

NEW PLAN

软考高级架构师

• — 段 新 征 程 •





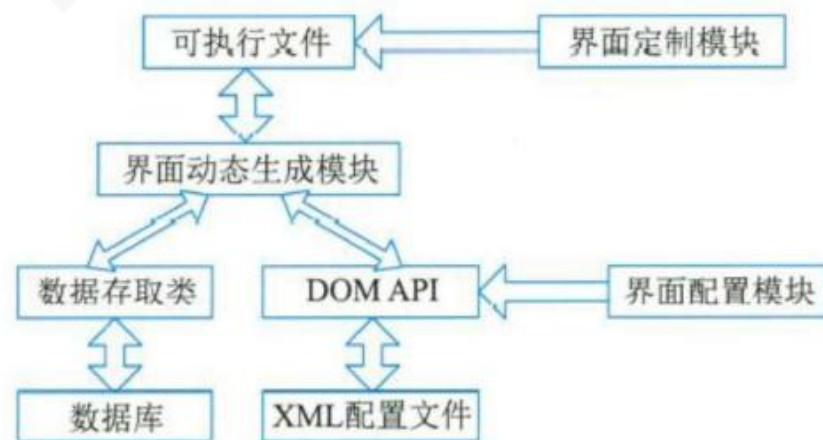
层次式架构设计





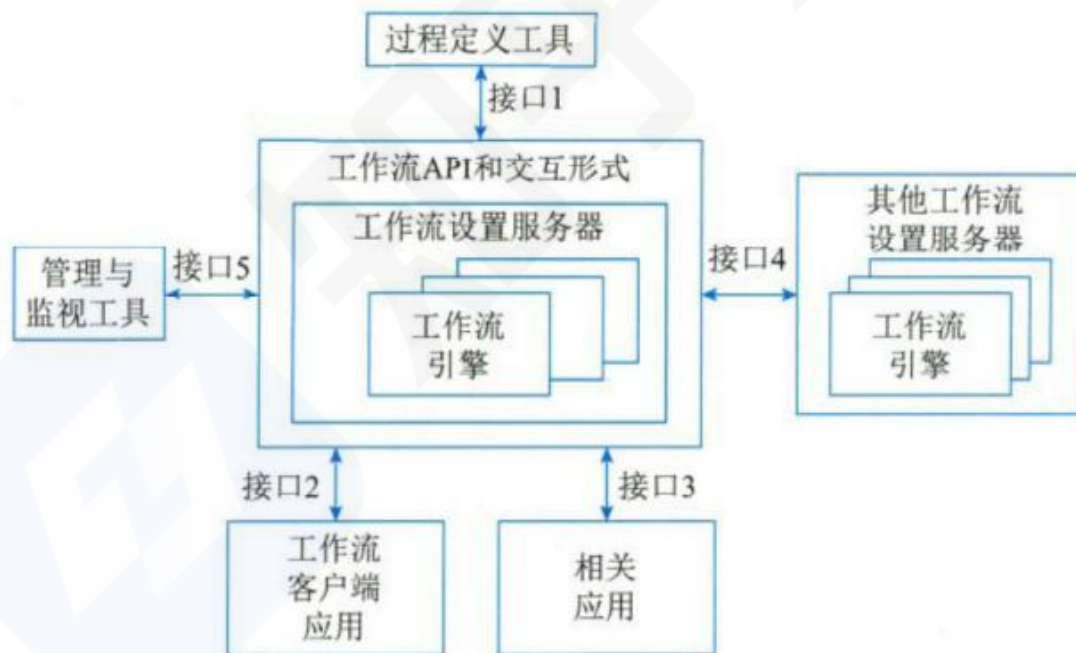
表现层框架设计

- 软件层次式体系结构是**最通用**的架构，也被叫作**N层架构模式**。大部分的应用会分成**表现层(或称为展示层)**、**中间层(或称为业务层)**、**数据访问层(或称为持久层)**和**数据层**。
- 一般会使用XML来设计表现层，现在更多的是使用动态拖拽生成表现层。
- UIP提供了一个**扩展的框架**，用于**简化用户界面与商业逻辑代码的分离的方法**，可以用它来写复杂的用户界面导航和工作流处理，并且它能够复用在不同的场景、并可以随着应用的增加而进行扩展。
- 使用UIP框架的应用程序把表现层分为了以下几层。
 - User Interface Components（用户界面组件）：**这个组件就是原来的表现层**，用户看到的和进行交互都是这个组件，它负责获取用户的数据并且返回结果。
 - User Interface Process Components（用户界面过程组件）：这个组件用于协调用户界面的各部分，使其配合后台的活动，例如导航和工作流控制，以及状态和视图的管理。**用户看不到这一组件，但是这些组件为User Interface Components提供了重要的支持功能。**
- 表现层动态生成设计：基于XML的界面管理技术可实现灵活的**界面配置(静态)**、**界面动态生成**和**界面定制(动态)**。其思路是用XML生成配置文件及界面所需的元数据，按不同需求生成界面元素及软件界面。



中层框架设计

- 组件设计：业务逻辑组件分为**接口和实现**类两个部分。接口用于定义业务逻辑组件，定义业务逻辑组件必须实现的方法是整个系统运行的核心。增加业务逻辑组件的接口，是为了提供更好的解耦，控制器无须与具体的业务逻辑组件耦合，而是面向接口编程。
