单位代码 **10006**  

课程名称 **软件体系结构**

分 类 号 **TP311.5**

****

**HDFS分布式文件系统的建模与分析**

|  |  |
| --- | --- |
| 学院 | 计算机学院 |
| 课程 | 软件体系结构 |
| 姓名 | SY1806213 郑鹏飞 |
| 姓名 | ZY1806304 关晗金 |
| 姓名 | ZY1806818 张 瑞 |

2019年 05 月30日

目录

[1. 项目简介 3](#_Toc10148694)

[1.1 主要功能介绍 4](#_Toc10148695)

[1.1.1 核心组件 5](#_Toc10148696)

[1.1.2 文件上传 5](#_Toc10148697)

[1.1.3 文件下载 6](#_Toc10148698)

[1.2 项目代码行数 6](#_Toc10148699)

[1.3 项目包类数量 7](#_Toc10148700)

[2. 需求分析 8](#_Toc10148701)

[2.1 功能需求 8](#_Toc10148702)

[2.2 质量属性 9](#_Toc10148703)

[2.2.1 性能质量属性 9](#_Toc10148704)

[2.2.2 安全性质量属性 9](#_Toc10148705)

[2.2.3 可用性质量属性 10](#_Toc10148706)

[2.2.4 可修改性质量属性 11](#_Toc10148707)

[2.2.5 可测试性质量属性 11](#_Toc10148708)

[2.2.6 易用性质量属性 12](#_Toc10148709)

[2.3 其他设计需求 12](#_Toc10148710)

[2.3.1 互操作性 12](#_Toc10148711)

[2.3.2 健壮性 13](#_Toc10148712)

[2.3.3 可移植性 13](#_Toc10148713)

[2.3.4 可重用性 13](#_Toc10148714)

[2.3.5 可扩充性 14](#_Toc10148715)

[3. 软件体系结构设计 15](#_Toc10148716)

[3.1 软件设计思路 15](#_Toc10148717)

[3.2 体系结构描述 16](#_Toc10148718)

[3.2.1 通信协议模块层 18](#_Toc10148719)

[3.2.2 核心模块层 23](#_Toc10148720)

[3.2.3 客户端模块层 37](#_Toc10148721)

[3.2.4 工具及安全模块层 43](#_Toc10148722)

[3.3 结构模型分析 47](#_Toc10148723)

[4. 设计特色分析 49](#_Toc10148724)

[4.1 数据高容错性 49](#_Toc10148725)

[4.1.1 特色介绍 49](#_Toc10148726)

[4.1.2 设计对比 51](#_Toc10148727)

[4.1.3 需求支撑 52](#_Toc10148728)

[4.2 大数据存储 52](#_Toc10148729)

[4.2.1 特色介绍 52](#_Toc10148730)

[4.2.2 设计对比 53](#_Toc10148731)

[4.2.3 需求支撑 54](#_Toc10148732)

[4.3 流式数据访问 55](#_Toc10148733)

[4.3.1 特色介绍 55](#_Toc10148734)

[4.3.2 设计对比 55](#_Toc10148735)

[4.3.3 需求支撑 56](#_Toc10148736)

[5. 组内分工情况 57](#_Toc10148737)

[6. 参考文献 58](#_Toc10148738)

[7. 附件：体系结构的分解过程 59](#_Toc10148739)

[7.1 HDFS总体架构图 59](#_Toc10148740)

[7.2 EA绘制HDFS类图 59](#_Toc10148741)

[7.3 Xmind文件 59](#_Toc10148742)

## 1. 项目简介

HDFS[6]（Hadoop Distributed File System[5]）是Hadoop项目的子项目，Hadoop是一个大数据分布式存储及计算平台，其架构[10]如图1.1所示。

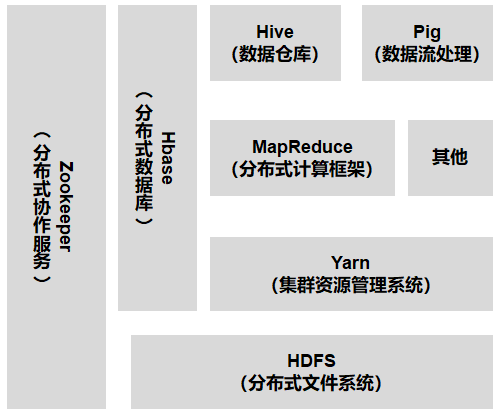


图1.1 Hadoop架构图

Hadoop项目的核心组件主要包括三个部分[8][9]：HDFS（分布式文件系统）、Yarn（集群资源管理系统）和MapReduce（分布式计算框架）。除此之外，Zookeeper用于解决分布式环境下的数据管理问题，Hive用于将类似SQL的查询语言HQL转化为MapReduce任务，Pig用于将脚本转化为MapReduce任务，在Hadoop上执行[2][3]。

Hadoop组件的语境图如图1.2所示[12]。虚线内的范围代表整个Hadoop系统，系统开始于任务的提交，该任务可以是HQL查询，也可以是Pig Latin脚本等，对于不同的任务，有不同的组件将其转化为MapReduce任务，MapReduce[11]在海量数据上进行计算，通过Yarn组件获得CPU等系统资源，通过Hbase随机访问结构化的数据执行任务。其中Hbase建立在HDFS文件系统之上，提供了对大规模数据的随机、实时读写访问，它将数据存储和并行计算完美地结合在了一起。

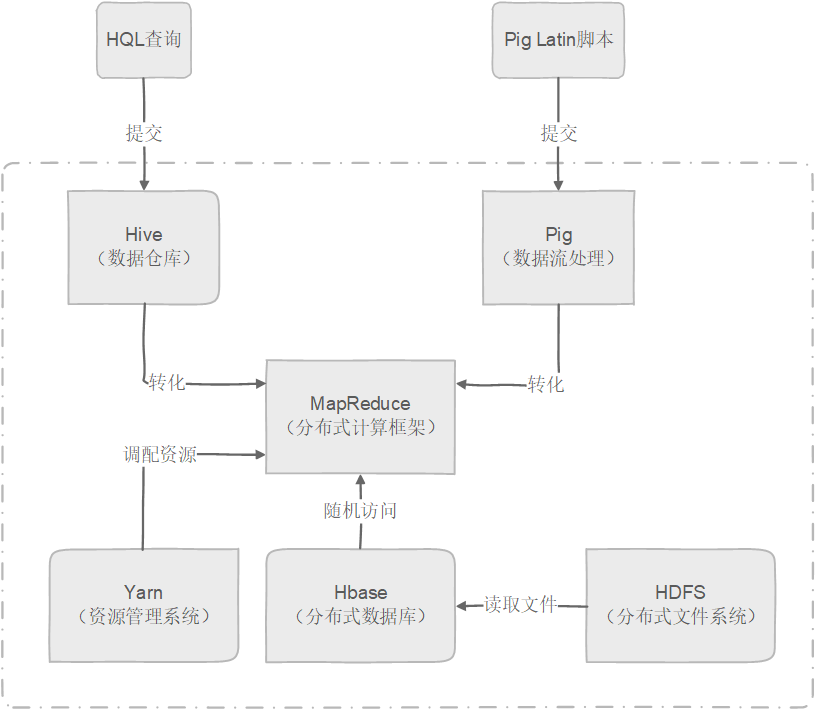


图1.2 hadoop项目语境图

本文我们主要讨论HDFS文件系统，HDFS是分布式计算中数据存储管理的基础，被设计成适合运行在通用硬件上、提供流式操作、能够处理超大文件的分布式文件系统。它所具有的高容错性、高可靠性、高可扩展性和高吞吐率等特征为海量数据提供了可靠高效的存储。

HDFS存储在实际应用中已达到PB级别。Facebook在2012年宣布他们拥有最大的单个HDFS集群，数据量超过100PB；2018年，雅虎部署了超过36000台服务器，跨17个YARN集群使用680PB的HDFS存储数据。

### 1.1 主要功能介绍

HDFS作为一个分布式文件系统，具有最主要的两个功能：上传和下载。而为了保障这两个功能的高效与可靠的实现，HDFS还提供了一些辅助功能，如心跳机制、安全模式、副本存放策略、负载均衡等。

#### 1.1.1 核心组件

1. 数据块（Block）：HDFS文件的存储形式以及HDFS文件处理最小单元
2. 名字节点（Namenode）：管理文件系统的命名空间包括目录信息及索引信息，同时保存数据块与数据节点之间的对应关系
3. 数据节点（Datanode）：负责执行名字节点发送的指令如创建、删除或者复制数据等
4. 客户端：包括命令行接口、浏览器接口以及代码API接口
5. 通信协议：抽象了HDFS各个节点之间的调用接口

#### 1.1.2 文件上传

HDFS文件上传的流程图如图1.3所示，HDFS客户端发起文件上传请求，通过RPC通信协议与Namenode建立通讯。Namenode会返回一个可存储的DataNode列表，客户端在收到这个列表后，就会建立与Datanode的数据流管道。在成功建立数据流通道后，HDFS客户端就可以像Datanode节点写入数据。当HDFS客户端完成整个文件中所有数据块的写操作后，就会关闭输出流，从而完成整个文件的写入流程。

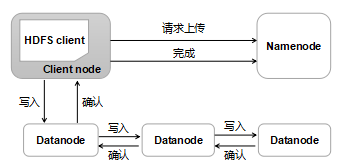


图1.3 文件上传流程图

#### 1.1.3 文件下载

HDFS文件下载功能流程如图1.4所示，HDFS客户端像Namenode发送文件下载请求，Namenode获得请求后检查文件是否存在以及用户是否有权限。若检查没有问题，Namenode返回数据对应Datanode的存储位置。HDFS客户端拿到存储信息后连接到对应的Datanode进行块的下载，当所有数据块下载完成后，客户端向Namenode反馈。

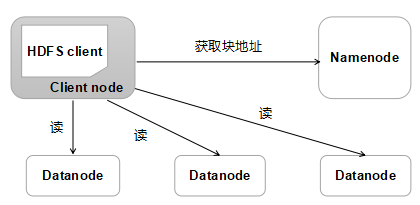


图1.4 文件下载流程图

### 1.2 项目代码行数

HDFS分布式文件系统的主要编程语言为java[27]，除此之外在HDFS的本地客户端模块是由C/C++编写，所以我们主要统计扩展名为.java、.h、.c的文件，统计结果如图1.5所示。

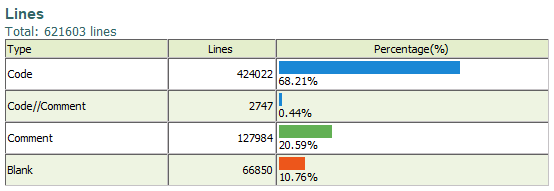


图1.5 代码行数统计

由统计结果，HDFS项目总行数为621603。其中代码行数为424022，注释行数为127984，代码//注释行数为2747，空行数为66850。

### 1.3 项目包类数量

项目的主要部分的包数量为85个，另外分别对java文件和c++文件进行类数量的统计，统计结果如下图1.6所示。

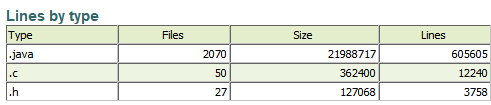


图1.6 类数量统计结果

由统计结果，java类的数量为2070，c++类的数量为27。

## 2. 需求分析

### 2.1 功能需求

HDFS（分布式文件系统）是Hadoop的核心组件之一[1]，用来将海量数据文件分布式地存储在很多的服务器上。与普通的单机文件系统相比，HDFS的文件系统会横跨多个机器，存储在多个机器的单机文件系统中。作为一个分布式文件系统，它为用户及上层应用提供了一般文件系统所共有的功能[17]：查看、上传、下载、拷贝、移动、删除等，其用例图如下图2.1所示。

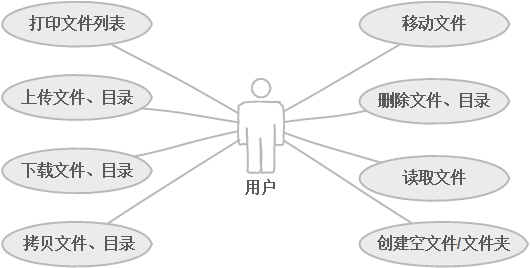


图2.1 用例图

在HDFS分布式文件系统中，参与者为使用此系统的用户，此参与者的用例共有8个，具体为对文件的基本操作，其用例描述如表2.1所示。

**表2.1 用例描述表**

|  |  |  |
| --- | --- | --- |
| 用例名称 | 用例描述 | 参与者 |
| 打印文件列表 | 递归获取HDFS路径下所有的文件名 | 用户 |
| 上传文件、目录 | 上传单个文件或整个目录里的文件夹及文件到目标存放路径 | 用户 |
| 下载文件、目录 | 下载单个文件或整个目录下的文件夹及文件到本地存放路径 | 用户 |
| 拷贝文件、目录 | 将现有集群的文件或目录复制到新的集群 | 用户 |
| 移动文件 | 将现有集群的文件移动到新的集群 | 用户 |
| 删除文件、目录 | 删除单个或整个目录下的文件夹及文件 | 用户 |
| 读取文件 | 读取指定的文件信息 | 用户 |
| 创建空文件/文件夹 | 指定路径下创建文件或级联创建目录 | 用户 |

### 2.2 质量属性

在运行时观察确定：性能、安全性、可用性、功能性、易用性。

不可通过观察系统运行确定：可修改性、可移植性、可重用性、可集成性、可测试性。

#### 2.2.1 性能质量属性

性能是指系统的响应能力，即对外部刺激（事件）做出反应所需要的时间或某段时间内所处理的事件个数，可用等待时间、处理期限、系统吞吐量、响应抖动、缺失率、数据丢失等指标来度量。HDFS分布式文件系统的主要优点为系统吞吐量高，所以我们使用压力测试下的系统吞吐量作为度量指标，其质量属性场景生成如表2.2所示。

**表2.2 性能质量属性场景生成**

|  |  |
| --- | --- |
| 场景 | 可能的值 |
| 源 | 用户 |
| 刺激 | 大量用户上传、下载文件 |
| 制品 | 文件系统 |
| 环境 | 用户正常使用HDFS文件系统 |
| 响应 | 大量的操作同时被处理  对HDFS存储进行负载均衡  Namenode内存资源大量占用 |
| 响应度量 | 系统每时间段内完成的事件个数与HDFS集群硬件资源水平保持一致 |

#### 2.2.2 安全性质量属性

HDFS分布式文件存储系统需要保证集群中数据块的安全性，为此HDFS提供了安全模式，当系统处于安全模式时会检查数据块的完整性。并且在此模式下，用户只能读取数据信息而不能由任何修改数据信息的操作。其质量属性场景如表2.3所示。

