上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

三层架构介绍

智能制造与信息工程研究所

2017年5月4日

目录

一、引例

饭店就餐

服务员只管接待客人

厨师只管烹炒客人要的美食

采购员只管按客人需求采购肉，蔬菜

他们各负其责共同协作为客人提供美食

顾客

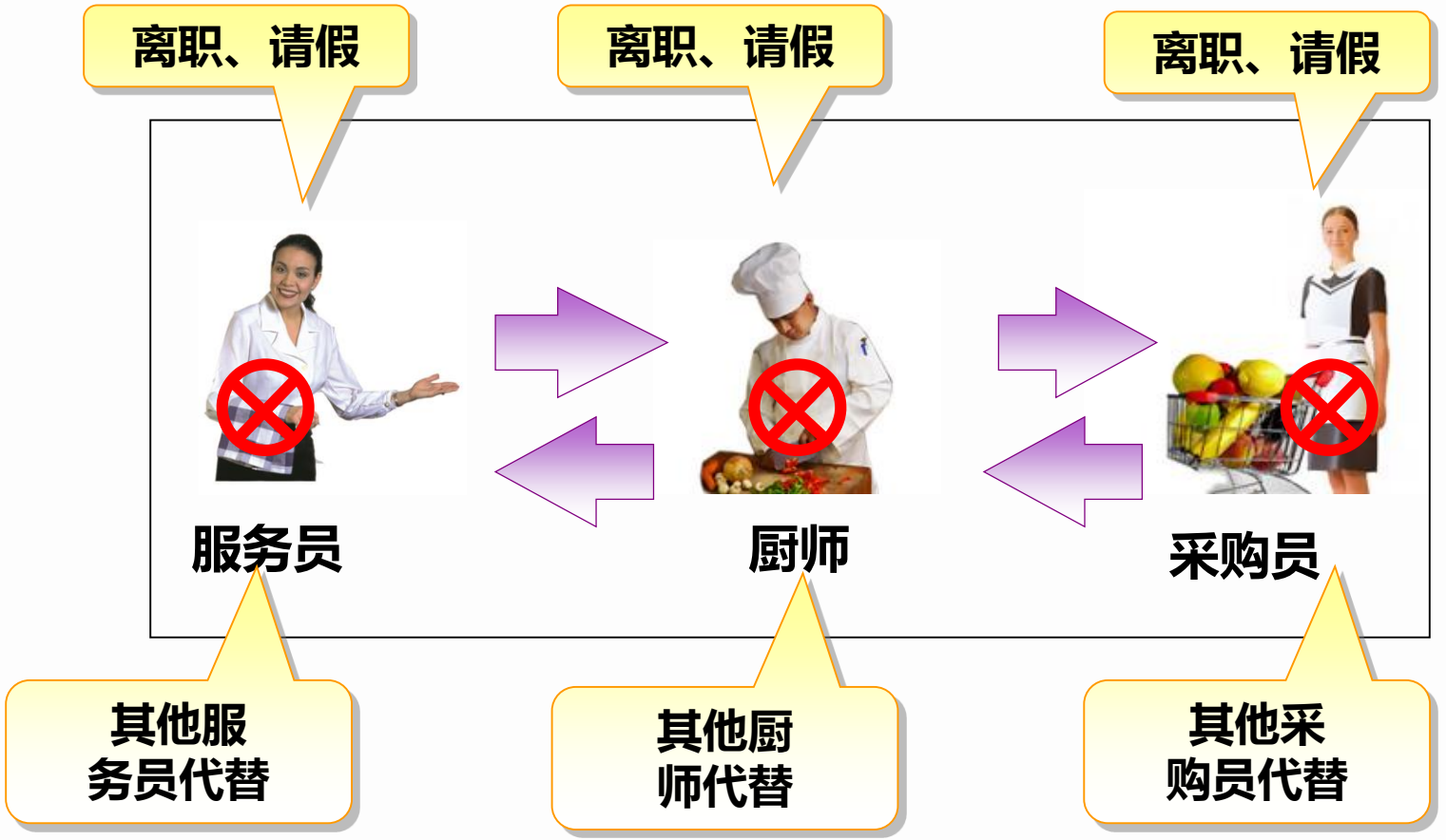
饭店

服务员

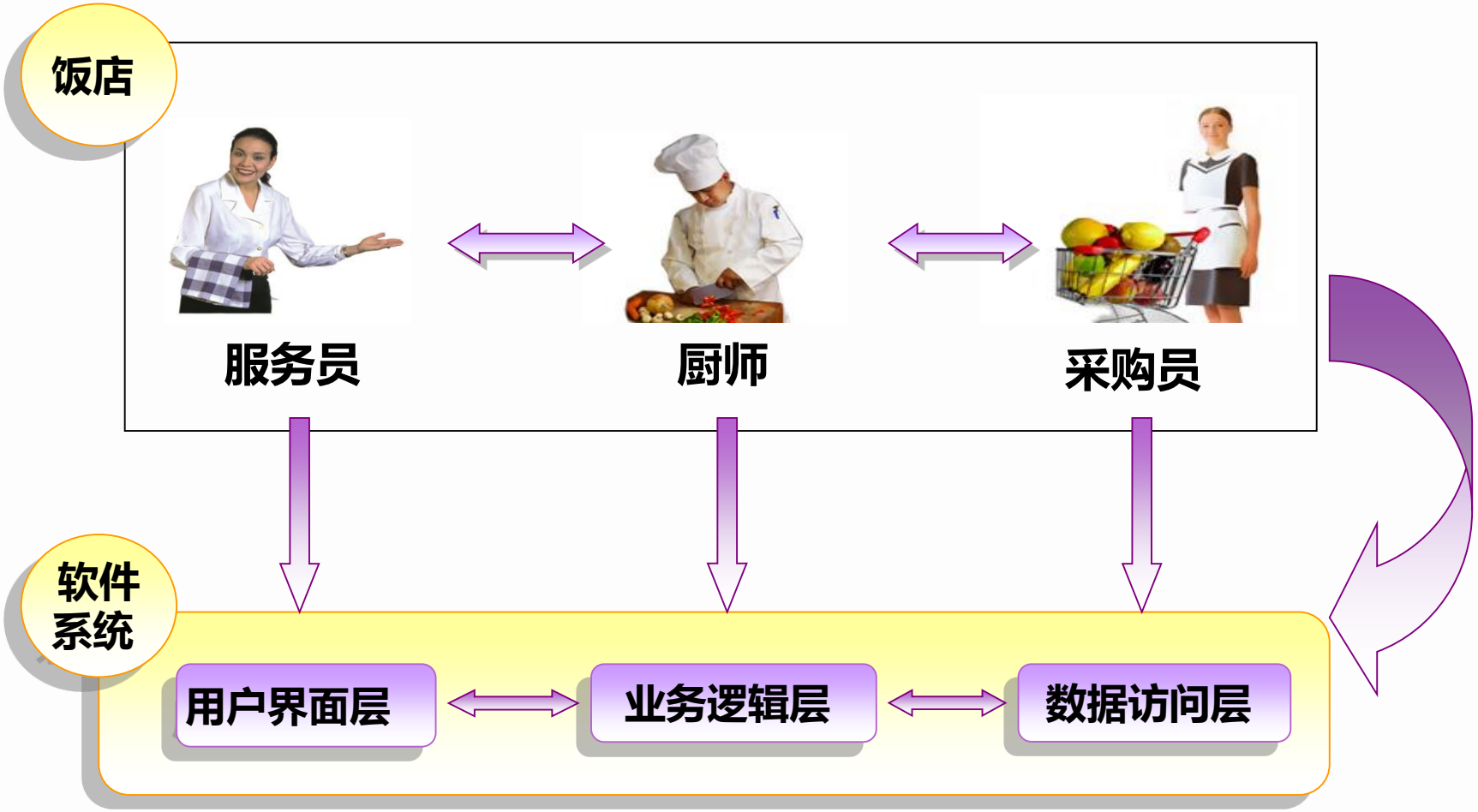
厨师

采购员

一、引例



一、引例



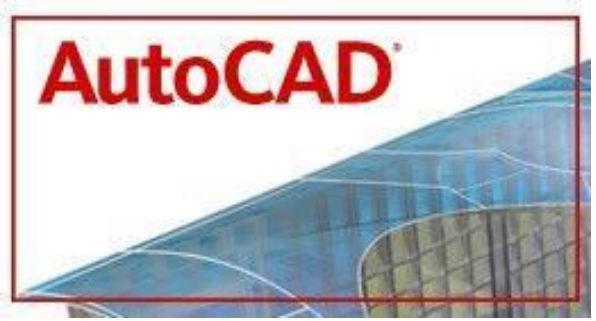
三层结构软件模型

目录

二、软件架构概述

软件架构（software architecture）

- 指在一定的设计原则基础上，从不同角度对组成系统的各部分进行搭配和安排，形成系统的多个结构而组成架构，它包括该系统的各个组件，组件的外部可见属性及组件之间的相互关系。组件的外部可见属性是指其他组件对该组件所做的假设。
- 在现代软件开发当中，软件架构设计起到至关重要的作用
- 自面向对象的语言普遍使用以后，促成了团队合作设计的热潮
- 尤其是大型复杂软件，没有一个好的软件架构，软件设计与开发几乎成了不可能完成的任务。

