第二章 软件体系结构风格

一.单选题 (共1题,100.0分)

1

采用rational rose的RUP软件生命周期模型， 需求分析是否要分析软件体系结构?

A、 是

B、 否

一.单选题 (共1题,100.0分)

1

采用rational rose的RUP软件生命周期模型，软件设计是否要设计软件体系结构?

A、 是

B、 否

实验八 用户权限系统-部署图

一、实验目的

1 熟悉用 UML 语言编写软件系统的部署图。

2 熟悉国家标准《面向对象软件的文档编制》文档中关于部署建模规范。

二、实验装置

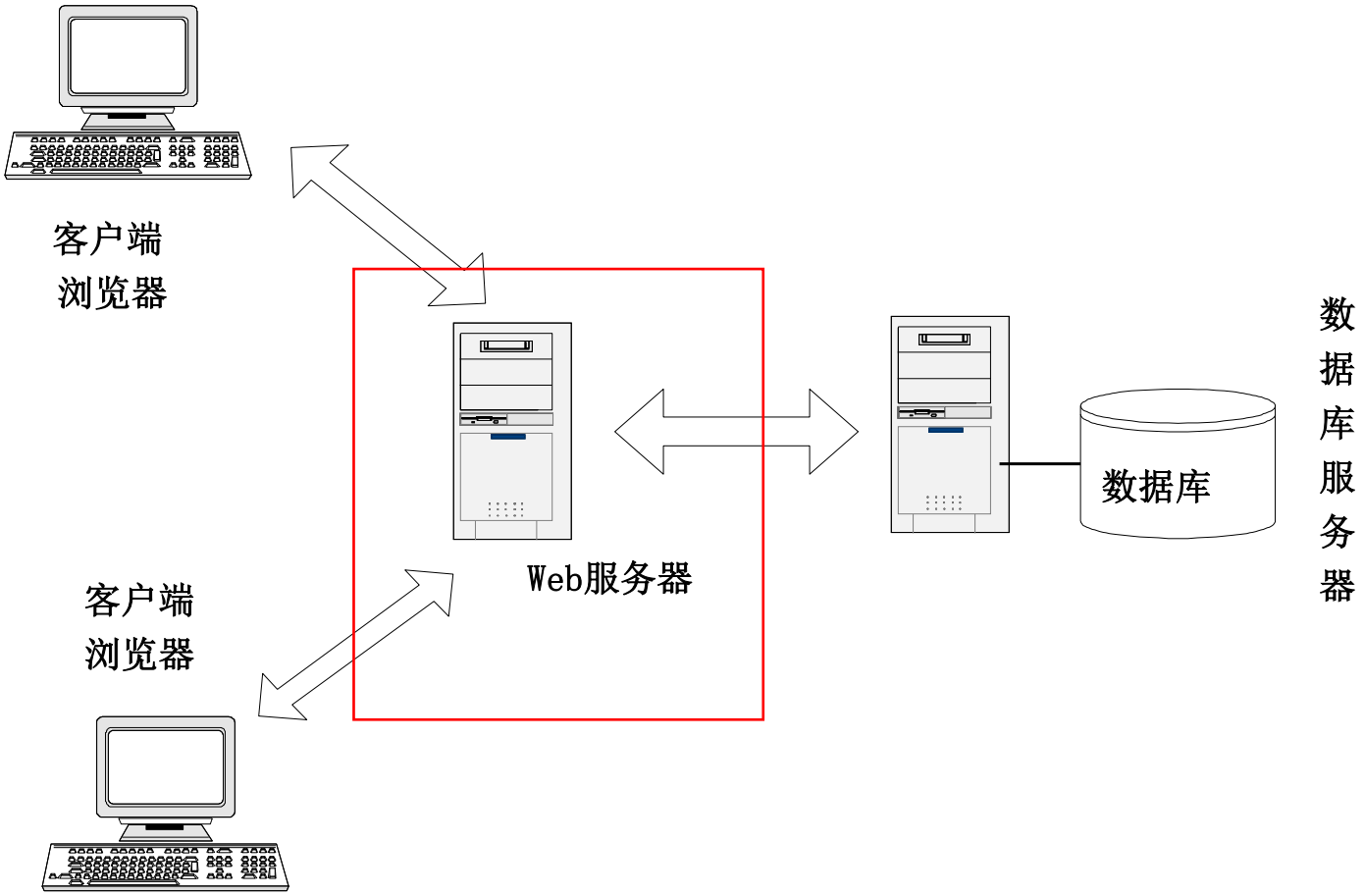
电脑， rational rose（UML 建模）

三、实验内容

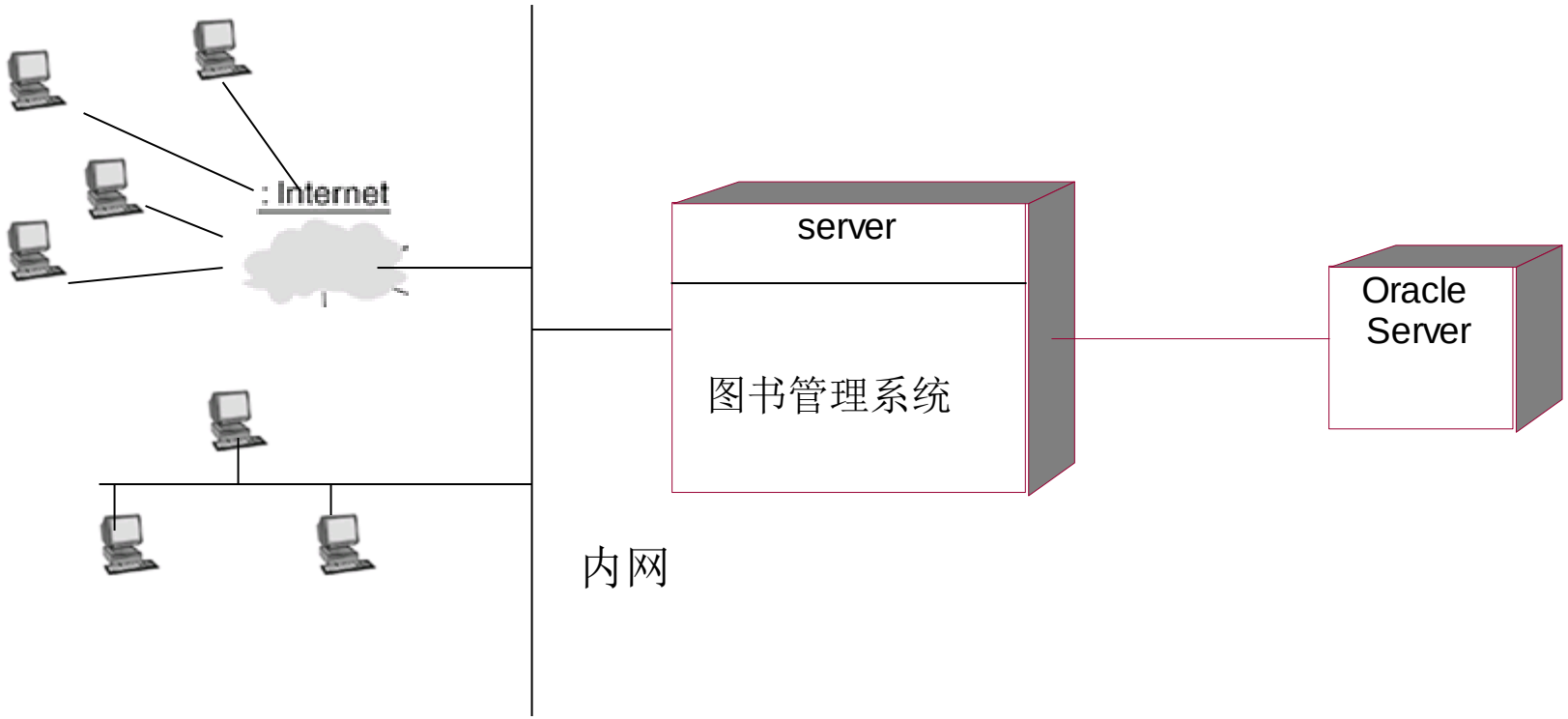
1. 根据具体软件风格建模系统的部署图。

2. 部署图建模内容参照国家标准《面向对象软件的文档编制》。

◇ 网络拓扑结构



部署图(Deployment Diagrams)



图书管理系统部署图

考勤系统-目的及要求

一　目的

1 综合运用软件工程知识分析、设计、开发一个软件系统：考勤系统。

2 培养软件系统分析、设计、开发和文档写作能力。

3 培养沟通能力和团队合作精神。

二　要求

1 开发一个能对大学生上课进行考勤的软件系统

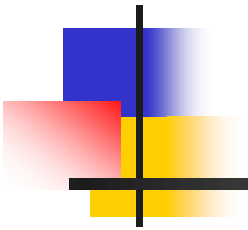
2 软件系统包含的功能：

（1）　基础信息管理子系统：课程信息管理、教师信息管理、学生信息管理

（2）　业务子系统：考勤功能（通过手机人脸识别进行考勤 ）、考勤统计

（3）　用户权限子系统：角色管理、功能管理、角色分配管理、权限分配管理

完成考勤系统的部署图(Deployment Diagrams)

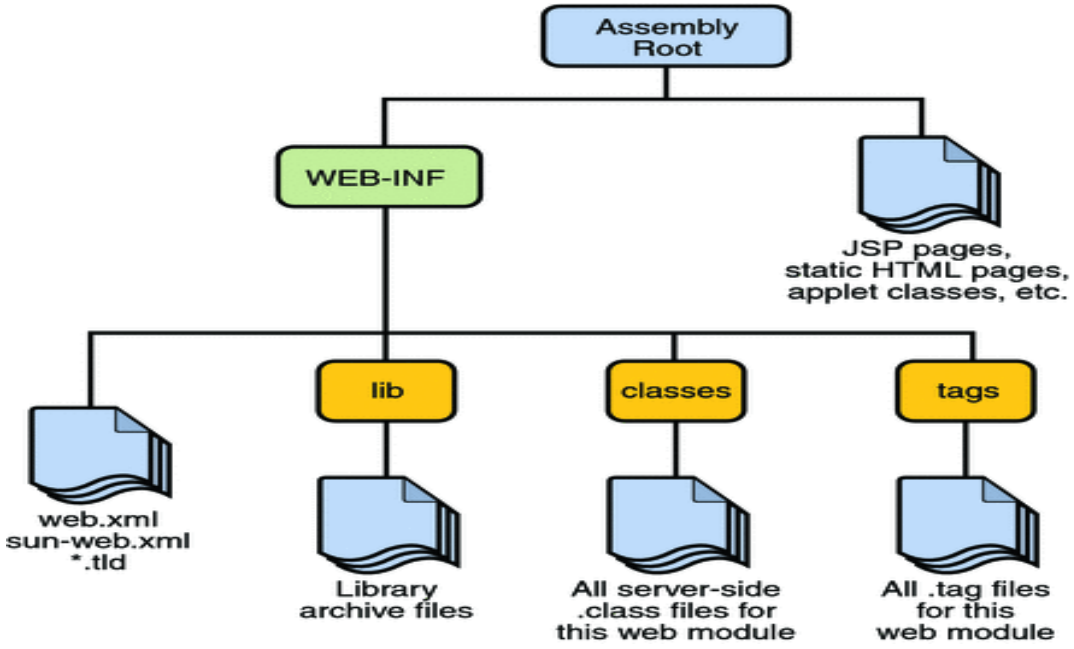


软件体系结构的特点之一就是抽象出了很多常见的系统构建模式，这些模式（或者说风格）是系统设计人员多年工作经验的总结。

2.1 软件体系结构风格概述

2.1.1 软件体系结构风格：

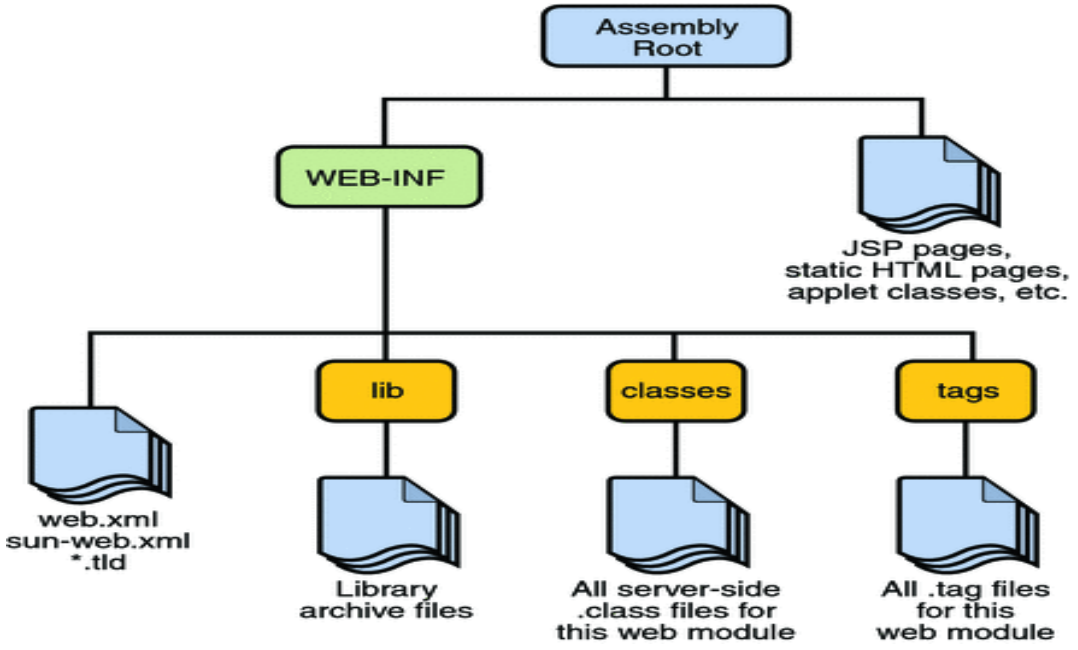
- 描述某一特定应用领域中系统组织方式的惯用模式



Web风格

- **软件体系结构风格（模式） = 经过实践证实的特定应用领域的软件体系框架或者拓扑结构**

- 体系结构风格，反映了领域中众多系统所共有的结构和语义特征，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。

