

目 录

1、 概述	1
1.1 项目技术介绍.....	1
1.2 B/S 架构拓扑图.....	1
1.3 MVC 模式	2
2、 架构	3
2.1 整体架构.....	3
2.2 项目架构.....	4
3、 架构实现.....	5
3.1 术语和原则.....	5
3.2 技术参考资料.....	5
3.3 项目结构.....	6
3.3.1 解决方案结构图	6
3.3.2 解决方案项目(文件夹)说明	6
3.4 项目间调用关系图.....	7
4、 关键技术介绍	7
4.1 AUTOFAC 的核心作用	7
4.1.1 Autofac 使用说明.....	8
4.2 数据访问层.....	8
4.2.1 数据访问持久类介绍	9
4.2.2 IBatis.Net 配置介绍.....	10
4.2.3 动态查询对象	13
4.3 业务逻辑层	14
4.3.1 业务逻辑层基类介绍	15
4.3.2 业务逻辑层实现	15
4.4 显示层.....	15

1、概述

1.1 项目技术介绍

开发工具：Visual Studio 2015

服务器：Windows Server 2008 R2(x64)、IIS7

数据库：Sql Server 2008 R2

后台技术：Asp.Net MVC 5、Autofac、IBatis.Net

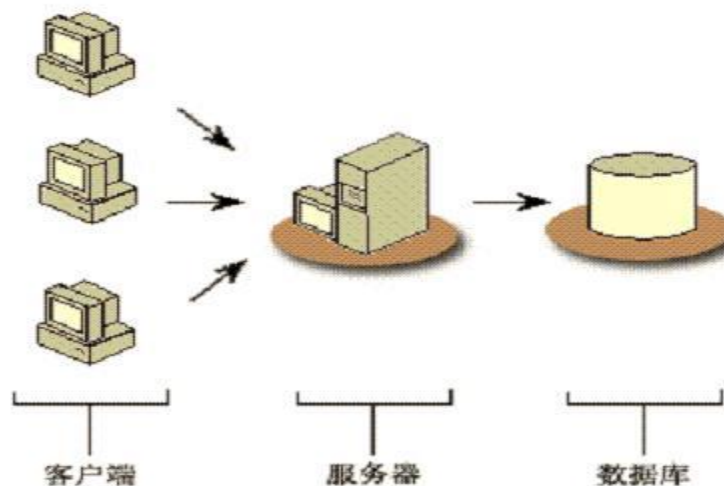
前端技术：Html5、JavaScript、CSS3、JQuery 2.X

浏览器：Chrome 28+、IE10+

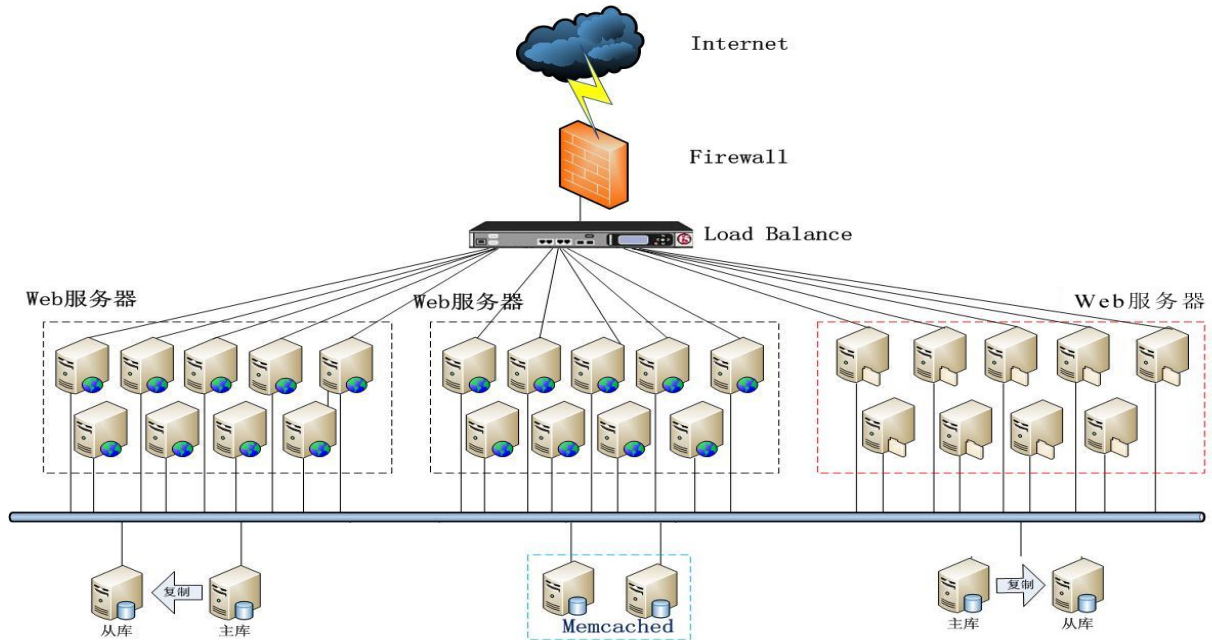
1.2 B/S 架构拓扑图

本系统架构可以同时满足单机（一台 Web 应用程序服务器）部署和基于负载均衡的分布式部署方式。以下是这两种部署的拓扑图。

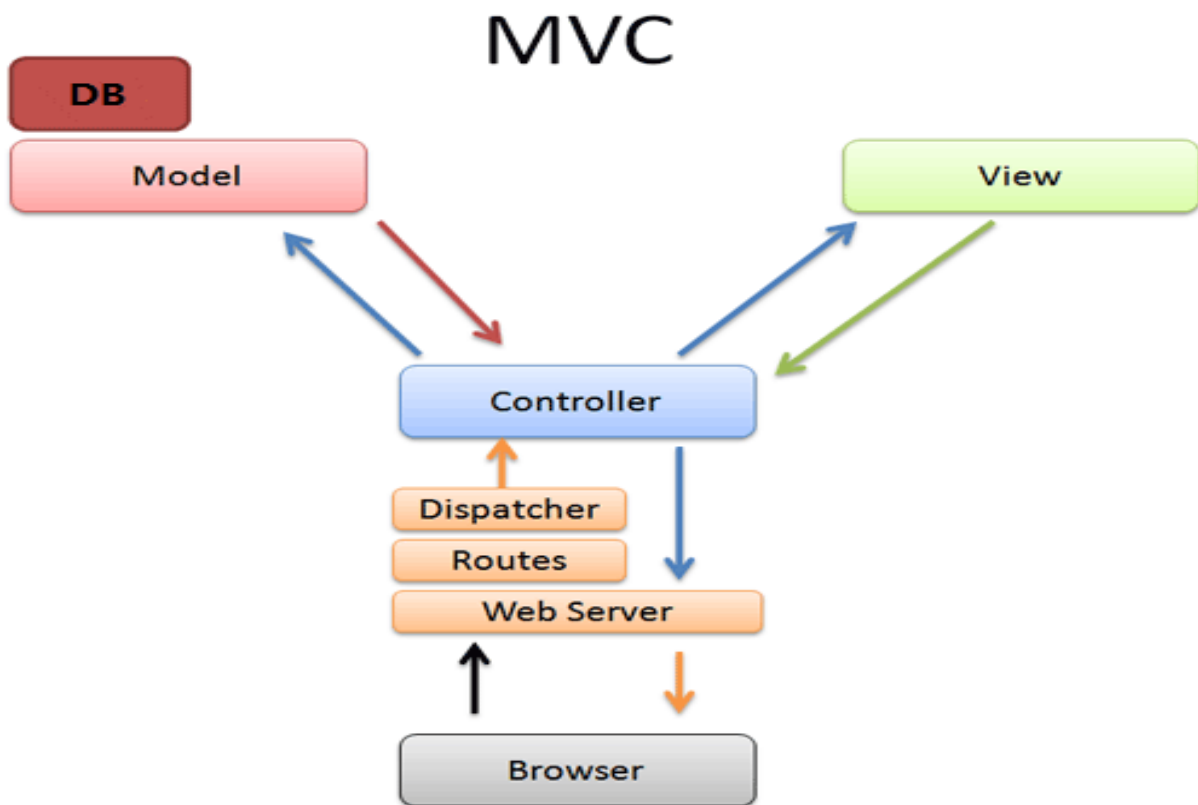
➤ 单机部署拓扑图



➤ 基于负载均衡的拓扑图



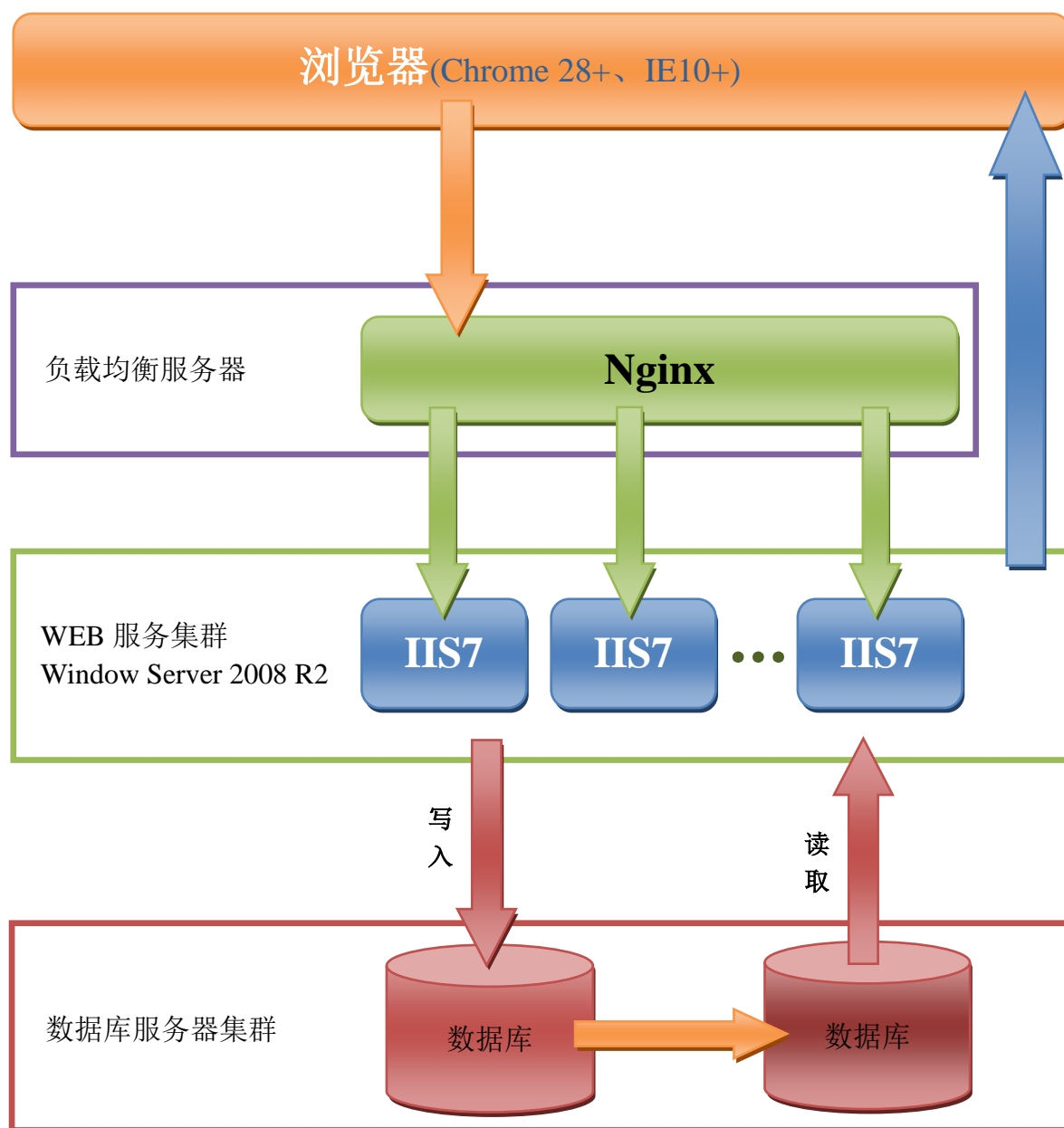
1.3 MVC 模式



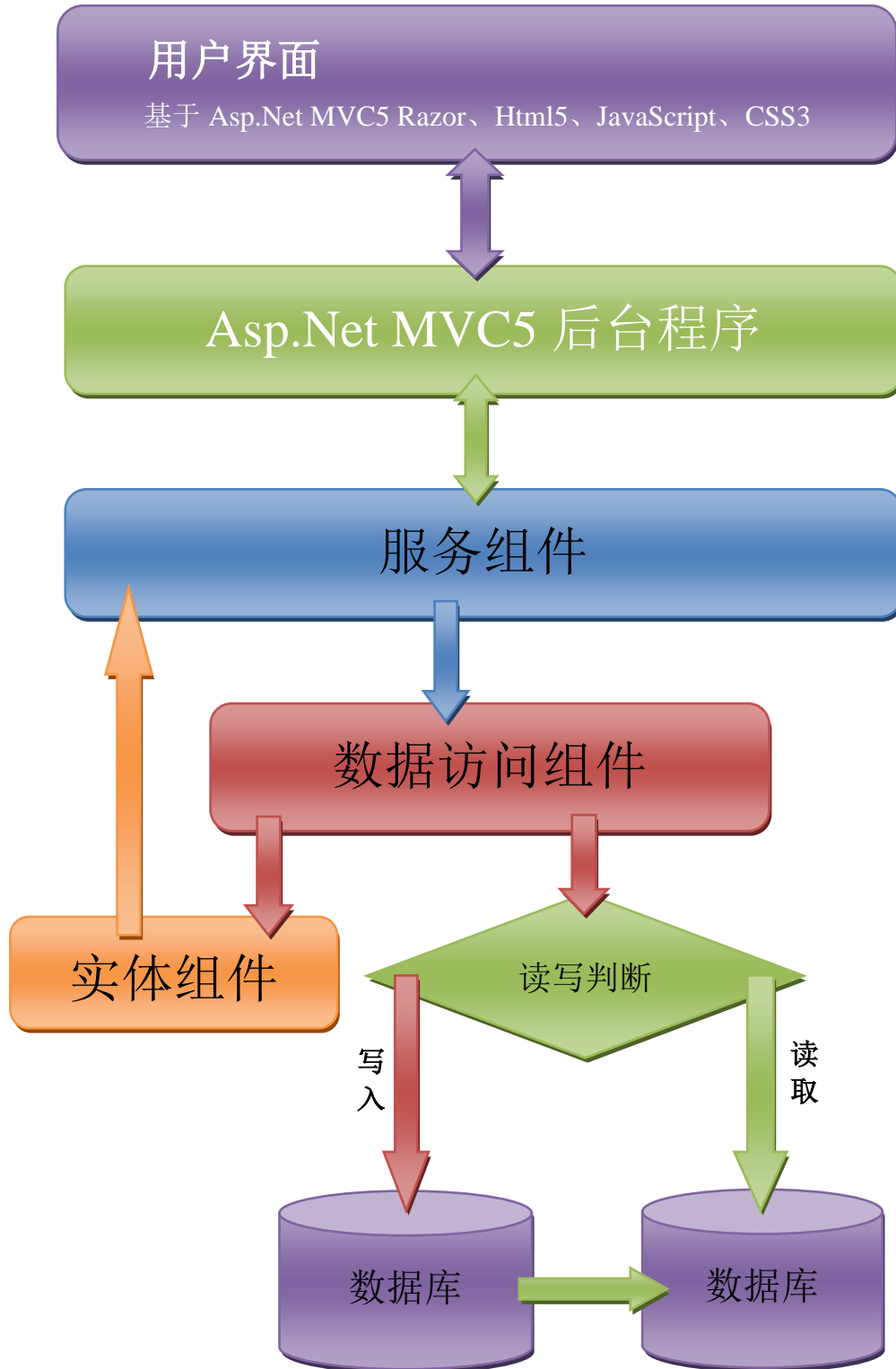
2、架构

