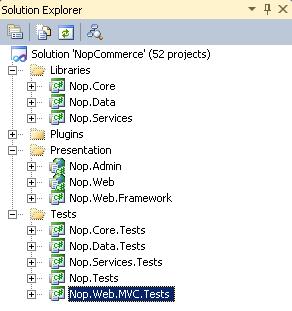
**nopCommerce的源代码结构和架构**

编写本文档是为了向程序员说明nopcommerce的解决方案结构，亦是程序员开发nopcommerce的居家必备良书。首先nopcommerce的源代码很容易拿到，它是开源的，所以你可以直接到网上下载。 在你打开VS以后项目和文件夹都会完整列出来，我们建议你在看此文档的同时也打开你的VS来浏览项目和文件。  


绝大多数的项目，目录和文件都顾名思义，你可以从名字就大概知道是做什么的。比如Nop.Plugin.Payments.PayPalStandard这个我都不用看项目代码就能猜到做什么的。  
**\Libraries\Nop.Core**  
Nop.Core项目包含nopcommerce的一系列核心类如缓存，事件，辅助类和业务对象(如订单和客户实体类)  
**\Libraries\Nop.Data**  
Nop.Data项目包含一系列的数据访问类和方法以从数据库或其他数据媒介读取和保存数据。它也有助于把数据访问逻辑和你的业务对象分离。 nopcommerce使用Entity Framework (EF) Code-First方法，允许你在nopcommerce代码中定义实体 (所有的核心实体类都在Nop.Core中定义)，再让EF生成数据库，这就是为什么会叫Code-First。你接下来可以用LINQ来查询对象，它自己会把查询转换为SQL语句并在数据库里执行。nopcommerce拥有牛B的API让你完全定制持久映射，你可以在这儿和这儿找到Code-First的资料。  
**\Libraries\Nop.Services**  
此项目包含一系列的核心服务，业务逻辑，验证，如果有数据的话还有数据的计算方法，也就是传说中的业务访问层（BAL）

\Plugins\ 文件夹中的那些项目

Plugins是VS的解决方案文件夹，硬盘中它是在你解决方案的根目录下。由于项目在编译时的输入路径是”..\..\Presentation\Nop.Web\Plugins\{Group}.{Name}\”，这样插件的DLL会自动地放到 \Presentation\Nop.Web\Plugins\文件夹中，用来放置已部署插件。这样也能让插件包含静态文件比如CSS或JS，就不用在项目之间拷贝这些文件了。

\Presentation\Nop.Admin

Nop.Admin是一MVC项目，如果你还从没用过ASP.NET MVC，请猛击[这儿](http://www.asp.net/mvc/" \o "nopcommerce, MVC" \t "_blank)有更多信息。可能你已经猜到这是表示层中的管理后台，你可以在 \Presentation\Nop.Web\Administration文件夹中找到它，此项目不能运行。

\Presentation\Nop.Web

Nop.Web也是一MVC项目，前台网店的表示层，这个才是你真正要跑起来的项目，它也是整个应用程序的起始项目。

\Presentation\Nop.Web.Framework

Nop.Web.Framework是一个表示层的类库项目，包括可以让后台和前台使用的一些共用的展示功能。

\Test\Nop.Core.Tests

Nop.Core.Tests是Nop.Core的测试项目

\Test\Nop.Data.Tests

Nop.Data.Tests是Nop.Data的测试项目

\Test\Nop.Services.Tests

Nop.Services.Tests是 Nop.Services的测试项目

\Test\Nop.Tests

Nop.Tests是一个类库，包含其它测试项目中要用的一共有类和辅助方法，此项目不包含任何测试用例

**扩展现有实体-添加新的属性**

**Updating an existing entity. How to add a new property.**

**扩展现有实体：如何添加一个新的属性？**

This tutorial covers how to add a property to the Affiliate entity that ships with the nopCommerce source code.

本教程将为代理商实体Affiliate entity添加一个属性，附带nopCom源码。

**The data model数据模型**

Entities will have two classes that are used to map records to a table. The first class defines the properties, fields, and methods consumed by the web application.

