

第三次作业：开源系统的软件体系结构的建模与分析

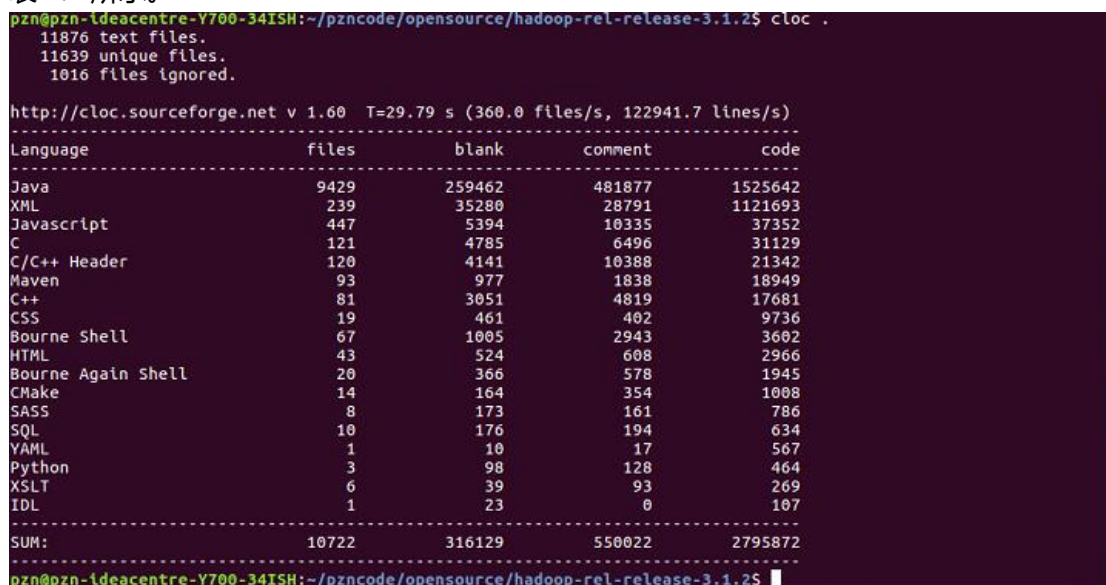
Hadoop 分布式文件系统（版本号：r3.1.2）

一、项目简介

HDFS 源于 Google 在 2003 年 10 月份发表的 GFS (Google File System) 论文, 它其实就是 GFS 的一个克隆版本。HDFS (Hadoop Distributed File System) 是 Hadoop 项目的核心子项目, 是分布式计算中数据存储管理的基础, 是基于流数据模式访问和处理超大文件的需求而开发的, 可以运行于廉价的商用服务器上。它所具有的高容错、高可靠性、高可扩展性、高获得性、高吞吐率等特征为海量数据提供了不怕故障的存储, 为超大数据集 (Large Data Set) 的应用处理带来了很大便利。

Hadoop 分布式文件系统 (HDFS) 是一种分布式文件系统, 设计用于在商用硬件上运行。它与现有的分布式文件系统有许多相似之处。但是, 与其他分布式文件系统的差异很大。HDFS 具有高度容错能力, 旨在部署在低成本硬件上。HDFS 提供对应用程序数据的高吞吐量访问, 适用于具有大型数据集的应用程序。HDFS 放宽了一些 POSIX 要求, 以实现对文件系统数据的流式访问。HDFS 最初是作为 Apache Nutch 网络搜索引擎项目的基础设施而构建的。

经过实验统计, 整个 Hadoop 项目的代码行数为 2795872 行, 如图 1.1 所示。其中包括三类主要的软件模块, 分别为: HDFS、MapReduce 和 Yarn, 如表 1.1 所示。HDFS 项目代码行数为 541845 行, 项目中包的数量为 214 个和类的数量为 2070 个, 总结结果如表 1.2 所示。



```
pzn@pzn-ideacentre-V700-34ISH:~/pzncode/opensource/hadoop-rel-release-3.1.2$ cloc .
11876 text files.
11639 unique files.
1016 files ignored.

http://cloc.sourceforge.net v 1.60 T=29.79 s (360.0 files/s, 122941.7 lines/s)
-----
Language             files      blank      comment      code
-----
Java                  9429       259462      481877      1525642
XML                   239        35280       28791       1121693
Javascript            447        5394        10335       37352
C                     121        4785        6496       31129
C/C++ Header         120        4141       10388       21342
Maven                 93         977        1838       18949
C++                   81        3051       4819       17681
CSS                   19         461        402        9736
Bourne Shell          67        1005       2943       3602
HTML                  43         524        608       2966
Bourne Again Shell    20         366        578       1945
CMake                 14         164        354       1008
SASS                   8          173       161        786
SQL                   10         176       194        634
YAML                   1          10        17         567
Python                3          98       128       464
XSLT                   6          39       93        269
IDL                    1          23        6         107
-----
SUM:                  10722      316129     550022     2795872
pzn@pzn-ideacentre-V700-34ISH:~/pzncode/opensource/hadoop-rel-release-3.1.2$
```

图 1.1 Hadoop 项目整体代码行数统计实验图

```

pzn@pzn-ideacentre-Y700-34ISH: ~/pzncode/opensource/hadoop-rel-release-3.1.2/hadoop-hdfs-project
pzn@pzn-ideacentre-Y700-34ISH:~/pzncode/opensource/hadoop-rel-release-3.1.2/hadoop-hdfs-project$ cloc .
  2404 text files.
  2384 unique files.
   123 files ignored.

http://cloc.sourceforge.net v 1.60 T=4.86 s (467.2 files/s, 153000.2 lines/s)
-----
Language             files            blank           comment           code
-----
Java                  2068             65195            123828            416546
XML                   51               2840             2555              97541
C                    50              1263             1984              8993
CSS                   9               189              118              8567
Javascript            19              867              483              3519
Maven                 7               48               100              1801
C/C++ Header         27              391             1896              1471
HTML                 14              131              208              1453
Bourne Shell         18              208              437               613
Bourne Again Shell   2               29               46               260
CMake                 5               40              100               224
XSLT                  2               7               22               57
-----
SUM:                  2272             71208            131777            541045
-----
pzn@pzn-ideacentre-Y700-34ISH:~/pzncode/opensource/hadoop-rel-release-3.1.2/hadoop-hdfs-project$

```

图 1.2 Hadoop 分布式文件系统代码行数统计实验图

```

pzn@pzn-ideacentre-Y700-34ISH:~/pzncode/opensource/hadoop-rel-release-3.1.2/hadoop-mapreduce-project$ cloc .
  1695 text files.
  1673 unique files.
   69 files ignored.

http://cloc.sourceforge.net v 1.60 T=3.69 s (437.6 files/s, 155104.3 lines/s)
-----
Language             files            blank           comment           code
-----
XML                   37              8790             6426             243374
Java                 1442            33744            63806            197934
C++                  63             1064             1375             6988
C/C++ Header         44              970             1207             3249
Maven                12              75              152             1672
HTML                 5               58               70              285
CMake                2               23               34              174
Bourne Again Shell   1               20               28              129
Bourne Shell         5               37              113              95
Python               1              12              17               71
XSLT                 1               2               14               24
CSS                  1               3               16              11
-----
SUM:                 1614            44798            73258            454006
-----
pzn@pzn-ideacentre-Y700-34ISH:~/pzncode/opensource/hadoop-rel-release-3.1.2/hadoop-mapreduce-project$

```

图 1.3 海量数据的分布式计算系统代码行数统计实验图

```

pzn@pzn-ideacentre-Y700-34ISH:~/pzncode/opensource/hadoop-rel-release-3.1.2$ cd hadoop-yarn-project/
pzn@pzn-ideacentre-Y700-34ISH:~/pzncode/opensource/hadoop-rel-release-3.1.2/hadoop-yarn-project$ cloc .
  4060 text files.
  3975 unique files.
   287 files ignored.

http://cloc.sourceforge.net v 1.60 T=7.66 s (487.9 files/s, 119013.1 lines/s)
-----
Language             files            blank           comment           code
-----
Java                  3143            93789            143856            544218
XML                   56              3040             2272             70279
Javascript            427            2563             8103            18515
C                    13              873              932             6871
Maven                 31              413              590             4995
C++                   9              363              227             2897
CSS                   6              250              229             1037
SASS                  8              173              161              786
SQL                  10              176              194              634
YAML                  1               10               17              567
C/C++ Header         18              204              841              528
Bourne Shell         10              93              314              276
Bourne Again Shell   1               20               39              248
CMake                 2               36               48              132
HTML                 2               12               34               50
XSLT                  2               4               25              48
-----
SUM:                  3739            102019            157882            652081
-----
pzn@pzn-ideacentre-Y700-34ISH:~/pzncode/opensource/hadoop-rel-release-3.1.2/hadoop-yarn-project$

```

图 1.4 Hadoop 集群的资源管理系统代码行数统计实验图

如图 1.2-1.4 所示，应用 cloc 代码行数统计工具，对 hdfs 的三个最主要的模块进行代码统计，综上，将 Hadoop 三个模块的代码行数统计得到表 1.1。

表 1.1 Hadoop 不同模块的代码行数统计表

模块名称	功能	类数量	代码行数
HDFS	Hadoop 分布式文件系统	2070	541845
MapReduce	海量数据的分布式计算	1453	454006
Yarn	Hadoop 集群的资源管理系统	3072	652081

表 1.2 hdfs 包和类的数量

包数量	类数量	代码行数
214	2070	541845

二、需求分析

对外部客户机而言，HDFS 就像一个传统的分级文件系统。可以创建、删除、移动或重命名文件，等等。但是 HDFS 的架构是基于一组特定的节点构建的，这是由它自身的特点决定的。这些节点包括 NameNode（仅一个），它在 HDFS 内部提供元数据服务；DataNode，它为 HDFS 提供存储块。由于仅存在一个 NameNode，因此这是 HDFS 的一个缺点（单点失败）。

存储在 HDFS 中的文件被分成块，然后将这些块复制到多个计算机中 (DataNode)。这与传统的 RAID 架构大不相同。块的大小（通常为 64MB）和复制的块数量在创建文件时由客户机决定。NameNode 可以控制所有文件操作。HDFS 内部的所有通信都基于标准的 TCP/IP 协议。

该软件在用来存储文件时必须能解决相应的异常，以实现文件更好的存储在服务器上。HDFS 的主要目标就是在存在故障的情况下也能可靠地存储数据。三个最常见的故障是名字节点故障，数据节点故障和网络断开。为了系统的可靠性，必须具备以下功能。

重新复制功能。一个数据节点周期性发送一个心跳包到名字节点。网络断开会造成一组数据节点子集和名字节点失去联系。名字节点根据缺失的心跳信息判断故障情况。名字节点将这些数据节点标记为死亡状态，不再将新的 IO 请求转发到这些数据节点上，这些数据节点上的数据将对 HDFS 不再可用，可能会导致一些块的复制因子降低到指定的值。

名字节点检查所有的需要复制的块，并开始复制他们到其他的数据节点上。重新复制在

有些情况下是不可或缺的，例如：数据节点失效，副本损坏，数据节点磁盘损坏或者文件的复制因子增大。

数据正确性功能。从数据节点上取一个文件块有可能是坏块，坏块的出现可能是存储设备错误，网络错误或者软件的漏洞。HDFS 客户端实现了 HDFS 文件内容的校验。当一个客户端创建一个 HDFS 文件时，它会为每一个文件块计算一个校验码并将校验码存储在同一个 HDFS 命名空间下一个单独的隐藏文件中。当客户端访问这个文件时，它根据对应的校验文件来验证从数据节点接收到的数据。如果校验失败，客户端可以选择从其他拥有该块副本的数据节点获取这个块。

