Chap. 3 分布式文件系统HDFS

王怡洋 副教授

大连海事大学 信息科学技术学院

内容提纲

*Chap. **3.1*** 分布式文件系统

*Chap. **3.2*** HDFS 简介

*Chap. **3.3*** 相关概念

*Chap. **3.4*** 体系结构

*Chap. **3.5*** 存储原理

*Chap. **3.6*** 数据读写过程

*Chap. **3.7*** 编程实践

本PPT是基于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》（2017年2月第2版） 林子雨 编著，人民邮电出版社

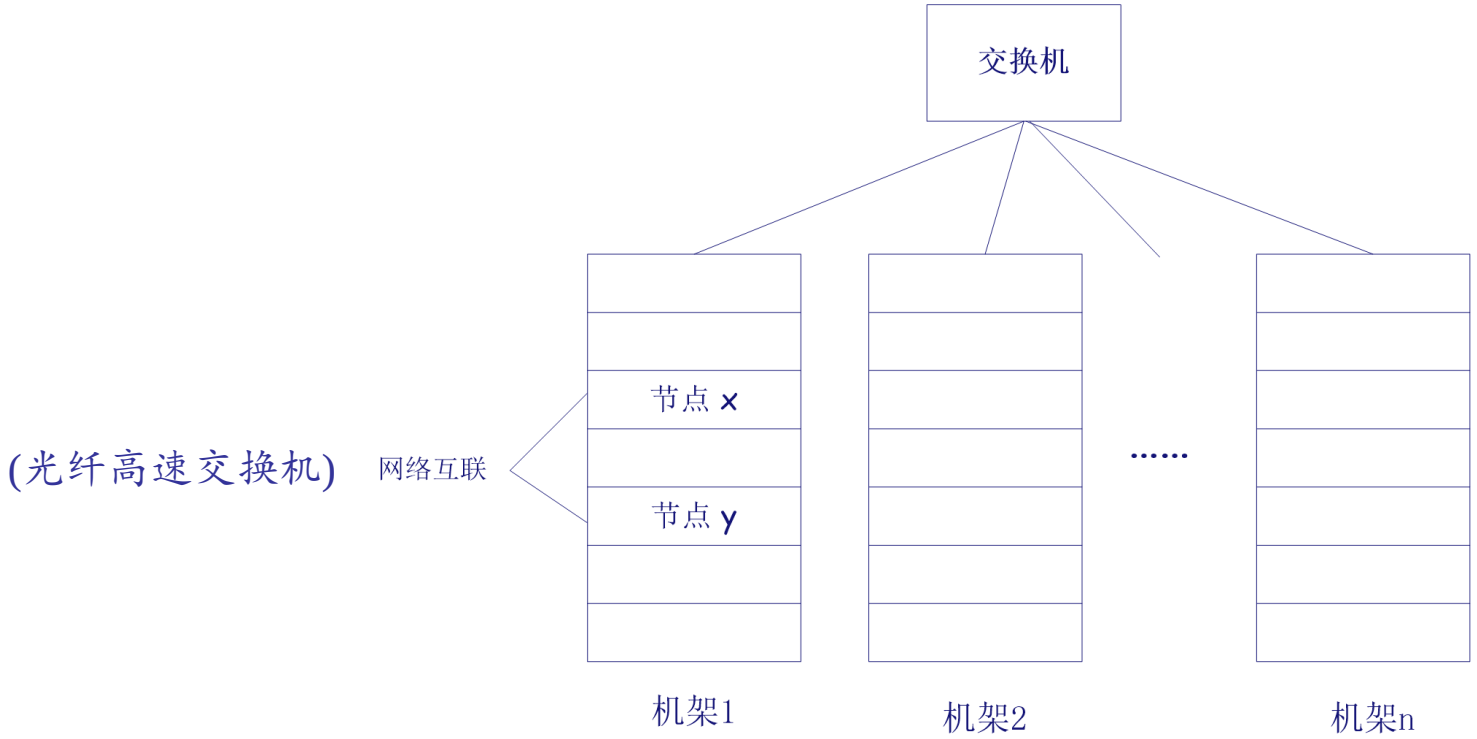
3.1 分布式文件系统

- 3.1.1 计算机集群结构
- 3.1.2 分布式文件系统的结构
- 3.1.3 分布式文件系统的设计需求

3.1.1 计算机集群结构

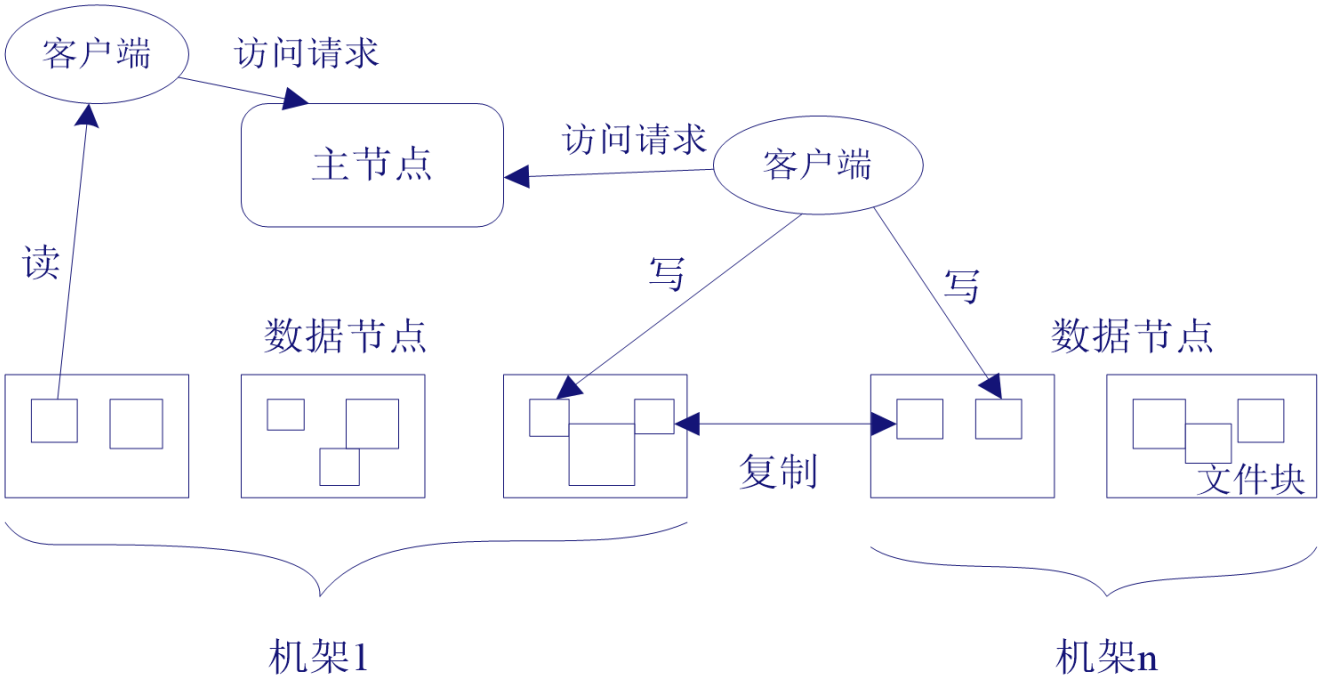
- 分布式文件系统把文件分布存储到多个计算机节点上，*成千上万的计算机节点构成计算机集群*

- 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



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：“主节点”(*Master Node*) 或者也被称为“名称结点”(*NameNode*)；“从节点”(*Slave Node*) 或者也被称为“数据节点”(*DataNode*)

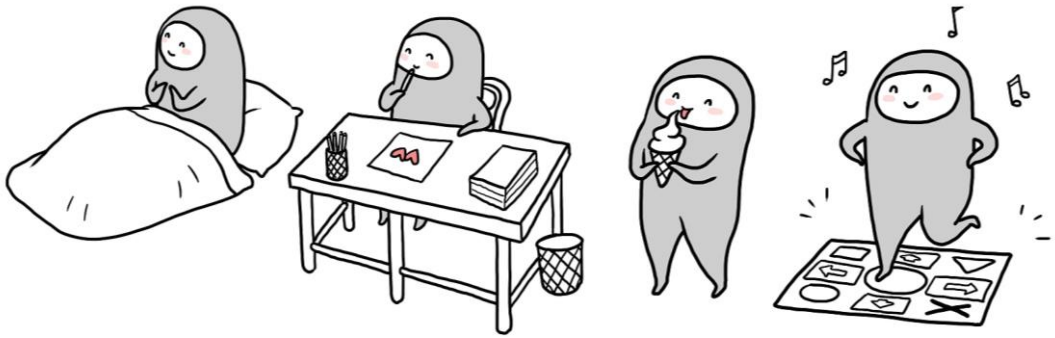


- 采用多副本存储：文件块被复制为多个副本，存储在不同的节点上，分布在不同的机架上。单个节点出现故障，可以快速调用副本重启节点上的计算，而不用重启整个计算过程。

3.2 HDFS简介

总体而言，HDFS要实现以下目标：

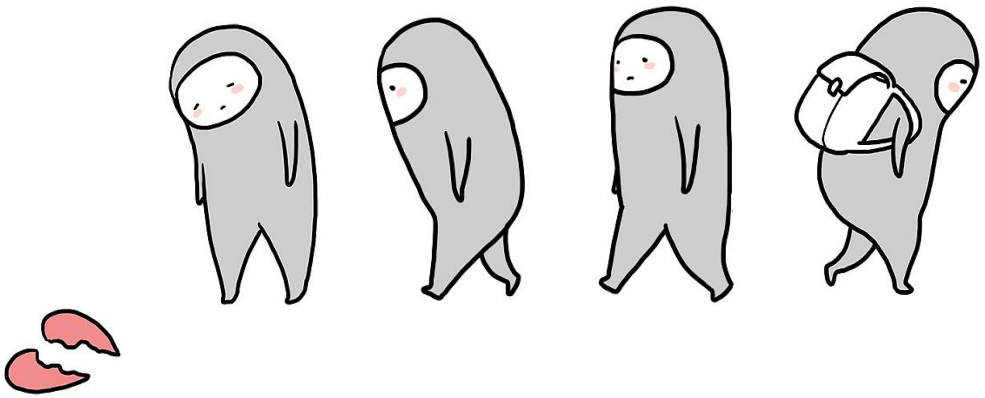
- 兼容廉价的硬件设备
- 流数据读写
- 大数据集
- 简单的文件模型
- 强大的跨平台兼容性



