架构设计-核心思维

介绍

架构的本质是管理复杂性，抽象、分层、分治和演化思维是我们工程师/架构师应对和管理复杂性的四种最基本武器。

最近团队来了一些新人，有些有一定工作经验，是以高级工程师/架构师身份进来的，但我发现他们大部分人思维偏应用和细节，抽象能力弱。所以作为团队技术培训的一部分，我整理了这篇文章，希望对他们树立正确的架构设计思维有帮助。我认为，对思维习惯和思考能力的培养，其重要性远远大于对实际技术工具的掌握。

由于文章内容较长，所以我把它分成两篇小文章，在第一篇《优秀架构师必须掌握的架构思维》中，我会先介绍抽象、分层、分治和演化这四种应对复杂性的基本思维。在第二篇《四个架构设计案例及其思维方式》中，我会通过四个案例，讲解如何综合运用这些思维，分别对小型系统，中型系统，基础架构，甚至是组织技术体系进行架构和设计。

#一、抽象思维

如果要问软件研发/系统架构中最重要的能力是什么，我会毫不犹豫回答是抽象能力。抽象(abstraction)这个词大家经常听到，但是真正理解和能讲清楚什么是抽象的人少之又少。抽象其实是这样定义的：

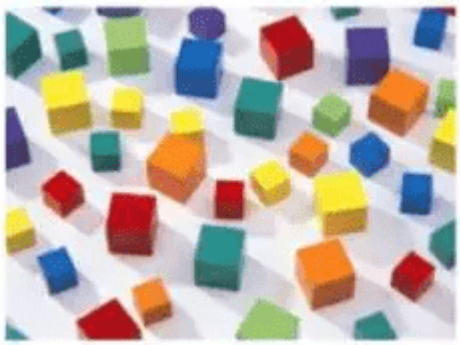
对某种事物进行简化表示或描述的过程，抽象让我们关注要素，隐藏额外细节。

举一个例子，见下图：

你看到什么？你看到的是一扇门，对不对？你看到的不是木头，也不是碳原子，这个门就是抽象，而木头或者碳原子是细节。另外你可以看到门上有个门把手，你看到的不是铁，也不是铁原子，门把手就是抽象，铁和铁原子是细节。

在系统架构和设计中，抽象帮助我们从大处着眼（get our mind about big picture），隐藏细节（temporarily hide details）。抽象能力的强弱，直接决定我们所能解决问题的复杂性和规模大小。

下图是我们小时候玩的积木，我发现小时候喜欢玩搭积木的，并且搭得快和好的小朋友，一般抽象能力都比较强。



上图右边的积木城堡就是抽象，这个城堡如果你细看的话，它其实还是由若干个子模块组成，这些模块是子抽象单元，左边的各种形状的积木是细节。搭积木的时候，小朋友脑袋里头先有一个城堡的大图（抽象），然后他/她大脑里头会有一个初步的子模块分解（潜意识中完成），然用利用积木搭建每一个子模块，最终拼装出最后的城堡。这里头有一个自顶向下的分治设计，然后自底向上的组合过程，这个分治思维非常重要，我们后面会讲。

我认为软件系统架构设计和小朋友搭积木无本质差异，只是解决的问题域和规模不同罢了。架构师先要在大脑中形成抽象概念，然后是子模块分解，然后是依次实现子模块，最后将子模块拼装组合起来，形成最后系统。所以我常说编程和架构设计就是搭积木，优秀的架构师受职业习惯影响，眼睛里看到的世界都是模块化拼装组合式的。

抽象能力不仅对软件系统架构设计重要，对建筑、商业、管理等人类其它领域活动同样非常重要。其实可以这样认为，我们生存的世界都是在抽象的基础上构建起来的，离开抽象人类将寸步难行。

这里顺便提一下抽象层次跳跃问题，这个在开发中是蛮普遍的。有经验的程序员写代码会保持抽象层次的一致性，代码读起来像讲故事，比较清晰易于理解；而没有经验的程序员会有明显的抽象层次跳跃问题，代码读起来就比较累，这个是抽象能力不足造成。举个例子:

