hadoop使用

Hadoop的shell操作

hdfs文件访问只能通过绝对路径：hdfs://hadoop01:9000/ss

hadoop fs(原生文件系统客户端)

```
hadoop fs -ls /
从本地文件系统到hdfs： hadoop fs -put
从hdfs到本地文件系统： hadoop fs -get
合并下载：     hadoop fs -getmerge
查看文件内容：hadoop fs -cat
强制删除：     hadoop fs -rm -r（递归） -f（强制的）
修改名字：     hadoop fs -mv
复制：          hadoop fs -cp
```

通过Java API方式操作：

1 Hadoop的eclipse连接的配置

1. 解压Hadoop包
2. 配置本地的Hadoop环境
3. 添加Windows兼容性插件
4. 重启eclipse
5. 配置可视化界面

2 hadoop的hdfs的API调用

导入jar包：hdfs包，common包
（1） 工程建立lib包，右键build path
（2） 使用maven管理jar包
（3） 可以手动建立一个jar包依赖库

Hadoop的configuration和filesystem讲解

实例获取：

```
* new对象
* 反射的方式
* 通过工厂类
* 静态方法（单例设计模式）
* 克隆
```

- **FileSystem**: 这个对象是hdfs抽象目录树的一个实例
- **Configuration**：加载配置文件的对象，用于读取配置文件：
  - core-defaule.xml
  - hdfs-default.xml　放置在hadoop_home/share/hadoop/hdfs/hdfs-2.7.6.jar
  - maprred-default.xml
  - yarn-default.xml
- **获取的是Windows本地文件系统**： FileSystem fs=FileSystem.get(conf)
- **获取的是分布式文件系统**： FileSystem fs=FileSystem.get(new URI(“hdfs://hadoop01:9000”), conf)

```
package com.ghgj.cn.hdfs;


public class testhdfs {
```

```
    public static void main(String[]) {
        /*FileSystem:这个对象是hdfs抽象目录树的一个实例
                    如果用户想要操作hdfs首先需要获取这个实例(静态方法)*/
        //Configuration对象：加载配置文件的对象 Hadoop集群的时候
        Configuration conf=new Configuration();

        FileSystem fs=FileSystem.get(conf);
        System.out.println(fs);
        //输出：org.apache.hadoop.fs.LocalFileSystem@3911c2a7 本地文件系统Windows下
        //Path hdfs内置对象  文件路径对象
        /*
        在文件上传的时候如果没有指定文件名，并且上传的父目录存在，则以原来的文件名创
        上传时如果上传的Path是一个已经存在的目录，则最终上传到该目录下，如果不存在，
        */
        Path src=new Path("D:\\test.txt");
        Path dst=new Path("/");
        fs.copyFromLocalFile(src, dst)
        //运行结果：  将src地址文件上传到代码运行的根目录

        //可以指定获取文件系统的URL连接  完全分布式的uri
        //uri和文件配置信息保持一致
        //FileSystem fs=FileSystem.get(uri, conf);
        FileSystem fs=FileSystem.get(new URI("hdfs://hadoop01:9000"), conf);
        System.out.println(fs);
        //输出：DFS[DFSClient[clientName=DFSClient_NONMAPREDUCE_-42332820_1, ugi=A
        System.out.println(fs instanceof DistributedFileSystem);
        //输出：true
        fs.close()
    }
}
```

将本地文件上传到hdfs文件系统，需要设置文件上传的权限：
1）代码提交的时候设置（run configurations）

```
Prguments是配置程序运行参数的
program arguments:程序中需要给的参数
VM ARGUMENT:jvm运行的时候需要的参数 -DHADOOP_USER_NAME=hadoop
```

```
package com.ghgj.cn.hdfs;

public class testhdfs {
    public static void main(String[]) {
        Configuration conf=new Configuration();
        FileSystem fs=FileSystem.get(new URI("hdfs://hadoop01:9000"), conf);
        Path src=new Path("D:\\test.txt");
        Path dst=new Path("/");
        fs.copyFromLocalFile(src, dst);
        fs.close()
    }
}
```

