

# 分布式系统概论 Lab1

指导老师：冯红伟

October 12, 2023

## Contents

<b>1 设计</b>	<b>3</b>
1.1 NameNode . . . . .	3
1.2 DataNode . . . . .	4
1.3 Client . . . . .	4
<b>2 实现</b>	<b>4</b>
2.1 NameNode . . . . .	4
2.2 DataNode . . . . .	5
2.3 Client . . . . .	5
2.4 Notes . . . . .	6
<b>3 上传</b>	<b>6</b>
3.1 截止日期 . . . . .	6
3.2 提交内容 . . . . .	6
<b>4 测试</b>	<b>6</b>
4.1 单元测试 . . . . .	6

4.2 手动测试 . . . . .	6
4.3 FsImage . . . . .	8
<b>Appendices</b>	<b>9</b>
<b>A 项目结构</b>	<b>9</b>

利用 Java Corba 实现分布式文件系统，参考 `hadoop`<sup>1</sup>。

HDFS (Hadoop Distributed File System) 是一个分布式文件系统，旨在解决大规模数据的存储和处理问题。它是 Apache Hadoop 项目的核心组件之一，为 Hadoop 提供了高可靠性、高吞吐量和高容错性等特性，广泛应用于大数据处理和分析场景中。

HDFS 的设计基于 Google File System (GFS) 的思想，采用了 master/slave 架构。其中，NameNode 作为 master 节点，维护文件系统的命名空间、文件目录结构以及文件与块的映射关系；DataNode 作为 slave 节点，负责存储和管理实际的数据块。客户端通过与 NameNode 交互获取文件的元数据信息，然后与 DataNode 直接交互读写文件数据。

## 1 设计

HDFS 文件系统包含一个 NameNode 和多个 DataNode，以及客户端程序。<sup>2</sup>

### 1.1 NameNode

NameNode 作为服务端，维护文件系统的元数据信息，包括文件目录结构、文件大小、文件所在的 DataNode 以及文件划分的数据块信息等。它会响应客户端的请求，并返回相关的元数据信息。它的主要功能如下：

- 管理文件系统的命名空间，包括文件和目录的创建、删除、移动、重命名等操作；
- 维护数据块的映射关系，即每个数据块所在的 DataNode 列表；
- 处理客户端的文件读写请求，并返回相应的 DataNode 地址和数据块位置信息；
- 处理 DataNode 的心跳信息和块状态报告，以维护数据块的完整性和可用性；<sup>3</sup>
- 备份和恢复文件系统的元数据，以保证文件系统的可靠性和可用性。

---

<sup>1</sup>hadoop 架构设计官方文档 [https://hadoop.apache.org/docs/r1.0.4/cn/hdfs\\_design.html](https://hadoop.apache.org/docs/r1.0.4/cn/hdfs_design.html)

<sup>2</sup>参考学习 <https://blog.csdn.net/hudiefenmu/article/details/37655491>

<sup>3</sup>本 Lab 不作要求

## 1.2 DataNode

DataNode 作为服务端，负责存储文件的数据块。它会接收来自客户端或其他 DataNode 的请求，并响应相应的数据块。它的主要功能如下：

- 存储数据块，并维护数据块的完整性和可用性；
- 处理客户端数据读写请求；
- 定期向 NameNode 发送心跳信息和块状态报告，以维护数据块的完整性和可用性。

## 1.3 Client

Client 作为客户端，向 NameNode 请求文件的元数据信息，然后与 DataNode 直接交互读写文件数据。它的主要功能如下：

- 向 NameNode 发送文件读写请求，并获取相应的 DataNode 地址和数据块位置信息；
- 向 DataNode 发送数据读写请求，并获取相应的数据块内容；
- 处理文件和目录的创建、删除、移动、重命名等操作。

# 2 实现

基于上述设计思路，你需要实现以下功能：

实现一个基于 Java CORBA 的分布式文件系统，包含一个 NameNode 和多个 DataNode，以及客户端程序。

## 2.1 NameNode

- 维护 `FsImage`，文件系统的命名空间和目录结构，管理文件的元数据信息，包括文件大小、所在的 DataNode 以及文件划分的数据块信息等。
- 响应客户端的请求，返回相应的元数据信息。

