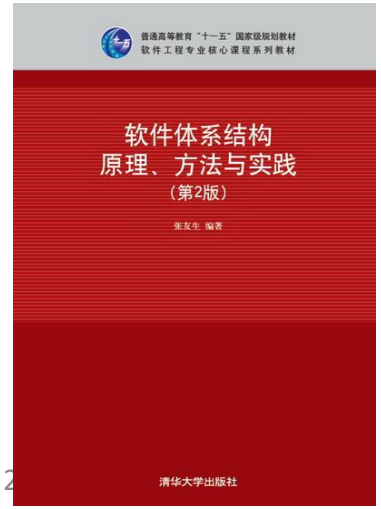
第2章 软件体系结构建模

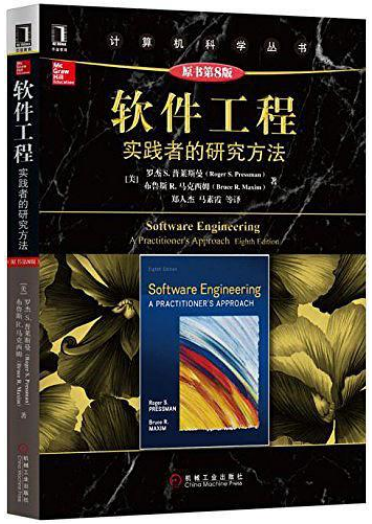
授课教师： 应时

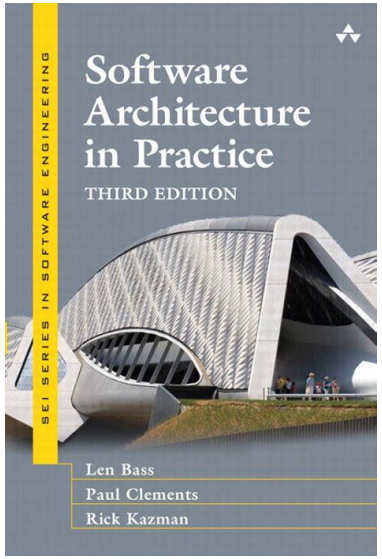
2019-10-15

参考书

- 张友生编著，软件体系结构原理、方法与实践（第2版），清华大学出版社，2014年1月。
- [美]罗杰 S. 普莱斯曼，译者: 郑人杰，软件工程：实践者的研究方法（原书第8版），机械工业出版社，2016年11月。
- 温昱，软件架构设计（第2版）—程序员向架构师转型必备，电子工业出版社，2012年07月。
- Len Bass, Paul Clements, Rick Kazman, Software Architecture in Practice (3rd Edition), Addison-Wesley Professional, Sep 25, 2012.







参考书

- 软件建模与设计：UML、用例、模式和软件体系结构，原书名：Software Modeling & Design: UML, Use Cases, Patterns, & Software Architectures，作者：(美)Hassan Gomaa，译者： 彭鑫、吴毅坚、赵文耘，机械工业出版社，2014 年8月。
- 软件架构建模和仿真：Palladio 方法，原书名：Modeling and Simulating Software Architectures: The Palladio Approach，作者： （德）拉尔夫•H.雷乌斯纳（Ralf H.Reussner） 等，译者： 李必信 等，机械工业出版社，2018 年9月。

第2章 软件体系结构建模

- 研究软件体系结构的首要问题是如何表示软件体系结构，即如何对软件体系结构建模。
- 根据建模的侧重点不同，可以将软件体系结构的模型分为5种：
  - 结构模型
  - 框架模型
  - 动态模型
  - 过程模型
  - 功能模型。
- 在这5个模型中，最常用的是结构模型和动态模型。

（1）结构模型

- 这是一个最直观、最普遍的建模方法。这种方法用体系结构的构件、连接件和其它概念，来刻画结构；并力图通过结构，来反映系统的重要语义内容，包括：
  - ✓ 系统的配置
  - ✓ 约束
  - ✓ 隐含的假设条件
  - ✓ 风格
  - ✓ 性质等
- 研究结构模型的核心是体系结构描述语言。

（2）框架模型

- 框架模型与结构模型类似，但它不太侧重描述结构的细节，而更侧重于整体的结构。
- 框架模型主要以一些特殊的问题为目标，建立只针对和适应这类问题的结构。

（3）动态模型

- 动态模型是对结构或框架模型的补充，研究系统的“大粒度”的行为及其性质。
- 动态模型描述的是：系统总体结构的配置或演化、通信通道或计算过程的建立或拆除等方面的行为。

- （4）过程模型
  - 过程模型研究构造系统的步骤和过程，因而其结构是遵循某些过程脚本的结果。
- （5）功能模型。
  - 功能模型认为体系结构是由一组功能构件，按层次组成，下层向上层提供服务。它可以看作是一种特殊的框架模型。
- **软件体系结构的5种模型各有所长，将5种模型有机地统一在一起，形成一个完整的模型，来刻画软件体系结构更合适。**

2.1 “4+1”视图模型

- Kruchten在1995年提出了一个“4+1”的视图模型。

- Philippe.B. Kruchten, The 4+1 View Model of architecture, IEEE Software, 1995, Volume:12, Issue:6, Pages: 42-50.

- Abstract: the 4+1 View Model organizes a description of a software architecture using five concurrent views, each of which addresses a specific set of concerns. Architects capture their design decisions in four views and use the fifth view to illustrate and validate them.

2.1 “4+1”视图模型

- Kruchten提出的“4+1”视图模型对软件体系结构的描述，是从以下5个不同的视角进行的：
  - 逻辑视图
  - 开发视图
  - 进程视图
  - 物理视图
  - 场景视图
- 每一个视图只关心系统的一个侧面，5个视图结合在一起，才能反映系统的软件体系结构的全部内容。

2.1 “4+1”视图模型

- “4+1”视图模型如图2-1所示。
  - 不同的软件体系结构视图承载不同的软件体系结构设计决策，支持不同的目标和用途。

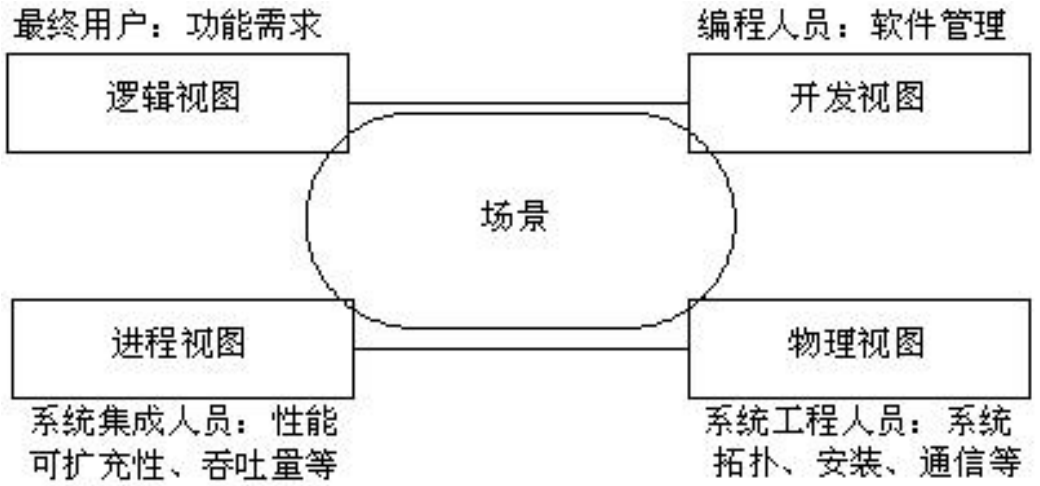


图2-1 “4+1”视图模型

- 逻辑视图：当采用面向对象的设计方法时，逻辑视图即对象模型。
- 开发视图：描述软件在开发环境下的静态组织。
- 进程视图：描述系统的并发和同步方面的设计。
- 物理视图：描述软件如何映射到硬件，反映系统在分布方面的设计。

2.1 “4+1”视图模型

➢ 逻辑视图（logic view）

- 主要支持系统的功能与服务需求，即系统提供给最终用户的服务。
- 逻辑视图关注功能，不仅包括用户可见的功能，还包括为实现用户功能而必须提供的“辅助功能”。它们可能是逻辑层、功能模块等。
- 在逻辑视图中，系统被分解成一系列的职责抽象，这些抽象主要源自问题领域。
- 这种分解不但可以用来进行功能分析，而且可用作标识在整个系统的各个不同部分的通用机制和设计元素。
- 可以用类图模型和对象图模型，来描述逻辑视图，并使用抽象、封装和继承等设计技术。

2.1 “4+1”视图模型

- 可以从Booch标记法中，导出逻辑视图的标记法。只不过此时要从体系结构级的范畴，来考虑这些符号，用Rational Rose进行体系结构设计。
- 图2-2是逻辑视图中使用的符号集合。

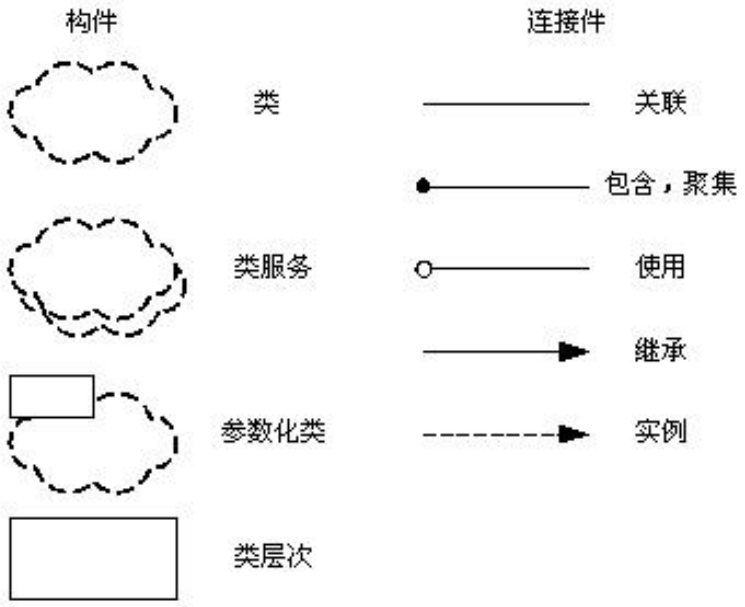


图2-2 逻辑视图中使用的标记符号

注：现在应使用UML进行描述，并使用IBM Rational工具。

2.1 “4+1”视图模型

- 类图用于表示类的存在，以及类与类之间的相互关系，是从系统构成的角度，来描述正在开发的系统。
- 一个类的存在不是孤立的，类与类之间以不同方式互相合作，共同完成某些系统功能。
- 关联关系表示两个类之间存在着某种语义上的联系，其真正含义要由附加在横线之上的一个短语，予以说明。

