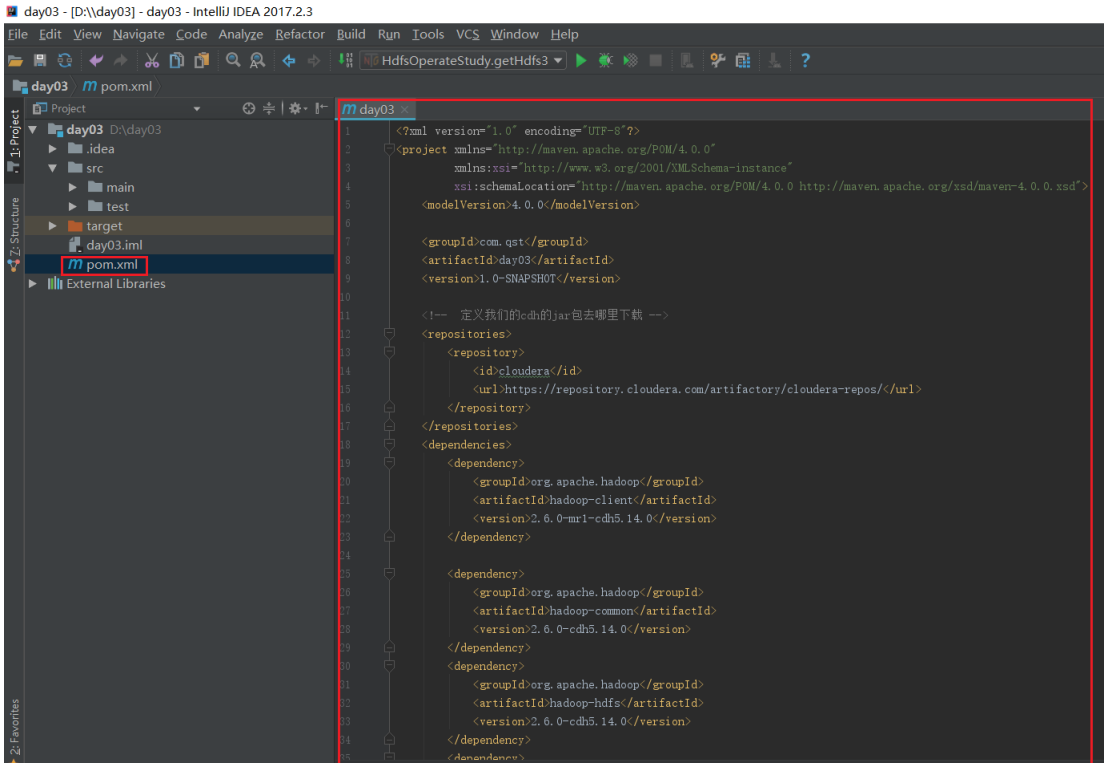
HDFS 的 API 操作

1.1、创建 maven 工程并导入 jar 包

由于 cdh 版本的所有的软件涉及版权的问题，所以并没有将所有的 jar 包托管到 maven 仓库当中去，而是托管在了 CDH 自己的服务器上面，所以我们默认去 maven 的仓库下载不到，需要自己手动的添加 repository 去 CDH 仓库进行下载，以下两个地址是官方文档说明，请仔细查阅

https://www.cloudera.com/documentation/enterprise/release-notes/topics/cdh_vd_cdh5_maven_repo.html

https://www.cloudera.com/documentation/enterprise/release-notes/topics/cdh_vd_cdh5_maven_repo_514x.html