实体将有两个类用于映射记录表：第一个类定义affiliate的属性、字段和方法。

**File System Location**文件位置: [Project Root]\Libraries\Nop.Core\Domain\Affiliates\Affiliate.cs

**Assembly程序**集: Nop.Core

**Solution Location解决方案中的位置**: Nop.Core.Domain.Affiliates.Affilate.cs

The second class is used to map the properties defined in the class above to their respective SQL columns. The mapping class is also responsible for mapping relationships between different SQL tables.

第二个类是将各属性分别映射到对应的SQL列，以及映射不同的SQL表之间的关系。

**File System Location**: [Project Root]\Libraries\Nop.Data\Mapping\Affiliates\AffiliateMap.cs   
**Assembly**: Nop.Data   
**Solution Location**: Nop.Data.Mapping.Affiliates.AffiliateMap.cs

Add the following property to the Affiliate class.

为Affiliate添加一个属性：

// Instance members must be virtual on data table objects like Affiliate.cs

// Virtual is required by data access frameworks so that these frameworks

// can implement more complex features like lazy loading.

public virtual string AffiliateWebSite { get; set; }

Add the following code to the constructor of the AffiliateMap class.

为AffiliateMap添加一个构造函数：

// This code maps a column in the database to the new property we created above

// This creates a nullable nvarchar with a length of 255 characters in the

// Affiliate SQL table

this.Property(m => m.AffiliateWebSite).HasMaxLength(255).IsOptional();

Because I’m all about results, at this point I would run the code, re-install the database, and verify that the column was created appropriately.

修改数据库，为Affiliate表添加列：AffiliateWebSite，允许为空，navrchar(255)。

重新编译程序

**The** **presentation model视图模型**

The presentation model is used to transport information from a controller to the view (read more at [asp.net/mvc](http://www.asp.net/mvc/" \t "blank)). Models have another purpose; defining requirements.

表示模型用于传输控制器的信息视图（参考[asp.net/mvc](http://www.asp.net/mvc/" \t "blank)）。模型的另一个目的，定义需求。

We configured our database to only store 255 characters for the AffiliateWebSite. If we try and save an AffiliateWebSite with 300 characters the application will break (or truncate the text). We want the application to protect users from failures the best we can, and our view models help enforce requirements like string length.

我们在数据库中设定AffiliateWebSite长度为255个字符，如果尝试保存300个字符的，程序将中断（或截断文本）。因此需要通过程序强制用户输入不超过255个字符，尽可能地降低出错。

**File System Location**: [Project Root]\Presentation\Nop.Web\Administration\Models\Affiliates\AffiliateModel.cs   
**Assembly**: Nop.Admin   
**Solution Location**: Nop.Admin.Models.Affiliates.AffiliateModel.cs

The validator class is used to validate the data stored inside of the model class (e.g. required fields, max length, and required ranges).

验证输入格式

**File System Location**: [Project Root]\Presentation\Nop.Web\Administration\Validators\Affiliates\AffiliateValidator.cs   
**Assembly**: Nop.Admin   
**Solution Location**: Nop.Admin.Validators.Affiliates.AffiliateValidator.cs

Add the property to our view model.

添加视图模型需要的属性：

// The NopResourceDisplayName provides the "key" used during localization

// Keep an eye out for more about localization in future blogs

[NopResourceDisplayName("Admin.Affiliates.Fields.AffiliateWebSite")]

public string AffiliateWebSite { get; set; }

The requirements code will be added in the constructor of the validator.

//I think this code can speak for itself

RuleFor(m => m.AffiliateWebSite).Length(0, 255);

**The view**

**File System Location**: [Project Root]\Presentation\Nop.Web\Administration\Views\Affiliates\ \_CreateOrUpdate.cshtml   
**Assembly**: Nop.Admin   
**Solution Location**: Nop.Admin.Views.Affiliates.\_CreateOrUpdate.cshtml

Views contain the html for displaying model data. Place this html under the "active" section.