**表2.3 安全性质量属性场景生成**

|  |  |
| --- | --- |
| 场景 | 可能的值 |
| 源 | 用户 |
| 刺激 | 用户启动HDFS系统或数据块丢失 |
| 制品 | 文件系统 |
| 环境 | HDFS文件系统正常启动与运行 |
| 响应 | 系统进入安全模式  系统关闭hadoop所有服务  系统检查数据块完整性  系统自动退出安全模式 |
| 响应度量 | 用户正常使用系统功能 |

#### 2.2.3 可用性质量属性

可用性是指系统修复故障的能力，也可以定义为系统正常运行的时间比例。衡量系统可用性一般可用公式MTBF / (MTBF + MTTR)，其中MTBF为系统的平均正常工作时间，而MTTR代表系统修复错误用的时间。该公式结果越大，证明系统的可用性越好。由于HDFS集群内的机器存在宕机风险，Namenode也存在单点失效的问题，HDFS的高可用性配置了一堆活动-备用Namenode，当活动Namenode失效后，备用Namenode就会接管它的任务，避免系统的重新启动，并且HDFS提供的副本存放策略将数据存放在三个节点上，避免了数据丢失的风险，保证了系统可用性，可用性的质量属性场景生成如表2.4所示。

**表2.4 可用性质量属性场景生成**

|  |  |
| --- | --- |
| 场景 | 可能的值 |
| 源 | Namenode |
| 刺激 | Namenode单点故障或机器崩溃 |
| 制品 | 文件系统 |
| 环境 | HDFS文件系统正常运行 |
| 响应 | 启动故障转移控制器  备用Namenode同步命名空间状态  启用备用Namenode恢复故障 |
| 响应度量 | 系统正常运行 |

#### 2.2.4 可修改性质量属性

可修改性是有关变更的成本问题，其关注可以修改的系统的任何方面，包括系统存在平台及系统运行环境等。HDFS使用Java实现，任何装有Java运行环境的机器都可以配置Namenode和Datanode。Java具有高可一致性，也就意味着HDFS可以轻易在不同平台间做到移植，可修改性质量属性场景生成如表2.5所示。

**表2.5 可修改性质量属性场景生成**

|  |  |
| --- | --- |
| 场景 | 可能的值 |
| 源 | 用户 |
| 刺激 | 用户希望移植HDFS系统到一台运行其他系统的机器 |
| 制品 | HDFS系统运行平台 |
| 环境 | HDFS文件系统运行时 |
| 响应 | 新机器上配置Java环境  新机器上部署HDFS文件系统 |
| 响应度量 | HDFS系统在新机器上正常运行 |

#### 2.2.5 可测试性质量属性

系统的可测试性是指通过测试揭示软件缺陷的容易程度。要想对系统进行正确的测试，就要能够控制每个组件的内部状态及输入，然后观察输出。在此，我们选择当系统出现新版本时作为可测试质量属性场景，其场景生成如表2.6所示。

**表2.6 可测试性质量属性场景生成**

|  |  |
| --- | --- |
| 场景 | 可能的值 |
| 源 | 用户 |
| 刺激 | HDFS文件系统发布新版本 |
| 制品 | 系统新特性 |
| 环境 | 部署时 |
| 响应 | 提供测试数据  准备测试环境 |
| 响应度量 | 执行测试的时间 |

#### 2.2.6 易用性质量属性

易用性关注对用户来说完成某个期望任务的容易程度和系统所提供的用户种类，主要分为学习系统的特性、有效使用系统、使系统适应用户需要等。易用性质量属性的场景生成如表2.7所示。

**表2.7 易用性质量属性场景生成**

|  |  |
| --- | --- |
| 场景 | 可能的值 |
| 源 | 用户 |
| 刺激 | 用户想要学习系统特性和有效使用系统 |
| 制品 | 文件系统 |
| 环境 | HDFS文件系统运行时 |
| 响应 | 用户界面交互 |
| 响应度量 | 响应时间、用户成功操作的比例 |

### 2.3 其他设计需求

#### 2.3.1 互操作性

互操作性是一种衡量一组部件（构成一个系统）与另一个系统协作的能力。HDFS分布式文件系统是Hadoop的一个子系统，除了HDFS，Hadoop还包括Yarn资源管理系统以及MapReduce。HDFS分布式文件系统负责海量数据的存储，Yarn主要负责管理集群的计算资源并根据计算框架的需求进行调度，而MapReduce是一个分布式程序运算框架。HDFS可类比为硬盘，存储文件；Yarn相当于操作系统，调配CPU等资源；MapReduce就好比电脑上的一个应用程序。运行一个应用程序需要去硬盘读取文件，并且应用程序的运行结果要保存到硬盘，所以MapReduce和HDFS之间存在着协作关系，需要衡量他们之间的传输效率。

#### 2.3.2 健壮性

健壮性是指软件对于规范要求外的输入情况的处理能力。一个健壮的系统对于规范要求以外的输入要能判断这个输入不符合规范，并且有合理的处理方式。HDFS的主要目标就是实现在失败情况下数据存储的可靠性。HDFS对于数据维护着多个副本，并且会遵循副本存放策略降低数据丢失的可能性。HDFS支持数据的均衡计划，当某个Datanode的空闲空间低于临界点时，会将数据从另一个Datanode转移到空闲的Datanode[14]。HDFS支持快照，快照保存着某个时间的数据副本，当HDFS数据损坏时，可以恢复到过去一个已知的正确时间点。

#### 2.3.3 可移植性

可移植性是指程序不做任何修改就可以在任何计算机上运行，良好的可移植性可以提高软件的生命周期。HDFS使用java语言编写，java语言本身具有跨平台性，所以HDFS可以简便的实现平台间的迁移。

#### 2.3.4 可重用性

可重用性是指利用标准化的软件模块快速构建特定的应用系统。

软件开发的全生命周期都有可重用的价值，包括项目的组织、软件需求、设计、文档、实现、测试方法和测试用例都是可以被重复利用或借鉴的有效资源。HDFS分布式文件系统也应为其他文件系统提供丰富的经验。

#### 2.3.5 可扩充性

可扩充性指系统对技术和业务需求变化的支持能力。当技术变化或业务变化时，不可避免将带来系统的改变。不仅要进行设计实现的修改，甚至要进行产品定义的修改。好的软件设计应在系统架构上考虑能以尽量少的代价适应这种变化，常用的技术有面向对象的分析与设计及设计模式。运用面向对象的开发技术，抽象和模块化代码。

## 3. 软件体系结构设计

### 3.1 软件设计思路

HDFS的架构遵循主/从架构模式，主要进程为Namenode，Datanode，Secondarynamenode，在启动过程中，启动顺序也是按照这个顺序。

对于磁盘的存储，我们每个磁盘都有默认的数据块，文件系统块的大小(默认512字节)一般为数据块的整数倍。对于HDFS同样有数据块这一概念，原先官方默认的数据块为64M，目前新版本的默认为128M。若想改变这一默认，需要修改HDFS的配置文件，名称为：dfs.blocksize，默认128M。

HDFS文件系统的数据是以块为基本单位进行存储的，如果剩余的存储不足128M，会将剩余部分单独存成一个块。例如基本块是128M，对于一个150M的文件，会分成两个块进行存储，一个是128M，另一个是22M。之所以采用如此大的块，是为了最小化寻址开销。通过这一点，我们很容易意识到，HDFS对于存储大数据具有大的优势，对于碎片化的小数据的存储效率会很低。以下是，HDFS存储的数据分布示意图如图3.1所示。

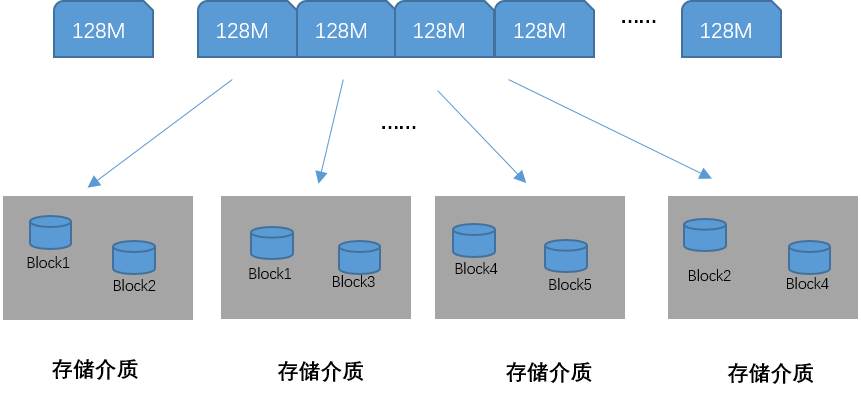


图3.1 HDFS数据存储分布示意图

通过划分基本block把数据分块存储到不同的存储介质上，同一个block可以存储多份，放置策略优先级如下：同client的节点上，不同机架的节点上，其他随机地点。通过循环冗余校验保证文件的完整性。当某一节点文件发生损坏时，利用其他副本取代当前文件。

其中最主要的核心模块是Namenode，负责文件系统的命名空间、文件名称和字节数、文件目录结构、文件属性（权限、创建时间、副本数）、该文件的数据块以及对应的DataNode节点。可以类比操作系统文件管理中的iNode节点，主要负用于对文件进行管理；DataNode主要负责文件的数据校验。

总之通过分块存储，冗余备份，NameNode管理，hdfs实现了一个分布式，容错性高的文件系统。

### 3.2 体系结构描述

Kruchten指出，软件体系结构有四个角度，它们从不同方面对系统进行描述：概念角度描述系统的主要构件及它们之间的关系；模块角度包含功能分解与层次结构；运行角度描述了一个系统的动态结构；代码角度描述了各种代码和库函数在开发环境中的组织。

其中概念角度在第一部分已经论述，代买角度和运行角度会在具体的模块分析中阐述，在这一节点，我们聚焦在模块角度：

所谓软件的模块划分是指在软件设计过程中，为了能够对系统开发流程进行管理，保证系 统的稳定性以及后期的可维护性，从而对软件开发按照一定的准则进行模块的划分。根据 模块来进行系统开发，可提高系统的开发进度，明确系统的需求，保证系统的稳定性。

软件设计过程中通过对软件进行模块划分可以达到一下的好处：

(1) 使程序实现的逻辑更加清晰，可读性强。

(2) 使多人合作开发的分工更加明确，容易控制。

(3) 能充分利用可以重用的代码。

(4) 抽象出可公用的模块，可维护性强，以避免同一处修改在多个地方出现。

(5) 系统运行可方便地选择不同的流程。

(6) 可基于模块化设计优秀的遗留系统，方便的组装开发新的相似系统，甚至一个全新的系统。

HDFS是基于功能模块进行划分的。从第一部分的需求分析中并结合HDFS的使用场景，可以抽象出，HDFS的主要功能模块[13]：

通信协议模块：分布式存储系统，文件分布在不同的主机或者联网的存储设备上，为了方便快捷的进行文件的备份、存储、转存必须对不同设备间的通信做好优化，这一步骤就是进行这个操作；

核心模块：所谓的核心模块，也就是HDFS的服务器端，在这里实现块管理、namenode管理、DataNode等核心功能；

客户端模块：相对于核心模块，这一模块为使用server提供的各种服务进行一些接口的设计；

应用模块：这一部分着重应用层的实现，提供了各种各样的方法，对文件系统进行各种操作；

工具及安全模块：为HDFS的正常使用提供必要的工具包支持，对文件系统的安全性进行软件级别的保障。

见附录：模块分布的思维导图

图3.2是HDFS的体系结构图[7]：



图3.2 HDFS的体系结构图

#### 3.2.1 通信协议模块层

RPC[18]（Remote Procedure Call Protocol）——远程过程调用协议，它是一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。RPC协议假定某些传输协议的存在，如TCP或UDP，为通信程序之间携带信息数据。在OSI网络通信模型中，RPC跨越了传输层和应用层。RPC使得开发包括网络分布式多程序在内的应用程序更加容易。

