第二次博客作业

150616-14051131-陈登博

基于度量分析程序结构

OO_5多线程电梯

- ElevatorSys类
  属性个数:0,方法个数:2,代码行数:52

方法名	行数	分支数
getStartTime	1	0
main	38	3

- Elv类
  属性个数:18,方法个数:7,代码行数:203

方法名	行数	分支数
Elv	13	0
getWork	1	0
run	100	30
fetch	32	11
isOn	3	0
ifResponsive	8	5
ifCarriable	7	4

- Floor类
  属性个数:4,方法个数:6,代码行数:28

方法名	行数	分支数
Floor	4	0
turnOn	4	2
turnOff	4	2
isOn	4	2
getMaxFloor	3	0
getMinFloor	3	0

- InputHandler类
  属性个数:0,方法个数:1,代码行数:33

方法名	行数	分支数
parseString	28	7

- MyUncaughtExceptionHandler类
  属性个数:0,方法个数:1,代码行数:5

方法名	行数	分支数
uncaughtException	3	0

- OldSchedule类
  属性个数:3,方法个数:2,代码行数:25

方法名	行数	分支数
FoolSchedule	6	1
ALSSchedule	14	2

- Req类
  属性个数:6,方法个数:7,代码行数:35

方法名	行数	分支数
Req	13	2
getDstFloor	1	0
getElvId	1	0
getDirection	1	0
getType	1	0
toString	7	1
getTime	1	0

