Hadoop

目录

**[1. Hadoop的发展 1](#_Toc25746)**

[1.1. 大数据的概念 1](#_Toc7151)

[1.2. Hadoop概述 4](#_Toc8172)

[1.3. Hadoop运行模式 7](#_Toc14693)

[vim /etrc/hosts，如图1-8。 9](#_Toc20890)

**[2. HDFS 17](#_Toc7248)**

[2.1. 分布式文件系统 17](#_Toc32416)

[2.2. HDFS简介 19](#_Toc23303)

[2.3. 基本指令 23](#_Toc7100)

[2.4. 安全模式 26](#_Toc30734)

[2.5. HDFS的数据结构 28](#_Toc26423)

[2.6. Hadoop插件 33](#_Toc12053)

[2.7. JavaAPI操作HDFS 42](#_Toc29276)

[2.8. HDFS的执行流程 52](#_Toc1809)

[2.9. HDFS的特点 55](#_Toc24517)

**[3. MapReduce的基本简介 57](#_Toc14818)**

[3.1. 分布式计算框架 57](#_Toc17616)

[3.2. MapReduce概述 58](#_Toc18020)

[3.3. MapReduce实现单词统计案例WordCount 62](#_Toc20478)

[3.4. 求平均值案例 80](#_Toc26595)

[3.5. MapReduce执行job过程 83](#_Toc8970)

**[4. MapReduce的序列化机制 84](#_Toc17987)**

[4.1. 序列化机制概述 84](#_Toc1785)

[4.2. 流量统计案例 86](#_Toc3171)

**[5. MR分区、排序、合并及shuffle过程 89](#_Toc4471)**

[5.1. 分区 89](#_Toc15990)

[5.2. 分区案例分析 92](#_Toc20020)

[5.3. 排序 98](#_Toc14455)

[5.4. 排序案例分析 99](#_Toc21542)

[5.5. 合并 103](#_Toc15582)

[5.6. Shuffle过程 107](#_Toc11737)

[5.7. MapReduce的主要特点 114](#_Toc32309)

**[6. Sqoop介绍 117](#_Toc29997)**

[6.1. 实现步骤 117](#_Toc8782)

[6.2. Sqoop基础指令（在Sqoop的bin目录下执行下列指令） 117](#_Toc17983)

# Hadoop的发展

## 大数据的概念

### 什么是大数据

大数据(big data,mega data)，或称巨量资料。比如TB、PB级别的数据。

“大数据”是需要新处理模式才能具有更强的决策力、洞察发现力和流程优化能力来适应海量、高增长率和多样化的信息资产。

### 大数据的特点

**1）Volume(大量)**

为了更准确地理解人们现在面临的数据量大小，再来看一组公式：

1024GB=1TB

1024TB=1PB

1024PB=1EB

1024 EB=1ZB

在电子商务平台eBay上，每天新增的数据量达到50TB，1年累计的数据量即达到18PB。与之相对地，根据IDC的研究报告，自人类开始记录历史以来，到2006年为止全人类全部的印刷书本文字加起来大约50PB。也就是说，仅eBay平台3年的新增数据，就超过了全人类全部书本的数据量。同时，在社交网站Facebook的计算机集群的磁盘空间中，目前已存储了超过100PB的数据，也就是说，仅Facebook一个网站存储的数据，就已经是人类书本数据量的2倍之多。

与海量的数据同时存在的还有越来越快的数据增长速度。根据IDC的统计，全球每年产生的数据达到将近8ZB。

**2）Velocity(高速)，持续的高速增长**

以1分钟为单位，看看在爆炸的数据世界中发生了什么。

（1）搜索：全球最大的搜索引擎Google处理了200万次搜索请求。

（2）音频：在Pandora音乐网站上，播放的音乐时长超过61 000小时。

（3）视频：YouTube的用户上传了总计时长48小时的视频，130万个视频被观看。

（4）应用：Apple的应用商店完成了4.7万次应用下载。

（5）电子商务：淘宝上产生了7万次页面访问，新增了35GB的数据。

**3）Variety(多样) ，网络日志、视频、图片、地理位置信息**

**4）Value(价值密度低)，以视频为例，连续不间断监控过程中，可能有用的数据仅仅有一两秒**

**5）Veracity（真实性）**

### 大数据的价值

第一、计算机科学在大数据出现之前，非常依赖模型以及算法。人们如果想要得到精准的结论，需要建立模型来描述问题，同时，需要理顺逻辑，理解因果，设计精妙的算法来得出接近现实的结论。因此，一个问题，能否得到最好的解决，取决于建模是否合理，各种算法的比拼成为决定成败的关键。然而，大数据的出现彻底改变了人们对于建模和算法的依赖。举例来说，假设解决某一问题有算法A 和算法B。在小量数据中运行时，算法A的结果明显优于算法B。也就是说，就算法本身而言，算法A能够带来更好的结果；然而，人们发现，当数据量不断增大时，算法B在大量数据中运行的结果优于算法A在小量数据中运行的结果。这一发现给计算机学科及计算机衍生学科都带来了里程碑式的启示：当数据越来越大时，数据本身（而不是研究数据所使用的算法和模型）保证了数据分析结果的有效性。即便缺乏精准的算法，只要拥有足够多的数据，也能得到接近事实的结论。数据因此而被誉为新的生产力。

当数据达到一定程度时，数据本身就可以说话了。

第二、当数据足够多的时候，不需要了解具体的因果关系就能够得出结论。

例如，Google 在帮助用户翻译时，并不是设定各种语法和翻译规则。而是利用Google数据库中收集的所有用户的用词习惯进行比较推荐。Google检查所有用户的写作习惯，将最常用、出现频率最高的翻译方式推荐给用户。在这一过程中，计算机可以并不了解问题的逻辑，但是当用户行为的记录数据越来越多时，计算机就可以在不了解问题逻辑的情况之下，提供最为可靠的结果。可见，海量数据和处理这些数据的分析工具，为理解世界提供了一条完整的新途径。

第三、由于能够处理多种数据结构，大数据能够在最大程度上利用互联网上记录的人类行为数据进行分析。大数据出现之前，计算机所能够处理的数据都需要前期进行结构化处理，并记录在相应的数据库中。但大数据技术对于数据的结构的要求大大降低，互联网上人们留下的社交信息、地理位置信息、行为习惯信息、偏好信息等各种维度的信息都可以实时处理，立体完整地勾勒出每一个个体的各种特征。

第四、从政府或社会角度，大数据时代到来，会催生很多新的就业岗位，所以大数据是一种新的生成力。

## Hadoop概述



官网地址：<http://hadoop.apache.org/>

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

在hadoop的官网有一句话是这样的说的，Hadoop是Apache中的顶级项目，他是一个开源的、可靠的、可扩展的系统架构，可利用分布式架构来存储海量的数据，以及实现分布式的计算。

### Hadoop的演变

Hadoop的创始人：Doug Cutting 与 Mike Cafarella，如图1-1所示：



图1-1 左起Doug Cutting、Mike Cafarella

Doug Cutting1985年毕业于美国斯坦福大学，他是 Lucene的创始人，同时也是Nutch和Hadoop的联合创立者，先是参与了Lucene的开发，然后是Nutch，然后是Hadoop。

Hadoop起源于Apache Nutch，Nutch是一个开源的java语言来实现的搜索引擎。

2004年，Cutting和同为程序员出身的Mike Cafarella决定开发一款可以代替当时的主流搜索产品的开源搜索引擎，这个项目被命名为Nutch。它是模仿google搜索引擎创立的开源搜索引擎，后归于Apache旗下。

Nutch主要完成抓取，提取内容等工作。和Solr一样，都是搜索引擎，并且都是基于Lucene的。支持全文搜索和爬虫。

但是遇到了一个难题是：Apache开源项目Nutch搜索引擎的开发者Doug Cutting等人正面临着如何将其架构扩展到可以处理数10亿规模网页的难题。

幸运的是，早在2003年，Google发表的第一篇关于其云计算核心技术论文《Google File System》简称为GFS，Cutting在了解了GFS系统后，他们敏锐地意识到，这样的技术架构可以帮助他们解决存储Nutch抓取网页和建立索引过程中产生的大量文件的问题，并提高管理这些存储节点的效率。因此在参考GFS技术的基础上，他们在2004年编写了一个开放源码的类似系统——NDFS（Nutch Distributed File System，Nutch分布式文件系统）。

在2004年，Google又公开发表了阐述其另一核心技术MapReduce的论文，让业界第一次真切感受到了MapReduce编程模型在解决大型分布式并行计算问题上的巨大威力和实用性。很快Nutch团队就将MapReduce技术应用于他们的项目，在2005年将Nutch的主要算法都移植到基于MapReduce和NDFS的框架下运行。

在完成了MapReduce和NDFS的开源实现后，Cutting等人为Nutch搭建了一个包含20个计算节点的平台，验证了这两个开源组件在解决搜索数百万网页问题情况下的有效性。但是在面对将这两项技术拓展到可以面对数10亿级网页的工作时，他们面临了巨大的资源压力。幸运的是，雅虎公司也发现了这两项技术的巨大潜力，将Nutch的开发者之一Doug Cutting招入公司，并建立了一个专门的团队提供支持，当时有一个100多人的团队共同来完善Hadoop，再后来，Yahoo把Hadoop贡献给Apache，成为Apache的的一个单独子项目Hadoop。

即Hadoop刚开始的诞生，最初主要是为了解决nutch搜索海量数据的分布式存储和管理问题。在nutch 0.8版本之前是nutch的子项目。

在0.8版本以后，独立出来成单独的项目，同时将NDFS改名为HDFS。

hadoop名字的来源：这个名字不是一个缩写，它是一个虚构的名字。该项目的创建者，Doug Cutting如此解释Hadoop的得名："这个名字是我孩子给一头吃饱了的棕黄色大象命名的。我的命名标准就是简短，容易发音和拼写，没有太多的意义，并且不会被用于别处。小孩子是这方面的高手。

### Hadoop的作用

1.存储海量数量

2.计算海量数据

Hadoop允许使用简单的编程模型在计算机集群中对大型数据集进行分布式处理。可以从单个服务器扩展到数千台机器，每个机器都提供本地计算和存储，而不是依靠硬件来提供高可用性。

此外，Hadoop集群的高可用性也非常良好，因为框架内的机制是可以够自动检测和处理故障。

### Hadoop的版本说明

Hadoop发展到现在已经经历了3个版本的更新迭代

第一代Hadoop和第二代Hadoop

第一代Hadoop称之为Hadoop1.0版本，分别是0.20.x,0.21.x和0.22.x，包含HDFS和MapReduce

第二代Hadoop称之为Hadoop2.0版本，分别是0.23.x和2.x，包含HDFS、MapReduce以及Yarn

注意：2.0版本完全不同于Hadoop 1.0，是一套全新的架构，加入了Yarn资源协调管理框架。

Yarn是一种新的 Hadoop 资源管理器，它是一个通用资源管理系统，可为上层应用提供统一的资源管理和调度，它的引入为集群在利用率、资源统一管理和数据共享等方面带来了巨大好处。

可以实现对资源的细粒度的封装（如CPU、内存、带宽），同时还可以协调多种不同的计算框架（如MapReduce、Spark）

Hadoop1.0和Hadoop2.0简要架构图，如图1-2所示：

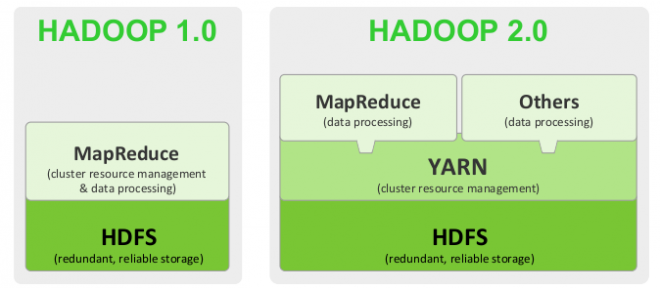


图1-2

第三代Hadoop称之为Hadoop3.0版本，在2017年9月发布，包含HDFS、Mapreduce、Yarn以及Ozone（臭氧）

Ozone是基于Apache Hadoop之上用来做亿万数量级对象存储的，可以针对不适合存储在HDFS上的小文件进行优化

但是现在在企业中为了项目的稳定性，主要是以Hadoop2.0版本为主

## Hadoop运行模式

### Hadoop的搭建

1.Hadoop单机模式的搭建：只支持Hdfs，不支持MapReduce

2.Hadoop伪分布式的搭建：在单台机器上利用多线程模拟多台机器的操作流程，既支持Hdfs，同时也支持MapReduce

3.Hadoop完全分布式的搭建：在多台机器上搭建Hadoop的集群模式，支持Hdfs和MapReduce

### Hadoop伪分布式的搭建

1.首先搭建Linux开发环境

2.启动Linux虚拟机，关闭防火墙

a.临时关闭防火墙 service iptables stop，如图1-3.

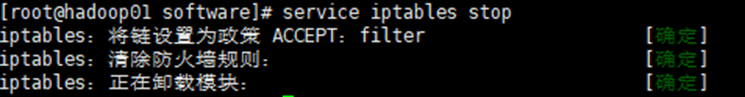


图1-3

b.永久关闭防火墙 chkconfig iptables off

3.修改虚拟机的主机名

a.查看主机名 hostname，如图1-4。

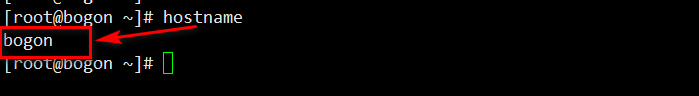


图1-4

b.修改主机名 vim /etc/sysconfig/network，如图1-5。

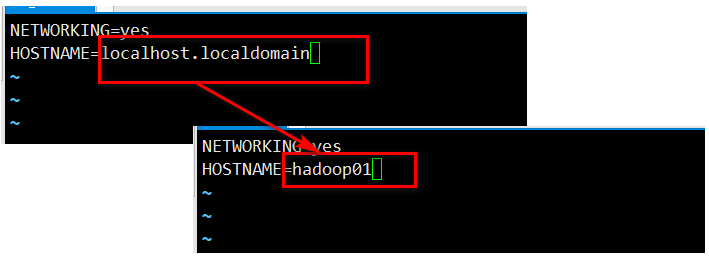


图1-5

4.将修改后的主机名生效

a.临时生效 hostname hadoop01，但是此时命令提示符处并未发生变化，如图1-6。



图1-6

b.永久生效（重启linux之后即可），此时命令提示符处的主机名已发生的变化，如图1-7。

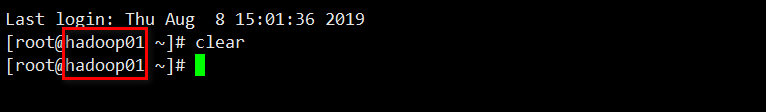


图1-7

5.修改hosts文件，做虚拟机的ip地址与虚拟机主机名的映射关系

vim /etrc/hosts，如图1-8。

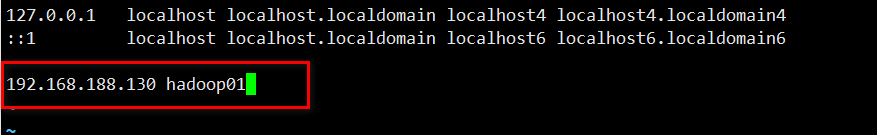


图1-8

6.设置ssh免密登录，分为2步骤执行

a.执行ssh-keygen(执行此步，一直回车即可)，如图1-9。

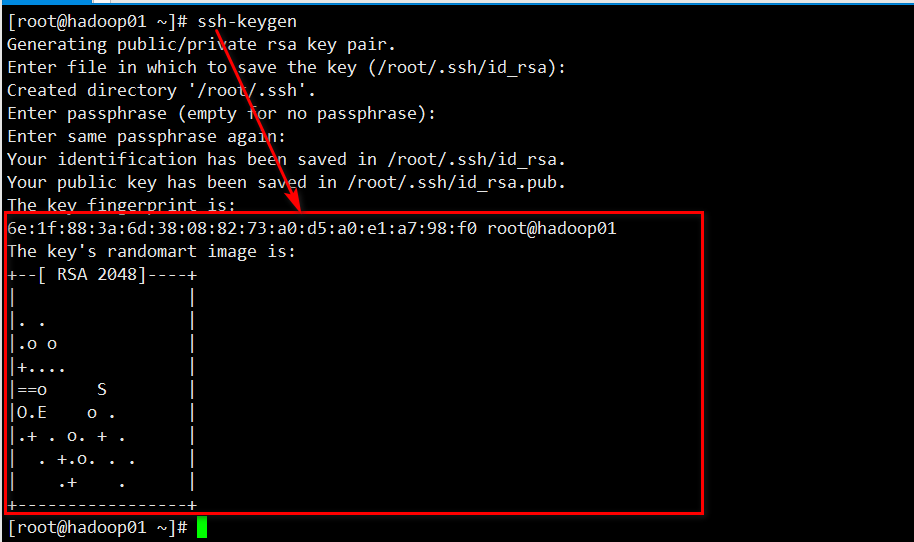


图1-9

b.执行ssh-copy-id root@hadoop01（此过程需要输入虚拟机的登录密码），如图1-10。

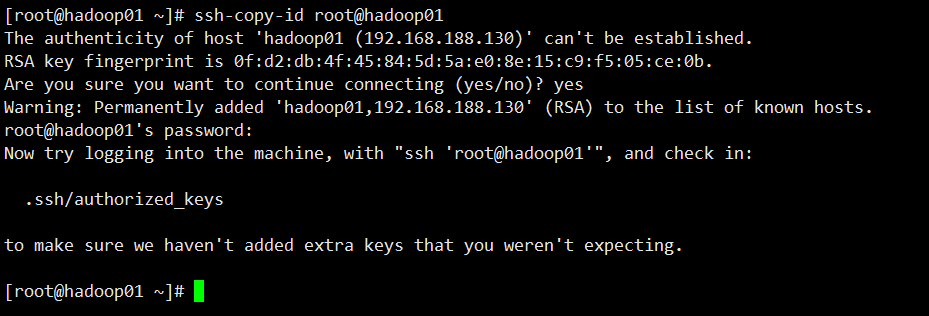


图1-10

7.利用xftp将hadoop与JDK安装包上传到在home目录下创建的software中，如图1-11、图1-12。

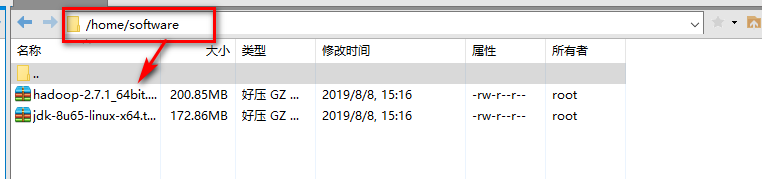


图1-11

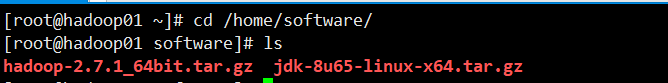


图1-12

8.解压jdk压缩包，并改名操作

a.解压 tar -xvf jdk-8u65-linux-x64.tar.gz，如图1-13。

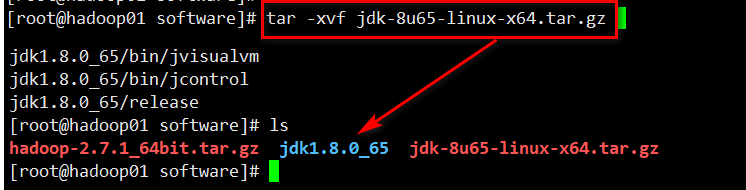


图1-13

b.将解压后的jdk改名为jdk8 执行指令mv jdk1.8.0\_65 jdk8，如图1-14。

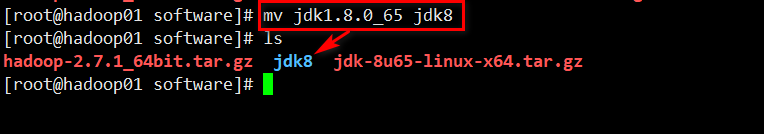


图1-14

9.在linux上配置jdk的环境变量，编辑指令 vim /etc/profile，如图1-15。

JAVA\_HOME=/home/software/jdk8

CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

PATH=$JAVA\_HOME/bin:$PATH

export JAVA\_HOME CLASSPATH PATH

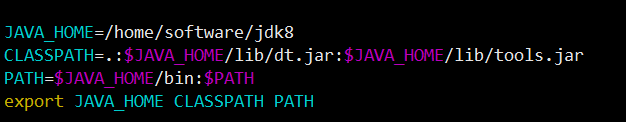


图1-15

10.配置完成之后，需要当前环境变量立即生效并查看jdk的版本号是否生效，如图1-16

执行指令 source /etc/profile java -version



图1-16

11.解压hadoop压缩包，如图1-17。

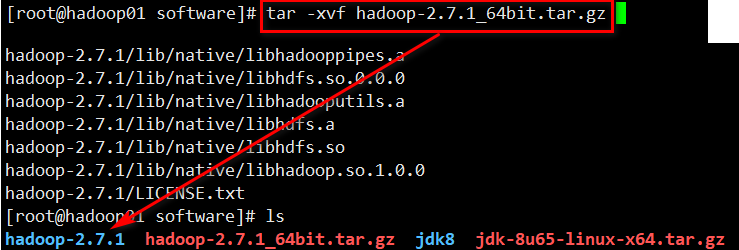


图1-17

12.编辑 vim hadoop-2.7.1/etc/hadoop/hadoop-env.sh，如图1-18。

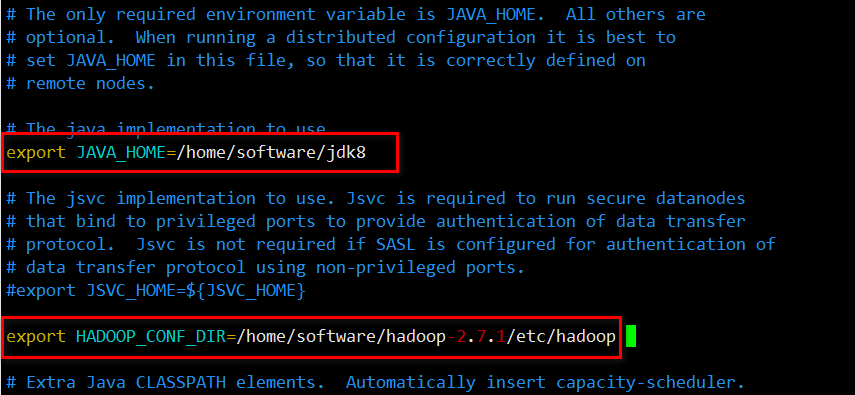


图1-18

13.编辑vim core-site.xml文件，如图1-19。

<configuration>  
<!--用来指定hdfs的老大，namenode的地址-->  
<property>  
<name>fs.defaultFS</name>  
<value>hdfs://hadoop01:9000</value>  
</property>  
<!--用来指定hadoop运行时产生文件的存放目录-->  
<property>  
<name>hadoop.tmp.dir</name>  
<value>/home/software/hadoop/tmp</value>  
</property>  
</configuration>



