列车售票系统性能评测报告

姓名：陈学海　学号：2020E8013282057

1. 设计思路

根据系统的性能测试参数可知查询占比达70%，买票与退票分别占据20%，10%。为了使得查询尽可能快，一种思路是维护每个区间的空闲座位，那么查询时索引相应的区间即可快速返回余票。而买票则从区间中取出一张座位，然后在其他重叠区间中删除该座位。退票时则向相应区间添加空闲座位，并向其他重叠区间中也添加该座位。

1.1 数据类设计

- Seat类：描述一个座位的相关属性，包括座位编号、是否正在被使用、当前座位各站点的使用情况。
- IntervalFreeSeat类：记录一个区间[departure, arrival]内的空闲座位信息，是一个空闲座位的集合。
- Train类：记录一辆火车的所有信息，实现了一辆车的查询、购买以及退票的方法。
- TicketingDS类：系统的操作接口，使用底层Train类的方法来实现上层的调用。

A. Seat类内部实现

```
...
private AtomicBoolean inUse; //该座位是否正在被使用的标志，为true时其他线程不能使用该Seat
private int stationUse; //该座位各站点的空闲情况，bit[i]==1表示在站点[i+1,i+2]被占用。
...
```

inUse是座位的锁标志，当我们需要操作一个座位时，首先要锁上该座位，也就是把inUse置true。statonUse记录这个座位在所有站点之间的占用情况，如果一段区间内stationUse的比特位都是0，那么就能购买这种座位。

B. IntervalFreeSeat 类内部实现

```
private ConcurrentHashMap<Integer, Seat> idleSeats; //区间空闲座位的集合，支持并发访问
private int[] counter = new int[2]; //区间内空闲座位数量，有2个counter轮番使用
```

我们使用idleSeats来记录一个[departure, arrival]区间的所有座位，支持并发访问。我们用counter来表示空闲座位的数量。查询时读出该区间的counter即可得到余票数，至于为什么需要两个counter, 以及查询时读哪counter参见Train类的实现。

C. Train类内部实现

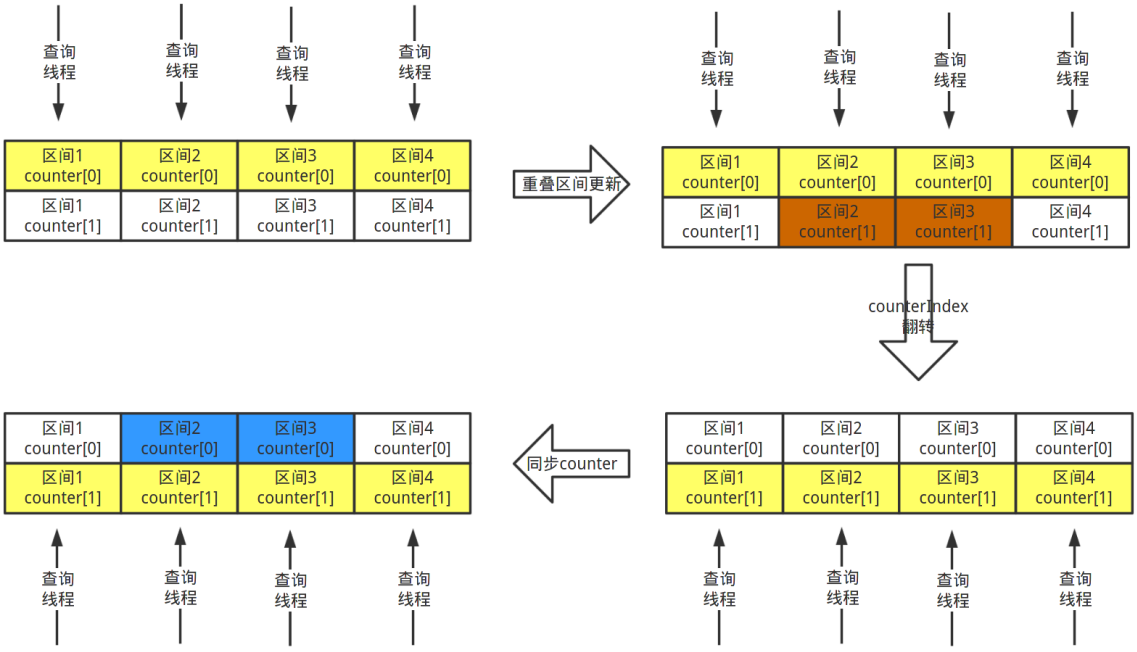
```
private Seat[] routeSeats; //车的所有座位，有coachNum*seatPerCoach个
private static AtomicLong tidAllocator = new AtomicLong(1); //车票id分配器
private IntervalFreeSeat[][] intervalSeats;//所有区间的空闲票信息，两维分别是
departure,arrival
private ConcurrentHashMap<Long, MyTicket> soldTickets; //已售出的车票
private AtomicStampedReference<Integer> counterIndex; //选择哪个counter，取值为0，1
private ReentrantLock trainLock; //火车的锁
```

