**Chapter 1 概述**

**1.1 软件体系结构**

软件体系结构定义了**软件的局部和总体计算部件的构成**，以及这些**部件之间的相互作用关系**。

总的来看，体系结构是由结构和功能各异、相互作用的**部件集合**，按照**层次**构成的

--

**体系结构是指整个系统构成的基本和主体形态。**

**软件体系结构是软件在设计构成上的基本、可供设计选择的形态和总体结构。**

**体系结构 = 组件 + 连接件 + 约束**

SE的目标：软件质量（易于理解、可度量、可复用、可文档化、易于交流和执行）

**1.2 当前的软件设计**

体系结构的设计选择对于软件的长远成功是至关重要的

体系结构描述的不规范性

体系结构的理论和工具

**1.3 软件设计的层次**

每个层次的设计可单独进行

低层次级为了高层次级服务

每个层次都包括：

构成系统的部件

部件合成系统的规则

**结构级**

**代码级**

**执行级**

**1.4 体系结构与软件的工程**

**实现体系结构指导下的软件工程设计，关键就是要建立软件结构的分析和构造方法。**

软件复用技术是范例

部件的抽取

识别部件之工作环境

**软件复用就是具有特定性能的模式或部件在不同应用中的多次频繁使用。**

体现：部件、模式

**体系结构对于软件具有宏观的视角，但体系结构设计的问题包含了软件结构的各个层次。**

**软件体系结构风格为大粒度的软件重用提供了机会。**

**1.5 软件体系结构的知识体系**

软件的体系结构应该是一个关于软件构成方面的具有层次性的知识体系。

-宏观的软件层次（高、中、低）

-软件的硬件层、基础控制描述层、资源及管理调度层、系统结构模式层、领域应用层

软件体系结构的概念不仅是几十年软件发展的反映，也是现时各类软件构成的实现。

**Chapter 2 软件体系结构的研究与发展**

**2.1 软件工程设计和软件体系结构**

软件设计的目标：**在时间个各类2.环境资源的限制下，最大限度的满足用户的需求**

1.便于维护和升级，因而应该是模块化的

2.设计应该是便于移植的（移植比重新设计花费要小的多）

3.设计过程应该受到理性的控制 Intellectual Control

4.设计应该表现出概念的完整性（包括内在结构和外在表现）

**2.2 什么是软件体系结构**

**软件体系结构是在识别可重用构件和连接件的基础上,研究软件结构的表达和分析的理论和技术。**

（1）现代定义

一个程序或计算系统的软件体系结构是一种结构，或者一种系统结构，它由软件元素，这些元素的外在可见性质以及元素之间的关系组成。

