新世纪高等学校规划教材 · 软件工程系列

软件系统设计与体系结构

张晓明◎主编

北京师范大学出版集团
BEIJING NORMAL UNIVERSITY PUBLISHING GROUP
北京师范大学出版社

内 容 简 介

本书主要参照国际标准SWEBOK V3.0的软件设计主题进行编写，并引入最新的云计算软件架构内容。全书共分9章，包括概述、经典软件体系结构风格、软件体系结构建模、软件设计模式、软件设计的关键问题、客户机—服务器软件架构设计、数据集成系统设计、云计算体系结构、软件设计的质量分析与评价。

在编写方式上，始终强调以软件设计能力素质培养为主线，通过标准描述、设计要点和设计示例，让读者对软件设计架构和设计原则具有明确的认识和实践参照。在每章的习题部分，编排了大量软件设计的实战内容和系统架构设计师的历年考题，便于读者全面地把握软件设计和体系结构的丰富内涵，形成解决一定复杂系统软件工程问题的能力。

本书主要面向软件工程、计算机科学与技术等本科专业，作为软件系列课程“软件设计与体系结构”“高级软件系统设计与体系结构”的教材或参考书，以及供“软件工程课程设计”“应用软件架构课程设计”等课程参考，对研究生、教师和科研人员开展软件技术开发也会有良好的帮助。

前　言

软件体系结构（Software Architecture）是对软件系统的高层设计，是从一个较高的层次来考虑组成系统的构件、构件之间的连接关系，以及系统需满足的约束等，属于架构模式。软件设计模式是中层模式，是针对系统局部设计问题给出的解决方案。一方面，在实现架构模式时，可能采用多种设计模式。在一个复杂的软件项目中，还可能需要采用多种架构模式。另一方面，对软件系统设计的高层和中层划分，也不是绝对的，而是在不断演变。这从云计算系统的设计中就可见一斑。

本书的编写参考了大量中外文文献，并结合作者的多年教学和科研经验，历经一年多而成。在写作之初的调研中，发现并归纳出以下几个问题。

一是现有书籍的可参考性问题：有的书籍突出软件架构的学术性和理论价值，部分内容适合软件工程专业的研究生考试，但不适合本科生教学，也难以开展软件架构实践；有的书籍强调了规范化应用，但将软件设计偏向基于 UML 的应用实践，却很少基于软件体系结构去开展软件的高层设计；还有的书籍的主要篇幅为翻译国外教材内容。

二是软件考级作用的关注度不够：在系统分析员和软件架构师考题中，软件架构和设计占了较大分量，但在现有的软件设计书籍中，几乎都忽略了这个作用，没能给考级的软件工程师提供重要的参考。

三是未能体现最新流行的软件架构设计内容，例如，缺少对 MVC 风格、Web 服务、云计算架构的内涵描述和应用示例。

因此，本书的内容必须解决以上问题，并满足软件工程专业规范的课程大纲要求。在取材方面，认真参考了软件设计方面的最新相关标准或规范，例如，2014 年的 IEEE 软件工程知识体系 SWEBOK V3.0、软件架构描述国际标准 IEEE1471-2000 和软件质量模型国际标准。

在内容安排上，主要参照 SWEBOK V3.0 规定的软件设计主题进行编排。同时，引入最新的云计算软件架构内容。本书一共分为 9 章，依次为概述、经典软件体系结构风格、软件体系结构建模、软件设计模式、软件设计的关键问题、客户机 - 服务器软件架构设计、数据集成系统设计、云计算体系结构、软件设计的质量分析与评价。

在编写方式上，始终强调以软件设计能力素质培养为主线，通过标准描述、设计要点和设计示例，让读者对软件设计架构和设计原则具有明确的认识和实践参照。同时，在每章的习题部分，编排了大量软件设计的实战内容和系统架构设计师的历年考题，便于读者全面地把握软件设计和体系结构的丰富内涵，形成解决一定复杂系统软件工程问题的能力。

