北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

分布式文件系统
HDFS

大数据处理技术

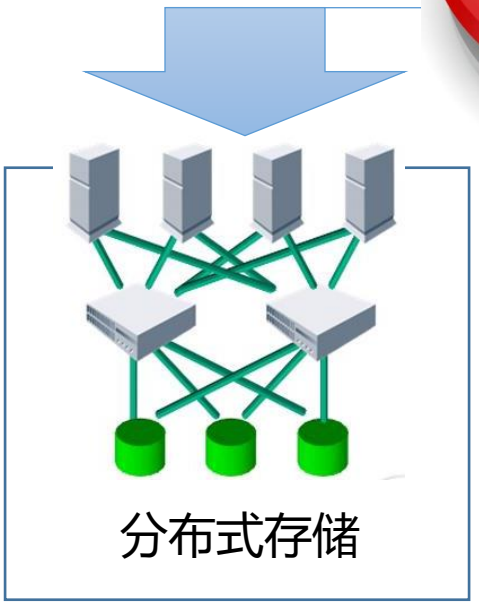
计算机学院

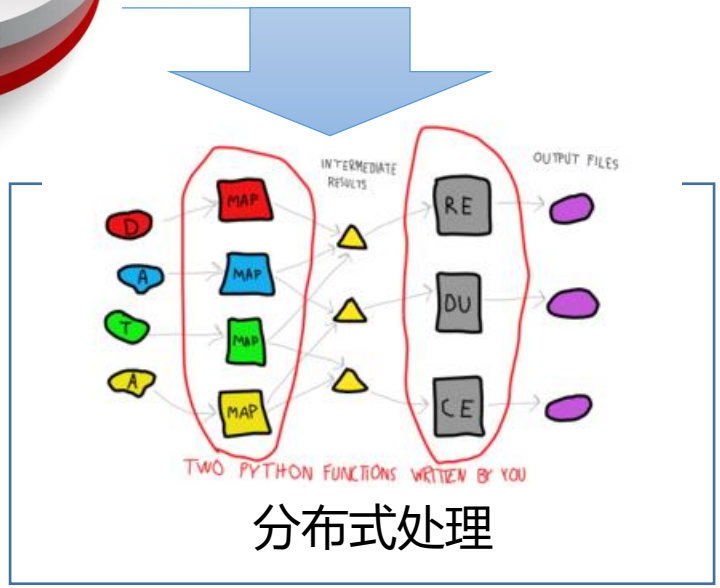
课程提纲

- HDFS简介
- HDFS相关概念
- HDFS体系结构
- HDFS存储原理
- HDFS数据读写过程
- HDFS编程实践

大数据技术

- 大数据两大核心技术



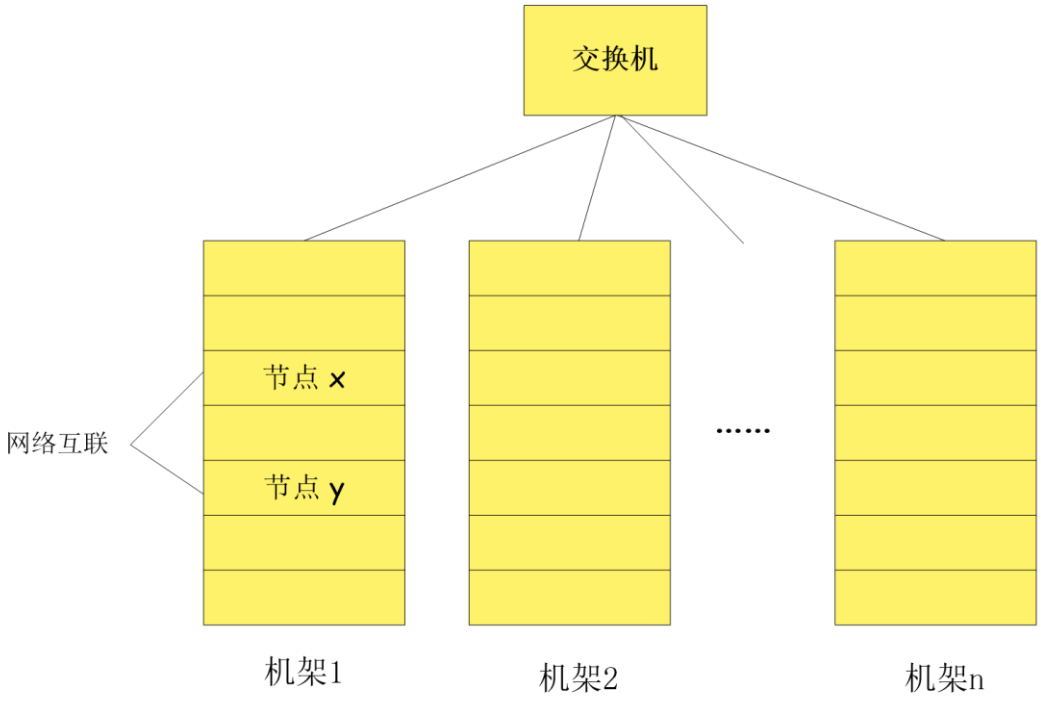


解决海量数据的存储问题

解决海量数据的处理问题

分布式文件系统

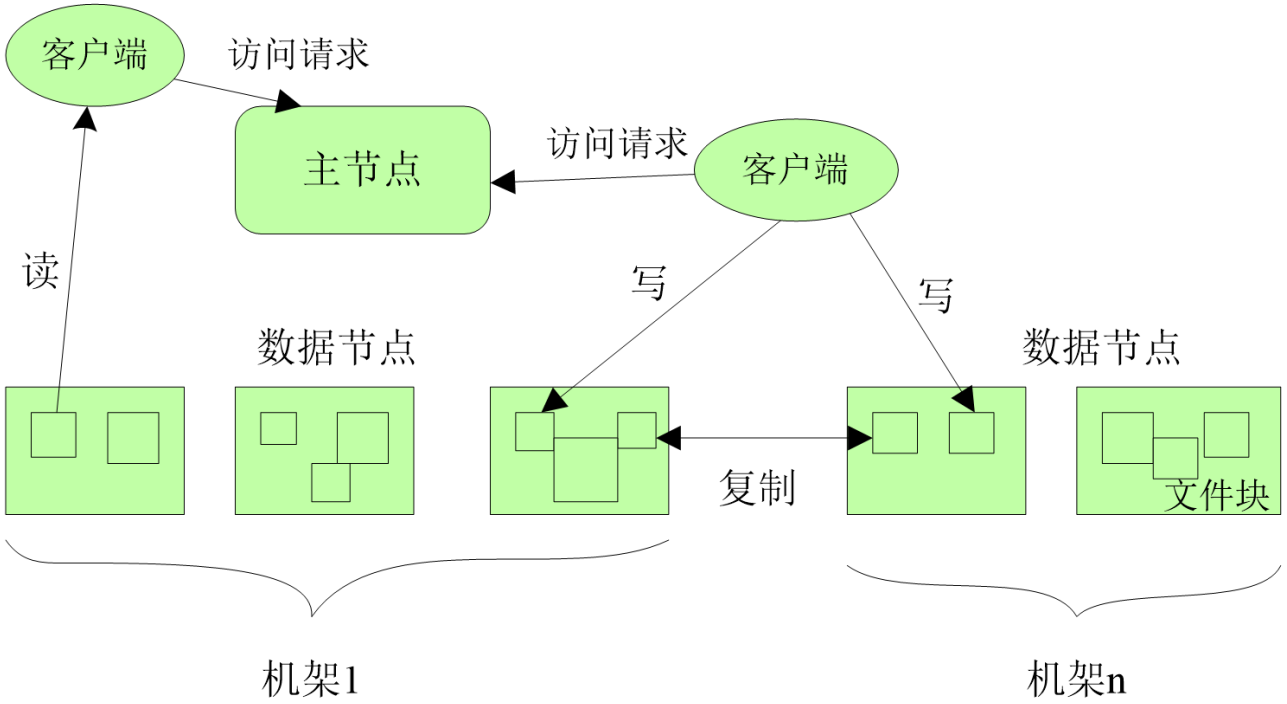
- 分布式文件系统把文件分布存储到多个计算机节点上，成千上万的计算机节点构成计算机集群



计算机集群的基本架构

分布式文件系统

- 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，这些节点分为两类，一类叫“主节点”(Master Node)或者也被称为“名称结点”(NameNode)，另一类叫“从节点”（Slave Node）或者也被称为“数据节点”(DataNode)



大规模文件系统的整体结构

HDFS简介

总体而言，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简介

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：

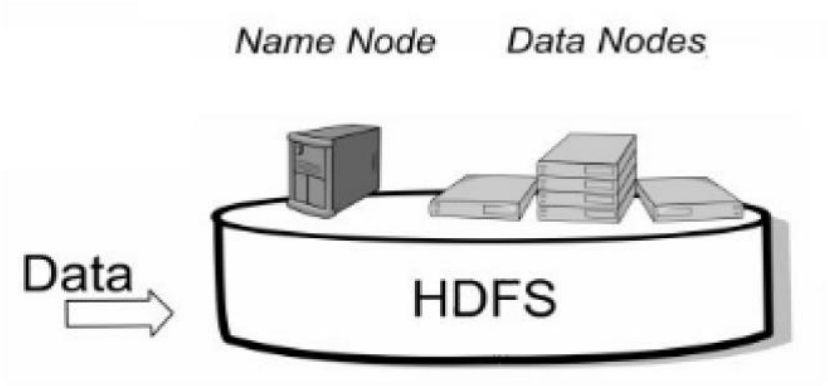
- 不适合低延迟数据访问
- 无法高效存储大量小文件
- 不支持多用户写入及任意修改文件

HDFS相关概念

- HDFS默认一个**数据块 Data Block** 64MB/128M，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，以降低分布式寻址开销，但也不宜过大
- HDFS采用抽象的块概念可以带来的明显的好处：
  - 支持大规模文件存储：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，不会受到单个节点的存储容量的限制
  - 简化系统设计：文件块大小是固定的，很容易计算出一个节点可以存储的文件块数以及一个文件的存储需求，方便元数据管理
  - 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

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

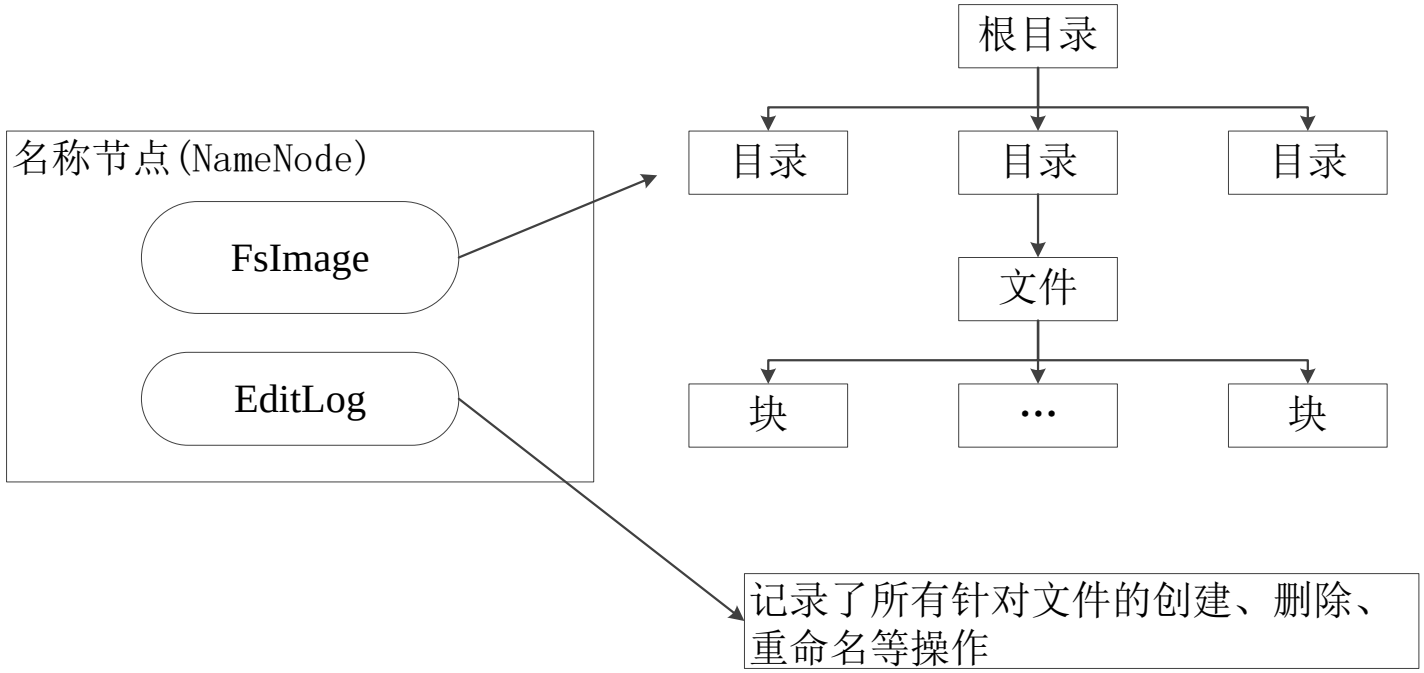
Blk B:
DN7, DN1, DN2

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

名称节点

- 在HDFS中，名称节点（NameNode）保存了两个核心的数据结构，即 FsImage 和 EditLog
  - FsImage维护文件树以及文件树中所有的文件和文件夹的元数据
  - EditLog中记录了所有针对文件的创建、删除、重命名等操作