**软件体系结构被定义为系统的基本组织结构，包括构件、构件之间的关系、环境以及管理系统设计和演化的原则。**

（2）传统定义

软件体系结构是设计过程的一个层次，它处理那些超越算法和数据结构的设计，研究整体结构的设计和描述的方法。

**体系结构 = 组件 + 连接件 + 约束**

**Architecture = Components   
 + Connectors**

**+ Constrains**

**2.3 软件体系结构的意义和目标**

**意义：**

软件体系结构是软件开发过程初期的产品，对于开发进度和软件质量的一切资金和劳务投入，可以获得最好的回报。

体系结构设计是形成的投资高汇报的重要因素。

正确有效的体系结构设计会给软件开发带来极大的便利。

**目标：**

外向目标：建立满足终端用户要求的系统需求。

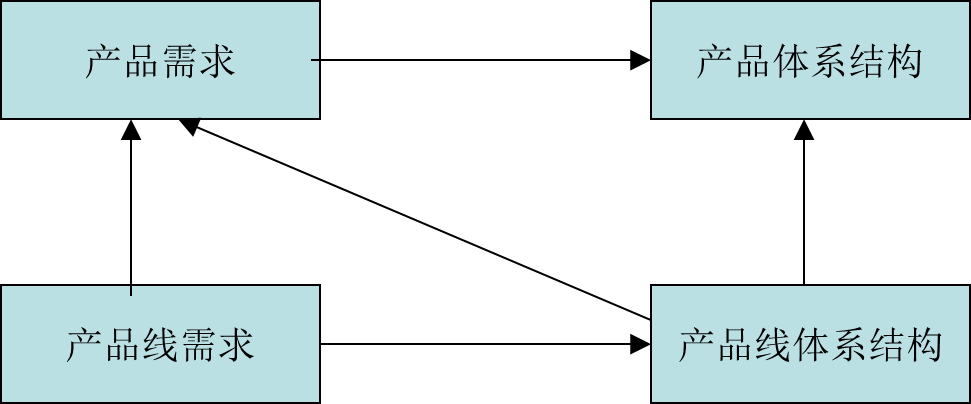
内向目标：建立满足后期设计者需要以及易于系统实现、维护和扩展的系统部件构成。

**2.4 软件体系结构的研究范畴**

（1）体系结构描述语言与工具：ADL（Architeture Description Language）、设计符号为体系结构建模

（2）产品线与标准：企业、需求、架构、系统

软件产品线是一个工程问题，它由一组软件密集型家族系统共享公共的、可管理的特征。



（3）软件体系结构风格及风格应用

软件体系结构风格是指一组设计词典、有关词典如何运用的限制条件、及词典语义的假设。

每个风格的应用仅适合于一定的目标，而不适用于其他目标。

（4）体系结构文档化

体系结构文档化的过程就是用一个或者多个视图来描述一个系统结构的过程。

David Garlan体系结构文档化4个视图

1.Context-based Views 基于文本的视图

2.Code-based Views 基于代码的视图

3.Run-time Views 运行时视图

4.Hardware-based Views 基于硬件的视图

**Chapter 3 软件体系结构的层次性**

**3.1 体系结构的基础和层次特性**

从建筑学看软件的构成：基础、层次、模式、角色划分

（1）基础

地基、材料、材料构成三个方面从根本上决定了建筑物的结构、性能、功用、建造方法，形成了建筑的**基础**。

构造软件同样需要**基础**。计算机硬件结构、软件的基本组成、构成软件的可用组块三个方面。

