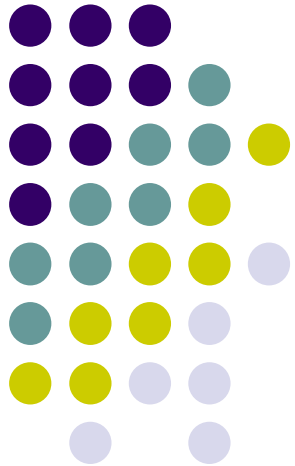
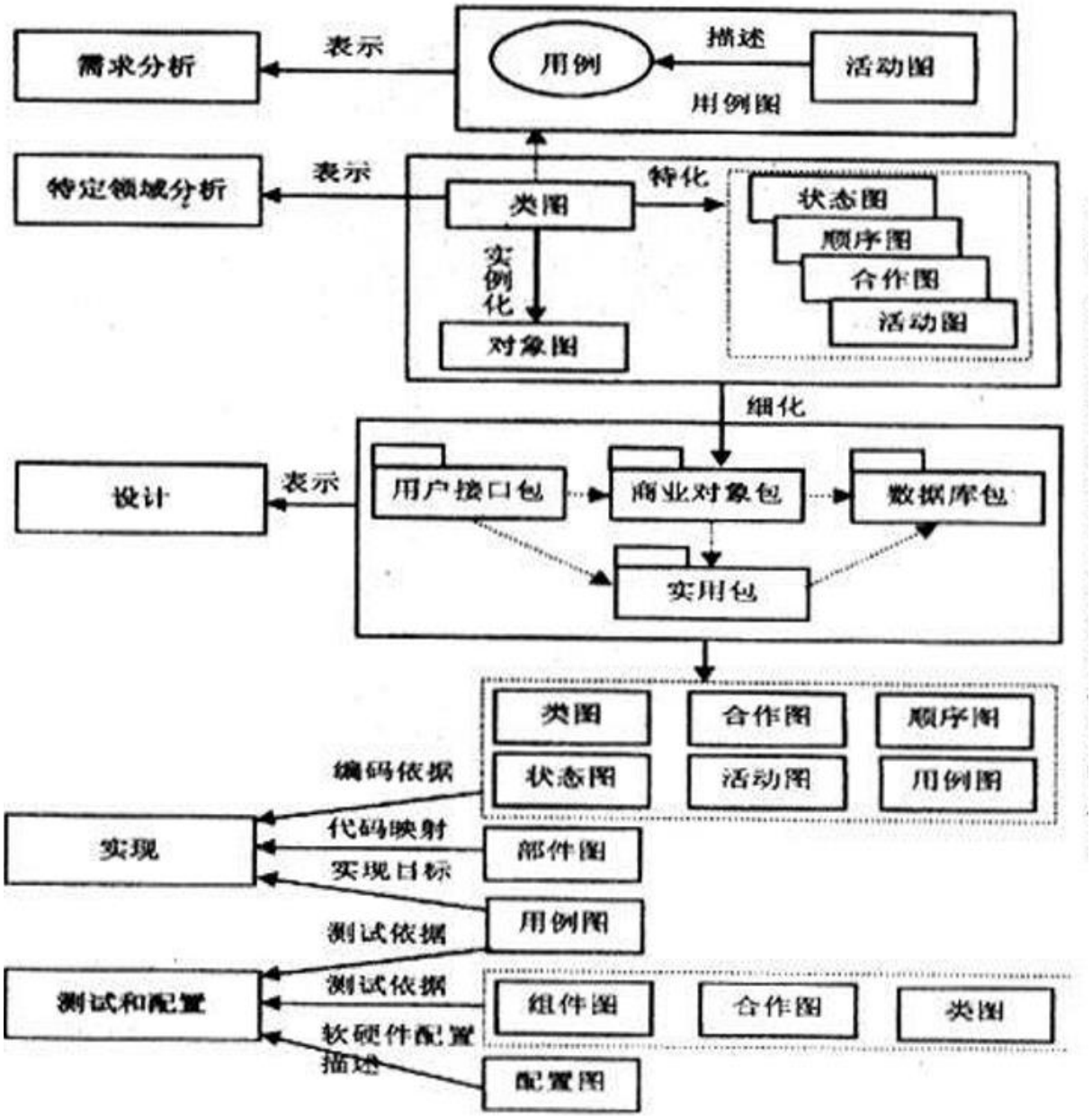
软件系统分析与设计

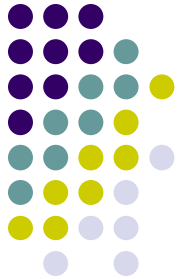
Lecture08 Physical modeling

by Dr Y.Yang
Associate Professor
School of Computer Science & Technology
Soochow University
yyang@suda.edu.cn



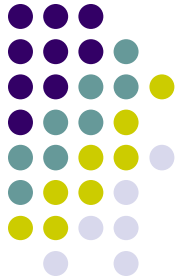

需求分析
表示
用例
描述
活动图
用例图
特定领域分析
表示
类图
特化
状态图
顺序图
合作图
活动图
实例化
对象图
细化
设计
表示
用户接口包
商业对象包
数据库包
实用包
类图
合作图
顺序图
状态图
活动图
用例图
编码依据
实现
代码映射
实现目标
部件图
用例图
测试依据
测试和配置
测试依据
软硬件配置描述
组件图
合作图
类图
配置图

Review



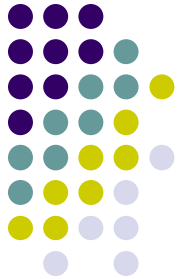
- 了解系统中工作流程和对象状态变化的描述方式
- 了解引起对象状态迁移的事件的描述方法
- 掌握**UML**绘制状态图的方法和步骤
- 掌握并发状态图的描述方法
- 掌握**UML**绘制活动图的方法和步骤
- 掌握时序图、协作图、状态图和活动图建立动态行为模型的方法和步骤

Today's Topics



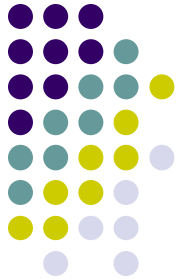
- 了解系统逻辑体系结构模型的组成
- 了解系统物理体系结构模型的组成
- 掌握UML中构件和构件图的描述方法
- 了解源代码构件、二进制代码构件和可执行代码构件的区别
- 掌握构件接口的描述方法
- 掌握UML中部署图的描述方法
- 掌握部署图中节点、构件和对象之间的关系

逻辑建模和物理建模（实现建模）



- 系统体系结构建模可分为逻辑体系结构建模和物理体系结构建模。
- 逻辑体系结构，就是对系统的用例、类、对象、接口以及相互之间的交互和协作进行描述。
- 物理体系结构则是对构件、节点的配置进行描述。