利用上面的数据成员可以实现可线性化的买票、退票以及查询。

buyTicket(passenger, departure, arrival)

1. 从intervalSeats[departure][arrival]中获取一个空闲的座位freeSeat。
2. 对所有与[departure, arrival]有重叠的区间进行判断，如果该区间在freeSeat.stationUse中的比特串全0，则从这些区间中删除freeSeat。如与[2,4]重叠的有[1,3],[1,4],[1,5],[2,3],[2,4],[2,5],[3,4],[3,5]。
3. 使用trainLock加锁开始更新counter。
4. 对第2步删除座位的区间进行counter更新，做法是将区间的counter[1-counterIndex]减1。
5. 翻转counterIndex, 0变1，1变0，同时更新counterIndex附带的时间戳。
6. 对所有更新的counter进行同步，使得counter[0]==counter[1]。
7. 释放锁，并构造票将其放入soldTickets中返回。

上面的步骤中，需要注意的是对counter的操作，所有修改counter的操作都必须是原子的，我们使用一个火车全局的counterIndex来指示查询线程读取哪个counter，只有当所有交叠区间的counter都更新完毕后counterIndex才会翻转，这样可以保证所有查询线程看到的都是相同的结果。具体可以见下图：



refundTicket(ticket)

1. 检查ticket是否有效，主要判断该票的信息是否与soldTickets中某张票一致。
2. 在soldTickets中删除该退票，并将对应的座位freeSeat置上锁标记。
3. 将freeSeat的stationUse进行更新，具体操作是将[departure, arrival]这段范围的bit全置0。
4. 对所有与[departure, arrival]交叠的区间进行判断，如果freeSeat.stationUse在该区间范围的bit全是0，那么就把freeSeat添加到该区间。
5. 给trainLock上锁，保证counter的更新是原子的。
6. 对第4步所有修改的区间的counter[1-counterIndex]加1。
7. 翻转counterIndex，同时写入新的时间戳。
8. 清除freeSeat的锁标记(inUse置false)。
9. 同步修改的counter, 使得counter[0]==counter[1]。
10. 释放锁。

inquiry(departure, arrival)

1. 读counterIndex的index值以及时间戳t1。
2. intervalSeats[departure][arrival]的counter[index]。
3. 再次读counterIndex的时间戳t2, 如果t1==t2, 则返回读到的counter值，否则继续执行第1步。

2. 正确性与性能分析

2.1 正确性说明

1. 每张车票都有一个唯一的编号tid,不能重复。

   每次买票都会从tidAllocator获取一个递增的数值，因此每张票的id都是唯一且不重复的。

2. 每一个tid的车票只能出售一次。退票后,原车票的tid作废。

   因为tidAllocator永远只会分配新值，所以售出的车票不会再被利用。

3. 每个区段有余票时,系统必须满足该区段的购票请求。

   如果区段有余票，那么该区段的intervalSeats必定有座位，购票请求可以得到满足。

4. 车票不能超卖,系统不能卖无座车票。

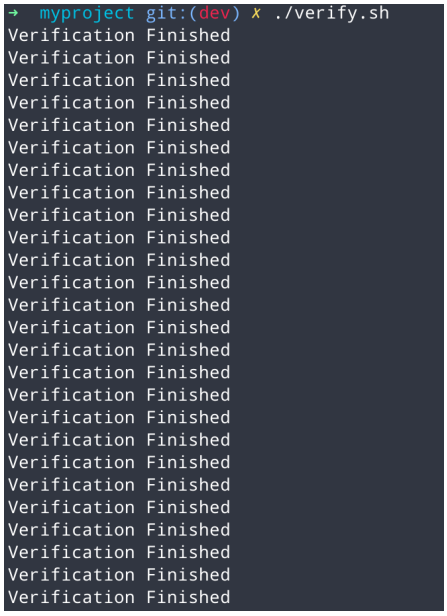
   如果一个区段卖光了，那么intervalSeats中空闲座位idleSeats是空的，会返回null，因此不会卖无座车票。

5. 买票、退票和查询余票方法均需满足可线性化要求。

   所有方法都是可线性化的，参见第3节可线性化分析。

2.2 正确性测试

1. 单线程verify测试，使用老师提供的verify工具进行验证。循环测试50轮，每轮调用10000个方法，全部测试通过。



2. 多线程可线性化测试，使用自己写的多线程可线性化测试工具cheker进行验证。循环测试50轮，每轮5个线程，每个线程执行40次方法调用，全部测试通过。

```
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
Test passed!!!
```