2）写在代码中，配置实例的时候制定user

- **FileSystem.get(URL uri, Configuration conf, String user)**

```
package com.ghgj.cn.hdfs;

public class testhdfs {
    public static void main(String[]) {
        Configuration conf=new Configuration();
        FileSystem fs=FileSystem.get(new URI("hdfs://hadoop01:9000"), conf, "hadoo
        Path src=new Path("D:\\test.txt");
        Path dst=new Path("/");
        fs.copyFromLocalFile(src, dst);
        fs.close()
    }
}
```

3）写在代码中，参数设置，系统对象

- **System.setProperty(“HADOOP_USER_NAME”， “hadoop”)**

```
package com.ghgj.cn.hdfs;

public class testhdfs {
    public static void main(String[]) {
        System.setProperty("HADOOP_USER_NAME", "hadoop");
```

```
        Configuration conf=new Configuration();
        FileSystem fs=FileSystem.get(new URI("hdfs://hadoop01:9000"), conf);
        Path src=new Path("D:\\test.txt");
        Path dst=new Path("/");
        fs.copyFromLocalFile(src, dst);
        fs.close()
    }
}
```

通过代码上传文件，文件的备份数replication=3.因为configuration对象是读取配置文件.xml
默认：配置文件对象加载的配置文件是来自于jar包中的。
实际上配置文件的加载是有顺序的：

1. jar包中的hdfs-default.xml
2. 工程的classpath(src)上的配置文件:
   只识别两种名字：hdfs-default.xml　　hdfs-site.xml
   ——如果需要更改配置文件，可以在src中新建配置文件.xml，指定修改的参数
3. 通过代码设置配置项

   ```
   Configuration conf=new Configuration();
   conf.set("dfs.replication", "5")
   ```

上传/下载细节学习：

回顾:

- hadoop的**分块存储**机制：一个block的是128M，上传的文件根据大小划分block的数量，交由不同的节点存储(hadoop01/hadoop02...),因此会出现Block ID（唯一）
- hadoop的**备份**机制：根据配置文件的dfs.replication参数决定备份数量。

上传目录在哪里？底层还是上传到某一个节点上

数据真实的存储目录由datanode负责

> 上传完成后在hadoop抽象目录树上只看到文件放在指定(根)目录里
> 真实数据的存储目录在Linux中的目录：__/home/hadoop/data/hadoopdata/data__

该目录下有两个文件：

1. current: 这是块池的目录，存储真实数据

- 由namenode初始化的时候生成**BP-733548501-192.168.75.162-1527835088435**
  所有的块信息都存储在这个目录中，**最终数据存储的目录：/home/hadoop/data/hadoopdata/data/current/BP-733548501-192.168.75.162-1527835088435/current/finalized/subdir0/subdir1**
- **文件在上传的时候会生成两个文件：**
  - blk_1073742080: 原始文件，blk_块的id（全局唯一）
  - blk_1073742080_1256.meta: 原始文件的元数据信息，包括原始文件的长度，创建时间和偏移量等信息
- **文件在下载的时候会生成一个.crc结尾的文件：**
  - crc文件的作用：校验下载的文件和上传的文件是否同一个文件，用于下载的时候进行*文件完整性的校验*。校验原理是*根据文件的起始偏移量个结尾偏移量*。

1. in_use.lock：锁文件，作用是标识datanode进程，一个节点上只允许开启一个datanode进程，这个文件用于锁定datanode进程，只允许一个datanode进程

附录namenode的version：

```
namespaceID=876436202
集群标识
clusterID=CID-5c85f965-3205-4ca1-ad7a-2485955eef22
cTime=0
storageType=NAME_NODE
块池ID  联邦模式下不同的namenode管理数据的blockpoolID不同
blockpoolID=BP-733548501-192.168.75.162-1527835088435
layoutVersion=-63
```

- 联邦模式：
  同一个集群可以有多个主节点namenode，地位是相同的，同一时间可以有多个活跃的namenode，这些namenode共同使用集群中所有的datanode，每个NameNode只负责管理集群中的datanode上的一部分数据。
- blockpoolID介绍：
  每个namenode进行数据管理靠的是Block Pool ID（块池ID），同一个集群的块池ID是相同的

hdfs中常用的API

```
package com.ghgj.cn.hdfs;

public class testhdfs {
    public static void main(String[]) {
        Configuration conf=new Configuration();
        FileSystem fs=FileSystem.get(new URI("hdfs://hadoop01:9000"), conf, "hadoo
```

```
        /*常用的API,代码块
        *......
        */
        }
}
```

