# 第3章

# HDFS

## 3.1 相关基本概念

(1) 文件系统。文件系统是操作系统提供的用于解决“如何在磁盘上组织文件”的一系列方法和数据结构。

(2) 分布式文件系统。分布式文件系统是指利用多台计算机协同作用解决单台计算机所不能解决的存储问题的文件系统。如单机负载高、数据不安全等问题。

(3) HDFS(Hadoop Distributed File System,Hadoop 分布式文件系统)。它是Hadoop 项目的核心子项目,是分布式计算中数据存储管理的基础,它是基于流式数据访问和处理超大文件的需求而开发的分布式文件系统,可以运行于廉价的商用服务器上。HDFS 源于谷歌公司在 2003 年 10 月份发表的 GFS(Google File System)论文。

## 3.2 HDFS 存储架构

HDFS 采用 Master/Slave 架构。一个 HDFS 集群是由一个或几个 NameNode 和一定数量的 DataNode 组成的。HDFS 上的文件是以数据块的形式存放的,这些数据块存储在一组 DataNode 上。NameNode 执行文件系统的命名空间操作,比如打开、关闭、重命名文件或目录;也负责确定数据块到具体 DataNode 节点的映射。DataNode 负责处理文件系统客户端的读写请求,并在 NameNode 的统一调度下执行数据块的创建、删除和复制。HDFS 存储架构如图 3-1 所示。

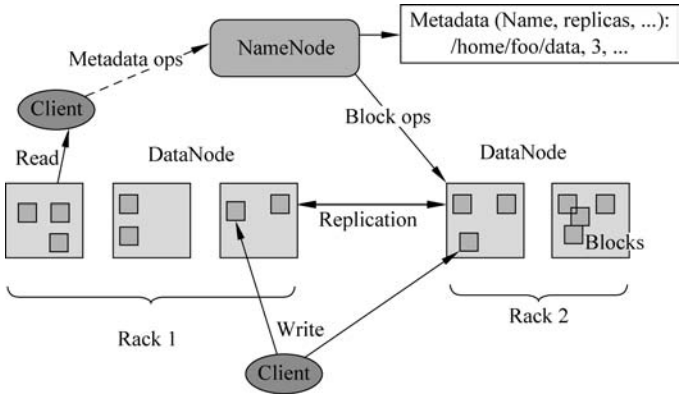


图 3-1 HDFS 存储架构

### 3.2.1 HDFS 写入流程

(1) Hadoop 客户端和 NameNode 通信请求上传文件,NameNode 检查目标文件是否已存在、父目录是否存在。

(2) NameNode 返回信息给 Hadoop 客户端是否可以上传。

(3) Hadoop 客户端会先对文件进行切分,如一个 Block(块)大小为 128MB,如果上传文件为 300MB,文件会被切分成 3 个 Block,其中两个 Block 为 128MB,一个 Block 为 44MB,并向 NameNode 发上传请求。

(4) NameNode 返回 DataNode 的服务器信息给 Hadoop 客户端。

(5) Hadoop 客户端请求一台 DataNode 上传数据(本质上是一个 RPC 调用,建立通道),第一个 DataNode 收到请求会继续调用第二个 DataNode,然后第二个 DataNode 调用第三个 DataNode,将整个通道建立完成,逐级返回 Hadoop 客户端。

(6) Hadoop 客户端开始往第一个 DataNode 上传第一 Block(先从磁盘读取数据放到一个本地内存缓存),以 Packet 为单位(一个 Packet 为 64KB),当然在写入时通道会进行数据校验,它并不是通过一个 Packet 进行一次校验而是以 checksum 为单位进行校验(512B),第一台 DataNode 收到一个 Packet 就会传给第二台,第二台传给第三台; 第一台每传一个 Packet 会放入一个应答队列等待应答。

(7) 当一个 Block 传输完成之后,Hadoop 客户端再次请求 NameNode 上传第二个 Block 的 DataNode 服务器,直至所有的 Block 上传完成。

### 3.2.2 HDFS 读取流程

(1) Hadoop 客户端向 NameNode 发送请求,获得存放在 NameNode 节点上文件的 Block 位置映射信息。

(2) NameNode 把文件所有 Block 的位置信息返回给 Hadoop 客户端。

(3) Hadoop 客户端拿到 Block 的位置信息后并行读取 Block 信息,Block 默认有 3 个副本,所以每一个 Block 只需要从一个副本读取。

