New Elevator Framework

Table of Contents

[Part 1: 设计思路 2](#_Toc340849867)

[Part 2:程序结构 4](#_Toc340849868)

[1、Loader 4](#_Toc340849869)

[2、NewElevatorFramework 4](#_Toc340849870)

[Part 3:SimulateProgram 5](#_Toc340849871)

[Part 4:Commons 6](#_Toc340849872)

[Part 5:Buttons and ButtonPanels 6](#_Toc340849873)

[Part 6:Passenger 7](#_Toc340849874)

[Part 7:Elevator 8](#_Toc340849875)

[Part 8:Scheduler 9](#_Toc340849876)

[Part 9:Utility 10](#_Toc340849877)

# 

# Part 1: 设计思路

在笔者看来，整个电梯调度的过程中有三个组成部分：乘客群、电梯群、电梯调度器。

现在关键的问题是这三个部分之间是怎么交互的。

就电梯而言，电梯并没有智能，它并不知道有没有人进入它内部，或者说电梯根本不知道进入电梯的物体具体是什么。但是电梯可以知道目前在电梯内部的物体有多重（从很多电梯超重后会有警报也可以看出）。电梯要去哪也是有调度器来告诉它的。

就乘客而言，乘客只关注电梯到了没，去哪，他并不知道电梯的一切，而且也不知道有电梯调度器的存在。乘客和电梯的交互都是通过按钮来执行的。

可见：{乘客对按钮的操作}=>{调度器收到task}=>{调度器把task分给电梯}=>{电梯完成目前assign 给它的task}。

具体一点，整个实体解构应该像下图：

（Defence door 是指保护门，当电梯还没到时保持关闭，避免乘客看到电梯的管道。）

Elevator Scheduler

Floor x

passengers

elevators

Defence

Door

Direction Button

n

如上图，乘客按下direction button ，elevator scheduler就可以通过查看所有button的状态查看到最新按下的按钮，于是，scheduler 就通知一部适合的电梯前往passenger所在地楼层，当电梯到达相应的楼层是，电梯通知scheduler ，scheduler打开相应的defence door让乘客进去。

在电梯内部的过程也一样，而且更简单，在这就不赘述了。

# Part 2:程序结构

“解决方案”NewElevatorFramework包含两个项目：Loader,NewElevatorFramework。

## 1、Loader

所谓的Loader 就是从保存elevators和passengers信息的xml文件中反序列化出相应的信息，用以初始化用于模拟调度的电梯和乘客数据。具体里面的代码不用深究，主要注意的有两点：

1. 在该文件夹下的elevators.xml, passenger1,2,3(.xml),需要是作为输入的数据信息，在将解决方案build好后，运行的时候，输入文件就是这几个中，当然，也可以根据自身的测试需要，按照以上文件的格式生成自己的测试文件。（by the way，elevator.xml里的电梯对passengers.xml里的某个乘客是无力的，你们可以自己看看….）
2. 这个项目内的源代码不用去看，主要注意它们从文件中读取信息后返回的结果结构就行了，这些结构会在项目NewElevatorFramework中用于初始电梯和乘客对象。

## 2、NewElevatorFramework

主程序在SimulateProgram.cs 内，文件SimulateProgram.cs

里只有定义了一个同名的类，顾名思义，class SimulateProgram就是电梯调度的模拟程序。

Commons.cs 定义了整个项目的通用接口。

Buttons.cs实现了Commons中的IButton接口，并定义了实际的两个类型：电梯外地方向按钮类型和电梯内部的楼层按钮类型

ButtonPanel.cs实现了Commons中的IButtonPanel接口，这个类是为了继承出SpecificallyButtonPanel.cs中的两个具体类（电梯外地按钮面板和电梯内的按钮面板）而存在的；这个形式和Button.cs中的实现是一样的，只是class ButtonPanel写的有点大了，所有另分了一个文件写它的两个子类。