3.2 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



3.3 HDFS相关概念

- 3.3.1 块
- 3.3.2 名称节点和数据节点

3.3.1 块

整个HDFS当中最核心的概念

为了分摊磁盘读写开销，也就是在大量数据间分摊磁盘寻址的开销

联系

块

区别

HDFS的一个块要比普通文件系统的块大很多

- 支持面向大规模数据存储
- 降低分布式节点的存储开销

thousands of **bytes** VS 64/128 **MB**

3.3.1 块

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

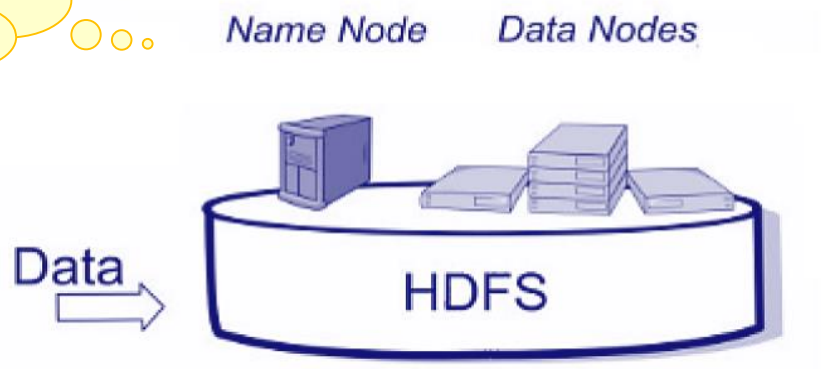
- **适合数据备份：**每个文件块都可以**冗余存储到多个节点上**，大大提高了系统的容错性和可用性

3.3.2 名称节点和数据节点

HDFS主要组件的功能

HDFS集群管家

- 文件是什么
- 文件被分成多少块
- 每个块和文件是怎么映射的
- 每个块被存储在哪个服务器上面



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中
• 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（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

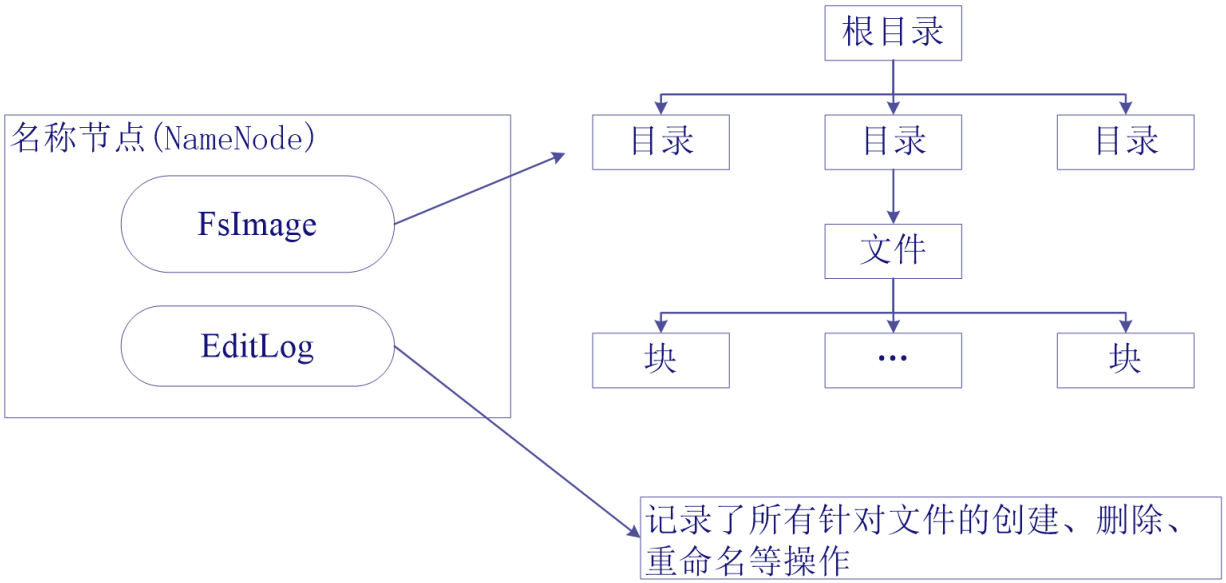


图3-3 名称节点的数据结构

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文件没有记录文件包含哪些块以及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

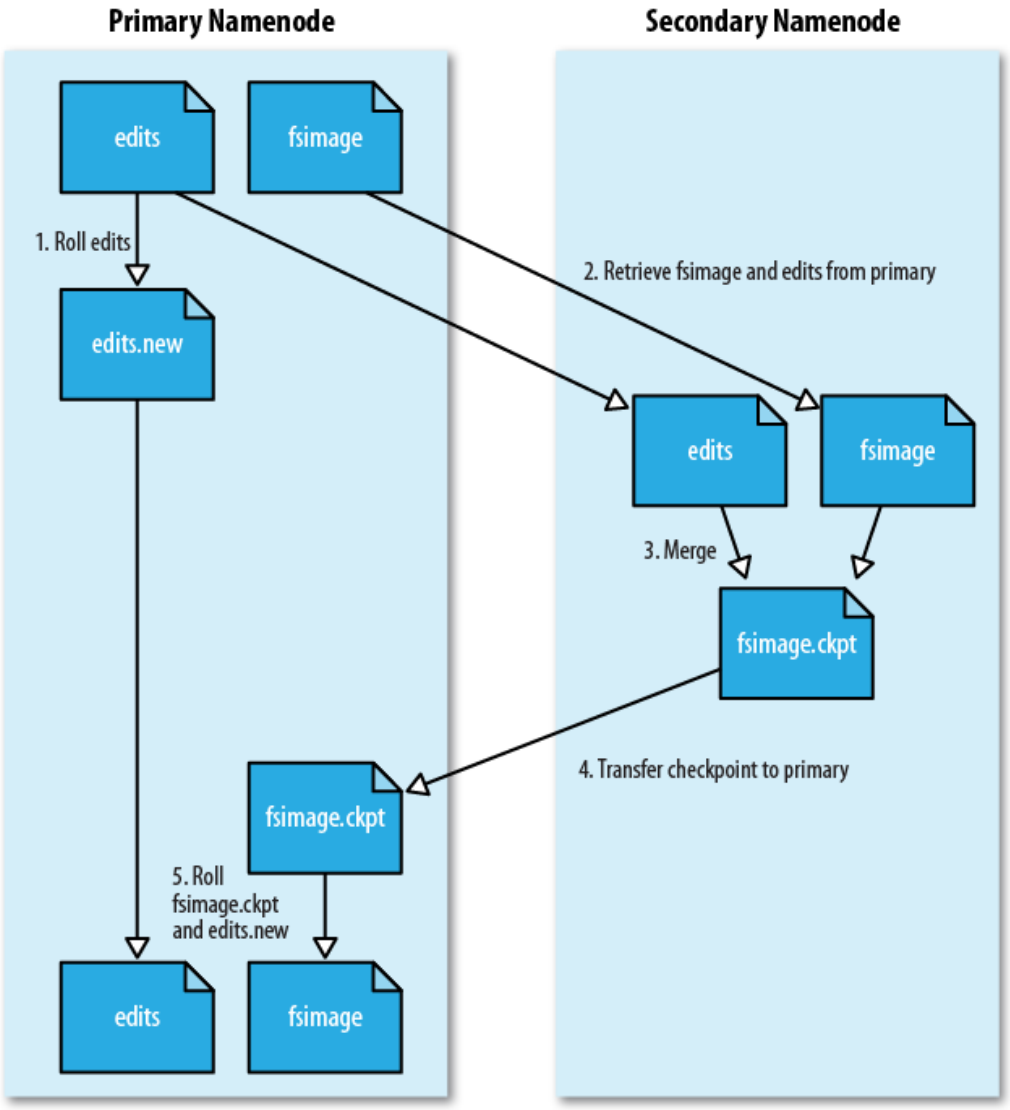
- 在名称节点运行期间，HDFS的所有更新操作都是直接写到EditLog中，久而久之，EditLog文件将会变得很大

- 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

3.4.1 HDFS体系结构概述

- HDFS采用了主从（Master/Slave）结构模型：一个集群包括一个名称节点和若干个数据节点。
- 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。
- 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。

