

2015-3-31

MAGICODES.NET 框架说明文档

V1

李文强

MAGICODES.NET 团队

目录

| | | |
|-------|------------------------------------|----|
| 1 | 关于 Magicodes.NET | 6 |
| 2 | 起步 | 6 |
| 2.1 | 下载 | 6 |
| 2.1.1 | Nuget 获取 | 6 |
| 2.1.2 | 源码下载 | 7 |
| 2.2 | 包含的内容 | 7 |
| 2.2.1 | Magicodes.Web | 8 |
| 2.2.2 | Plus 解决方案目录 | 10 |
| 2.2.3 | Models 解决方案目录 | 10 |
| 2.2.4 | T4 解决方案目录 | 11 |
| 2.2.5 | Magicodes.Core | 11 |
| 2.2.6 | Magicodes.Core.Web | 11 |
| 2.2.7 | Magicodes.Utility | 12 |
| 2.2.8 | Magicodes.Web.Interfaces | 12 |
| 2.3 | 工具与插件 | 13 |
| 2.3.1 | Nuget | 13 |
| 2.3.2 | Devart T4 Editor | 13 |
| 2.3.3 | Entity Framework Power Tools | 14 |
| 2.3.4 | Web Essentials | 14 |
| 2.4 | 开发自己的插件 | 14 |
| 2.4.1 | 选择插件类型 | 14 |
| 2.4.2 | 策略程序集 | 16 |
| 2.5 | 社区与博客 | 16 |
| 2.5.1 | 社区 | 16 |

| | | |
|-------|--|----|
| 2.5.2 | 官方博客 | 16 |
| 2.5.3 | 反馈 | 16 |
| 2.5.4 | 加入我们 | 16 |
| 2.6 | 协议 | 17 |
| 2.6.1 | 协议许可的权利 | 17 |
| 2.6.2 | 协议规定的约束和限制 | 17 |
| 2.6.3 | 有限担保和免责声明 | 17 |
| 2.6.4 | 其他 | 17 |
| 3 | 插件式架构 | 18 |
| 3.1 | 插件结构 | 18 |
| 3.2 | 插件策略 | 18 |
| 3.2.1 | 日志策略 | 19 |
| 3.2.2 | 邮件策略 | 21 |
| 3.2.3 | 支付策略 | 24 |
| 3.3 | 事件管理 | 24 |
| 3.3.1 | OnApplication_PreInitialize | 24 |
| 3.3.2 | OnApplication_InitializeComplete | 24 |
| 3.3.3 | BeginRequest | 24 |
| 3.3.4 | EndRequest | 25 |
| 3.3.5 | OnConfiguration_Config | 25 |
| 3.3.6 | OnConfiguration_AppBuilder | 25 |
| 3.4 | 配置管理 | 27 |
| 3.4.1 | 账户配置 (AccountAuthenticationInfo) | 27 |
| 3.4.2 | 后台站点配置 (AdminSiteConfigInfo) | 27 |

| | | |
|-------|--|----|
| 3.4.3 | 邮件配置 (MailConfigInfo) | 27 |
| 3.4.4 | 站点信息配置 (SiteConfigInfo) | 27 |
| 3.4.5 | 系统配置 (SystemConfigInfo) | 27 |
| 3.5 | 插件商店..... | 27 |
| 4 | 插件案例..... | 28 |
| 4.1 | 开发人员面板..... | 28 |
| 4.2 | 声明式认证 (Magicodes.Strategy.Identity) | 28 |
| 4.3 | 内容发布 (Magicodes.CMS) | 28 |
| 4.4 | 博客 (Magicodes.Blogs) | 28 |
| 5 | 移动设备支持..... | 28 |
| 6 | MVC..... | 28 |
| 6.1 | Mvc 插件..... | 28 |
| 6.1.1 | 插件 Bundle..... | 28 |
| 6.1.2 | 插件资源请求..... | 32 |
| 6.2 | MVC 门户插件..... | 33 |
| 6.2.1 | 插件配置 | 33 |
| 6.2.2 | 路由注册规则 | 34 |
| 6.2.3 | Magicodes.NET 官网 (Magicodes.Mvc.Default) | 34 |
| 6.3 | MVC 后台插件..... | 34 |
| 6.3.1 | 插件配置 | 34 |
| 6.3.2 | 路由注册规则 | 35 |
| 6.3.3 | Magicodes.NET 后台 (Magicodes.Admin) | 35 |
| 6.3.4 | 插件后台菜单配置 | 35 |

| | | |
|--------|--|----|
| 6.4 | MVC 插件..... | 36 |
| 6.4.1 | 插件配置 | 36 |
| 6.4.2 | 路由注册规则 | 37 |
| 6.4.3 | 开发人员面板插件 (Magicodes.DevelopersPanel) | 37 |
| 6.5 | 路由..... | 38 |
| 6.5.1 | 注册 MVC | 38 |
| 6.5.2 | 注册区域..... | 38 |
| 6.5.3 | 注册 WebAPI | 38 |
| 6.5.4 | 注册 OData..... | 38 |
| 7 | 安全性..... | 38 |
| 7.1 | AdminControl | 39 |
| 7.2 | AdminAttribute | 39 |
| 7.3 | 阻止 XSS (跨站脚本攻击) 攻击 | 39 |
| 7.4 | ValidateAntiforgeryToken——阻止 CSRF 攻击 | 39 |
| 7.5 | Bind——重复提交 | 39 |
| 7.6 | IsPostedFromThisSiteAttribute——防御伪造攻击 | 39 |
| 7.7 | 防止密码暴力破解..... | 39 |
| 7.8 | 异常日志..... | 39 |
| 8 | 性能分析..... | 39 |
| 9 | WebAPI..... | 39 |
| 10 | OData | 39 |
| 11 | 代码生成 (T4) | 39 |
| 11.1 | 公用库&模板..... | 40 |
| 11.1.1 | Library 目录..... | 40 |

| | | |
|--------|---|----|
| 11.1.2 | Magicodes.T4 库 | 41 |
| 11.1.3 | Templates 目录 | 41 |
| 11.2 | 辅助生成特性..... | 41 |
| 11.2.1 | T4FormGroupAttribute..... | 41 |
| 11.2.2 | T4GenerationIgnoreAttribute | 42 |
| 11.2.3 | T4OnlyVerificationAttribute | 42 |
| 11.2.4 | T4OdataGridAttribute | 42 |
| 11.2.5 | T4ReadOnlyFieldAttribute..... | 42 |
| 11.3 | 辅助类和函数..... | 42 |
| 11.3.1 | T4DataType..... | 42 |
| 11.3.2 | T4Helper | 42 |
| 11.4 | 关于生成 CRUD | 43 |
| 11.5 | 关于配置信息的生成..... | 48 |
| 12 | SignalR..... | 51 |
| 13 | 声明式认证 (ASP.NET Identity) | 51 |
| 14 | 文档协议管理器 | 51 |
| 14.1 | 插件配置文件支持..... | 52 |
| 14.2 | PDF 查看器插件 (Magicodes.PDFViewer) | 53 |
| 15 | 数据访问 | 53 |
| 15.1 | Repository 模式 | 53 |
| 15.2 | Code First | 55 |
| 15.3 | EF 基于代码的迁移..... | 55 |
| 15.3.1 | 概要 | 56 |
| 15.3.2 | 主要命令 | 56 |
| 15.3.3 | 关闭自动迁移 | 56 |

| | | |
|--------|--|----|
| 15.3.4 | 生成迁移 | 56 |
| 15.4 | CommonBusinessModelBase | 57 |
| 15.5 | API | 57 |
| 15.5.1 | 消息 (MessageDataRepositoryBase) | 57 |
| 15.5.2 | 后台菜单 (SiteAdminNavigationRepositoryBase) | 57 |
| 16 | 响应式布局 | 57 |
| 17 | 【前端】模块化 | 57 |
| 18 | 【前端】MVVM | 57 |
| 19 | 前端插件 | 57 |
| 19.1 | ODataGird | 57 |

1 关于 MAGICODES.NET

Magicodes.NET，是由 Magicode.NET 团队打造的一个前后端插件快速开发框架。目的让前后端开发更快速、简单。让开发者都能快速上手、所有设备都可以适配、所有项目都适用。

Magicodes.NET 是完全开源的。它的代码托管、开发、维护都依赖 GitHub 平台（即将迁移到 GitHub）。

2 起步

简要介绍 Magicodes.NET，以及如何下载、使用，还有插件，等等。

2.1 下载

您可以从官网（<http://www.magicodes.net/>）获取下载信息。

注意：本项目使用 VISUAL STUDIO 2013 开发，请小伙伴们自备 VISUAL STUDIO 2013。

2.1.1 Nuget 获取

您可以通过 Nuget 按需获取，以便应用于自己的项目。

注意：暂时请不要使用 NUGET 获取，因为框架还在不断迭代，并没有推送稳定的 NUGET 包。

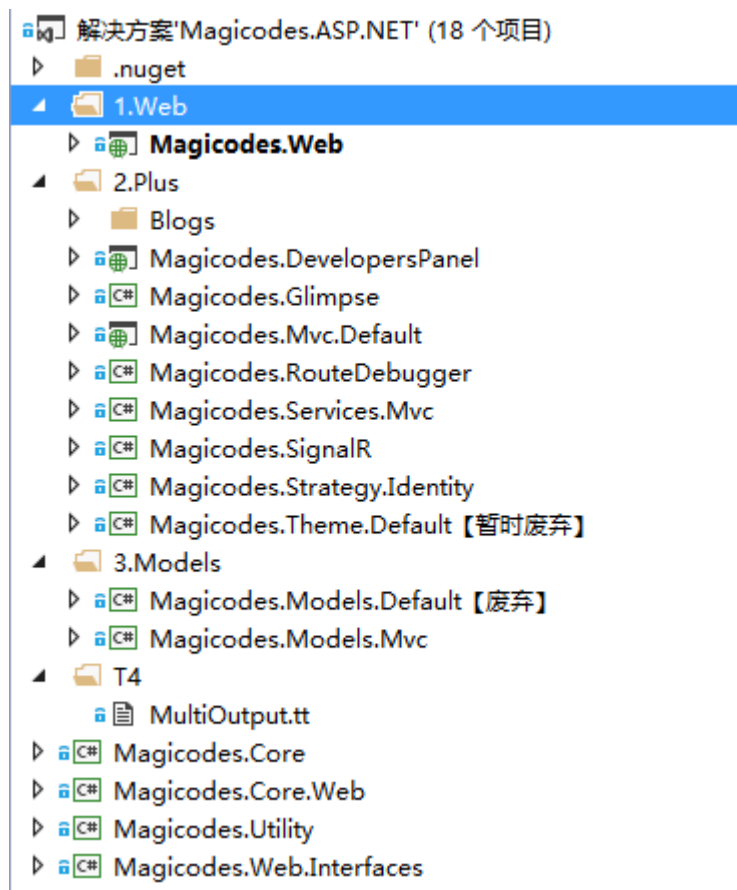
- PM> Install-Package Magicodes.Core
- PM> Install-Package Magicodes.Core.Web
- PM> Install-Package Magicodes.Utility
- PM> Install-Package Magicodes.Web.Interfaces
- PM> Install-Package Magicodes.Web.Default

2.1.2 源码下载

Magicodes.NET 源码现已托管至 GitHub，您可以从官网（<http://www.magicodes.net/>）前往访问，也可以直接访问此地址：<https://github.com/magicodes/Magicodes.NET>。

