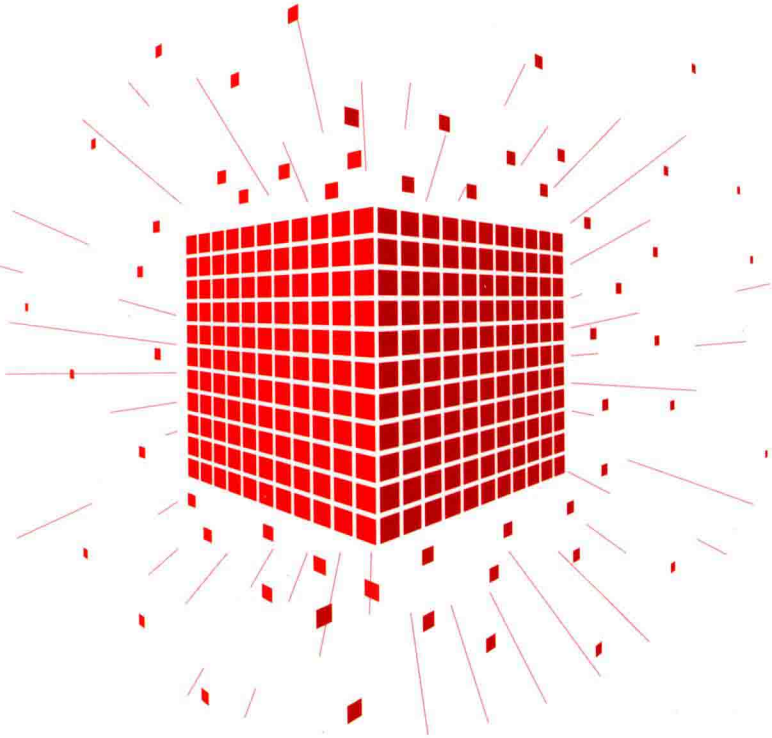
HZ BOOKS
华章IT

基于Hadoop 2.7.1版本，全面描述HDFS 2.X的核心技术与解决方案

对HDFS的几个主要使用场景进行了细粒度剖析，包括源码分析，融入了作者多年的开发经验

大数据技术丛书



An In-Depth Guide to Hadoop HDFS

深度剖析 Hadoop HDFS

林意群◎编著

机械工业出版社
China Machine Press

技术丛书

深度剖析 Hadoop HDFS

林意群◎编著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

深度剖析 Hadoop HDFS / 林意群编著 . —北京：机械工业出版社，2017.3
（大数据技术丛书）

ISBN 978-7-111-56207-8

I. 深… II. 林… III. 数据处理软件 IV. TP274

中国版本图书馆 CIP 数据核字（2017）第 040479 号

深度剖析 Hadoop HDFS

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）
责任编辑：吴 怡 责任校对：李秋荣
印 刷：北京诚信伟业印刷有限公司 版 次：2017 年 4 月第 1 版第 1 次印刷
开 本：186mm × 240mm 1/16 印 张：21
书 号：ISBN 978-7-111-56207-8 定 价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426 88361066 投稿热线：（010）88379604
购书热线：（010）68326294 88379649 68995259 读者信箱：hzit@hzbook.com

版权所有 · 侵权必究
封底无防伪标均为盗版
本书法律顾问：北京大成律师事务所 韩光 / 邹晓东

Preface 前　言

我上大学时，就开始在CSDN上写技术博客，目的在于记录平时遇到的一些问题以及研究的技术细节，好在将来可以进行查阅。随着时间的增长，我开始专注于某个技术模块，因为这样可以让我对具体某项技术有更深入的研究，写出的内容也会更加系统化，而HDFS就是其中一个我持续研究的技术模块。同时作为一名Hadoop社区的活跃贡献者，我也会将社区上一些比较有意思的东西分享到博客上，许多博友给了不少反馈，描述他们在工作中碰到的一些实际问题。在这样不断的写作、交流过程中，我得到了快速成长。目前大数据领域相关的书籍并不是很多，而专门讲解其中一个模块的书则更少，所以我将我过去一年多时间内关于HDFS的博客文章进行了整理、改进，同时也加入了一些新的内容。可以这么说，本书的内容源自博客，但是超越博客。

