



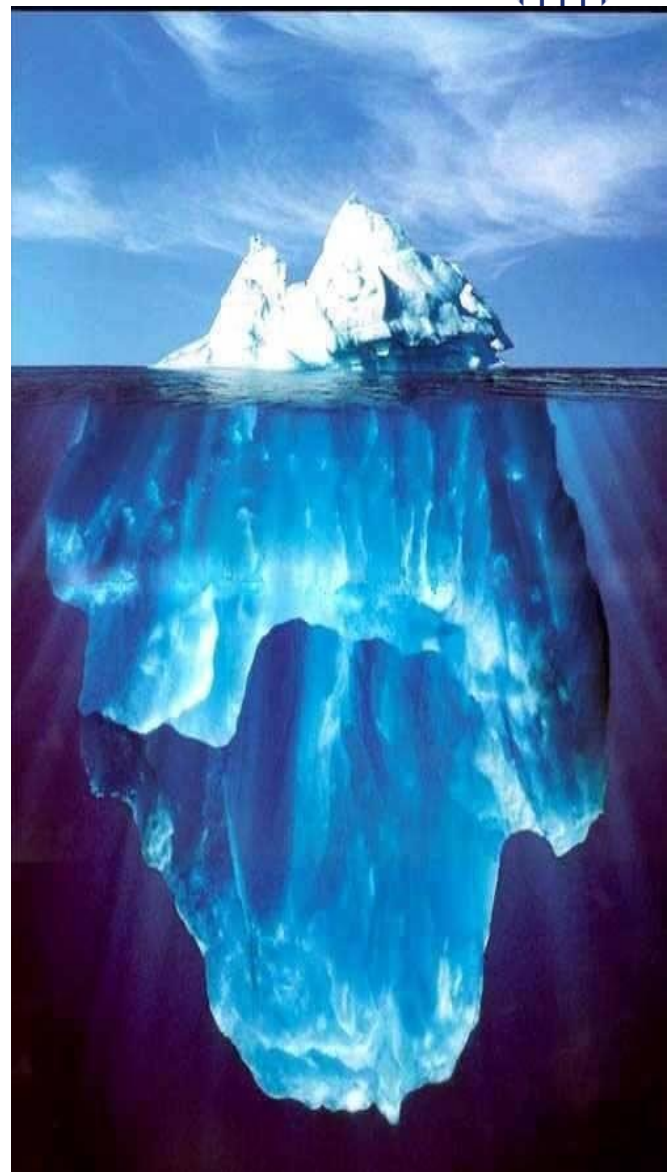
第五章 分布式文件系统

2022年



第一部分 大数据处理架构Hadoop

- 2.1 概述
- 2.2 Hadoop项目结构
- 2.3 Hadoop的安装与使用
- 2.4 Hadoop集群的部署与使用



2.1 概述

- 2.1.1 Hadoop简介
- 2.1.2 Hadoop发展简史
- 2.1.3 Hadoop的特性
- 2.1.4 Hadoop的应用现状

- Hadoop是Apache软件基金会旗下的一个开源分布式计算平台，为用户提供了系统底层细节透明的分布式基础架构
- Hadoop是基于Java语言开发的，具有很好的跨平台特性，并且可以部署在廉价的计算机集群中
- Hadoop的核心是分布式文件系统HDFS（Hadoop Distributed File System）和MapReduce
- Hadoop被公认为行业大数据标准开源软件，在分布式环境下提供了海量数据的处理能力
- 几乎所有主流厂商都围绕Hadoop提供开发工具、开源软件、商业化工具和技术服务，如谷歌、雅虎、微软、思科、淘宝等，都支持Hadoop



Hadoop的标志

- Hadoop最初是由Apache Lucene项目的创始人Doug Cutting开发的文本搜索库。Hadoop源自始于2002年的Apache Nutch项目——一个开源的网络搜索引擎并且也是Lucene项目的一部分
- 在2004年，Nutch项目也模仿GFS开发了自己的分布式文件系统NDFS（Nutch Distributed File System），也就是HDFS的前身
- 2004年，谷歌公司又发表了另一篇具有深远影响的论文，阐述了MapReduce分布式编程思想
- 2005年，Nutch开源实现了谷歌的MapReduce

2.1.2 Hadoop发展简史



- 到了2006年2月，Nutch中的NDFS和MapReduce开始独立出来，成为Lucene项目的一个子项目，称为Hadoop，同时，Doug Cutting加盟雅虎
- 2008年1月，Hadoop正式成为Apache顶级项目，Hadoop也逐渐开始被雅虎之外的其他公司使用
- 2008年4月，Hadoop打破世界纪录，成为最快排序1TB数据的系统，它采用一个由910个节点构成的集群进行运算，排序时间只用了209秒
- 在2009年5月，Hadoop更是把1TB数据排序时间缩短到62秒。Hadoop从此名声大震，迅速发展成为大数据时代最具影响力的开源分布式开发平台，并成为事实上的大数据处理标准

2.1.3 Hadoop的特性



Hadoop是一个能够对大量数据进行分布式处理的软件框架，并且是以一种可靠、高效、可伸缩的方式进行处理，它具有以下几个方面的特性：

- 高可靠性
- 高效性
- 高可扩展性
- 高容错性
- 成本低
- 运行在Linux平台上
- 支持多种编程语言

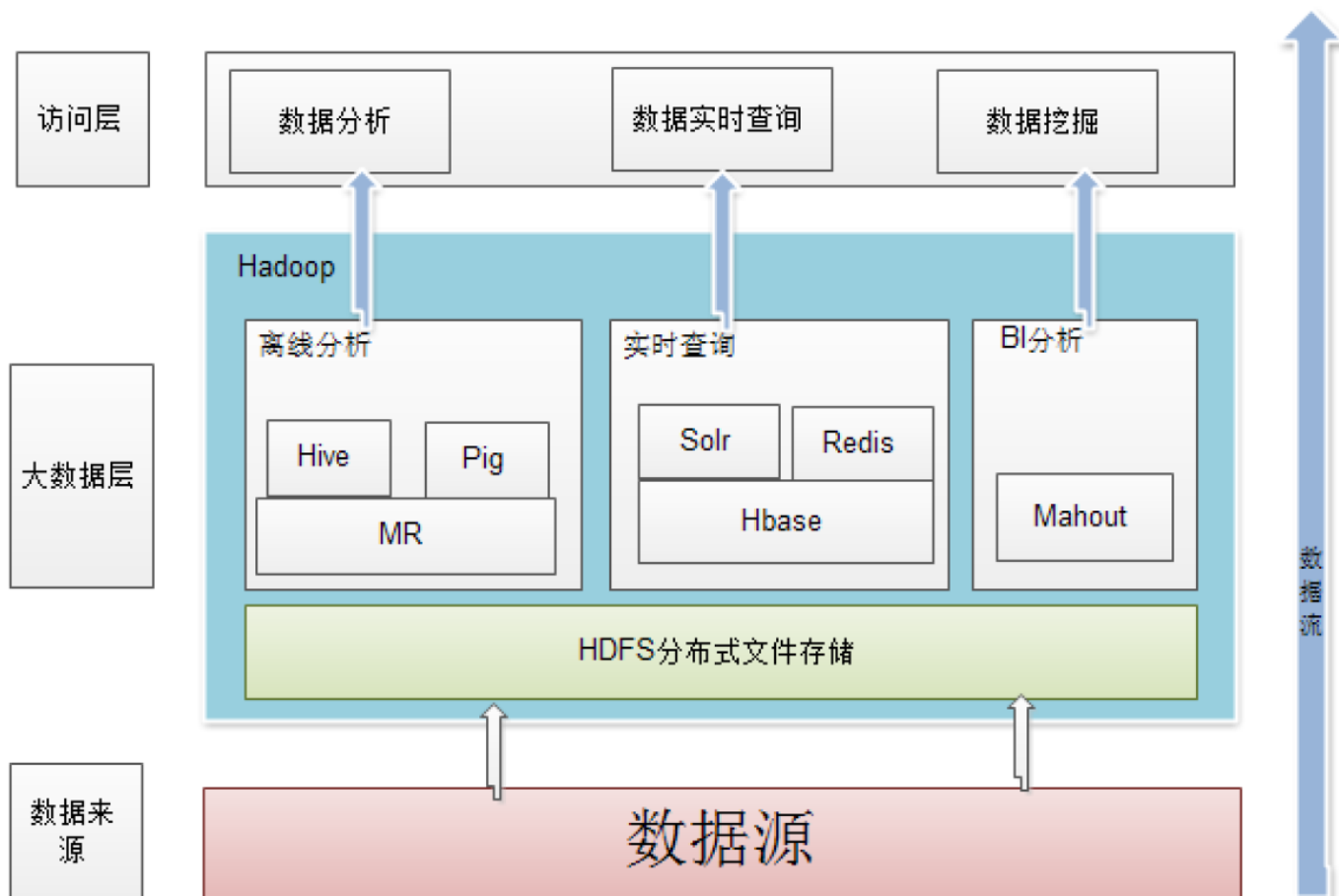
2.1.3 Hadoop的应用现状



- Hadoop凭借其突出的优势，已经在各个领域得到了广泛的应用，而互联网领域是其应用的主阵地
- 2007年，雅虎在Sunnyvale总部建立了M45——一个包含了4000个处理器和1.5PB容量的Hadoop集群系统
- Facebook作为全球知名的社交网站，Hadoop是非常理想的选择，Facebook主要将Hadoop平台用于日志处理、推荐系统和数据仓库等方面
- 国内采用Hadoop的公司主要有百度、淘宝、网易、华为、中国移动等，其中，淘宝的Hadoop集群比较大

2.1.3 Hadoop的应用现状

Hadoop在企业中的应用架构

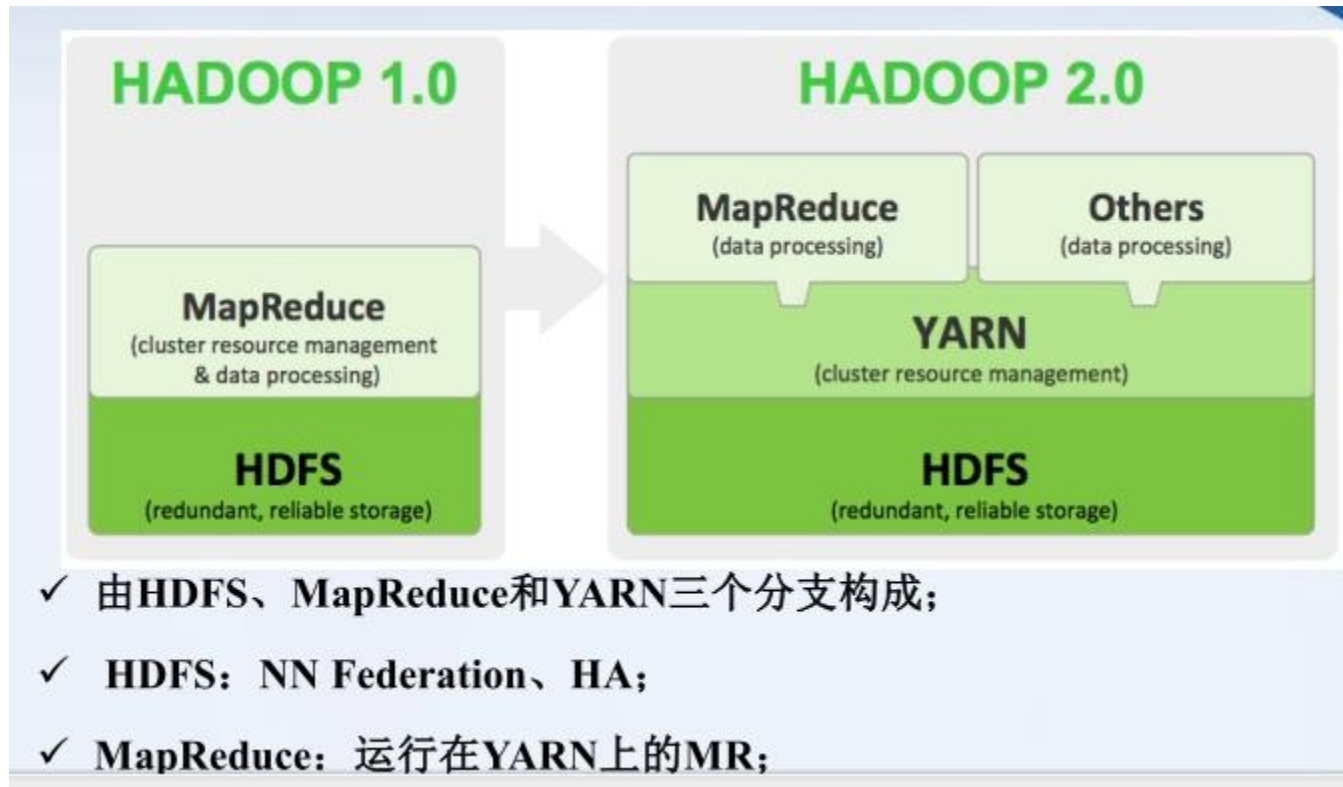


2.1.4 Apache Hadoop版本演变



- Apache Hadoop版本分为两代，我们将第一代Hadoop称为Hadoop 1.0，第二代Hadoop称为Hadoop 2.0
- 第一代Hadoop包含三个大版本，分别是0.20.x，0.21.x和0.22.x，其中，0.20.x最后演化成1.0.x，变成了稳定版，而0.21.x和0.22.x则增加了NameNode HA等新的重大特性
- 第二代Hadoop包含两个版本，分别是0.23.x和2.x，它们完全不同于Hadoop 1.0，是一套全新的架构，均包含HDFS Federation和YARN两个系统，相比于0.23.x，2.x增加了NameNode HA和Wire-compatibility两个重大特性
- Hadoop 2.0是基于JDK 1.7开发的，而JDK 1.7在2015年4月已停止更新，于是Hadoop社区基于JDK1.8重新发布一个新的Hadoop版本，也就是Hadoop3.0

