实验2 实战HDFS

本实验的知识地图如图2-1所示（表示重点表示难点）。

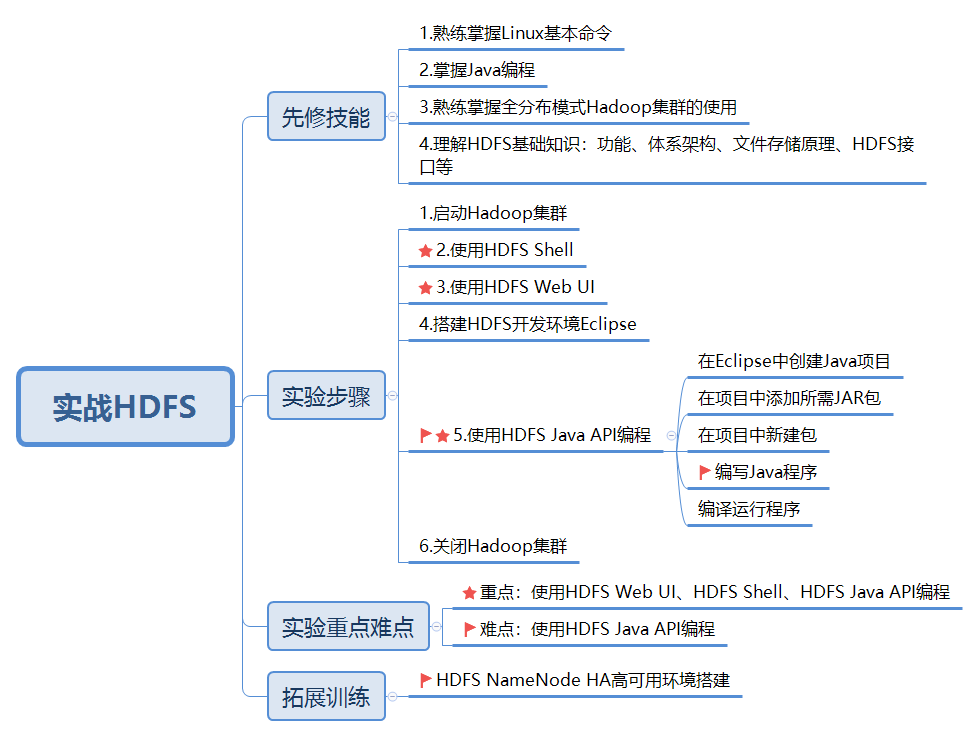


图2-1 实验2实战HDFS知识地图

一、实验目的

1. 理解HDFS体系架构。

2. 理解HDFS文件存储原理和数据读写过程。

3. 熟练掌握HDFS Web UI界面的使用。

4. 熟练掌握HDFS Shell常用命令的使用。

5. 熟练掌握HDFS项目开发环境的搭建。

6. 掌握使用HDFS Java API编写HDFS文件操作程序。

二、实验环境

本实验所需的软件环境包括全分布模式Hadoop集群、Eclipse。

三、实验内容

1. 启动全分布模式Hadoop集群，守护进程包括NameNode、DataNode、SecondaryNameNode、ResourceManager、NodeManager和JobHistoryServer。

2. 查看HDFS Web界面。

3. 练习HDFS Shell文件系统命令和系统管理命令。

4. 在Hadoop集群主节点上搭建HDFS开发环境Eclipse。

5. 使用HDFS Java API编写HDFS文件操作程序，实现上传本地文件到HDFS的功能，采用本地执行和集群执行的两种执行方式测试，观察结果。

6. 使用HDFS Java API编写HDFS文件操作程序，实现查看上传文件在HDFS集群中位置的功能，采用本地执行和集群执行的两种执行方式测试，观察结果。

7. 关闭全分布模式Hadoop集群。

四、实验原理

（一）初识HDFS

HDFS（Hadoop Distributed File System）是Hadoop分布式文件系统，是Hadoop三大核心之一，是针对谷歌文件系统GFS（Google File System）的开源实现（The Google File System, 2003）。HDFS是一个具有高容错性的文件系统，适合部署在廉价的机器上，HDFS能提供高吞吐量的数据访问，非常适合大规模数据集上的应用。大数据处理框架如MapReduce、Spark等要处理的数据源大部分都存储在HDFS上，Hive、HBase等框架的数据通常也存储在HDFS上。简而言之，HDFS为大数据的存储提供了保障。HDFS在Hadoop 2.0生态系统中地位如图2-2所示。

Hive

MapReduce

Spark

Impala

ZooKeeper

Kafka

Flume

YARN

HDFS

Common

Pig

Mahout

Sqoop

HBase

Ambari

Spark SQL

图2-2 HDFS在Hadoop 2.0生态系统中地位

（二）HDFS体系架构

HDFS采用Master/Slave架构模型，一个HDFS集群包括一个NameNode和多个DataNode。名称节点NameNode为主节点，数据节点DataNode为从节点，文件被划分为一系列的数据块（Block）存储在从节点DataNode上，NameNode是中心服务器，不存储数据，负责管理文件系统的名字空间（namespace）以及客户端对文件的访问。HDFS体系架构如图2-3所示。



图2-3 HDFS体系架构

HDFS体系架构中主要包括NameNode和DataNode。下面将详细讲述各个组件和功能。

1. NameNode

NameNode运行在日常硬件上，通常只有一个，是整个文件系统的管理节点。它维护着整个文件系统的文件目录树，包括文件/目录的元数据和每个文件对应的数据块列表，它负责接收用户的操作请求。作为HDFS主服务节点的核心，它主要完成下面任务：

（1）管理命名空间。

（2）控制客户端对文件的读写。

（3）执行常见文件系统操作，比如文件的重命名、复制、移动、打开、关闭以及目录操作。

HDFS中除了主NameNode外还有一个辅助NameNode，称为SecondaryNameNode。从名称上看，SecondaryNameNode似乎是作为NameNode的备份而存在的，事实上并非如此。SecondaryNameNode具有独立的角色和功能，通常认为它和NameNode是协同工作的。SecondaryNameNode主要有如下特征和功能：它是HDFS高可用性的一个解决方案，但不支持热备，使用前配置即可；定期对NameNode中的内存元数据进行更新和备份；默认安装在NameNode相同节点，但是建议安装在不同节点以提高可靠性。

2. DataNode

DataNode也运行在日常硬件上，通常有多个，它为HDFS提供真实文件数据的存储服务。HDFS数据存储在DataNode上，数据块的创建、复制和删除都在DataNode上执行。DataNode将HDFS数据以文件的形式存储在本地的文件系统中，但并不知道有关HDFS文件的信息。DataNode把每个HDFS数据块存储在本地文件系统的一个单独的文件中，并不在同一个目录创建所有的文件，实际上，它用试探的方法来确定每个目录的最佳文件数目，并且在适当的时候创建子目录。在同一个目录中创建所有的本地文件并不是最优的选择，这是因为本地文件系统可能无法高效地在单个目录中支持大量的文件。当一个DataNode启动时，它会扫描本地文件系统，产生一个这些本地文件对应的所有HDFS数据块的列表，然后作为报告发送到NameNode，这个报告就是块状态报告。

另外，客户端是用户操作HDFS最常用的方式，HDFS在部署时都提供了客户端。严格地说，客户端并不算是HDFS的一部分。客户端可以支持打开、读取、写入等常见操作，并且提供了类似Shell的命令行方式来访问HDFS中的数据，也提供了API作为应用程序访问文件系统的客户端编程接口。

（三）HDFS文件存储原理

1. Block

在传统的文件系统中，为了提高磁盘读写效率，一般以数据块为单位，而不是以字节为单位。HDFS也同样采用了块的概念。

HDFS中的数据以文件块Block的形式存储，Block是最基本的存储单位，每次读写的最小单元是一个Block。对于文件内容而言，一个文件的长度大小是N，那么从文件的0偏移开始，按照固定的大小，顺序对文件进行划分并编号，划分好的每一个块称一个Block。Hadoop 2.0中默认Block大小是128MB，以一个N=256MB的文件为例，被切分成256/128=2个Block。不同于普通文件系统，HDFS中如果一个文件小于一个数据块的大小，并不占用整个数据块存储空间。Block的大小可以根据实际需求进行配置，可以通过HDFS配置文件hdfs-site.xml中的参数dfs.blocksize来定义块大小，但要注意，数字必须是2K，文件的大小可以不是Block大小的整数倍，这时最后一个块可能存在剩余。例如，一个文件大小是260MB，在Hadoop 2.0中占用三个块，第三个块只使用了4MB。

为什么HDFS数据块设置的这么大呢？原因是和普通的本地磁盘文件系统不同，HDFS存储的是大数据文件，通常会有TB甚至PB的数据文件需要管理，所以数据的基本单元必须足够大才能提高管理效率。而如果还使用像Linux本地文件系统EXT3的4KB单元来管理数据，则会非常低效，同时会浪费大量的元数据空间。

2. Block副本管理策略

HDFS采用多副本方式对数据进行冗余存储，通常一个数据块的多个副本会被分布到不同的DataNode上。

HDFS提供可靠的算法实现在分布式环境中存储大量数据。简单来说，每个数据块Block都存在副本以提高容错性。默认情况下每个块存在3个副本，例如，存储一个100MB的文件默认情况下需要占用300MB的磁盘空间。数据块的信息会定期由DataNode报送给NameNode，任何时候，当NameNode发现一个块的副本个数少于3个或者多于3个时都会进行补充或者删除。副本放置的基本原则是保证并非所有的副本都在同一个机架（Rack）上。例如，对于默认的3个块副本，在同一个机架上存放两个副本，在另一个机架上存放另一个副本，如图2-4所示。这样放置的好处在于提供高容错性的同时降低延时，注意一个Rack可能包含多个DataNode，而数据分布在不同DataNode可以提高数据读写并发。对于多于3个副本的情况，其它副本将会随机分布在不同DataNode，同时保证同一个机架中最多存在两个副本。可以通过配置文件hdfs-site.xml中的参数dfs.replication来定义Block副本数。

Block：

1

2

3

机架1

1

2

1

机架2

2

3

2

机架3

3

3

1

图2-4 Block副本在机架中的放置策略

图2-5显示了Hadoop集群中机架之间的逻辑连接结构，可以看到，通过交换机，同一个机架和不同机架的计算机物理连接在一起，同一个机架内的计算机可以直接通过单层交换机连接，速度很快，而不同机架之间的通信需要经过多层交换机，速度稍慢。数据块存放时以机架为独立单元，这样既有高容错性，也可以保证并发性能。

Hadoop集群

核心交换机1

核心交换机k

机架1

机架2

单层交换机

单层交换机

计算机1

计算机2

计算机1

计算机2

**…**

**…**

**…**

**…**

机架N

单层交换机

计算机1

计算机2

**…**

图2-5 Hadoop集群中机架逻辑结构

3. 数据读取

HDFS的真实数据分散存储在DataNode上，但是读取数据时需要先经过NameNode。HDFS数据读取的基本过程为：首先客户端连接到NameNode询问某个文件的元数据信息，NameNode返回给客户一个包含该文件各个块位置信息（存储在哪个DataNode）的列表；然后，客户端直接连接对应DataNode来并行读取块数据；最后，当客户得到所有块后，再按照顺序进行组装，得到完整文件。为了提高物理传输速度，NameNode在返回块的位置时，优先选择距离客户更近的DataNode。

客户端读取HDFS上的文件时，需要调用HDFS Java API一些类的方法，从编程角度来看，主要经过以下几个步骤，如图2-6所示。

（1）客户端生成一个FileSystem实例（DistributedFileSystem对象），并使用此实例的open()方法打开HDFS上的一个文件。

（2）DistributedFileSystem通过RPC调用向NameNode发出请求，得到文件的位置信息，即数据块编号和所在DataNode地址，对于每一个数据块，名称节点返回保存数据块的数据节点的地址，通常按照DataNode地址与客户端的距离从近到远排序。

（3）FileSystem实例获得地址信息后，生成一个FSDataInputStream对象实例返回给客户端，此实例封装了一个DFSInputStream对象，负责存储数据块信息和DataNode地址信息，并负责后续的文件内容读取工作。