本书不会是纯源码分析的书籍。首先，我把工作实践中遇到的许多经验写入了书中，第7章便属于纯实践型的经验总结。其次，本书会是一个比较“新”的书，这里的“新”并不是指所分析的代码版本新，而是包含了HDFS未来的一些比较棒的功能特性，以及Hadoop社区目前在做的一些事情。在这本书中，你会看到许多与社区相关的JIRA，了解如何从社区上找到问题的解决办法。期待本书能给你带来更多的启发。

本书适合具有一定Java语言基础的同学，尤其适合以下读者朋友：

- 大数据架构师、开发者、运维工程师。
- 高年级本科生或研究生。
- 热衷于分布式存储技术的爱好者。

本书分为三大部分，“核心设计篇”介绍HDFS的基本原理、数据管理与策略等，“细节实现篇”介绍HDFS的块处理、流量处理、结构分析等，“解决方案篇”介绍数据管理技术与方案、数据读写技术、异常处理等。

第一部分“核心设计篇”包括内容如下：

第1章介绍HDFS现有的数据存储方式，主要介绍其中的内存存储和异构存储两个方面。

第 2 章介绍 HDFS 目前内部几种主要的功能机制，包括缓存管理、快照管理等。

第 3 章介绍 HDFS 比较新颖的一些功能，以及目前较少被人用到的功能特性。

第二部分“细节实现篇”包括内容如下：

第 4 章介绍 HDFS 的块处理相关操作，主要处理场景包括块如何组织、上报处理的过程以及多余块的清除。

第 5 章介绍 HDFS 的流量处理过程，包括 HDFS 目前流量处理的场景以及 Balancer 工具的数据平衡原理和优化。

第 6 章介绍 HDFS 一些特殊的结构对象类，包括这些类的作用、原理以及运用场景。

第三部分“解决方案篇”包括内容如下：

第 7 章介绍与 HDFS 相关的多套运维管理的操作方案，包括数据迁移、数据监控等方面。

第 8 章介绍 HDFS 写磁盘时的一些优化策略和改造方案。

第 9 章介绍 HDFS 的一些异常场景，并给出了相应的解决方案。

由于笔者水平有限，本书难免会有出错或者介绍不明确的地方，恳请读者批评指正，可以发送关于本书的意见和建议到我的个人邮箱：yqtin@apache.org。本书所涉及的源码，大家可以从 Hadoop 的 Git 地址上进行下载：https://github.com/apache/hadoop，其中，不同的分支对应不同版本的代码。相关 Git 地址和 CSDN 博客地址如下：

- Git 地址：https://github.com/linyiqun
- CSDN 地址：http://blog.csdn.net/androidlushangderen

感谢机械工业出版社的吴怡编辑，在我写作的过程中，不断指出其中的不足之处，督促和引导我完成本书的编写。

感谢蘑菇街数据平台部的同事们，在工作中不断地给予我帮助和支持，协助我解决各种各样的问题，于是才有了本书中所展现的精彩内容。

林意群

2017 年 2 月

试读结束，需要全本PDF请购买 www.ertongbook.com

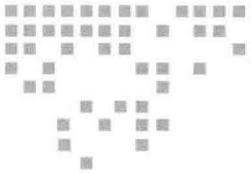
Contents 目　录

第二部分　细节实现篇

试读结束，需要全本PDF请购买 www.ertongbook.com

第一部分 Part 1

核心设计篇



Chapter 1 第 1 章

HDFS 的数据存储

本章将从 HDFS 的数据存储开始说起，因为正是先有了数据的存储，才有后续的写入和管理等操作。HDFS 的数据存储包括两块：一块是 HDFS 内存存储，另一块是 HDFS 异构存储。HDFS 内存存储是一种十分特殊的存储方式，将会对集群数据的读写带来不小的性能提升，而 HDFS 异构存储则能帮助我们更加合理地把数据存到应该存的地方。

1.1 HDFS 内存存储

HDFS 的内存存储是 HDFS 所有数据存储方式中比较特殊的一种，与之后将会提到的 HDFS 缓存有一些相同之处：都用机器的内存作为存储数据的载体。不同之处在于：HDFS 缓存需要用户主动设置目标待缓存的文件、目录，其间需要使用 HDFS 缓存管理命令。而 HDFS 内存存储策略：LAZY_PERSIST 则直接将内存作为数据存放的载体，可以这么理解，此时节点的内存也充当了一块“磁盘”。只要将文件设置为内存存储方式，最终会将其存储在节点的内存中。综合地看，HDFS 缓存更像是改进用户使用的一种功能，而 HDFS 内存存储则是从底层扩展了 HDFS 的数据存储方式。本节将对 HDFS 内存存储策略进行更细致的分析。

