第二章　Hadoop文件系统

徐 辰
cxu@dase.ecnu.edu.cn

華東師範大學

Hadoop发展简史

- Hadoop源于Lucene（Nutch）
  - Apache Lucene项目：Doug Cutting开发的文本搜索库
  - 2002年，Apache Nutch开源
    - 一个网络搜索引擎
    - Lucene项目的一部分

Hadoop发展简史

- 借鉴谷歌论文中的思想
  - 2003年，谷歌发表GFS论文
  - 2004年，Nutch项目也模仿GFS开发了自己的分布式文件系统NDFS（Nutch Distributed File System），也就是HDFS的前身
  - 2004年，谷歌公司发表MapReduce论文
  - 2005年，Nutch开源实现了谷歌的MapReduce

Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google File System. In SOSP (pp. 29–43).

Dean, J., & Ghemawat, S. (2004). MapReduce : Simplified Data Processing on Large Clusters. In OSDI (pp. 137–149).

Hadoop发展简史

- Hadoop的诞生
  - 2006年2月，Nutch中的NDFS和MapReduce开始独立出来，成为Lucene项目的一个子项目，称为Hadoop。同时，Doug Cutting加盟雅虎
  - 2008年1月，Hadoop正式成为Apache顶级项目，Hadoop也逐渐开始被其他公司使用
  - 2008年4月，Hadoop采用一个由910个节点构成的集群排序1TB数据只用了209秒
  - 在2009年5月，Hadoop把1TB数据排序时间缩短到62秒

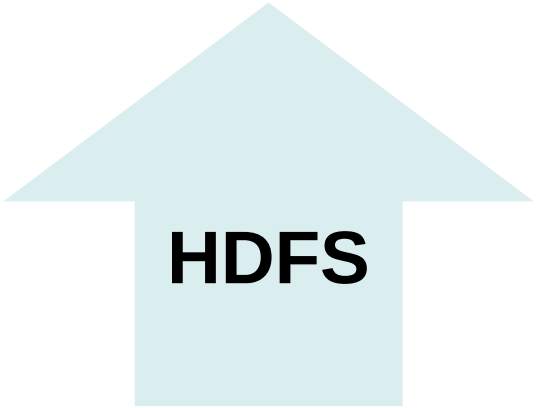
Hadoop与MapReduce

开源项目		学术论文
Hadoop	Hadoop Distributed File System (HDFS)	Google File System (GFS)
	Hadoop MapReduce	Google MapReduce

Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google File System. In SOSP (pp. 29–43).

Dean, J., & Ghemawat, S. (2004). MapReduce : Simplified Data Processing on Large Clusters. In OSDI (pp. 137–149).

Hadoop核心项目



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

MapReduce：分布式计算框架

大纲

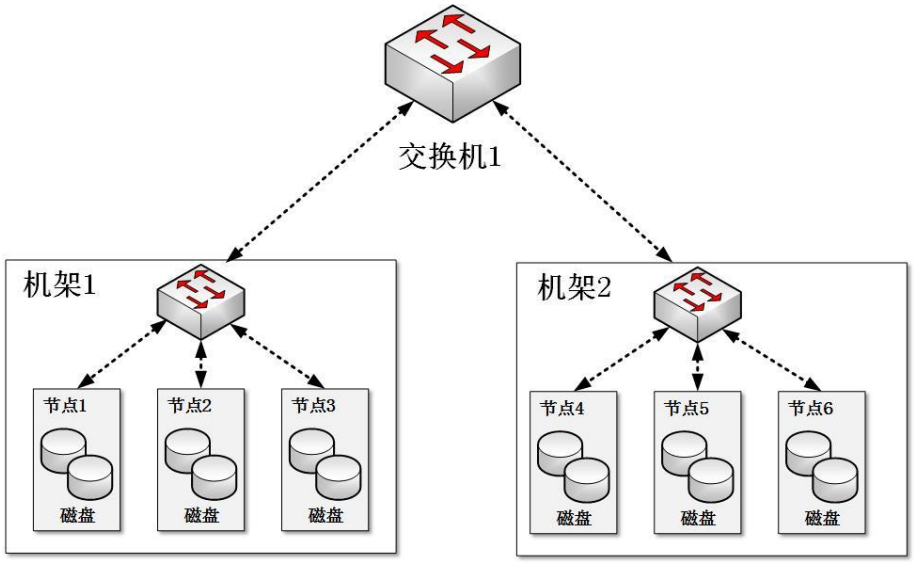
- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 编程示例

需要解决的问题

- 如何存储上百GB/TB级别大文件?
  - 例如，某一主题的网页构成的数据集存成一个大文件
- 如何保证文件系统的容错?
  - 集群由低廉的普通服务器甚至个人PC组成，节点发生故障是普遍的现象

集群网络拓扑

- 机架（Rack）
- 节点



需要解决的问题

- 如何存储上百GB/TB级别大文件?
  - 例如，某一主题的网页构成的数据集存成一个大文件
- 如何保证文件系统的容错?
  - 集群由低廉的普通服务器甚至个人PC组成，节点发生故障是普遍的现象
- 如何进行大文件的并发读写控制?
  - 文件的并发读写往往需要加锁等一系列复杂的措施来避免读写冲突

文件分块存储

Example:

