

HDFS分布式文件系统

一、课前准备

1. 安装VMware15虚拟化软件
2. 安装CentOS 7虚拟机3个
3. 安装3节点的hadoop3集群（以上参考文档¹）
4. 某台虚拟机节点安装IDEA

二、课堂主题

本堂课主要围绕HDFS进行讲解。主要包括三方面

1. 架构原理
2. 核心概念
3. HDFS命令行
4. HDFS编程

三、课堂目标

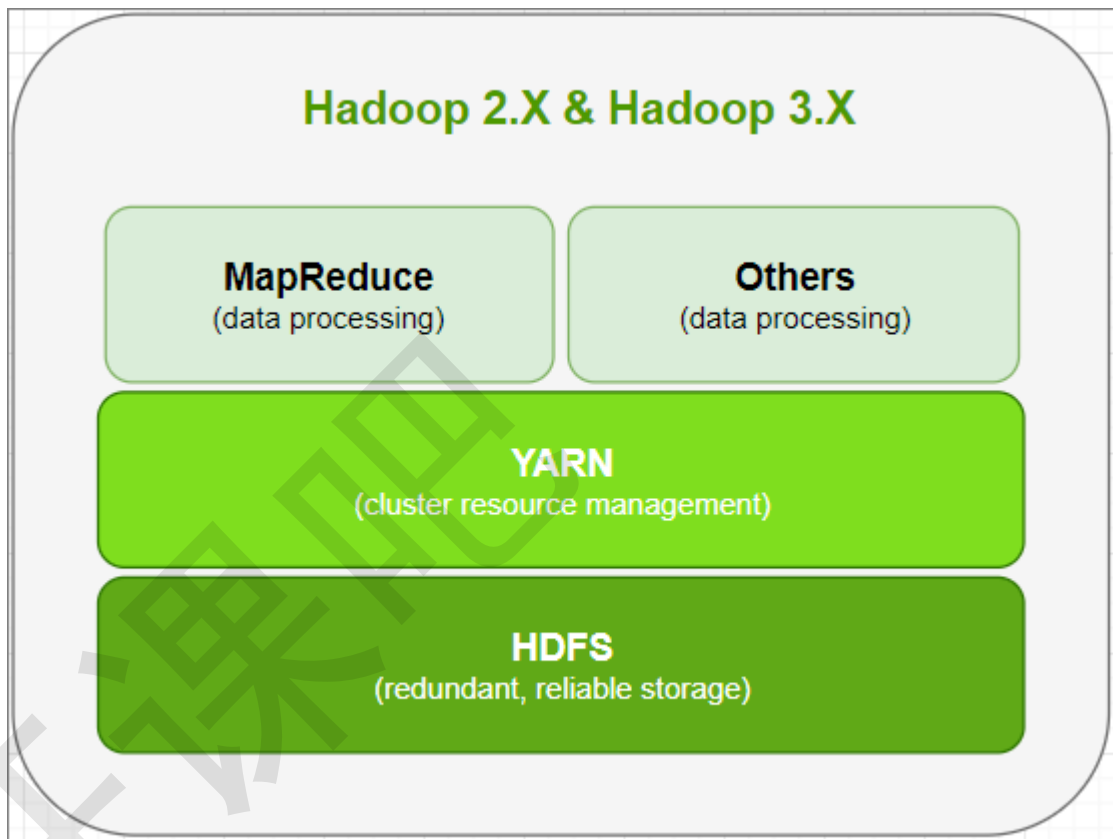
1. 理解分布式思想
2. 理解并描述HDFS工作原理
3. 理解HDFS容错机制
4. 会搭建高可用集群
5. 理解HDFS如何解决大量小文件存储问题
6. 掌握如何使用java api操作HDFS

四、知识要点

1. Hadoop是什么（20分钟）

1.1 Hadoop架构

Hadoop由三个模块组成：**分布式**存储HDFS、分布式计算MapReduce、资源调度引擎Yarn



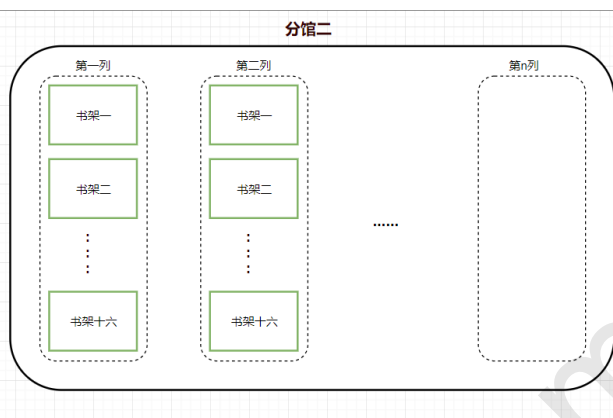
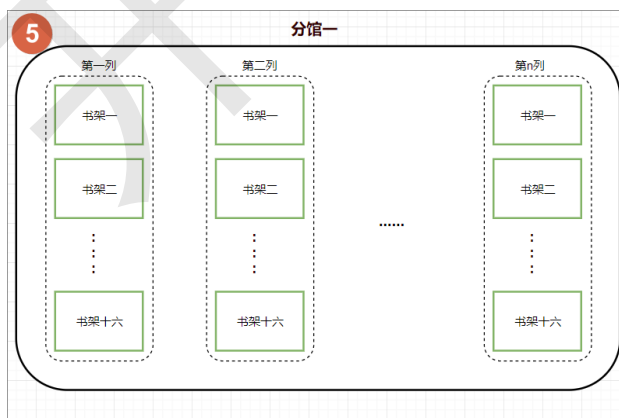
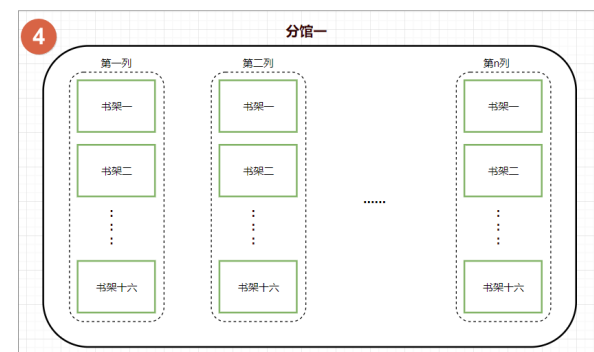
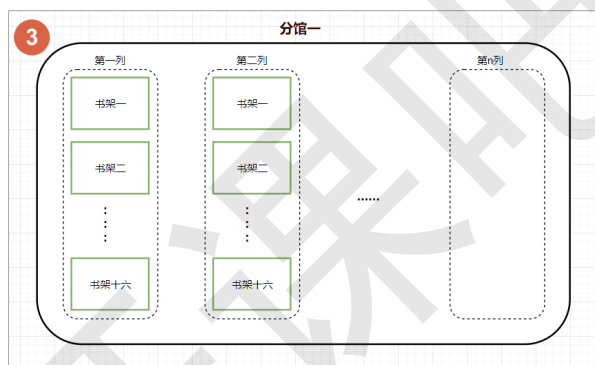
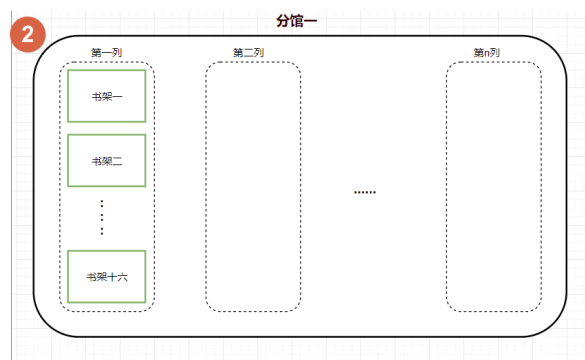
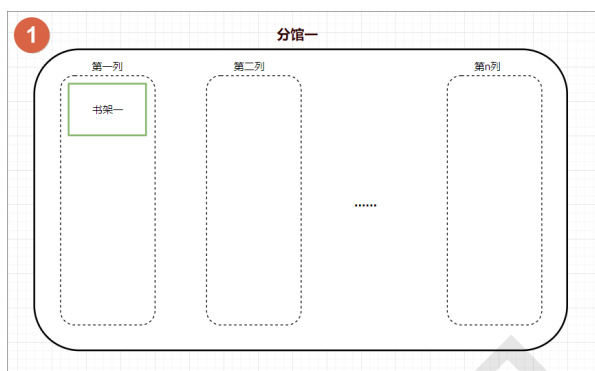
1.2 从生活中寻找灵感

1.2.1 存储书籍

分馆、列、书架、书

举例：国家图书馆从无到有开始创建，图书逐渐增多，日常需要统计书籍情况

- 1、初期一个分馆、规划很多列（一列放置多个书架）、一个书架
- 2、持续增加藏书，增加书架
- 3、增加第二列书架
- 4、增加 n 列书架
- 5、再增加一个分馆