(4) Hadoop 客户端从 DataNode 上取回文件的所有 Block 按照一定的顺序组成最终需要的文件。

## 3.3 HDFS 的优点与缺点

### 3.3.1 HDFS 的优点

(1)高容错性。数据自动保存多个副本。它通过增加副本的形式提高容错性。某一个副本丢失以后,它可以自动恢复。

(2) 适合处理高吞吐量。HDFS 通过移动计算而不是移动数据,并将数据位置暴露给计算框架,对计算的时延不敏感。

(3) 适合存储和管理大规模数据。HDFS 处理数据达到太字节(TB)甚至皮字节(PB)级别的数据,文件数量达百万规模以上,节点处理可达 1 万节点的规模。

(4) 适合一次写入,多次读取。文件一旦写入不能修改,只能追加,能够保证数据的一致性。

(5) 适合处理非结构化数据。HDFS 可以处理多类型的数据(音频、视频、文本)。

### 3.3.2 HDFS 的缺点

(1) 不适合低延时数据访问。HDFS 适合高吞吐率的场景,就是在某一段时间内写入大量的数据。但是 HDFS 不适合低延时的数据访问,比如毫秒级以内读取数据是很难做到的。

(2) 不适合小文件存储。小文件是指小于 HDFS 系统的 Block 大小的文件(1.0 版本默认为 64MB,2.0 版本默认为 128MB)的文件,小文件存储会占用 NameNode 大量的内存来存储文件、目录和块信息。而 NameNode 的内存是有限的。

(3) 不支持文件随机修改。HDFS 不允许多个线程同时写文件;仅支持数据追加,不支持文件的随机修改。

## 3.4 HDFS Shell 常用命令

HDFS Shell 可以直接与 Hadoop 分布式文件系统进行交互。HDFS Shell 常用命令如表 3-1 所示。

**表 3-1 HDFS Shell 常用命令**

| 命令参数 | 功能描述 | 命令参数 | 功能描述 |
|---|---|---|---|
| -ls | 查看指定路径的目录结构 | -text | 源文件输出为文本格式 |
| -du | 统计目录下所有文件大小 | -mkdir | 创建空白文件夹 |
| -mv | 移动文件 | -put | 上传文件 |
| -cp | 复制文件 | -get | 下载文件 |
| -rm | 删除文件/空白文件夹 | -help | 获得帮助信息 |
| -cat | 查看文件内容 | -cat | 查看文件内容 |

例如,列出文件或文件夹 Shell 命令为 hadoop fs -ls/,运行结果如下:

```
[root@bigdata01 /]# hadoop fs -ls /
Found 13 items
drwxr-xr-x   - root supergroup          0 2020-06-12 11:02 /HIVE
drwxr-xr-x   - root supergroup          0 2020-06-05 14:14 /InvertedIndex
drwxr-xr-x   - root supergroup          0 2020-09-28 10:21 /flumelogs
drwxr-xr-x   - root supergroup          0 2020-06-14 22:48 /hadoop
drwx--x--x   - root supergroup          0 2020-09-10 10:17 /hbase
drwxr-xr-x   - root supergroup          0 2020-06-06 11:15 /hive
drwxr-xr-x   - root supergroup          0 2020-06-03 14:30 /input
drwxr-xr-x   - root supergroup          0 2020-06-03 15:00 /out
drwxr-xr-x   - root supergroup          0 2020-06-04 16:33 /output
drwxr-xr-x   - root supergroup          0 2020-06-05 14:49 /output3
-rw-r--r--   1 root supergroup         33 2020-09-15 11:11 /people
-rw-r--r--   1 root supergroup       2545 2020-10-07 13:38 /profile
drwx-wx-wx   - root supergroup          0 2020-06-03 14:18 /tmp
```

