第二单元总结分析

OO2018课程组

北航计算机学院

内容提纲

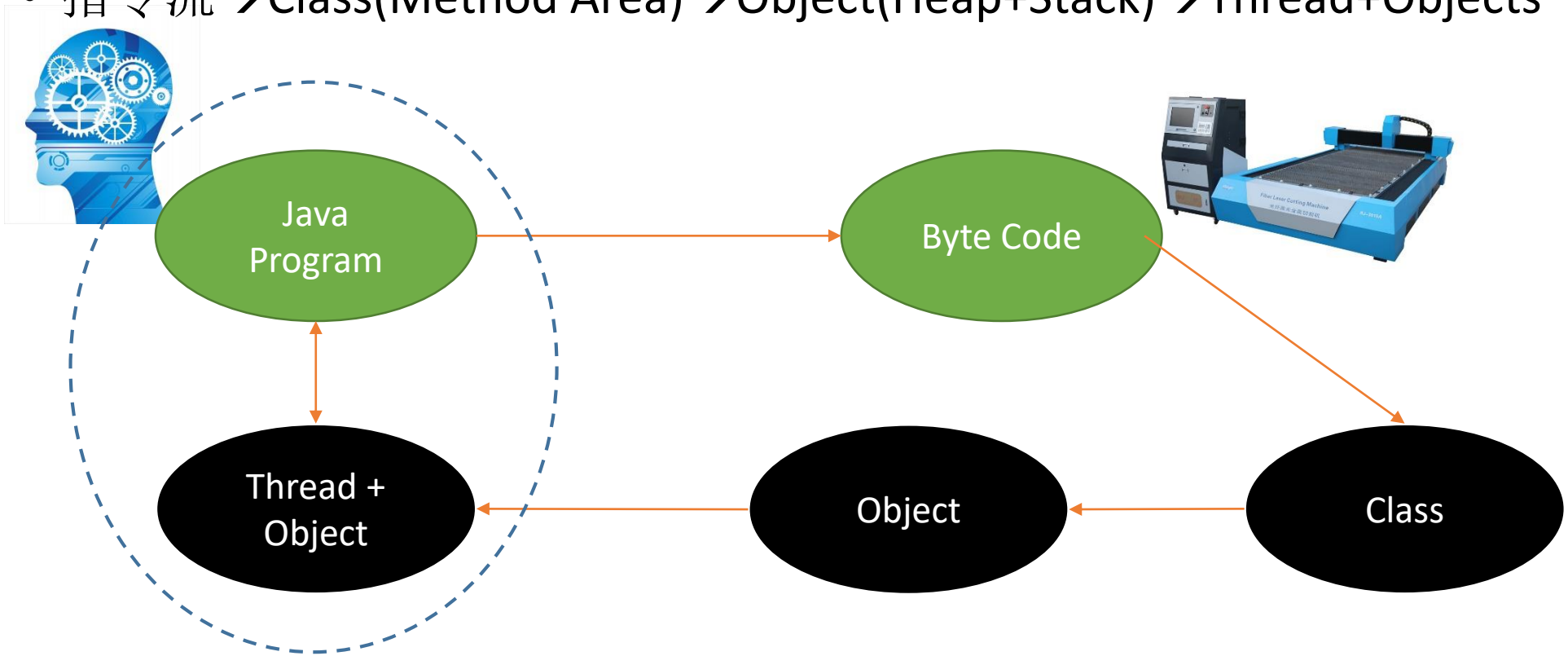
- 单元知识点回顾
- 作业训练要点分析
- 本次作业

单元知识点回顾

- 对象不只是一个逻辑概念
  - 也是运行时概念
- 静态时对象
  - 类定义其规格：方法、属性
  - 对象通过相应类型的变量来引用和访问，对象的存储和管理属于运行时概念，被屏蔽
  - 对象一旦创建，类型不会变化；对象引用可以进行类型转换
- 运行时对象
  - 存储
  - 访问管理

单元知识点回顾

- 介绍JVM的基本机理是为了学习和体会对象的运行时特性
  - Java程序→class文件： byte[]
  - 指令流→Class(Method Area)→Object(Heap+Stack)→Thread+Objects



单元知识点回顾

- 线程是在静态层面刻画对象运行时特性的机制
  - Thread.start是个异步方法
  - Thread.run不让我调用(?)
  - Thread似乎失控了：你无法在代码逻辑中确定其状态，并对其进行确定性的控制
- Thread的双重职责
  - 根据功能管理所需对象及其访问
  - 根据系统设计要求，与其他线程交互（等待/唤醒/...）

单元知识点回顾

- 线程在执行控制上具有独立性
  - 不具有互相调用关系
  - Q：如果我在一个线程方法中调用了另一个线程对象的方法会发生什么？
- 线程之间的关系
  - 父子关系：创建和启动线程
  - 协作关系：生产者-消费者
  - 同步关系：通过共享对象，或者主动状态控制进行同步(wait,notify)
- 访问共享对象是个充满变数的事情
  - 稍有不慎，程序状态失控
  - 减少共享
  - 不得不共享时，同步控制

单元知识点回顾

- 生产者---消费者
  - 1个生成者---1个消费者
  - 1个生产者---多个消费者
  - 多个生产者---1个消费者
  - 多个生产者---多个消费者
  - 单一产品的生产与消费
  - 多种产品的生产与消费
- 一种多线程协同的设计模式
  - Producer
  - Consumer
  - Queue<something>
  - Controller

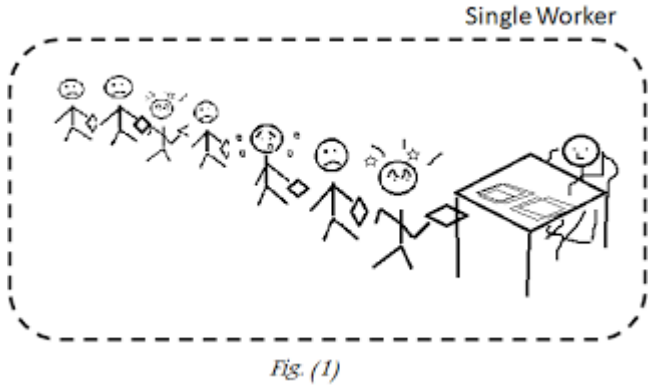


Fig. (1)

