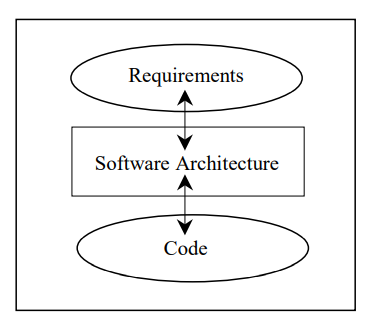
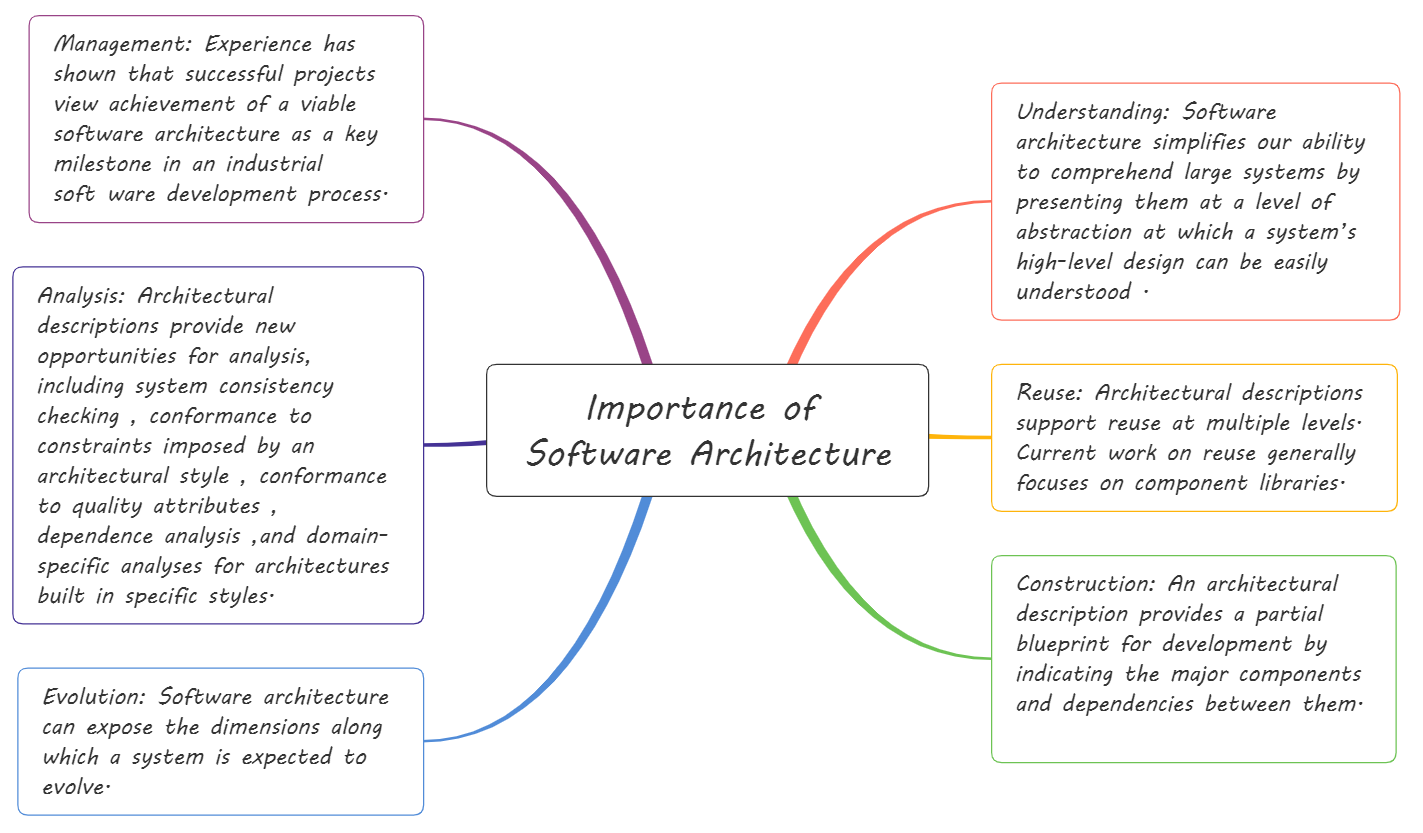
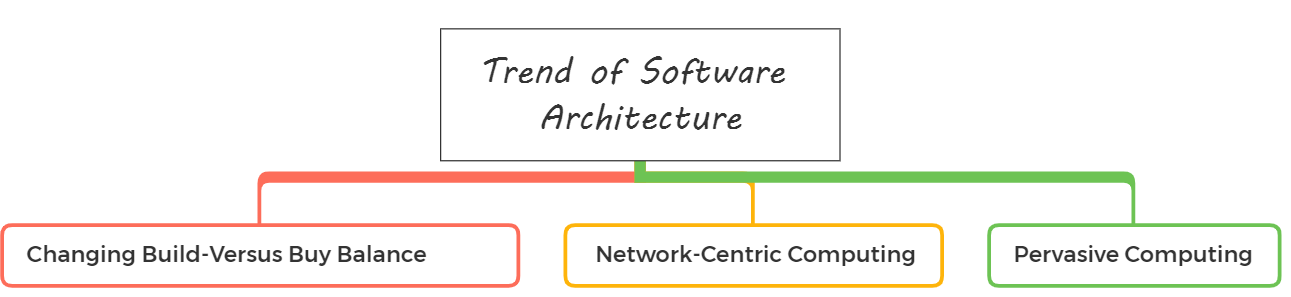
1. 软件体系结构设计的一个核心问题是能否使用重复的体系结构模式，即能够达到体系结构级的复用。软件体系结构是连接需求和代码的桥梁：