名称节点

- FsImage文件：包含文件系统中所有目录和文件的元数据信息，如修改和访问时间、访问权限、分块情况等
- FsImage文件没有记录文件包含的每个块存储在哪个数据节点，文件块位置信息只存储在内存中
- 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

名称节点

- 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，使得内存中的元数据和实际的同步
- 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
- 名称节点在启动的过程中处于“安全模式”，只对个提供读操作，不提供写操作

名称节点

- 启动过程结束后，系统就会退出安全模式，进入正常运行状态，对外提供读写操作
- 名称节点启动成功并进入到正常运行状态以后，HDFS中的更新操作会重新写到EditLog文件中，而不是FsImage
  - 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，相对而言，EditLog 要小很多，写入EditLog是更加高效的
  - 每次执行写操作之后，且在向客户端发送成功代码之前， EditLog文件都需要同步更新

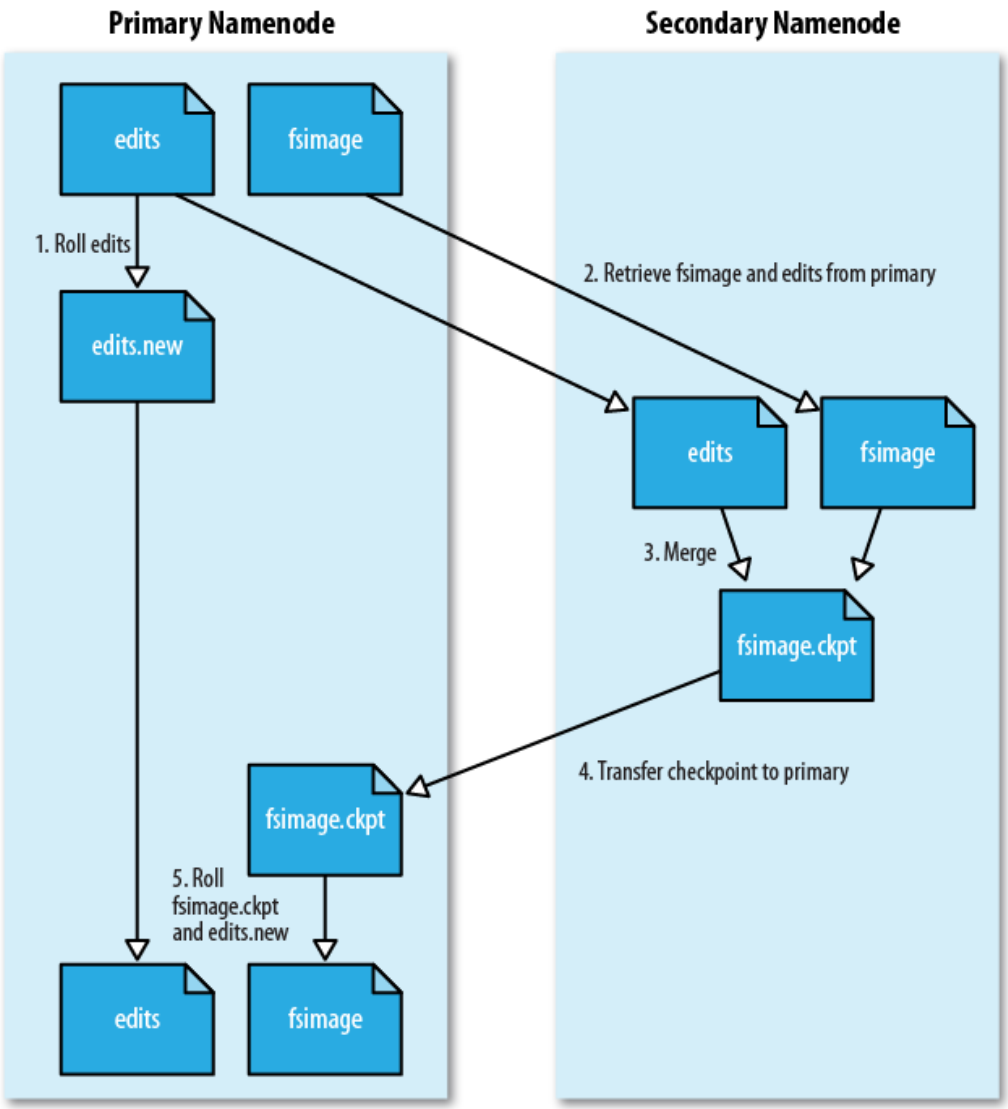
名称节点

如何解决运行期间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

SecondaryNameNode 第二名称节点

第二名称节点



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；

（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

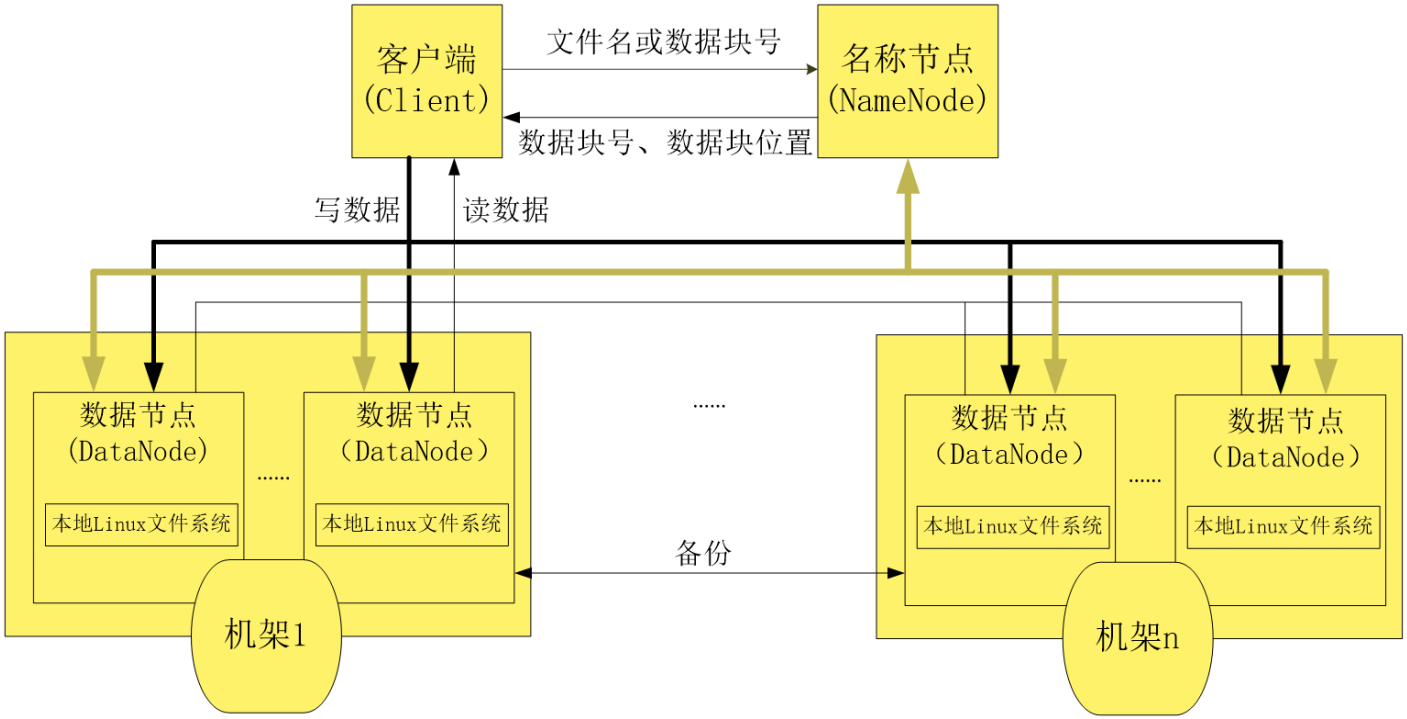
（5）NameNode将从SecondaryNameNode接收到的新的FsImage替换旧的FsImage文件，同时将edit.new替换EditLog文件，通过这个过程EditLog就变小了。

数据节点

- 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

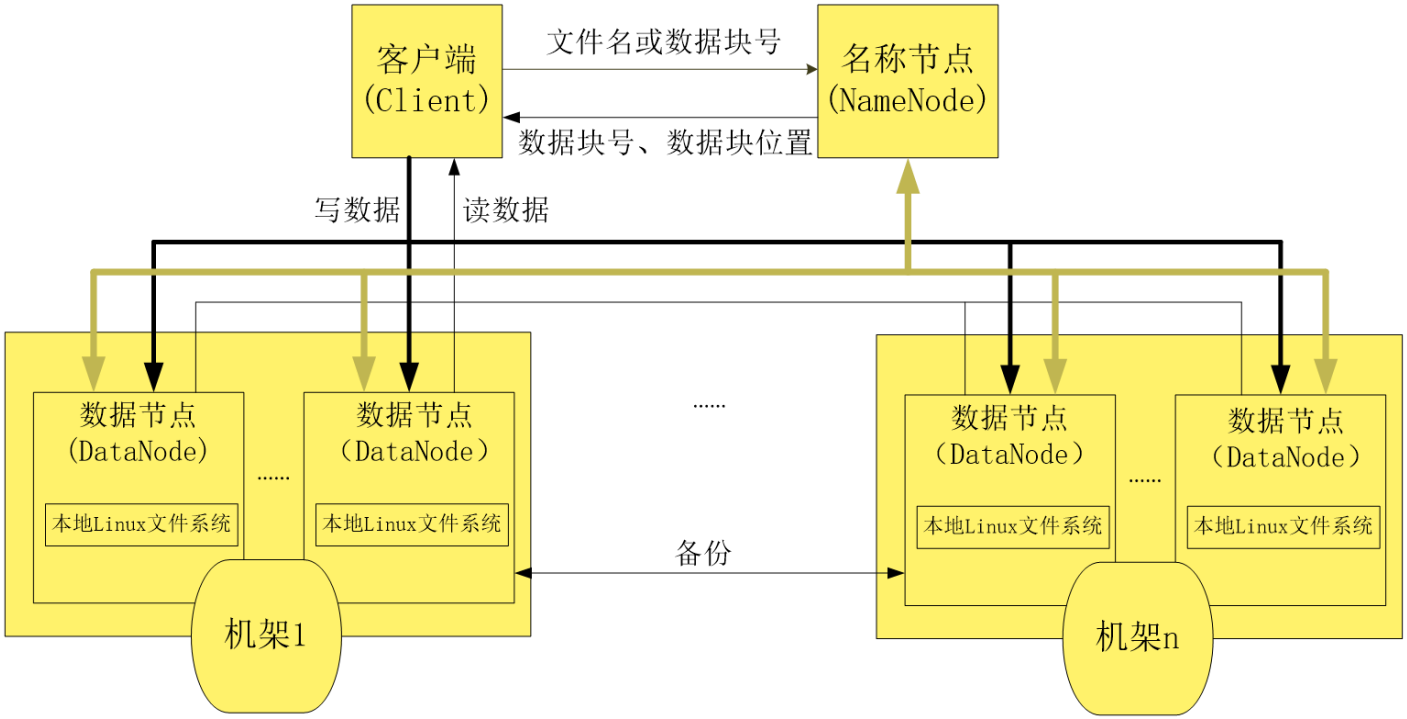
HDFS体系结构

- 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



HDFS体系结构

- 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
- 每个数据节点会周期性的向名称节点报告自己的状态，没有按时报告的数据节点会被标记为“宕机”，不再给它分配I/O请求



HDFS命名空间管理

- HDFS的命名空间包含目录、文件和块
- 在HDFS 1.0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

HDFS体系结构

HDFS 1.0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：

- 命名空间的限制：名称节点的元数据是保存在内存中的，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。
- 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。
- 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。
- 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。