在视图中添加一行：

<tr>

   < td class="adminTitle">

        @Html.NopLabelFor(model => model.AffiliateWebSite):

   < /td>

   < td class="adminData">

        @Html.EditorFor(model => model.AffiliateWebSite)

        @Html.ValidationMessageFor(model => model.Active)

   < /td>

</tr>

**The controller**

In this case the controller is responsible for mapping the domain data model to our view model and vice versa. The reason I choose the affiliate model to update is because of the simplicity. I want this to be an introduction to the nopCommerce platform and I would like to keep it as simple as possible.

在这种情况下，控制器负责域数据模型映射到视图模型，反之亦然。这里之所以选择“代理商”模型来更新是因为他比较简单。以便尽可能简单地为大家介绍如何扩展现有实体属性。

**File System Location**: [Project Root]\Presentation\Nop.Web\Administration\Controllerss\AffiliateController.cs   
**Assembly**: Nop.Admin   
**Solution Location**: Nop.Admin.Controllers.AffiliateController.cs

We're going to make three updates to the AffiliateController class.

* Data Model -> View Model
* Create View Model -> Data Model
* Edit View Model -> Data Model

Normally I would write tests for the following code and verify that model mapping is working correctly, but I'll skip unit testing to keep it simple.

我们将要进行三次更新AffiliateController类。

数据模型 - 视图模型

创建视图模型 - >数据模型

编辑视图模型 - >数据模型

通常情况下，我会写下面的代码测试和验证模型的映射正常工作，但我会跳过单元测试，以保持它的简单。

In the method PrepareAffiliateModel add the following code below the model.Active = affiliate.Active:

找到Private void PrepareAffiliateModel方法，在model.Active = affiliate.Active后中添加代码：

// Data Model -> Ceate/Edit View Model

model.AffiliateWebSite = affiliate.AffiliateWebSite;

In the public ActionResult Create(AffiliateModel model, bool continueEditing) method add the following code below affiliate.Active = model.Active:

找到public ActionResult Create(AffiliateModel model, bool continueEditing)方法,在affiliate.Active = model.Active后添加代码：

// Create View Model -> Data Model

affiliate.AffiliateWebSite = model.AffiliateWebSite;

A similar change is required in public ActionResult Edit(AffiliateModel model, bool continueEditing):

最后，在public ActionResult Edit(AffiliateModel model, bool continueEditing)方法中添加以下代码：

// Edit View Model -> Data Model

affiliate.AffiliateWebSite = model.AffiliateWebSite;

**Troubleshooting**

* Recreate the database. Either your own custom SQL script or use the nopCommerce installer.
* Stop the development web server between schema changes.
* Post a detailed comment on [our forums](http://www.nopcommerce.com/boards/).

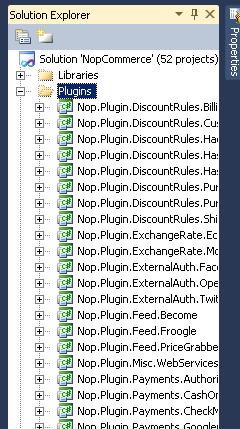
**如何编写nopCommerce插件**

插件（Plug-in，又叫addin、add-in、addon或add-on）是一种电脑程序，通过和应用程序的互动，用来替应用程序增加一些所需要的特定的功能。(Wikipedia)

插件用来扩展nopCommerce的功能，nopcommcer有多种类型的插件。比如支付方式中的paypal，税率供应商，配送计算方式（UPS,USP,Fedex），小部件（live chat功能）等等。nopCommerce本身也自带了很多不同的插件。你可以在官网上搜索是否已经有人上传了满足你需要的插件。如果没有，哥这就手把手带你编写一个出来。