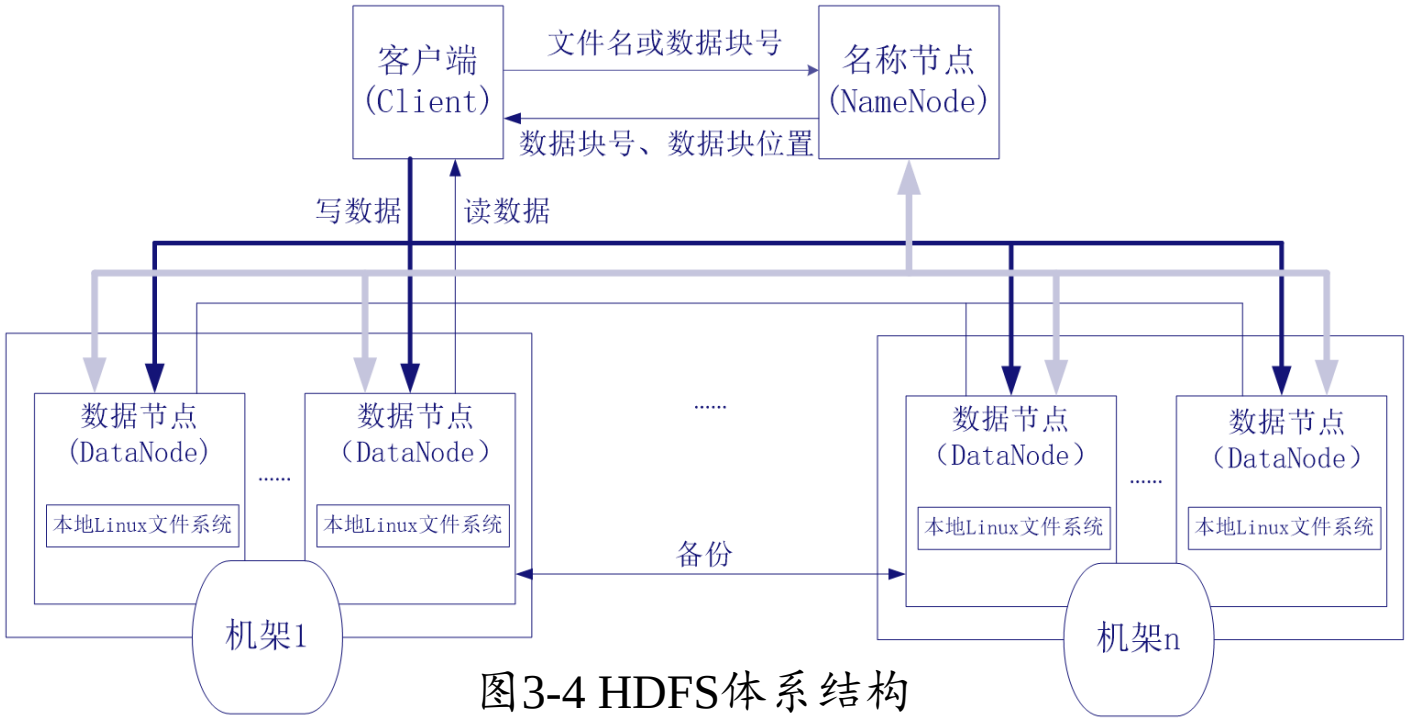


图3-4 HDFS体系结构

3.4.2 HDFS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

3.4.5 HDFS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：

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

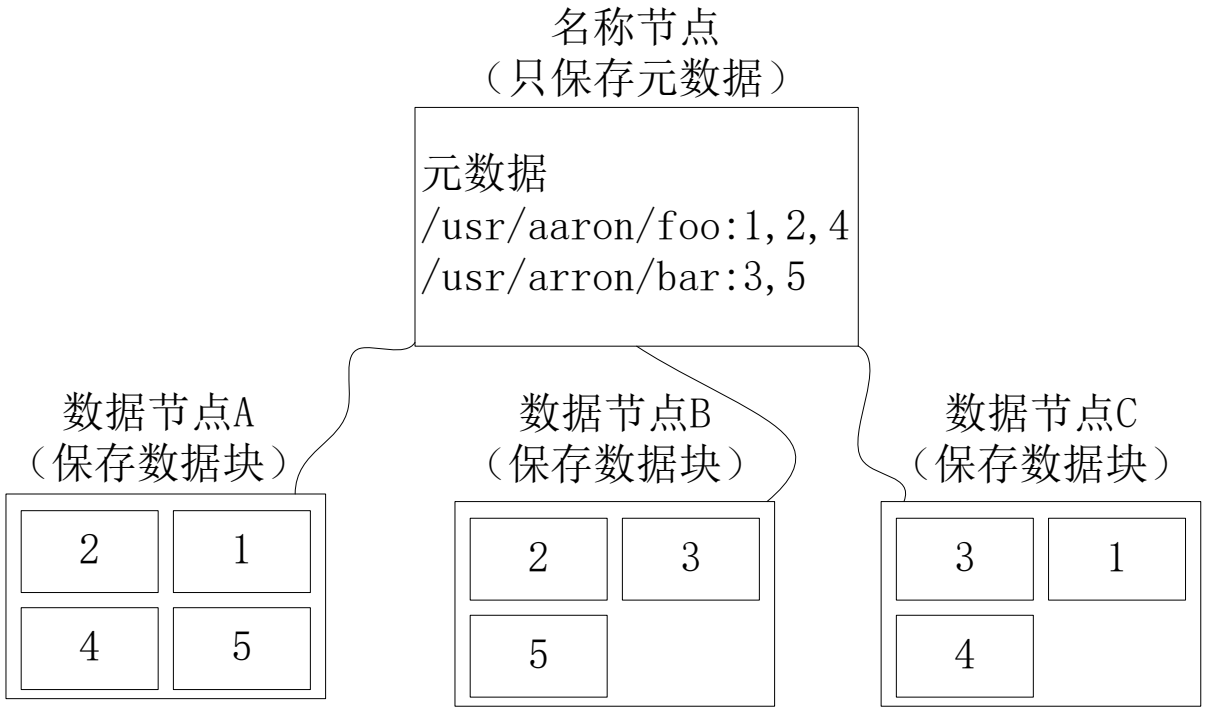
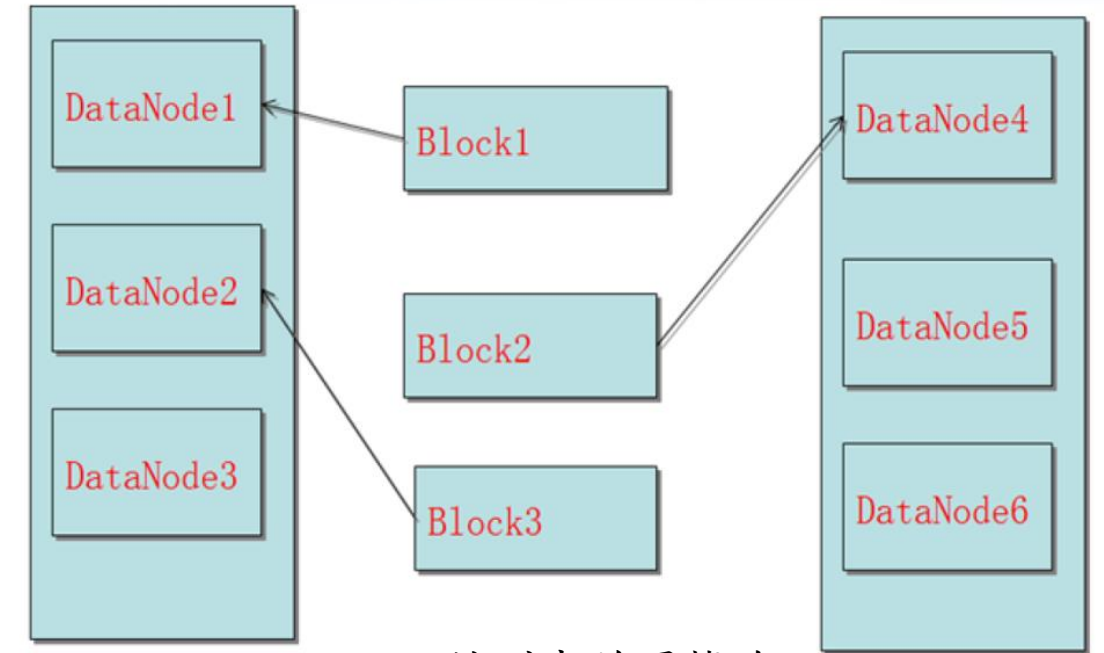


图3-5 HDFS数据块多副本存储

3.5.2 数据存取策略

- **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

- **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("hdfs://localhost:9000/user/Hadoop/test.txt");
                        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

- 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 HDFS数据读写过程

读取文件

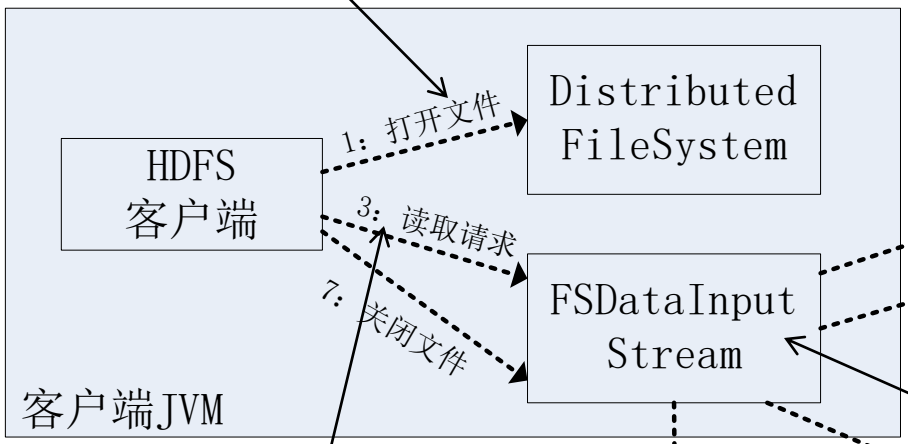
```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.FSDataInputStream;

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("hdfs://localhost:9000/user/Hadoop/test.txt");
                        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();
                }
        }
}
}
```