2.3 性能分析

使用Test.java对售票系统进行性能评价，各参数与benchmar.txt一致。每个线程调用100万次方法，得到如下的性能结果：

```
[user083@panda7 UCAS_garbage_ticketingsystem]$ ./perf.sh
ThreadNum: 4 BuyAvgTime(ns): 14509 RefundAvgTime(ns): 12716 InquiryAvgTime(ns): 304 ThroughOut(t/s): 781000
ThreadNum: 8 BuyAvgTime(ns): 9493 RefundAvgTime(ns): 10837 InquiryAvgTime(ns): 341 ThroughOut(t/s): 1941000
ThreadNum: 16 BuyAvgTime(ns): 8530 RefundAvgTime(ns): 10950 InquiryAvgTime(ns): 327 ThroughOut(t/s): 3858000
ThreadNum: 32 BuyAvgTime(ns): 8069 RefundAvgTime(ns): 10844 InquiryAvgTime(ns): 330 ThroughOut(t/s): 7584000
ThreadNum: 64 BuyAvgTime(ns): 8907 RefundAvgTime(ns): 13183 InquiryAvgTime(ns): 344 ThroughOut(t/s): 10083000
ThreadNum: 128 BuyAvgTime(ns): 14198 RefundAvgTime(ns): 22261 InquiryAvgTime(ns): 383 ThroughOut(t/s): 11696000
```

3. 可线性化与进展性分析

3.1 查询

我们使用了几种方法来保证可线性化。首先，在某一时刻，所有查询线程看到的系统状态是一致的，要么全部读的时counter[0]，要么全部读的是counter[1]。当有线程在更新重叠区间的counter时，查询线程读到的依旧都是旧值，而一旦counter全部更新完毕，counterIndex翻转后，查询线程就都读到新值了。其次，为了保证读counterIndex, 以及根据counterIndex读counter之间没有发生翻转，我们给counterIndex附加了时间戳。在读counter之前读一下时间戳，读完之后再读一下时间戳。若两者一致，则没有翻转，否则继续读取。

因此，可以认为查询的可线性化点是最后成功读到counter的那一行：

```
int counter = intervalSeats[departure-1][arrival-1].getCounter(index);
```

根据上面的描述可知inquiry是可线性化、deadlock-free、lock-free的。此外有可能线程每次读取index和counter之间发生index翻转，所以有可能会饥饿。

3.2 购买

买票时首先会从区间idleSeats中取出一个没有被占用的座位(inUse==false)，然后删除交叠区间中该座位，并且原子地更新重叠区间的counter值，可以认为购买的可线性化点在翻转index的那一行：

```
counterIndex.set(1-index, oldStamp[0] + 1);
```

翻转之后，查询线程就可以查到购买之后剩下的余票，退票线程也可以退这张购买的票了。因为锁在用完会被释放，因此是可线性化、deadlock-free的。

3.3 退票

退票时先检查合法性，之后锁住该票对应的座位，把座位添加到与退票区间重叠的一些区间，然后原子地更新counter值，更新完后才释放座位的锁(inUse置为false)。在释放锁前，买线程不会使用这张锁住的座位，因此是可线性化的。可以认为退票的可线性化点在翻转index这一行：

```
counterIndex.set(1-index, oldStamp[0] + 1);
```

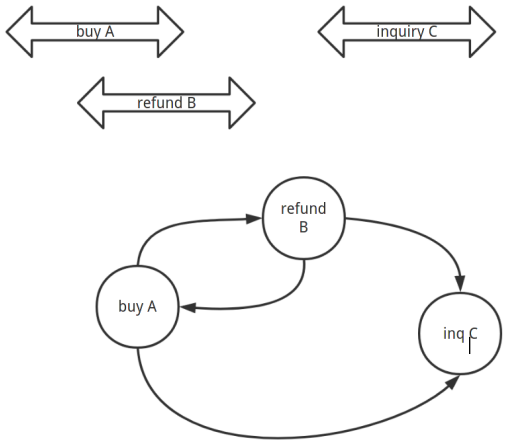
翻转之后，查询线程可以查到退掉的座位，买线程也可以开始购买(循环探测，最坏情况是等到freeSeat的锁标记inUse置false)。所以退票是可线性化、deadlock-free的。

4. 可线性化验证工具checker

4.1 验证思路

判断一段trace是否可线性化，关键是能否找到这些trace的一个全序，使得售票系统按照这个全序执行时每一步的状态都是合法的。一个直观的想法就是按照trace的偏序关系来建图，在图中找到一条路径，使得系统按照该路径执行是合法的。具体建图原则如下：

- 如果trace中某一事件(buy,inquiry,refund) A的结束时间早于B的开始时间，建立一条A到B的边。
- 如果两个事件A,B的执行时间有重叠，那么建立A->B, B->A的双向边。