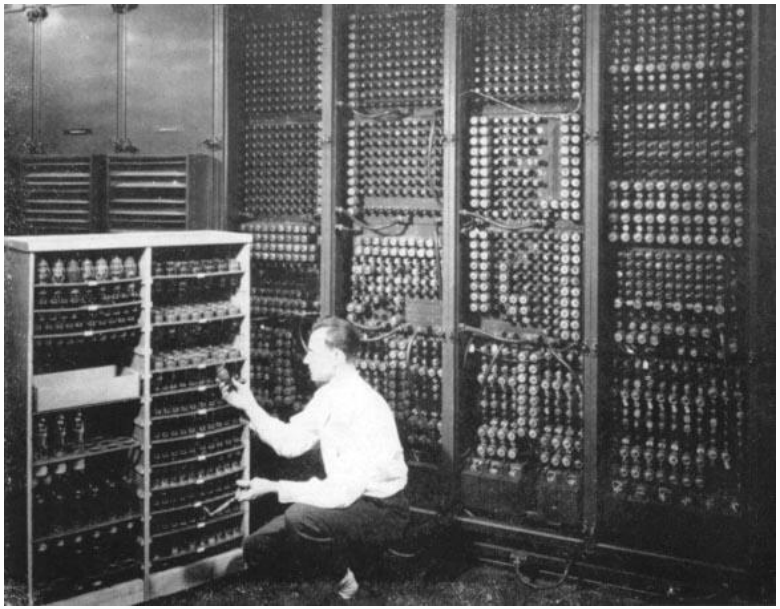


二、软件架构概述

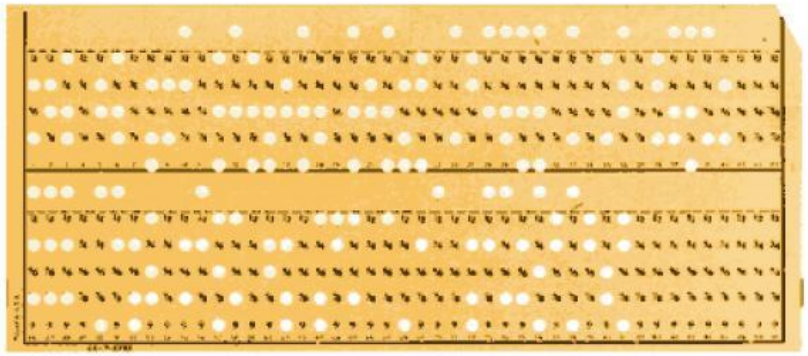
- 软件架构演变

集中式架构

- 出现于电子计算机新兴时期
- 程序功能单一
- 代码量小
- 计算机执行速度慢
- 使用二进制编码和机器指令
- 编程语言简单
- 输入输出混杂，兼具命令交互功能



第一台电子计算机“EMIAC”

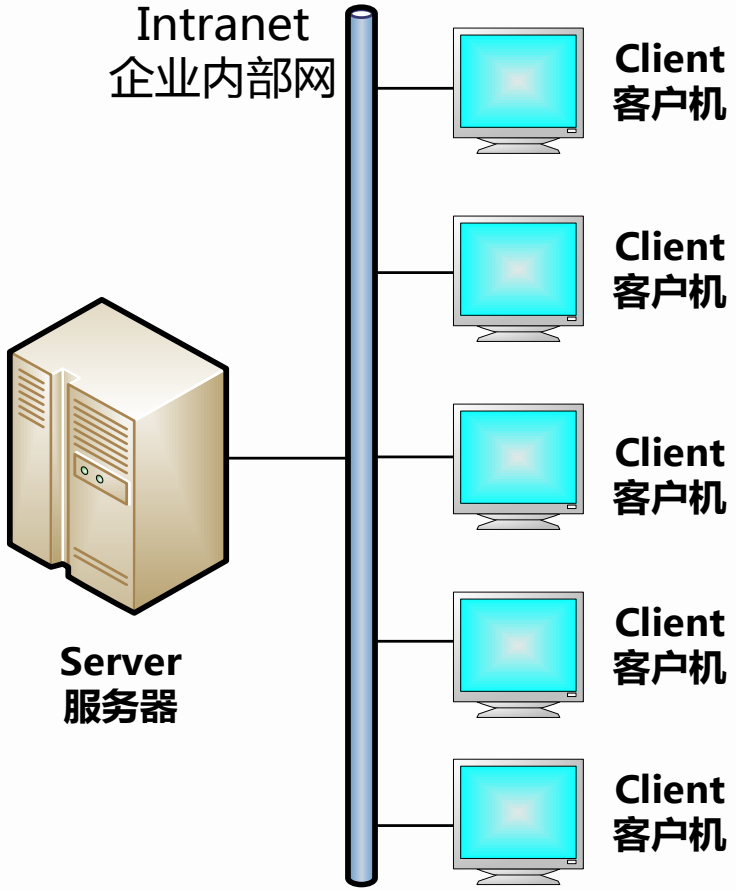


向EMIAC输入指令的穿孔纸带

二、软件架构概述

软件架构演变

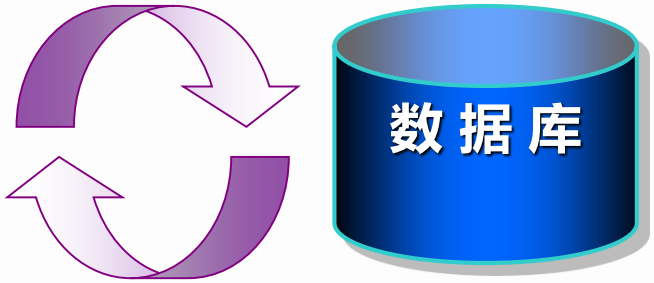
- 客户机/服务器架构
  - 两层结构，应用程序逻辑通常分布在客户和服务器两端，客户端发出数据资源访问请求，服务器端将结果返回客户端，客户端将数据进行计算（可能涉及到运算、汇总、统计等等）并将结果呈现给用户。
  - 在过去应用系统开发过程中得到了广泛的应用



二、软件架构概述

软件架构演变

- 客户机/服务器两层架构缺点
  - 难以维护：结构用户界面、业务逻辑和数据逻辑相互交错，通常在第一次部署时较容易，但难以升级或改进。
  - 难以扩展：随着系统的升级或应用的需求发生变化，客户端和服务器端的应用程序都需要进行修改，系统复杂成都大大增加，难以扩展，给应用维护和升级带来了极大的不便；
  - 安全性差：客户端程序可以直接访问数据库，可以通过编程语言或者数据库提供的工具直接对数据库进行操作，不安全。
  - 性能不佳：客户端直接与数据库建立连接，当有大量的并发用户存在时，会使数据库不堪重负，性能迅速下降，甚至死机。
  - 负载繁重：大量的数据传输增加了网络的负载