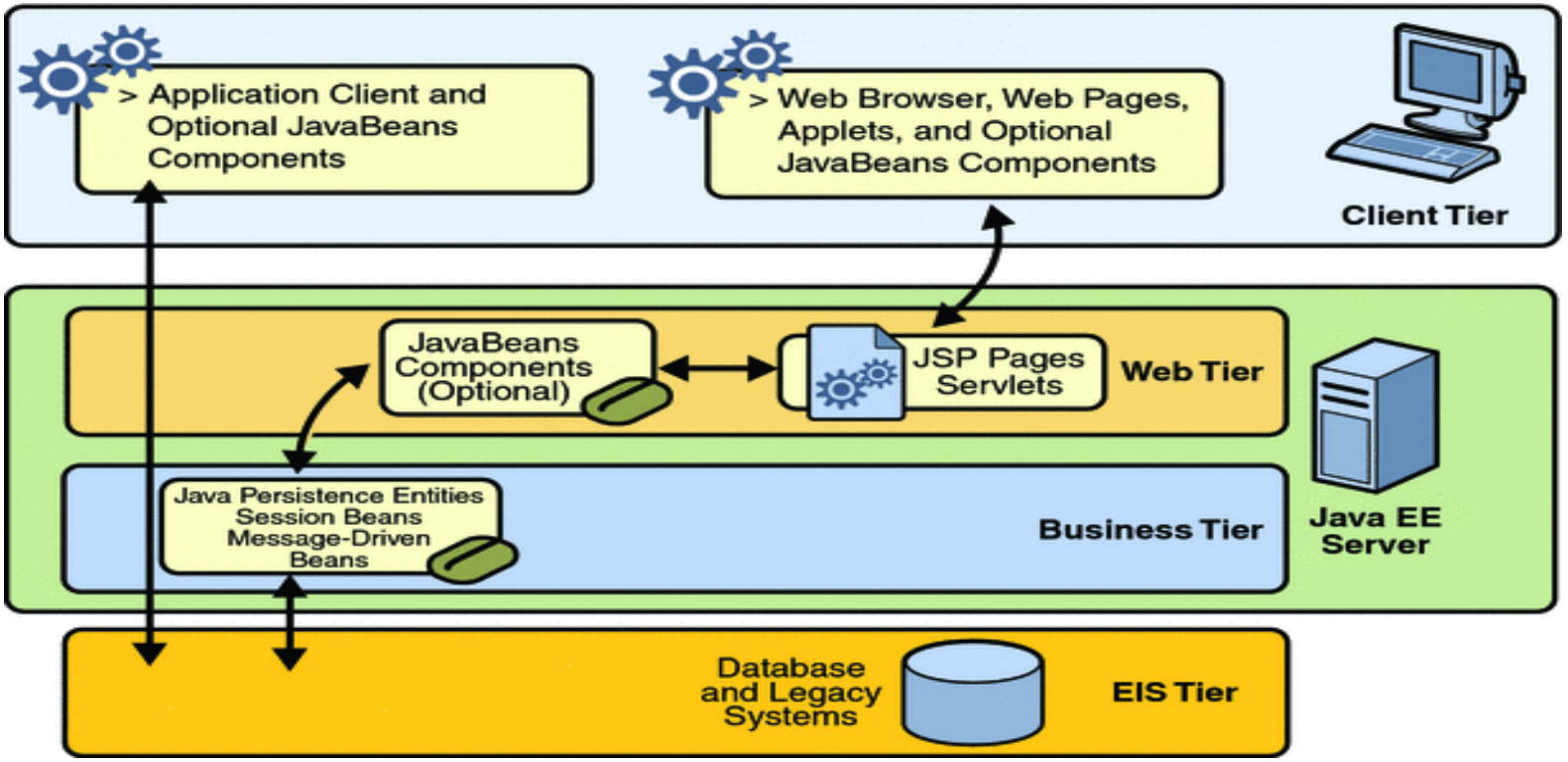


Web风格

- 软件体系结构风格定义了用于描述系统的术语表和一组指导构建系统的规则。
  即一个体系结构定义一个词汇表和一组约束。

  - 词汇表中包含了一些构件和连接件类型。
  - 约束指出系统是如何将这些构件和连接件组合起来。

例 Web Tier and JavaEE Application



构件：JSP, Servlet, JavaBeans,Java Persistence Entities,Session Beans, Message-Driven Beans

连接件：JavaEE Server(Tomcat)

约束：Web.xml…

例：　C/S风格

构件：　Client, Server

连接器：　C/S间的通讯协议

2.1.2 使用软件体系结构风格的益处

（1）可以重用已经成功的软件体系结构,达到体系结构级别的软件重用。

（2）有利于团队的交流。

例如：如果有人把系统描述成Web风格，则不必给出设计细节，我们立即会明白系统是组织和运行的。

2.2 软件体系结构风格

Garlan和Shaw给出的软件体系结构风格的分类

1. 管道-过滤器风格
2. 面向对象风格
3. 事件驱动风格
4. 分层风格
5. 数据共享风格
6. 解释器风格
7. 反馈控制环风格
8. 异构风格的集成

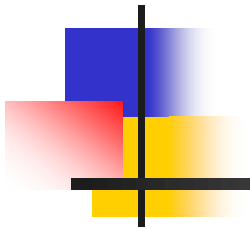
2.2 软件体系结构风格

补充：

1. C/S ： JavaSE/c++语言实现
2. B/S ： JavaEE/Asp.net技术实现
3. Web Service：Java WebService

（1、2、3 共用）数据库

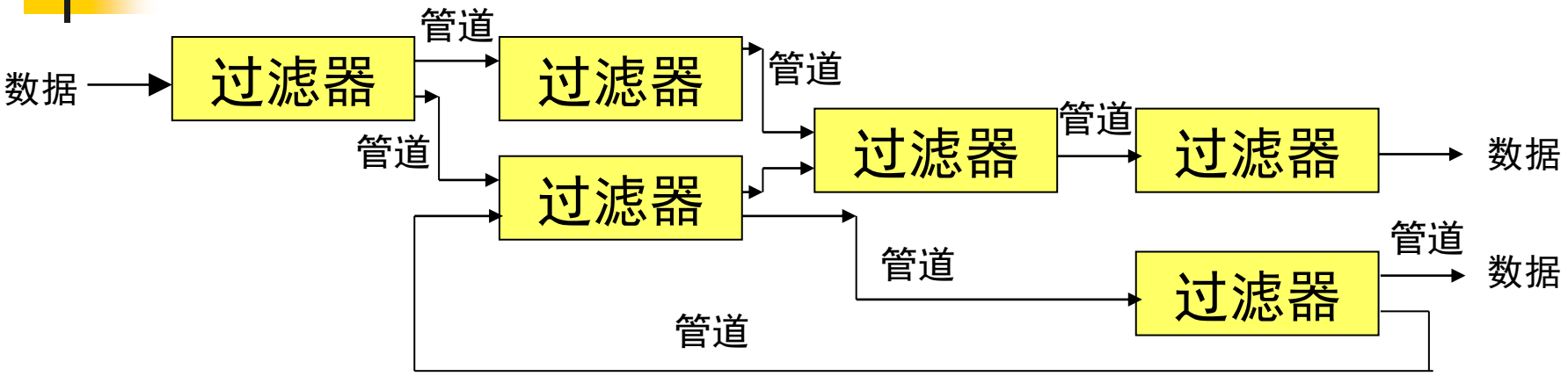
4. 云计算（云服务）：hadoop平台
5. 大数据 ：hadoop平台



特别注意：体系结构风格不是对软件进行分类的标准。它仅仅是从不同角度描述软件。

例如一个系统采用了分层风格，但这并不妨碍它用面向对象的方法来实现。同一个系统采用多种风格造成了所谓体系结构风格的异构组合。

2.2.1管道-过滤器风格



- 构件：过滤器
- 连接件：数据流传输的管道
- 管道-过滤器风格的体系结构是面向数据流的软件体系结构。
- 在管道-过滤器风格下，每个功能模块都有一组输入和输出。
  - 功能模块称作**过滤器**（filters）；
  - 功能模块间的连接可以看作输入、输出数据流之间的通路，所以称作**管道**（pipes）。

2.2.1.1 管道-过滤器风格特性

特征：系统中构件之间通过数据流松散耦合。

也就是说，**构件之间的依赖仅仅是数据流**，而不是通常的接口函数调用或消息传递。

（1）过滤器是独立运行的构件

- 非邻近的过滤器之间不共享状态
- 过滤器自身无状态
- 各过滤器在输入具备后完成自己的计算。完整的计算过程包含在过滤器之间的拓扑结构中。

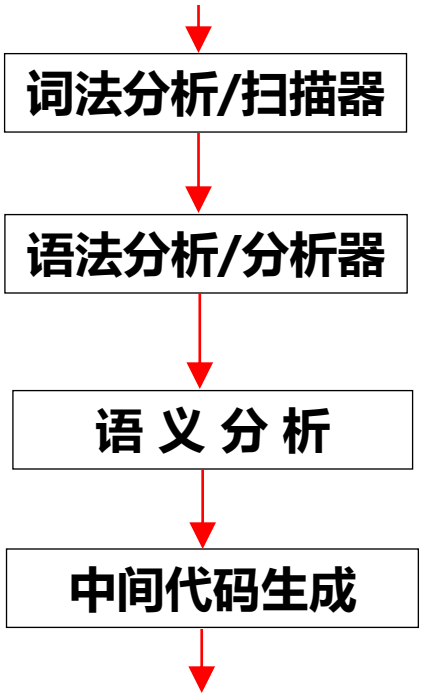
（2）过滤器对其处理上下连接的过滤器“无知”

- 对相邻的过滤器不施加任何限制

2.2.1.2 管道-过滤器风格优点

（1）由于每个构件的行为不受其他构件的影响，因此，整个系统的行为比较易于理解。

- 设计者可以将系统抽象成一个“黑匣子”，其输入是系统中第一个过滤器的输入管道，输出是系统中最后一个过滤器的输出管道，而其内部各功能模块的具体实现对用户完全透明。



2.2.1.2 管道-过滤器风格优点

（2）支持功能模块的复用。任意两个过滤器只要在相互的输入、输出管道格式上达成一致，就可以连接在一起。

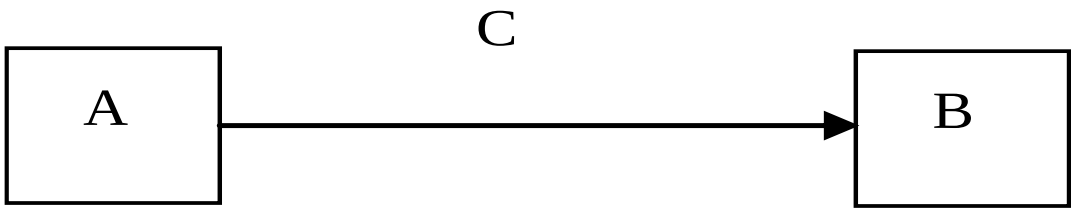


图　管道-过滤器支持功能模块复用

过滤器A和过滤器B只要对管道C中传输的数据格式达成一致就可以实现互连，其中过滤器A并不关心过滤器B如何处理管道C的内容，而过滤器B也不知道管道C的内容究竟是如何产生的，即在管道过滤器模式中，过滤器之间仅需很少的信息交换就可以完成互连。

2.2.1.2 管道-过滤器风格优点

（3）具有较强的可维护性与可扩展性。可维护性体现在系统过滤器部件的更新或升级。

- 由于技术改进等原因，过滤器A的实现发生了改变，采用新技术开发的过滤器D具有和A完全相同的输入、输出管道接口，这时可以直接将A替换为D，而无须对系统中的其他过滤器或管道进行任何修改。

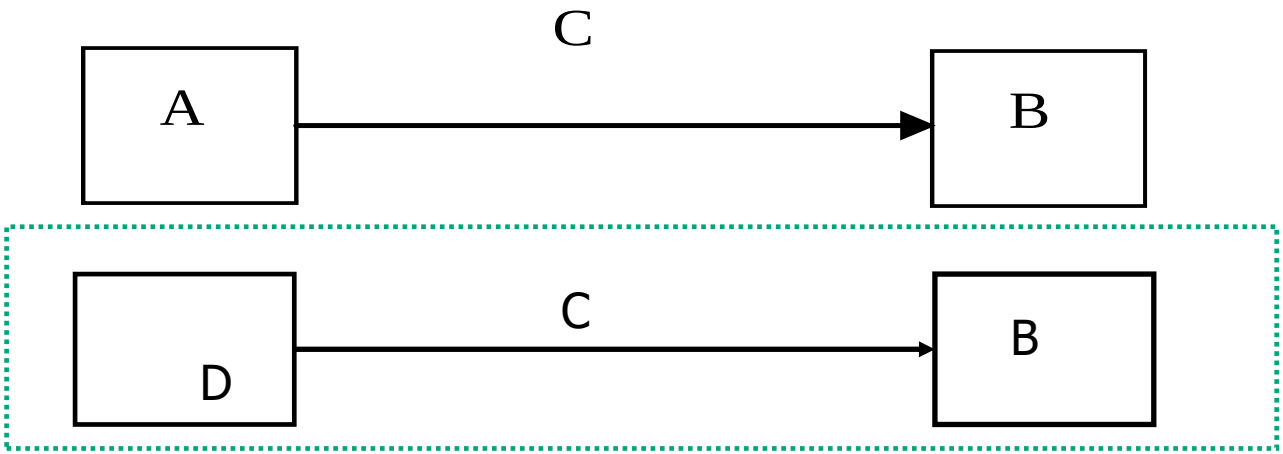


图 管道-过滤器的可扩展性

2.2.1.2 管道-过滤器风格优点

（4）支持特殊的分析：如吞吐量计算和死锁检测等。利用管道-过滤器模式图，可以很容易地得到系统的资源使用与请求状态图，然后，根据操作系统原理等相关理论中的死锁检测方法就可以分析出系统目前所处的状态，是否存在死锁可能及如何消除死锁等问题。

（5）支持并发执行。基于管道-过滤器模式的系统存在很多并行的过滤器，这样的系统在实际运行时，可以将存在并发可能的多个过滤器看作多个并发的任务并行执行，从而大大提高了系统的整体效率，加快了处理速度。当然，在调度并行任务的时候，必须有相应的并行算法作基础，否则，可能导致系统功能的混乱。

2.2.1.3 管道-过滤器风格不足

- 交互式处理能力弱

  管道-过滤器模型**适于数据流的处理和变换，不适合为与用户交互频繁的系统建模**。在这种模型中，每个过滤器都有自己的数据，这些数据或者是从磁盘存储器中读取来，或者是由另一个过滤器的输出导入进来，整个系统没有一个共享的数据区。这样，当用户要操作某一项数据时，要涉及到多个过滤器对相应数据的操作，其实现较为复杂。对于以上的缺点，可以对每个过滤器增加相应的用户控制接口，使得外部可以对过滤器的执行进行控制。

2.2.1.3 管道-过滤器风格不足

- 管道-过滤器风格往往导致系统**处理过程的成批操作**。
- 设计者也许不得不花费精力协调两个相对独立但又存在某种关系的数据流之间的关系，例如多过滤器并发执行时数据流之间的**同步问题**等。
- 根据实际设计的需要，设计者也需要**对数据传输进行特定的处理**（如为了防止数据泄漏而采取加密等手段），导致过滤器必须对输入、输出管道中的数据流进行解析或反解析，增加了过滤器具体实现的复杂性。

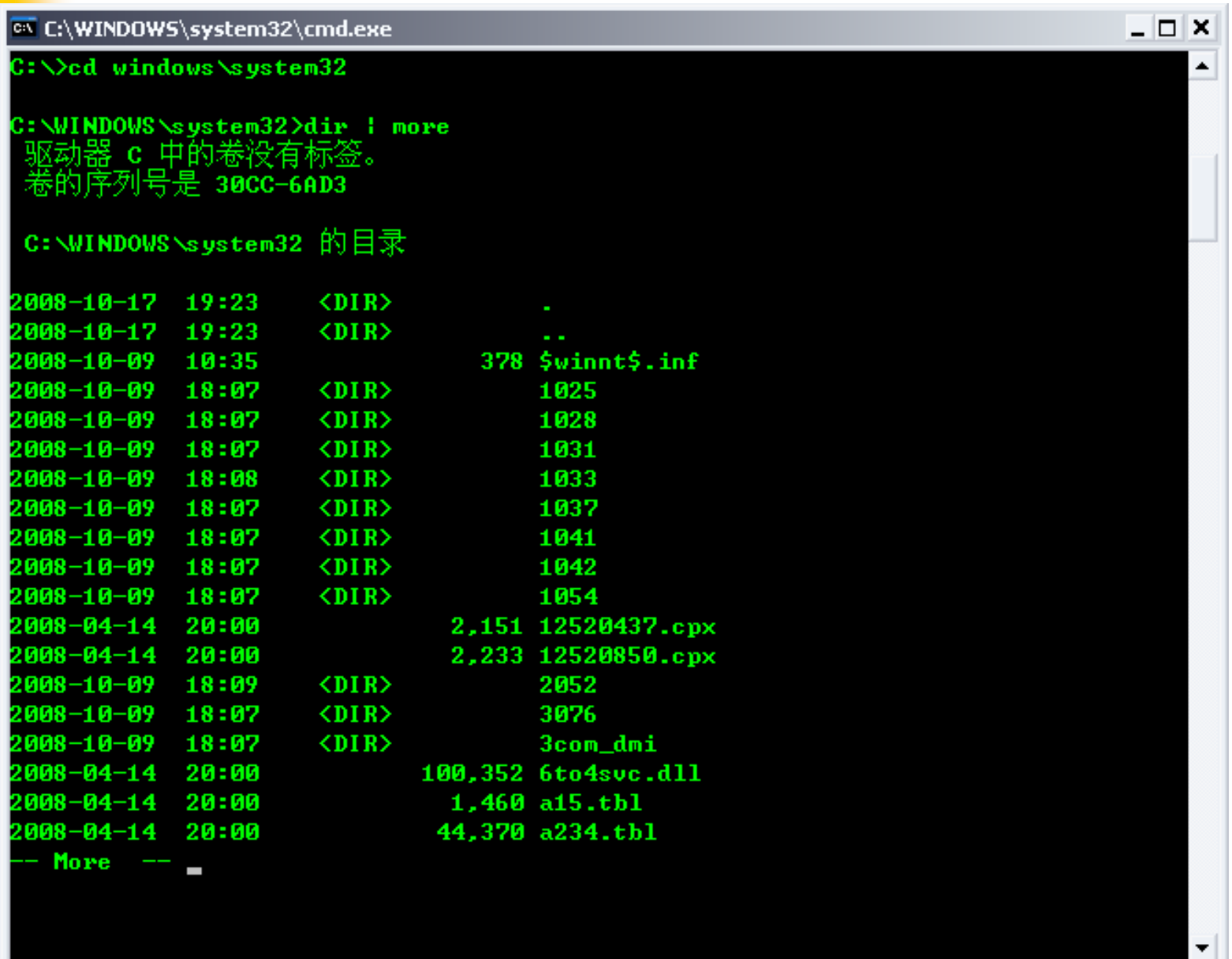
2.2.1.4 管道-过滤器风格实例

实例1： DOS 中的管道命令

- DOS允许在命令中出现用竖线字符“|”分开的多个命令，将符号“|”之前的命令的输出，作为“|”之后命令的输入，这就是“管道功能”，竖线字符“|”是管道操作符。
- 例如，命令dir | more使得当前目录列表在屏幕上逐屏显示。dir的输出是整个目录列表，它不出现在屏幕上而是由于符号“|”的规定，成为下一个命令more的输入，more命令则将其输入，more命令则将其输入一屏一屏地显示，成为命令行的输出。