图1-19

14.编辑hdfs-site.xml文件，如图1-20。

<configuration>  
<!--指定hdfs保存数据副本的数量，包括自己，默认值是3-->  
<!--如果是伪分布模式，此值是1-->  
<property>  
<name>dfs.replication</name>  
<value>1</value>  
</property>  
<!--设置hdfs的操作权限，false表示任何用户都可以在hdfs上操作文件-->  
<property>  
<name>dfs.permissions</name>  
<value>false</value>  
</property>  
</configuration>

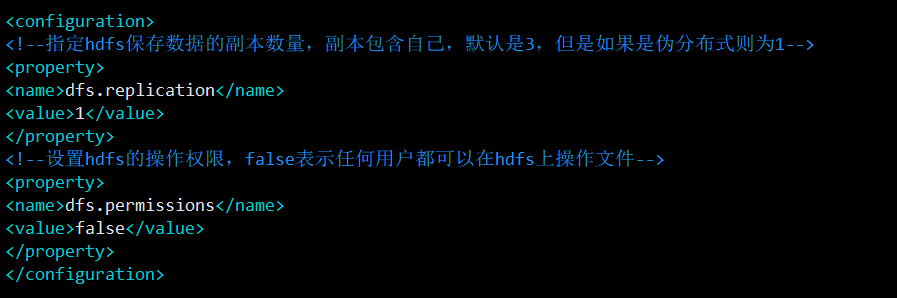


图1-20

15.拷贝模板文件mapred-site.xml.template并重命名为mapred-site.xml  
执行：cp mapred-site.xml.template mapred-site.xml后编辑，如图1-21。

<configuration>  
<property>  
<!--指定mapreduce运行在yarn上-->  
<name>mapreduce.framework.name</name>  
<value>yarn</value>  
</property>  
</configuration>

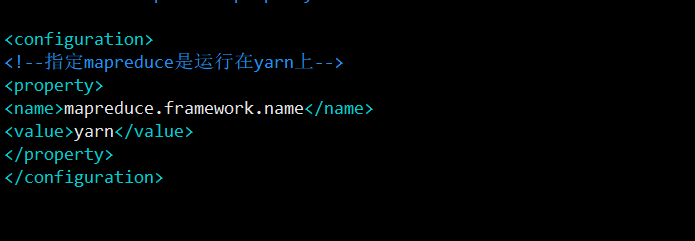


图1-21

16.编辑文件yarn-site.xml，如图1-22。

<configuration>  
<!-- Site specific YARN configuration properties -->  
<property>  
<!--指定yarn的老大 resoucemanager的地址-->  
<name>yarn.resourcemanager.hostname</name>  
<value>hadoop01</value>  
</property>  
<property>  
<!--NodeManager获取数据的方式-->  
<name>yarn.nodemanager.aux-services</name>  
<value>mapreduce\_shuffle</value>  
</property>  
</configuration>



图1-22

17.修改slaves文件，将里面的localhost修改为虚拟机的主机名，如图1-23。



图1-23

18.配置hadoop的环境变量，后需要立即生效，执行source /etc/profile，如图1-24。

JAVA\_HOME=/home/software/jdk8

HADOOP\_HOME=/home/software/hadoop-2.7.1

CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

PATH=$JAVA\_HOME/bin:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin:$PATH

export JAVA\_HOME CLASSPATH PATH HADOOP\_HOME

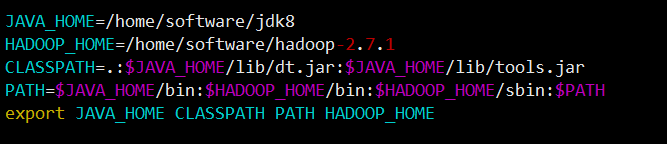


图1-24

19.在使用hadoop之前需要先对namenode执行格式化操作，如图1-25。

执行：hadoop namenode -format

如果不好使，可以重启linux  
当出现：successfully，证明格式化成功

计算机生成了可选文字:
16 / 39 / 31 39 ： 42 ： 27 INFO common.Storage: Storage directory /home/software/hadoop-2.7.1 
/tmp/dfs/name has been successfully formatted. 
16 / 39 / 31 39 ： 42 ： 27 INFO namenode.NNStorageRetentionManager: Going to retain 1 images 

图1-25

20.执行start-all.sh,启动hadoop，如图1-26。

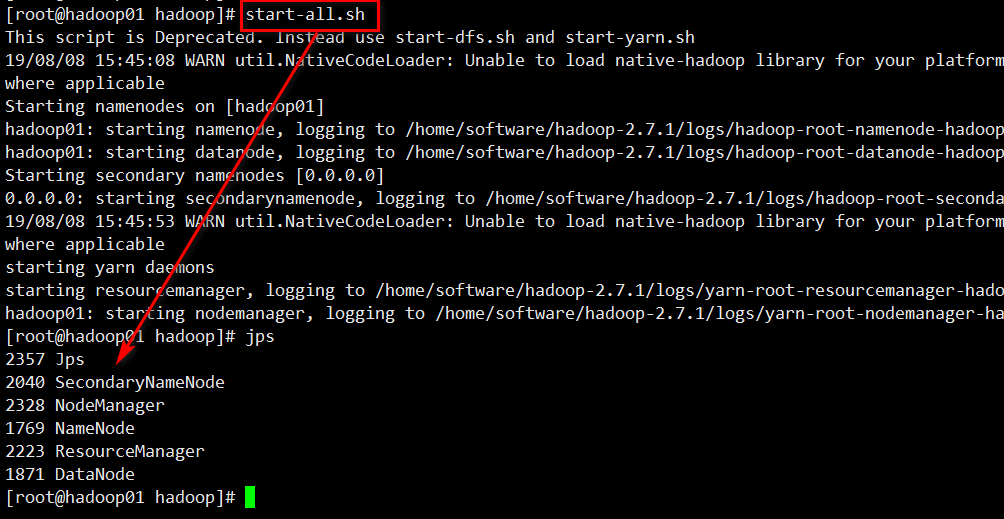


图1-26

21.浏览器访问

使用虚拟机的ip地址：50070或者虚拟机的主机名：50070来进行访问，如图1-28。

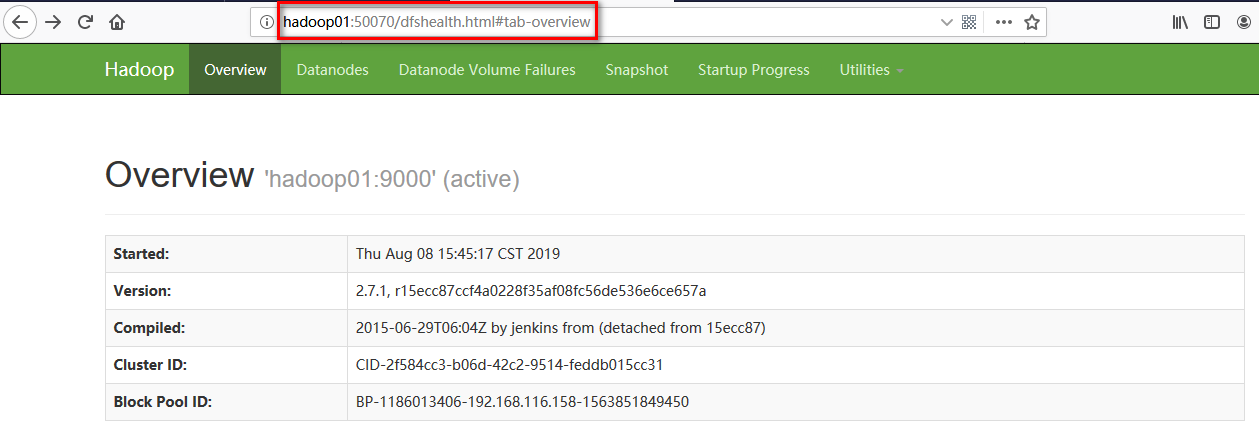


图1-28

# HDFS

## 分布式文件系统

Hadoop是一个由多主机集群共同对外服务的系统，它有两个基础框架，其中一个是分布式文件系统框架。分布式文件系统有不少优势和特点。比如，相对传统本地文件系统的优势，还有其自身的特点。

本地文件系统如Linux操作系统的ext3、ext4，Windows的NTFS等文件系统，它们管理本地的磁盘存储资源、提供文件到存储位置的映射，并抽象出一套文件访问接口供用户使用。也就是说这些常见的文件系统，通常是给单台主机使用的。这类文件系统虽然通过一些高可用框架可以提供，某种程度上的分布式存储。

但随着互联网企业的高速发展，这些企业对数据存储的要求越来越高，而且模式各异。如电商主站的大量商品图片，其特点是文件较小，但数量巨大；而类似于youtube，优酷这样的视频服务网站，其后台存储着大量的视频文件，尺寸大多在数十兆到数吉字节不等。这些应用场景都是ext3、NTFS传统文件系统不能充分解决的。

分布式文件系统在新的需求下具有优势，它能使用更多节点，而不是单一节点的存储方案，就成为了一个可代替的选择。分布式文件系统将数据存储在物理上分散的多个存储节点上，对这些节点的资源进行统一的管理与分配，并向用户提供文件系统访问接口。其主要解决了单一节点的本地文件系统在文件大小、文件数量、打开文件数等的限制问题。

目前比较主流的一种分布式文件系统架构，如图2-1所示，通常包括主控服务器（或称元数据服务器、名字服务器等，通常会配置备用主控服务器以便在故障时接管服务），多个数据服务器（或称存储服务器，存储节点等），以及多个客户端，客户端可以是各种应用服务器，也可以是终端用户。

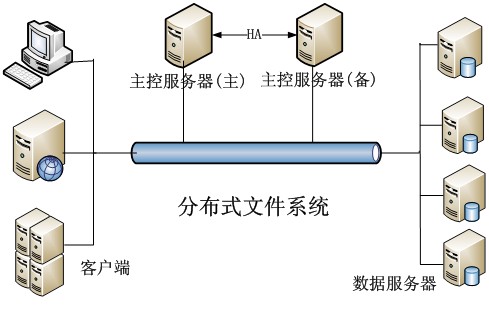
[](http://blog.chinaunix.net/attachment/201202/18/20196318_1329546508RJC2.jpg)

图2-1

分布式文件系统的大数据存储解决策略，归根结底是将大问题分解为小问题。大量的文件，均匀分布到多个数据服务器上后，每个数据服务器存储的文件数量就少了。这种一定程度上解决了，单个服务器有文件个数上限的问题。另外通过使用将多个小文件（1KB、1MB等）合并成一个大文件的方式，总能把单个数据服务器上存储的文件数，降到单机能解决的规模；对于很大的文件（TB、PB），将大文件划分成多个相对较小的片段，存储在多个数据服务器上。对解决单个服务器，存储容量受限，也很有帮助。

理论上，分布式文件系统可以只有客户端和多个数据服务器组成，客户端根据文件名决定将文件存储到哪个数据服务器，但是一旦有数据服务器失效时，问题就变得复杂。客户端并不知道数据服务器宕机的消息，仍然连接数据服务器进行数据读写，导致整个系统的可靠性极大的降低。而且完全由客户端决定数据分配时非常不灵活的，其不能根据文件特性制定不同的分布策略。

于是，迫切的需要能知道各个数据服务器的服务状态。数据服务器的状态管理，可分为分散式和集中式两种方式。

分散式状态管理是让多个数据服务器相互管理，即每个服务器向其他所有的服务器发送心跳信息，但这种方式开销较大。例如一个包含10000台主机的集群，如果一个节点向它自己和其他所有节点，都发送一条容量为1MB的信息，集群公共网络瞬间就会产生10GB的带宽需求。这种情况下控制不好，很容易影响到正常的数据服务，而且工程实现较为复杂。

集中式状态管理是指通过一个独立的服务器（如上图中的主控服务器）来管理数据服务器，每个服务器向其汇报服务状态来达到集中管理的目的，这种方式简单易实现，目前很多分布式文件系统都采用这种方式如GFS、HDFS、TFS、MooseFS。主控服务器在负载较大时会出现单点故障，较多的解决方案是配置备用服务器，以便在故障时接管服务。如果需要，主备之间要进行数据的同步。

其中HDFS就是Hadoop采取的一种分布式文件系统，用来完成大数据的存储工作。

## HDFS简介

HDFS是Hadoop的核心概念之一，也是Hadoop的重点内容。HDFS架构图描述了Hadoop分布式存储框架的系统结构，如图2-2所示，能够帮助理解HDFS是如何实现功能、以及存储架构如何运行。

计算机生成了可选文字:
HDFS Arch itecture 
Metadata 0 
lien 
Re d Datanodes 
Rack 1 
Namenode 
Metadata (Name, replicas 
/hom e/foo/data 3 
引 OC OPS 
0 
Replication 
Write 
lien 
Datanodes 
B IOC s 
Rack 2 

图2-2

图中部分名词解释如下：

HDFS Architecture：HDFS 架构

Client：客户端

NameNode：管理者节点

DataNode：工作者节点

Metadata(Name,replicas)：元数据（名称，副本）

Metadata ops：元数据操作

Replication：复制

Block ops：块操作

HDFS为了保证数据存储的可靠性和读取性能，对数据进行切块后进行复制并存储在集群的多个节点中。

HDFS中存在一个名字节点NameNode和多个数据节点DataNode。

NameNode的主要功能：存储元数据信息，将元数据保存在内存或者磁盘中以及保存文件、block、DataNode之间的映射关系。

DataNode的主要功能：存储block内容，将内容存储在磁盘中，维护了block id到文件的映射关系。

HDFS采用Master/Slave架构。一个HDFS集群是由一个NameNode和一定数目的DataNode组成。NameNode是一个中心服务器，负责管理文件系统的名字空间(namespace)以及客户端对文件的访问。它是HDFS对外服务的首要节点，客户端要连接到NameNode之后，才能获取文件系统的元数据，并且进一步知道怎样访问DataNode并获取实际数据。

集群中的各个DataNode一般是一个独立的物理主机服务器，服务器上的“DataNode”进程负责管理它所在节点上的存储。HDFS暴露了文件系统的名字空间，也就是NameNode，让用户能够通过NameNode以文件的形式在DataNode集群上面存储数据。从内部看，一个文件其实被分成一个或多个数据块，这些块存储在一组DataNode上。NameNode执行文件系统的名字空间操作，比如打开、关闭、重命名文件或目录。它也负责确定数据块到具体DataNode节点的映射。DataNode负责处理文件系统客户端的读写请求。在NameNode的统一调度下进行数据块的创建、删除和复制。

DataNode数据节点，在NameNode调度下完成数据存储任务的同时，为了防止DataNode宕机造成数据丢失，对于文件块要有备份。策略是一个文件块准备三个在不同位置的副本。并且基于机架感知技术，使它们能够防止单节点故障、机架故障。实现存储上的数据安全性。机架感知技术就是告诉 Hadoop， 集群中哪台机器属于哪个机架的一种网络技术。

NameNode和DataNode被设计成可以在普通的商用机器上运行。这些机器一般运行着GNU/Linux操作系统(OS)。HDFS采用Java语言开发，因此任何支持Java的机器都可以部署NameNode或DataNode。由于采用了可移植性极强的Java语言，使得HDFS可以部署到多种类型的机器上。一个典型的部署场景是一台机器上运行一个NameNode实例，而集群中的其他机器分别运行一个DataNode实例。

集群中单一NameNode的结构大大简化了系统的架构。NameNode是所有HDFS元数据的仲裁者和管理者，这样，用户数据永远不会流过NameNode。

NameNode负责维护文件系统的名字空间，任何对文件系统名字空间或属性的修改都将被NameNode记录下来。应用程序可以设置HDFS保存的文件的副本数目。文件副本的数目称为文件的副本系数，这个信息也是由NameNode保存的。NameNode也影响了写文件时的数据块大小，Hadoop 1.0时，数据块大小时64MB；Hadoop 2.0时数据块大小调整为128MB。也可以通过修改参数：

<name>dfs.block.size</name>

确定一个自己认为合适的数据块容量。

上面段落提到数据块块大小的问题。从大数据处理角度来看，块越大越好。所以从技术的发展，以后的块会越来越大，因为块大，会减少磁盘寻址次数，从而减少寻址时间。

HDFS被设计成能够在，一个大集群中跨机器可靠地存储超大文件。它将每个文件存储成一系列的数据块，除了最后一个，所有的数据块都是同样大小的。NameNode全权管理数据块的复制，它周期性地从集群中的每个DataNode接收心跳信号和块状态报告。接收到心跳信号意味着该DataNode节点工作正常。块状态报告包含了一个该DataNode上所有数据块的列表。NameNode根据这个列表来判断，此DataNode节点上存储数据的状态。

在HDFS的架构中NameNode节点只存储元数据信息，并不存储数据本身。NameNode对于元数据信息的管理方式是，放在内存一份，提供给客户或者任务进程快速访问查询；同时，也会通过fsimage和edits文件，将元数据信息持久化到磁盘上。

