**目录**

[1. 概述 1](#_Toc467338197)

[2. 详细 1](#_Toc467338198)

[2.1分解 1](#_Toc467338199)

[2.2小结 1](#_Toc467338200)

[3. 存储相关的数据结构 1](#_Toc467338201)

[3.1 NamespaceDetails与xxx.ns文件 1](#_Toc467338202)

[ DiskLoc 1](#_Toc467338203)

[ MemoryMappedFile 1](#_Toc467338204)

[ DurableMappedFile 1](#_Toc467338205)

[ DeleteRecord 1](#_Toc467338206)

[ NamespaceDetail 1](#_Toc467338207)

[3.2Extent相关结构 1](#_Toc467338208)

[ Extent 1](#_Toc467338209)

[ Record 1](#_Toc467338210)

[4. 参考资料 1](#_Toc467338211)

**jar cvfm MyWordCount.jar manifest.data \*.class**

**你好**

本篇源码为MongoDB2.6.12，以阅读Linux环境为准。

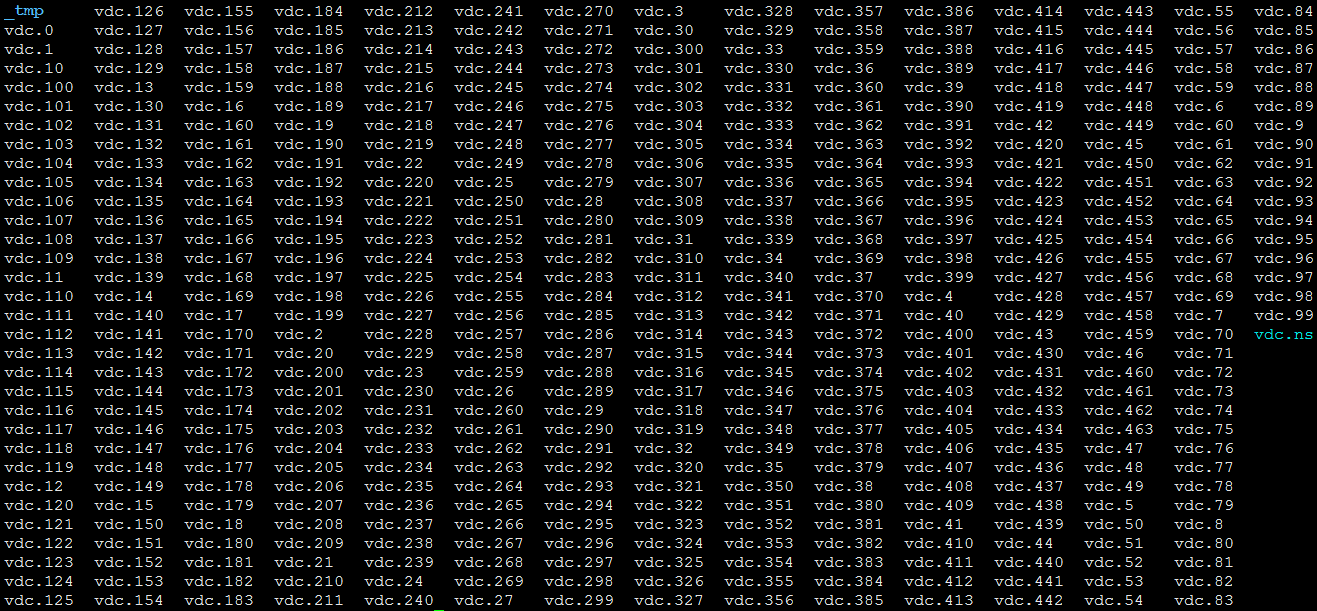
# 概述

每个数据库都有独立文件，开启directoryperdb选项后，这些每个数据库的文件会被放在独立的目录下。

数据库文件在内部被切分成块，每个块只保存一个namespace的数据。名字空间用于区分不同的存储类别，比如collection有独立的namespace，每个索引也有独立的namespace。

在一个块中会保存多条记录，每条记录是BSON格式的，记录与记录之前通过双向链表连接。**索引数据也存在数据文件中**，不过索引是被组织成B-Tree结构。

对于每个数据库，有一个命名空间ns文件，用于保存每个名字空间对应的元数据。通过这些元数据来找到名字空间中的存储块位置。举例，我的数据库是vdc，数据目录是：/home/mongodb2.4/data/vdc/，这个目录下的文件结构及信息如下：



这些文件的分配顺序是vdc.0，vdc.1，vdc.2，vdc.3，……。大小是64M，128M，256M，512M，1024M，2048M，2048M，……。其中vdc.ns文件一开始是16M，后面随着元信息数量的增加，vdc.ns文件会逐步变大，大小是不是按照16M整数倍增长暂时不确定（猜测应该不是），当前vdc.ns文件的大小是500M。