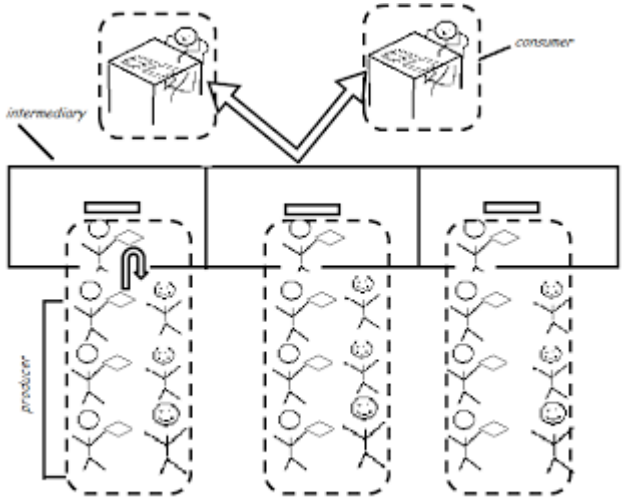


Fig. (3)

http://www.javacodegeeks.com/2012/02/concurrency-pattern-producer-and.html

单元知识点回顾

- 对象锁也是一个神奇的东西
  - 每个对象有且只有一把锁
  - 对象既可以是“房间”：使用其自身的锁来控制对它的访问
  - 对象也可以是一把“锁”：使用它来控制互斥进入其他“房间”
- 如何使用锁并不是一个琐碎问题
  - 线程代码加锁
  - 中间过程on-demand加锁
  - 共享对象加锁
- 我们推荐：共享对象加锁
  - 线程安全类

单元知识点回顾

- 线程安全
  - “线程**，你不用担心什么，只管做你的事情，保证不给你添乱...”
- 如何导致线程不安全
  - 读写冲突
  - Check-then-act
  - Read-modify-write
- 线程安全类的特征
  - 自己管理自己的访问控制（任意时候都假设会有多个线程共享访问）
  - 每个“房间”都配备了相应的“门禁”

单元知识点回顾

- 我什么时候知道该用多线程？
  - 面向对象分析/设计时
- 线程类与线程实例
  - 几类线程
  - 几个线程
- 继承、接口、调用形成了面向问题分解和归纳的设计结构
  - 关注静态规格
- 线程、协同、同步形成了面向性能和资源控制的设计结构
  - 关注动态行为

作业训练要点分析

- 作业5训练要点分析
- 作业6训练要点分析
- 作业7训练要点分析

作业5训练要点分析

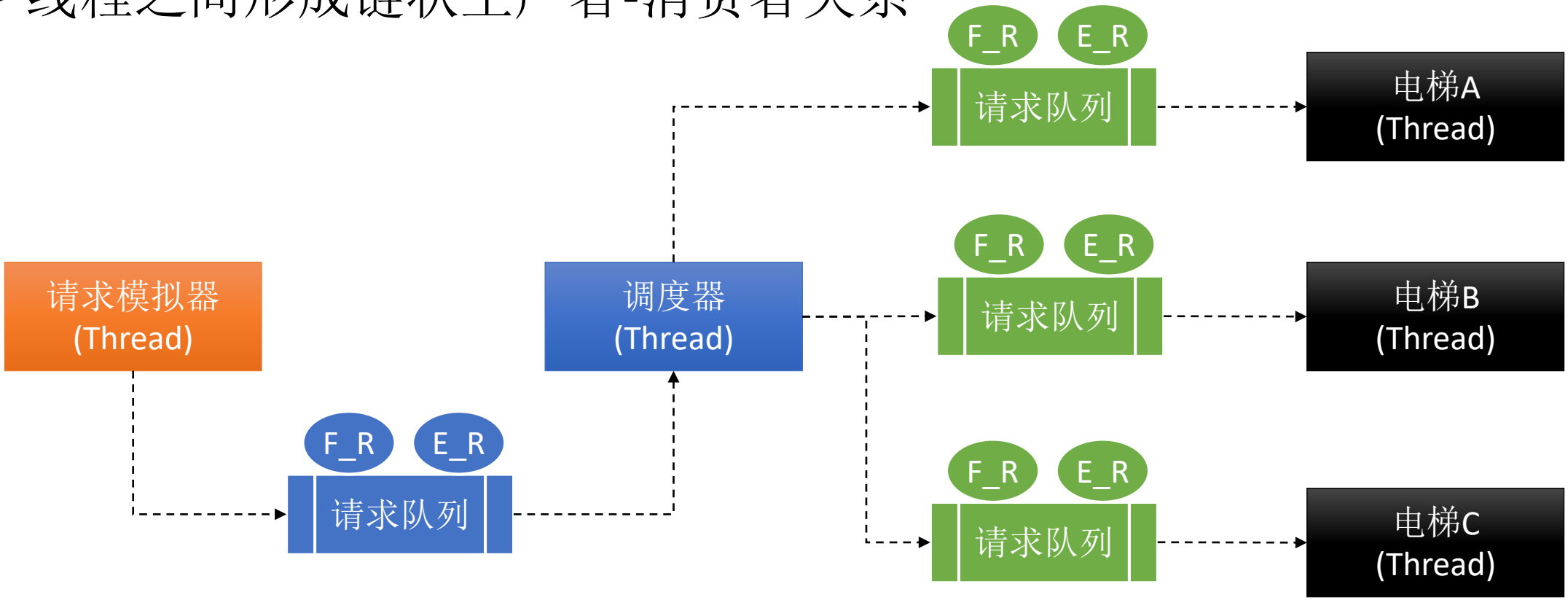
- 理解和实践线程交互
- 线程交互模式识别

作业5训练要点分析

- 理解和实践线程交互
  - 电梯、乘客在问题域中就具有并发行为
  - 电梯之间“竞争”响应请求
  - 乘客之间“竞争”使用电梯
- 本次作业的难点是如何构造电梯之间、电梯与请求之间的协作关系