##### 3.2.1.1 软件体系结构风格

HDFS中定义了Client端、DataNode、NameNode，三者的数据通信协议[19]，故通信协议的设计就十分的重要。

图3.3为三者之间通信所用到的部分协议。

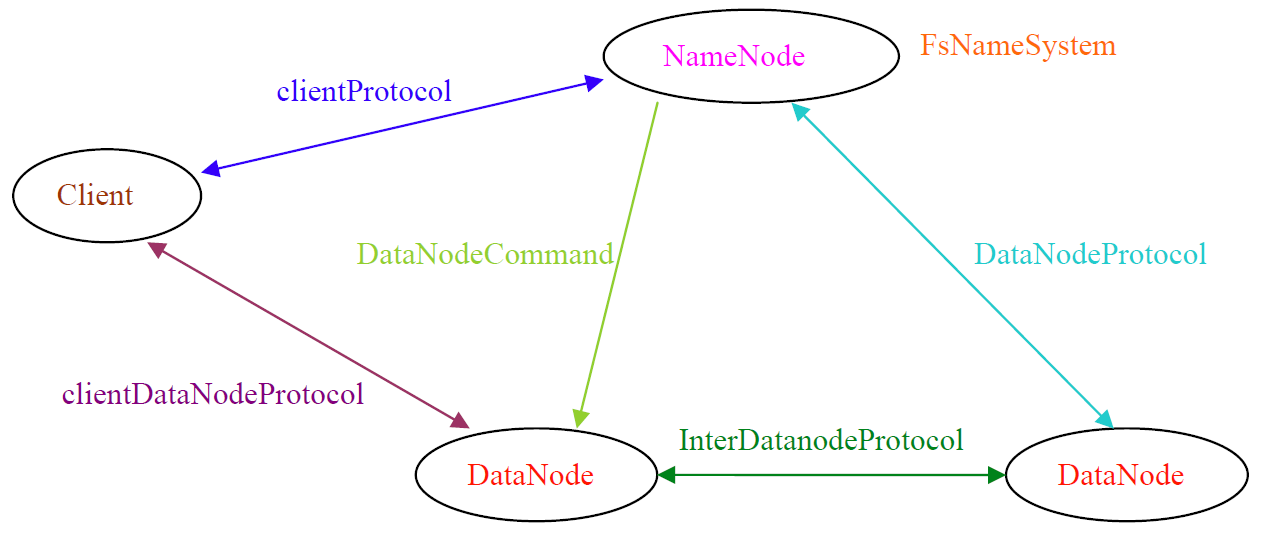


图3.3 HDFS中通信协议的使用图

图3.4为通信协议模块部分类图。

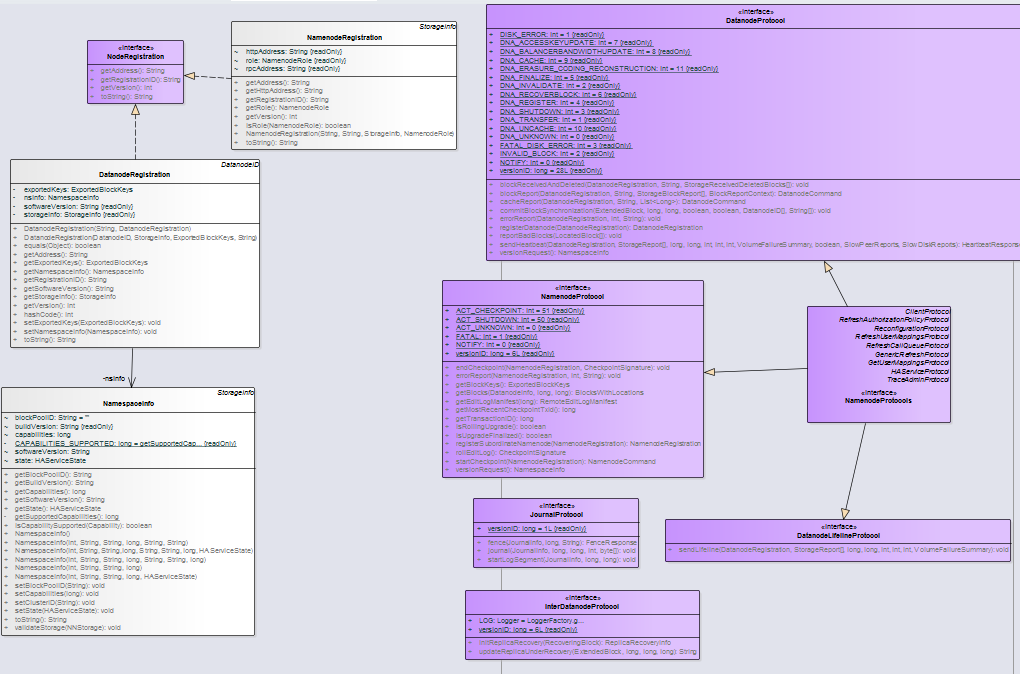


图3.4 通信协议模块部分类图

##### 3.2.1.2 策略/战术

通信协议注重对通信带宽，资源使用，内存消耗等因素的考虑。在该模块中采用的性能战术，其目标是在一定的时间限制内到达系统的事件生成一个响应。事件达到系统后系统或者对改事件进行处理，或者由于某些原因处理被阻塞。例如，DataNode与NameNode之间使用的心跳机制，如果DataNode在一定的时间内未发送心跳信号，或者心跳信号发生丢失等问题，那么NameNode在该通信中将无法获得该事件的信号，便触发安全机制，以维护数据的正常存取功能，并进行数据的备份等其他操作。

在通信中存在资源的争用，这些事件可能是单个流也可能是多个流。争用同一个资源的多个流或者相同流争用同一个资源的不同事件会增加等待事件。此外资源的可用性，即使没有争用的存在，如果资源不可用，那么通信将无法正常进行下去。资源离线、组件故障或者其他原因都会导致资源不可用。

在通信中需要处理通讯的周期，如果通信较为频繁则计算资源将会超量占用资源，设计多个流之间的和谐通信周期有利于降低资源开销从而降低通信消耗。通过减少处理需求，并限制通信信号的等待时间，并添加通信等待队列及这，可以控制通信的平均响应时长。

此外在通信过程中需要关注数据的完整性和安全性，为此通信的实体之间需要进行身份的注册，在通信协议层中相对于的就是Register等一系列类。通过进行注册的方式可以限制数据消息的访问，达到数据传输中的安全性。

##### 3.2.1.3 设计模式

在DataNode和NameNode进行通信中，使用了**命令模式**。如图3.5所示。

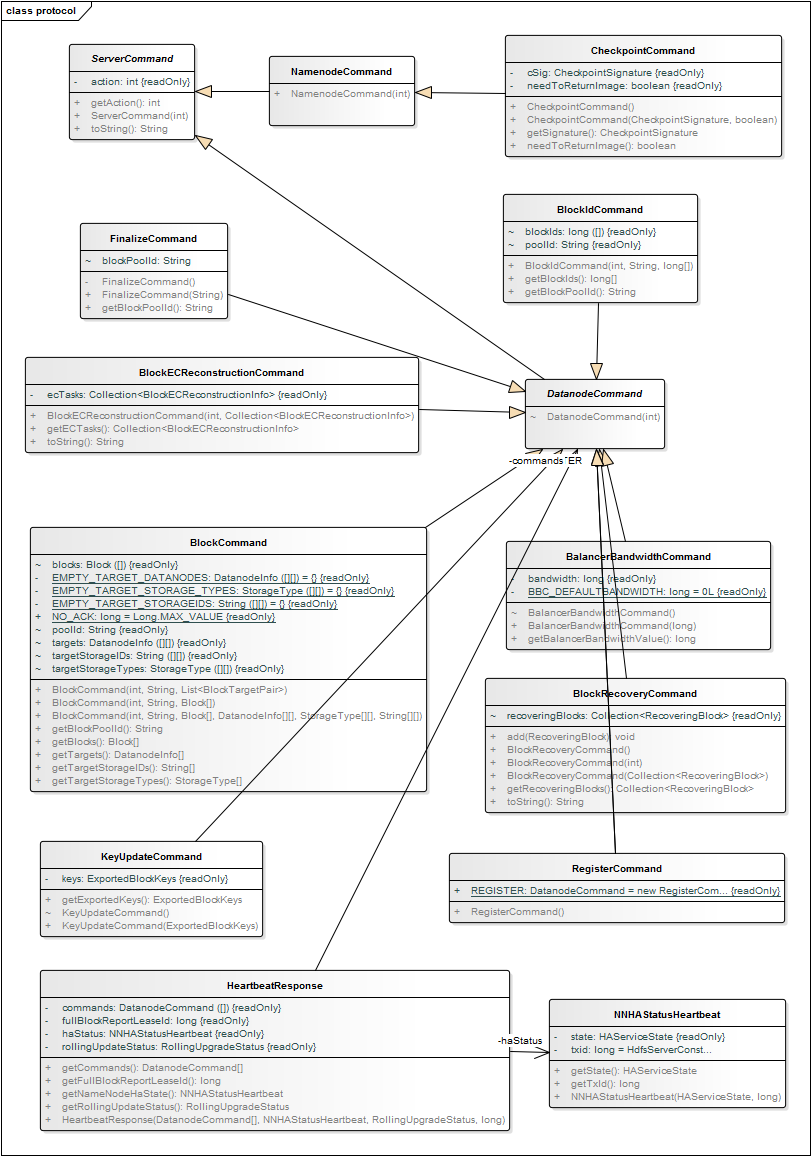


图3.5 通信协议模块中的命令模式

在上述的命令模式中定义了DataNode与NameNode之间通信的部分Command。

以DataNode与NameNode的定期心跳为例。名字节点根据数据节点的定期心跳，判断节点是否正常工作。在心跳上报过程中，数据节点发送能够描述当前节点负载的一些信息，例如数据的存储容量、目前已使用量等。名字节点根据这些信息估计数据节点的工作状态，均衡各个节点的负载。在远程方法sendHeart执行结束之后，会携带名字节点到数据节点的指令，数据节点执行这些指令，保证HDFS系统的健康、稳定运行。上述的这些指令均最后有DataNode.processCommand()方法处理，方法的主体为一个case语句，根据命令编号执行不同的方法。

HDFS在数据通信协议中使用了**代理模式**。如图3.6所示。

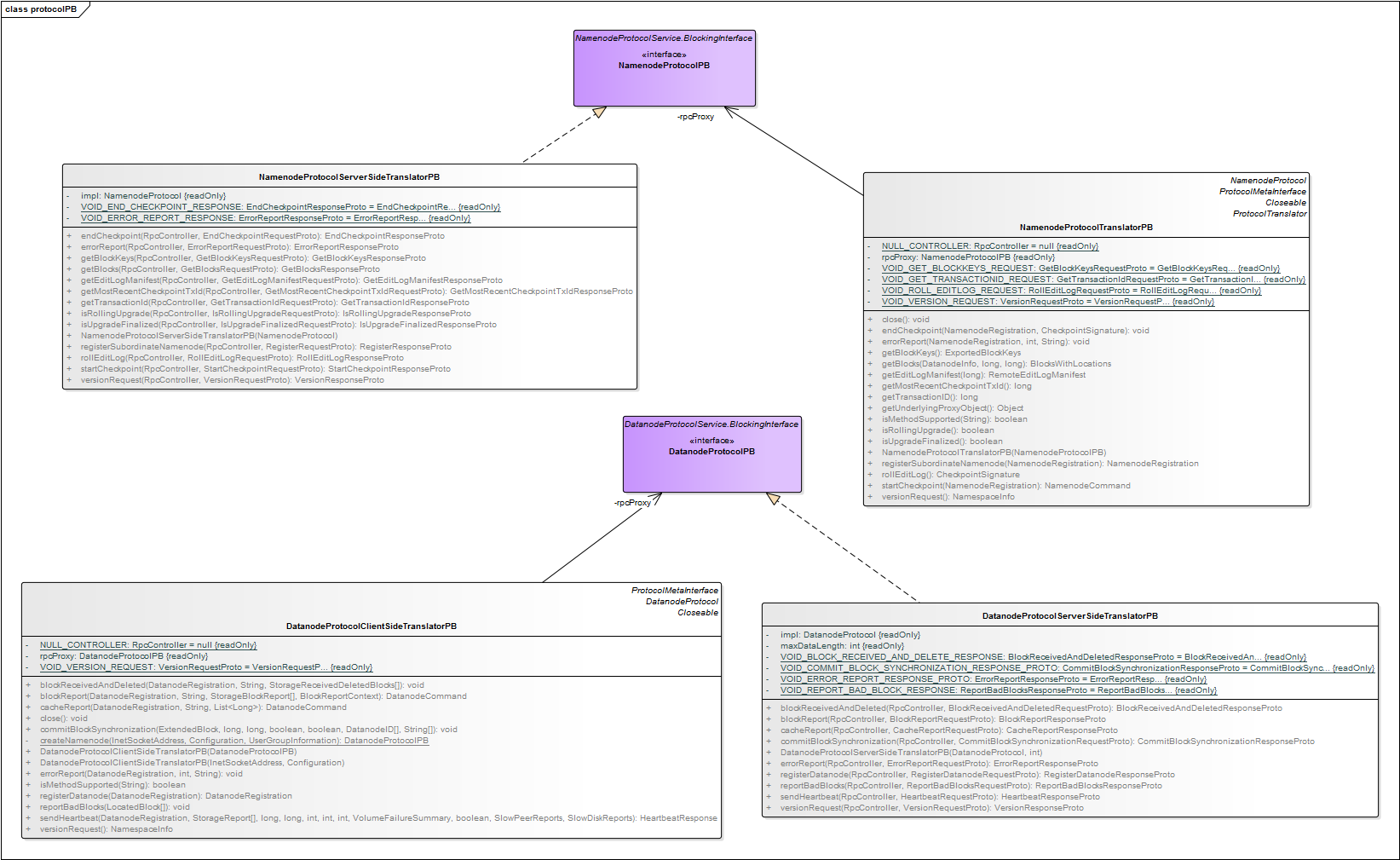


图3.6 通信协议模块中的代理模式

通信协议中的的RPC采用客户端/服务器模式。客户端调用进程发送一个有进程参数的调用信息到服务进程，然后等待应答消息。在服务器端，进程保持休眠状态直到调用信息到达为止。当一个调用信息到达，服务器获得进程参数，计算结果，发送答复信息，然后等待下一个调用信息。客户端调用进程接收答复信息，获得进程结果，然后调用执行继续进行。

##### 3.2.1.4 领域框架

通信协议层时基于protocol buffer[20]持久化框架进行实现的。protocol buffer是google的一种数据交换的格式，它独立于语言，独立于平台。协议缓冲区（Protobuf）是一种序列化结构化数据的方法，该方法用作自定义远程过程调用（RPC）系统的基础，该系统几乎用于Google的所有机器间通信。该方法涉及描述一些数据的结构的接口描述语言和从该描述生成源代码以生成或解析表示结构化数据的字节流的程序，有助于开发程序通过电线相互通信或存储数据。

google提供了多种语言的实现，包括java、c++ 、python、C#等[21]。每一种实现都包含了相应语言的编译器以及库文件。由于它是一种二进制的格式，比使用xml进行数据交换快许多。作为一种效率和兼容性都很优秀的二进制数据传输格式，可以用于诸如网络传输，配置文件，数据存储等诸多领域。

##### 3.2.1.5 技术分析

HDFS中提供两种通信协议，一种为HDFS中基于Hadoop RPC 框架实现的接口，另一种是HDFS中基于TCP或者HTTP实现的接口。

Hadoop RPC调用使得HDFS进程能够像本地调用一样调用另一个进程中的方法，目前Hadoop RPC调用是基于Protobuf实现。其中接口的定义在org.apache.hadoop.hdfs.protocol和org.apache.hadoop.hdfs.server.protocol包中，接口详情如表3.1所示。

**表3.1 可通信协议模块接口详情表**

|  |  |  |
| --- | --- | --- |
| 协议名称 | 使用者 | 备注信息 |
| clientProtocol | Client  NameNode | 用于访问NameNode。它包含了文件角度上的HDFS功能。 |
| clientDataNodeProtocol | Client  DataNode | 用于访问DataNode。一部分是支持HDFS文件读取操作，另一部分是支持DFSAdmin中与数据节点管理相关的命令。 |
| DataNodeProtocol | DataNode  NameNode | 用于DataNode和NameNode通信。DataNode会使用该接口与NameNode握手、注册、发送心跳、进行全量或增量的数据块汇报。 |
| InterDataNodeProtocol | DataNode | DataNode与DataNode之间的接口，主要用于租约恢复操作。 |
| 其他Protocol |  | 主要包括安全相关接口，HA相关接口 |

#### 3.2.2 核心模块层

HDFS的核心模块主要包含两大部分：Namenode和Datanode。Namenode实现了命名空间管理层以及数据块存储管理层中的数据块管理功能，而Datanode实现了数据块存储管理层中的存储管理部分，该层的逻辑架构图如图3.7所示。

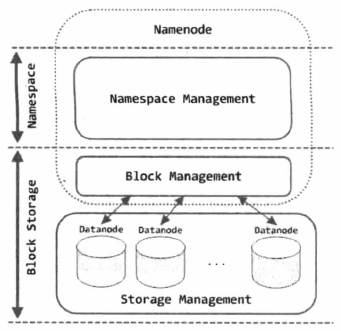


图3.7 核心模块逻辑架构

Datanode在Namenode上注册，并定期向Namenode发送心跳保持连接，Namenode响应数据块操作指令给Datanode如复制、删除等。Namenode和Datanode共同完成了HDFS的数据块存储管理。