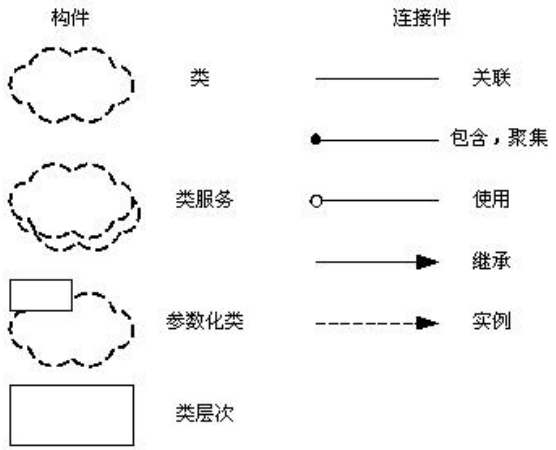


图2-2 逻辑视图中使用的标记符号

2.1 “4+1”视图模型

- 在表示包含关系的图符中，带有空心圆的一端表示整体，相反的一端表示部分。
- 在表示使用关系的图符中，带有空心圆的一端连接在请求服务的类，相反的一端连接在提供服务的类。
- 在表示继承关系的图符中，箭头由子类指向基类。

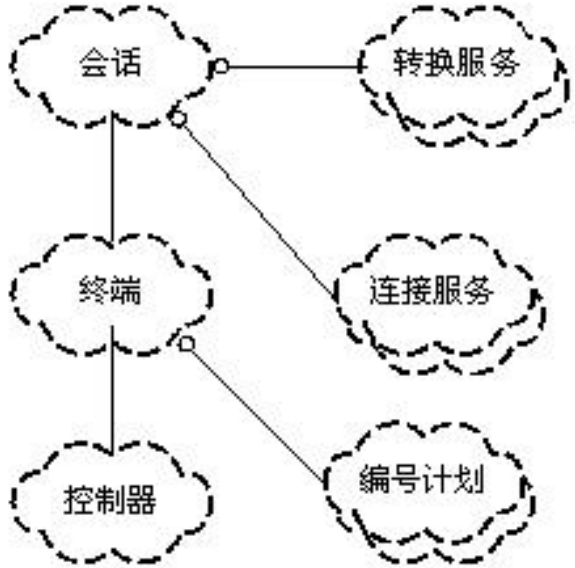


图2-3 某通信系统体系结构逻辑视图

2.1 “4+1”视图模型

- 逻辑视图中使用的风格为面向对象的风格。
- 逻辑视图设计中要注意的主要问题是：要保持一个单一的、内聚的对象模型，贯穿整个系统。

2.1 “4+1”视图模型

- 例如，图2-3是某通信系统体系结构（ACS）中的主要类。
  - ACS的功能是在终端之间建立连接，这种终端可以是电话机、主干线、专用线路、特殊电话线、数据线或ISDN线路等。
  - 不同的线路，由不同的线路接口卡，进行支持。

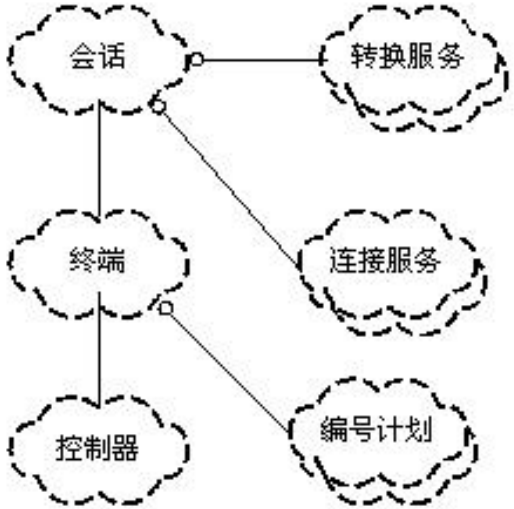
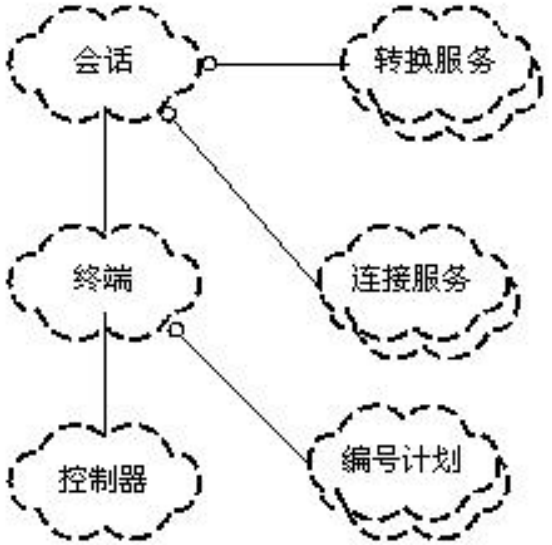
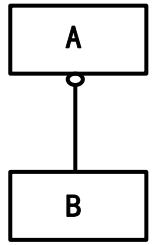


图2-3 某通信系统体系结构逻辑视图

2.1 “4+1”视图模型

- 控制器对象的作用是：译码并把所有符号，加入到线路接口卡中。
- 终端对象的作用是：保持终端的状态，代表本条线路的利益，参与协商服务。
- 会话对象的作用是：代表在终端上进行的会话，它使用转换服务（目录、逻辑地址映射到物理地址，路由等）和连接服务，在终端之间建立语音路径。



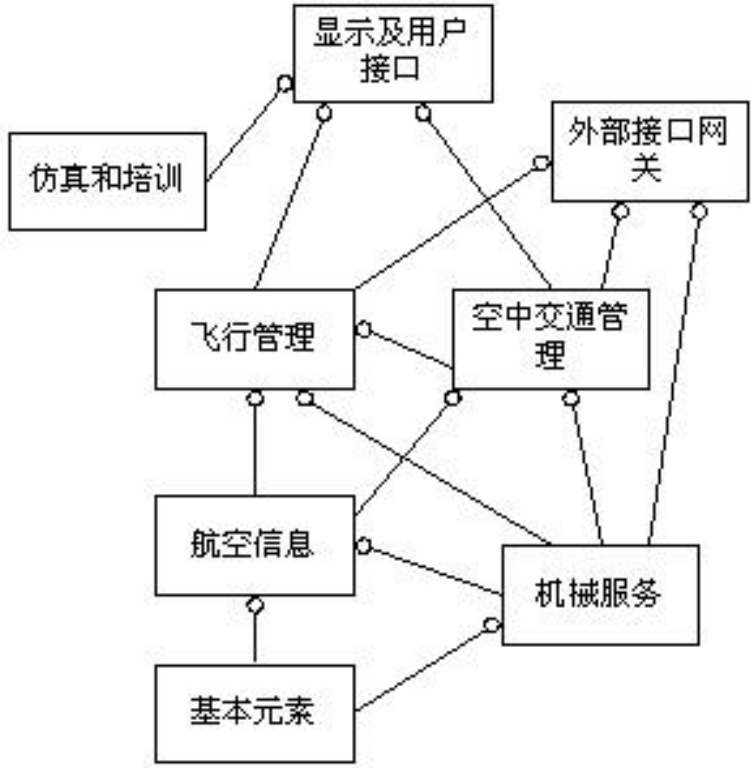


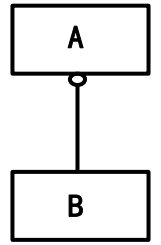
注：A使用B

图2-3 某通信系统体系结构逻辑视图

2.1 “4+1”视图模型

- 对于规模更大规模的系统，体系结构层次中包含数十，甚至数百个类。
- 例如，图2-4是一个空中交通管制系统的顶级类图，该图包含了8组类。





注：A使用B

图2-4 空中交通管制系统的一级类图

2.1 “4+1”视图模型

➢ 开发视图（development view）

- 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。
- 开发视图关注程序包，不仅包括要编写的源程序，还包括可以直接使用的第三方SDK和现成框架、类库，以及开发的系统将运行于其上的系统软件或中间件。
- 开发视图和逻辑视图之间可能存在一定的映射关系：比如逻辑层一般会映射到多个程序包等。
- 在确定了软件包含的所有元素之后，完整描述开发人员应该了解和关注的元素及其结构关系。
- 可以在确定每个元素之前，先列出开发视图的原则。

2.1 “4+1”视图模型

- 开发视图（development view）
- **开发视图要考虑软件内部在开发方面的技术性需求，如**
  - **软件开发的容易性**
  - **软件的可重用性**

2.1 “4+1”视图模型

- 与逻辑视图一样，可以使用Booch标记法中某些符号，来表示开发视图，如图2-5所示。

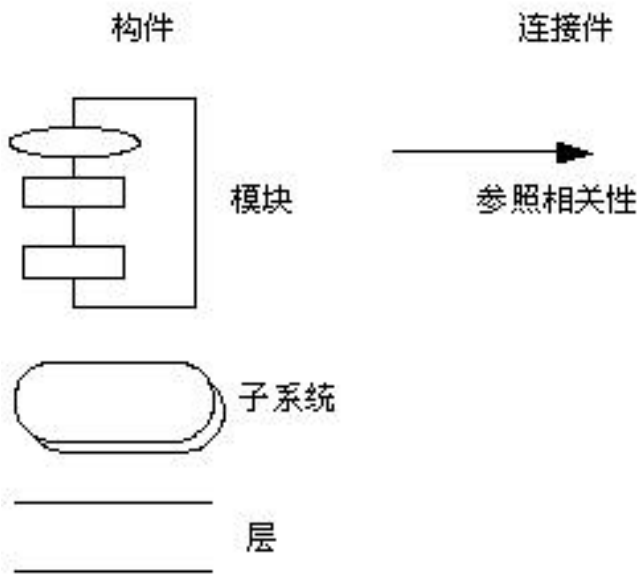


图2-5 开发视图中使用的标记符号

2.1 “4+1”视图模型

- 在开发视图中，最好采用4-6层子系统。
- 每个子系统仅仅能与同层或更低层的子系统通讯，这样可以使每个层次的接口既完备又精练，避免了各个模块之间很复杂的依赖关系。
- 设计时要做到：对于各个层次，层次越低，应使其通用性越强。
  - 这样，可以保证应用程序的需求发生改变时，所做的改动最小。
- 开发视图所用的风格通常是层次结构风格。