（4）客户端向FSDataInputStream发出读取数据的read()调用。

（5）FSDataInputStream收到read()调用请求后，FSDataInputStream封装的DFSInputStream选择与第一个数据块最近的DataNode，并读取相应的数据信息返回给客户端，在数据块读取完成后，DFSInputStream负责关闭到相应DataNode的链接。

（6）DFSInputStream依次选择后续数据块的最近DataNode节点，并读取数据返回给客户端，直到最后一个数据块读取完毕。DFSInputStream从DataNode读取数据时，可能会碰上某个DataNode失效的情况，则会自动选择下一个包含此数据块的最近的DataNode去读取。

（7）客户端读取完所有数据块，然后调用FSDataInputStream的close()方法关闭文件。

客户端JVM

1

3

5

2

数据节点

DataNode

2

3

5

5

3

1

2

客户端

DistributedFileSytem

FSDataInputStream

DFSInputStream

名称节点

NameNode

2：获取数据块信息

4：读取请求

7：关闭文件

3：生成对象实例

5：读取数据

6：读取数据

元数据

(名称，块个数，地址，……)

1：打开文件

6：读取数据

**…**

数据节点

DataNode

数据节点

DataNode

图2-6 HDFS数据读取过程

从图2-6可以看出，HDFS数据读取分散在了不同的DataNode节点上，基本上不存在单点问题，水平扩展性强。对于NameNode节点，只需要传输元数据（块地址）信息，数据IO压力较小。

4. 数据写入

HDFS的设计遵循“一次写入，多次读取”的原则，所有数据只能添加不能更新。数据会被划分为等尺寸的块写入不同的DataNode中。每个块通常保存指定数量的副本（默认3个）。HDFS数据写入基本过程如图2-7所示，基本过程为：客户端向NameNode发送文件写请求，NameNode给客户分配写权限，并随机分配块的写入地址——DataNode的IP，兼顾副本数量和块Rack自适应算法，例如副本因子是3，则每个块会分配到三个不同的DataNode，为了提高传输效率，客户端只会向其中一个DataNode复制一个副本，另外两个副本则由DataNode传输到相邻DataNode。

机架N

机架2

机架1

客户端

Client

名称节点

NameNode

核心交换机

交换机

交换机

数据节点DataNode1

**…**

**…**

**…**

交换机

**…**

1：写请求

2：返回DataNode1、3、4的IP

3：通过核心交换机与机架N的交换机通信

4：写入数据

4

5：流水线复制

数据节点DataNode2

数据节点DataNode3

数据节点DataNode4

数据节点DataNode5

数据节点DataNode6

图2-7 HDFS数据写入基本过程

从编程角度来说，将数据写入HDFS主要经过以下几个步骤，如图2-8所示。

（1）创建和初始化FileSystem，客户端调用create()来创建文件

（2）FileSystem用RPC调用NameNode，在文件系统的命名空间中创建一个新的文件，名称节点首先确定文件原来不存在，并且客户端有创建文件的权限，然后创建新文件。

（3）FileSystem返回DFSOutputStream，客户端开始写入数据。

（4）DFSOutputStream将数据分成块，写入data queue。data queue由Data Streamer读取，并通知名称节点分配数据节点，用来存储数据块（每块默认复制3块）。分配的数据节点放在一个数据流管道（pipeline）里。Data Streamer将数据块写入pipeline中的第一个数据节点，一个数据节点将数据块发送给第二个数据节点，第二个数据节点将数据发送给第三个数据节点。

（5）DFSOutputStream为发出去的数据块保存了ack queue，等待pipeline中的数据节点告知数据已经写入成功。

（6）当客户端结束写入数据，则调用stream的close函数。此操作将所有的数据块写入pipeline中的数据节点，并等待ack queue返回成功。

（7）通知名称节点写入完毕。

客户端JVM

1

3

5

2

数据节点DataNode

2

3

5

5

3

1

2

客户端

DistributedFileSytem

FSDataOutputStream

DFSOutputStream

2：创建文件元数据

3：写入数据

6：关闭文件

4：写入数据包

1：创建文件请求

5：接收确认包

4

4

5

5

7：写操作完成

元数据

(名称，块个数，地址，……)

名称节点

NameNode

数据节点DataNode

数据节点DataNode

图2-8 HDFS数据写入过程

如果数据节点在写入的过程中失败，DFSOutputStream关闭pipeline，将ack queue中的数据块放入data queue的开始，当前的数据块在已经写入的数据节点中被名称节点赋予新的标示，则错误节点重启后能够察觉其数据块是过时的，会被删除。失败的数据节点从pipeline中移除，另外的数据块则写入pipeline中的另外两个数据节点。名称节点（NameNode）则被通知此数据块是复制块数不足，会再创建第三份备份。

数据写入过程中，如果某个DataNode出现故障，DataStreamer将关闭到此节点的链接，故障节点将从DataNode链中删除，其他DataNode将继续完成写入操作，NameNode通过返回值发现某个DataNode未完成写入任务，会分配另一个DataNode完成此写入操作。对于一个数据块Block，只要有一个副本写入成功，就视为写入完成，后续将启动自动恢复机制，恢复指定副本数量。

数据写入可以看作是一个流水线pipeline过程，具体来说，客户端收到NameNode发送的块存储位置DataNode列表后，将做如下工作。

（1）选择DataNode列表中的第一个DataNode1，通过IP地址建立TCP连接。

（2）客户端通知DataNode1准备接收块数据，同时发送后续DataNode的IP地址给DataNode1，副本随后会拷贝到这些DataNode。

（3）DataNode1连接DataNode2，并通知DataNode2连接DataNode3，前一个DataNode发送副本数据给后一个DataNode，依次类推。

（4）ack确认消息遵从相反的顺序，即DataNode3收到完整块副本后返回确认给DataNode2，DataNode2收到完整块副本后返回确认给DataNode1。而DataNode1最后通知客户端所有数据块已经成功复制。对于3个副本，DataNode1会发送3个ack给客户端 表示3个DataNode都成功接收。随后，客户端通知NameNode，完整文件写入成功，NameNode更新元数据。

（5）当客户端接到通知流水线已经建立完成后，将会准备发送数据块到流水线中，然后逐个数据块按序在流水线中传输。这样以来，客户端只需要发送一次，所有备份将在不同DataNode之间自动完成，提高了传输效率。

（四）HDFS接口

1. HDFS Web UI

HDFS Web UI主要面向管理员，提供服务器基础统计信息和文件系统运行状态的查看功能，不支持配置更改操作。从该页面上，管理员可以查看当前文件系统中各个节点的分布信息，浏览名称节点上的存储、登录等日志，以及下载某个数据节点上某个文件的内容。HDFS Web UI地址为http://NameNodeIP:50070，进入后可以看到当前HDFS文件系统的Overview、Summary、NameNode Journal Status、NameNode Storage等信息。其概览效果如图2-9所示。

