

# 大数据的概念

2018年1月21日 17:57

## 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。

目前对于大数据特点的总结是5V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

1024ZB=1YB

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 ) E-mail：全球所有电子邮件用户发出了2.04亿封电子邮件。
- ( 2 ) 搜索：全球最大的搜索引擎Google处理了200万次搜索请求。
- ( 3 ) 图片：图片分享网站Flickr的用户上传了3 125张新照片，2 000万张照片被浏览。
- ( 4 ) 音频：在Pandora音乐网站上，播放的音乐时长超过61 000小时。

- ( 5 ) 视频：YouTube的用户上传了总时长48小时的视频，130万个视频被观看。
- ( 6 ) 社交网站：Facebook网站的用户分享了684 478篇文章，超过600万页面被点击。
- ( 7 ) 微博：Twitter的用户发出了10万
- ( 8 ) 应用：Apple的应用商店完成了4.7万次应用下载。
- ( 9 ) 电子商务：eBay上产生了7万次页面访问，新增了35GB的数据。
- ( 10 ) 通信：在中国产生了时长531万分钟的移动通话，发出了165万条短信。

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。

大数据是人工智能的基础

大数据是物理网的基础

大数据和区块链

大数据是一种新的生产力。

# Hadoop概述



官方网址：<http://hadoop.apache.org/>

## Welcome to Apache™ Hadoop®!

### What Is Apache Hadoop?

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

The Apache Hadoop software library is a framework that allows for the distributed processing of **large data** sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

Apache的Hadoop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数据，以及实现分布式的计算。**

Hadoop的两个作用：①存储海量数据 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的创始人 Doug Cutting 与 Mike Cafarella



**Doug Cutting** is an advocate and creator of open-source search technology. He originated **Lucene** and, with Mike Cafarella, **Nutch**, both open-source search technology projects which are now managed through the Apache Software Foundation. He and Mike Cafarella are also the co-founders of Hadoop.

1985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。



**Mike Cafarella** is a computer scientist specializing in database management systems. He is currently an associate professor of computer science at University of Michigan.[1] Along with Doug Cutting, he is one of the original co-founders of the Hadoop and Nutch open-source projects.

## Hadoop演变

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》，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。

#### 课外：Hadoop相关历史趣闻

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。在此之前，Cutting所在的公司 Architext(其主要产品为Excite搜索引擎)因没有顶住互联网经济泡沫的冲击而破产，那时的Cutting正处在Freelancer的生涯 中，所以他希望自己的项目能通过一种低开销的方式来构建网页中的大量算法。

幸运的是，Google这时正好发布了一项研究报告，报告中介绍了两款 Google为支持自家的搜索引擎而开发的软件平台。这两个平台一个是GFS(Google File System)，用于存储不同设备所产生的海量数据;另一个是

MapReduce，它运行在GFS之上，负责分布式大规模数据。基于这两个平台，Cutting最引人瞩目的作品——Hadoop诞生了(后来大家习惯认为Hadoop于2006年1月28日诞生的)。

谈到Google对他们的“帮助”，Cutting说：“我们开始设想用4~5台电脑来实现这个项目，但在实际运行中牵涉了大量繁琐的步骤需要靠人工来完成。Google的平台让这些步骤得以自动化，为我们实现整体框架打下了良好的基础。”

说起Google，Cutting也是它成长的见证人之一，这里有一段鲜为人知的故事。早在Cutting供职于Architext期间，有两个年轻人曾去拜访这家公司，并向他们兜售自己的搜索技术，但当时他们的Demo只检索出几百万条网页，Excite的工程师们觉得他们的技术太小儿科，于是就在心里鄙视一番，把他们给送走了。但故事并未到此结束，这两个年轻人回去之后痛定思痛，决定自己创业。于是，他们开了一家自己的搜索公司，取名为Google。这两个年轻人就是Larry Page和Sergey Brin。在Cutting看来，Google的成功主要取决于，反向排序之后再存储的设计和对自身技术的自信。

出于对时间成本的考虑，在从Architext离职四年后，Cutting决定结束这段Freelancer的生涯，找一家靠谱的公司，进一步完善Hadoop的性能。他先后面试了几家公司，其中也包括IBM，但IBM似乎对他的早期项目Lucene更感兴趣，至于Hadoop则不置可否。就在此时，Cutting接受了当时Yahoo!搜索项目负责人Raymie Stata的邀请，于2006年正式加入Yahoo!。在Yahoo!，有一支一百人的团队帮助他完善Hadoop项目，这期间开发工作进行得卓有成效。不久之后，Yahoo!就宣布，将其旗下的搜索业务的架构迁移到Hadoop上来。两年后，Yahoo!便基于Hadoop启动了第一个应用项目“webmap”——一个用来计算网页间链接关系的算法。Cutting的时任上司(后为Hortonworks CEO)Eric Baldeschwieler曾说：“在相同的硬件环境下，基于Hadoop的webmap的反应速度是之前系统的33倍。”



