尚硅谷大数据技术之 Hadoop（HDFS）

## （作者：尚硅谷大数据研发部）

### 版本：V3.3

第 **1** 章 **HDFS** 概 述

* 1. **HDFS** 产出背景及定义

1. **HDFS** 产生背景

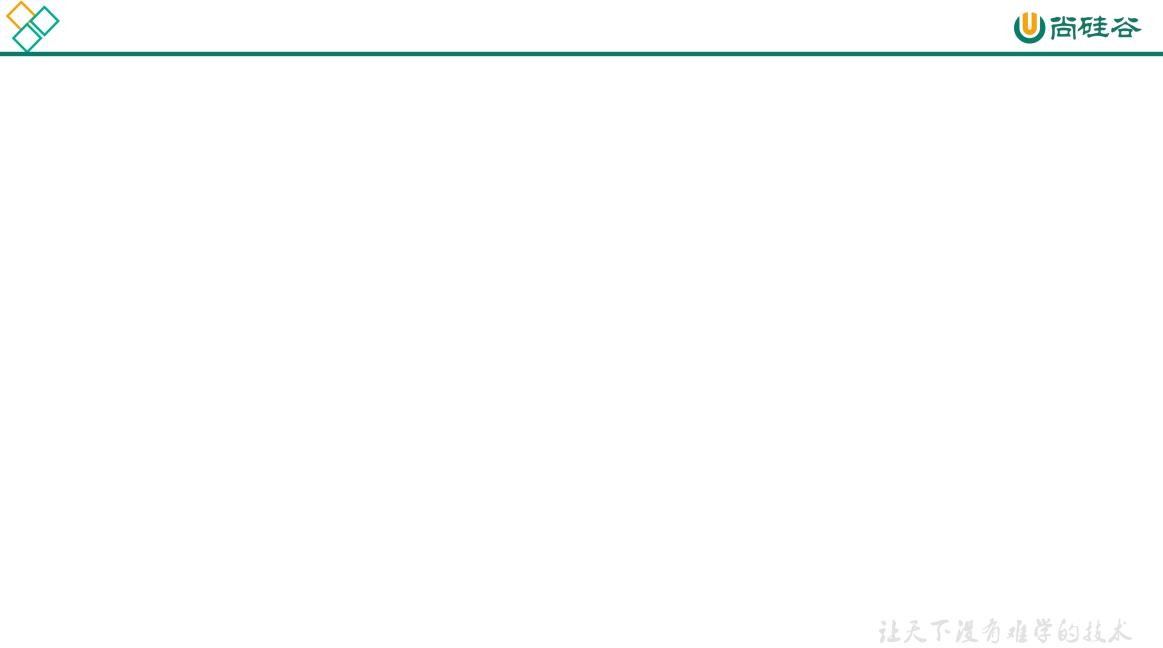
### 随着数据量越来越大，在一个操作系统存不下所有的数据，那么就分配到更多的操作系统管理的磁盘中，但是不方便管理和维护，迫切需要一种系统来管理多台机器上的文件，这就是分布式文件管理系统。HDFS 只是分布式文件管理系统中的一种。

1. **HDFS** 定义

### HDFS（Hadoop Distributed File System），它是一个文件系统，用于存储文件，通过目录树来定位文件；其次，它是分布式的，由很多服务器联合起来实现其功能，集群中的服务器有各自的角色。

HDFS 的使用场景：适合一次写入，多次读出的场景。一个文件经过创建、写入和关闭之后就不需要改变。

* 1. **HDFS** 优缺点



HDFS优点

1. 高容错性
   * 数据自动保存多个副本。它通过增加副本的形式，提高容错性。

副本

* + 某一个副本丢失以后，它可以自动恢复。

副本

1. 适合处理大数据
   * 数据规模：能够处理数据规模达到GB、TB、甚至PB级别的数据；
   * 文件规模：能够处理百万规模以上的文件数量，数量相当之大。
2. 可构建在廉价机器上，通过多副本机制，提高可靠性。

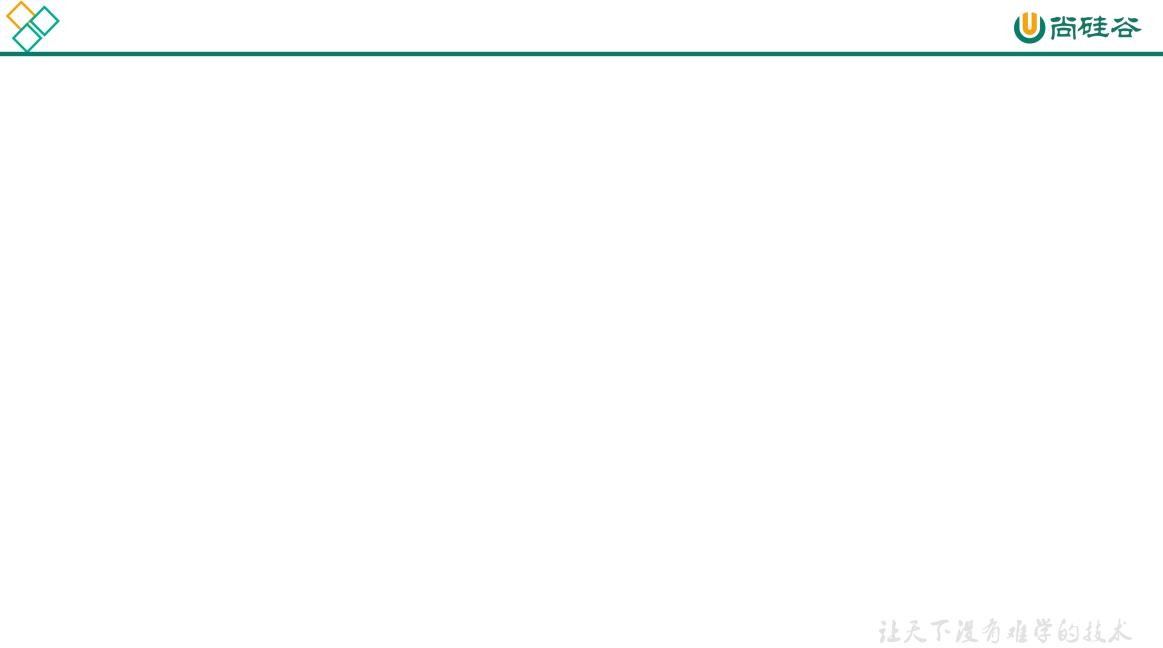
副本

副本

副本

副本

副本



HDFS缺点

1. 不适合低延时数据访问，比如毫秒级的存储数据，是做不到的。
2. 无法高效的对大量小文件进行存储。
   * 存储大量小文件的话，它会占用NameNode大量的内存来存储文件目录和

