



软件工程

第四章 软件架构

刘铭

2015年9月27日

主要内容



- **4.1 软件架构**

- 4.1.1 软件架构设计的背景
- 4.1.2 架构设计概念
- 4.1.3 架构设计模型

- **4.2 体系结构设计**

- **4.3 界面设计**

设计=天才+创造力

- 每个工程师都希望做设计工作，因为这里有“创造性”——客户需求、业务要求和技术限制都在某个产品或系统中得到集中的体现。
- “设计”是最充分展现工程师个人价值的工作。



“设计”的本质

- 什么是设计？设计是你身处两个世界——技术世界和人类的目标世界——而你尝试将这两个世界结合在一起。

——**Mitch Kapor**, 《软件设计宣言》



从建筑设计看软件设计

- “设计良好的建筑应该展示出坚固、适用和令人赏心悦目的特点。”
- 对好的软件设计来说也是如此
 - 适用：软件要符合开发的目标，满足用户需求；
 - 坚固：软件应该不含任何妨碍其功能的缺陷；
 - 赏心悦目：使用软件的体验应该是愉快的。



“软件设计”的定义

- 设计：为问题域的外部可见行为的规约增添实际的计算机系统实现所需的细节，包括关于人机交互、任务管理和数据管理的细节。

——Coad/Yourdon

- 关于“软件设计”的几个小例子：

- 需求1：教学秘书需要将学生的综合成绩按高到低进行排序
- 设计1：void OrderScores (struct * scores[]) { 冒泡排序算法, step1; step2;... }
- 需求2：数据字典“销售订单”
- 设计2：关系数据表Order(ID, Date, Customer, ...),
 OrderItem(No, PROD, QUANTITY, ..)
- 需求3：“查询满足条件的图书”
- 设计3：图形化web用户界面



The screenshot shows a search form for books. At the top right is a logo of three stacked books and the text "图书商品组合搜索". Below it is a search bar with the placeholder "请输入书名或作者名" and a dropdown menu set to "全部". To the right of the search bar are several input fields labeled "商品类别", "书名", "作者", "译者", "出版社", "书号", "丛书名", "原书名", and "原出版社". Below these fields is a checkbox labeled "只搜有货商品". At the bottom right is a blue button with the text "搜索 图书 商品".

设计的目标：质量

- “设计阶段”是软件工程中形成质量的关键阶段，其后所有阶段的活动都要依赖于设计的结果。
- “编写一段能工作的灵巧的代码是一回事，而设计能支持某个长久业务的东西则完全是另一回事。”



软件质量

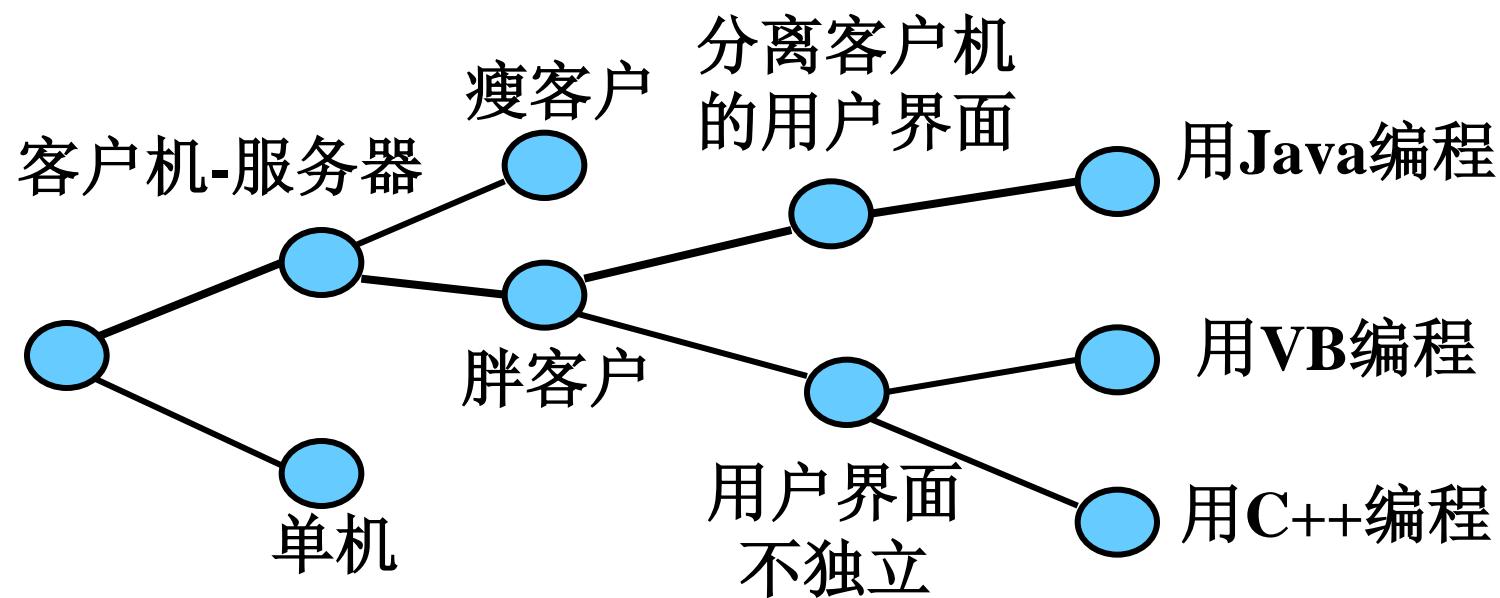


- 外部质量：面向最终用户
 - 如易用性、效率、可靠性、正确性、完整性等
- 内部质量：面向软件工程师，技术的角度
 - 如可维护性、灵活性、可重用性、可测试性等

大多数软件设计师只关注外部质量忽视内部质量，导致软件变更代价大

设计=不断的作出决策

- 解决“**How to do**”，就需要不断的做出各种“设计决策”，在各类需求之间进行“折中”，使得最终设计性能达到最优。



主要内容



- **4.1 软件架构**

- 4.1.1 软件架构设计的背景
- 4.1.2 架构设计概念
- 4.1.3 架构设计模型

- **4.2 体系结构设计**

- **4.3 界面设计**

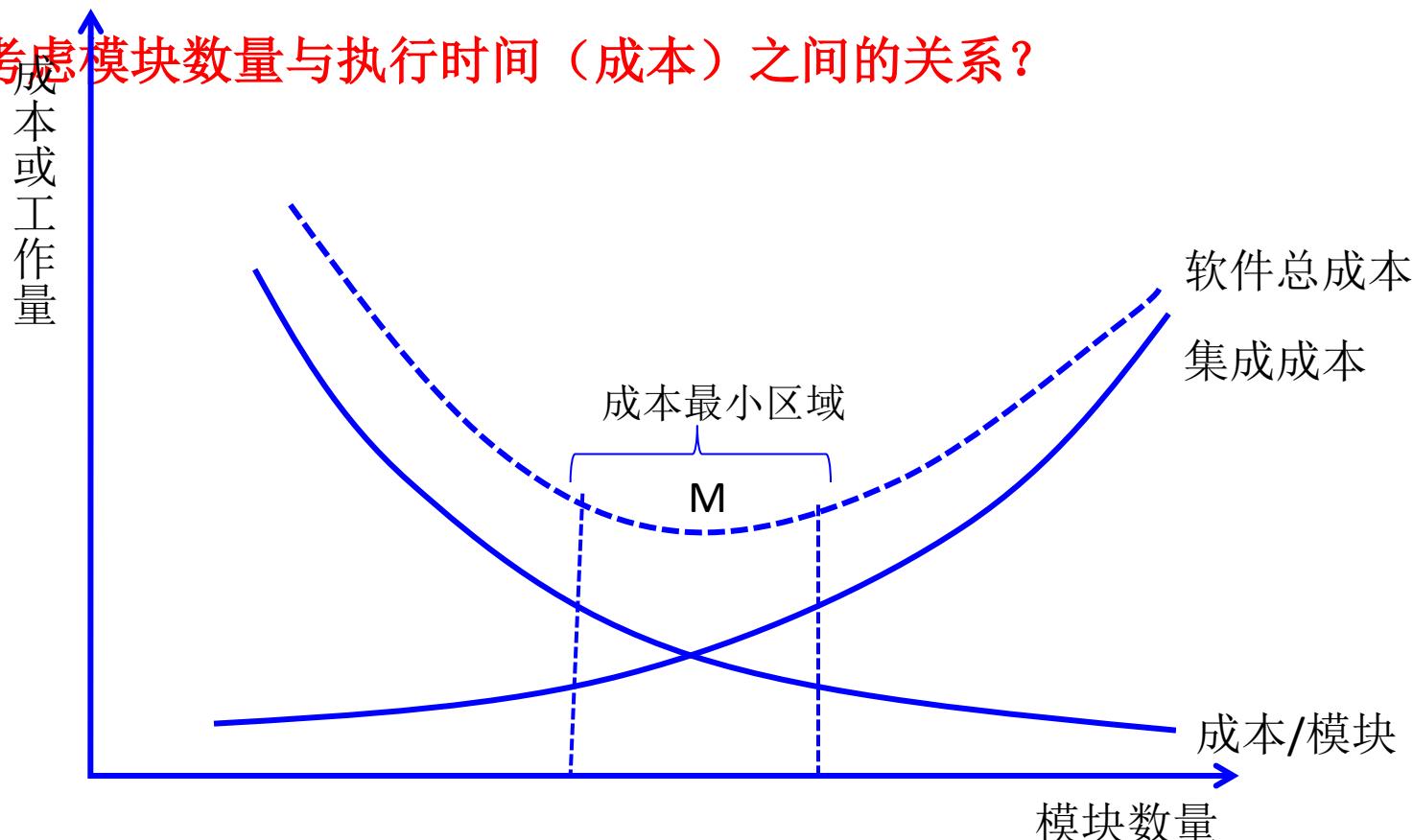
设计概念

- **抽象**: 强调关键特征, 忽略实现细节
 - 过程抽象: 具有明确和有限功能的指令序列
 - 数据抽象: 描述数据对象的冠名数据集合
- **体系结构**: 程序构件(模块)的结构、构件交互的形式、构件的数据结构
 - 功能模型: 表示系统的功能层次结构
 - 框架模型: 解决某一类问题的处理流程
 - 结构模型: 程序构件的一个有组织的集合

设计概念：模块化和软件成本

- **模块化**: 分治策略，将软件划分为独立构件（模块）

- 合理划分模块数量 复用度？复用价值？
- 考虑模块数量与执行时间（成本）之间的关系？



设计概念

- **信息隐藏：**每个模块对其他所有模块都隐藏自己的设计决策
 - 软件只通过定义清晰的接口通信
 - 每一个接口尽可能暴露最少的信息
 - 如果内部细节发生变化，外部的受到影响应当最小
- **功能独立：**是模块化、抽象概念和信息隐藏的结果
 - 功能专一，避免与其他模块过多交互

设计概念

- **重构**: 是一种重新组织的技术，不改变外部行为而是改进内部结构
- **设计类**: 精化分析类、创建新的设计类
 - 用户接口类（边界类）: 人机交互所必需的抽象
 - 领域类（实体类）: 分析类的精化
 - 过程类（控制类）: 底层业务抽象
 - 持久类: 持续存在的数据存储
 - 系统类: 软件管理和控制功能

设计建模原则

- 经常关注待建系统的架构
- 数据设计与功能设计同等重要
- 必须设计接口（包括内部接口和外部接口）
- 用户界面设计必须符合最终用户要求
- 功能独立的构件级设计
- 构件之间以及构件与外部环境之间松散耦合
- 设计表述（模型）应该做到尽可能易于理解
- 设计应该迭代式进行。每一次迭代，设计者都应该尽力简化

设计建模补充原则

- 设计必须实现所有包含在需求模型中的明确需求，而且必须满足利益相关者期望的所有隐含需求。
- 对于那些生成代码的人和那些进行测试以及随后维护软件的人而言，设计必须是可读的、可理解的指南。
- 设计必须提供软件的全貌，从实现的角度说明数据域、功能域和行为域。

主要内容



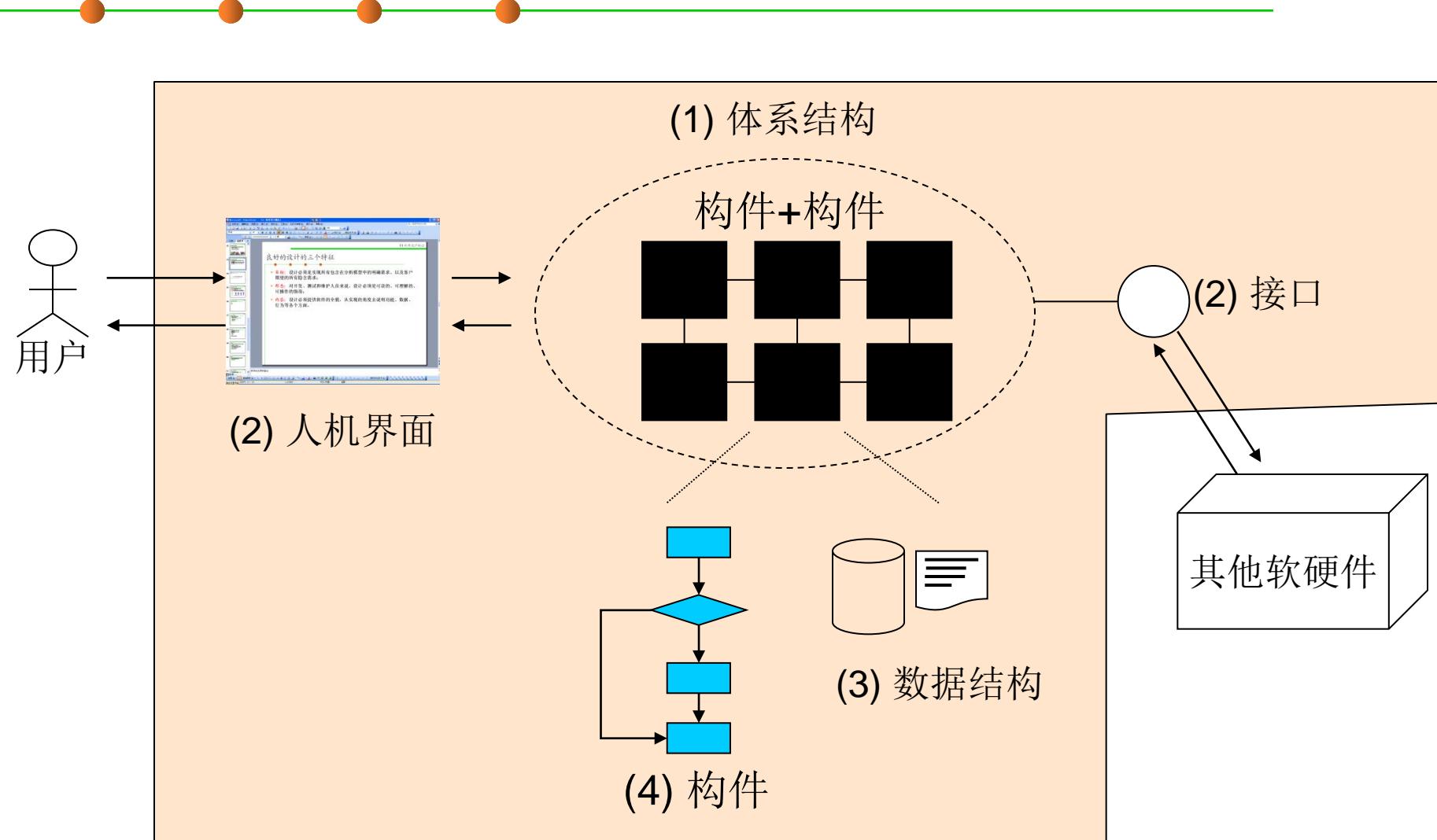
- **4.1 软件架构**

- 4.1.1 软件架构设计的背景
- 4.1.2 架构设计概念
- 4.1.3 架构设计模型

- **4.2 体系结构设计**

- **4.3 界面设计**

软件设计的元素

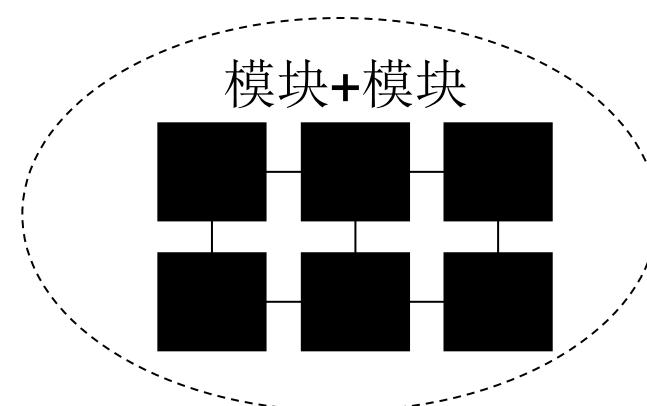


软件设计的元素

- **体系结构设计**: 定义了软件的主要结构元素之间的联系，也用于达到系统所定义需求的体系结构风格和设计模式以及影响体系结构实现方式的约束
- **构件级设计**: 将软件体系结构的结构元素变换为对软件构件的过程性描述
- **接口设计**: 描述了软件和协作系统之间、软件和使用人员之间是如何通信的
- **数据/类设计**: 将分析类模型转化为设计类的实现以及软件实现所要求的数据结构

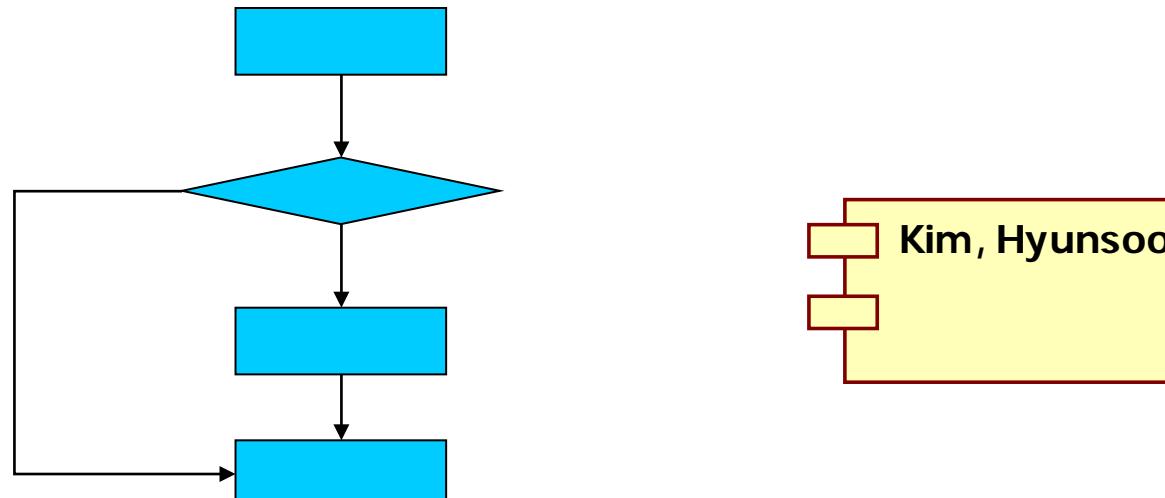
软件体系结构设计

- 选择适合于需求的软件体系结构风格，软件架构设计；
- 如何以最佳的方式划分一个系统，如何标识构件，构件之间如何通信，信息如何沟通，系统的元素如何能够独立地进化
- 例如：基于功能层次结构建立系统：
 - 采用某种设计方法，将系统按功能划分成模块的层次结构
 - 确定每个模块的功能
 - 建立与已确定的软件需求的对应关系
 - 确定模块间的调用关系
 - 确定模块间的接口
 - 评估模块划分的质量



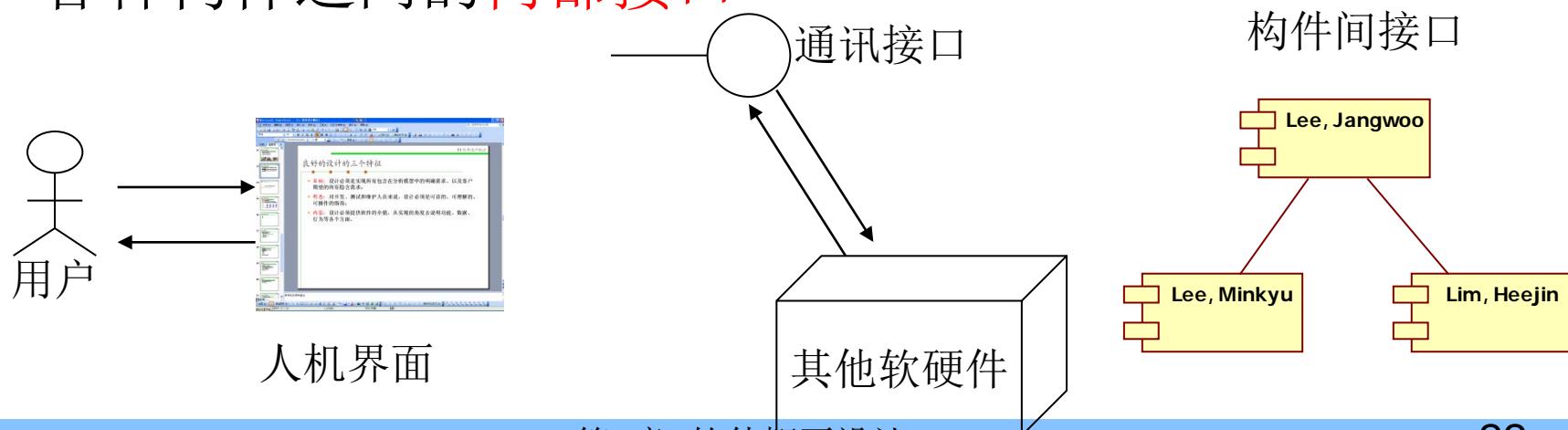
构件级设计

- 构件是面向软件体系架构的可复用软件模块
- 完整地描述了每个软件构件的内部细节
 - 为本地数据对象定义数据结构，为构件内的处理定义算法细节
 - 定义允许访问所有构件操作（行为）的接口



接口设计

- 接口是类、构件或其它分类（包括子系统）的外部可见的（公共的）操作说明，而没有内部结构的规格说明
 - 用户界面（UI）
 - 与其它系统、硬件的外部接口
 - 各种构件之间的内部接口



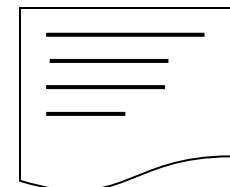
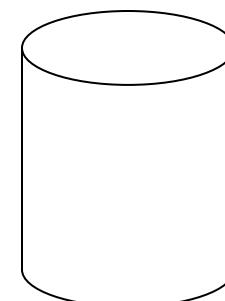
数据设计

■ 体系结构级数据设计

- 确定软件涉及的文件系统的结构以及数据库的模式、子模式，进行数据完整性和安全性的设计
- 确定输入、输出文件的详细的数据结构

■ 构件级数据设计

- 结合算法设计，确定算法所必需的逻辑数据结构及其操作
- 确定对逻辑数据结构所必需的那些操作的程序模块(软件包)
- 确定和限制各个数据设计决策的影响范围
- 确定其详细的数据结构和使用规则
- 数据的一致性、冗余性设计

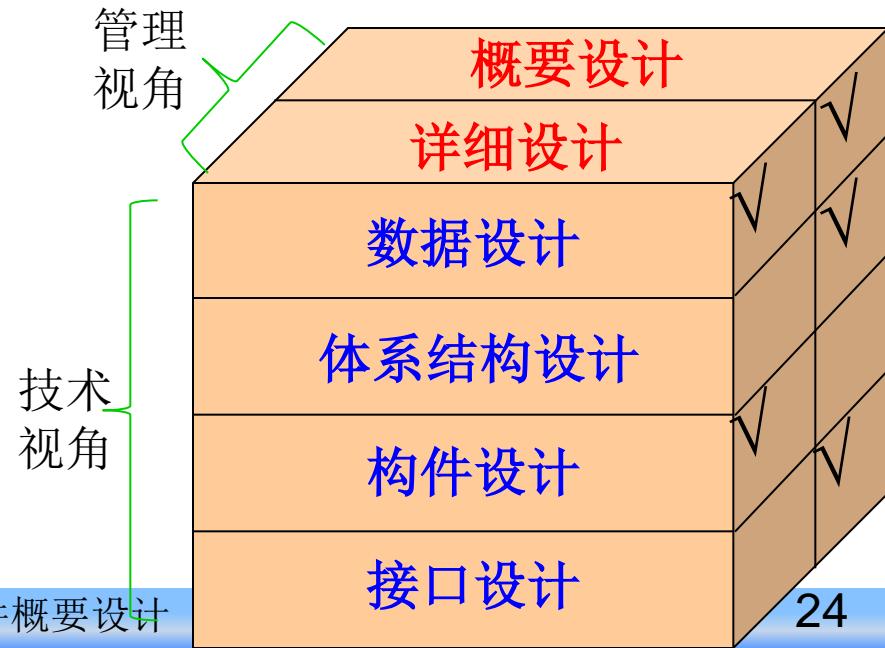


数据结构和数据库课程详细介绍

软件设计的两大阶段

- 从工程管理的角度看，软件设计包括：

- 概要设计：将软件需求转化为数据结构和软件的系统结构
- 详细设计：即构件设计，通过对软件结构表示进行细化，得到软件的详细的数据结构和算法



主要内容

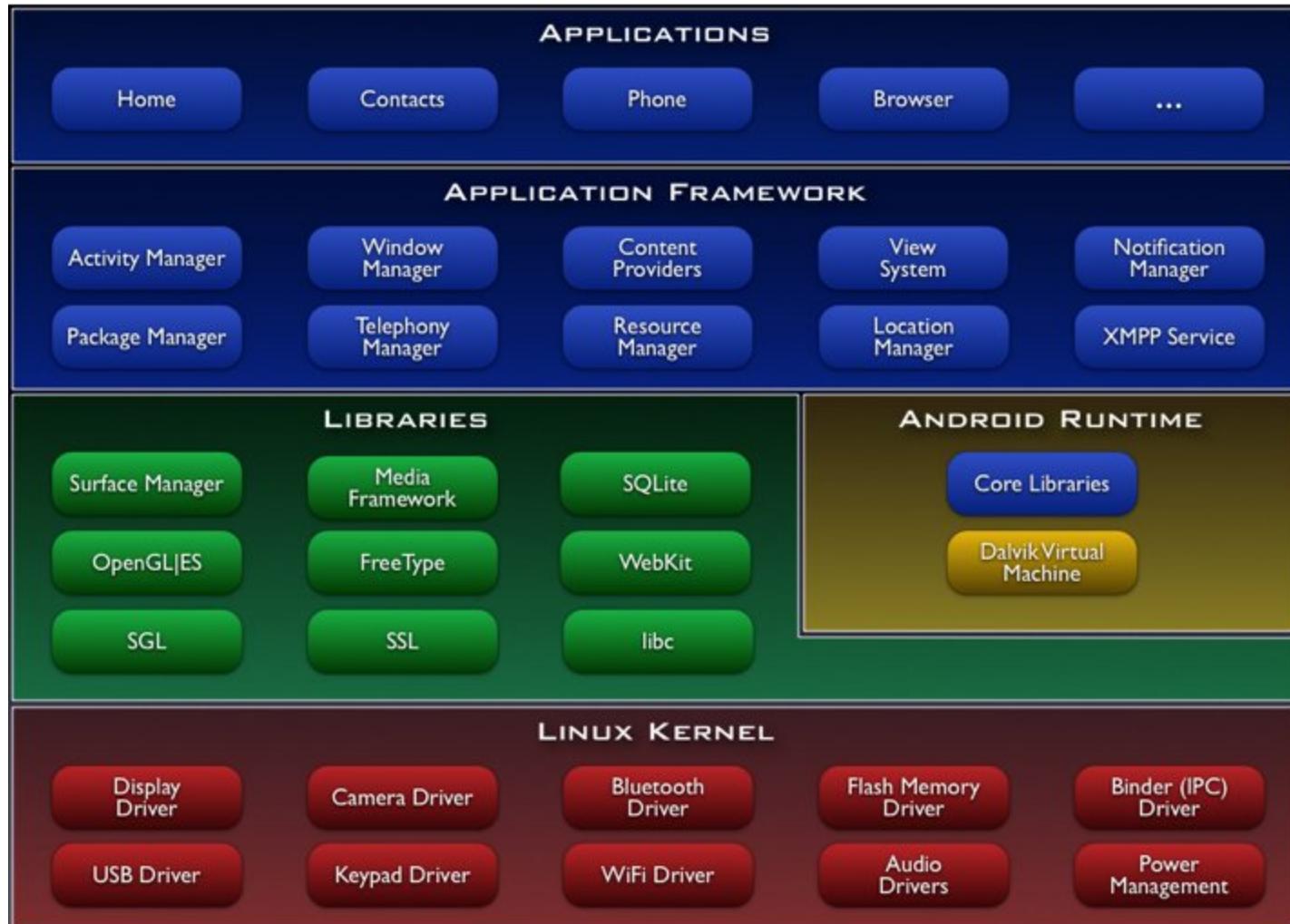


- **4.1 软件设计**
- **4.2 体系结构设计**
 - 4.2.1 体系结构的背景与定义
 - 4.2.2 体系结构的基本概念
 - 4.2.3 体系结构风格
 - 4.2.4 体系结构设计
- **4.3 界面设计**

你早已接触过“体系结构” ...