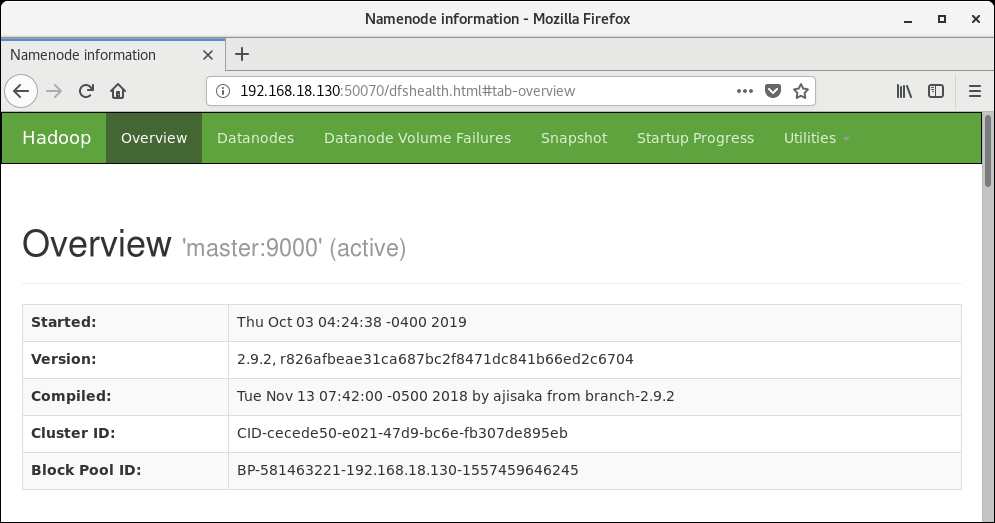


图2-9 HDFS Web UI之概览

HDFS Web UI的概要效果如图2-10所示，从图2-10中可以看到容量、活动节点等信息。

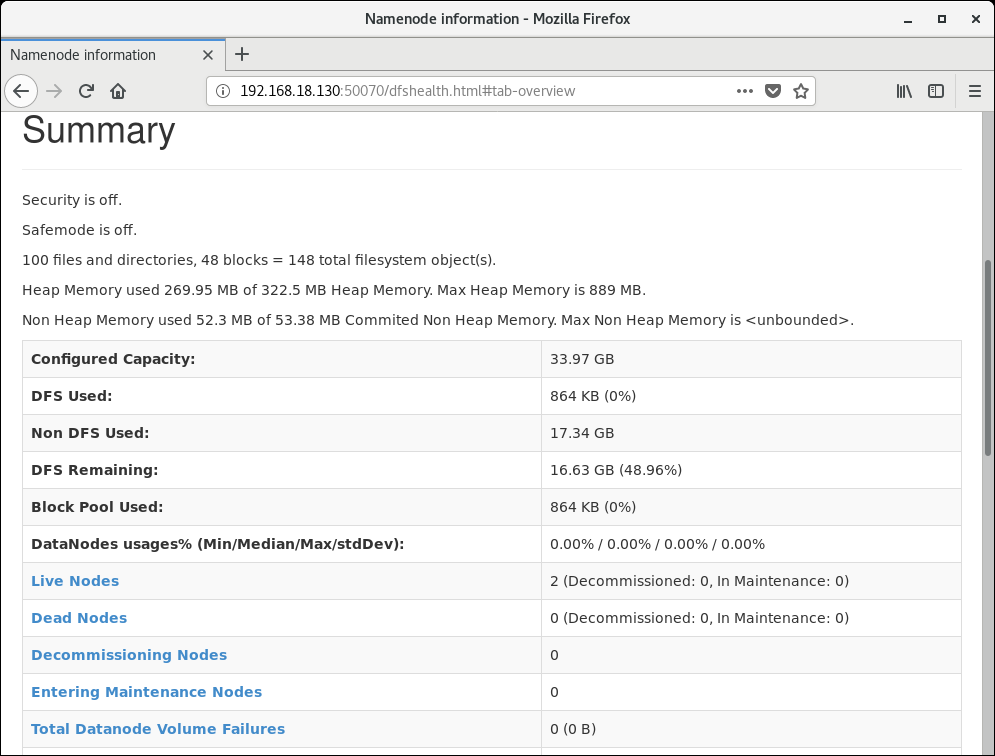


图2-10 HDFS Web UI之概要

我们可以通过首页顶端菜单项『Utilities』→『Browse the file system』查看目录，如图2-11和图2-12所示。

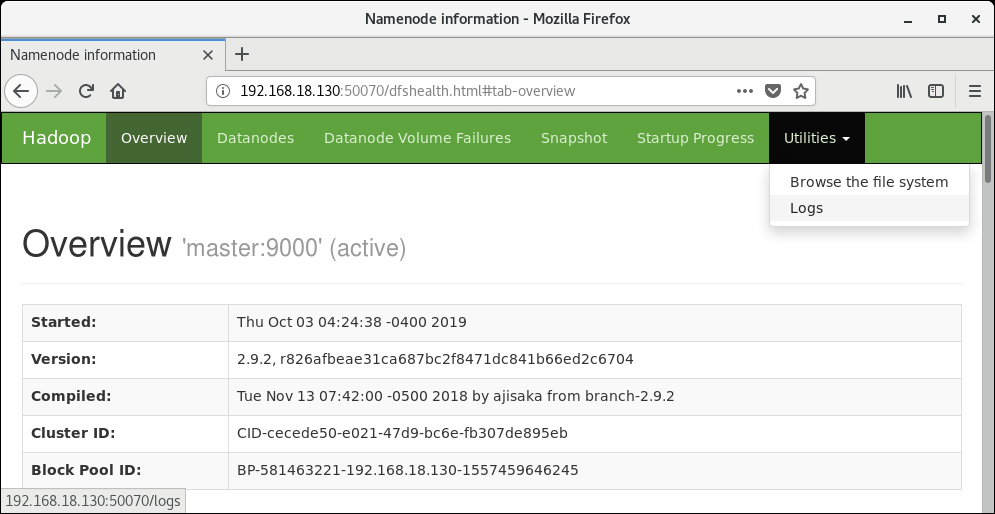


图2-11 使用HDFS Web UI进入查看HDFS目录及文件

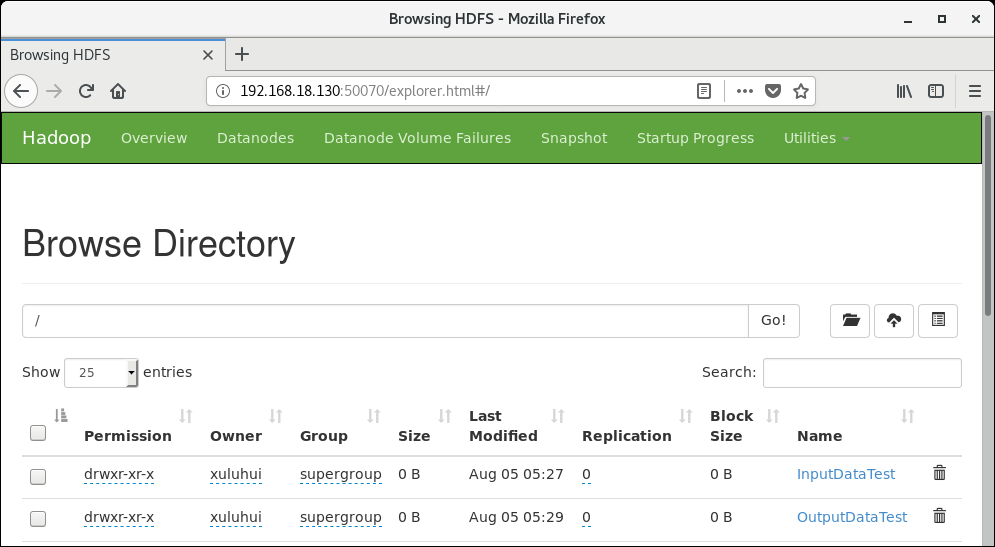


图2-12 使用HDFS Web UI查看HDFS目录及文件

2. HDFS Shell

1）HDFS文件系统命令

HDFS文件系统命令的入口是“hadoop fs”，其语法是“hadoop fs [generic options]”，而命令“hdfs dfs [generic options]”也可以使用，它们两者的区别在于：“hadoop fs”使用面最广，可以操作任何文件系统，比如本地文件、HDFS文件、HFTP文件、S3文件系统等；而“hdfs dfs”则是专门针对HDFS文件系统的操作。“hadoop fs”命令的完整帮助如下所示。

[xuluhui@master ~]$ hadoop fs

Usage: hadoop fs [generic options]

[-appendToFile <localsrc> ... <dst>]

[-cat [-ignoreCrc] <src> ...]

[-checksum <src> ...]

[-chgrp [-R] GROUP PATH...]

[-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]

[-chown [-R] [OWNER][:[GROUP]] PATH...]

[-copyFromLocal [-f] [-p] [-l] [-d] <localsrc> ... <dst>]

[-copyToLocal [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]

[-count [-q] [-h] [-v] [-t [<storage type>]] [-u] [-x] <path> ...]

[-cp [-f] [-p | -p[topax]] [-d] <src> ... <dst>]

[-createSnapshot <snapshotDir> [<snapshotName>]]

[-deleteSnapshot <snapshotDir> <snapshotName>]

[-df [-h] [<path> ...]]

[-du [-s] [-h] [-x] <path> ...]

[-expunge]

[-find <path> ... <expression> ...]

[-get [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]

[-getfacl [-R] <path>]

[-getfattr [-R] {-n name | -d} [-e en] <path>]

[-getmerge [-nl] [-skip-empty-file] <src> <localdst>]

[-help [cmd ...]]

[-ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [<path> ...]]

[-mkdir [-p] <path> ...]

[-moveFromLocal <localsrc> ... <dst>]

[-moveToLocal <src> <localdst>]

[-mv <src> ... <dst>]

[-put [-f] [-p] [-l] [-d] <localsrc> ... <dst>]

[-renameSnapshot <snapshotDir> <oldName> <newName>]

[-rm [-f] [-r|-R] [-skipTrash] [-safely] <src> ...]

[-rmdir [--ignore-fail-on-non-empty] <dir> ...]

[-setfacl [-R] [{-b|-k} {-m|-x <acl\_spec>} <path>]|[--set <acl\_spec> <path>]]

[-setfattr {-n name [-v value] | -x name} <path>]

[-setrep [-R] [-w] <rep> <path> ...]

[-stat [format] <path> ...]

[-tail [-f] <file>]

[-test -[defsz] <path>]

[-text [-ignoreCrc] <src> ...]

[-touchz <path> ...]

[-truncate [-w] <length> <path> ...]

[-usage [cmd ...]]

Generic options supported are:

-conf <configuration file> specify an application configuration file

-D <property=value> define a value for a given property

-fs <file:///|hdfs://namenode:port> specify default filesystem URL to use, overrides 'fs.defaultFS' property from configurations.

-jt <local|resourcemanager:port> specify a ResourceManager

-files <file1,...> specify a comma-separated list of files to be copied to the map reduce cluster

-libjars <jar1,...> specify a comma-separated list of jar files to be included in the classpath

-archives <archive1,...> specify a comma-separated list of archives to be unarchived on the compute machines

The general command line syntax is:

command [genericOptions] [commandOptions]

部分HDFS文件系统命令的说明如表2-1所示。

表2-1 HDFS文件系统命令说明（部分）

|  |  |
| --- | --- |
| 命令选项 | 功能 |
| -ls | 显示文件的元数据信息或者目录包含的文件列表信息 |
| -mv | 移动HDFS文件到指定位置 |
| -cp | 将文件从源路径复制到目标路径 |
| -rm | 删除文件，“-rm -r”或者“-rm -R”可以递归删除文件夹，文件夹可以包含子文件夹和子文件 |
| -rmdir | 删除空文件夹，**注意**：如果文件夹非空，则删除失败 |
| -put | 从本地文件系统复制单个或多个源路径上传到HDFS，同时支持从标准输入读取源文件内容后写入目标位置 |
| -get | 复制源路径指定的文件到本地文件系统目标路径指定的文件或文件夹 |
| -cat | 将指定文件内容输出到标准输出stdout |
| -mkdir | 创建指定目录 |
| -setrep | 改变文件的副本系数，选项-R用于递归改变目录下所有文件的副本系数选项-w表示等待副本操作结束才退出命令 |

2）HDFS系统管理命令

HDFS系统管理命令的入口是“hdfs dfsadmin”，其完整帮助如下所示。

[xuluhui@master ~]$ hdfs dfsadmin

Usage: hdfs dfsadmin

Note: Administrative commands can only be run as the HDFS superuser.

[-report [-live] [-dead] [-decommissioning] [-enteringmaintenance] [-inmaintenance]]

[-safemode <enter | leave | get | wait>]

[-saveNamespace]

[-rollEdits]

[-restoreFailedStorage true|false|check]

[-refreshNodes]

[-setQuota <quota> <dirname>...<dirname>]

[-clrQuota <dirname>...<dirname>]

[-setSpaceQuota <quota> [-storageType <storagetype>] <dirname>...<dirname>]

[-clrSpaceQuota [-storageType <storagetype>] <dirname>...<dirname>]

[-finalizeUpgrade]

[-rollingUpgrade [<query|prepare|finalize>]]

[-refreshServiceAcl]

[-refreshUserToGroupsMappings]

[-refreshSuperUserGroupsConfiguration]

[-refreshCallQueue]

[-refresh <host:ipc\_port> <key> [arg1..argn]

[-reconfig <namenode|datanode> <host:ipc\_port> <start|status|properties>]

[-printTopology]

[-refreshNamenodes datanode\_host:ipc\_port]

[-getVolumeReport datanode\_host:ipc\_port]

[-deleteBlockPool datanode\_host:ipc\_port blockpoolId [force]]

[-setBalancerBandwidth <bandwidth in bytes per second>]

[-getBalancerBandwidth <datanode\_host:ipc\_port>]

[-fetchImage <local directory>]

[-allowSnapshot <snapshotDir>]

[-disallowSnapshot <snapshotDir>]

[-shutdownDatanode <datanode\_host:ipc\_port> [upgrade]]

[-evictWriters <datanode\_host:ipc\_port>]

[-getDatanodeInfo <datanode\_host:ipc\_port>]

[-metasave filename]

[-triggerBlockReport [-incremental] <datanode\_host:ipc\_port>]

[-listOpenFiles]

[-help [cmd]]

Generic options supported are:

-conf <configuration file> specify an application configuration file

-D <property=value> define a value for a given property

-fs <file:///|hdfs://namenode:port> specify default filesystem URL to use, overrides 'fs.defaultFS' property from configurations.

-jt <local|resourcemanager:port> specify a ResourceManager

-files <file1,...> specify a comma-separated list of files to be copied to the map reduce cluster

-libjars <jar1,...> specify a comma-separated list of jar files to be included in the classpath

-archives <archive1,...> specify a comma-separated list of archives to be unarchived on the compute machines

The general command line syntax is:

command [genericOptions] [commandOptions]

3. HDFS API

HDFS使用Java语言编写，所以提供了丰富了Java编程接口供开发人员调用，当然HDFS同时支持其它语言如C++、Python等编程接口，但它们都没有Java接口方便。凡是使用Shell命令可以完成的功能，都可以使用相应Java API来实现，甚至使用API可以完成Shell命令不支持的功能。

实际开发中，HDFS Java API最常用的类是org.apache.hadoop.fs.FileSystem。常用的HDFS Java类如表2-2所示。

表2-2 HDFS Java API常用类

|  |  |
| --- | --- |
| 类名 | 说明 |
| org.apache.hadoop.fs.FileSystem | 通用文件系统基类，用于与HDFS文件系统交互，编写的HDFS程序都需要重写FileSystem类，通过该类，可以方便地像操作本地文件系统一样操作HDFS集群文件 |
| org.apache.hadoop.fs.FSDataInputStream | 文件输入流，用于读取HDFS文件 |
| org.apache.hadoop.fs.FSDataOutputStream | 文件输出流，向HDFS顺序写入数据流 |
| org.apache.hadoop.fs.Path | 文件与目录定位类，用于定义HDFS集群中指定的目录与文件绝对或相对路径 |
| org.apache.hadoop.fs.FileStatus | 文件状态显示类，可以获取文件与目录的元数据、长度、块大小、所属用户、编辑时间等信息；同时可以设置文件用户、权限等内容 |

关于HDFS API的更多信息读者请参考官网https://hadoop.apache.org/docs/r2.9.2/api/index.html。

五、实验步骤

（一）启动Hadoop集群

启动Hadoop守护进程，只需在节点上依次执行以下3条命令即可。

start-dfs.sh

start-yarn.sh

mr-jobhistory-daemon.sh start historyserver

start-dfs.sh命令会在节点上启动NameNode、DataNode和SecondaryNameNode服务。start-yarn.sh命令会在节点上启动ResourceManager、NodeManager服务。mr-jobhistory-daemon.sh命令会在节点上启动JobHistoryServer服务。请注意，同全分布模式Hadoop集群，即使对应的守护进程没有启动成功，Hadoop也不会在控制台显示错误消息，读者可以利用jps命令一步一步查询，逐步核实对应的进程是否启动成功。

（二）使用 HDFS Shell

【案例2-1】在/usr/local/hadoop-2.9.2目录下创建目录HelloData，在该目录下新建2个文件file1.txt和file2.txt，在其下任意输入一些英文测试语句。使用HDFS Shell命令完成以下操作：首先创建HDFS目录/InputData，然后将file1.txt和file2.txt上传至HDFS目录/InputData下，最后查看这两个文件内容。