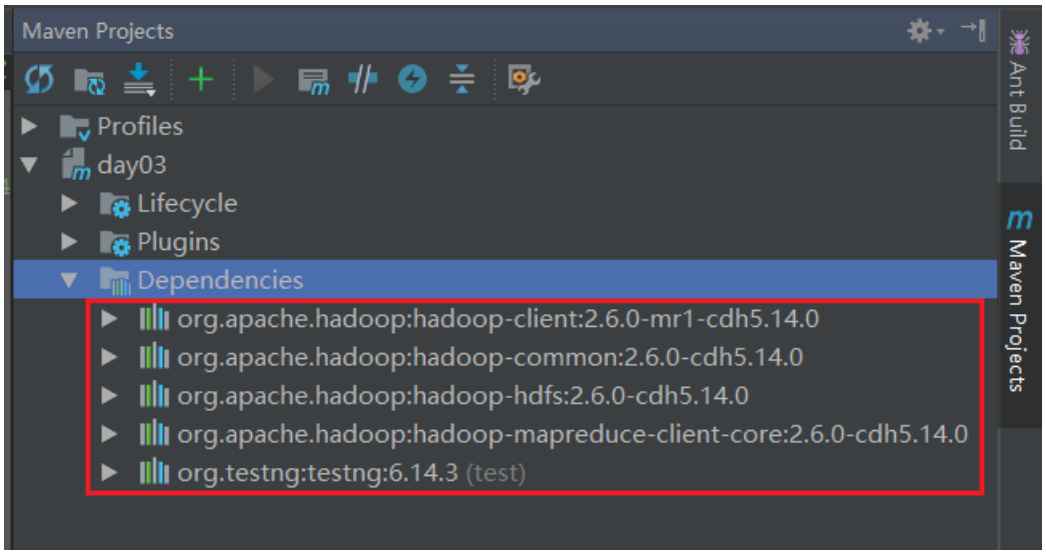



```
<repositories>
        <repository>
```

```
            <id>cloudera</id>

<url>https://repository.cloudera.com/artifactory/cloudera-repos/</url>
        </repository>
    </repositories>
    <dependencies>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-client</artifactId>
            <version>2.6.0-mr1-cdh5.14.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-common</artifactId>
            <version>2.6.0-cdh5.14.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-hdfs</artifactId>
            <version>2.6.0-cdh5.14.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-mapreduce-client-core</artifactId>
            <version>2.6.0-cdh5.14.0</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/junit/junit -->
        <dependency>
            <groupId>junit</groupId>
```

```
            <artifactId>junit</artifactId>
            <version>4.11</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>RELEASE</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.0</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <encoding>UTF-8</encoding>

                </configuration>
            </plugin>

            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>2.4.3</version>
                <executions>
```

```
                <execution>
                    <phase>package</phase>
                    <goals>
                        <goal>shade</goal>
                    </goals>
                    <configuration>
                        <minimizeJar>true</minimizeJar>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```



2、使用文件系统方式访问数据

在 java 中操作 HDFS，主要涉及以下 Class：

Configuration：该类的对象封转了客户端或者服务器的配置；FileSystem（抽象类）：该类的对象是一个文件系统对象，可以用该对象的一些方法来对文件进行操作，

通过 FileSystem 的静态方法 get 获得该对象。

FileSystem fs = FileSystem.get(conf)

get 方法从 conf 中的一个参数 fs.defaultFS 的配置值判断具体是什么类型的文件系统。如果我们的代码中没有指定 fs.defaultFS，并且工程 classpath 下也没有给定相应的配置，conf 中的默认值就来自于 hadoop 的 jar 包中的 core-default.xml，默认值为：file:///，则获取的将不是一个 DistributedFileSystem 的实例，而是一个本地文件系统的客户端对象

3、获取 FileSystem 的几种方式

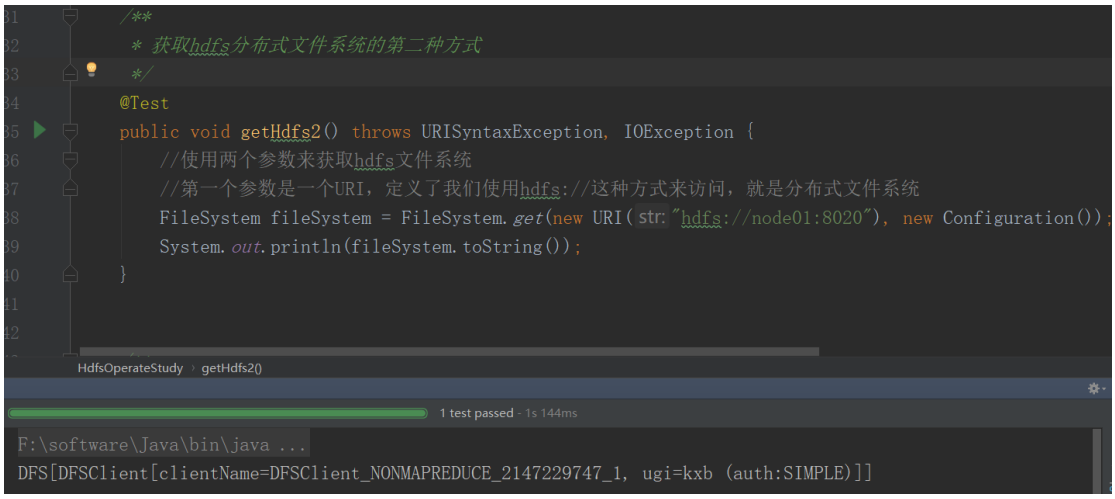
第一种方式获取 FileSystem

