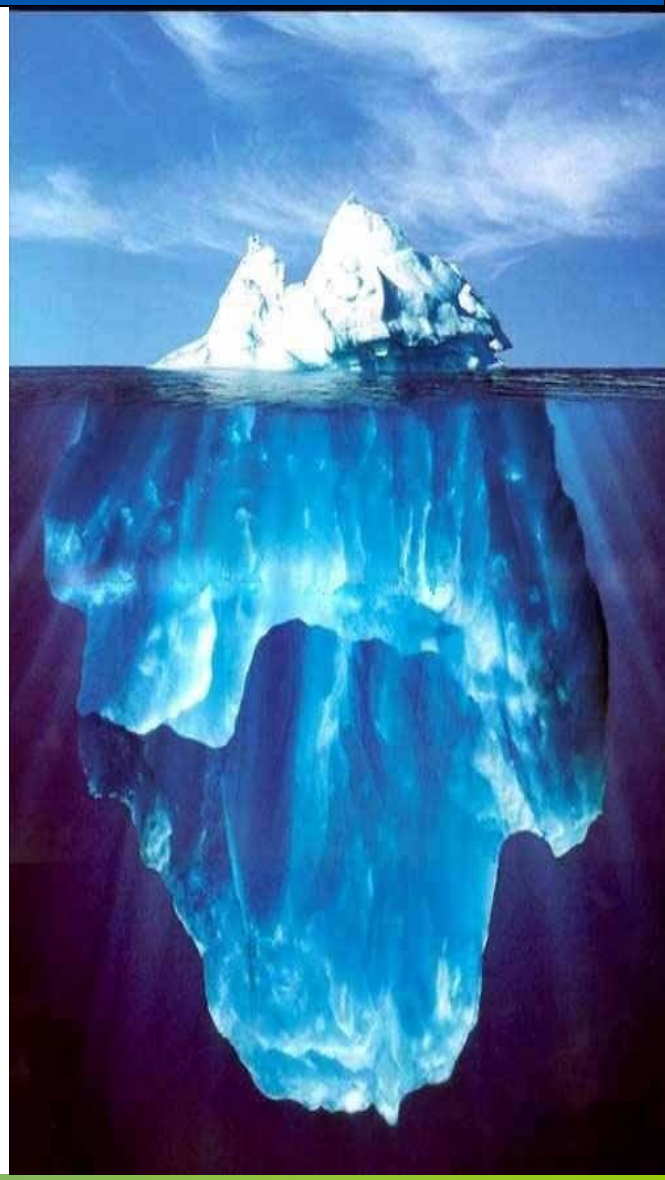
大数据技术及应用

第三章 分布式文件系统HDFS

提纲



3.1 分布式文件系统

- **3.1.1** 计算机集群结构
- **3.1.2** 分布式文件系统的结构

3.1.1计算机集群结构

•分布式文件系统把文件分布存储到多个计算机节点上，成千上万的计算机节点构成计算机集群

•与之前使用多个处理器和专用高级硬件的并行化处理装置不同的是，目前的分布式文件系统所采用的计算机集群，都是由普通硬件构成的，这就大大降低了硬件上的开销

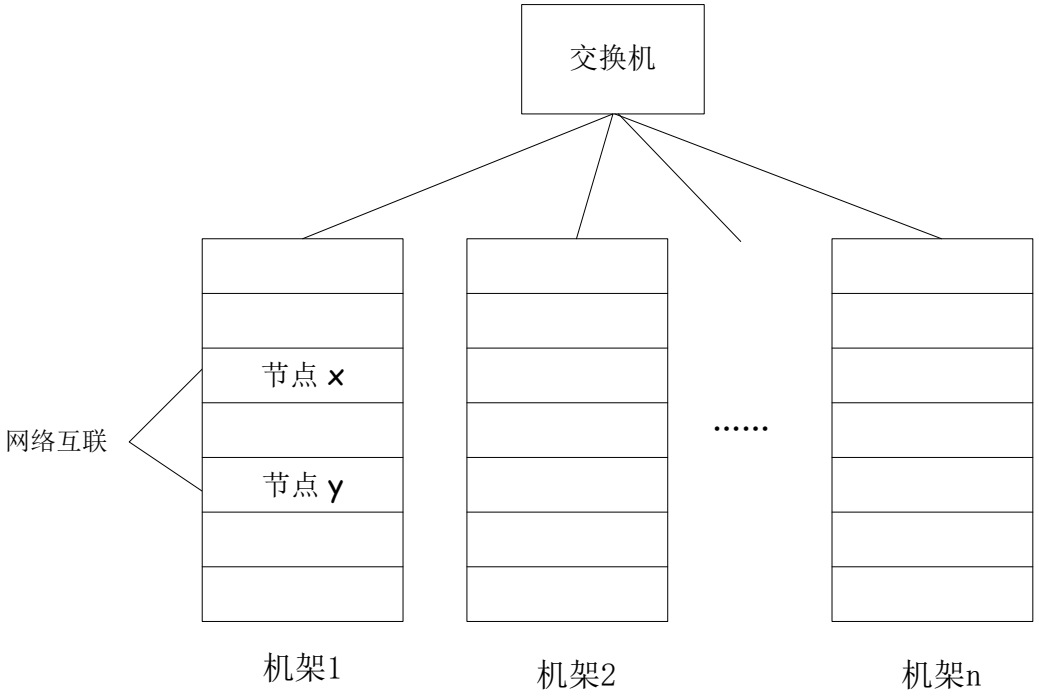


图3-1 计算机集群的基本架构

节点放机架上，每个机架放8-64个节点，通过网络互联。

3.1.2分布式文件系统的结构

- **Windows/linux：**把磁盘空间划分为每512字节一组，称为“磁盘块”，是文件系统读写操作的最小单位，BLOCK是磁盘块的整倍数

- **分布式文件系统HDFS：**块默认大小64MB。如果一个文件小于数据块的大小，只占用实际大小的空间。

3.1.2分布式文件系统的结构

分布式文件系统在物理结构上是由计算机集群中的多个节点构成的，节点分为两类，一类叫“主节点”(Master Node)或称为“名称结点”(NameNode)，另一类叫“从节点”（Slave Node）或称为“数据节点”(DataNode)。