2.1.4 Apache Hadoop版本演变



2.1.5 Hadoop各种版本



- Apache Hadoop
- Hortonworks
- Cloudera (CDH: Cloudera Distribution Hadoop)
- MapR
-

选择 Hadoop 版本的考虑因素:

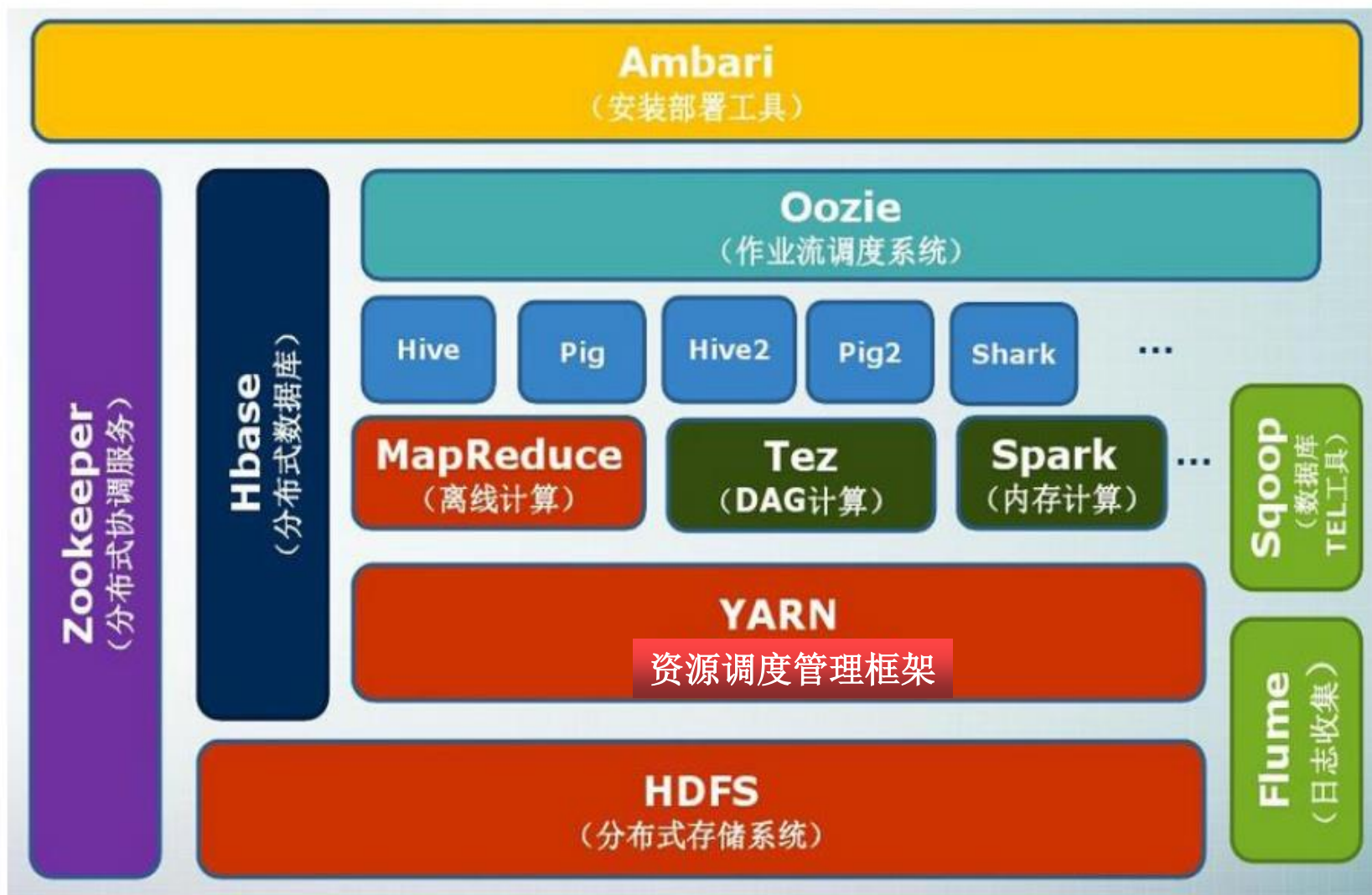
- 是否开源 (即是否免费)
- 是否有稳定版
- 是否经实践检验
- 是否有强大的社区支持

2.1.5 Hadoop各种版本

厂商名称	开放性	易用性 (★)	平台功能	性能 (★)	本地支持	总体评价 (★)
apache	完全开源、Hadoop就是托管在apache社区里面	安装: 2 使用: 2 维护: 2	Apache是标准的Hadoop平台, 所有厂商都是在apache的平台上面进行改进	2	没有	2
cloudera	与Apache功能同步, 部分代码开源	安装: 5 使用: 5 维护: 5	有自主研发的产品如: impala、navigator等	4.5	2014年刚进入中国, 上海	4.5
hortonworks	与apache功能同步, 也是完全开源	安装: 4.5 使用: 5 维护: 5	是apache hadoop平台的最大贡献者, 如 Tez	4.5	没有	4.5
MapR	在apache的hadoop版本上面修改很多	安装: 4.5 使用: 5 维护: 5	在apache平台上面优化很多、从而形成自己的产品	5	没有	3.5
星环	核心组件与apache同步、底层的优化比较多、完全封闭的一个平台	安装: 5 使用: 4 维护: 4	有自主的Hadoop产品如 Inceptor、Hyperbase	4	本地厂商	4

2.2 Hadoop项目结构

Hadoop的项目结构不断丰富发展，已经形成一个丰富的Hadoop生态系统

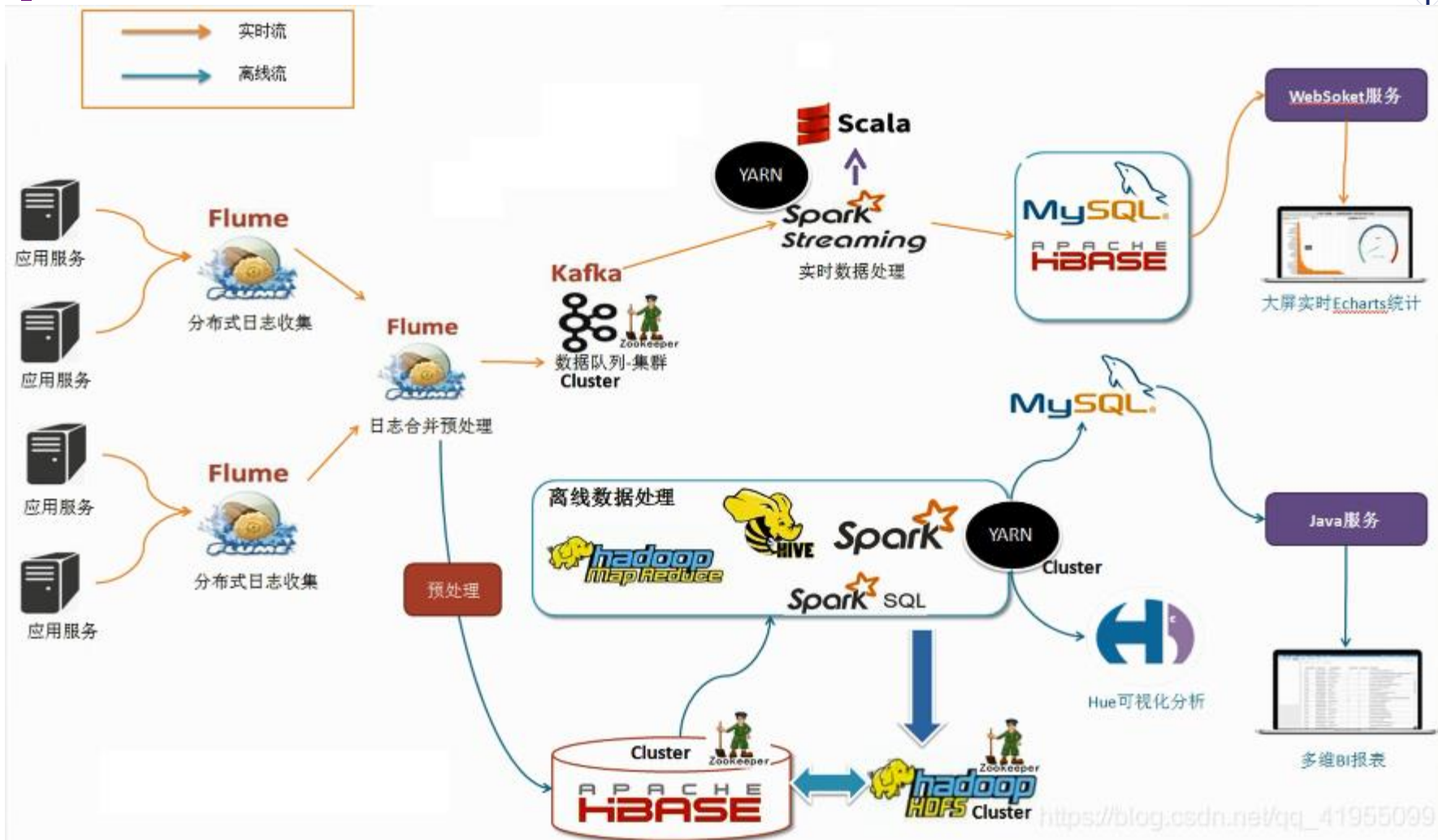


2.2 Hadoop项目结构



组件	功能
HDFS	分布式文件系统
MapReduce	分布式并行编程模型
YARN	资源管理和调度器
Tez	运行在YARN之上的下一代Hadoop查询处理框架
Hive	Hadoop上的数据仓库
HBase	Hadoop上的非关系型的分布式数据库
Pig	一个基于Hadoop的大规模数据分析平台，提供类似SQL的查询语言Pig Latin
Sqoop	用于在Hadoop与传统数据库之间进行数据传递
Oozie	Hadoop上的工作流管理系统
Zookeeper	提供分布式协调一致性服务
Storm	流计算框架
Flume	一个高可用的，高可靠的，分布式的海量日志采集、聚合和传输的系统
Ambari	Hadoop快速部署工具，支持Apache Hadoop集群的供应、管理和监控
Kafka	一种高吞吐量的分布式发布订阅消息系统，可以处理消费者规模的网站中的所有动作流数据
Spark	类似于Hadoop MapReduce的通用并行框架

2.2 Hadoop项目结构



2.3 Hadoop的安装与使用

- 2.3.1 Hadoop安装之前的预备知识
- 2.3.2 安装Linux虚拟机
- 2.3.3 安装双操作系统
- 2.3.4 详解Hadoop的安装与使用



高校大数据课程

公 共 服 务 平 台

详细安装教程请参考：《大数据技术原理与应用（第3版） 第二章 大数据处理架构Hadoop 学习指南》，给出了每步安装命令和效果截图
访问地址：<http://dbllab.xmu.edu.cn/blog/2544-2/>

2.3.1 Hadoop安装之前的预备知识

（一）Linux的选择

（1）选择哪个Linux发行版？

- 在Linux系统各个发行版中，CentOS系统和Ubuntu系统在服务端和桌面端使用占比最高，网络上资料最是齐全，所以建议使用CentOS 或Ubuntu
- 在学习Hadoop方面，虽然两个系统没有多大区别，但是推荐使用Ubuntu操作系统