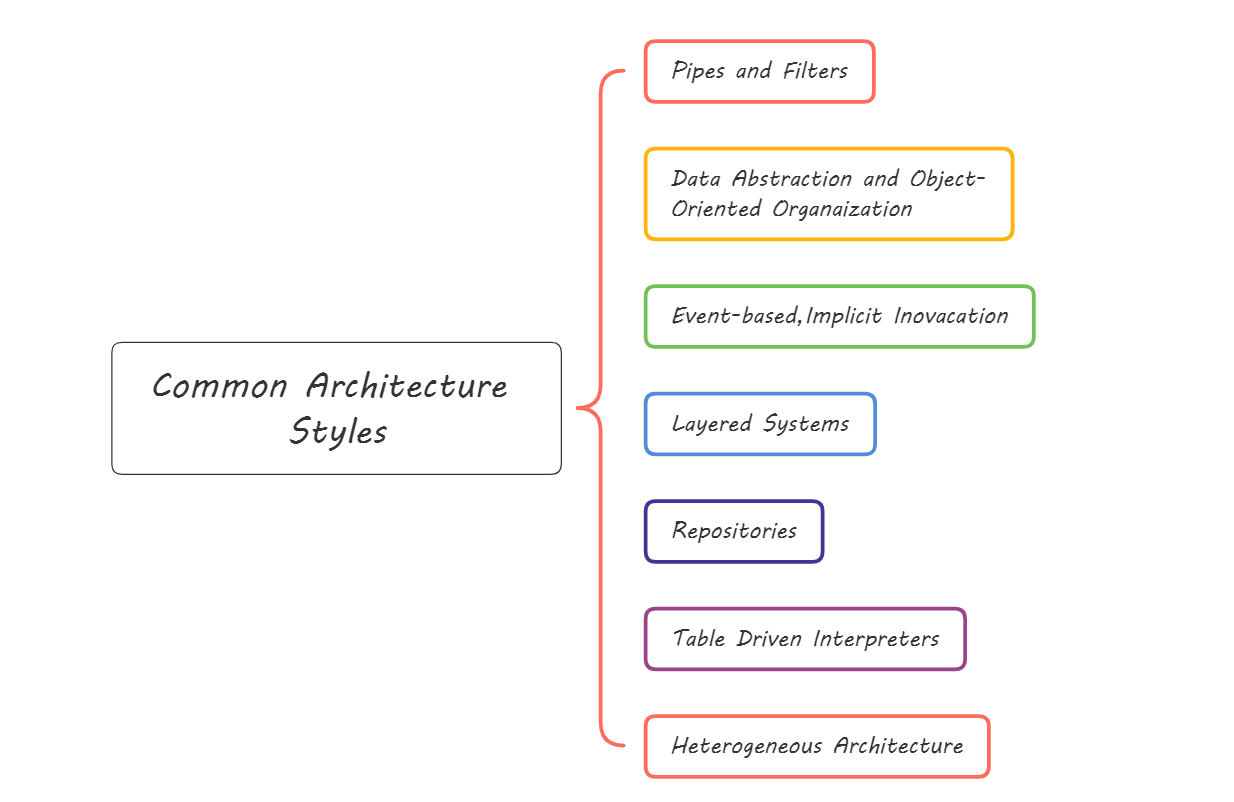
1. 软件体系结构的重要性



1. 软件体系结构未来发展趋势及方向



1. 软件体系结构从架构风格上可以分为以下几类：



* 管道和过滤器风格

在管道/过滤器风格的软件体系结构中，每个构件都有一组输入和输出，构件读输入的数据流，经过内部处理，然后产生输出数据流。这个过程通常通过对输入流的变换及增量计算来完成，所以在输入被完全消费之前，输出便产生了。因此，这里的构件被称为过滤器，这种风格的连接件就像是数据流传输的管道，将一个过滤器的输出传到另一过滤器的输入。此风格特别重要的 过滤器必须是独立的实体，它不能与其它的过滤器共享数据，而且一个过滤器不知道它上游和下游的标识。一个管道/过滤器网络输出的正确性并不依赖于过滤器进 行增量计算过程的顺序。

管道/过滤器风格的软件体系结构具有许多很好的特点：

(1)使得软构件具有良好的隐蔽性和高内聚、低耦合的特点；

(2)允许设计者将整个系统的输入/输出行为看成是多个过滤器的行为的简单合成；

(3)支持软件重用。重要提供适合在两个过滤器之间传送的数据，任何两个过滤器都可被连接起来；