（1）在本地Linux文件系统/usr/local/hadoop-2.9.2目录下创建一个名为HelloData的文件夹，使用的命令如下所示。

mkdir /usr/local/hadoop-2.9.2/HelloData

（2）在HelloData文件夹下创建2个文件file1.txt和file2.txt。创建file1.txt文件使用的命令如下所示。

vim /usr/local/hadoop-2.9.2/HelloData/file1.txt

然后在file1.txt中写入如下测试语句。

Hello Hadoop

Hello HDFS

创建file2.txt文件使用的命令如下所示。

vim /usr/local/hadoop-2.9.2/HelloData/file2.txt

然后在file2.txt中写入如下测试语句。

Hello Xijing

Hello ShengDa

（3）使用“hadoop fs”命令创建HDFS目录/InputData，使用的命令如下所示。

hadoop fs -mkdir /InputData

（4）查看HDFS目录/InputData是否创建成功，使用的命令及效果如图2-13所示。

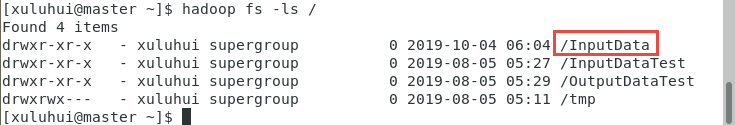


图2-13 查看HDFS目录/InputData是否创建成功

（5）将file1.txt和file2.txt上传至HDFS目录/InputData下，使用的命令如下所示。

hadoop fs -put /usr/local/hadoop-2.9.2/HelloData/\* /InputData

（6）查看HDFS上文件file1.txt和file2.txt内容，使用的命令及效果如图2-14所示。

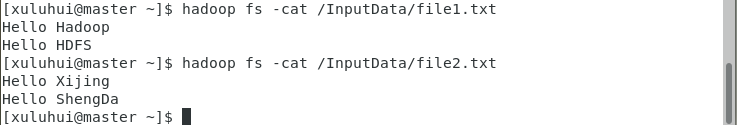


图2-14 查看HDFS文件file1.txt和file2.txt内容

【案例2-2】使用HDFS Shell系统管理命令打印出当前文件系统整体信息和各个节点的分布信息。

使用的命令如下所示，效果如图2-15所示。

hdfs dfsadmin -report

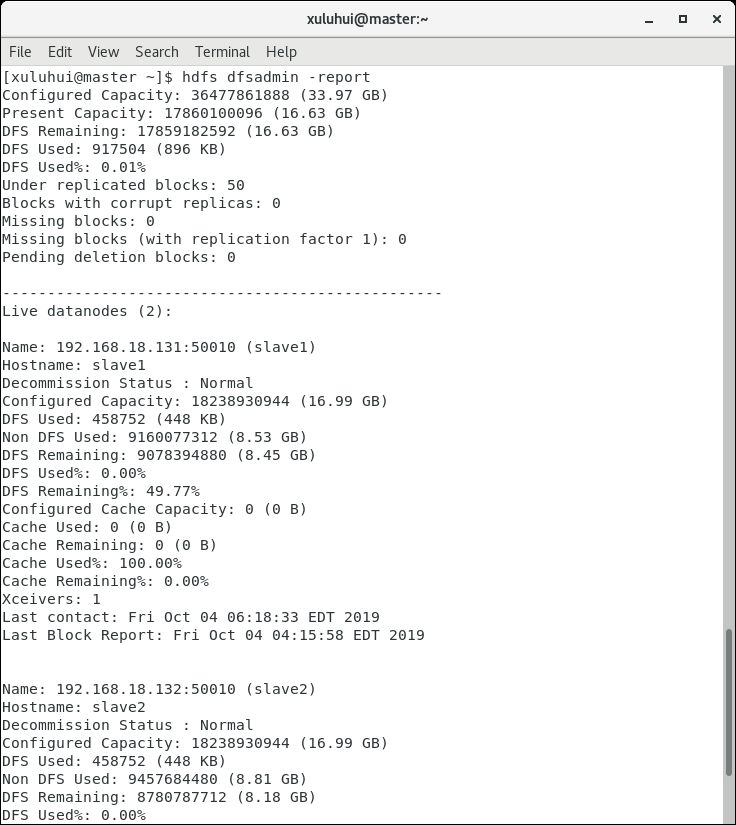


图2-15 命令“hdfs dfsadmin -report”执行效果

（三）使用HDFS Web UI

【案例2-3】通过HDFS Web UI查看【案例2-1】中创建的HDFS目录/InputData及其下文件。

（1）打开浏览器，输入HDFS Web UI的地址如http://192.168.18.130:8080，进入到HDFS Web主界面【Namenode information】，选择首页顶端菜单项『Utilities』→『Browse the file system』进入到界面【Browsing HDFS】查看目录，如图2-16所示。

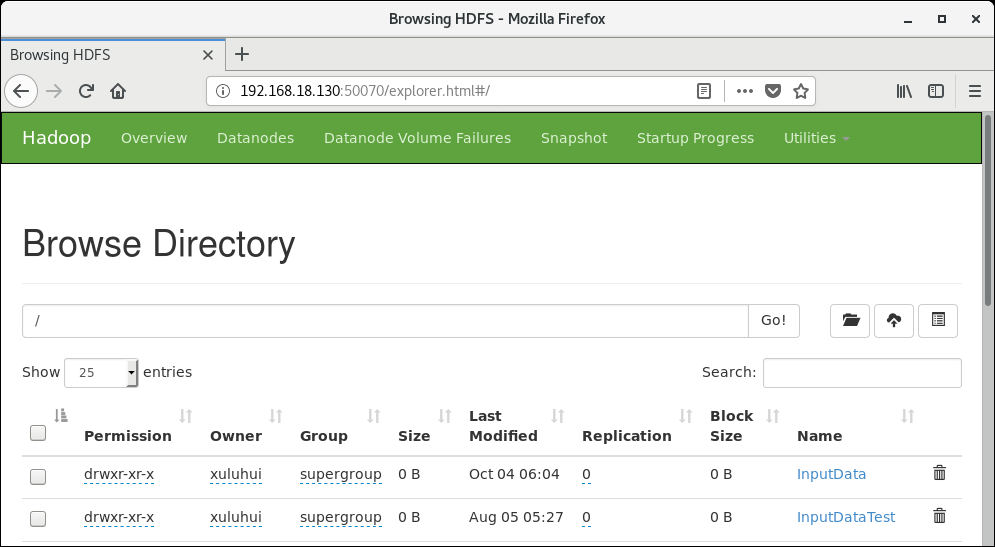


图2-16 通过HDFS Web UI查看创建的HDFS目录/InputData

（2）单击目录“InputData”，进入该目录，如图2-17所示，从图2-17中可以看出，该目录下有两个文件file1.txt和file.txt，它们的副本数“Replication”均为3，块大小“Block Size”均为128M。

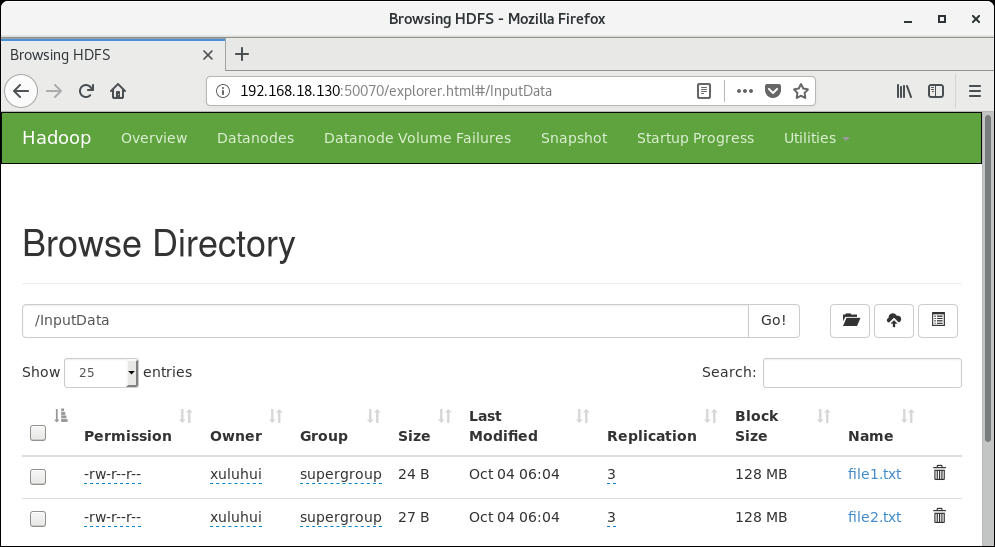


图2-17 通过HDFS Web UI查看/InputData下文件

（3）单击文件“file1.txt”，进入窗口【File information】，如图2-18所示，从图2-18中可以看到，该文件的块号“Block ID”为1073742656，该文件在DataNode节点slave1、slave2上存放。

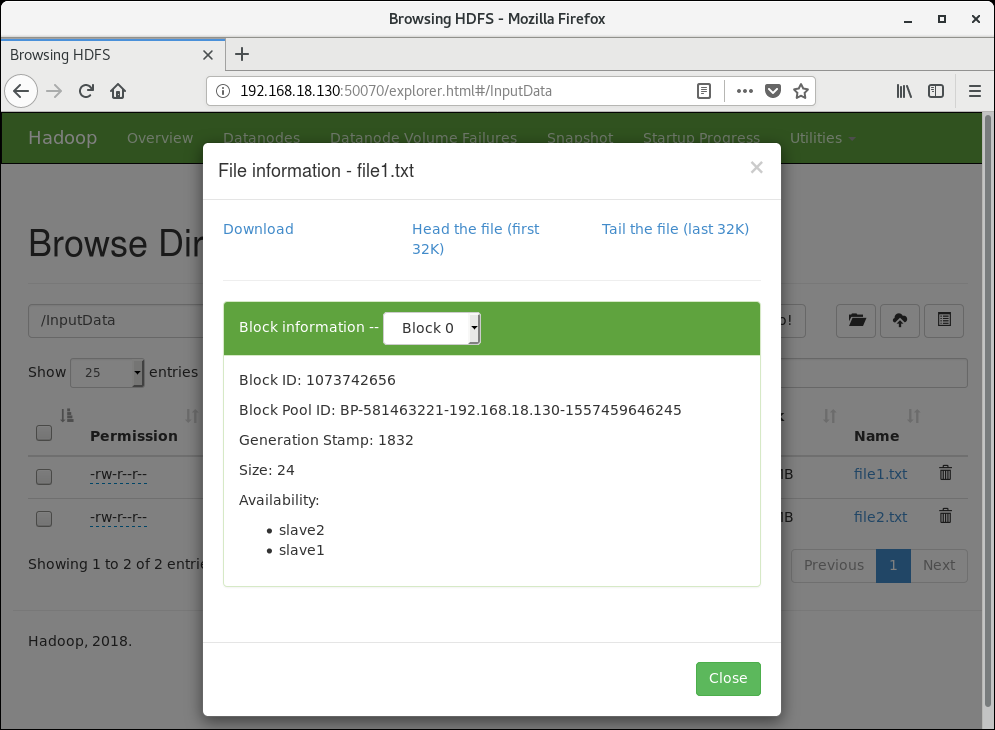


图2-18 通过HDFS Web UI查看文件file1.txt信息

（四）搭建HDFS开发环境Eclipse

在Hadoop集群主节点上搭建HDFS开发环境Eclipse。

1. 获取Eclipse

Eclipse官方下载地址为https://www.eclipse.org/downloads/packages，建议读者下载较新版本，本实验选用的是2018年9月发布的Linux 64位版本Eclipse IDE 2018-09 for Java Developers，其安装包文件eclipse-java-2018-09-linux-gtk-x86\_64.tar.gz例如存放在master机器的/home/xuluhui/Downloads中。

2. 安装Eclipse

在master机器上解压eclipse-java-2018-09-linux-gtk-x86\_64.tar.gz到安装目录如/usr/local下，使用命令如下所示。

su root

cd /usr/local

tar -zxvf /home/xuluhui/Downloads/eclipse-java-2018-09-linux-gtk-x86\_64.tar.gz

3. 打开Eclipse IDE

进入/usr/local/eclipse中通过可视化桌面打开Eclipse IDE，默认的工作空间为“/home/xlh/eclipse-workspace”。Eclipse启动界面如图2-19所示，其主界面如图2-20所示。