（2）选择32位还是64位？

- 如果电脑比较老或者内存小于2G，那么建议选择32位系统版本的Linux
- 如果内存大于4G，那么建议选择64位系统版本的Linux

2.3.1 Hadoop安装之前的预备知识



(二) 系统安装方式：选择虚拟机安装还是双系统安装

- 建议电脑比较新或者配置内存**4G**以上的电脑可以选择虚拟机安装
- 电脑较旧或配置内存小于等于**4G**的电脑强烈建议选择双系统安装，否则，在配置较低的计算机上运行**Linux**虚拟机，系统运行速度会非常慢
- 鉴于目前教师和学生的计算机硬件配置一般不高，建议在实践教学中采用双系统安装，确保系统运行速度

（三）关于Linux的一些基础知识

•Shell

- 是指“提供使用者使用界面”的软件（命令解析器），类似于DOS下的command和后来的cmd.exe。它接收用户命令，然后调用相应的应用程序

•sudo命令

- sudo是ubuntu中一种权限管理机制，管理员可以授权给一些普通用户去执行一些需要root权限执行的操作。当使用sudo命令时，就需要输入您当前用户的密码

•输入密码

- 在Linux的终端中输入密码，终端是不会显示任何你当前输入的密码，也不会提示你已经输入了多少字符密码，读者不要误以为键盘没有响应

•输入法中英文切换

- linux中英文的切换方式是使用键盘“shift”键来切换，也可以点击顶部菜单的输入法按钮进行切换。Ubuntu自带的Sunpinyin中文输入法已经足够读者使用

•Ubuntu终端复制粘贴快捷键

- 在Ubuntu终端窗口中，复制粘贴的快捷键需要加上 shift，即粘贴是ctrl+shift+v

（四）Hadoop安装方式

•Hadoop包括三种安装模式：

- 单机模式：只在一台机器上运行，存储是采用本地文件系统，没有采用分布式文件系统HDFS；
- 伪分布式模式：存储采用分布式文件系统HDFS，但是，HDFS的名称节点和数据节点都在同一台机器上；
- 分布式模式：存储采用分布式文件系统HDFS，而且，HDFS的名称节点和数据节点位于不同机器上。

2.3.2 安装Linux虚拟机



一、材料和工具

1、下载VirtualBox虚拟机软件

<https://download.virtualbox.org/virtualbox/6.1.4/VirtualBox-6.1.4-136177-Win.exe>

2. 下载Ubuntu LTS 16.04（或18.04）ISO映像文件

Ubuntu LTS 16.04下载: <https://www.ubuntu.org.cn/download/ubuntu-kylin>

Ubuntu LTS 18.04下载: <https://ubuntu.com/download/desktop>

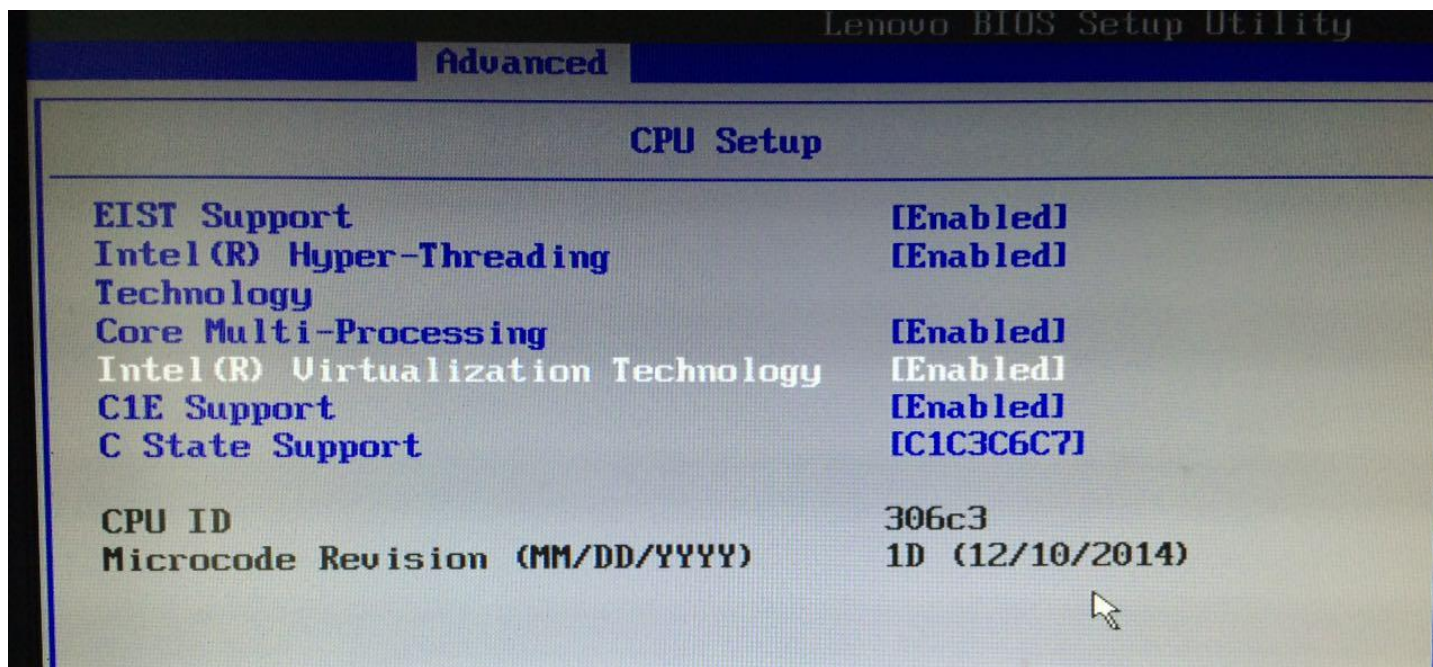
2.3.2 安装Linux虚拟机



二、步骤

(一) 确认系统版本

如果选择的系统是64位Ubuntu系统，那么在安装虚拟机前，我们还要进入BIOS开启CPU的虚拟化



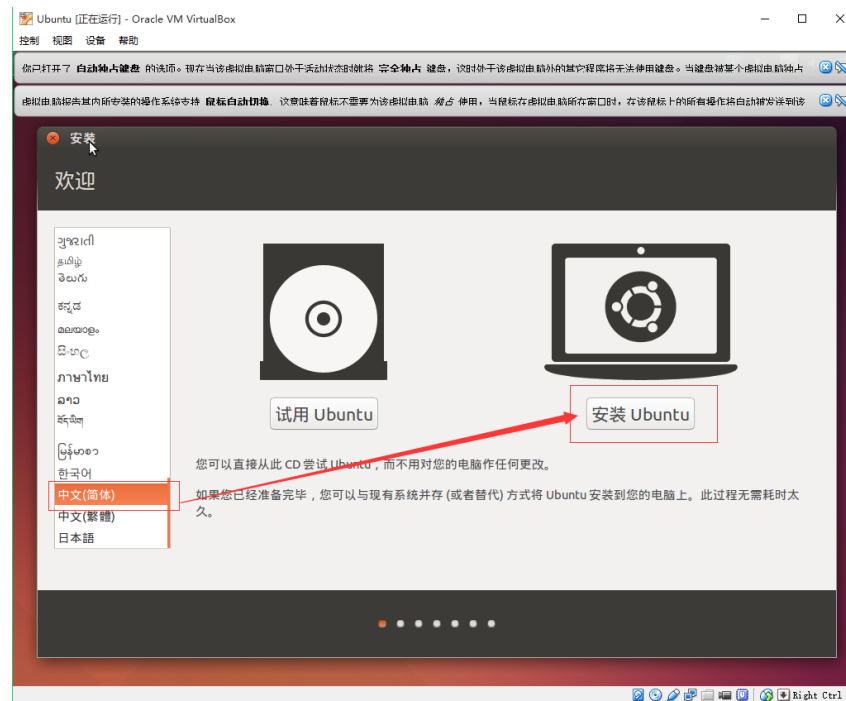
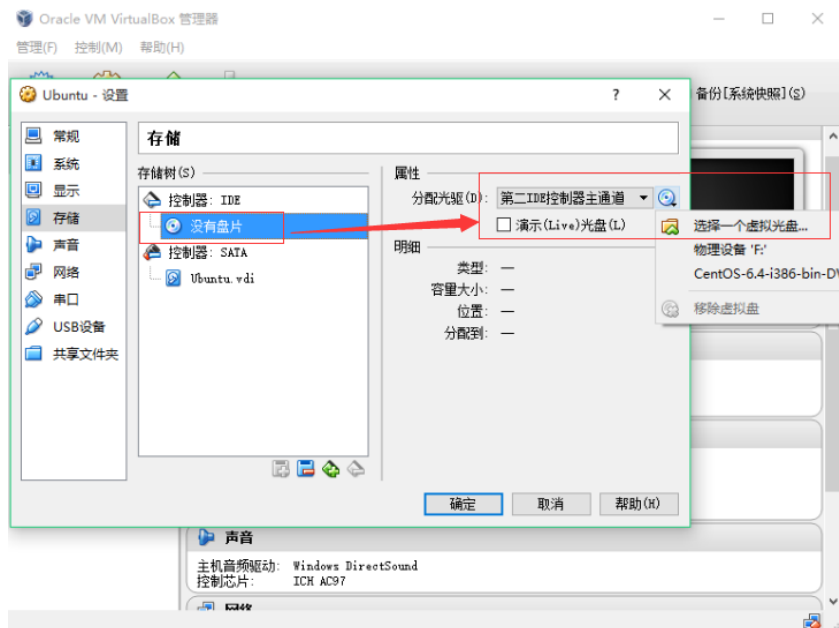
(二)安装前的准备

- 1.打开VirtualBox，点击“创建”按钮，创建一个虚拟机
- 2.给虚拟机命名，选择操作系统，版本
- 3.选择内存大小，这里设置的1024M
- 4.创建虚拟硬盘
- 5.选择虚拟硬盘文件类型VDI
- 6.虚拟硬盘选择动态分配
- 7.选择文件存储的位置和容量大小
- 8.点击创建



2.3.2 安装Linux虚拟机

(三)安装Ubuntu



2.3.3 安装双操作系统

- 第一步：制作安装U盘
- 具体可参考百度经验文章
- <http://jingyan.baidu.com/article/59703552e0a6e18fc007409f.html>
- 第二步：双系统安装
- 具体可参考百度经验文章
- <http://jingyan.baidu.com/article/dca1fa6fa3b905f1a44052bd.html>

安装后Window和Ubuntu 16.04（或18.04）都可以用，默认
windows优先启动

可以在电脑启动时，选择进入Ubuntu系统而不是 Windows系统

2.3.4 Hadoop的安装与使用（单机/伪分布式）



Hadoop基本安装配置主要包括以下几个步骤：

- 创建Hadoop用户
- SSH登录权限设置
- 安装Java环境
- 单机安装配置
- 伪分布式安装配置

详细安装配置过程请参考

《**Hadoop安装教程_单机/伪分布式配置_Hadoop3.1.3/Ubuntu18.04**》

<http://dblab.xmu.edu.cn/blog/2441-2/>

创建Hadoop用户

如果安装 Ubuntu 的时候不是用的 “hadoop” 用户，那么需要增加一个名为 hadoop 的用户

首先按 **ctrl+alt+t** 打开终端窗口，输入如下命令创建新用户：

```
$ sudo useradd -m hadoop -s /bin/bash
```

上面这条命令创建了可以登陆的 hadoop 用户，并使用 /bin/bash 作为 shell

接着使用如下命令设置密码，可简单设置为 hadoop，按提示输入两次密码：

```
$ sudo passwd hadoop
```

可为 hadoop 用户增加管理员权限，方便部署，避免一些对新手来说比较棘手的权限问题：

```
$ sudo adduser hadoop sudo
```

SSH是什么？

SSH 为 Secure Shell 的缩写，是建立在应用层和传输层基础上的安全协议。SSH 是目前较可靠、专为远程登录会话和其他网络服务提供安全性的协议。利用 SSH 协议可以有效防止远程管理过程中的信息泄露问题。SSH最初是UNIX系统上的一个程序，后来又迅速扩展到其他操作平台。SSH是由[客户端](#)和[服务端](#)的软件组成，服务端是一个守护进程(daemon)，它在后台运行并响应来自客户端的连接请求，客户端包含ssh程序以及像scp（远程拷贝）、slogin（远程登陆）、sftp（安全文件传输）等其他的应用程序

配置SSH的原因：

Hadoop名称节点（NameNode）需要启动集群中所有机器的Hadoop守护进程，这个过程需要通过SSH登录来实现。Hadoop并没有提供SSH输入密码登录的形式，因此，为了能够顺利登录每台机器，需要将所有机器配置为名称节点可以无密码登录它们

