《大数据技术原理与应用（第3版）》

http://dblab.xmu.edu.cn/post/bigdata

温馨提示：编辑幻灯片母版，可以修改每页PPT的厦大校徽和底部文字

第3章 分布式文件系统HDFS

（PPT版本号：2020年12月版本）

林子雨 博士/副教授

厦门大学计算机科学系

E-mail: ziyulin@xmu.edu.cn

主页：http://dblab.xmu.edu.cn/post/linziyu

本章配套教学视频

《大数据技术原理与应用（第3版）》
在线视频观看地址

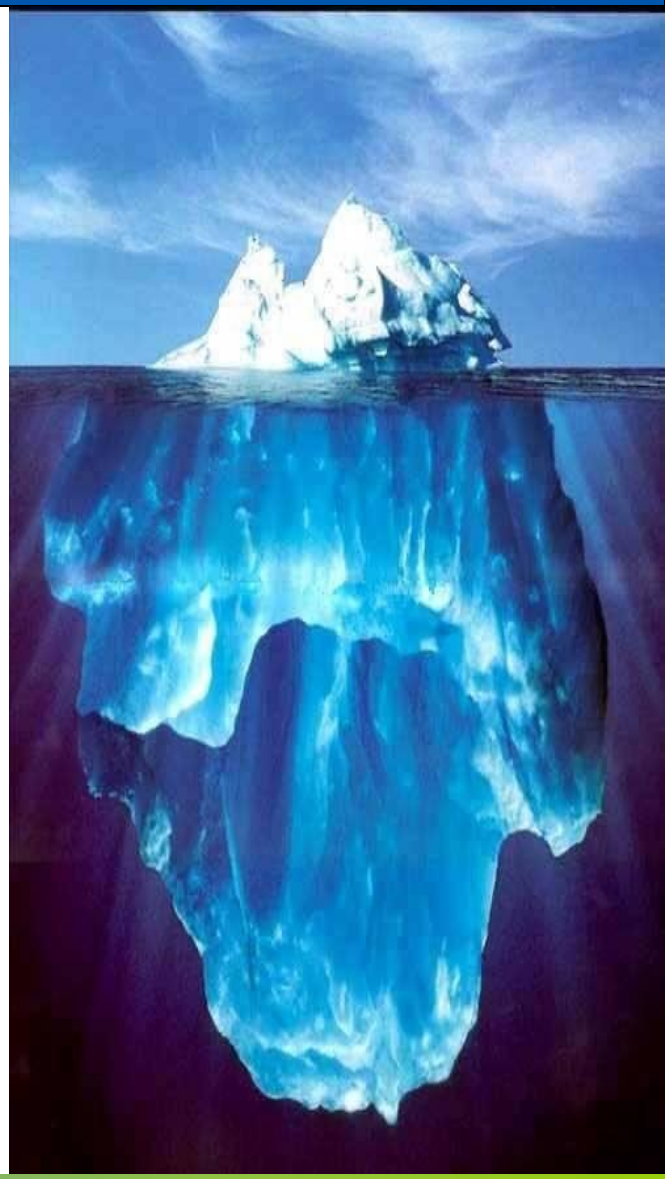
http://www.icourse163.org/course/XMU-1002335004

提纲

- 3.1 分布式文件系统
- 3.2 HDFS简介
- 3.3 HDFS相关概念
- 3.4 HDFS体系结构
- 3.5 HDFS存储原理
- 3.6 HDFS数据读写过程
- 3.7 HDFS编程实践

本PPT是如下教材的配套讲义：
《大数据技术原理与应用
——概念、存储、处理、分析与应用》
（2021年1月第3版）
ISBN:978-7-115-54405-6
厦门大学 林子雨 编著，人民邮电出版社

欢迎访问《大数据技术原理与应用》教材官方网站：
http://dblab.xmu.edu.cn/post/bigdata3



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

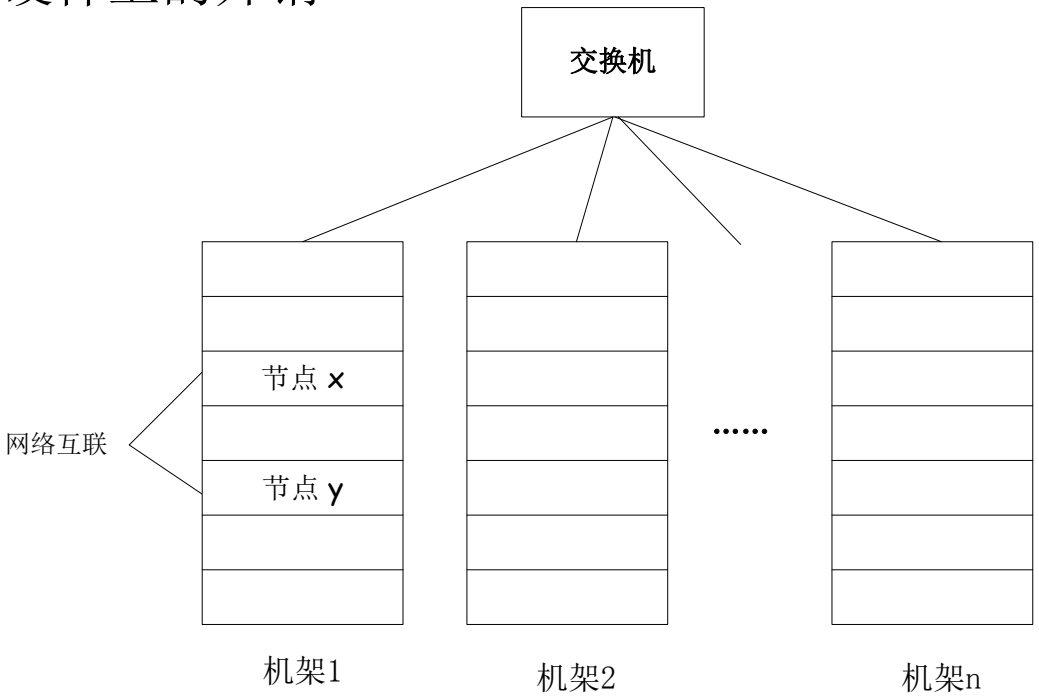


图3-1 计算机集群的基本架构

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，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)

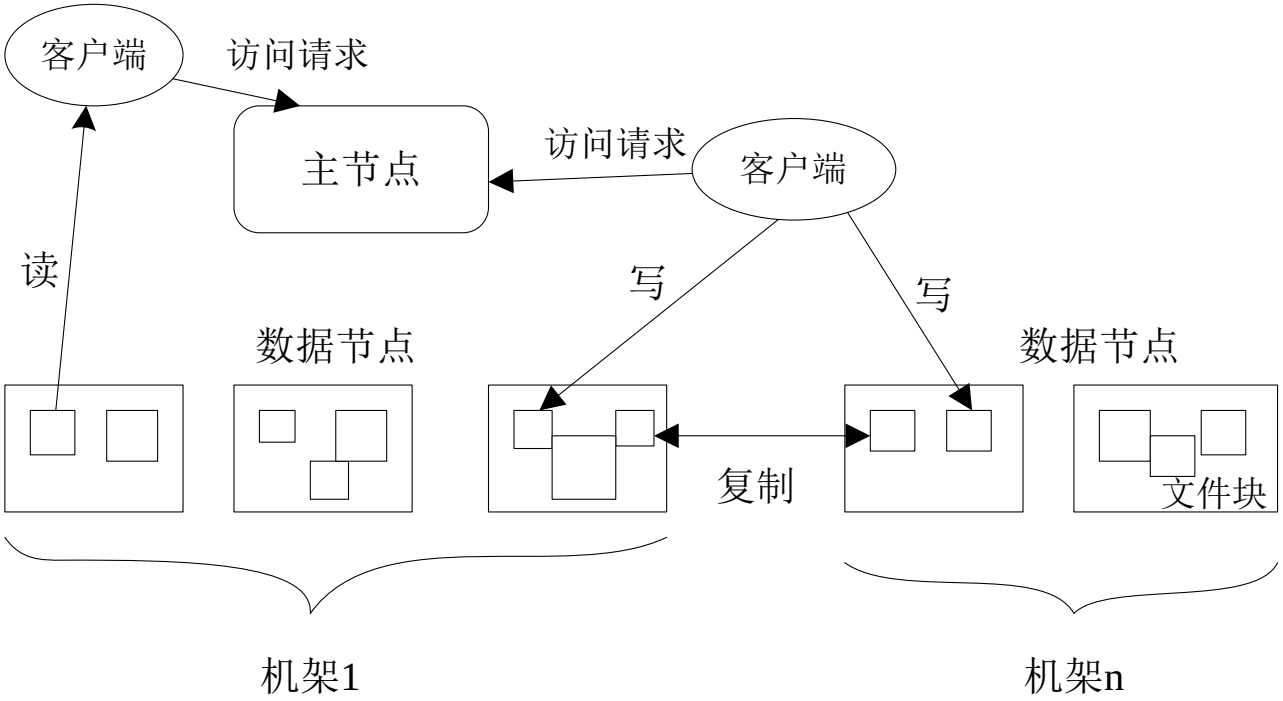


图3-2 大规模文件系统的整体结构

3.2 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：

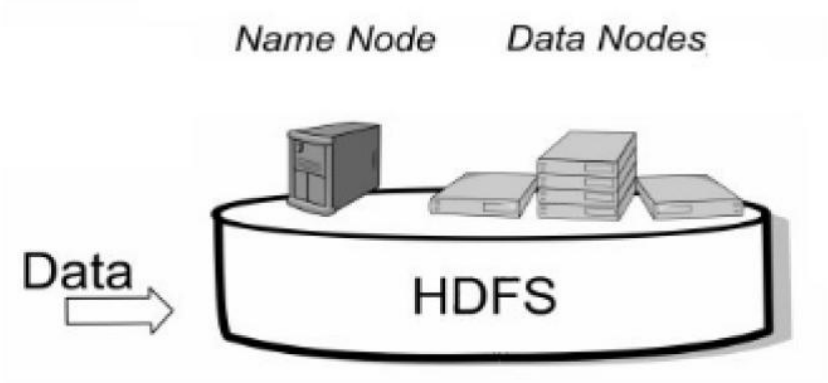
- **支持大规模文件存储**：文件以块为单位进行存储，一个大规模文件可以被分拆成若干个文件块，不同的文件块可以被分发到不同的节点上，因此，一个文件的大小不会受到单个节点的存储容量的限制，可以远远大于网络中任意节点的存储容量

- **简化系统设计**：首先，大大简化了存储管理，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

- 在HDFS中，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用于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中记录了所有针对文件的创建、删除、重命名等操作
- 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

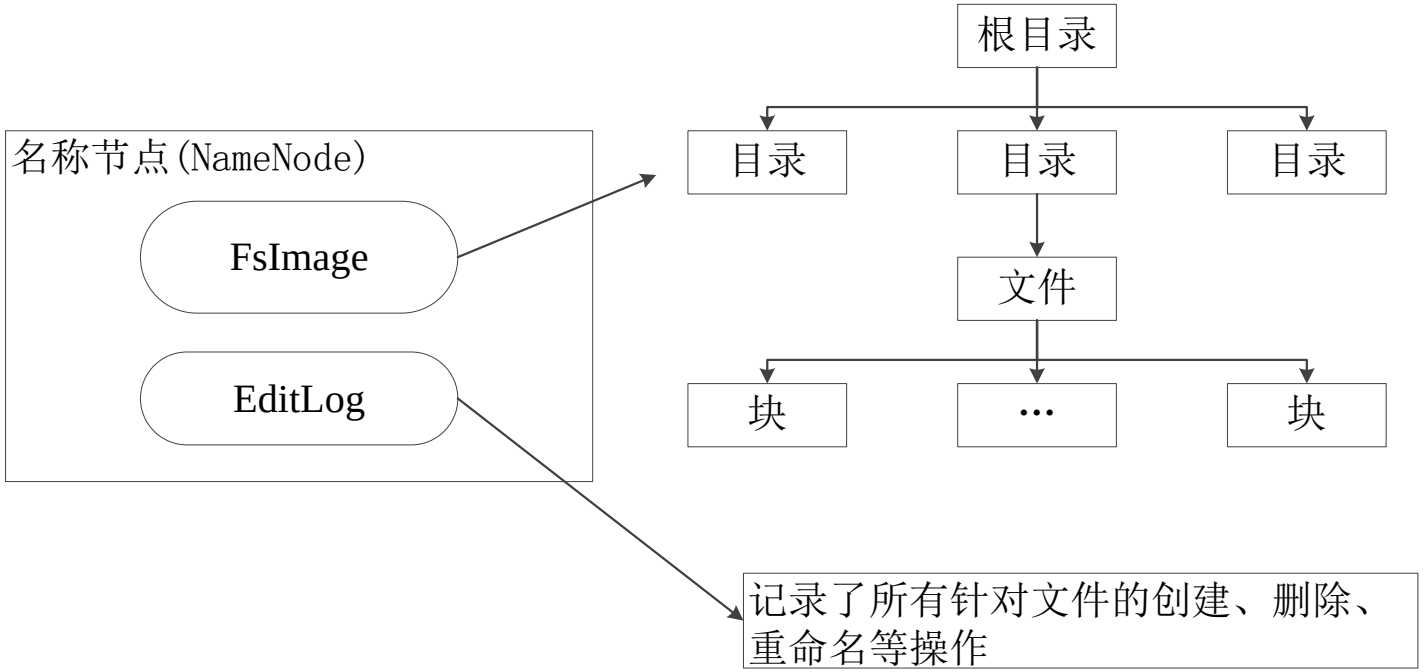


图3-3 名称节点的数据结构

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，并包含此类信息：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文件没有记录每个块存储在哪个数据节点。而是由名称节点把这些映射信息保留在内存中，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，使得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

