软件体系结构风格

lliao@seu.edu.cn

目　录

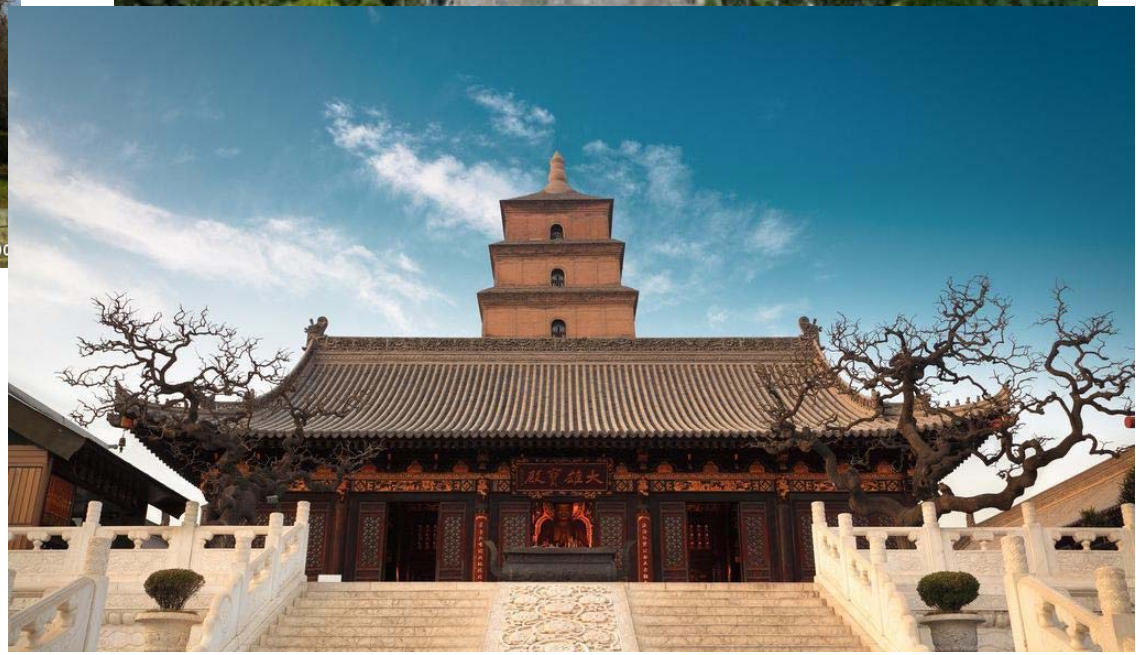
第3章 软件体系结构风格

- (一) 软件体系结构风格概述
- (二) 管道/过滤器风格
- (三) 数据抽象和面向对象风格
- (四) 事件驱动的风格
- (五) 层次系统风格
- (六) 仓库风格
- (七) C/S风格
- (八) 三层C/S风格
- (九) B/S风格
- (十) C/S与B/S混合软件体系结构
- (十一) 结束语

一、软件体系结构风格概述

- Christopher Alexander, The Timeless Way of Building, p247, 1979
  - 每个模式是一个由三部分组成的规则，表达了特定环境、问题和解(solution)之间的关系。
  - 作为现实世界的一个成分，每个模式表达了下列三者之间的一种关系：特定环境，在该环境中反复出现的力(forces)的系统，以及协调这些力的某种空间排列。
  - 作为语言的一个成分，每个模式是一条指令，展示了这种空间排列如何被一再重复使用，目的是协调同特定环境相关的力的系统。
  - 简单地说，模式既是存在于现实世界中的事物，又是告诉我们如何以及何时创造该事物的规则。模式既是过程，又是事物；既是活生生的事物的描述，又是创造该事物的过程的描述。


昵图网 www.nipic.com
By:610youyou No.20130506001846391000



一、软件体系结构风格概述

- 软件体系结构设计的一个核心问题是能否使用重复的体系结构模式，即能否达到体系结构级的软件重用。也就是说，能否在不同的软件系统中，使用同一体系结构。基于这个目的，学者们开始研究和实践软件体系结构的风格和类型问题。

一、软件体系结构风格概述...

- 软件体系结构风格是描述某一特定应用领域中系统组织方式的惯用范型(idiomatic paradigm)。
- 软件体系结构风格定义的主要内涵：
  - 定义了一个系统家族，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的。

一、软件体系结构风格概述…

- 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。

一、软件体系结构风格概述…

- （1）提供一个词汇表；
- （2）定义一套配置规则；
- （3）定义一套语义解释原则；
- （4）定义对基于这种风格的系统所进行的分析。

一、软件体系结构风格概述…

- Garlan和Shaw对通用体系结构风格的分类：
  （1）数据流风格：**批处理序列；管道/过滤器**
  （2）调用/返回风格：**主程序/子程序；面向对象风格；层次结构**
  （3）独立构件风格：**进程通讯；事件系统**
  （4）虚拟机风格：**解释器；基于规则的系统**
  （5）仓库风格：**数据库系统；超文本系统；黑板系统**

- 特别注意：体系结构风格不是对软件进行分类的标准。它仅仅是表示描述软件的不同角度而已。
  - 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。

二、管道-过滤器风格

- 概述
  - 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）。
  - 管道-过滤器风格的特性之一在于过滤器的相对独立性，即过滤器独立完成自身功能，相互之间无需进行状态交互。

二、管道-过滤器风格

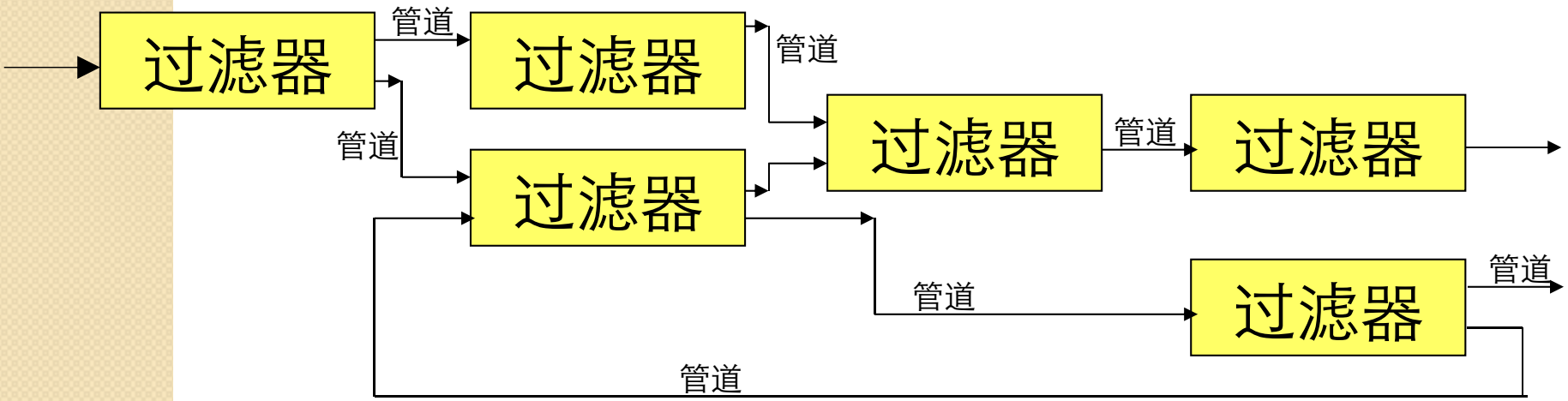


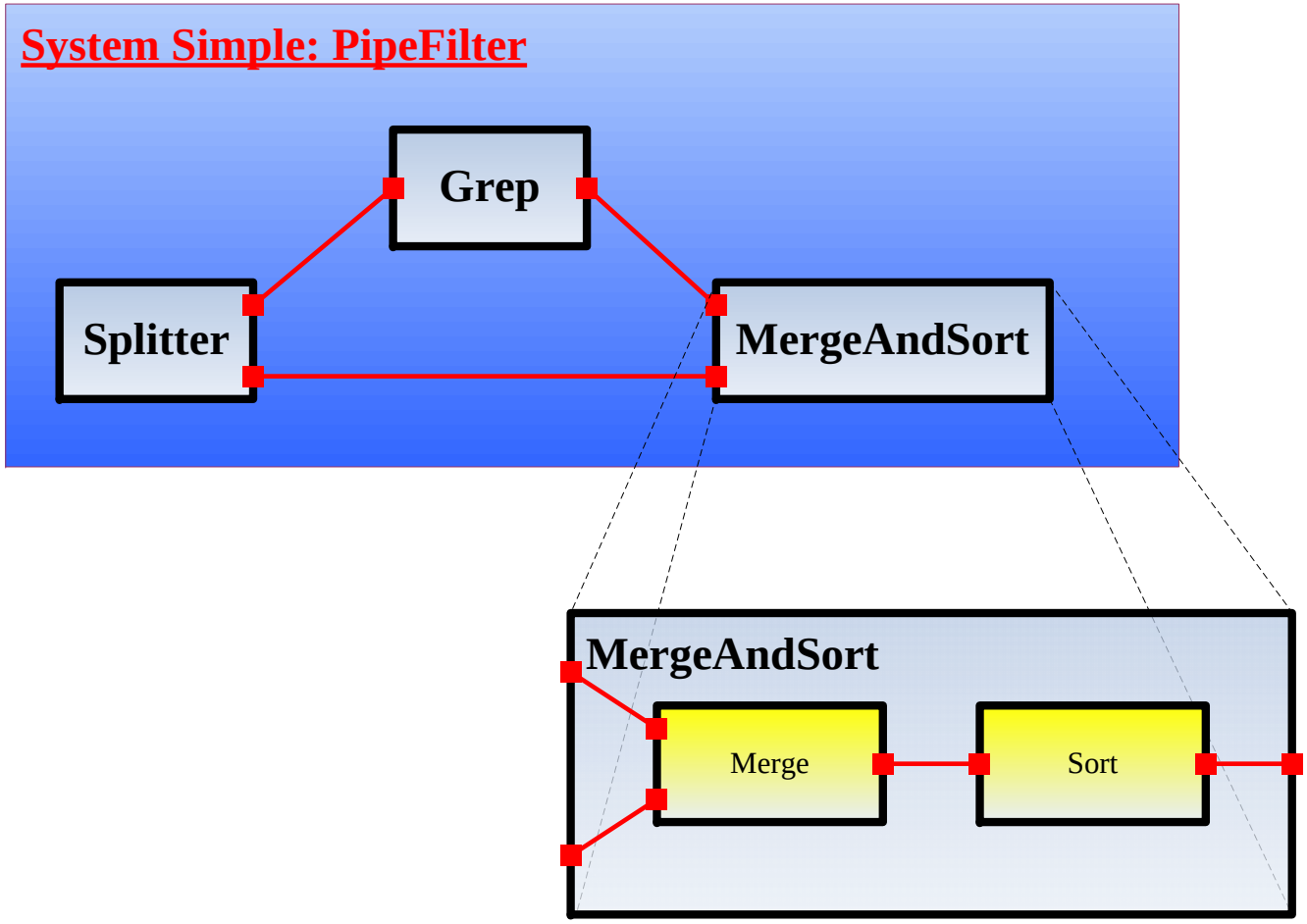
图 3-1 管道 – 过滤器风格的体系结构

二、管道-过滤器风格

- 过滤器是独立运行的构件
  - 非临近的过滤器之间不共享状态
  - 过滤器自身无状态
- 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
- 结果的正确性不依赖于各个过滤器运行的先后次序
  - 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。

二、管道-过滤器风格

- 一个采用了嵌套的管道过滤器的系统示例：

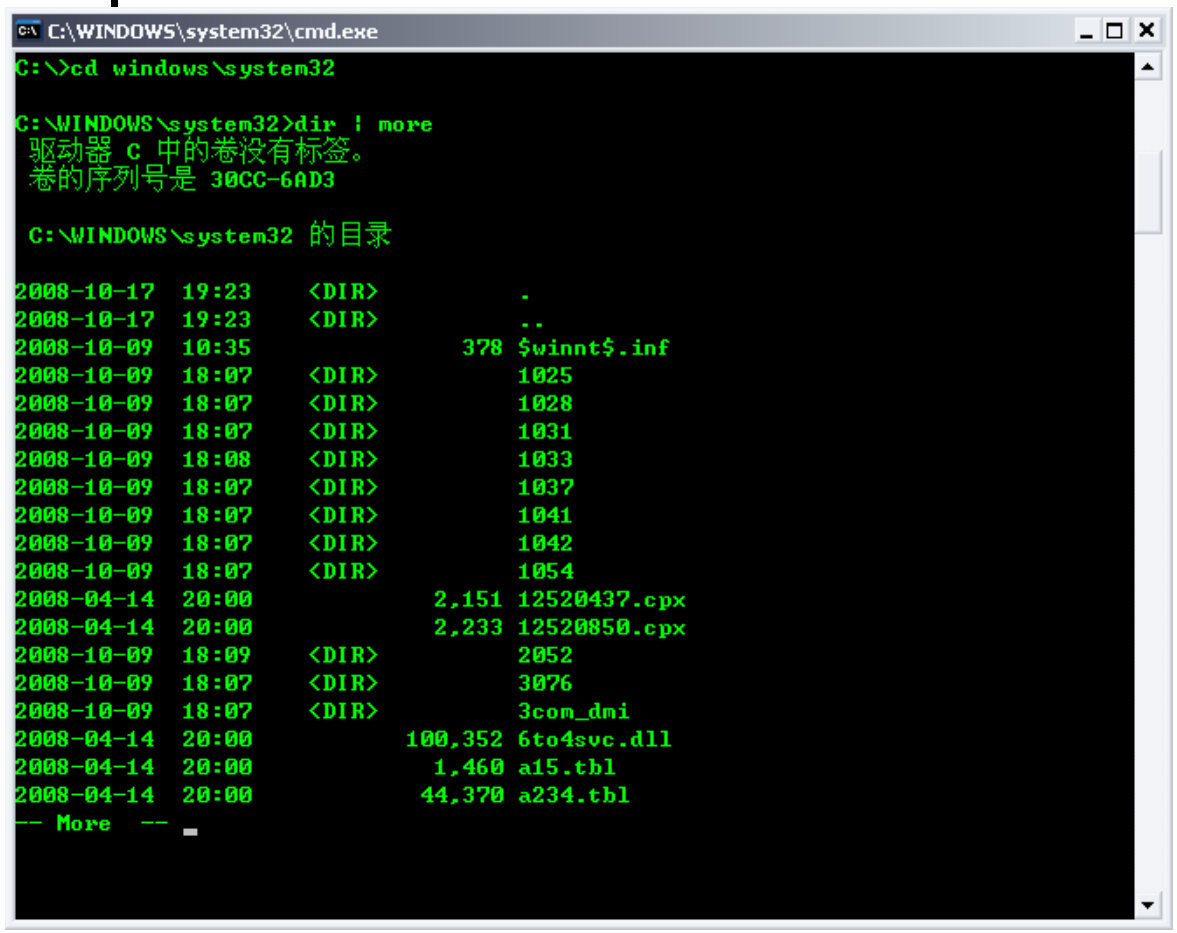


管道-过滤器风格实例

- 在Unix和Dos中，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：Unix shell中:
  - cat file|grep xyz|sort|uniq>out
  - 即找到含xyz的行，排序、去掉相同的行，最后输出
- 例如：DOS 中：
  - 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。

管道-过滤器风格 实例

- dir | more



管道-过滤器风格实例

- 实例：
  - 传统的编译器，一个阶段的输出是另一个阶段的输入（包括词法分析、语法分析、语义分析和代码生成）

管道-过滤器风格优点

- 设计者可以将整个系统的输入、输出特性简单地理解为各个过滤器功能的合成。
  - 设计人员将整个系统的输入输出行为理解为单个过滤器行为的叠加与组合。这样可以将问题分解，化繁为简。将系统抽象成一个“黑箱”，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。

管道-过滤器风格优点

- 管道-过滤器风格支持功能模块的复用
  - 任何两个过滤器，只要它们之间传送的数据遵守共同的规约，就可以相连接。每个过滤器都有自己独立的输入输出接口，如果过滤器间传输的数据遵守其规约，只要用管道将它们连接就可以正常工作。

管道-过滤器风格优点

- 基于管道-过滤器风格的系统具有较强的可维护性和可扩展性。
  - 旧的过滤器可以被替代，新的过滤器可以添加到已有的系统上。软件的易于维护和升级是衡量软件系统质量的重要指标之一，在管道-过滤器模型中，只要遵守输入输出数据规约，任何一个过滤器都可以被另一个新的过滤器代替，同时为增强程序功能，可以添加新的过滤器。这样，系统的可维护性和可升级性得到了保证。

管道-过滤器风格优点

- 支持一些特定的分析，如吞吐量计算和死锁检测等。
  - 利用管道-过滤器风格的视图，可以很容易的得到系统的资源使用和请求的状态图。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。

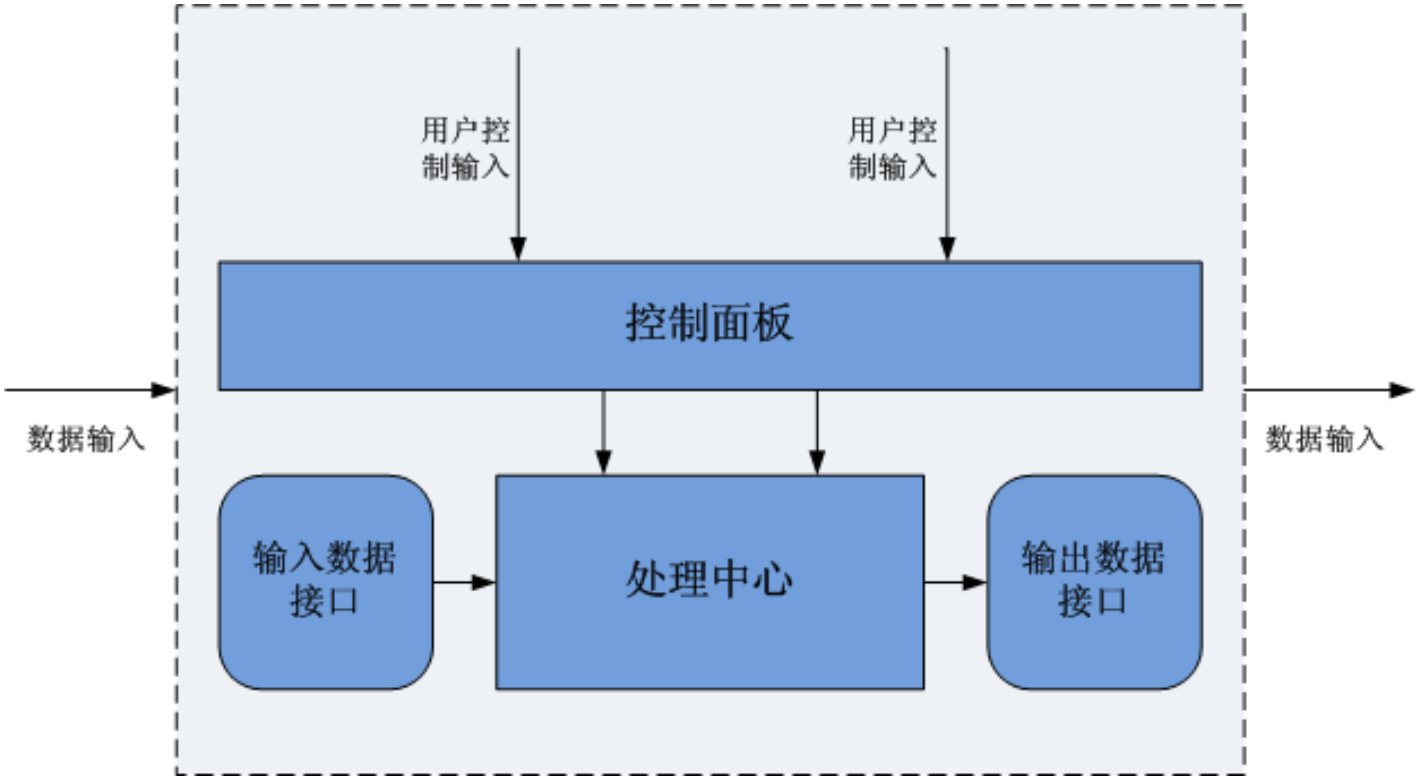
管道-过滤器风格优点

- 管道-过滤器风格具有并发性
  - 每个过滤器作为一个单独的执行任务，可以与其它过滤器并发执行。过滤器的执行是独立的，不依赖于其它过滤器的。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，从而大大提高系统的整体效率，加快处理速度。