2.2.1.4 管道-过滤器风格实例

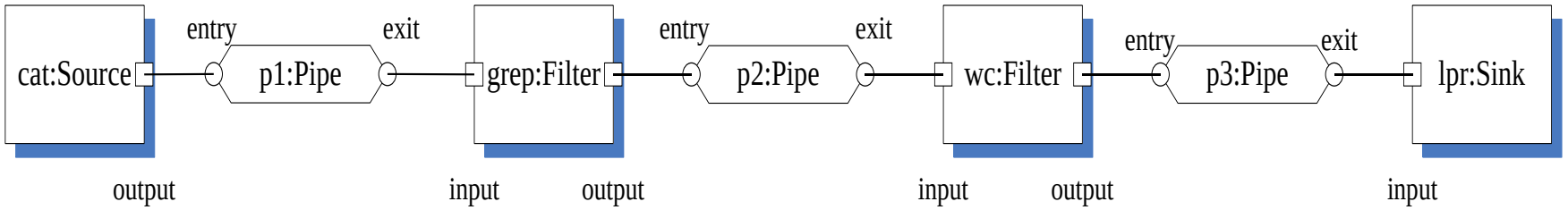
dir | more



2.2.1.4 管道-过滤器风格实例

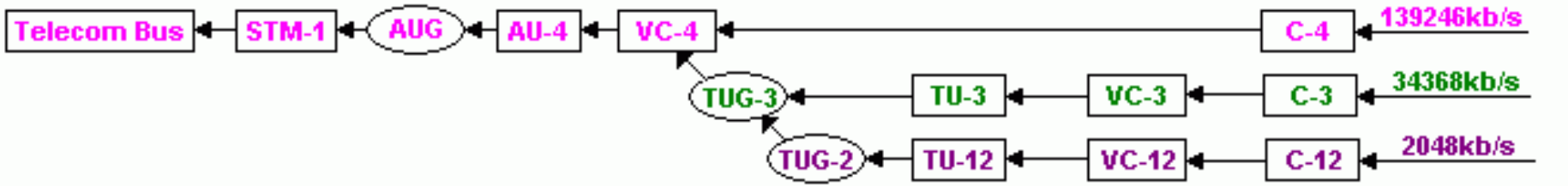
实例2：Unix系统中的管道过滤器结构

“cat a.txt | grep -o soft | wc -w | lpr”



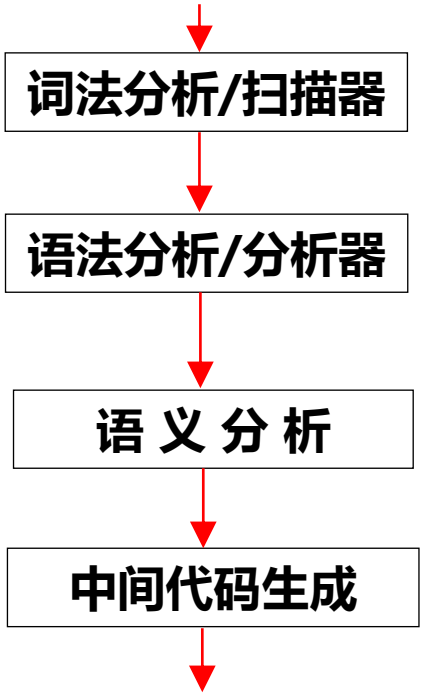
2.2.1.4 管道-过滤器风格实例

实例**3**：通讯协议的信息封装(e.g. SDH)



2.2.1.4 管道-过滤器风格实例

- **实例4：**编译系统。一个普通的编译系统包括词法分析器，语法分析器，语义分析与中间代码生成器，优化器，目标代码生成器等一系列对源程序进行处理的过程。可以将编译系统看作一系列过滤器的连接体，按照管道-过滤器的体系结构进行设计。



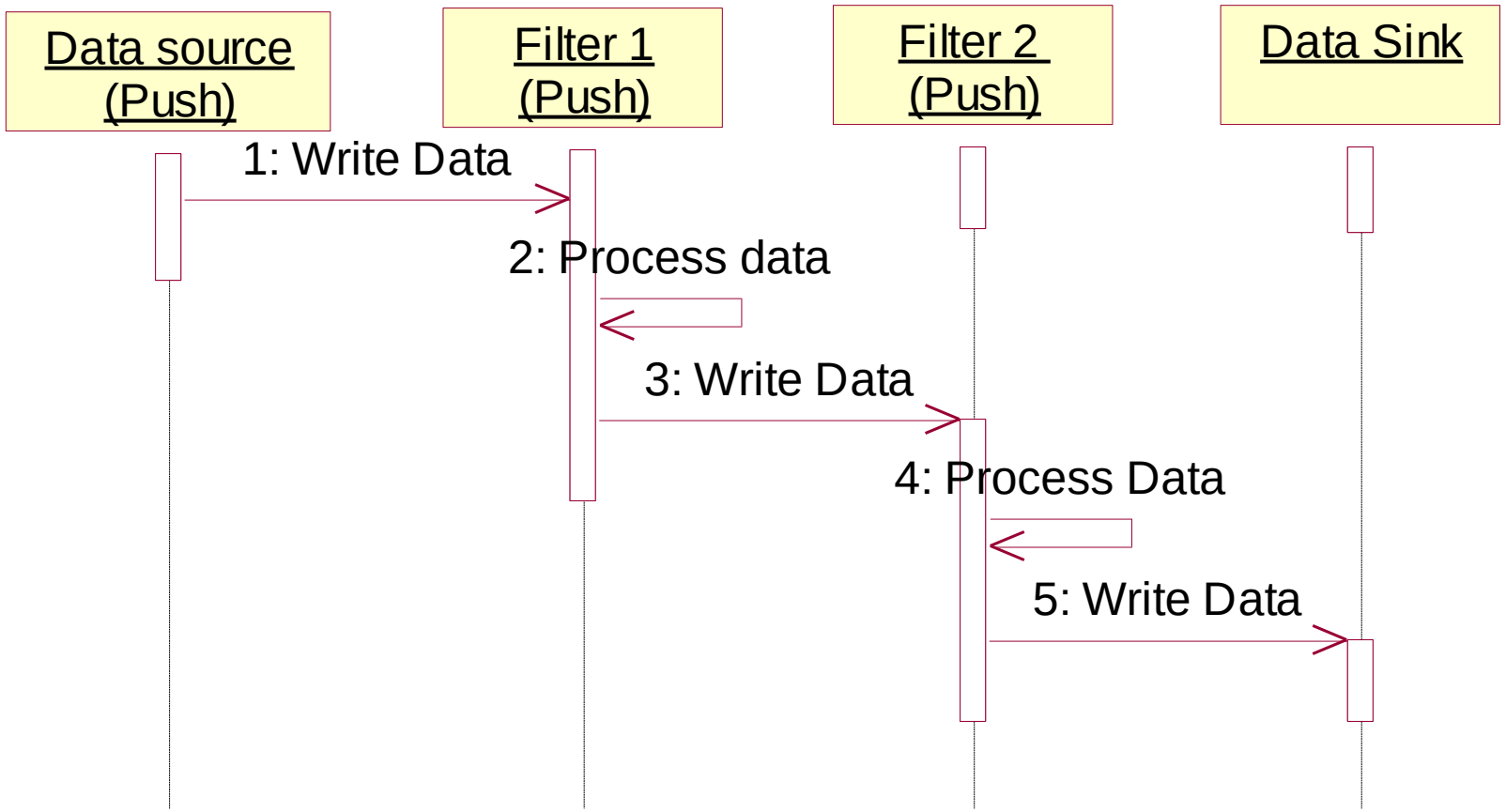
•编译器

2.2.1.4 管道-过滤器风格实例

实例5： 数字图像处理系统

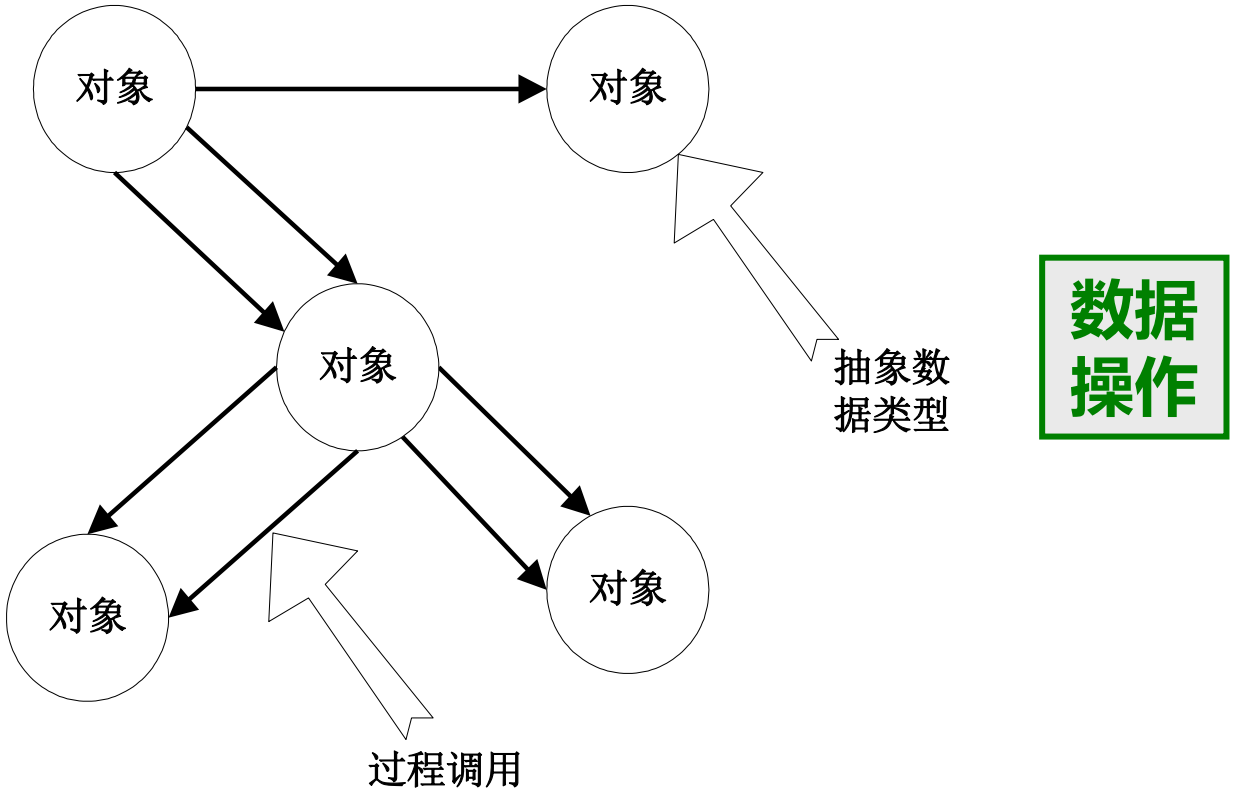
- 图像捕捉
- 灰度化
- 拉伸
- 边缘处理
- 模板匹配
- 识别

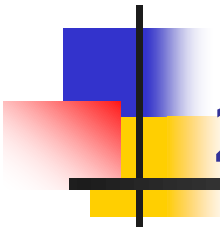
用UML表示的一个过滤器模型顺序图



2.2.2 面向对象风格

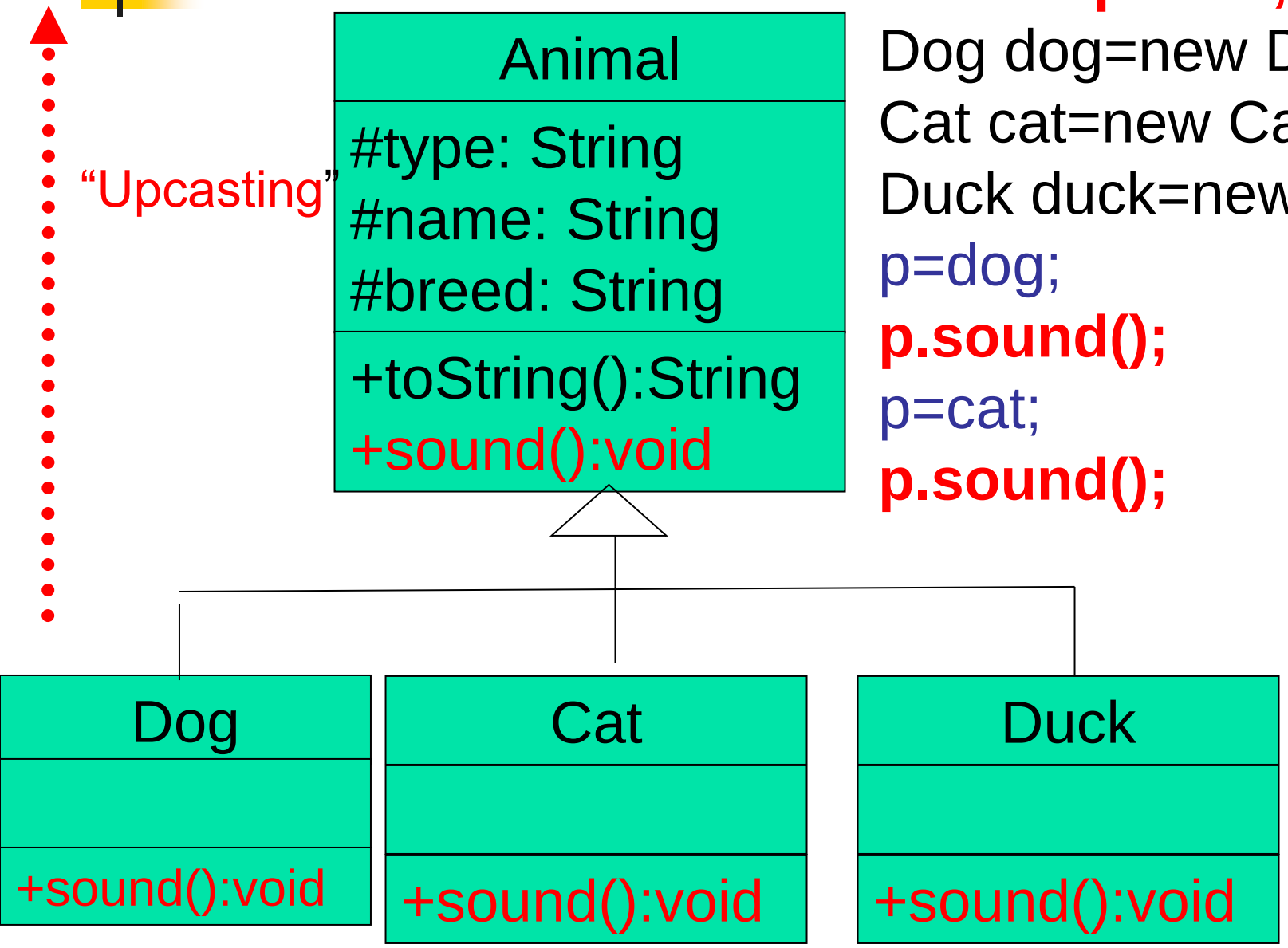
- 构件：类、对象、接口
- 连接件：类或者对象间的过程(方法)调用





2.2.2.1 面向对象风格特点

抽象、
封装、
继承、
多态



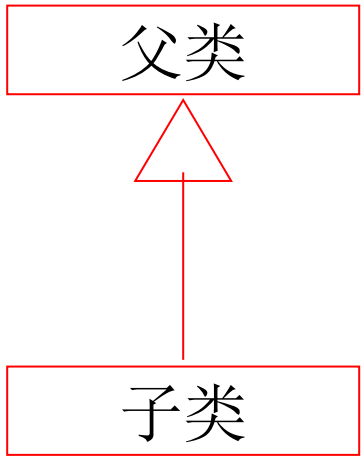
Animal p=null;
Dog dog=new Dog();
Cat cat=new Cat();
Duck duck=new Duck();
p=dog;
p.sound();
p=cat;
p.sound();

