本节为统领全书的纲要性章节，是后续多个章节的伏笔，因此不建议大家跳读。另一方面，也**不建议大家死磕** —— 有的东西你这会儿没看懂也没关系，正常。接着往下看实战就完了，跟着做跟着跑，不知不觉你就会有“啊原来前面那个xx说的是这个xx啊”的奇妙感觉。

设计模式之道

设计模式，究竟有着怎样的力量?

> 每一个模式描述了一个在我们周围不断重复发生的问题，以及该问题的解决方案的核心。这样，你就能一次又一次地使用该方案而不必做重复劳动。 —— Christopher Alexander

设计模式是“拿来主义”在软件领域的贯彻实践。和很多人的主观臆断相反，设计模式不是一堆空空如也、晦涩鸡肋的理论，它是一套现成的工具 —— 就好像你想要做饭的时候，会拿起厨具直接烹饪，而不会自己去铸一口锅、磨一把菜刀一样。

用做数学题来打比方，可能大家会更能体会这种概念 —— 我们解题目的时候，往往会用到很多公式/现成的解题方法。比如已知直角三角形两边长，求另一边，我们会直接用勾股定理（我想应该没有人会每求一次边长都自己推一遍勾股定理才用吧）。

识别题目特征 —— catch题目想要考查的知识点 —— 快速在脑海中映射出它对应的解决方法，这个过程在我们学生时代几乎是一个本能的、条件反射一样的脑回路机制。在学习设计模式时，如果各位可以回忆起这种“从映射到默写”的思维方式，相信这个学习过程会是轻松的、自然的。

SOLID设计原则

> "SOLID" 是由罗伯特·C·马丁在 21 世纪早期引入的记忆术首字母缩略字，指代了面向对象编程和面向对象设计的五个基本原则。

设计原则是设计模式的指导理论，它可以帮助我们规避不良的软件设计。SOLID 指代的五个基本原则分别是：

- 单一功能原则（Single Responsibility Principle）
- 开放封闭原则（Opened Closed Principle）
- 里式替换原则（Liskov Substitution Principle）
- 接口隔离原则（Interface Segregation Principle）
- 依赖反转原则（Dependency Inversion Principle）

糟糕，又出现了看似高大上的东西，而且是五个！

别怕，这五个原则，都不难，而且并不是每一个都要求大家掌握，因为在 JavaScript 设计模式中，主要用到的设计模式基本都围绕“单一功能”和“开放封闭”这两个原则来展开。

在本节，我们不会对设计原则作任何进一步的介绍——在没有实际操作的情况下，干讲理论没有任何意义，反而会挫伤初学者的积极性。具体的原则、理论，我们都会放在后续的实战小节里结合实例一起讲解。