元数据失效找回功能。FsImage 和 Editlog 是 HDFS 的核心数据结构。这些文件的损坏会导致整个集群的失效。因此，名字节点可以配置成支持多个 FsImage 和 EditLog 的副本。任何 FsImage 和 EditLog 的更新都会同步到每一份副本中。

同步更新多个 EditLog 副本会降低名字节点的命名空间事务交易速率。但是这种降低是可以接受的，因为 HDFS 程序中产生大量的数据请求，而不是元数据请求。名字节点重新启动时，选择最新一致的 FsImage 和 EditLog。

名字节点对于一个 HDFS 集群是单点失效的。假如名字节点失效，就需要人工的干预。还不支持自动重启和到其它名字节点的切换。

根据需求，对于其质量属性进行分析如下表所示。

表 2.1 可靠性（容错性）

场景	可能的值
源	用户
刺激	副本丢失或宕机
制品	系统的处理器、通信通道、进程
环境	正常操作
响应	保存多个副本，默认存 3 份
响应度量	时间间隔

表 2.2 易用性

场景	可能的值
源	用户

刺激	大量数据同时涌入
制品	系统的处理器、通信通道、进程
环境	正常操作
响应	一次写入，多次读取，高吞吐量
响应度量	时间间隔

三、软件体系结构设计

HDFS 系统的体系结构如图 3.1 所示, 其中存在 Namenode、Datanode、DFSClient, 三者之间的通信通过 IPC 建立, 最底层仍然是 TCP, 通过 Hadoop 的 RPC 机制作出 Demo。Namenode 充当 HDFS 的控制器, DFSClient 做任何事是必须向 NameNode 发出请求, 等 Namenode 处理完请求给予响应后, 才允许和 Datanode 交互。DFSClient 与 Datanode 不存在请求与被响应的策略, 因此 Hadoop HDFS 并未使用 RPC 来实现他们俩之间的通信, 而是直接通过底层的 socket 连接 Datanode 的 ServerSocket, 以实现数据的传输。

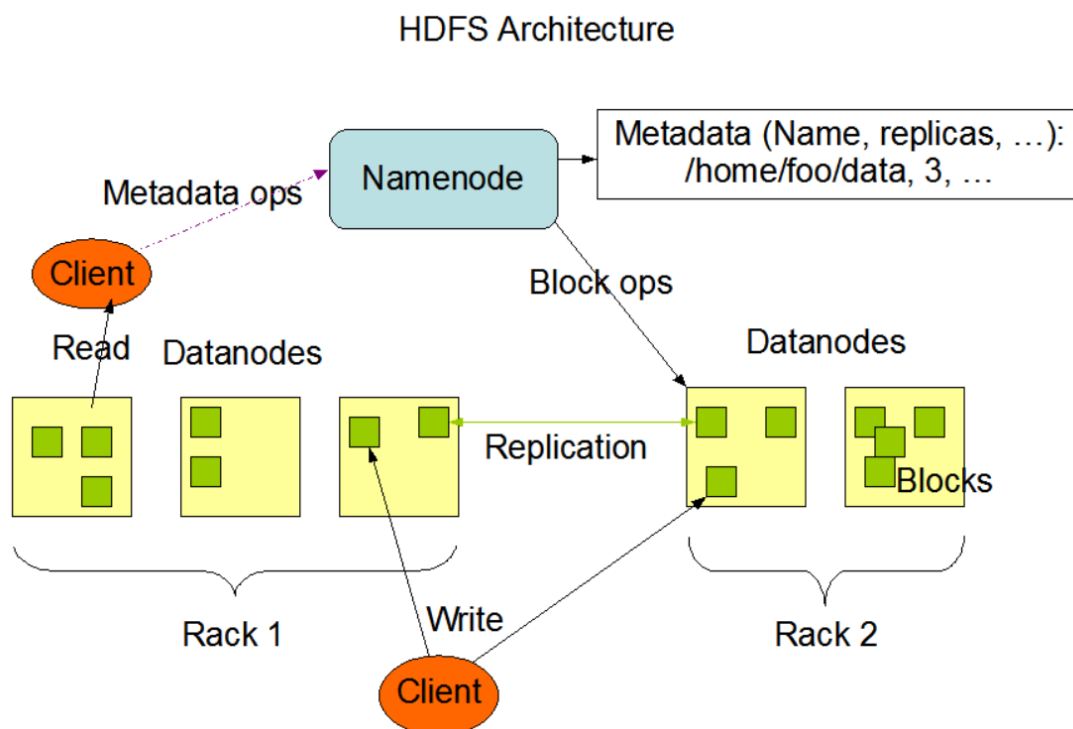


图 3.1 HDFS 软件体系结构图

1 HDFS 中提供两种通信协议

1)Hadoop RPC 接口：HDFS 中基于 Hadoop RPC 框架实现的接口

2)流式接口：HDFS 中基于 TCP 或者 HTTP 实现的接口

2 Hadoop RPC 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 调用基于 Protobuf 实现。

Hadoop RPC 包括以下几个接口：

1)ClientProtocol:

在 org.apache.hadoop.hdfs.protocol 包中的 ClientProtocol.java 文件中，是客户端和 Namenode 间的接口。定义了所有由客户端发起的，由 Namenode 响应的操作，这个接口有 80 多个方法，可以分为以下几类：

HDFS 文件读相关的操作:

客户端调用 getBlockLocations()方法获取 HDFS 文件指定范围内所有数据块的位置信息，每个数据块的位置信息指的是存储这个数据块副本的所有 Datanode 的信息，而且会按照与当前客户端的距离远近排序；如图 3.2 所示，该图为 HDFS 读取流程图。客户端来调用 reportBadBlocks()方法向 Namenode 汇报错误的数据块信息。

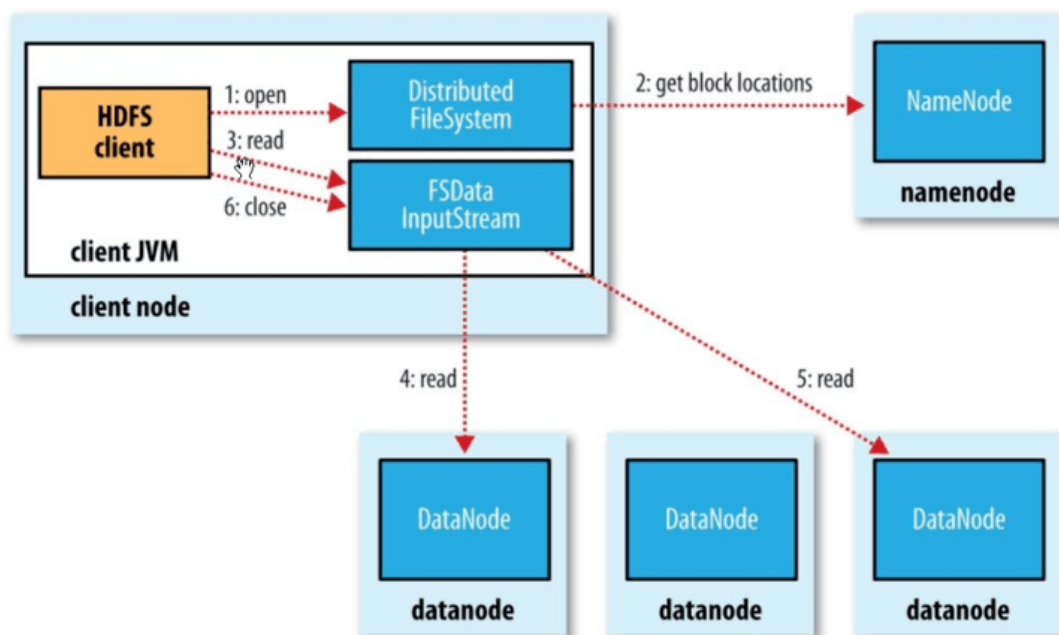


图 3.2 HDFS 读取流程图

HDFS 文件写以及追加写的相关操作:

在 HDFS 客户端操作中最重要的一部分就是写入一个新的 HDFS 文件，或者打开一个已有的 HDFS 文件执行追加写操作。ClientProtocol 中定义了 8 种方法支持 HDFS 写操作：create() 方法用于在 HDFS 的文件系统目录树中创建一个新的空文件。然后再调用

addBlock()方法获取存储文件数据的数据块的位置信息，最后客户端根据位置信息建立数据流管道写入数据。

append()方法用于打开一个已有的文件，如果这个文件的最后一个数据块没有写满，则返回这个数据块的位置信息，如果写满，则新建一个数据块返回信息。

客户端调用 addBlock()方法向指定文件添加一个新的数据块，并获取数据块副本的所有数据节点的位置信息。

当客户端完成了整个文件的写入操作后，会调用 complete()方法通知 Namenode。以上是正常情况，如遇到错误时，会有以下方法：

abandonBlock()当客户端获取了一个新的申请的数据块，发现无法建立到存储这个数据块副本的某些数据节点单位连接时，会调用这个方法通知 Namenode 放弃这个数据块，之后客户端再次调用 addBlock()方法获取新的数据块，并在参数中将无法链接的节点传给名字节点，避免再次分到这个节点上。

getAdditionalDatanode()和 updatePipeline()如果客户端已经成功建立了数据管道，在客户端写某个数据块，存储这个数据块副本的某个数据节点出现错误时，客户端首先会调用 getAdditionalDatanode()方法向 Namenode 申请一个新的 Datanode 来替代出现故障的 Datanode。然后客户端会调用 updateBlockForPipeline()方法向 Namenode 申请为这个数据块分配新的时间戳，这样故障节点上的没能写完整的数据块的时间戳就会过期，在后续的块汇报操作中会被删除，最后客户端就可以使用新的时间戳建立新的数据管道，来执行对数据块的写操作。数据流管道建立成功后，客户端还需要调用 updatePipeline()方法更新 Namenode 中当前数据块的数据管道信息。

renewLease() 当写的过程中 client 发生异常时，对于任意一个 Client 打开的文件都需要 Client 定期调用该方法更新租约，如果 Namenode 长时间没有收到 Client 的租约更新消息，就会认为 Client 发生故障，这是就会触发一次租约恢复操作，关闭文件并且同步所有数据节点上这个文件数据块的状态，确保 HDFS 系统中这个文件是正确且一致保存的。如果写的时候 Namenode 节点出错，就涉及 HDFS 的 HA。