作业5训练要点分析

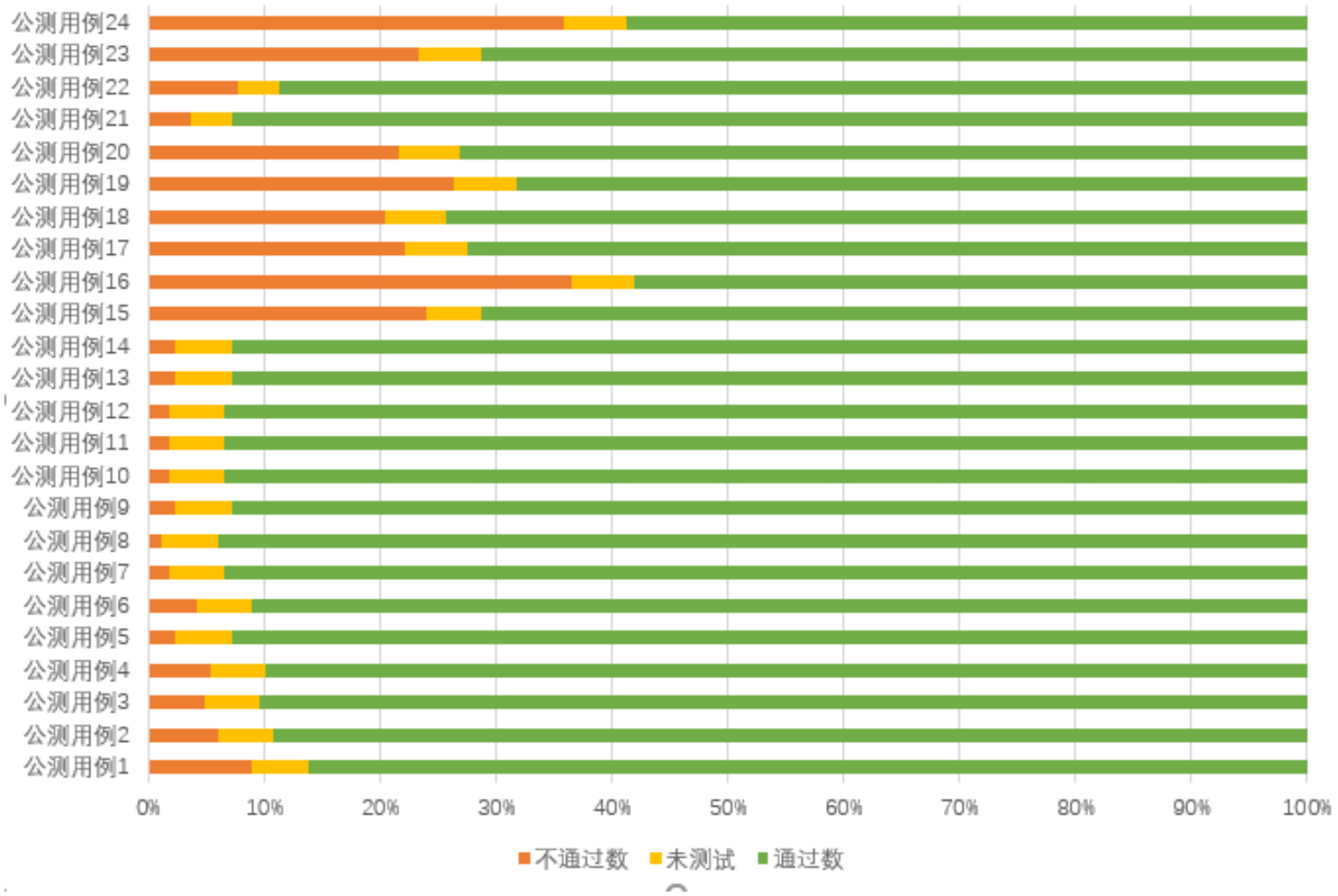
- 线程之间形成链状生产者-消费者关系

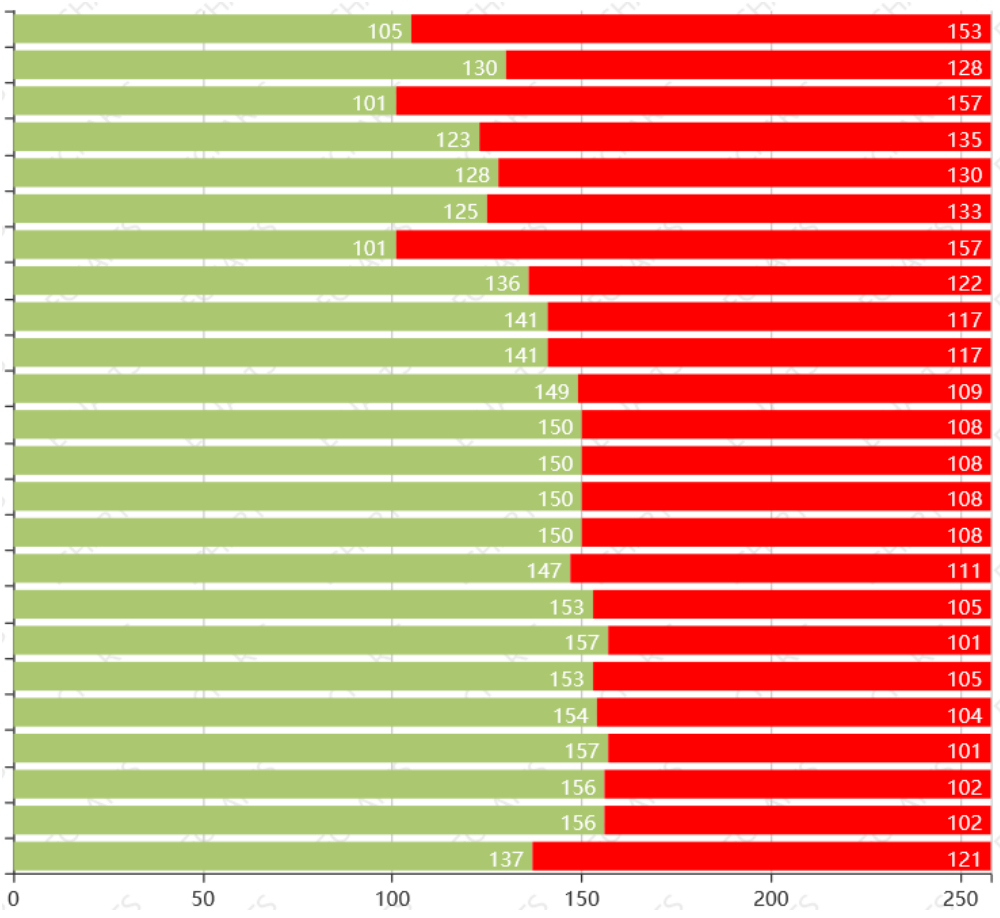


作业5训练要点分析

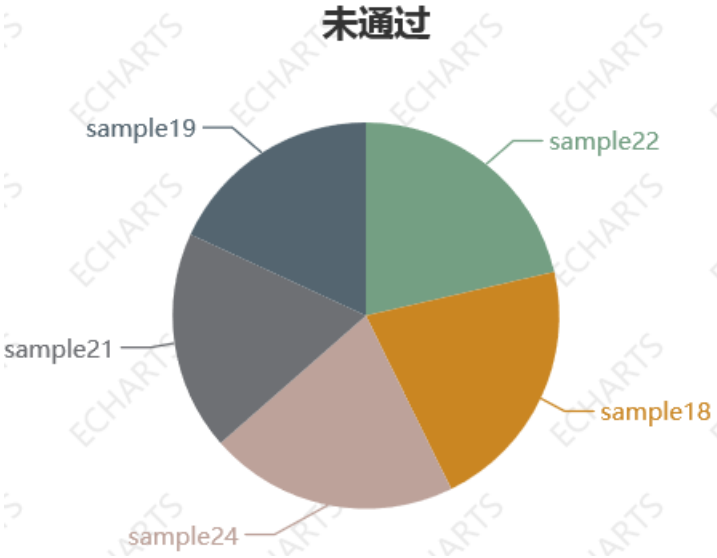
- 如何支持捎带？
  - 之前的设计：调度器持续监控电梯的运动状态变化，从而扫描队列找到相应的请求进行捎带调度
  - 多线程设计：调度器(线程)如何知道电梯(线程)的状态？
    - 方案A：调度器拥有对所有电梯线程对象的引用，把电梯线程当成一般对象进行访问
    - 方案B：设置一个status board，电梯线程把自己的状态发布到board，调度器从中读取电梯状态