(4)系统维护和增强系统性能简单。新的过滤器可以添加到现有系统中来；旧的可以被改进的过滤器替换掉；

(5)允许对一些如吞吐量、死锁等属性的分析；

(6)支持并行执行。每个过滤器是作为一个单独的任务完成，因此可与其它任务并行执行。

但是，这样的系统也存在着若干不利因素。

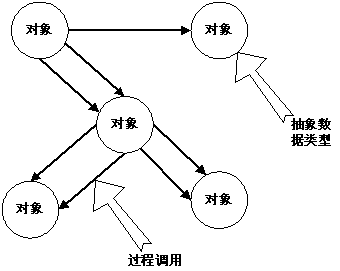
(1)通常导致进程成为批处理的结构。这是因为虽然过滤器可增量式地处理数据，但它们是独立的，所以设计者必须将每个过滤器看成一个完整的从输入到输出的转换。

(2)不适合处理交互的应用。当需要增量地显示改变时，这个问题尤为严重。

(3)因为在数据传输上没有通用的标准，每个过滤器都增加了解析和合成数据的工作，这样就导致了系统性能下降，并增加了编写过滤器的复杂性

* 数据抽象与面向对象风格

抽象数据类型概念对软件系统有着重要作用，目前软件界已普遍转向使用面向对象系统。这种风格建立在数据抽象和面向对象的基础上，数据的表示方法和它们的相应操作封装在一个抽象数据类型或对象中。这种风格的构件是对象，或者说是抽象数据类型的实例。对象是一种被称作管理者的构件，因为它负责保持资源的完整性。对象是通过函数和过程的调用来交互的。



