一、软件体系结构概论

1.1软件危机

软件危机的表现 (P1)

- 软件成本日益增长
- 开发进度难以控制
- 软件质量差
- 软件维护困难

软件危机的原因(P2)

- 用户需求不明确
- 缺乏正确的理论指导
- 软件规模越来越大
- 软件复杂度越来越高

如何克服软件危机(P3)

软件工程三要素：`方法`，`工具`和`过程`，其中：

- 软件工程方法为软件开发提供了"如何做"的技术，是完成软件工程项目的技术手段
- 软件工具是人们开发软件的活动中智力和体力的扩展和延伸，为软件工程方法提供了自动化的或半自动化的软件支撑环境
- 软件过程过程则是将软件工程的方法和工具综合起来以达到合理利用、及时地进行计算机软件开发的目的

1.2构件和软件重用

- 软件重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近软件元素的过程
- 软件元素包括代码、测试用例、设计文档、设计过程、需求分析文档甚至领域知识。通常把这种可重用的元素称作软件构件，简称构件。
- 可重用的软件元素越大，就说明重用的**粒度**越大。

构件获取(P6)

在建立`基于构件的软件开发`（Component-Based Software Development, CBSD)中, 构件有多种不同的途径：

- 从现有的构件中获得符合要求构件，直接使用或作适用性修改，得到可重用的构件。
- 通过遗留工程，将具有潜在重用价值的构件提取出来，得到可重用的构件。
- 从市场购买现成的商业软件，即COTS(Commercial Off-The-Shell) 构件。
- 开发新的符合要求的构件。

构件管理(P7)

构件分类方法可以分为三大类，分别为关键字分类法、刻面分类法和超文本组织方法。

关键字分类法

关键字分类法是一种最简单的构件库组织方法，其基本思想是：根据领域分析的结果将应用领域的概念按照从抽象到具体的顺序逐次分解为树状或有向无回路图结构。每一个概念用一个描述性的关键字表示。不可分解的原子级关键字包含隶属于其它的某些构件。

刻面分类法

超文本组织方法

超文本方法与基于数据库系统的构件库组织方法不同，它是基于全文检索技术。其主要思想是：所有构件必须辅以详尽的功能或行为说明文档；说明中出现的重要概念或构件以网状链接的方式相互连接；检索者在阅读文档的过程中按照联想思维方法任意跳转到包含相关概念或构件的文档；全文检索系统将用户给出的关键字与说明文档的文字进行匹配，实现构件的浏览式检索。

1.3软件结构的定义和兴起

软件体系结构的定义(P19)

软件体系结构的定义有以下几个典型：

- 软件体系结构是具有一定形式的结构化的元素，即构件的集合，包括处理构件、数据构件和连接构件。
- 软件体系结构是软件设计过程中的一个层次，这一层次超越计算过程中的算法设计和数据结构设计。
- 软件体系结构是一个抽象的系统规范，主要包括用其行为来描述的功能构件和构件之间的相互连接、接口和关系。

软件体系结构为软件系统提供了一个结构、行为和属性的高级抽象，由构成系统的元素的描述、这些元素的相互作用、指导元素集成的模式以及这些模式的约束组成。 软件体系结构不仅指定了系统的组织结构和拓扑结构,并且显示了系统需求和构成元素之间的对应关系,提供了一些设计决策的基本原理.

软件体系结构的意义(P19)

体系结构是风险承担者进行交流的手段

体系结构代表了系统的公共的高层次的抽象。这样，系统的大部分有关人员(即使不是全部)能把它作为建立一个互相理解的基础，形成统一认识，相互交流。

体系结构是早期设计决策的体现

- 软件体系结构明确了对系统实现的约束条件
- 软件体系结构决定了开发和维护组织的组织结构

- 软件体系结构制约着系统的质量属性
- 通过研究软件体系结构可能预测软件的质量
- 软件体系结构使推理和控制更改更简单
- 软件体系结构有助于循序渐进的原型设计
- 软件体系结构可以作为培训的基础

软件体系结构是可传递和可重用的模型

软件体系结构级的重用意味着体系结构的决策能在具体相似的多个系统中发生影响,这比代码级的重用要有更大的好处.