```
HdfsOperateStudy.java
14      /**
15       * 通过fileSystem获取分布式文件系统的几种方式
16       */
17      //获取hdfs分布式文件系统的第一种方式
18      @Test
19      public void getFileSystem1() throws IOException {
20          //如果configuration 不做任何配置，获取到的是本地文件系统
21          Configuration configuration = new Configuration();
22          //覆盖我们的hdfs的配置，得到我们的分布式文件系统
23          configuration.set("fs.defaultFS","hdfs://node01:8020/");
24          FileSystem fileSystem = FileSystem.get(configuration);
25          System.out.println(fileSystem.toString());
26      }
27
        HdfsOperateStudy

1 test passed - 1s 104ms
F:\software\Java\bin\java ...
DFS[DFSClient[clientName=DFSClient_NONMAPREDUCE_1999581417_1, ugi=kxb (auth:SIMPLE)]]

==============================================
```

```
import org.apache.commons.io.IOUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;
import org.testng.annotations.Test;
```

```java
import java.io.*;
import java.net.URI;
import java.net.URISyntaxException;
public class HdfsOperateStudy {
    /**
     * 通过 fileSystem 获取分布式文件系统的几种方式
     */
    //获取 hdfs 分布式文件系统的第一种方式
    @Test
    public void getFileSystem1() throws IOException {
        //如果 configuration 不做任何配置，获取到的是本地文件系统
        Configuration configuration = new Configuration();
        //覆盖我们的 hdfs 的配置，得到我们的分布式文件系统
        configuration.set("fs.defaultFS","hdfs://node01:8020/");
        FileSystem fileSystem = FileSystem.get(configuration);
        System.out.println(fileSystem.toString());
    }
}
```

第二种方式获取 FileSystem




```java
    /**
     * 获取 hdfs 分布式文件系统的第二种方式
```

```
     */
    @Test
    public void getHdfs2() throws URISyntaxException, IOException {
        //使用两个参数来获取 hdfs 文件系统
        //第一个参数是一个 URI，定义了我们使用 hdfs://这种方式来访问，
就是分布式文件系统
        FileSystem fileSystem = FileSystem.get(new
URI("hdfs://node01:8020"), new Configuration());
        System.out.println(fileSystem.toString());
    }
```

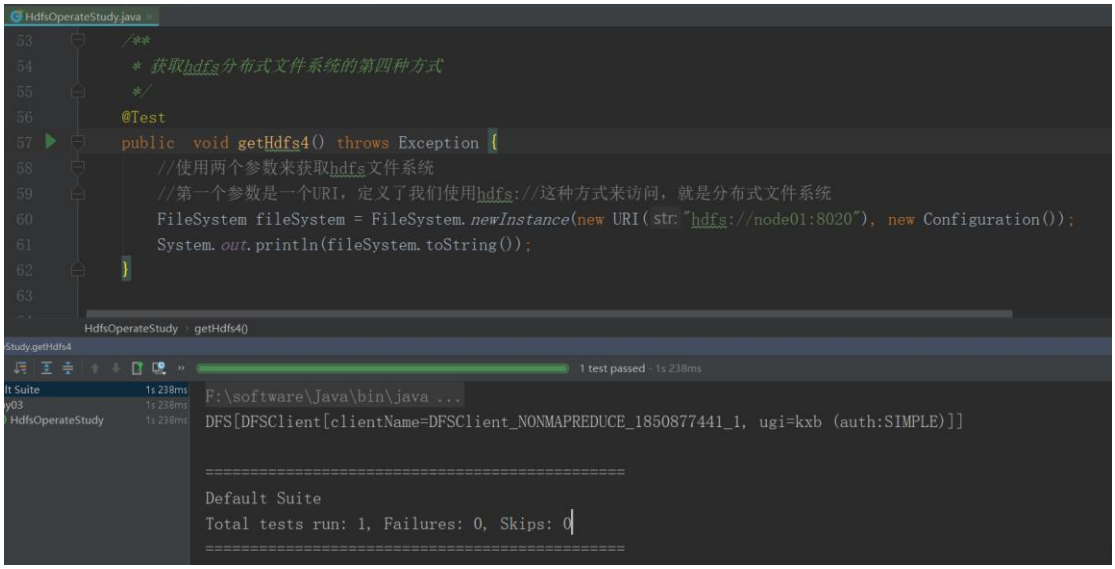
第三种方式获取 FileSystem