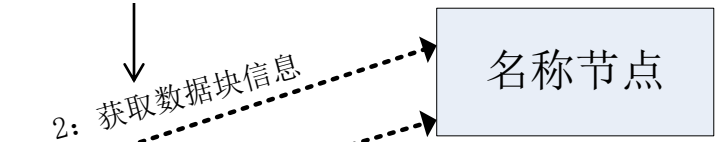
3.6.1 读数据的过程

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



客户端节点

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



2：获取数据块信息

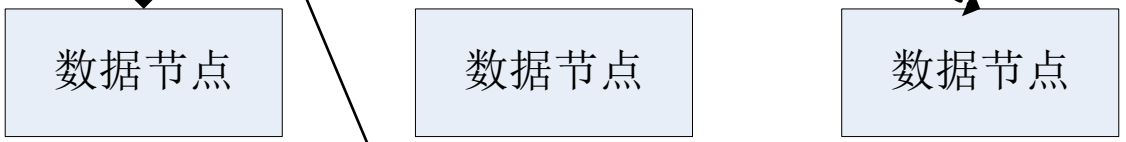
5：获取数据块信息
（可能发生）通过ClientProtocal.getBlockLocations()
查找下一个数据块

FSDataInputStream封装了DFSInputStream

4:读取数据

6:读取数据

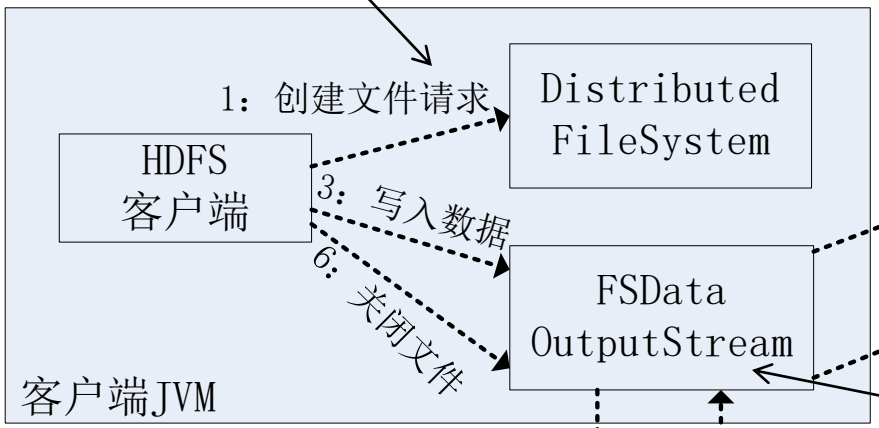
客户端获得输入流FSDataInputStream以后
调用read()函数开始读取数据
输入流根据前面的排序结果
选择距离客户端最近的数据节点
建立连接并读取数据



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）

2：创建文件元数据

名称节点

DFSOutputStream调用
ClientProtocal.complete()方法
通知名称节点关闭文件

7：写操作完成

FSDataOutputStream封装了DFSOutputStream

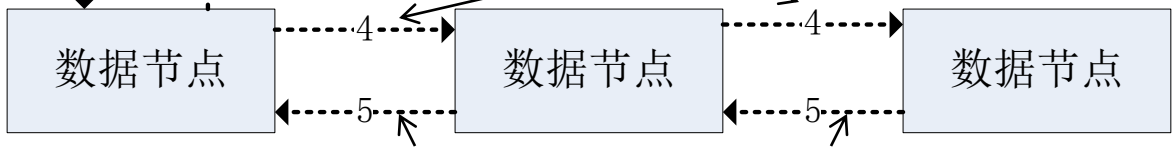
客户端节点

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

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”



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

《大数据技术原理与应用 第三章 Hadoop分布式文件系统 学习指南》

http://dblab.xmu.edu.cn/blog/290-2/

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
```

3.7.1 HDFS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

3.7.1 HDFS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
```

3.7.1 HDFS常用命令

实例：

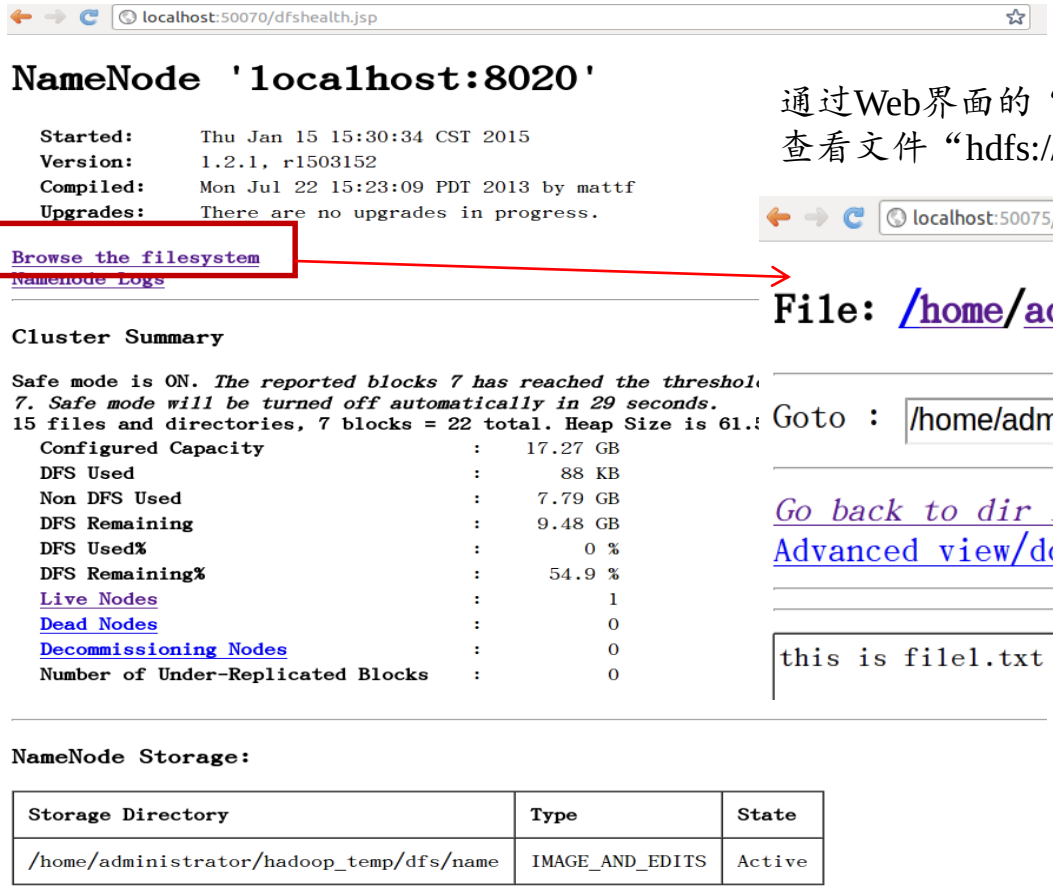
hadoop fs -cat <path>:将<path>指定的文件的内容输出到标准输出（stdout）