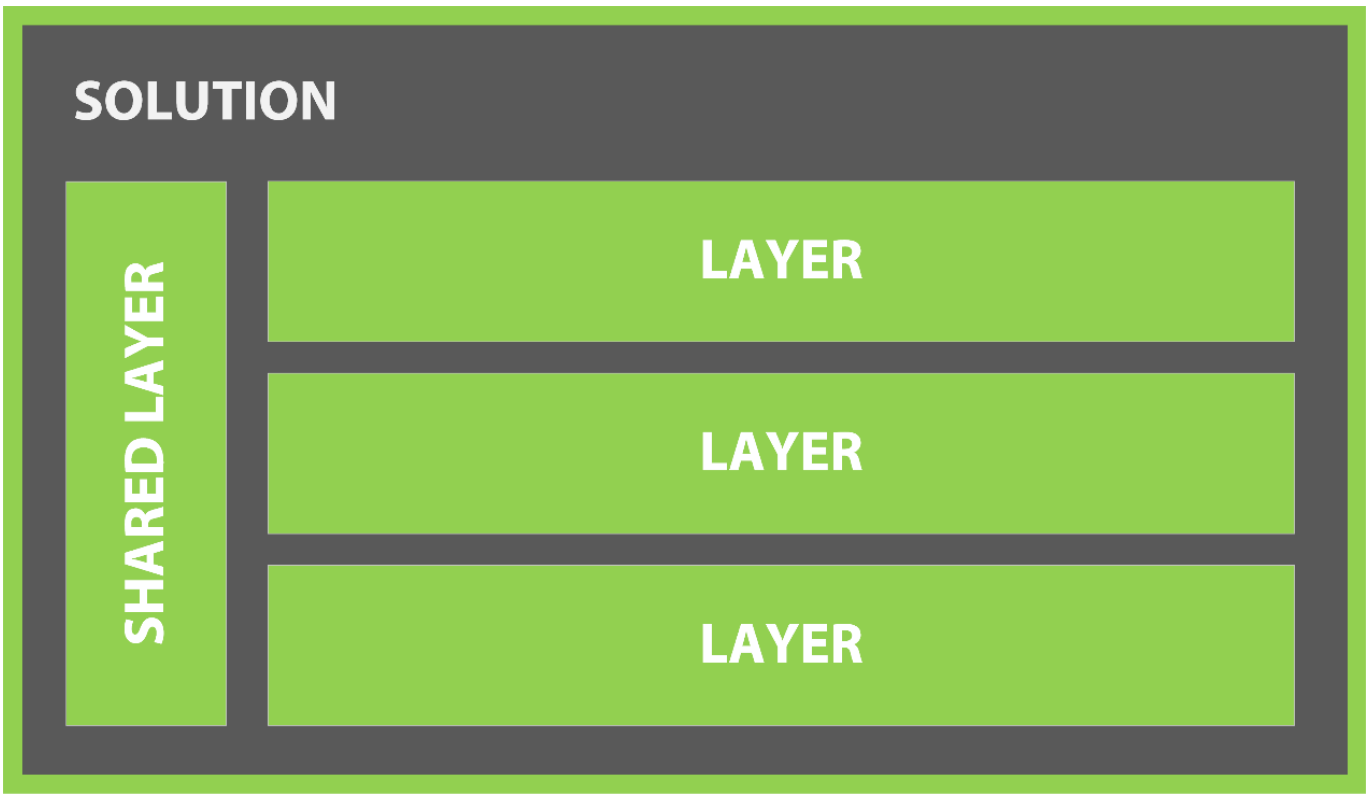
一个电商网站在处理订单时，一般会走这样一个流程：

- 1. 更新库存(InventoryUpdate)
- 2. 打折计算(Discounting)
- 3. 支付卡校验(PaycardVerification)
- 4. 支付(Pay)
- 5. 送货(Shipping)

