

HiMall系统技术说明书

2017-02-25

**Himall系统技术说明书**

**2017年02月**

**Copyright © Hishop.com**

**目录**

[一、前言 5](#_Toc26231)

[二、名词解释 5](#_Toc2205)

[三、系统架构概况 6](#_Toc26731)

[（一）技术路线 6](#_Toc878)

[1、 采用完全面向对象设计思想。 6](#_Toc7059)

[2、 采用MVC的开发模式。 7](#_Toc3656)

[3、 采用ORM框架实现数据的持久化。 7](#_Toc24743)

[4、 采用插件机制实现相同业务的可热插拔 7](#_Toc5700)

[5、 采用策略机制实现相同业务的可配置变更 8](#_Toc10931)

[6、 UI基于主流的Bootstrap框架自主设计 8](#_Toc23358)

[7、 基于JQuery插件式开发 8](#_Toc5890)

[（二）系统架构图 8](#_Toc30392)

[四、系统架构详解 10](#_Toc27908)

[（一）表现层 10](#_Toc7815)

[1、表现层静态结构图 10](#_Toc3766)

[2、基于Area 的路由映射 11](#_Toc10800)

[（二）服务层 11](#_Toc25890)

[（三）数据层 11](#_Toc22493)

[（四）核心层 13](#_Toc29156)

[1、定时任务机制 13](#_Toc30294)

[2、异常处理机制 13](#_Toc13789)

[3、权限控制机制 14](#_Toc865)

[4、日志机制 14](#_Toc24187)

[5、插件（策略）机制 15](#_Toc2670)

[五、系统可扩展性介绍 16](#_Toc30269)

[（一）Web服务可扩展性 16](#_Toc11660)

[1、Web服务器负载均衡 17](#_Toc31324)

[2、Web服务支持动、静态文件分离部署 17](#_Toc21223)

[（二）缓存服务可扩展性 18](#_Toc1732)

[（三）数据库可扩展性 19](#_Toc871)

[（四）整体解决方案拓扑图 21](#_Toc4796)

[六、系统所使用的组件清单 22](#_Toc11055)

一、前言

本文档主要为您讲述Himall的技术实现，通过对各层次结构详细说明，让您可以更加清楚的了解Himall的系统组成和设计思想，并可以对其有一个全局的把握。本文针对的读者需要有一定的软件设计基础。

# 二、名词解释

|  |  |  |
| --- | --- | --- |
| 序号 | 名词 | 解释 |
| 1 | MVC | MVC一般指根据模型、视图以及控制器思想实现的开发模式，本文中特指微软ASP.NET MVC解决方案 |
| 2 | Plugins(插件) | 指通过实现主程序特定接口完成特定功能，并且能支持在不中断主程序运行的情况下加入到主程序中的模块。针对同一接口实现的插件通常可以同时加入到主程序中运行,即相同插件之间具有共存性。 |
| 3 | Strategies(策略) | 指通过实现主程序特定接口完成特定功能，只能在主程序运行前通过配置文件进行配置，并且针对同一接口只能配置一种策略，即相同策略之间具有排他性。 |

*注：上表中仅列出了在本文中有特殊含意的名词，其它未列出的名词按其通用含意理解。*

# 三、系统架构概况

**（一）技术路线**

Himall系统采用最新的技术方式，结合成熟的设计模式与解决方案，具体表现在以下几个方面：

1. **采用完全面向对象设计思想。**

有别于传统软件面对过程开发方式，Himall系统采用完全的面向对象的思想进行设计，无论是表现层的UI呈现，还是业务层复杂的业务实现，或者是数据库的访问方式均采用这种方式。得益于面向对象思想的设计，使Himall的开发过程更加符合人的思维方式，其具有的重用性也使得代码量和开发周期大大减少。