- 超市的多个收银台，顾客排长队 === 服务器并发处理的性能和容量
- 十字路口的车辆等待转弯 === 通过缓存来提高交通吞吐率
- 分层、构件化、服务化、标准化、 ...
- 缓存、分离/松散耦合、队列、复制、冗余、代理、 ...
- C/S、B/S、负载平衡、 ...
- 如何用它们解决具体软件问题，在博弈中寻求平衡(折中)，就是软件体系结构所关注的问题。
- 新的架构层出不穷，但所遵循的基本原理都是相通的。

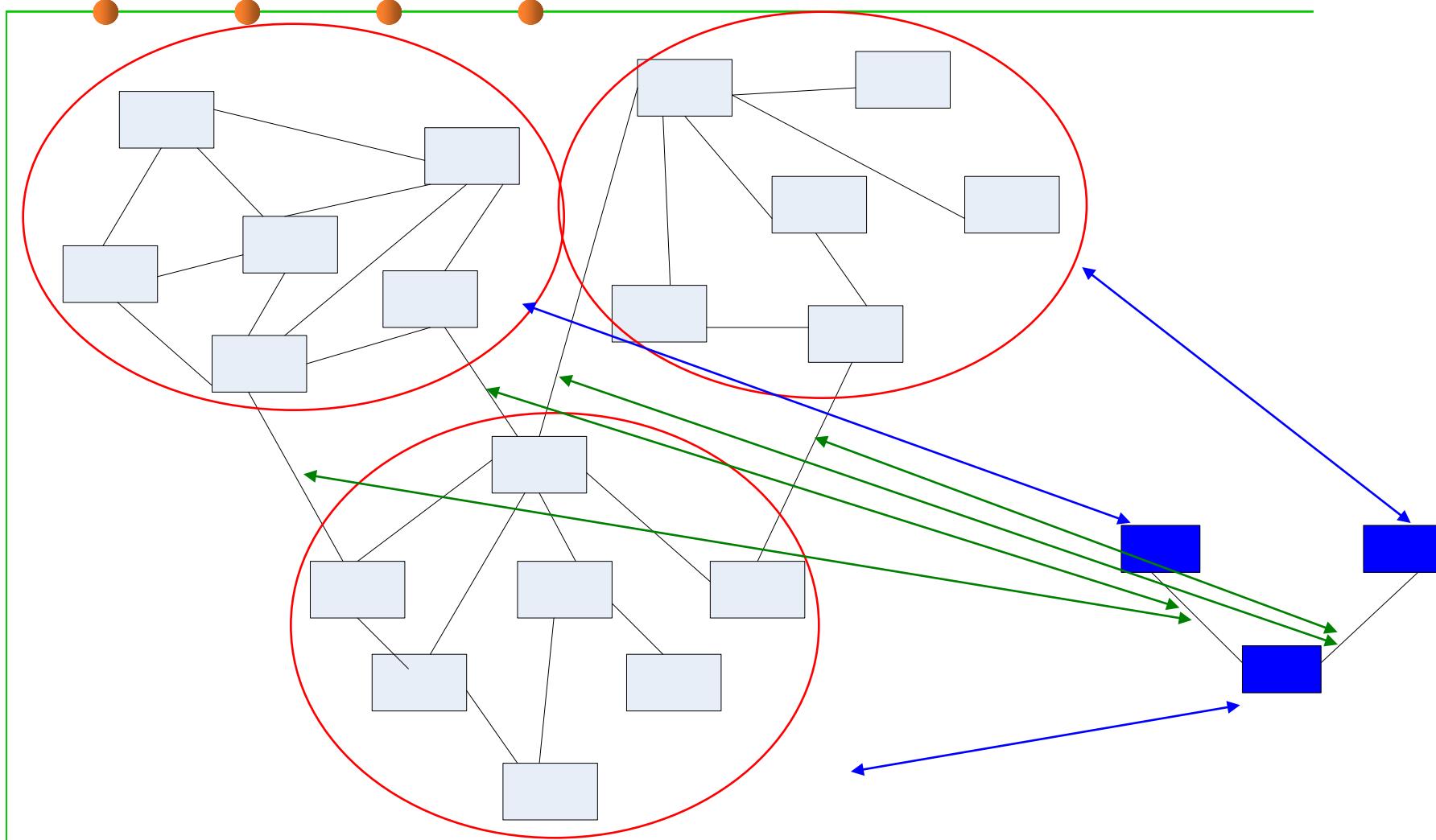
示例：Google Android体系结构



1. 软件体系结构：软件构建的方式



软件体系结构的例子



“体系结构”的共性

- 共性：

- 一组基本的构成要素——构件
- 这些要素之间的连接关系——连接件
- 这些要素连接之后形成的拓扑结构——物理分布
- 作用于这些要素或连接关系上的限制条件——约束
- 质量——性能

归纳：SA的定义



- **软件体系结构(SA):**

- 是一个关于系统形式和结构的综合框架，包括系统构件和构件的整合
- 从一个较高的层次来考虑组成系统的构件、构件之间的连接，以及由构件与构件交互形成的拓扑结构
- 这些要素应该满足一定的限制，遵循一定的设计规则，能够在一定的环境下进行演化
- 反映系统开发中具有重要影响的设计决策，便于各种人员的交流，反映多种关注，据此开发的系统能完成系统既定的功能和性能需求

体系结构 = 构件 + 连接件 + 拓扑结构 + 约束 + 质量

Architecture = Components + Connectors + Topology + Constraints + Performance

2. 为什么体系结构如此重要

- 软件体系结构的表示有助于对计算机系统开发感兴趣的各方开展交流
- 体系结构突出了早期的设计决策，这些决策对随后的所有软件工程工作有深远的影响，同时对系统作为一个可运行实体的最后成功有重要作用
- 体系结构“构建了一个相对小的，易于理解的模型，该模型描述了系统如何构成以及其构件如何一起工作”
- 结论：对于大规模的复杂软件系统来说，对总体的系统结构设计和规格说明比起对计算的算法和数据结构的选择已经变得明显重要得多。

3. 软件体系结构要回答的基本问题

- 软件的**基本构造单元**是什么？
- 这些构造单元之间如何**连接**？
- 最终形成何种样式的**拓扑结构**？
- 每个典型应用领域的**典型体系结构**是什么样子？
- 如何进行软件体系结构的**设计与实现**？
- 如何对已经存在的软件体系结构进行**修改**？
- 使用何种**工具**来支持软件体系结构的设计？
- 如何对软件的体系结构进行**描述**，并据此进行**分析和验证**？

4. 软件体系结构的目标

- 软件体系结构关注的是：

- 如何将复杂的软件系统划分为模块、如何规范模块的构成和性能、以及如何将这些模块组织为完整的系统。

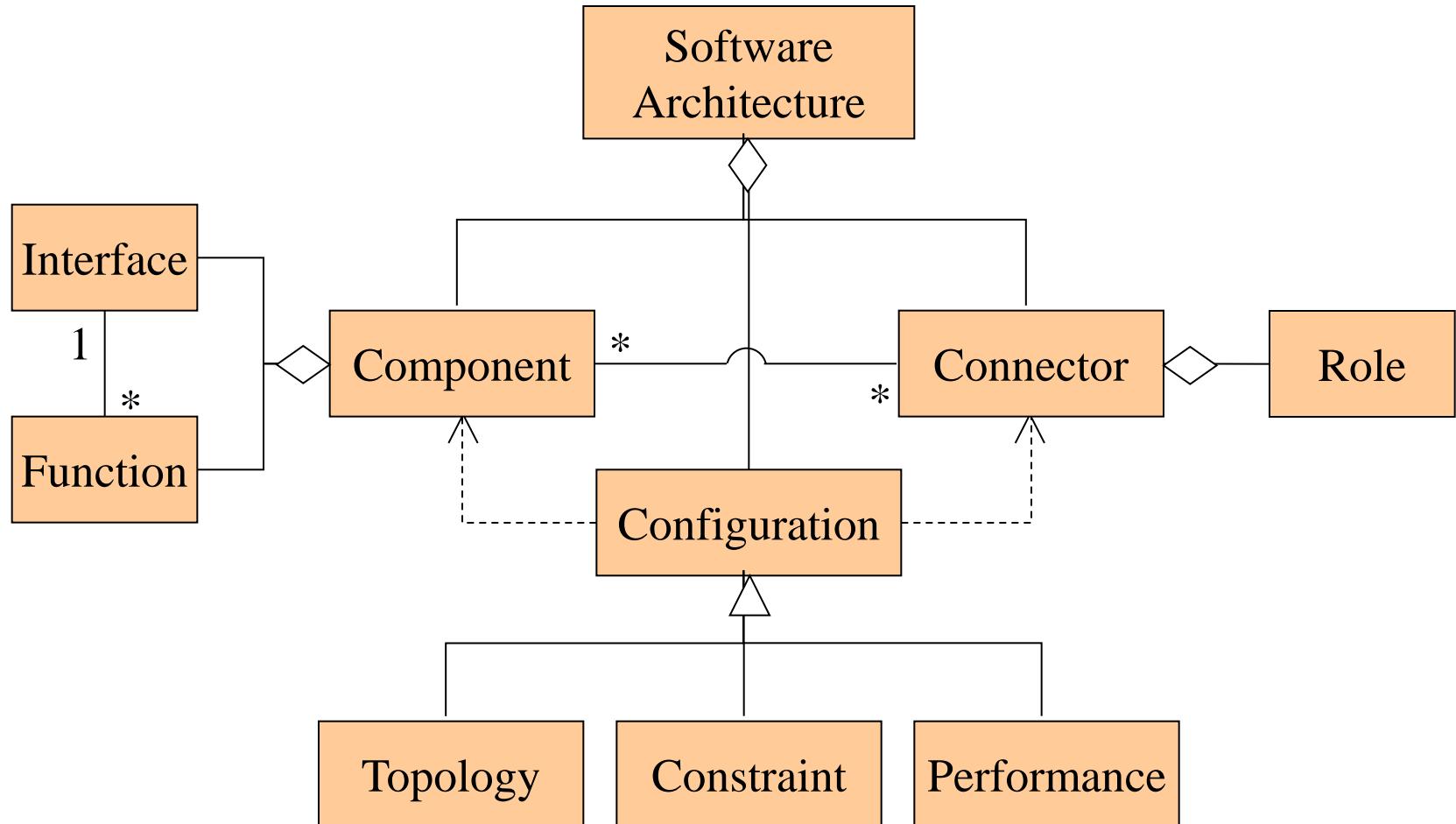
- 主要目标：

- 建立一个一致的系统及其视图集，并表达为最终用户和软件设计者需要的结构形式，支持用户和设计者之间的交流与理解。

主要内容

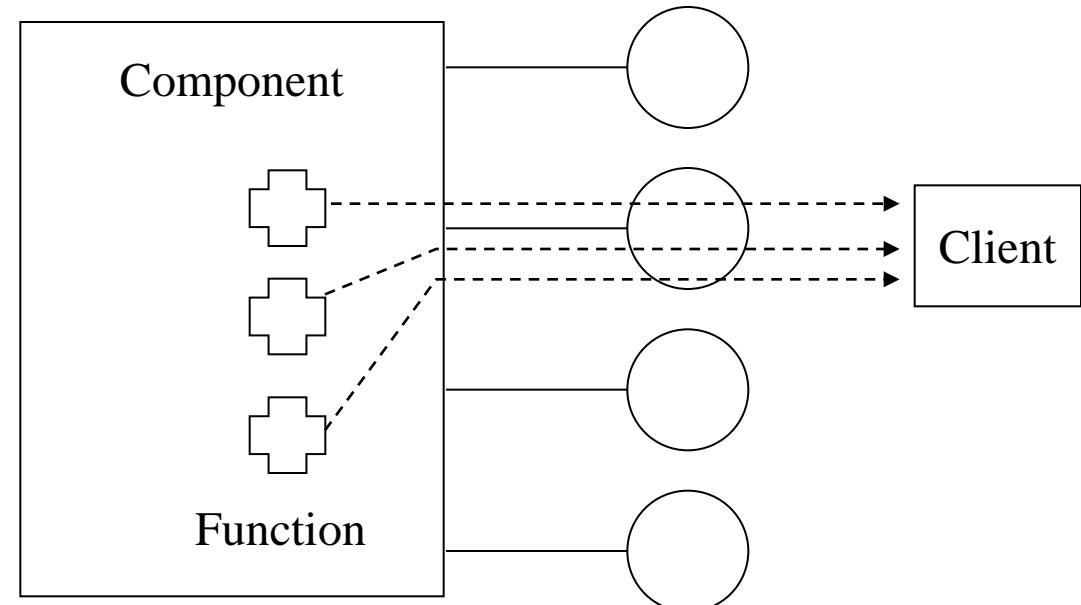
- 4.1 软件设计
- 4.2 体系结构设计
 - 4.2.1 体系结构的背景与定义
 - 4.2.2 体系结构的基本概念
 - 4.2.3 体系结构风格
 - 4.2.4 体系结构设计
- 4.3 界面设计

SA中的核心概念



1. 构件(Component)

- “构件”是具有某种功能的可复用的软件结构单元，表示了系统中主要的计算元素和数据存储，具有提供的接口描述。
- 构件是一个抽象的概念，任何在系统运行中承担一定功能、发挥一定作用的软件体都可看作是构件。
 - 程序函数、模块
 - 对象、类
 - 数据库
 - 文件
 -



构件组成

- 构件组成

- 接口

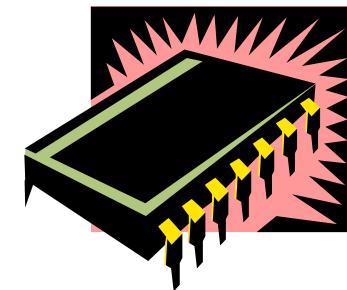
- 构件接口是构件间的契约
 - 一个接口提供一种服务，完成某种逻辑行为

- 实现（功能）

- 构件接口服务的实现
 - 构件核心逻辑实现

接口(Interface)与功能(Function)

- 构件作为一个封装的实体，只能通过其**接口(Interface)**与外部环境交互，表示了构件和外部环境的**交互点**，**内部具体实现则被隐藏起来(Black-box)**；
- 构件接口与其内部实现应严格分开
- 构件内部所实现的功能以**方法、操作(functions、behaviors)**的形式体现出来，并通过接口向外发布，进而产生与其它构件之间的关联。



2. 连接(Connection)



- **连接(Connection)**: 构件间建立和维护行为关联与信息传递的途径;
- 连接需要两方面的支持:
 - 连接发生和维持的机制——实现连接的物质基础(**连接的机制**);
 - 连接能够正确、无二义、无冲突进行的保证——连接正确有效的进行信息交换的规则(**连接的协议**)。
 - 简称“机制”(mechanism)和“协议”(protocol)。

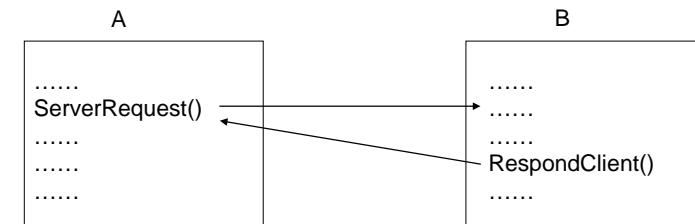
连接的机制

- 从连接目的与手段看：

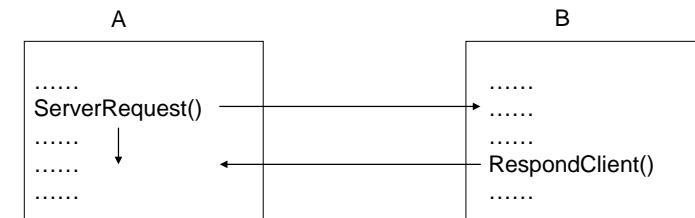
- 函数/过程调用； A { call B.f(); } B { f(); }
- 事件/消息发送； A { send msg (m) to B; } B { receive m and do sth; }
- 数据传输； A {write data to DB; } B {read data from DB; }
- ..

- 除了连接机制/协议的实现难易之外，影响连接实现复杂性的因素之一是“**有无连接的返回信息和返回的时间**”，分为：

- 同步 (Synchronous)
- 异步 (Asynchronous)



同步连接机制



异步连接机制

连接的协议(Protocol)

- 协议(**Protocol**)是连接的规约(**Specification**);
- 连接的规约是建立在物理层之上的有意义信息形式的表达规定
 - 对过程调用来说：参数的个数和类型、参数排列次序；
例：double GetHighestScore (int courseID, String classID) {...}
 - 对消息传送来说：消息的格式
例：class Message {int msgNo; String bookName; String status;}
- 目的：使双方能够互相理解对方所发来的信息的语义。

连接件(Connector)

- 连接件(**Connector**): 表示构件之间的交互并实现构件之间的连接, 如:

- 管道(pipe)
- 过程调用(procedure call)
- 事件广播(event broadcast)
- 客户机-服务器(client-server)
- 数据库连接(SQL)



- 典型连接件: **CICS、MQ、JMS**
- 连接件也可看作一类**特殊的构件**, 区别在于:
 - 一般构件是软件功能设计和实现的承载体;
 - 连接件是负责完成构件之间信息交换和行为联系的专用构件

主要内容



- **4.1 软件设计**
- **4.2 体系结构设计**
 - 4.2.1 体系结构的背景与定义
 - 4.2.2 体系结构的基本概念
 - 4.2.3 体系结构风格
 - 4.2.4 体系结构设计
- **4.3 界面设计**

从“建筑风格”开始

- Architectural style constitutes a mode of classifying architecture largely by morphological characteristics in terms of form, techniques, materials, etc. (建筑风格等同于建筑体系结构的一种可分类的模式，通过诸如外形、技术和材料等形态上的特征加以区分)。
- 之所以称为“风格”，是因为经过长时间的实践，它们已经被证明具有良好的工艺可行性、性能与实用性，并可直接用来遵循与模仿(复用)。

中国古典建筑



宫殿风格



園林风格



江南建筑风格

第7章 软件概要设计

欧洲古典建筑



文艺复兴风格



巴洛克风格



拜占庭风格

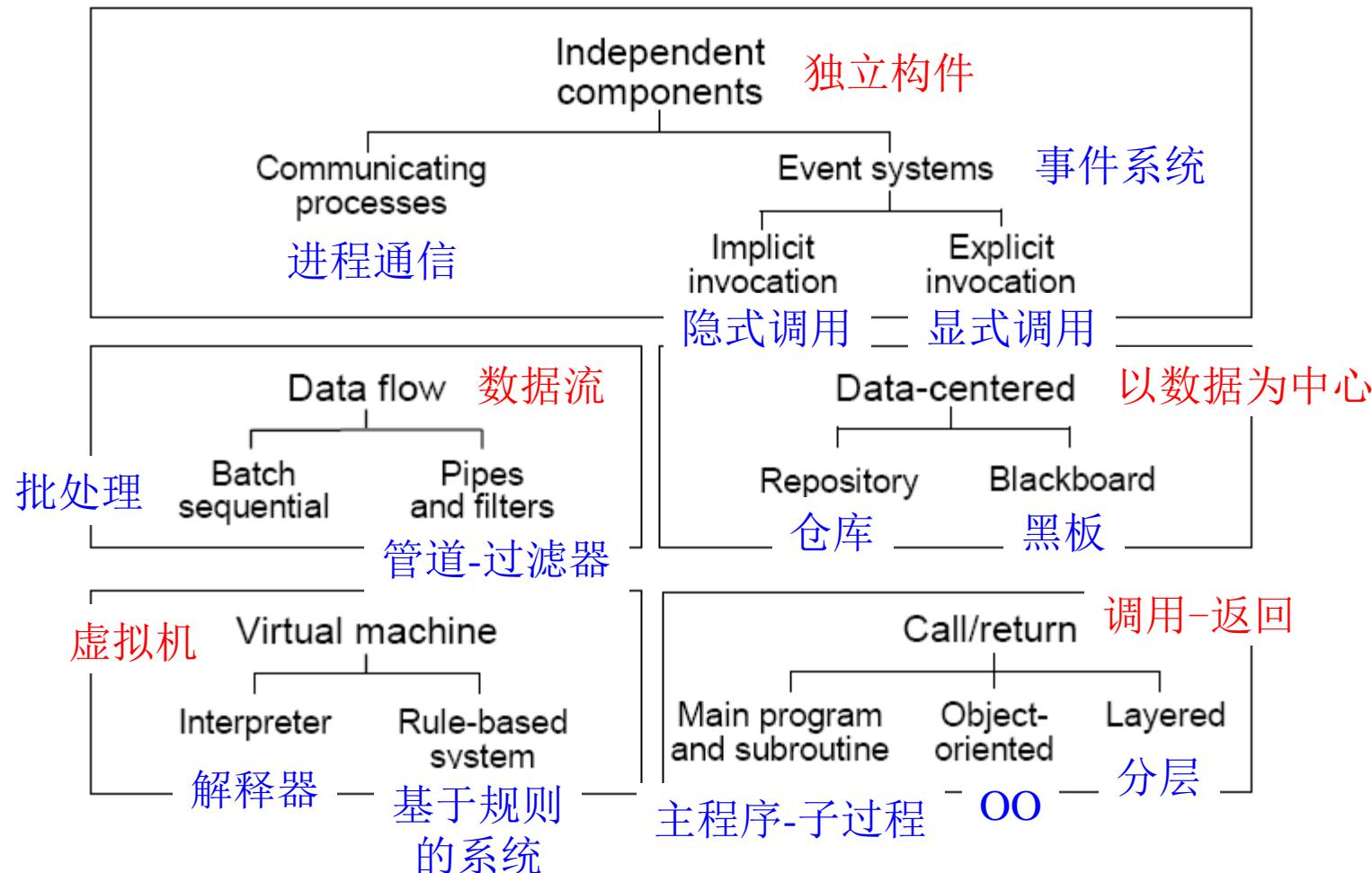


哥特风格

软件体系结构风格

- 软件系统同建筑一样，也具有若干特定的“风格”
(software architectural style);
 - 这些风格在实践中被多次设计、应用，已被证明具有良好的性能、可行性和广泛的应用场景，可以被重复使用；
 - 实现“软件体系结构级”的复用。
- 定义：
 - 描述特定领域中软件系统家族的组织方式的惯用模式，反映了领域中众多系统所**共有的结构和语义特性**，并指导如何将各个模块和子系统有效地组织成一个完整的系统。

