第5章

HDFS技术

HDFS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，是基于硬盘迭代模式访问和处理超大离线文件的需求而开发的项目，可以运行于廉价的商用服务器或PC上。

HDFS有以下优点。

(1) 高容错性：上传的数据自动保存多个副本(默认为3个副本)。它通过增加副本的数量来增加它的容错性。如果某一个副本丢失，则HDFS机制会复制其他机器上的副本，而不必关注它的实现。

(2) 适合大数据的处理：能够处理千兆字节、太字节甚至拍字节级别的数据。

(3) 基于硬盘迭代的I/O写入：一次写入，多次读取。文件一旦写入，就不能修改，只能增加，这样可以保证数据的一致性。

(4) 可以装在廉价的机器上。

HDFS有以下缺点。

(1) 低延时数据访问：它在低延时的情况下是不行的，它适合高吞吐率的场景，即在某一时间内写入大量的数据。对低延时要求高的情况一般使用Spark来完成。

(2) 小文件的存储：如果存放大量的小文件，它会大量占用NameNode的内存存储文件、目录、块信息，这样会对NameNode节点造成负担，小文件存储的寻道时间会超过文件的读取时间。这违背了HDFS的设计目标。

(3) 不能并发写入、文件不能随机修改：一个文件只能由一个线程写，不能由多个线程同时写；仅支持文件的追加，不支持文件的随机修改。

5.1 HDFS 架构

关于 HDFS 架构的讲解视频可扫描二维码观看。

HDFS 的高可用是 HDFS 持续为各类客户端提供读写服务的能力基础，因为客户端在对 HDFS 的读写操作之前都要访问 NameNode 服务器，客户端只有从 NameNode 获取元数据之后才能继续进行读写。所以 HDFS 的高可用关键在于 NameNode 上元数据的持续可用。Hadoop 官方提供了一种 QuorumJournalManager 来实现高可用。在高可用配置下，editlog 存放在一个共享存储的地方，这个共享存储由若干个 JournalNode 组成，一般是 3 个虚拟机(JN 集群)，每个 JournalNode 专门用于存放来自 NameNode 的编辑日志(editlog)，编辑日志由活跃状态的名称节点写入。

要有两个 NameNode 节点，二者之中只能有一个处于活跃状态，另一个是备用状态。只有活跃状态的节点才能对外提供读写 HDFS 的服务，也只有活跃状态的节点才能向 JournalNode 写入编辑日志；备用状态的名称节点只负责从 JN 集群中的 JournalNode 节点复制数据到本地存放。另外，各个 DataNode 会定期向两个 NameNode 节点报告自己的状态(心跳信息、块信息)。

一主一从的两个 NameNode 节点同时和 3 个 JournalNode 节点构成的组保持通信，活跃状态的 NameNode 节点负责往 JournalNode 集群写入编辑日志，备用状态的 NameNode 节点负责观察 JournalNode 组中的编辑日志，并且把日志拉取到备用状态节点，再加入到两节点各自的 fsimage 镜像文件。这样一来就能确保两个 NameNode 的元数据保持同步。一旦活跃状态不可用，提前配置的 Zookeeper 会把备用状态节点自动变为活跃状态，继续对外提供服务。

对于 HA 群集的正确操作至关重要，因此一次只能有一个 NameNode 节点处于活跃状态。否则，命名空间状态将在两者之间迅速产生分歧，出现数据丢失或其他不正确的结果。为了确保这个属性并防止所谓的“分裂大脑情景”，JournalNode 将只允许一个 NameNode 作为“领导者”。在故障切换期间，变为活跃状态的 NameNode 节点将简单地接管写入 JournalNode 的角色，这将有效地防止其他 NameNode 节点继续处于活动状态，允许新的节点安全地进行故障切换。HDFS 高可用架构如图 5.1 所示。

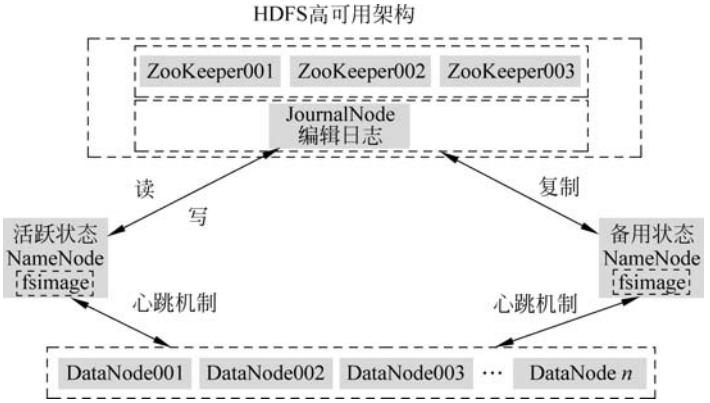

图 5.1 HDFS 高可用架构

5.2 HDFS 命令

关于 HDFS 命令的讲解视频可扫描二维码观看。

在 HDFS 中所有的 Hadoop 命令均由 bin/hadoop 脚本引发，不指定参数运行 Hadoop 脚本会打印所有命令的描述。本节将介绍常用的 HDFS 命令的操作。

5.2.1 version 命令

用法：hadoop version

version 命令可以打印 Hadoop 版本详细信息，示例如下：

```
[hadoop@Slave001 ~]$ hadoop version
    Hadoop 2.6.5
    Subversion https://github.com/apache/hadoop.git -r e8c9fe0b4c252caf2ebf1464220599650f119997
    Compiled by sjlee on 2016-10-02T23:43Z
    Compiled with protoc 2.5.0
    From source with checksum f05c9fa095a395faa9db9f7ba5d754
    This command was run using /home/hadoop/software/hadoop-2.6.5/share/hadoop/common/hadoop-common-2.6.5.jar
```

5.2.2 dfsadmin 命令

dfsadmin 命令可以查看集群存储空间使用情况和各个节点存储空间使用情况，示例如下：

```
[hadoop@Slave001 ~]# hadoop dfsadmin -report
    DEPRECATED: Use of this script to execute hdfs command is deprecated.
    Instead use the hdfs command for it.
    Configured Capacity: 37139136512 (34.59 GB)
    Present Capacity: 30914732032 (28.79 GB)
    DFS Remaining: 30734471168 (28.62 GB)
    DFS Used: 180260864 (171.91 MB)
    DFS Used%: 0.58%
    Under replicated blocks: 99
    Blocks with corrupt replicas: 0
    Missing blocks: 0
    -------------------------------------------------
    Live datanodes (2):
    Name: 192.168.153.201:50010 (Slave001)
    Hostname: Slave001
    Decommission Status : Normal
    Configured Capacity: 18569568256 (17.29 GB)
    DFS Used: 90128384 (85.95 MB)
    Non DFS Used: 3115909120 (2.90 GB)
```

```
DFS Remaining: 15363530752 (14.31 GB)
DFS Used%: 0.49%
DFS Remaining%: 82.73%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Last contact: Tue Dec 26 22:33:14 CST 2017

Name: 192.168.153.202:50010 (Slave002)
Hostname: Slave002
Decommission Status : Normal
Configured Capacity: 18569568256 (17.29 GB)
DFS Used: 90132480 (85.96 MB)
Non DFS Used: 3108495360 (2.90 GB)
DFS Remaining: 15370940416 (14.32 GB)
DFS Used%: 0.49%
DFS Remaining%: 82.77%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Last contact: Tue Dec 26 22:33:12 CST 2017
```

5.2.3 jar 命令

jar 命令是运行 jar 包文件的命令。用户可以把自己的 MapReduce 代码捆绑到 jar 文件中，使用 jar 命令使程序运行起来。

语法格式如下：

```
hadoop jar <jar> [mainClass]
```

其中，<jar>是 jar 包文件名称；[mainClass]是可选选项，指定运行主类。

使用 hadoop jar 命令可以在 Hadoop 集群中运行 WordCount.jar 程序，示例如下：

```
[hadoop@Slave001 ~]# hadoop jar WordCount.jar
```

5.2.4 fs 命令

fs 命令是运行通用文件系统命令。在 hadoop 命令后面跟上 fs 命令，表示对 HDFS 中的文件进行操作。

语法格式如下：

```
hadoop fs [GENERIC_OPTIONS] [COMMAND_OPTIONS]
```

[GENERIC_OPTIONS]：通用选项；
[COMMAND_OPTIONS]：命令选项。
fs 常用的基本选项如下。

1. cat

cat 命令可以在 HDFS 中查看指定文件或指定文件夹下所有文件内容。
语法格式如下：

```
hadoop fs -cat <hdfs:pathFile>
```

例如，查看 HDFS 中 input 目录下所有文件的内容。

```
[hadoop@Slave001 ~]# hadoop fs -cat /input/*
```

查看 HDFS 中 input 目录下 part-r-00000 文件中的内容。

```
[hadoop@Slave001 ~]# hadoop fs -cat /input/part-r-00000
```

注：
(1) /input/* 中的 * 代表所有，/input/* 代表 input 目录中的所有文件。
(2) /input/part* 代表 input 目录中文件名以 part 开始的所有文件。