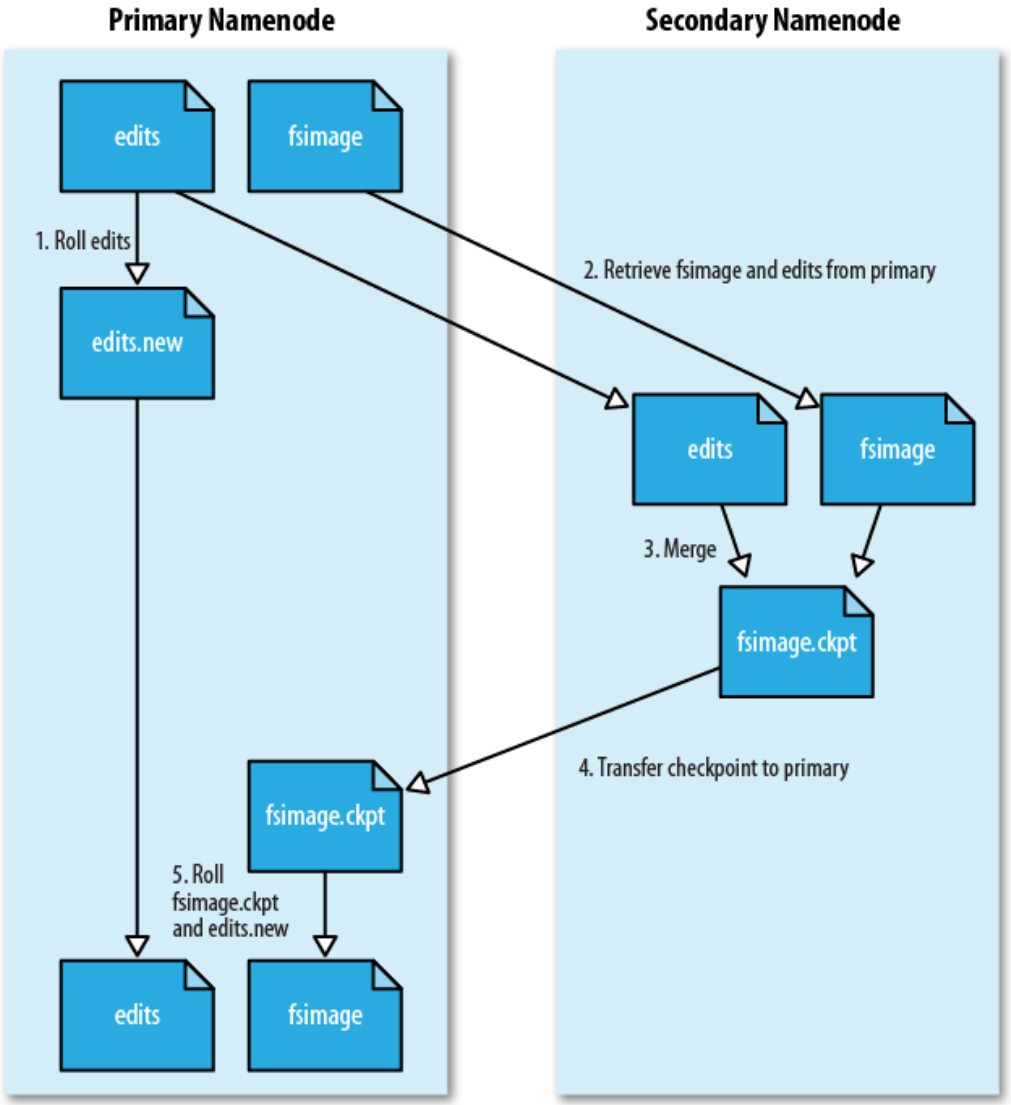
•在名称节点运行期间，HDFS的所有更新操作都是直接写到EditLog中，久而久之， EditLog文件将会变得很大

•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）（如图3-4所示）。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

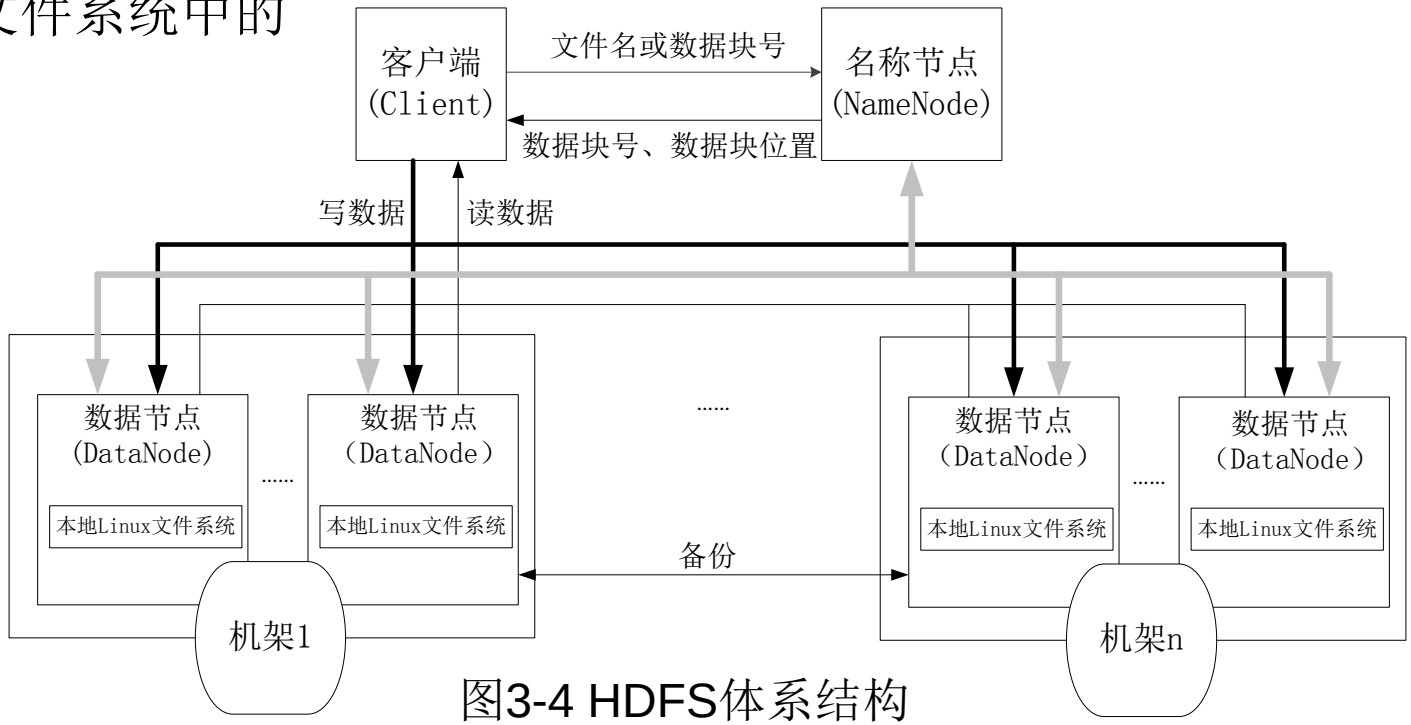


图3-4 HDFS体系结构

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，如图3-5所示，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。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

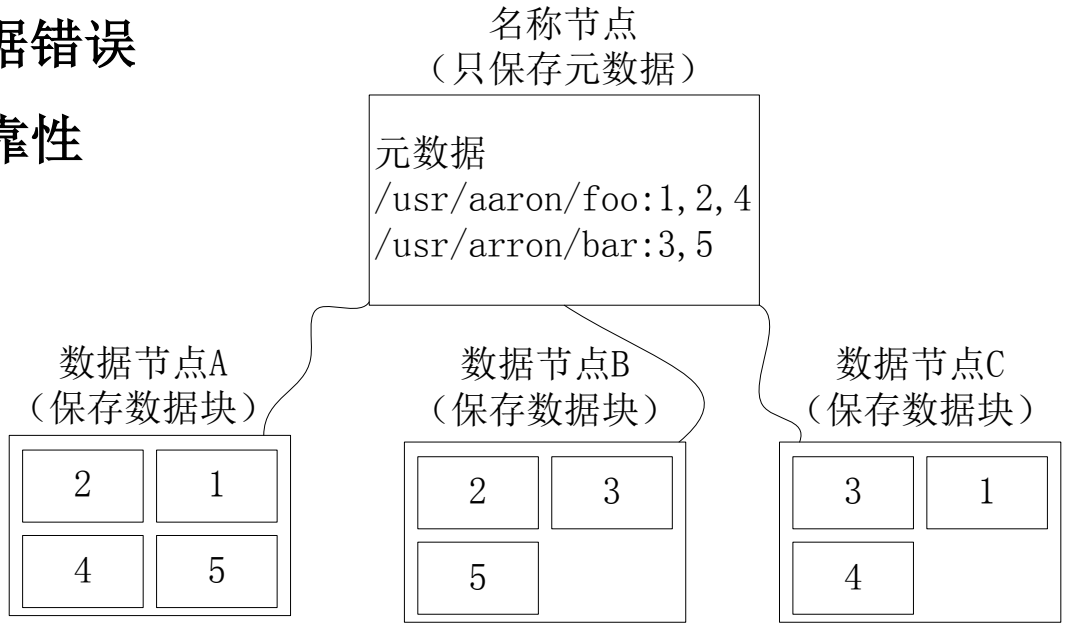


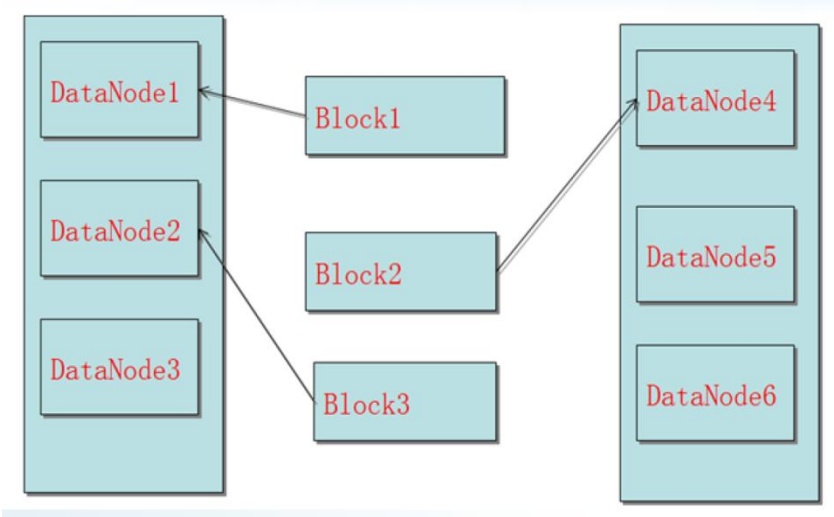
图3-5 HDFS数据块多副本存储

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

3.5.3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();
                        conf.set("fs.defaultFS","hdfs://localhost:9000");

conf.set("fs.hdfs.impl","org.apache.hadoop.hdfs.DistributedFileSystem");
                        FileSystem fs = FileSystem.get(conf);
                        Path file = new Path("test");
                        FSDataInputStream getIt = fs.open(file);
                        BufferedReader d = new BufferedReader(new
InputStreamReader(getIt));
                        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();
                        conf.set("fs.defaultFS","hdfs://localhost:9000");
                        conf.set("fs.hdfs.impl","org.apache.hadoop.hdfs.DistributedFileSystem");
                        FileSystem fs = FileSystem.get(conf);
                        byte[] buff = "Hello world".getBytes(); // 要写入的内容
                        String filename = "test"; //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);
                        os.close();
                        fs.close();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

