HDFS 原理剖析

目录

1、HDFS 工作机制

工作机制的学习主要是为加深对分布式系统的理解，以及增强遇到各种问题时的分析解决能力，形成一定的集群运维能力

PS：很多不是真正理解 hadoop 工作原理的人会常常觉得 HDFS 可用于网盘类应用，但实际并非如此。要想将技术准确用在恰当的地方，必须对技术有深刻的理解

1.1、概述

1、HDFS 集群分为两大主要角色：namenode、datanode (secondarynamenode 和 client)
2、namenode 负责管理整个文件系统的元数据，并且负责响应客户端的请求
3、datanode 负责管理用户的文件数据块，并且通过心跳机制汇报给 namenode
4、文件会按照固定的大小(dfs.blocksize)切成若干块后分布式存储在若干台 datanode 上

5、每一个文件块可以有多个副本，并存放在不同的 datanode 上
6、datanode 会定期向 namenode 汇报自身所保存的文件 block 信息，而 namenode 则会负责保持文件的副本数量
7、HDFS 的内部工作机制对客户端保持透明，客户端请求访问 HDFS 都是通过向 namenode 申请来进行

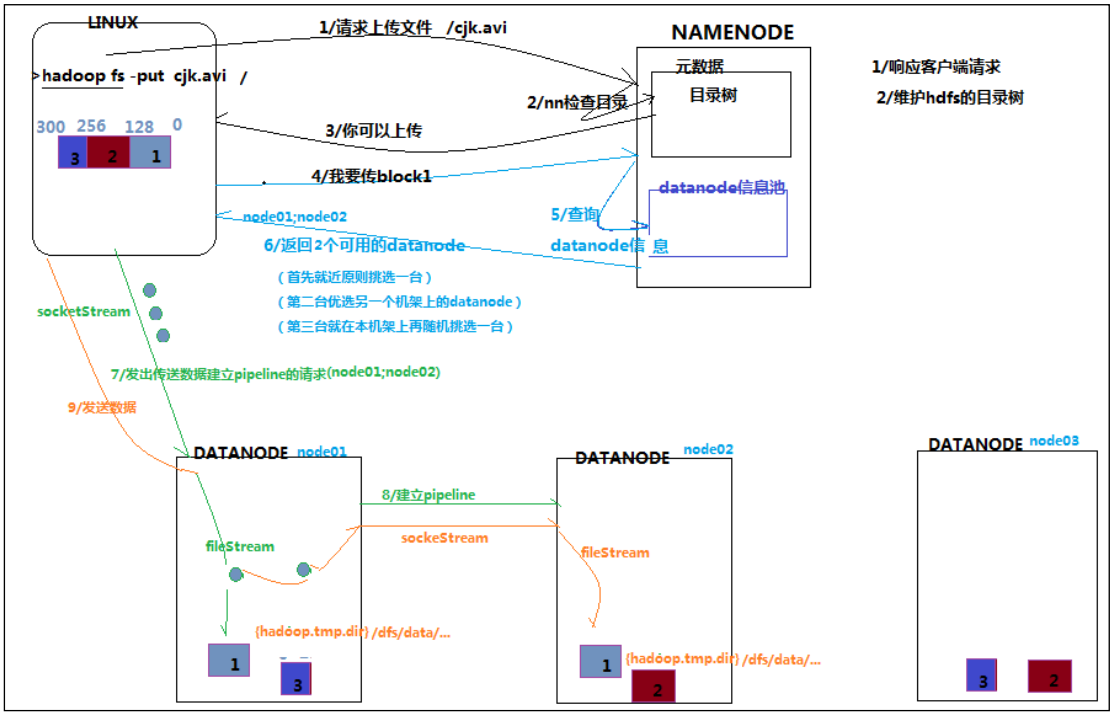
1.2、HDFS 写数据流程

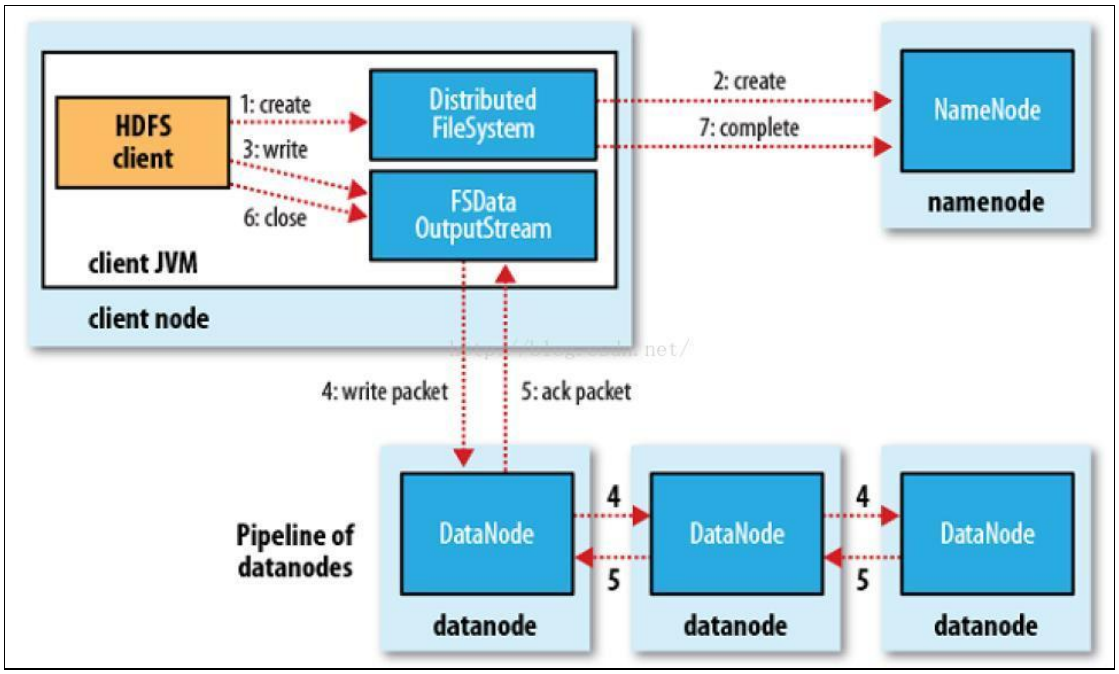
1.2.1、概述

客户端要向 HDFS 写数据，首先要跟 namenode 通信以确认可以写文件并获得接收文件 block 的 datanode，然后，客户端按顺序将文件逐个 block 传递给相应 datanode，并由接收到 block 的 datanode 负责向其他 datanode 复制 block 的副本

1、 client 发写数据请求
2、 namenode 相应请求，然后做一系列校验，如果能上传该数据，则返回该文件的所有切块应该被存在哪些 datanode 上的 datanodes 列表