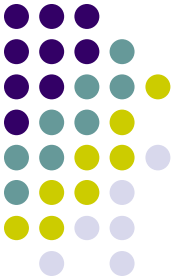
逻辑体系结构



- 系统的逻辑体系结构就是软件体系结构。逻辑体系结构把系统的系统的各种功能分配到系统的不同组成部分，并详细的描述各个组成部分之间是如何协调工作来实现这些功能的。

- 为了清晰的描述一个复杂的系统，通常把系统分为若干较小的子系统。如果需要，子系统还可以继续细分。在**UML**中引入“包”的机制，一个包相当于一个子系统。

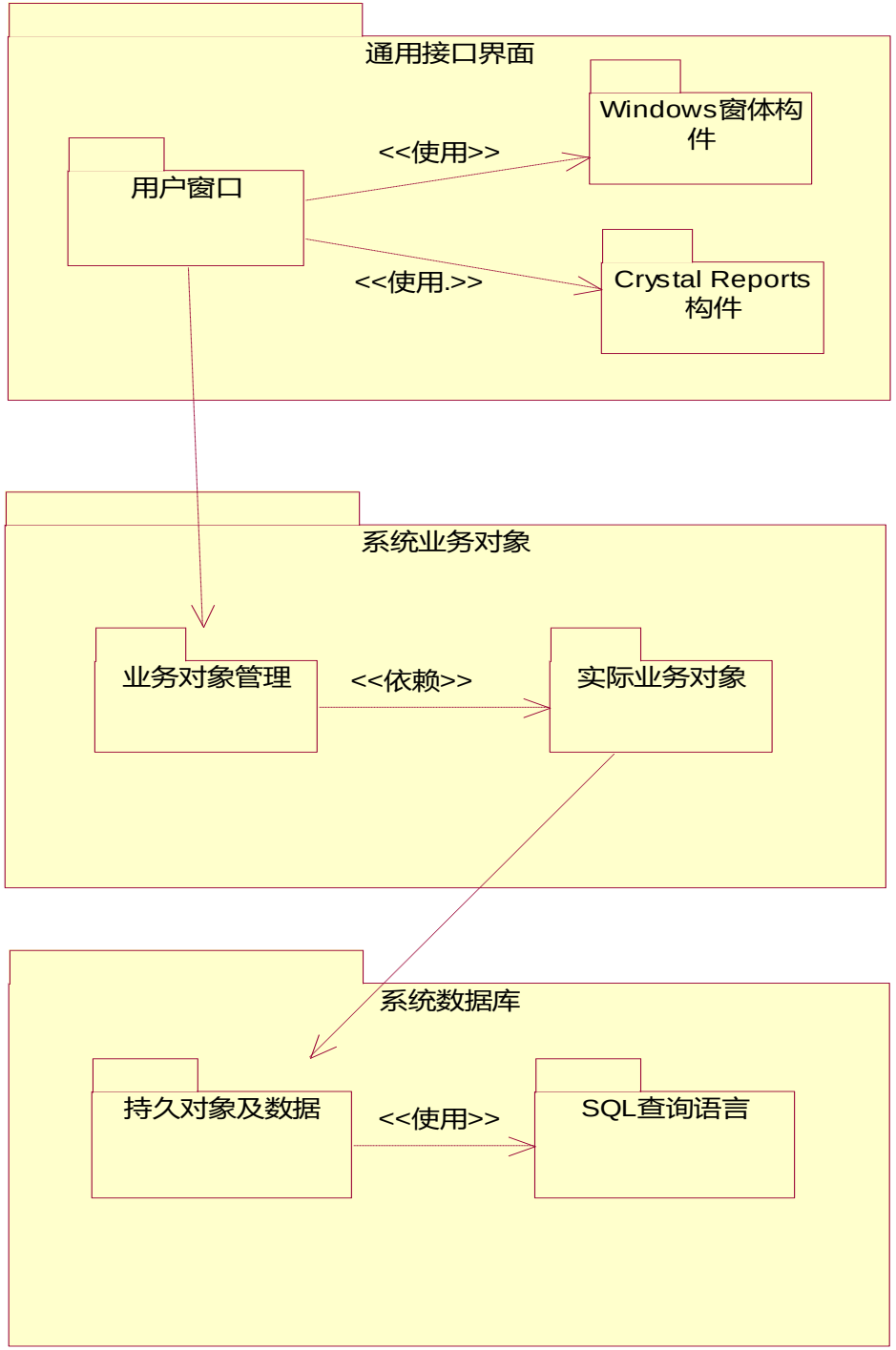
包是逻辑建模中重要的分组机制



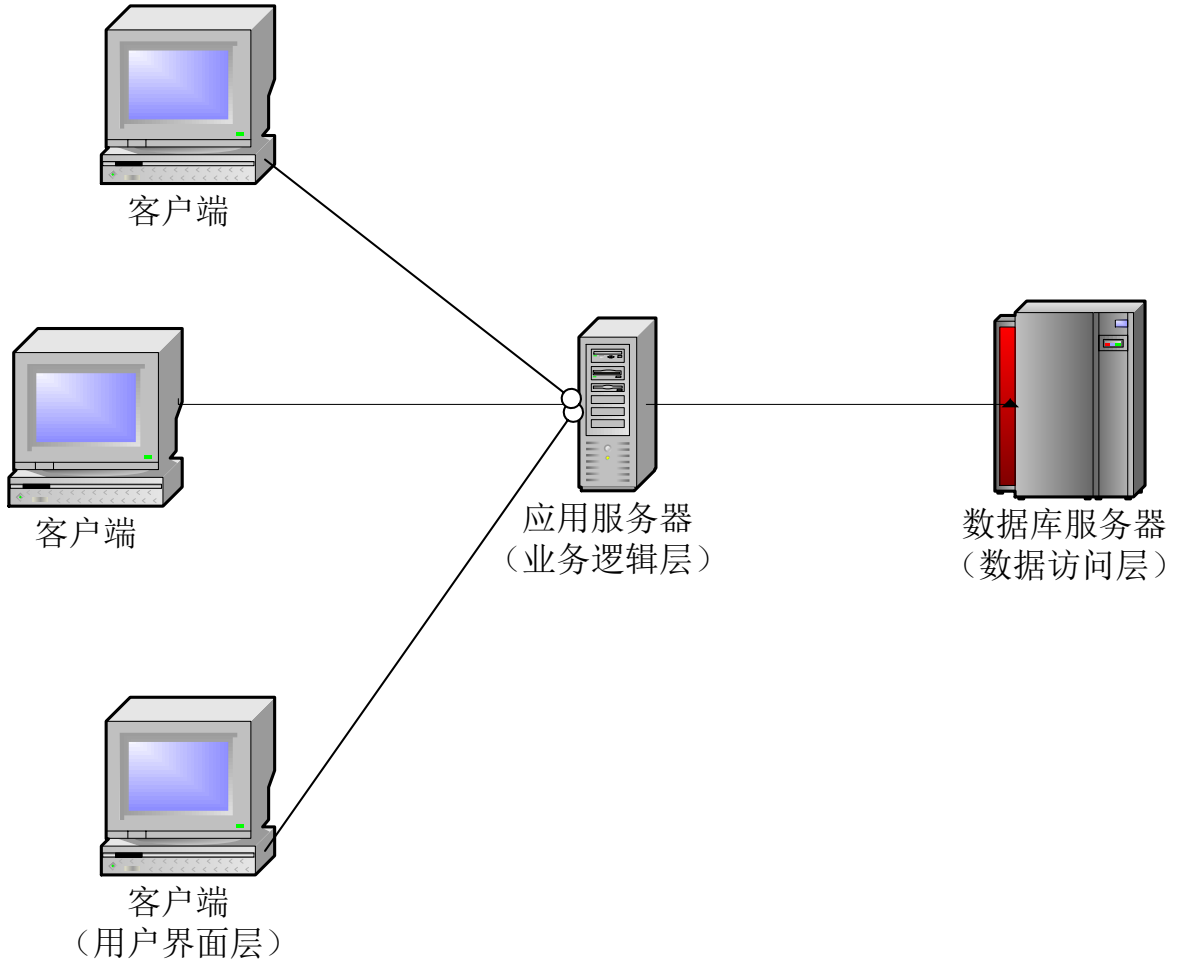
- 包是**UML**中的模型元素之一，可以包含其他的包和模型元素。包之间可以有关联，形成依赖关系。
- 包是一种分组机制，包中拥有或涉及的所有模型元素叫做包的内容。包的实例是没有意义的。
- 包仅有在建模的时候有用，而不需要转换成可执行的系统。
- 包常用来描述逻辑体系结构。

逻辑体系结构模型的作用

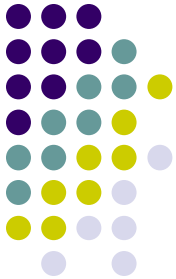
- 指出系统应该具有的功能
- 为完成这些功能要涉及哪些类，这些类之间如何联系
- 类和它们的对象如何协作才能实现这些功能
- 指明系统中各功能实现的顺序
- 根据逻辑体系结构模型，制定出相应的开发进度计划


通用接口界面
Windows窗体构件
用户窗口
<<使用>>
<<使用.>>
Crystal Reports构件
系统业务对象
业务对象管理
<<依赖>>
实际业务对象
系统数据库
持久对象及数据
<<使用>>
SQL查询语言

举例：三层通用逻辑体系结构



三层结构的数据库应用模式的优点



- 性能优势：应用服务器承担了客户端的连接功能，只需要通过一个或者少量的连接来访问数据库服务器。数据库服务器可以专门处理实际的数据库访问操作，只需要维护少量的客户端连接，大大提高了效率。另外，应用服务器可以对客户端任务进行分析，对于相同的数据库数据请求，可以提供同一个数据集数据，避免了多次访问数据库服务器。

- 减少数据库连接：可大大减少数据库服务器需要的客户端连接数目，减少投资。

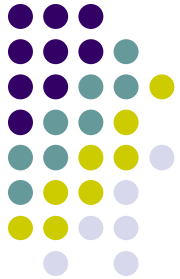
- 增强系统的可靠性：应用服务器处于数据库服务器和客户端之间，屏蔽了客户端和数据库服务器之间的直接连接。因此，当数据库服务器出现故障时，应用服务器可以自动连接后备数据库服务器，动态切换比较容易。应用服务器本身可以实现负载均衡的功能，将数据库访问请求分配给不同的数据库服务器，很容易提高系统的运行效率。由于应用服务器本身不维护数据库数据，因此当它出现故障时，很容易被替换成另外的应用服务器。

Continued…

- 客户端分发方便：三层结构中的客户端应用程序只包含用户界面程序和专门的三层数据库连接文件，由于不需要安装数据库访问引擎，可减少客户端安装程序的复杂度，便于客户端程序的分发。在业务逻辑更改的情况下不需要更改客户端程序，大大减少了客户端程序升级的次数。

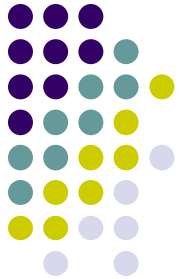
- 集中业务逻辑：应用服务器中可以集中放置一些通用的业务逻辑代码，这样更改业务逻辑代码时不影响客户端程序；和存储过程相比，可减少数据库服务器的负担；在其他项目开发时，可比较方便地重复利用业务逻辑代码。

物理体系结构（实现建模）



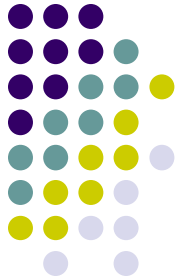
- 物理体系结构涉及到系统的详细描述（根据系统所包含的硬件和软件）。
- 物理体系结构显示了硬件的结构，包括了不同的节点和这些节点之间如何连接，还显示了代码模块的物理结构和依赖关系，并显示了对进程、程序、构件等软件在运行时的物理分配。
- **UML**提供两种物理体系结构描述图：构件图和配置图。

构件图和部署图



- 构件是逻辑体系结构中元素的物理实现。
- 构件图描述系统中不同物理构件及其相互之间的关系，表达系统代码本身的结构。
- 部署图（配置图）由节点构成，节点代表系统的硬件，构件在节点上驻留并执行。配置图描述系统软件构件与硬件之间的关系，它表达的是运行时的系统结构。

物理体系结构模型的作用



- 指明系统中的类和对象所涉及的具体程序或进程。
- 这些程序和进程的执行依赖具体计算机。
- 指明系统中配置的计算机和其他硬件设备。
- 指明系统中各种计算机和硬件设备如何进行连接。
- 明确不同的代码文件之间相互的依赖关系。
- 如果修改某个代码文件，表明哪些相关（与之有依赖关系）的代码文件需要重新进行编译。