## 基本指令

完成Hadoop软件安装之后，就可以通过各种控制命令，来管理和应用HDFS分布式文件系统了。下面就是一些常见命令及方法的介绍。

1.执行如下命令：

hadoop fs -mkdir /park01

在HDFS 的根目录下，创建 park01目录，如图2-3所示。

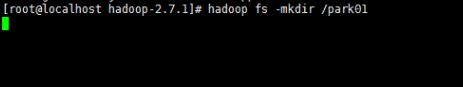


图2-3

2.执行如下命令：

hadoop fs –ls /

查看HDFS根目录下有哪些目录，如图2-4所示。

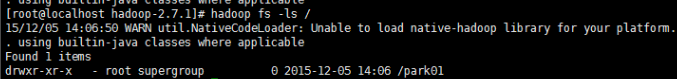


图2-4

3.执行如下命令：

hadoop fs –put /root/1.txt /park

将Linux操作系统root目录下的1.txt放在HDFS的park目录下，如图2-5所示。

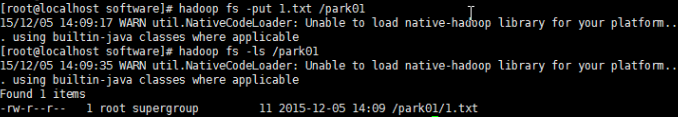


图2-5

4.执行如下命令：

hadoop fs –get /park01/1.txt ./

把HDFS文件系统下park01目录的文件下载到linux的当前目录下，如图2-6所示。

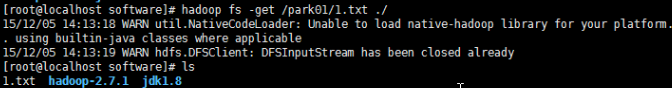


图2-6

5.执行如下命令：

hadoop fs -rm /park/文件名

删除HDFS 的park目录的指定文件，如图2-7所示。

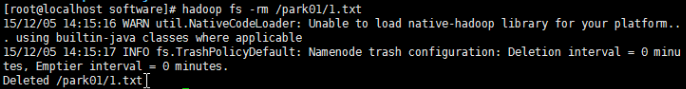


图2-7

6.执行如下命令：

hadoop fs -rmdir /park02

删除park02目录，但是前提目录里没有文件，如图2-8所示。

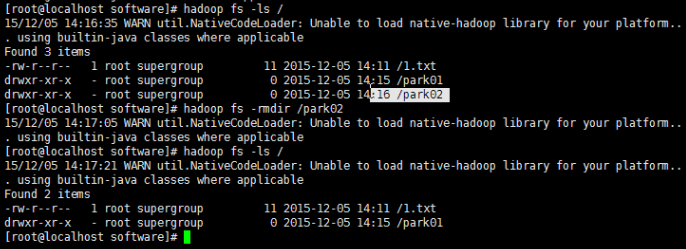


图2-8

7.执行如下命令：

hadoop fs -rmr /park01

删除park01目录，即使目录里有文件，如图2-9所示。



图2-9

8.执行如下命令：

hadoop fs -cat /park01/1.txt

查看park目录下的a.txt文件，如图2-10所示。



图2-10

9.执行如下命令：

hadoop fs -tail /park/1.txt

查看park目录下a.txt文件末尾的数据，如图2-11所示。



图2-11

10.执行如下命令：

hadoop jar xxx.jar

执行jar包，如图2-12所示。



图2-12

11.执行如下命令：

hadoop fs -cat /park/result/part-r-00000

查看 /park/result/part-r-00000文件的内容，如图2-13所示。



图2-13

12.执行如下命令：

hadoop fs –mv /park02 /park03

将HDFS上的park02目录重名为park03命令，如图2-14所示。

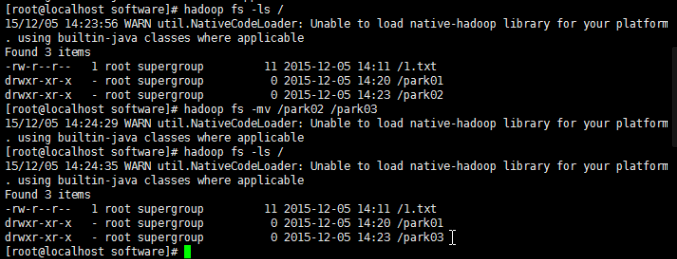


图2-14

13.执行如下命令：

hadoop fs -mv /1.txt /park03

将根目录下的1.txt移动到/park03目录下，如图2-15所示。

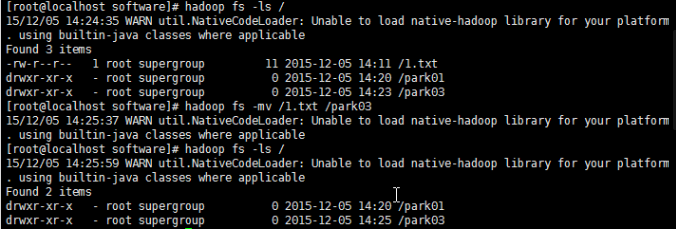


图2-15

14.执行如下命令：

hadoop fs –touchz /park/3.txt

创建一个空文件，如图2-16所示。



图2-16

## 安全模式

Hadoop的安全模式，是集群系统启动过程中需要经历的一个过程。在重新启动HDFS后，NameNode活动起来会立即进入安全模式。此时不能操作HDFS中的文件，只能查看目录文件名等，读写操作都不能进行。NameNode启动时，需要载入fsimage文件到内存，同时执行edits文件中各项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（这个步骤不需要SNN的参与）和一个空的编辑文件。

此时NameNode文件系统对于客户端来说是只读的。在此阶段NameNode收集各个DataNode的报告，当数据块达到最小复本数以上时，会被认为是“安全”的，在一定比例的数据块被确定为安全后，再经过若干时间，安全模式结束。当检测到副本数不足的数据块时，该块会被复制直到到达最小副本数，系统中数据块的位置并不是NameNode维护的，而是以块列表的形式存储在DataNode中。

在需要时，可以通过手动开启/关闭安全模式可执行下面的命令进入/离开安全模式，如图2-17、图2-18所示。

执行命令如下，进入安全模式：

hadoop dfsadmin –safemode enter

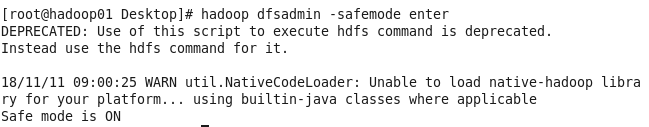


图2-17

执行命令如下，离开安全模式：

hadoop dfsadmin -safemode leave

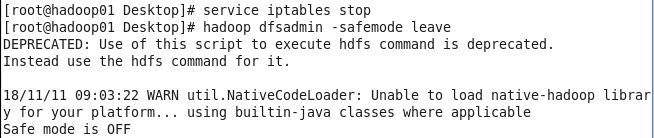


图2-18

注意，这方面内容了解即可，不建议去手动进入或离开安全模式。满足block副本数量和DataNode节点数量条件，并且能够维持一定时间之后，Hadoop会自动离开安全模式。 这个配置主要是对集群的稳定程度做进一步的确认，最好让Hadoop自动离开安全模式。

## HDFS的数据结构

### HDFS的数据块（Block）

HDFS数据结构中一个重要的概念就是数据块（Block）。计算机使用一个存储设备，通常需要建立一个文件系统，文件系统的结构帮助计算机识别存储空间、安排数据存储。计算机难以使用一个庞大的裸存储，就像我们刚刚学习写字的时候，难以直接使用一张庞大的白纸写字一样。那时候我们需要将较大的一张白纸裁成16开或32开的小张纸，然后在小张纸上画出田字格、米字格，然后在田字格、米字格里写汉字。以此来完成文字的书写。文件系统也是这样，它也需要将庞大的裸存储裁成小存储结构，这个小存储结构就叫数据块。然后文件系统在数据块中再划分出细节结构，就可以写入数据了。

Windows操作系统的文件系统NTFS数据块的大小是4096B；Linux操作系统的文件系统ext3数据块大小也是4096B；常用的数据库MySQL、Oracle数据块大小默认是8192B。而HDFS的块大小，在Hadoop 1.x版本时默认为64MB。在Hadoop 2.X时默认为128MB。

当数据块的结构越大，那么对同样大小的文件来说，需要使用的数据块数量越少，生成的元数据也就越少。更少的元数据也意味着使用更少的内存。这就是HDFS为什么能完成超大规模数据集存储的另一个原因：数据块更大、使用元数据更少、同样多的数据需要的内存空间更少。当然这不是没有代价的，HDFS对小文件的支持很差，每个大小小于128MB的小文件存储到HDFS都是在浪费这个分布式存储的资源。并且让整个系统处于低效之中。

### HDFS的元数据

NameNode是用来维护元数据信息的，其需要管理元数据信息、文件名、文件大小、文件块信息等。NameNode为了快速查询，把元数据信息存到内存里。要注意的是，元数据只有存储在内存中，才能足够快的帮助客户读写文件系统。如果元数据大量存储在磁盘上，就有可能造成读写数据时，要先到磁盘查找元数据，再将元数据读取到内存，才能完成客户的数据读写请求。此外为了应对服务宕机而引起的元数据丢失，也要持久化到本地文件里。NameNode不存储具体块数据，只存储元数据信息。DataNode用于负责存储块数据，Secondarynamenode是用来协助NameNdoe进行元数据合并的。

### Fsimage文件和edits文件

NameNode主机中的fsimage 文件，也是记录元数据信息的文件。与fsimage文件在同一个目录中的edits文件，是记录元数据信息改动的文件。只要元数据发生变化，这个edits文件就会有对应记录。fsimage和edits是NameNode的元数据，元数据在内存中保留一份，同时会在磁盘上保留一份副本。如图2-19所示，标红的abc元数据在内存中存储一份，在fsimage也有存储。

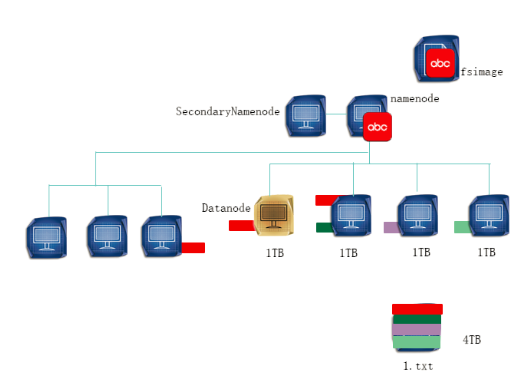


图2-19

fsimage和edits文件会定期做合并，这个周期默认是3600s。fsimage根据edits里改动记录进行元数据更新。元数据信息如果丢失，HDFS就不能正常工作了。fsimage和edits文件的形态如图2-20所示。

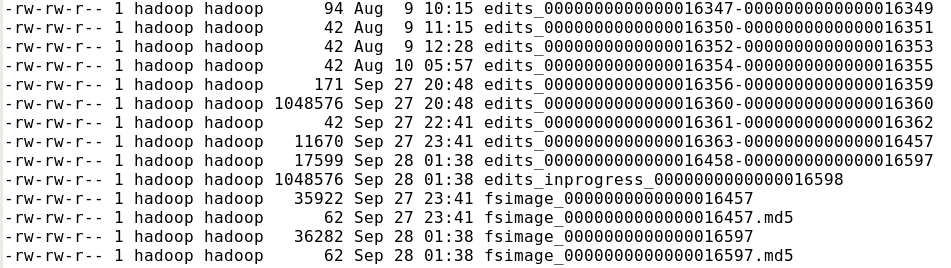


图2-20

在执行Hadoop安装任务的最后，执行了 hadoop namenode -format 这个指令，实际的作用是创建了初始的fsimage文件和edits文件，其中edit-inprogress\*\*是正在被改写的文件。

### SecondaryNameNode执行文件合并

除了NameNode，SecondaryNameNode也会参与fsimage和edits文件的管理工作。SecondaryNameNode负责将fsimage文件定期和edits文件做合并。两个文件被SecondaryNameNode合并之后，将合并后的元数据文件fsimage传给NameNode。这个SecondaryNameNode相当于NameNode辅助节点。这些概念的逻辑结构如图2-21所示。

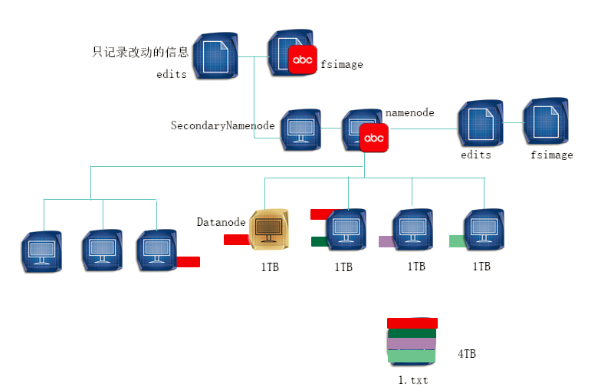


图2-21

Hadoop集群最开始启动的时候，创建Fsimage和edits文件，是这个NameNode做的。此外，NameNode会做一次文件合并工作，这么做的目的是确保元数据信息是最新的，以完成上次停集群的时候，可能还没来的及做完的合并工作。但以后的合并工作，就交给SN（SecondaryNameNode）去做了。这种SN机制是Hadoop1.0的机制。

Hadoop 1.0版本利用了SecondaryNameNode做fsimage和edits文件的合并，但是这种机制达不到热备的效果。其达不到元数据的实时更新，也就意味着了当NameNode挂了的时候，元数据信息可能还会丢失。所以Hadoop1.0的NameNode存在单点故障问题。架构如图2-22。

总结：

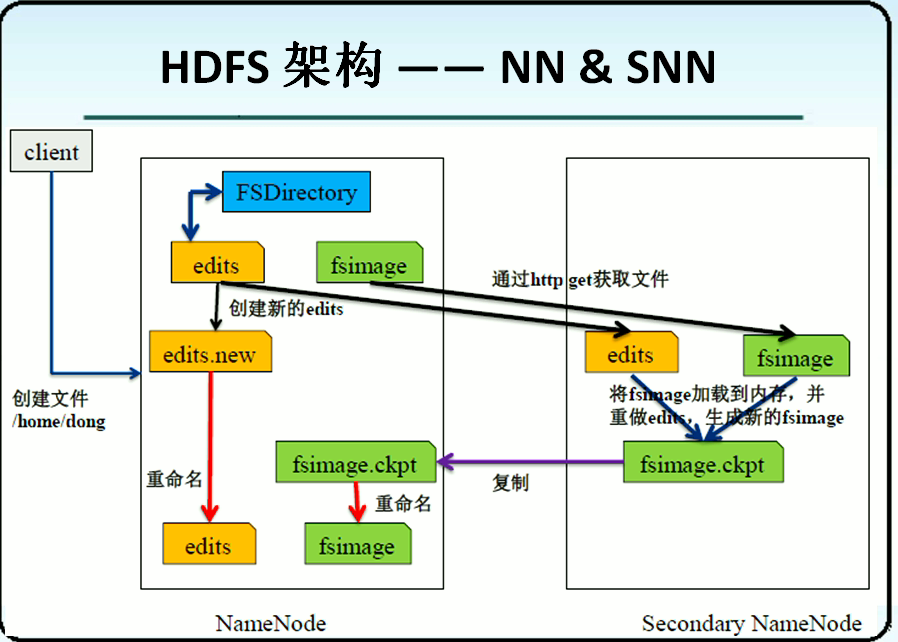


图2-22

触发文件合并的前提条件：

当edits文件的数据大小为64M

当fstime的时间达到3600s时候

1.当edits日志文件的数据达到64M或者fstime中的时间达到3600s开始触发文件合并的过程

2.在Namenode中就会生成一个新的日志文件edits.new

3.SecondNamenode会通过http get将namenode中的原始日志文件edits和磁盘文件fsimage获取过来执行合并操作，形成新的磁盘文件fsimage.ckpt

4.将新产生的磁盘文件fsimage.ckpt复制到namenode中，将原始日志文件删除后将edits.new改名为edits；将原始磁盘文件fsimage删除，并将新的磁盘文件fsimage.ckpt改名为fsimage

在练习Hadoop指令时，练习环境中fsimage和edits丢失的常见错误，就是HDFS命令执行没有结果，或者丢失NameNode进程、格式化信息丢失等。这里总结这些现象，处理方法如下：

1）先把集群停掉；

2）把tmp目录删掉；

3）创建一个新的tmp；

4）执行重新格式化命令：

hadoop namenode -format

5）启动集群。

注意，不能在生产环境中这么做。在生产环境里，做法是一般是第一次执行完格式化之后，在配置文件，配置禁止格式化。

## Hadoop插件

### 安装步骤

1）下载Hadoop插件，注意插件的版本要和用的Hadoop版本保持一致，如图2-23所示。

计算机生成了可选文字:
4 
hadoopbin for hadoop2 ． 70 .zip 
hadoop-eclipse-plugin-2.7.1 .jar 
2015 / 7 / 25 1532 
2015 / 5 / 29 21 ： 33 
350 压 缩 ZIP 文 亻 牛 
Executable Jar File 
782 KB 
3 乙 935 KB 

图2-23

2）将插件jar包放在eclipse安装目录的plugins目录下。

3）将Hadoop安装包解压到指定的一个目录（后面要用这个安装目录），如图2-24所示。

计算机生成了可选文字:
> 此 电 脑 > 教 字 (D:) > 
hadoop 
名 称 
hadoop 一 2 ． 70 
虚 拟 机 
过 
hadoop 一 2 ． 70 
64bit.tar.gz 
修 改 日 耻 
2015 / 9 / 4 1 5 ： 24 
2015 / 8 / 31 23 ： 54 
2015 / 9 / 1 3 5 ： 57 
文 亻 牛 夹 
文 亻 牛 夹 
350 压 缩 
1 
20 氵 571 KB 

图2-24

4）重启Eclipse， 选择菜单栏中的“Window”→“Preferences”命令，在打开的”Preferences”对话框中，选项左边Hadoop Map/Reduce选项， 单击右侧“Browse”按钮， 找到Hadoop的安装目录， 然后单击“Apply”按钮，再单击“OK”按钮，如图2-25所示。

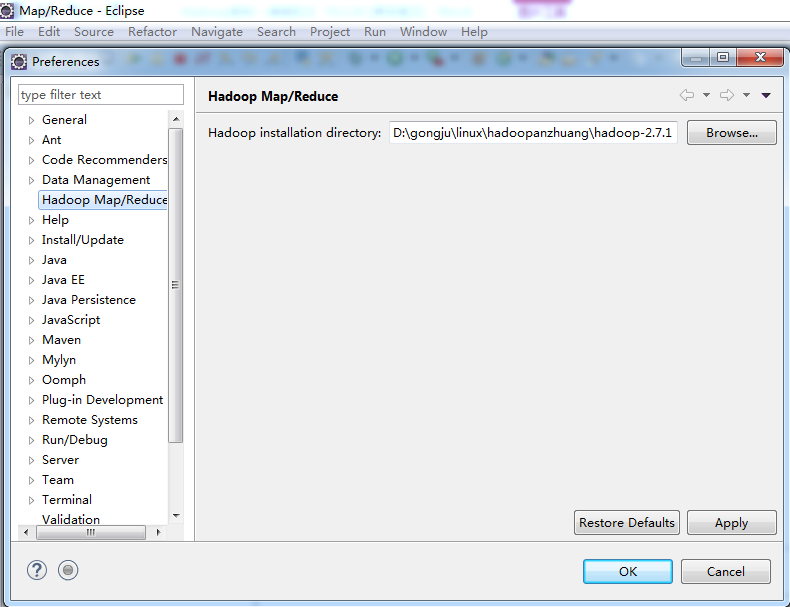


图2-25

5. 选择菜单栏中的“Window”→“Show View”命令，在弹出的“Show View”对话框中，调出Map/Reduce视图，再单击“OK”按钮，如图2-26所示。

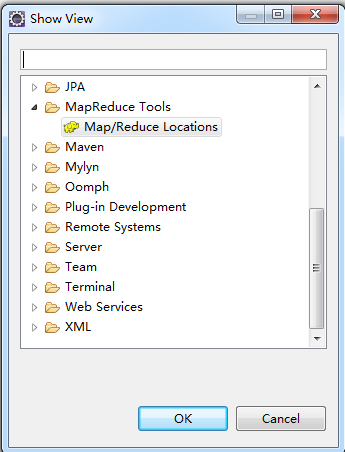


图2-26

6. 在Map/Reduce视图下，单击右上角带有“+”号的蓝色大象按钮，新建Hadoop客户端连接，如图2-27所示。

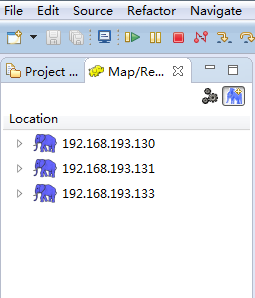


图2-27

7. 在下面的选项卡里，填好NameNode节点的IP地址及相应的端口号，如图2-28所示。

计算机生成了可选文字:
Define Hadoop location 
Define the location Of a Hadoop infrastructure for running MapReduce applications. 
General Advanced parameters 
Locatlon name: 19205 & 234 ． 21 
Map/Reduce(V2) Master 
Host: 19205 & 234 ． 21 
Port: 50020 
m e: root 
SOCKS proxy 
DFS Master— 
Use M/R Master host 
Host: 
Port: 
1 920 5 & 234 ． 21 
9000 
囗 Enable SOCKS proxy 
host 
Host: 
Port: 
1080 

  图2-28

9. Hadoop的权限修改

如图3-29所示，填写的用户名是“root”。每次更改文件，可能都需要以root用户登录，或伪装成root用户，这样比较麻烦。可以在hdfs-site.xml文件中配置，属性值改成false即可，如图2-29所示。

计算机生成了可选文字:
<configuration> 
<propertY> 
< ！ 一 指 定 hdfs 保 存 数 据 副 本 的 数 量 ， 包 括 自 己 ， 
< ！ 一 伪 分 布 式 模 式 ， 此 值 必 须 为 1 一 > 
<n a me>d fs · replicat ion</name> 
<value>l</value> 
</propertY> 
'-propertY> 
•<n a me>dfs permiss ions•</name> 
<value>false•</value> 
</propertY> 
</configuration> 
默 认 为 3 一 

   图2-29

更改完之后，需要重新启动Hadoop。

由于下载Hadoop 2.0以上版本时，在Hadoop 2.0的bin目录下没有winutils.exe。需将资料里hadoopbin\_for\_hadoop2.7.1zip包里的所有文件，复制到hadoop2.7.1安装目录的bin目录下即可，如图2-30所示。