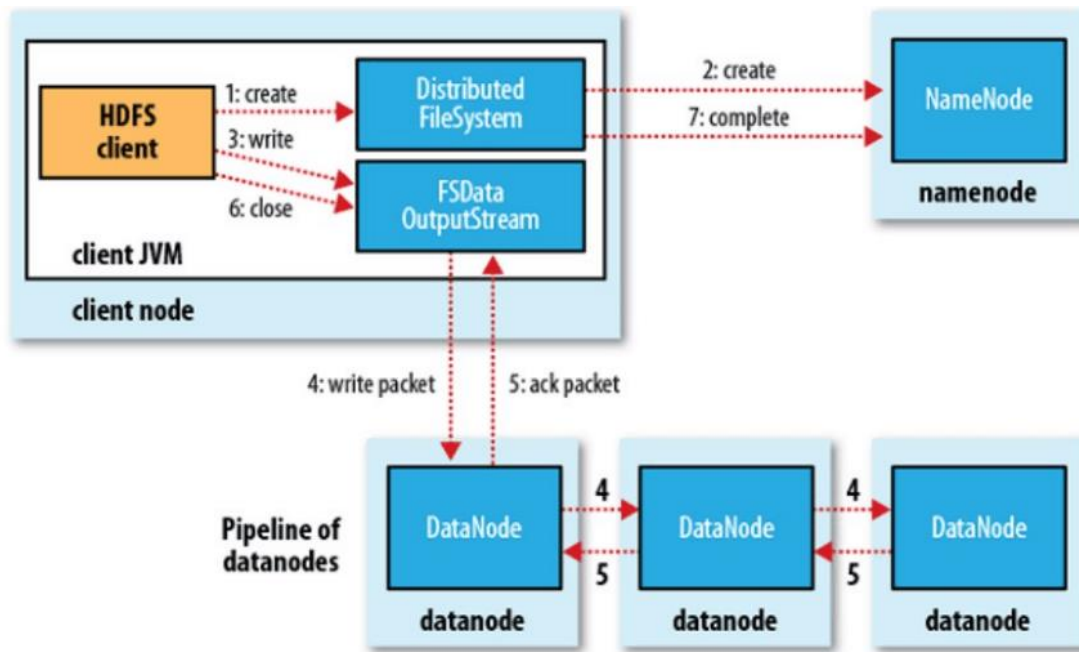


图 3.3 HDFS 写入流程图

2) ClientDatanodeProtocol:

在 org.apache.hadoop.hdfs.protocol 包中的 ClientDatanodeProtocol.java 文件中，是客户端与数据节点的接口（这个方法主要是用于客户端获取数据节点信息时调用，真正的数据读写交互在流式接口进行）

(1) getReplicaVisibleLength()

客户端调用该方法从数据节点获取某个数据块副本真实的数据长度。当客户端读取一个文件时，需要获取这个文件对应的所有数据块的长度，用于建立数据块的输入流，然后读取数据。

(2) getBlockLocalPathInfo()

HDFS 对于本地读取，也就是 Client 和 Datanode 在同一物理机上时，调用该方法优化了文件读取。

(3) refreshNamenoders()

在用户管理员命令行中有一个 'hdfs dfsadmin datanode host:port' 命令，用于出发指定的 Datanode 重新加载配置文件，该命令底层就是调用这个接口。

(4) deleteBlockPool()

在用户管理员命令行中有一个 'hdfs dfsadmin datanode deleteBlockPool datanode-host:port blockpoolId [force]' 命令，用于从指定 Datanode 删除 blockpoolId 对应的块池，如果 force 参数被设置，则强制删除。

3) DatanodeProtocol:

在 org.apache.hadoop.hdfs.server.protocol 包中的 DatanodeProtocol.java 文件中，是数据节点与名字节点的通信接口。DatanodeProtocol 定义的方法分为三类：Datanode 启动相关、心跳相关以及数据块读写相关。

(1)Datanode 启动相关方法

一个完整的 Datanode 启动操作会与 Namenode 进行 4 次交互，首先调用 versionRequest() 与 Namenode 进行握手操作，然后调用 registerDatanode() 向 Namenode 注册当前的 Datanode，接着调用 blockReport()汇报 Datanode 上存储的所有数据块，最后调用 cacheReport()汇报 Datanode 缓存的所有数据块。

(2)心跳相关方法

Datanode 会定期(由 dfs.heartbeat.interval 配置项配置，默认 3 秒)向 Namenode 发送心跳，如果 Namenode 长时间没有接到 Datanode 发送的心跳，则 Namenode 会认为该 Datanode 失败。

(3)数据块读写相关方法

Datanode 在进行数据块读写操作时与 Namenode 交互的方法包括 repotBadBlocks(),blockReceivedAndDeleted()以及 commitBlockSynchronization()方法。

Datanode 会调用 repotBadBlocks()方法向 Namenode 汇报损坏的数据块，Datanode 会在三种情况下调用：DataBlockScanner 线程定期扫描数据节点上储存的数据块，发现数据块的校验出现错误时；数据流管道写数据时，Datanode 接受了一个新的数据块，进行数据块校验操作出现错误时；进行数据块复制操作，Datanode 读取本地存储的数据块时，发现本地数据块副本的长度小于 Namenode 记录的长度时。

Datanode 会定期（默认 5 分钟，不可更改）调用 blockReceivedAndDeleted()方法向 Namenode 汇报 Datanode 新接受的数据块或者删除的数据块。

commitBlockSynchronization()方法用于在租约恢复操作时同步数据块的状态。

4)InterDatanodeProtocol:

在org.apache.hadoop.hdfs.server.protocol包中的InterDatanodeProtocol.java文件中，是 Datanode 与 Datanode 间的通信接口，主要用于租约恢复操作。

5)NamenodeProtocol:

在 org.apache.hadoop.hdfs.server.protocol 包中的 NamenodeProtocol.java 文件中，是第二名字节点与名字节点间的接口。其他接口：主要包括安全相关接口，HA 相关接口。

3 流式接口

流式接口是 HDFS 中基于 TCP 或 HTTP 实现的接口，在 HDFS 中，流式接口包括了基

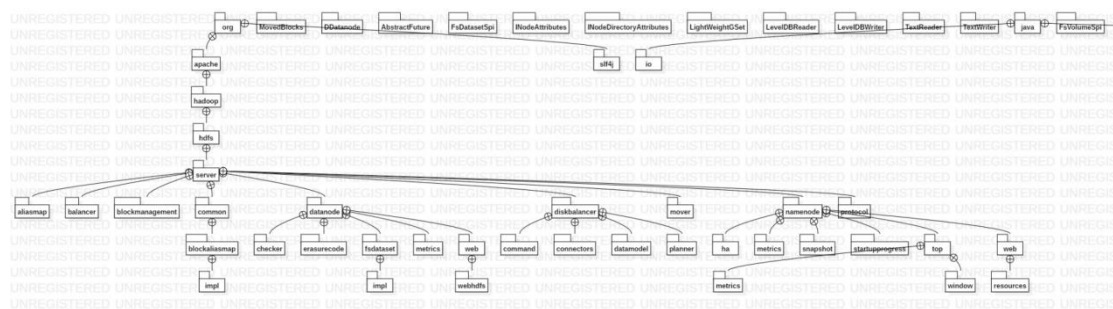
于 TCP 的 DataTransferProtocol 接口, 以及 HA 架构中的 Active Namenode 和 Standby Namenode 之间的 HTTP 接口。

1)DataTransferProtocol

2)Active Namenode 和 Standby Namenode 之间的 HTTP 接口

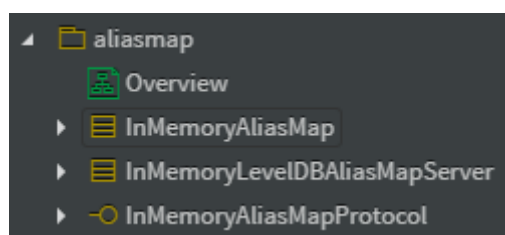
由于 Standby Namenode 成功地将读入的 editlog 文件与当前的命名空间合并, 从而始终保持着一个最新版本的命名空间, 所以 Standby Namenode 只需定期将自己的命名空间写入一个新的 fsimage 文件, 并通过 HTTP 协议将这个 fsimage 文件传回 Active Namenode 即可。

四、核心模块 Server 包结构分析



通过阅读源码和分析, 发现共包含主要的包有以下几个, 分别为: net, protocolPB, qjournal, security, server, tools, util, web 等。其中, hdfs 软件系统的最核心的模块是其 server 的部分, server 部分共包含 9 个包, 每个包下都包含若干个类, 对于每个包进行分析。

1 aliasmap 包: 共包含两个类和一个接口。

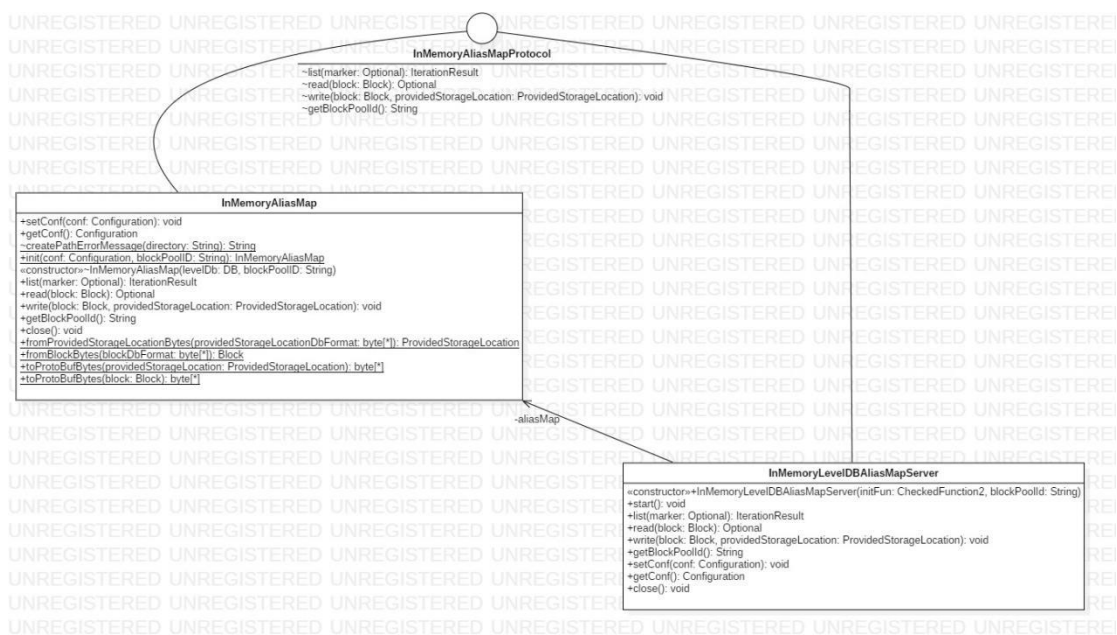


1.1 InMemoryAliasMap 类是 InMemoryAliasMapProtocol 接口的一个实现, 用于与 LevelDB 一起使用。

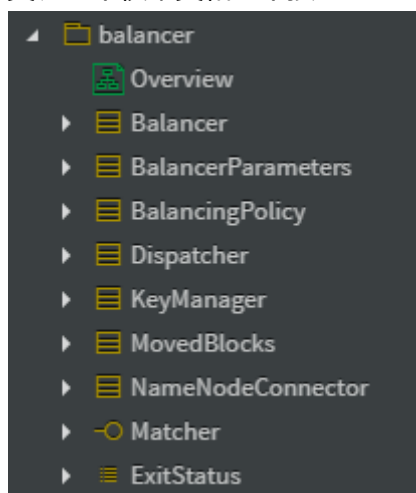
1.2 InMemoryAliasMapProtocol 接口是客户端用于读取/写入有关所提供块的别名的数据的协议, 用于 {@link org.apache.hadoop.hdfs.server.common.blockaliasmap.BlockAliasMap} 的内存中实现。