讨论软件的体系结构必须首先建立一个**基础**：一旦确立了基础，各种观点的比较就有了共同的标准语言。

（2）层次

建筑是由基本材料到基础构件再到整体框架逐**层次**发展和构成的历程。

软件的体系结构也是由使用最基本的材料开始，到认识常用基础构件再到组装和构造整体框架的发展过程。

（3）模式

形态和构件组成了建筑模式。20多年建立了现代建筑学的250余种“场景－问题－解决方案”模式，涵盖不同的规模和形态

这些建筑学的思想在软件结构的研究中也得到了重视，提出并发展了软件“**软件设计模式**”的概念,进而又提出了“软件体系结构模式”的概念。

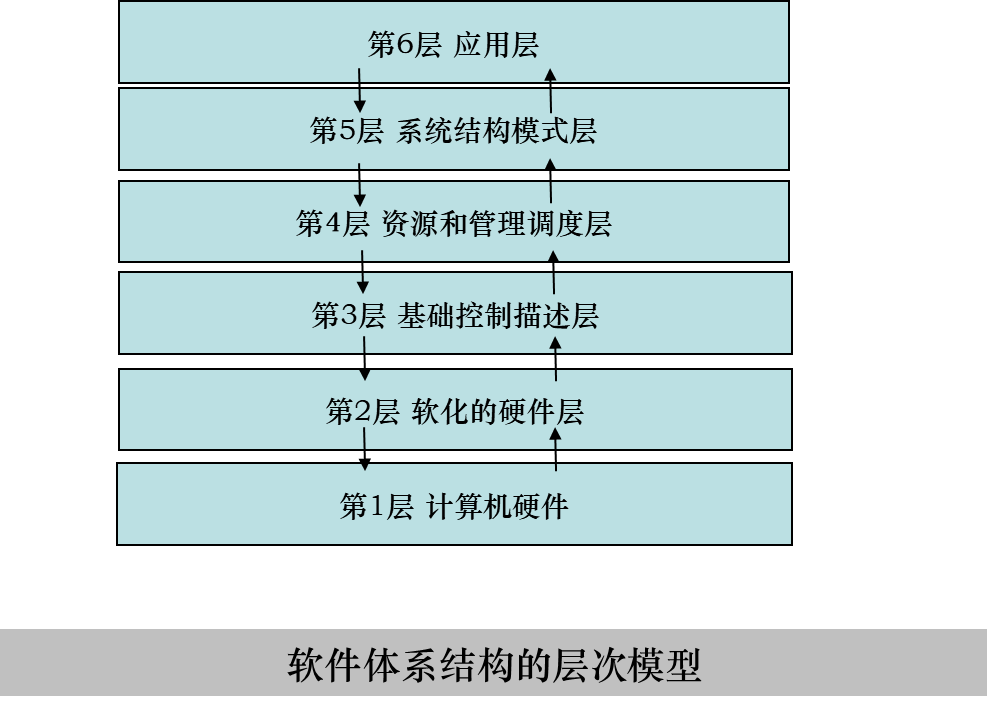
（4）角色划分

建筑业中存在建筑设计师、建筑工程师、建筑工人等角色划分。

相对应地，软件工程行业中也存在对应的角色划分：架构师、软件工程师、程序员等。

**3.2 软件体系结构的层次结构模型**

6层



**硬件基础层**

-软件运行的物质基础

**软化的硬件层**

-对硬件结构和性能抽象的基础上，实现硬件的操作和控制描述🡪软化的硬件层

处理器：状态和指令集合

中断：状态和中断服务

**基础控制描述层**

-建立在高级程序语言描述上的纯粹软件描述层，包括了高级语言所支持的所有程序控制和数据描述概念

程序控制概念：顺序、条件、选择、循环、变量、参数、过程、函数、模块

数据描述的概念：数组、队列、链表、堆栈、树、图、指针、记录

-支持该层面的软件系统模型：主程序/子程序、结构化程序、模块化程序、面向对象程序

-支持该层面的设计工具：程序设计语言、结构化分析、面向对象分析设计

**资源和管理层**

-在基础控制描述层建立的一切数据对象和操作，都需要在操作系统的协调和控制下才能实际的实现其设计的作用和功能，进程管理、消息处理、I/O etc.

-该层考虑的设计模式：共享资源、同步、分时系统、异常处理、并行、进程、线程、消息、远程调用 etc.

-该层的软件系统模型：进程控制、分时系统、消息机制 etc .

**系统结构模式层**

-最高层次的软件结构概念

-属于体系结构风格或系统级别的设计模式

-最高的抽象描述层

-该层包含的概念有：解释器、编译器、编辑器、管道/过滤器、黑板、C/S 、B/S、框架 etc.

**应用层**

-从纯粹应用领域出发所建立的系统结构概念

-是系统结构模式层的概念经过领域应用命名的直接引用：企业管理、公文处理、控制系统、CAD系统、ERP系统，etc.

**3.3 从层次模型看软件体系结构**

数据库、网络、JavaEE、Spring ，etc.