3.6 HDFS数据读写过程

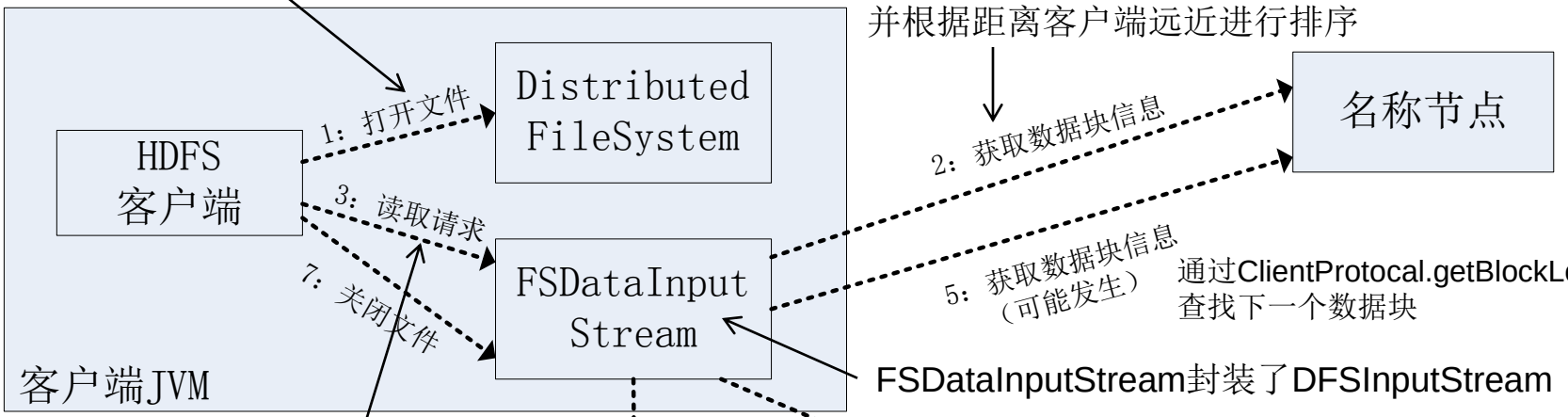
•FileSystem是一个通用文件系统的抽象基类，可以被分布式文件系统继承，所有可能使用Hadoop文件系统的代码，都要使用这个类
•Hadoop为FileSystem这个抽象类提供了多种具体实现
•DistributedFileSystem就是FileSystem在HDFS文件系统中的具体实现
•FileSystem的open()方法返回的是一个输入流FSDataInputStream对象，在HDFS文件系统中，具体的输入流就是DFSInputStream；FileSystem中的create()方法返回的是一个输出流FSDataOutputStream对象，在HDFS文件系统中，具体的输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

```
Configuration conf = new Configuration();
conf.set("fs.defaultFS","hdfs://localhost:9000");
conf.set("fs.hdfs.impl","org.apache.hadoop.hdfs.DistributedFileSystem");
FileSystem fs = FileSystem.get(conf);
FSDataInputStream in = fs.open(new Path(uri));
FSDataOutputStream out = fs.create(new Path(uri));
```

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



通过ClientProtocal.getBlockLocations()
查找下一个数据块

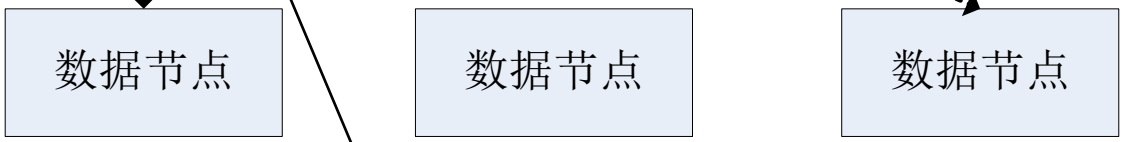
FSDataInputStream封装了DFSInputStream

客户端节点

4:读取数据

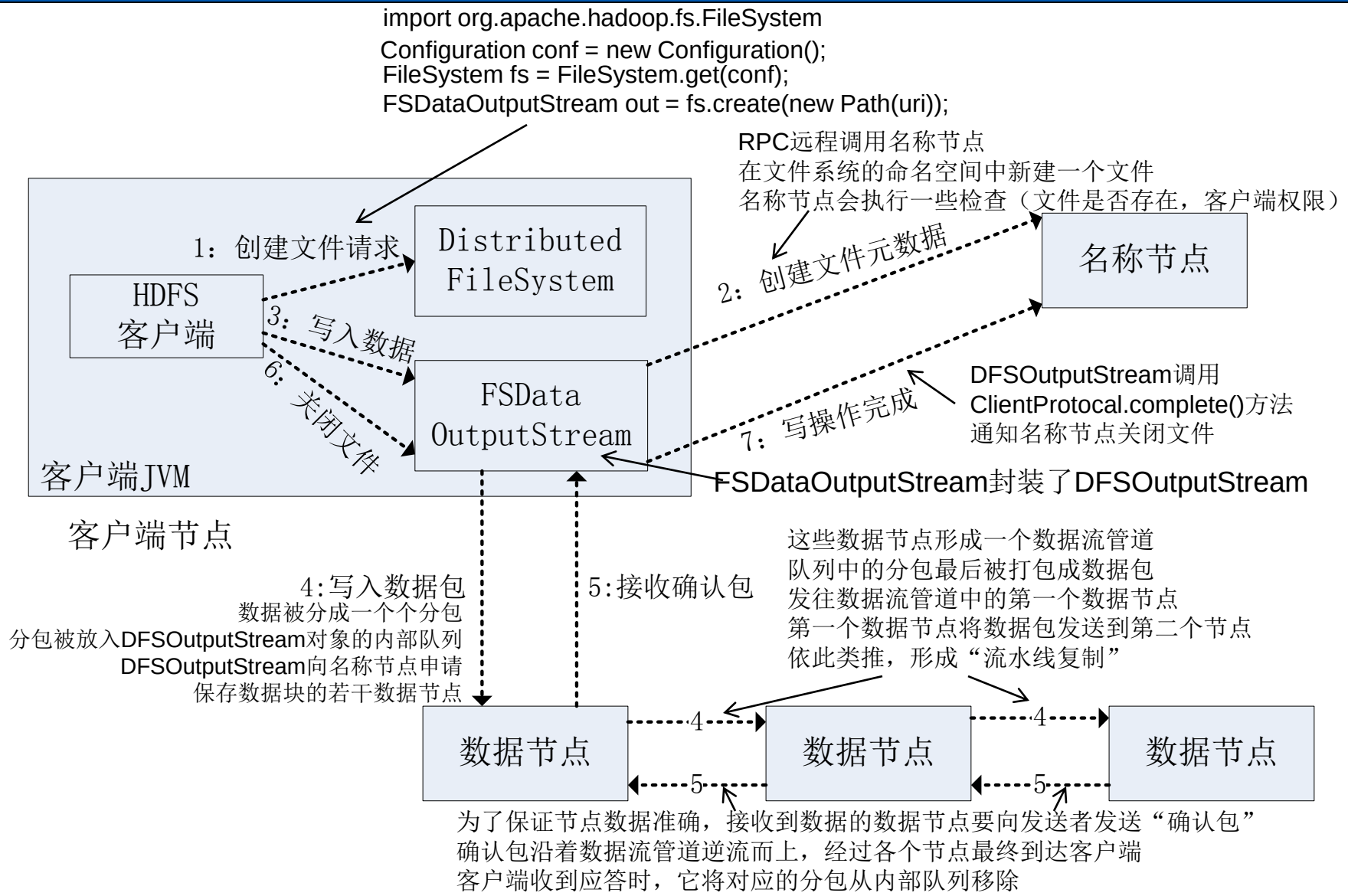
6:读取数据

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
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
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
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点
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
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

学习HDFS编程实践，具体请参见厦门大学数据实验室建设的高校大数据课程公共服务平台上的技术文章：

《大数据技术原理与应用（第3版） 第三章 分布式文件系统HDFS 学习指南》

访问地址： http://dblab.xmu.edu.cn/blog/2460-2/

高校大数据课程

公 共 服 务 平 台

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，其中，fs命令可以说是HDFS最常用的命令

利用该命令可以查看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

```
hadoop@ubuntu:/usr/local/hadoop$ ./bin/hadoop fs -mkdir
hdfs://localhost:9000/tempDir
hadoop@ubuntu:/usr/local/hadoop$ ./bin/hadoop fs -ls hdf
s://localhost:9000/
Found 6 items
drwxr-xr-x   - hadoop supergroup          0 2020-02-06 1
0:04 hdfs://localhost:9000/bigdatacase
drwxr-xr-x   - hadoop supergroup          0 2020-02-19 1
9:39 hdfs://localhost:9000/hbase
drwxr-xr-x   - hadoop supergroup          0 2020-02-27 1
0:57 hdfs://localhost:9000/tempDir
drwx-wx-wx   - hadoop supergroup          0 2020-01-30 0
9:38 hdfs://localhost:9000/tmp
drwxr-xr-x   - hadoop supergroup          0 2020-01-30 1
9:13 hdfs://localhost:9000/user
drwxr-xr-x   - hadoop supergroup          0 2020-01-30 1
9:21 hdfs://localhost:9000/usr
```

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

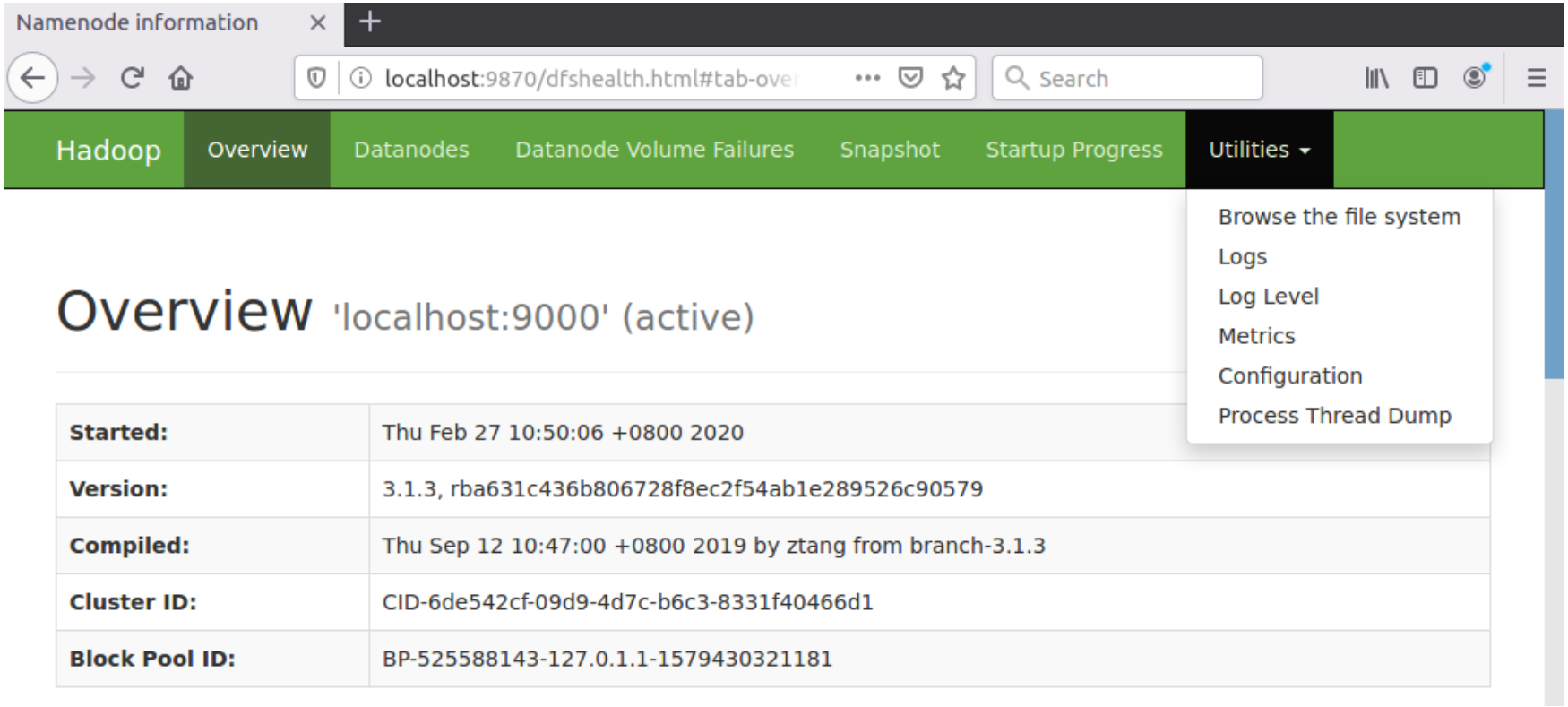
```
hadoop@ubuntu:/usr/local/hadoop$ ./bin/hadoop fs -copyFromLocal /home/ha
doop/mergefile/* hdfs://localhost:9000/tempDir
```

```
hadoop@ubuntu:/usr/local/hadoop$ ./bin/hadoop fs -cat hdfs://localhost:9
000/tempDir/*2020-02-27 11:08:25,938 INFO sasl.SaslDataTransferClient: S
ASL encryption trust check: localHostTrusted = false, remoteHostTrusted
= false
this is file1.txt
this is file2.txt
this is file3.txt
this is file4.abc
this is file5.abc
```