2.1 “4+1”视图模型

- 例如，图2-6表示的是空中交通管制系统的五层结构图。
- 图2-6开发视图与图2-4逻辑视图是相关的。
  - 第1层和第2层组成了一个领域无关的分布式基础设施，贯穿于整个产品线中，并且与硬件平台、操作系统或数据库管理系统等无关。
  - 第3层增加了空中交通管制系统的框架，以形成一个领域特定的软件体系结构。
  - 第4层使用该框架建立一个功能平台。
  - 第5层则依赖于具体客户和产品，包含了大部分用户接口以及与外部系统的接口。

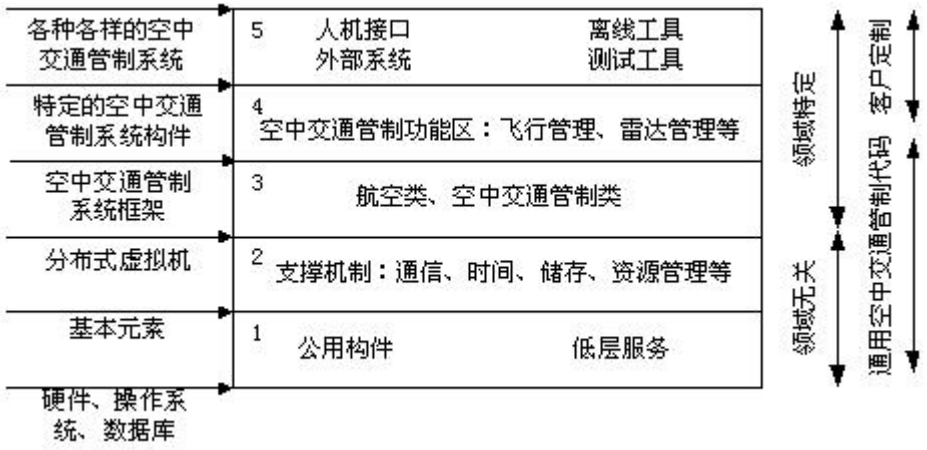


图2-6 空中交通管制系统的五层结构—开发视图

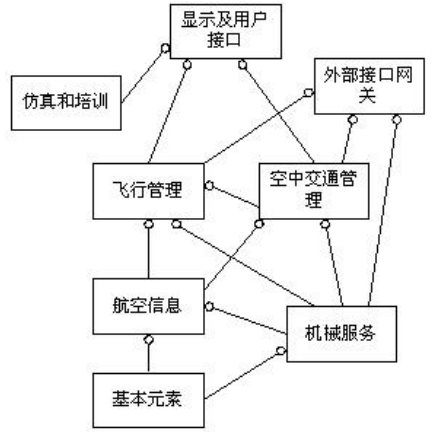


图2-4 空中交通管制系统的一级类图--逻辑视图

2.1 “4+1”视图模型

➢ 进程视图（process view）

- 进程视图也称为并发视图，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。
- 进程视图关注进程、线程、对象等运行时概念，以及相关的并发、同步、通信等问题。
- 进程视图和开发视图的关系：开发视图一般偏重程序包在编译时期的静态依赖关系，而这些程序运行起来之后会表现为对象、线程、进程。进程视图比较关注的正是这些运行时单元的交互问题。

2.1 “4+1”视图模型

- 进程视图（process view）

- **进程视图强调：**
  - **并发性**
  - **分布性**
  - **容错能力**
  - **逻辑视图中的主要抽象，是如何适合于进程结构的。**

2.1 “4+1”视图模型

- 进程视图也定义逻辑视图中的各个类的操作，具体是在哪一个线程中被执行的。
- 在最高层抽象中，进程结构可以看作是构成一个执行单元的一组任务。
- 进程可以被看成一系列独立的，通过逻辑网络相互通信的程序。它们是分布的，通过总线或局域网、广域网等硬件资源连接起来。

2.1 “4+1”视图模型

- 通过进程视图，可以从进程角度，了解一个目标系统最终执行情况。
  - 例如在以计算机网络作为运行环境的图书管理系统中，服务器需要对那些对应于各个不同客户机的进程，进行管理，并决定某个特定进程（如查询子进程、借还书子进程）的唤醒、启动、关闭等操作，从而控制整个运行环境协调有序地工作。

2.1 “4+1”视图模型

- 通过扩展Booch对Ada任务的表示法，来表示进程视图。
- 从体系结构角度来看，进程视图的标记元素如图2-7所示。

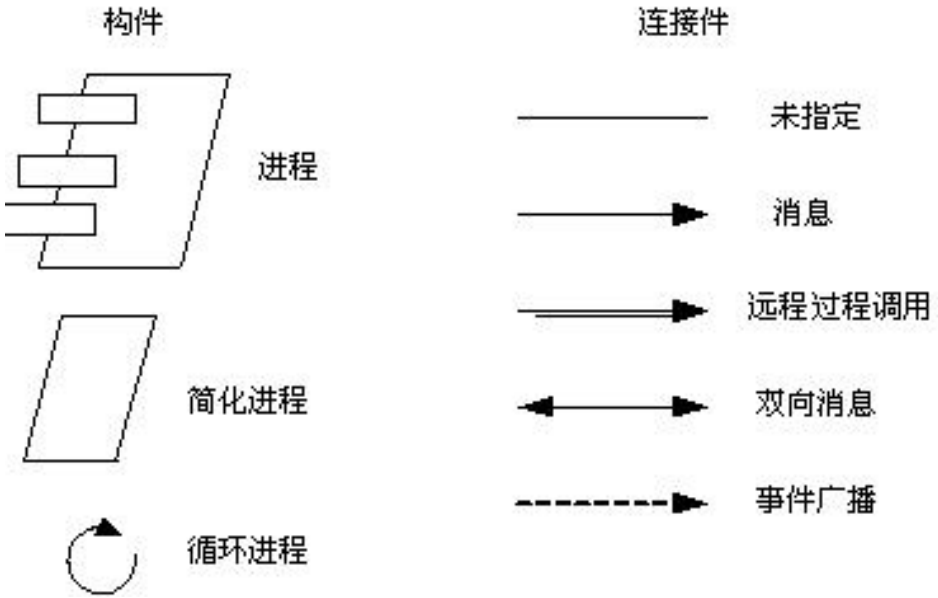


图2-7 进程视图中使用的标记符号

2.1 “4+1”视图模型

- 有很多风格适用于进程视图，如管道和过滤器风格、客户/服务器风格（多客户/单服务器，多客户/多服务器）等。
- 图2-8是2.1.1节中的ACS系统的局部进程视图。

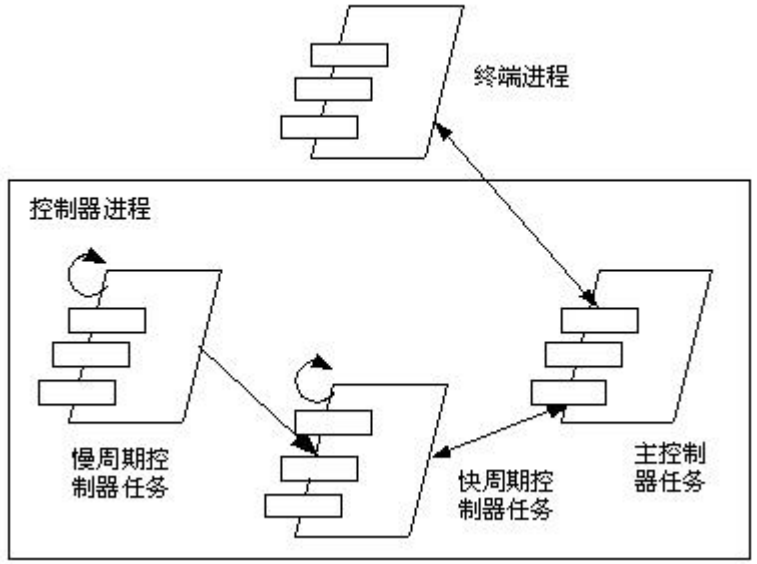


图2-8 ACS系统的进程视图（局部）

2.1 “4+1”视图模型

- 在图2-8中，所有终端均由同一个终端进程进行处理，由其输入队列中的消息驱动。
- 控制器对象在组成控制器进程的3个“控制器任务”进程之一中执行。
  - 慢循环周期（200ms）任务进程，扫描所有被挂起（suspend）的终端，把任何一个活动的终端，置入快循环周期（10ms）任务进程的扫描列表。
  - 快循环周期任务进程，检测任何显著的状态改变，并把改变的状态，传递给主控制器任务进程，
  - 主控制器任务进程解释改变，通过消息与相应的终端进程，进行通讯。
- 通过共享内存，实现在控制器的进程间传递消息。

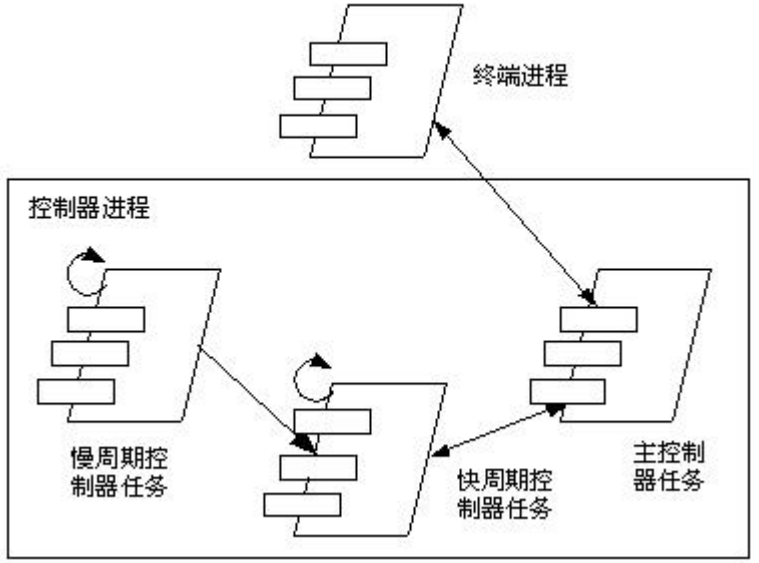


图2-8 ACS系统的进程视图（局部）

2.1 “4+1”视图模型

➢ 物理视图（physical view）

- 物理视图关注“目标程序及其依赖的运行库和系统软件”最终如何被安装或部署到物理机器上，以及如何部署机器和网络，以配合软件系统的可靠性、可伸缩性等要求。
- 物理视图和进程视图的关系：物理视图重视目标程序的静态位置问题，而进程视图特别关注目标程序的动态执行情况；物理视图是综合考虑软件系统和整个IT系统相互影响的架构视图。
- 物理视图决定拓扑结构、系统安装、通讯等方面的问题。