HDFS存储原理

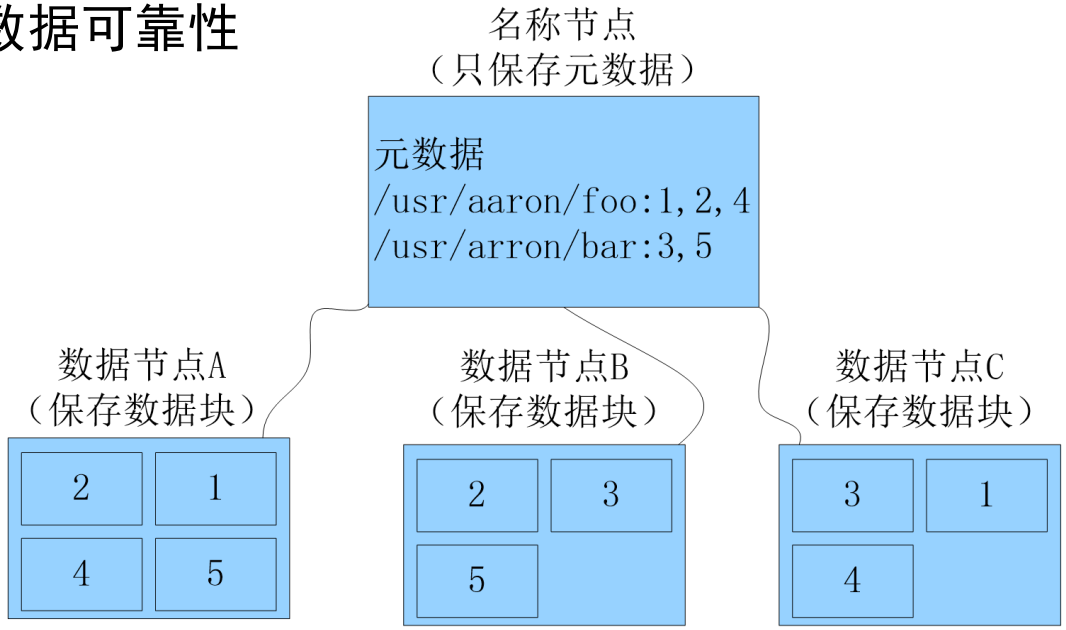
- 冗余数据保存
- 数据存取策略
- 数据错误与恢复

HDFS存储原理

- 机架感知（**Rack Awareness**）
  - 机架内的机器之间的网络速度通常都会高于跨机架机器之间的网络速度，并且机架之间机器的网络通信通常受到上层交换机间网络带宽的限制
  - 希望不同节点之间的通信能够尽量发生在同一个机架之内，而不是跨机架
  - 为了提高容错能力，名称节点会尽可能把数据块的副本放到多个机架上

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
  - 加快数据传输速度
  - 容易检查数据错误
  - 保证数据可靠性

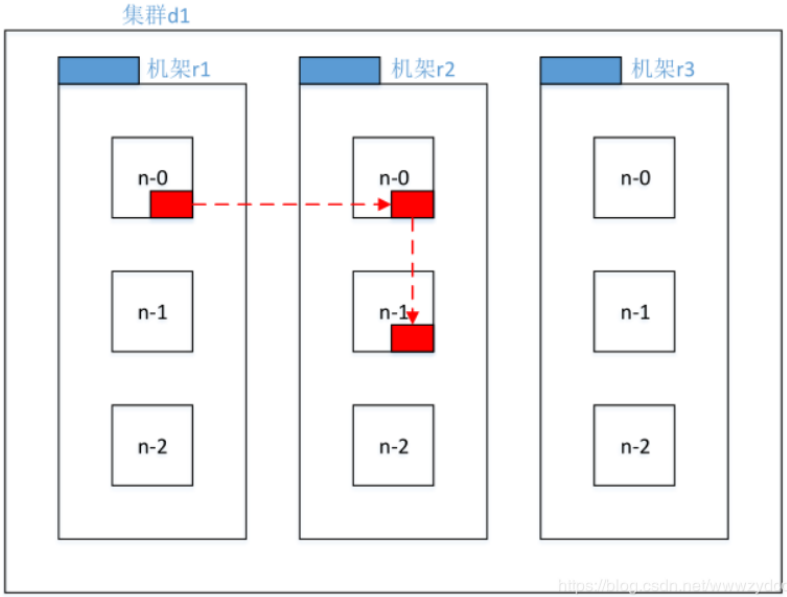


HDFS数据块多副本存储

数据存取策略

数据存放

- 第一个副本：如果是集群内发起写操作请求，放置在发起请求的数据节点上；如果来自集群外，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：与第二个副本相同机架的其他节点上
- 更多副本：随机节点



数据存取策略

数据读取：就近原则

- 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，开启机架感知后可以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

数据错误与恢复

- HDFS部署在大规模的集群上，并且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。

数据错误与恢复

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。
- HDFS设置了备份机制，把这些核心文件备份到第二名称节点SecondaryNameNode上。当名称节点出错时，就可以根据备份服务器中的FsImage和Editlog数据进行恢复，但需要暂停服务一段时间

数据错误与恢复

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（负载严重不均衡时也可做调整）

数据错误与恢复

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

HDFS数据读写过程

```java
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 HDFSReadFile {
    public static void main(String[] args) {
        try {
            Configuration conf = new Configuration();
            conf.set("fs.defaultFS", "hdfs://localhost:9000");
            FileSystem fs = FileSystem.get(conf);
            Path filename = new Path("input/localfile.txt");
//          Path filename = new Path("hdfs://localhost:9000/user/meihui/input/localfile.txt");
            FSDataInputStream is = fs.open(filename);
            BufferedReader d = new BufferedReader(new InputStreamReader(is));
            String content = d.readLine();
            System.out.println(content);
            d.close();
            fs.close();
        }catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

For HDFS, the current working directory is the HDFS home directory /user/<username> that often has to be created manually

HDFS数据读写过程

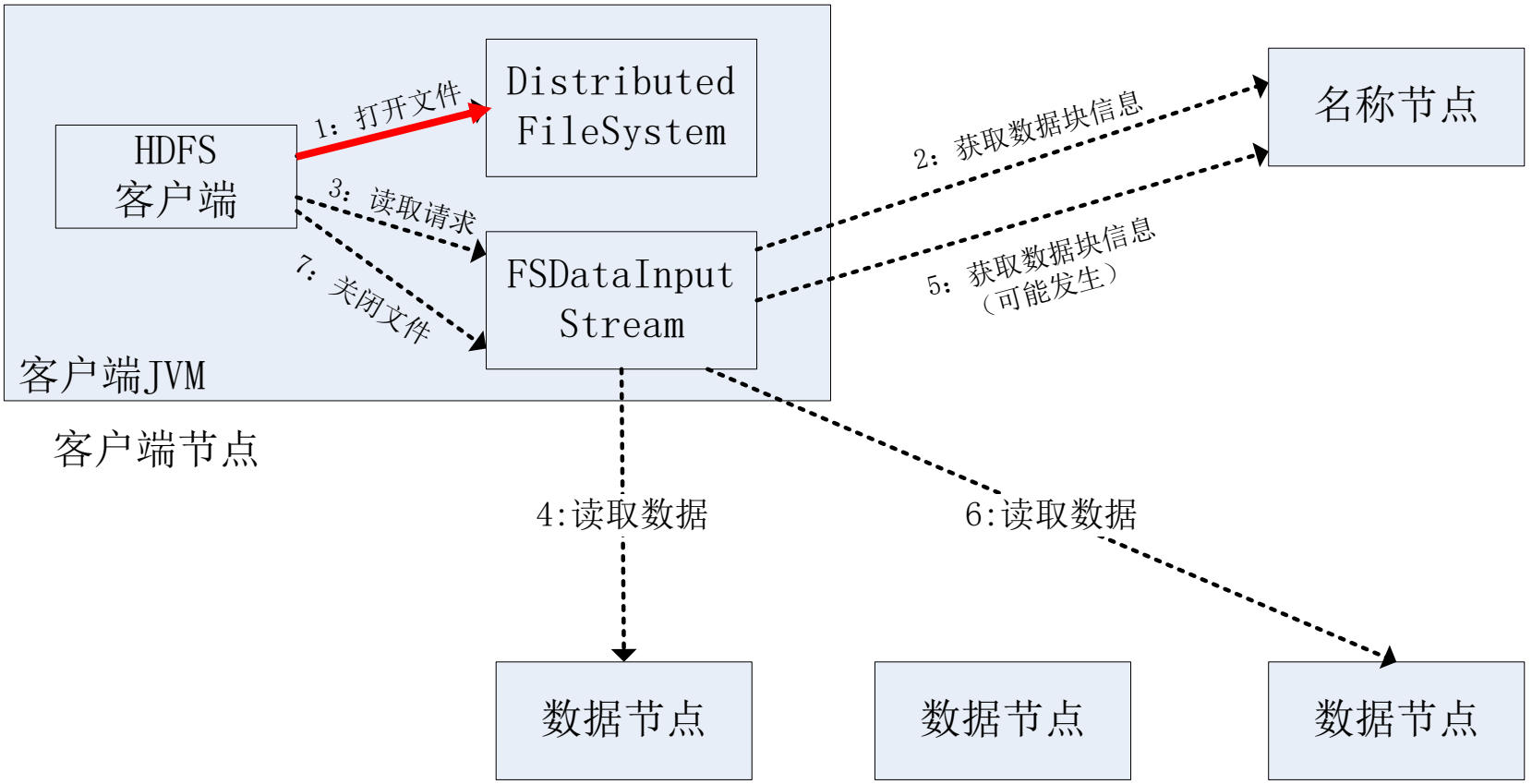
```java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.FSDataOutputStream;

public class HDFSWriteFile {
    public static void main(String[] args) {
        try {
            Configuration conf = new Configuration();
            conf.set("fs.defaultFS", "hdfs://localhost:9000");
            FileSystem fs = FileSystem.get(conf);
            String filename = "newfile.txt";
            FSDataOutputStream os = fs.create(new Path(filename));
            os.writeChars("This is a new file.");
            System.out.println("Created File: " + filename);
            os.close();
            fs.close();
        }catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

HDFS数据读写过程

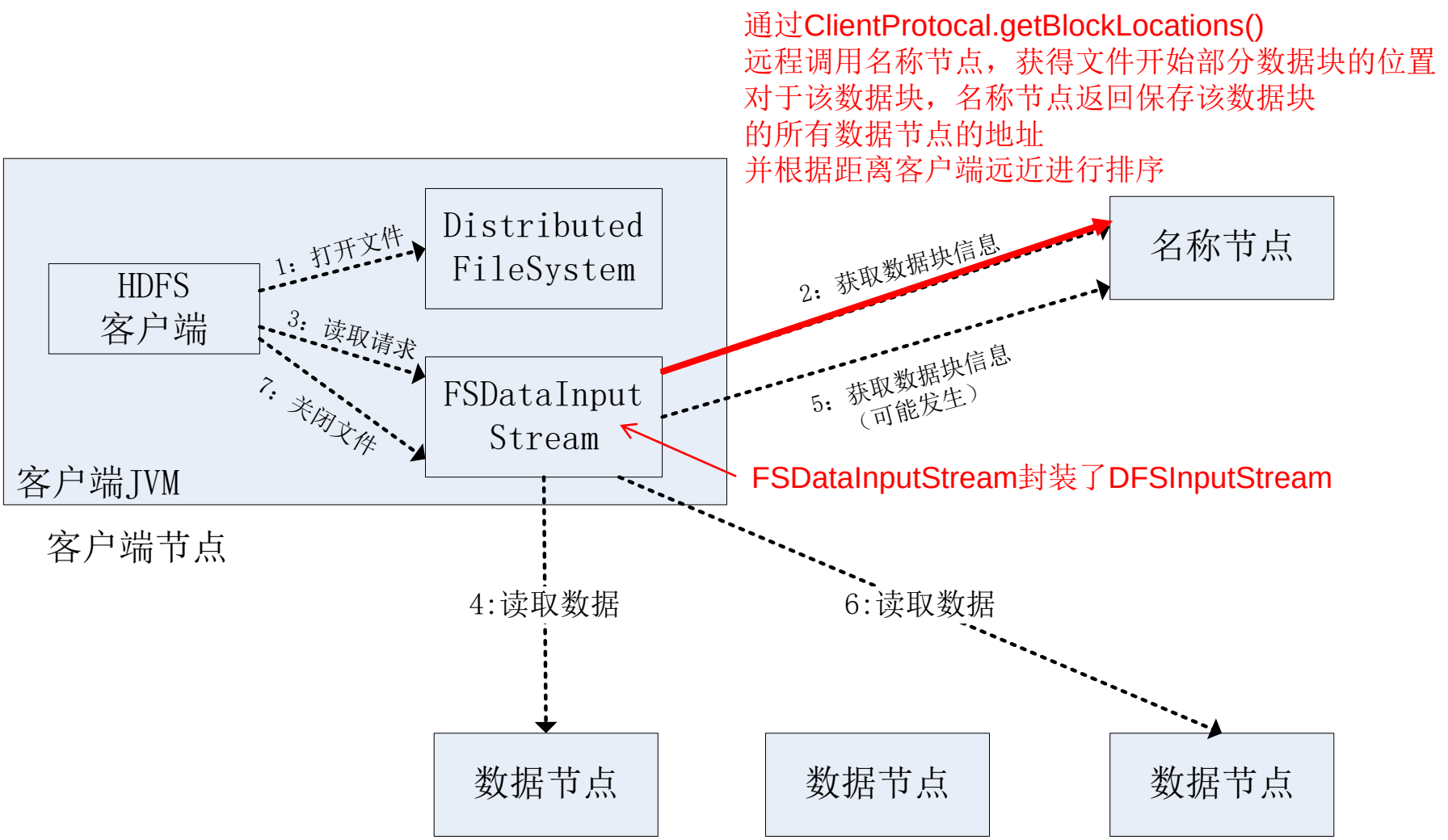
- FileSystem是一个通用文件系统的抽象基类，可以被分布式文件系统继承，所有可能使用Hadoop文件系统的代码，都要使用这个类
- Hadoop为FileSystem这个抽象类提供了多种具体实现
- DistributedFileSystem就是FileSystem在HDFS文件系统中的具体实现
- FileSystem的open()方法返回的是一个输入流FSDataInputStream对象，在HDFS文件系统中，具体的输入流就是DFSInputStream；FileSystem中的create()方法返回的是一个输出流FSDataOutputStream对象，在HDFS文件系统中，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