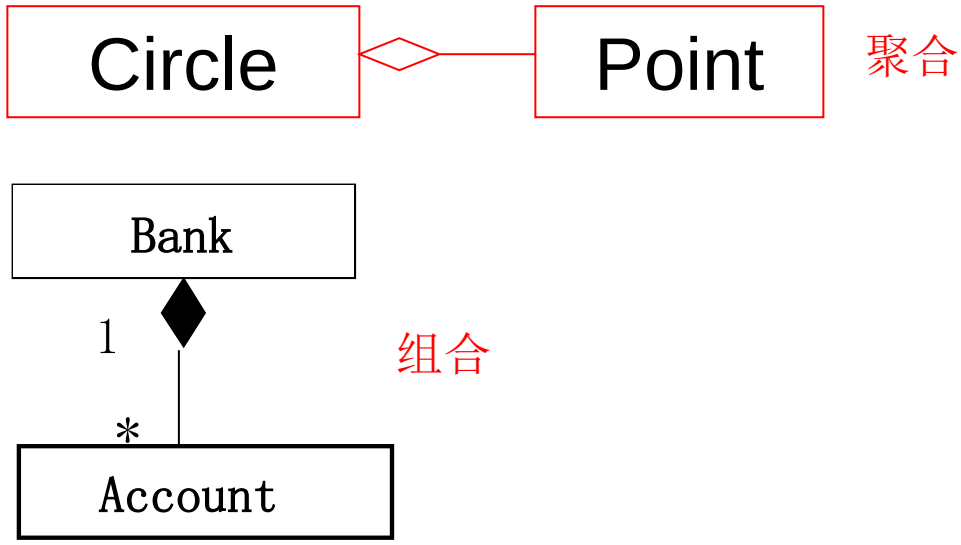
2.2.2.2面向对象风格优点

- 高度模块性
  - 数据与其相关操作被组织为对象， 成为模块组织的基本单位
- 封装功能
  - 一组功能和其实现细节被封装在一个对象中，具有功能的接口被暴露出来
- 重用
  - 对象的相对独立性可被反复重用，通过拼装形成不同的软件系统
- 灵活性
  - 对象在组织过程中，相互关系可以任意变化，只要接口兼容
- 易维护性
  - 对象接近于人对问题和解决方案模型的思维方式，易于理解和修改

2.2.2.3面向对象风格缺点

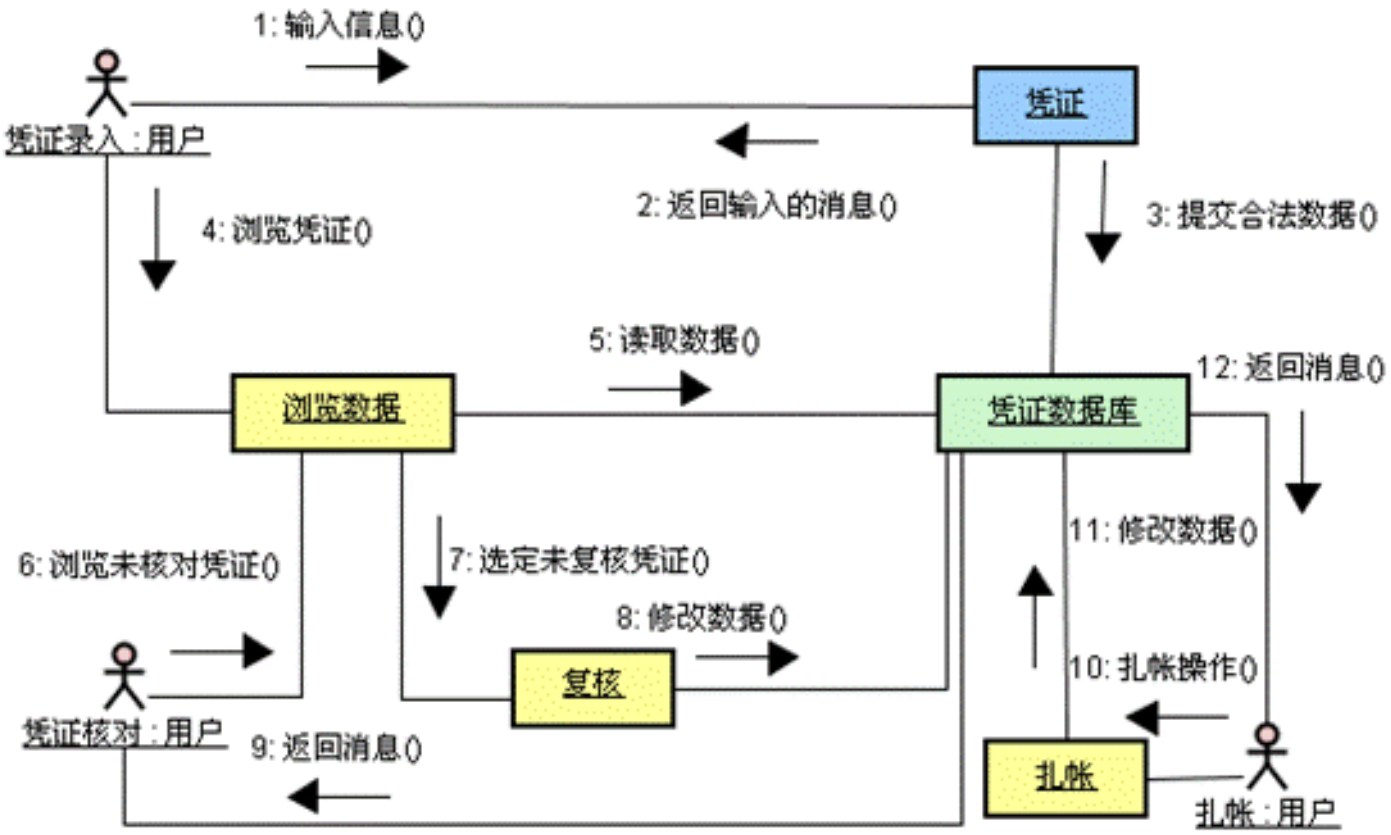
- 继承关系导致树状层次，该层次不利于系统的理解和修改。
  - 在关键系统中，慎用继承，而用聚合（组合）。





2.2.2.4 面向对象风格的系统设计基本原则

- 将逻辑上的实体映射为对象，实体之间的关系映射为对象之间的关系。
- 对象利用关系来访问对方公开的接口，完成某个特定任务；
- 一组对象之间相互协作，完成总体目标。



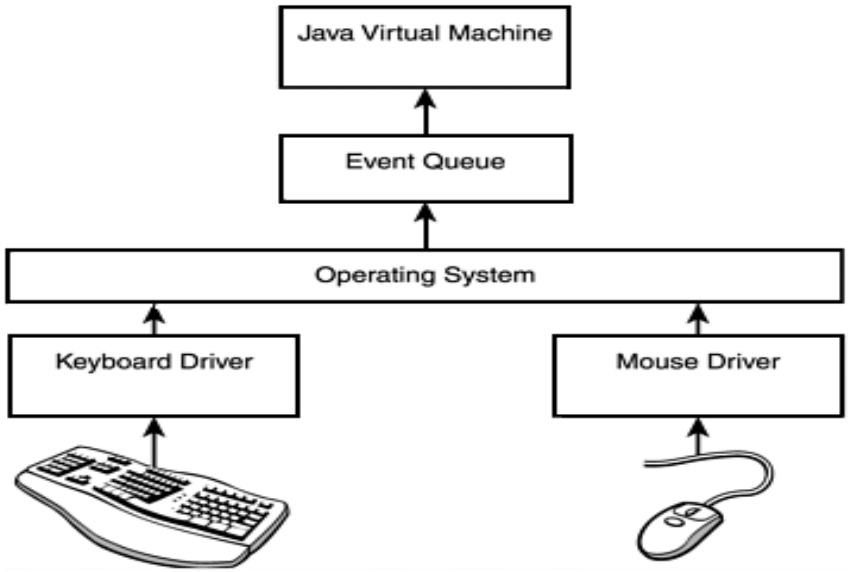
一个银行系统对象之间的协作实例

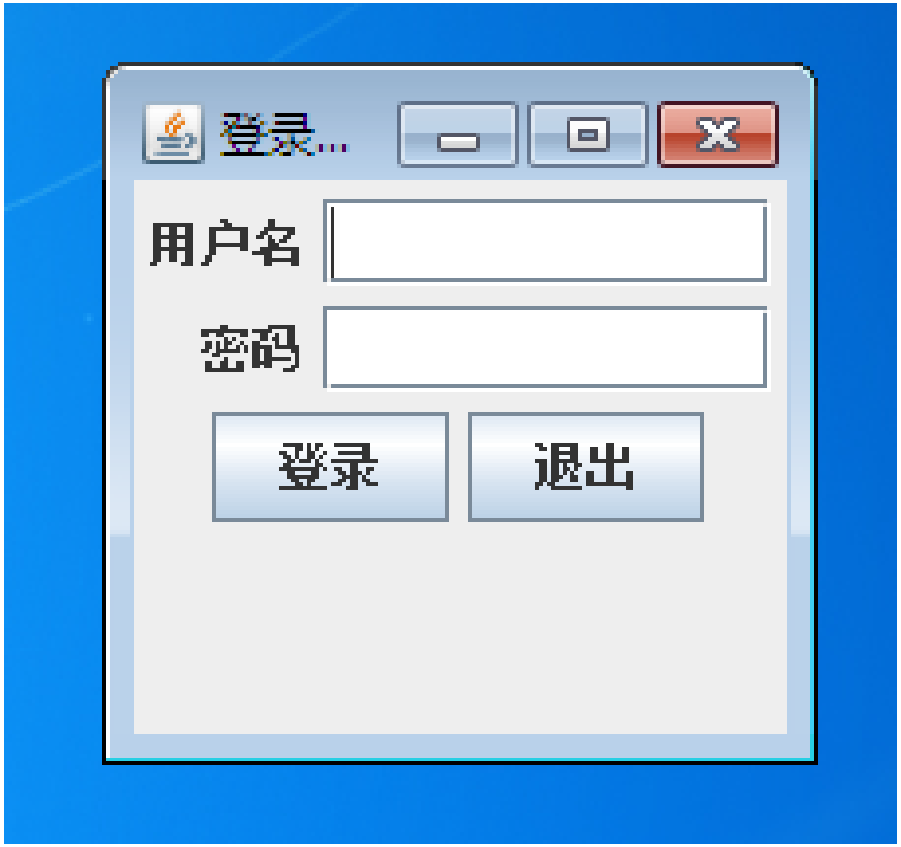
2.2.3 基于事件的隐式调用风格

- 面向对象风格缺点：对象名（ID）的依赖性。
  调用另外一个对象的方法，需知道该对象的名字：
  tom.getName();