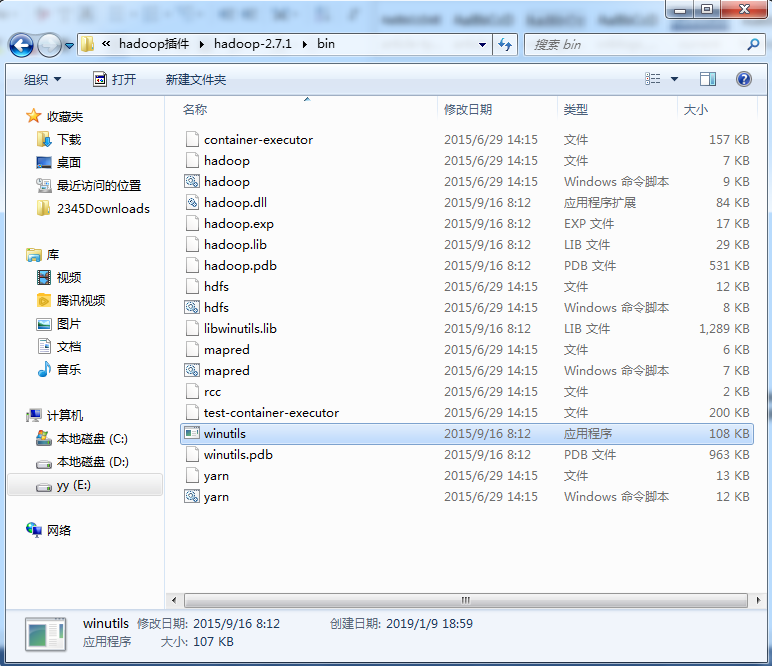


图2-30

然后配置变量环境HADOOP\_HOME 和path。右击系统桌面上的“计算机”图标，在弹出的快捷菜单中选择“属性”命令，在打开的“系统”窗口中选择左边的“高级系统设置”项，打开“系统属性”对话框，在“高级”选项卡中单击底部的“环境变量”按钮，打开“环境变量”对话框。单击“系统变量”栏中的“新建”按钮，打开“新建系统变量”对话框，输入变量名“HADOOP\_HOME”，选择变量值为Hadoop的安装目录，单击“确定”按钮，添加HADOOP\_HOME系统变量。在“环境变量”对话框中选择path变量，单击“编辑”按钮，将HADOOP\_HOME添加到path路径下，如图2-31、图2-32所示。

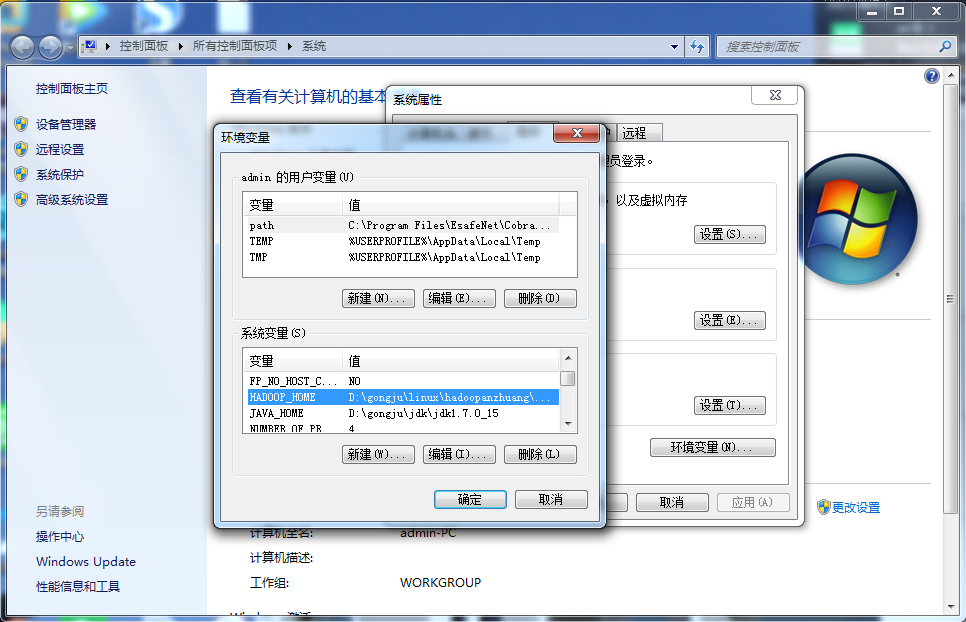


图2-31



图2-32

11 . 将hadoop.dll文件复制到C:\Windows\System32下面，然后重新启动，如图2-33所示。

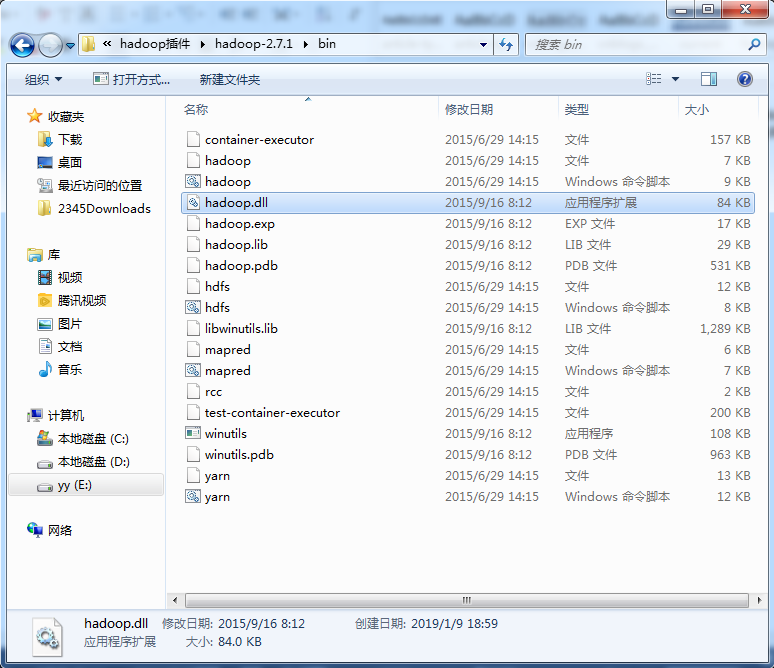


图2-33

在Eclipse中的Project Explorer选项卡中展开DFS Locations，即可查看到HDFS文件系统的视图。

### Hadoop插件操作HDFS文件系统

插件安装完成， 用户也可以通过可视化的方式对HDFS文件系统进行增、删、查及下载等操作。

1.新增目录

在HDFS文件系统目录的视图中，右击蓝色大象图标项下面的文件，在弹出的快捷菜单栏中选择“Create new directory”命令，打开“Create subfolder”对话框，填入新建目录名称“park”并单击“OK”按钮，完成新目录创建，如图2-35、图2-36所示。

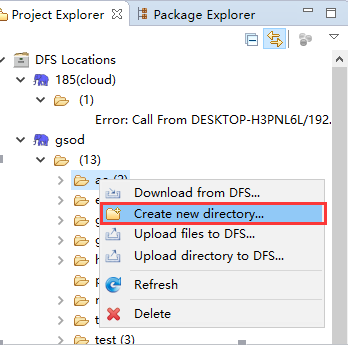


图2-35

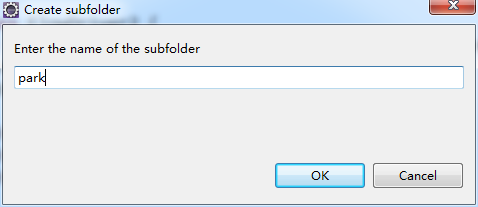


图2-36

选中目录，右击单击刷新创建的目录就可以看到目录。如下图2-39所示：

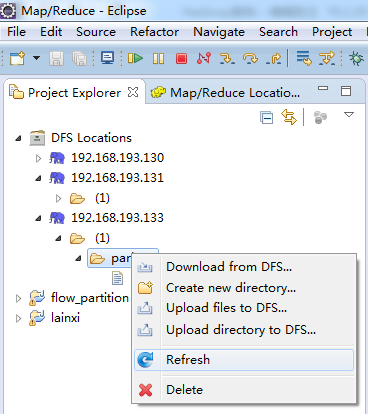
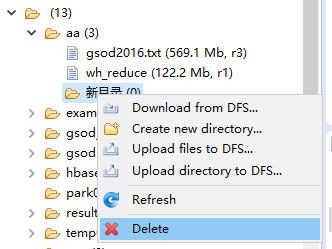


图2-37

2.删除目录/文件

选中需要删除的目录/文件右击，在弹出的快捷菜单中选择“Delete”命令，即可删除该目录/文件。如下图2-38所示：

  
图2-38

3．上传文件

选择文件上传的位置，右击，在弹出的快捷菜单中选择“Upload files to DFS”命令，再刷新目录，即可看到上传的文件，如图2-39所示。



图2-39

4.下载文件

选中需下载的文件，右击，在弹出的快捷菜单中选择“Download from DFS”命令，即可下载该问价，如图2-40所示

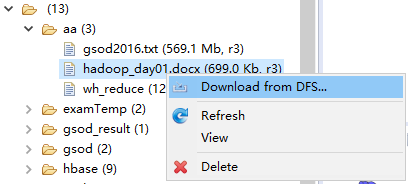


图2-40

## JavaAPI操作HDFS

### 创建MapReduce工程

1) 创建MapReduce工程；单击“File”菜单，选择“New”->“Other”命令，打开“New”对话框。选择“Map/Reduce Project”项，再单击“Next”按钮，如图2-41、图2-42所示。

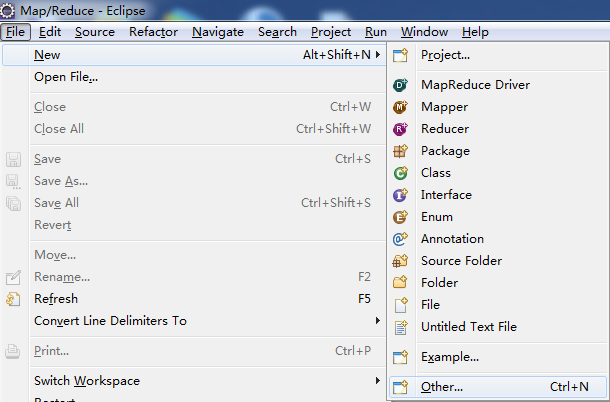


图2-41

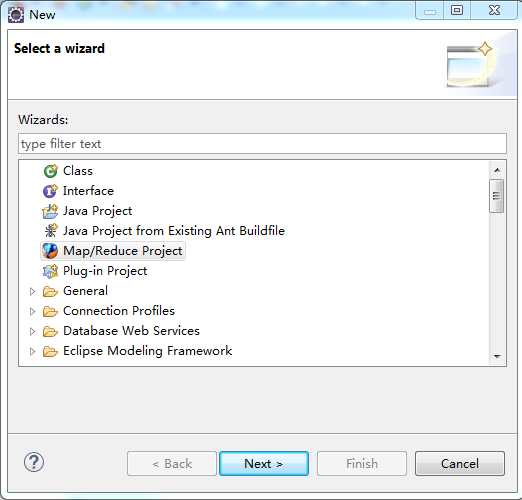


图2-42

2) 在打开的“New MapReduce Project Wizard”对话框的“Project name”文本框中填写项目名称“HDFS\_Test”，再单击“Finish”按钮，如图2-43所示。

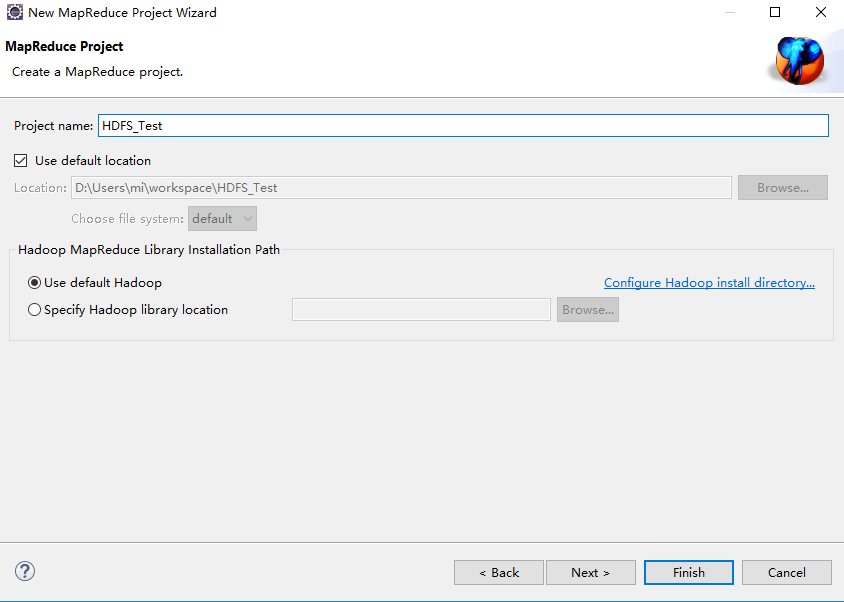


图2-43

在HDFS\_Test的Project Explorer 视图中，能够看到如下的包依赖，这样就可以不用导入Hadoop相关jar包。这也是插件给编程带来的好处。如下图2-44所示：

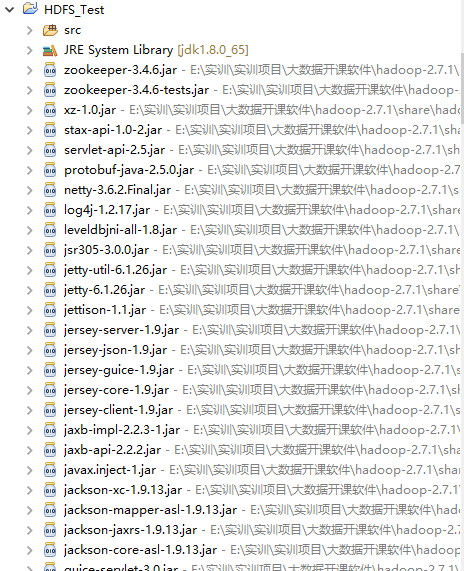


图2-44

### HDFS API创建目录实现

右击Project Explorer选项卡中的File项，在弹出的快捷菜单中选择“New”->“Class”命令，如图2-45所示。在打开的“New Java Class”对话框中填入类名“TestDemo”，再单击“Finish”按钮，完成TestDemo类的创建，如图2-46所示：

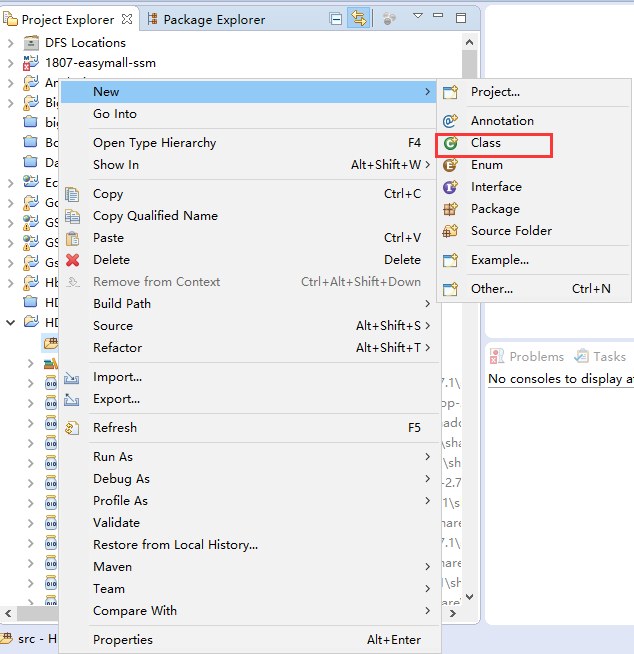


图2-45

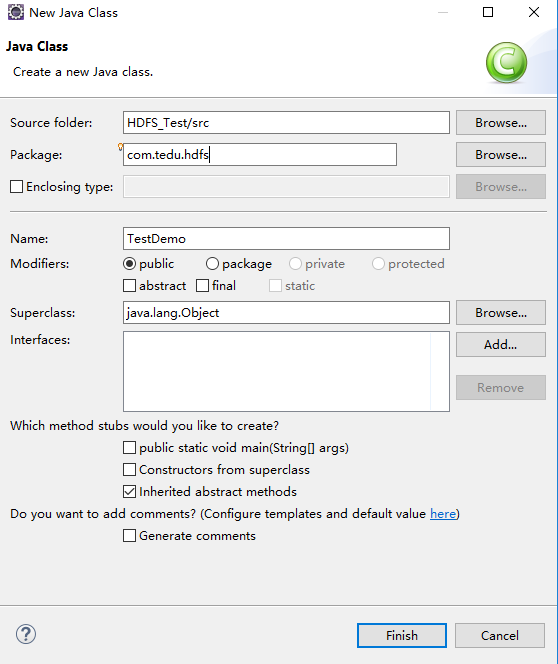


图2-46

在主函数中编写创建目录的代码实现，使用HDFS API需用到两个重要的类，一个是Configuration类是Hadoop的环境变量类，另一个是FileSystem类，是一个文件系统类。首先创建Configuration对象，然后根据NameNode 地址和 环境变量获取 文件系统，接着调用文件系统的创建目录方法，最后关闭文件系统。具体代码实现如下：





在代码段中右击，在弹出的快捷菜单中选择“Run As”->”Run on Hadoop”命令，如图2-47所示。

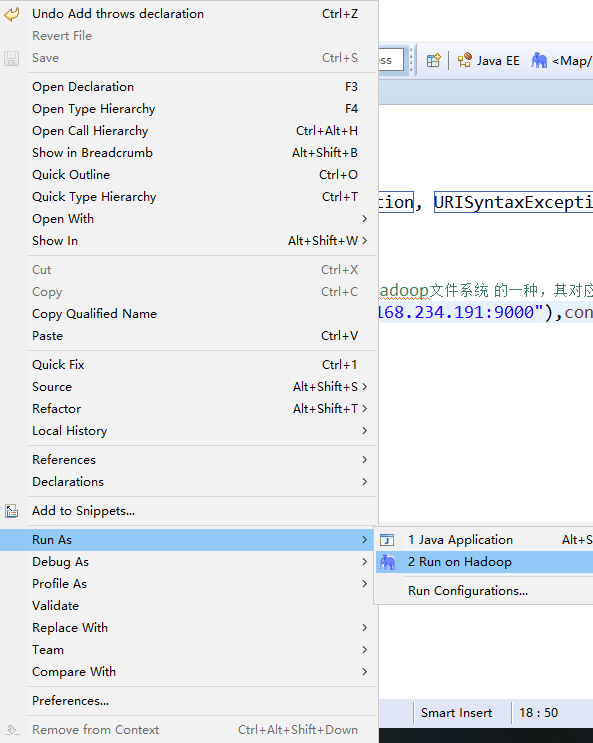


图2-47

运行结束后刷新HDFS插件中的根目录，就可以看到该目录。如下图2-48所示：

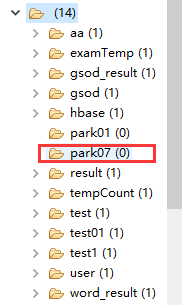


图2-48

### CopyBytes方法

通过输入输出流实现文件的下载/上传，Hadoop中提供了IOUtils类中copyBytes(in, out, conf)方法，它实现了文件上传至HDFS或者从HDFS中下载文件到本地。方法API介绍如下：



从一个流复制到另一个流。最后关闭输入和输出流。

参数：

in - 要读取的InputStrem

out - 要写入的OutputStream

conf - 配置对象

### 文件下载download

在单元测试中，首先创建Configuration对象，然后根据NameNode 地址和环境变量获取 文件系统，接着获取filesystem中下载的文件的输入流，接着创建下载文件的输出流，然后调用copyBytes(in, out, conf)，最后关闭文件系统。具体代码如下所示：



在该方法处右击，在弹出的快捷菜单中选择“Run As”->“JUnit Test”命令,运行结果如下图2-49所示：

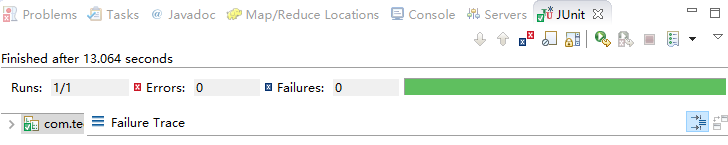


图2-49

当刷新工程时，在工程下可以看到下载的文件，如下图2-50所示：

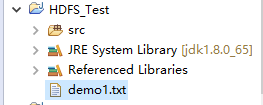


图2-50

### 文件的上传upload

在单元测试中，首先创建Configuration对象，然后根据NameNode地址和环境变量获取文件系统，接着创建上传文件的输入流，接着创建filesystem中上传的文件的输出流，然后调用copyBytes(in, out, conf)，最后关闭文件系统。具体代码如下所示：