1. **采用MVC的开发模式。**

Himall摈弃了传统WebForm的开发方式，选用目前主流的MVC开发方式，该方式颠覆了WebForm时代的设计思想，抛弃了所有服务器控件，使用简洁服务器代码或者纯净的HTML进行页面呈现，使得表现层逻辑与业务分离。随着技术的不断进步，现在需要用越来越多的方式来访问应用程序。MVC模式允许使用各种不同样式的视图来访问同一个服务器端的代码。它包括任何WEB（HTTP）浏览器、无线浏览器（wap）、移动APP（IOS 和 安卓），比如，用户可以通过电脑也可通过手机来订购某样产品，虽然订购的方式不一样，但处理订购产品的方式是一样的。由于模型返回的数据没有进行格式化，所以同样的构件能被不同的界面使用。例如，很多数据可能用HTML来表示，但是也有可能用WAP来表示，也可以是原生的APP应用，而这些表示所需要的命令是改变视图层的实现方式，而控制层和模型层无需做任何改变。

1. **采用ORM框架实现数据的持久化。**

作为面向对象思想实现的一个重要产物，ORM框架从诞生开始就引起了广泛关注。目前ORM框架技术已经成熟并得到广泛使用，Himall采用了微软的Entity Framework（以下简称EF）作为系统的ORM框架，它目前支持多表映射，支持多种主流数据库。使用EF后，软件开发过程直接操作对象进行数据存储，避免了大量使用SQL语句，减少了系统开发的复杂性，同时也提高了系统的稳定性。

1. **采用插件机制实现相同业务的可热插拔**

Himall系统对于相同业务的不同实现采用了插件机制，例如信任登录、快递以及支付方式等业务均采用了该机制。使用插件机制后，各业务的实现仅依赖于主程序的核心层以及对应的插件接口，插件与主程序之间完全松耦合，主程序不直接操作和干涉插件的内部实现，仅在相应的业务场景调用对应插件的对应接口方法。插件可以在主程序行时实时的热插入（拔出）系统。因此，各类插件可以单独开发，然后根据客户需要加入到系统中。

1. **采用策略机制实现相同业务的可配置变更**

对于某些业务可能存在多种不同的实现方式，Himall采用策略机制通过配置文件对这些业务进行切换。通过这种方式，系统通过接口对业务进行抽象，使主程序不再依赖于某种业务的具体实现，而通过配置可以动态的调整需要使用的具体实现方式，因此使用策略机制后，系统具有很高的可扩展性。

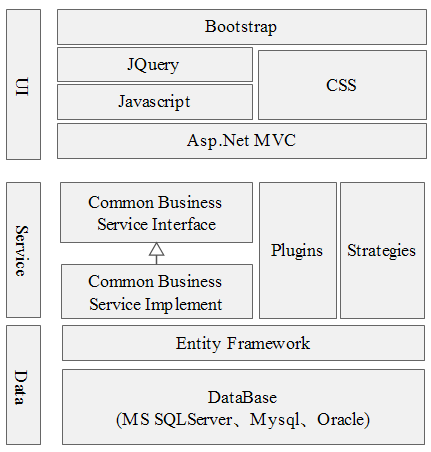
1. **UI基于主流的Bootstrap框架自主设计**

扁平风格是目前UI的主流风格，Himall基于业界最优秀的开源前端框架之一的Bootstrap对平台和商家中心进行了UI设计，使得界面外观上简洁清爽、优雅大气，代码结构上简单明了易于维护。

1. **基于JQuery插件式开发**

Himall系统中，界面上多处采用开源或自主开发的JQuery插件，例如使用开源日期插件、数字格式插件，自主开发的表格插件、联动下拉框插件等。通过使用这些JQuery插件，使得UI业务代码与呈现代码分离，通用配置可使各插件在不同情景下呈现不同的数据和样式，极大的提高了代码的可复用性和可维护性。

**（二）系统架构图**



**四、系统架构详解**

**（一）表现层**

**1、表现层静态结构图**



释义：

|  |  |  |
| --- | --- | --- |
| 序号 | 控制器名称 | 功能 |
| 1 | BaseController | 通用基类，定义UI层通用属性（如配置信息）、处理所有未捕获的系统异常 |
| 2 | BaseWebController | 前台通用基类，定义了当前用户信息 |
| 3 | BaseMemberController | 前台非匿名访问基类，定义了当前登录用户的信息 |
| 4 | BaseSellerController | 商家中心基类，定义了当前登录商家管理员账号相关信息 |
| 5 | BaseAdminController | 平台中心基类，定义了当前登录平台管理员账号相关信息 |