创建文件夹 Shell 命令为 hadoop fs -mkdir/wcdoc,运行结果如下:

```
[root@bigdata01 /]# hadoop fs -mkdir /wcdoc
[root@bigdata01 /]# hadoop fs -ls /
Found 15 items
drwxr-xr-x   - root supergroup          0 2020-06-12 11:02 /HIVE
drwxr-xr-x   - root supergroup          0 2020-06-05 14:14 /InvertedInde
drwxr-xr-x   - root supergroup          0 2020-09-28 10:21 /flumelogs
drwxr-xr-x   - root supergroup          0 2020-06-14 22:48 /hadoop
drwx--x--x   - root supergroup          0 2020-09-10 10:17 /hbase
drwxr-xr-x   - root supergroup          0 2020-06-06 11:15 /hive
drwxr-xr-x   - root supergroup          0 2020-06-03 14:30 /input
drwxr-xr-x   - root supergroup          0 2020-06-03 15:00 /out
drwxr-xr-x   - root supergroup          0 2020-06-04 16:33 /output
drwxr-xr-x   - root supergroup          0 2020-06-05 14:49 /output3
-rw-r--r--   1 root supergroup         33 2020-09-15 11:11 /people
-rw-r--r--   1 root supergroup       2545 2020-10-07 13:38 /profile
drwx-wx-wx   - root supergroup          0 2020-06-03 14:18 /tmp
drwxr-xr-x   - root supergroup          0 2020-10-08 13:13 /user
drwxr-xr-x   - root supergroup          0 2020-10-08 13:14 /wcdoc
```

上传文件 Shell 命令为 hadoop fs -put/wc. txt/wcdoc,运行结果如下:

```
[root@bigdata01 /]# hadoop fs -put /wc.txt /wcdoc
[root@bigdata01 /]# hadoop fs -ls /wcdoc
Found 4 items
drwxr-xr-x   - root supergroup          0 2020-10-08 13:24 /wcdoc/bin
drwxr-xr-x   - root supergroup          0 2020-10-08 13:24 /wcdoc/boot
drwxr-xr-x   - root supergroup          0 2020-10-08 13:24 /wcdoc/dev
-rw-r--r--   1 root supergroup         53 2020-10-08 13:25 /wcdoc/wc.txt
```

下载文件 Shell 命令为 hadoop fs -get/wcdoc/wc. txt/wc1. txt,运行结果如下:

```
[root@bigdata01 /]# hadoop fs -get /wcdoc/wc.txt /wc1.txt
[root@bigdata01 /]# ls
bin   etc        file3.txt  lib    mnt    people.json  run   sys   usr      wc.txt
boot  file1.txt  hadoop     lib64  opt    proc         sbin  tmp   var      words.txt
dev   file2.txt  home       media  people root         srv   tools wc1.txt
```

查看文件内容 Shell 命令为 hadoop fs -cat/wcdoc/wc. txt,运行结果如下:

```
[root@bigdata01 /]# hadoop fs -cat /wcdoc/wc.txt
hello hadoop
hello mapreduce
bye hadoop
```

删除文件(夹)Shell 命令为 hadoop fs -rm/wcdoc/wc. txt,运行结果如下:

```
[root@bigdata01 /]# hadoop fs -rm /wcdoc/wc.txt
20/10/08 13:29:28 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Empt
ier interval = 0 minutes.
Deleted /wcdoc/wc.txt
```

## 3.5 HDFS 的 Java API

通过编程的形式操作 HDFS,其核心是使用 HDFS 提供的 Java API 构造一个访问客户端对象,然后通过客户端对象对 HDFS 上的文件进行操作(增加、删除、查找)。

在 Java 中操作 HDFS,创建一个客户端实例主要涉及以下两个类：一是 Configuration 类,该类的对象封装了客户端或者服务器的配置,从中获取 Hadoop 集群的配置信息；二是 FileSystem 类,该类的对象是一个文件系统对象。

FileSystem 对象的一些方法可以对文件进行操作,HDFS 常用 Java API 如表 3-2 所示。

**表 3-2 HDFS 常用 Java API**

| 方 法 名 | 功 能 |
|---|---|
| copyFromLocalFile(Path src,Path dst) | 从本地磁盘复制文件到 HDFS |
| copyToLocalFile(Path src,Path dst) | 从 HDFS 复制文件到本地磁盘 |
| mkdirs(Path f) | 建立子目录 |
| rename(Path src,Path dst) | 重命名文件或文件夹 |
| delete(Path f) | 删除指定文件 |

搭建测试项目具体步骤如下所述。

(1) 创建一个 MAVEN 项目,并在项目的 pom.xml 文件中引入 hadoop-common、hadoop-hdfs、hadoop-client 以及单元测试 junit 的依赖,pom.xml 文件中的依赖包如下：

```
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-common</artifactId>
    <version>2.7.4</version>
</dependency>
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-hdfs</artifactId>
    <version>2.7.4</version>
</dependency>
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-client</artifactId>
    <version>2.7.4</version>
</dependency>
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-MapReduce-client-core</artifactId>
    <version>2.7.4</version>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
</dependency>
<dependency>
    <groupId>org.apache.zookeeper</groupId>
    <artifactId>zookeeper</artifactId>
```

```
    <version>3.4.10</version>
</dependency>
```

