HDFS 基础使用

目录

1、HDFS 前言

HDFS：Hadoop Distributed File System Hadoop 分布式文件系统，主要用来解决海量数据的存储问题

1、 设计思想

分而治之：将大文件，大批量文件，分布式的存放于大量服务器上。以便于采取分而治之的方式对海量数据进行运算分析

2、 在大数据系统架构中的应用

为各类分布式运算框架（MapReduce，Spark，Tez，Flink，…）提供数据存储服务

3、 重点概念：数据块/副本，负载均衡，心跳机制，副本存放策略，元数据/元数据管理，安全模式，机架感知…

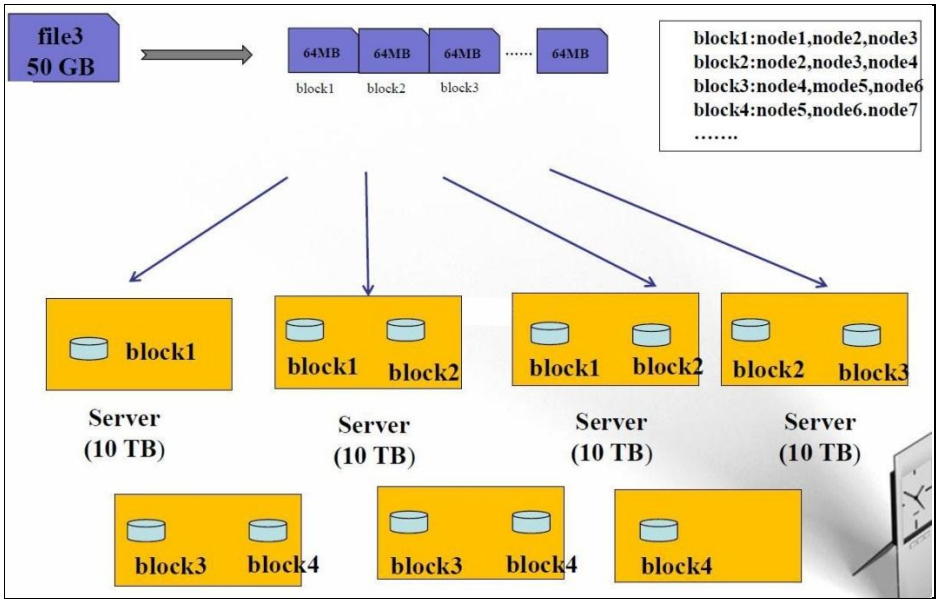
2、HDFS 相关概念和特性

2.1、HDFS 设计思路

HDFS 被设计成用来使用低廉的服务器来进行海量数据的存储，那是怎么做到的呢？

1、 大文件被切割成小文件，使用分而治之的思想让很多服务器对同一个文件进行联合管理

2、 每个小文件做冗余备份，并且分散存到不同的服务器，做到高可靠不丢失

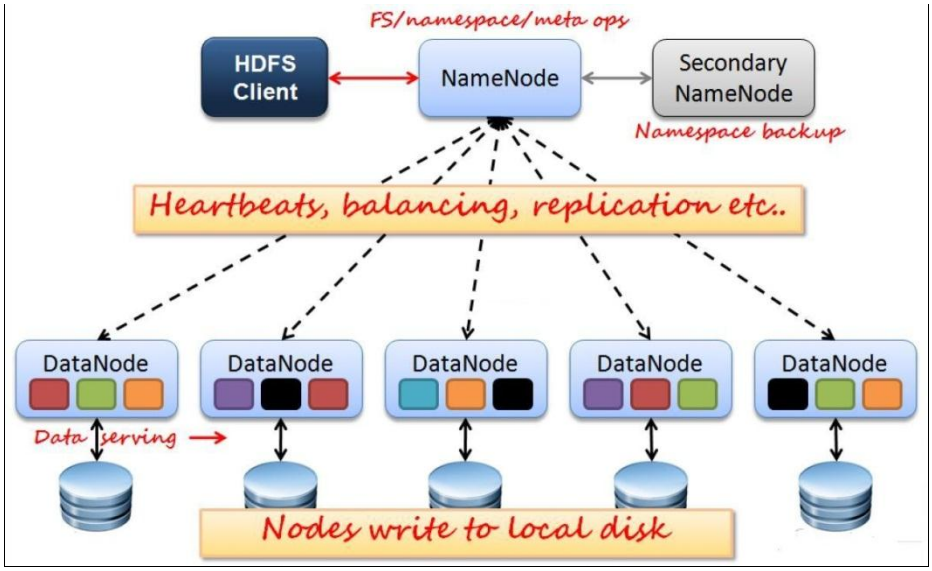


2.2、HDFS 架构

主节点 Namenode：集群老大，掌管文件系统目录树，处理客户端读且请求

SecondaryNamenode：严格说并不是 namenode 备份节点，主要给 namenode 分担压力之用

从节点 Datanode：存储整个集群所有数据块，处理真正数据读写



2.3、概念和特性

首先，它是一个文件系统，用于存储文件，通过统一的命名空间——目录树来定位文件
其次，它是分布式的，由很多服务器联合起来实现其功能，集群中的服务器都有各自清晰的角色定位

重要特性如下：
1、HDFS 中的文件在物理上是分块存储（block），块的大小可以通过配置参数(dfs.blocksize)来规定，默认大小在 hadoop2.x 版本中是 128M，老版本中是 64M

2、HDFS 文件系统会给客户端提供一个统一的抽象目录树，客户端通过路径来访问文件，形如：hdfs://namenode:port/dir-a/dir-b/dir-c/file.data
hdfs://hadoop02:9000/soft/hadoop-2.6.5-centos-6.7.tar.gz

3、目录结构及文件分块位置信息(元数据)的管理由 namenode 节点承担
namenode 是 HDFS 集群主节点，负责维护整个 hdfs 文件系统的目录树，以及每一个路径（文件）所对应的 block 块信息（block 的 id，及所在的 datanode 服务器）

4、文件的各个 block 的存储管理由 datanode 节点承担
datanode 是 HDFS 集群从节点，每一个 block 都可以在多个 datanode 上存储多个副本（副本数量也可以通过参数设置 dfs.replication，默认是 3）

5、HDFS 是设计成适应一次写入，多次读出的场景，且不支持文件的修改

(PS：适合用来做数据分析，并不适合用来做网盘应用，因为，不便修改，延迟大，网络开销大，成本太高)

3、HDFS 优缺点

3.1、HDFS 优点

可构建在廉价机器上 　　通过多副本提高可靠性，提供了容错和恢复机制 高容错性 　　数据自动保存多个副本，副本丢失后，自动恢复 适合批处理 　　移动计算而非数据，数据位置暴露给计算框架 适合大数据处理 　　GB、TB、甚至 PB 级数据，百万规模以上的文件数量，10K+节点规模 流式文件访问 　　一次性写入，多次读取，保证数据一致性