大文件

6440MB

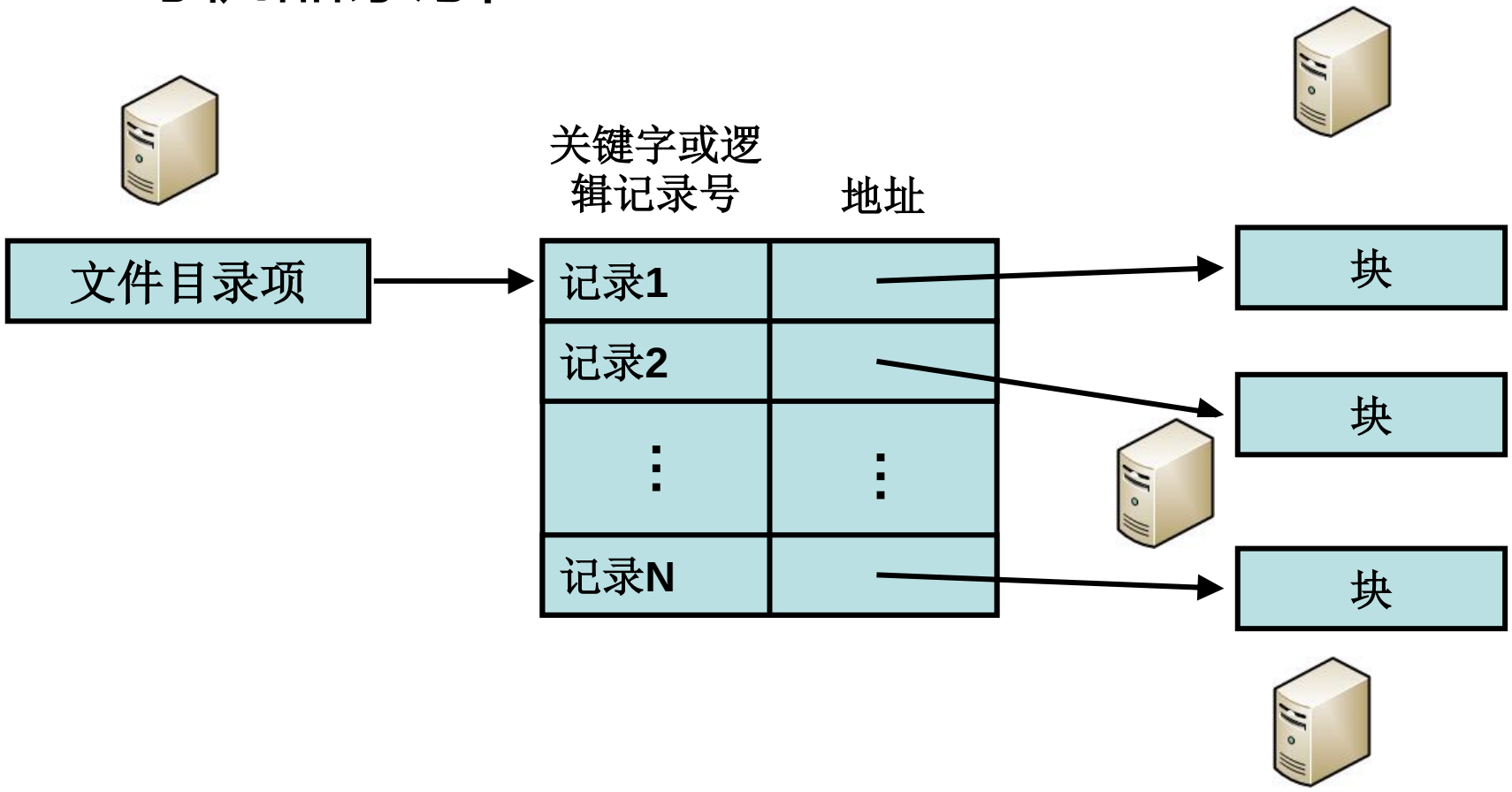
e.g., block size = 64MB

文件由数据块集合组成

- 通常每块大小为 64MB
- 每个数据块在本地文件系统中是以单独的文件进行存储(e.g. NTFS)

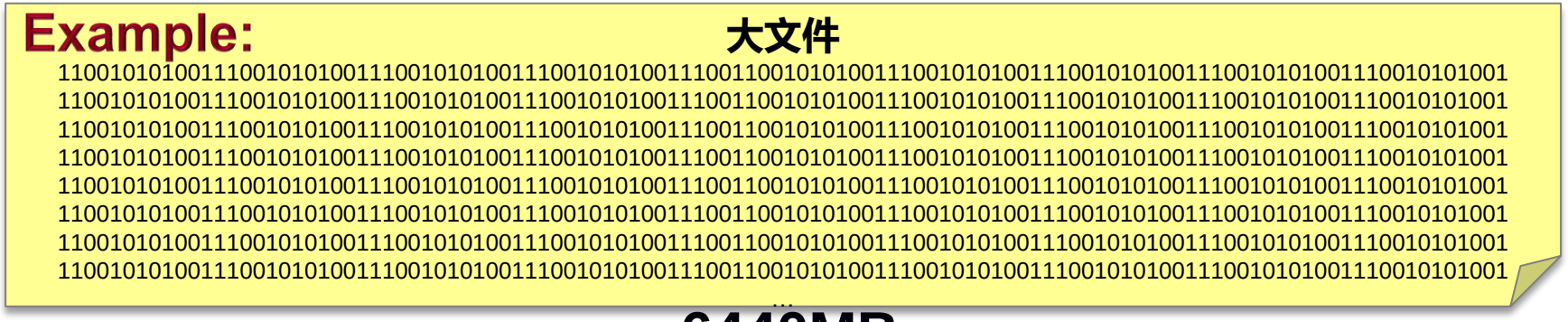
回顾：索引文件

- 跨机器索引



分块冗余存储

Example: **大文件**



6440MB

Block 1	Block 2	Block 3	Block 4	Block 5	Block 6	…	Block 100	Block 101
64MB	64MB	64MB	64MB	64MB	64MB		64MB	40MB

Block 202	Block 203	Block 204	Block 205	Block 206	Block 207	…	Block 301	Block 302
64MB	64MB	64MB	64MB	64MB	64MB		64MB	40MB