经典体系结构风格



主要软件体系结构风格

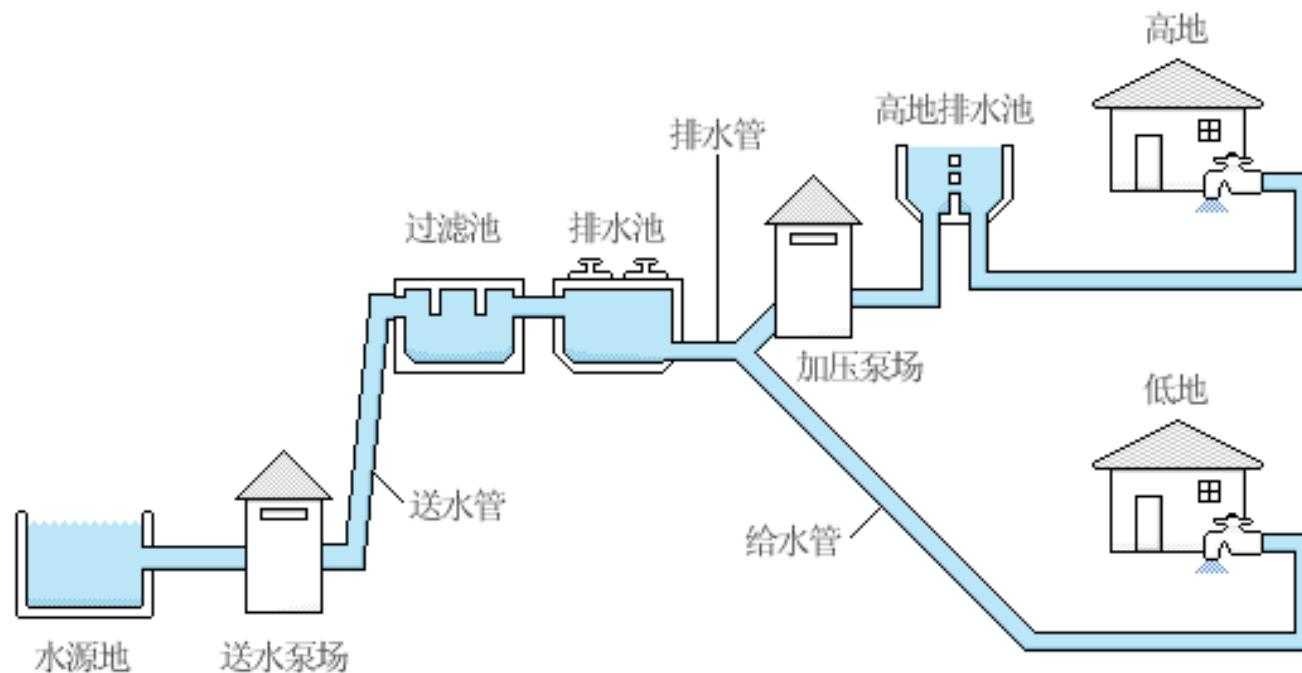
- 1. 数据流风格
- 2. 以数据为中心的风格(仓库)
- 3. 主程序-子过程
- 4. 面向对象风格
- 5. 层次风格
 - ◆ 客户机-服务器 (C/S)
 - ◆ 浏览器-服务器 (B/S)
- 6. *模型-视图-控制器(**MVC**)

数据视角

功能视角
调用返回

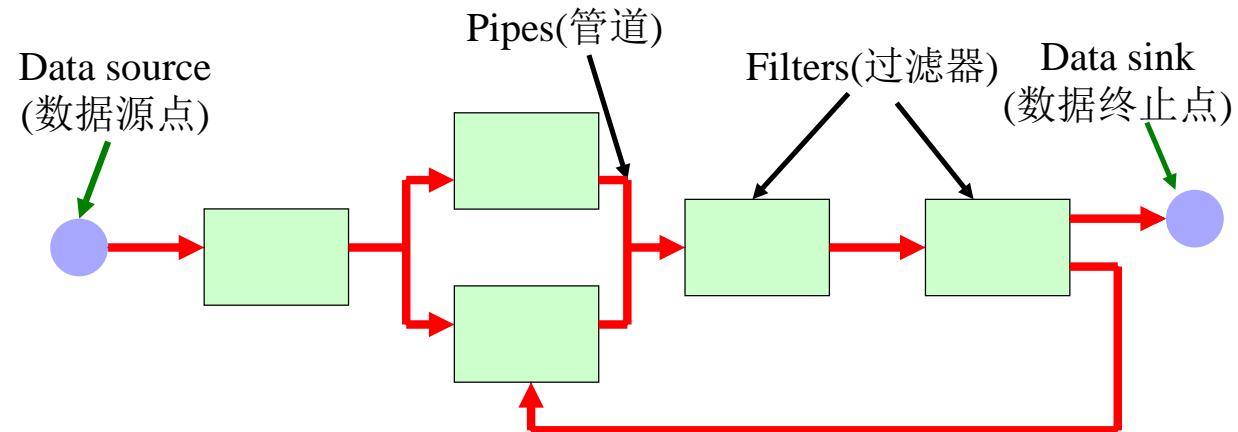


现实中的“数据流”体系结构

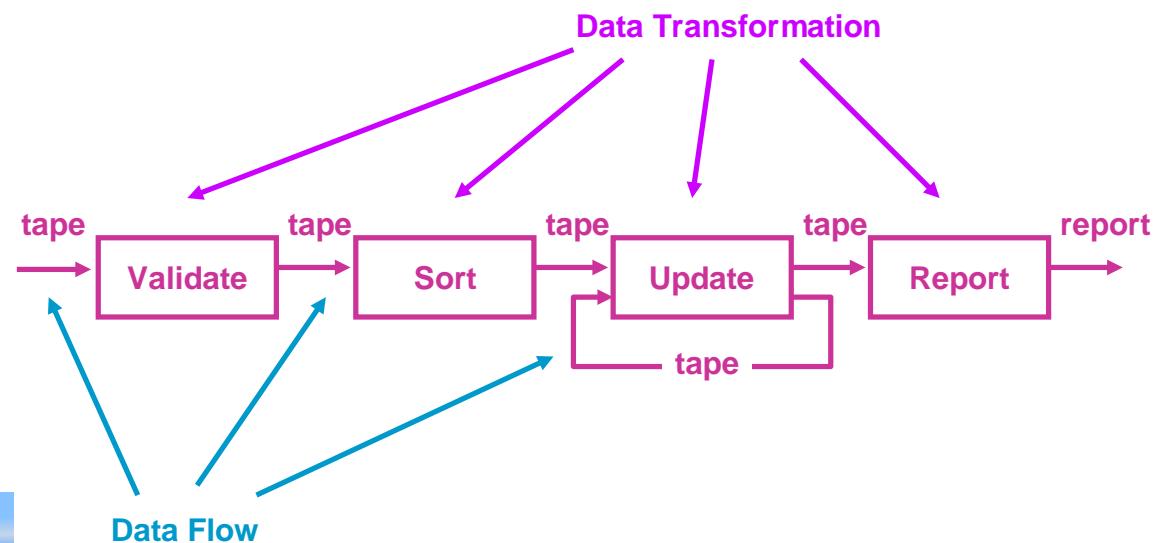


1. 数据流风格

- 管道-过滤器风格

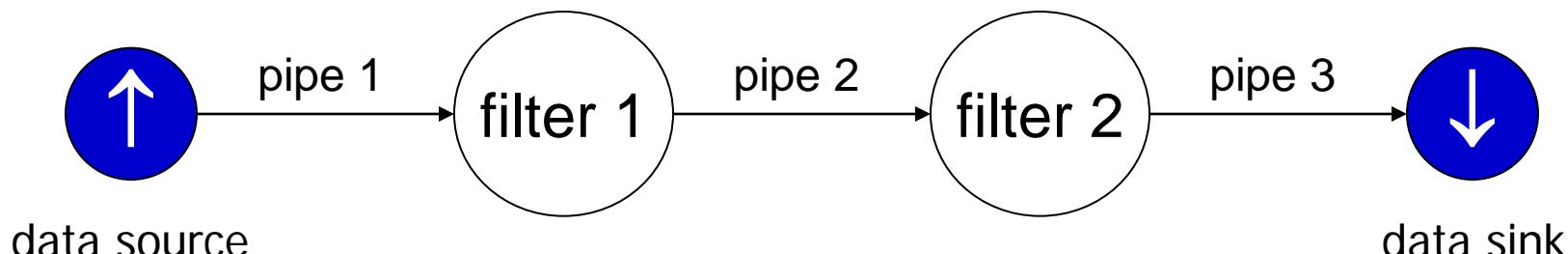


- 顺序批处理风格

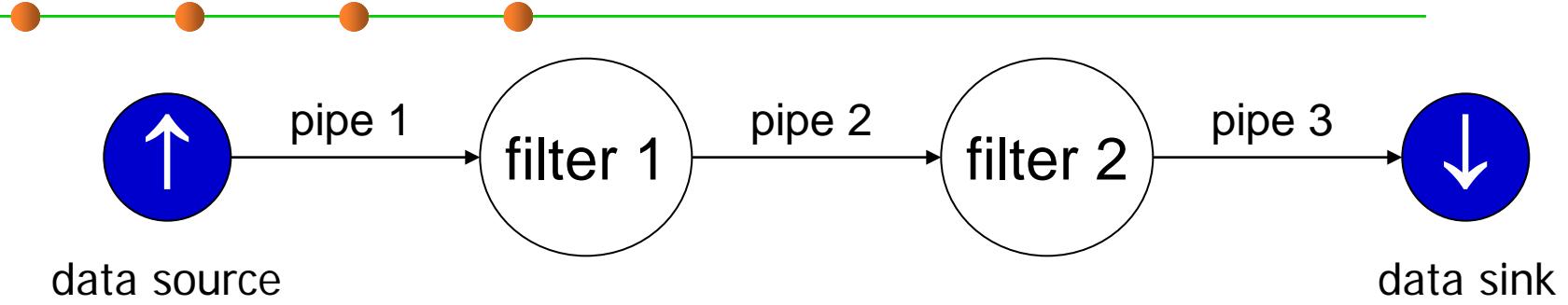


1. 软件中的数据流风格

- 情境：数据源源不断的产生，系统需要对这些数据进行若干处理（分析、计算、转换等）。
- 解决方案：
 - 把系统分解为几个序贯的处理步骤，这些步骤之间通过数据流连接，一个步骤的输出是另一个步骤的输入；
 - 每个处理步骤由一个过滤器构件(Filter)实现；
 - 处理步骤之间的数据传输由管道(Pipe)负责。
- 每个处理步骤(过滤器)都有一组输入和输出，过滤器从管道中读取输入的数据流，经过内部处理，然后产生输出数据流并写入管道中。



1. 软件中的数据流风格



```
filter1 {  
    Read data  $d$  from pipe1;  
    Deal with  $d$  and transform it to  $d'$ ;  
    Write  $d'$  to pipe2;  
}
```

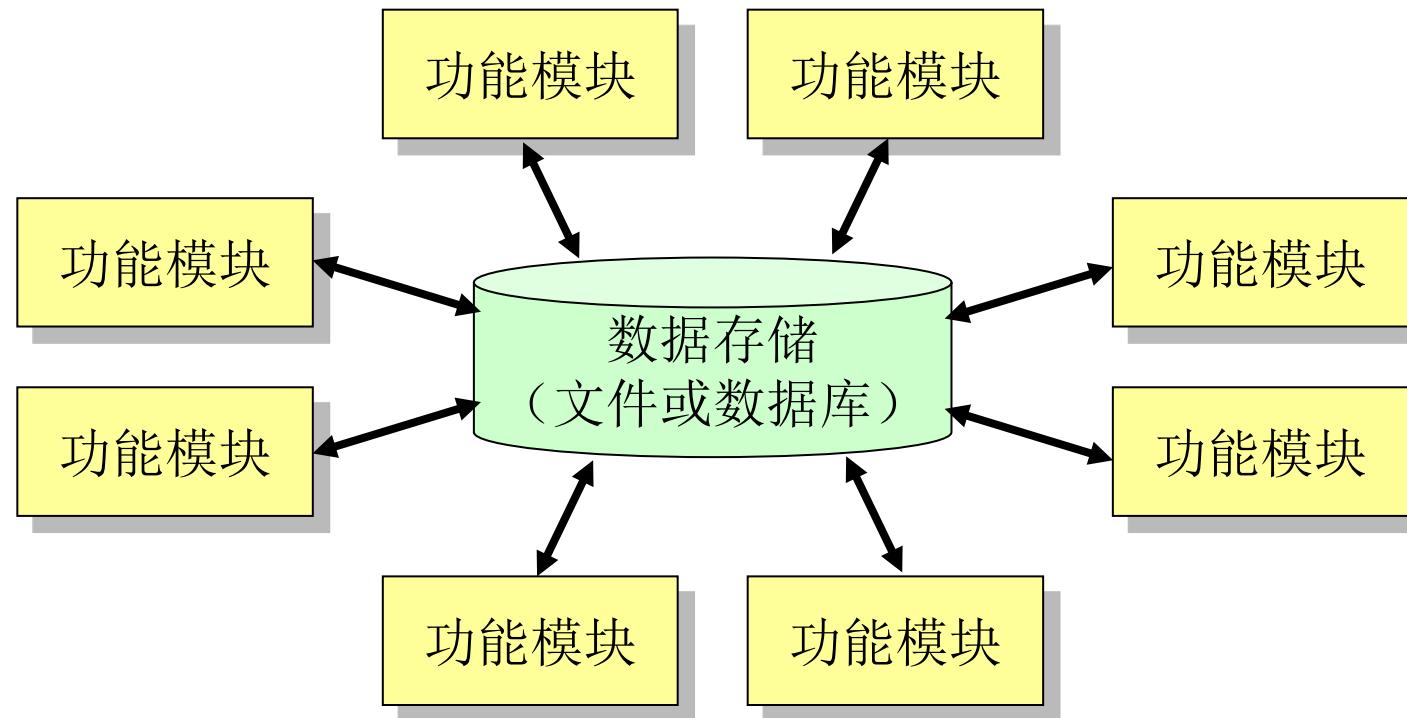
```
filter2 {  
    Read data  $d'$  from pipe2;  
    Deal with  $d'$  and transform it to  $d''$ ;  
    Write  $d''$  to pipe3;  
}
```

现实中的典型应用领域:

编译器、Unix管道、图像处理、信号处理、网络监控与管理等

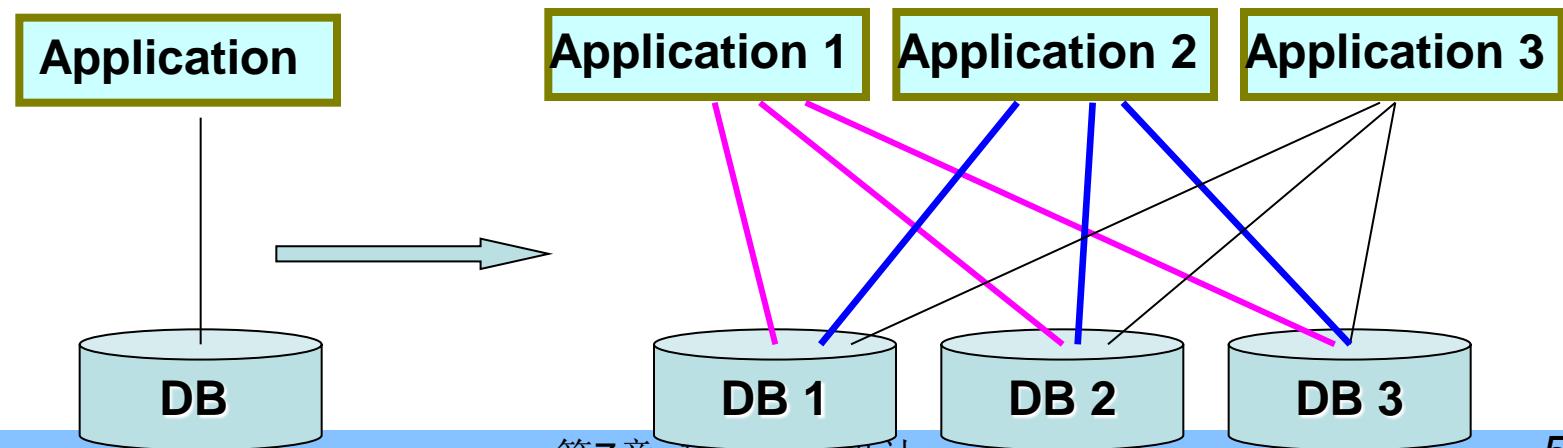
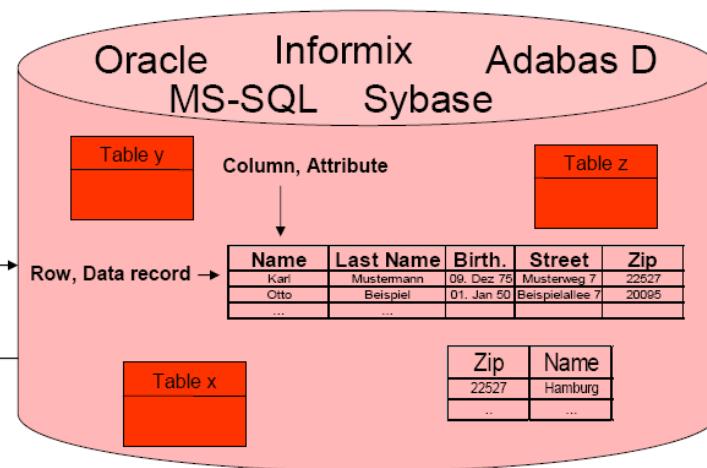
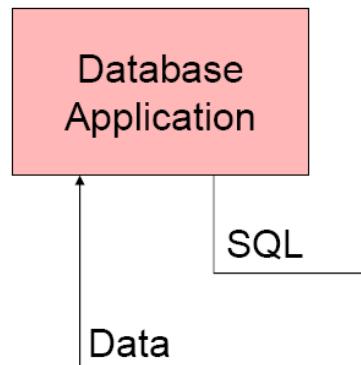
2. 以数据为中心的体系结构风格

- 以数据为中心的体系结构风格(也称仓库风格)

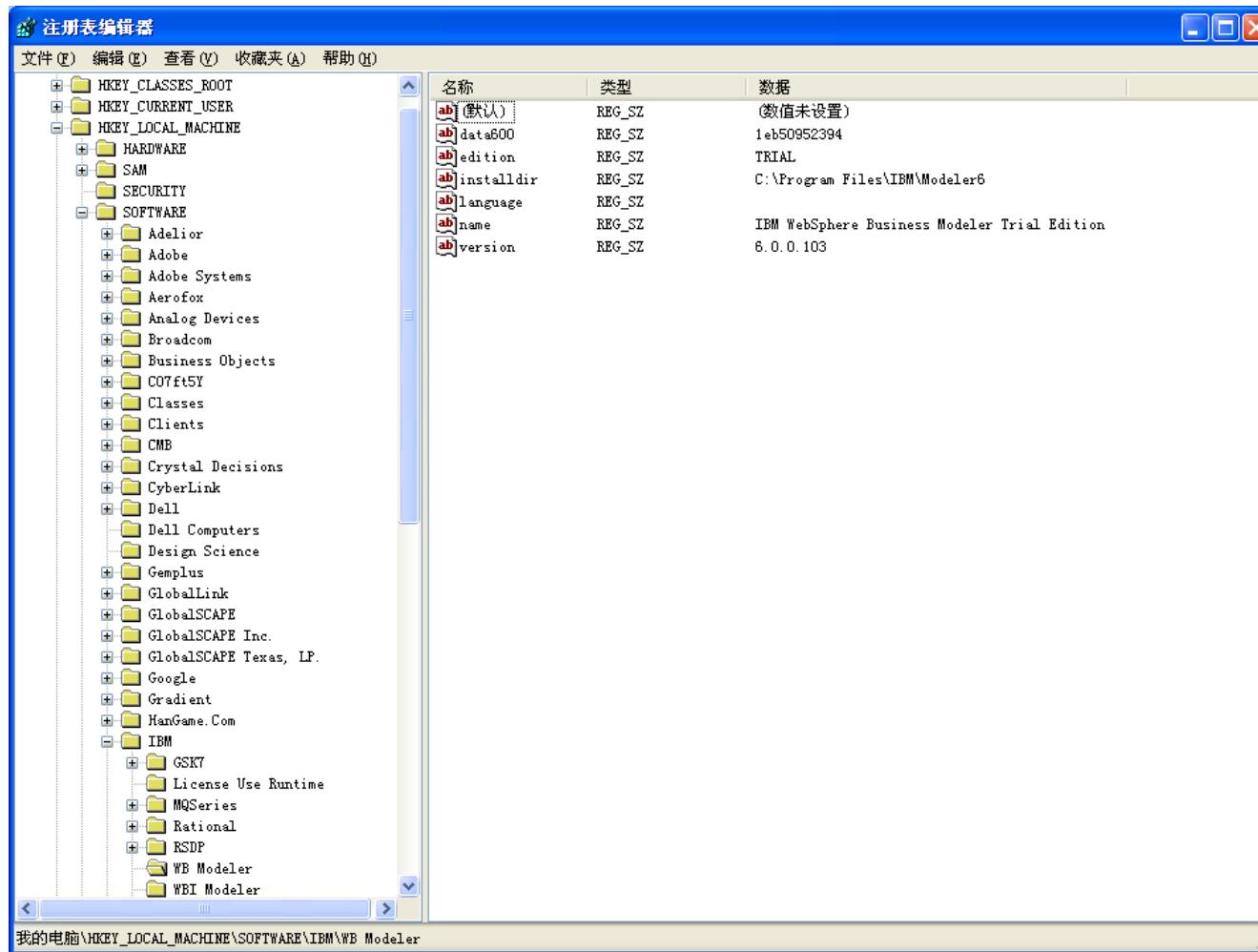


示例1：基于数据库的系统结构

VB、PB等开发的典型信息系统软件

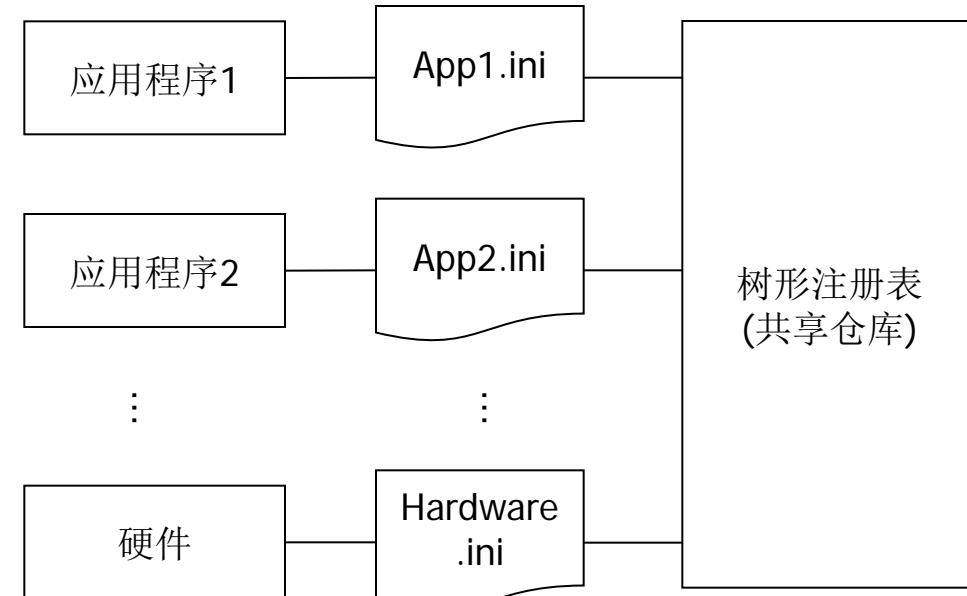


例1：注册表(Windows Registry)



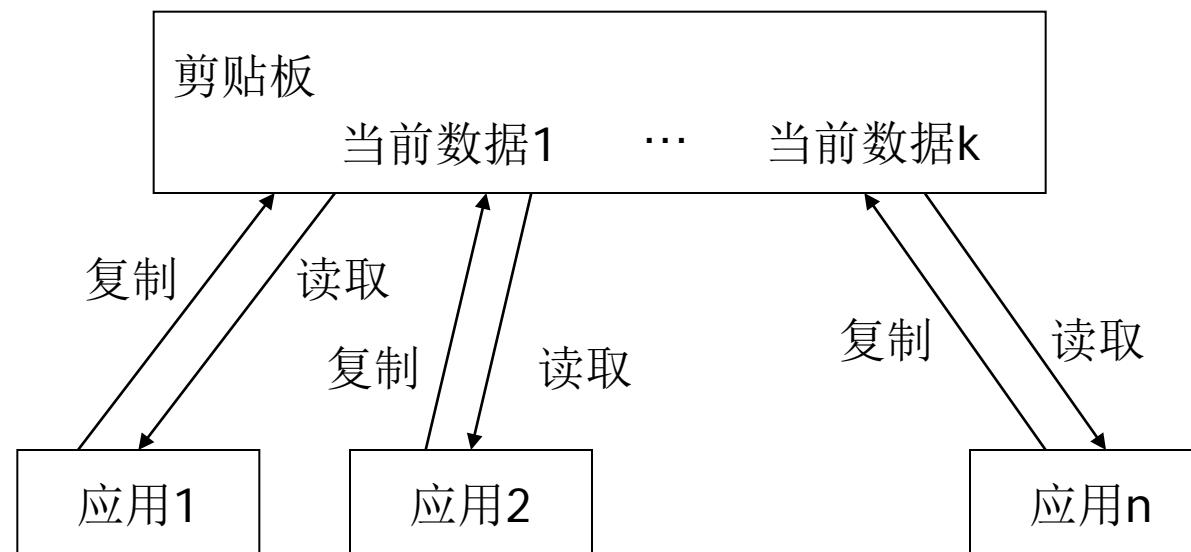
注册表的结构

- 最初，硬件/软件系统的配置信息均被各自保存在一个配置文件中(.ini)；
- 这些文件散落在系统的各个角落，很难对其进行维护；
- 为此，引入注册表的思想，将所有.ini文件集中起来，形成共享仓库，为系统运行起到了集中的资源配置管理和控制调度的作用。
- 注册表中存在着系统的所有硬件和软件配置信息，如启动信息、用户、**BIOS**、各类硬件、网络、**INI**文件、驱动程序、应用程序等；
- 注册表信息影响或控制系统/应用软件的行为，应用软件安装/运行/卸载时对其进行添加/修改/删除信息，以达到改变系统功能和控制软件运行的目的。



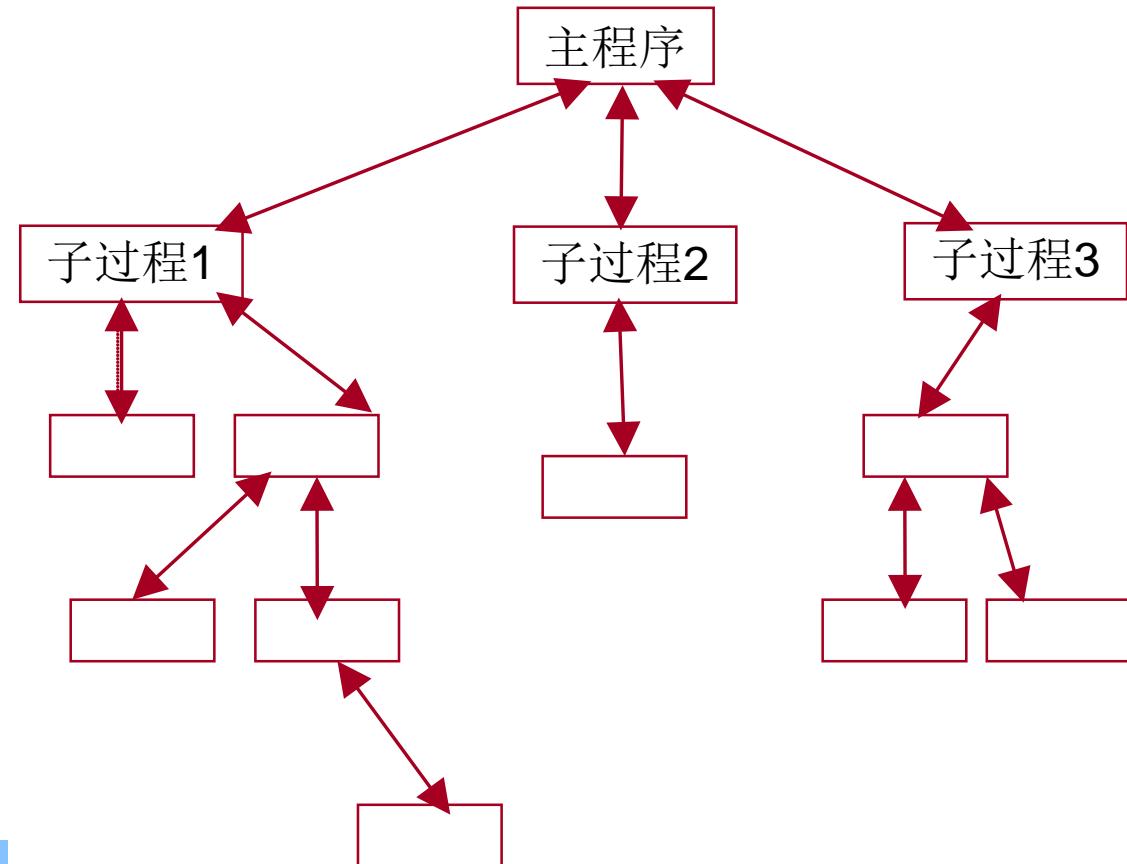
例2：剪贴板 (Clipboard)

- 剪贴板是一个用来进行短时间的数据存储并在文档/应用之间进行数据传递和交换的软件程序
 - 用来存储带传递和交换信息的公共区域(形成共享数据仓库);
 - 不同的应用程序通过该区域交换格式化的信息;
 - 访问剪贴板的方式：copy & paste.