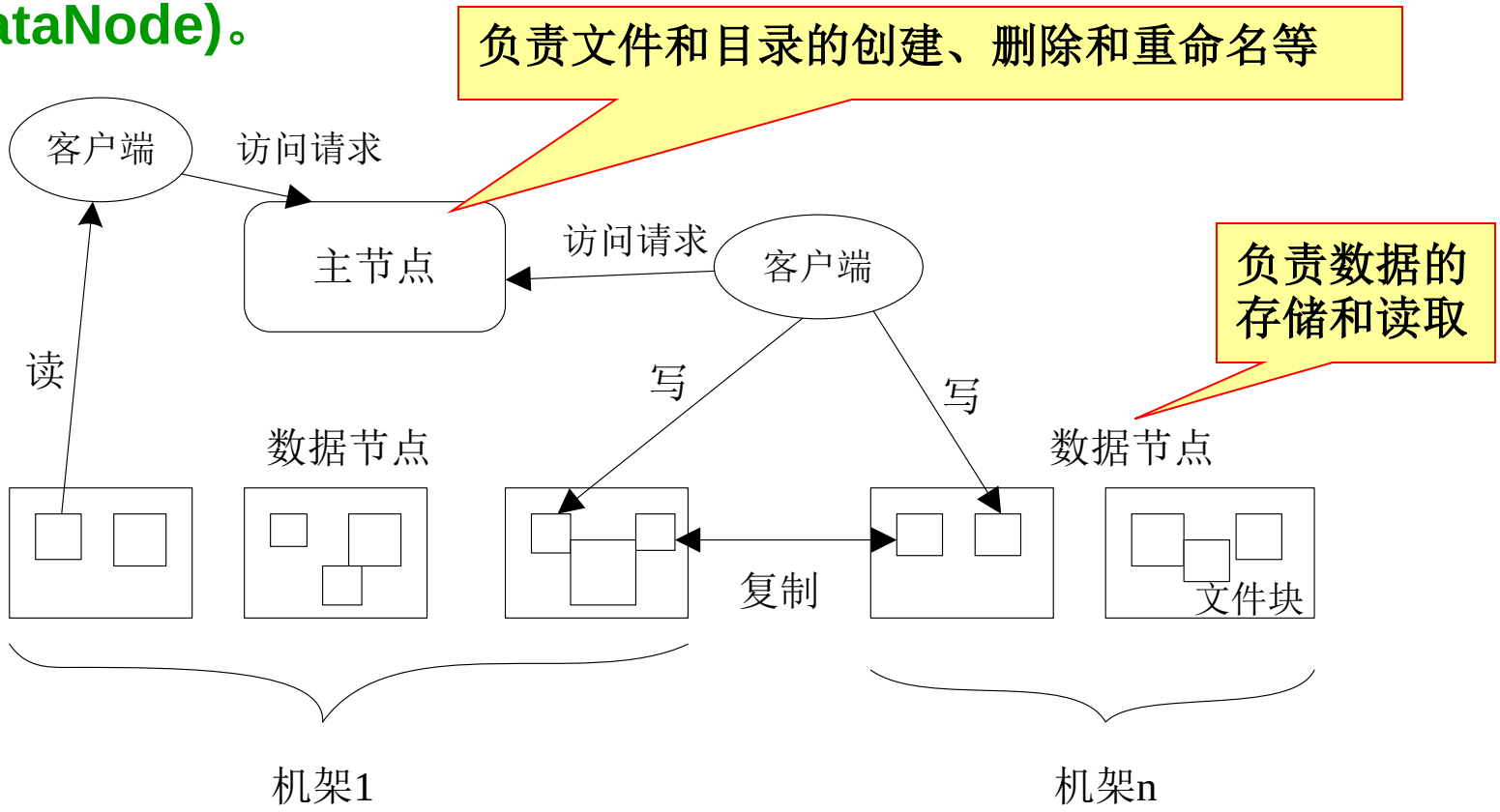


图3-2 大规模文件系统的整体结构

3.2 HDFS简介

HDFS要实现以下目标：

- 兼容廉价的硬件设备
- 流数据读写
- 大数据集
- 简单的文件模型
- 强大的跨平台兼容性

HDFS特殊的设计，在实现上述优良特性的同时，也使得自身具有一些应用局限性，主要包括以下几个方面：

- 不适合低延迟数据访问
- 无法高效存储大量小文件
- 不支持多用户写入及任意修改文件

3.3.1块

➢HDFS默认一个块64MB，一个文件被分成多个块，以块作为存储单位，块的大小远远大于普通文件系统，可以最小化寻址开销。

➢HDFS采用抽象的块概念可以带来以下几个明显的好处：

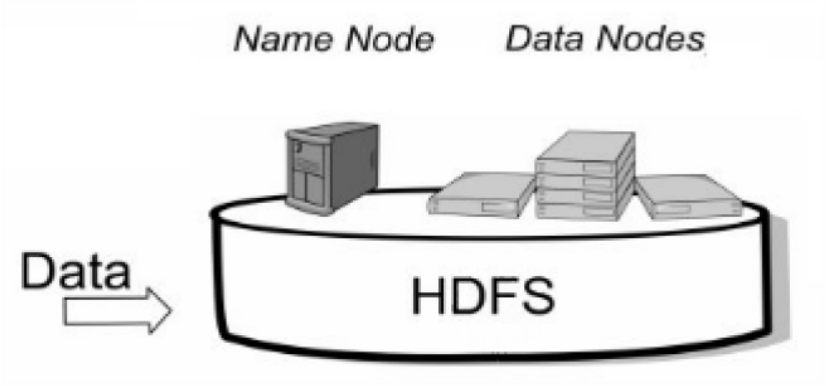
●支持大规模文件存储：文件以块为单位进行存储，一个大规模文件可以被分拆成若干个文件块，不同的文件块可以被分发到不同的节点上，因此，一个文件的大小不会受到单个节点的存储容量的限制，可以远远大于网络中任意节点的存储容量

●简化系统设计：首先，简化了存储管理，因为文件块大小是固定的，这样就可以很容易计算出一个节点可以存储多少文件块；其次，方便了元数据的管理，元数据不需要和文件块一起存储，可以由其他系统负责管理元数据

●适合数据备份：每个文件块都可以冗余存储到多个节点上，大大提高了系统的容错性和可用性。

3.3.2名称节点和数据节点

HDFS主要组件的功能



metadata

File.txt=
Blk A:
DN1, DN5, DN6

Blk B:
DN7, DN1, DN2

Blk C:
DN5, DN8,DN9

NameNode	DataNode
• 存储元数据	• 存储文件内容
•元数据保存在内存中	•文件内容保存在磁盘
• 保存文件,block， datanode之间的映射关系	•维护了block id到datanode本地文件的映射关系

3.3.2名称节点和数据节点

名称节点的数据结构

•名称节点NameNode负责管理分布式文件系统的命名空间Namespace，保存了两个核心的数据结构，即FsImage和EditLog：

FsImage维护文件系统树以及文件树中所有的文件和文件夹的元数据。

操作日志文件EditLog中记录了所有对文件的操作（创建/删除/重命名）。

•名称节点记录了每个文件中各个块所在的数据节点的位置信息。

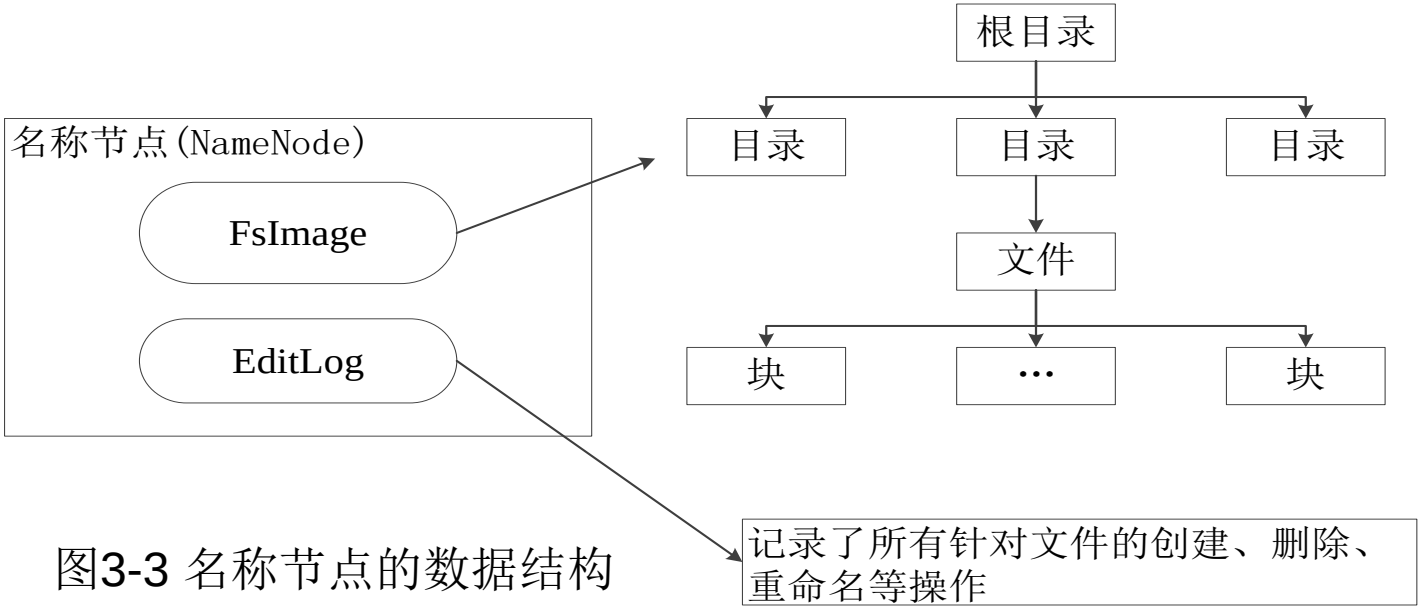


图3-3 名称节点的数据结构

3.3.2名称节点和数据节点

FsImage文件

- FsImage文件包含文件系统中所有目录和文件inode的序列化形式。每个inode是一个文件或目录的元数据的内部表示，包含信息：文件的复制等级、修改和访问时间、访问权限、块大小以及组成文件的块。对于目录，则存储修改时间、权限和配额元数据

- FsImage文件没有记录块存储在哪个数据节点。而是由名称节点把这些映射保留在内存中，当数据节点加入HDFS集群时，数据节点会把自己所包含的块列表告知给名称节点，此后会定期执行这种告知操作，以确保名称节点的块映射是最新的。

3.3.2名称节点和数据节点

名称节点的启动

- 在名称节点启动的时候，它会将**FsImage**文件中的内容加载到内存中，之后再执行**EditLog**文件中的各项操作，使内存中的元数据和实际的同步，存在内存中的元数据支持客户端的读操作。
- 一旦在内存中成功建立文件系统元数据的映射，则创建一个新的**FsImage**文件和一个空的**EditLog**文件。
- 名称节点起来之后，**HDFS**中的更新操作会重新写到**EditLog**文件中，因为**FsImage**文件一般都很大（**GB**级别的很常见），如果所有的更新操作都往**FsImage**文件中添加，这样会导致系统运行的十分缓慢，但是，如果往**EditLog**文件里面写就不会这样，因为**EditLog** 要小很多。每次执行写操作之后，且在向客户端发送成功代码之前，**edits**文件都需要同步更新。