在运行代码之前，需要确保上传的文件是否存在，如果不存在需创建。我们将工程下面的2.txt文件上传到park01目录下，如下图2-51所示：

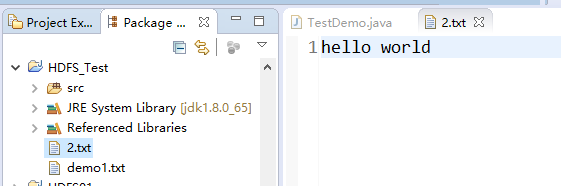


图2-51

在该方法处右击，在弹出的快捷菜单中选择“Run As”->“JUnit Test”命令，运行结果如下图2-52：

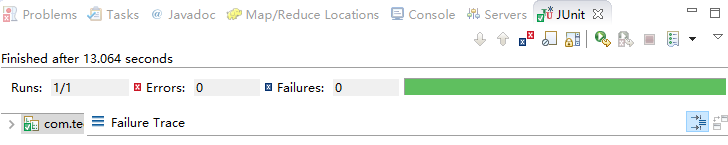


图2-52

刷新HDFS文件系统中的park01目录，就可以查看到上传的2.txt文件。如下图2-53所示：

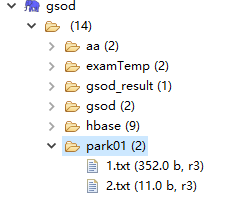


图2-53

## HDFS的执行流程

### HDFS的写流程

在HDFS架构中，当客户端发起一个写数据请求，并指定上传文件的路径，然后去找NameNode。NameNode首先会判断路径合法性，然后会判断此客户端是否有写权限。如果都满足，NameNode会给客户端返回一个输出流。此外，NameNode会为文件分配块存储信息。注意，NameNode也是分配块的存储信息，但不做物理切块工作。如下图2-54所示：



HDFS client：HDFS 客户端

Name Node：管理者节点

Data Node：工作者节点

Create file：传输文件

Write packet：传包

Send ack：返回信息

Close file：传输文件完毕

图2-54

客户端拿到输出流以及块存储信息之后，就开始向DataNode写数据。因为一个块数据，有三个副本，所以图里有三个DataNode。packet初学时可以简单理解为就是一块数据。

HDFS架构中的DataNode是数据节点。用于实际存储文件块。为了防止DataNode宕机造成的文件块数据丢失，HDFS给文件块配置了数据冗余机制。一个文件块有三个副本，当第一个副本存储在一台主机上之后，这台主机会产生第二个副本，传递给在本机架的另一台主机，存储第二个副本。存储第二个副本的主机，会产生第三副本，并将这个副本传递给和第一副本、第二副本不在同一个机架的主机，由这台主机存储生成第三个副本。

生成第二个副本、第三个副本时，节点通过ack确认机制，向上游节点发送确认，这么做的目的是确保块数据复制的完整性。

HDFS通过最上游节点，向客户端发送ack,如果块数据没有发送完，就继续发送下一块。如果所有块数据都已发完，就可以关闭流了。

所有块数据都写完后，Client通知NameNode关闭流。文件写过程就关闭了。

要注意到文件块也就是数据块，传递第二个数据块副本、第三数据块副本时，不是使用Client网络，而是Hadoop集群内网络。这与数据块大小的问题有关。从大数据处理角度来看，块越大越好。因为数据块大，会减少磁盘寻址次数，从而减少寻址时间。也要看到数据块大的另一个影响，就是占用网络带宽，生成冗余数据时使用Hadoop内网络，对于提高网络带宽性能有帮助。

### HDFS的读流程

理解了HDFS架构中的数据写过程，接下来继续了解HDFS中的数据读过程，如下图2-55所示：

计算机生成了可选文字:
HDFS client 
Client nod 
Open file 
Get block 
location 
Close file 
k 
ad block 
ame N Od 
Data Node 
Data Node 
Data Node 

HDFS client：HDFS 客户端

Name Node：管理者节点

Data Node：工作者节点

Open file：打开文件

Get block location：得到文件块的位置

Read block：读取文件块

Close file：读取文件完毕

图2-55

客户端发出读数据请求，NameNode是HDFS的对外服务主机，是响应外部请求的管理节点。客户端提交给NameNode的Open File指定读取的文件路径， NameNode会检索自己内存中存储元数据信息，或者检索持久化的元数据fsimage文件和edits文件。

NameNode将文件的元数据信息检索结果，返回给客户端。客户端根据返回的元数据信息，去对应的DataNode去读取数据块。这里要注意是客户端去对应的DataNode读取数据块，而不是NameNode去读取数据块，然后将数据块传递给DataNode。也就是NameNode既不读数据、也不传递数据。客户端根据返回的元数据信息，去对应的DataNode去读块数据时，假如一个文件特别大，比如1TB，会分成好多块，此时，NameNode并是不一次性把所有的元数据信息返回给客户端。客户端读完此部分后，再去NameNode要下一部分的元数据信息，再接着读。

客户端读完数据之后，通知NameNode关闭文件流。

## HDFS的特点

### 优点

1.HDFS在大数据架构中一个显著的特点就是支持超大文件。

一般来说HDFS存储的文件，可以支持TB和PB级别的数据容量。

2.HDFS也能检测和快速应对硬件故障。

在集群环境中，硬件故障是常见性问题。因为有上千台服务器连在一起，故障率很高，因此故障检测和自动恢复是HDFS文件系统的一个设计目标。假设某一个DataNode挂掉之后，因为数据是有备份的，还可以从其他节点里找到。NameNode通过心跳机制来检测DataNode是否还存活。

3.Hadoop使用简化的一致性模型。

这个特征的好处是，对于外部使用用户，不需要了解Hadoop底层细节，比如文件的切块，文件的存储，节点的管理。

4.在Hadoop 2.0版本允许数据的追加，不允许数据的修改。

虽然如此，但是仍然不建议使用数据追加。因为一个文件存储在HDFS上后，适合一次写入，多次读取的场景。因为存储在HDFS上的文件都是超大文件，当上传完这个文件到Hadoop集群后，会进行文件切块，分发，复制等操作。如果文件被修改，会导致重新触发这个过程，而这个过程耗时是最长的。

5.HDFS还有高容错性的特点。

数据在HDFS上自动保存多个副本，副本丢失后自动恢复。集群可构建在廉价的机器上，实现线性扩展。当集群增加新节点之后，NameNode也可以感知，将数据分发和备份到相应的节点上。

6.Hadoop可以使用普通商用硬件。

Hadoop并不需要运行在昂贵且高可靠的硬件上。它是设计运行在普通商用硬件（在各种零售店都能买到的普通硬件）的集群上的，因此至少对于庞大的集群来说，节点故障的几率还是非常高的。HDFS遇到上述故障时，被设计成能够继续运行且不让用户察觉到明显的中断。

### 缺点

1.不能做到低延迟数据访问。

由于Hadoop针对高数据吞吐量做了优化，牺牲了获取数据的延迟。所以对于低延迟数据访问，不适合Hadoop。对于低延迟的访问需求，HBase是更好的选择。

2.不适合大量的小文件存储。

由于NameNode将文件系统的元数据存储在内存中，因此该文件系统所能存储的文件总数受限于NameNode的内存容量。根据经验，每个文件、目录和数据块的存储信息大约占150字节。因此，如果有一百万个小文件，每个小文件都会占一个数据块，那至少需要300MB内存。如果是上亿级别的，就会超出当前硬件的能力。

3.不适合多用户写入文件、修改文件。

对于上传到HDFS上的文件，不支持修改文件。Hadoop2.0虽然支持了文件的追加功能，但是还是不建议对HDFS上的文件进行修改。因为效率低下。HDFS适合一次写入，然后多次读取的场景。

# MapReduce的基本简介

## 分布式计算框架

Hadoop计算框架是出现比较早的一个分布式计算框架，它主要是基于MapReduce的开发模式，一个开源实现功能非常强大的分布式计算框架，由Java开发完成。Hadoop分布式计算框架包括两个部分，计算框架MapReduce与用来存储计算数据的存储框架HDFS。

MapReduce源自于google的MapReduce论文，发表于2004年12月， Hadoop MapReduce是google MapReduce克隆版。MapReduce是一种分布式计算模型，用以进行大数据量的计算。它屏蔽了分布式计算框架的细节，将计算抽象成map和reduce两部分， 其中Map对数据集上的独立元素进行指定的操作，生成键-值对形式中间结果。Reduce则对中间结果中相同“键”的所有“值” 进行规约，以得到最终结果。MapReduce非常适合在大量计算机组成的分布式并行环境里进行数据处理。使用MapReduce的好处是可以让不懂分布式编程的程序员也可以把程序运行分布式环境下， 比如任务分配，逻辑切块，位置追溯都不需要考虑。程序员可以把主要精力放在核心业务上，大大缩短任务周期。此计算框架最大的优势在于，分布式的运算可以大大缩短运算周期。

分布式计算简单来说，是把一个大计算任务拆分成多个小计算任务分布到若干台机器上去计算，然后再进行结果汇总，目的在于分析计算海量的数据。雷达监测的海量历史信号中分析异常信号(外星文明)，淘宝双十一实时计算各地区的消费习惯等例子，都是用分布式计算框架分析海量数据的。

海量计算最开始的方案是提高单机计算性能，如大型机，后来由于数据的爆发式增长、单机性能却跟不上，才有分布式计算这种妥协方案。因为计算一旦拆分，问题会变得非常复杂，像一致性、数据完整、通信、容灾、任务调度等问题也都来了。

举个例子，产品要求从数据库中100G的用户购买数据，分析出各地域的消费习惯金额等。如果没什么时间要求，程序员小明就写个对应的业务处理服务程序，部署到服务器上，让它慢慢跑就是了，小明预计10个小时能处理完。 后面产品嫌太慢，让小明想办法加快到3个小时。

平常开发中类似的需求也很多，总结出来就是，数据量大、单机计算慢。 如果用Hadoop、storm之类成本较高、而且有点大才小用。 当然让老板买更好的服务器配置也是一种办法。

Mapreduce是Hadoop的一种分布式计算框架，在分布式并行的环境中进行数据处理。

## MapReduce概述

### MapReduce的由来及定义

MapReduce是一个分布式计算框架（编程模型），简称MR，最初由由谷歌的工程师开发，基于GFS的分布式计算框架, 主要用于搜索领域，解决海量数据的计算问题。后来Cutting根据《Google MapReduce》,设计了基于HDFS的MapReduce分布式计算框架。

MapReduce框架对于程序员的最大意义在于，不需要掌握分布式计算编程，不需要考虑分布式编程里可能存在的种种难题，比如任务调度和分配、文件逻辑切块、位置追溯等工作。这样，软件工程师能够把大部分精力放在核心业务层面上，大大简化了分布式程序的开发和调试周期。

MapReduce由两个阶段组成： Map阶段和Reduce阶段，用户只需要实现map()和reduce()两个函数，即可实现分布式计算，非常简单。这两个函数的形参是key、value对，表示函数的输入信息。

### MapReduce的体系架构图

前面介绍的MapReduce的由来及定义，接下来介绍一下MapReduce的体系架构图，如下图3-1所示：

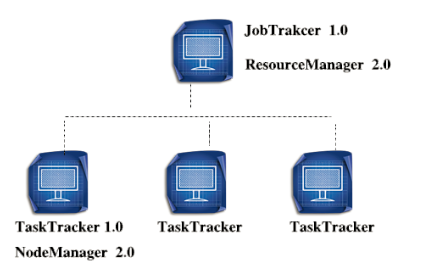


图 3-1

MapReduce与HDFS一样使用了master/slave结构，master/slave结构:是一个基于分而治之思想设计模式，将一个任务（原始任务）分解为若干个语义等同的子任务，并由专门的工作者线程来并行执行这些任务，原始任务的结果是通过整合各个子任务的处理结果形成的。JobTrakcer和TaskTacker是Hadoop 1.0的版本，ResourceManager和NodeManager是Hadoop 2.0的版本。JobTracker/ResourceManager是任务调度者。TaskTrakcer/NodeManager: 任务执行者。下面来分别介绍一下它们各自的职能。

JobTracker / ResourceManager的主要工作职能：知道管理哪些机器，即管理哪些NodeManager；要有检测机制，能够检测到NodeManager的状态变换，通过RPC心跳来实现；任务的分配和调度，ResourceManager能够做到细粒度的任务分配，比如某一个任务需要占用多大内存，需要多少计算资源。

ResourceManager是Hadoop 2.0版本之后引入了yarn，有yarn来管理Hadoop之后，jobtracker就被替换成了ResourceManager。

TaskTracker / NodeManager的主要工作职能：能够收到ResourceManager发过来的任务，并进行任务处理。这里处理任务指的是Map任务或Reduce任务。

### 基于Map和Reduce的并行计算模型

通过前面介绍的MapReduce的由来及定义和MapReduce体系架构图，已经对MapReduce有了一定的理解，下面来介绍一下Map和Reduce并行计算模型，如图3-2所示：

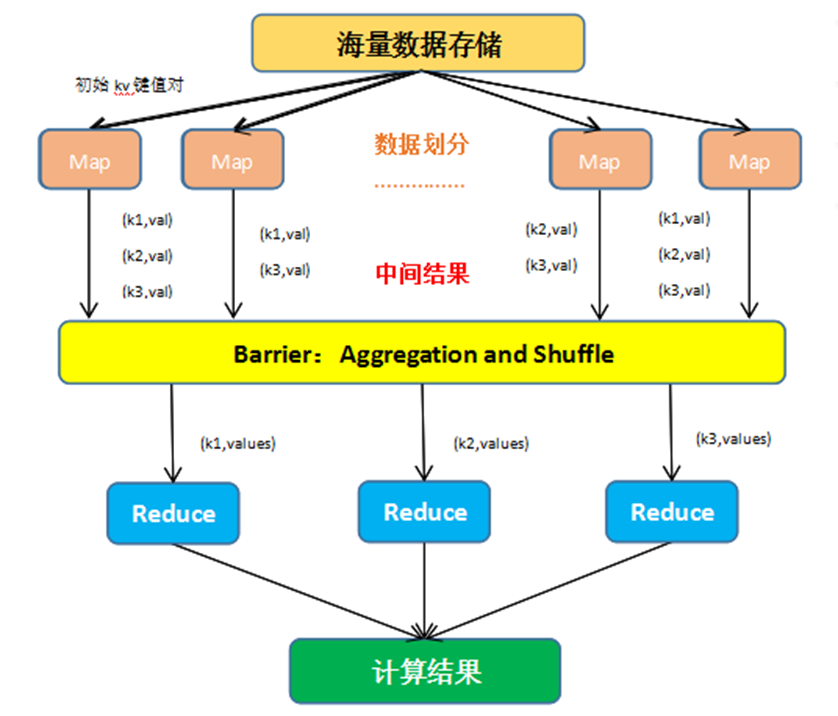


图3-2

MapReduce 并行计算模型结合用户实现的Map 和 Reduce 函数。用户自定义的 Map 函数处理了一个基于 key/value的集合，MapReduce把中间所有具有相同key 的value集合在一起后传递给 Reduce 函数，用户自定义的Reduce函数合并所有具有相同key的value值， 形成一个较小 value 值的集合。下面详细介绍一下map任务和reduce任务处理的内容。

map任务处理：读取输入文件中的内容，解析成key、value对。对输入文件的每一行，解析成key、value对。每一个键值对调用一次map函数；对输入的key、value进行处理，转换成新的key、value输出；对输出的key、value进行分区； 对相同分区的数据，按照key进行排序(默认按照字典顺序进行排序)、分组。相同key的value放到一个集合中。

reduce任务处理：对多个map任务的输出，按照不同的分区，通过网络copy到不同的reduce节点。这个过程并不是map将数据发送给reduce，而是reduce主动去获取数据； 对于多个map任务的输出进行合并、排序。reduce函数的逻辑是：对输入的key、value进行处理，转换成新的key、value输出；把reduce端的输出保存到文件中(HDFS)。

## MapReduce实现单词统计案例WordCount

### 案例前准备

1.创建一个park的目录；右键单击蓝色大象下面的文件，在弹出的菜单栏中选择“create new directory”命令，打开“Create subfolder”对话框，填上新建目录名称“park”， 并单击“OK”按钮，完成目录创建。如图3-3，图3-4所示：

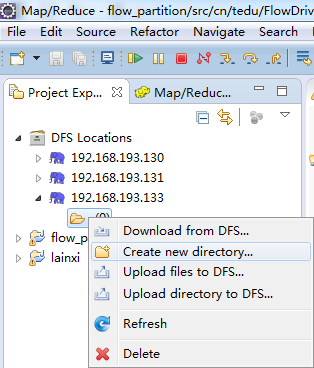


图3-3

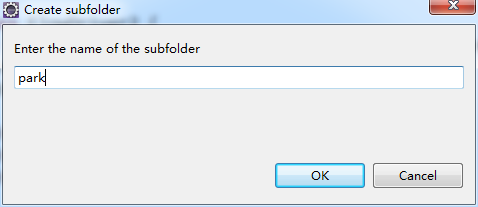


图3-4

2.右击创建好的park目录，在弹出的快捷菜单中选择“Upload files to DFS”命令，在弹出的对话框中选择words.txt文件，单击“确定”按钮，将其上传到park目录下，如图3-5，图3-6所示：

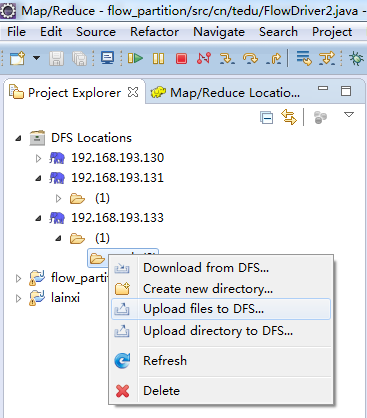


图3-5

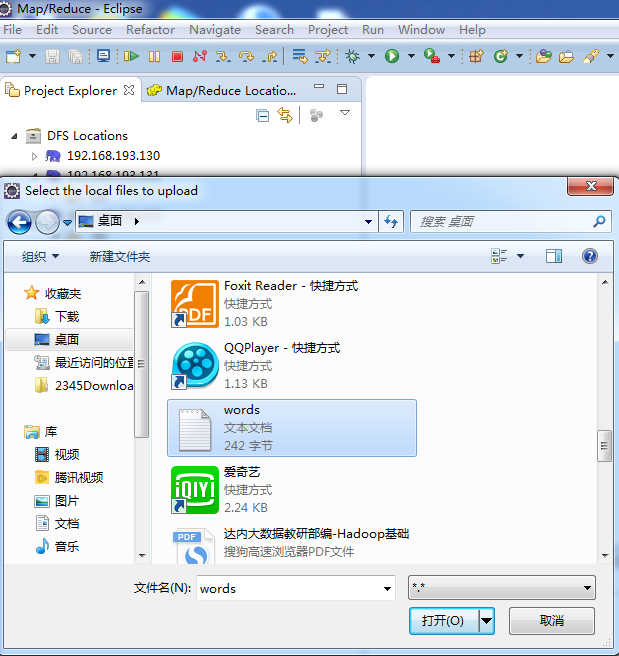


图3-6

3.右键单击park文件夹，在弹出的快捷菜单中选择“Refresh”命令， 可以看见words.txt文件,证明words.txt已经成功上传到HDFS上，如图3-7所示：

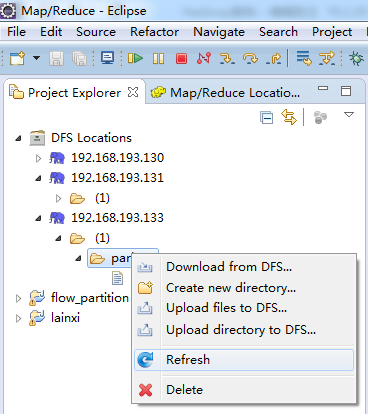


图3-7

创建一个新项目；单击鼠标右键，在弹出的快捷菜单中选择 “New”->“Project”命令， 打开“New Project”对话框，再单击“Next”按钮， 如图3-8所示：