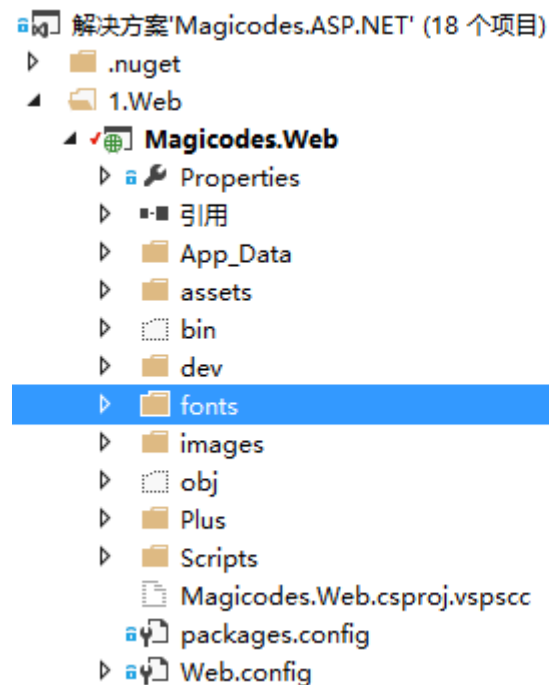
2.2 包含的内容

具体请以下载的源码为准：



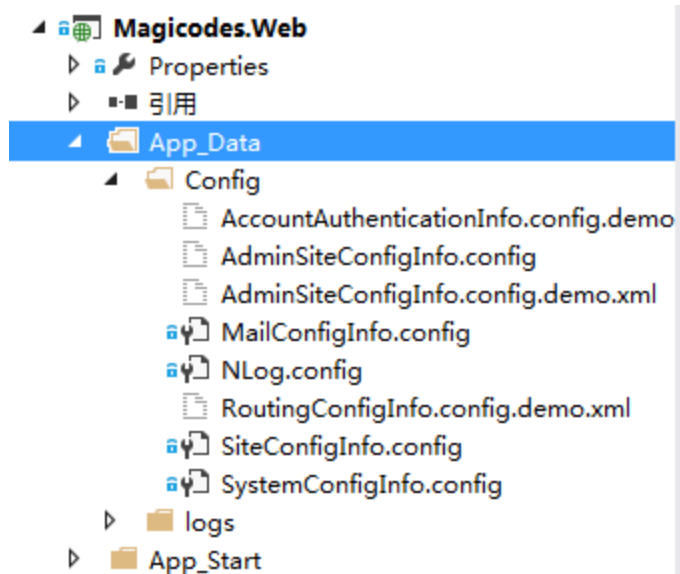
2.2.1 Magicodes.Web

Magicodes.Web 项目位于 1.Web 解决方案目录下，该项目为网站门户，是框架平台的宿主和入口。

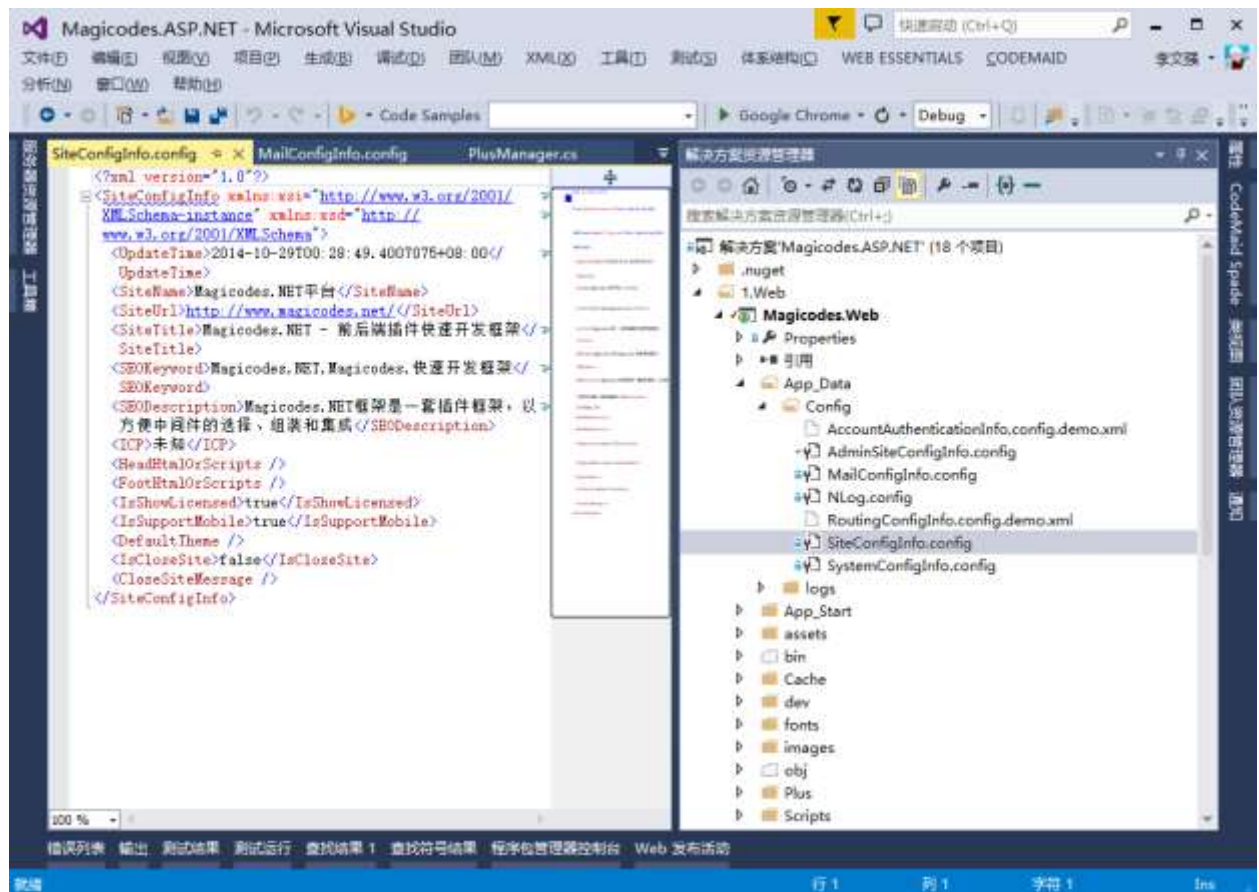


其中，

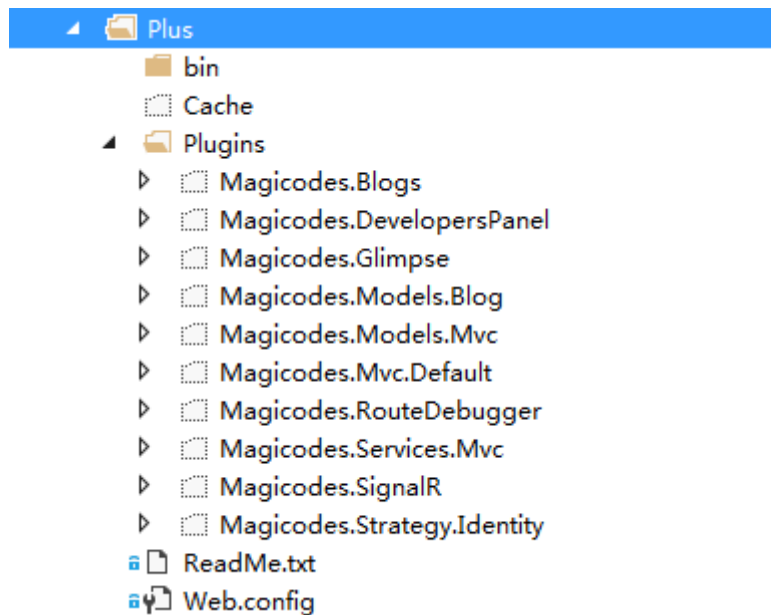
- App_Data：网站配置文件以及日志等文件存放目录



其下面的 Config 目录用于存放系统的所有配置文件。比如邮箱配置，日志配置，站点信息配置，系统配置等等。如：



- Dev：开发版本的页面资源
- Plus：插件目录。所有的插件最终都需要安装在此目录下。

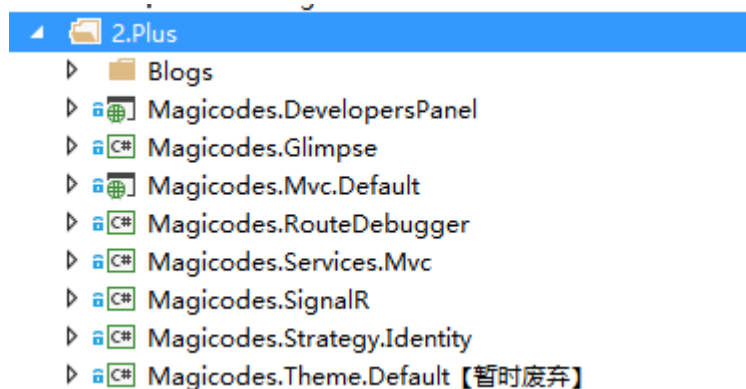


注意：当你重新编译插件项目时，会自动往此目录添加插件内容，因此一般情况，请不要更改此目录。

- Scripts：脚本资源目录。
- Web.config：全局 Web.config。

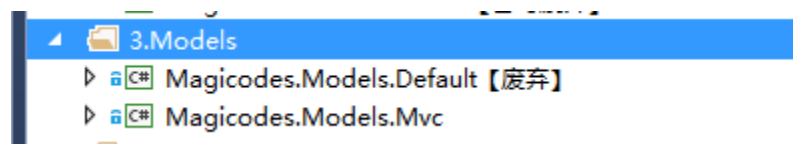
2.2.2 Plus 解决方案目录

2.Plus 解决方案目录为插件代码目录。目前有以下插件（具体以最新代码为准）：



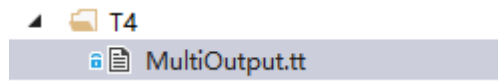
2.2.3 Models 解决方案目录

3.Models 解决方案目录为数据模型目录。一般只存放通用的数据模型库。



2.2.4 T4 解决方案目录

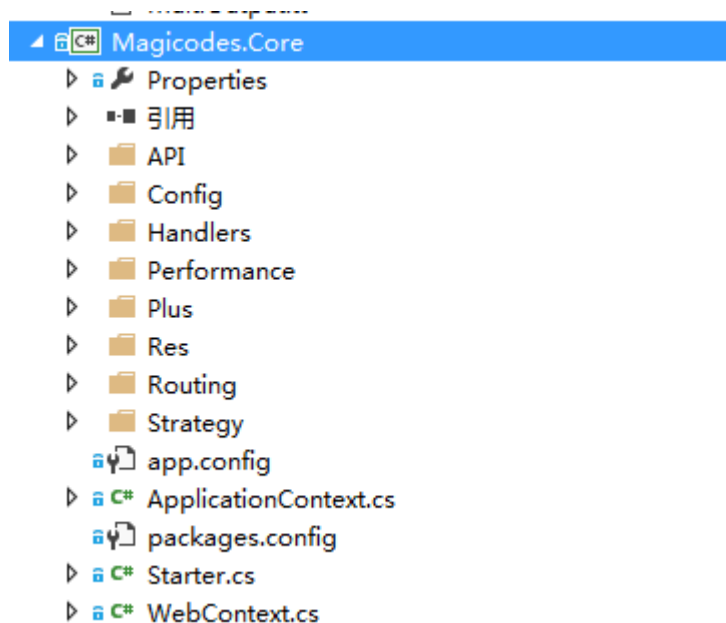
T4 解决方案目录用于存放通用的 T4 模板。如：



MultiOutput.tt 模板封装了一些函数用于输出多个代码文件并且添加到项目中。

2.2.5 Magicodes.Core

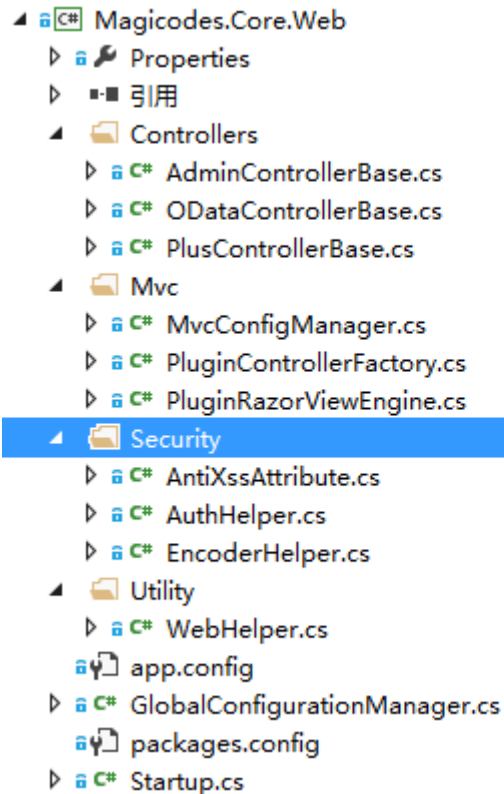
框架核心库。插件框架逻辑以及实现的核心库，一般插件不需要直接饮用：



2.2.6 Magicodes.Core.Web

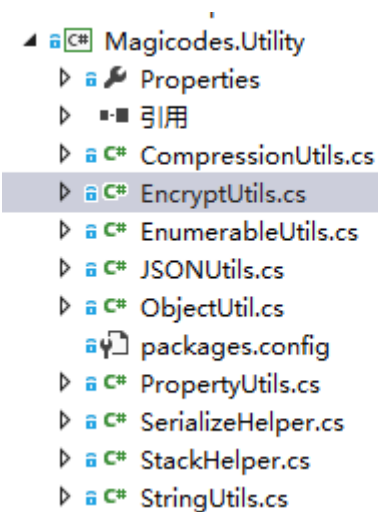
框架 Web 核心库（主要为 MVC 等机制），其定义了一些控制器的基类，Mvc

PluginRazorViewEngine，Web 安全机制以及 Web 通用的一些辅助类等等。新建 MVC 插件需要引用此项目。



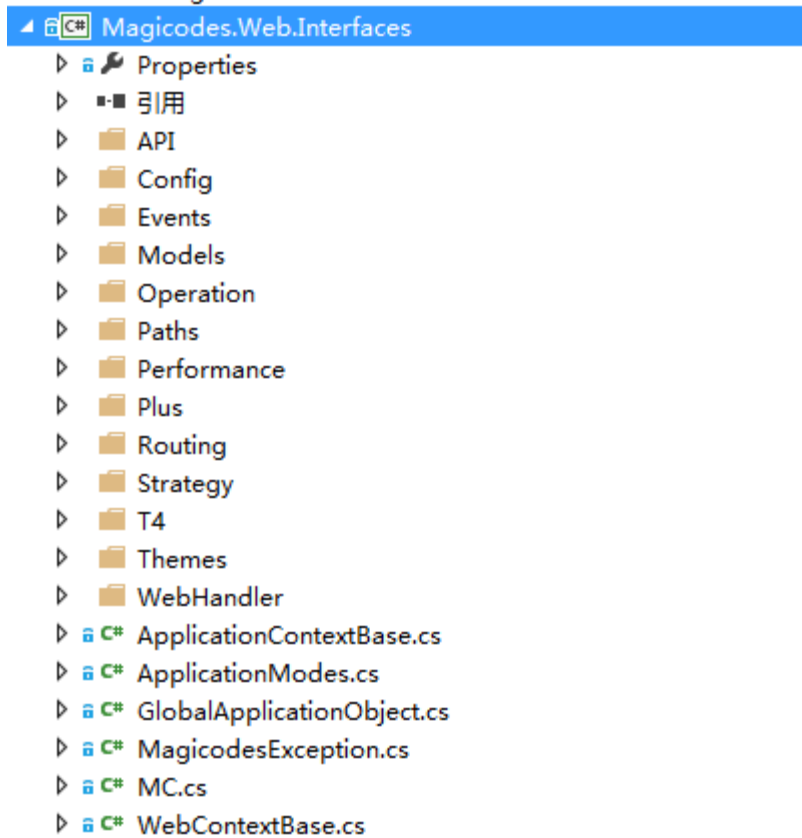
2.2.7 Magicodes.Utility

通用代码类库。比如序列化与反序列化，字符串处理等等。



2.2.8 Magicodes.Web.Interfaces

框架接口基础库。所有的插件必须引用此库。该库封装了很多事件、接口、基类等以便插件使用，是特别重要的一个类库：



2.3 工具与插件

在使用此项目时，建议您安装或者启用 Visual Studio 的以下功能或插件：

2.3.1 Nuget

必选扩展。

项目的包基本上都依赖 Nuget 进行安装，请务必安装此插件并且启用了 Nuget 包还原（以供编译时自动下载缺失的包）。

2.3.2 Devart T4 Editor

可选扩展。

用于编辑 T4 模板时提供智能提示。



2.3.3 Entity Framework Power Tools

可选扩展。

用于将已写好的数据库生成 Code First 代码。

2.3.4 Web Essentials

前端开发必选扩展。

用于压缩 Js 和 Css 以及提供前端的各种辅助功能。具体介绍请参阅官方文档。

注意：如果未安装此扩展，有可能您对框架源码中的 Js 的更改不会产生作用，因为您更改了 Js 文件后不会自动压缩，而框架中使用的是其压缩文件。

2.4 开发自己的插件

在开发插件之前，首先需要明确需要开发哪种类型的插件：

2.4.1 选择插件类型

```
public enum AssemblyTypes
{
    /// <summary>
```

```

    /// WF:流程程序集。用于承载流程表单资源以及相关代码
    /// </summary>
    WF = 0,
    /// <summary>
    /// Resource: 资源程序集。用于承载资源，可以承载代码
    /// </summary>
    Resource = 1,
    /// <summary>
    /// Code: 普通程序集，用于承载代码
    /// </summary>
    Code = 2,
    /// <summary>
    /// Theme: 主题程序集。用于承载系统主题。
    /// </summary>
    Theme = 3,
    /// <summary>
    /// Strategy: 策略程序集，用于加载策略
    /// </summary>
    Strategy = 4,
    /// <summary>
    /// Models: 模型程序集
    /// </summary>
    Models = 5,
    /// <summary>
    /// MVC: MVC程序集
    /// </summary>
    MVC = 6
}

