系统架构:系统架构师是怎样炼成的

坦率的讲，除了少数对开发程序极其热爱并愿意为之奋斗终身的编程者来说，对于大多数开发人员，写代码只是他们未来获得职业提升的一个必不可少的积累阶段，在做开发的时间里，他们会积极学习各种知识，经验，培养自己的商业头脑，包括扩展自己各方面的资源，这些积累会为他们未来成为管理者或创业打下牢固的基础。

成为架构设计师是广大开发者职业发展道路之一，架构师究竟是个什么样的职业？需要具备什么基本能力？如何才能成为一个优秀的架构设计师以及架构设计师需要关注哪些内容？针对有关问题，本期我们为您采访了(微软认证专家，系统分析员，希赛顾问团顾问，中国计算机学会会员) 张友邦，他会就相关问题与大家分享他的看法。

“在我工作的六年多时间里，除了第一年是纯粹编码以外，其余时间都在做和架构设计有关的工作，当然也还一直在写各种各样的代码。”张友邦认为架构设计可能看起来很神秘，新入门或没有架构设计经验的程序员刚开始的时候会有种不知所措的感觉，但其实架构设计是件很容易的事，它只是软件系统开发中的一个环节而已，整个软件系统的开发和维护以及变更还涉及到很多事情，包括技术、团队、沟通、市场、环境等等。

同时，张友邦表示，虽然架构设计是件容易的事情，但也不是大多数没有架构设计经验的程序员想象中的画画框图那么简单。把几台服务器一摆，每一台服务器运行什么软件分配好，然后用网络连接起来，似乎每个企业级应用都是如此简间单单的几步。但现实生活中的软件系统实实在在可以用复杂大系统来形容，从规划、开发、维护和变更涉及到许许多多的人和事。架构设计就是要在规划阶段都把后面的事情尽量把握进来，要为稳定性努力，还要为可维护性、扩扩展性以及诸多的性能指标而思前想后。除了技术上的考虑，还要考虑人的因素，包括人员的组织、软件过程的组织、团队的协作和沟通等。

另外，架构设计还需要方法论的指导。张友邦强调，这些方法论的思路包括，至上而下的分析，关注点分离，横向/纵向模块划分等。有时候觉得架构设计决策就像是浏览Google Earth，实际上反映的是一种自上而下的决策过程。对问题的分解是软件思维的基本素质，可以有横向分解、纵向分解以及两者的结合。能不能有效快速准确的分解问题，是软件开发人员需要首先训练的项目。另外，架构设计中图形化的工具非常有用，它能把系统的结构和运作机制以图形化的方式表达出来。也正因为这样才有了架构设计就是画框图的误会。再者，架构设计是一个工程性质的工作，对当事人的实际从业经验要求较高。只有对市场上的各种技术有较全面的了解之后才有可能设计出一个尽可能满足各种设计约束的架构。

在谈到架构师需要具备的能力上，张友邦认为架构师首先必须具有丰富的开发经验，是个技术主管。因为他必须清楚什么是可以实现的，实现的方式有哪些，相应的难度怎么样，实现出来的系统面对需求变化的适应性等一系列指标。另外，需要对面向过程、面向对象、面向服务等设计理念有深刻的理解，可以快速的察觉出实现中的问题并提出相应的改进(重构)方案(也就是通常说的反模式)。这些都需要长期的开发实践才能真正的体会到，单从书本

上很难领会到，就算当时理解了也不一定能融会到实践中去。

在技术能力上，软件架构师最重要也是最需要掌握的知识是构件通信机制方面的知识，包括进程内通信(对象访问、函数调用、数据交换、线程同步等)以及进程外(包括跨计算机)的通信(如 RMI、DCOM、Web Service)。在 WEB 应用大行其道的今天，开发者往往对服务器间的通信关注的比较多，而对进程内的通信较少关注。进程外跨机器通信是构建分布式应用的基石，它是架构设计中的鸟瞰视图;而进程内的通信是模块实现的骨架，它是基石的基石。如果具体到一个基于.Net 企业级架构设计，首先需要的是语言级别的认识，包括.NET 的 CLR、继承特性、委托和事件处理等。然后是常用解决方案的认识，包括 ASP.NET Web Service、.NET Remoting、企业服务组件等。总之，丰富的开发实践经验有助于避免架构师纸上谈兵式的高来高去，给代码编写人员带来实实在在的可行性。

其次，具有足够的行业业务知识和商业头脑也是很重要的。行业业务知识的足够把握可以给架构师更多的拥抱变化的能力，可以在系统设计的时候留出一些扩展的余地来适应可能来临的需求变化。有经验的设计人员可能都碰到过这样的事，一厢情愿的保留接口在需求变化中的命中率非常低。也就是说，在系统设计之初为扩展性留下来的系统接口没能在需求变化的洪流中发挥真正的作用，因为需求的变化并没有按照预想的方向进行，到最后还是不得不为变化的业务重新设计系统。这就是因为对业务知识的理解和对市场或者商业的判断没有达到一个实用的、可以为架构扩展性服务的水平。

再次，张友邦提到，架构设计师对人的关注必须提升到架构设计之初来纳入考虑的范围，包括沟通以及对人员素质的判断。软件过程是团队协作共同构建系统的过程，沟通能力是将整个过程中多条开发线粘合在一起的胶水。大家都应该碰到过事后说“原来是这样啊，我不知道啊”或者某个开发人员突然高声呼喊“为什么这里的数据没有了”之类的。沟通的目的就是尽量避免多条开发线的混乱，让系统构建过程可以有条理的高效进行。另外，对人的关注还表现在对团队成员的素质判断上，比如哪些开发人员对哪些技术更熟悉，或者哪些开发人员容易拖进度等。只有合理的使用人力资源，让合适的人做合适的事情才能让整个软件过程更加高效。

另外，张友邦认为架构师应时刻注意新软件设计和开发方面的发展情况，并不断探索更有效的新方法、开发语言、设计模式和开发平台不断很快地升级，软件架构师需要吸收这些新技术新知识，并将它们用于软件系统开发工作中。但对新技术的探索应该在一个理性的范围内进行，不能盲目的跟风。解决方案提供商永远都希望你能使用它提供的最新技术，而且它们在推广自己的解决方案的时候往往是以自己的产品为中心，容易给人错觉。比如数据库，往往让人觉得它什么都能做，只要有了它其它什么都不重要了。但事实上并不是如此，对于小型应用可以将许多业务逻辑用 script 的方式放入数据库中，但很少看到大型应用采用这样的做法。对于新东西需要以一种比较的观点来判断，包括横向的比较和纵向的比较，最后得出一些性能、可移植性以及可升级等指标。另外，新入行的开发人员往往关心新技术动向而忽略了技术的历史，而从 DOS 时代一路杀过来的开发者就对现在的技术体系有较全面的把握。

构架师不是通过理论学习可以搞出来的，不学习并且亲自实践相关知识肯定是不行的。就像前面说到的，架构设计是一个工程性质的事情，只有在不断实践的基础上才能逐渐熟悉起来。实践的内容并不是去深挖各种语言的特性，因为系统架构师是设计应用系统架构而不是

设计语言(除非你是要实现 DSL)。更多的时候需要带着一种比较的眼光去实践，把不同的实现方式下的优缺点做个总结，做到自己心里有数，等具体的上下文环境下才好判断采用什么样的方式方法。把基础打牢的同时掌握一定的方法，架构设计不是想象中的那么难。

张友邦，男，微软认证专家，系统分析员，希赛顾问团顾问，中国计算机学会会员。1980年生于四川宜宾，2002 年获得国防科技大学宇航科学与工程系空间工程专业学士学位，2004年初成立长沙石斑软件有限公司并担任总经理，2006 年底出任广州快网信息技术有限公司技术总监，2007 年 10 月任湖南新邮信息技术有限公司软件中心副经理。主要研究领域包括软件架构与设计、WEB RIA、流媒体与计算机图形图像。受国家自然科学基金资助，于 2001年发表国家级核心刊物学术论文一篇。

系统架构:小议软件架构设计要点

2009 年上半年计算机技术与软件专业技术资格（水平）考试日期：2009 年 5 月 23、24 日。另外，部分考试科目从 2009 年上半年开始将采用新修编的考试大纲，具体见：

如何更好地进行软件架构设计，这是软件工程领域中一个永恒的重点话题。过去几十年来，国际软件工程界在软件架构设计方面已经获得了长足发展，大量图书、文章和文献记载了这方面的成熟经验与成果。软件架构设计往往是一件非常复杂的工作，涉及到很多细节和方方面面，可探讨的话题也非常之多。囿于篇幅限制，以下只能根据笔者个人理解，遴选出软件架构设计的个别要点，结合当前流行的敏捷软件工程思想，与大家分享一下自己在软件架构设计方面的心得和体会。