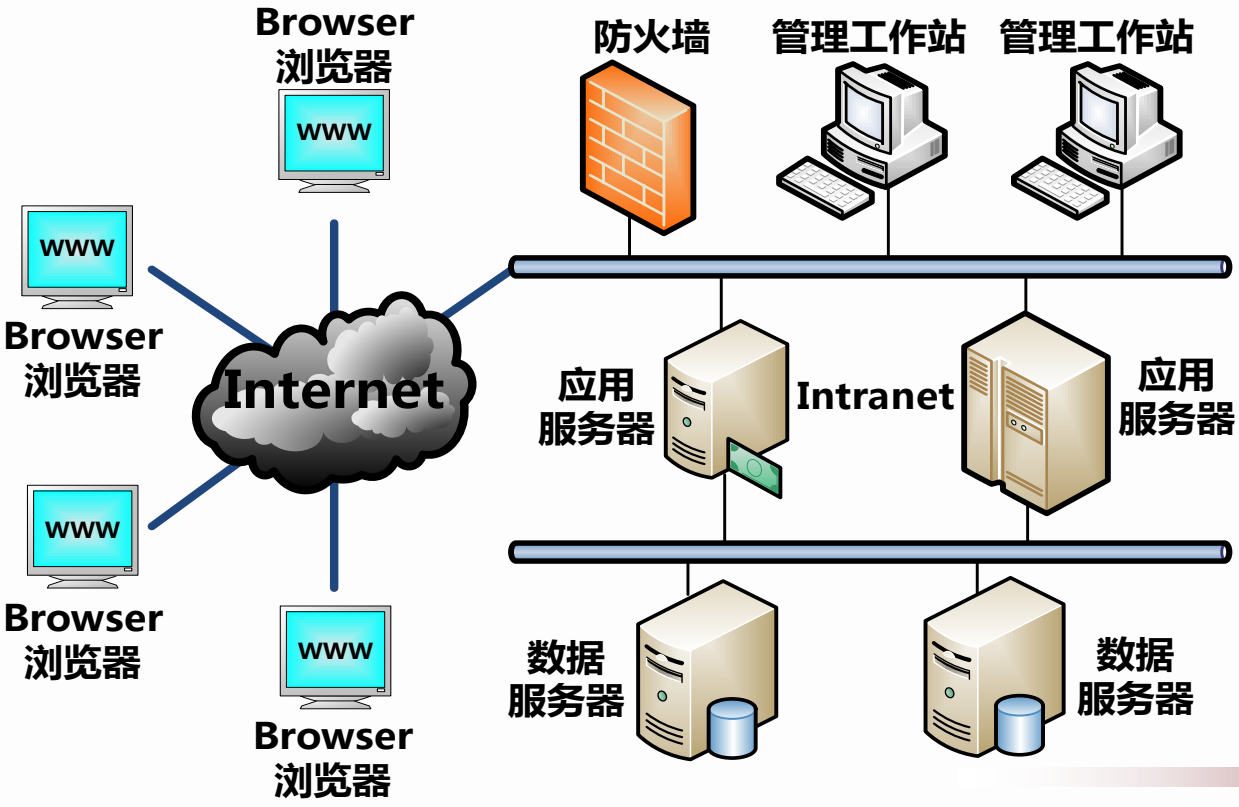


二、软件架构概述

软件架构演变

三层架构模式

- 在客户端与数据库之间加入了一个“中间层”，负责应用业务处理
- 客户端接受用户的输入并向应用服务提出请求，应用服务从数据库服务中获得数据，进行计算并将结果提交给客户端，客户端将结果呈现给用户。



二、软件架构概述

软件架构演变

- **三层架构模式优点**
  - 体现了“任务职责分解”的思想：让专业的人和设备来做专业的事
  - 开发人员可以只关注整个结构中的其中某一层；
  - 可以很容易的用新的实现来替换原有层次的实现；
  - 可以降低层与层之间的依赖；
  - 有利于标准化；
  - 利于各层逻辑的复用。
  - 代码的可读性和功能的扩展性有着很好的提高
- **四层乃至更多层次架构的停滞**
  - 多层设计降低了系统的性能。这是不言而喻的。如果不采用分层式结构，很多业务可以直接造访数据库，以此获取相应的数据，如今却必须通过中间层来完成。
  - 有时会导致级联的修改。这种修改尤其体现在自上而下的方向。如果在表示层中需要增加一个功能，为保证其设计符合分层式结构，可能需要在相应的业务逻辑层和数据访问层中都增加相应的代码。

目录

三、三层架构

三层架构定义

- 在软件开发过程中，运用分层、分模块的思想来设计软件结构，将每一类型的操作固定在一个层(或模块)中。
- 常见的是在客户端与数据库之间加入了一个“中间层”，也叫组件层，从而形成了三层结构。
- 提示
  - 超脱于所谓的C/S与B/S之上，不是指物理上的三层，不是简单地放置三台机器就是三层体系结构，也不仅仅有B/S应用才是三层体系结构。三层结构实际上是在C/S体系下盛行的两层结构客户端/服务器层的一种进化。
- 要点
  - 把同类型的操作作为单独的类模块抽象出来,供其它模块重复调用
- 目的
  - 不仅提高软件的可用性，同时也可以大大提高软件的重用性和拓展性，实现“高内聚、低耦合”

分散关注、松散耦合、逻辑复用、标准定义

三、三层架构

三层架构组成

- 数据访问层（Data Access Layer，简称DAL）,这一层的工作

主要是对原始数据（数据库或者文本文件等存放数据的形式）的操作，注意是对数据的操作，而不是数据库， 为业务逻辑层或表示层提供数据服务 .

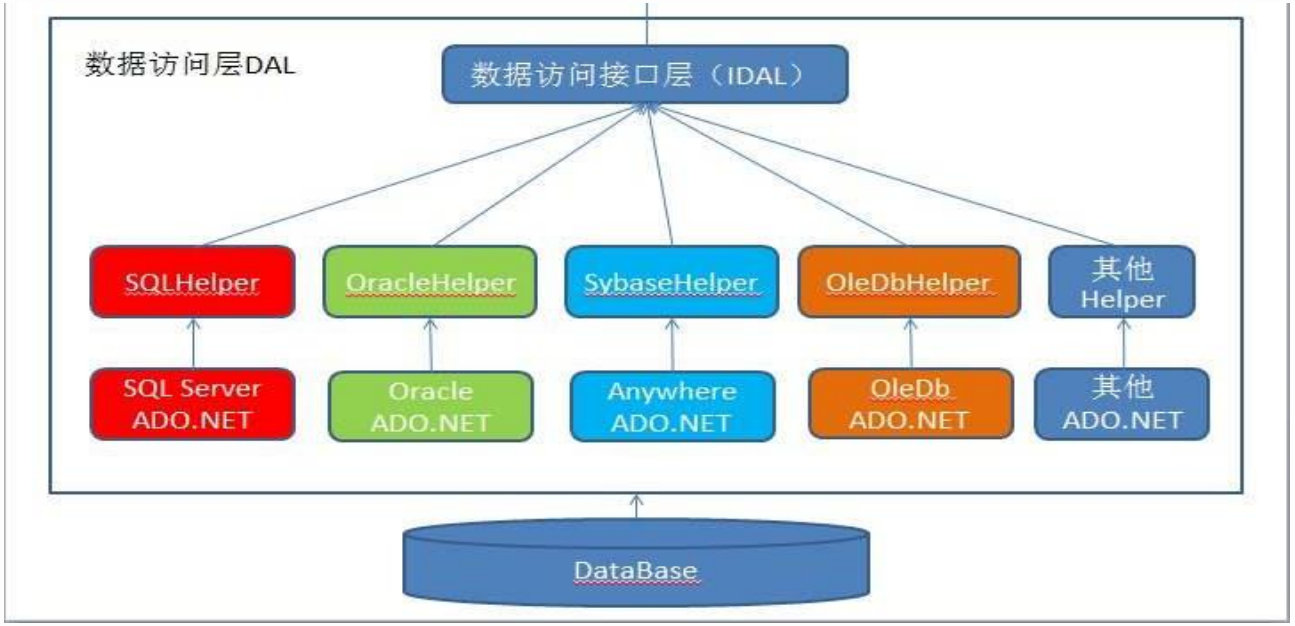
用户通过这一层向系统提交请求或发出指令。系统通过这一层接受用户请求或指令，然后，将指令消化吸收后调用下一层，再将调用的结果展现到这一层。

它是系统的核心业务处理层，负责接收用户界面层的指令和数据，消化吸收后，进行组织业务逻辑的处理，并将结果返回给用户界面层。

三、三层架构

数据访问层

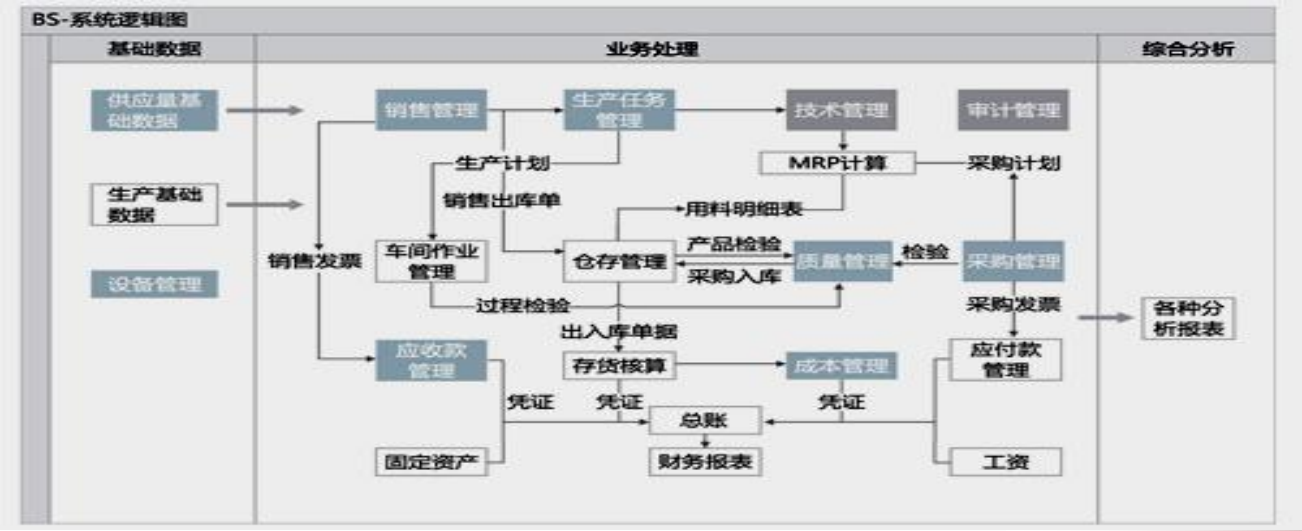
- 数据访问项目执行从数据库（或其他数据服务）获取数据或向数据库发送数据的功能。
  - 从“业务规则”层接收请求，从“数据服务”获取数据或向其发送数据。
  - 使用SQL语言/存储过程对数据库数据进行增删改查操作。
  - 将数据库查询结果返回到“业务规则”层。



