DapperMVC4基础框架文档

目录

[DapperMVC4基础框架文档 1](#_Toc477523785)

[一、 框架基础环境及技术 4](#_Toc477523786)

[二、 关于Dapper 4](#_Toc477523787)

[三、 关于Autofac 4](#_Toc477523788)

[四、 框架目录 5](#_Toc477523789)

[五、 框架展望及扩展 5](#_Toc477523790)

[六、 编码基本要求 8](#_Toc477523791)

[七、 主键生成策略 10](#_Toc477523792)

[八、 代码管理 11](#_Toc477523793)

[九、 工作分配 11](#_Toc477523794)

**前序：**

基于框架最开始的设想，首先是基于本身自己的熟悉程度选择了ASP.NET MVC+Sql Server的环境基础去进行其它技术的选择来搭建框架。首先是选择运行速度快捷且轻量级的开源Dapper.Net作为系统的ORM框架，然后是考虑到代码的易管理和扩展行选择Autofac来做控制反转解决系统各层之间的耦合问题，最后是选用log4net这个应用最广也是最稳定的插件作为日志系统。

1. 框架基础环境及技术
2. 项目类型：ASP.NET MVC4
3. 数据库：Sql Server2008
4. ORM：Dapper.Net
5. Ioc：Autofac
6. 日志系统：log4net
7. 关于Dapper
8. 性能优越。

其实在各大网站上，我们大概都会看到这样的一个对比效果图，在超过500次类的序列化的过程中所表现的性能，我们发现dapper是第二名，

当然第一名谁也无法超越，越底层的当然久越快，同时也就越麻烦。

1. 轻量级。

核心代码只有一个SqlMapper.cs，编译完成之后只有120k。

1. 开源。

源码可以直接在github上面下载。

1. 支持多数据库。

Dapper可以在所有Ado.net Providers下工作，包括sqlite, sqlce, firebird, oracle, MySQL, PostgreSQL 和 SQL Server。

1. 简单的CURD。

参考代码。

1. 关于Autofac
   * + 1. 先理解IOC。

IOC全称是Inversion Of Control，控制反转，是一种全新的设计模式，用来解决程序间的耦合问题。

* + - 1. Autofac。

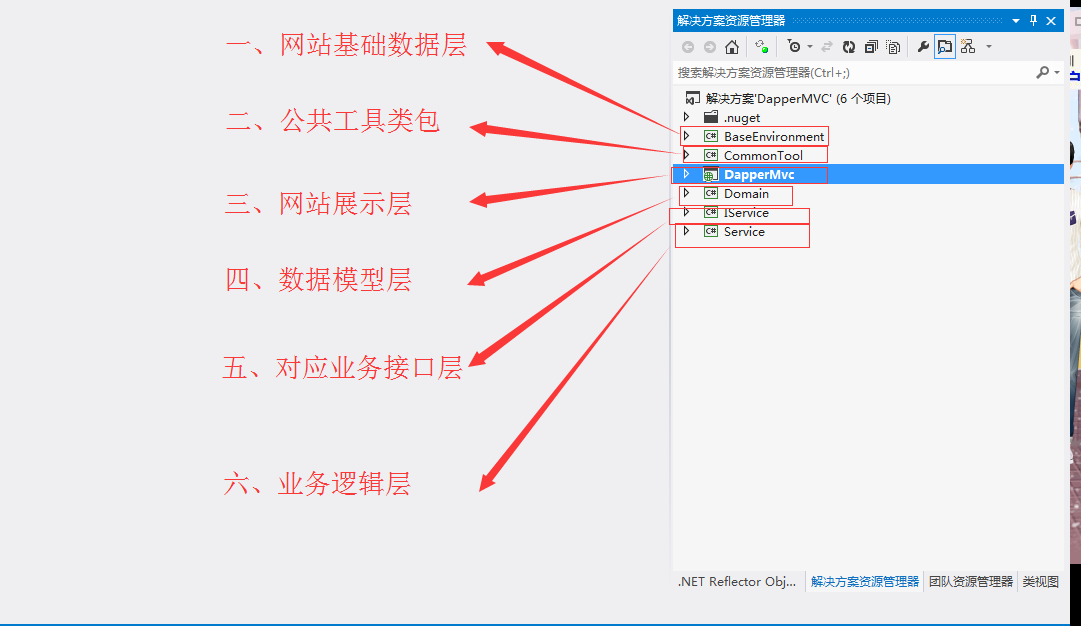
基于.net平台的IOC框架，可以通过NuGet添加，Autofac程序包里包含了所需的所有核心功能。