架构决定成败

软件架构是软件产品、软件系统设计当中的主体结构和主要矛盾。任何软件都有架构，哪怕一段短小的 HelloWorld 程序。软件架构设计的成败决定了软件产品和系统研发的成败。软件架构自身所具有的属性和特点，决定了软件架构设计的复杂性和难度。

这几年流行一个说法（管理谚语）：“细节决定成败”，这句话其实只说对了一半。细节确实很重要，很多项目、产品就输在细节的执行上。一方面，战术细节固然很重要，但另一方面，战略全局也同样重要，对应的我们可以说：“战略决定成败”。战略性失败，就好比下一盘围棋，局部下得再漂亮、再凌厉，如果罔顾大盘，己方连空都不够了，还有官子（细节）获胜的机会吗？必然是中盘告负。

类似地，正确的软件架构设计，应该既包括战略全局上的设计，也包括战术细节（关键路径）上的设计。有一种错误的观点认为，软件架构设计只要分分层和包，画一个大体的轮廓草图，就完事了。这种“纸上谈兵”型的架构师行为是非常有害的。事实上，既然软件架构是软件建筑的主体结构、隐蔽工程、承重墙和要害部位，那么软件架构也必然要落实到实际的算法和代码，不但要有实现代码，还要包括对这部分架构进行测试的代码，以保证获得高质量的、满足各种功能和非功能质量属性要求的架构。除了完成概念、模型设计外，软件架构师一定要参与实际的编码、测试和调试，做一位真正的 hands-on practitioner，这已经成为

了敏捷软件工程所倡导的主流文化。

两个架构

我们在日常的软件产品和系统开发中，实际上会遇到两种、两个部分的软件架构，即待开发的应用部分的软件架构（简称“应用架构”），以及既有的基础平台部分的软件架构（简称“基础架构”）。这两部分架构之间是互为依赖、相辅相成的关系，它们共同组成了整个软件产品和系统的架构。

基础架构的例子包括：.NET 和 J2EE 等主流的基础平台和各种公共应用框架，由基础库 API、对象模型、事件模型、各种开发和应用的扩展规则等内容组成。我们只有熟悉基础架构的构造细节、应用机理，才能有效地开发出高质量、高性能的上层应用。然而，开发一个面向最终用户的软件应用系统和产品，仅仅掌握一般的计算机高级编程语言知识和基础平台架构、API 的使用知识显然是不够的，我们还需要根据客户应用的类型和特点，在基础架构之上，设计出符合用户要求的高质量应用软件。

熟悉 OOA、OOD 抽象建模技术、设计原则以及架构模式和设计模式等等方法技术，不但有助于我们更好地理解和利用基础平台架构，也有助于我们设计开发出更高质量的应用软件架构。

风险驱动、敏捷迭代的架构设计与开发

软件架构将随着软件产品和系统的生命周期而演化，其生命期往往超过了一个项目、一次发布，甚至有可能长达数年之久，因而软件架构无论对于客户还是开发商来说都是一项极其重要的资产。

软件架构的设计应该遵循什么样的开发过程？或者说，有没有更好的、成熟的软件架构设计和开发过程？回答是，21 世纪的软件架构设计应该优先采用敏捷迭代的开发方式和方法。与传统做法不同，敏捷迭代开发主张软件架构采用演进式设计（evolutionary design），一个软件产品或系统的架构是通过多次迭代，乃至多次发布，在开发生命周期中逐步建立和完善起来的。

好的软件架构不是一蹴而就的。在架构设计开发过程中，我们应该尽量避免瀑布式思维，通过一个“架构设计阶段”来完成系统的架构设计乃至详细设计，然后再根据架构图纸和模型，在“编码实现阶段”按图索骥进行架构的编码与实现。这种传统做法的错误在于认为软件架构就是图纸上的模型，而不是真正可以高质量执行的源代码。几十年的软件工程实践表明，没有经过代码实现、测试、用户确认过的架构设计，往往会存在着不可靠的臆想、猜测和过度设计、过度工程，极易造成浪费和返工，导致较高的失败率。

风险是任何可能阻碍和导致软件产品/系统研发失败的潜在因素和问题。软件架构是软件产品和系统研发的主要矛盾和主要技术风险，软件架构的质量决定了整个软件系统和产品的质量。不确定性往往是软件架构设计当中一种最大的潜在风险。因此，软件架构的设计与开发应该遵循风险驱动的原则，在整个开发生命周期内至始至终维护一张风险问题清单，随着迭代的前进，根据风险的实时动态变化，首先化解和处理最主要的架构风险，再依次化解和处理次要的架构风险。

架构设计的可视化建模

软件架构设计的难度源于软件设计问题本身的复杂性，一个复杂的软件系统往往存在大量复杂的、难于被人类所理解的细节和不确定因素。抽象与建模是人类自诞生以来就已掌握的

理解复杂事物的方法，因而人类所从事的软件设计工作本质上也是一个不断建模的过程。我们可以通过各种抽象的模型和视图，从各个不同层次、宏观和微观的角度来理解复杂的软件架构，以保证作出正确和有效的设计。

有人认为：“软件架构就是源代码（source codes）”以及“源代码就是设计”。这种说法其实是片面的。什么是真正的软件？我们知道，最终可以在电脑上执行的真正的软件其实是二进制代码 0 和 1，借助编译器我们把高级编程语言翻译成底层的汇编语言、机器语言等，没有人能直接、完整地看到二进制程序在 CPU 上的实际运行状况（runtime），人们大多只能通过各种调试工具、窗口视图等方式来间接地动态观察这些真正的软件的运行片段。因此，Java、C#、C++ 等等设计时（design time）源代码在本质上也是一种模型，虽然是一种经处理后可执行的静态模型，但显然它们并不是真实软件和软件架构的全部。可见，源代码模型（有时也叫实现模型）与 UML 模型其实都是软件架构的一种模型（逻辑反映），差别就在于抽象层次的不同。完整的软件架构（建筑）不仅仅包括源代码（实现模型），还包括了需求模型、分析模型、设计模型、实现模型和测试模型等等许多模型，软件架构本身就是一组模型的集合。

UML、SysML 是当前国际上流行的软件/系统架构可视化建模语言。在编写实际的代码之前，利用包图、类图、活动图、交互图、状态图等等各种标准图形符号对软件架构进行建模，探讨和交流各种可行的设计方案，发现潜在的设计问题，保证具体编码实现之前抽象设计的正确性，被实践证明是一种非常有效和高效、敏捷的工作方式。

架构设计的重用

重用（Reuse）是在软件工程实践中获得高效率、高质量产品和系统开发的一种基本手段和主要途径，通过有组织的、系统和有效的重用，我们往往可以获得 10 倍率以上的效率提升。而一个优秀的、有长久生命力的软件架构（比方主流的一些框架软件），其本身或其组件被重用的次数越多，其体现的价值也就越大。

软件重用有各种不同的范围、层次、粒度和类型，从函数重用、类重用、构件/组件重用、库（API）重用，到框架重用、架构重用、模式重用，再到软件设计知识、思想的重用等等，重用的效能和效果各有不同。

软件工程经过几十年的发展，已经积累了大量的软件架构模式和设计模式，它们记载、蕴藏了大量成熟、已经验证的软件设计知识、思想和经验。我们平时对各种基础平台、主流框架和 API 的应用和调用，本身就是一种最为普遍的重用形式。而一个优秀、成熟的软件研发组织，必然会在日常开发中注意收集各种软件设计知识和经验，建立和维护基于架构模式和设计模式等内容的软件重用知识库，积极主动和频繁地运用各种软件模式来解决实际工程问题。

系统架构设计师:hibernate 配置

24.2 hibernate 配置

Hibernate 同时支持 xml 格式的配置文件，以及传统的 properties 文件配置方式，一般我们采用 xml 型配置文件。xml 配置文件提供了更易读的结构和更强的配置能力，可以直接对映射文件加以配置.