3.2、HDFS 缺点

不适于以下操作：

低延迟数据访问
　　比如毫秒级
　　低延迟与高吞吐率
小文件存取
　　占用 NameNode 大量内存　150b* 1000W = 15E,1.5G
　　寻道时间超过读取时间
并发写入、文件随机修改
　　一个文件只能有一个写者
　　仅支持 append

HDFS 不适合存储小文件

元信息存储在 NameNode 内存中
　　一个节点的内存是有限的
存取大量小文件消耗大量的寻道时间
　　类比拷贝大量小文件与拷贝同等大小的一个大文件
NameNode 存储 block 数目是有限的
　　一个 block 元信息消耗大约 150 byte 内存
　　存储 1 亿个 block，大约需要 20GB 内存
　　如果一个文件大小为 10K，则 1 亿个文件大小仅为 1TB（但要消耗掉 NameNode 20GB 内存）

4、HDFS 的 shell(命令行客户端)操作

HDFS 提供 shell 命令行客户端，使用方法如下：

```
[hadoop@hadoop03 ~]$ hadoop fs -ls /
Found 7 items
-rw-r--r--   2 hadoop supergroup  269980405 2017-07-13 05:03 /hadoop-2.6.5-centos-6.7.tar.gz
drwxr-xr-x   - hadoop supergroup          0 2017-07-13 18:10 /hbase
drwxr-xr-x   - hadoop supergroup          0 2017-07-12 01:01 /sqoop
drwxr-xr-x   - hadoop supergroup          0 2017-07-12 01:42 /sqoopdata
drwxrwx---   - hadoop supergroup          0 2017-07-12 01:09 /tmp
drwxr-xr-x   - hadoop supergroup          0 2017-07-12 01:11 /user
drwxr-xr-x   - hadoop supergroup          0 2017-07-30 19:17 /wc
```

HDFS 支持的其他命令行参数如下：

```
[-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] <localsrc> ... <dst>]
[-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
[-count [-q] <path> ...]
[-cp [-f] [-p] <src> ... <dst>]
[-createSnapshot <snapshotDir> [<snapshotName>]]
[-deleteSnapshot <snapshotDir> <snapshotName>]
[-df [-h] [<path> ...]]
[-du [-s] [-h] <path> ...]
[-expunge]
[-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
[-getfacl [-R] <path>]
[-getmerge [-nl] <src> <localdst>]
[-help [cmd ...]]
[-ls [-d] [-h] [-R] [<path> ...]]
[-mkdir [-p] <path> ...]
[-moveFromLocal <localsrc> ... <dst>]
[-moveToLocal <src> <localdst>]
[-mv <src> ... <dst>]
[-put [-f] [-p] <localsrc> ... <dst>]
[-renameSnapshot <snapshotDir> <oldName> <newName>]
[-rm [-f] [-r|-R] [-skipTrash] <src> ...]
[-rmdir [--ignore-fail-on-non-empty] <dir> ...]
[-setfacl [-R] [{-b|-k} {-m|-x <acl_spec>} <path>]|[--set <acl_spec> <path>]]
[-setrep [-R] [-w] <rep> <path> ...]
[-stat [format] <path> ...]
[-tail [-f] <file>]
[-test -[defsz] <path>]
[-text [-ignoreCrc] <src> ...]
[-touchz <path> ...]
[-usage [cmd ...]]
```

常用命令参数介绍：

-help

功能：输出这个命令参数手册

[hadoop@hadoop02 ~]$ hadoop -help

[hadoop@hadoop02 ~]$ hadoop fs -help

[hadoop@hadoop02 ~]$ hadoop fs -help ls

-ls

功能：显示目录信息

示例： hadoop fs -ls hdfs://hadoop02:9000/

备注：这些参数中，所有的 hdfs 路径都可以简写成 hadoop fs -ls / 等同上条命令的效果

-mkdir

功能：在 hdfs 上创建目录

示例：hadoop fs -mkdir -p /aa/bb/cc/dd

-put
功能：等同于 copyFromLocal，进行文件上传
示例：hadoop fs -put /aaa/jdk.tar.gz /bbb/jdk.tar.gz.2

-get
功能：等同于 copyToLocal，就是从 hdfs 下载文件到本地
示例：hadoop fs -get /aaa/jdk.tar.gz

-getmerge
功能：合并下载多个文件
示例：比 **getmerge** 如 hdfs 的目录 /aaa/下有多个文件:log.1, log.2,log.3,...
hadoop fs -getmerge /aaa/log.* ./log.sum

-cp
功能：从 hdfs 的一个路径拷贝 hdfs 的另一个路径
示例： hadoop fs -cp /aaa/jdk.tar.gz /bbb/jdk.tar.gz.2

-mv
功能：在 hdfs 目录中移动文件
示例： hadoop fs -mv /aaa/jdk.tar.gz /

-rm
功能：删除文件或文件夹
示例：hadoop fs -rm -r /aaa/bbb/

-rmdir
功能：删除空目录
示例：hadoop fs -rmdir /aaa/bbb/ccc

-moveFromLocal
功能：从本地剪切到 hdfs
示例：hadoop fs - moveFromLocal /home/hadoop/a.txt /aa/bb/cc/dd

-moveToLocal
功能：从 hdfs 剪切到本地
示例：hadoop fs - moveToLocal /aa/bb/cc/dd /home/hadoop/a.txt

-copyFromLocal
功能：从本地文件系统中拷贝文件到 hdfs 文件系统去
示例：hadoop fs -copyFromLocal ./jdk.tar.gz /aaa/

-copyToLocal
功能：从 hdfs 拷贝到本地
示例：hadoop fs -copyToLocal /aaa/jdk.tar.gz

-appendToFile
功能：追加一个文件到已经存在的文件末尾
示例：hadoop fs -appendToFile ./hello.txt hdfs://hadoop-server01:9000/hello.txt
可以简写为：
hadoop fs -appendToFile ./hello.txt /hello.txt

-cat

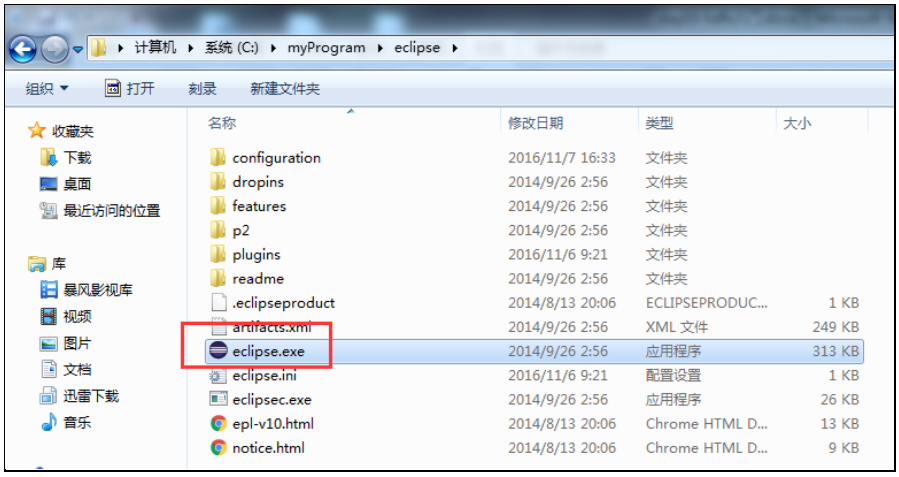
功能：显示文件内容 hadoop fs -cat /hello.txt
-tail **功能：显示一个文件的末尾** 示例：hadoop fs -tail /weblog/access_log.1
-text **功能：以字符形式打印一个文件的内容** 示例：hadoop fs -text /weblog/access_log.1
-chgrp **-chmod** **-chown** **功能：linux 文件系统中的用法一样，对文件所属权限** 示例： hadoop fs -chmod 666 /hello.txt hadoop fs -chown someuser:somegrp /hello.txt
-df **功能：统计文件系统的可用空间信息** 示例：hadoop fs -df -h / **-du** **功能：统计文件夹的大小信息** 示例：hadoop fs -du -s -h /aaa/*
-count **功能：统计一个指定目录下的文件节点数量** 示例：hadoop fs -count /aaa/
-setrep **功能：设置 hdfs 中文件的副本数量** 示例：hadoop fs -setrep 3 /aaa/jdk.tar.gz
补充：查看 dfs 集群工作状态的命令 **hdfs dfsadmin -report**

