UML建模语言

UML建模语言

目录

返回

1 UML概述

UML（Unified Modeling Language）是软件界第一个统一的建模语言，该方法结合了Booch, OMT, 和OOSE方法的优点，统一了符号体系，并从其它的方法和工程实践中吸收了许多经过实际检验的概念和技术。

它是一种标准的表示, 已成为国际软件界广泛承认的标准。它是第三代面向对象的开发方法，是一种基于面向对象的可视化的通用(General)建模语言。为不同领域的用户提供了统一的交流标准 — UML图。

UML应用领域很广泛，可用于软件开发建模的各个阶段，商业建模（Business Modeling），也可用于其它类型的系统。

什么是模型？为什么要建模？

模型是一个系统的完整的抽象。人们对某个领域特定问题的求解及解决方案，对它们的理解和认识都蕴涵在模型中。

通常，开发一个计算机系统是为了解决某个领域特定问题，问题的求解过程，就是从领域问题到计算机系统的映射。

UML作为一种可视化的建模语言，提供了丰富的基于面向对象概念的模型元素及其图形表示元素。

1.1 UML的形成

九十年代中，面向对象方法已经成为软件分析和设计方法的主流。

1994年10月Jim Rumbaugh和Grady Booch共同合作把他们的OMT和Booch方法统一起来，到1995年成为“统一方法”（Unified Method）版本0.8。随后，Ivar Jacobson加入，并采用他的用例(User case)思想,到1996年，成为“统一建模语言”版本0.9。

1997年1月，UML版本1.0被提交给OMG组织，作为软件建模语言标准的候选。其后的半年多时间里，一些重要的软件开发商和系统集成商都成为“UML伙伴”，如IBM,Mircrosoft,HP等.1997年11月7日被正式采纳作为业界标准。

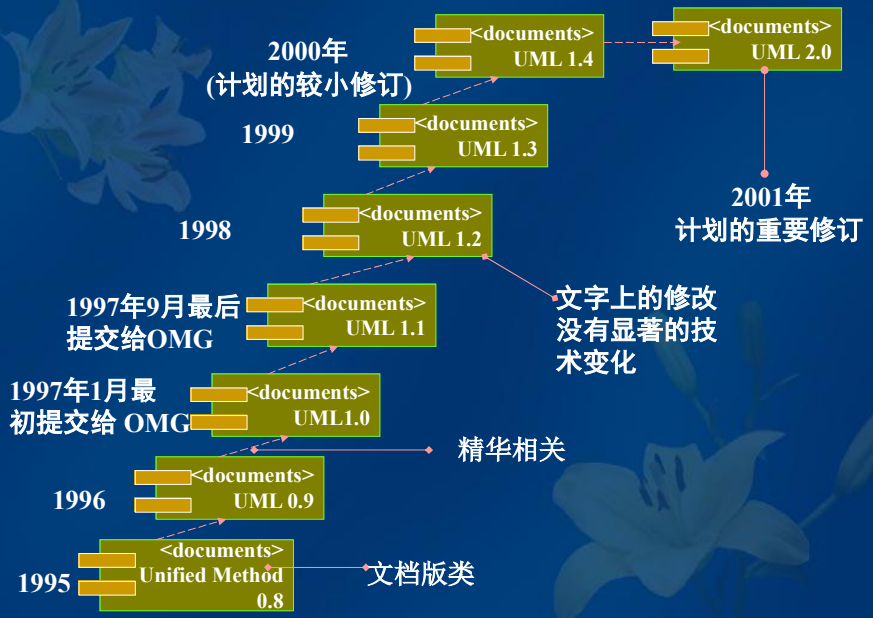


图1

1.2 UML的主要内容

UML的定义包括UML语义和UML表示法两个部分。

(1) UML语义　描述基于UML的精确元模型(meta-model)定义。元模型为UML的所有元素在语法和语义上提供了简单、一致、通用的定义性说明,使开发者能在语义上取得一致,消除了因人而异的表达方法所造成的影响。此外UML还支持对元模型的扩展定义。

UML支持各种类型的语义。如布尔、表达式、列表、阶、名字、坐标、这字符串和时间等，还允许用户自定义类型。

(2) UML表示法　定义UML符号的表示法,为开发者或开发工具使用这些图形符号和文本语法为系统建模提供了标准.。这些图形符号和文字所表达的是应用级的模型,在语义上它是UML元模型的实例。

UML的主要构成

UML是一种标准化的图形建模语言，它是面向对象分析与设计的一种标准表示。由:

- 视图(views),
- 图(Diagrams),
- 模型元素(Model elements)
- 通用机制(general mechanism)

等几个部分构成。

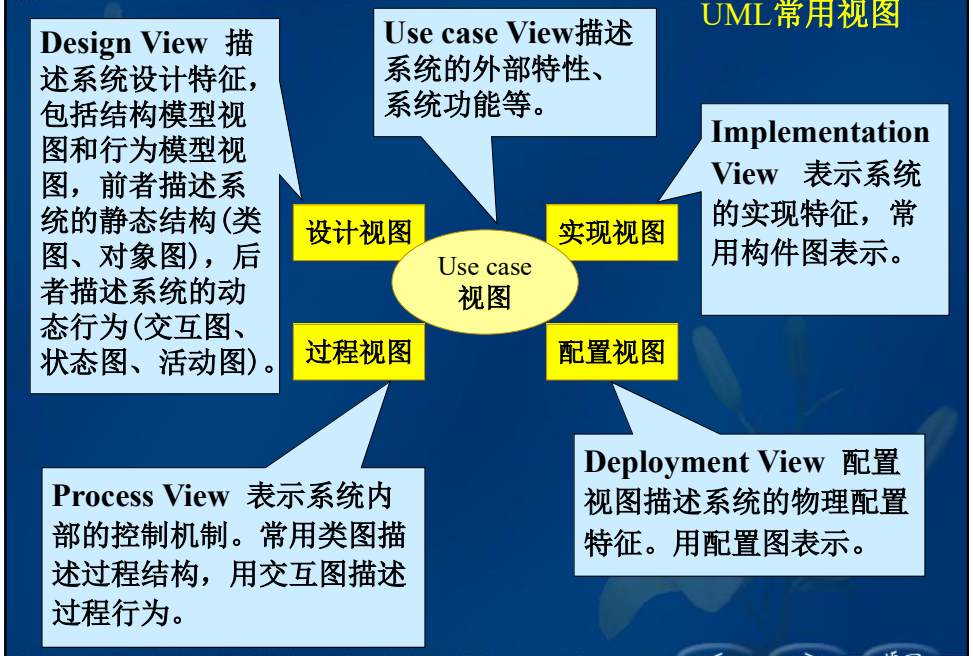
视图(views)

一个系统应从不同的角度进行描述，从一个角度观察到的系统称为一个视图（view）。

视图由多个图（Diagrams）构成，它不是一个图表（Graph），而是在某一个抽象层上，对系统的抽象表示。

如果要为系统建立一个完整的模型图，需定义一定数量的视图，每个视图表示系统的一个特殊的方面。另外，视图还把建模语言和系统开发时选择的方法或过程连接起来。

UML常用视图



图(Diagrams)

UML语言定义了五种类型，9种不同的图，把它们有机的结合起来就可以描述系统的所有视图。

- 用例图(Use case diagram) 从用户角度描述系统功能，并指出各功能的操作者。
- 静态图(Static diagram)，表示系统的静态结构。包括类图、对象图、包图。
- 行为图(Behavior diagram)，描述系统的动态模型和组成对象间的交互关系。包括状态图、活动图。
- 交互图(Interactive diagram)，描述对象间的交互关系。包括顺序图、合作图。
- 实现图(Implementation diagram) 用于描述系统的物理实现。包括构件图、部署图。

模型元素(Model elements)

代表面向对象中的类，对象，关系和消息等概念，是构成图的最基本的常用的元素。一个模型元素可以用在多个不同的图中，无论怎样使用，它总是具有相同的含义和相同的符号表示。

通用机制(general mechanism)

用于表示其他信息，比如注释，模型元素的语义等。另外，为了适应用户的需求，它还提供了扩展机制(Extensibility mechanisms)，包括构造型(Stereotype)、标记值(Tagged value)和约束(Constraint)。使用UML语言能够适应一个特殊的方法（或过程），或扩充至一个组织或用户。

UML在演变过程中还提出了一些新的概念。在UML标准中新加了模板(Stereotypes)、职责(Responsibilities)、扩展机制(Extensibility mechanisms)、线程(Threads)、过程(Processes)、分布式(Distribution)、并发(Concurrency)、模式(Patterns) 、合作(Collaborations)、活动图(Activity diagram)等新概念,并清晰地区分类型(Type)、类(Class)和实例(Instance)、细化(Refinement)、接口(Interfaces)和组件(Components)等概念。

1.3 UML的特点

(1) 统一标准

UML统一了Booch、OMT和OOSE等方法中的基本概念，已成为OMG的正式标准，提供了标准的面向对象的模型元素的定义和表示。

(2) 面向对象

UML还吸取了面向对象技术领域中其他流派的长处。UML符号表示考虑了各种方法的图形表示,删掉了大量易引起混乱的、多余的和极少使用的符号,也添加了一些新符号。

(3) 可视化、表示能力强

系统的逻辑模型或实现模型都能用UML模型清晰的表示，可用于复杂软件系统的建模。

(4) 独立于过程

UML是系统建模语言，独立于开发过程。

(5) 易掌握、易用

由于UML的概念明确，建模表示法简洁明了，图形结构清晰，易于掌握使用。

2 通用模型元素

模型元素是UML构造系统的各种元素，是UML构建模型的基本单位。模型元素代表面向对象中的类，对象，关系和消息等概念，是构成图的最基本的常用的概念。分为以下两类：

1、基元素

是已由UML定义的模型元素。如：类、结点、构件、注释、关联、依赖和泛化等。

2、构造型元素

在基元素的基础上构造的新的模型元素，是由基元素增加了新的定义而构成的，如扩展基元素的语义（不能扩展语法结构）,也允许用户自定义。构造型用括在双尖括号《》中的字符串表示。

目前UML提供了40多个预定义的构造型元素。如使用《Use》、扩展《 Extend 》。

2.1 模型元素

可以在图中使用的概念统称为模型元素。模型元素在图中用其相应的视图元素（符号）表示，图2给出了常用的元素符号：类、对象、结点、包和组件等。

图2

模型元素与模型元素之间的连接关系也是模型元素，常见的关系有关联（association）、泛化（generalization）、依赖(dependency)和聚合(aggregation)，其中聚合是关联的一种特殊形式。这些关系的图示符号如图3所示。

图3

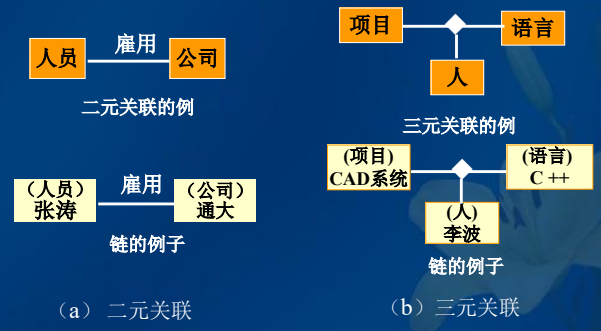
关联：连接（connect）模型元素及链接(link)实例。
依赖：表示一个元素以某种方式依赖于另一种元素。
泛化：表示一般与特殊的关系，即“一般”元素是“特殊”关系的泛化。
聚合：表示整体与部分的关系。
除了上述的模型元素外，模型元素还包括消息，动作和版类（stereotype）等。

2.2 关联和链

关联（association）是两个或多个类之间的一个关系。链（link）是关联的具体体现。

关联的表示

如图 4 (a) (b) 所示，关联有二元关联(binary)、三元关联(ternary)、多元关联(higher order)。



（a） 二元关联　　（b）三元关联

图 4

关联的重数

重数(multiplicity)表示多少个对象与对方对象相连接(图5)，常用的重数符号有：

“0..1”　　表示零或1
“0..*”或“*”　表示零或多个
“1..*”　　表示1或多个
“1, 3, 7”　表示1或3或7（枚举型）

重数的默认值为1。

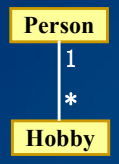
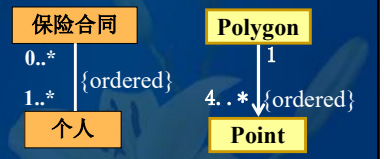


图5 带有多重性关联

有序关联与导航（导引）

在关联的多端标注{ordered}指明这些对象是有序的(图6 ）。

关联可以用箭头，表示该关联使用的方向(单向或双向)，称为导引或导航(navigation)。



(a)指定链接之间有明确的顺序　　(b)单向关联

图6

2.4 约束

UML中提供了一种简便、统一和一致的约束（constraint），是各种模型元素的一种语义条件或限制。一条约束只能应用于同一类的元素。

约束的表示

如果约束应用于一种具有相应视图元素的模型元素，它可以出现在它所约束元素视图元素的旁边。

通常一个约束由一对花括号括起来（{constraint}），花括号中为约束内容（图8）。

如果一条约束涉及同一种类的多个元素，则要用虚线把所有受约束的元素框起来，并把该约束显示在旁边（如或约束）。

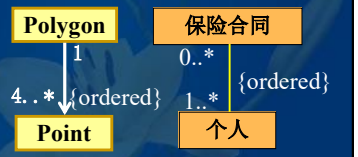


图8

约束可分为：对泛化的约束、关联的约束

- 对泛化的约束

应用于泛化的约束，显示在大括号里，若有多个约束，用逗号隔开。如果没有共享，则用一条虚线通过所有继承线，并在虚线的旁边显示约束， 如图9所示：

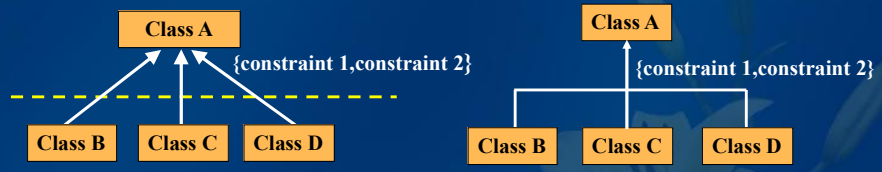


图9 对泛化的约束的两种表示方法

- 对泛化有以下常用的约束：

1、**complete:** 说明泛化中所有子元素都已在模型中说明，不允许再增加其它子元素。

2、**disjoint:** 父类对象不能有多于一个型的子对象。

3、**incomplete:** 说明不是泛化中所有子元素都已说明，允许再增加其它子元素。

4、**overlapping:** 给定父类对象可有多于一个型的子对象，表示重载。

返回

- 关联的约束

对关联有以下常用的约束：

1、**implicit:**该关联只是概念性的，在对模型进行精化时不再用。
2、**ordered:**具有多重性的关联一端的对象是有序的。
3、**changeable:**关联对象之间的链(Link)是可变的（添加、修改、删除）。
4、**addonly:**可在任意时刻增加新的链接。
5、**frozen:**冻结已创建的对象，不能再添加、删除和修改它的链接。
6、**xor:**“或约束”，某时刻只有一个当前的关联实例。

图10 对象类的xor关联

- 对消息，链接角色和对象的约束
- 自定义约束

2.6 依赖

- 依赖关系描述的是两个模型元素（类，组合，用例等）之间的语义上的连接关系，其中一个模型元素是独立的，另一个模型元素是非独立的（或依赖的）。如图11表示类A依赖于类B的一个友元依赖关系。