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
```

3.7.2 HDFS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”

3.7.3 HDFS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上是否存在某个文件？

准备工作：在Ubuntu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

具体请参见：

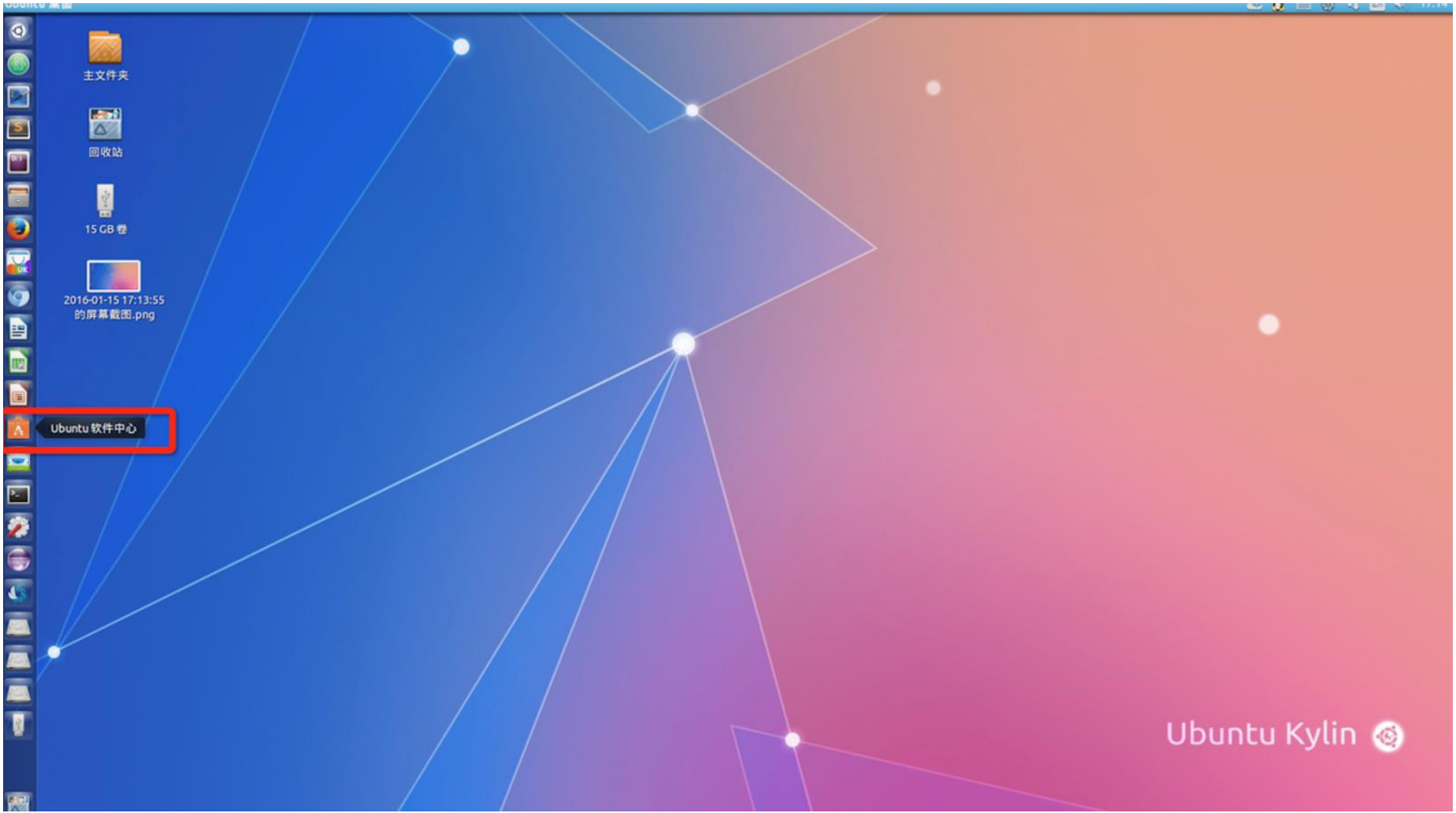
《大数据技术原理与应用 第三章 Hadoop分布式文件系统 学习指南》

http://dblab.xmu.edu.cn/blog/290-2/

3.7.3 HDFS常用Java API及应用实例

（1）在Ubuntu中安装Eclipse

利用Ubuntu左侧边栏自带的软件中心安装软件，在Ubuntu左侧边栏打开软件中心