5、HDFS 的 Java API 操作

hdfs 在生产应用中主要是客户端的开发，其核心步骤是从 hdfs 提供的 api 中构造一个 HDFS 的访问客户端对象，然后通过该客户端对象操作（增删改查）HDFS 上的文件

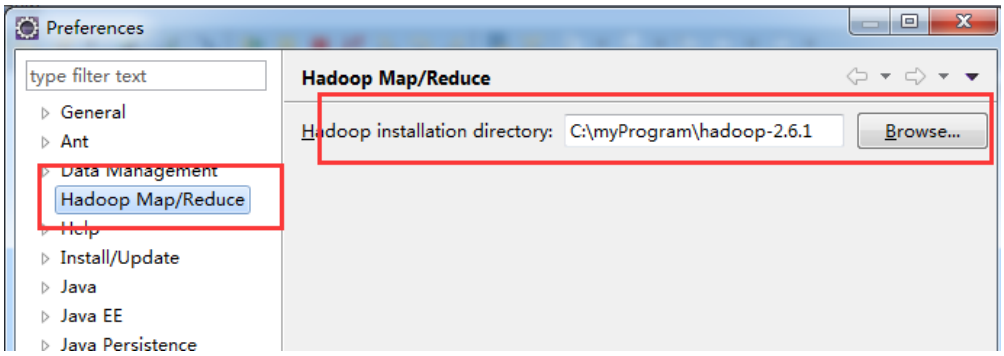
5.1、利用 eclipse 查看 hdfs 集群的文件信息

1、 下载一个 eclipse 开发工具 eclipse-jee-luna-SR1-win32-x86_64.zip
2、 解压到一个文件夹 C:\myProgram\eclipse
3、 把 hadoop-eclipse-plugin-2.6.5.jar 放入到 eclipse/plugins 文件夹下
4、 双击启动 eclipse



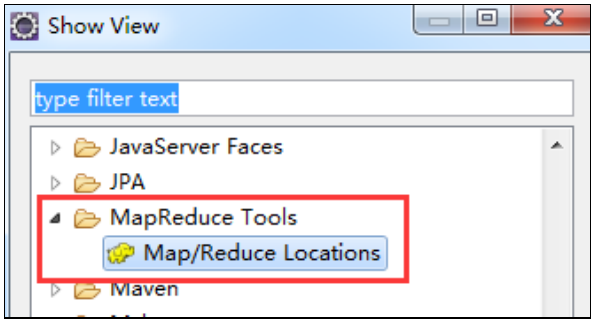
5、 将” windows 平台编译 hadoop-2.6.1.zip”解压到 windows 系统下一个文件夹下，文件夹的路径最好不要带中文。我的目录是：C:\myProgram\hadoop-2.6.1

6、 打开了 eclipse 之后，点击 windows –> prefrences -> 会出现一个对话框。找到如图所示 Hadoop MapReduce 选项：然后把你安装的 hadoop 路径配置上，就是上一步你解压的那个文件夹：C:\myProgram\hadoop-2.6.1