图11

依赖的形式可能是多样的，针对不同的依赖的形式，依赖关系有不同的变体(varieties)：

依赖的形式可能是多样的，针对不同的依赖的形式，依赖关系有不同的变体(varieties)：

〈1〉抽象(abstraction)：从一个对象中提取一些特性，并用类方法表示。

〈2〉绑定(binding)：为模板参数指定值，以定义一个新的模板元素。

〈3〉组合(combination)：对不同类或包进行性质相似融合。

〈4〉许可(permission)：允许另一个对象对本对象的访问。

〈5〉使用(usage)：声明使用一个模型元素需要用到已存在的另一个模型元素，这样才能正确实现使用者的功能(包括调用、实例化、参数、发送)。

〈6〉跟踪(trace)：声明不同模型中元素的之间的存在一些连接。

〈7〉访问或连接(access)：允许一个包访问另一个包的内容。

〈8〉调用(call)：声明一个类调用其他类的操作的方法。

〈9〉导出(derive)：声明一个实例可从另一个实例导出。

〈10〉友元(friend)：允许一个元素访问另一个元素，不管被访问的元素是否具有可见性。

〈11〉引入(import)：允许一个包访问另一个包的内容，并为被访问组成部分增加别名。

〈12〉实例(instantiation)：关于一个类的方法创建了另一个类的实例声明。

〈13〉参数(parameter)：一个操作和它参数之间的关系。

〈14〉实现(realize)：说明和其实之间的关系。

〈15〉精化(refine)：声明具有两个不同语义层次上的元素之间的映射。

〈16〉发送(send)：信号发送者和信号接收者之间的关系。

2.7 细化

有两个元素A和B，若B元素是A元素的详细描述，则称B, A元素之间的关系为B元素细化A元素。

细化与类的抽象层次有密切的关系，在构造模型时要经过逐步细化, 逐步求精的过程。

如图12所示，类B是类A细化的结果。

图12

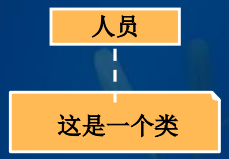


图13

5.2.8 注释

注释用于对UML语言的元素或实体进行说明，解释和描述。通常用自然语言进行注释。

3 用例建模

1992年由Jacobson提出了Use case的概念及可视化的表示方法—Use case图，受到了IT界的欢迎，被广泛应用到了面向对象的系统分析中。用例驱动的系统分析与设计方法已成为面向对象的系统分析与设计方法的主流。

用例模型由Jacobson在开发AXE系统中首先使用，并加入由他所倡导的OOSE和Objectory方法中。用例方法引起了面向对象领域的极大关注。自1994年Ivar Jacobson的著作出版后，面向对象领域已广泛接纳了用例这一概念，并认为它是第二代面向对象技术的标志。

3.1 用例建模概述

用例建模技术，用于描述系统的功能需求。在宏观上给出模型的总体轮廓。通过对典型用例的分析，使开发者能够有效地了解用户的需求。

图14

3.2 用例模型(Use case model)

用例模型描述的是外部执行者(Actor)所理解的系统功能。它描述了待开发系统的功能需求。

它驱动了需求分析之后各阶段的开发工作，不仅在开发过程中保证了系统所有功能的实现，而且被用于验证和检测所开发的系统，从而影响到开发工作的各个阶段和 UML 的各个模型。

用例模型由若干个用例图构成，用例图中主要描述执行者和用例之间的关系。在UML中，构成用例图的主要元素是用例和执行者及其它们之间的联系。

创建用例模型的工作包括：

定义系统、确定执行者和用例、描述用例、定义用例间的关系、确认模型。

一、执行者(Actor)

执行者是指用户在系统中所扮演的角色。执行者在用例图中是用类似人的图形来表示，但执行者可以是人，也可以是一个外界系统。

注意：用例总是由执行者启动的。

如何确定执行者：

1、谁使用系统的主要功能(主执行者)？
2、谁需要从系统获得对日常工作的支持和服务？
3、需要谁维护管理系统的日常运行（副执行者）？
4、系统需要控制哪些硬件设备？
5、系统需要与其它哪些系统交互？
6、谁需要使用系统产生的结果（值）？

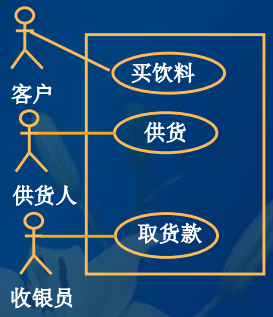


图15自动售货系统

回例1

二、用例(use case)

从本质上讲，一个用例是用户与计算机之间的一次典型交互作用。在UML中，用例被定义成系统执行的一系列动作（功能）。

用例有以下特点：

- 用例捕获某些用户可见的需求，实现一个具体的用户目标。
- 用例由执行者激活，并将结果值反馈给执行者。
- 用例必须具有功能上的完整描述。

如何确定用例：

1、与系统实现有关的主要问题是什么？
2、系统需要哪些输入/输出？这些输入/输出从何而来？到哪里去？
3、执行者需要系统提供哪些功能？
4、执行者是否需要对系统中的信息进行读、创建、修改、删除或存储？

回例1

3.3 用例图

用例图描述了系统的功能需求，它是从执行者的角度来理解系统，由“执行者”、“用例”和“用例之间的关系”3类模型元素构成。

图中还有另外两种类型的连接,即《使用》和《扩展》关系，是两种不同形式的泛化关系。

《Use》表示一个用例使用另一个用例。

《Extend》通过向被扩展的用例添加动作来扩展用例。

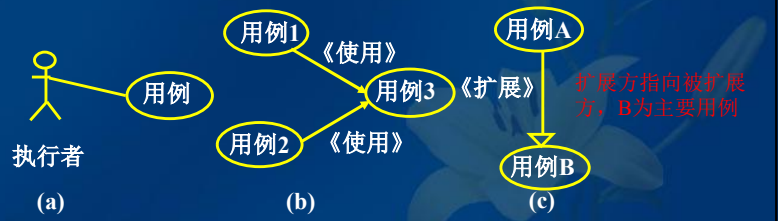


图16 用例图的元素

用例图实例

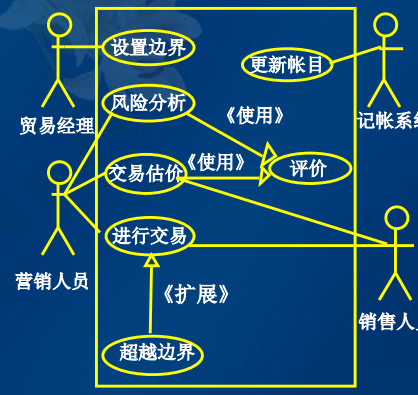


图17 金融贸易系统

图18 应用生命周期用例图

例1 建立项目与资源管理系统的Use case图

系统的主要功能是：项目管理，资源管理和系统管理。项目管理包括项目的增加、删除、更新。资源管理包括对资源和技能的添加、删除和更新。系统管理包括系统的启动和关闭，数据的存储和备份等功能。

1、分析确定系统的执行者(角色)

项目管理员、资源管理员、系统管理员、备份数据系统。

2、确定用例

项目管理，资源管理和系统管理。

3、对用例进行分解，画出下层的Use case图

对上层的用例进行分解,并将执行者分配到各层次的Use case图中。

还应画出相应的执行者描述模板及用例描述模板。

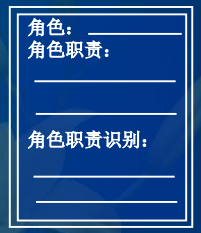


图19角色描述模板

例1 项目与资源管理系统（PRMS）

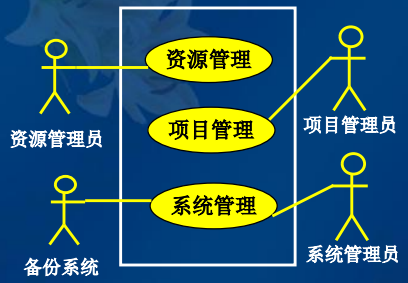
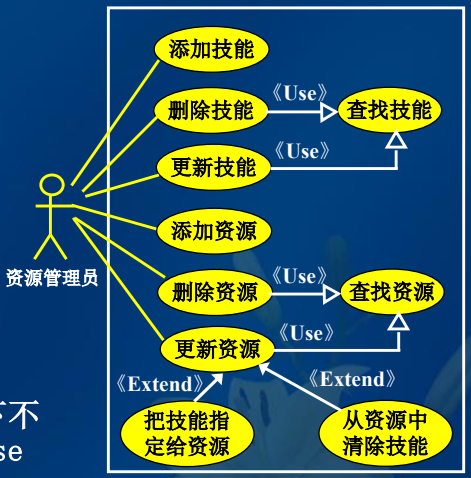


图20 PRMS高层Use Case图

Use Case图可以自顶而下不断精化，抽象出不同层次的Use Case图。



21 资源管理Use Case图

例1 项目与资源管理系统（PRMS）

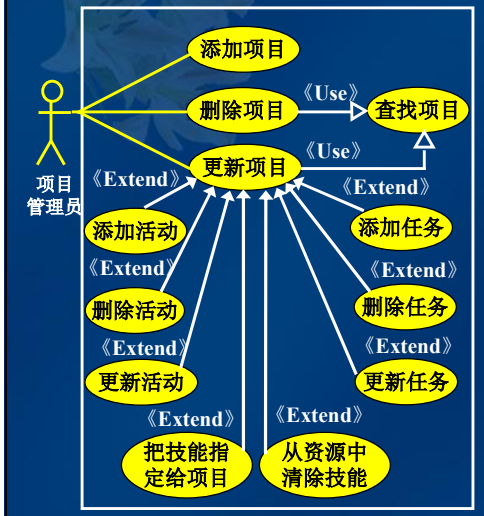


图22 项目管理Use Case图

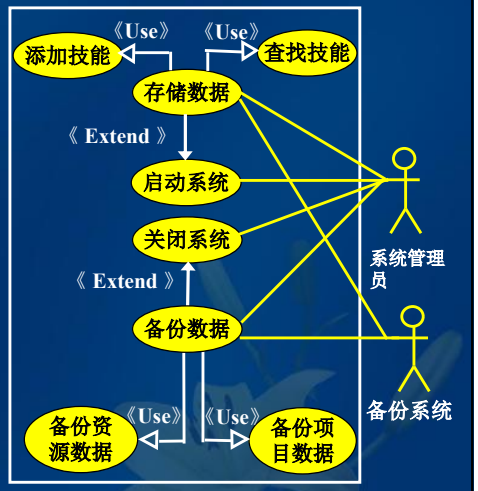


图23 系统管理Use Case图

案 例

现有一医院病房监护系统，病症监视器安置在每个病房，将病人的病症信号实时传送到中央监视系统进行分析处理。在中心值班室里，值班护士使用中央监视系统对病员的情况进行监控，根据医生的要求随时打印病人的病情报告，定期更新病历，当病症出现异常时，系统会立即自动报警,并实时打印病人的病情报告，立及更新病历。

要求根据现场情景，对医院病房监护系统进行需求分析， 建立系统的Use case model。

情景教学

例2 医院病房监护系统

监视病情

产生病情报告

更新病历

经过初步的需求分析，得到系统功能要求：

1、监视病员的病症（血压、体温、脉搏等）

2、定时更新病历

3、病员出现异常情况时报警。

4、随机地产生某一病员的病情报告。

二、简单的需求分析说明

系统名称：医院病房监护系统

根据分析系统主要实现以下功能：

1、病症监视器可以将采集到的病症信号（组合），格式化后实时的传送到中央监护系统。

2、中央监护系统将病人的病症信号开解后与标准的病症信号库里的病症信号的正常值进行比较，当病症出现异常时系统自动报警。

3、当病症信号异常时，系统自动更新病历并打印病情报告。

4、值班护士可以查看病情报告并进行打印。

5、医生可以查看病情报告，要求打印病情报告，也可以查看或要求打印病历。

6、系统定期自动更新病历。

三、用UML的静态建模机制定义并描述系统的静态结构

（一）建立系统的用例图

1、通过以下六个问题识别角色

(1)谁使用系统的主要功能？

(2)谁需要系统的支持以完成日常工作任务？

(3)谁负责维护，管理并保持系统正常运行？

(4)系统需要应付（或处理）哪些硬设备？

(5)系统需要和哪些外部系统交互？

(6)谁（或什么）对系统运行产生的结果（值）感兴趣？

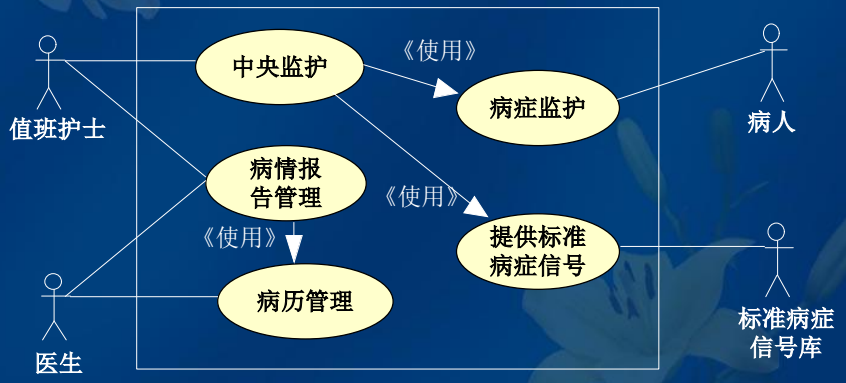
通过回答这六个问题以后，再进一步分析可以识别出本系统的四个角色：值班护士，医生，病人，标准病症信号库。

角色描述模板

角色：病　人 角色职责： 提供病症信号 角色职责识别： 负责生成、实时提供各种病症信号。	角色：医　生 角色职责： 对病人负责，负责处理病情的变化 角色职责识别： (1)需要系统支持以完成其日常工作 (2)对系统运行结果感兴趣	角色：值班护士 角色职责： 负责监视病人的病情变化 角色职责识别： (1)使用系统主要功能 (2)对系统运行结果感兴趣	角色:标准病症信号库 角色职责： 负责向系统提供病症信号的正常值 角色职责识别： (1)负责保持系统正常运行 (2)与系统交互

通过分析可以初步识别出系统的用例为：中央监护，病症监护，提供标准病症信号，病历管理，病情报告管理。顶层用例图为：

通过分析可以初步识别出系统的用例为：中央监护，病症监护，提供标准病症信号，病历管理，病情报告管理。顶层用例图为：



将用例细化，可以得到分解的用例：

1、中央监护

分解为：a 分解信号 将从病症监护器传送来的组合病症信号分解为系统可以处理的信号。

b 比较信号 将病人的病症信号与标准信号比较 。

c 报警　如果病症信号发生异常（即高于峰值），发出报警信号。

d 数据格式化 将处理后的数据格式化以便写入病历库 。

2、病症监护

分解为：e 信号采集 采集病人的病症信号。

f 模数转化 将采集来的模拟信号转化为数字信号。

g 信号数据组合 将采集到的脉搏，血压等信号数据组合为一组信号数据。

h 采样频率改变 根据病人的情况改变监视器采样频率。

3、提供标准病症信号 i（此用例不分解）

4、病历管理

分解为：j 生成病历

k 查看病历

l 更新病历

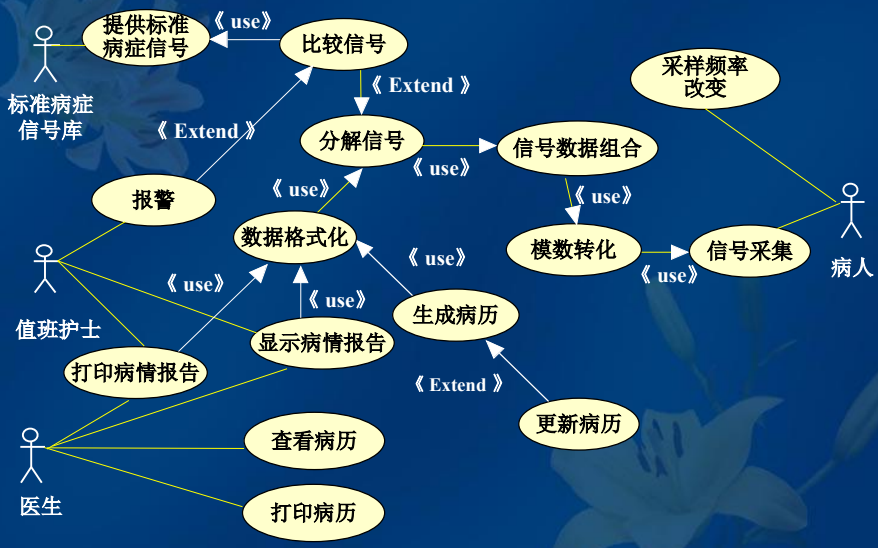
m 打印病历

5、病情报告

分解为：n 显示病情报告 在显示器上显示病情

o 打印病情报告 在打印机打印病情报告

给出细化的用例图



4 静态建模

任何建模语言都以静态建模机制为基础，标准建模语言UML也不例外。所谓静态建模是指对象之间通过属性互相联系，而这些关系不随时间而转移。

类和对象的建模，是UML建模的基础。我们认为，熟练掌握基本概念、区分不同抽象层次以及在实践中灵活运用，是三条最值得注意的建模基本原则。

UML的静态建模机制包括：

用例图(Use case diagram)