本书的学习提倡理论与实践相结合，宜安排 50 学时左右。在软件设计的实验过程中，读者更能体会到软件体系结构风格的合理选择和设计模式的合理运用。在工具环境方面，书中推荐了软件体系结构建模工具 ArchStudio、AcmeStudio，而 UML 建模工具有很多，如 VS.NET、StarUML、Visio、Rose 和 PowerDesigner。在软件体系结构风格方面，建议对 MVC 风格、B/S 结构、Web 服务应用等安排实际训练。在云计算方面，可以通过虚拟机方式配置 Hadoop 伪分布式计算环境，学习 MapReduce 计算模型及其软件架构原理。

下表给出一个学时分配参考。

序　号	名　称	理论学时分配	实践学时分配	实　验	其他训练
1	概述	4			阅读论文：第 1 章习题 1
2	经典软件体系结构风格	6	2	MVC 实验	第 2 章习题 3 或 4
3	软件体系结构建模	6	4	软件体系结构建模实验	第 3 章习题 5、6、7、8
4	软件设计模式	4	2	设计模式实验	第 4 章习题 2
5	软件设计的关键问题	4	2	数据持久化实验或组件应用实验	第 5 章习题 5、7
6	客户机－服务器软件架构设计	4	2	网络应用系统设计实验	第 6 章习题 3、5
7	数据集成系统设计	2	2	Web 服务应用实验	第 7 章习题 8 或 9
8	云计算体系结构	2	2	Hadoop 系统架构实验	第 8 章习题 11
9	软件设计的质量分析与评价	2			第 9 章习题 2
参考学时		34	16		

本书主要供具有一定程序设计基础的本科生使用，面向软件工程、计算机科学与技术等专业，可以作为软件系列课程“软件设计与体系结构”“高级软件系统设计与体系结构”的教材或参考书，以及供“软件工程课程设计”“应用软件架构课程设计”等课程参考，还可以供研究生的软件设计课程使用。

在本书策划和编写过程中，得到了中国石油大学（北京）陈明教授的亲切指导。同时，北京师范大学出版社对本书的出版给予了大力支持并付出辛勤劳动。书中有的内容引用了参考文献和互联网发布信息。此外，研究生徐日华、陈亚峰和曹国清在相关组件应用测试中也做了许多工作。在此一并表示感谢！

由于软件设计内容涉及面广、技术发展迅速，加之作者水平有限和时间仓促，书中难免有不妥之处，敬请广大读者不吝赐教。请联系邮箱 993103401@qq.com。

张晓明

2017 年 9 月于北京

目　录

第1章 概述

学习目标 ▶▶▶

（1）掌握软件体系结构的含义，了解软件设计中软件架构和设计模式的关系。

（2）熟悉软件体系结构的发展情况。

（3）了解软件架构师的职业要求。

1.1 软件工程知识体系

软件设计是软件工程的核心内容，它既是“过程”，也是这个过程的“结果”。软件设计由软件体系结构设计和软件详细设计两种活动组成，不仅涉及软件体系结构、构件、接口以及系统或构件的其他特征，还涉及软件设计质量分析和评估、软件设计的符号、软件设计的策略和方法等。

目前，IEEE 最新发布的软件工程知识体系（SWEBOK V3.0），将软件工程知识体系做了许多更新。更新内容如下。

- 在软件设计和软件测试中新增了人机界面的内容。
- 把软件工具的内容从原先的“软件工程工具和方法”中移到其他知识域中，并将该知识域重命名为“软件工程模型和方法”，使其更关注方法。
- 更突出了架构设计和详细设计的不同。
- 在软件设计中增加了硬件问题的新主题。
- 在软件设计中增加了面向方面（aspect-oriented）设计的讨论。
- 新增了软件重构、迁移和退役的新主题。
- 更多地讨论了建模和敏捷方法。
- 在多个知识域中增加了对保密安全性的考虑。
- 合并了多个标准中的参考文献，并进行更新和遴选，极大减少了文献数量。

软件工程实践知识域包含 11 个：软件需求、软件设计、软件构造、软件测试、软件维护、软件配置管理、软件工程管理、软件工程过程、软件工程模型和方法、软件质量、软件工程职业实践。软件工程教育基础知识域包含 4 个：软件工程经济学、计算基础、数学基础