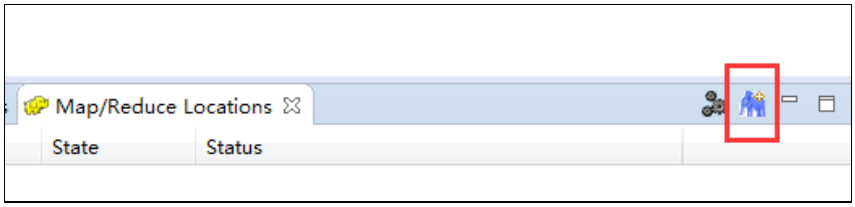
然后保存

7、 然后点击 windows → show view → other 在列表中找到图中这个东西：

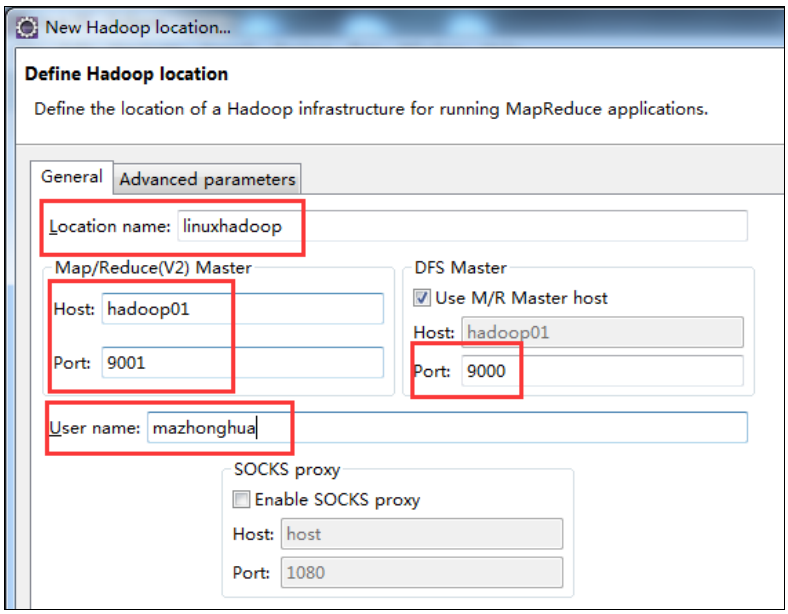


然后双击

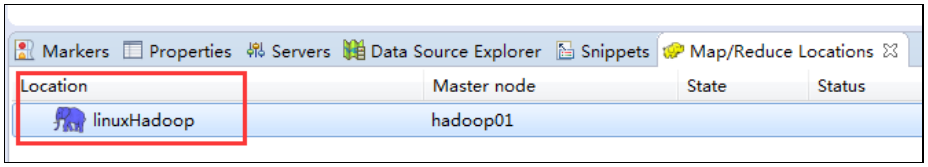
8、 会出现这么一个显示框，如下图



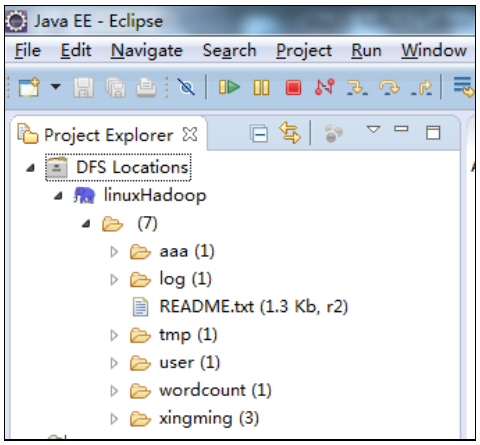
9、咱们点击红框中这个东西，会出现相应的这么一个对话框，修改相应的信息，



10、填完以上信息之后，点击 finish 会出现：



11、最重要的时候在左上角的这个地方会出现：



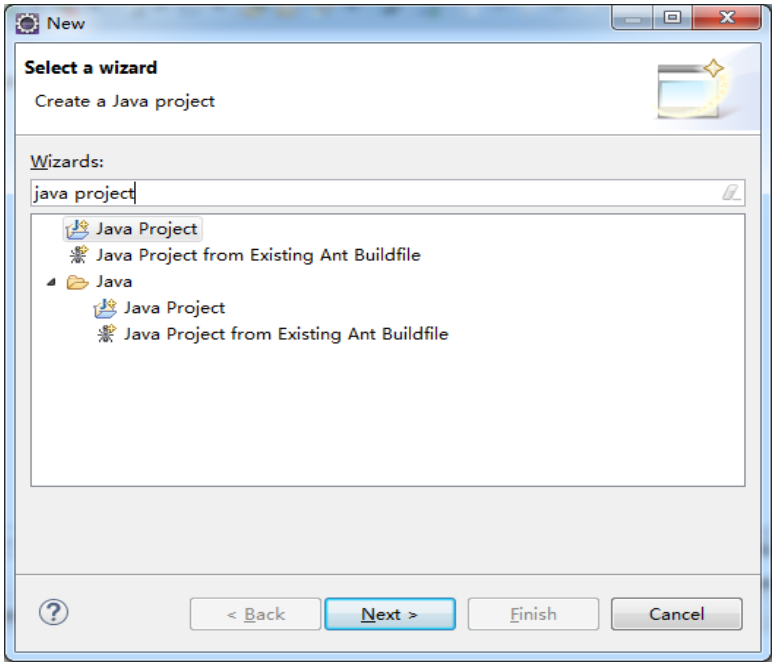
至此，我们便完成了，利用 hadoop 的 eclipse 插件链接 hdfs 集群实现查看 hdfs 集群文件的功能，大功告成。!!!! 恭喜各位

5.2、搭建开发环境

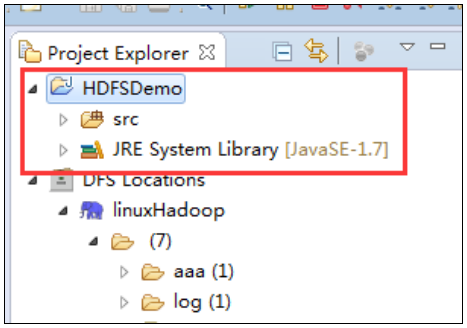
可以有两种方式：

第一种：建立普通 java 工程，加入操作 hdfs 的 jar 包

1、 按键 ctrl + N 搜索 java project 建立普通 java 工程，如图所示：



2、 输入项目名字，点击确定，生成一个普通 java 工程



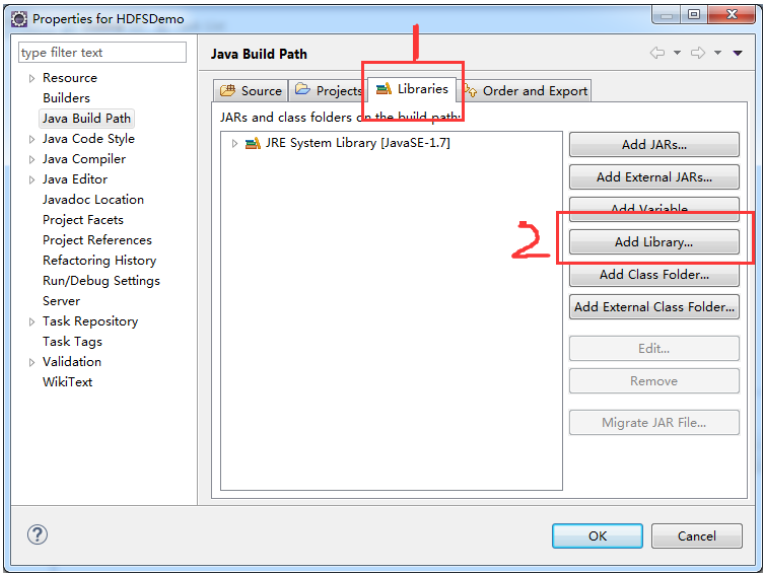
注意：建议大家使用 jdk 的版本和 linux 服务上的 jdk 版本一致。在这儿我选择的都是 jdk1.7 的大版本

