

一、入门

1.简介

HDFS: 全称Hadoop Distributed File System 中文hadoop分布式文件系统

说明: HDFS是hadoop内的一个子技术

作用: 解决海量数据存储问题

特点: 分布式文件存储系统(多台计算机联合存储) 突破单体服务器的存储瓶颈



单体文件系统



分布式文件系统

2.HDFS体系架构

HDFS核心进程

NameNode

补充: 一个HDFS集群的主节点, 一个集群只有1个.

1. 基于内存存储管理文件的元数据信息。(NameNode内存要求高。)

文件名	类型	大小	权限	用户	组
ceshi.log	文件	500MB	rw-rw-rw-	root	root

2. 是HDFS集群的管理者master: 管理集群中所有的datanode。

datanode12	ip地址	磁盘容量	磁盘使用情况
datanode13	ip地址	磁盘容量	磁盘使用情况

目的: 掌握datanode健康状况, 了解磁盘容量, 数据分布的负载均衡。
均衡使用datanode的磁盘空间。

集合多个datanode服务器的网络带宽, 提高数据传输速度。

3. 接受客户端文件操作的请求(文件元数据操作请求)。

4. NameNode存储了文件拆分后的block分布信息:

block0	--[ip1,ip2]	--起始位置	--大小	--checksum
block1	--所在dn的ip	--起始位置	--大小	--checksum

DataNode

补充: HDFS集群的从节点, 一个集群有多个.

1. 管理存储数据文件切分后的block(128MB), 存放硬盘上。廉价机器。

2. 是HDFS的从机, slave

heartBeat: 定期向namenode发送心跳(3s), 告知datanode(ip 磁盘容量), 如果超过10分钟, 无心跳, 则认为NameNode死亡。

blockreport: 定期上报datanode中的存储的block的信息:

dn1, 发送 dn1上存储的所有文件的block的信息, 如果NameNode没有收到的block信息, 则判断该block在该dn上

失效。

3. 接收数据datablock上传下载的客户请求。

HDFS核心概念

block: 文件切分后的数据块。

大小默认128MB

原因: 现有服务器机房局域网网络带宽千兆带宽==125MB/s

说明:

block过大: 导致单个block通过磁盘和网络读取时间过长, 影响该部分数据的处理速度, 并且增加单个block传输的失败几率, 重试成本过高, 浪费IO资源。

block过小: block个数过多, 导致namenode内存过度占用, 导致不足。

结论: 100MB/s的带宽, block设置为128MB

220MB/s左右的带宽, block设置为256MB。

replication:副本(副本因子)

每个block在hdfs的datanode会存储多份。默认replication=3, 每个block有3分。

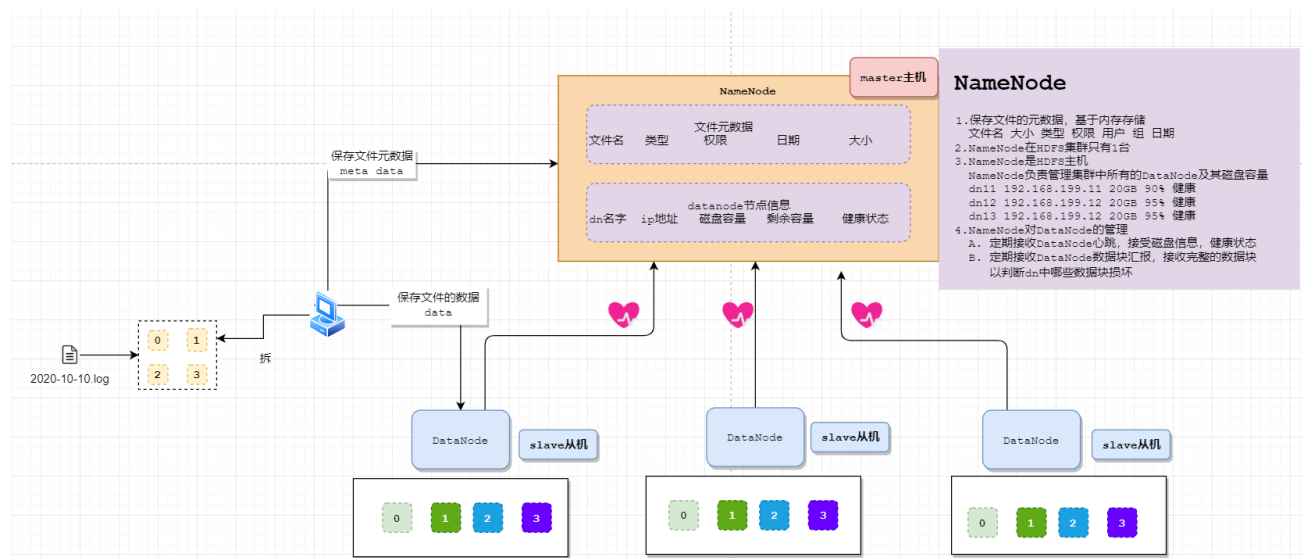
原因: 防止datanode因为单点故障, 导致数据丢失。

实战参数: 一般block的replication就是3个。

checksum: 校验和

datanode定期, 计算本节点存储的block的checksum, 判断是否和之前的checksum保持一致。

说明: datanode掌握block文件是否损坏的判断手段。



Block 数据块

概念：文件拆分后的数据单元，也是DataNode保存的数据单元

大小默认：128M (hadoop2.x) ，老版本hadoop1.x默认为64M

HDFS的block大小原则：

block过大：客户端和DataNode传输block的失败概率变高，导致block多次重传，导致最终传输效率低

block过小：一个文件拆分的block过小，导致过多的block描述信息，过多占用NameNode空间，导致内存占用率高

结论：结合目前常规硬盘的读写速度，大文件读取100M/s ~ 200M/s 和 网络千兆带宽 (125M/s)

设计block大小为128M，可以实现读取block的时间为1s

如果HDFS的内部网络和磁盘环境，可以实现200MB/s以上的读写速度，block大小可以设置为256MB/s (修改hdfs-site.xml)