和工程基础。相关学科有 7 个：计算机工程、计算机科学、管理、数学、项目管理、质量管理、系统工程。

按照 SWEBOK V3.0 的描述，软件设计的主题分解结构如图 1.1 所示。

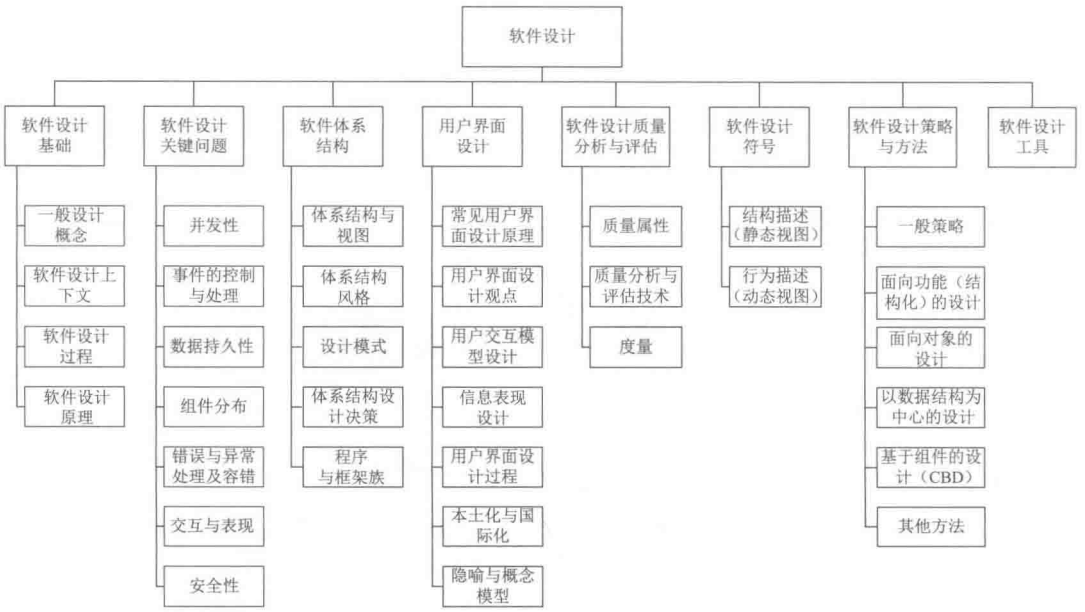

图 1.1　软件设计的主题分解

下面介绍其中部分内容，而软件体系结构将在后续重点描述。

1.1.1　软件设计基础

软件设计主题包括：一般设计概念、软件设计上下文、软件设计过程和软件设计原理。

（1）一般设计概念

软件并不是有设计的唯一领域。一般而言，可以将设计看作一种问题求解，例如，所谓难题或没有确定性答案的问题，对于理解设计的限制就很有趣。从一般意义来说，其他许多符号和概念对于理解设计也有趣，如目标、约束、候选方案、代表、答案等。

（2）软件设计上下文

为理解软件设计的作用，理解其所处的上下文——软件工程生命周期——很重要，这样，理解软件需求分析、软件设计、软件构造、软件测试就非常重要。

（3）软件设计过程

通常认为软件设计分为两步。

- 体系结构设计：描述软件如何分解和组织成组件（软件体系结构）。
- 详细设计：描述这些组件的特定行为。

（4）软件设计原理

根据牛津英语词典，“原理”是“一个基本的真理或普遍的法则，用来作为推理的基础或行动的指南”。软件设计原理是针对许多不同的软件设计方法和概念的关键描述，包括抽象、耦合与聚合、分解与模块化、封装与信息隐藏、接口和实现的分离、充分性、完备性与原始性、关系分解。

1.1.2　软件设计关键问题

设计软件时，必须处理许多关键问题，包括并发性、事件的控制与处理、数据持久性、组件分布、错误与异常处理及容错、交互与表现、安全性。

（1）并发性

如何将软件分解为多个进程、任务和线程，以处理相关的效率、原子性、同步和调度问题。

（2）事件的控制与处理

如何组织数据和控制流，如何通过不同的机制（如隐含调用和回调）处理交互和临时事件。

（3）数据持久性

如何处理需要长时间生存的数据。

（4）组件分布

如何将软件分布到各个硬件上，组件如何通信，如何使用中间件来处理异构软件。

（5）错误与异常处理及容错

如何阻止和容许故障，如何处理异常条件。

（6）交互与表现

如何设计结构和组织来实现与用户的交互及信息的表现形式（例如，使用模型—控制器—视图方法，将表现与业务逻辑分离）。

（7）安全性

安全性设计涉及如何防止信息和其他资料的非授权泄露、生成、更改、删除和拒绝，也涉及通过限制损失、连续服务、加速维护与恢复、安全失效与恢复，来容忍安全相关的攻击或冲突。基本的安全概念是访问控制，并确保加密方法的合理应用。