由于本项目可能需要面向互联网，且接入到本系统的用户众多。因此单节点部署是不能应对如此大并发的用户访问。故此，我们在做出了如下的设计。

2.1 整体架构



2.2 项目架构



3、架构实现


3.1 术语和原则

- 三层架构：指的是数据访问层、业务逻辑层和界面层。
- IoC(Inversion of Control)：即控制反转，是指对象间的依赖交由第三方处理。即如果 A 对象中使用 B 对象，则 B 对象不在 A 对象中创建，而是交由其他对象创建，然后将 B 对象赋值给 A 对象。
- DI(Dependency Injection)：即依赖注入，是指对象的间的依赖交由第三方指定。即如果 A 对象依赖 B 对象，则 B 对象的创建由第三方指定。DI 是 IoC 实现的一种方式，也是目前的主流的方式。
- AOP：面向切面编程，即动态在方法执行前、后或异常发生时动态追加执行代码。
- ORM：对象关系映射，即将数据库中的关系数据映射为程序中的对象。
- 面向接口编程：即所有具体类的依赖均依赖于接口或者抽象类，而不允许依赖具体实现类。这样做的好处是可以灵活扩展依赖（只需要实现不同的依赖接口或抽象类）。
- IBatis.Net：ORM 框架的一种实现，采用 Sql 映射的方式来实现关系数据和程序对象间的映射。
- Autofac：是一款基于 .Net Framework 的轻量级 IoC 框架。

3.2 技术参考资料

- Autofac 系列博文：<http://www.cnblogs.com/WeiGe/p/3871451.html>
- IBatis.Net 系列博文：<http://www.cnblogs.com/hjf1223/archive/2007/10/18/928862.html>

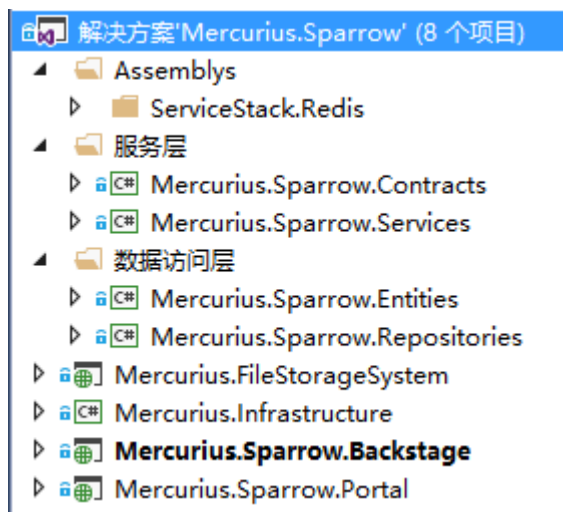
- 参考资料： Asp.Net MVC3.zip  IBatis.Net资料.zip

- Visual Studio 支持(IBatis.Net 配置文件 XSD):  Schemas.zip

(将该压缩包中的文件解压到 Visual Studio 安装目录下的 Xml\Schemas 目录后，Visual Studio 将提供 IBatis.Net 配置文件编写时的智能提示。)