Replication 副本因子

概念：每个block块，都会在HDFS提供一个备份，replication副本个数，默认值为 3

场景：如果datanode磁盘损坏，导致本机block丢失

解决：1.DataNode损坏块，在数据汇报的时候，就不会发送block信息

2.NameNode知道损坏的数据块，发送远程操作命令

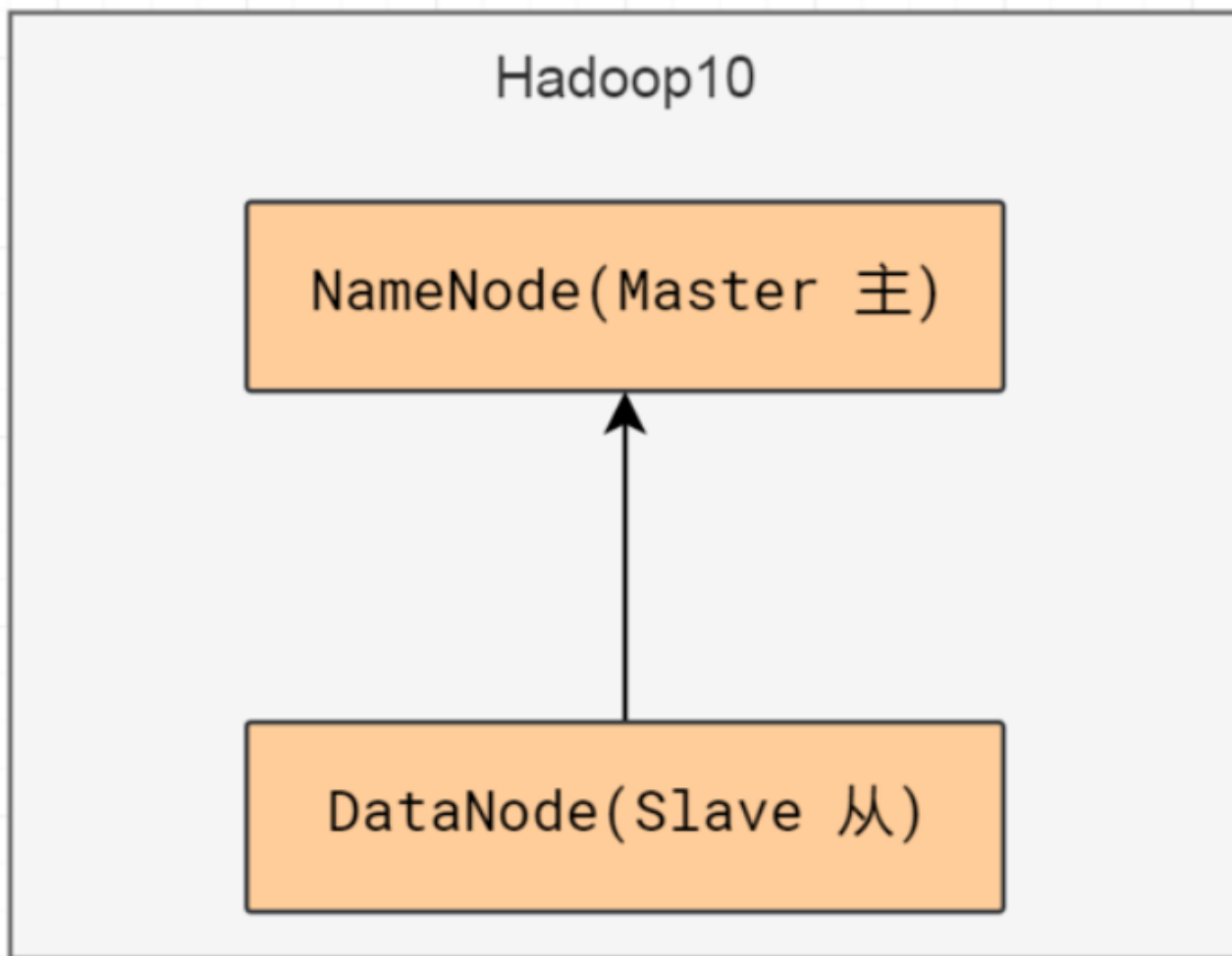
删除DataNode上损坏的block，从该block的副本所在datanode进行数据同步恢复

Checksum 校验和

概念：Datanode中保存数据block的数字指纹

作用：DataNode保存block块，同时保存block数字指纹，定期计算block块的数字指纹，和已存数字指纹对比，验证block块是否完整

二、安装（伪分布式）



- 服务器准备

```
# 1. 准备虚拟机hadoop10
# 1. 设置hostname
hostnamectl set-hostname hadoop10
# 2. 配置hosts(linux+windows)
vim /etc/hosts
-----以下是文件信息-----
192.168.199.8 hadoop10
补充:
    一定要配置windows对集群中所有节点的映射关系。
# 3. 关闭防火墙
systemctl stop firewalld #关闭防火墙
systemctl disable firewalld # 禁止防火墙开机启动。
# 4. 安装jdk1.8
[root@hadoop10 modules]# tar -zxvf jdk-8u171-linux-x64.tar.gz -C /opt/installs/
[root@hadoop10 installs]# mv jdk1.8.0_171/ jdk1.8
# 5. 配置jdk环境变量。
[root@hadoop10 installs]# vim /etc/profile
# JAVA
# JAVA_HOME
export JAVA_HOME=/opt/installs/jdk1.8/
# PATH

export PATH=$PATH:/opt/installs/jdk1.8/bin/
```

加载配置

```
source /etc/profile
```

验证

```
java
```

- 安装

1.解压、配置环境变量

```
# 解压
[root@hadoop10 modules]# tar zxvf hadoop-2.9.2.tar.gz -C /opt/installs/
[root@hadoop10 installs]# mv hadoop-2.9.2 hadoop2.9.2
# 配置环境变量
vim /etc/profile
-----以下是环境变量-----
# 配置HADOOP_HOME
export HADOOP_HOME=/opt/installs/hadoop2.9.2
# 配置PATH
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
# 生效配置信息(重新执行profile中的指令, 加载配置信息)
source /etc/profile
```

```
# hadoop目录结构
[root@hadoop10 ~]# tree -L 1 /opt/installs/hadoop2.9.2/
/opt/installs/hadoop2.9.2/
├─ bin # hadoop客户端操作相关的脚本程序, hdfs、hadoop、yarn    重要
├─ etc # 配置目录xml、文本文件                                重要
├─ include # 一些C的头文件, 无需关注
├─ lib # 第三方native实现C实现
├─ libexec # hadoop运行时候, 加载配置的脚本
├─ LICENSE.txt
├─ logs # 系统运行日志目录, 排查故障!                        重要
├─ NOTICE.txt
├─ README.txt
├─ sbin # hadoop服务器端操作相关脚本, 通常用于启动服务例如: start|top-dfs.sh 重要
└─ share # hadoop运行的依赖jars、内嵌webapp
```

2.初始化配置文件(\$HADOOP_HOME/etc/hadoop)

```
hadoop-env.sh - hadoop环境配置(jdk)
core-site.xml - Hadoop核心配置文件
hdfs-site.xml - HDFS的个性化配置文件 副本因子
slaves        - 在哪个节点启动datanode
```

```
# hadoop-env.sh
```

```
# jdk安装目录
JAVA_HOME=/opt/installs/jdk1.8

# core-site.xml
# 配置hdfs入口
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://hadoop10:9000</value>
</property>
# 配置 数据保存位置 (hadoop的根目录下新建data目录,会自动创建)
<property>
  <name>hadoop.tmp.dir</name>
  <value>/opt/installs/hadoop2.9.2/data</value>
</property>

# hdfs-site.xml
# 配置副本个数
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>

<property>
  <name>dfs.namenode.secondary.http-address</name>
  <value>hadoop10:50090</value>
</property>
<property>
  <name>dfs.namenode.secondary.https-address</name>
  <value>hadoop10:50091</value>
</property>

# slaves (建议使用vi编辑)
# 配置从机datanode的ip
hadoop10
```

3.格式化HDFS(第一次安装HDFS 格式化文件系统)

```
# 一旦hadoop配置启动失败，清空data下的文件，再重新格式化。
#初始化namenode和datanode存放数据的目录
hdfs namenode -format
```

4.启动HDFS

```
# 启动hdfs
start-dfs.sh
# 关闭hdfs
stop-dfs.sh
```

5.验证

```
# 查看hdfs进程
[root@hadoop10 installs]# jps
2225 NameNode # master namenode主机
4245 Jps
2509 SecondaryNameNode
2350 DataNode # slave datanode从机

# 查看hdfs Web服务
1. 查看namenode的web服务
  http://hadoop10:50070

2. 查看datanode的Web服务
  http://hadoop10:50075
```

搭建知识补充:

1.日志查看 (查看hdfs运行异常)

```
# namenode启动运行日志
hadoop-用户名-namenode-主机名.log

# datanode启动日志
hadoop-用户名-datanode-主机名.log
```

2.HDFS配置错误修正

```
# 1. 关闭启动的hdfs程序(NN DN)

# 2. 修改错误的配置文件。

# 3. data目录清空, 重新格式化
hdfs namenode -format
```

场景: 格式化或者启动hadoop失败。

说明:

hadoop/data文件夹

作用: 保存datanode和namenode持久化的数据。




时机:

1. 格式化hdfs namenode -format 会初始化该目录下的文件。
 2. hdfs运行期间产生的数据, 会操作该目录中的数据。
- 必要操作: 删除格式化或者启动数据保存的文件目录。





3.web界面

NameNode的web管理器地址: http://ip:50070.

