欢迎使用 SharpSword 1.22.0.0 Framework→

#==
#　　注册/登录： http://code.taobao.org
#　　代码库SVN： http://code.taobao.org/svn/SharpSword （SVN账户密码就是注册的用户账户和密码）
#==

#快速开发框架(SharpSword)技术及功能点（服务端)

00.框架概述

框架采取“约定优于配置原则”，从实体，服务层，应用层，DTO传输对象，入参，返回值，配置参数，都进行了约定，

“避免重复代码”是我们在开发软件时所具备的最重要的思想之一。我们在开发企业WEB应用程序时都有一些类似的需求，例如：用户登录，权限验证、数据有效性验证、多语言/本地化，日志，缓存，数据持久化等等。另外我们也会运用一些最佳实践，例如分层体系结构、领域驱动设计、依赖注入等。SharpSword实现了其中一部分思想，让我们在系统开发过程中，能够有效的减少重复劳动。提供一个稳定，易于扩展的基架。

系统框架内部采用了一种完全模块化的方式进行设计，每个组件都有特定的任务和清晰的职责，彼此之间相互补充，就像我们玩的拼图一样，使整个系统变得非常扁平。这种架构的主要优势之一是便于扩展。框架中的大部分元素都是可以被移除的，并且很容易使用其他的元素代替。从而允许对框架本身的内部行为做极大程度的定制。因此，当我们框架在使用一个组件时，我们通常不会直接引用该组件，而是通过接口提供的抽象来引用。这样组件之间就没有通过刚性连接捆绑在一起。从而让组件之间极大程度的保持了相对独立性。实际上，一个组件仅仅知道另一个组件的接口，也即定义的操作契约。如果我们想扩展或想修改一个组件，只要创建实现组件接口的类，替换掉原来的组件即可。而这一些替换操作，系统框架都会帮我们完成，而我们需要做的，只要按照框架约定和设置的规则来开发即可，大大减少开发人员负担。让开发人员将重点关注到业务逻辑层面而非技术实现层面。

致谢:

框架开发过程中参考了一些优秀的开源框架

Orchard：http://www.orchardproject.net
nopCommerce：https://www.nopcommerce.com
ABP：https://aspnetboilerplate.com
ENode：https://github.com/tangxuehua/enode
SignalR：https://github.com/SignalR/SignalR
ASP.NET Core MVC ： https://github.com/aspnet/Mvc

01.ASP.NET MVC 4.0，.Net 4.5 以上

02.依赖注入(Dependency Injection)

AutoFac： https://autofac.org

03.动态代理(Dynamic Proxy)

Castle.Core： http://www.castleproject.org

Autofac.Extras.DynamicProxy： https://github.com/autofac/Autofac.Extras.DynamicProxy

http://autofac.readthedocs.io/en/latest/advanced/interceptors.html

04.ORM

默认实现为： EntityFramework 6.1.3 (https://www.nuget.org/packages/EntityFramework)

可方便切换至： NHibernate(https://www.nuget.org/packages/NHibernate)

05.动态编译(Roslyn,CodeDom)

项目中需要动态配置系统参数，需要动态生成视图，需要编译.cs文件成dll包，都需要使用动态编译。当然我们可以发挥想象，动态编译还可以做出更加有趣的事情

Roslyn： https://github.com/dotnet/roslyn

06.软删除(SoftDelete)

有时候。项目中需要采取软删除的方式来实现数据的逻辑删除。框架提供了软删除功能。让我们采取与物理删除具有同样语义的删除行为来实现软删除。

这一切系统框架自动帮我们完成。而不需要开发人员关注逻辑删除细节，数据删除时间，删除用户，无需手工处理。当我们数据查询的时候系统会自动过滤掉软删除的数据

07.实体审计(Entity Auditing)

当我们在开发基于数据库的业务系统的时候，有时候我们需要将数据审计功能带入到对应的数据表，比如：创建时间，创建用户，修改时间，修改用户，删除时间，删除用户

实体审计功能，让我们将此繁杂啰嗦的重复劳动交给框架去自动实现，我们需要做的就是让实体实现对应的接口即可。开发人员创建，修改，删除实体数据，无需管理这些信息。

操作后，框架会自动将此类信息记录到对应的数据表行里。

08.事件总线(EventBus)