体系结构是关于软件的构成部件及其连接的分层的结构框架

体系结构包括软件的内在概念和外在操作

体系结构分析与设计涵盖并指导着从逻辑结构设计到运行实现的软件工程的全部过程

软件体系结构的层次模型说明了软件的构成和设计知识贯穿散布在计算机各领域中

软化的硬件层：组成、汇编、接口

基础控制描述层：语言、DS、OO、DB

资源及管理调度层：OS

系统结构模式层：DB、SE、DC

应用层:Background

建立统一、系统的软件知识体系

**Chapter 4 软件体系结构的设计原理**

**体系结构设计中遵循的原理**

1.抽象

抽象是人们用来处理复杂性问题的基本原理之一

数据抽象、实体抽象、行为抽象、过程抽象、对象抽象、虚拟机抽象

软件设计: 抽象无处不在

2.封装

将抽象的属性和行为结合在一起

为不同的抽象提供了明确的界限

有利于非功能特性的实现

可变性

可复用性

3.数据隐藏

对用户隐藏部件的实现细节

接口与实现相分离

信息的隐藏

如何隐藏取决于技术

隐藏哪些取决于具体的应用

用来更好地处理系统的复杂性和减少各部件之间的耦合

4.模块化

良好定义的分界将构成应用的逻辑结构物理地分割成代码实体

模块是一个应用的功能和责任的物理容器

优点：可复用

缺点：额外复杂的系统资源管理

5.注意点分离

不同和无关联的责任应该在软件系统中分离开来，让他们出现在不同的部件中

相互协作完成某一个特定任务的部件应该和在其他任务中执行计算的部件分离开来

避免过多暴露所造成的对应用设计的负担和混乱，保证了组件运行的可靠和安全

6.耦合和内聚

模块功能高内聚模块间关联低耦合

7.充分性、完备性和原始性

原始性是指部件应该完成的操作都可以容易地得到实现（人的价值是最大的）

8.策略和实现的分离

策略部件负责处理上下文相关的决策、信息的语义和解释的知识、把不相交计算组合形成结果、对参数值进行选择等问题

实现部件负责全面规范算法的执行，执行中不需要上下文相关信息进行决策

实现部件因为独立于上下文环境，因而更容易重用和维护

策略部件因为与特定的应用相关，通常随时间的变化而改变

9.接口与实现的分离

接口定义了部件所提供的功能并规范了功能的使用方法

实现部分包括了部件所提供功能的实际代码

强调一个客户只应该知道他需要知道的东西

是一种信息隐藏技术

接口和实现的分离可以很好地支持可变性

10.分而治之

自上而下的分析，自下而上的实现

横向的分解

纵向的分解

11.层次化

**软件的非功能特性**

**虽然非功能性在软件体系结构中非常重要，但很难对他们的效果和作用进行衡量**

1.可变性

为应对改变而设计的系统比没有进行这样的考虑而设计的系统更容易适应用户，因而该系统的适应性要好得多，生命周期要长得多

2.互操作性

互操作性是指不同的计算机系统、网络、操作系统和应用程序一起工作并共享信息的能力 。

将系统设计成具有互操作性的部件集合本身就自然地对系统的功能构成进行了分割。

3.效率

软件运行过程中对资源的使用情况、以及对系统的响应时间、存储消耗和I/O吞吐量的影响

效率问题不仅仅是设计精良算法的问题，而且是部件操作责任合理的分配、部件之间的耦合关系等体系结构的问题

良好结构、丰富功能和高效率等方面需要权衡利弊

4.可靠性

是软件系统在各种情况下维持其功能的能力，区分为2个方面的内容：

容错性：当错误事件发生时确保正确的系统响应，必要时采取内部补救措施

健壮性：健壮性不要求软件在发生错误之后还能继续执行，他只需要保证软件能以明确和可接收的方式终止

5.可测试性

测试变得越来越难、越来越昂贵

支持可测试性的软件体系结构可以为错误探测和改正、以及代码调试和部件的临时集成给予支持

6.可重用性

“通过已经存在的来获得想要的”的软件设计和实现方法（有利于Cost、Time、Quality的平衡）

用重用进行软件开发

为重用进行软件开发

**Chapter 5 部件和连接器**

**部件和连接器被公认为体系结构的两大类构成部分**

部件是软件功能设计和实现的载体

连接器是专门承担连接作用的特殊部件

**部件**

部件是软件系统的结构块单元，是软件功能设计和实现的载体，任何具有独立结构和行为特性的软件体都可以称为部件。

系统是部件及其关联的集合。

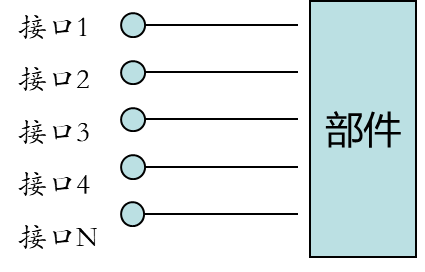
使用的时候：一个部件至少有一个接口，每一个接口代表对外联系的一种角色，这是部件与外界发生联系的窗口。

设计系统时候：需要根据对部件的功能、与其他部件的关联、对部件的特殊性要求，建立内部处理和控制结构。

（1）部件的表达形式

不同软件设计环境下服务于不同目的，部件具有不同的类型或名称

部件的一般表达形式



（2）部件的类别

按照层次划分：基础部件、中层部件、高层部件

按照应用范围分：专用部件、通用部件

按照功能分：数据服务部件、功能服务部件、逻辑处理部件、界面部件 etc.