1.1.3　软件设计质量分析与评估

（1）质量属性

对于得到一个高质量（可维护性、可移植性、可测试性、可追踪性、正确性、健壮性、目的的适应性）的软件设计，多种质量属性都认为是重要的。一个有趣的区分是在运行时间可区别的质量属性（性能、安全性、可用性、功能性、可使用性），在运行时间不可区别的质量属性（可修改性、可移植性、可复用性、可集成性、可测试性），与体系结构本质质量相关的质量属性（概念完整性、正确性、完备性和可构造性）。

（2）质量分析与评估技术

有多种工具和技术来帮助人们确保软件设计的质量。

① 软件设计评审：有正式的和半正式的，通常是以小组方式进行，来验证和保证设计结果的质量（例如，体系结构评审、设计评审和检查、基于场景的技术、需求追踪）。

② 静态分析：正式或半正式的静态（不可执行的）分析技术，可以用于评估一个设计（例如，故障树分析或自动交叉检查）。

③ 模拟与原型：是评价设计的动态的技术。例如，性能模拟或可行性原型。

（3）度量

使用度量可以评定或定量估计软件设计的不同方面：规模、结构、质量。已经提出了许多度量，它们多数依赖产生设计的方法。这些度量可以分为两类。

① 面向功能（结构化）设计的度量：通过功能分解得到的设计结构，通常表示为结构图（有时称为层次图），可以计算其多种度量。

② 面向对象设计的度量：设计的总体结构通常表示为类图，可以计算多种度量，也可以计算每个类内部的内容的度量。

1.1.4 软件设计符号

表达软件设计成果的符号和语言，一些主要用于描述设计的结构组织，另一些用于描述软件行为。某些符号主要在体系结构设计中使用，另一些则主要用于详细设计，少部分可以用于这两个步骤。另外，一些符号通常用于特定方法的上下文中（参见软件设计策略与方法子域）。这里，将符号分为结构描述（静态视图）和行为描述（动态视图）两类。

（1）结构描述（静态视图）

下面的符号，主要是（但不总是）图形，描述和表示软件设计的结构方面，即它们描述主要的组件和组件间的联系（静态视图）。

① 体系结构描述语言（Architecture Description Languages，ADL）：文本（通常是形式化的）语言，以组件和组件间相互联系的方式描述软件体系结构。