# Hadoop版本说明

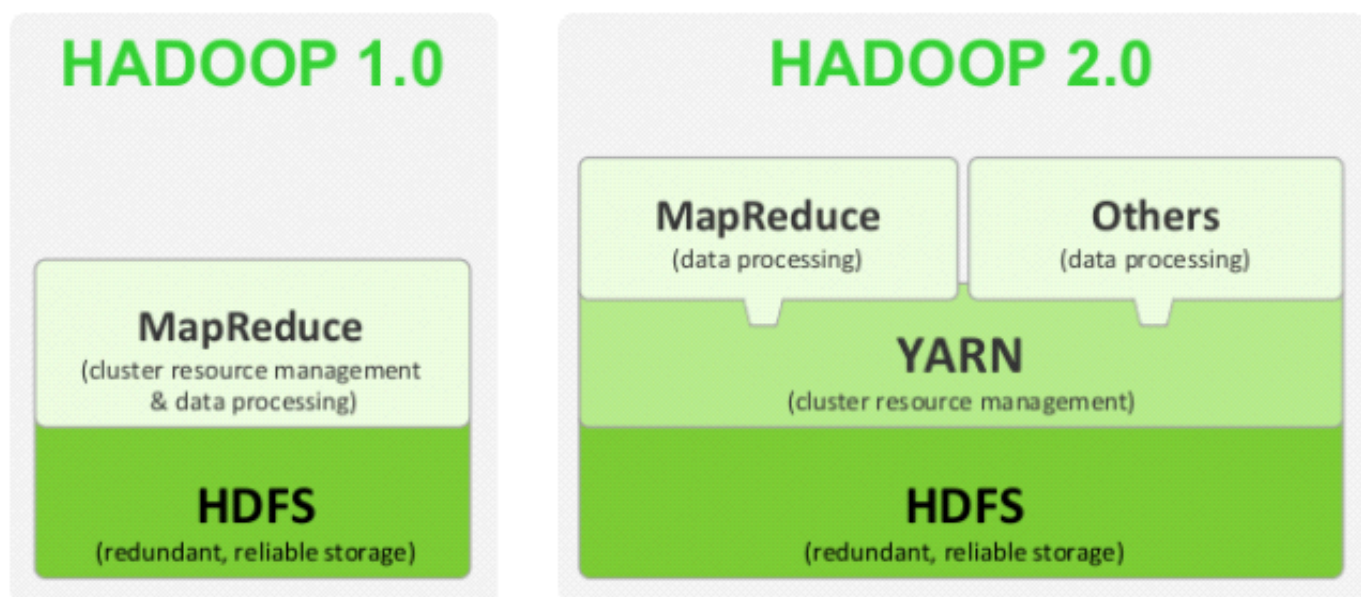
## 第一代Hadoop和第二代Hadoop

第一代Hadoop称为Hadoop1.0,分别是0.20.x,0.21.x和0.22.x

第二代Hadoop称为Hadoop2.0,分别是0.23.x和2.x。

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。

## Hadoop1.0和Hadoop2.0简要架构图



Yarn 资源调度框架——>实现对资源的细粒度封装（cpu，内存，带宽）

此外，还可以通过yarn协调多种不同计算框架（MR，Spark）

**最新Hadoop3.0** 2017年9月释出的。

# Hadoop伪分布式安装

下载地址：

<http://hadoop.apache.org/releases.html>



注意：

source表示源码

binary表示二进制包（安装包）

## 安装模式

**单机模式：**不能使用HDFS，只能使用MapReduce,所以单机模式最主要的目的是在本机调试mapreduce代码

**伪分布式模式：**用多个线程模拟多台真实机器，即模拟真实的分布式环境。

**完全分布式模式：**用多台机器（或启动多个虚拟机）来完成部署集群。

安装步骤：

### 0.关闭防火墙

**执行：**service iptables stop 这个指令关闭完防火墙后，如果重启，防火墙会重新建立，所以，如果想重启后防火墙还关闭，

**需额外执行：**chkconfig iptables off

### 1.配置主机名

执行：`vim /etc/sysconfig/network`

编辑主机名

注意：主机名里不能有下滑线，或者特殊字符#\$，不然会找不到主机导致无法启动

这种方式更改主机名需要重启才能永久生效，因为主机名属于内核参数。

如果不想重启，可以执行：`hostname hadoop01`。但是这种更改是临时的，重启后会恢复原主机名。

所以可以结合使用。先修改配置文件，然后执行:`hostname hadoop01`。可以达到不重启或重启都是主机名都是同一个的目的

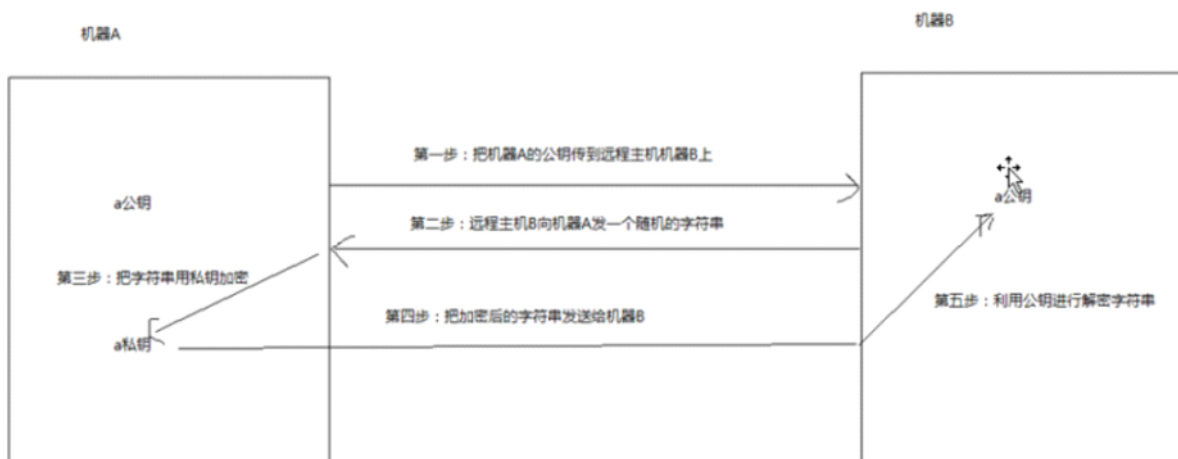
```
1 hadoop x +
NETWORKING=yes
HOSTNAME=hadoop01
```

### 2.配置hosts文件

执行：`vim /etc/hosts`

```
1 hadoop x +
127.0.0.1 localhost
::1 localhost
192.168.234.190 hadoop01
```

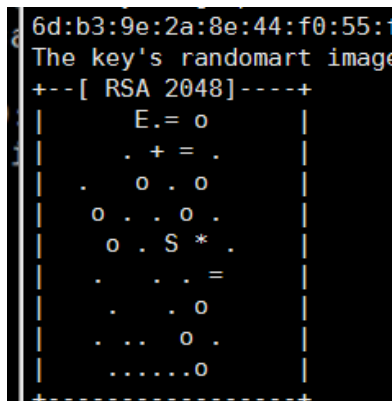
### 3.配置免密钥登录



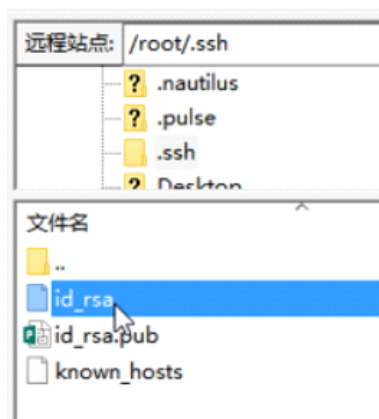
在hadoop01节点执行：

执行：`ssh-keygen`

然后一直回车



生成节点的公钥和私钥，生成的文件会自动放在/root/.ssh目录下



然后把公钥发往远程机器，比如hadoop01向hadoop01发送

**执行：ssh-copy-id root@hadoop01**

此时，hadoop02节点就是把收到的hadoop秘钥保存在

/root/.ssh/authorized\_keys 这个文件里，这个文件相当于访问白名单，凡是在此白名单存储的秘钥对应的机器，登录时都是免密码登录的。

当hadoop01再次通过ssh远程登录hadoop01时，发现不需要输入密码了。

```
[root@Hadoop01 ~]# ssh hadoop02
Last login: Thu Sep  1 00:07:38 2016 from 192.168.234.1
[root@localhost ~]# exit
logout
Connection to hadoop02 closed.
```

在hadoop02节点执行上述步骤，让hadoop02节点连接hadoop01免密码登录

#### 4.配置自己节点登录的免密码登录

如果是单机的伪分布式环境，节点需要登录自己节点，即hadoop01要登录hadoop01  
但是此时是需要输入密码的，所以要在hadoop01节点上

**执行：ssh-copy-id root@hadoop01**

#### 5.安装和配置jdk

执行：vi /etc/profile

2) 在尾行添加

#set java environment

```
JAVA_HOME=/usr/local/src/java/jdk1.7.0_51
PATH=$JAVA_HOME/bin:$PATH
CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
export JAVA_HOME PATH CLASSPATH
```

保存退出

3 ) source /etc/profile 使更改的配置立即生效

4 ) java -version 查看JDK版本信息。如显示1.7.0证明成功。

## 6.上传和解压hadoop安装包

**执行：**tar -xvf hadoop……

**目录说明：**

bin目录：命令脚本

etc/hadoop:存放hadoop的配置文件

lib目录：hadoop运行的依赖jar包

sbin目录：启动和关闭hadoop等命令都在这里

libexec目录：存放的也是hadoop命令，但一般不常用

**最常用的就是bin和etc目录**

## 7.配置hadoop-env.sh

这个文件里写的是hadoop的环境变量,主要修改hadoop的java\_home路径

切换到 etc/hadoop目录

**执行：**vim hadoop-env.sh

修改java\_home路径和hadoop\_conf\_dir 路径

```
# The java implementation to use.
export JAVA_HOME=/home/software/jdk1.8

# The jsvc implementation to use. Jsvc is required to run secure datanodes
# that bind to privileged ports to provide authentication of data transfer
# protocol.  Jsvc is not required if SASL is configured for authentication of
# data transfer protocol using non-privileged ports.
#export JSVC_HOME=${JSVC_HOME}

export HADOOP_CONF_DIR=/home/software/hadoop-2.7.1/etc/hadoop

# Extra Java CLASSPATH elements.  Automatically insert capacity-scheduler.
```

**然后执行：**source hadoop-env.sh 让配置立即生效

## 8.修改core-site.xml

在etc/hadoop目录下

**执行：**vim core-site.xml

 **配置如下：**

```

<configuration>
<!--用来指定hdfs的老大，namenode的地址-->
<property>
<name>fs.default.name</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-2.7.1/tmp</value>
</property>
</configuration>

```

```

<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-2.7.1/tmp</value>

</property>

</configuration>

```

## 9.修改 hdfs-site.xml

 配置如下：

```

<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即可-->
<property>
<name>dfs.replication</name>
<value>1</value>
</property>

```

```

<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>
```

## 10.修改 mapred-site.xml

这个文件初始时是没有的，有的是模板文件，mapred-site.xml.template  
所以需要拷贝一份，并重命名为mapred-site.xml

**执行：**cp mapred-site.xml.template mapred-site.xml

📖 配置如下：

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

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

yarn是资源协调工具，

## 11.修改yarn-site.xml

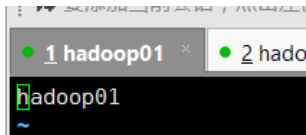
📖 配置如下：

```
<configuration>
<!-- Site specific YARN configuration properties -->
<property>
<!--指定yarn的老大 resourcemanager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>
<!-- Site specific YARN configuration properties -->
<property>
<!--指定yarn的老大 resourcemanager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>
```

## 12.配置slaves文件



## 13.配置hadoop的环境变量



配置代码：

```
unset i
unset -f pathmunge
#java env
JAVA_HOME=/home/software/jdk1.8
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 PATH CLASSPATH HADOOP_HOME
"/etc/profile" 84L, 2042C
```

JAVA\_HOME=/home/software/jdk1.8

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 PATH CLASSPATH HADOOP\_HOME