2.1 “4+1”视图模型

- 物理视图（physical view）
- 当软件运行于不同的节点上时，各视图中的构件，都直接或间接地对应于系统的不同节点上。
- 从软件到节点的映射，要有较高的灵活性。

2.1 “4+1”视图模型

- 大型系统的物理视图可以与进程视图的映射一起，以多种形式出现，也可单独出现。
- 图2-9是物理视图的标记元素集合。

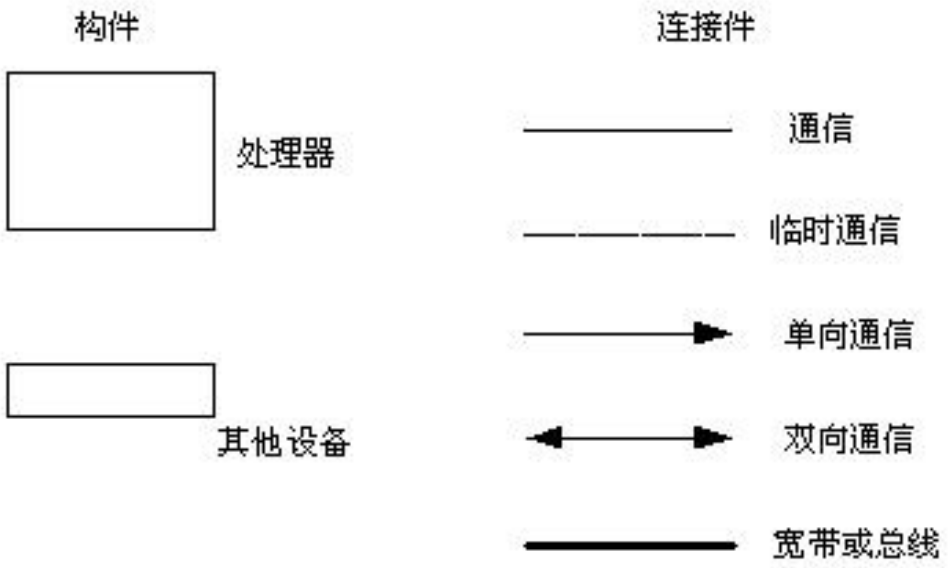


图2-9 物理视图中使用的标记符号

2.1 “4+1”视图模型

- 图2-10是某IT平台的可能硬件配置。
  - C、F和K是3个具有不同能力的计算机类型，支持3个不同的可执行文件。
- 图2-11和图2-12是进程视图到两个不同的物理视图映射，分别对应一个小型的ACS和大型的ACS。

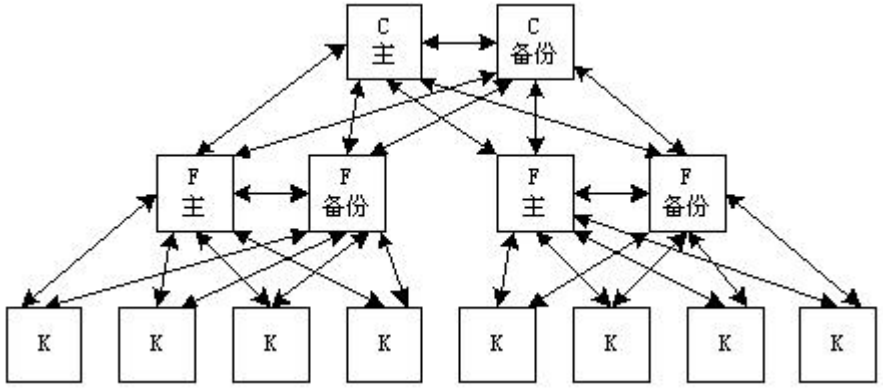


图2-10 某IT平台的物理视图

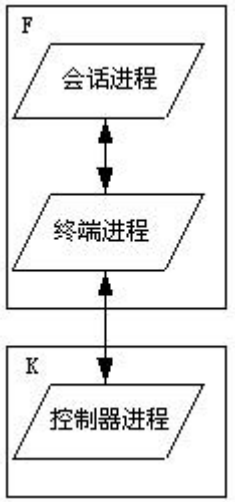


图2-11 具有进程分配的小型ACS系统的物理视图

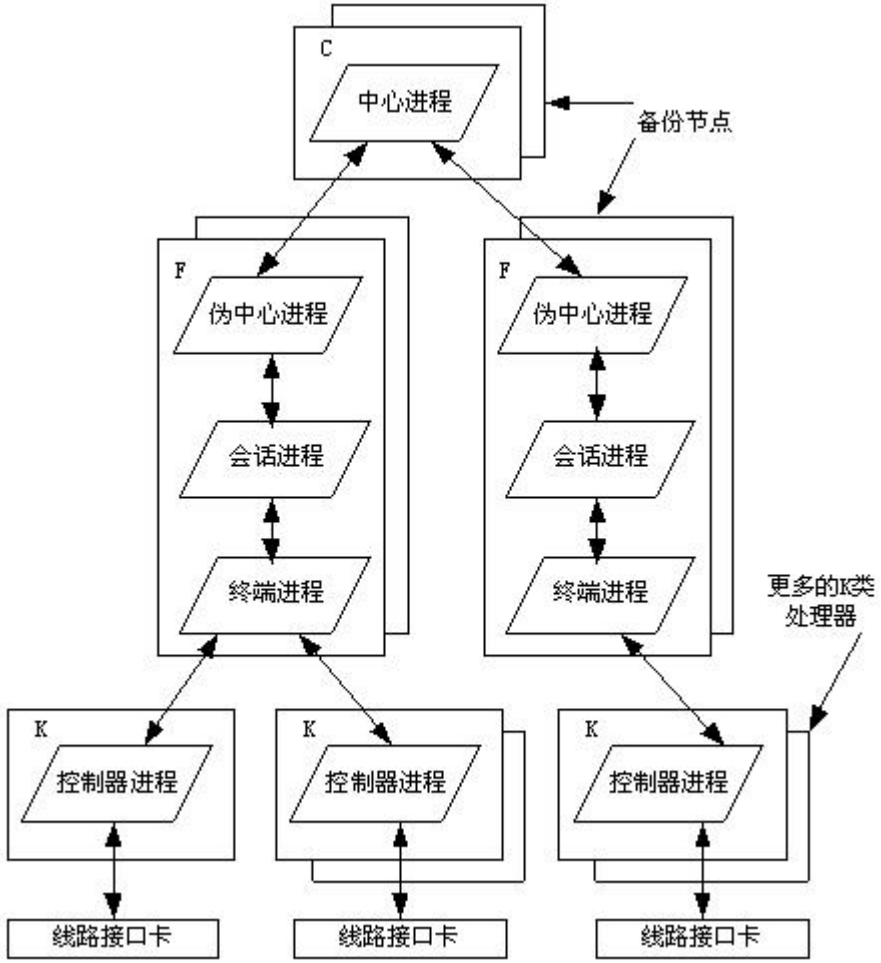


图2-12 具有进程分配的大型ACS系统的物理视图

2.1 “4+1”视图模型

➢ 场景（scenarios）

- 场景可以看作是那些重要系统活动的抽象，它使4个视图有机联系起来，从某种意义上说场景是最重要的需求抽象。
- 在开发体系结构时，它可以帮助设计师，找到体系结构的构件和它们之间的交互协作关系。
- 同时，也可以用场景，来分析一个特定的视图，或描述不同视图构件间是如何相互协作的。
- 场景可以用文本表示，也可以用图形表示。

2.1 “4+1”视图模型

- 例如，图2-13是一个小型ACS系统的场景片段，相应的文本表示如下：

① 小王的电话控制器，检测到电话机从挂机到摘机状态的转变，并发送一个消息，以唤醒相应的终端对象。

② 终端在获得了一些必要的资源后，发出拨号音，通知控制器，可以开始拨号。

③ 控制器拨打号码，并传给终端。

④ 终端使用编号计划，分析号码。

⑤ 当拨号成功时，终端就打开一个会话。

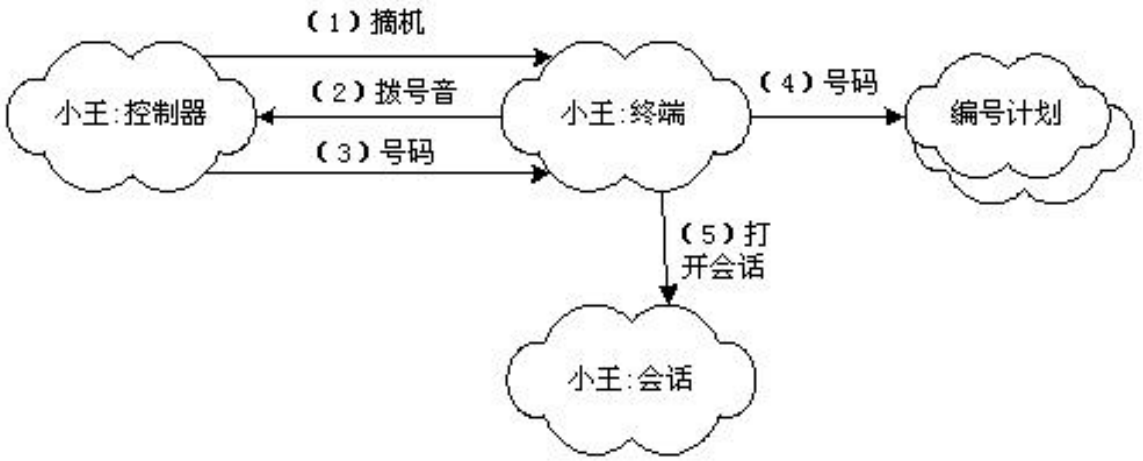


图2-13 本地呼叫场景的一个原型

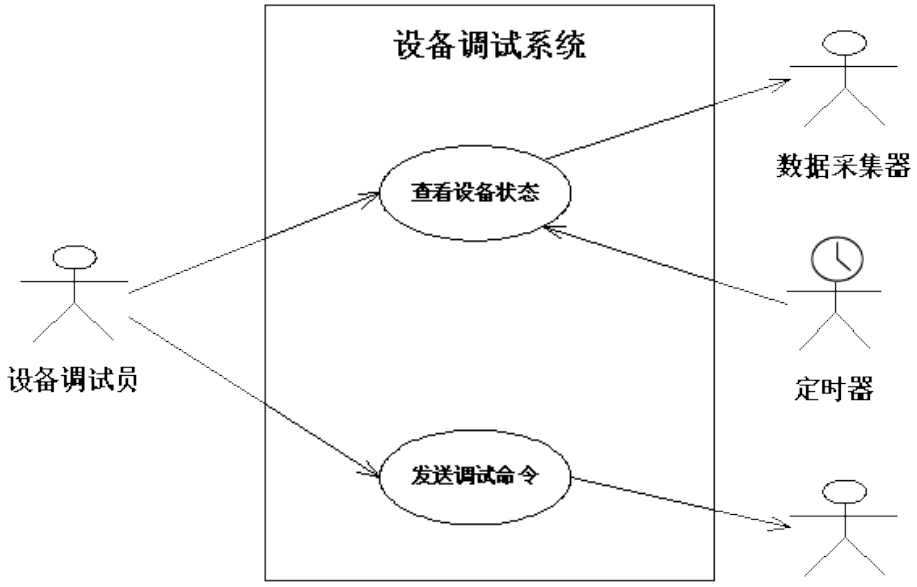
2.1 “4+1”视图模型

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

- 对于不同的软件系统来说，其所侧重的实体也有所不同。
  - 管理信息系统，比较侧重于从逻辑视图和开发视图，来描述系统。
  - 实时控制系统，则比较注重于从进程视图和物理视图，来描述系统。