基于事件的方式来解耦各个服务或者模块之间数据传输调用(框架内置了比如：异常，实体操作，接口执行等等事件，供我们调试)，增强系统扩展性，快速适应业务变化需求

我们在开发系统的时候，总会遇到这样的需求：当我们做了某某某的时候，需要我们做某某某事情，过一段时间后，业务需求方又说，我同时又要做某某事情。

比如：当顾客下订单完成后，我们需要给他发送邮件，给他发送短信，通知银行从顾客的信用卡里扣掉货款等，如果我们不抽象出一个东西出来，

那么所有的事情都会耦合在一个方法里。出现新的需求，方法代码又将更改，非常不符合开闭原则。因此我们抽象出事件总线触发事件，当我们需要处理事件的时候，

我们只要新增一个针对当前事件的订阅类即可。具有非常高的扩展性，快速实现需求。

09.插件式(PlugIn),模块化(Module)开发方式

我们将所有业务功能模板看成是此框架的一个插件，在这样的思想下，就能利用模块化，将业务模块进行拆分，然后自由组装，各模块互相通过事件总线方式相互调用和进行数据传输

10.数据统一验证（DtoValidator，Asp.NET MVC只能做到Action方法的参数验证，SharpSword实现了Services层方法的参数有效性验证）

我们再实际的业务服务类开发的时候，有些业务需要对入参进行数据校验，即：对数据的合法性进行验证，防止非法数据进入到业务系统。比如：我们需要判断一个字符串长度，数值的合法范围，时间是否合法等等，而这些东西又是非常重复枯燥的事情。手工去校验，即增加了工作量，代码量也会急剧增加，而且不易扩展和维护。系统框架从框架级别对此做了设计
只要是服务类，方法参数是复杂类型，系统框架自动完成参数校验，框架默认使用:Validator作为验证参数，同时为了摆脱默认特性方式定义验证带来的入侵性，
框架集成了第三方FluentValidation验证，以申明的方式来定义验证规则。零侵入性实现数据校验扩展。
FluentValidation：https://github.com/JeremySkinner/fluentvalidation

11.基于工作单元(UnitOfWork)设计

当我们在基于数据库业务系统开发的时候，进行数据持久化，不可避免的会碰到事务处理。如果我们直接使用原生的ADO.NET事务方式，那么将会带来极大的繁杂性，
针对开发人员来说，非常的不友好，而且每个人的使用习惯的不同，会带来非常不一致的代码，给代码的维护，升级，扩展都带来灾难。为了解决此问题，我们引入工作单元，
默认里，系统服务类的所有公开方法都是事务性的（当然我们可以关闭事务），开发人员无需手工处理事务(Transaction)，在开发的过程中开发人员不会感觉到事务的存在。
这一切系统框架都帮开发人员自动完成。业务代码里也不会出现啰嗦的事务代码，让业务方法更加关注业务，而不是关注具体的技术细节。
关于工作单元阐述：http://www.cnblogs.com/xishuai/p/3750154.html

12.简洁且统一的数据访问接口(Data Access Interface)

提供完全基于对象的数据操作，解放开发人员，更加专注于业务的实现(建议CUD操作采取对象方式，复杂Query采取自定义SQL语句方式)
完全支持自定义SQL或者存储过程(StoredProcedure)2种访问方式且将2种方式完全融合在一起，提供统一接口
数据库可灵活切换（默认实现了MSSQL,MYSQL,Oracle,PostgreSQL）
框架无缝支持集成Dapper扩展或者扩展自IDbConnection的所有第三方插件，更好的带来接口便利和实现更加优雅的代码
Dapper：https://github.com/StackExchange/dapper-dot-net

13.灵活的缓存(Cache)器设计

系统默认注册了：Memory(进程内）， HttpPreRequest(基于一次http请求) 2种缓存器，其他缓存器作为插件形式给出的。在需要使用下面缓存器只要在HOST项目引用下插件项目即可
系统默认将会为你完成所有配置，默认使用您引用的缓存器来实现缓存（当然系统里可以同时使用多个缓存器，具体使用在项目演示里有说明）。框架默认带了下面2个插件缓存器。