## 14.格式化namenode

为什么要格式化

**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

```
168.234.21-1472748147462
16/09/01 09:42:27 INFO common.Storage: Storage directory /home/software/hadoop-2.7.1
/tmp/dfs/name has been successfully formatted.
16/09/01 09:42:27 INFO namenode.NNStorageRetentionManager: Going to retain 1 images
```



# Hadoop的启动

2016年9月1日 10:10

切换到sbin目录，

**执行：start-dfs.sh 启动hadoop相关的服务**

```
[root@Hadoop01 sbin]# jps
4689 DataNode
4573 NameNode
4958 Jps
4847 SecondaryNameNode
```

**！ 注：如果在启动时，报错：Cannot find configuration directory: /etc/hadoop**

**解决办法：**

编辑 etc/hadoop下的 hadoop-env.sh 文件，添加如下配置信息：

```
export HADOOP_CONF_DIR=/home/software/hadoop-2.7.1/etc/hadoop
```

```
# The java implementation to use.
export JAVA_HOME=/home/software/jdk1.8
export HADOOP_CONF_DIR=/home/software/hadoop-2.7.1/etc/hadoop
# The jsvc implementation to use. Jsvc is required to run secure datanodes
```

然后执行 source hadoop-env.sh 使配置立即生效

**执行：start-yarn.sh 启动yarn相关的服务**

```
[root@Hadoop01 sbin]# jps
5168 Jps
4689 DataNode
5139 NodeManager
5050 ResourceManager
4573 NameNode
4847 SecondaryNameNode
```

**在浏览器访问：**

192.168.234.21:50070来访问hadoop的管理页面

nation × +

http://192.168.234.21:50070/dfshealth.html#tab-overview

谷歌 网址大全 360搜索 游戏中心 Links Login Ex Maven

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities ▾

## Overview 'hadoop01:9000' (active)

Started:	Thu Sep 01 10:15:15 PDT 2016
Version:	2.7.1, r15ecc87ccf4a0228f35af08fc56de536e6ce657a
Compiled:	2015-06-29T06:04Z by jenkins from (detached from 15ecc87)
Cluster ID:	CID-21aa2306-6c67-4b79-b3e8-ec9b5f3776eb
Block Pool ID:	BP-45509429-192.168.234.21-1472748147462

## Summary

Security is off

或在sbin目录下执行: sh start-all.sh 或sh stop-all.sh 启动或关闭hadoop

# 启动失败的解决办法

2018年8月24日 14:08

## 常见错误

1.执行Hadoop指令，比如格式化：hadoop namenode -format

报：command找不到错误

检查：/etc/profile的Hadoop配置

2.少HDFS相关进程，比如少namenode,datanode,

可以去Hadoop 安装目录下的logs目录，查看对应进程的启动日志文件。

解决方法：①先停止HDFS相关的所有的进程（stop-dfs.sh 或 kill -9）

②再启动HDFS（start-dfs.sh）

另外一种解决方式：①先停止HDFS相关的所有的进程 ②删除tmp(元数据目录)再创建一个新的 ③需要

做一次格式化指令：hadoop namenode -format

④启动HDFS

# Hadoop-HDFS命令

2016年9月2日 20:33

## HDFS常用指令

命令	说明
1.执行：hadoop fs -mkdir /park	在hdfs的根目录下，创建 park目录
2.执行：hadoop fs -ls /	查看hdfs根目录下有哪些目录
3.执行：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
4.执行：hadoop fs -get /park/jdk /home	把hdfs文件系统下park目录的文件下载到linux的home目录下
5.执行：hadoop fs -rm /park/文件名	删除hdfs的park目录的指定文件
6.执行：hadoop fs -rmdir /park	删除park目录，但是前提目录里没有文件
7.执行：hadoop fs -rmr /park	删除park目录，即使目录里有文件
8.执行：hadoop fs -cat /park/a.txt	查看park目录下的a.txt文件
9.执行：hadoop fs -tail /park/a.txt	查看park目录下a.txt文件末尾的数据
10.执行：haddop jar xxx.jar	执行jar包
11.执行：hadoop fs -cat /park/result/part-r-00000	查看 /park/result/part-r-00000文件的内容
12.执行：hadoop fs -mv /park02 /park01	将HDFS上的park02目录重名为park01命令。
13.执行：hadoop fs -mv /park02/1.txt /park01	将park02目录下的1.txt移动到/park01目录下
14.执行：hadoop fs -touchz /park/2.txt	创建一个空文件
15.执行：hadoop fs -getmerge /park /root/tmp	将park目录下的所有文件合并成一个文件，并下载到linux的root目录下的tmp目录
16.执行：hadoop dfsadmin -safemode leave  离开安全模式  执行：hadoop dfsadmin -safemode enter  进入安全模式	<p>离开hadoop安全模式</p> <p>在重新启动HDFS后，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。</p> <p>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。</p> <p>在此阶段NameNode收集各个DataNode的报告，当数据块达到最小副本数以上时，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</p> <p>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。</p> <p>当启动报如下错误时：</p> <pre>org.apache.hadoop.dfs.SafeModeException: Cannot delete /user/hadoop/input. Name node is in safe mode</pre>
17.执行：hadoop dfsadmin -rollEdits	手动执行fsimage文件和Edis文件合并元数据
18.执行：hadoop dfsadmin -report	查看存活的datanode节点信息
19.执行：hadoop fsck /park	汇报/park目录 健康状况
20.执行：hadoop fsck /park/1.txt -files -blocks -locations -racks	查看1.txt 这个文件block信息以及机架信息

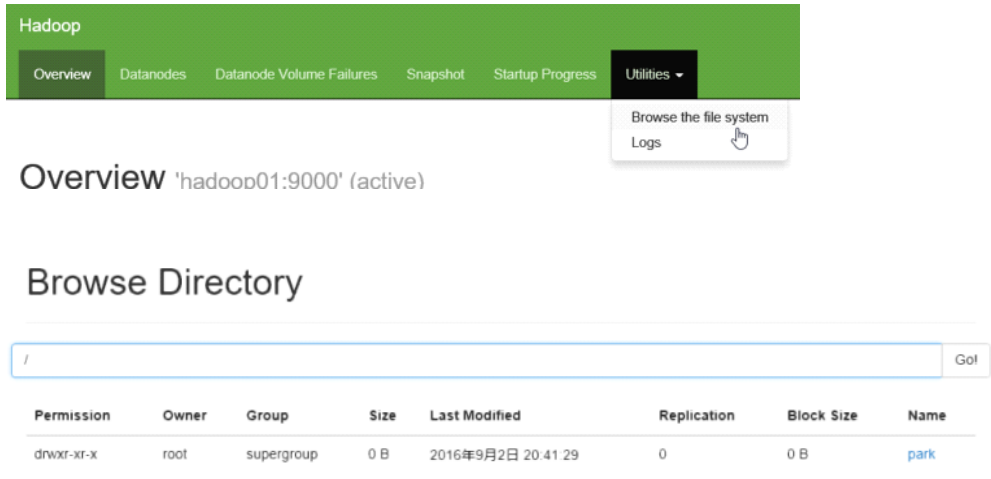
	<pre>/park/1.txt 4 bytes, 1 block(s): OK 0. BP-1831578705-192.168.234.190-1449283150894:blk_1073741826_1002 len=4 repl=1 [/default-rack/192.168.234.190:50010]  Status: HEALTHY Total size: 4 B Total dirs: 0 Total files: 1 Total symlinks: 0 Total blocks (validated): 1 (avg. block size 4 B) Minimally replicated blocks: 1 (100.0 %) Over-replicated blocks: 0 (0.0 %) Under-replicated blocks: 0 (0.0 %) Mis-replicated blocks: 0 (0.0 %) Default replication factor: 1 Average block replication: 1.0 Corrupt blocks: 0 Missing replicas: 0 (0.0 %) Number of data-nodes: 1 Number of racks: 1 FSCK ended at Fri Dec 04 21:30:42 PST 2015 in 12 milliseconds</pre>
21.hadoop fs -expunge	手动清空hdfs回收站

其他指令

命令	说明
hadoop fs -cp /park01/1.txt /park02	将HDFS上 /park01下的1.txt拷贝一份到 /park02目录下。 目标路径可以有多个，用空格隔开，比如： hadoop fs -cp /park01/1.txt /park02 /park03.....
hadoop fs -du /park/1.txt	查看HDFS上某个文件的大小。也可以查看指定目录，如果是目录的话，则列出目录下所有的文件及其大小，比如： hadoop fs -du /park
hadoop fs -copyFromLocal /home/1.txt /park01	将本地文件1.txt上传到/park01目录下
hadoop fs -copyToLocal /park01/1.txt /home	将HDFS上的1.txt 拷贝到本地文件系统
hadoop fs -lsr /	递归查看指定目录下的所有内容