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://localhost:9870”访问HDFS文件系统

通过Web界面的”Utilities”菜单下面的“Browse the filesystem”查看文件



3.7.3HDFS常用Java API及应用实例

利用Java API与HDFS进行交互

实例：文件的过滤与合并

准备工作：在Ubuntu系统中安装开发工具Eclipse

具体请参见：

《大数据技术原理与应用（第3版） 第三章 分布式文件系统HDFS 学习指南》

访问地址： http://dblab.xmu.edu.cn/blog/2460-2/

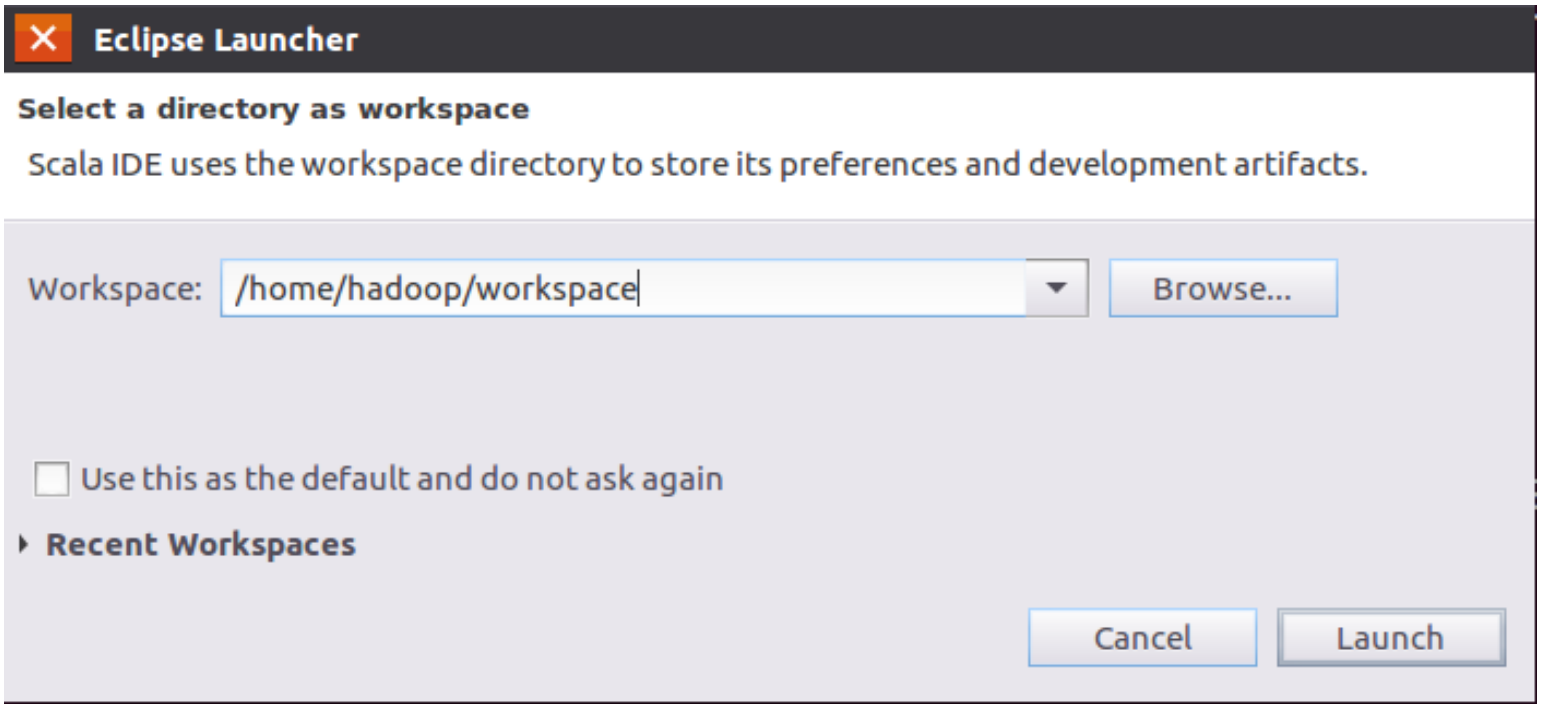
3.7.3HDFS常用Java API及应用实例

现在要执行的任务是：假设在目录“hdfs://localhost:9000/user/hadoop”下面有几个文件，分别是file1.txt、file2.txt、file3.txt、file4.abc和file5.abc，这里需要从该目录中过滤出所有后缀名不为“.abc”的文件，对过滤之后的文件进行读取，并将这些文件的内容合并到文件“hdfs://localhost:9000/user/hadoop/merge.txt”中。

3.7.3HDFS常用Java API及应用实例

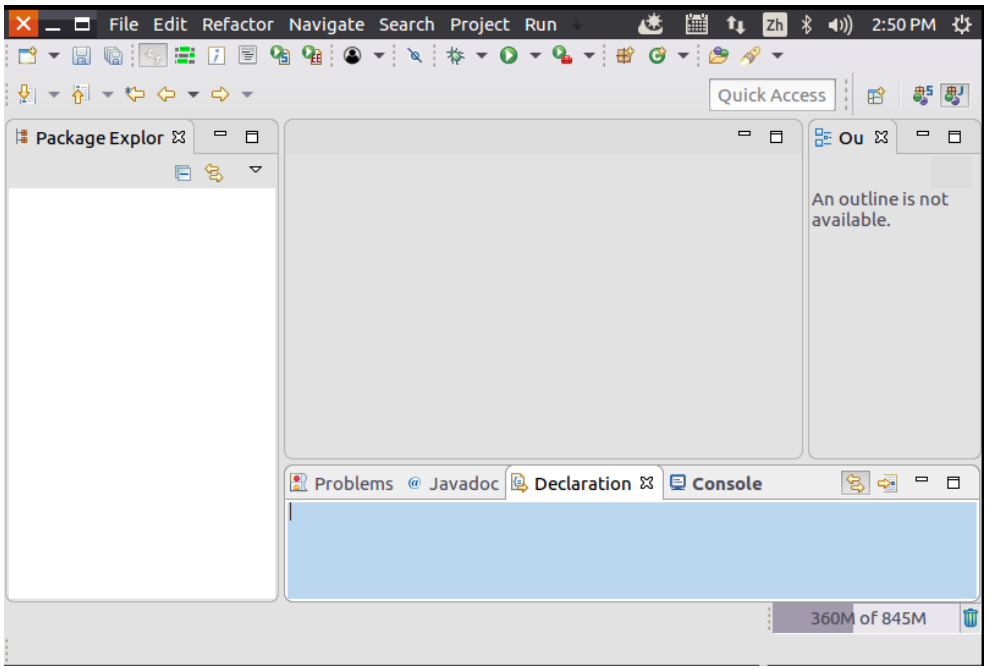
一、在Eclipse中创建项目

启动Eclipse。当Eclipse启动以后，会弹出如下图所示界面，提示设置工作空间（workspace）。



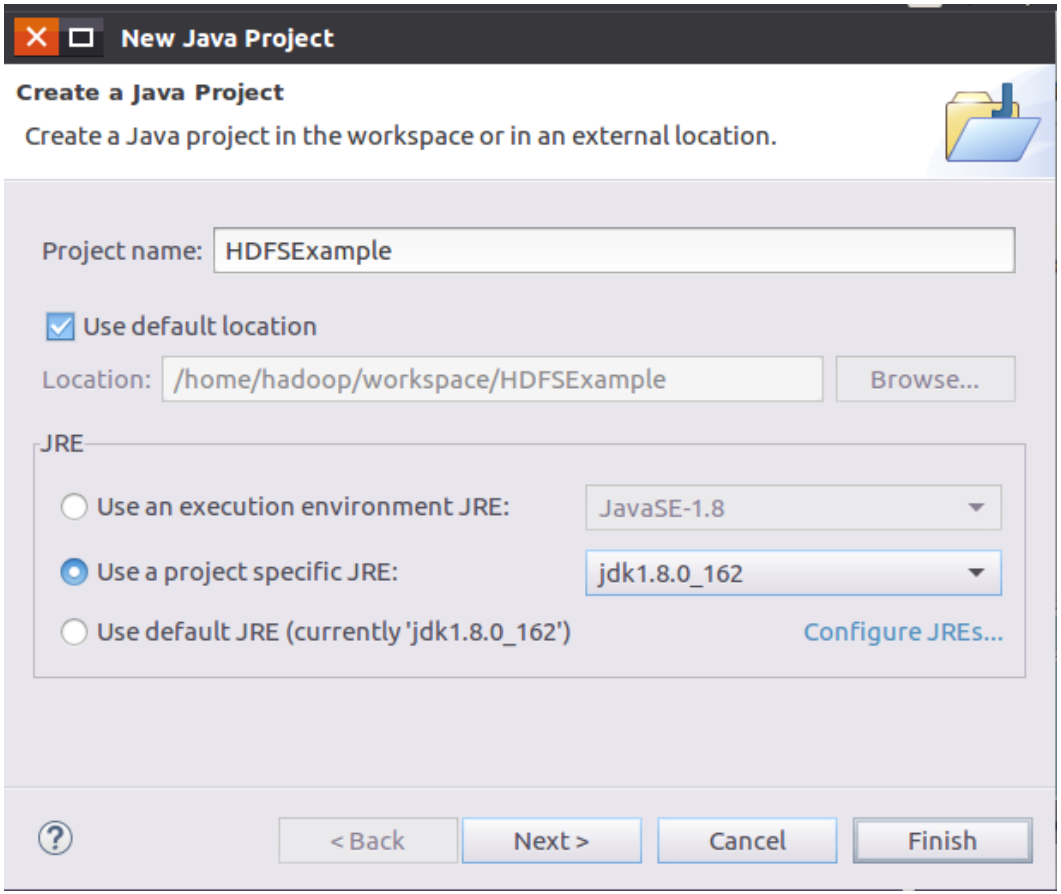
3.7.3HDFS常用Java API及应用实例

可以直接采用默认的设置“/home/hadoop/workspace”，点击“Launch”按钮。可以看出，由于当前是采用hadoop用户登录了Linux系统，因此，默认的工作空间目录位于hadoop用户目录“/home/hadoop”下。
Eclipse启动以后，会呈现如下图所示的界面。