- workflow设计：业务流程的全部或部分自动化，在此过程中，文档、信息或任务按照一定的过程规则流转，实现组织成员间的协调工作以达到业务的整体目标。它解决的主要问题是：使在多个参与者之间按照某种预定义的规则传递文档、信息或任务的过程自动进行，从而实现某个预期的业务目标，或者是促使此目标的实现。





中层框架设计

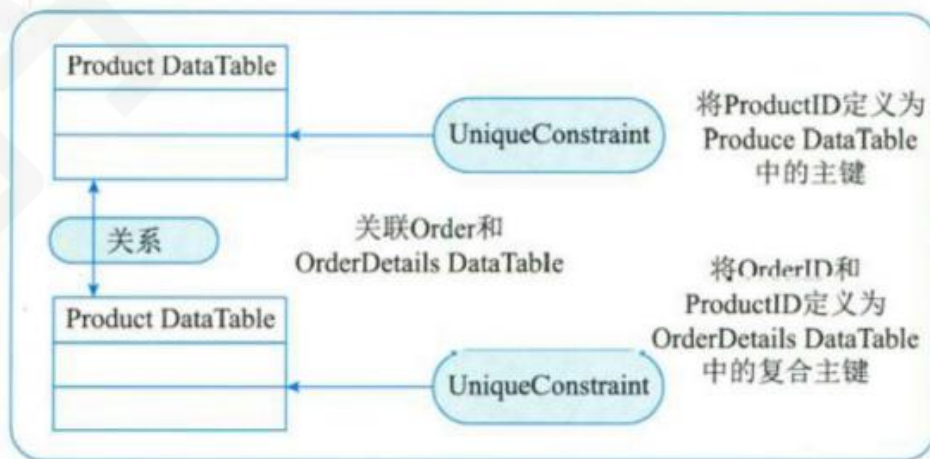
- interface 1: **过程定义导入/导出接口**。这个接口的特点是：转换格式和API调用，从而支持过程定义信息间的互相转换。
- interface 2: **客户端应用程序接口**。通过这个接口工作流机可以与任务表处理器交互，代表用户资源来组织任务。然后由任务表处理器负责，从任务表中选择、推进任务项。由任务表处理器或者终端用户来控制应用工具的活动。
- interface 3: **应用程序调用接口**。允许工作流机直接激活一个应用工具，来执行一个活动。典型的是调用以后台服务为主的应用程序，没有用户接口。当执行活动要用到的工具，需要与终端用户交互，通常是使用客户端应用程序接口来调用那个工具，这样可以为用户安排任务时间表提供更多的灵活性。
- interface 4: **工作流机协作接口**。其目标是定义相关标准，以使不同开发商的工作流系统产品相互间能够进行无缝的任务项传递。
- interface 5: **管理和监视接口**。提供的功能包括用户管理、角色管理、审查管理、资源控制、过程管理和过程状态处理器等。
- 用工作流的思想组织业务逻辑，优点是：将应用逻辑与过程逻辑分离，在不修改具体功能的情况下，通过修改过程模型改变系统功能，完成对生产经营部分过程或全过程的集成管理，可有效地把人、信息和应用工具合理地组织在一起，发挥系统的最大效能。