三、三层架构

业务逻辑层

- 业务规则项目包含业务对象本身以及应用于它们的规则。
- 主要业务对象所在的位置，它们实现业务实体或系统对象，系统的业务规则将在这些对象中编码。
  - 从“用户界面”层接受请求。
  - 根据编码的业务规则处理请求。
  - 从“数据访问”层获取数据或将数据发送到“数据访问”层。
  - 将处理结果传递回“用户界面”层。



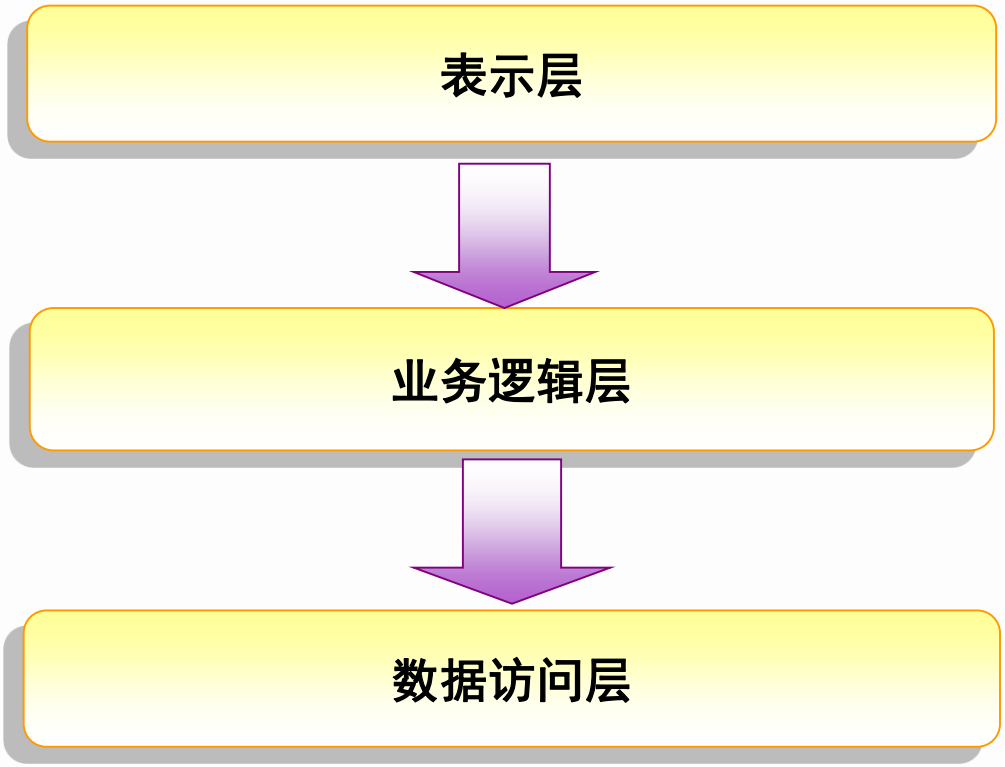
三、三层架构

用户界面层

- 用户界面项目是指在应用程序中实现的客户端。在分布式应用程序结构中，用户服务可以是 Web 客户端或 Windows 客户端，这具体取决于特定的应用程序。例如，在开发Web 应用程序时，可能需要提供具有标准 Windows 用户界面或 Web 用户界面。通常，这种一般类型的应用程序包含以下功能：
  - 管理 Web 页或 Windows 界面的呈现和行为
  - 显示数据
  - 捕获数据
  - 数据验证检查
  - 为用户提供任务指南
  - 向“业务规则”发送用户输入
  - 从“业务规则”接收结果
  - 向用户显示错误