2. copyFromLocal

copyFromLocal 命令类似于 put 命令，它与 put 命令的不同之处在于 copyFromLocal 命令复制的源地址必须是本地文件地址。
语法格式如下：

```
hadoop fs -copyFromLocal <local:pathFile> <hdfs:pathDirectory>
```

3. copyToLocal

copyToLocal 命令的作用与 put 命令很像，都是上传文件到 HDFS 中，它们的区别在于 copyToLocal 的源路径只能是一个本地文件，而 put 的源路径可能是多个文件，也可能是标准输入。
语法格式如下：

```
hadoop fs -copyToLocal <hdfs:pathFile> <local:pathDirectory>
```

除了限定目标路径是一个本地文件外，copyToLocal 命令和 get 命令类似。

4. cp

语法格式如下：

```
hadoop fs -cp <hdfs:pathFile> <hdfs:pathDirectory>
```

cp 命令可以将 HDFS 中的指定文件复制到 HDFS 目标路径中。这个命令允许有多个源路径，但此时目标路径必须是一个目录。

示例如下：

```
[hadoop@Slave001 ~]# hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2
[hadoop@Slave001 ~]# hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2 /user/
hadoop/dir
```

5. du

语法格式如下：

```
hadoop fs -du <hdfs:pathDirectory>
```

du 命令是显示文件或文件夹属性的命令，可以显示指定文件的大小、多个指定文件的大小、指定目录中所有文件的大小和指定多个目录中所有文件的大小。

示例如下：

```
[hadoop@Slave001 ~]# hadoop fs -du /input /output
73962       /output/relation
8829        /output/word
14215       /output/word_count
53          /input/2017-12-22.1514227013843
41          /input/2017-12-26.1514227103154
3297        /input/choose_column
```

6. dus

语法格式如下：

```
hadoop fs -dus <hdfs:pathDirectory>
```

dus 命令可以显示指定文件目录的大小或者指定多个文件目录的大小。

示例如下：

```
[hadoop@Slave001 ~]$ hadoop fs -dus /input
3169072     /input
[hadoop@Slave001 ~]$ hadoop fs -dus /input /output
3169072     /input
8829        /output
```

7. expunge

expunge 命令的字面意思是“清除”，它在 Hadoop 中的作用是清空回收站。

语法格式如下：

```
hadoop fs -expunge
```

示例如下：

```
[hadoop@Slave001 ~]$ hadoop fs -expunge
17/12/26 22:31:55 INFO fs.TrashPolicyDefault: NameNode trash configuration: Deletion
interval = 0 minutes, Emptier interval = 0 minutes.
```

8. get

语法格式如下：

```
hadoop fs -get <hdfs:pathFile> <local:pathDirectory>
```

get 命令从 HDFS 中复制指定文件、指定目录下所有文件和指定多个文件到本地文件目录。执行 get 命令之前，本地文件目录必须事先存在。get 命令是一个常用的下载命令。

示例如下：

(1) 在 HDFS 中复制 input 目录下 word_count 文件到本地 file 目录中。

```
[hadoop@Slave001 file]$ hadoop fs -get /input/word_count ~/file
```

(2) 复制 HDFS 中 input 和 output 目录所有文件到本地 file 目录中。

```
[hadoop@Slave001 file]$ hadoop fs -get /input /output ~/file
[hadoop@Slave001 file]$ ls
input    output
[hadoop@Slave001 file]$ cd input/
[hadoop@Slave001 input]$ ls
2017-12-22.1514227013843    2017-12-26.1514227103154    choose_column
city_data  major    monitor_data  score    word
```

9. getmerge

语法格式如下：

```
hadoop fs -getmerge <hdfs:pathDirectory> <hdfs:localFile>
```

getmerge 命令将 HDFS 中指定目录下的所有文件加载到本地文件中。如果文件名

不存在，则在本地创建新文件；如果文件名存在，则覆盖原文件内所有内容。

示例如下：

```
[hadoop@Slave001 file]$ hadoop fs -getmerge /output/word_count ~/file/output
[hadoop@Slave001 file]$ ls
file    output
[hadoop@Slave001 file]$ cat output
Just  出现：1 次
...
Not  出现：1 次
Noticing  出现：1 次
```

10. ls

语法格式如下：

```
hadoop fs -ls <hdfs:pathDirectory>
```

ls 命令在 HDFS 中显示指定文件的详细内容。如果是目录，则直接返回其子文件的列表。

详细内容包括权限、用户、文件所在组、文件大小、创建日期和路径等信息。

示例如下：

（1）显示 HDFS 中 Word 文件的详细属性。

```
[hadoop@Slave001 file]$ hadoop fs -ls /input/word
-rw-r--r--   3 hadoop   supergroup     8829 2017-12-25 22:15 /input/word
```

（2）显示 HDFS 中 input 目录下所有文件的详细属性。

```
[hadoop@Slave001 file]$ hadoop fs -ls /input
Found 4 items
-rw-r--r--   3 hadoop supergroup   482 2017-12-26 00:19 /input/major
-rw-r--r--   3 hadoop supergroup   2954128 2017-12-25 04:33 /input/monitor
-rw-r--r--   3 hadoop supergroup   56812 2017-12-25 23:40 /input/score
-rw-r--r--   3 hadoop supergroup   8829 2017-12-25 22:15 /input/word
```

11. lsr

语法格式如下：

```
hadoop fs -lsr <hdfs:pathDirectory>
```

lsr 命令是 ls -R 的简写，用来递归显示 HDFS 中指定目录下所有子文件。

示例如下：

```
[hadoop@Slave001 file]$ hadoop fs -lsr /input
lsr: DEPRECATED: Please use 'ls -R' instead.
-rw-r--r--   3 hadoop supergroup   53 2017-12-26 02:36 /input/151422013843
-rw-r--r--   3 hadoop supergroup   41 2017-12-26 02:38 /input/151422103154
-rw-r--r--   3 hadoop supergroup   3297 2017-12-25 23:05 /input/choose
-rw-r--r--   3 hadoop supergroup   14540 2017-12-25 04:52 /input/city_data
-rw-r--r--   3 hadoop supergroup   482 2017-12-26 00:19 /input/major
-rw-r--r--   3 hadoop supergroup   2954128 2017-12-25 04:33 /input/monitor
-rw-r--r--   3 hadoop supergroup   56812 2017-12-25 23:40 /input/score
-rw-r--r--   3 hadoop supergroup   8829 2017-12-25 22:15 /input/word
```

12. mkdir

语法格式如下：

```
hadoop fs -mkdir <paths>
```

mkdir 命令可以在 HDFS 中创建新目录，但它只能创建一级目录。创建多级目录时上一级目录必须先存在，或者使用-p 参数。

示例如下：

(1) 使用 mkdir 命令在 HDFS 的 input 目录下创建一个 file 目录。

```
[hadoop@Slave001 ~]$ hadoop fs -mkdir /input/file
[hadoop@Slave001 ~]$ hadoop fs -ls /input
Found 4 items
drwxr-xr-x   - hadoop supergroup    0 2017-12-26 23:30 /input/file
-rw-r--r--   3 hadoop supergroup  482 2017-12-26 00:19 /input/major
-rw-r--r--   3 hadoop supergroup  56812 2017-12-25 23:40 /input/score
-rw-r--r--   3 hadoop supergroup  8829 2017-12-25 22:15 /input/word
```

(2) 使用 mkdir 命令在 HDFS 的 input 目录下创建一个 file2 目录，在 output 目录下也创建一个 file2 目录。

```
[hadoop@Slave001 ~]$ hadoop fs -mkdir /input/file2 /output/file2
[hadoop@Slave001 ~]$ hadoop fs -ls /input /output
Found 6 items
drwxr-xr-x    - hadoop supergroup  0 2017-12-26 23:30 /input/file
drwxr-xr-x    - hadoop supergroup  0 2017-12-26 23:32 /input/file2
...
-rw-r--r--  3  hadoop supergroup  8829 2017-12-25 22:15 /input/word
Found 10 items
drwxr-xr-x    - hadoop supergroup  0 2017-12-25 23:41 /output/averager
drwxr-xr-x    - hadoop supergroup  0 2017-12-25 23:06 /output/choose
...
drwxr-xr-x    - hadoop supergroup  0 2017-12-26 23:32 /output/file2
```

（3）使用 mkdir 命令在 HDFS 中创建一个多级目录/file/file1/file2/file3。

```
[hadoop@Slave001 ~]$ hadoop fs -mkdir -p /file/file1/file2/file3
[hadoop@Slave001 ~]$ hadoop fs -ls -R /file
drwxr-xr-x   - hadoop supergroup     0 2017-12-26 23:35 /file/file1
drwxr-xr-x   - hadoop supergroup     0 2017-12-26 23:35 /file/file1/file2
drwxr-xr-x   - hadoop supergroup     0 2017-12-26 23:35 /file/file1/file2/file3
```