（3）部件的特性

1.部件的接口特性：

完备性：使用者可以用它来完成部件应该能够完成的一切工作

最小化：部件的接口或界面中任一操作，都不能由其他操作组合而实现

正交性

方便性

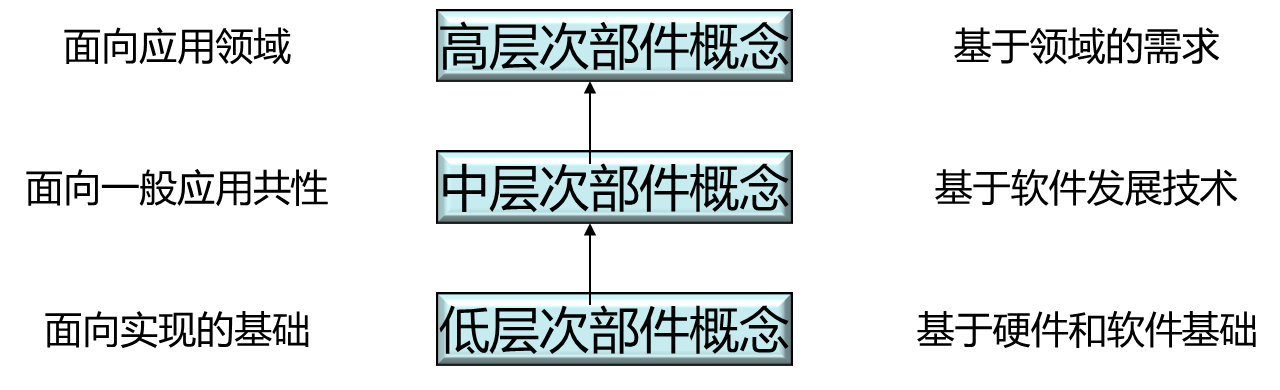
效率

2.部件的运行特性：中断处理、并行调度、多用户服务

3.部件的远程服务特性（服务部件）

多源性请求

4.部件的概念等级或层次



**连接**

连接是部件间建立和维持行为关联和信息传递的途径，是系统复杂性的来源。

简单连接和复杂连接

机制和协议

机制：让连接得以发生和维持

协议：连接能够正确、无二义性、无冲突的进行

（1）连接的种类

过程调用、控制、事件、消息发送、数据传送 etc.

（2）连接的实现机制

计算机硬件提供了实现一切连接的基础

高层次的连接建立在低层次的连接之上，实现连接在不同的层次上有不同的概念或方法

无论多么复杂的连接关系，其实现都是基于以下基本连接机制：

过程调用、中断、I/O、DMA、进程、线程、共享、同步、并/串行、事件、并发 etc.