HDFS控制台页面

通过访问：192.168.234.21:50070来查看hdfs系统



将文件上传到hdfs后，如果大于128M,会被切成两块

[Download](#)

Block information --

Block 0  
Block 1

Block ID: 1073741826

Block Pool ID: BP-45509429-192.168.234.21-1472748147462

Generation Stamp: 1002

Size: 47043070

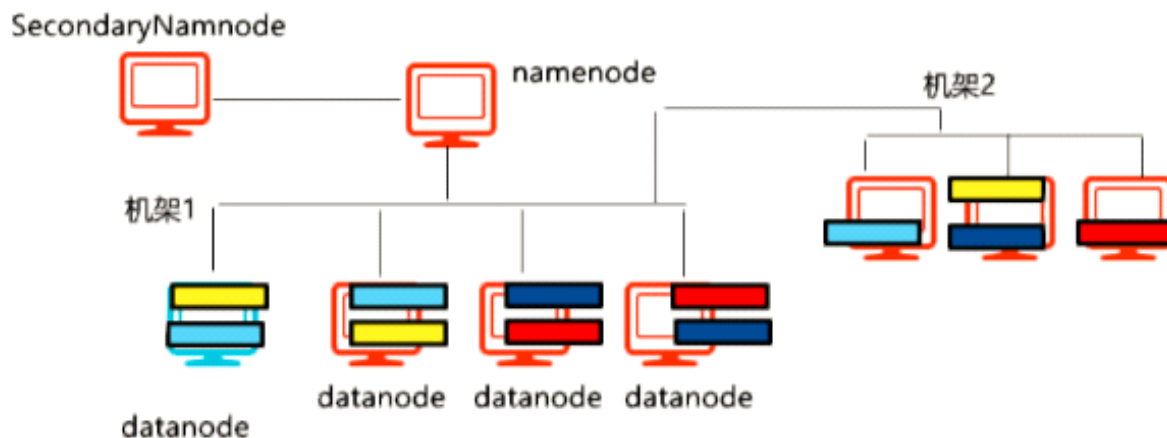
Availability:

- hadoop01

# HDFS细节

2018年8月24日 15:15

## HDFS架构图



## 知识点

1.HDFS Hadoop Distributed File System Hadoop的分布式文件系统，可以存储海量数据（文件，图片，影音等），实际工作中，一般存储的都是用户的访问日志（.txt）

2.HDFS之所以能够存储海量数据，原因是一个可扩展的分布式架构，硬盘存储空间不够，加服务器即可。

3.HDFS是基于Google的一篇论文《Google File System》

4.namenode，名字节点。最主要的职责是管理和存储HDFS的元数据信息（比如文件名，文件大小，文件切块的数量，每个文件块的大小，文件块编号，存储在哪个datanode上），可以通过指令：

```
hadoop fsck /park01/1.txt -files -blocks -locations
```