1.3 InMemoryLevelDBAliasMapServer 类是 Namenode 进入 {@link InMemoryAliasMap} 的入口点。

该包中类间关系的示意图如下图所示。



2 balancer 包：共包含 7 个类、一个枚举类和一个接口。



2.1 Balancer 类是一种工具，可在某些数据节点已满或新的空节点加入群集时平衡 HDFS 群集上的磁盘空间使用量。该工具作为应用程序部署，可以在应用程序添加和删除文件时由群集管理员在实时 HDFS 群集上运行。

2.2 BalancerParameter 类用于定义 Balancer 中的属性和方法等参数。

2.3 Balancing policy 类由于 datanode 可能包含多个块池，{@link Pool}意味着{@link Node}，但不是相反的。

2.4 Dispatcher 类是在 DataNode 之间移动的调度块。

2.5 ExitStatus 枚举类是与每个退出状态关联的值直接映射到命令行中进程的退出代码。

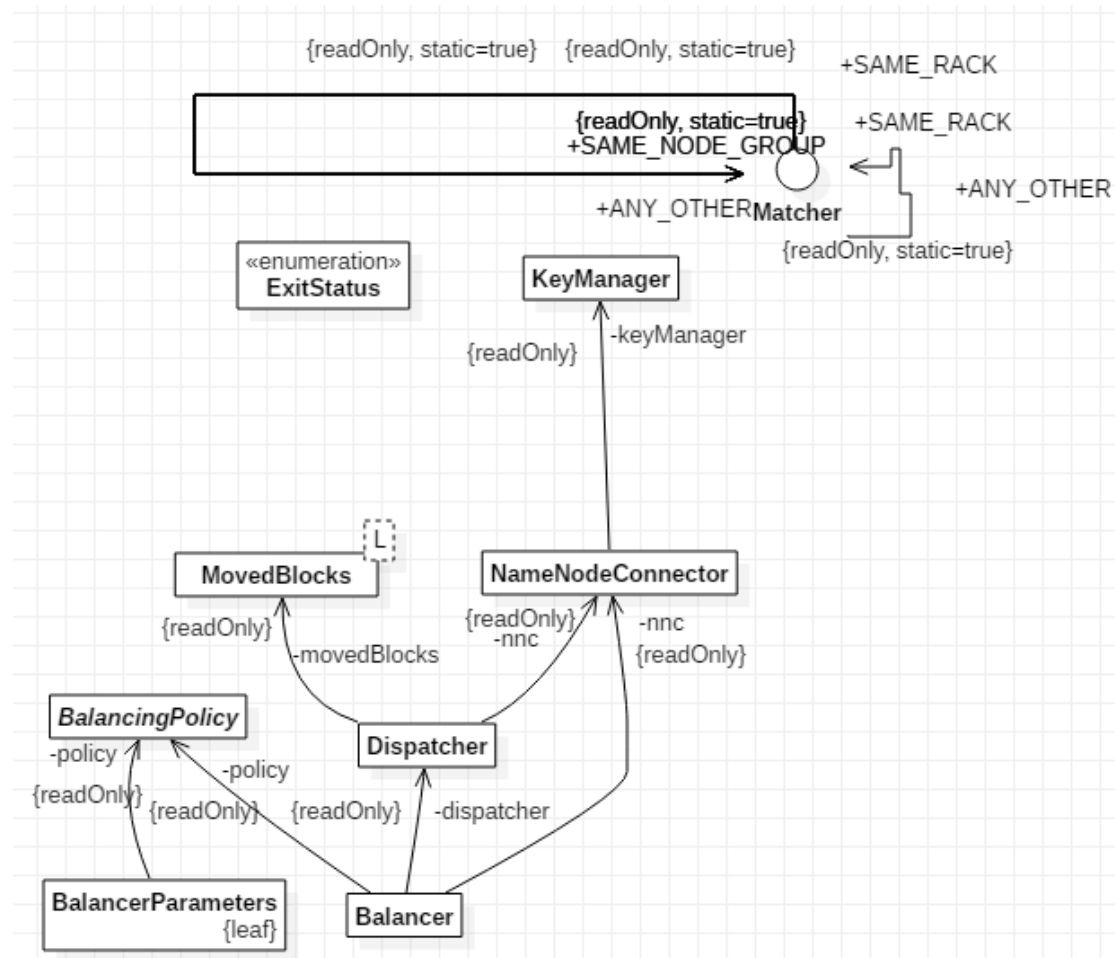
2.6 KeyManager 类是该类提供密钥和令牌管理的实用程序。

2.7 Matcher 接口是用于匹配节点的匹配器接口。

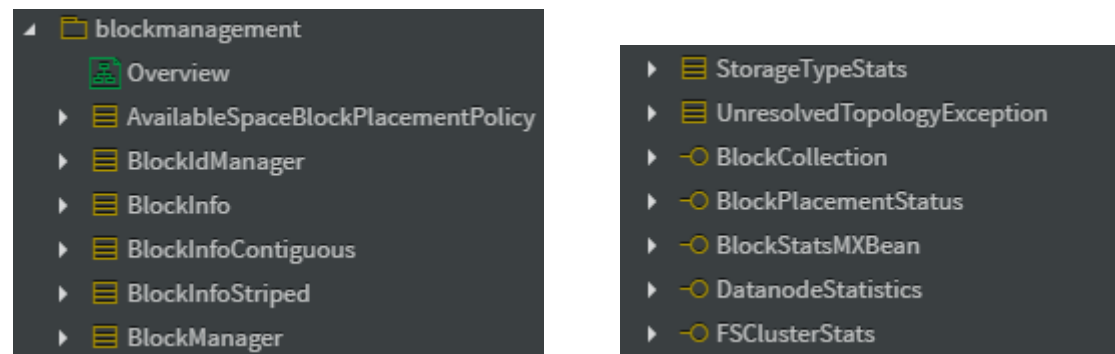
2.8 MovedBlocks 类是此窗口确保保持在固定时间间隔内移动的块（默认为 1.5 小时）。旧窗口有较旧的块；当前窗口具有更新的块；清除方法会触发检查旧窗口中的块是否超过固定时间间隔。经过确认后则会清除旧窗口，然后将当前窗口中的块移动到旧窗口。

2.9 NameNodeConnector 该类提供了访问 NameNode 的实用程序。

该包中类间关系的示意图如下图所示。



3 Blockmanagement 包：共包含 55 个类和 5 个接口。



3.1 AvailableSpaceBlockPlacementPolicy 类主要用来空间平衡块放置策略。

3.2 BlockCollection 接口被块管理器使用，用于公开 Block / BlockUnderConstruction 集

合的一些特征。

3.3 BlockIdManager 类用来分配生成戳记和块 ID。{@link FSNamesystem}负责在{@link FSEditLog}中持久分配。

3.4 BlockInfo 类对于给定块（或擦除编码块组），BlockInfo 类维护 1) 它所属的{@link BlockCollection}，以及 2) 块的副本或属于擦除和存储编码块组的块的数据节点。

3.5 BlockInfoContiguous 类是{@link BlockInfo}的子类，用于具有复制方案的块。

3.6 BlockInfoStriped 类是{@link BlockInfo}的子类，在擦除编码中呈现一个块组。我们仍然使用存储阵列作为块组中的每个块存储 DatanodeStorageInfo。对于 $(m + k)$ 块组，第一 $(m + k)$ 个存储单元被排序并严格映射到相应的块。通常，属于组的每个块仅存储在一个 DataNode 中。但是，某些块可能会被过度复制。因此，存储阵列的大小可以大于 $(m + k)$ 。因此，目前我们使用额外的字节数组来记录每个条目的块索引。

3.7 BlockManager 类是保留与存储在 Hadoop 集群中的块相关的信息。对于块状态管理，它尝试在任何事件（例如，停用，名称节点故障转移，数据节点故障）下维护“活动副本的数量 = 预期冗余的数量”的安全属性。维护模式的动机是允许管理员快速修复节点而无需支付退役费用。因此，对于维护模式，活动副本的数量不必等于预期冗余的数量。如果任何副本处于维护模式，则安全属性扩展如下。这些属性仍适用于零维护副本的情况，因此我们可以将这些安全属性用于所有方案。A. 实时复制品数量 \geq 维护复制的最小数量 #。B. 实时复制品数量 \leq 预期冗余数量 #。C. 实时复制副本和维护副本数量 \geq 预期冗余数量 #。对于常规复制，维护的最小实时复制副本数由 DFS_NAMENODE_MAINTENANCE_REPLICATION_MIN_KEY 确定。此号码必须 \leq DFS_NAMENODE_REPLICATION_MIN_KEY。对于擦除编码，用于维护的最小活复制副本是 BlockInfoStriped # getRealDataBlockNum。另一个安全属性是满足块放置策略。虽然策略是可配置的，但策略应用于的副本是实时副本+维护副本。

3.8 BlockManagerFaultInjector 类用于注入某些故障进行测试。

3.9 BlockManagerSafeMode 类为块管理器安全模式信息。在名称节点启动期间，计算安全块的数量，即至少具有最小副本数量的那些，并计算安全块与系统中块总数的比率，即块的大小。当比率达到{@link #threshold}且已注册足够的实时数据节点时，需要等待安全模式{@link #extension}间隔。延长期过后，在安全块比率达到{@link #threshold}且注册了足够的实时数据节点之前，它不会离开安全模式。

3.10 BlockPlacementPolicies 类为 BlockPlacementPolicy 类的复数。

3.11 BlockPlacementPolicy 类用于选择放置块副本所需的目标数。

3.12 BlockPlacementPolicyDefault 类负责选择放置块副本所需的目标数量。副本放置策略是，如果编写器位于 datanode 上，则第一个副本将放置在本地计算机上，否则将是随机

数据节点。第二个副本放置在另一个机架上的 datanode 上。第三个副本放置在作为第二个副本的机架的不同节点上的 datanode 上。

3.13 BlockPlacementPolicyRackFaultTolerant 类负责选择放置块副本所需的目标数量。策略是它尽力将副本放置到大多数机架上。

3.14 BlockPlacementPolicyWithNodeGroup 类负责选择所需数量的目标，以便在具有节点组层的环境中放置块副本。副本放置策略调整为：如果编写器位于 datanode 上，则第一个副本放置在本节点（或本地节点组或本地机架上）上，否则为随机数据节点。第二个副本放置在与第一个副本节点位于不同机架上的 datanode 上。第三个副本放置在 datanode 上，该 datanode 位于不同的节点组上，但与第二个副本节点位于同一个机架上。

3.15 BlockPlacementPolicyWithUpgradeDomain 类负责选择所需数量的目标，以放置符合升级域策略的块副本。这是副本放置策略。如果编写器位于 datanode 上，则第一个副本将放置在本节点计算机上，否则将是随机数据节点。第二个副本放置在另一个机架上的 datanode 上。第三个副本放置在作为第二个副本的机架的不同节点上的 datanode 上。所有 3 个副本都具有唯一的升级域。

3.16 BlockPlacementStatus 为块放置接口。

3.17 BlockPlacementStatusDefault 类为实现了接口 BlockPlacementStatus 的类。

3.18 BlockPlacementStatusWithNodeGroup 类用于 @see BlockPlacementPolicyWithNodeGroup 的 @see BlockPlacementStatus 的实现。

3.19 BlockPlacementStatusWithUpgradeDomain 类用于 @see BlockPlacementPolicyWithUpgradeDomain 的 @see BlockPlacementStatus 的实现。