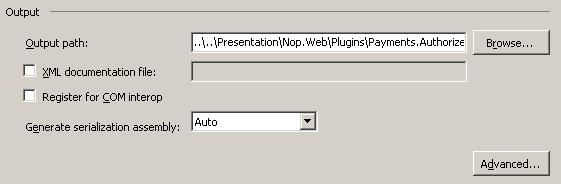
插件结构，所用文件，所在位置

1.你第一件事就是要在解决方案中新建一个“类库”项目。最好的办法是把插件都放在解决方案根目录（不过小心不要和Nop.Web下边的plugins目录搞混了，那儿是放已布置插件的），而且最好把插件也都放在解决方案目录的plugin目录中（关于更多解决方案文件夹的信息，请[猛击此处](http://msdn.microsoft.com/en-us/library/sx2027y2.aspx)）



最好以这种方法来命名：”Nop.Plugin.{Group}.{Name}”。{Group}是你插件的分类（比如支付），{Name}是你的插件名（比如”AuthorizeNet”），那么Authorize.NET的支付插件就会有这样的名字：Nop.Plugin.Payments.AuthorizeNet。

2.一旦建立了插件项目，把输入路径改为”..\..\Presentation\Nop.Web\Plugins\{Group}.{Name}\”，比如 Authorize.NET 支付插件就会有这样的输入路径: “..\..\Presentation\Nop.Web\Plugins\Payments.AuthorizeNet\”。搞定以后，对应的插件DLL就会被拷贝到 \Presentation\Nop.Web\Plugins\ 文件夹，nopCommerce内核会搜索此文件夹。



a.在项目菜单，点击属性  
b.选择生成选项卡  
c.点击输入路径旁边的浏览按钮选择一个输入目录

你要在debug和release模式下都要做此步骤。

3.下一步你就要为你的每一个插件建立一个Description.txt，此文件包含描述插件的信息。你可以从其它插件目录中拷出来。比如Authorize.NET支付插件的Description.txt就有如下内容：

Group: Payment methods

FriendlyName: Credit Card

SystemName: Payments.AuthorizeNet

Version: 1.00

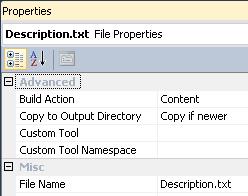
SupportedVersions: 2.30

Author: nopCommerce team

DisplayOrder: 1

FileName: Nop.Plugin.Payments.AuthorizeNet.dll

其实所有的信息你都能看懂，不过有一些注意事项。**SystemName**必须唯一。Version字段是你插件的版本号，你可以将它设置为你喜欢的任何值。**SupportedVersions**可以包含一个由逗号分隔的（确保nopCommerce当前版本包含在此列表中，否则此插件没戏）支持版本清单。**FileName**是用这个格式：Nop.Plugin.{Group}.{Name}.dll（是你插件的assembly文件名）。要确保此文件的“拷贝到输入目录”属性是“Copy if newer”

  
4.所需的最后一个步骤是创建一个类实现IPlugin接口（Nop.Core.Plugins命名空间）。nopCommerce有BasePlugin类已经实现了一些IPlugin方法，这样你就不用苦逼地再写一遍。nopCommerce还提供一些从IPlugin派生特定的接口。例如，俺们有“IPaymentMethod”接口，用于创建新的付款插件，它包含了一些特定的用于付款的方法如ProcessPayment（）或GetAdditionalHandlingFee（）。nopCommerce目前有以下特定的插件接口：  
IExternalAuthenticationMethod. 用来建立外部认证方法如 Facebook, Twitter, OpenID, etc.  
IWidgetPlugin. 让你可以创建小部件，小部件在你网站的某些地方出现，如左边的Live chat框  
IExchangeRateProvider. 用于获得货币汇率.  
IDiscountRequirementRule. 允许你创建新的折扣规则比如”帐单寄到的国家必须是……“  
ISMSProvider. 短信提供商，让你可以在下单时收到短信通知。  
IPaymentMethod. 用于处理支付流程的插件。  
IPromotionFeed. 这些插件用于向Froogle或PriceGrabber提供产品信息  
IShippingRateComputationMethod..这些插件是用于获取可用的配送方法和正确的运费。例如，UPS，UPS，FEDEX等。  
ITaxProvider. 税率提供商用于获取税率。