② 类图和对象图：用于表示类或对象的集合，以及它们之间的联系。

③ 组件图：用于表示组件集合和组件间联系。

④ 协作责任卡（Collaboration Responsibilities Cards，CRC）：用于表示组件（类）的名称、责任和协作组件名称。

⑤ 部署图：用于表示一组（物理）节点及其相互联系，表示了系统的物理外观。

⑥ 实体联系图：用于表示存储在信息系统中数据的模型。

⑦ 接口描述语言（Interface Description Languages，IDL）：类编程语言，用于定义软件组件的接口（输出的操作的名字和类型）。

⑧ Jackson 结构图：以顺序、选择和重复的方式描述数据结构。

⑨ 结构图：用于描述程序的调用结构（一个模块调用其他模块，调用某个模块的其他模块）。

（2）行为描述（动态视图）

下面的符号和语言，一些是图形的，一些是文本的，用于描述软件和组件的动态行为。多数符号用于详细设计。

① 活动图：用于表示从活动（在一个状态机内进行的非原子执行）到活动的控制流。

② 协作图：用于表示发生在一组对象之间的交互，重点是对象、对象的链接、对象在链接上交换的消息。

③ 数据流图：用来表示数据在一组处理过程之间的流动。

④ 决策表和决策图：用于表示条件和行动的复杂组合。

⑤ 流程图和结构化流程图：用于表示控制流和要完成的对应活动。

⑥ 序列图：用于表示一组对象之间的交互，重点在按时间顺序的消息交换。

⑦ 状态变迁与状态图：用于表示状态机中状态之间的控制流。

⑧ 形式化描述语言：文本语言，使用来自数学的基本符号（例如，逻辑、集合、顺序）来严格和抽象地定义软件组件接口和行为，通常形式是前置条件和后置条件。

⑨ 伪码和程序设计语言：结构化的类编程语言，通常在详细设计阶段，用于描述一

个过程或方法的行为。

1.1.5　软件设计策略与方法

有各种一般的策略来帮助指导设计过程。与一般的策略不同，方法则更为专业，方法通常建议和提供了与方法一起使用的一组符号，并描述了遵循方法时要经历的过程，以及使用方法的指南。这些方法作为传递知识的手段和软件工程师小组的公共构架，很有用处。

（1）一般策略

在设计过程中常常被引用的、有用的一般策略是分而治之和逐步求精、自顶向下与自底向上、数据抽象与信息隐藏、使用启发式规则、使用模式和模式语言、使用迭代和增量方法。

（2）面向功能（结构化）的设计

这是软件设计的一个经典方法，分解的中心集中在标识主要的软件工程上，然后以自顶向下的方式，不断详细描述和精化这些功能。结构化设计通常在结构化分析后进行，产生数据流图对应的过程描述。研究人员提出了各种策略（例如，变换分析和事务分析）和启发式方法（例如，扇入 / 扇出、影响范围 / 控制范围）来将一个数据流图变换为通常用结构图表示的软件体系结构。

（3）面向对象的设计

已经提出了许多基于对象的软件设计方法，这个领域从 20 世纪 80 年代中期的基于对象的设计（名词 = 对象，动词 = 方法，形容词 = 属性）发展到面向对象的设计，其中，继承和多态性起着关键的作用，再发展到基于组件的设计，其中，可以定义和访问元信息（例如，通过反射）。虽然面向对象设计源于数据抽象，但人们提出了责任驱动的设计作为面向对象设计的另一个选择。

（4）以数据结构为中心的设计

以数据结构为中心的设计（如 Jackson 方法、Warnier-Orr 方法）从程序要操纵的数据结构开始，而不是从程序要完成的功能开始。软件工程师首先描述输入 / 输出的数据结构（例如，使用 Jackson 的结构图），然后基于这些数据结构图来开发程序的控制结构。人们提出了各种启发式规则来处理特殊情况，例如，当输入结构与输出结构不匹配时的情况。

（5）基于组件的设计

一个软件组件是一个独立的单元，具有良定义的接口和可以独立组合和部署的依赖性。基于组件的设计要解决为了改进复用而提供、开发、集成这些组件有关的问题。

1.2　软件复用

复用（Re-use）也称为“重用”，是指“利用现成的东西”。被复用的对象可以是有形的物体，也可以是无形的成果。复用是智慧的表现，因为人类总是在继承了前人的成果，不断加以利用、改进或创新后才会进步。

在通信领域，复用（Multiplexing）技术是指一种在传输路径上综合多路信道，然后恢复原机制或解除终端各信道复用技术的过程。在数据通信中，复用技术提高了信道传输效率，有广泛应用。

把复用的思想用于软件开发，称为软件复用，是指在两次或多次不同的软件开发过程

中重复使用相同或相似软件元素的过程。软件元素包括程序代码、测试用例、设计文档、设计过程、需要分析文档甚至领域知识。通常，可重用的元素也称作软构件，可重用的软构件越大，重用的粒度越大。

1.2.1 软件复用标准

1968 年，McIlroy 在 NATO（北大西洋公约组织）的软件工程会议上正式提出了软件复用的概念。至今为止，复用的对象也从早期的代码复用扩展到对软件开发过程中一切有价值的信息的复用，包括需求、需求规约、设计、源代码、测试计划和测试案例等。

软件复用包括开发可复用软件构件和基于可复用构件的开发两个生命周期。在这两个生命周期中，采用一个适当的标准以识别和开发可复用软件将大大促进软件复用的实践。为此，NATO 制定了一整套软件复用的指导性标准，以帮助 NATO 及其参与国和承包商的项目管理部门进行有效的软件复用。这套标准包括《可复用软件构件开发指南》、《可复用软件构件库管理指南》和《软件复用过程指南》3 个文档，分别从软件生命周期的各个阶段对软件复用进行指导，以便最大限度地减少复用代价和增加复用收益。