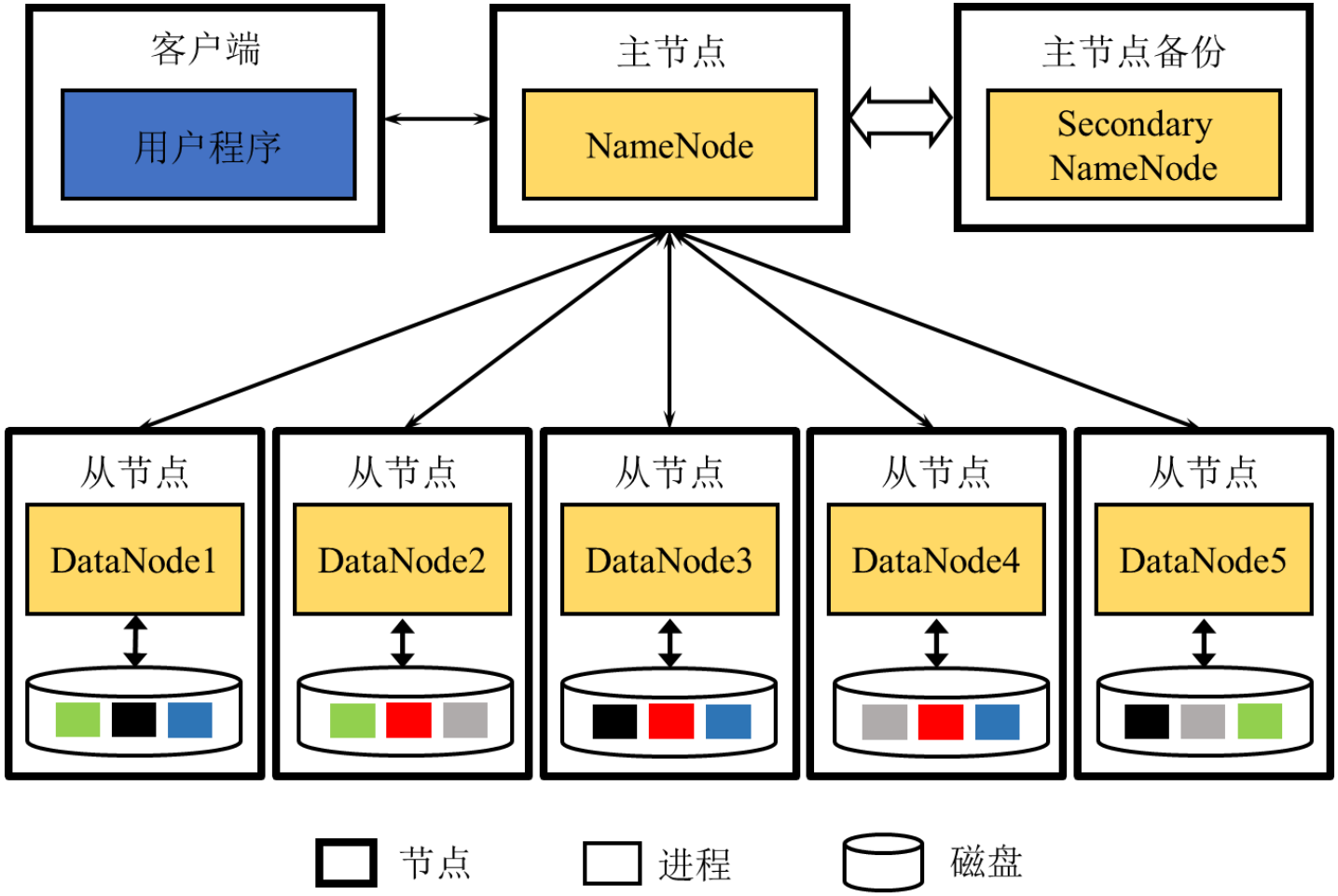
简化文件读写

- 避免读写冲突
  - 文件一次写入后不再修改，而仅允许多次读取
- 避免随机写
  - 仅支持顺序写入，而不允许随机写入

大纲

- 设计思想
- 体系架构
  - 架构图
  - 应用程序执行流程
- 工作原理
- 容错机制
- 编程示例

HDFS架构图

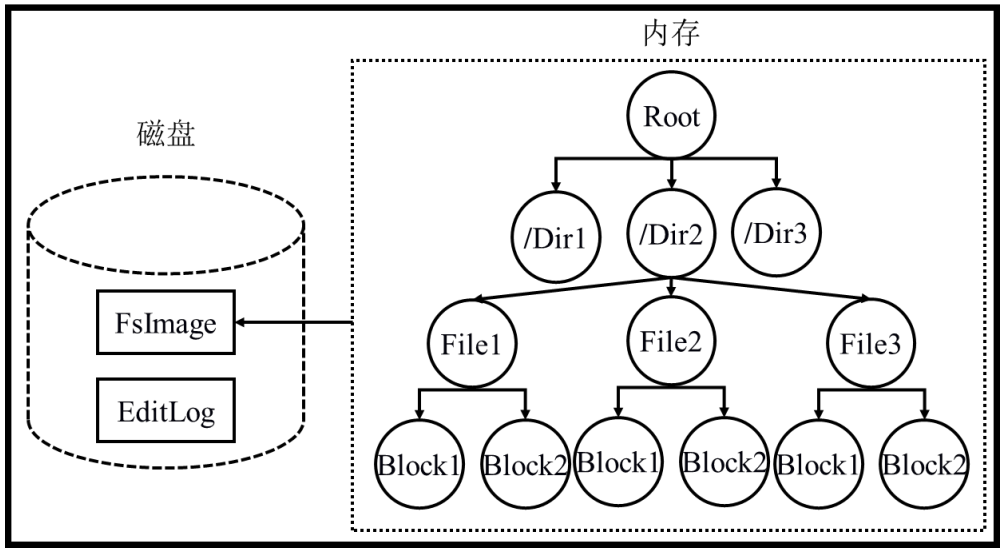

客户端
用户程序
主节点
NameNode
主节点备份
Secondary NameNode
从节点
DataNode1
从节点
DataNode2
从节点
DataNode3
从节点
DataNode4
从节点
DataNode5
节点
进程
磁盘

HDFS角色

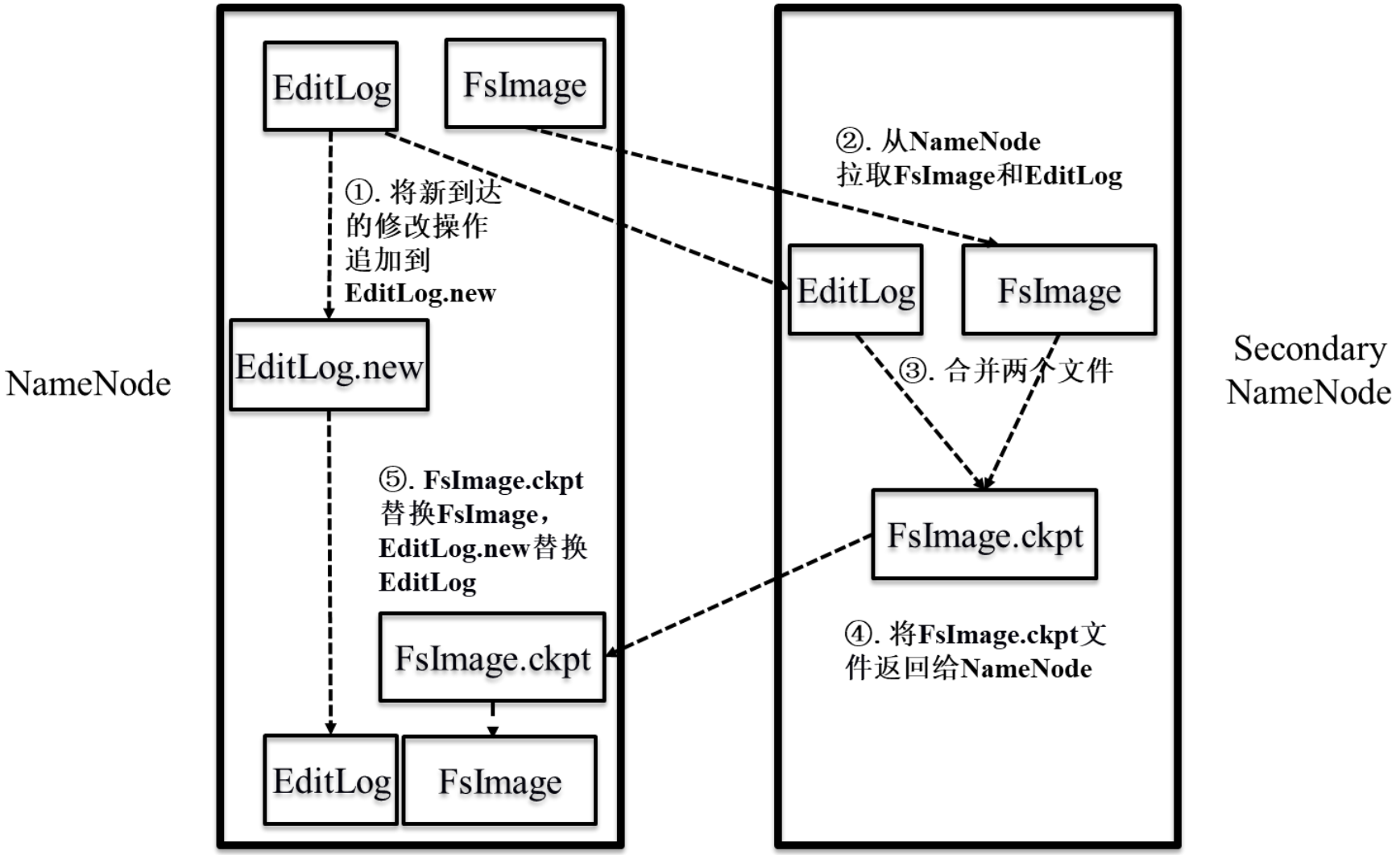
- **NameNode**
  - 负责HDFS的管理工作，包括管理文件目录结构、位置等元数据、维护 DataNode的状态等
  - 并不实际存储文件
- **SecondaryNameNode** – NameNode的备份
  - 充当NameNode的备份
  - 一旦NameNode发生故障时利用Secondary NameNode进行恢复
- **DataNode**
  - 负责数据块的存储
  - 为客户端提供实际的文件数据