```
43      /**
44       * 获取hdfs分布式文件系统的第三种方式
45       */
46      @Test
47      public  void getHdfs3() throws IOException {
48          Configuration configuration = new Configuration();
49          configuration.set("fs.defaultFS","hdfs://node01:8020");
50          FileSystem fileSystem = FileSystem.newInstance(configuration);
51          System.out.println(fileSystem.toString());
52      }
53
        HdfsOperateStudy › getHdfs3()

        1 test passed - 1s 125ms
F:\software\Java\bin\java ...
DFS[DFSClient[clientName=DFSClient_NONMAPREDUCE_-811439946_1, ugi=kxb (auth:SIMPLE)]]

===============================================
Default Suite
Total tests run: 1, Failures: 0, Skips: 0
===============================================
```

```
    /**
     * 获取 hdfs 分布式文件系统的第三种方式
     */
    @Test
    public  void getHdfs3() throws IOException {
        Configuration configuration = new Configuration();
        configuration.set("fs.defaultFS","hdfs://node01:8020");
```

```
        FileSystem fileSystem = FileSystem.newInstance(configuration);
        System.out.println(fileSystem.toString());
    }
```

第四种方式获取 FileSystem




```
    /**
     * 获取 hdfs 分布式文件系统的第四种方式
     */
    @Test
    public  void getHdfs4() throws Exception {
        //使用两个参数来获取 hdfs 文件系统
        //第一个参数是一个 URI，定义了我们使用 hdfs://这种方式来访问，
就是分布式文件系统
        FileSystem     fileSystem     =     FileSystem.newInstance(new
URI("hdfs://node01:8020"), new Configuration());
        System.out.println(fileSystem.toString());
    }
```

1. hdfs 上面创建文件夹

```
/**
 * hdfs上面创建文件夹
 */
@Test
public void createHdfsDir() throws  Exception{
    //获取分布式文件系统的客户端对象
    FileSystem fileSystem = FileSystem.get(new URI("hdfs://node01:8020"), new Configuration());
    fileSystem.mkdirs(new Path("/abc/bbc/ddd"));
    fileSystem.close();

}
```

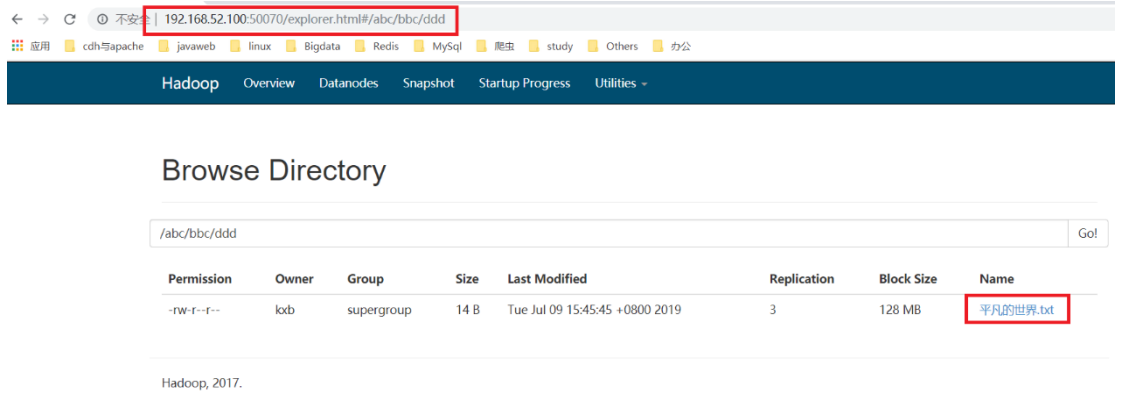
```
/**
 * hdfs 上面创建文件夹
 */
@Test
public void createHdfsDir() throws  Exception{
    //获取分布式文件系统的客户端对象
    FileSystem fileSystem = FileSystem.get(new URI("hdfs://node01:8020"), new Configuration());
    fileSystem.mkdirs(new Path("/abc/bbc/ddd"));
    fileSystem.close();