软件体系结构的发展史(P22)

四个阶段:

- 无体系结构阶段。以汇编语言进行小规模应用程序开发为特征。
- 萌芽阶段。出现了程序设计结构主题，以控制流图和数据流图构成软件结构为特征。
- 初期阶段。出现了从不同侧面描述系统的结构模型，以UML为典型代表。
- 高级阶段。高层抽象结构为中心，不关心具体的建模细节，划分了体系结构模型和传统软件体系结构的界限，该阶段以Kruchten提出的"4+1"模型为标志。

二、软件体系结构建模

模型分为5种（P29):

- 结构模型
- 框架模型
- 动态模型
- 过程模型
- 功能模型

2.1 "4+1"视图模型

4：逻辑视图、进程视图、物理视图、开发视图

1：场景视图

图见P29

逻辑视图

逻辑视图主要支系统的功能需求，即系统提交给最终用户的服务。

开发视图

开发视图也称模块视图，主要侧重于软件模块的组织和管理。

进程视图

进程视图也称并发视图，侧重于系统的运行特性，主要关注一些非功能性的需求，例如系统的性能和可用性。

物理视图

物理视图只要考虑如何把软件映射到硬件上，它通常考虑的到系统性能、规模、可靠性等。解决系统拓扑结构、系统安装、通信等问题。

场景

场景可以看作那些重要系统活动的抽象，它使4个视图有机联系起来，从某种意义上来说场景是最重要的需求抽象。

逻辑视图和开发视图描述系统结构的静态结构，而进程视图和物理视图描述系统的动态结构。

2.2 软件体系结构的核心模型

体系结构的核心模型由5中元素组成：构件、连接件、配置、端口和角色。其中，构件、连接件和配置是最基本的元素。

- 构件是具有某种功能的可重用的软件模板单元，表示了系统中主要的计算元素和数据储存。
    - 复合构件
    - 原子构件
- 连接件表示了构件之间的交互，简单的连接件如管道、过程调用、事件广播等，更为复杂的交互如客户-服务器通信协议，数据库和应用之间的SQL连接等。
- 配置表示了构件和连接件的拓扑逻辑和约束。

2.3软件体系结构的生命周期模型

软件体系结构的生命周期(P38)

图见P38

- 软件体系结构的非形式化描述
- 软件体系结构的规范描述和分析
- 软件体系结构的求精及其验证
- 软件体系结构的实施
- 软件体系结构的演化和扩展
- 软件体系结构的提供、评价和度量
- 软件体系结构的终结

三、软件体系结构风格

3.1 经典体系结构风格

管道与过滤器(P51)

每个构件都有一个输入和输出，构件读输入的数据流，经过内部处理，然后产生输出数据流。

特点：

- 使得软构件具有良好的隐蔽性和高内聚、低耦合的特点。
- 允许设计师将整个系统的输入/输出行为看成是多个过滤器的行为的简单合成。
- 支持 软件重用。
- 系统维护和增强系统性能简单。
- 允许对一些如吞吐量、死锁、等属性的分析。
- 支持并行执行。

缺点：

- 通常导致进程成为批处理的结构。
- 不适合处理交互应用。
- 因为数据传输没有通用的标准，每个过滤器都增加了解析和合成的工作，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。

基于事件的系统(P52)

隐式调用系统的主要优点：

- 为软件重用提供了强大的支持。当需要讲一个构件加入到现存系统中时，只需将他注册到系统的事件中。
- 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。

隐式调用的主要缺点：

- 构件放弃了系统计算的控制。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过程，它不能保证这些过程被调用的顺序。
- 数据交换问题。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。
- 既然过程的语义必须依赖于被触发事件的上下文约束，关于确定性的推理存在问题。

分层系统(P54)

层次结构有许多可取的属性：

- 支持基于抽象程度递增的系统设计，使设计师可以把一个复杂的系统按递增的步骤进行分解。
- 支持功能增强，因为每一层至多和相邻的上下层进行交互，因此功能的改变最多影响相邻的上下层。
- 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。

不足之处：

- 并不是每个系统都可以很容易的划分为分层的模式，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，设计师不得不把一些低级或高级的功能综合起来。
- 很难找到一个合适的、真确的层次抽象方法。

仓库系统及知识库(P54)

黑板系统的传统应用是信号处理领域，如语音和模式识别。

黑板系统主要由三部分组成：

- 知识源。知识源中包含独立的、与应用程序相关的知识吗，知识源之间不直接进行通信，他们之间的交互只通过黑板来完成。
- 黑板树结构。
- 控制。控制完全有黑板的转态驱动，黑板转态的改变决定使用的特定知识。

