 分布式集群实战

（1）下载 HBase 软件并配置，操作命令如下：

```
wget -c http://archive.apache.org/dist/hbase/2.2.4/hbase-2.2.4-bin.tar.gz
tar -xzf hbase-2.2.4-bin.tar.gz
mkdir -p /data/\mv hbase-2.2.4 /data/hbase/
useradd hadoop
chown -R hadoop:hadoop /data/hbase/
```

（2）在 Hadoop 配置的基础上，配置环境变量 HBASE_HOME，编辑 vim /etc/profile 文件，在末尾加入以下代码：

```
export HBASE_HOME=/data/hbase/
export  PATH=$PATH:$HBASE_HOME/bin
```

（3）执行 vi /data/hbase/conf/hbase-env.sh 操作命令，在配置文件中加入以下代码：

```
export JAVA_HOME=/usr/java/jdk1.8.0_131/
```

（4）执行 vi /data/hbase/conf/hbase-site.xml 操作命令，在配置文件中修改，代码如下：

```
<configuration>
       <property>
              <name>hbase.rootdir</name>
              <value>hdfs://node1:9000/hbase</value>
       </property>
       <property>
```

```
        <name>hbase.cluster.distributed</name>
        <value>true</value>
    </property>
    <property>
        <name>hbase.master.port</name>
        <value>60000</value>
    </property>
    <property>
        <name>hbase.zookeeper.quorum</name>
        <value>node1:2181,node2:2181,node3:2181</value>
    </property>
    <property>
        <name>hbase.zookeeper.property.dataDir</name>
     <value>/usr/local/zookeeper/data</value>
    </property>
    <property>
        <name>hbase.unsafe.stream.capability.enforce</name>
    <value>false</value>
    </property>
</configuration>
```

（5）配置 Regionservers 文件，执行 vi /data/hbase/conf/regionservers 操作命令，去掉默认的 localhost，加入以下代码：

```
node1
node2
node3
```

（6）将 node1 部署完成的 HBase 所有文件、目录同步至 node2 和 node3 节点，操作命令如下：

```
for i in 'seq 2 3';do rsync -av /data/hbase root@node$i:/data/ ;done
```

3.20 HBase 集群测试及故障排错

（1）启动与停止 HBase 服务，操作命令如下，结果如图 3-21 所示。

```
/data/hbase/bin/start-hbase.sh
```

如果 Master 上出现 HMaster、HQuormPeer，Slave 上出现 HRegionServer、HQuorumPeer，则表示启动成功。

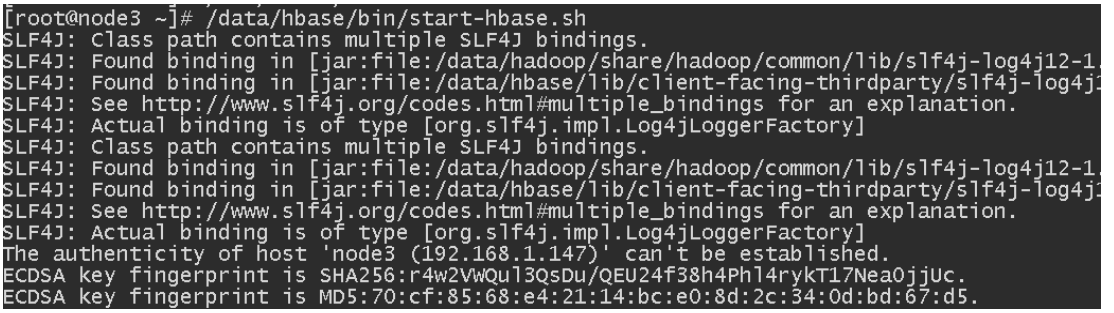


图 3-21　启动 HBase 服务

HBase 有内置的 Zookeeper，如果没有安装 Zookeeper，当启动 HBase 时会有一个 HQuorumPeer 进程。

如果用外置的 Zookeeper 管理 HBase，则先启动 Zookeeper，然后启动 HBase，启动后会有一个 QuorumPeerMain 进程。HQuorumPeer 表示使用的是 HBase 管理的 Zookeeper。QuorumPeerMain 表示使用的是 Zookeeper 独立的进程，二选一即可，结果如图 3-22 所示。

图 3-22　HBase 服务实战操作

（2）执行 hbase shell 命令进入 hbase 命令模式，执行 status 命令输出结果是 1 台 master 服务器和 2 台 servers 服务器全部成功启动。

```
/data/hbase/bin/hbase shell
```

（3）当执行 status 命令，出现图 3-23 所示的报错信息时，说明 Hadoop 的安全模式打开了，再重新启动 HBase 即可。操作代码如下，结果如图 3-24 所示。

```
hdfs dfsadmin -safemode leave
```