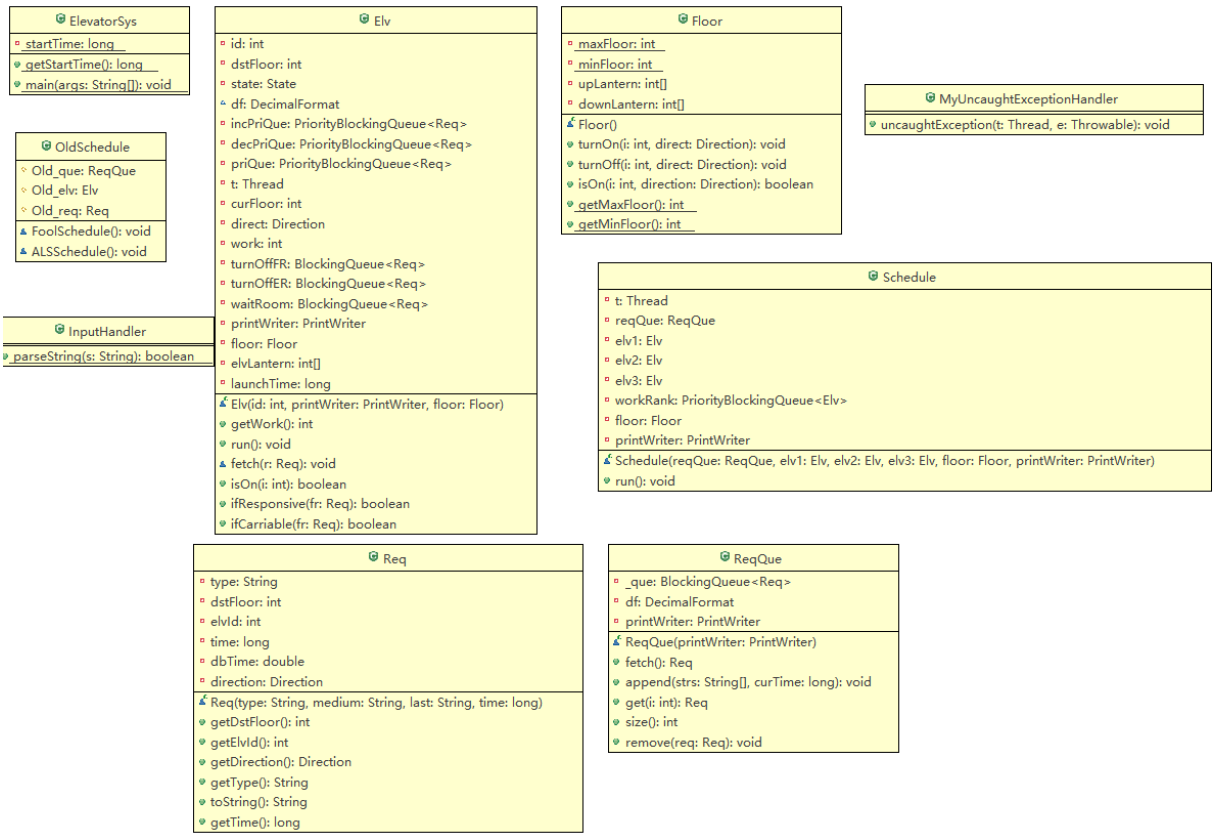
- ReqQue类
  属性个数:3,方法个数:6,代码行数:46

方法名	行数	分支数
ReqQue	3	0
fetch	4	2
append	10	2
get	10	2
size	3	0
remove	3	0

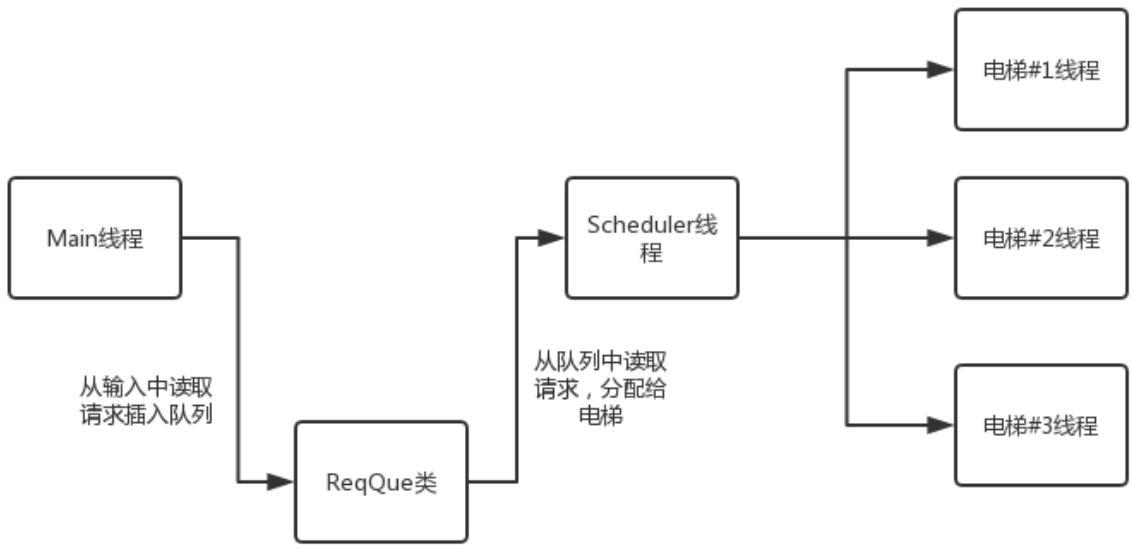
- Schedule类
  属性个数:8,方法个数:2,代码行数:90

方法名	行数	分支数
Schedule	10	0
run	59	25

- 类图



- 线程协作关系



- 优缺点
  - 优点：很大程度上继承了之前设计电梯的思想，设计难度变小，易于实现。
  - 缺点：在多线程处理上有些欠缺，Elv类过于冗长，不符合单一职责原则。

OO_6IFTTT

- Detail类
  属性个数:2,方法个数:4,代码行数:66

方法名	行数	分支数
Detail	8	0
register	37	9
print	6	0
close	3	0

- FileManager类
  属性个数:1,方法个数:1,代码行数:15

方法名	行数	分支数
getNewFile	10	2

- FileMap类
  属性个数:2,方法个数:3,代码行数:14

方法名	行数	分支数
FileMap	4	0
getKey	3	0
getValue	3	0

- FileSystemMonitor类
  属性个数:15,方法个数:8,代码行数:290

方法名	行数	分支数
FileSystemMonitor	16	2
FileSystemMonitor	15	2
addTask	8	1
addTask	6	0
hasSummary	4	0
start	5	1
getPaht	3	0
run	212	70

- IFTTT类
  属性个数:0,方法个数:1,代码行数:71

方法名	行数	分支数
main	65	9

- InputHandler类
  属性个数:1,方法个数:5,代码行数:73

方法名	行数	分支数
parseInput	32	10
parseDir	9	1
parseTrigger	11	2
isDirectory	4	0
parseTask	7	0