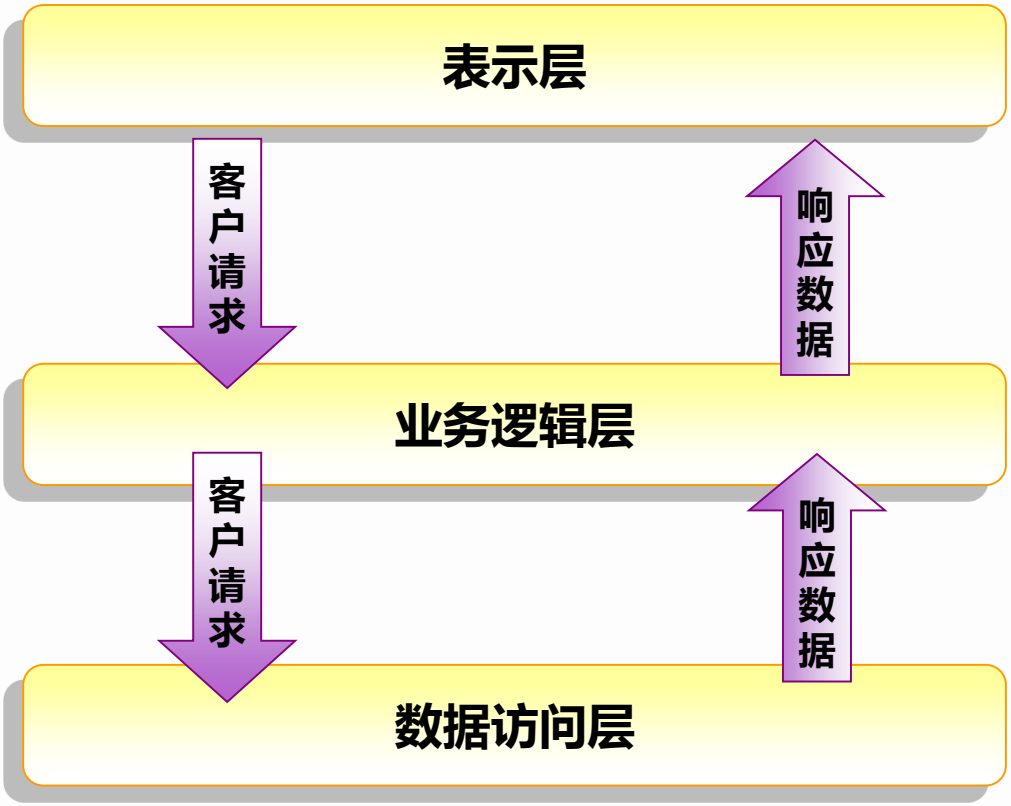
三、三层架构

- 各层依赖关系



三、三层架构

三层间数据传递方向



信息流：

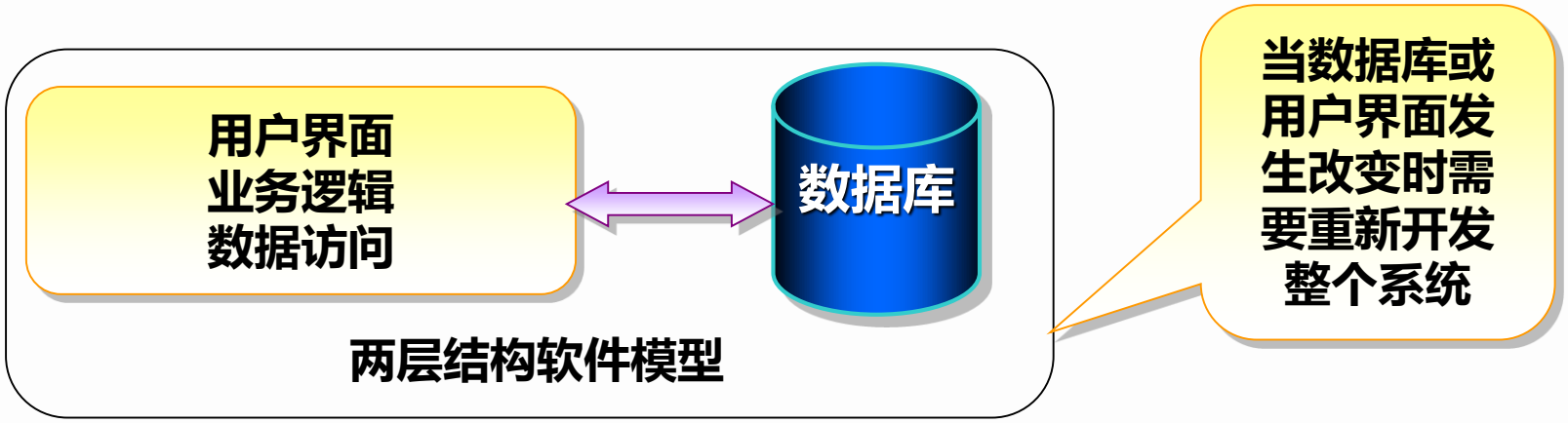
三、三层架构

- 三层架构核心思想——“职责分解，分层处理”
  - 三层结构的设计原则是，表现层和数据层只依赖业务层，而业务层不能依赖表现层和数据层
    - 在其他两层中，最好不要出现任何“业务逻辑”！也就是说，要保证数据访问层和表现层的中的函数功能的原子性，即最小性和不可再分。
  - 示例：一个任务可由三个函数(function)来实现
    - fun1的任务是把界面上的数据读取两个数字中，然后把这两个数据传给fun2，他不管fun2拿这两个数据来干什么的。
    - fun2的任务就是把这两个数字相减或相加，fun1和fun3是不管的（这一层是的操作是根据具体业务来）。
    - 第三个fun3的任务就是访问数据库，把fun2操作运算过的一个数字保存到数据库中
    - 再大一点，可以用三个类来分开实现。更大一点来说，可以建三个不同的项目来实现（企业级架构即是如此）

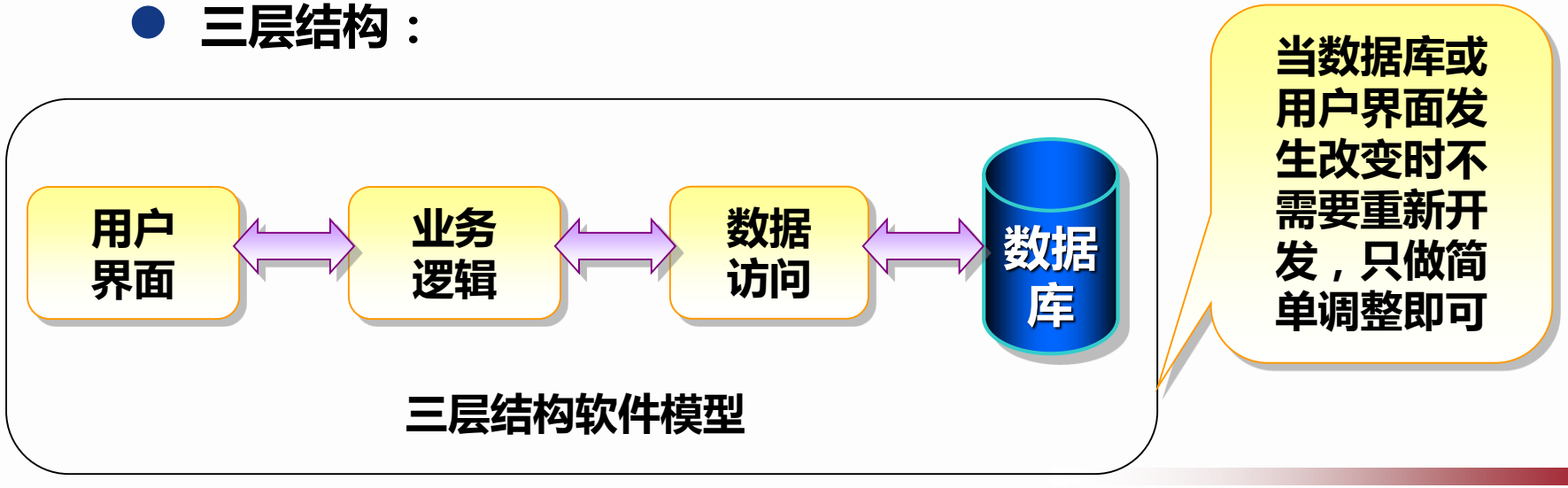
三、三层架构

为什么要使用三层架构

- 两层结构：

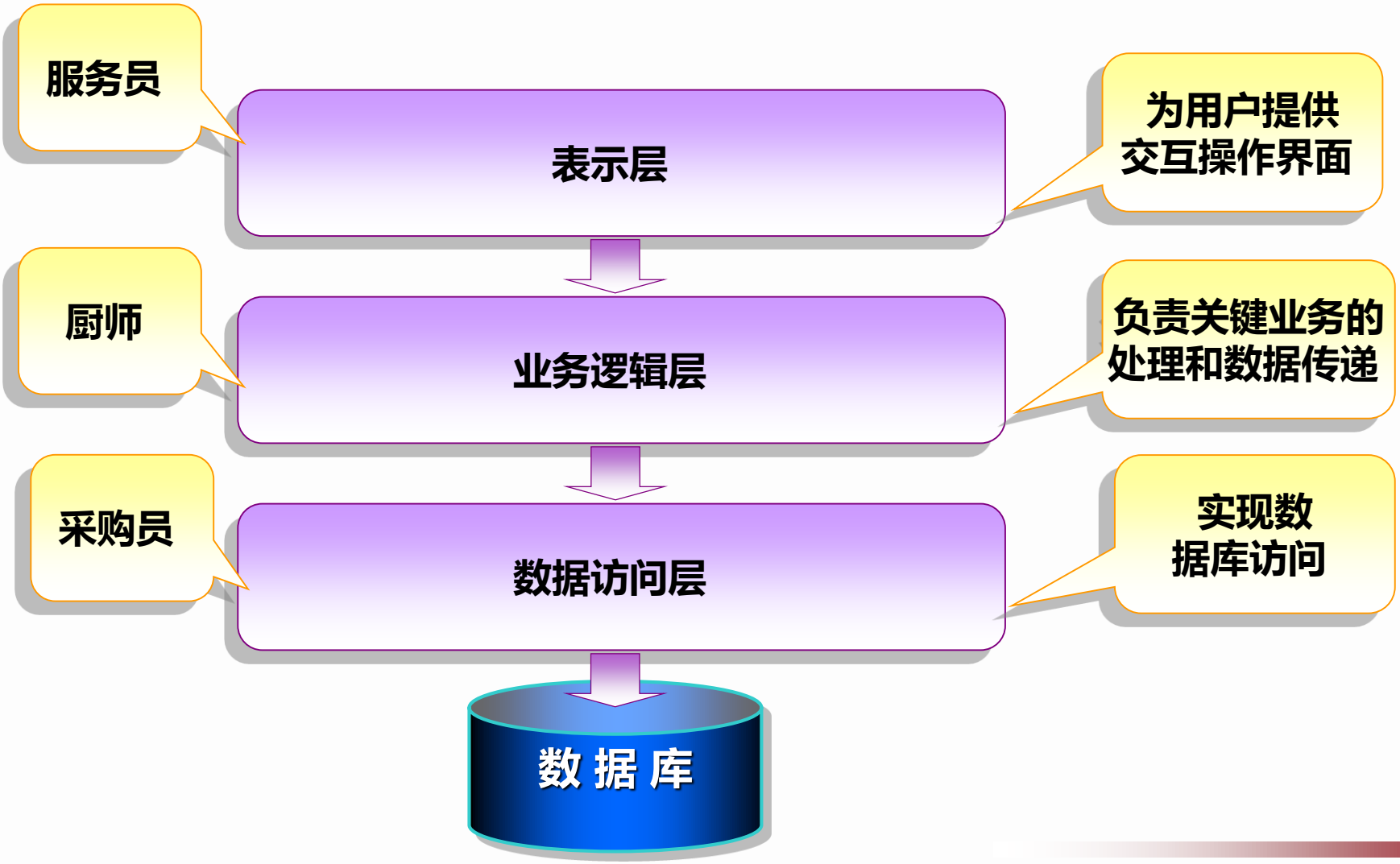


- 三层结构：



三、三层架构

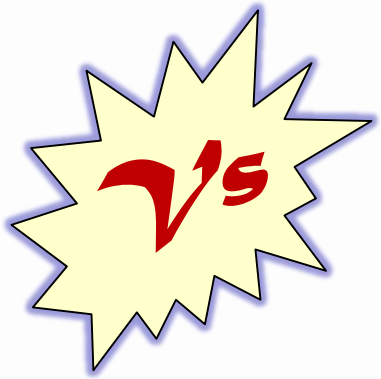
为什么要使用三层架构



三、三层架构

为什么要使用三层架构

- 优点：
  - 开发人员可以只关注整个结构中的其中某一层，分工实现；
  - 可以很容易的用新的实现来替换原有实现；
  - 可以降低层与层之间的依赖；
  - 有利于标准化；
  - 利于各层逻辑的复用。
  - 代码的可读性和功能的扩展性有着很好的提高
  - 各层相互隔离，安全性高
  - 减少了由于客户端被破解而给数据库带来损失的风险