构件与构件图

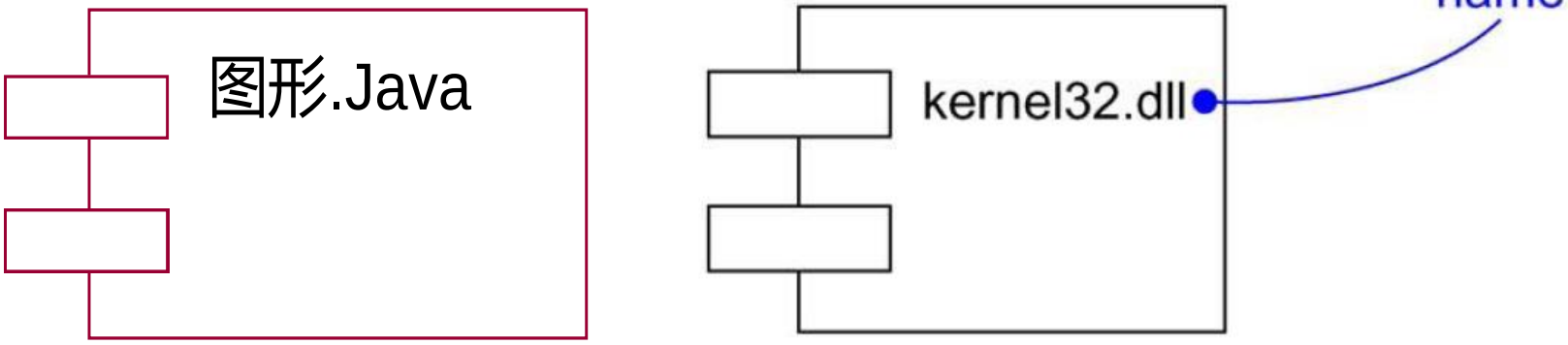
构件 **Component**

- 构件是系统中遵从一组接口并提供实现的一个物理的、可替换的单元。构件是软件复用的基本物理实现单元，是逻辑模型元素如类、接口、协同等的物理包。

- **UML**中，对象库、可执行体、**com+**构件和企业级**java beans**都可以描述成构件。

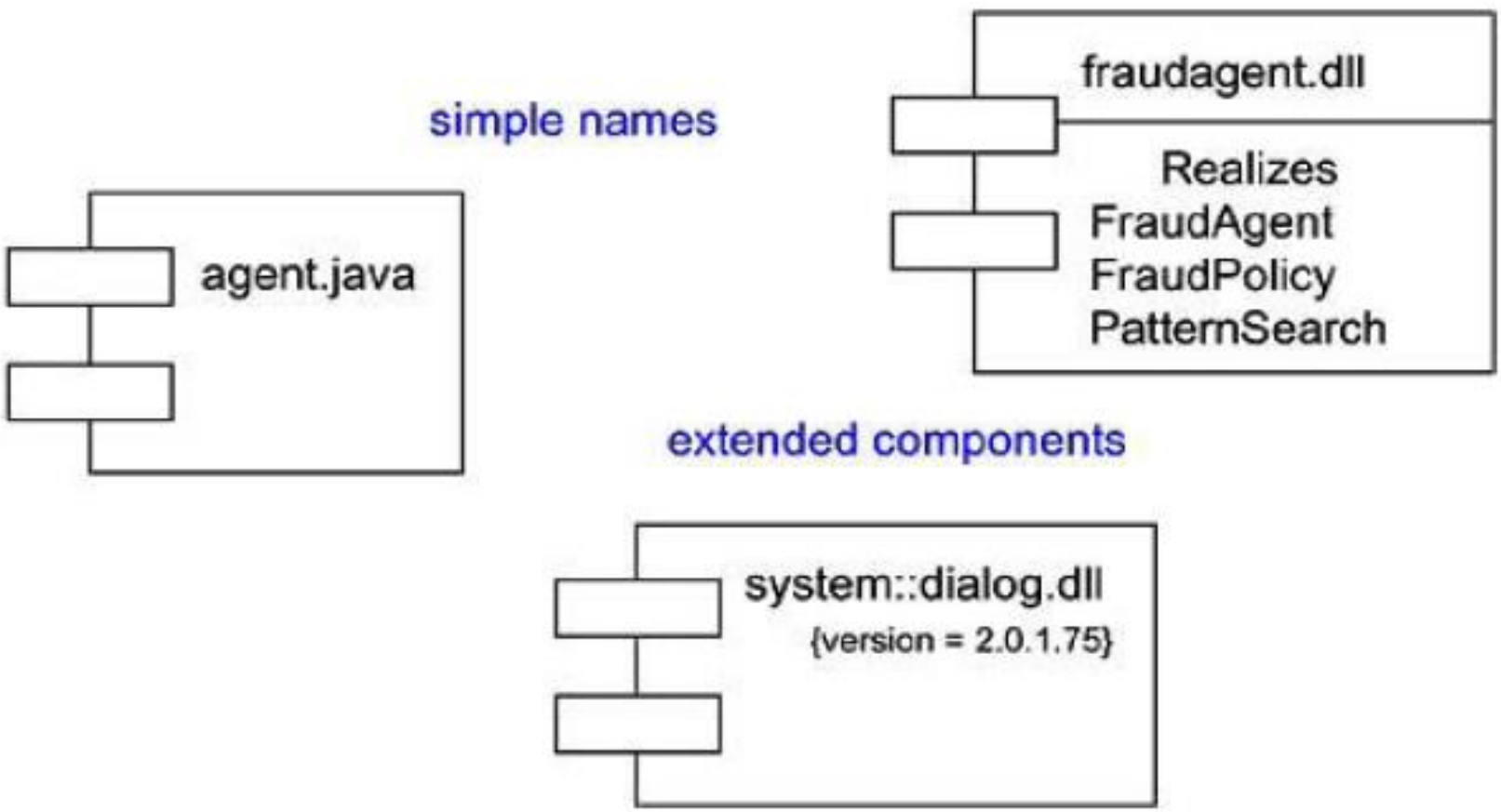
- 构件内部模型元素所实现的服务，通过其提供的一组接口来实现。

构件的图符表示



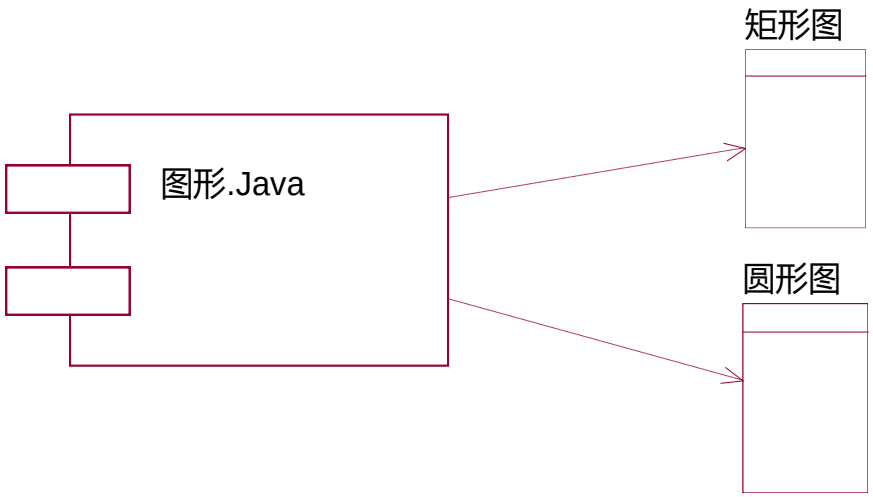
- 构件中可以包含类；类可以有实例，同样，构件也可以有实例。

more

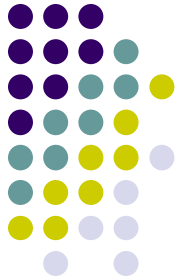

simple names
agent.java
fraudagent.dll
Realizes
FraudAgent
FraudPolicy
PatternSearch
extended components
system::dialog.dll
{version = 2.0.1.75}

构件与类

- 构件是一组逻辑元素的物理实现。构件包含类，类通过构件来实现，构件与类之间是依赖关系。

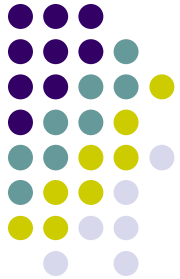


构件与类的相同点



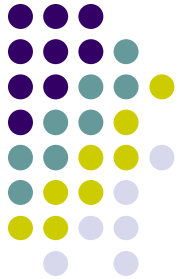
- 二者都有名称（名词/名词词组）
- 都可以实现一组接口
- 都可以参与依赖、继承、关联等关系和交互
- 都可以被嵌套
- 都可以有实例

构件与类的显著不同点



- 抽象的方式不同：
  - 构件：是程序代码的物理抽象；构件可以驻留在节点上。
  - 类：是逻辑抽象；不能单独存在于节点上。
- 抽象的级别不同
  - 构件：表示一个物理模块，构件可以包含多个类；构件依赖它所包含的类
  - 类：表示一个逻辑模块，只能从属于某个构件，类通过构件来实现。
- 访问方式不同
  - 构件：不直接拥有属性和操作，只能通过接口访问其操作
  - 类：直接拥有自己的属性和操作，可以直接访问其操作
- 与包的关系不同
  - 构件：包可以包含成组的逻辑模型元素，也可以包含物理的构件
  - 类：一个类可以出现在多个构件中，却只能在一个包内定义。

软件构件的特点



- 接口：用来描述一个构件所能提供的服务的操作的集合。
- 操作：构件通过消息传递方式进行操作，每个操作由输入输出变量、前置条件和后置条件来决定。
- 实例化：构件的实例化代表运行期间的可执行软件模块。
- 与配置环境的亲和性
- 能与同环境下其他构件进行交互
- 构件可以是可执行代码、二进制代码和源代码形式。
- 可替换的物理实体
- 系统的组成部分：构件可以在多个系统中复用

构件的种类

- **UML**中，软件构件分为源代码构件、二进制代码构件和可执行程序构件，等等。

1. 源代码构件

   - **<<file>>**：表示包含源代码或数据的文件
   - **<<page>>**：表示**web**页
   - **<<document>>:** 表示文档（包含文档，而不是可编译代码）

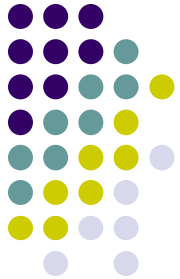
2. 二进制代码构件

- 二进制代码构件也称链接时构件，是源代码构件经编译后产生的目标代码文件，或者是编译一个或者多个源代码构件而产生的静态库文件或动态库文件。

- 目标代码文件和静态库文件是在运行前链接成可执行程序构件；而动态库文件也叫动态链接库（**DLL**），是运行时才链接成可执行程序构件。

- **<<library>>:**用来指出构件是静态库或动态库。

3. 可执行程序构件



- 可执行程序构件也称运行时构件，是系统执行时使用的构件（可执行程序构件）。一个可执行程序构件表示处理机上运行的一个可执行单元，如由**DLL**实例化形成的**COM**＋对象。

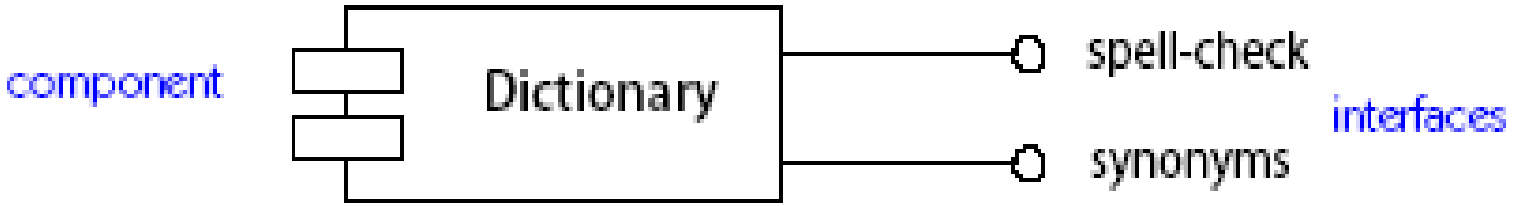
- **<<application>>:**用来表示一个可执行程序

- **<<table>>：**表示一个数据库表

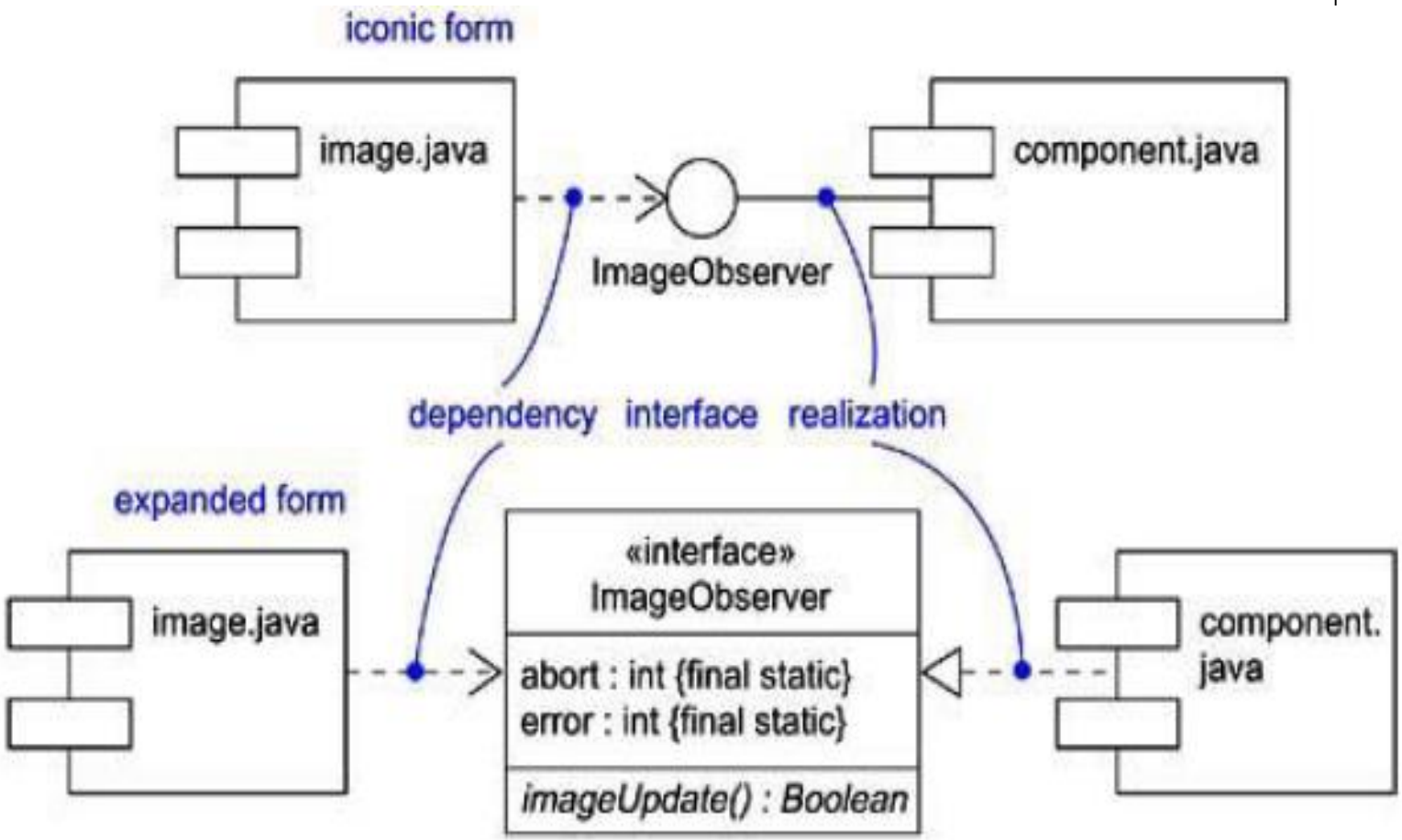
构件的接口

- 一个构件可以定义对其他构件来说是可见的接口。接口可以是源代码级定义的接口，也可以是运行时使用的二进制接口（如**OLE**）。

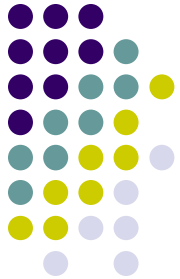
- 构件有两种接口：
  - 输出接口：构件实现接口，即构件被使用的接口。一个构件可以有多个输出接口。
  - 输入接口：构件使用的接口，即使用其他构件的接口。构件必须遵从该接口并以此为基础来构造构件。一个构件可以遵从多个输入接口。



More


iconic form
image.java
component.java
ImageObserver
dependency
interface
realization
expanded form
image.java
«interface»
ImageObserver
abort : int {final static}
error : int {final static}
imageUpdate() : Boolean
component.
java

构件图

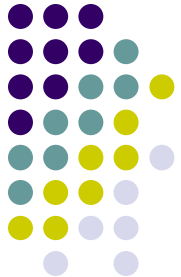


- 构件图主要用于建立系统的静态实现视图模型，通过构件之间的依赖关系描述系统软件的组织结构。构件图中的构件没有实例，只有部署图中才能标识构件的实例。

- **UML**中，构件图用于建立以下模型：
  - 系统业务模型
  - 系统开发管理模型
  - 系统实现模型
  - 系统物理配置模型
  - 集成系统模型