Browse Directory


Go!   

Show entries Search:


	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
	-rw-r--r--	root	supergroup	182.05 MB	Oct 21 16:14	1	128 MB	jdk-8u171-linux-x64.tar.gz	

Showing 1 to 1 of 1 entries

Previous **1** Next

File information - jdk-8u171-linux-x64.tar.gz 

[Download](#) [Head the file \(first 32K\)](#) [Tail the file \(last 32K\)](#)

Block information -- 

[Block 0](#)
[Block 0](#)
[Block 1](#)

Block ID: 1073741829
Block Pool ID: BP-149113955-192.168.199.10-1603267682217
Generation Stamp: 1001
Size: 134217728
Availability:

- hadoop10

Close

三、HDFS客户端操作

1.HDFS命令

1. 命令所在目录
\${hadoop}/bin之下/hdfs

2. HDFS的文件系统结构
和Linux类似。

3. 命令格式
hdfs dfs -xxx -参数

2. 常见命令

命令语法	含义	示例代码
hdfs dfs -ls [-R] hdfs文件路径	查看文件元数据信息	hdfs dfs -ls /
hdfs dfs -mkdir -p /目录a/目录b	新建文件夹，如果父目录不存在则添加-p参数	hdfs dfs -mkdir -p /hdfs/file
hdfs dfs -put linux文件路径 hdfs目录	文件上传	hdfs dfs -put /opt/models/jdk /hdfs
hdfs dfs -get hdfs文件 linux目录	文件下载	hdfs dfs -get /hdfs/jdk1.8 /opt
hdfs dfs -cat hdfs文档路径	查看文件内容	hdfs dfs -cat /hdfs/Test.java
hdfs dfs -rm hdfs文件路径	删除文件	hdfs dfs -rm /hdfs/Test.java
hdfs dfs -rm -r hdfs文件夹	删除文件夹，非空使用-rmr	hdfs dfs -rm -r /hdfs
hdfs dfs -chmod [-R] 权限运算值 hdfs文件 hdfs dfs -chmod [-R] u+x hdfs文件	修改hdfs文件权限	hdfs dfs -chmod o+w /hdfs
hdfs dfs -appendToFile linux本地A文件 HDFS远程B文件	将A文件内容追加到HDFS的B文件的末尾。	hdfs dfs -appendToFile /etc/profile /Test1.java
hdfs dfs -mv /hdfs/demo1/wordcount1.log /hdfs/demo2	移动HDFS文件系统内部文件	hdfs dfs -mv /hdfs/demo1/wordcount1.log /hdfs/demo2

3. Java操作HDFS

- 核心API

API	含义和作用
Configuration	配置信息，封装hdfs操作的相关配置文件信息
FileSystem	HDFS的分布式文件系统工具，操作HDFS文件。
PATH	表示操作路径(new Path("/hdfs/a.log"))

- windows开发环境准备(开发和测试需要)

```
# 1. 将hadoop2.9.2的软件，解压到window中，路径不能有中文，不能有空格。
# 2. 拷贝hadoop的windows执行环境工具到bin路径下，替换客户端工具。
# 3. 配置环境变量
HADOOP_HOME=hadoop安装路径
PATH=hadoop安装路径/bin
# 4. 重启IDEA
```

- 添加log4j.properties
- HDFS依赖

```
<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-client</artifactId>
  <version>2.9.2</version>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
</dependency>
```

- 编程步骤(文件上传)

```
/**
 * 需求：向hdfs集群上传一个文件？
 *
 * 命令：hdfs dfs -put 本地文件路径 hdfs远程路径位置。
 */
@Test
public void test1() throws IOException {
    // 1. 初始化配置
    Configuration conf = new Configuration();
    conf.set("fs.defaultFS", "hdfs://hadoop10:9000");
    // 2. 获得操作hdfs的客户端。
    FileSystem fs = FileSystem.get(conf);
    // 使用客户端的方法(命令)，操作hdfs
    fs.copyFromLocalFile(new Path("D:/xxx.txt"), new Path("/data"));
}
```

```

// 3. 关闭资源。
if (fs != null) {
    fs.close();
}
}

```

异常：Permission denied: user=Administrator, access=WRITE, inode="/hdfs":root:supergroup:drwxr-xr-x

解决办法：为上传的hdfs目录添加写w权限 `hdfs dfs -chmod 777 /hdfs`

- 其他API操作

```

/**
 * 需求：从hdfs中下载 /hdfs/demo.tar.gz 文件?
 * 命令：hdfs dfs -get /hdfs/demo.tar.gz D:/
 */
@Test
public void test2() throws IOException {
    //1. 初始化配置文件
    Configuration conf = new Configuration();
    conf.set("fs.defaultFS", "hdfs://hadoop10:9000");

    //2. 获得hdfs操作客户端
    FileSystem fs = FileSystem.get(conf);
    fs.copyToLocalFile(new Path("/hdfs/demo.tar.gz"), new Path("D:/demo.tar.gz")); // 拷贝到本地。
    //
    //3. 关闭资源
    fs.close();
}

//返回值boolean, 是否创建成功。
boolean isok = fileSystem.mkdir(new Path("/hdfs/test1"));

//判断文件是否存在
boolean isexist = fileSystem.exists(new Path("/hdfs/test1"));

/**
删除文件
参数1：远端hdfs的文件路径
参数2：是否递归删除，如果给false，删除无法递归则会抛出异常
返回值：是否删除成功
*/
Boolean deleteOK = fileSystem.delete(new Path("hdfs文件路径"), true);

// 获得目录下所有文件和目录的元数据== hdfs dfs -ls /hdfs
FileStatus[] fileStatuses = fs.listStatus(new Path("/hdfs"));
for (FileStatus file : fileStatuses) {
    //1. 获得文件路径
    Path path = file.getPath();
}

```

```

//2. 获得文件权限
FsPermission permission = file.getPermission();
//3. 获得文件副本数
short replication = file.getReplication();
//4. 获得文件修改时间,long的时间戳
long modificationTime = file.getModificationTime();
//5. 获得文件大小, 单位B
long len = file.getLen();
}

/**
 * 读取文件信息及其block信息
 * 功能: 递归获得目录下的所有文件的元数据信息, 相当于hdfs的ls命令
 * listFiles参数1: 要查看的目录
 * listFiles参数2: 是否递归 -R
 */
RemoteIterator<LocatedFileStatus> files = fs.listFiles(new Path("/hdfs"), true);
while(files.hasNext()){
    //迭代获得每个file元数据
    LocatedFileStatus file = files.next();
    //1. 获得文件路径
    Path path = file.getPath();
    //2. 获得文件权限
    FsPermission permission = file.getPermission();
    //3. 获得文件副本数
    short replication = file.getReplication();
    //4. 获得文件修改时间,long的时间戳
    long modificationTime = file.getModificationTime();
    //5. 获得文件大小, 单位B
    long len = file.getLen();
    //6. 获得block的切片分布信息
    BlockLocation[] blockLocations = file.getBlockLocations();
}