3.3.2名称节点和数据节点

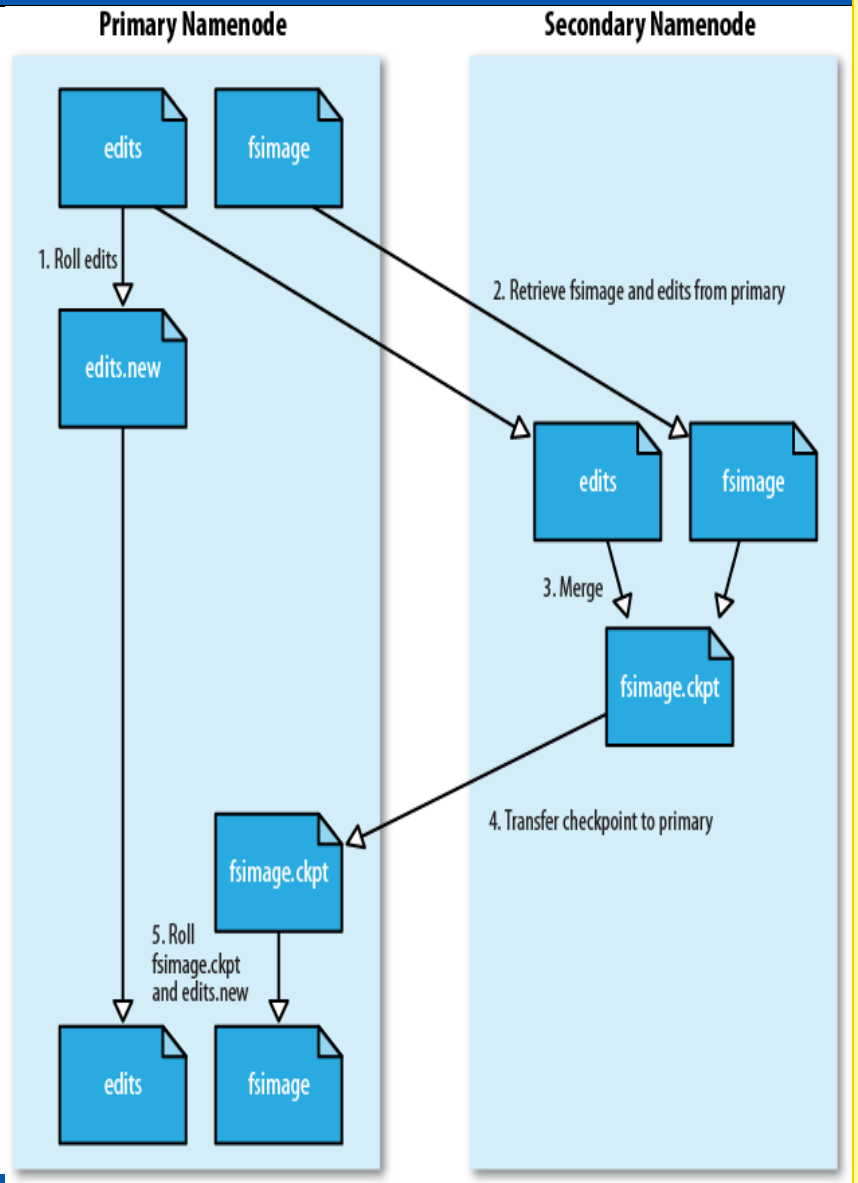
名称节点运行期间EditLog不断变大的问题

- 在名称节点运行期间，HDFS的所有更新操作都是直接写到EditLog中，久而久之， EditLog文件将会变得很大。
- 虽然这对名称节点运行时候是没有什么明显影响的，但是，当名称节点重启的时候，名称节点需要先将FsImage里面的所有内容映像到内存中，然后再一条一条地执行EditLog中的记录，当EditLog文件非常大的时候，会导致名称节点启动操作非常慢，而在这段时间内HDFS系统处于安全模式，一直无法对外提供写操作，影响了用户的使用

如何解决？答案是：SecondaryNameNode第二名称节点

第二名称节点是HDFS架构中的一个组成部分，它是用来保存名称节点中对HDFS 元数据信息的备份，并减少名称节点重启的时间。SecondaryNameNode一般是单独运行在一台机器上。

3.3.2名称节点和数据节点



SecondaryNameNode的工作情况：

（1）SecondaryNameNode会定期和NameNode通信，请求其停止使用EditLog文件，暂时将新的写操作写到一个新的文件edit.new上来，这个操作是瞬间完成，上层写日志的函数完全感觉不到差别；

（2）SecondaryNameNode通过HTTP GET方式从NameNode上获取到FsImage和EditLog文件，并下载到本地的相应目录下；（3）SecondaryNameNode将下载下来的FsImage载入到内存，然后一条一条地执行EditLog文件中的各项更新操作，使得内存中的FsImage保持最新；这个过程就是EditLog和FsImage文件合并；

（4）SecondaryNameNode执行完（3）操作之后，会通过post方式将新的FsImage文件发送到NameNode节点上；

（5）NameNode将从SecondaryNameNode接收到的新的FsImage替换旧的FsImage文件，同时将edit.new替换EditLog文件，通过这个过程EditLog就变小了

3.3.2名称节点和数据节点

数据节点（DataNode）

•数据节点是分布式文件系统HDFS的工作节点，负责数据的存储和读取，会根据客户端或者是名称节点的调度来进行数据的存储和检索，并且向名称节点定期发送自己所存储的块的列表。

•每个数据节点中的数据会被保存在各自节点的本地Linux文件系统中。

3.4 HDFS体系结构

3.4.1 HDFS体系结构概述

3.4.2 HDFS命名空间管理

3.4.3 通信协议

3.4.4 客户端

3.4.5 HDFS体系结构的局限性

3.4.1HDFS体系结构概述

HDFS采用了主从（Master/Slave）结构模型，一个HDFS集群包括一个名称节点NameNode和若干个数据节点DataNode。名称节点作为中心服务器，负责管理文件系统的命名空间及客户端对文件的访问。集群中的数据节点一般是一个节点运行一个数据节点进程，负责处理文件系统客户端的读/写请求，在名称节点的统一调度下进行数据块的创建、删除和复制等操作。每个数据节点的数据实际上是保存在本地Linux文件系统中的。

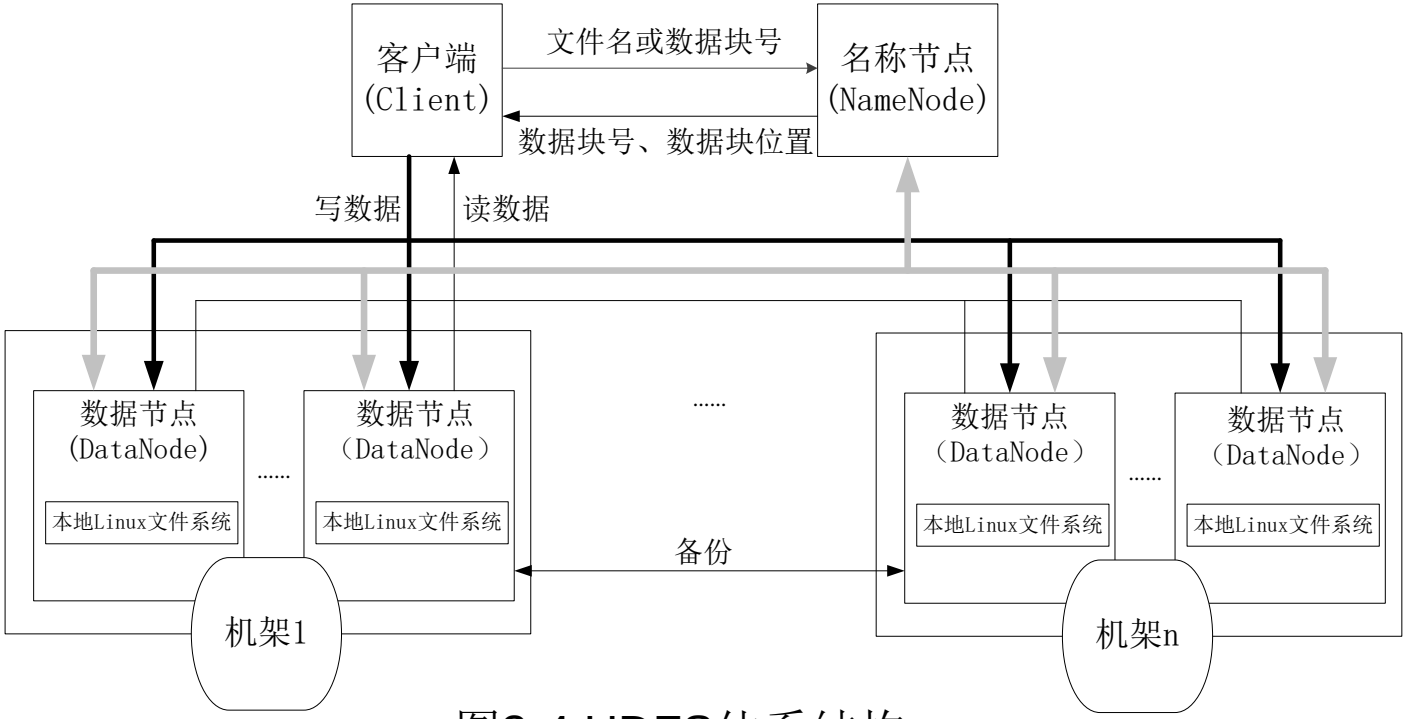


图3-4 HDFS体系结构

3.4.2HDFS命名空间管理

- HDFS的命名空间包含目录、文件和块。
- 在HDFS1.0体系结构中，在整个HDFS集群中只有一个命名空间，并且只有唯一一个名称节点，该节点负责对这个命名空间进行管理。
- HDFS使用的是传统的分级文件体系，因此，用户可以像使用普通文件系统一样，创建、删除目录和文件，在目录间转移文件，重命名文件等。

3.4.3通信协议

- HDFS是一个部署在集群上的分布式文件系统，因此，很多数据需要通过网络进行传输。
- 所有的HDFS通信协议都是构建在TCP/IP协议基础之上的。
- 客户端通过一个可配置的端口向名称节点主动发起TCP连接，并使用客户端协议与名称节点进行交互。
- 名称节点和数据节点之间则使用数据节点协议进行交互。
- 客户端与数据节点的交互是通过RPC（Remote Procedure Call）来实现的。在设计上，名称节点不会主动发起RPC，而是响应来自客户端和数据节点的RPC请求。

3.4.4客户端

- 客户端是用户操作HDFS最常用的方式，HDFS在部署时都提供了客户端。
- HDFS客户端是一个库，暴露了HDFS文件系统接口，这些接口隐藏了HDFS实现中的大部分复杂性。
- 严格来说，客户端并不算是HDFS的一部分。
- 客户端可以支持打开、读取、写入等常见的操作，并且提供了类似Shell的命令行方式来访问HDFS中的数据。
- 此外，HDFS也提供了Java API，作为应用程序访问文件系统的客户端编程接口。

3.4.5HDFS体系结构的局限性

HDFS只设置唯一一个名称节点，这样做虽然大大简化了系统设计，但也带来了一些明显的局限性，具体如下：

（1）命名空间的限制：名称节点是保存在内存中的，因此，名称节点能够容纳的对象（文件、块）的个数会受到内存空间大小的限制。

（2）性能的瓶颈：整个分布式文件系统的吞吐量，受限于单个名称节点的吞吐量。

（3）隔离问题：由于集群中只有一个名称节点，只有一个命名空间，因此，无法对不同应用程序进行隔离。

（4）集群的可用性：一旦这个唯一的名称节点发生故障，会导致整个集群变得不可用。

3.5 HDFS存储原理

3.5.1 冗余数据保存

3.5.2 数据存取策略

3.5.3 数据错误与恢复

3.5.1冗余数据保存

作为一个分布式文件系统，为了保证系统的容错性和可用性，HDFS采用了多副本方式对数据进行冗余存储，通常一个数据块的多个副本会被分布到不同的数据节点上，这种多副本方式具有以下几个优点：