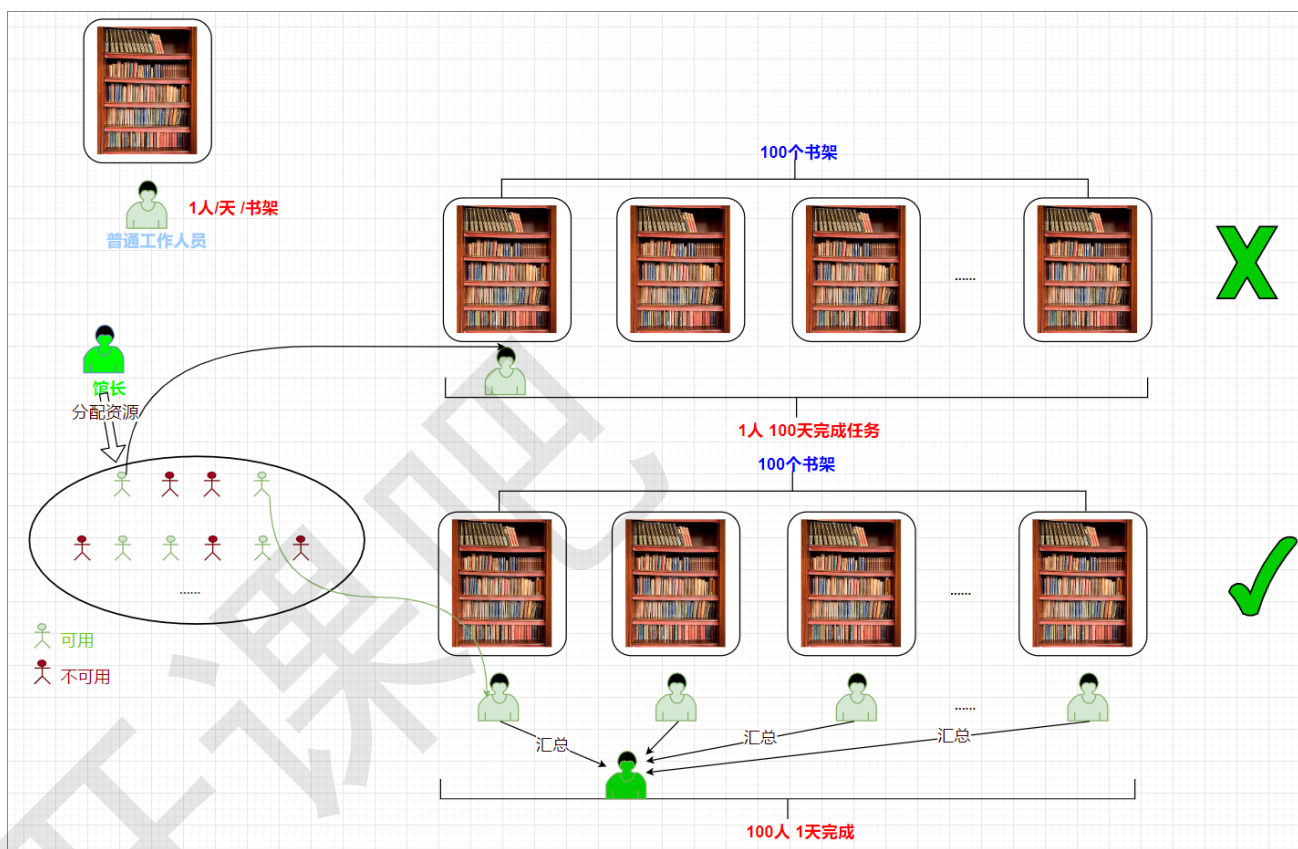
1.2.2 分配人员、统计书籍

现需要统计所有书籍中，书名包含"Hadoop"关键字的书有多少本？

馆长：分配普通工作人员干活

普通工作人员（资源）：图书馆的工作人员

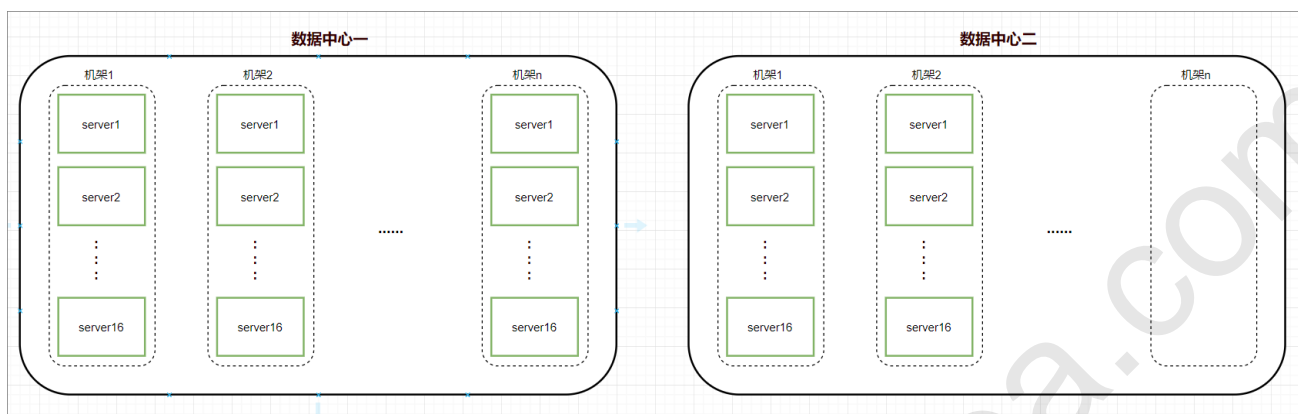
事（逻辑）：统计包含Hadoop关键字的书



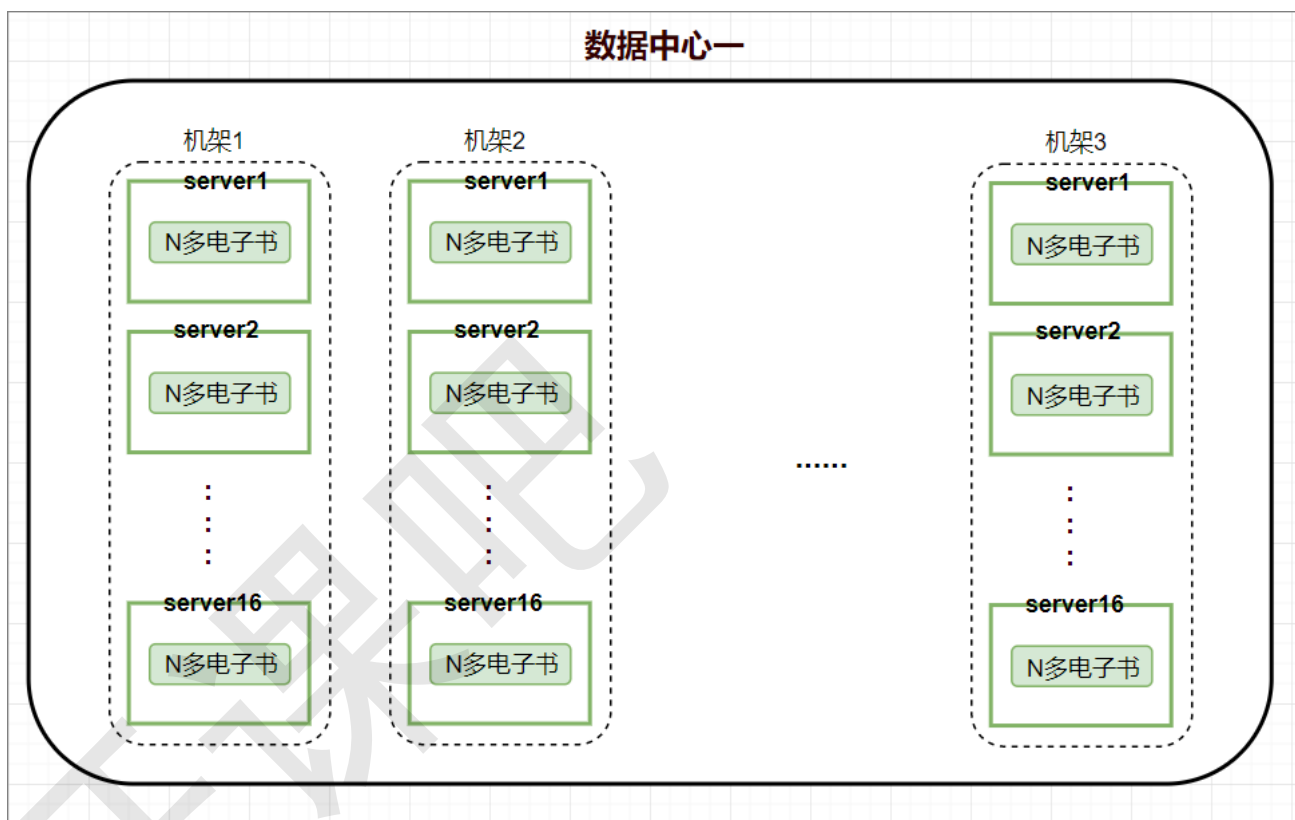
1.2.3 存储书的电子版

数据中心、机架、服务器、大文件

数据中心、机架、服务器、服务器存储电子书 -> HDFS



以一个数据中心为例



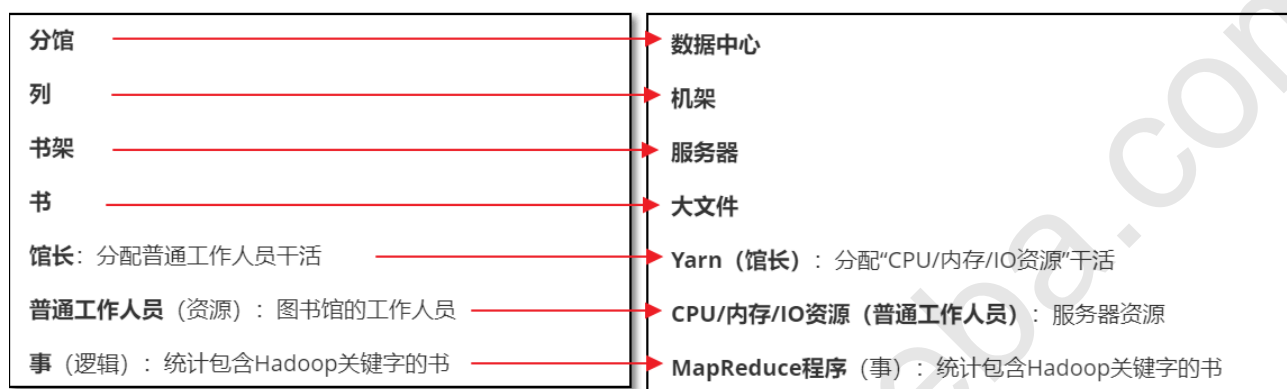
1.2.4 分配资源、统计书籍

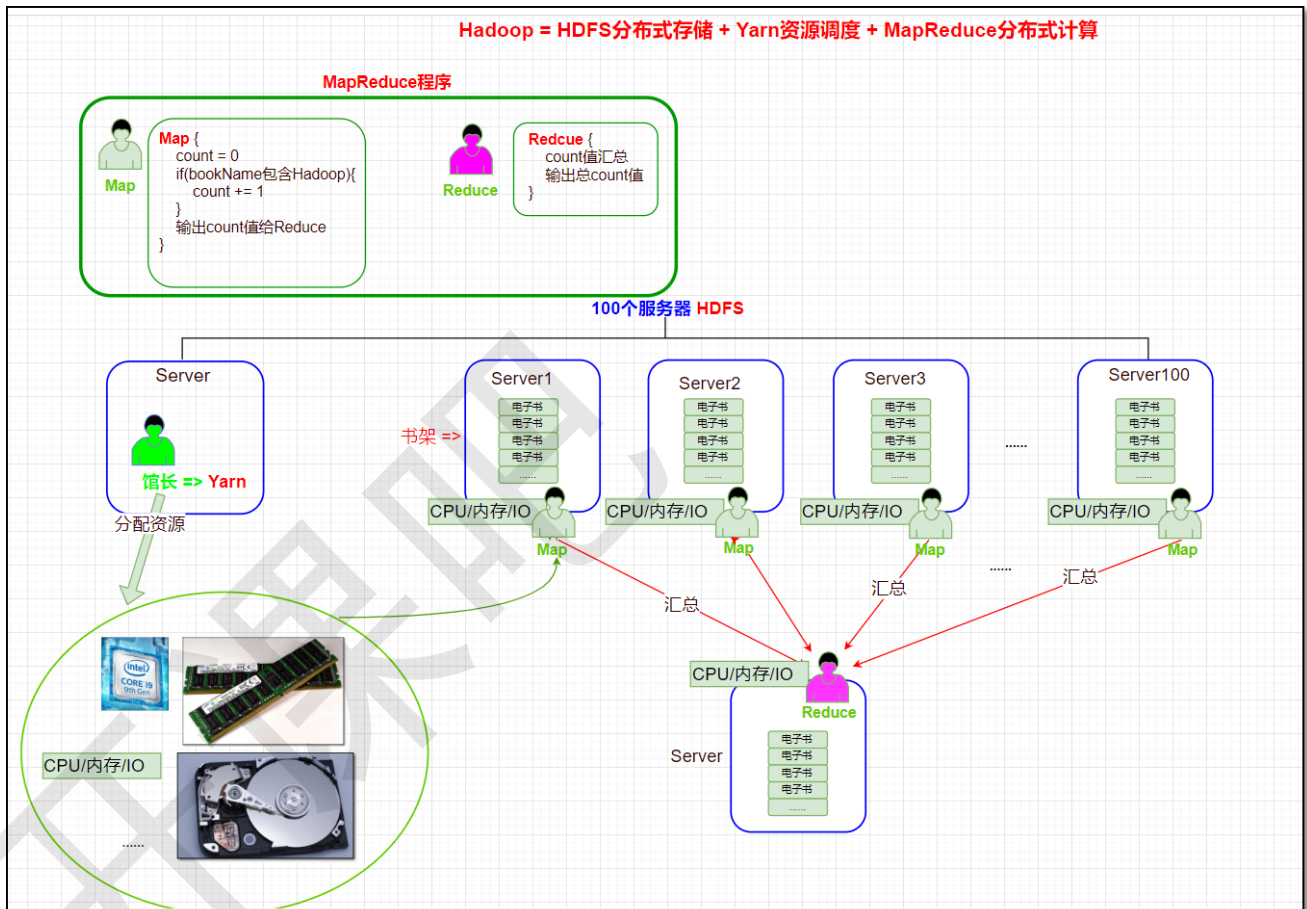
现需要MapReduce程序统计所有书籍中，书名包含"Hadoop"关键字的书有多少本？