13. mv

语法格式如下：

```
hadoop fs -mv <hdfs:sourcepath> [hdfs:sourcepath …] <hdfs:targetPath>
```

mv 命令可以在 HDFS 中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。

示例如下：

（1）将 HDFS 中的/input/major 文件移动到/file/file1/file2 中。

```
[hadoop@Slave001 ~]$ hadoop fs -mv /input/major /file/file1/file2
[hadoop@Slave001 ~]$ hadoop fs -ls /file/file1/file2
Found 2 items
drwxr-xr-x   - hadoop supergroup   0 2017-12-26 23:35 /file/file1/file2/file3
-rw-r--r--   3 hadoop supergroup   482 2017-12-26 00:19 /file/file1/file2/major
```

（2）使用 mv 命令将 HDFS 中的/input/score 文件和/input/word 文件移动到/file/file1/file2/file3 目录中。

```
[hadoop@Slave001 ~]$ hadoop fs -mv /input/score /input/word /file/file1/file2/file3
[hadoop@Slave001 ~]$ hadoop fs -ls /file/file1/file2/file3
Found 2 items
-rw-r--r--   3 hadoop supergroup   56812 2017-12-25 23:40 /file/file1/file2/file3/
score
-rw-r--r--   3 hadoop supergroup   8829 2017-12-25 22:15 /file/file1/file2/
file3/word
[hadoop@Slave001 ~]$ hadoop fs -ls /input
Found 7 items
-rw-r--r--   3 hadoop supergroup    14543 2017-12-25 04:52 /input/city_data
…
drwxr-xr-x   - hadoop supergroup     0 2017-12-26 23:32 /input/file2
```

14. put

语法格式如下：

```
hadoop fs -put <local:pathFile> [local:pathFile] <hdfs:pathDirctory>
```

put 命令可以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。HDFS 中接收文件的目录必须事先存在。

示例：从本地上传 city_data 文件和 monitor_data 文件到 HDFS 的 test 目录中。

```
[hadoop@Slave001 ~]$ ls
aaaip9.jar   city_data  file   monitor_data  rrwquy.jar
[hadoop@Slave001 ~]$ hadoop fs -mkdir /test
[hadoop@Slave001 ~]$ hadoop fs -put city_datamonitor_data /test
[hadoop@Slave001 ~]$ hadoop fs -ls /test
Found 2 items
-rw-r--r--   3 hadoop supergroup   145430 2017-12-26 23:48 /test/city_data
-rw-r--r--   3 hadoop supergroup   2954128 2017-12-26 23:48 /test/monitor_data
```

15. rm

语法格式如下：

```
hadoop fs -rm <hdfs:pathFile> [hdfs:pathFile]
```

rm 命令是用于删除一个指定的文件或多个指定文件的命令，并且加上-r 参数可以删除指定目录。

示例如下：

```
[hadoop@Slave001 ~]$ hadoop fs -ls /test
Found 2 items
-rw-r--r--   3 hadoop supergroup   145430 2017-12-26 23:48 /test/city_data
-rw-r--r--   3 hadoop supergroup   2954128 2017-12-26 23:48 /test/monitor_data
[hadoop@Slave001 ~]$ hadoop fs -rm /test/city_data /test/monitor_data
17/12/26 23:52:19 INFO fs.TrashPolicyDefault: NameNode trash configuration: Deletion
interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /test/city_data
17/12/26 23:52:20 INFO fs.TrashPolicyDefault: NameNode trash configuration: Deletion
interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /test/monitor_data
```

16. rmr

语法格式如下：

```
hadoop fs -rmr <hdfs:pathDirectory> [hdfs:pathDirectory]
```

rmr 命令可以删除目录或递归删除子文件。如果使用-rmr 命令删除一个目录，不管

目录下是否有其他文件，均被一并删除。

示例如下：

```
[hadoop@Slave001 ~]$ hadoop fs -ls -R /test /file
drwxr-xr-x   - hadoop supergroup  0 2017-12-26 23:35 /file/file1
drwxr-xr-x   - hadoop supergroup  0 2017-12-26 23:40 /file/file1/file2
...
-rw-r--r--   3 hadoop supergroup  482 2017-12-26 00:19 /file/file1/file2/major
-rw-r--r--   3 hadoop supergroup  36365 2017-12-20 21:51 /file/input
[hadoop@Slave001 ~]$ hadoop fs -rmr /test /file
rmr: DEPRECATED: Please use 'rm -r' instead.
17/12/26 23:55:49 INFO fs.TrashPolicyDefault: NameNode trash configuration: Deletion
interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /test
17/12/26 23:55:49 INFO fs.TrashPolicyDefault: NameNode trash configuration: Deletion
interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /file
[hadoop@Slave001 ~]$ hadoop fs -ls -R /test /file
ls: `/test': No such file or directory
ls: `/file': No such file or directory
```

17. tail

语法格式如下：

```
hadoop fs -tail [-f] <hdfs:pathFile>
```

tail 命令可以将文件尾部 1KB 的内容输出到标准输出。并且 tail 命令支持-f 选项，加上-f 选项表示实时显示文件内容。

示例如下：

```
[hadoop@Slave001 ~]$ hadoop fs -tail /input/city_data
90100751295   PMS_淳溪长一村 4#   淳溪供电所   南京
90100751318   PMS_新杨 4#(加工厂边)   淳溪供电所   南京
......
90100796714   PMS_固城湖佳苑#1   南京市高淳区供电公司   南京
90100796715   PMS_固城湖佳苑#2   南京市高淳区供电公司   南京
```

18. text

语法格式如下：

```
hadoop fs -text <hdfs:pathFile>
```

text 命令可以将 HDFS 中的源文件以文本格式输出。

19. touchz

语法格式如下：

```
hadoop fs -touchz <hdfs:newFile>
```

touchz 命令可以在 HDFS 中创建一个 0B 的空文件。

示例如下：

```
[hadoop@Slave001 ~]$ hadoop fs -touchz /newfile
[hadoop@Slave001 ~]$ hadoop fs -ls /newfile
-rw-r--r--   3 hadoop supergroup    0 2017-12-27 00:01 /newfile
```

5.3 API 的使用

HDFS 是一个分布式文件系统。既然是文件系统，就可以对其文件进行操作。例如，新建文件、删除文件、读取文件内容等。Python 操作 HDFS 常用的模块有 hdfs 和 pyhdfs 两种，下面将分别讲解如何使用 Python 模块对 HDFS 中的文件进行操作。

5.3.1 hdfs 模块

关于 hdfs 模块的讲解视频可扫描二维码观看。

hdfs 模块是 Python 提供的第三方库模块，它提供了直接对 Hadoop 中 HDFS 操作的能力，hdfs 模块是 HDFS 的 API 和命令行接口。

1. 安装 hdfs 模块

在使用 hdfs 模块前需要安装 hdfs 模块，在 Python 中所有的第三方模块均采用 pip 命令安装。

在 Windows 下使用 pip 命令安装 hdfs 模块有以下两种方式。

(1) 命令行方式安装：在 Windows 任务栏的搜索文本框中输入“运行”后按 Enter 键，弹出“运行”对话框，在对话框中输入 cmd 后按 Enter 键，进入管理员终端交互界面，如图 5.2 所示。

图 5.2 命令行方式安装 hdfs 模块

(2) PyCharm 方式安装：在 Terminal 窗口中输入 pip install hdfs 命令，如图 5.3 所示。

图 5.3　PyCharm 方式安装 hdfs 模块

验证 hdfs 模块安装是否成功：在控制台终端输入 pip list，如果查看到安装的 hdfs 模块则说明安装成功，如图 5.4 所示。

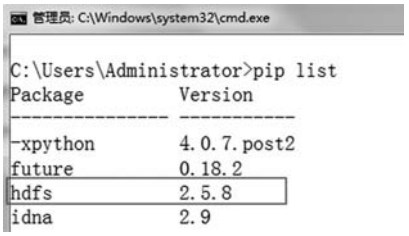

图 5.4　查看安装结果

2. 连接 HDFS

hdfs 模块中的 Client 类非常关键，使用这个类可以实现连接 HDFS 的 NameNode，对 HDFS 上的文件进行查、读、写等操作。

参数解析如下。

url：主机名或 IP 地址，后跟 NameNode 的端口号，例如，url＝"http://192.168.153.101:50070"。还可以指定多个以逗号分隔的 URL 连接高可用集群，例如 url＝["http://192.168.153.101:50070"," http://192.168.153.102:50070"]。

proxy：指定代理，默认为 None。

root：指定根路径，其将作为传递给客户端所有 HDFS 的前缀。

timeout：设置连接超时，已转发到请求处理程序，如果达到超时，则将引发适当的异常。

session：设置发出所有请求的实例。

连接 HDFS 实例如下：

```
class HDFSTest():
    def __init__(self):
        self.client = Client(url = "http://192.168.153.101:50070")
        print("返回操作 HDFS 对象: ", self.client)
```

3. status()函数

status()函数用于查看文件或者目录的状态，有两个接收参数：hdfs_path 参数是要查看的 HDFS 路径；strict 参数用于是否开启严格模式，严格模式下目录或文件不存在则