5.namenode不存储文件块

6.namenode除了存储元数据信息以外，还会通过RPC心跳机制来管理各个datanode

7.namenode会把元数据信息放在namenode服务器的内存里，目的是供用户快速查询。

8.namenode为了确保元数据存储的可靠性，会将元数据落地，存储的目录路径有core-site.xml里的hadoop.tmp.dir来指定的。

注意：此属性默认是放在linux的 /tmp 目录，所以在工作中一定要更换此目录。

9.namenode底层是通过两个文件来进行元数据管理：

①Edits文件 当客户端发起写请求时，Edits文件都会进行记录

写请求，比如：`-mkdir -put -mv`

②Fimage文件 存储元数据信息的

以上这两个文件会定期做一次合并，合并的目的是确保Fimage文件里的数据是最新的。合并周期默认是3600s（1小时）

10.以上这两个文件可以在配置的元数据目录tmp/dfs/name/current找到

11.格式化指令：`hadoop namenode -format`

它的作用是在元数据目录生成新的Edits和Fimage文件。

这个指令 在初次Hadoop时需要执行一次。

12.格式化指令很危险，因为它会清空之前所有的元数据。所以在实际工作，初次使用完之后，会通过配置文件的使得这个指令失效。

13.Edits和Fimage文件可以通过指令手动合并：`hadoop dfsadmin -rollEdits`

14.为了当namenode宕机后，SNN能够工作，我们需要SNN具有元数据数据，

所以Hadoop底层机制是让SNN来做元数据的合并工作，通过这机制使得NN和SNN都有元数据信息了。

**注意：以上这个机制存在一个问题，就是可能会造成元数据丢失**

**比如：**

3：00 Edits和Fimage 合并一次

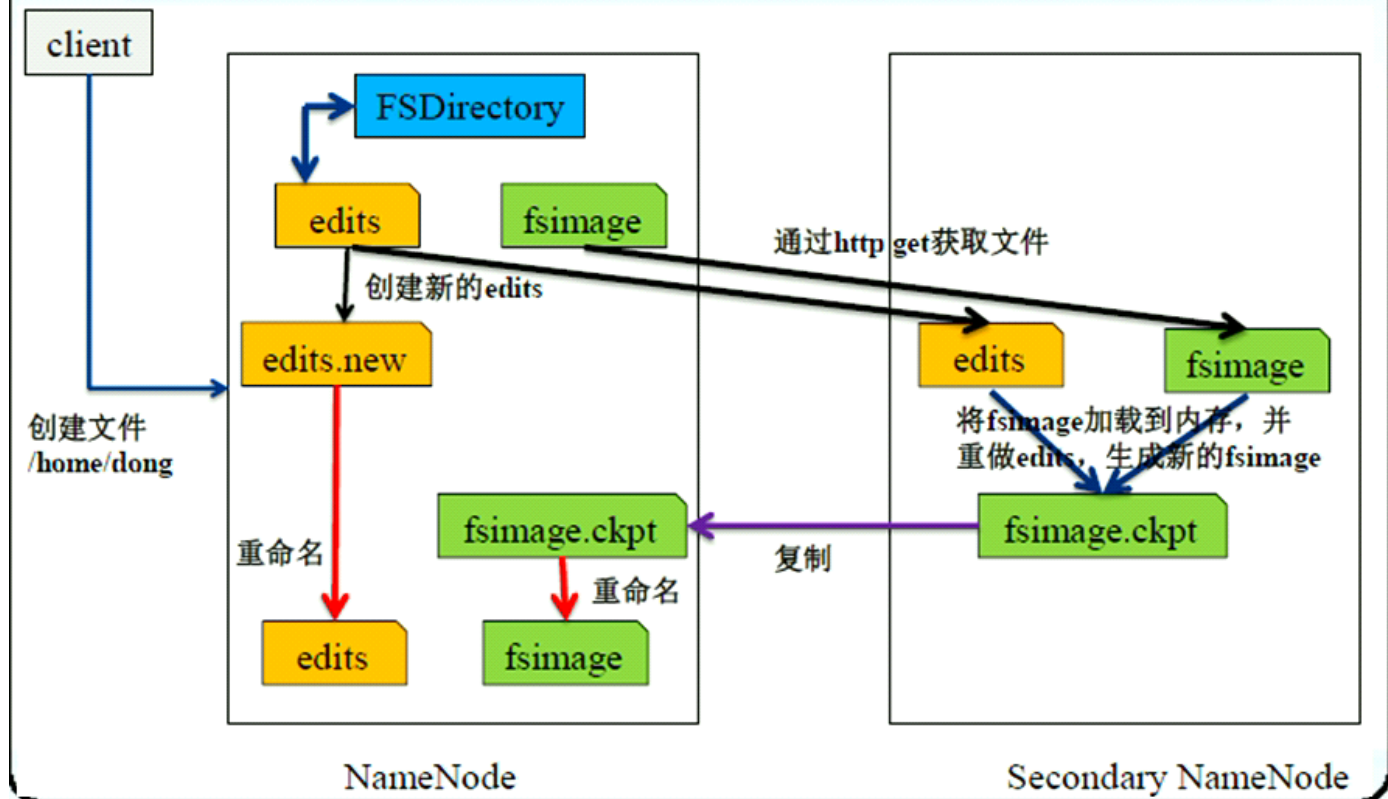
3：30 namenode 宕机了

相当于丢失了30分钟之内的元数据信息。

这个机制是Hadoop1.0的机制，是存在问题的，即SNN合并不能达到实时热备，所以会元数据丢失，即Hadoop1.0的namenode还是会存在单点故障问题。



# HDFS 架构 —— NN & SNN

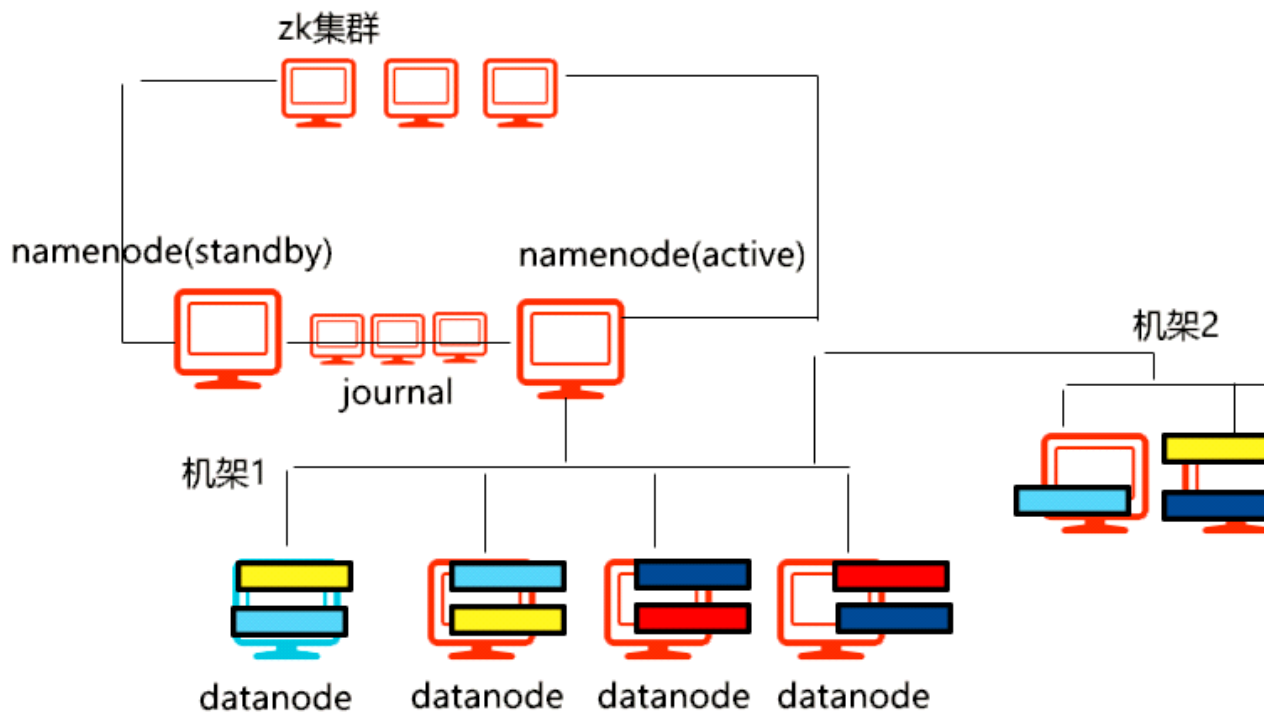


15. 下图是Hadoop2.0 集群的架构图。

Hadoop2.0如果伪分布式模式，会看到SNN，但实际没有作用

Hadoop2.0如果是完全分布式，就没有SNN进程了

## Hadoop 2.0 HA 架构



16.可以指令单独启停某个进程:

启动：

```
hadoop-daemon.sh start namenode
```

```
hadoop-daemon.sh start datanode
```

```
hadoop-daemon.sh start secondarynamenode
```

停止：

```
hadoop-daemon.sh stop namenode
```

此外，掌握：`start-dfs.sh` 启动和HDFS相关的所有进程

17.当namenode挂掉再次启动时，底层会将Edits和Fsimage合并一次。

3：10 namenode 宕机

即3:00 ~ 3：10会产生一些操作记录，

所以通过这种机制，可以确保namenode宕机再启动之后，Fsimage是最新的

18.当整个HDFS启动（namenode和datanode）时，底层每台datanode都会向namenode汇报自身的存储状态信息（存储了哪些文件块），namenode收到这写数据后，汇总并检查。检查文件块是否缺少，数据是否丢失，以及每个文件块的副本数量是否达到要求（集群环境，3副本），如果检查有问

题，HDFS会进入安全模式，在安全模式下，要完成数据的修复、副本的复制。安全模式直到数据恢复完毕之后自动退出

19.如果HDFS处于安全模式，对外只能提供读服务，不能提供写服务。

可以通过指令：

```
hadoop dfsadmin -safemode enter
```

```
hadoop dfsadmin -safemode leave
```

20.如果是伪分布式模式，在hdfs-site.xml 配置副本数量，只能配置1，因为如果大于1，会使得HDFS一直安全模式而不退出（因为副本数量一直满足不了要求，就一台服务器，只能存一个副本）

21.datanode，数据节点，专门用来存储和管理文件块。

22.datanode会定期向namenode发送心跳，心跳周期默认是3s

23.配置datanode地址列表是在slaves配置文件里配置的

24.HDFS存储文件的方式是切块存储，

Hadoop 1.0 切块大小64MB

Hadoop 2.0 切块大小128MB

比如：上传来个文件①1.txt 257MB ②2.txt 100MB ,并设定副本数=1

①1.txt 1块：128MB 2块：128MB 3块：1MB

②2.txt 1块：100MB

综上，共4个文件块。

**强调：**切块是以文件为单位的，不同文件的文件块不能共用。

块是多大，在磁盘上就占多大。比如1MB文件块就占1MB，不会浪费127MB的磁盘空间的。

25.HDFS的特点：当文件上传HDFS之后，就不允许修改此文件。所以HDFS的适用场景：once-write-many-read （一次写入，多次读取）

26.HDFS不允许修改数据，但允许追加数据

27.HDFS是否适合存储海量的小文件（比如几kb,几mb）？

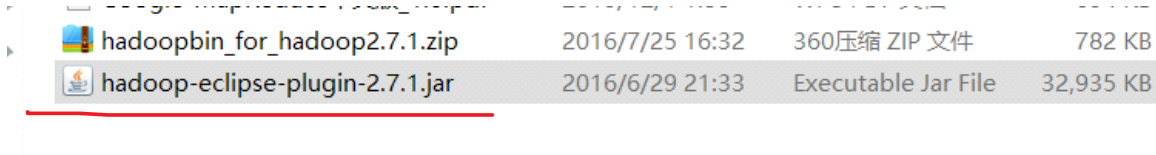
不适合，因为每个文件都会占用一条元数据信息，根据经验，一条元数据大约在150字节，即如果由海

量小文件时，会占用大量的namenode服务内存空间。

# Hadoop插件

## 安装步骤

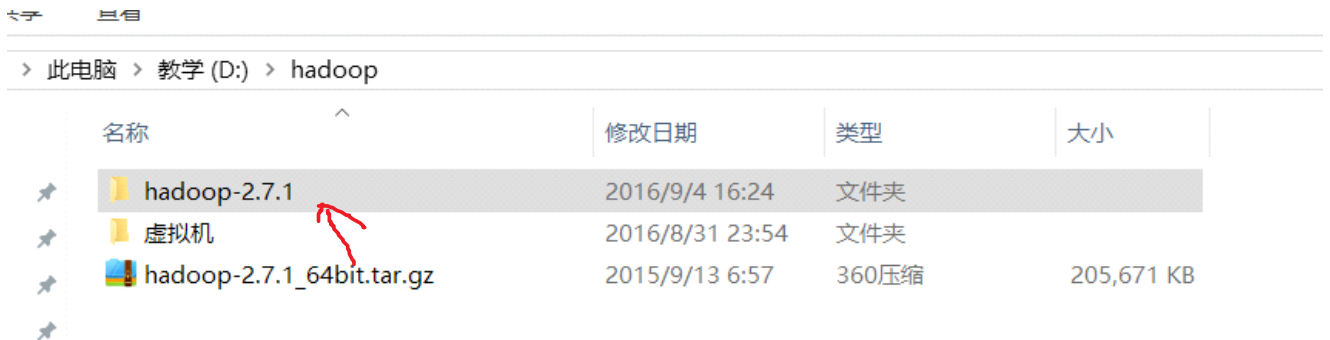
1. 下载hadoop插件，注意:插件的版本要和用的hadoop版本保持一致



hadoopbin_for_hadoop2.7.1.zip	2016/7/25 16:32	360压缩 ZIP 文件	782 KB
hadoop-eclipse-plugin-2.7.1.jar	2016/6/29 21:33	Executable Jar File	32,935 KB