安装Java环境

- Java环境可选择 Oracle 的 JDK，或是 OpenJDK
- 建议采用手工方式安装Java环境
 - 具体请参考网络教程：<http://dblab.xmu.edu.cn/blog/2441-2/>
 - 到Java官网下载安装文件jdk-8u162-linux-x64.tar.gz
 - 在Linux命令行界面中，执行如下Shell命令（注意：当前登录用户名是hadoop）：

```
$cd /usr/lib  
$sudo mkdir jvm #创建/usr/lib/jvm目录用来存放JDK文件  
$cd ~ #进入hadoop用户的主目录  
$cd Downloads #注意区分大小写字母，刚才已经通过FTP软件把JDK安装包jdk-8u162-linux-x64.tar.gz上传到该目录下  
$sudo tar -zxvf ./jdk-8u162-linux-x64.tar.gz -C /usr/lib/jvm #把JDK文件解压到/usr/lib/jvm目录下
```

- 下面继续执行如下命令，设置环境变量：

```
$cd ~  
$vim ~/.bashrc
```

请在这个文件的开头位置，添加如下几行内容：

```
export JAVA_HOME=/usr/lib/jvm/jdk1.8.0_162  
export JRE_HOME=${JAVA_HOME}/jre  
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib  
export PATH=${JAVA_HOME}/bin:$PATH
```

继续执行如下命令让.bashrc文件的配置立即生效：

```
$source ~/.bashrc
```


- 这时，可以使用如下命令查看是否安装成功：

```
$java -version
```

如果能够在屏幕上返回如下信息，则说明安装成功：

```
hadoop@ubuntu:~$ java -version  
java version "1.8.0_162"  
Java(TM) SE Runtime Environment (build 1.8.0_162-b12)  
Java HotSpot(TM) 64-Bit Server VM (build 25.162-b12, mixed mode)
```

Hadoop 3 安装文件的下载

Hadoop 3 可以到官网下载，需要下载 **hadoop-3.1.3.tar.gz** 这个格式的文件，这是编译好的，另一个包含 **src** 的则是 Hadoop 源代码，需要进行编译才可使用

- 如果读者是使用虚拟机方式安装Ubuntu系统的用户，请用虚拟机中的Ubuntu自带firefox浏览器访问本指南，再点击下载地址，才能把hadoop文件下载虚拟机ubuntu中。请不要使用Windows系统下的浏览器下载，文件会被下载到Windows系统中，虚拟机中的Ubuntu无法访问外部Windows系统的文件，造成不必要的麻烦。
- 如果读者是使用双系统方式安装Ubuntu系统的用户，请进去Ubuntu系统，在Ubuntu系统打开firefox浏览器，再点击下载

选择将 Hadoop 安装至 /usr/local/ 中

```
$ sudo tar -zxf ~/下载/hadoop-3.1.3.tar.gz -C /usr/local # 解压到/usr/local中
$ cd /usr/local/
$ sudo mv ./hadoop-3.1.3/ ./hadoop # 将文件夹名改为hadoop
$ sudo chown -R hadoop:hadoop ./hadoop # 修改文件权限
```

Hadoop 解压后即可使用。输入如下命令来检查 Hadoop 是否可用，成功则会显示 Hadoop 版本信息：

```
$ cd /usr/local/hadoop
$ ./bin/hadoop version
```

Hadoop 默认模式为非分布式模式（本地模式），无需进行其他配置即可运行。

伪分布式安装配置

- Hadoop 可以在单节点上以伪分布式的方式运行，Hadoop 进程以分离的 Java 进程来运行，节点既作为 NameNode 也作为 DataNode，同时，读取的是 HDFS 中的文件
- Hadoop 的配置文件位于 `/usr/local/hadoop/etc/hadoop/` 中，伪分布式需要修改2个配置文件 **core-site.xml** 和 **hdfs-site.xml**
- Hadoop的配置文件是 xml 格式，每个配置以声明 property 的 name 和 value 的方式来实现

伪分布式安装配置

实验步骤:

- ❑ 修改配置文件: core-site.xml, hdfs-site.xml, mapred-site.xml
- ❑ 初始化文件系统hadoop namenode -format
- ❑ 启动所有进程start-all.sh
- ❑ 访问web界面, 查看Hadoop信息
- ❑ 运行实例

修改配置文件 **core-site.xml**

```
<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>file:/usr/local/hadoop/tmp</value>
    <description>Abase for other temporary directories.</description>
  </property>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

- **hadoop.tmp.dir**表示存放临时数据的目录，即包括**NameNode**的数据，也包括**DataNode**的数据。该路径任意指定，只要实际存在该文件夹即可
- **name**为**fs.defaultFS**的值，表示**hdfs**路径的逻辑名称

修改配置文件 **hdfs-site.xml**

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop/tmp/dfs/name</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/usr/local/hadoop/tmp/dfs/data</value>
  </property></configuration>
```

- **dfs.replication**表示副本的数量，伪分布式要设置为1
- **dfs.namenode.name.dir**表示本地磁盘目录，是存储**fsimage**文件的地方
- **dfs.datanode.data.dir**表示本地磁盘目录，HDFS数据存放**block**的地方

关于三种Shell命令方式的区别：

1. `hadoop fs`
2. `hadoop dfs`
3. `hdfs dfs`

- `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

- 2.4.1 集群节点类型
- 2.4.2 集群规模
- 2.4.3 集群硬件配置
- 2.4.4 集群网络拓扑
- 2.4.5 集群的建立与安装
- 2.4.6 集群基准测试
- 2.4.7 在云计算环境中使用Hadoop

2.4.1 Hadoop集群中有哪些节点类型



- Hadoop框架中最核心的设计是为海量数据提供存储的**HDFS**和对数据进行计算的**MapReduce**
- MapReduce**的作业主要包括：（1）从磁盘或从网络读取数据，即**IO**密集工作；（2）计算数据，即**CPU**密集工作
- Hadoop**集群的整体性能取决于**CPU**、内存、网络以及存储之间的性能平衡。因此运营团队在选择机器配置时要针对不同的工作节点选择合适硬件类型
- 一个基本的**Hadoop**集群中的节点主要有
 - NameNode**: 负责协调集群中的数据存储
 - DataNode**: 存储被拆分的数据块
 - JobTracker**: 协调数据计算任务
 - TaskTracker**: 负责执行由**JobTracker**指派的任务
 - SecondaryNameNode**: 帮助**NameNode**收集文件系统运行的状态信息

2.4.2 集群硬件配置



在集群中，大部分的机器设备是作为Datanode和TaskTracker工作的Datanode/TaskTracker的硬件规格可以采用以下方案：

- 4个磁盘驱动器（单盘1-2T），支持JBOD(Just a Bunch Of Disks，磁盘簇)
- 2个4核CPU,至少2-2.5GHz
- 16-24GB内存
- 千兆以太网

NameNode提供整个HDFS文件系统的NameSpace(命名空间)管理、块管理等所有服务，因此需要更多的RAM，与集群中的数据块数量相对应，并且需要优化RAM的内存通道带宽，采用双通道或三通道以上内存。硬件规格可以采用以下方案：

- 8-12个磁盘驱动器（单盘1-2T）
- 2个4核/8核CPU
- 16-72GB内存
- 千兆/万兆以太网

SecondaryNameNode在小型集群中可以 and NameNode 共用一台机器，较大的群集可以采用与NameNode相同的硬件

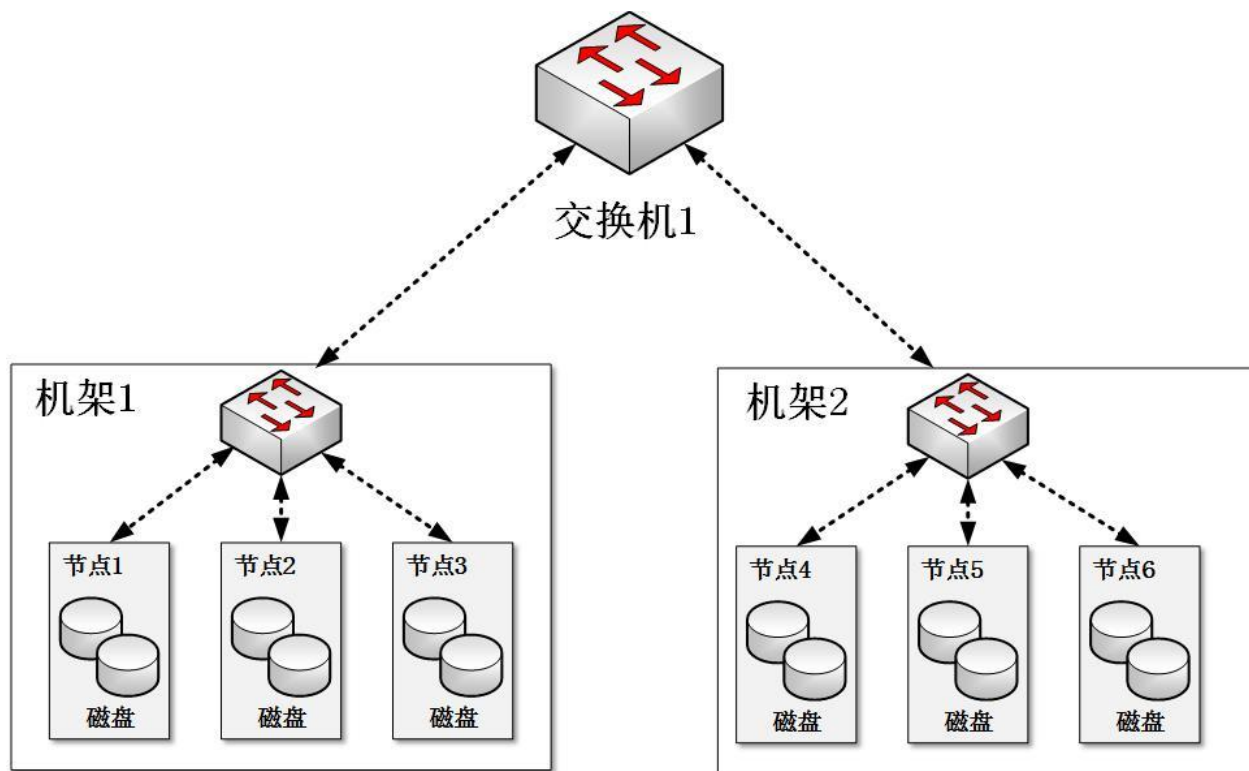
2.4.3 集群规模要多大



- **Hadoop**集群规模可大可小，初始时，可以从一个较小规模的集群开始，比如包含**10**个节点，然后，规模随着存储器和计算需求的扩大而扩大
- 如果数据每周增大**1TB**，并且有三个**HDFS**副本，然后每周需要一个额外的**3TB**作为原始数据存储。要允许一些中间文件和日志（假定**30%**）的空间，由此，可以算出每周大约需要增加一台新机器。存储两年数据的集群，大约需要**100**台机器
- 对于一个小的集群，名称节点（**NameNode**）和**JobTracker**运行在单个节点上，通常是可以接受的。但是，随着集群和存储在**HDFS**中的文件数量的增加，名称节点需要更多的主存，这时，名称节点和**JobTracker**就需要运行在不同的节点上
- 第二名称节点（**SecondaryNameNode**）会和名称节点可以运行在相同的机器上，但是，由于第二名称节点和名称节点几乎具有相同的主存需求，因此，二者最好运行在不同节点上

2.4.4 集群网络拓扑

- 普通的Hadoop集群结构由一个两阶网络构成
- 每个机架（Rack）有30-40个服务器，配置一个1GB的交换机，并向上传输到一个核心交换机或者路由器（1GB或以上）
- 在相同的机架中的节点间的带宽的总和，要大于不同机架间的节点间的带宽总和



2.4.5 集群的建立与安装



采购好相关的硬件设备后，就可以把硬件装入机架，安装并运行Hadoop
安装Hadoop有多种方法：

- (1) 手动安装
- (2) 自动化安装
 - 为了缓解安装和维护每个节点上相同的软件的负担，可以使用一个自动化方法实现完全自动化安装，比如Red Hat Linux' Kickstart、Debian或者Docker
 - 自动化安装部署工具，会通过记录在安装过程中对于各个选项的回答来完成自动化安装过程。

2.4.6 Hadoop集群基准测试



- 如何判断一个Hadoop集群是否已经正确安装？可以运行基准测试
- Hadoop自带有一些基准测试程序，被打包在测试程序JAR文件中
- 用TestDFSIO基准测试，来测试HDFS的IO性能
- 用排序测试MapReduce：Hadoop自带一个部分排序的程序，这个测试过程的整个数据集都会通过洗牌（Shuffle）传输至Reducer，可以充分测试MapReduce的性能