C2风格

C2风格中的系统组织规则如下：

- 系统中的构件和连接件都有一个顶部和底部
- 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。
- 一个连接件可以和任意数目的其它构件和连接件连接。
- 当两个连接件进行直接连接时，必须由其中一个的底部连接到另一个的顶部。

3.3 三层C/S结构风格

各层的功能(P59)

三层C/S体系结构是将应用功能分成表示层、功能层和数据层三个部分。

表示层

表示层是应用的用户接口部分，它负担着用户与应用之间的对话功能。

功能层

功能层相当于应用的大体，它是将具体的业务处理逻辑编入程序中。

数据层

数据层就是数据库管理系统，负责管理对数据库的读写。

三层C/S结构的优点

- 允许合理的规划三层结构的功能，使之在逻辑上保持相对独立性，从而使整个系统的逻辑结构更为清晰，能提高系统和软件的可维护性和可扩展性。
- 允许更灵活有效的选择相应的平台和硬件系统，使之在处理负荷能力上与处理特性上分别适用于结构清晰的三层；并且这些平台和各个组成部分可以具有良好的可升级性和开放性。
- 三层C/S结构中，应用的各层可以进行并发，各层也可以选择各自最适合的开发语言。使之能并行的而且是高效的进行开发，达到较高的性能价格比；对每一层的处理逻辑的开发和维护也会更容易些。
- 允许充分利用功能层有效的隔离开表示层和数据层，未授权的用户难以绕过功能层而利用数据库工具或黑客手段去非法的访问数据库，这就为严格的安全管理奠定了坚实的基础；整个系统的管理层次也更加合理和可控制。

> 值得注意的是：三层C/S结构各层次间的通信效率若不高，即使分配给各层的硬件能力很强，其作为整体来说页达不到所要求的的性能。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。

3.6 正交软件体系结构

(P70)

正交软件体系结构是由组织层和线索的构件组成。

3.10 特定领域软件体系结构(Domain Specific Software Architecture， DSSA)

DSSA就是一个特定应用领域中为一组应用提交组织结构参考的标准软件体系结构。

优点：

- 相当于过去的开发方法，系统开发、维护的工作量大幅度减少，整个应用系统的构件重用程序相当大。
- 便于系统开发的组织管理。
- 系统有较好的环境适应性，构建的升级引发应用系统的升级，并在构件库中合理的控制粒度，使系统的总体结构设计与算法和模块化设计同等重要，并灵活的保证新、老应用系统的共存。

DASS与体系结构风格的比较(P99)

在软体系结构的发展过程中，因研究者触发点不同，出现两个相互直交的方法和学科领域： 以问题域为出发点的DASS和以解决域为出发点的软件体系结构风格。

四、软件体系结构描述

ADL是这样一种形式化语言，它在底层语义模式的支持下，为软件系统的概念体系结构建模提供了具体的语法和概念框架。基于底层语义的工具为体系结构的表示、分析、演化、细化、设计过程等提供支持。其三个基本元素如下。(P106)

- 构件： 计算或数据存储单元。
- 连接件：用于构件之间交互建模的体系结构构造块及其支配这些交互的规则。
- 体系结构配置： 描述体系结构的构件与连接件的连接图。

八、基于服务的体系结构

面向服务的体系结构(Service-Oriented Architecture，SOA)

SOA概述

SOA的特征(P198)

- 松散耦合
- 粗粒度服务
- 标准化接口

面向服务的分析和设计(P199)

SOA有三个主要的抽象级别，分别为操作、服务和业务流程。

SOA关键技术(P201)

服务栈	主要技术
发现服务层	UDDI, DISCO
描述服务层	WSDL. XML, Schema
消息格式层	SOAP, RESET
编码格式层	XML
传输协议层	HTTP, TCP/IP, SMTP等

十一、软件体系结构评估

软件体系结构评估描述

软件质量属性(P258)

- 性能
- 可靠性
  - 容错
  - 健壮性
- 可用性
- 安全性
- 可修改性
  - 可维护性
  - 可扩展性
  - 结构重组
  - 可移植性
- 功能性
- 开变性
- 集成性
- 互操作性

十二、基于体系结构的软件开发

MVC把交互系统的组成部分分解为模型、视图、控制三种构件。

所谓设计模式，简单来说，就是一些设计面向对象的软件开发的经验总结