3.3 项目结构

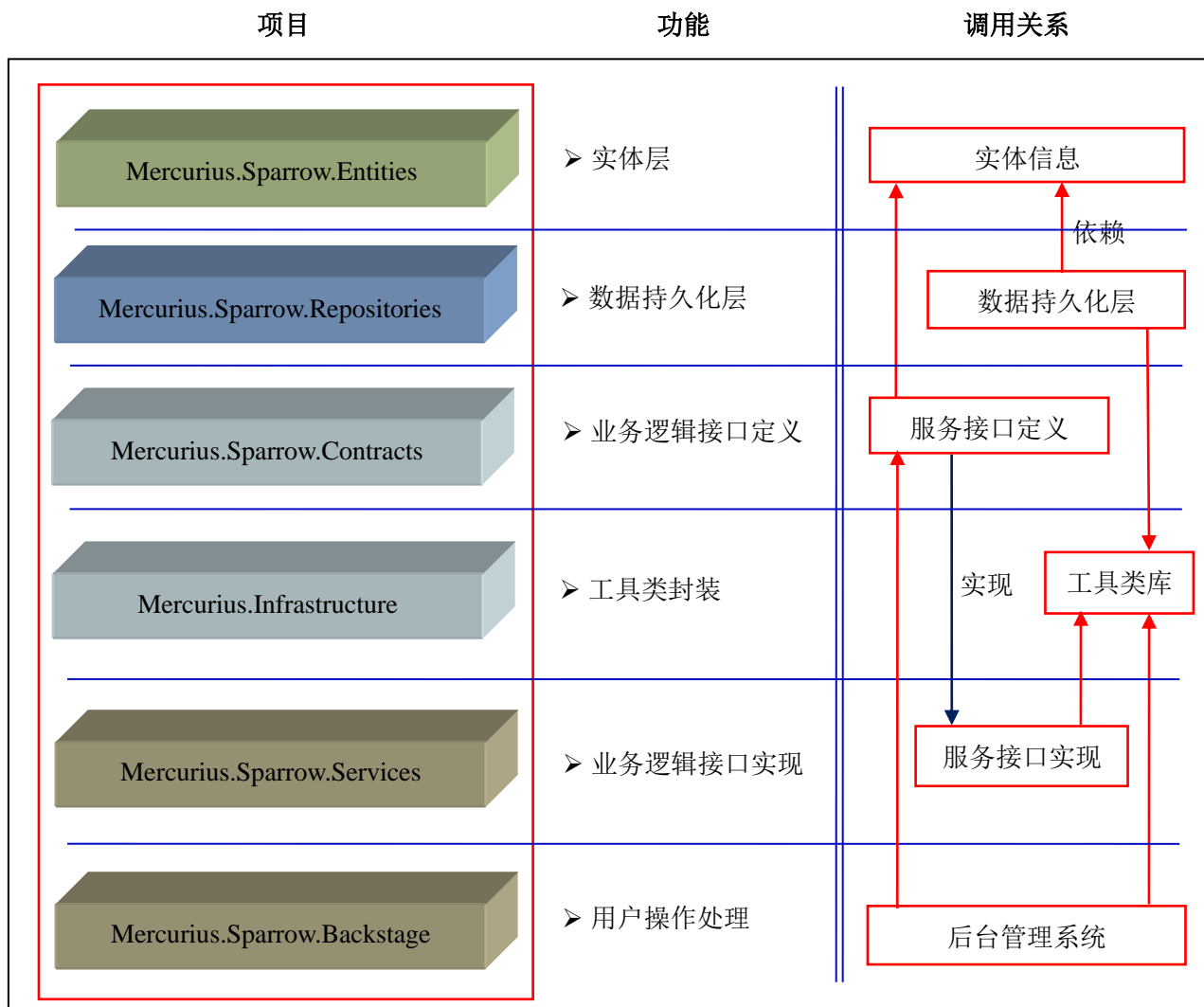
3.3.1 解决方案结构图



3.3.2 解决方案项目 (文件夹) 说明

- [Assemblys]解决方案文件夹 包含解决方案所需要的外部程序集。
- [服务层]解决方案文件夹 包含服务接口项目和服务接口实现项目。
- [数据访问层]解决方案文件夹 包含系统数据库实体类和数据持久化实现。
- **Mercurius.Infrastructure** 项目 包含公司积累的常用帮助类。
- **Mercurius.Sparrow.Entities** 项目 包含系统项目数据访问实体类。
- **Mercurius.Sparrow.Repositories** 项目 包含系统项目数据持久化实现。
- **Mercurius.Sparrow.Contracts** 项目 包含系统业务接口定义。
- **Mercurius.Sparrow.Services** 项目 包含业务接口定义的实现。
- **Mercurius.Sparrow.Portal** 项目 基于 Asp.Net MVC5 的前台系统。
- **Mercurius.Sparrow.Backstage** 项目 基于 Asp.Net MVC5 的后台管理系统。
- **Mercurius.FileStorageSystem** 项目 基于 Asp.Net MVC5 的文件管理系统，用于统一管理上传文件并提供文件上传、下载和删除的 Web API 接口。

3.4 项目间调用关系图



注: Mercurius.FileStorageSystem、Mercurius.Sparrow.Portal 项目的调用关系与上图一致。

4、 关键技术介绍

4.1 Autofac 的核心作用

注意：如对 Autofac 不熟悉，请先学习 [《3.2 技术参考资料》](#) 章节提供的 Autofac 参考资料，本节内容默认开发者已经对 Autofac 有了解。

在 OMS 系统中，所有对象的创建和生命周期管理都交由 Autofac 来管理，这样做的好处有如下几点：

- 1、对象间的依赖由 Autofac 指定，这样可以灵活替换依赖的对象，便于扩展。
- 2、对象的生命周期由 Autofac 统一管理，这可以最大化的优化系统性能。

4.1.1 Autofac 使用说明

Autofac 作为 IoC 容器，要在程序中发挥作用，则必须在程序启动时加载 Autofac。为了使其在 Asp.Net MVC 应用程序启动时加载 Autofac，我们需要做如下处理：

- 1、配置 Autofac：Autofac 的配置统一放在如下图(图 4.1.1-1)所示的项目目录中（其它的 Web 项目将这里的文件以添加现有文件链接的方式添加到项目中）：

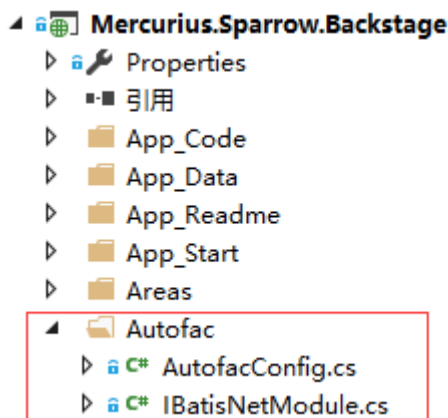


图 4.1.1-1

AutofacConfig.cs 文件：Autofac 的配置加载类，用于在 Autofac 容器中注册对象、设置对象依赖规则以及控制对象的生命周期（对象在何时创建、何时回收）。

IBatisNetModule.cs 文件：IBatisNet 相关对象的注册和生命周期控制。

- 2、在 Global.asax 文件的 Application_Start 方法的最后添加如下的代码：

```
// 设置Asp.Net依赖解析。  
DependencyResolver.SetResolver(new AutofacDependencyResolver(AutofacConfig.Container));
```

4.2 数据访问层

注意：如对 IBatis.Net 不熟悉，请先学习 [《3.2 技术参考资料》](#) 章节提供的 IBatis.Net 参考资料，本节内容默认开发者已了解 IBatis.Net 的使用。

数据访问层是基于 IBatis.Net，同时还支持数据库读写库分离。实现数据持久化时，必须先在 Mercurius.Sparrow.Repositories 项目对应的模块文件夹下的 SqlMap 文件夹（如无 SqlMap 文件夹，则添加一个）中追加一个 SqlMap 的配置文件(需要将该 xml 配置文件的“生成操作”属性设置为“嵌入