图2-19 Eclipse IDE 2018-09启动界面

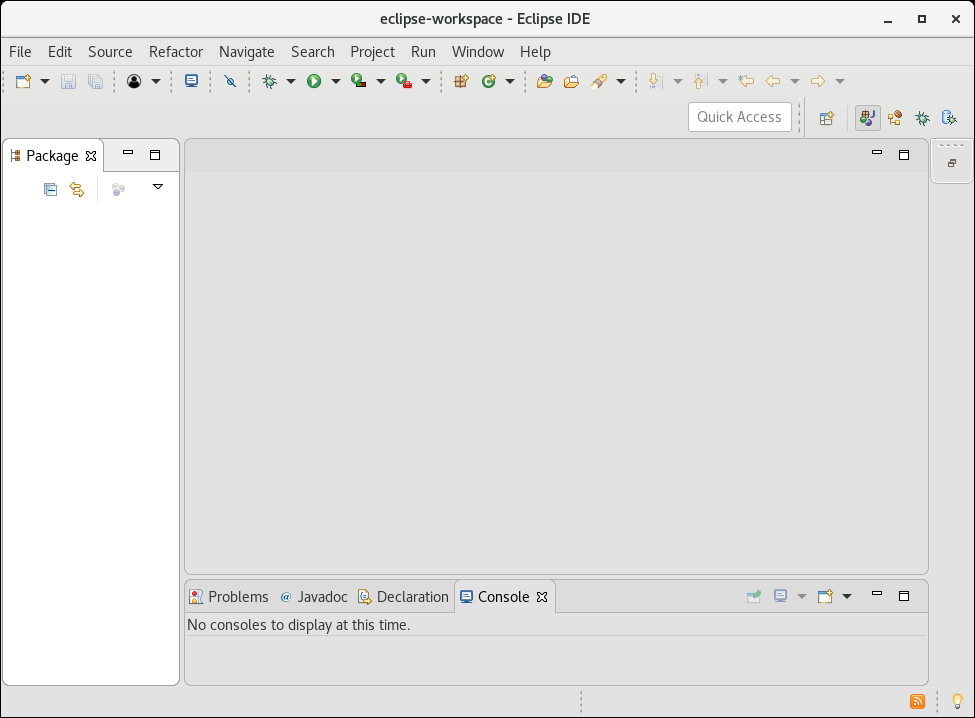


图2-20 Eclipse IDE主界面

（五）使用HDFS Java API编程

【案例2-4】使用HDFS Java API编写HDFS文件操作程序，实现上传本地文件到HDFS的功能，采用本地执行和集群执行的两种执行方式测试，观察结果。

1. 在Eclipse中创建Java项目

打开Eclipse IDE，进入主界面，选择菜单『File』→『New』→『Java Project』，创建Java项目“HDFSExample”，如图2-21所示。本书中关于HDFS编程实例均存放在此项目下。

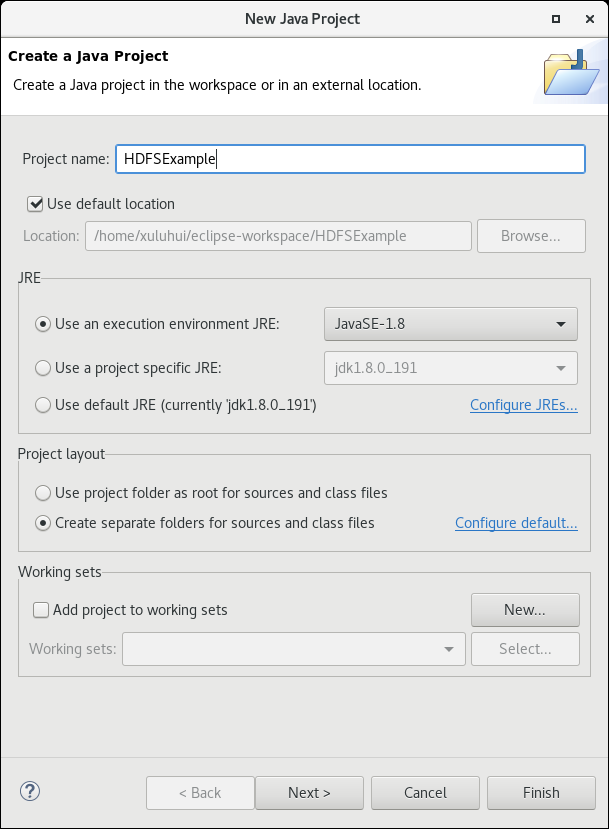


图2-21 创建Java项目“HDFSExample”

2. 在项目中添加所需JAR包

为了编写关于HDFS文件操作的应用程序，需要向Java工程中添加JAR包，这些JAR包中包含了可以访问HDFS的Java API，这些JAR包都位于Linux系统的$HADOOP\_HOME/share/hadoop目录下，对于本书而言，就是在/usr/local/hadoop-2.9.2/share/hadoop目录下。读者可以按以下步骤添加该应用程序编写时所需的JAR包。

（1）右键单击Java项目“HDFSExample”，从弹出的菜单中选择『Build Path』→『Configure Build Path…』，如图2-22所示。

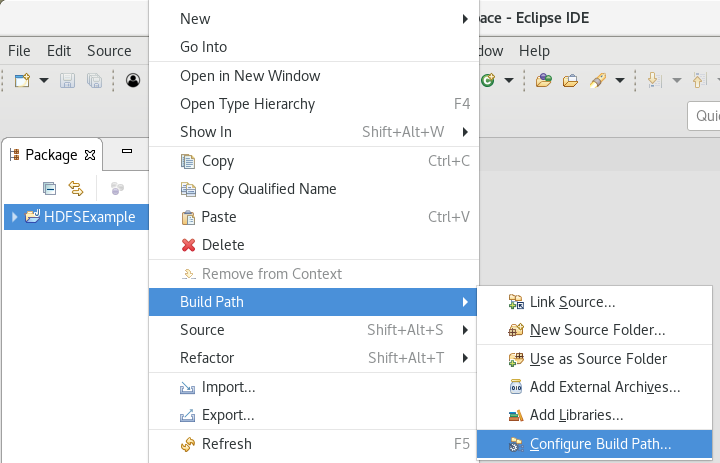


图2-22 进入“HDFSExample”项目“Java Build Path”

（2）进入窗口【Properties for HDFSExample】，可以看到添加JAR包的主界面，如图2-23所示。

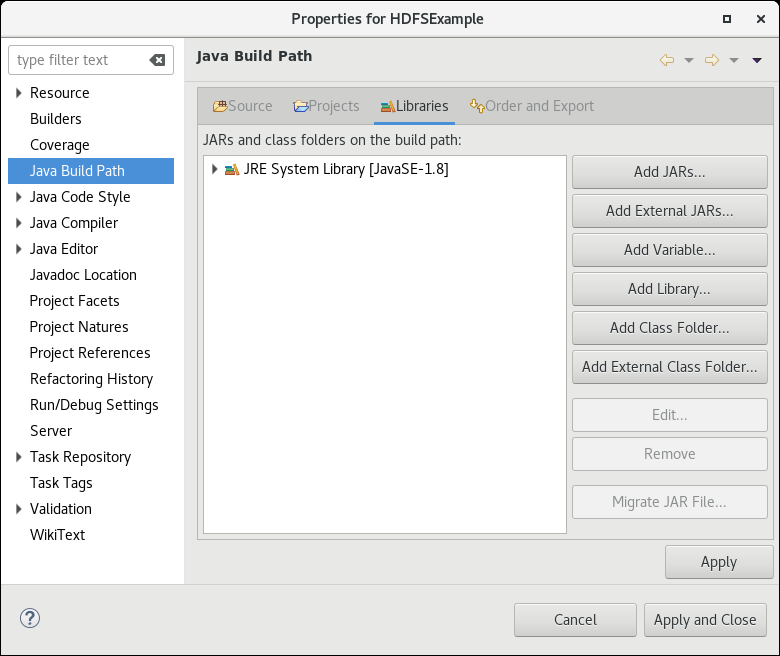


图2-23 添加JAR包主界面

（3）单击图中的按钮Add External JARS，依次添加的jar包包括：

* $HADOOP\_HOME/share/hadoop/hdfs/hadoop-hdfs-2.9.2.jar
* $HADOOP\_HOME /share/hadoop/hdfs/lib/\*，即其下所有jar包
* $HADOOP\_HOME/share/hadoop/common/hadoop-common-2.8.1.jar
* $HADOOP\_HOME/share/hadoop/common/lib/\*即其下所有jar包

这里编者为了方便，导入了$HADOOP\_HOME/share/hadoop/hdfs/lib和$HADOOP\_HOME/share/hadoop/common/lib下的所有jar包，读者也可以根据实际编程需要导入必要的jar包。

其中添加JAR包hadoop-hdfs-2.9.2.jar的过程如图2-24所示，找到此JAR包后选中并单击右上角的OK按钮，这样就成功把hadoop-hdfs-2.9.2.jar增加到了当前Java项目中。添加其它jar包的过程同此，不再赘述。

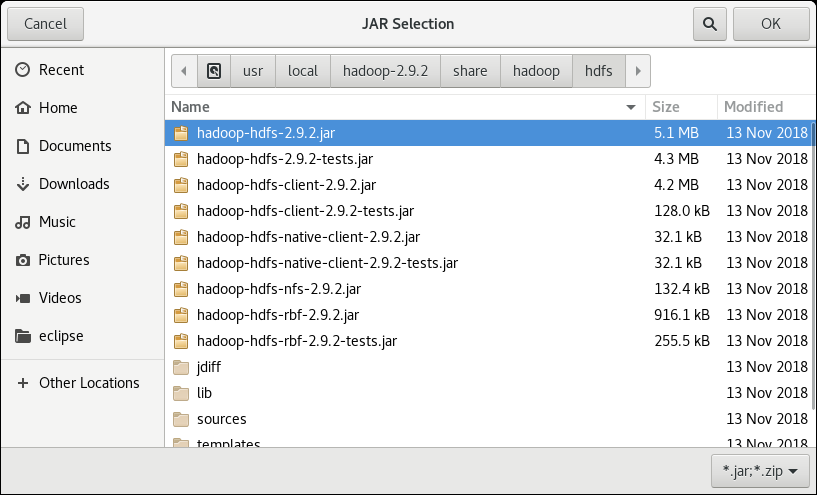


图2-24 添加hadoop-hdfs-2.9.2.jar到Java项目中

（4）完成JAR包添加后的界面如图2-25所示，单击按钮Apply and Close。

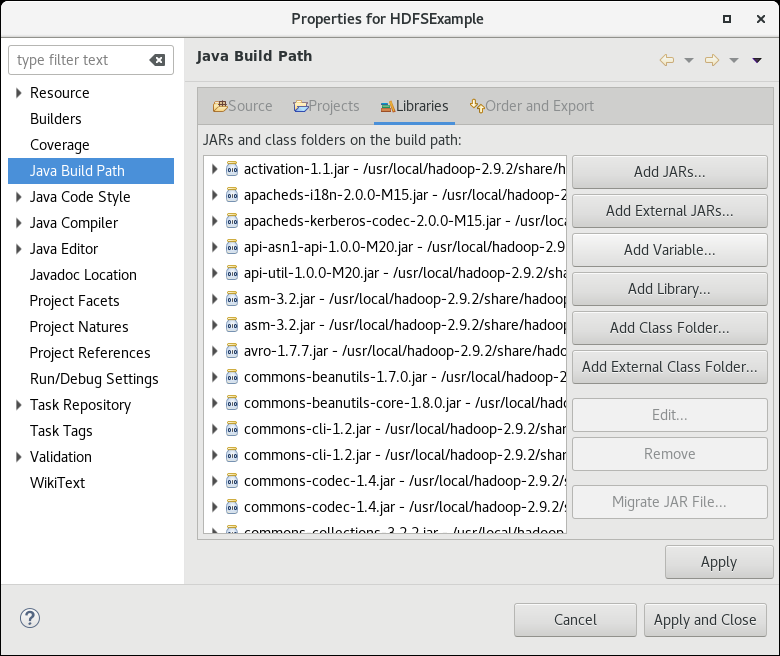


图2-25 完成JAR包添加后的界面

（5）自动返回到Eclipse界面，如图2-26所示，从图中可以看到，项目“HDFSExample”目录树下多了“Referenced Libraries”，内部有以上步骤添加进来的两个JAR包。