* + - 1. Autofac在我框架的集成。



如果需要增加新的服务类只需要添加一个新的Service和对应的IService就可以在Controller里面进行调用了，这样就满足了OCP原则，即对修改关闭，对扩展打开。

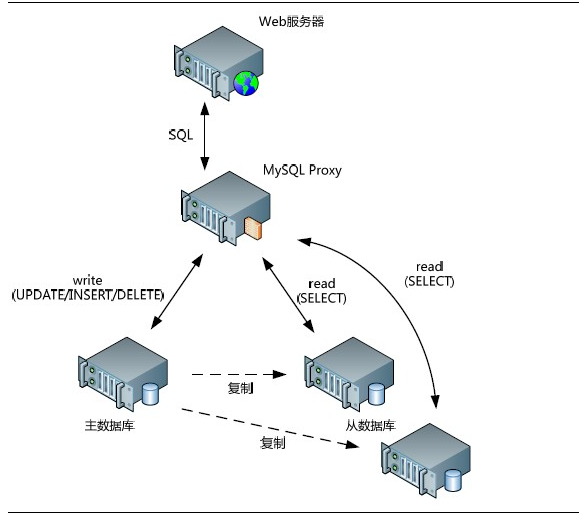
1. 框架目录



1. 框架展望及扩展
   * + 1. 数据库：

* 多数据库服务器

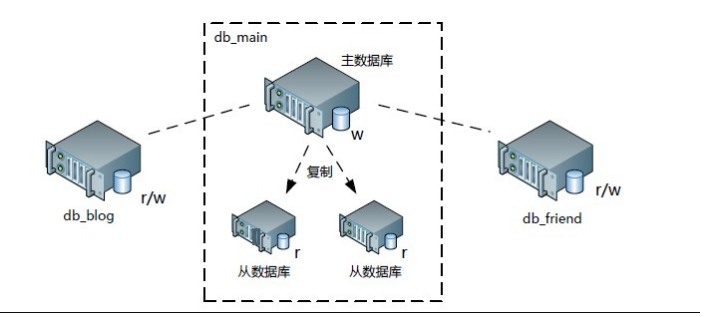
读写分离，一般程序对数据库的读操作是要远远超过写操作的，那么我们便可以进行读写分库，原理是用一台主服务器来进行写操作，多台从服务器同步主服务器数据库进行读操作。



* 垂直分区

由于系统的越来越庞大，站点的加多，写操作自然也会越来越频繁，如果写的操作占操作中的80%,那么从服务器必须花费80%去同步主服务器中的数据，只有20%的时间去处理用户的读操作了，而且如果写操作再大一些的话，写服务器已经无法承受写的压力了。那么我们便可以采用垂直分区的方法来缓解这样的压力。

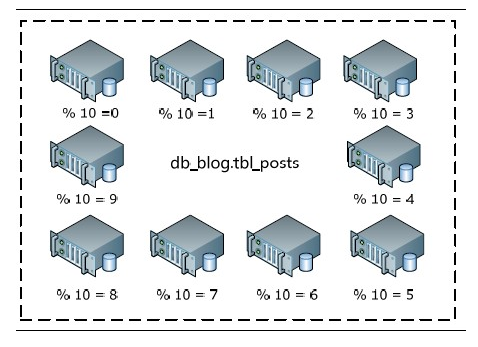
垂直分区其实就是将彼此不经常交互的数据分类分别放在不同的服务器中，最简单的方法是将不同的数据库分布到不同的服务器上，你会发现有很多的数据库之间并不存在关系，或者不需要进行join查询，那么我们就讲他们分离开。



* 水平分区

还是当写操作太频繁时，如超过某一张表的极限时我们便需要做水平分区，即将同一个数据表按一定规则分别放在不同的数据库中。

例如加入我们拥有10个服务器，我们可以根据userid%10将用户数据分别存放在不同的数据库中，这样的话我们就可以用户分别存放在这10个数据库分区中。



* 非关系型数据库

NoSQL，泛指非关系型的数据库。随着互联网[web2.0](http://baike.baidu.com/view/733.htm" \t "_blank)网站的兴起，传统的关系数据库在应付web2.0网站，特别是超大规模和高并发的[SNS](http://baike.baidu.com/subview/8258/5896174.htm" \t "_blank)类型的web2.0纯[动态网](http://baike.baidu.com/view/528572.htm" \t "_blank)站已经显得力不从心，暴露了很多难以克服的问题，而非关系型的数据库则由于其本身的特点得到了非常迅速的发展。NoSQL数据库的产生就是为了解决大规模数据集合多重数据种类带来的挑战，尤其是大数据应用难题。