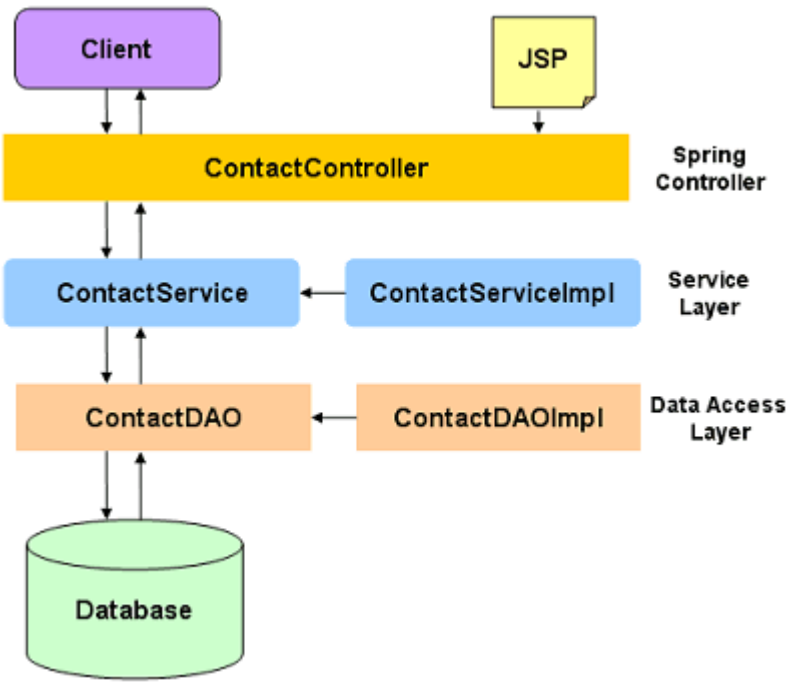
上述流程中的抽象是在同一个层次上的，比较清晰易于理解，但是没有经验的程序员在实现这个流程的时候，代码层次会跳，比方说主流程到支付卡校验一块，他的代码会突然跳出一行某银行API远程调用，这个就是抽象跳跃，银行API调用是细节，应该封装在PaycardVerification这个抽象里头。

#二、分层思维

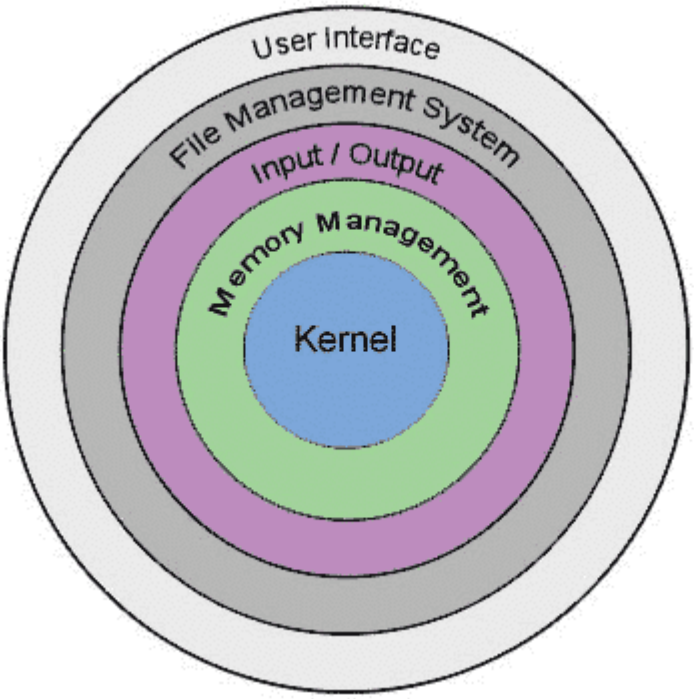
除了抽象，分层也是我们应对和管理复杂性的基本思维武器，如下图，为了构建一套复杂系统，我们把整个系统划分成若干个层次，每一层专注解决某个领域的问题，并向上提供服务。有些层次是纵向的，它贯穿所有其它层次，称为共享层。分层也可以认为是抽象的一种方式，将系统抽象分解成若干层次化的模块。