最新的软件复用国际标准主要如下。

（1）IEEE 1517-2010 信息技术—软件生命周期过程—复用过程标准。

（2）IEEE/EIA 12207.0 标准—信息技术—软件生命周期过程标准。

IEEE 1517 标准从两方面说明了系统级复用的实践。

（1）消费者复用：利用资产进行软件系统和产品的开发、运行及维护。

（2）生产者复用：资产的开发、管理和维护。

1.2.2 组件复用的含义和分类

对象管理组（Object Management Group，OMG）的建模语言规范中将组件定义为“系统中一种物理的、可代替的部件，它封装了实现并提供了一系列可用的接口。一个组件代表一个系统中实现的物理部分，包括软件代码（源代码、二进制代码、可执行代码）或者一些类似内容，如脚本或者命令文件”。

组件复用是软件复用的核心技术，旨在利用已有组件创建新组件以提高组件软件开发效率，组件复用通过包容和聚合来实现。包容时外部组件包含内部组件的接口；聚合时外部组件直接向外公开内部组件的接口。

软件复用的主要思想是将软件看成是由不同功能成分的“组件”所组成的有机体，每一个组件在设计编写时都被设计成可以完成同类工作的通用部件。这样，当完成各种工作的组件被建立起来后，编写一特定软件的工作就变成了将各种不同组件连接起来，这对于软件产品的最终质量和维护工作都有本质的改变。

因此，软件组件（Software Components）是共生于软件复用的，基于组件的软件复用是产品复用的主要形式，软件组件技术是当前复用研究的焦点。一般来说，软件组件是一种定义良好的、独立的、可复用的二进制代码，包括功能模块、被封装的对象类、软件框架和软件系统模型等。与面向对象编程语言不同，组件技术是一种更高层次的对象技术。它独立于语言，面向应用程序，只规定组件的外在表现形式，而不关心其实现方法。因此，组件最重要的特征是具有独立于应用的接口，这个优点可以使它不加修改或者基本不加修

改就可作为一个部件和其他组件一起组装成更大的软件。

在软件生命周期各阶段，组件复用可以是设计复用、代码复用或开发过程的其他产品复用。如程序模板、源代码或目标模块、需求说明、规格说明、程序说明、数据说明、测试说明和测试用例等。

软件组件是软件提供业务或技术功能的基本单元或元素，这些单元可以独立地部署。业务组件（Business Components）是执行业务逻辑、遵循一定的业务规则并且管理相应的数据；而技术组件（Technical Components）则是提供相应的平台以便业务组件可以运行，如权限管理、组件管理等。

（1）技术层面的复用

技术层面的复用通常包含 3 类。

① 通用基本组件：是应用系统的基本构成成分，如基本的数据结构、用户界面元素等，它们大同小异地存在于各种应用系统中。

② 领域共性组件：是应用系统所属领域的共性构成成分，它们存在于该领域的各个应用系统中。

③ 应用专用组件：是每个应用系统的特有构成成分。应用系统开发中的重复劳动主要在于前两类构成成分的重复开发。

在应用系统的前两类中，越靠底层的部分越容易复用，所以人们在软件复用方面的研究和应用经历了从底层到高层的过程，先后经历了库函数、面向对象、软件组件、开发框架等。目前各种开发框架复用已经得到广泛应用，例如，在系统整体结构设计规划阶段，包括全局组织与控制结构，组件间通信、同步和数据访问的协议，伸缩性和性能设计选择等。由于开发框架本身的通用性造成其组件粒度比较小，抽象程度比较高。所以，开发框架复用覆盖了可复用软件组件的所有活动。当开发同一领域中的新应用时，可以根据开发模型复用确定新应用的需求规格，并以此为基础选择可复用组件进行组装，从而形成新系统。

（2）业务层面的复用

许多开发者的软件复用大多是从技术角度出发的，如 J2EE 组件框架是一个以库、类和接口形式提供的基础架构，最终构成应用的业务逻辑和表现 / 控制逻辑则要由建立在这个框架上的业务组件实现。而实际上，应用软件最终要解决的却是应用问题或者说是业务问题，如果软件能够在更高层次的业务层面上进行大范围复用，那么对提高软件开发效率的作用将会更大。