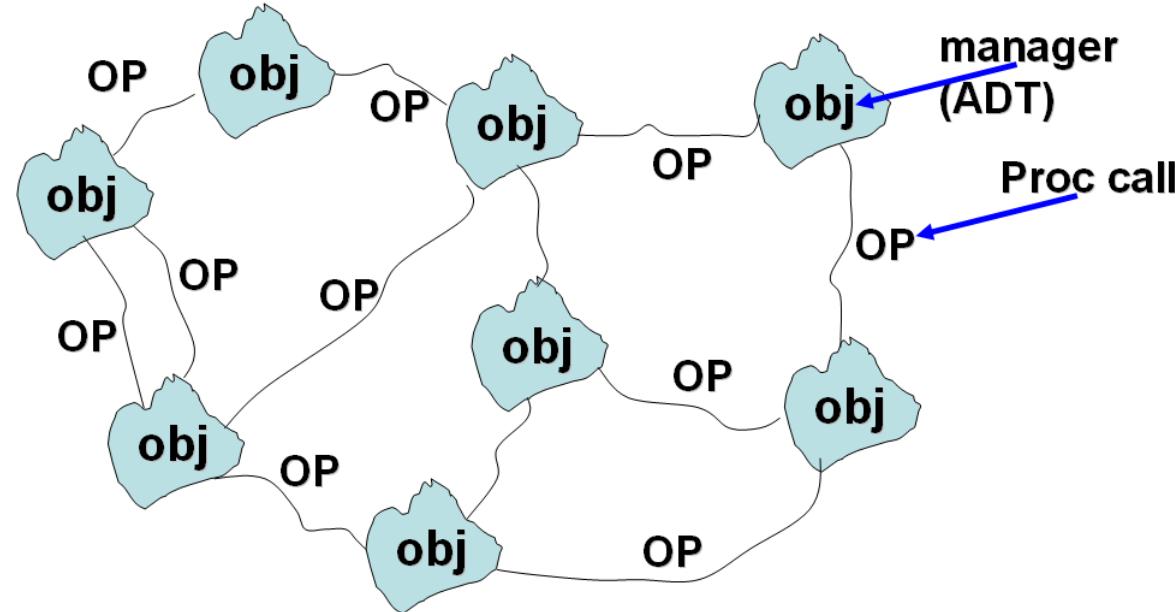
3. 主程序-子过程

- 该风格是结构化程序设计的一种典型风格，从功能的观点设计系统，通过逐步分解和逐步细化，得到系统体系结构。
 - 构件：主程序、子程序
 - 连接器：调用-返回机制
 - 拓扑结构：层次化结构
- 本质：**将大系统分解为若干模块(模块化)**，主程序调用这些模块实现完整的系统功能。



4. 面向对象风格

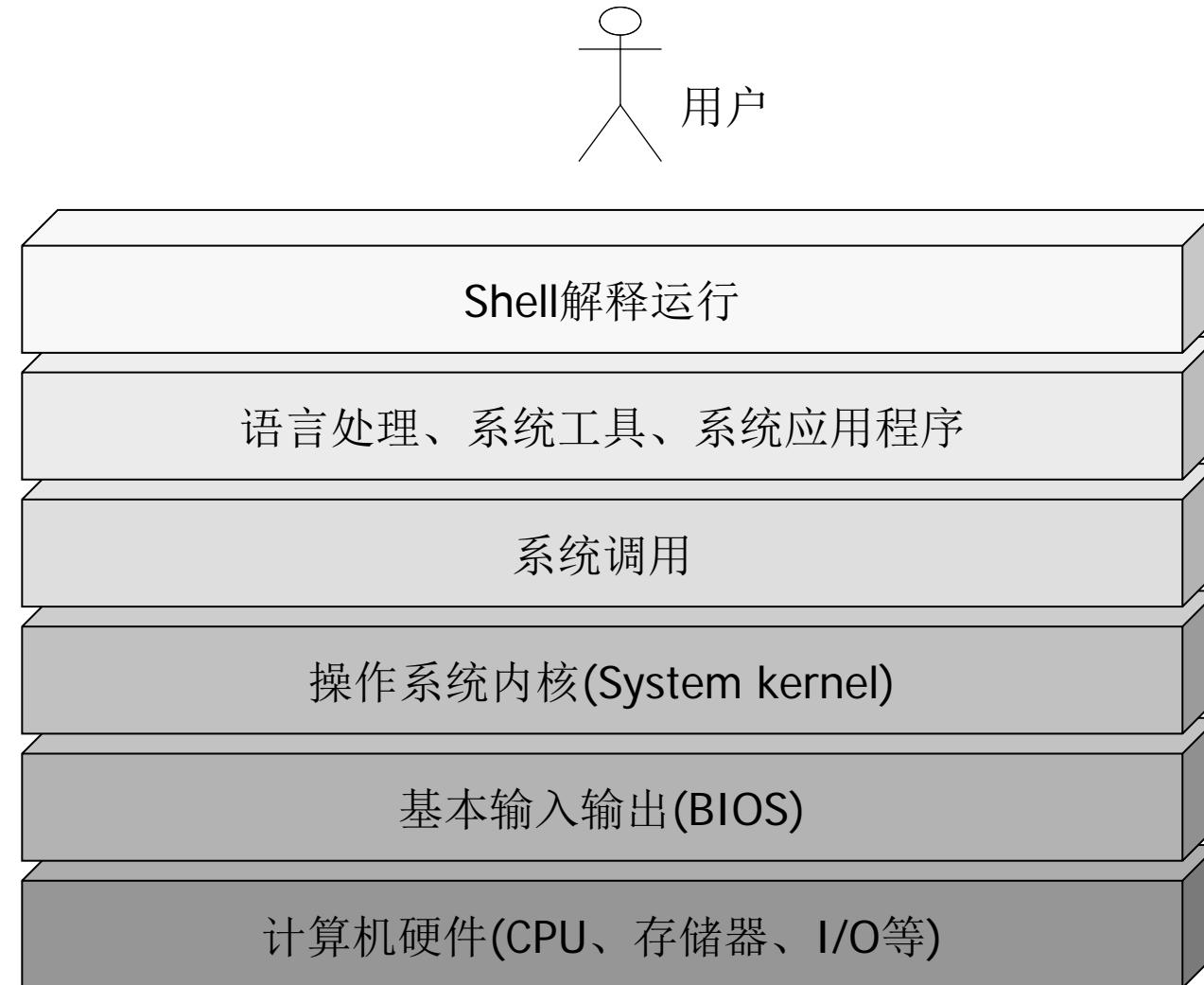
- 系统被看作对象的集合，每个对象都有一个它自己的功能集合；
- 数据及作用在数据上的操作被封装成抽象数据类型(ADT)；
- 只通过接口与外界交互，内部的设计决策则被封装起来
 - 构件：类
 - 连接件：类之间通过函数调用、消息传递实现交互



5. 层次结构

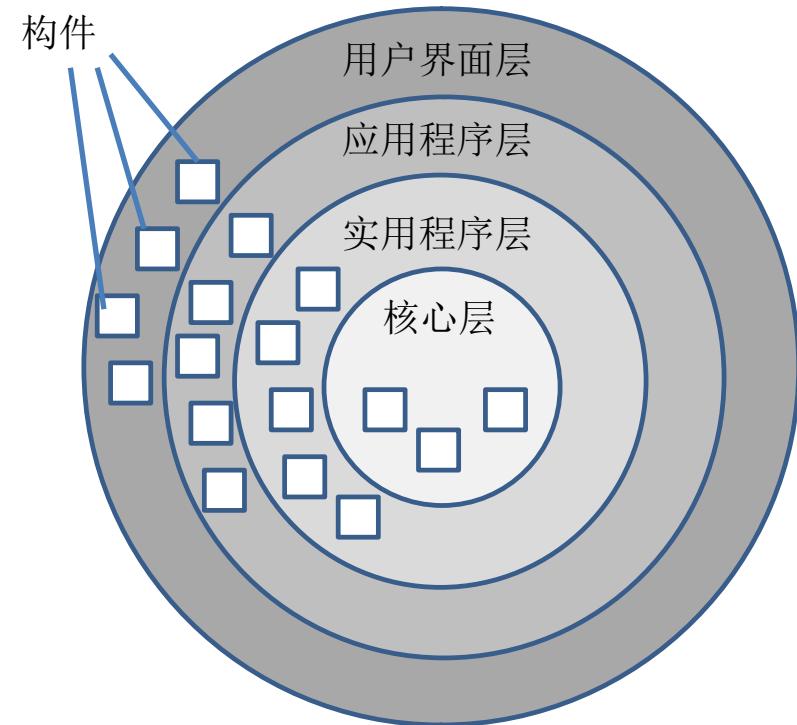
- 高楼大厦必须从底层开始建设；
- 层次化早已经成为一种复杂系统设计的普遍性原则；
- 两个方面的原因：
 - 事物天生就是从简单的、基础的层次开始发生的；
 - 众多复杂软件设计的实践，大到操作系统，中到网络系统，小到一般应用，几乎都是以层次化结构建立起来的。

计算机操作系统的层次结构



层次系统

- 在层次系统中，系统被组织成若干个层次，每个层次由一系列构件组成；
- 层次之间存在接口，通过接口形成**call/return**的关系
 - 下层构件向上层构件提供服务
 - 上层构件被看作是下层构件的客户端

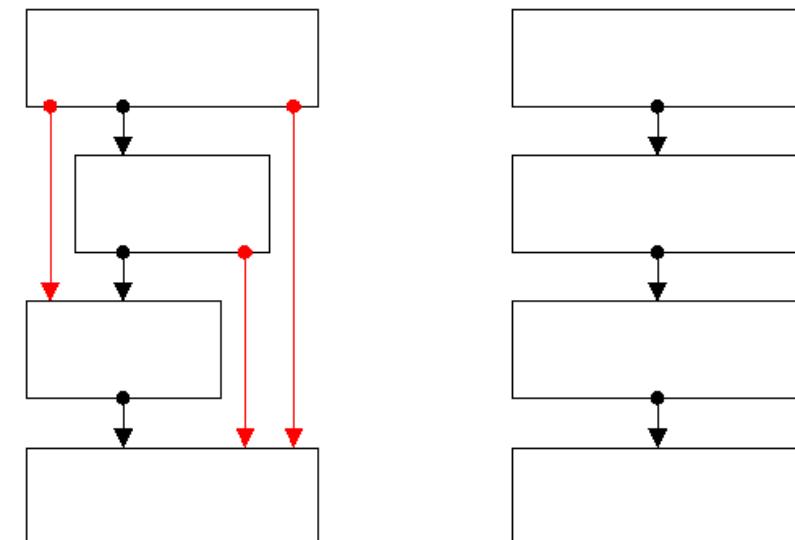


层次系统的优点

- 这种风格支持基于可增加抽象层的设计，允许将一个复杂问题分解成一个增量步骤序列的实现。
- 不同的层次处于不同的抽象级别：
 - 越靠近底层，抽象级别越高；
 - 越靠近顶层，抽象级别越低；
- 由于每一层最多只影响两层，同时只要给相邻层提供相同的接口，允许每层用不同的方法实现，同样为软件复用提供了强大的支持。

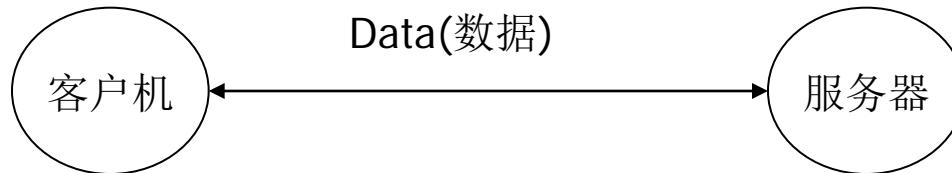
严格分层和松散分层

- 严格分层系统要求严格遵循分层原则，限制一层中的构件只能与对等实体以及与它紧邻的下面一层进行交互
 - 优点：修改时的简单性
 - 缺点：效率低下
- 松散的分层应用程序放宽了此限制，它允许构件与位于它下面的任意层中的组件进行交互
 - 优点：效率高
 - 缺点：修改时困难



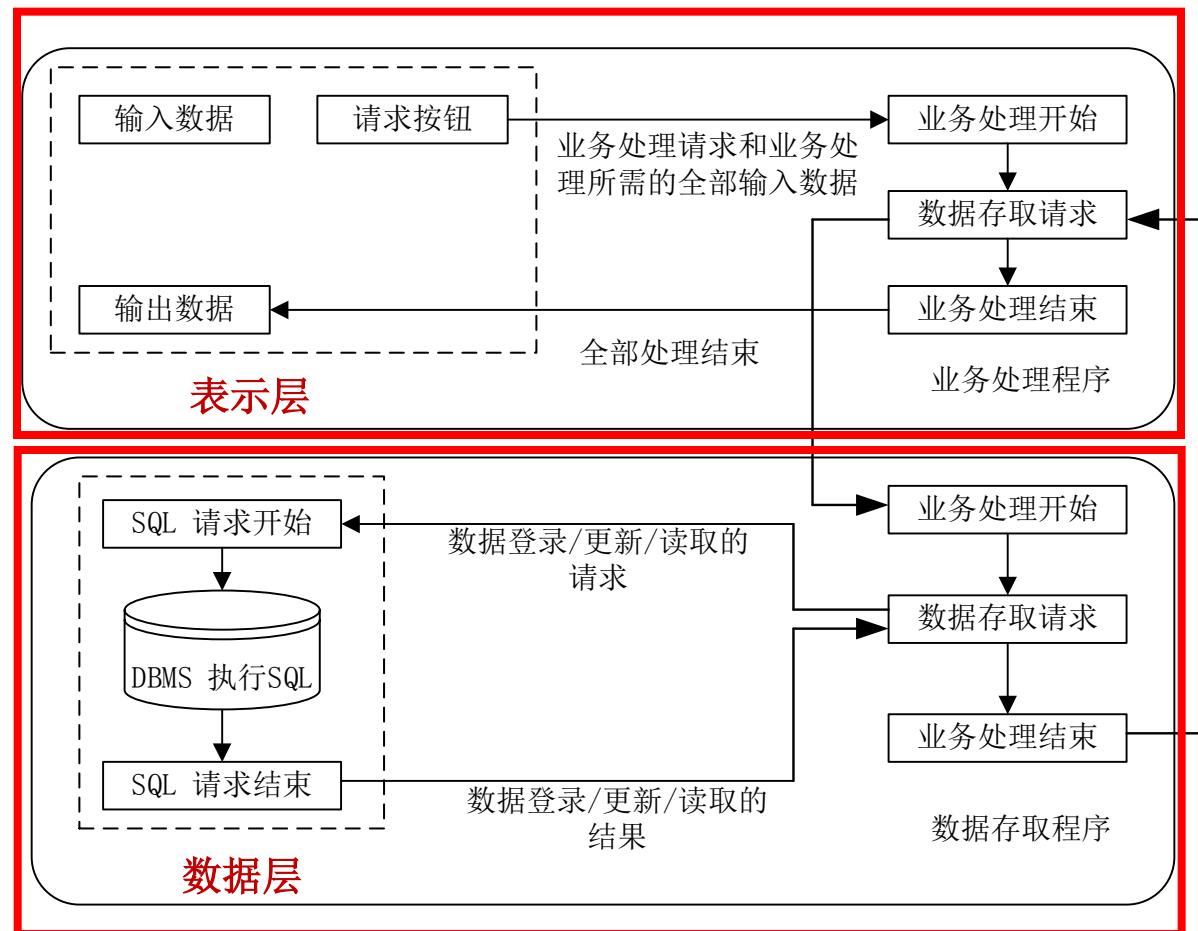
“客户机-服务器”体系结构

- 客户机/服务器(**Client/Server, C/S**)：一个应用系统被分为两个逻辑上分离的部分，每一部分充当不同的角色、完成不同的功能，多台计算机共同完成统一的任务。
 - 客户机(前端，front-end)：用户交互、业务逻辑、与服务器通讯的接口；
 - 服务器(后端，back-end)：与客户机通讯的接口、业务逻辑、数据管理。
- 一般的，
 - 客户机为完成特定的工作向服务器发出请求；
 - 服务器处理客户机的请求并返回结果。



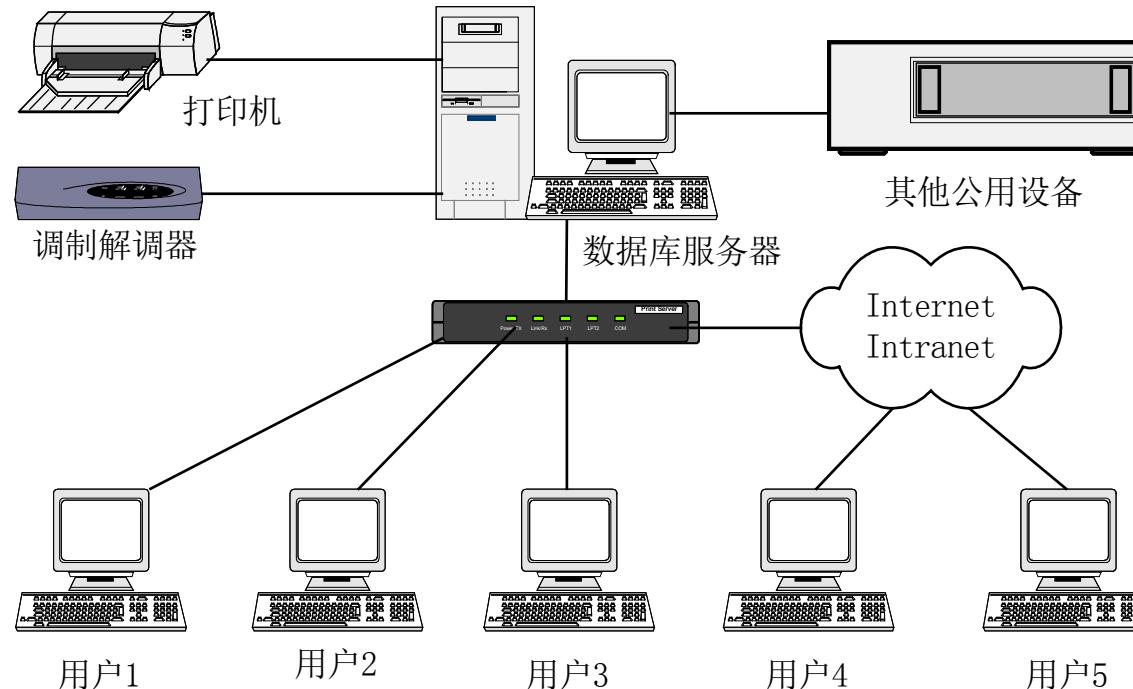
一个例子：客户端/服务器结构(C/S)

- 软件实体之间的关系：



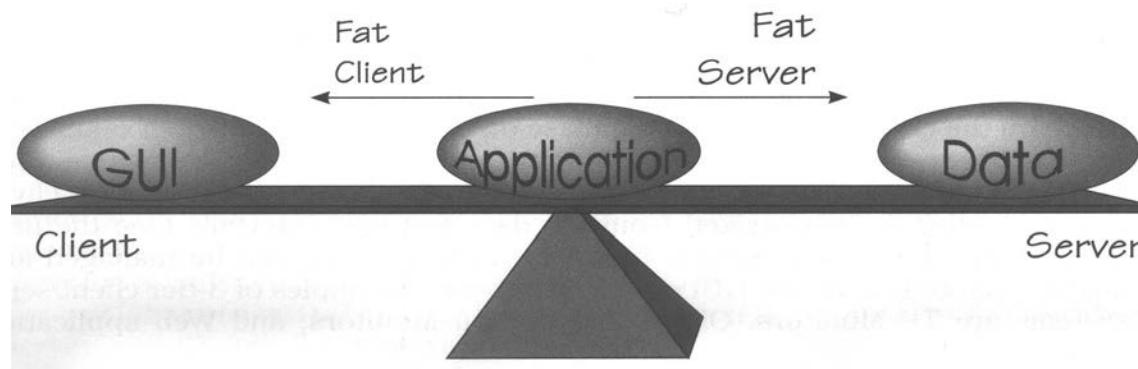
客户端/服务器结构(C/S)

- 硬件和网络环境之间的关系:



胖客户端与瘦客户端

- 业务逻辑的划分比重：在客户端多一些还是在服务器端多一些？
 - 胖客户端：客户端执行大部分的数据处理操作
 - 瘦客户端：客户端具有很少或没有业务逻辑

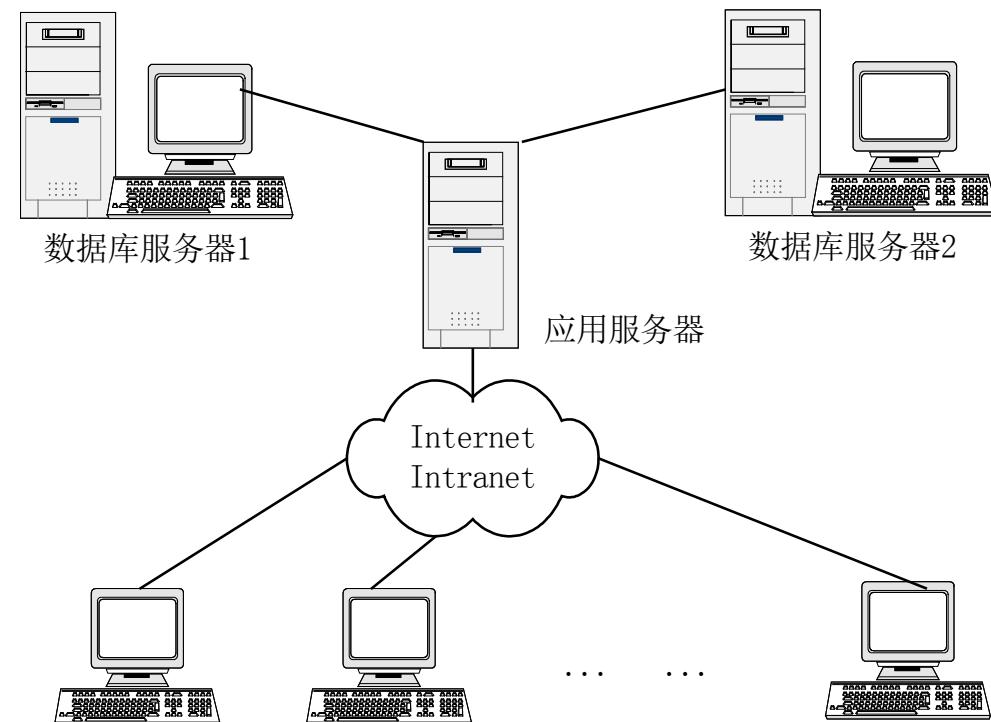


胖瘦客户端存在的问题？

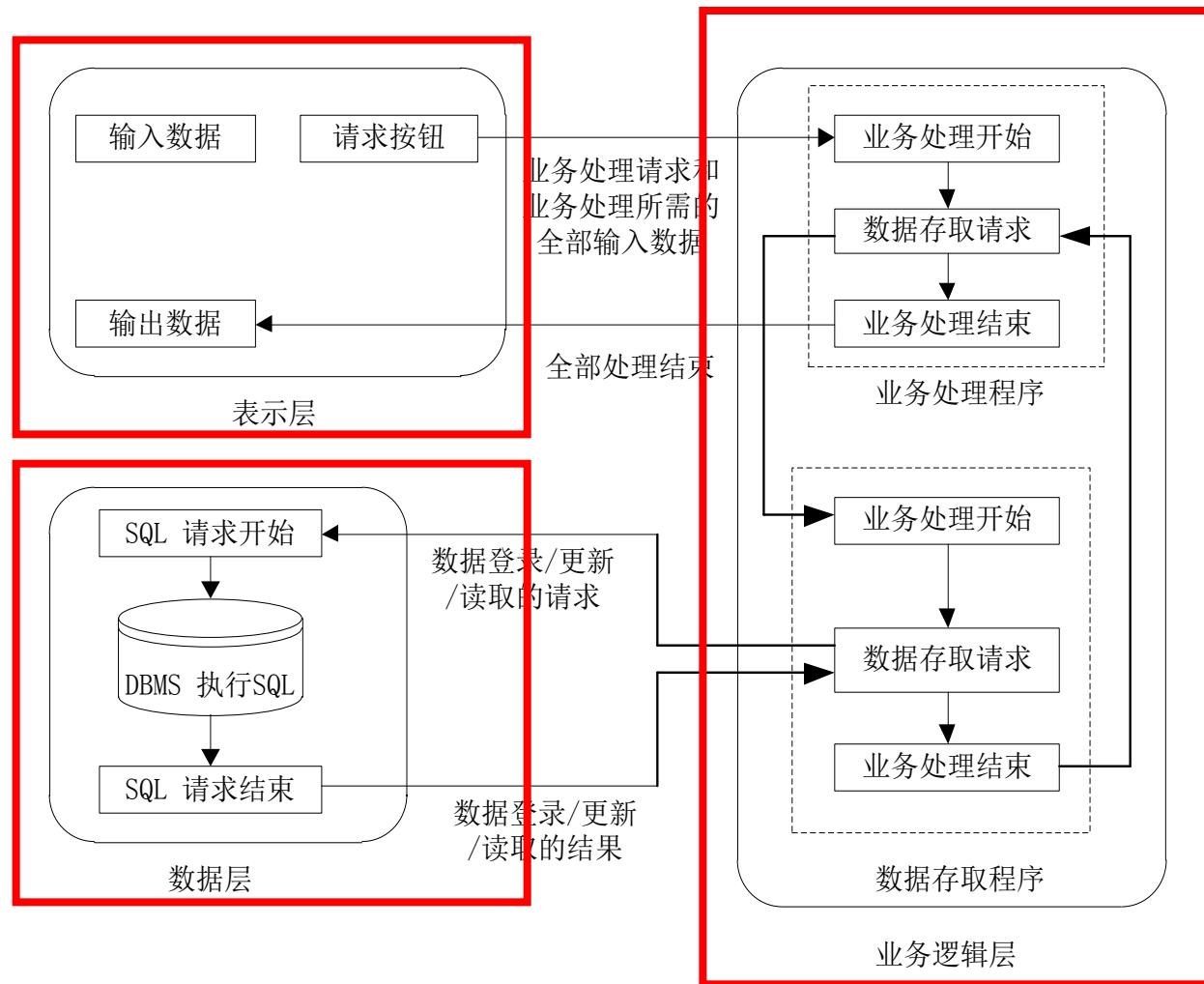
**RIA (Rich Internet Application)
—富客户端**