24.3Hibernate 的使用问题 24.3.1 标示符 l)Increment 2)Identity 3)Sequence 4)Assign

5)Hilo

24.3.2 注意的问题

l)Lazy 2)Inverse 3)Outer-join

4)大数据量删除更新

24.4 技术经理和高级程序员如何利用 Spring 整合 hibernate l)SessionFactory

2}HibernateDaoSupport 3） HibernateTemplate

系统架构设计师:处理图像

处理图像

如果您的应用程序显示大量图像文件（例如，.jpg 和 .gif 文件），则您可以通过以位图格式预先呈现图像来显著改善显示性能。要使用该技术，请首先从文件中加载图像，然后使用 PARGB 格式将其呈现为位图。下面的代码示例从磁盘中加载文件，然后使用该类将图像呈现为预乘的、Alpha 混合 RGB 格式。例如： [C#]

if (image != null && image is Bitmap) {

Bitmap bm = (Bitmap)image;

Bitmap newImage = new Bitmap(bm.Width, bm.Height, System.Drawing.Imaging.PixelFormat.Format32bppPArgb); using (Graphics g = Graphics.FromImage(newImage)) {

g.DrawImage(bm, new Rectangle(0,0, bm.Width, bm.Height)); }

image = newImage; }

[Visual Basic .NET]

If Not(image Is Nothing) AndAlso (TypeOf image Is Bitmap) Then Dim bm As Bitmap = CType(image, Bitmap)

Dim newImage As New Bitmap(bm.Width, bm.Height, _ System.Drawing.Imaging.PixelFormat.Format32bppPArgb) Using g As Graphics = Graphics.FromImage(newImage)

g.DrawImage(bm, New Rectangle(0, 0, bm.Width, bm.Height)) End Using

image = newImage End If

系统架构设计师:管理可用资源

管理可用资源

公共语言运行库 (CLR) 使用垃圾回收器来管理对象生存期和内存使用。这意味着无法再访问的对象将被垃圾回收器自动回收，并且自动回收内存。由于多种原因无法再访问对象。例如，可 能没有对该对象的任何引用，或者对该对象的所有引用可能来自其他可作为当前回收周期的一部分进行回收的对象。尽管自动垃圾回收使您的代码不必负责管理对象 删除，但这意味着您的代码不再对对象的确切删除时间具有显式控制。请考虑下列原则，以确保您

能够有效地管理可用资源：

1)确保在被调用方对象提供 Dispose 方法时该方法得到调用。如果您的代码调用了支持 Dispose 方法的对象，则您应该确保在使用完该对象之后立即调用此方法。调用 Dispose 方法可以确保抢先释放非托管资源，而不是等到发生垃圾回收。除了提供 Dispose 方法以外，某些对象还提供其他管理资源的方法，例如，Close 方法。在这些情况下，您应该参考文档资料以了解如何使用其他方法。例如，对于 SqlConnection 对象而言，调用 Close 或 Dispose 都足可以抢先将数据库连接释放回连接池中。一种可以确保您在对象使用完毕之后立即调用 Dispose 的方法是使用 Visual C# .NET 中的 using 语句或 Visual Basic .NET 中的 Try/Finally 块。 下面的代码片段演示了 Dispose 的用法。

C# 中的 using 语句示例：

using(StreamReader myFile = new StreamReader("C:\\ReadMe.Txt")){ string contents = myFile.ReadToEnd(); //... use the contents of the file

} // dispose is called and the StreamReader’s resources released Visual Basic .NET 中的 Try/Finally 块示例： Dim myFile As StreamReader

myFile = New StreamReader("C:\\ReadMe.Txt") Try

String contents = myFile.ReadToEnd() ’... use the contents of the file Finally

myFile.Close()

End Try 注：在 C# 和 C++ 中，Finalize 方法是作为析构函数实现的。在 Visual Basic .NET 中，Finalize 方法是作为 Object 基类上的 Finalize 子例程的重写实现的。

2)如果您在客户端调用过程中占据非托管资源，则请提供 Finalize 和 Dispose 方法。如果您在公共或受保护的方法调用中创建访问非托管资源的对象，则应用程序需要控制非托管资源的生存期。在图 8.1 中，第一种情况是对非托管资源的调用，在此将打开、获取和关闭资源。在此情况下，您的对象无须提供 Finalize 和 Dispose 方法。在第二种情况下，在方法调用过程中占据非托管资源；因此，您的对象应该提供 Finalize 和 Dispose 方法，以便客户端在使用完该对象后可以立即显式释放资源。

系统架构设计师:规范

8.2.5 规范

您可以使用许多工具和技术来帮助您对应用程序进行规范，并且生成度量应用程序性能所需的信息。这些工具和技术包括：

1)Event Tracing for Windows (ETW)。 该 ETW 子系统提供了一种系统开销较低（与性能日志和警报相比）的手段，用以监控具有负载的系统的性能。这主要用于必须频繁记录事件、错误、警告或审核的服务器应 用程序。

2)Enterprise Instrumentation Framework (EIF)。EIF 是一种可扩展且可配置的框架，您可以使用它来对智能客户端应用程序进行规划。它提供了一种可扩展的事件架构和统一的 API - 它使用 Windows 中内置的现有事件、日志记录和跟踪机制，包括 Windows Management Instrumentation (WMI)、Windows Event Log 和 Windows Event Tracing。它大大简化了发布应用程序事件所需的编码。

如果您计划使用 EIF，则需要通过使用 EIF .msi 在客户计算机上安装 EIF。如果您要在智能客户端应用程序中使用 EIF，则需要在决定应用程序的部署方式时考虑这一要求。

3)Windows Management Instrumentation (WMI)。WMI 组件是 Windows 操作系统的一部分，并且提供了用于访问企业中的管理信息和控件的编程接口。系统管理员常用它来自动完成管理任务（通过使用调用 WMI 组件的脚本）。

4)调试和跟踪类。.NET Framework 在 System.Diagnosis 下提供了 Debug 和 Trace 类来对代码进行规范。Debug 类主要用于打印调试信息以及检查是否有断言。Trace 类使您可以对发布版本进行规范，以便在运行时监控应用程序的完好状况。在 Visual Studio .NET 中，默认情况下启用跟踪。在使用命令行版本时，您必须为编译器添加 /d:Trace 标志，或者在 Visual C# .NET 源代码中添加 #define TRACE，以便启用跟踪。对于 Visual Basic .NET 源代码，您必须为命令行编译器添加 /d:TRACE=True。
系统架构设计师:规划 SOA 参考架构

2009 年上半年计算机技术与软件专业技术资格（水平）考试日期：2009 年 5 月 23、24 日。另外，部分考试科目从 2009 年上半年开始将采用新修编的考试大纲，具体见：

SOA 参考架构 (Reference Architecture) 是一个框架，使各个项目都有一个遵从的依据，借以促进一致性、最佳实践典范，和标准化。参考架构并不受限于目前的 IT 现况，而应该针对一个经过深思熟虑的愿景目标，可以说是 IT 指导未来所有的新开发工作，借以实现该目标的参考依据。一般来说，2-3 年的规划，是一个比较合适的涵盖范围，既能提供足够的时间来达成面向服务的转型，而又不至于过于长远而虚幻。因此，参考架构提供了一个沟通目标愿景的方 法，协助部门和角色各异的 IT 人员，逐渐朝向该目标会合。
高效的 SOA 需要采用新的方法来对待 IT 基础设施，并且根据个别企业的需求来量身定做，并将服务基础架构、共享的技术服务、安全服务，以及信息 / 数据、和遗留系统访问服务等，全部定义在内。

为了满足 SOA 的要求，所有公司都需要 SOA 参考架构和路线图，来指导部署一套能随时间演进、而逐渐丰富的工业级服务基础设施，同时指导对面向服务应用的开发和管理。 此外，企业也需要对参与 SOA 架构的各个个别系统的设计，进行监管，并在适当的地方，建立通用服务，透过协作来发挥更高的效率。对于这些举措，连接端点的标准化（通过建立定义清晰的契约和接口），是达成 IT 系统一致性的先决条件。
SOA 参考架构指导所有实施 SOA 的各个项目，能共同朝向企业级服务，和 SOA 基础架构标准方向的集中发展，尽早使企业从中获益。换句话说，参考架构规划的重点，在于开发一个特定于某个企业需要、切实可行的路线图，以填补当前和愿景 目标之间的鸿沟；评估用于开发、部署和管理、监控的现有系统和技术，定义目标状态愿景，目标参考架构模型。 SOA 参考架构可说是指导 SOA 成功的蓝图，其作用包括：

促进 IT 与业务的紧密配合： 参考架构的制定，以业务驱动力和 IT 目标为出发点，分析 SOA 解决方案能对这些驱动力带来多大的正面影响，进而为从目前 IT 现况演化到愿景架构，定出实现架构、相关规范及路线图。参考架构因此提供了从业务和 IT 目标，到实现架构间的可跟踪性，是业务与 IT 之间进行沟通的重要媒介，是企业实现业务灵活性、可管理性和变更规划的基础。

协助企业向重用、团队协作和资源共享的文化迁移：参考架构确立了 SOA 架构标准和技术部署的最佳实践，为日后各个 SOA 的实施项目，订立架构遵从性的度量标准和指标。 参考架构并非一成不变。在一个新的 SOA 策略与规划迭代中，SOA 的参考架构和规范标准，可能需要针对新的业务、IT 情况，和已实施的 SOA 项目中得到的反馈，进行调整，因此，SOA 参考架构不仅是 IT 模板，也是也描述 SOA 原则和标准的活文档。