- 类图(Class diagram)
- 对象图(Object diagram)
- 包图(Package diagram)

构件图(Component diagram)

配置图(Deployment diagram)

4.1 对象类与对象

面向对象的开发方法的基本任务是建立对象模型，是软件系统开发的基础。UML中的对象类图(Class Diagram)与对象图(Object Diagram)表达了对象模型的静态结构，能够有效地建立专业领域的计算机系统对象模型。

一、类图与对象图

对象类简称类，是面向对象模型的最基本的模型元素，用类图来描述。类图(Class diagram)由系统中使用的类以及它们之间的关系组成，是描述系统的一种图式，分为长式和短式。类及类型名均用英文大写字母开头，属性及操作名为小写字母开头。常见类型有：Char,Boolean,Double,Float, Integer, Object,Short,String等。类图是构建其它图的基础。

对象是对象类的实例(instance), 用对象图来描述。对象图亦分长式和短式。

图6.24 类图与对象图

图25 对象图

(1)属性(attribute)

属性用来描述类的特征，表示需要处理的数据。

属性定义：

visibility attribute-name : type = initial-value {property-string}

可见性 属性名：类型=缺省值{约束特性}

其中：可见性(visibility)表示该属性对类外的元素是否可见。分为：

- public（+） 公有的，即模型中的任何类都可以访问该属性。
- private（-） 私有的，表示不能被别的类访问。
- protected（#） 受保护的，表示该属性只能被该类及其子类访问。

如果可见性未申明，表示其可见性不确定。

(2) 操作

对数据的具体处理方法的描述则放在操作部分，操作说明了该类能做些什么工作。操作通常称为函数，它是类的一个组成部分，只能作用于该类的对象上。

操作定义：

visibility operating-name(parameter-list): return-type {property-string}

可见性 操作名（参数表）：返回类型{约束特性}

其中：可见性同上。

参数表：参数名：类型，…

Parameter-name :type =default-value

返回类型：操作返回的结果类型。

二、类的识别

是面向对象方法的一个难点，但又是建模的关键。常用的方法有：

1、名词识别法

2、系统实体识别法

3、从用例中识别类

4、利用分解与抽象技术

关键是要定义类的“属性”及“操作”。

4.2 UML中类之间的关系

UML中类的关系有关联(association)、聚集(aggregation)、泛化(generalization)、依赖(depending)和细化(refinement)。

一、关联

关联是类之间的连结,分为:

1、常规关联(图6.26)

2、多元关联

3、有序关联

4、受限关联

5、或关联(图6.27)

6、关联类(图6.28)

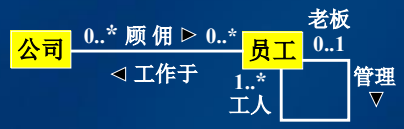


图26 顾佣关联

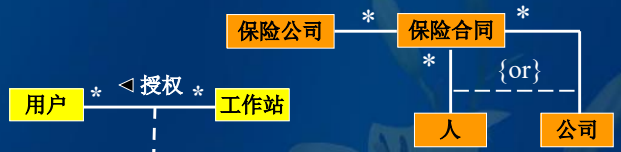


图27 或关联

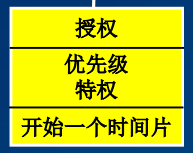


图28 关联类

7、其它关联

递归关联(Recursive association)

即一个类到自身的关联。

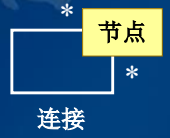


图29 递归关联

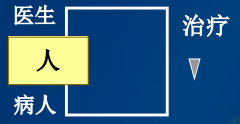


图30 带有职责的递归关联

二、聚集(aggregation)

聚集是一种特殊的关联，它指出类间的“整体-部分”关系。又分为:

1、共享聚集(shared aggregation)

其“部分”对象可以是任意“整体”对象的一部分。当“整体”端的重数不是1时，称聚集是共享的。

图31 共享聚集

2、组合聚集(composition aggregation)

其“整体”(重数为0、1)拥有它的“部分”。部分仅属于同一对象，整体与部分同时存在。

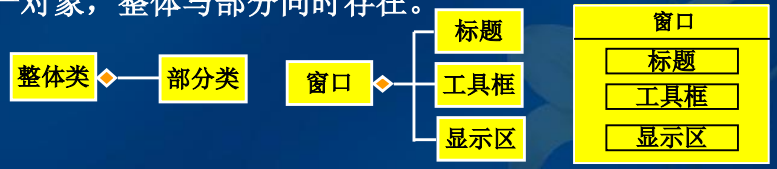


图32 组合聚集

三、泛化

泛化指出类之间的“一般与特殊关系”，即继承关系。父类与子类之间构成类的分层结构。

- 抽象类 指没有实例的类，定义一些抽象的操作，即不提供实现方法的操作，只提供操作的特征。并附以{abstract}。
- 交叠泛化 在继承树中，若存在某种具有公共父类的多重继承，称为是交叠(overlapping)的。否则是不交的(disjoint)。
- 完全泛化 一般类特化出它所有的子类，称为完全泛化，记为{complete}。
- 不完全泛化 即未特化出它所有的子类，称为是不完全泛化的，表示为{incomplete}.

有关泛化的约束

三、泛化

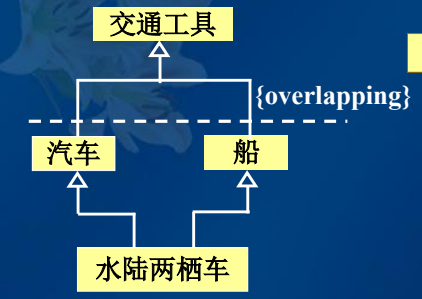


图6.33 重叠泛化

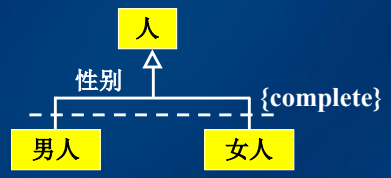


图34 完全泛化

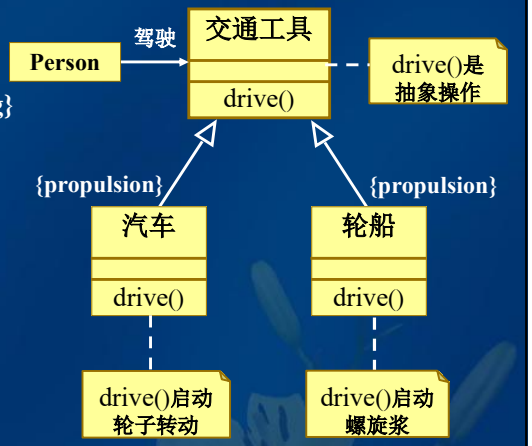


图35 泛化中的多态性
及带识别名称的泛化

继承性的实例

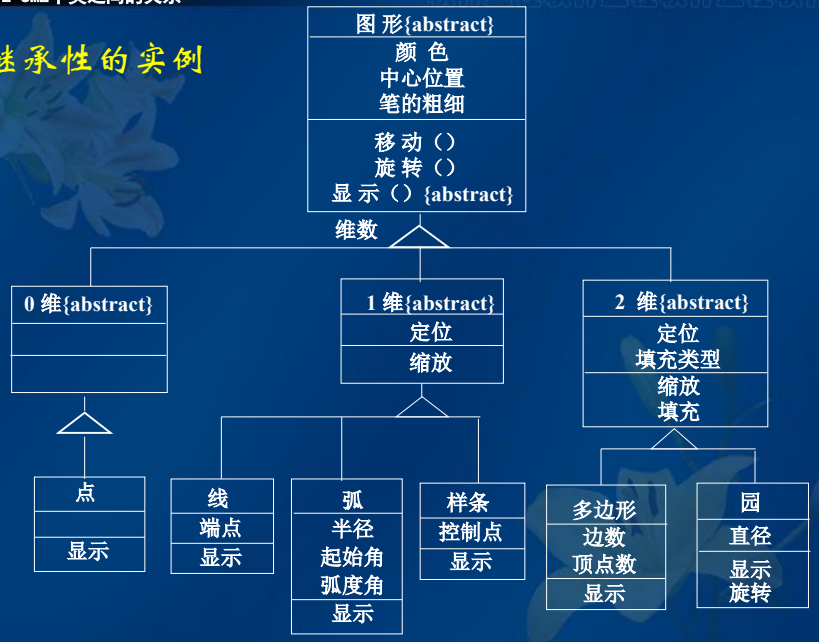


图36 泛化关系

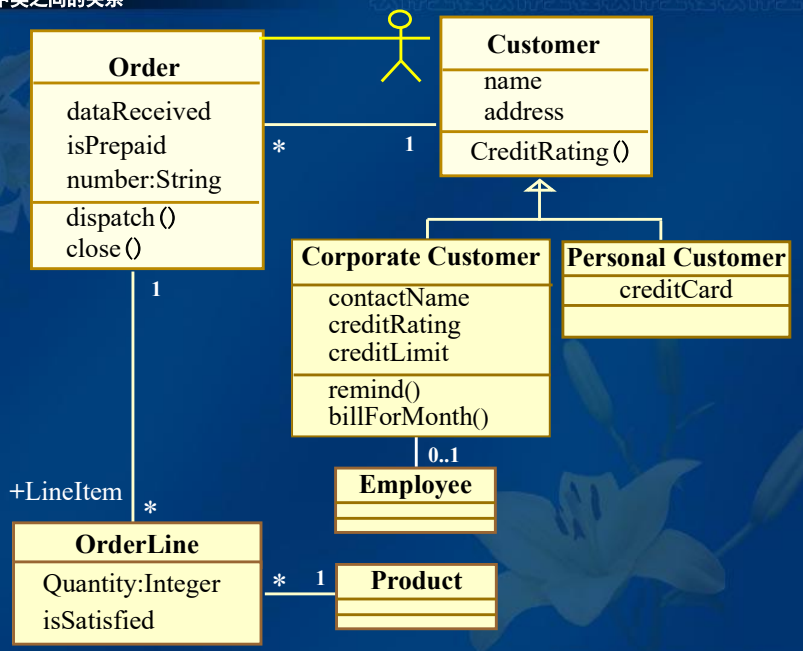


图37 泛化关系

五、类图的抽象层次和细化(Refinement)关系

需要注意的是，虽然在软件开发的不同阶段都使用类图，但这些类图表示了不同层次的抽象。

在需求分析阶段，类图是研究领域的概念；在设计阶段，类图描述类与类之间的接口；

而在实现阶段，类图描述软件系统中类的实现。

按照Steve Cook和John Dianiels的观点，类图分为三个层次：概念层、说明层、实现层。

需要说明的是，这个观点同样也适合于其他任何模型，只是在类图中显得更为突出。描述了类图的抽象层次和细化(Refinement)关系。

概念层

概念层(Conceptual)类图描述应用领域中的概念。实现它们的类可以从这些概念中得出，但两者并没有直接的映射关系。事实上，一个概念模型应独立于实现它的软件和程序设计语言。

说明层

说明层(Specification)类图描述软件的接口部分，而不是软件的实现部分。面向对象开发方法非常重视区别接口与实现之间的差异，但在实际应用中却常常忽略这一差异。这主要是因为OO语言中类的概念将接口与实现合在了一起。大多数方法由于受到语言的影响，也仿效了这一做法。现在这种情况正在发生变化。可以用一个类型（Type ）描述一个接口，这个接口可能因为实现环境、运行特性或者用户的不同而具有多种实现。

实现层

只有在实现层(Implementation)才真正有类的概念，并且揭示软件的实现部分。这可能是大多数人最常用的类图，但在很多时候，说明层的类图更易于开发者之间的相互理解和交流。

理解以上层次对于画类图和读懂类图都是至关重要的。但是由于各层次之间没有一个清晰的界限，所以大多数建模者在画图时没能对其加以区分。画图时，要从一个清晰的层次观念出发；而读图时，则要弄清它是根据哪种层次观念来绘制的。要正确地理解类图，首先应正确地理解上述三种层次。虽然将类图分成三个层次的观点并不是UML的组成部分，但是它们对于建模或者评价模型非常有用。尽管迄今为止人们似乎更强调实现层类图，但这三个层次都可应用于UML，而且实际上另外两个层次的类图更有用。

（二）识别系统的类

通过名词识别法和系统实体识别法等方法可以识别出系统的十二个类,以下用类图这种简单明了的方法分别表示出类的名称，属性\操作。见下图：

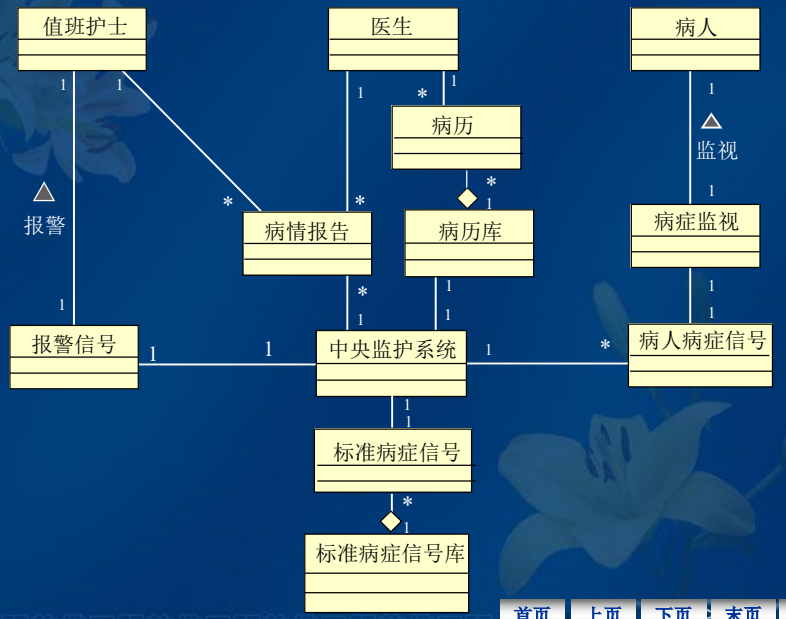
值班护士　医生　病人　病症监视　中央监护系统　报警信号　标准病症信号库　病历库　病人病症信号　病情报告　病历　标准病症信号

值班护士	医生	病人	病症监视
用户名 密码	用户名 密码	姓名 性别 年龄 病症	采集频率 病症信号
查看病情报告（） 打印病情报告（）	查看病情报告（） 要求打印病情报告（） 查看病历（） 要求打印病历（）	提供病症信号（）	格式化信号数据（） 采集信号（） 信号组合（）

中央监护系统	报警信号	标准病症信号库	病历库
输入 输出	声音 灯光 文字	类型 大小 容量	类型 大小 容量
分解信号（） 比较信号（） 报警（） 数据格式化（）	报警（） 数模转化（）	提供标准信号（）	生成病历（） 更新病历（） 查看病历（） 打印病历（）

病人病症信号	病情报告	病历	标准病症信号
脉搏 血压 体温	标题 格式	格式 病人基本情况 打印时间	脉搏 血压 体温
生成病症信号（）	生成病情报告（） 查看病情报告（） 打印病情报告（）	生成病历（） 查看病历（） 打印病历（）	生成标准信号（）

在类图中标明类之间的关系：

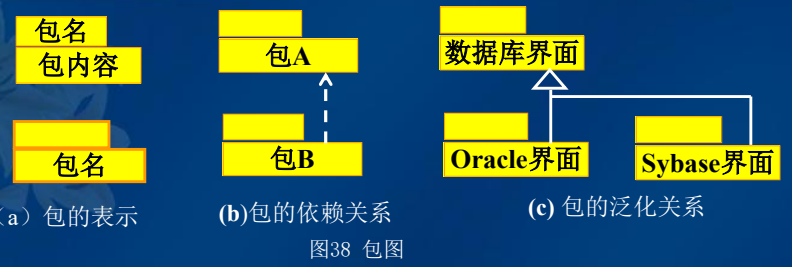


首页 上页 下页 末页 退出

4.3 包图

一个最古老的软件方法问题是:怎样将大系统拆分成小系统。UML中解决该问题的思路之一是将许多类集合成一个更高层次的单位,形成一个高内聚、低耦合的类的集合。UML中这种分组机制叫包(Package)。引入包是为了降低系统的复杂性。

包是一种组合机制，把各种各样的模型元素通过内在的语义连在一起成为一个整体就叫包，构成包的模型元素称为包的内容，包通常用于对模型的组织管理，因此有时又将包称为子系统（subsystem）。包拥有自己的模型元素，包与包之间不能共用一个相同的模型元素，包的实例没有任何语义（含义）。仅在模型执行期间包才有意义。



