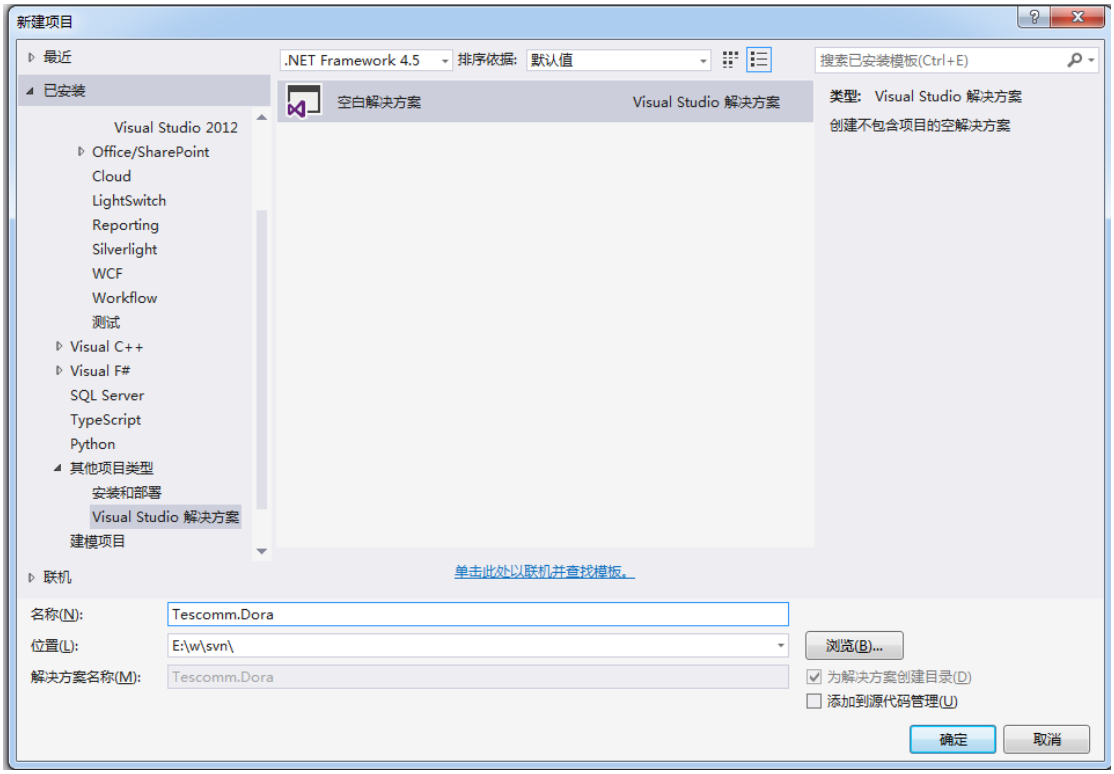
泰合佳通基础应用平台开发指南

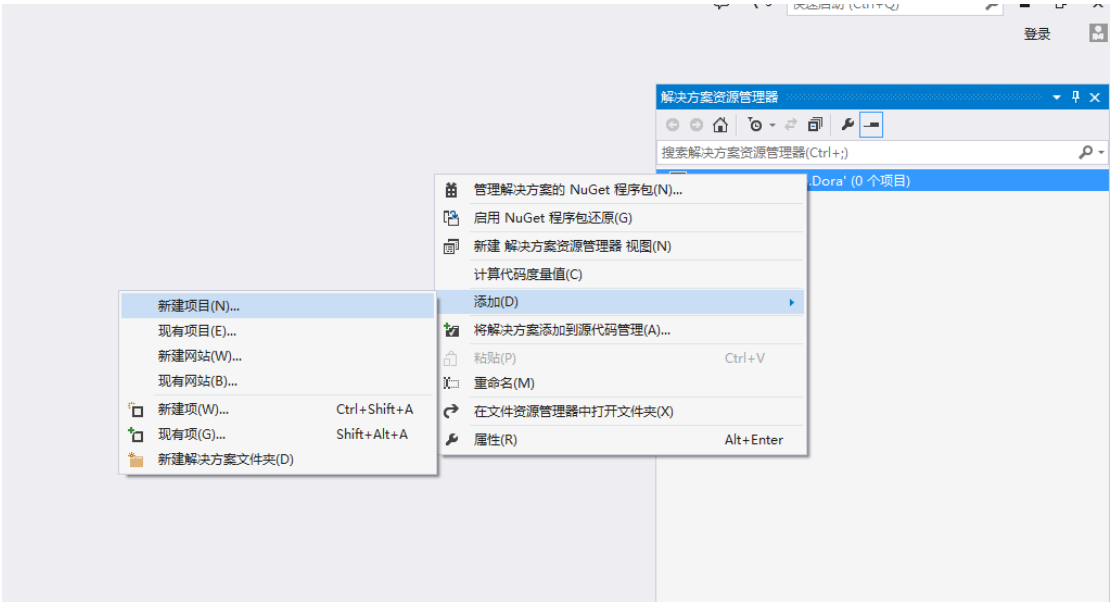
版本	时间	修改人	内容
0.2.0924	2015-09-24	黄庆鑫	增加 web 应用的创建步骤及 Nuget 源安装步骤, 更新数据库组件接口
0.1.0821	2015-08-21	黄庆鑫	初稿