2.4.7 在云计算环境中使用Hadoop



- Hadoop不仅可以运行在企业内部的集群中，也可以运行在云计算环境中
- 可以在Amazon EC2中运行Hadoop。EC2是一个计算服务，允许客户租用计算机（实例），来运行自己的应用。客户可以按需运行或终止实例，并且按照实际使用情况来付费
- Hadoop自带有一套脚本，用于在EC2上面运行Hadoop
- 在EC2上运行Hadoop尤其适用于一些工作流。例如，在Amazon S3中存储数据，在EC2上运行集群，在集群中运行MapReduce作业，读取存储在S3中的数据，最后，在关闭集群之前将输出写回S3中；如果长期使用集群，复制S3数据到运行在EC2上的HDFS中，则可以使得数据处理更加高效，因为，HDFS可以充分利用数据的位置，S3则做不到，因为，S3与EC2的存储不在同一个节点上

本章小结

- Hadoop被视为事实上的大数据处理标准，本章介绍了Hadoop的发展历程，并阐述了Hadoop的高可靠性、高效性、高可扩展性、高容错性、成本低、运行在Linux平台上、支持多种编程语言等特性
- Hadoop目前已经在各个领域得到了广泛的应用，雅虎、Facebook、百度、淘宝、网易等公司都建立了自己的Hadoop集群
- 经过多年发展，Hadoop项目已经变得非常成熟和完善，包括Common、Avro、Zookeeper、HDFS、MapReduce、HBase、Hive、Chukwa、Pig等子项目，其中，HDFS和MapReduce是Hadoop的两大核心组件
- 本章最后介绍了如何在Linux系统下完成Hadoop的安装和配置，这个部分是后续章节实践环节的基础

第二部分 HDFS

5.1 分布式文件系统

- 5.1.1 计算机集群结构
- 5.1.2 分布式文件系统的结构

5.1.1 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

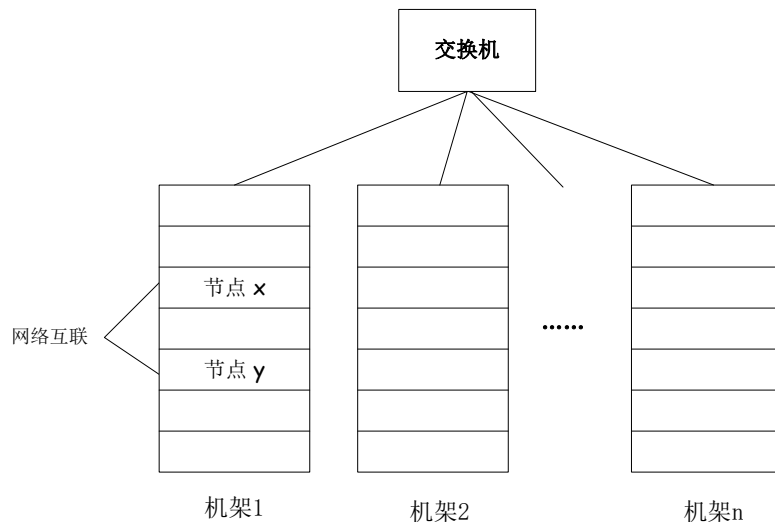


图3-1 计算机集群的基本架构

5.1.2 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)

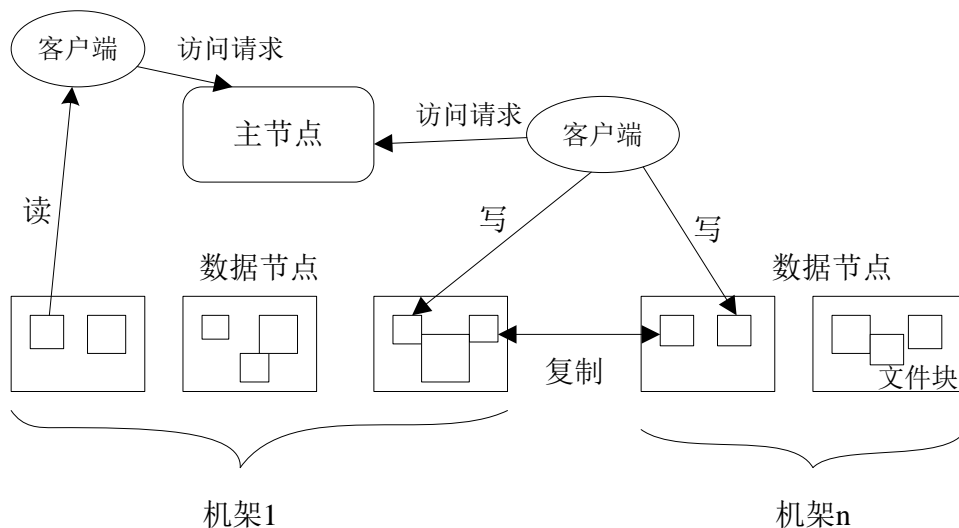


图3-2 大规模文件系统的整体结构

5.2 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：

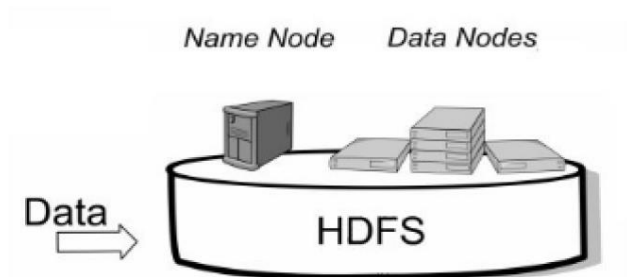
- 不适合低延迟数据访问
- 无法高效存储大量小文件
- 不支持多用户写入及任意修改文件

5.5.1 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

5.5.2 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

5.5.2 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

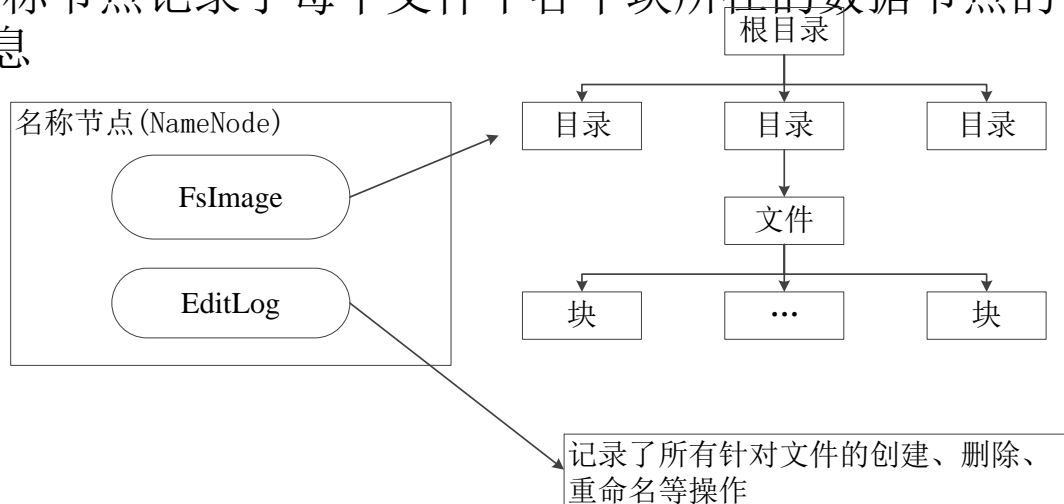


图3-3 名称节点的数据结构

5.5.2 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。

5.5.2 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

5.5.2 名称节点和数据节点

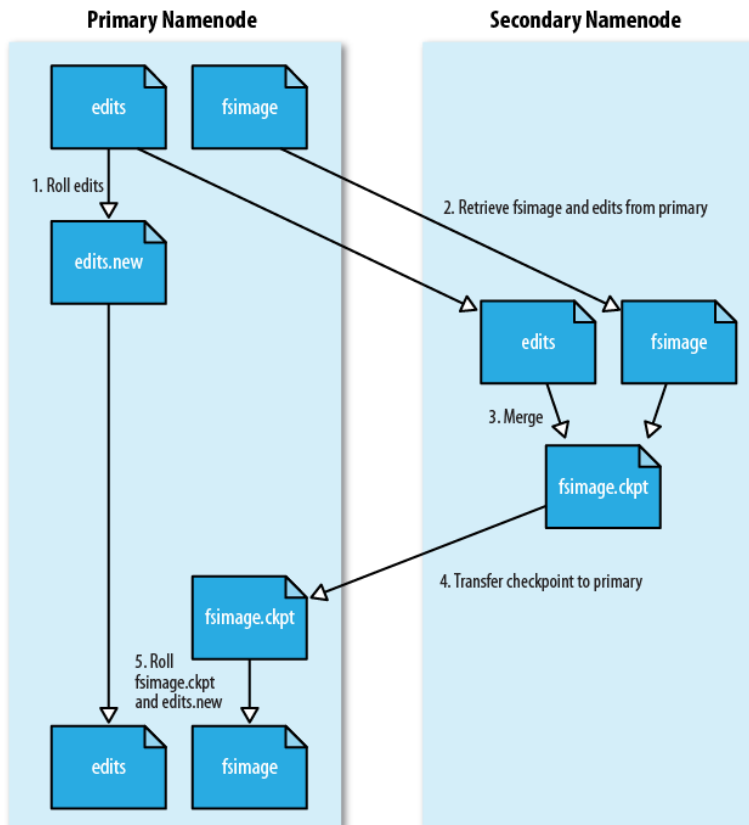
名称节点运行期间EditLog不断变大的问题

- 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

5.5.2 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

5.5.2 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

5.4HDFS体系结构

- 5.4.1 HDFS体系结构概述
- 5.4.2 HDFS命名空间管理
- 5.4.3 通信协议
- 5.4.4 客户端
- 5.4.5 HDFS体系结构的局限性

The diagram illustrates the HDFS architecture. At the top, the **客户端 (Client)** and **名称节点 (NameNode)** are connected by two arrows: one labeled **文件名或数据块号** (file name or data block number) pointing from Client to NameNode, and another labeled **数据块号、数据块位置** (data block number, data block location) pointing from NameNode to Client. Below the NameNode, a horizontal line represents the network. From this line, arrows point to multiple **数据节点 (DataNode)** boxes. The left side shows **机架1** (Rack 1) containing two DataNodes, each with a **本地Linux文件系统** (local Linux file system). The right side shows **机架n** (Rack n) also containing two DataNodes with local file systems. Arrows labeled **写数据** (write data) and **读数据** (read data) show the flow between the Client and the DataNodes. A double-headed arrow labeled **备份** (backup) connects the DataNodes across racks. Ellipses indicate additional racks and DataNodes.