完成建图后就可以开始DFS查询合法路径了，我们从入度为0的节点开始搜索，双向边不影响入度，只考虑指向本节点的单向边数。上图中A、B、C的入度分别是0，0，2。那么我们需要对A或B开始进行DFS搜索，如果找到一条合法路径，就找到了一个线性化序列，否则这个图就是不可线性化的。

DFS过程中我们维护一个状态机，状态机的状态就是当前售票系统的状态，包括各区间有那些空闲座位、已售出的票有哪些等等。每沿着边到达一个节点时，根据这个节点的动作来更新状态机的状态（可以理解为replay）。如果当前节点的动作无法在状态机中合法实现，如动作是买，但是此时状态机显示没有余票了；或者查询余票数与状态机的余票不一致等等。那么这条路径就是非法的，我们回到上一个节点并还原上一个节点的状态继续找其他边。

除了对状态进行检查，还需要对偏序关系进行检查。我们遍历到某个节点B时，需要对这个节点做如下检查: 如果存在一条A->B的单向边且A不在当前走过的路径上，那么我们就需要终止该路径的探索了，因为合法路径必然先完成A才会完成B，这样可以保证DFS时的每一步都是符合偏序关系的。

4.2 验证过程

- 首先运行Trace.java得到多线程的Trace, Trace.java的参数要求见checker/README
- 运行命令java -jar checker.jar --coach xx --seat xx --station xx < trace即可。
- 如果可线性化的话会打印出线性化序列，当然也可以加参数--no-path-info关闭打印。

```
➜  myproject git:(dev) ✗ java -jar checker.jar --coach 3 --seat 5 --station 5 < trace
:) Linearizable!
[ 2348042, 2386594]Line:   10 Inq leftTick 15 departure 4 arrival 5
[13671090,14975642]Line:    1 Buy ticketID 1 coach 1 seat 1 departure 3 arrial 4
[14217802,14971312]Line:    2 Buy ticketID 2 coach 1 seat 1 departure 4 arrial 5
[13844713,14980950]Line:    3 Buy ticketID 3 coach 1 seat 2 departure 2 arrial 3
[14585163,14973687]Line:    4 Buy ticketID 4 coach 1 seat 3 departure 2 arrial 5
[37909621,37982745]Line:    5 Ref ticketID 1 coach 1 seat 1 departure 3 arrial 4
[37965494,38006002]Line:    6 Ref ticketID 2 coach 1 seat 1 departure 4 arrial 5
[38042249,38092045]Line:    9 Ref ticketID 3 coach 1 seat 2 departure 2 arrial 3
[38155949,38224603]Line:   16 Ref ticketID 4 coach 1 seat 3 departure 2 arrial 5
[38498937,38536092]Line:    7 Buy ticketID 5 coach 1 seat 1 departure 3 arrial 5
[38547476,38553692]Line:   11 Inq leftTick 14 departure 1 arrival 5
[38581279,38602790]Line:    8 Ref ticketID 5 coach 1 seat 1 departure 3 arrial 5
[38641761,38645113]Line:   12 Inq leftTick 15 departure 4 arrival 5
[38832425,38836756]Line:   13 Inq leftTick 15 departure 1 arrival 2
[38944310,38954786]Line:   14 Inq leftTick 15 departure 3 arrival 5
[38962818,38966100]Line:   18 Inq leftTick 15 departure 4 arrival 5
[38992849,38995713]Line:   15 Inq leftTick 15 departure 3 arrival 4
[39056753,39130016]Line:   19 Buy ticketID 6 coach 1 seat 1 departure 1 arrial 5
[39133857,39162702]Line:   17 Buy ticketID 7 coach 1 seat 2 departure 3 arrial 5
[39229539,39232472]Line:   20 Inq leftTick 13 departure 1 arrival 4
[39212917,39241971]Line:   21 Buy ticketID 8 coach 1 seat 3 departure 2 arrial 3
[39216619,39295399]Line:   23 Buy ticketID 9 coach 1 seat 3 departure 3 arrial 5
[39180022,39301126]Line:   25 Buy ticketID 10 coach 1 seat 2 departure 1 arrial 3
[39307691,39309576]Line:   22 Inq leftTick 12 departure 4 arrival 5
[39280593,39333113]Line:   26 Ref ticketID 6 coach 1 seat 1 departure 1 arrial 5
[39364611,39392477]Line:   24 Buy ticketID 12 coach 1 seat 1 departure 3 arrial 4
[39342960,39366427]Line:   27 Buy ticketID 11 coach 1 seat 1 departure 4 arrial 5
[39421321,39428096]Line:   31 Inq leftTick 12 departure 4 arrival 5
```

4.3 验证结果

用脚本循环测试50次，全部通过。

```
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
:) Linearizable!
Test passed!!!
```