块信息。这样是不可取的，因为NameNode的内存总是有限的；

* + 小文件存储的寻址时间会超过读取时间，它违反了HDFS的设计目标。

1. 不支持并发写入、文件随机修改。
   * 一个文件只能有一个写，不允许多个线程同时写；

* 仅支持数据append（追加），不支持文件的随机修改。

上传文件

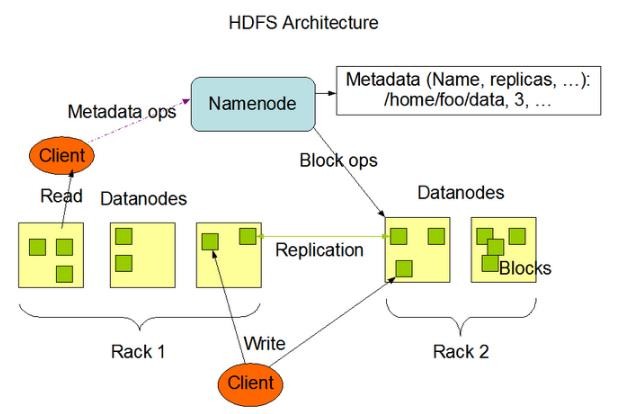
ss.txt

上传文件

ss.txt

HDFS

* 1. **HDFS** 组成架构



HDFS组成架构

1. NameNode（nn）：就是Master，它

是一个主管、管理者。

* 1. 管理HDFS的名称空间；
  2. 配置副本策略；
  3. 管理数据块（Block）映射信息；
  4. 处理客户端读写请求。

1. DataNode：就是Slave。NameNode

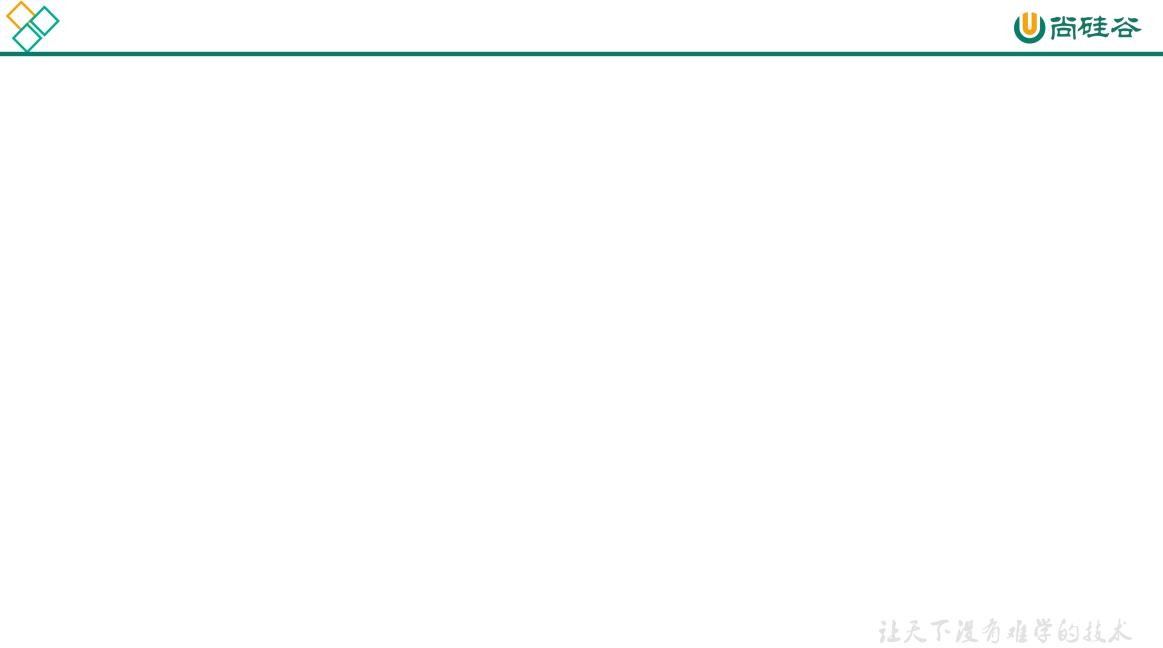
下达命令，DataNode执行实际的操作。

1. 存储实际的数据块；
2. 执行数据块的读/写操作。



HDFS组成架构

1. Client：就是客户端。
   1. 文件切分。文件上传HDFS的时候，Client将文件切分成一个一个的Block，然后进行上传；
   2. 与NameNode交互，获取文件的位置信息；
   3. 与DataNode交互，读取或者写入数据；
   4. Client提供一些命令来管理HDFS，比如NameNode格式化；
   5. Client可以通过一些命令来访问HDFS，比如对HDFS增删查改操作；
2. Secondary NameNode：并非NameNode的热备。当NameNode挂掉的时候，它并不能马上替换NameNode并提供服务。
3. 辅助NameNode，分担其工作量，比如定期合并Fsimage和Edits，并推送给NameNode ；
4. 在紧急情况下，可辅助恢复NameNode。
   1. **HDFS** 文件块大小（面试重点）



HDFS 文件块大小

HDFS 中的文件在物理上是分块存储（ Block ） ， 块的大小可以通过配置参数

( dfs.blocksize）来规定，默认大小在Hadoop2.x/3.x版本中是128M，1.x版本中是64M。

1）集群中的block

2）如果寻址时间约为10ms，

即查找到目标block的时间为

10ms。

3）寻址时间为传输时间的1% 时，则为最佳状态。（专家） 因此，传输时间

=10ms/0.01=1000ms=1s

block1

block2

5 block大小

=1s\*100MB/s=100MB

… …

4）而目前磁盘的传输速率普

遍为100MB/s。

blockn



HDFS文件块大小

思考：为什么块的大小不能设置太小，也不能设置太大？

（1）HDFS的块设置太小，会增加寻址时间，程序一直在找块的开始位置；

（2）如果块设置的太大，从磁盘传输数据的时间会明显大于定位这个块开

始位置所需的时间。导致程序在处理这块数据时，会非常慢。

总结：**HDFS**块的大小设置主要取决于磁盘传输速率。

第 **2** 章 **HDFS** 的 **Shell** 操作（开发重点）

# 基本语法

### hadoop fs 具体命令 OR hdfs dfs 具体命令两个是完全相同的。

* 1. 命令大全

[atguigu@hadoop102 hadoop-3.1.3]$ bin/hadoop fs

[-appendToFile <localsrc> ... <dst>] [-cat [-ignoreCrc] <src> ...]

[-chgrp [-R] GROUP PATH...]

[-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]

[-chown [-R] [OWNER][:[GROUP]] PATH...]

[-copyFromLocal [-f] [-p] <localsrc> ... <dst>]

[-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>] [-count [-q] <path> ...]

[-cp [-f] [-p] <src> ... <dst>]

[-df [-h] [<path> ...]]

[-du [-s] [-h] <path> ...]

[-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>] [-getmerge [-nl] <src> <localdst>]

[-help [cmd ...]]

[-ls [-d] [-h] [-R] [<path> ...]]

[-mkdir [-p] <path> ...]

[-moveFromLocal <localsrc> ... <dst>] [-moveToLocal <src> <localdst>]

[-mv <src> ... <dst>]

[-put [-f] [-p] <localsrc> ... <dst>]

[-rm [-f] [-r|-R] [-skipTrash] <src> ...]

[-rmdir [--ignore-fail-on-non-empty] <dir> ...]

<acl\_spec> <path>]]

[-setrep [-R] [-w] <rep> <path> ...]

[-stat [format] <path> ...]

[-tail [-f] <file>]

[-test -[defsz] <path>]

[-text [-ignoreCrc] <src> ...]

* 1. 常用命令实操
     1. 准备工作

1）启动 Hadoop 集群（方便后续的测试）

[atguigu@hadoop102 hadoop-3.1.3]$ sbin/start-dfs.sh [atguigu@hadoop103 hadoop-3.1.3]$ sbin/start-yarn.sh

2）-help：输出这个命令参数

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -help rm

1. 创建/sanguo 文件夹

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -mkdir /sanguo

* + 1. 上传

1）-moveFromLocal：从本地剪切粘贴到 HDFS

[atguigu@hadoop102 hadoop-3.1.3]$ vim shuguo.txt

输入：

shuguo

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -moveFromLocal ./shuguo.txt

/sanguo

2）-copyFromLocal：从本地文件系统中拷贝文件到 HDFS 路径去

[atguigu@hadoop102 hadoop-3.1.3]$ vim weiguo.txt

输入：

weiguo

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -copyFromLocal weiguo.txt

/sanguo

3）-put：等同于 copyFromLocal，生产环境更习惯用put

[atguigu@hadoop102 hadoop-3.1.3]$ vim wuguo.txt

输入：

wuguo

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -put ./wuguo.txt /sanguo

4）-appendToFile：追加一个文件到已经存在的文件末尾

[atguigu@hadoop102 hadoop-3.1.3]$ vim liubei.txt

输入：

liubei

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -appendToFile liubei.txt

/sanguo/shuguo.txt

* + 1. 下载

1）-copyToLocal：从HDFS 拷贝到本地

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -copyToLocal

/sanguo/shuguo.txt ./

2）-get：等同于 copyToLocal，生产环境更习惯用 get

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -get

/sanguo/shuguo.txt ./shuguo2.txt

* + 1. **HDFS** 直接操作

### 1）-ls: 显示目录信息

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -ls /sanguo

2）-cat：显示文件内容

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -cat /sanguo/shuguo.txt

3）-chgrp、-chmod、-chown：Linux 文件系统中的用法一样，修改文件所属权限

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -chmod 666

/sanguo/shuguo.txt

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -chown atguigu:atguigu

/sanguo/shuguo.txt

4）-mkdir：创建路径

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -mkdir /jinguo

5）-cp：从 HDFS 的一个路径拷贝到HDFS 的另一个路径

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -cp /sanguo/shuguo.txt

/jinguo

6）-mv：在 HDFS 目录中移动文件

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -mv /sanguo/wuguo.txt /jinguo [atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -mv /sanguo/weiguo.txt

/jinguo

7）-tail：显示一个文件的末尾 1kb 的数据

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -tail /jinguo/shuguo.txt

8）-rm：删除文件或文件夹

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -rm /sanguo/shuguo.txt

9）-rm -r：递归删除目录及目录里面内容

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -rm -r /sanguo

10）-du 统计文件夹的大小信息

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -du -s -h /jinguo

27 81 /jinguo

[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -du -h /jinguo

14 42 /jinguo/shuguo.txt

7 21 /jinguo/weiguo.txt

6 18 /jinguo/wuguo.tx

说明：27 表示文件大小；81 表示 27\*3 个副本；/jinguo 表示查看的目录

11）-setrep：设置 HDFS 中文件的副本数量



[atguigu@hadoop102 hadoop-3.1.3]$ hadoop fs -setrep 10 /jinguo/shuguo.txt

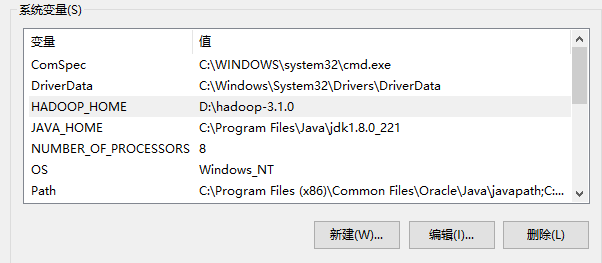
这里设置的副本数只是记录在NameNode 的元数据中，是否真的会有这么多副本，还得看 DataNode 的数量。因为目前只有 3 台设备，最多也就 3 个副本，只有节点数的增加到 10 台时，副本数才能达到 10。