返回 raise，否则返回 None。

status()函数示例如下：

```
class HDFSTest():
    def __init__(self):
        self.client = Client(url = "http://192.168.153.101:50070")

    def status(self):
        c = self.client
        print(c.status(hdfs_path = "/input", strict = False))
if __name__ == '__main__':
    h = HDFSTest()
    h.status()
```

输出结果说明如下。

如果 strict＝False，/input 文件不存在，则返回结果为 None。

如果 strict＝True，/input 文件不存在，则返回结果为：

```
hdfs.util.HdfsError: File does not exist: /input
```

如果/input 文件存在，则返回结果为：

```
{'accessTime': 0, 'blockSize': 0, 'childrenNum': 0, 'fileId': 16403, 'group': 'supergroup',
'length': 0, 'modificationTime': 1594872452916, 'owner': 'hadoop', 'pathSuffix': '',
'permission': '755', 'replication': 0, 'storagePolicy': 0, 'type': 'DIRECTORY'}
```

4. content()函数

content()函数用于列出目录或文件详情，有两个接收参数：hdfs_path 参数是要查看的 HDFS 路径；strict 参数用于是否开启严格模式，严格模式下目录或文件不存在则返回 raise，否则返回 None。

content()函数示例如下：

```
class HDFSTest():
    def __init__(self):
        self.client = Client(url = "http://192.168.153.101:50070")
    def content(self):
        c = self.client
        print(c.content(hdfs_path = "/", strict = True))
        print(c.content(hdfs_path = "/input", strict = True))
        print(c.content(hdfs_path = "/output", strict = True))
if __name__ == '__main__':
    h = HDFSTest()
    h. content()
```

结果如图 5.5 所示。

图 5.5　content()函数示例结果

5. list()函数

list()函数用于列出指定目录下的所有文件，有两个接收参数：hdfs_path 参数是要查看的 HDFS 路径；status 参数传入布尔类型值，如果值为 True 表示将以元组方式返回当前目录下的所有文件的文件名和文件的状态，如果值为 False 表示只查看指定目录下的所有文件不返回其状态。

list()函数示例如下：

```
class HDFSTest():
    def __init__(self):
        self.client = Client(url="http://192.168.153.101:50070")
    def list(self):
        c = self.client
        array = c.list(hdfs_path="/", status=True)
        for a in array:
            print(a)
if __name__ == '__main__':
    h = HDFSTest()
    h.list()
```

结果如图 5.6 所示。

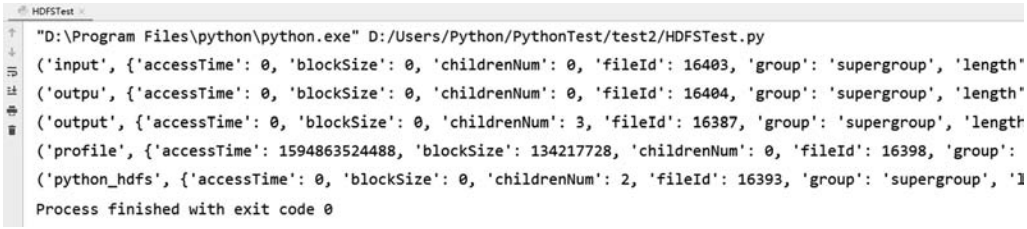

图 5.6　list()函数示例结果

6. 列出指定目录下所有文件及子文件

hdfs 模块中没有提供对指定目录下所有文件及子文件的函数，但可以通过组合函数方式实现该功能。

第一步：使用 status()函数获取指定路径的文件状态，利用字段获取 type 的值，如果值为 DIRECTORY 说明是目录，如果值为 FILE 说明是文件。

第二步：根据获取的文件状态来区别是文件还是目录。

第三步：如果是文件则直接输出文件的路径，如果是目录则递归调用 lists()函数，再次执行第一步的内容。

示例如下：

```
class HDFSTest():
    def __init__(self):
        self.client = Client(url = "http://192.168.153.101:50070")
    # 判断是目录还是文件
    def dirOrFile(self, path):
        c = self.client
        dict = c.status(hdfs_path = path, strict = True)
        type = dict["type"]
        return type

    # 列出指定目录下的所有文件
    def lists(self, path):
        c = self.client
        array = c.list(hdfs_path = path, status = False)
        for a in array:
            if path == "/":
                b = path + a
                type = self.dirOrFile(b)
                if type == "FILE":
                    print("文件: ", b)
                else:
                    print("目录: ", b)
                    self.lists(b)
            else:
                b = path + "/" + a
                type = self.dirOrFile(b)
                if type == "FILE":
                    print("文件: ", b)
                else:
                    print("目录: ", b)
                    self.lists(b)
if __name__ == '__main__':
    h = HDFSTest()
    h.lists("/input")
```

7. makedirs()函数

makedirs()函数用于在 HDFS 中远程创建目录，支持目录递归创建。有两个接收参数：hdfs_path 参数是要创建的 hdfs 路径；permission 参数用于对新创建的目录设置权限。

makedirs()函数示例如下：

```
class HDFSTest():
    def __init__(self):
        self.client = Client(url="http://192.168.153.101:50070")
    # 在HDFS中创建新目录
    def mkdir(self, path, per):
        c = self.client
        c.makedirs(hdfs_path=path, permission=per)
if __name__ == '__main__':
    h = HDFSTest()
    h.mkdir("/tmp", 777)
```

结果如图 5.7 所示。

```
[hadoop@Master001 ~]$ hadoop fs -ls -R /
20/07/17 05:04:32 WARN util.NativeCodeLoader: Unable to load native-hadoop library
a classes where applicable
drwxr-xr-x   - hadoop supergroup          0 2020-07-16 04:46 /output
drwxrwxrwx   - hadoop supergroup          0 2020-07-17 05:04 /python_hdfs
-rw-r--r--   3 hadoop supergroup        871 2020-07-16 06:07 /python_hdfs/date
-rwxrwxrwx   2 hadoop supergroup       2398 2020-07-16 05:59 /python_hdfs/profile
drwxrwxrwx   - dr.who supergroup          0 2020-07-17 05:04 /python_hdfs/tmp
```

图 5.7　makedirs()函数示例结果

注意，当使用 PyCharm 远程创建目录时会出现 hdfs.util.HdfsError：Permission denied：user＝dr.who，access＝WRITE，inode＝"/"：hadoop：supergroup：drwxr-xr-x 异常，原因是使用的其他用户没有控制 HDFS 目录的权限。解决方案有多种，这里介绍一种最安全的方式，即通过集群管理员为远程操作用户创建一个独立并有权限访问的目录，之后的远程访问均在该目录下进行，具体操作步骤如下。

第一步：创建目录。

```
[root@Master001 ~]# hadoop fs -mkdir /test
[root@Master001 ~]# hadoop fs -ls /
drwxr-xr-x   - hadoop supergroup          0 2020-07-17 05:13 /test
```

第二步：设置权限。

```
[root@Master001 ~]# hadoop fs -chmod 777 /test
[root@Master001 ~]# hadoop fs -ls /
drwxrwxrwx   - hadoop supergroup          0 2020-07-17 05:13 /test
```

第三步：在 PyCharm 指定目录下创建目录。

```
c = self.client
c.makedirs(hdfs_path="/test/tmp", permission=777)
[root@Master001 ~]# hadoop fs -ls -R /
drwxrwxrwx   - dr.who supergroup          0 2020-07-17 05:16 /test/tmp
```

8. rename()函数

rename()函数为文件或目录重命名，接收两个参数：hdfs_src_path 参数为原始路径

或名称；hdfs_dst_path 参数为修改后的文件或路径。rename()函数将 hdfs_dst_path 参数设置为与原始路径不同的路径可以实现文件的移动效果。

rename()函数示例如下：

```
class HDFSTest():
    def __init__(self):
        self.client = Client(url = "http://192.168.153.101:50070")
    # 重命名
    def rename(self, oldName, newName):
        c = self.client
        c.rename(oldName, newName)
if __name__ == '__main__':
    h = HDFSTest()
    h.rename("/test/tmp", "/test/tmp2")
```

输出结果：

```
[root@Master001 ~]# hadoop fs -ls /test
drwxrwxrwx   - dr.who supergroup          0 2020-07-17 05:16 /test/tmp
[root@Master001 ~]# hadoop fs -ls /test
drwxrwxrwx   - dr.who supergroup          0 2020-07-17 05:16 /test/tmp2
```

9. resolve()函数

resolve()函数返回指定路径的绝对路径，返回值为 str 类型，接收一个参数：hdfs_path 参数是指定的 HDFS 路径。

resolve()函数示例如下：

```
class HDFSTest():
    def __init__(self):
        self.client = Client(url = "http://192.168.153.101:50070")
    # 返回绝对路径
    def resolve(self):
        c = self.client
        print(c.resolve("/test/tmp2"))
if __name__ == '__main__':
    h = HDFSTest()
    h.resolve()
```