我们可以将参考架构的内容，粗分为两大部分： 对服务建立一套共同的词汇和做法，包括：

服务的正式定义 - 例如服务必须由契约 (contract)、接口 (interface)，和实现 (implementation) 所组成

服务的分类（核心业务功能服务，数据服务，展现服务等），以及各类服务的设计原则和建议

接口标准 (JMS, RMI, HTTP 等)，建议的接口样式（例如：尽量采用粗粒度、异步的服务调用模式），可靠性要求等

需要遵从的 WS-* 标准 安全策略

服务版本控制策略

服务和数据模型采用规范 服务生命周期定义

与服务基础设施，例如企业服务总线 (ESB)、业务流程管理 (BPM)、注册库 (Registry)、资产库 (Repository) 等相关的规范，包括： 必须支持什么样的部署配置

必须具备什么样的能力 各个部件的责任

部件之间的耦合关系和原则，应避免的事项，例如，展现服务和业务流程服务不可直接调用数据服务，而必须通过核心业务服务；换句话说，数据服务不可直接与展现服务和业务流程服务有耦合关系

各个部件应支持那些科技和标准（例如：SCA, SDO…） 有哪些安全顾虑需要考虑，如何管理权限

要采用哪些产品

由于在规划服务基础设施参考架构时，需要涵盖几类 SOA 参与者和干系人 (stakeholders) 各自不同的顾虑，包括架构师、程序员、和负责部署、运营、监控的 IT 人员，我们可以采用一个针对服务基础设施参考架构调整过的 4+1 视图（如下），来协助我们分离顾虑，来将不同层面的规范和目标架构一一制定，通过逻辑、实现、部署，和进程等四个视图，配合最佳实践典范和模式，来对参考 架构的各个层面，进行描述和规范，如附图。

参考架构的规划过程（如下图），应先起于业务驱动力 (business drivers) 分析，可有助确保目标架构能够支持业务的发展策略和方向，展现 SOA 建设对业务的价值，彰显 SOA 投资的正当性，并获得相关业务部门的经费赞助。以金融行业为例，业务驱动力包括像是：

提升效率

借贷流程优化

呼叫中心优化

增长销售额，并显著超越同业 快速灵活地推出创新的金融商品

扩展客户关系，统合客户数据 交叉销售

依据关系定价的策略

降低成本

这一类的价值驱动。分析业务的价值驱动后，接着考虑各种 IT 目标，以及它们如何支持这些业务驱动力；例如：

从关注各业务线烟囱 / 竖井式的应用系统，转向专注于跨系统 / 业务线的流程 / 服务

IT 资产重用

提高跨部门协作的业务流程的透明度

并且应订立评价标准，作为日后考核实现各价值驱动力的指标。接着下来，我们可以根据业务和 IT 驱动力，进一步制定恰当的 SOA 策略，考虑采用 SOA，将对那些业务线，在那些驱动力方面，产生最大的价值，据以订立实施 SOA 项目的优先级别。

√ √ 代表 SOA 项目能产生很大的正向影响，依此类推。 针对各价值驱动力，我们可以参考附图中的矩阵分析方式，从价值链或业务线中的各个主要的职能（纵向），和各个识别出来的业务和 IT 驱动力（横向），对 SOA 所可能产生的正面影响力，一一进行评估，进而挑选出面向服务解决方案最能嘉惠的业务能力，作为首期实施 SOA 项目的切入点。图中的范例只是一个三万尺高空的起点，在真实的情况下，往往会针对范例中某个或某几个得分较高的业务线，往下展开细化，对该业务线中的各个 主要业务能力，进一步进行 SOA 价值驱动力分析；换句话说，价值链分析中的各个职能域，应该自粗到细，逐步钻取、drill down 到适当的深度，才能更精准地确定 SOA 能对哪些要迫切改善的业务能力，带来最大价值。

依据业务和 IT 驱动力，选定了 SOA 策略后。接着应根据企业目前的现况，和未来 2-3 年的目标，进行差距分析，并根据最佳实践典范 (best practices)，制定 SOA 发展的蓝图、路线图和指导规范，完成参考架构的规划。接着便可开始根据路线图中制定的项目，以渐进的方式，通过一致的服务工程方法，一个接一个项目去执 行，并在此过程中，逐渐将蓝图中的服务基础设施搭建起来。

系统架构设计师:考虑用户的观点

8.2.1.1 考虑用户的观点

当您为智能客户端应用程序确定合适的性能目标时，您应该仔细考虑用户的观点。对于智能客户端应用程序而言，性能与可用性和用户感受有关。例如，只要用户能够继续工作并且获得有关操作进度的足够反馈，用户就可以接受漫长的操作。在 确定要求时，将应用程序的功能分解为多个使用情景或使用案例通常是有用的。您应该识别对于实现特定性能目标而言关键且必需的使用案例和情景。应该将许多使 用案例所共有且经常执行的任务设计得具有较高性能。同样，如果任务要求用户全神贯注并且不允许用户从其切换以执行其他任务，则需要提供优化的且有效的用户 体验。如果任务不太经常使用且不会阻止用户执行其他任务，则可能无须进行大量调整。对于您识别的每个性能敏感型任务，您都应该精确地定义用户的操作以及应用程序的响应方式。您还应该确定每个任务使用的网络和客户端资源或组件。该信息将影响性能目标，并且将驱动对性能进行度量的测试。可 用性研究提供了非常有价值的信息源，并且可能大大影响性能目标的定义。正式的可用性研究在确定用户如何执行他们的工作、哪些使用情景是共有的以及哪些不是 共有的、用户经常执行哪些任务以及从性能观点看来应用程序的哪些特征是重要的等方面可能非常有用。如果您要生成新的应用程序，您应该考虑提供应用程序的原 型或模型，以便可以执行基本的可用性测试。

8.2.1.2 考虑应用程序操作环境

对应用程序的操作环境进行评估是很重要的，因为这可能对应用程序施加必须在您制定的性能目标中予以反映的约束。位于网络上的服务可能对您的应用程序施加性能约束。例如，

您可能需要与您无法控制的 Web 服务进行交互。在这种情况下，需要确定该服务的性能，并且确定这是否将对客户端应用程序的性能产生影响。您 还应该确定任何相关服务和组件的性能如何随着时间的变化而变化。某些系统会经受相当稳定的使用，而其他系统则会在一天或一周的特定时间经受变动极大的使 用。这些区别可能在关键时间对应用程序的性能造成不利影响。例如，提供应用程序部署和更新服务的服务可能会在星期一早上 9 点缓慢响应，因为所有用户都在此时升级到应用程序的最新版本。另外，还需要准确地对所有相关系统和组件的性能进行建模，以便可以在严格模拟应用程序的实际部署环境的环境中测试您的应用程序。对于每个系统，您都应该确定性能概况以及最低、平均和最高性能特征。然后，您可以在定义应用程序的性能要求时根据需要使用该数据。您还应该仔细考虑用于运行应用程序的硬件。您将需要确定在处理器、内存、图形功能等方面的目标硬件配置，或者至少确定一个如果得不到满足则无法保证性能的最低配置。通常，应用程序的业务操作环境将规定一些更为苛刻的性能要求。例如，执行实时股票交易的应用程序将需要执行这些交易并及时显示所有相关数据。

系统架构设计师:浅谈架构

2009 年下半年全国计算机等级考试你准备好了没?考计算机等级考试的朋友,2009 年下半年全国计算机等级考试时间是 2009 年 9 月 19 日至 23 日。

不得不说的就是规范性的东西，我认为规范是个很重要的东西，当然，规范不只是说大家统一用某种形式命名变量，方法等等，这只是对程序员而言的规范，如果这个划做横向规范的话，那么纵向规范就是面对客户的规范。对程序员的规范，我不想多说了，注释，变量，方法，文档。当然未必每个人都做到了这些。我想说的是对客户的规范问题。

对客户的规范有很多中，比如小细节 CS 系统中的 Anchor 怎么设置，Dock 怎么设置，如何让页面看起来更加让用户舒心，如何做焦点设置。大到如何给客户做培训，如何防止用户看到不友好页面，如何简化用户操作等等，这些都是属于规范性范畴。对于焦点设置，我有深刻体会，前段时间找工作，某网站输入搜索条件以后，按钮回车老是达不到焦点上去，非要我去移下鼠标点击，很不爽。

第二点，对于一个完善的架构，日志处理机制是必须做好的，日志处理不只是简单的说输出完成这么简单。首先，必须要通过配置控制在什么时候输出，在什么地方输出，如何输出，怎么记录，是记录数据库还是日志文件中。如何灵活让用户控制日志输出方式。

第三点，对于一个完善的架构，异常处理机制也是一个重点。异常怎么处理，如何记录，是记录到系统中，还是异常文件，还是数据库异常表，或者发给技术部邮件等等，如何做异常记录，在产生异常以后更容易让用户，技术人员看到异常产生的原因，这个是一个比较重要的模块。