- MyExceptionHandler类
  属性个数:0,方法个数:1,代码行数:5

方法名	行数	分支数
uncaughtException	3	0

- ObjFile类
  属性个数:8,方法个数:31,代码行数:309

方法名	行数	分支数
ObjFile	15	3
update	15	3
ifwasFile	3	1
getLastObjFile	3	0
getLastLength	3	0
getName	4	2
getAbsolutePath	7	0
isFIle	7	0
isDirectory	7	0
getLength	15	4
getModifiedTime	7	2
getLastModifiedTime	3	2
getWorkingDirectory	3	0
getParent	3	0
setRenamed	3	0
getRenamed	3	0
setPathchanged	3	0
getPathchanged	3	0
toString	3	0
exists	7	0
listFiles	12	0
createNewFile	12	4
createNewFolder	12	4
deleteFile	17	4
suicide	7	0
renameTo	22	3
rewrite	13	2
getDeleteFiles	9	2

方法名	行数	分支数
getNewFiles	25	6
getModified	13	8
getSizeChanged	11	3

- Recover类
  属性个数:0,方法个数:2,代码行数:54

方法名	行数	分支数
pathChanged	27	6
copy	18	2

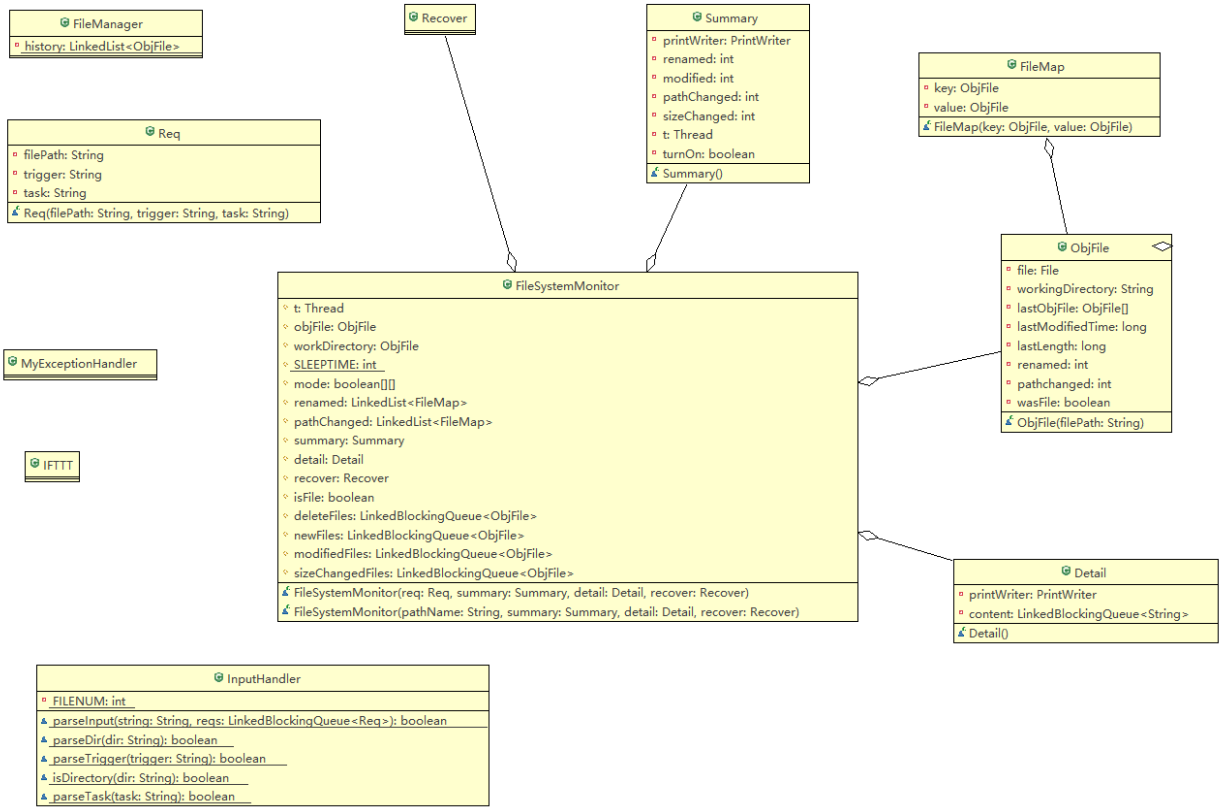
- Req类
  属性个数:3,方法个数:6,代码行数:32

方法名	行数	分支数
Req	5	0
getFilePath	3	0
getTrigger	3	0
getTask	3	0
isSame	5	0
toString	3	0