```

1. 流程程序集

暂未实现。

2. 资源程序集

用于存放嵌入式资源。

3. 普通代码程序集

用于承载业务代码。

4. 主题程序集

暂未实现。

5. 策略程序集

用于承载策略代码。框架会自动扫描此程序集内实现的策略并加载。

6. 模型程序集

用于承载数据业务模型。

7. MVC 程序集

用于承载 MVC 代码（控制器、模型、视图等）。

具体可以参阅后续的 MVC 小节。

2.4.2 策略程序集

在此之前，请先阅读【插件策略】。

2.5 社区与博客

2.5.1 社区

暂无。

2.5.2 官方博客

目前请关注 <http://www.cnblogs.com/codelove/>

官方博客目前尚在开发中，请关注官网更新：<http://magicodes.net/>

2.5.3 反馈

所有有关 Magicodes.NET 的问题，请在此

<https://github.com/magicodes/Magicodes.NET/issues?q=is%3Aissue> 页面根据类型提交相应 Issues，群内不再作解答。QQ 群仅限技术交流，谢谢合作。

QQ 群：85318032

个人邮箱：liwq@magicodes.net

2.5.4 加入我们

具体请参阅：<http://www.magicodes.net/home/Contact>。

由于代码已经迁移至 GitHub，您可以直接在 GitHub 上获取最新代码然后提交您的修改，我们热忱欢迎您的加入。

2.6 协议

2.6.1 协议许可的权利

企业用户可以在网站页面底部保留“Powered by Magicodes.NET”标识的情况下将本软件应用于非商业用途，而不必支付软件授权费用。个人开发者可以在网站页面底部保留“Powered by Magicodes.NET”标识的情况下将本软件应用于商业用途，而不必支付软件授权费用。

获得商业授权之后，您可以将本软件应用于商业用途。商业授权用户享有反映和提出意见的权力，相关意见将被作为首要考虑，但没有一定被采纳的承诺或保证。

您拥有使用本软件构建的网站中全部文件、文章及相关信息的所有权，并独立承担与文章内容相关的法律义务。

2.6.2 协议规定的约束和限制

在未获得商业授权之前，任何个人或团体机构（如公司、政府、学校、社会团体等各类组织）不得将本软件用于商业用途（包括但不限于企事业单位网站、经营性网站、以盈利为目的或实现盈利的网站）。

不能将本软件及其相关软件的商业授权进行租借，出售或发放子授权。

禁止将 Magicodes.NET 程序的任何代码用于其它软件，禁止在 Magicodes.NET 整体或部分源代码的基础上发展其它程序。与其它条款无抵触的前提下，允许以自用为目的的进行进行二次开发或整合。

如果您未能遵守本协议，您的商业授权及许可的权利将被回收,并承担相应法律责任。

2.6.3 有限担保和免责声明

用户出于自愿而使用本软件，您必须了解使用本软件的的风险，且同意自己承担使用本软件的风险。

在任何情况下，对于因使用本软件或无法使用本软件而导致的任何损害赔偿，Magicodes.NET 团队均无须承担法律责任, 即使 Magicodes.NET 团队曾经被告知有可能出现该等损害赔偿。

Magicodes.NET 团队不对使用本软件构建的网站中的文章或信息承担责任。

2.6.4 其他

有关 Magicodes.NET 产品最终用户授权协议、商业授权与技术服务的详细内容，均由 Magicodes.NET 产品官方网站独家提供。Magicodes.NET 团队拥有在不事先通知的情况下，修改授权协议和服务价目表的权利，修改后的协议或价目表对自改变之日起的新授权用户生效。

电子文本形式的授权协议如同双方书面签署的协议一样，具有完全的和等同的法律效力。您一旦开始安装、复制或使用 Magicodes.NET 产品，即被视为完全理解并接受本协议的各项条款，在享有上述条款授予的权力的同时，受到相关的约束和限制。协议许可范围以外的行为，将直接违反本授权协议并构成侵权，我们有权随时终止授权，责令停止损害，并保留追究相关责任的权力。

Magicodes.NET 团队为 Magicodes.NET 产品的开发团队，依法独立拥有 Magicodes.NET 产品著作权和所有权。您一旦安装、复制或使用 Magicodes.NET 产品，表示您已经同意本协议条款。

Magicodes.NET 团队拥有对本协议的最终解释权。

版权所有(C)2014-2020，Magicodes.NET 团队保留所有权利

3 插件式架构

3.1 插件结构

3.2 插件策略

策略，是为了实现某个目标或者针对某些问题而制定的应对方案，以最终实现目标。在本框架中，策略（Strategy），则是为了实现某些功能或者处理某些特定问题而制定的通用方案或者规则。比如发送短信，这是系统中常用的功能，也许短信服务商有很多，实现发短信的方式也有很多，但是对于系统来说，只需要的是发送短信这个功能而已，如何让系统的插件都能够使用这个功能，那么我们就需要定制统一的接口来规范，那么根据各个服务商提供的短信发送方式。

对于框架来说，将常用的策略纳入框架是很有利于插件调用的。目前，Magicodes 框架提供了以下策略：

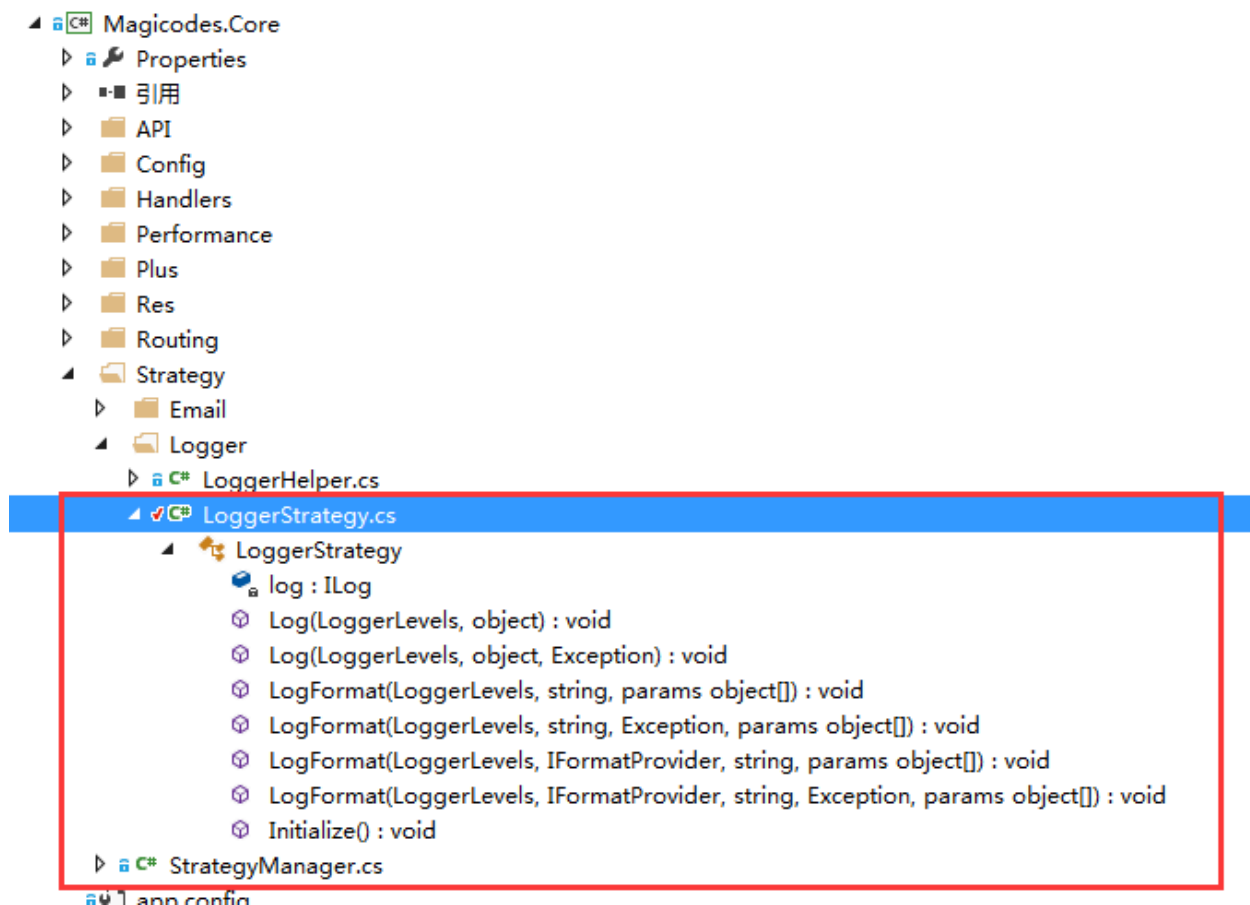
- Logger——日志策略
- Cache——缓存策略
- Email——邮件策略
- ScriptMin——脚本资源压缩策略

- Session——会话策略
- SMS——短信策略

.....

3.2.1 日志策略

该策略默认已经集成在 Magicodes.Core 中。



其实现方式如下：

- 实现 LoggerStrategyBase 虚拟类（接口）

```

/// <summary>
/// 日志策略
/// </summary>
public class LoggerStrategy : LoggerStrategyBase

```

- 实现策略初始化方法

```

/// <summary>
/// 初始化设置配置文件地址
/// </summary>

public override void Initialize()
{
    var properties = new NameValueCollection();
    properties["configType"] = "FILE";
    properties["configFile"] = "~/App_Data/config/NLog.config";
    LogManager.Adapter = new Common.Logging.NLog.NLogLoggerFactoryAdapter(properties);
    log = LogManager.GetLogger("Magicodes.Core.Strategy.Logger");
}

```

在初始化函数里，对 Common.Logging.NLog 进行了配置，以便记录日志。

- 实现日志策略函数。

```

/// <summary>
/// 日志策略
/// </summary>
0 个引用 | 李文强 | 4 个更改
public class LoggerStrategy : LoggerStrategyBase
{
    ILog log;
    0 个引用 | 李文强 | 3 个更改
    public override void Log(LoggerLevels loggerLevels, object message) {...}

    6 个引用 | 李文强 | 2 个更改
    public override void Log(LoggerLevels loggerLevels, object message, Exception exception) {...}

    11 个引用 | 李文强 | 2 个更改
    public override void LogFormat(LoggerLevels loggerLevels, string format, params object[] args) {...}

    1 个引用 | 李文强 | 2 个更改
    public override void LogFormat(LoggerLevels loggerLevels, string format, Exception exception, params object[] args) {...}

    1 个引用 | 李文强 | 2 个更改
    public override void LogFormat(LoggerLevels loggerLevels, IFormatProvider formatProvider, string format, params object[] args) {...}

    1 个引用 | 李文强 | 2 个更改
    public override void LogFormat(LoggerLevels loggerLevels, IFormatProvider formatProvider, string format, Exception exception, params
        object[] args) {...}

    /// <summary>
    /// 初始化设置配置文件地址
    /// </summary>
    6 个引用 | 李文强 | 3 个更改
    public override void Initialize() {...}
}

```

这里是使用 Common.Logging 的函数进行日志记录，如：

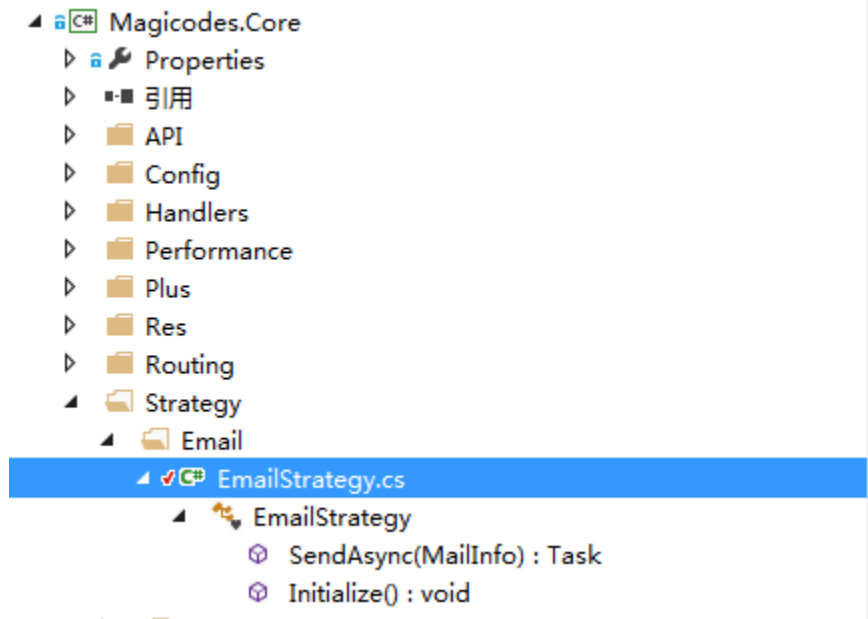
```

    ILog log;
    9 个引用 | 李文强 | 3 个更改
    public override void Log(LoggerLevels loggerLevels, object message)
    {
        switch (loggerLevels)
        {
            case LoggerLevels.Trace:
                log.Trace(message);
                break;
            case LoggerLevels.Debug:
                log.Debug(message);
                break;
            case LoggerLevels.Info:
                log.Info(message);
                break;
            case LoggerLevels.Warn:
                log.Warn(message);
                break;
            case LoggerLevels.Error:
                log.Error(message);
                break;
            case LoggerLevels.Fatal:
                log.Fatal(message);
                break;
            //性能警告
            //TODO: 需要记录性能警报信息
            case LoggerLevels.PerformanceWarn:
                log.Warn(message);
                break;
            default:
                break;
        }
        ExcuteOnLog(loggerLevels, message);
    }
}