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。由以上的缺点，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。

管道-过滤器风格不足



改进的过滤器

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。

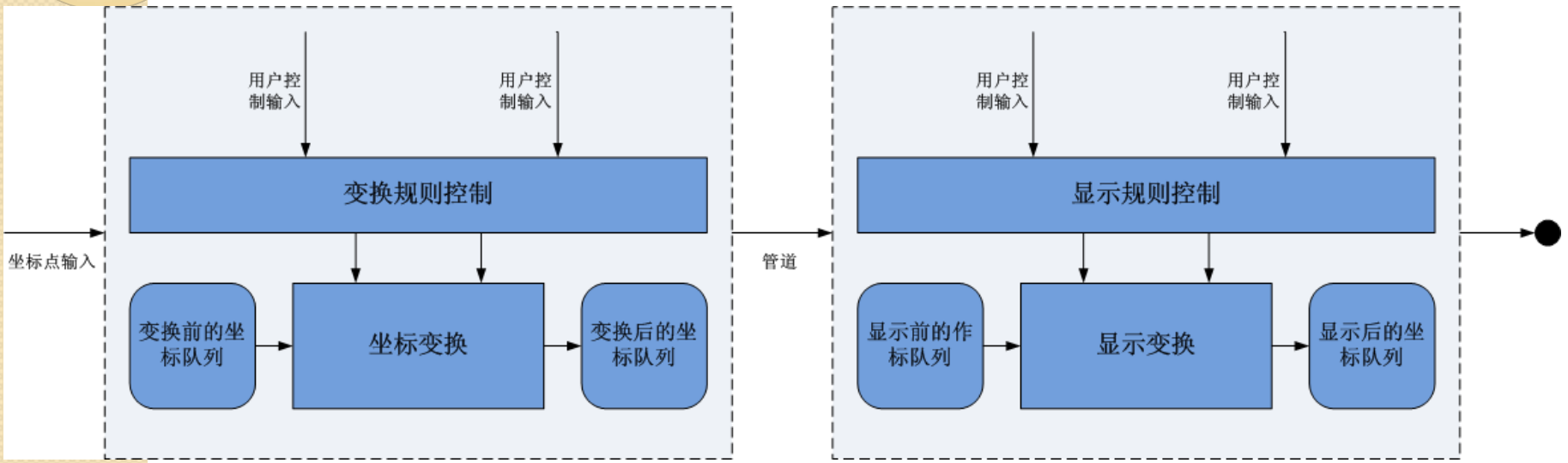
管道-过滤器风格练习

- 需求描述：假设有一批实时的二维坐标点数据需要变换（即对点的横、纵坐标进行缩放），并在屏幕上进行显示，要求外部要能设置变换规则（如缩放倍数）和显示规则（如显示模式和显示颜色）。
- 请根据此需求，进行管道过滤器风格的体系结构建模。

管道-过滤器风格练习

- 体系结构建模
  - 这是一个对坐标点的数据流进行顺序处理的过程，可以应用管道-过滤器体系结构建模。将这个系统分为两个过滤器，一个为坐标点数据流变换过滤器，另一个为坐标点数据流实时显示过滤器。其中，坐标点数据流变换过滤器有一个外部控制接口对变换规则如缩放倍数进行设置，坐标点数据流实时显示过滤器有一个外部控制接口对显示规则如显示模式和显示颜色进行设置。整个系统的体系结构如图所示。

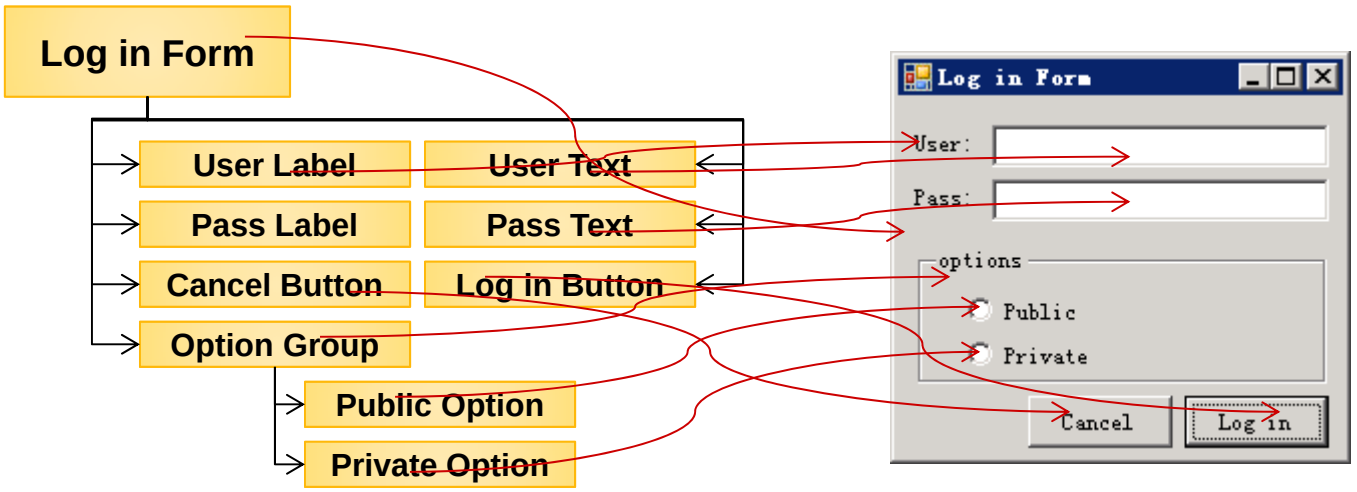
管道-过滤器风格练习



系统体系结构图

三、面向对象风格

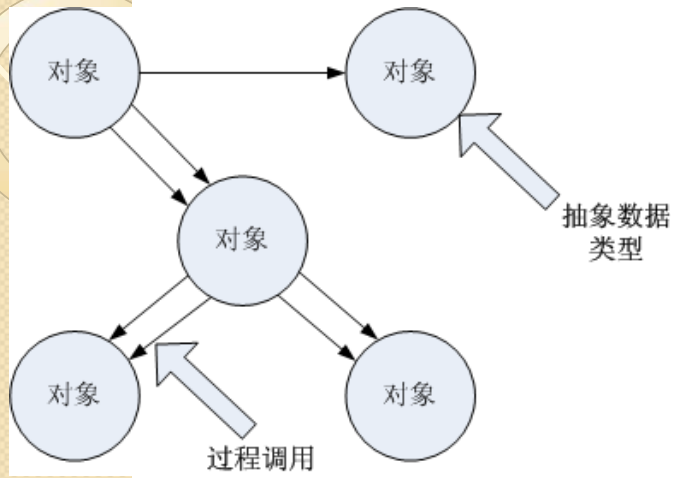
- 概述
  - 面相对象模式集数据抽象、抽象数据类型、类继承为一体，使软件工程公认的模块化、信息隐藏、抽象、重用性等原则在面向对象风格下得以充分实现。
- 应用场合
  - 面向对象的体系结构模式适用于数据和功能分离的系统中，同样也适合于问题域模型比较明显，或需要人机交互界面的系统。大多数应用事件驱动风格的系统也常常应用了面向对象风格


Log in Form
User Label
User Text
Pass Label
Pass Text
Cancel Button
Log in Button
Option Group
Public Option
Private Option
Log in Form
User:
Pass:
options
Public
Private
Cancel
Log in

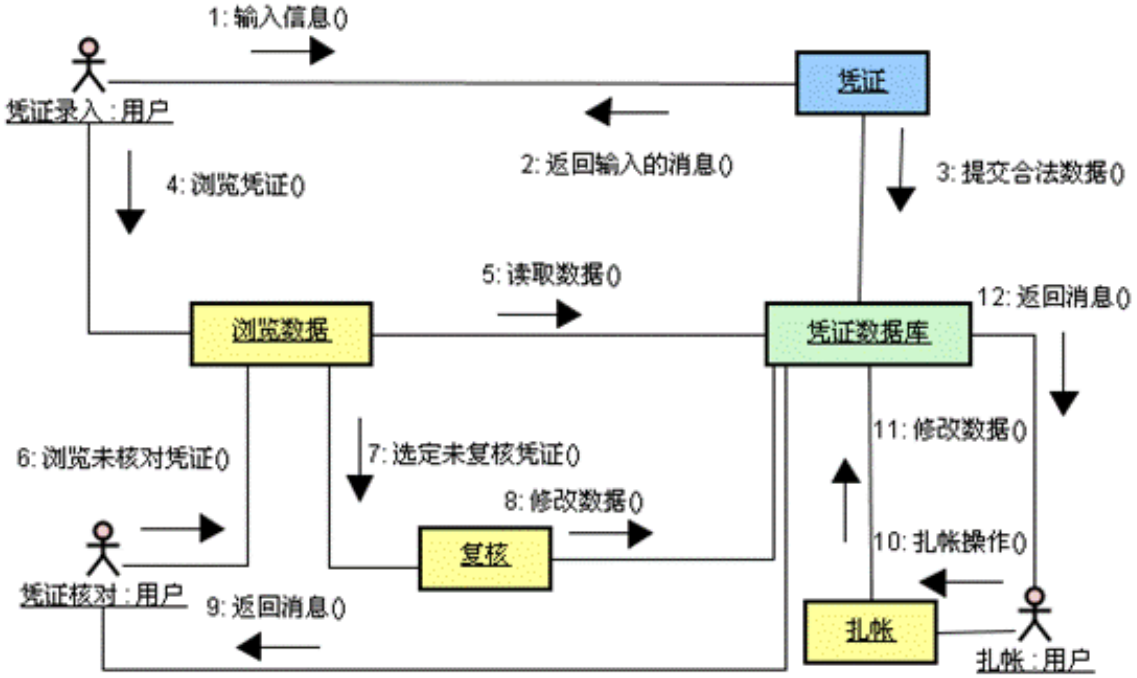
面向对象风格设计原则

- 面向对象风格系统设计时有下述几条基本原则
  - 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应用关系。
  - 对象利用应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。

面向对象风格



面向对象风格的抽象描述



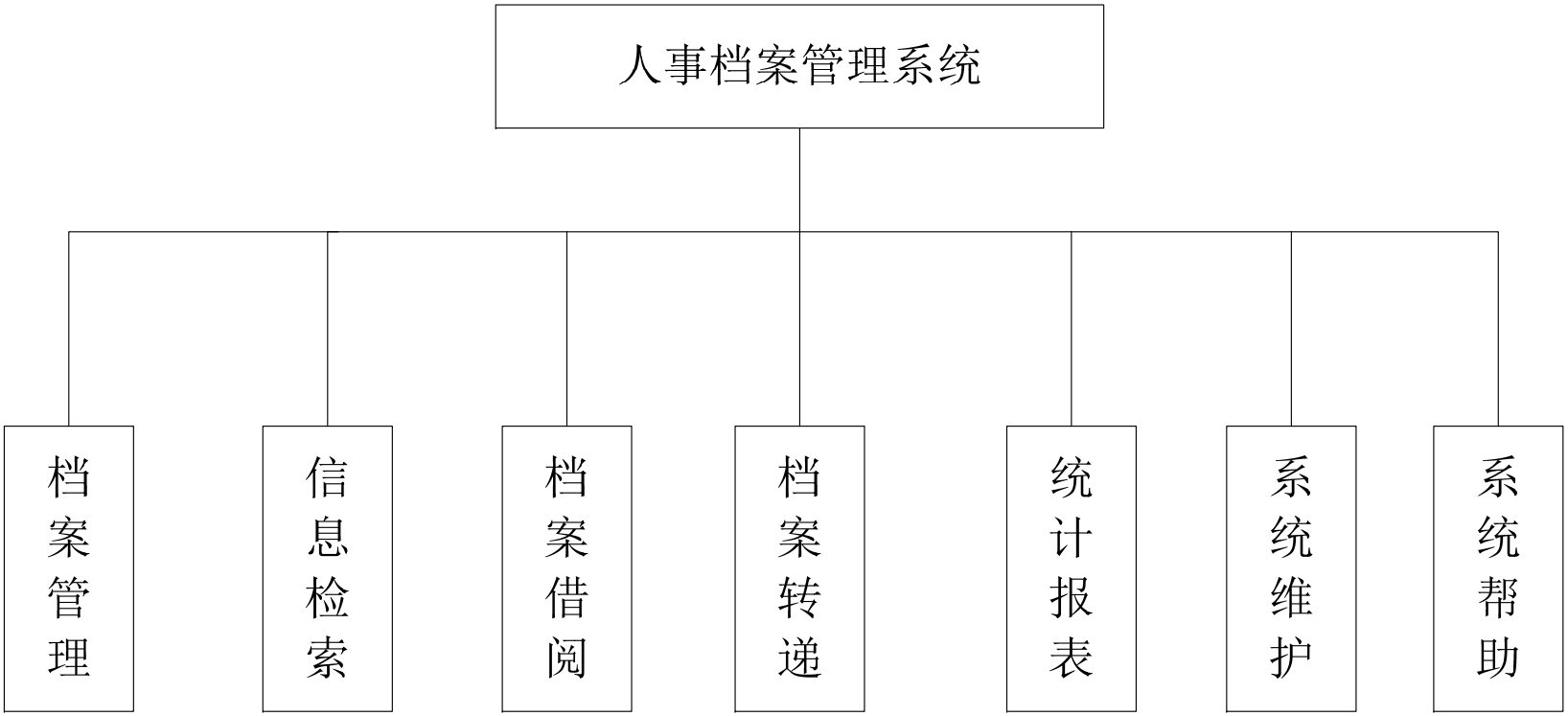
一个银行系统对象之间的协作实例

数据的表示及其相应操作封装在一个抽象数据类型或对象中。这种风格的组件是对象，或者说是抽象数据类型的实例。对象是一种被称作管理者的组件，因为它负责保持资源的完整性。对象是通过函数和过程的调用来交互的。

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
- 代码共享
  - 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

面向对象风格实例——人事档案管理系统



系统功能结构

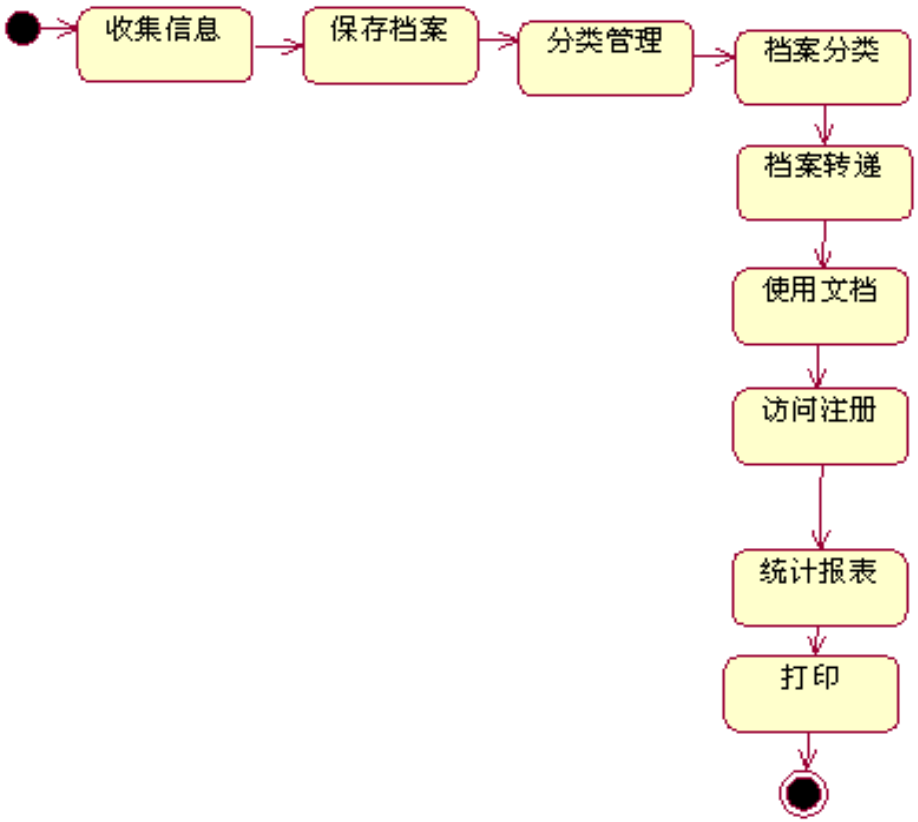
面向对象风格实例——人事档案管理系统

- 系统功能介绍：
  - 档案管理根据高校人事档案管理的特点，本模块可通过录入各类人事档案信息，来构造档案数据库，编制各种目录索检。针对档案材料录入工作量较大，在该功能模块中设置了多种方式快速录入法，如对指定的部分内容可采用代码录入和菜单选项等输入方法.
  - 信息检索该模块主要是检索有关的人事档案信息，其检索方式为姓氏笔画检索目录。在具体检索中又可分为精确查询和模糊查询，并可将检索内容动态输出，满足档案查询的需要。
  - 档案借阅该模块主要是对档案的借阅情况、归还情况、利用登记等方面进行管理。它能为研究如何更有效地利用人事档案资料提供必要的信息。