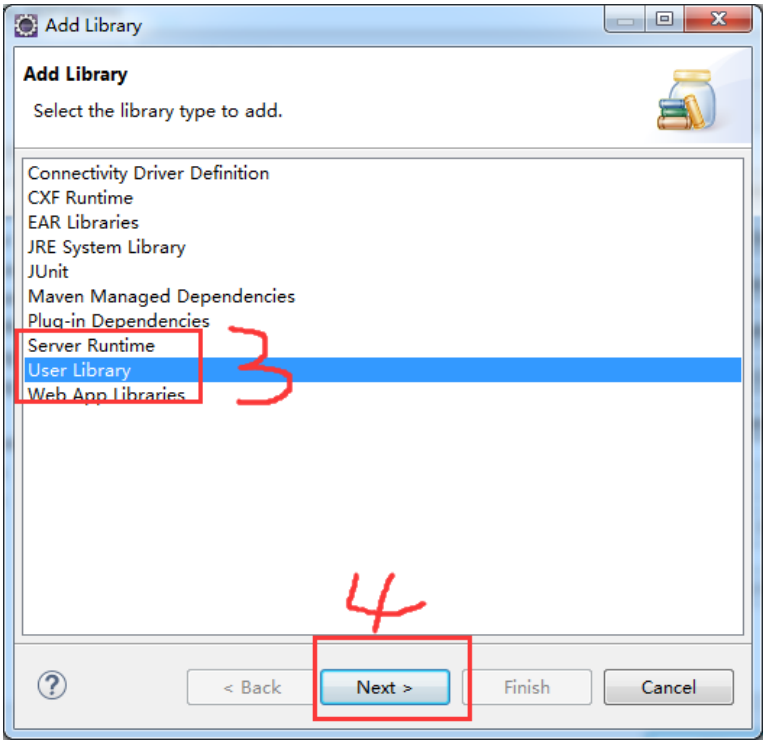
3、 加入依赖 jar，有两种方式

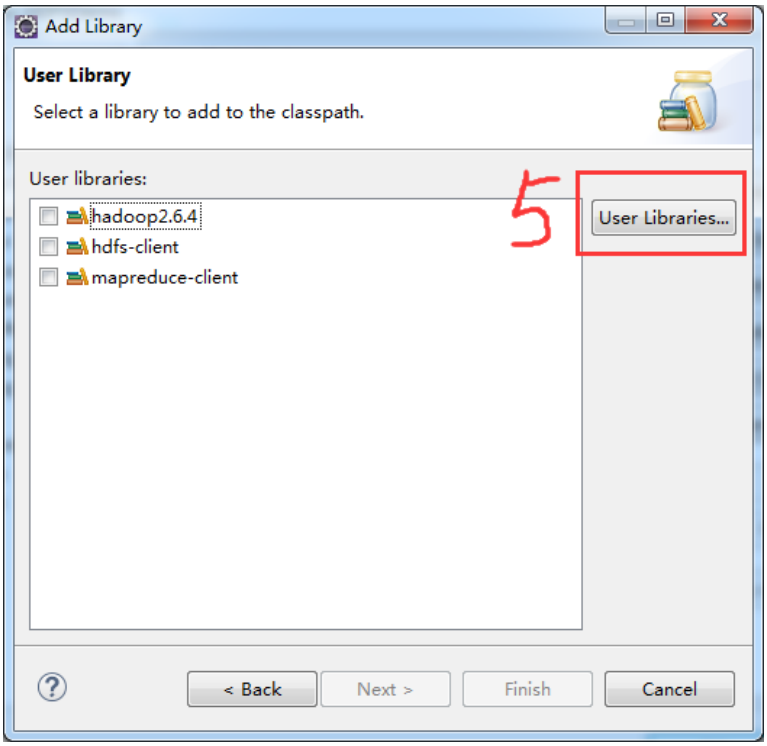
A、 先在项目中建立一个文件夹 lib，然后添加 hadoop 安装包下 share/hadoop/common 和 share/hadoop/hdfs 下的 jar 和它所依赖的 lib 目录下的 jar 包，然后把加入的所有的 jar 包都 add classpath

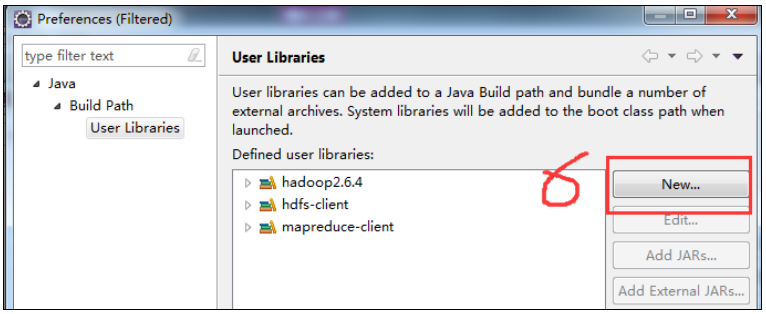
B、 添加自己的依赖库，具体操作请按图所示操作进行：

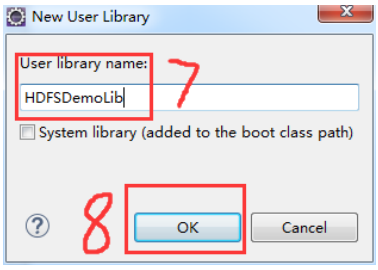
1、在项目名字上鼠标右键选择 Build Path ,然后点击右边出现的 Configure Build Path, 出现：

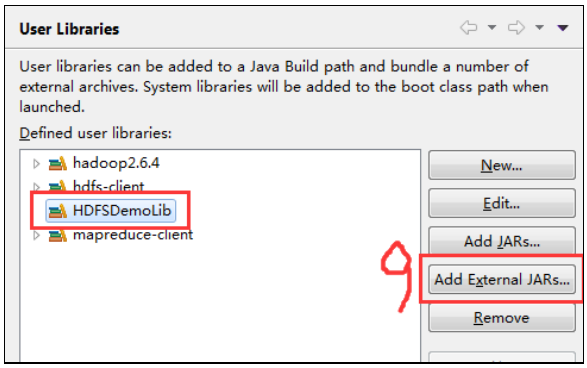

Properties for HDFSDemo
type filter text
Resource
Builders
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
Project Facets
Project References
Refactoring History
Run/Debug Settings
Server
Task Repository
Task Tags
Validation
WikiText
Java Build Path
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
JRE System Library [JavaSE-1.7]
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
OK
Cancel
1
2

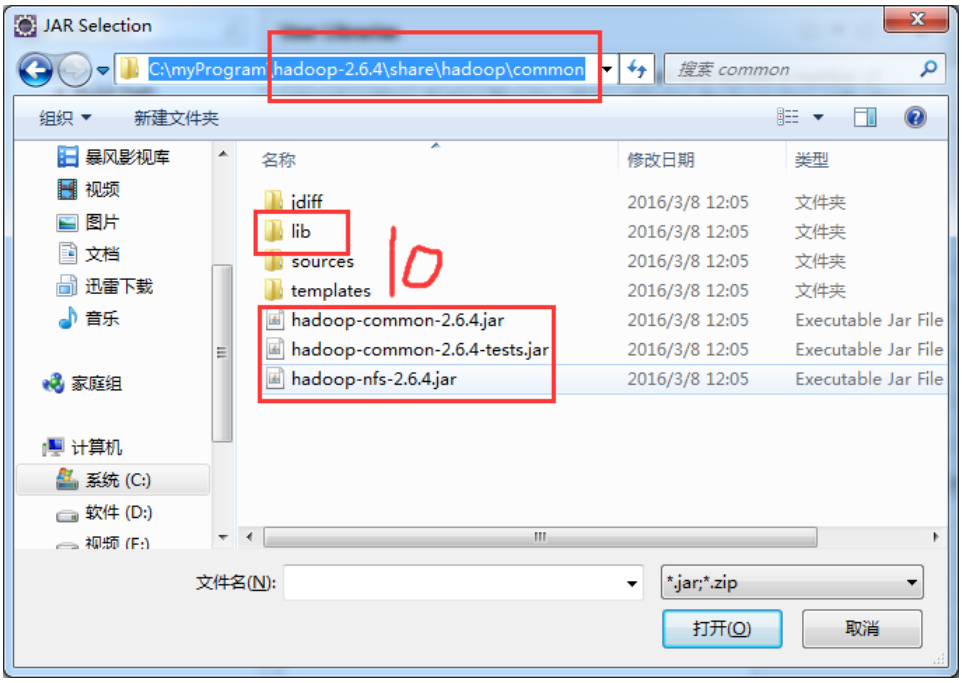

Add Library
Add Library
Select the library type to add.
Connectivity Driver Definition
CXF Runtime
EAR Libraries
JRE System Library
JUnit
Maven Managed Dependencies
Plug-in Dependencies
Server Runtime
User Library
Web App Libraries
3
4
< Back
Next >
Finish
Cancel

