基本概念

体系结构的设计就是设计组件与组件之间的联系。软件体系结构是具有一定形式的结构化元素，即构件的集合，包括处理构件、数据构件和连接构件。处理构件负责对数据进行加工，数据构件是被加工的信息，连接构件把体系结构的不同部分组合连接起来。

对于软件体系结构没有统一明确的定义，但是从结构模式角度来说，软件体系结构是一个程序或，系统的构件的组织结构，是它们之间的关联关系及其支配系统设计和演变的原则和方针。一个软件体系结构是由一组构件、连接件和它们之间的约束组成的，同时还包括系统需求和结构元素之间的对应关系。

连接件是构件的黏合剂，它也是一组对象。它把不同的构件连接起来，形成体系结构的一部分，一般表现为框架式对象或转换式对象(调用远程构件资源)。约束一般为对象连接时的规则，或指明构件连接 的势态和条件。例如 ,上层构件可要求下层构件的服务，反之不行;；两个对象不得递归地发消息；代码复制迁移的一致性约束；以及在什么条件下此种连接无效 等等。

可以将软件体系结构抽象地描述如下：软件体系结构 = 构件 + 连接件 + 约束。

几种常见的软件体系结构及特点

管道和过滤器

把系统任务分成为几个独立的功能模块。这些功能模块采用通过系统的数据流连接。

在管道-过滤器风格下，每个功能模块都有一组输入和输出。功能模块称作过滤器。功能模块间的连接可以看作输入、输出数据流之间的通路，所以称作管道。

管道-过滤器风格的特性之一在于过滤器的相对独立性，即过滤器独立完成自身功能，相互之间无需进行状态交互。

优点：

设计者可以将整个系统的输入、输出特性简单的理解为各个过滤器功能的合成。

基于管道-过滤器风格的系统具有较强的可维护性和可扩展性。

支持一些特定的分析，如吞吐量计算和死锁检测等。

管道-过滤器风格具有并发性。

缺点：

管道和过滤器常常导致批处理方式；在维护两个分离但相关的数据流时，有一定的困难；可能对数据传输标准有一定的要求，这就导致了为每一个过滤器进行数据语法分析和编码的额外工作，同时，在编写过滤器时增加了复杂性，降低了性能。

层次结构：

分层结构采取层次化的组织方法 ,每一层向上一层提供服务，并利用下一层的服务。比较常见的有 TCP/ IP 协议，OSI 七层结构等。在 OSI 形成的过程中 ，把通信协议有关的不同任务交给不同的层来完成，使每一层的函数既得到良好的定义又与其他层分开。服务的定义是从底层向顶层形成的。

优点：

层次结构风格支持系统设计过程中的逐级抽象。

基于层次结构风格的系统具有较好的可扩展性。

层次结构风格支持软件复用。

缺点：

并不是每个系统都可以很容易地划分为分层的模式

很难找到一个合适的、正确的层次抽象方法。

C/S 风格

客户机/服务器（C/S）体系结构是基于资源不对等，且为实现共享而提出来的。C/S 体系结构有三个主要组成部：数据库服务器、客户应用程序和网络。

优点：

界面和操作可以很丰富，安全性高，响应速度快。

缺点：

适用面窄，用户群固定，维护成本高。

B/S 风格

B/S 体系结构主要利用不断成熟的 WWW 浏览器技术，特别是浏览器嵌入的多种脚本语言。B/S 体系结构有三个主要组成部分：浏览器，Web 服务器，数据库服务器。

优点：

维护和升级方式简单，交互性较强。

缺点：

在速度和安全性上需要花费巨大的设计成本，通常需要刷新页面（Ajax 等可以一定程度缓解该问题），通信开销大。

面向对象

面向对象的程序设计其实是一种观念，主要是完成对象的封装、继承、多态等特性。在这种结构中，数据的表示和与之相连的属性的基本构造都被封装在一个抽象的数据类型或对象中。这种结构的部件就是对象，或者说是抽象数据类型的实例。对象之间通过函数和过程调用发生相互作用。这里所说的对象就是 C++，JAVA 中的类，都是封装了特定功能的属性和事件。

优点：

对象的隐藏并不影响它的使用者，更改它的实现，不会对系统造成任何危害。

将它们所操纵的数据与访问这些数据的操作绑在一起，是设计者易于将问题分解成相关对象之间的相互作用。

缺点：

为了实现某个对象与其他对象相互作用，必须知道对象的标志。这与管道形成了鲜明的对比；会产生连锁反应。比如 A 调 B，C 调 B ，那么 A 对 B 的调用可能会对 C 造成影响。