面向对象风格实例——人事档案管理系统

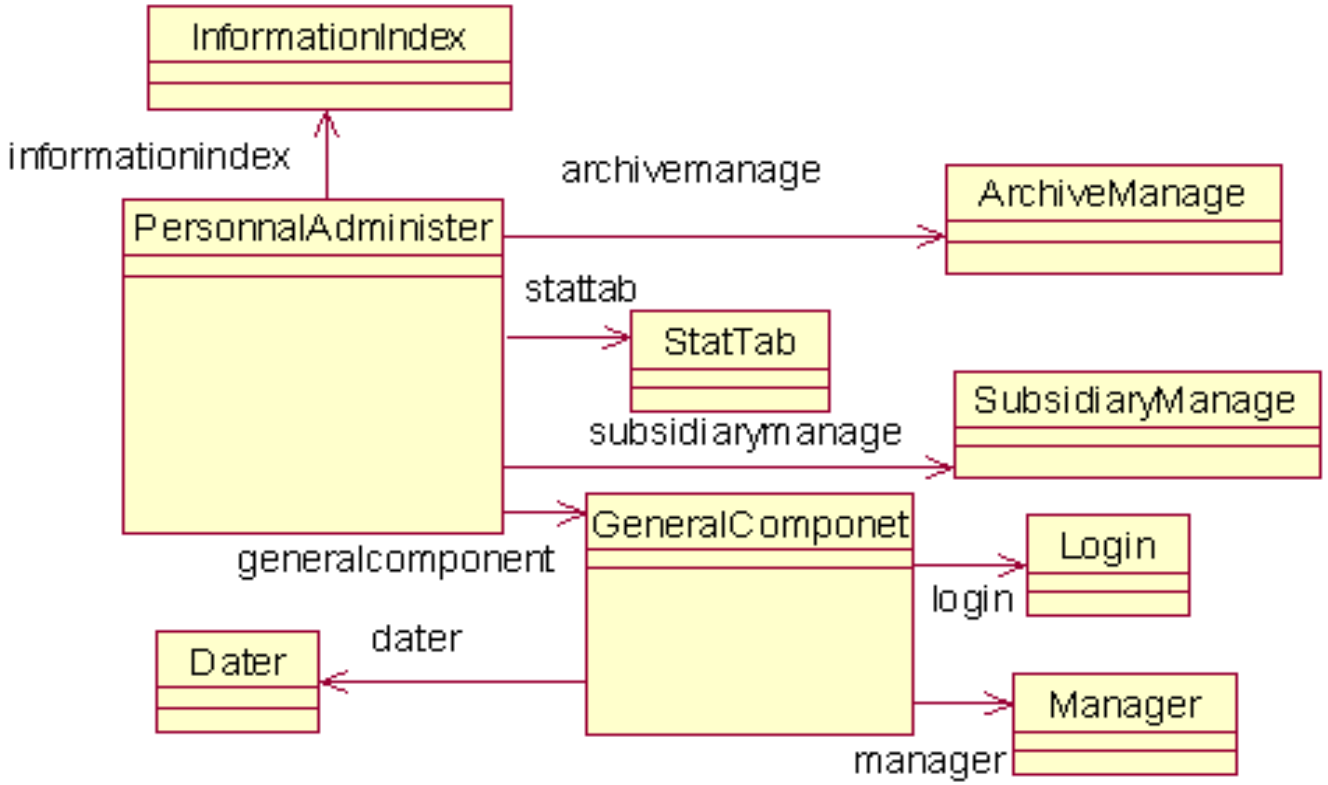
- 档案转递该模块包括人事档案的转进和转出管理，编制清单，并能在档案转递后，对已变更档案数据库进行相应地调整，以完成相应档案的删加。
- 统计报表该模块主要用于统计库存的各类人事档案的实际数量，及每年归档的各类档案数量，并可完成相应的图形绘制和报表打印。其中，在报表生成中，该模块可根据管理人员对报表的自定义设置来生成相应的非范式报表。
- 系统维护由于高校人事档案的数据管理是一项非常重要的工作，尤其是它的安全可靠性。因此，在进入本模块操作之前，系统会提醒用户输入姓名、操作口令和权限级别。同时该功能模块还包括操作员管理、口令修改、重新登录、权限级别设置、系统日志及系统初始化六个子模块。
- 系统帮助本模块提供了在线联机帮助，可实现帮助主题的查询，还提供了计算器、日记/日历等系统工具和关于本系统的简介。

面向对象风格实例——人事档案管理系统



系统活动图

面向对象风格实例——人事档案管理系统



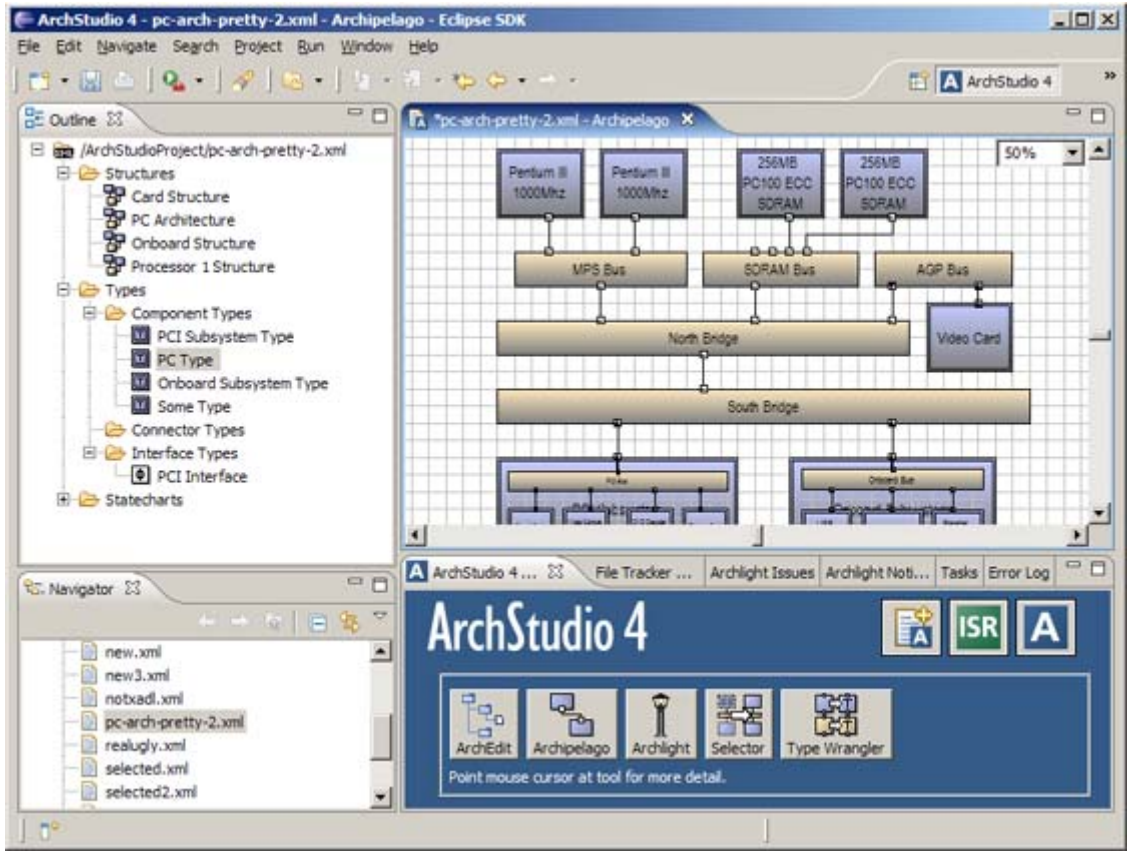
系统类结构图

面向对象风格实例

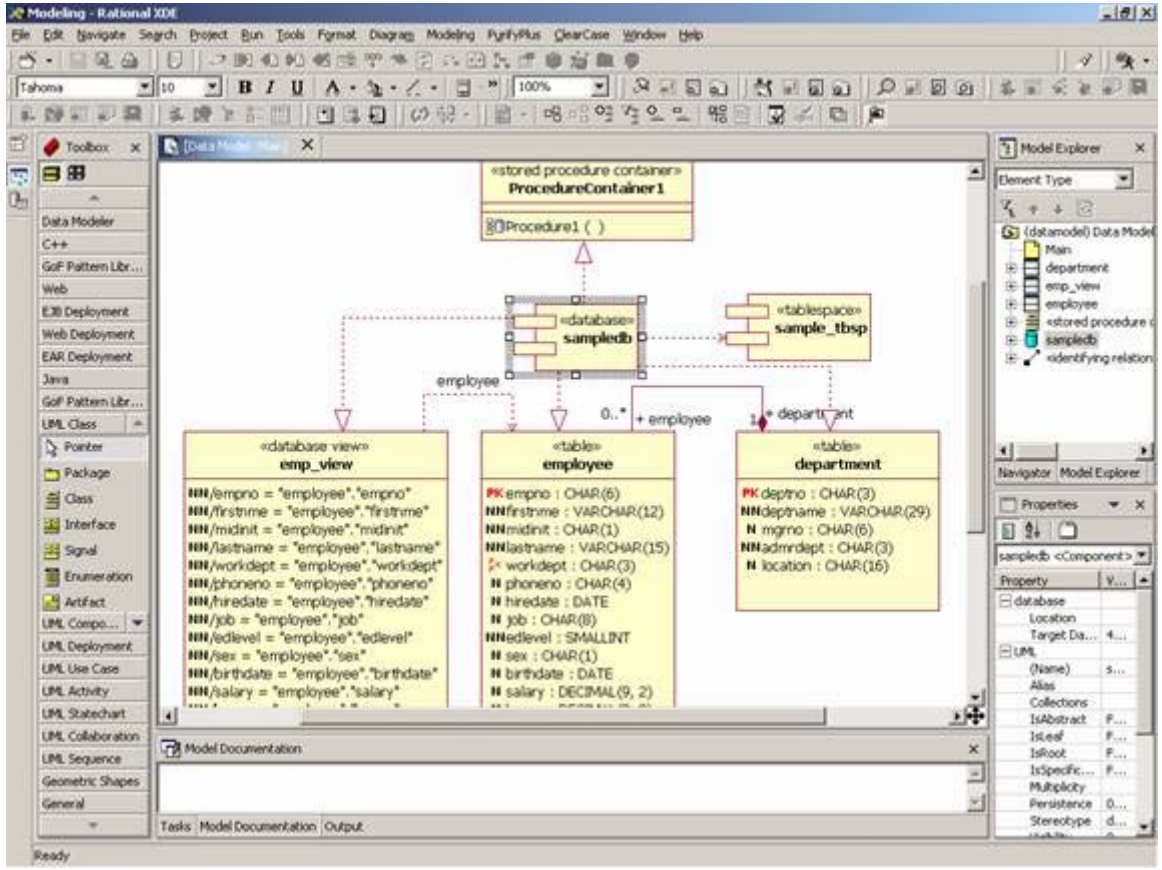
- 构件的描述方法:利用GUI体系结构框架自动生成工具，可以完成下述几点功能:
  - 生成构件模型，包括构件的属性、接口和实现;
  - 建立连接器模型，包括协议、属性和实现;
  - 体系结构的抽象和封装;
  - 类型和类型检查;
  - 主动规范，提供设计向导;
  - 多视图模式，对不同层次的用户显示不同的内容;
  - 生成实现，如将构件对应为面向对象技术中的类;
  - 将系统的修改动态映射到实现。

面向对象风格实例

GUI体系结构框架自动生成工具，*ArchStudio*

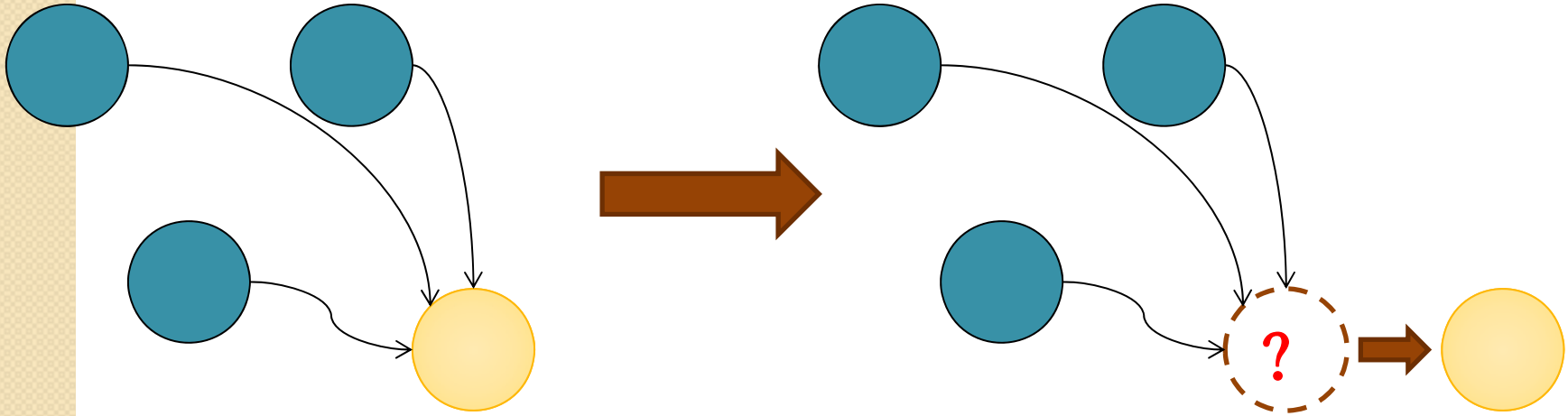


GUI体系结构框架自动生成工具，IBM Rational XDE



面向对象风格不足

面向对象风格最大的不足在于如果一个对象需要调用另一个对象，它就必须知道那个对象的标识（对象名或对象引用），这样就无形之中增强了对象之间的依赖关系。如果一个对象改变了自己的标识，就必须通知系统中所有和它有调用关系的对象，否则系统就无法正常运行。



四、事件驱动风格

- **基于事件的系统风格的思想是组件不直接调用一个过程，而是触发或广播一个或多个事件。**

- 系统中的其它组件中的过程在一个或多个事件中注册，当一个事件被触发，系统自动调用在这个事件中注册的所有过程，这样，一个事件的触发就导致了另一模块中的过程的调用。

- 从体系结构上说，这种风格的组件是一些模块，这些模块既可以是一些过程，又可以是一些事件的集合。过程可以用通用的方式调用，也可以在系统事件中注册一些过程，当发生这些事件时，过程被调用。

事件驱动风格特征

- 事件驱动系统具有以下一些特点：
  - 系统是由若干子系统或元素所组成的一个整体；
  - 系统有一定的目标，各子系统在某一种消息机制的控制下，为了这个目标而协调行动；
  - 在一个系统的若干子系统中，必定有一个子系统起着主导作用，而其他子系统则处于从属地位；

事件驱动风格特征

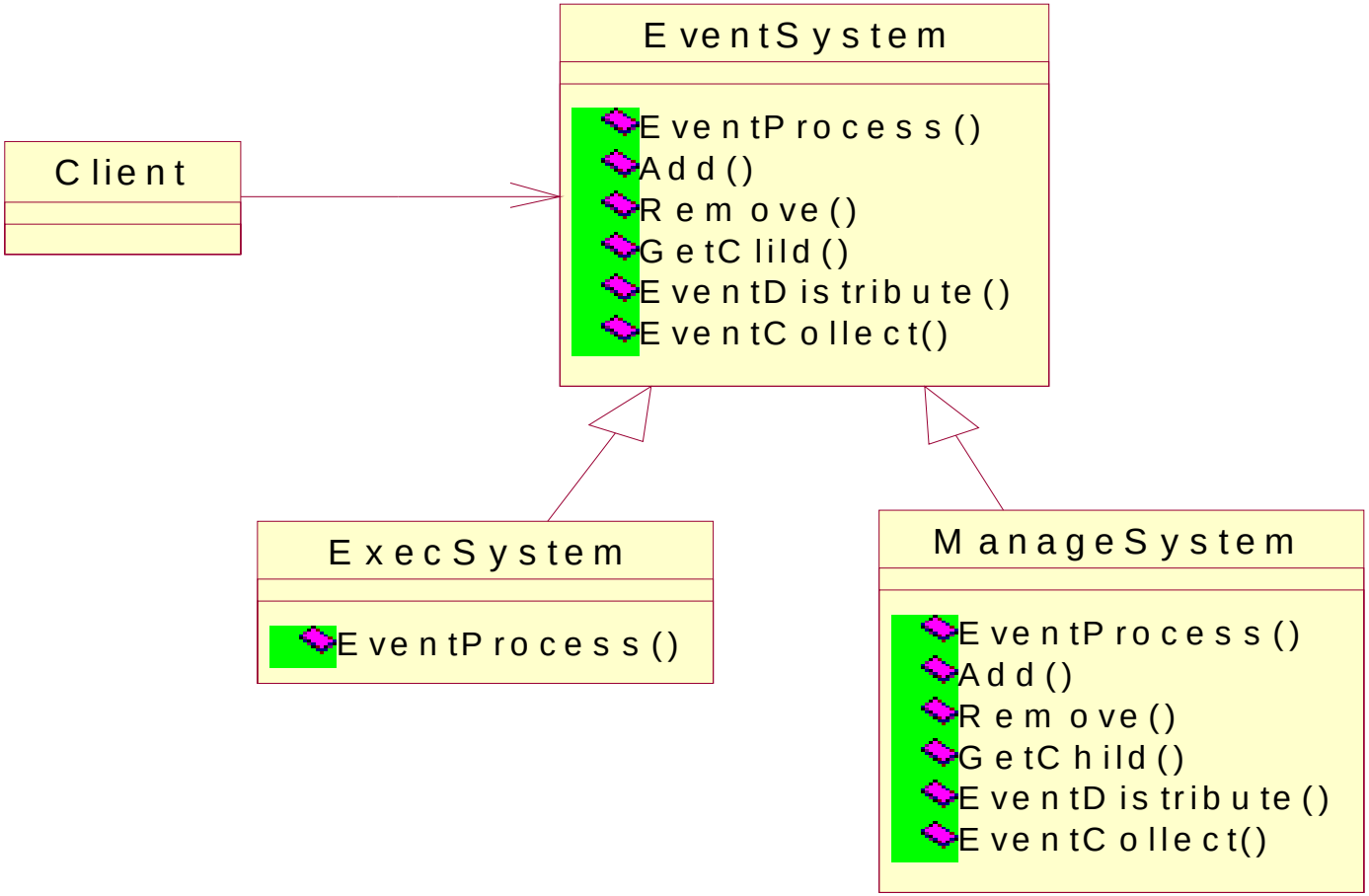
- 事件驱动系统具有以下一些特点（续）：
  - 任一系统和系统内的任一元素，都有1个事件收集机制和1个事件处理机制，通过这种机制与周围环境发生作用和联系；
  - 事件的触发者并不知道哪些组件会被这些事件影响。这样不能假定组件的处理顺序，甚至不知道哪些过程会被调用。

事件驱动风格基本结构

事件驱动系统具有某种意义上的递归性，形成了“部分－整体”的层次结构，可以用属性结构加以表示。一个简单的表示方法是为执行系统定义一些类，另外定义一些类作为这些执行系统的容器类，也就是管理系统。

事件驱动风格

- 事件驱动风格的基本结构，如下图:



事件驱动风格设计原则

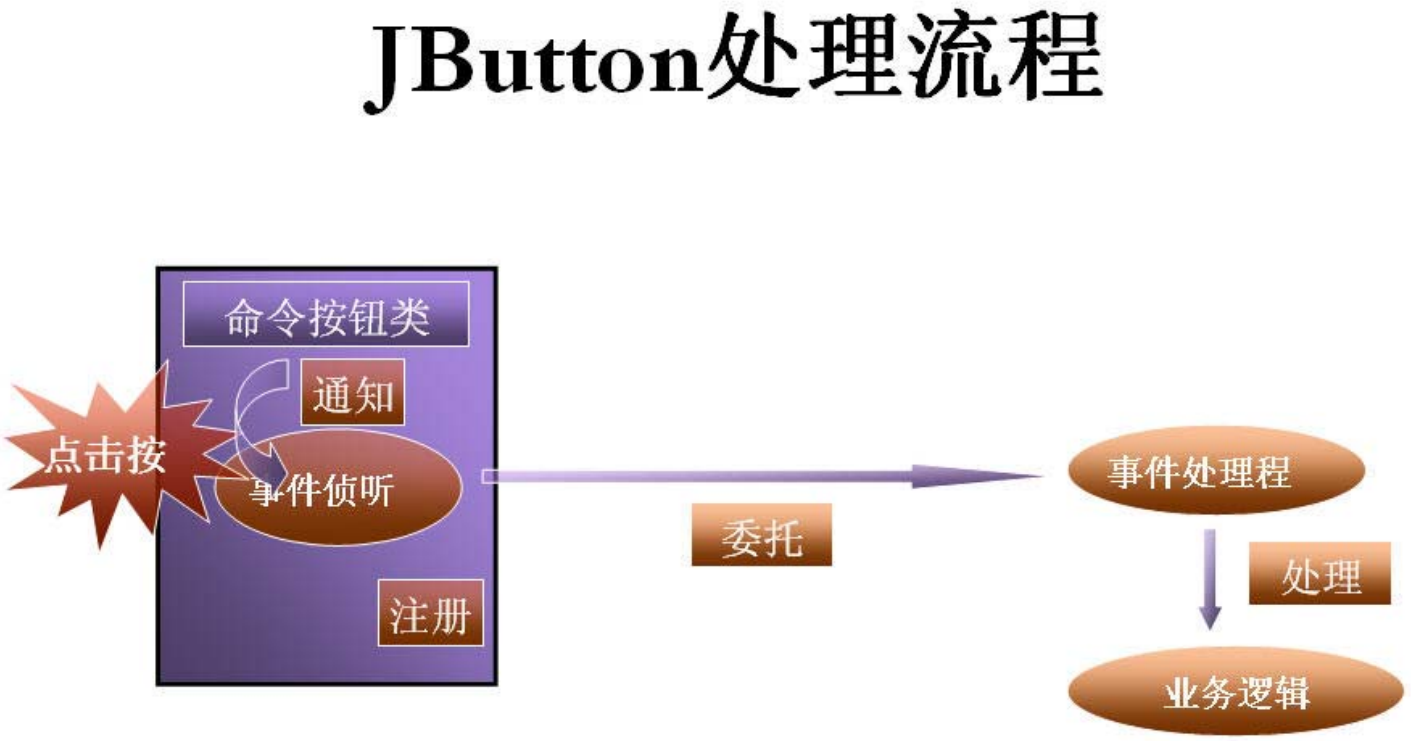
- 事件驱动风格系统设计时有下述几条基本原则
  - 从系统论的角度来看待描述的对象，合理分解子系统，保证各个子系统的独立性和社会性；
  - 无论事件系统多么复杂，子系统性质的差异多么大，任何子系统都可以分为2类：管理系统和执行系统。
  - 为了达到系统的目标，系统内的各个子系统通过传递消息和执行消息来协同操作。
  -

