导读：如何学好设计模式

什么是设计模式

> 设计模式（Design Pattern）代表了最佳的实践，通常被有经验的面向对象的软件开发人员所采用。设计模式是软件开发人员在软件开发过程中面临的一般问题的解决方案。这些解决方案是众多软件开发人员经过相当长的一段时间的试验和错误总结出来的。

上面的解释来自于某度某科，是比较标准的定义，可以从中筛选出几个关键字来帮助我们理解什么是设计模式：

- 最佳实践
- 解决方案
- 试验和错误总结

从上面的三个关键词中可以总结出，设计模式就是在针对编码过程中遇到的问题总结出来的最佳解决方案。

那么这些问题指的是什么问题呢？

面向对象的程序应该具有可维护性、代码可复用性、扩展性及灵活性，要解决的问题就是代码可维护性问题、复用性问题、扩展性问题、灵活性问题。

简单来说，设计模式就是指导你如何写出可维护、可复用、可扩展及灵活的代码。

设计模式分类

设计模式总共有23种，总体来说可以分为三大类：**创建型模式（Creational Patterns）、结构型模式（Structural Patterns）和行为型模式（Behavioral Patterns）。**

分类	关注点	包含
创建型模式	关注于对象的创建，同时隐藏创建逻辑	工厂模式 抽象工厂模式 单例模式 建造者模式 原型模式
结构型模式	关注类和对象之间的组合	适配器模式 过滤器模式 装饰模式 享元模式 代理模式 外观模式 组合模式 桥接模式
行为型模式	关注对象之间的通信	责任链模式 命令模式 中介者模式 观察者模式 状态模式 策略模式 模板模式 空对象模式 备忘录模式 迭代器模式 解释器模式 访问者模式

上面的三种分类，有助于在开发时思考当前场景应该使用哪种分类。

大家不一定要全部记住，有个大概的了解即可。

学习设计模式

为什么要学设计模式

写出可维护、可复用、可扩展及灵活的代码是我们的目的，也是学习设计模式的理由，但是这个理由对我们来说太抽象，下面从“读”和“写”两方面来说明到底为什么要学习设计模式。

读

作为开发人员，不可避免的要接触其他人写的代码，有的是一些知名的库或框架，例如 Spring、Shiro 等。

但是当我们去阅读这些框架源码的时候会发现无从下手，因为类太多了，关系太复杂，而且很多类的命名看不懂，比如 xxxBuilder、xxxStrategy、xxxFilter 等，一个词看不懂就可能导致你直接放弃继续阅读。

如果没有学过设计模式，自然看不懂，学习设计模式可以有效的帮助你阅读代码，即便不能百分百帮到你，至少也能帮到百分之三四十。

写

每一个开发人员必然喷过其他人写的代码，觉得其他人的代码写的很垃圾，尤其是要扩展功能或者修改功能的时候，恨不得全部删掉重新再写，其实在其他人看来你的代码也是如此。所以写出一手让人无话可说的代码是很有必要的，不仅可以满足你的小小成就感，也可以让你的程序更快速稳定的发展。

在一个项目组中，如果大家都学习过设计模式，那么当你阅读或修改同事写的代码时也将得心应手，少了很多麻烦。

如何学好设计模式

现如今网上和书上都有大量的设计模式的教程，但是他们大部分都有一个共同点：仅仅使用生活中的例子。

比如前几年我第一次学习设计模式，在学到适配器模式时，教程中抛出了一个电器的插头问题：

> 你家插座只有三头的，但电器插头是两头的，咋办？弄个插头适配器将两头转换成三头。

nice，这个例子简单明了，作为新手的我瞬间明白了适配器的含义，就是在不兼容的双方中间做一层转化。

但是后来发现在实际编码中根本用不上这个设计模式，因为我根本不会用。

生活中的例子的确可以帮助我们理解设计模式，这是毋庸置疑的，但是想要真正用好设计模式，实际项目中的案例是必不可少的，这也是我写这门课的原因，希望通过分析实际案例，能够帮到更多想要学习设计模式的同行。

下面给出几点更加具体的建议：

- 从生活例子中去理解设计模式；
- 从实际案例去了解设计模式的使用场景；
- 动手实践，在学完实际案例之后，不妨动手写一写，不要写生活中的例子，自己构造一个小功能，用上你的设计模式；
- **改变自己的意识**，在开发或修改一个功能时，首先要下意识地去思考这个功能将来在修改和扩展上会遇到什么问题，能否使用上设计模式。记住一定要思考、一定要思考、一定要思考，即便最终用不上，也能让你回顾一遍设计模式的内容，使其知识更牢固。很多开发者不是不会用，而是根本没有想过要用设计模式，久而久之这方面的能力自然就弱化了。