3.20 BlockReconstructionWork 类由 {@link BlockManager # computeReconstructionWorkForBlocks} 在内部使用，以表示通过复制或擦除编码重建块的任务。通过将数据从 srcNodes 传输到目标来完成重建。

3.21 BlockReportLeaseManager 类用于 BlockReportLeaseManager 管理块报告租约。

3.22 BlocksMap 类维护从块到其元数据的映射。块的元数据当前包括它所属的 blockCollection 以及存储块的 datanode。

3.23 BlockStatsMXBean 接口是这是用于检索与块管理相关的统计信息的接口。

3.24 BlockStoragePolicySuite 类是块存储策略的集合。

3.25 BlockToMarkCorrupt 类是用于构建 “toCorrupt” 列表，该列表是由于块报告而应被视为损坏的块列表。

3.26 BlockUnderConstructionFeature 类是表示块的正在构造的特征。这通常是为写入或追加而打开的文件的最后一个块。

3.27 CacheReplicationMonitor 类扫描名称系统，根据需要调度要缓存的块。CacheReplicationMonitor 在 NameNode 首次启动时执行完整扫描，之后以可配置的间隔执行完整扫描。

3.28 CombinedHostFileManager 类使用 json 文件管理 datanode 配置。有关 json 格式，请参阅{@link CombinedHostsFileReader}。

3.29 CorruptReplicasMap 类存储有关文件系统中所有损坏块的信息。仅当所有副本都已损坏时，块才被视为已损坏。在报告 Block 的副本时，我们会隐藏任何损坏的副本。一旦发现 Block 具有预期数量的良好副本，就会删除这些副本。映射：块 -> TreeSet<DatanodeDescriptor>。

3.30 DatanodeAdminManager 类用于管理 DataNodes 的退役和维护状态。后台监视器线程定期检查退役或进入维护状态的 DataNode 的状态。

3.31 DatanodeDescriptor 类使用临时信息（例如，运行状况，容量，与 Datanode 关联的块）扩展 DatanodeInfo 类，该信息是对 Namenode 的私有，即此类不向客户端公开。

3.32 DatanodeManager 类管理数据节点，包括退役和其他活动。

3.33 DatanodeStatistics 接口为统计数据节点的接口。

3.34 DatanodeStats 类用于对数据节点的统计。对于退役/退役节点，仅计算使用的容量。

3.35 DatanodeStorageInfo 类数据节点具有一个或多个存储。Datanode 中的存储由此类表示。

3.36 ErasureCodingWork 类为擦除编码工作的类。

3.37 ExcessRedundancyMap 类将数据节点映射到一组多余的冗余详细信息。这个类是线程安全的。

3.38 FSClusterStats 接口用于检索群集的和负载相关的统计信息。

3.39 HeartbeatManager 类管理从数据节点接收的心跳。数据节点列表和统计信息由心跳管理器锁同步。

3.40 Host2NodesMap 类是从主机名到 datanode 描述符的映射。

3.41 HostConfigManager 类是该接口抽象了如何管理 datanode 配置。每个实现都定义了自己的持久配置方式。例如，它可以使用一个 JSON 文件来存储所有数据节点的配置；或者它可以使用一个文件来存储服务中的数据节点和另一个文件来存储解除授权请求的数据节点。这些文件控制 NameNode 期望在群集中看到的 DataNode。

3.42 HostFileManager 类管理 HDFS 的包含和排除文件。

3.43 HostSet 类允许对匹配通配符地址进行有效查询。对于具有相同主机地址的 InetSocketAddress A 和 B，我们定义 A 和 B 之间的部分顺序， $A \leq B$ iff $A.getPort() == B.getPort() \parallel B.getPort() == 0$ 。

3.44 InvalidateBlocks 类为每个包含最近失效的块的命名机器保留一个 Collection，并且这些块被认为存在于相关机器上。

3.45 LocatedBlockBuilder 类是为了建立块的位置。

3.46 LowRedundancyBlocks 类保留低冗余块的优先级队列。块具有冗余优先级，优先级{@link #QUEUE_HIGHEST_PRIORITY}表示最高优先级。拥有优先级队列允许{@link BlockManager}选择首先复制哪些块 - 尝试优先处理风险最高或被认为最有价值的数据。选择给予添加块的优先级的策略在{@link #getPriority (BlockInfo, int, int, int, int) } 中实现。

3.47 NumberReplicas 类是一个不可变对象，用于存储活动副本的数量和已停用的副本的数量。

3.48 OutOfLegacyGenerationStampsException 类当名称节点用完 V1 (旧版) 生成标记时，抛出此异常。

3.49 PendingDataNodeMessages 类在备用节点中，我们可以在命名空间中实际可用之前或在命名空间中具有过时状态时接收有关块的消息。在这些情况下，我们将这些与块相关的消息排入此结构中。

3.50 PendingReconstructionBlocks 类对所有获得更强冗余的块进行簿记。它执行以下操作：1) 记录此时刻获得更强冗余的块。2) 粗粒计时器，用于跟踪重建请求的年龄。3) 一个线程，定期识别从未成功的重建请求。

3.51 PendingRecoveryBlocks 类跟踪每个块的恢复尝试及其超时，以确保我们不会同时进行多次恢复，并仅在恢复超时到期后重试。

3.52 ProvidedStorageMap 类允许我们在数据节点本地存储和存储之间进行管理和多路复用。

3.53 ReplicationWork 类复制工作的类。

3.54 ReplicaUnderConstruction 类 ReplicaUnderConstruction 包含有关正在构建的副本（或属于块组的块）的信息。GS 是指副本的长度和状态由 datanode 报告。不能保证，但预计数据节点实际上有相应的副本。

3.55 SequentialBlockGroupIdGenerator 类通过递增到目前为止分配的最大块组 ID 来生成下一个有效块组 ID，其中保留前 2^{10} 个块组 ID。HDFS-EC 引入了一个分层协议来命名块和组：连续：{保留块 ID | 国旗 | block ID} Striped：{保留块 ID | 国旗 | 阻止组 ID | 组中的索引}

在 n 位保留块 ID 之后，ID 中的第 (n + 1) 位区分连续 (0) 和条带 (1) 块。对于条带块，位 (n + 2) 到 (64-m) 表示其块组的 ID，而最后的 m 位表示其组的索引。值 m 由组中的最大块数 (MAX_BLOCKS_IN_GROUP) 确定。请注意，{@link #nextValue () }方

法需要外部锁以保证 ID 没有冲突。

3.56 SequentialBlockIdGenerator 类通过递增到目前为止分配的最大块 ID（从 $2^{30} + 1$ 开始）生成下一个有效块 ID。过去曾经随机分配的块 ID。因此，我们可能会在顺序遍历 ID 空间时发现一些冲突。但是，鉴于 ID 空间的稀疏性，冲突应该很少，并且可以在检测到时跳过。

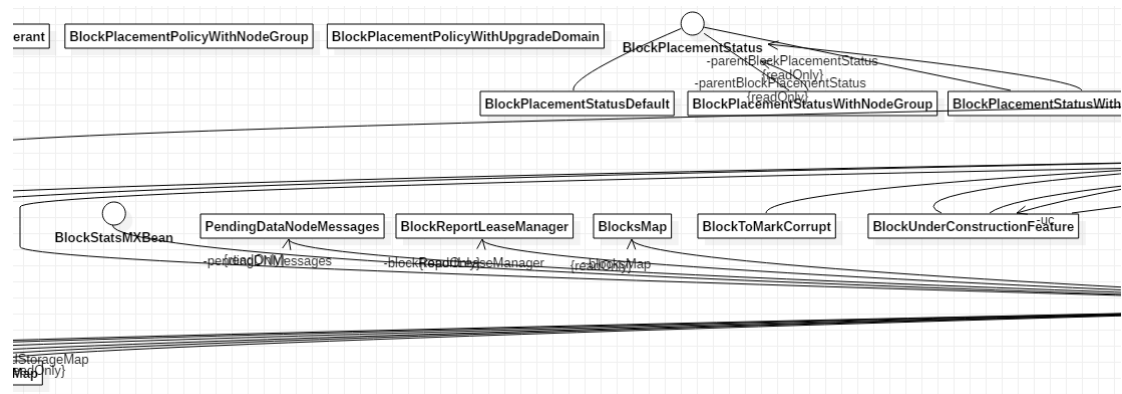
3.57 SlowDiskTracker 类汇总来自通过核心模块接收{@link SlowDiskReports}的信息。

3.58 SlowPeerTracker 类汇总来自通过核心模块接收的{@link SlowPeerReports}的信息。

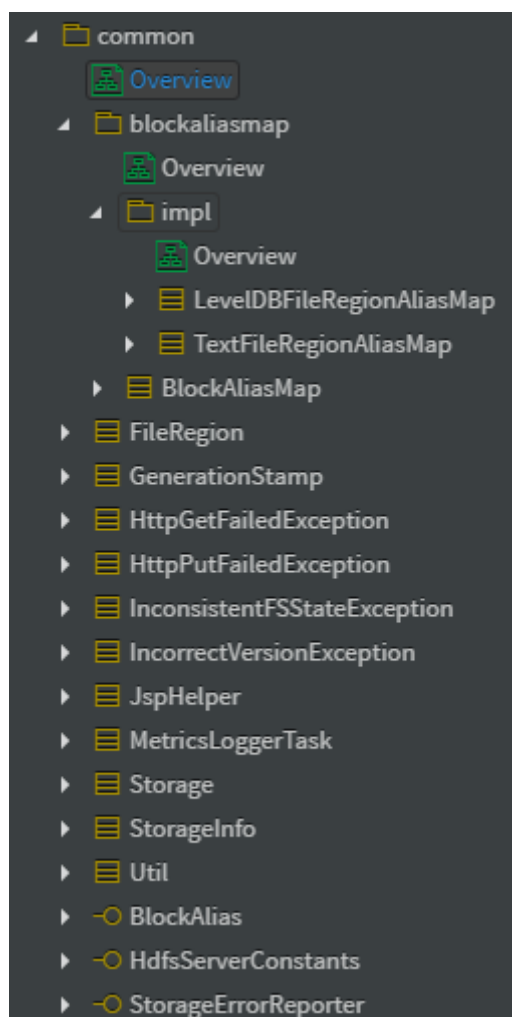
3.59 StorageTypeStats 类用于统计每个存储格式。

3.60 UnresolvedTopologyException 异常类。如果解析节点的拓扑路径失败，则抛出此异常。

该包中类间关系的示意图如下图所示。特别地，由于该包内类和接口的数目较多，类间关系也较多，则所插该图为整个类图的节选。



4 common 包：共包含 1 个包、11 个类和 4 个接口。



4.1 BlockAlias 接口用于加载提供的块的接口。

4.2 FileRegion 类用于表示作为文件区域的提供的块，即可以使用（路径，偏移，长度）来描述。

4.3 GenerationStamp 类是一个 Hadoop FS 原语，由 long 标识。

4.4 HdfsServerConstants 接口包含一些方便的内部 HDFS 常量。

4.5 HttpGetFailedException 类为 http 获取失败异常类。

4.6 HttpPutFailedException 类为 http 传送失败异常类。

4.7 InconsistentFSStateException 类当文件系统状态不一致且无法恢复时，抛出异常。

4.8 IncorrectVersionException 类当外部版本与应用程序的当前版本不匹配时，抛出异常。

4.9 JspHelper 类为 JSP 帮助类。

4.10 MetricsLoggerTask 类可用作实用程序将指标转储到日志。

4.11 Storage 类存储信息文件。本地存储信息存储在单独的文件 VERSION 中。它包含节点类型，存储布局版本，命名空间 ID 和 fs 状态创建时间。本地存储可以驻留在多个目录中。每个目录应包含与其他目录相同的 VERSION 文件。在启动期间，Hadoop 服务器（名称节