2.2.3.1 Java 的事件驱动

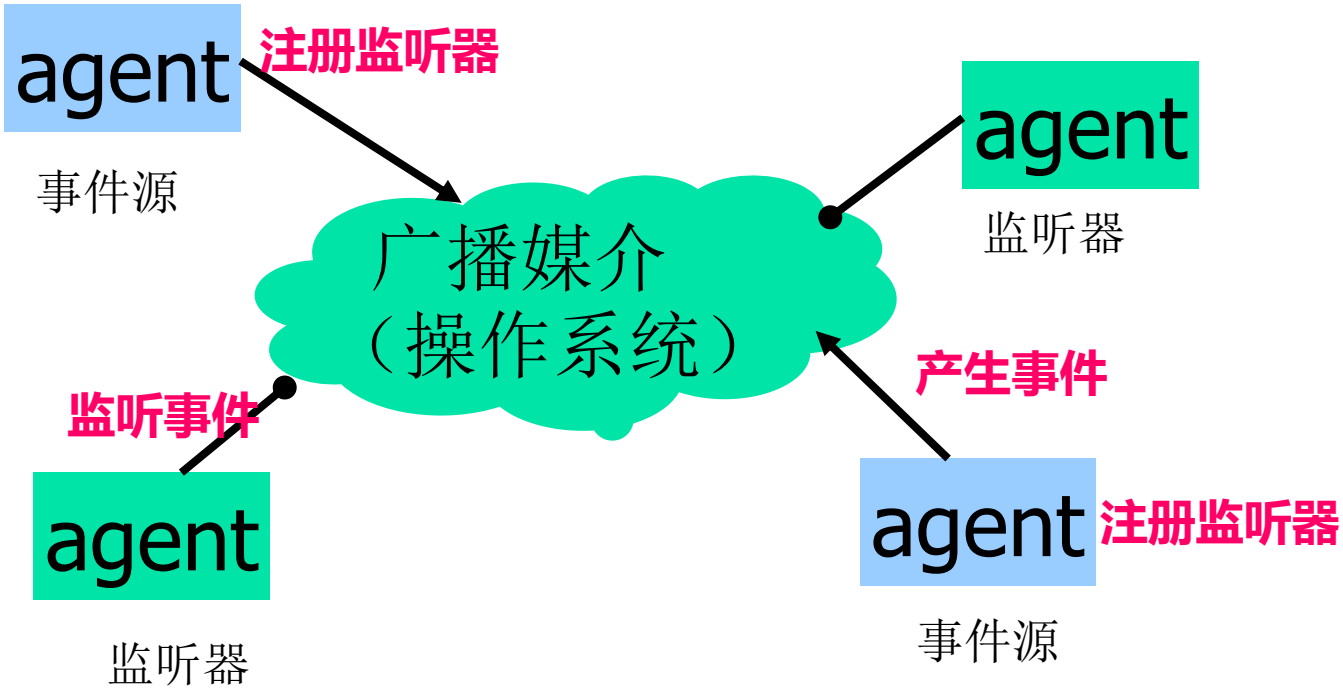
- Java GUI应用程序和Applet程序是基于事件驱动的



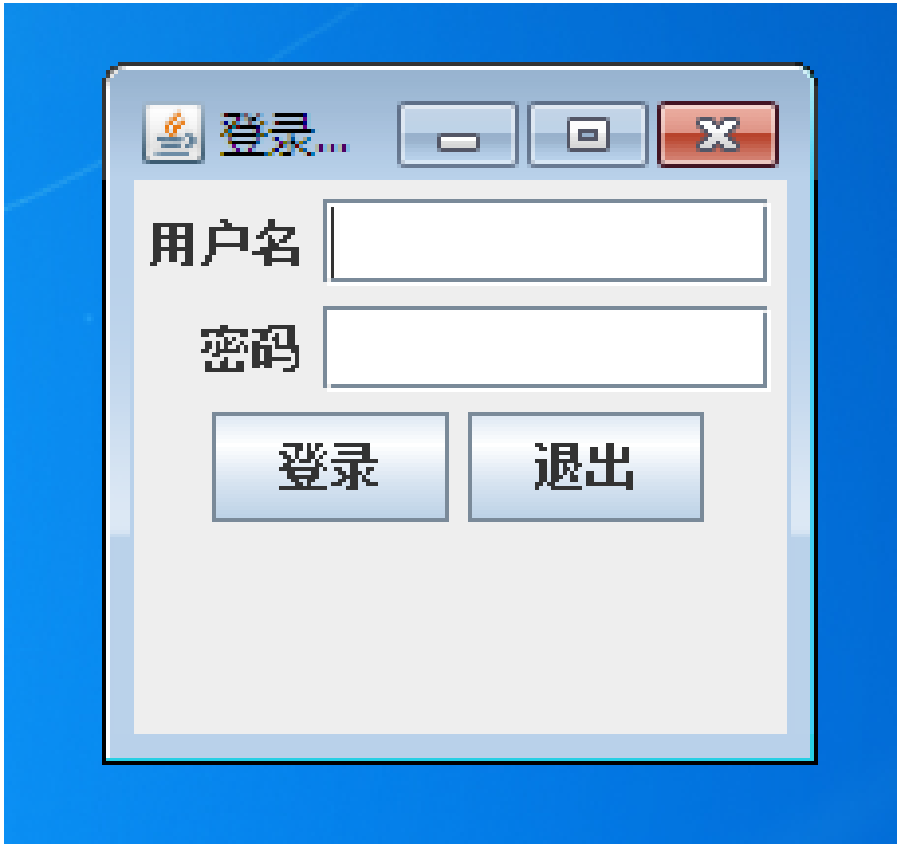

登录...
用户名
密码
登录
退出

- 构件：类，对象

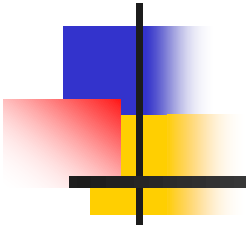
连接件：事件源注册监听器



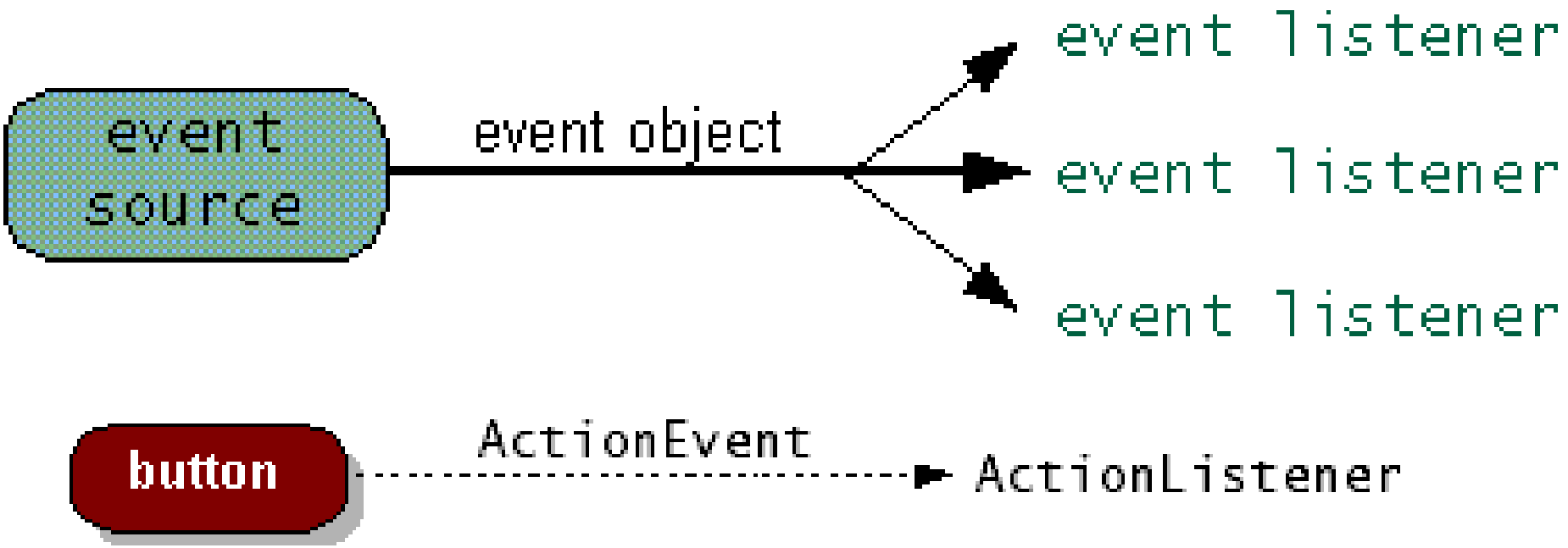
Java委托事件处理机制



登陆和退出构件注册了相应的监听器

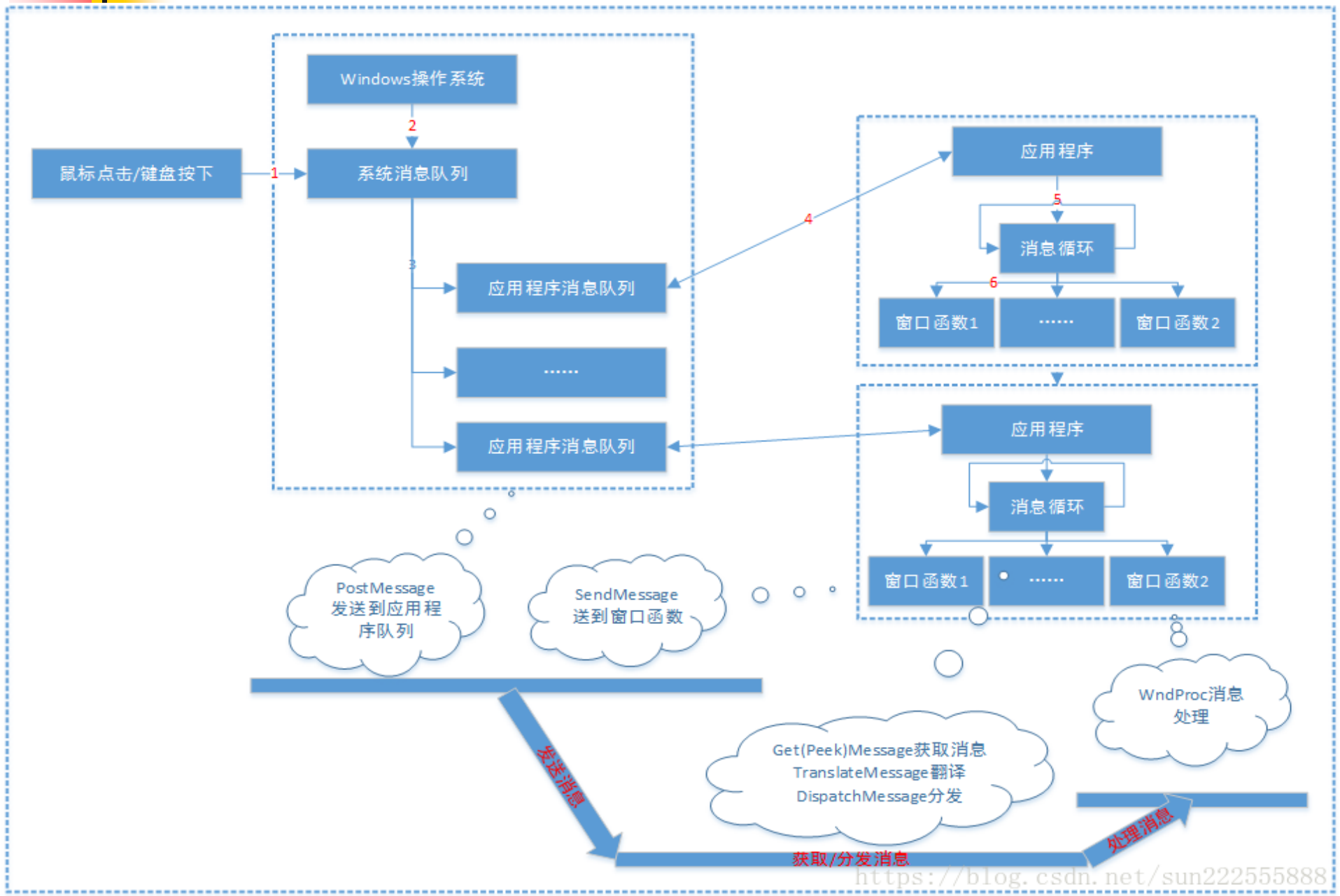


- 基于事件的隐式调用风格的思想是
  - 构件（事件源：按钮、窗口等）不直接调用一个过程，而是触发或广播一个或多个事件（点击鼠标左键，关闭窗口）。
  - 系统中的其他构件（监听器）监听该事件
  - 事件源构件注册监听器构件
  - 当一个事件被触发，系统自动调用监听这个事件的所有监听器相应方法。这样，一个构件（事件源：按钮、窗口等）产生的事件触发了另一构件（监听器）中的过程的调用。


event listener
event source
event object
event listener
event listener
button
ActionEvent
ActionListener

Login.java

2.2.3.2 windows消息机制(win32编程) Windows.cpp



2.2.3.3 隐式调用系统的优点

- 为软件重用提供了强大的支持。当需要将一个构件加入现存系统时，只需将它注册到系统的事件中。
- 为改进系统带来了方便。当用一个构件代替另一个构件时，不会影响到其他构件的接口。

2.2.3.4 隐式调用系统的缺点

- 构件放弃了对系统计算的控制。
  - 一个构件触发一个事件时．不能确定其他构件是否会响应它。而且即使它知道事件注册了哪些构件的过程，它也不能保证这些过程被调用的顺序。
- 数据交换的问题。
  - 大数据量的数据交互往往没法用事件携带
- 可能会对正确性保证带来问题
  - 对事件触发的方法调用次序无法控制

2.2.4 分层系统风格

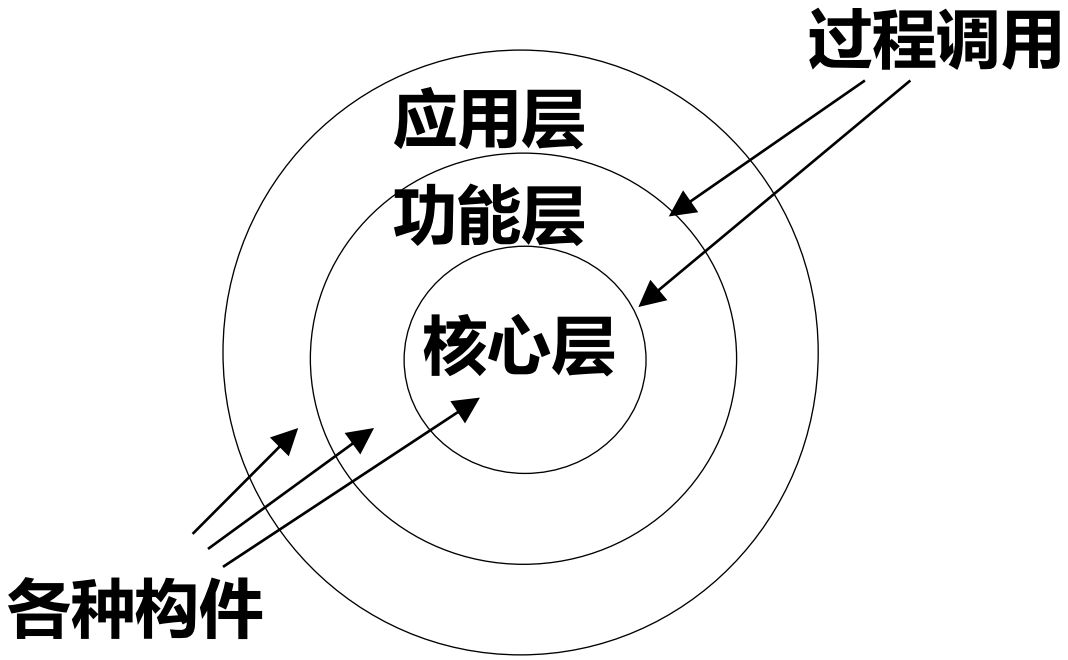
分层系统风格体现出一种层次结构

- 每一层都向它的上一层提供服务
- 每一层都使用它的下一层的服务

下层无需知道上层的存在，每一层对其上层隐藏其实现细节

分层模型

- 构件：各层
- 连接件：定义层间交互的协议



分层风格优点

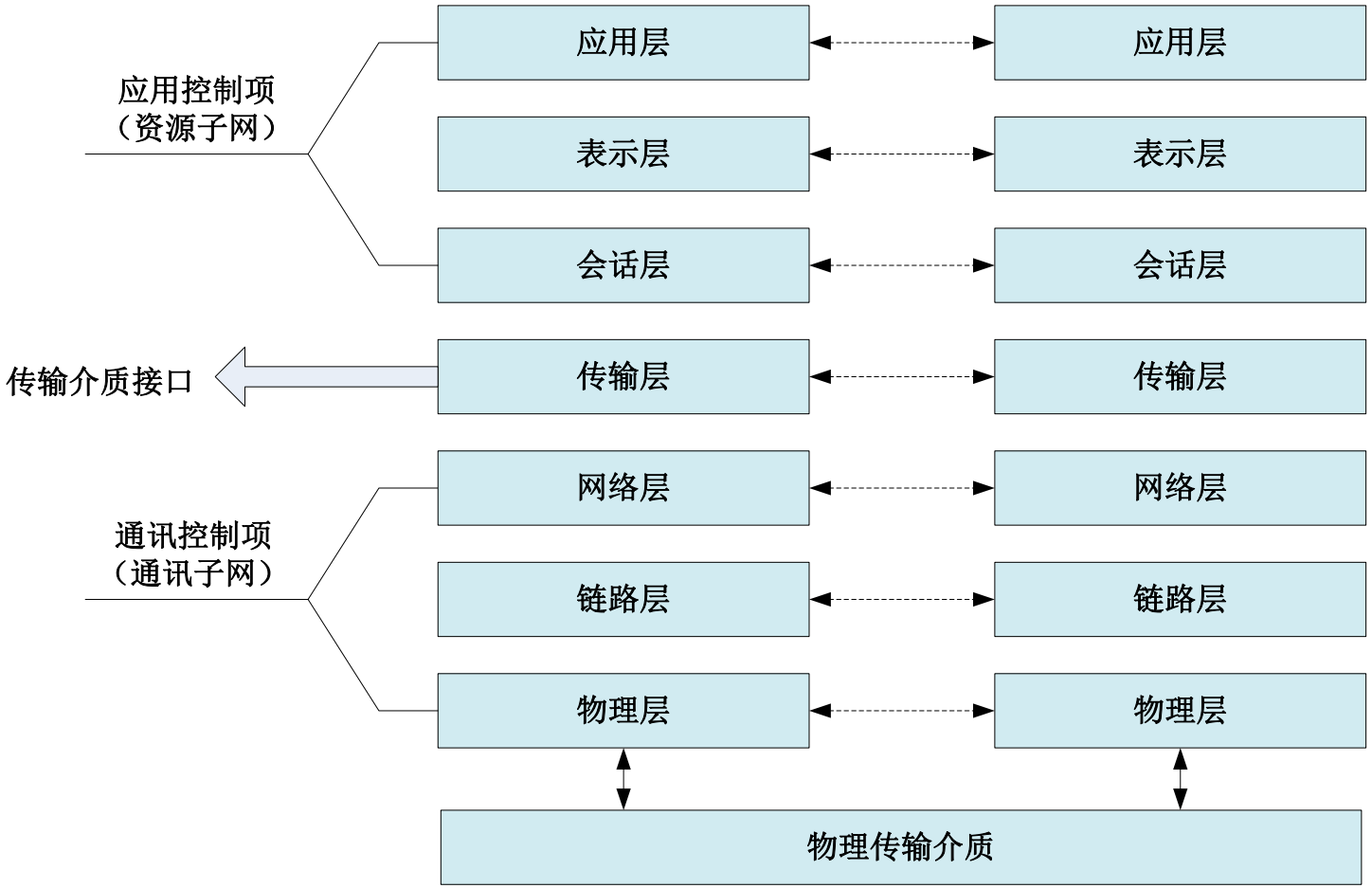
- 分层风格便于将复杂系统进行分解
- 基于分层风格的系统具有较好的可扩展性
  - 只要保持接口一致，就可以将某一层的构件替代，而不会影响到系统的其他部分
- 分层风格支持软件复用