3.7.3HDFS常用Java API及应用实例

选择“File->New->Java Project”菜单，开始创建一个Java工程，会弹出如下图所示界面。



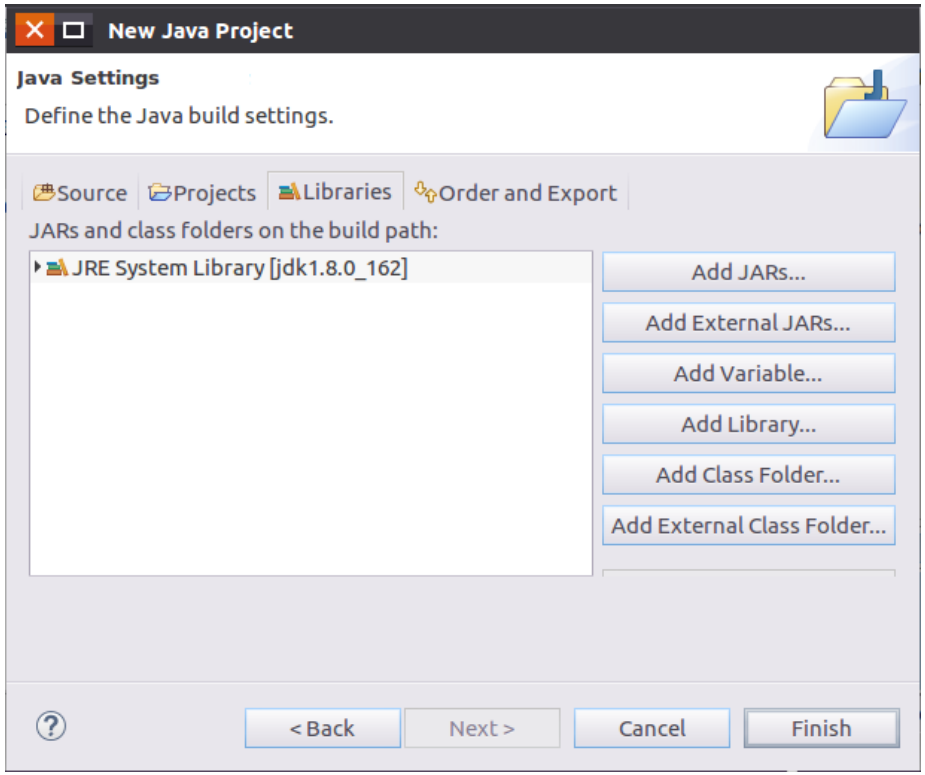
3.7.3HDFS常用Java API及应用实例

在“Project name”后面输入工程名称“HDFSExample”，选中“Use default location”，让这个Java工程的所有文件都保存到“/home/hadoop/workspace/HDFSExample”目录下。在“JRE”这个选项卡中，可以选择当前的Linux系统中已经安装好的JDK，比如jdk1.8.0_162。然后，点击界面底部的“Next>”按钮，进入下一步的设置。

3.7.3HDFS常用Java API及应用实例

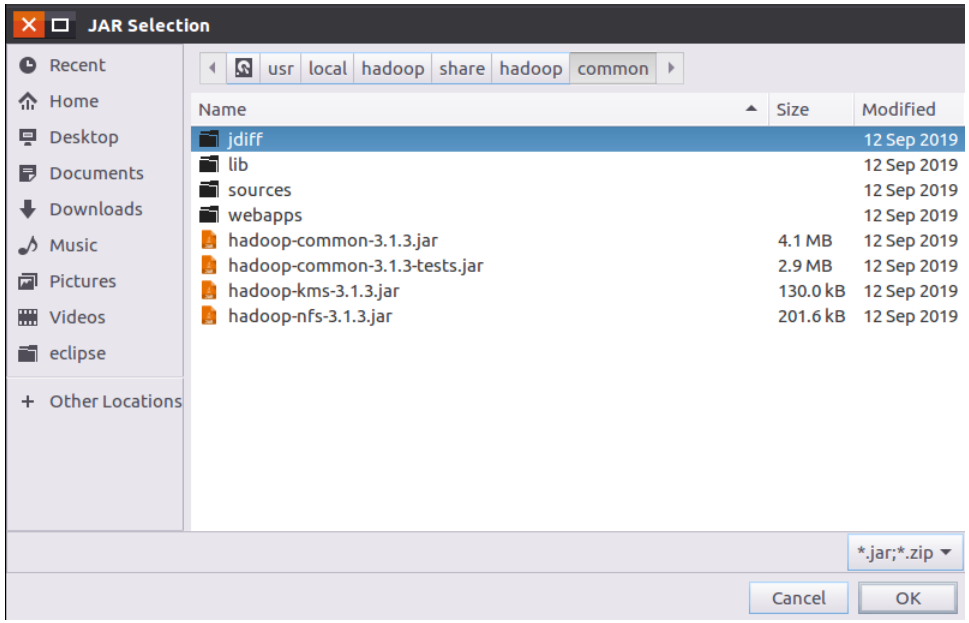
二、为项目添加需要用到的JAR包

进入下一步的设置以后，会弹出如下图所示界面。



3.7.3HDFS常用Java API及应用实例

需要在这个界面中加载该Java工程所需要用到的JAR包，这些JAR包中包含了可以访问HDFS的Java API。这些JAR包都位于Linux系统的Hadoop安装目录下，对于本教程而言，就是在“/usr/local/hadoop/share/hadoop”目录下。点击界面中的“Libraries”选项卡，然后，点击界面右侧的“Add External JARs…”按钮，会弹出如下图所示界面。



3.7.3HDFS常用Java API及应用实例

在该界面中，上面的一排目录按钮（即“usr”、“local”、“hadoop”、“share”、“hadoop”和“common”），当点击某个目录按钮时，就会在下面列出该目录的内容。

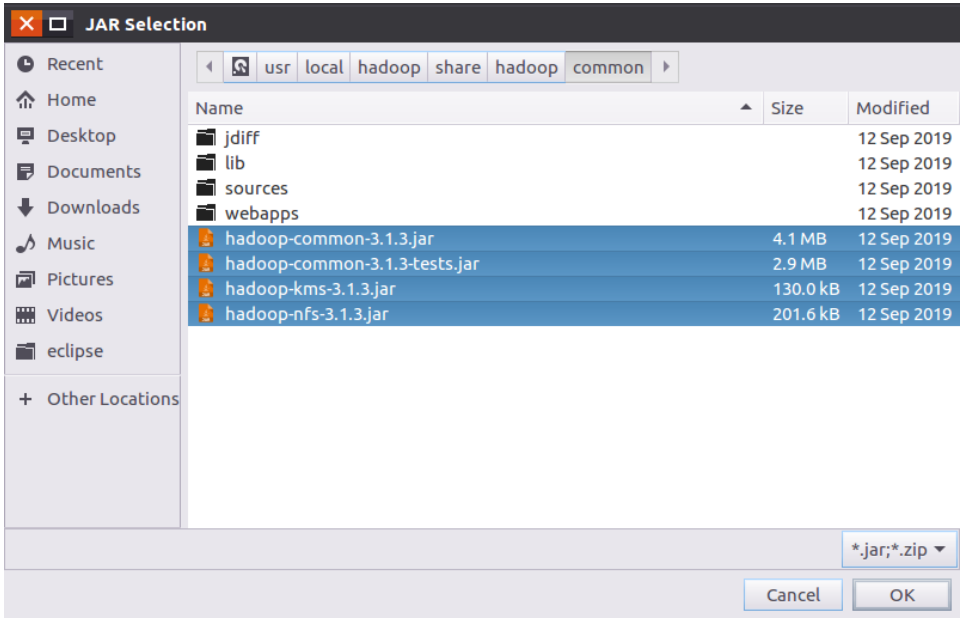
为了编写一个能够与HDFS交互的Java应用程序，一般需要向Java工程中添加以下JAR包：

- “/usr/local/hadoop/share/hadoop/common”目录下的所有JAR包，包括hadoop-common-3.1.3.jar、hadoop-common-3.1.3-tests.jar、haoop-nfs-3.1.3.jar和haoop-kms-3.1.3.jar，注意，不包括目录jdiff、lib、sources和webapps；
- “/usr/local/hadoop/share/hadoop/common/lib”目录下的所有JAR包；
- “/usr/local/hadoop/share/hadoop/hdfs”目录下的所有JAR包，注意，不包括目录jdiff、lib、sources和webapps；

“/usr/local/hadoop/share/hadoop/hdfs/lib”目录下的所有JAR包。

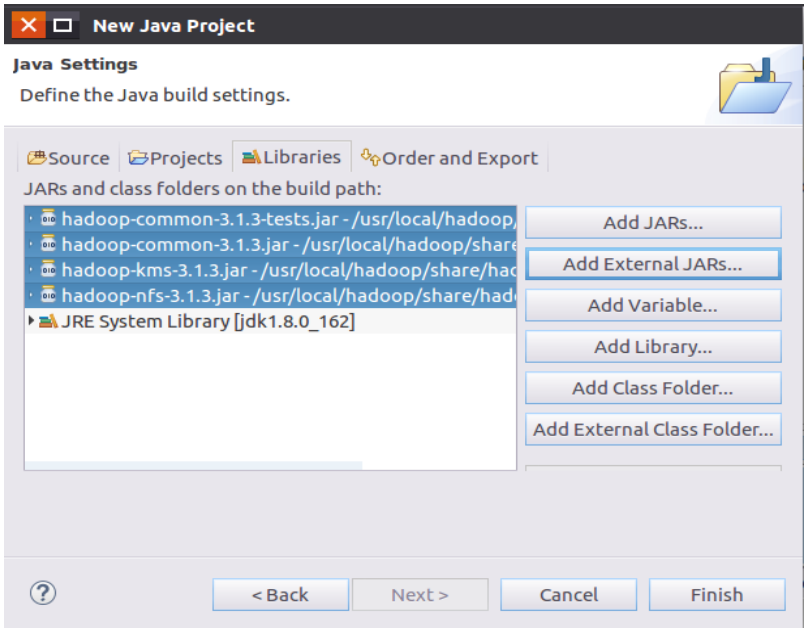
3.7.3HDFS常用Java API及应用实例

比如，如果要把“/usr/local/hadoop/share/hadoop/common”目录下的hadoop-common-3.1.3.jar、hadoop-common-3.1.3-tests.jar、haoop-nfs-3.1.3.jar和haoop-kms-3.1.3.jar添加到当前的Java工程中，可以在界面中点击目录按钮，进入到common目录，然后，界面会显示出common目录下的所有内容（如下图所示）。



3.7.3HDFS常用Java API及应用实例

请在界面中用鼠标点击选中hadoop-common-3.1.3.jar、hadoop-common-3.1.3-tests.jar、haoop-nfs-3.1.3.jar和haoop-kms-3.1.3.jar（不要选中目录jdiff、lib、sources和webapps），然后点击界面右下角的“确定”按钮，就可以把这两个JAR包增加到当前Java工程中，出现的界面如下图所示。



3.7.3HDFS常用Java API及应用实例

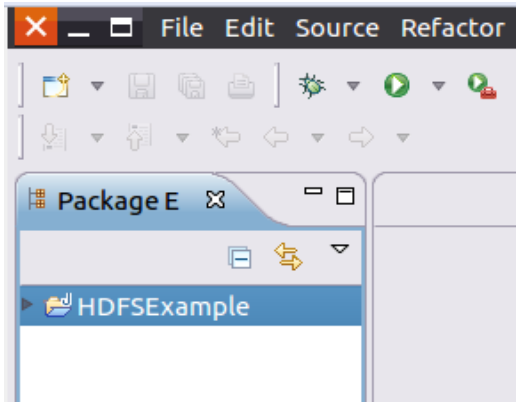
从这个界面中可以看出，hadoop-common-3.1.3.jar、hadoop-common-3.1.3-tests.jar、haoop-nfs-3.1.3.jar和haoop-kms-3.1.3.jar已经被添加到当前Java工程中。然后，按照类似的操作方法，可以再次点击“Add External JARs…”按钮，把剩余的其他JAR包都添加进来。需要注意的是，当需要选中某个目录下的所有JAR包时，可以使用“Ctrl+A”组合键进行全选操作。全部添加完毕以后，就可以点击界面右下角的“Finish”按钮，完成Java工程HDFSExample的创建。