三层C/S体系结构

- 在客户端与数据库服务器之间增加了一个中间层
 - 表示层：用户界面—界面设计
 - 业务逻辑层：业务处理—程序设计
 - 数据层：数据存储—数据库设计



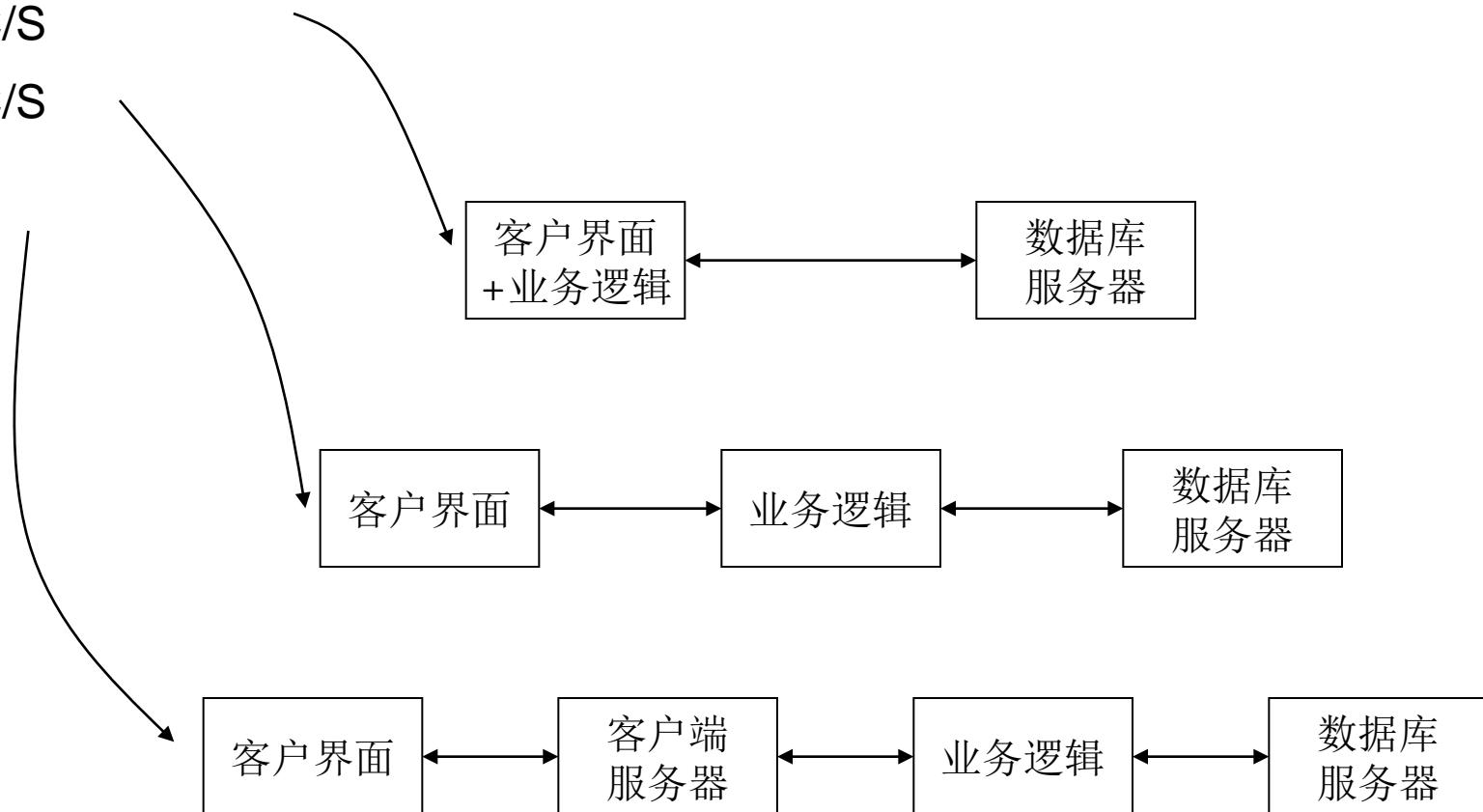
三层C/S结构



客户机/服务器的层次性

- “客户机-服务器”结构的发展历程：

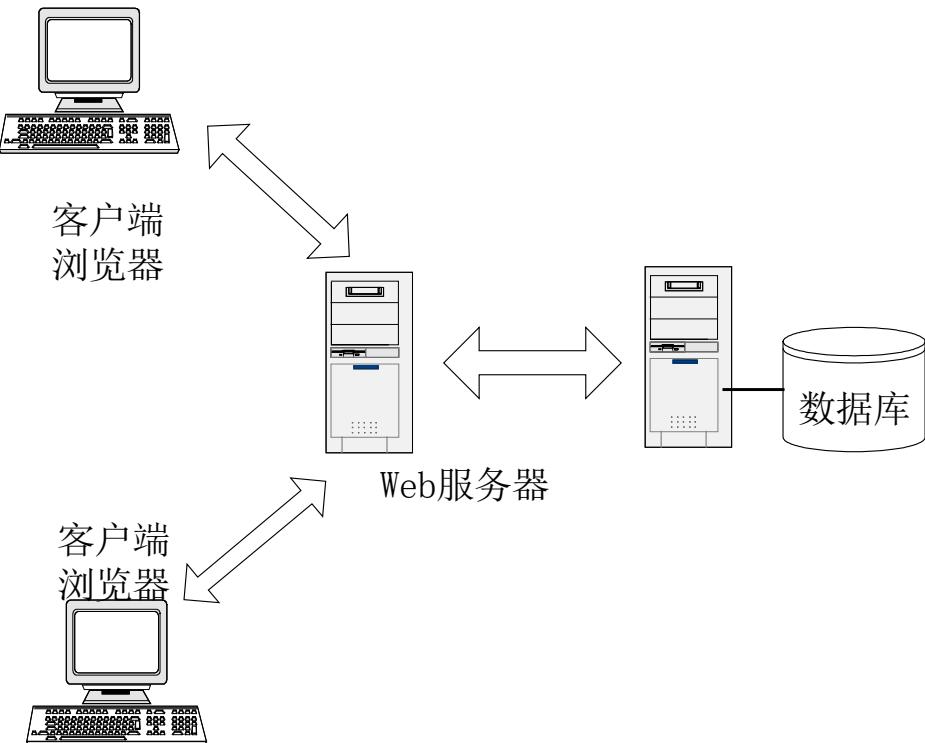
- 两层C/S（仓库体系风格）
- 三层C/S
- 多层C/S



B/S结构

- 浏览器/服务器(B/S)是四层C/S风格的一种实现方式。

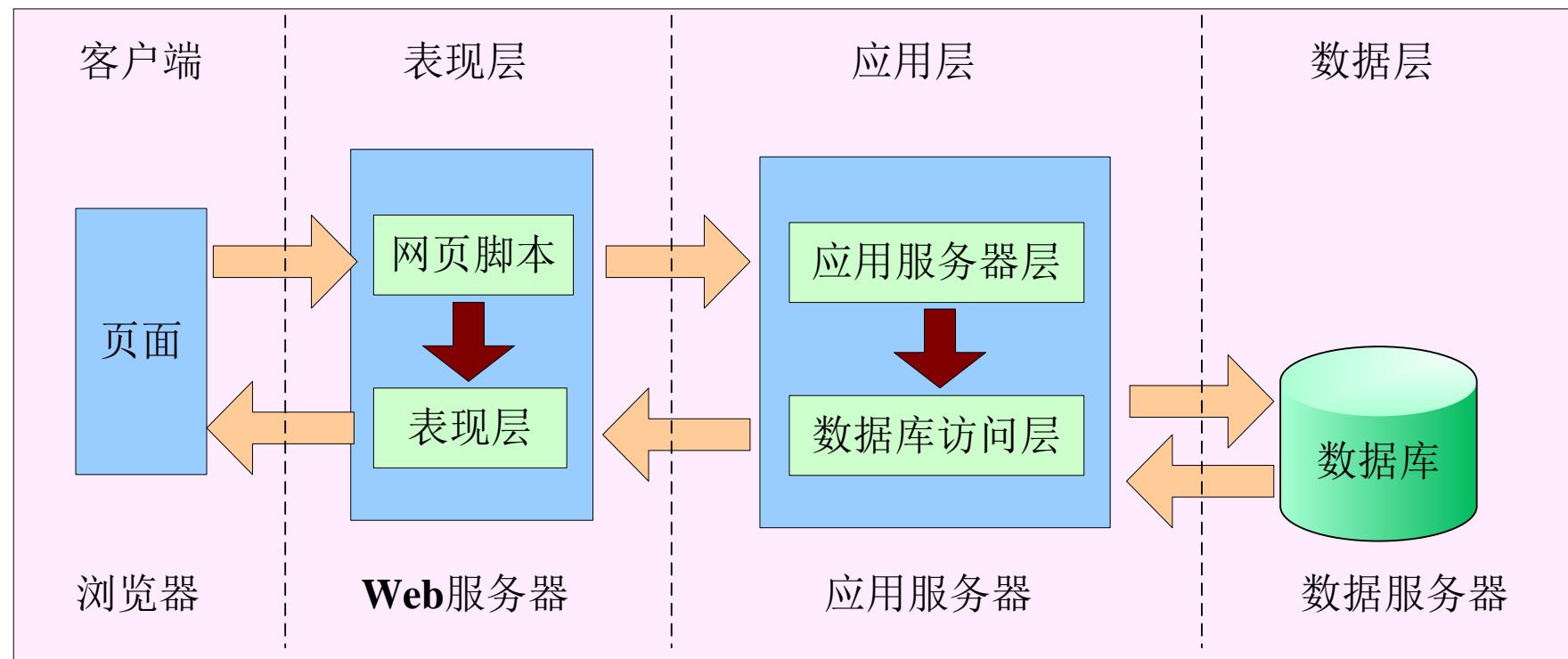
- 表现层：浏览器
- 逻辑层：
 - Web服务器
 - 应用服务器
- 数据层：数据库服务器



- B/S与四层C/S结构的区别：

- C/S：表现层仍部署在客户端；
- B/S：客户端除了浏览器之外无任何程序需要部署。

B/S结构

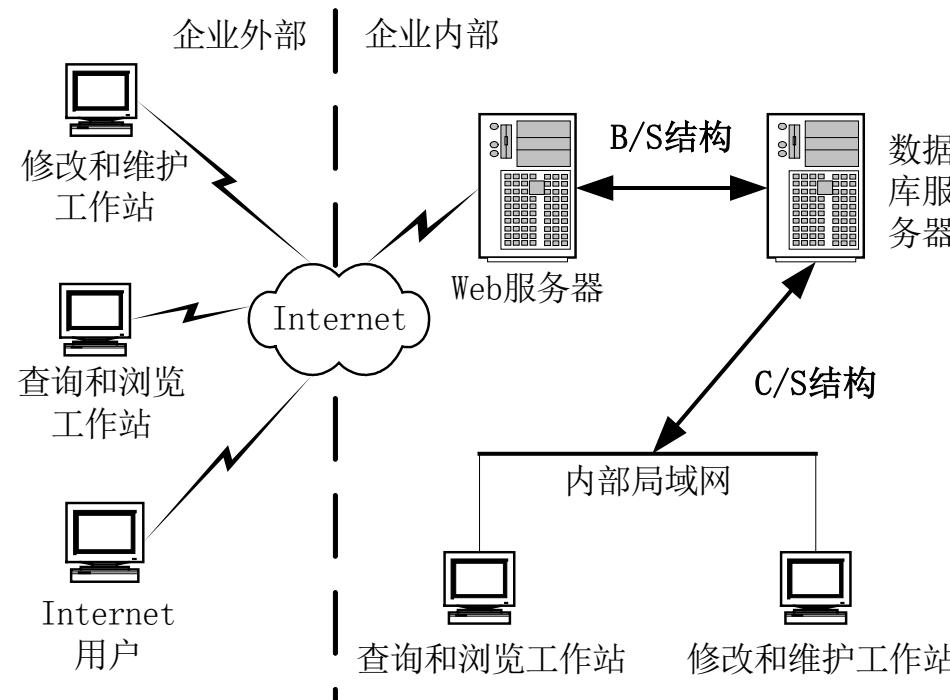


B/S结构

- 基于**B/S**体系结构的软件，系统安装、修改和维护全在服务器端解决，**系统维护成本低**：
 - 客户端无任何业务逻辑，用户在使用系统时，仅仅需要一个浏览器就可运行全部的模块，真正达到了“零客户端”的功能，很容易在运行时自动升级。
 - 良好的灵活性和可扩展性：对于环境和应用条件经常变动的情况，只要对业务逻辑层实施相应的改变，就能够达到目的。
- **B/S**成为真正意义上的“瘦客户端”，从而具备了很高的稳定性、延展性和执行效率。
- **B/S**将服务集中在一起管理，统一服务于客户端，从而具备了良好的容错能力和负载平衡能力。

C/S+B/S混合模式

- 为了克服**C/S**与**B/S**各自的缺点，发挥各自的优点，在实际应用中，通常将二者结合起来；
- 遵循“内外有别”的原则：
 - 企业内部用户通过局域网直接访问数据库服务器
 - C/S结构；
 - 交互性增强；
 - 数据查询与修改的响应速度高；
 - 企业外部用户通过Internet访问Web服务器/应用服务器
 - B/S结构；
 - 用户不直接访问数据，数据安全；



M/C结构

- 移动端/云端结构(**Mobile/Cloud**):
 - 客户端不是传统的客户机，而是各类移动终端设备，如智能手机、平板、智能家电、可穿戴设备等。
 - 服务端也不是传统的服务器，而是扩展到云环境下，支持高可伸缩性、按需付费等特性。
- 可以看作是**C/S**结构的扩展。
- 优势：移动，可以做到**anytime & anywhere**使用软件的功能。
- 客户端程序的体现形式：各类**App**

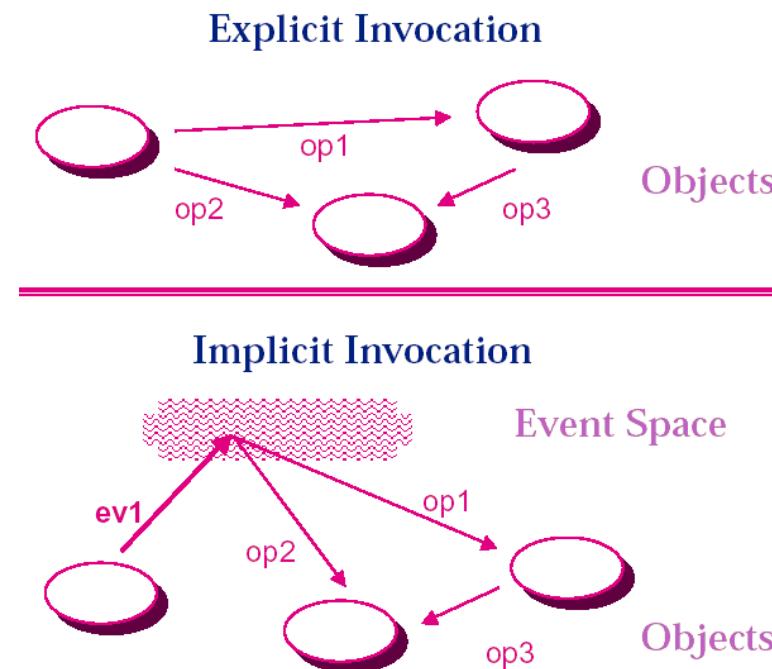
6. 显式调用 vs. 隐式调用

- 显式调用:

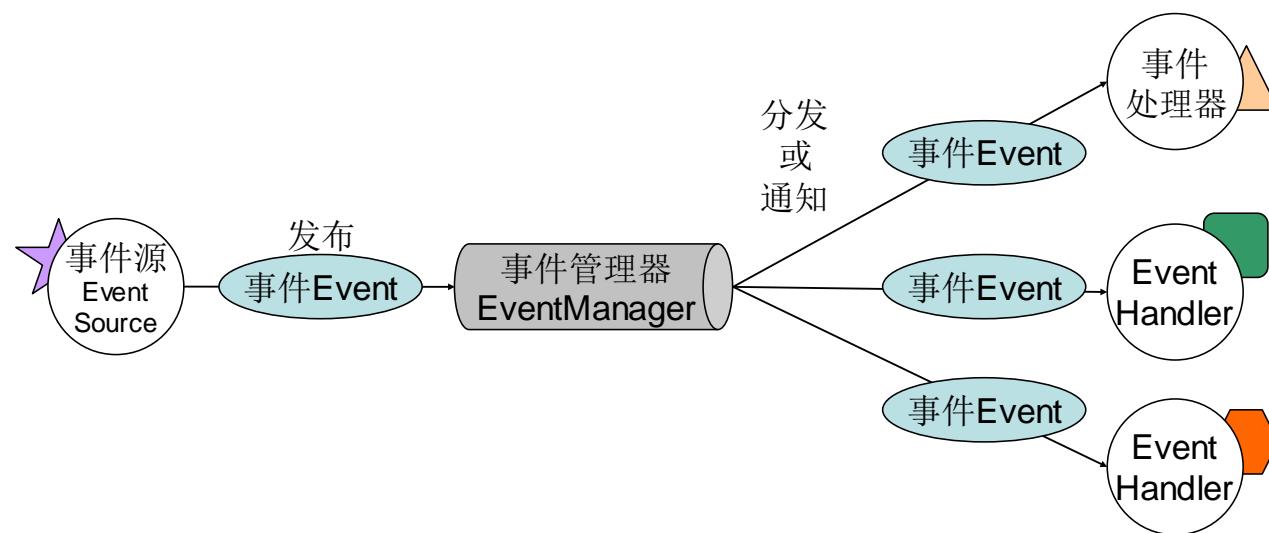
- 各个构件之间的互动是由显性调用函数或程序完成的。
- 调用过程与次序是固定的、预先设定的。

- 隐式调用:

- 调用过程与次序不是固定的、预先未知；
- 各构件之间通过事件的方式进行交互；

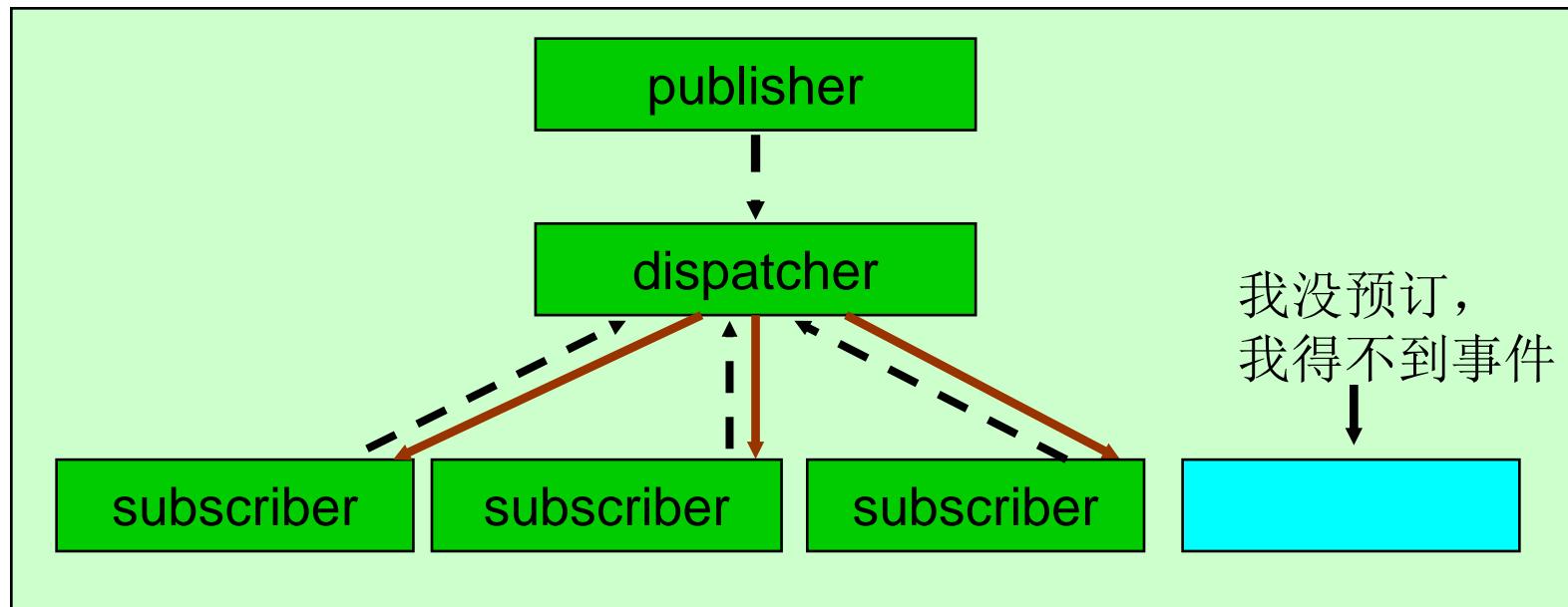


事件系统的基本构成



功能	描述
分离的交互	事件发布者并不会意识到事件订阅者的存在。
多对多通信	采用发布/订阅消息传递，一个特定事件可以影响多个订阅者。
基于事件的触发器	控制流由接收者确定（基于发布的事件）。
异步	通过事件消息传递支持异步操作。

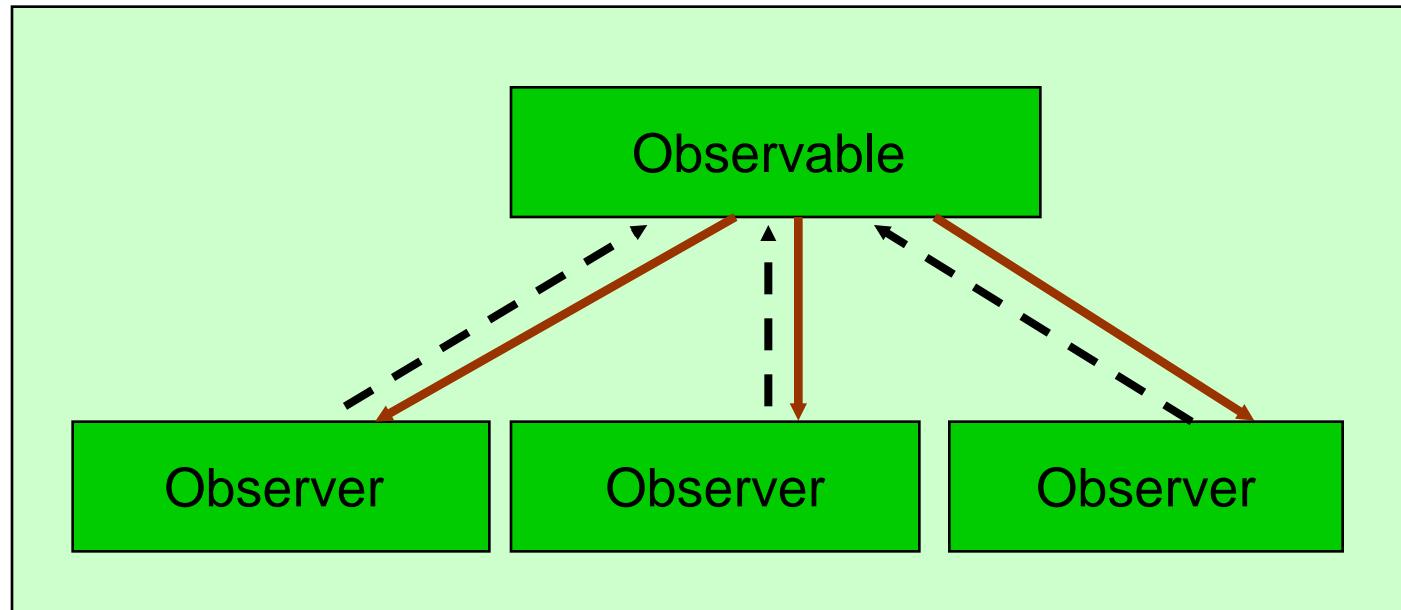
事件风格的实现策略之一：选择广播式



有目的广播，
只发送给那些已经注册过的订阅者

— · — · — · → Publish event
..... → Subscribe
—→ Send event

事件风格的实现策略之二：观察者模式



Observable/Observer module

Legend: → Register event → Send event

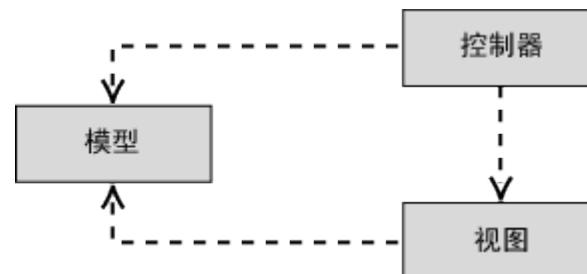
问题

- ——用户界面需要频繁的修改，它是“不稳定”的。
- ——业务逻辑/数据 与 用户界面 之间应尽量少的避免直接通信。
- 问题：如何让 Web 应用程序的用户界面与业务逻辑功能实现模块化，以便使程序开发人员可以轻松地单独修改各个部分而不影响其他部分？