图3-4 HDFS体系结构

5.4.2 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

5.4.3 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

5.4.4 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

5.4.5 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。

5.5 HDFS 存储原理

- 5.5.1 冗余数据保存
- 5.5.2 数据存取策略
- 5.5.3 数据错误与恢复

5.5.1 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

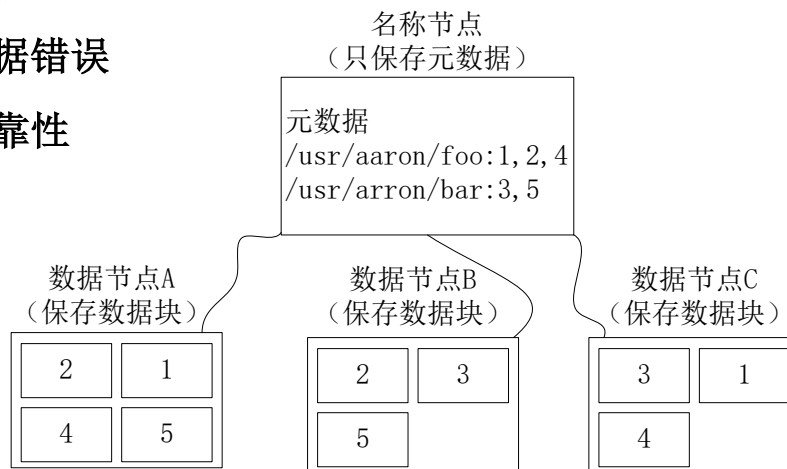
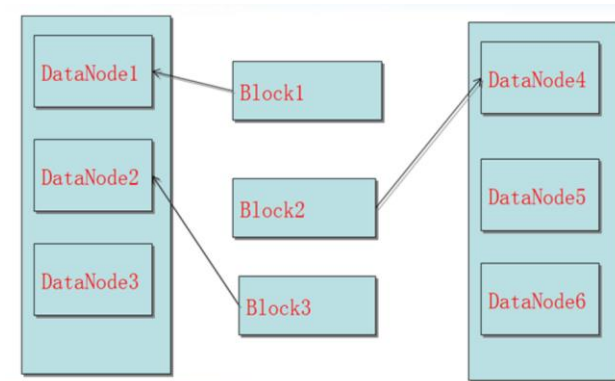


图3-5 HDFS数据块多副本存储

5.5.2 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



5.5.2 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

5.5.3 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。

5.5.3 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

5.5.3 数据错误与恢复

5. 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

5.6 HDFS数据读写过程

- 5.6.1 读数据的过程
- 5.6.2 写数据的过程

5.6 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();
        }
    }
}
```

5.6 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();
        }
    }
}
```

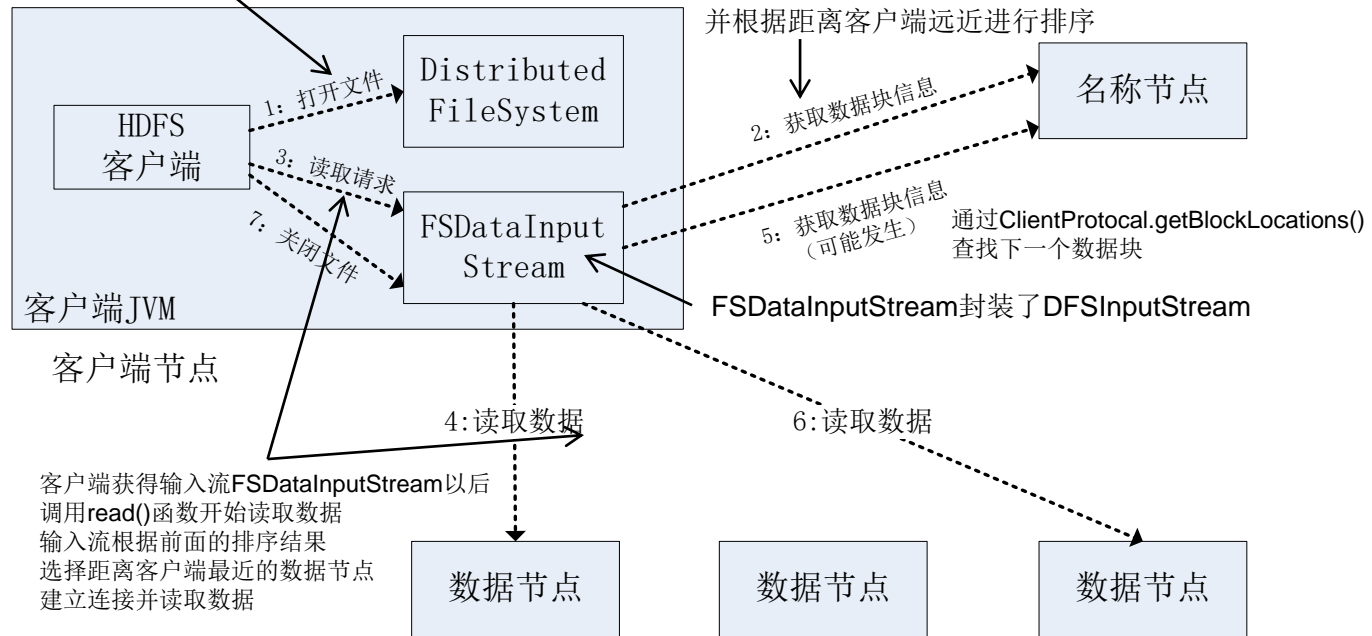
5.6 HDFS数据读写过程

- **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文件系统的具体实现
- **FileSystem**的**open()**方法返回的是一个输入流

```
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));
```

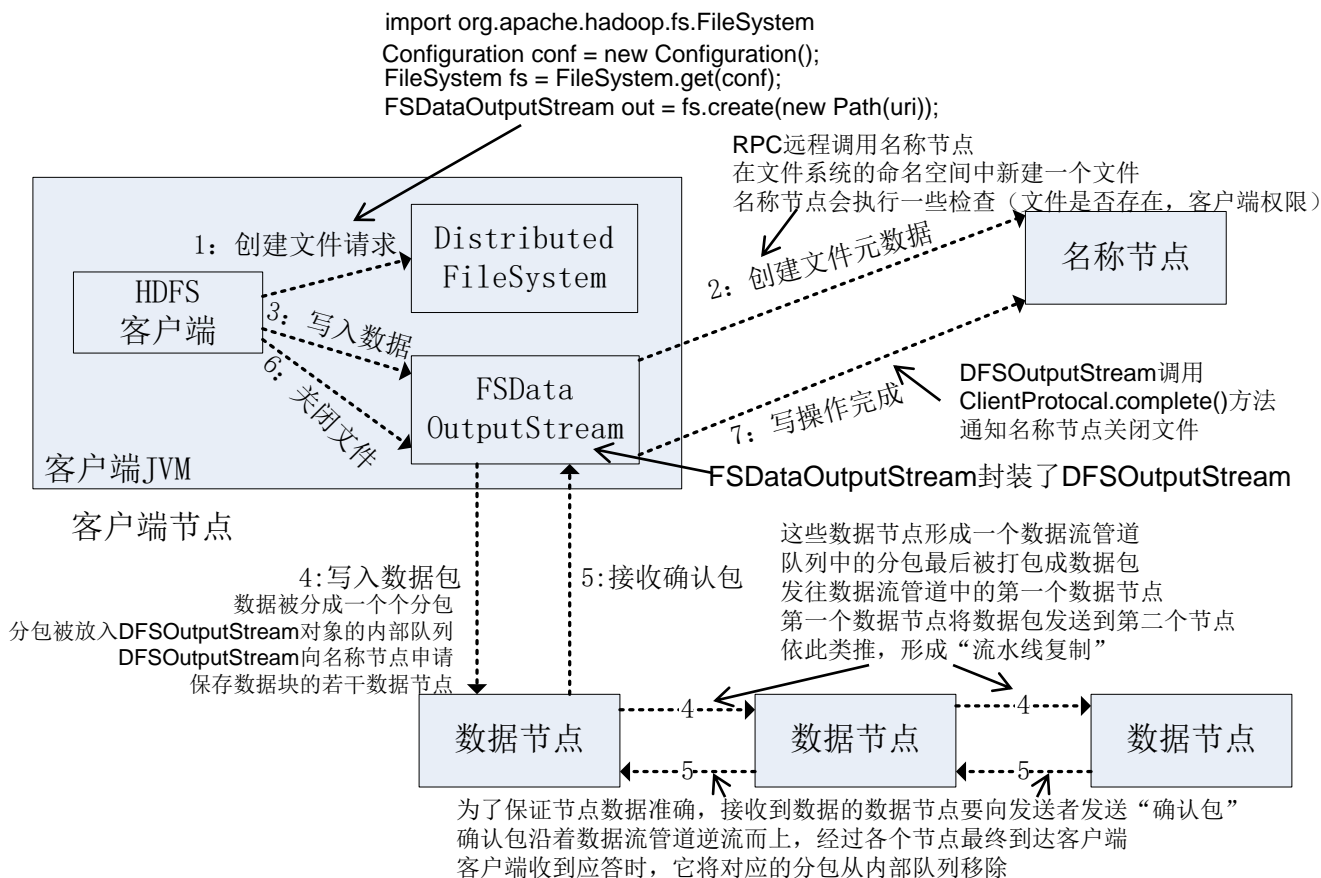
5.6.1 读数据的过程

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



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

5.6.2 写数据的过程



5.7 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：

2460-2/



高校大数据课程

公 共 服 务 平 台

5.7 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

```
$ 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
```

5.7.1 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

5.7.1 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

```
hadoop@ubuntu:/usr/local/hadoop$ ./bin/hadoop fs -mkdir
hdfs://localhost:9000/tempDir
hadoop@ubuntu:/usr/local/hadoop$ ./bin/hadoop fs -ls hdfs://localhost:9000/
Found 6 items
drwxr-xr-x  - hadoop supergroup          0 2020-02-06 10:04 hdfs://localhost:9000/bigdatacase
drwxr-xr-x  - hadoop supergroup          0 2020-02-19 19:39 hdfs://localhost:9000/hbase
drwxr-xr-x  - hadoop supergroup          0 2020-02-27 10:57 hdfs://localhost:9000/tempDir
drwx-wx-wx  - hadoop supergroup          0 2020-01-30 09:38 hdfs://localhost:9000/tmp
drwxr-xr-x  - hadoop supergroup          0 2020-01-30 09:13 hdfs://localhost:9000/user
drwxr-xr-x  - hadoop supergroup          0 2020-01-30 09:21 hdfs://localhost:9000/usr
```

5.7.1 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

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

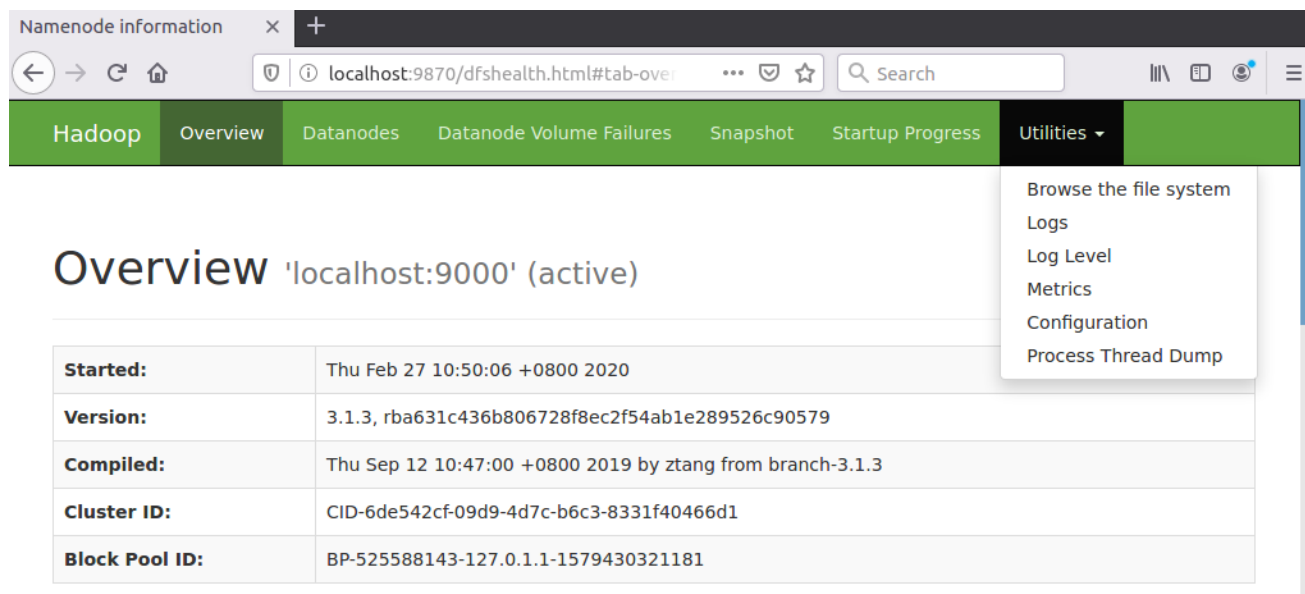
```
hadoop@ubuntu:/usr/local/hadoop$ ./bin/hadoop fs -cat hdfs://localhost:9000/tempDir/*2020-02-27 11:08:25,938 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localhostTrusted = false, remoteHostTrusted = false
this is file1.txt
this is file2.txt
this is file3.txt
this is file4.abc
this is file5.abc
```

5.7.2 HDFS的Web界面

在配置好Hadoop集群之后，可以通过浏览器登录

“http://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



The screenshot shows the Hadoop Web UI interface. The top navigation bar includes links for Hadoop, Overview, Datanodes, Datanode Volume Failures, Snapshot, Startup Progress, and Utilities. The Utilities menu is open, showing options: Browse the file system, Logs, Log Level, Metrics, Configuration, and Process Thread Dump. The main content area displays the Overview page for 'localhost:9000' (active).