第四点，对于一个完善的架构，配置文件是必须的，有些项目只是简单的对 web.confg 里加些配置，我认为这根本不够完善，对于配置而言，有很多需要配置的内容，比如系统连接哪种数据库，客户信息，再比如是否记录日志，异常等，是否允许用户注册等等灵活功能的控制完全可以在配置中实现。

第五点，对于一个完善的架构，如何做好权限是很重要的一块内容，比如权限如何控制，怎么处理用户，组，模块，部门等等之间的关系，工作流如何做，如何让权限与工作流做良好匹配，比如某审批人员出差了，如何处理其审批流程等等，虽然这点，我自己也在不断研究，但我想这一块非常重要。

第六点，对于一个完善的架构，流水号生成功能也相当重要，任何一种系统，不管是信息管理系统还是电子商务平台，一定都会要求按一定格式生成某套流水号，流水号也必须有灵活性，这点非常重要。

第七点，对于一个完善的架构，必须要有代码生成功能，比如基础业务类生成，实体类生成，最好可以控制数据库主外键关系等等，这样能减少程序员的很多无趣的工作量。这是我目前总结的几个重要点，另外当然包括多语言，多皮肤等等，我想这些目前来说还未必非常重要。

当我想到的时候我还会做一些补充。

系统架构设计师:如何成为软件架构师

2009 年上半年计算机技术与软件专业技术资格（水平）考试日期：2009 年 5 月 23、24 日。另外，部分考试科目从 2009 年上半年开始将采用新修编的考试大纲，具体见：

那么要成为架构师的途径似乎只有现在较为流行的软件学院和个人自我培养了。关于软件学院我接触过不少，其宗旨绝大部分都是造就（or 打造）企业需要的软件架构师（or 程序员 or 人才）。教师来源与企业、学员来源与企业、人才输送到企业是他们办学的手段。尽管各个如雨后春笋般出现的软件学院口号差不多，但恐怕大多只是为了圈钱卖学位了事...架构师不是通过理论学习可以搞出来的，不过不学习相关知识那肯定是不行的。参考软件企业架构师需求、结合目前架构师所需知识，总结架构师自我培养过程大致如下仅供参考：

1、架构师胚胎（程序员）学习的知识是语言基础、设计基础、通信基础等，应该在大学完成，内容包括 java、c、c++、uml、RUP、XML、socket 通信（通信协议）——学习搭建应用系统所必须的原材料。

2、架构师萌芽（高级程序员）学习分布式系统、组建等内容，可以在大学或第一年工作时间接触，包括分布式系统原理、ejb、corba、com/com+、webservice（研究生可以研究网络计算机、高性能并发处理等内容）

3、架构师幼苗（设计师）应该在掌握上述基础之上，结合实际项目经验，透彻领会应用设计模式，内容包括设计模式（c++版本、java 版本）、ejb 设计模式、J2EE 架构、UDDI、软件设计模式等。在此期间，最好能够了解软件工程在实际项目中的应用以及小组开发、团队管理。

系统架构设计师:软件架构师之路

2009 年上半年计算机技术与软件专业技术资格（水平）考试日期：2009 年 5 月 23、24 日。另外，部分考试科目从 2009 年上半年开始将采用新修编的考试大纲，具体见：　　架构师（Architecture）是目前很多软件企业最急需的人才，也是一个软件企业中薪水最高的技术人才。换句话说，架构师是企业的人力资本，与人力资源相比其能够通过架构、创新使企业获得新的产品、新的市场和新的技术体系。那么什么是架构师、架构师的作用、如何定位

一个架构师和如何成为一个架构师呢？这是许多企业、许多程序员朋友希望知道的或希望参与讨论的话题内容。

所谓架构师通俗的说就是设计师、画图员、结构设计者，这些定义范畴主要用在建筑学上很容易理解。小时候到河中玩耍，经常干的事就是造桥，步骤如下：1、在沙滩上画图；2、选择形状好看、大小适合的石头；3、搭建拱桥。其中我们挑出来画图的那位光 PP 小孩就是传说中的“架构师”了。

在软件工程中，架构师的作用在于三方面：

1、行业应用架构，行业架构师往往是行业专家，了解行业应用需求，其架构行为主要是将需求进行合理分析布局到应用模型中去，偏向于应用功能布局；

2、2、应用系统技术体系架构，技术架构师往往是技术高手中的高手，掌握各类技术体系结构、掌握应用设计模式，其架构行为考虑软件系统的高效性、复用性、安全性、可维护性、灵活性、跨平台性等；

3、3、规范架构师是通过多年磨砺或常年苦思顿悟后把某一类架构抽象成一套架构规范，当然也有专门研究规范而培养的规范架构师。他们的产物往往也分为应用规范和技术规范两类。

与建筑学类似，如果软件系统没有一个好的架构是不可能成为成功的软件系统的。没有图纸的建筑工地、没有设计的造桥工程都是不可以想象的混乱世界。建筑工程如是，软件工程中亦然！

由于国内合格、胜任的软件架构师极为少见，直接导致了我国民族软件产业水平的落后。在未来以信息产业为主导的社会，信息产业水平的低下将直接影响国家核心竞争力。究其原因，无企业非急功近利、个人缺乏引导。

企业的急功近利是有无法克服的原因的，那就是社会发展总体水平。“生存是第一位的，赚钱是第一位的”，多年来许多客户抱怨国内的软件公司无法信任、系统项目累做累败、公司越换越差，但因国外不可能给中国做应用系统项目还不得不找国内软件公司做。由于人月费用低、公司开发成本高，软件企业对于应用只能草草了事，拿钱走人（很多公司拿不到后期尾款）。这样的环境下，企业几乎无法投入更多资源培养自己的架构师，加上眼花缭乱的跳槽风气企业更是不愿投入……

系统架构设计师:使用分页和惰性加载

8.1.7.4 使用分页和惰性加载

在大多数情况下，您应该仅在需要时检索或显示数据。如果您的应用程序需要检索和显示大量信息，则 您应该考虑将数据分解到多个页面中，并且一次显示一页数据。这可以使用户界面具有更高的性能，因为它无须显示大量数据。此外，这可以提高应用程序的可用 性，因为用户不会同时面对大量数据，并且可以更加容易地导航以查找他或她需要的确切数据。例如，如果您的应用程序显示来自大型产品目录的产 品数据，则您可以按照字母顺序显示这些项，并且将所有以“A”开头的产品显示在一个页面上，将所有以“B”开头的产品显示在下一个页面上。然后，您可以让 用户直接导航到适当的页面，以便他或她无须浏览所有页面就可以获得他或她需要的数据。以这种方式将数据分页还使您可以根据需要获取后台的数据。例如，您可能只需要获取第一页信息以便显示并且让用户与其进行交互。然后，您可以获取后台中的、已经准备好供用户使用的下一页数据。该技术在与数据缓存技术结合使用时可能特别有效。您 还可以通过使用惰性加载技术来提高智能客户端应用程序的性能。您无须立即加载可能在将来某个时刻需要的数据或资源，而是可以根据需要加载它们。您可

以在构 建大型列表或树结构时使用惰性加载来提高用户界面的性能。在此情况下，您可以在用户需要看到数据时（例如，在用户展开树节点时）加载它。

17.2.2 应用实践

1)简单实用的架构（对网通的例子进行 Spring 解析） 2)加解密接口

18 系统架构师面对轻量级和重量级架构的选择（不限于 EJB 和 Spring） 18.1 没有系统级架构的开发 18.2 新浪网窄告发布系统分析 l)大并发量 2)高响应率

3)都需要什么（过程）

系统架构设计师:优化显示速度

8.1.7.5 优化显示速度

根据您用于显示用户界面控件和应用程序窗体的技术，您可以用多种不同的方式来优化应用程序的显示速度。当您的应用程序启动时，您应该考虑尽可能地显示简单的用户界面。这将减少启动时间，并且向用户呈现整洁且易于使用的用户界面。而且，您应该努力避免引用类以 及在启动时加载任何不会立刻需要的数据。这将减少应用程序和 .NET Framework 初始化时间，并且提高应用程序的显示速度。当您需要显示对话框或窗体时，您应该在它们做好显示准备之前使其保持隐藏状态，以便减少需要的绘制工作量。这将有助于确保窗体仅在初始化之后显示。如果您的应用程序具有的控件含有覆盖整个客户端表面区域的子控件，则您应该考虑将控件背景样式设置为不透明。这可以避免在发生每个绘制事件时重绘控件的背景。您可以通过使用 SetStyle 方法来设置控件的样式。使用 ControlsStyles.Opaque 枚举可以指定不透明控件样式。您应该避免任何不必要的控件重新绘制操作。一种方法是在设置控件的属性时隐藏控件。在 OnPaint 事件中具有复杂绘图代码的应用程序能够只重绘窗体的无效区域，而不是绘制整个窗体。OnPaint 事件的 PaintEventArgs 参数包含一个 ClipRect 结构，它指示窗口的哪个部分无效。这可以减少用户等待查看完整显示的时间。使用标准的绘图优化，例如，剪辑、双缓冲和 ClipRectangle。这还将通过防止对不可见或要求重绘的显示部分执行不必要的绘制操作，从而有助于改善智能客户端应用程序的显示性能。

