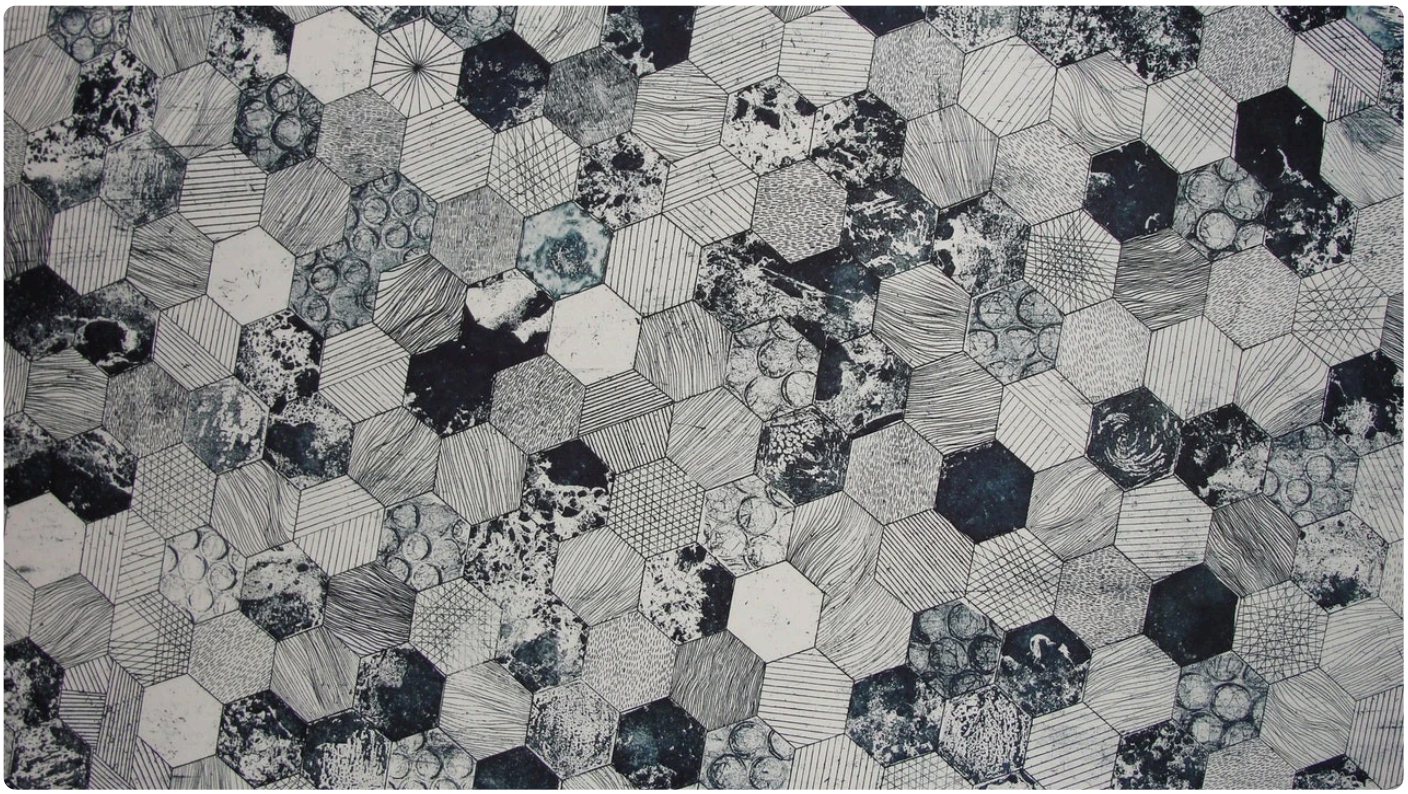
# 03 | 面向对象、设计原则、设计模式、编程规范、重构，这五者有何关系？

王争 · 设计模式之美



在上一节课中，我们讲到，要具备编写高质量代码的能力，你需要学习一些编程方法论，其中就包含面向对象（我们可以把它看成一种设计思想）、设计原则、设计模式、编程规范、重构技巧等。而我们整个专栏的内容也是围绕着这几块展开讲解的。所以，今天我就先来简单介绍一下这几个概念，并且说一说它们之间的联系。