结果如图 5.8 所示。

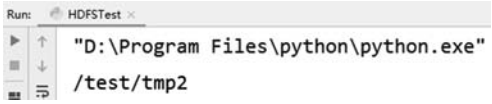

图 5.8 resolve()函数示例结果

10. set_replication()函数

set_replication()函数用于设置文件在 HDFS 上的副本数量，Hadoop 集群模式下的副本默认保存 3 份，接收两个参数：hdfs_path 参数是 HDFS 中文件的路径；replication 参数是文件的副本数量。

set_replication()函数示例如下：

```
class HDFSTest():
    def __init__(self):
        self.client = Client(url = "http://192.168.153.101:50070")
    # 设置文件在 HDFS 上的副本数量
    def set_replication(self, hdfs_path, replication):
        c = self.client
        c.set_replication(hdfs_path, replication)
if __name__ == '__main__':
    h = HDFSTest()
    h.set_replication("/test/profile_2", 2)
```

结果如图 5.9 所示。

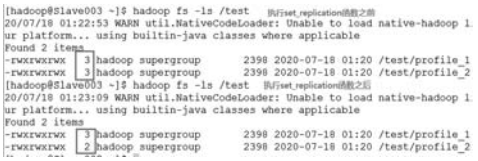

图 5.9 set_replication()函数示例结果

11. read()函数

read()函数用于读取文件的信息，与 hadoop fs -cat hdfs_path 类似。参数如下。

hdfs_path：HDFS 路径。

offset：读取位置。

length：读取长度。

encoding：指定编码。

chunk_size：字节的生成器，必须和 encoding 一起使用满足 chunk_size 设置，即 yield。

delimiter：设置分隔符，必须和 encoding 一起设置。

progress：读取进度回调函数，读取一个 chunk_size 回调一次。

read()函数示例 1：以下例子只能读取英语文件内容，中文内容将以十六进制方式呈现。

```
class HDFSTest():
    def __init__(self):
```

```
        self.client = Client(url="http://192.168.153.101:50070")
    # 读取文件信息
    def read1(self, path):
        c = self.client
        with c.read(hdfs_path=path) as obj:
            for i in obj:
                print(i)
if __name__ == '__main__':
    h = HDFSTest()
    h.read1("/test/data")
```

结果如图 5.10 所示。

```
HDFSTest
"D:\Program Files\python\python.exe" D:/Users/Python/PythonTest/test2/HDFSTest.py
b'\xe6\xb5\x8b\xe8\xaf\x95\xe6\x95\xb0\xe6\x8d\xae\n'   中文十六进制输出
b'test data\n'
b'\n'
b'\xe4\xb9\x9d\xe5\xb0\x9a\xe7\xa7\x91\xe6\x8a\x80\xe6\xac\xa2\xe8\xbf\x8e\xe6\x82\xa8\n'
b'Welcome to JiuShang Technology\n'
```

图 5.10 read()函数示例 1 结果

read()函数示例 2：读取指定文件的内容，并设置编码级。

```
class HDFSTest():
    def __init__(self):
        self.client = Client(url="http://192.168.153.101:50070")
    # 读取文件信息,并设置编码级
    def read2(self, path):
        c = self.client
        with c.read(hdfs_path=path, encoding="utf8") as obj:
            for i in obj:
                print(i)
if __name__ == '__main__':
    h = HDFSTest()
    h.read2("/test/data")
```

结果如图 5.11 所示。

```
Run: HDFSTest
"D:\Program Files\python\python.exe" D:/Users/Python/PythonTest/test2/HDFSTest.py
测试数据
test data
九尚科技欢迎您
Welcome to JiuShang Technology
```

图 5.11 read()函数示例 2 结果

read()函数示例 3：从头到尾读取长度为 6B 的数据，如果 length 为 None 将读取整个文件。

```
class HDFSTest():
    def __init__(self):
        self.client = Client(url = "http://192.168.153.101:50070")
    def read3(self, path):
        c = self.client
        with c.read(hdfs_path = path, encoding = "utf8", length = 8) as obj:
            for i in obj:
                print(i)
if __name__ == '__main__':
    h = HDFSTest()
    h.read3("/test/data")
```

结果如图 5.12 所示。

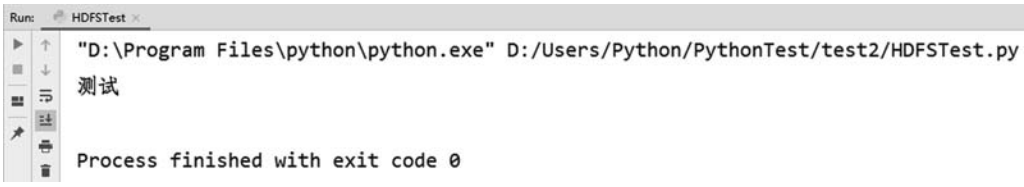

图 5.12　read()函数示例 3 结果

read()函数示例 4：从 length 为 6B 位置开始，读取 length 为 20B 长度的数据。

```
class HDFSTest():
    def __init__(self):
        self.client = Client(url = "http://192.168.153.101:50070")
    # 从 6 位置读取 20 长度数据
    def read4(self, path):
        c = self.client
        with c.read(hdfs_path = path, encoding = "utf8", offset = 6, length = 20) as obj:
            for i in obj:
                print(i, end = "")
if __name__ == '__main__':
    h = HDFSTest()
    h.read4("/test/data")
```

结果如图 5.13 所示。

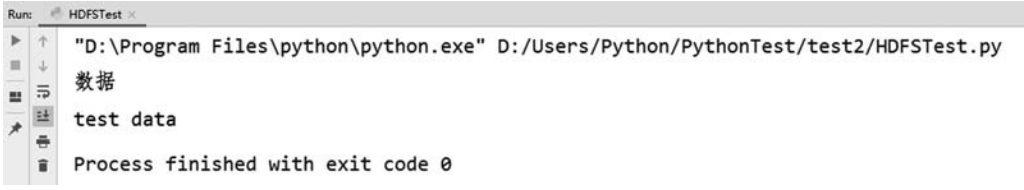

图 5.13　read()函数示例 4 结果

read()函数示例5：设置按指定分隔符分隔读取内容，默认分隔符为\n。

```
class HDFSTest():
    def __init__(self):
        self.client = Client(url="http://192.168.153.101:50070")
    def read5(self, path):
        c = self.client
        p = c.read(hdfs_path=path, encoding="utf8", delimiter=" ")
        with p as d:
            for i in d:
                print(i)
if __name__ == '__main__':
    h = HDFSTest()
    h.read5("/test/data")
```

结果如图5.14所示。

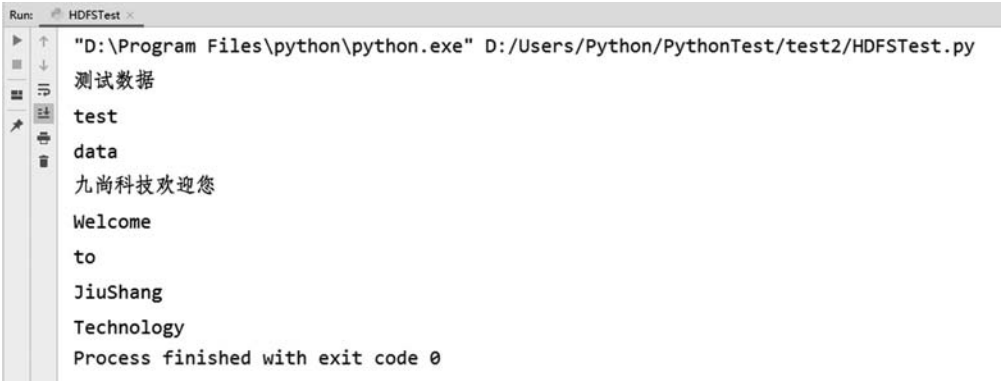

图5.14 read()函数示例5结果

12. download()函数

download()函数从HDFS中下载文件到本地。参数如下。

hdfs_path：HDFS路径。

local_path：下载的本地路径。

overwrite：是否覆盖，默认为False。

n_threads：启动线程数量，默认为1，不启用多线程。

temp_dir：下载过程中文件的临时路径。

download()函数示例：在PyCharm中下载数据到Windows本地。

```
class HDFSTest():
    def __init__(self):
        self.client = Client(url="http://192.168.153.101:50070")
    def download(self, hdfs, local):
        c = self.client
```

```
        c.download(hdfs_path=hdfs, local_path=local)
if __name__ == '__main__':
    h = HDFSTest()
    h.download("/test/data", "D:/tmp")
```

结果如图 5.15 所示。

图 5.15　download()函数示例结果

13. upload()函数

upload()函数从本地上传文件到 HDFS，等同于 hadoop fs -copyFromLocal local_file hdfs_path。参数如下。

hdfs_path：HDFS 路径。

local_path：本地文件路径。

n_threads：并行线程的数量，默认为 1。

temp_dir：文件已经存在的情况下的临时路径。

chunk_size：设置块大小，默认值为 2 ** 16。

progress：报告进度的回调函数，默认值为 None。

cleanup：上传错误时是否删除已经上传的文件，默认值为 True。

upload()函数示例如下：

```
class HDFSTest():
    def __init__(self):
        self.client = Client(url="http://192.168.153.101:50070")
    def upload(self, hdfs, local):
        c = self.client
        c.upload(hdfs_path=hdfs, local_path=local)
if __name__ == '__main__':
    h = HDFSTest()
    h.upload("/test/new_test.txt","D:/tmp/test.txt")
```

