面向对象设计与设计模式

概述

为什么要学习面向对象设计和设计模式

- 程序设计（包括面向对象编程OOP），教你如何编写“正确”的代码；
- 数据结构和算法，教你如何编写“高效率”的代码（时间复杂度、空间复杂度）；
- 面向对象设计（OOD）和设计模式，教你如何编写“高质量”的代码（可读性、可维护性、可扩展性等）

好代码 vs 烂代码

- 写出“能用”代码的人比比皆是，但是，并不是每个人都能写出“好用”的代码。只会写能用的代码，我们永远成长不成大牛，成长不成最优秀的那批人。
- 实际上，写烂代码和好代码花费的时间是差不多的。
- 当你把写高质量代码培养成一种开发习惯之后，在你在编写代码的时候，自然就有一种代码质量意识，自然而然就可以写出不错的代码。即便项目的代码质量因为各种原因有所妥协，但你起码知道什么样的代码是高质量代码，丝毫不影响你具备写出高质量代码的能力。

好代码 vs 烂代码

- 一方面，在目前这种快糙猛的开发环境下，很多人并没有太多时间去思考如何写高质量代码；另一方面，在烂代码的熏陶下，在没有人指导的环境里，很多人也搞不大清楚高质量代码到底长什么样。
- 这就导致很多人写了多年代码，代码功力一点都没长进，编写的代码仍然只是能用即可，能运行就好。平日的工作就是修修补补、抄抄改改，一直在做重复劳动，能力也一直停留在“会干活”的层面，就像高速路上的收银员，只能算是一个“熟练工”。

Talk is cheap，show me the code

- 实际上，代码能力是一个程序员最基础的能力，是基本功，是展示一个程序员基础素养的最直接的衡量标准。你写的代码，实际上就是你名片。
- 看到让人眼前一亮的代码，都会立刻对作者产生无比的好感和认可。
- 从代码就能看出一个人基础是否扎实。代码写得好，能让你脱颖而出。

提高复杂代码的设计和开发能力

- 大部分工程师比较熟悉的都是编程语言、工具、框架这些东西，因为每天的工作就是在框架里根据业务需求，填充代码。
- 但是，开发一个跟业务无关的比较通用的功能模块，面对这样稍微复杂的代码设计和开发，可能就会有点力不从心，想写出易扩展、易用、易维护的代码，并不容易。
- 如何分层、分模块？应该怎么划分类？每个类应该具有哪些属性、方法？怎么设计类之间的交互？该用继承还是组合？该使用接口还是抽象类？怎样做到解耦、高内聚低耦合？该用单例模式还是静态方法？用工厂模式创建对象还是直接 new 出来？如何避免引入设计模式提高扩展性的同时带来的降低可读性问题？……

让读源码、学框架事半功倍

- 优秀的开源项目、框架、中间件，代码量、类的个数都会比较多，类结构、类之间的关系极其复杂，常常调用来调用去。所以，为了保证代码的扩展性、灵活性、可维护性等，代码中会使用到很多设计模式、设计原则或者设计思想。
- 学好面向对象设计和设计模式相关的知识，不仅能让你更轻松地读懂开源项目，还能更深入地参透里面的技术精髓，做到事半功倍。

代码质量评判标准

- 烂代码，比如命名不规范、类设计不合理、分层不清晰、没有模块化概念、代码结构混乱、高度耦合等等。这样的代码维护起来非常费劲，添加或者修改一个功能，常常会牵一发而动全身，让你无从下手，恨不得将全部的代码删掉重写！

如何评价代码质量的高低？

- 灵活性（flexibility）、可扩展性（extensibility）、可维护性（maintainability）、可读性（readability）、可理解性（understandability）、易修改性（changeability）、可复用（reusability）、可测试性（testability）、模块化（modularity）、高内聚低耦合（high cohesion loose coupling）、高效（high efficiency）、高性能（high performance）、安全性（security）、兼容性（compatibility）、易用性（usability）、整洁（clean）、清晰（clarity）、简单（simple）、直接（straightforward）、少即是多（less code is more）、文档详尽（well-documented）、分层清晰（well-layered）、正确性（correctness、bug free）、健壮性（robustness）、鲁棒性（robustness）、可用性（reliability）、可伸缩性（scalability）、稳定性（stability）、优雅（elegant）、好（good）、坏（bad）……

最常用的、最重要的评价标准

- 包括：可维护性、可读性、可扩展性、灵活性、简洁性（简单、复杂）、可复用性、可测试性。

可维护性（maintainability）

- 所谓“代码易维护”就是指，在不破坏原有代码设计、不引入新的 bug 的情况下，能够快速地修改或者添加代码。
- 所谓“代码不易维护”就是指，修改或者添加代码需要冒着极大的引入新 bug 的风险，并且需要花费很长的时间才能完成。
- 如果代码分层清晰、模块化好、高内聚低耦合、遵从基于接口而非实现编程的设计原则等等，那就可能意味着代码易维护。
- 如果 bug 容易修复，修改、添加功能能够轻松完成，那我们就可以主观地认为代码对我们来说易维护。

