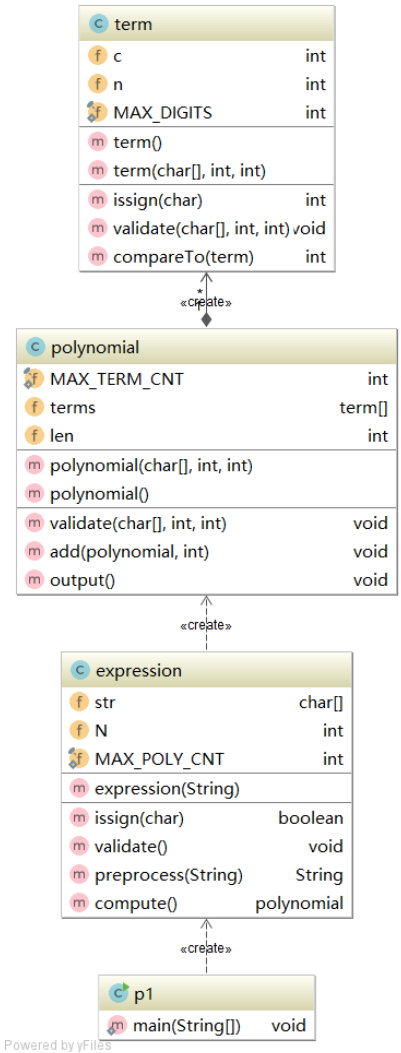
总结性博客（1）

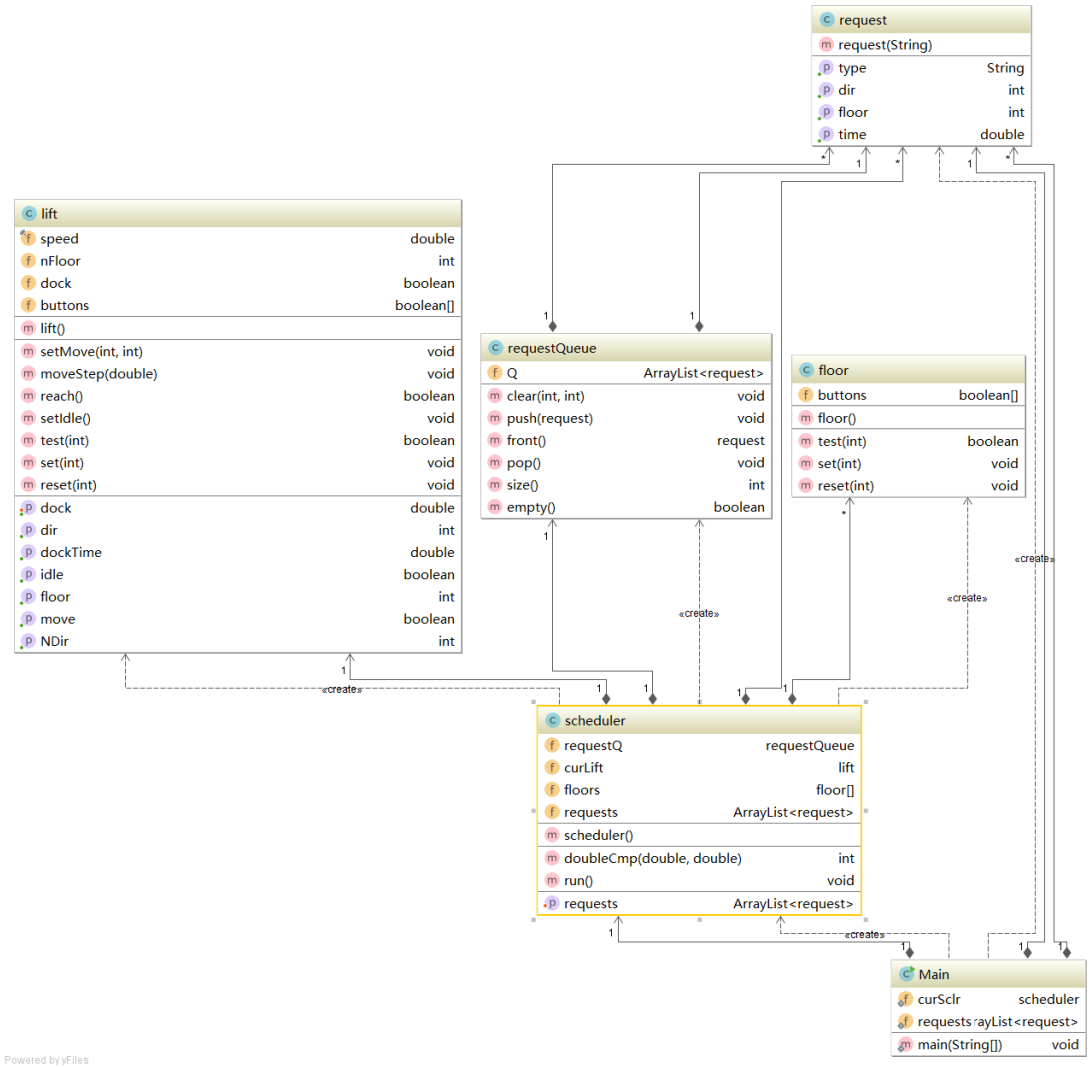
一、程序结构分析

Project 1



程序结构较为简单。入口类创建 expression，expression 类创建 polynomial，polynomial 类创建 term。每一级实例都是通过一个字符串构造，在构造方法中进行解析，将字符串分拆为下一级的待解析字符串，并用该字符串切片构造下一级对象作为当前对象的成员。同时 polynomial 和 expression 类包含计算用方法。各个类的属性个数和方法个数均在 3 个左右，长度分布较为均衡；各方法较为独立，没有同一类中较深的方法调用。唯未使用正则表达式，以至于解析过程较为繁琐。此外，解析过程应与数据分离，而非直接使用字符串构造数据。

Project 2



上图为 Project 2 类图。此时程序结构尚清晰。其主要特点为考虑电梯具有三种内部状态，分别为 move, idle, dock，电梯的运动由状态之间的转移体现。这样的设计使得程序实现较为简便，但实现上存在一些问题。在上面的实现中，我将 lift 的状态控制完全暴露出来，由 scheduler 进行控制。这样的实现导致 lift 的 public 方法过多，但功能却过少，几乎不具备自主控制的功能。与此同时，scheduler 的 run 方法长度达到了 50 行，且内部逻辑关系复杂，难以处理。我原本期望这样的设计能够便于后期开发：新的调度策略只需要重写 run 方法，但这样的设计也难以实现，因为 lift 的仍然包含了一些与调度策略耦合的数据（如 dockTime）。此外，request 解析没有和数据分离仍是问题，且 floor 尚缺乏功能。

Project 3

上图为 Project 3 类图。由于 Project 2 转为 Project 3 中 lift 功能出现大规模变化，以至于 scheduler 大部分方法被 deprecate，为使实际的类间关系更加清晰，上图略去了 ALS_scheduler 的基类 scheduler 和继承关系。程序实现分为以下几个步骤：