（3）连接的协议

协议是连接的规约，是实现有意义连接的保证。

连接的规约是建立在物理层之上的有意义信息形式的表达规定。

即使是简单的连接，也有协议在起作用。

（4）连接的特性

连接的方向性

控制的渠道（主控、被控）和信息的传送（传送方、接收方）

双向性（复杂连接的双工通讯）

连接的角色

角色是对连接的双方所处地位不同的表达（调用者、被调用者；C/S；B/S；对等网的连接；中断源、中断处理者；）

角色和地位的不同在连接的实施中表现为所进行的操作不同、期望获得的信息不同

连接的激发

激发是指引起连接行为的方式

主动方的行为激发：操作调用、事件触发

从动方：状态查询、中断

连接的响应特性

响应特性包括从动方对连接请求处理的实时性、方式（同步、异步）、并发处理能力

响应特性大大增加了实现的复杂性

（5）连接的不匹配和解决方法

连接是使多个部件实现互联和协同工作的机制

产生连接冲突和不匹配的原因有多个方面：实现机制、协议、特性、硬件。

解决方法：A与B部件的修改（直接修改A或者B、转换器、代理、版本的一致）

**连接器**

任何部件都不是孤立存在的，只有部件间的联系才能发挥和实现部件设计的功能

连接器是实现部件与部件之间联系的特殊机制或特殊部件

连接器是实现部件与部件之间联系（调用、消息传递、数据转换传送、部件间实时并行的协调控制等）的特殊机制或特殊部件

简单的连接器从结构上退化为部件之间的直接连接，复杂的连接器需要专门的机构来完成，所以连接器也是部件

连接器承担了实现部件间信息和行为关联的作用，是系统复杂性的来源，对系统的各种性能有着重要的影响

连接器也是部件，是特殊的部件。

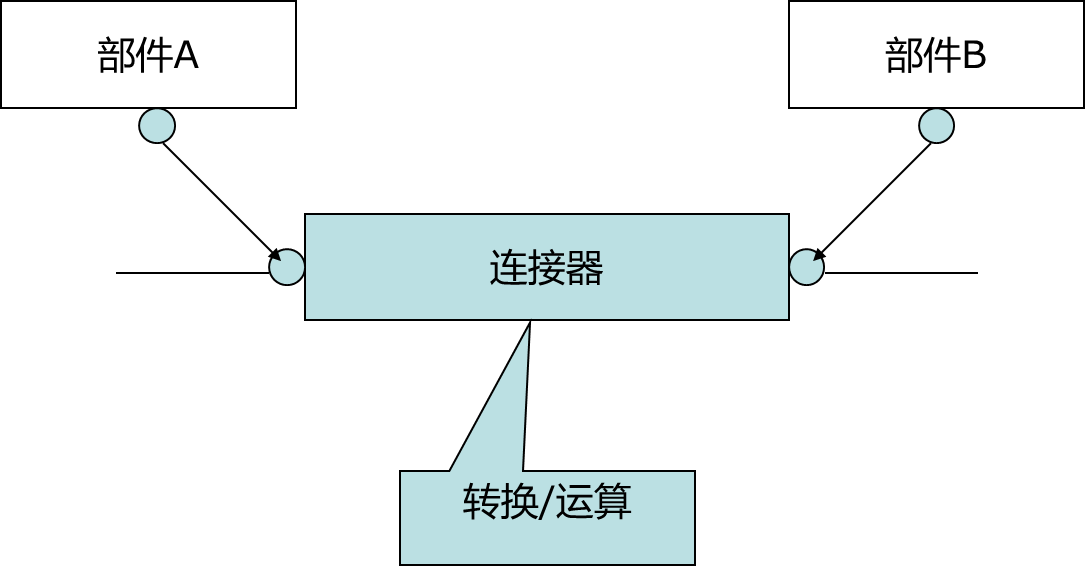
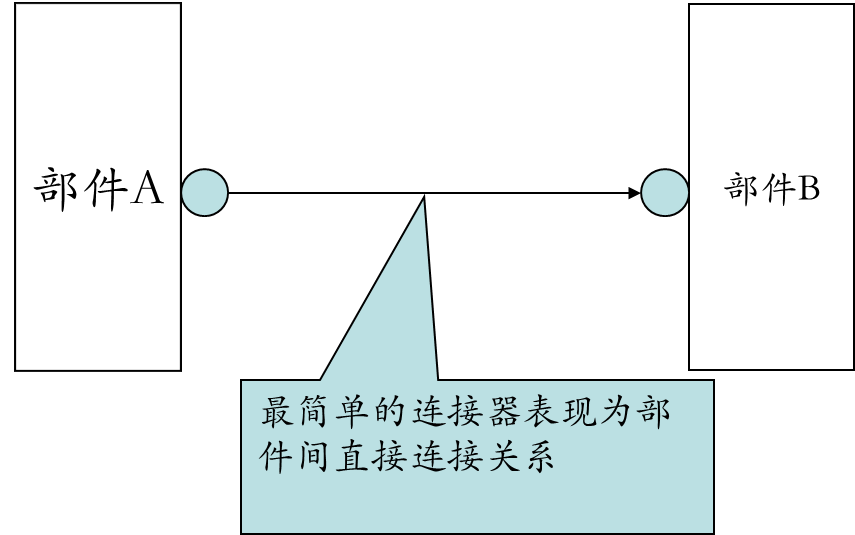
（1）连接器的层次性

