软件工程——

软件架构学习笔记

亓龙雨 201805130163 计科18.2

## 架构的概念

在软件行业，架构的概念一直没有一个完整、统一的定义，软件架构的概念分为主要量大派别：组成派和决策派。组成派认为软件架构是：软件系统的架构将系统描述为计算组件及组件之间的交互，“组件”是广泛意义上的元素，并不是指具体的技术组件。组成派的定义关注架构实践中的客体——软件，以及软件本身为描述对象；另外分析了软件的组成，即软件由承担不同计算任务的组成组件，这些组件通过相互交互完成更高层次的计算。而决策派认为，软件架构保护了软件设计过程中一些列问题的重要决策，软件架构并不仅仅关注软件本身的结构和行为，还注重其他特性：使用、功能、性能、弹性、重用、可理解性、经济、和计算限制、权衡、以及美学等。决策派的定义更关注架构实践中的主体——人，以人的决策为描述对象，并且归纳了架构决策的类型，指出架构决策不仅包括关于软件系统的组织、元素、子系统和架构风格等几类决策，还包括关于众多非功能需求的决策。

说的更通俗一点，组成派关注架构的结果，而决策派关注架构的过程。而笔者认为，架构的是一个演进的过程，没有一成不变的架构，也没有一蹴而就的架构，架构需要根据业务的发展而进行相应的调整，所以我们应该更加关注架构演进的过程和原因，以及在当时背景下的做出的权衡和妥协。没有完美的架构，架构也没有一个标准的解决方案，因为面对不同的业务需求，面对不同的客观因素，有不同的架构设计方案，也就是有不同的设计决策。组成派和决策派的架构概念并不矛盾，它们只不过是所站的角度不同罢了：在具体的软件架构实际中，总是同时体现这两“派”的架构概念。

架构决策是分层次依次展开的，决策制定的顺序往往是先制定技术无关的决策，后制定技术相关的决策，后者在前者的指导下进行。架构师一定要考虑更多的实际开发中要涉及到的技术问题，从而不断细化架构方案，这样才能为开发人员提供更多的指导和限制，也才能真正降低后续开发中的重大技术风险。所以，架构师的架构方案不是高来高去的，而是需要能够落地的，并且在落地的过程中遇到的技术细节问题，架构师是需要参与解决的。这样架构师才能与开发人员无缝对接，架构方案才能指导开发人员进行开发。

## 子系统、框架和架构

好的架构必须使每个关注点相互分离。把变化点错落有致地封装到软件系统的不同部分，为此，必须进行关注点分离。通过关注点的分离来达到“系统中的一部分发生了变化，不会影响其他部分”的目标。可以通过如下的一下手段：

1、 通过职责划分来分离关注点。面向对象设计的关键所在，就是职责的识别和分配。无论是对象、模块，还是子系统，它们所承担的职责都应该具有高内聚性，否则对象之间的松耦合性就失去了基础，成为空谈。所以我们一直在提“高内聚、低耦合”，高内聚才是低耦合的基础。而我们经常使用的设计模式和架构模式为特定上下文中重复出现的问题提供了通用的职责划分方案，也是问题解决方案。

2、 利用软件系统各部分的通用性的不同进行关注点分离。不同的通用程度意味着变化的可能性不同。这一点与面向对象设计里面的重用发布等价原则异曲同工。

3、 先考虑大粒度的子系统，而暂时忽略子系统是如何通过更小粒度的模块和类组成的。避免陷入过多的细节当中，所谓“忘却是一种能力”，就是指架构师必须有在更高层面思考的能力。

根据职责分离关注点、根据通用性分离关注点、根据不同粒度级别分离关注点是3种位于不同“维度”的思维方式，我们可以综合运用这些手段。

无论是架构模式还是设计模式，重点关注的都是如何提供一个“协作模型”，这个协作模型通过明确协作中不同的角色锁担任的职责，达到“为特定上下文的问题提供解决方案”的目的。

框架技术有助于把通用关注点和专用关注点分离开来，结果是带来了更好的易修改性和可重用性。

作为架构师，必须思考“软件单元是如何组成粒度更大的整体的”这一问题。类、模块、子系统、系统、集成系统，都是软件单元的具体形态，只不过粒度不同罢了。软件系统越复杂，不同粒度的分解层次就越多。这里所谓的系统，是指由多个元素组成的逻辑实体，它完成一组特定的目标或负担一定的职责。系统可以仅包含软件，也可以仅包含硬件，或者两者都包含。子系统是特殊的系统，只不过在特定的上下文中，这个系统作为更大系统的一部分出现。系统需要架构设计，而子系统如果足够复杂，则也需要架构设计。而不同类型的软件系统需要不同的软件架构设计，一个系统的不同子系统也应当有不同的软件架构。

当然，一个系统的粒度是具有相对性的，同一个软件单元，在不同场景下，我们会以不同的粒度看待它。所有的系统都有子系统，而所有系统都是更大系统的子系统。实践不同场景使得软件组成单元具有递归性。懂得了细节是相对而言的这一点，软件架构师才能够游刃有余地根据情况忽略应该被忽略的细节，抓住设计大局。

框架的概念，框架是一种半成品。是可以通过某种回调机制进行扩展的软件系统或子系统的半成品。框架从某种程度上来说，是为了软件重用。而软件重用本身又有一对内在的矛盾，即重用几率和重用价值，重用几率大小和重用带来的价值大小之间的矛盾。软件单元的粒度越大，可重用的几率就越小，但其重用所带来的价值就越大，相反，软件单元的粒度越小，可重用几率就越大，但其重用所带来的价值就越小。框架的智慧就在于此：为了追求重用所带来的价值最大化，将容易变化的的部分封装成扩展点，并辅以回调机制将它们纳入框架的控制范围之内。