中层框架设计

- 实体设计：业务逻辑层**实体提供对业务数据及相关功能(在某些设计中)的状态编程访问**。业务逻辑层实体可以使用**具有复杂架构的数据**来构建，这种数据通常来自数据库中的多个相关表。
- 在应用程序中表示业务逻辑层实体的方法有很多(从以数据为中心的模型到更加面向对象的表示法)，如XML、通用DataSet、有类型的DataSet等。
- 如下左图是业务实体用XML表示，右图所示为用于Order业务逻辑层实体的通用DataSet对象。此DataSet对象具有两个Data Table 对象，分别保存订单信息和订单详细信息。每个DataTable具有一个对应的UniqueConstraint对象，用于标识表中的主键。此外，该DataSet 还有一个Relation对象，用于将订单详细信息与订单相关联。

```
<?xml version="1.0"?>
<Product xmlns="urn:aUniqueNamespace">
  <ProductID>1</ProductID>
  <ProductName>Chai</ProductName>
  <QuantityPerUnit>10 boxes x 20 bags</QuantityPerUnit>
  <UnitPrice>18.00</UnitPrice>
  <UnitsInStock>39</UnitsInStock>
  <UnitsOnOrder>0</UnitsOnOrder>
  <ReorderLevel>10</ReorderLevel>
</Product>
```





中层框架设计

- 业务框架位于**系统架构的中间层**，是实现系统功能的核心组件。采用容器的形式，便于系统功能的开发、代码重用和管理。下图便是在**吸收了SOA 思想之后的一个三层体系结构的简图**。
- 业务层采用**业务容器**的方式存在于整个系统当中，采用此方式可以大大**降低业务层和相邻各层的耦合**，表示层代码只需要将业务参数传递给业务容器，而不需要业务层多余的干预。如此一来，可以有效地防止业务层代码渗透到表示层。
- 在业务容器中，业务逻辑是按照**Domain Model—Service—Control**思想来实现的。
 - Domain Model：是**领域层业务对象**，它仅仅包含**业务相关的属性**。
 - Service：是**业务过程实现的组成部分**，是应用程序的不同功能单元，通过在这些服务之间定义良好的接口和契约联系起来。
 - Control：**服务控制器**，是服务之间的纽带，不同服务之间的切换就是通过它来实现的。