}
```

2.hdfs 的文件上传

```
/**
 * hdfs的文件上传
 */
@Test
public void uploadFileToHdfs() throws  Exception{
    //获取分布式文件系统的客户端
    FileSystem fileSystem = FileSystem.get(new URI("hdfs://node01:8020"), new Configuration());
    //通过copyFromLocalFile 将我们的本地文件上传到hdfs上面去
    fileSystem.copyFromLocalFile(false,new Path("file:///f:\\平凡的世界.txt"),new Path("/abc/bbc/ddd"));
    fileSystem.close();
}
```

```
/**
 * hdfs 的文件上传
 */
@Test
```

```
    public void uploadFileToHdfs() throws  Exception{
        //获取分布式文件系统的客户端
        FileSystem        fileSystem        =        FileSystem.get(new URI("hdfs://node01:8020"), new Configuration());
        //通过 copyFromLocalFile 将我们的本地文件上传到 hdfs 上面去
        fileSystem.copyFromLocalFile(false,new Path("file:///f:\\平凡的世界.txt"),new Path("/abc/bbc/ddd"));
        fileSystem.close();
    }
```



3. 遍历 hdfs 上面所有的文件

```
    /**
     * 遍历hdfs上面所有的文件
     */
    @Test
    public void listHdfsFiles() throws  Exception{
        FileSystem fileSystem = FileSystem.get(new URI( str: "hdfs://node01:8020"), new Configuration());
        Path path = new Path( pathString: "/");
        //alt  +  shift  +  l  提取变量
        RemoteIterator<LocatedFileStatus> locatedFileStatusRemoteIterator = fileSystem.listFiles(path,  recursive: true);
        //遍历迭代器，获取我们的迭代器里面每一个元素
        while (locatedFileStatusRemoteIterator.hasNext()){
            LocatedFileStatus next = locatedFileStatusRemoteIterator.next();
            Path path1 = next.getPath();
            System.out.println(path1.toString());
        }
        fileSystem.close();
    }
```

```
1 test passed - 1s 406ms
F:\software\Java\bin\java ...
hdfs://node01:8020/test/input01/install.log
hdfs://node01:8020/tmp/hadoop-yarn/staging/history/done/2019/07/07/000000/job_1562489571734_0001-1562495427028-root-Qua
hdfs://node01:8020/tmp/hadoop-yarn/staging/history/done/2019/07/07/000000/job_1562489571734_0001_conf.xml
hdfs://node01:8020/tmp/hadoop-yarn/staging/history/done/2019/07/08/000000/job_1562489571734_0003-1562541030821-root-xxx
hdfs://node01:8020/tmp/hadoop-yarn/staging/history/done/2019/07/08/000000/job_1562489571734_0003_conf.xml
hdfs://node01:8020/tmp/hadoop-yarn/staging/history/done_intermediate/root/job_1562654499490_0001-1562655100314-root-Qua
hdfs://node01:8020/tmp/hadoop-yarn/staging/history/done_intermediate/root/job_1562654499490_0001.summary
hdfs://node01:8020/tmp/hadoop-yarn/staging/history/done_intermediate/root/job_1562654499490_0001_conf.xml
hdfs://node01:8020/tmp/hadoop-yarn/staging/history/done_intermediate/root/job_1562654499490_0002-1562655136918-root-Qua
hdfs://node01:8020/tmp/hadoop-yarn/staging/history/done_intermediate/root/job_1562654499490_0002.summary
hdfs://node01:8020/tmp/hadoop-yarn/staging/history/done_intermediate/root/job_1562654499490_0002_conf.xml
hdfs://node01:8020/tmp/logs/root/logs/application_1562489571734_0001/node02.hadoop.com_45667
hdfs://node01:8020/tmp/logs/root/logs/application_1562489571734_0003/node01.hadoop.com_34677
hdfs://node01:8020/tmp/logs/root/logs/application_1562654499490_0001/node01.hadoop.com_40300
```

```
//遍历 hdfs 上面所有的文件


    @Test
    public void listHdfsFiles() throws  Exception{
        FileSystem        fileSystem        =        FileSystem.get(new
URI("hdfs://node01:8020"), new Configuration());
        Path path = new Path("/");
        //alt  +  shift  +  l  提取变量
        RemoteIterator<LocatedFileStatus>
locatedFileStatusRemoteIterator = fileSystem.listFiles(path, true);
        //遍历迭代器，获取我们的迭代器里面每一个元素
        while (locatedFileStatusRemoteIterator.hasNext()){
            LocatedFileStatus                next                =
locatedFileStatusRemoteIterator.next();
            Path path1 = next.getPath();
            System.out.println(path1.toString());
        }
        fileSystem.close();
    }
```