2014-11-22

MAGICODES.NET 框架说明文档

V1

李文强

MAGICODES.NET 团队

目录

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。

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。

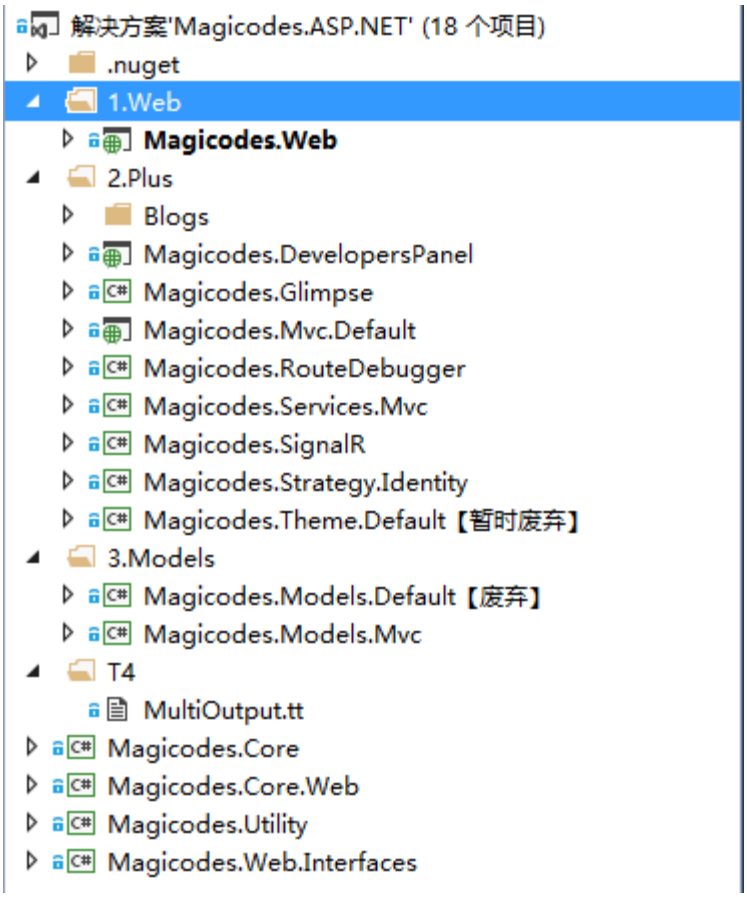
- **PM> Install-Package Magicodes.Core**
- **PM> Install-Package Magicodes.Core.Web**
- **PM> Install-Package Magicodes.Utility**
- **PM> Install-Package Magicodes.Web.Interfaces**
- **PM> Install-Package Magicodes.Web.Default**