的资源”。

SqlMap 配置文件 namespace 的命名规则建议按如下方式命名(“<>”包含的表示必填):

SqlMap 文件 namespace: Mercurius.Sparrow.Repositories.<模块名>.<业务名>

然后在 Mercurius.Sparrow.Repositories 项目下的 StatementNamespaces 类的以模块名命名的子类中添加一个只读静态变量, 变量的定义如下:

Public static readonly StatementNamespace <业务名> = "<SqlMap 文件的 namespace>";

以上配置完成后, 需要在 Mercurius.Sparrow.Backstage 项目 App_Data\IBatisNet 目录下的 SqlConfig-*.xml(包含 SqlConfig-Reader.xml 和 SqlConfig-Writer.xml)文件的 sqlMaps 元素中追加一个子元素, 格式为: <sqlMap embedded="<模块名>.<业务名>.xml, Mercurius.Sparrow.Repositories" />

4.2.1 数据访问持久类介绍

所有的服务层实现类必须使用 Persistence 类(该类位于 Mercurius.Sparrow.Repositories 项目的 Support 目录下), 在该类中封装了 IBatis.Net 常用 API 以简化 IBatis.Net 的使用。

IBatis.Net API 封装: 简化 IBatis.Net API 的使用, 目前提供的方法及其功能如下:

方法名	参数	返回值	说明
Create	ns:Statement 命名空间 innerId:SqlMap 内部命令 Id parameterObject:SqlMap 引用参数	无	如启用读写库分离 则在写库中添加数据
Update	ns:Statement 命名空间 innerId:SqlMap 内部命令 Id parameterObject:SqlMap 引用参数	int (受影响的行数)	如启用读写库分离 则在写库中修改数据
Delete	ns:Statement 命名空间 innerId:SqlMap 内部命令 Id searchObject:SqlMap 引用参数	int (受影响的行数)	如启用读写库分离 则在写库中删除数据
QueryForObject	ns:Statement 命名空间 innerId:SqlMap 内部命令 Id searchObject:SqlMap 引用参数, 可 null	泛型对象	如启用读写库分离 则从读库中获取数据
QueryForList	ns:Statement 命名空间 innerId:SqlMap 内部命令 Id searchObject:SqlMap 引用参数, 可 null	泛型集合 (IList<T>)	如启用读写库分离 则从读库中获取数据
QueryForPaginatedList	ns:Statement 命名空间 innerId:SqlMap 内部命令 Id totalRecords:out 参数, 总记录数 searchObject:SqlMap 引用参数, 可 null	泛型集合 (IList<T>)	如启用读写库分离 则从读库中获取数据
ExecuteStoredProcedure	rw:指定操作是针对读库还是写库	无	

	produceName:过程名称 commandHandler:DbCommand 处理回调, 可 null		
QueryForDataTable	ns:Statement 命名空间 innerId:SqlMap 内部命令 Id searchObject:SqlMap 引用参数, 可 null	DataTable 对象	如启用读写库分离 则从读库中获取数据

4.2.2 IBatis.Net 配置介绍

IBatis.Net 配置文件分为以下三类:

- 1、 providers.config(位于 Mercurius.Sparrow.Backstage 项目的 App_Data\IBatisNet 目录下): 配置数据库提供者程序, 主要作用是配置所使用的数据库提供者参数。如下图所示的 Sql Server 数据库提供者配置, **要使用该 provider 时其中 enabled 属性必须设置为 true**(其他配置信息比较简单, 从数据的名称就可以看出是配置什么了, 故不作解释)。如 Sql Server 的数据库提供者版本发生变化, 只需修改 assemblyName 属性即可。

```
<provider
  name="sqlServer2.0"
  enabled="true"
  default="false"
  description="Microsoft SQL Server, provider V2.0.0.0 in framework .NET V2.0"
  assemblyName="System.Data, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
  connectionClass="System.Data.SqlClient.SqlConnection"
  commandClass="System.Data.SqlClient.SqlCommand"
  parameterClass="System.Data.SqlClient.SqlParameter"
  parameterDbTypeClass="System.Data.SqlDbType"
  parameterDbTypeProperty="SqlDbType"
  dataAdapterClass="System.Data.SqlClient.SqlDataAdapter"
  commandBuilderClass="System.Data.SqlClient.SqlCommandBuilder"
  usePositionalParameters = "false"
  useParameterPrefixInSql = "true"
  useParameterPrefixInParameter = "true"
  parameterPrefix="@@"
  allowMARS="false" />
```

- 2、SqlConfig-*.xml(位于 Mercurius.Sparrow.Backstage 项目的 App_Data\IBatisNet\MSSQL 目录下): 主要配置数据库连接及管理 SqlMap 文件。

解读 SqlConfig-*.xml 配置文件(以 SqlConfig-Reader.xml 为例)

导入外部资源(提供数据库访问参数信息):

```
<properties resource="App_Data/properties.config" />
```

IBatis.Net 全局配置:

```
<settings>
  <!-- 使用启用SqlMapConfig.xsd来验证映射XML文件 -->
  <setting validateSqlMap="false" />

  <!-- 是否启用缓存 -->
  <setting cacheModelsEnabled="true" />

  <!-- 是否使用全局完整命名空间 -->
  <setting useStatementNamespaces="true" />

  <!-- 是否使用反射机制访问C#中对象的属性 -->
  <setting useReflectionOptimizer="true" />

  <!-- 是否使用嵌入的方式声明可变参数 -->
  <setting useEmbedStatementParams="true" />
</settings>
```

