**学习报告**

6月17日 何田田

## 学习进度

### 6月15日 个人开发流程与工程师成长 ✔



### 6月16日 程序设计的代码风格 ✔



### 6月17日 测试/重构的线上练习



## 个人开发流程与工程师成长

学习目标：项目开发的基本流程、工程师的能力成长方向、如何评价工程师的能力、工程师如何在职场中发展并获得认可，以及何为工程师的“肌肉记忆”。

### 项目的流程

个人软件过程：

规划：对开发、测试的时间分别进行预估

开发：



成熟的工程师并不会没完没了地写程序，而会把更多的时间花在对项目开发效率的改进和质量的把控上。

### 个人能力成长方向

如何衡量和证明自己的能力、着重提高哪些方向的能力。

代码能力：开发实践和积累调试经验。

掌握细分行业的背景知识（如新闻、广告、地图等）：抽象各个行业领域内的问题，以及能够用抽象逻辑来描述并解决行业领域内的问题。

良好的软件架构设计能力和软件工程的思维方式：工程实践、积累经验

职业素养：与人沟通、自我认知与管理、文字文档拟写、合理承诺预期。

### 工程师开发能力的衡量

工作量和项目质量：

1. 项目或者任务到底有多大？

对于一个项目来说，完成的代码行数（Loc, Lines of codes）或者功能点（ Function points）越多，往往意味着项目的任务量越大，开发复杂性也越高。能够开发较大的项目，从一定程度上反映了工程师优秀的开发能力。

1. 项目花费的时间是多少？

对于一个相同规模的项目，一个工程师使用了明显更短的时间，往往说明他具备更优秀的开发能力。

1. 项目的质量如何？

我们同时也要关注项目完成的质量，如果质量不够好的话，即使开发时间相较另一个工程师明显更短，也不能说明这个工程师更出色。

4.项目是否按照预定时间进行了交付？

一是项目完成时间与计划完成时间一致，甚至适度提前；二是各个模块的开发时间与预估开发时间一致。模块交付、项目交付的时间与预估时间尽可能一致，比使用更少的时间进行开发更重要。

对于代码质量的评估，我们可以从两方面入手：

1.测试过程中发现的需要修改的代码行数或者功能数。

2.软件开发结束进行项目交付时，使用者反馈的缺陷的数量。

### 职场中的工程师

软件工程师（ SDE）往往分为初级工程师（SDE I）、中级工程师

（SDE Ⅱ）、高级工程师（ Senior SDE）、首席工程师（ Principal SDE）和一些更高的职位，如：技术合伙人（ Partner Engineer）、杰出工程师（ Distinguished Engineer）、技术专家（ Technical Fellow）等。

其中初级的工程师通常处于刚刚走出校门、但尚未有充分的工程实践的状态；中级的工程师往往可以独立进行开发，也知道在遇到问题时如何描述和解决问题；高级工程师一般为一个小组的领导、或者是团队的技术带头人，影响着一整个小组；对于首席工程师，则大多数会影响一个10人以上的大团队，是影响团队成败的关键角色。

对于技术合伙人、杰出工程师、技术专家这样的高级职务，他们的影响力可能就不仅仅在一个项目中体现，他要确定整个企业的技术风格，让企业的技术有选择地对外输出，并正向影响整个行业的发展。

如何在自己逐步成熟的同时又能得到大家的认可呢？

第一步，你要先学会和技术岗位的其他同事沟通，同时让自己可以和设计师、测试工程师沟通顺畅。你需要以你扎实的技术和良好的沟通表达能力赢得别人的支持，而不是无休止地争论、让办公室充满怨气和不快。

### 工程师的肌肉记忆

一个初学者最需要的就是训练自己的代码能力。

为避免提出这种低层次问题，最好的解决方法是“肌肉记忆”。与其在被人测试时尴尬地犯错，不如早早地加强练习，让你可以在几乎不经过大脑的情况下，把基本的代码写到很难出错。