Yarn (馆长)：分配“CPU/内存/IO资源”干活

CPU/内存/IO资源 (普通工作人员)：服务器资源

MapReduce程序 (事)：统计包含Hadoop关键字的书





1.3 分布式是什么

分布式：利用一批通过网络连接的、廉价普通的机器，完成单个机器无法完成的存储、计算任务

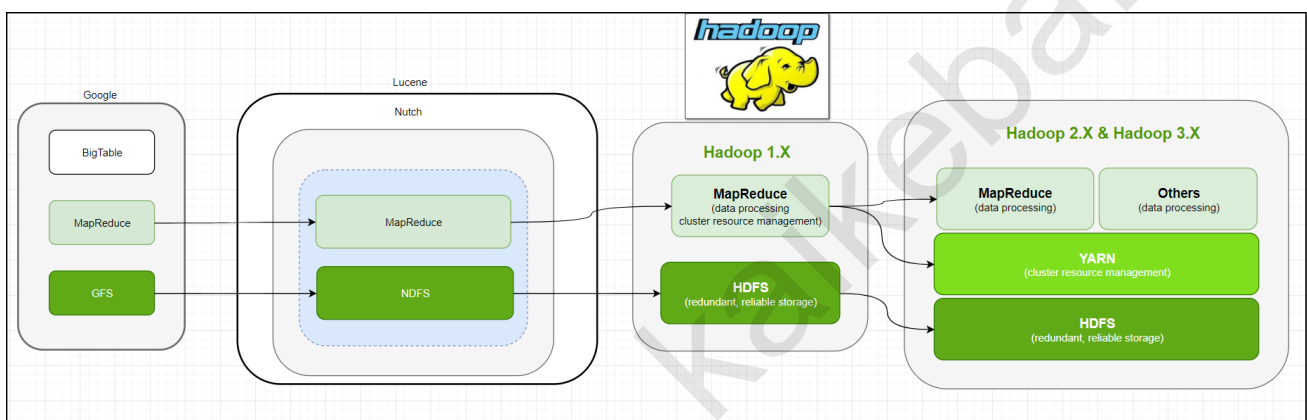
1.4 HDFS是什么

Hadoop分布式文件系统

1.5 为什么使用HDFS

高可用、容错、可扩展

1.6 Hadoop历史



- Hadoop作者Doug Cutting
- Apache Lucene是一个文本搜索系统库

- Apache Nutch作为前者的一部分，主要包括web爬虫、全文检索；2003年“谷歌分布式文件系统GFS”论文，2004年开源版本**NDFS**
- 2004年“谷歌MapReduce”论文，2005年Nutch开源版MapReduce



2. HDFS初体验（10分钟）

2.1 HDFS命令

linux shell命令风格

参考课件《Hadoop fs命令》

2.2 HDFS编程

java 编程的方式




2.3 WEB UI界面

node-02:50070/explorer.html#/

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Browse Directory

Permission denied: user=dr.who, access=WRITE, inode="/":bruce:supergroup:drwxr-xr-x

/ Go!   

Show 25 entries Search:

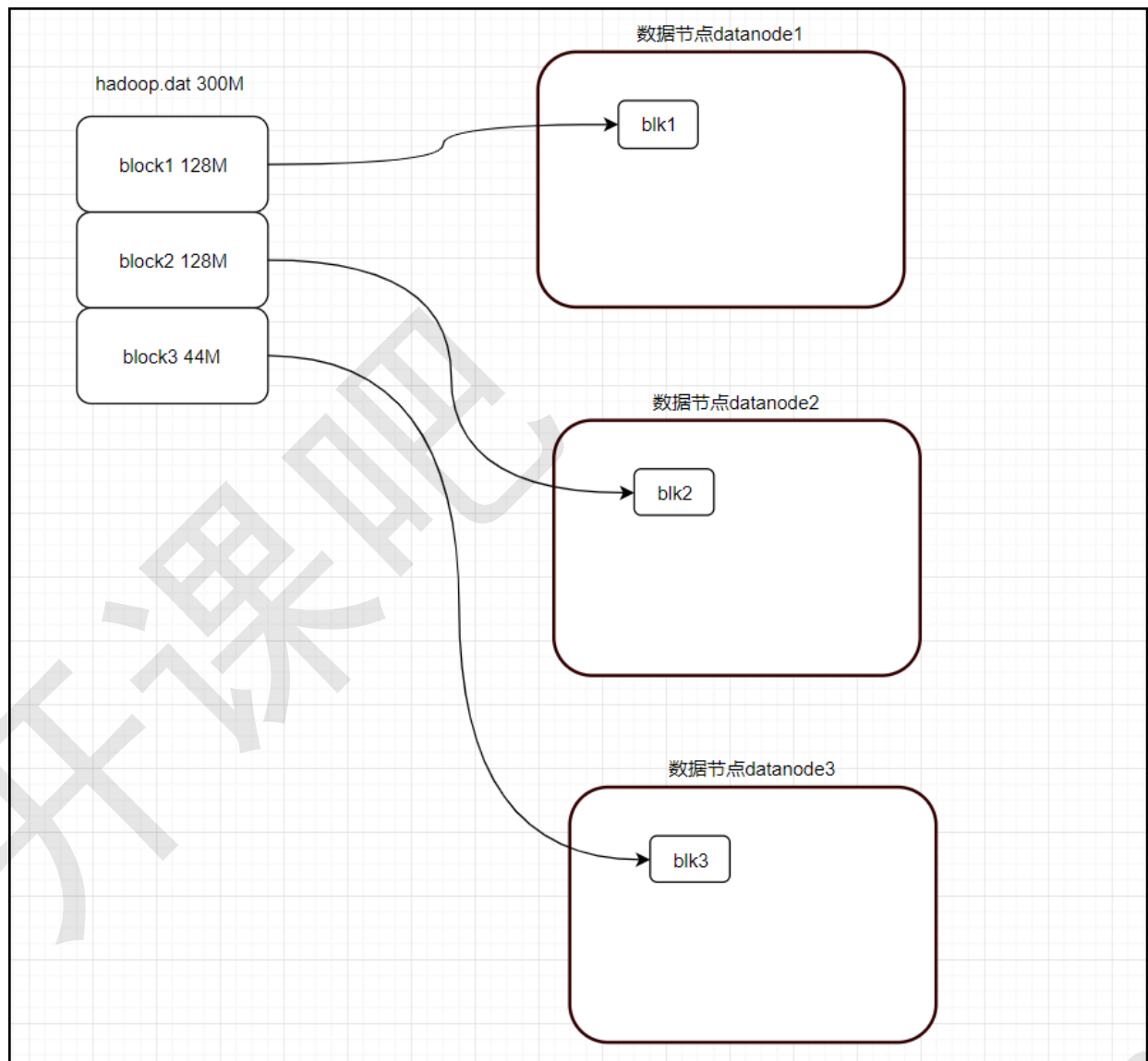
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	bruce	supergroup	0 B	Jun 21 18:04	0	0 B	0621
-rw-r--r--	bruce	supergroup	0 B	Jun 21 18:00	3	128 MB	1.txt
-rw-r--r--	bruce	supergroup	75 B	Jun 27 22:02	3	128 MB	BigFile.txt
-rw-r--r--	bruce	supergroup	21.61 KB	Jun 27 13:23	3	128 MB	NOTICE.txt
-rw-r--r--	bruce	supergroup	1.33 KB	Jun 28 16:48	3	128 MB	README.txt
-rw-r--r--	bruce	supergroup	110 B	Jun 28 13:58	3	128 MB	fs.txt
-rw-r--r--	bruce	supergroup	8.38 KB	Jun 21 18:02	3	128 MB	hadoop
-rw-r--r--	bruce	supergroup	139.79 MB	Jun 28 17:11	3	128 MB	hdfs.mp4
drwxr-xr-x	bruce	supergroup	0 B	May 08 11:45	0	0 B	home

3. 核心概念block (20分钟)

3.1 数据块block

3.1.1 HDFS block块

- HDFS3.x上的文件，是按照128M为单位，切分成一个个block，分散的存储在集群的不同数据节点datanode上
- 问：HDFS中一个44M大小的块会不会占据128M的空间？
- 问：这样存储有没有问题？