点和数据节点) 从它们读取本地存储信息。服务器在运行时为每个存储目录保持锁定, 以便其他节点无法启动共享相同的存储。当服务器停止 (正常或异常) 时, 将释放锁。

4.12 StorageErrorReporter 接口{@link JournalManager}的实现可用于报告的接口避免了日记管理器与实例化它们的存储之间的循环依赖关系。

4.13 StorageInfo 类为存储信息的通用类。TODO namespaceID 应该很长并计算为哈希 (地址+端口) 。

4.14 Util 类为工具类。

4.15 Blockaliasmap 包: 包括 1 个包、1 个类。

4.15.1 BlockAliasMap 类用于读取和写入提供的块的块映射的抽象类。

4.15.2 impl 包: 包括 3 个类。

4.15.2.1 InMemoryLevelDBAliasMapClient 类是 InMemoryAliasMapServer 的客户端。Datanode 和 fs2img 使用它来存储和检索基于给定 Block 的 FileRegions。

4.15.2.2 LevelDBFileRegionAliasMap 类是基于 LevelDB 的{@link BlockAliasMap}实现。

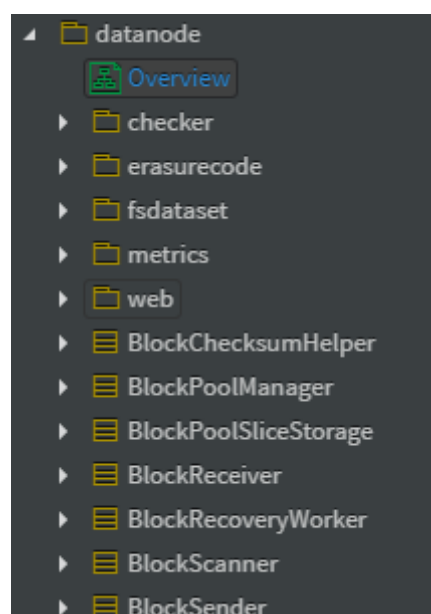
4.15.2.3 TextFileRegionAliasMap 类用于存储为文本文件的块映射, 具有指定的分隔符。

该包中类间关系的示意图如下图所示。特别地, 由于该包内类和接口的数目较多, 类间关系也较多, 则所插该图为整个类图的节选。

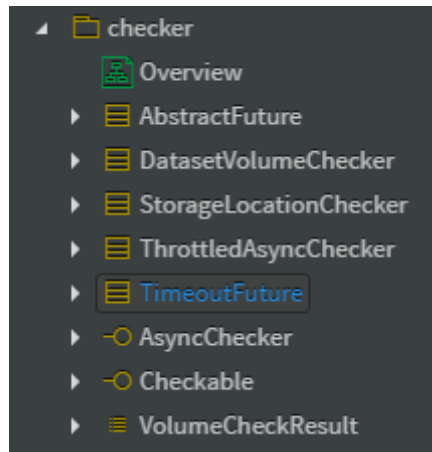


5 Datanode 包: 共包含 5 个包、44 个类和 4 个接口。

由于该包内包和类的数量较多, 因此将列举几个具有代表性的包和其中的类的功能。



5.1 checker 包: 共包含 1 个枚举类、5 个类和 2 个接口。

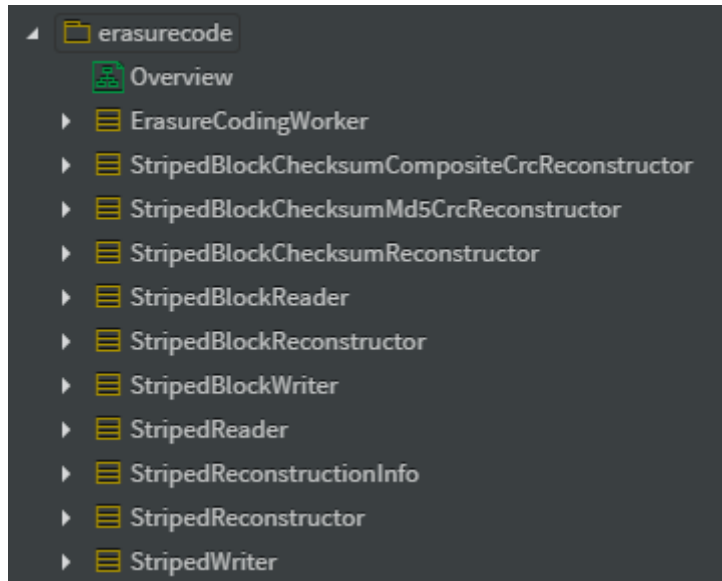


5.1.1 `AbstractFuture` 类{@link `ListenableFuture`}的抽象实现，仅供高级用户使用。创建{@code `ListenableFuture`}的更常见方法包括实例化{@link `SettableFuture`}，将任务提交到{@link `ListeningExecutorService`}，并从现有的任务中派生{@code `Future`}，通常使用{@link `Futures` # `transform (ListenableFuture , com.google.common.base.Function) Futures.transform`}和{@link `Futures` # `catch (ListenableFuture , Class , com.google.common.base.Function , java.util.concurrent.Executor) Futures.catching`}。该类实现{@code `ListenableFuture`}中的所有方法。子类应该提供一种通过受保护的方法{@link `#set (Object)`}，{@link `#setFuture (ListenableFuture)`}和{@link `#setException (Throwable)`}来设置计算结果的方法。子类也可以覆盖{@link `#interruptTask ()`}，如果对{@link `#cancel (boolean)`} `cancel (true)` 的调用成功取消未来，则会自动调用它。子类应该很少覆盖其他方法。

5.1.2 `AsyncChecker` 接口可用于计划对给定{@link `Checkable`}的异步检查。如果检查成功安排，则返回{@link `ListenableFuture`}。

5.1.3 `Checkable` 接口是一个对象，可以通过调用其{@link `#check`}方法来探测其当前状况。

5.2 Erasurecode 包：共包括 11 个类。



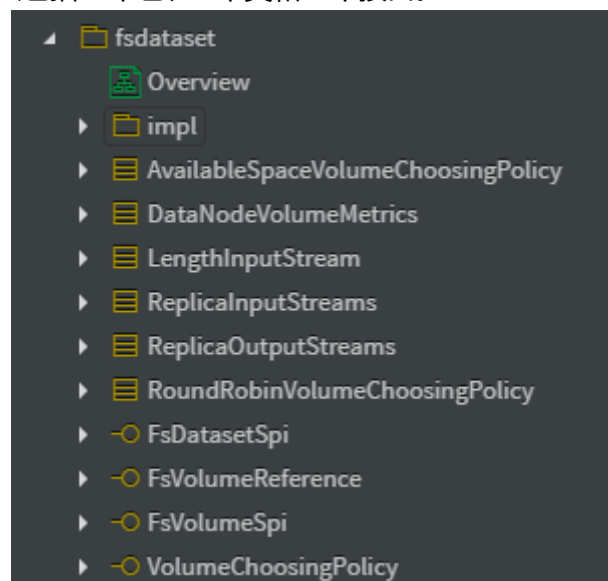
5.2.1 ErasureCodingWorker 类是处理擦除编码重建工作命令。这些命令将作为 Datanode 心跳响应的一部分从 Namenode 发出。BPOfferService 将工作委托给此类以处理 EC 命令。

5.2.2 StripedBlockChecksumCompositeCrcReconstructor 类在重构的块 CRC 上计算条带化复合 CRC。

5.2.3 StripedBlockReader 类是用于从一个源 DN 读取块数据，它包含块读取器，读取缓冲区和条带块索引。仅为一个源分配一次 StripedBlockReader，并且 StripedReader 与源具有相同的数组顺序。通常我们只需要分配最小数量（minRequiredSources）的 StripedReader，并为新的源 DN 分配新的，如果某些现有 DN 无效或缓慢。如果某个源 DN 已损坏，请将相应的 blockReader 设置为 null，并且永远不会再读取它。

5.2.4 StripedBlockWriter 类为条带块写入器，将重建数据写入远程目标 datanode。

5.3 fsdataset 包：共包括 1 个包、6 个类和 4 个接口。



5.3.1 impl 包：共包括 17 个类。

5.3.1.1 BlockPoolSlice 类块池切片表示存储在卷上的块池的一部分。总而言之，在群集中共享块池 ID 的所有 BlockPoolSlices 代表单个块池。此类由{@link FsVolumeImpl}同步。

5.3.1.2 FsDatasetFactory 类是用于创建{@link FsDatasetImpl}对象的工厂。

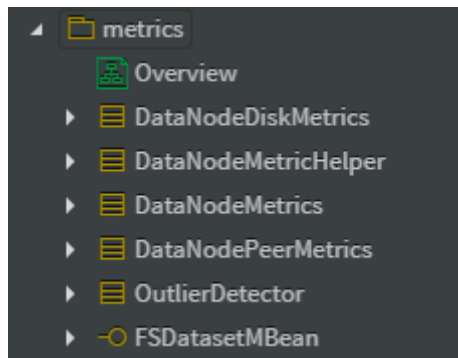
5.3.1.3 MappableBlock 类表示由 DataNode mmapped 的 HDFS 块。

5.3.2 AvailableSpaceVolumeChoosingPolicy 类是 DN 卷选择策略，在考虑分配新副本分配的位置时考虑每个可用卷上的可用空间量。默认情况下，此策略更喜欢将副本分配给具有更多可用空间的卷，以便随着时间的推移平衡 DN 中所有卷的可用空间。使用细粒度锁可以同时选择不同存储类型的卷。

5.3.3 ReplicaInputStreams 类包含副本的数据和校验和的输入流。

5.3.4 ReplicaOutputStreams 类包含副本的数据和校验和的输出流。

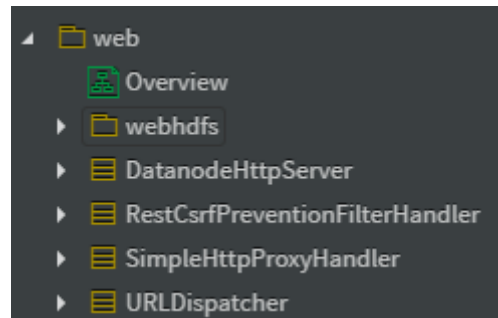
5.4 Metrics 包：总包括 5 个类和 1 个接口。



5.4.1 DataNodeMetrics 类用于维护各种 DataNode 统计信息并通过度量标准接口发布它们。这也为 RPC 注册了 JMX MBean。该类有许多可公开访问的度量变量；这些变量（对象）具有更新其值的方法；例如：{@link #blocksRead}.inc ()。

5.4.2 FSDatasetMBean 接口定义了获取数据节点的 FSDataset 状态的方法。它也用于通过 JMX 发布（因此我们遵循 JMX 命名约定。）注意我们没有使用 MetricsDynamicMBeanBase 来实现它，因为 FSDatasetMBean 的接口是稳定的，应该作为接口发布。数据节点运行时统计信息在另一个 MBean @see org.apache.hadoop.hdfs.server.datanode.metrics.DataNodeMetrics 中报告。