（a）包的表示　(b)包的依赖关系　(c) 包的泛化关系

图38 包图

包的内容：可以是类的列表,也可以是另一个包图,还可以是一个类图。

包之间的关系有依赖和泛化（继承）。

依赖关系：两个包中的任意两个类存在依赖关系，则包之间存在依赖关系。表示为：- - - - - →

泛化关系：使用继承中通用和特例的概念来说明通用包和专用包之间的关系。例如,专用包必须符合通用包的界面,与类继承关系类似。表示为：——▷

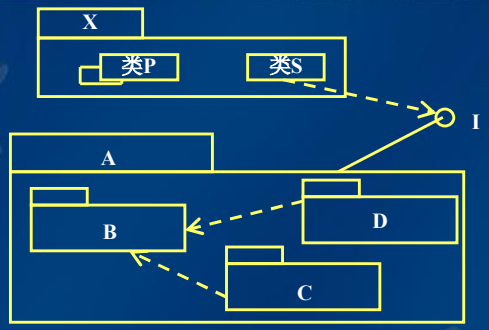


图39 包图举例

和类一样包也有可见性，利用可见性控制外部包对包的内容的存取方式，UML中定义了四种可见性：私有(private)，公有(public)，保护(protected)和实现(implementation)。缺省值为公有。

包也可以有接口，接口与包之间用实线相连，接口通常由包的一个或多个类实现。

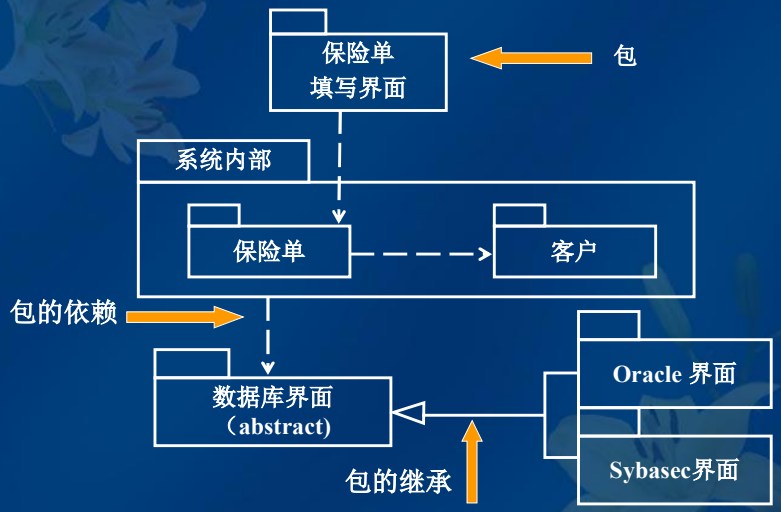


图40 包图

包图举例

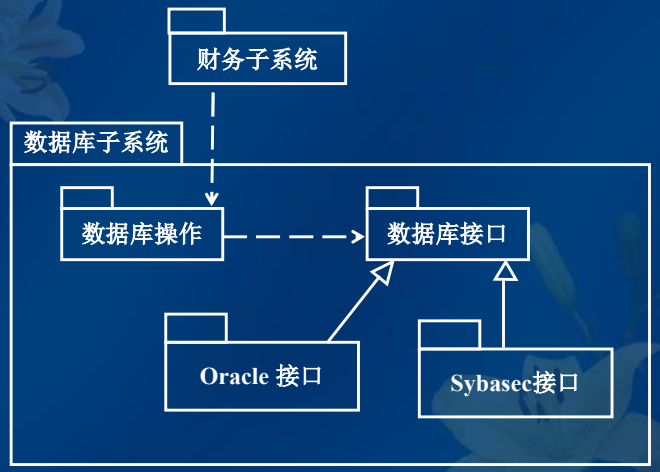
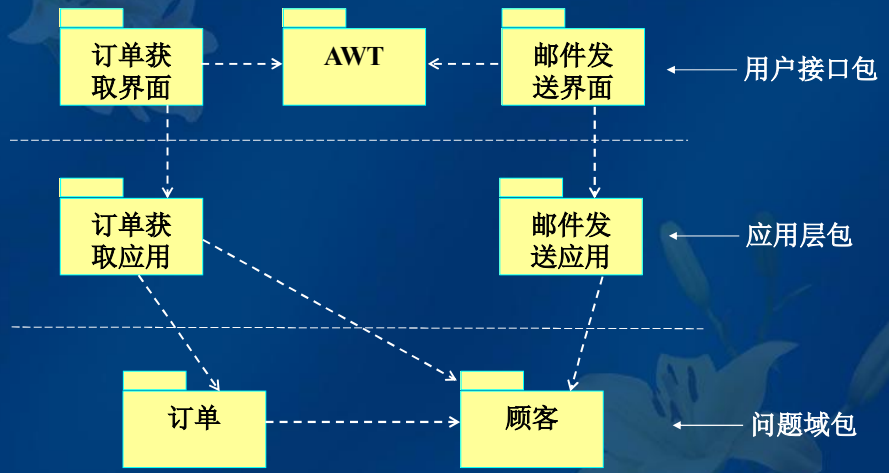
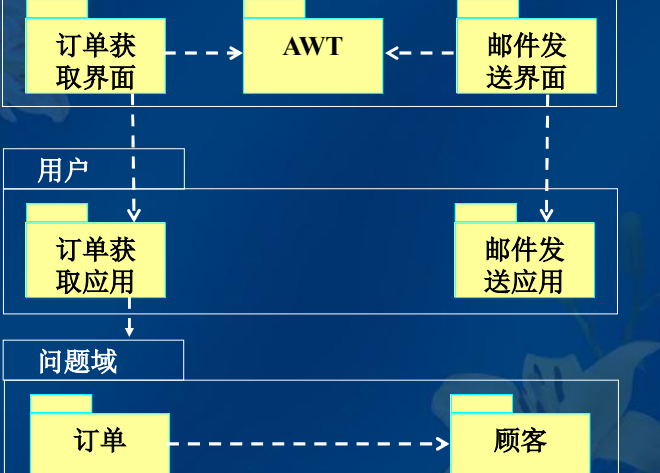


图41 包图举例

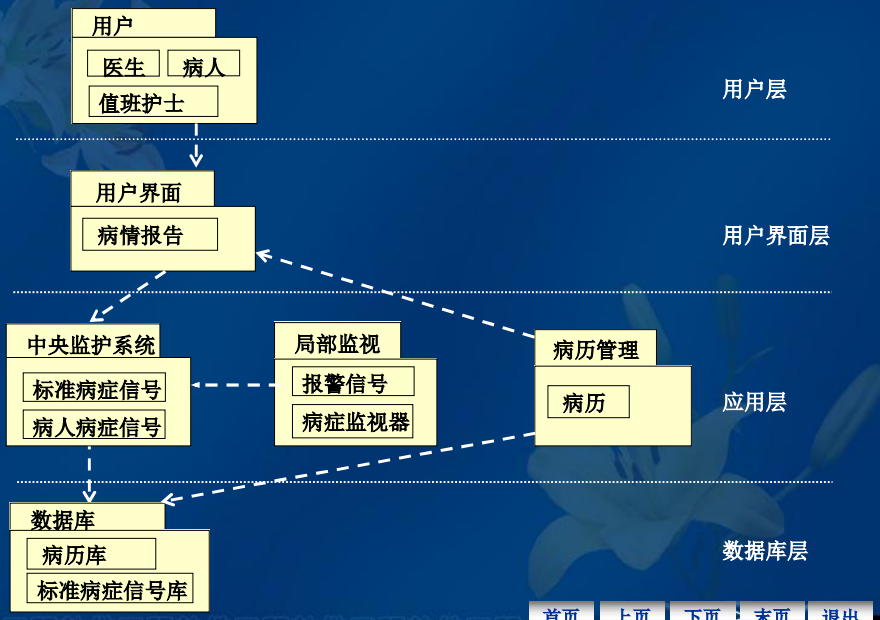
包图举例



包图另一种划分方式



（三）用包图描述医院监护系统的体系结构



5 动态建模

- 动态模型主要描述系统的动态行为和控制结构。动态行为包括系统中对象生存期内可能的状态以及事件发生时状态的转移，对象之间动态合作关系，显示对象之间的交互过程以及交互顺序，同时描述了为满足用例要求所进行的活动以及活动间的约束关系。
- 在动态模型中，对象间的交互是通过对象间消息的传递来完成的。对象通过相互间的通信（消息传递）进行合作，并在其生命周期中根据通信的结果不断改变自身的状态。
- UML消息的图形表示是用带有箭头的线段。

图6.42

动态模型

动态模型主要描述系统的动态行为和控制结构。包括4类图：状态图、活动图、顺序图、合作图。

状态图(state diagram)：状态图用来描述对象，子系统，系统的生命周期。

活动图(activity diagram)：着重描述操作实现中完成的工作以及用例实例或对象中的活动，活动图是状态图的一个变种。

顺序图(sequence diagram)：是一种交互图，主要描述对象之间的动态合作关系以及合作过程中的行为次序，常用来描述一个用例的行为。

合作图(collaboration diagram)：用于描述相互合作的对象间的交互关系，它描述的交互关系是对象间的消息连接关系。

UML中的消息

一、简单消息(simple) ——→

表示控制流，描述控制如何从一个对象传递到另一个对象，但不描述通信的细节。

二、同步消息(synchronous) ——→

是一种嵌套的控制流，用操作调用实现。操作的执行者要到消息相应操作执行完并回送一个简单消息后，再继续执行。

三、异步消息(asynchronous) ——→

是一种异步的控制流，消息的发送者在消息发送后就继续执行，不等待消息的处理。

5.1 状态图

状态图(State Diagram)用来描述一个特定对象的所有可能的状态及其引起状态转移的事件。一个状态图包括一系列的状态以及状态之间的转移。

- **状态** 所有对象都具有状态,状态是对象执行了一系列活动的结果。当某个事件发生后,对象的状态将发生变化。状态图中定义的状态有:

初态—状态图的起始点,一个状态图只能有一个初态。

终态—是状态图的终点。而终态则可以有多个。

中间状态—可包括三个区域:名字域、状态变量与活动域。

复合状态—可以进一步细化的状态称作复合状态。

初态

终态

迁移

中间态

响应事件的内部动作或活动的列表,定义为:
事件名 (参数表[条件])/动作表达式

状态变量 是状态图所显示的类的属性。

活动 列出了在该状态时要执行的事件和动作。有3个标准事件:

entry事件用于指明进入该状态时的特定动作。
exit事件用于指明退出该状态时的特定动作。
do事件用于指明在该状态中时执行的动作。

} 无参数

例:

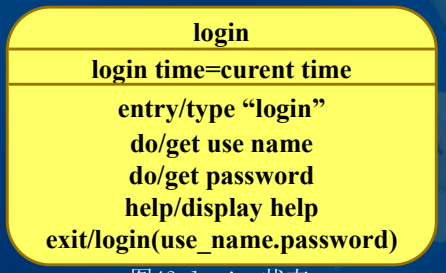


图43 login 状态

- **状态迁移** 一个对象的状态的变迁称为状态迁移。通常是由事件触发的,此时应标出触发转移的事件表达式。如果转移上未标明事件,则表示在源状态的内部活动执行完毕后自动触发转移。

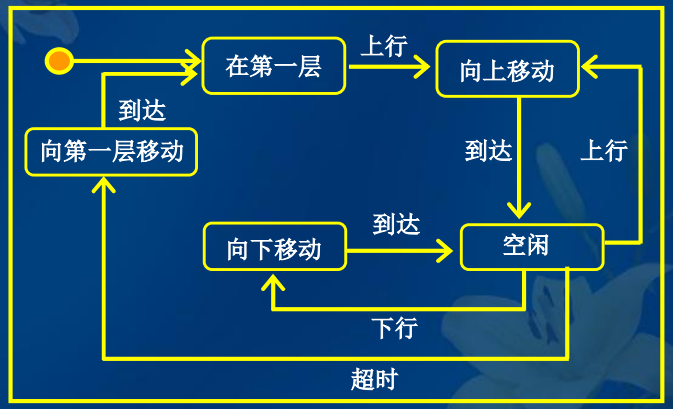
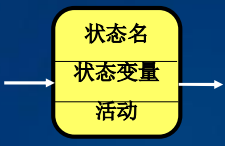


图44 电梯状态图

- **细化的状态表示**

UML给出了电梯细化的状态表示(图47)。



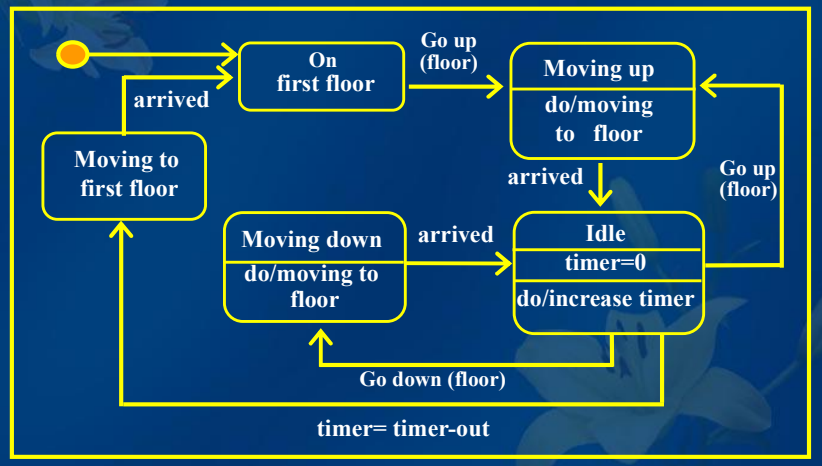


图47 细化电梯状态图

- 嵌套状态图

状态图可能有嵌套的子状态图，且子状态图可以是另一个状态图。子状态又可分为两种：“与”子状态和“或”子状态，

图45 或关系的子状态

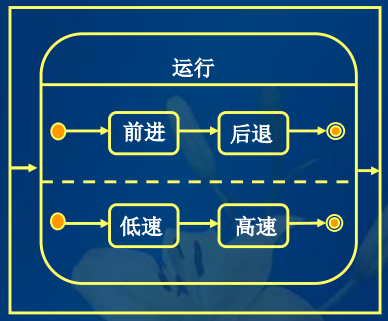


图46 与子状态及或子状态

- 事件

事件是激发状态迁移的条件或操作。

在UML中，有4类事件：

1、某条件变为真；表示状态迁移的上的警戒条件。

2、收到来自外部对象的信号 (signal) 表示为状态迁移上的事件特征，也称为消息。

3、收到来自外部对象的某个操作中的一个调用，表示为状态迁移上的事件特征，也称为消息。

4、状态迁移上的时间表达式。

- 状态图之间的消息发送

状态图之间可以发送消息，用虚箭头表示。

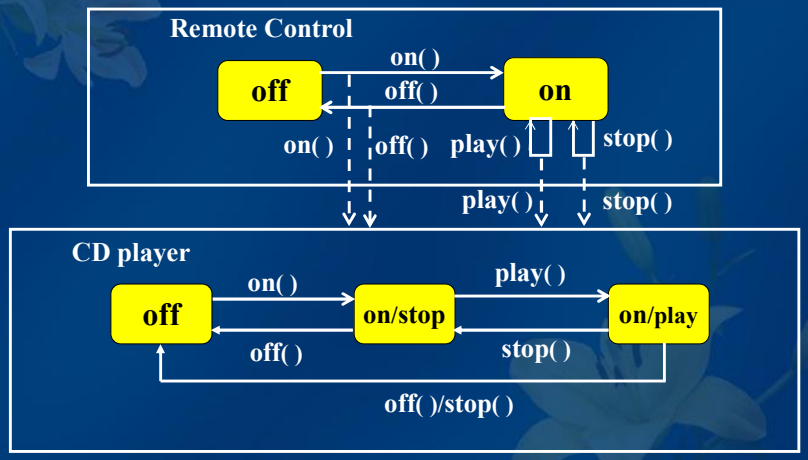
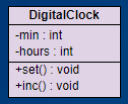
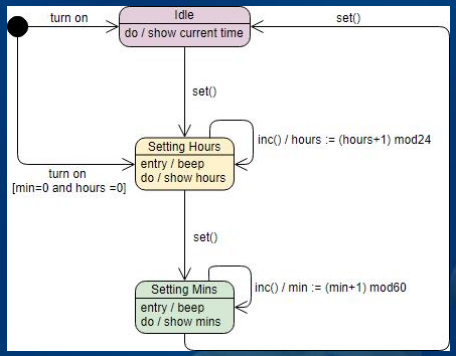


