# Hadoop概述

**历史：**

Google：搜索引擎，爬取整个互联网上的数据，并且对数据进行处理和存储。

《Google file system》GFS，《MapReduce》

Doug cutting：Lucene，Nutch

Hadoop：HDFS（hadoop distributed file system），MapReduce

Hadoop的设计初衷是为了解决Nutch的海量数的存储和处理的需求，后来人们发现它非常适合解决大数据场景下的数据存储和处理问题，用于处理海量的离线数据。

传统数据

GB、TB级别的数据

数据增长较慢

主要为结构化的数据

统计和报表

大数据

TB、PB级别的数据

持续的高速增长，可能伴随爆发式的增长

半结构化、非结构化的数据

预测性分析，推荐

大数据指的是海量数据的获取、存储、聚合、管理这些数据以及对数据进行深度分析的新技术和新能力。

**版本：**

Hadoop1.0:HDFS,MapReduce

Hadoop2.0:HDFS,MapReduce,Yarn

Hadoop3.0:17年12月中旬推出

* **名字的起源：**

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。Google就是由小孩命名的。"

# Hadoop下载安装

## 下载

下载地址：<http://hadoop.apache.org/releases.html>

## 安装

Hadoop的安装分为单机模式、伪分布式模式 和 完全分布式模式。

单机模式是Hadoop的默认模式。当首次解压Hadoop的源码包时，Hadoop无法了解硬件安装环境，便保守地选择了最小配置。在这种默认模式下所有3个XML文件均为空。当配置文件为空时，Hadoop会完全运行在本地。因为不需要与其他节点交互，单机模式就不使用HDFS，也不加载任何Hadoop的守护进程。该模式主要用于开发调试MapReduce程序的应用逻辑。