  - 捎带调度的动态性要求
    - 电梯运行过程中不断出现新的请求，无法提前做好预判
    - 实际工程考虑：中心调度器关注请求到电梯的分配（强调运动量或负载均衡）；电梯调度器关注捎带处理（强调请求响应效率）。

作业5公测整体情况对比

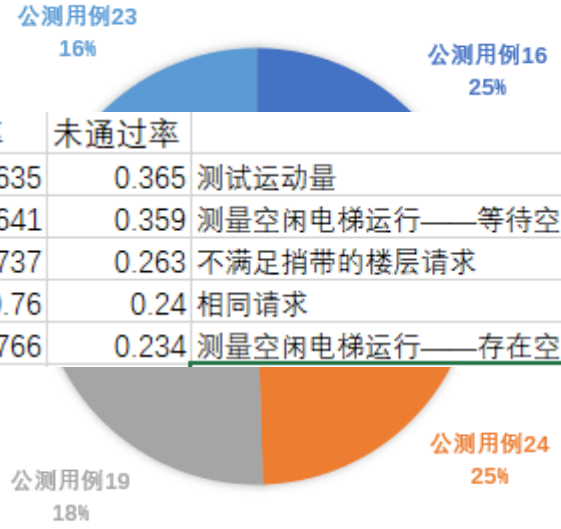




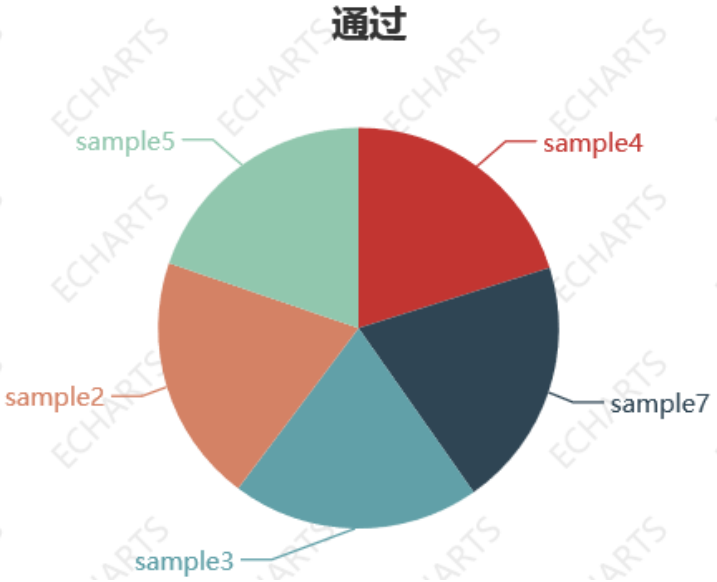
作业5公测疼点与甜点对比



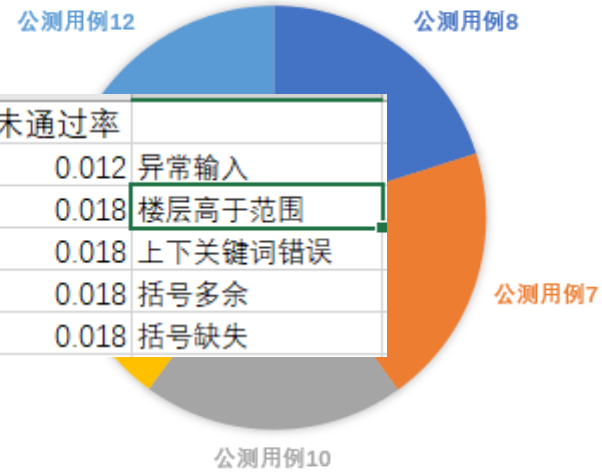
Top5 failed in 2017:
涉及多个电梯的捎带功能，包括多个运动方向、所处楼层、请求间隔时间等。