结果如图 5.16 所示。

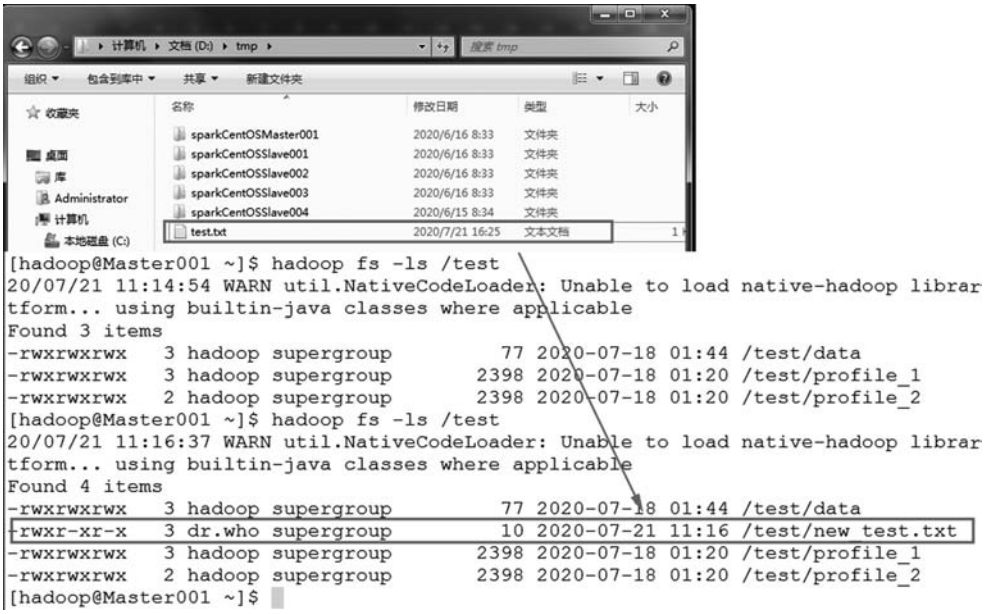

图 5.16 upload()函数示例结果

14. set_permission()函数

set_permission()函数用于修改权限，与 hadoop fs -chmod 777 hdfs_path 类似，接收两个参数：hdfs_path 参数为 HDFS 中要被修改的文件路径；permission 参数为文件被修改后的权限。

set_permission()函数示例如下：

```
class HDFSTest():
    def __init__(self):
        self.client = Client(url = "http://192.168.153.101:50070")
    def set_permission(self, path, p):
        c = self.client
        c.set_permission(hdfs_path = path, permission = p)
if __name__ == '__main__':
    h = HDFSTest()
    h.set_permission("/test/new_test.txt", 777)
```

结果如图 5.17 所示。

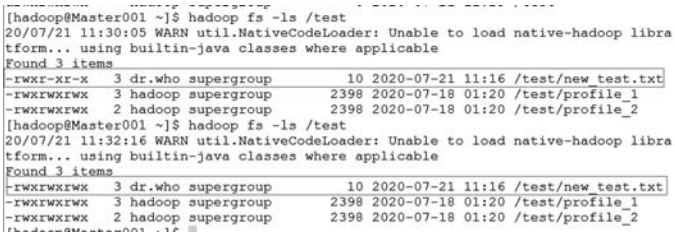

图 5.17 set_permission()函数示例结果

15. delete()函数

delete()函数用于删除指定的文件及目录，接收3个参数。

hdfs_path：指定需要删除的文件及目录的路径。

recursive：是否递归删除指定的文件及目录，默认值为False。

ship_trash：是否将删除的文件移到回收站，而不是直接删除，默认值为True。

delete()函数示例如下：

```
class HDFSTest():
    def __init__(self):
        self.client = Client(url="http://192.168.153.101:50070")
    def delete(self, path, r):
        c = self.client
        c.delete(hdfs_path=path, recursive=r)
if __name__ == '__main__':
    h = HDFSTest()
    h.delete("/test/tmp", True)
```

结果如图5.18所示。

```
[hadoop@Master001 ~]$ hadoop fs -ls -R /test
20/07/21 12:21:11 WARN util.NativeCodeLoader: Unable to load native-hadoop library
tform... using builtin-java classes where applicable
-rwxrwxrwx   3 dr.who supergroup         10 2020-07-21 11:16 /test/new_test.txt
drwxrwxrwx   - hadoop supergroup          0 2020-07-21 12:17 /test/tmp
-rwxrwxrwx   3 hadoop supergroup       2398 2020-07-18 01:20 /test/tmp/profile_1
-rwxrwxrwx   2 hadoop supergroup       2398 2020-07-18 01:20 /test/tmp/profile_2
[hadoop@Master001 ~]$ hadoop fs -ls -R /test
20/07/21 12:21:33 WARN util.NativeCodeLoader: Unable to load native-hadoop library
tform... using builtin-java classes where applicable
-rwxrwxrwx   3 dr.who supergroup         10 2020-07-21 11:16 /test/new_test.txt
[hadoop@Master001 ~]$
```

图5.18　delete()函数示例结果

5.3.2　pyhdfs模块

关于pyhdfs模块的讲解视频可扫描二维码观看。

pyhdfs模块是Python提供的第三方库模块，它提供了直接对Hadoop中HDFS操作的能力，pyhdfs模块是HDFS的API和命令行接口。

1. 安装pyhdfs模块

使用pyhdfs模块前需要安装pyhdfs模块，在Python中所有的第三方模块均采用pip安装。

在Windows下使用pip命令安装pyhdfs模块有以下两种方式。

(1) 命令行方式安装：运行→cmd→pip install pyhdfs，如图5.19所示。

(2) PyCharm方式安装：在Terminal界面输入pip install pyhdfs命令，如图5.20所示。

验证pyhdfs模块是否安装成功：在控制台终端输入pip list，如果查看到安装的pyhdfs模块则说明安装成功，如图5.21所示。

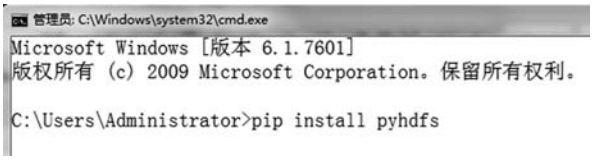

图 5.19 命令行方式安装 pyhdfs 模块

图 5.20 PyCharm 方式安装 pyhdfs 模块

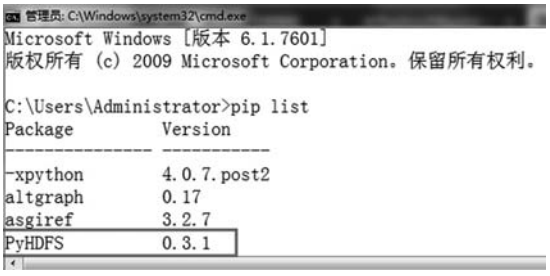

图 5.21 查看安装结果

2. 连接 HDFS

pyhdfs 模块中的 HdfsClient 类非常关键。使用这个类可以实现连接 HDFS 的 NameNode，对 HDFS 上的文件进行查、读、写等操作。

参数解析如下。

hosts：主机名或 IP 地址，与 port 之间需要用逗号隔开，如 hosts="192.168.153.101:9000"，支持高可用集群，例如，["192.168.153.101,9000","192.168.153.102,9000"]。

randomize_hosts：随机选择 host 进行连接，默认为 True。

user_name：连接的 Hadoop 平台的用户名。

timeout：每个 NameNode 节点连接等待的秒数，默认为 20s。

max_tries：每个 NameNode 节点尝试连接的次数，默认为 2。

retry_delay：在尝试连接一个 NameNode 节点失败后，尝试连接下一个 NameNode 的时间间隔，默认为 5s。

requests_session：设置连接 HDFS 的 HTTPRequest 请求的模式为 session。

代码示例如下：

```
import pyhdfs
class HDFSTest2:
    def __init__(self):
        self.client = pyhdfs.HdfsClient(hosts = "192.168.153.101,50070", user_name =
"hadoop")
    def test(self):
        print(self.client)
if __name__ == '__main__':
    h = HDFSTest2()
    h.test()
```

结果如图 5.22 所示。

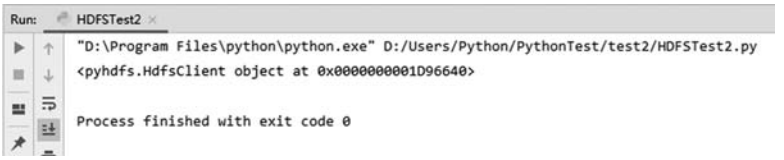

图 5.22　输出结果

注意，在 Windows 下使用 PyCharm 中 pyhdfs 模块连接 HDFS 时，需要设置 Windows 中的 hosts 文件，否则将无法在连接集群中根据节点名字找到从节点，如图 5.23 所示。