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。

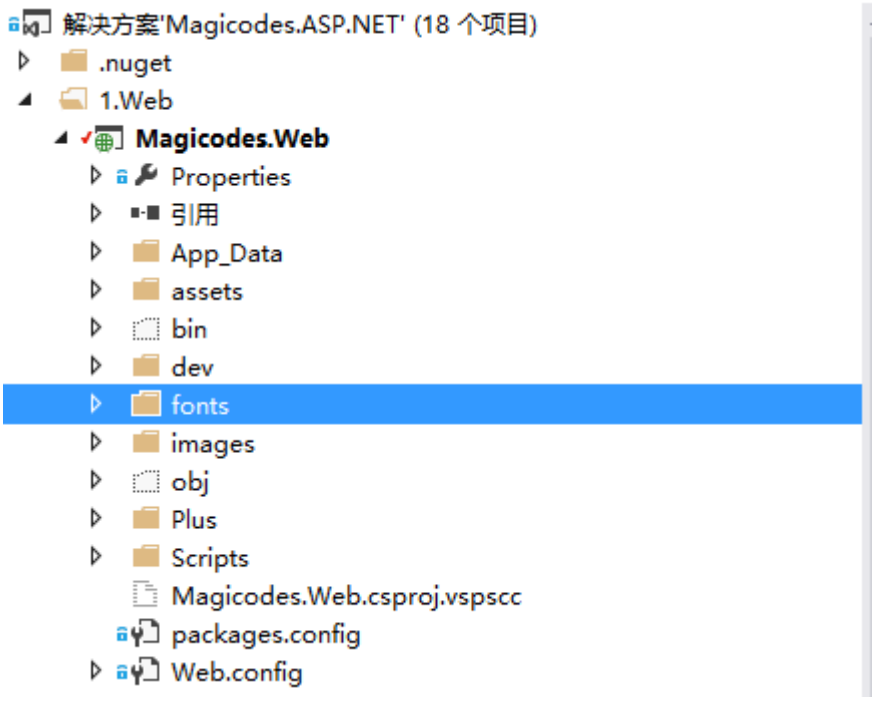
2.2 包含的内容

具体请以下载的源码为准：



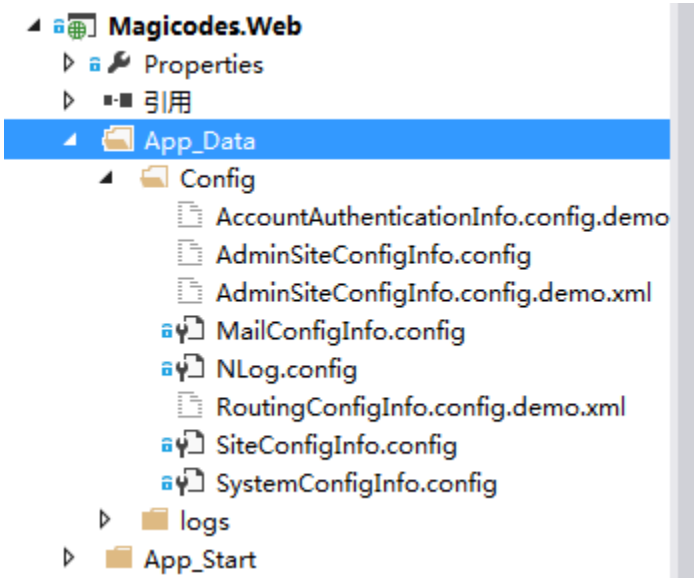
2.2.1 Magicodes.Web

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。

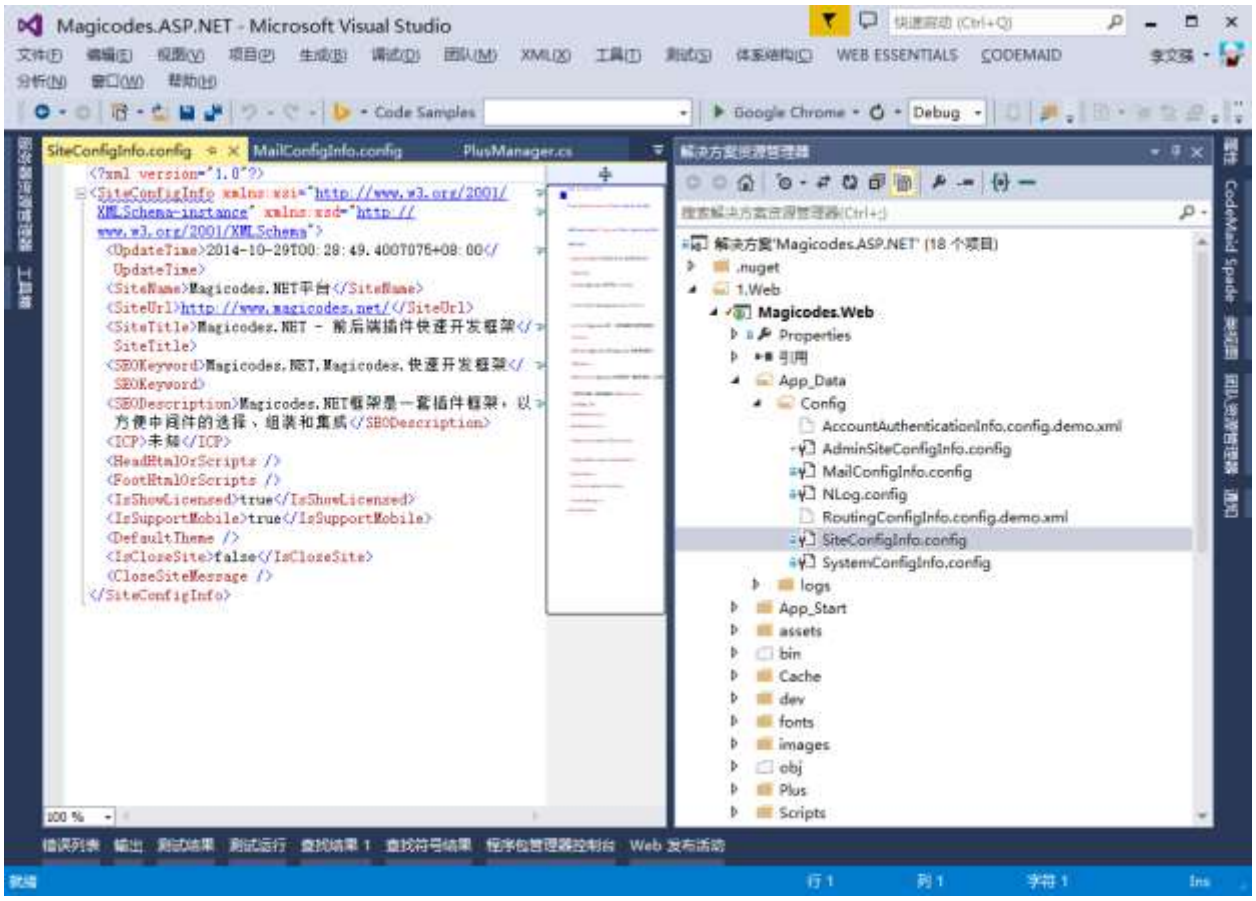


其中，

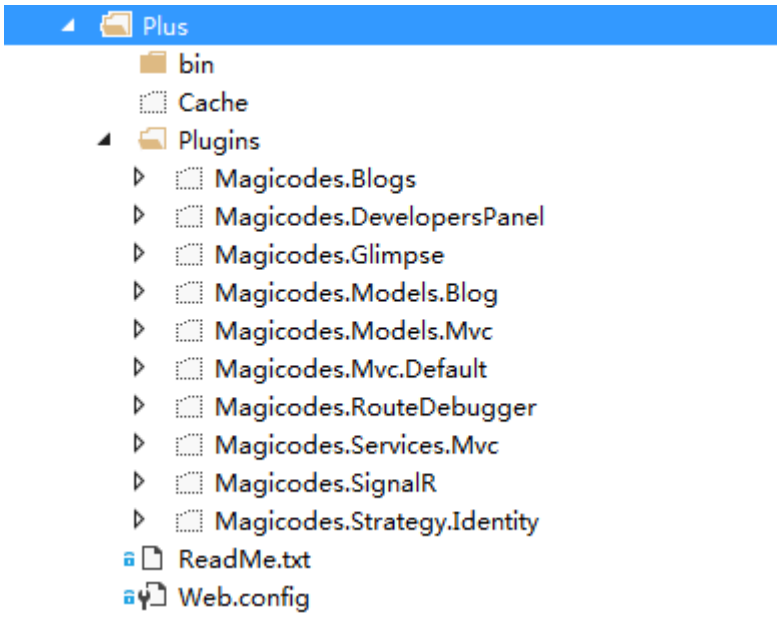
- App_Data：网站配置文件以及日志等文件存放目录