7 模型-视图-控制器 (MVC)

■ Model-View-Controller (MVC):

- **模型(Model, M):** 用于管理应用系统的行为和数据，并响应为获取其状态信息(通常来自视图)而发出的请求，还会响应更改状态的指令(通常来自控制器); ——对应于传统B/S中的业务逻辑和数据
- **视图(View, V):** 用于管理数据的显示; ——对应于传统B/S中的用户界面
- **控制器(Controller, C):** 用于解释用户的鼠标和键盘输入，以通知模型和视图进行相应的更改。 ——在传统B/S结构中新增的元素

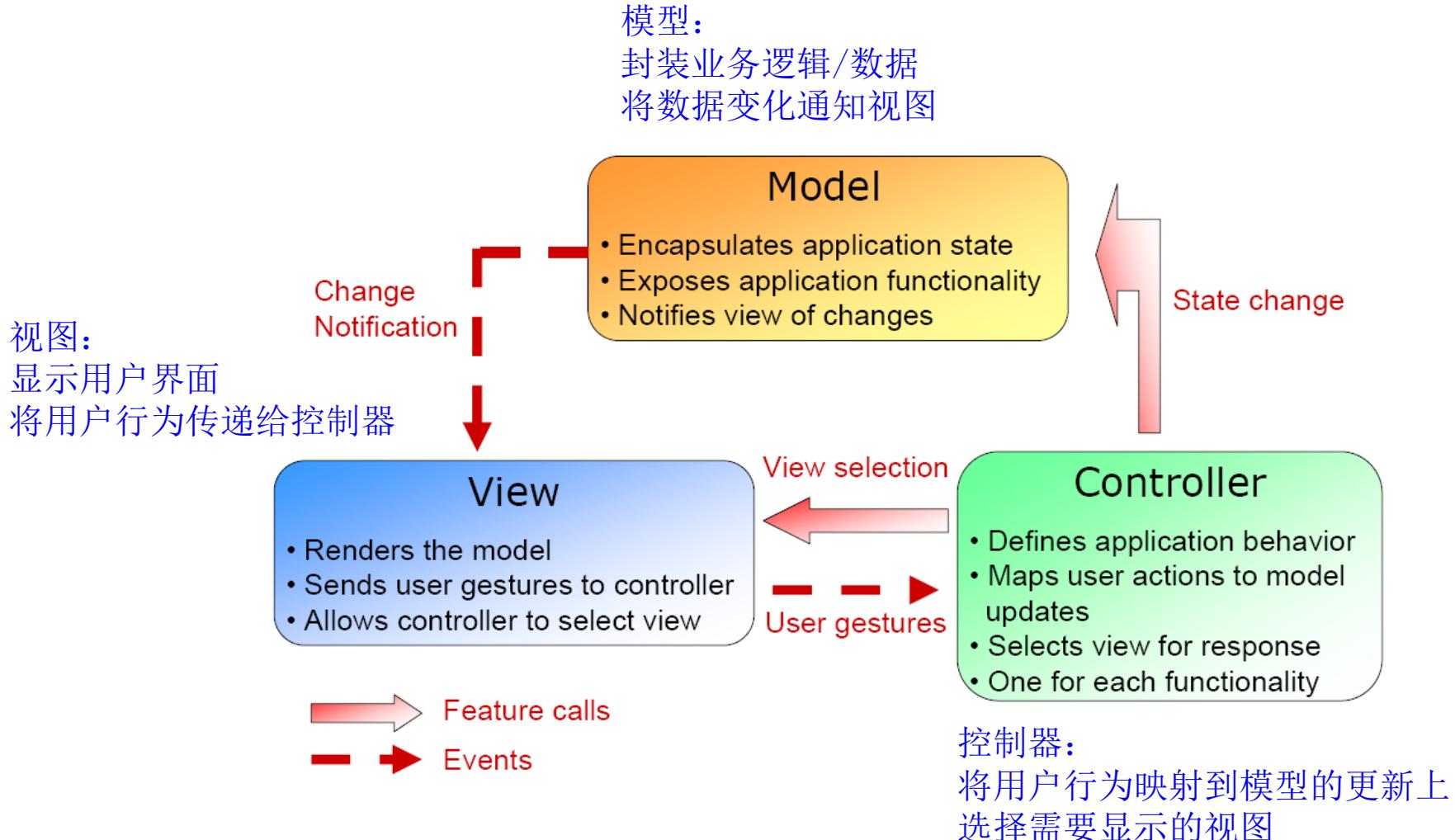


解决方案：Model-View-Controller (MVC)



- MVC是一种软件体系结构，它将应用程序的数据模型 / 业务逻辑、用户界面分别放在独立的构件中，从而对用户界面的修改不会对数据模型/业务逻辑造成很大影响。

MVC运行机制



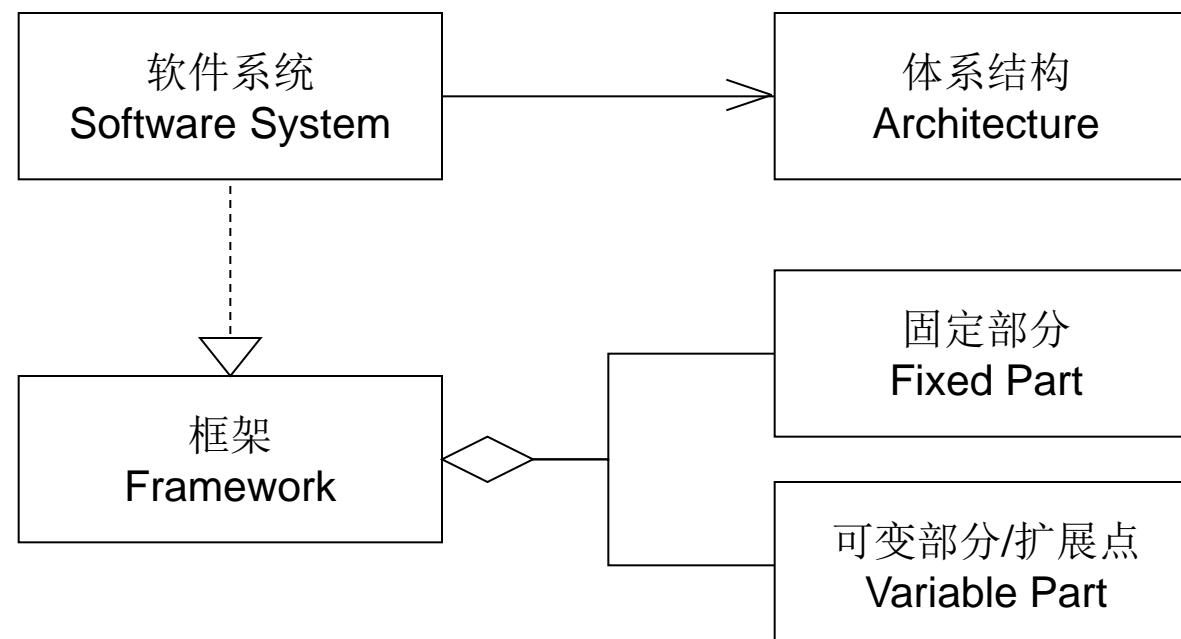
MVC各层次的实现技术



- 针对不同层次，采用不同的实现技术：
 - 用户界面层：HTML/JavaScript/CSS、jQuery、JSP、AJAX、Flex、HTML5、Dojo、Bootstrap、Node.js...
 - 控制层：PHP、Python、Servlet、Ruby、...
 - 业务逻辑层：JavaBean、Pojo、...
 - 持久化层：JDBC、JDO、Hibernate、iBatis、...
- **Struts、Django、CI、Rails**等以不同的编程语言(**Java、Python、PHP、Ruby**)分别实现了这一架构，提供了一个半成品，帮助开发人员迅速地开发符合**MVC**架构的应用程序，它们都是“框架**Framework**”。

Framework vs Architecture (框架和体系结构)

- 框架(**Framework**)：可实例化的、部分完成的软件系统或子系统，它为一组系统或子系统定义了统一的体系结构(**architecture**)，并提供了构造系统的基本构造块(**building blocks**)，还为实现具体功能定义了扩展点(**extending points**)。
- 框架实现了体系结构级别的复用。

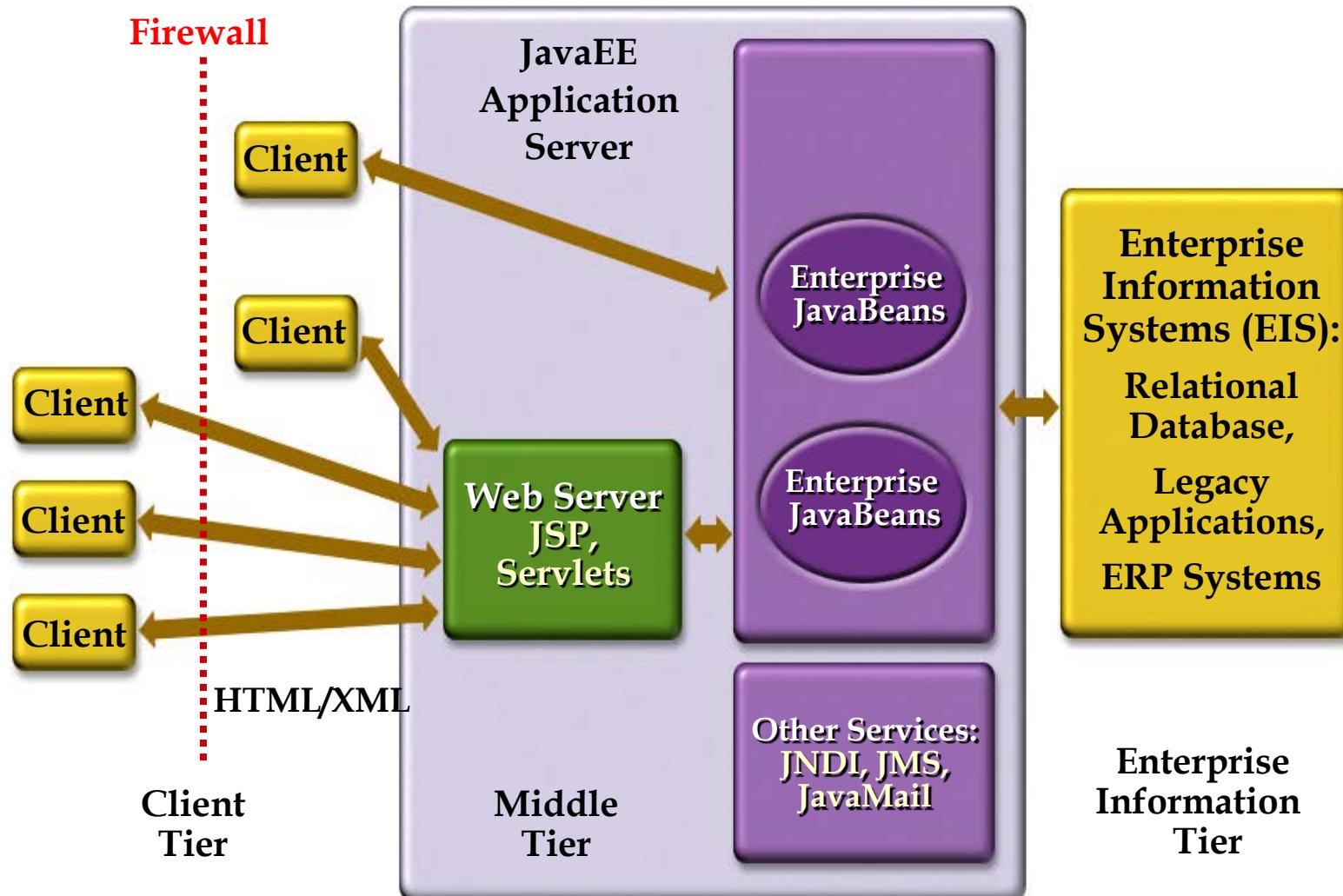


MVC Framework

- Java: Struts、Spring
- Python: Django、Pylons、Tornado
- PHP: CodeIgniter、Zend、CakePHP、ThinkPHP、Yii
- Ruby: Rails、Sinatra
- [http://en.wikipedia.org/wiki/Comparison
of web application frameworks](http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks)
- 每个框架均覆盖了**M**、**V**、**C**、**Persistence**四部分，提供了各类基础库的支持。

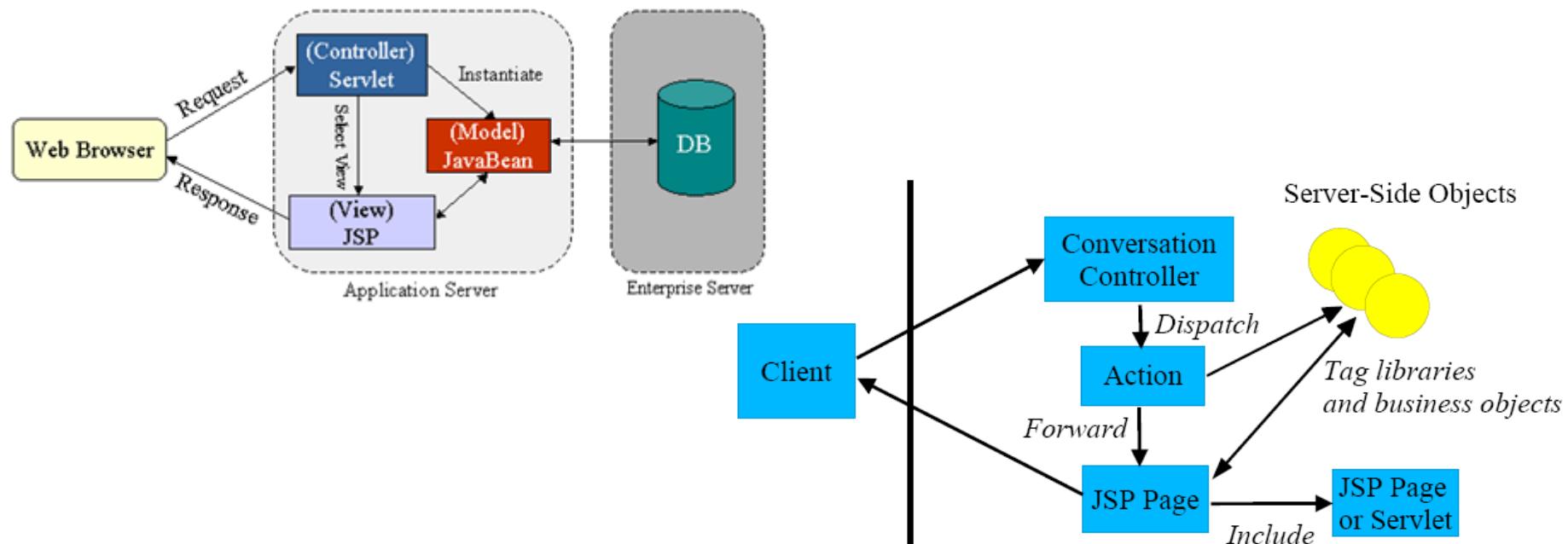


以JavaEE和Struts为例



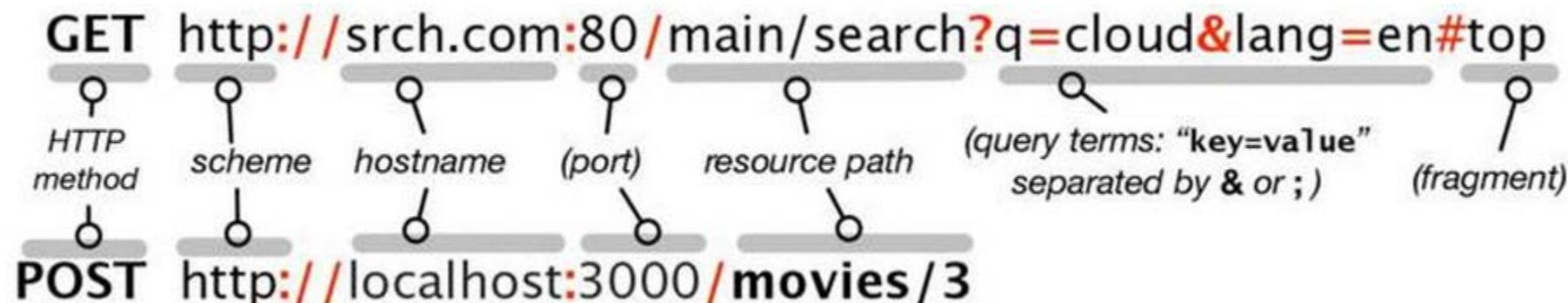
JavaEE MVC Model

- **Servlet**作为控制器，负责处理用户请求，并将其转发给扮演**Model**角色的**JavaBean**，而**JSP**作为纯粹的**View**；
- **Servlet**还决定在处理完该请求之后，将向用户显示哪个**JSP**页面(**View**)以显示处理结果；
- **JSP**页面获取**Servlet**的处理结果并动态显示给用户。



前端请求的URI (Uniform Resource Identifier)

- 用户通过浏览器发出的请求，通过HTTP协议传递到服务器端。
- 具体形式即为URI



FilterDispatcher: 前端控制器/调度器

- 前端请求的**URI**中包含了一个**resource path**, 代表着用户的请求。
- 通过**HTTPRequest**发送至**MVC**的**front controller**, 它相当于服务端的入口、总调度。——在**Struts**中, 实现为**FilterDispatcher**。
- **FilterDispatcher**接收到请求之后, 根据配置文件将请求转发到具体执行动作的**Model (Struts中称之为action)**。
- 配置文件**struts.xml**:

```
<struts>
    <package name="01" extends="struts-default">
        <action name="submit" class="SubmitAction" method="submit">
            <param name="param1">value1</param>
            <result>result.jsp</result>
        </action>
    </package>
</struts>
```

Action(model)

```
public class SubmitAction extends ActionSupport {  
    private String param1;  
    public String execute() throws Exception{  
        ...  
    }  
    public void submit(String param1){  
        ...  
    }  
}
```

```
<s:form action="submit" method="true">  
    <s:textfield label="Message" name="Param1" />  
    <s:submit value="submit" />  
</s:form>
```

主要内容



- **4.1 软件设计**
- **4.2 体系结构设计**
 - 4.2.1 体系结构的背景与定义
 - 4.2.2 体系结构的基本概念
 - 4.2.3 体系结构风格
 - 4.2.4 体系结构设计
- **4.3 界面设计**

面向对象的设计的两个阶段



- **系统设计(System Design)**

- 相当于概要设计(即设计系统的体系结构)；
 - 选择解决问题的基本途径；
 - 决策整个系统的结构与风格；

- **对象设计(Object Design)**

- 相当于详细设计(即设计对象内部的具体实现)；
 - 细化需求分析模型和系统体系结构设计模型；
 - 识别新的对象；
 - 在系统所需的应用对象与可复用的商业构件之间建立关联；
 - 识别系统中的应用对象；
 - 调整已有的构件；
 - 给出每个子系统/类的精确规格说明。

系统设计概述

- 设计系统的体系结构

- 选择合适的分层体系结构策略，建立系统的总体结构：分几层？每层的功能分别是什么？

- 定义数据的存储策略

- 识别设计元素

- 识别“包”(package)、“子系统”(sub-system)

- 部署子系统

- 选择硬件配置和系统平台，将子系统分配到相应的物理节点，绘制部署图(deployment diagram)

主要内容



- 4.1 软件设计
- 4.2 体系结构设计
- 4.3 界面设计
 - 4.3.1 用户界面概述
 - 4.3.2 用户界面的设计原则
 - 4.3.3 界面分析和设计

1. 何谓“用户界面”？



河南省突发环境事件应急指挥系统

来自应急办的admin 欢迎您登陆
2011年9月1日 星期一 您当前处理的事件是：苯泄漏

应急值守 新增 查看

1. 苯泄漏	2010-09-13	化工厂爆
2. 化工厂发生...	2010-09-29	化工厂爆
3. 化工厂发生...	2010-09-29	化工厂爆
4. 赤泥库漫坝	2011-06-14	荥阳市高
5. 高速公路危...	2011-07-02	课河

应急知识

- 危化品处置知识
- 突发环境事件信息报告办法
- 国家危险废物名录
- 中华人民共和国环境保护法
- 中华人民共和国水污染防治法
- 工业污染源现场检查技术规范
- 环境影响评价技术导则 地下水环境
- 危险化学品安全管理条例
- 淮河流域水污染防治暂行条例
- 江西九江地震
- 舟曲泥石流
- 松花江水污染带给我们的重大启示

应急预案

- 国家突发公共事件总体应急预案.mht
- 国家突发环境事件应急预案.mht
- 河南省突发环境事件应急预案.pdf
- 上街区环境保护局突发环境污染防治应急预案.docx
- 环境污染事故应急预案(定稿).rar

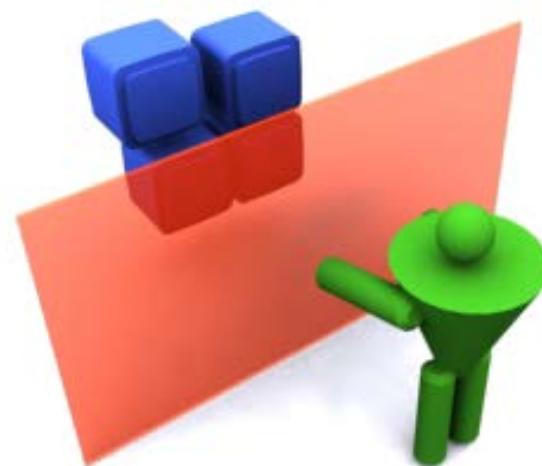
公告

- 环保部环境工程评估中心莅安指导工作 2011年6月26日
- 国家环保部领导莅临 2011年6月26日
- 临安调研指导环保工作 2011年6月26日
- 省、市环保部门对固始县水源水质状况进行调研 2011年6月26日

何谓“用户界面”？

- **用户界面(User Interface, UI):** 使用者(人)与软件/硬件设备(计算机)之间搭建的一种沟通交流的手段和媒介。
- **UI用以支持二者之间的有效交互(interaction):**
 - 输入界面(input): 使用者输入特定指令或信息，以访问软件/硬件所提供的功能；
 - 输出界面(output): 软件/硬件向使用者展示反馈信息。

☞ 概括： UI就是人和工具之间的界面

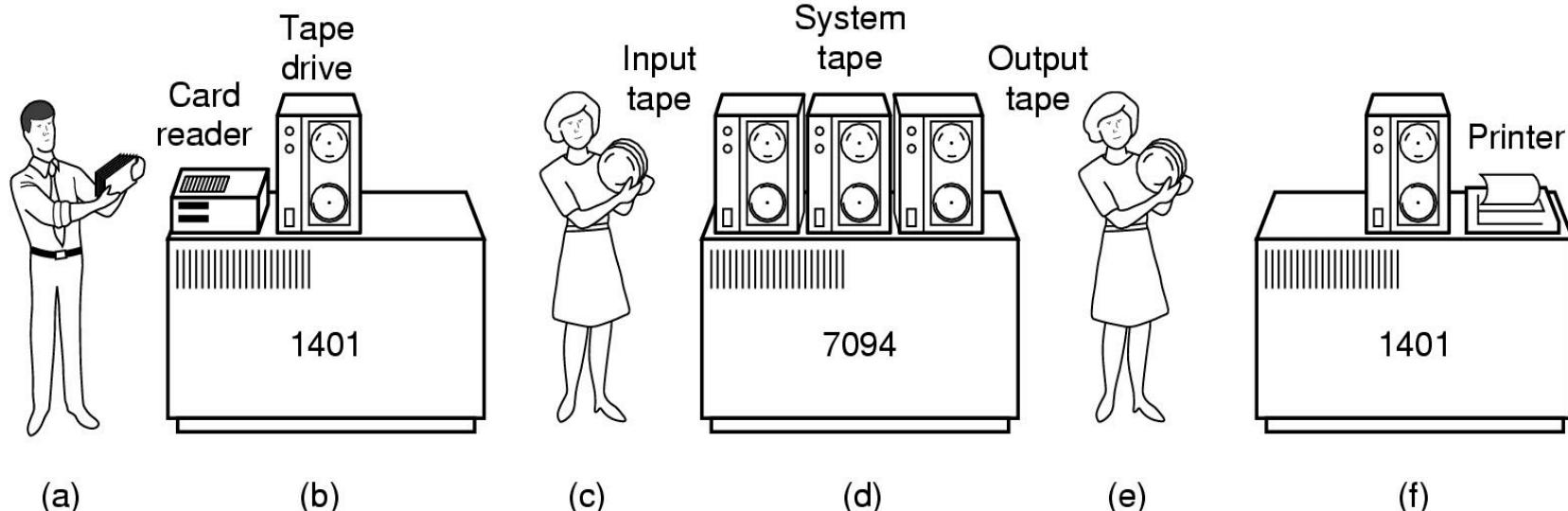


界面设计

- 界面设计是人与机器之间传递和交换信息的媒介，包括硬件界面和软件界面，是计算机科学与心理学、设计艺术学、认知科学和人机工程学的交叉研究领域。

2. 软件界面的典型类型

- 批处理界面(Batch interfaces): 无任何交互的UI，用户一次输入所有信息，系统接收之后开始计算，将输出结果统一提供给用户。一旦计算开始，将不再接受任何其他输入。