今天的内容相当于专栏的一个教学大纲，或者说学习框架。它能让你对整个专栏所涉及的知识点，有一个全局性的了解，能帮你将后面零散的知识更系统地组织在大脑里。

话不多说，我们就一块来看一下，接下来的这 8 个月我们到底要学习哪些内容吧！

## 面向对象

现在，主流的编程范式或者是编程风格有三种，它们分别是面向过程、面向对象和函数式编程。面向对象这种编程风格又是这其中最主流的。现在比较流行的编程语言大部分都是面向对

象编程语言。大部分项目也都是基于面向对象编程风格开发的。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、设计模式编码实现的基础。

所以，在专栏的最开始，我们会详细地讲解面向对象编程的相关的知识，为学习后面的内容做铺垫。对于这部分内容，你需要掌握下面这 7 个大的知识点。

面向对象的四大特性：封装、抽象、继承、多态

面向对象编程与面向过程编程的区别和联系

面向对象分析、面向对象设计、面向对象编程

接口和抽象类的区别以及各自的应用场景

基于接口而非实现编程的设计思想

多用组合少用继承的设计思想

面向过程的贫血模型和面向对象的充血模型

## 设计原则

设计原则是指导我们代码设计的一些经验总结。设计原则这块儿的知识有一个非常大的特点，那就是这些原则听起来都比较抽象，定义描述都比较模糊，不同的人会有不同的解读。所以，如果单纯地去记忆定义，对于编程、设计能力的提高，意义并不大。对于每一种设计原则，我们需要掌握它的设计初衷，能解决哪些编程问题，有哪些应用场景。只有这样，我们才能在项目中灵活恰当地应用这些原则。

对于这一部分内容，你需要透彻理解并且掌握，如何应用下面这样几个常用的设计原则。

SOLID 原则 –SRP 单一职责原则

SOLID 原则 –OCP 开闭原则

SOLID 原则 –LSP 里式替换原则

SOLID 原则 –ISP 接口隔离原则

SOLID 原则 –DIP 依赖倒置原则

DRY 原则、KISS 原则、YAGNI 原则、LOD 法则

## 设计模式

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。大部分设计模式要解决的都是代码的可扩展性问题。设计模式相对于设计原则来说，没有那么抽象，而且大部分都不难理解，代码实现也并不复杂。这一块的学习难点是了解它们都能解决哪些问题，掌握典型的应用场景，并且懂得不过度应用。

经典的设计模式有 23 种。随着编程语言的演进，一些设计模式（比如 Singleton）也随之过时，甚至成了反模式，一些则被内置在编程语言中（比如 Iterator），另外还有一些新的模式诞生（比如 Monostate）。

在专栏中，我们会重点讲解 23 种经典的设计模式。它们又可以分为三大类：创建型、结构型、行为型。对于这 23 种设计模式的学习，我们要有侧重点，因为有些模式是比较常用的，有些模式是很少被用到的。对于常用的设计模式，我们要花多点时间理解掌握。对于不常用的设计模式，我们只需要稍微了解即可。

我按照类型和是否常用，对专栏中讲到的这些设计模式，进行了简单的分类，具体如下所示。

### 1. 创建型

常用的有：单例模式、工厂模式（工厂方法和抽象工厂）、建造者模式。

不常用的有：原型模式。

### 2. 结构型

常用的有：代理模式、桥接模式、装饰者模式、适配器模式。

不常用的有：门面模式、组合模式、享元模式。

### 3. 行为型

常用的有：观察者模式、模板模式、策略模式、职责链模式、迭代器模式、状态模式。

不常用的有：访问者模式、备忘录模式、命令模式、解释器模式、中介模式。

## 编程规范

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。即便你可能对设计原则不熟悉、对设计模式不了解，但你最起码要掌握基本的编码规范，比如，如何给变量、类、函数命名，如何写代码注释，函数不宜过长、参数不能过多等等。

对于编码规范，考虑到很多书籍已经讲得很好了（比如《重构》《代码大全》《代码整洁之道》等）。而且，每条编码规范都非常简单、非常明确，比较偏向于记忆，你只要照着来做可以。它不像设计原则，需要融入很多个人的理解和思考。所以，在这个专栏中，我并没有花太多的篇幅来讲解所有的编码规范，而是总结了我认为的最能改善代码质量的 20 条规范。如果你暂时没有时间去看那些经典的书籍，看我这些就够了。

除此之外，专栏并没有将编码规范单独作为一个模块来讲解，而是跟重构放到了一起。之所以这样做，那是因为我把重构分为大重构和小重构两种类型，而小重构利用的知识基本上就是编码规范。