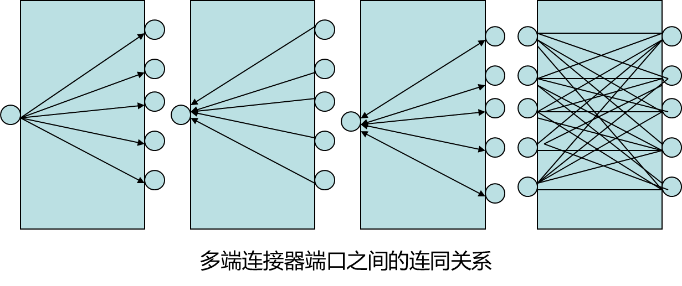
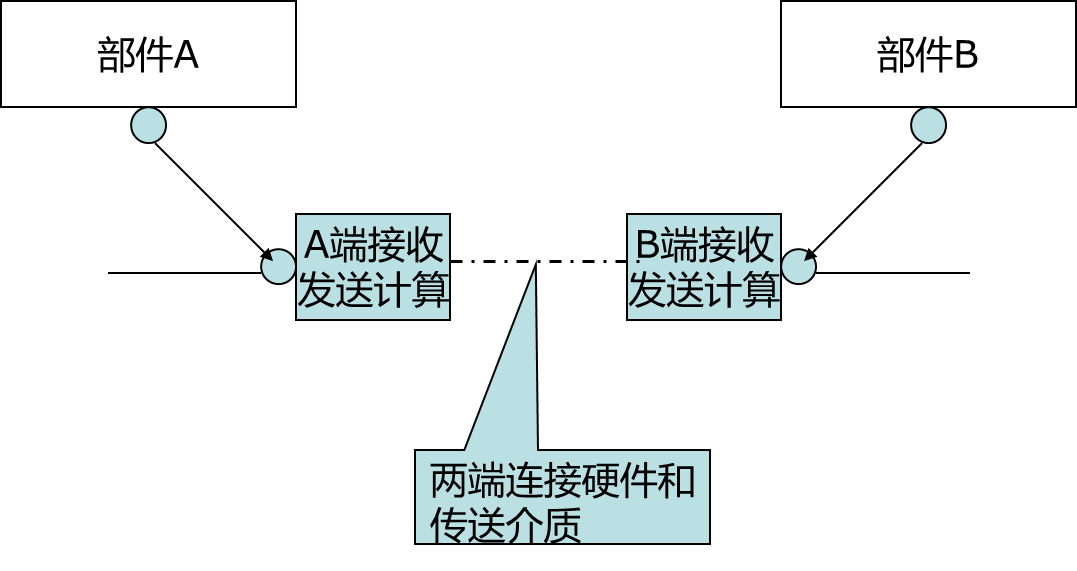
连接器的最基本形式是由计算机硬件提供的。