第 **3** 章 **HDFS** 的 **API** 操 作

# 客户端环境准备

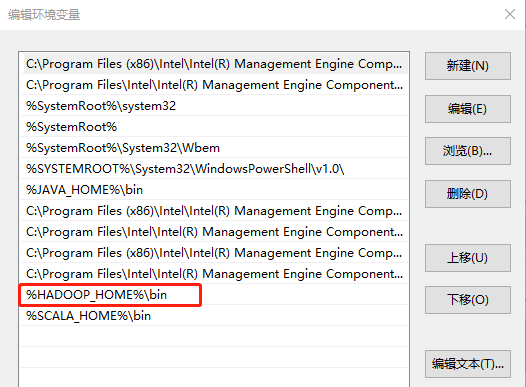
### 找到资料包路径下的 Windows 依赖文件夹，拷贝 hadoop-3.1.0 到非中文路径（比如 d:\）。

1. 配置 **HADOOP\_HOME** 环境变量



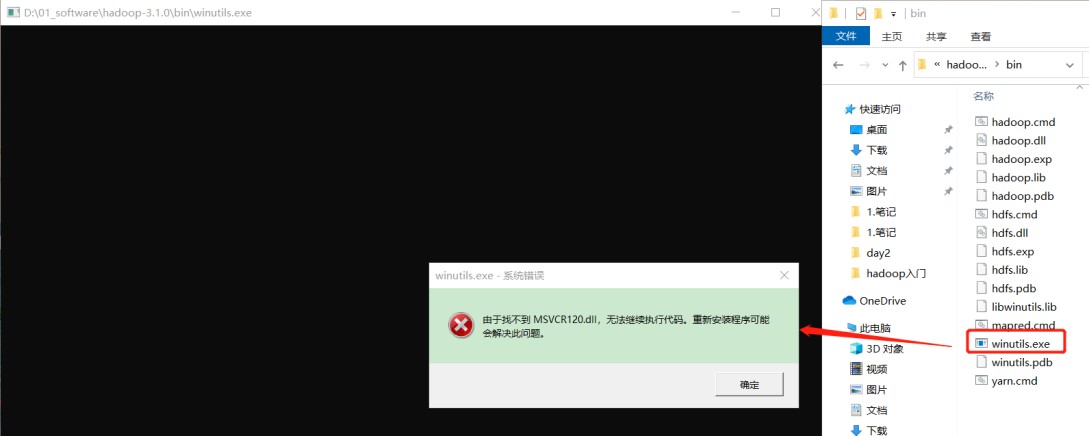
1. 配置 **Path** 环境变量。

### 注意：如果环境变量不起作用，可以重启电脑试试。



验证 Hadoop 环境变量是否正常。双击 winutils.exe，如果报如下错误。说明缺少微软运

行库（正版系统往往有这个问题）。再资料包里面有对应的微软运行库安装包双击安装即可。



1. 在 **IDEA** 中创建一个 **Maven** 工程 **HdfsClientDemo**，并导入相应的依赖坐标**+**日志添加

<dependencies>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-client</artifactId>

<version>3.1.3</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.30</version>

</dependency>

</dependencies>

### 在项目的 src/main/resources 目录下，新建一个文件，命名为“log4j.properties”，在文件

中填入

log4j.rootLogger=INFO, stdout log4j.appender.stdout=org.apache.log4j.ConsoleAppender log4j.appender.stdout.layout=org.apache.log4j.PatternLayout log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n log4j.appender.logfile=org.apache.log4j.FileAppender log4j.appender.logfile.File=target/spring.log log4j.appender.logfile.layout=org.apache.log4j.PatternLayout log4j.appender.logfile.layout.ConversionPattern=%d %p [%c] - %m%n

1. 创建包名：**com.atguigu.hdfs**
2. 创建 **HdfsClient** 类

public class HdfsClient {

@Test

public void testMkdirs() throws IOException, URISyntaxException, InterruptedException {

// 1 获取文件系统

Configuration configuration = new Configuration();

// FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:8020"), configuration);

FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:8020"), configuration,"atguigu");

// 2 创建目录

fs.mkdirs(new Path("/xiyou/huaguoshan/"));

// 3 关闭资源

fs.close();

}

}

### 执行程序

客户端去操作HDFS 时，是有一个用户身份的。默认情况下，HDFS 客户端API 会从采用 Windows 默认用户访问 HDFS，会报权限异常错误。所以在访问 HDFS 时，一定要配置用户。

org.apache.hadoop.security.AccessControlException: Permission denied: user=56576, access=WRITE, inode="/xiyou/huaguoshan":atguigu:supergroup:drwxr-xr-x

* 1. **HDFS** 的 **API** 案例实操
     1. **HDFS** 文件上传（测试参数优先级）

### 编写源代码

@Test

public void testCopyFromLocalFile() throws IOException, InterruptedException, URISyntaxException {

// 1 获取文件系统

Configuration configuration = new Configuration();

configuration.set("dfs.replication", "2");

FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:8020"), configuration, "atguigu");

// 2 上传文件

fs.copyFromLocalFile(new Path("d:/sunwukong.txt"), new

Path("/xiyou/huaguoshan"));

// 3 关闭资源

fs.close();

｝

1. 将 **hdfs-site.xml** 拷贝到项目的 **resources** 资源目录下

<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

</configuration>

### 参数优先级

参数优先级排序：（1）客户端代码中设置的值 >（2）ClassPath 下的用户自定义配置文

件 >（3）然后是服务器的自定义配置（xxx-site.xml）>（4）服务器的默认配置（xxx-default.xml）

* + 1. **HDFS** 文件下载

@Test

public void testCopyToLocalFile() throws IOException, InterruptedException, URISyntaxException{

// 1 获取文件系统

Configuration configuration = new Configuration();

FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:8020"), configuration, "atguigu");

// 2 执行下载操作

// boolean delSrc 指是否将原文件删除

// Path src 指要下载的文件路径

// Path dst 指将文件下载到的路径

// boolean useRawLocalFileSystem 是否开启文件校验

fs.copyToLocalFile(false, new

Path("/xiyou/huaguoshan/sunwukong.txt"), new Path("d:/sunwukong2.txt"), true);

// 3 关闭资源

fs.close();

}

### 注意：如果执行上面代码，下载不了文件，有可能是你电脑的微软支持的运行库少，需

要安装一下微软运行库。

* + 1. **HDFS** 文件更名和移动

@Test

public void testRename() throws IOException, InterruptedException, URISyntaxException{

// 1 获取文件系统

Configuration configuration = new Configuration();

FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:8020"), configuration, "atguigu");

// 2 修改文件名称

fs.rename(new Path("/xiyou/huaguoshan/sunwukong.txt"), new

Path("/xiyou/huaguoshan/meihouwang.txt"));

// 3 关闭资源

fs.close();

}

* + 1. **HDFS** 删除文件和目录

@Test

public void testDelete() throws IOException, InterruptedException, URISyntaxException{

// 1 获取文件系统

Configuration configuration = new Configuration();

FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:8020"), configuration, "atguigu");

// 2 执行删除

fs.delete(new Path("/xiyou"), true);

// 3 关闭资源

fs.close();

}

* + 1. **HDFS** 文件详情查看

### 查看文件名称、权限、长度、块信息

@Test

public void testListFiles() throws IOException, InterruptedException, URISyntaxException {

// 1 获取文件系统

Configuration configuration = new Configuration();

FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:8020"), configuration, "atguigu");

// 2 获取文件详情

RemoteIterator<LocatedFileStatus> listFiles = fs.listFiles(new Path("/"),

true);

while (listFiles.hasNext()) {

LocatedFileStatus fileStatus = listFiles.next();

System.out.println("========" + fileStatus.getPath() + "========="); System.out.println(fileStatus.getPermission()); System.out.println(fileStatus.getOwner()); System.out.println(fileStatus.getGroup()); System.out.println(fileStatus.getLen()); System.out.println(fileStatus.getModificationTime()); System.out.println(fileStatus.getReplication()); System.out.println(fileStatus.getBlockSize()); System.out.println(fileStatus.getPath().getName());

// 获取块信息

BlockLocation[] blockLocations = fileStatus.getBlockLocations();

System.out.println(Arrays.toString(blockLocations));

}

// 3 关闭资源

fs.close();

}

* + 1. **HDFS** 文件和文件夹判断

@Test

public void testListStatus() throws IOException, InterruptedException, URISyntaxException{

// 1 获取文件配置信息

Configuration configuration = new Configuration();

FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:8020"), configuration, "atguigu");

// 2 判断是文件还是文件夹

FileStatus[] listStatus = fs.listStatus(new Path("/"));

for (FileStatus fileStatus : listStatus) {

// 如果是文件

if (fileStatus.isFile()) {

System.out.println("f:"+fileStatus.getPath().getName());

}else {

System.out.println("d:"+fileStatus.getPath().getName());