面向对象的系统的优点：

(1) 因为对象对其它对象隐藏它的表示，所以可以改变一个对象的表示，而不影响其它的对象。

(2) 设计者可将一些数据存取操作的问题分解成一些交互的代理程序的集合。

但是，面向对象的系统也存在着某些问题：

(1)为了使一个对象和另一个对象通过过程调用等进行交互，必须知道对象的标识。只要一个对象的标识改变了，就必须修改所有其他明确调用它的对象。

(2)必须修改所有显式调用它的其它对象，并消除由此带来的一些副作用。例如，如果A使用了对象B，C也使用了对象B，那么，C对B的使用所造成的对A的影响可能是料想不到的。

* 基于事件的隐式调用风格

基于事件的隐式调用风格的思想是构件不直接调用一个过程，而是触发或广播一个或多个事件。系统中的其它构件中的过程在一个或多个事件中注册，当一个事件被触发，系统自动调用在这个事件中注册的所有过程，这样，一个事件的触发就导致了另一模块中的过程的调用。

从体系结构上说，这种风格的构件是一些模块，这些模块既可以是一些过程，又可以是一些事件的集合。过程可以用通用的方式调用，也可以在系统事件中注册一些过程，当发生这些事件时，过程被调用。

基于事件的隐式调用风格的主要特点是事件的触发者并不知道哪些构件会被这些事件影响。这样不能假定构件的处理顺序，甚至不知道哪些过程会被调用，因此，许多隐式调用的系统也包含显式调用作为构件交互的补充形式。

支持基于事件的隐式调用的应用系统很多。例如，在编程环境中用于集成各种工具，在数据库管理系统中确保数据的一致性约束，在用户界面系统中管理数据，以及在编辑器中支持语法检查。例如在某系 统中，编辑器和变量监视器可以登记相应Debugger的断点事件。当Debugger在断点处停下时，它声明该事件，由系统自动调用处理程序，如编辑程 序可以卷屏到断点，变量监视器刷新变量数值。而Debugger本身只声明事件，并不关心哪些过程会启动，也不关心这些过程做什么处理。

隐式调用系统的主要优点有：

(1)为软件重用提供了强大的支持。当需要将一个构件加入现存系统中时，只需将它注册到系统的事件中。

(2)为改进系统带来了方便。当用一个构件代替另一个构件时，不会影响到其它构件的接口。

隐式调用系统的主要缺点有：

(1)构件放弃了对系统计算的控制。一个构件触发一个事件时，不能确定其它构件是否会响应它。而且即使它知道事件注册了哪些构件的构成，它也不能保证这些过程被调用的顺序。

(2)数据交换的问题。有时数据可被一个事件传递，但另一些情况下，基于事件的系统必须依靠一个共享的仓库进行交互。在这些情况下，全局性能和资源管理便成了问题。

(3)既然过程的语义必须依赖于被触发事件的上下文约束，关于正确性的推理存在问题。

* 层次系统风格

层次系统组织成一个层次结构，每一层为上层服务，并作为下层客户。在一些层次系统中，除了一些精心挑选的输出函数外，内部的层只对相邻的层可见。这样的系统中构件在一些层实现了虚拟机(在另一些层次系统中层是部分不透明的)。连接件通过决定层间如何交互的协议来定义，拓扑约束包括对相邻层间交互的约束。

这种风格支持基于可增加抽象层的设计。这样，允许将一个复杂问题分解成一个增量步骤序列的实现。由于每一层最多只影响两层，同时只要给相邻层提供相同的接口，允许每层用不同的方法实现，同样为软件重用提供了强大的支持。

层次系统有许多可取的属性：

(1)支持基于抽象程度递增的系统设计，使设计者可以把一个复杂系统按递增的步骤进行分解；

(2)支持功能增强，因为每一层至多和相邻的上下层交互，因此功能的改变最多影响相邻的上下层；

(3)支持重用。只要提供的服务接口定义不变，同一层的不同实现可以交换使用。这样，就可以定义一组标准的接口，而允许各种不同的实现方法。

但是，层次系统也有其不足之处：

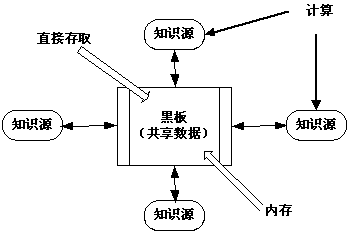
(1)并不是每个系统都可以很容易地划分为分层的模式，甚至即使一个系统的逻辑结构是层次化的，出于对系统性能的考虑，系统设计师不得不把一些低级或高级的功能综合起来；