创建应用解决方案

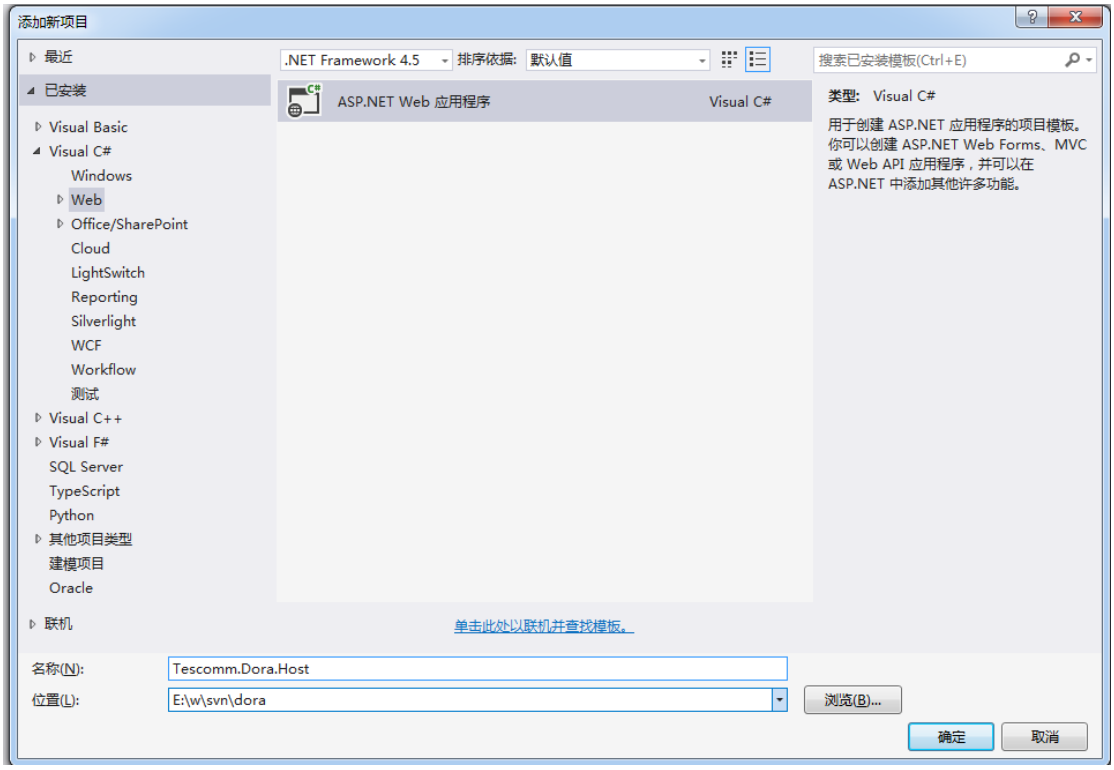
1、通过 vs 文件菜单创建空白解决方案，填好解决方案名称和路径后确定。

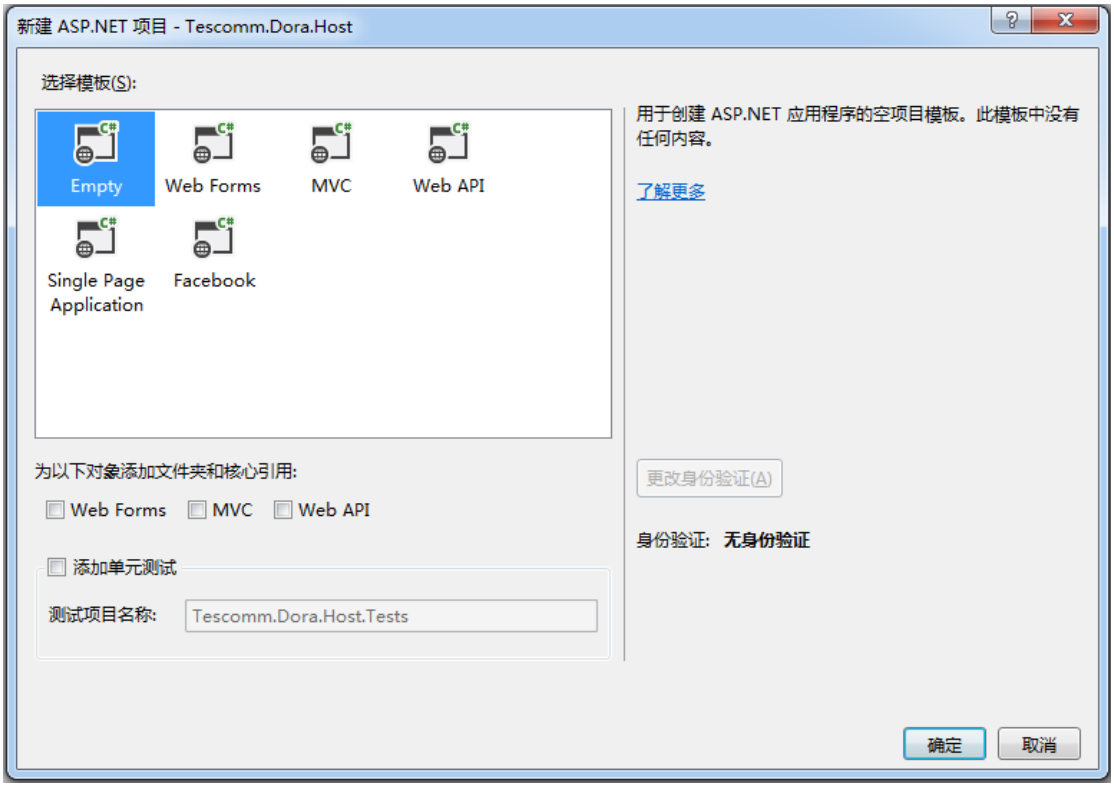


2、web 宿主项目

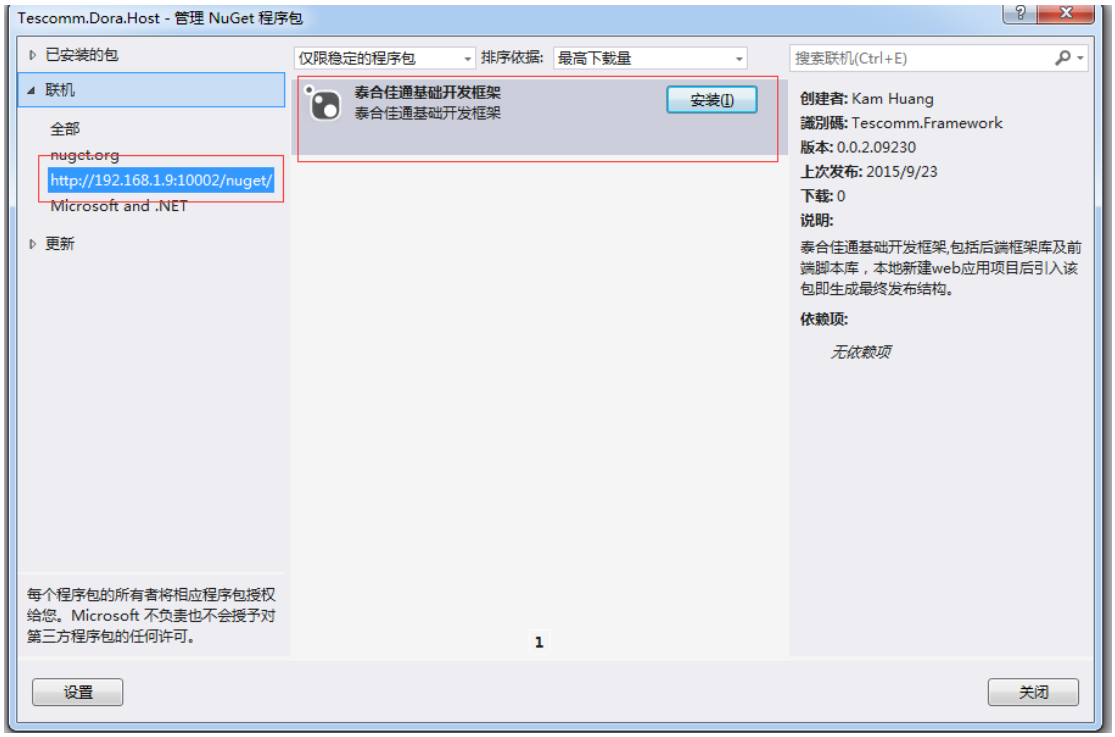


创建空白 web 应用程序

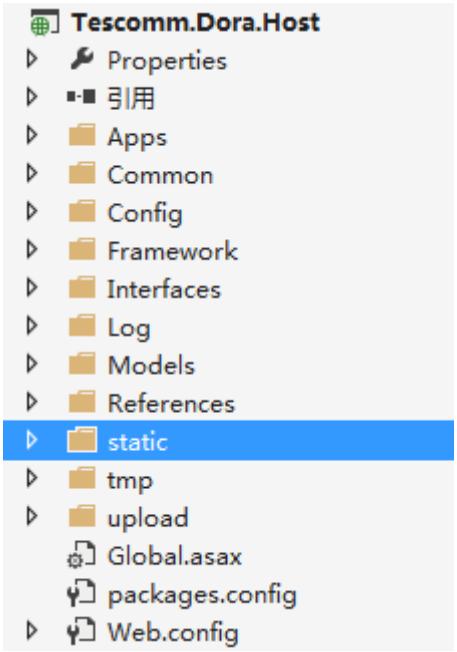

添加新项目
最近
已安装
Visual Basic
Visual C#
Windows
Web
Office/SharePoint
Cloud
LightSwitch
Reporting
Silverlight
WCF
Workflow
测试
Visual C++
Visual F#
SQL Server
TypeScript
Python
其他项目类型
建模项目
Oracle
联机
.NET Framework 4.5
排序依据:
默认值
搜索已安装模板(Ctrl+E)
ASP.NET Web 应用程序
Visual C#
类型: Visual C#
用于创建 ASP.NET 应用程序的项目模板。你可以创建 ASP.NET Web Forms、MVC 或 Web API 应用程序，并可以在 ASP.NET 中添加其他许多功能。
单击此处以联机并查找模板。
名称(N):
Tescomm.Dora.Host
位置(L):
E:\w\svn\dora
浏览(B)...
确定
取消


新建 ASP.NET 项目 - Tescomm.Dora.Host
选择模板(S):
Empty
Web Forms
MVC
Web API
Single Page Application
Facebook
用于创建 ASP.NET 应用程序的空项目模板。此模板中没有任何内容。
了解更多
为以下对象添加文件夹和核心引用:
Web Forms
MVC
Web API
添加单元测试
测试项目名称:
Tescomm.Dora.Host.Tests
更改身份验证(A)
身份验证: 无身份验证
确定
取消