## 程序设计的代码风格

引言：几名工程师合作开发一个项目时，做的最多的一件事情就是“看代码”，每个人都需要能够看懂其他人的代码。这个时候，代码规范就变得尤为重要。

### 代码没有规范的危害

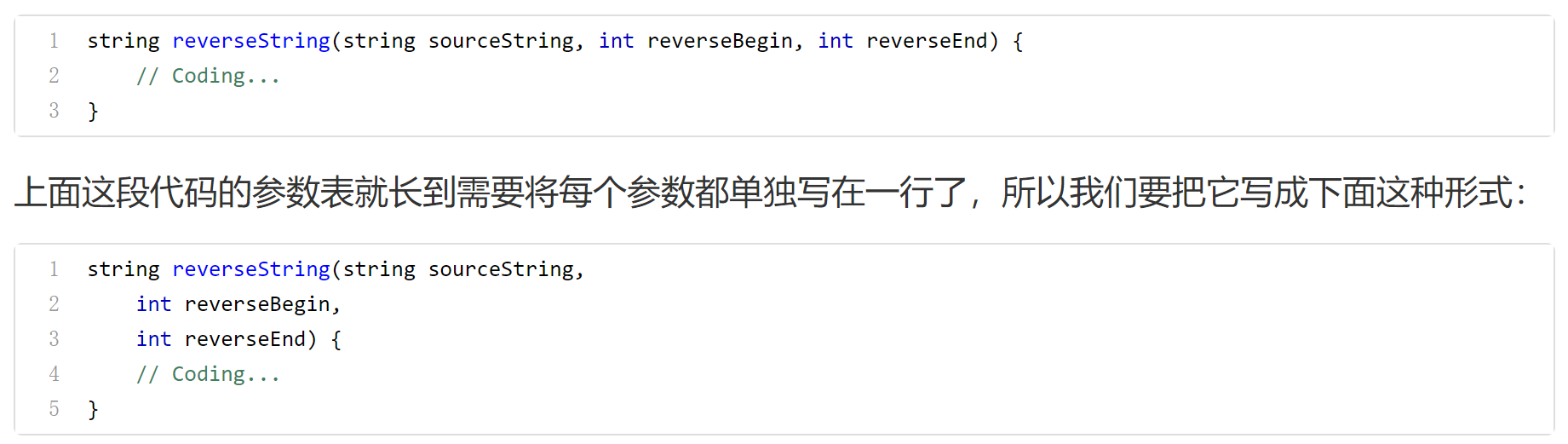
**缩进：4个**（现在有很多的IDE可以通过设置来将Tab键扩展定义为几个空格，Tab键在不同的情况下会显示为不同的长度，而4个空格的距离从可读性来说是最合适的。）

命名风格：格式一致，并且有意义，能根据变量名看出它的作用，千万不要为了方便来使用缩写作为变量名。

### 每行不要写太长

以前的计算机和打印机显示的行宽为80字符，所以我们一般每一行代码的行宽也会限制在**80**字符以内。不过，由于现在的显示器等原因，很多时候我们也会将行宽的限制扩展到**120**字符。

超出行宽的情况一般会有两个，一个是当函数的参数表非常长时，我们会将每个参数都单独写在一行另一个是if语句中的逻辑判断特别多时，我们会将每个&&逻辑单独写成一行。





### 正确使用小括号

在复杂的条件表达式中，一定记得要用括号清楚的表示它们的逻辑优先级。特别是在你记不住它们的优先级分别是怎样的时候，使用括号就更有必要了！



&&前面是0时,&&符号后面的不计算.  
||前面不是0时,||号后面的不计算.

### 加上必要的大括号

在我们的实际编码过程中，大家都会建议无论循环体是只有一行还是有几行，都要把大括号写上不然的话，在有多层控制嵌套时，没有大括号就很难看清楚结构和对应关系

### 命名风格

**常量**：全部大写

**类名**：大驼峰—首字母大写，其后单词首字母大写

**方法函数名**：小驼峰—首字母小写，其后单词首字母大写

**变量名**：小驼峰或下划线—全部小写，单词之间用下划线连接

**系统保留**：前置下划线（\_），在命名中不要使用前置下划线

### 命名方法

声明类的时候，把成员变量都声明为 private

在一般的情况下，我们在操作类的私有变量时，会将给成员变量赋值的方法**以set开头**来命名，取私有变量的值的时候则是**以get开头**来命名。

一个叫做 square（）的方法，这个方法返回一个bool型的值，表示这个矩形是或者不是一个正方形。对于这一类的方法，在命名的时候应当以**is开头**，来代表这是一个返回true或者 false的方法。