除了编码规范，我们还会介绍一些代码的坏味道，让你知道什么样的代码是不符合规范的，应该如何优化。参照编码规范，你可以写出可读性好的代码；参照代码的坏味道，你可以找出代码存在的可读性问题。

## 代码重构

在软件开发中，只要软件在不停地迭代，就没有一劳永逸的设计。随着需求的变化，代码的不停堆砌，原有的设计必定会存在这样那样的问题。针对这些问题，我们就需要进行代码重构。重构是软件开发中非常重要的一个环节。持续重构是保持代码质量不下降的有效手段，能有效避免代码腐化到无可救药的地步。

而重构的工具就是我们前面罗列的那些面向对象设计思想、设计原则、设计模式、编码规范。实际上，设计思想、设计原则、设计模式一个最重要的应用场景就是在重构的时候。我们前面讲过，虽然使用设计模式可以提高代码的可扩展性，但过度不恰当地使用，也会增加代码的复杂度，影响代码的可读性。在开发初期，除非特别必须，我们一定不要过度设计，应用复杂的设计模式。而是当代码出现问题的时候，我们再针对问题，应用原则和模式进行重构。这样就能有效避免前期的过度设计。

对于重构这部分内容，你需要掌握以下几个知识点：

重构的目的（why）、对象（what）、时机（when）、方法（how）；

保证重构不出错的技术手段：单元测试和代码的可测试性；

两种不同规模的重构：大重构（大规模高层次）和小重构（小规模低层次）。

希望你学完这部分内容之后，不仅仅是掌握一些重构技巧、套路，更重要的是建立持续重构意识，把重构当作开发的一部分，融入到日常的开发中。

## 五者之间的联系

关于面向对象、设计原则、设计模式、编程规范和代码重构，这五者的关系我们前面稍微提到了一些，我这里再总结梳理一下。

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。

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。比如，“开闭原则”是很多设计模式（策略、模板等）的指导原则。

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。

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。当我们追本逐源，看清这个本质之后，很多事情怎么做就清楚了，很多选择怎么选也清楚了。比如，在某个场景下，该不该用这个设计模式，那就看能不能提高代码的可扩展性；要不要重构，那就看重代码是否存在可读、可维护问题等。

## 重点回顾

今天的内容到此就讲完了。我画了一张图，总结了专栏中所涉及的知识点。在学习后面的课程的时候，你可以经常翻出来看一下，建立全局意识，不至于迷失在零碎的知识点中。

## 课堂讨论

今天课堂讨论的话题有两个。

1. 在今天讲到的内容中，你觉得哪一部分内容对提高代码质量最有效？为什么？除了我罗列的这些内容之外，你还知道哪些可以提高代码质量的方法？
2. 我们知道，最经典的设计模式书籍是 GoF 的《设计模式》，它的中文全称是《设计模式：可复用面向对象软件的基础》，英文全称是“Design Patterns: Elements of Reusable Object-Oriented Software”。为什么它在标题中会特意提到“面向对象”呢？

欢迎在留言区写下你的想法，和同学一起交流和分享。如果有收获，也欢迎你把这篇文章分享给你的朋友。

**AI智能总结**

本文深入探讨了面向对象编程、设计原则、设计模式、编程规范和代码重构之间的关系。首先介绍了面向对象编程的基本概念和特性，然后详细讲解了设计原则，包括SOLID原则、DRY原则、KISS原则、YAGNI原则和LOD法则。接着，文章介绍了设计模式的分类和应用场景，包括创建型、结构型和行为型设计模式。编程规范部分主要解决代码的可读性问题，包括变量、类、函数命名规范等。最后，文章强调了代码重构的重要性，介绍了重构的目的、对象、时机和方法，以及保证重构不出错的技术手段。整体而言，本文以系统性的方式介绍了编写高质量代码所需的基本概念和技能，对于想要提高编程能力的读者具有很高的参考价值。文章深入浅出地阐述了面向对象编程、设计原则、设计模式、编程规范和代码重构之间的联系，为读者提供了全面的技术视角和实用指导。



## 全部留言 (365)

最新　精选

**陈拾柒**

2019-11-15

再好的理论，应用不到实际中也是白费。所以对于提升代码质量，最有效的是编码规范，其次是设计原则，再次是代码重构，最后才是面向对象和设计模式。
整个排序是按照我理解的难易程度来进行的。一般一个项目都是多人开发，多人并行开发中，考虑到团队中技术水平不一，保持代码质量最好的方式就是先制定编码规范，大家最容易达成一致并遵守。其次是设计原则，因为它相对来讲更明确，违反设计原则基本等于硬伤，更容易