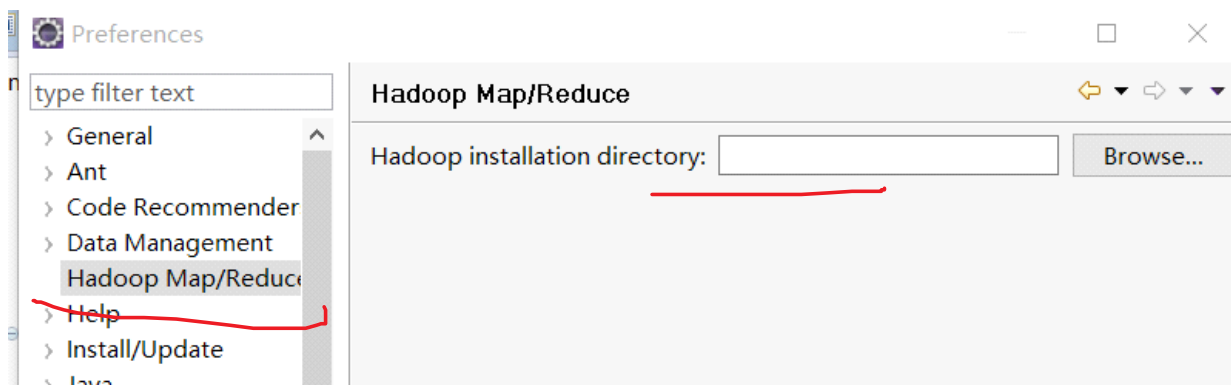
2. 将插件jar包放在eclipse安装目录的plugins目录下

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）

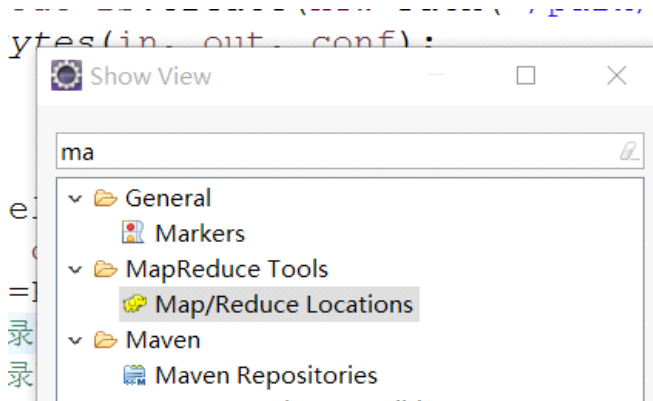


名称	修改日期	类型	大小
hadoop-2.7.1	2016/9/4 16:24	文件夹	
虚拟机	2016/8/31 23:54	文件夹	
hadoop-2.7.1_64bit.tar.gz	2015/9/13 6:57	360压缩	205,671 KB

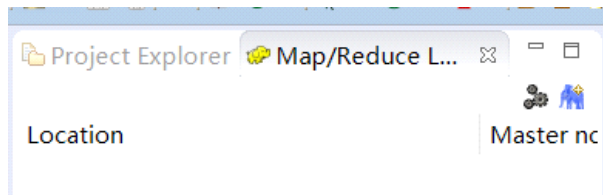
4. 重启eclipse，windows=>下发现多出Map/Reduce选项卡，点击=>选择hadoop的安装目录，然后点击apply，点击确定



5. 点击open perspective，调出map/reduce视图，或者show view，调出map/reduce 视图



6.在map/reduce视图下，点击蓝色的大象，新建hadoop客户端连接



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

**Define Hadoop location**  
Define the location of a Hadoop infrastructure for running MapReduce applications.

General Advanced parameters

Location name: 192.168.234.21

Map/Reduce(V2) Master

Host: 192.168.234.21

Port: 50020

DFS Master

☒ Use M/R Master host

Host: 192.168.234.21

Port: 9000

User name: root

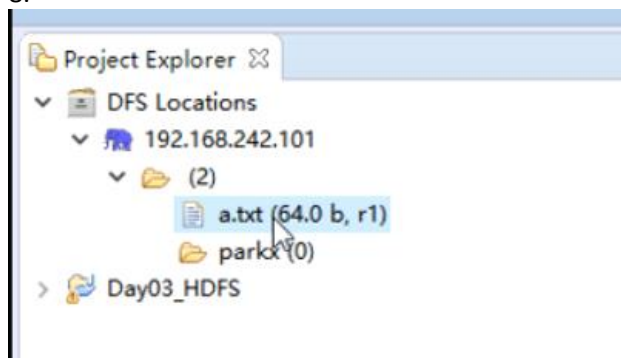
SOCKS proxy

☐ Enable SOCKS proxy

Host: host

Port: 1080

8.



## 9.hadoop的权限修改

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

这个可以配置，在hdfs-site.xml来配置：

属性值改成false即可

```
<configuration>
<property>
<!--指定hdfs保存数据副本的数量，包括自己，默认为3-->
<!--伪分布式模式，此值必须为1-->
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>dfs.permissions</name>
<value>false</value>
</property>
</configuration>
:wq
```

更改完之后，需要重新启动hadoop

## 插件使用常见问题解决办法

运行Hadoop2的WordCount.java代码时出现了这样错误，

```
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Exception in thread "main" java.lang.NullPointerException
    at java.lang.ProcessBuilder.start(Unknown Source)
    at org.apache.hadoop.util.Shell.runCommand(Shell.java:482)
    at org.apache.hadoop.util.Shell.run(Shell.java:455)
    at org.apache.hadoop.util.Shell$ShellCommandExecutor.execute(Shell.java:715)
    at org.apache.hadoop.util.Shell.execCommand(Shell.java:808)
    at org.apache.hadoop.util.Shell.execCommand(Shell.java:791)
    at
```

## 分析：

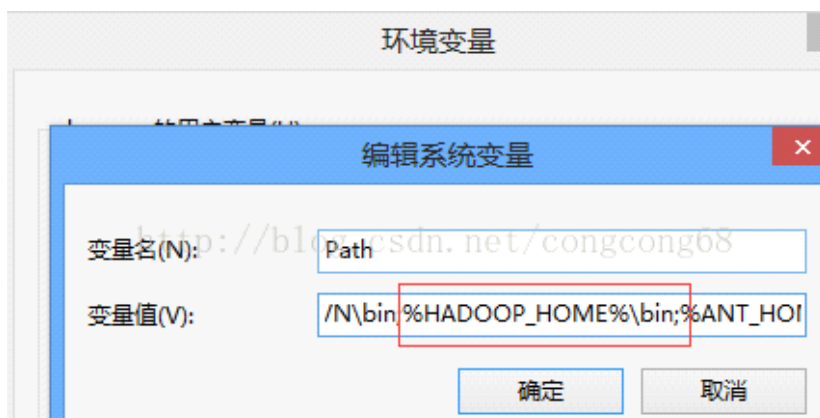
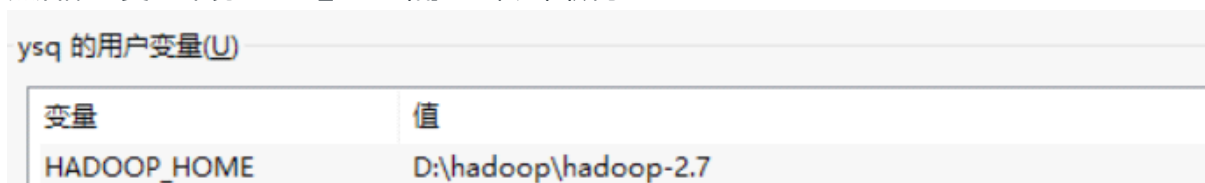
下载Hadoop2以上版本时，在Hadoop2的bin目录下没有winutils.exe

## 解决办法：

将资料里hadoopbin\_for\_hadoop2.7.1.zip包里的所有文件，拷贝到hadoop2.7.1安装目录的bin目录下即可



然后配置变量环境HADOOP\_HOME 和path，如图所示：



### 问题三. Exception in thread

"main" java.lang.UnsatisfiedLinkError:org.apache.hadoop.io.nativeio.NativeIO\$Windows.access0(Ljava/lang/String;I)Z

