ConcurrentTicketing

一个用于模拟售卖车票的系统，系统主要关注于并发场景下的性能。

主体数据结构

Route Map

这是系统最基础的数据结构，它使用JDK提供的`ConcurrentHashMap`实现，尽管它其实不会发生并发读写的情况，但这里为了严谨还是采用了线程安全的数据结构，同时其`get`方法拥有与`HashMap`相同的性能，对于本程序也可直接替换为`HashMap`。

它主要用于存储车次到`Route`的映射关系，是在系统启动时便生成好的，后续操作时仅会对其进行查询。

Route

使用`CopyOnWriteArrayList<CopyOnWriteArrayList<AtomicInteger>>`这个二维数组来存储`Route`到`Seat`的映射，同时这两个数组也是仅在创建时对其进行初始化，之后仅会进行查询操作，对于本程序也可直接替换为`ArrayList<ArrayList<AtomicInteger>>`。

Seat

使用`AtomicLong`来存储这个座位在不同站点的售出情况，按照站点序号把Long的对应位置置为1，当查询时可通过&操作获取结果，售票和退票时先读取当前值，再通过`compareAndSet`操作来保证不会冲突。

Ticket

使用`AtomicInteger`来生成基础的`TicketSeed`，同时为避免因所有车次共用一个寄存器而造成性能瓶颈，所以为每一个`Route`创建一个`AtomicInteger`来将负载均衡。

真正的`TicketID`则采用64位的结构，其中低24位为获取到的`TicketSeed`，中间的8位为车次，而高32位则由时间、车次等信息拼凑的字符串的HashCode构成。

[^{63} hashCode(passenger + route + coach + seat) ^{32}][^{31} route id ^{24}][^{23} ticket seed ^{0}]

因本系统只解决售票时的并发问题，因此未使用数据库等技术存储已售票信息。因此对于退票信息的验证，仅采用重新生成HashCode的方式来验证。同时对于已退的票会使用`hasReturn`字段来防止错误，错误原因在于系统仅会验证票面信息对应的座位是否已售出，如未售出则报错，当一张票被退后，座位却又被售出，这时再退票就会出现异常，因此使用`hasReturn`字段进行标记。

设计思路

Buy

Lock-free: 首先方法是非阻塞的，当发现没有空座后会返回`Null`。方法使用CAS语句来实现的Lock-free，但为了保证可线性化，未能实现完全的Wait-free，即当有一个线程无数次的被其他线程抢先一步修改Seat的值，其就会一直不能返回结果。

方法实现上采用了遍历所有座位的方法，同时在每个线程开始遍历前，会把车厢和座位数组的顺序都使用`shuffle`方法打乱下，保证在售出票较少的情况下不会所有线程都去竞争同样的位置。

Refund

Lock-free: 首先方法是非阻塞的，当发现没有退票信息错误后会返回`False`。方法使用CAS语句来实现的Lock-free，但为了保证可线性化，未能实现完全的Wait-free，理由与`Buy`方法一致。

方法会首先判断退票信息是否为伪造的，为了不增加题目本意之外的瓶颈，这里采用验证Hash值的方法来实现的，正规系统中建议使用数据库以及行锁来实现。验证票为本系统生成的后，会尝试去退票，并验证该座位是否已售出。

Query

Wait-free: 首先方法是非阻塞的，方法使用了Get语句来实现的Wait-free，同时保证了可线性化。

方法采用遍历的方式实现的，此方法为三个方法中平均执行时间最长的，其性能受座位数量影响很大，建议可仿照12306系统中的实现，当发现余票数量大于某个数值后，就停止查询，能够极大的提升方法性能，不过就与本课程要求不符。

测试数据

正确性测试

正确性测试使用了同学编写的大合集：https://github.com/specialpointcentral/TrainTicketingSystem

其中融合了老师的单线程验证程序，同学编写的多线程验证程序和可线性化验证程序，本项目通过了全部的测试，同时在理论验证阶段也未发现违反可线性化的点。

由于源项目使用了GPL协议，为了避免协议传染，这里不提供整合后的版本，请前往上方地址自行下载，具体测试结果如下图所示：

```
[10/10] Trace has been generated
[INFO] Tests run: 10, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 6.785 s - in ticketingsystem.TraceVerifyTest
[INFO] Running ticketingsystem.UnitTest
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.001 s - in ticketingsystem.UnitTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 74, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  02:24 min (Wall Clock)
[INFO] Finished at: 2020-12-25T23:34:36+08:00
[INFO] ------------------------------------------------------------------------
```

性能测试

> 注：由于服务器使用人数较多，较难得到不受其他用户影响的成绩，因此下面成绩仅为某一次的测试结果。

```
// 测试参数
final static int routenum = 100;
final static int coachnum = 10;
final static int seatnum = 100;
final static int stationnum = 16;
final static int testnum = 1000000; // small test size
final static int testnum = 10000000; // big test size
final static int retpc = 10;
final static int buypc = 30;
final static int inqpc = 100;
```