```

四、HDFS原理相关

1.Trash回收站

HDFS为了规避由于用户的误操作, 导致的数据删除丢失, 用户可以在构建HDFS的时候, 配置HDFS的垃圾回收功能。所谓的垃圾回收, 本质上是在用户删除文件的时候, 系统并不会立即删除文件, 仅仅是将文件移动到垃圾回收的目录。然后更具配置的时间, 一旦超过该时间, 系统会删除该文件, 用户需要在到期之前, 将回收站的文件移除垃圾站, 即可避免删除

```
<!--开启垃圾回收,需要在core-site.xml中添加如下配置,然后重启hdfs即可-->
<!--垃圾回收,1440 minites == 1天-->
<property>
  <name>fs.trash.interval</name>
  <value>1440</value>
</property>
```

验证(日志会记录文件删除的回收站的位置)

```
[root@hadoop10 ~]# hdfs dfs -rm -r /Test1.java
20/11/13 18:28:50 INFO fs.TrashPolicyDefault: Moved: 'hdfs://hadoop10:9000/Test1.java' to trash
at: hdfs://hadoop10:9000/user/root/.Trash/Current/Test1.java
```

恢复回收站数据(本质上就是移动文件)

```
hdfs dfs -mv /user/root/.Trash/Current/Test1.java /
```

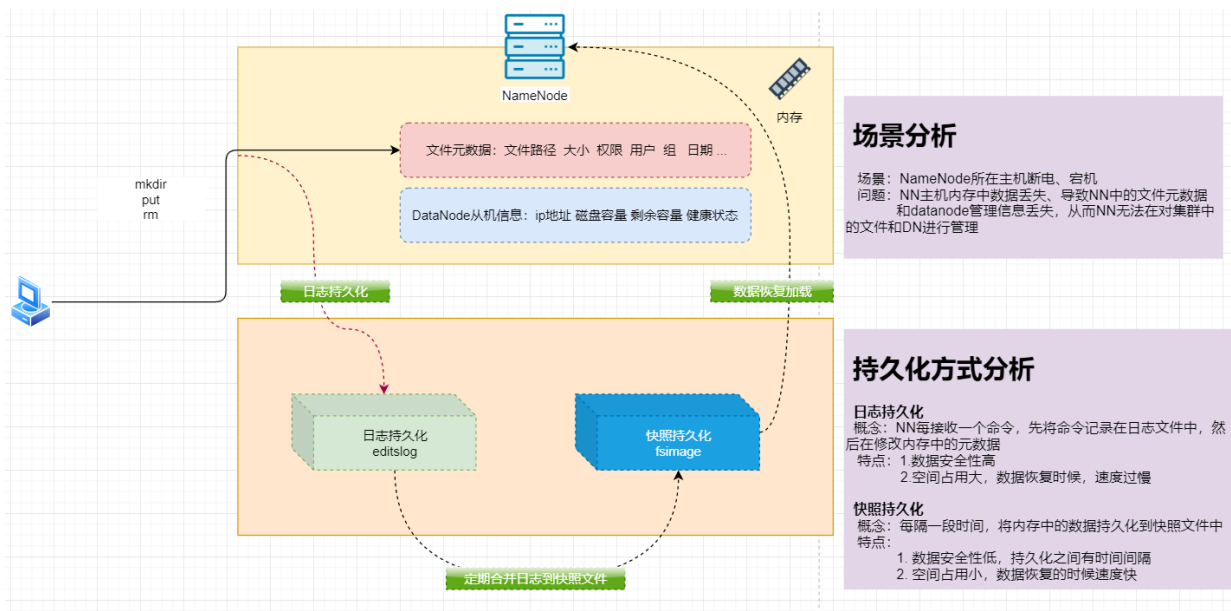
2.NameNode持久化

- 场景引入

问题: NameNode宕机, 导致内存中的文件元数据丢失怎么办?

解决: NameNode会将内存中的元数据持久化到磁盘中。

- 持久化方案分析

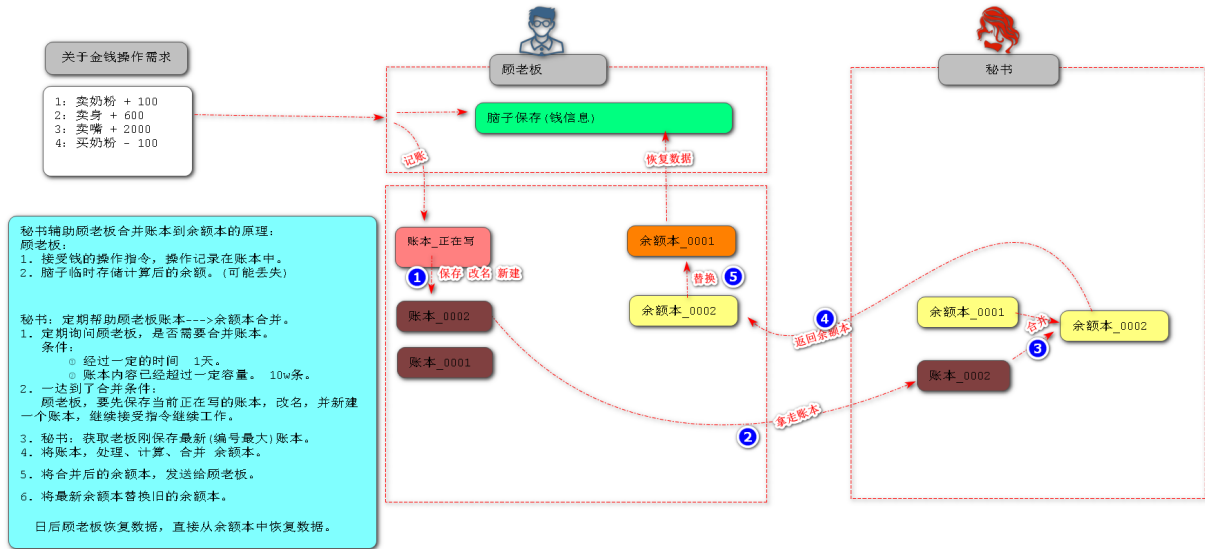


- 持久化结论

- # 1. HDFS接受客户端的文件操作后。
- # 2. 先将操作的命令 以日志的方式记录到editslog中。
- # 3. 然后再将指令对应的文件元数据的修改操作, 修改内存中的元数据信息。
- # 4. SNN定期负责将editslog中的文件合并到fsimage中。