3.7.3HDFS常用Java API及应用实例

三、编写Java应用程序

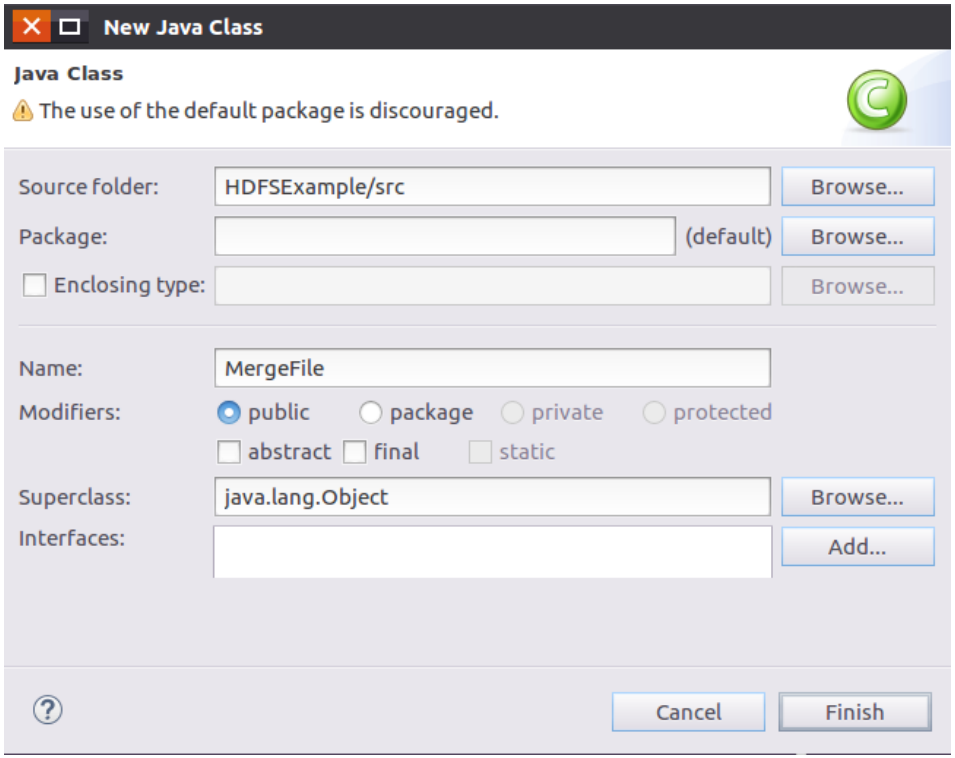
下面编写一个Java应用程序，用来检测HDFS中是否存在一个文件。

请在Eclipse工作界面左侧的“Package Explorer”面板中（如下图所示），找到刚才创建好的工程名称“HDFSExample”，然后在该工程名称上点击鼠标右键，在弹出的菜单中选择“New→Class”菜单。



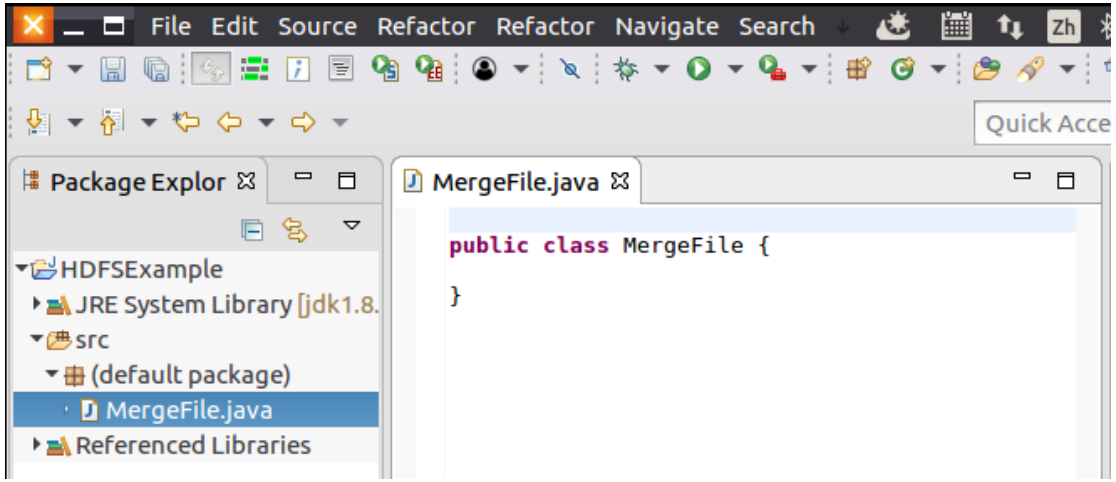
3.7.3HDFS常用Java API及应用实例

选择“New→Class”菜单以后会出现如下图所示界面。



3.7.3HDFS常用Java API及应用实例

在该界面中，只需要在“Name”后面输入新建的Java类文件的名称，这里采用名称“MergeFile”，其他都可以采用默认设置，然后，点击界面右下角“Finish”按钮，出现如下图所示界面。



3.7.3HDFS常用Java API及应用实例

可以看出，Eclipse自动创建了一个名为“MergeFile.java”的源代码文件，请在该文件中输入以下代码：

```
import java.io.IOException;
import java.io.PrintStream;
import java.net.URI;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;

/**
 * 过滤掉文件名满足特定条件的文件
 */
class MyPathFilter implements PathFilter {
    String reg = null;
    MyPathFilter(String reg) {
        this.reg = reg;
    }
    public boolean accept(Path path) {
                                                    if (!(path.toString().matches(reg)))
                                                                                    return true;
                                                    return false;
                                    }
}
/***
 * 利用FSDataOutputStream和FSDataInputStream合并HDFS中的文件
 */
public class MergeFile {
                                    Path inputPath = null; //待合并的文件所在的目录的路径
                                    Path outputPath = null; //输出文件的路径
                                    public MergeFile(String input, String output) {
                                                    this.inputPath = new Path(input);
                                                    this.outputPath = new Path(output);
                                    }
                                    public void doMerge() throws IOException {
                                                    Configuration conf = new Configuration();
                                                    conf.set("fs.defaultFS","hdfs://localhost:9000");
        conf.set("fs.hdfs.impl","org.apache.hadoop.hdfs.DistributedFileSystem");
                                                    FileSystem fsSource = FileSystem.get(URI.create(inputPath.toString()), conf);
                                                    FileSystem fsDst = FileSystem.get(URI.create(outputPath.toString()), conf);
                                                                                    //下面过滤掉输入目录中后缀为.abc的文件
                                                    FileStatus[] sourceStatus = fsSource.listStatus(inputPath,
                                                                                    new MyPathFilter(".*\\.abc"));
                                                    FSDataOutputStream fsdos = fsDst.create(outputPath);
                                                    PrintStream ps = new PrintStream(System.out);
                                                    //下面分别读取过滤之后的每个文件的内容，并输出到同一个文件中
                                                    for (FileStatus sta : sourceStatus) {
                                                                                    //下面打印后缀不为.abc的文件的路径、文件大小
                                                                                    System.out.print("路径：" + sta.getPath() + "    文件大小：" + sta.getLen()
                                                                                                                                    + "   权限：" + sta.getPermission() + "   内容：");
                                                                                    FSDataInputStream fsdis = fsSource.open(sta.getPath());
                                                                                    byte[] data = new byte[1024];
                                                                                    int read = -1;

                                                                                    while ((read = fsdis.read(data)) > 0) {
                                                                                                                    ps.write(data, 0, read);
                                                                                                                    fsdos.write(data, 0, read);
                                                                                    }
                                                                                    fsdis.close();
                                                    }
                                                    ps.close();
                                                    fsdos.close();
                                    }
                                    public static void main(String[] args) throws IOException {
                                                    MergeFile merge = new MergeFile(
                                                                                                                    "hdfs://localhost:9000/user/hadoop/",
                                                                                                                    "hdfs://localhost:9000/user/hadoop/merge.txt");
                                                    merge.doMerge();
                                    }
}
```

3.7.3HDFS常用Java API及应用实例

四、编译运行程序

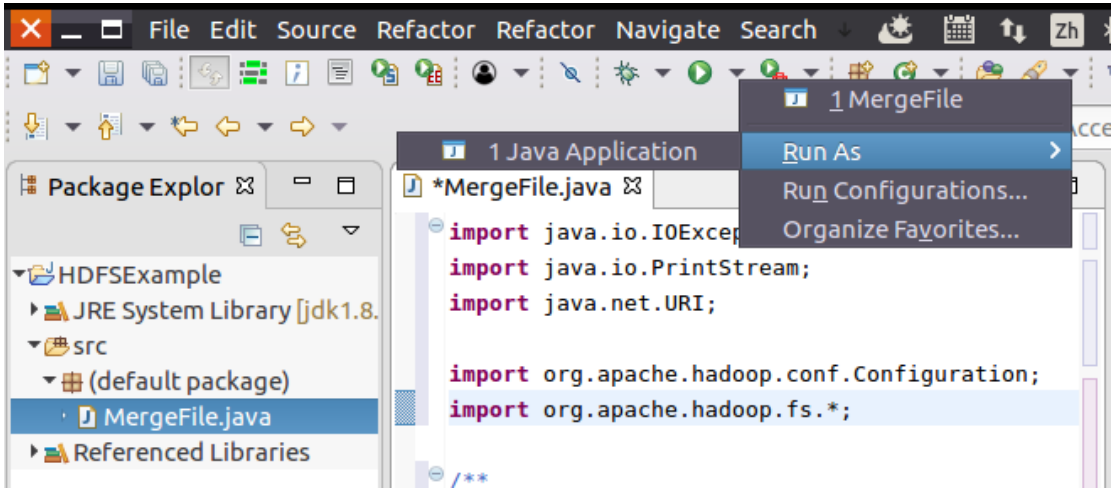
在开始编译运行程序之前，请一定确保Hadoop已经启动运行

然后，要确保HDFS的“/user/hadoop”目录下已经存在file1.txt、file2.txt、file3.txt、file4.abc和file5.abc，每个文件里面有内容。这里，假设文件内容如下表所示。

文件名称	文件内容
file1.txt	this is file1.txt
file2.txt	this is file2.txt
file3.txt	this is file3.txt
file4.abc	this is file4.abc
file5.abc	this is file5.abc

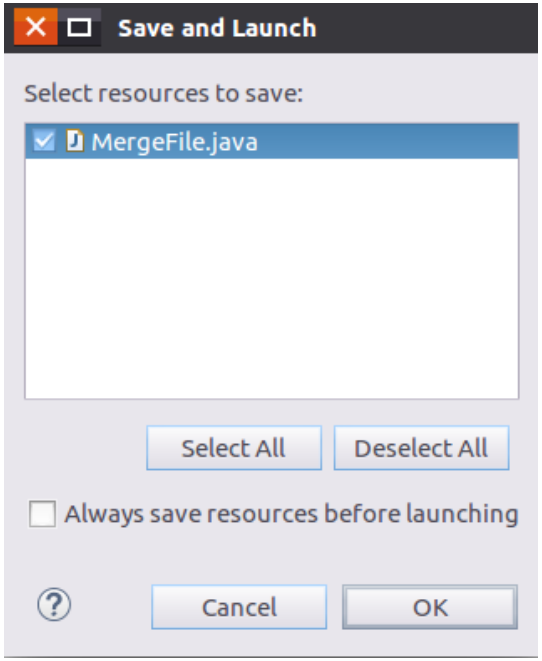
3.7.3HDFS常用Java API及应用实例

现在就可以编译运行上面编写的代码。可以直接点击Eclipse工作界面上部的运行程序的快捷按钮，当把鼠标移动到该按钮上时，在弹出的菜单中选择“Run As”，继续在弹出来的菜单中选择“Java Application”，如下图所示。



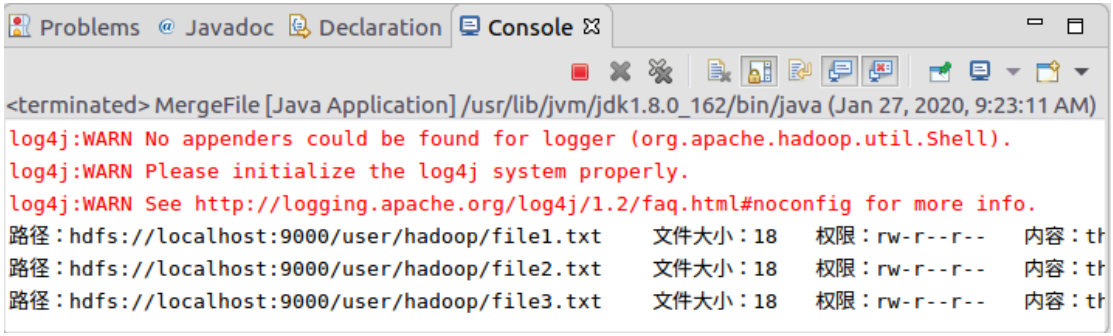
3.7.3HDFS常用Java API及应用实例

然后，会弹出如下图所示界面。



3.7.3HDFS常用Java API及应用实例

在该界面中，点击界面右下角的“OK”按钮，开始运行程序。程序运行结束后，会在底部的“Console”面板中显示运行结果信息（如下图所示）。同时，“Console”面板中还会显示一些类似“log4j:WARN…”的警告信息，可以不用理会。