可读性（readability）

- 软件设计大师 Martin Fowler 曾经说过："**Any fool can write code that a computer can understand. Good programmers write code that humans can understand.**" 翻译成中文就是：“任何傻瓜都会编写计算机能理解的代码。好的程序员能够编写人能够理解的代码。”
- 代码的可读性应该是评价代码质量最重要的指标之一。
- 我们需要看代码是否符合编码规范、命名是否达意、注释是否详尽、函数是否长短合适、模块划分是否清晰、是否符合高内聚低耦合等等。

可扩展性（extensibility）

- 代码的可扩展性表示，我们在不修改或少量修改原有代码的情况下，通过扩展的方式添加新的功能代码。
- 说直白点就是，代码预留了一些功能扩展点，你可以把新功能代码，直接插到扩展点上，而不需要因为要添加一个功能而大动干戈，改动大量的原始代码。
- 关于代码的扩展性，在后面讲到“对修改关闭，对扩展开放”这条设计原则的时候，会有详细讲解。

可复用性（reusability）

- 代码的可复用性可以简单地理解为，尽量减少重复代码的编写，复用已有的代码。
- 代码可复用性跟 DRY（Don’t Repeat Yourself）这条设计原则的关系紧密。

灵活性（flexibility）

- 如果一段代码易扩展、易复用或者易用，我们都可以称这段代码写得比较灵活。

简洁性（simplicity）

- 有一条非常著名的设计原则，即KISS 原则：“Keep It Simple，Stupid”。
- 思从深而行从简，真正的高手能云淡风轻地用最简单的方法解决最复杂的问题。这也是一个编程老手跟编程新手的本质区别之一。

可测试性（testability）

- 代码可测试性的好坏，能从侧面上非常准确地反应代码质量的好坏。
- 代码的可测试性差，比较难写单元测试，那基本上就能说明代码设计得有问题。

如何才能写出高质量的代码？

- 这个问题等同于在问，如何写出易维护、易读、易扩展、灵活、简洁、可复用、可测试的代码？
- 要写出满足这些评价标准的高质量代码，需要掌握一些更加细化、更加能落地的编程方法论，包括面向对象设计思想、设计原则、设计模式、编码规范、重构技巧等。而所有这些编程方法论的最终目的都是为了编写出高质量的代码。

五者的关系

- 关于面向对象、设计原则、设计模式、编程规范和代码重构，这五者的关系。
- 面向对象编程因为其具有丰富的特性（封装、抽象、继承、多态），可以实现很多复杂的设计思路，是很多设计原则、设计模式等编码实现的基础。

五者的关系

- 设计原则是指导我们代码设计的一些经验总结，对于某些场景下，是否应该应用某种设计模式，具有指导意义。
- 设计模式是针对软件开发中经常遇到的一些设计问题，总结出来的一套解决方案或者设计思路。应用设计模式的主要目的是提高代码的可扩展性。
- 从抽象程度上来讲，设计原则比设计模式更抽象。设计模式更加具体、更加可执行。

五者的关系

- 编程规范主要解决的是代码的可读性问题。编码规范相对于设计原则、设计模式，更加具体、更加偏重代码细节、更加能落地。
- 持续的小重构依赖的理论基础主要就是编程规范。重构作为保持代码质量不下降的有效手段，利用的就是面向对象、设计原则、设计模式、编码规范这些理论。
- 实际上，面向对象、设计原则、设计模式、编程规范、代码重构，这五者都是保持或者提高代码质量的方法论，本质上都是服务于编写高质量代码这一件事的。

面向对象

- 封装、抽象、继承、多态
- 面向对象编程 与 面向过程编程
- 面向对象分析、设计、编程
- 接口 与 抽象类
- 基于接口而非实现编程
- 多用组合少用继承
- 贫血模型和充血模型

设计原则

- SOLID原则之SRP单一职责原则
- SOLID原则之OCP开闭原则
- SOLID原则之LSP里氏替换原则
- SOLID原则之ISP接口隔离原则
- SOLID原则之DIP依赖倒置原则
- DRY原则、KISS原则、YAGNI原则、LOD法则

编程规范与代码重构

- 改善代码质量的编程规范
- 代码重构的目的、对象、时机、方法
- 单元测试和代码的可测试性
- 大重构（大规模高层次）
- 小重构（小规模低层次）

设计模式

- GOF经典23个设计模式：
- 创建型：单例、工厂方法、建造者、原型、抽象工厂
- 结构型：代理、桥接、装饰者、适配器、门面、组合、享元
- 行为型：观察者、模板、策略、职责链、迭代器、状态、访问者、备忘录、命令、解释器、中介

课程主要内容

- 设计原则与思想
  - 代码质量评判标准
  - 面向对象、设计原则、编程规范、代码重构
- 设计模式与范式
  - 创建型、结构型、行为型

参考资料

- 主要参考资料
- 1. 《设计模式之美》，王争，极客时间专栏课程。
- 2. 《图解设计模式》，【日】结诚浩 著，人民邮电出版社
- 其他参考资料
- 3. 《设计模式：可复用面向对象软件的基础》GoF 机械工业出版社
- 4. 《设计模式之禅（第二版）》，秦小波 著，机械工业出版社

Quotes

- ***Talk is cheap, show me the code.***

  -- Linus Torvalds

- ***Any fool can write code that a computer can understand.***
  Good programmers write code that humans can understand.

  -- Martin Fowler