将用户输入的纸带上的
数据写入磁带

将磁带作为计算设备的输入，
进行计算，得到输出结果

打印计算结果

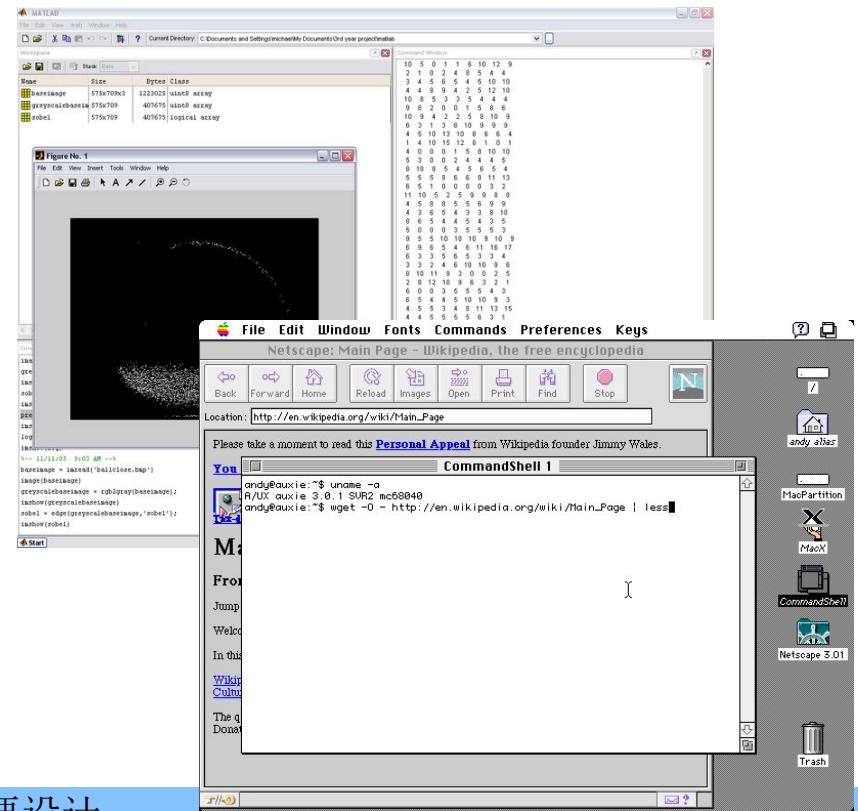
软件界面的典型类型

- 命令行式用户界面(Command line interfaces, CLI): 用户通过键盘输入字符型的命令串, 系统通过命令解释器理解用户指令, 进而将输出结果打印到屏幕上。

```
bash-2.05b$ cat metadata.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pkgmetadata SYSTEM "http://www.gentoo.org/dtd/metadata.dtd">
</pkgmetadata>
<herd>base-system</herd>
</pkgmetadata>
bash-2.05b$ sudo /etc/init.d/bluetooth status
Password:
* status: stopped
bash-2.05b$ ping -q -c1 en.wikipedia.org
PING rr.cktpa.wikimedia.org (207.142.131.247) 56(84) bytes of data.
--- rr.cktpa.wikimedia.org ping statistics ---
1 packets transmitted. 1 received. 0% packet loss. time 0ms
rtt min/avg/max/mdev = 112.076/112.076/112.076/0.000 ns
bash-2.05b$ grep -i /dev/sda /etc/fstab | cut --fields=-3
/dev/sda1          /mnt/usbkey
/dev/sda2          /mnt/ipod
bash-2.05b$ date
Wed May 25 11:36:56 PDT 2005
bash-2.05b$ lsmod    Displays a list of files and subdirectories in a directory.
Module
joudev
ipw200
iieee80211
iee80211_crypt
e1000
bash-2.05b$ [REDACTED]
DIR [drive:][path][filename] [/P] [/W] [/A[[:lattributes]]] [/O[[:sortord]]]
        [/S] [/B] [/L] [/C[H]]
[drive:][path][filename] Specifies drive, directory, and/or files to list.
/P    Pauses after each screenful of information.
/W    Uses wide list format.
/A    Displays files with specified attributes.
    attributes D Directories R Read-only files H Hidden files
            S System files A Files ready to archive - Prefix meaning "not"
/O    List by files in sorted order.
    sortord N By name (alphabetic) S By size (smallest first)
            E By extension (alphabetic) D By date & time (earliest first)
            G Group directories first - Prefix to reverse order
            C By compression ratio (smallest first)
/S    Displays files in specified directory and all subdirectories.
/B    Uses bare format (no heading information or summary).
/L    Uses lowercase.
/C[H] Displays file compression ratio; /CH uses host allocation unit size.

Switches May be preset in the DIRCMD environment variable. Override
preset switches by prefixing any switch with - (hyphen)--for example, -W.

C:>_
```



软件界面的典型类型

- 图形化用户界面(**Graphical user interfaces, GUI**)：用户通过图标、控件与系统进行交互，使用文本标签或超链接等方式提供导航信息。用户可直接操作这些图形化元素以向系统发出指令。



Android



Windows



iPad

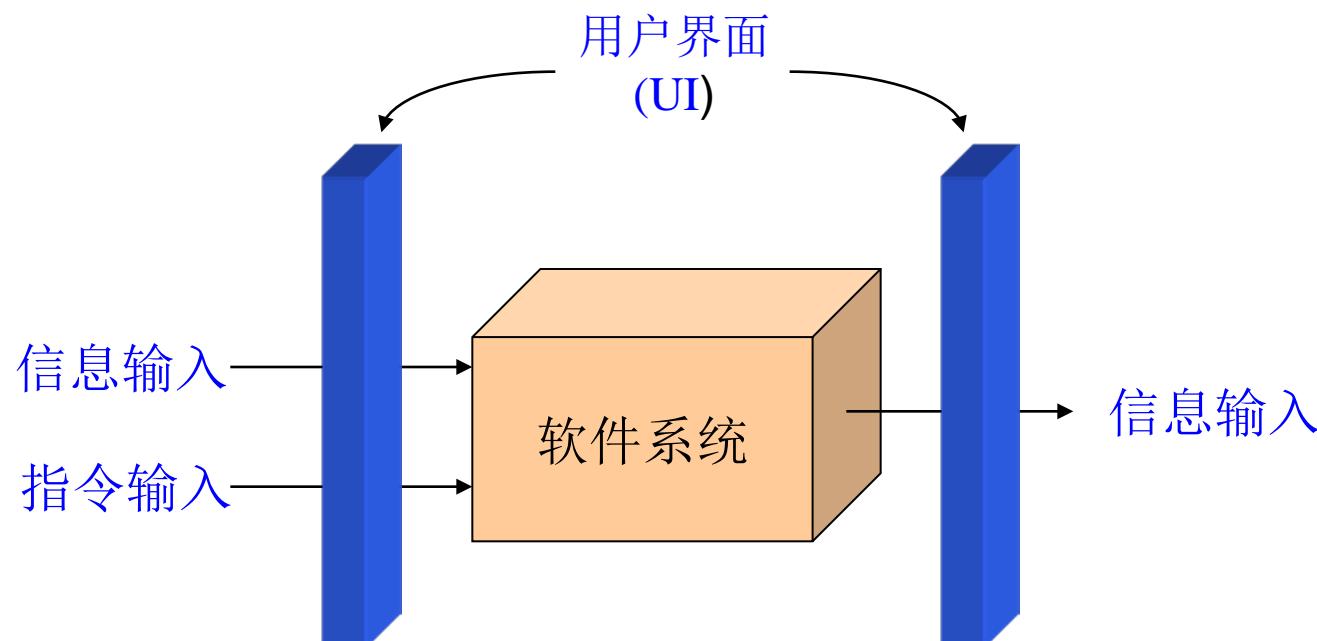
思考：下一代界面？

- 模式识别：语音输入
- 计算机视觉：手势输入
- 主动获取：情境感知（位置、状态等传感器）
- 人机互动：siri waston



3. 用户界面的组成部分

- 信息输入：如何输入数据？
- 指令输入：如何向系统发出行为指令？
- 信息输出：如何向用户显示最终产生的结果？



数据输入方式

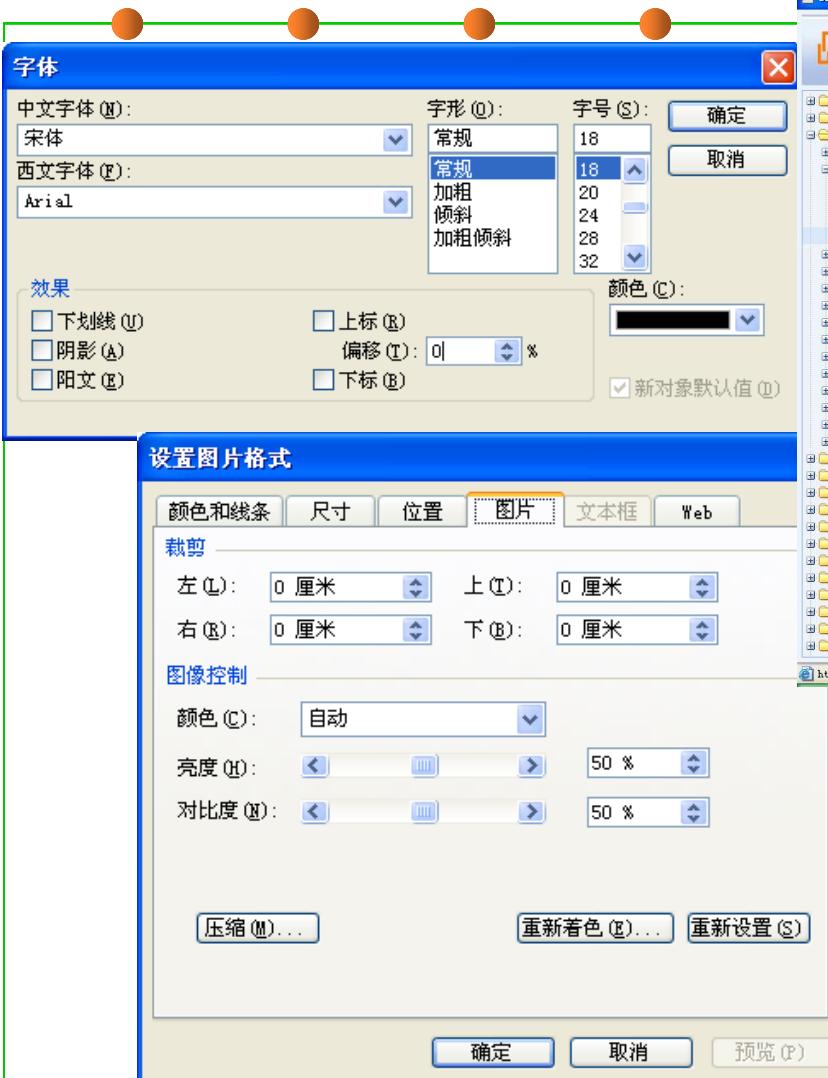
- **输入方式：**

- 键盘输入：文本编辑框、...
- 鼠标输入：列表框、下拉列表框、单选框、复选框、...
- 硬件接口输入：条码扫描、读卡器、触摸屏、摄像头、麦克风...
- 软件接口输入：读取文件、从数据库导入、RMI、WebService...

- **输入的连续性：**

- 单一输入；
- 连续输入；

数据输入方式



This screenshot displays two windows from a manufacturing management system. The top window is titled '采购发票' (Purchase Invoice) and shows a search form with fields like '发票编号' (Invoice Number), '制单人' (Bill Preparer), and '开票日期' (Issuance Date). The bottom window is titled '颜色配置' (Color Configuration) and shows a list of color codes with their corresponding RGB values and hex codes. A color palette is also visible.

颜色编码	颜色名称	红色	绿色	蓝色	标志	颜色预览
02	blue	0	0	255	1	Blue
03	green	0	255	0	1	Green
04	orange	255	200	0	1	Orange
05	黄色	128	0	0	1	Yellow
06	magenta	241	15	182	1	Magenta
07	cyan	200	204	255	1	Cyan

指令输入方式



■ 输入方式：

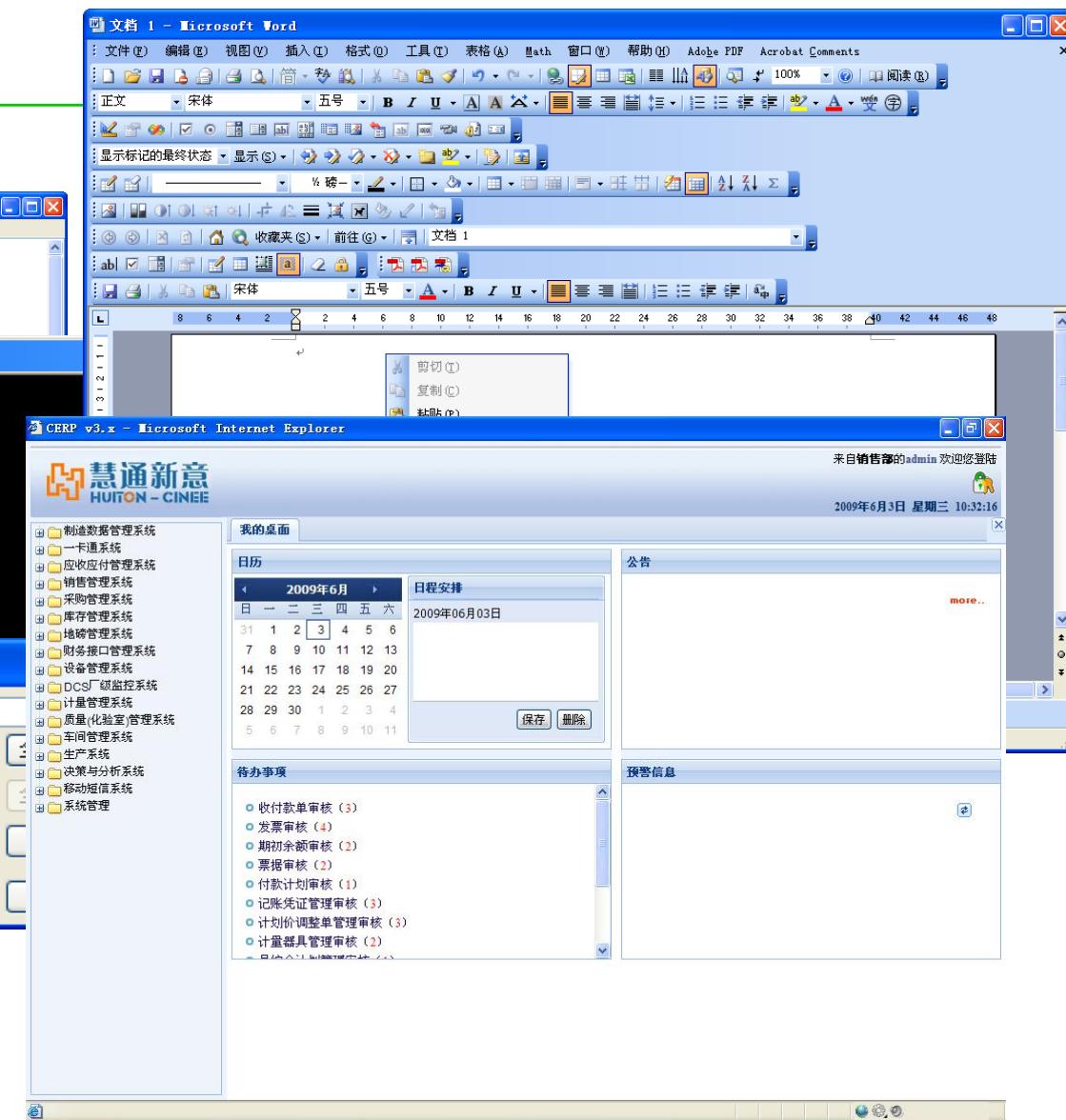
- 键盘输入：命令行、快捷键、...
- 批处理输入：.bat
- 直接操纵图形元素：按钮、图标、菜单(弹出式菜单、下拉菜单)、工具条、...
- 硬件接口输入：麦克风、...

指令输入方式

```
config_builder - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
@ECHO OFF
SETLOCAL

SET WL_HOME=E:\bea\weblogic81
SET JAVA_VENDOR=Sun
SET JAVA_HOME=C:\Program Files\Java\j2sdk1.4.2_05\jre

CALL "%WL_HOME%\bin\cmd.exe - cmd"
C:>dir
驱动器 C 中的卷没有标签。
卷的序列号是 C890-68A3
C:>\ 的目录
2006-12-10 07:08          0 AUTOEXEC.BAT
2007-04-07 16:06    <DIR>      autorun.inf
IF "%1" = 2007-01-28 08:23    <DIR>      b23afbd7f780d9f98b10c8
) ELSE (2007-03-26 07:00      17,044 cmd.txt
%JAVA_H2006-12-10 07:08          0 CONFIG.SYS
)2006-12-10 07:17    <DIR>      DELL
2007-04-07
2007-05-24 拼写和语法
2007-04-07
2007-07-21 不在词典中: rainy@hit.edu.cn
更改为(I): rainy@hit.edu.cn
建议: (没有建议)
添加(A) 自动更正(R)
将词添至(W): CUSTOM.DIC
```



信息输出方式

- **输出方式：**

- 单一数据：文本编辑框、列表框、单选框、复选框、 ...
- 成组数据：表格、树、 ...
- 大量数据：图形显示输出、 ...
- 软件接口输出：写入文件、写入数据库、输出到其它软件(Word、Excel、HTML、 ...)
- 动态数据：进度条、仪表盘、 ...

常用的数据显示方式

时间段：200909 部门：

部门指标比较

时间段：200909

部门	考核分	指标名称	实际值	基准值	标准值	量价值	上期实际	历史最优	单位	
生产部门	17.36	熟料产量	4435.00	4433.00	3587.50	10.00	4435.00	4435.00	吨	
		水泥产量	7000.00	6781.22	21175.00		7000.00	7000.00	7000.00	吨
		回转窑运转率	77.00	76.71	80.75		77.00	77.00	77.00	%
		水泥磨运转率	80.75	80.58	80.75		80.75	80.75	80.75	%
		电力消耗	9500.00	8000.00	10500.00		9500.00	9500.00	9500.00	千瓦时
		燃料消耗	5250.00	6000.00	5250.00		5250.00	5250.00	5250.00	克
终端能耗	9900.00	12285.00	12600.00		9900.00	9900.00	9900.00	千瓦时		

时间段：200910

正在复制...

从 '28_Crete Workshop' 到 '28_Crete Workshop'

剩余 45 秒

报表查询

熟料销售量 水泥销售量

熟料产量：4435.00 吨
水泥产量：7000.00 吨
回转窑运转率：77.00 %
水泥磨运转率：80.75 %
熟料销售量：6300.00 吨
水泥销售量：4725.00 吨

热料产量 水泥产量 回转窑运转率 水泥磨运转率 电力消耗
燃料消耗 终端能耗 熟料销售量 水泥销售量 熟料销售价

请购买FineReport 注册

CERP v3.x - Microsoft Internet Explorer

来自采购部的管理员 欢迎您登陆
2009年5月7日 星期四 16:19:24

采购需求单

单据编号：APP2009050002 制单人：管理员 制单日期：2009-05-07 单据状态：审核中

需求部门：采购部 业务日期：2009-05-07 总数量：370 总金额：0
需求说明：

需求信息	备注	审核信息	采购计划	采购合同	采购到货

采购项目	物料名称	规格型号	需求日期	需求数量	单位	单价	金额	计划价格
01	砂岩		2009-05-08	100.00	吨	0.00	0.00	0.00
01	干粉煤灰		2009-05-09	120.00	吨	0.00	0.00	0.00
02	混粉煤灰		2009-05-10	150.00	吨	0.00	0.00	0.00

付款单查询

2009年6月3日 星期三 10:27:41

来自销售部的admin 欢迎您登陆

单据日期	单据编号	单据状态	付款类别	业务类型	付款方式	部门
2009-05-18	FK200905180001	已通过	采购发票	蓝字	发票付款	销售部
2009-05-14	FK200905140001	已通过	采购发票	蓝字	发票付款	财务部
2009-05-13	FK200905130001	新增	采购发票	蓝字	发票付款	财务部
2009-04-24	FK200904240001	已通过	采购发票	蓝字	发票付款	销售部

我的桌面 智能新意慧通 HUNTON - CINEE

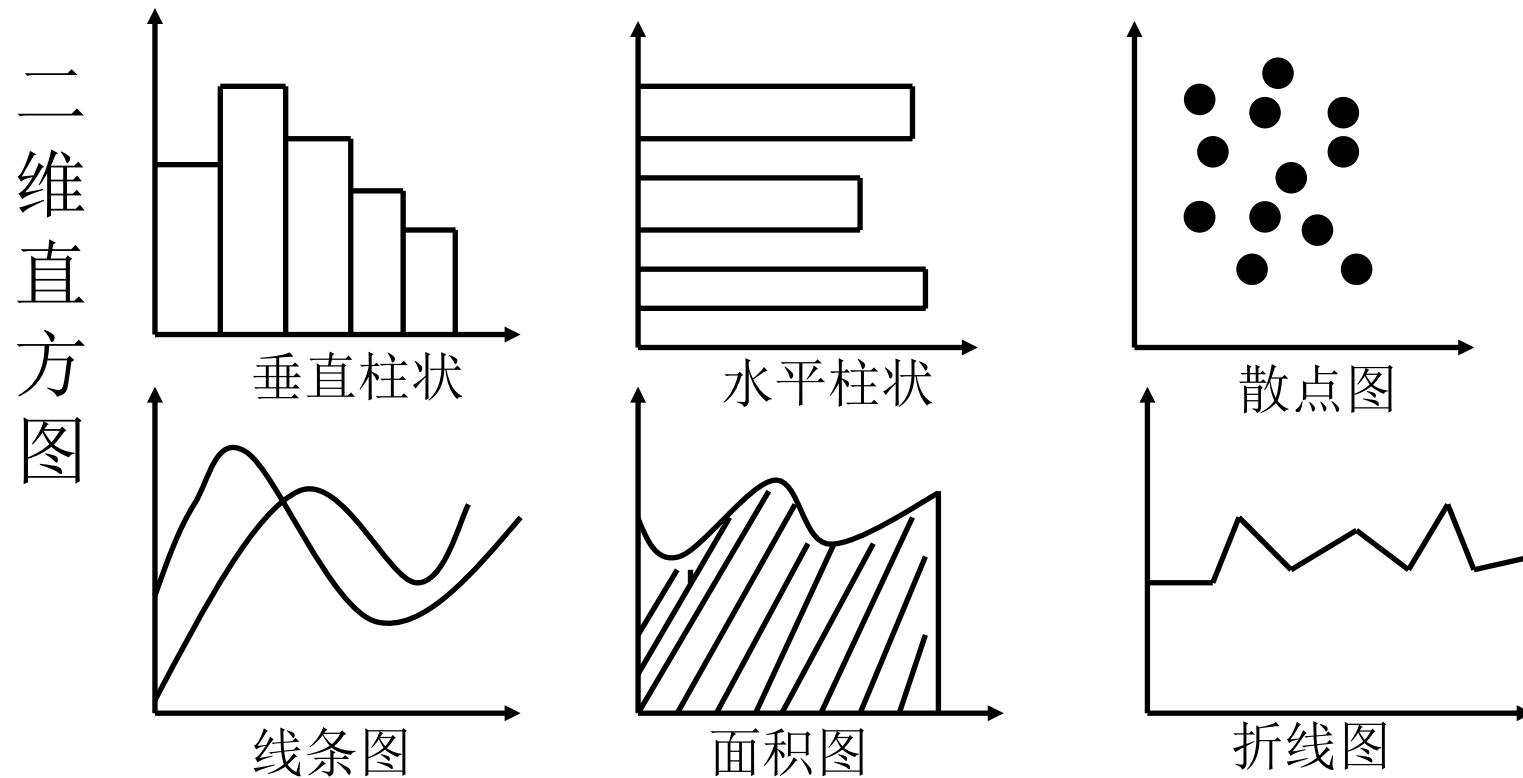
打印 打印预览 反审核 反作废 返回

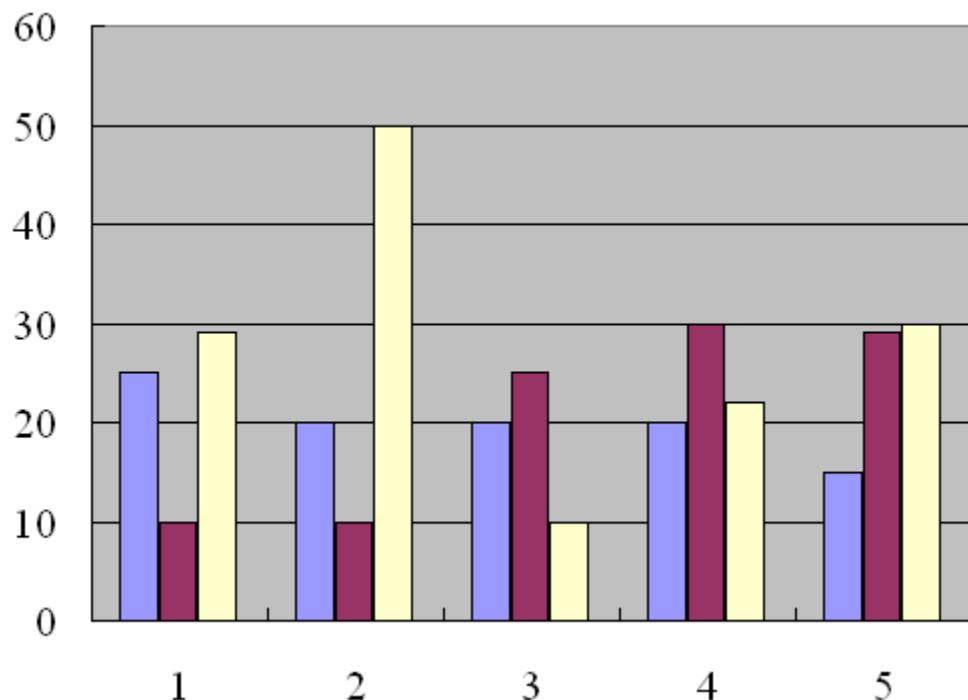
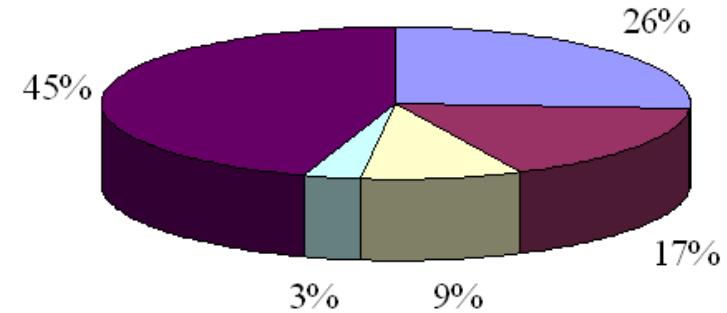
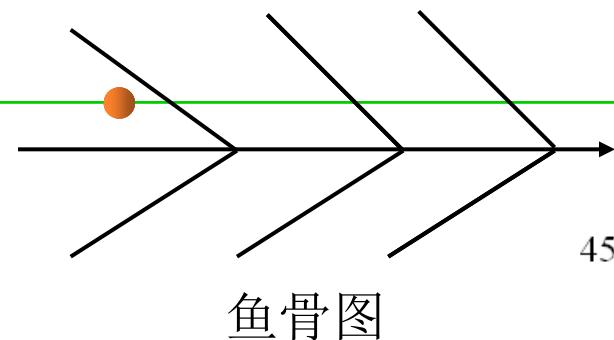
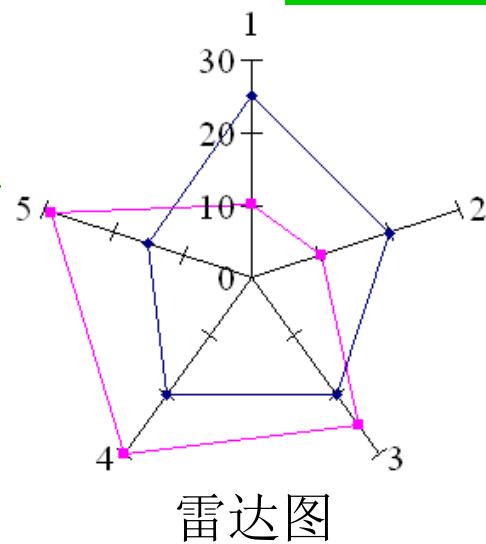
采购管理系统
采购基础数据管理
采购供应商管理
采购需求管理
采购需求录入
采购需求审核
采购需求查询
采购计划管理
采购合同管理
采购到货管理
采购到货通知单管理
采购折标管理
采购报表管理

