# 设计模式笔记

阅前总结：

模式（解决软件设计中常见问题的模板）如何解决问题的模板，设计模式三点好处：

* 模式是行之有效的解决方法：他们提供固定的解决方法来解决在软件开发中出现的问题，这些都是久经考验的反应了开发者的经验和见解的使用模式来定义的技术。
* 模式可以很容易地重用：一个模式通常反映了一个可以适应自己需要的开箱即用的解决方案。这个特性让它们很健壮。
* 模式善于表达：当我们看到一个提供某种解决方案的模式时，一般有一组结构和词汇可以非常优雅地帮助表达相当大的解决方案。

模式不是一个确切的解决方案，仅仅威我们提供一个解决方案而非能解决所有设计问题，优势：

* 模式的重用可以帮助防止在应用程序开发过程中出现的一些可能导致重大问题的小问题。这意味着当代码是建立在行之有效的模式上时，我们可以花更少的时间去关心我们的代码结构，从而能花更多的时间关注我们的解决方案的整体质量。这是因为模式可以鼓励我们在更好的结构化和有组织的方式下编码，这将避免在未来由于清洁的目的而去重构它。
* 模式可以提供一个不需要绑定到一个特定问题的书面的概括性的解决方案。这个广义的方法意味着不用管我们正在处理的应用程序 (许多情况下的编程语言) 设计模式的应用可以提高我们的代码的结构。
* 某些模式可以通过避免重复来减小我们代码的文件大小。通过鼓励开发者更仔细地看待他们的解决方案来减少重复的地方，如通过将类似的执行流程作为一个一般性的函数来减少函数的数量，这样我们就可以减小代码库的总体大小，这也成为使代码更DRY。
* 模式增加了开发者的词汇，这使得交流更快速。
* 经常使用的模式可通过收集其他使用这些模式的开发人员贡献给设计模式社区的经验来改进。在某些情况下，这将导致全新模式的创建，同时也可以提供改进的指导大家如何使用特定的模式才是最好的。这可以确保基于模式的解决方案继续变得比特别的解决方案更健壮。

模式原型和三条规则：

* 解决一类特定的问题：模式不能假设仅仅关注原理或者策略。它们需要关注解决方案。这是好的模式最重要的因素之一。
* 问题的解决方案不是表面上的：我们发现解决问题的技术常常首先试图源自于某个众所周知的原理。最佳的设计模式通常间接地提供问题的解决方案-认为模式是与设计相关的最具有挑战性的问题必然的解决方法。
* 所描述的想法一定得到了证明：设计模式需要提供所描述的它们运行的证据，如果没有这些证据，就不会认真的考虑这个设计。如果模式事实上是高度理论性的话，那么只有冒险者才可能试着用它。
* 它必须说明与代码之间关系：在某些情况下，模式似乎说明了一种类型的模块。虽然实现的可能就是这个方法，但是官方的模式说明一定要更深入的描述系统结构和机制，以解释它与代码之间的关系。

三条准则：

* 适用性-模式怎样才能被认为是成功的。
* 有用性-为什么认为这个模式是成功的？
* 可用性 -因为设计得到广泛的应用，所以认为这个设计就是模式吗？如果是这样的话，那么需要说明。重新审核或者定义模式的时候，牢记以上规则非常重要。

代码的评判标准

1. 可维护性
2. 可读性

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

1. 可扩展性
2. 灵活性（可以适应大多数场景）
3. 简洁性（代码简单、逻辑清晰）
4. 可复用性
5. 可测试性

## 设计模式分类

### 创建型

常用：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：原型模式

### 结构型

常用：代理模式、桥接模式、装饰者模式、适配器模式

不常用：门面模式、组合模式、享元模式

### 行为型

常用：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：访问者模式、备忘录模式、命令模式、解释器模式、中介模式



## 设计模式的结构

* **上下文环境**
* 在这种环境下产生的系统的**力量**
* 一类**配置，**考虑到允许这种力量在自己的上下文环境中解决这一点，现在让我们对一种设计模式的组件元素，一探究竟。一种设计模式应该具有：
* 模式名称和相应的描述
* **上下文概述**-在设计模式中的上下文对响应用户需求是很有效的。
* **问题声明**-一类问题的声明，能让我们理解模式的意图。
* **解决方案**-在可理解的列表和看法上，对用户的问题如何被解决的一种描述。
* **设计**-模式设计，特别是与之交互的用户行为的描述。
* **实现**-对模式如何被实现的一种指引
* **例证**-在模式中的一种类的虚拟化表示
* **例子**-模式实现的一种最下的形式
* **共同条件**-可能会有其他的什么模式会被用到，以对被描述的模式进行支持？
* **关系**-与该模式相似的模式有哪些？是最相似的吗？
* **已知的使用**-模式没有被正常使用？如果是，在哪，怎样做到的？
* **讨论**-有激动人心的获利模式想法的团队或者是作者

## JavaScript 反模式

反模式是一个值得为此专门编写编写总结文档的糟糕设计。Javascript的反模式例子如下:

* 在全局上面文中定义大量污染全局命令空间的变量
* 在调用setTimeout和setInterval时传递字符串(会用eval来执行)而不是函数。
* 修改Object的原型 (这是最糟糕的反模式)
* 使用内联Javascript
* 在本应使用document.createElement的地方使用document.write。document.write被错误的用了相当多的年头，它有相当多的缺点，包括如果在页面加载后执行它可能会覆盖我们的页面。再有它不能工作在XHTML下，这也是另外一个我们使用像document.createElement这种对DOM友好方法的原因。

知道反模式对成功来说很关键。一旦我们能识别这些反模式，我们就能够重构我们的代码使项目的整体质量立马提升。