- 两层架构缺点：
  - 难以适应需求变化
  - 不易维护
  - 安全性差
  - 客户端参与运算，体积庞大
  - 客户端配置要求较高
  - 通信量大
  - 可移植性差

三、三层架构

为什么要使用三层架构

- 缺点：
  - 降低了系统的性能。如果不采用分层式结构，很多业务可以直接造访数据库，以此获取相应的数据，如今却必须通过中间层来完成。
  - 有时会导致级联的修改。这种修改尤其体现在自上而下的方向。如果在表示层中需要增加一个功能，为保证其设计符合分层式结构，可能需要在相应的业务逻辑层和数据访问层中都增加相应的代码。

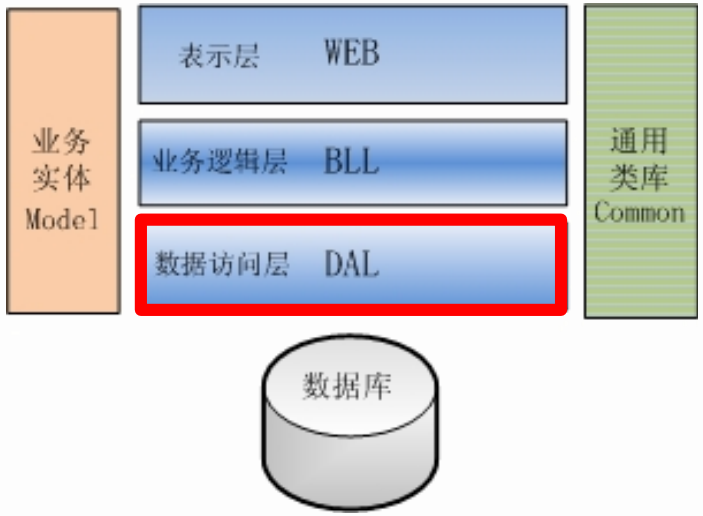
因此，在设计和开发一个应用系统时，需要考虑系统自身特点，选择合适的架构和开发方式

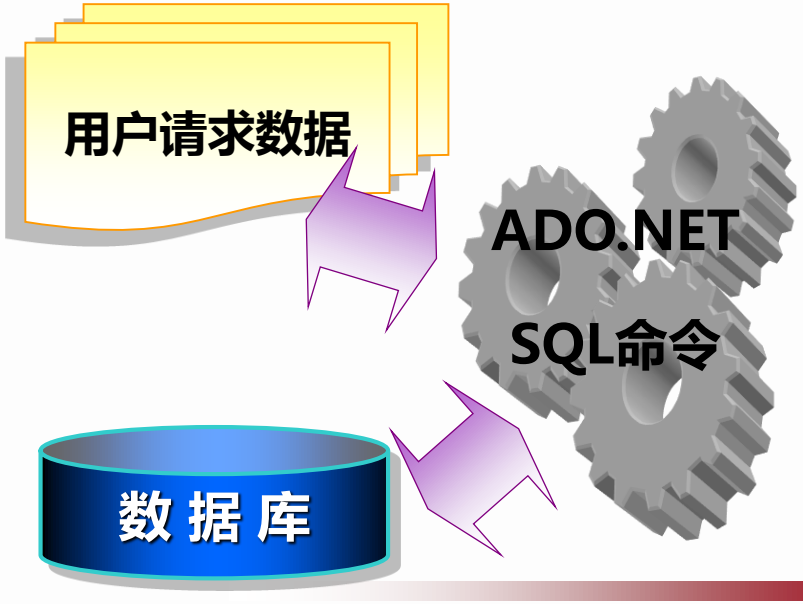
目录

四、开发实例

数据访问层（DAL）

- **功能：**
  - 主要提供数据存储及查询功能，并需承担部分数据验证的功能。对数 据库操作的代码都写在这里，例如：执行SQL语句、执行存储过程的代码等都写在这里面，为用户提供应用系统的使用界面及功能。
- **实现：**
  - 在Web.Config配置文件中定义连接字符串以访问数据库。建立数据集（xsd文件），定义SQL添加、查询、更改、删除等语句，封装成方法供上层调用。