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。

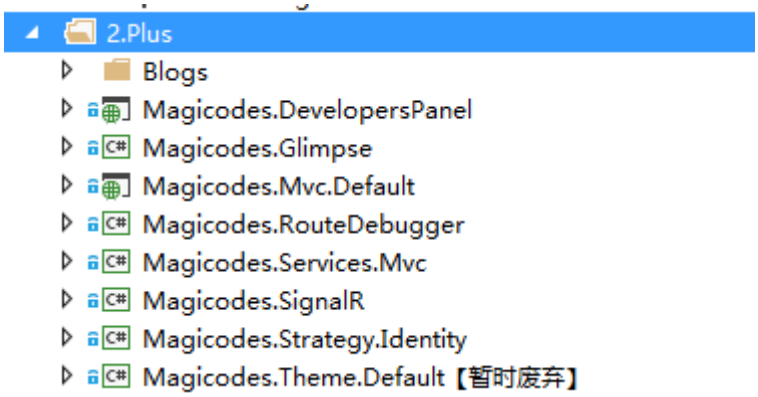


注意：当你重新编译插件项目时，会自动往此目录添加插件内容，因此一般情况，请不要更改此目录。

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

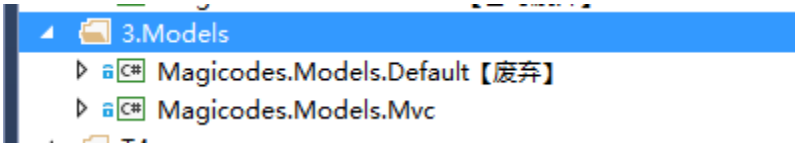
2.2.2 Plus 解决方案目录

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）：



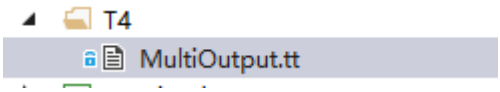