名称	通过率	未通过率	
公测用例16	0.635	0.365	测试运动量
公测用例24	0.641	0.359	测量空闲电梯运行——等待空闲电梯
公测用例19	0.737	0.263	不满足捎带的楼层请求
公测用例15	0.76	0.24	相同请求
公测用例23	0.766	0.234	测量空闲电梯运行——存在空闲电梯



Top5 passed in 2017:
主要涉及输入有效性检查、重复请求等基本功能测试



名称	通过率	未通过率	
公测用例8	0.988	0.012	异常输入
公测用例7	0.982	0.018	楼层高于范围
公测用例10	0.982	0.018	上下关键词错误
公测用例11	0.982	0.018	括号多余
公测用例12	0.982	0.018	括号缺失

作业5的主要设计问题

- 没有或错误的线程安全控制
  - 对于“托盘”类没有加锁，导致多个线程访问同一个“托盘”时发生不安全问题。或者对加锁理解错误导致线程安全不稳定，常表现为相同的输入，前后两次输出不同。
- 在线程类代码中直接进行访问控制
- 在线程类的run方法中展开细节
- 没有交互的继承设计（Scheduler类）
  - 子类和父类之间没有交互
- 面条代码
  - 一个类差不多就一个方法，使用大量的分支控制，长度甚至达500行

作业6训练要点分析

- 程序并发特征由待处理输入的特征决定
  - 多个平级目录需要进行处理
  - 注意理解与多电梯系统的并发特征差异
- 实践线程的主从协同模式
- 基于锁的线程同步设计
- 实践线程安全设计
- 使用代码进行测试

作业6训练要点分析

- 实践线程的主从协同模式
  - 不同线程的角色不同，工作模式也不同
  - 主动工作模式
    - 主动扫描或检查外部世界是否发生变化
    - 例：端口扫描
    - 例：扫描文件信息的变化、扫描网页的变化
  - 被动工作模式
    - 不断查看某个信号量被设置（内部设计）
    - 例：tray是否available
    - 例：判断是否有新的数据到达

作业6训练要点分析

- 实践线程的主从协同模式
  - 采用主动模式的线程需要与外部环境对象进行交互
  - 采用被动模式的线程需要按照一定协议与主动模式线程交互
  - 需要几个主动工作模式线程？
  - 需要几个被动工作模式线程？
  - 主动线程与被动线程如何协同？

作业6训练要点分析

- 实践线程的主从协同模式
  - 需要几类线程？
    - 系统需要“感知”多少类不同的外部对象状态变化？
      - 触发器包括renamed, modified(time), path-changed, size-changed
    - 如何知道是否发生变化？
      - 建立快照snapshot（多叉树）
      - 基于快照的diff
      - 快照的更新
    - 获取快照线程(主动工作，周期如何确定)，不关心是否发生变化
    - 快照diff线程(被动工作，一旦有’新’快照)，发现变化，不关心I/O操作
    - 触发控制线程(被动工作，一旦发现变化)，该触发哪个/哪些触发器(执行哪个任务)？
  - 几类线程之间如何协作？
    - 生产者-消费者
    - Tray: 快照、变化

作业6训练要点分析

- 实践线程的主从协同模式
  - 需要创建多少个线程？
    - 方案A：只要遇到目录，就创建一个线程对象
    - 方案B：以顶目录为根，按照目录层次(如3层)限制来创建线程对象
    - 方案C：了解输入的顶层目录深度，综合决定需要创建多少个线程对象
  - 有哪些共享信息？
    - 快照、文件属性变化、summary、detail
  - 在指定文件中记录信息本质上是同步行为
    - 方案A：触发控制线程独立输出到指定文件
    - 方案B：触发控制线程独立输出到一个内存对象(buffer)，然后集中输出到外存文件

作业6训练要点分析

- 基于锁的线程同步设计
  - 每个对象都内置了一个lock
  - 任何一个对象都可以用来控制线程进入**受控区域**的执行
    - 又称为临界区
  - 为什么锁会发生效果？
  - 锁加在何处？
  - 使用哪个锁？

作业6训练要点分析

- 基于锁的线程同步设计
  - 为什么锁会发生效果？
    - (1)多个线程会执行到相同的代码区域
    - (2)多个线程使用相同对象的锁(即相同的锁)
    - (3)JVM确保在任何时候只能有一个线程获得进入受控区域的锁
    - (4)线程可以主动放弃锁
    - (5)或者执行结束自动释放锁
  - 锁加在何处？
    - 锁住线程执行体
    - 锁住共享对象
  - 使用哪个锁？
    - 与受控区域计算相关的锁
    - 与受控区域计算无关的锁

作业6训练要点分析

- 基于锁的线程同步设计
  - 锁加在何处
    - 锁住线程执行体
    - 锁住共享对象
  - 哪一个会取得预期效果？

线程同步--->线程在<共享对象>的访问上同步
线程在<共享对象>的锁上同步

```
//Thread body
run(){
    …
    synchronized(e){
        …
        try{…wait()…}catch (InterrupedException ex){}
    }
}
```

```
//Shared Object body
method(){
    …
    synchronized(e){
        …
        try{…wait()…}catch (InterrupedException ex){}
    }
}
```

作业6训练要点分析

- 基于锁的线程同步设计
  - 使用哪个锁？
    - 受控区域计算相关的锁
    - 受控区域计算无关的锁

在完成了线程设计、线程同步设计之后，锁的选择也是关键问题。

每个受控区域都需要一把锁：与受控区域计算任务相关的共享对象是合适的锁

```
//Shared Object body
method(){
    …
    synchronized(e){
        …
        e.func();
        …
    }
}
```

