前端开发核心知识进阶：50 讲从夯实基础到突破瓶颈
来自 Lucas ... · 盐选专栏

查看详情 >

揭秘前端设计模式（1）

设计模式——我认为这是一个一言难尽的概念。维基百科对设计模式的定义为：

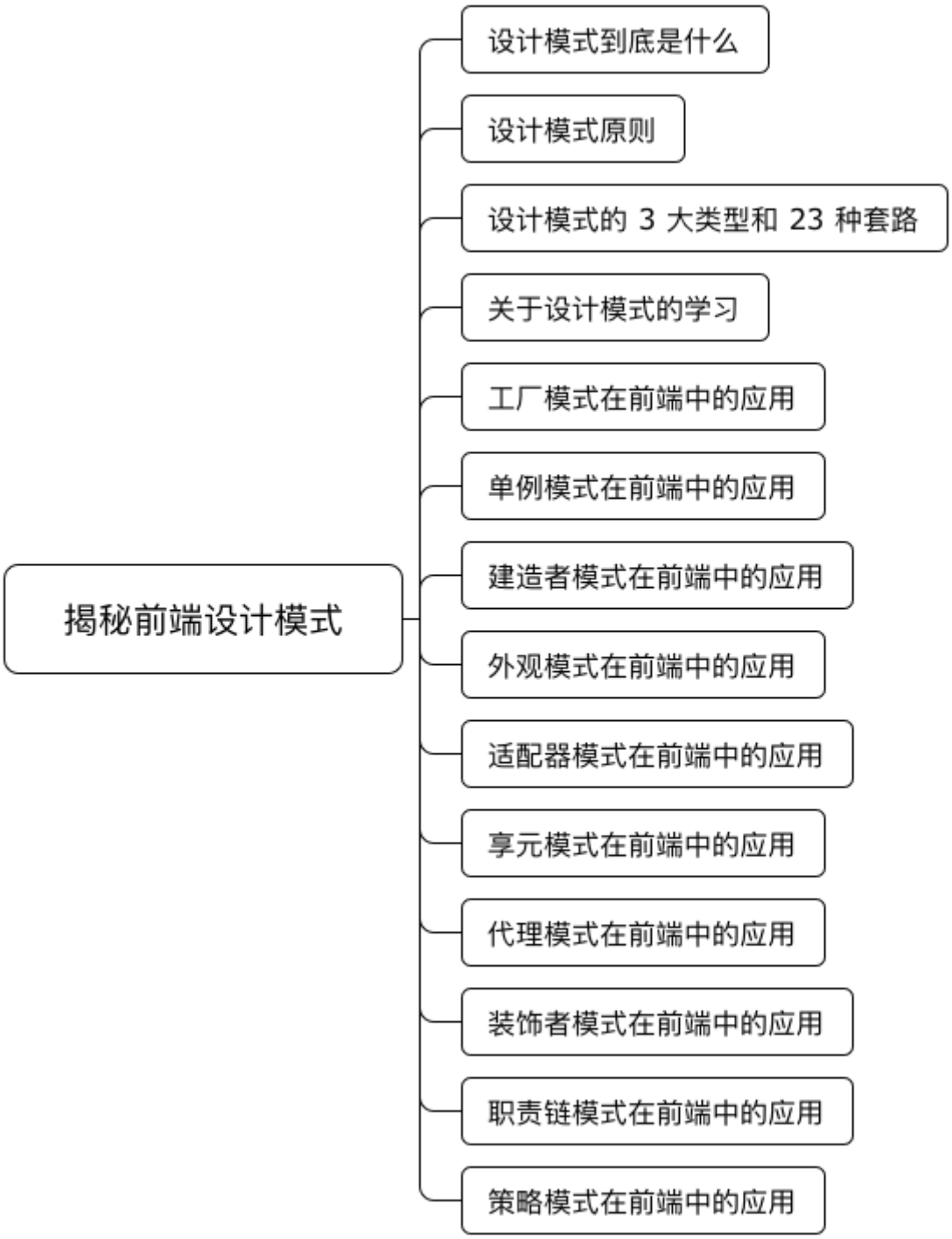
在软件工程中，设计模式（Design Pattern）是对软件设计中普遍存在（反复出现）的各种问题，所提出的解决方案。这个术语是由埃里希·伽玛（Erich Gamma）等人在 1990 年代从建筑设计领域引入到计算机科学的。设计模式并不是直接用来完成代码的编写，而是描述在各种不同情况下，要怎么解决问题的一种方案。

为什么「一言难尽」呢？首先从设计模式的概念可以看出：这是一套理论，干巴巴的描述其所有内容并没有太大意义。我们不会在面试中提出：「请你解释一下设计模式」、「你会多少种设计模式」这种问题。设计模式一般认为有 23 种，**这 23 种设计模式的本质是面向对象设计原则的实际运用，是对类的封装性、继承性和多态性，以及类的关联关系和组合关系的总结应用。**

那么对于 JavaScript 或者前端开发来说，设计模式似乎是一个有些遥远的概念。我们应该如何了解并学习设计模式呢?