2.2.3 Models 解决方案目录

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

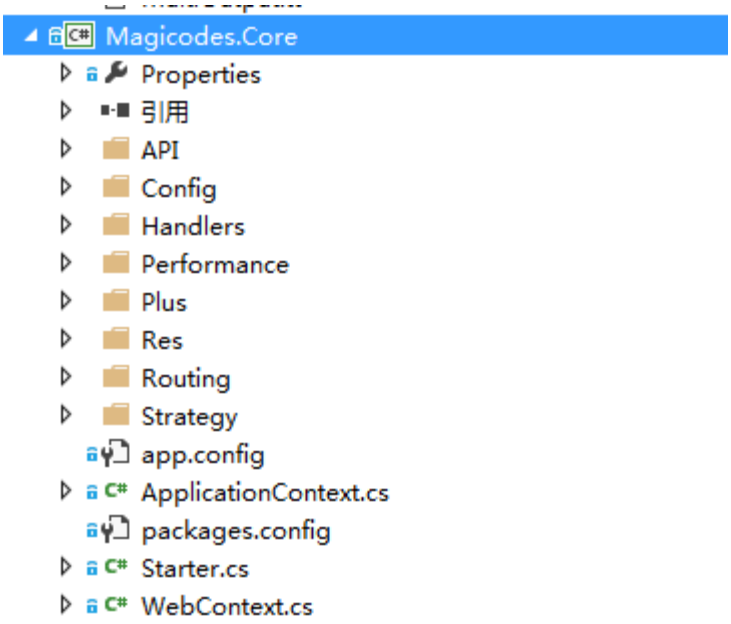
T4 解决方案目录用于存放通用的 T4 模板。如：



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。

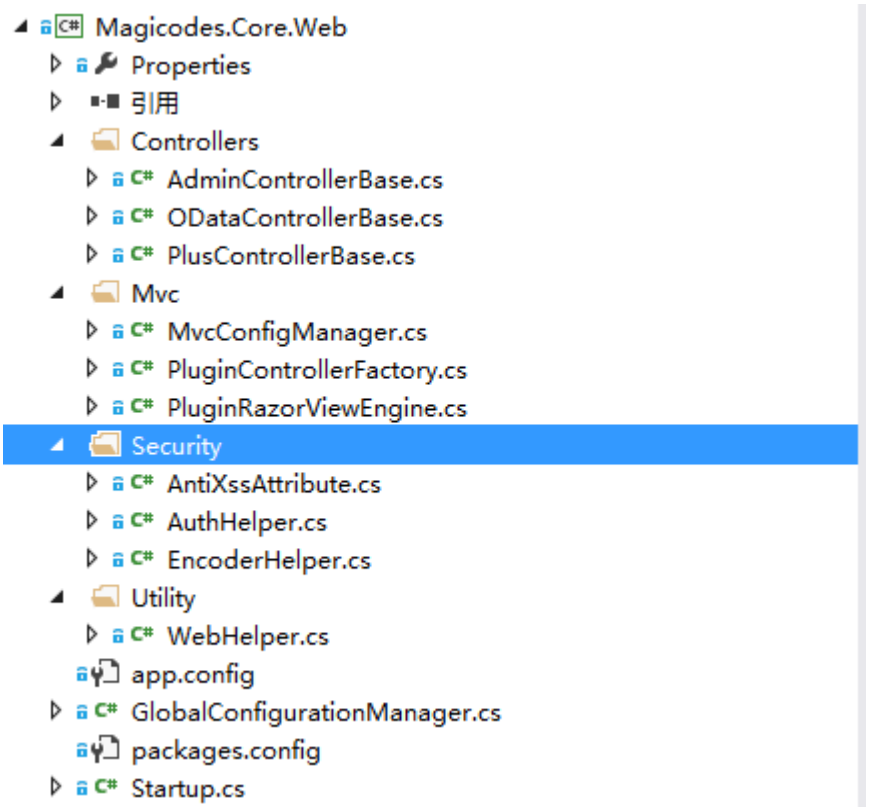
2.2.5 Magicodes.Core

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：



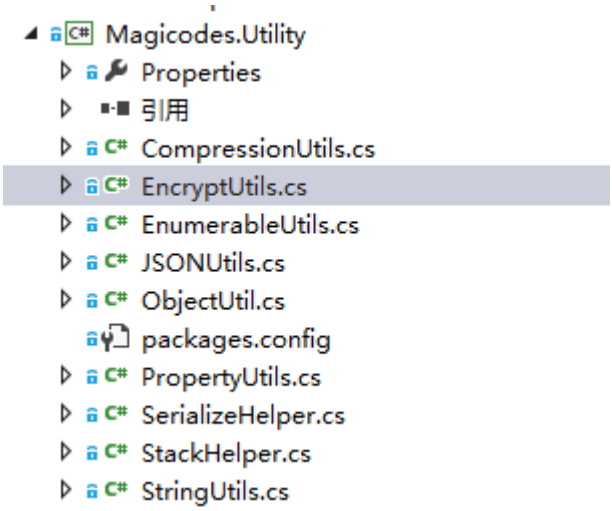