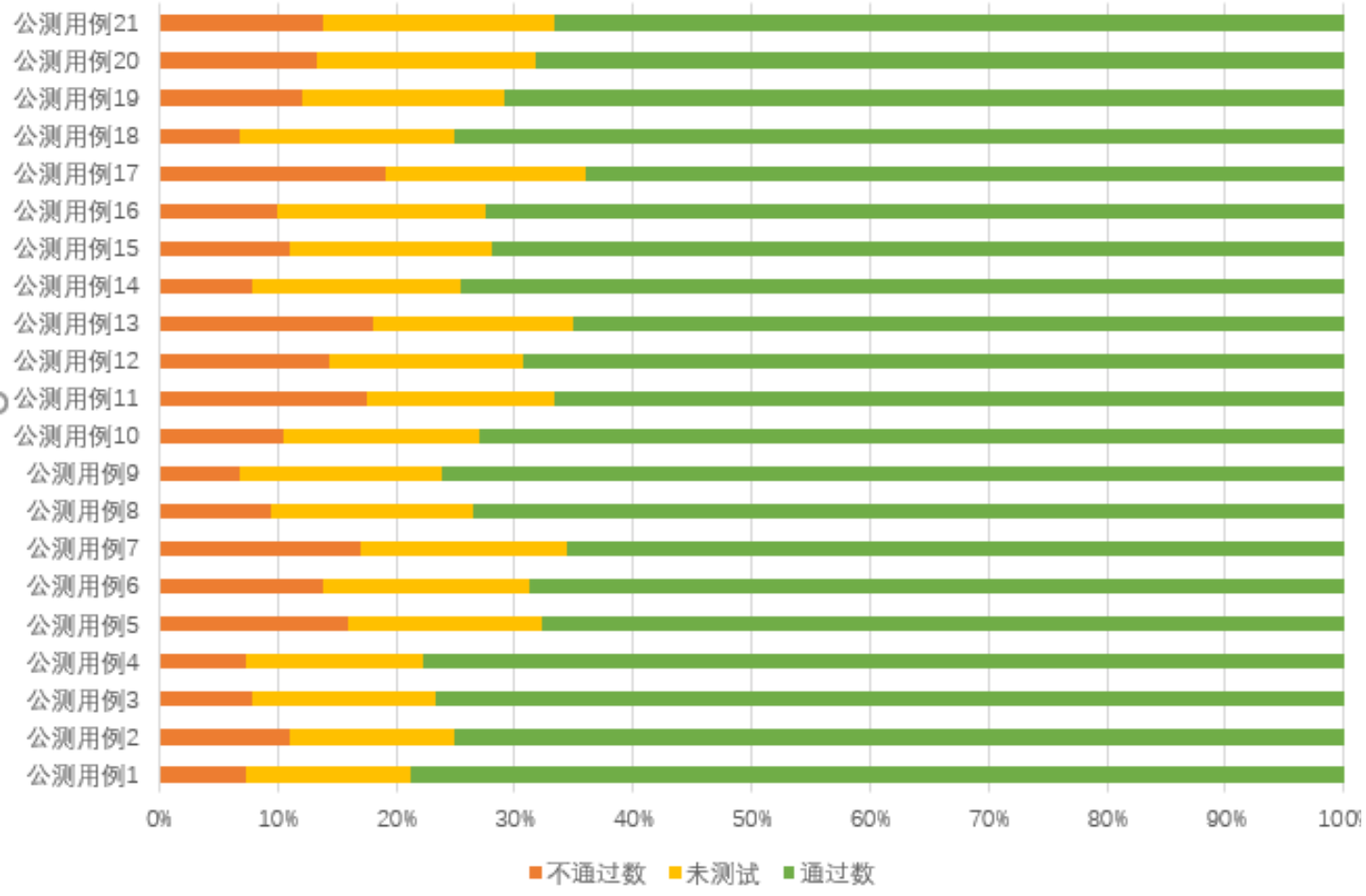
```
//Shared Object body
synchronized method(){
    …
}
```

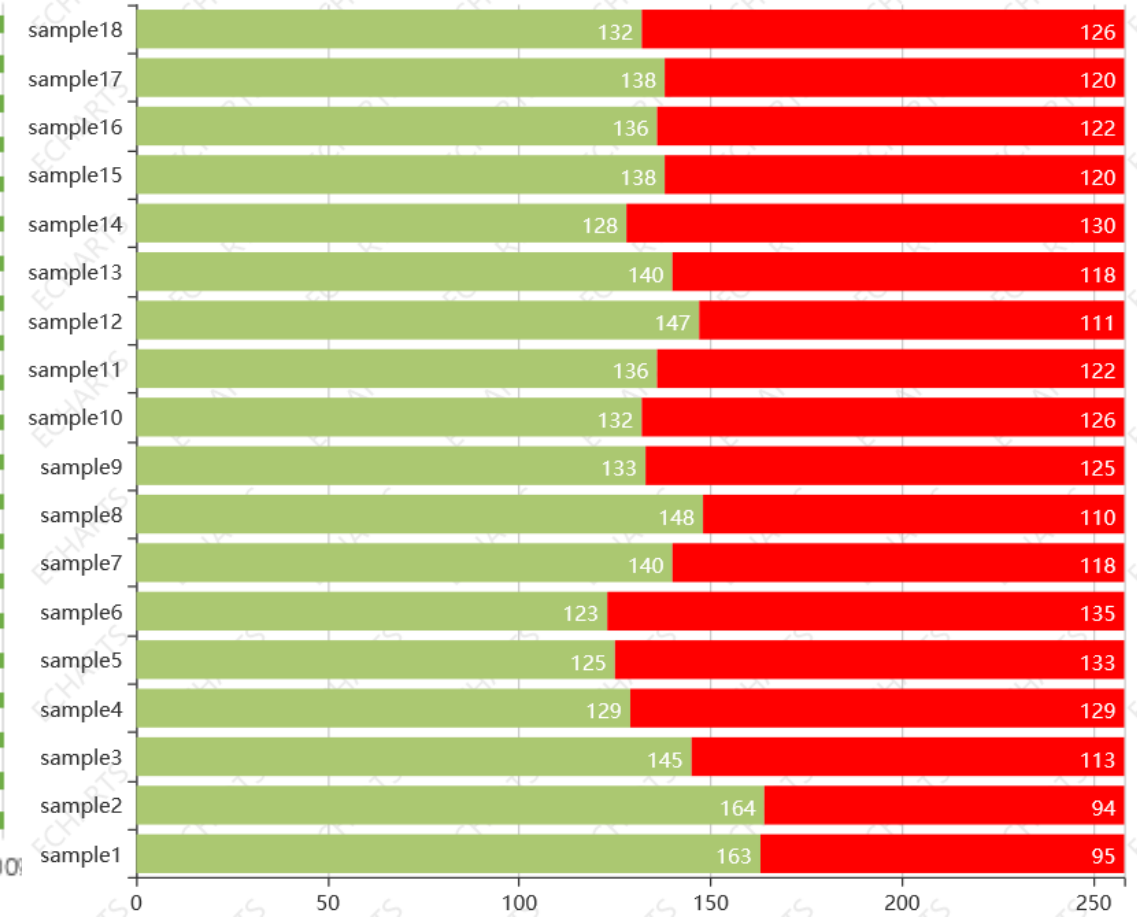
```
//Shared Object body
method(){
    …
    synchronized(e){
        …
        //no actions with e
    }
}
```