得到认同。代码重构是需要长期去做的，甚至不是在开发过程中而是在维护过程中去做的，所以排在第三位。后面两项更多的要考虑实际情况，不同水平的人甚至相同水平的人都会有自己不同的理解，比较难以达成统一，项目排期足够还好，一旦排期不足，很可能在当时没那个成本去谈论这些。

第二个问题老师已经回答了，因为面向对象的特性是其他的基石。

建议老师在后续课程中，除了讨论如何提升自己的代码质量，也讨论一下在一个多人团队中如何提升团队的代码质量，有哪些比较好的手段可以去应用。包括在项目排期比较紧的时候，可以有哪些方式去保证代码质量。

作者回复: 嗯嗯，加餐里有讲到的。

共 6 条评论> 72

Yayu
2019-11-05

面向对象的特征也未必包含“继承”这一点吧，比如 Go 语言就没有提供“继承”这个特性，取而代之的是，推荐使用“组合”。但不能说它不支持面向对象编程。那么我们在探讨“面向对象”这个范式时，需要更深刻的去思考“面向对象”的本质是什么。而不是用Java 中的概念来一以概之。希望王争老师参考。

作者回复: 后面会讲到，等讲到了就明白了

共 10 条评论> 18

zachary
2019-11-06

争哥，设计模式有三种类型：
创建型：这个好理解，就是用来创建对象的一些模式
结构型和行为型不好理解，结构体现在哪里？行为又体现在哪里？能否说的更具体点？

作者回复: 后面会讲到的，别急哈

共 2 条评论> 7

Miaozhe
2019-11-08

我看到设计原则中还有迪米特法则(Law of Demeter, LoD)，这个也是解决God类的一种方式。另外这个法则，好像跟组合复用相背呢。帮忙解释一下。

作者回复: 学了后面章节就明白了

4

**胖大海**
2019-11-05

对于重构，我有话（槽）要说（吐）：我是做企业项目外包的，我们给客户交付的项目大多在客户方面庞杂缺少弹性的IT合规性要求和业务快速变动的现实的双重夹击下痛苦地演进着。重构意味着不改变系统已有功能的情况下优化项目代码，但面临着合规性和流程的限制导致这类无业务功能更新的发布不会被甲方IT放行；另一方面不断的新需求的变更又不断地劣化着已有代码的系统架构以及业务代码的设计。不知道有没有面临同样问题的同学，在无法改变现有大环境的前提下进行有限的优化？我们现在能做的也就是在每一次追加的新需求变更时进行小幅度的改善，但这无法受到任何工作流程的保护和管理，仅能靠程序员个人的良心来做。另外，如何保证重构不会因为代码结构的变化引入新的bug呢？单元测试以及覆盖率较高的自动化测试吗？

作者回复: 单元测试是一个非常有效的手段，后面的章节中会讲到。

共 2 条评论>

4

**初学者**
2020-11-23

一直跟着pojo，controller，service转写不出好的代码

作者回复: 是的

3

**Geek_222ec6**
2019-11-09

设计原则是指导设计模式的更抽象的理论（这么说应该对吧）

另外有个困惑就是在编码规范的时候，自我感觉英语还可以，但是函数，变量命名的时候经常会犯纠结症和强迫症。。。不知道怎么命名才好。

作者回复: 理解是对的。这个正常，一个好的命名确实要花点时间想的

共 2 条评论>

3

**奇小易**

2019-11-27

我觉得编程规范是最有效的一种方法，选择它就表示认可了可读性是代码质量最核心的一点。因为它不仅好学、好上手，用起来效果还明显。不像其它的方法对个人经验的依赖性强。还有个例子证明它的重要性，听说Google入职前几个月最先做的事情是把他们的编程规范熟记于心。

作者回复: 有code readability认证，需要每个人通过

2

**大豆腐**

2019-11-05

面向对象的特性，有些地方说是4个，有些地方说是3个，一脸懵逼

作者回复: 我后面的文章会讲到的 别急

2

**划时代**

2019-11-05

老师我有疑问，在没有充分必要的情况下是不是不采用面向对象编程（继承+多态），大多数情况下使用基于对象的方式进行编程（封装+组合）？

作者回复: 哈哈 感觉你说的继承、多态、封装、组合可以任意组合起来用啊。并没有说面向对象就是继承+多态，基于对象就是封装+组合呀

共 2 条评论>

1