（4）当要停止 HBase 时执行 stop-hbase.sh 命令。

```
/data/hbase/bin/stop-hbase.sh
```

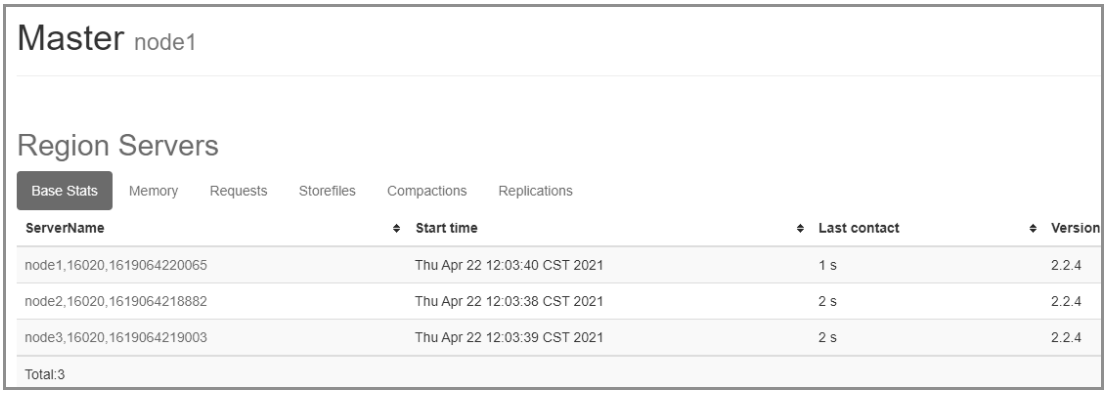
```
Took 0.0005 seconds
hbase(main):002:0> status

ERROR: org.apache.hadoop.hbase.ipc.ServerNotRunningYetException: Server is not running ye
        at org.apache.hadoop.hbase.master.HMaster.checkServiceStarted(HMaster.java:2924)
        at org.apache.hadoop.hbase.master.MasterRpcServices.isMasterRunning(MasterRpcServ
        at org.apache.hadoop.hbase.shaded.protobuf.generated.MasterProtos$MasterService$2
        at org.apache.hadoop.hbase.ipc.RpcServer.call(RpcServer.java:393)
        at org.apache.hadoop.hbase.ipc.CallRunner.run(CallRunner.java:133)
        at org.apache.hadoop.hbase.ipc.RpcExecutor$Handler.run(RpcExecutor.java:338)
        at org.apache.hadoop.hbase.ipc.RpcExecutor$Handler.run(RpcExecutor.java:318)

For usage try 'help "status"'

Took 10.2834 seconds
```

图 3-23　HBase 服务实战操作

```
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.2.4, r67779d1a325a4f78a468af3339e73bf075888bac, 2020年 03月 11日 星期三 12:57:39
Took 0.0047 seconds
hbase(main):001:0> status
1 active master, 0 backup masters, 3 servers, 0 dead, 1.0000 average load
Took 0.8564 seconds
hbase(main):002:0>
hbase(main):003:0* version
2.2.4, r67779d1a325a4f78a468af3339e73bf075888bac, 2020年 03月 11日 星期三 12:57:39 CST
Took 0.0005 seconds
hbase(main):004:0>
hbase(main):005:0* list
TABLE
Student
1 row(s)
```

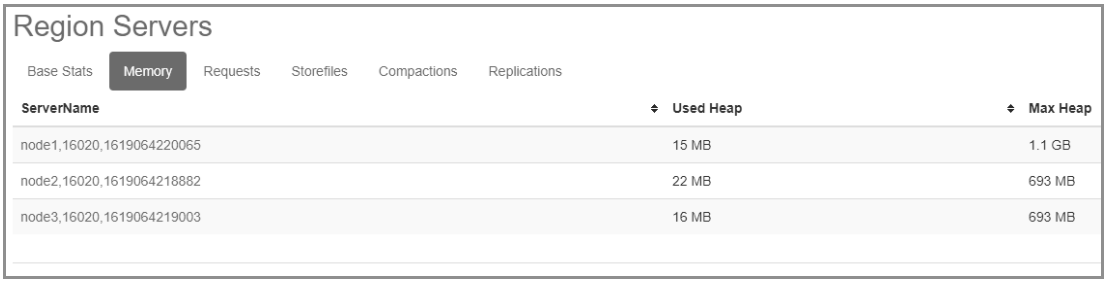
图 3-24　重新启动 HBase

（5）在浏览器访问 HBase Web 界面，URL 为 http://192.168.1.145:16010/master-status，访问结果如图 3-25 所示。



（a）节点状态界面

图 3-25　浏览器访问 HBase Web 界面



（b）节点内存界面

图 3-25　（续）

3.21　HMaster 及 RegionServer 剖析

HMaster 是 HBase 集群中的主服务器，负责监控集群中的所有 RegionServer，并且是所有元数据更改的接口。在分布式集群中，HMaster 服务器通常运行在 HDFS 的 NameNode 上。

HMaster 通过 Zookeeper 避免单点故障，在集群中可以启动多个 HMaster，但 Zookeeper 的选举机制能够保证同时只有一个 HMaster 处于 Active（活跃）状态，其他的 HMaster 处于热备份状态。