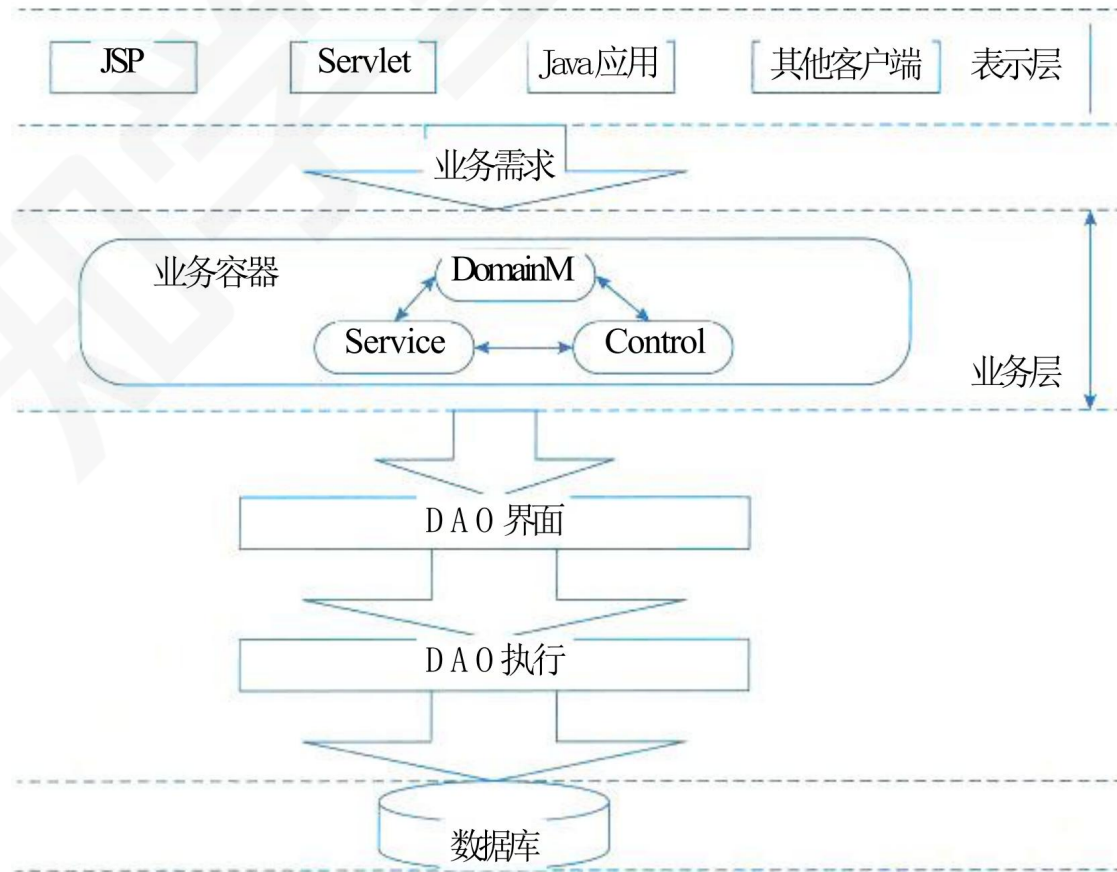


图13-10业务框架在整个系统架构中的位置



数据访问层设计

5种数据访问模式：

- **在线访问**：会占用一个数据库连接，读取数据，每个数据库操作都会通过这个连接不断地与后台的数据源进行交互。比如使用pl/sql或者navicat连接数据库
- **Data Access Object**：是标准J2EE设计模式之一，开发人员常常用这种模式将底层数据访问操作与高层业务逻辑分离开。
- **Data Transfer Object**：是经典EJB设计模式之一。DTO本身是这样一组对象或是数据的容器，它需要跨不同的进程或是网络的边界来传输数据。这类对象本身应该不包含具体的业务逻辑，并且通常这些对象内部只能进行一些诸如内部一致性检查和基本验证之类的方法，而且这些方法最好不要再调用其他的对象行为。
- **离线数据模式**是**以数据为中心**，数据从数据源获取之后，将按照某种预定义的结构存放在系统中，成为应用的中心。离线，对数据的各种操作独立于各种与后台数据源之间的连接或是事务
- **对象/关系映射**(Object/Relation Mapping, O/R Mapping)：大多数应用中的数据都是依据关系模型存储在关系型数据库中；而很多应用程序中的数据在开发或是运行时则是以对象的形式组织起来的。那么，对象/关系映射就提供了这样一种工具或是平台，能够帮助将**应用程序中的数据转换成关系型数据库中的记录**；或是**将关系数据库中的记录转换成应用程序中代码便于操作的对象**