我认为设计模式不能停留在理论上，而是应该结合到实际代码当中。因此打算通过两讲内容来介绍：本讲内容先介绍基本概念，分享一些经典的设计模式书籍以及相关经验，也许稍微有些「无趣」；第二讲内容将深入结合前端开发，挑选那些我们一直使用的、会用到的设计模式进行讲解。

相关知识点如下：



设计模式到底是什么

之前提到，设计模式是一种经验总结，它就是一套兵法，一共包含了 23 个套路。最终目的是为了更好的代码重用性、可读性、可靠性、可维护性。

在平常开发中，「也许你不知道，但是已经在使用设计模式了」。在之前课程内容的学习中，我们其实也有所提及，比如单例模式（细心的读者还能找到单例模式实现的课程出处吗？）、发布订阅模式、原型模式等。

如果到此，仍然不明白设计模式到底是指什么，别着急，请继续阅读以下内容。

设计模式原则

既然是一套理论，是一种约定和规范，那么设计模式也就有自己的模式原则。总体来说，其六大原则包括：

开闭原则

里氏替换原则

依赖反转原则

接口隔离原则

最小知道原则

合成复用原则

如图：

我们来逐一了解：

开闭原则（Open Close Principle）

理解开闭原则，就要了解开和闭。**这里的开是指对扩展开放，闭是说对修改关闭**。想想我们有一套实现、提供一个服务，这样的程序需要能够随时进行扩展、随时支持第三方的自定义配置，但是不能去修改已用的实现代码。

比如我们做了一个 UI 组件轮子，业务方在使用时显然不能够修改我们的代码，但是仍然可以进行扩展。再比如著名的 Draft.js 库，在实现一个编辑器时，提供了灵活的插件机制，实现了热插拔效果，使得整个程序的扩展性好，易于维护和升级。甚至 Redux 库、Koa 库等基本所有库都有开闭原则的体现。

对于面向对象类型的语言来说，想要严格遵守开闭原则，往往需要使用接口和抽象类，这个我们会在具体设计中再次提到。

里氏替换原则（Liskov Substitution Principle）

里氏代换原则就稍微有些抽象，但它是面向对象设计的基本原则之一。

里氏代换原则要求，任何基类可以发挥作用的地方，子类一定可以发挥作用。

这句话怎么理解呢？想想我们的继承实现，里氏替换原则就是继承复用的基础。只有当派生类可以随时替换掉其基类，同时功能不被破坏，基类的方法仍然能被使用，这才是真正的继承，继承才能真正地实现复用，当然，派生类也需要随时能够在基类的基础上增加新的行为。

事实上，里氏代换原则是对开闭原则的补充。

依赖反转原则（Dependence Inversion Principle）

该原则要求针对接口编程，依赖于抽象。更多理论内容我并不打算展开，后续在程序设计中会结合实例提及。

接口隔离原则（Interface Segregation Principle）

接口隔离的意思或者目的是减少耦合的出现。在大型软件架构中，使用多个相互隔离的接口，一定比使用单个大而全的接口要好。

最少知道原则，又称迪米特法则（Demeter Principle）

最少知道顾名思义，是指：一个系统的功能模块应该最大限度地不知晓其他模块的出现，减少感知，模块应相对独立。

合成复用原则（Composite Reuse Principle）

合成复用原则是指：尽量使用合成 / 聚合的方式，而不是使用继承。这是很有意思的一点，我们在之前的课程中提到过：基于原型的继承在很多程度上「优于」基于类的继承，原因就在于基于原型的继承模式体现了可组合性，能够规避「大猩猩和香蕉」等问题的出现。组合是非常优秀的编程思想，这一点在函数式编程范畴中得到了最大程度的印证。

设计模式的三大类型和二十三种套路

设计模式并没有什么困难的，大体上所有的设计模式可以归结为三大类：

创建型

结构型

行为型

如图：

对于 Java 来说，它还包括了 J2EE 类型设计模式。

我们分别来看：

创建型（Creational Patterns）

创建型的五种设计模式提供了更加灵活的对象创建方式，同时可以隐藏创建的具体逻辑。与直接使用 new 运算符实例化对象相比，这些模式具有更强的灵活性以及可定制性。

结构型（Structural Patterns）

结构型的七种设计模式关注类和对象的组合，结合继承的概念，这些设计模式能使得对象具有更加灵活的功能设定。

行为型（Behavioral Patterns）

行为型的十一种设计模式聚焦于对象和类之间的通信，这是构建大型程序架构必不可少的环节。

关于设计模式的学习

设计模式使代码编写真正工程化，我们说设计模式是软件工程的基石脉络，如同大厦的结构一样。其实我认为没有必要刻意地去学习设计模式，因为有关设计模式的思想一定是在实际工程开发中慢慢体会总结的。但是这需要开发者做到「非常有心」，才能够自己去慢慢积累，为了能够培养这种「用心」，读者去专门了解设计模式似乎也是一种捷径和方式。两节课程的设置足以帮助大家培养设计模式思想，同时我再分享一些关于设计模式的经典资料：