}

}

// 3 关闭资源

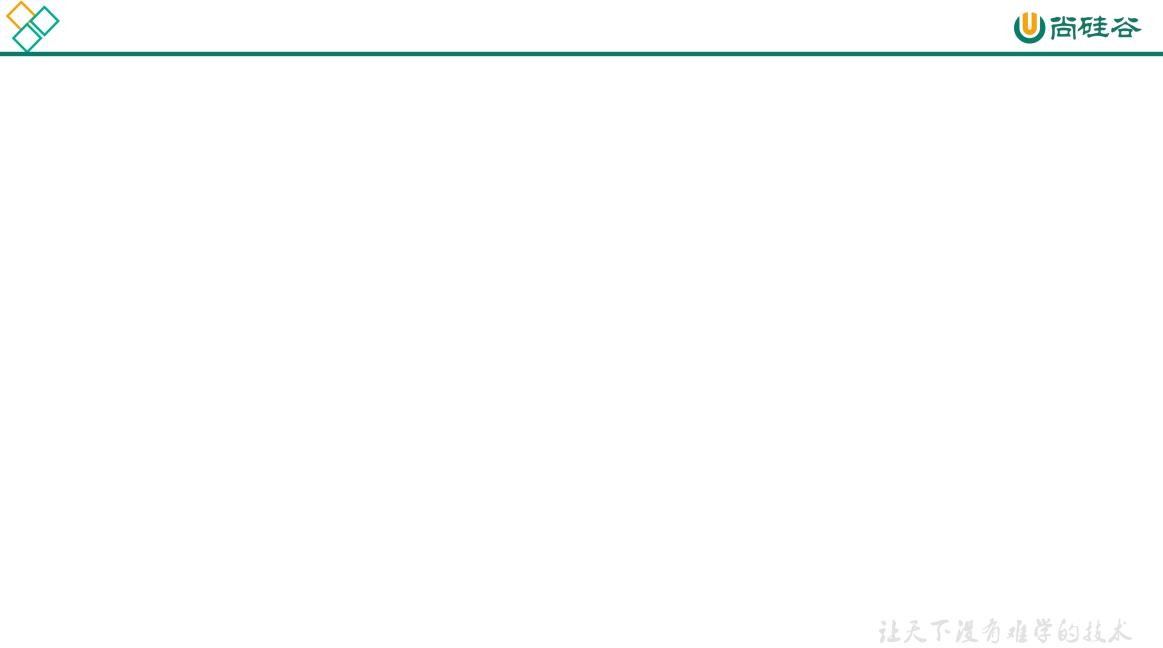
fs.close();

}

第 **4** 章 **HDFS** 的读写流程（面试重点）

* 1. **HDFS** 写数据流程

# 剖析文件写入



**HDFS**的写数据流程

客户端

create

1 向NameNode请求上传文件/user/atguigu/ss.avi

NameNode

检查目录树是否可以创建文件

* 1. 检查权限；
  2. 检查目录结构（目录是否存在）

Distributed

FileSystem

write

close

2 响应可以上传文件

3 请求上传第一个Block（0-128M），请返回DataNode

4返回dn1，dn2，dn3节点，表示采用这三个节点存储数据

200m

副本存储节点选择

* 1. 本地节点
  2. 其他机架一个节点
  3. 其他机架另一个节点

8 传输数据完成

0-128m

6 dn1应答成功

DataNode1

DataNode2

DataNode3

ss.avi

6 dn2应答成功

6 dn3应答成功

5 请求建立Block传输通道

Bytebuffer

5 请求建立通道

Bytebuffer

5 请求建立通道

Bytebuffer

7 传输数据 Packet（64k）

packet（chunk512byte+chunksum4byte）

7 blk\_1

7 blk\_1

7 blk\_1

FSDataOu

tputStream

元数据

HDFS

client

### 客户端通过Distributed FileSystem 模块向NameNode 请求上传文件，NameNode 检

查目标文件是否已存在，父目录是否存在。

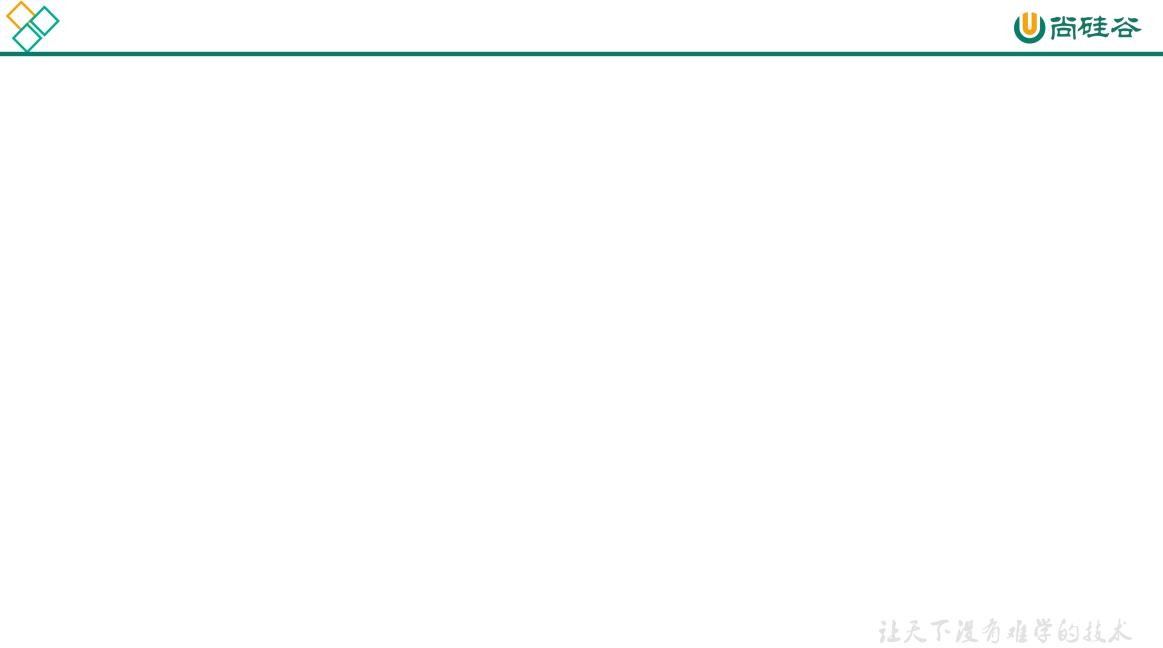
* + - 1. NameNode 返回是否可以上传。
      2. 客户端请求第一个 Block 上传到哪几个 DataNode 服务器上。
      3. NameNode 返回 3 个DataNode 节点，分别为dn1、dn2、dn3。
      4. 客户端通过 FSDataOutputStream 模块请求dn1 上传数据，dn1 收到请求会继续调用

dn2，然后dn2 调用dn3，将这个通信管道建立完成。

* + - 1. dn1、dn2、dn3 逐级应答客户端。
      2. 客户端开始往 dn1 上传第一个Block（先从磁盘读取数据放到一个本地内存缓存），以 Packet 为单位，dn1 收到一个 Packet 就会传给 dn2，dn2 传给 dn3；dn1 每传一个 packet会放入一个应答队列等待应答。
      3. 当一个 Block 传输完成之后，客户端再次请求 NameNode 上传第二个Block 的服务器。（重复执行 3-7 步）。
    1. 网络拓扑**-**节点距离计算

在 HDFS 写数据的过程中，NameNode 会选择距离待上传数据最近距离的 DataNode 接收数据。那么这个最近距离怎么计算呢？

节点距离：两个节点到达最近的共同祖先的距离总和。



网络拓扑-节点距离计算

Distance(/d1/r1/n0, /d1/r1/n0)=0（同一节点上的进程） Distance(/d1/r1/n1, /d1/r1/n2)=2（同一机架上的不同节点）