分层风格不足

- 并不是所有的系统都适合用分层风格来描述的
- 很难找到合适的、正确的层次抽象方法。
  - 层数太少，分层不能完全发挥这种风格的可复用性、可扩展性等优点。
  - 层数过多，则引入不必要的复杂性和层间传输开销
  - 目前，没有可行的广为人们认可的层粒度和层任务的分配方法。

分层风格实例

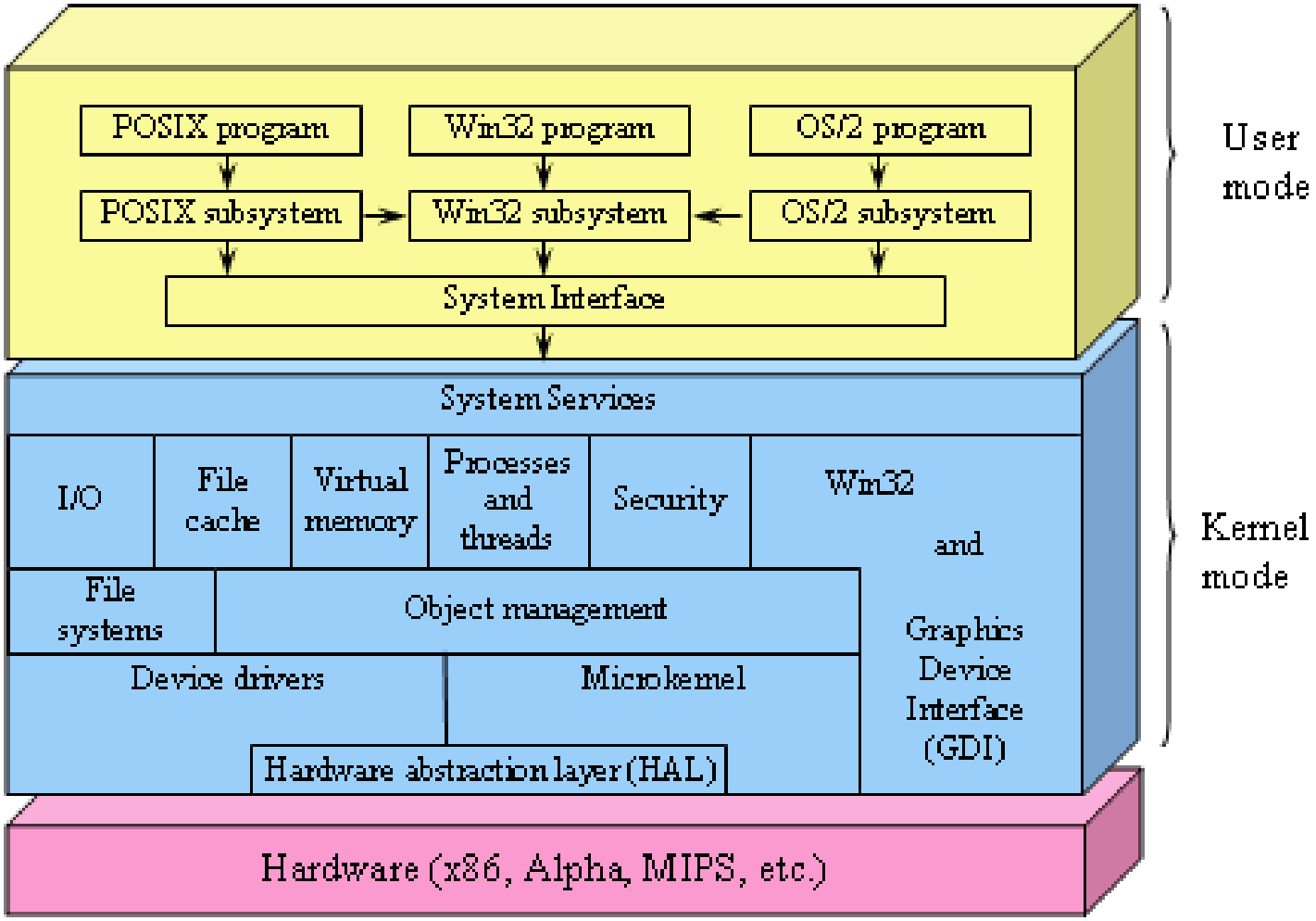
ISO/OSI网络体系结构



ISO/OSI网络体系结构

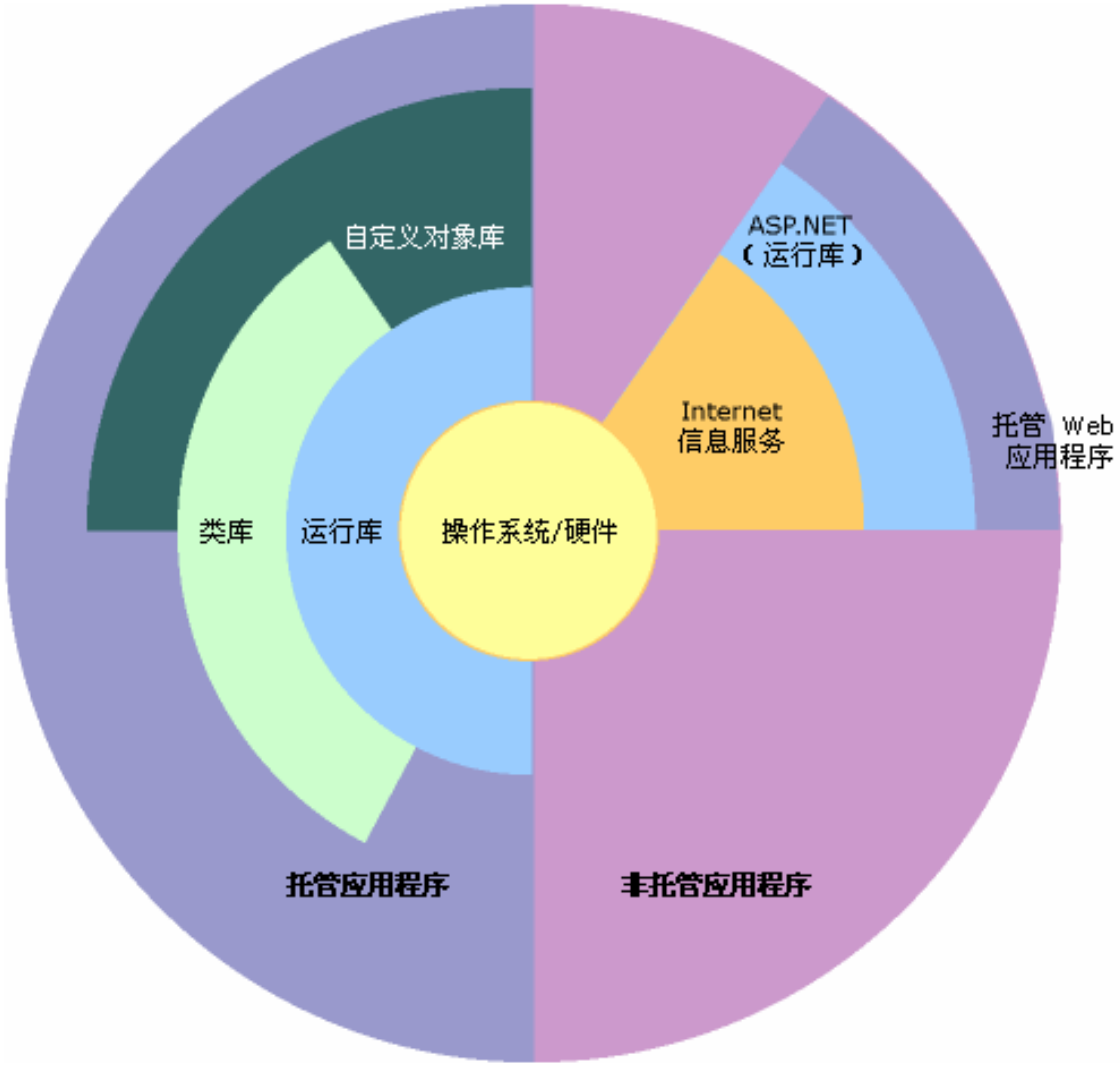
分层风格实例

操作系统的分层结构



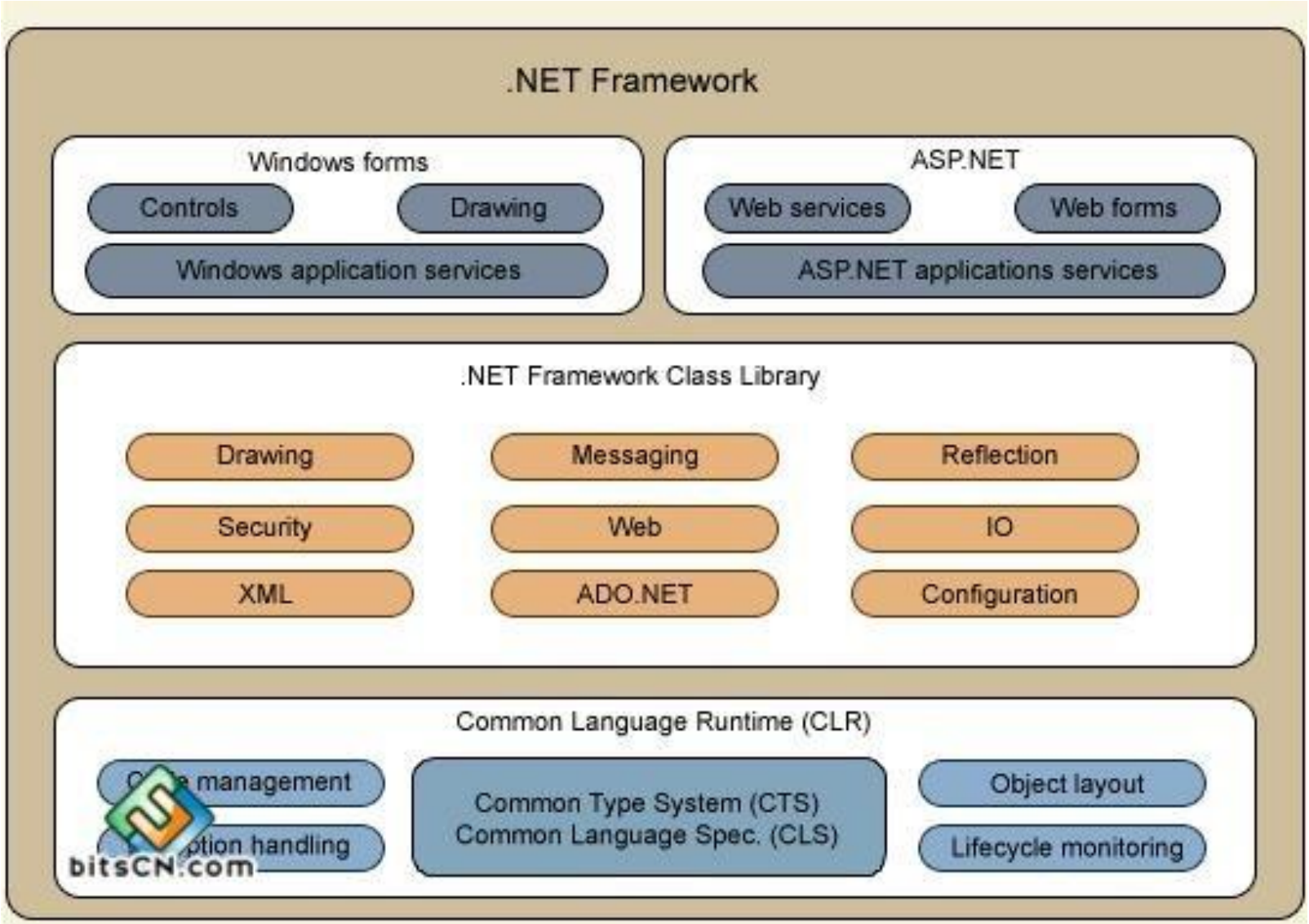
分层风格实例

.Net平台也是一个明显的分层系统：



分层风格实例

.Net 框架分层模图



分层风格实例 Android系统框架

Android系统采用了分层的架构，总共四层，如图所示，由上到下分别是应用程序层、应用程序框架层、系统运行库层和Linux内核层，每一层都使用其下面各层所提供的服务。

2.2.5客户/服务器风格

◇ 产生背景

在集中式计算技术时代广泛使用的是大型机/小型机计算模型。它是通过一台物理上与宿主机相连接的非智能终端来实现宿主机上的应用程序。

20世纪80年代以后，集中式结构逐渐被以PC机为主的微机网络所取代。个人计算机和工作站的采用，永远改变了协作计算模型，从而导致了分散的个人计算模型的产生。

◇ 基本概念

客户/服务器（client/server，C/S）计算机技术在信息产业中占有重要的地位。

C/S软件体系结构是基于资源不对等，且为实现共享而提出来的，是20世纪90年代成熟起来的技术，C/S体系结构定义了工作站如何与服务器相连，以实现数据和应用分布到多个处理机上。

