分配3：故障恢复

分配日期：2021年2月25日，星期四
截止日期：2021年3月18日，星期四，晚上11：59（午夜）
开始代码：https://github。未来数据/cs245-作为3个公开数据
链接到课堂评分表：https://www。gradescope.com/courses/218759

概述

在我们关于故障恢复的讲座中，[1]在[2]中，我们讨论了确保数据库的一致性和容错性的技术。在此分配中，您将使用预写重做日志记录结合持久存储来实现一个简单的键值存储的故障恢复。其结果将是一个具有原子事务的数据库，该事务可以在崩溃后恢复，同时保证外部一致性。您还将实现日志截断，以减少崩溃后恢复所需的工作量。

我们已经为您实现了预写日志和一个持久性存储引擎，您不应该修改这些实现。我们还首先使用了一个事务管理器，以确保事务是原子性的。您在此分配中进行的修改将增加事务管理器的持久性，以便它可以在崩溃后恢复。

为了测试您的解决方案，我们将通过将“持久化”数据保存在日志中，并将存储引擎保存在内存中来模拟持久性。未持久化的数据和事务管理器都会在模拟崩溃期间被销毁。测试代码将重新初始化您实现的事务管理器，它应该从已保存的数据恢复任何提交的写操作。

设置

软件依赖关系

- 与赋值1中一样，您应该使用Java 11.如果IDE提示您选择JUnit的一个版本，则应该选择JUnit4。

*强烈建议您通过像IntelliJ这样的IDE来运行测试。*您将得到更好的日志输出和堆栈跟踪。为此，右键单击IDE中的测试或整个文件，然后单击“运行”。”

若要构建代码并在命令行中（而不是在IDE中）上运行测试，请运行

```
mvn包
Xms1536m-jartarget/tests.jar
```

若要提交代码，请运行

```
create_submission.sh
```

这将创建一个名为student_submission.zip的文件，您应该将其上传到梯度望远镜。

第一部分（你们成绩的70%）：实施长期交易

对于这部分，您将专门修改事务管理器。java(在
/as3)，以实现容错能力。我们提供的起始代码公开了以下API，您应该保留这个API及其所有属性
：

- ●启动（长txID）
  - ○指示新事务的开始。我们将保证txID总是增加的（即使是在崩溃中）。
- ●读取(长txID，长键)
  - ○通过任何事务返回密钥的最新提交值。
- ●写入（长txID，长密钥，字节[]值）
  - ○表示对数据库的写入操作。注意，在提交写的事务之前，这样的写操作对于读取()调用不可见。为简单起见，我们在写入键后不会从txID本身读取相同的键。
- ●提交(长txID)
  - ○提交一个事务，并使其写入操作对后续的读取操作可见。
- ●中止(长txID)
  - ○中止交易。
- ●写入持久化（长键、长标记、字节[]值）
  - ○您现在可以忽略此呼叫；我们将在作业的第二部分中描述它。

此外，您必须实现一种方法，即事务管理器的初始化和恢复API：

- ● initAndRecover (LogManager lm, StorageManager sm)
  - ○使用提供的日志管理器和存储管理器初始化事务管理器。我们将在测试期间使用我们自己的日志管理器和存储管理器的实例化来调用这个方法。我们保证此方法将在调用交易管理器的任何其他方法之前调用该方法。

作为第一步，我们建议浏览事务管理器，并了解它当前如何实现原子事务。注意，读取应该总是返回特定键的最新提交值，并且在没有崩溃的情况下，写总是原子提交的。您不应该在提交期间尝试验证读集或写集，以提供与我们提供的事务管理器不同的事务隔离级别。如果运行JUnit测试，则所有事务测试都应该成功，但任何与恢复相关的测试都应该失败。

在此分配的第1部分中，您正在向事务管理器提供的事务添加持久性。这意味着已成功提交的事务应该在崩溃后保持提交。您将通过在日志管理器和存储管理器类（我们提供的类）中使用您的事务管理器实现调用方法来实现这一点。

日志管理器类和存储管理器类有不同的持久性保证。只能通过附加日志记录来修改该日志。日志记录是大小有限的字节数组，其格式将被您设计。附加日志记录是原子的，直到附加被持久化。相比之下，存储管理器会异步保存更改，其中对同一键的写可以按顺序持久化，但对不同键的写可以不按顺序持久化。这些类的api有：