3.7.3HDFS常用Java API及应用实例

如果程序运行成功，这时，可以到HDFS中查看生成的merge.txt文件，比如，可以在Linux终端中执行如下命令：

```
$ cd /usr/local/hadoop
$ ./bin/hdfs dfs -ls /user/hadoop
$ ./bin/hdfs dfs -cat /user/hadoop/merge.txt
```

可以看到如下结果：

```
this is file1.txt
this is file2.txt
this is file3.txt
```

3.7.3HDFS常用Java API及应用实例

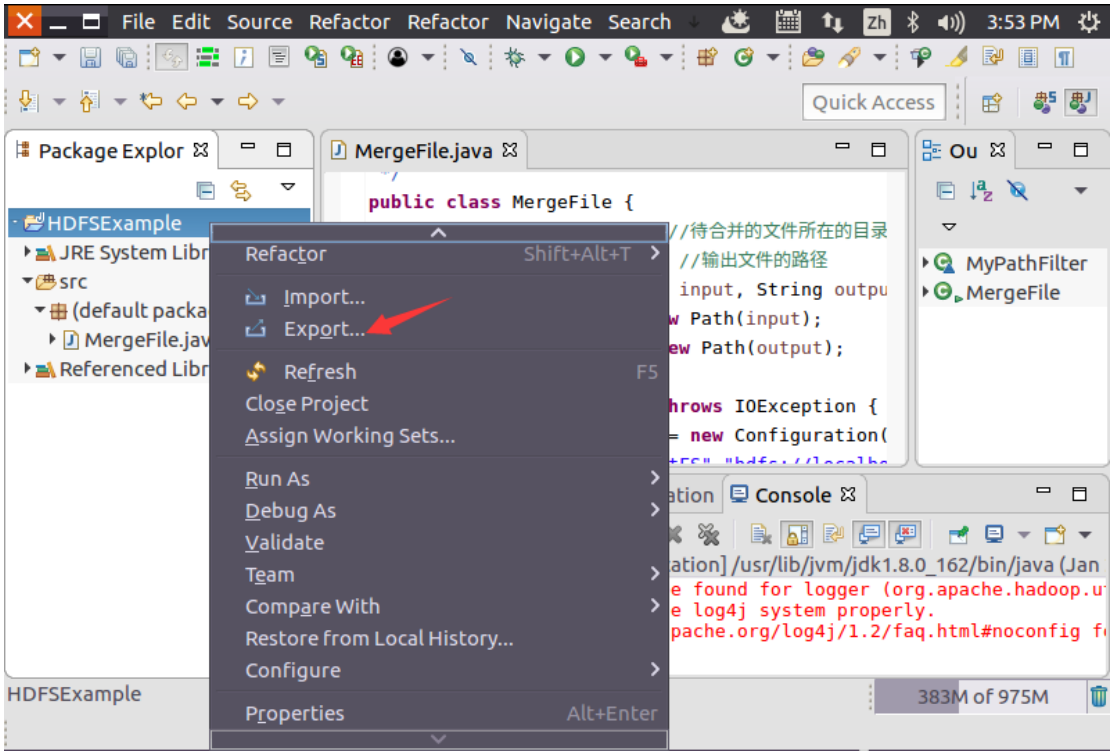
四、应用程序的部署

下面介绍如何把Java应用程序生成JAR包，部署到Hadoop平台上运行。首先，在Hadoop安装目录下新建一个名称为myapp的目录，用来存放我们自己编写的Hadoop应用程序，可以在Linux的终端中执行如下命令：

```
$ cd /usr/local/hadoop
$ mkdir myapp
```

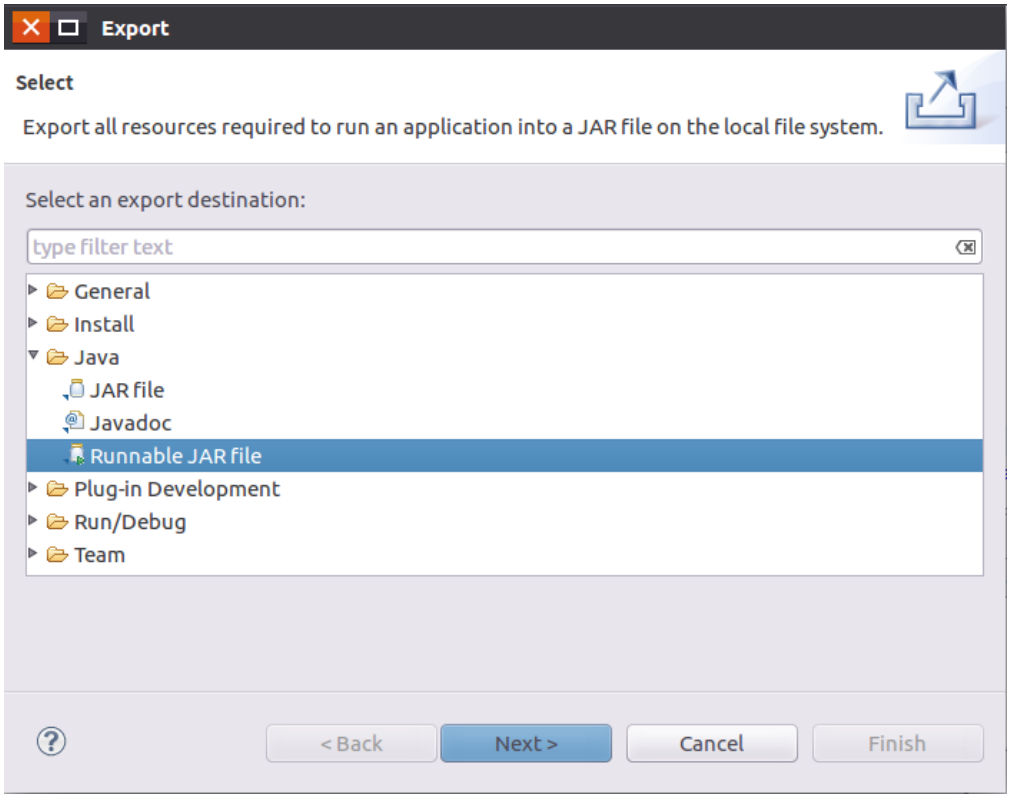
3.7.3HDFS常用Java API及应用实例

然后，请在Eclipse工作界面左侧的“Package Explorer”面板中，在工程名称“HDFSExample”上点击鼠标右键，在弹出的菜单中选择“Export”，如下图所示。



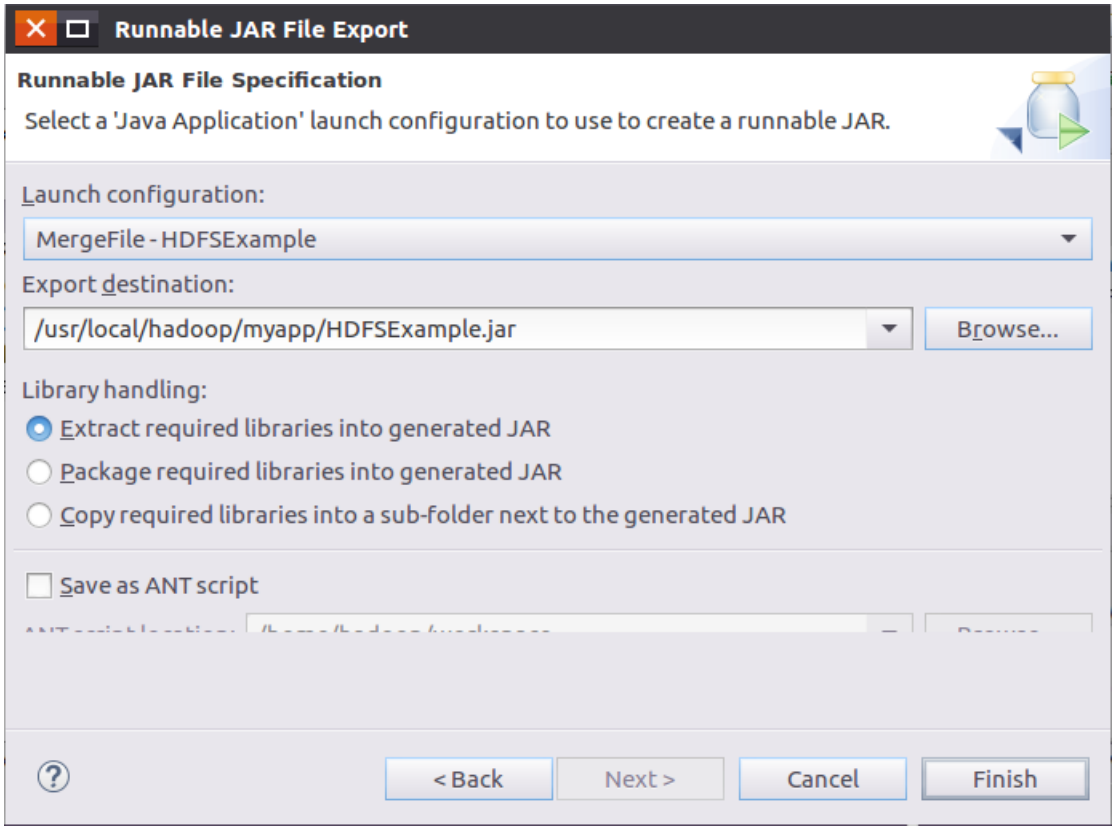
3.7.3HDFS常用Java API及应用实例

然后，会弹出如下图所示界面。



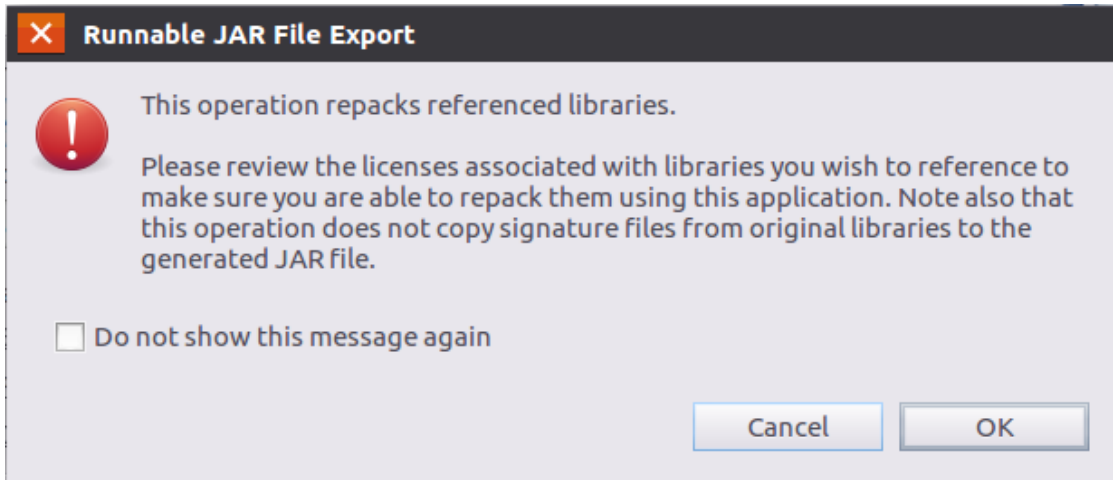
3.7.3HDFS常用Java API及应用实例

在该界面中，选择“Runnable JAR file”，然后，点击“Next>”按钮，弹出如下图所示界面。



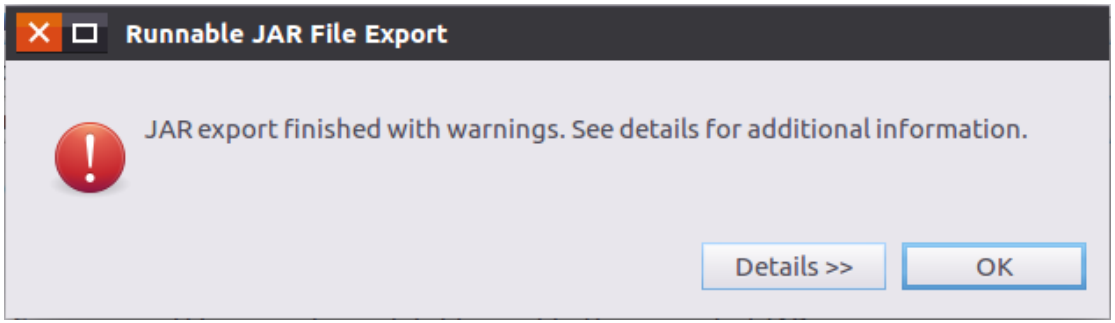
3.7.3HDFS常用Java API及应用实例

在该界面中，“Launch configuration”用于设置生成的JAR包被部署启动时运行的主类，需要在下拉列表中选择刚才配置的类“MergeFile-HDFSExample”。在“Export destination”中需要设置JAR包要输出保存到哪个目录，比如，这里设置为“/usr/local/hadoop/myapp/HDFSExample.jar”。在“Library handling”下面选择“Extract required libraries into generated JAR”。然后，点击“Finish”按钮，会出现如下图所示界面。