2.2.6 Magicodes.Core.Web

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。



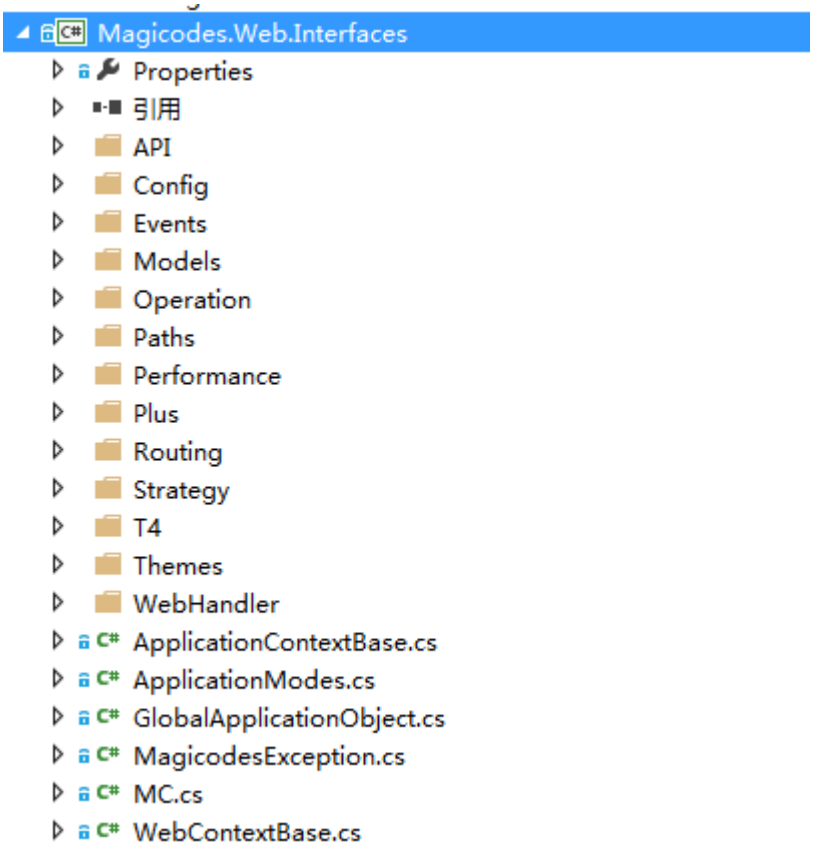
2.2.7 Magicodes.Utility

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

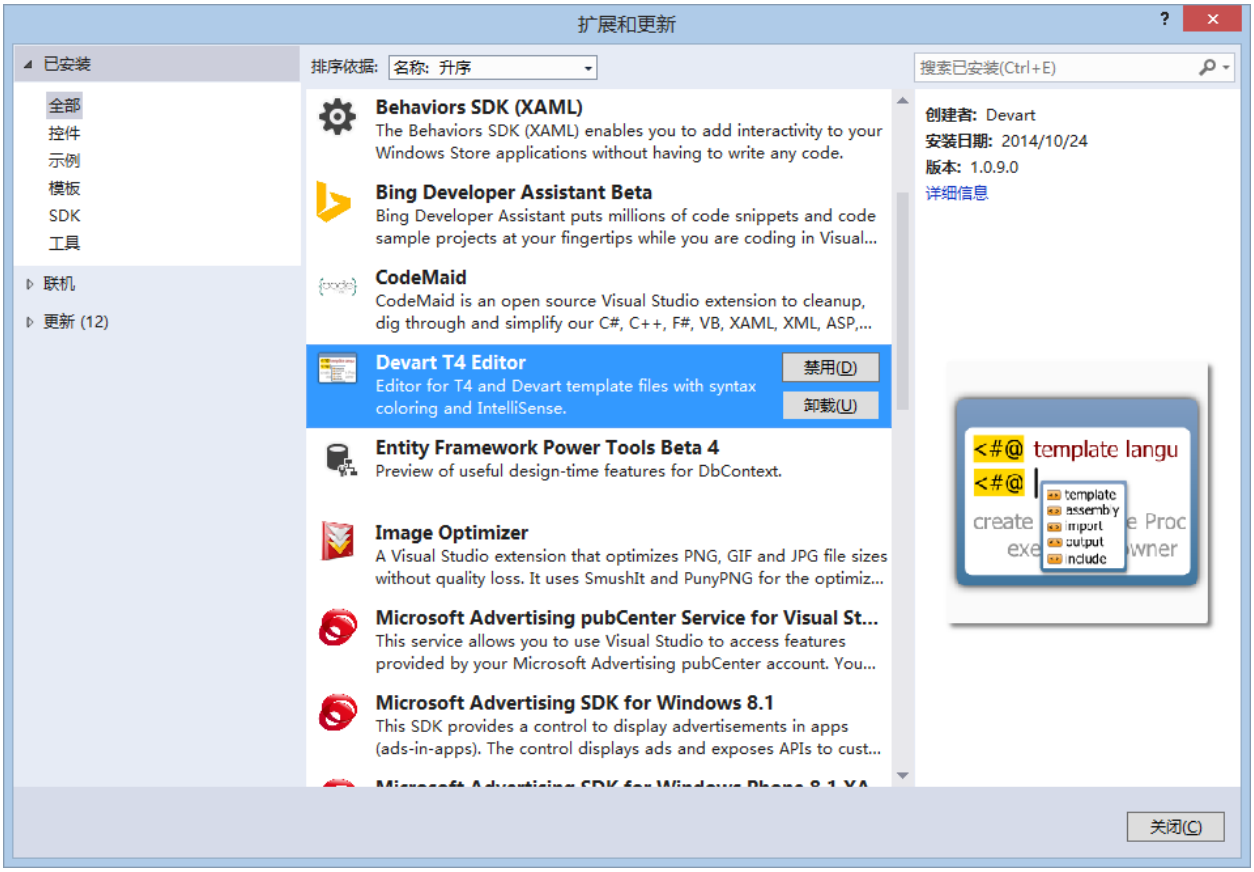
必选扩展。

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,
    /// <summary>
    /// MVC门户程序集
    /// </summary>
    MVCHome = 7