（1）加快数据传输速度

（2）容易检查数据错误

（3）保证数据可靠性

数据块1被分别存放到数据节点A和C上

数据块2被存放在数据节点A和B上。

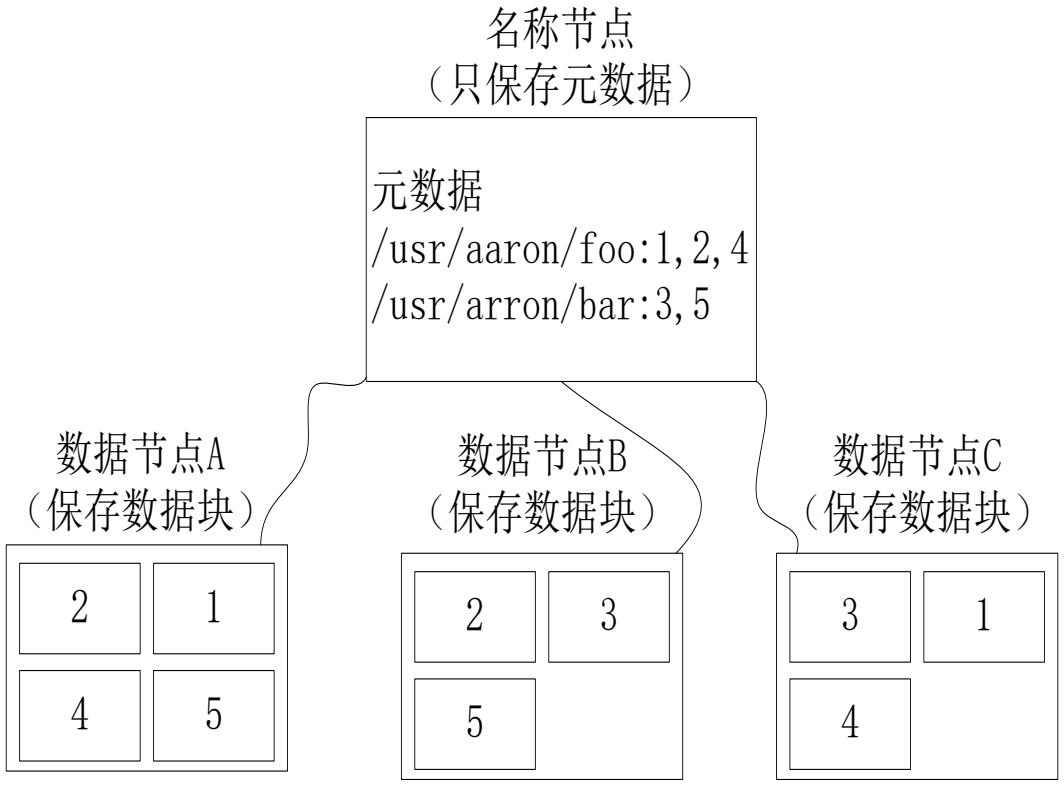
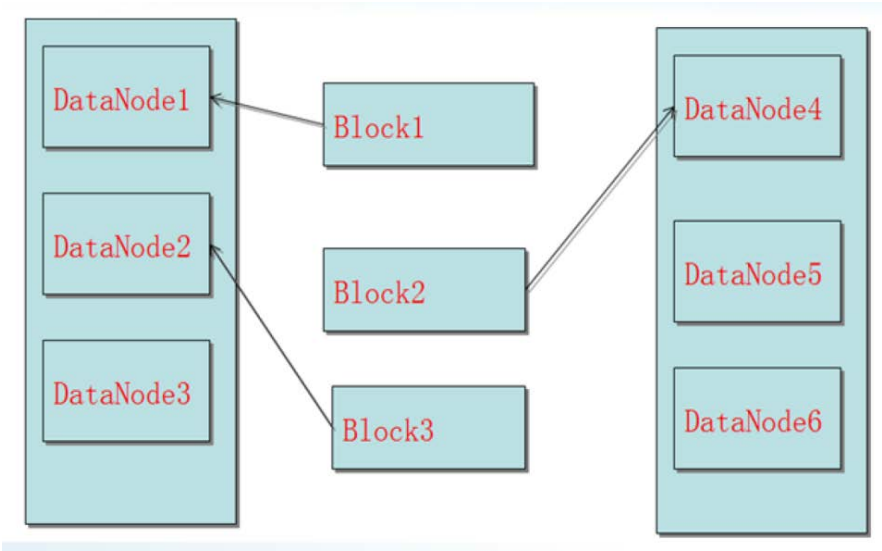


图3-5 HDFS数据块多副本存储

3.5.2数据存取策略

1.数据存放

- 第一个副本：放置在上传文件的数据节点；如果是集群外提交，则随机挑选一台磁盘不太满、CPU不太忙的节点；
- 第二个副本：放置在与第一个副本不同的机架的节点上；
- 第三个副本：与第一个副本相同机架的其他节点上；
- 更多副本：随机节点。



Block的副本放置策略

3.5.2数据存取策略

2. 数据读取

•HDFS提供了一个API可以确定一个数据节点所属的机架ID，客户端也可以调用API获取自己所属的机架ID；

•当客户端读取数据时，从名称节点获得数据块不同副本的存放位置列表，列表中包含了副本所在的数据节点，可以调用API来确定客户端和这些数据节点所属的机架ID，当发现某个数据块副本对应的机架ID和客户端对应的机架ID相同时，就优先选择该副本读取数据，如果没有发现，就随机选择一个副本读取数据。

3.5.2数据存取策略

3. 数据复制

- **HDFS采用流水线复制的策略。**
- **HDFS将文件写入本地，分成多个块。每个块都向名称节点发起请求，得到数据节点列表。连接第一个节点，写入；请求连接第二节点，写入……**

3.5.3数据错误与恢复

HDFS具有较高的容错性，可以兼容廉价的硬件，它把硬件出错看作一种常态，而不是异常，并设计了相应的机制检测数据错误和进行自动恢复，主要包括以下几种情形：名称节点出错、数据节点出错和数据出错。

1. 名称节点出错

名称节点保存了所有的元数据信息，其中，最核心的两大数据结构是FsImage和Editlog，如果这两个文件发生损坏，那么整个HDFS实例将失效。因此，HDFS设置了备份机制，把这些核心文件同步复制到备份服务器SecondaryNameNode上。当名称节点出错时，就可以根据备份服务器SecondaryNameNode中的FsImage和Editlog数据进行恢复。

3.5.3数据错误与恢复

2. 数据节点出错

•每个数据节点会定期向名称节点发送“心跳”信息，向名称节点报告自己的状态；

•当数据节点发生故障，或者网络发生断网时，名称节点就无法收到来自一些数据节点的心跳信息，这时，这些数据节点就会被标记为“宕机”，节点上面的所有数据都会被标记为“不可读”，名称节点不会再给它们发送任何I/O请求；

•这时，有可能出现一种情形，即由于一些数据节点的不可用，会导致一些数据块的副本数量小于冗余因子；

•名称节点会定期检查这种情况，一旦发现某个数据块的副本数量小于冗余因子，就会启动数据冗余复制，为它生成新的副本；

•HDFS和其它分布式文件系统的最大区别就是可以调整冗余数据的位置。

3.5.3数据错误与恢复

3. 数据出错

•网络传输和磁盘错误等因素，都会造成数据错误；

•客户端在读取到数据后，会采用**md5**和**sha1**对数据块进行校验，以确定读取到正确的数据；

•在文件被创建时，客户端就会对每一个文件块进行信息摘录，并把这些信息写入到同一个路径的隐藏文件里面；

•当客户端读取文件的时候，会先读取该信息文件，然后，利用该信息文件对每个读取的数据块进行校验，如果校验出错，客户端就会请求到另外一个数据节点读取该文件块，并且向名称节点报告这个文件块有错误，名称节点会定期检查并且重新复制这个块。

3.6 HDFS数据读写过程

- **3.6.1 读数据的过程**
- **3.6.2 写数据的过程**

3.6 HDFS数据读写过程

读取文件

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.FSDataInputStream;

public class Chapter3 {
    public static void main(String[] args) {
        try {
            Configuration conf = new Configuration();
            FileSystem fs = FileSystem.get(conf);
            Path filename = new Path("hdfs://localhost:9000/user/hadoop/
test.txt");
            FSDataInputStream is = fs.open(filename);
            BufferedReader d = new BufferedReader(new InputStreamReader(i
s));
            String content = d.readLine(); //读取文件一行
            System.out.println(content);
            d.close(); //关闭文件
            fs.close(); //关闭hdfs
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