事件驱动风格设计原则

- 事件驱动风格系统设计时有下述几条基本原则(续)
  - 在一个完整系统中，必须有这样一个子系统，它没有上级，必须收集系统外的事件及下级发出的事件。
  - 管理类型的子系统一般不执行具体操作，它的主要功能是按照自己的职能指挥下级完成任务，功能性操作一般由执行类型的子系统完成。
  - 在一般情况下，除最高级管理子系统外，子系统一般是“有问才答”，即使在必要的情况下需要积极寻找事件时，也必须征得上级系统得许可，保证了系统的控制流不会分散。

事件驱动风格实例

- Java中的button实现



事件驱动风格实例

```
private void  initialize() {  //窗口初始化代码
…
//btnPress就是这次点击操作中的事件源
Buttton  btnPress = new JButton();
//向事件源btnPress植入侦听器对象ButtonEventHandler
btnPress.addActionListener (new ButtonEventHandler(this));
 …
}
class ButtonEventHandler implements ActionListener {
        //窗体对象
        private EventDemo form = null;
        //通过构造体传入窗体对象，
        //作用在于让侦听器对象明白事件源处于
        //哪个窗体容器中
        public ButtonEventHandler(EventDemo form) {
```

事件驱动风格 实例

```
        this.form = form;
    }
    //委托方法
    public void actionPerformed(ActionEvent e) {
    //该方法将会把事件的处理权交给窗体容器类的btnPress_Click方法处理。
     this.form.btnPress_Click(e);
    }
}

//真正的事件处理代码片断:
private void btnPress_Click(ActionEvent e) {
  String message = "你点击的按钮名叫:"
      + ((JButton) e.getSource()).getName();
    this.txtMessage.setText(message);
}
```

事件驱动风格优点

- 事件驱动风格非常适合于描述系统族，在属于同一族的任何系统中，系统的高级管理子系统的描述是完全类似的，便于重用；
- 由于最高管理子系统牢牢的掌握着控制权，又因为各同级子系统一般不直接发生关系，因此容易实现并发处理和多任务操作；
- 基于事件驱动风格的系统具有良好的可扩展性，设计者只需为某个对象注册一个事件处理接口就可以将该对象引入整个系统，同时并不影响其它的系统对象。

事件驱动风格不足

- 组件放弃了对系统计算的控制。一个组件触发一个事件时，不能确定其它组件是否会响应它。而且即使它知道事件注册了哪些组件的构成，它也不能保证这些过程被调用的顺序。
- 数据交换的问题。有时数据可被一个事件传递，但另一些情况下，基于事件的系统必须依靠一个共享的仓库进行交互。在这些情况下，全局性能和资源管理便成了问题。
- 既然过程的语义必须依赖于被触发事件的上下文约束，关于正确性的推理存在问题。

事件驱动风格和面向对象风格的关系

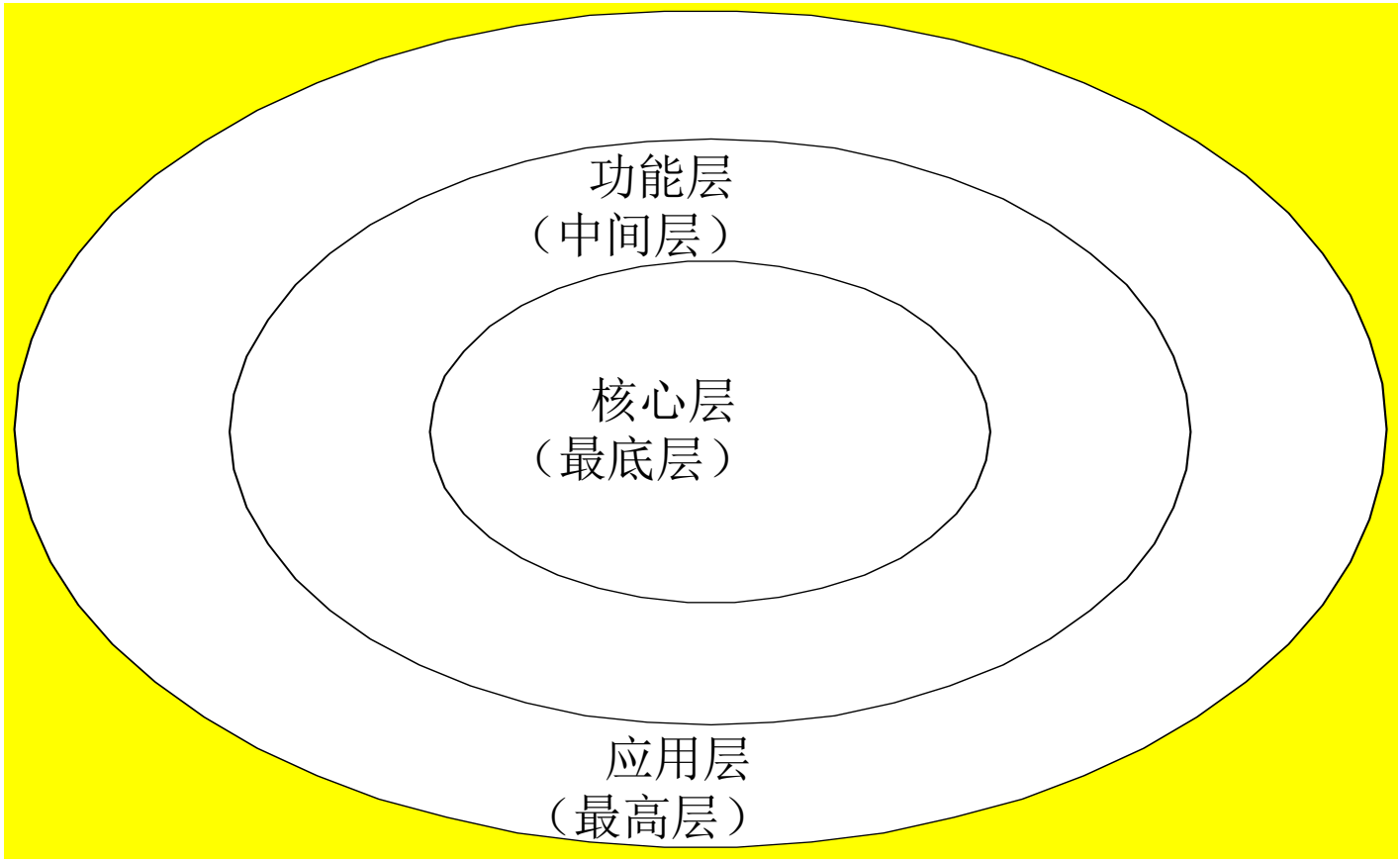
- 基于面向对象风格的系统由多个封装起来的对象构成，对象之间通过消息传递实现通信，而事件驱动正是对消息传递机制的一种实现。所以基于事件驱动风格的系统往往都是面向对象的。

五、层次系统风格

- 一个分层系统采用层次化的组织方式构建，系统中的每一层都要承担两个角色。
  - 首先，它要为结构中的上层提供服务；
  - 其次，它要作为结构中下面层次的客户，调用下层提供的功能函数。

分层风格特征

- 一个概念上的分层模型如下图所示：

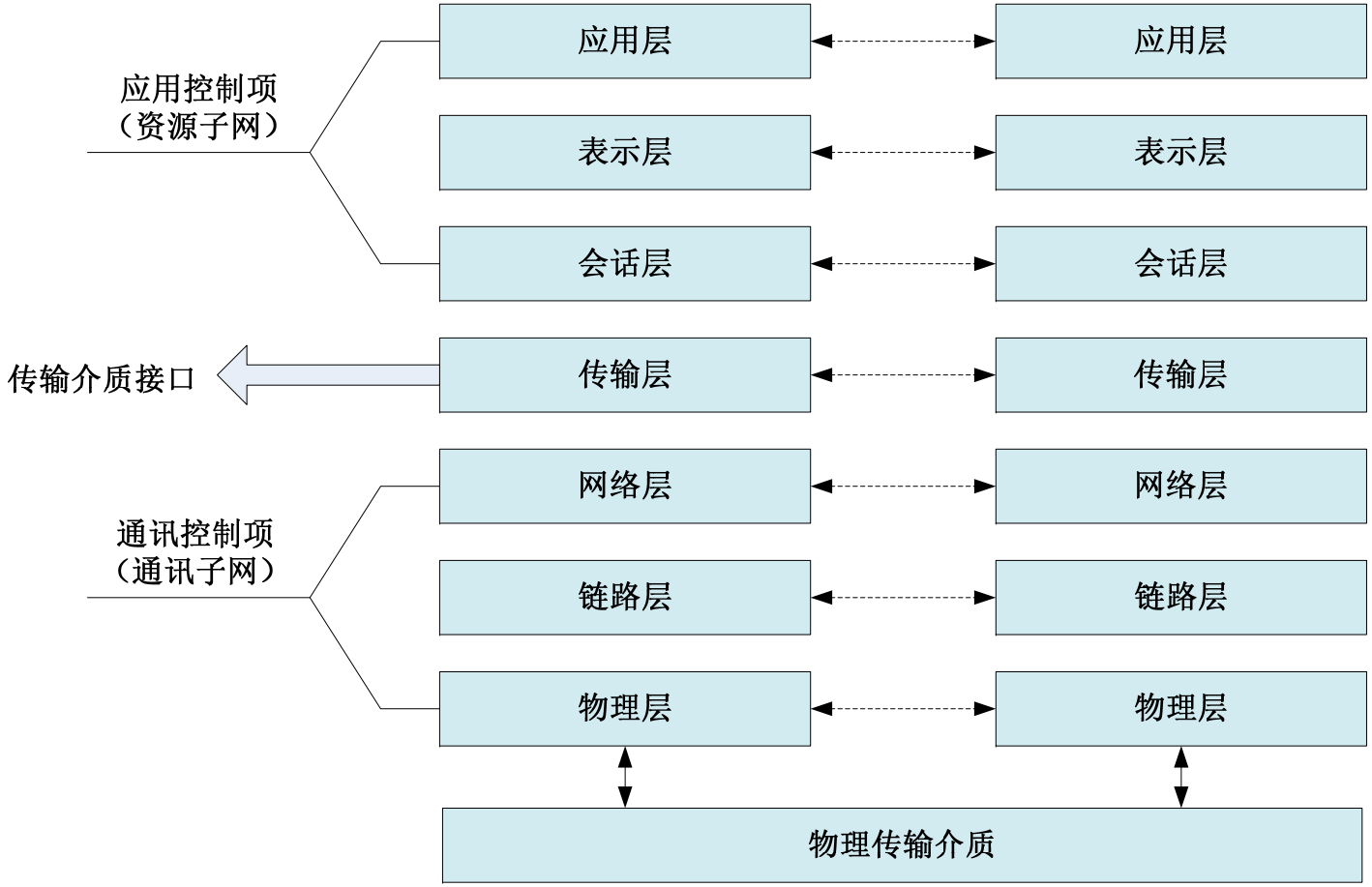


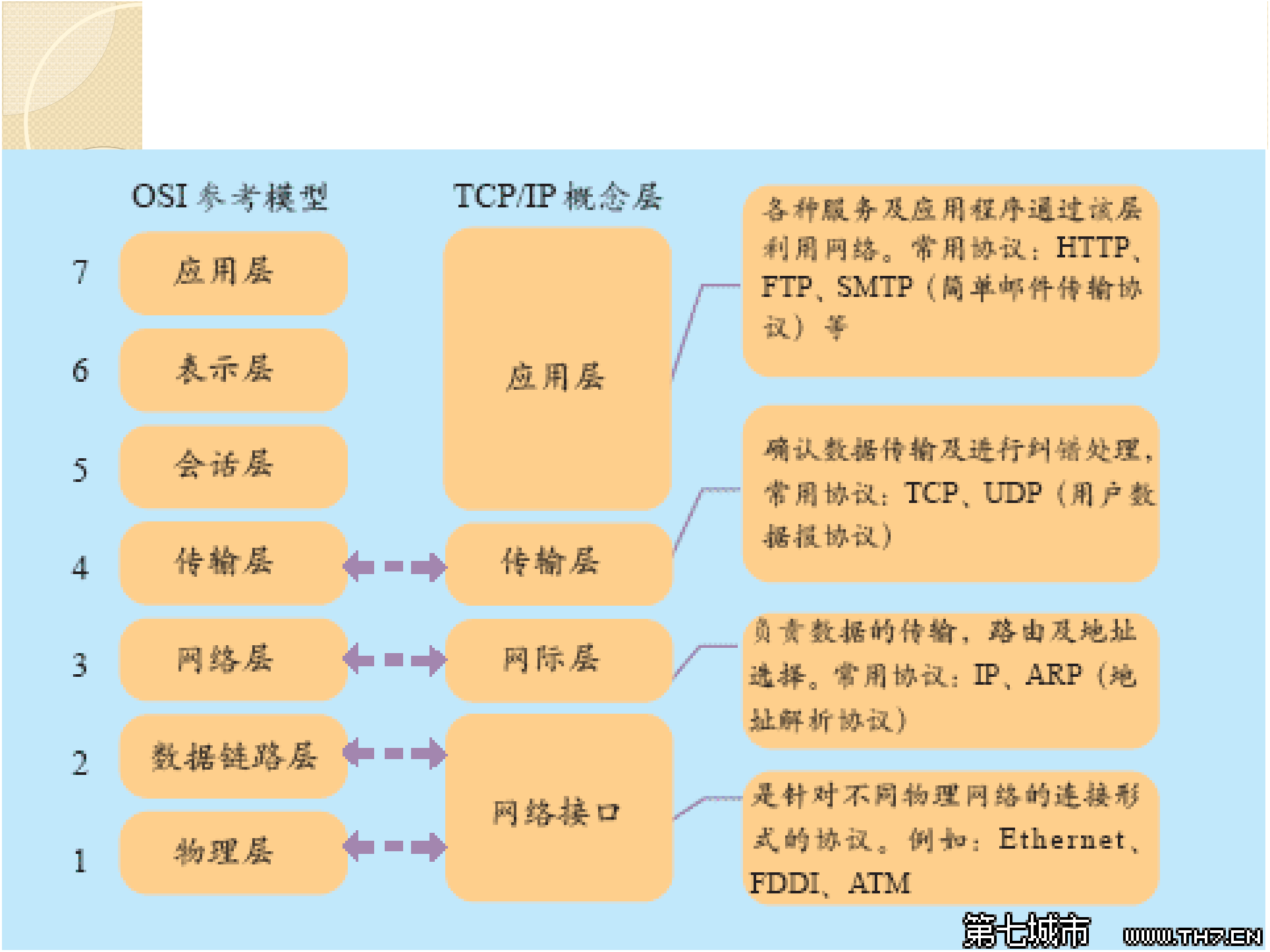
分层风格实例

- 分层风格实例：计算机网络的设计
- 概述
  - 网络协议设计者将计算机网络中的各个部分按其功能划分为若干个层次（Layer），其中的每一个层次都可以看成是一个相对独立的黑箱、一个封闭的系统。用户只关心每一层的外部特性，只需要定义每一层的输入、数据处理和输出等外部特性。

分层风格实例

- ISO/OSI网络体系结构





分层风格优点

- 分层风格具有一些系统设计者无法抗拒的优势：
  - 分层风格支持系统设计过程中的逐级抽象，允许将一个复杂问题分解成一个增量步骤序列的实现
  - 基于分层风格的系统具有较好的可扩展性
  - 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，甚至即使一个系统的逻辑结构是层次化的，出于对系统性能的考虑，系统设计师不得不把一些低级或高级的功能综合起来；
- 对于抽象出来的功能具体应该放在哪个层次上也是设计者头疼的一个问题,很难找到一个合适的、正确的层次抽象方法。