读数据的过程

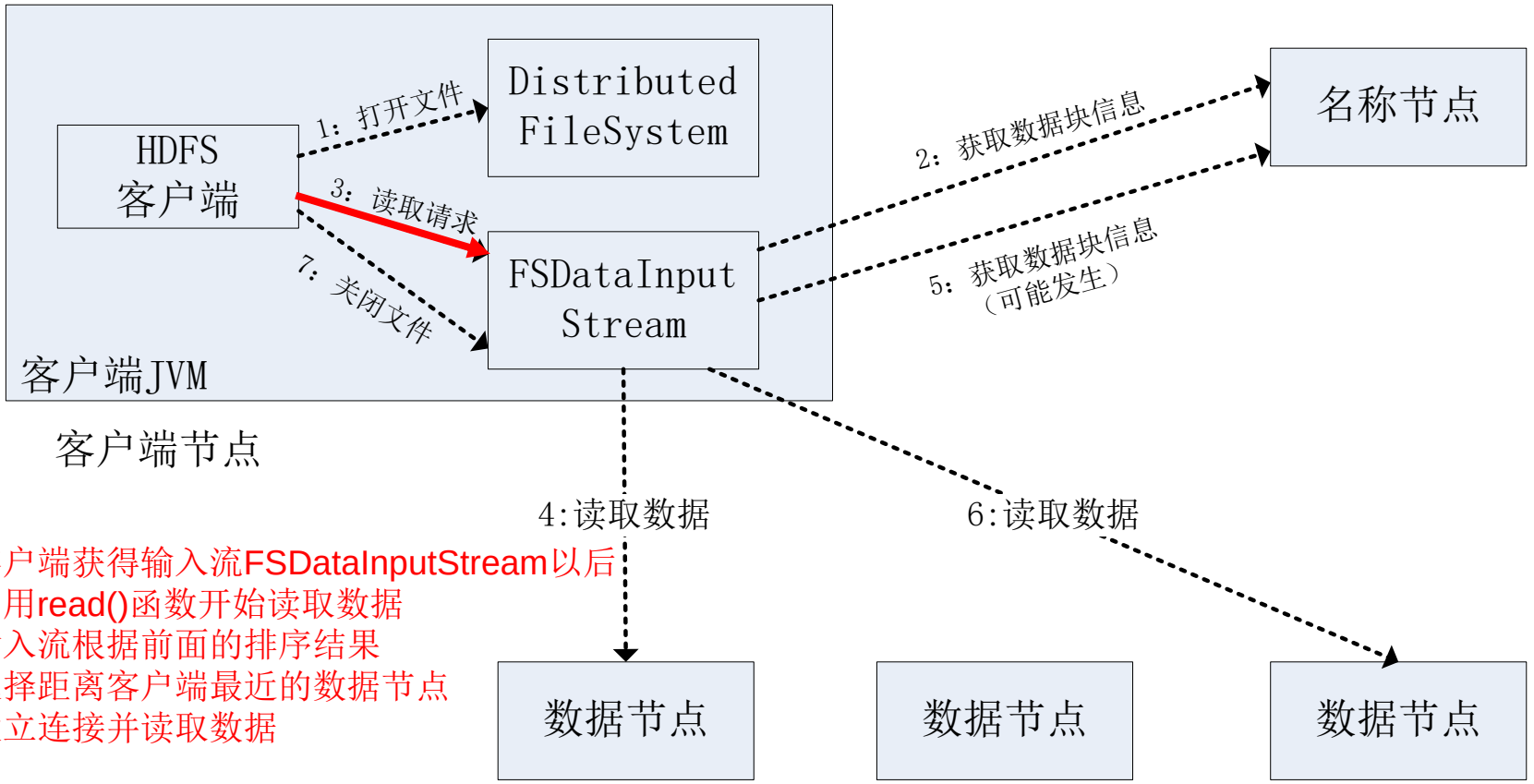
```
Configuration conf = new Configuration();
FileSystem fs = FileSystem.get(conf);
FSDataInputStream in = fs.open(new Path(uri));
```



读数据的过程

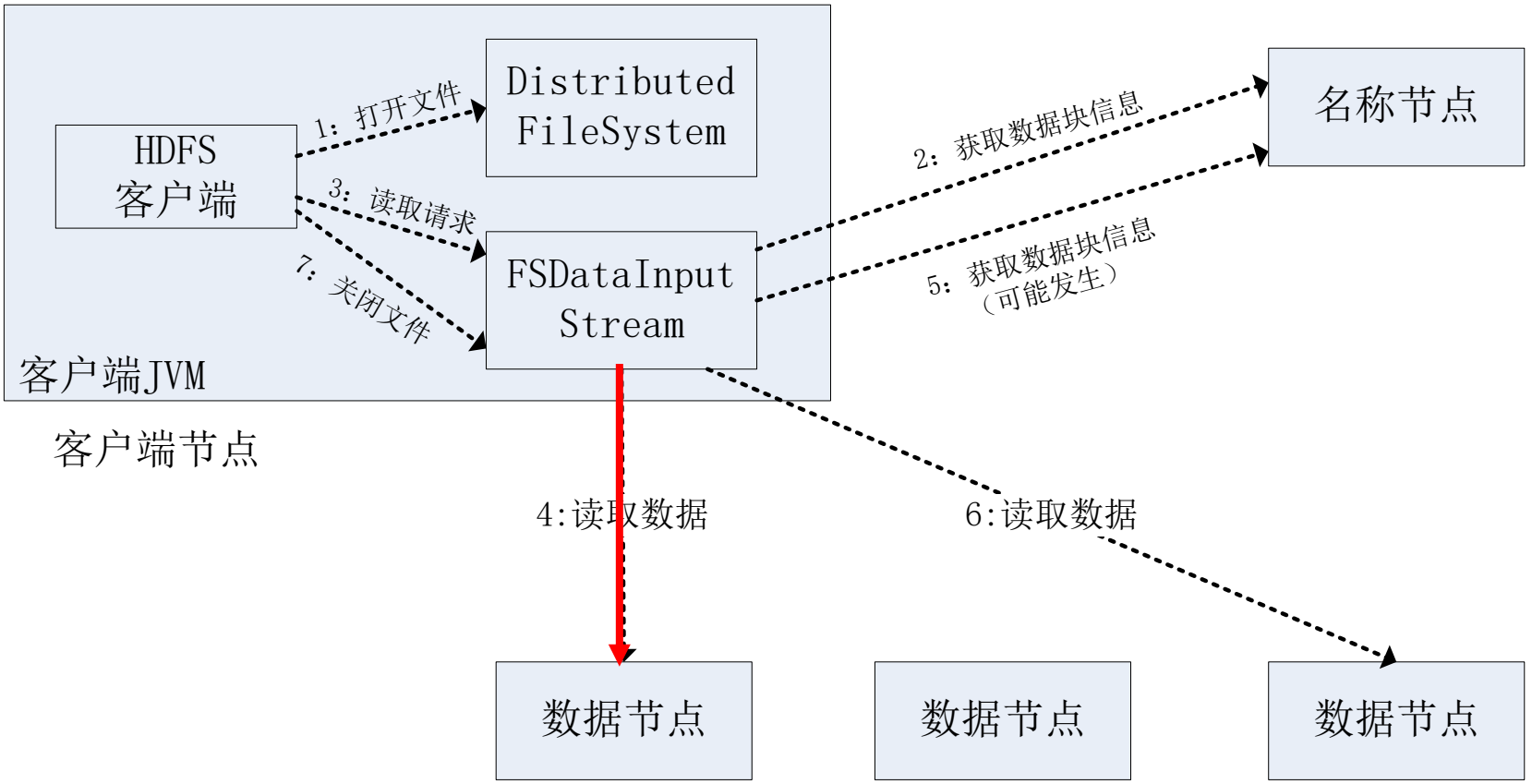

通过ClientProtocal.getBlockLocations()
远程调用名称节点，获得文件开始部分数据块的位置
对于该数据块，名称节点返回保存该数据块
的所有数据节点的地址
并根据距离客户端远近进行排序
HDFS
客户端
1：打开文件
Distributed
FileSystem
3：读取请求
7：关闭文件
FSDataInput
Stream
客户端JVM
客户端节点
2：获取数据块信息
名称节点
5：获取数据块信息
（可能发生）
FSDataInputStream封装了DFSInputStream
4:读取数据
6:读取数据
数据节点
数据节点
数据节点

读数据的过程



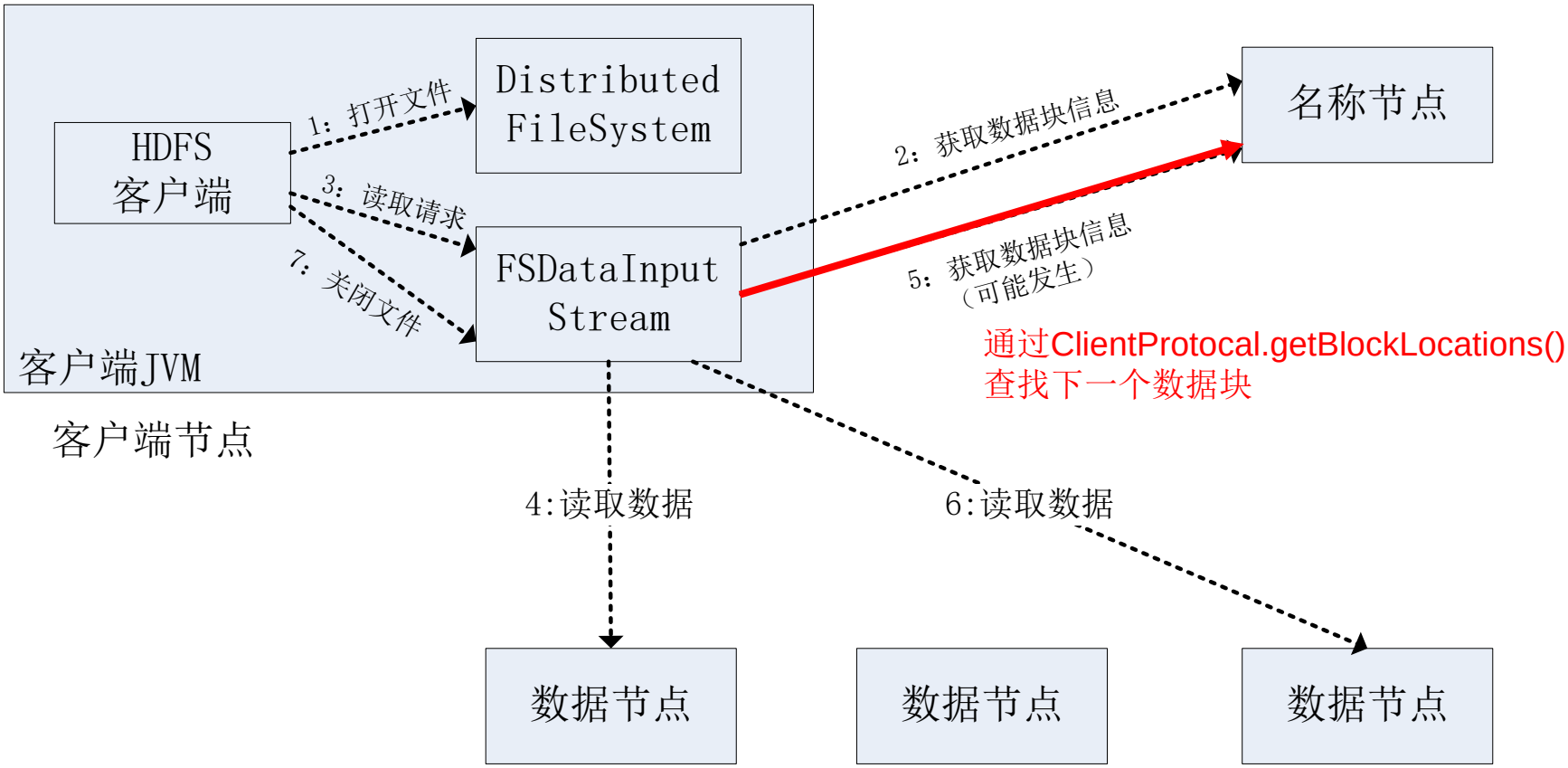
客户端获得输入流FSDataInputStream以后
调用read()函数开始读取数据
输入流根据前面的排序结果
选择距离客户端最近的数据节点
建立连接并读取数据