3.6 HDFS数据读写过程

写入文件

```
import org.apache.hadoop.conf.Configuration
import org.apache.hadoop.fs.FileSystem
import org.apache.hadoop.fs.FSDataOutputStream
import org.apache.hadoop.fs.Path

public class Chapter3 {
    public static void main(String[] args) {
        try {
            Configuration conf = new Configuration();
            FileSystem fs = FileSystem.get(conf);
            byte[] buff = "Hello world".getBytes(); // 要写入的内容
            String filename = " hdfs://localhost:9000/user/hadoop/test.txt "; //要写入的文件名
            FSDataOutputStream os = fs.create(new Path(filename));
            os.write(buff,0,buff.length);
            System.out.println("Create:"+ filename);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

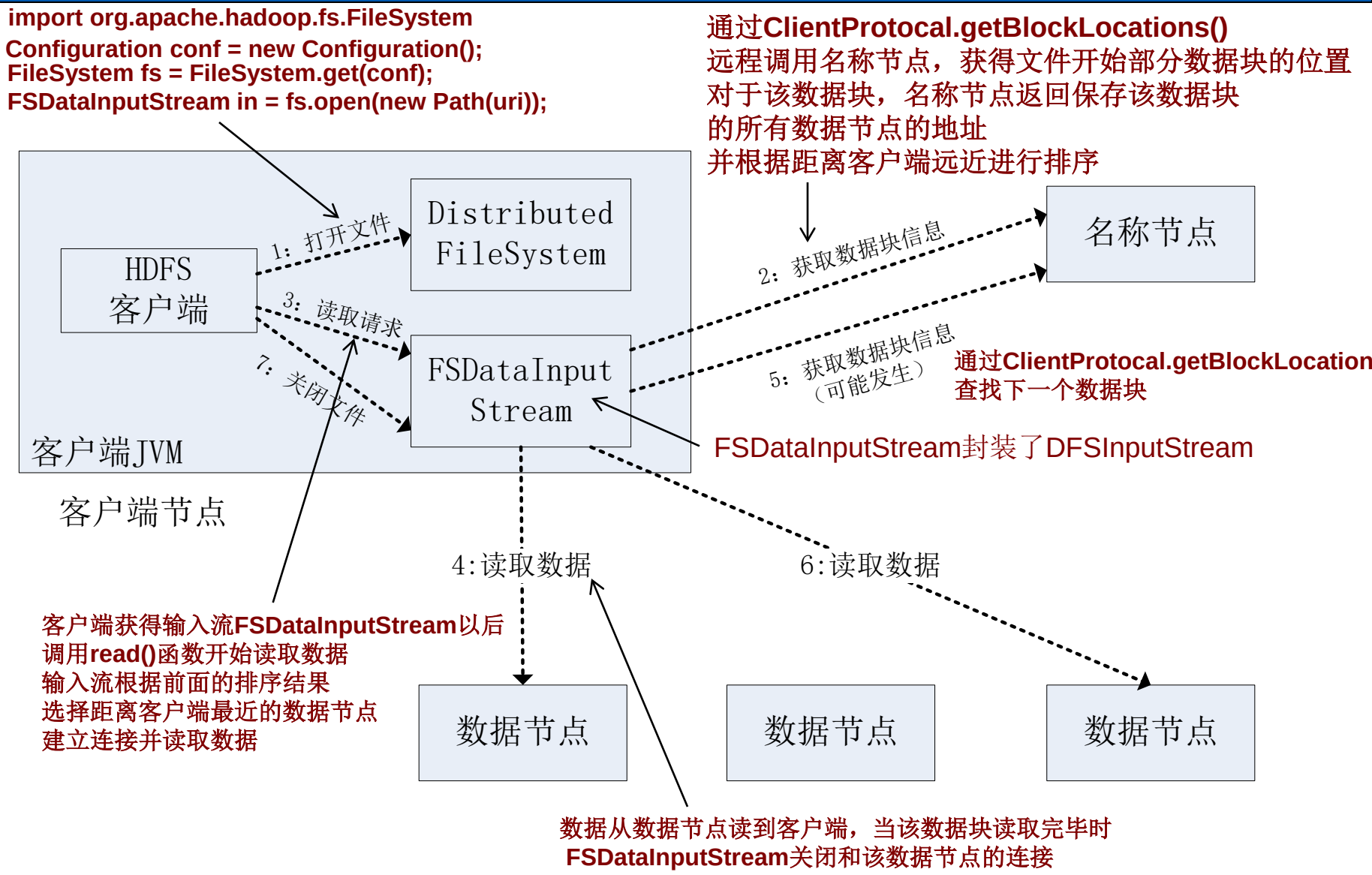
3.6 HDFS数据读写过程

- **FileSystem**是一个通用文件系统的抽象基类，可以被分布式文件系统继承，所有可能使用**Hadoop**文件系统的代码，都要使用这个类；
- **Hadoop**为**FileSystem**这个抽象类提供了多种具体实现；
- **DistributedFileSystem**就是**FileSystem**在**HDFS**文件系统中的具体实现；
- **FileSystem**的**open()**方法返回的是一个输入流**FSDataInputStream**对象，在**HDFS**文件系统中，具体的输入流就是**DFSInputStream**；**FileSystem**中的**create()**方法返回的是一个输出流**FSDataOutputStream**对象，在**HDFS**文件系统中，具体的输出流就是**DFSOutputStream**。

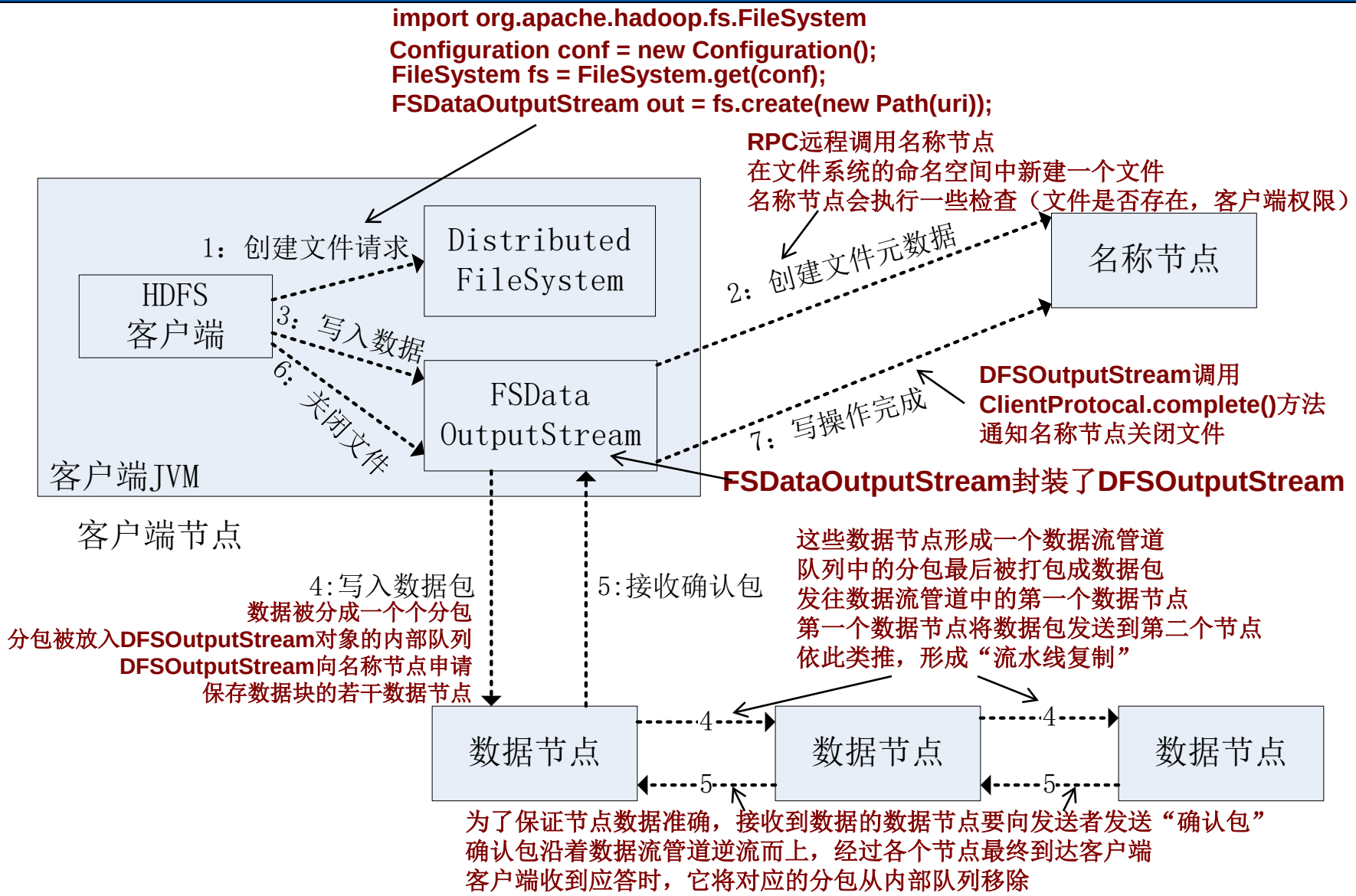
```
Configuration conf = new Configuration();
FileSystem fs = FileSystem.get(conf);
FSDataInputStream in = fs.open(new Path(uri));
FSDataOutputStream out = fs.create(new Path(uri));
```

注意：创建一个Configuration对象时，其构造方法会默认加载工程项目下两个配置文件，分别是hdfs-site.xml以及core-site.xml，这两个文件中会有访问HDFS所需的参数值，主要是fs.defaultFS，指定了HDFS的地址（比如hdfs://localhost:9000），有了这个地址客户端就可以通过这个地址访问HDFS了

3.6.1读数据的过程


import org.apache.hadoop.fs.FileSystem
Configuration conf = new Configuration();
FileSystem fs = FileSystem.get(conf);
FSDataInputStream in = fs.open(new Path(uri));
通过ClientProtocal.getBlockLocations()
远程调用名称节点，获得文件开始部分数据块的位置
对于该数据块，名称节点返回保存该数据块
的所有数据节点的地址
并根据距离客户端远近进行排序
HDFS
客户端
Distributed
FileSystem
名称节点
1：打开文件
2：获取数据块信息
3：读取请求
5：获取数据块信息
（可能发生）
7：关闭文件
FSDataInput
Stream
通过ClientProtocal.getBlockLocation
查找下一个数据块
FSDataInputStream封装了DFSInputStream
客户端JVM
客户端节点
4:读取数据
6:读取数据
客户端获得输入流FSDataInputStream以后
调用read()函数开始读取数据
输入流根据前面的排序结果
选择距离客户端最近的数据节点
建立连接并读取数据
数据节点
数据节点
数据节点
数据从数据节点读到客户端，当该数据块读取完毕时
FSDataInputStream关闭和该数据节点的连接

3.6.2写数据的过程


import org.apache.hadoop.fs.FileSystem
Configuration conf = new Configuration();
FileSystem fs = FileSystem.get(conf);
FSDataOutputStream out = fs.create(new Path(uri));
RPC远程调用名称节点
在文件系统的命名空间中新建一个文件
名称节点会执行一些检查（文件是否存在，客户端权限）
1：创建文件请求
Distributed FileSystem
名称节点
HDFS 客户端
2：创建文件元数据
3：写入数据
6：关闭文件
FSData OutputStream
7：写操作完成
DFSOutputStream调用
ClientProtocal.complete()方法
通知名称节点关闭文件
客户端JVM
FSDataOutputStream封装了DFSOutputStream
客户端节点
这些数据节点形成一个数据流管道
队列中的分包最后被打包成数据包
发往数据流管道中的第一个数据节点
第一个数据节点将数据包发送到第二个节点
依此类推，形成“流水线复制”
4:写入数据包
数据被分成一个个分包
分包被放入DFSOutputStream对象的内部队列
DFSOutputStream向名称节点申请
保存数据块的若干数据节点
5:接收确认包
数据节点
4
数据节点
4
数据节点
5
5
为了保证节点数据准确，接收到数据的数据节点要向发送者发送“确认包”
确认包沿着数据流管道逆流而上，经过各个节点最终到达客户端
客户端收到应答时，它将对应的分包从内部队列移除

3.7 HDFS编程实践

Hadoop提供了关于HDFS在Linux操作系统上进行文件操作的常用Shell命令以及Java API。同时还可以利用Web界面查看和管理Hadoop文件系统。

备注：Hadoop安装成功后，已经包含HDFS和MapReduce，不需要额外安装。而HBase等其他组件，则需要另外下载安装。

在学习HDFS编程实践前，我们需要启动Hadoop。执行如下命令：