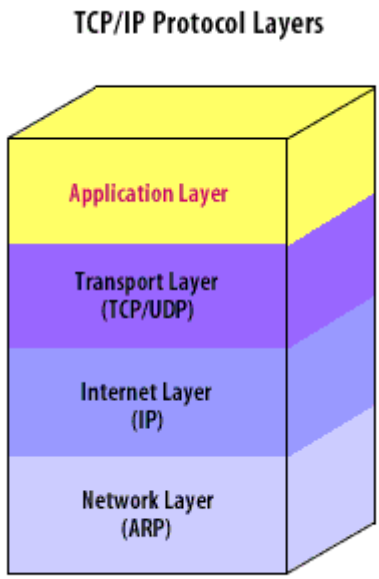
分层架构的案例很多，一个中小型的Spring Web应用程序，我们一般会设计成三层架构：



操作系统是经典的分层架构，如下图：



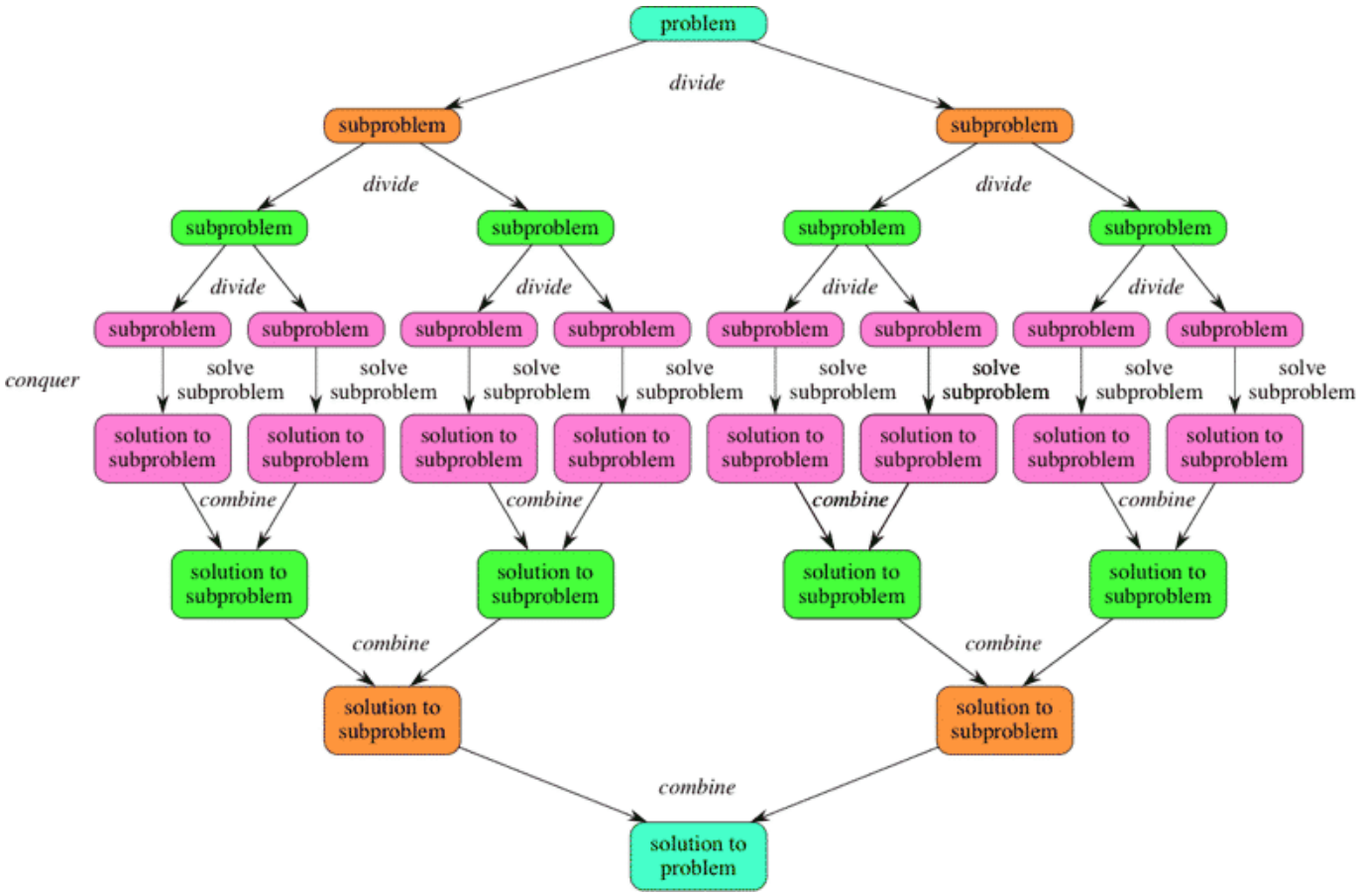
TCP/IP协议栈也是经典的分层架构，如下图：



如果你关注人类文明演化史，你会发现今天的人类世界也是以分层方式一层层搭建和演化出来的。今天的互联网系统可以认为是现代文明的一个层次，其上是基于互联网的现代商业，其下是现代电子工业基础设施，诸如此类。

#三、分治思维

分而治之(divide and combine或者split and merge)也是应对和管理复杂性的一般性方法，下图展示一个分治的思维流程：



对于一个无法一次解决的大问题，我们会先把大问题分解成若干个子问题，如果子问题还无法直接解决，则继续分解成子子问题，直到可以直接解决的程度，这个是分解(divide)的过程；然后将子子问题的解组合拼装成子问题的解，再将子问题的解组合拼装成原问题的解，这个是组合(combine)的过程。

面试时为了考察候选人的分治思维，我经常会面一个分治题：给你一台8G内存/500G磁盘空间的普通电脑，如何对一个100G的大文件进行排序？假定文件中都是字符串记录，一行约100个字符。