2.1 “4+1”视图模型

- **设计案例：设计某型号设备调试系统的软件体系结构。**
  - **设备调试员通过使用该系统，可以察看设备状态（设备的状态信息由专用的数据采集器实时采集）、发送调试命令。**
  - **该系统的用例图如左下图所示。该系统的需求如右下表所示。**



设备调试系统的用例图

设备调试系统的需求

非功能需求			功能需求
约束	运行期质量属性	开发期质量属性	
程序的嵌入式部分必须用C语言开发 一部分开发人员没有嵌入式开发经验	高性能	易测试性	察看设备状态 发送调试命令

2.1 “4+1”视图模型

- 不同视图着眼于满足不同的设计需求和设计约束
  - 逻辑视图：设计满足功能需求的软件体系结构
  - 开发视图：设计满足开发期质量属性的软件体系结构
  - 进程视图：设计满足运行期质量属性的软件体系结构
  - 物理视图：设计满足部署要求的软件体系结构

2.1 “4+1”视图模型

- **逻辑视图：设计满足功能需求的架构**
  - 首先根据功能需求进行初步设计，进行大粒度的职责划分。如右图所示。
  - 应用层负责设备状态的显示，并提供模拟控制台，供用户发送调试命令。应用层使用通讯层和嵌入层进行交互，但应用层不知道通讯的细节。
  - 通讯层负责在RS232协议之上，实现一套专用的“应用协议”。当应用层发送来包含调试指令的协议包，由通讯层负责按RS232协议，将之传递给嵌入层。当嵌入层发送来原始数据，由通讯层将之解释成应用协议包，发送给应用层。
  - 嵌入层负责对调试设备的具体控制，以及高频度地从数据采集器读取设备状态数据。设备控制指令的物理规格被封装在嵌入层内部，读取数采器的具体细节也被封装在嵌入层内部。

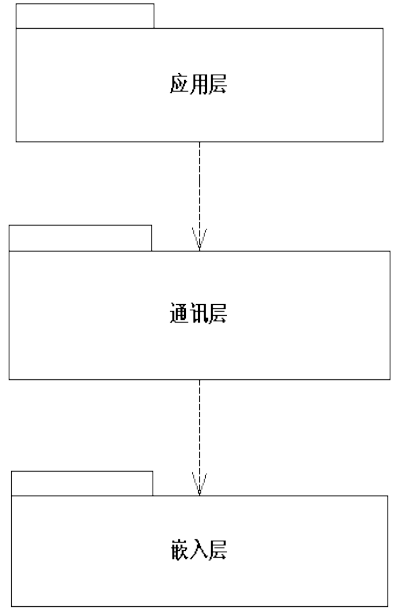


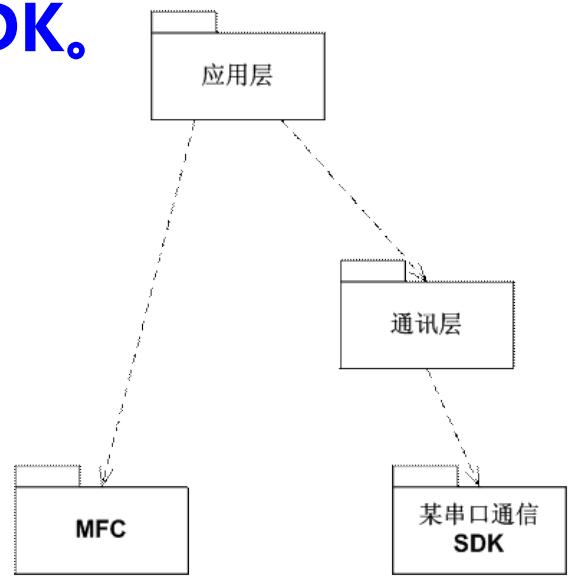
图 设备调试系统架构的逻辑视图

2.1 “4+1”视图模型

- **开发视图：设计满足开发期质量属性的架构**
  - 软件架构的开发视图应当为开发人员提供切实的指导。任何影响全局的设计决策都应由架构设计来完成，这些决策如果"漏"到了后边，最终到了大规模并行开发阶段才发现，可能造成"程序员碰头临时决定"的情况大量出现，软件质量必然将下降甚至导致项目失败。
  - 其中，采用哪些现成框架、哪些第三方SDK、乃至哪些中间件平台，都应该考虑是否由软件体系结构的开发视图确定下来。

2.1 “4+1”视图模型

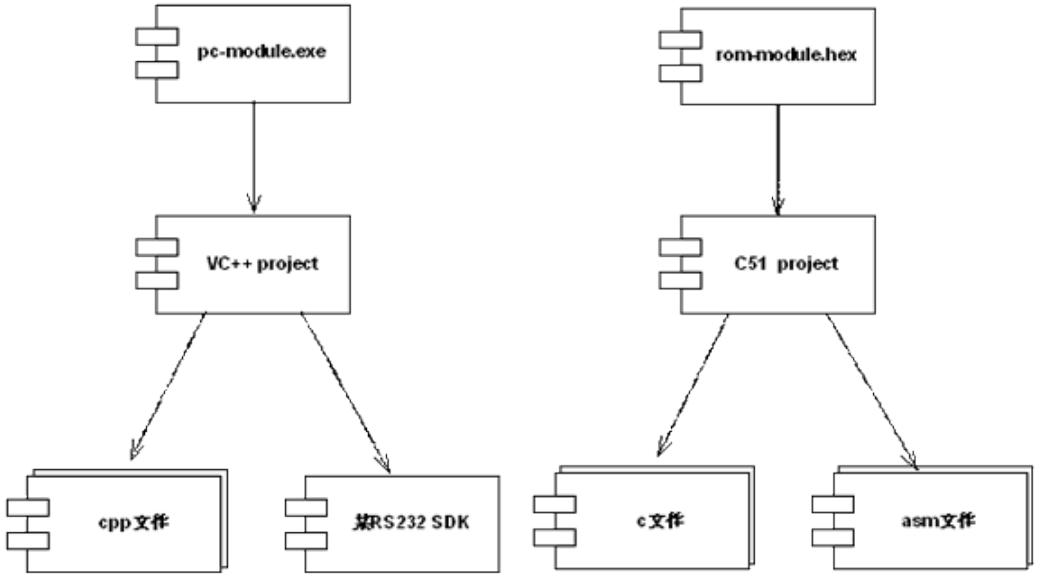
- 开发视图：设计满足开发期质量属性的架构
- 下图展示了设备调试系统的（一部分）软件架构开发视图：应用层将基于MFC设计实现，而通讯层采用了某串口通讯的第三方SDK。



图：设备调试系统架构的开发视图

2.1 “4+1”视图模型

- 开发视图：设计满足开发期质量属性的架构
- 约束应该是每个架构视图都应该关注和遵守的一些设计限制。例如，考虑到"一部分开发人员没有嵌入式开发经验"这条约束情况，架构师有必要明确说明系统的目标程序是如何编译而来的：
  - 右图展示了整个系统的桌面部分的目标程序pc-moduel.exe、以及嵌入式模块rom-module.hex是如何编译而来的。
- 这个全局性的描述无疑对没有经验的开发人员提供了实在感，利于更全面地理解系统的软件体系结构。