图48 消息发送状态图

状态图示例-数字时钟





状态图示例

自动存取款机 存取款的过程

1、插入银行卡
2、等待输入密码
3、选择服务类型
4、存款
5、取款
6、退出服务（结束退出）

5.2 时序图

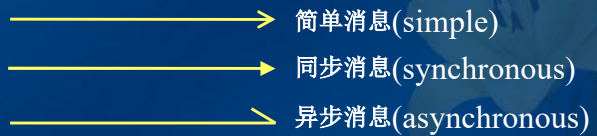
一、概述

- 时序图(Sequence Diagram)用来描述对象之间动态的交互行为,着重体现对象间消息传递的时间顺序。
- 当执行一个用例行为时，其中的每条消息对应一个类操作或状态机中引起转换的触发事件。

5.2 时序图

一、概述

- 时序图存在两个轴:水平轴表示一组对象,垂直轴表示时间。
- 时序图中的对象用一个带有垂直虚线的矩形框表示，并标有对象名和类名。垂直虚线是对象的生命线,用于表示在某段时间内对象是存在的。
- 对象间的通信通过在对象的生命线之间消息来表示，消息的箭头类型指明消息的类型。



二、消息

- 简单消息(simple)：表示消息类型不确定或与类型无关。或者是一同步消息的返回消息。
- 同步消息(synchronous)：表示发送对象必须等待接收对象完成消息处理后，才能继续执行。
- 异步消息(asynchronous)：表示发送对象在消息发送后，不必等待消息处理后，可立即继续执行。
- 消息延迟：用倾斜箭头表示。
- 消息串：包括消息和控制信号，控制信息位于信息串的前部。

控制信息{ 条件控制信息　如：［x＞0］
　　重复控制信息　如：＊［I＝1..n］

当收到消息时,接收对象立即开始执行活动,即对象被激活了，通过在对象生命线上显示一个细长矩形框来表示激活。

三、时序图的形式

有两种使用时序图的方式：一般格式和实例格式。

实例格式详细描述一次可能的交互。没有任何条件和分支或循环，它仅仅显示选定情节（场景）的交互（图49）。

而一般格式则描述所有的情节。因此，包括了分支，条件和循环。

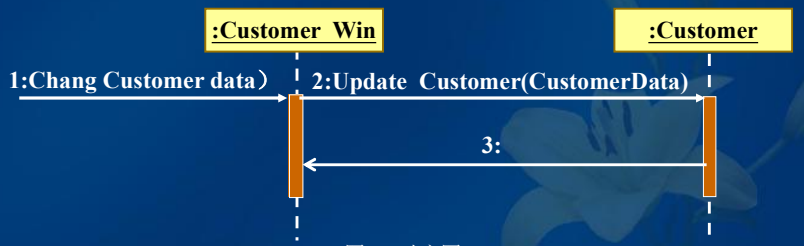


图49 时序图

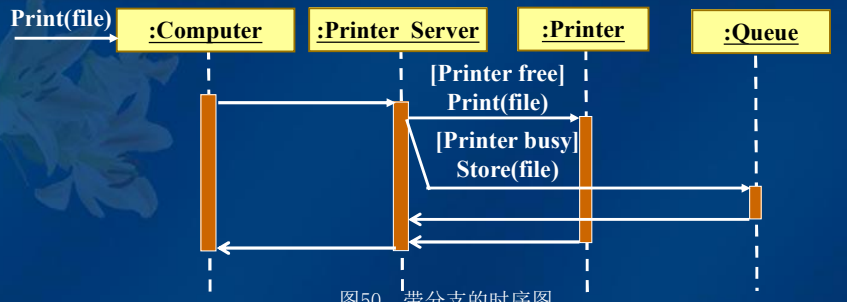


图50 带分支的时序图

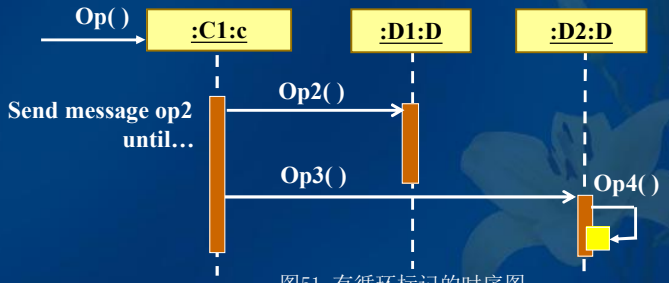


图51 有循环标记的时序图

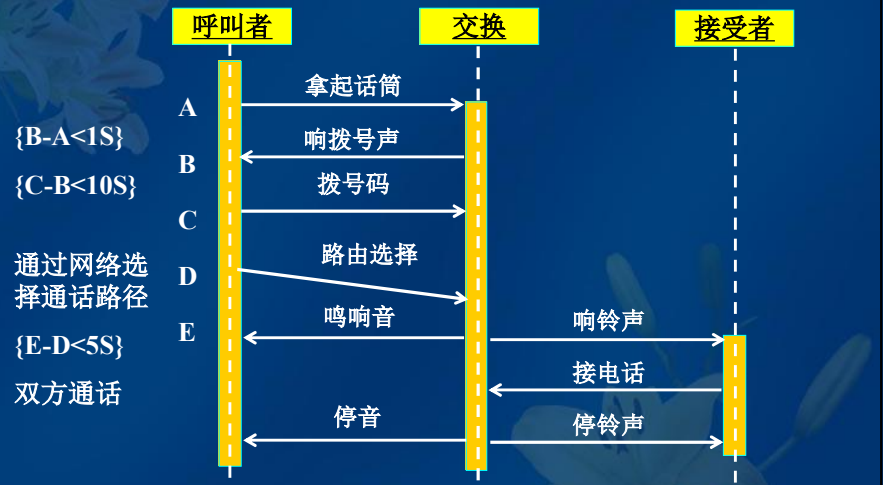


图52 打电话的时序图

创建对象与对象的消亡

在时序图中，还可以描述一个对象通过发送一条消息来创建另一个对象。

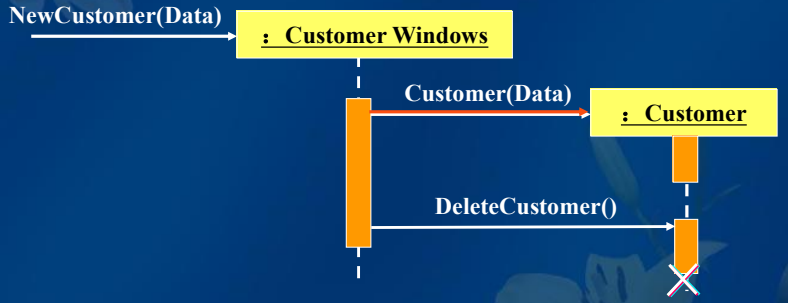


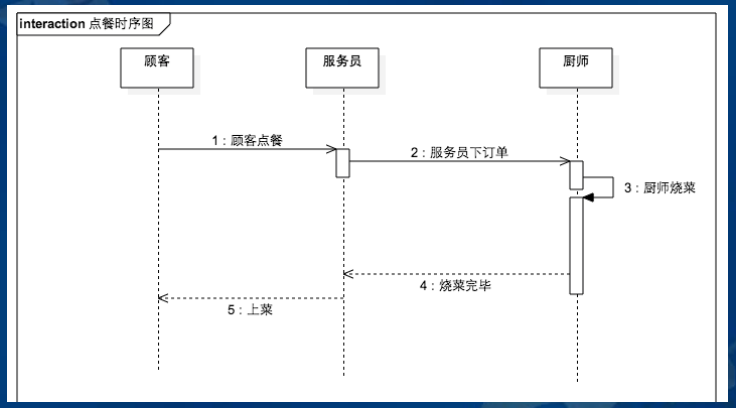
图53 创建或删除对象

当对象消亡(destroying)时，用符号 □ 表示。

时序图创建步骤

- 确定交互过程的上下文；
- 识别参与过程的交互对象；
- 为每个对象设置生命线；
- 从初始消息开始，依次画出随后消息；
- 考虑消息的嵌套，标示消息发生时的时间点，则采用FOC（focus of control）；
- 说明时间约束的地点

时序图示例



时序图练习

- 用户打开微信扫描二维码支付流程
  - 1，用户输入手机密码
  - 2，打开手机
  - 3，打开微信扫一扫
  - 4，返回微信扫一扫界面
  - 5.1 扫描商家收款码
  - 5.2 商家生成收款二维码
  - 5.3 返回收款二维码
  - 5.4 识别商家收款码
  - 6，提示用户输入微信支付密码
  - 7.1 输入微信支付密码
  - 7.2 微信验证用户输入密码正确
  - 7.3 向商家汇款
  - 7.4 汇款成功
  - 8，提示用户支付成功

5.3 活动图

一、概述

活动图(Activity Diagram)的应用非常广泛，它既可用来描述操作(类的方法)的行为，也可以描述用例和对象内部的工作过程，并可用于表示并行过程。

活动图是由状态图变化而来的，它们各自用于不同的目的。活动图描述了系统中各种活动的执行的顺序。刻化一个方法中所要进行的各项活动的执行流程。

活动图中一个活动结束后将立即进入下一个活动(在状态图中状态的变迁可能需要事件的触发)。

活动图是一种描述交互的方式，描述采取何种动作，做什么（对象状态改变），何时发生（动作序列），以及在何处发生（泳道）。

5.3 活动图

活动图的作用

- 描述一个操作的执行过程中所完成的工作或动作
- 描述对象内部的工作
- 显示如何执行一组相关的动作，以及这些动作如何影响周围对象
- 描述用例的执行
- 处理多线程应用

二、活动图的模型元素

构成活动图的模型元素有：活动、转移、对象、信号、泳道等。

1、活动

是构成活动图的核心元素，是具有内部动作的状态，由隐含的事件触发活动的转移。

活动的解释依赖于作图的目的和抽象层次，在概念层描述中，活动表示要完成的一些任务；在说明层和实现层中，活动表示类中的方法。

活动用圆角框表示，标注活动名。

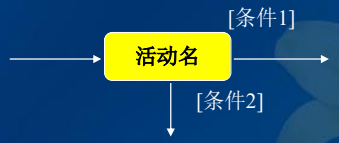


图 54 活动

二、活动图的模型元素

活动还有其它的图符：初态、终态、判断、同步。

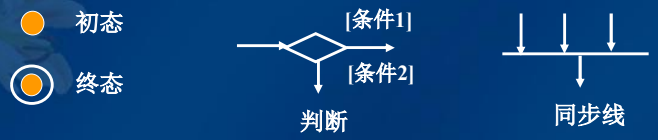


图 55 活动

2、转移

转移描述活动之间的关系，描述由于隐含事件引起的活动变迁，即转移可以连接各活动及特殊活动（初态、终态、判断、同步线）。

转移用带箭头的直线表示，可标注执行该转移的条件，无标注表示顺序执行。

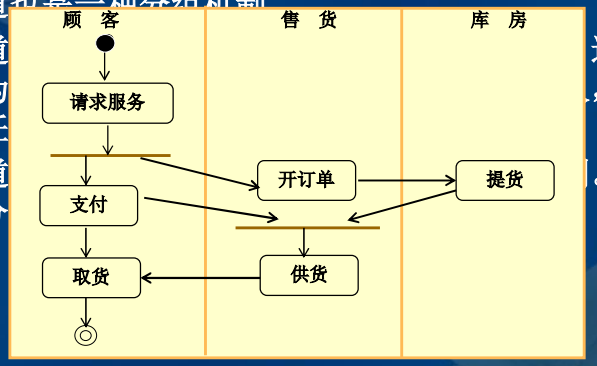
3、泳道

泳道进一步描述完成活动的对象，并聚合一组活动。用于对活动图中的活动进行分组，用于描述对象之间的合作关系。

泳道也是一种分组机制

泳道 这些纵向区域称为 ，或者部门的责任

泳道 。可为提取类及分



三、活动图举例

1、活动图中只有一个起点一个终点，表示方式和状态图一样，泳道被用来组合活动，通常根据活动的功能来组合。具体说泳道有如下目的：直接显示动作在哪一个对象中执行，或显示的是一项组织工作的哪部分。泳道用纵向矩形来表示，如图60。

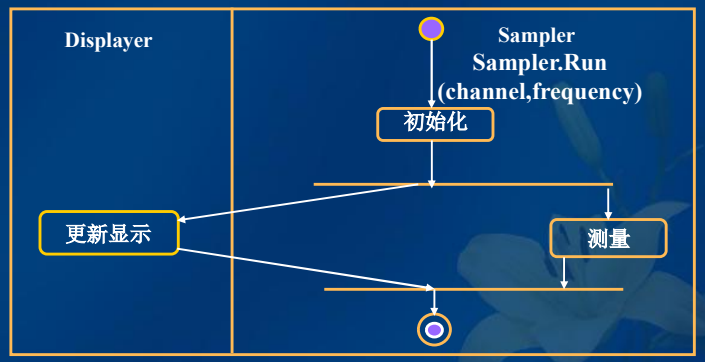


图 60 泳道

4、对象流

活动图中可以出现对象，对象作为活动的输入/输出，用虚箭头表示。

图57 对象流

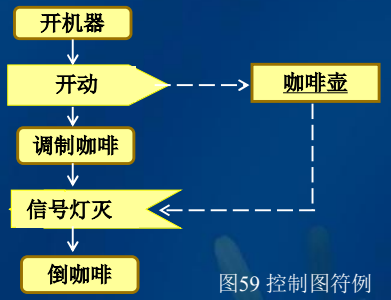


图59 控制图符例

5、控制图符

活动图中可发送和接收信号，发送符号对应于与转移联系在一起的发送短句。接收符号也同转移联系在一起。

图 58 控制图符

2、活动图中可发送和接收信号，发送符号对应于与转移联系在一起的发送短句。接收符号也同转移联系在一起。转移又分两种：发送信号的转移和接收信号的转移。发送和接收信号可以和消息的的发送对象和接收对象联系在一起，如图61。

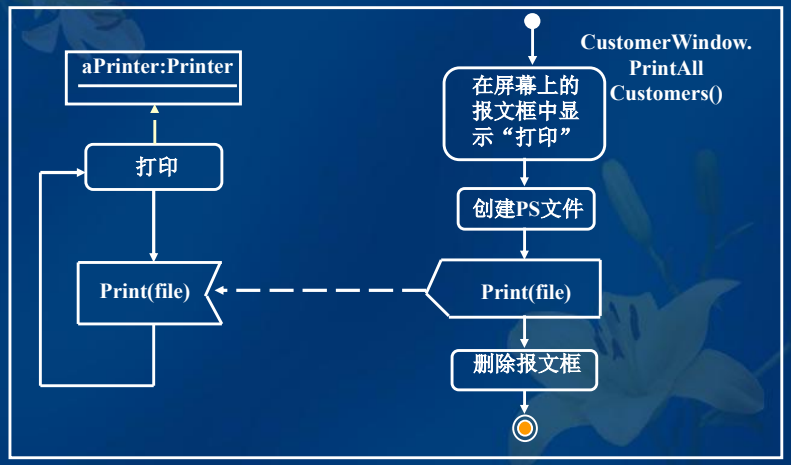


图61

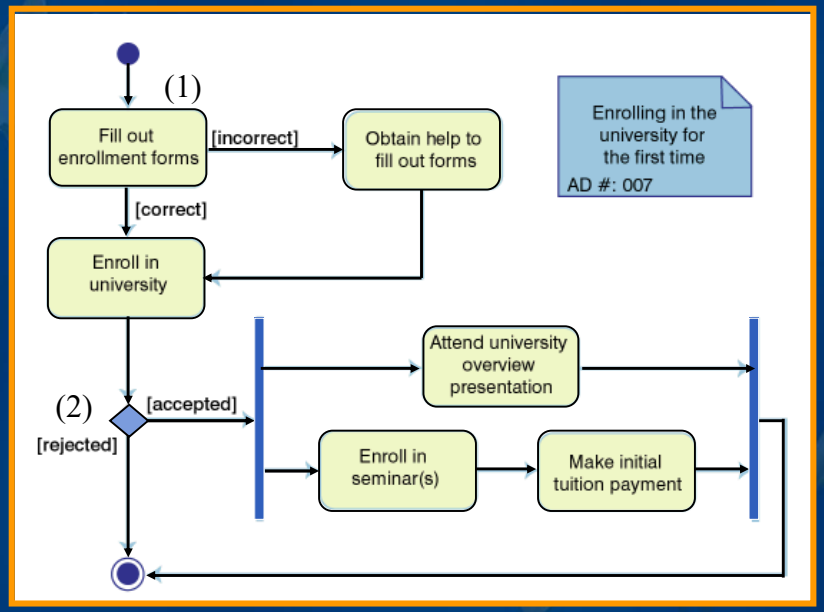


图62 活动图举例

活动图示例

描述一个场景如下：
1）公司业务员打电话给客户，确定一个约定
2）如果约定的地点是在公司内部，那么行政部的同事需要为会面安排一间会议室
3）如果约定的地点时在公司外部，那么业务员需要用笔记本准备一份陈述报告
4）业务员与客户在约定的时间和地点见面
5）业务员准备好会议用纸
6）如果会议产生一个陈述，业务员需要记录下来备案。