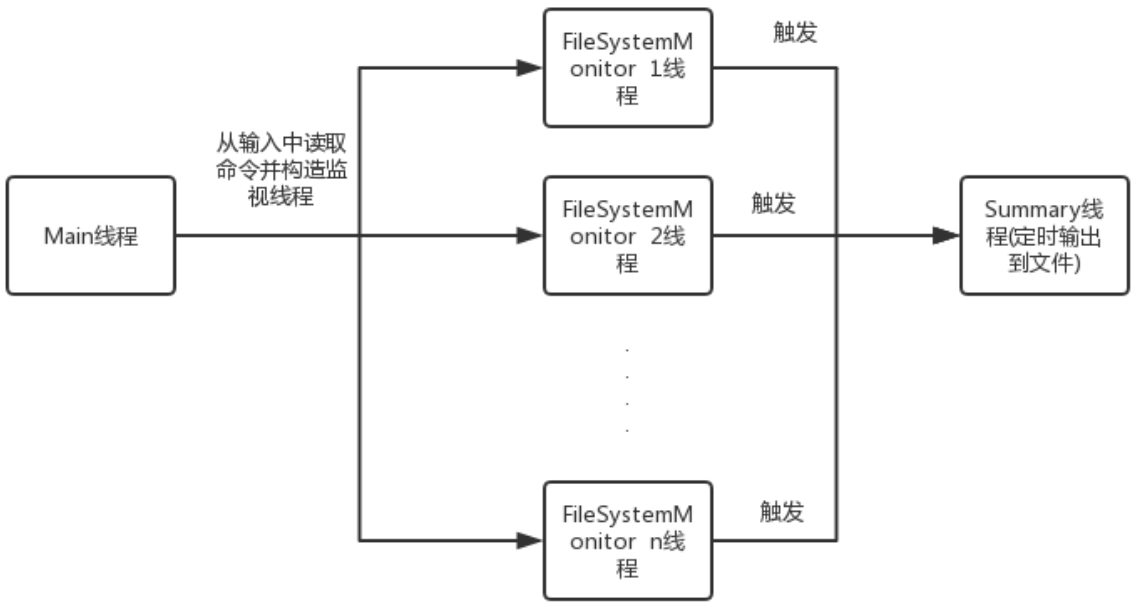
- Summary类
  属性个数:7,方法个数:5,代码行数:60

方法名	行数	分支数
Summary	10	0
register	15	6
run	12	2
close	4	0
start	3	0

- 类图



- 线程协作关系



- 优缺点
  - 优点：采用了一个工作区一个线程的思路，简化了文件监控的操作
  - 缺点：在监控器线程的编写中没有做到高内聚低耦合，导致FileSystemMonitor类过于庞大

OO_7出租车调度系统模拟

- Customer类
  属性个数:8,方法个数:11,代码行数:127

方法名	行数	分支数
Customer	13	0
toString	3	0
getStart	3	0
getTime	3	0
getEnd	3	0
equals	5	0
addTaxi	3	0
tryToGetOn	52	13
isLogged	3	0
setLogged	5	1
log	16	0

- InputRequest类
  属性个数:3,方法个数:2,代码行数:72

方法数	行数	分支数
InputRequest	5	0
parseInput	57	8

- Map类
  属性个数:3,方法个数:3,代码行数:66

方法数	行数	分支数
Map	8	1
getEdges	3	0
init	46	7

- MyExceptionHandler类
  属性个数:0,方法个数:1,代码行数:5

方法数	行数	分支数
MyExceptionHandler	3	0

- RequestQueue类
  属性个数:1,方法个数:6,代码行数:39

方法数	行数	分支数
RequestQueue	3	0
poll	3	0
peek	3	0
isEmpty	3	0
offer	3	0
traverse	18	3