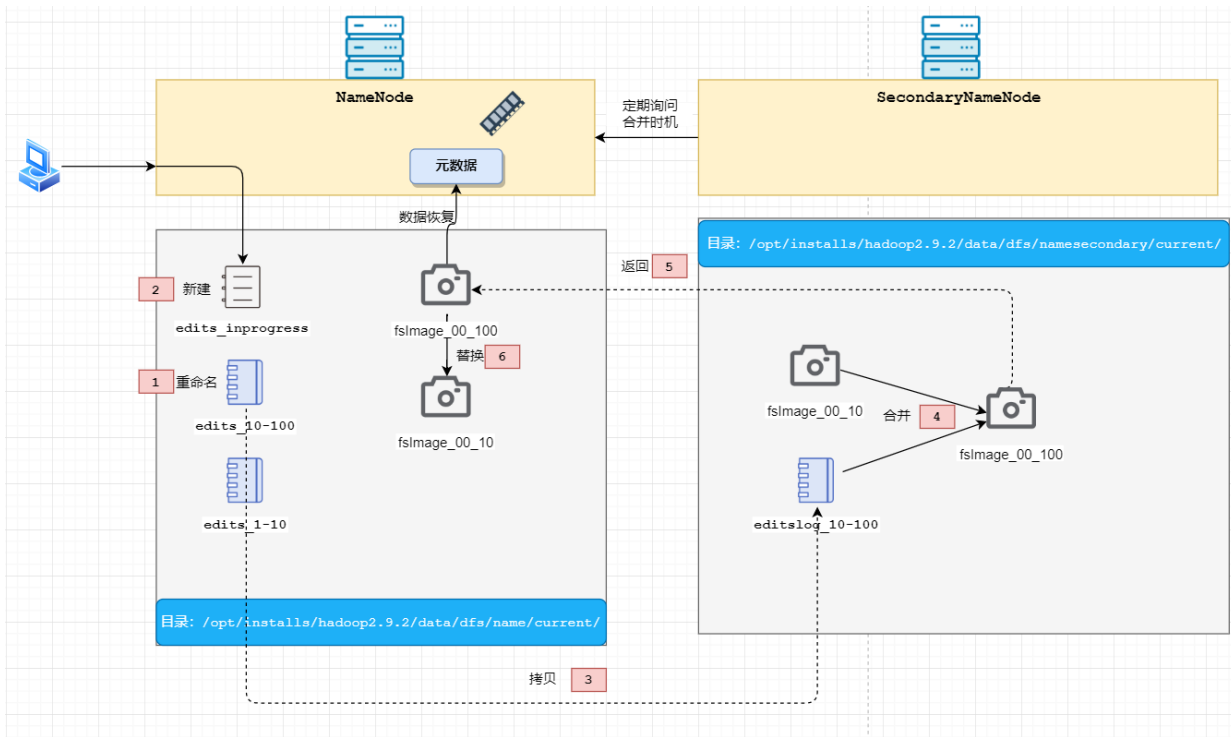
3.checkpoint机制

- 生活案例



- SNN的checkpoint工作机制

1. SecondaryNameNode向NameNode发起合并请求
2. NameNode将当前的Editslog文件保存改名edits, 并新建EditsLog继续持久化工作。
3. 将改名后的edits文件和本地的FSImage(旧)发送给sencondaryNameNode
4. SecondaryNameNode负责将FSImage(旧)+edits文件合并成FSImage(新)
5. 将新的FSImage(新)发送给NameNode保存。



SNN工作原理

SNN检查合并日志文件条件

1. SNN定期询问NN，是否达到了合并editslog条件
2. 合并条件
 - ① editslog是否已经持续写入1小时
 - ② editslog是否已经持续记录10w条记录

辅助NN做editslog为fsimage过程 (checkpoint检查点)：

1. NN： 将当前正在编写的editslog文件保存，改名：edits_01-10
2. NN： 创建一个新的edislog文件，edis_inprogress_11，继续接收客户端请求
3. SNN： 拷贝NN的最新的edits_01_10
4. SNN： 将最新的editslog和旧的fsimage，合并成最新的fsimage
5. SNN： 将合并后的fsimage，返回给NN
6. NN： 将最新的fsimage，替换旧的fsimage，旧的fsimage删除即可

- checkpoint触发条件(时机)

每1分钟检查一次触发条件。(SNN没隔1分钟，访问一次NN)

1. 每隔1小时触发一次checkPoint
2. 每100w次操作，触发一次checkpoint

下面的配置信息对应hdfs-site.xml

name	value默认	含义
dfs.namenode.checkpoint.period	3600	3600秒触发一次，数据合并。checkpoint
dfs.namenode.checkpoint.txns	1000000	100w次操作触发一次
dfs.namenode.checkpoint.check.period	60	1分钟检查一次操作次数

- SecondaryNameNode节点定制(集群中讲)

SNN和NN在一个服务器是上，存在单点故障。

场景：一旦服务器磁盘崩坏，持久化的数据就会全部丢失。

解决：SNN 和NN放在不同的服务器上。

```
# 1. 配置hdfs-site.xml中的SNN的节点地址
<property>
  <name>dfs.namenode.secondary.http-address</name>
  <value>hadoop10:50090</value>
</property>
<property>
  <name>dfs.namenode.secondary.https-address</name>
  <value>hadoop10:50091</value>
</property>

# 2. 将修改hdfs-site的配置同步到其他节点。

# 2. 单独启动SecondaryNameNode
hadoop-deamon start secondarynamenode
```

4.HDFS启动流程-NameNode安全模式(SafeMode)

时机

HDFS启动过程中，会进入一个特殊状态（SafeMode），该状态下NameNode暂停接受客户端文件修改操作（只能接收文件读操作）

HDFS启动流程-启动过程自动完成。

1. NameNode启动，加载最新的fsimage恢复数据，并加载未合并的editslog_inprogress，进一步恢复数据。--- NN管理内存数据完整。

2. 等待接受DataNode的心跳 HeartBeat

DN的本节点地址 健康状态 磁盘容量 剩余容量 版本号。

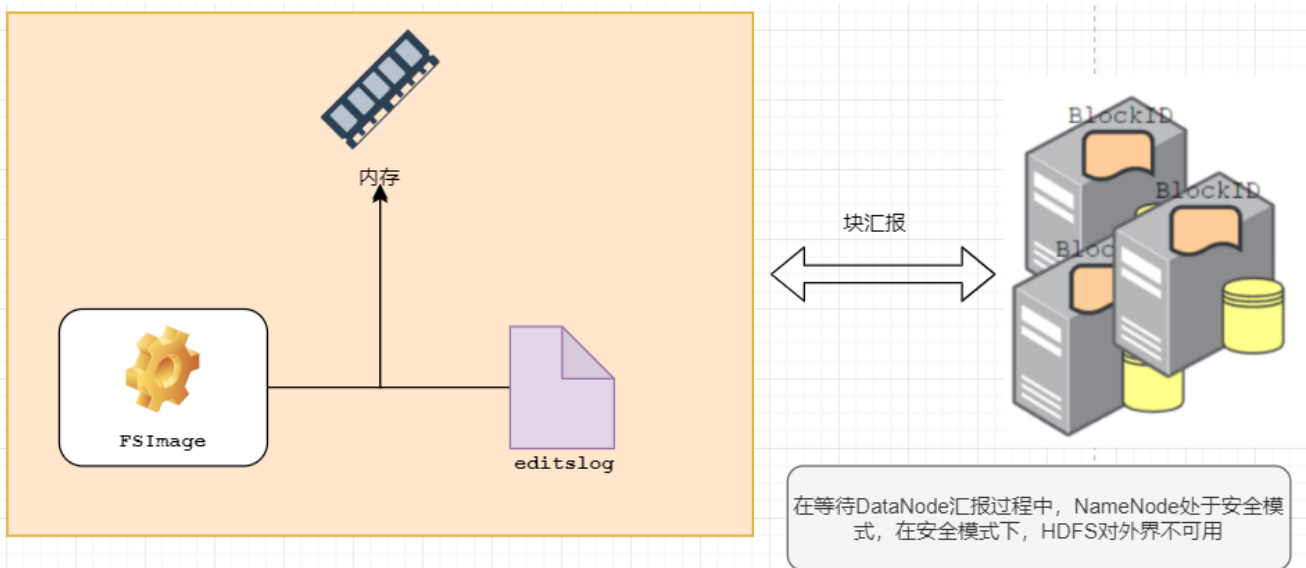
3. 等待接受DataNode的块报告 Block Report，判断是否满足最小副本因子（默认值1 `dfs.namenode.replication.min`），达到了，则认为当前Block是安全的，完整的。