NameNode

- FsImage：内存中文件目录结构及其元信息在磁盘上的快照
- EditLog：针对目录及文件修改的操作



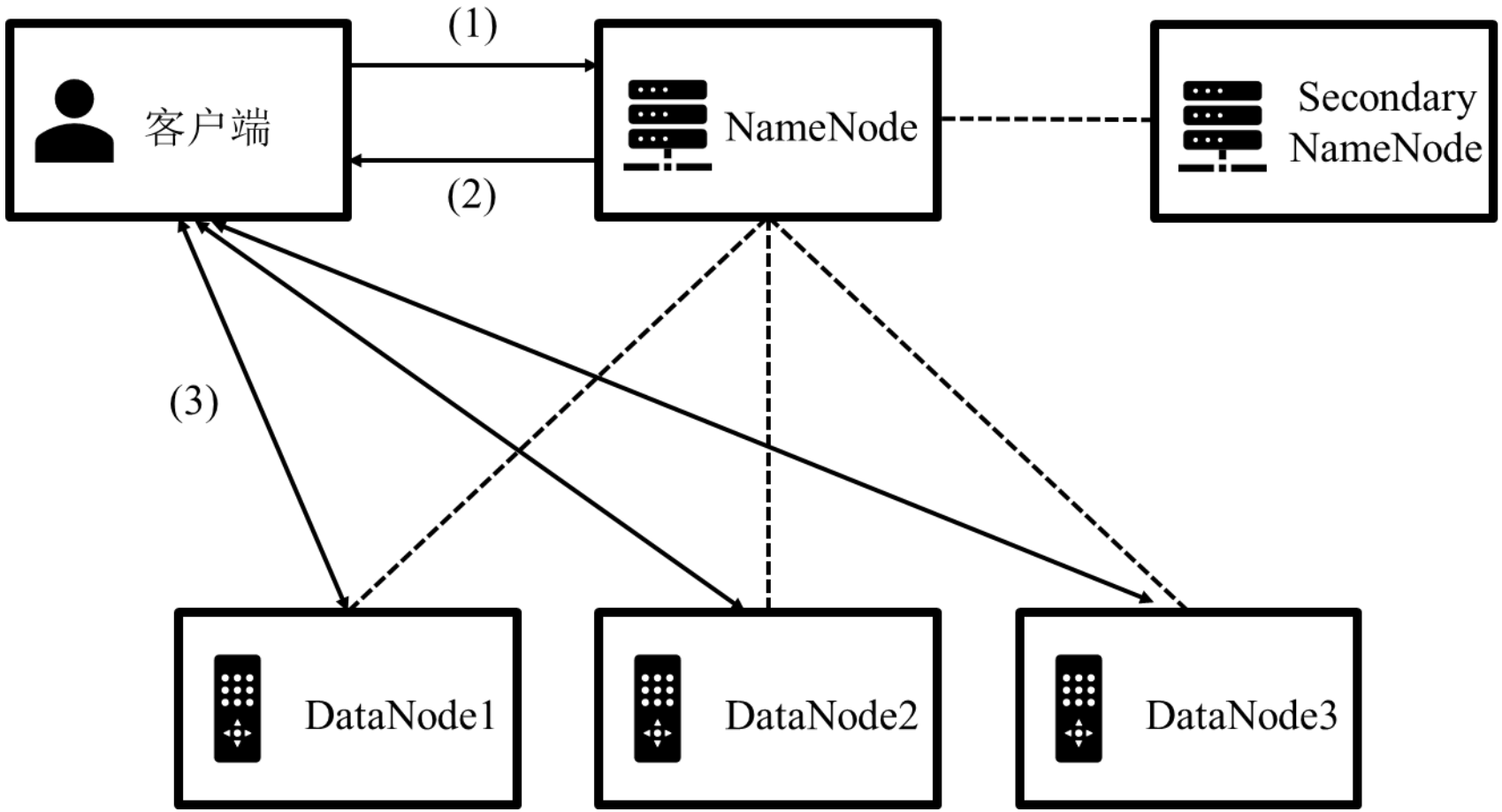
NameNode与SecondaryNameNode



大纲

- 设计思想
- 体系架构
  - 架构图
  - 应用程序执行流程
- 工作原理
- 容错机制
- 编程示例

应用程序执行流程


(1)
客户端
NameNode
Secondary
NameNode
(2)
(3)
DataNode1
DataNode2
DataNode3

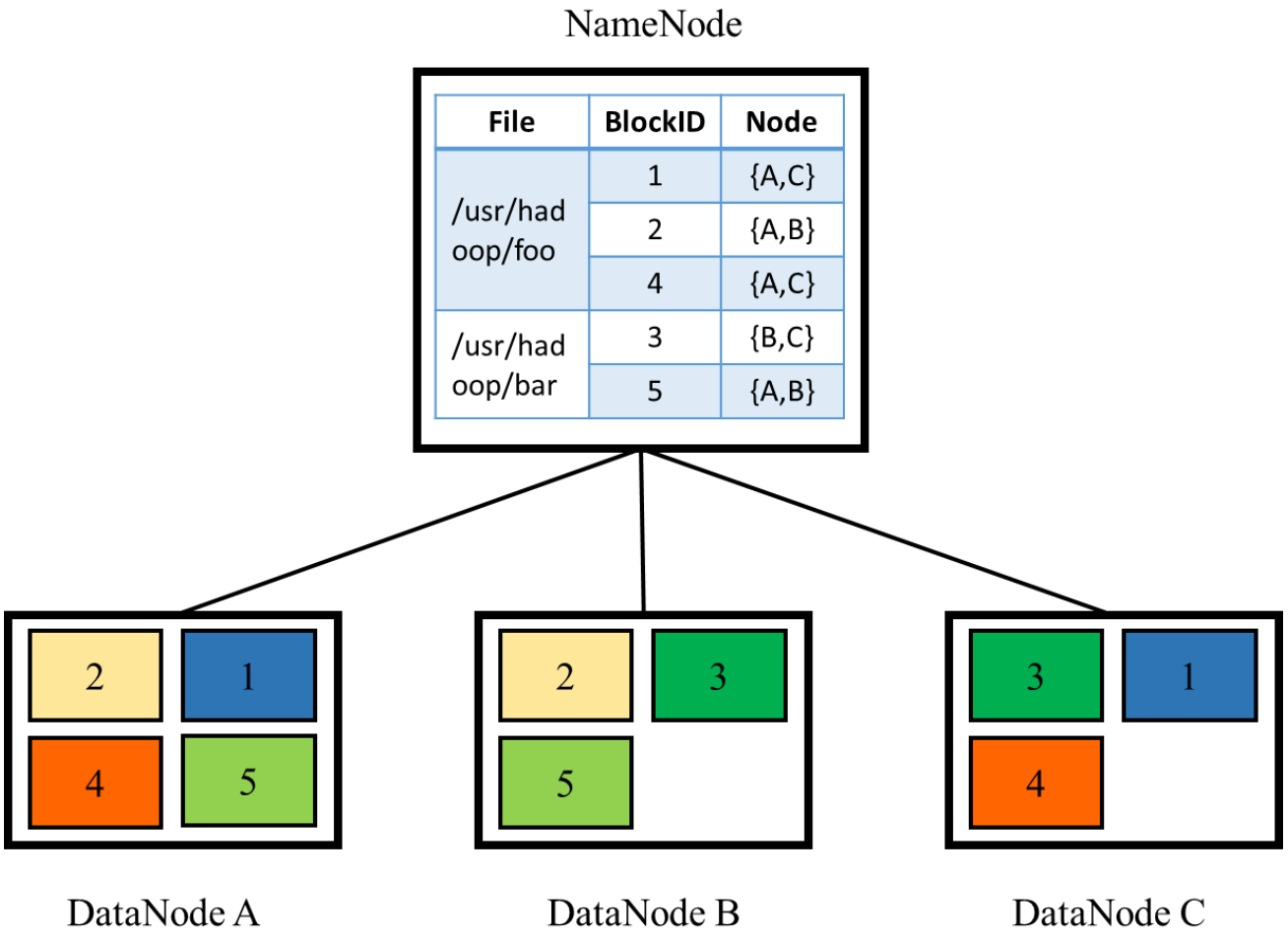
应用程序执行流程

1. 客户端向NameNode发起文件操作请求
2. NameNode反馈
   - 如果是读写文件操作，则NameNode告知客户端文件块存储的位置信息。
   - 如果是创建、删除、重命名目录或文件等操作，NameNode修改文件目录结构成功后结束。
   - 对于删除操作，HDFS并不会立即去删除DataNode上的数据块，而是等到特定时间才会真正删除。