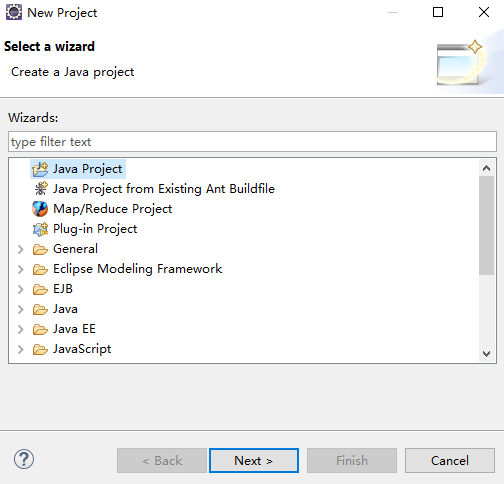


图3-8

填写项目名称“WordCount”，单击“Finish”按钮，完成项目创建。如图3-9所示：

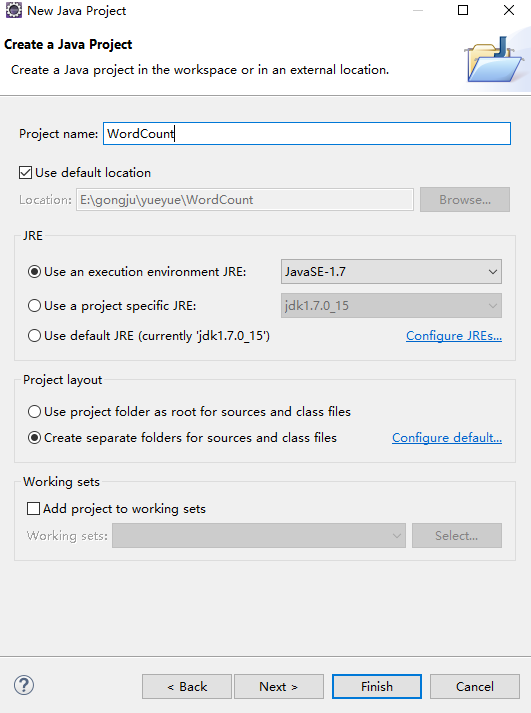


图3-9

新建一个lib文件夹；单击鼠标右键，选择“New”->“Folder”命令， 打开New Folder对话框。选择WordCount文件夹，在下面的Folder name文本框中输入“lib”，再单击“Finish”按钮，完成lib目录的创建，如图3-10所示。

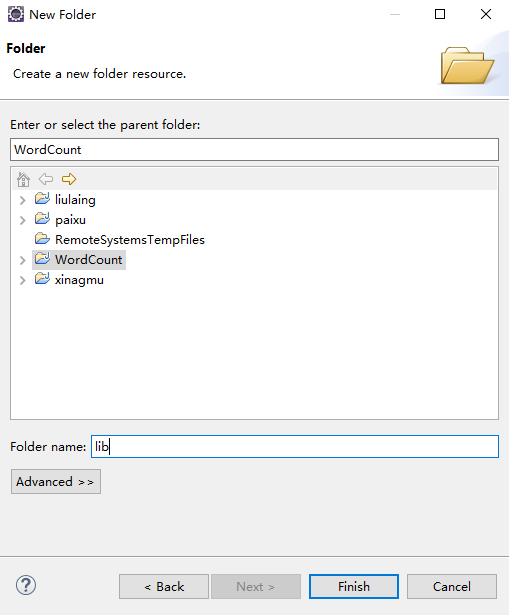


图3-10

7.将项目中所需要的jar全部导入到bin的目录下，如图3-11所示：

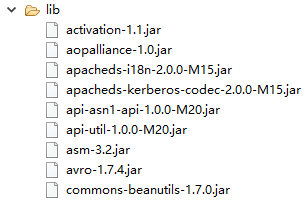


图3-11

8.选中jar包，单击鼠标右键，单击Build Path，然后单击Add to Build Path，如图3-12所示：

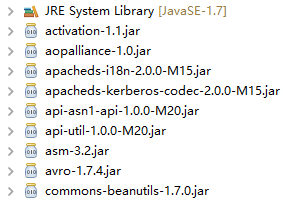


图3-12

9.新建一个类WCMapper；单击鼠标右键，选择“New”->“Class”命令，打开New Java Class对话框，将类名称WCMapper填写上，再单击Finish完成。类WCMapper创建完成。如图3-13所示：

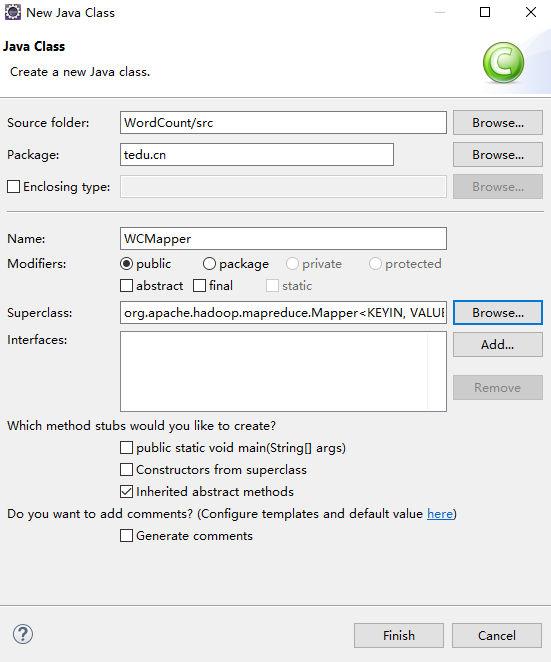


图3-13

10.类WCMapper需要继承一个Mapper类，因此单击Superclass文本框后的

“Browse”按钮，打开的“Superclass Selection”对话框，在“Choose a type”文本框中输入“Mapper”，然后选择“Mapper-org.apache.hadoop.mapreduce”项，如图3-14所示。最后单击“OK”按钮，WCMapper就继承了Mapper类。

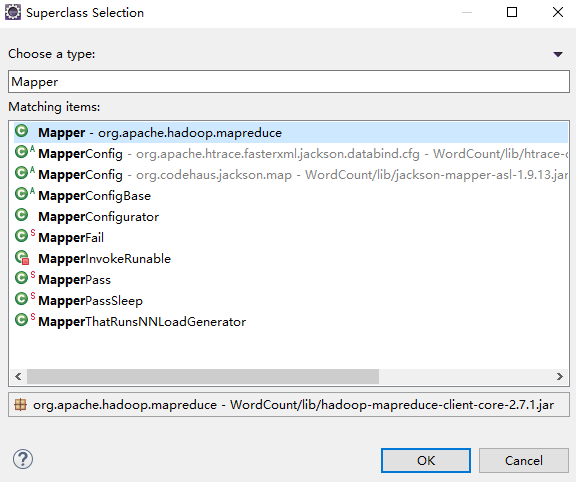


图3-14

11.新建一个类WCReducer；单击鼠标右键，选择“New”->“Class”命令，打开New Java Class对话框，将类名称WCReducer填写上，再单击Finish完成。类WCReducer创建完成。如图3-15所示：

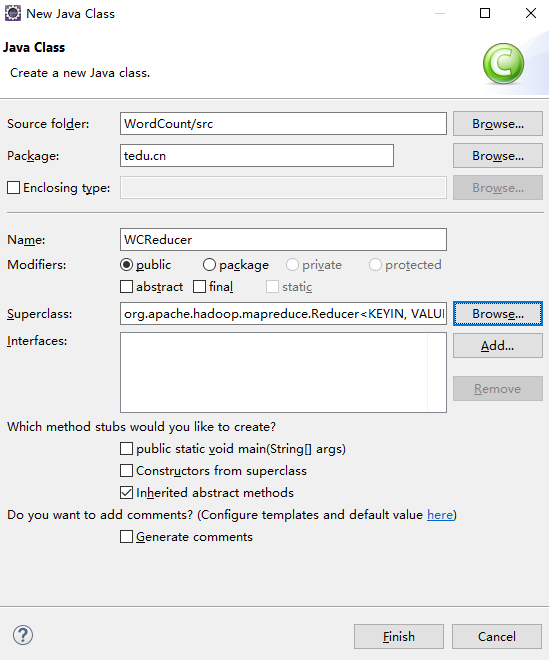


图3-15

12.类WCReducer需要继承一个Reducer类；因此单击Superclass文本框后的

“Browse”按钮，打开的“Superclass Selection”对话框，在“Choose a type”文本框中输入“Reducer”，然后选择“Reducer-org.apache.hadoop.mapreduce”项，如图3-16所示。最后单击“OK”按钮，WCReducer就继承了Reducer类。

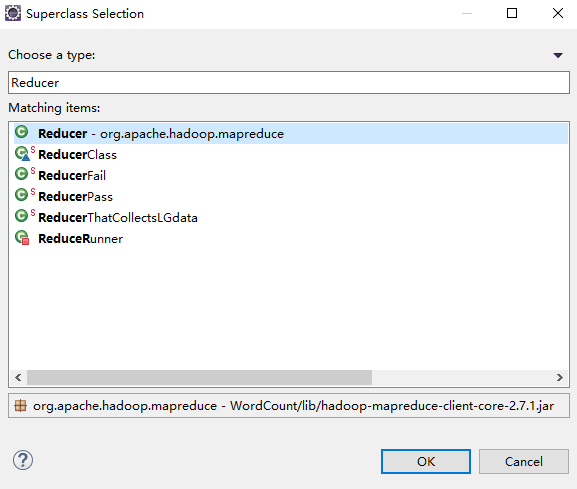


图3-16

13.新建一个类WCDriver；单击鼠标右键，选择“New”->“Class”命令，打开New Java Class对话框，将类名称WCDriver填写上，最后单击Finish。类WCDriver创建完成。如图3-17所示：

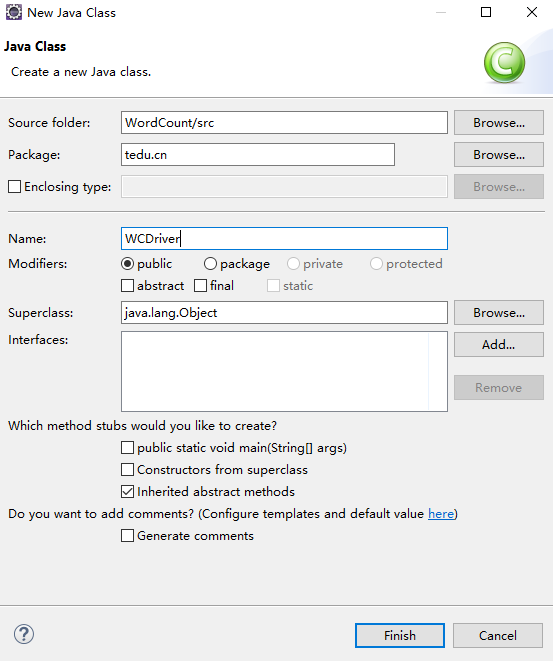


图3-17

### 案例分析

HDFS负责存储数据，MR计算框架负责计算HDFS上的数据，比如words.txt文件。即MR想要计算某个文件，这个文件必须先要在HDFS上进行存储。MR计算文件，会对文件进行逻辑切块，每一块是一个InputSpilt，每有一个InputSplit，会启动一个Map任务。逻辑切块的大小是128MB。现在假设words.txt正好是128MB，则会有一个InputSpilt，有一个map任务分配给集合里某一个TaskTracker处理。如果小于128MB，也是一个InputSpilt，有一个map任务分配给集群里某一个TaskTracker处理。如果是129MB，将会有两个InpuSpilt，两个map任务，此时会根据集群里TaskTracker数量及状态分配map任务，这个工作MR框架会做的，不需要程序员来考虑。此外，因为涉及到了逻辑切块，所以涉及到：要处理的文件在HDFS的位置信息（在哪台DataNode节点上）、计算start、end位置信息以及位置追溯等问题，统统都交给MR框架来做。不需要考虑。MapReudce计算框架抽象出两个任务阶段，Map任务阶段和Reduce任务阶段。

此案例是将words.txt文件中的单词出现的总次数统计出来，使用map函数将单词进行拆分，使用reduce函数进行汇总。如图所示3-18所示：是MR实现wordcount的全过程。

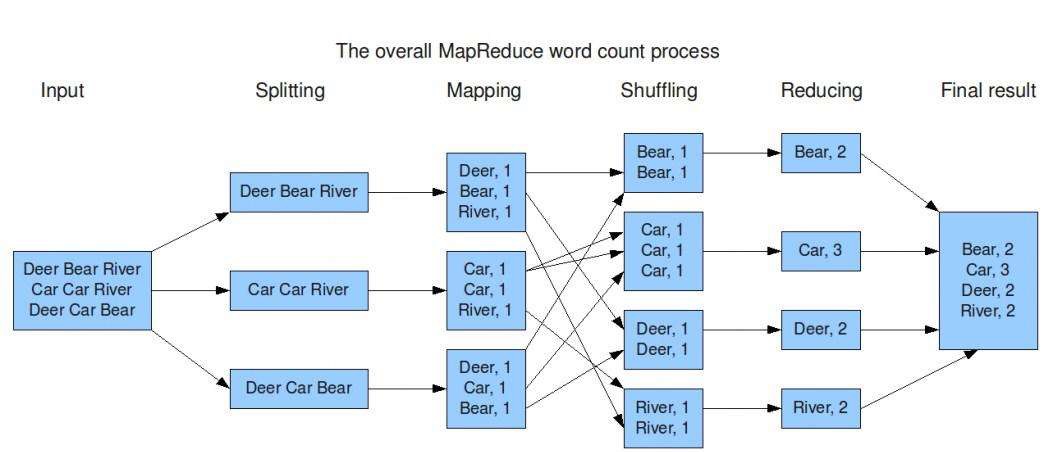


图3-18

MapReduce实现WordCount的过程：首先将文件拆分成splits，由于测试用的文件较小，所以每个文件为一个split，并将文件按行分割形成<key, value>对，这一步由MapReduce框架自动完成，其中偏移量(即key值)包括了回车所占的字符数。将分割好的<key, value>对交给用户定义的map方法进行处理，生成新的<key, value>对。得到map方法输出的<key, value>之后，Mapper会将它们按照key值进行排序，并执行Shuffing过程，将key值相同的value值累加，得到Mapper的最终输出结果。

Reducer先对Mapper接收的数据进行排序，再交由用户自定义的reduce方法进行处理，得到新的<key, value>对，并作为WordCount的输出结果。

通过上面的介绍，已经对MapReduce实现WordCount有了一定的理解，下面来介绍一下map和reduce输入和输出的具体内容。

map中输入和输出的内容

Map输入：0 Deer Bear River

Map输入：17 Car Car River

Map输入：32 Deer Car Bear

Map输出：Deer 1 Bear 1 River 1

Map输出：Car 1 Car 1 River 1

Map输出：Deer 1 Car 1 Bear 1

reduce中输入和输出的内容

Reduce输入：Bear 1 1

Reduce输入：Car 1 1 1

Reduce输入：Deer 1 1

Reduce输入：River 1 1

Reduce输出：Bear 2

Reduce输出：Car 3

Reduce输出：Deer 2

Reduce输出：River 2

### WordCount代码实现

Map任务阶段的目标：从文件里，按行读取数据，一行一行的进行处理。利用MR框架编写Map阶段代码： 写一个类继承org.apache.hadoop.mapreduce.

Mapperr<KEYIN, VALUEIN, KEYOUT, VALUEOUT>；定义Mapper的4个泛型类型；重写map()方法，在map方法里编写map任务代码；map函数()参数说明：之前曾提到，一个InputSpilt会对应一个Map任务，在InputSpilt里，实际就包含着待处理的文件数据。比如现在要处理的words.txt数据，MR框架会按行进行处理，并形成一个key/value对。默认的情况下，key是一个长整型，表示行的行首的偏移量。value是一个文本类型，代表每一行数据。context:我们利用context 输出我们自定义的输出map<E,V>。Mapper<LongWritable, Text,

Text, LongWritable>，Mapper中4个参数的含义：第一个参数LongWritable的含义：输入key类型，行首偏移量；第二个参数LongWritable的含义：输入value类型，读取到的一行文本内容；第三个参数Text的含义：输出key类型；第四个参数 LongWritable的含义：输出value类型。最后需要牢记的是：每有一行数据，就会调用一次map()方法。

MapTask阶段代码示例：



Reduce任务阶段的目标：对多个map任务的输出进行合并、排序。利用MR框架编写Map阶段代码：写一个类继承org.apache.hadoop.mapreduce.Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT>；



需要指定Reducer泛型类型，例如： Reducer<Text, LongWritable, Text, LongWritable>，Reducer中4个参数的含义：第一个参数Text的含义：输入key的类型；第二个参数 LongWritable的含义：输入value集合中元素的类型； 第三个参数Text的含义： 输出key的类型；第四个参数 LongWritable的含义：输出value的类型。接着重写reduce()方法，在reduce方法里编写reduce任务代码。最后需要牢记的是：一组相同的key，调用一次reduce。相当于一次方法的执行，计算一个key对应value中的值。

ReduceTask阶段代码示例：



java程序函数入口是主函数，MR程序也是通过java语言编写的，想要运行MR程序，所以需要创建一个类，在main方法写驱动程序,把Map函数和Reduce函数组织在一起。创建一个对象把Map和Reduce都放在这个对象中,我们把这个对象称作Job。Job提交器是yarn集群中的一个客户端，它负责将mr程序需要的信息全部封装成一个配置文件里面，然后连同我们mr程序所在的一个jar包。一起提交给yarn，由yarn去启动mr程序。

Driver任务阶段代码示例：



程序的运行结果，如图3-19所示：

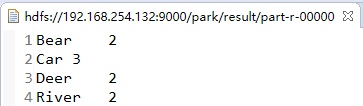


图3-19

## 求平均值案例

### 案例分析

对输入文件中数据进行运算学生平均成绩。输入文件中的每行内容均为一个学生的姓名和他相应的成绩。要求在输出中每行有两个间隔的数据，其中，第一个代表学生的姓名，第二个代表其平均成绩。

Map处理的一个纯文本文件，文件中存放的数据时每一行表示一个学生的姓名和他相应一科成绩。Mapper处理的数据是由InputFormat分解过的数据集，其中 InputFormat的作用是将数据集切割成小数据集InputSplit，每一个InputSplit将由一个Mapper负责处理。此外，InputFormat中还提供了一个RecordReader的实现，并将一个InputSplit解析成<key,value>对提供给了map函数。InputFormat的默认值是TextInputFormat，它针对文本文件，按行将文本切割成InputSlit，并用 LineRecordReader将InputSplit解析成<key,value>对，key是行在文本中的位置，value是文件中的一行。Map的结果会通过partion分发到Reducer，Reducer做完Reduce操作后，将通过以格式OutputFormat输出。

Mapper最终处理的结果对<key,value>，会送到Reducer中进行合并，合并的时候，有相同key的键/值对则送到同一个 Reducer上。

Reducer是所有用户定制Reducer类的基础，它的输入是key和这个key对应的所有value的一个迭代器，同时还有 Reducer的上下文。Reduce的结果由Reducer.Context的write方法输出到文件中。

此案例中的数据：

tom 69

tom 84

tom 69

jary 89

jary 90

jary 82

rose 78

rose 80

rose 82

输出的结果：

tom 74

Jary 87

rose 80

### 平均值案例代码实现

Map任务阶段的目标：从文件里，按行读取数据，一行一行的进行处理。

每有一行数据，就会调用一次map()方法。

MapTask阶段代码示例：



Reduce任务阶段的目标：对多个map任务的输出进行合并、排序。利用

一组相同的key，调用一次reduce。相当于调用一次，计算一个key对应的个数。

ReduceTask阶段代码示例：





java程序函数入口是主函数，MR程序也是通过java语言编写的，想要运行MR程序，所以需要创建一个类，在main方法写驱动程序,把Map函数和Reduce函数组织在一起。创建一个对象把Map和Reduce都放在这个对象中,我们把这个对象称作Job。Job提交器是yarn集群中的一个客户端，它负责将mr程序需要的信息全部封装成一个配置文件里面，然后连同我们mr程序所在的一个jar包。一起提交给yarn，由yarn去启动mr程序。

Driver任务代码示例：



程序的运行结果，如图3-20所示：



图3-20

## MapReduce执行job过程

通过以上Mapreduce案例分析，下面介绍MapReduce执行job的详细过程。如图3-21所示：

计算机生成了可选文字:
MapReduce 
1 ： rupj0b 
」 0 匯 《 i&it 
program 
client WM 
client node 
3 ： job 。 
旧 到 u 凸 
2:get new “ ． 
4:subnutjob 
tri 
input 邓 s 
Job 丨 
5 ： initialize job 
ßbtrackernode 
Shared 
rile5ystem ； 、 
& retrierejob 
力 № b 翮 t 
TaskTncÅer 
childJVM 
Child 
1 上 tun ： 
備 叩 Ta 嫩 
ReduæTask 
tasktrader node 

图3-21

1.run job阶段。收集job环境参数阶段，比如conf的参数设置，比如用户自定义的Mapper类和Reducer类以及相应的输出kv类型，并且检测是否正确。此外会检查输入路径和输出路径是否合法，并且检查是否具有权限。

2.get new jobID。jobID是全局唯一的，用来标识一个mrJob，我们可以根据jobID查询一个job，或kill一个job。并返回给JobClient。

3.JobClient把job的运算资源上传到HDFS上，（运算资源指定的是：比如程序代码，文件的切片数量）。

4.submit job。jobClient通知JobTracker可以启动执行任务了，并且把运算资源存放的路径告诉给JobTracker。

5.6.7 jobTrakcer通过文件切片数量得到Map任务数量。然后TaskTracker通过心跳去领任务。这里的策略是数据本地化策略，目的避免网络读取数据，节省带宽。Map任务为了确保读取的行是完整一行，不可避免的会发生网络传输数据（这个数据量很小）。

