应用对象组件的多层信息系统分析①

饶 浩

(韶关学院 信息管理系, 韶关 512005)

摘 要: 随着现代网络技术的快速发展, 信息系统的架构面临着安全性、扩展性、稳定性、性能等方面的新挑战. 提出信息系统的多层级体系结构, 面向对象组件的多层系统构建方式, 有利于系统的维护修改和功能扩充, 方便开发人员理解与掌握, 具有更好的稳定性、可扩展性.

关键词: 信息系统; 对象组件; 业务逻辑层

Analysis of Multi-tier Application Information System Based on Object Component

RAO Hao

(Department of Information Management, Shaoguan University, Shaoguan 512005, China)

Abstract: The development of network technology enables information systems to face security, scalability, stability, performance and other aspects of the new challenges. Multi-Tier application information system was proposed in this paper.The object component improves the performance of system. Multi-tier application information system was propitious to be maintained and understood and can provide flexible system architecture and adapt to the demand of enterprise informatization.

Key words: information system; object component; business logic layer

1 引言

早期业务系统结构模型, 技术方面以 ASP 或 PHP 为主, 系统采用两层级的混合式结构. 底层是数据库层, 系统的设计和编码采用单一技术, 所有对数据库的操作、对业务逻辑的处理、校验处理、动态或静态页面、访问安全等, 都嵌在数据业务逻辑层. 在早期互联网的发展时期, 系统用户少, 并发量小, 因此两层级的结构模式对于用户需求而言是绰绰有余的. 简单的 Web 页面制作也促进了小型软件系统的普及应用.

随着互联网的进一步普及, 系统业务需求量快速增长, 用户应用能力也不断增强. 两层级的系统模型显得笨拙固化. 对于包含数百条业务规则的系统, 数据和业务逻辑之间的强耦合导致系统模块独立性差. 每一条业务规则的修改和扩展, 都可能引发其它关联的错误, 容易导致异常的产生, 使程序不能正常运行. 系统维护和更新变得愈发困难, 扩展性极差. 由于业务的不断更新增长, 两层级的系统结构无法适应现实需求的变化. 由此, 基于 J2EE 技术或.net 技术的三层系统结构应运而生. 三层系统结构包含表示层、业务逻辑层、数据访问层. 三层系统结构的划分较为合理, 业务逻辑层负责与其它两层通信, 处理和协调所有数据流. 表示层和数据访问层之间不能直接访问. 减少了层级间的依赖性, 降低了系统开发和维护的复杂性, 能适应系统业务需求的扩展.

由于三层系统的开发过程主要围绕功能点进行, 界面和执行代码位于同一层, 执行不同功能的代码之间, 存在各种依赖关系, 多个业务封装在同一个功能模块内. 对于大中型信息系统, 数据和业务逻辑之间的耦合使得维护比较困难. 当需要改变某个业务规则, 或需要替换某个功能时, 将牵涉到模块内的其它不相关的功能代码, 不利于系统业务的扩展.

为了更好的提高模块独立性, 使得模块松耦合, 提

① 基金项目:2013 年教育部人文社会科学研究项目(13YJCZH144)
收稿时间:2014-07-29;收到修改稿时间:2014-10-16

高系统的可扩展性和可维护性, 对于新的系统模型的构建, 可应用对象组件的开发思想. 应用程序的功能由一些松耦合的组件单元构建而成, 对迅速变化的业务环境具有良好适应力. 业务功能之间通过具有统一定义的接口联系起来, 接口的定义采用中立方式, 独立于具体实施的编程语言、操作系统和硬件平台. 因此, 构建在不同系统中的业务可以以通用的方式进行交互.

2 系统架构分析

多层信息系统的思想, 是在表示层、业务逻辑层、数据访问层的三层结构基础上, 根据需要扩展出控制器层、持久化层等新层级, 每一层的注重点不同, 有利于更好的分解系统结构. 以下就几个关键层级的功能与结构进行分析.

2.1 控制器层

早期软件模型的系统业务和界面融合一起, 使得管理和维护十分艰难. 新系统模型结构分离出控制器层, 控制器是该层核心, 控制器连接界面和系统业务, 二者为松耦合, 松耦合后的系统对于集群部署后的维护和管理更加轻松. 控制器采用 MVC 实现, 即模型(Model)、视图(View)、控制(Controller)模式. 控制器的组成部分包括: 前端控制器、映射表工厂、动作处理工厂、动作处理部件、视图转发部件.