blk-001:hadoop02 hadoop03
blk-002:hadoop03 hadoop04
3、 client 拿到 datanode 列表之后，开始传数据
4、 首先传第一块 blk-001，datanode 列表就是 hadoop02,hadoop03, client 就把 blk-001 传到 hadoop02 和 hadoop03 上
5、 …… 用传第一个数据块同样的方式传其他的数据块
6、 当所有的数据块都传完之后，client 会给 namenode 返回一个状态信息，表示数据已全部写入成功，或者是失败的信息
7、 namenode 接收到 client 返回的状态信息来判断当次写入数据的请求是否成功，如果成功，就需要更新元数据信息

1.2.2、详细步骤图





1.2.3、详细步骤文字说明

1、使用 HDFS 提供的客户端 Client，向远程的 namenode 发起 RPC 请求

2、namenode 会检查要创建的文件是否已经存在，创建者是否有权限进行操作，成功则会为文件创建一个记录，否则会让客户端抛出异常；

3、当客户端开始写入文件的时候，客户端会将文件切分成多个 packets，并在内部以数据队

列“data queue（数据队列）”的形式管理这些 packets，并向 namenode 申请 blocks，获取用来存储 replicas 的合适的 datanode 列表，列表的大小根据 namenode 中 replication 的设定而定；

4、开始以 pipeline（管道）的形式将 packet 写入所有的 replicas 中。客户端把 packet 以流的方式写入第一个 datanode，该 datanode 把该 packet 存储之后，再将其传递给在此 pipeline 中的下一个 datanode，直到最后一个 datanode，这种写数据的方式呈流水线的形式。