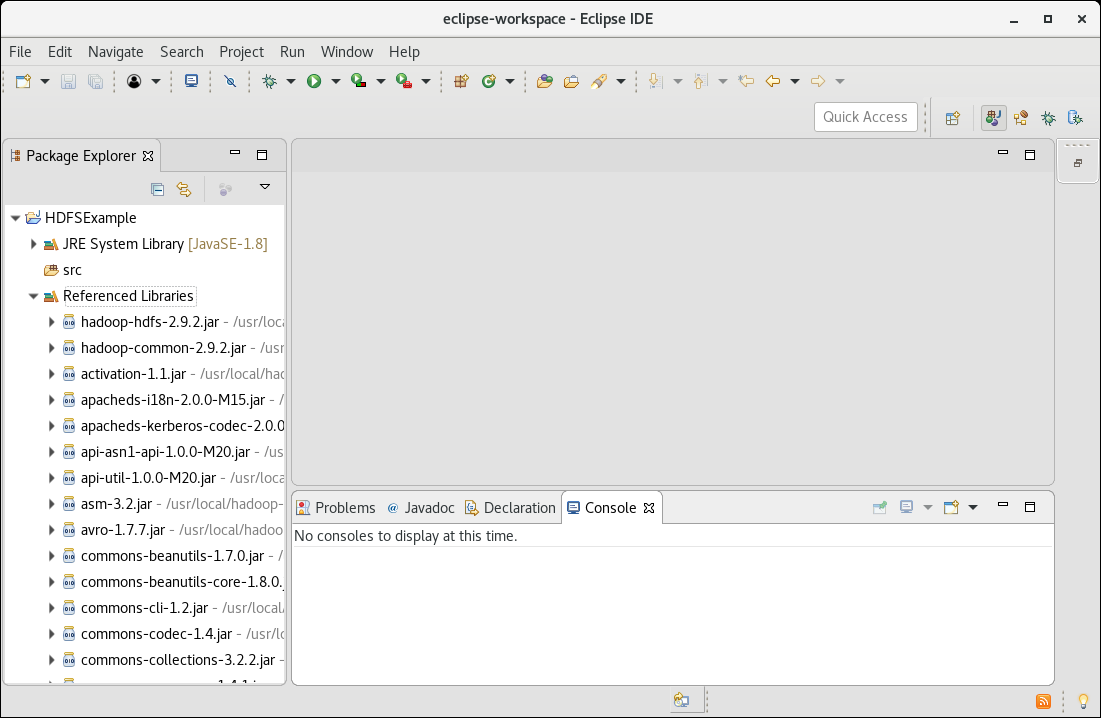


图2-26 添加JAR包后“HDFSExample”项目目录树变化

3. 在项目中新建包

（1）右键单击Java项目“HDFSExample”，从弹出的菜单中选择『New』→『Package』，如图2-27所示。

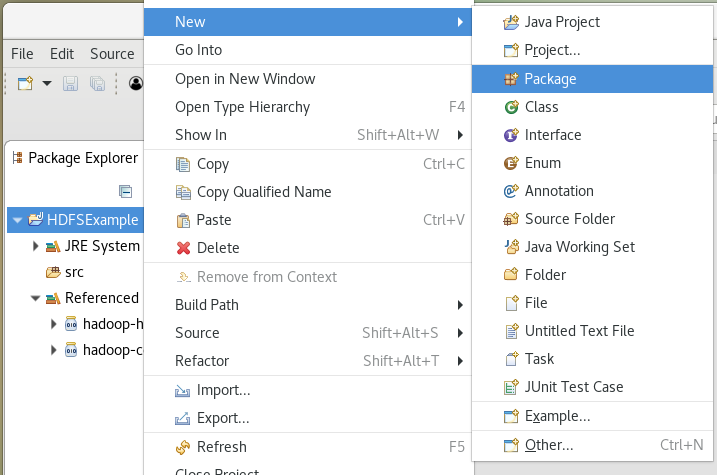


图2-27 进入“ZooKeeperExample”项目新建包窗口

（2）进入窗口【New Java Package】，输入新建包的名字，例如“com.xijing.hdfs”，如图2-28所示，完成后单击Finish按钮。

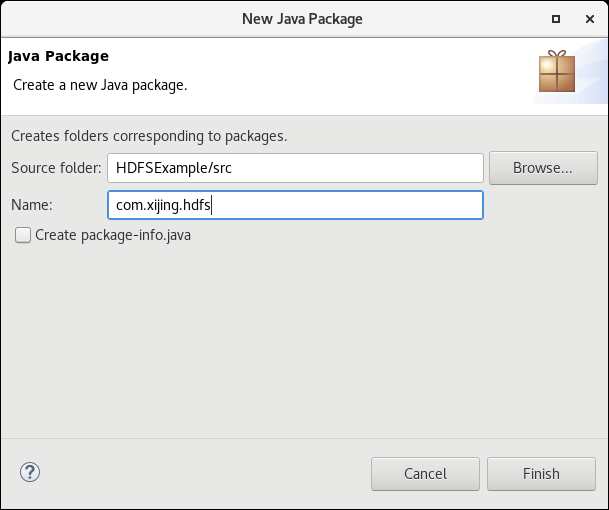


图2-28 新建包“com.xijing.hdfs”

4. 编写Java程序

下面编写一个Java应用程序，借助HDFS Java API，实现上传本地文件到HDFS的功能，等价于HDFS Shell命令“hadoop fs -put”。

（1）右键单击Java项目“HDFSExample”中目录“src”下的包“com.xijing.hdfs”，从弹出的菜单中选择『New』→『Class』，如图2-29所示。

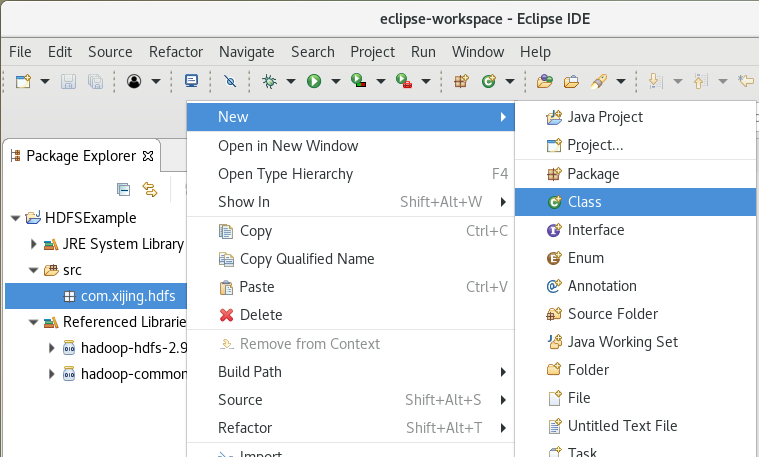


图2-29 进入“com.xijing.hdfs”包的新建类窗口

（2）进入窗口【New Java Class】。可以看出，由于上步在包“com.xijing.hdfs”下新建类，故此处不需要选择该类所属包；输入新建类的名字，例如“UploadFile”，之所以这样命名，是因为本程序实现的是上传本地文件到HDFS，建议读者命名时也要做到见名知意；读者还可以选择是否创建main函数。本实验中新建类“UploadFile”的具体输入和选择如图2-30所示。完成后单击Finish按钮。

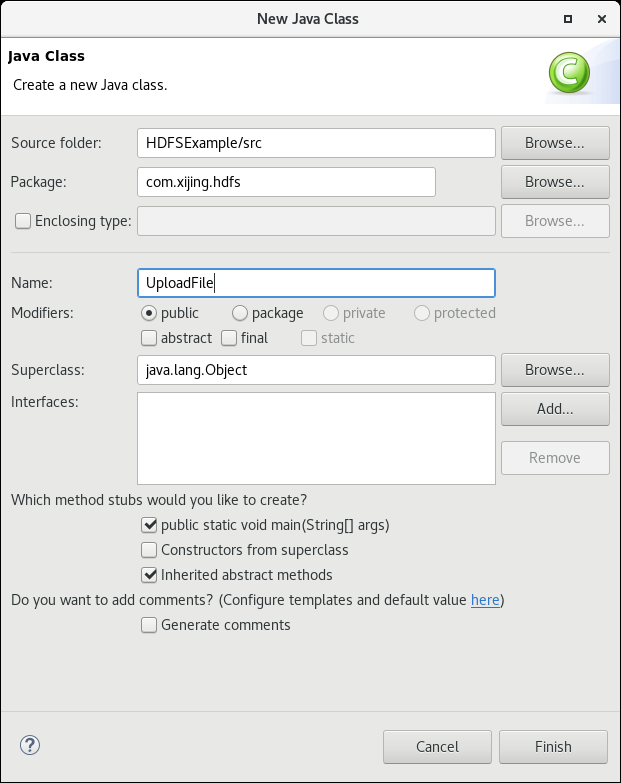


图2-30 新建类“ZnodeCDRW”

（3）自动返回到Eclipse界面，可以看到，Eclipse自动创建一个名为UploadFile.java的源代码文件，包、类、main方法已出现在代码中，如图2-31所示。

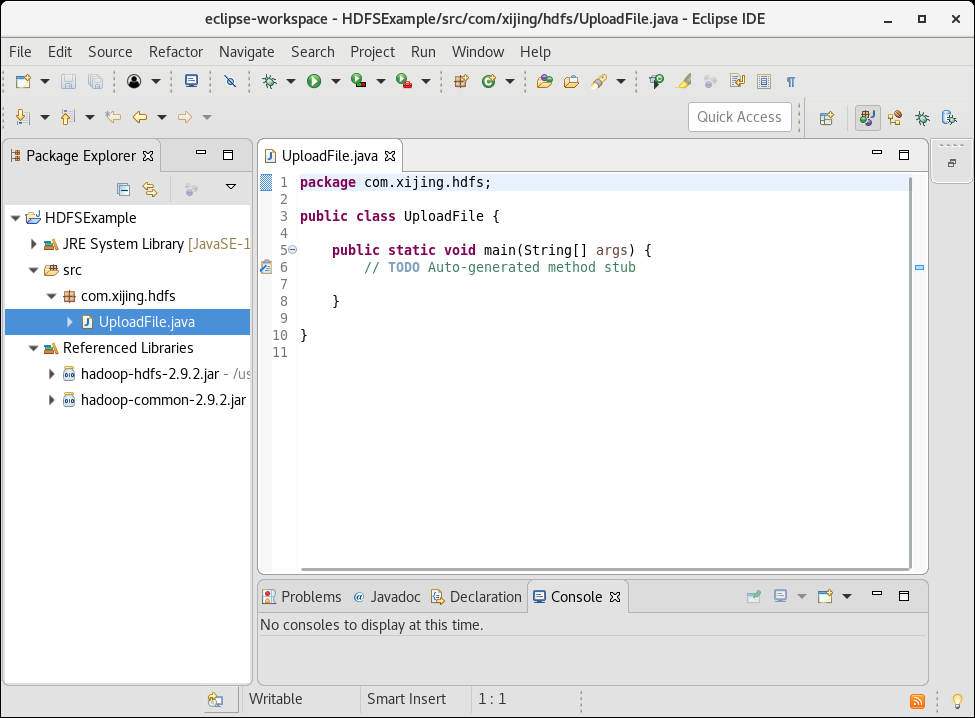


图2-31 新建类“UploadFile”后的Eclipse界面

（4）为实现程序功能，在该文件中添加代码，该程序完整代码如下所示。

package com.xijing.hdfs;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileStatus;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

public class UploadFile {

public static void main(String[] args) throws IOException {

Configuration conf = new Configuration();

FileSystem hdfs = FileSystem.get(conf);

Path src = new Path("/usr/local/hadoop-2.9.2/HelloData/file1.txt");

Path dst = new Path("file1.txt");

hdfs.copyFromLocalFile(src, dst);