}
```

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）。

8. **MVC 门户程序集**

用于承载 MVC 门户代码，在运行时只允许存在一个门户程序集。

下面就以主要的几种类型进行解说。

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

QQ 群：85318032

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。

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。

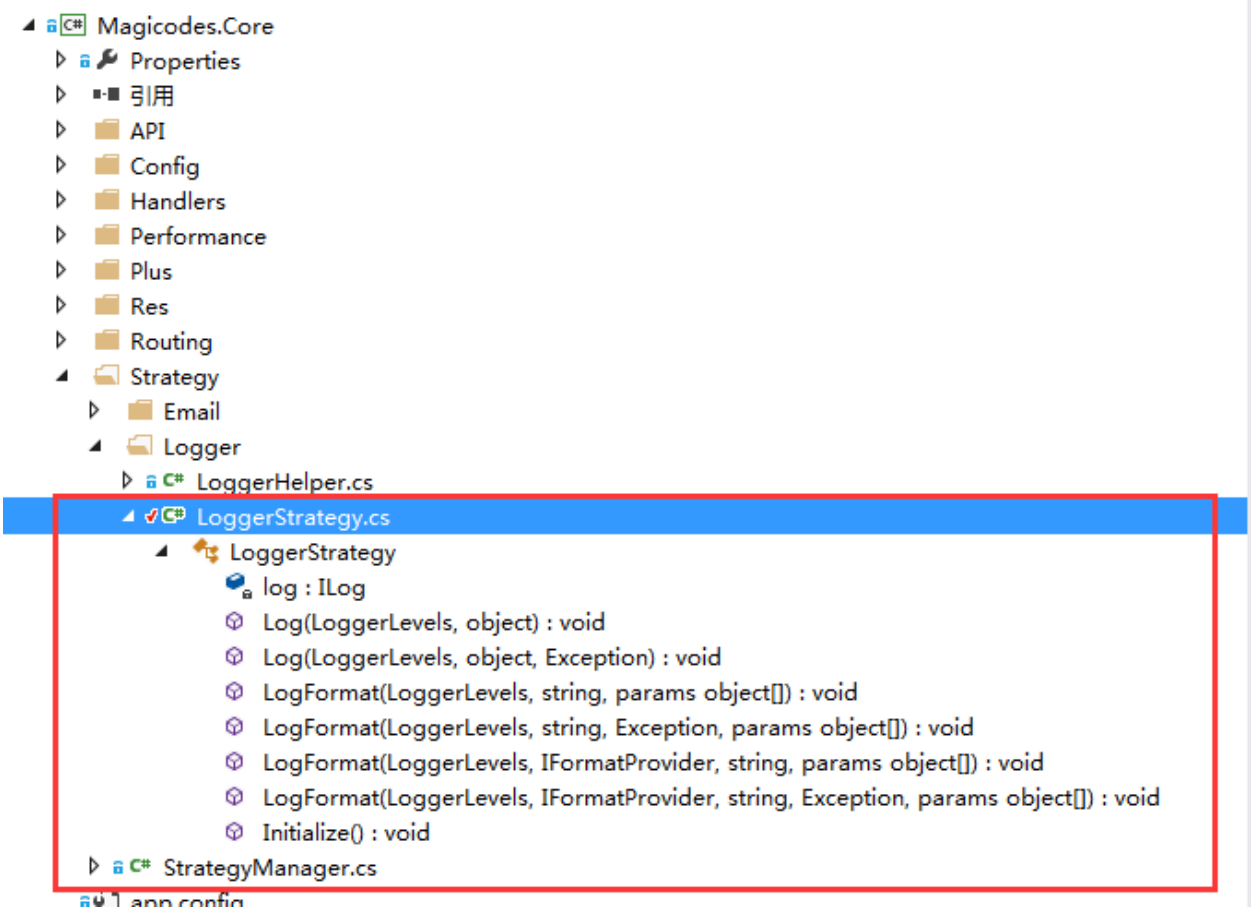