Redis： https://github.com/imperugo/StackExchange.Redis.Extensions
Memached： https://www.nuget.org/packages/Memcached.ClientLibrary

可以根据需要自由切换或自由扩展新的缓存器，可以方便实现统一业务实现不同的缓存方案或者同时拥有不同实现方案
开发人员在开发的时候，无需了解具体实现，只要定义ICacheManager接口属性或者采取构造函数方式注入即可

14.灵活的日志(Logger)记录器设计

系统默认实现Log4Net，TextLog，DataBaseLog，控制台监控方式（需要加载插件SharpSword.ApiMonitor项目）方式，可自由切换或自由扩展新的记录器
用户开发的时候，无需了解实现，只要构造函数定义ILogger或者ILogger<>即可（当然定义成属性亦可），系统框架默认记录全部等级的日志，建议
正式环境里我们只记录ERR级别的错误。

15.自定义的视图模板(ViewEngine)引擎

就如您看当前页面一样，就是使用我们自定义的模板视图引擎实现的，当我们需要开发页面视图模板，模块插件化，邮件模板动态生成，短信模板动态生成，数据报表生成等等功能的时候，我们将可以非常方便的实现模板的动态输出。

16.作业任务（Job/Task）模块

让我们非常轻松的实现一些简单的作业任务。可以以代码或者配置的方式实现调度（调度器，作业任务，执行线程进行拆分，分布式锁）

系统默认注册了定时预热作业，即：每个15分钟自动访问一次，不让IIS应用程序池回收

17.多语言本地化(Localization)支持

系统提供本地化支持，框架默认实现了，内嵌本地化资源配置和基于文件夹的本地化资源配置。另外如果未能满足实际需要，我们可以非常方便的进行本地化资源配置扩展。

18.定义了邮件(Email)，短信(SMS)发送接口，方便扩展和使用

系统框架定义了邮件和短信组件接口，并没有为我们具体实现，因此当我们需要使用此组件的时候，需要我们自己进行扩展和实现。

19.方便扩展的资源查找器(ResourceFinder)

在系统开发的时候，尤其是组件开发的时候，我们组件提供方需要带一些静态的资源文件，如：js,css,html,img等等，如果我们能够以一个dll的方式，将这些资源内嵌到dll里，组件提供方仅仅提供dll就可以，那么这样将极大的方便我们的部署。而且针对所有的资源我们能够提供一个更高层面的管理，读取接口。增强我们的组件可复用性。

在框架内部，我们默认实现了内嵌（Assembly Finder），本地（LocalFile Finder）2种资源查找器，当然如果觉得不够，比如：我们的资源是保存在数据库的或者保存在网络上的某个地方的，那么我们完全可以根据接口，扩展出我们的定制资源查找器。而框架提供的高层接口，完全不会变化。

20.灵活的参数(Settings)配置设计

当我们在开发模块的时候，需要配置一些全局或者模块参数，比如：当我们设计支付模块的时候，不同的支付方式会有一些不同的参数。比如：支付宝支付，我们需要在我们实现代码里传入商户编号，数据签名密钥等等。如果按照将这些文件仅仅耦合在支付代码里，比如：配置在web.config里。那么我们将极大的限制了代码的扩展。如果业务需求方需要将这些固定参数配置到XML文件或者JSON文件或者数据库里的时候，我们的做法只能去修改已经正常工作的代码，这对系统的稳定性有时候会带来灾难。

因此处于此种目的，也为了提高代码的复用性，稳定性，和扩展性。我们将参数配置模块独立出来，能让我们非常方便的将参数注入到我们期望的对象里。

系统默认实现JSON和web.config,DB(MSSQL) 3种参数配置方式，如果还不能满足我们的需求，可以扩展自定义的参数获取方式

21.方便开发的工具插件(Tools)

DTO生成器，作业任务管理器，接口访问日志管理器，API接口调试器，API接口文档生成器，系统缓存查看器，事务方法查看器，自定义SQL查询语句查看器

22.通过命令行(Command)方式管理系统

通过命令行我们可以进行系统所有操作(就好比cmd命令一样)，框架提供了命令行扩展接口，每个模块我们可以自定义不同的命令行操作，方便管理员操控系统。

23.框架时钟(Clock)