##### 3.2.3.1 Namenode子模块

Namenode是HDFS中的主节点，主要负责文件系统目录的管理、数据块以及数据节点的管理、租约管理和缓存管理等。该子模块下主要包括INode相关类、Feature相关类、Secondary Namenode等。

1. **INode相关类**

HDFS文件系统中的INode类似于Linux系统中的inode，其保存了文件的信息如类型、权限、所有者标识等；除此之外，INode还保存了数据块索引，指向了文件或目录所保存在的磁盘位置。INode相关类的类图如下图3.8所示。

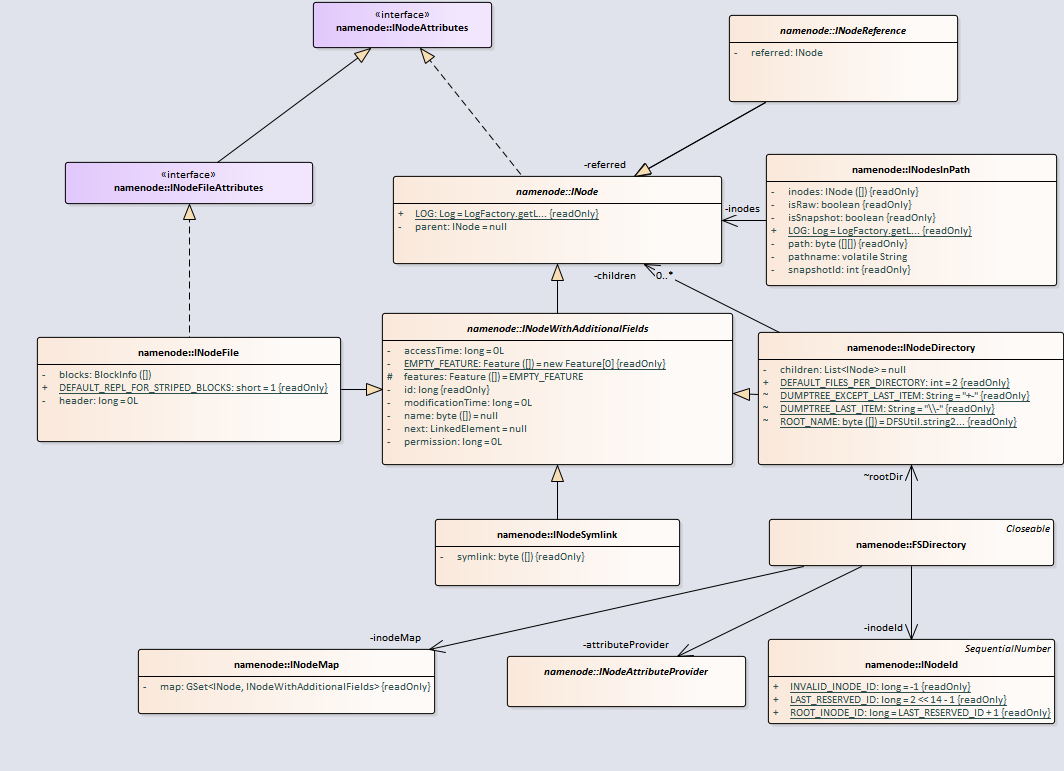


图3.8 INode类图

1. 体系结构风格

INode子模块采用了面向对象的体系结构风格，该模块用INodeFile类抽象为一个文件，并且将文件的各个功能分割为单独的对象，每个对象包含自己的数据和行为，这些对象之间存在着很低的耦合关系，便于对文件的各个属性或功能进行修改而不会影响文件的其他属性。

1. 策略/战术

INode模块采用了可修改性战术，由于INode相关类之间的低耦合性，使得模块的的局部化修改不会产生连锁反应，影响其他模块。

1. 采用的设计模式

INode子模块使用了策略模式，实现了代码复用。并且由于文件具有统一的格式，所以不会存在不适合某个子类的行为被继承下来，使用策略模式非常合适。

1. 技术分析

INode类是一个抽象类，保存了HDFS目录和文件的共同信息，包括父节点的INode节点引用、文件/目录名、访问权限、修改时间、访问时间、路径名等。INode实现了INodeAttributes接口，有四个子类实现了INode类的抽象接口如表3.2所示。

**表3.2 INode相关类**

|  |  |
| --- | --- |
| 类名 | 作用 |
| INodeReference | 处理文件或目录的访问路径 |
| INodesInPath | 实现了路径的存储与访问 |
| INodeDirectory | 保存了HDFS文件系统的目录 |
| INodeWithAdditionalFields | 实现了其他抽象方法 |

在HDFS中，我们使用INodeFile抽象成一个文件，INodeFile继承INodeWithAdditionalFields类，并实现了INodeFileAttributes接口，INodeFile类保存了文件头的header字段和文件对应的数据块信息block字段。

1. **Feature相关类**

Feature描述了INode的特性，INode抽象类中定义了Feature接口，INode的所有特性都要实现这个接口。Feature相关类的类图如图3.9所示。

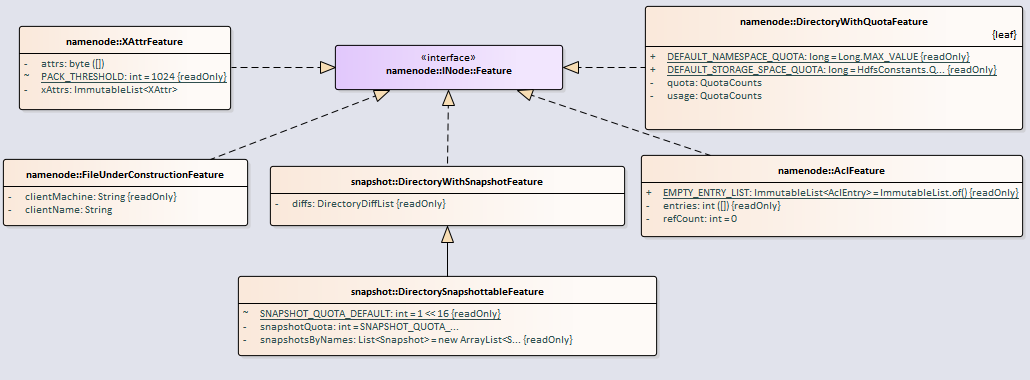


图3.9 Feature相关类类图

Feature子模块的体系架构分析如下：

1. **体系架构风格**

Feature子模块中每个特性所代表的类都实现了INode中的Feature接口，再依次向外扩展，体现了体系架构风格中的面向对象风格。每个特性都有自己的属性和方法，将其封装起来，对其他特性隐藏它的表示，改变一个特性的表示不影响其他的特性。也可以向一个文件添加多种特性。

1. **策略/战术**

Feature子模块也使用了可修改性战术，各特性之间彼此低耦合，在对某个特性进行局部化修改时，不会影响其他特性。

1. **设计模式**

Feature子模块整体采用了策略模式，在向文件添加Feature类的相关特性时使用了装饰者模式。因为一个文件可以具有多种特性，使用装饰者模式可以很方便地为文件动态添加特性，具有很高的灵活性。但与此同时为排错带来了困难。

1. **技术分析**

由Feature相关类的类图，INode的特性主要包括6种如下表3.3所示。

**表3.3 Feature相关类**

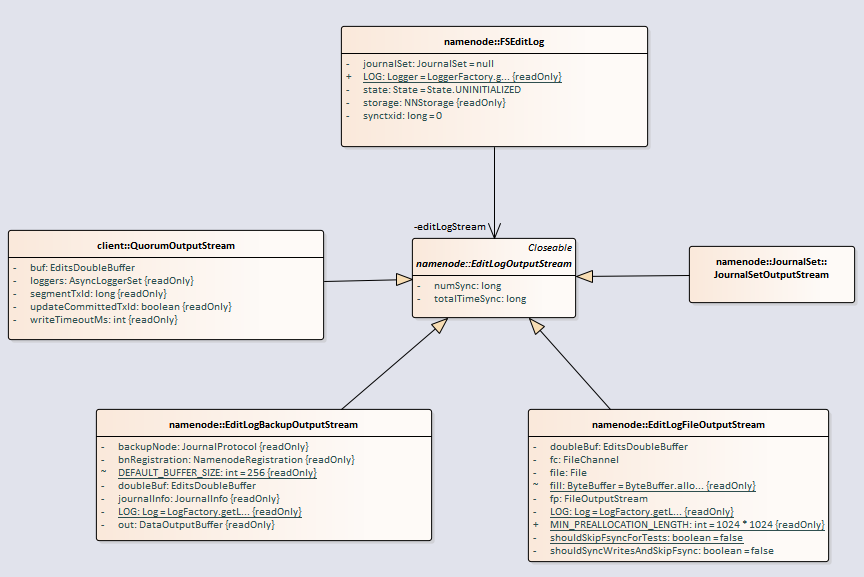
|  |  |
| --- | --- |
| 类名 | 作用 |
| XAttrFeature | 支持文件系统扩展属性的特性 |
| FileUnderConstructionFeature | 正在构建的文件特性 |
| DirectoryWithSnapshotFeature | 带有快照的目录特性 |
| DirectorySnapshottableFeature | 可添加快照的目录特性 |
| AclFeature | 安全特性 |
| DirectoryWithQuotaFeature | 支持磁盘配额的目录特性 |

DirectorySnapshottableFeature类继承自DirectoryWithSnapshotFeature类。

1. **Secondary Namenode**

Secondary Namenode是通过合并editlog和fsimage来帮助Namenode保持最新的快照，类似于Windows的恢复点。

HDFS使用FSEditLog来管理日志文件editlog，客户端所有写操作都会记录到editlog中，随着Namenode的运行实时更新。FSEditLog的editLogStream字段负责记录操作到editlog文件中，该字段是EditLogOutputStream抽象类类型的，该抽象类定义了向磁盘上写editlog文件的接口。FSEditLog输出流相关类的类图如下图3.10所示。

图3.10 FSEditLog输出流类图

与EditLogOutputStream相对应，EditLogInputStream类抽象了从磁盘等持久化存储上读取editlog文件的相关接口。其类图结构与EditLogOutputStream相似，不同的存储系统有与之对应的子类。FSEditLog输入流相关类的类图如下图3.11所示。

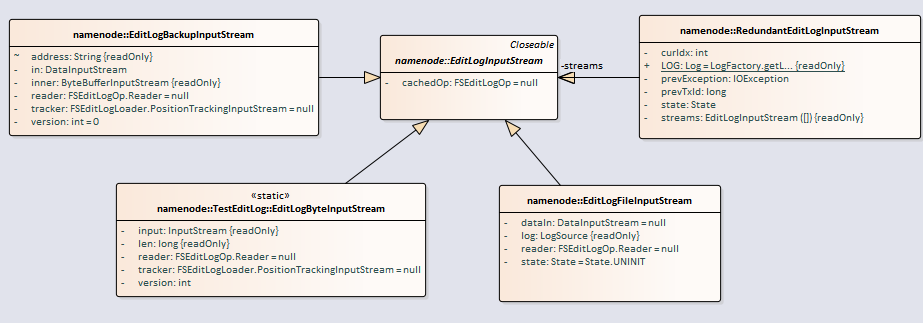


图3.11 FSEditLog输入流类图

Namenode将命名空间信息记录在一个叫做fsimage（命名空间镜像）的二进制文件中，fsimage将文件系统目录中每个文件、目录的信息保存为一条记录，包括该文件、目录的名称、大小、用户等信息。fsimage中的信息与Namenode保持同步。HDFS会定期将FSEditLog和fsimage合并，以保持fsimage和Namenode内存中记录的命名空间完全同步。刚才我们说到，FSEditLog类负责操作editlog文件，与此对应，FSImage类用来管理fsimage文件，FSImage类可以将命名空间保存到fsimage文件中，并且能加载fsimage和editlog文件。

FSEditLog与fsimage的合并流程如图3.12所示。

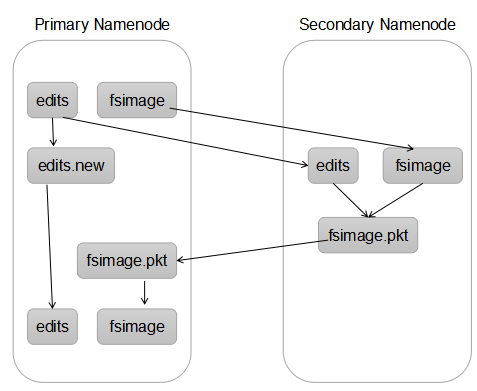


图3.12 FSEditLog与fsimage合并流程图

第一步HDFS将更新记录写入到新文件edits.new。

第二步将fsimage和editlog发送至Secondary Namenode。

第三步将两者合并，生成一个新文件fsimage.ckpt。

第四步将生成的fsimage.ckpt发送至Namenode。

第五步Namenode将fsimage.ckpt重命名为fsimage，edits.new为edits。

1. **体系架构风格**

FSEditLog相关类用于操作editlog文件，其输入流和输出流类为抽象类，对于不同的存储系统，有与之相对的子类对该抽象父类进行扩充。FSImage类用于对fsimage文件进行管理。在此模块，没有体现出明显的体系架构风格。

1. **策略/战术**

本模块采用了可用性战术，editlog文件负责记录客户端的操作，fsimage用于记录Namenode的命名空间信息，两者定期结合，同步Namenode的状态，这使得当Namenode发生故障时，可从fsimage恢复Namenode的状态。

1. **设计模式**

由图3.9与图3.10可以看出，两图均只有一个抽象父类和多个子类，代表在不同的场景里，该行为具有不同的实现算法。实际地，FSEditLog类对于操作输入流与输出流均采用了策略模式，对待不同的存储系统扩展了不同方法对其进行处理。

1. **技术分析**

由上图3.11可见，FSEditLog中的editLogStream变量为EditLogOutputStream类型，该类型有4个子类如下表3.4所示。

**表3.4 EditLogOutputStream子类**

|  |  |
| --- | --- |
| 类名 | 作用 |
| EditLogFileOutputStream | 抽象了本地文件系统上editlog文件的输出流 |
| EditLogBackupOutputStream | 抽象了备用节点editlog文件的输出流 |
| QuorumOutputStream | 抽象了Quorum集群上editlog文件的输出流 |
| JournalSetOutputStream | 执行聚合的写入操作 |