3. 对于读写文件操作，客户端获知具体位置信息后再与DataNode进行读写交互

大纲

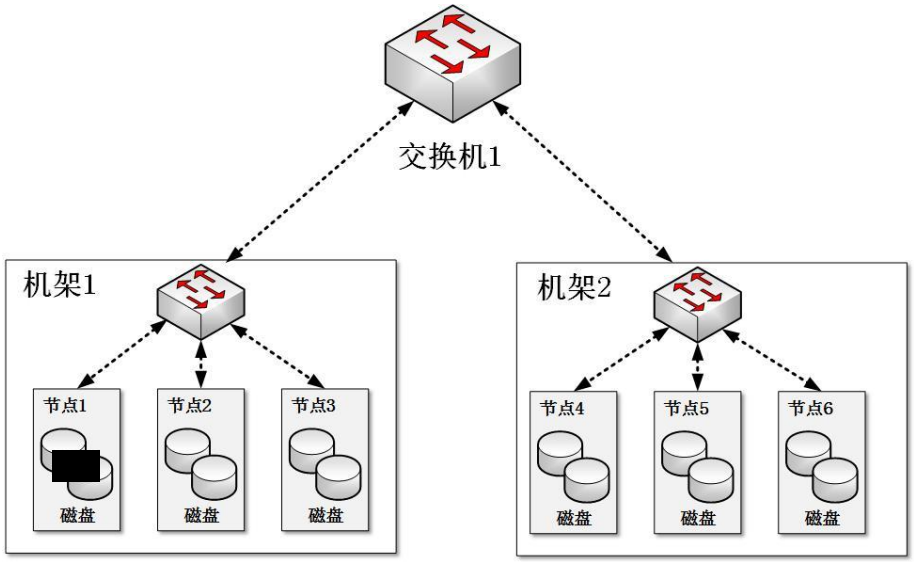
- 设计思想
- 体系架构
- 工作原理
  - 文件分块与备份
  - 文件写入
  - 文件读取
  - 文件读写与一致性
- 容错机制
- 编程示例

文件分块与备份


NameNode
File
BlockID
Node
/usr/hadoop/foo
1
{A,C}
2
{A,B}
4
{A,C}
/usr/hadoop/bar
3
{B,C}
5
{A,B}
2
1
4
5
2
3
5
3
1
4
DataNode A
DataNode B
DataNode C

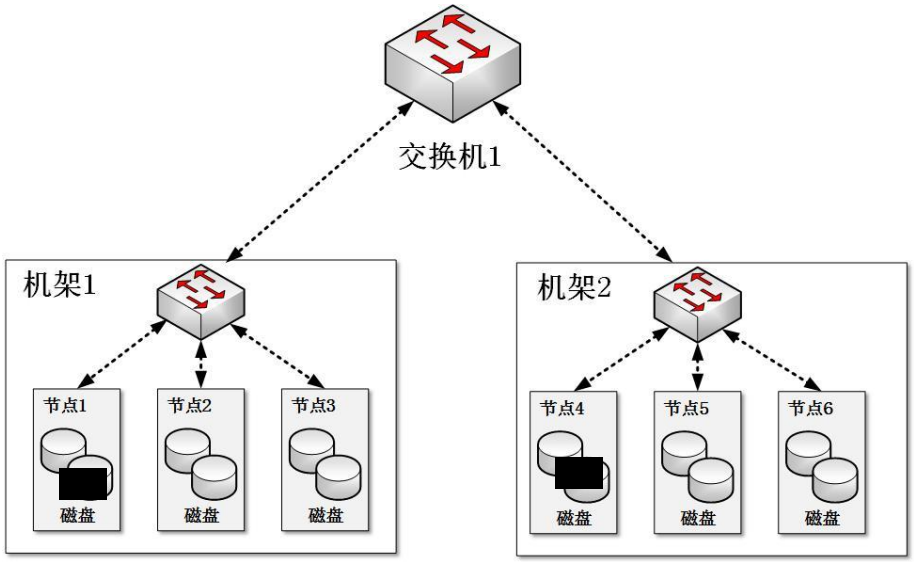
文件块存放策略（启发式）

- 第一个副本：
  - 若客户端和某一DataNode位于同一物理节点，那么HDFS将第一个副本放置在该DataNode
  - 如果客户端不与任何的DataNode在同一物理节点，那么HDFS随机挑选一台磁盘不太满、CPU不太忙的节点
  - 为了支持快速写入