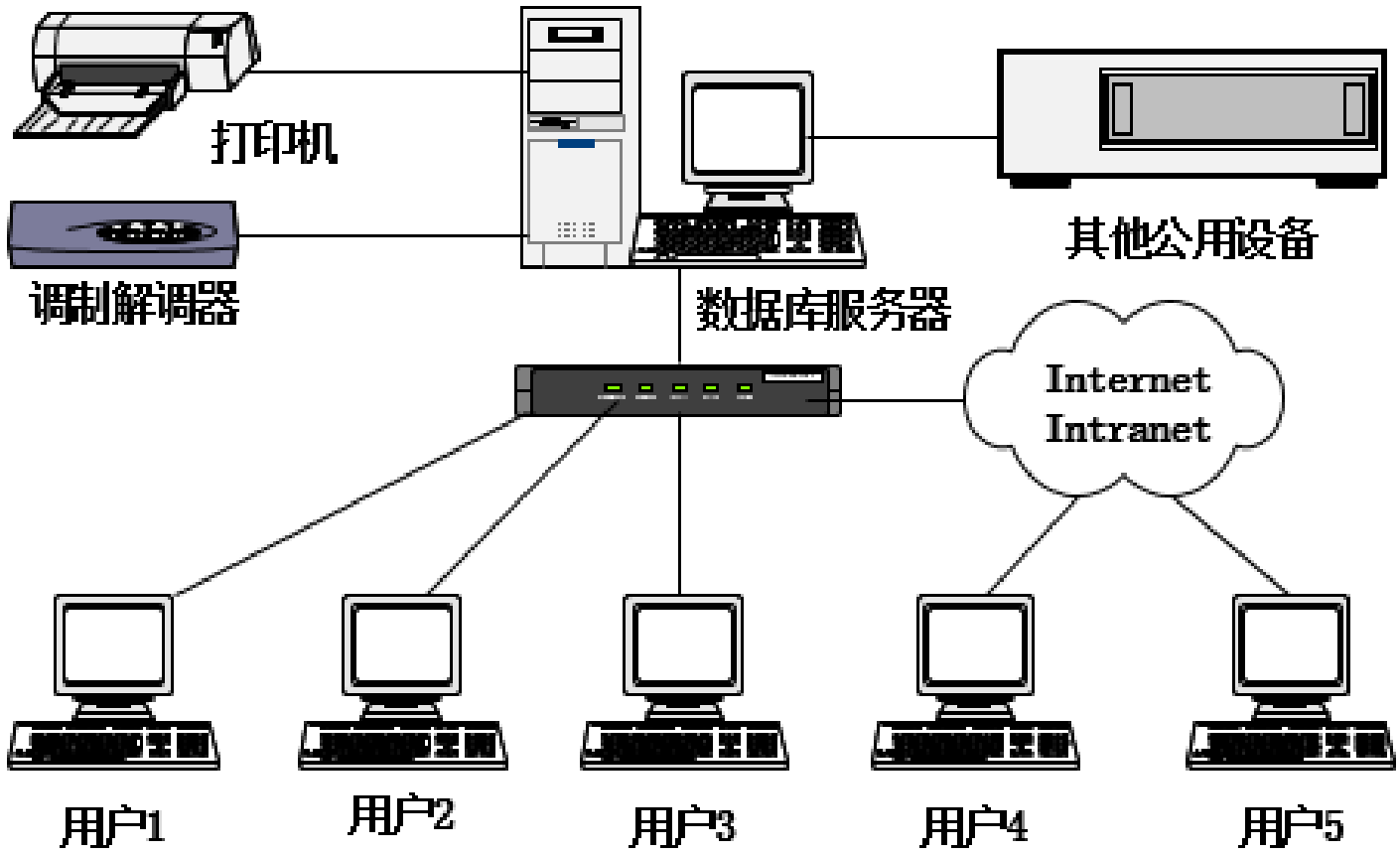
C/S体系结构有三个主要组成部分(构件、连接件)：

数据库服务器

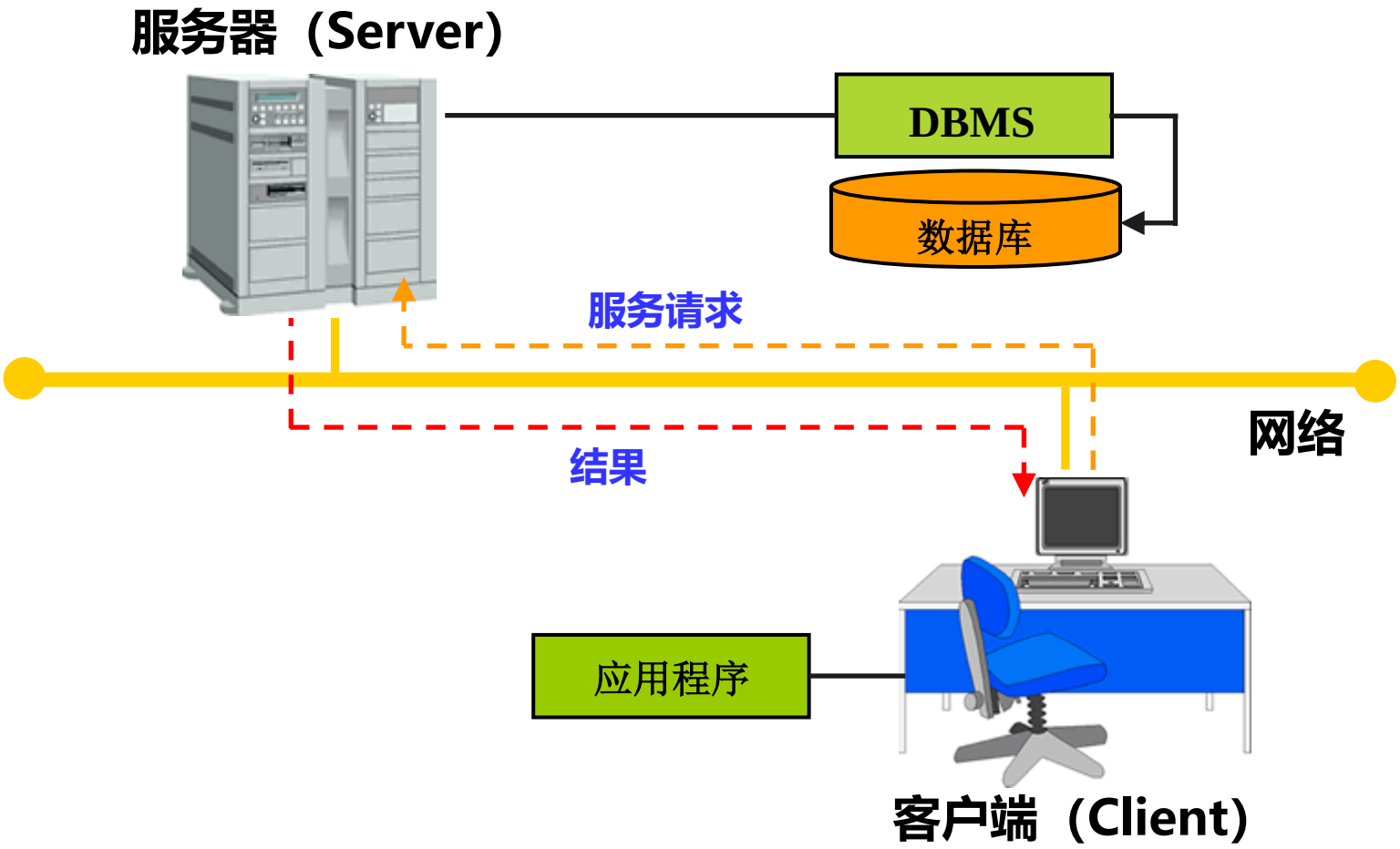
客户应用程序

网络

◇ 系统拓扑结构


打印机
调制解调器
数据库服务器
其他公用设备
Internet
Intranet
用户1
用户2
用户3
用户4
用户5

◇ 工作原理



◇ 任务分配

服务器（服务器构件）：

(1) 数据库安全性的要求；

(2) 数据库访问并发性的控制；

(3) 数据库前端的客户应用程序的全局数据完整性规则；

(4) 数据库的备份与恢复。

◇ 任务分配

客户应用程序（客户构件）：

(1) 提供用户与数据库交互的界面；

(2) 向数据库服务器提交用户请求并接收来自数据库服务器的信息；

(3) 利用客户应用程序对存在于数据库服务器端的数据执行应用逻辑要求。

网络（连接件）：

完成数据库服务器和客户应用程序之间的数据传输，即它是某种进程间的通信机制

◇C/S实例：客户机和服务器socket通信

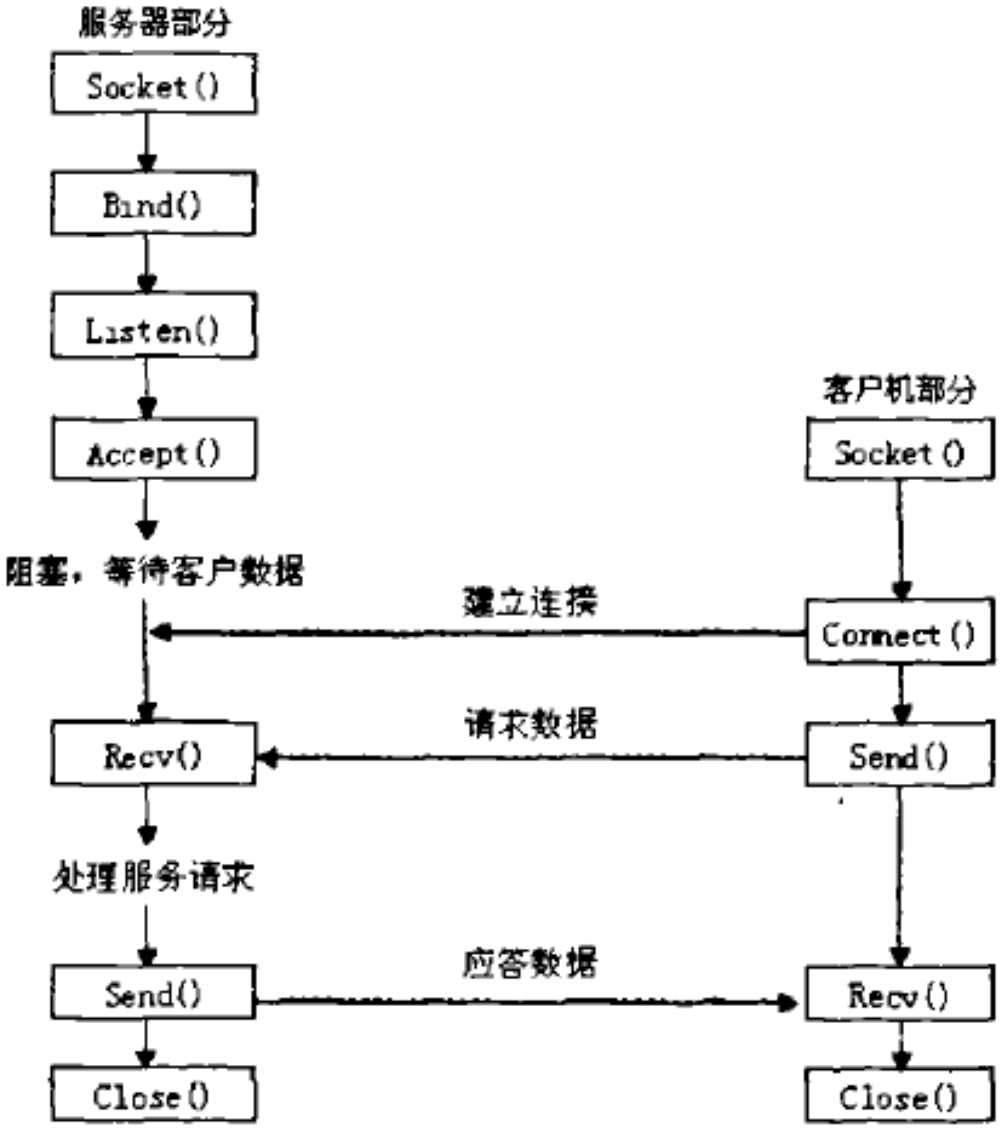
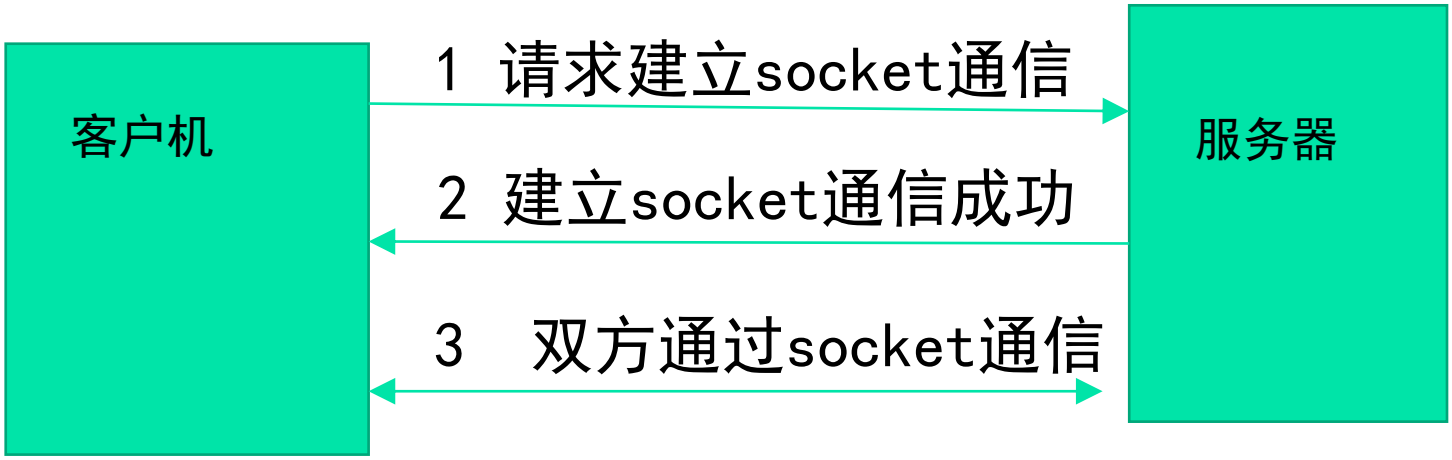