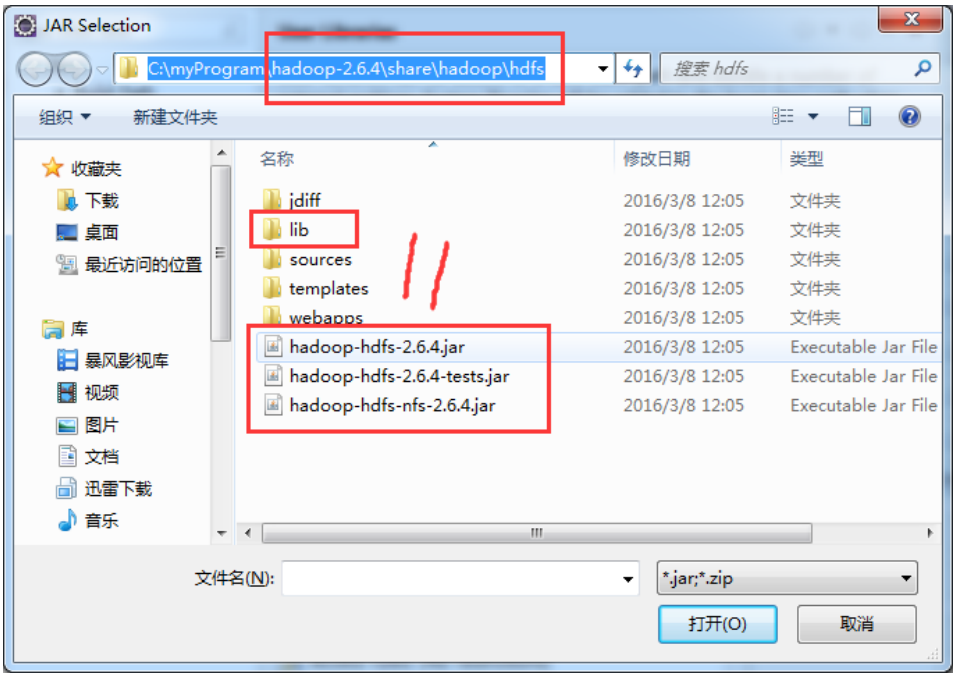

Add Library
User Library
Select a library to add to the classpath.
User libraries:
hadoop2.6.4
hdfs-client
mapreduce-client
5
User Libraries...
< Back
Next >
Finish
Cancel


Preferences (Filtered)
type filter text
Java
Build Path
User Libraries
User Libraries
User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.
Defined user libraries:
hadoop2.6.4
hdfs-client
mapreduce-client
6
New...
Edit...
Add JARs...
Add External JARs...

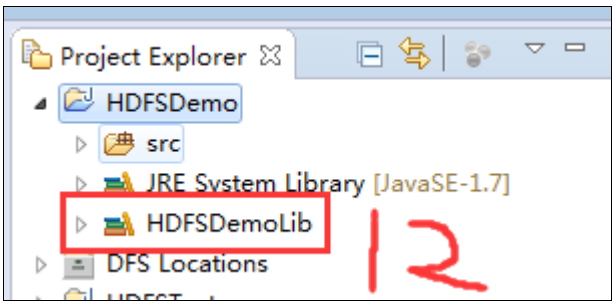

New User Library
User library name:
HDFSDemoLib
7
System library (added to the boot class path)
8
OK
Cancel


User Libraries
User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.
Defined user libraries:
hadoop2.6.4
hdfs-client
HDFSDemoLib
mapreduce-client
9
New...
Edit...
Add JARs...
Add External JARs...
Remove





都添加好了，点击 OK ，再点击 Finish，再点击 OK，项目会变成这样：



到此表示，我们利用 HDFS 的 api 编写业务代码所依赖的 jar 包都添加完成，接下来便可

以愉快的玩耍代码了。

5.3、FileSystem 实例获取讲解（重点）

在 java 中操作 hdfs，首先要获得一个客户端实例：

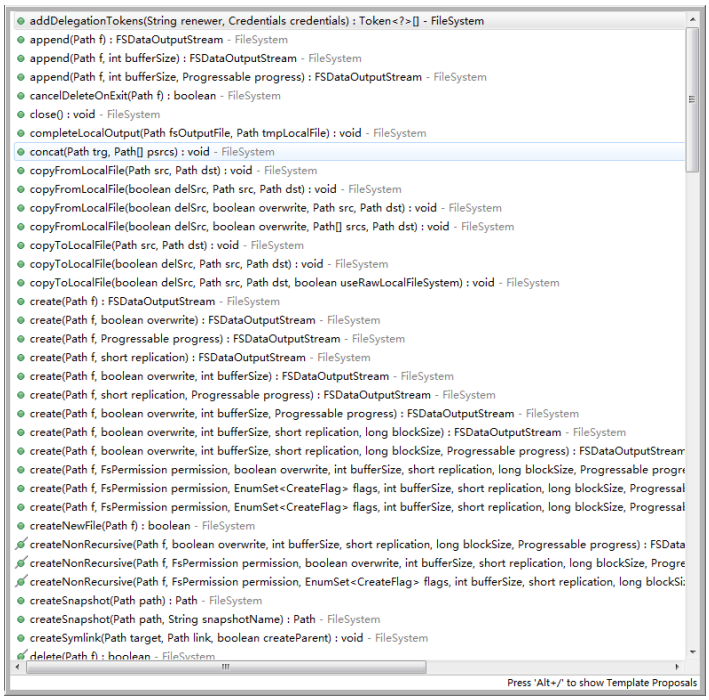
```
Configuration conf = new Configuration()
FileSystem fs = FileSystem.get(conf)
```

而我们的操作目标是 HDFS，所以获取到的 fs 对象应该是 DistributedFileSystem 的实例；
get 方法是从何处判断具体实例化那种客户端类呢？
--从 conf 中的一个参数 fs.defaultFS 的配置值判断；

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

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

    Configuration conf = new Configuration();

    // 如果我们没有给conf设置文件系统，那么fileSystem默认获取的是本地文件系统的一个实例
    // 如果我们设置了下面的这个参数，这表示获取的是该uri的文件系统的实例，就是我们需要的hdfs集群的一个filesystem
    conf.set("fs.defaultFS", "hdfs://hadoop01:9000");

    FileSystem fileSystem = FileSystem.get(conf);

    System.out.println(fileSystem.getUri());
}
```

5.4、DistributedFileSystem 实例所具备的方法介绍



5.5、HDFS 常用 Java API 代码演示

1、 建立文件夹

```
@Test
public void testMkdir() throws Exception{
    System.out.println(fs.mkdirs(new Path("/ccc/bbb/aaa")));
    fs.close();
}
```

2、 上传文件