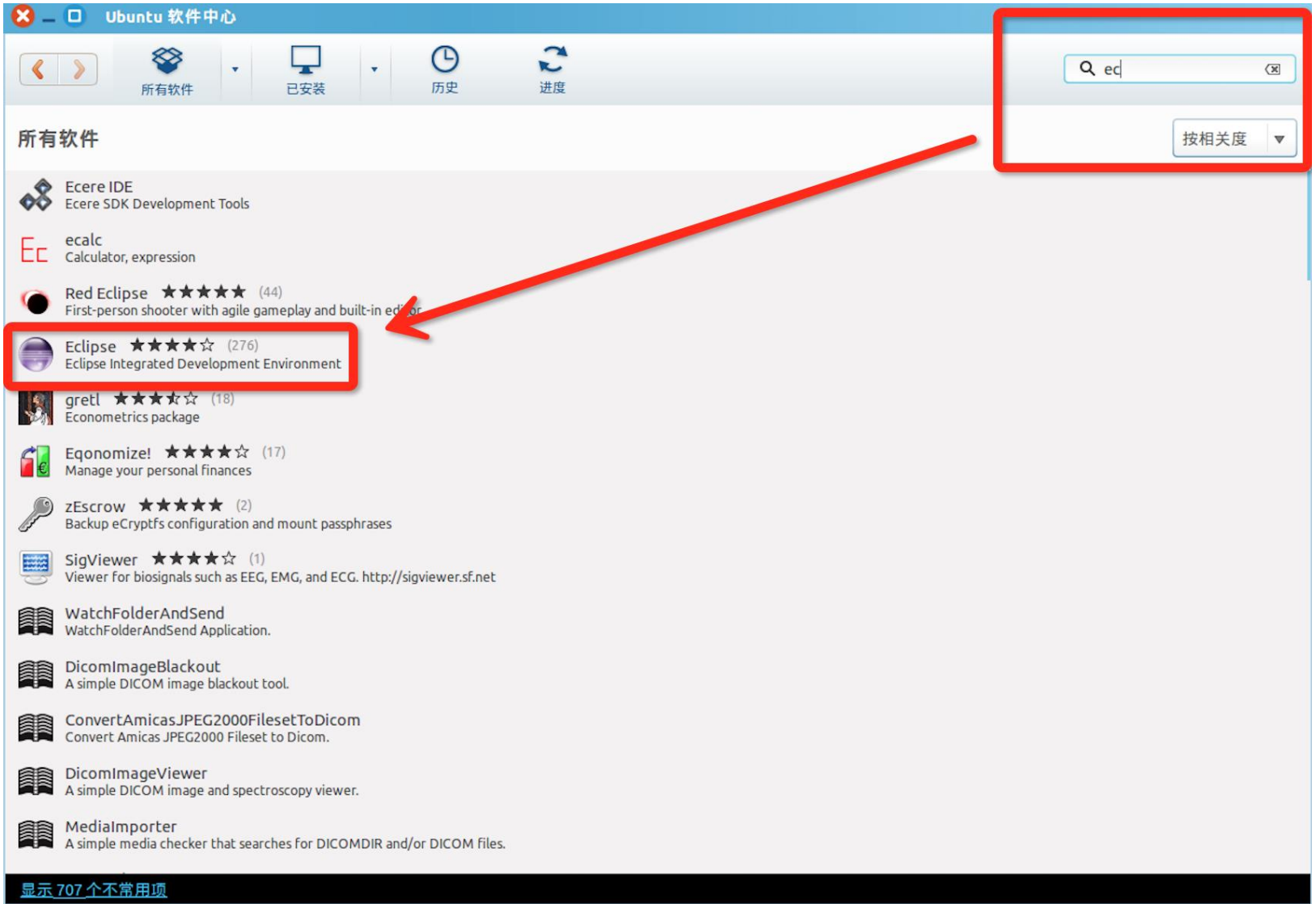


3.7.3 HDFS常用Java API及应用实例

打开软件中心后，呈现如下界面

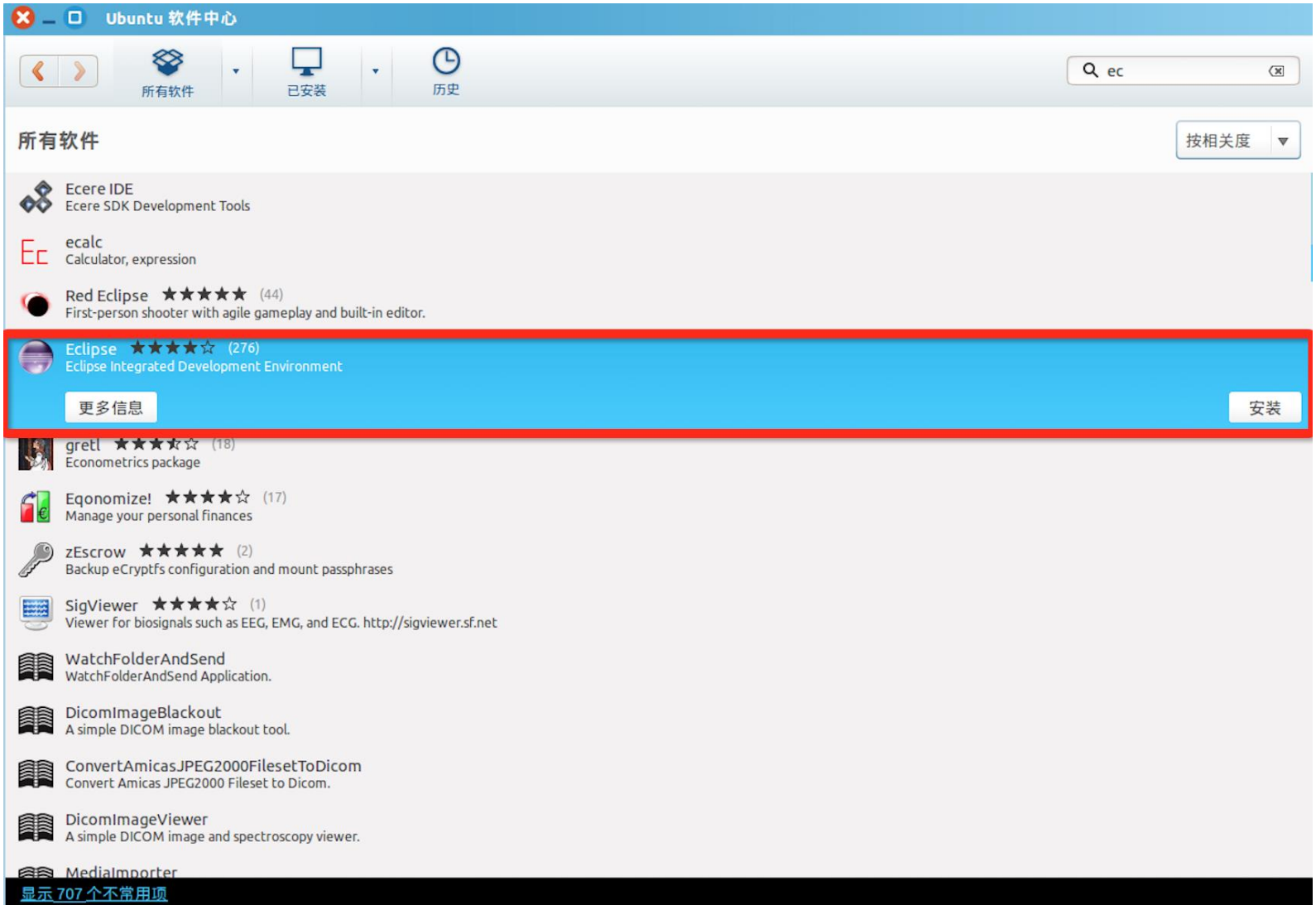
3.7.3 HDFS常用Java API及应用实例

在软件中心搜索栏输入“ec”，软件中心会自动搜索相关的软件



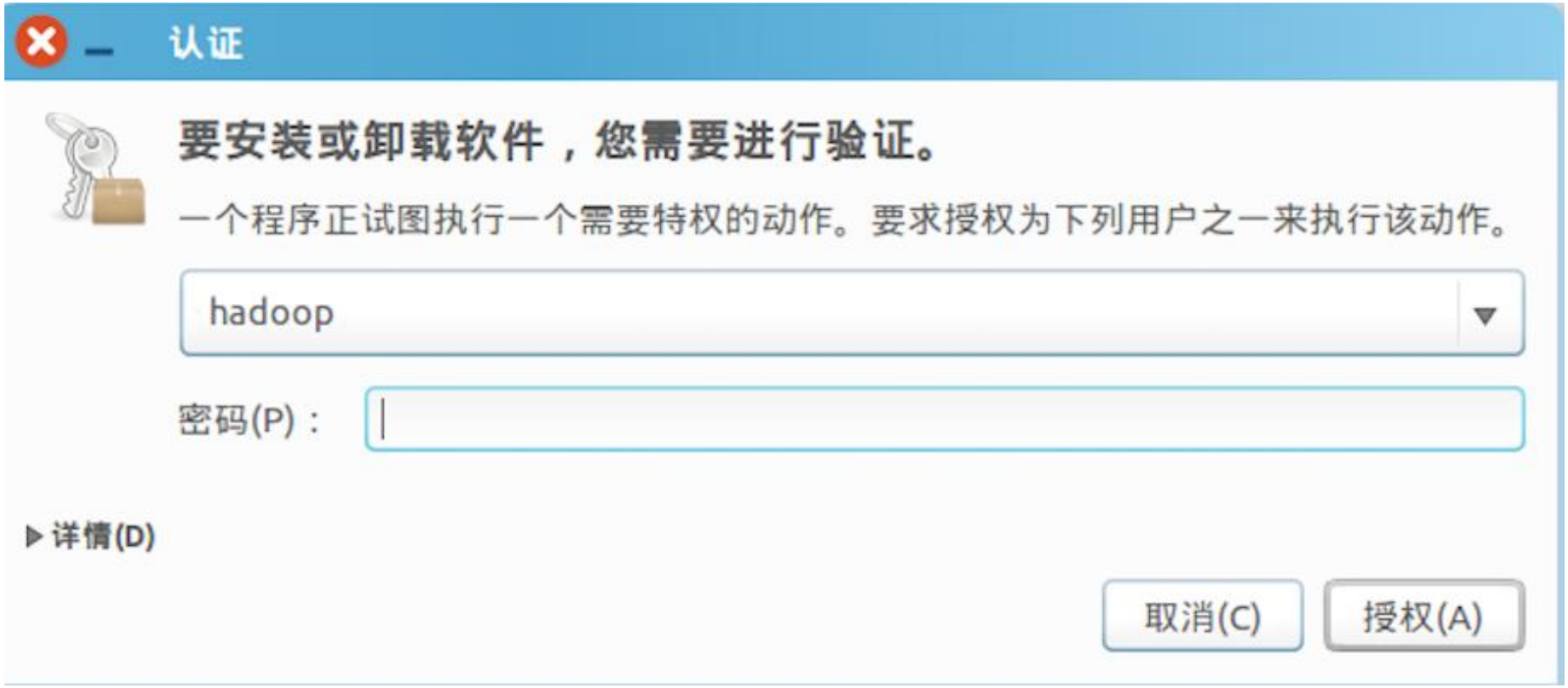
3.7.3 HDFS常用Java API及应用实例

点击如下图中Eclipse，进行安装



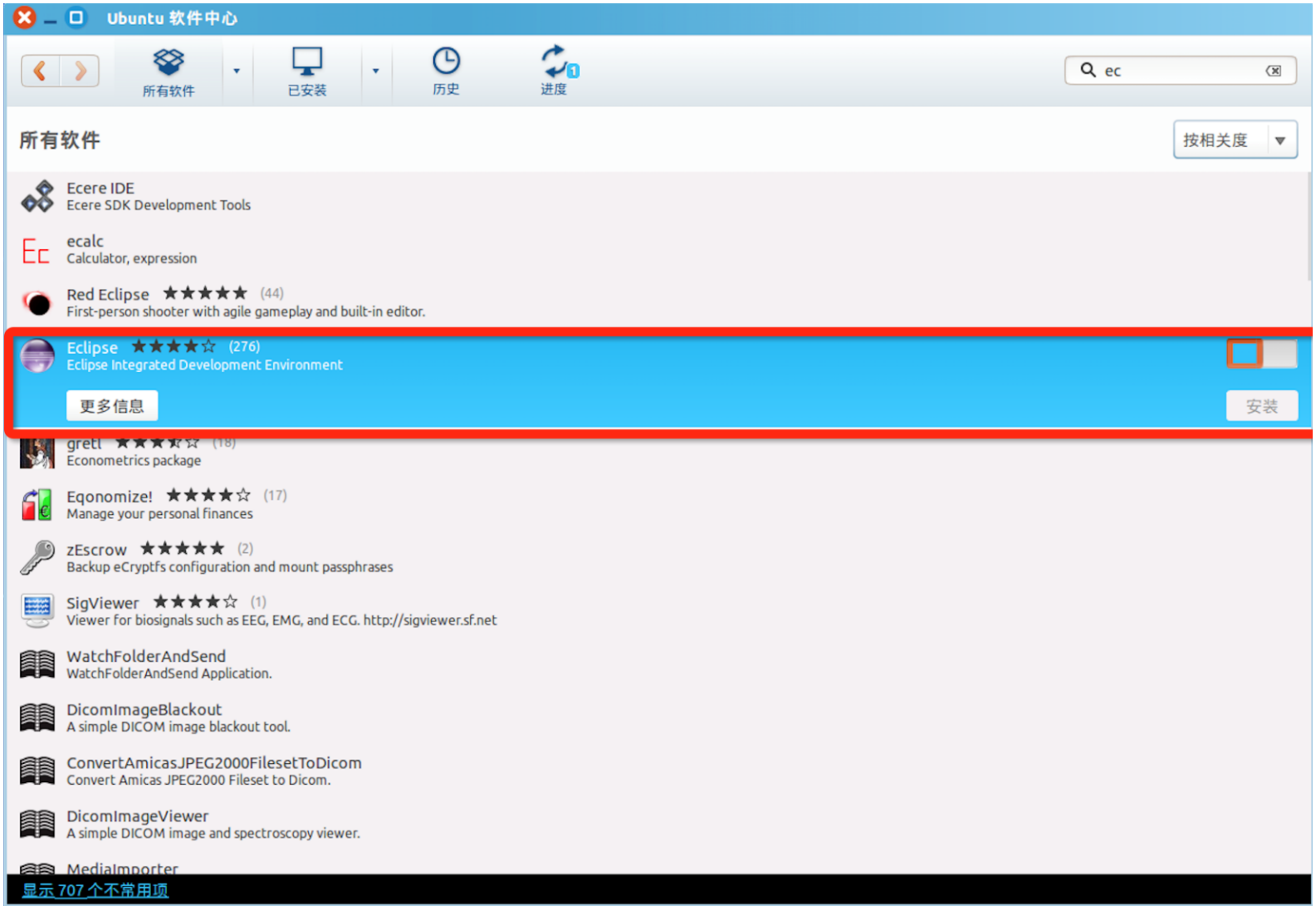
3.7.3 HDFS常用Java API及应用实例

安装需要管理员权限，Ubuntu系统需要用户认证，弹出“认证”窗口，请输入当前用户的登录密码



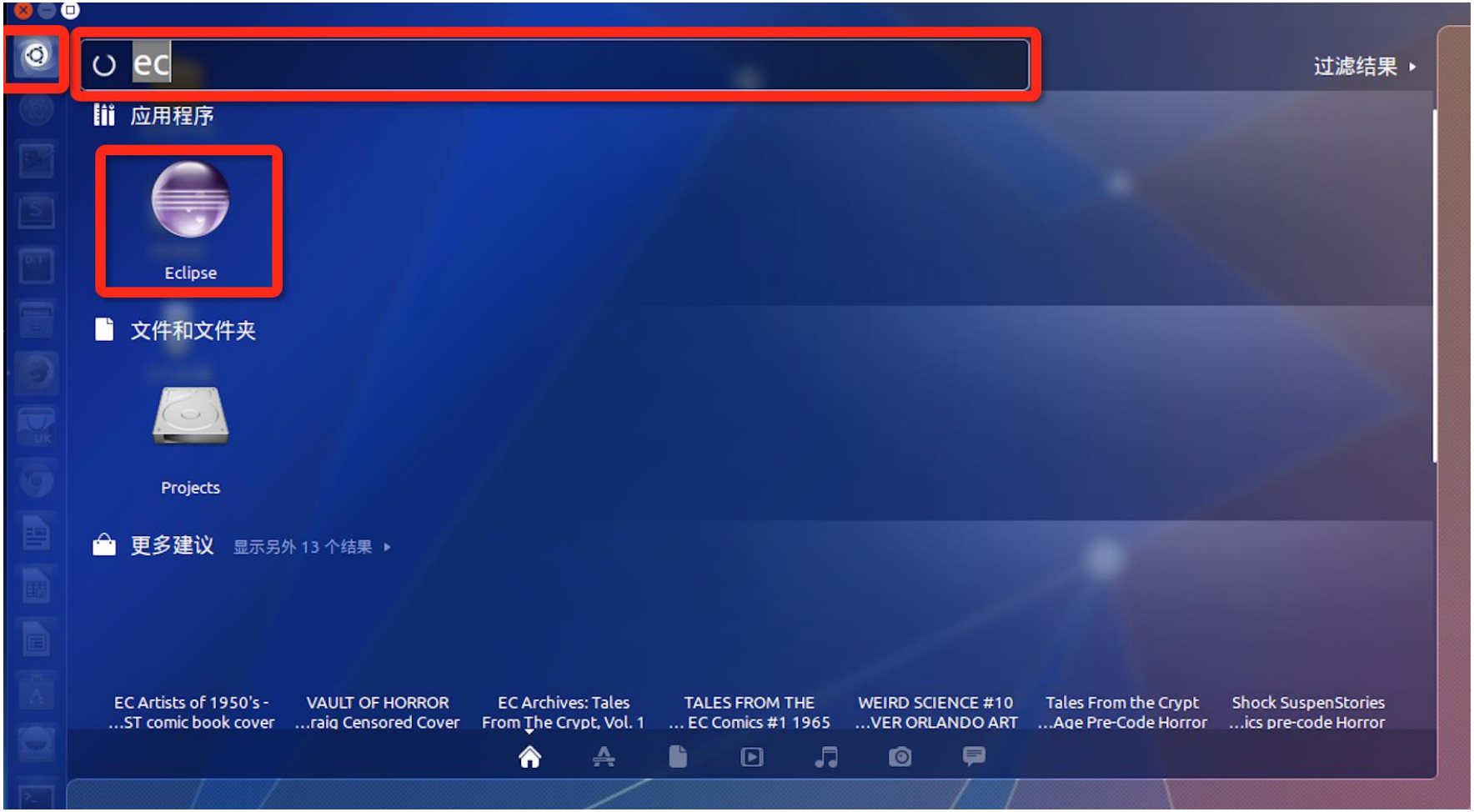
3.7.3 HDFS常用Java API及应用实例

ubuntu便会进入如下图的安装过程中,安装结束后安装进度条便会消失。