作业6训练要点分析

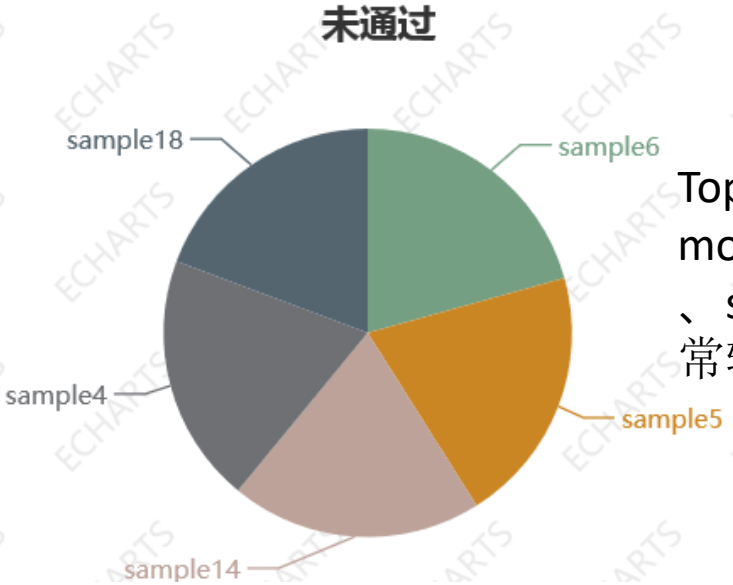
- 实践线程安全设计
  - 所有共享对象都应设计为线程安全的类
  - Snapshot，ChangeBoard，Record
- 编写代码来进行测试
  - 程序输入不是都通过显式的命令行或控制行
  - 大多数软件都采取监听、扫描等方式来获取外部输入，这时手工测试不仅仅是效率问题，而是个可行性问题
  - 测试程序的角色
    - 模拟产生所关注的输入
    - 调用被测模块进行打靶式测试
    - 调用相关模块获取信息做测试正确性判断