如果开启了日志等选项，还有其他文件生成。下面逐步分析这些文件的结构。

# 详细

## 2.1分解

每个数据库有相应的数据文件（数据+索引）和命名空间文件，数据文件从16MB开始分配，新的数据文件比之前大一倍，最大为2GB。如下：



文件使用MMAP进行内存映射，会将**所有**的数据文件映射到**虚拟内存**中，当实际访问到时数据被载入到物理内存进行相应的运算处理。

在32bit的机器上，每个进程由4GB的虚存地址空间，1GB内核用，大约0.5GB被mongod的stack占用，剩下大约2.5GB的空间用于映射数据文件。如下：



在64bit机器上（0x0000000000000000-0x0000ffffffffffff是用户空间，这个大小达到256TB），这个空间将非常大。每个数据文件被切分为一个个小块，块之间通过双向链表连接。在名字空间文件中，保存了一些元信息，包括大小，块数，第1块的位置，最后一块的位置，被删除的块的链表及索引信息。

在**名字空间文件**中，保存了每个名字空间的存储信息，大致如下：



在**数据文件**中，基本单位是**Extent**，MongoDB把数据和索引组织到Extent中，如下是一个数据文件的概要结构：



1. 每个文件有多个Extent，每个collection有多个Extent；
2. 每个Extent只会包含一个collection的数据**或者**索引；
3. 同一个collection的数据**或**索引会分布在多个Extent内，这些Extent也可以分布在多个文件内；
4. 同一个Extent不会既有数据又有索引；
5. 每一个Namespace（collection或者index）的入口在ns文件中，从此可以得到第一个extent的位置信息。

每个Extent中放着多个**Record**，这些Record会被Extent中的结构串接成双向链表，如下：



每个Record中包含了一个记录头和BSON文档以及一些额外的padding信息（额外分配的未用空间，如果文档变大时不至于将文档迁移到别处）。记录的结构大致如下：



数据库中的一些大小参数含义：



所以当删除document或者collection，抑或是index，fileSize的大小不会发生变化。因此数据库所使用磁盘空间的大小会不断增加，但是这不意味着浪费，删除后的空间表示着预留的空间多了而已。这些预留空间会在后面插入新的record时使用。

另外，可以通过使用--smallfiles和--nopreallocate选项，让数据文件小一点且不预分配文件，这样就不会有太多的预留空间，整体占用就会小一点了。

如果collection和index的数量非常庞大，可以使用--nsize来扩充ns文件的大小。

## 2.2小结

回头来看看整体结构：



通过这些信息，整个document的结构就完全串联起来了。

# 存储相关的数据结构

## 3.1 NamespaceDetails与xxx.ns文件



如上图是ns文件相关的所有的数据结构，内存映射文件部分是直接映射，结构相对简单。核心结构是NamespaceDetails（其中省略部分成员）：

**class NamespaceDetails {**

**DiskLoc \_firstExtent;**

**DiskLoc \_lastExtent;**

**/\* 该表是否是capped，capped-table是ring-buffer类型的table，MongoDB中用来存放oplog \*/**

**int \_isCapped;**

**/\* 每个元素关联1个free\_record\_list链表，大小各不相同。capped类型的collection中\_deleteList的含义略有不同。由此可知每个ns包含这样1个链表 \*/**

**DiskLoc \_deletedList[19];**

**};**

PS:在capped collection（固定集合）中，\_deleteList[0]串联所有free record。\_deleteList[1]指向prev\_extent的last record，当current extent变更时，\_deleteList[1]会被更新。

根据各个成员的含义，ns文件与数据文件之间的大致关系如下：



根据如上数据结构的关系以及文件关系，我们知道每个表都有一个\_deleteList来维护已经删除的空间。

如下是上述涉及的到结构的主要成员（忽略部分成员）的含义：

### DiskLoc

该类定义在diskloc.h/record.cpp文件中。

吐槽：在定义和实现的文件方面，MongoDB里面有很多这样的情况，不知道代码编写者是怎么想的，头文件和实现文件名称不一致。

**class DiskLoc {**

**/\* 文件编号，数据文件以db.0,db.1,db.2,…形式存储 \*/**

**int \_a;**

**/\* 文件中的偏移量 \*/**

**int ofs;**

**};**

### MemoryMappedFile

该类定义在mmap.h/mmap\_posix.cpp中。

**class MemoryMappedFile {**

**/\* 打开后文件的文件描述符，HANDLE被定为int类型 \*/**

**HANDLE fd;**

**/\* 该变量在win平台下使用，linux中不用 \*/**

**HANDLE maphandle;**

**/\* 数据库有多个数据文件，每个文件都进行内存映射，views保存每个文件映射的指针 \*/**

**std::vector<void\*> views;**

**/\* 文件大小 \*/**

**unsigned long long len;**

**};**

### DurableMappedFile

该类定义在durable\_mapped\_file.h/cpp文件中，继承自MemoryMappedFile类。该文件中同时定义了PointerToDurableMappedFile类，该类用map<void\*, DurableMappedFile\*> \_views来记录所有的文件映射信息。