对于框架来说，将常用的策略纳入框架是很有利于插件调用的。目前，Magicodes 框架提供了以下策略：

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

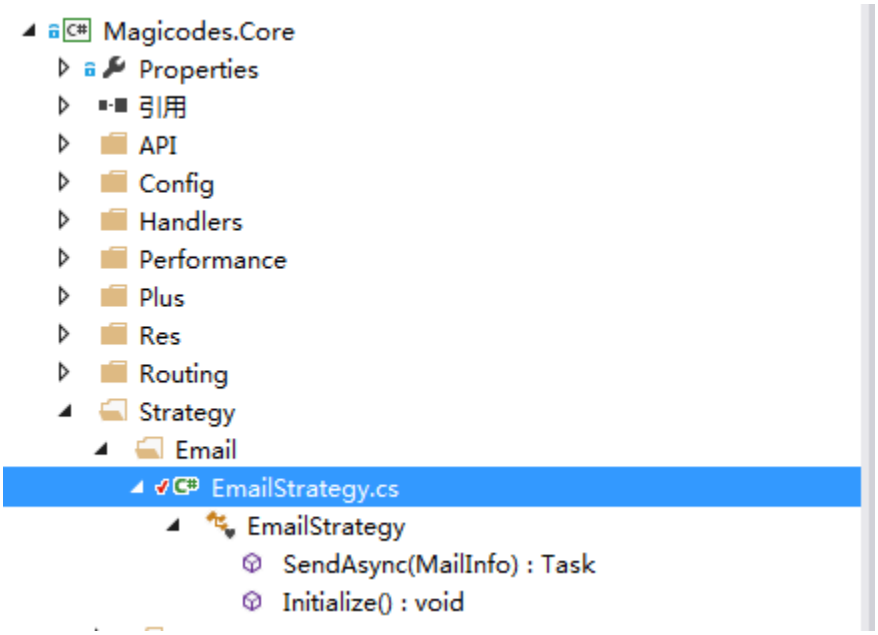
```
/// <summary>
/// 日志策略
/// </summary>
0 个引用|李文强|4 个更改
public class LoggerStrategy : LoggerStrategyBase
{
    ILog log;
    0 个引用|李文强|3 个更改
    public override void Log(LoggerLevels loggerLevels, object message)...