这是一个典型的分治问题，100G的大文件肯定无法一次加载到内存直接排序，所以需要先切分成若干小问题来解决。那么8G内存的计算机一次大概能排多大的数据量，可以在有限的时间内排完呢？也就是100G的大文件要怎么切法，切成多少份比较合适？这个是考察候选人的时间空间复杂度估算能力，需要一定的计算机组织和算法功底，也需要一定实战经验和sense。实际上8G内存的话，操作系统要用掉一部分，如果用Java开发排序程序，大致JVM可用2~4G内存，基于一般的经验值，一次排1G左右的数据应该没有问题（我实际在计算机上干过1G数据的排序，是OK的）。所以100G的文件需要先切分成100份，每份1G，这样每个子文件可以直接加载到内存进行排序。对于1G数据量的字符串排序，采用Java里头提供的快速排序算法是比较合适的。

好，经过有限时间的排序（取决于计算机性能，快的一天内能排完），假定100个1G的文件都已经排好了，相当于现在硬盘上有100个已经排好序的文件，但是我们最终需要的是一个排好序的文件，下面该怎么做？这个时候我们需要把已经解决的子问题组合起来，合并成我们需要的最终结果文件。这个时候该采用什么算法呢？这里考察候选人对外排序和归并排序算法的掌握程度，我们可以将100个排好序的文件进行两两归并排序，这样不断重复，我们就会得到50个排好序的文件，每个大小是2G。然后再两两归并，不断重复，直到最后两个文件归并成目标文件，这个文件就是100G并且是排好序的。因为是外排序+归并排序，每次只需要读取当前索引指向的文件记录到内存，进行比较，小的那个输出到目标文件，内存占用极少。另外，上面的算法是两路归并，也可以采用多路归并，甚至是采用堆排序进行优化，但是总体分治思路没有变化。