课程说明

课程内容

本课程每一篇文章主要包含三大部分：

- 解释和理解设计模式；
- 至少介绍一个实际案例（实际案例有些是我自己写的，有些来自于已有的框架或库）；
- 设计模式优缺点。

必要准备

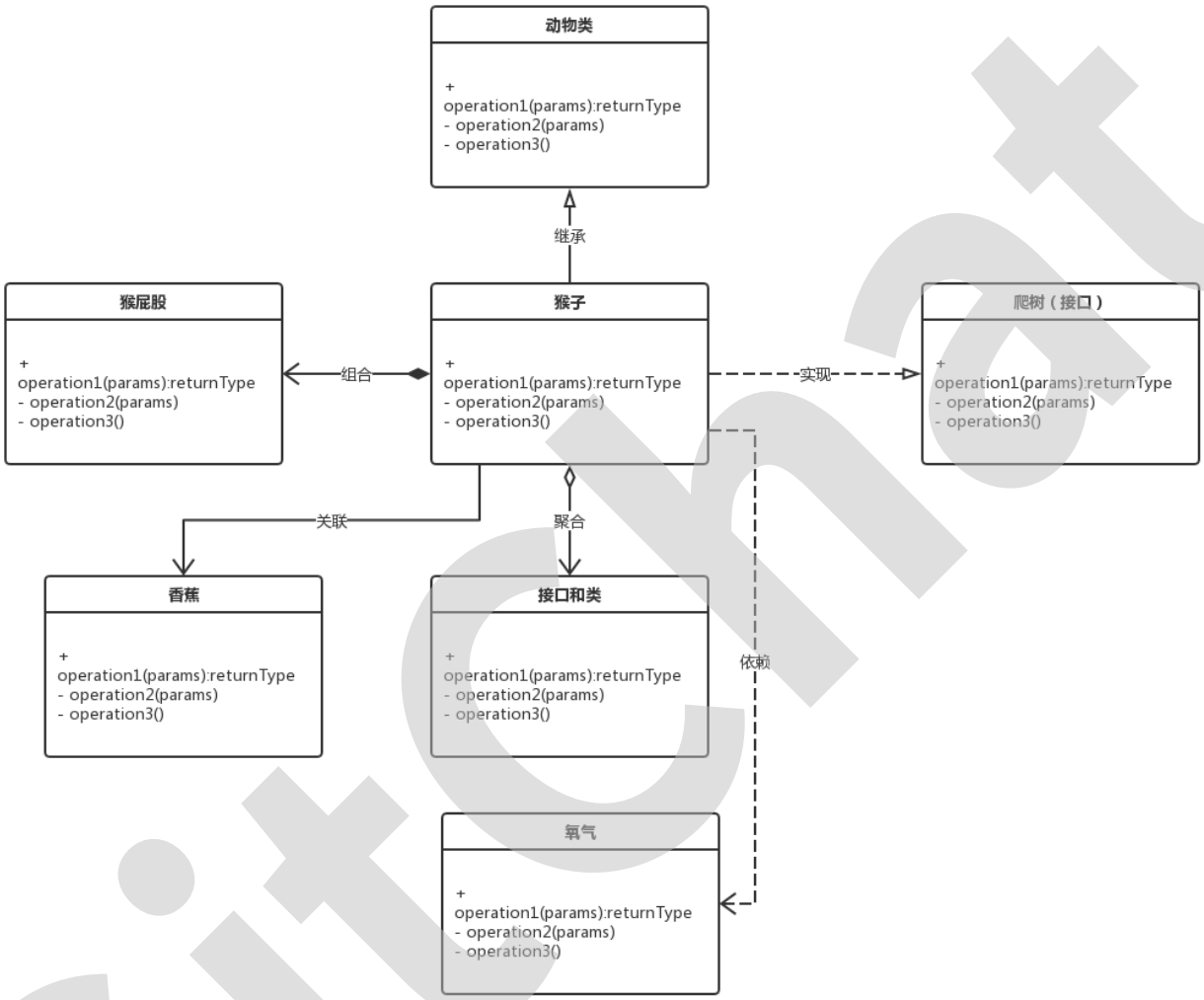
本课程将使用 Java 语言讲解设计模式，虽然设计模式与语言本身无关，但是本课程中有许多实际案例都是来自于知名的 Java 框架源码，如果没有 Java 基础，学习效果可能不佳。

除了要求 Java 基础之外，还需要了解 UML 图，如果不了解 UML，只需要知道以下几种 UML 关系即可：

- **泛化**：可以简单的理解为继承关系；
- **实现**：一般是接口和实现类之间的关系；
- **关联**：一种拥有关系，比如老师类中有学生列表，那么老师类和学生类就是拥有关系；
- **聚合**：整体与部分的关系，但是整体和部分是可以分离而独立存在的，如汽车类和轮胎类；

- **组合**：整体与部分的关系，但是二者不可分离，分离了就没有意义了，例如，公司类和部门类，没有公司就没有部门；
- **依赖**：一种使用关系，例如创建 A 类必须要有 B 类。

参考下图：



记不住也没关系，后续课程主要使用泛化和实现这两种，先记住这两种即可，如果有遇到看不懂的再回头来看一眼。