```
$ cd /usr/local/hadoop
$ ./bin/hdfs namenode -format    #格式化hadoop的hdfs文件系统
$ ./sbin/start-dfs.sh #启动hadoop
```

3.7.1HDFS常用命令

HDFS有很多shell命令，

fs命令：查看HDFS文件系统的目录结构、上传和下载数据、创建文件等。

该命令的用法为：

hadoop fs [genericOptions] [commandOptions]

备注：Hadoop中有三种Shell命令方式：

•hadoop fs适用于任何不同的文件系统，比如本地文件系统和HDFS文件系统；

hadoop dfs只能适用于HDFS文件系统；

；hdfs dfs跟hadoop dfs的命令作用一样，也只能适用于HDFS文件系统。

3.7.1HDFS常用命令

实例：

hadoop fs -ls <path>:显示<path>指定的文件的详细信息

hadoop fs -mkdir <path>:创建<path>指定的文件夹

```
administrator@ubuntu:~/hadoop/hadoop-1.2.1/bin$ ./hadoop fs -mkdir hdfs://127.0.0.1:9000/tempDir

administrator@ubuntu:~/hadoop/hadoop-1.2.1/bin$ ./hadoop fs -ls hdfs://127.0.0.1:9000/
Found 4 items
drwxr-xr-x   - administrator supergroup          0 2015-04-26 16:30 /hbase
drwxr-xr-x   - administrator supergroup          0 2015-04-26 15:44 /home
drwxr-xr-x   - administrator supergroup          0 2015-04-26 16:46 /tempDir
drwxr-xr-x   - administrator supergroup          0 2015-04-26 15:55 /user
```

3.7.1HDFS常用命令

实例：

hadoop fs -cat <path>:将<path>指定的文件的内容输出到标准输出（stdout）

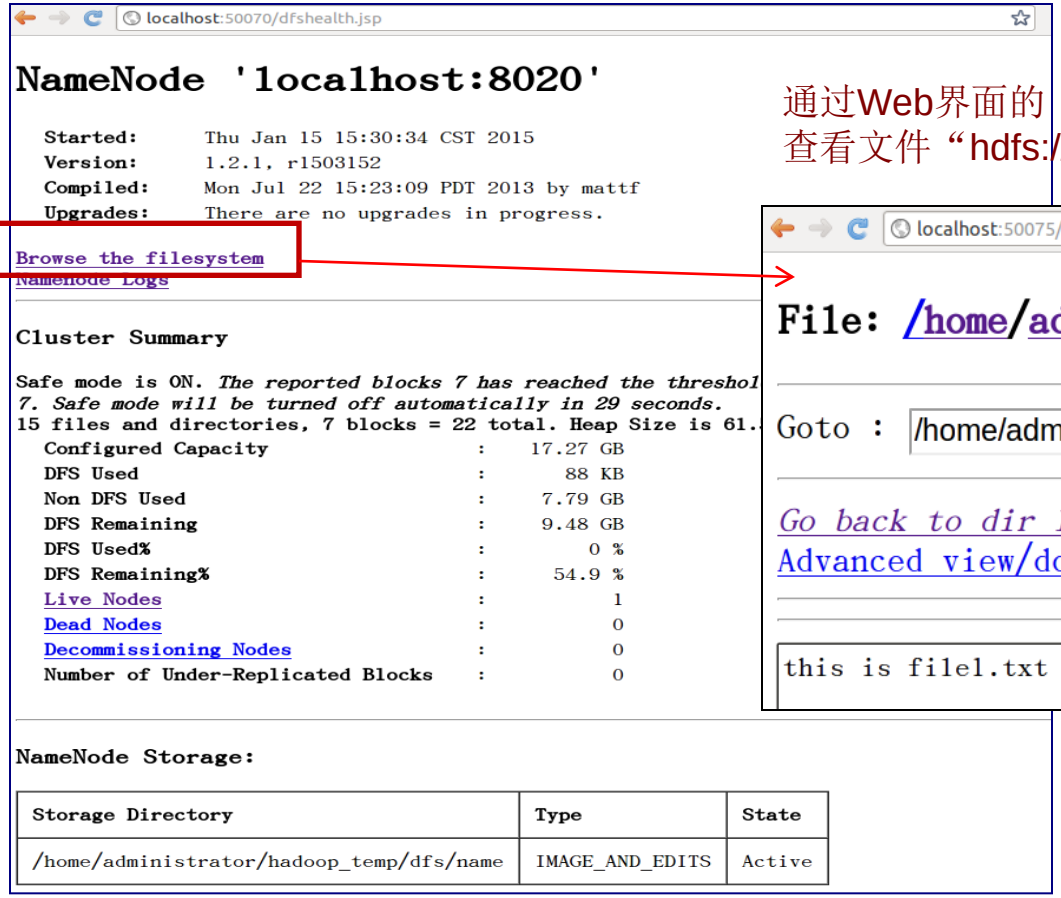
hadoop fs -copyFromLocal <localsrc> <dst>:将本地源文件<localsrc>复制到路径<dst>指定的文件或文件夹中