当我们在进行分布式开发，不可避免需要处理各个物理机时间同步问题。框架针对时间做了同步处理，对系统时钟(Clock)等都做了约束。让开发者无须处理复杂的分布式时钟同步问题

24.服务层方法审计(Auditing)跟踪记录

很多时候我们在开发，调试，性能调优过程中，需要知道我们执行的方法调用者是谁，执行时间，调用方法是什么，返回值，参数等。来方便我我们的开发。框架针对方法审计这块做了非常方便的扩展，只要我们按照约定让服务类实现某些空接口，框架将会自动对上述信息进行记录。而无需开发人员在方法业务代码里耦合记录代码。系统框架会自动帮我们完成我们所需的所有信息。系统默认实现了文本记录方式。我们可以非常方便实现记录到其他存储介质。记录方法执行审计，将系统框架默认是打开的，建议在正式环境里关闭审计。

系统框架默认使用文本记录，为了使用方便，同时提供了基于MSSQL,MYSQL(SharpSword.Auditing.SqlServer)以及MongoDB(SharpSword.Auditing.MongoDB)的存储实现（请参见相应的插件项目）

25.对实时通讯(Realtime)的封装

当我们再开发基于互联网的系统，有时候我们需要在系统与系统之间实时推送(Push)数据，比如：扫码登录，实时消息，系统间消息推送等等。

系统封装了SignalR，让我们在开发基于实时系统更加方便。

26.对会员（用户）消息通知（Notifications）封装

当我们在开发一个互联网应用的时候，不可避免的需要给用户发送通知消息或者会员用户与用户之间的消息传递，以及当某些资源被操作（比如：评价，删除，新增，回复等等）触发消息通知，框架封装了消息发布，消息分发器，消息存储，消息订阅使得让我们在开发的时候非常简单我方便。

27.WEBAPI模块

27.0 全新的API接口访问框架，方便对接口版本快速开发，接口版本快速迭代，接口管理，接口授权等

27.1 基于缓存模块的扩展，让开发使用基于接口级别的缓存非常方便,可以使用配置特性或者配置文件完成配置，系统自动管理缓存键，无需写任何一行代码，

27.2 基于接口缓存都采取了二级缓存设计，防止大并发缓存穿透，每一级的缓存可以使用不同的缓存服务器（防止单一缓存服务器失效）

27.3 缓存的更新操作，都采取事件总线的方式来进行通知，这样让我们可以采取后台线程并发方式进行数据缓存更新(后台线程订阅相关事件，并行进行缓存更新)

27.4 对接口上送RequestDto和下送ResponseDto进行了框架级代码约束，防止开发人员出错。

27.5 非常方便的接口拆分，合并，扩展（通常我们可以将不同的API拆分到不同的程序集，实现接口隔离。或者我们将其他接口程序集打包提供统一接口层）

27.6 非常多的方便基于接口开发，调用的设计....。

27.7 查看接口设计开发建议可以打开文件夹：trunk\src\DevelopmentTools\WEBAPI接口规范.txt

27.8 VS接口自动创建模板：trunk\src\DevelopmentTools\Api_VS_ItemTemplates\Class.cs

28.动态WEBAPI(DynamicApi)

直接从应用层或者服务层方法映射成API，无需手工写API接口

29.API客户端调用SDK自动生成

为了统一API接口调动，方便接口调用方对接口的数据调动，系统框架针对此需求，专门定制了自动生成API客户单调动的SDK插件（C#，安卓，IOS，PHP等）

并且数据校验，数据签名已经包含在调用SDK里实现，这样让我们能够更加快速的实现项目开发，同时也无需维护多套客户端。接口完成，调用的SDK就已经生成了。

30.OAuth2.0授权

框架提供了基于OAuth2.0授权抽象，如果我们在实际项目开发中，需要使用第三方登录，我们可以依据抽象，非常方便的实现各种第三方账户登录系统

31.第三方支付抽象(Pay)

框架默认提供了基于第三方支付抽象（比如：支付宝，微信支付，财付通，银行直连等等），让我们在开发第三方支付需求的时候，非常方便的实现扩展。

32.很多方便开发的点……

#
SharpSword do easy do better.
#**