图 5.23　设置 hosts 文件

3. get_home_directory()函数

get_home_directory()函数用于返回所连接集群的根目录。

get_home_directory()函数示例如下：

```
class HDFSTest2:
    # 获取对 HDFS 操作的对象
    def __init__(self):
        self.client = pyhdfs.HdfsClient(hosts = "192.168.153.101, 50070", user_name =
"hadoop")
    # 返回这个用户的根目录
    def get_home_directory(self):
```

```
        c = self.client
        print(c.get_home_directory())
if __name__ == '__main__':
    h = HDFSTest2()
    h.get_home_directory()
```

结果如图 5.24 所示。

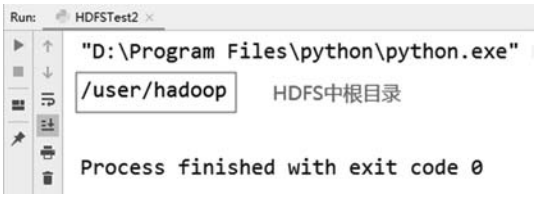

图 5.24 get_home_directory()函数示例结果

4. get_active_namenode()函数

get_active_namenode()函数用于返回当前活动的 NameNode 的地址。

get_active_namenode()函数示例如下：

```
class HDFSTest2:
    # 获取对 HDFS 操作的对象
    def __init__(self):
        self.client = pyhdfs.HdfsClient(hosts = "192.168.153.101,50070", user_name =
"hadoop")
    # 返回可用的 NameNode 节点
    def get_active_namenode(self):
        c = self.client
        nameNode = c.get_active_namenode()
        print(nameNode)
if __name__ == '__main__':
    h = HDFSTest2()
    h.get_active_namenode()
```

结果如图 5.25 所示。

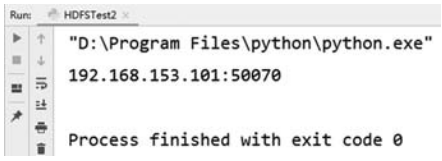

图 5.25 get_active_namenode()函数示例结果

5. listdir()函数

listdir()函数用于返回指定目录下的所有文件，由于 pyhdfs 可以设置访问的用户，因

此在操作 HDFS 中的文件时不需要设置其他用户的权限。listdir()函数中 path 参数是指定的 HDFS 路径。

listdir()函数示例如下：

```
class HDFSTest2:
    # 获取对 HDFS 操作的对象
    def __init__(self):
        self.client = pyhdfs.HdfsClient(hosts = "192.168.153.101,50070", user_name =
"hadoop")

    # 查询 HDFS 中根目录下的所有文件
    def listdir(self, hdfsPath):
        c = self.client
        dir = c.listdir(path = hdfsPath)
        for d in dir:
            print(d, end = "\t")
if __name__ == '__main__':
    h = HDFSTest2()
    h.listdir("/")
```

结果如图 5.26 所示。

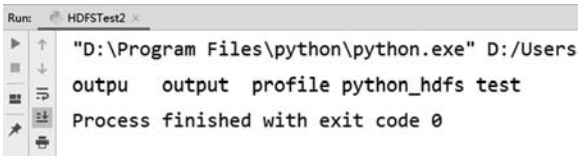

图 5.26 listdir()函数示例结果

6. open()函数

open()函数用于远程打开 HDFS 中的文件，返回 IO[bytes]类型，利用 read()函数读取指定文件的数据，返回 AnyStr 类型。

open()函数示例如下：

```
class HDFSTest2:
    # 获取对 HDFS 操作的对象
    def __init__(self):
        self.client = pyhdfs.HdfsClient(hosts = "192.168.153.101,50070", user_name =
"hadoop")
    # 打开 hdfs 中文件
    def open(self, filDir):
        c = self.client
        file = c.open(path = filDir)
        print(file.read().decode(encoding = "utf8"))
if __name__ == '__main__':
    h = HDFSTest2()
    h.open("/input/data")
```

结果如图 5.27 所示。

图 5.27 open()函数示例结果

7. copy_from_local()函数

copy_from_local()函数用于从本地上传文件到集群，接收两个参数：localsrc 参数用于设置本地文件路径；dest 参数用于设置 HDFS 中文件路径，如果 dest 参数对应的路径不存在则创建一个新路径。

copy_from_local()函数示例如下：

```
class HDFSTest2:
    # 获取对 HDFS 操作的对象
    def __init__(self):
        self.client = pyhdfs.HdfsClient(hosts = "192.168.153.101,50070", user_name =
"hadoop")
    # 从本地上传文件至集群
    def copy_from_local(self, local, hdfsPath):
        c = self.client
        c.copy_from_local(localsrc = local, dest = hdfsPath)
if __name__ == '__main__':
    h = HDFSTest2()
    h.copy_from_local("D:/tmp/test.txt", "/input/dd/newTest")
```

输出结果：

```
[hadoop@Slave003 ~]$ hadoop fs -ls /input
-rw-r--r--   3 hadoop supergroup         77 2020-07-23 04:20 /input/data
[hadoop@Slave003 ~]$ hadoop fs -ls /input
-rw-r--r--   3 hadoop supergroup         77 2020-07-23 04:20 /input/data
-rwxr-xr-x   3 hadoop supergroup         10 2020-07-24 06:12 /input/newTest
[hadoop@Slave003 ~]$ hadoop fs -ls -R /input
-rw-r--r--   3 hadoop supergroup         77 2020-07-23 04:20 /input/data
drwxr-xr-x   - hadoop supergroup          0 2020-07-24 06:14 /input/dd
-rwxr-xr-x   3 hadoop supergroup         10 2020-07-24 06:14 /input/dd/newTest
-rwxr-xr-x   3 hadoop supergroup         10 2020-07-24 06:12 /input/newTest
```

8. copy_to_local()函数

copy_to_local()函数用于从集群的 HDFS 中下载文件到本地，接收两个参数：src 参

数为 hdfs 中的文件路径；localdest 参数为本地的文件存储路径。

copy_to_local()函数示例如下：

```
class HDFSTest2:
    # 获取对 HDFS 操作的对象
    def __init__(self):
        self.client = pyhdfs.HdfsClient(hosts = "192.168.153.101,50070", user_name =
"hadoop")
    # 从集群下载文件到本地
    def copy_to_local(self, hdfsPath, local):
        c = self.client
        c.copy_to_local(src = hdfsPath, localdest = local)
if __name__ == '__main__':
    h = HDFSTest2()
    h.copy_to_local("/input/data", "D:/tmp")
```

9. mkdirs()函数

mkdirs()函数用于在集群的 HDFS 中创建新目录，path 参数用于传入需要创建的路径。

mkdirs()函数示例如下：

```
class HDFSTest2:
    # 获取对 HDFS 操作的对象
    def __init__(self):
        self.client = pyhdfs.HdfsClient(hosts = "192.168.153.101,50070", user_name =
"hadoop")
    # 创建新目录
    def mkdirs(self, hdfsPath):
        c = self.client
        c.mkdirs(path = hdfsPath)
if __name__ == '__main__':
    h = HDFSTest2()
    h.mkdirs("/input/tmp")
```

结果如图 5.28 所示。

```
[hadoop@Slave003 ~]$ hadoop fs -ls -R /input
20/07/24 06:46:54 WARN util.NativeCodeLoader: Unable to load native-hadoop library
iltin-java classes where applicable
-rw-r--r--   3 hadoop supergroup         77 2020-07-23 04:20 /input/data
drwxr-xr-x   - hadoop supergroup          0 2020-07-24 06:14 /input/dd
-rwxr-xr-x   3 hadoop supergroup         10 2020-07-24 06:14 /input/dd/newTest
-rwxr-xr-x   3 hadoop supergroup         10 2020-07-24 06:12 /input/newTest
drwxr-xr-x   - hadoop supergroup          0 2020-07-24 06:45 /input/tmp
[hadoop@Slave003 ~]$
```

图 5.28　mkdirs()函数示例结果

10. exists()函数

exists()函数用于查看指定的文件或目录是否存在，如果存在则返回 True，否则返回 False。

exists()函数示例如下：

```
class HDFSTest2:
    # 获取对 HDFS 操作的对象
    def __init__(self):
        self.client = pyhdfs.HdfsClient(hosts = "192.168.153.101,50070", user_name = "hadoop")
    # 查看文件是否存在
    def exists(self, hdfsPath):
        c = self.client
        result = c.exists(path = hdfsPath)
        print("结果: ", result)
if __name__ == '__main__':
    h = HDFSTest2()
    h.exists("/input")
```