5、最后一个 datanode 成功存储之后会返回一个 ack packet（确认队列），在 pipeline 里传递至客户端，在客户端的开发库内部维护着"ack queue"，成功收到 datanode 返回的 ack packet 后会从"data queue"移除相应的 packet。

6、如果传输过程中，有某个 datanode 出现了故障，那么当前的 pipeline 会被关闭，出现故障的 datanode 会从当前的 pipeline 中移除，剩余的 block 会继续剩下的 datanode 中继续以 pipeline 的形式传输，同时 namenode 会分配一个新的 datanode，保持 replicas 设定的数量。

7、客户端完成数据的写入后，会对数据流调用 close()方法，关闭数据流；

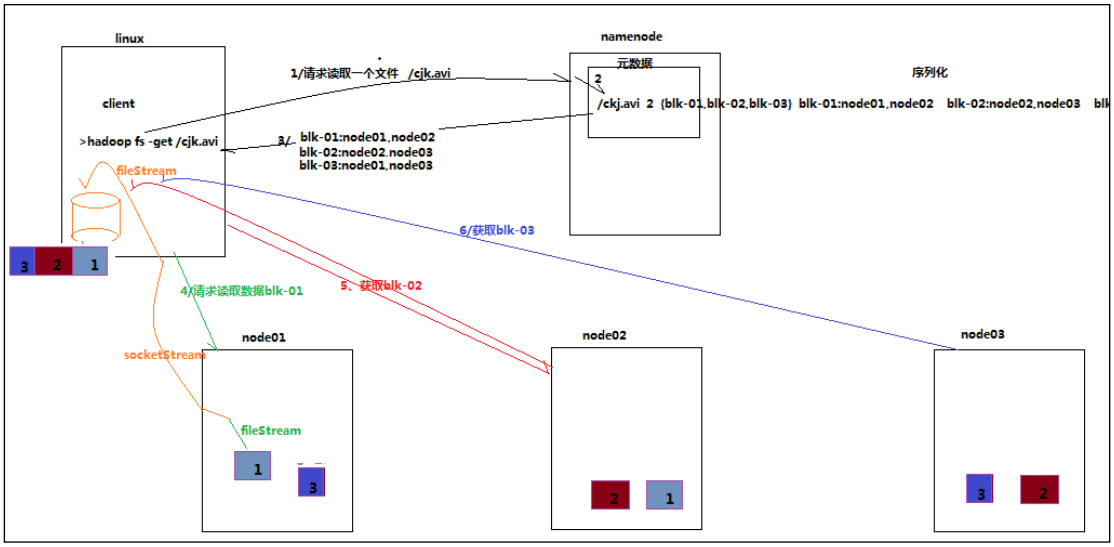
8、只要写入了 dfs.replication.min（最小写入成功的副本数）的复本数（默认为 1），写操作就会成功，并且这个块可以在集群中异步复制，直到达到其目标复本数（dfs.replication 的默认值为 3），因为 namenode 已经知道文件由哪些块组成，所以它在返回成功前只需要等待数据块进行最小量的复制。