1. `open(filepath, r/w4)`: 返回文件的元数据，包括大小、文件块所在 DataNode (一个文件由多个块组成，可能分布在不同的

---

<sup>4</sup>同一个文件只能有一个写的 open 请求，可以有多个读的 open 请求

DataNode 上)、创建时间、修改时间和访问时间<sup>5</sup>。若路径指向不存在的文件，则新建文件。当前文件正在被写入时，新的 open(filename, w) 请求将会返回 null

2. close(filepath): 更新文件的元数据写入硬盘

## 2.2 DataNode

- 存储文件的数据块，大小为 4KB。
- 维护自己的一个目录，用于存储数据文件。
- 响应来自客户端的请求，返回相应的数据块。
  1. byte[] read(block id): 返回某个数据块的所有数据
  2. append(block id, byte[] bytes to write): 向某个数据块末尾写入/追加数据

## 2.3 Client

- 向 NameNode 请求文件的元数据信息。
- 与 DataNode 直接交互读写文件数据。
- 提供的用户接口：
  1. open(filepath, r/w): 向 NameServer 请求获得文件的元数据，在内存中存储打开的文件信息（元数据和读写模式），返回打开的文件信息在内存中的标识符 fd
  2. append(fd, byte[] bytes): 检查 fd 的读取模式，仅当 w ∈ mode 时向文件追加数据
  3. byte[] read(fd): 检查 fd 的读取模式，仅当 r ∈ mode 时返回某个文件的所有内容，否则返回 null
  4. close(fd): 关闭打开的文件，释放内存中 fd 的文件信息，向 NameNode 发送请求，将文件元数据更新持久化写入硬盘

---

<sup>5</sup>三种时间戳的定义参考 Unix 文件系统，本次 Lab 对文件的访问控制不作要求，即不需要实现用户和用户组

## 2.4 Notes

实验基于以下假设，以简化 HDFS 的实现。

- 节点间信道可靠
- 节点不会失效，即不需要实现心跳机制。NameNode 数据不会损坏，不需要实现 EditLog
- 数据块不支持随机访问，即只能向末尾追加内容，只能读取整块数据
- 服务发现已经实现，你可以将每个 Node 的 IP:port 硬编码或者作为程序启动时的命令行参数

## 3 上传

### 3.1 截止日

2023/11/14 23:59(GMT+8)

### 3.2 提交内容

1. 中期检查（5%），此项目是为了防止有同学最后一个周才开始Lab而导致时间不足，需要提交半成品的设计文档（说明实现的部分）和代码。
2. 设计文档（35%），介绍你的设计思路和实现。包括但不限于 FslImage 的设计，文件和数据块的映射和数据块定位、多副本的实现、读数据多个 DataNode 可用时的选择策略。
3. 实现代码（60%），基于 Java Corba 实现 HDFS 三个模块，实现 2 部分提到的函数（6 分/个），要求良好代码风格和模块化设计（占 12 分）。

## 4 测试

测试分为两部分，JUnit 单元测试和手动测试。

### 4.1 单元测试

已经提供在项目文件的 src/test 目录。

### 4.2 手动测试

在命令行执行以下命令

```
# terminal 1
```

```

cd src && mkdir bin
idlj -fall api.idl
javac src/*.java src/api/*.java src/impl/*.java src/utils/*java
-d bin/
orbd -ORBInitialPort 1050 -ORBInitialHost localhost

# terminal 2 启动 NameNode
java -cp bin/ NameNodeLancher -ORBInitialPort 1050 -ORBInitialHost
localhost

# terminal 3 启动 DataNode1
java -cp bin/ DataNodeLancher -ORBInitialPort 1050 -ORBInitialHost
localhost

# terminal 4 启动 DataNode2
java -cp bin/ DataNodeLancher -ORBInitialPort 1050 -ORBInitialHost
localhost

# terminal 5 启动 Client1
java -cp bin/ ClientLancher -ORBInitialPort 1050 -ORBInitialHost
localhost

```

在命令行启动客户端后，将执行以下操作

```

>> open test.txt w
INFO: fd=1
>> read 1
INFO: READ not allowed
>> append 1 hello world
INFO: write done
>> close 1
INFO: fd 1 closed
>> open test.txt rw
INFO: fd=2
>> read 2
hello world