- Scheduler类
  属性个数:3,方法个数:3,代码行数:26

方法数	行数	分支数
Scheduler	6	0
run	12	2
start	3	0

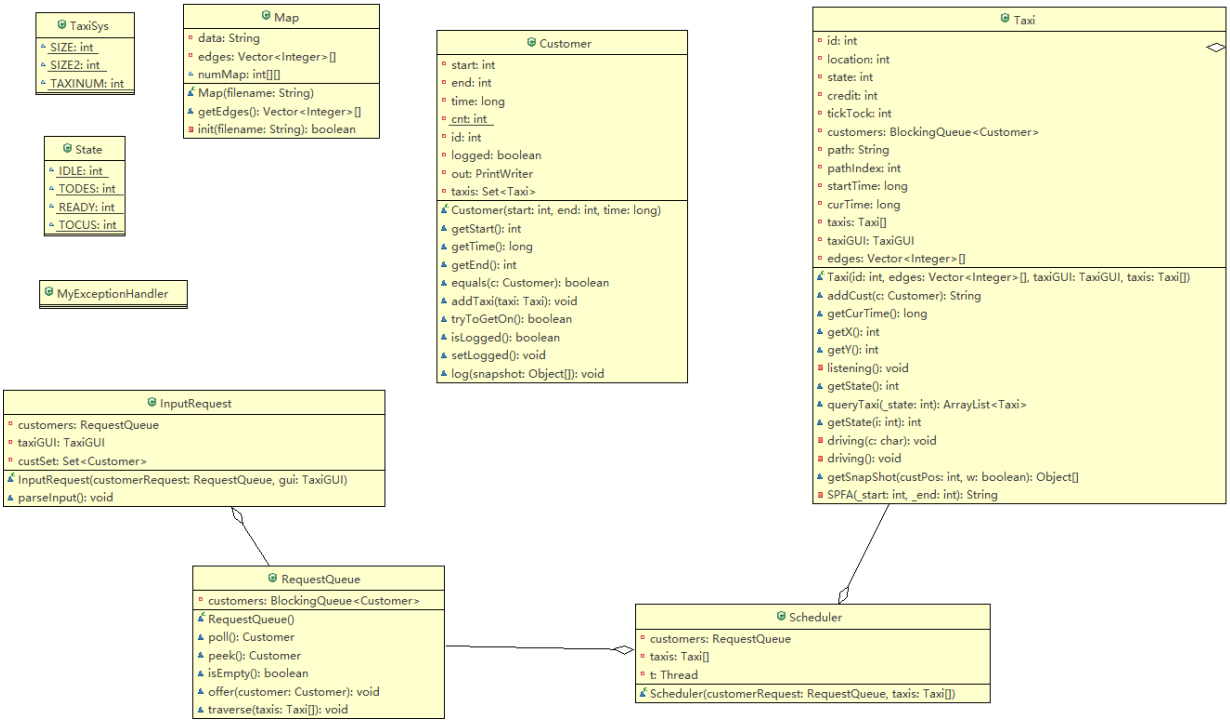
- Taxi类
  属性个数:13,方法个数:15,代码行数:226

方法数	行数	分支数
Taxi	14	0
toString	1	0
addCust	10	1
getCurTime	3	0
getX	3	0
getY	3	0
listening	7	1
getState()	3	0
queryTaxi	8	1
getState(int i)	3	0
driving(char c)	8	4
driving()	9	0
run	75	20
getSnapShot	9	1
SPFA	39	7