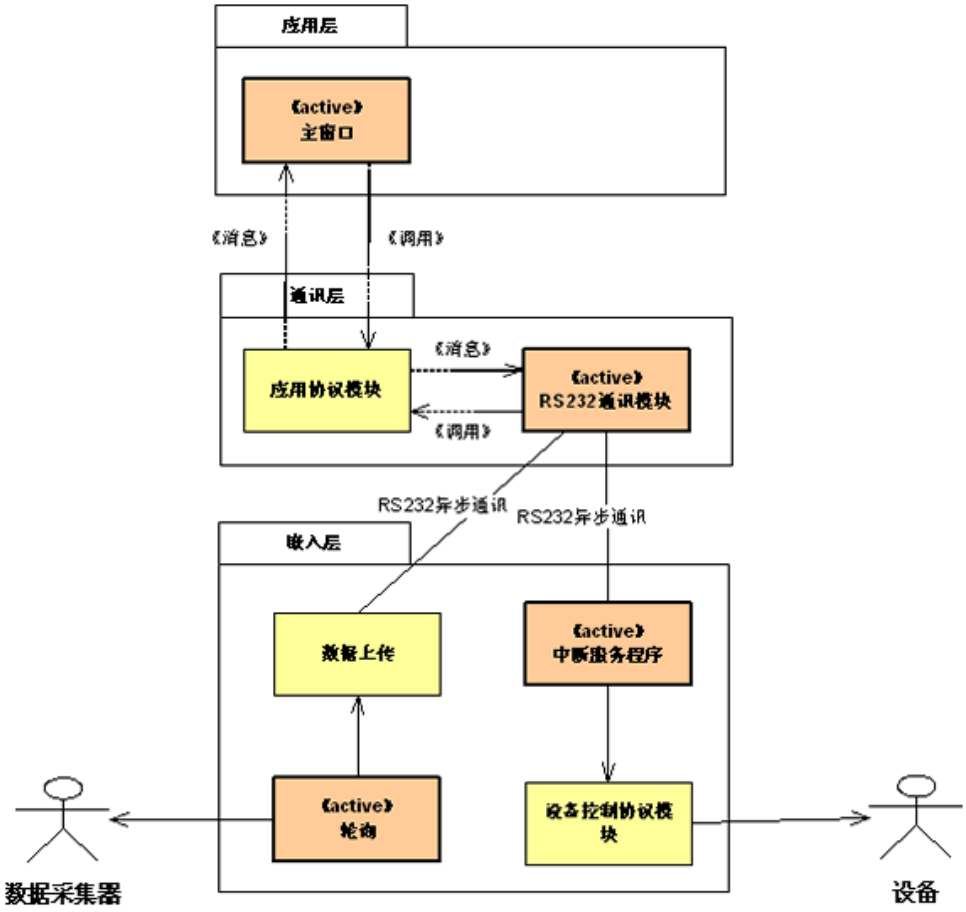
图： 设备调试系统架构的开发视图

2.1 “4+1”视图模型

- **进程视图：设计满足运行期质量属性的架构**
  - 性能是软件系统运行期间所表现出的一种质量水平，一般用系统响应时间和系统吞吐量来衡量。
  - 为了达到高性能的要求，软件架构师应当针对软件的运行时情况进行分析与设计，这就是软件架构的进程视图的目标。
  - 进程视图关注进程、线程、对象等运行时概念，以及相关的并发、同步、通信等问题。

2.1 “4+1”视图模型

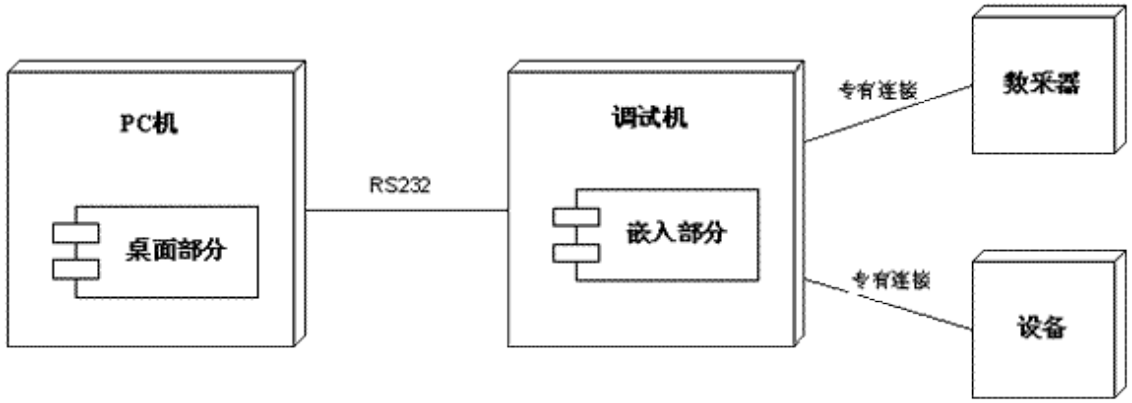
- 右图是设备调试系统架构的进程视图。架构师为了满足高性能需求，采用了多线程的设计：
  - 应用层中的线程代表主程序的运行，它直接利用了MFC的主窗口线程。无论是用户交互，还是串口的数据到达，均采取异步事件的方式处理，缩短了系统的响应时间。
  - 通讯层有独立的线程控制着“上上下下”的数据，并设置了数据缓冲区，使数据的接收和数据的处理相对独立，从而数据接收，不会因暂时的处理忙碌而停滞，增加了系统吞吐量。
  - 嵌入层的设计中，分别通过时钟中断和RS232口中断来激发相应的处理逻辑，达到轮询和收发数据的目的。



设备调试系统架构的处理视图

2.1 “4+1”视图模型

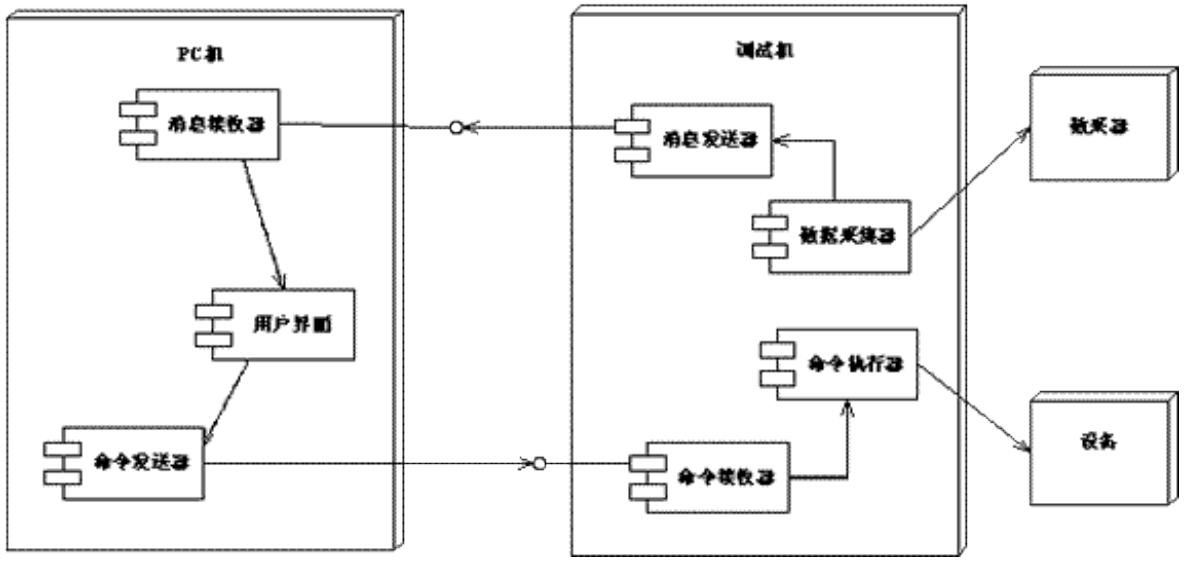
- **物理视图：和部署相关的架构决策**
  - 软件最终要驻留、安装或部署到硬件才能运行，而软件架构的物理视图关注"目标程序及其依赖的运行库和系统软件"最终如何安装或部署到物理机器，以及如何部署机器和网络来配合软件系统的可靠性、可伸缩性等要求。
  - 下图所示的物理架构视图表达了设备调试系统软件和硬件的映射关系。可以看出，嵌入部分驻留在调试机中（调试机是专用单板机），而PC机上是常见的桌面可执行程序的形式。



设备调试系统架构的物理视图

2.1 “4+1”视图模型

- 还可以根据具体情况的需要，通过物理架构视图，更明确地表达具体目标模块及其通讯结构，如图所示。



设备调试系统架构的物理视图

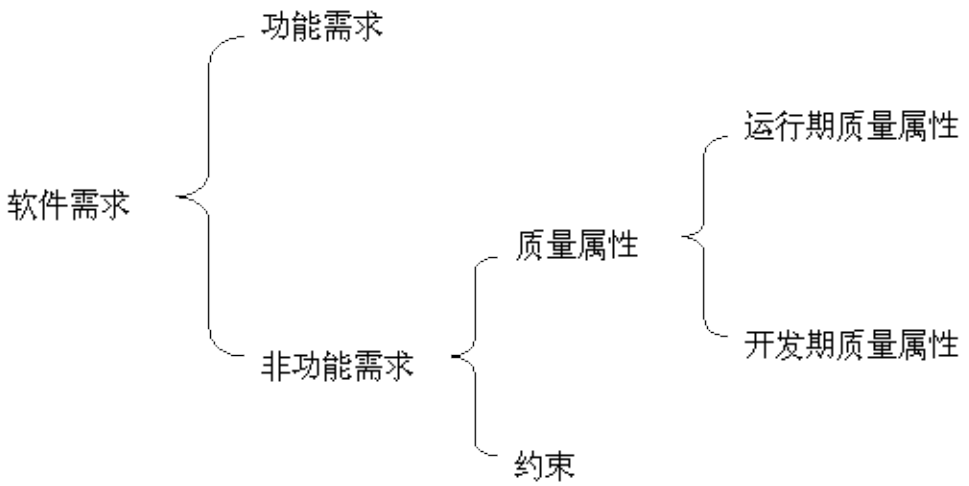
2.1 “4+1”视图模型

➢ 设计案例的小结

- 各类需求对架构设计的影响截然不同。
  - ✓ 深入理解软件需求分类的复杂性
  - ✓ 明确区分功能需求、约束、运行期质量属性、开发期质量属性等不同种类的需求。
- 案例展示了如何通过“4+1视图”方法，针对不同的需求，进行架构设计，从而确保重要的需求——被满足。

2.1 “4+1”视图模型

- 软件系统的需求种类也相当复杂，具体分类如左下图所示。
- 超市系统的需求：右下表列举了一个典型的超市系统的需求子集。



非功能需求			功能需求
约束	运行期质量属性	开发期质量属性	
项目预算有限	高性能	易理解性	提高收银效率
用户的平均电脑操作水平偏低	易用性	模块间松散耦合	任意商品项可单独取消
要求能在 Linux 上运行	……	……	通过收银终端的按键组合，可以使收银过程从“逐项录入状态”进入“选择取消状态”
开发人员分散在不同地点			……
……			

2.1 “4+1”视图模型

- **功能需求就是"软件有什么用，软件需要做什么"。**
  - **注意把握功能需求的层次性是软件需求的最佳实践。**
- **以该超市系统为例：**
  - **超市主管领导希望通过软件来"提高收银效率"。**
  - **这可能需要为收银员提供一系列功能来促成这个目的，比如供收银员使用的“任意商品项可单独取消”功能有利于提供收银效率（有人曾在超市有过被迫整单取消，然后一车商品重新扫描收费的痛苦经历）。**
  - **而具体到这个超市系统，系统分析员可能会决定要提供的具体功能为：通过收银终端的按键组合，可以使收银过程从"逐项录入状态"进入"选择取消状态"，从而取消某项商品。**