伪分布模式 Hadoop守护进程运行在本地机器上，模拟一个小规模的的集群。可以使用HDFS和MapReduce

完全分布模式 Hadoop守护进程运行在一个真实集群上

A.单机安装

B.伪分布式安装

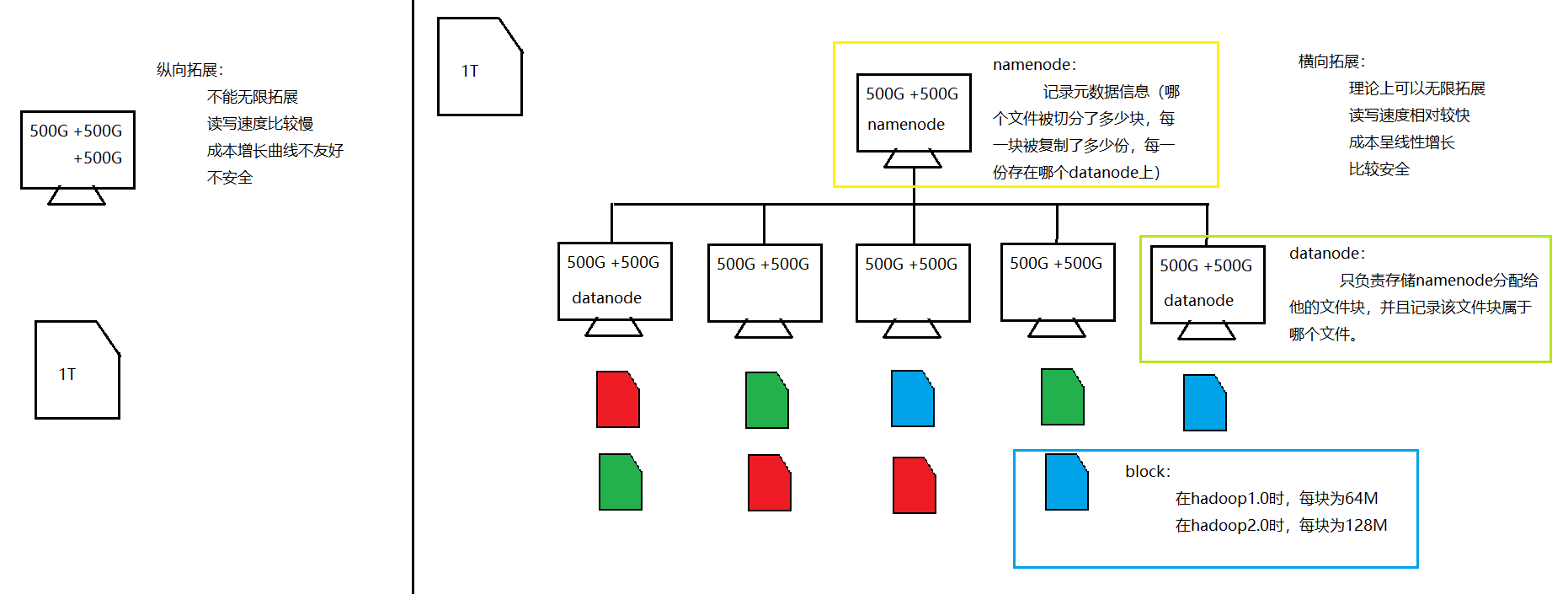
参看：hadoop伪分布式安装配置

C.完全分布式安装

# HDFS的特点

## HDFS概述:

问题：给我们一个超大文件怎么来存储？



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

存储元数据信息：文件、block、datanode之间的映射关系

元数据保存在内存和磁盘中

DataNode

存储block内容

存储在磁盘中

维护了block id到文件的映射关系

## HDFS优点

**支持超大文件**

支持超大文件。一般来说hadoop的文件系统会存储TB级别或者PB级别的数据。在企业的应用中，数据节点有可能有上千个。

**检测和快速应对硬件故障**

在集群的环境中，硬件故障是常见的问题。因为有上千台服务器连接在一起，这样会导致高故障率。因此故障检测和自动恢复是hdfs文件系统的一个设计目标。

**流式数据访问**

Hdfs的数据处理规模比较大，应用一次需要访问大量的数据，同时这些应用一般都是批量处理，而不是用户交互式处理。应用程序能以流的形式访问数据集。主要的是数据的吞吐量，而不是访问速度。

**简化的一致性模型**

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。

**高容错性**

数据自动保存多个副本，副本丢失后自动恢复

**可构建在廉价机器上**

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

## HDFS缺点

**低延迟数据访问**

低延迟数据。如和用户进行交互的应用，需要数据在毫秒或秒的范围内得到响应。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，所以对于低延迟来说，不适合用hadoop来做。

**大量的小文件**

Hdfs支持超大的文件，是通过数据分布在数据节点，数据的元数据保存在名字节点上。名字节点的内存大小，决定了hdfs文件系统可保存的文件数量。虽然现在的系统内存都比较大，但大量的小文件还是会影响名字节点的性能。

**多用户写入文件、修改文件**

Hdfs的文件只能有一次写入，不支持追加写入（2.0版本支持追加），也不支持修改。

**不支持超强的事务**

没有像关系型数据库那样，对事务有强有力的支持。

# HDFS技术细节

## Block

最基本的存储单位。

在HDFS中，有一个特别重要的概念：数据块(Block)。前面介绍到，在HDFS中存储的文件都是超大数据的文件，我们可以把这个超大规模的文件以一个标准切分成几块，分别存储到不同的磁盘上。这个标准就称为Block。Block 默认的大小为64(128)M。这样做有以下几点好处：

文件块可以保存在不同的磁盘上。在HDFS系统中，一个文件可以分成不同的Block存储在不同的磁盘上。

简化存储系统。这样不需要管理文件，而是管理文件块就可以了。

有利于数据的复制。在HDFS系统中，一个block一般会复制3份

对于文件内容而言，一个文件的长度大小是size，那么从文件的０偏移开始，按照固定的大小，顺序对文件进行划分并编号，划分好的每一个块称一个Block。HDFS默认Block大小是128MB，以一个256MB文件为例，共分为256/128=2个Block.