### 注释

注释分为“行注释”和“块注释”两种。

行注释和块注释并不是语乂上的“一行的注释”或“一大段的注释”，而是指语法中的意思。例如在C++中，//就是行注释，/\* \*/就是块注释。

我们建议**总是使用行注释**，因为这样我们可以非常方便地去对一整块代码进行注释。



只有3种情况下可以使用块注释：

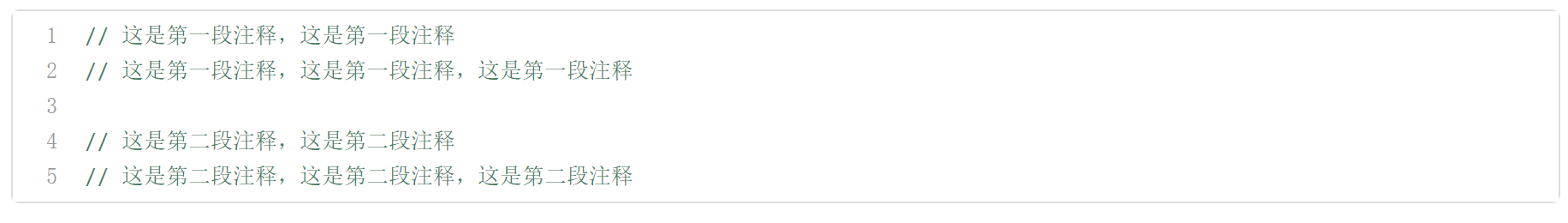
1.当我们需要把一大段代码注释掉的时候；

2.当自动化文档生成工具的语法要求使用块注释，或约定俗成使用块注释作为代码内文档的时候；

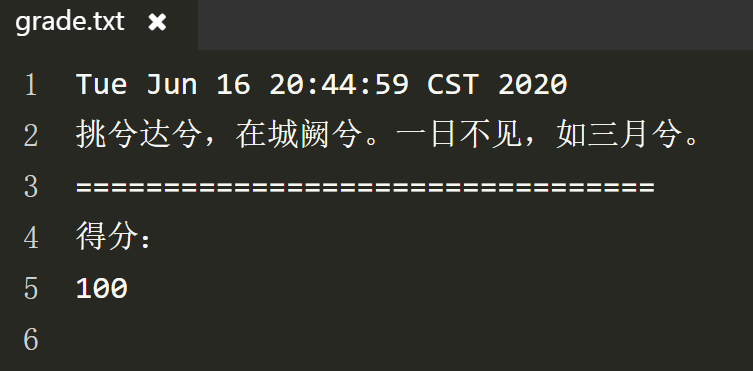
3.编程语言只有块注释语法的时候。

在其他情况下，我们建议永远都不要使用块注释。

多行注释：如果我们的注释是需要跨行的，我们仍然需要使用行注释。



### 改善代码风格 项目实践



这个项目实践确实让我苦恼了一下，第一次做是直接做了一个注释（后来忘记保存了而且注释其实也很混乱），第二次做建了一个笔记，但是还是没有找全，因为平时IDE自动帮忙整理了自己也没有太在意代码风格的问题，很多归纳都是靠看题目代码总结出来，没有办法进行真正意义上的推广举一反三，第三次改才发现makefile这个提示，惭愧的说我之前都不知道这个2333，借着这个机会第一次认识了它，查了一下感觉是又陌生又熟悉，不知道以后会不会再见面hhh



## 测试/重构的线上练习

学习目标：常见的白盒测试方法，包括单元测试、代码审查、集成测试，以及用于评估和指导白盒测试的代码覆盖；如何进行黑盒测试，以及如何设计黑盒测试用例；

在软件开发的过程中，重构常常是软件开发循环的一部分。重构既不修正错误，也不增加新的功能；但是通过重构，可以让代码提升可读性和可维护性。在本章里，你将会了解几种需要被重构的情况，以及如何实现对代码的重构。

### 避免过长的子程序

第一个重构工作—将过长的子程序重构成较小的很多个子程序。

### 合并重复代码

在实际工程中，通过合并重复代码，使得代码量大大降低，可以明显提高代码的可维护性，尤其是在重复代码部分的逻辑需要调整时，可以避免出现忘记修改其中某处导致程序异常的情况

### 合并两个类

不仅仅承担过少职责的类不应该存在，承担过多职责的类同样要避免

### 让if语句更短

### 不要使用全局变量

破坏局域性：

当源代码的每一个模块都非常独立时，代码就会更容易被理解。当全局变量出现在多个子程序里，甚至在多个子程序中被修改时，就破坏了各个子程序的局域性，带来了隐式耦合。

这使得包含这个全局变量的子程序非常难以读懂，维护代码的工程师必须要阅读大量的代码才能厘清程序的数据流—有哪些子程序访问了全局变量的值，又有哪些子程序修改了全局变量的值。

一个比较常见的错误——在C++类定义的头文件中使用了一个或多个全局变量



命名空间污染：

当你将一个变量放在全局而不是某一个命名空间下时，有一定几率会遇到全局变量冲突的情况。幸运的话，变量名冲突会导致链接失败等错误；如果非常不幸，编译器没有提示这个错误的话，各个模块可能会发生意想不到的事情。