**2、基于Area 的路由映射**

对于大规模的WEB应用，从规模上通过Area 划分成小的单元，每个相当于独立的子系统，包含自己的Models、Views和Controller在内的目录结构和配置文件。Himall的UI层组织结构也是按此规则，建立了Admin、SellerAdmin以及Web三个区域并分别配置路由信息。

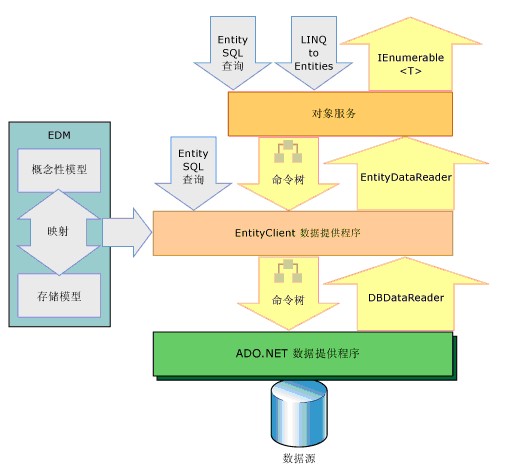
**（二）服务层**

IOC模式是目前常用的解耦方法之一，它能降低不同层级之间的的耦合度，使以前直接的实例之间的调用变更为对接口的调用，即上层不再依赖于具体实例而依赖于下层所提供的接口，此方式也属于面向接口编程。面向接口而非面向实例可以使程序更加稳定，不会因为实例中的方法修改导致上层代码的级联修改，只要接口不发生变化，则上层代码无需改变。Himall中使用AutoFac来实现IOC模式，所有服务层代码分成两个部分，一部分是接口，另一部分为实例。通常情况下，AutoFac需要通过配置文件指明实现类与接口之间的关系，而Himall中采用了默认配置的方式简化此操作，即按一定的规则编写接口和实现类就可以自动建立对应关系。

**（三）数据层**

数据层主要采用微软的Entity Framework（以下简称EF）做为ORM框架。使用EF的好处是应用程序可以通过更加以应用程序为中心的概念性模型、不再对特定的数据引擎或存储架构具有硬编码依赖性、可以在不更改应用程序代码的情况下更改概念性模型与特定于存储的架构之间的映射以及语言集成查询支持可为查询提供针对概念性模型的编译时语法验证。

EF架构图如下：



借助于EF，Himall项目中基本上不再有直接的SQL语句编写（复杂业务或性能要求高的场景除外）。脱离了SQL语句的编写使得代码的利用率大大提高，程序出错的概率大大降低，也为无缝迁移数据库类型创造了良好的基础。

在EF的默认配置中，业务模型和EF其它部分是在同一目录下。业务模型在项目中是无依赖且被其它几乎所有项目所引用（除Core外），业务模型混杂在EF中将导致EF被其它层引用，即整个EF对其它可视。这种引用关系严重破坏了各相邻层之间单向依赖性，因此在Himall中我们对EF的配置进行了改进，将EF的映射关系部分与模型代码部分分离到两个独立的项目之中，这样其它层就可以单独对模型进行引用，而不需要引用EF的其它部分。

业务模型属性的很多时候不仅仅来源于数据库对应的物理属性，也会来自于业务自身。这种属性不需要进行持久化，仅在程序运行过程中使用。为了应对这种场景，Himall系统中使用分部类建立了与EF自建模型相对应模型文件。

**（四）核心层**

核心层（Core）为Himall最底层项目，该项目不依赖于任何项目，是与具体业务无关接口与方法的集合。它的作用主要是为上层提供通用帮助方法，如IOHelper、SecureHelper等，它还定义和实现了相关机制，如异常处理、日志处理、定时任务、权限等。主要机制的详细情况如下：