描述层次太低，使用这些基础原始的连接形式直接描述系统上层复杂的联系是不恰当的

软件体系结构层次模型的每个层次中都有关于连接的概念，他们都是在基本的连接形式上经过复合和概念提升而产生的，并且成为了复杂系统不同层次连接设计的出发点。

（2）连接器的表达形式





（3）连接器的特性

连接的关系 1：1、1：n、n：1、m：n

连接的角色和方向

连接的交互方式（信号、语言式）

连接的可扩展性

连接的互操作性

连接请求响应特性

连接请求的处理策略

连接的代价、处理速度或能力

**Chapter 6 体系结构的一般描述**

**通用的一般的描述方法**

（1）主子程序

系统结构被映射为主程序和一系列具有调用关系的子过程的集合。

（2）数据抽象和面向对象

数据抽象和面向对象设计是在主程序和子过程设计基础上建立和发展起来的重要的软件描述方法。

数据抽象是面向对象设计的理论基础

类和对象是该描述方法的基础粒度，而非模块或者包。

**本质上没有逃出主子程序的思想**

面向对象已经成为大多数软件系统设计的基础和出发点，一些重要的软件设计思想都是在此基础上提出来的

（3）层次结构

层次化是一种复杂系统设计的带普遍性的设计原则。

层次化设计来源于2个原因

事物总是从最简单的、带基础的层次开始发生的

来自众多复杂软件设计的实践，几乎所有的系统都是从层次化结构建立起来的

**理论的形式化方法**

（1）Z Notation

Z Notation使用标准的逻辑操作符和集合操作符以及他们标准的常规语义。

集合及其构造形式

集合操作符号

逻辑操作符号

使用Z Notation可以描述数学对象的模型。这些对象模型与程序计算对象的相似之处是选择Z作为体系结构和软件工程描述语言的原因。

在研究文献和报告中，大量的采用Z Notation作为体系结构的形式化描述。

（2）CSP

CSP提供丰富的描述进程行为的语义集，主要包括语义、实施、行为、组合、并发、通信等，被广泛应用于分布式设计的建模和分析中。

（3）类属理论

类属理论是一种表达对象关系的数学语言。最初由Samuel Eilenberg 和Sanders Maclane提出的。

提供了概念划分的统一性

类属是由一个对象集合和一个关联两个对象的箭头集合构成的。

对象是以某种逻辑表达的形式规范，箭头定义了这些规范之间的关系。

软件研究者把他看成是表达抽象和依赖关系的工具

在体系结构的部件和连接器描述中都被广泛采用

（4）化学抽象机模型

一个CHAM是从定义分子和分子溶液开始的。

分子构成了CHAM的基本元素，溶液是分子的聚合并被解释为CHAM的状态。

一个CHAM还包含转换规则T、T1……，他们定义了一个转换关系s->s1 ,指示了溶液可以发展（也就是状态变化）的方式。

借助于称为膜的封装结构，任何溶液可以被看成是相对于其他溶液的分子，转换可以被约束和局限在一个膜内。

被称为闭锁的可逆操作用来从溶液中选择提取分子，并把溶液的剩余部分放入一个膜内。膜是半渗透的，只允许特定的分子进入或离开膜。

**软件体系结构集成环境**

（1）UniCon

UnicCon是基于部件和连接器的。

体系结构的描述模型是由基于可识别的称为部件和连接器的个体元素构成的。

部件和连接器元素都具有类型、规范和实现。

（2）Darwin

（3）Wright

Wright体系结构描述语言用于描述软件系统的体系结构

可以描述体系结构的风格、系统族、体系结构实例和单个系统

提供对计算构件和连接件的描述

交互关系

构件间的交互关系是指模块之间的通讯

Wright体系结构描述语言基于交互

Wright语言的描述

根据构件、连接件和配置等基本体系结构元素的抽象而构造系统

构件作为计算部件，连接件形式化为交互模式

（4）ACME

ACME将系统描述成通过连接器实现交互关系的部件的图

采用图形化编辑方式直接描述构件和连接件

**设计模式的特点与应用**

设计模式是从许多优秀的软件系统中总结出的成功的、能够实现可维护性复用的设计方案，使用这些方案将避免我们做一些重复性的工作，而且可以设计出高质量的软件系统。

**The “4+1” View Model of Software Architecture**

Logical view、Process view、Development view、Physical view、Scenario

**体系结构风格**

数据流、调用和返回、独立构件、虚拟机、数据中心

**一些面向对象的设计法则**

优先使用（对象）组合，而非（类）继承

针对接口编程，而非（接口的）实现

开放－封闭法则（OCP）

Liskov替换法则（LSP）

1. ACM是因为阿兰·麦席森·图灵做出了什么重要贡献而设立了图灵奖。
2. 从1966年到今天2018年，共有多少位图灵奖得主，他们大部分人的第一专业是什么？
3. 在各大期刊会议中，我们经常能看到引文索引的概念，那么请问什么是引文索引。
4. 刊物和会议分类分为哪几类
5. 三大科技文献检索系统是哪几个