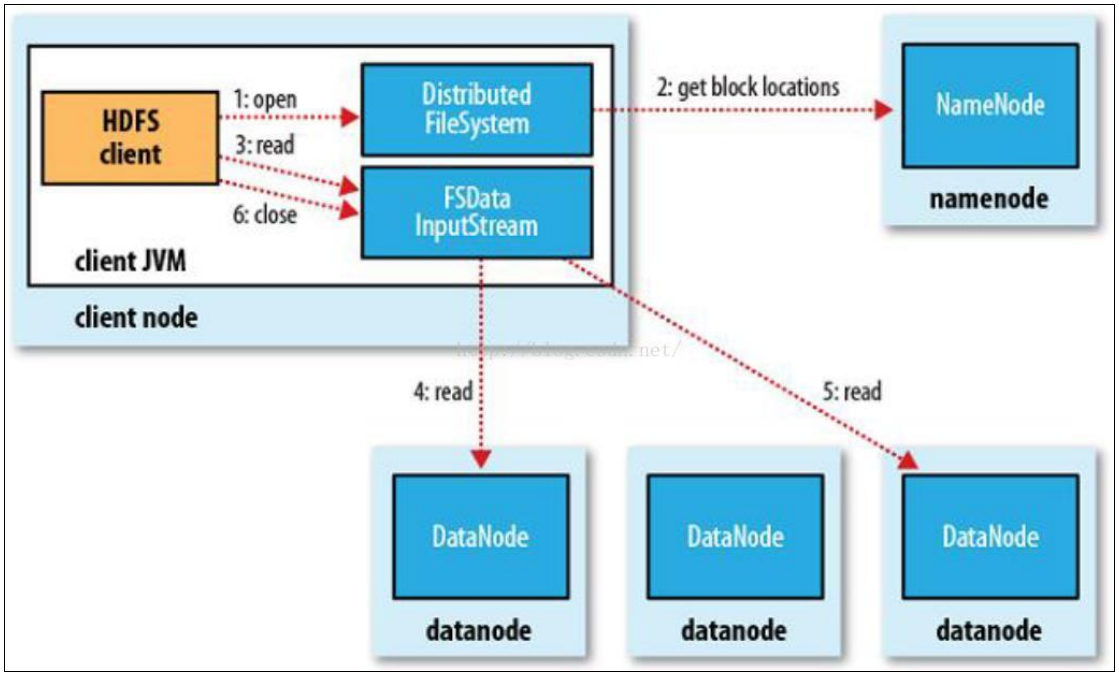
1.3、HDFS 读数据流程

1.3.1、概述

客户端将要读取的文件路径发送给 namenode，namenode 获取文件的元信息（主要是 block 的存放位置信息）返回给客户端，客户端根据返回的信息找到相应 datanode 逐个获取文件的 block 并在客户端本地进行数据追加合并从而获得整个文件

1.3.2、详细步骤图





1.3.3、详细文字说明

1、使用 HDFS 提供的客户端 Client，向远程的 namenode 发起 RPC 请求；
2、namenode 会视情况返回文件的全部 block 列表，对于每个 block，namenode 都会返回有该 block 拷贝的 datanode 地址；
3、客户端 Client 会选取离客户端最近的 datanode 来读取 block；如果客户端本身就是 datanode，那么将从本地直接获取数据；
4、读取完当前 block 的数据后，关闭当前的 datanode 链接，并为读取下一个 block 寻找最佳的 datanode；
5、当读完列表 block 后，且文件读取还没有结束，客户端会继续向 namenode 获取下一批的 block 列表；
6、读取完一个 block 都会进行 checksum 验证，如果读取 datanode 时出现错误，客户端会通知 namenode，然后再从下一个拥有该 block 拷贝的 datanode 继续读。

2、NameNode 工作机制

学习目标：理解 namenode 的工作机制尤其是元数据管理机制，以增强对 HDFS 工作原理的理解，及培养 hadoop 集群运营中“性能调优”、“namenode”故障问题的分析解决能力

问题场景：
1、Namenode 服务器的磁盘故障导致 namenode 宕机，如何挽救集群及数据？
2、Namenode 是否可以有多个？namenode 内存要配置多大？namenode 跟集群数据存储能力有关系吗？
3、文件的 blocksize 究竟调大好还是调小好？结合 mapreduce

……

诸如此类问题的回答，都需要基于对 namenode 自身的工作原理的深刻理解

一条元数据 = 150byte
1000W = 1.5G
10E = 150G

2.1、NameNode 职责

1、负责客户端请求（读写数据请求）的响应
2、维护目录树结构（元数据的管理：查询，修改）
3、配置和应用副本存放策略
4、管理集群数据块负载均衡问题

2.2、NameNode 元数据管理

WAL（Write ahead Log）：预写日志系统

在计算机科学中，预写式日志（Write-ahead logging，缩写 WAL）是关系数据库系统中用于提供原子性和持久性（ACID 属性中的两个）的一系列技术。在使用 WAL 的系统中，所有的修改在提交之前都要先写入 log 文件中。