如果您的显示包含动画或者经常更改某个显示元素，则您应该使用双缓冲或多缓冲，在绘制当前图像的过程中准备下一个图像。System.Windows.Forms 命名空间中的 ControlStyles 枚举适用于许多控件，并且 DoubleBuffer 成员可以帮助防止闪烁。启用 DoubleBuffer 样式将使您的控件绘制在离屏缓冲中完成，然后同时绘制到屏幕上。尽管这有助于防止闪烁，但它的确为分配的缓冲区使用了更多内存。

系统架构设计师辅导:IOC 技术应用

17.1 IOC 技术应用

1) 我们看看我们常用的配置文件应用（对象级的反转）

2) 在设计模式中，我们已经习惯一种思维编程方式： 接口驱动 3) 其实就是 javabean 的思想，注入和发射思想 17.1.1 IOC 的技术结构（面向技术经理和开发人员） 1) XML 设置

2) 配置性能和对象还原

3) 反射机制应用方式反射的代价 4) 可配性（替代很多设计模式） 5) 减少硬性编码
DriverManagerDataSource BasicDataSource
JndiObjectFactoryBean

系统架构设计师系:统架构师对技术的把握

3 系统架构师对技术的把握

1)新技术的更新，关注点和深度不同（技术风险） 2)对公司技术实力和技术方向的正确把握

3)不追求最新、不能把架构风险轻易带入系统。注意前期对新技术的测试。

4)设计模式解决了设计可扩展性问题，并不等于解决了性能问题，性能问题要进行瓶颈测试，并对设计和性能的矛盾进行权衡。非功能性问题（并发、网络、事务、操作系统、安全、稳定性、性能） 5)设计原则

系统架构设计师之路:详解面向过程

2009 年上半年计算机技术与软件专业技术资格（水平）考试日期：2009 年 5 月 23、24 日。另外，部分考试科目从 2009 年上半年开始将采用新修编的考试大纲，

1. 面向过程编程(OPP) 和面向对象编程(OOP)的关系

关于面向过程的编程(OPP)和面向对象的编程(OOP),给出这它们的定义的人很多，您可以从任何资料中找到很专业的解释，但以我的经验来看，讲的相对枯燥一点，不是很直观。除非您已经有了相当的积累，否则说起来还是比较费劲。

我是个老程序员出身，虽然现在的日常工作更多倾向了管理，但至今依然保持编码的习惯，这句话什么意思呢？我跟大家沟通应该没有问题。无论你是在重复我走过的路，或者已经走在了我的前面，大家都会有那么一段相同的经历，都会在思想层面上有一种理解和默契，所以我还是会尽量按照大多数人的常规思维写下去。

面向过程的编程(OPP)产生在前，面向对象的编程(OOP)产生在后，所以面向对象的编程(OOP)一定会继承前者的一些优点，并摒弃前者存在的一些缺点，这是符合人类进步的自然规律。两者在各自的发展和演变过程中，也一定会相互借鉴，互相融合，来吸收对方优点，从而出现在某些方面的趋同性，这些是必然的结果。即使两者有更多的相似点，也不会改变它们本质上的不同,因为它们的出发点就完全是两种截然不同的思维方式。关于两者的关系，我的观点是这样的：面向对象编程(OOP)在局部上一定是面向过程(OP)的，面向过程的编程(OPP)在整体上应该借鉴面向对象(OO)的思想。这一段说的的确很空洞，而且也一定会有引来争议，不过，我劝您还是在阅读了后面的内容之后，再来评判我观点的正确与否。

像 C++、C#、Java 等都是面向对象的语言，c,php(暂且这么说，因为 php4 以后就支持 OO)都是面向过程的语言，那么是不是我用 C++写的程序一定就是面向对象，用 c 写的程序一定就是面向过程呢？这种观点显然是没有真正吃透两者的区别。语言永远是一种工具，前辈们每创造出来的一种语言，都是你用来实现想法的利器。我觉得好多人用 C#,Java 写出来的代码，要是仔细看看，那实际就是用面向对象(OO)的语言写的面向过程(OP)的程序。

所以，即使给关羽一根木棍，给你一杆青龙偃月刀，他照样可以打得你满头是包。你就是扛着个偃月刀，也成不了关羽，因为你缺乏关羽最本质的东西---绝世武功。同样的道理，如果你没有领会 OO 思想，怎么可能写得出真正的 OO 程序呢？面向对象(OO)和面向过程(OP)绝对是两种截然不同的思维方式。

那是不是面向过程就不好，也没有存在的必要了？我从来没有这样说过。事实上，面向过程的编程(OPP)已经存在了几十年了，现在依然有很多人在使用。它的优点就是逻辑不复杂的情况下很容易理解，而且运行效率远高于面向对象（OO）编写的程序。所以，系统级的应用或准实时系统中，依然采用面向过程的编程(OPP)。当然，很多编程高手以及大师级的人物，他们由于对于系统整体的掌控能力很强，也喜欢使用面向过程的编程(OPP)，比如像 Apache,QMail,PostFix,ICE 等等这些比较经典的系统都是 OPP 的产物。

像 php 这些脚本语言，主要用于 web 开发，对于一些业务逻辑相对简单的系统，也常使用面向过程的编程(OPP)，这也是 php 无法跨入到企业级应用开发的原因之一，不过 php5 目前已经能够很好的支持 OO 了。

2.详解面向过程的编程(OPP)

在面向对象出现之前，我们采用的开发方法都是面向过程的编程(OPP)。面向过程的编程中最常用的一个分析方法是“功能分解”。我们会把用户需求先分解成模块，然后把模块分解成大的功能，再把大的功能分解成小的功能，整个需求就是按照这样的方式，最终分解成一个一个的函数。这种解决问题的方式称为“自顶向下”，原则是“先整体后局部”，“先大后小”，也有人喜欢使用“自下向上”的分析方式，先解决局部难点，逐步扩大开来，最后组合出来整个程序。其实，这两种方式殊路同归，最终都能解决问题，但一般情况下采用“自顶向下”的方式还是较为常见，因为这种方式最容易看清问题的本质。

我举个例子来说明面向过程的编程方式：用户需求：老板让我写个通用计算器。

最终用户就是老板，我作为程序员，任务就是写一个计算器程序。OK，很简单，以下就是用 C 语言完成的计算器：

假定程序的文件名为：main.c。

```
int main(int argc, char *argv[]){  //变量初始化
int nNum1,nNum2;  char cOpr;  int nResult;
nNum1 = nNum2 = 0;  cOpr = 0;  nResult = 0;  //输入数据
printf("Please input the first number:rn");  scanf("%d",&nNum1);
printf("Please input the operator:rn");  scanf("%s",&cOpr);
printf("Please input the second number:rn");  scanf("%d",&nNum2);
//计算结果  if( cOpr == ’+’ ){
nResult = nNum1 + nNum2;  }else if( cOpr == ’-’ ){
nResult = nNum1 - nNum2;  }else{
printf("Unknown operator!");  return -1;  }
//输出结果
printf("The result is %d!",nResult);  return 0;  }
```

抛开细节不讲，我想大多数人差不多都会这么实现吧，很清晰，很简单，充分体现了“简单就是美”的原则，面向过程的编程就是这样有条理的按照顺序来逐步实现用户需求。

凡是做过程序的人都知道，用户需求从来都不会是稳定的，最多只能够做到“相对稳定”。用户可能会随时提出加个功能，减个功能的要求，也可能会要求改动一下流程，程序员最烦的就是频繁地变动需求，尤其是程序已经写了大半了，但这种情况是永远无法避免的，也不能完全归罪到客户或者需求分析师。

以我们上面的代码为例，用户可能会提出类似的要求：
首先，你程序中实现了“加法”和“减法”，我还想让它也能计算“乘法”、“除法”。
其次，你现在的人机界面太简单了，我还想要个 Windows 计算器的界面或者 Mac 计算器的界面。