**处理请求。控制器，模型和视图。**

现在你可以在Admin area > Configuration > Plugins看到我们的插件了，不过正如你所想，这个杯具的插件啥都不能做，甚至连个配置的界面都没有。现在让我们来创建一个配置页面。我们现在需要做的是创建一个控制器，模型和视图。

1. MVC控制器负责响应对一个ASP.NET MVC网站的请求。每个浏览器请求被映射到一个特定的控制器。
2. 一个视图包含被发送到浏览器的HTML标记和内容。视图是相当于一个ASP.NET MVC应用程序的页面。
3. 一个MVC模型包含视图或控制器以外的所有应用程序逻辑。

关于MVC模式在[这里](http://www.asp.net/mvc/tutorials/understanding-models-views-and-controllers-cs" \o "MVC)你可以找到更多的信息。

那么，我们可以开工了：

创建模型。新插件中加入一个Models文件夹，然后按你需要新加入一个模型类。  
创建视图。在插件项目中新加一个Views文件夹，再在里边添加一个{Name}文件夹，此处{Name}是指你的插件名。然后再添加一个Configure.cshtml文件。很重要的一点：此视图应该要注明是嵌入资源。  
创建控制器。在插件项目中新加一个controller文件夹，再新加一个控制器类。最好的命名办法是像{Group}{Name}Controller.cs这样如PaymentAuthorizeNetController。再好好地命名一个action方法用于配置。哥叫它“Configure”。准备一个模型类并将其传给这个视图：Nop.Plugin.{Group}.{Name}.Views. {Group}{Name}.Configure，即那个嵌入视图。比如你在Authorize.NET支付插件中的PaymentAuthorizeNetController实现你就会比较清楚。

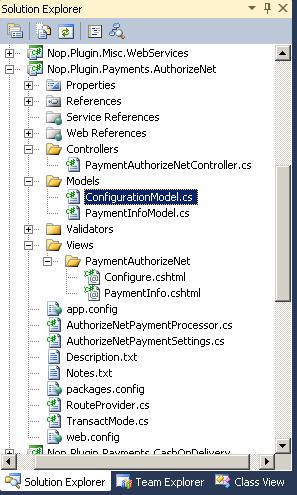
提示一：从其它插件项目中拷贝web.config到你项目里来，这样在做视图的时候有智能感知（老丁：啊？真的么？这和拷文件有什么关系？）。智能感知即微软的自动完成亮点。

提示二：搞定以上步骤最简单的办法是直接把其它插件项目拷贝过来，然后文件和文件夹改名。

提示三：如果你想限制后台（店主）控制器的一些action方法，只用在方法上加[AdminAuthorize]属性即可。

提示四：接下来要确保所有第三方的程序集引用的“拷贝到本地”属性设为false，这样可以减小部署包的大小。

比如Authorize.NET插件的项目结构会如下图：



路由

现在我们要为插件注册相应的路由。ASP.NET路由用于把浏览器发送的请求映射成MVC控制器相应的action方法，接下来的步骤你会读到很多详细关于路由的信息。

1. 新建如下文件：RouteProvider.cs，它会向nopcommerce告知关于插件路由信息。比如下边的RouteProvider添加了一个新的路由，可以通过浏览器路径http://www.yourStore.com/Plugins/PaymentAuthorizeNet/Configure来访问：

public partial class RouteProvider : IRouteProvider

{

public void RegisterRoutes(RouteCollection routes)

{

routes.MapRoute("Plugin.Payments.AuthorizeNet.Configure",

"Plugins/PaymentAuthorizeNet/Configure",

new { controller = "PaymentAuthorizeNet", action = "Configure" },

new[] { "Nop.Plugin.Payments.AuthorizeNet.Controllers" }

);

}

public int Priority

{

get

{

return 0;