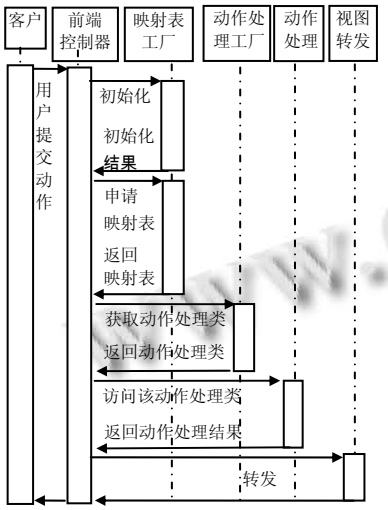

图 1　控制器层时序图

前端控制器处理所有从界面提交过来的相关请求, 并集中了控制逻辑, 避免逻辑的重复, 完成主要的请求处理操作, 同时也扮演应用程序控制器的角色. 系统需要一个集中的访问点来处理请求. 如果没有集中访问点, 多个请求的共用控制代码会重复出现, 降低模块化程度. 该层实现两个主要功能: 首先, 对于表示层的请求, 根据用户的配置, 定位需要的业务逻辑并执行, 这是操作管理; 其次, 对于业务逻辑的执行结果, 根据用户的配置, 定位对应的视图, 这是视图管理.

映射表工厂是一个操作和视图配置的解释和管理的工厂. 当从映射表工厂中得到该请求对应的处理动作关系表后, 即可向动作处理工厂获取该具体动作处理类, 然后交由该类处理, 动作处理工厂负责定位并获取所需要的操作. 最后把结果返回给用户并转发到对应的界面去. 控制器把系统用例和界面松耦合地连接起来, 可以更轻松地扩展和配置新的系统用例和界面, 并可以让多种用户界面共享同一个系统用例. 即使增加新的访问方式或新的用户界面, 也可以方便地整合并继续应用现有的系统用例模型.

2.2 业务逻辑层

业务逻辑层用于实现业务模型的特定功能. 业务代表模式封装对业务服务的访问, 抽象并隐藏业务服务层的实现细节, 例如对于服务远程调用(RMI)所需要的寻址等; 把底层的错误或异常转换为程序级别的错误信息, 便于用户理解; 当调用服务的时候发生错误或异常, 业务代表可以直接进行一定次数的重试; 对服务数据进行缓存, 提高运行效率. 以上这些对于客户端是透明的, 因此, 采用业务代表可以降低客户端和业务逻辑层之间的耦合.

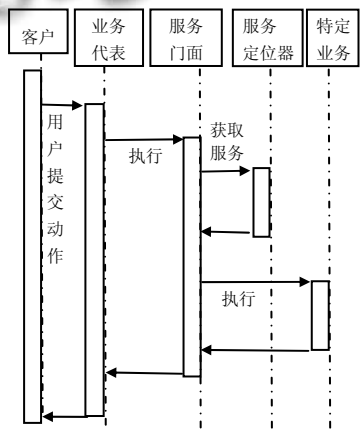

图 2　业务逻辑层时序图

服务门面的作用是控制客户端对业务服务的访问, 降低远程客户端和业务服务组件的交互所造成的网络

负载. 一般采用 Enterprice JavaBean 实现, 客户端对其进行远程调用, 在不同容器上相同的服务门面的集群可以使得网络负载均衡.

业务逻辑层使用服务定位器透明而统一地实现对业务服务或业务组件的寻址. 服务定位器能够隐藏寻址机制的实现细节, 封装这一机制对不同实现的依赖. 系统通过服务定位器实现重用, 降低代码的复杂性, 提供唯一的控制点, 提供对业务组件或服务的缓存机制, 改善系统性能. 服务定位器也采用单一模式实现, 因为通常一个系统中只采用一个服务定位器.

2.3 持久化层

持久化, 就是将对象保存到可以永久性保存的存储媒介中. 持久化层是在面向对象结构中一个专门负责对象持久化的类层次, 将数据使用者和数据实体相互关联. 持久化层使得对象的存贮对于程序设计者是透明的, 设计者可以专注于应用逻辑的开发, 不必考虑如何存贮对象, 也不用知道存贮在何种持久化机制中.

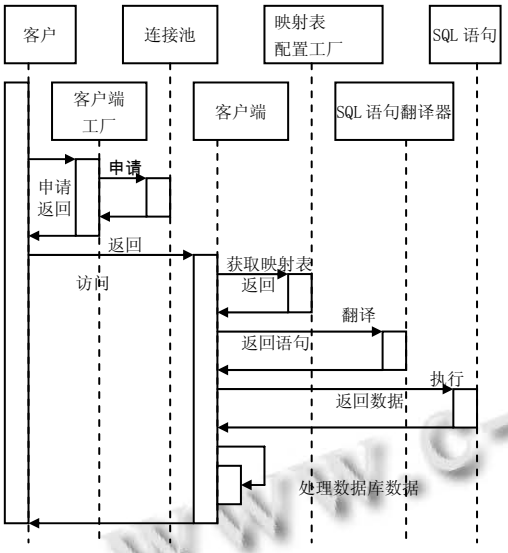

图 3 持久化层时序图

持久器属于持久化层, 系统底层采用的是关系型数据库, 持久器须实现对象-关系型数据映射, 对系统的对象模型和数据库可以存储的关系模型实现相互转化. 持久器包括客户端工厂、映射表配置工厂、SQL 语句翻译器等主要部件. 客户向客户端工厂发送消息, 获取一个客户端, 客户执行客户端的某个具体操作, 客户端向映射表配置工厂获取该操作对应映射配置, 根据预先配置好的映射表对业务对象进行操作, 利用 SQL 语句翻译器提供对用户配置的 SQL 语句和参数进行翻译的方法, 得到可供数据库直接执行的 SQL 语句.