3.7.3 HDFS常用Java API及应用实例

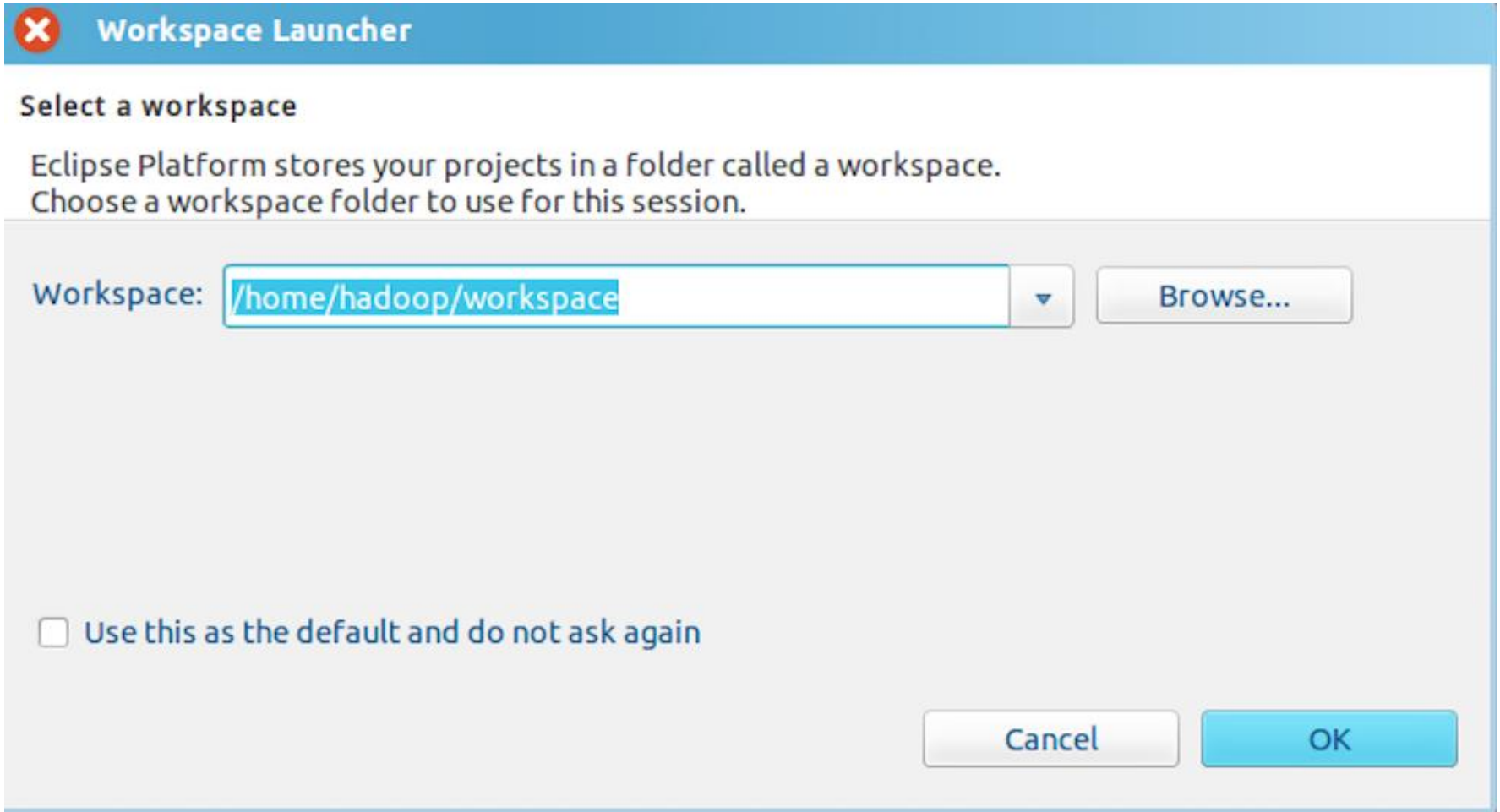
点击Ubuntu左侧边栏的搜索工具，输入“ec”，自动搜索已经安装好的相关软件,打开Eclipse



3.7.3 HDFS常用Java API及应用实例

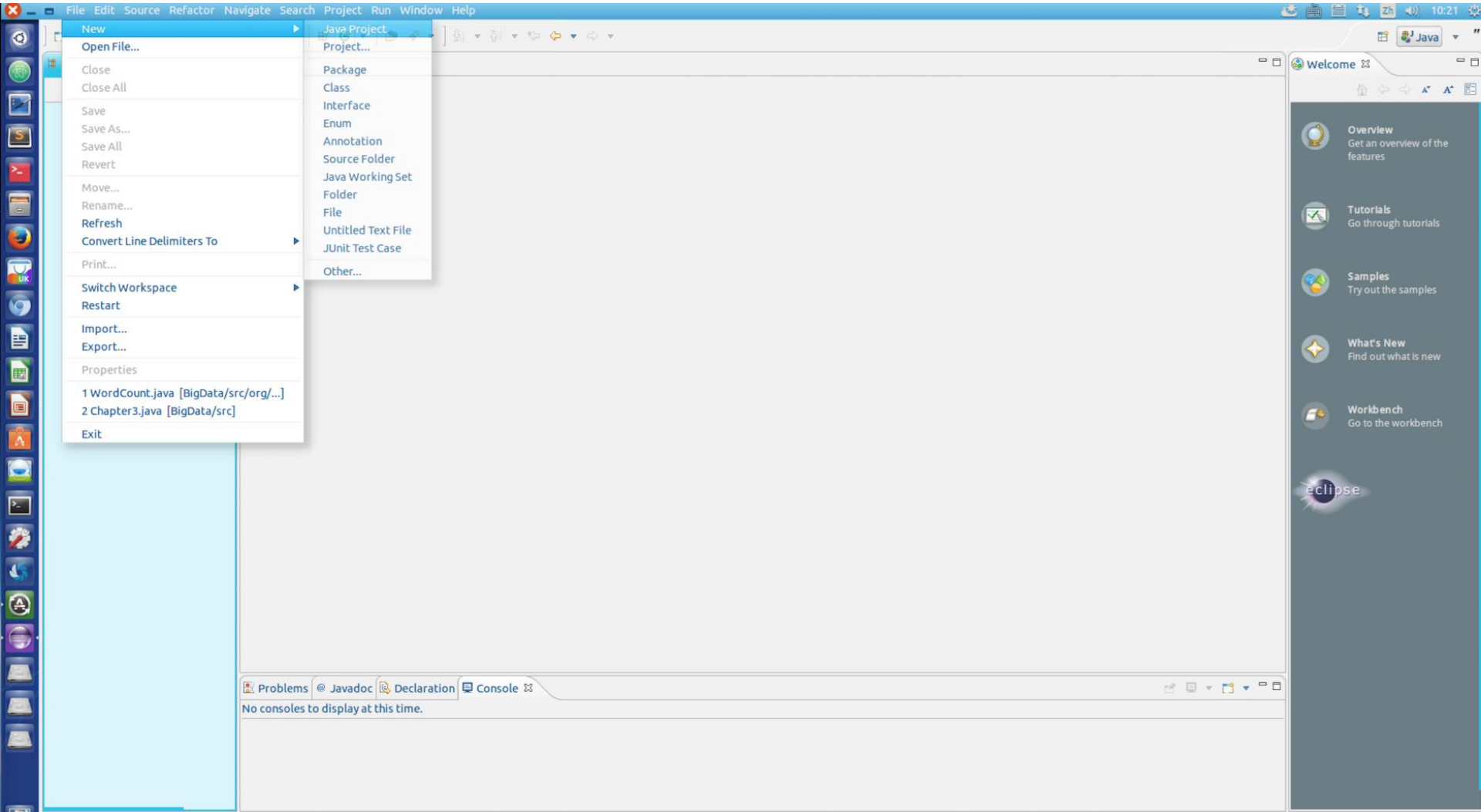
（2）在Eclipse创建项目

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



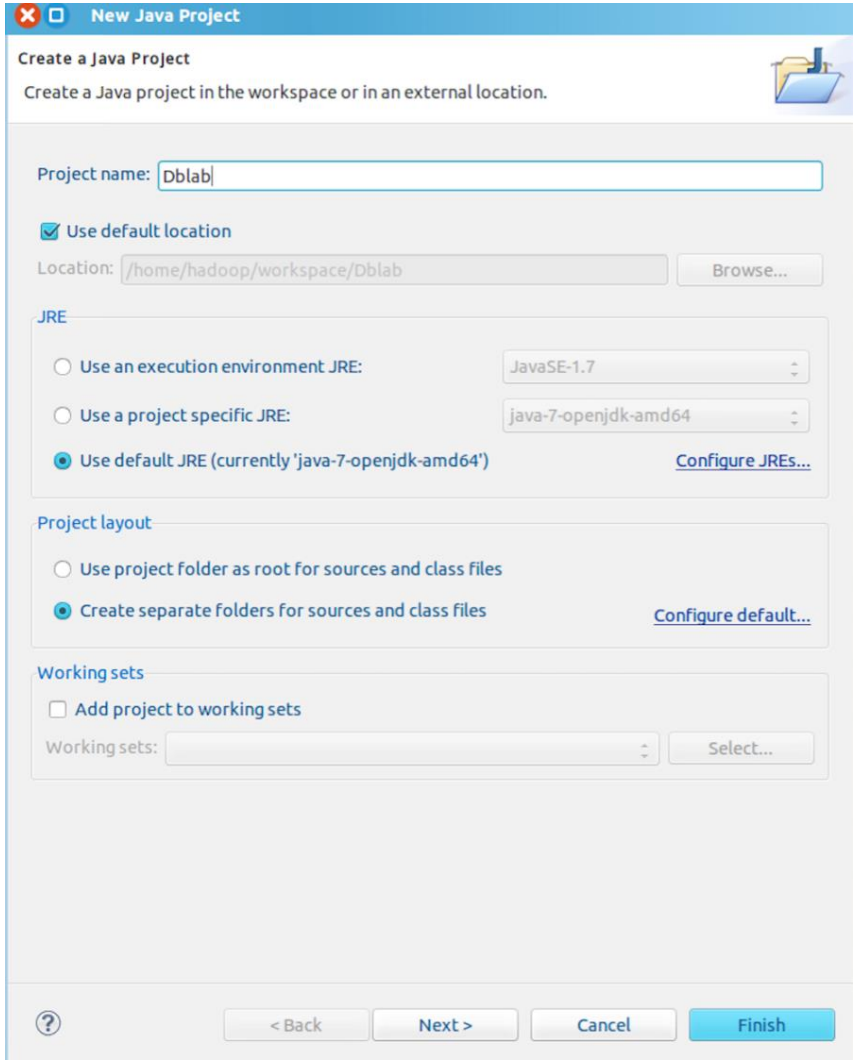
3.7.3 HDFS常用Java API及应用实例

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



3.7.3 HDFS常用Java API及应用实例

输入项目名称，本教程输入的项目名称是“Dblab”，其他不用改动，点击“Finish”按钮即可。