DN的本节点的全部Block的信息：block的id offset length。

4. NameNode发现HDFS集群中所有的block的安全（完整）比例是否达到99.9%（`dfs.namenode.safemode.threshold-pct`），如果达到，则立刻退出安全模式。

① 为什么要设置最小副本因子为1：只要有1块完整，block数据即是完整，没必要等全部完整。

5. HDFS退出安全模式，才能正常工作。



手动安全模式

理由：实际开发中为了对HDFS进行维护，会手动NameNode进入安全模式

注意：safemode安全模式下，只能对HDFS中文件进行读操作，不能进行写操作（上传 修改 删除 移动）

0. 查看hdfs安全模式状态

```
hdfs dfsadmin -safemode get
```

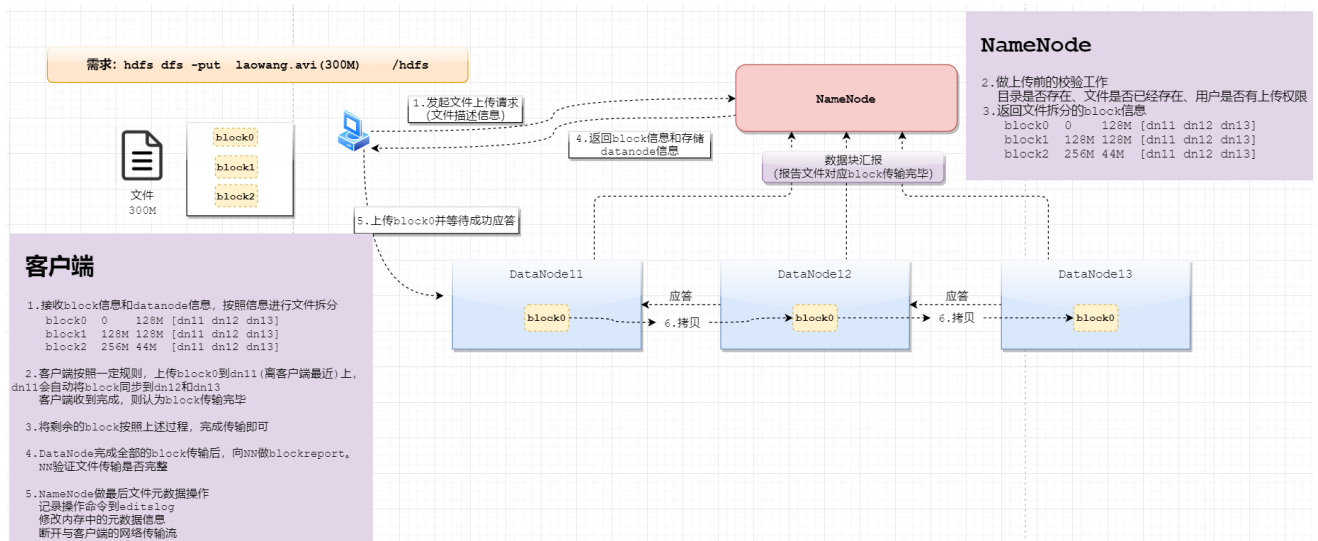
1. 进入安全模式

```
hdfs dfsadmin -safemode enter
```

2. 退出安全模式

```
hdfs dfsadmin -safemode leave
```

5.文件上传流程



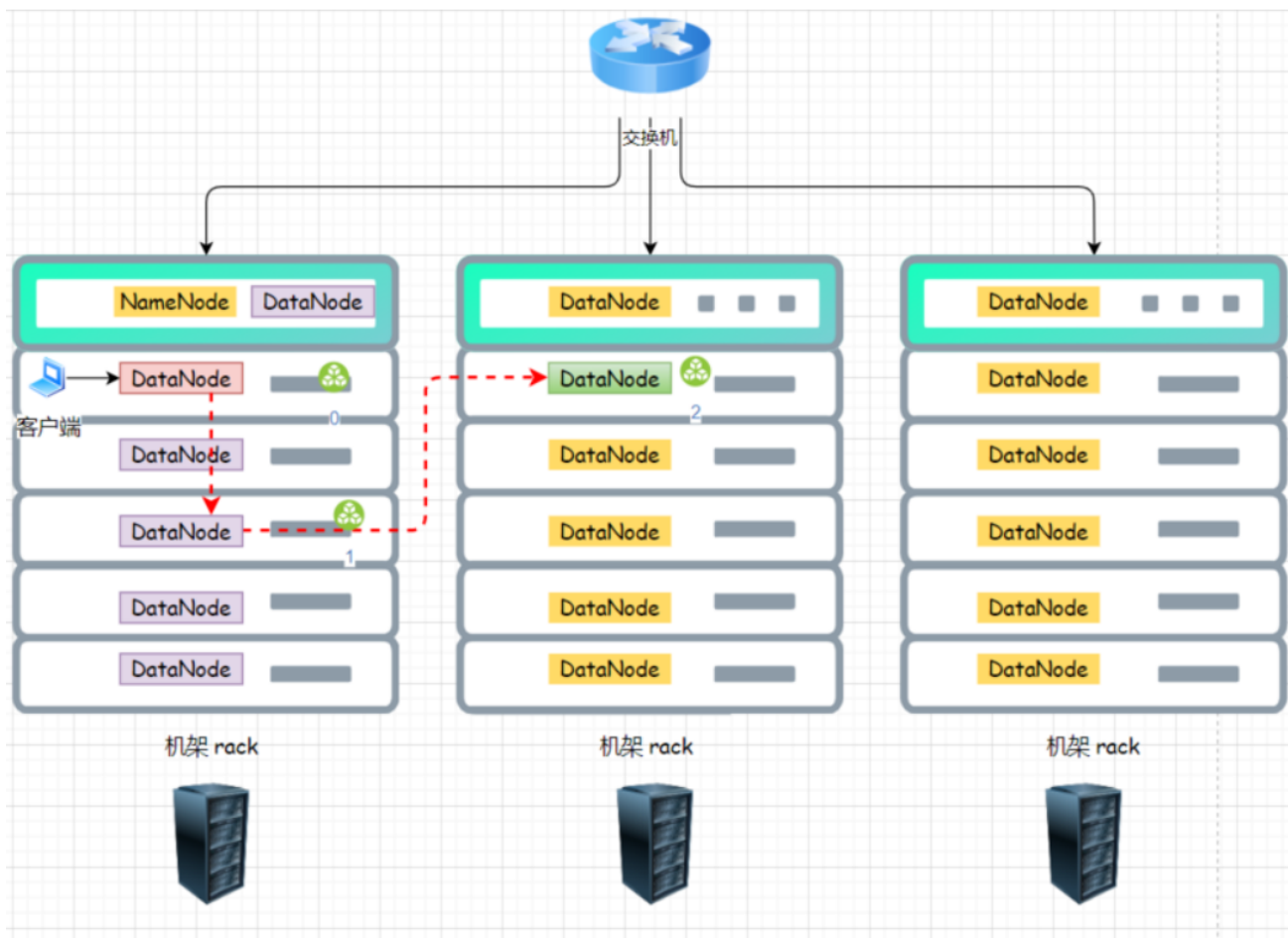
6.副本存放机制—机架感知(rack-aware)