作业6公测整体情况对比

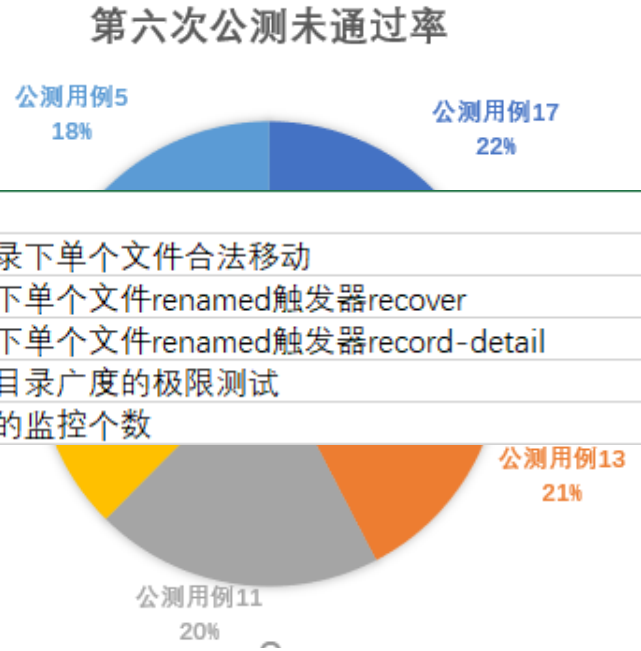




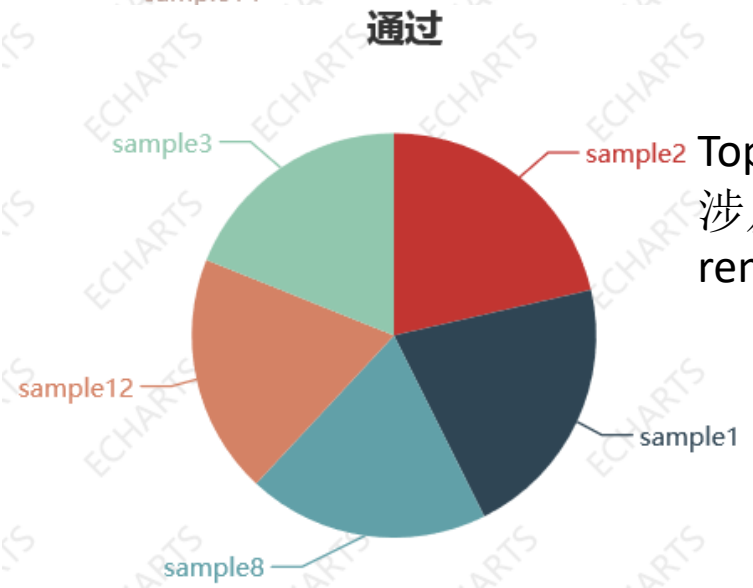
作业6公测疼点和甜点对比



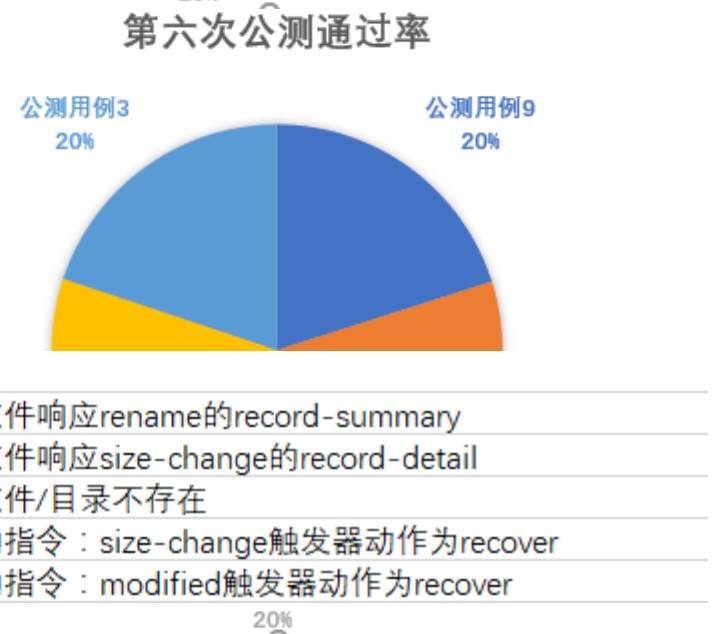
Top5 failed in 2017: modify、path-change、size-change以及异常输入检测。



名称	通过率	未通过	
公测用例17	0.81	0.19	将目录下单个文件合法移动
公测用例13	0.82	0.18	目录下单个文件renamed触发器recover
公测用例11	0.825	0.175	目录下单个文件renamed触发器record-detail
公测用例7	0.831	0.169	较大目录广度的极限测试
公测用例5	0.841	0.159	过多的监控个数



Top5 passed in 2017: 涉及无效监控目录、rename和size-change。



名称	通过率	未通过	
公测用例9	0.931	0.069	监控文件响应rename的record-summary
公测用例18	0.931	0.069	监控文件响应size-change的record-detail
公测用例1	0.926	0.074	监控文件/目录不存在
公测用例4	0.926	0.074	非法的指令：size-change触发器动作为recover
公测用例3	0.921	0.079	非法的指令：modified触发器动作为recover

作业6训练要点分析

- 存在的主要问题
  - 面条代码仍然比较严重
    - 有同学的线程run方法展开去对监控目录进行处理，无法体现层次化设计
  - 未对触发器和处理任务进行适当的抽象，形成继承层次，导致出现较多的剪贴代码（进行文件的访问和处理）
  - 对同步控制锁的理解问题
    - 专门设计实现一个Lock类
  - 使用过多线程，几乎每个目录（子目录）构造一个线程
  - 类的均衡性仍然有待改进

作业7训练要点分析

- 实践面向对象分析
- 认识和实践线程交互的动态性
- 实践线程工作状态控制
- 实践设计原则
- 继续实践线程安全设计

作业7训练要点分析

- 实践面向对象分析
  - 有外到内，首先把系统作为一个整体分析其外部环境的交互
    - 乘客(请求)
    - 出租车
  - 每个对象管理哪些数据？
    - 请求：位置、时间
    - 出租车：状态、位置、信用
  - 对象之间进行哪些交互？
    - 出租车-请求
    - 出租车-路网
    - 出租车-抢单窗口
  - 每个对象行使哪些行为？

作业7训练要点分析

- 认识和实践对象状态变化的动态性
  - 出租车的位置、服务/行驶状态、信用在不断发生变化
  - 抢单窗口具有时效性
- 基于状态的出租车对象管理
  - 物理位置
  - 服务行驶状态
  - 信用
- 设计为线程还是共享对象？
  - 出租车
  - 乘客请求

作业7训练要点分析

- 检查设计原则
  - 具有挑战性
  - 如果脑袋中没有一个设计蓝图，就难以做出有效的检查
    - 必须通过阅读对方代码来识别其设计信息，然后加以判断
    - 这是通往设计师（架构师）的必经之路
  - 这门课重点需要把握的设计要求
    - 类的均衡性
    - 基于继承/接口的层次性
    - 数据管理的局部化
    - 有保护的交互
      - 线程交互
      - 方法调用

作业

- 教学模块问卷调查：了解同学们的学习效果和相关建议
- 总结性博客作业
  - (1)从多线程的协同和同步控制方面，分析和总结自己三次作业来的设计策略及其变化。
  - (2)基于度量来分析自己的程序结构
    - 度量类的属性个数、方法个数、每个方法规模、每个方法的控制分支数目、类总代码规模
    - 计算经典的OO度量
    - 画出自己作业的类图，并自我点评优点和缺点
    - 通过UML的协作图(sequence diagram)来展示线程之间的协作关系（别忘记默认存在的主线程）
    - 从设计原则检查角度，自我检查自己程序的设计，并按照设计原则列出所存在的问题或不足

作业

- (3)分析自己程序的bug
  - 分析未通过的公测用例和被互测发现的bug：特征、问题所在的类和方法
  - 特别注意分析哪些问题与线程安全相关
  - 关联分析bug位置与设计结构之间的相关性
  - 从分类树角度分析程序在设计上的问题
- (4)分析自己发现别人程序bug所采用的策略
  - 列出自己所采取的测试策略及有效性，并特别指出是否结合被测程序的代码设计结构来设计测试用例
  - 分析自己采用了什么策略来发现线程安全相关的问题
- (5) 心得体会
  - 从线程安全和设计原则两个方面来梳理自己在本单元三次作业中获得的心得体会