读数据的过程

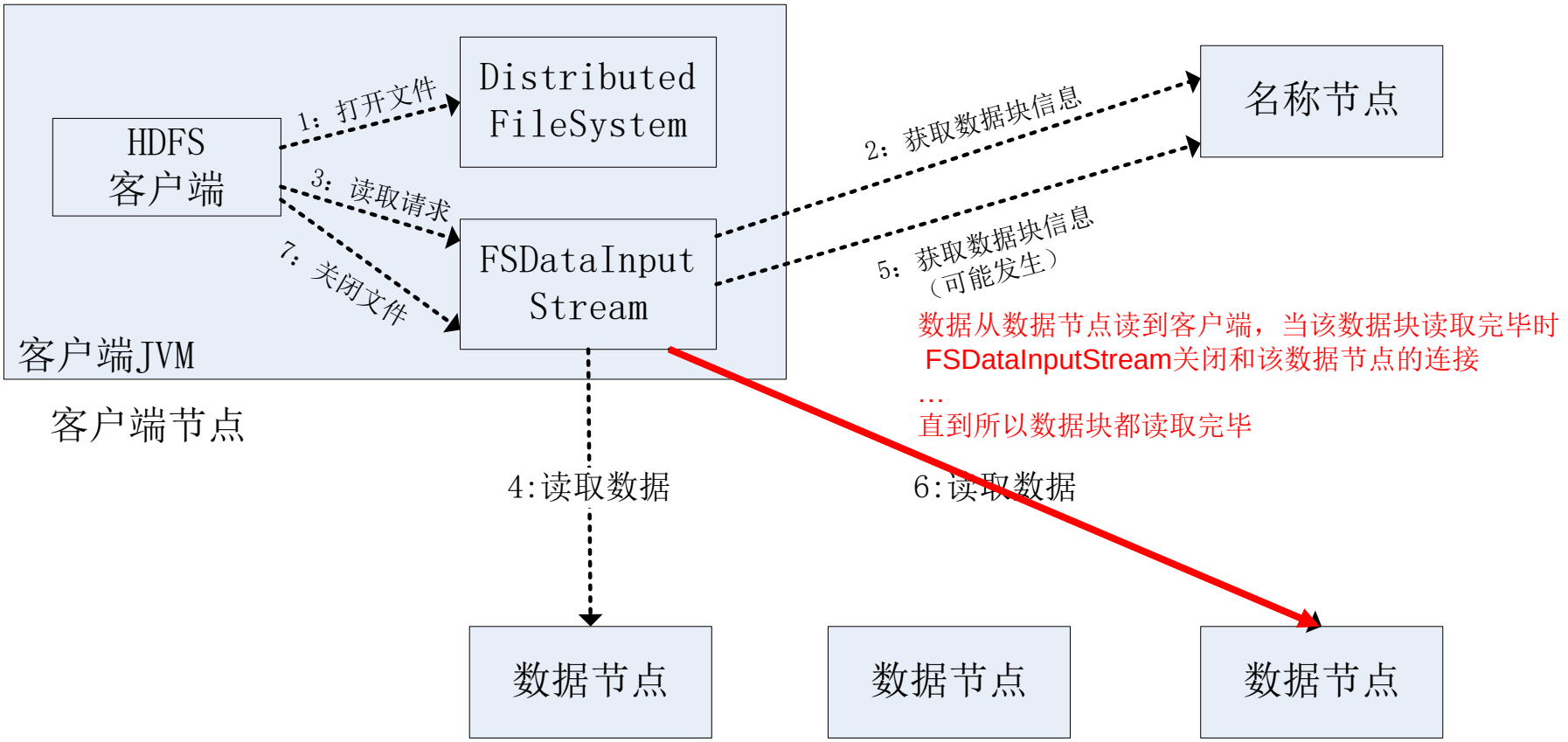


数据从数据节点读到客户端，当该数据块读取完毕时
FSDataInputStream关闭和该数据节点的连接

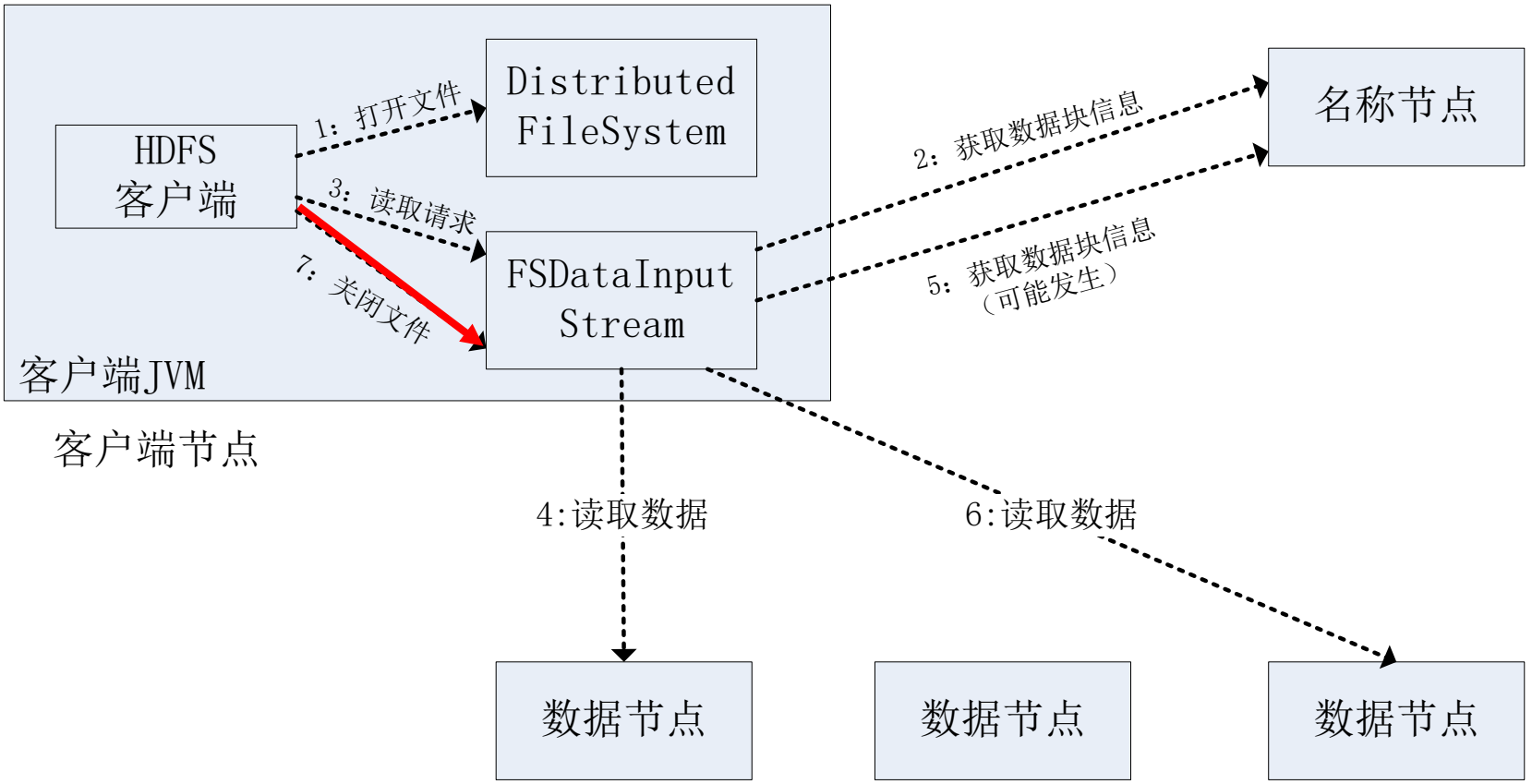
读数据的过程


HDFS
客户端
Distributed
FileSystem
FSDataInput
Stream
客户端JVM
客户端节点
1：打开文件
3：读取请求
7：关闭文件
2：获取数据块信息
5：获取数据块信息
（可能发生）
名称节点
通过ClientProtocal.getBlockLocations()
查找下一个数据块
4:读取数据
6:读取数据
数据节点
数据节点
数据节点

读数据的过程

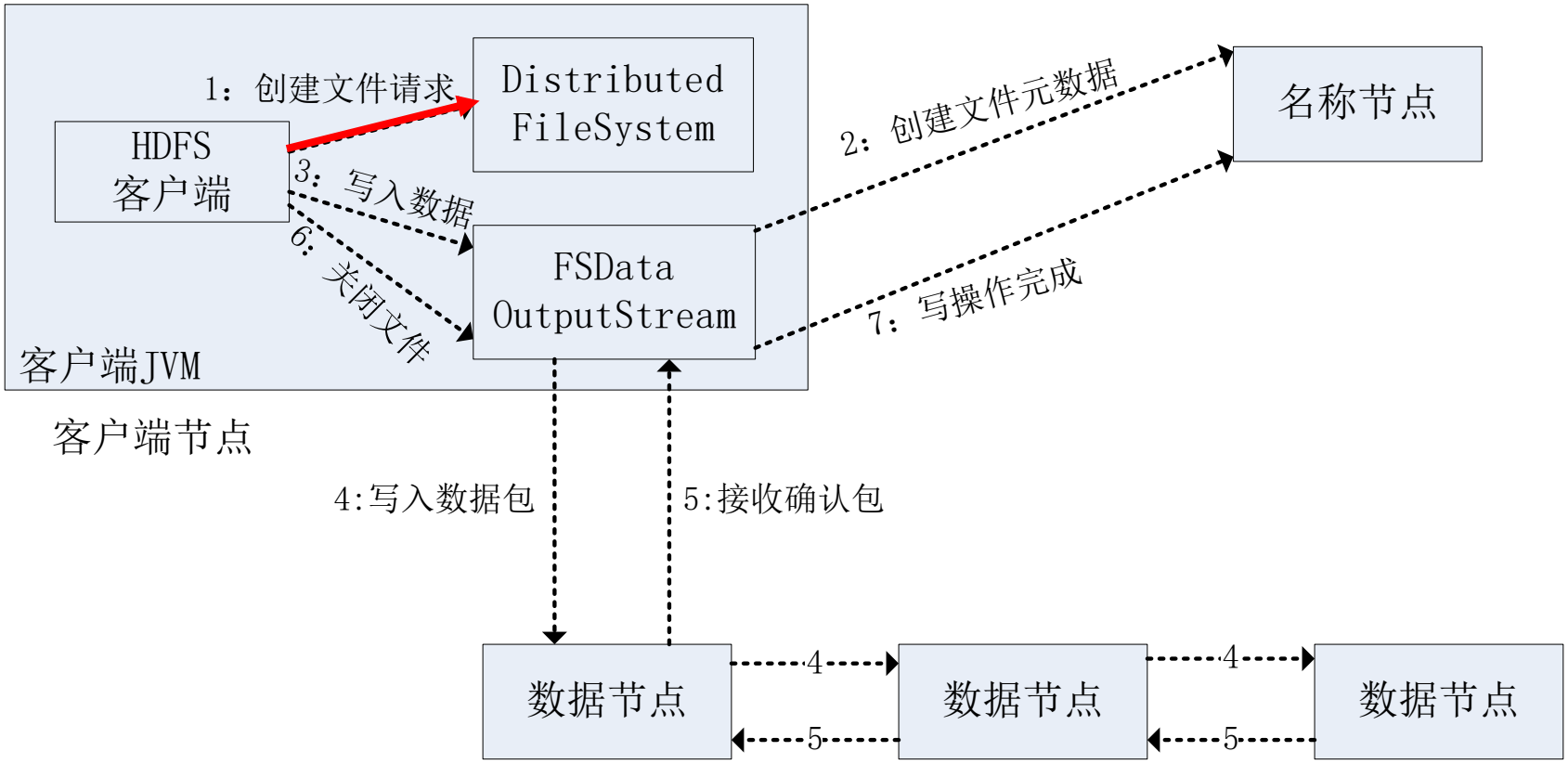

HDFS
客户端
Distributed
FileSystem
FSDataInput
Stream
1：打开文件
2：获取数据块信息
3：读取请求
4:读取数据
5：获取数据块信息
（可能发生）
6:读取数据
7：关闭文件
名称节点
客户端JVM
客户端节点
数据从数据节点读到客户端，当该数据块读取完毕时
FSDataInputStream关闭和该数据节点的连接
…
直到所以数据块都读取完毕
数据节点
数据节点
数据节点