Passenger.cs、 Elevator.cs、Scheduler.cs 这三个文件定义的是乘客、电梯、调度器的具体实现。

Utility.cs 中定义了一个静态类Utility，提供一系列帮助调试的工具函数。

# Part 3:SimulateProgram

在主程序中：

* 会先生成一个SimulateProgram的对象；
* 然后载入记录有电梯、乘车信息的xml，完成对电梯对象，乘客的对象的生成和初始化；
* 根据取得的信息构造一个电梯调度器；
* 初始化模拟器的时钟，开始一模拟时钟地每个tick为单位进行电梯调度模拟

SimulateProgram中定义了在模拟过程中乘客的行为：

即passengerTakeActions函数，每次该函数调用，它都会遍历所有乘客，判断他们的状态，并根据他们所处的状态（刚到，已到达，未到达还在等电梯，未到达但已经在电梯内等），驱使乘客采取相应的操作。

elevatorsRun函数，每次该函数调用，它都会督促所有电梯完成本个tick应该完成的事。

schedulerDepatchTasks函数，每次调用它，都会对电梯进行任务分配。

# Part 4:Commons

Commons中分成两个部分，第一部分是各种enum的声明，第二部分是各种接口的声明。

Enum声明中，有两个可能不是很容易理解：

IndexOfOutsidButton，在具体程序程序实现中，每个按钮在按钮面板中都有一个序号，以方便按钮的查找（实际上就是数组和其下标。

MotionOfElevator，这是模拟现实物理世界的运动限制而给出的量，Velocity是电梯的最高运行时速，DecelerationSpace是电梯的减速距离，即电梯在最高速的状态下开始减速后还要运行多远才会停下。

接口声明中，人、电梯、调度器中只有电梯定义了接口，因为在目前的调度结果下，只有电梯可能需要作为参数传来传去，本着减小一点耦合性的观念，给电梯定义了一个接口。

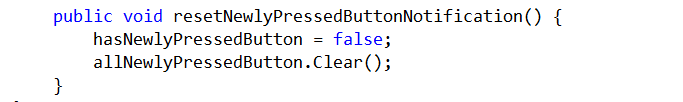
# Part 5:Buttons and ButtonPanels

这两部分中，Button很好懂，ButtonPanel中有一个属性：

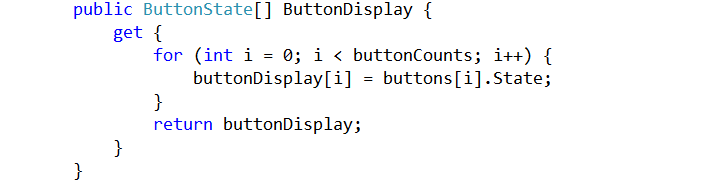
AllNewlyPressedButton，这个属性是和

方法resetNewlyPressedButtonNotification结合在一起的，

所谓的newlyPressedButton，就是从上一次resetNewlyPressedButtonNotification（）执行后到调用newlyPressedButton过程中新按的按钮。（看一下具体实现）



ButtonPanel中的两个数组buttonDisplay和buttons，其实看一下和buttonDisplay 相对应的属性ButtonDisplay就知道，buttonDisplay有两个意义：



1. 在实际中可以认为是乘客可以看到的面板按钮的状态；
2. 在程序中是面板按钮状态的快速查找数组；

# Part 6:Passenger

这个类型即是模拟中的乘客。

在这个类型中可以看出，passenger仅与各种button进行交互：

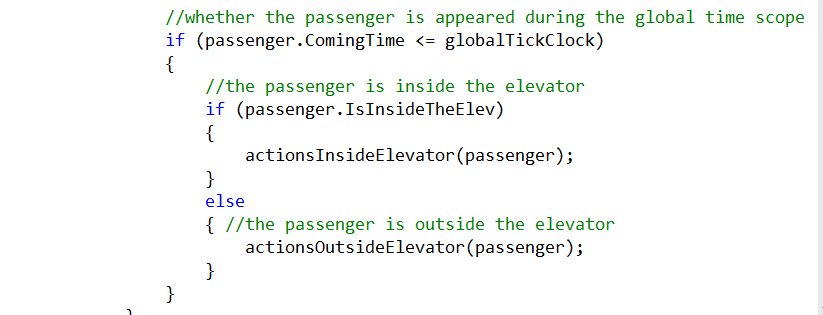
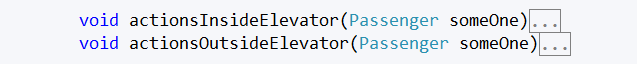