}

}

}

2. 一些特写的插件接口（像上边讲的）和“IMiscPlugin”接口有一个方法“GetConfigurationRoute”。它应该向控制器返回一个用于插件后台配置的路由。实现你插件的“GetConfigurationRoute”方法，可以告知nopCommerce你插件的后台配置路由是什么。如果你插件不需要后台配置，那么此方法将返回NULL，比如下边这样：

public void GetConfigurationRoute(out string actionName,  
out string controllerName,  
out RouteValueDictionary routeValues)  
{  
actionName = “Configure”;  
controllerName = “PaymentAuthorizeNet”;  
routeValues = new RouteValueDictionary()  
{  
{ “Namespaces”, “Nop.Plugin.Payments.AuthorizeNet.Controllers” },  
{ “area”, null }  
};  
}

只要你有这个配置方法，插件安装以后你就能在Admin > Configuration > Plugins找到一个配置链接。

处理“安装”和“卸载”方法

这是可选步骤。一些插件需要有一定的安装逻辑，比如插件要添加一些本地资源数据。在你的IPlugin实现中（大多数情况下是直接从BasePlugin类继承下来），重载以下方法：

1. Install:在插件安装时会调用此方法，你可以在此初始化任何设置，添加新的本地资源数据或添加新的数据库表（如果需要的话）

2.Uninstall:在卸载插件时会调用此方法。

**重要说明**：如果你重载这些方法，不要隐藏基类的实现。比如重载”Install”的时候要记得调用base.Install(),Authorize.NET的install方法如下：

public override void Install()  
{  
var settings = new AuthorizeNetPaymentSettings()  
{  
UseSandbox = true,  
TransactMode = TransactMode.Authorize,  
TransactionKey = “123″,  
LoginId = “456″  
};  
\_settingService.SaveSetting(settings);

base.Install();  
}

提示：已安装的插件列表可以在\App\_Data\InstalledPlugins.txt找到，这个列表是在安装的时候创建的。

升级nopCommerce可能会让插件挂掉

一些插件可能在新版本的nopCommerce中挂掉无法工作。如果在升级后有问题，请删除插件再到nopCommerce官网看看是否有些版本的插件下载。大部分的插件作者都会把他们的插件升级到新的版本，不过少数插件并不会跟随着一起升级从而不再支持新版本。不过大多数情况下，你可以打开相应的 Description.txt文件并编辑**SupportedVersions**字段。

小结

希望此文能让你开始nopCommerce的插件之旅并搞个出类拔萃的插件。

**nopCommerce常见开发问题汇总**

以下列出的是程序猿攻城湿在开发nopCommerce的时候经常提出的问题。它们也表现出nopCommerce团队对一些架构的选择。

文章说明：文章来源自[我的博客](http://www.cnblogs.com/dingsea/admin/www.dingsea.com)，于原文之理解而翻译，并非出版社那种按字词翻译风格。欢迎各位提出意见，也欢迎各位转载不过务必注明本文原址。更多 nopcommerce的文章请关注[http://www.dingsea.com/?tag=nopcommerce](http://www.dingsea.com/?tag=nopcommerce" \t "_blank)，或者加入我们的QQ群 101675096

**有哪些要求？**

NopCommerce的技术和系统要求可以在[这儿](http://www.nopcommerce.com/technologysystemrequirements.aspx)找到（英文）

**程序猿如何向nopCommerce项目贡献代码？**

NopCommerce代码托管在codeplex Mercurial代码库，用户[点此访问](http://nopcommerce.codeplex.com/SourceContro/list/changesets)。借此公共代码库，用户可找到将要发布的修改和以前的设计决策。如果想知道codeplex Mercurial对版本树的支持请在[这儿](http://blogs.msdn.com/b/codeplex/archive/2010/03/05/codeplex-mercurial-support-for-forks.aspx)和[这儿](http://tekpub.com/view/dotnet-oss/7)找更多信息。程序猿可以很容易地在我们的[扩展页面](http://www.nopcommerce.com/extensions.aspx)上传插件和语言包并分享给他人。要上传扩展，请在浏览器中访问[我的帐号](http://www.nopcommerce.com/account.aspx)，选择“Your contributions and extensions”选项卡，然后点击“Upload a new extension”按钮。