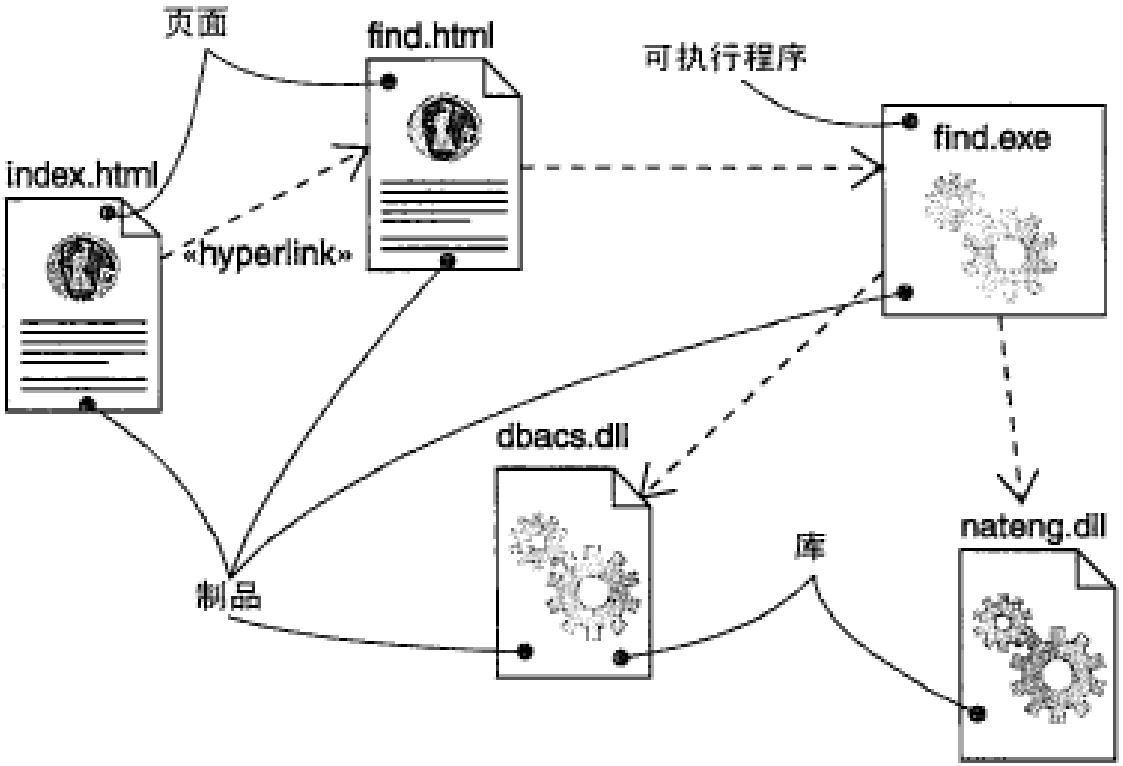
- Modeling source code
- Modeling executable releases
- Modeling physical databases
- Modeling adaptable systems

制品（Artifact）

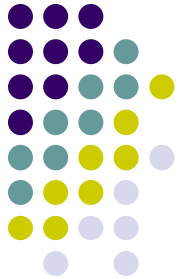


- 制品存在于比特的物质世界中。是物理建模时重要的构造块。
- 制品包括：可执行程序、库、表、文件和文档。

制品图


页面
find.html
可执行程序
index.html
find.exe
«hyperlink»
dbacs.dll
制品
库
nateng.dll

对源代码建模



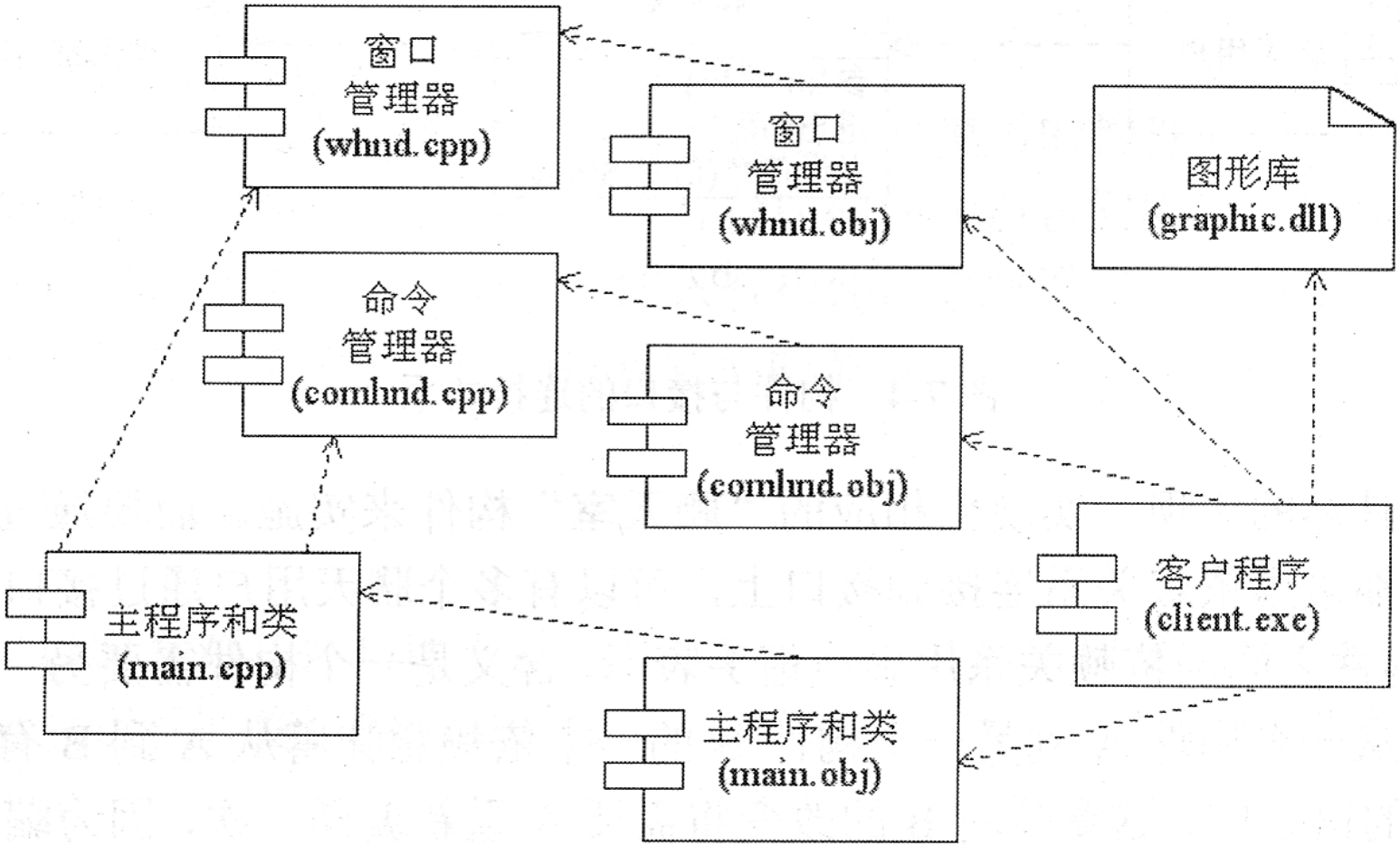
根据一个客户的软件系统从C++源代码构件编译成相应的二进制代码（目标码）构件，最后通过连接成为可执行程序的过程，尝试画出一个构件图，表达源代码和可执行代码之间的实现模型：

1. 有三个源代码构件“main.cpp”、“comhnd.cpp”和“whnd.cpp”共同构成一个完整的客户软件源代码程序。这三个构件中，“main.cpp”构件需依赖（即调用）“comhnd.cpp”构件和“whnd.cpp”构件才能达到系统要求的完整功能。

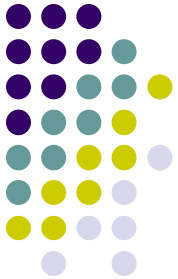
2. 这三个源代码构件分别经过编译，产生相应的三个二进制代码构件：“main.obj”构件、“comhnd.obj”构件和“whnd.obj”构件，这三个二进制代码构件依赖于对应的三个源代码构件才能产生。

3. 三个二进制代码构件经过连接，形成一个可执行构件“client.exe”。可执行构件“client.exe”的产生依赖于对应的三个二进制代码构件；同时，它在执行过程中还要调用一个动态链接库“graphic.dll”才能完成系统要求的功能。（动态链接库可用注释节点表示。依赖关系也用虚线箭头表示）

构件图


窗口
管理器
(whmd.cpp)
窗口
管理器
(whmd.obj)
图形库
(graphic.dll)
命令
管理器
(comlmd.cpp)
命令
管理器
(comlmd.obj)
客户程序
(client.exe)
主程序和类
(main.cpp)
主程序和类
(main.obj)

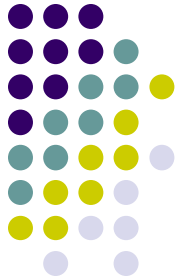
注意事项



- 每一个构件图只是系统静态视图的某一个图形表示，描述系统的一个侧面，因而不能面面俱到。

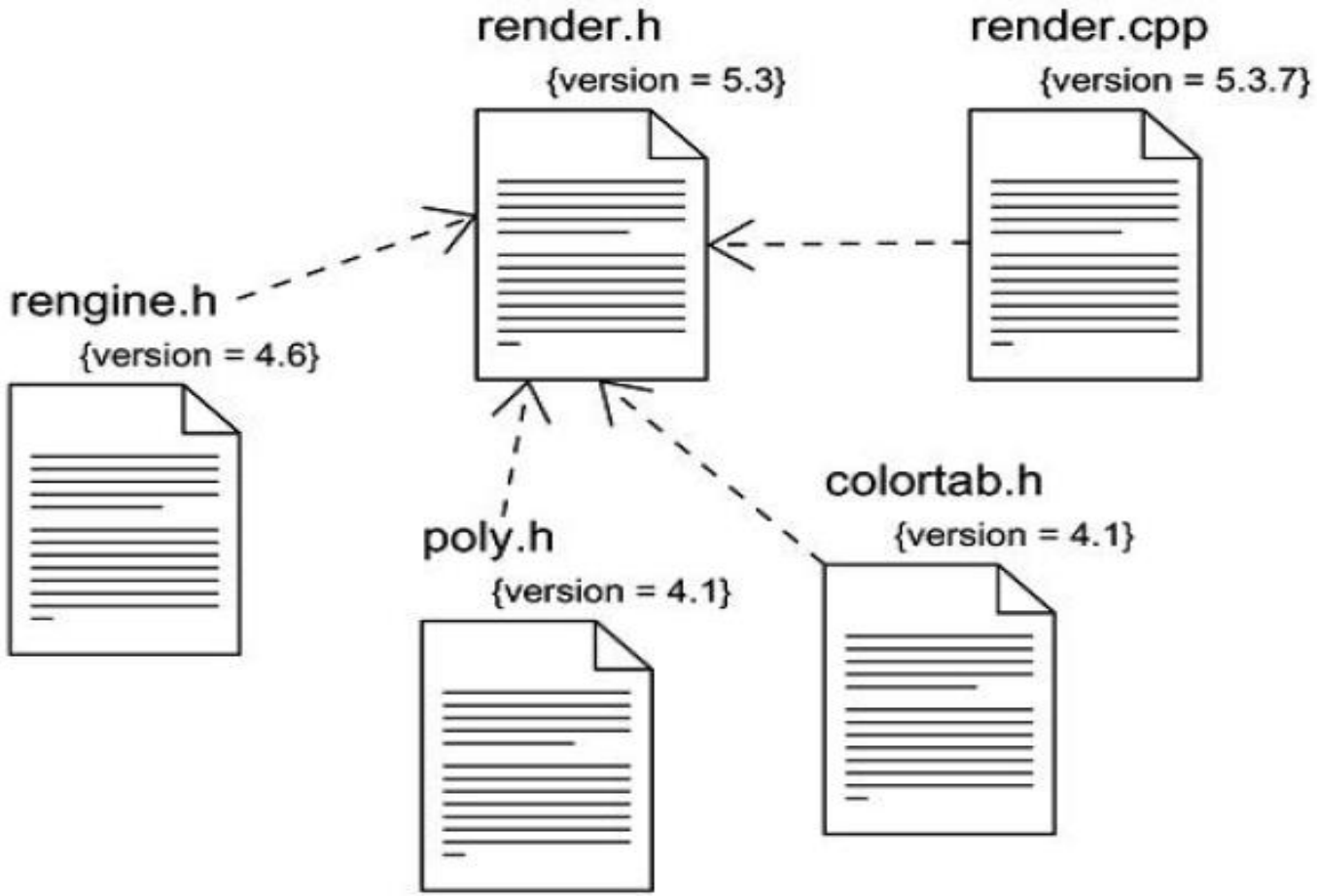
- 一个结构良好的构件应具备以下特点：
  - 从物理结构上对软件系统进行抽象；
  - 提供一组小的、定义完整的接口实现；
  - 构件应包含与其功能有关的一组类，以便满足接口要求
  - 与其他构件相对独立，构件之间一般只有依赖和实现的关系