```

3.2.2 邮件策略

邮件策略在 Magicodes.Core 默认已经实现，如下图所示：



具体实现代码如下所示：

```
using Magicodes.Web.Interfaces;
using Magicodes.Web.Interfaces.Config.Info;
using Magicodes.Web.Interfaces.Strategy.Email;
using Magicodes.Web.Interfaces.Strategy.Logger;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Mail;
using System.Text;
using System.Threading.Tasks;

//=====
//
//      Copyright (C) 2014-2016 Magicodes团队
//      All rights reserved
//
//      filename :EmailStrategy
//      description :
//
//      created by 雪雁 at 2014/10/22 13:57:45
//      http://www.magicodes.net
//=====
namespace Magicodes.Core.Strategy.Email
{
    /// <summary>
    /// 框架默认的邮件策略
    /// </summary>
    class EmailStrategy : IMailStrategy
```

```

{
    /// <summary>
    /// 发送邮件
    /// </summary>
    /// <param name="message">邮件信息</param>
    /// <returns></returns>
    public Task SendAsync(MailInfo message)
    {
        var mailConfig =
GlobalApplicationObject.Current.ApplicationContext.ConfigManager.GetConfig<MailConfigInfo>();
        if (mailConfig == null)
        {

GlobalApplicationObject.Current.ApplicationContext.ApplicationLog.Log(LoggerLevels.Warn,
string.Format("没有找到邮箱配置, 邮件【{0}】无法发送!", message.Subject));
            return Task.FromResult(false);
        }
        try
        {
            var mail = new MailMessage()
            {
                From = new MailAddress(mailConfig.MailFrom, mailConfig.FromNickName),
                To =
                {
                    new MailAddress(message.Destination)
                },
                Subject = message.Subject,
                Body = message.Body,
                BodyEncoding = System.Text.Encoding.UTF8,
                SubjectEncoding = System.Text.Encoding.UTF8,
                IsBodyHtml = true,
                Priority = MailPriority.High,
            };
            var mailClient = new SmtpClient()
            {
                Host = mailConfig.SmtpServer,
                Port = mailConfig.SmtpPort,
                UseDefaultCredentials = false,
                Credentials = new NetworkCredential(userName: mailConfig.UserName, password:
mailConfig.Password),
                EnableSsl = mailConfig.EnableSsl,
            };
            mailClient.Send(mail);
        }
        catch (Exception ex)
        {

GlobalApplicationObject.Current.ApplicationContext.ApplicationLog.Log(LoggerLevels.Error,
string.Format("发送邮件失败【{0}】!", message.Destination), ex);
            return Task.FromResult(false);
        }
        return Task.FromResult(true);
    }
}

```



```

    }

    public void Initialize()
    {
    }
}
}

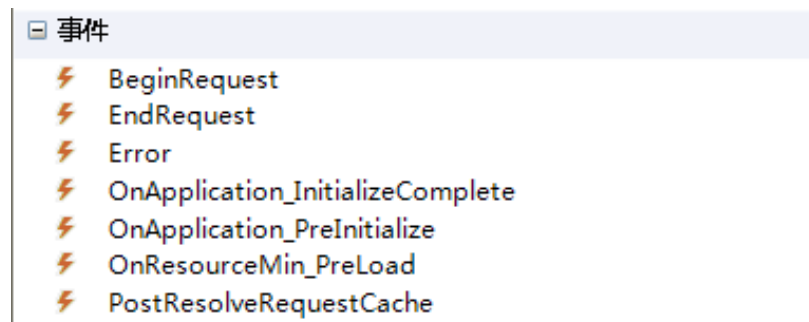
```

3.2.3 支付策略

3.3 事件管理

注意：事件管理器目前还不够合理，后期会重构此块。所以以下内容请对比最新代码，请以最新代码内容为准。

通过 `GlobalApplicationObject.Current.EventsManager` 可以访问事件管理器，默认有以下事件可以使用：



3.3.1 OnApplication_PreInitialize

应用程序预初始化事件。

在执行完框架的预初始化方法后，会调用此事件。

3.3.2 OnApplication_InitializeComplete

应用程序类初始化完成事件。

在执行完框架初始化方法后，会调用此事件。

3.3.3 BeginRequest

开始请求事件。

在 ASP.NET 响应请求时作为 HTTP 执行管线链中的第一个事件发生。

3.3.4 EndRequest

请求结束事件。

除了以上事件，在 Magicodes.Core.Web 中也有一些事件以供使用。您可以通过访问 Magicodes.Core.Web. GlobalConfigurationManager 类来获取。其主要有以下事件（具体请以最新代码为准）：

3.3.5 OnConfiguration_Config

配置事件。原理：

```
GlobalConfiguration.Configure(config =>
{
    OnConfiguration_Config(config, null);
});
```

Demo：

```
public static void MapHttpAttributeRoutes()
{
    GlobalConfigurationManager.OnConfiguration_Config +=
GlobalConfigurationManager_OnConfiguration_Config_MapHttpAttributeRoutes;
}
static void
GlobalConfigurationManager_OnConfiguration_Config_MapHttpAttributeRoutes(object sender, EventArgs
e)
{
    HttpConfiguration config = (HttpConfiguration)sender;
    config.MapHttpAttributeRoutes();
}
```

3.3.6 OnConfiguration_AppBuilder

App 配置构建事件。

原理：

```
[assembly: OwinStartup(typeof(Magicodes.Core.Web.Startup))]
namespace Magicodes.Core.Web
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            GlobalConfigurationManager.AppBuilder(app);
        }
    }
}
```

Demo：

```

using Magicodes.Core.Web;
using Magicodes.Web.Interfaces.Plus;
using Owin;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

//=====
//
//      Copyright (C) 2014-2016 Magicodes团队
//      All rights reserved
//
//      filename :Plus
//      description :声明式认证插件启动类
//
//      created by 雪雁 at 2014/10/25 20:54:54
//      http://www.magicodes.net
//=====
namespace Magicodes.Strategy.Identity
{
    class Plus : IPlus
    {
        public void Initialize()
        {
            GlobalConfigurationManager.OnConfiguration_AppBuilder +=
GlobalConfigurationManager_OnConfiguration_AppBuilder;
        }

        void GlobalConfigurationManager_OnConfiguration_AppBuilder(object sender, EventArgs e)
        {
            var app = (IApplicationBuilder)sender;
            AppAuthConfig.ConfigureAuth(app);
        }

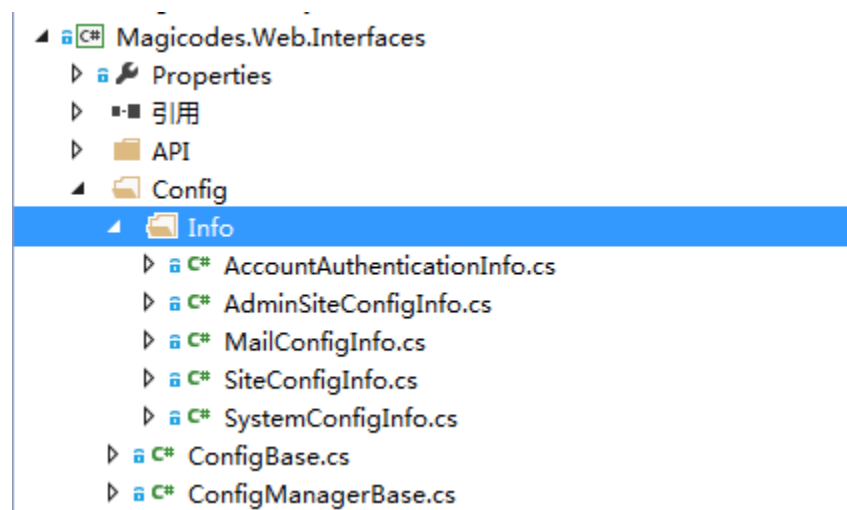
        public void Install()
        {
        }

        public void Uninstall()
        {
        }
    }
}

```

3.4 配置管理

Magicodes.NET 提供了一些默认的配置类以供使用，如下所示：



3.4.1 账户配置（AccountAuthenticationInfo）

3.4.2 后台站点配置（AdminSiteConfigInfo）

3.4.3 邮件配置（MailConfigInfo）

3.4.4 站点信息配置（SiteConfigInfo）

3.4.5 系统配置（SystemConfigInfo）

3.5 插件商店

4 插件案例

4.1 开发人员面板

4.2 声明式认证 (`MAGICODES.STRATEGY.IDENTITY`)

4.3 内容发布 (`MAGICODES.CMS`)

4.4 博客 (`MAGICODES.BLOGS`)

5 移动设备支持

6 MVC

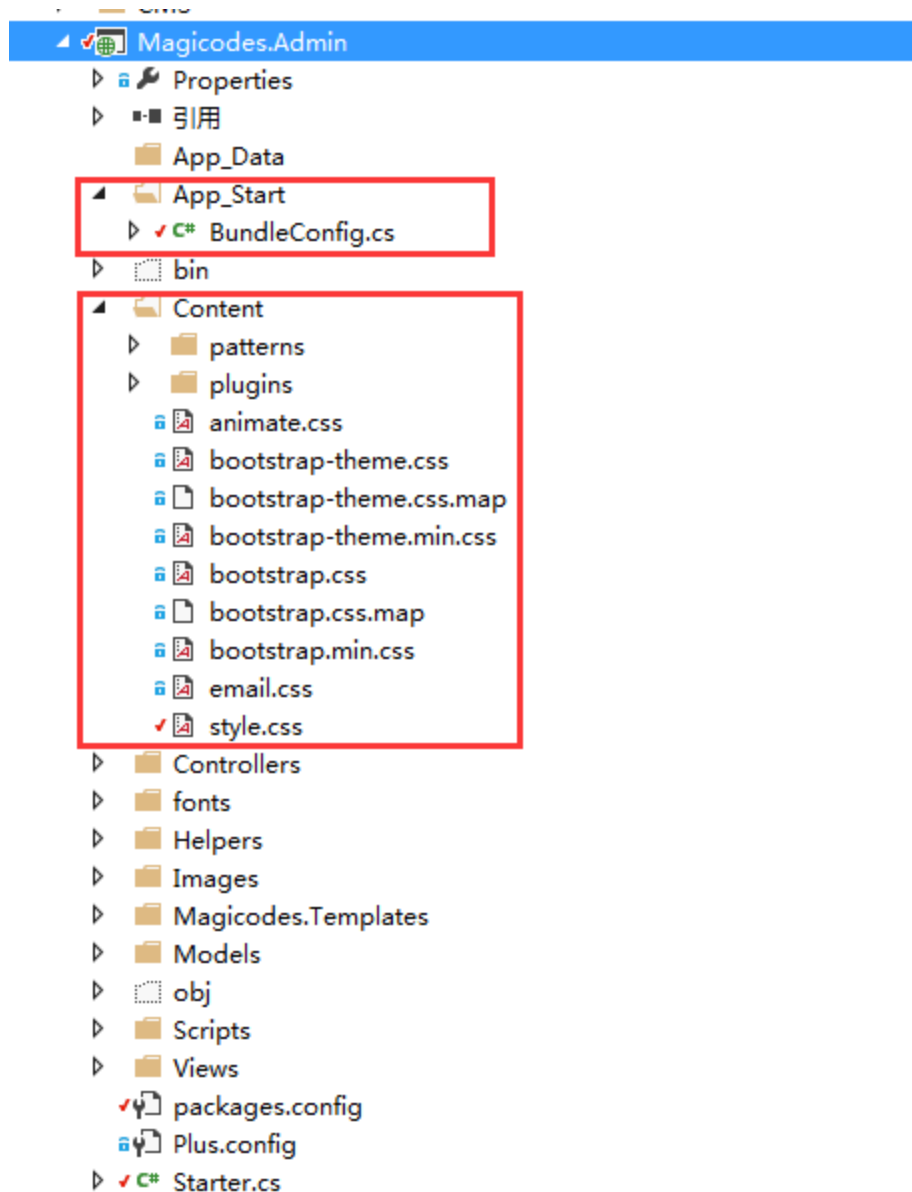
6.1 MVC 插件

MVC 插件需要将插件类型设置为 MVC、MVCAAdmin 或者 MVCHome，一个项目中只允许存在一个 MVCHome 类型的 Mvc 插件。

6.1.1 插件 Bundle

Microsoft.AspNet.Web.Optimization 是一个不错的组件，可以将多个资源合并并且能够压缩资源（css 和 js），相信很多人都挺喜欢用的。

那么在 Magicodes.NET 里面是怎么处理它的呢？我们来看看 Magicodes.Admin。



如图所示，Magicodes.Admin 插件中包含了不少资源。而类 BundleConfig 是用来处理 Bundle 的。

在 BundleConfig 中，我们可以看到以下代码：

```
//~/plus/Plugins/Magicodes.Admin
var plusPath = "~/ " + Starter.PlusPath;
//~/Magicodes.Admin/bundles
var plusName = "~/ " + Starter.PlusName + "/bundles";
// CSS style (bootstrap/inspinia)
bundles.Add(new StyleBundle(plusName + "/Content/css").Include(
    plusPath + "/Content/bootstrap.min.css",
    plusPath + "/Content/animate.css",
    plusPath + "/Content/style.css"));
```

上面仅仅是一部分，如图所示还有很多：

```
namespace Magicodes.Admin
{
    1 个引用 | magicodes, 15 天之前 | 2 个更改
    public class BundleConfig
    {
        1 个引用 | magicodes, 15 天之前 | 2 个更改
        public static void RegisterBundles(BundleCollection bundles)
        {
            //plus/Plugins/Magicodes.Admin
            var plusPath = "" + Starter.PlusPath;
            //Magicodes.Admin/bundles
            var plusName = "" + Starter.PlusName + "/bundles";
            // CSS style (bootstrap/inspinia)
            bundles.Add(new StyleBundle(plusName + "/Content/css").Include(
                plusPath + "/Content/bootstrap.min.css",
                plusPath + "/Content/animate.css",
                plusPath + "/Content/style.css"));

            // Font Awesome icons
            bundles.Add(new StyleBundle(plusName + "/font-awesome/css").Include(
                plusPath + "/fonts/font-awesome/css/font-awesome.min.css"));

            // jQuery
            bundles.Add(new ScriptBundle(plusName + "/jquery").Include(
                plusPath + "/Scripts/jquery-2.1.1.min.js"));

            // jQueryUI CSS
            bundles.Add(new ScriptBundle(plusName + "/jqueryuiStyles").Include(
                plusPath + "/Scripts/plugins/jquery-ui/jquery-ui.min.css"));

            // jQueryUI
            bundles.Add(new StyleBundle(plusName + "/jqueryui").Include(
                plusPath + "/Scripts/plugins/jquery-ui/jquery-ui.min.js"));

            // Bootstrap
            bundles.Add(new ScriptBundle(plusName + "/bootstrap").Include(
                plusPath + "/Scripts/bootstrap.min.js"));

            // Inspinia script
            bundles.Add(new ScriptBundle(plusName + "/inspinia").Include(
                plusPath + "/Scripts/inspinia.js"));
        }
    }
}
```