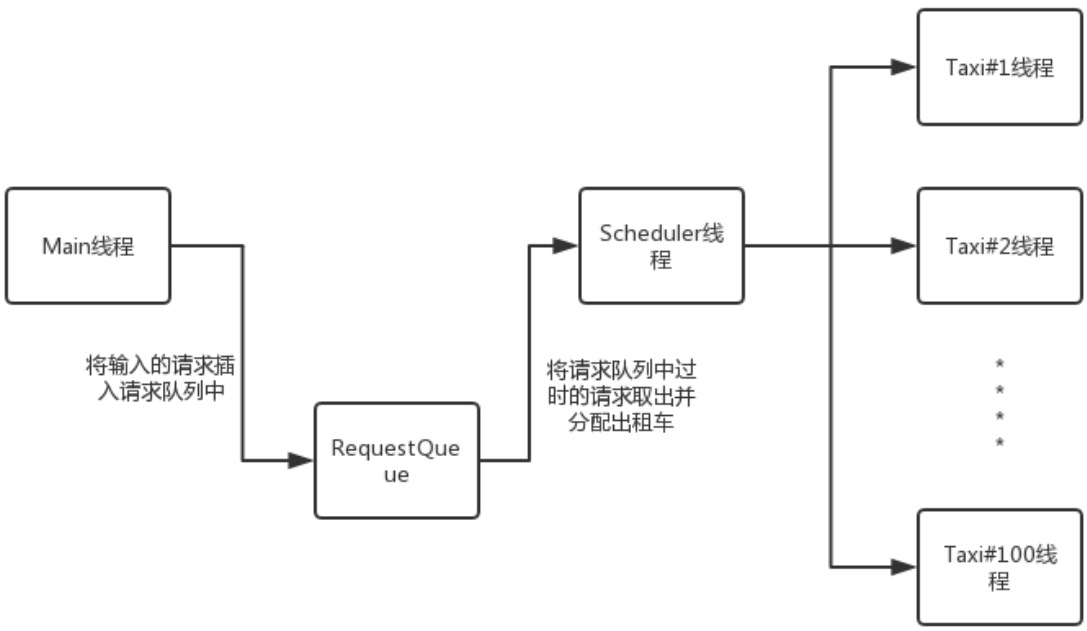
- TaxiSys类
  属性个数:3,方法个数:1,代码行数:20

方法数	行数	分支数
main	15	0

- 类图



- 线程协作关系



- 优缺点
  - 优点：比起之前几次作业，本次作业各个类的大小稍微均衡了一点。
  - 缺点：在程序性能问题上仍有提升空间，在开启多个请求的时候程序容易卡慢。

Bug分析

- OO_5

```
输入
(FR,2,UP);(FR,3,UP);(FR,4,UP);(FR,5,UP);(ER,#2,3)
输出
1491827006232:[FR, 2, UP, 0.0] / (#1, 2, UP, 1, 3.0)
1491827009232:[FR, 3, UP, 0.0] / (#2, 3, UP, 2, 6.0)
1491827012232:[FR, 4, UP, 0.0] / (#3, 4, UP, 3, 9.0)
1491827021232:[FR, 5, UP, 0.0] / (#1, 5, UP, 4, 18.0)
1491827021232:[ER, #2, 3, 0.0] / (#2, 3, STILL, 2, 18.0)
```

实际上不能称之为BUG，主要原因是在理解指导书上的偏差。我采取的思想是如果当前请求如果不能被任何电梯执行，那么这条请求就被阻塞，而该请求之后的请求也理所当然地被阻塞。但是经过咨询助教才发现如果后面有可以执行的请求，那么后面的请求反而会先执行。因此导致了该扣分。修改方式只需要改变指令分配策略，将后面可以继续执行的指令继续分配给电梯即可。

- OO_6

  在处理单个文件的修改触发时，错误地将工作区更新了两遍，一遍在文件记录之前，一遍再文件记录之后。这样在文件记录之前的更新抹去了所有的修改信息，导致无法输出单个文件修改信息。解决方案是删除在文件记录之前的这次工作区更新，只保留之后的。

- OO_7

  在处理重复请求时，用到了`Set`类的方法，但是没有考虑到`Set`类的`contains`方法所操作的对象是我`new`出来的，这些对象的地址自然不同，因此对于重复请求的判断失效。解决方法可以是每次添加新的用户到请求队列中时从头到尾扫描一遍队列，检测其中是否有重复请求，有则不添加，没有则添加并执行。

探测BUG所用策略

从边界值考虑，构造测试样例。例如OO6对方代码：

```
public static void rename(String path, String newname) {
    try {
        rwl.writeLock().lock();
        File fi = new File(path);
        if (fi.isDirectory())
            throw new Exception();
        String newpath = fi.getParent() + newname;
        if (!fi.renameTo(new File(newpath)))
            throw new Exception();
    } catch (Exception e) {
        System.out.printf("Rename %s to %s failed\n", path, newname);
    }
    finally {
        rwl.writeLock().unlock();
    }
}
```

这是对方的重命名代码，可以看出在产生newpath的过程中对方仅仅将父目录与新文件名拼接，缺少文件分隔符，经过测试也发现，对于D:/test/x.txt进行重命名为y.txt的操作，结果产生的是生成了D:/testy.txt，这显然不是重命名了。

心得体会

这三次作业让我感觉到一个好的设计的关键，在拿到一个作业之前首先不应该急着去编码，而是将整个程序的框架设计好，这样会大大减少后续的调试时间，而且不会陷入复杂的逻辑怪圈中。

这三次作业强调的最多的就是多线程之间的协同了，多线程相比于单线程有着更多难以捉摸的行为，至少从调试难度上就比单线程大了许多，因此多线程更多的是用输出进行调试。但是输出调试法也不是万能的，貌似正确的程序也可能存在潜在的死锁或者崩溃问题，因此需要正确的设计线程之间的同步关系。用`synchronized`关键字可以帮助我们控制多个线程之间的协同，但是在后来的作业尤其是出租车调度系统作业中，我也感受到在不合适位置放置的`synchronized`会给整个程序带来多大的性能损耗，这一点在以后的作业中需要特别注意。

最后，经过这几次作业的洗礼，我感受到如果在提交作业之前先进行一遍人工全局查错，那么整个程序的BUG就会少很多。因为在写完程序之后，整个程序的框架已经了然于胸了，人工查错能找到那些由我们自己构造的测试样例所无法测出的BUG，好几次我都是在人工查错的过程中发现了我的代码存在的重大BUG，因此避免了在互测过程中被扣分。