结果如图 5.29 所示。

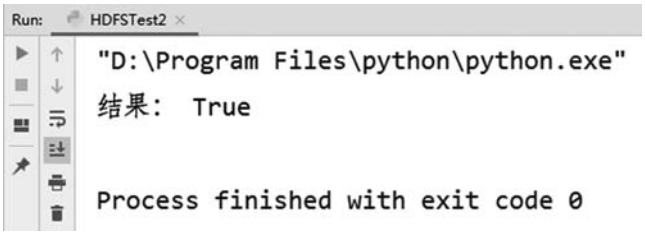

图 5.29 exists()函数示例结果

11. get_file_status()函数

get_file_status()函数用于返回指定 HDFS 路径的路径对象，path 参数为 HDFS 文件或目录路径。

get_file_status()函数示例如下：

```
class HDFSTest2:
    # 获取对 HDFS 操作的对象
    def __init__(self):
        self.client = pyhdfs.HdfsClient(hosts = "192.168.153.101,50070", user_name = "hadoop")
    def get_file_status(self, hdfsPath):
        c = self.client
        status = c.get_file_status(hdfsPath)
```

```
            print(status["type"])
            if status["type"] == "DIRECTORY":
                print(f"{hdfsPath}: 该文件是目录!")
            elif status["type"] == "FILE":
                print(f"{hdfsPath}: 该文件是文件!")
if __name__ == '__main__':
    h = HDFSTest2()
    h.get_file_status("/input/data")
```

结果如图 5.30 所示。

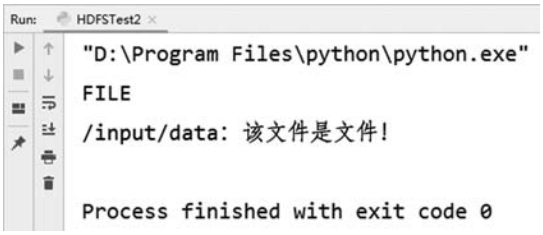

图 5.30　get_file_status()函数示例结果

12. delete()函数

delete()函数用于删除 HDFS 文件，该函数只能删除文件或者空目录，如果删除的目录下有文件的目录，将抛出 HdfsPathIsNotEmptyDirectoryException 异常。

delete()函数示例如下：

```
class HDFSTest2:
    # 获取对 HDFS 操作的对象
    def __init__(self):
        self.client = pyhdfs.HdfsClient(hosts = "192.168.153.101,50070", user_name =
"hadoop")
    def delete(self):
        c = self.client
        c.delete("/input")
if __name__ == '__main__':
    h = HDFSTest2()
    h.delete()
```

5.4　本章小结

HDFS 是 Hadoop 核心之一，是 Hadoop 中的分布式文件系统，用于存储大量数据。首先，本章通过对 HDFS 中基本概念和特点的概述，让读者对 Hadoop 分布式文件系统有基本的认识。其次，对 HDFS 命令的讲解让读者熟悉开发或运维环境下的操作。最后，对 HDFS API 的讲解让读者熟悉生产模式下的开发方式。本章为重点，学好本章内容有

助于 Hadoop 生态圈其他工具的学习。

5.5 课后习题

一、填空题

1. HDFS 的存储单位为块，每个块的默认大小为_____。

2. 进行 HDFS 负载均衡时，不能导致数据块备份_____。

3. 搭建大数据集群时，_____和 core-site. xml 文件是用于对 HDFS 进行设置的文件。

4. 使用 start-dfs. sh 命令启动集群时会启动_____、_____和_____进程。

5. 大数据集群中，数据会自动保存多个副本，通过增加副本的形式提高大数据集群的_____。

二、判断题

1. Hadoop 是一个由 Apache 基金会所开发的分布式系统基础架构。 ( )

2. Hadoop 的框架的两个核心是 HDFS 和 MapReduce。 ( )

3. HDFS 是分布式文件系统，用于存储数据。 ( )

4. HDFS 是分布式计算中数据存储管理的基础。 ( )

5、HDFS 以流式数据访问模式来存储超大文件，运行于商用硬件集群上。 ( )

6. HDFS 是以通过移动计算而不是移动数据的方法来做数据计算的框架。 ( )

7. HDFS 的构造思路是，一次写入，多次读取，文件一旦定稿，就不能被修改。 ( )

三、选择题

1. 在搭建 Hadoop 集群环境中，( )文件不属于 HDFS 配置的文件。

A. hadoop-env. sh　　B. core-site. xml

C. hdfs-site. xml　　D. slaves

2. 在大数据集群中设置免密，以下描述正确的是( )。

A. 不设置免密，集群就不能正常运行

B. 由于 Hadoop 集群在运行时 Master 虚拟机需要对 Slave 虚拟机进行监控，如果使用密码登录将大大降低集群效率，因此需要设置免密

C. 设置免密时，需要将 Master 虚拟机生成的密钥文件分发到 Slave 虚拟机

D. 设置免密是为了方便操作集群时可以快速地切换到另一个虚拟机

3. HDFS 是分布式计算中数据存储管理的基础，以下对 HDFS 特点的描述有误的是( )。

A. 高容错性　　B. 适合批处理

C. 适合存放大量小文件　　D. 成本低

4. 以下对 HDFS 特点的描述有误的是( )。

A. 低延时数据访问，不适合实时要求高的场所

B. 对计算机的性能要求比较高

C. 适合超大文件

D. 不能并发写入，不能随机修改

5. 以下对 NaomeNode 进程的描述正确的是(　　)。

A. NameNode 进程节点是用接收 Client 提交的请求，并处理请求

B. NameNode 进程节点是用来存储真实数据的

C. NameNode 进程节点是用来存储元数据和真实数据的

D. 一个集群中可以同时存储多个 NameNode 进程

6. 以下对 SecondaryNameNode 进程的描述正确的是(　　)。

A. SecondaryNameNode 进程是一个辅助进程，它用于定期合并 NameNode 生成的快照文件

B. SecondaryNameNode 进程是一个辅助进程，它可以帮助 NameNode 管理集群

C. SecondaryNameNode 进程没有作用

D. ScondaryNameNode 进程是一个辅助进程，它可以帮助 NamaNode 进程接收请求和处理请求

7. 以下对 DataNode 进程的描述正确的是(　　)。

A. DataNode 进程是用于接收 Client 的请求并处理请求的进程

B. DataNode 进程是数据节点进程，它主要作用是用来存储真实数据

C. DataNode 进程是数据节点进程，它用于存储元数据

D. 在大数据集群中只能有一台虚拟机上有 DataNode 进程

8. (　　)命令不属于 HDFS 的命令。

A. hadoop fs -cat　　B. hadoop fs -ls

C. hadoop fs -put　　D. hadoop fs -tail

四、编程题

1. 利用 HDFS API 编写代码，HDFS 中创建/input/test 目录和/output/test 目录，并将 Word.txt 文件传到 input/test 目录中，把 Word2.txt 文件传到 output/test 目录中，并分区显示 Word.txt 和 Word2.txt 文件，最后删除 input 和 output 目录，要求在一个作业程序中完成。

2. 利用 HDFS API 编写代码，在 HDFS 中实现选择创建目录、创建文件并写入内容、读取指定文件内容、显示指定目录下所有文件、删除目录等功能。

5.6 实训

1. 实训目的

掌握 pyhdfs 模块的应用。

2. 实训任务

利用 HDFS API 编写代码在 HDFS 中创建 input 目录、从本地加载/etc/profile 文件

到 HDFS 中，并验证是否加载成功。

3. 实训步骤

使用 pyhdfs 操作大数据集群中的 HDFS，可以通过 PyCharm 工具远程连接方式操作 HDFS；也可以在大数据集群中创建扩展名为.py 的文件，通过 Python 命令的方式操作 HDFS。前者用于开发环境，便于大数据开发工程师的程序编写，如果需要在生产环境中自动化地操作 HDFS 则需要使用后者。

(1) 在大数据集的任意一台 Slave 虚拟机中创建 hdfs.py 文件，并修改权限为可执行文件。

```
[root@Slave001 ~]# touch hdfs.py

[root@Slave001 ~]# chmod u+x hdfs.py

[root@Slave001 ~]# ll hdfs.py
-rwxr--r--. 1 root root 0 8月  26 22:01 hdfs.py
```

(2) 在 hdfs.py 文件中编写代码。

```
# -*-coding:UTF-8-*-
import pyhdfs


class HDFSTest:
    def __init__(self):
        self.client = pyhdfs.HdfsClient(hosts="192.168.153.111,50070", user_name=
"root")

    # 从本地上传数据到 HDFS 中
    def uploadData(self, local, hdfsPath):
        c = self.client
        c.copy_from_local(localsrc=local, dest=hdfsPath)

    # 验证是否上传成功
    def verify(self, hdfsPath):
        c = self.client
        b = c.exists(path=hdfsPath)
        if b:
            print("文件上传成功")
        else:
            print("文件上下失败")


if __name__ == '__main__':
    h = HDFSTest()
```

```
    # 第一步：设置输入源和目标路径
    inputSrc = "/etc/profile"
    output = "hdfs:///input" + inputSrc
    # 第二步：从本地上传数据到 HDFS 中
    h.uploadData(local = inputSrc, hdfsPath = output)
    # 第三步：验证上传是否成功
    h.verify(output)
```

（3）在大数据集群中运行 hdfs.py 文件。

```
[root@Slave001 ~]# python3 hdfs.py
文件上传成功
```