- LogManager
  - ○附件日志记录（字节[]记录）
    - ■原子将记录附加并保存到日志的末尾（这意味着之前所有追加都成功）。
    - ■返回附加文件之前的日志长度。
    - ■日志记录的最大长度为128字节。
  - ○ getLogEndOffset ()
    - ■返回最后一个附加后的日志偏移量。
  - ○读取记录（内部位置，大小）
    - ■返回日志的切片[位置，位置+大小)。将数组索引拖出范围异常，如果范围内的任何索引超过了日志的结束或被截断。
  - ○设置日志运行量偏移量(int偏移量)
    - ■持久地将偏移量存储为当前日志截断偏移量，并截断（删除）日志直到该点。您可以忽略此操作，直到此作业的第二部分。
  - ○ getLogTruncationOffset ()
    - ■返回当前的日志截断偏移量。您可以忽略此操作，直到此作业的第二部分。
- StorageManager
  - ○队列写（长键、长标记、字节[]值）
    - ■您应该使用日志偏移作为标记，但在此任务的第2部分之前是不必要的。
    - ■对不同键的写入可能会按照对队列写的调用顺序无序保存，但对相同键的写入将按照对队列写的顺序保存。从概念上讲，您可以将这看作是每个键都有一个单独的队列，从中可以按顺序排出写入，但不保证队列之间的排序。
    - ■每当每次写入都被持久化时，存储管理器都会在事务管理器上调用写入的持久化()回调，并按其持久性排序。您可以忽略此操作，直到此作业的第二部分。
  - ○ Map<Long, TaggedValue> readStoredTable ()
    - ■返回在上次崩溃前持续存在的键到值的映射。返回数据库的第一次初始化的空映射。

您应该考虑如何使用日志管理器和存储管理器的接口来存储写入操作。您需要设计一种序列化格式，以日志记录的形式存储写。我们建议使用缓冲写操作，直到提交，i。e.在提交期间，只将更改写入日志管理器和存储管理器。我们还建议使用重做日志记录，即在日志记录格式中包含新写入的值（与撤消日志记录相反，其中前面的值已写入日志。)因为附加有最大长度，所以您需要考虑如何跨多个记录分割事务的写入（请参阅下面关于工作负载的其他重要细节——例如，您永远不需要跨记录拆分单独的写入()调用）。

请注意，为了使实现正确，必须向存储管理器设置写入队列；仅基于日志进行恢复是不够的。

提示：使用这些类可以将对象序列化为字节数组。

- ByteBuffer
- ByteArrayOutputStream

提示：实现一个自定义的Record类，它可以序列化为字节，并从字节反序列化。(不要使用Java的内置序列化函数来执行这一点。您应该编写自己的序列化函数，将记录打包到字节数组中。)

最后，您将在in恢复()方法中实现恢复。此时，所有的恢复正确性测试都应该成功，但恢复性能测试可能不会成功。

重要细节：

- 数据格式
  - 键很长
  - 日志偏移量是整数
  - 值是字节数组。
- 测试工作负载属性：
  - 一个工作负载中所有事务的写()调用总数最多为100万次。而且，每次测试最多将开始100万个事务。
  - 每个事务最多将涉及1000个要写入()的调用，并且每个写入值的长度最多将为100个字节。
  - 任何时候并发未完成事务的最大数量为10。
  - 在写入相同的键后，我们不会在具有相同txID的事务中读取相同的键。
  - 我们将日志偏移量限制为1Gb，假设您的记录格式不太浪费，那么您在任何工作负载上都不应该达到它。

在测试中第一次调用初始化和恢复时，我们保证日志和数据库将是空的。在崩溃后的后续调用中，我们保证传递给事务管理器的日志将是您的日志，未修改的，并且数据库将在中

一些与该日志相一致的状态。(简而言之，您应该可以随意为日志值使用自定义序列化格式和元数据，您将能够在恢复时进行解析。)

日志管理器和存储管理器的所有方法都可能通过抛出运行时异常而崩溃。这些模拟硬停止碰撞故障，比如停电。您不应该在事务管理器的实现中捕获这些代码，而应该只允许测试代码来处理异常。您可以假设，在任何崩溃之后，将不会有对有失败请求的事务管理器的进一步调用。相反，测试代码将总是构建一个新的事务管理器实例，然后在调用事务管理器的任何方法之前进入恢复。

您也不能将状态存储在可能用于恢复的事务管理器的静态成员中。该分配的精神是，在崩溃后会出现一个新的事务管理器，并只使用日志管理器和存储管理器中保存的数据恢复事务处理。不符合这种精神的解决方案将会受到惩罚。

第2部分（你成绩的30%）：截断日志并快速恢复

在分配的这部分中，您将截断日志以降低恢复成本。当排队的写入被持久化时，存储管理器将调用写入持久化。您将需要实现某种方法来跟踪哪些已提交的写操作还没有被存储管理器持久存储。从这些信息中，您应该能够推导出一个安全的日志截断点，该点不会超过日志中尚未持久保存到存储管理器的任何写入点。然后，你应该定期进行检查。e.在您的写入实现中，调用日志管理器上的设置日志运行点来截断日志。在恢复时，您应该从最后一个日志截断点恢复恢复。您的目标是最小化您需要从日志中读取以实现恢复的次数。

在这一点上，恢复-性能测试也应该会成功。此测试将检查日志是否在操作过程中被截断，并且恢复是否会对日志使用少量的读取I/Os。

（可选的）优化方面的挑战

一旦所有的正确性和恢复测试都通过了，您就可以尝试优化解决方案，以尽量减少事务管理器对每个事务日志的写操作次数。您可以在排行榜测试。java文件中看到基准测试。它提交了几个事务，每个事务都可以进行500个小写操作。要获得这个挑战的排行榜资格，您的实现必须首先通过所有的正确性测试。我们会给特别有效的解决方案给予奖励，但你可以在不做这个挑战的情况下获得完整的分数。

提交说明

见readme.md。