六、仓库风格

- 在仓库风格中，有两种不同的组件：
  - **中央数据结构**说明当前状态
  - **独立组件**在中央数据存贮上执行

- 信息交互方式的差异导致了控制策略的不同。主要的控制策略有两种，正是依据这两种不同的控制策略，基于数据共享风格的系统被分成两个子类：

六、仓库风格

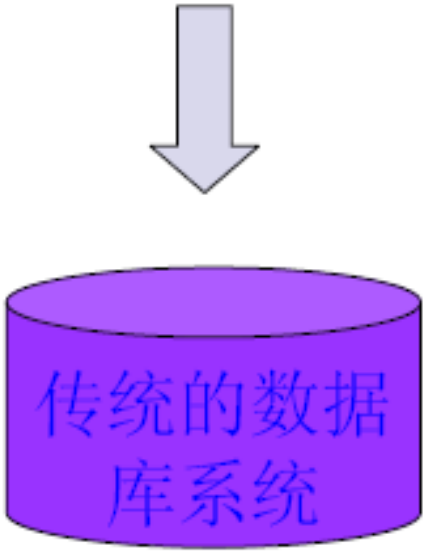
软件部件

- 表示当前状态的中心数据结构
- 一组相互独立的处理中心数据的部件

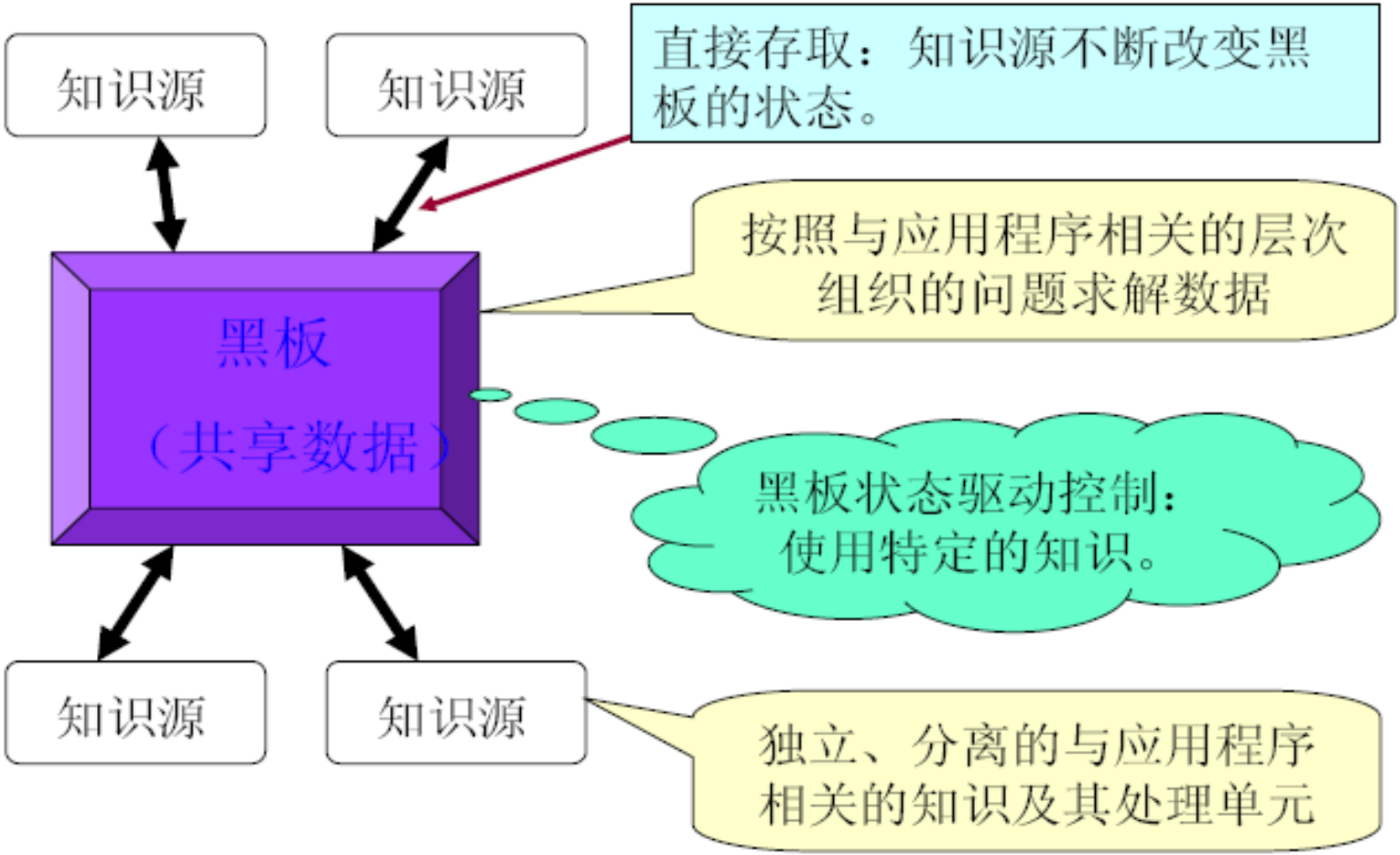
部件交互方式

根据输入数据流的事务处理类型决定执行哪个处理过程

根据中心数据结构的当前状态触发进行执行



六、仓库风格


知识源
知识源
直接存取：知识源不断改变黑板的状态。
黑板
（共享数据）
按照与应用程序相关的层次组织的问题求解数据
黑板状态驱动控制：使用特定的知识。
知识源
知识源
独立、分离的与应用程序相关的知识及其处理单元

六、仓库风格

黑板系统主要有三个部分组成：

- **知识源**：知识源中包含独立的、与应用程序相关的知识，知识源之间不直接进行通信，它们之间的交互通过黑板来完成。
- **黑板数据结构**：黑板数据是按照与应用程序相关的层次来组织的解决问题的数据，知识源通过不断的改变黑板数据来解决问题。

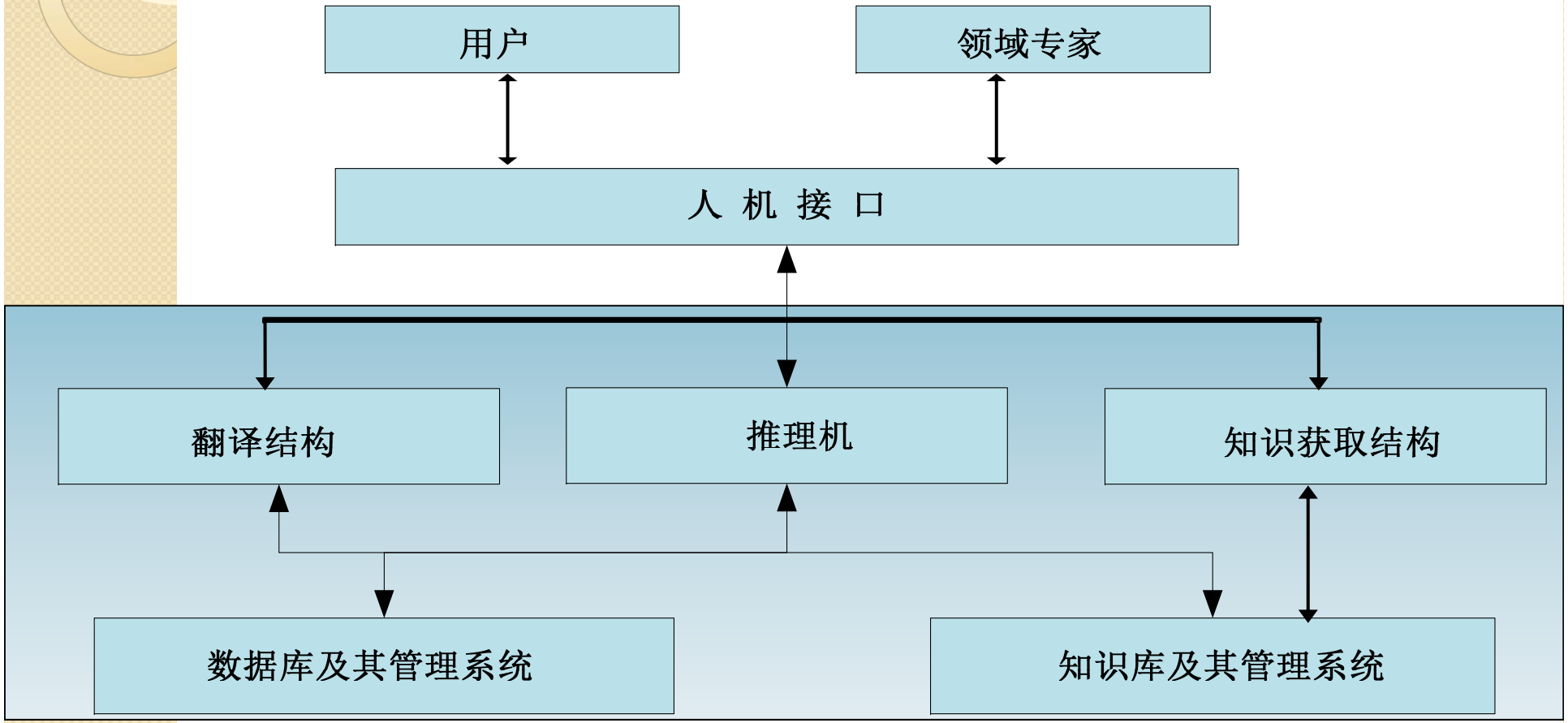
六、仓库风格

- **控制器**：控制完全由黑板的状态驱动，黑板状态的改变决定使用的特定知识。防止两个知识源同时写黑板的某一空间。控制器完全由黑板状态驱动和决定。一旦黑板数据的改变需要知识源的时候，控制器会及时通知知识源随即进行响应。

仓库风格实例

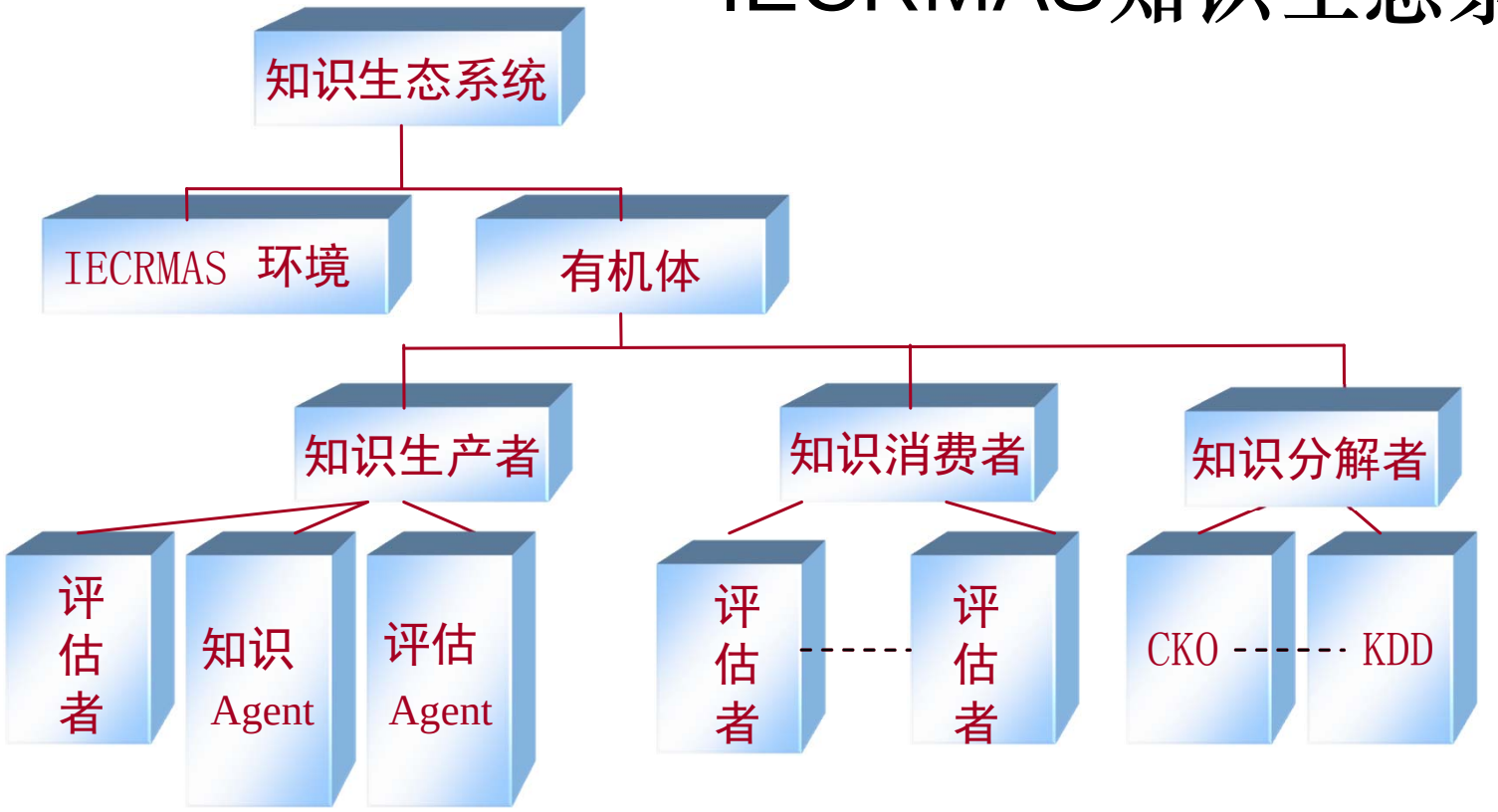
- 专家系统（ES，Expert System）
  - 专家系统实质就是一组程序；
  - 从功能上：可定义为“一个在某领域具有专家水平解题能力的程序系统”，能像领域专家一样工作，运用专家积累的工作经验与专门知识，在很短时间内对问题得出高水平的解答。
  - 从结构上讲：可定义为“由一个专门领域的知识库，以及一个能获取和运用知识的机构构成的解题程序系统”。

- ES一般结构如下图所示：



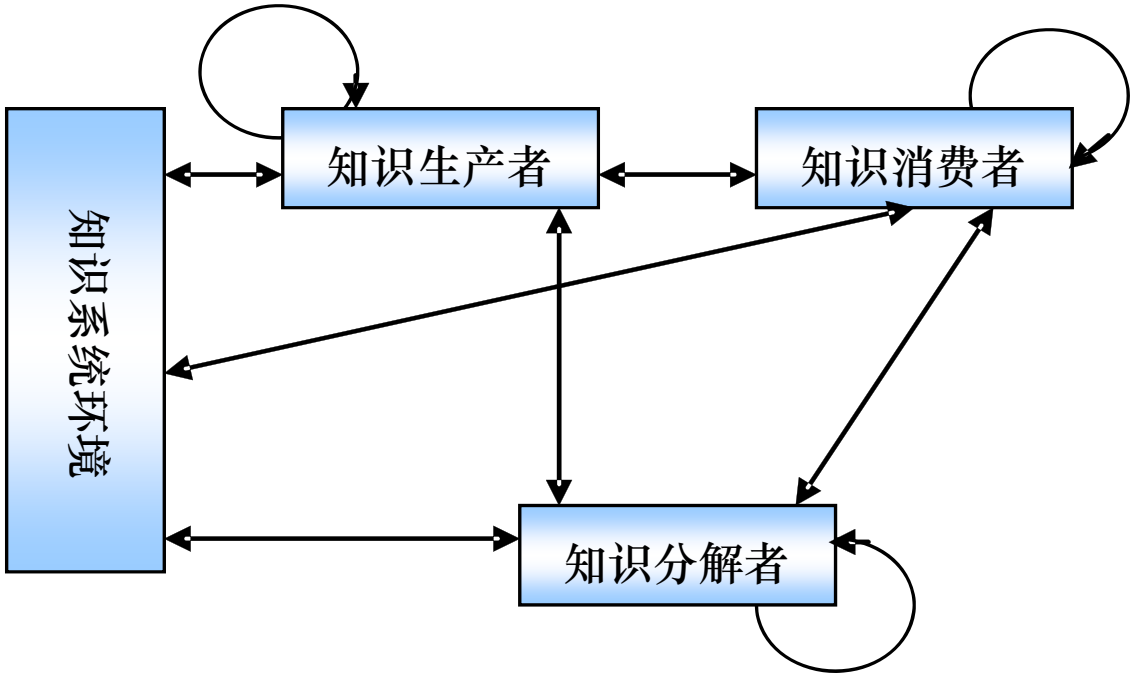
仓库风格实例系统

IECRMAS知识生态系统

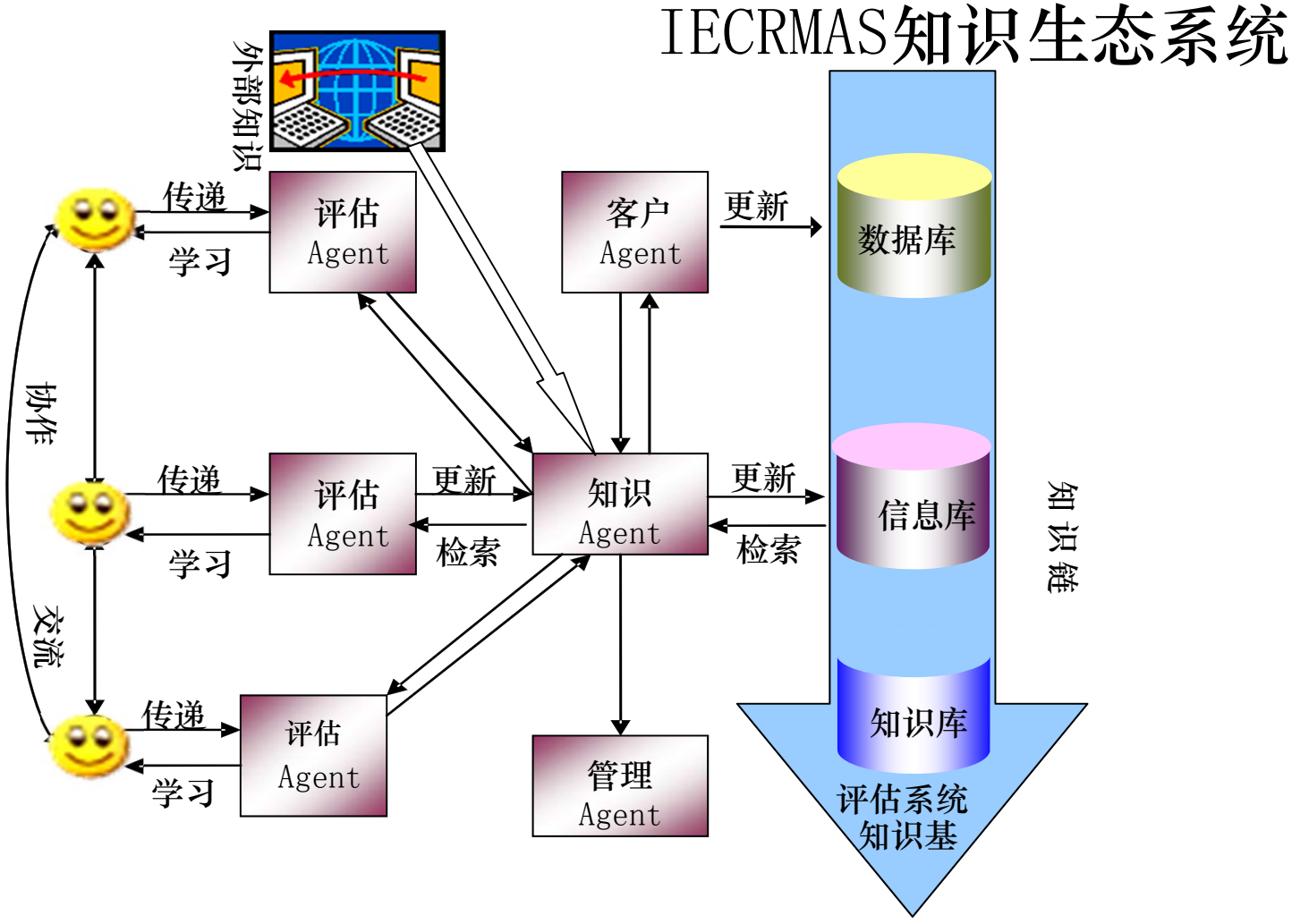


IECRMAS知识生态系统的构建

- IECRMAS知识生态系统



IECRMAPS知识生态系统中知识流动



仓库风格优点

- 解决问题的多方法性：
  - 对于一个专家系统，针对于要解决的问题，如果在其领域中没有独立的方法存在，而且对解空间的完全搜索也是不可行的，在黑板模式中可以用多种不同的算法来进行试验，并且也允许用不同的控制方法。
- 具有可更改性和可维护性：
  - 因为在黑板模式中每个知识源是独立的，彼此之间的通信通过黑板来完成，所以这使整个系统更具有可更改性和可维护性。

仓库风格优点

- 有可重用的知识源：
  - 由于每个知识源在黑板系统中都是独立的，如果知识源和所基于的黑板系统有理解相同的协议和数据，我们就可以重用知识源。
- 支持容错性和健壮性：
  - 在黑板模式中所有的结果都是假设的，并且只有那些被数据和其它假设强烈支持的才能够生存。这对于噪声数据和不确定的结论有很强的容错性。

仓库风格缺点

- 测试困难：
  - 由于黑板模式的系统有中央数据构件来描述系统的体现系统的状态，所以系统的执行没有确定的顺序，其结果的可再现性比较差，难于测试。
- 不能保证有好的求解方案：
  - 一个黑板模式的系统所提供给我们的往往是所解决问题的一个百分比，而不是最佳的解决方案。
- 效率低：
  - 黑板模式的系统在拒绝错误假设的时候要承受多余的计算开销，所以导致效率比较低。

仓库风格缺点

- 开发成本高：
  - 绝大部分黑板模式的系统需要用几年的时间来进化，所以开发成本较高。
- 缺少对不并行机的支持：
  - 黑板模式要求黑板上的中心数据同步并发访问，所以缺少对不并行机的支持。