由图3.12，输入流EditLogInputStream共有4个子类如下表3.5所示。

**表3.5 EditLogInputStream子类**

|  |  |
| --- | --- |
| 类名 | 作用 |
| EditLogFileInputStream | 抽象了本地文件系统上editlog文件的输入流 |
| EditLogBackupInputStream | 抽象了备用节点editlog文件的输入流 |
| EditLogByteInputStream | 抽象本地文件系统上editlog文件的输入字节流 |
| RedundantEditLogInputStream | 抽象接口上editlog文件的输入流 |

##### 3.2.3.2 Datanode子模块

Datanode以存储数据块的形式存储文件，同时Datanode会响应客户端的读写请求和Namenode的指令如创建、删除、复制等操作。

1. **体系结构风格**

Datanode的逻辑架构图如图3.13所示。Datanode采用分层架构，共分为三层：服务层、逻辑层和数据层。该架构类似于三层C/S架构，只不过表示层改为服务层，从应用的用户接口改为与Namenode或客户端的交互接口。

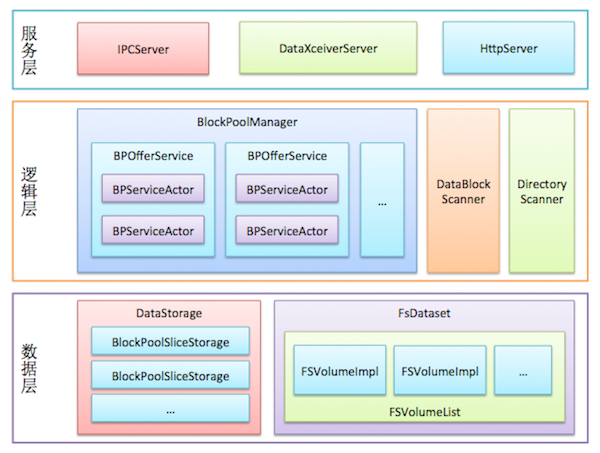


图3.13 Datanode逻辑架构图

该架构将数据存储这一功能划分为三层，每一层只解决问题的一部分，通过各层的协作提供整体的解决方案。大的问题被分解为一系列小问题，降低了问题的规模与复杂度。同时分层架构具有良好的可扩展性，为系统的版本更新提供了一个灵活的框架，提高代码重用。由于不同的层次负责不同的功能，层与层之间的耦合度降低，架构易于维护。

在此三层架构中其中数据层负责在本地磁盘存储数据块的部分抽象，该部分主要包括DataStorage（数据存储）和FsDataset（文件系统数据集）；逻辑层负责执行HDFS逻辑的部分抽象，包括BlockPoolManager（管理池块的接口类）、DataBlockScanner（检查数据块）、DirectoryScanner（更新数据）；服务层用于支持其他节点于Datanode通信。

1. **策略/战术**

每个Datanode都会初始化一个数据块扫描器周期性的验证Datanode上存储的所有数据的正确性，并把发现损坏的数据块报告Namenode。此扫描器可视为数据块的健康监视器，及时检测到数据错误，这一定程度上保证了系统的可用性，为系统的可用性战术，防止错误发展成故障。

1. **设计模式**

在获取FsDataset对象时，使用了工厂模式如图3.14所示。

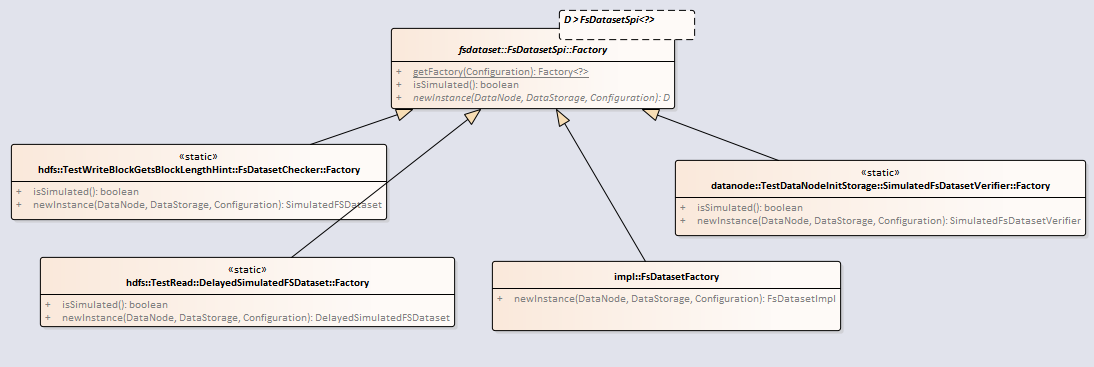


图3.14 FsDataSet工厂模式

1. **技术分析**

在数据层，DataStorage负责管理与组织磁盘存储目录，DataStorage的继承关系如图3.15所示。

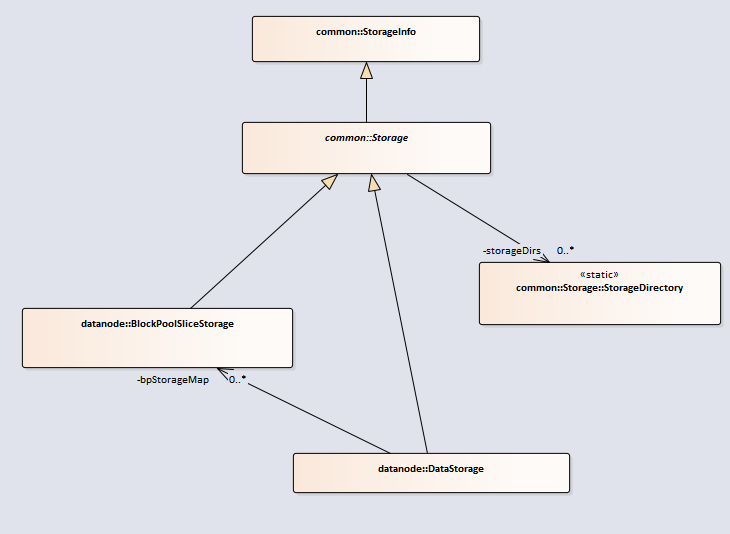


图3.15 DataStorage类的继承关系

DataStorage的父类为Storage，Storage是一个抽象类，它继承StorageInfo和StorageDirectory。StorageInfo描述存储的基本信息。Storage为Datanode、Namenode提供抽象的存储服务。其功能如表3.6所示。

**表3.6 DataStorage功能**

|  |  |
| --- | --- |
| 函数 | 功能 |
| RecoverTransitionRead() | 在所有存储目录上初始化指定块池 |
| doTransition() | 恢复块池的异常状态 |
| doUpgrade() | 对单个存储目录下的指定块池目录进行升级 |
| doRollback() | 对单个存储目录下的指定块池目录进行回滚操作 |
| doFinalize() | 执行升级提交操作 |

FsDatasetImpl相关类负责管理与组织数据块及其元数据文件，FsDatasetImpl相关类的类图如图3.16所示。

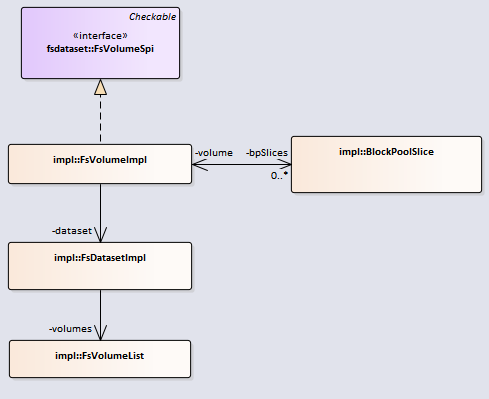


图3.16 FdDatasetImpl相关类类图

BlockPoolSlice：管理一个指定块池在一个指定存储目录下的所有数据块。

FsVolumeImpl：管理Datanode一个存储目录下的所有数据块。

FsVolumeList：Datanode可以定义多个存储目录，每个存储目录下的数据块使用一个FsVolumeImpl对象管理，FsVolumeList类保存Datanode上所有FsVolumeImpl对象。

##### 3.2.3.3 数据块管理模块

BlockManager类保存并且管理HDFS集群中所有数据块的元数据，负责统一调度Datanode的存储信息。其主要成员包括：blocksMap、DatanodeManager和DocommissionManager。其中blocksMap管理着Namenode上数据块的元数据，Namenode会在Datanode启动后建立数据块与对应Datanode节点之间的对应关系，并将其保存在blocksMap中；DatanodeManager负责接受管理来自DataNode的消息；DocommissionManager负责管理要检修的节点信息。

1. **体系架构风格**

本模块主要应用于Datanode和Namenode之间的通信。其架构是一个单向的通信模型，所有信息都是Datanode主动向Namenode发送。Datanode对Namenode节点会建立一个BPServiceActor线程负责同Namenode建立连接，定时发送心跳和存储块信息，其模型图如图3.17所示。

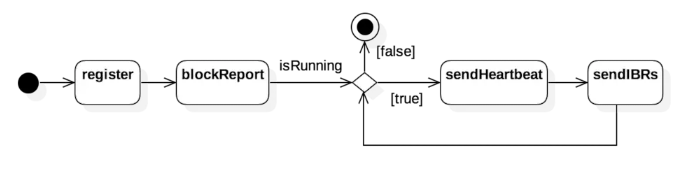


图3.17 BPServiceActor模型图

当启动BPServiceActor后，首先Namenode注册，然后进行一次完整的block信息上报，之后定时发送心跳。

1. **策略/战术**

每次DataNode发送心跳后，BlockManager中的HeartbeatManager都会更新其最近访问时间，同时HeartbeatManager还有一个HeartbeatManager,Monitor线程通过轮询检测是否有Datanode过期。一旦有Datanode过期，则将其从DatanodeManager中删除。除此之外，BlockManager还有几个独立线程负责周期性遍历数据情况，优化节点间Block的分布情况。此为该模块的性能战术，删除过期的Datanode可清理内存，防止浪费系统资源，可提升资源的可用性。

Block容灾备份保证了模块的可用性。为了避免由于节点故障导致的数据丢失，HDFS采取多地备份策略。其由两部分组成，一是在文件首次创建时，通过数据管道同时在多个Datanode中创建Block，保证创建时就拥有多个版本。第二是在节点故障时，通过复制现有副本文件到另一个节点中，保证了副本的冗余度。

1. **设计模式**

BlockManager模块在block放置策略处使用了策略者模式，首先定义一个默认的放置策略，然后根据不同的场景，子类对默认的放置策略进行修改，形成不同的处理方式如图3.18所示。

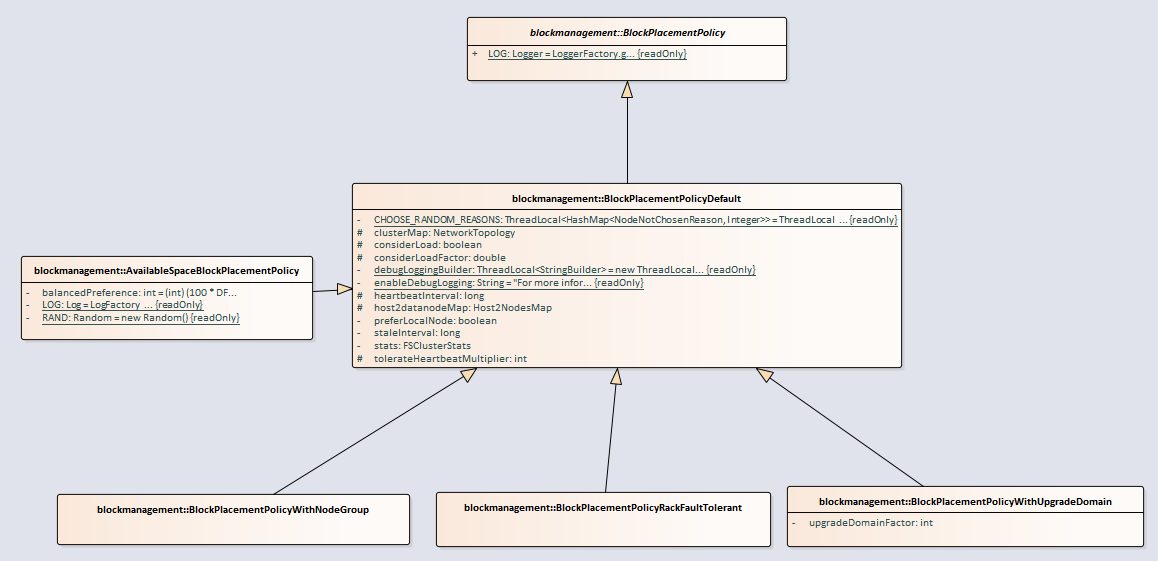


图3.18 BlockPlacementPolicy类图

1. **技术分析**

BlockManager在初始化时启动了两个线程：pendingReplications和replicationThread，这两个线程共同完成了BlockManager的核心逻辑。

pendingReplications由PendingReplicationMonitor类实现，此线程会遍历所有数据块复制任务，将超过指定时间没有确定的复制请求加入超时队列。ReplicationThread由ReplicationMonitor实现，ReplicationMonitor为一个线程类，该线程会周期性的触发数据块的复制和删除任务，然后将超时队列中保存的任务重新加入队列。

#### 3.2.3 客户端模块层

HDFS核心模块中介绍了包括DataNode、NameNode的实现细节。HDFS目前提供了三个客户端接口[13]：DistributedFileSystem、FsShell和DFSAdmin。DistributedFileSystem为用户开发基于HDFS的应用程序提供了API；FsShell工具使用户可以通过HDFS Shell命令执行常见的文件系统操作，如创建文件、删除文件、创建目录等；DFSAdmin则向系统管理员提供了管理HDFS的工具，如执行升级、管理安全模式等操作。