(2)很难找到一个合适的、正确的层次抽象方法。

* 仓库风格

在仓库风格中，有两种不同的构件：中央数据结构说明当前状态，独立构件在中央数据存贮上执行，仓库与外构件间的相互作用在系统中会有大的变化。

控制原则的选取产生两个主要的子类。若输入流中某类时间触发进程执行的选择，则仓库是一传统型数据库；另一方面，若中央数据结构的当前状态触发进程执行的选择，则仓库是一黑板系统。



黑板系统主要由三部分组成：

(1)知识源。知识源中包含独立的、与应用程序相关的知识，知识源之间不直接进行通讯，它们之间的交互只通过黑板来完成。

(2)黑板数据结构。黑板数据是按照与应用程序相关的层次来组织的解决问题的数据，知识源通过不断地改变黑板数据来解决问题。

(3)控制。控制完全由黑板的状态驱动，黑板状态的改变决定使用的特定知识。

* C2风格

C2体系结构风格可以概括为：通过连接件绑定在一起的按照一组规则运作的并行构件网络。C2风格中的系统组织规则如下：

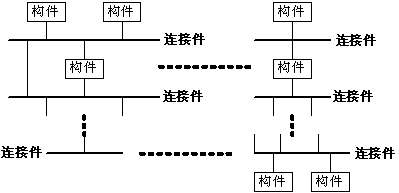
(1)系统中的构件和连接件都有一个顶部和一个底部；

(2)构件的顶部应连接到某连接件的底部，构件的底部则应连接到某连接件的顶部，而构件与构件之间的直接连接是不允许的；

(3)一个连接件可以和任意数目的其它构件和连接件连接；

(4)当两个连接件进行直接连接时，必须由其中一个的底部到另一个的顶部。

如下图所示，图中构件与连接件之间的连接体现了C2风格中构建系统的规则。



C2风格是最常用的一种软件体系结构风格。从C2风格的组织规则和结构图中，我们可以得出，C2风格具有以下特点：

(1)系统中的构件可实现应用需求，并能将任意复杂度的功能封装在一起；

(2)所有构件之间的通讯是通过以连接件为中介的异步消息交换机制来实现的；

(3)构件相对独立，构件之间依赖性较少。系统中不存在某些构件将在同一地址空间内执行，或某些构件共享特定控制线程之类的相关性假设。

* 三层C/S结构

三层C/S结构是将应用功能分成表示层、功能层和数据层三部分。其解决方案是：对这三层进行明确分割，并在逻辑上使其独立。原来的数据层作为DBMS已经独立出来，所以关键是要将表示层和功能层分离成各自独立的程序，并且还要使这两层间的接口简洁明了。三层C/S结构各层间的通信效率若不高，即使分配给各层的硬件能力很强，其作为整体来说也达不到所要求的性能。此外，设计时必须慎重考虑三层间的通信方法、通信频度及数据量。这和提高各层的独立性一样是三层C/S结构的关键问题。

**三层C/S的功能：**

1.表示层

表示层是应用的用户接口部分，它担负着用户与应用间的对话功能。它用于检查用户从键盘等输入的数据，显示应用输出的数据。为使用户能直观地进行操作，一 般要使用图形用户接口(GUI)，操作简单、易学易用。在变更用户接口时，只需改写显示控制和数据检查程序，而不影响其他两层。检查的内容也只限于数据的 形式和值的范围，不包括有关业务本身的处理逻辑。

图形界面的结构是不固定的，这便于以后能灵活地进行变更。例如，在一个窗口中不是放入几个功能，而是按功能分割窗口，以便使每个窗口的功能简洁单纯。在这层的程序开发中主要是使用可视化编程工具。

