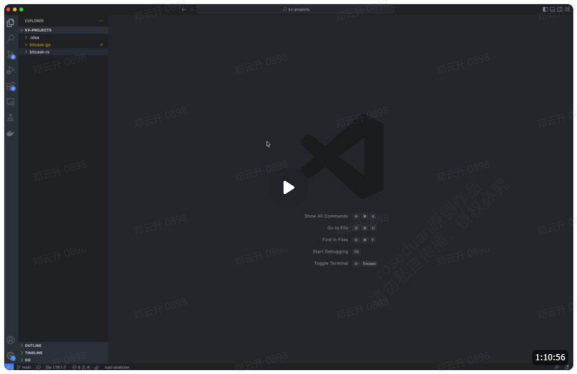
12 从零实现 KV 存储—Merge 数据清理

理论讲解

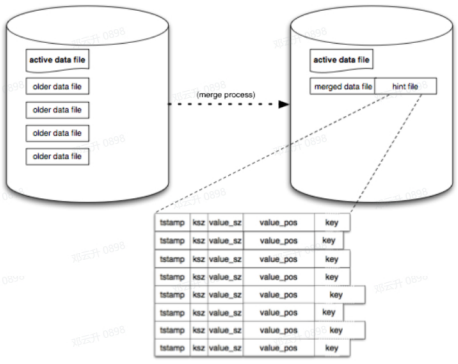
Go 编码部分

Rust 编码部分



Merge 概览

merge 的主要功能是清理磁盘上的无效数据，避免由于数据逐渐增加，并且无效数据过多，导致 bitcask 存储引擎的数据目录空间的膨胀。



bitcask 论文中对 Merge 只是做了简单的描述，但是怎么去实现还需要我们仔细斟酌，Merge 的主要设计目标有两个点，一是清理旧的数据，重写有效的数据，二是生成只包含索引的 hint 文件，并且我们需要尽量保证这个过程不对前台正常的读写造成太大的影响。

清理无效数据

对于第一点，将磁盘上的无效数据清理掉，具体的做法其实有很多种。

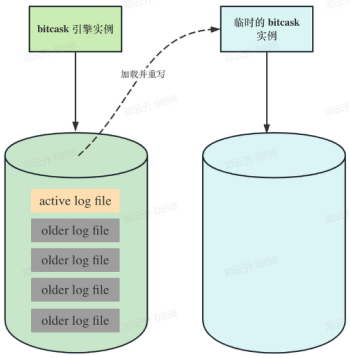
我们可以按编号从小到大读取数据文件，并且依次取出其中的每一条日志记录，然后跟内存中的索引进行比较，如果和内存索引对应的文件 id 和偏移 offset 一致，说明这是有效的数据，然后直接调用 Put 接口重写这条数据即可，一个文件中的数据重写完了，就将其删除掉。

但是这样会使用面向外层调用者的 Put 接口，增加这个方法的锁竞争。

这样做还有一个不太好处理的点，就是如果一个文件中的数据写到了一半，但是出错了，这时候新的数据文件中有新重写的数据，而旧的文件又不能删除掉。解决这个问题的一个方法是将一个文件中的数据全部放到一个事务中执行，

只有当事务提交成功之后，才能够删除文件。但是如果一个文件中的数据太多的话，提交事务之前，一般会将数据批量缓存到内存中，这样可能会造成内存容量的膨胀。

所以我们可以换一种思路，使用一个临时的文件夹，假如叫 merge，在这个临时目录中新启动一个数据库实例，这个实例和正在运行的数据库实例互不冲突，因为它们是不同的进程。将原来的目录中的数据文件逐一读取，并取出其中的日志记录，和内存索引进行比较，如果是有效的，我们将其重写到 merge 这个数据目录中，避免和原来的目录竞争。



这样一来，我们重写数据的时候，并不会影响到原来的数据目录上的 bitcask 引擎实例，对用户正常写数据的影响降到了最低。

hint 索引文件

再来看第二点，如何生成 hint 索引信息，这个其实比较简单，我们在重写数据到 merge 目录的时候，实际上还是会得到一个位置索引信息，将这个索引和原始的 key 保存到一个新的 hint 文件里就可以了，这个 hint 文件我们还是可以沿用数据文件的结构，也采用日志追加的方式。

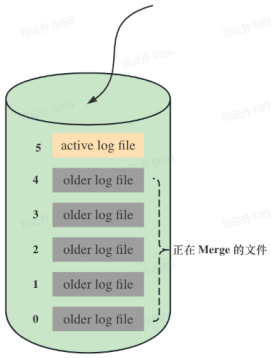
重启校验 merge

merge 的时候还需要考虑一个点，就是在 merge 的过程当中，如果出现了异常，例如进程退出，或者系统崩溃等，导致 merge 没有完成，我们应该如何处理这种情况？

一个简单直观的办法是，我们可以在数据全部重写完成后，在磁盘上增加一个标识 merge 完成的文件，当重启数据库的时候，我们查看其是否有对应的 merge 目录，找到了这个目录之后，说明发生过 merge，然后在这个目录中查找是否有 merge 完成的文件，如果没有的话，则说明这是一次无效的 merge，我们直接将 merge 这个目录删除掉。

如果是有效的 merge，那么这个目录中的数据文件，都是存放的全部有效的数据，然后还有一个对应的 hint 索引文件。我们需要将这些数据文件拷贝到原始数据目录中，然后把对应的 hint 索引文件也拷贝过去，并且把临时的 merge 目录删除掉，这样下次启动的时候，便能够和原来的正常启动保持一致了。

还需要注意的一个细节是，我们添加的这个标识 merge 完成的文件中应该记录什么内容呢，或者是空的就可以？



回想这样一个问题，在 merge 的过程当中，可能又有新的数据写进来了，这部分没有参与 merge 的数据，在加载索引的时候，还是需要和原来一致，并不能从 hint 文件中加载。还有便是前面提到了，我们应该将旧的文件删除掉，把 merge 完成的数据文件替换过来，但如果是 merge 过程中新生成的数据文件，我们肯定不能删除掉，所以还是需要知道最近的一个没有参与 merge 的文件的 id。

所以在 merge 完成标识的这个文件中，我们可以记录这个文件 id，在后续系统启动的时候能够使用。