数据访问层设计

- **工厂模式**在数据库访问层的应用：首先**定义一个操纵数据库的接口DataAccess**，然后根据**数据库的不同，由类工厂决定实例化哪个类**。因为DataAccess的具体实现类有一些共同的方法，所以先从DataAccess实现一个抽象AbstractDataAccess类，包含一些公用方法。然后，分别为SQL Server、Oracle和MySQL数据库编写三个数据访问的具体实现类。现在已经完成了所要的功能，下面需要创建一个Factory类，来实现自动数据库切换的管理。这个类很简单，主要的功能就是根据数据库类型，返回适当的数据库操纵类。
- **事务处理设计**：JavaBean中使用**JDBC**方式进行事务处理：在JDBC中，打开一个连接对象Connection时，默认是auto-commit模式，每个SQL语句都被当作一个事务，即每次执行一个语句，都会自动地得到事务确认。为了能将多个SQL语句组合成一个事务，要将auto-commit模式屏蔽掉。在auto-commit模式屏蔽掉之后，如果不调用commit()方法，SQL语句不会得到事务确认。在最近一次commit()方法调用之后的所有SQL会在方法commit()调用时得到确认。
- **连接对象管理设计（数据库连接池）**：通过**资源池**解决资源频繁分配、释放所造成的问题。有了这个连接池，下面就可以提供一套自定义的分配、释放策略。当客户请求数据库连接时，首先看连接池中是否有未分配出去的连接。如果存在空闲连接则把连接分配给客户，并标记该连接为已分配。若连接池中没空闲连接，就在已经分配出去的连接中，寻找一个合适的连接给客户，此时该连接在多个客户间复用。当客户释放数据库连接时，可以根据该连接是否被复用，进行不同的处理。如果连接没有使用者，就放入到连接池中，而不是被关闭。



数据架构规划与设计

XML文档分为两类：

- 一类是**以数据为中心**的文档，这种文档通常用于存储和传输结构化数据，数据元素的顺序通常不是关键因素，而更关注数据本身。一个典型的例子是使用XML来表示电子商务产品目录的数据。左图中，数据元素（如产品ID、名称和价格）都是规则的，并且存储了有关产品的结构化信息。**顺序不重要，只要能够访问和解析这些数据。**
- 另一类是**以文档为中心**的文档，这种文档用于发布描述性信息，内容可能比较零散，元素之间的顺序可能很重要。一个典型的例子是使用XML来表示一篇新闻文章，右图中，元素的顺序很重要，因为它们定义了文章的结构，如标题、作者和内容。**这种XML文档用于在网页上发布描述性信息，文章的内容和结构通常由文档的作者或编辑精心安排。**

```
<products>
  <product>
    <id>101</id>
    <name>Laptop</name>
    <price>799.99</price>
  </product>
  <product>
    <id>102</id>
    <name>Smartphone</name>
    <price>299.99</price>
  </product>
</products>
```

```
<article>
  <title>Breaking News</title>
  <author>John Doe</author>
  <content>
    <paragraph>This is a breaking news story...</paragraph>
    <paragraph>More details to follow...</paragraph>
  </content>
</article>
```



数据架构规划与设计

经提出的XML文档的存储方式有两种：

- 基于文件的存储方式。基于文件的存储方式是指将XML文档按其原始文本形式存储，主要存储技术包括操作系统文件库、通用文档管理系统和传统数据库的列。这种存储方式需维护某种类型的附加索引，以建立文件之间的层次结构。基于文件的存储方式的特点：无法获取XML文档中的结构化数据；通过附加索引可以定位具有某些关键字的XML文档，一旦关键字不确定，将很难定位；查询时，只能以原始文档的形式返回，即不能获取文档内部信息；文件管理存在容量大、管理难的缺点。
- 数据库存储方式。数据库在数据管理方面具有管理方便、存储占用空间小、检索速度快、修改效率高和安全性好等优点。一种比较自然的想法是采用数据库对XML文档进行存取和操作，这样可以利用相对成熟的数据库技术处理XML文档内部的数据。数据库存储方式的特点：能够管理结构化和半结构化数据；具有管理和控制整个文档集合本身的能力；可以对文档内部的数据进行操作；具有数据库技术的特性，如多用户、并发控制和一致性约束等；管理方便，易于操作。



物联网层次架构设计

物联网可以分为三个层次，**底层是用来感知数据的感知层**，即利用传感器、二维码、RFID等设备随时随地获取物体的信息。**第二层是数据传输处理的网络层**，即通过各种传感网络与互联网的融合，将对象当前的信息实时准确地传递出去。**第三层则是与行业需求结合的应用层**，即通过智能计算、云计算等将对象进行智能化控制。