```

```
>> exit  
INFO: bye
```

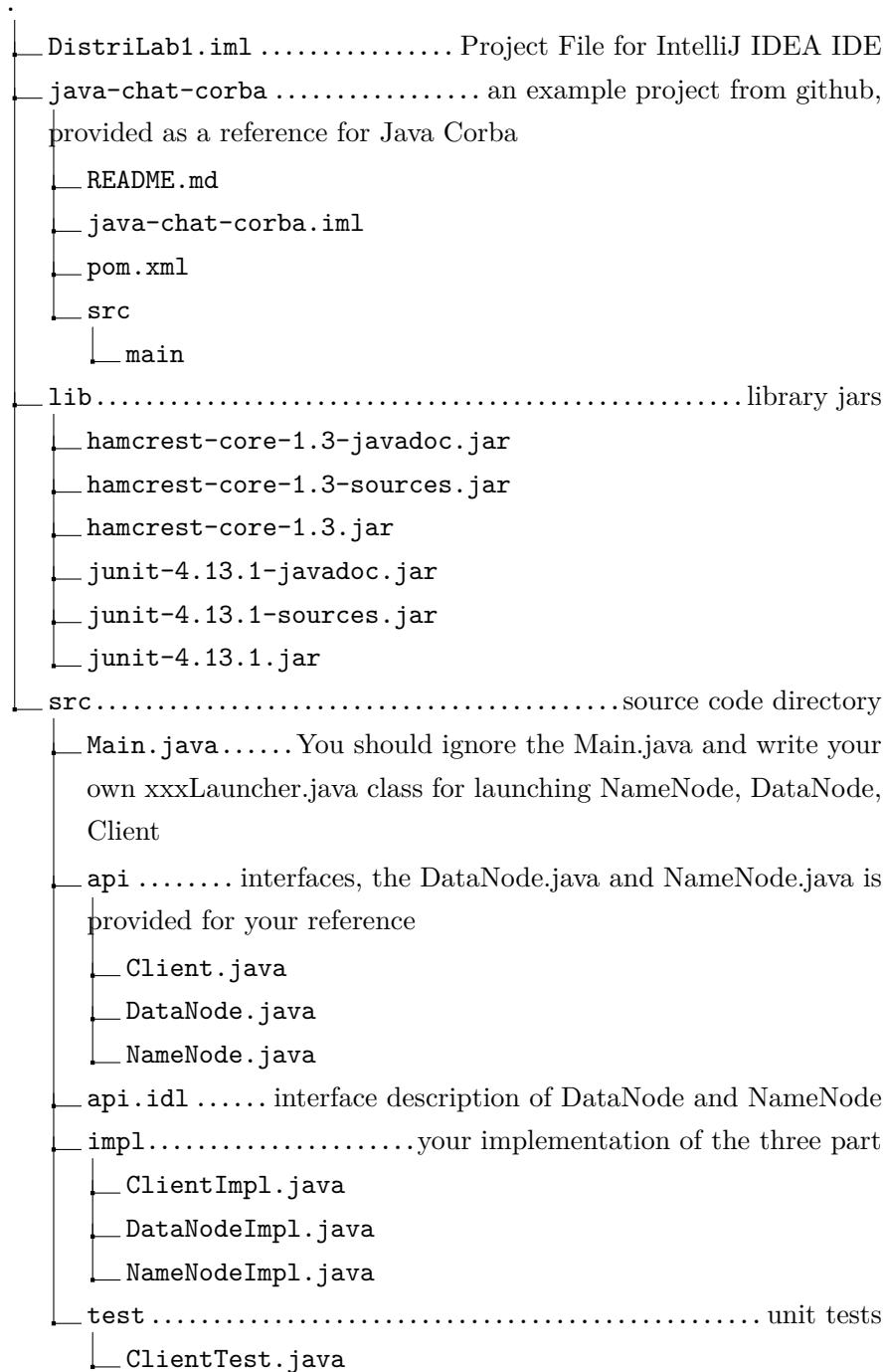
操作期间将检查服务器端文件状态

### 4.3 FsImage

请在设计文档中插入你的 NameNode 保存在磁盘上的 FsImage。并介绍该文件的格式，在手动测试 4.2 期间将会检查 close 后该文件的状态

# Appendices

## A 项目结构



```
    ┌── DataNodeTest.java  
    └── NameNodeTest.java  
utils .....helper classes  
    ├── FileDesc.java  
    └── FileSystem.java
```

9 directories, 23 files