Log 文件中通常包括 redo 和 undo 信息。这样做的目的可以通过一个例子来说明。假设一个程序在执行某些操作的过程中机器掉电了。在重新启动时，程序可能需要知道当时执行的操作是成功了还是部分成功或者是失败了。如果使用了 WAL，程序就可以检查 log 文件，并对突然掉电时计划执行的操作内容跟实际上执行的操作内容进行比较。在这个比较的基础上，程序就可以决定是撤销已做的操作还是继续完成已做的操作，或者是保持原样。

WAL 允许用 in-place 方式更新数据库。另一种用来实现原子更新的方法是 shadow paging，它并不是 in-place 方式。用 in-place 方式做更新的主要优点是减少索引和块列表的修改。ARIES 是 WAL 系列技术常用的算法。在文件系统中，WAL 通常称为 journaling。PostgreSQL 也是用 WAL 来提供 point-in-time 恢复和数据库复制特性。

NameNode 对数据的管理采用了**两种**存储形式：内存和磁盘

首先是**内存**中存储了一份完整的元数据，包括目录树结构，以及文件和数据块和副本存储地的映射关系；
1、内存元数据 metadata（全部存在内存中）

其次是在**磁盘**中也存储了一份完整的元数据。有三种格式：

2、磁盘元数据镜像文件 fsimage_0000000000000000555
fsimage_0000000000000000**555**
等价于

edits_0000000000000000**001**-0000000000000000018

……

edits_0000000000000000444-0000000000000000**555**

合并之和

3、数据历史操作日志文件 edits：edits_0000000000000000001-0000000000000000018

（可通过日志运算出元数据，全部存在磁盘中）

4、数据预写操作日志文件 edits_inprogress_0000000000000000556

（存储在磁盘中）

metadata = 最新 fsimage_0000000000000000555 + edits_inprogress_0000000000000000556

metadata = 所有的 edits 之和（edits_001_002 + …… + edits_444_555 + edits_inprogress_556）

VERSION（存放 hdfs 集群的版本信息）文件解析：

```
#Sun Jan 06 20:12:30 CST 2017    ##  集群启动时间
namespaceID=844434736    ##  文件系统唯一标识符
clusterID=CID-5b7b7321-e43f-456e-bf41-18e77c5e5a40      ##  集群唯一标识符
cTime=0      ## fsimage 创建的时间，初始为 0，随 layoutVersion 更新
storageType=NAME_NODE      ##节点类型
blockpoolID=BP-265332847-192.168.123.202-1483581570658    ##  数据块池 ID，可以有多个
layoutVersion=-60      ## hdfs 持久化数据结构的版本号
```

查看 edits 文件信息：

hdfs oev -i edits_0000000000000000482-0000000000000000483 -o edits.xml

cat edits.xml

```
[hadoop@hadoop02 current]$ hdfs oev -i edits_0000000000000000482-0000000000000000483 -o edits.xml
[hadoop@hadoop02 current]$ cat edits.xml
<?xml version="1.0" encoding="UTF-8"?>
<EDITS>
  <EDITS_VERSION>-60</EDITS_VERSION>
  <RECORD>
    <OPCODE>OP_START_LOG_SEGMENT</OPCODE>
    <DATA>
      <TXID>482</TXID>
    </DATA>
  </RECORD>
  <RECORD>
    <OPCODE>OP_END_LOG_SEGMENT</OPCODE>
    <DATA>
      <TXID>483</TXID>
    </DATA>
  </RECORD>
</EDITS>
```

查看 fsimage 镜像文件信息：

hdfs oiv -i fsimage_0000000000000000348 -p XML -o fsimage.xml

cat fsimage.xml

fsimage.xml

```xml
<?xml version="1.0"?>
<fsimage>
    <NameSection>
        <genstampV1>1000</genstampV1>
        <genstampV2>1014</genstampV2>
        <genstampV1Limit>0</genstampV1Limit>
        <lastAllocatedBlockId>1073741838</lastAllocatedBlockId>
        <txid>348</txid>
    </NameSection>
    <INodeSection>
        <lastInodeId>16481</lastInodeId>
        <inode>
            <id>16385</id>
            <type>DIRECTORY</type>
            <name></name>
            <mtime>1498287531287</mtime>
            <permission>hadoop:supergroup:rwxr-xr-x</permission>
            <nsquota>9223372036854775807</nsquota>
            <dsquota>-1</dsquota>
        </inode>
        <inode>
            <id>16386</id>
            <type>DIRECTORY</type>
            <name>wordcount</name>
            <mtime>1498288647443</mtime>
```