（2）编写 Java 类 HDFS_API_TEST，代码和相关说明如下：

```
package com.chapter03.hdfsdemo;
import java.io.FileNotFoundException;
import java.io.IOException;
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.LocatedFileStatus;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.RemoteIterator;
import org.junit.Before;
import org.junit.Test;
public class HDFS_API_TEST{
    FileSystem fs = null;
    @Before
    public void init()throws Exception{
        //构造配置参数对象
        Configuration conf = new Configuration();
        //设置访问的 HDFS 的 URI
        conf.set("fs.defaultFS","hdfs://172.16.106.69:9000");
        //设置本机的 Hadoop 路径
        System.setProperty("hadoop.home.dir","D:\\hadoop");
        //设置客户端访问身份
        System.setProperty("HADOOP_USER_NAME","root");
        //通过 FileSystem 的静态 get()方法获取文件系统客户端对象
        fs = FileSystem.get(conf);
    }
    @Test
    public void AddFileToHdfs()throws IOException{
        //上传文件本地路径
        Path src = new Path("D:/hdfs.txt");
        //HDFS 的目标路径
        Path dst = new Path("/teshdfs");
        //上传文件方法
        fs.copyFromLocalFile(src,dst);
        //关闭资源
        fs.close();
    }
    //从 HDFS 中复制文件到本地文件系统
    @Test
          public void DownloadFileToLocal()throws IllegalArgumentException,IOException {
        //下载文件
        fs.copyToLocalFile(new Path("/testFile"),new Path("D:/"));
    }
    //创建、删除、重命名文件
```

```
@Test
public void MkdirAndDeleteAndRename()throws Exception{
    //创建目录
    fs.mkdirs(new Path("/test1"));
    fs.rename(new Path("/test1"),new Path("/test3"));
    //删除文件夹,如果是非空文件夹,则参数 2 必须赋值 true
    fs.delete(new Path("/test2"), true);
}
//查看目录信息
@Test
public void ListFiles()throws FileNotFoundException, IllegalArgumentException, IOException {
    //获取迭代器对象
    RemoteIterator<LocatedFileStatus> listFiles = fs.listFiles(new Path("/"), true);
    while (listFiles.hasNext()){
        LocatedFileStatus fileStatus = listFiles.next();
        //打印当前文件名
        System.out.println(fileStatus.getPath().getName());
        //打印当前文件块大小
        System.out.println(fileStatus.getBlockSize());
        //打印当前文件权限
        System.out.println(fileStatus.getPermission());
        //打印当前文件内容长度
        System.out.println(fileStatus.getLen());
        //获取该文件块信息(包含长度、数据块、DataNode 的信息)
        BlockLocation[] blockLocations = fileStatus.getBlockLocations();
        for (BlockLocation bl : blockLocations){
            System.out.println("block-length:" + bl.getLength() + "--" + "block-
        offset:" + bl.getOffset());
            String[] hosts = bl.getHosts();
            for (String host : hosts){
                System.out.println(host);
            }
        }
                }
}
//查看文件及文件夹信息
@Test
public void ListFileAll()throws FileNotFoundException,IllegalArgumentException,
IOException {
    //获取 HDFS 系统根目录的元数据信息
    FileStatus[] listStatus = fs.listStatus(new Path("/"));
        String filelog = "文件夹--       ";
        for (FileStatus fstatus : listStatus){
            //判断是文件还是文件夹
            if (fstatus.isFile())
                filelog = "文件--         ";
            System.out.println(filelog + fstatus.getPath().getName());
        }
    }
}
```

(3) 运行测试。

对于 Windows 机器上没有 Hadoop 的环境的客户端,需要下载 Hadoop 的服务进程 winutils.exe,下载地址为 https://codeload.github.com/srccodes/hadoop-common-2.2.0-bin/zip/master。解压后放在一个没有中文的文件夹下。在程序中指定路径,本例把它放在 D 盘 hadoop 目录下,同时在类的 init()方法中加入语句: System.setProperty("hadoop.home.dir", "D:\\hadoop")。指定客户端 Hadoop 的伪安装路径后,可以利用单元测试 junit 进行各个方法的运行测试。

## 3.6 本章小结

本章主要介绍了 Hadoop 中非常重要的分布式存储文件系统 HDFS,分析了 HDFS 的存储架构以及常用 Shell 命令和 Java API,并且对 Java API 的编程进行了实例演示。