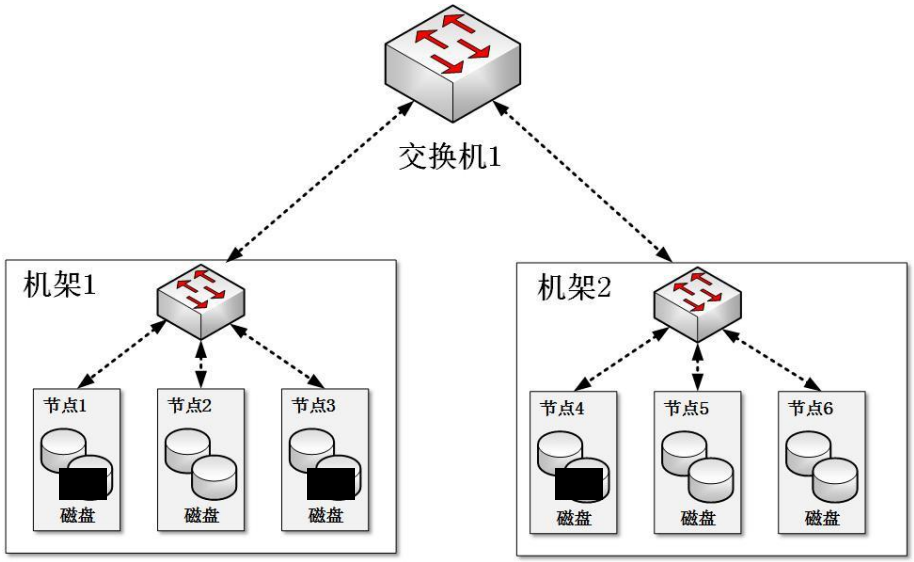
文件块存放策略（启发式）

- 第二个副本：
  - NameNode将第二个副本放置在与第一个副本不同的机架的某一节点上
  - 有利于整体上减少跨机架的网络流量



文件块存放策略（启发式）

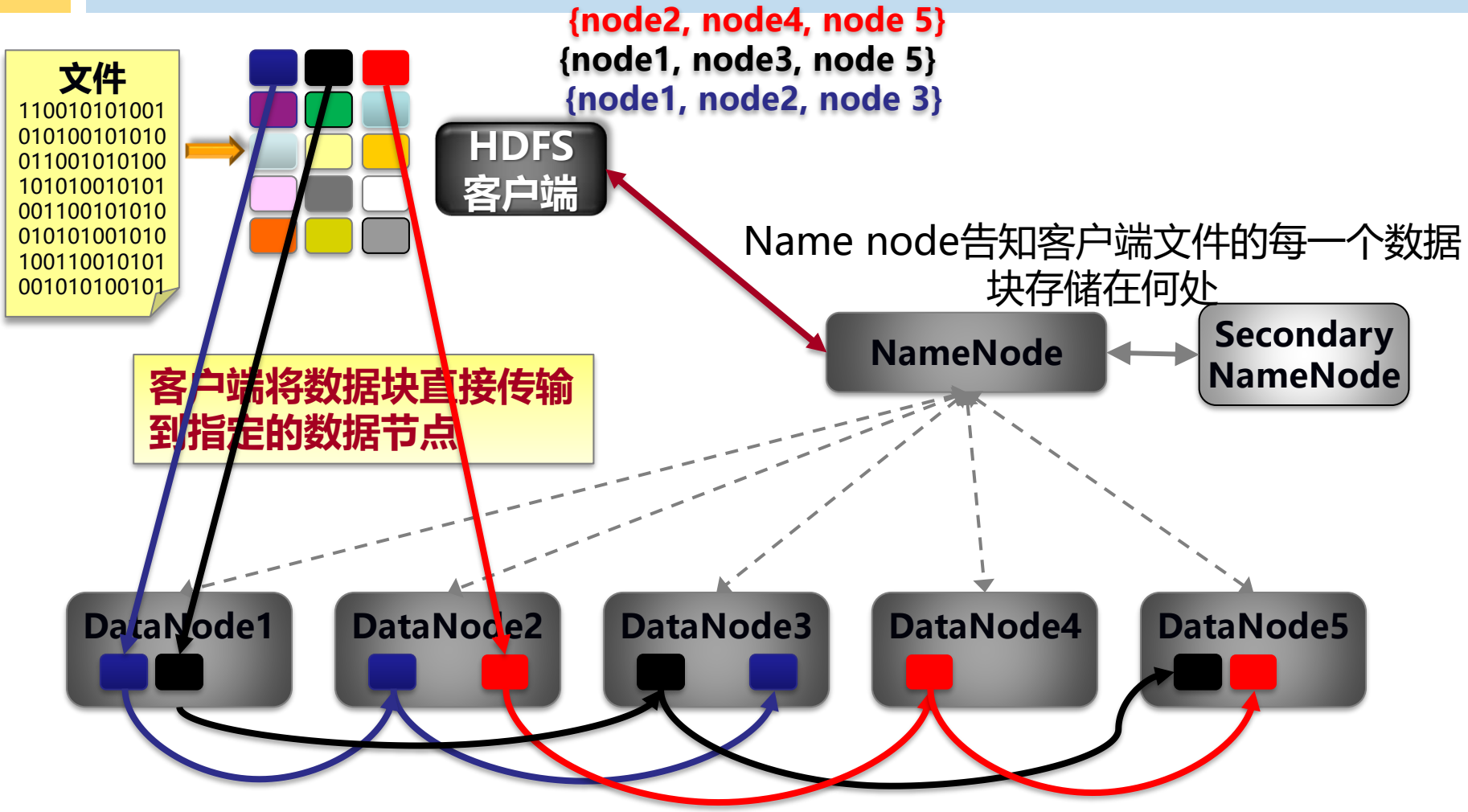
- 第三个副本：
  - NameNode将第三个副本放置在第一个副本所在机架的不同节点上
  - 应对第一个副本所在节点宕机且交换机故障