##### 3.2.3.1 软件体系结构风格

图3.19为客户端模块中的部分类图，图中省略了类的部分成员及函数。

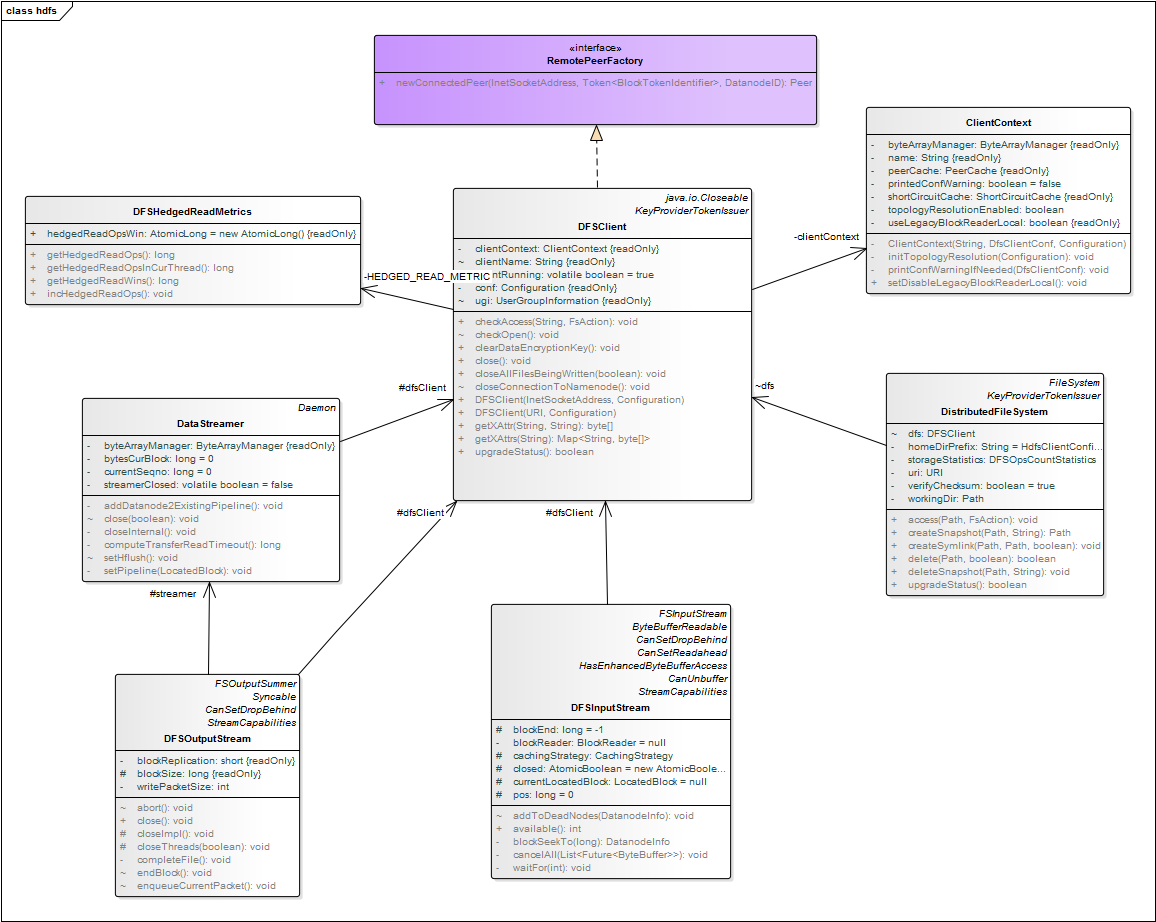


图3.19 客户端模块的部分类图

DFSClient类封装了HDFS复杂的交互逻辑，对外提供了简单的接口。客户端模块主要分析DFSClient的逻辑进行分析。DistributedFileSystem、FsShell和DFSAdmin都是通过直接或者间接地持有DFSClient对象的引用，然后调用DFSClient提供的接口方法对HDFS进行管理和操作的[23]。

##### 3.2.3.2 策略/战术

DFSClient是一个真正实现了分布式文件系统客户端的类，是用户使用HDFS各项功能的起点。DFSClient会连接到HDFS，对外提供管理文件/目录、读写文件以及管理与配置HDFS系统等功能。DFSClient封装了HDFS复杂的交互逻辑，对外提供了简单的接口。在实现该封装中，采用了易用性战术。

易用性与用户完成任务的难易程度以及系统为用户提供的支持种类有关。有两种类型的战术支持易用性，第一类针对的是系统运行期间支持用户的战术；第二类基于用户接口设计的迭代特性，它在设计时支持接口开发人员。DFSClient对外提供的复杂接口被抽象封装为四类：DFSClient的构造方法和关闭方法、管理与配置文件系统相关方法、操作HDFS文件与目录方法、读写HDFS文件方法。借助DFSClient的封装，DistributedFileSystem、FsShell和DFSAdmin才能分别向用户提供对HDFS文件系统的操作和管理。

DFSClient的设计采用了可测试性战术。该战术目标是允许在完成一个软件开发的增量后，轻松地对软件进行测试。其中分为两类用于测试的战术，一类为提供输入并捕获输出；另一类为内部监视。在DFSClient中封装了多数操作均与存储在HDFS文件系统中的文件相关，用户进行输入后，很容易地可以测试返回结果是否正确。比如通过客户端上传一个新的文件到HDFS中，借助fs –ls命令可以判断该文件是否成功存储至文件系统。在DFSClient所封装的执行操作中，有额外进行log日志的类在记录用户的操作行为，以及底层数据的返回结果，可以借助内部日志信息来支持开发的可测试性。

##### 3.2.3.3 设计模式

在Client进行初始化时需要读取配置项等文件进行构造，在此阶段使用了建造者模式。BlockReaderFactory及其相关部分类图如图3.20所示。

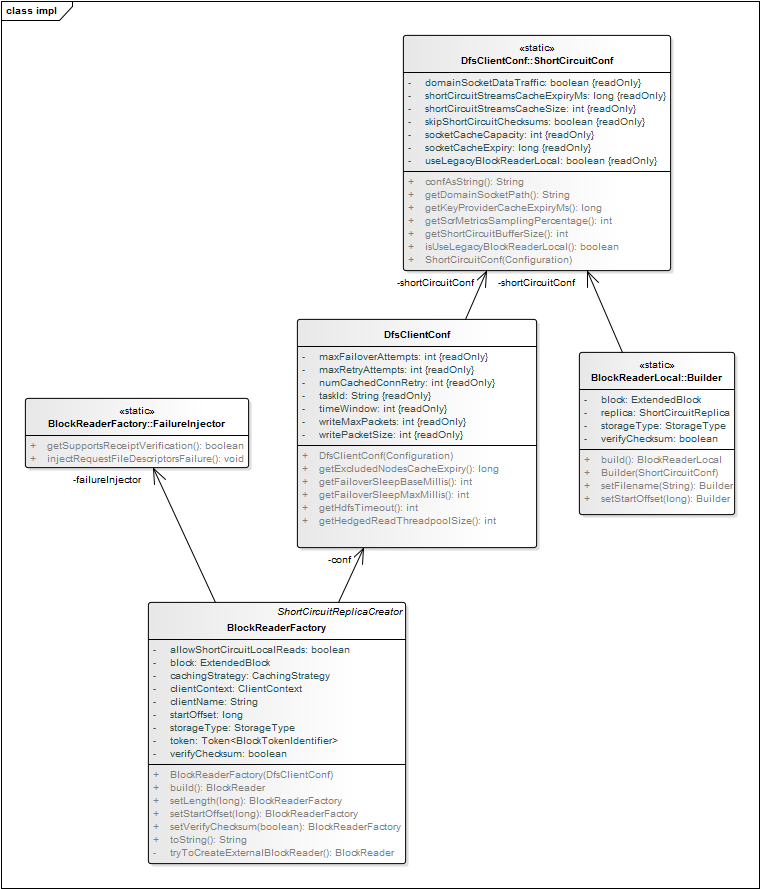


图3.20 客户端模块中的建造者模式

BlockReader类的创建代码调用BlockReaderFactory类中的build函数，其中支持四种方式进行构造：getLegacyBlockReaderLocal()、getBlockReaderLocal()、getRemoteBlockReaderFromDomain()、getRemoteBlockReaderFromTcp()。

在进行文件读取时采用策略模式。ReaderStrategy部分类图如图3.21所示。

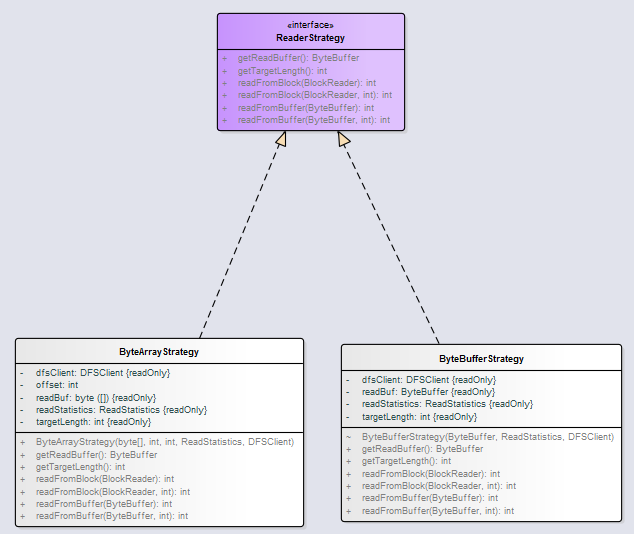


图3.21 客户端模块中的策略模式

在读取文件时，支持不同的读取策略，此处使用常见的策略模式。

客户端使用装饰者模式，DataStreammer的部分类图如图3.22所示。

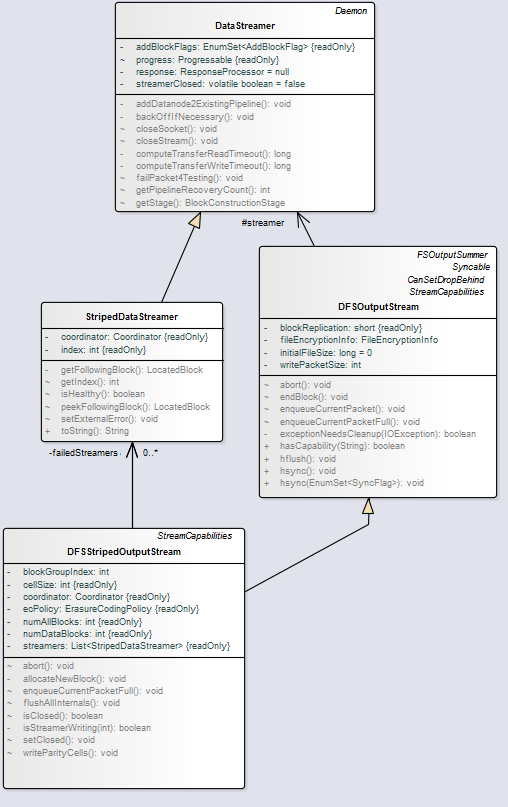


图3.22 客户端模块中的装饰者模式

##### 3.2.3.4 技术分析

DFSClient对底层文件的操作进行了封装，因此该类的构造流程十分重要。在DFSClient中提供了5种重载的构造方法，分别为：DFSClient(Configuration)、DFSClient(InetSocketAddress, Configuration)、DFSClient(URI, ClientProtocol, Configuration, FileSystem.Statistics)、DFSClient(URI, Configuration)、DFSClient(URI, Configuration, FileSystem.Statistics)。DFSClient类的构造方法主要完成以下两个功能。

1．读入配置文件，初始化成员变量。相关初始化成员信息见表3.7所示。

**表3.7 DFSClient初始化成员信息表**

|  |  |
| --- | --- |
| 变量名 | 备注信息 |
| conf | HDFS配置信息 |
| stats | Client状态统计信息，包括Client读、写字节数等 |
| dtpReplaceDatanodeOnFailure | 当Client读写数据时，如果Datanode出现故障，是否进行Datanode替换的策略 |
| localInterfaceAddrs | 本地接口地址 |
| readahead | 预读取字节数 |
| readDropBehind | 读取数据后，是否立即从操作系统缓冲区中删除 |
| writeDropBehind | 写数据后，是否立即从操作系统缓冲区删除 |
| hedgedReadThresholdMillis | 保存了触发”hedgedread”机制的时长，当Client发现一个数据块读取操作太慢时，那么Client会启动另一个并发操作读取数据块的另一个副本，之后Client会返回先完成读取副本的数据。 |

2． 获取NameNode的RPCProxy引用，提供DFSClient调研NameNodeRPDC方法。

HDFS管理员可以借助DFSAdmin工具管理与配置HDFS，DFSAdmin是通过持有DistributedFileSystem对象的引用，然后进一步调用DFSClient类提供的方法执行管理与配置的。DFSAdmin通过传递不同的参数来实现对HDFS系统的管理。表3.8为DFSClient文件系统管理与调用方法对应表。

**表3.8 DFSClient文件系统管理与调用方法对应表**

|  |  |  |  |
| --- | --- | --- | --- |
| DFSAdmin参数 | DFSClient | ClientControl | 备注信息 |
| -report | datanodeReoprt() | getDatanodeReport() | 获取当前集群中所有的Datanode的信息 |
| -safemode | setSafeMode() | setSafeMode() | 进入、离开或者查询安全模式。处于安全模式中的HDFS，命名空间是只读的 |
| -allowSnapshot | allowSnapshot() | allowSnapshot() | 开启指定目录的快照功能 |
| -disallowSnapshot | disallowSnapshot() | disallowSnapshot() | 关闭指定目录的快照功能 |
| -saveNamespace | saveNamespace() | saveNamespace() | 保存当前的NameNode元素到fsimage |
| -rollEdits | rollEdits() | rollEdits() | 提交当前的edits\_inprogress文件 |
| -restoreFailedStorage | restoreFailedStorage() | restoreFailedStorage() | 开启或者关闭失败存储的回复 |
| -refreshNodes | refreshNodes() | refreshNodes() | 刷新hosts以及exclude文件 |
| -setSpaceQuota | setSpaceQuota() | setSpaceQuota() | 设置磁盘空间配额 |

HDFS的写流程如图3.23所示[22]。

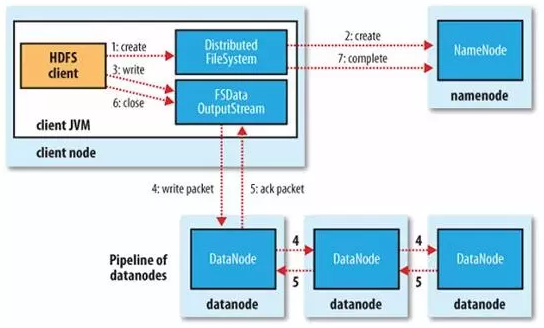


图3.23 HDFS的写流程

HDFS的读流程如图3.24所示[22]。

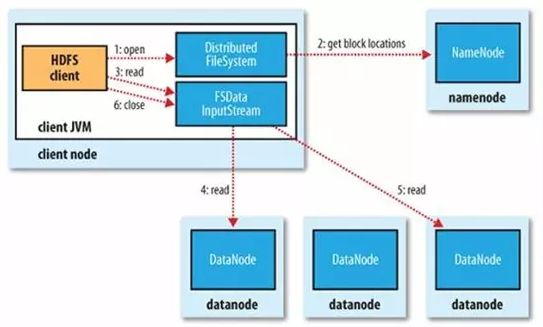


图3.24 HDFS的读流程

#### 3.2.4 工具及安全模块层

包括hdfs.tools和hdfs.server.balancer，这两个包提供查询HDFS状态信息工具dfsadmin、文件系统检查工具fsck和HDFS均衡器balancer（通过start-balancer.sh启动）的实现。图3.25为工具模块的部分类图