2. 功能层

功能层相当于应用的本体，它是将具体的业务处理逻辑地编入程序中。例如，在制作订购合同的时要计算合同金额，按照定好的格式配置数据、打印订购合同，而 处理所需的数据则要从表示层或数据层取得。表示层和功能层之间的数据交往要尽可能简洁。例如，用户检索数据时，要设法将有关检索要求的信息一次传送给功能 层(参见图2)，而由功能层处理过的检索结果数据也一次传送给表示层。在应用设计中，一定要避免进行一次业务处理，在表示层和功能层间进行多几次数据交换 的笨拙设计。

通常，在功能层中包含有：确认用户对应用和数据库存取权限的功能以及记录系统处理日志的功能。这层的程序多半是用可视化编程工具开发的，也有使用COBOL和C语言的。

3. 数据层

数据层就是DBMS，负责管理对数据库数据的读写。DBMS必须能迅速执行大量数据的更新和检索。现在的主流是关系数据库管理系统(RDBMS)。因此，一般从功能层传送到数据层的要求大都使用SQL语言。

**三层C/S结构的优点：**

1。 具有灵活的硬件系统构成

对于各个层可以选择与其处理负荷和处理特性相适应的硬件。这是一个与系统可缩放性直接相关的问题。例如，最初用一台Unix工作站作为服务器，将数据层 和功能层都配置在这台服务器上。随着业务的发展，用户数和数据量逐渐增加，这时就可以将Unix工作站作为功能层的专用服务器，另外追加一台专用于数据层 的服务器。若业务进一步扩大，用户数进一步增加，则可以继续增加功能层的服务器数目，用以分割数据库。清晰、合理地分割三层结构并使其独立，可以使系统构 成的变更非常简单。因此，被分成三层的应用基本上不需要修正。

2。 提高程序的可维护性

三层C/S结构中，应用的各层可以并行开发，各层也可以选择各自最适合的开发语言。

3。 利于变更和维护应用技术规范

因为是按层分割功能，所以各个程序的处理逻辑变得十分简单。

4。 进行严密的安全管理

越关键的应用，用户的识别和存取权限设定愈重要。在三层C/S结构中，识别用户的机构是按层来构筑的，对应用和数据的存取权限也可以按层进行设定。例如，即使外部的入侵者突破了表示层的安全防线，若在功能层中备有另外的安全机构，系统也可以阻止入侵者进入其他部分。

此外，系统管理简单，可支持异种数据库，有很高的可用性。

C/S和B/S 的优缺点比较：

C/S和B/S是当今世界开发模式技术架构的两大主流技术。C/S是美国 Borland公司最早研发，B/S是美国微软公司研发。

1、C/S架构软件的优势与劣势

（1）、应用服务器运行数据负荷较轻。

最简单的C/S体系结构的数据库应用由两部分组成，即客户应用程序和数据库服务器程序。二者可分别称为前台程序与后台程序。运行数据库服务器程序 的机器，也称为应用服务器。一旦服务器程序被启动，就随时等待响应客户程序发来的请求；客户应用程序运行在用户自己的电脑上，对应于数据库服务器，可称为 客户电脑，当需要对数据库中的数据进行任何操作时，客户程序就自动地寻找服务器程序，并向其发出请求，服务器程序根据预定的规则应答，送回结果，应用 服务器运行数据负荷较轻。

（2）、数据的储存管理功能较为透明。

在数据库应用中，数据的储存管理功能，是由服务器程序和客户应用程序分别独立进行的，前台应用可以违反的规则，并且通常把那些不同的（不管是已知 还是未知的）运行数据，在服务器程序中不集中实现，例如访问者的权限，编号可以重复、必须有客户才能建立定单这样的规则。所有这些，对于工作在前台程序上 的最终用户，是“透明”的，他们无须过问（通常也无法干涉）背后的过程，就可以完成自己的一切工作。在客户服务器架构的应用中，前台程序不是非常“瘦小 ”，麻烦的事情都交给了服务器和网络。在C/S体系的下，数据库不能真正成为公共、专业化的仓库，它受到独立的专门管理。