- 感知层：用于识别物体、采集信息。感知层包括二维码标签和识读者、RFID标签和读写器、摄像头、GPS、传感器、M2M终端、传感器网关等，主要功能是识别对象、采集信息，与人体结构中皮肤和五官的作用类似。**感知层解决的是人类世界和物理世界的获取数据问题。**
- 网络层：用于传递信息和处理信息。网络层包括通信网与互联网的融合网络、网络管理中心、信息中心和智能处理中心等。网络层将感知层获取的信息进行传递和处理，类似于人体结构中的神经中枢和大脑。**网络层解决的是传输和预处理感知层所获得数据的问题。**
- 应用层：实现广泛智能化。应用层是物联网与行业专业技术的深度融合，结合行业需求实现行业智能化，这类似于人们的社会分工。物联网应用层利用经过分析处理的感知数据，为用户提供丰富的特定服务。**应用层解决的是信息处理和人机交互的问题。**



层次是架构案例分析

电子商务网站(网上商店PetShop)

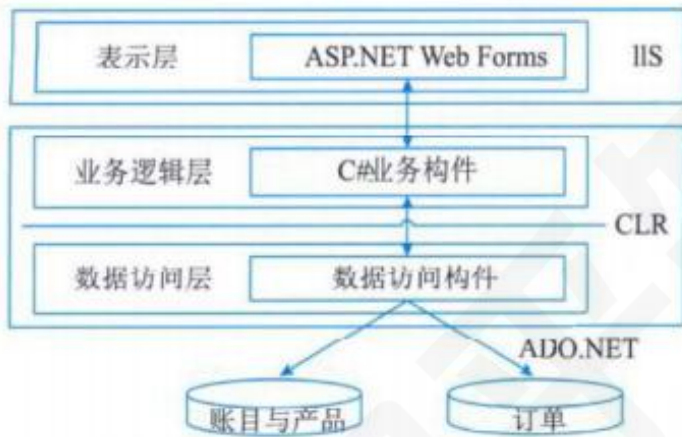


图 13-15 Net 中标准的 BS 分层式结构

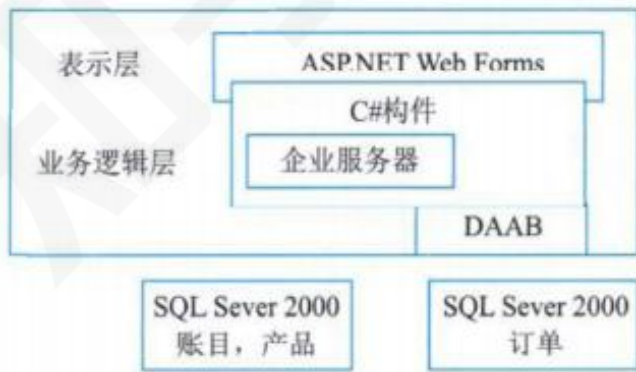


图 13-16 PetShop 2.0 的体系架构

从图13-16中可以看到，并没有明显的数据库访问层设计。这样的设计虽然提高了数据库访问的性能，但也同时导致了业务逻辑层与数据库访问的职责混乱。

PetShop 3.0纠正了此前层次不明的问题，将数据库访问逻辑作为单独的一层独立出来。



层次是架构案例分析

PetShop 4.0基本上延续了3.0的结构，但在性能上作了一定的改进，引入了缓存和异步处理机制，同时又充分利用了ASP.Net 2.0的新功能MemberShip。可以看到，在数据访问层中，完全采用了“面向接口编程”思想。抽象出来的IDAL模块，脱离了与具体数据库的依赖，从而使得整个数据访问层有利于数据库迁移。DALFactory模块专门管理DAL对象的创建，便于业务逻辑层访问。SQLServerDAL 和OracleDAL模块均实现IDAL模块的接口，其中包含的逻辑就是对数据库的Select、Insert、Update和Delete操作。因为数据库类型的不同，对数据库的操作也有所不同，代码也会因此有所区别。