Distance(/d1/r2/n0, /d1/r3/n2)=4（同一数据中心不同机架上的节点）

Distance(/d1/r2/n1, /d2/r4/n1)=6（不同数据中心的节点）

集群d1

集群d2

机架r1

机架r2

机架r3

机架r4

机架r5

机架r6

n-0

n-0

n-0

n-0

n-0

n-0

n-1

n-1

n-1

n-1

n-1

n-1

n-2

n-2

n-2

n-2

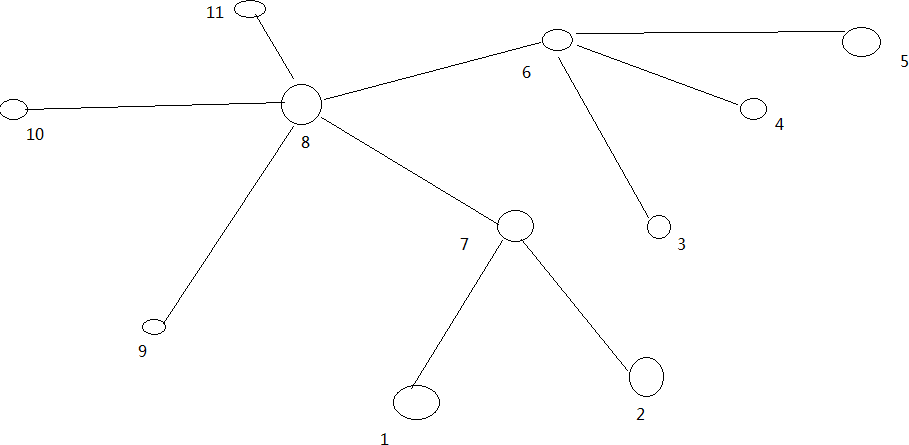
n-2

n-2

例如，假设有数据中心 d1 机架 r1 中的节点 n1。该节点可以表示为/d1/r1/n1。利用这种

标记，这里给出四种距离描述。

大家算一算每两个节点之间的距离。



* + 1. 机架感知（副本存储节点选择）

1. 机架感知说明
   1. 官方说明

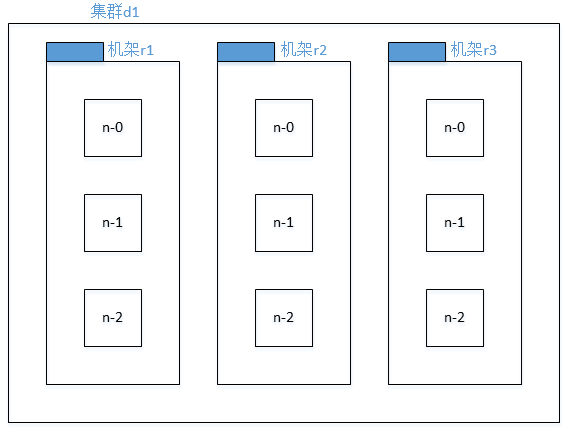
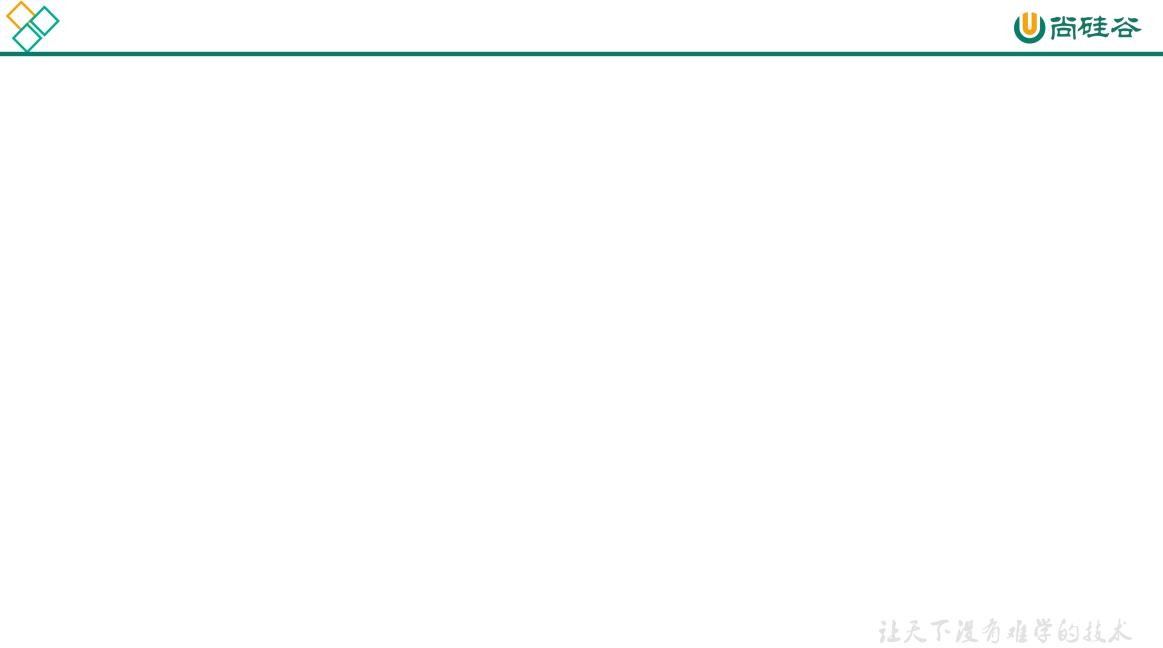
<http://hadoop.apache.org/docs/r3.1.3/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html#Data_Replication>

For the common case, when the replication factor is three, HDFS’s placement policy is to put one replica on the local machine if the writer is on a datanode, otherwise on a random datanode, another replica on a node in a different (remote) rack, and the last on a different node in the same remote rack. This policy cuts the inter-rack write traffic which generally improves write performance. The chance of rack failure is far less than that of node failure; this policy does not impact data reliability and availability guarantees. However, it does reduce the aggregate network bandwidth used when reading data since a block is placed in only two unique racks rather than three. With this policy, the replicas of a file do not evenly distribute across the racks. One third of replicas are on one node, two thirds of replicas are on one rack, and the other third are evenly distributed across the remaining racks. This policy improves write performance without compromising data reliability or read performance.

### 源码说明

Crtl + n 查找BlockPlacementPolicyDefault，在该类中查找 chooseTargetInOrder 方法。

1. **Hadoop3.1.3** 副本节点选择



Hadoop3.1.3副本节点选择

第一个副本在Client所处的节点上。

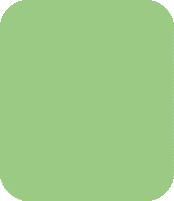
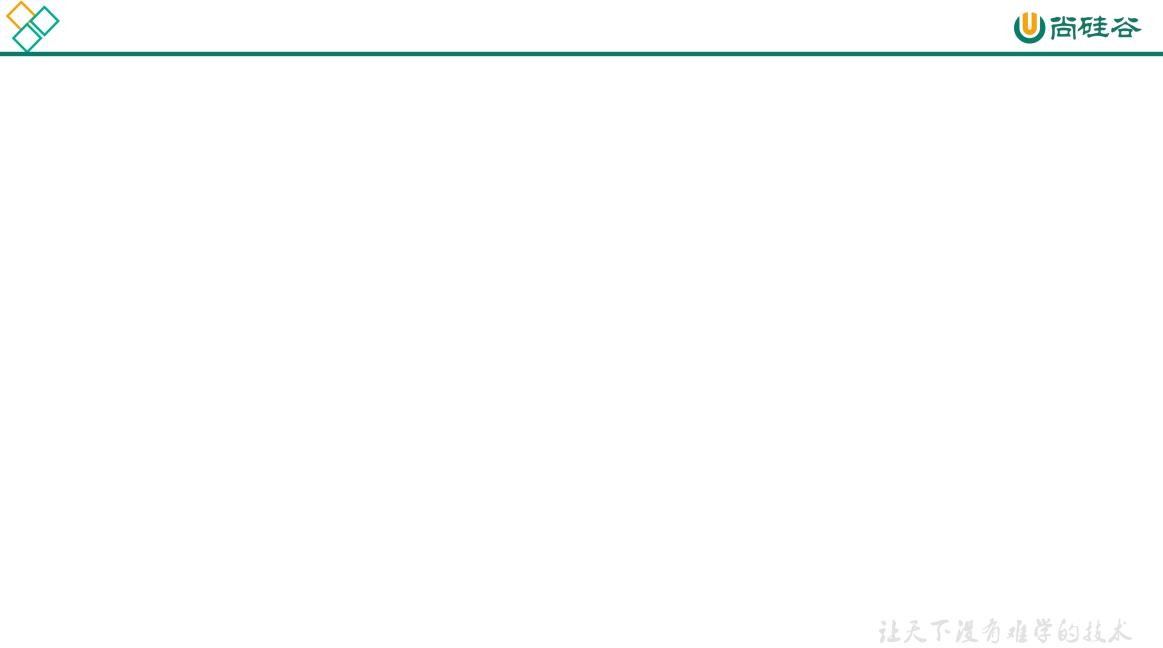
如果客户端在集群外，随机选一个。

第二个副本在另一个机架的随机

一个节点

第三个副本在第二个副本所在机架的随机节点

* 1. **HDFS** 读数据流程



HDFS的读数据流程

客户端

create

NameNode

Distributed

read FileSystem

1 请求下载文件/user/atguigu/ss.avi

2 返回目标文件的元数据

close

200m

5 请求读数据blk\_2

0-128m

ss.avi

DataNode1

DataNode2

DataNode3

1. 请求读数据blk\_1
2. 传输数据

6 传输数据

7 blk\_1

7 blk\_2

7 blk\_1

7 blk\_2

7 blk\_1

7 blk\_2

FSDataInpu tStream

元数据

/user/atguigu/ss.avi

{[blk\_1,blk\_2],[blk\_1,blk\_2],[blk\_1,blk\_2]}

HDFS

client

### 客户端通过 DistributedFileSystem 向 NameNode 请求下载文件，NameNode 通过查

询元数据，找到文件块所在的DataNode 地址。

1. 挑选一台DataNode（就近原则，然后随机）服务器，请求读取数据。
2. DataNode 开始传输数据给客户端（从磁盘里面读取数据输入流，以 Packet 为单位来做校验）。
3. 客户端以 Packet 为单位接收，先在本地缓存，然后写入目标文件。