持久化层把内部的业务逻辑和数据处理逻辑分离开来, 降低系统的耦合度, 提高系统的灵活性, 由于持久器所需要的配置都采用文件形式, 所以持久器使得系统具有良好的可移植性、可维护性和可扩展性.

3 系统实施

以常见的进销存信息管理系统为例,将系统简化为信息维护、进货管理、销售管理、报损管理等模块. 信息维护负责商品分类信息和供应商信息的维护; 进货管理负责商品的进货; 销售管理负责商品的销售; 报损管理负责商品的报损工作.

通过分析进销存系统的结构以及业务角色的划分, 可以将进存销系统扩展为六个层次结构. 每一层所专注的目的不同: 表示层以网页形式展示系统界面的相关内容, 收集用户的输入, 与用户对话, 处理系统用户之间及业务之间进行交互的各种程序逻辑, 向用户显示结果输出等. 界面可应用 HTML、JSP、ASP、PHP 实现; 控制器层用于连接界面与进存销系统用例, 提供连接关系配置功能, 该层可采用开源框架实现, 例如 Struts、Spring 等; 业务逻辑层建立进存销系统需要的所有业务用例, 集中对数据业务逻辑进行处理, 接收表示层发出的请求, 进行有效性验证, 应用业务规则调用数据访问层, 以获取响应表示层请求所需的信息. 采用配置文件的方式连接表示层和业务层, 使得扩展和配置简单轻松, 新的业务及界面可以采用“热插拔”的方式融入原系统; 持久化层是进存销系统用例对象数据持久器, 作用在于作用在于实现对象和关系的映射, 提供映射关系的配置和管理, 使得下层的用例对象持久化. 映射操作的处理逻辑置于此层, 把数据处理逻辑与业务对象相分离, 降低业务对象模型和关系模型耦合度; 连接池扮演中间件的角色, 用于解决系统频繁连接数据库造成的高负载. 将数据库连接作为中间件, 通过数据库连接池, 应用程序可以重用池中已有的连接, 减少用户申请与数据库建立或释放连接的开销, 从而提高数据库访问的并发量; 数据访问层采用 MySQL 数据库作支持, 对系统数据进行管理, 执行数据的查询、添加、修改、删除等功能.

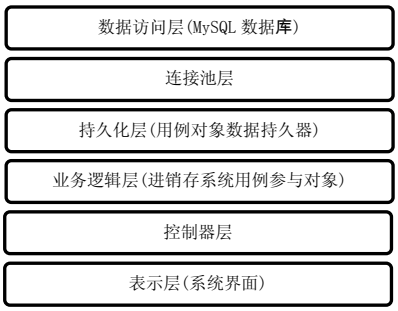

图 4 多层进销存系统结构

系统以实际的业务为中心进行单元划分, 当其中某一个业务需要更改, 只要接口没有改变, 可以直接替换. J2EE 技术中的 EJB 针对这方面提供了良好的支持, 每个业务都封装在一个 EJB 中, 所有的业务可以以“热插拔”的形式提供. 更改某个业务, 只需要将其“拔”下来, 把新的业务“插”回去即可. 这对于用户来说是完全透明的, 而且也使得业务之间松耦合, 松耦合后的进存销系统具备更好的稳定性和可扩展性.

4 结语

应用对象组件的多层级信息系统, 虽然需要管理更多的文件, 甚至更高的开发成本. 但通过细化层次、分解结构、应用对象组件, 降低了系统复杂度和模块的耦合度, 提高了业务的扩展能力和开发效率. 由此建立的信息系统, 能够更自如地应对业务规则的变化, 便于扩展和配置新的系统用例和界面, 从而拥有更好的软件可维护性.

参考文献

1 曾艳阳,康凤举,张建春.一种基于 SOA 的指控仿真系统的分层体系结构.系统仿真学报,2011,(8):1714–1718.
2 张志杰.基于分层结构的管理信息系统架构设计.计算机技术与发展,2010,(10):146–149.
3 钱碧伟,谢冬青,周再红,熊伟.OXVoD:一个基于分层结构的 P2P 视频点播系统.计算机工程与应用,2010,(7):203–207.
4 王伟,李建荣,罗四维.基于多层结构的短信息应用系统研究与实现.计算机工程与设计,2009,(10):2547–2550.
5 王文发,马燕,李红达.基于.NET 的四层结构及其在综合信息系统中的应用.计算机工程与设计,2009,(4):912–914.
6 林金山,林建兵,谢怀生,许振和.用 Java 设计的基于三层结构的答疑系统.计算机工程与设计,2008,(5):1308–1311.