四、开发实例

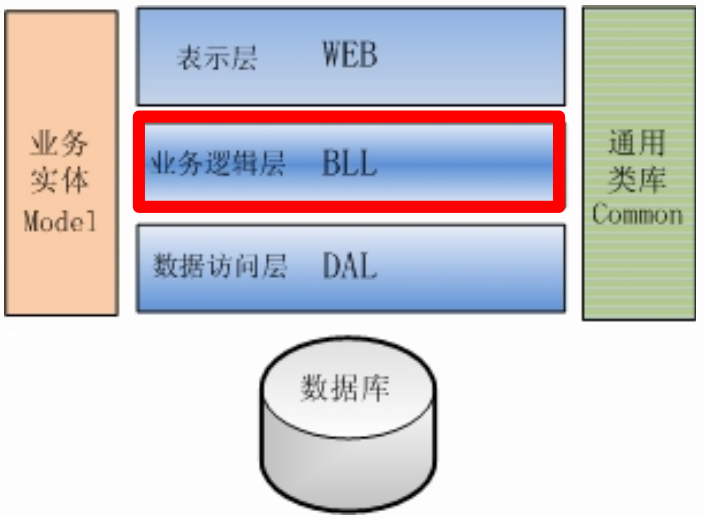
业务逻辑层（BLL）

- **功能：**
  - 提供所有与数据库的操作。

    包括：从数据库返回数据集，

    向数据库更新数据，

    承担部分数据验证的功能。



- **实现：**
  - 业务逻辑功能的实现按照操作对象归类，全部放在.CS类文件中。按照需求分析报告和软件设计报告实现系统流程、功能。

是表示层与数据访问层之间的桥梁，负责数据处理、传递

四、开发实例

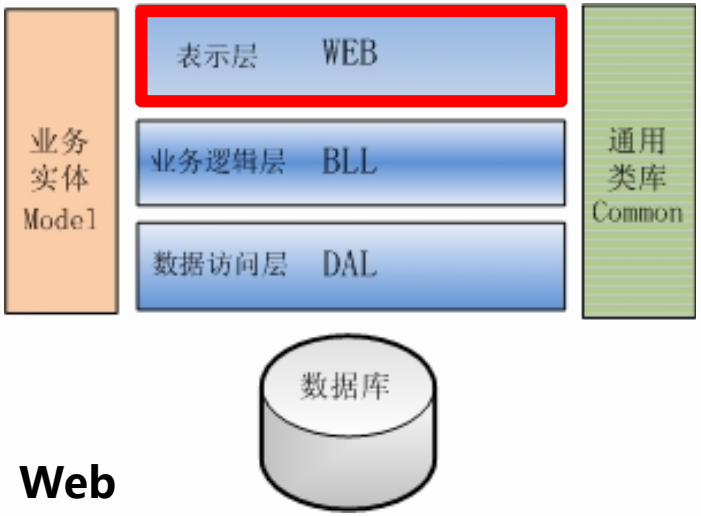
用户界面层（UI）

- 功能：
  - 为用户提供应用系统的使用界面
  - 提供相关功能的入口
- 实现
  - 界面设计部分：

    使用母版页、服务器控件、用户控件、Web页及css样式表来控制及实现。
  - 功能部分：

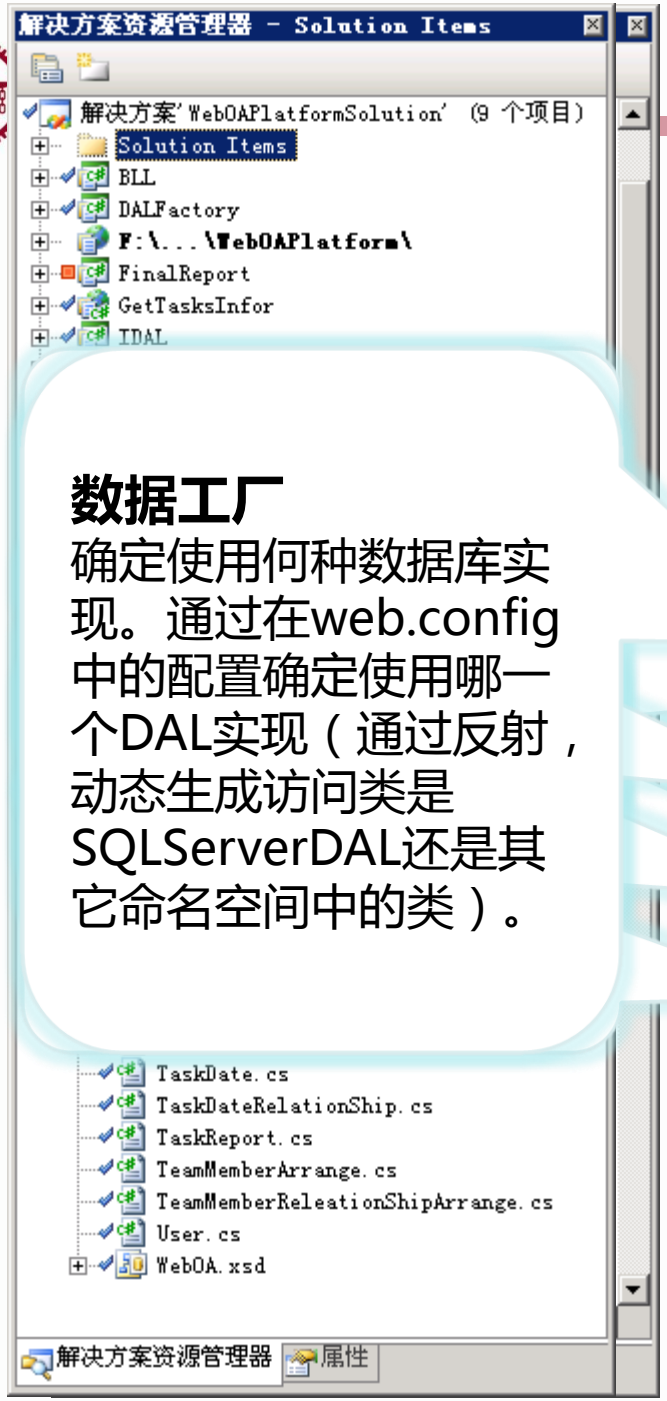
    服务器控件：实现模板的公共功能；

    用户控件：实现一些通用的构件



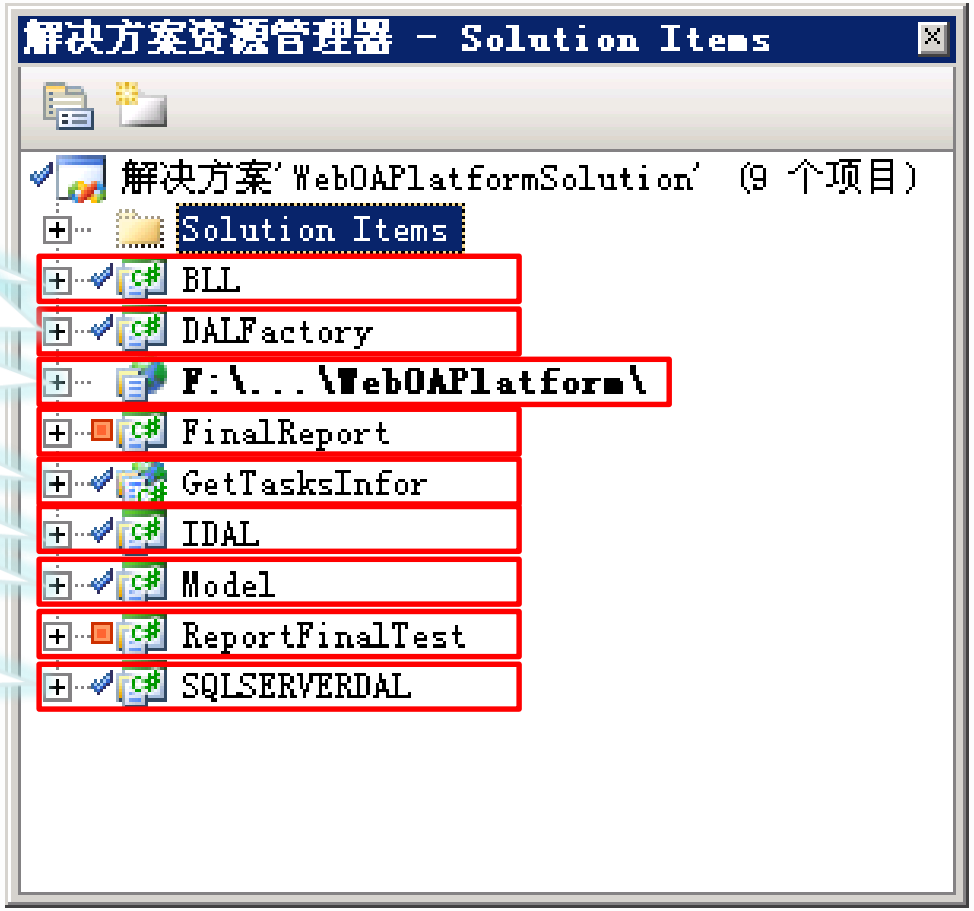
为用户提供一种交互式操作界面

四、开发实例

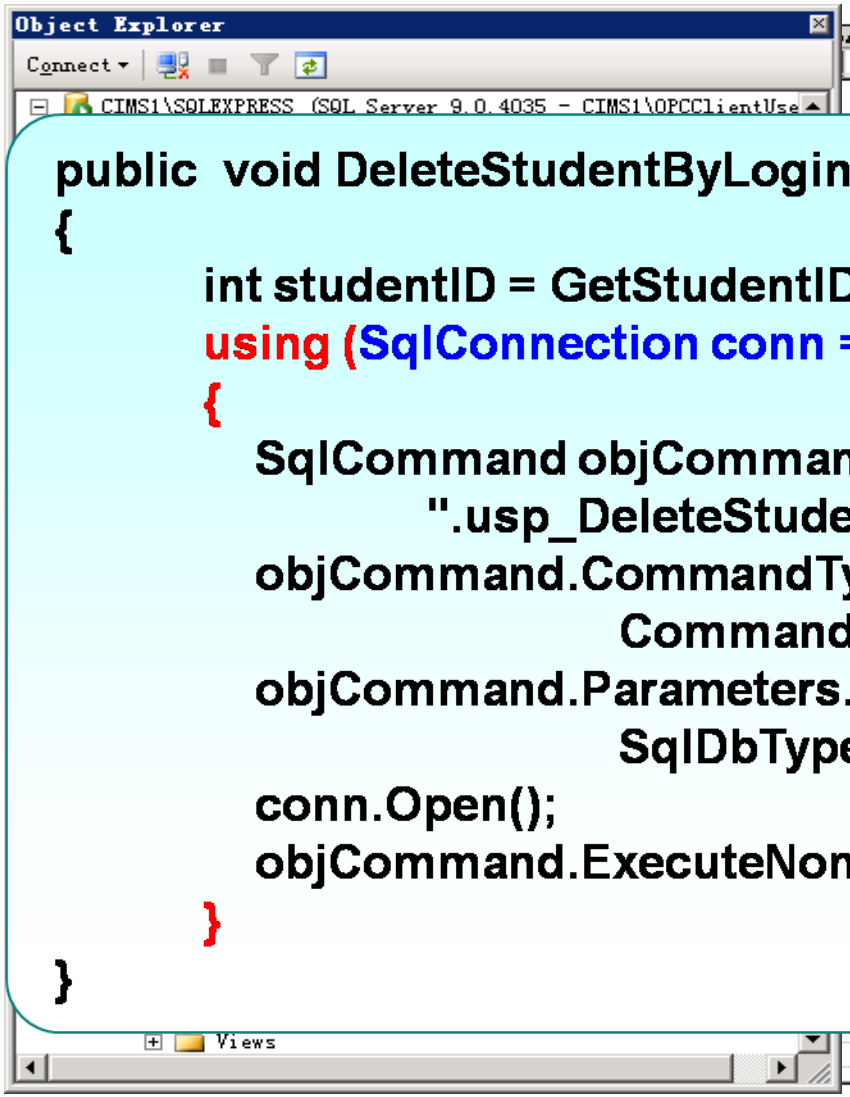


数据工厂

确定使用何种数据库实现。通过在web.config中的配置确定使用哪一个DAL实现（通过反射，动态生成访问类是SQLServerDAL还是其它命名空间中的类）。



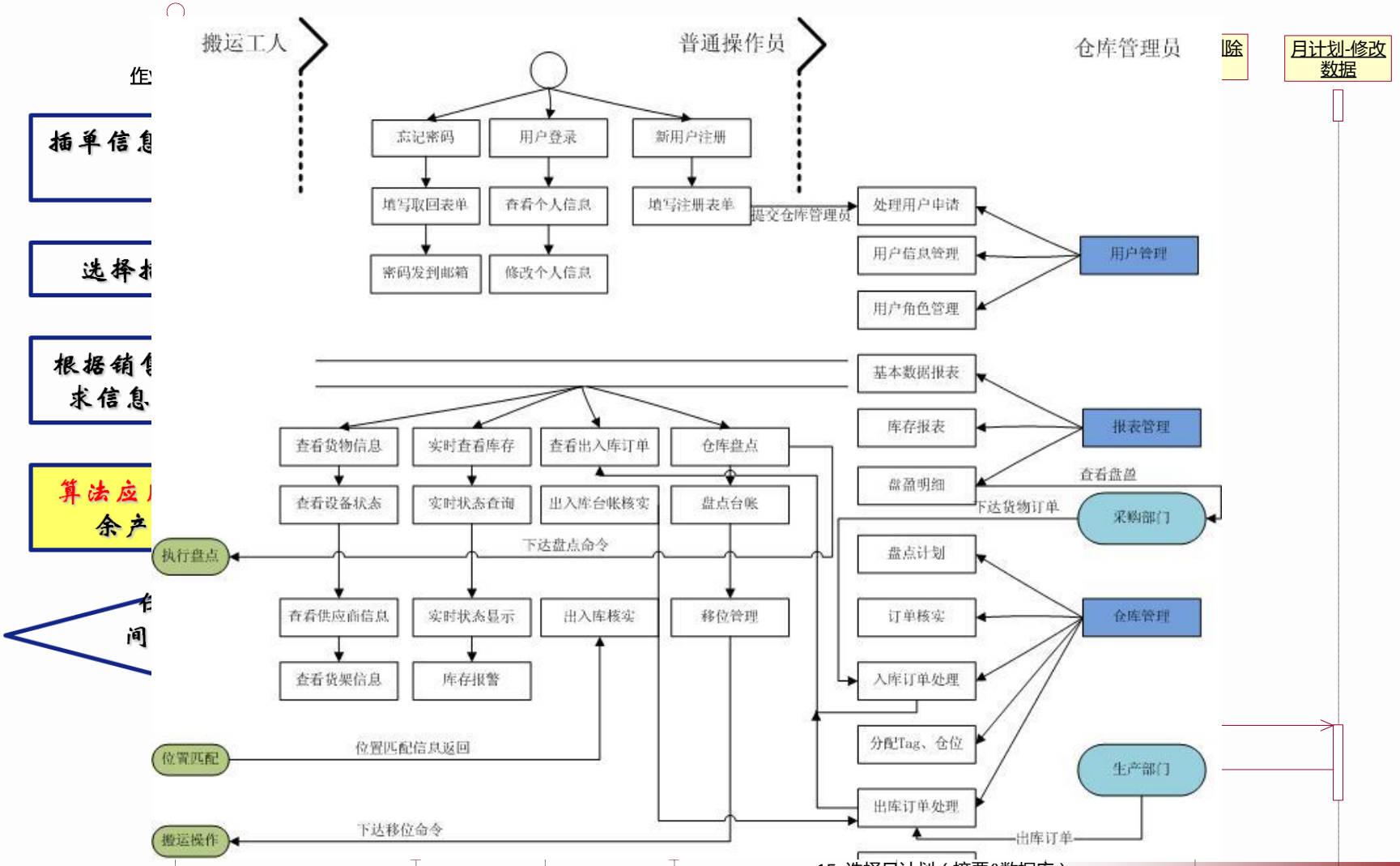
开发实例——数据库与访问层




```
public  void DeleteStudentByLoginID (string loginID)
{
        int studentID = GetStudentIDByLoginID(loginID);