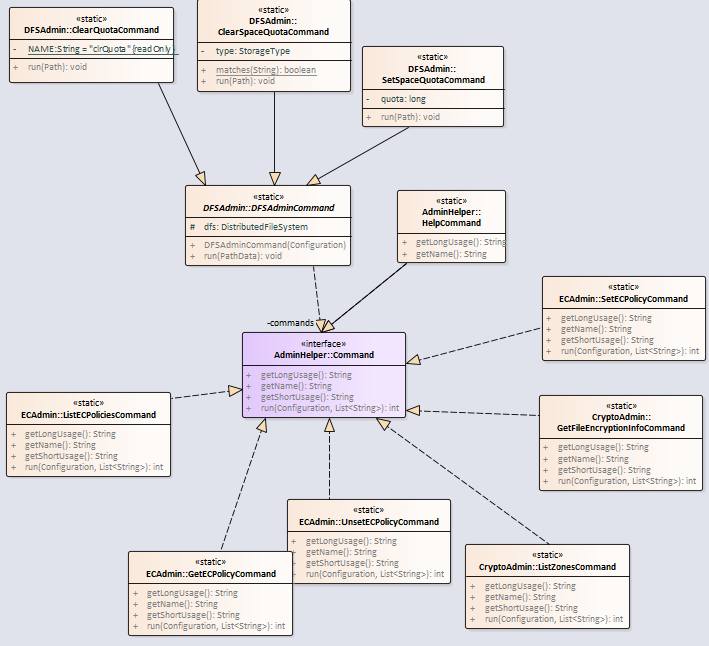


图3.25 工具模块的部分类图

首先我们使用enterprise architect工具生成这个包的类图（注意：为了便于展示，在类图的属性方法项目中没有把所有的内容列举出来），这个包是一个工具集的代码实现，可以看到所有的command都继承自Adminhelper这个base class。这样做的好处是一方面便于代码的管理，对于一些公共的变量的属性修改，没必要每次遇到都改一遍，直接在base class 上进行操作就可以。另一方面，这在逻辑上也符合面向对象程序的观点，虽然不同的command 完成的任务不同，但是他们多是属于同一类的，那么就应该先写一个基类。

##### 3.2.4.1 软件体系结构风格

本模块采用了面向对象的体系结构风格，同时在是实现过程中，形成了层次结构。子类可以调用父类的方法并且同不同的兄弟之间共享父节点的方法和属性。他带来的优点也是显而易见的，支持重用，支持功能增强，因为每一层至多和相邻的上下层交互，因此功能的改变最多影响相邻的上下层。

##### 策略/战术

可修改性战术：防止发生连锁反应，采用了信息隐蔽，同一层次的兄弟姐妹彼此间是隐藏的，不存在直接耦合，对其中一个节点进行修改最多只会影响到其子节点；局部化修改，上图列出来的类的分布中，为了节省空间，绘图美观，有很多子类并没有呈现，无独有偶，这些子类都是专门为了完成某个人物而存在的，这样对于不同的需求，都能对应到相应的子类上，同样当需求发生变化，进行代码的重写时也可以很强的针对性。

##### 3.2.4.3 设计模式

本模块采用了面向对象的体系结构风格，同时在是实现过程中，形成了层次结构。子类可以调用父类的方法并且同不同的兄弟之间共享父节点的方法和属性。他带来的优点也是显而易见的，支持重用，支持功能增强，因为每一层至多和相邻的上下层交互，因此功能的改变最多影响相邻的上下层。使用到迭代器模式，相关类图如图3.26所示。

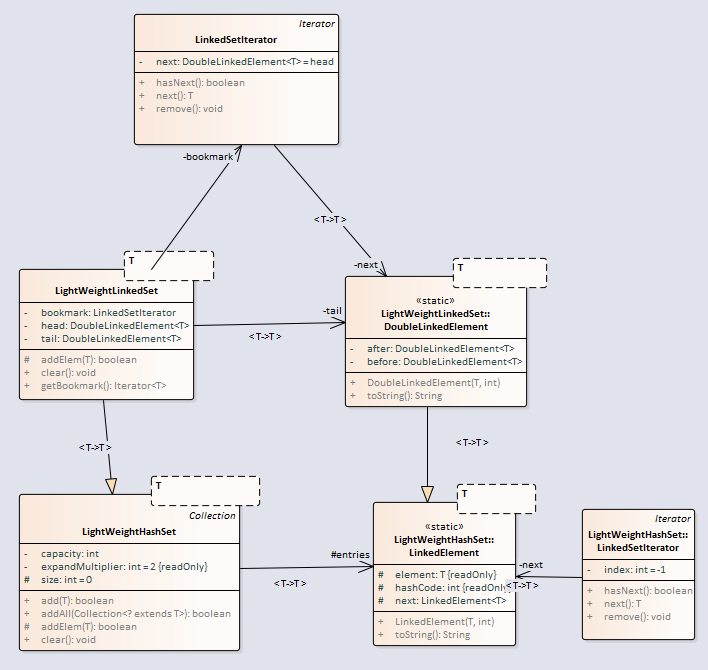


图3.26 工具及安全模式中采用的迭代器模式类图

在应用模块层采用了访问者模式，其部分类图如图3.27所示。

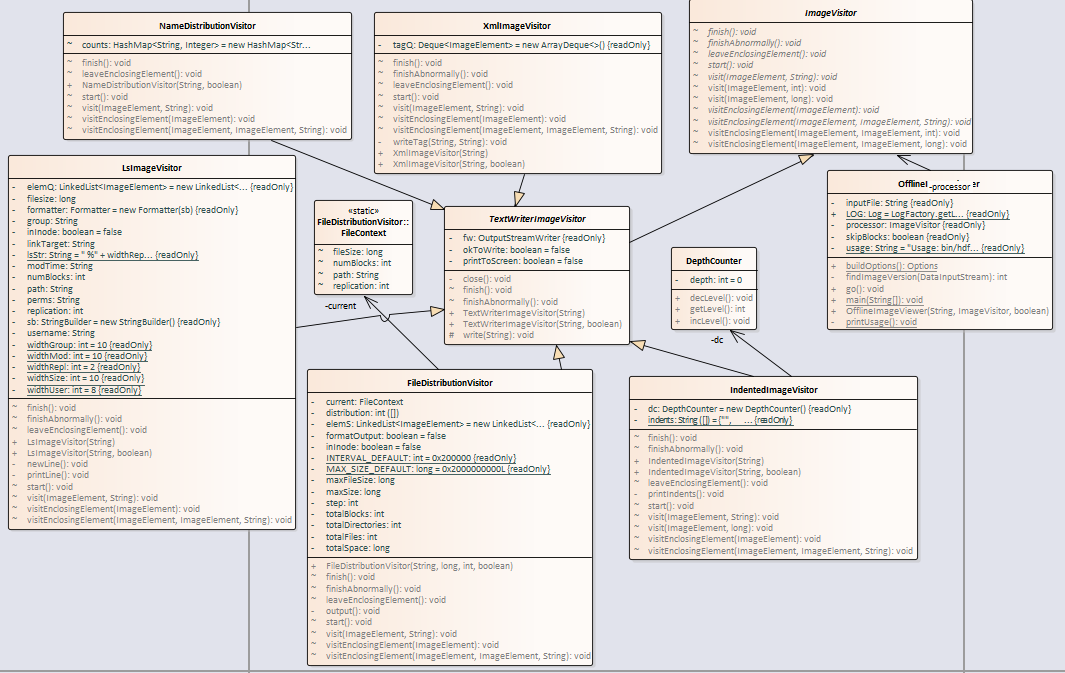


图3.27 工具及安全模式中采用的访问者模式类图

ImageVisitor 类定义了一系列抽象的方法，TextWriterImageVisitor 继承了该接口，又定义了一系列的visitor 继承TextWriterImageVisitor。这是设计模式中的访问者模式，它改变了元素类的执行算法。通过这种方式，元素的执行算法可以随着访问者改变而改变。这种类型的设计模式属于行为型模式。根据模式，元素对象已接受访问者对象，这样访问者对象就可以处理元素对象上的操作。

##### 3.2.4.4 技术分析

代码的重用与分离是软件工程中很重要的一环，而作为应用模块层，不同的操作只是上下文不同、所需的参数不同，因此代码的复用很有必要，在这一模块，通过定义抽象接口，抽象类的方法，实现了不同操作共享同一数据结构，在不同的上下文中对数据进行不同的操作。

主要使用了下面的策略：

1、模块功能单一化，模块功能要单一，在模块中要使用某些有别的模块的代码产生的数据时不要直接调用其它模块中定义的函数，而是利用参数的形式将数据从外部传递过来，从而使模块尽量减少对其他模块的依赖，保持功能单一。

2、头文件尽量减少包含其它头文件，要注意这样一个原则：尽量使用指向对象的指针或引用，而不要直接使用对象，这样做有两个好处：一是提高了程序的执行效率(函数调用避免了对实参对象的拷贝)，二是有利于代码的重用(解除头文件之间的依赖关系)。

3、隐藏实现细节：头文件只提供给使用者最低限度的必要信息，而将任何不需要使用者关心的实现细节隐藏起来。4合理设计接口。接口与实现是一个相对的概念，接口应与实现分离，接口本身反映了系统设计人员对系统的抽象理解。接口是一个类或模块对外交换信息的窗口，这个窗口既可以实现外部对类或模块的调用，同时也可以实现类或模块对外部功能的调用。接口可以表现为一个函数指针、一个虚函数、或者一个抽象类。

### 3.3 结构模型分析

随着信息技术深入生产生活，信息系统的数据量以前所未有的速度迅速增长，存储系统不断膨胀。云计算延伸和发展出的云存储系统正逐步成为网站和网络服务方用于存放管理自己数据信息的首要选择。HDFS作为分布式的存储系统，非常适应云计算背景下的商业与应用环境，是当前大批量数据存储的首要选择方案。

云存储是指通过集群技术、网格技术以及分布式文件系统等技术，将网络中大量不同类型的存储设备有效结合起来进行协同工作，共同对外提供业务访问服务和数据存储功能的一个云计算系统。云存储的架构模型如图3.28所示[25]。

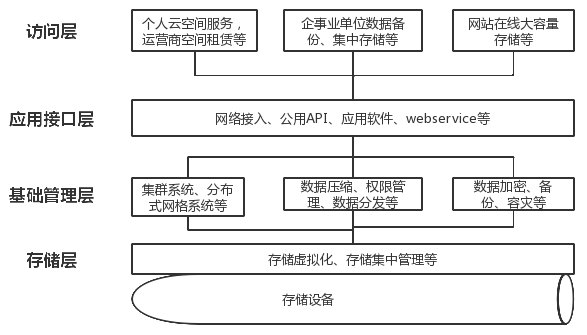


图3.28 云存储架构图

HDFS采用master/slave架构，一个HDFS集群是由一个Namenode和两个以上的Datanodes组成。Namenode是一个中心服务器，负责管理文件系统的名字空间（namespace）以及客户端对文件的访问。集群中的Datanode一般是一个节点一个，负责管理它所在节点上的存储。HDFS暴露了文件系统的名字空间，用户能够以文件的形式在上面存储数据。从内部看，一个文件其实被分成一个或多个数据块，这些块存储在一组Datanode上。Namenode执行文件系统的名字空间操作，比如打开、关闭、重命名文件或目录。它也负责确定数据块到具体Datanode节点的映射。Datanode负责处理文件系统客户端的读写请求。在Namenode的统一调度下进行数据块的创建、删除和复制。

## 4. 设计特色分析

HDFS是一个高度容错性的系统，适合部署在廉价的机器上。HDFS能提供高吞吐量的数据访问，非常适合大规模数据集上的应用。此外HDFS提供高吞吐量来访问应用程序的数据，适合那些有着超大数据集的应用程序。HDFS放宽了POSIX的要求，这样可以实现流式访问文件系统中的数据。除此之外，HDFS还是易于扩展的分布式文件系统。

### 4.1 数据高容错性

#### 4.1.1 特色介绍

##### 4.1.1.1 副本机制

HDFS对每个文件的数据进行分块存储，其中每个数据块保存多个副本，这些副本分布在不同的机器节点上，副本存放策略控制这些副本的存放，保证了文件存储的可靠性和高性能，其存放策略如图4.1所示。

第一副本：存放在离客户端最近机架的任意节点，若客户端为本机，则存放在本机上。

第二副本：存放在与第一个副本不同机架的任意节点上。

第三副本：存放在与第二个副本相同机架的不同节点上。

若存在更多副本，则随意挑选节点存放。

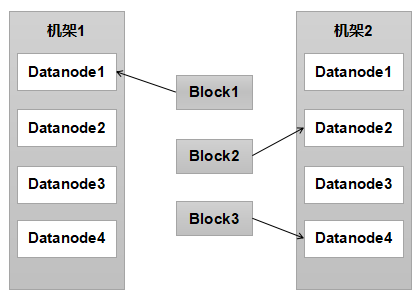


图4.1 副本存放策略

##### 4.1.1.2 心跳机制

HDFS是主从结构的，为了实时判断Datanode是否存活，需要建立心跳机制，使Datanode定时的向Namenode发送心跳包，报告自己的存活状态，来维持长连接。此心跳包的内容为自己存储的block列表信息。当由于网络故障之类的原因，导致Datanode发出的心跳包没有被Namenode正常收到的时候，Namenode就不会将任何新的IO操作派发给那个Datanode，该Datanode上的数据被认为是无效的，因此Namenode会检测是否有文件block的副本数目小于设置值，如果小于就自动开始复制新的副本并分发到其他Datanode节点。

##### 4.1.1.3 一致性检验

HDFS采用的一致性检验：检测文件block的完整性，HDFS会记录每个新创建的文件的所有block的校验和。当以后检索这些文件的时候，从某个节点获取block，会首先确认校验和是否一致，如果不一致，会从其他Datanode节点上获取该block的副本。

##### 1.1.1.4 安全模式

安全模式是HDFS的一种保护机制，用于保证集群中数据块的安全，在此模式中，只能查看元数据信息，不能执行任何修改元数据信息的操作。若Namenode发现HDFS集群中的数据块丢失比率达到一定比例时，Namenode就会进入安全模式。当HDFS集群启动时会首先进入安全模式，因为Namenode不包含Datanode信息，Datanode会向Namenode通过心跳机制汇报可用的数据块信息，当汇报完成后，就会自动推出安全模式。

##### 4.1.1.5 集群负载均衡

HDFS负载均衡的理想状态为节点均衡、机架均衡和磁盘均衡[24]。由于在HDFS集群中容易出现集器之间磁盘利用率平衡的情况，所以需要对HDFS进行数据的负载均衡调整，即对各个节点上数据的存储分布进行调整，使得每一个Datanode存储的数据与其应将相匹配，即占用率相当。