2.3、NameNode 元数据存储机制

A、内存中有一份完整的元数据(内存 metadata)

B、磁盘有一个“准完整”的元数据镜像（fsimage）文件(在 namenode 的工作目录中)

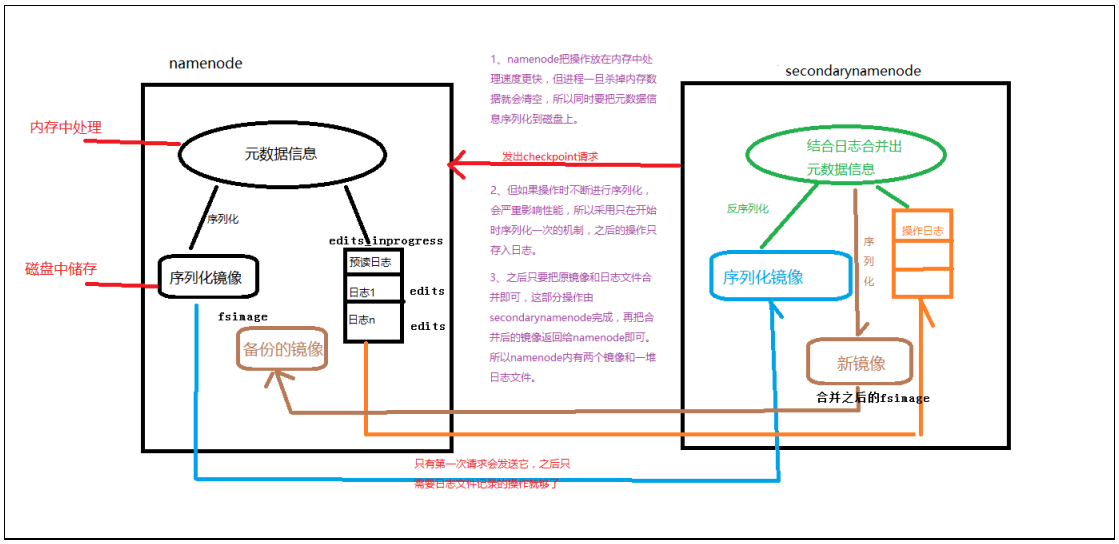
C、用于衔接内存 metadata 和持久化元数据镜像 fsimage 之间的操作日志（edits 文件）

PS：当客户端对 hdfs 中的文件进行新增或者修改操作，操作记录首先被记入 edits 日志文件中，当客户端操作成功后，相应的元数据会更新到内存 metadata 中

2.4、元数据的 CheckPoint

每隔一段时间，会由 secondary namenode 将 namenode 上积累的所有 edits 和一个最新的 fsimage 下载到本地，并加载到内存进行 merge（这个过程称为 checkpoint）

CheckPoint 详细过程图解：



2.5、CheckPoint 触发配置

```
dfs.namenode.checkpoint.check.period=60   ##检查触发条件是否满足的频率，60 秒
dfs.namenode.checkpoint.dir=file://${hadoop.tmp.dir}/dfs/namesecondary
                ##以上两个参数做 checkpoint 操作时，secondary namenode 的本地工作目录
dfs.namenode.checkpoint.edits.dir=${dfs.namenode.checkpoint.dir}

dfs.namenode.checkpoint.max-retries=3   ##最大重试次数
dfs.namenode.checkpoint.period=3600     ##两次 checkpoint 之间的时间间隔 3600 秒
dfs.namenode.checkpoint.txns=1000000    ##两次 checkpoint 之间最大的操作记录
```

2.6、CheckPoint 附带作用

Namenode 和 SecondaryNamenode 的工作目录存储结构完全相同，所以，当 Namenode 故障退出需要重新恢复时，可以从 SecondaryNamenode 的工作目录中将 fsimage 拷贝到 Namenode 的工作目录，以恢复 namenode 的元数据

3、DataNode 工作机制

问题场景：
1、集群容量不够，怎么扩容？
2、如果有一些 datanode 宕机，该怎么办？
3、datanode 明明已启动，但是集群中的可用 datanode 列表中就是没有，怎么办？