1.1.1 HDFS 内存存储原理

对于内存存储的存储策略，可能很多人会存有这么几种看法：

- 数据临时维持在内存中，服务一停止，数据全部丢失。

- 数据存在于内存中，在服务停止时做持久化处理，最终将数据全部写入到磁盘。

仔细来看以上这 2 种观点，其实都有不小的瑕疵：

- 第一个观点，服务一旦停止，内存数据全部丢失，这是无法接受的，我们只能容忍内存中少量的数据丢失。这个观点的另一个问题是，内存的存储空间是有限的，在服务运行过程中如果不及时处理一部分数据，内存空间迟早会被耗尽。
- 第二个观点，在服务停止退出的时候做持久化操作，同样会面临上面提到的内存空间的限制问题。如果机器的内存足够大，数据可能会很多，那么最后写入磁盘的阶段速度会很慢。

所以一般情况下，通用的、比较好的做法是异步持久化，什么意思呢？在内存存储新数据的同时，持久化距离当前时刻最远（存储时间最早）的数据。换一个通俗的解释，好比有个内存数据块队列，在队列头部不断有新增的数据块插入，就是待存储的块，因为资源有限，需要把队列尾部的块，也就是更早些时间点的块持久化到磁盘中，这样才有空间存储新的块。然后形成这样的一个循环，新的块加入，老的块移除，保证了整体数据的更新。

HDFS 的 LAZY_PERSIST 内存存储策略用的就是这套方法，原理如图 1-1。

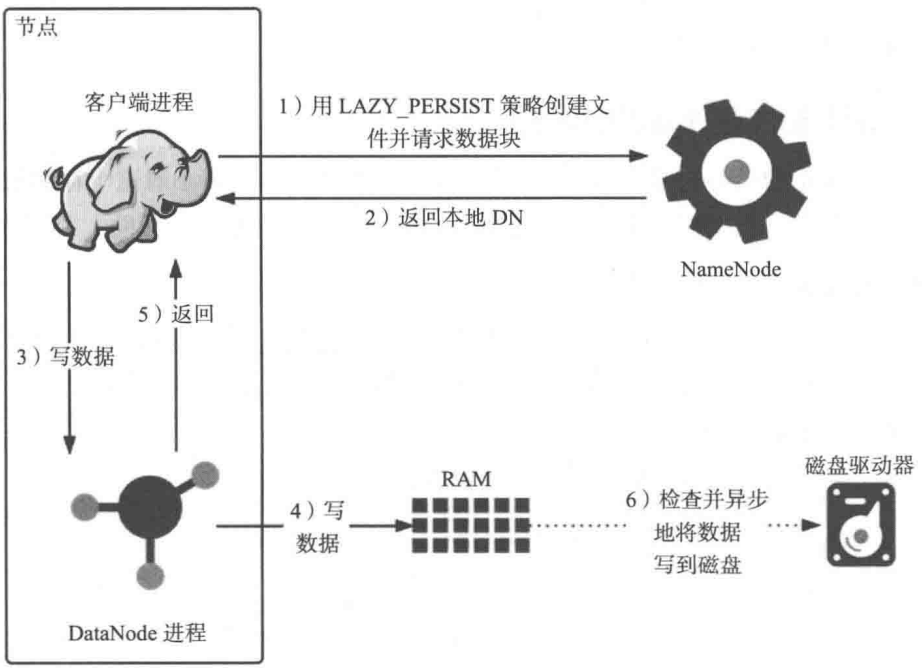


图 1-1 LAZY_PERSIST 策略原理图

上面描述的原理在图中的表示是第 4 个步骤和第 6 个步骤。第 4 步写数据到内存中，第 6 步异步地将数据写到磁盘。前面几个步骤是如何设置 StorageType 的操作，在下文中会