Bundles 写好后，我们需要在插件初始化时设置，如下所示：

```
public class Starter : IPlus
{
    //插件名
    public const string PlusName = "Magicodes.Admin";
    public const string PlusPath = "/plus/Plugins/Magicodes.Admin";

    public void Initialize()
    {
        GlobalApplicationObject.Current.EventsManager.OnApplication_InitializeComplete +=
        EventsManager_OnApplication_InitializeComplete;

        GlobalConfigurationManager.ODataBuilder.EntitySet<SiteAdminNavigation>("SiteAdminNavigations");
    }

    void EventsManager_OnApplication_InitializeComplete(object sender, ApplicationArgs e)
    {
        BundleConfig.RegisterBundles(BundleTable.Bundles);
    }

    public void Install()
    {
    }
}
```

```

    {
        throw new NotImplementedException();
    }

    public void Uninstall()
    {
        throw new NotImplementedException();
    }
}

```

注意：bundles 名称是必须的，插件名称也是必须的，以便系统能够识别为插件资源并且为 bundle 资源。关于其原理，您可以查看 GlobalConfigurationManager 类的 RegisterBundlesRequest 方法。

然后，我们就可以在 View 中如此写：

```

@Styles.Render("~/fonts/font-awesome/css/font-awesome.min.css")
@Styles.Render("~/bundles/Content/css")

```

还是一样的语法。不过为了支持此语法，我们还有两步骤：

1. 移除 Views 目录下 Web.config 文件中配置的命名空间“System.Web.Optimization”
2. 添加命名空间如右侧所示“<add namespace=“Magicodes.Admin” />”，如：

```

<system.web.webPages.razor>
  <host factoryType="System.Web.Mvc.MvcWebRazorHostFactory, System.Web.Mvc, Version=5.0.0.0, Culture=neutral,
    PublicKeyToken=31BF3856AD364E35" />
  <pages pageBaseType="System.Web.Mvc.WebViewPage">
    <namespaces>
      <clear />
      <add namespace="System.Web.Mvc" />
      <add namespace="System.Web.Mvc.Ajax" />
      <add namespace="System.Web.Mvc.Html" />
      <add namespace="System.Web.Routing" />
      <add namespace="Magicodes.Admin" />
      <add namespace="Magicodes.Admin.Models" />
      <add namespace="Magicodes.Core.Web.Utility" />
    </namespaces>
  </pages>
</system.web.webPages.razor>

```

3. 添加下面两个方法：

1. namespace Magicodes.Admin
2. {
3. public static class Scripts
4. {
5. /// <summary>
6. /// 呈现路径集的插件链接标记。


```

7.         /// </summary>
8.         /// <param name="html"></param>
9.         /// <param name="path"></param>
10.        /// <returns></returns>
11.        public static IHtmlString Render(params string[] paths)
12.        {
13.            var strs = paths.Select(p => p.Replace("~/", "~/ " + Starter.PlusName +
14.                "/")).ToArray();
15.            return System.Web.Optimization.Scripts.Render(strs);
16.        }
17.    }

```

```

namespace Magicodes.Admin
{
    public static class Styles
    {
        /// <summary>
        /// 呈现路径集的插件链接标记。
        /// </summary>
        /// <param name="html"></param>
        /// <param name="path"></param>
        /// <returns></returns>
        public static IHtmlString Render(params string[] paths)
        {
            var strs = paths.Select(p => p.Replace("~/", "~/ " + Starter.PlusName +
112.                "/")).ToArray();
            return System.Web.Optimization.Styles.Render(strs);
        }
    }
}

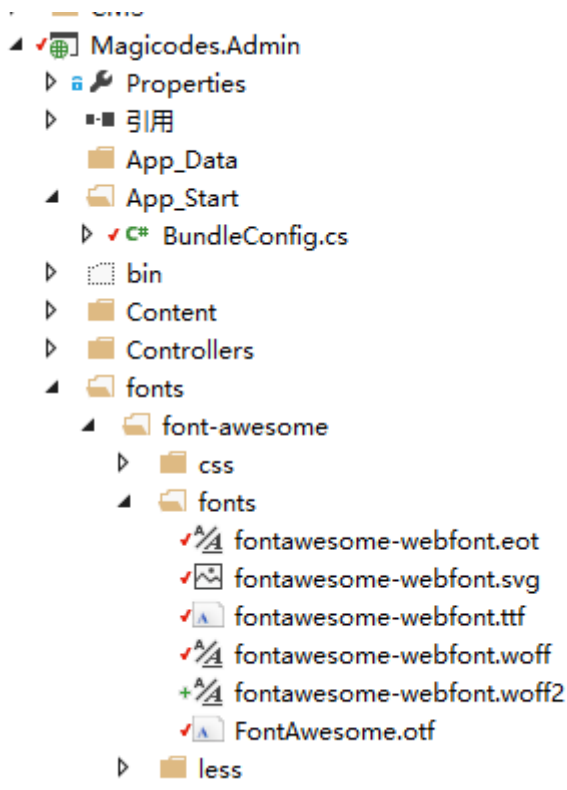
```

6.1.2 插件资源请求

有时候，我们并不想把所有的资源都 bundle，再说了，bundle 只能 bundle css 和 js，图片怎么办，字体文件怎么破？如下所示，如下请求：

```
@Styles.Render("~/fonts/font-awesome/css/font-awesome.min.css")
```

别慌，Magicodes.NET 已经考虑到了这种情况，CSS 中的字体文件亦会指向插件对应的目录，如下所示：



注意：如果字体文件无法加载，请确定是否已经添加了相应的 Mime 类型。比如上面的扩展名为“.woff2”的文件。

这个跳转是在 Magicodes.Core.Web 中实现的，在 MVC 插件初始化完成就已经自动配置好了，有兴趣的朋友可以去查看相关代码。

关于图片资源，由于加上插件名并不友好，于是提供了以下方式：

```

```

6.2 MVC 门户插件

门户插件是用于显示门户页面，同一个系统中只能存在一个。

6.2.1 插件配置

配置如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<PlusConfigInfo>
  <AssemblyType>MVC</AssemblyType>
  <MvcPlusType>MVCHome</MvcPlusType>
  ...
</PlusConfigInfo>
```

6.2.2 路由注册规则

```
RouteTable.Routes.MapRoute(
    name: "MCV_" + mvcHomePlus.PlusName,
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional, pluginName = mvcHomePlus.PlusName });
```

如上所示，门户程序集的默认页为 Home/Index

6.2.3 Magicodes.NET 官网（Magicodes.Mvc.Default）

Magicodes.Mvc.Default 插件为框架默认的门户程序集，页面如下所示：



6.3 MVC 后台插件

后台插件为了呈现后台页面，同一个系统中只能存在一个。其会自动将该插件下得控制器注册到 Admin 前缀的 url。

6.3.1 插件配置

配置文件示例如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<PlusConfigInfo>
  <AssemblyType>MVC</AssemblyType>
  <MvcPlusType>MVCAdmin</MvcPlusType>
  ...
  <PlusAdminMenus>...</PlusAdminMenus>
</PlusConfigInfo>
```

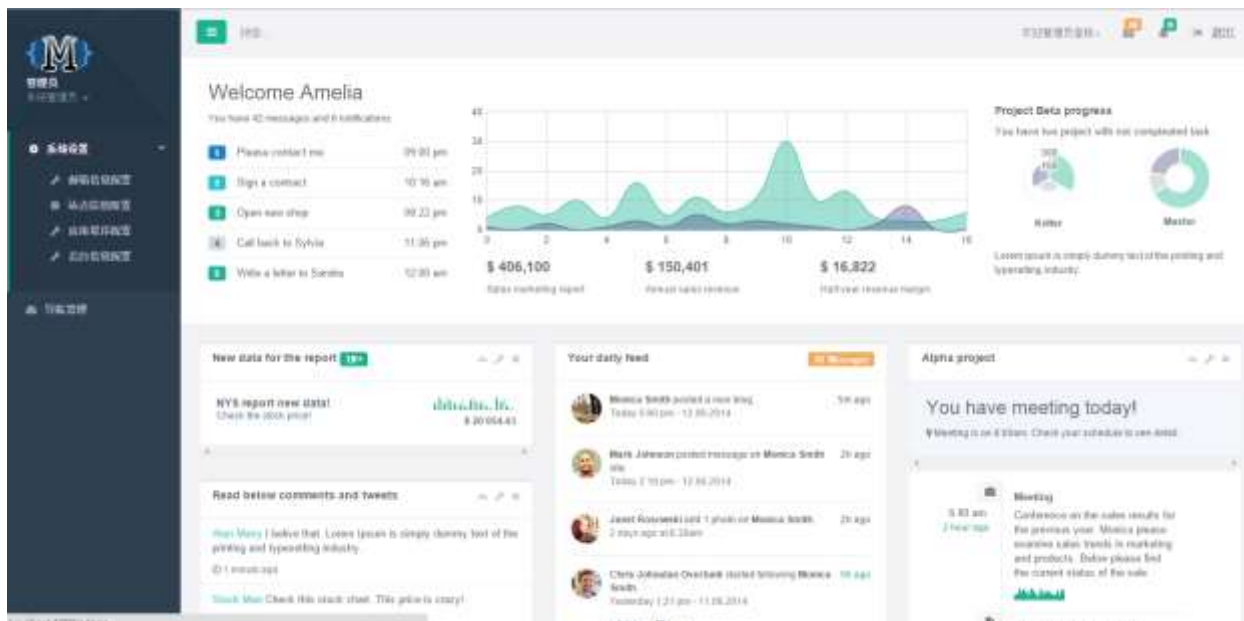
6.3.2 路由注册规则

```
//此类型插件只支持一个
case MvcPlusTypes.MVCAdmin:
{
    RouteTable.Routes.MapRoute(
        name: "MCV_" + mvcPlus.PlusName,
        url: "admin/{controller}/{action}/{id}",
        defaults: new { action = "Index", controller = "Admin", id = UrlParameter.Optional, pluginName
            = mvcPlus.PlusName });
}
break;
```

如上所示，在输入/admin 时，框架会自动去查找该插件中的 Admin/Index。

6.3.3 Magicodes.NET 后台（Magicodes.Admin）

Magicodes.Admin 目前为默认的后台 MVC 插件。页面如下所示：



6.3.4 插件后台菜单配置

现在，插件配置文件中已经支持配置后台菜单，如下所示：

```

<?xml version="1.0" encoding="utf-8" ?>
<PlusConfigInfo>
  <AssemblyType>MVC</AssemblyType>
  <MvcPlusType>MVCAdmin</MvcPlusType>
  ...
  <PlusAdminMenus>
    <Menu>
      <Id>{CBCB461B-CA2C-442B-AF6A-CD9796B2C760}</Id>
      <Text>导航管理</Text>
      <Href>/Admin/SiteAdminNavigations</Href>
      <IconCls>fa fa-tachometer</IconCls>
    </Menu>
    <Menu>
      <Id>{6AAB6E0C-4490-46AE-9123-112FB81C9393}</Id>
      <Text>系统设置</Text>
      <Href></Href>
      <IconCls>fa fa-cog</IconCls>
      <SubMenus>
        <Menu>
          <Text>站点信息配置</Text>
          <Href>/Admin/Config/SiteConfigInfo</Href>
          <IconCls>fa fa-cog</IconCls>
        </Menu>
        <Menu>
          <Text>邮箱信息配置</Text>
          <Href>/Admin/Config/MailConfigInfo</Href>
          <IconCls>fa fa-wrench</IconCls>
        </Menu>
        <Menu>
          <Text>后台信息配置</Text>
          <Href>/Admin/Config/AdminSiteConfigInfo</Href>
          <IconCls>fa fa-wrench</IconCls>
        </Menu>
        <Menu>
          <Text>应用程序配置</Text>
          <Href>/Admin/Config/SystemConfigInfo</Href>
          <IconCls>fa fa-wrench</IconCls>
        </Menu>
      </SubMenus>
    </Menu>
  </PlusAdminMenus>
</PlusConfigInfo>

```

6.4 MVC 插件

MVC 插件为了呈现插件页面。

6.4.1 插件配置

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<PlusConfigInfo>
  <AssemblyType>MVC</AssemblyType>
</PlusConfigInfo>
```

6.4.2 路由注册规则