绘制一个构件时掌握的技巧

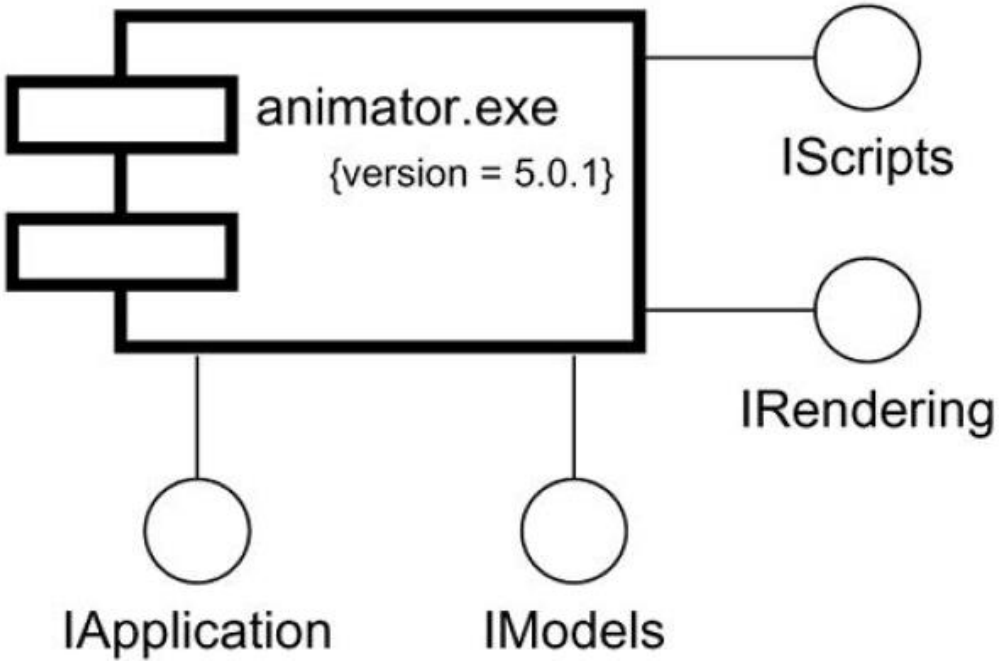


- 为构件标识一个准确表达其意义的名字
- 接口一般采用短式图符表示
- 只有必须显示接口的操作时采用长式表示
- 只显示那些对理解构件功能有重要影响的接口
- 构件为源代码或库时，注意显示有关版本标记

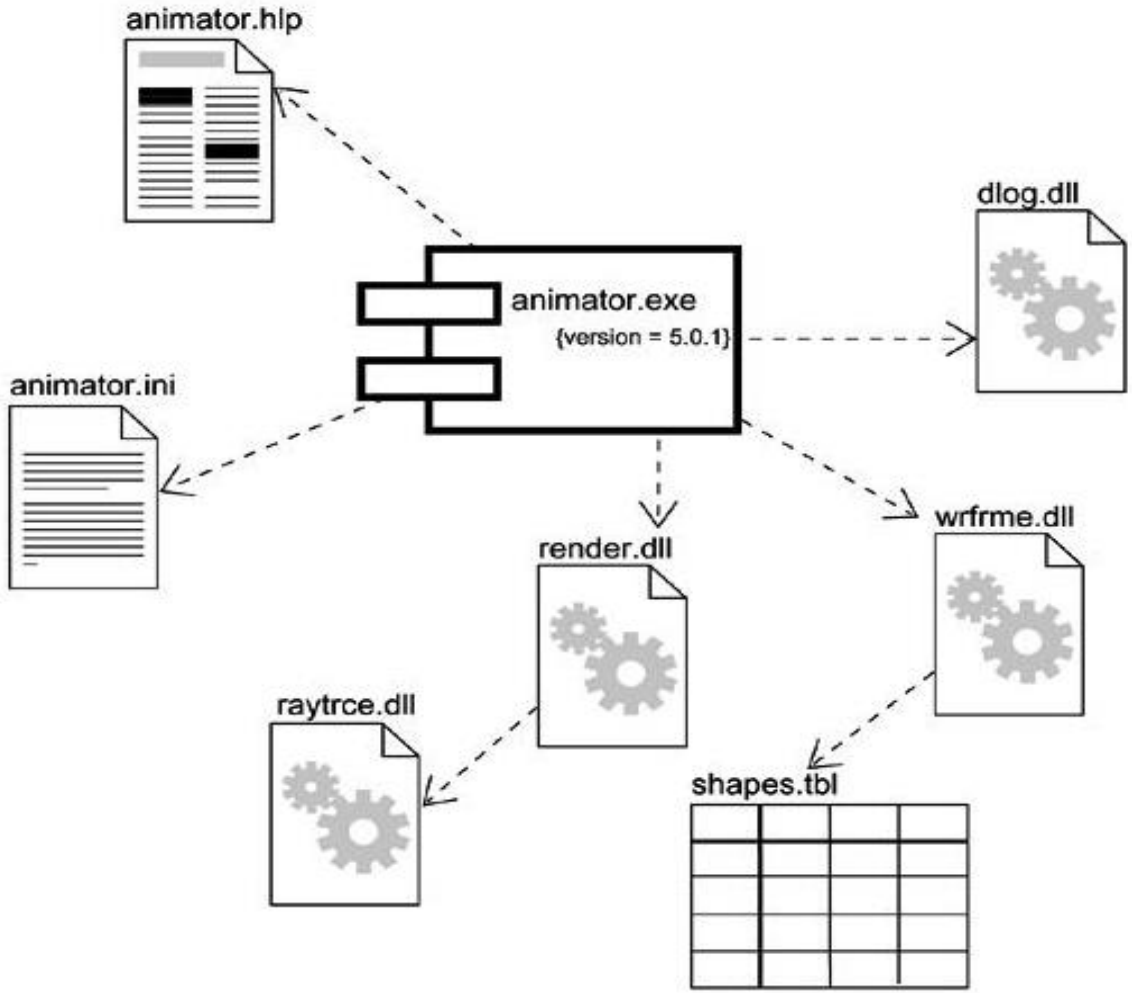
Example（制品图）


render.h
{version = 5.3}
render.cpp
{version = 5.3.7}
rengine.h
{version = 4.6}
poly.h
{version = 4.1}
colortab.h
{version = 4.1}

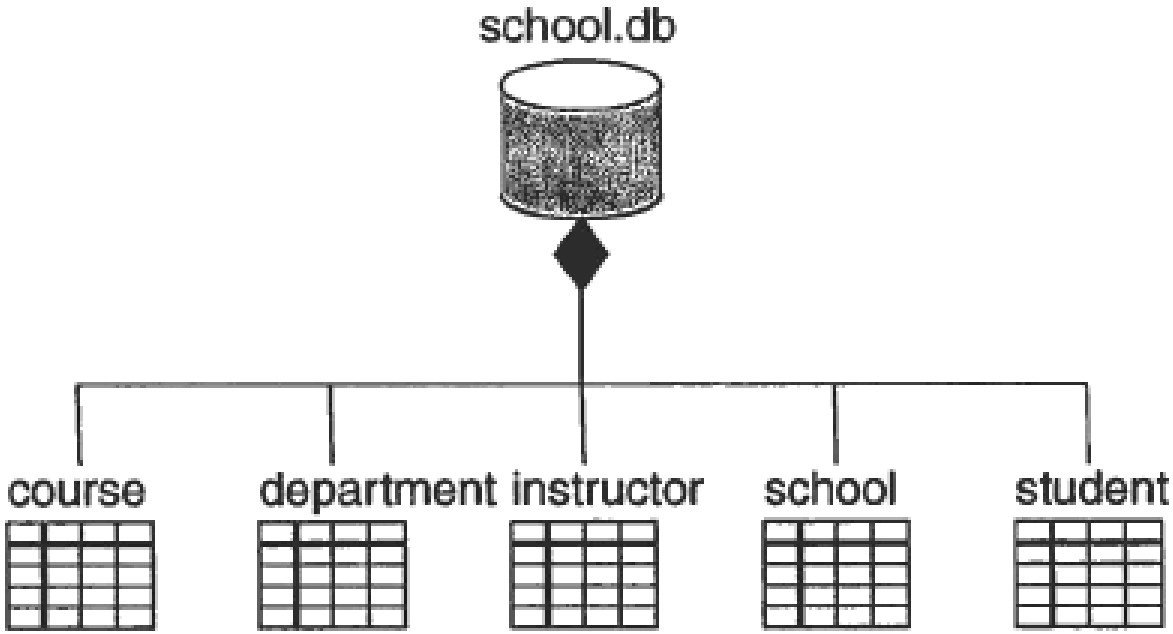
example


animator.exe
{version = 5.0.1}
IScripts
IRendering
IApplication
IModels

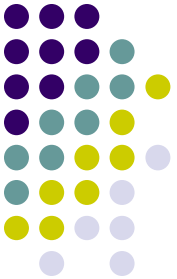
example


animator.hlp
dlog.dll
animator.exe
{version = 5.0.1}
animator.ini
render.dll
wrfrme.dll
raytrce.dll
shapes.tbl