第 **5** 章 **NameNode** 和 **SecondaryNameNode**

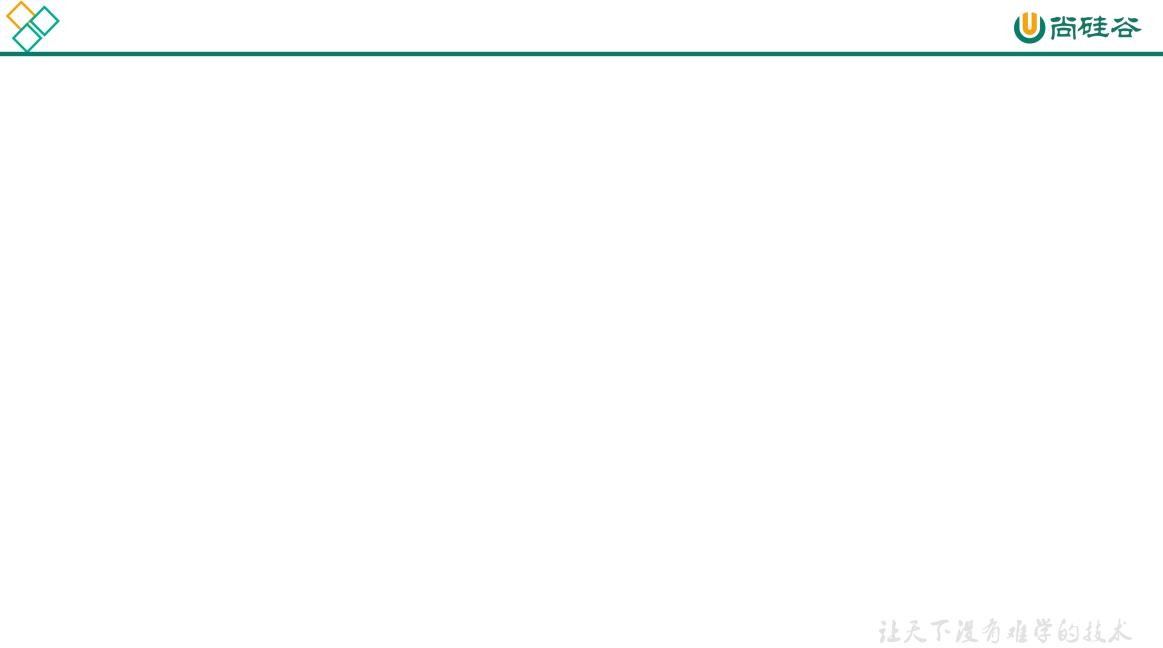
* 1. **NN** 和 **2NN** 工作机制

### 思考：NameNode 中的元数据是存储在哪里的？

首先，我们做个假设，如果存储在 NameNode 节点的磁盘中，因为经常需要进行随机访问，还有响应客户请求，必然是效率过低。因此，元数据需要存放在内存中。但如果只存在内存中，一旦断电，元数据丢失，整个集群就无法工作了。因此产生在磁盘中备份元数据的FsImage。

这样又会带来新的问题，当在内存中的元数据更新时，如果同时更新FsImage，就会导致效率过低，但如果不更新，就会发生一致性问题，一旦 NameNode 节点断电，就会产生数据丢失。因此，引入 Edits 文件（只进行追加操作，效率很高）。每当元数据有更新或者添加元数据时，修改内存中的元数据并追加到Edits 中。这样，一旦 NameNode 节点断电，可以通过 FsImage 和Edits 的合并，合成元数据。

但是，如果长时间添加数据到Edits 中，会导致该文件数据过大，效率降低，而且一旦断电，恢复元数据需要的时间过长。因此，需要定期进行 FsImage 和 Edits 的合并，如果这个操作由NameNode 节点完成，又会效率过低。因此，引入一个新的节点SecondaryNamenode， 专门用于 FsImage 和Edits 的合并。



NameNode工作机制

NameNode

1 请求是否需

要CheckPoint

Secondary NameNode

CheckPoint触发条件：

2 请求执行 1）定时时间到

CheckPoint 2）Edits中的数据满了

client

3记录操作日志、

更新滚动日志

1 加载编辑日志和镜像文件到内存

6 生成新的

Fsimage

5 加载到内存并合并

2 元数据的增删改请求

/user/atguigu/ss.avi

fsimage.chkpoint

7 拷贝到nn

8重命名成Fsimage

3 滚动正在写的Edits

4 拷贝到2nn

edits\_inprogress\_002

edits\_001

edits\_001

fsimage

fsimage.chkpoint

fsimage

edits\_inprogress\_001

内存

内存128G

（每个Block占元数据150byte）

4 内存数据增删改

1. 第一阶段：**NameNode** 启动

### 第一次启动 NameNode 格式化后，创建 Fsimage 和 Edits 文件。如果不是第一次启

动，直接加载编辑日志和镜像文件到内存。

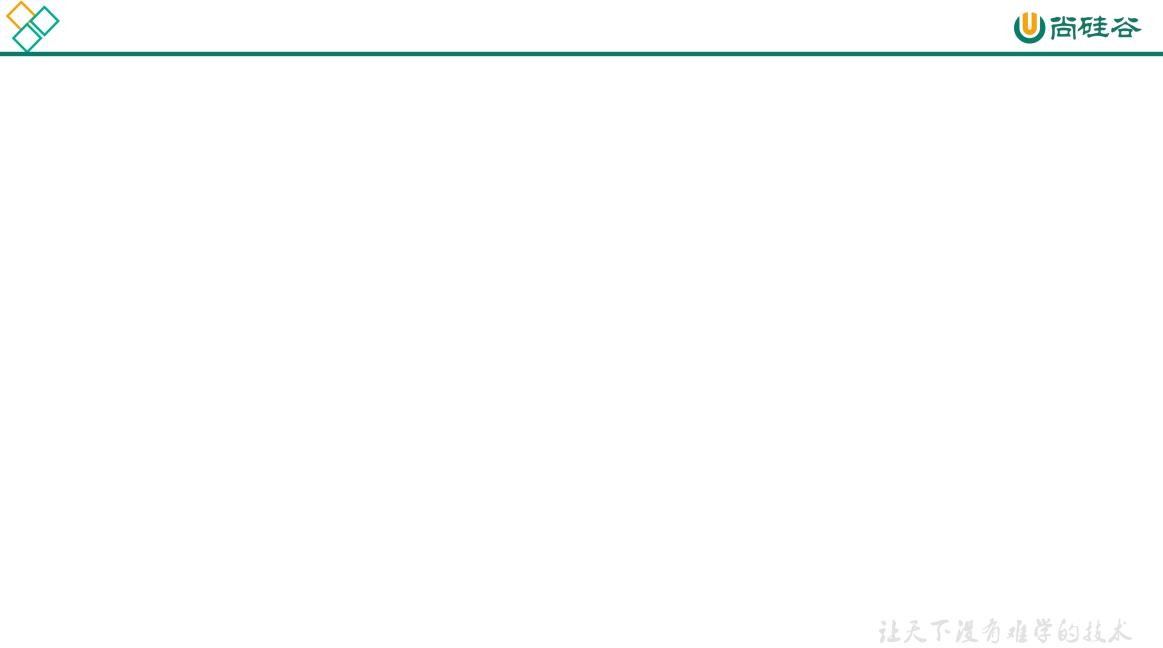
* 1. 客户端对元数据进行增删改的请求。
  2. NameNode 记录操作日志，更新滚动日志。
  3. NameNode 在内存中对元数据进行增删改。

1. 第二阶段：**Secondary NameNode** 工作

### Secondary NameNode 询问 NameNode 是否需要 CheckPoint。直接带回 NameNode

是否检查结果。

1. Secondary NameNode 请求执行 CheckPoint。
2. NameNode 滚动正在写的 Edits 日志。
3. 将滚动前的编辑日志和镜像文件拷贝到 Secondary NameNode。
4. Secondary NameNode 加载编辑日志和镜像文件到内存，并合并。
5. 生成新的镜像文件 fsimage.chkpoint。
6. 拷贝 fsimage.chkpoint 到 NameNode。
7. NameNode 将 fsimage.chkpoint 重新命名成 fsimage。
   1. **Fsimage** 和 **Edits** 解析



Fsimage和Edits概念

NameNode被格式化之后，将在/opt/module/hadoop-3.1.3/data/tmp/dfs/name/current目录中产生如下文件

fsimage\_0000000000000000000

fsimage\_0000000000000000000.md5 seen\_txid

VERSION

1. Fsimage文件：HDFS文件系统元数据的一个永久性的检查点，其中包含HDFS文件系统的所有目 录和文件inode的序列化信息。
2. Edits文件：存放HDFS文件系统的所有更新操作的路径，文件系统客户端执行的所有写操作首先

会被记录到Edits文件中。

1. seen\_txid文件保存的是一个数字，就是最后一个edits\_的数字
2. 每次NameNode启动的时候都会将Fsimage文件读入内存，加载Edits里面的更新操作，保证内存