七、C/S风格

- 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。

七、C/S风格

- 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：数据库服务器、客户应用程序和网络。

C/S风格体系结构

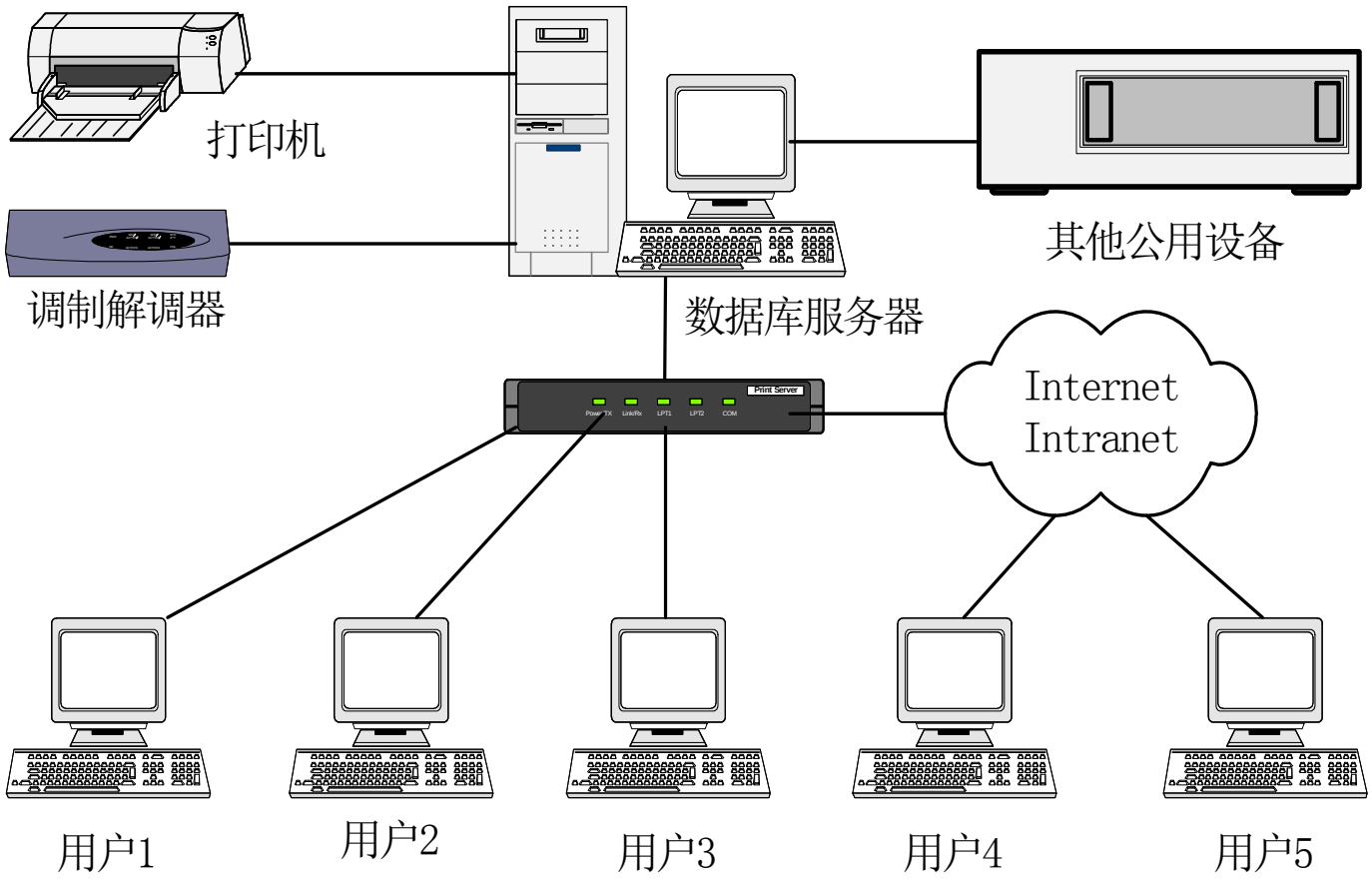


图 3-6 C/S体系结构示意图

C/S风格任务分配

- 服务器

（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。

C/S风格任务分配

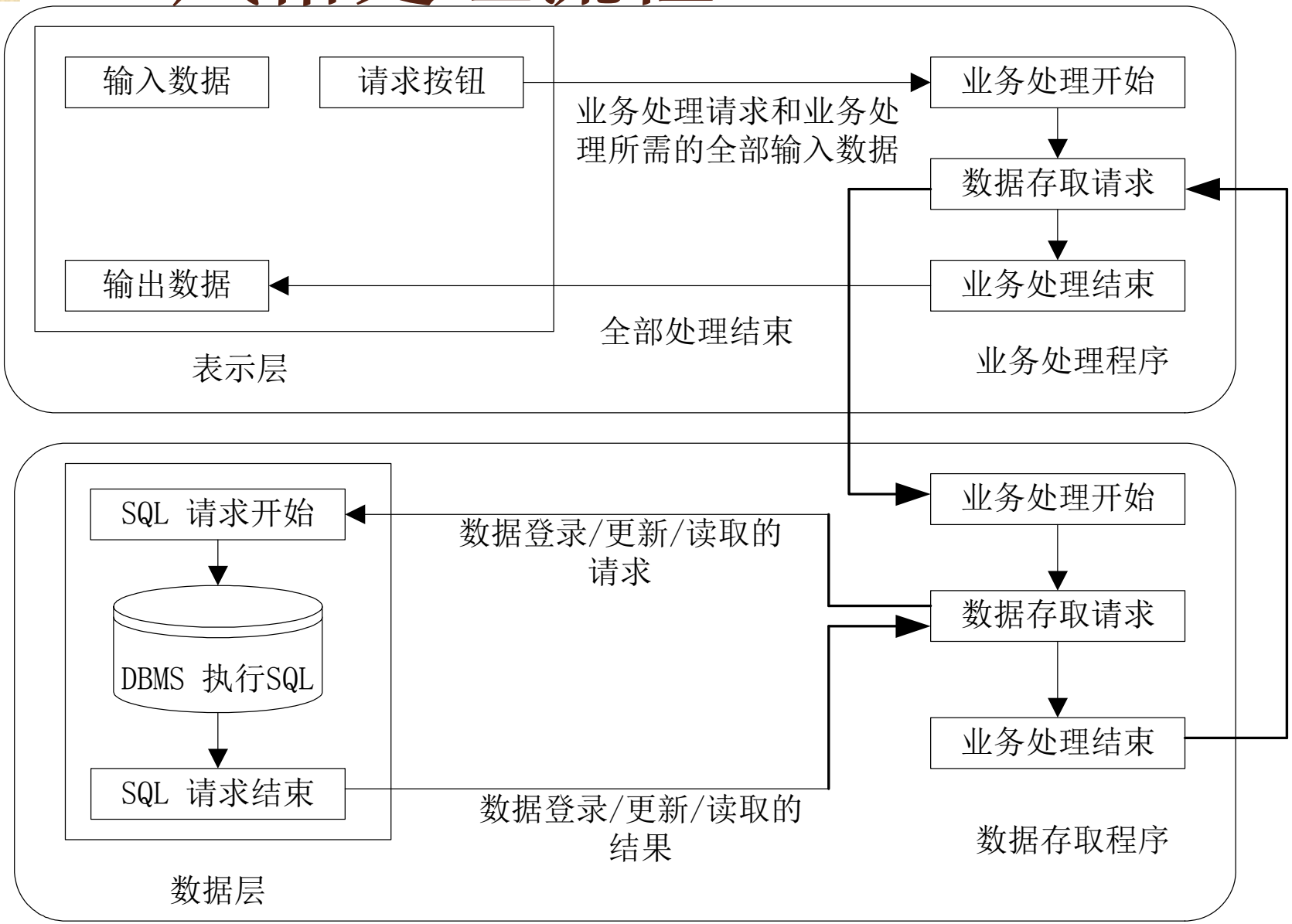
- 客户应用程序

（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客户端的数据执行应用逻辑要求。

C/S风格处理流程


输入数据
请求按钮
业务处理请求和业务处理所需的全部输入数据
业务处理开始
数据存取请求
业务处理结束
输出数据
全部处理结束
表示层
业务处理程序
SQL 请求开始
DBMS 执行SQL
SQL 请求结束
数据登录/更新/读取的请求
数据登录/更新/读取的结果
业务处理开始
数据存取请求
业务处理结束
数据层
数据存取程序

C/S风格优点

◎ C/S 体系结构具有强大的数据操作和事务处理能力，模型思想简单，易于人们理解和接受。

◎ 系统的客户应用程序和服务器构件分别运行在不同的计算机上，系统中每台服务器都可以适合各构件的要求，这对于硬件和软件的变化显示出极大的适应性和灵活性，而且易于对系统进行扩充和缩小。

◎ 在C/S体系结构中，系统中的功能构件充分隔离，客户应用程序的开发集中于数据的显示和分析，而数据库服务器的开发则集中于数据的管理，不必在每一个新的应用程序中都要对一个DBMS进行编码。将大的应用处理任务分布到许多通过网络连接的低成本计算机上，以节约大量费用。

C/S风格缺点

- 二层C/S结构是单一服务器且以局域网为中心的，所以难以扩展至大型企业广域网或Internet;
- 软、硬件的组合及集成能力有限;
- 客户端程序设计复杂，客户机的负荷太重，难以管理大量的客户机，系统的性能容易变坏;
- 数据安全性不好。因为客户端程序可以直接访问数据库服务器，那么，在客户端计算机上的其他程序也可想办法访问数据库服务器，从而使数据库的安全性受到威胁。
- 软件维护和升级时，每个客户端都需要升级。

八、三层C/S风格

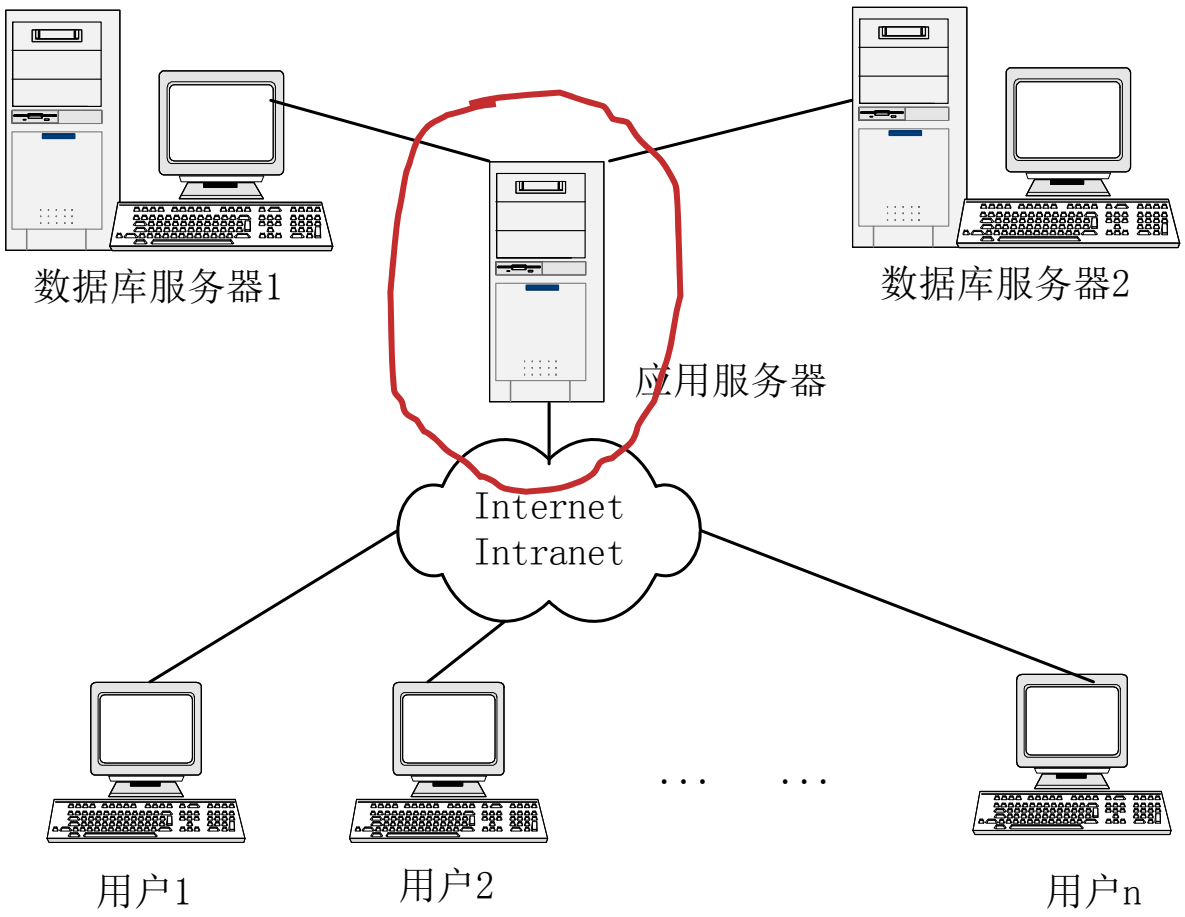
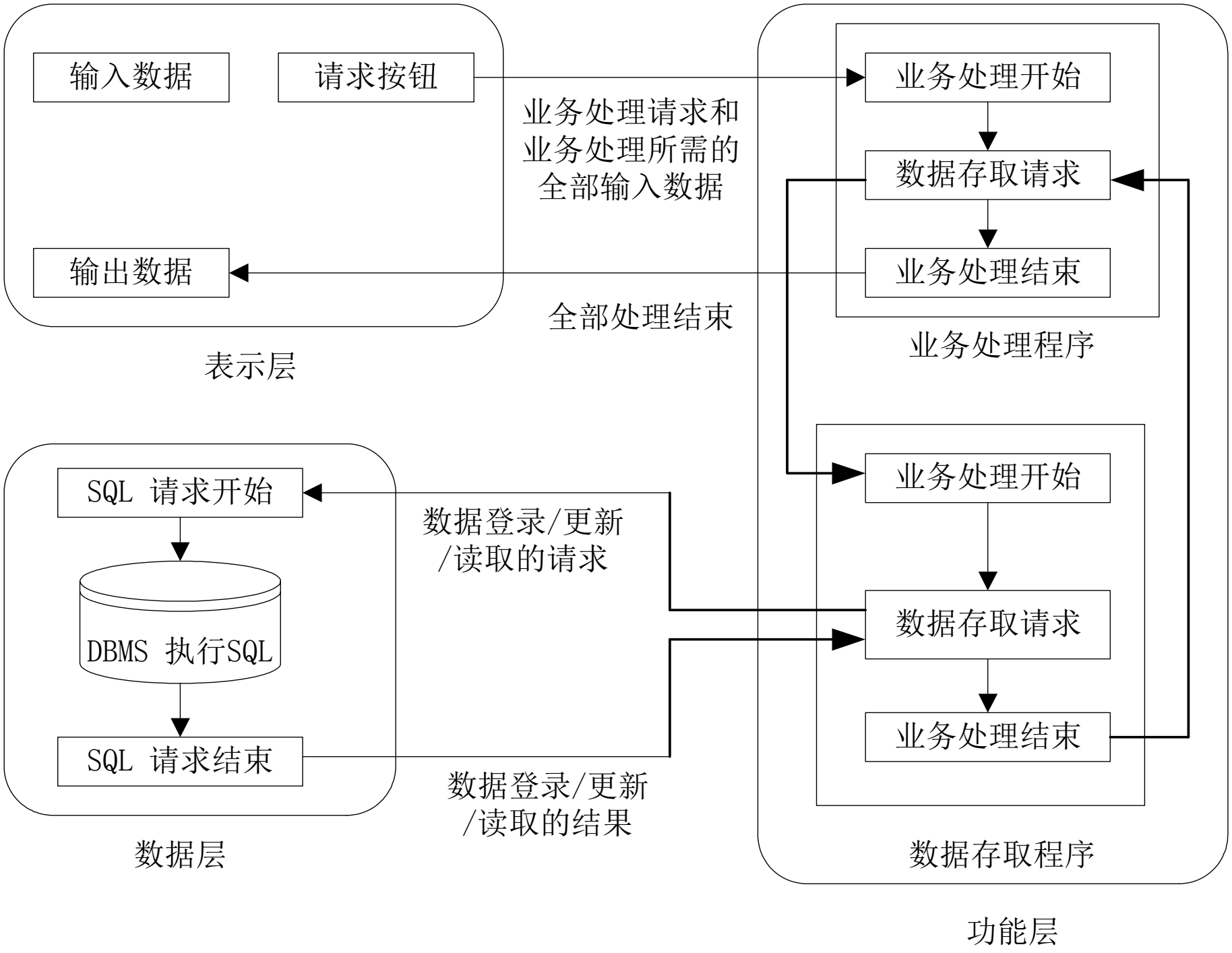


图 3-8 三层C/S体系结构示意图

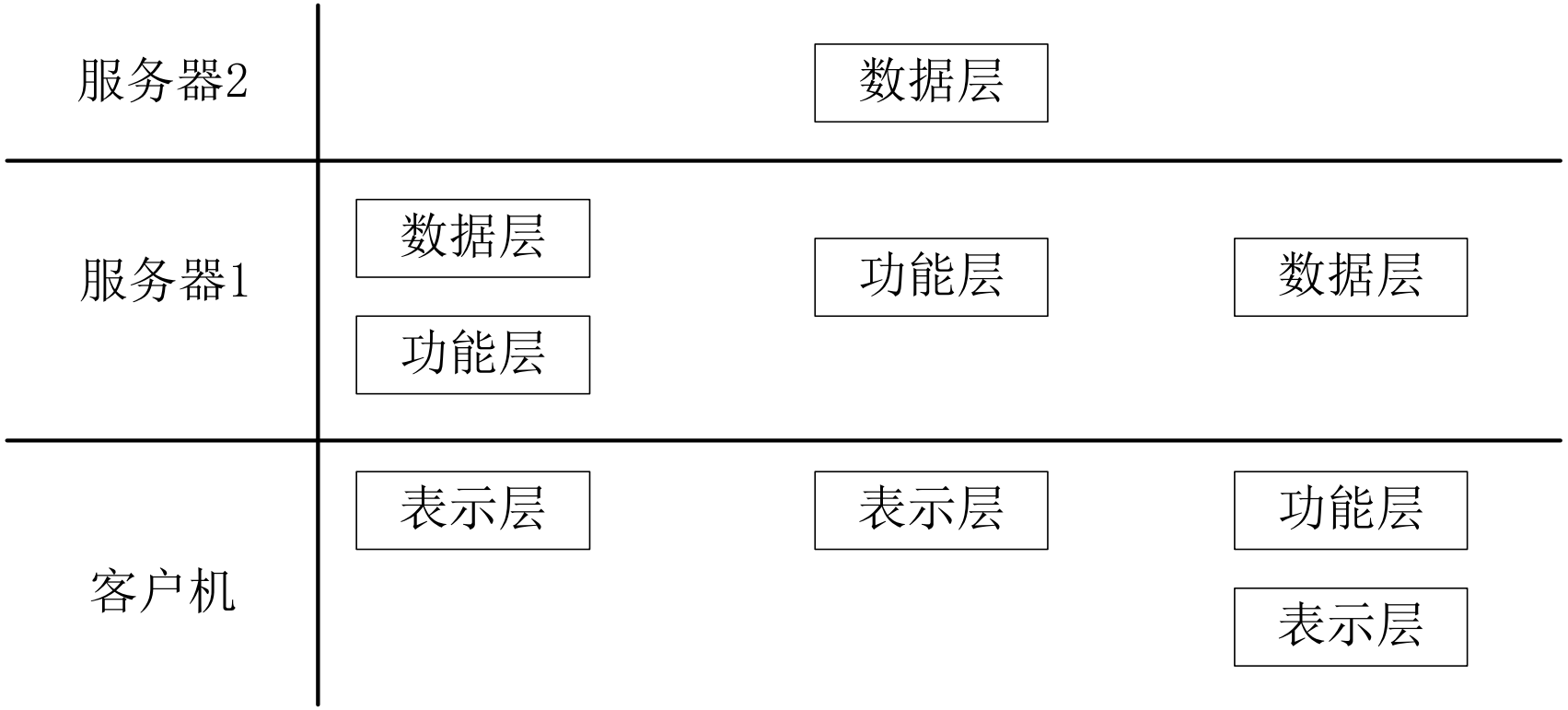

输入数据
请求按钮
输出数据
表示层
业务处理请求和
业务处理所需的
全部输入数据
全部处理结束
业务处理开始
数据存取请求
业务处理结束
业务处理程序
SQL 请求开始
DBMS 执行SQL
SQL 请求结束
数据层
数据登录/更新
/读取的请求
数据登录/更新
/读取的结果
业务处理开始
数据存取请求
业务处理结束
数据存取程序
功能层