5.5 web 包：共包括 1 个包和 4 个类。



5.5.1 webhdfs 包：共包含 5 个类。

5.5.1.1 HdfsWriter 类为写入类。

5.5.1.2 DataNodeUGIProvider 类根据对 Web 的 SDHDFS 请求的请求创建 UGI。请注意，DN 不会对 UGI 进行身份验证- NN 将在后续操作中对其进行身份验证。

5.5.2 SimpleHttpProxyHandler 类是简单的会话层 HTTP 代理。它在上下文中获取 HTTP 响应，假设远程对等体快速合理且响应很小。上层应该过滤掉恶意输入。

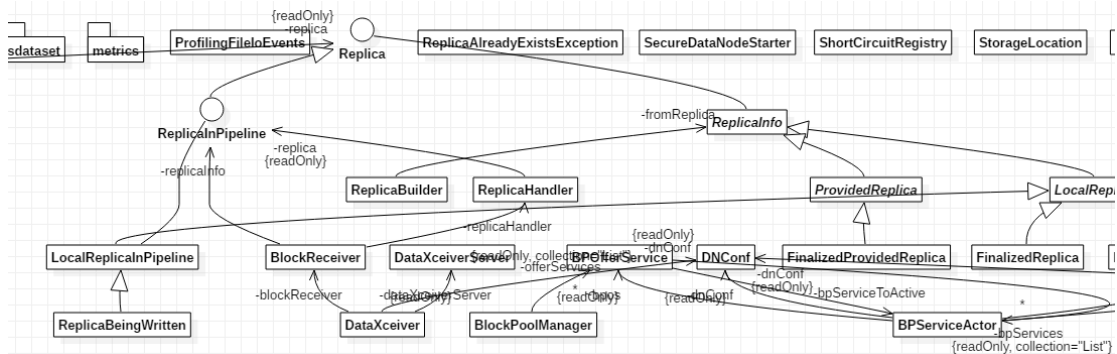
5.5.3 RestCsrfPreventionFilterHandler 类与{@link RestCsrfPreventionFilter}集成的 Netty 处理程序。如果过滤器确定允许请求，则此处理程序将请求转发到 Netty 管道中的下一个处理程序。否则，此处理程序将删除请求并立即发送 HTTP 400 响应。

5.6 DataNodeMXBean 接口是数据节点信息的 JMX 管理界面。最终用户不应该实现这些接口，而是通过 JMX API 访问此信息。

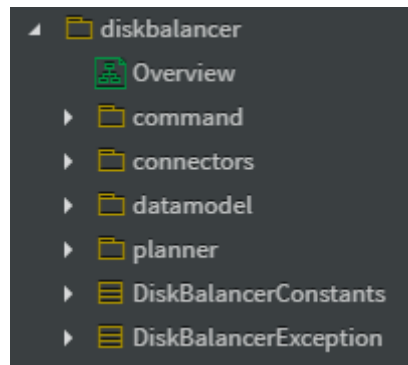
5.7 DataStorage 类是数据存储信息文件。

5.8 SecureDataNodeStarter 类用于在安全集群中启动 datanode 的实用程序类，首先在主启动之前获取特权资源并将其交给 datanode。

该包中类间关系的示意图如下图所示。特别地，由于该包内类和接口的数目较多，类间关系也较多，则所插该图是整个类图的节选。



6 diskbalancer 包：共包括 4 个包和 2 个类。

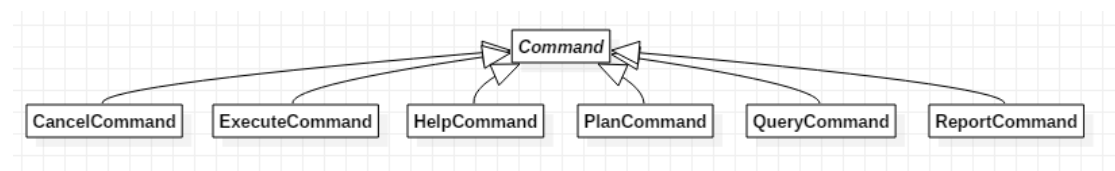


6.1 command 包：共包括 7 个类。

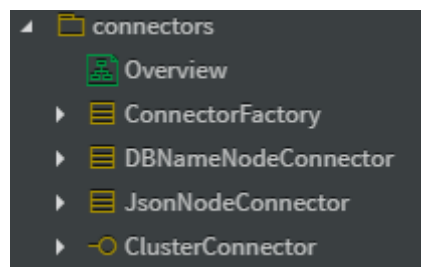
6.1.1 ReportCommand 类执行 report 命令。此命令将报告特定 DataNode 或前 X X DataNode 的卷信息，这些 DataNode 受益于运行 DiskBalancer。这是通过读取群集信息，通过 NodeDataDensity 对 DiskbalancerNodes 进行排序并打印出信息来完成的。

6.1.2 ExecuteCommand 类用于执行一个给定的计划。

该包中类间关系的示意图如下图所示。



6.2 connectors 包：共包括 3 个类和一个接口。



6.2.1 ClusterConnector 接口隐藏了有关我们如何与 HDFS 集群通信的所有细节。此接口返回 diskbalancer 理解的类中的数据。

6.2.2 JsonNodeConnector 类是一种了解 JSON 数据集群模型的连接器。

6.3 datamodel 包：共包括 4 个类。

6.3.1 DiskBalancerDataNode 类表示群集中存在的 DataNode。它还包含一个名为 nodeDataDensity 的度量，它允许我们在一组节点之间进行比较。

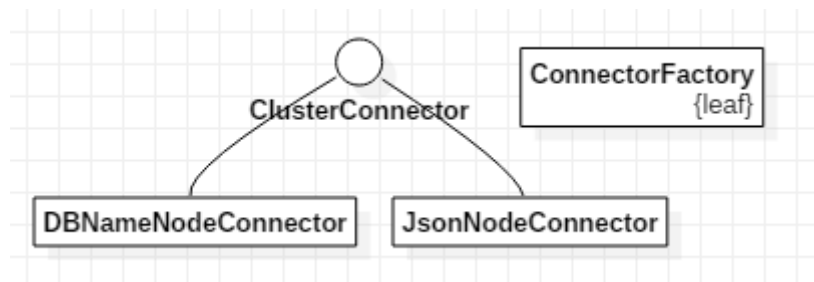
6.4 planner 包：共包括 4 个类和 2 个接口。

6.4.1 Planner 接口：Planner 界面允许创建不同的计划者。

6.4.1 PlannerFactory 类：返回基于用户定义标记的计划程序。

6.5 DiskBalancerConstants 类 Disk Balancer 使用的常量。

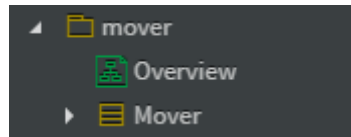
该包中类图如下图所示。



该包中类图如下图所示。

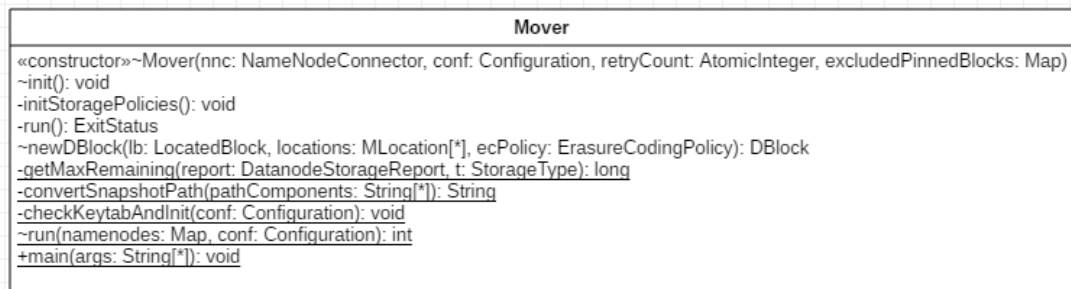


7 move 包：共包括 1 个类。

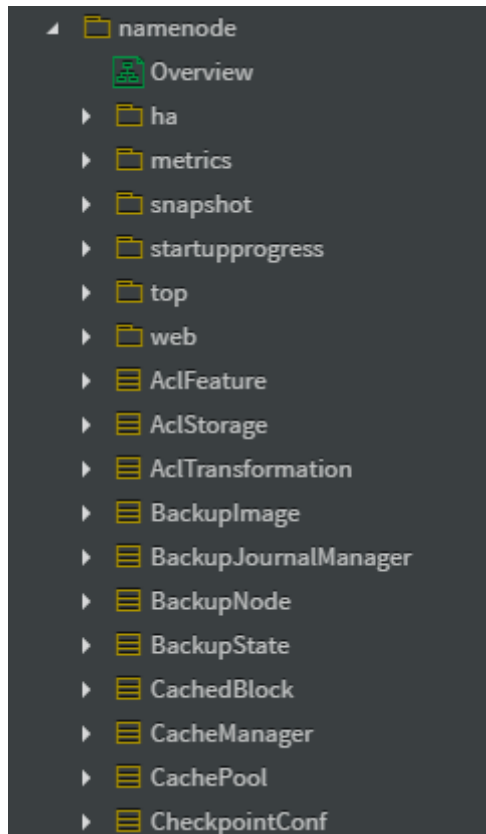


7.1 Mover 类用于数据的移动的类。

该包中类图如下图所示。



8 namenode 包：共包括 6 个包、6 个枚举类、14 个接口和 115 个类。



8.1 ha 包：共包含 8 个类和 1 个接口。

8.1.1 `ActiveState` 类是 namenode 的活动状态。在此状态下, namenode 提供 namenode 服务并处理类型为{@link `OperationCategory` #WRITE}和{@link `OperationCategory` #READ}的操作。

8.1.2 `EditLogTailer` 类表示一个线程, 它定期从编辑日志中读取并将包含在其中的事务应用于给定的 `FSNamesystem`。

8.2 metrics 包：共包括 1 个类和 3 个接口。

8.2.1 `ECBlockGroupsMBean` 接口定义了 在 `NameNode` 的 `FSNamesystem` 中获取与 {@link `org.apache.hadoop.hdfs.protocol.BlockType` #STRIPED}类型的块有关的状态的方法。它也用于通过 JMX 发布。 @see `FSNamesystemMBean` 中报告所有块的聚合状态 名称 节点 运行时 活动 统计 信息 在 @see `org.apache.hadoop.hdfs.server.namenode.metrics.NameNodeMetrics` 中报告。