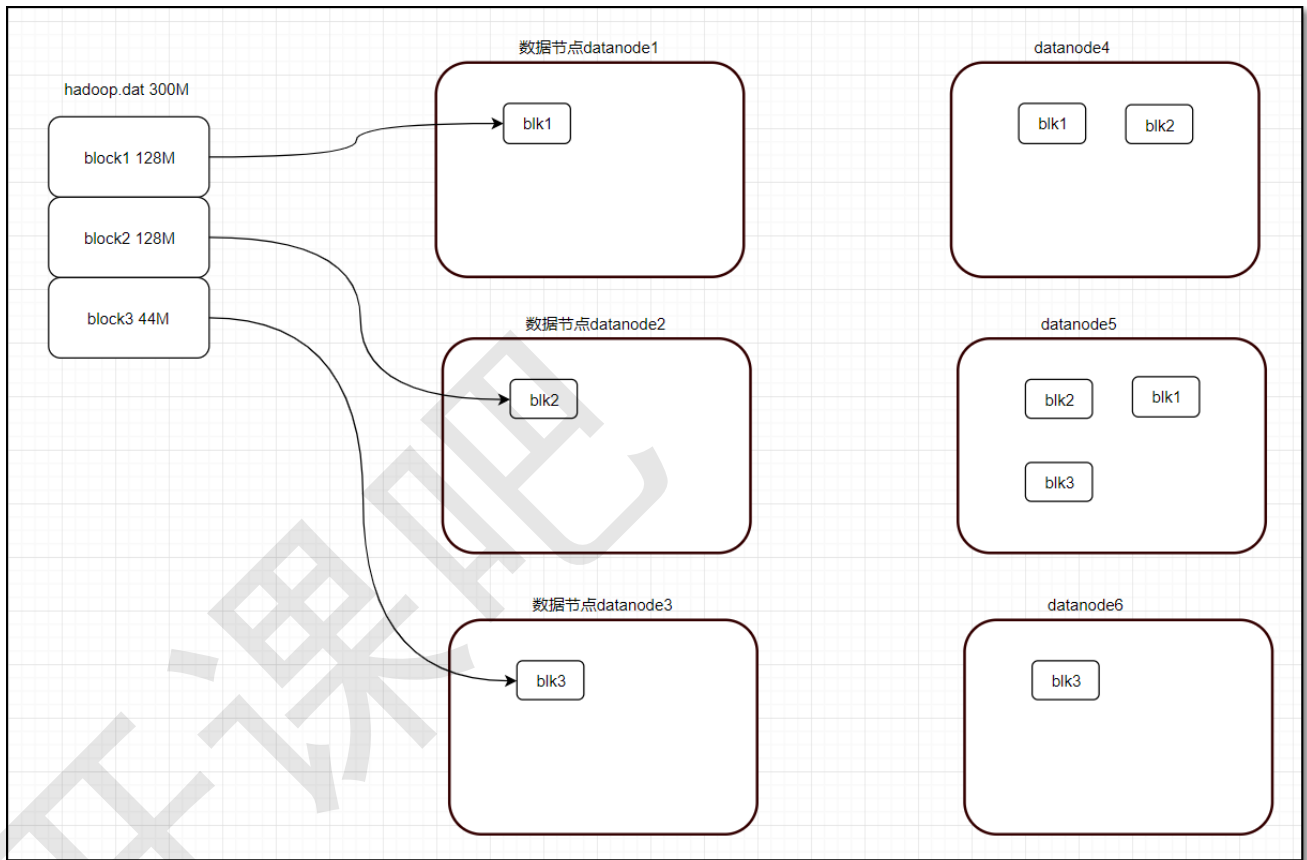


3.2 block副本

保证数据的可用及容错

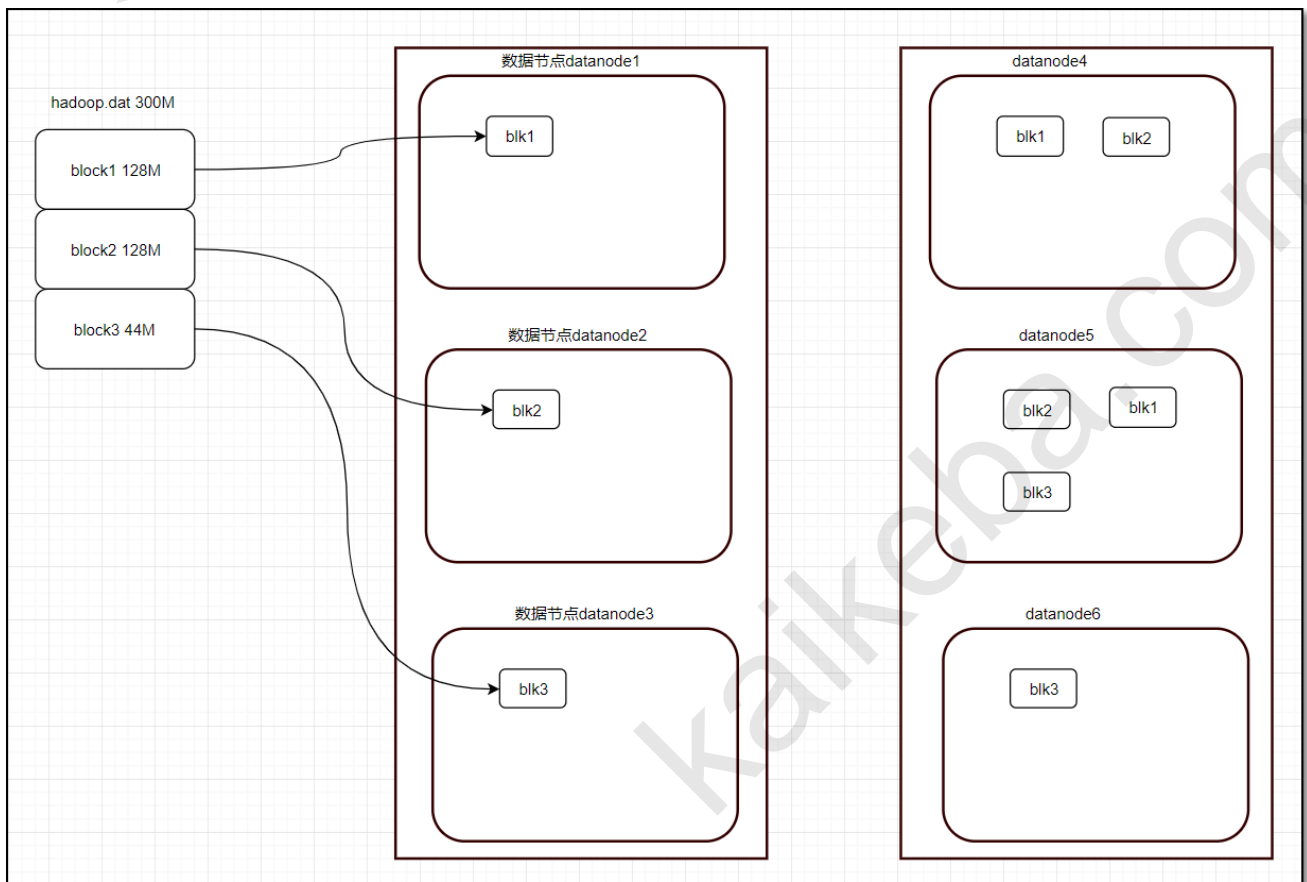
- replication = 3
- dfs-site.xml

```
<property>
  <name>dfs.replication</name>
  <value>3</value>
</property>
```



- 实际机房中，会有**机架**，每个机架上若干服务器

3.3 机架存储策略



3.4 block的一些操作

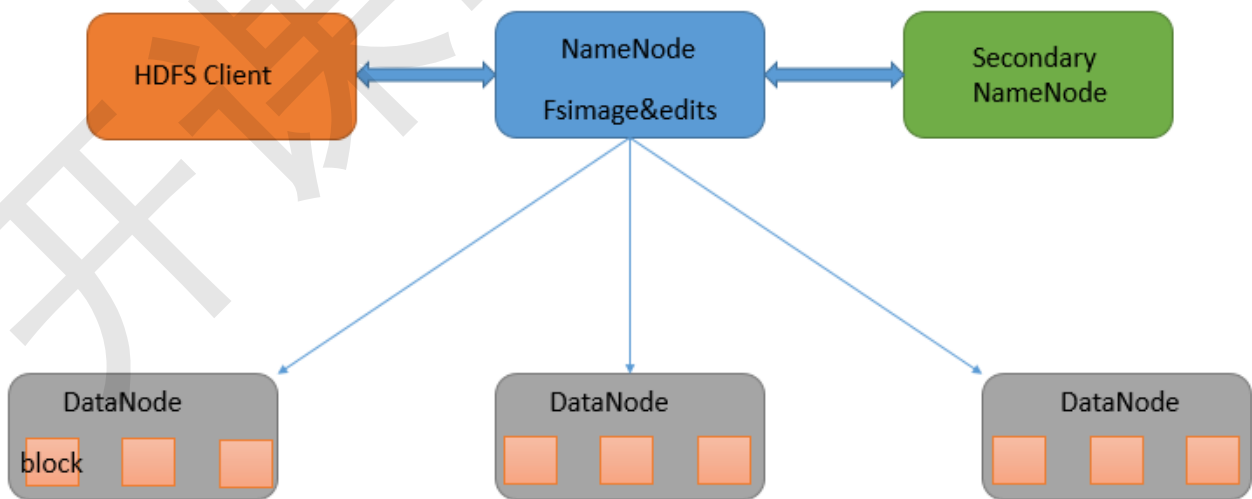
- 设置文件副本数，有什么用？

```
hadoop fs -setrep -R 4 /path
```