具体提到。所以异步存储的大体步骤可以归纳如下：

1）对目标文件目录设置 StoragePolicy 为 LAZY_PERSIST 的内存存储策略。

2）客户端进程向 NameNode 发起创建 / 写文件的请求。

3）客户端请求到具体的 DataNode 后 DataNode 会把这些数据块写入 RAM 内存中，同时启动异步线程服务将内存数据持久化写到磁盘上。

内存的异步持久化存储是内存存储与其他介质存储不同的地方。这也是 LAZY_PERSIST 名称的源由，数据不是马上落盘，而是懒惰的、延时地进行处理。

1.1.2 Linux 虚拟内存盘

这里需要了解一个额外的知识点：Linux 虚拟内存盘。之前笔者也一直有个疑惑，内存也可以当作一个块盘使用？内存不就是临时存数据用的吗？于是在学习此模块知识之前，特意查了相关的资料。其实在 Linux 中，的确有将内存模拟为一个块盘的技术，叫**虚拟内存盘**（RAM disk）。这是一种模拟的盘，实际数据都是存放在内存中的。虚拟内存盘可以在某些特定的内存式存储文件系统下结合使用，比如 tmpfs、ramfs。关于 tmpfs 的具体内容，大家可以查阅维基百科等资料。通过此项技术，我们就可以将机器内存利用起来，作为一块独立的虚拟盘供 DataNode 使用了。

1.1.3 HDFS 的内存存储流程分析

下面讲述本章的核心内容：HDFS 内存存储的主要流程。不要小看这个存储策略，里面的过程可并不简单，在下面的内容中，笔者会给出比较多的过程图，帮助大家理解。

1. HDFS 文件内存存储策略设置

要想让文件数据存储到内存中，一开始要做的操作是设置此文件的存储策略，即上面提到的 LAZY_PERSIST，而不是使用默认的存储策略：StoragePolicy.DEFAULT，默认策略的存储介质是 DISK 类型的。设置存储策略的方法目前有以下 3 种：

第一种方法，通过命令行的方式，调用如下命令：

```
hdfs storagepolicies -setStoragePolicy -path <path> -policy LAZY_PERSIST
```

这种方式比较方便、快速。

第二种方法，调用对应的程序方法，比如调用暴露在外部的 create 文件方法，但是得带上参数 CreateFlag.LAZY_PERSIST。如下所示：

```
FSDataOutputStream fos =
        fs.create(
            path,
            FsPermission.getFileDefault(),
```

```
            EnumSet.of(CreateFlag.CREATE, CreateFlag.LAZY_PERSIST),
            bufferLength,
            replicationFactor,
            blockSize,
            null);
```

上述方式最终调用的是 DFSClient 的 create 同名方法，如下所示：

```
// DFSClient 创建文件方法
public DFSOutputStream create(String src, FsPermission permission,
    EnumSet<CreateFlag> flag, short replication, long blockSize,
    Progressable progress, int buffersize, ChecksumOpt checksumOpt)
    throws IOException {
  return create(src, permission, flag, true,
      replication, blockSize, progress, buffersize, checksumOpt, null);
}
```

方法经过 RPC 层层调用，经过 FSNamesystem，最终会到 FSDirWriteFileOp 的 startFile 方法，在此方法内部，会有设置存储策略的动作：

```
static HdfsFileStatus startFile(
    FSNamesystem fsn, FSPermissionChecker pc, String src,
    PermissionStatus permissions, String holder, String clientMachine,
    EnumSet<CreateFlag> flag, boolean createParent,
    short replication, long blockSize,
    EncryptionKeyInfo ezInfo, INode.BlocksMapUpdateInfo toRemoveBlocks,
    boolean logRetryEntry)
    throws IOException {
  assert fsn.hasWriteLock();

  boolean create = flag.contains(CreateFlag.CREATE);
  boolean overwrite = flag.contains(CreateFlag.OVERWRITE);
  // 判断 CreateFlag 是否带有 LAZY_PERSIST 标识，来判断是否是内存存储策略
  boolean isLazyPersist = flag.contains(CreateFlag.LAZY_PERSIST);

  ...
  // 在此设置策略
  setNewINodeStoragePolicy(fsd.getBlockManager(), newNode, iip,
                           isLazyPersist);
  fsd.getEditLog().logOpenFile(src, newNode, overwrite, logRetryEntry);
  if (NameNode.stateChangeLog.isDebugEnabled()) {
    NameNode.stateChangeLog.debug("DIR* NameSystem.startFile: added " +
        src + " inode " + newNode.getId() + " " + holder);
  }
  return FSDirStatAndListingOp.getFileInfo(fsd, src, false, isRawPath);
}
```

这部分的过程调用见图 1-2。

试读结束，需要全本PDF请购买 www.ertongbook.com