System.out.println("Upload to " + conf.get("fs.defaultFS"));

FileStatus files[] = hdfs.listStatus(dst);

for (FileStatus file:files){

System.out.println(file.getPath());

}

}

}

本例中首先实例化Configuration对象，然后实例化了一个FileSystem对象hdfs，接着用到了FileSystem类的方法copyFromLocalFile(src, dst)，其功能是复制本地文件到目标文件系统指定路径下，最后通过代码“System.out.println(file.getPath())”将上传到HDFS上的文件路径显示出来。

5. 编译运行程序

1）本地执行

单击Eclipse工具栏中的Run按钮，直接运行UploadFile，执行结果如图2-32所示。从图2-32中可以看出，在/home/xuluhui/eclipse-workspace/HDFSExample目录下增加一个“file1.txt”文件，本地文件系统发生的变化如图2-33所示，file1.txt没有上传到HDFS上，使用命令“hadoop fs -ls /”查看不到file1.txt。

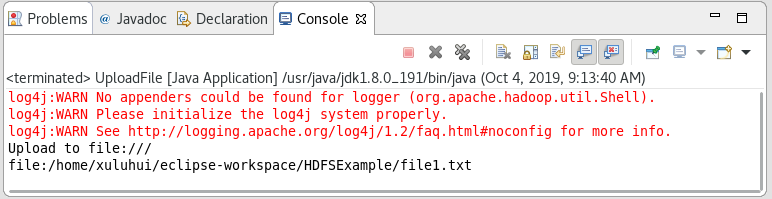


图2-32 UploadFile本地执行结果

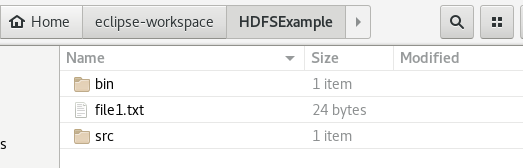


图2-33 UploadFile本地执行后本地文件系统发生的变化

因此，上述代码情况下，本地执行的方法是错误的。读者可以按照如下提示在上述代码“Configuration conf = ……”和“FileSystem hdfs = ……”之间加入一行代码，以达到本地执行目的。

……

Configuration conf = new Configuration();

conf.set("fs.defaultFS", "hdfs://master:9000"); //新加入代码行

FileSystem hdfs = FileSystem.get(conf);

……

修改后的UploadFile本地执行结果如图2-34所示。从图2-34中可以看出，在HDFS目录/user/xuluhui下增加一个“file1.txt”文件。

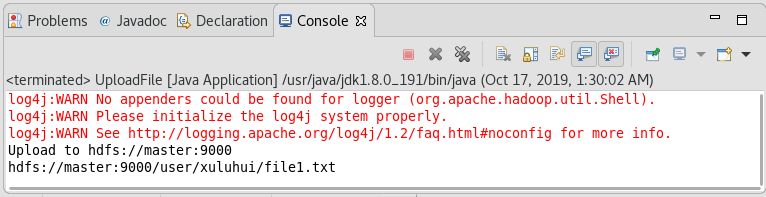


图2-34 修改代码后UploadFile本地执行结果

2）集群执行

（1）打包代码，生成jar文件。

第一步，右键单击Java项目“HDFSExample”，从弹出的快捷菜单中选择『Export…』，如图2-35所示。

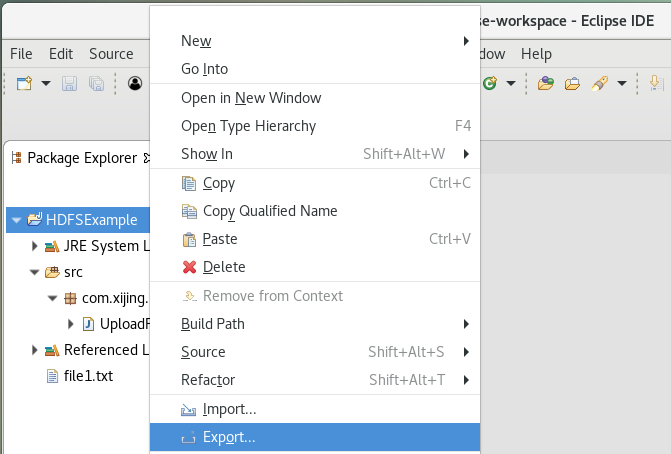


图2-35 生成jar包（1）

第二步，在弹出的“Export”对话框中，选择“JAR file”，单击按钮Next，如图2-36所示。

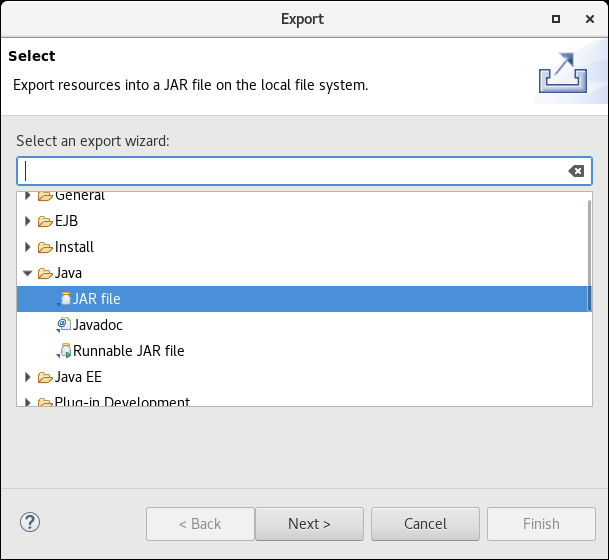


图2-36 生成jar包（1）

第三步，在弹出的“JAR Export”对话框中，指定jar包的存放路径等，本例中jar包的存放路径及文件名是/home/xuluhui/eclipse-workspace/HDFSExample/hdfsexample.jar，单击按钮Finish，如图2-37所示。

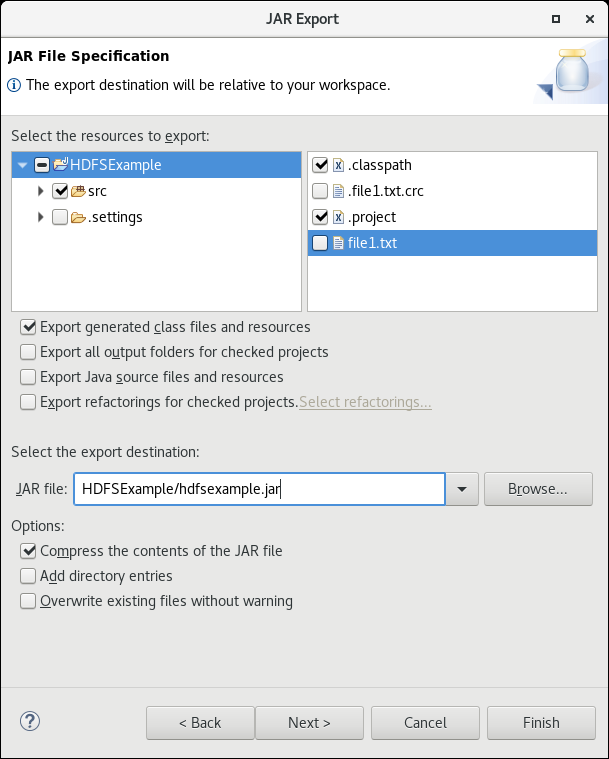


图2-37 生成jar包（3）

（2）使用命令“hadoop jar”将hdfsexample.jar提交到Hadoop集群执行，使用的命令及集群执行结果如图2-38所示。

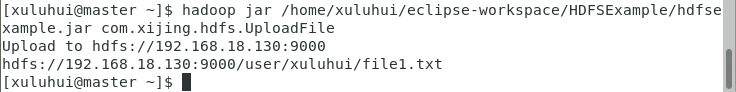


图2-38 UploadFile集群执行结果

（3）通过“hadoop fs -ls /”验证文件是否已上传，集群执行jar包和验证文件file1.txt是否上传成功如图2-39所示，发现本地文件已成功上传至HDFS。

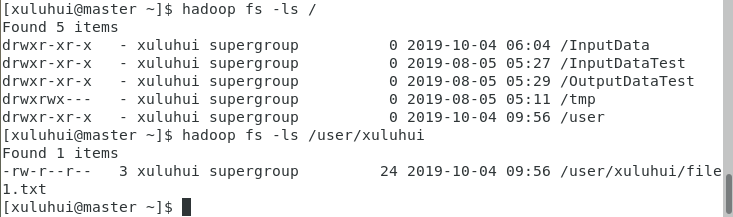


图2-39 验证本地文件是否已上传到HDFS

【案例2-5】使用HDFS Java API编写HDFS文件操作程序，实现查看HDFS文件的信息及在集群中位置的功能，采用本地执行和集群执行的两种执行方式测试，观察结果。

在已创建的Java项目“HDFSExample”包“com.xijing.hdfs”中新建类“FileLocation”，具体新建过程与【案例2-4】相同，此处不再赘述，查看【案例2-4】上传文件的“file1.txt”在HDFS集群中位置的完整代码如下所示。

package com.xijing.hdfs;

import java.io.IOException;

import java.net.URI;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.BlockLocation;

import org.apache.hadoop.fs.FileStatus;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

public class FileLocation {

public static void main(String[] args) {

String uri = "hdfs://master:9000/user/xuluhui/file1.txt";

Configuration conf = new Configuration();

try {

FileSystem fs = FileSystem.get(URI.create(uri),conf);

Path fpath = new Path(uri);

FileStatus filestatus = fs.getFileStatus(fpath);

// 文件名称

System.out.println("FileName: "+ filestatus.getPath().getName());

// 文件路径

System.out.println("FilePath: "+ filestatus.getPath());

// 文件大小

System.out.println("FileLength: "+ filestatus.getLen());

// 文件权限

System.out.println("FilePermission: "+ filestatus.getPermission());

// 文件所有者

System.out.println("FileGroup: "+ filestatus.getOwner());

// 获取文件存储的块信息

BlockLocation[] blockLocations = fs.getFileBlockLocations(filestatus, 0, filestatus.getLen());

for (BlockLocation blockLocation : blockLocations) {

// 获取文件块存储的主机节点

String[] hosts = blockLocation.getHosts();

for (String host : hosts) {

System.out.println("StoredNode: "+host);

}

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

将【案例2-4】生成的/home/xuluhui/eclipse-workspace/HDFSExample/hdfsexample.jar删除，重新打包，将FileLocation也打包进去。使用命令“hadoop jar”将全新的hdfsexample.jar提交到Hadoop集群执行，使用的命令及集群执行结果如图2-40所示。

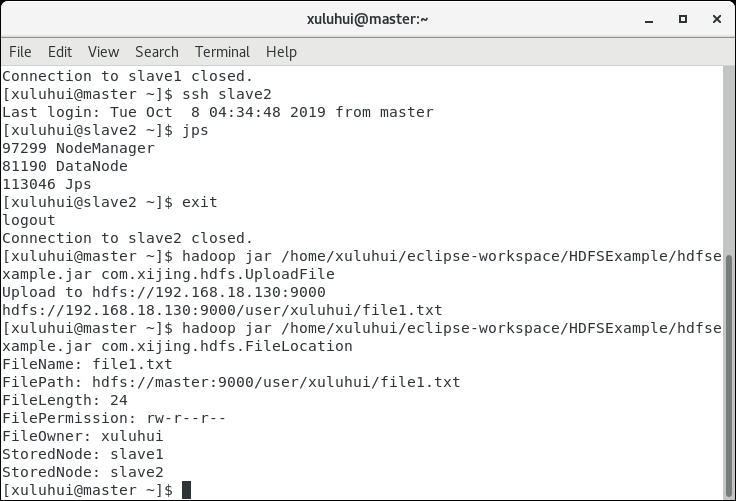


图2-40 FileLocation集群执行结果

当然，读者也可以使用本地执行的方式执行FileLocation，执行结果如图2-41所示。

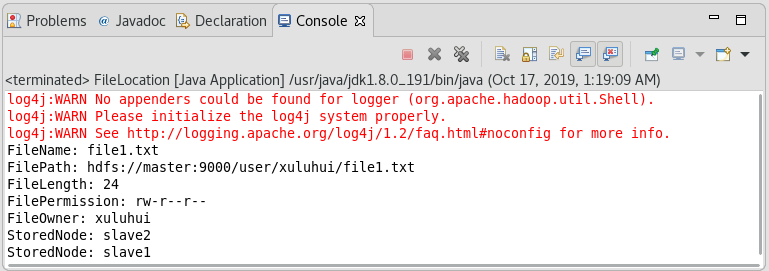


图2-41 FileLocation本地执行结果

使用HDFS Java API实现在HDFS上创建目录的程序代码如下所示，其功能等价于HDFS Shell命令“hadoop fs -mkdir”。

package com.xijing.hdfs;

import java.io.IOException;

import java.net.URI;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

public class CreateDir {

public static void main(String[] args) {

String uri = "hdfs://master:9000";