考虑要素:

- HDFS集群的DN有很多台节点。
- 节点内部的网络数据传输, 速度最快。
- 机架内可以安装多台服务器, 节点之间网络带宽, 由于机架之间的网络带宽。

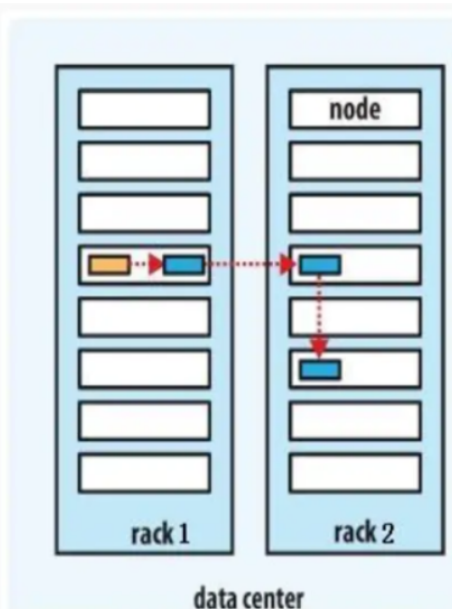
机架感知-副本存放策略(hadoop2.7.6以前)--旧版本

- 第一个block副本放在客户端所在的服务器的datanode中。
- 第二个block副本放置在本机架内的其它数据节点datanode上
- 第三个block副本放置在不同机架的随机某个节点上。(防止某个机架数据丢失)



副本存放策略(hadoop2.8.4以后)--新版本

1. 第一个block副本，放在client所在的节点
2. 第二个block副本，放在另一个机架上的某个节点上。
3. 第三个block副本，放在第二个机架的不同节点上。



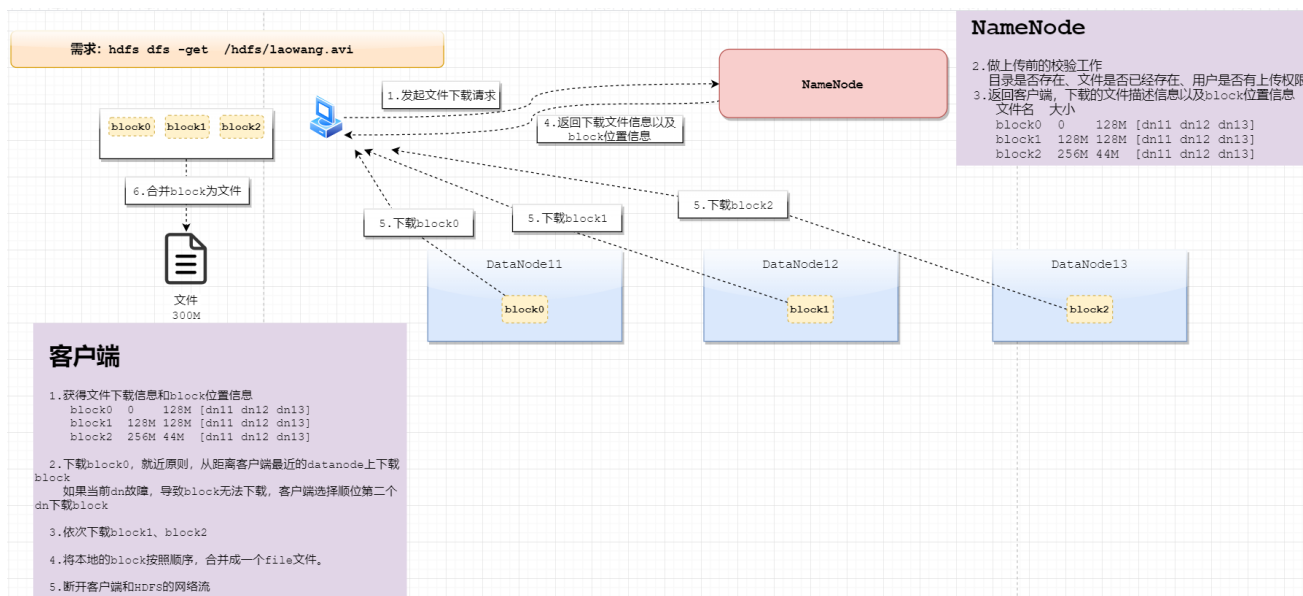
第一副本： 放置在上传文件的DataNode上；
如果是集群外提交，则随机挑选一台磁盘不太慢、CPU不太忙的节点上；