**1、定时任务机制**

Himall中定时任务采用开源组件Quartz.Net。该组件是一款成熟稳定的任务组件，它可以用来创建简单或为运行十个，百个，甚至是好几万个任务这样复杂的程序。在Himall中，任务的实现代码必须继承于IJob接口，然后在配置文件中对任务的执行周期等参数进行配置，任务即会在指定的时间或周期自动执行。

**2、异常处理机制**

Himall的异常处理机制终结于UI层。异常的终极处理采用AOP思想，将对异常处理的实现置于BaseController的OnException方法中，从而，系统下层所有未处理的异常均会被其所捕获并做处理。异常处理时，对于非继承于HimallException的异常将记录日志并返回友好错误消息，对于继承于HimallException的异常将直接返回基异常信息的具体内容。

**3、权限控制机制**

Himall权限控制机制也是采用AOP思想，在BaseAdminController和BaseSellerController中的OnAuthorization方法中统一处理，每个Action的权限均在对应的Action上进行标记。

**4、日志机制**

Himall日志机制分为两大类，一是系统日志，另一个是操作日志。系统日志主要侧重于对系统运行的监控，及时的记录系统中各类异常情况，它不面向用户，仅供开发人员使用。操作日志则是系统功能需求的一部分，它详细记录下系统中每个用户所有重要操作的时间及操作内容，用于向系统管理员展示系统其它用户的操作行为。

**系统日志**：采用开源日志组件Log4net。Log4Net是一款优秀、稳定的开源组件，广泛用于各类大中型系统的日志机制中，它支持层级日志体系、可使用XML灵活的进行配置。Himall的系统日志分为三大级别：Info、Debug和Error。三个级别的日志分别通过配置文件写入不同的目录，各级别的日志按天分文件记录。日志的内容主要包括日志记录时的时间、调用日志的方法名、相关自定义日志信息、相关堆栈信息。在Himall中，绝大部分写日志的行为都是自动进行的，凡继承于HimallException的异常均会自动记录日志信息，所有未经处理的非继承于Himall的异常均会在BaseController中集中处理并记录日志。

**操作日志**：操作日志的载体是数据库，系统中所有重要的操作通过Action上标记Attribute的方式进行记录。记录操作日志的过程是异步的，使得日志记录的过程不会影响到用户操作的响应速度。

**5、插件（策略）机制**

Himall对于相同业务场景的不同实现采用了插件化设计理念。例如，支付方式有很多种不同的选择（如支付宝、银联等），按传统开发方法直接在主程序中实现这些支付方式就会导致每增加或修改某种支付方式都将导致主程序发生变更，而且所为支付方式所实现的代码与主程序耦合度高，无法简单直接的用于其它产品和项目。而使用插件机制后，主程序将不再实现具体的支付方式，仅仅预留接口，主程序只是从流程上调用抽象的接口方法，而接口的具体实现与主程序再无关系。

插件机制静态结构如下：



核心层（Core）中对通用插件接口（IPlugin）进行了定义，所有具体业务插件接口都须要继承至IPlugin。目前Himall共有三种类型的业务插件，分别是快递插件、支付插件和信任登录插件，开发插件时只需要实现对应插件接口即可。

策略机制与插件机制类似，都是仅依赖于主程序公开的接口，主程序不对具体策略的实现直接依赖，仅依赖于策略接口。不同于插件的是，同一类型的策略往往在主程序中只能同时运行其中一种实现，不同实现之间是不能同时运行的，因此策略也不具体热插拔性，只能在系统启动时，通过已配置好的配置文件读取要加载的策略，系统一旦启动便不能再做修改。目前Himall中使用策略机制的主要是缓存，默认提供两套缓存机制的实现（Asp.net缓存与Memcached缓存），通过在Web.config中配置加载到系统中。