    0 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);
```

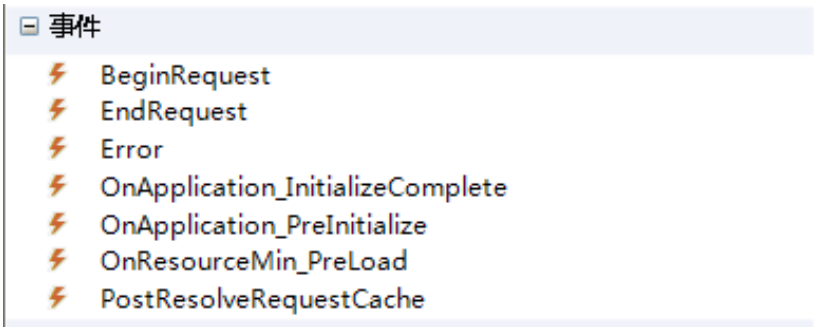
```
        }

        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：

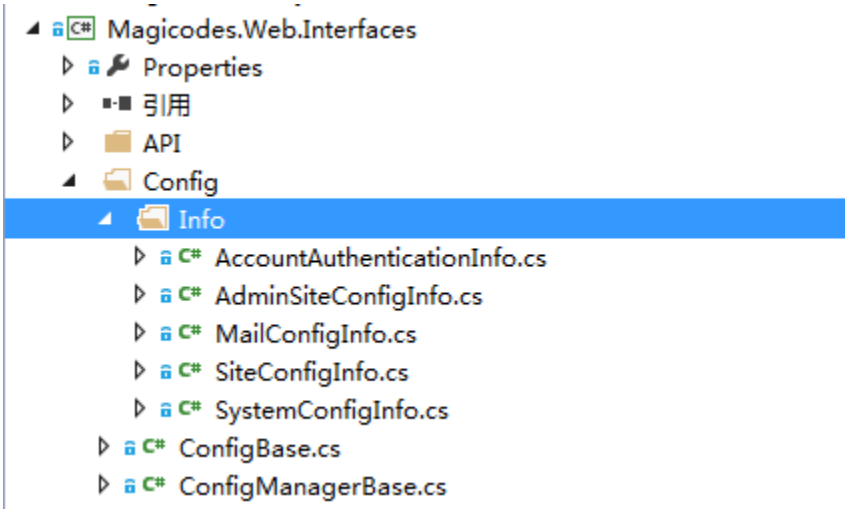
```
[assembly: OwinStartup(typeof(Magicodes.Core.Web.Startup))]
namespace Magicodes.Core.Web
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            GlobalConfigurationManager.AppBuilder(app);
        }
    }
}
```

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 = (IAppBuilder)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：



3.5 插件商店

4 插件案例

4.1 监控面板（Magicodes.Glimpse）

4.2 路由调试（Magicodes.RouteDebugger）

4.3 声明式认证（Magicodes.Strategy.Identity）

5 移动设备支持

6 MVC

6.1 MVC 插件

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

6.1.2 博客插件（Magicodes.Blogs）

6.2 路由

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：

13 声明式认证（ASP.NET IDENTITY）

14 文档协议

14.1 PDF 查看器插件

15 数据访问

15.1 REPOSITORY

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

16 响应式布局

17 【前端】模块化

18 【前端】MVVM