```
@Test
public void testCopyFromLocal() throws Exception {
    // 要上传的文件所在的本地路径
    // 要上传到hdfs的目标路径*/
    Path src = new Path("C:/software/hadoop-eclipse-plugin-2.6.4.jar");
    Path dst = new Path("/");
    fs.copyFromLocalFile(src, dst);
    fs.close();
}
```

3、 下载文件

```
@Test
public void testCopyToLocal() throws Exception{
    fs.copyToLocalFile(false, new Path("/hadoop-eclipse-plugin-2.6.4.jar"), new Path("d:/"), true);
    fs.close();
}
```

4、 删除文件或者文件夹

```
@Test
public void testRemoveFileOrDir() throws Exception{
    // 删除文件或者文件夹，如果文件夹不为空，这第二个参数必须有， 而且要为true
    fs.delete(new Path("/ccc/bbb"), true);
    fs.close();
}
```

5、 重命名文件或者文件夹

```
    @Test
    public void testRenameFileOrDir() throws Exception{
//      fs.rename(new Path("/ccc"), new Path("/vvv"));
        fs.rename(new Path("/hadoop-eclipse-plugin-2.6.4.jar"),
                new Path("/eclipsePlugin.jar"));
        fs.close();
    }
```

6、 查看目录信息，只显示该文件夹下的文件信息

```
@Test
public void testListFiles() throws Exception{
    RemoteIterator<LocatedFileStatus> listFiles = fs.listFiles(new Path("/"), true);
    while(listFiles.hasNext()){
        LocatedFileStatus fileStatus = listFiles.next();
        System.out.println(fileStatus.getPath());
        System.out.println(fileStatus.getPath().getName());
        System.out.println(fileStatus.getBlockSize());
        System.out.println(fileStatus.getPermission());
        System.out.println(fileStatus.getReplication());
        System.out.println(fileStatus.getLen());

        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 str: hosts){
                System.out.print(str+"\t");
            }
            System.out.println();
        }

        System.out.println("---------------------------------------");
    }
}
```

7、 查看文件及文件夹信息

```
@Test
public void testListStatus() throws Exception{
    FileStatus[] listStatus = fs.listStatus(new Path("/"));
    String flag = "";
    for(FileStatus status:listStatus){
        if(status.isDirectory()){
            flag = "Directory";
        }else{
            flag = "File";
        }
        System.out.println(flag+"\t"+status.getPath().getName());
    }
    fs.close();
}
```

5.6、HDFS 流式数据访问

```
/**
 * 相对那些封装好的方法而言的更底层一些的操作方式 上层那些 mapreduce spark 等运算框架，去 hdfs 中获取数据的时候，就是调的这种底层的 api
 */
public class StreamAccess {
    FileSystem fs = null;
    @Before
    public void init() throws Exception {
        Configuration conf = new Configuration();
        System.setProperty("HADOOP_USER_NAME", "root");
        conf.set("fs.defaultFS", "hdfs:// hadoop01:9000");
        fs = FileSystem.get(conf);
        // fs = FileSystem.get(new URI("hdfs://hadoop01:9000"), conf, "hadoop");
```

```
    }

    @Test
    public void testDownLoadFileToLocal() throws IllegalArgumentException,IOException {
        // 先获取一个文件的输入流----针对 hdfs 上的
        FSDataInputStream in = fs.open(new Path("/jdk-7u65-linux-i586.tar.gz"));
        // 再构造一个文件的输出流----针对本地的
        FileOutputStream out = new FileOutputStream(new File("c:/jdk.tar.gz"));
        // 再将输入流中数据传输到输出流
        IOUtils.copyBytes(in, out, 4096);
    }

    @Test
    public void testUploadByStream() throws Exception {
        // hdfs 文件的输出流
        FSDataOutputStream fsout = fs.create(new Path("/aaa.txt"));
        // 本地文件的输入流
        FileInputStream fsin = new FileInputStream("c:/111.txt");
        IOUtils.copyBytes(fsin, fsout, 4096);
    }

    /**
     * hdfs 支持随机定位进行文件读取，而且可以方便地读取指定长度 用于上层分布式运
    算框架并发处理数据
     */
    @Test
    public void testRandomAccess() throws IllegalArgumentException, IOException {
        // 先获取一个文件的输入流----针对 hdfs 上的
        FSDataInputStream in = fs.open(new Path("/iloveyou.txt"));
        // 可以将流的起始偏移量进行自定义
        in.seek(22);
        // 再构造一个文件的输出流----针对本地的
        FileOutputStream out = new FileOutputStream(new File("d:/iloveyou.line.2.txt"));
        IOUtils.copyBytes(in, out, 19L, true);
    }
}
```

5.7、经典案例

在 mapreduce 、spark 等运算框架中，有一个核心思想就是将运算移往数据，或者说，就是要在并发计算中尽可能让运算本地化，这就需要获取数据所在位置的信息并进行相应范围读取。以下模拟实现：获取一个文件的所有 block 位置信息，然后读取指定 block 中的内容

```
@Test
public void testCat() throws IllegalArgumentException, IOException {

    FSDataInputStream in = fs.open(new Path("/weblog/input/access.log.10"));
    // 拿到文件信息
    FileStatus[] listStatus = fs.listStatus(new Path("/weblog/input/access.log.10"));
    // 获取这个文件的所有 block 的信息
    BlockLocation[] fileBlockLocations = fs.getFileBlockLocations(
            listStatus[0], 0L, listStatus[0].getLen());

    // 第一个 block 的长度
    long length = fileBlockLocations[0].getLength();
    // 第一个 block 的起始偏移量
    long offset = fileBlockLocations[0].getOffset();

    System.out.println(length);
    System.out.println(offset);

    // 获取第一个 block 写入输出流
    // IOUtils.copyBytes(in, System.out, (int)length);
    byte[] b = new byte[4096];

    FileOutputStream os = new FileOutputStream(new File("d:/block0"));
    while (in.read(offset, b, 0, 4096) != -1) {
        os.write(b);
        offset += 4096;
        if (offset > length)
            return;
    }
    os.flush();
    os.close();
    in.close();
}
```

6、HDFS 核心设计

6.1、HADOOP 心跳机制（heartbeat）

1、 Hadoop 是 Master/Slave 结构，Master 中有 NameNode 和 ResourceManager，Slave 中有 Datanode 和 NodeManager

2、 Master 启动的时候会启动一个 IPC（Inter-Process Comunication，进程间通信）server 服务，等待 slave 的链接

3、 Slave 启动时，会主动链接 master 的 ipc server 服务，并且每隔 3 秒链接一次 master，这个间隔时间是可以调整的，参数为 dfs.heartbeat.interval，这个每隔一段时间去连接一次的机制，我们形象的称为心跳。Slave 通过心跳汇报自己的信息给 master，master 也通过心跳给 slave 下达命令

4、 NameNode 通过心跳得知 Datanode 的状态
ResourceManager 通过心跳得知 NodeManager 的状态

5、 如果 master 长时间都没有收到 slave 的心跳，就认为该 slave 挂掉了。!!!!!

Namenode 感知到 Datanode 掉线死亡的时长计算：

HDFS 默认的超时时间为 10 分钟+30 秒。
这里暂且定义超时时间为 timeout
计算公式为：
timeout = 2 * heartbeat.recheck.interval + 10 * dfs.heartbeat.interval

而默认的 heartbeat.recheck.interval 大小为 5 分钟，dfs.heartbeat.interval 默认的大小为 3 秒。
需要注意的是 hdfs-site.xml 配置文件中的 heartbeat.recheck.interval 的单位为毫秒，dfs.heartbeat.interval 的单位为秒
所以，举个例子，如果 heartbeat.recheck.interval 设置为 5000（毫秒），dfs.heartbeat.interval 设置为 3（秒，默认），则总的超时时间为 40 秒