不同于普通文件系统的是，HDFS中，如果一个文件小于一个数据块的大小，并不占用整个数据块存储空间

## NameNode

NameNode维护着HDFS中的元数据信息，包括文件和Block之间关系的信息、Block备份数量信息、Block和DataNode之间的关系信息，数据格式参照如下：

FileName replicas block-Ids id2host

例如： /test/a.log , 3 , {b1,b2} , [{b1:[h0,h1,h3]},{b2:[h0,h2,h4]}]

NameNode中的元数据信息存储在内存/文件中，内存中为实时信息，文件中为数据镜像作为持久化存储使用。

文件包括：

fsimage 元数据镜像文件。存储NameNode元数据信息，并不是实时同步内存中的数据。

edits 操作日志文件

fstime 保存最近一次checkpoit的时间

当有写请求时，NameNode会首先写editlog到磁盘edits文件中，成功后才会修改内存，并向客户端返回

所以，fsimage中的数据并不是实时的数据，而是在达到条件时再进行更新，更新过程需要SNN参与

NameNode的metadata信息会在启动后加载到内存中

## SecondaryNameNode

SecondaryNameNode并不是NameNode的热备份，而是协助者帮助NameNode进行元数据的合并，从另外的角度来看可以提供一定的备份功能，但并不是热备，这种合并过程可能会造成极端情况下数据丢失！可以从snn中恢复部分数据，但是无法恢复全部。

何时触发数据合并？

根据配置文件设置的时间间隔：fs.checkpoint.period 默认3600秒

根据配置文件设置的edits log大小 fs.checkpoint.size 默认64MB

合并过程(参看文档图)：

达到条件后 snn会将nn中的fsimage和edits文件拷贝过来，同时nn中会创建一个新的edits.new文件，新的读写请求会写入到这个edits.new中，在snn中将拷贝过来的fsimage和edits合并为一个新的fsimage，最后snn将合并完成的fsimage文件拷贝回nn中替换之前的fsimage，nn再将edtis.new改为edits

由于NameNode实时数据都在内存中，此处的合并指的是磁盘中的持久化的数据的处理。

判断：snn可以对元数据做一定程度的备份，但是不是热备，对不对？

思考：什么情况下可能造成NameNode元数据信息丢失?

snn并不是nn的热备，但是能保存大部分备份数据。原因就在于edits.new中的数据丢失了就找不回来了

通常NameNode和SNN要放置到不同机器中以此提升性能，并提供一定的元数据安全性。

## DataNode

在hadoop中，数据是存放在DataNode上面的。是以Block的形式存储的。

DataNode节点会不断向NameNode节点发送心跳报告。

初始化时，每个数据节点将当前存储的数据块告知NameNode节点。

通过向NameNode主动发送心跳保持与其联系（3秒一次）

后续DataNode节点在工作的过程中，数据节点仍会不断的更新NameNode节点与之对应的元数据信息，并接受来自NameNode节点的指令，创建、移动或者删除本地磁盘上的数据块。

如果10分钟都没收到dn的心跳，则认为其已经lost，并copy其上的block到其他dn。

## Block副本放置策略：

第一个副本：放置在上传文件的DN，如果是集群外提交，就随机选择一台磁盘不太满，cpu不太忙的节点

第二个副本：放置在第一个副本不同机架的节点上

第三个副本：放置在与第二个副本相同机架的节点上

更多副本：随机节点

\*\*机架感知策略(参看文章)

# HDFS的shell操作

hadoop fs -mkdir /user/trunk

hadoop fs -ls /user

hadoop fs -lsr /user (递归的)

hadoop fs -put test.txt /user/trunk

hadoop fs -put test.txt . (复制到hdfs当前目录下，首先要创建当前目录)

hadoop fs -get /user/trunk/test.txt (复制到本地当前目录下)

hadoop fs -cat /user/trunk/test.txt

hadoop fs -tail /user/trunk/test.txt (查看最后1000字节)

hadoop fs -rm /user/trunk/test.txt

hadoop fs -rmdir /user/trunk

hadoop fs -help ls (查看ls命令的帮助文档)