加载数据库提供者配置:

```
<providers resource="App_Data/IBatisNet/providers.config" />
```

数据库连接配置:

从上面提到的导入的外部资源文件中读

```
<database>
  <provider name="sqlServer2.0"/>
  <dataSource name="MSSQLDataSource" connectionString="Data Source=${rhost};Initial Catalog=${rsid};Persist
</database>
```

全局类型别名(简化类型的引用)配置:

```
<alias>
  <typeAlias alias="DataTable" type="System.Data.DataTable" />
  <typeAlias alias="Params" type="System.Collections.IDictionary" />
  <typeAlias alias="Entry" type="Mercurius.Sparrow.Entities.Chart.Entry, Mercurius.Sparrow.Entities" />
  <typeAlias alias="Series" type="Mercurius.Sparrow.Entities.Chart.Series, Mercurius.Sparrow.Entities" />
  <typeAlias alias="SearchObject" type="Mercurius.Sparrow.Entities.SearchObject, Mercurius.Sparrow.Entities" />
</alias>
```

导入 IBatis.Net 使用的 SqlMap 配置文件：

```
<sqlMaps>
  <!-- 基础模块 -->
  <sqlMap embedded="Core.MSSQL.Logger.xml, Mercurius.Sparrow.Repositories" />
  <sqlMap embedded="Core.MSSQL.OperationRecord.xml, Mercurius.Sparrow.Repositories" />
  <sqlMap embedded="Core.MSSQL.Dictionary.xml, Mercurius.Sparrow.Repositories" />
  <sqlMap embedded="Core.MSSQL.SystemSetting.xml, Mercurius.Sparrow.Repositories" />
  <sqlMap embedded="Core.MSSQL.Globalization.xml, Mercurius.Sparrow.Repositories" />
  <sqlMap embedded="Core.MSSQL.RepositoryUtils.xml, Mercurius.Sparrow.Repositories" />

  <!-- 文件存储模块 -->
  <sqlMap embedded="Storage.MSSQL.File.xml, Mercurius.Sparrow.Repositories" />

  <!-- 基于角色访问控制 (RBAC) 模块 -->
  <sqlMap embedded="RBAC.MSSQL.User.xml, Mercurius.Sparrow.Repositories" />
  <sqlMap embedded="RBAC.MSSQL.Role.xml, Mercurius.Sparrow.Repositories" />
  <sqlMap embedded="RBAC.MSSQL.Button.xml, Mercurius.Sparrow.Repositories" />
  <sqlMap embedded="RBAC.MSSQL.Permission.xml, Mercurius.Sparrow.Repositories" />
  <sqlMap embedded="RBAC.MSSQL.Organization.xml, Mercurius.Sparrow.Repositories" />
  <sqlMap embedded="RBAC.MSSQL.HomeShortcut.xml, Mercurius.Sparrow.Repositories" />

  <!-- 动态属性模块 -->
  <sqlMap embedded="Dynamic.MSSQL.ExtensionProperty.xml, Mercurius.Sparrow.Repositories" />

  <!-- Web API管理模块 -->
  <sqlMap embedded="WebApi.MSSQL.Api.xml, Mercurius.Sparrow.Repositories" />
  <sqlMap embedded="WebApi.MSSQL.User.xml, Mercurius.Sparrow.Repositories" />
  <sqlMap embedded="WebApi.MSSQL.Role.xml, Mercurius.Sparrow.Repositories" />

  <!-- 新闻中心模块 -->
  <sqlMap embedded="NewsCenter.MSSQL.News.xml, Mercurius.Sparrow.Repositories" />
  <sqlMap embedded="NewsCenter.MSSQL.NewsComment.xml, Mercurius.Sparrow.Repositories" />
</sqlMaps>
```

3、SqlMap 配置文件：这些文件集中在 Mercurius.Sparrow.Repositories 项目的各个模块下的 SqlMap 文件夹中，其作用是提供 IBatis.Net 需要使用的 statement。

解读 SqlMap 配置文件（以 Mercurius.Sparrow.Repositories 项目下的 Core\SqlMap\Logger.xml 文件为例）

指定 namespace 属性(命名规则请遵照 [《4.2 数据访问层》](#) 章节的命名规范)：

```
<sqlMap namespace="Mercurius.Sparrow.Repositories.Core.Logger" xmlns="http://ibatis.apache.org/mapping">
```

配置当前 SqlMap 所用类型的别名(与 SqlConfig-*.xml 中的作用相同)：

```
<alias>
  <typeAlias alias="Log" type="Mercurius.Sparrow.Entities.Core.Log, Mercurius.Sparrow.Entities" />
  <typeAlias alias="LogSO" type="Mercurius.Sparrow.Entities.Core.SO.LogSO, Mercurius.Sparrow.Entities" />
</alias>
```

增加 Statements 子元素：请参见 [《3.2 技术参考资料》](#)。

4、 properties.config(在 Mercurius.Sparrow.Backstage 项目的 App_Data 目录下): 提供数据库连接参数。

```
<?xml version="1.0" encoding="utf-8"?>
<settings>
  <!-- Oracle数据库(读库) -->
  <add key="rhost" value="192.168.5.97" />
  <add key="rport" value="1521" />
  <add key="rsid" value="orcl" />
  <add key="rusername" value="oms" />
  <add key="rpassword" value="devdev" />
  <!-- Oracle数据库(写库) -->
  <add key="whost" value="192.168.5.97" />
  <add key="wport" value="1521" />
  <add key="wsid" value="orcl" />
  <add key="wusername" value="oms" />
  <add key="wpassword" value="devdev" />
</settings>
```