2.1 “4+1”视图模型

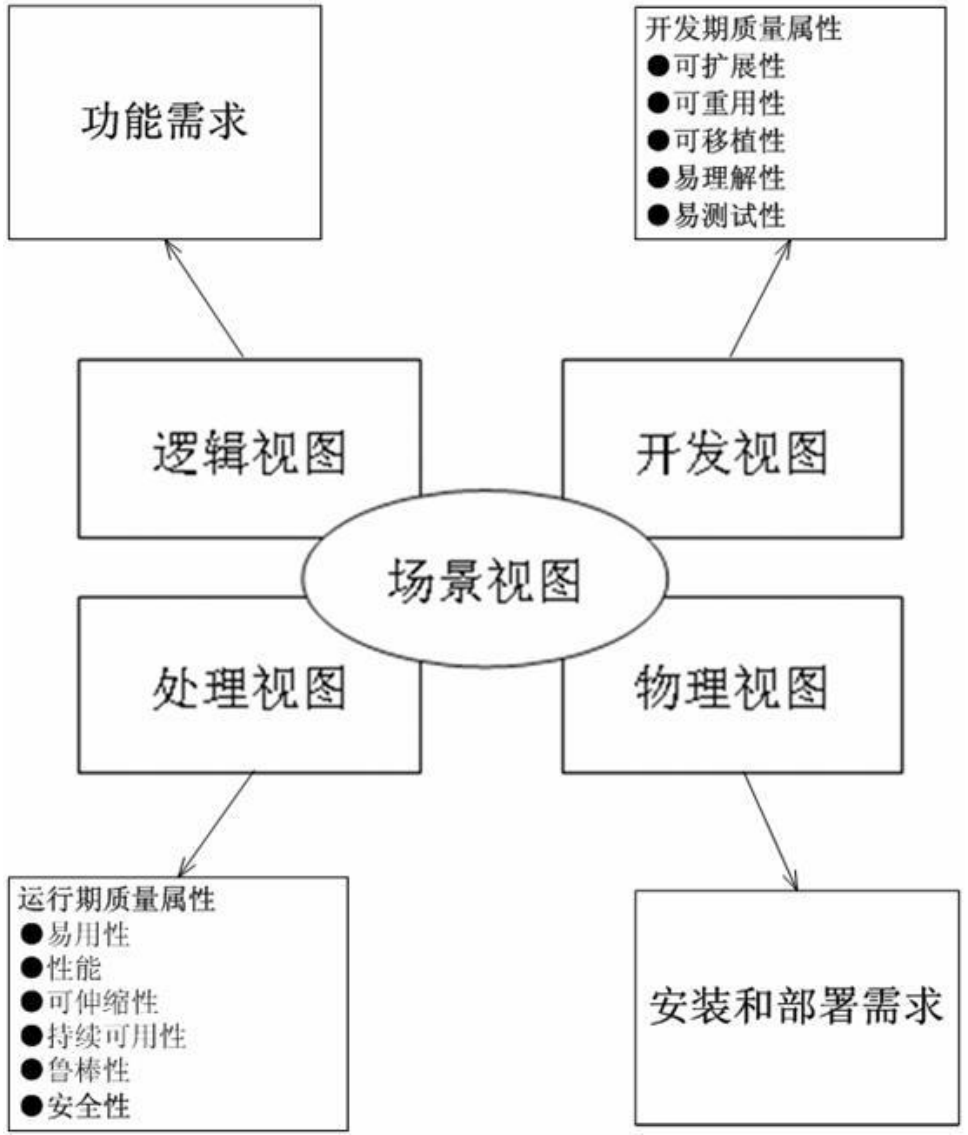
- 非功能需求可以分为如下三类：
- 约束：
  - 全面理解要设计的软件系统所面临的约束，是开发出用户满意软件的前提。
  - 约束性需求既包括企业级的商业考虑（例如"项目预算有限"），也包括最终用户级的实际情况（例如"用户的平均电脑操作水平偏低"）；既可能包括具体技术的明确要求（例如"要求能在Linux上运行"），又可能需要考虑开发团队的真实状况（例如"开发人员分散在不同地点"）。
  - 这些约束性需求当然对架构设计影响很大，比如受到"项目预算有限"的限制，架构师就不应选择昂贵的技术或中间件等，而考虑到开发人员分散在不同地点"，就更应注重软件模块职责划分的合理性、松耦合性等等。

2.1 “4+1”视图模型

- 非功能需求又可以分为如下三类：
- **运行期质量属性：**
  - 这类需求主要指软件系统在运行期间表现出的质量水平。
  - 运行期质量属性非常关键，因为它们直接影响着客户对软件系统的满意度。
  - 常见的运行期质量属性包括：软件系统的易用性、性能、可伸缩性、持续可用性、鲁棒性、安全性等。
  - 在超市系统的案例中，用户对高性能提出了具体要求（真正的性能需求应该量化），例如，他们不能容忍金额合计超过2秒的延时。

2.1 “4+1”视图模型

- 非功能需求又可以分为如下三类：
- **开发期质量属性：**
  - **开发期质量属性是开发人员最为关心的，要达到怎样的目标应根据项目的具体情况而定，而过度设计会花费额外的代价。**



2.2 软件体系结构的核心模型

- 综合软件体系结构的概念，体系结构的核心模型由5种元素组成：
  - 构件
  - 连接件
  - 配置（configuration）
  - 端口
  - 角色（role）
- 其中构件、连接件和配置是最基本的元素。

2.2 软件体系结构的核心模型

- （1）构件是具有某种功能上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存储。
  - 构件有两种：复合构件和原子构件
    - ✓ 复合构件由其它复合构件和原子构件，通过连接而成。
    - ✓ 原子构件是不可再分的构件，其内部是由实现该构件的一组类组成的。
  - 这种构件的划分，提供了体系结构的分层表示能力，有助于简化体系结构的设计。

2.2 软件体系结构的核心模型

- （2）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。
  - 复杂的连接件如：客户-服务器通信协议、数据库和应用之间的SQL连接等。复杂的连接件用于表示更为复杂的静态连接结构和动态协作结构。

- （3）配置表示了构件和连接件的拓扑逻辑及其相关的约束。

2.2 软件体系结构的核心模型

➢ 构件

- 构件作为一个被封装的实体，只能通过其接口与外部环境交互。
- 构件的接口由一组端口组成，每个端口表示了构件和外部环境的一个特定的交互点，它可以包含一组操作签名（特征标记）。
- 一个端口可以非常简单。
  - ✓ 如过程调用。
- 一个端口也可以表示更为复杂的接口界面（包含一些约束）。
  - ✓ 如必须以某种顺序调用的一组过程调用。
- 通过不同的端口类型，一个构件可以提供多种接口。

2.2 软件体系结构的核心模型

➢ 连接件

- 连接件作为建模软件体系结构的主要实体，同样也有接口。
- 连接件的接口由一组角色组成，连接件的每一个角色，定义了该连接件表示的交互的参与者。
- 二元连接件有两个角色，如：
  - ✓ RPC连接件的角色为caller和callee，
  - ✓ pipe连接件的角色是reading和writing，
  - ✓ 消息传递连接件的角色为sender和receiver。
- 有的连接件有多于两个的角色
  - ✓ 如事件广播连接件，有一个事件发布者角色，和任意多个事件接收者角色。

2.2 软件体系结构的核心模型

- 基于以上所述，可将软件体系结构的核心模型表示为图2-14所示。

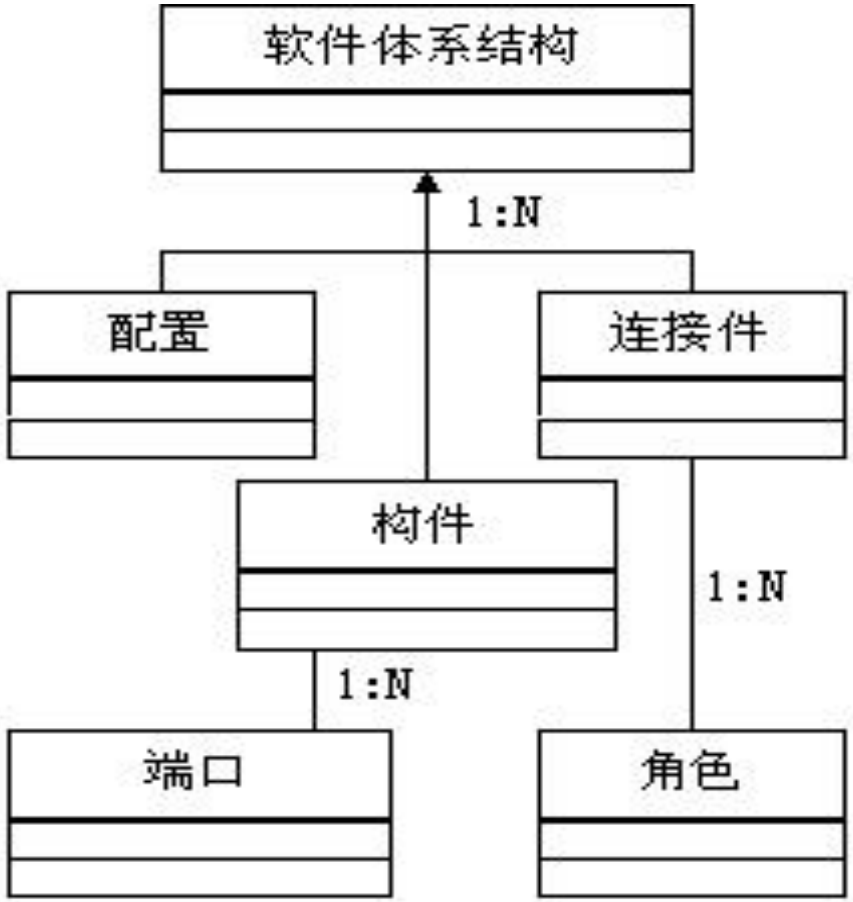


图2-14 软件体系结构的核心模型

2.3 软件体系结构的生命周期

- 对于软件项目的开发来说，一个清晰的软件体系结构是首要的。
- 传统的软件开发过程可以划分为从概念直到实现的若干个阶段，包括问题定义、需求分析、软件设计、软件实现及软件测试等。

- 软件体系结构的建立应位于需求分析之后，软件设计之前。

2.3.1 软件开发的主要阶段

➢ 1、需求分析阶段

- 需求是指用户对目标软件系统在功能、行为、性能、设计约束等方面的期望。
- 需求分析阶段的任务是根据需求，决定系统的功能。
- 分析师应对目标对象和环境作细致深入的调查，收集目标对象的基本信息，从中找出有用的信息，这是一个抽象思维、逻辑推理的过程，其结果是软件规格说明。