策略机制的静态结构如下：



# 五、系统可扩展性介绍

Himall在设计之初便考虑了对大数据量以及大访问量的可扩展性，因此系统各层次在设计时便加入了相应的机制。

**（一）Web服务可扩展性**

一般而言，Web服务器可能遭遇的瓶颈主要集中在大量的并发访问上。针对大并发量的情况，Himall的Web服务可以做以下扩展：

**1、Web服务支持动、静态文件分离部署**

网站中有很多需要发送给浏览器但却不会因为日常操作而发生变化的资源。例如用于描绘网站外观的图片和CSS文件、在浏览器中运行的JavaScript代码、不含动态组件的HTML网页文件等都属于这样的资源。由于发送这些文件时不会用到应用程序代码，因此没有必要把它们放到应用程序服务器上,而且放在同一台服务器上会造成请求时的响应缓慢。因此，分离开静态文件与动态文件，有利于网站整体响应速度的提高。

Himall支持将所有静态文件进行分离，分离后部署与访问示意图如下：

****

**2、Web服务器负载均衡**

负载均衡是最常用的解决大并发的手段之一。Himall系统Web部分支持由1到任意台服务器的扩展。网络拓扑图如下：



**（二）缓存服务可扩展性**

Himall支持多种缓存机制，包括Asp.net缓存和Memcached缓存，并且支持其它KV缓存与Himall的集成。当前网站访问量增大，各类缓存数据增多时，往往一台服务器再无法支撑。通过分布式缓存系统（如Memecached）可以使缓存与应用服务器分离，并且分离后缓存服务器可以建立集群并向其中任意增加服务器节点。示意图如下：



**（三）数据库可扩展性**

Himall使用开源的MySQL作为数据库，数据层采用EF做为ORM框架，通过使用不同的EF数据库连接器，可以在不修改任何程序代码的情况无缝切换到Oracle或者第三方的数据库。（以下未特别注明的数据库均以Mysql为例）

对于数据库访问量大并且一台数据库服务器无法满足需求时，可以做数据库简单的读写分离。



当某一张表数据量过大时（如订单表、商品表等），可以采用垂直分割数据库的形式，按数据库按业务功能进行分割，从而一部分功能的数据库查询不会影响到另一个功能的正常使用，也能整体提高查询效率和网站的响应速度。拓扑结构如下图所示：



当数据库单表过大造成查询极为缓慢时，可以采用水平拆表，通过一定的规则，将大表的数据拆分到多个数据库的多个表中，将一个大表查询分解成多个小表的查询，从而提高数据库查询效率。拓扑结构如下：



**（四）整体解决方案拓扑图**



**六、系统所使用的组件清单**

|  |  |  |  |
| --- | --- | --- | --- |
| **组件名称** | **是否开源** | **开源协议** | **源码库地址** |
| Asp.net Mvc | 是 | Apache License 2.0 | https://github.com/aspnet/Mvc |
| EntityFramework | 是 | Apache License 2.0 | https://github.com/aspnet/EntityFramework |
| Razor | 是 | Apache License 2.0 | https://github.com/aspnet/Razor |
| .net framework 4.5.1 | 是 | MS-RSL | http://referencesource.microsoft.com/ |
| AutoFac | 是 | MIT | https://github.com/autofac/Autofac |
| Apache Log4net | 是 | Apache License 2.0 | https://github.com/apache/log4net |
| Quartznet | 是 | Apache License 2.0 | https://github.com/quartznet/quartznet |
| AutoMapper | 是 | Apache License 2.0 | https://github.com/AutoMapper/AutoMapper |
| Newtonsoft.Json | 是 | MIT | https://github.com/JamesNK/Newtonsoft.Json |
| Jquery | 是 | MIT | https://github.com/jquery/jquery |
| AutoNumeric | 是 | MIT | https://github.com/BobKnothe/autoNumeric |
| Bootstrap | 是 | MIT | https://github.com/twbs/bootstrap |
| ArtDialog | 是 | LGPL | https://github.com/aui/artDialog |
| Echarts | 是 | MIT | https://github.com/ecomfe/echarts |
| Lodop | 否 |  | http://www.lodop.net/ |

**六、系统环境要求**

 WEB 服务器要求：Windows2008+

 数据库服务器环境：Linux、Windows2008+

 语言环境：ASP.NET 4.5

 数据库环境： Mysql 或 Oracle

推荐使用环境：Windows2008+.Net4.5+IIS7+MySQL5.6