**我如何报告一个缺陷？**

nopCommerce使用Codeplex作为官方缺陷跟踪系统，如果发现一个缺陷，可通过在Codeplex创建一个任务来报告给nopCommerce团队。程序猿或用户也可以在我们的[Bug Reports](http://www.nopcommerce.com/boards/f/17/bug-reports.aspx)论坛版块发帖子来告知新发现的缺陷。如果你的缺陷已经被记录当然最好，正因为此，验证那些没有被记录的缺陷更为重要（比较拗口）。报告重复的缺陷会分心而且让我们在新的开发和改缺陷上时间更少。

**nopCommerce的数据访问层**

Nop.Data项目包含一系列的类和函数来读取和写入数据库或是其它数据存储介质。Nop.Data项目有助于将数据访问的逻辑从你的业务对象中 分离出来。NopCommerce使用Entity Framework (EF) Code-First，Code-First允许程序员在源代码中定义实体（所有核心实体都在Nop.Core项目中定义），然后使用EF来生成基于C# 类的数据库，这就是为何被称为Code-First。你可以用LINQ来查询你的对象，它会悄悄地把代码转化为SQL语句并在数据库执行。 Nopcommerce有流利的API用于完全定制化的持久映射。如果想了解更多Code-First请访问[这儿](http://weblogs.asp.net/scottgu/archive/2010/07/16/code-first-development-with-entity-framework-4.aspx)和[这儿](http://blogs.msdn.com/b/adonet/archive/2011/03/15/ef-4-1-code-first-walkthrough.aspx)

**控制反转和依赖注入**

控制反转和依赖注入是两个密不可分的方法用来分离你应用程序中的依赖性。控制反转[Inversion of Control (IoC)](http://en.wikipedia.org/wiki/Inversion_of_control) 意味着一个对象不会新创建一个对象并依赖着它来完成工作。相反，它们从外部获取它们想要的对象。依赖注入[Dependency Injection (DI)](http://en.wikipedia.org/wiki/Dependency_injection" \t "blank) 意味着在没有对象的干预下，一般通过能传入构造参数和一系列属性的框架组件完成。马丁虎老二（Martin Fowler）写过一篇关于依赖注入和控制反转的牛B文章，我就不要再抄到这儿了，你可以在[这儿](http://martinfowler.com/articles/injection.html)找到。NopCommerce使用[Autofac](http://code.google.com/p/autofac/)类库作为IOC容器。只要你写了一个服务和此服务已实现的适当接口，你应该在任何实现了**IDependencyRegistrar**接口(Nop.Core.Infrastructure.DependencyManagement 命名空间).的类里注册它。比如所有nopCommerce的核心服务都在Nop.Web.Framework类库的DependencyRegistrar类中已注册。

public class DependencyRegistrar : IDependencyRegistrar

{

public virtual void Register(ContainerBuilder builder, ITypeFinder typeFinder)

{

builder.Register(c => c.Resolve().Request)                 .As()                 .InstancePerHttpRequest();             builder.Register(c => c.Resolve().Response)                 .As()                 .InstancePerHttpRequest();

你想创建多少依赖注册类都可以。每一个类实现了**IDependencyRegistrar**接口的类都有一个**Order**属性，可以用它来替换一个现有的依赖。要覆盖nopcommerce的依赖，设置order属性为大于0。Nopcommerce会对依赖排序，并按顺序运行，数字越大你的对象越迟被注册。

**我如何注册新的路由（路由？我觉得还是用routes比较好呢？）**

ASP.NET路由主要用于接受进来的浏览器请求并把它映射到具体的MVC控制器action上。[在此](http://www.asp.net/mvc/tutorials/asp-net-mvc-routing-overview-cs)有更多信息。Nopcommerce有一个叫**IRouteProvider**的接口，用于在应用程序开始时注册路由。所有的核心路由都在Nop.Web项目中的**RouteProvider**注册。

public partial class RouteProvider : IRouteProvider

{

public void RegisterRoutes(RouteCollection routes)

{

//home page

routes.MapLocalizedRoute(“HomePage”,

“”,

new { controller = “Home”, action = “Index”},

new[] { “Nop.Web.Controllers” });