中的元数据信息是最新的、同步的，可以看成NameNode启动的时候就将Fsimage和Edits文件进行了合并。

1. **oiv** 查看 **Fsimage** 文件

### （1）查看 oiv 和 oev 命令

[atguigu@hadoop102 current]$ hdfs

**oiv** apply the offline fsimage viewer to an fsimage

**oev** apply the offline edits viewer to an edits file

hdfs oiv -p 文件类型 -i 镜像文件 -o 转换后文件输出路径

（3）案例实操

[atguigu@hadoop102 current]$ pwd

/opt/module/hadoop-3.1.3/data/dfs/name/current

[atguigu@hadoop102 current]$ hdfs oiv -p XML -i fsimage\_0000000000000000025 -o /opt/module/hadoop-3.1.3/fsimage.xml

[atguigu@hadoop102 current]$ cat /opt/module/hadoop-3.1.3/fsimage.xml

将显示的 xml 文件内容拷贝到 Idea 中创建的 xml 文件中，并格式化。部分显示结果如

下。

<inode>

<id>16386</id>

<type>DIRECTORY</type>

<name>user</name>

<mtime>1512722284477</mtime>

<permission>atguigu:supergroup:rwxr-xr-x</permission>

<nsquota>-1</nsquota>

<dsquota>-1</dsquota>

</inode>

<inode>

<id>16387</id>

<type>DIRECTORY</type>

<name>atguigu</name>

<mtime>1512790549080</mtime>

<permission>atguigu:supergroup:rwxr-xr-x</permission>

<nsquota>-1</nsquota>

<dsquota>-1</dsquota>

</inode>

<inode>

<id>16389</id>

<type>FILE</type>

<name>wc.input</name>

<replication>3</replication>

<mtime>1512722322219</mtime>

<atime>1512722321610</atime>

<perferredBlockSize>134217728</perferredBlockSize>

<permission>atguigu:supergroup:rw-r--r--</permission>

<blocks>

<block>

<id>1073741825</id>

<genstamp>1001</genstamp>

<numBytes>59</numBytes>

</block>

</blocks>

</inode >

思考：可以看出，Fsimage 中没有记录块所对应 DataNode，为什么？

在集群启动后，要求 DataNode 上报数据块信息，并间隔一段时间后再次上报。

1. **oev** 查看 **Edits** 文件

### 基本语法

hdfs oev -p 文件类型 -i 编辑日志 -o 转换后文件输出路径

### 案例实操

[atguigu@hadoop102 current]$ hdfs oev -p XML -i edits\_0000000000000000012-0000000000000000013 -o /opt/module/hadoop- 3.1.3/edits.xml