用户需求开始多了，我得琢磨琢磨该如何去写这段代码了。我今天加了“乘”“除”的运算，明天保不齐又得让我加个“平方”、“立方”的运算，这要是把所有的运算都穷尽了，怎么也得写个千八百行代码吧。还有，用户要求界面能够更换，还得写一大堆界面生成的代码，又得来个千八百行。以后，这么多代码堆在一起，怎么去维护，找个变量得半天，看懂了代码得半天，万一不小心改错了，还得调半天。另外，界面设计我也不擅长，得找个更专业的人来做，做完了之后再加进来吧。这个过程也就是“软件危机”产生的过程。伴随着软件广泛地应用于各个领域，软件开发的规模变得越来越大，复杂度越来越高，而其用户的需求越来越不稳定。
根据用户提出的两个需求，面向过程的编程该如何去应对呢？想大家都很清楚怎么去改。Very easy，把“计算”和“界面”分开做成两个独立的函数，封装到不同的文件中。 假定程序的文件名为：main.c。 #include "interface.h" #include "calculate.h"
int main(int argc, char *argv[]){ //变量初始化
int nNum1,nNum2; char cOpr; int nResult;
nNum1 = nNum2 = 0; cOpr = 0; nResult = 0; //输入数据
if(getParameters(&nNum1,&nNum2,&cOpr) == -1) return -1; //计算结果
if(calcMachine(nNum1,nNum2,cOpr,&nResult) == -1) return -1; //输出结果
printf("The result is %d!",nResult); return 0; }
interface.h:
int getParameters(int *nNum1,int * nNum2,char *cOpr); interface.c:
int getParameters(int *nNum1,int * nNum2,char *cOpr){ printf("Please input the first number:rn"); scanf("%d",nNum1);
printf("Please input the operator:rn"); scanf("%s",cOpr);
printf("Please input the second number:rn"); scanf("%d",nNum2);
return 0;
}
calculate.h:
int calcMachine(int nNum1,int nNum2,char cOpr, int *nResult); calculate.c:
int calcMachine(int nNum1,int nNum2,char cOpr,int *nResult){ if(cOpr == ’+’){
*nResult = nNum1 + nNum2; }else if(cOpr == ’-’){
*nResult = nNum1 - nNum2; }else{
printf("Unknown operator!"); return -1; };
return 0; }
面向过程的编程(OPP)就是将用户需求进行“功能分解”。把用户需求先分解成模块(.h,.c)，

再把模块(.h,.c)分解成大的功能(function)，然后把大的功能(function)分解成小的功能(function)，如此类推。

功能分解是一项很有技术含量的工作，它不仅需要分解者具有丰富的实战经验，而且需要科学的理论作为指导。如何分解，分解原则是什么，模块粒度多大合适？这些都是架构师的要考虑的问题，也是我们后面要着重讲的内容。

面向过程的编程(OPP)优点是程序顺序执行，流程清晰明了。它的缺点是主控程序承担了太多的任务，各个模块都需要主控程序进行控制和调度，主控和模块之间的承担的任务不均衡。

有的人把面向过程定义为：算法＋数据结构，我觉得也很准确。面向对象的编程中算法是核心，数据处于从属地位，数据随算法而流动。所以采用面向过程的方式进行编程，一般在动手之前，都要编写一份流程图或是数据流图。

系统架构师 J2EE 技术分析

12 系统架构师 J2EE 技术分析　12.1 J2EE 的企业级解决方案　12.2 J2EE 技术介绍

系统架构师 UML 如何赋予实施，用到实处

7 系统架构师 UML 如何赋予实施，用到实处

1)要让 UML 指引项目的开发而不是一个装饰品.如何同步你的设计文档和需求文档、类变化

2)以 CA 用例为例

3)作为交流的一种工具，不需要繁杂的 UML 图。　4)各种 UML 图的实际设计应用

系统架构师处理系统架构与人员组织的关系

14 系统架构师处理系统架构与人员组织的关系

1) 人员任务分配，应该说是架构体系和软件结构化的发展使开发人员的门槛降低，分分工更加明确，让不同层次的人员都可以参与进来，但对上层人员的要求越来越高。（万金油问题）

15 系统架构师的架构设计思想的平衡问题（不可成为纸上谈兵）　1) 不能一味追求技术　2) 不能不注重技术

3) 如何设计是令人满意的（受成本、技术力量、时间、需求的制约）没有最好，只有相对适合。

4) 设计模式一定要适用，并不是用到任何地方都可以见效。（要看具体的需求项目，网通内部管理系统的 DAO 问题、窄告 XML 解析展示的性能问题）　5) 面向对象的思想一定适合吗？

6) 恰当的系统设计（设计方案要根据具体的项目而定）商务安全平台（多屏蔽区、多数据库、数据安全）网通

系统架构师的数据库分析层次（角度）

4 系统架构师的数据库分析层次（角度）
1)数据库的选择要根据项 目本身的需求、成本等综合因素，而不是个人喜好
2)表关联问题，不拘泥数据库范式（主键无意义根据情况）
3)主键设计问题
4)系统级解决方案 DBA 协商解决。（索引字段问题、分区、系统参数、建表参数、数据仓库）
5)数据库记录容量斟酌并且不要忽略一个事实，数据库也是一个 socket 的网络连接
6)程序问题一定在系统级解决方案之后（存储过程、开发人员水平的提高）。
7)主干表的分析存储结构和类图结构应用问题。（迭代开发面临的问题）

系统架构师基本素质

1 引言
2 系统架构师在产品和项目的需求阶段的任务 2.1 国家文物总局文物平台同各省文物馆藏系统分析 2.1.1 原有软硬件设施分析 2.1.2 系统硬件需求分析 2.1.3 系统软件需求分析 2.1.4 系统潜在应用风险 2.1.5 系统性能需求
3 系统架构师对技术的把握
4 系统架构师的数据库分析层次（角度）
5 系统开发的前奏和组织问题（与技术无关的问题，但却很重要。不断修订交由项目经理具体实施）
6 系统架构师的开发语言和开发工具的制定（其实这是项目经理的事，不过需要你去力推执行，一切都要要有为以后维护、升级着想）7 系统架构师 UML 如何赋予实施，用到实处 8 架构设计理念解决的实际问题 8.1 最原始的方式（jsp ＝＝ java）
8.2 开发复杂度和时间、成本问题
9 架构师如何设计简单的中小项目或产品(网通内部办公系统)
10 架构师如何应用 MVC 前台分层结构去设计业务逻辑比较复杂的系统 10.1 设计原理 10.2 解决的问题 10.3 使用误区
11 分布式架构非中间件大中型或高性能设计方案 11.1 以新浪广告设计为例介绍
12 系统架构师 J2EE 技术分析 12.1 J2EE 的企业级解决方案
12.2 J2EE 技术介绍
13 分布式架构中间件大中型或高性能设计方案
13.1 分布式设计是前面所有知识点综合和改进版本 13.2 中间件为我们解决了非功能性问题 13.3 中间件的解决方案并不完美（要评估） 13.4 分布式设计的性能问题
14 系统架构师处理系统架构与人员组织的关系
15 系统架构师的架构设计思想的平衡问题（不可成为纸上谈兵） 16 系统架构师如何设计和使用 ORM（具体由技术经理督促实施） 17 系统架构师的框架另一个选择 Spring（轻量级的选择） 17.1 IOC 技术应用
17.1.1IOC 的技术结构（面向技术经理和开发人员） 17.1.2IOC 字符和原子数据注入（面向具
体编程人员） 17.1.3 模式应用 IOC 工厂模型（面向具体编程人员） 17.2 AOP 技术应用
17.2.1 为了使用拦截器要做三件事
17.2.2 应用实践

18 系统架构师面对轻量级和重量级架构的选择（不限于 EJB 和 Spring）
19 系统架构师如何看待测试技术（千万不要开发完毕才去测试）

20 系统架构师如何看待 Spring 对利用 AOP 和 IOC 这两个有价值的技术
20.1 Spring 事务的 AOP 应用思想
20.2 Spring 和 IOC 远程实现封装
21 技术经理和高级程序员如何理解架构开发的实施
22 技术经理和高级程序员如何进行 Spring 的 EJB 整合（项目经理和高级程序员关注库的问题）
23 技术经理和高级程序员如何利用 Spring 对 JDBC 支持
24 技术经理和高级程序员如何利用 Spring 同 Hibernate 集成（项目经理和高级程序员关注库的问题） 24.1 Hibernate
24.2 hibernate 配置 24.3 Hibernate 的使用问题
24.3.1 标示符
24.3.2 注意的问题
24.4 技术经理和高级程序员如何利用 Spring 整合 hibernate
25 技术经理和高级程序员如何利用 Spring 同 WEB 的整合（项目经理和高级程序员关注库的问题）
26 技术经理和高级程序员如何利用 Spring 整合 Struts（MVC 的一种）27 Spring 编程问题解答
系统架构师如何看待 Spring 对利用 AOP 和 IOC 这两个有价
值的技术

20 系统架构师如何看待 Spring 对利用 AOP 和 IOC 这两个有价值的技术
面向方面的编程，即 AOP，是一种编程技术，它允许程序员对横切关注点或横切典型的职责分界线的行为（例如日志和事务管理）进行模块化。AOP 的核心构造是方面，它将那些影响多个类的行为封装到可重用的模块中。 20.1Spring 事务的 AOP 应用思想 PlatformTransactionManager TransactionProxyFactoryBean
Spring 提供的外部声明式事务方案 20.2Spring 和 IOC 远程实现封装
l)LocalStatelessSessionProxyFactoryBean
2)SimpleRemoteStatelessSessionProxyFactoryBean
系统架构师如何设计和使用 ORM（具体由技术经理督促实
施）