1） 输入并根据字符串构造 request 对象并保存。

2） Main 调用 ALS_scheduler 进行调度

a） 检查当前时间是否有新请求，调用 floor 和 lift 类检查是否是相同请求，不是时加入当前请求队列。

b） 如果当前有主请求，则更新捎带请求。

c） 移动电梯；如果到达主请求或捎带请求对应楼层，则使电梯进入 dock 状态。

上述过程中，Project 2 中的问题仍然存在。但 floor 有了一定功能，且除前述问题之外的其它类、方法和属性也尚均衡。

二、程序测试回顾

Project 1~2 的实现较为清晰，没有发现较多 bug。但在实现 Project 3 的过程中，我出现了一系列的 bug。问题主要出在对相同请求的处理上。出现这样的 bug 主要是因为我对 dock 状态的设计存在问题：在进入 dock 状态时电梯就失去了大部分对当前请求的记忆，以至于对相同请求的判定被迫只能根据 scheduler 中保存的相关状态完成，给处理带来了麻烦。这样的设计中 dock 状态难以与 idle 状态区分，无法体现设计 dock 状态的目的（即处理相同请求和设计 still 的请求）。但修改这一实现又会需要大幅度修改 lift，说明 lift 与调度策略的耦合仍然过强。这说明原有的设计仍需改进。

分析其它人的 bug 时我采用的主要方法有阅读对方代码，并考虑可能的问题；此外重要的方法是用程序生成测试数据，并将对方与我的程序的输出进行比较。后者往往能快速发现一些 bug，但通过前者我得以发现了一些隐蔽的 bug。

三、心得体会

在 Project 2 到 Project 3 的迁移中我花费大量时间重写代码，却仍然留下了一系列的 bug，其重要原因在于 Project 2 中尽管前瞻性地留下了策略调整的空间，但这样的空间是我根据可能的策略预留的，而一旦与实际需求不匹配就会带来麻烦。如果在编写 Project 2 时就使用了实现了充分的封装，则带来的麻烦会小得多。从中看来，明晰的工程化方法会比“优雅”的设计更有效。