总体上这是一个非常好的面试题，除了考察候选人的分治思维之外，还考察对各种排序算法（快排，外排序，归并排序，堆排序）的理解，计算的时间空间复杂度估算，计算机的内外存特性和组织，文件操作等等。实际上能完全回答清楚这个问题的候选人极少，如果有幸被我面到一个，我会如获至宝，因为这个人有成长为优秀架构师的潜质。

另外，递归也是一种特殊的分治技术，掌握递归技术的开发人员，相当于掌握了一种强大的编程武器，可以解决一些一般开发人员无法解决的问题。比方说最近我的团队在研发一款新的服务框架，其中包括契约解析器(parser)，代码生产器(code generator)，序列化器(serializer)等组件，里头大量需要用到递归的思维和技术，没有这个思维的开发人员就干不了这个事情。所以我在面试候选人的时候，一般都会出递归相关的编程题，考察候选人的递归思维。

大自然中递归结构比比皆是，如下图，大家有兴趣不妨思考，大自然通过递归给我们人类何种启示?

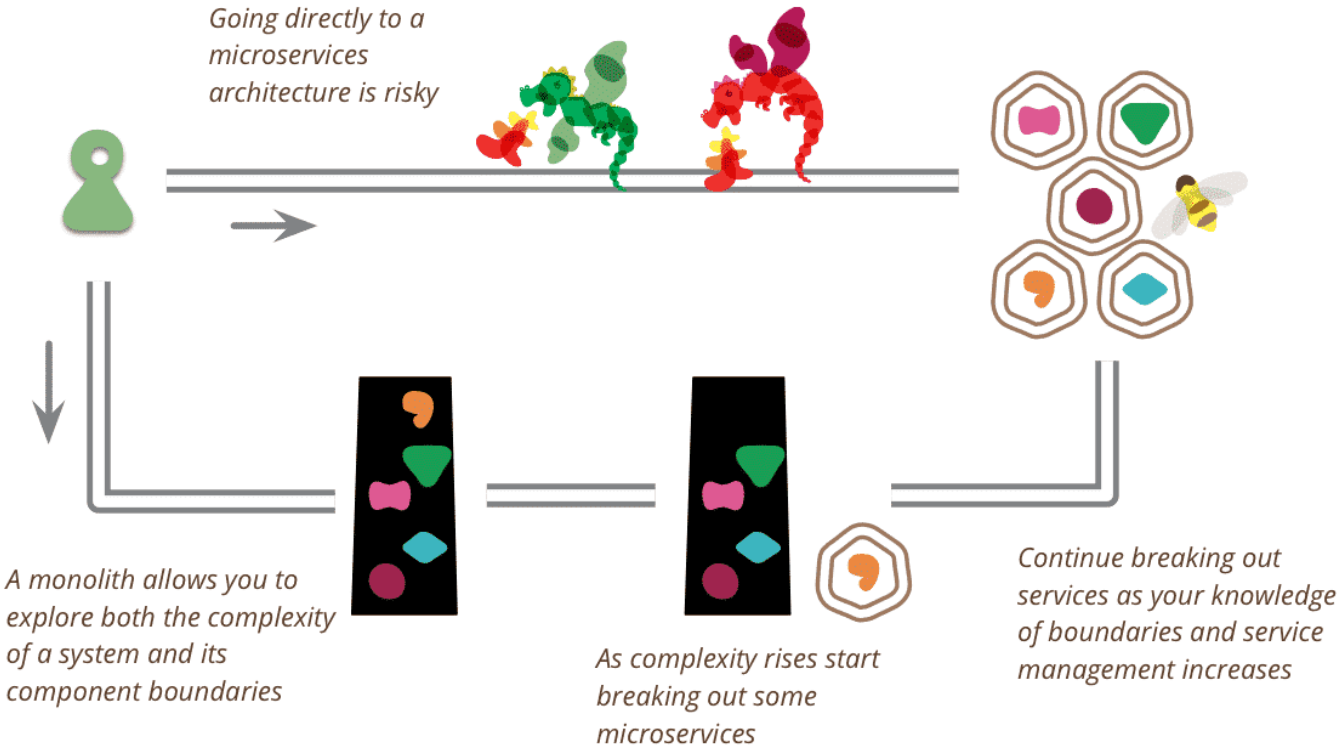
#四、演化思维

社区里头经常有人在讨论：架构是设计出来的？还是演化出来的？我个人基于十多年的经验认为，架构既是设计出来的，同时也是演化出来的，对于互联网系统，基本上可以说是三分设计，七分演化，而且是在设计中演化，在演化中设计，一个不断迭代的过程。