        using (SqlConnection conn = new SqlConnection(connString))
        {
           SqlCommand objCommand = new SqlCommand(dboOwner +
                   ".usp_DeleteStudent", conn);
           objCommand.CommandType =
                           CommandType.StoredProcedure;
           objCommand.Parameters.Add("@StudentID",
                           SqlDbType.Int).Value = studentID;
           conn.Open();
           objCommand.ExecuteNonQuery();
        }
}
```

四、开发实例

开发实例——业务逻辑层

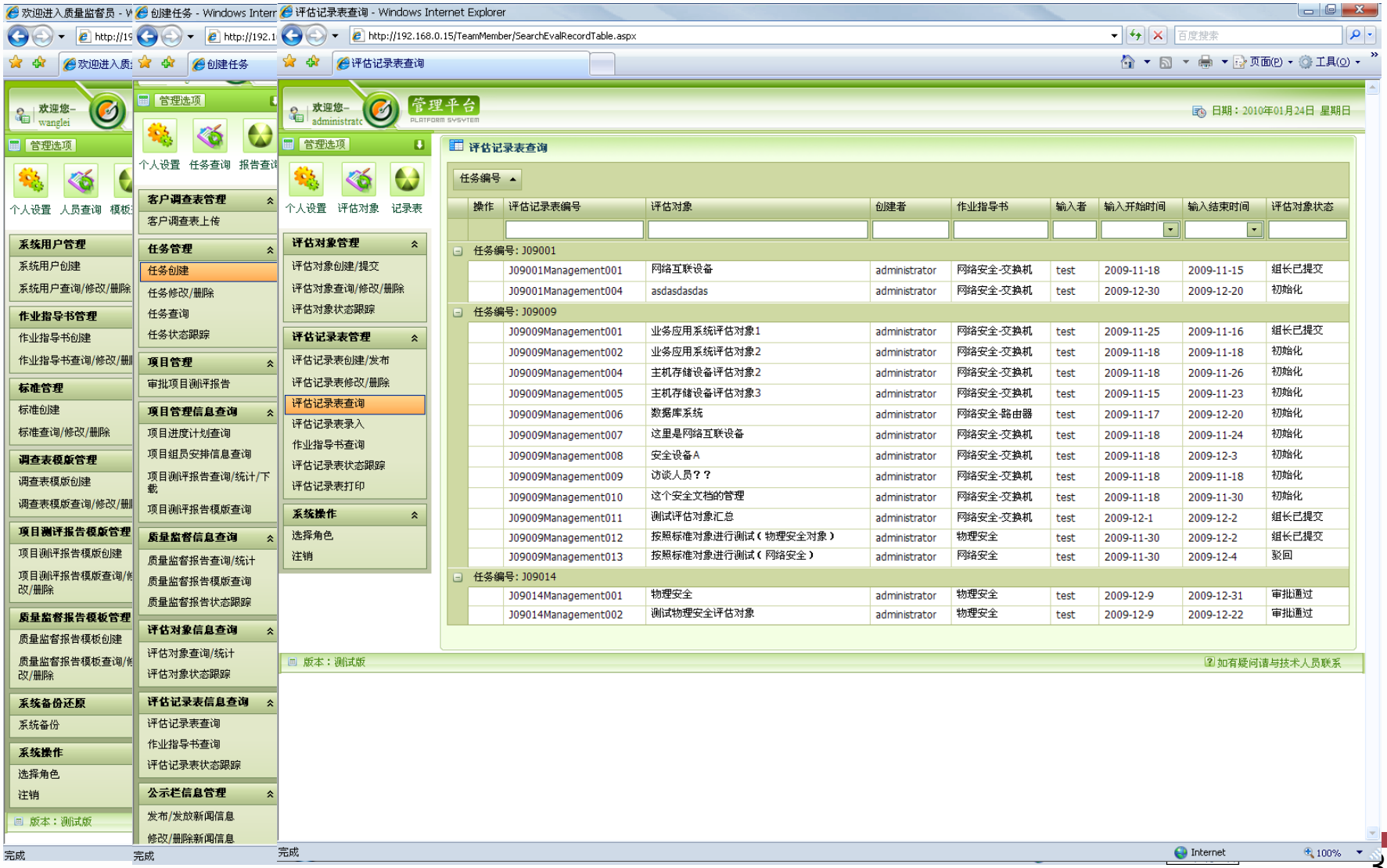


算法模型图

部分图

四、开发实例

开发实例——用户界面层



目录

五、总结展望

小结

- 三层架构将系统的整个业务应用划分为用户界面层 – 业务逻辑层 – 数据访问层。这样有利于系统的开发、维护、部署和扩展
  - 便于开发：每层做些什么其它层是完全看不到的，因此更改、更新某层，都不再需要重新编译或者更改全部的层，开发人员可以只关注整个结构中的其中某一层，分工实现，并行开发
  - 便于维护：降低层与层之间的依赖，结构灵活而且性能更佳
  - 便于部署：每一层都可以在仅仅更改很少量的代码后，就能放到物理上不同的服务器上使用
  - 便于扩展：可以很容易的用新的实现来替换原有实现。如数据访问代码和业务逻辑层分离，当数据库服务器更改后，只需要更改数据访问的代码，因为业务逻辑层是不变的，因此不需要更改或者重新编译业务逻辑层。

谢 谢！

Q & A