读数据的过程


HDFS
客户端
Distributed
FileSystem
FSDataInput
Stream
1：打开文件
2：获取数据块信息
3：读取请求
4:读取数据
5：获取数据块信息
（可能发生）
6:读取数据
7：关闭文件
名称节点
数据节点
数据节点
数据节点
客户端JVM
客户端节点

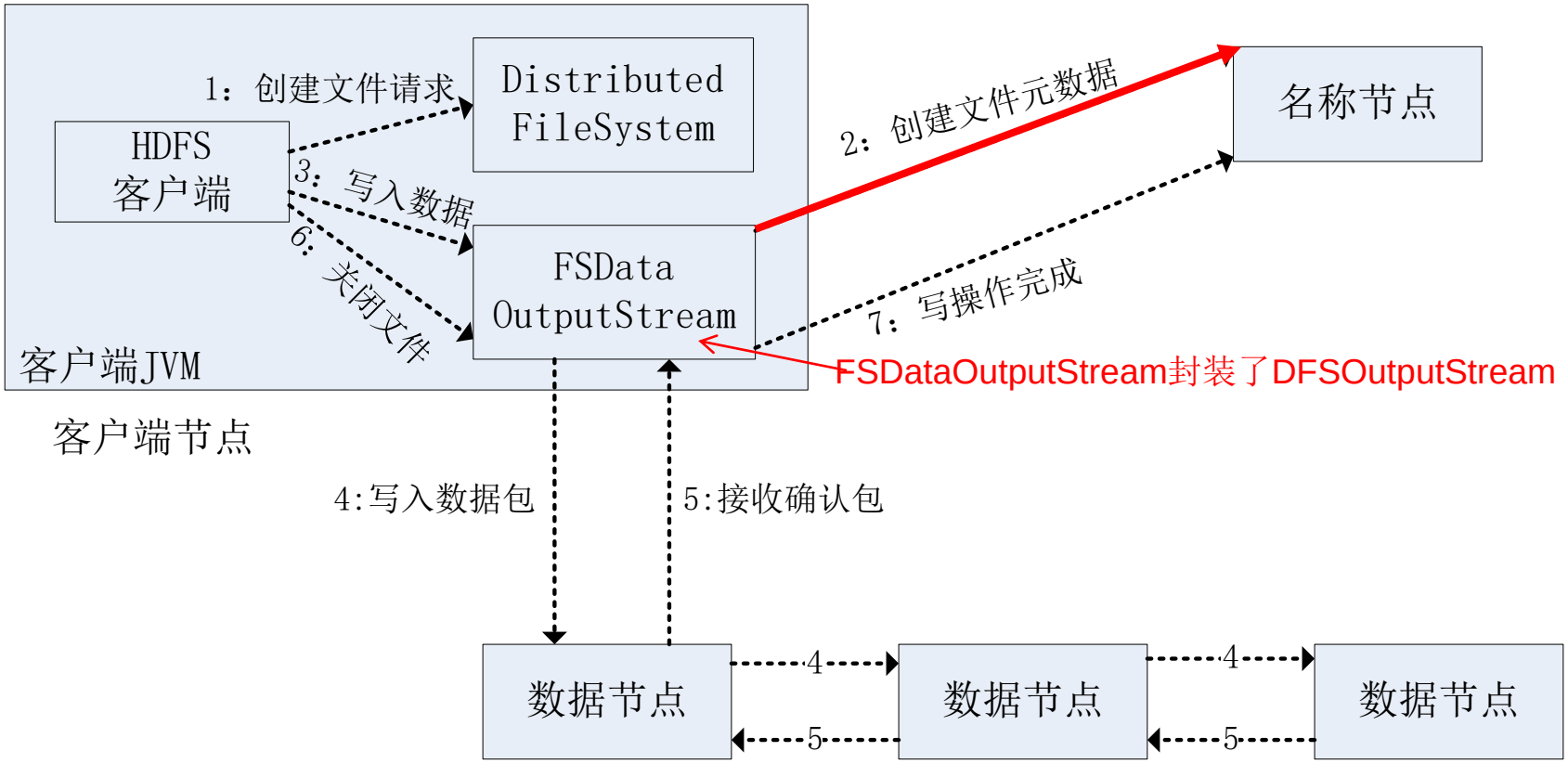
写数据的过程

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

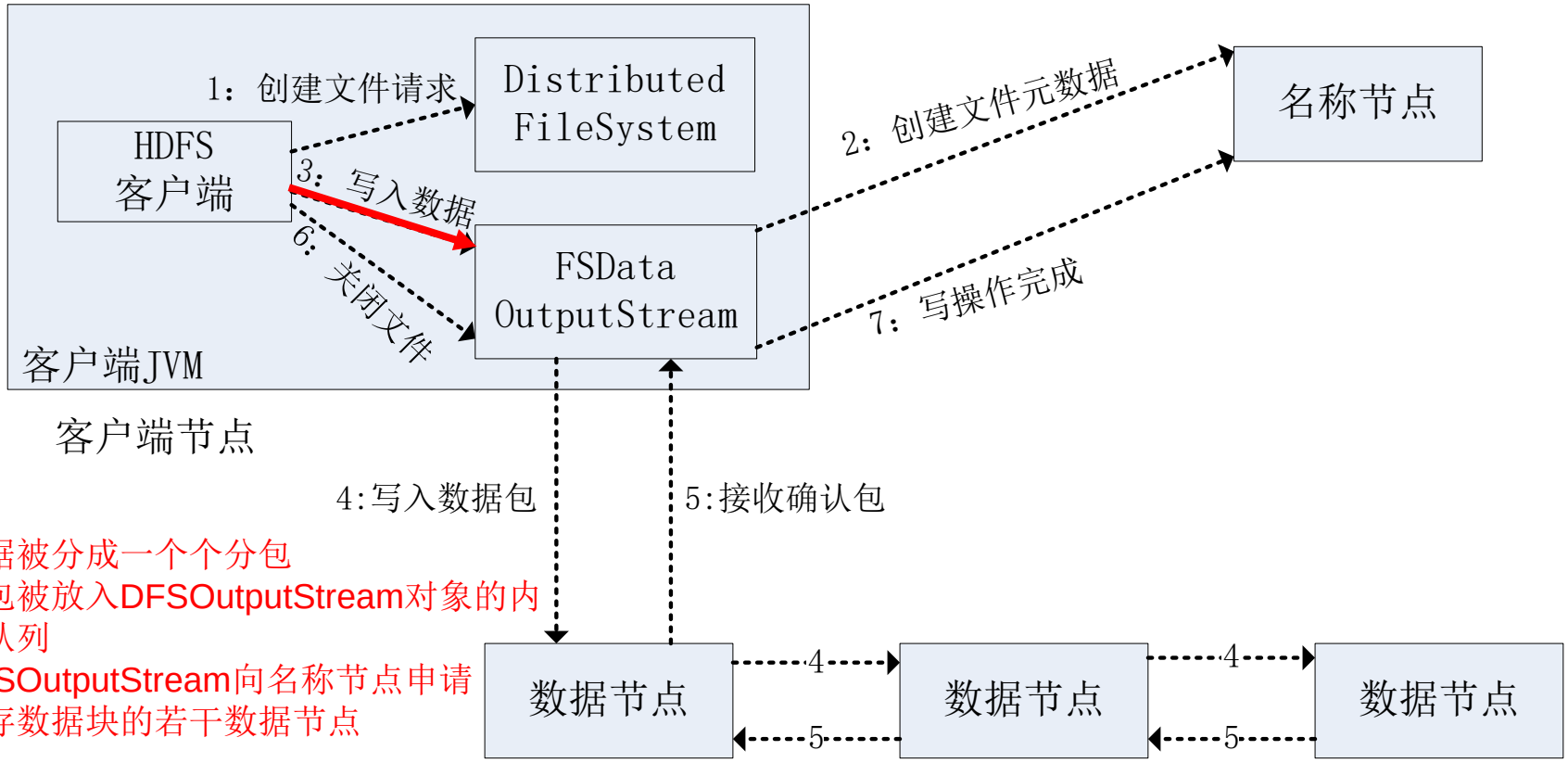


写数据的过程

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）

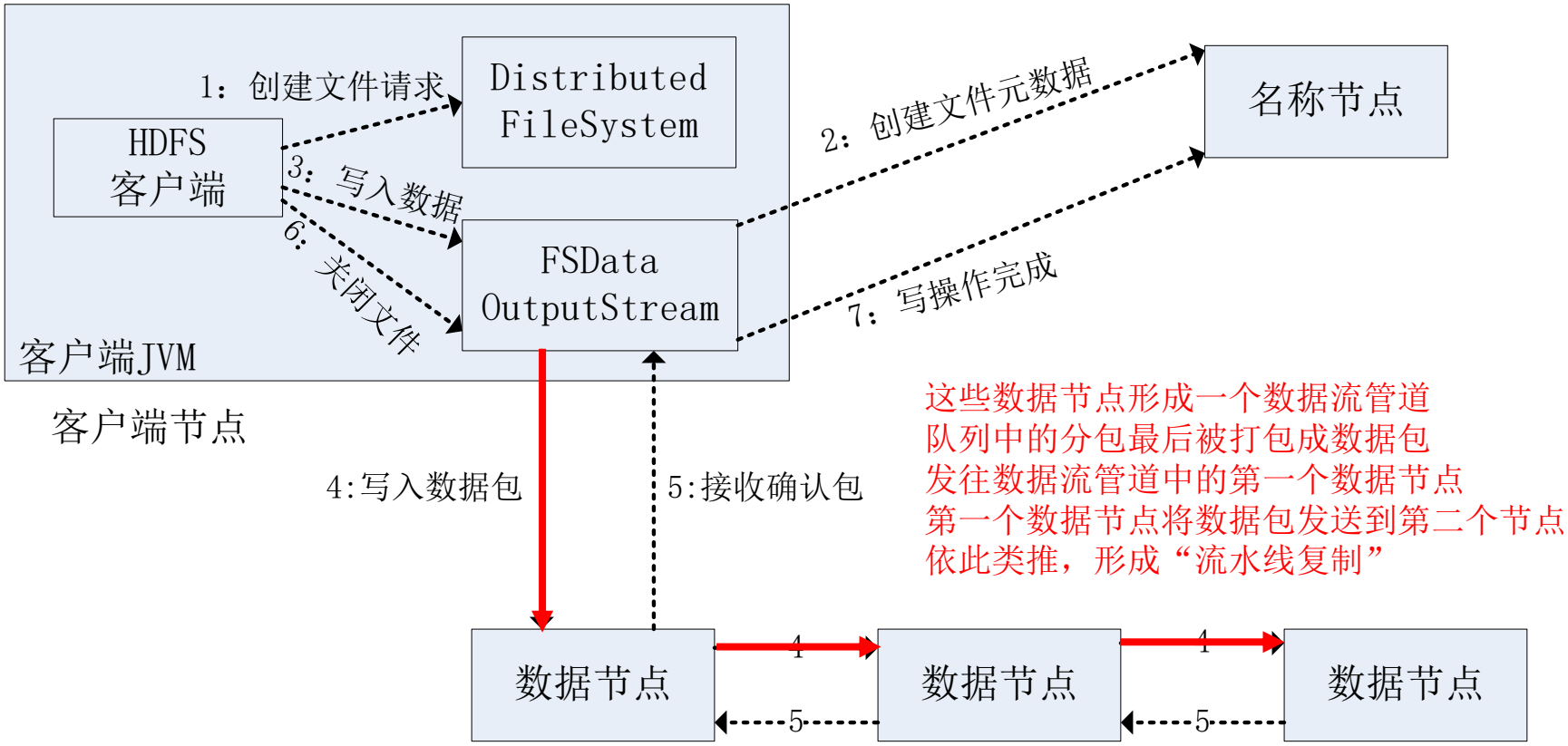


写数据的过程

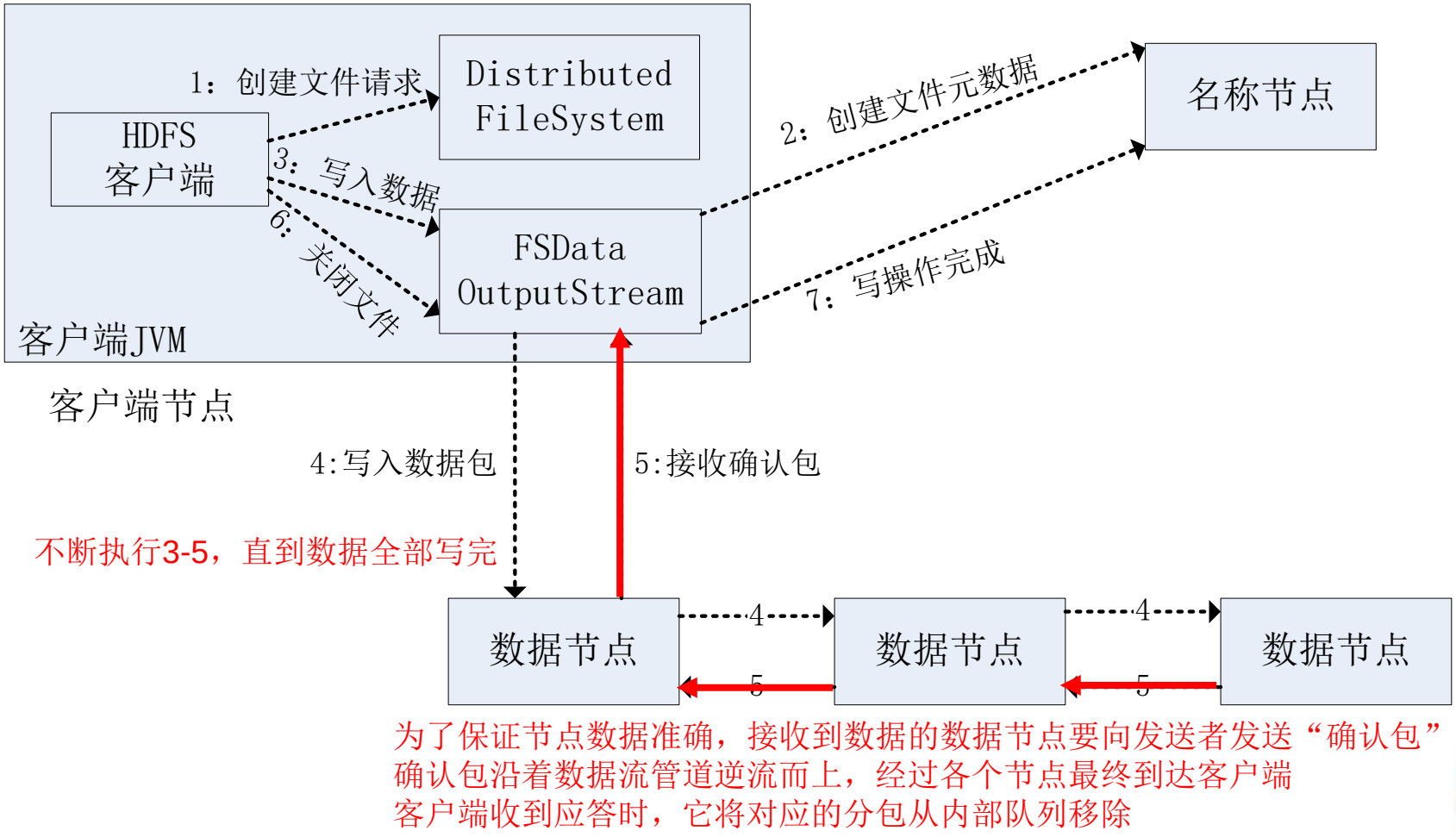


数据被分成一个个分包
分包被放入DFSOutputStream对象的内部队列
DFSOutputStream向名称节点申请保存数据块的若干数据节点

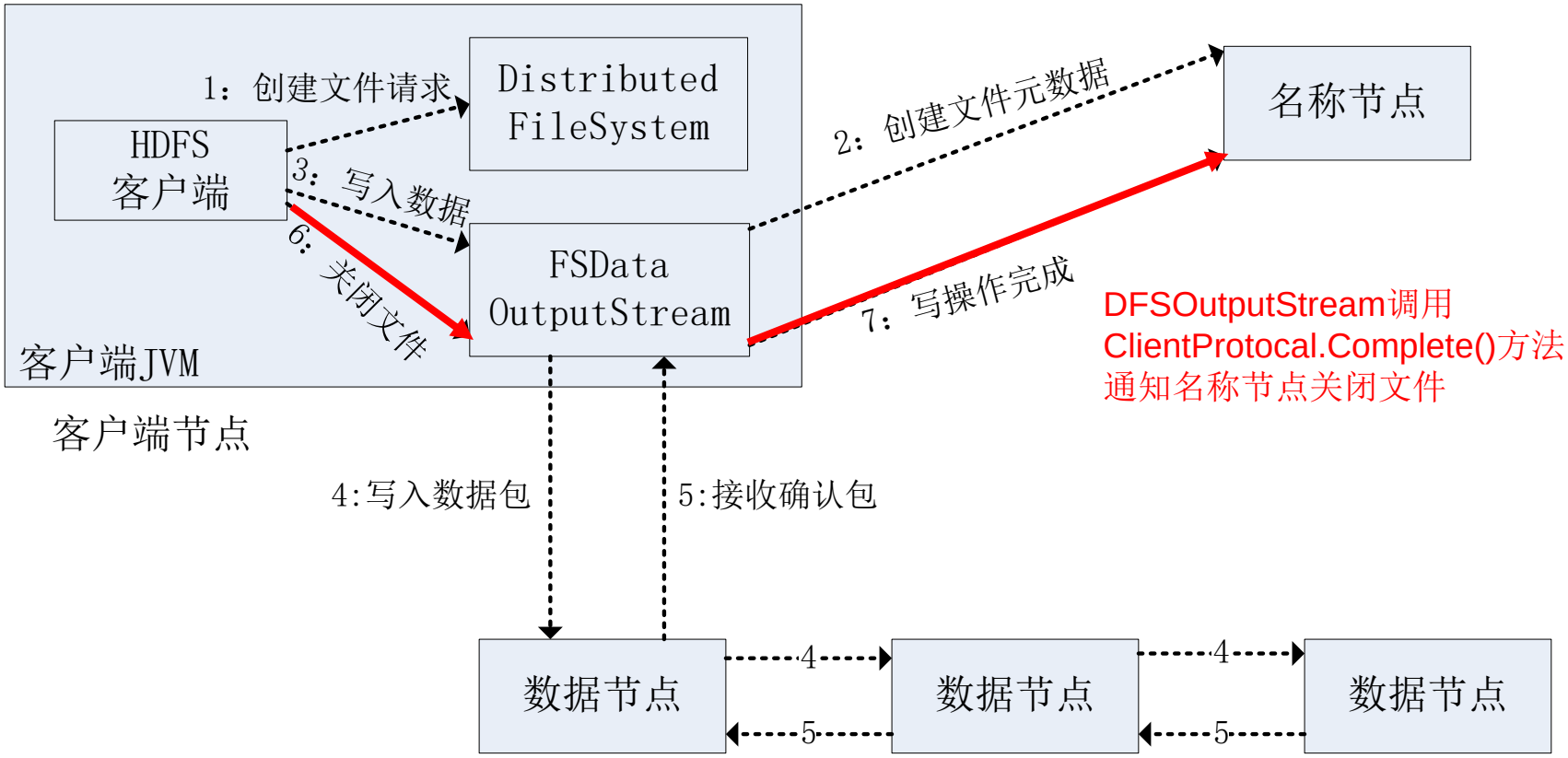
写数据的过程


1：创建文件请求
Distributed FileSystem
HDFS 客户端
3：写入数据
6：关闭文件
FSData OutputStream
客户端JVM
客户端节点
2：创建文件元数据
名称节点
7：写操作完成
4:写入数据包
5:接收确认包
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”
数据节点
4
5
数据节点
4
5
数据节点

写数据的过程


1：创建文件请求
Distributed FileSystem
HDFS 客户端
3：写入数据
6：关闭文件
FSData OutputStream
客户端JVM
2：创建文件元数据
名称节点
7：写操作完成
客户端节点
4:写入数据包
5:接收确认包
不断执行3-5，直到数据全部写完
数据节点
4
5
数据节点
4
5
数据节点
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

写数据的过程


1：创建文件请求
Distributed FileSystem
HDFS 客户端
2：创建文件元数据
名称节点
3：写入数据
6：关闭文件
FSData OutputStream
7：写操作完成
客户端JVM
客户端节点
DFSOutputStream调用
ClientProtocal.Complete()方法
通知名称节点关闭文件
4:写入数据包
5:接收确认包
数据节点
4
5
数据节点
4
5
数据节点

HDFS编程实践

- Hadoop提供了关于HDFS上进行文件操作常用的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，需要启动Hadoop
  - start-dfs.sh
  - jps: 查看java进程
  - 若成功启动，可以看到NameNode、DataNode和SecondaryNameNode

HDFS操作常用Shell命令

- http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/FileSystemShell.html