由于大部分软件的开发过程是从抽象级别较高的形态向抽象级别较低的形态演化，所以较高级别的复用容易带动较低级别的复用，因而复用的级别越高，可得到的回报也越大。因为无论使用哪种技术，都需要首先形成一个个遵循一定业务规则、执行一定业务逻辑并管理一定数据，可在某一领域内多种项目中重复使用的组件。它们不同于 Struts、Hibernate 这样的技术组件或者由技术组件形成的框架，后者并不能解决特定的业务问题，而是为业务组件提供赖以生存的运行基础。

因此，组件技术正呈现业务化的发展趋势。随着技术层次组件的积累和成熟，企业应用开发迫切需要的不再是细粒度的技术组件，而是粗粒度的业务组件，以业务组件为中心的面向组件开发才能够真正提升开发速度、降低开发成本，并改善软件质量。

1.2.3 主流的组件复用技术

面向对象技术的组件模型为软件体系结构设计和大型应用软件开发提供了强有力的支持，目前已经为开发者广泛接受。现在主要有两种组件模型统治着市场：微软的 OLE/COM 和 Sun 的 JavaBean/EJB，它们构成了实现级组件模型工业标准的竞争与互操作并存的格局。

（1）OLE/COM

微软的 OLE/COM 是基于分布式对象模型的开放标准，得到许多系统软件开发商、独立软件开发商（ISV）和用户的支持。OLE 实际上是建立在组件对象模型（COM）基础上的一组高层次技术。从基本中间件功能视图的角度来说，COM 支持对象的定义、创建、调度、引用及对象之间的通信，提供接口定义语言（IDL）。

COM 支持由不同程序设计语言或不同编译器实现的对象之间的二进制兼容。OLE/COM 结构的另一要素是 Automation，允许客户程序动态构造请求（包括方法名、相关参数的类型和取值等），并将请求发送到远端对象，任何符合 OLE/COM 规范的对象都能自动提供其所能支持的接口信息。

在有目录和其他支持的网络中，COM 变成了分布式 COM（DCOM）。

为了把 COM、DCOM 和 MTS 统一起来，形成真正适合于企业级应用的构件技术，诞生了 COM+。COM+ 是一种中间件技术的规约，其要点是提供建立在操作系统上的、支持分布式企业级应用的“服务”。COM+ 与 Windows DNA 一起，使得用户可以采用 Microsoft 公司的技术开发服务器端的构件。

COM+ 的核心是改进的 COM/DCOM 和 MTS 的集成，但是 COM+ 增加了一些非常重要的构件服务，如负载平衡、驻留内存数据库、事件模型、队列服务等。

（2）JavaBean

JavaBean 是基于 Java 环境，可视的、可操纵的和可复用的组件。JavaBean 为软件构造工具所利用，也能通过程序接口直接操纵，Java 类库中提供了相应的控制类。JavaBean 系统扩展了 JavaApplet 以适应基于组件的软件开发所需的更复杂的软件组件。JavaBean 组件模型是 Sun 制定的关于 Bean 的软件组件标准，规定设计所有 Bean 所依据的框架，确保 Bean 在具备特定功能的同时，还能被可视化软件构造工具所识别、操纵，并能将这些设计信息保存下来，指导运行时的行为。

作为可视化组件，所有 JavaBean 都具备如下特征：内省（introspection）机制，能够告诉软件构造工具其所能完成的功能，从而允许软件构造工具在设计时对其加以操纵。用户定制（customization）机制，允许程序员在软件开发阶段利用软件构造工具改变 Bean 的外观和行为方式。事件（event）机制，能捕捉、引发事件，并将其所能产生和处理的事件告知软件构造工具。特性（properties）机制，除在软件开发阶段支持用户定制外，还使软件系统能够在运行时刻对 Bean 进行加工和控制。保持（persistence）机制，保存程序员开发时利用构造工具对 Bean 所做的修改，并在运行时予以恢复。

1.3 软件体系结构概述

下面分别阐述软件体系结构的概念、主题和发展情况。