3.7.3HDFS常用Java API及应用实例

可以忽略该界面的信息，直接点击界面右下角的“OK”按钮，启动打包过程。打包过程结束后，会出现一个警告信息界面，如下图所示。



3.7.3HDFS常用Java API及应用实例

可以忽略该界面的信息，直接点击界面右下角的“OK”按钮。至此，已经顺利把HDFSExample工程打包生成了HDFSExample.jar。可以到Linux系统中查看一下生成的HDFSExample.jar文件，可以在Linux的终端中执行如下命令：

```
$ cd /usr/local/hadoop/myapp
$ ls
```

可以看到，“/usr/local/hadoop/myapp”目录下已经存在一个HDFSExample.jar文件。

3.7.3HDFS常用Java API及应用实例

由于之前已经运行过一次程序，已经生成了merge.txt，因此，需要首先执行如下命令删除该文件：

```
$ cd /usr/local/hadoop
$ ./bin/hdfs dfs -rm /user/hadoop/merge.txt
```

现在，就可以在Linux系统中，使用hadoop jar命令运行程序，命令如下：

```
$ cd /usr/local/hadoop
$ ./bin/hadoop jar ./myapp/HDFSExample.jar
```

3.7.3HDFS常用Java API及应用实例

上面程序执行结束以后，可以到HDFS中查看生成的merge.txt文件，比如，可以在Linux终端中执行如下命令：

```
$ cd /usr/local/hadoop
$ ./bin/hdfs dfs -ls /user/hadoop
$ ./bin/hdfs dfs -cat /user/hadoop/merge.txt
```

可以看到如下结果：

```
this is file1.txt
this is file2.txt
this is file3.txt
```

本章小结

- 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

附录A：主讲教师林子雨简介

主讲教师：林子雨

单位：厦门大学计算机科学系
E-mail: ziyulin@xmu.edu.cn
个人网页：http://dblab.xmu.edu.cn/post/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn

扫一扫访问个人主页

林子雨，男，1978年出生，博士（毕业于北京大学），全国高校知名大数据教师，现为厦门大学计算机科学系副教授，曾任厦门大学信息科学与技术学院院长助理、晋江市发展和改革局副局长。中国计算机学会数据库专业委员会委员，中国计算机学会信息系统专业委员会委员。国内高校首个“数字教师”提出者和建设者，厦门大学数据库实验室负责人，厦门大学云计算与大数据研究中心主要建设者和骨干成员，2013年度、2017年度和2020年度厦门大学教学类奖教金获得者，荣获2019年福建省精品在线开放课程、2018年厦门大学高等教育成果特等奖、2018年福建省高等教育教学成果二等奖、2018年国家精品在线开放课程。主要研究方向为数据库、数据仓库、数据挖掘、大数据、云计算和物联网，并以第一作者身份在《软件学报》《计算机学报》和《计算机研究与发展》等国家重点期刊以及国际学术会议上发表多篇学术论文。作为项目负责人主持的科研项目包括1项国家自然科学青年基金项目(No.61303004)、1项福建省自然科学青年基金项目(No.2013J05099)和1项中央高校基本科研业务费项目(No.2011121049)，主持的教改课题包括1项2016年福建省教改课题和1项2016年教育部产学协作育人项目，同时，作为课题负责人完成了国家发改委城市信息化重大课题、国家物联网重大应用示范工程区域试点泉州市工作方案、2015泉州市互联网经济调研等课题。中国高校首个“数字教师”提出者和建设者，2009年至今，“数字教师”大平台累计向网络免费发布超过1000万字高价值的研究和教学资料，累计网络访问量超过1000万次。打造了中国高校大数据教学知名品牌，编著出版了中国高校第一本系统介绍大数据知识的专业教材《大数据技术原理与应用》，并成为京东、当当网等网店畅销书籍；建设了国内高校首个大数据课程公共服务平台，为教师教学和学生学习大数据课程提供全方位、一站式服务，年访问量超过200万次，累计访问量超过1000万次。

附录B：大数据学习路线图

大数据学习路线图访问地址：http://dblab.xmu.edu.cn/post/10164/

附录C：林子雨大数据系列教材

了解全部教材信息：http://dblab.xmu.edu.cn/post/bigdatabook/

附录D：《大数据导论（通识课版）》教材

开设全校公共选修课的优质教材

本课程旨在实现以下几个培养目标：

- 引导学生步入大数据时代，积极投身大数据的变革浪潮之中
- 了解大数据概念，培养大数据思维，养成数据安全意识
- 认识大数据伦理，努力使自己的行为符合大数据伦理规范要求
- 熟悉大数据应用，探寻大数据与自己专业的应用结合点
- 激发学生基于大数据的创新创业热情

高等教育出版社 ISBN:978-7-04-053577-8 定价：32元 版次：2020年2月第1版
教材官网：http://dblab.xmu.edu.cn/post/bigdataintroduction/

附录E：《大数据导论》教材

- 林子雨 编著《大数据导论》
- 人民邮电出版社，2020年9月第1版
- ISBN:978-7-115-54446-9 定价：49.80元

教材官网：http://dblab.xmu.edu.cn/post/bigdata-introduction/

开设大数据专业导论课的优质教材

扫一扫访问教材官网

附录F：《大数据技术原理与应用（第3版）》教材

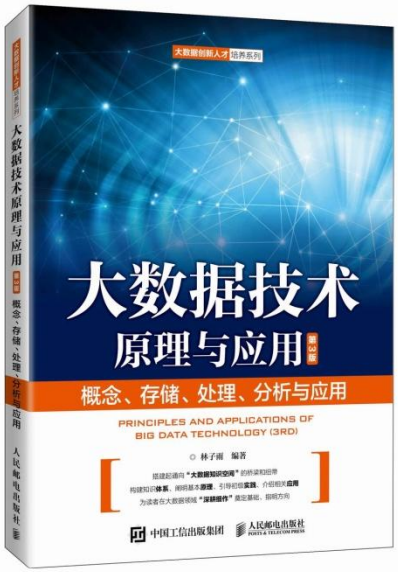
《大数据技术原理与应用——概念、存储、处理、分析与应用（第3版）》，由厦门大学计算机科学系林子雨博士编著，是国内高校第一本系统介绍大数据知识的专业教材。人民邮电出版社　ISBN:978-7-115-54405-6　定价：59.80元

全书共有17章，系统地论述了大数据的基本概念、大数据处理架构Hadoop、分布式文件系统HDFS、分布式数据 库HBase、NoSQL数据库、云数据库、分布式并行编程模型MapReduce、Spark、流计算、Flink、图计算、数据可视化以及大数据在互联网、生物医学和物流等各个领域的应用。在Hadoop、HDFS、HBase、MapReduce、Spark和Flink等重要章节，安排了入门级的实践操作，让读者更好地学习和掌握大数据关键技术。

本书可以作为高等院校计算机专业、信息管理等相关专业的大数据课程教材，也可供相关技术人员参考、学习、培训之用。

欢迎访问《大数据技术原理与应用——概念、存储、处理、分析与应用》教材官方网站：
http://dblab.xmu.edu.cn/post/bigdata3

扫一扫访问教材官网



附录G：《大数据基础编程、实验和案例教程（第2版）》

本书是与《大数据技术原理与应用（第3版）》教材配套的唯一指定实验指导书

大数据教材

1+1黄金组合

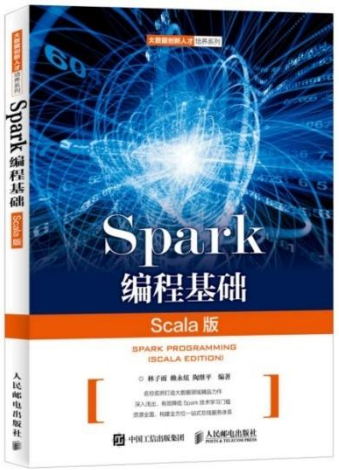
厦门大学林子雨编著

配套实验指导书

- 步步引导，循序渐进，详尽的安装指南为顺利搭建大数据实验环境铺平道路
- 深入浅出，去粗取精，丰富的代码实例帮助快速掌握大数据基础编程方法
- 精心设计，巧妙融合，八套大数据实验题目促进理论与编程知识的消化和吸收
- 结合理论，联系实际，大数据课程综合实验案例精彩呈现大数据分析全流程

林子雨编著《大数据基础编程、实验和案例教程（第2版）》
清华大学出版社 ISBN:978-7-302-55977-1 定价：69元 2020年10月第2版

附录H：《Spark编程基础（Scala版）》



《Spark编程基础（Scala版）》

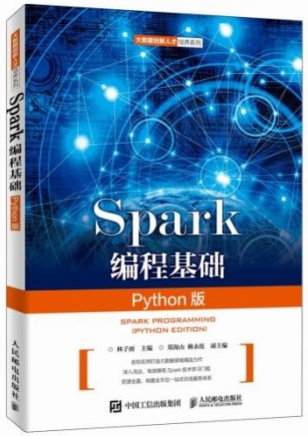
厦门大学 林子雨，赖永炫，陶继平 编著

披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-48816-9
教材官网：http://dblab.xmu.edu.cn/post/spark/

本书以Scala作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Scala语言基础、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、授课视频、上机实验指南等。

附录I：《Spark编程基础（Python版）》



《Spark编程基础（Python版）》

厦门大学 林子雨，郑海山，赖永炫 编著

披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-52439-3

教材官网： http://dblab.xmu.edu.cn/post/spark-python/

本书以Python作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Structured Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、上机实验指南等。

附录J：高校大数据课程公共服务平台

高校大数据课程

公 共 服 务 平 台

http://dblab.xmu.edu.cn/post/bigdata-teaching-platform/

扫一扫访问平台主页

扫一扫观看3分钟FLASH动画宣传片

附录K：高校大数据实训课程系列案例教材

为了更好满足高校开设大数据实训课程的教材需求，厦门大学数据库实验室林子雨老师团队联合企业共同开发了《高校大数据实训课程系列案例》，目前已经完成开发的系列案例包括：

《电影推荐系统》（已经于2019年5月出版）

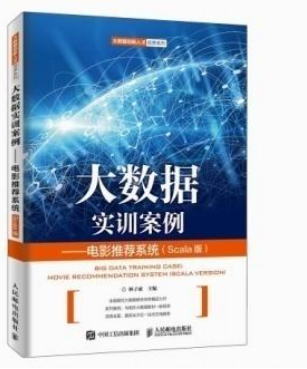
《电信用户行为分析》（已经于2019年5月出版）

《实时日志流处理分析》

《微博用户情感分析》

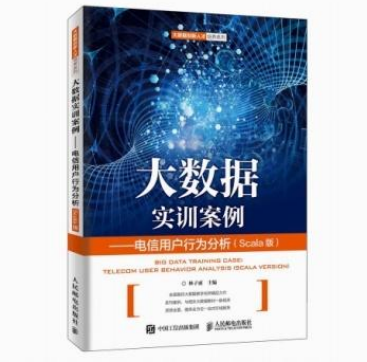
《互联网广告预测分析》

《网站日志处理分析》



系列案例教材将于2019年陆续出版发行，教材相关信息，敬请关注网页后续更新！

http://dblab.xmu.edu.cn/post/shixunkecheng/



扫一扫访问大数据实训课程系列案例教材主页


Thank You!