在互联网软件系统的整个生命周期过程中，前期的设计和开发大致只占三分，在后面的七分时间里，架构师需要根据用户的反馈对架构进行不断的调整。我认为架构师除了要利用自身的架构设计能力，同时也要学会借助用户反馈和进化的力量，推动架构的持续演进，这个就是演化式架构思维。

当然一开始的架构设计非常重要，架构定系统基本就成型了，不容马虎。同时，优秀的架构师深知，能够不断应对环境变化的系统，才是有生命力的系统，架构的好坏，很大部分取决于它应对变化的灵活性。所以具有演化式思维的架构师，能够在一开始设计时就考虑到后续架构的演化特性，并且将灵活应对变化的能力作为架构设计的主要考量。

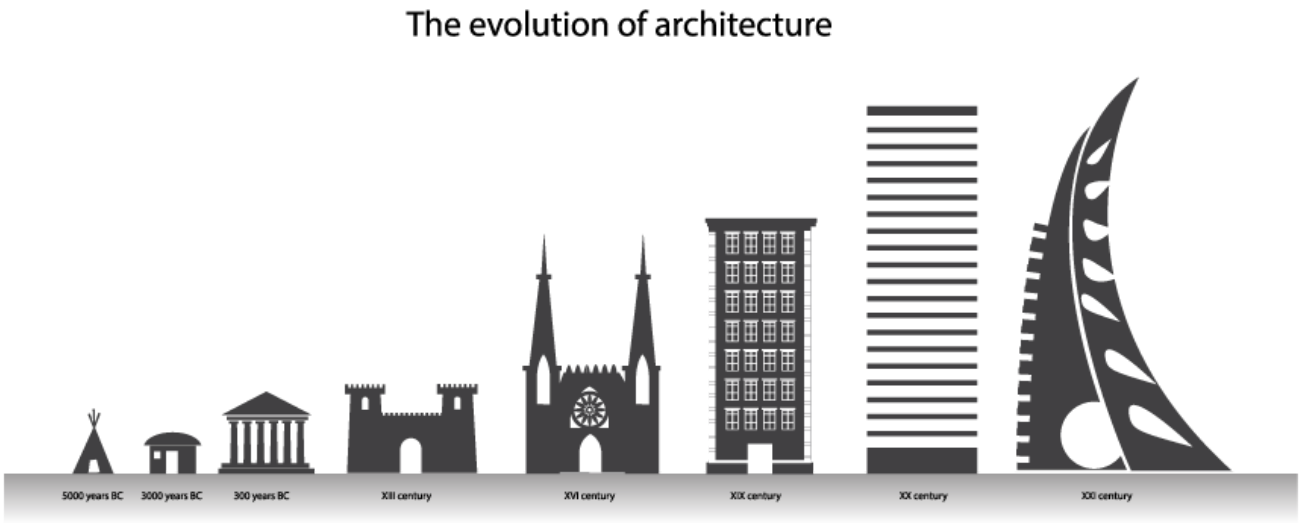
当前，社区正在兴起一种新的架构方法学~演化式架构，微服务架构就是一种典型的演化式架构，它能够快速响应市场用户需求的变化，而单块架构就缺乏这种灵活性。马丁·福乐曾经在其博客上给出过一张微服务架构的演化路线图[附录8.2]，可以用来解释设计式思维和演化式思维的差异，如下图所示：