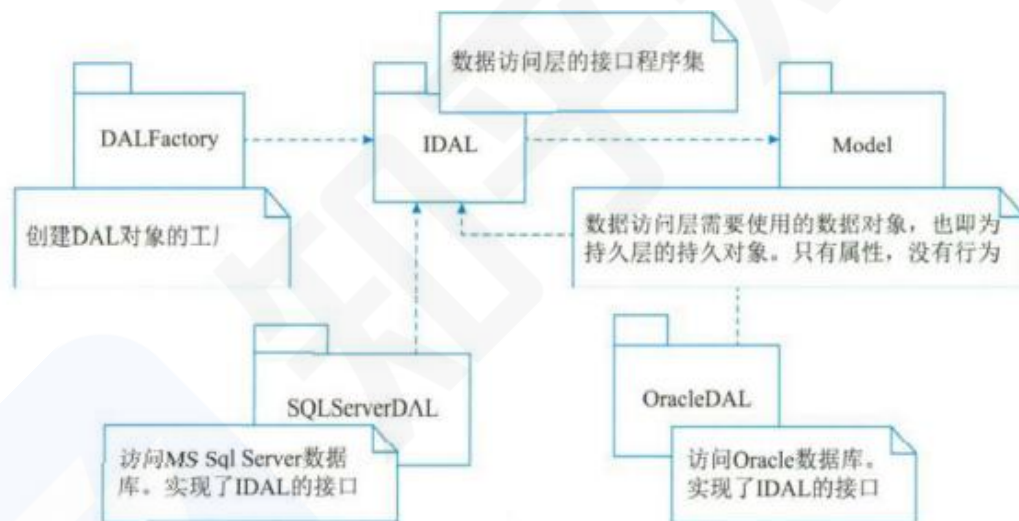


图 13-19 数据访问层的模块结构图



层次是架构案例分析

此外，抽象出来的IDAL模块，除了解除了向下的依赖之外，对于其上的业务逻辑层同样仅存在弱依赖关系，如图13-20所示。图13-20中，**BLL是业务逻辑层的核心模块**，它包含了整个系统的核心业务。在业务逻辑层中，不能直接访问数据库，而必须通过数据访问层。注意，图13-20中对数据访问业务的调用，是通过**接口模块IDAL来完成的**。既然与具体的数据访问逻辑无关，则层与层之间的关系就是松散耦合的。如果此时需要修改数据访问层的具体实现，只要不涉及IDAL的接口定义，那么业务逻辑层就不会受到任何影响。毕竟，具体实现的SQLServerDAL和OracleDAL根本就与业务逻辑层没有半点关系。

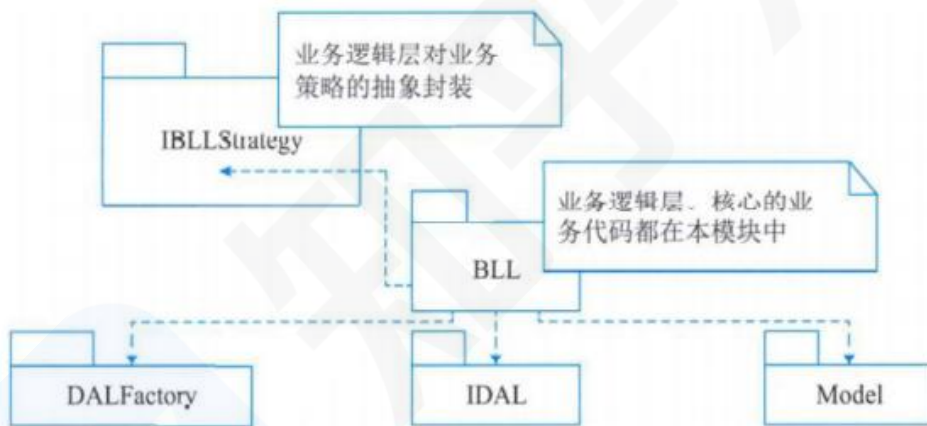


图 13-20 业务逻辑层的模块结构图

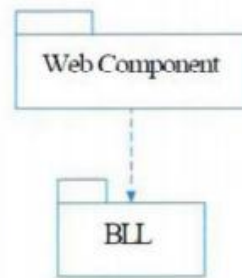


图13-21 表示层的模块结构



层次是架构案例分析

基于物联网架构的电子小票服务系统

采用感知层、网络层和应用层的3层物联网体系架构模型：



T H E E N D

功不唐捐，玉汝于成！

• 开 启 新 征 程 •