- 查看文件的块信息？

```
hdfs fsck /02-041-0029.mp4 -files -blocks -locations
```

4. HDFS体系架构 (20分钟)



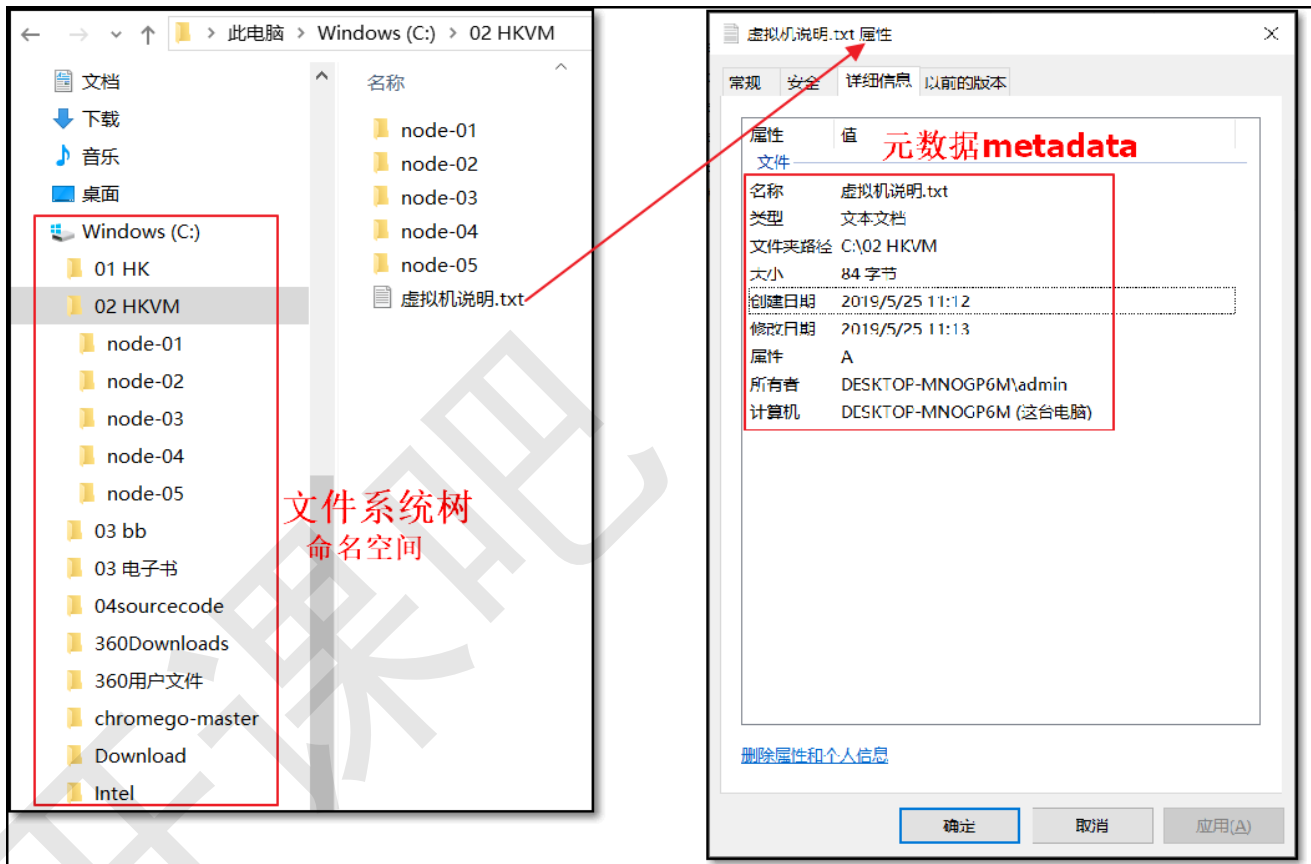
HDFS是主从架构Master/Slave、管理节点/工作节点

4.1 NameNode

fs文件系统，用来存储、读取数据

读文件 -> 找到文件 -> 在哪 + 叫啥？

window -> 虚拟机说明.txt

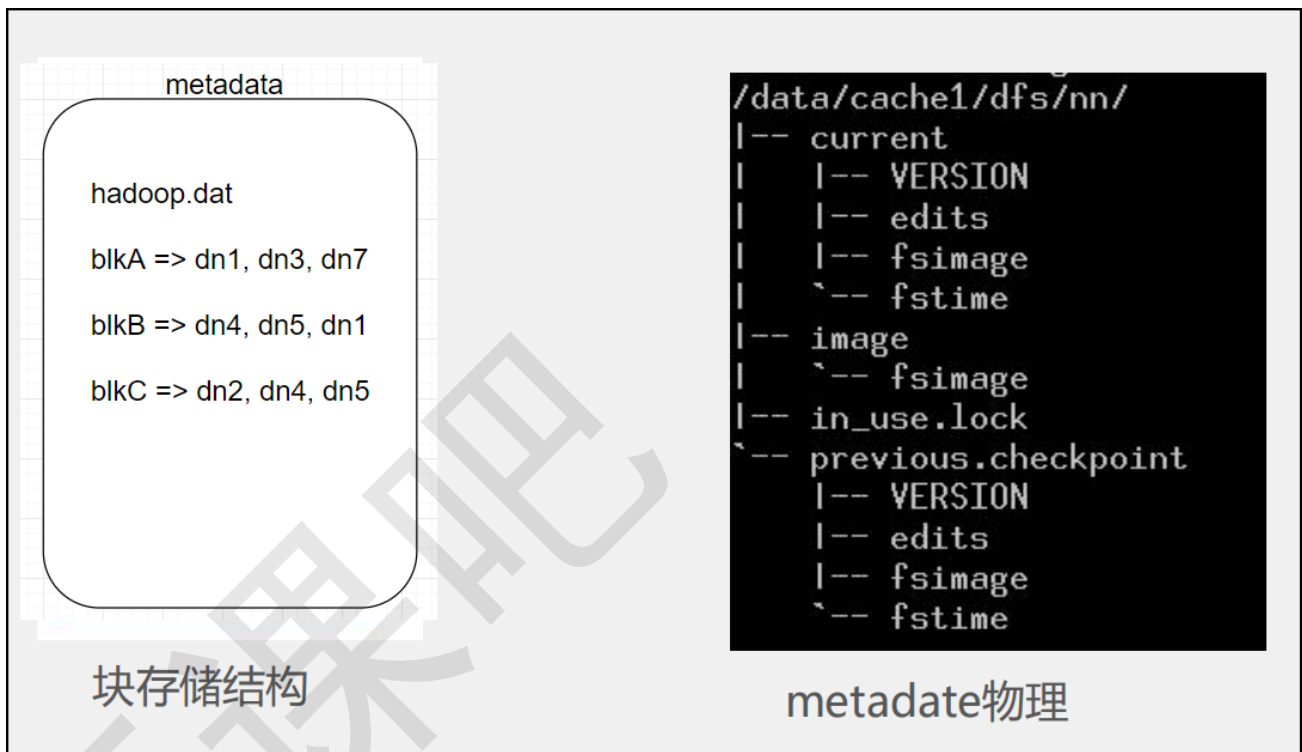


HDFS也是fs，它也有metadata；由NameNode存储在其内存中

1. 管理节点，负责管理文件系统和命名空间，存放了HDFS的元数据；
2. 元数据信息包括文件系统树、整棵树所有的文件和目录、每个文件的块列表、块所在的datanode等；
3. 元数据信息以命名空间镜像文件fsimage和编辑日志（ edits log ）的方式保存

fsimage：元数据镜像文件，保存了文件系统目录树信息以及文件和块的对应关系

edits log：日志文件，保存文件系统的更改记录



4.2 DataNode

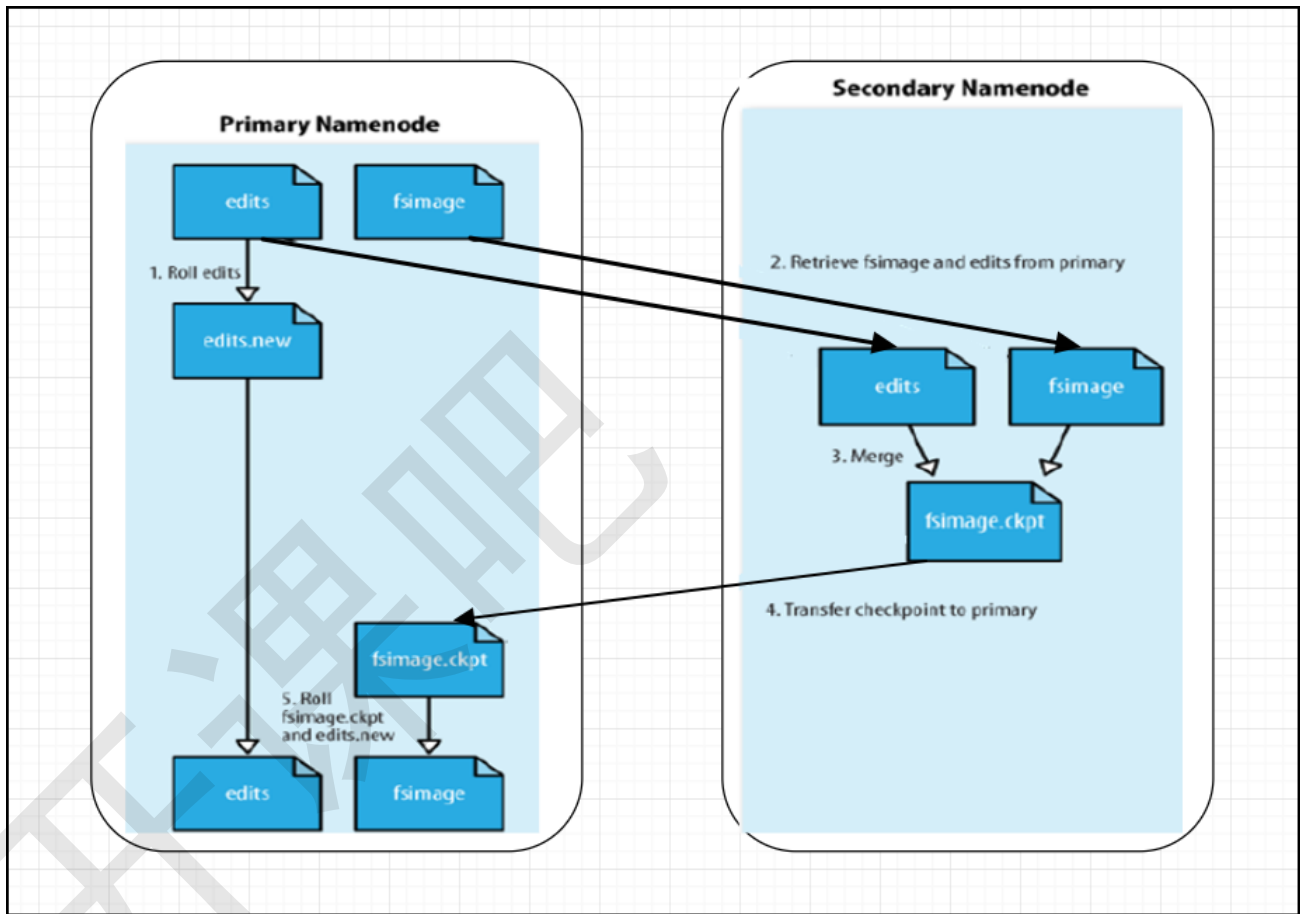
- 存储block，以及block元数据包括数据块的长度、块数据的校验和、时间戳

4.3 SeconddaryNameNode

问：

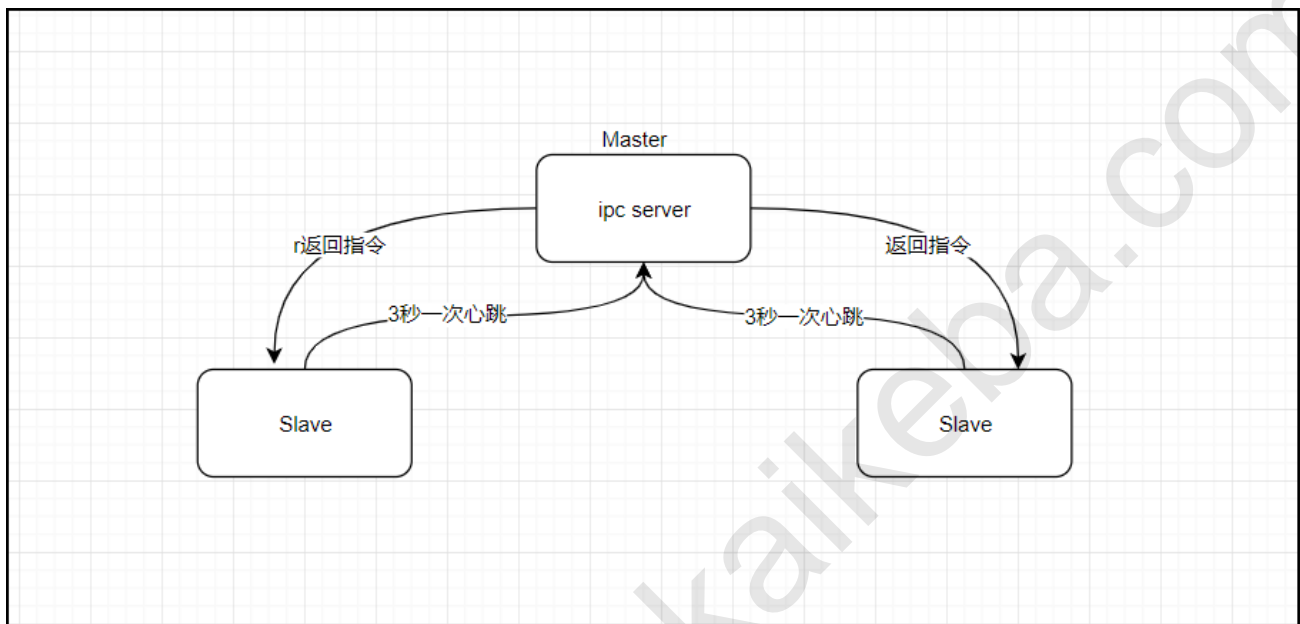
1. 为什么元数据存储>NameNode在内存中？
2. 这样做有什么问题？
3. 怎么解决？

它一般部署在另外一台节点上，因为它需要占用大量的CPU时间，并需要与namenode一样多的内存，来执行合并操作



6. HDFS机制

6.1 心跳机制



工作原理：

1. master启动的时候，会开一个ipc server在那里。

2. slave启动，连接master，每隔3秒钟向master发送一个“心跳”，携带状态信息；
3. master通过这个心跳的返回值，向slave节点传达指令

作用：

1. Namenode全权管理数据块的复制，它周期性地从集群中的每个Datanode接收心跳信号和块状态报告 (Blockreport)。接收到心跳信号意味着该Datanode节点工作正常。块状态报告包含了一个该 Datanode上所有数据块的列表
2. DataNode启动后向NameNode注册，通过后，周期性（1小时）的向 NameNode上报所有的块的列表；
每3秒向NameNode发一次心跳，返回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NameNode超过10分钟没有收到某个DataNode 的心跳，则认为该节点不可用。
3. hadoop集群刚开始启动时，会进入安全模式（99.9%），就用到了心跳机制