8.2.2 `NameNodeMetrics` 类用于维护各种 `NameNode` 活动统计信息并通过 metrics 接口发布它们。

8.3 snapshot 包：共包括 15 个类和 1 个接口。

8.3.4 `FileWithSnapshotFeature` 类具有快照相关信息的文件的功能。

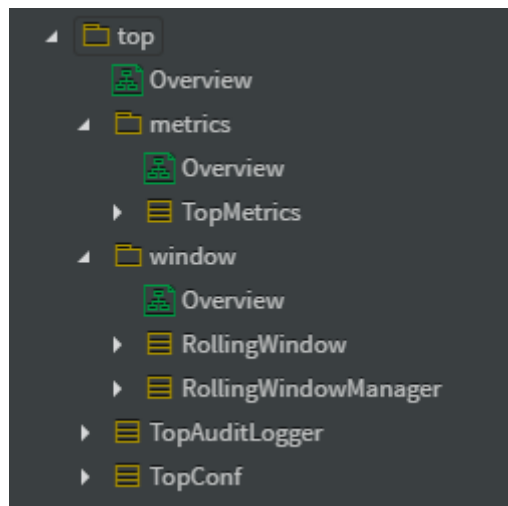
8.3.5 `Snapshot` 类是名称系统中的子树。

8.4 startupprogress 包：共包括 3 个枚举类和 8 个类。

8.4.1 AbstractTracking 类用于跟踪进度的内部数据结构的抽象基础。对于原始长属性，{@link Long # MIN_VALUE}用作标记值，表示该属性未定义。

8.4.2 StartupProgressMetrics 类 将 {@link StartupProgress} 链接到 {@link MetricsSource}以通过 JMX 公开其信息。

8.5 top 包：共包括 2 个包和 2 个类。



8.5.1 metrics 包：共包括 1 个类。

8.5.1.1 TopMetrics 类。

8.5.2 window 包：共包含 2 个类。

8.5.2.1 RollingWindow 类用于公开随时间发生的事件的滚动窗口视图的类。根据发生时间报告事件。滚动窗口覆盖的最后一个时段中的事件总数可以通过{@link #getSum (long) }方法检索。

8.5.3 TopAuditLogger 类{@link AuditLogger}，可将记录的数据直接发送到指标系统。顶级服务由名称节点直接使用时使用

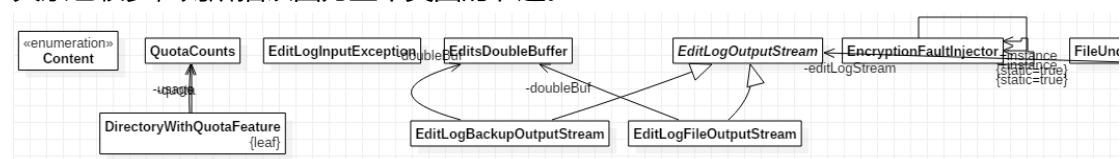
8.5.4 TopConf 类在是 NNTop 配置经常出现。

8.6 web 包：共包含 1 个包。

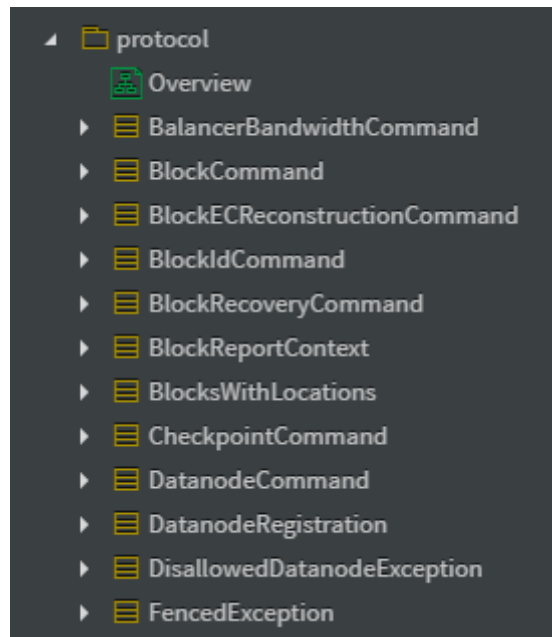
8.6.1 resources 包：共包含 1 个类。

8.6.1.1 NamenodeWebHdfsMethods 类为 Web-hdfs NameNode 实现。

该包中类间关系的示意图如下图所示。特别地，由于该包内类和接口的数目较多，类间关系也较多，则所插该图为整个类图的节选。



9 protocol 包：共包括 7 个枚举类和 30 个类。



9.1 StorageBlockReport 是阻止 Datanode 存储的报告类。

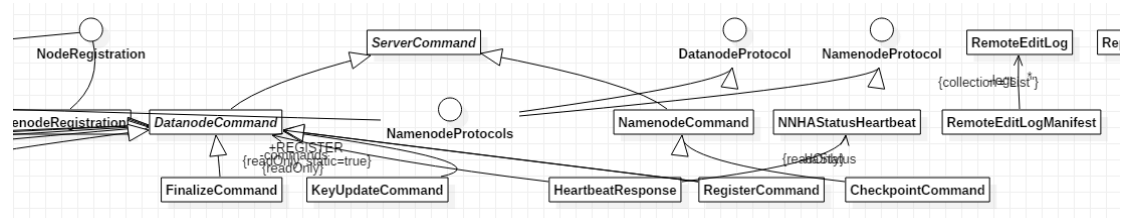
9.2 NamenodeCommand 类是 name-node 命令的基类。由 name-node 发出以通知其他名称节点应该做什么。

9.3 NamenodeProtocol 接口是辅助 NameNode 用于与 NameNode 通信的协议。它用于获取名称节点状态的一部分。

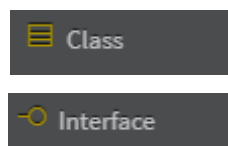
9.4 NamenodeRegistration 类是用于在注册过程中由下级名称节点发送到活动名称节点的信息。

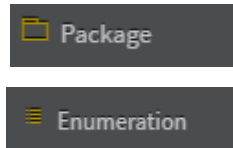
9.5 StorageReceivedDeletedBlocks 类是每个 Datanode 存储接收和删除块的报告。

该包中类间关系的示意图如下图所示。特别地，由于该包内类和接口的数目较多，类间关系也较多，则所插该图为整个类图的节选。



图例和注记如下所示。





五、设计特色分析

HDFS 架构在与同类产品相比之下，具有以下优点。

- 1) 高容错性：数据自动保存为多个副本，它通过增加副本的形式，提高容错性。某一个副本丢失以后，它可以自动恢复，这是由 HDFS 内部机制实现的，我们不必关心。
- 2) 适合批处理：移动计算而非数据，数据位置暴露给计算框架
- 3) 适合大数据处理：能够处理数据规模达到 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) 流式数据访问：一次写入，多次读取，不能修改，只能追加，它能保证数据的一致性。
- 5) 商用性：可构建在廉价机器上

由于 HDFS 架构的以上优点，其可以支撑的需求如下。

它所具有的高容错、高可靠性、高可扩展性、高获得性、高吞吐率等特征为海量数据提供了不怕故障的存储，为超大数据集的应用处理带来了很大便利。

HDFS 默认会将文件分割成 block，64M 为 1 个 block。然后将 block 按键值对存储在 HDFS 上，并将键值对的映射存到内存中。

通过分析可得，项目通过需要改进的地方如下。

- 1) 不适合低延时数据访问：它适合高吞吐率的场景，就是在某一时间内写入大量的数据。但是它在低延时的情况下是不行的，比如毫秒级的来存储数据，它很难做到。
- 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 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。小文件存储的寻道时间会超过读取时间，它违反了 HDFS 的设计目标。
- 3) 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。

六、组内分工情况

小组成员：庞振男、张椿溪

工作内容：阅读文献学习 Hadoop 分布式文件系统的组织结构、阅读设计文档和项目源码、学习逆向工程工具对系统进行分析、总结并制作思维导图、撰写软件体系结构文档等

分工情况:

庞振男: 阅读设计文档和项目源码、学习逆向工程工具对系统进行分析、撰写软件体系结构文档

张椿溪: 阅读文献学习 Hadoop 分布式文件系统的组织结构、总结并制作思维导图、撰写软件体系结构文档

七、参考文献

学习参考链接:

[1] 项目代码: <https://github.com/apache/hadoop/releases/tag/rel%2Frelease-3.1.2>

[2] 设计文档: <https://hadoop.apache.org/docs/r3.1.2/>

[3] Hadoop-hdfs 学习手册:

http://hadoop.apache.org/docs/r1.0.4/cn/hdfs_user_guide.html

[4] 百度百科 hdfs 讲解: <https://baike.baidu.com/item/hdfs/4836121?fr=aladdin>

[5] Hadoop-hdfs 设计文档:

<https://hadoop.apache.org/docs/r3.1.2/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html>

[6] Eclipse 逆向工程: https://wiki.eclipse.org/Java_reverse_engineering

[7] 思维导图参考 blog1:

<https://www.cnblogs.com/QuestionsZhang/p/10330068.html>

[8] 思维导图参考 blog2:

https://blog.csdn.net/cun_chen/article/details/50379489

[9] hdfs 框架及原理: <https://www.cnblogs.com/codeOfLife/p/5375120.html>

[10] Hdfs 源码分析 blog: <https://www.cnblogs.com/cxzdy/p/5034700.html>

参考文献:

[1] 《Hadoop Security Design》

<http://www.carfield.com.hk/document/distributed/hadoop-security-design.pdf>

[2] 《HDFS Architecture》

<http://www.corejavaguru.com/bigdata/hadoop/hdfs-architecture>

[3] 贡佩,晁玉蓉,樊华,崔超飞,陈伟.基于 Hadoop 的数据分析系统设计[J].数字技术与应用,2019(03):180+182.

[4] 罗青. Hadoop 平台下基于 HDFS 的小文件存储问题的优化与实现[D].华中科技大学,2019.

[5] 李晓玮.浅谈大数据 Hadoop 技术[J].电脑知识与技术,2017,13(32):10-11.

工具使用参考：

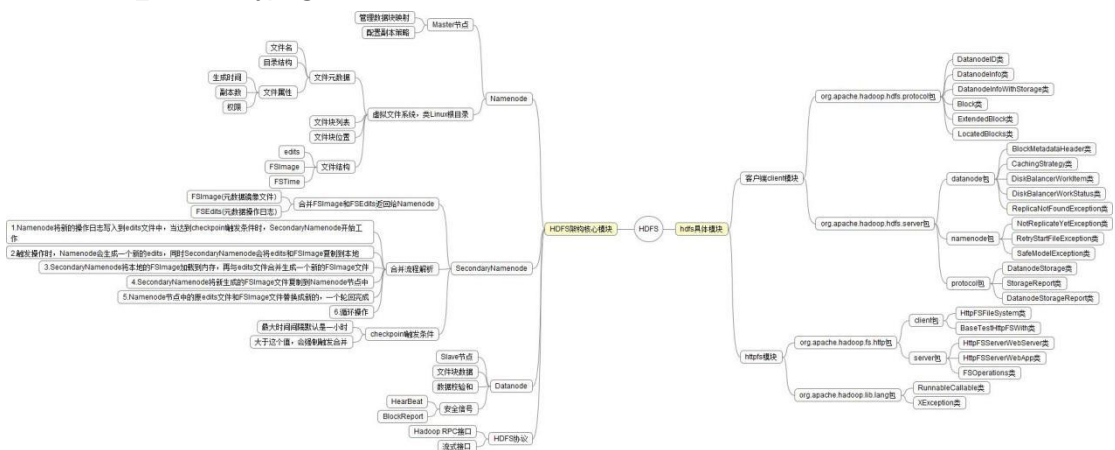
[1] starUML 使用参考：<https://blog.csdn.net/luansha0/article/details/82260678>
[2] starUML 官网：<http://staruml.io/>
[3] Eclipse 官网：<https://www.eclipse.org/>
[4] Hadoop 官网：<http://hadoop.apache.org/>
[5] Freemind 官网：<https://freemind.en.softonic.com/>
[6] 统计代码行数工具 Cloc 官网：<http://cloc.sourceforge.net/>

八、 附件：体系结构的分解过程

附件 1：思维导图

HDFS_freemind.mm

HDFS_freemid.jpeg



附件 2：相关 UML 图详见附件