```
<property>
        <name>heartbeat.recheck.interval</name>
        <value>5000</value>
</property>
<property>
        <name>dfs.heartbeat.interval</name>
        <value>3</value>
</property>
```

6.2、HDFS 安全模式

问题引出：集群启动后，可以查看目录，但是上传文件时报错，打开 web 页面可看到 namenode 正处于 safemode 状态，怎么处理？

解释：safemode 是 namenode 的一种状态（active/standby/safemode 安全模式）

namenode 进入安全模式的原理：

a、 namenode 发现集群中的 block 丢失率达到一定比例时（0.1%），namenode 就会进入安全模式，在安全模式下，客户端不能对任何数据进行操作，只能查看元数据信息（比

如 ls/mkdir）

这个丢失率是可以手动配置的，默认是 dfs.safemode.threshold.pct=0.999f

b、如何退出安全模式？

1、找到问题所在，进行修复（比如修复宕机的 datanode）

2、或者可以手动强行退出安全模式（但是并没有真正解决问题）

在 hdfs 集群正常冷启动时，namenode 也会在 safemode 状态下维持相当长的一段时间，此时你不需要去理会，等待它自动退出安全模式即可

正常启动的时候进入安全的原理：

(原理：namenode 的内存元数据中，包含文件路径、副本数、blockid，及每一个 block 所在 datanode 的信息，而 fsimage 中，不包含 block 所在的 datanode 信息，那么，当 namenode 冷启动时，此时内存中的元数据只能从 fsimage 中加载而来，从而就没有 block 所在的 datanode 信息——>就会导致 namenode 认为所有的 block 都已经丢失——>进入安全模式——>datanode 启动后，会定期向 namenode 汇报自身所持有的 blockid 信息，——>随着 datanode 陆续启动，从而陆续汇报 block 信息，namenode 就会将内存元数据中的 block 所在 datanode 信息补全更新——>找到了所有 block 的位置，从而自动退出安全模式)

安全模式常用操作命令：

```
hdfs  dfsadmin  -safemode  leave    //强制 NameNode 退出安全模式
hdfs  dfsadmin  -safemode  enter    //进入安全模式
hdfs  dfsadmin  -safemode  get      //查看安全模式状态
hdfs  dfsadmin  -safemode  wait     //等待，一直到安全模式结束
```

如果你使用的版本是 2.X 之前的版本，那么这个 hdfs 命令可以替换成 hadoop，它们都在 bin 目录下

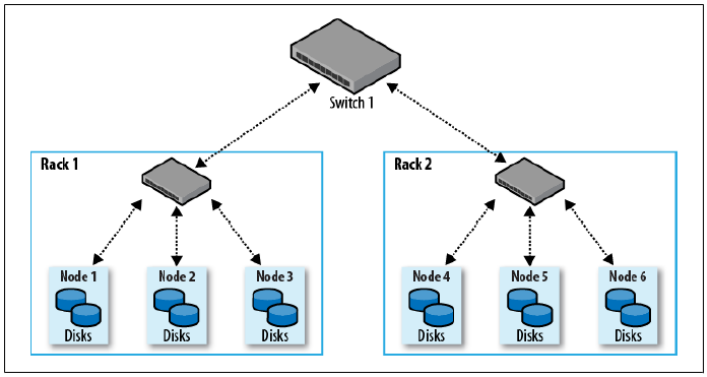
6.3、HDFS 副本存放策略

1、 作用：

数据分块存储和副本的存放，是保证可靠性和高性能的关键

2、 方法：

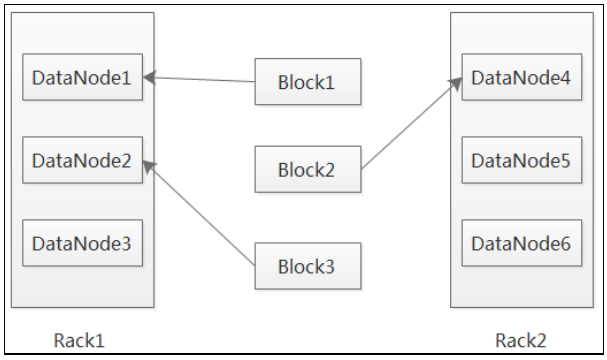
将每个文件的数据进行分块存储，每一个数据块又保存有多个副本，这些数据块副本分布在不同的机器节点上



3、 存放说明：

在多数情况下，HDFS 默认的副本系数是 3

Hadoop 默认对 3 个副本的存放策略如下图：其中 Block1，Block2，Block3 分别表示 Block 的三个副本：



第一个 block 副本放在和 client 所在的 node 里(如果 client 不在集群范围内，则这第一个 node 是随机选取的，系统会尝试不选择哪些太满或者太忙的 node)。

第二个副本放置在与第一个节点不同的机架中的 node 中（近乎随机选择，系统会尝试不选择哪些太满或者太忙的 node)。

第三个副本和第二个在同一个机架，随机放在不同的 node 中。

4、 修改副本数：

第一种方式：修改集群文件 hdfs-site.xml

```
<property>
        <name>dfs.replication</name>
        <value>1</value>
</property>
```

第二种方式：命令设置

bin/hadoop fs -setrep -R 1 /

6.4、负载均衡

机器与机器之间磁盘利用率不平衡是 HDFS 集群非常容易出现的情况
尤其是在 DataNode 节点出现故障或在现有的集群上增添新的 DataNode 的时候分析数据块分布和重新均衡 DataNode 上的数据分布的工具

命令：**sbin/start-balancer.sh**
sbin/start-balancer.sh -threshold 5
自动进行均衡非常慢，一天能移动的数据量在 10G-10T 的级别，很难满足超大集群的需求
原因：HDFS 集群默认不允许 balance 操作占用很大的网络带宽，这个带宽是可以调整的
hdfs dfsadmin -setBalanacerBandwidth newbandwidth
hdfs dfsadmin -setBalanacerBandwidth 10485760
该数值的单位是字节，上面的配置是 10M/s，默认是 1M/s
另外，也可以在 hdfs-site.xml 配置文件中进行设置：

```
<property>
<name>dfs.balance.bandwidthPerSec</name>
<value>10485760</value>
<description> Specifies the maximum bandwidth that each datanode can utilize for the
balancing purpose in term of the number of bytes per second. </description>
</property>
```

sbin/start-balancer.sh -t 10%
机器容量最高的那个值　和　最低的那个值得差距　不能超过 10%