2.3.1 软件开发的主要阶段

1、需求分析阶段

- 获取需求包括：需求获取、生成类图、对类分组、把类打包和需求评审等过程。
- 需求获取的主要工作是：
  - ✓ 定义满足用户需要的业务层面上的服务和功能等
  - ✓ 定义软件质量属性
  - ✓ 定义系统须满足的一些非功能需求
- 获取了需求之后，就可以利用工具，制作类图。
- 然后对类进行分组，简化类图结构，使之更清晰。
- 分组之后，再要把类簇，进行打包。

2.3.1 软件开发的主要阶段

➢ 1、需求分析阶段

- 最后，组织一个由不同代表（例如，系统分析师、需求分析师、系统设计师、客户代表、测试人员等）组成的小组，对需求分析方法及其结果进行仔细的审查。
- 审查的主要内容包括：
  - ✓ 所获取的需求是否真实反映了用户的要求
  - ✓ 需求分析所用的技术是否正确和恰当
  - ✓ 需求分析文档是否充分、正确、规范、清晰

2.3.1 软件开发的主要阶段

➢ 2、建立软件体系结构阶段

- 在建立体系结构的初期，选择一个合适的体系结构风格是首要的。
- 在建立软件体系结构时，设计师要
  - ✓ 设计系统的总体（整体）结构
  - ✓ 确定恰当的构件
  - ✓ 确定构件间的相互作用以及它们的约束
  - ✓ 构造一个满足用户需求的系统结构
  - ✓ 设计各个构件间的接口
- 一旦设计了软件体系结构，设计小组必须邀请独立于系统开发的外部人员，对体系结构进行评审。

2.3.1 软件开发的主要阶段

- 3、详细设计阶段
  - 设计各模块内部的具体、详细的算法和数据结构。
  - 为编码实现，提供详细指南。

2.3.1 软件开发的主要阶段

➢ 4、实现阶段

- 对设计阶段确定的的算法及数据类型，用某种编码语言实现。
- 整个实现过程是以复审后的文档化的体系结构说明书为基础的，每个构件必须满足软件体系结构中说明的对其它构件的责任。

2.3.1 软件开发的主要阶段

➢ 4、实现阶段

- 工作基础：
  - ✓ 在体系结构说明书中，已经定义了系统中的构件与构件之间的关系。
  - ✓ 在体系结构层次上，构件接口及其约束反映了非常重要的设计要求和限定条件。
- 在实现时，可以从代码构件库中，查找符合接口约束的代码构件，必要时开发新的满足要求的构件。
- 然后，按照设计提供的结构，通过组装支持工具，把这些构件的实现体，组装起来，完成整个软件系统的连接与合成。
- 最后一步是测试，包括单个构件的功能性测试和被组装应用的整体功能和性能测试。

2.3.2 软件体系结构的生命周期

- 软件体系结构在系统开发的全过程中起着基础的作用，是设计的起点和依据，同时也是装配和维护的指南。
- 与软件本身一样，软件体系结构也有其生命周期，图2-15表示了体系结构的生命周期。

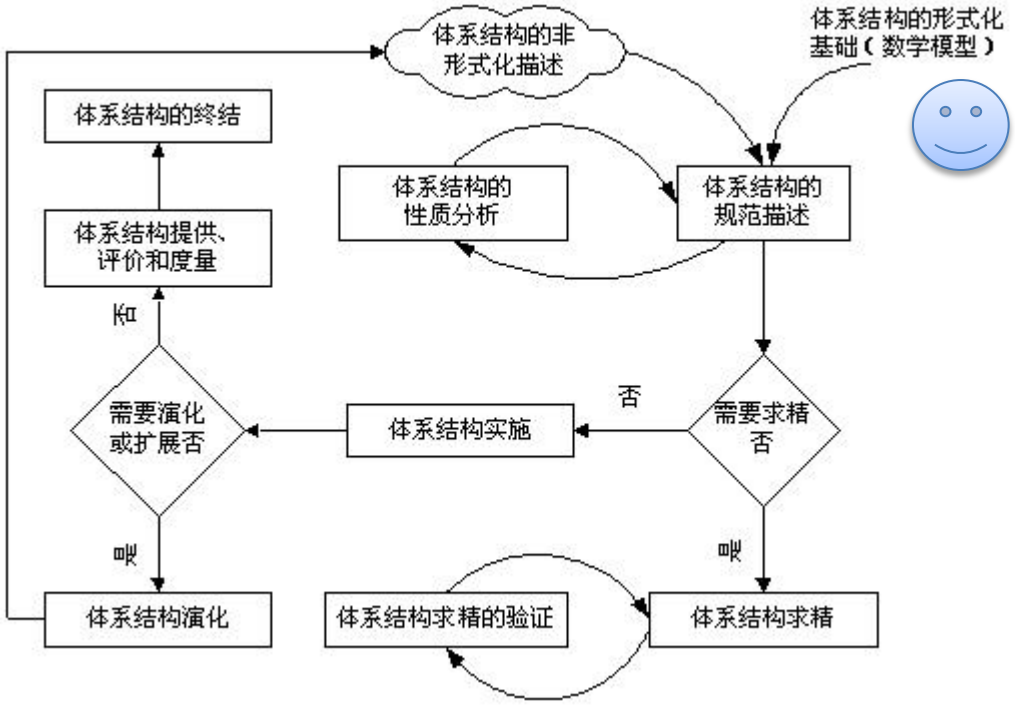


图2-15 软件体系结构的生命周期模型

2.3.2 软件体系结构的生命周期

- 1、软件体系结构的非形式化描述
  - 在软件体系结构的非形式化描述（software architecture informal description）阶段，通常采用自然语言，对软件体系结构进行描述。
  - 但是这阶段的工作，却是创造性和开拓性的。

- 软件体系结构的设计思想是非常重要的。
- 软件设计师常常用非形式化的自然语言，表示与设计相关的概念和原则。

2.3.2 软件体系结构的生命周期

➢ 2、软件体系结构的规范描述和分析

- 软件体系结构的规范描述和分析（software architecture specification and analysis）阶段，通过运用合适的形式化数学理论模型，对第1阶段的体系结构的非形式化描述，进行规范定义，从而得到软件体系结构的形式化规范描述。
- 软件体系结构的形式化规范描述，是精确的、无歧义的。在其上可以分析软件体系结构的一些性质，如无死锁性、安全性、活性等。
- 分析软件体系结构的性质，有利于选择合适的软件体系结构。即，它对软件体系结构的选择，可以发挥指导性作用，避免盲目选择。

2.3.2 软件体系结构的生命周期

➢ 3、软件体系结构的求精及其验证

- 大型系统的软件体系结构，总是通过从抽象到具体，逐步求精而达到的。
- 如果软件体系结构的抽象粒度过大，就需要对体系结构进行求精、细化，直至能够在系统设计中实施为止。
- 在软件体系结构的每一步求精过程中，需要对不同抽象层次的软件体系结构进行验证，以判断较具体的软件体系结构是否与较抽象的软件体系结构的语义一致，并能实现抽象的软件体系结构。

2.3.2 软件体系结构的生命周期

➢ 4、软件体系结构的实施

- 软件体系结构的实施（software architecture enactment）阶段，将求精后的软件体系结构实施于系统的设计中，并将软件体系结构的构件和连接件等有机地组织在一起，形成系统设计的框架，以便据此实施于软件设计和构造中。

2.3.2 软件体系结构的生命周期

➢ 5、软件体系结构的演化和扩展

- 当体系结构实施后，就进入软件体系结构的演化和扩展阶段。
- 常常是性能、容错、安全性、互操作性、自适应性等非功能性质，影响着软件体系结构的扩展和改动，这称为软件体系结构的演化。
- 软件体系结构演化涉及到对软件体系结构的理解，需要进行软件体系结构的逆向工程和再造工程。

2.3.2 软件体系结构的生命周期

➢ 6、软件体系结构的提供、评价和度量

- 对软件体系结构，进行定性的评价和定量的度量，以利于对软件体系结构的重用，并取得经验教训。

2.3.2 软件体系结构的生命周期

➢ 7、软件体系结构的终结

- 如果一个软件系统的软件体系结构进行多次演化和修改，软件体系结构已变得难以理解，更重要的是不能达到系统设计的要求，不能适应系统的发展。
- 这时，对该软件体系结构的再造工程既不必要、也不可行，这意味着该软件体系结构已经过时，应该摈弃，用全新的满足系统设计要求的软件体系结构取而代之。
- 这个阶段被称为软件体系结构的终结（software architecture termination）阶段。

2.4 软件体系结构抽象模型

- 基于抽象代数理论，建立软件体系结构抽象模型，描述构件、连接件和软件体系结构的定义，以及它们的属性和动态行为。

- 基于软件体系结构抽象模型，讨论不同类型软件系统结构的相互关系，给出软件体系结构范式及其建立方法。

2.4 软件体系结构抽象模型

- 抽象代数是把数和数上的运算，作进一步的抽象后，构造出抽象的代数系统，并进而研究其上代数运算的结构和性质。
- 我们利用抽象代数的数学方法，为待解决的实际问题，构造或定义一个代数系统。
- 用这个代数系统刻画问题，研究问题，构造问题解决的方案，验证问题、问题解决方案和问题求解结果的各种性质。

2.4 软件体系结构抽象模型

- 用抽象代数系统上代数运算，去描述人们在软件开发过程，对软件体系结构所进行的各种加工、变换和处理，诸如：对构件、连接件和配置，进行组合、替换、重配置、演化等操作。

2.4 软件体系结构抽象模型

- 赵会群，王国仁，高远，软件体系结构抽象模型，计算机学报，2002(7):730-736。

第 25 卷 第 7 期
2002 年 7 月

计 算 机 学 报
CHINESE J. COMPUTERS

Vol. 25 No. 7
July 2002

软件体系结构抽象模型

赵会群[1)] 王国仁[2)] 高 远[2)]

[1)](北方工业大学计算机科学系 北京 100041)

[2)](东北大学计算机科学与工程系 沈阳 110004)

摘 要 针对软件体系结构描述语言(Architecture Description Languages,ADLs)在动态、可进化和可扩充特征描述能力的不足,用抽象代数理论对组件、连接器和软件体系结构的属性和行为特征进行抽象,提出了组件连接"运算"这一新的观点,给出了在不同连接运算下的软件体系结构范式和事件依赖的软件体系结构范式定理. 该文提出的较完整的软件体系结构理论体系,为进一步研究软件体系结构和基于软件体系结构的相关研究奠定了理论基础.

关键词 组件,连接器,软件体系结构,抽象代数,软件可靠性

中图法分类号: TP311

The End