对物理数据库建模



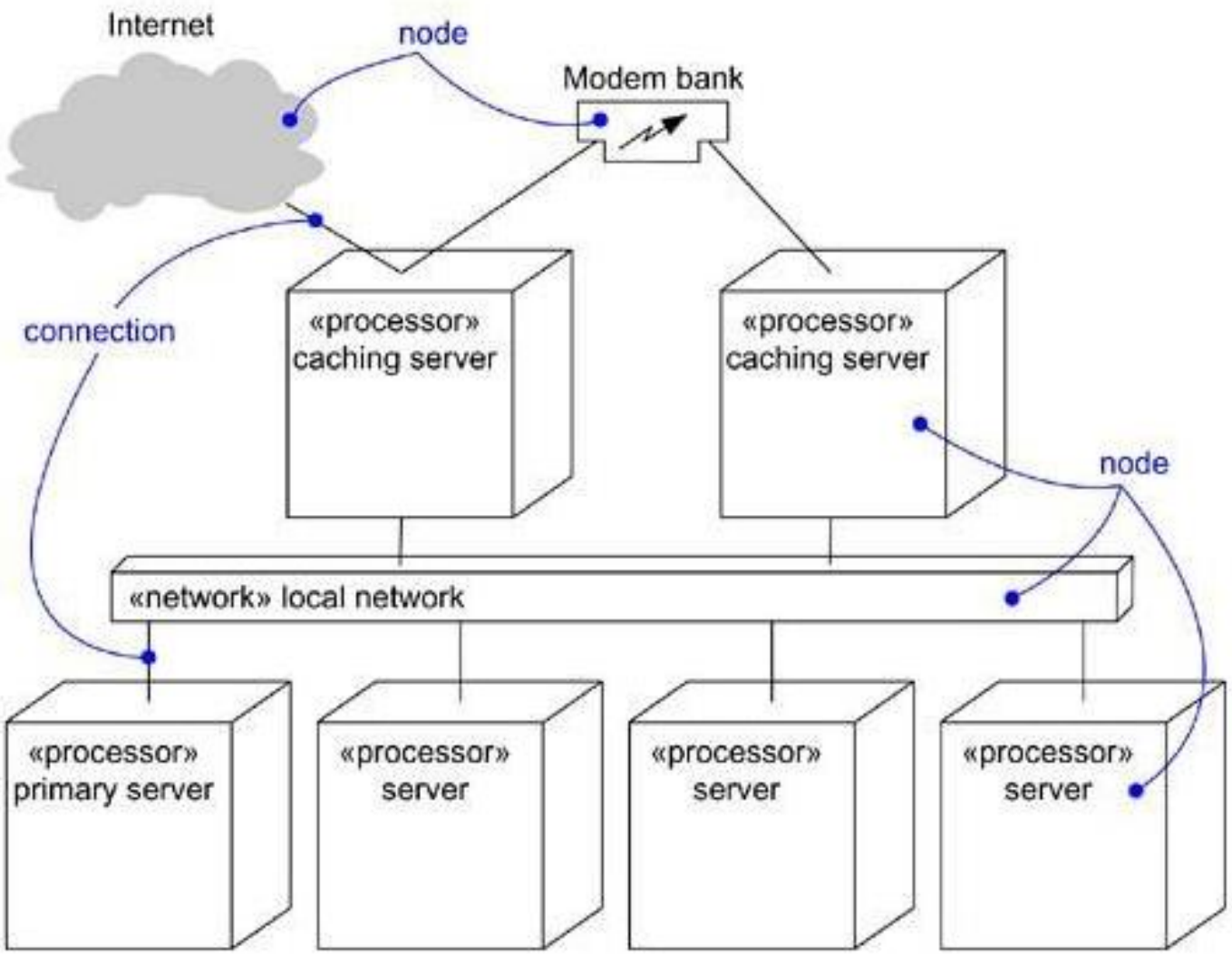
部署图



- 部署图由节点和节点之间的联系组成，描述了处理器、设备和软件构件运行时的体系结构。在这个体系结构上可以看到某个节点上在执行哪个构件，在构件中实现了哪些逻辑元素（类、对象、协作等），完成了哪些功能，最终可以从这些元素追踪到系统需求分析（用例图）。

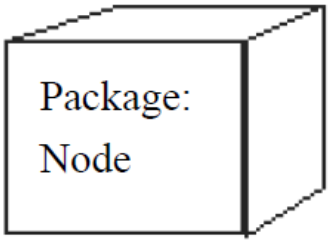
- 部署图的基本元素有节点、连接、构件、对象、依赖等。

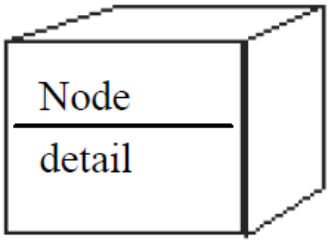
example


Internet
node
Modem bank
connection
«processor»
caching server
«processor»
caching server
node
«network» local network
«processor»
primary server
«processor»
server
«processor»
server
«processor»
server

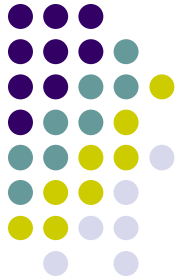
节点（node）

- 节点是某种计算资源的物理对象，包括计算机、外部设备等。
- 节点既可以看作类型，也可以看作实例。当节点被看作实例时，节点名应有下划线。
- 节点用三维立方体表示。
- 一个节点必须有一个名字，节点名是一个标识符，写在立方体中间。节点名也可以用长式、短式表示。可以附加如**<<printer>><<router>><<carcontroller>>**等符号表示特定的设备类型。





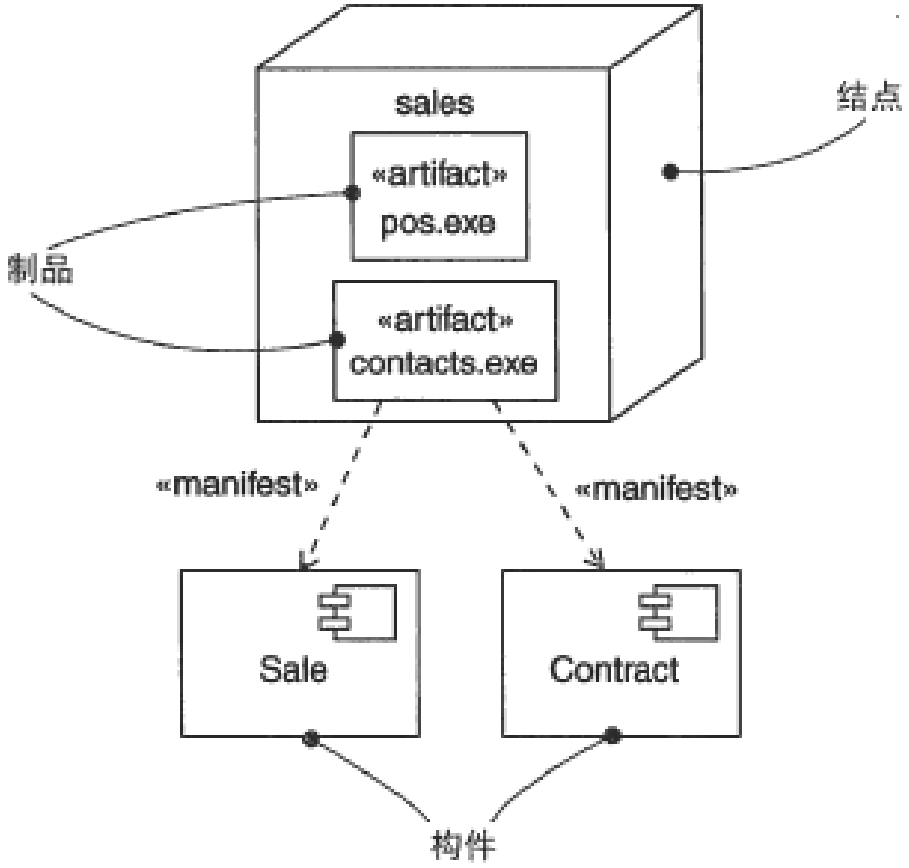
节点与构件



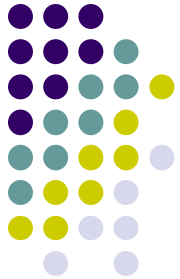
- 可执行程序构件实例可以包含在节点实例符号终，表示它们在该节点实例上驻留并执行。

- 注意，某些节点不支持某种构件

节点和“制品”（“构件”）


sales
结点
«artifact»
pos.exe
制品
«artifact»
contacts.exe
«manifest»
«manifest»
Sale
Contract
构件

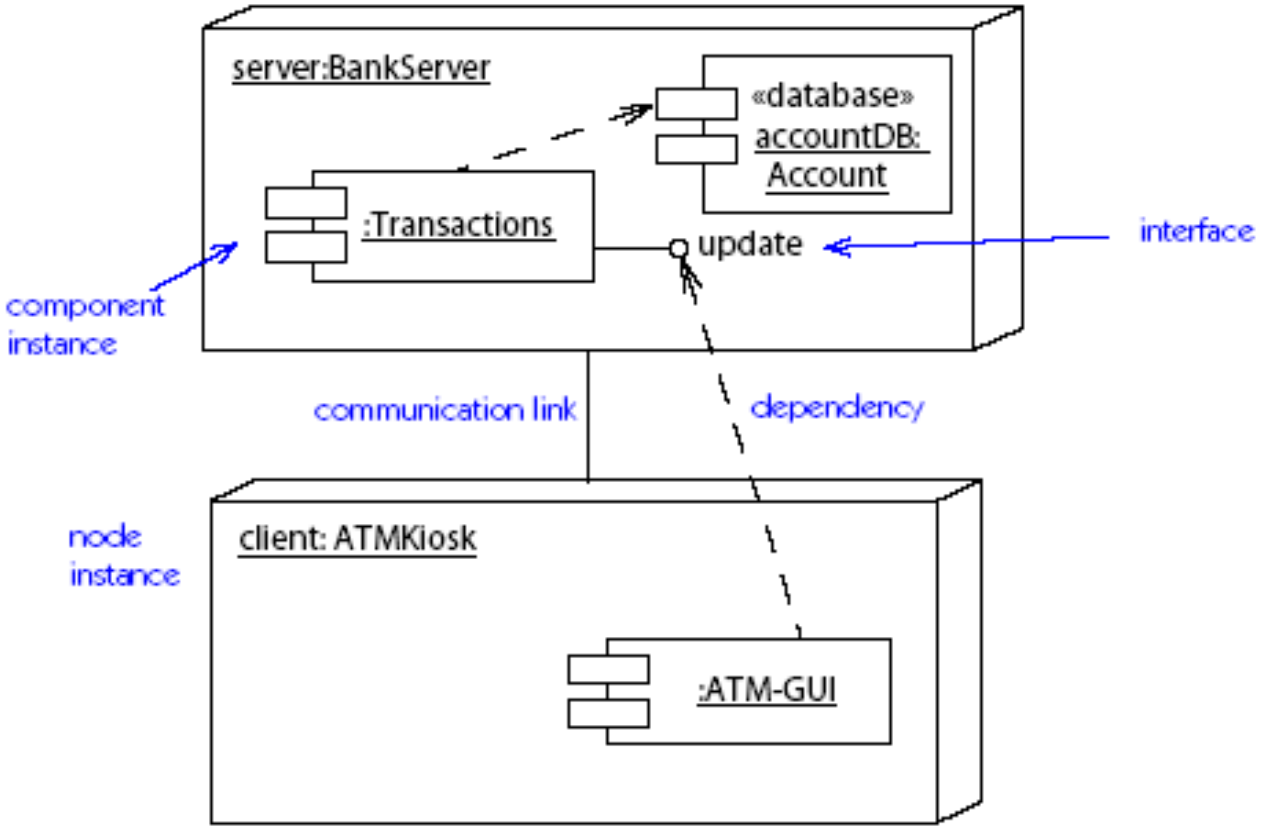
节点与对象



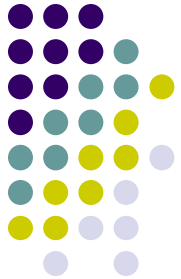
- 一个对象也可以画在节点实例中，表示它驻留在该节点上。
- 构件和对象的实例可以驻留在节点上，还可以从一个节点向另一个节点迁移，节点执行构件。
- 分布式系统中，一个对象可以在系统的生存期中在不同的节点之间传送。