你想创建多少RouteProvider都可以。比如，如果你的插件有定制路由，需要注册，于是你可以创建一个实现IRouteProvider接口的新类，再根据插件具体注册路由。

**数据校验**

数据校验是一个用以保证程序操作干净，正确和有用数据的流程。很多.NET程序猿使用Data Annotation Validators，不过nopCommerce用的是[Fluent Validation](http://fluentvalidation.codeplex.com/" \t "blank)， 一个有着文艺青年般的接口和lambda表达式构成的.NET的小型验证库，用以生成符合你业务需求的校验规则 。在nopCommerce中你必须要通过2步来添加一个校验到一些模型中：1.创建一个继承自AbstractValidator的类并把所有必须的验 证逻辑都放入其中，看下边这些应该有所启发：

public class AddressValidator : AbstractValidator     {         public AddressValidator(ILocalizationService localizationService)         {             RuleFor(x => x.FirstName)                 .NotEmpty()                 .WithMessage(localizationService.GetResource(“Address.Fields.FirstName.Required”))                 .When(x => !x.FirstNameDisabled);

2.给你的模型类加上ValidatorAttribute属性，比如下边代码：

[Validator(typeof(AddressValidator))]

public class AddressModel : BaseNopEntityModel

{  
当一个视图模型被提交到控制器，ASP.NET会执行相应的校验。

**计划任务**

有了计划任务，你可以安排一个任务在指定的时期里，在后台运行。比如nopCommerce会定时地发送队列中的email。任务是在ASP.NET线程池中由单独的线程执行。创建一个新的任务有如下基本步骤：

1. 定义一个有**ITask**接口的类，它只有一个无参数的方法：**Execute**。在任务要执行的时候，这个方法会被调用，你懂的。
2. 为了设定一个计划任务，程序员必须在数据库相应的表中添加一个**ScheduleTask**记录。你可以使用**IScheduleTaskService**来添加记录

**事件暴露和处理**

事件是把消息广播给感兴趣的部分。事件是由数据驱动的如添加，更新和删除数据。NopCommerce允许程序员“监听”他们感兴趣的事件。程序员要想玩转事件基本上有如下两条路走，一个程序员要么发布某个事件让其它人来用，要么用别的程序员编好并发布的事件。

1. 程序员为了发布一个事件，必须先取得一个**IEventPublisher**实例再使用相应的数据一起调用**Publish**方法。
2. 程序员要监听一个事件，他必须实现一个新的**IConsumer**泛型接口，一旦有人使用这个事件，nopCommerce会用反射来寻找并注册这个事件的实现。

**设置API**

正如其它网站平台，nopCommerce也有例如“网店名称”或“启用单页购买”这类设置，在nopCommerce中有两种办法来管理设置。

你可以用实现**ISettingService**接口的方法**SetSetting**和**GetSettingByKey**来加载和保存单个设置。而在nopCommerce中最牛B的处理设置的办法是创建一个**ISettingService**接口的新实现。每个设置将变为C#属性，程序员在需要时应该使用setting类构建函数注入设置。以下是setting类的示例代码。

/\* 小结：本文是按老丁于原文之理解而翻译，并非出版社那种按字词翻译风格。欢迎各位提出意见，也欢迎各位转载不过务必注明本文原址。更多 nopcommerce的文章请关注http://www.dingsea.com/?tag=nopcommerce，或者加入我们的QQ群 101675096讨论。  
\*/  
public class MediaSettings : ISettings

{

public int AvatarPictureSize { get; set; }

public int ProductThumbPictureSize { get; set; }

public int ProductDetailsPictureSize { get; set; }

public int ProductThumbPictureSizeOnProductDetailsPage { get; set; }

public int ProductVariantPictureSize { get; set; }

public int CategoryThumbPictureSize { get; set; }

public int ManufacturerThumbPictureSize { get; set; }

public int CartThumbPictureSize { get; set; }

public bool DefaultPictureZoomEnabled { get; set; }

public int MaximumImageSize { get; set; }

}