Configuration conf = new Configuration();

try {

FileSystem fs = FileSystem.get(URI.create(uri),conf);

Path dfs = new Path("/test");

boolean flag=fs.mkdirs(dfs);

System.out.println(flag?"directory creation success":"directory creation failure");

} catch (IOException e) {

e.printStackTrace();

}

}

}

使用HDFS Java API实现读取HDFS上文件的程序代码如下所示，其功能等价于HDFS Shell命令“hadoop fs -cat。

package com.xijing.hdfs;

import java.io.BufferedReader;

import java.io.InputStreamReader;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.fs.FSDataInputStream;

public class ReadFile {

public static void main(String[] args) {

try {

Configuration conf = new Configuration();

conf.set("fs.defaultFS","hdfs://master:9000");

conf.set("fs.hdfs.impl","org.apache.hadoop.hdfs.DistributedFileSystem");

FileSystem fs = FileSystem.get(conf);

Path file = new Path("test");

FSDataInputStream getIt = fs.open(file);

BufferedReader d = new BufferedReader(new InputStreamReader(getIt));

String content = d.readLine(); //读取文件一行

System.out.println(content);

d.close(); //关闭文件

fs.close(); //关闭hdfs

} catch (Exception e) {

e.printStackTrace();

}

}

}

使用HDFS Java API实现新建HDFS文件并写入内容的程序代码如下所示。

mport org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.FSDataOutputStream;

import org.apache.hadoop.fs.Path;

public class WriteFile {

public static void main(String[] args) {

try {

Configuration conf = new Configuration();

conf.set("fs.defaultFS","hdfs://master:9000");

conf.set("fs.hdfs.impl","org.apache.hadoop.hdfs.DistributedFileSystem");

FileSystem fs = FileSystem.get(conf);

byte[] buff = "Hello world".getBytes(); // 要写入的内容

String filename = "test"; //要写入的文件名

FSDataOutputStream os = fs.create(new Path(filename));

os.write(buff,0,buff.length);

System.out.println("Create:"+ filename);

os.close();

fs.close();

catch (Exception e) {

e.printStackTrace();

}

}

}

（六）关闭Hadoop集群

关闭全分布模式Hadoop集群的命令与启动命令次序相反，只需在主节点master上依次执行以下3条命令即可关闭Hadoop。

mr-jobhistory-daemon.sh stop historyserver

stop-yarn.sh

stop-dfs.sh

执行mr-jobhistory-daemon.sh stop historyserver时，其\*historyserver.pid文件消失；执行stop-yarn.sh时，\*resourcemanager.pid和\*nodemanager.pid文件依次消失；stop-dfs.sh，\*namenode.pid、\*datanode.pid、\*secondarynamenode.pid文件依次消失。

六、实验报告要求

实验报告以电子版和打印版双重形式提交。

实验报告主要内容包括实验名称、实验类型、实验地点、学时、实验环境、实验原理、实验步骤、实验结果、总结与思考等。实验报告格式如表1-9所示。

七、拓展训练——搭建HDFS NameNode HA

【案例2-6】假设某一集群共有8台机器，其中JournalNode和ZooKeeper保持奇数个节点，每个节点进程分布如表2-3所示。试对该集群进行HDFS NameNode HA高可用环境搭建。

表2-3 HDFS集群规划表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | master1 | master2 | master3 | slave1 | slave2 | slave3 | slave4 | slave5 |
| NameNode | √ | √ |  |  |  |  |  |  |
| DataNode |  |  |  | √ | √ | √ | √ | √ |
| JournalNode | √ |  | √ |  |  |  |  |  |
| ZooKeeper | √ | √ | √ |  |  |  |  |  |
| DFSZKFailover-Controller | √ | √ |  |  |  |  |  |  |

HDFS NameNode高可用的环境搭建具体步骤如下：

（1）安装ZooKeeper集群

在master1、master2、master3三个节点上安装ZooKeeper集群，具体方法参见第5章，此处不再赘述。

（2）安装全分布模式Hadoop集群

在master1、master2、master3、slave1、slave2、slave3、slave4、slave5八个节点上安装Hadoop，具体方法参见实验1，与实验1不同之处在于配置文件core-site.xml、hdfs-site.xml、slaves的内容有差异，具体如下所示。

* 配置文件core-site.xml

<configuration>

<!-- 集群中命名服务列表，名称自定义 -->

<property>

<name>fs.defaultFS</name>

<value>hdfs://xijingcluster</value>

</property>

<!-- NameNode、DataNode、JournalNode等存放数据的公共目录，用户可以单独指定这3类节点的目录，其中hdfsdata是自动生成的 -->

<property>

<name>hadoop.tmp.dir</name>

<value>/usr/local/hadoop-2.9.2/hdfsdata</value>

</property>

<!—ZooKeeper集群的地址和端口，注意，数量一定是奇数，且不少于3个节点 -->

<property>

<name>ha.zookeeper.quorum</name>

<value>master1:2181,master2:2181,master3:2181</value>

</property>

<configuration>

* 配置文件hdfs-site.xml

<configuration>

<!-- 给HDFS集群起名字，此名字必须和core-site.xml中的一致 -->

<property>

<name>dfs.nameservices</name>

<value>xijingcluster</value>

</property>

<!—指定NameService是xijingcluster时有哪些NameNode -->

<property>

<name>dfs.ha.namenodes.xijingcluster</name>

<value>master1,master2</value>

</property>

<!-- 指定RPC地址 -->

<property>

<name>dfs.namenode.rpc-address.xijingcluster.master1</name>

<value>master1:8020</value>

</property>

<property>

<name>dfs.namenode.rpc-address. xijingcluster.master2</name>

<value>master2:8020</value>

</property>

<!-- 指定HTTP地址 -->

<property>

<name>dfs.namenode.http-address. xijingcluster.master1</name>

<value>master1:50070</value>

</property>

<property>

<name>dfs.namenode.http-address. xijingcluster.master2</name>

<value>master2:50070</value>

</property>

<!-- 指定xijingcluster是否启动自动故障恢复，即当NameNode发生故障时，是否自动切换到另一台NameNode -->

<property>

<name>dfs.ha.automatic-failover.enabled.xijingcluster</name>

<value>true</value>

</property>

<!-- 指定xijingcluster的两个NameNode共享edits文件目录时，使用的JournalNode集群信息 -->

<property>

<name>dfs.namenode.shared.edits.dir</name>

<value>qjournal://master1:8485;master2:8485;master3:8485/xijingcluster</value>

</property>

<!-- 指定xijingcluster出现故障时，哪个实现类负责执行故障切换，实现类有两个：ConfiguredFailoverProxyProvider和RequestHedgingProxyProvider -->

<property>

<name>dfs.client.failover.proxy.provider.xijingcluster</name>

<value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>

</property>

<!-- 指定JournalNode集群在对NameNode目录进行共享时，自己存储数据的磁盘路径，其中journal是启动JournalNode时自动生成的 -->

<property>

<name>dfs.journalnode.edits.dir</name>

<value>/usr/local/hadoop-2.9.2/hdfsdata/journal</value>

</property>

<!-- 一旦需要NameNode切换，使用sshfence方式进行操作，除此之外，还提供有shell方式 -->

<property>

<name>dfs.ha.fencing.methods</name>

<value>sshfence</value>

</property>

<!-- 使用sshfence方式进行故障切换，需要配置无密码登录，指定使用ssh通信时所用密钥的存储位置 -->

<property>

<name>dfs.ha.fencing.ssh.private-key-files</name>

<value>/home/xuluhui/.ssh/id\_rsa</value>

</property>

<configuration>

* 配置文件slaves

在slaves中添加哪些节点是DataNode，这里指定slave1~slave5，此配置文件的内容如下。

slave1

slave2

slave3

slave4

slave5

（3）启动集群

① 启动ZooKeeper集群

在master1~master3上分别执行如下命令：

zkServer.sh start

在master1~master3上使用命令“zkServer.sh status”查看每个节点的ZooKeeper状态，实验4已讲述过，正确的状态是只有一个节点是leader，其余均为follower。

② 格式化ZooKeeper集群

格式化ZooKeeper集群的目的是在ZooKeeper集群上建立HA的相应节点，在master1上执行如下命令：

hdfs zkfc -formatZK

③ 启动JournalNode集群

在master1~master3上分别执行如下命令：

hadoop-daemon.sh start journalnode

格式化集群的某一个NameNode，只有第一次启动时需要进行格式化，这里选择master1，在master1上执行如下命令：

hdfs namenode -format

④ 启动刚格式化过的NameNode

由于上步格式化了节点master1的NameNode，因此在master1上执行如下命令：

hadoop-daemon.sh start namenode

将刚格式化的NameNode信息同步到备用NameNode上（第一次启动时需要，以后不需要），在master2上执行如下命令：

hdfs namenode -bootstrapStandby

然后在master2上启动NameNode，执行如下命令：

hadoop-daemon.sh start namenode

⑤ 启动所有的DataNode

DataNode是在slaves文件中配置的，在master1上执行如下命令：

hadoop-daemon.sh start datanode

⑥ 启动ZKFailoverController

在master1、master2上分别执行如下命令；

hadoop-daemon.sh start zkfc

⑦ 验证HDFS NameNode高可用机制的故障自动转移功能

打开http://master1:50070和http://master2:50070两个Web页面，观察哪个节点是Active NameNode，哪个节点是Standby NameNode。假设此处master1是Active的，通过“jps”命令获取该节点的NameNode进程id，然后执行命令“kill -9 pid”（其中pid是进程NameNode的进程号）将该进程杀死。刷新两个HDFS节点的Web界面，可以看到master2节点的状态由原来的Standby变成现在的Active，并且HDFS还能进行读写操作。这就说明，高可用机制的故障自动转换功能是正常的，HDFS NameNode是高可用的，而且主备NameNode切换过程对用户来说是不透明的。

思考与练习题

1. 使用HDFS Java API编写HDFS文件操作程序，实现对HDFS的文件操作：重命名文件、删除文件。

2. 使用HDFS Java API编写HDFS文件操作程序，实现对HDFS的目录操作：读取某个目录下的所有文件，删除目录。

参考文献

[1] 蔡斌. Hadoop技术内幕:深入解析Hadoop Common和HDFS架构设计与实现原理[M]. 北京:机械工业出版社,2013.

[2] GHEMAWAT S, GOBIOFF H, LEUNG S-T. The Google file system[C]// SOSP '03 Proceedings of the nineteenth ACM symposium on Operating systems principles, 2003,37(5):29-43.

[3] Apache Software Foundation. Apache Hadoop 2.9.2-HDFS Architecture[EB/OL]. [2018-11-13]. https://hadoop.apache.org/docs/r2.9.2/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

[4] Apache Software Foundation. Apache Hadoop 2.9.2-HDFS Users Guide[EB/OL]. [2018-11-13]. https://hadoop.apache.org/docs/r2.9.2/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html.

[5] Apache Software Foundation. Apache Hadoop 2.9.2-HDFS Commands Guide[EB/OL]. [2018-11-13]. https://hadoop.apache.org/docs/r2.9.2/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html.

[6] Apache Software Foundation. Apache Hadoop 2.9.2-Apache Hadoop Main 2.9.2 API[EB/OL]. [2018-11-13]. https://hadoop.apache.org/docs/r2.9.2/api/index.html.

[7] Apache Software Foundation. Apache Hadoop 2.9.2-HDFS High Availability Using the Quorum Journal Manager[EB/OL]. [2018-11-13]. https://hadoop.apache.org/docs/r2.9.2/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html.