除了按按钮之外，就只有进电梯，出电梯的行为：

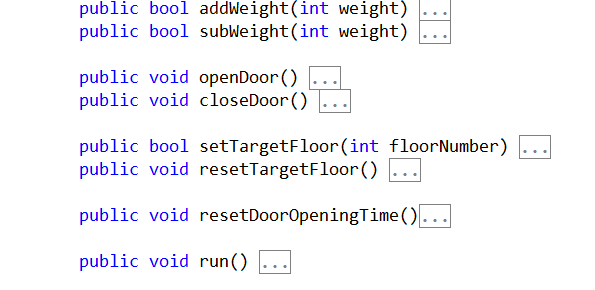


在模拟环境中具体的判断行为(合适进电梯，出电梯)，是在模拟环境下给出的，即(SimulateProgram.cs)：



* 
* 

# Part 7:Elevator

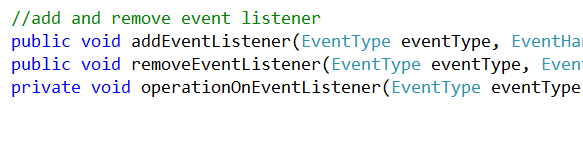


对电梯来说，乘客的进出就是重量的变化，即add、sub weight；

setTargetFloor是确定电梯目前的目的地；resetTargetFloor，是电梯每到一个楼层，在调度器还没有给它新的目的地之前，所进行的操作，表示此时电梯没有要去的目的地。

Run是电梯的运行。

这几个是电梯的基本方法，下面几个是电梯的事件相关的方法：



这里模仿了Javascript的事件添加、移除接口；

这里的EventType指的是开门事件和关门时间，因为电梯开门关门的时候必须通知调度器，这样调度器才好把电梯外地defensive door给关上，并更新相应按钮的状态。

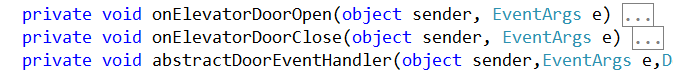
# Part 8:Scheduler

这是电梯调度器。

调度器的使用有点繁琐，首先它有个构造函数，可以通过个构造函数生成一个调度器；然后需要用函数bindWithElevators（）将它与一组电梯绑定在一起，这样才能真正使用。

函数despatchQueriesToElev()是重中之重，因为这里面包含的应该是调度算法。如果使用者愿意的话，可以不用对其他地方进行修改，只在这个函数内填入算法（辅助函数随意加），就可改变整个电梯调度。

目前调度器中自带一个naïve的bus算法，它是让4个电梯逢层就停（不管怎么说，这是个稳定而且一定可行的算法……）。



这是时间处理程序，是调度器对电梯开关门的处理。

# Part 9:Utility

以log开头的三个函数是用来记录日志的，在Utility.cs文件的最开头有两个宏定义，这两个宏定义可以控制日志对控制台的输出；

log是普通日志在Debug模式（vs2012中）下，存在时，会向控制台输出，字体为绿色；

logWarning 是警告日志，在Debug模式，且：

时，向控制台输出，字体为黄色；

logError是错误日志，一定会向控制台输出，字体为红色；

函数saveLogRecord，在debug模式下会将日志保持到文件；

函数outputAnalysisResult，是专门针对已经完成了的模拟，将模拟过程中乘客的信息，时间花费打印，并计算平均时间。

函数howManyPassengerInTotal，返回模拟程序拥有的乘客数；

函数howManyPassengerArrived，返回已经达到的乘客人数；

函数howManyPassengerNonarrived，返回未到达目的地的乘客人数；