以上这类问题的解答，有赖于对 datanode 工作机制的深刻理解

3.1、概述

1、Datanode 工作职责：

存储管理用户的文件块数据
定期向 namenode 汇报自身所持有的 block 信息（通过心跳信息上报）
（PS：这点很重要，因为，当集群中发生某些 block 副本失效时，集群如何恢复 block 初始副本数量的问题）

```
<property>
    <!—HDFS 集群数据冗余块的自动删除时长，单位 ms，默认一个小时 -->
    <name>dfs.blockreport.intervalMsec</name>
    <value>3600000</value>
    <description>Determines block reporting interval in milliseconds.</description>
</property>
```

2、Datanode 掉线判断时限参数

datanode 进程死亡或者网络故障造成 datanode 无法与 namenode 通信，namenode 不会立即把该节点判定为死亡，要经过一段时间，这段时间暂称作超时时长。HDFS 默认的超时时长为 10 分钟+30 秒。如果定义超时时间为 timeout，则超时时长的计算公式为：
timeout = 2 * heartbeat.recheck.interval + 10 * dfs.heartbeat.interval
而默认的 heartbeat.recheck.interval 大小为 5 分钟，dfs.heartbeat.interval 默认为 3 秒。
需要注意的是 hdfs-site.xml 配置文件中的 heartbeat.recheck.interval 的单位为毫秒，dfs.heartbeat.interval 的单位为秒。
所以，举个例子，如果 heartbeat.recheck.interval 设置为 5000（毫秒），dfs.heartbeat.interval 设置为 3（秒，默认），则总的超时时间为 40 秒。

```
<property>
        <name>heartbeat.recheck.interval</name>
        <value>5000</value>
</property>
<property>
        <name>dfs.heartbeat.interval</name>
        <value>3</value>
</property>
```

3.2、观察验证 DATANODE 功能

上传一个文件，观察文件的 block 具体的物理存放情况：
在每一台 datanode 机器上的这个目录中能找到文件的切块：
/home/hadoop/hadoopdata/data/current/BP-771296455-192.168.123.106-1504830258603/current/finalized/subdir0/subdir0

```
[hadoop@hadoop07 subdir0]$ ll
total 404516
-rw-rw-r--. 1 hadoop hadoop 134217728 Sep  8 08:25 blk_1073741825
-rw-rw-r--. 1 hadoop hadoop   1048583 Sep  8 08:25 blk_1073741825_1001.meta
-rw-rw-r--. 1 hadoop hadoop  64484319 Sep  8 08:25 blk_1073741826
-rw-rw-r--. 1 hadoop hadoop    503791 Sep  8 08:25 blk_1073741826_1002.meta
-rw-rw-r--. 1 hadoop hadoop        66 Sep 13 07:57 blk_1073741827
-rw-rw-r--. 1 hadoop hadoop        11 Sep 13 07:57 blk_1073741827_1003.meta
-rw-rw-r--  1 hadoop hadoop    171432 Sep 15 08:36 blk_1073741828
-rw-rw-r--  1 hadoop hadoop      1347 Sep 15 08:36 blk_1073741828_1004.meta
-rw-rw-r--  1 hadoop hadoop    134368 Sep 22 08:35 blk_1073741829
-rw-rw-r--  1 hadoop hadoop      1059 Sep 22 08:35 blk_1073741829_1005.meta
-rw-rw-r--  1 hadoop hadoop  24594131 Sep 15 08:37 blk_1073741830
-rw-rw-r--  1 hadoop hadoop    192151 Sep 15 08:37 blk_1073741830_1006.meta
```

4、SecondaryNamenode 工作机制

SecondaryNamenode 的作用就是分担 namenode 的合并元数据的压力。所以在配置 SecondaryNamenode 的工作节点时，一定切记，不要和 namenode 处于同一节点。但事实上，只有在普通的伪分布式集群和分布式集群中才有会 SecondaryNamenode 这个角色，在 HA 或者联邦集群中都不再出现该角色。在 HA 和联邦集群中，都是有 standby namenode 承担

就是 CheckPoint 的工作机制
请看元数据的 CheckPoint