设计模式的核心思想——封装变化

设计模式出现的背景，是软件设计的复杂度日益飙升。软件设计越来越复杂的“罪魁祸首”，就是**变化**。

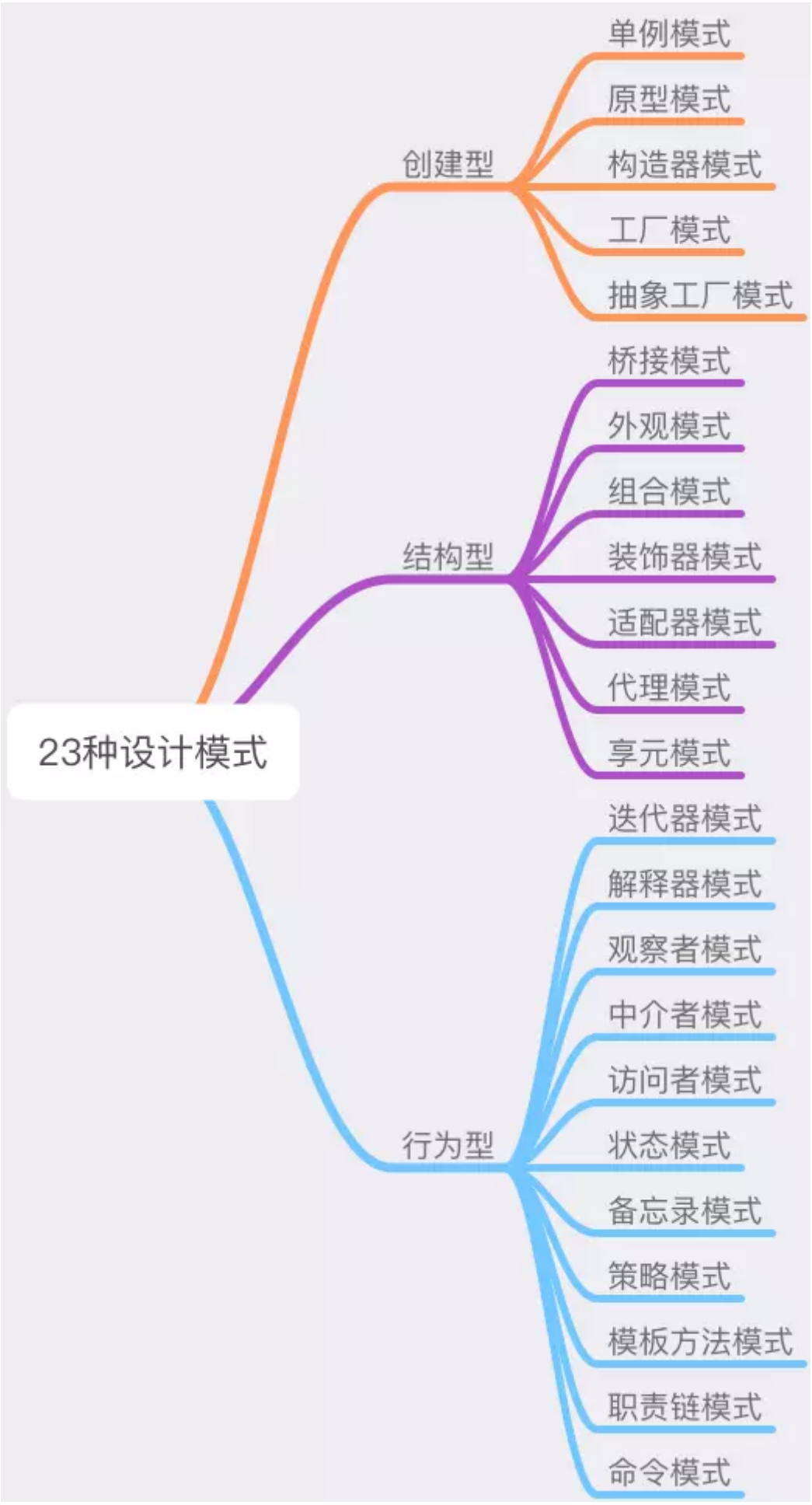
这一点相信大家不难理解——如果说我们写一个业务，这个业务是一潭死水，初始版本是 1.0，100 年后还是 1.0，不接受任何迭代和优化，那么这个业务几乎可以随便写。反正只要实现功能就行了，完全不需要考虑可维护性、可扩展性。

但在实际开发中，不发生变化的代码可以说是不存在的。我们能做的只有将这个变化造成的影响**最小化**——**将变与不变分离，确保变化的部分灵活、不变的部分稳定。**

这个过程，就叫“封装变化”；这样的代码，就是我们所谓的“健壮”的代码，它可以经得起变化的考验。而设计模式出现的意义，就是帮我们写出这样的代码。

设计模式的“术”

所谓“术”，其实就是指二十年前 GOF 提出的最经典的23种设计模式。二十年前，四位程序员前辈（Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides）通过编写《设计模式：可复用面向对象软件的基础》这本书，阐述了设计模式领域的开创性成果。在这本书中，将23种设计模式按照“创建型”、“行为型”和“结构型”进行划分：


23种设计模式
创建型
单例模式
原型模式
构造器模式
工厂模式
抽象工厂模式
结构型
桥接模式
外观模式
组合模式
装饰器模式
适配器模式
代理模式
享元模式
行为型
迭代器模式
解释器模式
观察者模式
中介者模式
访问者模式
状态模式
备忘录模式
策略模式
模板方法模式
职责链模式
命令模式

前面我们说过，设计模式的核心思想，就是“封装变化”。确实如此，无论是创建型、结构型还是行为型，这些具体的设计模式都是在用自己的方式去封装不同类型的变化 —— 创建型模式封装了创建对象过程中的变化，比如下节的工厂模式，它做的事情就是将创建对象的过程抽离；结构型模式封装的是对象之间组合方式的变化，目的在于灵活地表达对象间的配合与依赖关系；而行为型模式则将是对象千变万化的行为进行抽离，确保我们能够更安全、更方便地对行为进行更改。

封装变化，封装的正是软件中那些不稳定的要素，它是一种防患于未然的行为 —— 提前抽离了变化，就为后续的拓展提供了无限的可能性，如此，我们才能做到在变化到来的时候从容不迫。

从 Java/C++ 到 JavaScript 的迁移

设计模式迁移到 JavaScript，不仅仅是从一类语言到另一类语言这么简单。强类型语言不仅和 JavaScript 之间存在着基本语法的差异，还存在着应用场景的差异。设计模式的“前端化”，正是我们后续十余个章节要做的事情。在这个过程中，场景是基础，代码是辅助，逻辑是主角。

OK，说了这么多，想必大家现在心里都对设计模式有了一套自己的全局观。下面我们就正式进入实战的环节，将目标设计模式各个击破~

（阅读过程中有任何想法或疑问，或者单纯希望和笔者交个朋友啥的，欢迎大家添加我的微信xyalinode与我交流哈~）