[atguigu@hadoop102 current]$ cat /opt/module/hadoop-3.1.3/edits.xml

将显示的 xml 文件内容拷贝到Idea 中创建的 xml 文件中，并格式化。显示结果如下。

<?xml version="1.0" encoding="UTF-8"?>

<EDITS>

<EDITS\_VERSION>-63</EDITS\_VERSION>

<RECORD>

<OPCODE>OP\_START\_LOG\_SEGMENT</OPCODE>

<DATA>

<TXID>129</TXID>

</DATA>

</RECORD>

<RECORD>

<OPCODE>OP\_ADD</OPCODE>

<DATA>

<TXID>130</TXID>

<LENGTH>0</LENGTH>

<INODEID>16407</INODEID>

<PATH>/hello7.txt</PATH>

<REPLICATION>2</REPLICATION>

<MTIME>1512943607866</MTIME>

<ATIME>1512943607866</ATIME>

<BLOCKSIZE>134217728</BLOCKSIZE>

<CLIENT\_NAME>DFSClient\_NONMAPREDUCE\_- 1544295051\_1</CLIENT\_NAME>

<CLIENT\_MACHINE>192.168.10.102</CLIENT\_MACHINE>

<OVERWRITE>true</OVERWRITE>

<PERMISSION\_STATUS>

<USERNAME>atguigu</USERNAME>

<GROUPNAME>supergroup</GROUPNAME>

<MODE>420</MODE>

</PERMISSION\_STATUS>

<RPC\_CLIENTID>908eafd4-9aec-4288-96f1- e8011d181561</RPC\_CLIENTID>

<RPC\_CALLID>0</RPC\_CALLID>

</DATA>

</RECORD>

<RECORD>

<OPCODE>OP\_ALLOCATE\_BLOCK\_ID</OPCODE>

<DATA>

<TXID>131</TXID>

<BLOCK\_ID>1073741839</BLOCK\_ID>

</DATA>

</RECORD>

<RECORD>

<OPCODE>OP\_SET\_GENSTAMP\_V2</OPCODE>

<DATA>

<TXID>132</TXID>

<GENSTAMPV2>1016</GENSTAMPV2>

</DATA>

</RECORD>

<RECORD>

<OPCODE>OP\_ADD\_BLOCK</OPCODE>

<DATA>

<TXID>133</TXID>

<PATH>/hello7.txt</PATH>

<BLOCK>

<BLOCK\_ID>1073741839</BLOCK\_ID>

<NUM\_BYTES>0</NUM\_BYTES>

<GENSTAMP>1016</GENSTAMP>

</BLOCK>

<RPC\_CLIENTID></RPC\_CLIENTID>

<RPC\_CALLID>-2</RPC\_CALLID>

</DATA>

</RECORD>

<RECORD>

<OPCODE>OP\_CLOSE</OPCODE>

<DATA>

<TXID>134</TXID>

<LENGTH>0</LENGTH>

<INODEID>0</INODEID>

<PATH>/hello7.txt</PATH>

<REPLICATION>2</REPLICATION>

<MTIME>1512943608761</MTIME>

<ATIME>1512943607866</ATIME>

<BLOCKSIZE>134217728</BLOCKSIZE>

<CLIENT\_NAME></CLIENT\_NAME>

<CLIENT\_MACHINE></CLIENT\_MACHINE>

<OVERWRITE>false</OVERWRITE>

<BLOCK>

<BLOCK\_ID>1073741839</BLOCK\_ID>

<NUM\_BYTES>25</NUM\_BYTES>

<GENSTAMP>1016</GENSTAMP>

</BLOCK>

<PERMISSION\_STATUS>

<USERNAME>atguigu</USERNAME>

<GROUPNAME>supergroup</GROUPNAME>

<MODE>420</MODE>

</PERMISSION\_STATUS>

</DATA>

</RECORD>

</EDITS >

思考：NameNode 如何确定下次开机启动的时候合并哪些Edits？

* 1. **CheckPoint** 时间设置

1. 通常情况下，**SecondaryNameNode** 每隔一小时执行一次。

### [hdfs-default.xml]

<property>

<name>dfs.namenode.checkpoint.period</name>

<value>3600s</value>

</property>

1. 一分钟检查一次操作次数，当操作次数达到 **1** 百万时，**SecondaryNameNode** 执行一次。

<property>

<name>dfs.namenode.checkpoint.txns</name>

<value>1000000</value>

<description>操作动作次数</description>

</property>

<property>

<name>dfs.namenode.checkpoint.check.period</name>

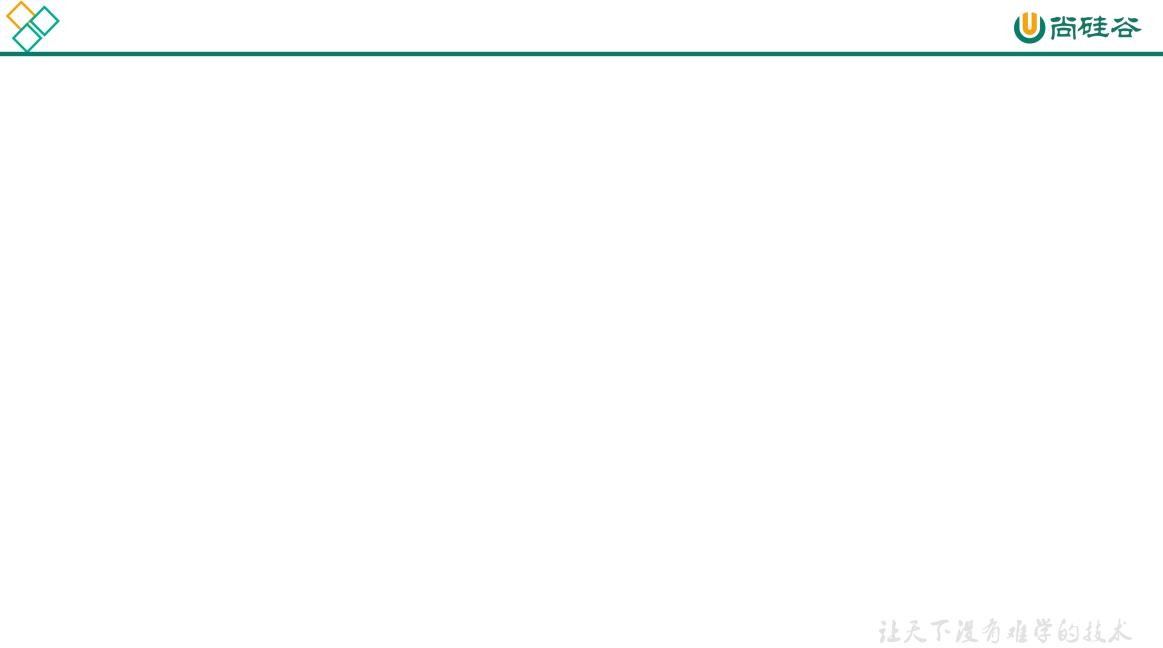
<value>60s</value>

<description> 1 分钟检查一次操作次数</description>

</property>

第 **6** 章 **DataNode**

* 1. **DataNode** 工作机制



DataNode工作机制

NameNode

2 注册成功

1 DataNode启动后向NameNode注册。

DataNode1

3 以后每周期（

6小时）上报所有块信息。

4 心跳每3秒一次，心跳返回结果带有NameNode给该DataNode的

命令

5 超过10分钟+30秒没有收到DataNode2的心跳， 则认为该节点不可用

DataNode2

DataNode3

Block1

Block2

Block3

Block3

Block1

Block2

数据、数据长度、校验和、

时间戳

数据、数据长度、校验和、时间戳

数据、数据长度、校验和、

时间戳

数据、数据长度、校验和、

时间戳

数据、数据长度、校验和、

时间戳

数据、数据长度、校验和、

时间戳

元数据

2 DataNode注册成功

### 一个数据块在DataNode 上以文件形式存储在磁盘上，包括两个文件，一个是数据

本身，一个是元数据包括数据块的长度，块数据的校验和，以及时间戳。

* + 1. DataNode 启动后向NameNode 注册，通过后，周期性（6 小时）的向 NameNode 上报所有的块信息。

DN 向 NN 汇报当前解读信息的时间间隔，默认 6 小时；

<property>

<name>dfs.blockreport.intervalMsec</name>

<value>21600000</value>

<description>Determines block reporting interval in milliseconds.</description>

</property>

DN 扫描自己节点块信息列表的时间，默认 6 小时

<property>

<name>dfs.datanode.directoryscan.interval</name>

<value>21600s</value>

<description>Interval in seconds for Datanode to scan data

directories and reconcile the difference between blocks in memory and on the disk.

Support multiple time unit suffix(case insensitive), as described in dfs.heartbeat.interval.

</description>

</property>

* + 1. 心跳是每 3 秒一次，心跳返回结果带有NameNode 给该 DataNode 的命令如复制块

数据到另一台机器，或删除某个数据块。如果超过 10 分钟没有收到某个DataNode 的心跳， 则认为该节点不可用。

* + 1. 集群运行中可以安全加入和退出一些机器。
  1. 数据完整性

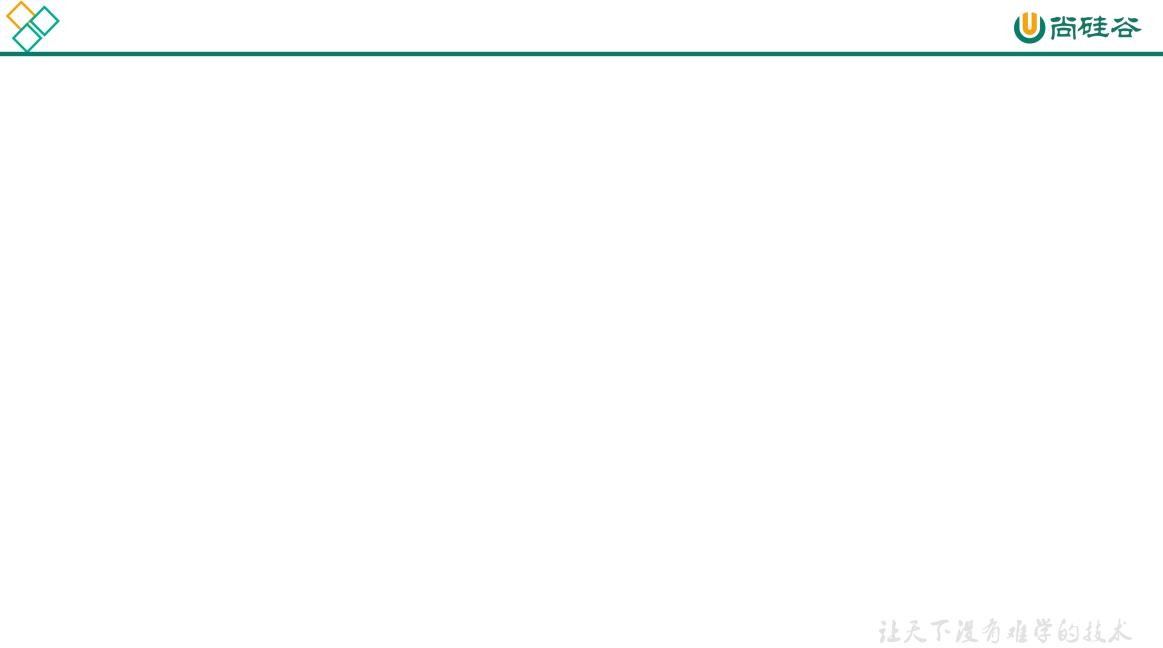
思考：如果电脑磁盘里面存储的数据是控制高铁信号灯的红灯信号（1）和绿灯信号（0），但是存储该数据的磁盘坏了，一直显示是绿灯，是否很危险？同理 DataNode 节点上的数据损坏了，却没有发现，是否也很危险，那么如何解决呢？

如下是 DataNode 节点保证数据完整性的方法。

1. 当 DataNode 读取 Block 的时候，它会计算CheckSum。
2. 如果计算后的 CheckSum，与Block 创建时值不一样，说明 Block 已经损坏。
3. Client 读取其他DataNode 上的 Block。

（4）常见的校验算法 crc（32），md5（128），sha1（160）

（5）DataNode 在其文件创建后周期验证CheckSum。



数据完整性

原始数据封装

对接收到的数据重新校验

待传输数据 奇偶校验位

待传输数据 奇偶校验位

0100 0001

网络传输

0100 0011

待传输数据 奇偶校验位 待传输数据 奇偶校验位

0100 1001

0100 1001

待传输数据 crc校验位

对原始数据重新crc计算，然后和传输过来的crc校验位比较

，看是否一致

待传输数据 crc校验位

110

0100 1001

110

0100 1001

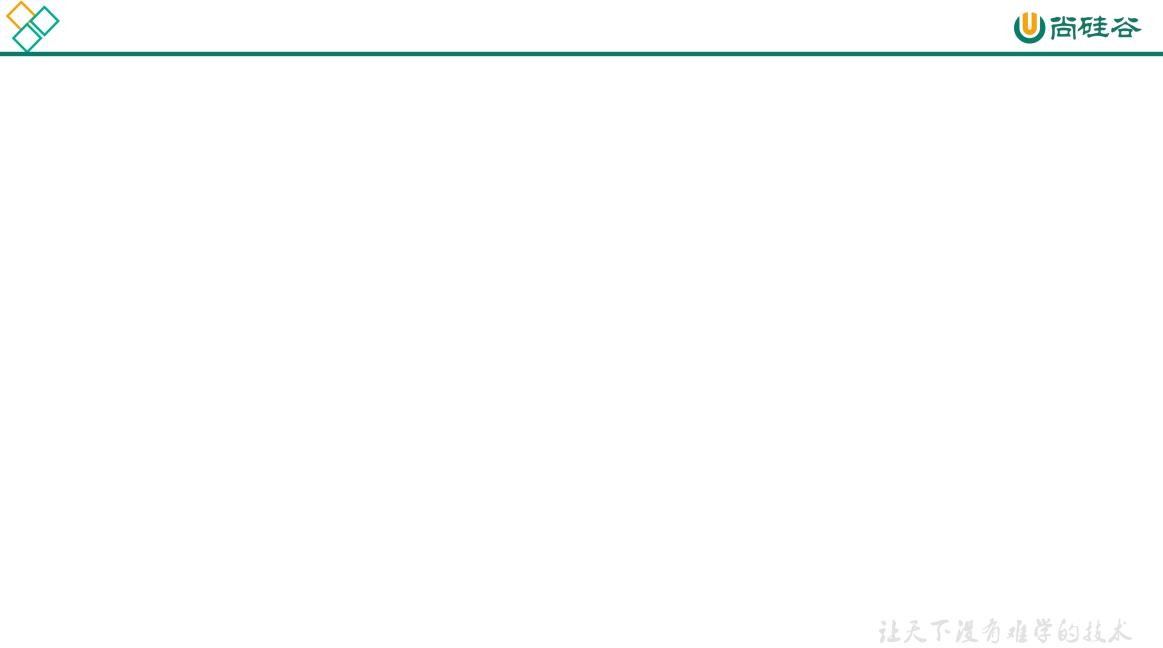
1

1

0

0

* 1. 掉线时限参数设置



DataNode掉线时限参数设置

1、DataNode进程死亡或

者网络故障造成DataNode

无法与NameNode通信

NameNode

2、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。

DataNode

3、HDFS默认的超时时长为10分钟+30秒。

4、如果定义超时时间为TimeOut，则超时时长的计算公式为：

TimeOut = 2 \* dfs.namenode.heartbeat.recheck-interval + 10 \* dfs.heartbeat.interval。

而默认的dfs.namenode.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。

<property>

<name>dfs.namenode.heartbeat.recheck-interval</name>

<value>300000</value>

</property>

<property>

<name>dfs.heartbeat.interval</name>

<value>3</value>

</property>