design-patterns-for-humans：这是一本非常著名的设计模式书 pdf

design-patterns-for-humans-cn：上本书的中文版 pdf

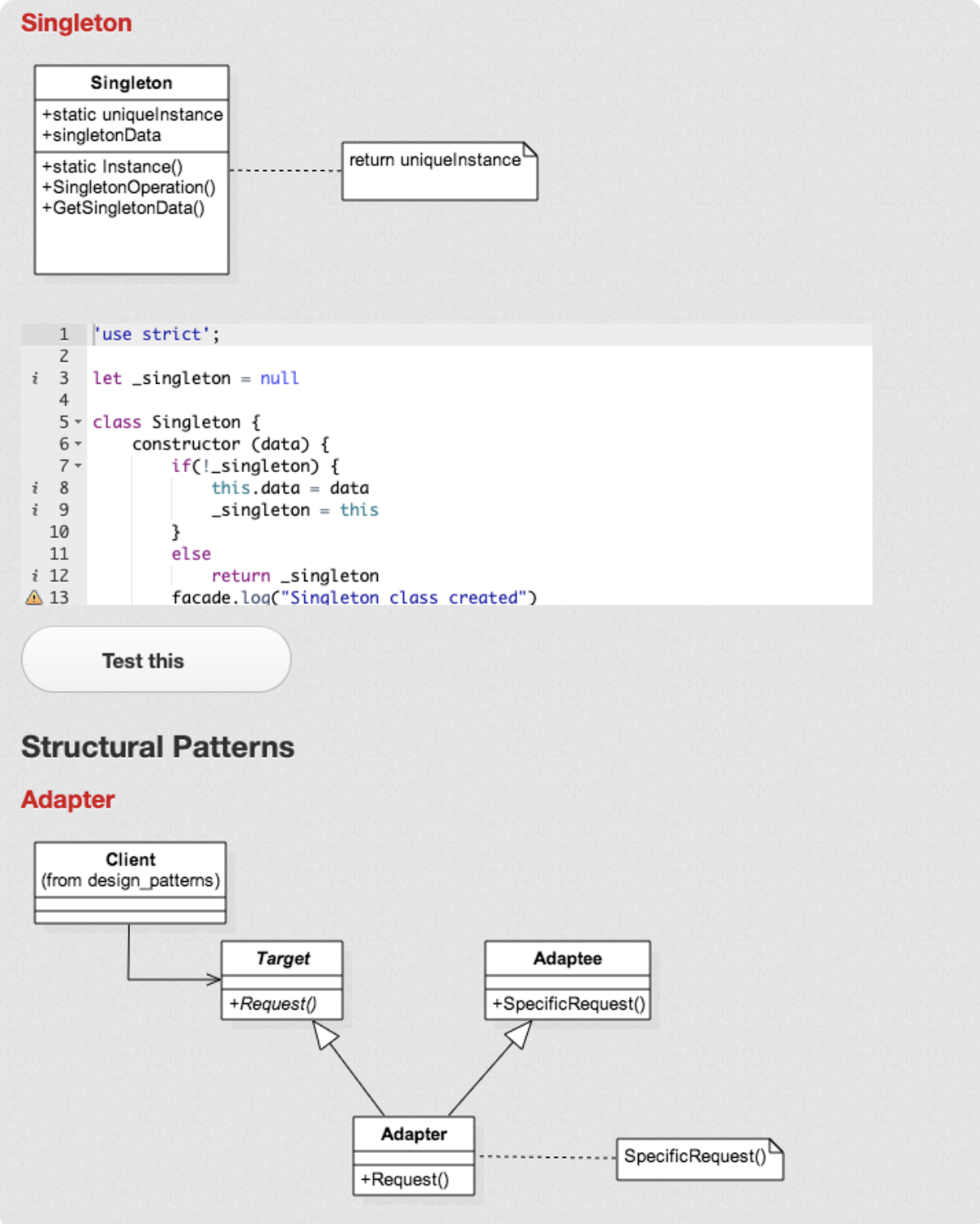
Learning JavaScript Design Patterns：addyosmani 大神的书 pdf

图说设计模式

其中强烈推荐《Learning JavaScript Design Patterns》，这本书在网上开源免费，其中的内容示例都是用 JavaScript 编写的，而且在代码实例编写当中剖析了很多 jQuery 等经典「轮子」的设计。

同时 GitHub 上也有一个不错的 repo：JsPattern-ES6，使用 ES6 重写了《JavaScript 模式》一书中的样例。

还有一个「神器」是：es6-design-patterns，如截图：



这个网站通过 UML 图解释设计模式，同时配以可以运行的代码示例，非常方便对每一种设计模式进行学习。

点击查看下一节

揭秘前端设计模式（2）