需求信息 备注 审核信息 采购计划 采购合同 采购到货

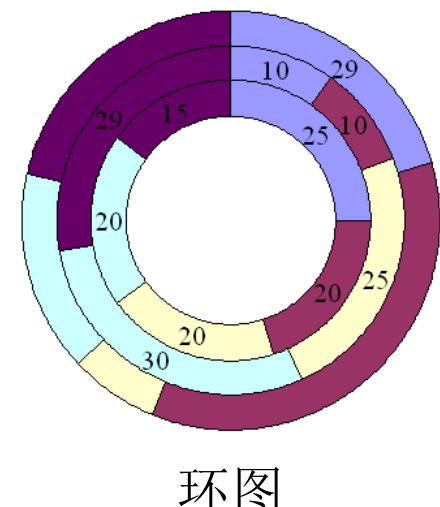
http://172.16.44.240:8087/CERP3x/org/fdm/main/mainPage.html#

常用的数据显示方式





饼图



排列图

主要内容

- 4.1 软件设计
- 4.2 体系结构设计
- 4.3 界面设计
 - 4.3.1 用户界面概述
 - 4.3.2 用户界面的设计原则
 - 4.3.3 界面分析和设计

1 UI设计的重要性

- 英国米德兰航空**092**空难：

- 1989年1月8日，英国米德兰航空公司的一架波音737-400从伦敦起飞，执行092次航班任务；
- 起飞13分钟，位于**28300英尺**高度时左发动机风扇叶片断裂，飞机产生剧烈振动；
- 机组错误的将右发动机关闭，导致飞机失去动力，最终坠毁在距离机场**27号跑道头900米**的高速公路旁，**47名旅客死亡**。

- 原因：

- 驾驶舱内报警信号器没有产生一个清楚的可识别左发动机故障的信号。

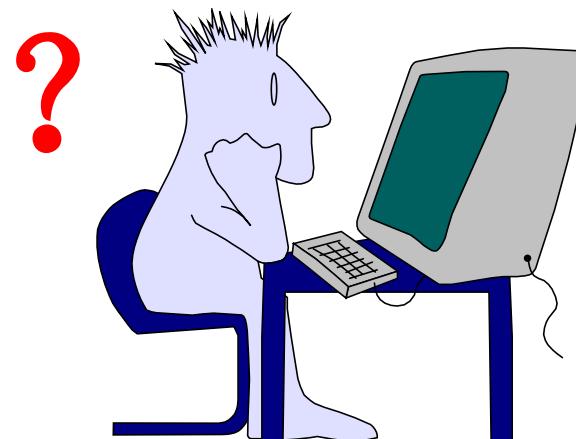
<http://www.xmatc.com/atcdata/html/44939.html>



1.UI设计的重要性

■ 界面设计对软件质量具有重要影响

- 不管软件提供了什么样的强大计算能力与功能，如果软件不方便使用，使用它常导致犯错或者不利于完成目标，那么用户是不会喜欢并接受该软件的。
- 用户界面是软件的直观形象，直接影响到用户对软件的感觉，用户通常从用户界面(而不是从功能)的好坏上来判断软件的质量。



2. UI设计的评价

- UI设计水平的高低决定了用户完成以下任务的难度以及所需要耗费的时间/成本：
 - 学习使用软件系统
 - 向软件系统提供输入
 - 从软件系统获取并理解输出
- 使用可用性(**Usability**)加以度量
 - *easy to learn*
 - *easy to use*
 - *easy to understand*

3. UI设计原则

- 用户目标导向设计
 - 不要炫耀技术，隐藏实现细节



IE8000 Microsoft Internet Explorer
CERP v3.x - Microsoft Internet Explorer

慧通新意 HUITON - CINEE

来自销售部的admin 欢迎您登陆

2009年6月3日 星期三 10:36:57

我的桌面 采购发票查询

反审核 反作废 返回

采购发票

发票编号: FP200905130001 制单人: 管理员 开票日期: 2009-05-13 单据状态: 已通过

业务类型: 蓝字 发票类型: 采购普通发票 发票日期: 2009-05-13 发票号: 205048787

供货单位: 河南省中原起重机 部门: 财务部 业务员: 管理员 付款条件:

备注:

发票信息 审核信息

	序号	物料编号	物料名称	规格型号	单位	数量	单价	金额	税额
<input type="checkbox"/>	1	0408010331	大头轴承		个	2.0000	10.00	20.00	3.4
<input type="checkbox"/>	2	0408010332	联杆衬套		个	5.0000	20.00	100.00	17.00
<input type="checkbox"/>	3	0408010333	大灯总成		个	8.0000	10.00	80.00	13.6

数量合计: 15.0000 金额合计: 200.00 税额合计: 34.00 小计: 234.00

http://172.16.44.240:8887/CERP3x/org/fdm/main/mainPage.html#

20

UI设计原则

- 新手、专家和中间用户
 - 让新人感到易学、易用
 - 让中间用户感到便捷
 - 让高手们感到无与伦比的成就感。

一个“在线升级”软件的示例

- 1、执行“在线升级”菜单命令
- 2、对话框，***最新版本已经发布，强烈建议您将***更新，您现在要升级吗？确定 取消
- 3、下载进度对话框 取消？
(if 取消) 对话框 正在下载***的升级文件，您决定放弃下载吗？如果放弃，您将暂时不能使用新版***的功能。
(下载完成进入卸载流程)
- 4、***软件正在运行，安装程序将关闭软件并将其卸载，是否继续？确定 取消？
- 5、您确实要完全卸载***及其所有的组件？是 否？
- 6、您要删除在本机的个人信息和历史记录吗？是 否？
- 7、卸载进度对话框
- 8、卸载完成对话框，***已成功从您的计算机中删除，确定？
- 9、欢迎安装对话框，“单击下一步继续”下一步 取消？
- 10、许可协议，“同意 不同意” 下一步 取消？
- 11、选择安装位置，下一步 取消？
- 12、选择开始菜单文件夹，安装 取消？
- 13、显示安装进度
- 13、安装完成对话框 完成 取消？
- 14、执行启动程序

UI设计规则



■ 黄金规则

- 置系统于用户的控制之下
- 减少用户的记忆负担
- 保持界面一致

■ 补充规则

- 个性化
- 宽容性
- 反馈
- 美观性
- 可用性
- 简洁性

(1) “置系统于用户的控制之下”



- 用户应当感觉系统在自己的控制之下，能够便捷的启动系统行为并获得所需要的结果；
- 即使系统取得控制权，也应给用户以必要的反馈：
 - 以不强迫用户进入不必要的或不希望的动作的方式来定义交互模式；
 - 提供灵活的交互；
 - 允许用户交互被中断和撤销；
 - 当技能级别增长时可以使交互流线化并允许定制交互；
 - 使用用户与内部技术细节隔离开来；
 - 允许用户与出现在屏幕上的任何对象进行直接交互。

(2) “减少用户的记忆负担”

- 用户在使用系统时，必须记住的东西越多，和系统交互时出错的概率就越高：
 - 减少对短期记忆的要求：不要求用户记住过去的动作和结果；
 - 建立有意义的缺省；
 - 定义直观的快捷方式：如快捷键；
 - 界面的视觉布局应该基于真实世界的象征：尽可能与实际使用的物理对象保持一致；
 - 以不断进展的方式揭示信息，而不是一下子全部展示出来。
 - 导航。



(3) “保持界面一致”

- 一致性要求UI遵循标准和常规的方式，让用户处在一个熟悉的和可预见的环境之中。
 - 在一个系统家族内的所有UI应保持一致的界面风格；
 - 如果过去的交互模型已经建立起了用户期望，除非有不得已的理由，否则不要改变它。
 - 界面的外观、界面上菜单/按钮/屏幕区域的命名和编码、图形元素在屏幕上的位置、图形元素的大小/颜色/间距/对齐方式等。
 - 例如：同样的按钮在所有窗口中保持一致的位置、始终使用一致的配色方案等

(4) “个性化”



- 个性化可以使不同的人按照自己的习惯、爱好、对系统的熟知程度来设置**UI**元素：
 - 排列菜单；
 - 排列数据项的次序；
 - 隐藏/显示某些图形元素；
 - 调整大小
 - 为新用户和高级用户提供不同复杂度的界面
 - 等等
- 用户调整之后，系统能够保存，用户下次登录时自动显示上次设置的状态。

(5) “宽容性”



- 以一种宽容的态度允许用户进行实验和出错，用户在出现错误时能够方便的从错误中恢复。
- 正确引导你的用户
- 所有的交互设计，都应该是以避免用户犯错为前提，而不是以实现的简单性为前提。等用户犯了错误再提示，这样会打击你用户使用的信心。

(6) “反馈”

- 在错误发生或程序运行时间较长时，界面应提供有意义的反馈，并有上下文帮助功能。
- 注意：不要打扰你的用户

(7) “美观性和可用性”



- 界面美观性是视觉上的吸引力，主要体现在：平衡和对称性、合适的色彩、各元素具有合理的对齐方式和间隔、相关元素适当分组、使用户可以方便地找到要操作的元素等。

美学不能取代功能(Functionality > Aesthetics)

(8) “简洁性”

- 避免使用许多复杂的图片和动画等造成用户操作时的分心；
- 界面布局应当适合清晰地表达信息；
- 具有与之匹配的导航性；

主要内容

- 4.1 软件设计
- 4.2 体系结构设计
- 4.3 界面设计
 - 4.3.1 用户界面概述
 - 4.3.2 用户界面的设计原则
 - 4.3.3 界面分析和设计

界面分析

- 用户分析

- 用户的背景（教育、年龄等）
- 用户相关知识的掌握（领域知识、计算机知识等）
- 用户工作的情况（时间、地点等）

- 任务分析和建模

- 用例
- 任务细化（系统顺序图）
- 对象细化（边界类图）
- 工作流分析（活动图）

- 显示内容分析

- 文字、表格、图形

- 工作环境分析

界面设计步骤

- **结构设计-->交互设计-->视觉设计**

- 结构设计也称概念设计，是界面设计的骨架。通过对用户研究和任务分析，制定出产品的整体架构（边界类为基础）
- 交互设计的目的是使产品让用户能简单使用（交互模型）
- 在结构设计的基础上，参照目标群体的心理模型和任务达成进行视觉设计

结构设计

- 结构设计 **Structure Design**

结构设计也成概念设计（**Conceptual Design**），是界面设计的骨架。通过对用户研究和任务分析，制定出产品的整体架构。基于纸质的的低保真原型（**Paper Prototype**）可提供用户测试并进行完善。

- 在结构设计中，目录体系的逻辑分类和语词定义是用户易于理解和操作的重要前提。如西门子手机的设置闹钟的词条是“重要记事”，让用户很难找到。

交互设计

■ 交互设计 **Interactive Design**

交互设计的目的是使产品让用户能简单使用。任何产品功能的实现都是通过人和机器的交互来完成的。因此，人的因素应作为设计的核心被体现出来。交互设计的原则如下：

- 1) 有清楚的错误提示。误操作后，系统提供有针对性的提示。
- 2) 让用户控制界面。“下一步”、“完成”，面对不同层次提供多种选择，给不同层次的用户提供多种可能性。
- 3) 允许兼用鼠标和键盘。同一种功能，同时可以用鼠标和键盘。提供多种可能性。
- 4) 允许工作中断。例如用手机写新短信的时候，收到短信或电话，完成后回来仍能够找到刚才正写的新短信。
- 5) 使用用户的语言，而非技术的语言。
- 6) 提供快速反馈。给用户心理上的暗示，避免用户焦急。
- 7) 方便退出。如手机的退出，是按一个键完全退出，还是一层一层的退出。提供两种可能性。
- 8) 导航功能。随时转移功能，很容易从一个功能跳到另外一个功能。
- 9) 让用户知道自己当前的位置，使其做出下一步行动的决定。

视觉设计

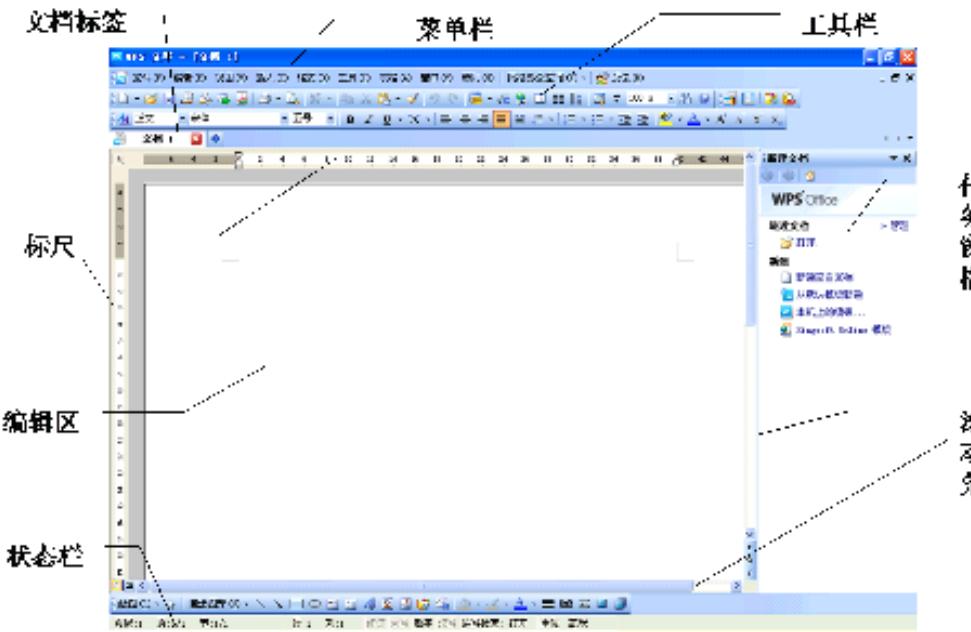
■ 视觉设计 **Visual Design**

在结构设计的基础上，参照目标群体的心理模型和任务达成进行视觉设计。包括色彩、字体、页面等。视觉设计要达到用户愉悦使用的目的。视觉设计的原则如下：

- 1) 界面清晰明了。允许用户定制界面。
- 2) 减少短期记忆的负担。让计算机帮助记忆，例：**User Name**、**Password**、**IE**进入界面地址可以让机器记住。
- 3) 依赖认知而非记忆。如打印图标的记忆、下拉菜单列表中的选择
- 4) 提供视觉线索。图形符号的视觉的刺激；**GUI**（图形界面设计）：**Where, What, Next Step**
- 5) 提供默认（**default**）、撤销（**undo**）、恢复（**redo**）的功能
- 6) 提供界面的快捷方式
- 7) 尽量使用真实世界的比喻。如：电话、打印机的图标设计，尊重用户以往的使用经验。
- 8) 完善视觉的清晰度。条理清晰；图片、文字的布局和隐喻不要让用户去猜。
- 9) 界面的协调一致。如手机界面按钮排放，左键肯定；右键否定；或按内容摆放。
- 10) 同样功能用同样的图形。
- 11) 色彩与内容。整体软件不超过5个色系，尽量少用红色、绿色。近似的颜色表示近似的意思。

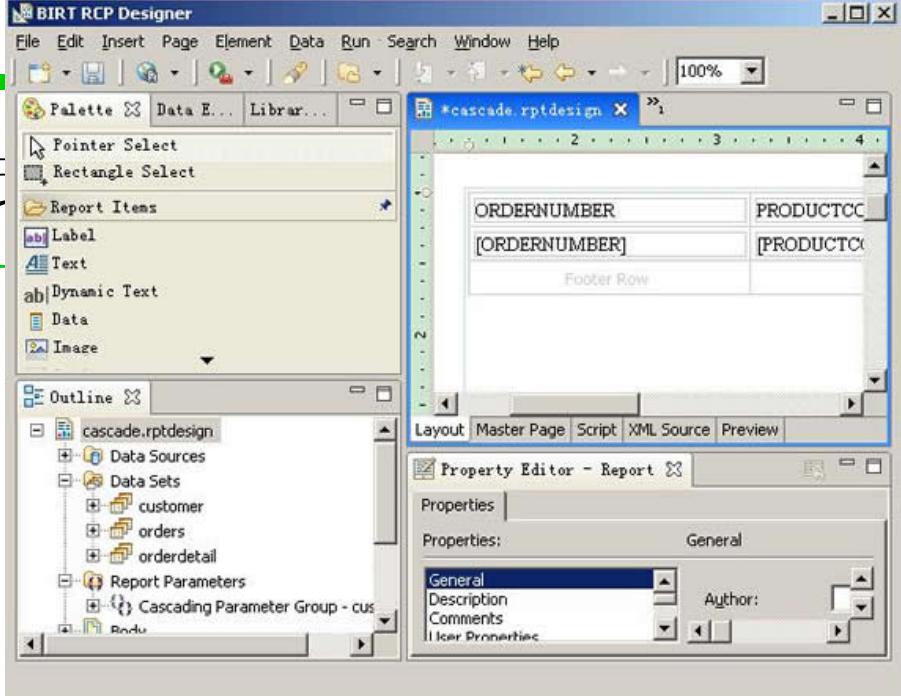
结构设计——界面设计模式

- Windows标准模式
- 导航模式



结构设计——界面设计模块

- 层叠模式
- 表格模式
- 向导模式



结构设计需考虑的若干因素

- 导航
- 界面布局
- 界面大小
- 界面颜色
- 菜单设计
- 信息表示
- 效率
- 安全性
- 用户支持
- 国际化

如果无法记住这么多因素
请在任何时候均参照**Windows**系统的UI

(1) 导航

- 单一界面、多界面?
 - 是否允许用户同时打开多个界面并行的操作?
 - 多界面是主流。
- 在多界面的情况下:
 - 可在任何界面都回到主界面;
 - 从主界面可直接到达任何界面: 例如通过搜索引擎;

(2) 界面布局

- 界面的典型区域：

- 标题栏
- 主菜单
- Logo
- 输入内容区域
- 输出内容区域
- 信息提示区域
- 帮助区域
- ...
- 还有其他的吗？

- 界面布局的原则：

- 多界面的布局格式要统一
- 标题栏目要醒目清晰、有实际意义
- 合理利用空间，不能太拥挤，也不能有太多的空白；
- 在简洁和复杂界面之间保持平衡；
- 少使用代码和缩写；
- 颜色、字符大小、字体要恰当；
- 对齐：
 - 左对齐、右对齐、中间对齐、按小数点对齐、...
- 分布：
 - 水平平均分布、垂直平均分布

(3) 界面大小



- **界面的大小:**

- 固定大小: 用户不可调整;
 - 可变大小: 用户可伸缩界面大小, 或者界面具有自适应性;

- **界面与屏幕的关系:**

- 界面应尽可能在屏幕范围内显示。
 - 如果超过屏幕范围, 则需要滚动条
 - 水平滚动条: 尽量避免;
 - 垂直滚动条: 可以接受, 但滚动范围不要太大;

- **界面与屏幕分辨率的关系:**

- 固定分辨率;
 - 自适应屏幕的分辨率;

(4) 界面中的颜色

- 颜色能够改善用户界面，帮助用户理解系统的复杂信息结构，有时颜色可以用于突出显示例外事件。
- 使用颜色的指导原则：
 - 避免使用太多的颜色：通常一个窗口内不要多于三种颜色；
 - 使用颜色支持用户的不同数据或任务(例如：错误信息用红色)；
 - 允许用户控制颜色(例如：用户可选择错误信息用其他颜色)；
 - 使用颜色时需要前后一致(例如：所有的错误信息都用红色)；
 - 在颜色的选取上尽可能符合人们的习惯用法；
 - 使用颜色的变化显示系统状态的变化；
 - 注意在低分辨率情况下的颜色显示；
 - 注意颜色的搭配；
 - 背景一般选用饱和度低的浅色，并且背景与前景色要协调搭配。

(5) 菜单设计

- 菜单合理分类:
 - 系统功能、逻辑顺序等
- 菜单命名
 - 简明、有意义、与它所代表的指令要一致
- 合理组织菜单界面的结构与层次
 - 广而浅优于窄而深的菜单树、最佳项数4-8、最深层次尽量少于4层
- 菜单项的安排应有利于提高菜单选取速度:
 - 可以依据使用频度、数字顺序、字母顺序、功能逻辑顺序安排。
 - 注意：隔离使用频度高但相邻相似项致使误操作的菜单项，如Copy与Cut
- 保持各级菜单显示格式和操作方式的一致性
- 为菜单项提供多于一种的选择途径
 - 键盘、鼠标、快捷方式等
- 对菜单选择和点取设定反馈标记:
 - 状态栏；

(6) 信息表示

- 信息的表示方式：文本方式和图形方式
 - 文本方式占据较少的屏幕空间
 - 图形方式所显示的内容比较直观
- 举例
 - 某系统需要显示库存商品的名称、规格、单价、数量等信息，应该选择哪一种方式表示信息？
 - 某系统需要按月汇总某个商品的销售情况，应该选择哪一种方式表示信息？
 - 某系统需要监控一个设备的压力和温度，应该选择哪一种方式表示信息？

信息表示

- 屏幕上只显示必要的数据，显示的数据与用户当前选择执行的任务密切有关；
- 一起使用的数据应放在一起显示：内聚性；
- 按使用频率、重要性、功能关系或使用顺序进行排序；
- 按数据的关系进行分组：
 - Tab页

(7) 效率

- 快速的系统响应：用户操作之后应马上给出响应；
- 给用户提示：目前进展到哪一步、下一步可能需要做什么事情；
- 使用用户输入减至最少：
 - 当同样的信息在多个地方都需要时，系统应该复制该信息，而不是由用户重复输入；
 - 如果有些输入数据项有缺省值，则给出缺省值；
 - 鼠标和键盘具有同等效用：快捷键、在各个编辑域之间的切换(回车键、Tab键、PageUp/PageDown键、Home/End键、Insert/Delete键、F1键)；
 - 检验标准：用户完成一个功能所需要操作键盘/鼠标的总次数。

(8) 安全性



- 输入的正确性校验:
 - 提示数据的允许范围和输入方法，对不可接受的值都给出出错信息；
- 提供**Undo/Redo**操作:
 - 随时撤消或重复之前的操作。
 - 单步/多步；
- 对重要的有破坏性的命令提供确认措施（“确认” 和 “取消”）；
- 在程序运行中提供中断功能：随时停止当前操作。

(9) 用户支持

- 提供信息提示与反馈
 - 用户界面应该提供清晰的系统提示和反馈信息；
 - 错误信息描述应当是简洁的、礼貌的、一致的和建设性的；
 - 设计错误信息时应该预见到用户的背景和经验；
- 良好的联机帮助
 - 帮助系统应该提供给用户多个不同的入口；
 - 帮助系统具有复杂的网络结构，从其中的每一个帮助页面都可以访问其他的信息页面；

(10) 国际化

- 用户界面通常是一个国家和一种语言所设计的，在面对其他国家时只好应急对付；
- 应设计“全球化”的软件，实现多语言性，用户可在不同语言之间进行切换。

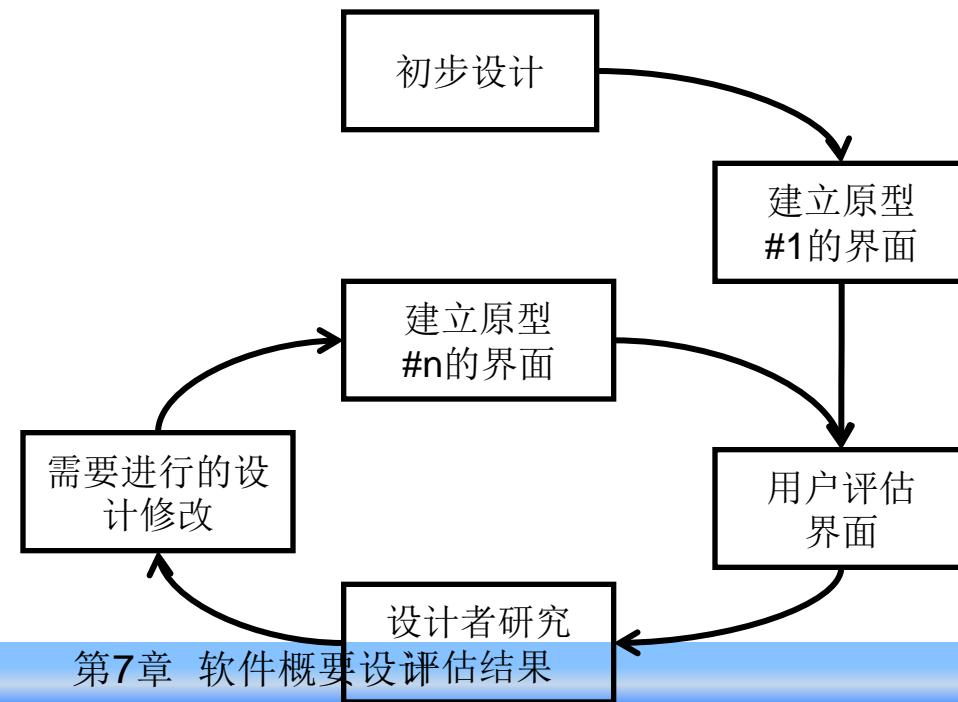
- [课堂讨论]解决方法有哪些？
 - Unicode标准

总结：UI设计的原则

- 如果无法记住这么多**UI**设计原则，请在任何时候均参照**Windows**系统的**UI**。
 - 布局
 - 字体
 - 字号
 - 颜色
 - 帮助
 - 提示
 - Redo/Undo
 - 快捷键
 - ...

界面设计评估

- UI的书面规格说明书的长度和复杂性：用户学习系统的难度；
- 命令的个数以及命令的平均参数个数：系统交互的时间和系统总体的效率；
- 动作和状态的数量：用户学习系统时所需记忆的内容的多少；
- 界面风格、帮助设施和错误处理协议：界面的复杂度和用户的接受程度。





结束

2015年9月27日