6.2 负载均衡

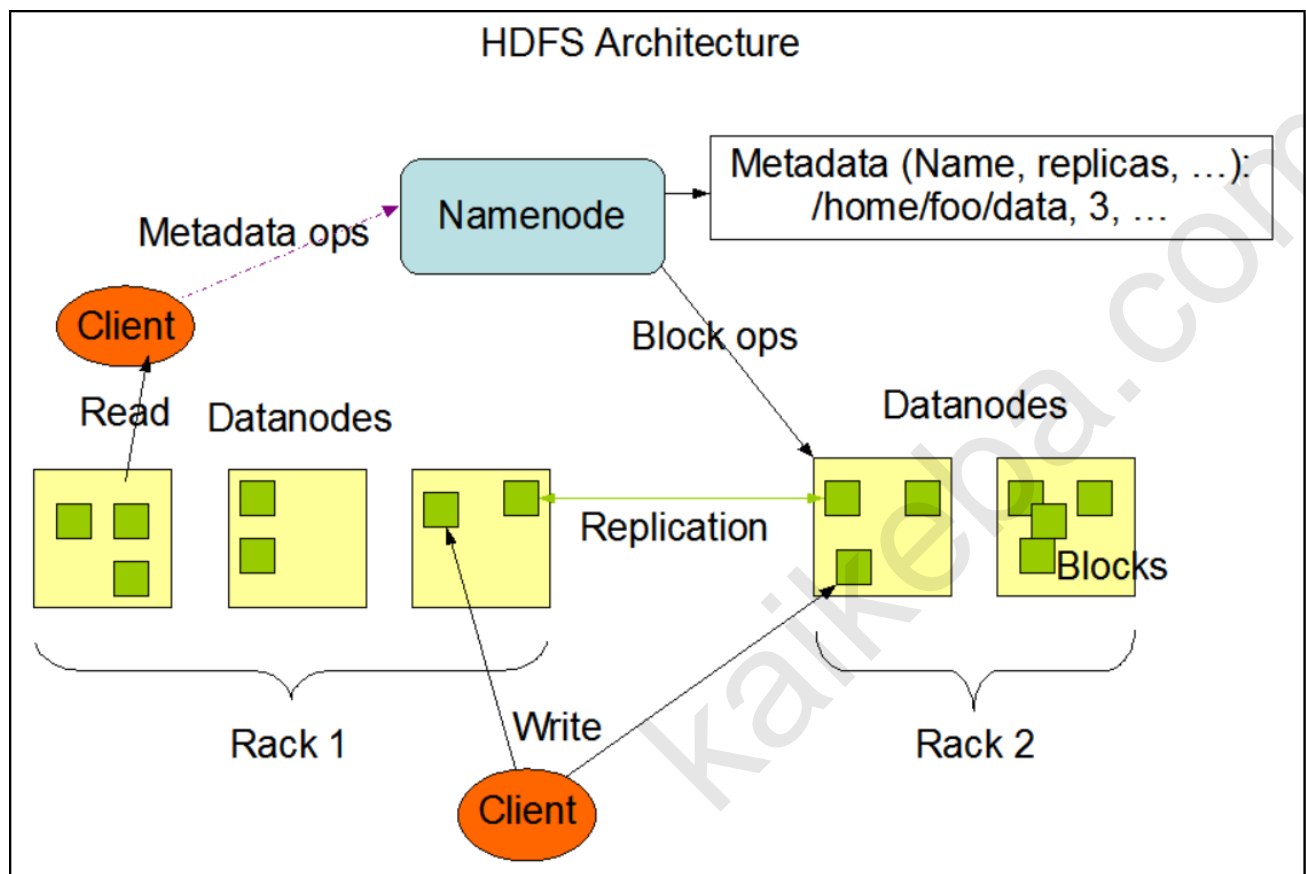
什么原因会有可能造成不平衡？

为什么需要均衡？

如何均衡？

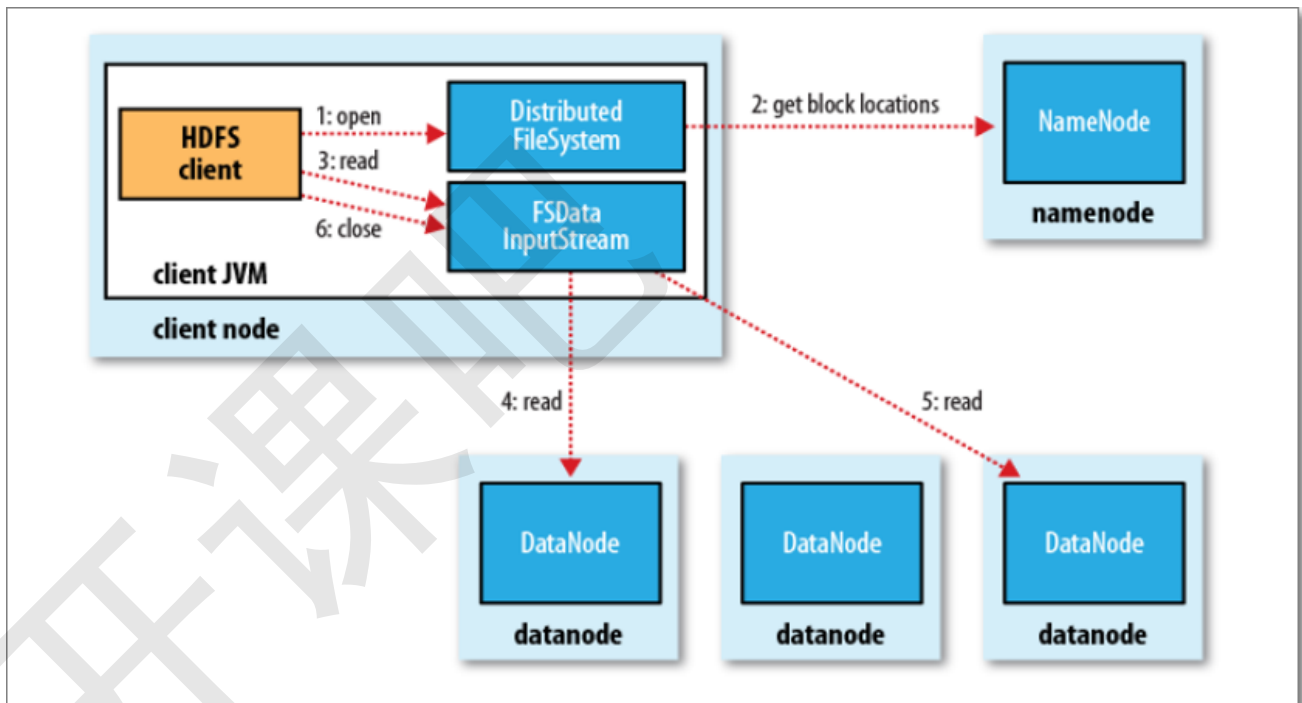
```
$HADOOP_HOME/sbin/start-balancer.sh -t 5%
```

7. HDFS读写流程（15分钟）



7.1 数据读流程

7.1.1 基本流程

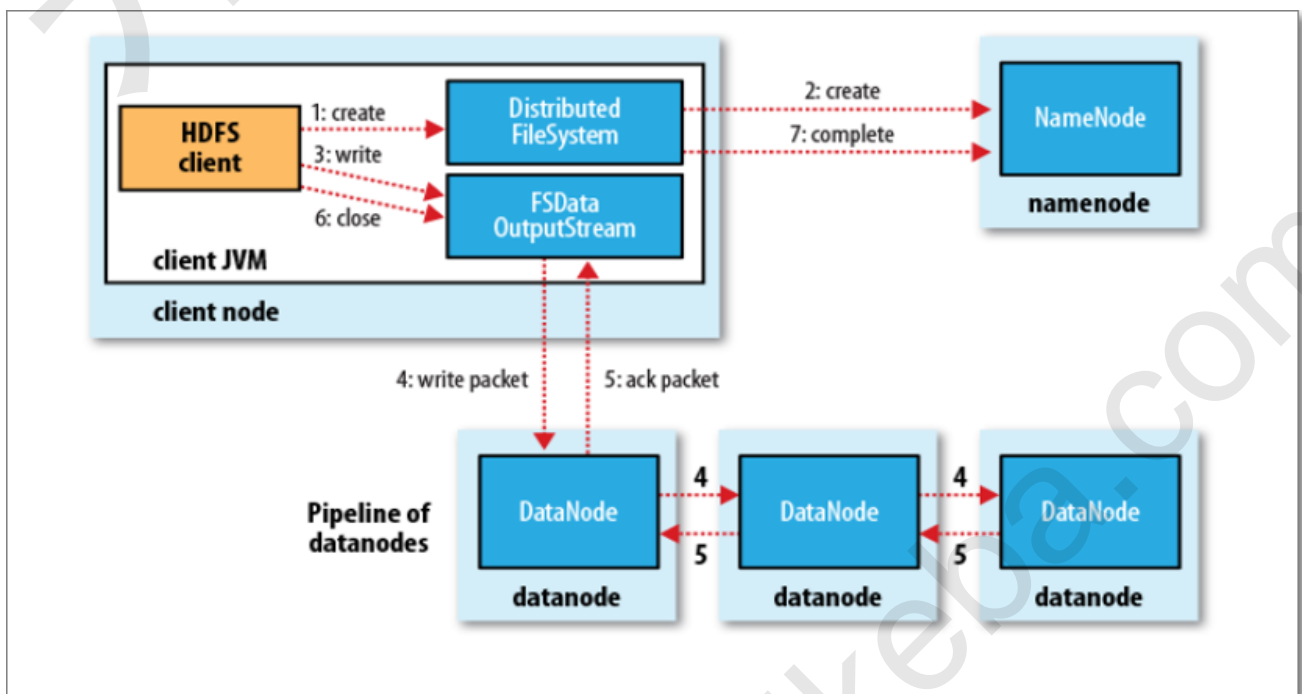
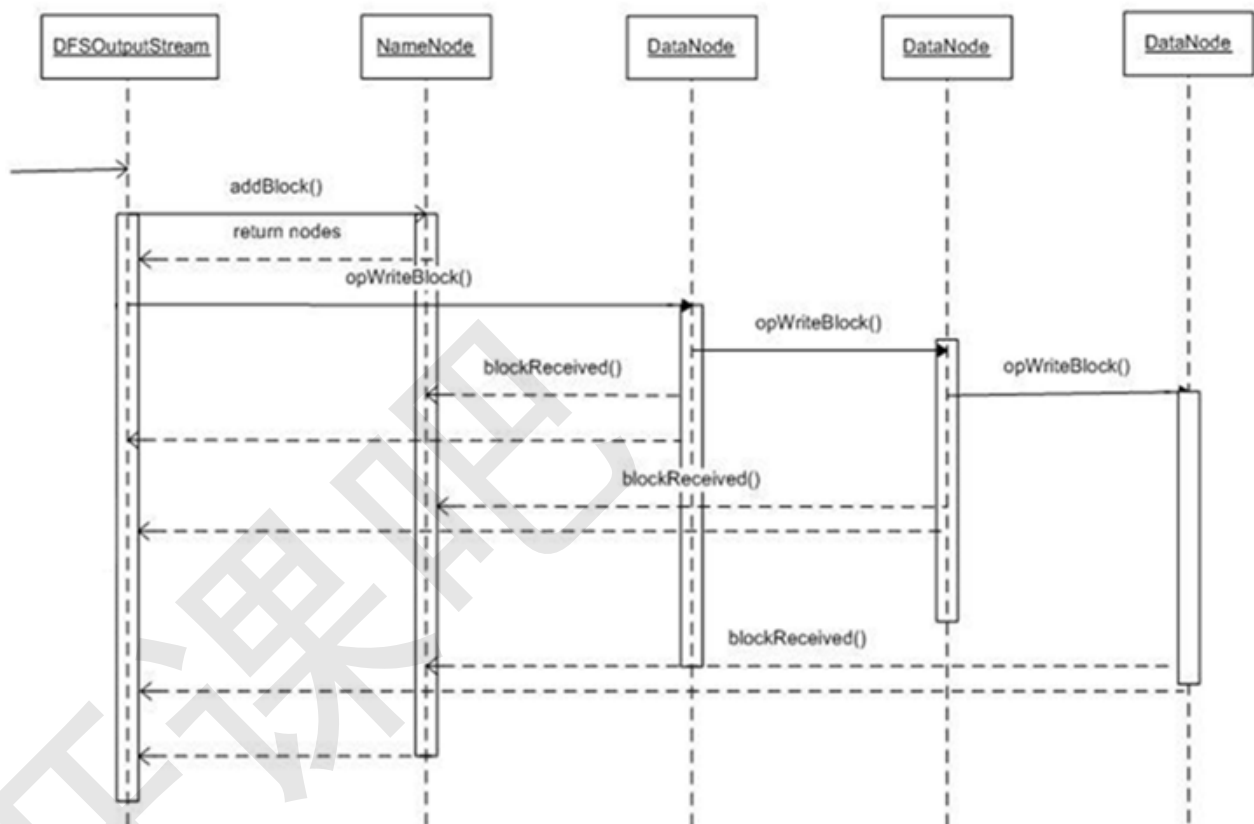