4.2.3 动态查询对象

在控制器中(Mercurius.Sparrow.Backstage 项目中)可以使用 DynamicQuery 对象来处理数据库的增、删、改和查询数据。使用方法如下图所示：

```
model.Columns = this.DynamicQuery.DbMetadata.GetColumns(id);
model.Search = this.DynamicQuery.Where(Condition.Eq("TableName", id)).Single<SearchInfo>();
model.Conditions = this.DynamicQuery.Where(Condition.Eq("TableName", id)).List<ConditionInfo>();

var showColumns = model.Search == null ? null : model.Search.GetVisibleColumns();
var orders = this.DynamicQuery.Where(Condition.Eq("TableName", id)).List<OrderInfo>().Select(o => (Order)o);

int totalRecords;
if (conditions.IsEmpty())
{
    model.DataSource = this.DynamicQuery.OrderBy(orders.ToArray())
        .PagedList(id, pageIndex < 1 ? 1 : pageIndex, SearchObject.DefaultPageSize, out totalRecords, showColumns);
}
```

动态查询提供的 API 如下表所示：

(泛型方法的类型 T 的要求:类需要添加 TableAttribute 特性,和数据库字段有映射关系的属性,需要添加 ColumnAttribute 特性。这两个特性均在 Mercurius.Infrastructure.Ado 命名空间下)

方法名	参数	返回值	说明
Create	tableName:表名称 columnValues:列-值集合	decimal	如果主键是自动增长列,则返回主键值,否则返回受影响的记录数。
Update	tableName:表名称 columnValues:列-值集合	int	返回受影响的行数

Update	tableName:表名称 fields:只更新的列值对象	int	返回受影响的行数
CreateOrUpdate	tableName:表名称 columnValues:列-值集合	int	返回受影响的行数
Remove	tableName:表名称	int	返回受影响的行数 删除条件通过 Where 方法添加。
Single	tableName:表名称 columns:返回的数据列, 可 null	DataRow	查询条件通过 Where 方法添加。
List	tableName:表名称 columns:返回的数据列, 可 null	DataTable	查询条件通过 Where 方法添加。
PagedList	tableName:表名称 pageIndex:当前页 pageSize:每页显示记录数 totalRecords:总记录数, out 参数 columns:返回数据的列, 可 null	DataTable	查询条件通过 Where 方法添加。
Create<T>	entity:实体类	decimal	如果主键是自动增长列, 则返回主键值, 否则返回受影响的记录数。
Update<T>	entity:实体类	int	返回受影响的行数
Update<T>	fields:需要更新的属性	int	返回受影响的行数
CreateOrUpdate<T>	entity:实体类	int	返回受影响的行数
Remove<T>		int	返回受影响的行数 删除条件通过 Where 方法添加。
Single<T>	columns:返回数据的列集合, 可 null	T	查询条件通过 Where 方法添加。
List<T>	columns:返回数据列集合, 可 null	IList<T>	查询条件通过 Where 方法添加。
PagedList<T>	pageIndex:当前页 pageSize:每页显示记录数 totalRecords:总记录数, out 参数 columns:返回数据的列, 可 null	IList<T>	查询条件通过 Where 方法添加。
Where	condition:查询条件	Criteria	查询信息对象, 支持追加 And、Or 条件
Where	conditions:查询条件集合(object)	Criteria	查询信息对象, 支持追加 And、Or 条件
OrderBy	orders:排序信息集合	DynamicQuery	动态查询对象
OrderBy	propertyName:查询属性名称 orderBy:排序方式, 默认值为升序	DynamicQuery	动态查询对象

4.3 业务逻辑层

实现业务逻辑层时, 必须先在 Mercurius.Sparrow.Contracts 项目相应的模块文件夹中创建业务逻

辑接口。然后在 Nebula.Board4.Services 项目对应的模块文件夹中创建业务逻辑接口的实现类。

4.3.1 业务逻辑层基类介绍

所有业务逻辑层实现类必须继承 ServiceSupport 类(位于 Mercurius.Sparrow.Services 项目的 Support 目录下), 该类封装了服务实现类的通用功能以简化业务逻辑的实现。

ServiceSupport 类的主要功能如下:

- 1、缓存支持
- 2、日志记录
- 3、提供 InvokeService(包括重载)和 InvokePagingService 方法集中处理缓存和日志, 利用这两个方法可以极大简化业务逻辑层方法对缓存和日志的处理。

4.3.2 业务逻辑层实现

- 1、必须继承 ServiceSupport 类

4.4 显示层

注意: 显示层的实现是基于 Asp.Net MVC5, 如果对 Asp.Net MVC5 不了解, 请参看 [《3.2 技术参考资料》](#) 提供的 Asp.Net MVC5 参考资料。

显示层基于 Asp.Net MVC5, 使用的技术都是 Asp.Net MVC5 所提供的, 无特别之处, 如对 Asp.Net MVC5 有了解的话, 无任何理解和扩展的障碍, 以下几点只需要注意一下就可以了:

- 1、所有控制器必须继承 CommonController 类。
- 2、BundleConfig 类的作用: 用于合并和压缩 Css、js 文件。