创建文件夹

- **fs.mkdirs(f)**

```
        //可以递归创建文件夹
        Path p=new Path("/test");
        fs.mkdirs(p);     //创建成功，返回true
```

删除文件夹，可以指定是否递归

- **fs.delete(f, recursive)**

```
        //默认情况下是递归删除的
        fs.delete(new Path("/test"));
        fs.delete(new Path("/test"), recursive);     //recursive=true/false:是否递归
        //判读目录或文件是否存在
        boolean ss=fs.exists(new Path("/aa.txt"));
        System.out.println(ss);     //输出是true or false
        //重命名:
        fs.rename(new Path("/aa.txt"), new Path("aa1803.txt"));
```

获取指定目录下的文件列表的状态，只能获取文件列表listFiles

重点掌握层级方法:

- **RemoteIterator listFiles = fs.listFiles(f, recursive)**
- **LocatedFileStatus next = listFiles.next()**
- **BlockLocation[] blockLocations = next.getBlockLocations()**
- **String[] hosts = blockLocations.getHosts()**

```
        RemoteIterator<LocatedFileStatus> listFiles = fs.listFiles(new Path("/"),f
        //循环遍历迭代器: listFiles.hasNext()判断迭代器中是否有下一个
        //listFiles.next()获取下一个元素
        while(listFiles.hasNext()){
            //每一个文件的状态信息
            LocatedFileStatus next = listFiles.next();
            system.out.println(next);
            /*
            *    next.getPath()
            *    next.getLen()
            *    next.getBlockSize()
                ......
            */
            //返回每一个文件的块信息，结果封装在数组中，数组的长度代表块的个数
            BlockLocation[] blockLocations = next.getBlockLocations();
            System.out.println(blockLocations.length);

            //bl代表每一个文件的每一个块的信息
            for(BlockLocation bl:blockLocations){
            System.out.println(bl+"\t");

            //返回的是获取的每一个块的存储节点
            String[] hosts = bl.getHosts();
            for(String host:hosts){
                system.out.println(host);
            }
            }
        }
```

输出:

“LocatedFileStatus{path=hdfs://hadoop01:9000/aa1803.txt; isDirectory=false; length=7; replication=2 ……..等文件信息}”

“块的个数”

“0(起始偏移量),7(结尾偏移量),hadoop02,hadoop01”

“hadoop02
hadoop01”

获取指定目录下的文件列表/目录的状态,只能获取状态信息listStatus

- **FileStatus[] listStatus = fs.listStatus(f)**

```
    FileStatus[] listStatus = fs.listStatus(new Path("/"));
    for(FileStatus f:listStatus){
    //是否目录，返回true or false
    system.out.println(f.isDirectory());
    //是否文件，返回true or false
    system.out.println(f.isFile);
    }
```

- 注意：**listFiles vs listStatus**
  listStatus只能获取*状态*信息

采用IO流的方式上传/下载文件

- 文件上传：本地（输入流） ————hdfs（输出流） **从本地读取写到hdfs**

```
package com.ghgj.cn.hdfs;

public class testhdfs {
    public static void main(String[] args) {
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(new URI("hdfs://hadoop01:9000"), conf, "had

       //本地的输入流：普通的输入流     hdfs输出流：hdfs专用的输出流，必须制定文件名
       FileInputStream in = new FileInputStream(new File("C:\\student.txt"));
       //hdfs的输出流----fs.creat()返回的是FSDataOutputStream
       FSDataOutputStream out = fs.creat(new Path("/aa/stu"))
       //IOUtils.copyBytes(InputStream in, OutputStream out, int buffSize)
       IOUtils.copyBytes(in, out, 4096)
}
```

- 文件下载：hdfs（输入流） ————本地（输出流） **从hdfs读取写到本地**

```
package com.ghgj.cn.hdfs;

public class testhdfs {
    public static void main(String[] args) {
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(new URI("hdfs://hadoop01:9000"), conf, "had
```

```
        //创建hdfs输入流----fs.open()返回的是FSDataInputStream
        FSDataInputStream in = fs.open(new Path("/aa/stu"))
        //创建本地的输出流
        FileOutputStream out = new FileOutputStream(new File("C:\\student"))
        //IOUtils.copyBytes(InputStream in, OutputStream out, int buffSize)
        IOUtils.copyBytes(in, out, 4096)
}
```