当我们解决了问题三时，在运行WordCount.java代码时，出现这样的问题

```
log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics2.lib.MutableMetricsFactory).
log4j:WARN Please initialize the log4jsystem properly.
log4j:WARN Seehttp://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Exception in thread "main" java.lang.UnsatisfiedLinkError:org.apache.hadoop.io.nativeio.NativeIO$Windows.access0(Ljava/lang/String;I)Z
    atorg.apache.hadoop.io.nativeio.NativeIO$Windows.access0(Native Method)
    atorg.apache.hadoop.io.nativeio.NativeIO$Windows.access(NativeIO.java:557)
    atorg.apache.hadoop.fs.FileUtil.canRead(FileUtil.java:977)
    atorg.apache.hadoop.util.DiskChecker.checkAccessByFileMethods(DiskChecker.java:187)
    atorg.apache.hadoop.util.DiskChecker.checkDirAccess(DiskChecker.java:174)
    atorg.apache.hadoop.util.DiskChecker.checkDir(DiskChecker.java:108)
    atorg.apache.hadoop.fs.LocalDirAllocator
```



```

$AllocatorPerContext.confChanged(LocalDirAllocator.java:285)
    at org.apache.hadoop.fs.LocalDirAllocator
$AllocatorPerContext.getLocalPathForWrite(LocalDirAllocator.java:344)
    at org.apache.hadoop.fs.LocalDirAllocator.getLocalPathForWrite(LocalDirAllocator.java:150)
    at org.apache.hadoop.fs.LocalDirAllocator.getLocalPathForWrite(LocalDirAllocator.java:131)
    at org.apache.hadoop.fs.LocalDirAllocator.getLocalPathForWrite(LocalDirAllocator.java:115)
    at org.apache.hadoop.mapred.LocalDistributedCacheManager.setup(LocalDistributedCacheManager.
java:131)

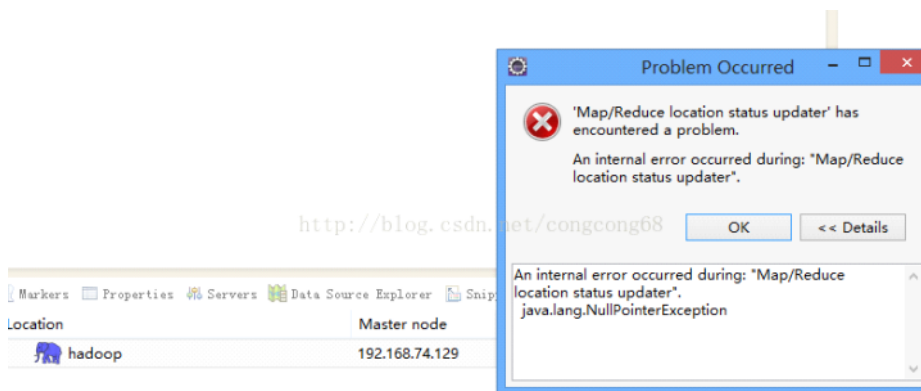
```

分析：

C:\Windows\System32下缺少hadoop.dll,把这个文件拷贝到C:\Windows\System32下面即可。然后重启

其他报错及解决办法

**问题一.** An internal error occurred during: "Map/Reduce location status updater". java.lang.NullPointerException



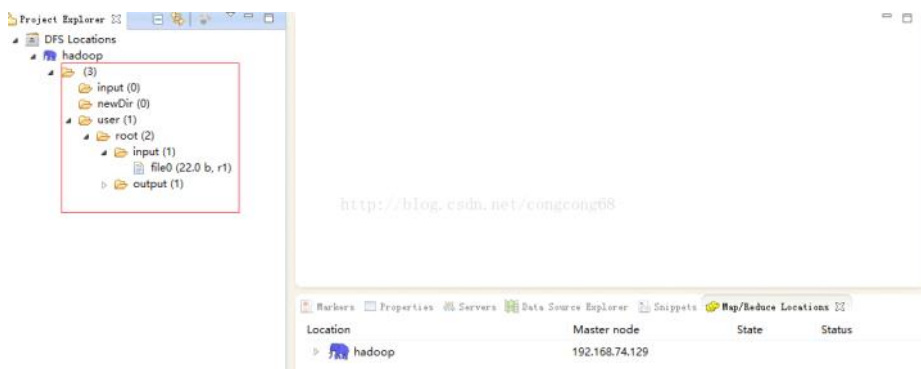
解决：

我们发现刚配置部署的Hadoop2还没创建输入和输出目录，先在hdfs上建个文件夹。

```
#bin/hdfs dfs -mkdir -p /user/root/input
```

```
#bin/hdfs dfs -mkdir -p /user/root/output
```

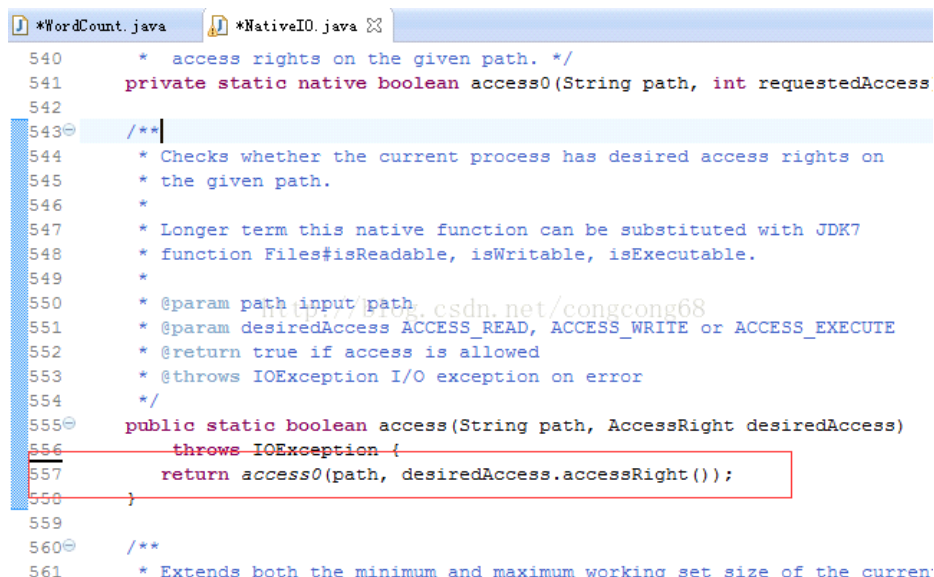
我们在Eclipse的DFS Locations目录下看到我们这两个目录，如图所示：



**问题二.** Exception in thread "main" java.lang.NullPointerException at java.lang.ProcessBuilder.start(Unknown Source)

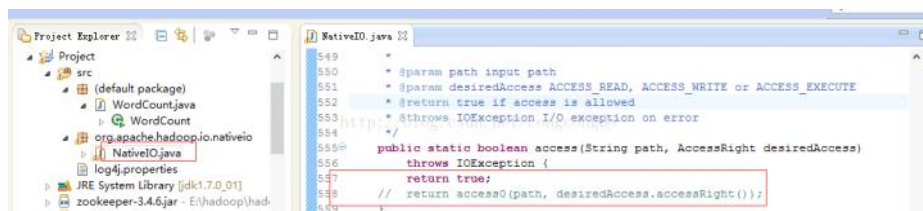
我们在继续分析：

我们在出现错误的的atorg.apache.hadoop.io.nativeio.NativeIO\$Windows.access(NativeIO.java:557)我们来看这个类NativeIO的557行，如图所示：



```
540      * access rights on the given path. */
541      private static native boolean access0(String path, int requestedAccess
542
543      /**
544       * Checks whether the current process has desired access rights on
545       * the given path.
546       *
547       * Longer term this native function can be substituted with JDK7
548       * function Files#isReadable, isWritable, isExecutable.
549       *
550       * @param path input path
551       * @param desiredAccess ACCESS_READ, ACCESS_WRITE or ACCESS_EXECUTE
552       * @return true if access is allowed
553       * @throws IOException I/O exception on error
554       */
555      public static boolean access(String path, AccessRight desiredAccess)
556      throws IOException {
557          return access0(path, desiredAccess.accessRight());
558      }
559
560      /**
561       * Extends both the minimum and maximum working set size of the current
```

Windows的唯一方法用于检查当前进程的请求，在给定的路径的访问权限，所以我们先给以能进行访问，我们自己先修改源代码，return true 时允许访问。我们下载对应hadoop源代码，hadoop-2.6.0-src.tar.gz解压，hadoop-2.6.0-src\hadoop-common-project\hadoop-common\src\main\java\org\apache\hadoop\io\nativeio下NativeIO.java 复制到对应的Eclipse的project，然后修改557行为return true如图所示：



```
549      * @param path input path
550      * @param desiredAccess ACCESS_READ, ACCESS_WRITE or ACCESS_EXECUTE
551      * @return true if access is allowed
552      * @throws IOException I/O exception on error
553      */
554      public static boolean access(String path, AccessRight desiredAccess)
555      throws IOException {
556          return true;
557          // return access0(path, desiredAccess.accessRight());
558      }
559
```

**问题四：**org.apache.hadoop.security.AccessControlException: Permissiondenied: user=zhengcy, access=WRITE, inode="/user/root/output":root:supergroup:drwxr-xr-x

我们在执行运行WordCount.java代码时，出现这样的问题

```
2014-12-18 16:03:24,092 WARN (org.apache.hadoop.mapred.LocalJobRunner:560) - job_local374172562
0001
org.apache.hadoop.security.AccessControlException: Permission denied: user=zhengcy, access=WRITE,
inode="/user/root/output":root:supergroup:drwxr-xr-x
    at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.checkFsPermission(FSPermission
Checker.java:271)
    at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.check(FSPermissionChecker.jav
a:257)
    at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.check(FSPermissionChecker.jav
a:238)
    at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.checkPermission(FSPermissionC
hecker.java:179)
```