```
RouteTable.Routes.MapRoute(
    name: "MCV_" + mvcPlus.PlusName,
    url: "_{pluginName}/{controller}/{action}/{id}",
    defaults: new { action = "Index", id = UrlParameter.Optional, pluginName =
mvcPlus.PlusName } );
```

如上所示，你可以通过_{pluginName}/{controller}/{action}/{id}来访问插件页面，另外，你也可以通过{controller}/{action}?pluginName={pluginName}来访问。

例如：

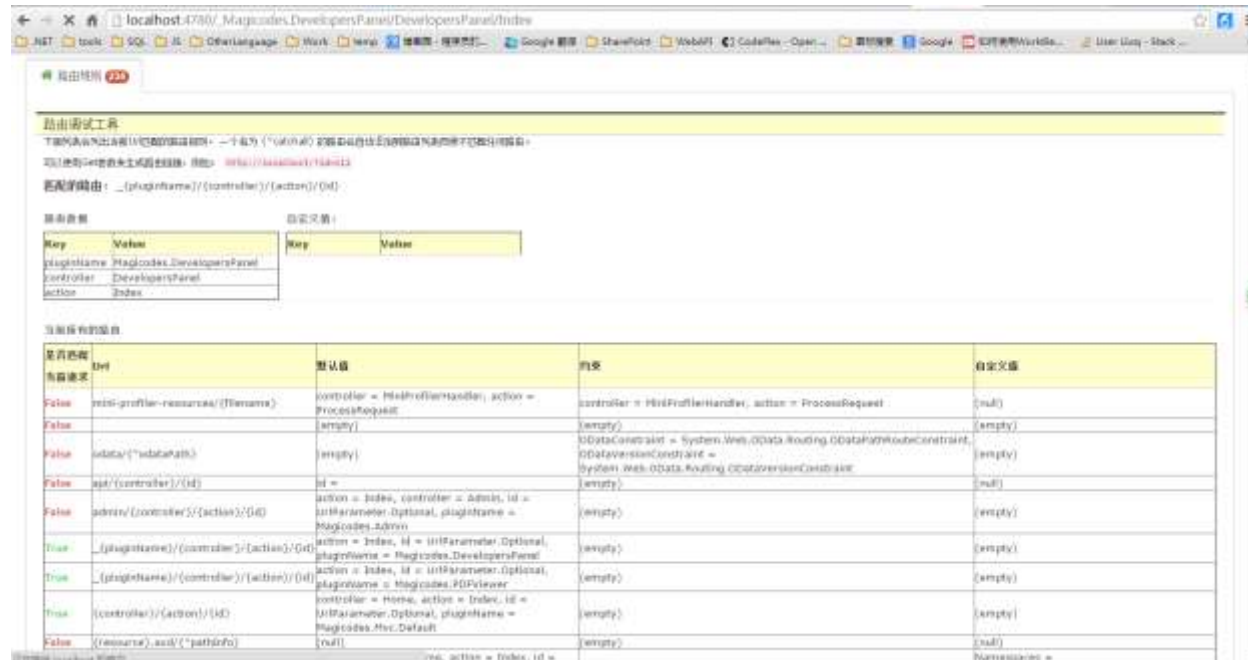
<http://localhost:4780/DevelopersPanel/Index?pluginName=Magicodes.DevelopersPanel>

<http://localhost:4780/Magicodes.DevelopersPanel/DevelopersPanel/Index>

注意：以上链接需要管理员登陆才能访问。

6.4.3 开发人员面板插件（Magicodes.DevelopersPanel）

根据上节的链接，我们可以很方便的访问插件页面。



The screenshot shows the Magicodes.DevelopersPanel web application. The routing configuration tool is displayed, showing a list of routes. The routes are configured for the Magicodes.DevelopersPanel plugin.

| Key | Value |
|------------|---------------------------|
| pluginName | Magicodes.DevelopersPanel |
| controller | DevelopersPanel |
| action | Index |

| 路由名称 | Url | 默认值 | 约束 | 自定义值 |
|--|---|--|---|------|
| mini-profile-resources/{Resource} | controller = MiniProfileHandler, action = ProcessRequest | controller = MiniProfileHandler, action = ProcessRequest | | |
| odata/{odataPath} | empty | empty | odataConstraint = System.Web.OData.Routing.ODataRouteConstraint, odataVersionConstraint = System.Web.OData.Routing.ODataVersionConstraint | |
| api/{controller}/{id} | id = | empty | | |
| admin/{controller}/{action}/{id} | action = Index, controller = Admin, id = UrlParameter.Optional, pluginName = Magicodes.Admin | empty | | |
| _{pluginName}/{controller}/{action}/{id} | action = Index, id = UrlParameter.Optional, pluginName = Magicodes.DevelopersPanel | empty | | |
| _{pluginName}/{controller}/{action}/{id} | action = Index, id = UrlParameter.Optional, pluginName = Magicodes.PDPViewer | empty | | |
| {controller}/{action}/{id} | controller = Home, action = Index, id = UrlParameter.Optional, pluginName = Magicodes.Mvc.Default | empty | | |
| {resource}.xml/{pathInfo} | null | empty | | |

6.5 路由

6.5.1 注册 MVC

6.5.2 注册区域

6.5.3 注册 WebAPI

6.5.4 注册 OData

7 安全性

从安全方面考虑，Magicodes.NET 提供了以下安全特性以供插件使用：

7.1 ADMINCONTROL

7.2 ADMINATTRIBUTE

7.3 **阻止 XSS（跨站脚本攻击）攻击**

7.4 VALIDATEANTIFORGERYTOKEN——**阻止 CSRF 攻击**

7.5 BIND——**重复提交**

7.6 ISPOSTEDFROMTHISSITEATTRIBUTE——**防御伪造攻击**

7.7 **防止密码暴力破解**

7.8 **异常日志**

8 性能分析

9 WEBAPI

10 ODATA

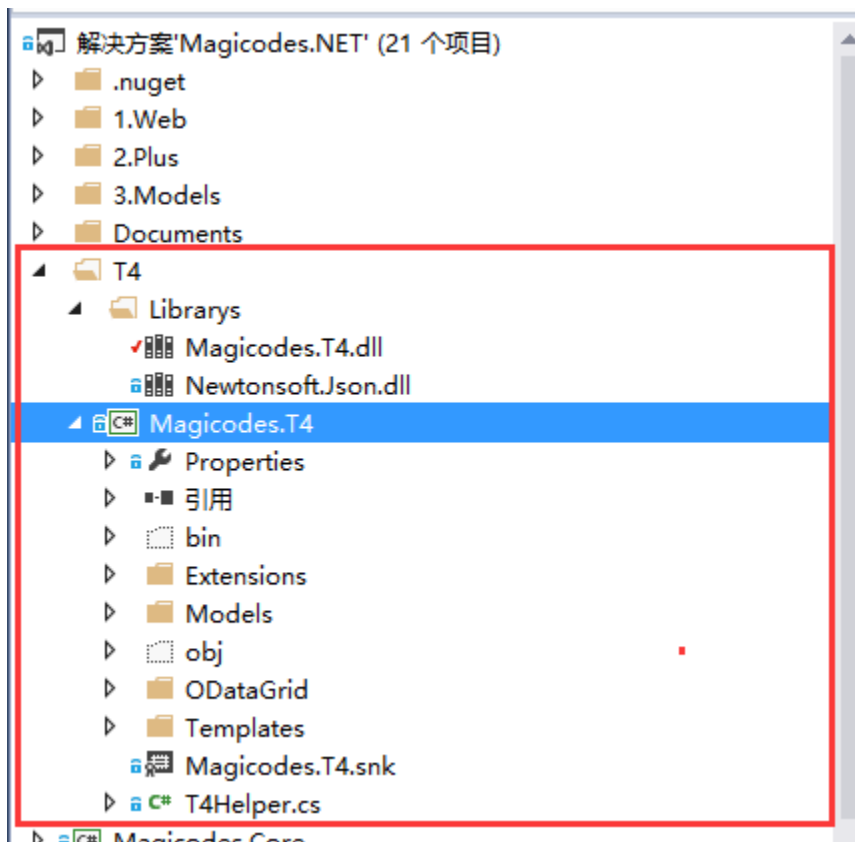
11 代码生成（T4）

还在加班写代码？还在不断地重写业务逻辑？还在一个一个的打补丁？太 Low 了，让 Magicodes.NET 帮你来编写代码吧。凝聚 Magicodes.NET 团队的智慧，让 Magicodes.NET 帮您编写代码。

让您更关注您的业务逻辑，这是 Magicodes.NET 框架设计理念之一。

11.1 公用库&模板

为了不重复造轮子，我们先定义点公共的东西。请看大屏幕，哦错了，是解决方案：



在解决方案 T4 目录下，承载了 Magicodes.NET 代码生成的期望与未来。让我们一起走近她吧。

11.1.1 Library 目录

此目录承载 T4 生成公用的组件，便于代码生成。比如：

```
<#@ assembly name="$(SolutionDir)\T4\Librarys\Newtonsoft.Json.dll" #>
```

上面代码在 T4 模板里面导入了组件 Newtonsoft.Json.dll。也就是 JSON.NET，为了便于 JSON 序列化。

我们可以在项目工程中的【生成事件】——》【后期生成事件命令行】中编写批处理来自动将类库复制到此目录，具体代码如下所示：

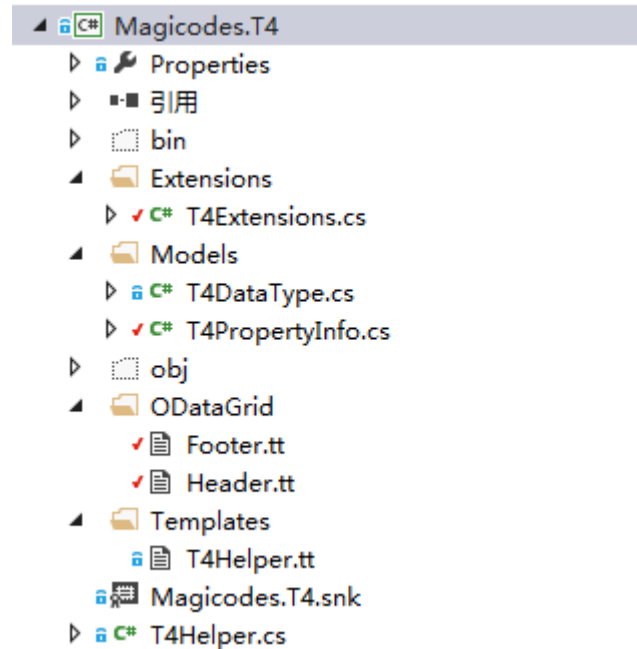
```
set "targetDir=$(SolutionDir)T4\Librarys"

if not exist %targetDir% md %targetDir%

echo f | xcopy "$(TargetPath)" %targetDir% /Y
```

11.1.2 Magicodes.T4 库

此库为 T4 代码公共库。其结构如下所示：



11.1.3 Templates 目录

此目录放置公共模板。比如 T4Helper.tt。

11.2 辅助生成特性

以下特性仅供代码生成时使用，并不会影响代码运行时逻辑。

11.2.1 T4FormGroupAttribute

表单组特性。

11.2.2 T4GenerationIgnoreAttribute

代码生成忽略特性。仅允许对类和属性使用。其中 IgnoreParts 表示忽略部分，其默认值为 All：

```
/// <summary>
/// 忽略部分
/// </summary>
public enum IgnoreParts
{
    /// <summary>
    /// 所有
    /// </summary>
    All = 0,
    /// <summary>
    /// 表单
    /// </summary>
    Form = 1,
    /// <summary>
    /// 表格
    /// </summary>
    Grid = 2
}
```

11.2.3 T4OnlyVerificationAttribute

唯一验证特性。仅允许对属性使用。

当 IgnoreDeletedData 设置为 True 时，将忽略已删除的数据。

11.2.4 T4OdataGridAttribute

ODataGrid 生成特性。仅允许对类使用。当

11.2.5 T4ReadOnlyFieldAttribute

只读特性。

11.3 辅助类和函数

11.3.1 T4DataType

11.3.2 T4Helper

11.4 关于生成 CRUD

自从放弃了 ASP.NET Scaffolding Template (ASP.NET 基架模板) (具体见博文：[Magicodes.NET 框架之路——让代码再飞一会 \(ASP.NET Scaffolding \)](#))，我就开始了代码生成的自主探索之旅。

首先介绍下我们的生成理念——基于 Model 以及辅助特性生成增删改查控制器与视图。

按照新的生成方式，我们的操作顺序如下：

1. 添加 Model，并且配置好生成特性（可以参考上面的辅助生成特性）
2. 添加“入口”T4 模板（暂且允许我这么称呼，它用于引用一些公共 T4 模板，并且设置生成规则/参数）
3. 设置生成参数
4. 保存生成或者运行生成
5. 查看并确认生成结果（控制器、视图目录以及相关视图文件是否已经成功生成）

接下来我们以 Magicodes.Admin 中的列表生成作为例子。

首先我们来看看 Model，举个例子：

```
/// <summary>
/// 后台导航菜单信息
/// </summary>
[Table("Nav_SiteAdminNavigation")]
[DisplayName("导航菜单管理")]
[T4DataTable(Title = "后台导航菜单管理", Description = "后台导航菜单来自插件配置文件，请确保已经配置了相关菜单。")]
public class SiteAdminNavigation : SiteAdminNavigationBase<string>
{
}
```

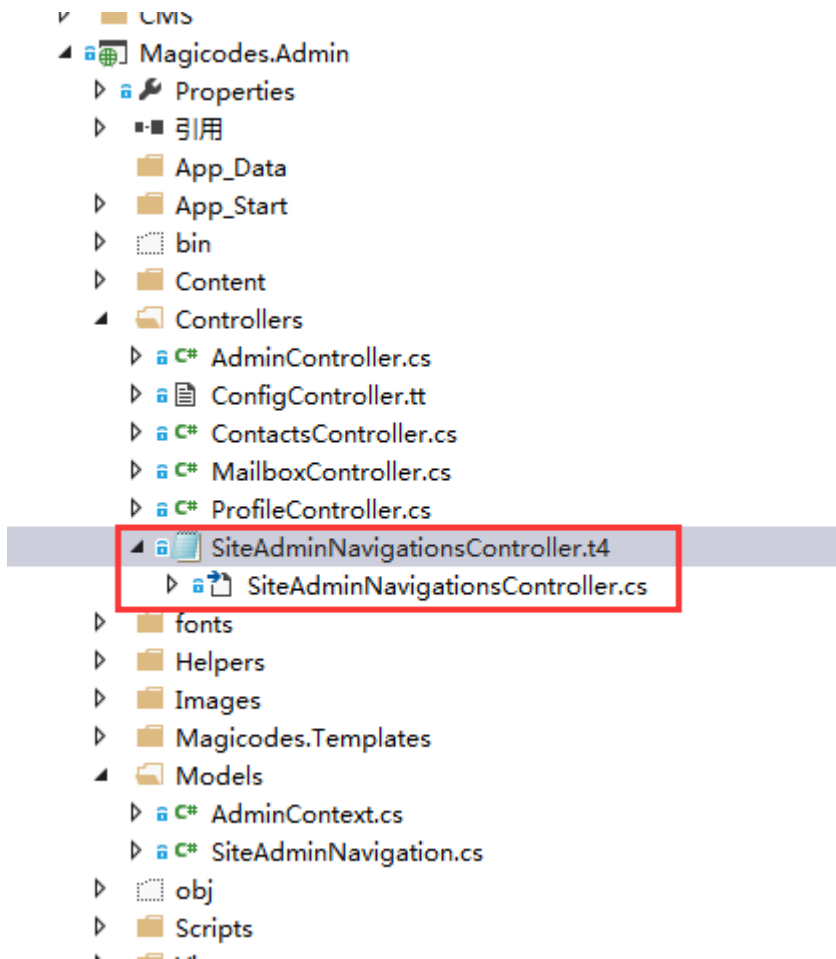
其继承自 SiteAdminNavigationBase：