1. 客户端与NameNode通讯获取文件的块位置信息，其中包括了块的所有冗余备份的位置信息：DataNode的列表
2. 客户端获取文件位置信息后直接同有文件块的DataNode通讯，读取文件
3. 如果第一个DataNode无法连接，客户端将自动联系下一个DataNode
4. 如果块数据的校验值出错，则客户端需要向NameNode报告，并自动联系下一个DataNode

7.1.2 容错

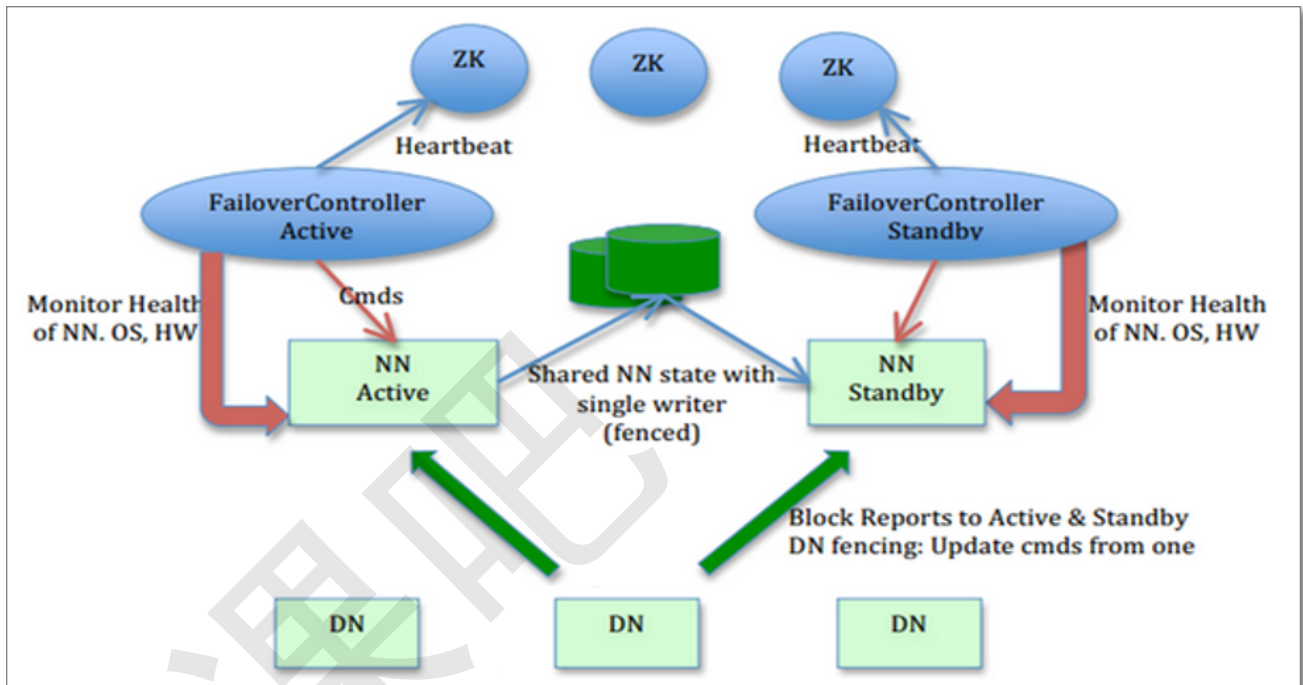
- 如果client从datanode读取block时，网络中断怎么办？
- client如何保证读取的数据是完整的？
- 如果client读取的数据不完整，怎么办？

7.2 数据写流程



7.3 源码剖析

8. Hadoop HA高可用

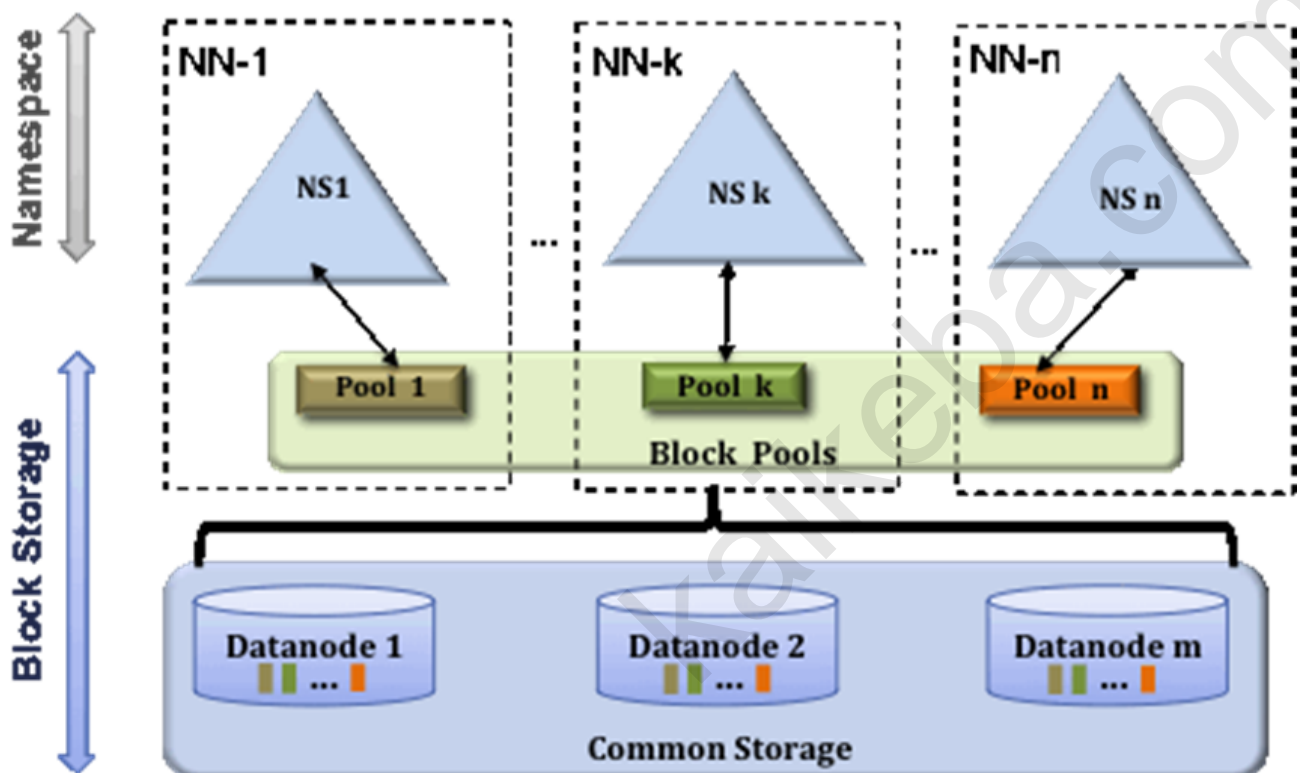


9. Hadoop联邦

9.1 为什么需要高可用与联邦

- 集群的元数据保存在namenode内存中
- 每个文件、目录、block占用约150字节
- 对于一个拥有大量文件的超大集群来说，内存将成为限制系统横向扩展的瓶颈。

9.2 联邦



10. HDFS编程

- 写数据

```
package com.kaikeba.hadoop.hdfs;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;

import java.io.*;
import java.net.URI;

/**
 * 将本地文件系统的文件通过java-API写入到HDFS文件
 */
public class FileCopyFromLocal {

    public static void main(String[] args){

        String source="/home/bruce/hdfs01.mp4"; //linux中的文件路径,demo存在一定数据

        //先确保/data目录存在
        String destination="hdfs://node-01:9000/data/hdfs01.mp4";//HDFS的路径

        InputStream in = null;
        try {
            in = new BufferedInputStream(new FileInputStream(source));
            //HDFS读写的配置文件
            Configuration conf = new Configuration();

            FileSystem fs = FileSystem.get(URI.create(destination),conf);

            //调用Filesystem的create方法返回的是FSDataOutputStream对象
            //该对象不允许在文件中定位,因为HDFS只允许一个已打开的文件顺序写入或追加
            OutputStream out = fs.create(new Path(destination));

            IOUtils.copyBytes(in, out, 4096, true);
        } catch (FileNotFoundException e) {
            System.out.println("exception");
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println("exception1");
            e.printStackTrace();
        }
    }
}
```

- 读数据

```
package com.kaikaba.hadoop.hdfs;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;

import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.net.URI;

/**
 * 从HDFS读取文件
 * 打包运行jar包 [bruce@node-01 Desktop]$ hadoop jar com.kaikaba.hadoop-1.0-SNAPSHOT.jar
com.kaikaba.hadoop.hdfs.FileReadFromHdfs
 */
public class FileReadFromHdfs {

    public static void main(String[] args) {
        try {
            //
            String srcFile = "hdfs://node-01:9000/data/hdfs01.mp4";
            Configuration conf = new Configuration();

            FileSystem fs = FileSystem.get(URI.create(srcFile), conf);
            FSDataInputStream hdfsInStream = fs.open(new Path(srcFile));

            BufferedOutputStream outputStream = new BufferedOutputStream(new
            FileOutputStream("/home/bruce/hdfs02.mp4"));

            IOUtils.copyBytes(hdfsInStream, outputStream, 4096, true);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

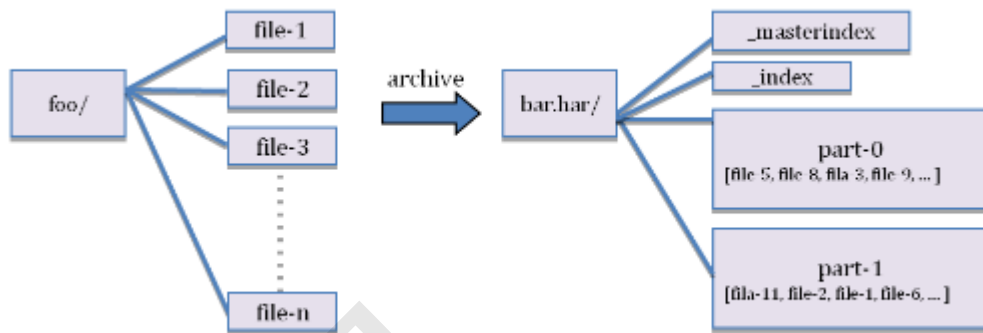
11. 存储大量小文件

11.1 有没有问题

- NameNode存储着文件系统的元数据，每个文件、目录、块大概有150字节的元数据；
- 因此文件数量的限制也由NN内存大小决定，如果小文件过多则会造成NN的压力过大

11.2 如何解决

1. HAR文件方案（本质启动mr程序，所以需要启动yarn）