图 2　面向连接 Socket 操作流程

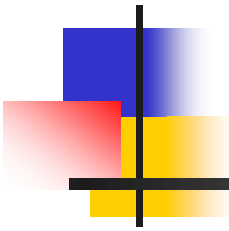
◇C/S实例：客户机和服务器socket通信



TcpClient.java　TcpServer.java

◇C/S实例：并发客户机和服务器socket通信

MulTcpServer.java　TcpClient.client



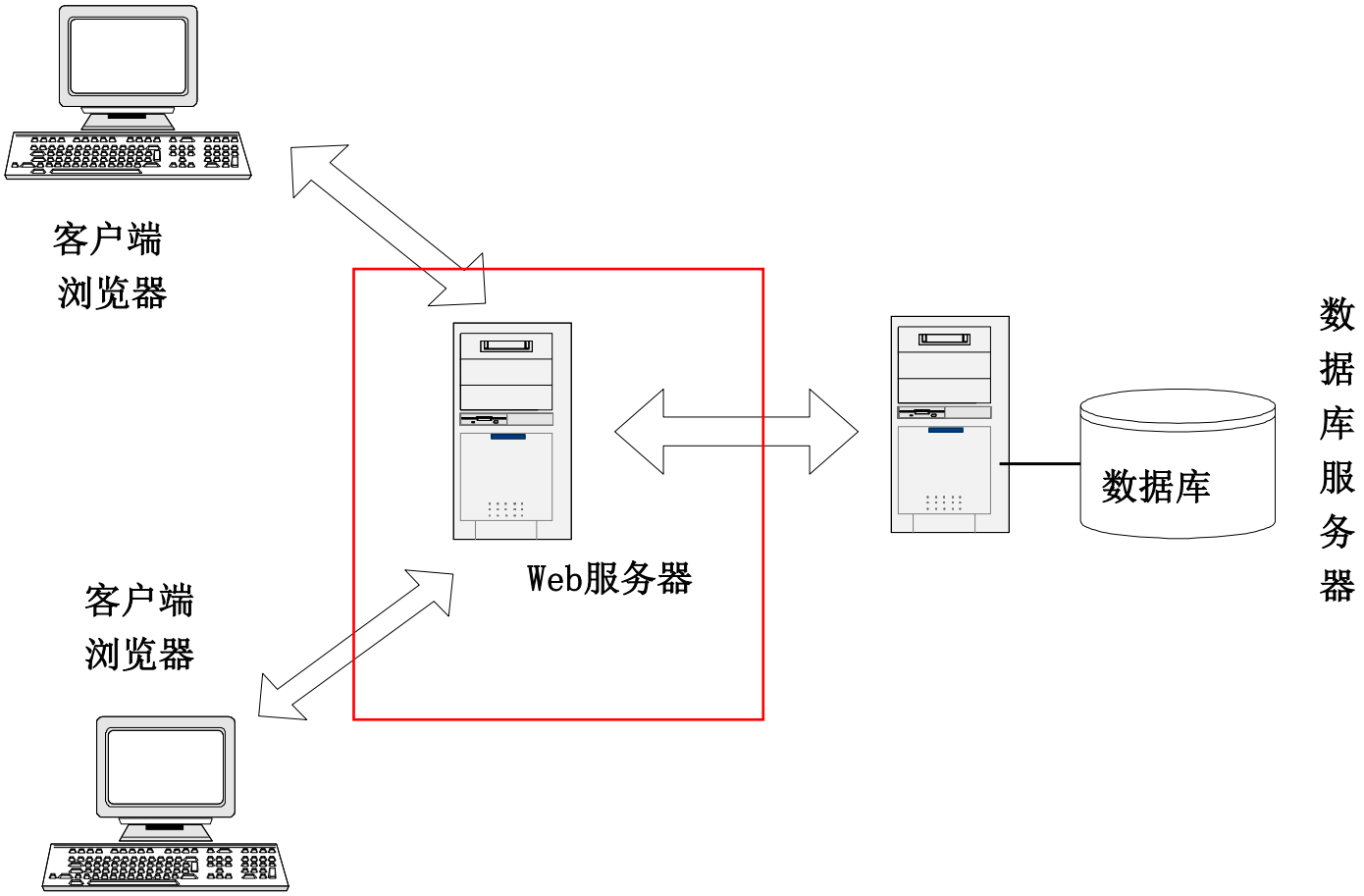
2.2.6 浏览器/服务器风格

浏览器/服务器（Browser/Server，B/S）三层应用结构的一种实现方式，其具体结构为：

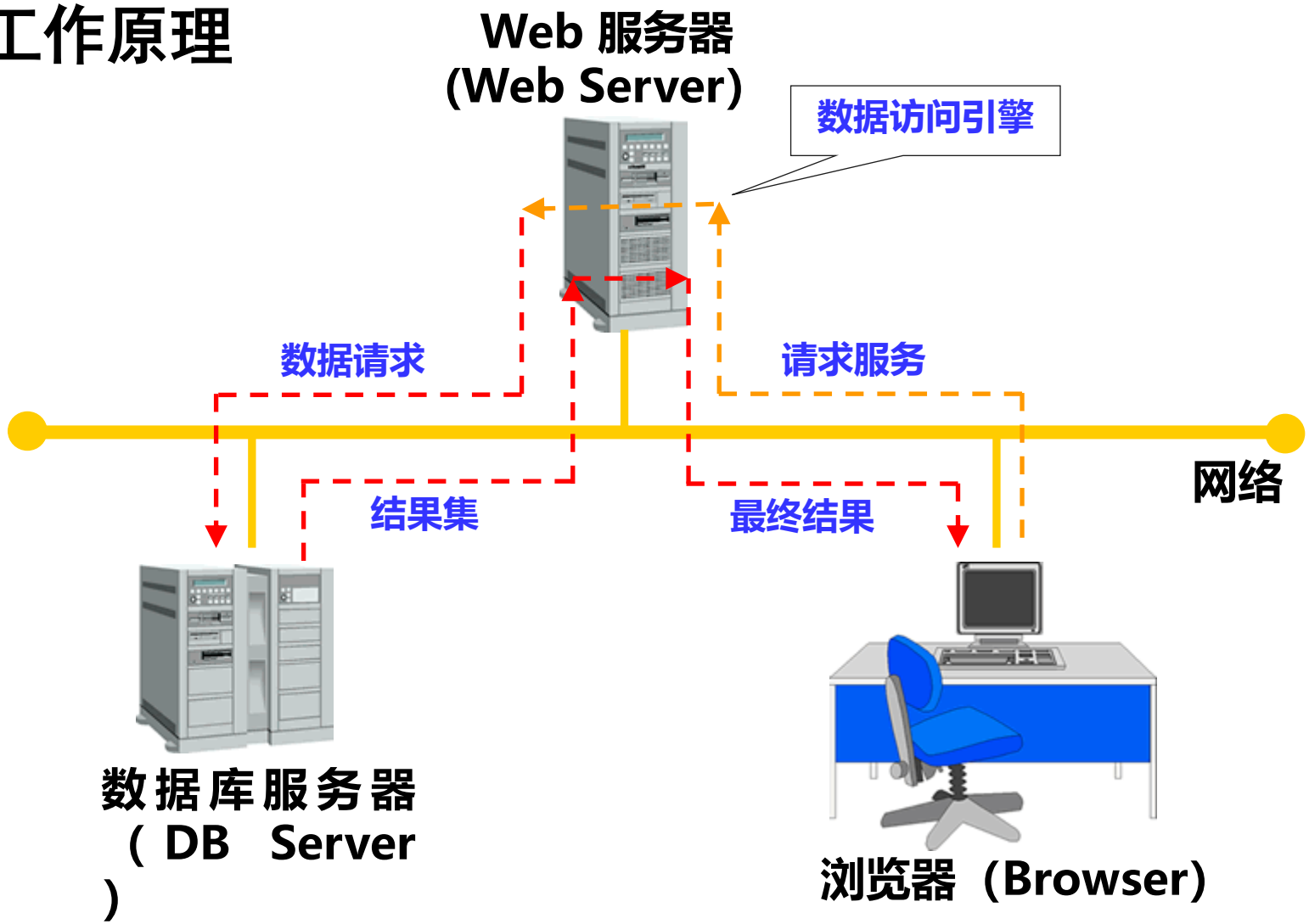
浏览器/Web服务器/数据库服务器。

B/S体系结构主要是利用成熟的WWW浏览器技术，结合浏览器的多种脚本语言，用通用浏览器就实现了原来需要复杂的专用软件才能实现的强大功能，并节约了开发成本。

◇ 网络拓扑结构



◇ 工作原理


Web 服务器
(Web Server)
数据访问引擎
数据请求
请求服务
结果集
最终结果
网络
数据库服务器
（DB Server）
浏览器（Browser）

B/S风格实例：JavaEE Tutorial

具体内容见补充课件

C/S与B/S对比

- C/S的优点：

能充分发挥客户端PC的处理能力，很多工作可以在客户端处理后再提交给服务器，减轻服务器端压力，对应的优点就是客户端响应速度快，如果用户的需求特别复杂，用C/S。

- C/S的缺点：

只适用于网速较快的网络环境如：局域网。随着互联网的飞速发展，移动办公和分布式办公越来越普及，很多情况下都是在不同的网络环境下办公的，而当前形式下网速又相对较慢，所以很多情况下，C/S不能很好的满足业务需求。

客户端需要安装专用的客户端软件及运行环境。首先涉及到安装的工作量，其次任何一台电脑出问题，如病毒、硬件损坏，都需要进行安装或维护。特别是有很多分部或专卖店的情况，不是工作量的问题，而是路程的问题。还有，系统软件升级时，每一台客户机需要重新安装，其维护和升级成本非常高。

C/S与B/S对比

- B/S最大的优点：

可以在任何地方进行操作而不用安装任何专门的软件。只要有一台能上网的电脑就能使用，客户端零维护。适用于用户群庞大，或客户需求经常发生变化的情况。

- B/S缺点：

应用服务器运行数据负荷较重。由于B/S架构管理软件只安装在服务器端（Server）上，网络管理人员只需要管理服务器就行了，用户界面主要事务逻辑在服务器（Server）端完全通过WWW浏览器实现，极少部分事务逻辑在前端（Browser）实现，所有的客户端只有浏览器，网络管理人员只需要做硬件维护。但是，应用服务器运行数据负荷较重，一旦发生服务器“崩溃”等问题，后果不堪设想。因此，许多单位都备有数据库存储服务器，以防万一。

在运行速度、数据安全、人机交互等方面，B/S远不如C/S。

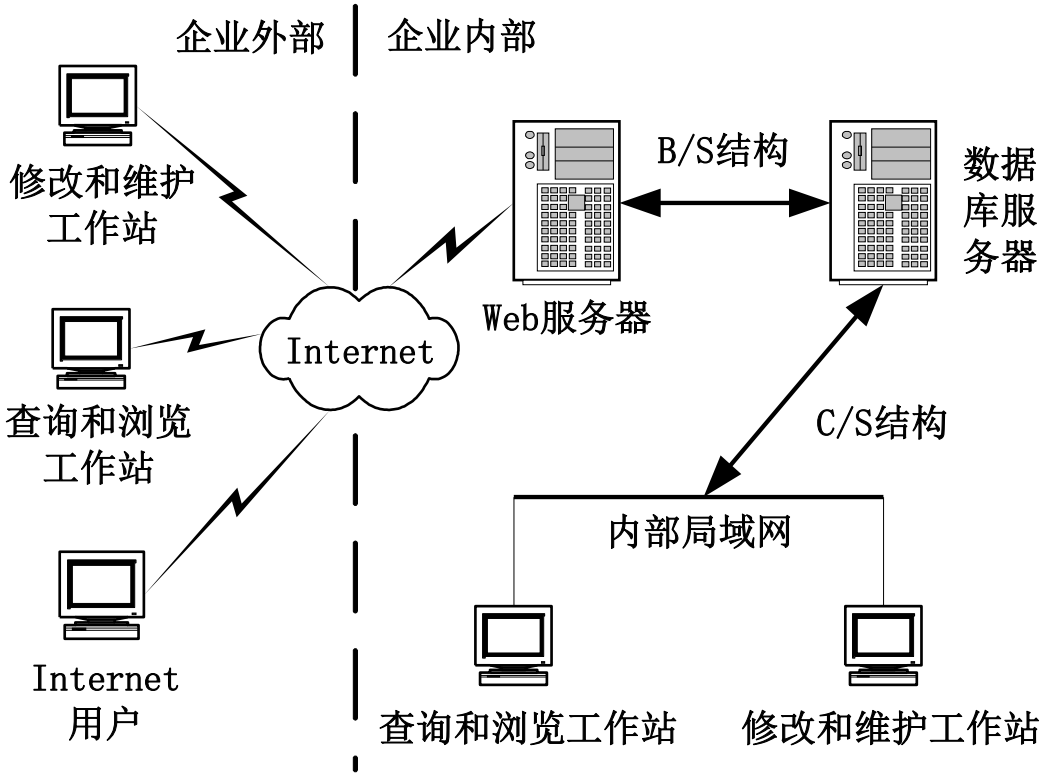
2.2.7 异构的结构风格

1 为什么要使用异构结构

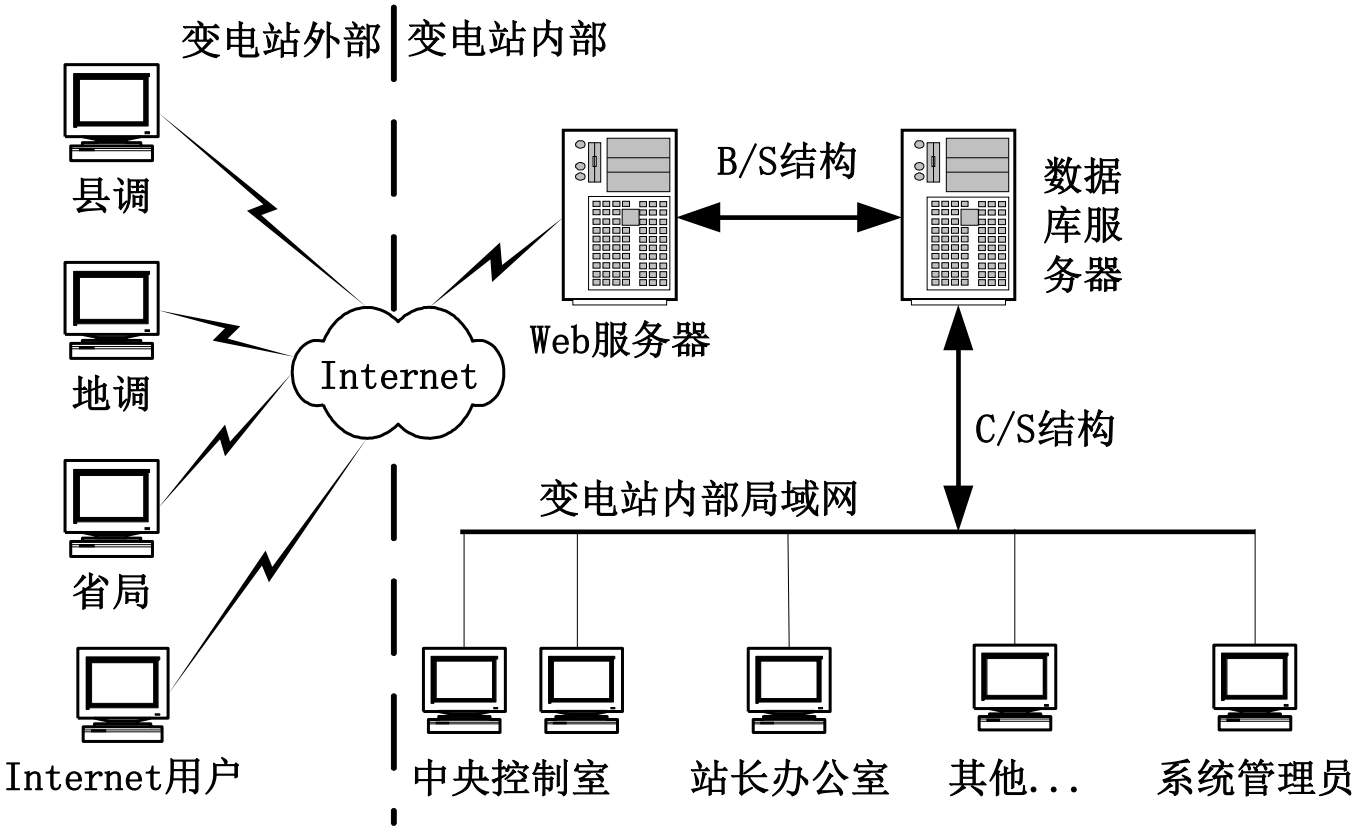
- 不同的结构有不同的处理能力的强项和弱点，一个系统的体系结构应该根据实际需要进行选择，以解决实际问题。
- 关于软件包、框架、通信以及其他一些体系结构上的问题，目前存在多种标准。
- 实际工作中，我们总会遇到一些遗留下来的代码，它们仍有效用，但是却与新系统有某种程度上的不协调。
- 即使在某一单位中，规定了共享共同的软件包或相互关系的一些标准，仍会存在解释或表示习惯上的不同。

2 异构结构的实例

◇ C/S与B/S混合--“内外有别”模型

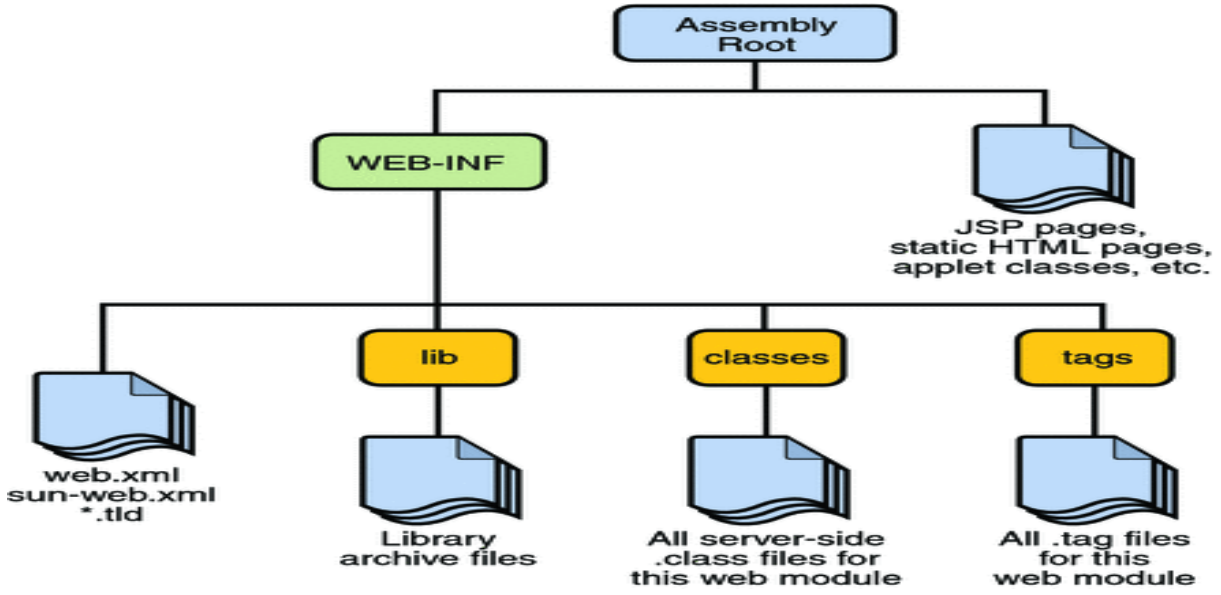


◇ 应用实例


变电站外部
变电站内部
县调
地调
省局
Internet用户
Internet
Web服务器
B/S结构
数据库服务器
C/S结构
变电站内部局域网
中央控制室
站长办公室
其他...
系统管理员

2.3 小结

- 软件体系结构风格（模式） = **经过实践证实的特定应用领域的软件体系结构**
  - 强调的是组织形式
  - 组织形式包含了一些构件和连接件



Web风格的软件共有的结构

- 使用软件体系结构风格的益处
  - 可以重用已经成功的软件体系结构，达到体系结构级别的软件重用
  - 有利于团队的交流。
- 软件体系结构风格的分类
  1. 管道-过滤器风格
  2. 面向对象风格
  3. 事件驱动风格
  4. 分层风格
  5. 客户/服务器风格
  6. B/S风格
  7. 异构风格的集成

不同风格的体系结构有不同的处理能力的强项和弱点，一个系统的体系结构风格应该根据实际需要进行选择，以解决实际问题。

thanks