3.7.3 HDFS常用Java API及应用实例

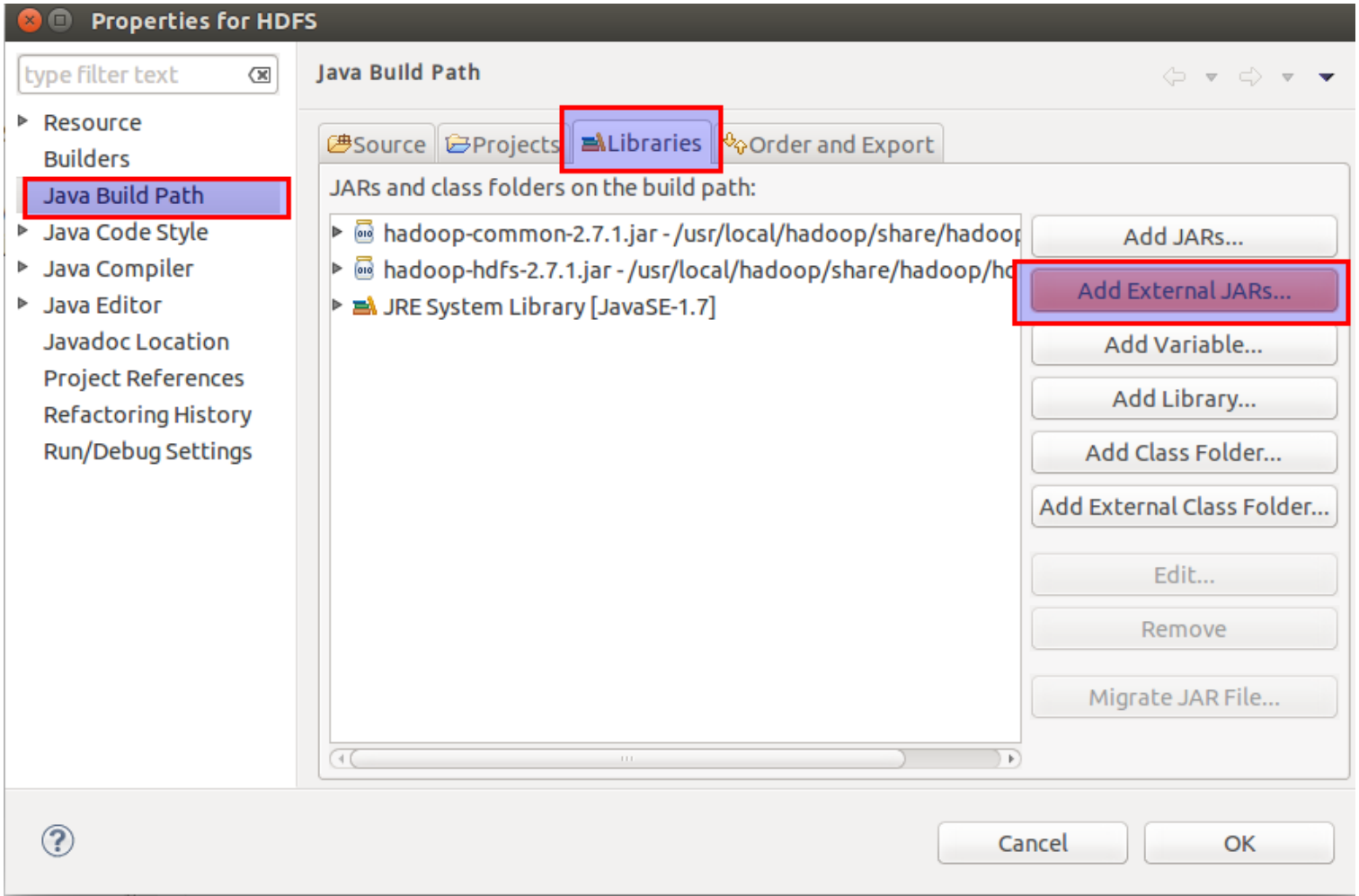
为项目加载所需要用到的jar包

如何获取jar包

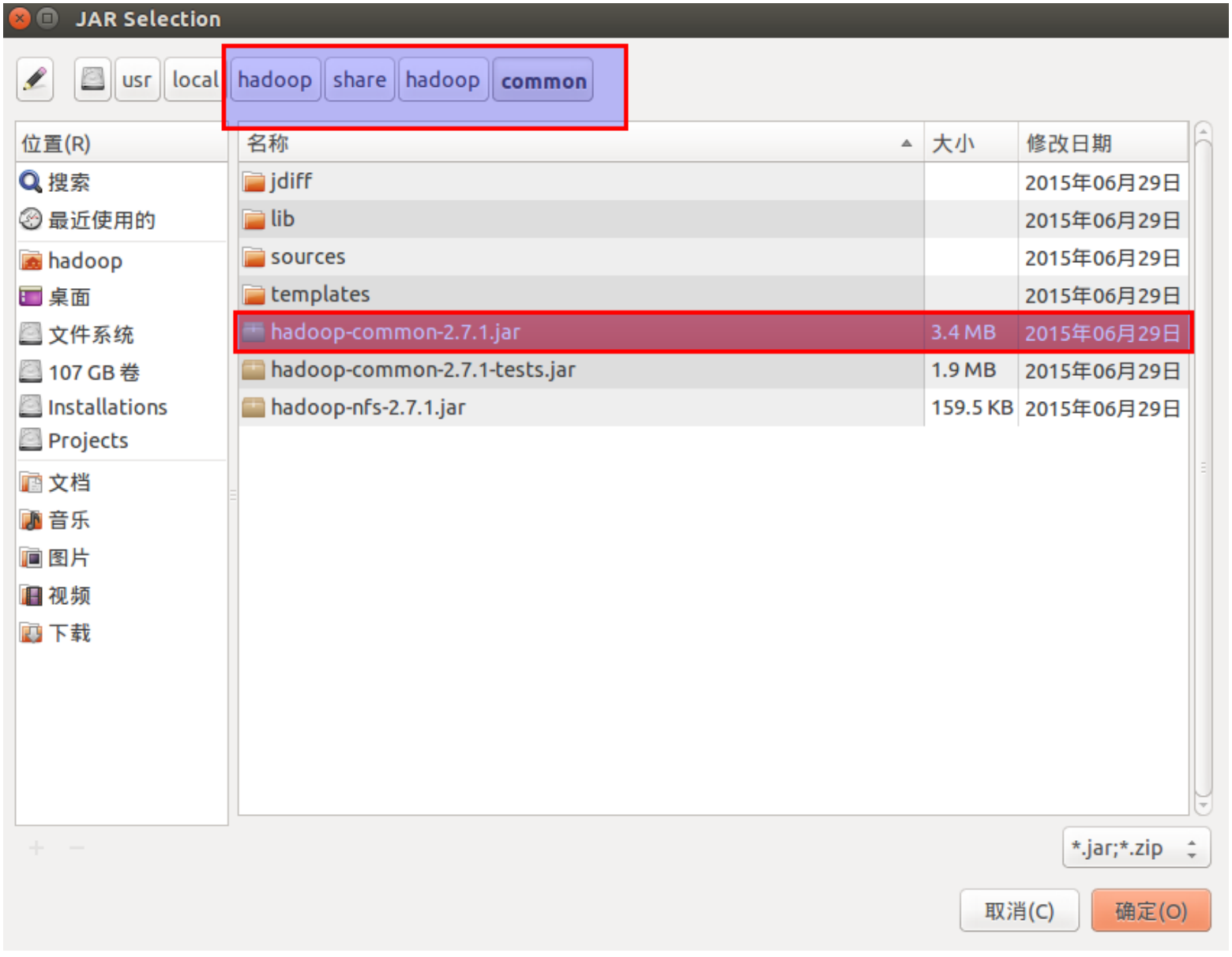
Java API所在的jar包都在已经安装好的hadoop文件夹里，路径：/usr/local/hadoop/share/hadoop(如果读者安装的hadoop不在此目录，请找到jar包所在的文件夹)

3.7.3 HDFS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



3.7.3 HDFS常用Java API及应用实例



3.7.3 HDFS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:///
```

3.7.3 HDFS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();
        }
    }
}
```