由于节点的失效或者增加，可能导致数据分布的不均匀，当某个Datanode节点的空闲空间大于一个临界值的时候，HDFS会自动从其他Datanode迁移数据过来。

#### 4.1.2 设计对比

Lustre需要使用相对比较健壮的对象存储服务器，使用简单的分条策略，但并没有在单个结点失效后主动重分布数据的能力。Lustre依赖于存储结点的磁盘阵列来容忍磁盘失效。

PVFS2在存储结点间结点进行数据分条，但没有使用校验和冗余机制以保证系统的可用性。PVFS2虽然使用普通的商用机器，但不能保证磁盘失效后系统运行的正常。

Ceph在实践中，扩容受“容错域”制约，一次只能扩一个“容错域”。因为Ceph没有中心化的元数据结点，导致数据放置策略受之影响。数据放置策略，即一份数据replica，放在哪台机器，哪块硬盘。Ceph因为去中心化，导致容纳数据的PG的位置，会根据crushmap的变化而变化。来了新的机器、硬盘，就要为一些受影响的PG计算新的位置。基于一致性哈希的技术，在扩容时也要面临同样的问题。正因为这个调整，导致Ceph受“容错域”制约。

中心化的HDFS会记录每一个文件对应的每一个数据块的存放位置。这个位置是不会经常变动的，只有在文件新创建；balancer重平衡；有硬盘坏了。中心节点针对损坏硬件上的数据重新放置时才会改变。

#### 4.1.3 需求支撑

在对于存储大规模数据集的场景中，需要有很高的容错性及高吞吐量。数据能否安全地存储，对于上层应用的效果十分重要，其影响结果远比服务器硬件的价值高。HDFS设计为能够在通用硬件上运行的分布式文件系统，其高容错性以及部署的廉价性，使得HDFS能够在大规模数据集上得到广泛的应用。此外异构软硬件平台可以得到很好地迁移，这更推动HDFS作为广泛使用的大数据集处理平台。

### 4.2 大数据存储

HDFS采用了master/slave架构的模式。一般情况下一个HDFS有一个namenode（高可用模式除外）和一定数目的DataNode，其中namenode是主节点，DataNode是从节点。

#### 4.2.1 特色介绍

HDFS适合一次写入，多次读出的场景，适合用来做数据分析，并不适合用来做网盘应用等文件系统。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版本中是128M，老版本中是64M。数据存储中的读取和写入过程，如图4.2所示。

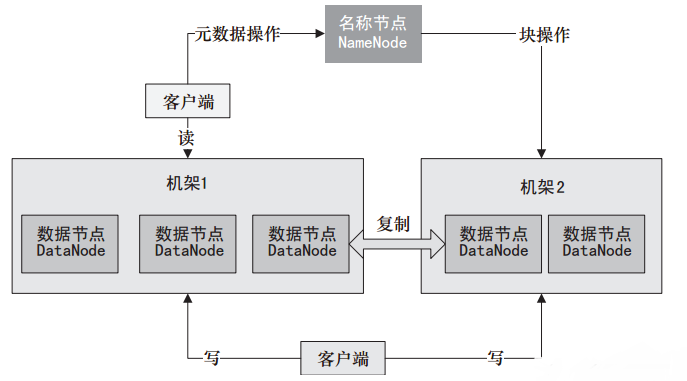


图4.2 数据存储的读取与写入过程

虽然HDFS以block块存储，对于大文件会被切分成很多以块大小的分块进行存储，但是如果文件小于HDFS的块大小，那么该文件的存储不会占用整个块的空间。比如一个10MB的文件，存储虽然在一个128MB的块上，但是该文件实际只用了10MB的空间，而不是128MB的空间。

HDFS适合大数据处理，其处理数据能够达到GB、TB甚至PB级别的数据；能够处理百万规模以上的文件数量；能够处理10K节点的规模。

#### 4.2.2 设计对比

大数据存储技术路线最典型的共有三种[16]：

第一种是采用MPP架构的新型数据库集群，重点面向行业大数据，采用Shared Nothing架构，通过列存储、粗粒度索引等多项大数据处理技术，再结合MPP架构高效的分布式计算模式，完成对分析类应用的支撑，运行环境多为低成本PC Server，具有高性能和高扩展性的特点，在企业分析类应用领域获得极其广泛的应用。

第二种是基于Hadoop的技术扩展和封装，围绕Hadoop衍生出相关的大数据技术，应对传统关系型数据库较难处理的数据和场景，例如针对非结构化数据的存储和计算等，充分利用Hadoop开源的优势，伴随相关技术的不断进步，其应用场景也将逐步扩大，目前最为典型的应用场景就是通过扩展和封装Hadoop来实现对互联网大数据存储、分析的支撑。对于非结构、半结构化数据处理、复杂的ETL流程、复杂的数据挖掘和计算模型，Hadoop平台更擅长。

第三种是大数据一体机，这是一种专为大数据的分析处理而设计的软、硬件结合的产品，由一组集成的服务器、存储设备、操作系统、数据库管理系统以及为数据查询、处理、分析用途而特别预先安装及优化的软件组成，高性能大数据一体机具有良好的稳定性和纵向扩展性。

Hadoop生态圈所提出的方法能够在进行廉价的部署，且满足当前数据的体量。MPP架构在企业新一代的数据仓库和结构化数据分析中，能够有效支撑PB级别的结构化数据分析。大数据一体机则存在价格昂贵，无法快速普遍推广使用的缺点。Hadoop所提出的HDFS在大数据时代是一个创造性工作，未来MPP技术与Hadoop技术相融合，将会为大数据处理注入更为强劲的动力。

#### 4.2.3 需求支撑

如果说现在是大数据时代了，其实是数据来源发生了质的变化。在互联网出现之前，数据主要是人机会话方式产生的，以结构化数据为主。所以大家都需要传统的RDBMS来管理这些数据和应用系统。那时候的数据增长缓慢、系统都比较孤立，用传统数据库基本可以满足各类应用开发。

互联网的出现和快速发展，尤其是移动互联网的发展，加上数码设备的大规模使用，今天数据的主要来源已经不是人机会话了，而是通过设备、服务器、应用自动产生的。传统行业的数据同时也多起来了，这些数据以非结构、半结构化为主，而真正的交易数据量并不大，增长并不快。机器产生的数据正在几何级增长，比如基因数据、各种用户行为数据、定位数据、图片、视频、气象、地震、医疗等等。

在Hadoop生态圈中作为底层数据的存储管理系统，其支撑上方的MapReduce以及其他的系统。借助HDFS对大数据集的优秀处理机制，能够对数据进行更加复杂多元的处理，从而降低了从业者的门槛，节省了开发的时间并大大提高了工作。在数据驱动一切的年代，各个领域对数据的关注度愈加的强烈。各行各业都在不断加强加大在数据方面的研究。对于数据的处理越精细，所带来的影响越深刻，因此HDFS在大数据处理方面具有否认的重要作用。

### 4.3 流式数据访问

在我们的普遍认知中，数据的价值会随着时间的流逝而降低，所以在事件出现后必须尽快对它进行处理，最好是数据出现时便立刻对其进行处理，发生一个事件处理一次，而不是缓存起来进行批处理。

#### 4.3.1 特色介绍

HDFS不能做到低延迟的数据访问，但是HDFS的吞吐量大，Hadoop适用于处理离线数据，不适合处理实时数据。HDFS的数据处理规模比较大，应用一次需要大量的数据，同时这些应用一般都是批量处理，而不是用户交互式处理。应用程序能以流的形式访问数据库。特点就是，数据像流水一样，不是一次过来而是一点一点地“流”过来。而处理流式数据也是一点一点处理。如果是全部收到数据以后再处理，那么延迟会很大，而且在很多场合会消耗大量内存。

流式数据处理具有数据实时持续不断到达、到达次序独立、数据来源众多格式复杂、数据规模大且十分关注存储、注重数据的整体价值而不关注个别数据等特点。

#### 4.3.2 设计对比

流式数据处理与批处理数据流程的对比如表4.1所示。

**表4.1 不同数据处理对比**

|  |  |  |
| --- | --- | --- |
|  | 批处理 | 流处理 |
| 数据范围 | 对数据集中的所有或大部分数据进行查询或处理 | 对滚动时间窗口内的数据或仅对最近的数据记录进行查询或处理 |
| 数据大小 | 大批量数据 | 单条记录或包含几条记录的微批量数据 |
| 性能 | 几分钟至几小时的延迟 | 只需大约几秒或几毫秒的延迟 |
| 分析 | 复杂分析 | 简单的响应函数、聚合和滚动指标 |

以Hadoop为代表的批处理大数据系统需先将数据汇聚成批，经批量预处理后加载至分析型数据仓库中，以进行高性能实时查询。这类系统虽然可对完整大数据集实现高效的即席查询，但无法查询到最新的实时数据，存在数据迟滞高等问题。相较于批处理大数据系统，以Spark Streaming、Storm、Flink为代表的流处理大数据系统将实时数据通过流处理，逐条加载至高性能内存数据库中进行查询。此类系统可以对最新实时数据实现高效预设分析处理模型的查询，数据迟滞低。然而受限于内存容量，系统需丢弃原始历史数据，无法在完整大数据集上支持Ad-Hoc查询分析处理。

#### 4.3.3 需求支撑

随着信息时代的到来，数据开始急剧膨胀，业务也变得很复杂，我们每个人都是个数据源，每时每刻都在产生着数据与个性化、实时化的需求，原本的人力以及服务器已经不足以支撑数据处理平台的业务。由此，流式数据处理便应运而生。

## 5. 组内分工情况

|  |  |  |
| --- | --- | --- |
| 时间 | 姓名 | 任务 |
| 0.5.14~05.21 | 全体成员 | 调研HDFS |
| 收集HDFS相关资料 |
| 阅读HDFS源码 |
| 关晗金 | 撰写项目简介 |
| 撰写需求分析 |
| 郑鹏飞 | 撰写HDFS文档模板 |
| 调研EA[25]并生成UML |
| 张瑞 | 调研StarUML[26] |
| 生成HDFS包图 |
| 05.21~05.30 | 全体成员 | 讨论HDFS框架设计 |
| 讨论并总结类图汇总流程 |
| 汇总并校对文档 |
| 关晗金 | 完善项目简介及需求分析 |
| 撰写核心模块层图 |
| 绘制核心模块UML |
| 汇总并统一类图 |
| 郑鹏飞 | 撰写通信协议模块层 |
| 绘制通信协议模块UML图 |
| 撰写客户端模块层 |
| 绘制客户端模块UML图 |
| 撰写设计特色分析 |
| 张瑞 | 撰写工具及安全模块层 |
| 绘制工具安全模块UML图 |
| 补充并完善设计特色分析 |
| 汇总并统一Xmind文件 |

## 6. 参考文献

[1] O’Malley O, Zhang K, Radia S, et al. Hadoop security design[J]. Yahoo, Inc., Tech. Rep, 2009.

[2] Borthakur D. HDFS architecture guide[J]. Hadoop Apache Project, 2008, 53: 1-13.

[3] Borthakur D. HDFS architecture[J]. Document on Hadoop Wiki. URL http://hadoop. apache. org/common/docs/r0, 2010, 20.

[4] Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system[C]//MSST. 2010, 10: 1-10.

[5] Clustered file system. <https://en.wikipedia.org/wiki/Clustered_file_system>

[6] HDFS. <https://en.wikipedia.org/wiki/Apache_Hadoop#HDFS>

[7] Apache Hadoop 3.1.2. <http://hadoop.apache.org/docs/r3.1.2/>

[8] Hadoop生态. <https://blog.csdn.net/qa962839575/article/details/44256769?ref=myread>

[9] Hadoop家族学习路线图. <http://blog.fens.me/hadoop-family-roadmap/>

[10] HDFS基本架构. <https://blog.csdn.net/qq_27384769/article/details/80325967>

[11] Mackey G, Sehrish S, Wang J. Improving metadata management for small files in HDFS[C]//2009 IEEE international conference on cluster computing and workshops. IEEE, 2009: 1-4.

[12] Thusoo A, Sarma J S, Jain N, et al. Hive-a petabyte scale data warehouse using hadoop[C]//2010 IEEE 26th international conference on data engineering (ICDE 2010). IEEE, 2010: 996-1005.

[13] 徐鹏, Hadoop 2.X 源码剖析.北京:电子工业出版社, 2016.3.

[14] Lu H, Hai-Shan C, Ting-Ting H. Research on Hadoop cloud computing model and its applications[C]//2012 third international conference on networking and distributed computing. IEEE, 2012: 59-63.

[15] 郝树魁. Hadoop HDFS 和 MapReduce 架构浅析[J]. 邮电设计技术, 2012, 7: 37-42.

[16] 大数据存储技术路线. <https://www.cnblogs.com/liangxiaofeng/p/5166795.html>

[17] HDFS 原理、架构与特性介绍. <https://cloud.tencent.com/developer/article/1043733>

[18] Hadoop RPC机制的使用. <https://www.cnblogs.com/edisonchou/p/4285817.html>

[19] HDFS通信协议. <https://yq.aliyun.com/articles/603537?utm_content=m_1000003387>

[20] Google Protocol Buffer 的使用和原理. <https://www.ibm.com/developerworks/cn/linux/l-cn-gpb/index.html>

[21] Protocol Buffer Source Code. <https://github.com/protocolbuffers/protobuf/releases>

[22] HDFS读写流程简介. <https://www.jianshu.com/p/f7a106c6c7b2>

[23] 从客户端逻辑看HDFS写入机制. <https://www.jianshu.com/p/ea6ef5f5b868>

[24] Hadoop HDFS负载均衡. <https://www.cnblogs.com/BYRans/p/5128162.html>

[25] Apache Hadoop HDFS Architecture. <https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/>

[25] EA使用教程. <https://www.cnblogs.com/xinyu1020/articles/1866227.html>

[26] StarUML使用简明教程. <https://blog.csdn.net/luansha0/article/details/82260678>

[27] Hadoop Source Code. <https://github.com/apache/hadoop>

## 7. 附件：体系结构的分解过程

### 7.1 HDFS总体架构图

由Blumind生成，目的:进行并说明HDFS体系结构的分解过程。

### 7.2 EA绘制HDFS类图

由Enterprise Architect生成，目的: 进行HDFS分解过程中的类图绘制。

### 7.3 Xmind文件

该文件夹下包括HDFS体系结构模块.xmind，由Xmind ZEN生成，目的:进行并说明HDFS体系结构的分解过程。模块展开.pdf(展开至具体的java类)以及模块折叠.pdf为Xmind导出的带水印的PDF文件。