```

    /// <summary>
    /// 后台站点导航
    /// </summary>
    [Serializable]
    44 个引用 | magicodes, 41 天之前 | 7 个更改
    public class SiteAdminNavigationBase<TUserKeyType> : CommonBusinessModelBase<Guid, TUserKeyType>
    {
        /// <summary>
        /// 父级Id
        /// </summary>
        [Display(Name = "父级Id")]
        1 个引用 | magicodes, 41 天之前 | 2 个更改
        public Guid? ParentId { get; set; }
        /// <summary>
        /// 显示名称
        /// </summary>
        [MaxLength(100)]
        [Display(Name = "显示文本")]
        [Required]
        1 个引用 | magicodes, 41 天之前 | 3 个更改
        public string Text { get; set; }
        /// <summary>
        /// 菜单链接
        /// </summary>
        [MaxLength(300)]
        [Display(Name = "菜单链接")]
        1 个引用 | magicodes, 57 天之前 | 2 个更改
        public string Href { get; set; }
        /// <summary>
        /// 图标样式
        /// </summary>
        [MaxLength(256)]
        [Display(Name = "图标样式")]
        1 个引用 | magicodes, 57 天之前 | 2 个更改
        public string IconCls { get; set; }
        /// <summary>
        /// 文本样式
        /// </summary>
        [MaxLength(256)]
        [Display(Name = "文本样式")]
        1 个引用 | magicodes, 57 天之前 | 2 个更改
        public string TextCls { get; set; }
    }

```

接下来，我们需要创建一个 T4 模板。如：SiteAdminNavigationsController.t4。



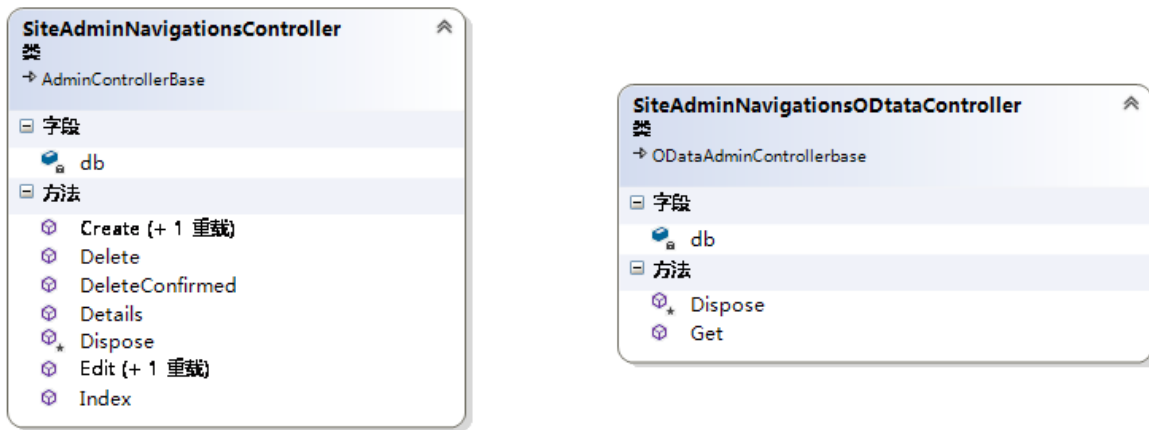
其内容如下所示：

```
<#@ include file="$(SolutionDir)\T4\Magicodes.T4\Templates\MVC\Imports.include.t4" #>
<#@ output extension=".cs" #>
<#@ import namespace="Magicodes.Admin.Models" #>
<#
    var MVCGenerateParams=new T4MVCGenerateInfo<SiteAdminNavigation,AdminContext>();
    MVCGenerateParams.UseAsync=true;
    //MVCGenerateParams.AreaName="";
    //MVCGenerateParams.ControllerRootName="";
    MVCGenerateParams.Namespace="Magicodes.Admin.Controllers";
    MVCGenerateParams.ControllerName="SiteAdminNavigations";
    MVCGenerateParams.IsAdminController=true;
    MVCGenerateParams.UseLayoutPage=true;
    #>
    <#@ include file="$(ProjectDir)Magicodes.Templates\Generate.include.t4" #>
```

从上述代码不难看出，这里主要是做一些生成的设置。首先第一行和最后一行分别引用了一些公共 T4 模板，用于提供代码生成逻辑以及视图模板。然后提供了一些参数用于设置是否异步、区域、控制器、命名空间、是否为后台控制器等等。当你打开这个 T4 文件，保存时，其就会生成对

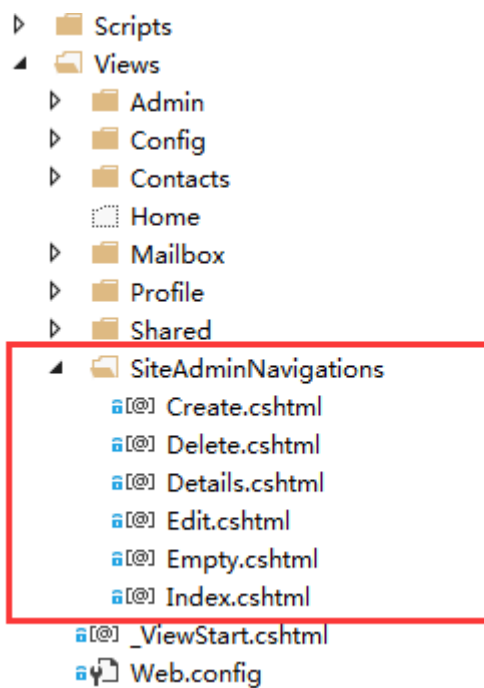
应的控制器以及相关视图（会自己创建视图目录并将相关视图添加到该目录）。我们依次来看看这些生成的代码。

控制器类图：



从类图中来看，增删改以及显示代码俱全，而查询是由右侧的 Odata 控制器提供（目前尚未将其绑定到查询界面）。

再看看视图：



我们再来看看界面：

| 导航菜单管理 | | | | | | | | | | |
|--------------------------------------|------|-----------------------------------|--------------|-------|-------------|-------------------|--------------------------------------|------|--------------------------------------|----|
| + 新建 | | | | | | | | | | |
| 父级id | 显示文本 | 菜单链接 | 图标样式 | 文本样式 | 是否显示于侧边栏或顶部 | 菜单名称数字类型 | 全部路由再路由的源地址 | 路由id | 排序号 | 备注 |
| 0a006b0c-4490-48ac-9123-1129d81c5000 | 系统管理 | /Admin/Config/MenuConfigInfo | fa fa-wrench | false | | FromChildrenCount | 70548180-8976-4366-8576-8763c7a2588b | 0 | 054x703b-158a-4979-ba8e-082aed164d | |
| | 系统设置 | | fa fa-cog | false | | FromChildrenCount | 70548180-8976-4366-8576-8763c7a2588b | 0 | 0a006b0c-4490-48ac-9123-1129d81c5000 | |
| 0a006b0c-4490-48ac-9123-1129d81c5000 | 站点管理 | /Admin/Config/StaticConfigInfo | fa fa-cog | false | | FromChildrenCount | 70548180-8976-4366-8576-8763c7a2588b | 0 | 1a3970c8-b47a-488a-8a27-76283c3234d | |
| 0a006b0c-4490-48ac-9123-1129d81c5000 | 应用管理 | /Admin/Config/SystemConfigInfo | fa fa-wrench | false | | FromChildrenCount | 70548180-8976-4366-8576-8763c7a2588b | 0 | 6a605287-c47a-488a-ba75-2c3b4858a4d | |
| | 系统管理 | /Admin/StaticAdmin/NavigationInfo | fa fa-rocket | false | | FromChildrenCount | 70548180-8976-4366-8576-8763c7a2588b | 0 | cbcb067b-ca2c-442b-a88e-cd8b7983a2c7 | |

新建

导航菜单管理 / 新建

保存 取消 返回

导航菜单管理 | 新建

父级id

显示文本

菜单链接

图标样式

文本样式

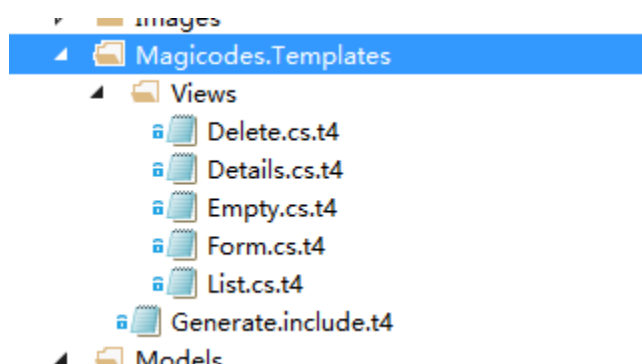
是否显示于侧边栏或顶部

菜单名称数字类型

从上图可以看出，简单的 CRUD 代码已经生成。但是投入到实际使用还有很多工作量，比如：

- 根据属性类型自动加载相应控件。（目前只做了部分，比如日历控件，下拉列表等尚未实现）
- 搜索尚未实现（如上一版一样，集成 OData 查询即可）
- 根据辅助生成特性来完善一些逻辑。比如这里明显是存在父子级关系的，那么我希望后面能够增加相应特性来生成更完善的代码。

不过目前这版，生成基础架构后，你可以基于生成的代码改改用了。不过，为了使生成的代码更加灵活，因此我将生成的 View 模板放在了 Magicodes.Admin 插件的 Magicodes.Templates 目录。如下图所示：



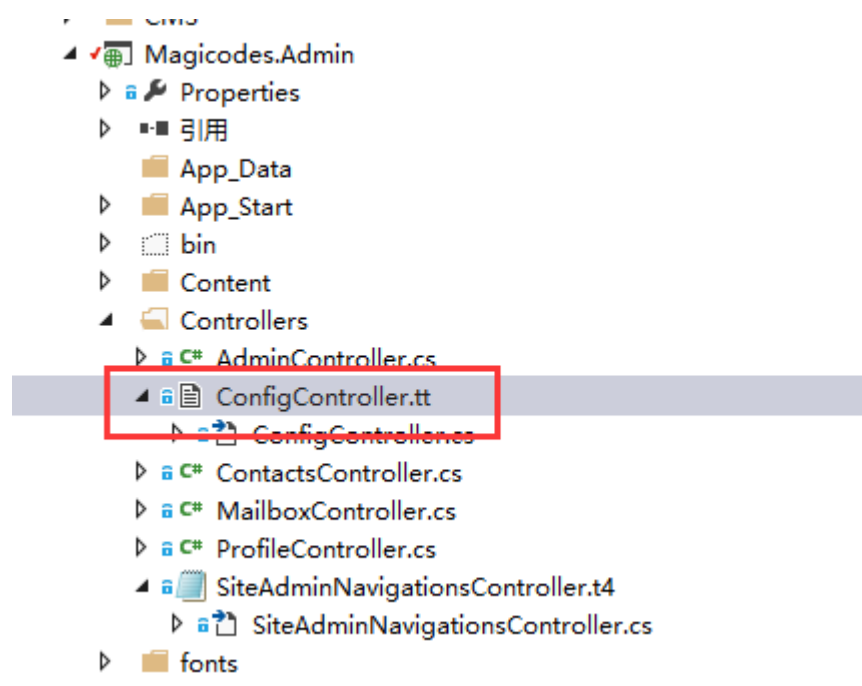
因此，换主题，改样式，你只要修改此处即可。

其实做到这一步，打通这条路子，我就耗费了不少时间（包括走了不少弯路）。希望有兴趣的朋友可以将我刚才提及的一些不完善的地方完善完善，我实在是心有余而力不足啊。

11.5 关于配置信息的生成

当然，这个在老的版本里已经实现了，新版本重构了，不过体验上会弱一点，比如保存没有提示等等。因为我真的是缺少时间。

首先我们来看看 T4 模板：



然后，我们再来看看具体代码：