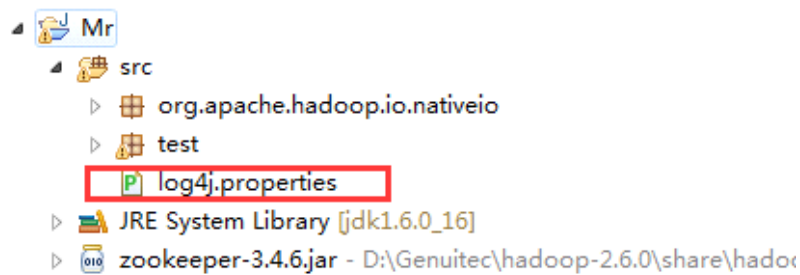
/user/root/input 把/test/\*文件上传到hdfs的/user/root/input中, 出现这样的问题,

解决:

是我们执行太多次了hadoopnamenode - format, 在创建了多个, 我们对应的hdfs目录删除hdfs-site.xml配置的保存datanode和namenode目录。

问题六: 在复制了hadoop.dll后, 运行WordCount, 发现运行一会没有任何信息输出就结束了

解决: 可以写一个log4j日志文件, 查看一下日志的输出, 可能从输出的日志中发现问题。



内容写为:

```
log4j.rootLogger=debug, stdout, R
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p - %m%n
log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=mapreduce_test.log
log4j.appender.R.MaxFileSize=1MB
log4j.appender.R.MaxBackupIndex=1
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n
log4j.logger.com.codefutures=DEBUG
```

问题七: 有了log4j日志输出后, 查看问题就比较方便了, 如果同一个MR执行两次, 会出现输出文件已存在的问题

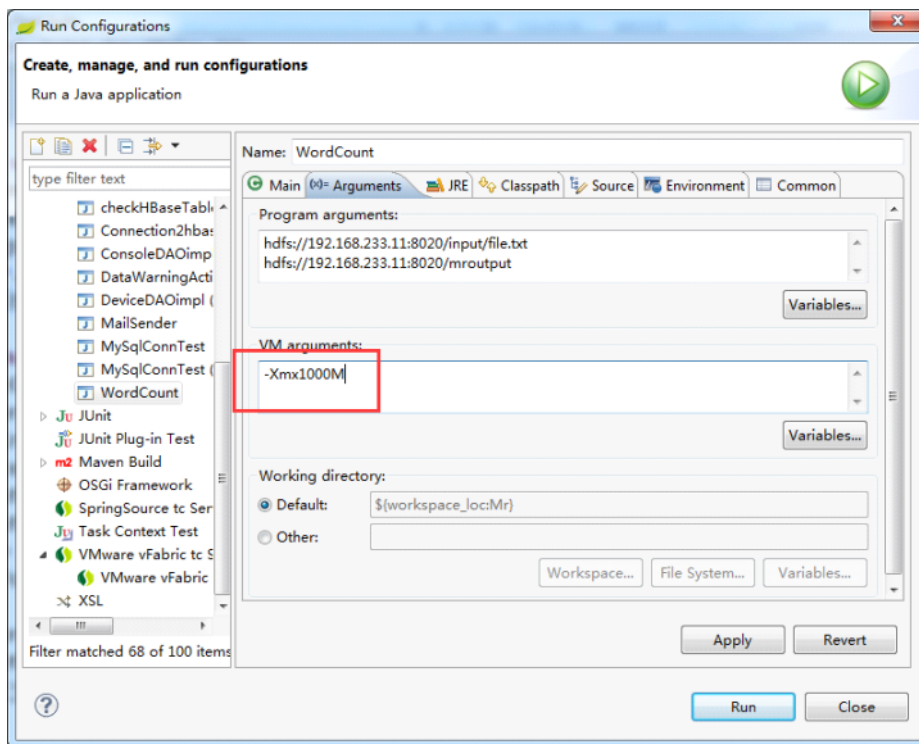
解决: 可以删除掉存在的输出文件, 也可以改代码中输出的路径

```
Exception in thread "main" org.apache.hadoop.mapred.FileAlreadyExistsException: Output directory hdfs://192.168.233.11:8020/mroutput already exists
    at org.apache.hadoop.mapreduce.lib.output.FileOutputFormat.checkOutputSpecs(FileOutputFormat.java:146)
    at org.apache.hadoop.mapreduce.JobSubmitter.checkSpecs(JobSubmitter.java:562)
    at org.apache.hadoop.mapreduce.JobSubmitter.submitJobInternal(JobSubmitter.java:432)
    at org.apache.hadoop.mapreduce.Job$10.run(Job.java:1296)
    at org.apache.hadoop.mapreduce.Job$10.run(Job.java:1293)
    at java.security.AccessController.doPrivileged(Native Method)
    at javax.security.auth.Subject.doAs(Subject.java:396)
    at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1628)
    at org.apache.hadoop.mapreduce.Job.submit(Job.java:1293)
    at org.apache.hadoop.mapreduce.Job.waitForCompletion(Job.java:1314)
    at test.WordCount.main(WordCount.java:87)
```

## 问题八: 出现内存溢出的问题 java.lang.OutOfMemoryError

```
WARN - job_local1845949011_0001
java.lang.Exception: java.lang.OutOfMemoryError: Java heap space
    at org.apache.hadoop.mapred.LocalJobRunner$Job.runTasks(LocalJobRunner.java:462)
    at org.apache.hadoop.mapred.LocalJobRunner$Job.run(LocalJobRunner.java:522)
Caused by: java.lang.OutOfMemoryError: Java heap space
    at org.apache.hadoop.mapred.MapTask$MapOutputBuffer.init(MapTask.java:983)
    at org.apache.hadoop.mapred.MapTask.createSortingCollector(MapTask.java:401)
    at org.apache.hadoop.mapred.MapTask.access$100(MapTask.java:81)
    at org.apache.hadoop.mapred.MapTask$NewOutputCollector.<init>(MapTask.java:695)
    at org.apache.hadoop.mapred.MapTask.runNewMapper(MapTask.java:767)
    at org.apache.hadoop.mapred.MapTask.run(MapTask.java:341)
    at org.apache.hadoop.mapred.LocalJobRunner$Job$MapTaskRunnable.run(LocalJobRunner.java:243)
    at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:441)
    at java.util.concurrent.FutureTask$Sync.innerRun(FutureTask.java:303)
    at java.util.concurrent.FutureTask.run(FutureTask.java:138)
    at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908)
    at java.lang.Thread.run(Thread.java:619)
```

解决: 右键WordCount, -->run Confi....



来自 <<https://my.oschina.net/muou/blog/408543>>

# HDFS 回收站机制

2016年1月18日 11:49

Hadoop回收站trash，默认是关闭的。

修改conf/core-site.xml, 增加

## 配置示例：

```
<property>
  <name>fs.trash.interval</name>
  <value>1440</value>
  <description>
    Number of minutes between trash checkpoints. If zero, the trash feature is disabled.
  </description>
</property>
```

注：value的时间单位是分钟，如果配置成0,表示不开启HDFS的回收站。

1440=24\*60,表示的一天的回收间隔，即文件在回收站存在一天后，被清空。

启动回收站后，比如我们删除一个文件：

```
[root@hadoop01 sbin]# hadoop fs -rm /word/1.txt
16/01/18 11:44:31 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
16/01/18 11:44:31 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 3 minutes, Emptier interval = 0 minutes.
Moved: 'hdfs://hadoop01:9000/word/1.txt' to trash at: hdfs://hadoop01:9000/user/root/.Trash/Current
```

我们可以通过递归查看指令，找到我们要恢复的文件放在回收站的哪个目录下

执行：hadoop fs -lsr /user/root/.Trash

```
[root@hadoop01 sbin]# hadoop fs -lsr /user/root/
lsr: DEPRECATED: Please use 'ls -R' instead.
16/01/18 11:46:39 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
drwx----- - root supergroup          0 2016-01-18 11:45 /user/root/.Trash
drwx----- - root supergroup          0 2016-01-18 11:44 /user/root/.Trash/160118114500
drwx----- - root supergroup          0 2016-11-16 06:15 /user/root/.Trash/160118114500/word
-rw-r--r--  3 ysq supergroup         49 2016-11-16 06:15 /user/root/.Trash/160118114500/word/1.txt
```

找到文件路径后，如果想恢复，执行hdfs的mv指令即可，（mv指令可用于文件的移动）