大纲

- 设计思想
- 体系架构
- 工作原理
  - 文件分块与备份
  - 文件写入
  - 文件读取
  - 文件读写与一致性
- 容错机制
- 编程示例

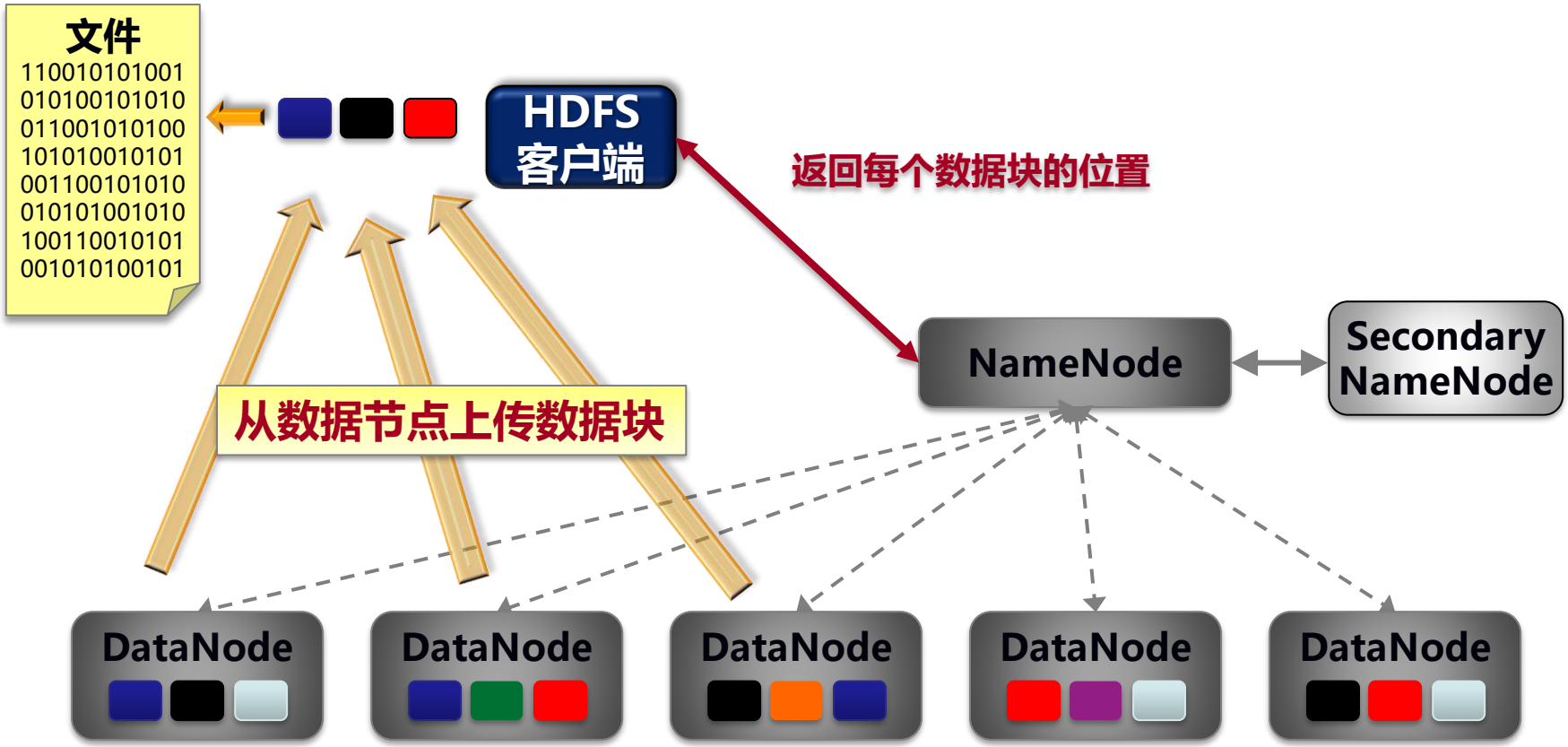
文件写入HDFS


{node2, node4, node 5}
{node1, node3, node 5}
{node1, node2, node 3}
文件
110010101001
010100101010
011001010100
101010010101
001100101010
010101001010
100110010101
001010100101
HDFS
客户端
Name node告知客户端文件的每一个数据
块存储在何处
NameNode
Secondary
NameNode
客户端将数据块直接传输
到指定的数据节点
DataNode1
DataNode2
DataNode3
DataNode4
DataNode5

大纲

- 设计思想
- 体系架构
- 工作原理
  - 文件分块与备份
  - 文件写入
  - 文件读取
  - 文件读写与一致性
- 容错机制
- 编程示例

从HDFS读取文件


文件
110010101001
010100101010
011001010100
101010010101
001100101010
010101001010
100110010101
001010100101
HDFS
客户端
返回每个数据块的位置
NameNode
Secondary
NameNode
从数据节点上传数据块
DataNode
DataNode
DataNode
DataNode
DataNode

大纲

- 设计思想
- 体系架构
- 工作原理
  - 文件分块与备份
  - 文件写入
  - 文件读取
  - 文件读写与一致性
- 容错机制
- 编程示例

文件读写与一致性

- “一次写入多次读取”
  - 一个文件经过创建、写入和关闭后就不得改变文件中的内容
  - 已经写入到HDFS文件，仅允许在文件末尾追加数据，即append操作
  - 当对一个文件进行写入操作或者追加操作，NameNode将拒绝其它针对该文件的读、写请求
  - 当对一个文件进行读取操作时，NameNode允许其它针对该文件的读请求。

简化的一致性模型

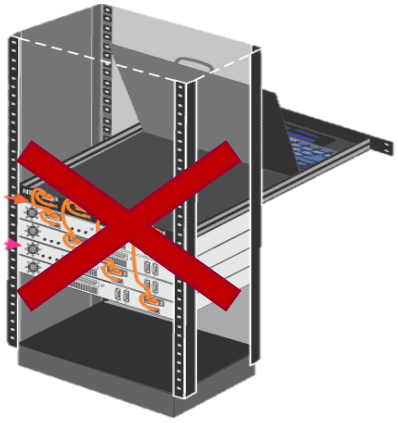
- 简化的好处
  - 避免读写冲突、用户编程无需考虑文件锁
- 问题
  - 假如用户的确需要修改已有文件中的内容，怎么办？
  - 如果HDFS允许修改文件中的已有内容，会带来哪些问题？

大纲

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 编程示例

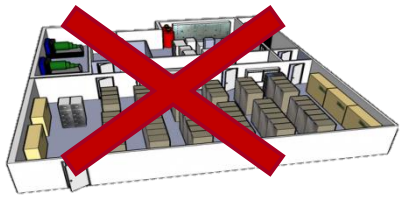
容错机制

故障是常见现象



故障类型:

- 磁盘错误和故障
- DataNode故障
- 交换机/机架故障
- NameNode故障
- 数据中心故障

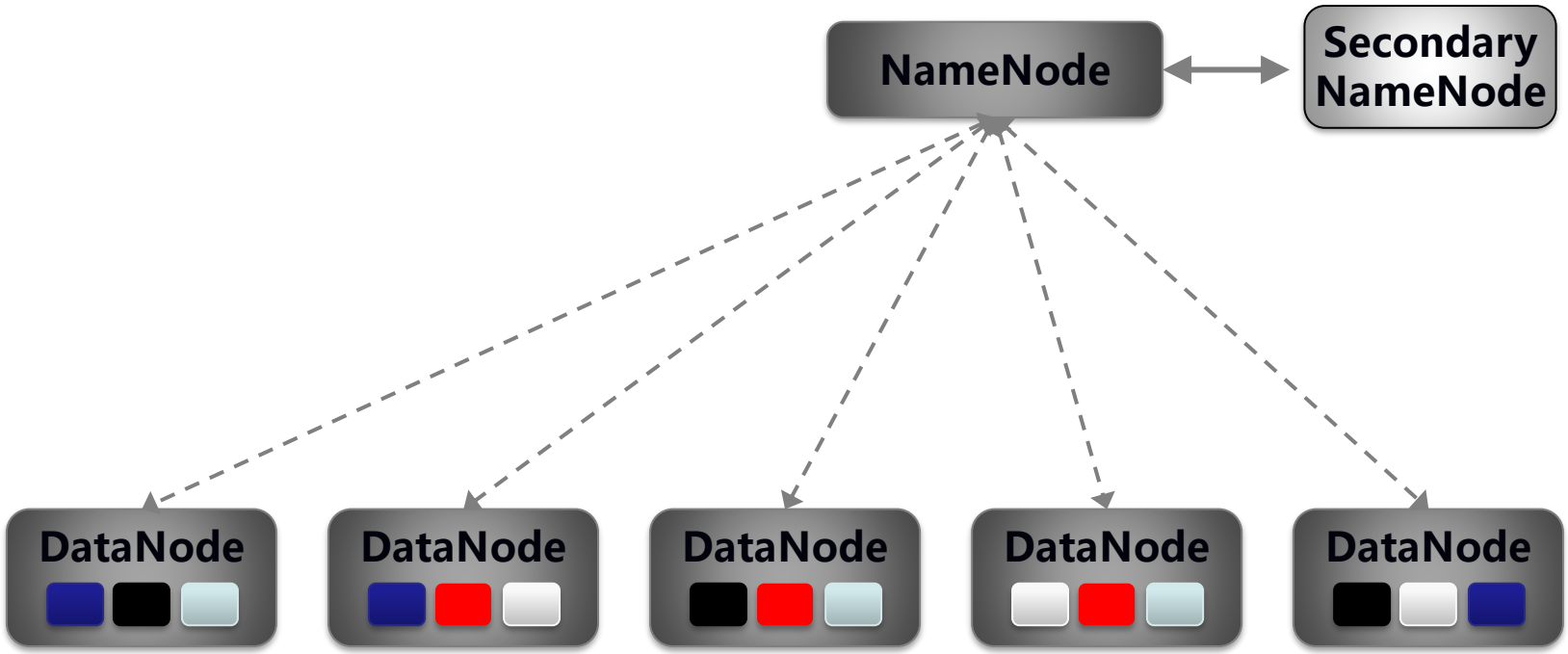


大纲

- 设计思想
- 体系架构
- 工作原理
- 容错机制
  - NameNode故障
  - DataNode故障
- 编程示例

NameNode故障

- 根据SecondaryNameNode中的FsImage和Editlog数据进行恢复

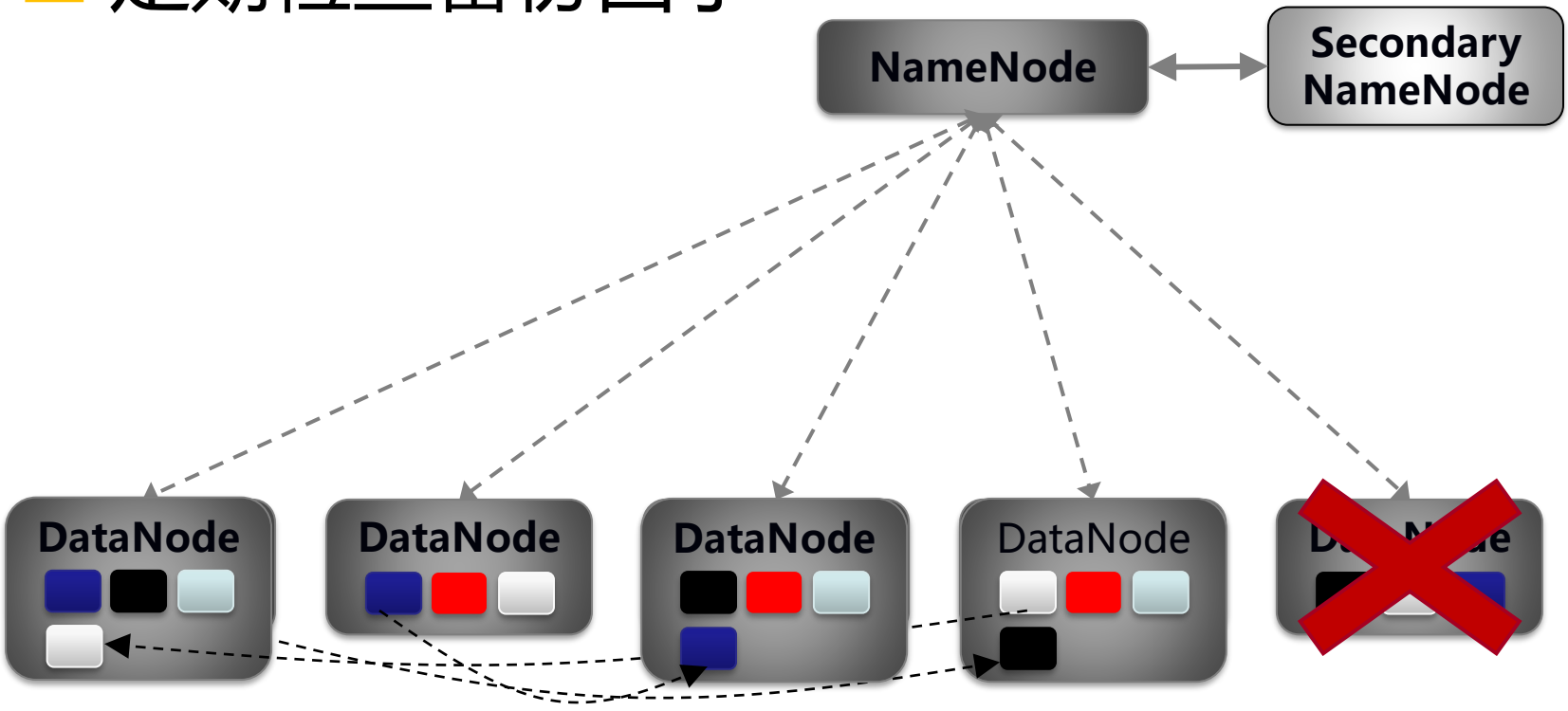


大纲

- 设计思想
- 体系架构
- 工作原理
- 容错机制
  - NameNode故障
  - DataNode故障
- 编程示例

DataNode故障

- “宕机”，节点上面的所有数据都会被标记为“不可读”
- 定期检查备份因子



大纲

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 编程示例

HDFS Shell

- 新建目录
  - ./bin/hdfs dfs –mkdir input
- 上传文件
  - ./bin/hdfs dfs -put ./README.txt ./input
- 查看文件
  - ./bin/hdfs dfs –cat ./input/README.txt
  - ./bin/hdfs dfs -ls /
- 拷贝
  - ./bin/hdfs dfs -cp ./input/README.txt /

Java程序框架


```
1  import org.apache.hadoop.conf.Configuration;
2  import org.apache.hadoop.fs.*;
3  ......
4
5  public class CustomProgram {
6
7    public void customOperation(String hdfsFilePath, ...) {
8      /* 步骤1：获取HDFS的文件系统对象 */
9      Configuration conf = new Configuration();
10     FileSystem fs = FileSystem.get(URI.create(hdfsFilePath), conf);
11     /* 步骤2：获取输入流hdfsInputStream或者输出流hdfsOutputStream */
12     FSDataInputStream hdfsInputStream = fs.open(new Path(hdfsFilePath));
13     FSDataOutputStream hdfsOutputStream = fs.create(new Path(hdfsFilePath));
14     /* 步骤3：利用输入或输出流操作HDFS文件 */
15     ......
16   }
17 }
```

大纲

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 编程示例
  - 写文件
  - 读文件

将文件写入HDFS


```
1  import org.apache.hadoop.conf.Configuration;
2  import org.apache.hadoop.fs.*;
3  import org.apache.hadoop.io.IOUtils;
4  ......
5
6  public class Writer {
7
8    public void write(String hdfsFilePath, String localFilePath) throws IOException {
9      /* 步骤1：获取HDFS的文件系统对象 */
10     Configuration conf = new Configuration();
11     FileSystem fs = FileSystem.get(URI.create(hdfsFilePath), conf);
12     /* 步骤2：获取输出流hdfsOutputStream */
13     FSDataOutputStream hdfsOutputStream = fs.create(new Path(hdfsFilePath));
14     /* 步骤3：利用输出流写入HDFS文件 */
15     // 读取本地文件的输入流
16     FileInputStream localInputStream = new FileInputStream(localFilePath);
17     // 将本地文件的输入流拷贝至HDFS文件的输出流
18     IOUtils.copyBytes(localInputStream, hdfsOutputStream, 4096, true);
19   }
20   ......
21 }
```

大纲

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 编程示例
  - 写文件
  - 读文件

从HDFS读文件


```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.io.IOUtils;
......

public class Reader {

  public void read(String hdfsFilePath, String localFilePath) throws IOException {
    /* 步骤1：获取HDFS的文件系统对象 */
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(URI.create(hdfsFilePath), conf);
    /* 步骤2：获取输入流hdfsInputStream */
    FSDataInputStream hdfsInputStream = fs.open(new Path(hdfsFilePath));
    /* 步骤3：利用输入流读取HDFS文件 */
    // 写入本地文件的输出流
    FileOutputStream localOutputStream = new FileOutputStream(localFilePath);
    // 将HDFS文件的输入流拷贝至本地文件的输出流
    IOUtils.copyBytes(hdfsInputStream, localOutputStream, 4096, true);
  }
  ......
}
```

课后阅读

- Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google File System. In SOSP (pp. 29–43).

本章小结

- 设计思想
- 体系架构
- 工作原理
- 容错机制
- 编程示例