```
# 创建archive文件
hadoop archive -archiveName test.har -p /testhar -r 3 th1 th2 /outhar # 原文件还存在，需手动删除
# 查看archive文件
hdfs dfs -ls -R har:///outhar/test.har
# 解压archive文件
hdfs dfs -cp har:///outhar/test.har/th1 hdfs:/unarchivef
hadoop fs -ls /unarchivef # 顺序
hadoop distcp har:///outhar/test.har/th1 hdfs:/unarchivef2 # 并行，启动MR
```

2. Sequence Files方案

其核心是以文件名为key，文件内容为value组织小文件。10000个100KB的小文件，可以编写程序将这些文件放到一个SequenceFile文件，然后就以数据流的方式处理这些文件，也可以使用MapReduce进行处理。一个SequenceFile是**可分割**的，所以MapReduce可将文件切分成块，每一块独立操作。不像HAR,SequenceFile**支持压缩**。在大多数情况下，以block为单位进行压缩是最好的选择，因为一个block包含多条记录，压缩作用在block之上，比reduce压缩方式（一条一条记录进行压缩）的压缩比高。把已有的数据转存为SequenceFile比较慢。比起先写小文件，再将小文件写入SequenceFile，一个更好的选择是直接将数据写入一个SequenceFile文件，省去小文件作为中间媒介。

- 向SequenceFile写入数据

```
import java.io.IOException;
import java.net.URI;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.Text;

public class SequenceFileWriteDemo {

    private static final String[] DATA = {

        "One, two, buckle my shoe",
```

```
"Three, four, shut the door",
"Five, six, pick up sticks",
"Seven, eight, lay them straight",
"Nine, ten, a big fat hen"
};

public static void main(String[] args) throws IOException {
    String uri = args[0];
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(URI.create(uri), conf);
    Path path = new Path(uri);

    IntWritable key = new IntWritable();
    Text value = new Text();
    SequenceFile.Writer writer = null;
    try {
        writer = SequenceFile.createWriter(fs, conf, path,
            key.getClass(), value.getClass());

        for (int i = 0; i < 100; i++) {
            key.set(100 - i);
            value.set(DATA[i % DATA.length]);
            System.out.printf("[%s]\t%s\t%s\n", writer.getLength(), key, value);
            writer.append(key, value);
        }
    } finally {
        IOUtils.closeStream(writer);
    }
}
```

- 读取SequenceFile文件

```
import java.io.IOException;
import java.net.URI;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.util.ReflectionUtils;

public class SequenceFileReadDemo {

    public static void main(String[] args) throws IOException {
        String uri = args[0];
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(URI.create(uri), conf);
        Path path = new Path(uri);
```

```
SequenceFile.Reader reader = null;
try {
    reader = new SequenceFile.Reader(fs, path, conf);
    Writable key = (Writable)
        ReflectionUtils.newInstance(reader.getKeyClass(), conf);
    Writable value = (Writable)
        ReflectionUtils.newInstance(reader.getValueClass(), conf);
    long position = reader.getPosition();
    while (reader.next(key, value)) {
        String syncSeen = reader.syncSeen() ? "*" : "";
        System.out.printf("[%s%s]\t%s\t%s\n", position, syncSeen, key, value);
        position = reader.getPosition(); // beginning of next record
    }
} finally {
    IOUtils.closeStream(reader);
}
}
```

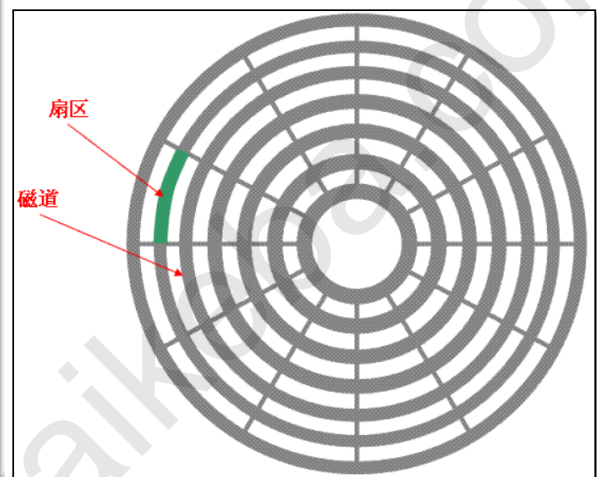
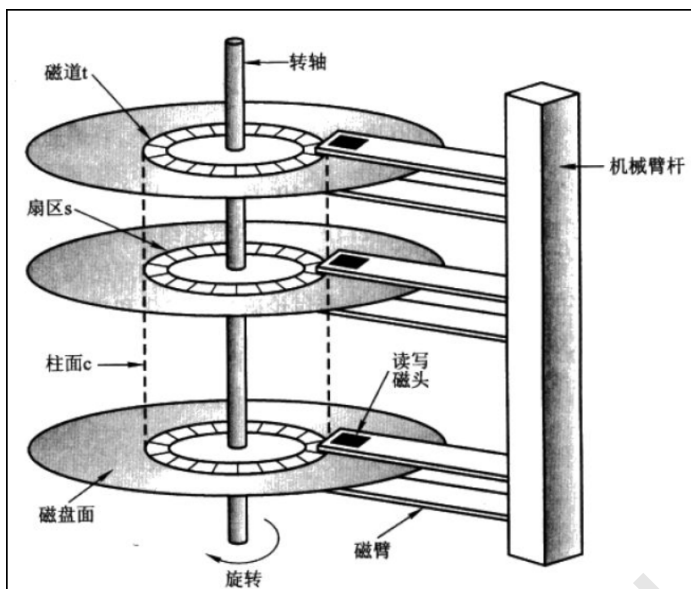
五、拓展点、未来计划、行业趋势（5分钟）

1. HDFS存储地位

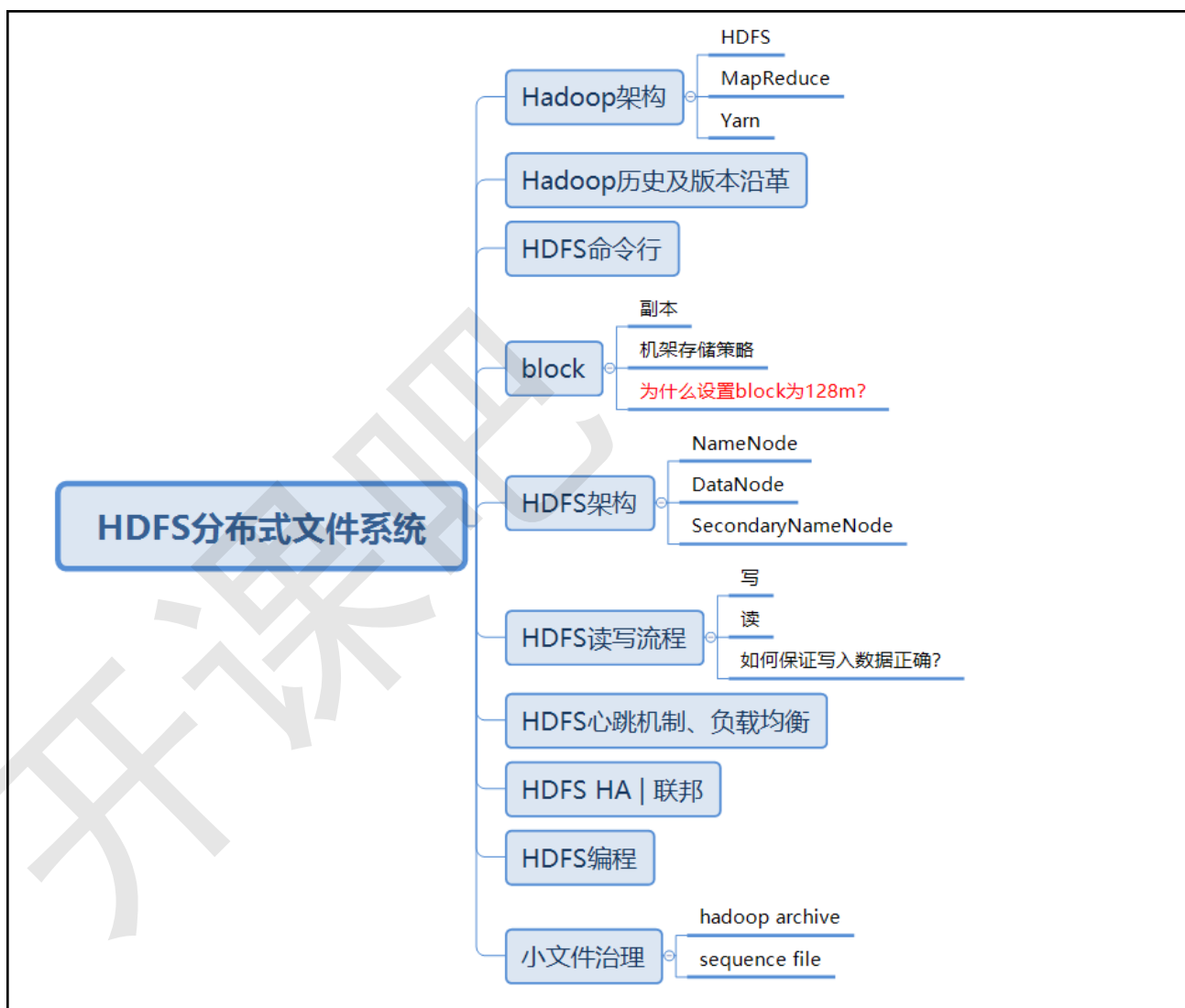
hdfs ->

2. block块为什么设置的比较大(面试)

- 磁盘块
- 为了最小化寻址开销；从磁盘传输数据的时间明显大于定位这个块开始位置所需的时间
- 问：块的大小是不是设置的越大越好呢？



六、总结（5分钟）



七、作业

- 第一堂课
 - Hadoop fs命令中“1. HDFS基本操作”
 - 描述向HDFS上传文件的流程
 - 描述从HDFS下载文件的流程
- 第二堂课
 - 利用java编程，向HDFS上传文件
 - 利用java编程，从HDFS下载文件

八、互动问答

九、题库 - 本堂课知识点

1. 搭建5节点的hadoop HA集群
2. Hadoop fs命令中“1. HDFS基本操作”

3. 利用java编程，向HDFS上传文件
4. 利用java编程，从HDFS下载文件
5. 描述向HDFS上传文件的流程
6. 描述从HDFS下载文件的流程