TaskTracker 去下载任务的计算资源。从而完整任务的计算。隐藏的思想： 移 动的是运算，而不是数据。

9.10启动jvm线程来处理一个Map任务或Reduce任务。

# MapReduce的序列化机制

## 序列化机制概述

序列化是指将结构化对象转化成为字节流以便在网络上传输或写到磁盘中进行永久存储的过程。反序列化是指将字节流转化成结构化对象的逆过程。

Hadoop在节点间的内部通讯使用的是RPC，RPC协议把消息翻译成二进制字节流发送到远程节点，远程节点再通过反序列化把二进制流转成原始的信息。也就是说传递的消息内容是需要经过Hadoop特定的序列化与反序列化操作的，因此，才需要使用Hadoop提供的数据类型，当然，如果想要自定义MR程序中key-value的数据类型，则需要实现相应的接口，如Writable、WritableComparable接口。

在编写MapReduce程序时，对于MapReduce的输入输出数据（key-value），只能使用Hadoop提供的数据类型，而不能使用Java本身的基本数据类型，比如，如果数据类型为long，那么在编写MR程序时，对应Hadoop的数据类型则为LongWritable。

序列化在分布式数据处理的两大领域经常出现：进程间通信和永久存储。

在Hadoop中，系统中多个节点上进程间的通信是通过“远程过程调用”(remote procedure call , RPC) 实现的。RPC协议将消息序列化成二进制流后发送到远程节点，远程节点接着将二进制流反序列化为原始消息。通常情况下，RPC序列化的特点如下：

紧凑：紧凑格式能充分利用网络带宽（数据中心最稀缺的资源）；快速：进程间通信形成了分布式系统的骨架， 所以需要尽量减少序列化和反序列化的性能开销，这是最基本的；

可扩展：为了满足新的需求，协议不断变化，所以客户端和服务器的过程中，需要直接引进相应的协议。例如，需要能够在方法的过程中增添新的参数，并且新的服务器需要能够接受来自客户端的老格式的消息(无新增的参数)；

互操作：对于某些系统来说，希望能支持以不同语言写的客户端与服务器交互，所以需要设计一种特定的格式来满足这一需求。

表面上看， 序列化框架对选用于数据持久存储的数据格式应该会有不同的要求。逼近，RPC的存活时间不到存储的数据1秒钟，永久存储的数据却可能在写到磁盘若干年后才会被读取。这样看来，对数据永久存储而言，RPC序列化格式的4大理想属性非常重要。我们希望存储格式比较紧凑（进而高效的使用存储空间）、快速（读/写数据的额外开销比较小）、可扩展（可以透明地读取老格式的数据）且可以互操作（以不同的语言读/写永久的数据）。

Hadoop使用的是自己的序列化格式Writable，它绝对紧凑、速度快，不太容易用java以外的语言进行扩展或使用。因为Writable是Hadoop的核心（大多数MapReduce程序都会为键和值使用它），所以在接下来的三个小节中，进行深入探讨，然后再从总体上看看序列化框架和Avro，后者是一个克服了Writable部分不足的序列化系统。

## 流量统计案例

通过前面的介绍，对序列化和反序列化都有了一定的了解，下面来通过一个案例深入了解一下序列化的内容。此案例是计算每个人的总流量。案例原始文件中的内容是：

手机号码 城市 姓名 流量

13877779999 bj zs 2145

13766668888 sh ls 1028

13766668888 sh ls 9987

13877779999 bj zs 5678

13544445555 sz ww 10577

13877779999 sh zs 2145

13766668888 sh ls 9987

输出的内容是：

13544445555 ww 10577

13766668888 ls 21002

13877779999 zs 9968

如果将Mapper传递出的参数拼成字符串并转换成Text这种方法做，后续还需将字符串split、类型转换、还需要记着原来的顺序等，开发时很麻烦，所以将传递参数封装到一个自定义类的对象中。

写一个Bean实现Writable接口，实现其中的write和readFields方法，将数据写入到二进制流中，从二进制数据流中读取数据。即可以完成序列化和反序列化，注意这两个方法中属性处理的顺序和类型，此后这个类的对象就可以用于MapReduce了。

JavaBean代码示例：





MapTask阶段代码示例：

在Map任务阶段中从文件里，按行读取数据，一行一行的进行处理。Map将输出电话号码以及FlowBean封装的对象。



ReduceTask阶段代码示例：

对多个map任务的输出进行合并、排序。Reduce端将输出电话号码，名字以及流量的总和。



Driver任务代码示例：

java程序函数入口是主函数，MR程序也是通过java语言编写的，想要运行MR程序，所以需要创建一个类，在main方法写驱动程序,把Map函数和Reduce函数组织在一起。创建一个对象把Map和Reduce都放在这个对象中,我们把这个对象称作Job。Job提交器是yarn集群中的一个客户端，它负责将mr程序需要的信息全部封装成一个配置文件里面，然后连同我们mr程序所在的一个jar包。一起提交给yarn，由yarn去启动mr程序。



程序的运行结果，如图4-1所示：

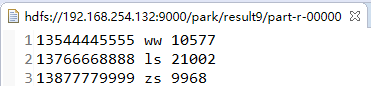


图4-1

# MR分区、排序、合并及shuffle过程

## 分区

### 分区的概述

分区操作是将map的结果按照规则分发到不同reduce中进行处理，从而按照分区得到多个输出结果。partitioner分区主要是用来提高效率的，例如从全国基站的数据中查找北京基站的数据，如果计算时不分区全国的数据都放在一起，查询的时候就相当于全表扫描效率非常低，如果在第一次进行MapReducer计算的时候按照省市进行分区，每个城市的基站数据都存储在对应的每个文件，那么下次再进行查询的时候直接从北京分区里直接查找，效率很高。分区的依据是具体业务需求，可以按照省市分区，时间进行分区等。Hadoop 默认的reduce数量是一个，即默认的分区是1个，假设有多个分区，Hadoop默认是用简单哈希散列算法，计算Mapper输出的key值， 然后进行分区。这样能够确保同一key落到同一分区，有个分区（reduce），就会几个结果文件，先分区，再按key合并。

partitioner是在map阶段完成后执行的。然后将分好区的数据传输到reduce端，也就是由Partitioner来决定每条记录应该送往哪个reducer节点。mapreduce中默认的分区是HashPartition类。核心代码如下：



上面的getPartition函数的作用：获取key的哈希值，使用key的哈希值对reduce任务数求模，这样做的目的是可以把(key,value)对均匀的分发到各个对应的reduce task节点上，达到reduce task节点的负载均衡。

在MapReduce中，分片、分区、排序和分组（Group）的关系图，如图5-1所示：

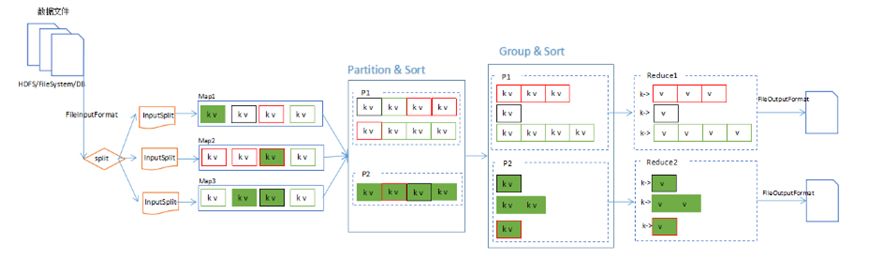


图 5-1

Map的结果，会通过partition分发到Reducer上。Mapper的结果，可能送到Combiner做合并，Combiner在系统中并没有自己的基类，而是用Reducer作为Combiner的基类，他们对外的功能是一样的，只是使用的位置和使用时的上下文不太一样而已。Mapper最终处理的键值对<key, value>，是需要送到Reducer去合并的，合并的时候，有相同key的键/值对会送到同一个Reducer那里。哪个key到哪个Reducer的分配过程，是由Partitioner规定的。它只有一个方法，

getPartition(Text key, Text value, int numPartitions)。

输入是Map的结果对<key, value>和Reducer的数目，输出则是分配的Reducer（整数编号）。就是指定Mapper输出的键值对到哪一个reducer上去。系统指定的Partitioner是HashPartitioner，它以key的Hash值对Reducer的数目取模，得到对应的Reducer。这样保证如果有相同的key值，肯定被分配到同一个reduce上。如果有N个reducer，编号就为0,1,2,3……(N-1)。

得到map给的记录后，他们该分配给哪些reducer来处理呢？Hadoop采用的默认的派发方式是根据散列值来派发的，但是实际中，这并不能很高效或者按照我们要求的去执行任务。例如，经过partition处理后，一个节点的reducer分配到了20条记录，另一个却分配到了10W万条，试想，这种情况效率如何。又或者，我们想要处理后得到的文件按照一定的规律进行输出，假设有两个reducer，我们想要最终结果中part-00000中存储的是"h"开头的记录的结果,part-00001中存储其他开头的结果，这些默认的partitioner是做不到的。所以需要我们自己定制partition来根据自己的要求，选择记录的reducer。自定义partitioner很简单，只需要自定义一个类，并且继承Partitioner类，重写其getPartition方法就好了，在使用的时候通过调用Job的setPartitionerClass指定一下即可。代码如下：

指定分区的类：



通过调用Job的setPartitionerClass指定一下：



Partitioner是partitioner的基类，如果需要定制partitioner也需要继承该类。Partitioner组件通过让Map对Key 进行分区，从而将不同分区的Key交由不同的Reduce处理。Partition属于map端，分区的总数与reduce的任务数相同。

## 分区案例分析

案例的前期准备工作：将文件上传到HDFS上、新建一个MapReduce Project的项目、新建一个Mapper的类、新建一个Reducer的类、新建一个Driver的类、新建一个FlowBean的类，继承Writable接口、新建一个FlowCountPartition的类，继承Partitioner接口。

想要实现自定义分区，需要继承Partitioner类，并重写getPartition（）方法。根据用户自定义的分区，把输出map的value值（本例中是一个一个的FlowBean对象），根据用户定义的条件放到对应的分区里。

比如如下的数据：

13877779999 bj zs 2145

13766668888 sh ls 1028

13766668888 sh ls 9987

13877779999 bj zs 5678

13544445555 sz ww 10577

13877779999 sh zs 2145

13766668888 sh ls 9987

我们如果想按地区分区，bj,sh,sz 三个区，

bj区：13877779999 bj zs 2145，13877779999 bj zs 5678

sh区：13766668888 sh ls 1028，13766668888 sh ls 9987，13877779999 sh zs 2145，13766668888 sh ls 9987

sz区：13544445555 sz ww 10577

分完区之后，在每个区里，会按照输出map的key值整理成如下形式，然后发送给reduce端：

bj区：[zs,{13877779999 bj zs 2145},{13877779999 bj zs 2145}]

sh区：[ls,{13766668888 sh ls 1028},{13766668888 sh ls 9987},{13766668888 sh ls 9987}],[zs,{13877779999 sh zs 2145}]

sz区：[ww,{13544445555 sz ww 10577}]

然后reduce端拿到数据后，根据用户的自定义代码进行每个区里的数据合并。

### 分区案例实现

Pojo实体类

MR中自定义数据类型需要实现Writable接口来序列化，java中原生序列化实现的是Serializable接口，在Hadoop并没有使用。写一个Bean实现Writable接口，实现其中的write和readFields方法，将数据写入到二进制流中，从二进制数据流中读取数据。即可以完成序列化和反序列化，注意这两个方法中属性处理的顺序和类型，此后这个类的对象就可以用于MapReduce了。

JavaBean代码示例:





Partitioner会将数据发往不同的reducer，这就要求reducer的数量应该大于等于Partitioner的数量，如果少于则在执行的过程中会报错。

Partition代码示例：

自定义分区Partitioner<KEY, VALUE> 其中key是mapper输出key的类型 value是mapper输出value的数据类型。



Map任务阶段的目标：从文件里，按行读取数据，一行一行的进行处理。

Map将输出电话号码以及FlowBean封装的对象。

MapTask阶段代码示例（姓名为输出key值）：



Reduce任务阶段的目标：对多个map任务的输出进行合并。Reduce端将输出电话号码，名字以及流量的总和。

ReduceTask代码示例（姓名为输出key值）：



java程序函数入口是主函数，MR程序也是通过java语言编写的，想要运行MR程序，所以需要创建一个类，在main方法写驱动程序,把Map函数和Reduce函数组织在一起。创建一个对象把Map和Reduce都放在这个对象中,我们把这个对象称作Job。Job提交器是yarn集群中的一个客户端，它负责将mr程序需要的信息全部封装成一个配置文件里面，然后连同我们mr程序所在的一个jar包。一起提交给yarn，由yarn去启动mr程序。

Driver代码示例：





程序的运行结果，如图5-2，5-3，5-4所示：



图5-2

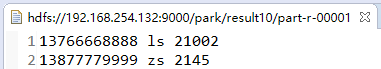


图5-3



图5-4

## 排序

### 排序的概述

排序是将一批无序的记录（数据）重新排列成按关键字有序的记录序列的过程。MapReduce的排序是默认按照Key排序的，key实现了WritableComparator接口compare to方法进行排序，在map输出的时候，会进行分片，在片内再对key进行排序。排序是为后一阶段的reduce的排序做好基础，以便归并排序的时候更快。

数据在Map之后会做排序（从内存缓冲区到磁盘的时候会做sort），所以Map操作只需要把数据直接写出即可，最后在本地做数据合并的时候也是会有排序的，详细可以参考MapReduce的过程，但是需要注意的是，因为我们需要进行的是数字的排序，所以在Map输出时，key的类型应该是Int类型才能按照数字的方式进行排序，如果是Text文本的话，那么是按照字典顺序来进行排序的（也就是先比较字符串中的第一个字符，如果相同再比较第二个字符，以此类推），而不是按照数字进行排序。

如何使用mapreduce来做全排序？最简单的方法就是使用一个partition，因为一个partition对应一个reduce的task，然而reduce的输入本来就是对key有序的，所以很自然地就产生了一个全排序文件。但是这种方法在处理大型文件时效率极低，因为一台机器必须处理所有输出文件，从而完全丧失了mapreduce所提供的并行架构的优势。

如果是分多个partition呢？则只要确保partition是有序的就行了。首先创建一系列排好序的文件；其次，串联这些文件（类似于归并排序）；最后得到一个全局有序的文件。比如有1000个1-10000的数据，跑10个ruduce任务，如果进行partition的时候，能够将在1-1000的数据分配到第一个reduce中，1001-2000的数据分配到第二个reduce中，以此类推。即第n个reduce所分配到的数据全部大于第n-1个reduce中的数据。这样，每个reduce出来之后都是有序的了，我们只要concat所有的输出文件，变成一个大的文件，就都是有序的了。

这时候可能会有一个疑问，虽然各个reduce的数据是按照区间排列好的，但是每个reduce里面的数据是乱序的啊？当然不会，不要忘了排序是MapReduce的天然特性----在数据达到reducer之前，mapreduce框架已经对这些数据按key排序了。

对于输入文件，会进行分片，对于一个split，有一个map任务进行处理，每个Map在内存中都有一个缓存区，map的输出结果会先放到这个缓冲区中,在缓冲区中，会进行预排序（即sort和comibner），以提高效率。Map执行过后，在数据进入reduce操作之前，数据将会按照key进行排序，利用这个特性可以实现大数据场景下排序的需求。Sort阶段就是对Map端输出的key进行排序。

## 排序案例分析

项目的前期准备：新建一个mapper类、新建一个reducer类、新建一个driver类。新建一个AuthPartitioner类，继承Partitioner类。

需求说明：有这样一组数字，要求利用3个reduce来处理，并且生成的三个结果文件，是整体有序的。

源数据：

82 239 231

23 22 213

123 232 124

213 3434 232

4546 565 123

231 231

2334 231

1123 5656 657

12313 4324 213

123 2 232 32

343 123 4535

12321 3442 453

1233 342 453

1231 322 452

232 343 455

3123 3434 3242

生成的三个结果文件：



实现思路：如果是一个reduce，那肯定就是全排序。但如果只单纯指定3个reduce数量，Hadoop会默认根据Key的hash进行分区，这样不会出现三个结果文件是全排序的情况。所以，需要自定义分区，在自定义分区里，根据数据的大小来对数据进行分区。

通过自定义分区，将源数据分区到0-100，101-1000，大于1000三个分区内。

### 排序案例实现

自定义分区代码示例：



Map任务阶段的目标：从文件里，按行读取数据，一行一行的进行处理。每有一行数据，就会调用一次map()方法。代码如下：

MapTask阶段代码示例：



Reduce任务阶段的目标：对多个map任务的输出进行合并、排序。一组相同的key，调用一次reduce。相当于调用一次，计算一个key对应的个数。代码如下：

ReduceTask阶段代码示例：



java程序函数入口是主函数，MR程序也是通过java语言编写的，想要运行MR程序，所以需要创建一个类，在main方法写驱动程序,把Map函数和Reduce函数组织在一起。创建一个对象把Map和Reduce都放在这个对象中,我们把这个对象称作Job。Job提交器是yarn集群中的一个客户端，它负责将mr程序需要的信息全部封装成一个配置文件里面，然后连同我们mr程序所在的一个jar包。一起提交给yarn，由yarn去启动mr程序。

Driver任务代码示例：



程序的运行结果，如图5-5，5-6，5-7所示：

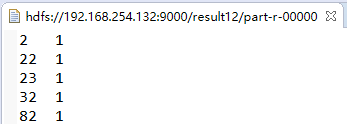


图5-5

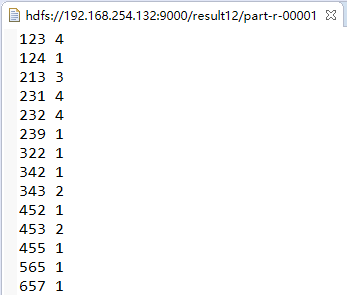


图5-6

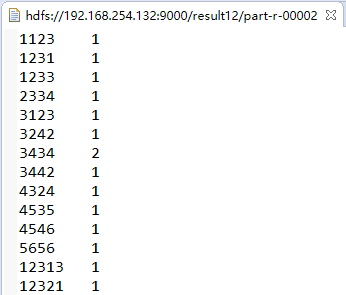


图5-7

## 合并

### 合并的概述

合并（combine）是把同一个key的键值对合并在一起，可以自定义的。combine函数把一个map函数产生的<key,value>对（多个key,value）合并成一个新的<key2,value2>。将新的<key2,value2>作为输入到reduce函数中这个value2亦可称之为values，因为有多个。这个合并的目的是为了减少网络传输。

很多MapReduce程序受限于集群上可用的带宽，所以它会尽力最小化需要在map和reduce任务之间传输的中间数据。Hadoop允许用户声明一个combiner 函数来处理map的输出，同时把自己对map的处理结果作为reduce的输入。因为combiner 函数本身只是一种优化，Hadoop并不保证对于某个map输出，这个方法会被调用多少次。换句话说，不管combiner 函数被调用多少次，对应的reduce输出结果都应该是一样的。  
　　下面我们以《权威指南》的例子来加以说明，假设1950年的天气数据读取是由两个map完成的，其中第一个map的输出如下：  
　　(1950, 0)  
　　(1950, 20)  
　　(1950, 10)  
第二个map的输出为：  
       (1950, 25)

（150,15）  
而reduce得到的输入为：(1950, [0, 20, 10, 25, 15])， 输出为：(1950, 25)  
　　由于25是集合中的最大值，我们可以使用一个类似于reduce 函数的combiner函数来找出每个map中输出的最大值，这样的话，reduce的输入就变成了：  
　　(1950, [20, 25])  
 各个函数对温度值的处理过程可以表示如下：

max(0, 20, 10, 25, 15) =max(max(0, 20, 10), max(25, 15)) = max(20, 25) = 25。

注意：并不是所有的函数都拥有这个属性的（有这个属性的函数我们称之为commutative和associative），例如，如果我们要计算平均温度，就不能这样使用combiner函数，因为mean(0, 20, 10, 25, 15) =14，而mean(mean(0, 20, 10),mean(25, 15)) = mean(10, 20) = 15。  
　　combiner 函数并不能取代reduce 函数（因为仍然需要reduce 函数处理来自不同map的带有相同key的记录）。但是他可以帮助减少需要在map和reduce之间传输的数据，就为这一点combiner 函数就值得考虑使用。

每一个MapperTask可能会产生大量的输出，combiner的作用就是在MapperTask端对输出先做一次合并，以减少传输到reducerTask的数据量。combiner是实现在Mapper端进行key的归并，combiner具有类似本地的reducer功能。如果不用combiner，那么，所有的结果都是reducer完成，效率会相对低下。使用combiner，先完成在Mapper的本地聚合，从而提升速度。

使用combiner合并操作需要新建一个combiner类，在Driver中代指定combiner类，代码如下：





### WordCount案例优化

合并的目的是减少Reduce端迭代的次数， combiner是实现Mapper端进行key的归并， combiner具有类似本地的reduce功能。如果不用combiner,那么所有的结果都是reduce完成，效率会很低。使用combiner先做合并，然后发往reduce。