第二副本： 放置在于第一个副本不同的机架的节点上；

第三副本： 与第二个副本相同机架的不同节点上；

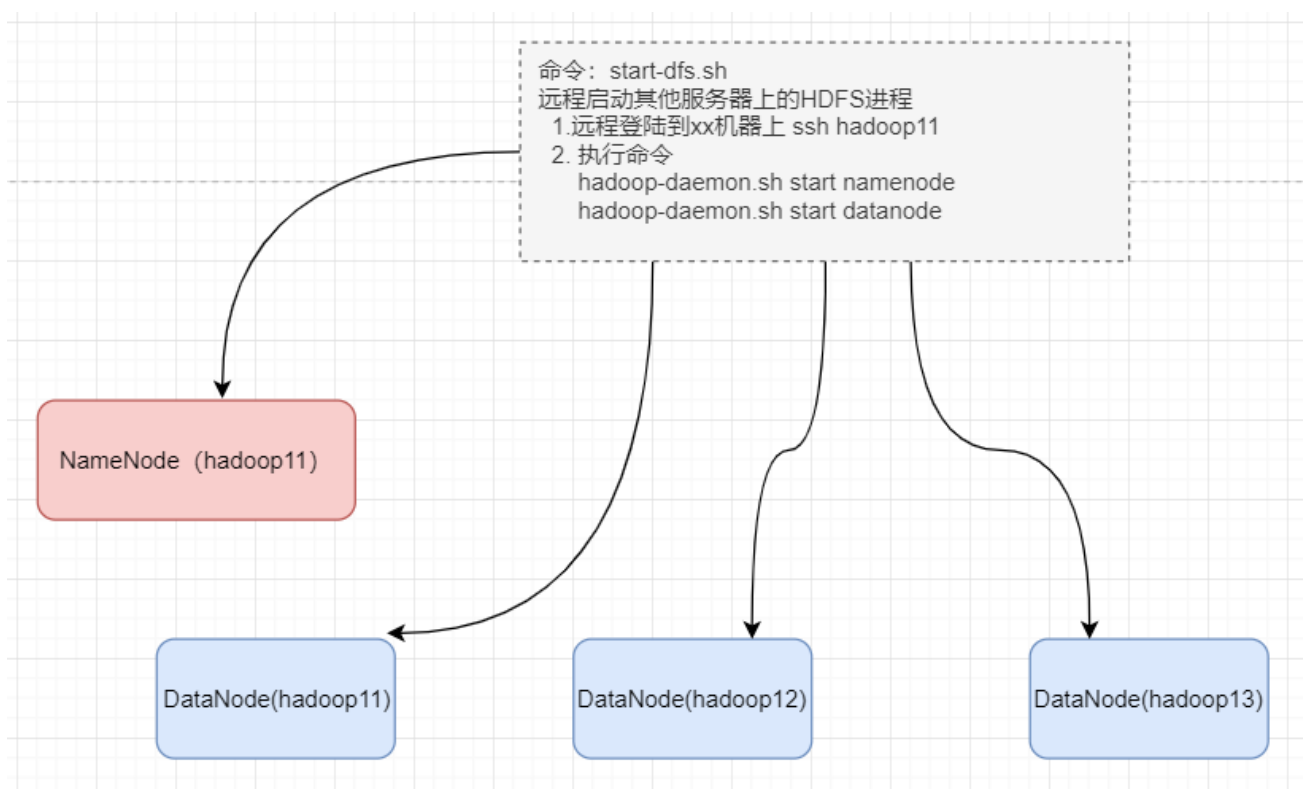
如果还有更多的副本： 随机放在节点中；

7.文件下载流程



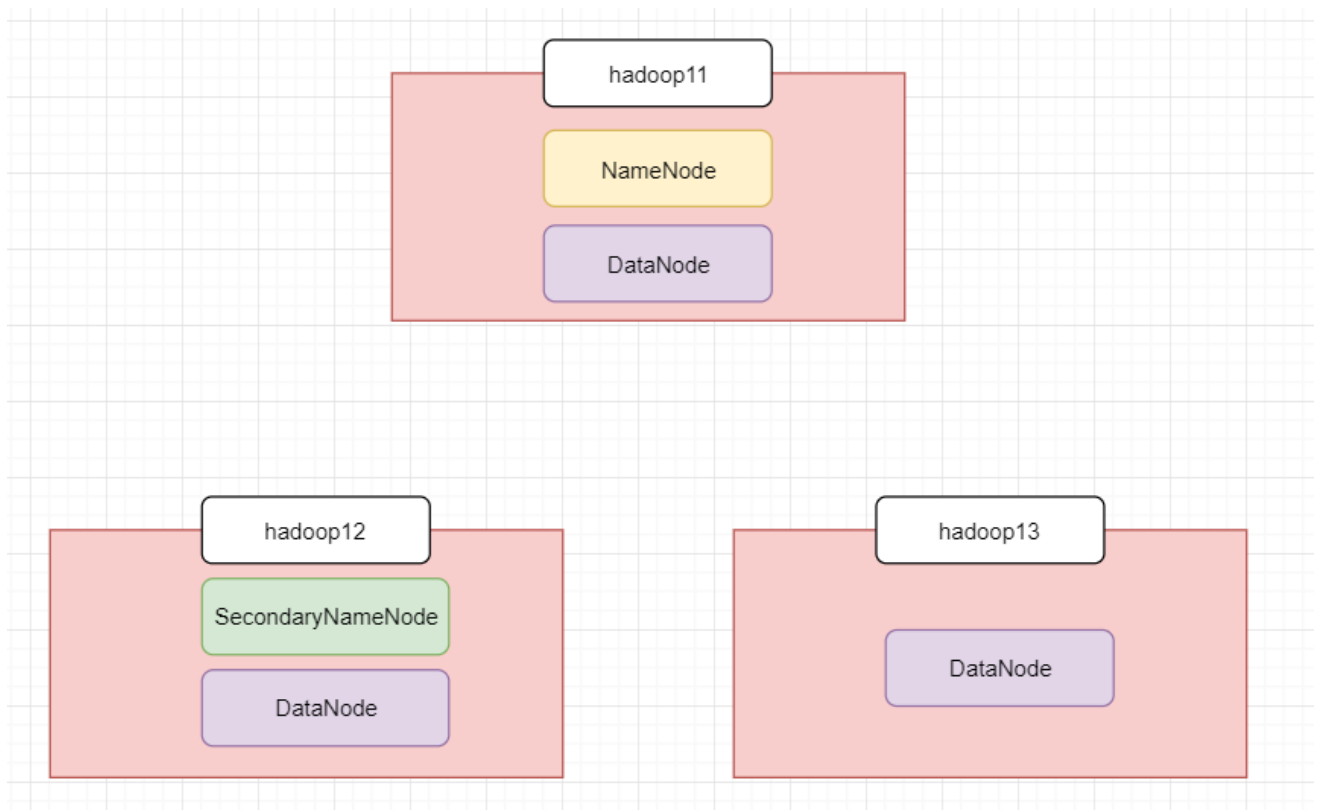
五、HDFS完全分布式集群搭建

1.SSH免密登录



2.HDFS分布式集群搭建

1.集群规划



2. 免密登录设置

1. 生成密钥

```
[root@hadoop11 hadoop]# ssh-keygen
```

2. 发送公钥到集群所有节点 hadoop11 hadoop12 hadoop13

```
[root@hadoop11 hadoop]# ssh-copy-id hadoop11
```

```
[root@hadoop11 hadoop]# ssh-copy-id hadoop12
```

```
[root@hadoop11 hadoop]# ssh-copy-id hadoop13
```

3. 验证免密登录效果

```
ssh root@hadoop13
```

3. 安装HDFS(1台)

1. 解压hadoop

2. 配置环境变量

```
HADOOP_HOME
```

```
PATH
```

3. 重新加载配置文件

4. 初始化配置文件(1台)

1. `hadoop-env.sh`
 `JAVA_HOME`
2. `core-site.xml`
 NN入口地址
 数据保存的data目录
3. `hdfs-site.xml`
 副本数
4. `slaves`（配置多个DN节点，最好使用vi编辑器）
 DN节点所在主机的IP

5.同步配置文件到其他节点

注意事项

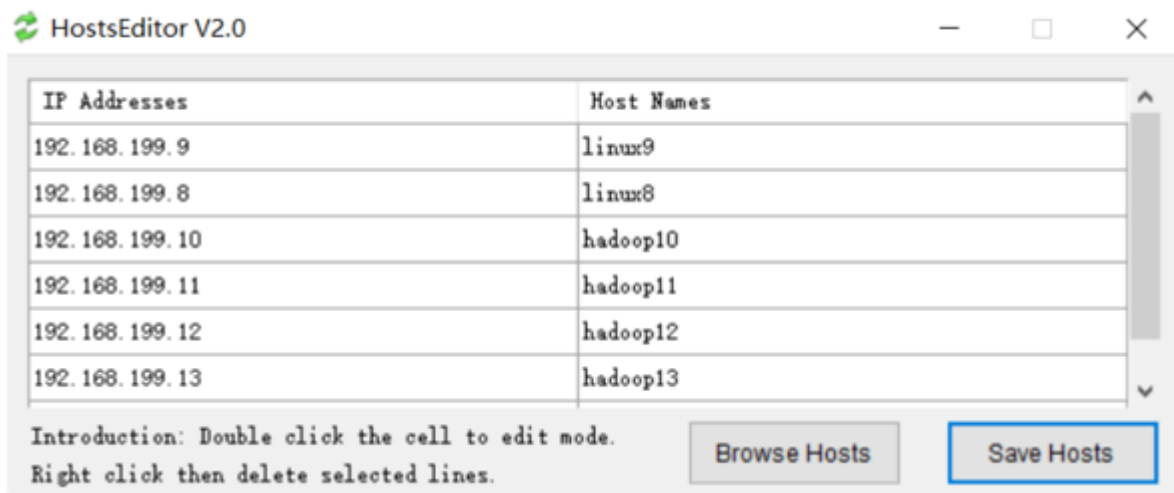
一个hadoop集群中，所有hadoop配置文件要完全一致。
将本机的hdfs配置同步到其他机器。

6.初始化HDFS集群

5. 初始化HDFS

1. 删除hadoop的data文件夹
2. 格式化集群
 在namenode节点执行格式化。
 `hdfs namenode -format`

7.配置windows访问HDFS的集群的ip映射



8.启动HDFS

在拥有免密登录权限的节点上执行：
`start-dfs.sh`

