《Software Architecture in Practice读书笔记》

我选择的是架构设计这一部分。

架构设计是人们对一个结构内的元素及元素间关系的一种主观映射的产物。架构设计是一系列相关的抽象模式，用于指导大型软件系统各个方面的设计。

属性驱动的设计用于设计构架以满足质量需求和功能需求。属性驱动的设计把一组质量属性场景作为输入，并且使用对质量属性实现和构架之间的关系的了解，对构架进行设计。该方法将分解过程建立在软件必须满足的质量属性之上。并且它是一个递归地分解过程，在每个阶段都选择战术和构架模式来满足一组质量属性场景，然后对功能进行分配，以实例化由该模式所提供的模块类型。因为属性驱动的设计是设计过程中构架的第一个连接点，所以一定是粗粒度的，视图的所有细节并不都是通过ADD得到的。

**进行属性驱动的设计有几个关键步骤：**

1. 选择要分解的模块；
2. 根据这些步骤对模块进行求精；
3. 对需要进一步分解的每个模块重复上述步骤：
4. 选择要分解的模块
5. 选择构架驱动因素
6. 选择构架模式
7. 实例化模块并使用多个视图分配功能
8. 定义子模块的接口---模块的接口展示了所提供的服务和所要求的属性
9. 验证并求精用例和质量属性场景。

所有步骤结束后，模块被分解了为子模块，并且每个子模块都有一个责任的集合，并且拥有了一组用例、接口、质量属性场景以及限制集合。对构架进行了充分设计，并且团队为开始构建构架做好准备后，就可以构造骨架系统。在这个过程中，最好首先对骨架子系统中的接口进行协商，然后再运用骨架系统来提高开发效率。很重要的一点还有构架设计必须按需求分析进行，但是并不需要在需求分析完成后再开始构架设计。

**架构的设计有七大原则：**

1. 开闭原则

开闭原则（Open-Closed Principle, OCP）是指一个软件实体如类、模块和函数应该对扩展开放，对修改关闭。所谓的开闭，也正是对扩展和修改两个行为的一个原则。强调的是用抽象构建框架，用实现扩展细节。可以提高软件系统的可复用性及可维护性。

1. 依赖倒置原则

依赖倒置原则（Dependence Inversion Principle,DIP）是指设计代码结构时，高层模块不应该依赖底层模块，二者都应该依赖其抽象。抽象不应该依赖细节；细节应该依赖抽象。通过依赖倒置，可以减少类与类之间的耦合性，提高系统的稳定性，提高代码的可读性和可维护性，并能够降低修改程序所造成的风险。

3、单一职责原则

单一职责（Simple Responsibility Pinciple，SRP）是指不要存在多于一个导致类变更的原因。假设我们有一个 Class 负责两个职责，一旦发生需求变更，修改其中一个职责的逻辑代码，有可能会导致另一个职责的功能发生故障。这样一来，这个 Class 存在两个导致类变更的原因。如何解决这个问题呢？我们就要给两个职责分别用两个 Class 来实现，进行解耦。后期需求变更维护互不影响。这样的设计，可以降低类的复杂度，提高类的可读性，提高系 统的可维护性，降低变更引起的风险。

1. 接口隔离原则

接口隔离原则（Interface Segregation Principle, ISP）是指用多个专门的接口，而不使用单一的总接口，客户端不应该依赖它不需要的接口。这个原则指导我们在设计接口时应当注意一下几点：

①一个类对一类的依赖应该建立在最小的接口之上。

②建立单一接口，不要建立庞大臃肿的接口。

③尽量细化接口，接口中的方法尽量少（不是越少越好，一定要适度）。

接口隔离原则符合我们常说的高内聚低耦合的设计思想，从而使得类具有很好的可读性、可扩展性和可维护性。

1. 迪米特法则

迪米特原则（Law of Demeter LoD）是指一个对象应该对其他对象保持最少的了解，又叫最少知道原则（Least Knowledge Principle,LKP），尽量降低类与类之间的耦合。

1. 里氏替换原则

里氏替换原则（Liskov Substitution Principle,LSP）是指如果对每一个类型为 T1 的对象 o1,都有类型为T2的对象o2,使得以T1定义的所有程序P在所有的对象o1都替换成o2时，程序P的行为没有发生变化，那么类型T2是类型T1的子类型。

7、合成复用原则

合成复用原则（Composite/Aggregate Reuse Principle,CARP）是指尽量使用对象组合(has-a)/聚合(contanis-a)，而不是继承关系达到软件复用的目的。可以使系统更加灵活，降低类与类之间的耦合度，一个类的变化对其他类造成的影响相对较少。

架构的基本需求主要是在满足功能属性的前提下，关注软件质量属性，架构设计则是为满足架构需求（质量属性）寻找适当的“战术”。软件属性包括功能属性和质量属性，但是，软件架构（及软件架构设计师）重点关注的是质量属性。因为，在大量的可能结构中，可以使用不同的结构来实现同样的功能性，即功能性在很大程度上是独立于结构的，架构设计师面临着决策（对结构的选择），而功能性所关心的是它如何与其他质量属性进行交互，以及它如何限制其他质量属性。

总而言之，架构设计过程中应当始终考虑到让软件系统易见、易学、易用。以技术为基础，以用户的体验为中心，提升软件易用性和界面友好性。