表示层是应用的用户接口部分，它担负着用户与应用间的对话功能。它用于检查用户从键盘等输入的数据，显示应用输出的数据。为使用户能直观地进行操作，一般要使用图形用户接口，操作简单、易学易用。在变更用户接口时，只需改写显示控制和数据检查程序，而不影响其他两层。检查的内容也只限于数据的形式和取值的范围，不包括有关业务本身的处理逻辑。

功能层相当于应用的本体，它是将具体的业务处理逻辑编入程序中。例如，在制作订购合同时要计算合同金额，按照定好的格式配置数据、打印订购合同，而处理所需的数据则要从表示层或数据层取得。表示层和功能层之间的数据交往要尽可能简洁。例如，用户检索数据时，要设法将有关检索要求的信息一次性地传送给功能层，而由功能层处理过的检索结果数据也一次性地传送给表示层。通常，在功能层中包含有确认用户对应用和数据库存取权限的功能以及记录系统处理日志的功能。功能层的程序多半是用可视化编程工具开发的，也有使用COBOL和C语言的。

数据层就是数据库管理系统，负责管理对数据库数据的读写。数据库管理系统必须能迅速执行大量数据的更新和检索。因此，一般从功能层传送到数据层的要求大都使用SQL语言。

三层C/S风格物理结构



（1）将数据层和功能层放在同一台服务器上

（2）将数据层和功能层放在不同的服务器上

（3）将功能层放在客户机上

三层C/S风格优点

◎ 允许合理地划分三层结构的功能，使之在逻辑上保持相对独立性，能提高系统和软件的可维护性和可扩展性。

◎ 允许更灵活有效地选用相应的平台和硬件系统，使之在处理负荷能力上与处理特性上分别适应于结构清晰的三层；并且这些平台和各个组成部分可以具有良好的可升级性和开放性。

◎ 应用的各层可以并行开发，可以选择各自最适合的开发语言。

◎ 利用功能层有效地隔离开表示层与数据层，未授权的用户难以绕过功能层而利用数据库工具或黑客手段去非法地访问数据层，为严格的安全管理奠定了坚实的基础。

三层C/S风格要注意的问题

- 三层C/S结构各层间的通信效率若不高，即使分配给各层的硬件能力很强，其作为整体来说也达不到所要求的性能。

- 设计时必须慎重考虑三层间的通信方法、通信频度及数据量。这和提高各层的独立性一样是三层C/S结构的关键问题。

排水管网及污染源信息管理系统

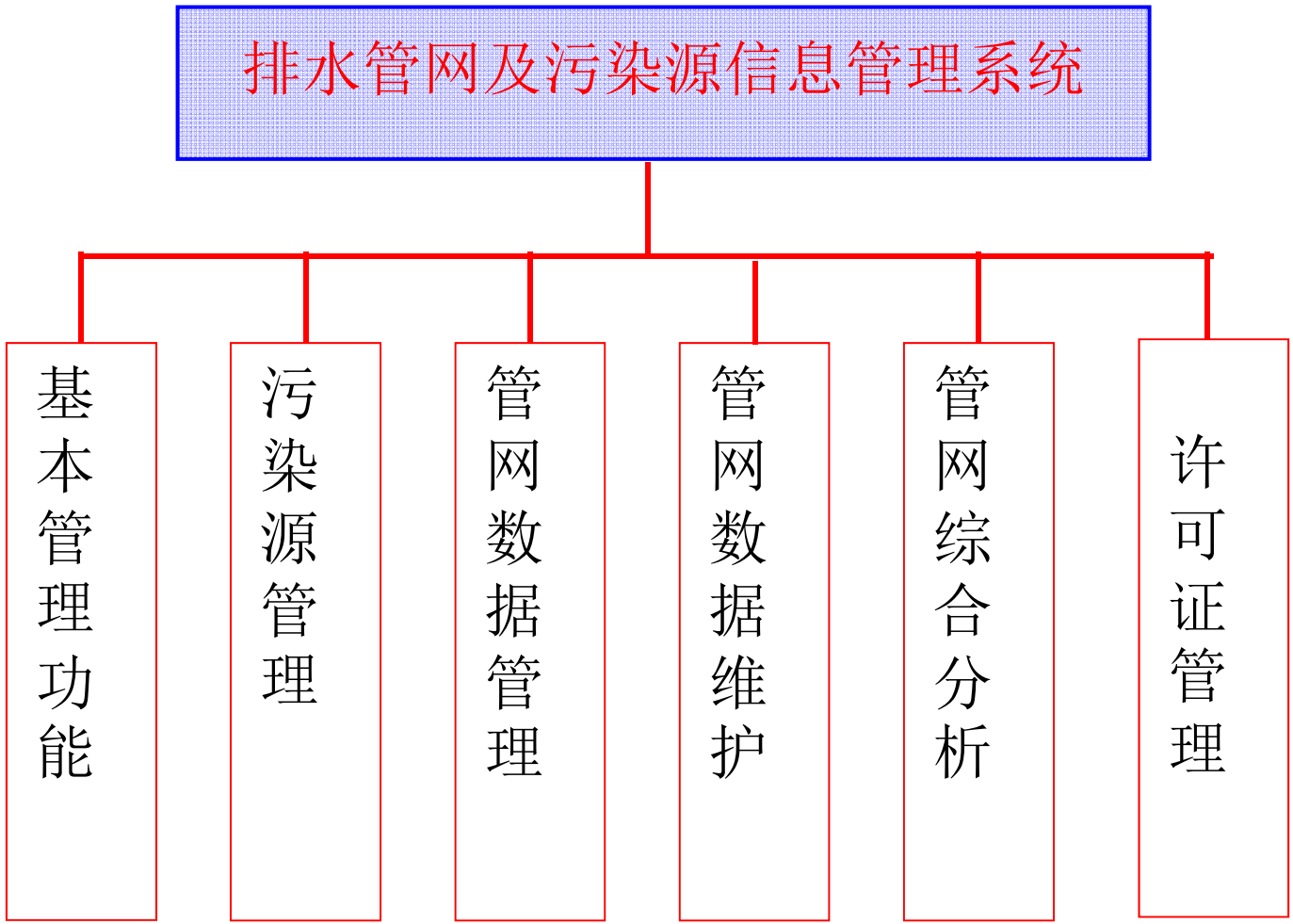
- 系统建设目标

1) 建立排水管线及污染源数据库及基础地形数据库，实现各类信息的数字化存储。

2) 建立排水管网及污染源信息管理系统，实现各部门对这些信息的共享。系统具有数据输入、编辑、查询、统计、分析、输出、更新等管理功能。

3) 建立数据更新机制，实现排水管网及污染源信息的动态管理，为城市的河道污染治理提供决策依据和技术服务。

系统功能框架图



系统体系结构

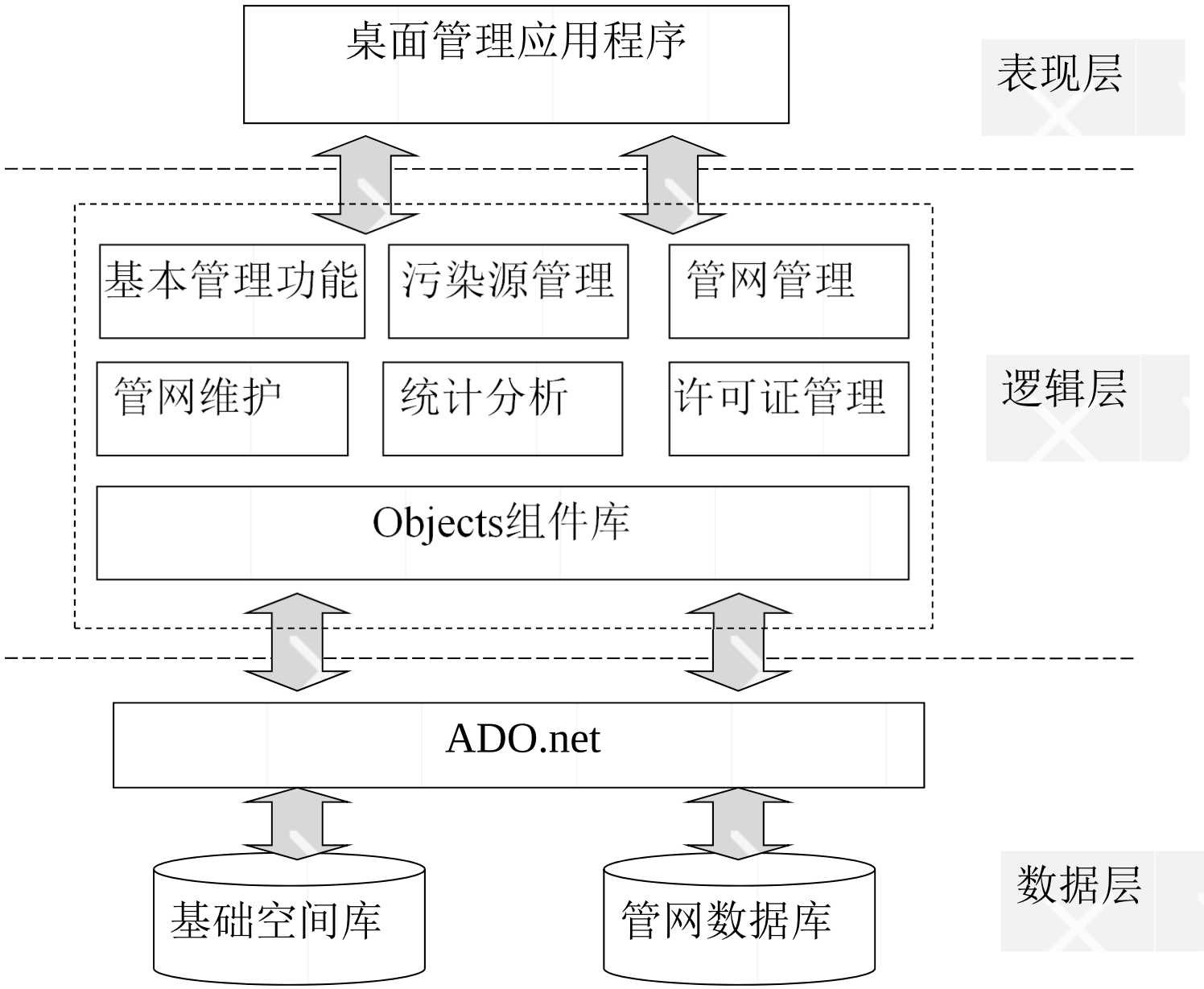
系统采用三层体系结构：

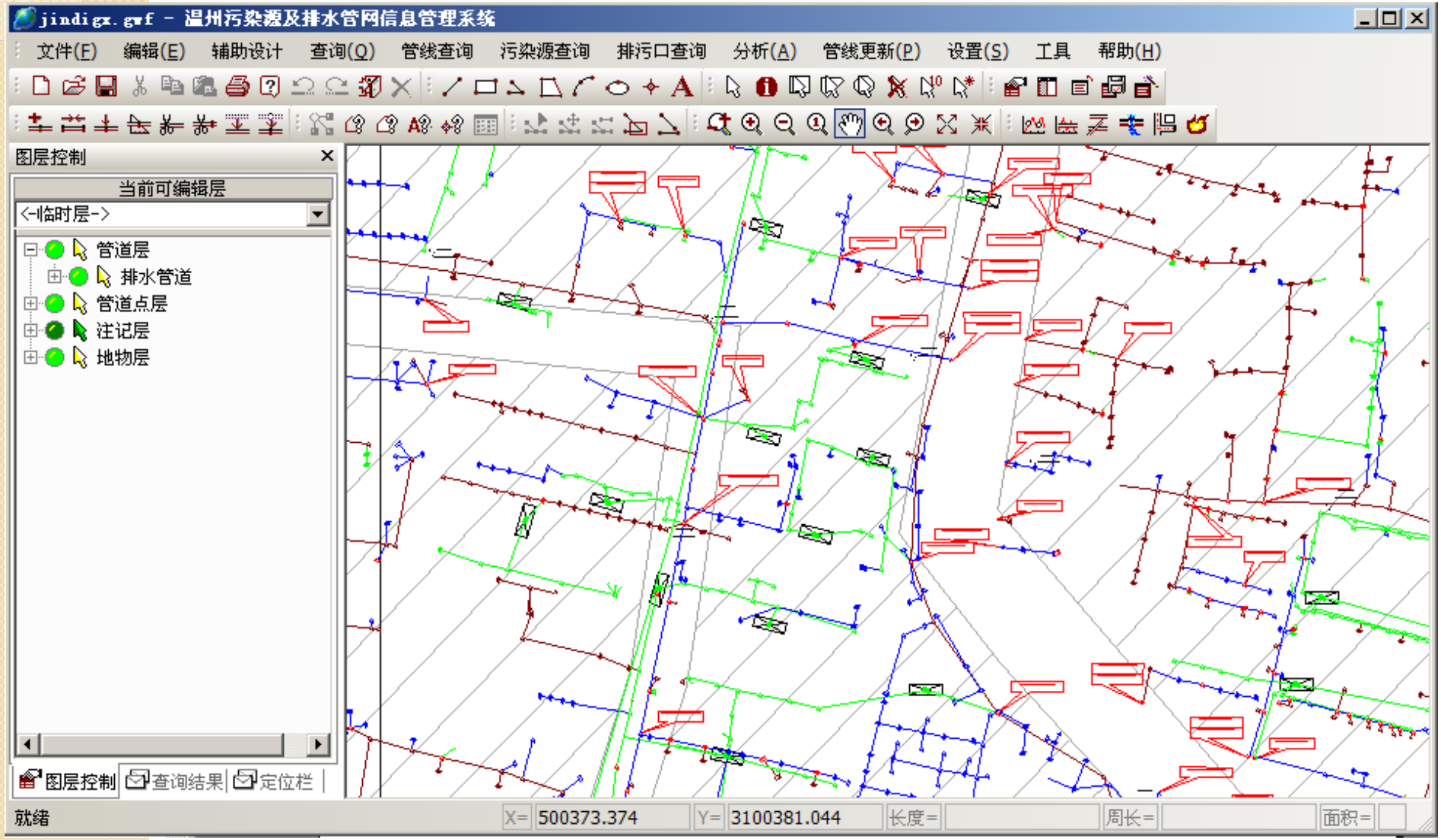
(1) 数据层：采用关系型数据库，实现各类数据的高效存储和管理。

(2) 逻辑层：采用Objects组件，通过空间数据引擎，负责数据库系统业务逻辑的实现。

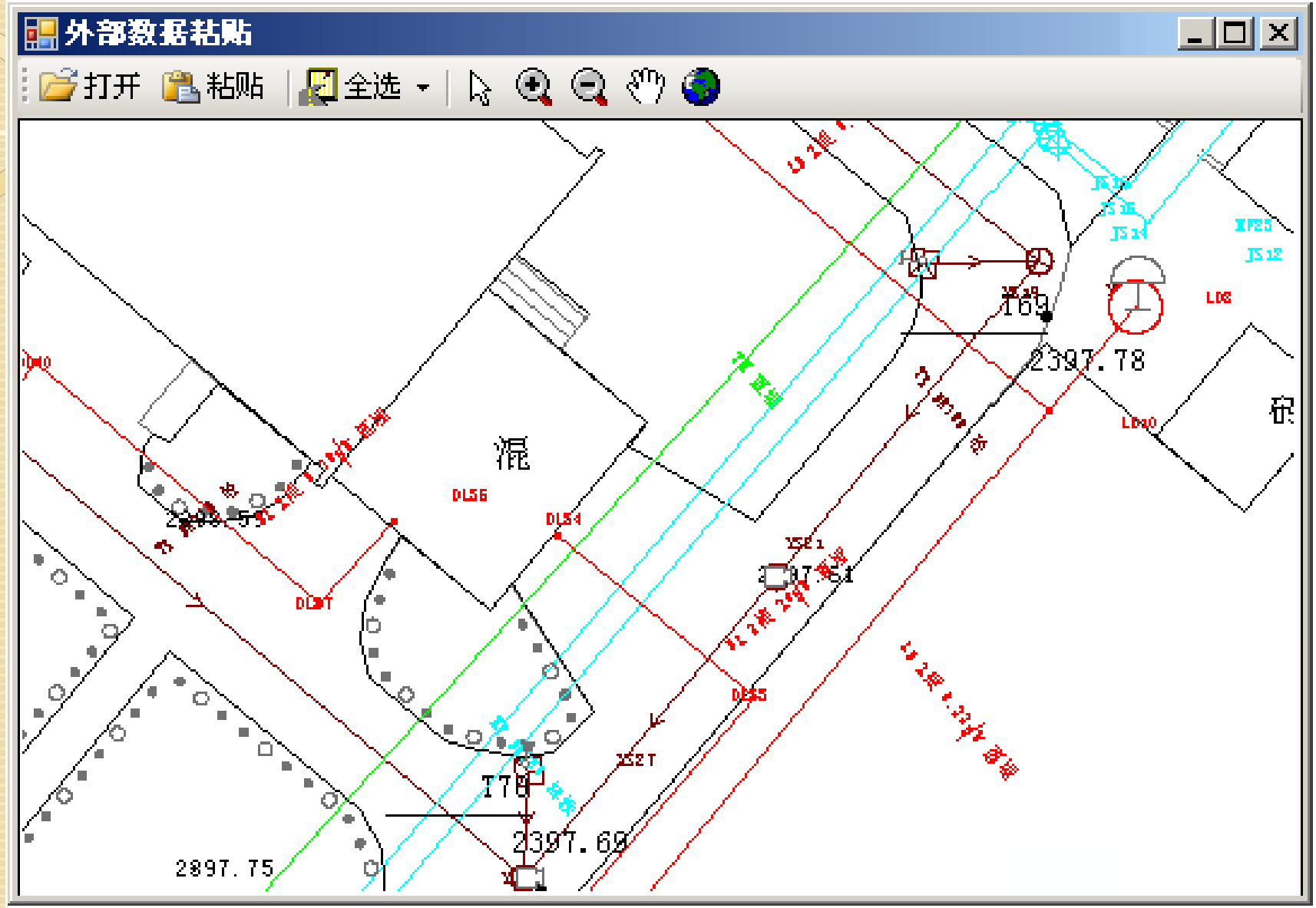
(3) 表现层：排水管网及污染源信息管理系统，满足相关部门对排水管线及污染源管理的要求。