HMaster 主要负责表和 RegionServer 的管理工作，主要负责的工作内容如下。

（1）管理用户对表的增、删、改、查操作。

（2）管理 RegionServer 的负载均衡，调整 Region 的分布。

（3）负责 Region 的分配和移除。

（4）负责处理 RegionServer 的故障转移。

HMaster 提供了一些基于元数据方法的接口，便于用户与 HBase 进行交互，如表 3-1 所示。

表 3-1　HMaster提供的接口

相 关 接 口	功　　能
HBase表	创建表、删除表、启用/失效表、修改表
HBase列表	添加列、修改列、移除列
HBase表Region	移动Region、分配和合并Region

当某台 RegionServer 出现故障时，总有一部分新写入的数据还没有持久化地存储到磁盘中，因此当迁移该 RegionServer 的服务时，需要从修改记录中恢复这部分还在内存中的数据，

HMaster 需要遍历该 RegionServer 的修改记录，并按 Region 拆分成小块移动到新的地址下。

当 HMaster 节点出现故障时，由于客户端是直接与 RegionServer 交互的，且 Meta 表也是存在于 Zookeeper 中，整个集群的工作会继续稳定运行。

HMaster 还会处理一些重要的工作，如 Region 的切片、RegionServer 的故障转移等，如果 HMaster 出现故障而且没有得到及时处理，这些功能都会受到影响。因此，还是要使 HMaster 尽快恢复正常。Zookeeper 组件提供了多 HMaster 的机制，提高了 HBase 的可用性和稳健性。

在 HDFS 中，DataNode 负责存储实际数据，RegionServer 主要负责响应用户的请求，向 HDFS 读写数据。一般在分布式集群中，RegionServer 运行在 DataNode 服务器上，实现数据的本地性。

每个 RegionServer 包含多个 Region，其负责的功能如下。

（1）处理分批给它的 Region。

（2）处理客户端读写请求。

（3）刷新缓存。

（4）处理 Region 分片。

（5）执行压缩。

RegionServer 是 HBase 中最核心的模块，其内部管理了一系列 Region 对象，每个 Region 由多个 HStore 组成，每个 HStore 对应表中一个列族的存储。

HBase 是按列进行存储的，将列族作为一个集中的存储单元，且 HBase 将具备相同 I/O 特性的列存储到一个列族中，这样可以保证读写的高效性。HBase 处理数据的流程如图 3-26 所示。

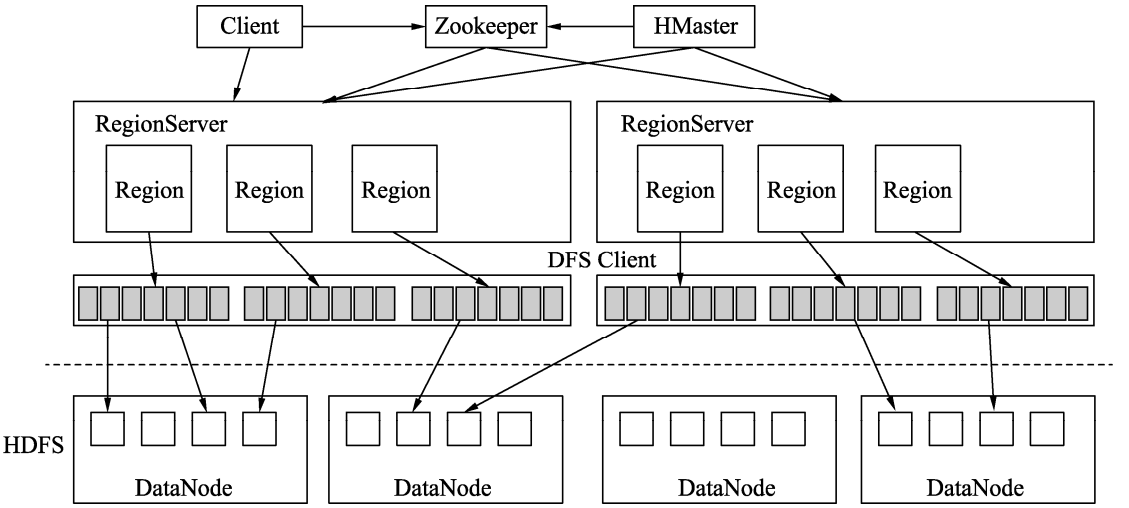


图 3-26　HBase 处理数据的流程

在图 3-26 中，RegionServer 最终将 Region 数据存储在 HDFS 中，采用 HDFS 作为底层存储。HBase 自身并不具备数据复制和维护数据副本的功能，所以依赖于 HDFS 提供可靠和稳定的存储。

HBase 也可以不采用 HDFS，使用如本地文件系统或云计算环境中的 Amazon S3。

本章 HBase 的内容都是以 HDFS 为底层存储来描述的。