**class DurableMappedFile {**

**/\* 内存文件映射返回的指针 \*/**

**void\* \_view\_write;**

**/\* 创建的内存文件映射时MAP\_PRIVATE时，用该成员 \*/**

**void\* \_view\_private;**

**bool \_willNeedRemap;**

**RelativePath \_p;**

**int \_fileSuffixNo;**

**};**

PS：**MAP\_PRIVATE**标志创建的内存区域是写时复制属性，映射区域的修改不会影响原始文件，映射区域修改时会复制一份文件进行修改，；**MAP\_SHARED**标志表示映射区域为所有进程共享，映射区域与文件内容保持一致（未更新时除外）。

问题：写时复制技术，通常用于进程fork时，而此处内存映射文件用到了MAP\_PRIVATE写时复制是什么样的背景呢？

### DeleteRecord

在record.h/cpp文件中，该结构用于将所有空闲的record连接起来。

**class DeleteRecord {**

**/\* 该record空间的大小 \*/**

**int \_lengthWithHeaders;**

**/\* 该record在extent中的偏移量，该成员有什么用呢？ \*/**

**int \_extentOfs;**

**/\* 下一个空闲record的位置 \*/**

**DiskLoc \_nextDeleted;**

**};**

### NamespaceDetail

在中namespace\_details.h/cpp中。

**class NamespaceDetail {**

**/\* 第1个Extent的位置 \*/**

**DiskLoc \_firstExtent;**

**/\* 最后1个Extent的位置 \*/**

**DiskLoc \_lastExtent;**

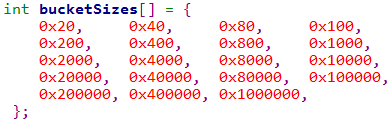
**/\* 空闲record的链表，每个元素依次是32byte~16M大小，每个元素关联1个链表 \*/**

**DiskLoc \_deleteList[19];**

**... ...**

**}**;

\_deleteList[19]每个元素维护的的链表的大小依次是：



所有>=4M的record都会被放在最后1个链表维护，因此\_deleteList[0]则表示[,32byte)，依次类推\_deleteList[17]表示[,4M)的链表。

## 3.2Extent相关结构

以Extent为核心的结构大致如下：



如下是从ns-extent-record三者关联的角度分析结构关系：



问题：

Extent结构中会有一个last record记录，这个last record记录的是整个collection的最后一个record的位置。如果最后一个record更新时，岂不是每一个Extent中的record都要更新么？这样是不是代价太大，还是只更新第一个Extent中的last record。

### Extent

在文件extent.h/cpp中。

**class Extent {**

**DiskLoc magic;**

**/\* 同一namespace中的xnext\_extent,xprev\_extent \*/**

**DiskLoc xnext, xprev;**

**/\* 该extent所属的namespace \*/**

**Namespace nsDiagnostic;**

**/\* extent结构大小 \*/**

**int length;**

**/\* first record位置 \*/**

**DiskLoc firstRecord;**

**/\* last record位置 \*/**

**DiskLoc lastRecord;**

**/\* 不知道什么用处，没有找到该成员的初始化地方 \*/**

**char \_extentData[4];**

**};**

### Record

在record.h/cpp文件中。

**class Record {**

**/\* record的大小 \*/**

**int \_lengthWithHeaders;**

**/\* record在extent中的偏移量 \*/**

**int \_extentOfs;**

**/\* 下1个record的偏移量 \*/**

**int \_nextOfs;**

**/\* 前1个record的偏移量 \*/**

**int \_prevOfs;**

**/\* record位置 \*/**

**char \_data[4];**

**};**

# 参考资料

1. http://www.thebigdata.cn/MongoDB/29695.html
2. http://www.mongoing.com/blog/file-storage
3. http://www.wtoutiao.com/p/15aGGng.html
4. https://www.qcloud.com/community/article/137
5. http://www.slideshare.net/mongodb/mongodb-london-2013understanding-mongodb-storage-for-performance-and-data-safety-by-christian-kvalheim-10gen