Overview 'localhost:9000' (active)	
Started:	Thu Feb 27 10:50:06 +0800 2020
Version:	3.1.3, rba631c436b806728f8ec2f54ab1e289526c90579
Compiled:	Thu Sep 12 10:47:00 +0800 2019 by ztang from branch-3.1.3
Cluster ID:	CID-6de542cf-09d9-4d7c-b6c3-8331f40466d1
Block Pool ID:	BP-525588143-127.0.1.1-1579430321181

5.7.3 HDFS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/>

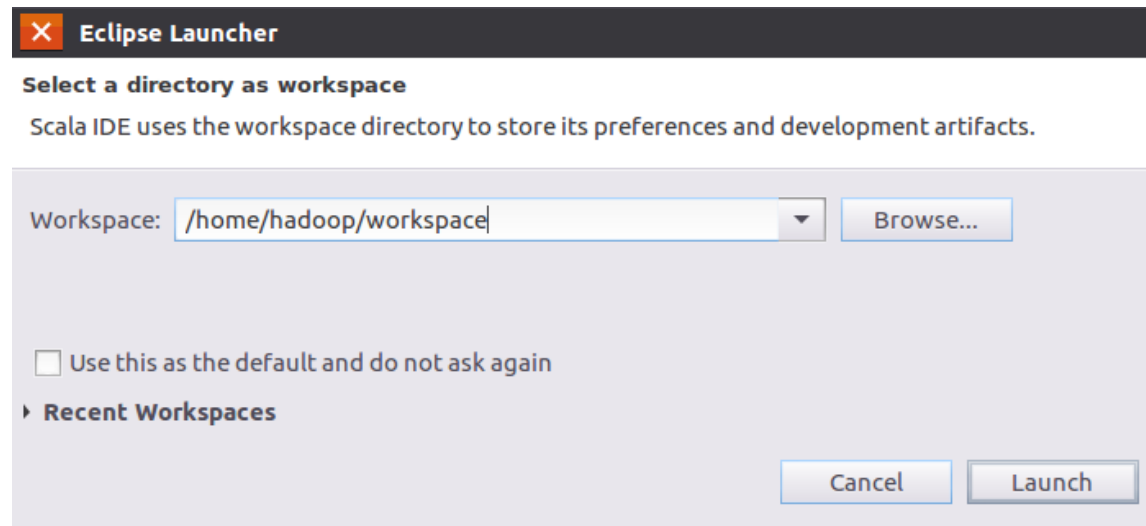
5.7.3 HDFS常用Java API及应用实例

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`、`file5.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。

5.7.3 HDFS常用Java API及应用实例

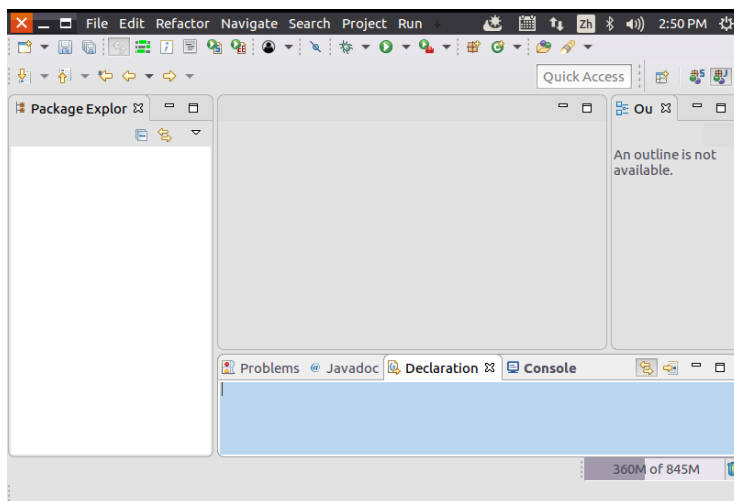
一、在Eclipse中创建项目

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）。



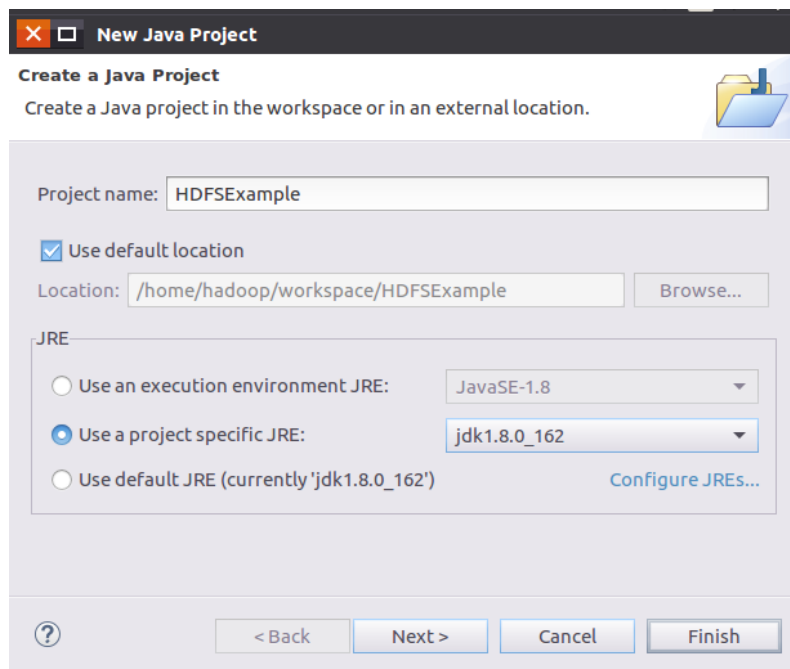
5.7.3 HDFS常用Java API及应用实例

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。



5.7.3 HDFS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。



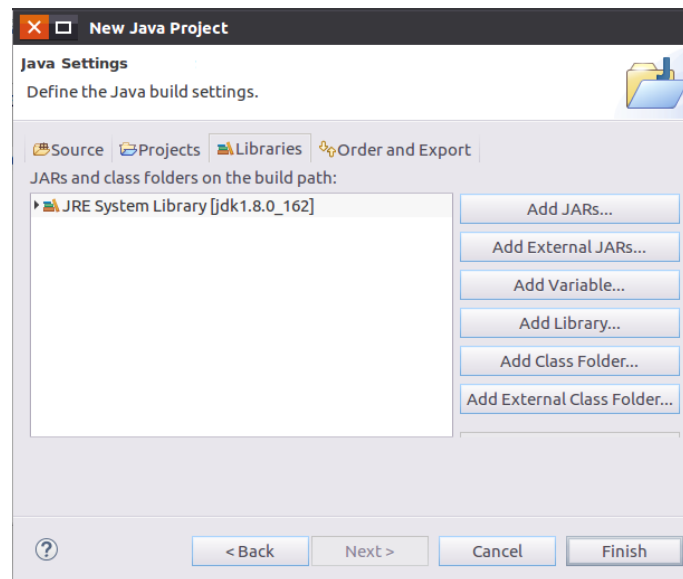
5.7.3 HDFS常用Java API及应用实例

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。

5.7.3 HDFS常用Java API及应用实例

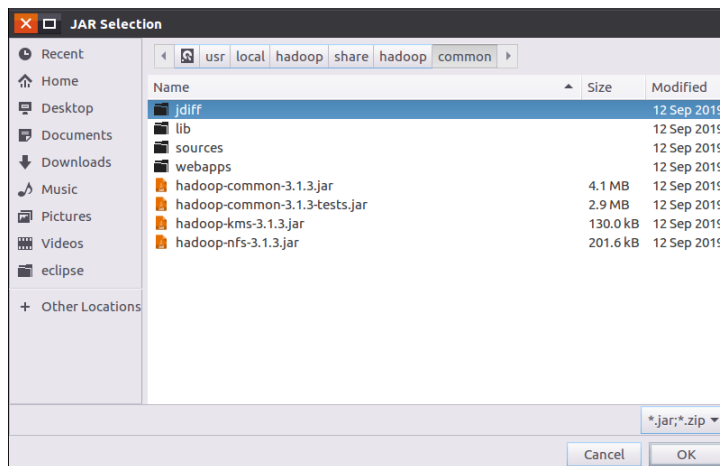
二、为项目添加需要用到的JAR包

进入下一步的设置以后，会弹出如下图所示界面。



5.7.3 HDFS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。



5.7.3 HDFS常用Java API及应用实例

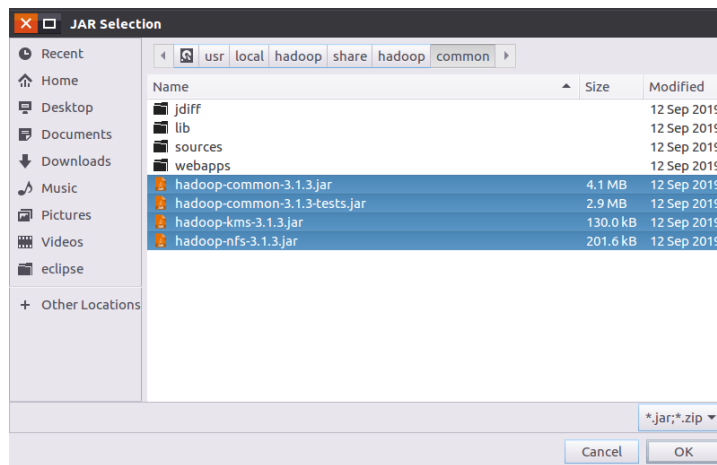
在该界面中，上面的一排目录按钮（即“usr”、“local”、“hadoop”、“share”、“hadoop”和“common”），当点击某个目录按钮时，就会在下面列出该目录的内容。

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-5.1.5.jar、hadoop-common-5.1.3-tests.jar、hadoop-nfs-5.1.5.jar和hadoop-kms-5.1.5.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。

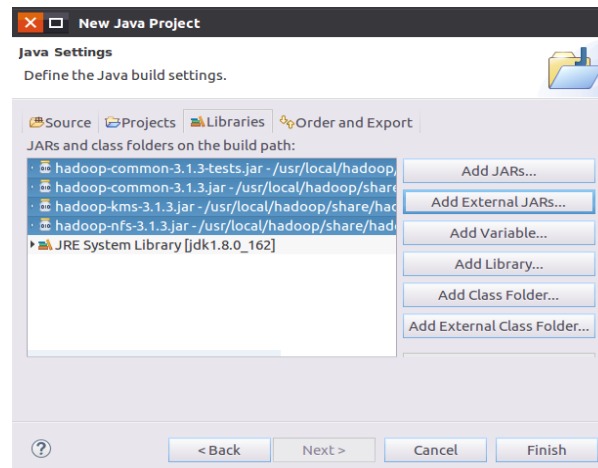
5.7.3 HDFS常用Java API及应用实例

比如，如果要把“/usr/local/hadoop/share/hadoop/common”目录下的hadoop-common-5.1.5.jar、hadoop-common-5.1.3-tests.jar、hadoop-nfs-5.1.5.jar和hadoop-kms-5.1.5.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）。



5.7.3 HDFS常用Java API及应用实例

请在界面中用鼠标点击选中hadoop-common-5.1.5.jar、hadoop-common-5.1.3-tests.jar、hadoop-nfs-5.1.5.jar和hadoop-kms-5.1.5.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。



5.7.3 HDFS常用Java API及应用实例

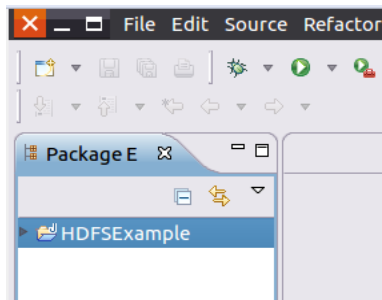
从这个界面中可以看出，hadoop-common-5.1.5.jar、hadoop-common-5.1.3-tests.jar、hadoop-nfs-5.1.5.jar和hadoop-kms-5.1.5.jar已经被添加到当前Java工程中。然后，按照类似的操作方法，可以再次点击“Add External JARs...”按钮，把剩余的其他JAR包都添加进来。需要注意的是，当需要选中某个目录下的所有JAR包时，可以使用“Ctrl+A”组合键进行全选操作。全部添加完毕以后，就可以点击界面右下角的“Finish”按钮，完成Java工程HDFSExample的创建。