- The general command line syntax is
  - bin/hadoop command [genericOptions] [commandOptions]

- 配置PATH环境变量
  - 在~/.bashrc中进行设置
  - 添加后执行source ~/.bashrc 使设置生效

```
export PATH=$PATH:/usr/local/hadoop/bin:/usr/local/hadoop/sbin:/usr/local/hbase/bin
```

HDFS操作常用Shell命令

实例：

hadoop fs -ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] <path>: 显示<path>指定的文件的详细信息

- -C: Display the paths of files and directories only.
- -d: Directories are listed as plain files.
- -h: Format file sizes in a human-readable fashion (eg 64.0m instead of 67108864).
- -q: Print ? instead of non-printable characters.
- -R: Recursively list subdirectories encountered.
- -t: Sort output by modification time (most recent first).
- -S: Sort output by file size.
- -r: Reverse the sort order.
- -u: Use access time rather than modification time for display and sorting.

hadoop fs -mkdir [-p] <paths>: 创建<path>指定的文件夹

```
meihui@meihui-VirtualBox:~$ hadoop fs -ls .
Found 1 items
drwxr-xr-x   - meihui supergroup          0 2019-04-27 16:57 input
meihui@meihui-VirtualBox:~$ hadoop fs -ls /user/meihui
Found 1 items
drwxr-xr-x   - meihui supergroup          0 2019-04-27 16:57 /user/meihui/input
meihui@meihui-VirtualBox:~$ hadoop fs -mkdir output
meihui@meihui-VirtualBox:~$ hadoop fs -ls /user/meihui
Found 2 items
drwxr-xr-x   - meihui supergroup          0 2019-04-27 16:57 /user/meihui/input
drwxr-xr-x   - meihui supergroup          0 2019-04-27 17:44 /user/meihui/output
```