节点之间的联系

- 通信关联
- 依赖关系



创建部署图的步骤



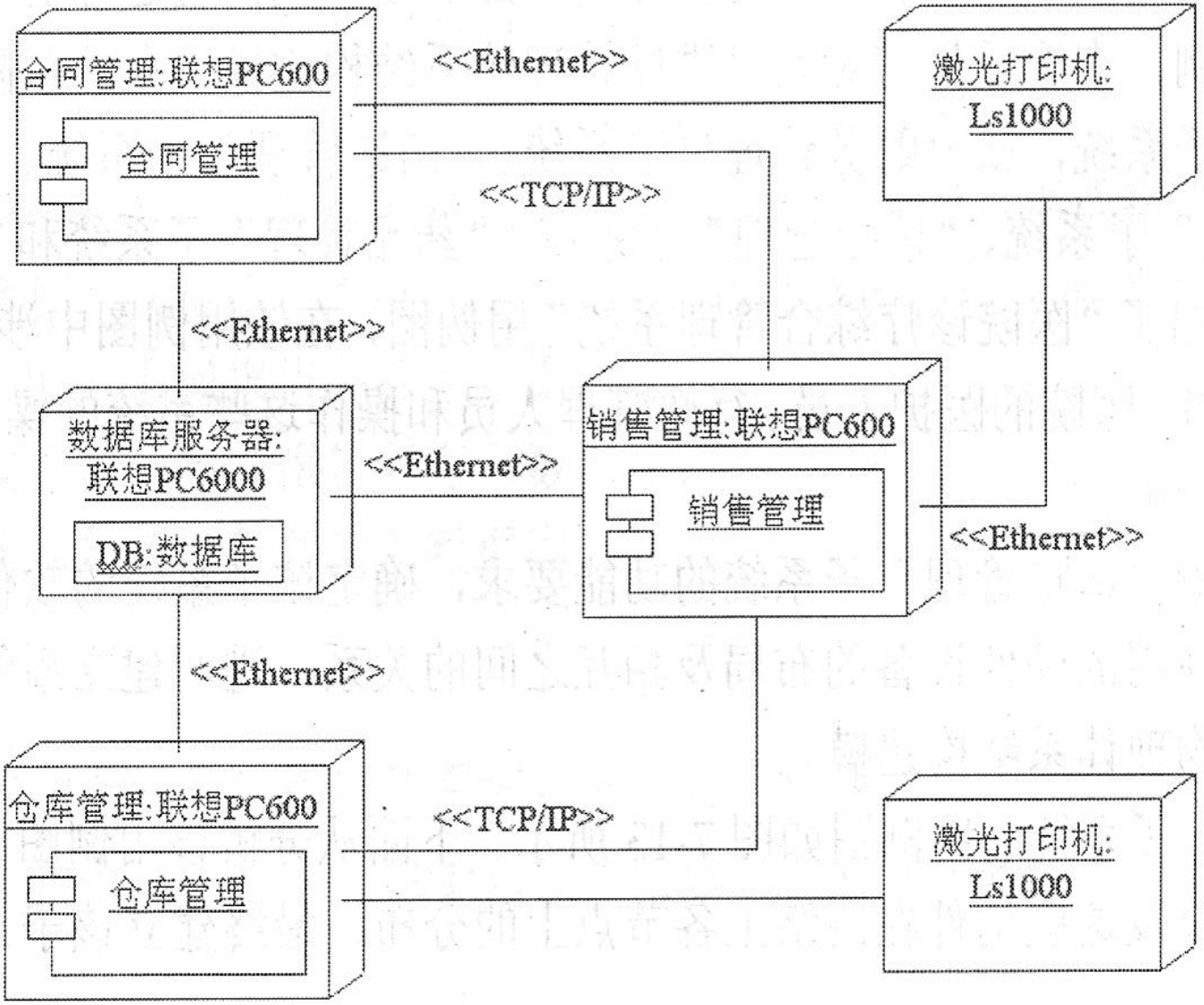
- 确定节点
- 描述节点属性
- 确定驻留构件
- 确定联系（依赖）
- 绘制部署图（精化和细化）

example

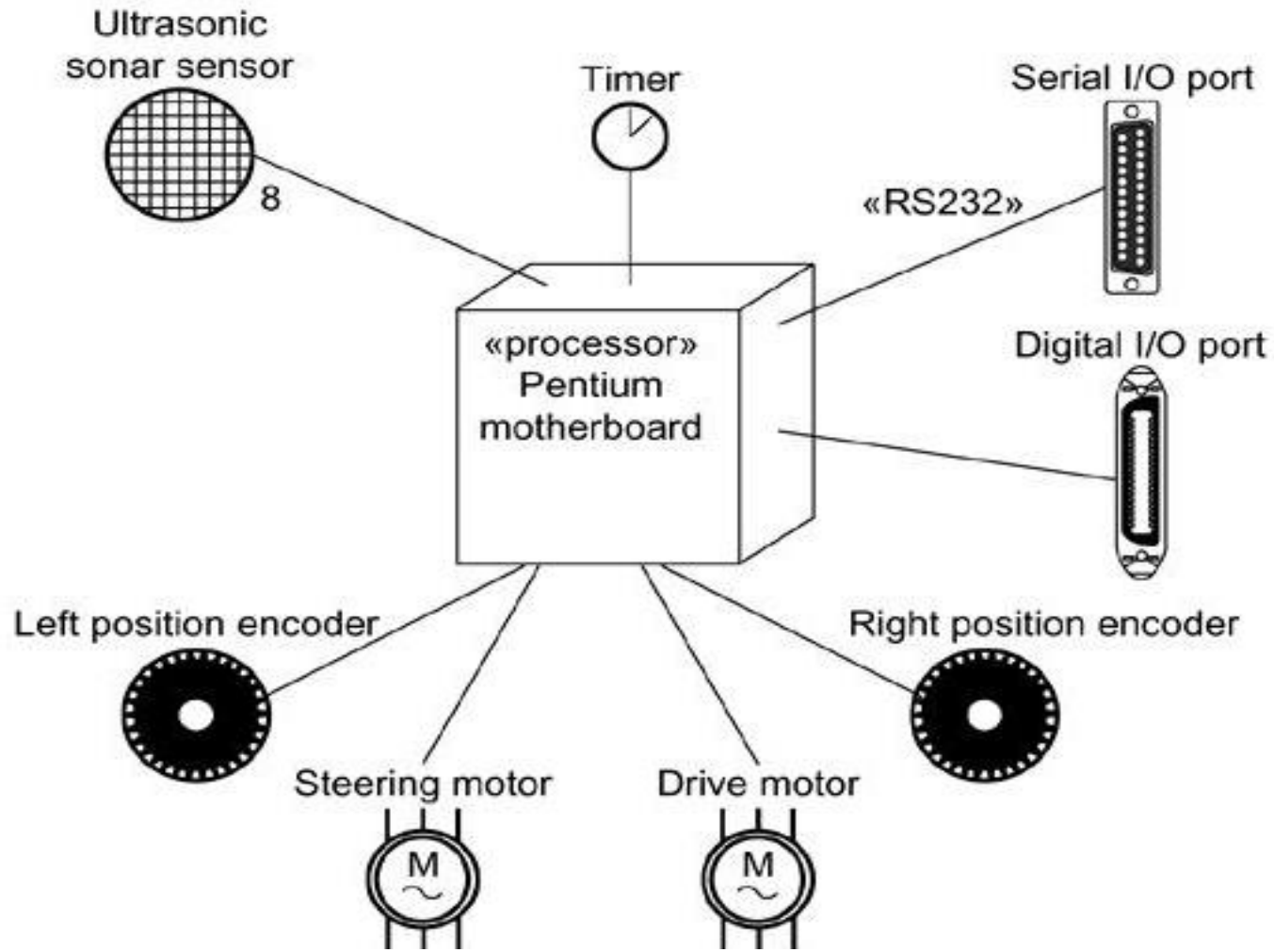
根据以下描述，试完成一个“销售管理系统”的部署图：

该系统采用的硬件设备有：

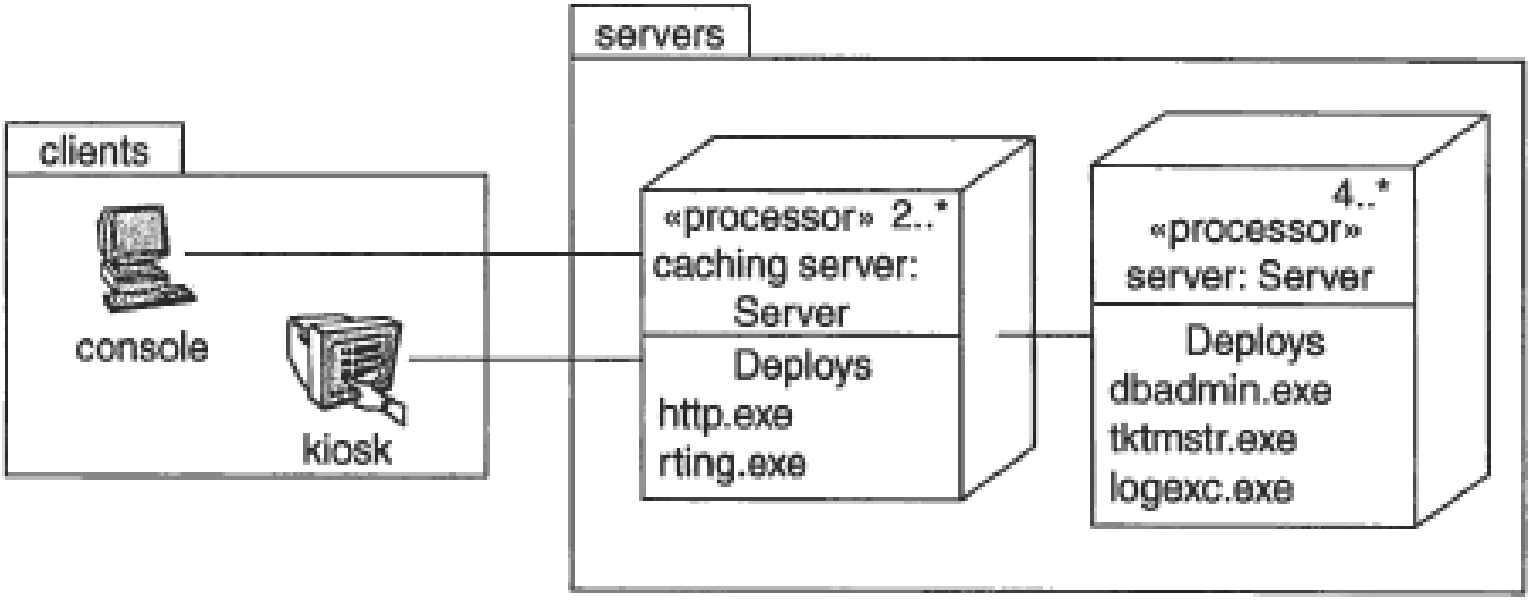
- **3台联想PC600作为终端机，分别用作销售管理、仓库管理和合同管理。**
- **1台联想PC6000作为网络数据库服务器。**
- **2台Ls1000打印机。一台为网络共享、一台为仓库管理专用。**
- **客户机之间采用《TCP/IP》通信协议，3个客户机与服务器之间采用《Ethernet》网连接。**


合同管理:联想PC600
合同管理
<<Ethernet>>
激光打印机:
Ls1000
<<TCP/IP>>
<<Ethernet>>
数据库服务器:
联想PC6000
DB:数据库
<<Ethernet>>
销售管理:联想PC600
销售管理
<<Ethernet>>
<<Ethernet>>
仓库管理:联想PC600
仓库管理
<<TCP/IP>>
激光打印机:
Ls1000

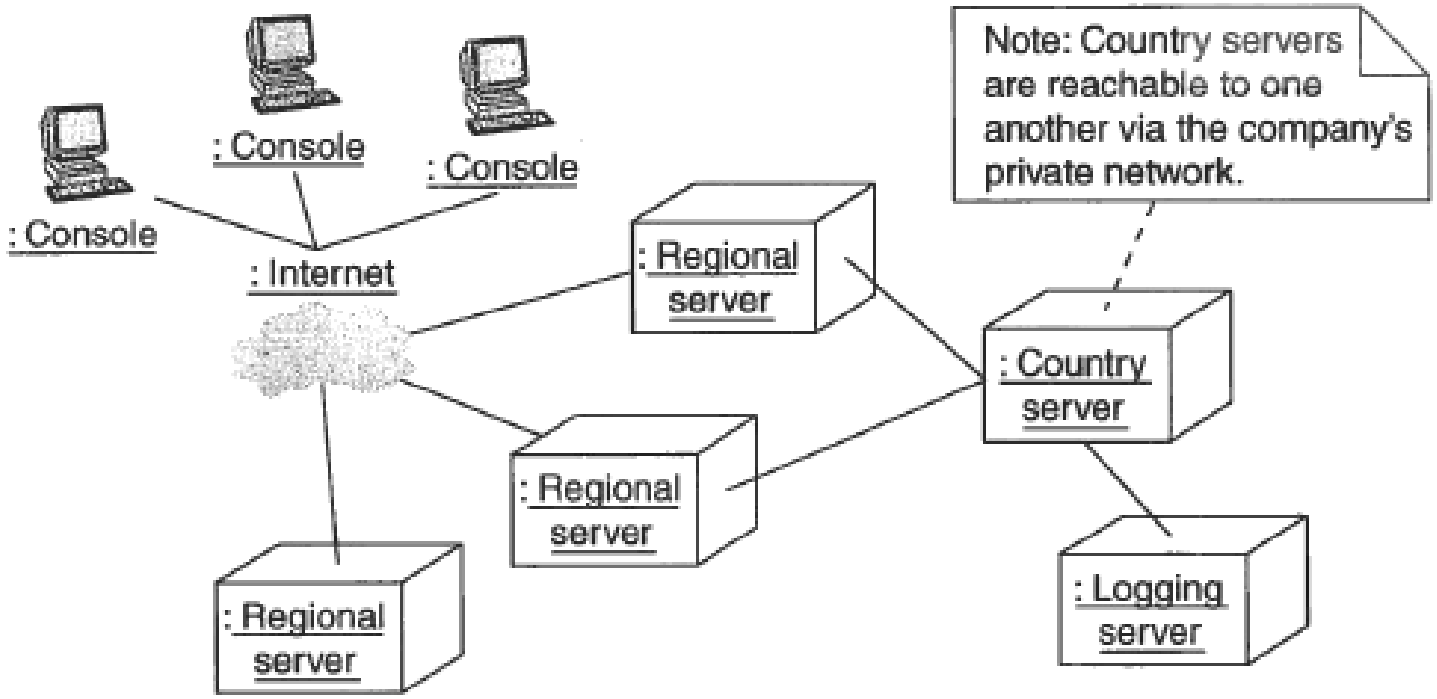
More（对嵌入式建模）


Ultrasonic
sonar sensor
8
Timer
Serial I/O port
«RS232»
«processor»
Pentium
motherboard
Digital I/O port
Left position encoder
Right position encoder
Steering motor
Drive motor
M
M