```
<#@ include file="$(SolutionDir)\T4\Magicodes.T4\Templates\T4Helper.tt" #>
<#@ import namespace="Magicodes.Web.Interfaces.Config" #>
<#@ import namespace="Magicodes.Web.Interfaces.Config.Info" #>
<#@ output extension=".cs" encoding="utf-8" #>
<#
    var assembly=typeof(SiteConfigInfo).Assembly;
    var types=assembly.GetTypes().Where(p => p.IsClass && p.IsSubclassOf(typeof(ConfigBase)));
    var viewDirPath=
Magicodes.T4.Templates.MVC.T4MvcHelper.CreateMvcViewsDirectoryIfNotExist(Host.TemplateFile,"Config
");
    foreach(var config in types)
    {
        //是否忽略生成此类
        var
T4GenerationIgnoreAttribute=config.GetAttribute<Magicodes.Web.Interfaces.T4.T4GenerationIgnoreAttr
ibute>(false);
        if(T4GenerationIgnoreAttribute!=null) continue;
        var
descriptionAttr=config.GetAttribute<System.ComponentModel.DescriptionAttribute>(false);
        var title=descriptionAttr == null ? "" : descriptionAttr.Description;
        var typeName=config.Name;
        var typeFullName=config.FullName;
        var Propertys=config.GetT4PropertyInfos();
        #>
        <#@ include file="$(SolutionDir)\T4\Magicodes.T4\Templates\Config\Views.tt" #>
    }
    #>
    <#@ include file="$(SolutionDir)\T4\Magicodes.T4\Templates\Config\Controller.tt" #>
```

生成的类如下所示：



视图文件如下：

- ▶ Scripts
- ▶ Views
 - ▶ Admin
 - ▶ Config
 - AdminSiteConfigInfo.cshtml
 - MailConfigInfo.cshtml
 - SiteConfigInfo.cshtml
 - SystemConfigInfo.cshtml
 - Contacts
 - Home



管理员
系统管理

- 系统设置
- 系统管理

站点信息配置

系统管理 / 网站信息管理

保存 取消 返回

站点信息配置

网站名称: Magcodes.NET平台

网站网址: http://www.magcodes.net/


网站描述: Magcodes.NET - 聚合提供快速开发框架

网站关键词: 为2018年4月 - 起提供只能为站力服务。

SEO关键词: magcodes.NET, magcodes 快速开发框架

SEO关键词: magcodes.NET框架是一套操作框架，以方便网站的搭建，快速部署。

备案号: 京ICP备18011111号



管理员
系统管理

- 系统设置
- 系统管理

邮箱信息配置

系统管理 / 邮箱信息管理

保存 取消 返回

邮箱信息配置

SMTP地址: smtp.163.com

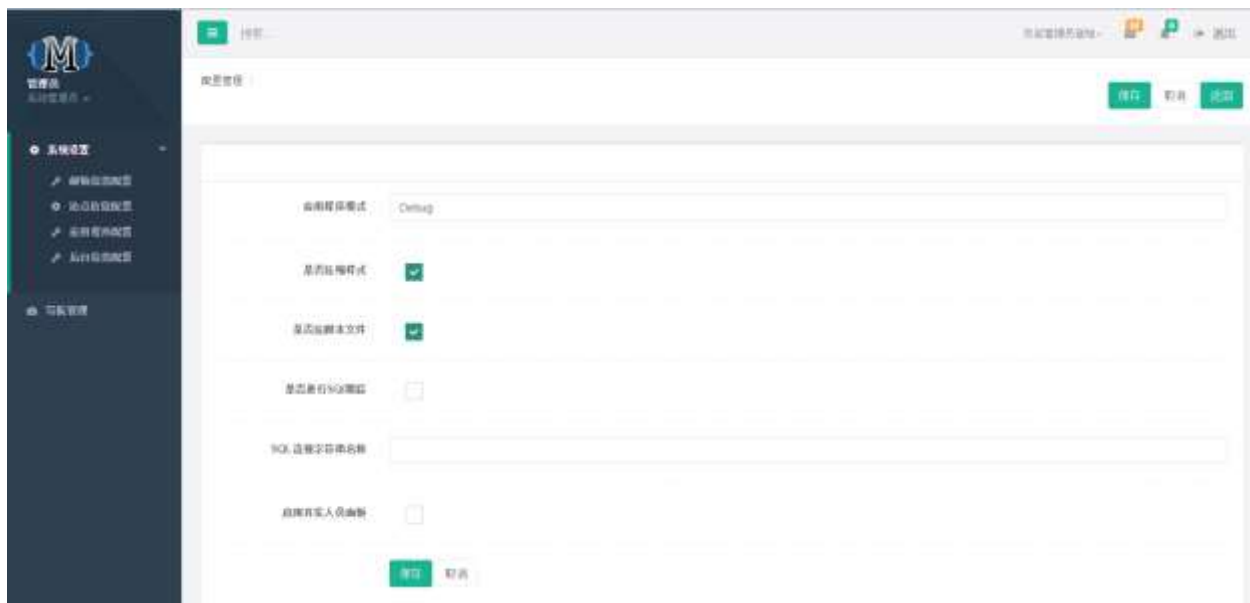
SMTP端口: 25

是否启用SSL: ☒

发件人邮箱: magcodes@outlook.com

发件人名称: magcodes.NET (QQ)

用户帐户: magcodes@outlook.com



12 SIGNALR

13 声明式认证（ASP.NET IDENTITY）

14 文档协议管理器

文档协议管理器用来告诉框架，该文件类型使用哪个 View 来展示（处理）。

14.1 原理

定义了 DocumentViewerController 控制器来处理文档显示，代码如下所示：

```
/// <summary>
/// 寻找默认的文档加载器
/// </summary>
[RoutePrefix("DocumentViewer")]
public class DocumentViewerController : PlusControllerBase
{
    [Route]
    [Route("Index")]
```

```

public ActionResult Index()
{
    //文件路径
    var filePath = Request["filePath"];
    //文件协议
    var contentType = Request["contentType"];
    if (string.IsNullOrEmpty(filePath))
        throw new HttpException("文件路径不能为空!");
    if (string.IsNullOrEmpty(contentType))
        throw new HttpException("contentType: 内容类型 (MIME 类型) 不能为空!");
    //获取协议
    var protocol =
ApplicationContext.DocumentsOpenProtocolManager.DocumentOpenProtocols.FirstOrDefault(p =>
p.ContentType.Equals(contentType, StringComparison.CurrentCultureIgnoreCase));
    if (protocol != null)
    {
        TempData["DocumentProtocolInfo"] = new DocumentProtocolInfo() { FilePath =
filePath, ContentType = contentType };
        //return RedirectToAction("Index", "PDFViewer", new { FilePath = filePath,
ContentType = contentType, pluginName = "Magicodes.PDFViewer" });
        return RedirectToAction(protocol.Action, protocol.Controller, new { pluginName =
protocol.PluginName, filePath = filePath, contentType = contentType });
    }
    //如果不存在, 则下载
    return File(filePath, contentType);
}
}

```

因此使用方式如下：

/DocumentViewer?contentType=[内容类型]&filePath=[访问路径]

如：

/DocumentViewer?contentType=application/pdf&filePath=/Attachments/upload/file/Magicodes.NET 框架说明文档.pdf

14.2 插件配置文件支持

自 V0.0.0.5 Beta 版开始，文档协议管理器已经重构，并且开始在插件配置文件中，允许定义文档协议。如：

```

<?xml version="1.0" encoding="utf-8" ?>
<PlusConfigInfo>
    <AssemblyType>MVC</AssemblyType>
    <MvcPlusType>MVC</MvcPlusType>
    <DocumentOpenProtocols>
        <DocumentOpenProtocol>
            <ContentType><![CDATA[application/pdf]]></ContentType>
            <Controller><![CDATA[PDFViewer]]></Controller>

```

```
<Action><![CDATA[Index]]></Action>
</DocumentOpenProtocol>
</DocumentOpenProtocols>
</PlusConfigInfo>
```

具体，您可以参考 Magicodes.PDFViewer 插件。

14.3 问题

由于时间问题，还有以下问题在实际使用中应该解决：

1. 类型和路径传参加密
2. 文件访问权限控制

14.4 PDF 查看器插件（MAGICODES.PDFVIEWER）

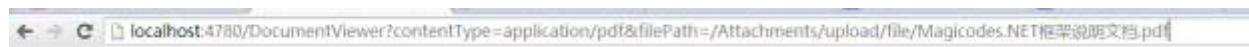
实现文档协议，现在以本插件为例子，控制器如下所示：

```
public ActionResult Viewer(DocumentProtocolInfo documentProtocolInfo)
{
    return View("Index", documentProtocolInfo);
}
public ActionResult Index()
{
    var data = TempData["DocumentProtocolInfo"] as DocumentProtocolInfo;
    return View(data);
}
```

视图以具体代码为准。使用了 PDFJS 插件。

打开链接如下：

<http://localhost:4780/DocumentViewer?contentType=application/pdf&filePath=/Attachments/upload/file/Magicodes.NET%E6%A1%86%E6%9E%B6%E8%AF%B4%E6%98%8E%E6%96%87%E6%A1%A3.pdf>



15 数据访问

15.1 REPOSITORY 模式

在许多应用程序中，业务逻辑访问来自很多数据存储的数据，例如 SQL Server 数据库，SharePoint 列表，WebServices 等等。直接访问的数据会带来以下问题：

1. 重复的代码
2. 编码时更易于出错
3. 业务数据的弱类型
4. 难以集中与数据相关的策略，如缓存
5. 无法从外部依赖孤立地测试业务逻辑

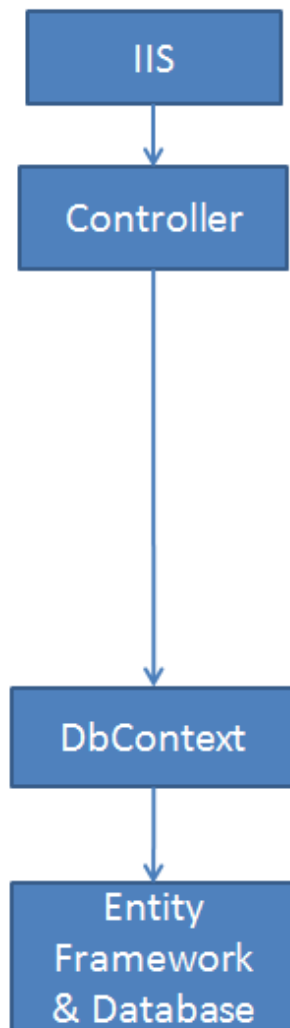
Repository 模式旨在创建数据访问层和应用程序的业务逻辑层之间的抽象层，能够帮助改变数据存储隔离应用程序，并可以方便的进行自动化单元测试或测试驱动开发（TDD）。

使用 Repository 模式实现以下一个或多个目标：

- 可以最大化的分离数据层，以便支持单元测试。
- 可以从任何位置访问数据源，并能够制定一致的访问规则和逻辑。
- 可以为数据源制定集中的策略，比如缓存策略。
- 可以通过数据和服务访问逻辑分离业务逻辑来提高代码的可维护性和可读性。
- 如果要使用强类型的业务实体，这样就可以在编译时就发现问题。
- 可以将一个行为与相关数据关联起来。例如，要计算字段或强制执行的实体中的数据元素之间的复杂关系或业务规则。
- 可以应用领域模型来简化复杂的业务逻辑。

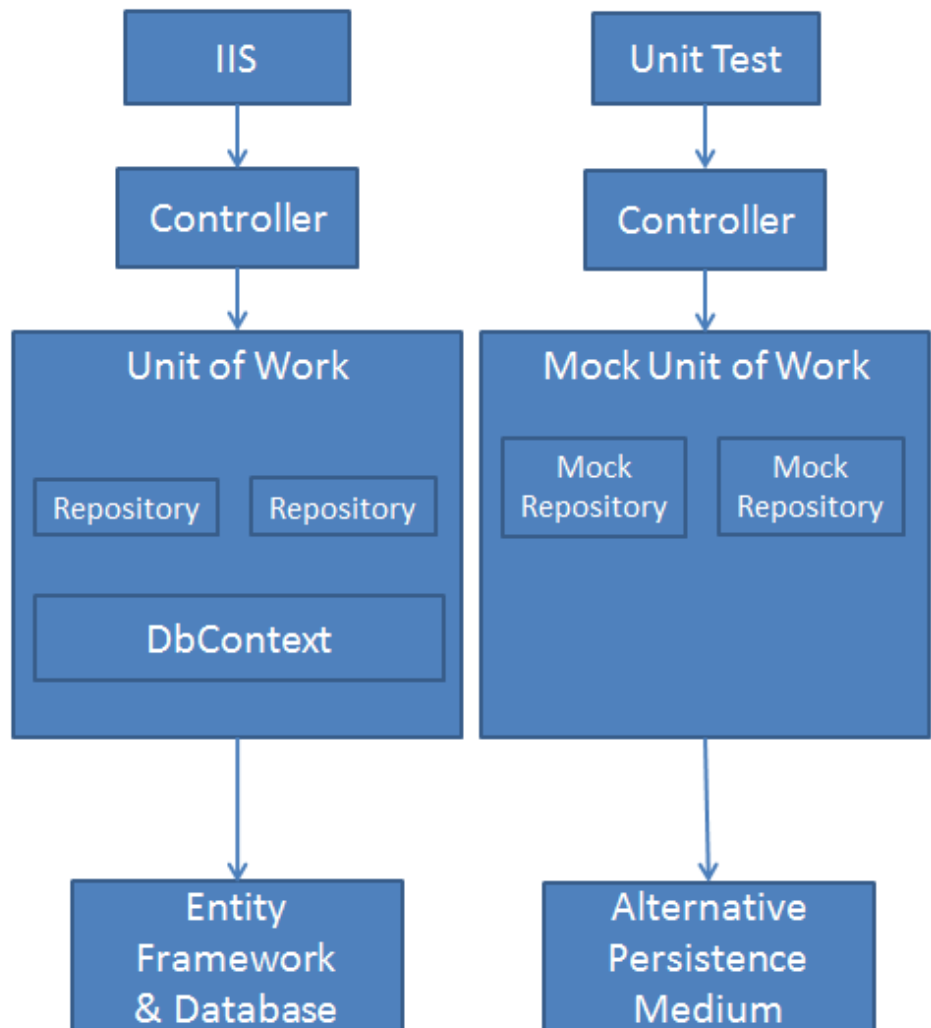
No Repository

Direct access to database context from controller.



With Repository

Abstraction layer between controller and database context. Unit tests can use a custom persistence layer to facilitate testing.



15.2 CODE FIRST

15.3 EF 基于代码的迁移

在之前，每次更改我们都需要编写 SQL 变更脚本以及相关的描述与记录，并且将这些脚本签入代码，然后还得告诉团队成员他们需要运行的脚本。最后，在部署项目的下一个版本时，我们还得根据历次的更改脚本来提供基于生产库的脚本。按照这种方式，我们需要花大量的精力来维护我们的数据库，而且不利于扩展，比如项目从 My Sql 迁移到 SQL Server 时。

EF 在 4.3 以及后续的版本中就提供了基于代码的迁移，也就是我们从之前的方式转变成了代码驱动来更改数据库的结构的方式。

本人一直使用这种方式来开发，体验相当不错。因此 Magicodes.NET 推荐使用 Code First 以及使用基于代码的迁移。

15.3.1 概要

数据迁移就相当于数据更改的补丁。比如数据模型增加了一个类，那么就相当于新增了一个 Table，我们需要将这个更改添加到数据库，故此我们需要生成自定义的代码迁移，能够帮我们执行这个更改。

15.3.2 主要命令

Enable-Migrations -EnableAutomaticMigrations -Force 启用自动迁移

Add-Migration [类名] 添加迁移

Enable-Migrations 启用迁移

Update-Database 更新迁移

Update-Database -Force -Verbose 强制更新迁移

15.3.3 关闭自动迁移

默认情况下，我们可以运行

Enable-Migrations -EnableAutomaticMigrations -Force

命令来启用自动迁移，然后就可以看到其会产生一个新的配置类。在这个配置中，我们需要禁用自动迁移，这不仅是我个人的喜好，而且也是大部分的一种习惯吧。我们需要明确迁移，虽然 Code First 自动迁移和手动迁移可以混合使用，但是其并不能保证每个自动迁移都正确完成，因此明确每次迁移内容以及迁移的正确性是一个很必要的工作。

15.3.4 生成迁移

- 打开【程序包管理器控制台】（工具——>NuGet程序包管理器——>程序包管理器控制台）
- 选择默认项目模型
- 执行添加迁移命令。
运行命令：Add-Migration CreateDb
- 更新数据库。
运行命令：Update-database

15.4 COMMONBUSINESSMODELBASE

15.5 API

15.5.1 消息（MessageDataRepositoryBase）

15.5.2 后台菜单（SiteAdminNavigationRepositoryBase）

16 响应式布局

17 【前端】模块化

18 【前端】MVVM

19 前端插件

19.1 ODATAGIRD