HDFS操作常用Shell命令

实例：

hadoop fs -put [- | <localsrc1> ..]. <dst>: 将本地文件上传到HDFS

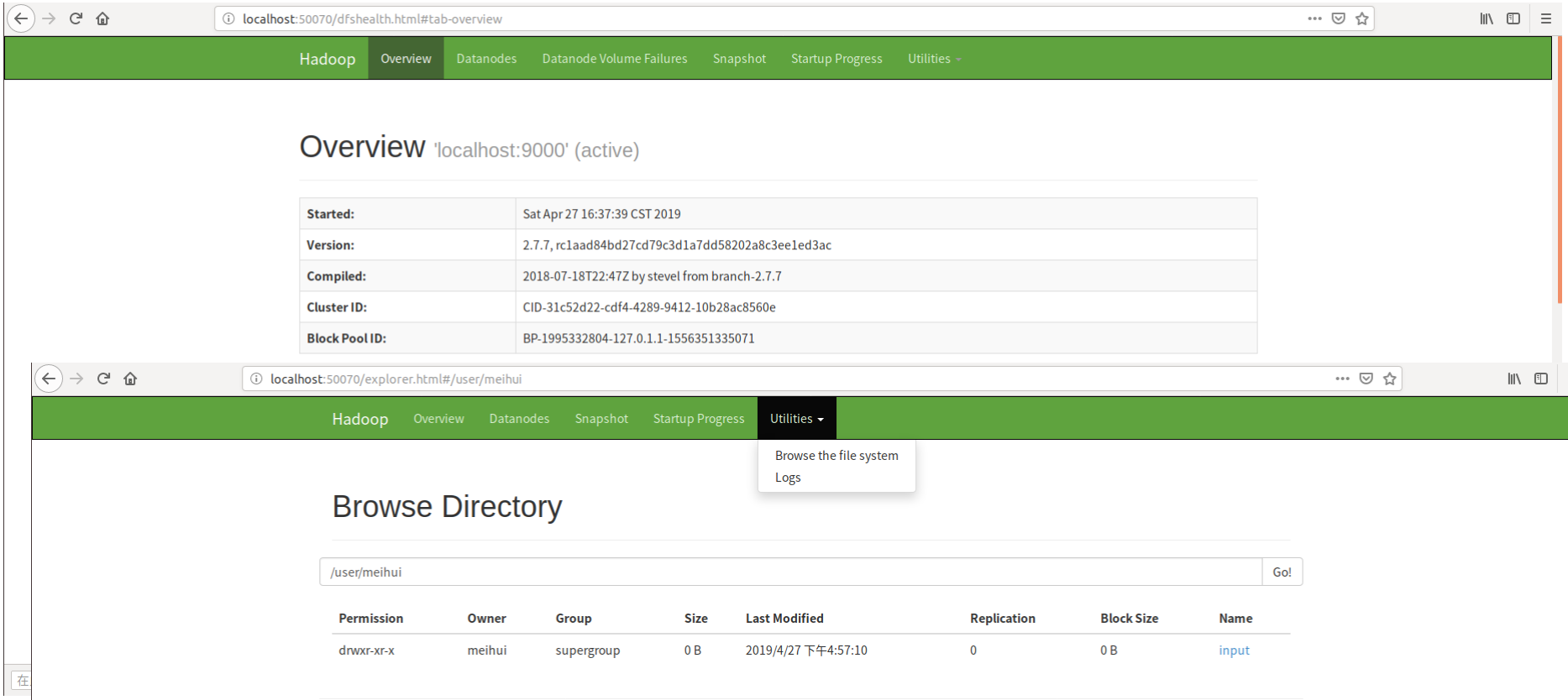
hadoop fs -get <src> <localdst>: 将HDFS文件下载到本地

```
meihui@meihui-VirtualBox:~$ hadoop fs -put localfile.txt output
meihui@meihui-VirtualBox:~$ hadoop fs -ls output
Found 1 items
-rw-r--r--   1 meihui supergroup         11 2019-04-27 17:45 output/localfile.txt
```

```
meihui@meihui-VirtualBox:~$ hadoop fs -get output/localfile.txt tmp
meihui@meihui-VirtualBox:~$ ls
eclipse-workspace  hbase          output  tmp        公共的  视频  文档  音乐
examples.desktop   localfile.txt  snap    zookeeper  模板    图片  下载  桌面
meihui@meihui-VirtualBox:~$ cd tmp/
meihui@meihui-VirtualBox:~/tmp$ ls
localfile.txt
```

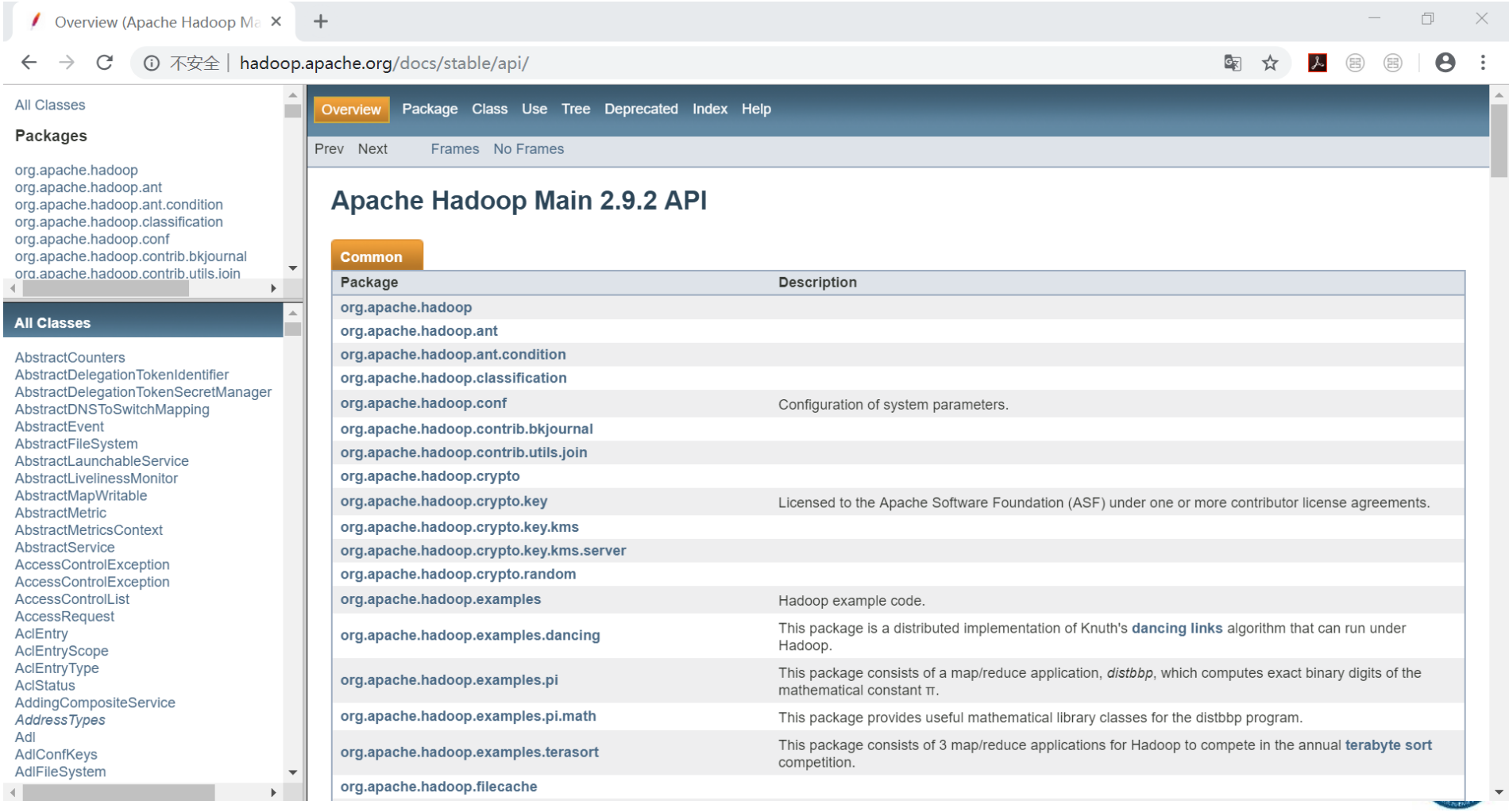
HDFS的Web界面

http://[NameNodeIP]:50070



HDFS常用Java API

- http://hadoop.apache.org/docs/stable/api/



HDFS常用Java API

- 实验二：HDFS操作
- 准备工作：安装Java IDE

- 实验内容：
  - 熟悉HDFS操作常用的Shell命令
  - 熟悉HDFS操作常用的Java API

HDFS常用Java API

- 为项目加载所需要用到的jar包

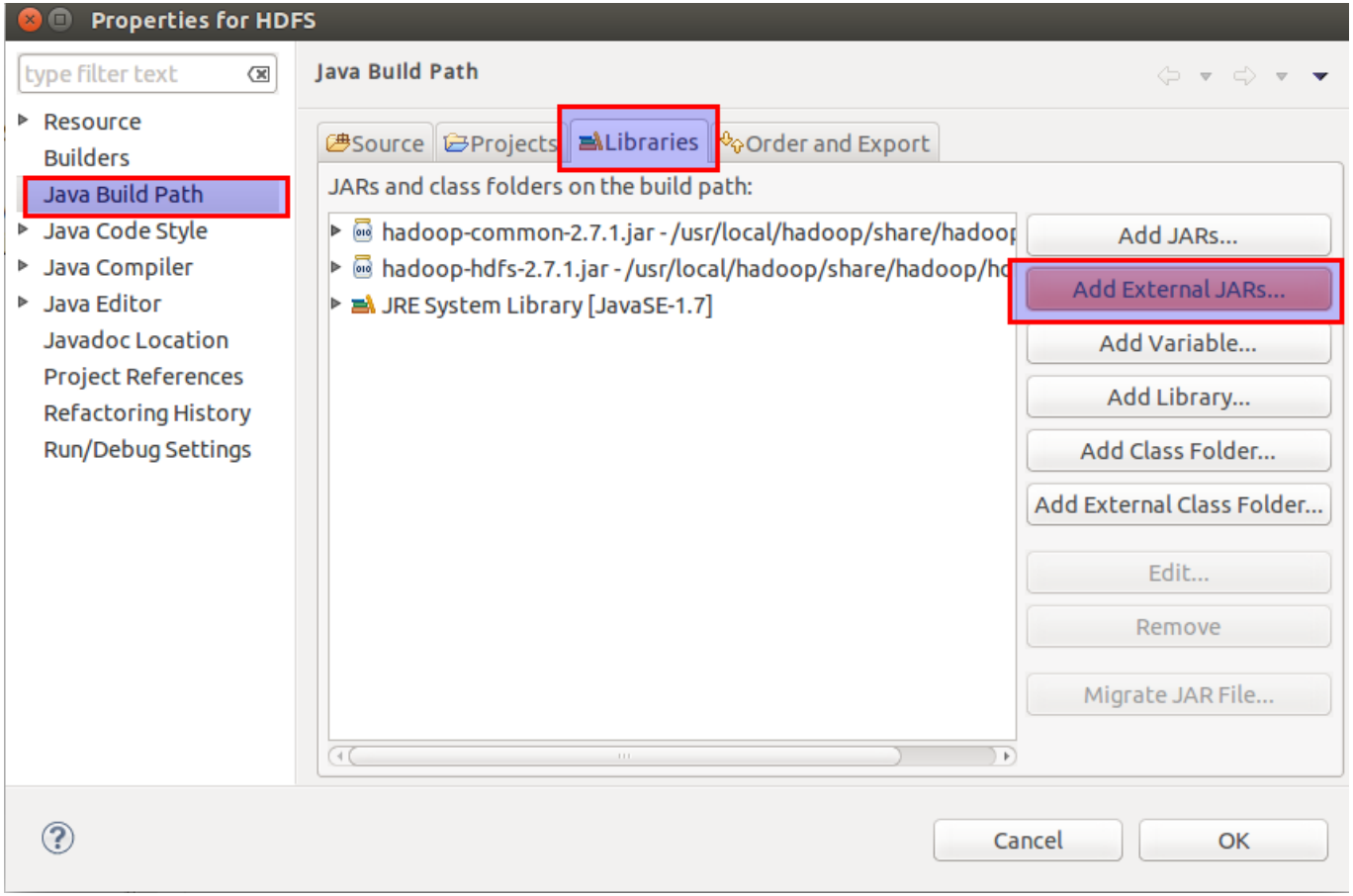
  路径：/usr/local/hadoop/share/hadoop

通常：

- /usr/local/hadoop/share/hadoop/common
- /usr/local/hadoop/share/hadoop/common/lib
- /usr/local/hadoop/share/hadoop/hdfs
- /usr/local/hadoop/share/hadoop/hdfs/lib

HDFS常用Java API

- 为项目加载所需要用到的jar包



HDFS常用Java API及应用实例

- 利用JAVA API与HDFS进行交互
- 实例：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test.txt文件？

- 第一步:放置配置文件到当前工程下面
- 需要把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。

HDFS常用Java API及应用实例

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

public class HDFSFileExists {
    public static void main(String[] args) {
          try {
              String filename = "hdfs://localhost:9000/user/meihui/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();
        }
    }
}
```

小结

- 分布式文件系统是大数据时代解决大规模数据存储问题的有效解决方案，HDFS开源实现了GFS，可以利用由廉价硬件构成的计算机集群实现海量数据的分布式存储
- 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，具有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
- 本章最后介绍了HDFS的数据读写过程以及HDFS编程实践方面的相关知识