上面的路线是一开始就直奔微服务架构，其实背后体现的是设计式架构的思维，认为架构师可以完全设计整个系统和它的演化方向。马丁认为这种做法风险非常高，一个是成本高昂，另外一个是刚开始架构师对业务域理解不深，无法清晰划分领域边界，开发出来的系统很可能无法满足用户需求。

下面的路线是从单块架构开始，随着架构师对业务域理解的不断深入，也随着业务和团队规模的不断扩大，渐进式地把单块架构拆分成微服务架构的思路，这就是演化式架构的思维。如果你观察现实世界中一些互联网公司（例如eBay，阿里，Netflix等等）的系统架构，大部分走得都是演化式架构的路线。

下图是建筑的演化史，在每个阶段，你可以看到设计的影子，但如果时间线拉得足够长，演化的特性就出来了。



#五、如何培养架构设计思维

良好的架构设计思维的培养，离不开工作中大量高质量项目的实战锻炼，然后是平时的学习、思考和提炼总结。

另外，基本的架构设计思维，其实在我们大学计算机课程（比如数据结构和算法）中可以找到影子，只不过当时以学习为主，问题域比较小和理想化。所以大学教育其实非常重要，基本的架构设计思维在那个时候就已经埋下种子，后面工程实践中进一步消化和应用，随着经验的积累，我们能够解决的问题域复杂性和规模逐渐变大，但基本的武器还是抽象、分层和分治等思维。

我认为一个架构师的成长高度和他大学期间的思维习惯的养成关系密切。我所知道世界一流的互联网公司，例如谷歌等，招聘工程师新人时，对数据结构和算法的要求可以用苛刻来形容，这个可以理解，谷歌级别公司要解决的问题都是超级复杂的，基本思维功底薄弱根本无法应对。

对于工作经验<5年的工程师新手，如果你大学时代是属于荒废型的，建议工作之余把相关课程再好好自学一把。个人推荐参考美国Berkeley大学的数据结构课程CS61B[附录8.1]进行学习，对建立抽象编程思维非常有帮助，我本人在研究生阶段自学过这门课程，现在回想起来确实受益匪浅，注意该课程中的所有Lab/Homework/Project都要实际动手做一遍，才有好的效果。

对于演化设计思维，当前的大学教育其实培养很少，相反，当前大学教育大都采用脱离现实场景的简化理想模型，有些还是固定答案的应试教学，这种方式会造成学生思维确定化，不利于培养演化式设计思维。我个人的体会，演化式设计思维更多在实际工作中通过实战锻炼和培养。

#结论

- 1. 架构的本质是管理复杂性，抽象、分层、分治和演化思维是架构师征服复杂性的四种根本性武器。
- 2. 掌握了抽象、分层、分治和演化这四种基本的武器，你可以设计小到一个类，一个模块，一个子系统，或者一个中型的系统，也可以大到一个公司的基础平台架构，微服务架构，技术体系架构，甚至是组织架构，业务架构等等。
- 3. 架构设计不是静态的，而是动态演化的。只有能够不断应对环境变化的系统，才是有生命力的系统。所以即使你掌握了抽象、分层和分治这三种基本思维，仍然需要演化式思维，在设计的同时，借助反馈和进化的力量推动架构的持续演进。
- 4. 架构师在关注技术，开发应用的同时，需要定期梳理自己的架构设计思维，积累时间长了，你看待世界事物的方式会发生根本性变化，你会发现我们生活其中的世界，其实也是在抽象、分层、分治和演化的基础上构建起来的。另外架构设计思维的形成，会对你的系统架构设计能力产生重大影响。可以说对抽象、分层、分治和演化掌握的深度和灵活应用的水平，直接决定架构师所能解决问题域的复杂性和规模大小，是区分普通应用型架构师和平台型/系统型架构师的一个分水岭。

#参考

1. Berkeley CS61B http://datastructur.es/sp17/
2. 单块优先 https://www.martinfowler.com/bliki/MonolithFirst.html

#参考文章

- https://www.cnblogs.com/lfs2640666960/p/9439857.html