```
administrator@ubuntu:~/hadoop/hadoop-1.2.1/bin$ ./hadoop fs -copyFromLocal /home/administrator/t
empfile/* hdfs://127.0.0.1:9000/tempDir
administrator@ubuntu:~/hadoop/hadoop-1.2.1/bin$ ./hadoop fs -ls  hdfs://127.0.0.1:9000/tempDir/
Found 8 items
-rw-r--r--   1 administrator supergroup         18 2015-04-26 16:48 /tempDir/file1.txt
-rw-r--r--   1 administrator supergroup         14 2015-04-26 16:48 /tempDir/file1.txt~
-rw-r--r--   1 administrator supergroup         18 2015-04-26 16:48 /tempDir/file2.txt
-rw-r--r--   1 administrator supergroup         18 2015-04-26 16:48 /tempDir/file3.txt
-rw-r--r--   1 administrator supergroup         18 2015-04-26 16:48 /tempDir/file4.abc
-rw-r--r--   1 administrator supergroup         18 2015-04-26 16:48 /tempDir/file5.abc
-rw-r--r--   1 administrator supergroup         17 2015-04-26 16:48 /tempDir/testFile
-rw-r--r--   1 administrator supergroup          0 2015-04-26 16:48 /tempDir/testFile~
administrator@ubuntu:~/hadoop/hadoop-1.2.1/bin$ ./hadoop fs -cat  hdfs://127.0.0.1:9000/tempDir/
*
this is file1.txt
this is file1
this is file2.txt
this is file3.txt
this is file4.abc
this is file5.abc
welcome to DBLab
```

3.7.2HDFS的Web界面

在配置好Hadoop集群之后，可以通过浏览器登录

“http://[*NameNodeIP*]:50070”访问HDFS文件系统



通过Web界面的“Browse the filesystem”
查看文件“hdfs://localhost/home/administrator/tempfile/file1.txt”

3.7.3HDFS常用Java API及应用实例

利用Java API与HDFS进行交互

实例：利用hadoop的java api检测伪分布式文件系统HDFS上是否有某文件？

准备工作：在系统中安装和配置Eclipse

第一步:放置配置文件到当前工程下面（eclipse工作目录的bin文件夹下面）

第二步：编写实现代码

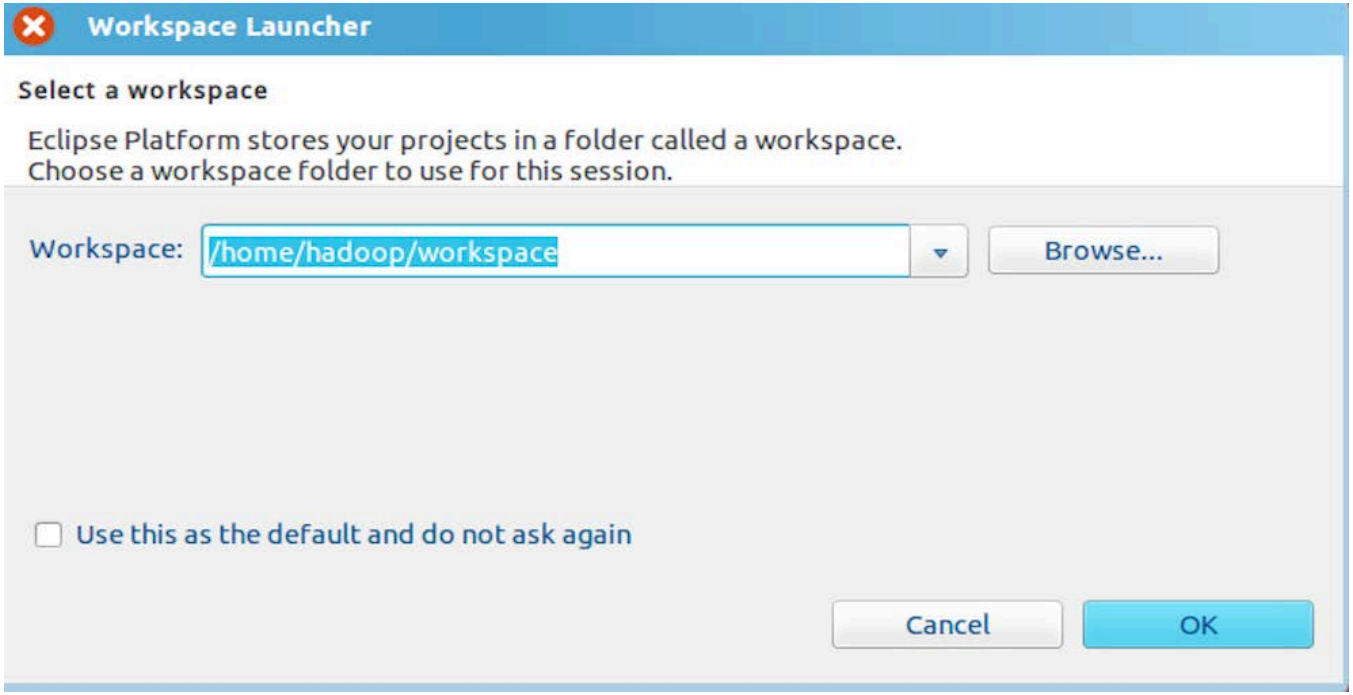
3.7.3HDFS常用Java API及应用实例

利用Java API进行交互，可以使用软件Eclipse编写Java程序。

（1）在虚拟机中安装Eclipse

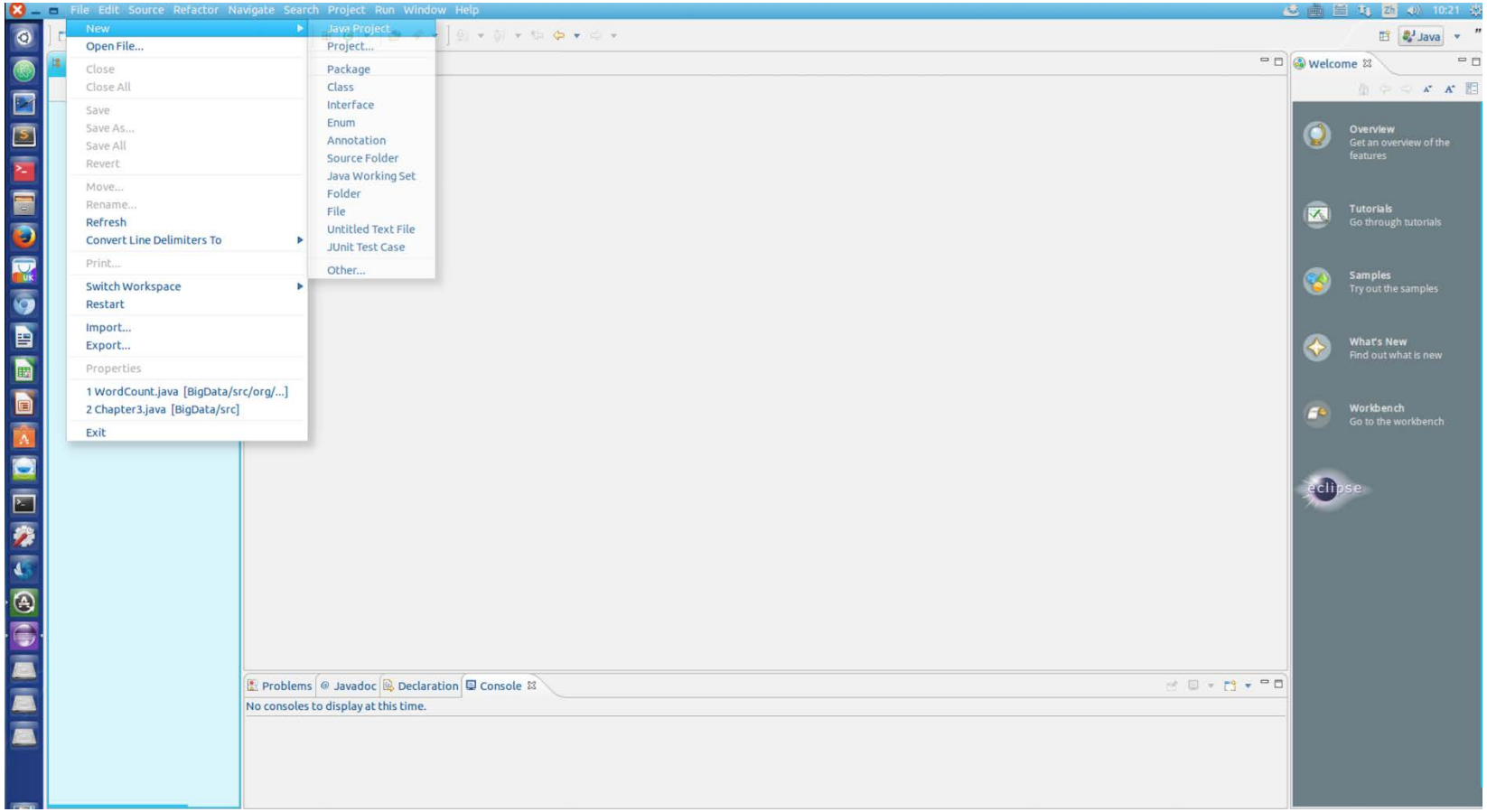
（2）在Eclipse创建项目

第一次打开Eclipse,需要填写workspace(工作空间)，用来保存程序所在的位置，这里按照默认，不需要改动，如下图



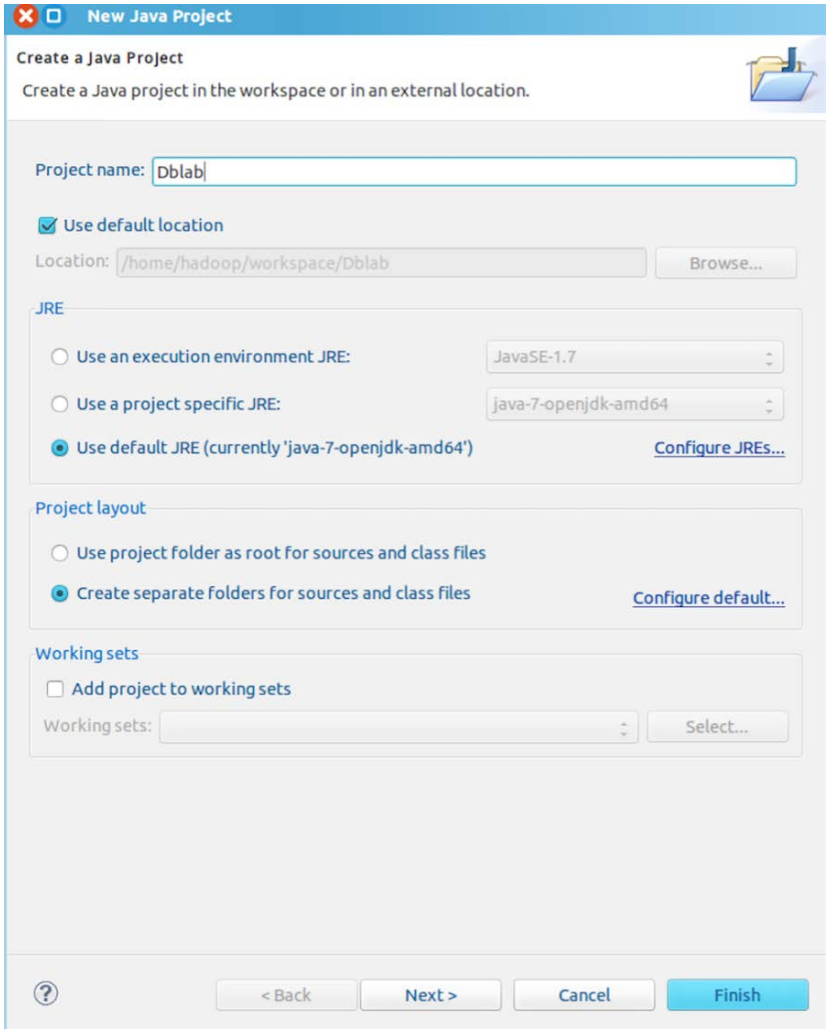
3.7.3HDFS常用Java API及应用实例

点击“OK”按钮，进入Eclipse软件。开始创建项目，选择顶部菜单File—>New—>Java Project,如下图



3.7.3HDFS常用Java API及应用实例

输入项目名称 “**Dblab**”，其他不用改动，点击“**Finish**”按钮即可。



3.7.3HDFS常用Java API及应用实例

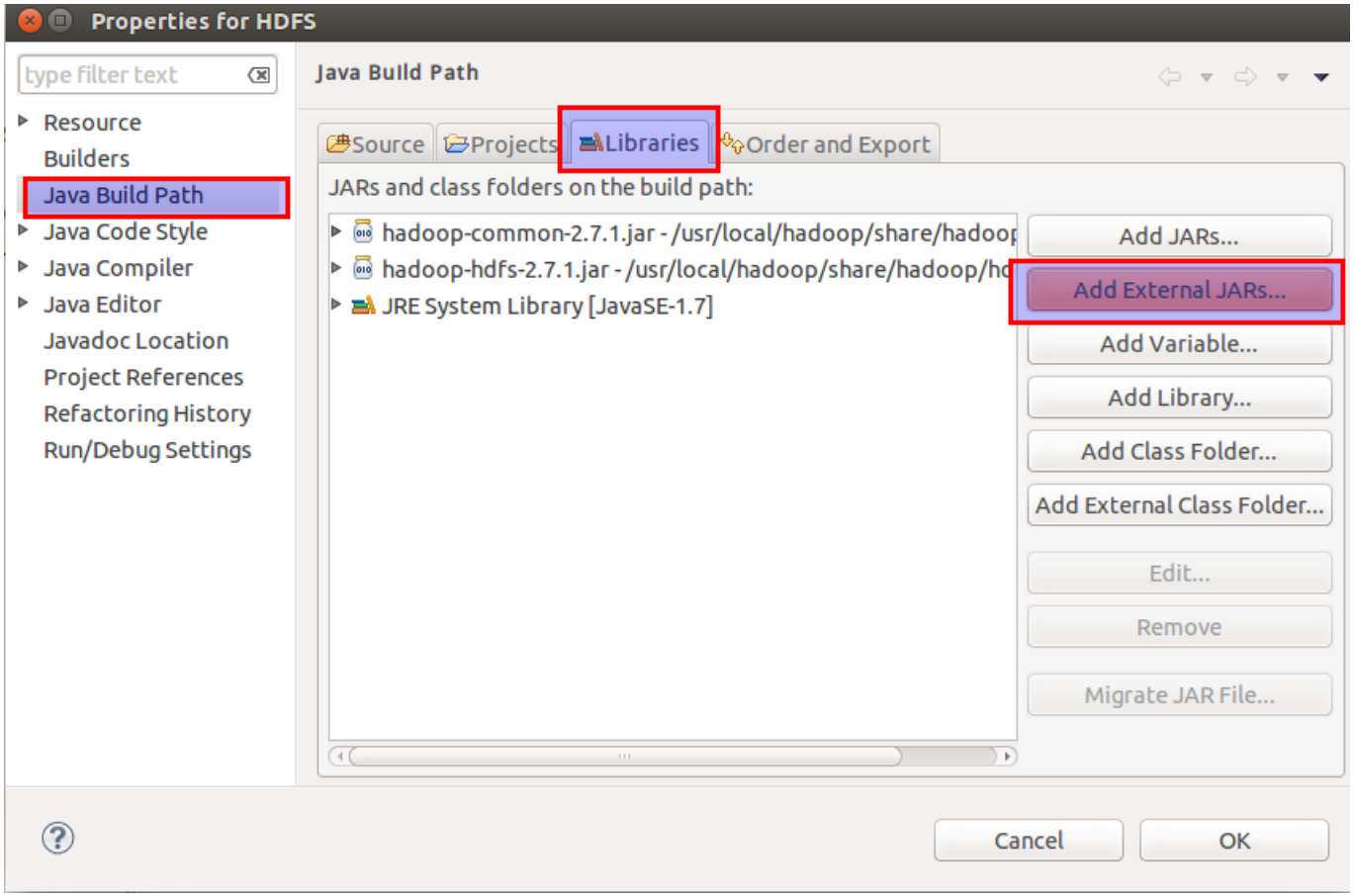
为项目加载所需要用到的jar包

如何获取jar包

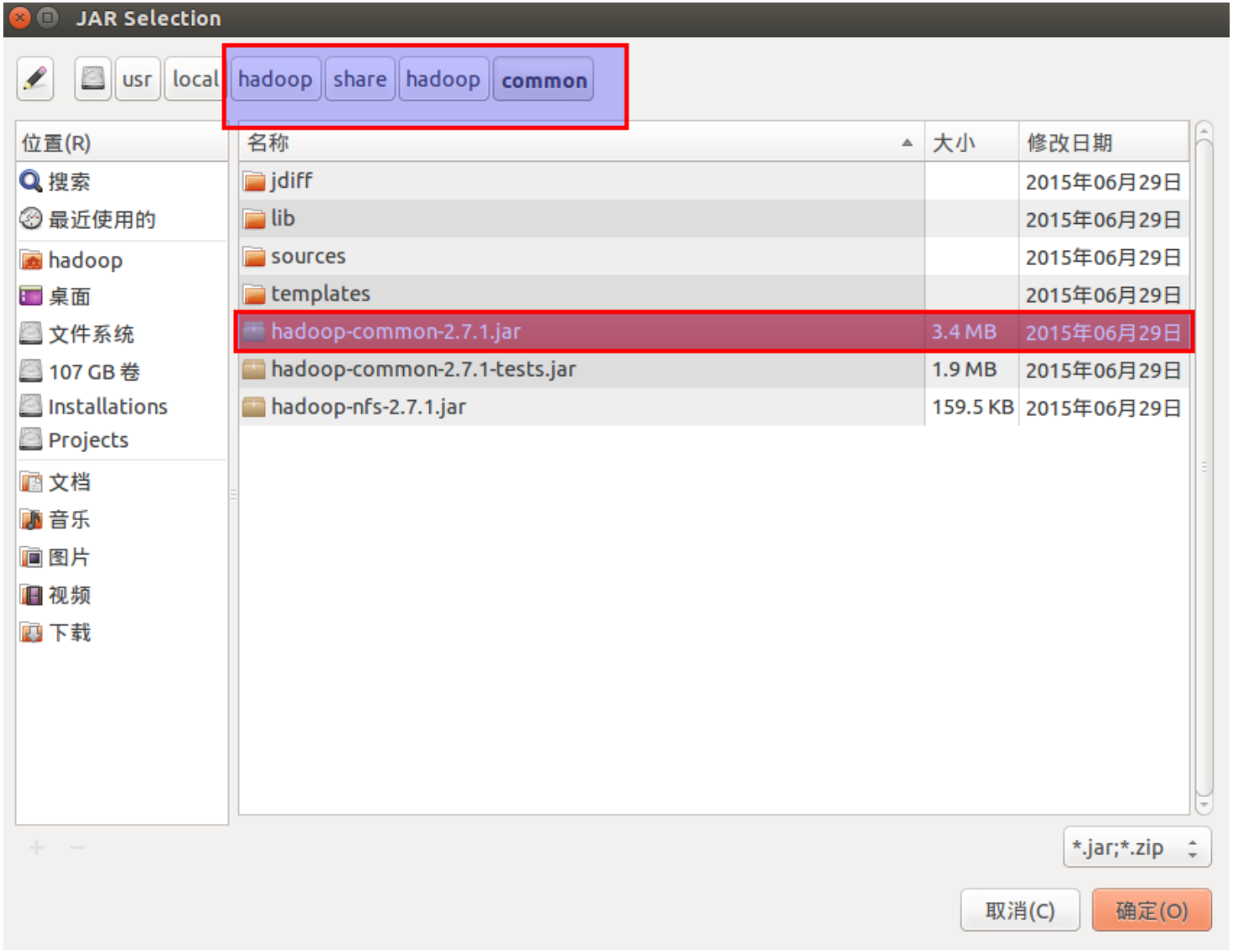
Java API所在的jar包都在已经安装好的hadoop文件夹里，路径：/usr/local/hadoop/share/hadoop(如果安装的hadoop不在此目录，请找到jar包所在的文件夹)

3.7.3HDFS常用Java API及应用实例

在所在项目中加载jar包,具体操作如下：在所选的Eclipse项目(Dblab)上右键点击—>弹出菜单中选择>Properties—>Java Build Path—>Libraries—>Add External JARS



3.7.3HDFS常用Java API及应用实例



3.7.3HDFS常用Java API及应用实例

编程实例

利用Hadoop 的Java API检测伪分布式文件系统HDFS上是否存在某个文件？

下面编写一个简单的程序来测试伪分布式文件系统HDFS上是否存在input.txt文件？

第一步:放置配置文件到当前工程下面
需要把集群上的core-site.xml和hdfs-site.xml(这两文件存在/hadoop/etc/hadoop目录下)放到当前工程项目下，即eclipse工作目录的bin文件夹下面。