# JavaAPI操作HDFS

Hadoop为我们提供了一套完整优秀的JavaAPI，整合了关于HDFS的所有操作，并且简单易学，非常好用。

package cn.tedu.hdfs;

import java.io.IOException;

import java.net.URI;

import java.net.URISyntaxException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.junit.Test;

/\*\*

\* HDFS:JavaPAI操作

\* @author dongcc

\*

\*/

public class HDFSTest {

/\*

\* 从HDFS中删除文件

\*/

@Test

public void delete() throws Exception{

Configuration conf = new Configuration();

FileSystem fs = FileSystem.get(new URI("hdfs://192.168.65.101:9000"), conf);

//3.删除文件

//fs.delete(new Path("/api/test"));//此方法已过时，相当于hadoop fs -rmr 既强大又危险，一般不推荐使用

fs.delete(new Path("/api/1.txt"), false);

fs.close();

}

/\*

\* 从HDFS中下载文件到本地

\*/

@Test

public void get() throws Exception{

Configuration conf = new Configuration();

FileSystem fs = FileSystem.get(new URI("hdfs://192.168.65.101:9000"), conf);

//3.下载文件到本地

fs.copyToLocalFile(new Path("/api/2.txt"), new Path("2.txt"));

//上述方法如若报错，请使用以下方法。

//fs.copyToLocalFile(false, new Path("/api/2.txt"), new Path("2.txt"), true);

fs.close();

}

/\*

\* 从本地上传文件到HDFS中

\*/

@Test

public void put() throws Exception{

Configuration conf = new Configuration();

//使用set方法来设置配置文件中的配置项，.set（属性名，属性值）

conf.set("dfs.replication", "1");

FileSystem fs = FileSystem.get(new URI("hdfs://192.168.65.101:9000"), conf);

//3.上传文件到HDFS中

fs.copyFromLocalFile(new Path("1.txt"), new Path("/api/2.txt"));

fs.close();

}

/\*

\* 在HDFS上创建文件夹

\*/

@Test

public void mkdir() throws Exception{

//1.创建配置类

Configuration conf = new Configuration();

//2.创建文件系统类

FileSystem fs = FileSystem.get(new URI("hdfs://192.168.65.101:9000"), conf);

//3.创建文件夹

fs.mkdirs(new Path("/api/test"));//该方法可以同时创建多级目录

//4.关闭连接

fs.close();

}

}

# 附录

|  |  |
| --- | --- |
| **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 | 离开hadoop安全模式    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。  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。    当启动报如下错误时:  org.apache.hadoop.dfs.SafeModeException: Cannot delete /user/hadoop/input. Name node is in safe mode |
| 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  计算机生成了可选文字: 'park/1.txt 4 bytes, 1 block { 蚂 ：  € · BP 一 18315787e5 一 1g2 · 1b8 · 234 · 1g9 一 144g2831598g4 ： blk 1e7374182b 1e92 repl=l ['default 一 rack/19  2 · 1b8 · 234 · 1g9 ： 5991e ]  Status ： 《 剡 一 T 卜 丫  Total  S IZ e ：  Total  dirs ：  Total  files ： 1  Total  s 声 1 ks ：  Total  blocks { v e 引 ：  Minimally replicated blocks ：  Over- replicated blocks ：  Under-replicated blocks ：  s 一 replicated blocks ：  Default replication factor ：  Average block r 1 at n ：  Co rrupt blocks ：  Missing replicas ：  Number of data-nodes  Number of racks ：  (avg.  1 { 1  € (€.6  € (€.6  block size 4 引  FSCK ended at Fri Dec €4 21 ： 39 ： 42  PST 2e15 in 12 milliseconds |

|  |  |
| --- | --- |
| 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 |