5.7.3 HDFS常用Java API及应用实例

三、编写Java应用程序

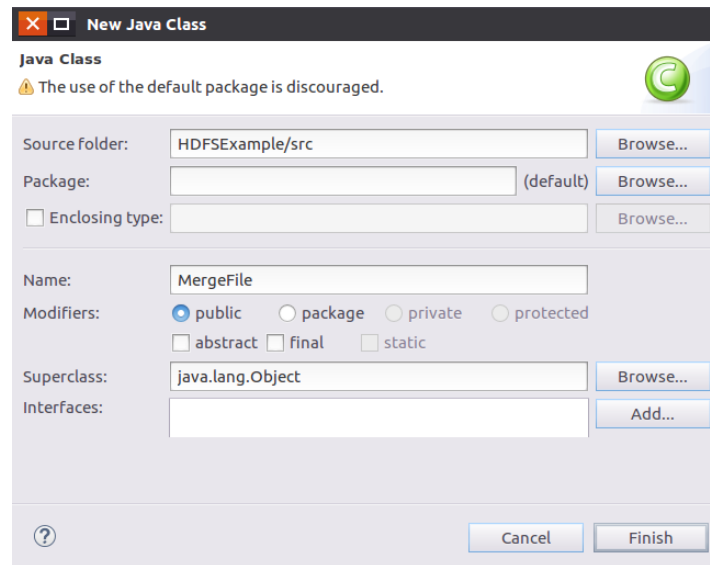
下面编写一个Java应用程序，用来检测HDFS中是否存在一个文件。

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。



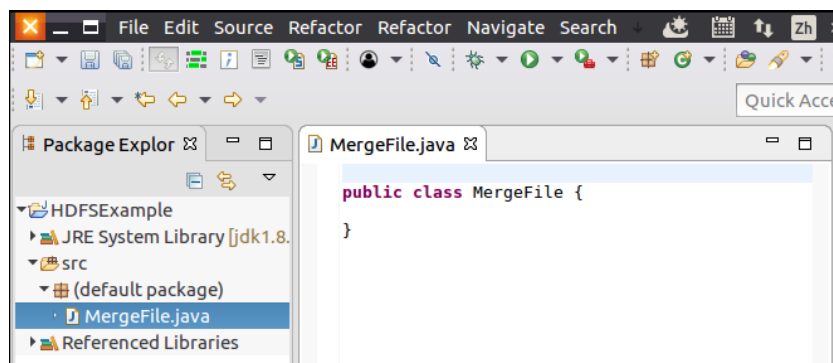
5.7.3 HDFS常用Java API及应用实例

选择“New→Class”菜单以后会出现如下图所示界面。



5.7.3 HDFS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。



5.7.3 HDFS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.InputStream;
import java.io.OutputStream;
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 false;
        return true;
    }
}

/**
 * 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");
        FileSystem fsSource = FileSystem.get(URI.create(inputPath.toString()), conf);
        FileSystem fsDst = FileSystem.get(URI.create(outputPath.toString()), conf);
        FileStatus[] sourceStatus = fsSource.listStatus(inputPath);
        FSDataOutputStream fsdos = fsDst.create(outputPath);
        PrintStream ps = new PrintStream(System.out);
        //下面分别读取上述之后的每个文件的内容，并输出到同一个文件中
        for (FileStatus sta : sourceStatus) {
            //下面打印后跟不为abc的文件的名称、文件大小
            System.out.print("源文件: " + sta.getPath() + " 文件大小: " + sta.getLength()
                + " 权限: " + sta.getPermission() + " 内容: ");
            FSDataInputStream fdis = fsSource.open(sta.getPath());
            byte[] data = new byte[1024];
            int read = -1;
            while ((read = fdis.read(data)) > 0) {
                ps.write(data, 0, read);
                fsdos.write(data, 0, read);
            }
            fdis.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();
    }
}
```

5.7.3 HDFS常用Java API及应用实例

四、编译运行程序

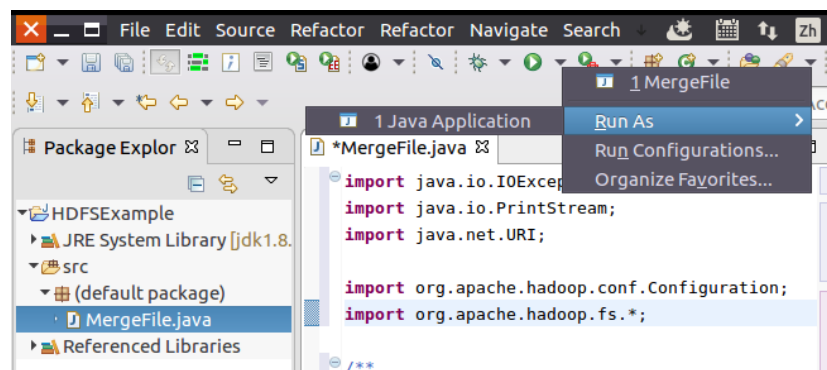
在开始编译运行程序之前，请一定确保Hadoop已经启动运行

然后，要确保HDFS的“/user/hadoop”目录下已经存在file1.txt、file2.txt、file5.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
file5.txt	this is file5.txt
file4.abc	this is file4.abc
file5.abc	this is file5.abc

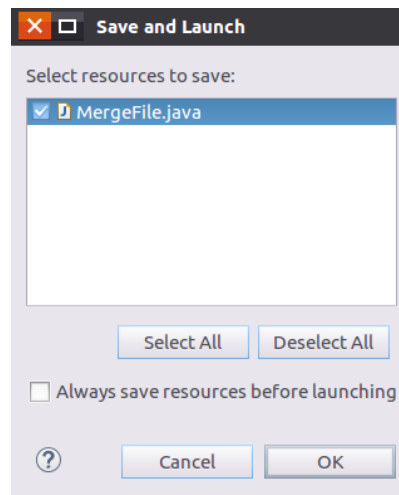
5.7.3 HDFS常用Java API及应用实例

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。



5.7.3 HDFS常用Java API及应用实例

然后，会弹出如下图所示界面。



5.7.3 HDFS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。



```

<terminated> MergeFile [Java Application] /usr/lib/jvm/jdk1.8.0_162/bin/java (Jan 27, 2020, 9:23:11 AM)
log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
路径: hdfs://localhost:9000/user/hadoop/file1.txt 文件大小: 18 权限: rw-r--r-- 内容: tf
路径: hdfs://localhost:9000/user/hadoop/file2.txt 文件大小: 18 权限: rw-r--r-- 内容: tf
路径: hdfs://localhost:9000/user/hadoop/file3.txt 文件大小: 18 权限: rw-r--r-- 内容: tf
  
```

5.7.3 HDFS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 file5.txt
```

5.7.3 HDFS常用Java API及应用实例

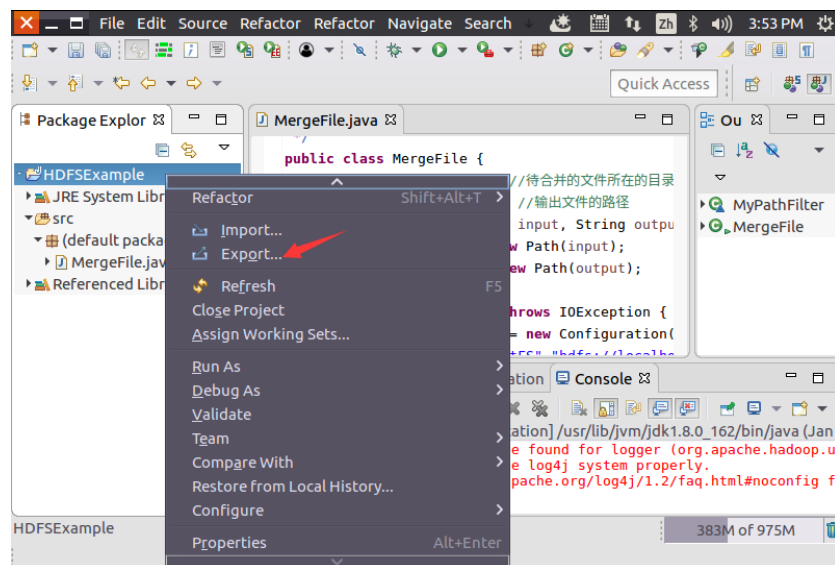
四、应用程序的部署

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
```

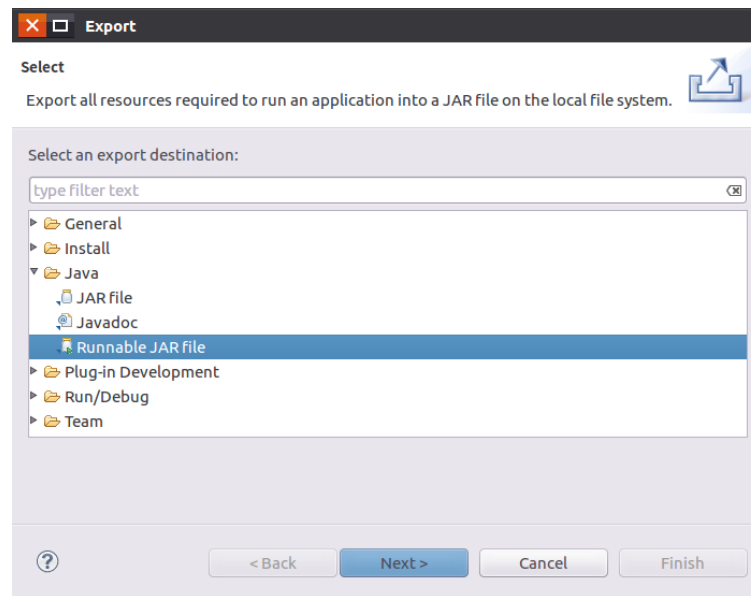
5.7.3 HDFS常用Java API及应用实例

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。



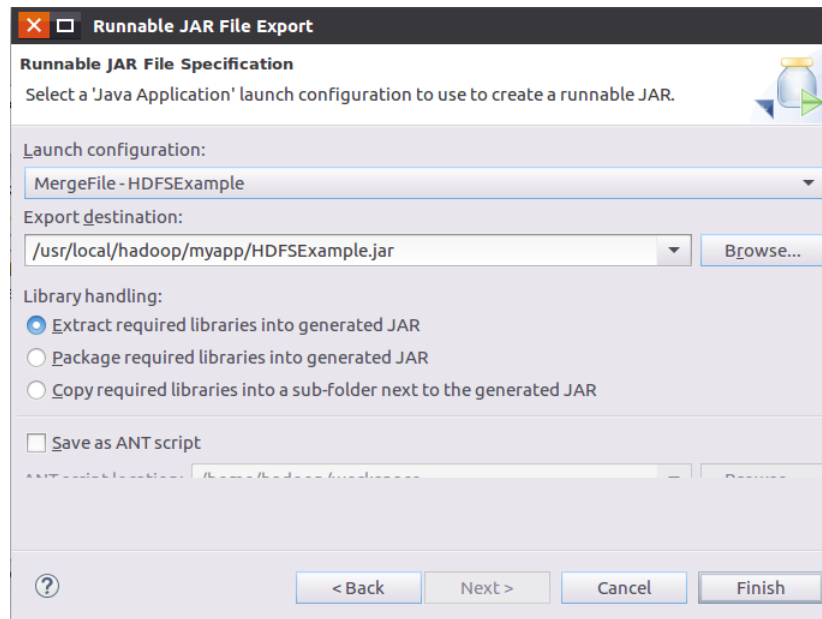
5.7.3 HDFS常用Java API及应用实例

然后，会弹出如下图所示界面。



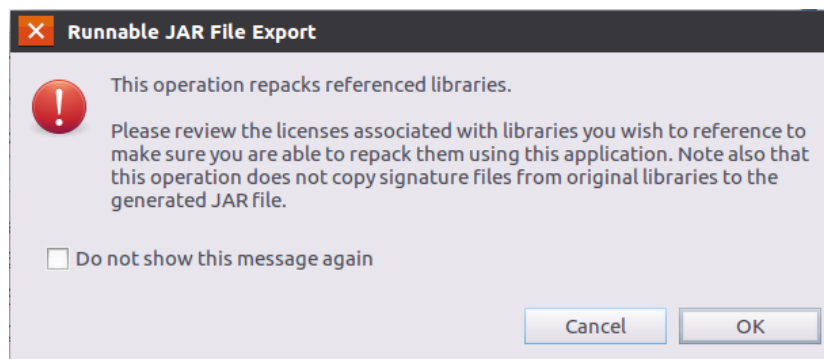
5.7.3 HDFS常用Java API及应用实例

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。



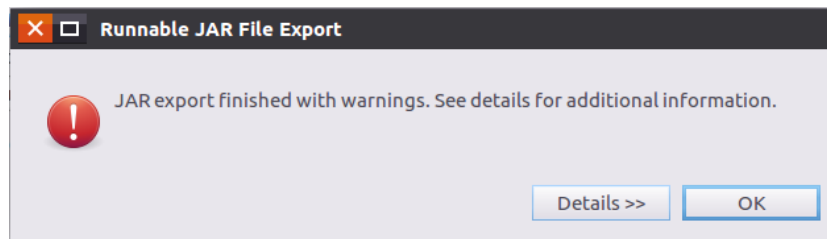
5.7.3 HDFS常用Java API及应用实例

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。



5.7.3 HDFS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。



5.7.3 HDFS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。

5.7.3 HDFS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
```

5.7.3 HDFS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 file5.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，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

感谢聆听