```
$ Wrong FS: hdfs://localhost:9000/user/hadoop/input/input.txt, expected: file:///
```

3.7.3HDFS常用Java API及应用实例

第二步：编写实现代码

```
import org.apache.hadoop.conf.Configuration
import org.apache.hadoop.fs.FileSystem
import org.apache.hadoop.fs.Path

public class Chapter3 {
    public static void main(String[] args) {
        try {
            String filename = "hdfs://localhost:9000/user/hadoop/test.txt";

            Configuration conf = new Configuration();

            FileSystem fs = FileSystem.get(conf);
            if(fs.exists(new Path(filename))){
                System.out.println("文件存在");
            }else{
                System.out.println("文件不存在");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

本章小结

- HDFS开源实现了GFS，可以利用由廉价硬件构成的计算机集群。
- HDFS具有兼容廉价的硬件设备、流数据读写、大数据集、简单的文件模型、强大的跨平台兼容性等特点。但是，也要注意到，HDFS也有自身的局限性，比如不适合低延迟数据访问、无法高效存储大量小文件和不支持多用户写入及任意修改文件等。
- 块是HDFS核心的概念，一个大的文件会被拆分成很多个块。HDFS采用抽象的块概念，支持大规模文件存储、简化系统设计、适合数据备份等优点。
- HDFS采用了主从（Master/Slave）结构模型，一个HDFS集群包括一个名称节点和若干个数据节点。名称节点负责管理分布式文件系统的命名空间；数据节点是分布式文件系统HDFS的工作节点，负责数据的存储和读取。
- HDFS采用了冗余数据存储，增强了数据可靠性，加快了数据传输速度。HDFS还采用了相应的数据存放、数据读取和数据复制策略，来提升系统整体读写响应性能。HDFS把硬件出错看作一种常态，设计了错误恢复机制。
- HDFS的数据读写过程以及HDFS编程实践方面的相关知识。