5.4 合作图

合作图(Collaboration Diagram)，也称为协作图，用于描述相互合作的对象间的交互关系和链接(Link)关系。

一个协作图显示了一系列的对象和在这些对象之间的联系以及对象间发送和接收的消息。对象通常是命名或匿名的类的实例，也可以代表其他事物的实例，例如协作、组件和节点

虽然时序图和合作图都用来描述对象间的交互关系，但侧重点不一样。时序图着重体现交互的时间顺序，合作图则着重体现交互对象间的静态链接关系。使用合作图可以显示对象角色之间的关系，如为实现某个操作或达到某种结果而在对象间交换的一组消息。如果需要强调时间和序列，最好选择时序图；如果需要强调上下文相关，最好选择合作图。合作图特别适合用来描述少量对象之间的简单交互。随着对象和消息数量的增多，理解合作图将越来越困难。

一、 合作图中的模型元素

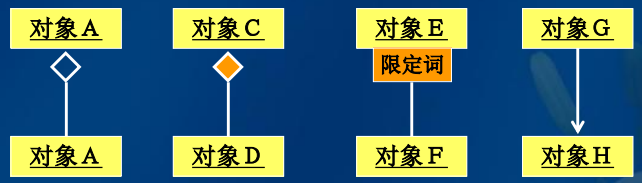
1、对象

合作图中对象的外观与时序图中的一样。如果一个对象在消息的交互中被创建，则可在对象名称之后标以{new}。类似地，如果一个对象在交互期间被删除，则可在对象名称之后标以{destroy}。

对象名{new}　　对象名{destroy}

2、链接(Link)

链接用于表示对象间的各种关系,包括组成关系的链接(Composition Link)、聚集关系的链接(Aggregation Link)、限定关系的链接(Qualified Link)以及导航链接(Navigation Link)。各种链接关系与类图中的定义相同,在链接的端点位置可以显示对象的角色名和模板信息。



对于链接还可以加上“角色”与“约束”，在链角色上附加的约束有global(全局)，local(局部)，parameter(参数)，self(自身)，broadcast(广播)。

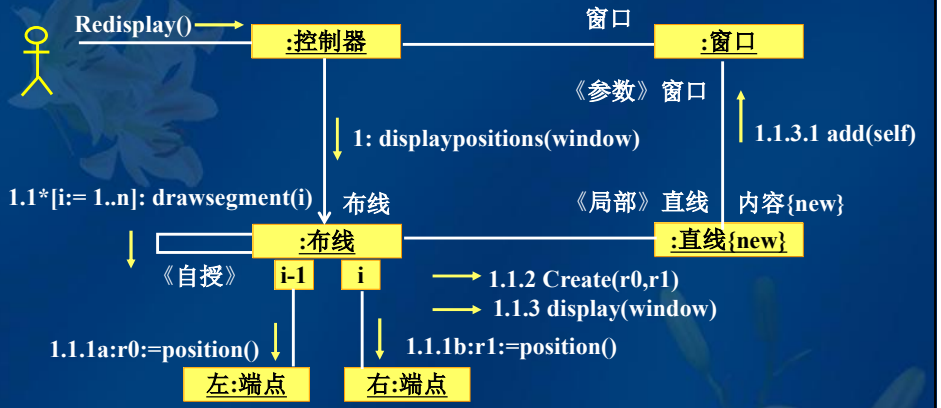
3、消息

在对象之间的静态链接关系上可标注消息，消息类型有简单消息，同步消息和异步消息三种。用标号表示消息执行的顺序。消息定义的格式如下：

消息类型　标号 控制信息：返回值：=消息名 参数表

Predecessor guard-condition sequence-expression return-value:=signature

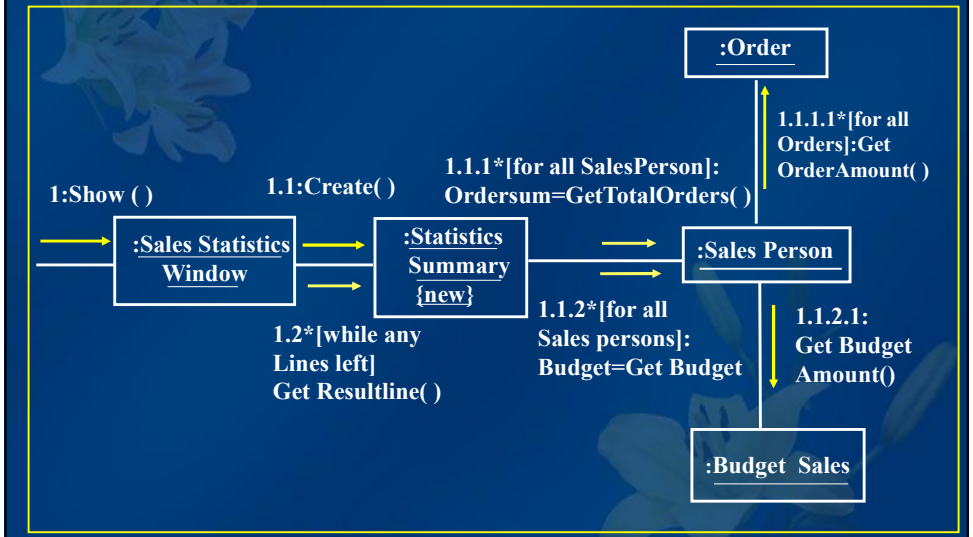
- 标号有 3 种：
  顺序执行：按整数大小执行。 1，2 …
  嵌套执行：标号中带小数点。1.1，1.2，1.3，…
  并行执行：标号中带小写字母。1.1.1a，1.1.1b，…
- 控制信息{ 条件控制信息　如：［x＞y］
  重复控制信息　如：＊［I＝1..n］



电路设计的合作图

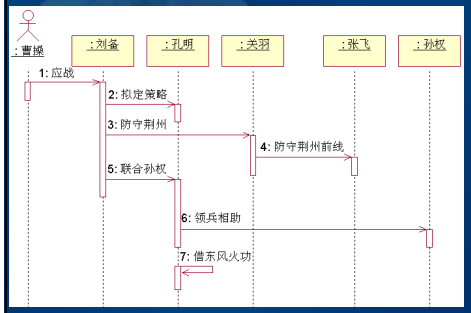
在控制器控制下进行布线，找出左端点r0和右端点r1，创建对象“直线”，并在窗口显示出来。

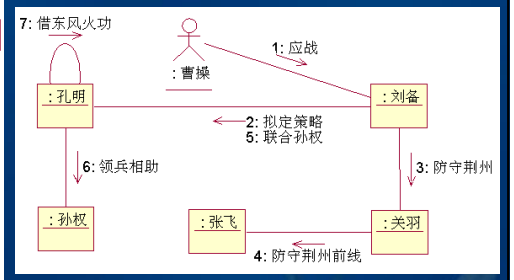
下图为一销售结果统计的合作图。



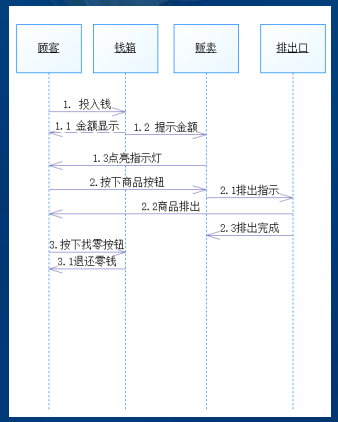
统计销售结果的合作图

时序图与协作图





合作图-操作示例



关于时序图与合作图

1、时序图与合作图都是交互图，它们有何不同？所描述的主要系统特征是什么？

2、时序图与合作图各适合于在哪类系统中使用？

关于状态图与活动图

1、状态图与活动图有何相同与不同之处？

2、在建立系统模型时，应该如何使用这两类模型？

6 实现模型

实现模型描述了系统实现时的一些特性，又称为物理体系结构建模。包括源代码的静态结构和运行时刻的实现结构。实现模型包括：

构件图(Component diagram) 显示代码本身的逻辑结构，它描述系统中存在的软构件以及它们之间的依赖关系。构件图的元素有构件，依赖关系和界面。

部署图(Deployment diagram) 描述了系统中硬件和软件的物理配置情况和系统体系结构。显示系统运行时刻的结构，部署图中的简单结点是指实际的物理设备以及在该结点上运行构件或对象。部署图还描述结点之间的连接以及通信类型。

6.1 构件图

构件（component）

- 构件定义：系统中遵从一组接口且提供其实现的物理的、可替换的部分。每个构件能实现一定的功能，为其他构件提供使用接口，方便软件的复用。对系统的物理方面建模时，它是一个重要的构造块。
- 构件与类的区别：
  - 类表示的是逻辑的抽象，构件是存在于计算机中的物理抽象。构件是可以部署的。类不行。
  - 构件表示的是物理模块，类是逻辑模块
  - 类可以直接拥有操作和属性，构件仅拥有可以通过其接口访问的操作。

构件图

构件（component）

- 构件的名称和类的名称的命名法则很是相似，有简单名和路径名之分。构件的描述如上图所示。
- 若构件的定义良好，该构件不直接依赖于构件的所支持的接口，在这种情况下，系统中的一个构件可以被支持正确接口的其他构件所替代。构件图符是一个矩形框（图66）。
- 构件对外提供的可见操作和属性称为构件的界面。界面的图符是一个小圆圈。用一条连线将构件与圆圈连起来。

构件图

构件可以看作包与类对应的物理代码模块，逻辑上与包，类对应，实际上是一个文件，可以有下列几种类型的构件：

1） 源代码构件；

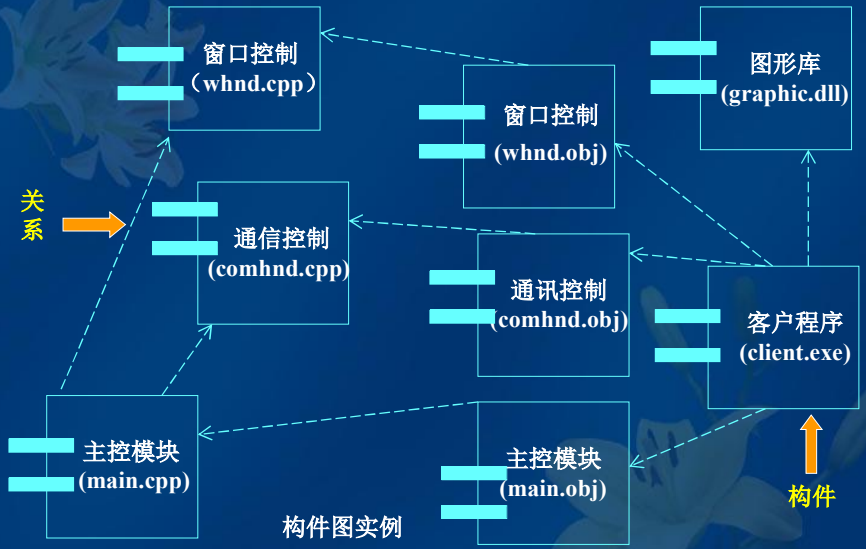
2） 二进制构件；

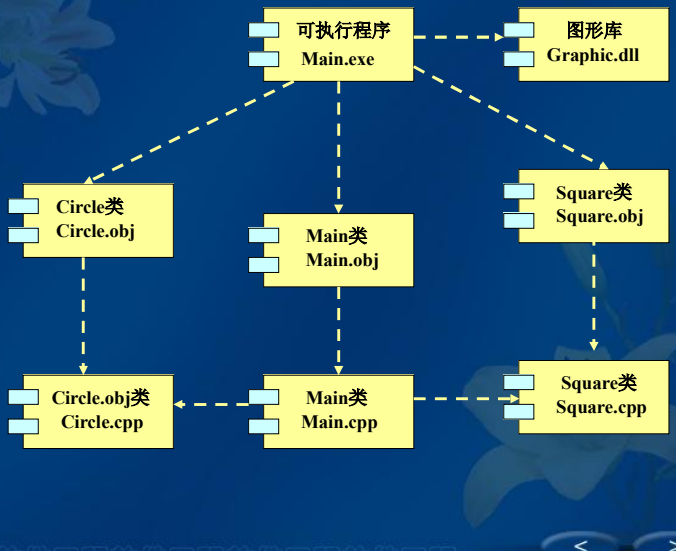
3） 可执行构件

构件图符是一个矩形框。

构件对外提供的可见操作和属性称为构件的界面。界面的图符是一个小圆圈。用一条连线将构件与圆圈连起来。

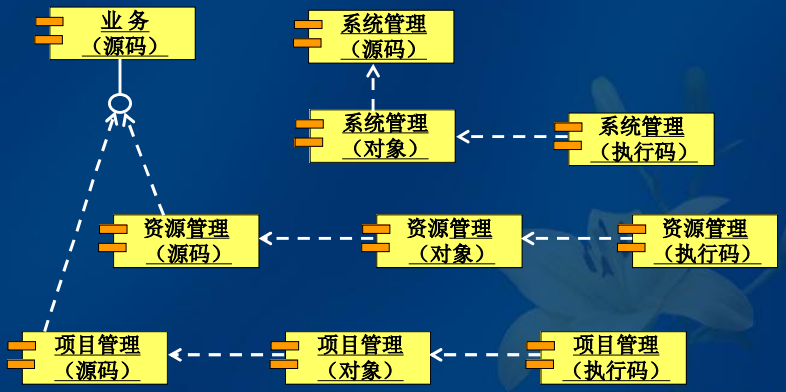
构件之间的依赖关系是指结构之间在编译，连接或执行时的依赖关系。用虚线箭头表示。





组件的依赖关系又分为：开发期的依赖和调用依赖。

- 开发期的依赖(Development –time Dependency)
  是指在编译阶段和连接阶段，组件之间的依赖关系。
- 调用依赖(Call Dependency)
  是指一个组件调用或使用另外一个组件服务。