16 系统架构师如何设计和使用 ORM（具体由技术经理督促实施）
1)JDBC 应用问题
2)持久化开发效率和应用效率的矛盾平衡
17 系统架构师的框架另一个选择 Spring（轻量级的选择） 1)时代的产物

2)集大成者，一个开发的骨架

系统架构师有哪些工作职责

系统构架师的职责

系统架构师的职责就是设计一个公司的基础架构，并提供关于怎样建立和维护系统的指 导方针。具体来讲，系统架构师的职责主要体现在以下几方面： 1 负责公司系统的架构设计、研发工作； 2 承担从业务向技术转换的桥梁作用；

3 协助项目经理制定项目计划和控制项目进度； 4 负责辅助并指导 SA 开展设计工作； 5 负责组织技术研究和攻关工作；

6 负责组织和管理公司内部的技术培训工作；

7 负责组织及带领公司内部员工研究与项目相关的新技术。

8 管理技术支撑团队并给项目、产品开发实施团队提供技术保障。

11 理解系统的业务需求，制定系统的整体框架(包括：技术框架和业务框架)

12 对系统框架相关技术和业务进行培训，指导开发人员开发。并解决系统开发、运行中出现的各种问题。 系统架构师的目的：

13 对系统的重用、扩展、安全、性能、伸缩性、简洁等做系统级的把握。

————系统架构师的工作在于针对不同的情况筛选出最优的技术解决方案，而不是沉在具体实现细节上。此外系统架构师是不可培养的，好的系统架构师也许不是一个优秀的程序员，但是不能不懂技术之间的差别，技术的发展趋势，采用该技术的当前成本和后继成本，该技术与具体应用的偶合程度，自己可以调配的资源状况，研发中可能会遇到的风险，如何回避风险。这些才是架构师需要考虑的主要内容。

另外，还必须注意，架构分为两种，

第一种是基础架构的设计规划，例如：OS，硬件，网络，各种应用服务器等等。 第二种是软件开发设计的架构师，他们负责规划程序的运行模式，层次结构，调用关系，规划具体的实现技术类型，甚至配合整个团队做好软件开发中的项目管理。

系统架构师与产品经理、项目经理、系统分析员

系统构架师与产品经理的关系及区别

产品经理通常是指负责产品设计的“专人”。一个优秀的理想的产品经理，应同时具备较高的商业素质和较强的技术背景。产品经理要有深厚的领域经验，也就是说，对该软件系统要应用到的业务领域非常之熟悉。比如，开发房地产销售软件的产品经理，应该对房地产公司的标准销售流程了如指掌，甚至比大多数销售人员还要清楚。如果开发的是通用产品，他/她还具备对市场、潜在客户需求的深刻洞察力。

那么，系统架构师与产品经理有什么不同呢?

我们不应该把二者混为一谈，这是不少论述和实践常犯的错误。我看来，如果把开发软件比作摄制电影，产品经理之于系统架构师，就正像编剧之于导演。产品经理虽然要有一定技术背景，但仍应属于“商业人士(business people)”，而系统架构师则肯定是一个技术专家。二者看待问题的立场、角度和出发点完全不同。

系统构架师与项目经理的关系及区别

软件项目经理是指对项目控制/管理，关注项目本身的进度、质量，分配、调动、协调、管理好人、财、物等资源的负责人。对于软件项目经理来讲，包括项目计划、进度跟踪/监控、质量保证、配置/发布/版本/变更管理、人员绩效评估等方面。优秀的项目经理需要的素质，并不仅在于会使用几种软件或是了解若干抽象的方法论原则，更重要的在于从大量项目实践中获得的宝贵经验，以及交流、协调、激励的能力，甚至还应具备某种个性魅力或领袖气质(Charisma)。

由此可见，项目经理和系统架构师在职责上有很大差异。混同这两个角色，往往也会导致低效、无序的开发。特别是，从性格因素上讲，单纯的技术人员倾向于忽视“人”的因素，而这正是管理活动的一个主要方面。另外，就像战争中的空军掩护(Air Cover)一样，专职的项目经理能够应付开发过程中大量的偶发事件和杂务，对于一个规模稍大的项目，这些杂务本身就能占用一个全职工作者的几乎全部时间。在一个项目中，推动项目发展的是系统构架师，而不是项目经理。项目经理的职责只是配合系统构架师，提供各个方面的支持。主要职责是与内外部沟通和管理资源(包括人)。系统构架师提出系统的总体构架，给出开发指导。一个项目中，项目经理的角色什么?如果他即使管理人员又是设计人员，则必须比别人强，能够有让别人服的东西。如果他只是项目管理人员，系统构架师有专门人员，就可以不用精通或者说了解 it 各个方面的知识，如果了解更好。另外，如果在一个项目没有人在技术构架上和开发指导上负全部责任，而是每个人都负责一快的架构、分析、设计、代码和实施等，最后肯定会失去管理。

系统构架师与系统分析员的关系及区别

系统分析员(System analyst)是指对系统开发中进行分析、设计和领导实施的人。一般意思上讲，系统分析员的水平将影响系统开发的质量，甚至成败。但在一个完善的系统开发队伍中，还需要有业务专家，技术专家和其他辅助人员。所以，系统分析员只是其中的角色之一。但我国许多的 IT 公司，一般只有系统分析员而没有技术专家。系统分析员固然是对特定系统进行分析、设计。所以他的任务、目标是明确的。他只是去执行任务，完成系统的最终设计。

系统架构师应该和系统分析员分开，但架构师必须具备系统分析员的所有能力，同时还应该具备设计员所没有的很多能力。 系统架构师是指导、检督系统分析员的工作，要求系统分析员按什么标准，什么工具，什么模式，什么技术去设计系统的。同时，系统架构师应该对系统分析员所提出的问题，碰到的难题及时地提出解决的方法。并检查、评审系统分析员的工作

系统架构师在产品和项目的需求阶段的任务

2 系统架构师在产品和项目的需求阶段的任务

不要到了开发后期才去考虑性能、安全和可维护问题、架构（设计方法）、硬件需求问题。

应该在需求设计完成开始要进行概要设计时就去考虑。

系统架构师在企业中发挥的作用

系统架构师该怎么来实现其“架构”企业的职能呢?尤其在设计企业 IT 策略时，该怎样体现架构师的价值?

这里以实例说明：

摩托罗拉的副总裁 Toby Redshaw 说，架构师是“IT 策略中的中枢”，而且这一角色对公司的影响确实非常大。当 Toby Reshaw 在 2001 年进入摩托罗拉并担任其策略暨架构副总裁时，他俨然一位购房者对一套摇摇欲坠的公寓进行估价一样。他并不是仅仅只作些表面上的修改，而是拟定了一个重建摩托罗拉整个基础结构的计划，这个计划可以彻底修整公司的 基础建设。就像一个建筑师设计一幢房子一样，Redshaw 拟出了一张技术构架蓝图，一座技术性的建筑，以便使被他称作“如意大利面条般错乱的应用程序，机器和管线”那些东西变得井然有序。他说，只要选择了正确的架构策略并用对了人，摩托就可以用比以前更快的速度生产出大量应用软件，而且可以减少维持重叠系统的费用。 Redshaw 说：“如果你连建筑架构都搞不好，就算你的石匠技术再高明，又有什么用?架构师是 IT 策略中的中枢。” 像 Redshaw 这样的系统架构师们在企业内部的影响力非常大。很久以来，虽然他们一直在信息技术部门担任重要职务，但是他们经常受委托提供全面概况分析，并提出一些关于如何遵照标准执行这些任务的建议，而这些对日常运作的影响极其有限。今天，随着各公司都在寻找重建他们的 IT 系统，使其更能有效节省成本，更灵活的方法，架构师愈来愈被看作是至关重要的因素。

一个定义明确的架构的目标在于降低运行复杂的运算系统的费用。一个公司可以采用一种特定的数据库配置，如微软的数据库，进而将系统标准化，而不需要让公司的每个部门安装它们自己所需要的数据库服务器。

Express 的技术架构副总裁 Andy Miller 说：“如果你没有一项强有力的架构策略，人人各行其是，最后以得到六种服务器和软件平台而告终，你的系统变成了大杂烩，而那将使你的费用激增。”把架构师独立出来有很多好处，比如系统的整体把握，质量上的保障，技术上的先进性，架构的灵活性，高效性，还可有效地降低成本。试想，1 个月薪 1w 的架构师+10 个月薪 5k 的工程师，肯定比 11 个月薪 6k 的高级工程师效果要好。一般来说，级别越高的架构师，经验更丰富，争相聘请的人也多，他们也是与公司全部的 IT 策略密切相关的专业人员。