* + - 1. 缓存：
* .Net平台内置的缓存类，Cache类。
* 分布式缓存，Memcache。
  + - 1. 服务器集成，负载均衡：

负载均衡(Load Balance)是集群技术（Cluster）的一种应用。负载均衡可以将工作任务分摊到多个处理单元，从而提高并发处理能力。目前最常见的负载均衡应用是Web负载均衡。根据实现的原理不同，常见的web负载均衡技术包括：DNS轮询、IP负载均衡和CDN。其中IP负载均衡可以使用硬件设备或软件方式来实现。

1. 编码基本要求
   * + 1. 命名规约

* 【强制】 代码中的命名严禁使用拼音与英文混合的方式，更不允许直接使用中文的方式。 说明：正确的英文拼写和语法可以让阅读者易于理解，避免歧义。注意，即使纯拼音命名方式也要避免采用。

反例： DaZhePromotion [打折] / getPingfenByName() [评分] / int 某变量 = 3

正例： alibaba / taobao / youku / hangzhou 等国际通用的名称，可视同英文。

* 【强制】类名使用UpperCamelCase风格，必须遵从驼峰形式，但以下情形例外：（领域模型的相关命名）DO / BO / DTO / VO等。

正例：MarcoPolo / UserDO / XmlService / TcpUdpDeal / TaPromotion 反例：macroPolo / UserDo / XMLService / TCPUDPDeal / TAPromotion

* 【强制】方法名、参数名、成员变量、局部变量都统一使用lowerCamelCase风格，必须遵从驼峰形式。

正例： localValue / getHttpMessage() / inputUserId

* 【强制】杜绝完全不规范的缩写，避免望文不知义。 反例： AbstractClass“缩写”命名成AbsClass；condition“缩写”命名成 condi，此类随意缩写严重降低了代码的可阅读性。
* 【强制】常量命名全部大写，单词间用下划线隔开，力求语义表达完整清楚，不要嫌名字长。

正例： MAX\_STOCK\_COUNT 反例： MAX\_COUNT

* + - 1. 注释规约
* .net编译器的几种注释方式。

单行注释：//Columns += item.Name + ",";

多行注释: /\* if (NameSpace == "DapperMvc.DataModel")

{

Name = item.Name;

}\*/

折叠代码：#region 折叠代码

if (NameSpace == "DapperMvc.DataModel")

{

Name = item.Name;

}

#endregion

* 类注释-在类的开始部分注释。

     /\*  
     \* 类说明  
     \*   
     \*   
     \*/  
   public class City  
   {}

* 类属性注释-在类的属性前注释。

   //属性说明  
   public string City\_Name  
     {  
         get { return this.\_city\_name; }  
         set { this.\_city\_name = value; }  
     }

* 类方法注释-在类的方法声明前注释。

  /// <summary>  
   /// 方法说明  
   /// </summary>  
   public void Insert()  
   {}

* 代码间注释-在代码之间注释

     try  
            {  
                //<说明>  
                ListViewItem lvi = Getlvi(car);  
                this.lvCar.Items.Add(lvi);  
            }  
            catch (MessageException ex)  
            {  
                MessageBox.Show(ex.Message);  
            }

1. 主键生成策略
2. 自增长主键策略 。
   * + - 优点 ：简单，不需要程序特别处理 。
       - 缺点 ：这种方法对以后如果项目移植到其它数据库上改动会比较大，oracle、    db2采用Sequence，mysql、sqlServer又采用自增长，通用性不好
3. 时间戳+随机数。
   * + - 优点：实现简单，与数据库无关，移植性较好 。
       - 缺点 ：长度太长，最少也得20位，不仅占空间并且建索引的话性能会比较差点吧。
4. 每次取主键最大值+1。
   * + - 优点：主键长度可控，移植性较好。
       - 缺点：并发写可能会造成主键冲突，对并发也不太好控制。
5. 单独建一个存放主键的表维护。
   * + - 优点：实现简单，移植性较好 。
       - 缺点：需要考虑并发问题，整个系统主键生成都依赖该表，性能影响可能较大 。
6. 生成GUID。（暂时取用）
   * + - 优点：实现简单，移植性较好 。
       - 缺点：排序会被打乱，可以配合创建时间字段一起使用进行插入顺序排序。
7. 代码管理

暂时取用SVN管理。

1. 工作分配
2. 整体模块设计。
3. 前端页面设计及编写。（可以模仿知乎或是百度，也可以网上找一套好看的开源前台框架）
4. 数据库设计。
5. 编码实施。