对软件架构的误解，其中一个最为普遍的就是，将架构（Acrchitecture）和框架（Framework）混为一谈。框架是软件，而架构不是软件。框架是一种特殊的软件，它并不能提供完整的解决方案（准确说，是不提供业务应用的完整解决方案，而框架本身也是为解决某一类问题而出现的），而是为构建解决方案（这里的解决方案即是业务应用的解决方案）提供良好的基础，所以说，框架是半成品。框架中的服务可以被最终应用直接调用，而框架中的扩展点是供应用开发人员定制的“可变化点”。而软件架构不是软件，而是关于软件如何设计的重要决策。软件架构决策涉及到如何将软件系统分解成不同的部分、各部分之间的静态结构关系和动态交互关系等。这些架构决策将体现在最终开发出的软件系统中。引入软件框架之后，整个开发过程变成了分两步走，而架构决策往往会体现在框架之中。软件架构是比具体代码高一个抽象层次的概念。架构势必被代码所体现和遵循，但任何一段具体的代码都代表不了架构（这里说的有点绝对，某一段具体的代码，代表不了整个架构，有时却可以代表某一个架构决策）。框架技术和架构技术的出现，都是为了解决软件系统日益复杂所带来的困难而采取“分而治之”的思想。先大局后局部，就出现了架构；先通用后专用，就出现了框架。架构是问题的抽象解决方案，它关注大局而忽略细节；框架是通用半成品，还必须根据具体需求进一步定制开发才能变成应用系统。当然，框架也有架构，而且很重要。框架和架构既有区别又有联系，前者是复合组件特例，后者是复合组件的大局设计。软件架构需要落地，需要了解细节，但也没有必要将所有设计决策事无巨细地落实，而是重点关注“重要决策”。软件架构设计的决策范围，应该着重放在“影响全局”的设计上，而不是所关注所有设计细节

对面向对象开发而言，类库和框架有很多共同之处，但它们确确实实又是不同的。类库是类的集合，这些类之间可能是相互独立的。与类库相比，框架和类库有着相似的形式，即框架也往往是类的集合，但不同之处在于，框架中的各个类不是孤立的，而框架中的业务逻辑代码是将不同的类“连”在一起，在它们之间建立协作关系。框架通过封装处理流程的控制逻辑，使它对开发者透明，来简化开发工作。这种封装也是框架和类库的区别之一。

框架可以分为应用框架、中间件框架和基础设施框架三大类，形成了“应用软件——中间件——基础设施”的宏观格局。框架也可以分为技术框架和业务框架。技术框架又称为水平框架，业务框架有称为垂直框架。框架的整个开发过程，包括四个主要阶段，即分析、设计、实现和稳定阶段。和应用开发一样，框架开发首先要确定框架的范围和目标。对特定领域框架层和跨领域框架层都要识别出通用点和扩展点，其次，为框架设计架构，将它用作实现阶段的蓝图。在设计阶段，也可以创建应用程序框架原型，然后在其上构建一个样本应用，并洞察框架设计中潜在的可改进之处。框架开发的稳定阶段还应提供相应的框架文档。面向对象技术的发展极大提够了框架的能力，并推动了框架技术被普遍接受。面向对象框架的组成部分包括具体类、抽象类和接口。抽象方法是面向对象支持“多态”的关键，面向对象框架借助抽象方法实现逆向控制。许多面向对象框架在利用抽象方法支持扩展的同时，还会借助“配置驱动”来降低使用框架的难度。框架也需要架构设计，但反过来，可以通过架构框架化，达到“架构重用”的目的。

## 软件架构的作用

软件架构对后期的软件维护，乃至改动力度比较大的软件升级都起着重要作用。因为软件架构给出了软件系统的一个全局的视角。对软件系统不断的修改会使系统架构慢慢变得混乱，因为业务的发展，是会慢慢腐蚀现有的系统架构的，所以架构需要根据业务的发展进行演进。

软件架构设计为什么这么难？因为它是跨越现实世界与计算机世界之间鸿沟的一座桥。从面向业务的需求，到最终的面向技术的软件系统，要跨越很大的鸿沟。软件架构设计就是要完成从面向业务到面向技术的转换，在鸿沟上驾起一座桥梁。软件架构包含了结构、协作和技术等方面的重要决策，为系统化的开发活动建立了基础。

架构的作用包括如下几个方面：

1、 上承业务目标：担负着完成业务目标而进行大局规划的职责。

2、 下接技术决策：软件架构师将面向业务的需求转化为面向技术的软件架构设计方案，为后面的技术开发工作提供了切实的指导和限制。

3、 控制复杂性：先进行架构设计，后进行详细设计和编码实现，运用了“基于问题深度分而治之”的理念，利用控制复杂性。

4、 组织开发：软件架构为开展系统化的团队开发奠定了基础，它规定了软件系统的各元素如何彼此相关的设计决策，从而可以把不同模块分配给不同小组分头并发进行，而软件架构设计方案在这些小组中间扮演了“桥梁”和“合作契约”的作用。

5、 利用迭代开发和增量交互

6、 提高质量：清晰的软件架构将各个模块的职责划分得有条不絮，每个模块都有清晰的接口，这相当于间接降低了开发难度。

软件架构会“磨损”——随着对软件系统不断地修改，软件架构将变得混乱。

所谓软件架构重构，是指对软件架构进行比较大的修改和调整，使它适应新需求及开发和维护的需要。软件架构重构属于“再工程”的一种情况，一般会经历逆向工程、重新规划、正向工程三个步骤。只有充分理解原有架构的设计思路，才能评估它在现在需求情况的“利”和“弊”，把合理的设计保留下来，把不妥的设计修改掉。