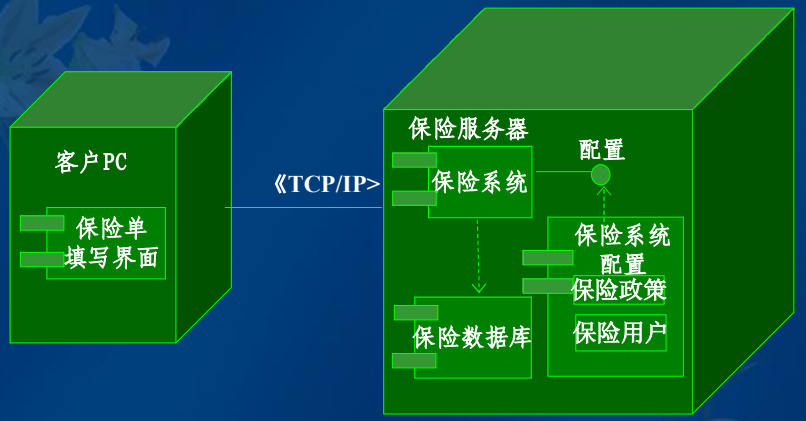


部署图

部署图用来描述系统硬件的物理拓扑结构以及在此结构上执行的软件，即系统运行时刻的结构。

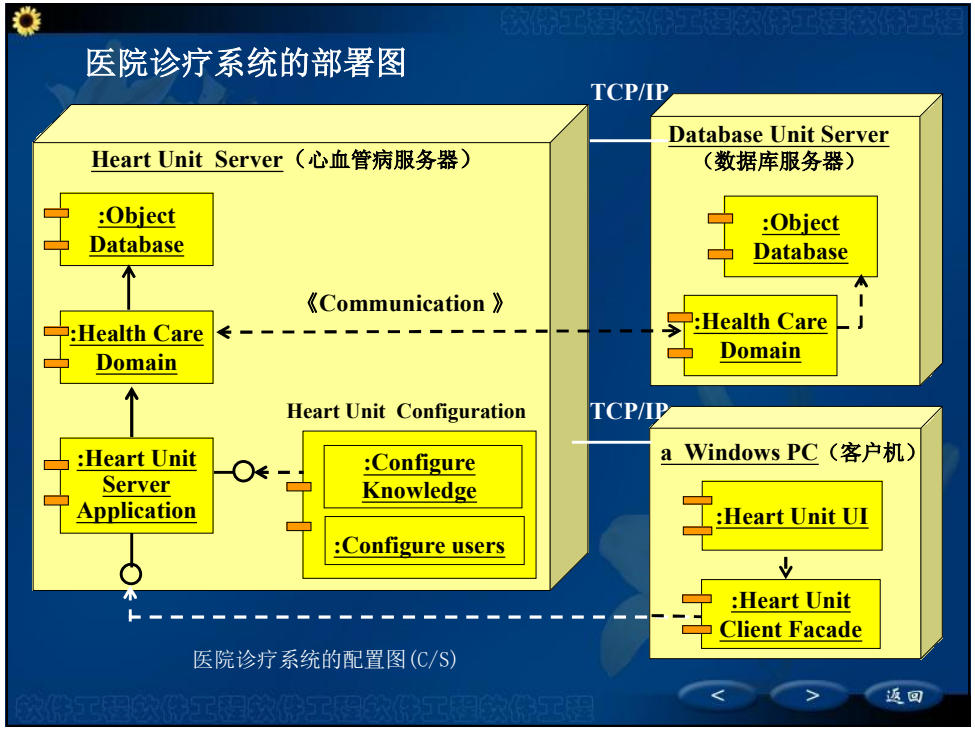
部署图可以显示计算机结点的拓扑结构和通信路径，结点上执行的软构件，软构件包含的逻辑单元等，特别对于分布式系统，部署图可以清楚的描述系统中硬件设备的配置，通信以及在各硬件设备上各种软构件和对象的配置。因此，部署图是描述任何基于计算机的应用系统的物理配置或逻辑配置的有力工具，部署图的元素有结点和连接。

部署图中的结点代表某种计算机构件，通常是某种硬件。同时结点还包括在其上运行的软构件，软构件代表可执行的物理代码模块。如一个可执行程序。 结点的图符是一个立方体。



保险系统的部署图

部署图各结点之间进行交互的通信路径称为连接，连接表示系统中的结点存在着联系，用结点之间的的连线表示连接，在连接的连线上要标注通信类型。



医院诊疗系统的配置图(C/S)

7 使用UML的过程

UML给出了面向对象建模的符号表示和规则，但未给出使用的过程和方法，因此，需要有使用UML的过程。

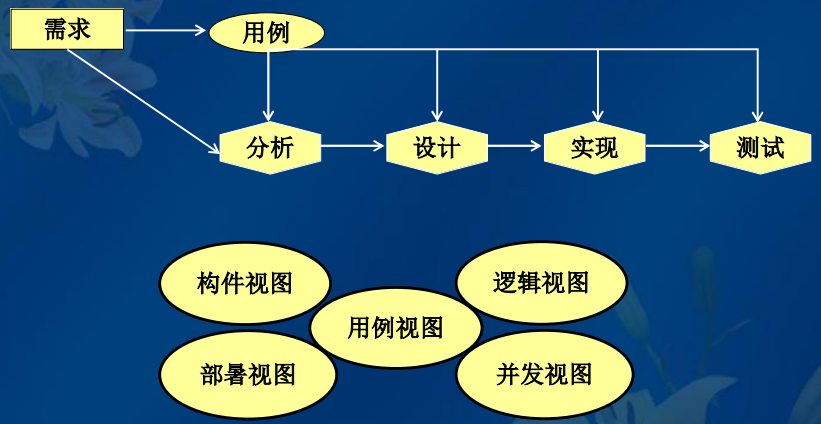
过程描述做什么、怎么做、何时做及为什么做，即描述一组特定次序的活动。

5.7.1 UML过程的基础

使用UML过程的基本特征是：用例驱动，以体系结构为中心，反复，渐增式。

1、用例驱动的系统

用例包含了功能描述，它们将影响后面所有阶段及视图。



用例对各阶段及视图的影响

2、以体系结构为中心

在开发的早期建立基础的体系结构（原型）是十分重要的，进一步对原型进行精化，建立一个易于修改、易理解和允许复用的系统。

主要工作是在逻辑上将系统划分为若干个子系统（UML包）。

3、反复

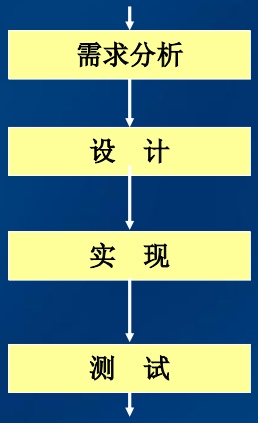
UML的建模型过程要经过若干次的反复。

4、渐增式

渐增式开发是在多次反复迭代的过程中，每次增加一些功能（或用例）的开发，每次迭代都包含了分析、设计、实现和测试。

7.1 面向对象的开发方法的一般过程

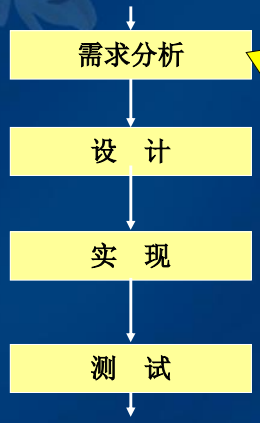
ＯＯ方法中执行主要活动的描述。主要步骤是分析、设计、实现及测试。



OO方法的步骤

7.1面向对象的开发方法的一般过程

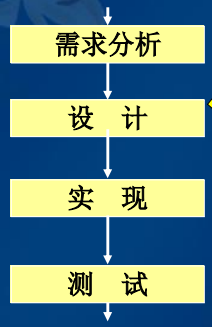
ＯＯ方法中执行主要活动的描述。主要步骤是分析、设计、实现及测试。



获取需求，建立需求模型。
分析的典型活动：
1、获取领域知识；
2、定义系统功能（用例图）；
3、确定合适的类；
4、建立类的静态模型（类图）；
5、描述对象的动态行为（状态图、协作图、时序图、活动图）；
6、验证（专家对模型作静态验证）；
7、给出基本的用户界面原型（整体结构的原型：主窗口的内容、窗口之间的导航等）。

7.1面向对象的开发方法的一般过程

ＯＯ方法中执行主要活动的描述。主要步骤是分析、设计、实现及测试。

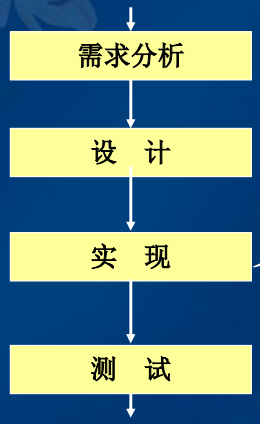


设计是分析结果在技术上的扩充和修改，重点是如何实现该系统。
设计中的典型活动：
1、将分析所得的类划分为功能包，并从技术层面（用户界面、数据库处理、通信等)增加新包，建立包之间的通信联系。
2、标识并发需求并建模。
3、指出系统输出的详细格式：如用户界面，报告，向其它系统发送的事务等。
4、数据管理(建立类与表单的对应关系，对数据库的访问机制)。
5、异常处理
6、分配类和构件（构件图、配置图）。

还应用伪代码或者文字给出类的规约。

7.1面向对象的开发方法的一般过程

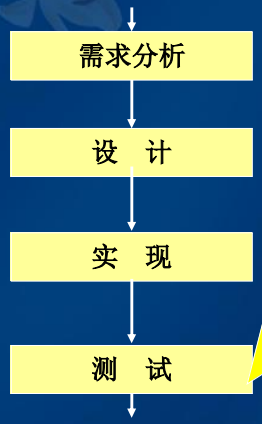
ＯＯ方法中执行主要活动的描述。主要步骤是分析、设计、实现及测试。



实现活动实际上就是编写程序代码，包括反复的编译、连结、排错等。
并应遵循传统的编程准则。

面向对象的开发方法的一般过程

OO方法中执行主要活动的描述。主要步骤是分析、设计、实现及测试。



测试的目的是发现代码中的错误，测试的关键是确定高效的测试用例。测试的主要步骤有：

1、面向对象的单元测试

测试 为封装的类和对象，但不能孤立地测试单个操作，应把操作作为类的一部分来测试。

2、面向对象的集成测试

集成测试的策略有：

①基于线程的测试(Thread-based testing)

②基于使用的测试(Use-based testing)

3、面向对象的确认测试

类似传统的确认测试和系统测试，根据动态模型和描述系统行为的脚本来设计测试用例，可用黑盒法。

8 面向对象的体系结构

体系结构建模（architecture modeling）首先要建立基本的模型，并将该模型映射到软硬件单元上。本节讨论用UML为系统体系结构建模。

体系结构概括了整体系统结构、功能部件分解、部件的本质和特性、部件的界面、部件之间的通讯协议和整体性布局策略及法则。

面向对象的体系结构与传统的体系结构不同，它强调的是分布式对象的分配、部件及其界面、持久对象和面向对象通讯方法。

8.1 系统设计的任务

在设计阶段要解决“如何做”的问题，首先要解决高层问题的决策，再逐步细化。系统设计是解决如何做的第一步，系统设计阶段的主要任务有：

1、将系统分解为子系统；

2、识别问题中固有的并发性；

3、把子系统分配给处理器和子任务；

4、选择数据存储管理的方法；

5、处理访问全局资源；

6、选择软件中的控制实现；

7、处理边界条件；

8、设置权衡的优先权。

显然，系统体系结构设计与程序设计之间没有绝对的界限。

UML体系结构设计

从一般意义上说，体系结构包括两个层面，即硬件体系结构和软件体系结构。

硬件体系结构指系统的硬件组织模式；而软件体系结构则描述软件的组织模式。这里我们主要关注软件体系结构的问题。

1、用包图或构件图描述的静态结构

2、基于配置图的软件体系结构

3、基于模式的软件体系结构

8.2 将系统分解为子系统

子系统的分解可以有分层和分块

分层：将软件系统组织为层次结构，每层是一个子系统。分层结构又分为封闭式和开放式。

分块：垂直分块是将系统分解为若干个相对独立的、弱耦合的子系统。每个子系统相当于一个块，每块实现一类服务。

常常使用分层和分块的混合结构，如图所示：

<table>
<tr><td colspan="3">应用程序包</td></tr>
<tr><td rowspan="3">人机对话控制</td><td>窗口图形</td><td rowspan="3">仿真程序包</td></tr>
<tr><td>屏幕图形</td></tr>
<tr><td>点阵图形</td></tr>
<tr><td colspan="3">操作系统</td></tr>
<tr><td colspan="3">计算机硬件</td></tr>
</table>

8.3 描述系统的体系结构

一、用包图进行系统建模

1、包是系统的一种分组机制，包由关系密切的一组模型元素构成，包还可以由其它包构成（嵌套）。下图描述了体系结构的包图。

包图是维护和控制系统总体结构的重要建模工具。

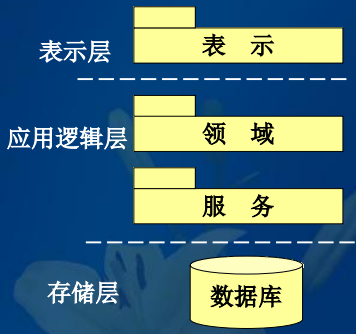


图72 UML包表达的体系结构单元

2、应用包图要解决的问题

(1)如何组织包？

应将概念或语义相近的模型元素(对象类)纳入一个包。即包具有高内聚性，包中的类具有功能相关性。

(2)如何确定包之间的关系？

包之间的联系有两种：依赖和泛化。

包之间的依赖关系，最常用的是输入依赖关系《Import》、《Access》，两者之间区别是后者不把目标包内容加到源包的名字空间。

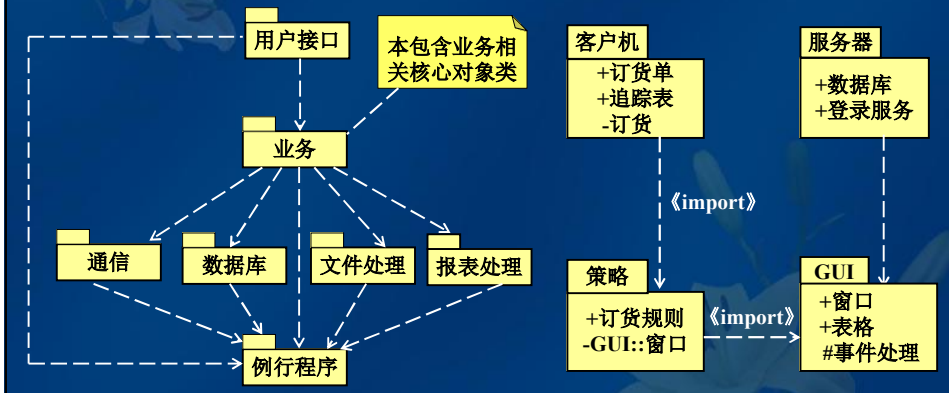


图73 信息管理系统的包图

图74 包的输入依赖关系

二、用配置图描述系统结构

配置图是一个构架，用来详细说明技术单元和它们之间的链接。又可分为硬件环境的配置图和软件环境的配置图。

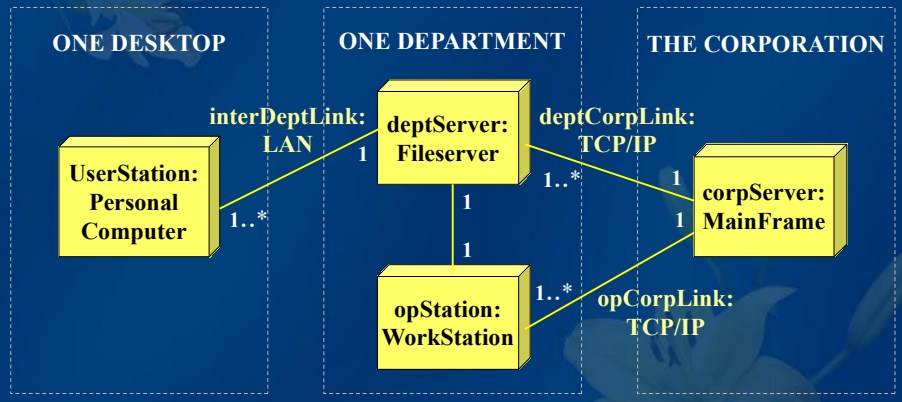


图74 三层C/S商业系统的配置图

并不是所有的系统都需要建立配置图，一个单机系统只需建立包图或构件图就行了。配置图主要用于在网络环境下运行的分布式系统或嵌入式系统的建模。

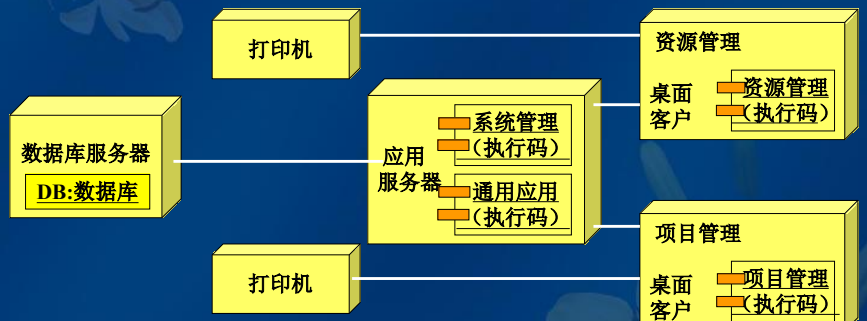
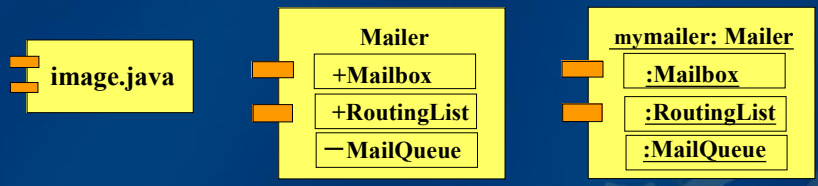