（3）、C/S架构的劣势是高昂的维护成本且投资大。

首先，采用C/S架构，要选择适当的数据库平台来实现数据库数据的真正“统一”，使分布于两地的数据同步完全交由数据库系统去管理，但逻辑上两地 的操作者要直接访问同一个数据库才能有效实现，有这样一些问题，如果需要建立“实时”的数据同步，就必须在两地间建立实时的通讯连接，保持两地的数据库服 务器在线运行，网络管理工作人员既要对服务器维护管理，又要对客户端维护和管理，这需要高昂的投资和复杂的技术支持，维护成本很高，维护任务量大。

其次，传统的C/S结构的软件需要针对不同的操作系统系统开发不同版本的软件，由于产品的更新换代十分快，代价高和低效率已经不适应工作需要。在JAVA这样的跨平台语言出现之后，B/S架构更是猛烈冲击C/S，并对其形成威胁和挑战。

2、B/S架构软件的优势与劣势

（1）、维护和升级方式简单。

目前，软件系统的改进和升级越来越频繁，B/S架构的产品明显体现着更为方便的特性。对一个稍微大一点单位来说，系统管理人员如果需要在几百甚至 上千部电脑之间来回奔跑，效率和工作量是可想而知的，但B/S架构的软件只需要管理服务器就行了，所有的客户端只是浏览器，根本不需要做任何的维护。无论 用户的规模有多大，有多少分支机构都不会增加任何维护升级的工作量，所有的操作只需要针对服务器进行；如果是异地，只需要把服务器连接专网即可，实现远程 维护、升级和共享。所以客户机越来越“瘦”，而服务器越来越“胖”是将来信息化发展的主流方向。今后，软件升级和维护会越来越容易，而使用起来会越来越简单，这对用户人力、物力、时间、费用的节省是显而易见的，惊人的。因此，维护和升级革命的方式是“瘦”客户机，“胖”服务器。

（2）、成本降低，选择更多。

大家都知道windows在桌面电脑上几乎一统天下，浏览器成为了标准配置，但在服务器操作系统上windows并不是处于绝对的统治地位。现在的趋势是凡使用B/S架构的应用管理软件，只需安装在Linux服务器上即可，而且安全性高。所以服务器操作系统的选择是很多的，不管选用那种操作系统都 可以让大部分人使用windows作为桌面操作系统电脑不受影响，这就使的最流行免费的Linux操作系统快速发展起来，Linux除了操作系统是免费的 以外，连数据库也是免费的，这种选择非常盛行。

（3）、应用服务器运行数据负荷较重。

由于B/S架构管理软件只安装在服务器端（Server）上，网络管理人员只需要管理服务器就行了，用户界面主要事务逻辑在服务器 （Server）端完全通过WWW浏览器实现，极少部分事务逻辑在前端（Browser）实现，所有的客户端只有浏览器，网络管理人员只需要做硬件维护。 但是，应用服务器运行数据负荷较重，一旦发生服务器“崩溃”等问题，后果不堪设想。因此，许多单位都备有数据库存储服务器，以防万一。

C/S 与 B/S 区别：

Client/Server是建立在局域网的基础上的，Browser/Server是建立在广域网的基础上的。

（1）硬件环境不同：

C/S 一般建立在专用的网络上， 小范围里的网络环境， 局域网之间再通过专门服务器提供连接和数据交换服务。

B/S 建立在广域网之上的， 不必是专门的网络硬件环境，例如电话上网， 租用设备， 信息自己管理， 有比C/S更强的适应范围， 一般只要有操作系统和浏览器就行。

（2）、对安全要求不同

C/S 一般面向相对固定的用户群， 对信息安全的控制能力很强。 一般高度机密的信息系统采用C/S 结构适宜，可以通过B/S发布部分可公开信息。

B/S 建立在广域网之上， 对安全的控制能力相对弱， 面向是不可知的用户群。

（3）、对程序架构不同

C/S 程序可以更加注重流程，可以对权限多层次校验，对系统运行速度可以较少考虑。

B/S 对安全以及访问速度的多重的考虑， 建立在需要更加优化的基础之上。 比C/S有更高的要求，B/S结构的程序架构是发展的趋势，从MS的。Net系列的BizTalk 2000 Exchange 2000等，全面支持网络的构件搭建的系统。SUN和IBM推的JavaBean构件技术等，使B/S更加成熟。