MapTask阶段代码示例：

Map任务阶段的目标：从文件里，按行读取数据，一行一行的进行处理。每有一行数据，就会调用一次map()方法。



ReduceTask阶段代示例：

Reduce任务阶段的目标：对多个map任务的输出进行合并、排序。一组相同的key，调用一次reduce。相当于调用一次，计算一个key对应的个数。



Combiner代码示例 实现combiner合并：



Driver阶段代码示例：

java程序函数入口是主函数，MR程序也是通过java语言编写的，想要运行MR程序，所以需要创建一个类，在main方法写驱动程序,把Map函数和Reduce函数组织在一起。创建一个对象把Map和Reduce都放在这个对象中,我们把这个对象称作Job。Job提交器是yarn集群中的一个客户端，它负责将mr程序需要的信息全部封装成一个配置文件里面，然后连同我们mr程序所在的一个jar包。一起提交给yarn，由yarn去启动mr程序。

在driver中指定Combiner类：job.setCombinerClass(WCCombiner.class);



程序的运行结果，如图5-8所示：

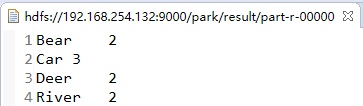


图5-8

## Shuffle过程

 Shuffle过程是MapReduce的核心，也被称为奇迹发生的地方。要想理解MapReduce， Shuffle是必须要了解的。Shuffle的正常意思是洗牌或弄乱，可能大家更熟悉的是Java API里的Collections.shuffle(List)方法，它会随机地打乱参数list里的元素顺序。如果你不知道MapReduce里Shuffle是什么，那么如图5-9所示：



图5-9

 这张是官方对Shuffle过程的描述。但我可以肯定的是，单从这张图你基本不可能明白Shuffle的过程，因为它与事实相差挺多，细节也是错乱的。后面我会具体描述Shuffle的事实情况，所以这里你只要清楚Shuffle的大致范围就成－怎样把map task的输出结果有效地传送到reduce端。也可以这样理解， Shuffle描述着数据从map task输出到reduce task输入的这段过程。

Shuffle过程，也称Copy阶段。reduce task从各个map task上远程拷贝一片数据，并针对某一片数据，如果其大小超过一定的阀值，则写到磁盘上，否则直接放到内存中。

Hadoop的集群环境，大部分的map task和reduce task是执行在不同的节点上的，那么reduce就要取map的输出结果。那么集群中运行多个Job时，task的正常执行会对集群内部的网络资源消耗严重。虽说这种消耗是正常的，是不可避免的，但是，我们可以采取措施尽可能的减少不必要的网络资源消耗。另一方面，每个节点的内部，相比于内存，磁盘IO对Job完成时间的影响相当的大。所以，从以上分析，shuffle过程的基本要求：完整地从map task端拉取数据到reduce task端，在拉取数据的过程中，尽可能地减少网络资源的消耗，尽可能地减少磁盘IO对task执行效率的影响。那么，Shuffle的设计目的就要满足以下条件：保证拉取数据的完整性，尽可能地减少拉取数据的数据量，尽可能地使用节点的内存而不是磁盘。

### Map阶段

从运算效率的出发点，map输出结果优先存储在map节点的内存中。每个map task都有一个内存缓冲区，存储着map的输出结果，当缓冲区达到阈值时，需要将缓冲区中的数据以一个临时文件的方式存到磁盘，当整个map task结束后再对磁盘中这个map task所产生的所有临时文件做合并，生成最终的输出文件。最后，等待reduce task来拉取数据。当然，如果map task的结果不大，能够完全存储到内存缓冲区，且未达到内存缓冲区的阀值，那么就不会有写临时文件到磁盘的操作，也不会有后面的合并。

详细过程如下：map task任务执行，输入数据的来源是：HDFS的block。当然在mapreduce概念中，map task读取的是split分片。split与block的对应关系：一对一（默认）。此处有必要说明一下block与splitblock（物理划分）：文件上传到HDFS，就要划分数据成块，这里的划分属于物理的划分，块的大小可配置（默认：Hadoop 1.0为64M，Hadoop 2.0为128M）可通过 dfs.block.size配置。为保证数据的安全，block采用冗余。机制：默认为3份，可通过dfs.replication配置。注意：当更改块大小的配置后，新上传的文件的块大小为新配置的值，以前上传的文件的块大小为以前的配置值。split（逻辑划分）： Hadoop中split划分属于逻辑上的划分，目的只是为了让map task更好地获取数据。split是通过Hadoop中的InputFormat接口中的getSplit（）方法得到的。

map执行后，得到key/value键值对。接下来的问题就是，这些键值对应该交给哪个reduce做？注意：reduce的个数是允许用户在提交job时，通过方法设置的！MapReduce提供partitioner接口解决上述问题。默认操作是：对key hash后再以reduce task数量取模，返回值决定着该键值对应该由哪个reduce处理。这种默认的取模方式只是为了平均reduce的处理能力，防止数据倾斜，保证负载均衡。 如果用户自己对Partition有需求，可以自行定制并设置到job上。接下来，需要将key/value以及Partition结果都写入到缓冲区，缓冲区的作用：批量收集map结果，减少磁盘IO的影响。当然，写入之前，这些数据都会被序列化成字节数组。而整个内存缓冲区就是一个字节数组。这个内存缓冲区是有大小限制的，默认100MB。当map task的输出结果很多时，就可能撑爆内存。需将缓冲区的数据临时写入磁盘，然后重新利用这块缓冲区。从内存往磁盘写数据被称为Spill(溢写)，由单独线程完成，不影响往缓冲区写map结果的线程。溢写比例：spill.percent(默认0.8)。当缓冲区的数据达到阀值，溢写线程启动，锁定这80MB的内存，执行溢写过程。剩下的20MB继续写入map task的输出结果。当溢写线程启动后，需要对这80MB空间内的key做排序(Sort)。排序是mapreduce模型的默认行为，也是对序列化的字节做的排序。排序规则：字典排序！map task的输出结果写入内存后，当溢写线程未启动时，对输出结果并没有做任何的合并。从官方图可以看出，合并是体现在溢写的临时磁盘文件上的，且这种合并是对不同的reduce端的数值做的合并。所以溢写过程一个很重要的细节在于， 如果有很多个key/value对需要发送到某个reduce端，那么需要将这些键值对拼接到一块，减少与partition相关的索引记录。如果client设置过Combiner，其会将有相同key的key/value对的value加起来，减少溢写到磁盘的数据量。注意：这里的合并并不能保证map结果中所有的相同的key值的键值对的value都合并了，它合并的范围只是这80MB，它能保证的是在每个单独的溢写文件中所有键值对的key值均不相同！溢写生成的临时文件的个数随着map输出结果的数据量变大而增多，当整个map task完成，内存中的数据也全部溢写到磁盘的一个溢写文件。也就是说，不论任何情况下，溢写过程生成的溢写文件至少有一个！但是最终的文件只能有一个，需要将这些溢写文件归并到一起，称为merge。merge是将所有的溢写文件归并到一个文件，结合上面所描述的combiner的作用范围，归并得到的文件内键值对有可能拥有相同的key，这个过程如果client设置过Combiner，也会合并相同的key值的键值对，如果没有，merge得到的就是键值集合，如{“aaa”, [5, 8, 2, …]}。如果在最终合并时，被合并的文件大于等于3个，则合并完会再执行一次Combiner，否则不会。注意：combiner的合理设置可以提高效率，但是如果使用不当会影响效率！

### Reduce阶段

当mapreduce任务提交后，reduce task就不断通过RPC从JobTracker那里获取map task是否完成的信息，如果获知某台TaskTracker上的map task执行完成，Shuffle后面的过程就开始启动。其实呢，reduce task在执行之前的工作就是：不断地拉取当前job里每个map task的最终结果，并对不同地方拉取过来的数据不断地做merge，也最终形成一个文件作为reduce task的输入文件。Copy过程，简单地拉取数据。Reduce进程启动一些数据copy线程（Fether），通过HTTP方式请求map task所在的TaskTracker获取map task的输出文件。因为map task早已结束，这些文件就归TaskTracker管理在本地磁盘。Merge过程，这里的merge如map端的merge操作，只是数组中存放的是不同map端copy过来的数值。Copy过来的数据会先放入内存缓冲区中，这里缓冲区的大小要比map端的更为灵活，它是基于JVM的heap size设置，因为shuffler阶段reducer不运行，所以应该把绝大部分的内存都给shuffle用。merge的三种形式：内存到内存、内存到磁盘、磁盘到磁盘。默认情况下，第一种形式不启用。当内存中的数据量达到一定的阀值，就启动内存到磁盘的merge。与map端类似，这也是溢写过程，当然如果这里设置了Combiner，也是会启动的，然后在磁盘中生成了众多的溢写文件。第二种merge方式一直在运行，直到没有map端的数据时才结束，然后启动第三种磁盘到磁盘的merge方式生成最终的那个文件。reducer的输入文件。不断地merge后，最后会生成一个“最终文件”。这个最终文件可能在磁盘中也可能在内存中。当然我们希望它在内存中，直接作为reducer的输入，但默认情况下，这个文件是存放于磁盘中的。当reducer的输入文件已定，整个shuffle才最终结束。然后就是reducer执行，把结果存放到HDFS上。

Reducer通过Http方式得到输出文件的分区。NodeManager为分区文件运行Reduce任务。复制阶段把Map输出复制到Reducer的内存或磁盘。一d旦Map任务完成，Reduce就开始复制输出。排序阶段合并map输出，然后走Reduce阶段。

### Shuffle过程的详细总结

1.Map Task的输出k v，一开始会进入溢写缓冲区中，对数据做处理，比如分区、排序等操作。

2.有几个Map Task，就有几个对应的溢写缓冲区。

3.溢写缓冲区默认是100MB，溢写阈值：0.8。（都可通过配置文件调节）

4.当缓冲区中的数据达到溢写阈值时，会发生Spill溢写过程。把内存中数据溢写到磁盘的文件上。

5.第4步生成的文件，称为Spill溢写文件。

6.每一个Spill文件里的数据都是已分好区，且排好序的。

7.当Spill过程结束之后，会发生Merge过程。目的是将多个Spill合成最后的结果文件（Finaloutput）。

8.结果文件的是一个已分好区，且已排序的文件。

9.Spill和Merge过程不一定会发生。

10.如果发生了Spill过程，最后存留在溢写缓冲区里数据，会Flush到文件中。目的是确保数据都落到文件中。

11.如果发生了Spill过程，但不一定会发生Merge。即如果只有一个Spill文件，则此文件就是最后的结果文件。

12.从性能调优的角度，可以加入Combiner中间过程，会减少数据在溢写缓冲区的存储，间接减少了Spill次数，即减少了磁盘的I/O次数。

13.如果加入了Combiner中间过程，在溢写缓冲区的处理阶段是一定会发生的。但是在Merge过程中，也可能会发生。

14.Merge的Combiner不发生的条件：Spill文件的数量<3。

15.从性能调优的角度，可以适当增大溢写缓冲区的大小，可以减少Spill的溢写次数。要根据服务的硬件情况来调节。一般服务器内存：32GB或64GB。结合集群的：slave节点数量+Job数量+每个Job的MapTask数量。

16.溢写缓冲区也叫环写缓冲区（环形缓冲区），注意：溢写阈值的参数可调，但是不要调成100%。目的是为了避免产生阻塞。此外，环形缓冲区的好处是每个MapTask重复利用同一块内存地址空间，可以减少内存碎片的产生，提高内存使用率，而且从GC角度来看，可以减少full gc发生的次数。

建议：看GC回收算法以及GC收集器，《深度理解Java虚拟机：JVM高级特性与最佳实践》（第2版）看第二章。

17.可以开启Map Task的压缩机制，将最后的结果文件做压缩。好处可以减少网络数据的传输。

18.当Merge过程结束后，所有的Spill文件被删除。

19.有几个Map Task ，就有几个最后结果文件。

20.最后的结果文件存到服务节点的本地磁盘上。

21.虽然一个Map Task处理的切片数据是128MB(满的情况），但是不能凭输入的数据大小来判断map的输出大小，要根据实际的业务代码来判断。

22.Map Task的输出结果有两类收集器：①DirectMapOutputCollector 在没有reducer组件的情况下使用。

②MapOutputBuffer 在有reducer组件的情况收集，在这个类中，包含了Spill 、溢写缓冲区相关的对象。

22.当Map阶段接收，reduce会Fetch自己分区的数据。

23.reduce 的Fetch结束后，会进行Merge 和Sort。

24.Merge和Sort结束后，会发生reduce，按相同key聚合，形成key iterator传给开发者。

25.Fetch线程数默认是5个，此参数可以调节。一般的做法是让此线程数接近或等于map task 数量。达到并行抓取的目的。

## MapReduce的主要特点

### MapReduce的主要功能

1.数据划分和计算任务调度

系统自动将一个作业(Job)待处理的大数据划分为很多个切片，每个切片对应于一个计算任务(Task)，并自动调度计算节点来处理相应的任务。作业和任务调度功能主要负责分配和调度计算节点(Map节点或Reduce节点)，同时负责监控这些节点的执行状态，并负责Map节点执行的同步控制。

2.数据/代码互定位-数据本地化策略

为了减少数据通信，一个基本原则是本地化数据处理，即一个计算节点尽可能处理其本地磁盘上所分布存储的数据，这实现了代码向数据的迁移;当无法进行这种本地化数据处理时，再寻找其他可用节点并将数据从网络上传送给该节点(数据向代码迁移)，但将尽可能从数据所在的本地机架上寻找可用节点以减少通信延迟。

3.系统优化

为了减少数据通信开销，中间结果数据进入Reduce节点前会进行一定的合并处理（combiner中间过程）一个Reduce节点所处理的数据可能会来自多个 Map节点，为了避免Reduce计算阶段发生数据相关性，Map节点输出的中间结果需使用一定的策略进行适当的划分处理，保证相关性数据发送到同一个 Reduce节点（分区）;此外，系统还进行一些计算性能优化处理，如对最慢的计算任务采用多备份执行、选最快完成者作为结果（任务的推测执行机制）。

4.出错检测和恢复:

以低端商用服务器构成的大规模MapReduce计算集群中，节点硬件(主机、磁盘、内存等)出错和软件出错是常态，因此 MapReduce需要能检测并隔离出错节点，并调度分配新的节点接管出错节点的计算任务。同时，系统还将维护数据存储的可靠性，用多备份冗余存储机制提高数据存储的可靠性，并能及时检测和恢复出错的数据。

### MapReduce的主要技术特征

1.向“外”横向扩展，而非向“上”纵向扩展

即MapReduce集群的构建完全选用价格便宜、易于扩展的低端商用服务器，而非价格昂贵、不易扩展的高端服务器。对于大规模数据处理，由于有大量数据存储需要，显而易见，基于低端服务器的集群远比基于高端服务器的集群优越，这就是为什么MapReduce并行计算集群会基于低端服务器实现的原因。

2.失效被认为是常态

MapReduce集群中使用大量的低端服务器，因此，节点硬件失效和软件出错是常态，因而一个良好设计、具有高容错性的并行计算系统不能因为节点失效而影响计算服务的质量，任何节点失效都不应当导致结果的不一致或不确定性；任何一个节点失效时，其他节点要能够无缝接管失效节点的计算任务；当失效节点恢复后应能自动无缝加入集群，而不需要管理员人工进行系统配置。MapReduce并行计算软件框架使用了多种有效的错误检测和恢复机制，如节点自动重启技术，使集群和计算框架具有对付节点失效的健壮性，能有效处理失效节点的检测和恢复。

3.把处理向数据迁移

传统高性能计算系统通常有很多处理器节点与一些外存储器节点相连，如用存储区域网络（Storage Area，SAN Network）连接的磁盘阵列，因此，大规模数据处理时外存文件数据I/O访问会成为一个制约系统性能的瓶颈。为了减少大规模数据并行计算系统中的数据通信开销，代之以把数据传送到处理节点（数据向处理器或代码迁移），应当考虑将处理向数据靠拢和迁移。MapReduce采用了数据/代码互定位的技术方法，计算节点将首先尽量负责计算其本地存储的数据，以发挥数据本地化特点，仅当节点无法处理本地数据时，再采用就近原则寻找其他可用计算节点，并把数据传送到该可用计算节点。

4.顺序处理数据、避免数据随机访问

大规模数据处理的特点决定了大量的数据记录难以全部存放在内存，而通常只能放在外存中进行处理。由于磁盘的顺序访问要远比随机访问快得多，因此MapReduce主要设计为面向顺序式大规模数据的磁盘访问处理。为了实现面向大数据集批处理的高吞吐量的并行处理，MapReduce可以利用集群中的大量数据存储节点同时访问数据，以此利用分布集群中大量节点上的磁盘集合提供高带宽的数据访问和传输。

5.为应用开发者隐藏系统层细节

软件工程实践指南中，专业程序员认为之所以写程序困难，是因为程序员需要记住太多的编程细节（从变量名到复杂算法的边界情况处理），这对大脑记忆是一个巨大的认知负担，需要高度集中注意力；而并行程序编写有更多困难，如需要考虑多线程中诸如同步等复杂繁琐的细节。由于并发执行中的不可预测性，程序的调试查错也十分困难；而且，大规模数据处理时程序员需要考虑诸如数据分布存储管理、数据分发、数据通信和同步、计算结果收集等诸多细节问题。MapReduce提供了一种抽象机制将程序员与系统层细节隔离开来，程序员仅需描述需要计算什么（What to compute），而具体怎么去计算（How to compute）就交由系统的执行框架处理，这样程序员可从系统层细节中解放出来，而致力于其应用本身计算问题的算法设计。

6.平滑无缝的可扩展性

这里指出的可扩展性主要包括两层意义上的扩展性： 数据扩展和系统规模扩展性。理想的软件算法应当能随着数据规模的扩大而表现出持续的有效性，性能上的下降程度应与数据规模扩大的倍数相当；在集群规模上，要求算法的计算性能应能随着节点数的增加保持接近线性程度的增长。绝大多数现有的单机算法都达不到以上理想的要求；把中间结果数据维护在内存中的单机算法在大规模数据处理时很快失效；从单机到基于大规模集群的并行计算从根本上需要完全不同的算法设计。奇妙的是，MapReduce在很多情形下能实现以上理想的扩展性特征。多项研究发现，对于很多计算问题，基于MapReduce的计算性能可随节点数目增长保持近似于线性的增长。

# Sqoop介绍

sqoop是Apache 提供的工具

用于hdfs和关系型数据库之间数据的导出和导入

可以从hdfs导出数据到关系型数据库，也可以从关系型数据库导入数据到hdfs。

## 实现步骤

1.准备sqoop安装包，官网地址：http://sqoop.apache.org

2.配置jdk环境变量和Hadoop的环境变量。因为sqoop在使用是会去找环境变量对应的路径，从而完整工作。

3.sqoop解压即可使用（前提是环境变量都配好了）

4.需要将要连接的数据库的驱动包加入sqoop的lib目录下（本例中用的是mysql数据库）

5.利用指令操作sqoop

## Sqoop基础指令（在Sqoop的bin目录下执行下列指令）

|  |  |
| --- | --- |
| 说明 | 指令示例 |
| 查看mysql所有数据库 | sh sqoop list-databases --connect jdbc:mysql://192.168.234.21:3306/ -username root -password root    IMG_256 |
| 查看指定数据库下的所有表 | sh sqoop list-tables --connect jdbc:mysql://192.168.234.21:3306/hive -username root -password root |
| 关系型数据库==>hdfs  sh sqoop import -help（查看import的帮助指令） | 实现步骤：  1.现在mysql数据库的mydb1数据下建立一张tabx表，并插入测试数据  建表：create table tabx (id int,name varchar(20));  插入：insert into tabx (id,name) values (1,'aaa'),(2,'bbb'),(3,'ccc'),(1,'ddd'),(2,'eee'),(3,'fff');  2.进入到sqoop的bin目录下，执行导入语句  导入：  sh sqoop import --connect jdbc:mysql://192.168.234.21:3306/mydb1 --username root --password root --table tabx --target-dir '/sqoop/tabx' --fields-terminated-by '|' --split-by id; 说明：注意标红处对应的含义。  此外，--split-by id表示的是tabx表在导入hdfs时分区的原则。本例是按id字段切分。 如果不想分区，则把--split-by id 替换成 -m 1即可。  sh sqoop import --connect jdbc:mysql://192.168.234.21:3306/mydb1 --username root --password root --table tabx --target-dir '/sqoop/tabx' --fields-terminated-by '|' -m 1; |
| hdfs==>关系型数据库 | 执行：sh sqoop export --connect jdbc:mysql://192.168.234.243:3306/mydb1 --username root --password root --export-dir '/sqoop/tabx' --table taby -m 1 --fields-terminated-by '|' 注：sqoop只能导出数据，不能自动建表。所以在导出之前，要现在mysql数据库里建好对应的表 |