系统体系结构图


桌面管理应用程序
表现层
基本管理功能
污染源管理
管网管理
管网维护
统计分析
许可证管理
逻辑层
Objects组件库
ADO.net
基础空间库
管网数据库
数据层


jindigx.gwf - 温州污染源及排水管网信息管理系统
文件(F) 编辑(E) 辅助设计 查询(Q) 管线查询 污染源查询 排污口查询 分析(A) 管线更新(P) 设置(S) 工具 帮助(H)
图层控制
当前可编辑层
<-临时层->
管道层
排水管道
管道点层
注记层
地物层
图层控制
查询结果
定位栏
就绪
X= 500373.374
Y= 3100381.044
长度=
周长=
面积=

管网数据维护

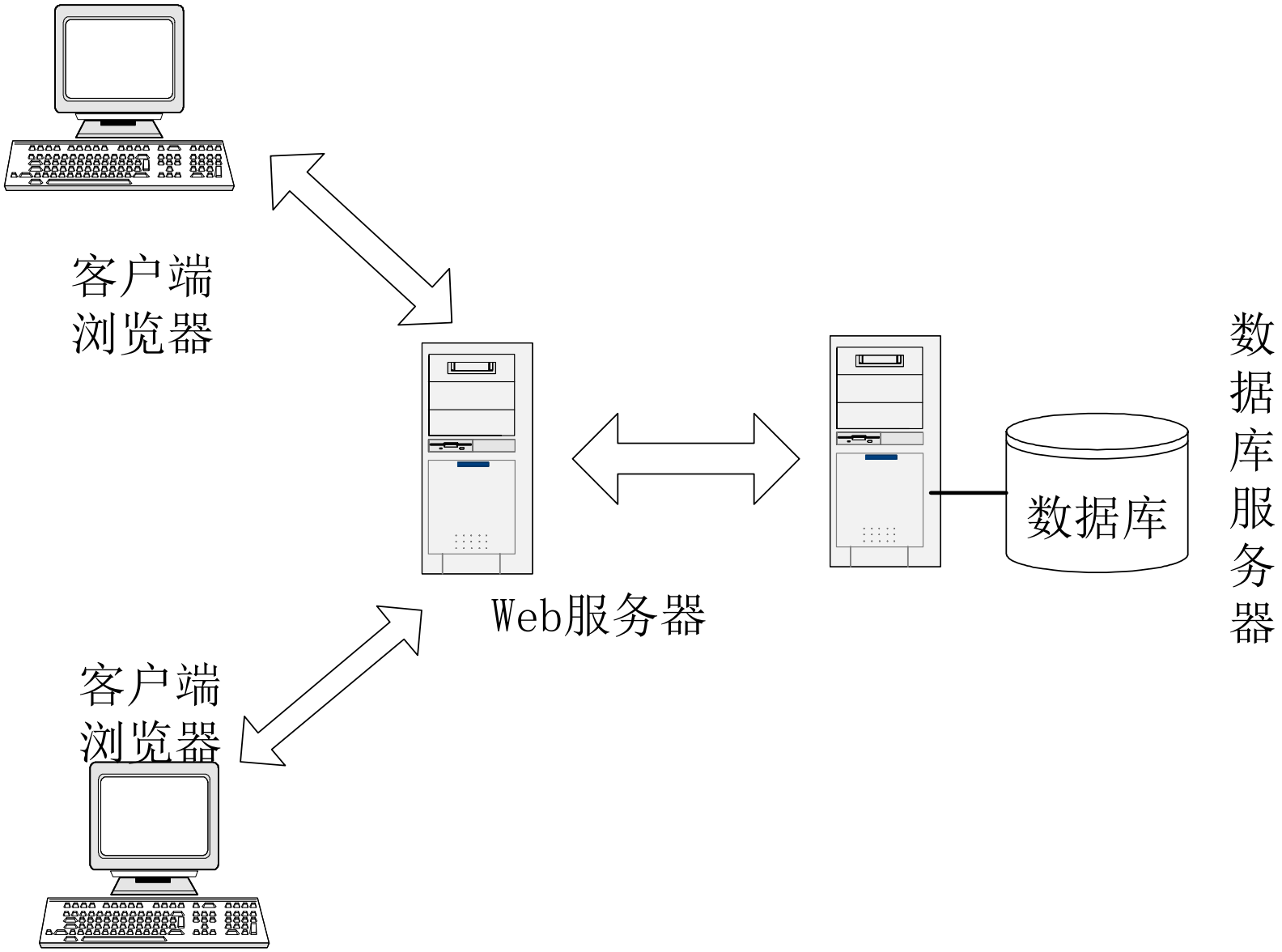


九、B/S风格

- 浏览器/服务器（B/S）风格就是上述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不断成熟的Web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。从某种程度上来说，B/S结构是一种全新的软件体系结构。

B/S风格体系结构



B/S风格优点

- 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。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。

- B/S体系结构还提供了异种机、异种网、异种应用服务的联机、联网、统一服务的最现实的开放性基础。

B/S风格缺点

- B/S体系结构的系统扩展能力差，安全性难以控制。

- 采用B/S体系结构的应用系统，在数据查询等响应速度上，要远远地低于C/S体系结构。

- B/S体系结构的数据提交一般以页面为单位，数据的动态交互性不强，不利于在线事务处理(OLTP)应用。

十、B/S与C/S混合软件体系结构

- 在三层C/S体系结构中，表示层负责处理用户的输入和向客户的输出。功能层负责建立数据库的链接，根据用户的请求生成访问数据库的SQL语句，并把结果返回给客户端。数据层负责实际的数据存储和检索，相应功能层的数据处理要求，并将结果返回给功能层。
- B/S是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

十、 B/S与C/S混合软件体系结构

- **B/S与C/S**混合软件体系结构是一种典型的异构体系结构。

- B/S软件体系结构，即Browser/Server （浏览器/服务器）结构，是随着Internet技术的兴起，对C/S体系结构的一种变化或者改进的结构。

十、B/S与C/S混合软件体系结构

- 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。即使在某段时间内某一种标准占统治地位，但变动最终是绝对的。

- 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。然而在许多场合，将技术与经济综合进行考虑时，总是决定不再重写它们。

- 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。

C/S与B/S混合之内外有别模型

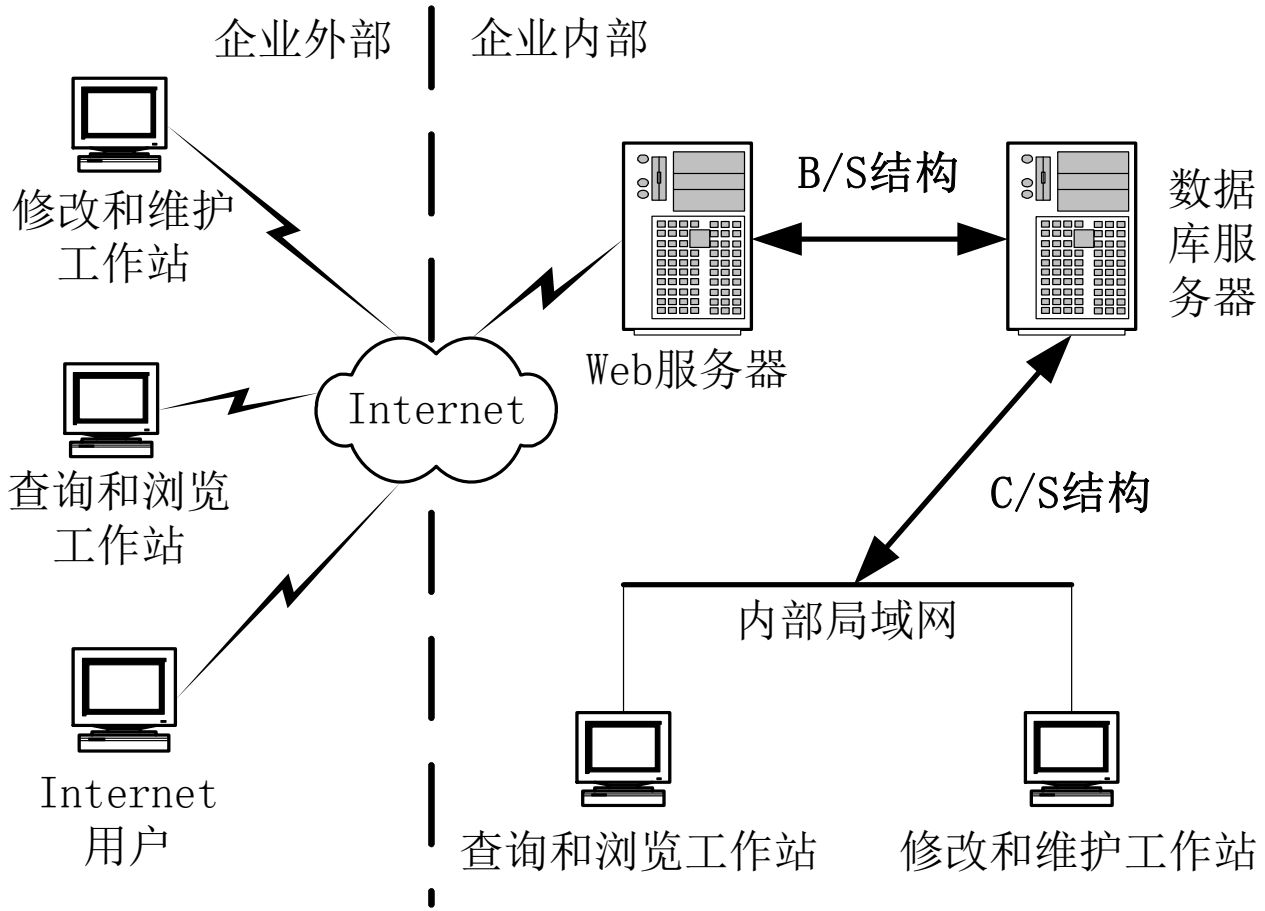


图 3-26“内外有别”模型

C/S与B/S混合之内外有别模型

●企业内部用户通过局域网直接访问数据库服务器，软件系统采取C/S体系结构；

●企业外部用户通过Internet访问Web服务器，通过Web服务器再访问数据库服务器，软件系统采用B/S结构。

●优点在于：外部用户不直接访问数据库服务器，保证数据库安全，企业内部用户的交互性较强，数据查询和修改的响应速度较快。

●缺点在于：外部用户修改和维护数据，速度较慢，较繁琐。

C/S与B/S混合之查改有别模型

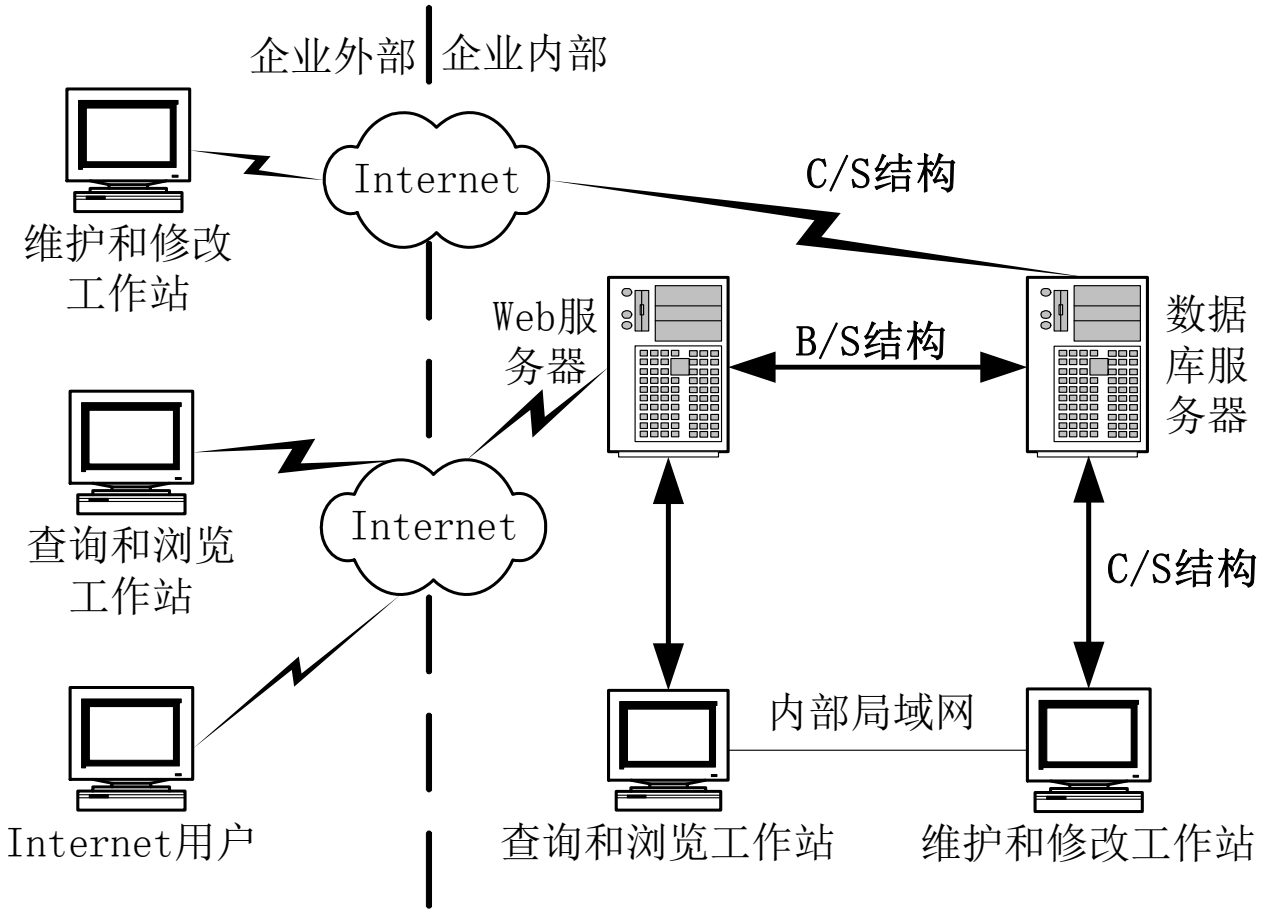


图 3-27“查改有别”模型

C/S与B/S混合之查改有别模型

- 无论用户通过局域网还是外网连接到系统，凡是需要执行维护和修改数据操作的，都使用C/S体系结构；
- 凡是需要执行查询浏览操作的，都使用B/S体系结构。
- 此结构可以拥有两种体系结构的共同优点；
- 缺点在于：企业数据容易暴露给外部用户，数据安全有一定威胁。

B/S与C/S混合软件体系结构案例

例如:
在实现某个变电站信息管理系统解决方案中，就使用了C/S与B/S混合软件体系结构的方式，其结构如图8所示。

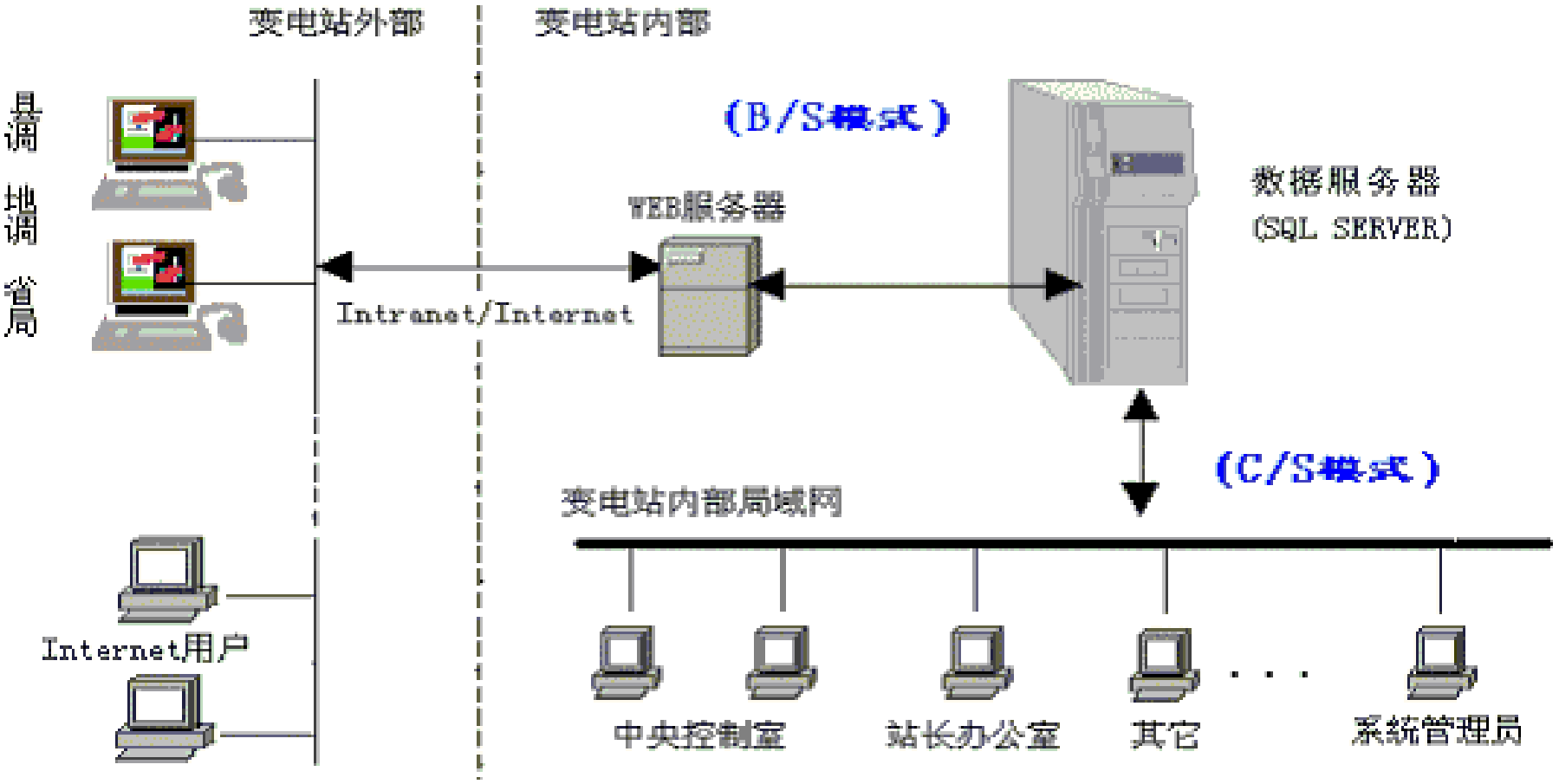


图8.　C/S与B/S混合软件体系结构

B/S与C/S混合软件体系结构案例

- 变电站内部用户通过局域网直接访问数据库服务器，外部用户（包括县调、地调和省局的用户及普通Internet用户）通过Internet访问Web服务器，再通过Web服务器访问数据库服务器。
- 该解决方案把B/S和C/S这两种软件体系结构进行了有机的结合，扬长避短，有效地发挥了各自的优势。同时，因外部用户只需一台接入Internet的计算机，就可以通过Internet查询运行生产管理情况，无须做太大的投入和复杂的设置。这样也方便所属电业局及时了解各变电站所的运行生产情况，对各变电站的运行生产进行宏观调控。

十一、小结

- 软件体系结构风格为大粒度的软件重用提供了可能。然而，对于应用体系结构风格来说，由于视点的不同，系统设计师有很大的选择空间。要为系统选择或设计某一个体系结构风格，必须根据特定项目的具体特点，进行分析比较后再确定，体系结构风格的使用几乎完全是特化的。

- 从上面的介绍中，我们知道，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。事实上，也存在一些系统，它们是由这些纯体系结构组合而成，即采用了异构软件体系结构。