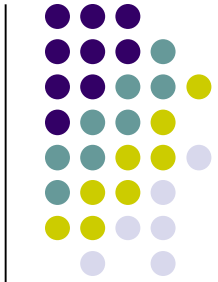
More（对客户/服务器系统建模）



More（对全分布式系统建模）



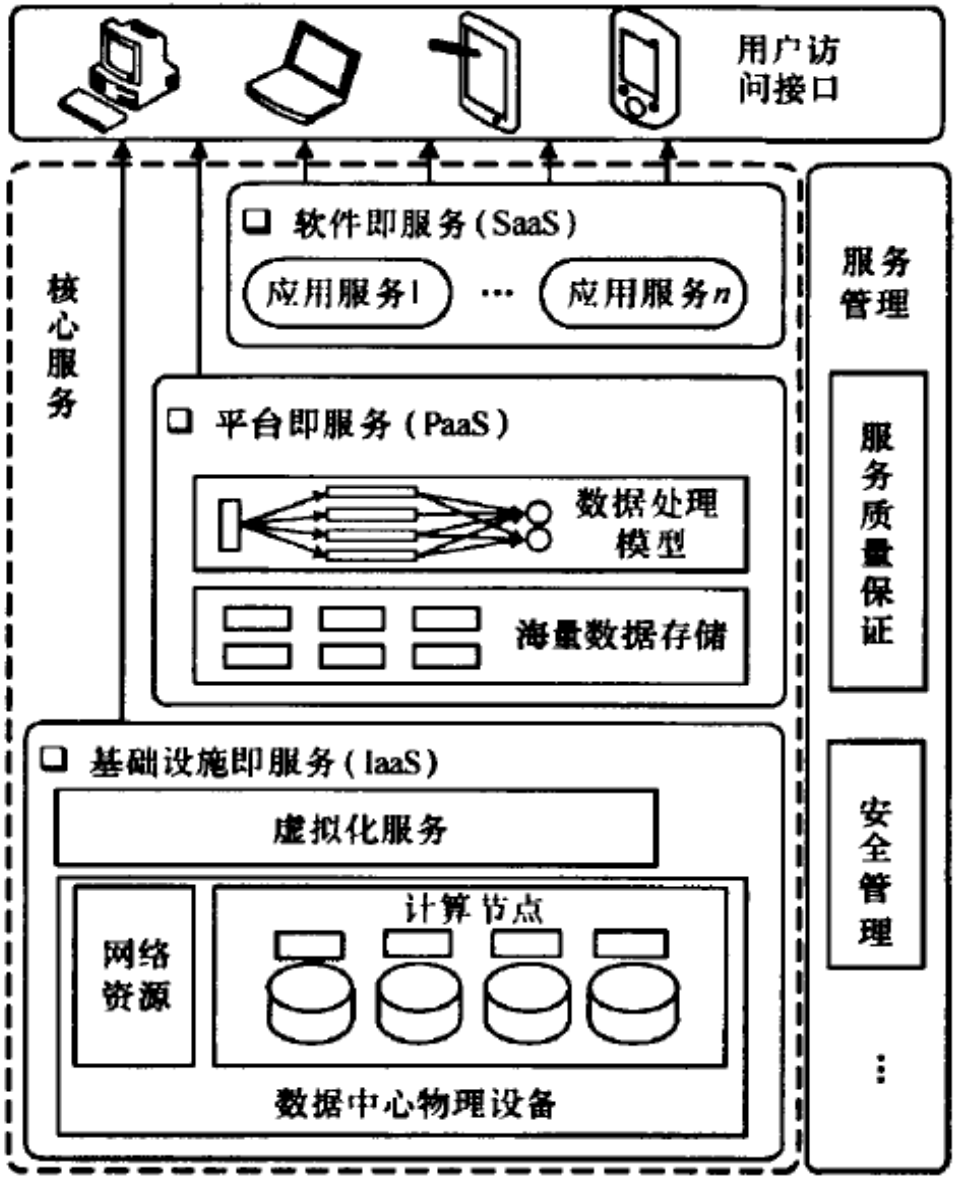
基于云计算的软件体系结构



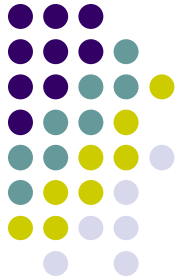
- “云计算”的构想最早在2006年由Google、Amazon等公司提出。
- 它被定义为一种利用互联网实现随时随地、按需、便捷地访问共享资源池(如计算设施、存储设备、应用程序等)的计算模式。
- 由集群计算、效用计算、网格计算、服务计算等技术发展而来，是分布式计算、互联网技术、大规模资源管理等技术的融合与发展。
- 云计算具有弹性服务、资源池化、按需服务、服务可计费和泛在接入等特性，使得用户只需连上互联网就可以源源不断地使用计算机资源，实现了“互联网即计算机”的构想。
- 基于云计算的软件已成为工业界发展的最新潮流。

- 基于云计算的体系结构的最大特点是计算机资源服务化，对用户来说，数据中心管理、大规模数据处理、应用程序部署等底层问题被完全屏蔽。

- 该体系结构具体可分为核心服务、服务管理和用户访问接口3层



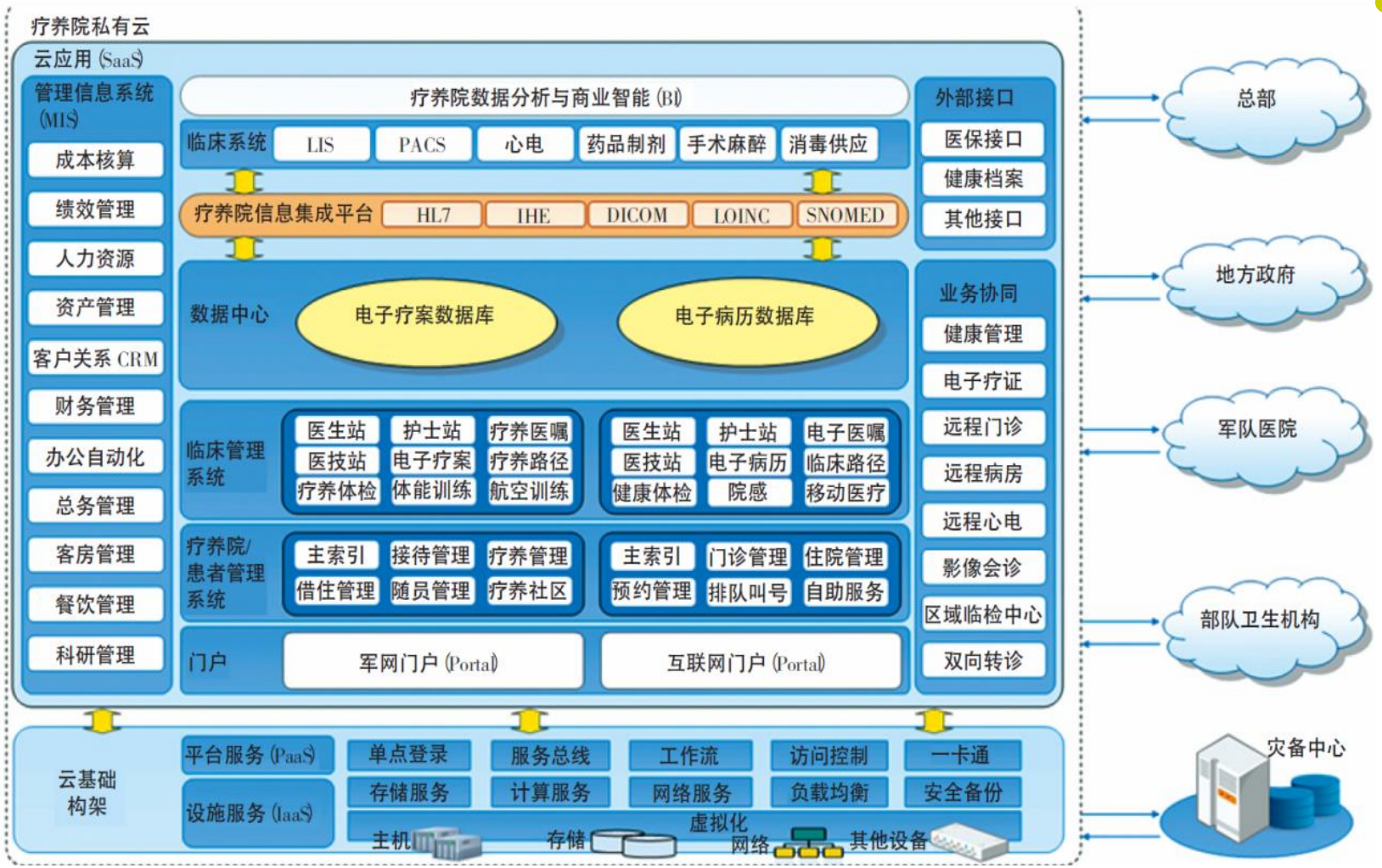
云计算体系结构中的核心服务层



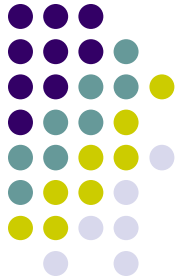
- 基础设施即服务层(IaaS, infrastructure as a service)
- 平台即服务层(PaaS，platform as a service)
- 软件即服务层(SaaS, software as a service)

- 在应用云计算体系结构的时候，软件设计师的主要精力可以集中在设计具体的SaaS中的应用，而把云计算平台的框架交给云计算服务商提供，对于设计师来说达到了透明化。

example


疗养院私有云
云应用 (SaaS)
管理信息系统 (MIS)
成本核算
绩效管理
人力资源
资产管理
客户关系 CRM
财务管理
办公自动化
总务管理
客房管理
餐饮管理
科研管理
疗养院数据分析与商业智能 (BI)
临床系统
LIS
PACS
心电
药品制剂
手术麻醉
消毒供应
疗养院信息集成平台
HL7
IHE
DICOM
LOINC
SNOMED
数据中心
电子疗案数据库
电子病历数据库
临床管理系统
医生站
护士站
疗养医嘱
医技站
电子疗案
疗养路径
疗养体检
体能训练
航空训练
医生站
护士站
电子医嘱
医技站
电子病历
临床路径
健康体检
院感
移动医疗
疗养院/患者管理系统
主索引
接待管理
疗养管理
借住管理
随员管理
疗养社区
主索引
门诊管理
住院管理
预约管理
排队叫号
自助服务
门户
军网门户 (Portal)
互联网门户 (Portal)
外部接口
医保接口
健康档案
其他接口
业务协同
健康管理
电子疗证
远程门诊
远程病房
远程心电
影像会诊
区域临检中心
双向转诊
云基础构架
平台服务 (PaaS)
设施服务 (IaaS)
单点登录
服务总线
工作流
访问控制
一卡通
存储服务
计算服务
网络服务
负载均衡
安全备份
主机
存储
虚拟化网络
其他设备
总部
地方政府
军队医院
部队卫生机构
灾备中心

review



- 什么是节点？其表示的意义是什么
- 节点有实例吗？在节点上可以驻留哪些模型元素？
- 通过节点之间不同的连接，部署图可以描述系统的哪几种建模模式？
- 什么是构件？构件与类的区别有哪些
- 简述构件图建模的步骤