（4）、软件重用不同

C/S 程序可以不可避免的整体性考虑，构件的重用性不如在B/S要求下的构件的重用性好。

B/S 对的多重结构，要求构件相对独立的功能。能够相对较好的重用。就如买来的餐桌可以再利用，而不是做在墙上的石头桌子。

（5）、系统维护不同

系统维护是软件生存周期中，开销大，相当重要。C/S 程序由于整体性，必须整体考察，处理出现的问题以及系统升级难，可能是再做一个全新的系统。 B/S 构件组成方面构件个别的更换，实现系统的无缝升级。系统维护开销减到最小，用户从网上自己下载安装就可以实现升级。

（6）、处理问题不同

C/S 程序可以处理用户面固定，并且在相同区域， 安全要求高的需求，与操作系统相关， 应该都是相同的系统。 B/S 建立在广域网上，面向不同的用户群，分散地域， 这是C/S无法作到的，与操作系统平台关系最小。

（7）、用户接口不同

C/S 多是建立在Window平台上，表现方法有限，对程序员普遍要求较高。 B/S 建立在浏览器上， 有更加丰富和生动的表现方式与用户交流， 并且大部分难度减低，降低开发成本。

（8）、信息流不同

C/S 程序一般是典型的中央集权的机械式处理，交互性相对低。B/S 信息流向可变化， B－B、 B－C、 B－G等信息流向的变化，更像交易中心。

* 基于层次消息总线的架构风格

JB/HMB风格的基本特征

目前对软件体系结构的研究集中在以下方面：各种体系结构风格的汇编和总结、体系结

构描述语言(architectural description languages，简称ADLS)、体系结构的形式化基础、体系结构分析技术、基于体系结构的软件开发、体系结构恢复和再工程、支持体系结构设计的工具和环境及特定领域的软件体系结构等。 青鸟工程在“九五”期间，对基于构件构架模式的软件工业化生产技术进行了研究，并实现了青鸟软件生产线系统151。以青鸟软件生产线的实践为背景，提出了基于层次消息总线的软件体系结构(Jade bird hierarchical message bus based style，以下简称JB/HMB风格)，设计了相应的体系结构描述语言，开发了支持软件体系结构设计的辅助工具集，并研究了采用JB/HMB风格进行应用系统开发的过程框架。

JB/HMB风格的提出基于以下的实际背景：

(1) 随着计算机网络技术的发展，特别是分布式构件技术的日渐成熟和构件互操作标准的出现，如CORBA，DCOM和EJB等，加速了基于分布式构件的软件开发趋势，具有分布和并发特点的软件系统已成为一种普遍的应用需求。

(2) 基于事件驱动的编程模式已在图形用户界面程序设计中获得广泛应用。在此之前的

程序设计中，通常使用一个大的分支语句(switch Statement)控制程序的转移，对不同的输人情况分别进行处理，程序结构不甚清晰。基于事件驱动的编程模式在对多个不同事件响应的情况下，系统自动调用相应的处理函数，程序具有清晰的结构。

(3) 计算机硬件体系结构和总线的概念为软件体系结构的研究提供了很好的借鉴和启发，

REST架构风格

首先，REST是Web自身的架构风格。REST也是Web之所以取得成功的技术架构方面因素的总结。REST是世界上最成功的分布式应用架构风格。它是为运行在互联网环境 的 分布式超媒体系统量身定制的。互联网环境与企业内网环境有非常大的差别，最主要的差别是两个方面：

可伸缩性需求无法控制：并发访问量可能会暴涨，也可能会暴跌。

安全性需求无法控制：无法控制客户端发来的请求的格式，很可能会是恶意的请求。而所谓的“超媒体系统”，即，使用了超文本的系统。可以把“超媒体”理解为超文本+媒体内容。