图75 项目与资源管理系统的配置图

三、用构件图来描述系统软件环境的配置

构件图又称为组件图，组件（Component）是系统的物理可替换的单位，代表系统的一个物理组件及其联系，表达的是系统代码本身的结构。

1、简单组件与扩充组件



2、组件的实例

表示运行期间可执行的软件模块。只有可执行的组件才有实例。

3、组件与类的异同

相同：性质的表示法相同（如可见性），都有实例。

不同：组件表示物理的事物，类代表事物的逻辑抽象。

组件可用于配置图的节点，而类不行。

4、组件与包的异同

相同：都是分组的机制。

不同：一个组件表示一个物理的代码模块，包可包含成组的模型元素或物理的组件。

一个类可以出现在多个组件中，却只能在一个包中定义。

5、组件之间的联系

主要是依赖关系，是指一个组件的模型元素使用另外一个系统的模型元素。组件还可以通过接口实现依赖关系。

9 面向对象体系结构的通用样式

硬件体系结构指系统的系统组织模式；而软件体系结构则描述软件的组织模式。

软件体系结构的通用样式又称为软件体系结构的通用模式。

什么是软件体系结构的通用模式？

设计样式(desing pattern)，由一些更基本的成分构成，是进行设计的“砖头”，可以用于同类的其它设计，也称为模型架构(model framework)。

9.1 基于模式的软件体系结构

硬件体系结构指系统的硬件组织模式；而软件体系结构则描述软件的组织模式。这里我们主要关注软件体系结构的问题。什么是软件体系结构的通用模式？

1、体系结构模式（architecrul pattern）

体系结构模式表示软件系统的基本结构化组织图式。体系结构模式可以作为具体软件体系结构的模板。

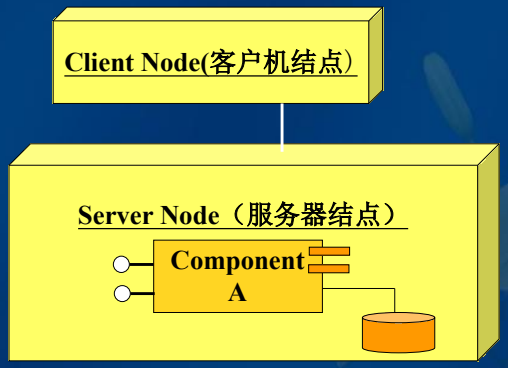
2、设计模式(desing pattern)

由一些更基本的成分构成，是进行设计的“砖头”，可以用于同类的其它设计，也称为模型架构(model framework)。它用于细化软件系统的子系统或组件。

体系结构图的标记法

通常采用一组简单的工程式样标记来构造体系结构图（architecture diagram）,这组标记是以UML的配置图为基础的。

配置图由多个结点（node),连接器（connector）构成。

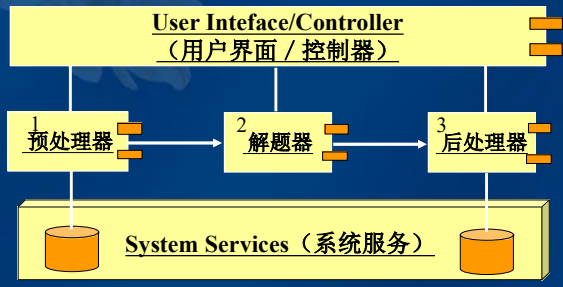


通用模式

- 流程处理系统
- 层状系统
- 客户机/服务器系统
- 三级和多级系统
- 代理

9.1 流程处理系统

流程处理系统（procedural processing system）以算法、数据结构为中心，按照Ｉ-Ｐ-Ｏ过程进行处理。



系统的主要特色是：三个处理部件之间是单向连接的，可能安装在不同的电脑上。

常用于数据与图象、模拟、数值解等。

优点：系统由各处理部件简单组合，易于扩充。处理部件易于复用。该结构适合于大规模并行计算，解决复杂的工程技术问题。

限制：主要以批处理方式执行，不适合交互方式。不易管理大量的不同格式的输入、输出数据。

9.2 层状系统

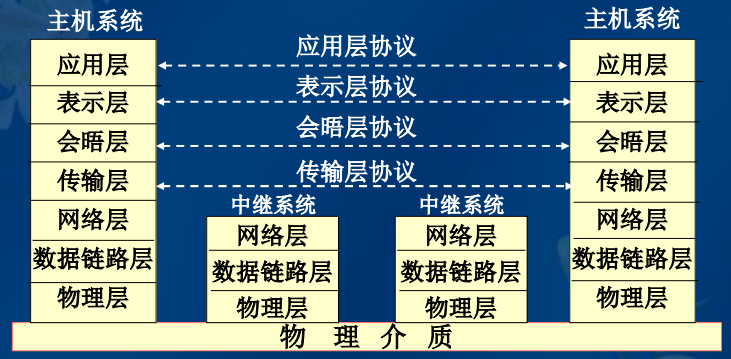
层(layer)，是一个部件或结点中的一组对象或函数，共同协作提供服务。如服务器中里层给外层提供服务。层状体系结构适用于应用服务器、数据库系统及操作系统等。

图 4 .15 层状系统

优点：1、按照功能层次划分，可降低系统复杂度，使系统设计更加清晰。
2、内层与外界隔离，可有效控制内层的函数和服务。
3 、新的运算及常用运算（查询）可在界面层中引入，由内层服务支持，可提高系统性能。
4 、独立的层，可以作为构件或结点使用。

限制：1、层数过多，系统性能下降。
2、标准化的层界面可能变得臃肿，使函数调用性能下降。

ISO/OSI 开放系统互连参考模型



Open System Interconnection Reference Mode

NOS依靠在各网络层次上（OSI七层参考模型）的协议实现通信。

9.3 客户机/服务器系统

在client/server模式下，客户机负责用户输入和展示，服务器处理低层的功能。

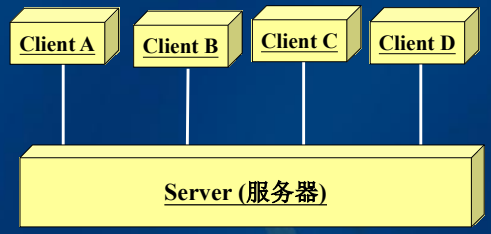


图14 客户机/服务器系统

优点：
1、客户机与服务器分离，两者开发可同时进行。
2、一个服务器可服务于多个客户机。

限制：
1、客户机与服务器的通信依赖于网络，可能出现网络阻塞的瓶颈(bottleneck)现象。
2、服务器及界面的改变将引起客户机的相应改变。

一、如何解决客户机/服务器系统的限制？

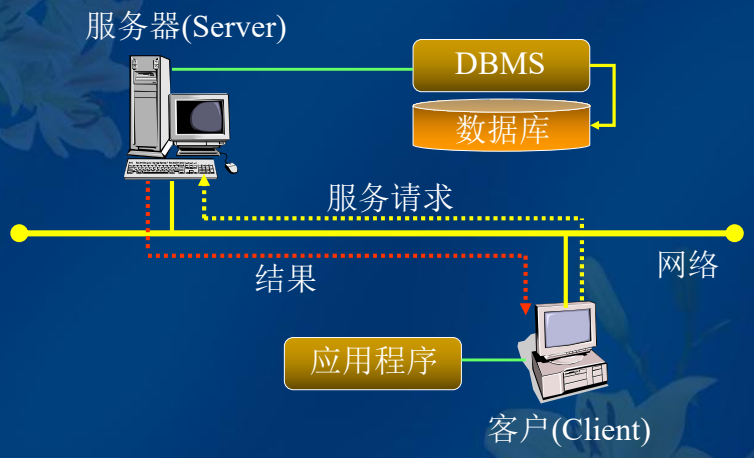
1 、针对网络瓶颈问题，尽量减少客户机作远程调用，如把一组运算组合起来，在一个远程调用中处理。

2 、为了解决服务器的变动引起客户机的改变，采用“轻型客户机” (thin cilent)。如Web浏览器/服务器模式，简称B/S（Browser/Server）模式。

□ 它无需在不同的客户机上安装不同的客户应用程序，而只需安装通用的浏览器软件。这样不但可以节省客户机的硬盘空间与内存，而且使安装过程更加简便、网络结构更加灵活。

□ 简化了系统的开发和维护。系统的开发者无须再为不同级别的用户设计开发不同的客户应用程序了，只需把所有的功能都实现在Web服务器上，并就不同的功能为各个组别的用户设置权限就可以了。

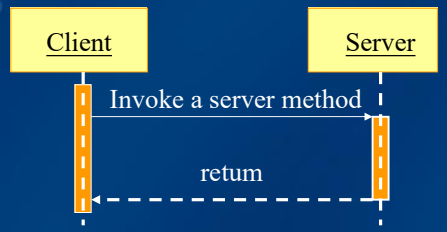
C/S结构的数据库



典型的客户/服务器(C/S)结构

C/S结构的工作模式

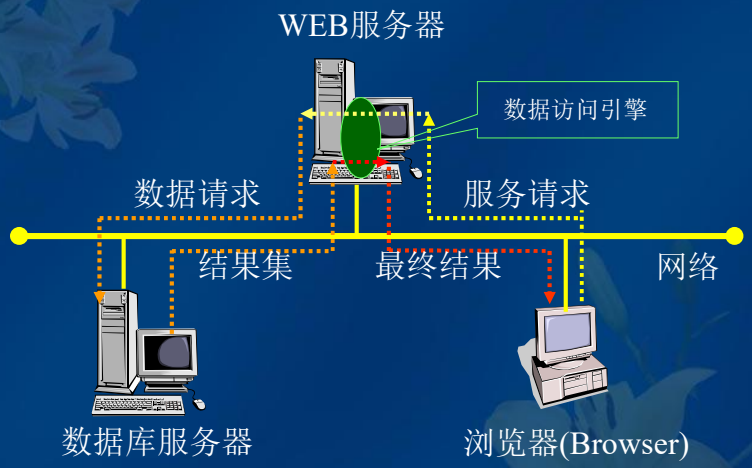
从图中可以看出，C/S结构是一种发请求、得结果的模式：客户机向服务器发出请求（数据请求、网页请求、文件传输请求等等），服务器响应这些请求，进行相应的操作，将得到的结果回传给客户机，客户机再将格式化后的结果呈现在最终用户的眼前。很明显地，客户机、服务器都必须遵循相同的通信协议。在这里，客户机和服务器都是纯软件的概念，多数情况下它们都是操作系统的应用进程。客户机可以是用户自己编写的应用程序或者就是WEB浏览器；服务器可以是数据库服务器、WEB服务器或者其它的服务进程。



C/S结构的设计要点：

服务器不公开其内在的数据结构和运算，所有服务器的参数，都只能通过运算来实现。

B/S结构的数据库



基于WEB的浏览器/服务器(B/S)结构

二、如何提高服务器的效率

1、把部分工作转移到客户机中执行，如查证输入数据的正确性，可在客户端进行。

2、将数据库的处理事项组合起来执行，可提高数据库的吞吐量(throughput)。

94 三级和多级系统

- 第一级是数据库管理结点(database management node)。
- 第二级或中间级是“商业逻辑结点” (business logic node),是指具体应用中实施的 程序逻辑和法则。
- 第三级是用户界面级，强调高效、方便易用的用户界面。

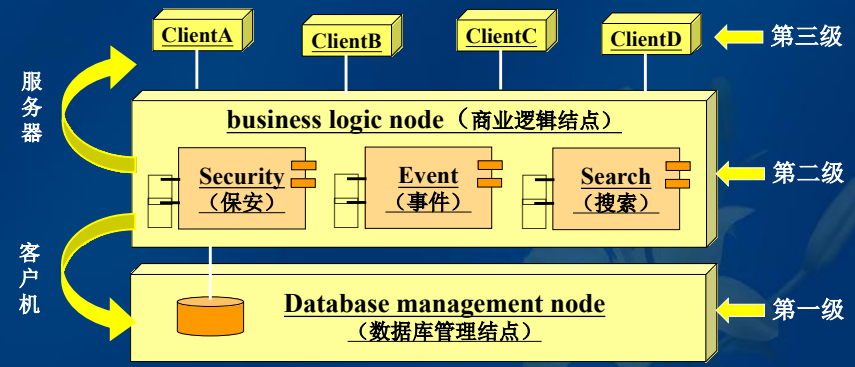


图16 三级体系结构

多级系统

可由三级系统的概念推广到多级系统(multi-tier system)，即由多个C/S对组成。

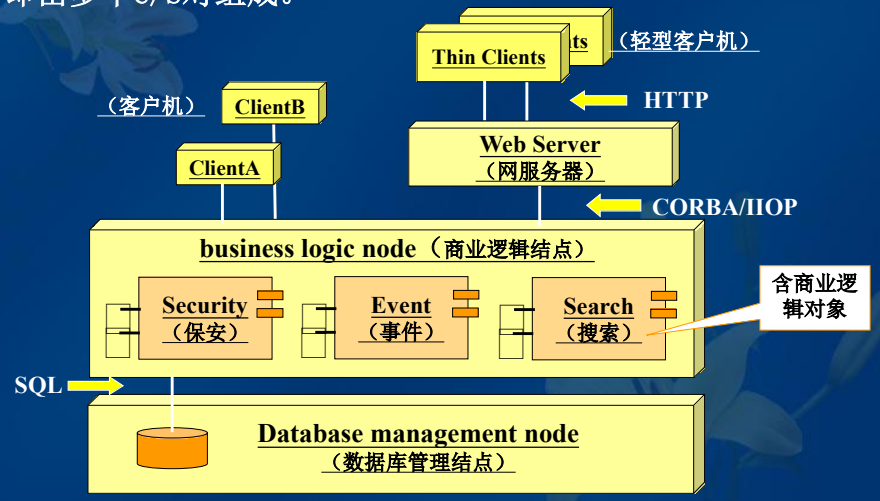


图17 四级体系结构

多级系统的特点

优点：

1、系统功能分布在多级或服务器上，系统易于维护和扩充。

2、进行分级控制，可对不同级的客户机提供不同水平的服务。

3、可方便地将中间级与企业的其它系统连接起来。

4、多级系统可以对同时使用系统的客户机提供服务。

限制：

1、各对客户机与服务器之间有多种不同的通讯协议，要求熟悉不同协议的专业人员。

2、数据行经的多级结点，分布在不同的计算机系统中，因此系统的整体运作性困难。

9.5 代理

- 从部件的角度看，代理是服务器。
- 从体系结构的角度看，代理是模拟企业工作流程中的行动者。在代理的体系结构中，需要一个控制器。

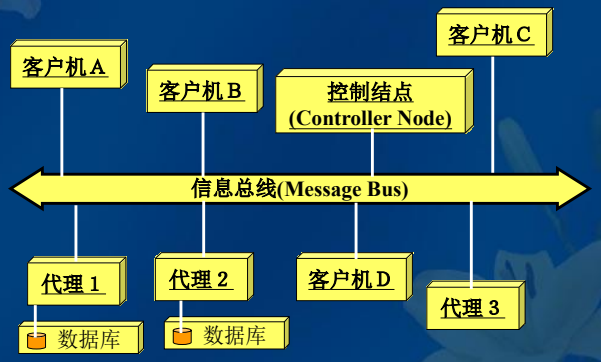


图18 一个有代理的体系结构

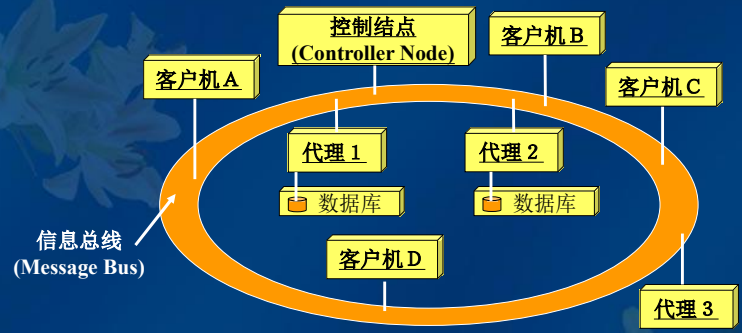


图1 9 有代理的体系结构描述

优点：对复杂任务的处理，代理体系结构具有高度灵活性，各组功能分布在不同的代理，系统易修改、扩充、伸缩性强，便于与企业级水平的软件整合。系统可采用渐进方式建立。

限制：使用公共消息总线，须解决系统安全性问题。要求有统一的通讯协议。系统性能调试困难。