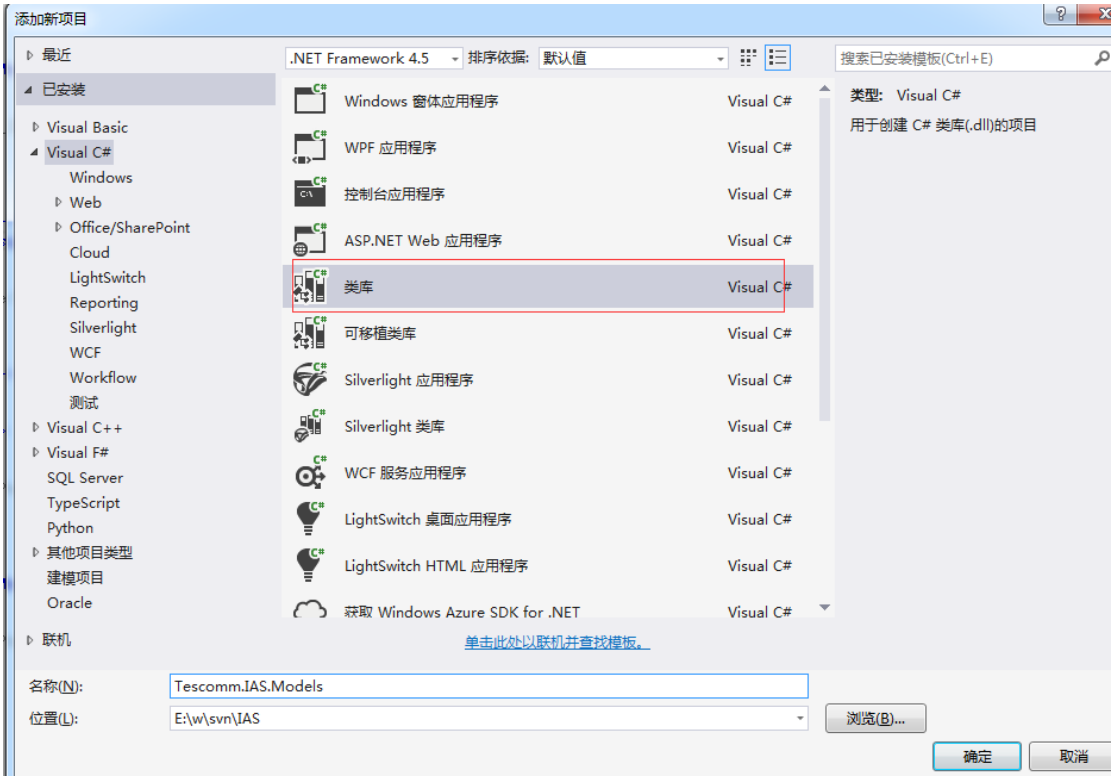
引入框架资源



安装完成，查看项目目录结构。



3、新建应用业务模型项目



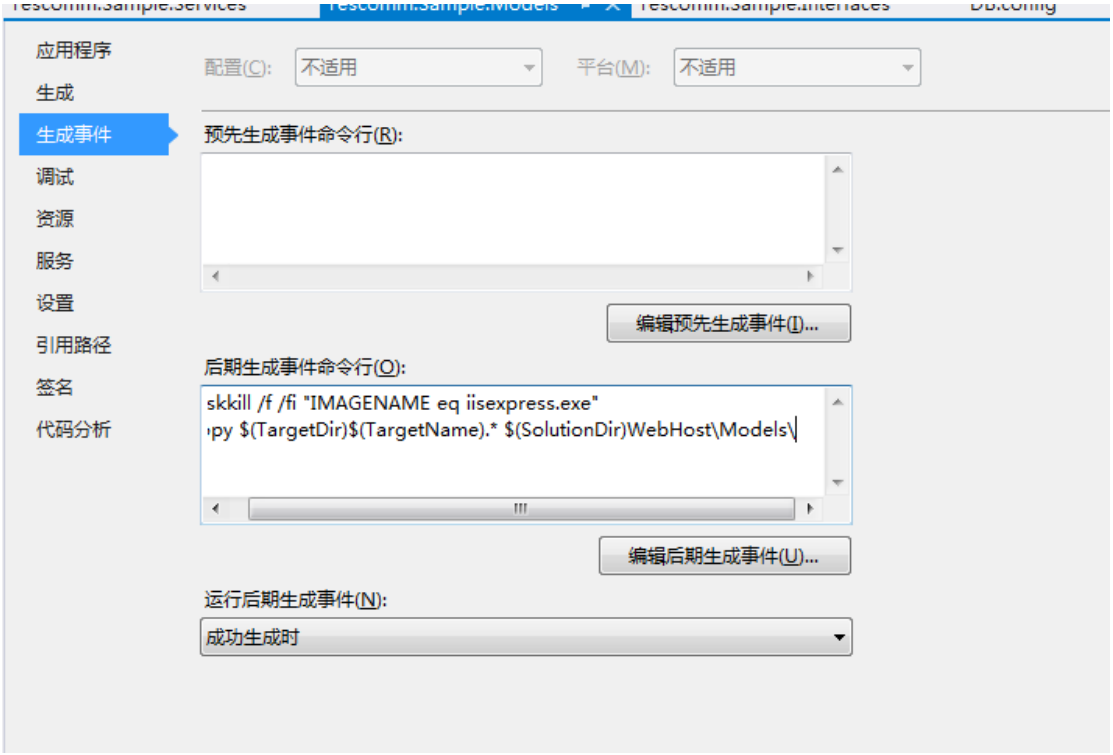
设置项目生成事件，将生成的 dll 文件输出宿主网站中对应文件目录下。

在生成事件命令行输入：

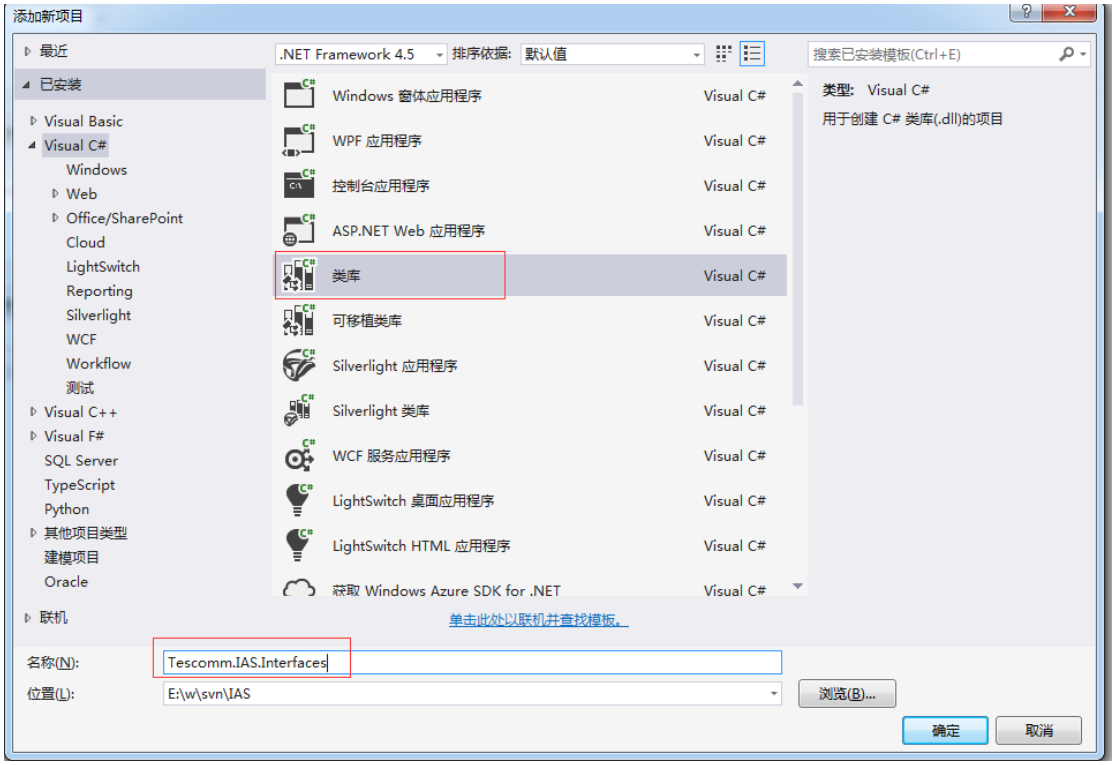
预先生成事件命令：taskkill /f /fi "IMAGENAME eq iisexpress.exe"

后期生成事件：copy $(TargetDir)$(TargetName).* $(SolutionDir)WebHost\Models\

注：WebHost 为最初创建的 web 应用项目的名称，根据实际项目进行替换。



4、新建服务接口项目和设置输出路径及项目自定义引用的“复制到本地”项设置为 False

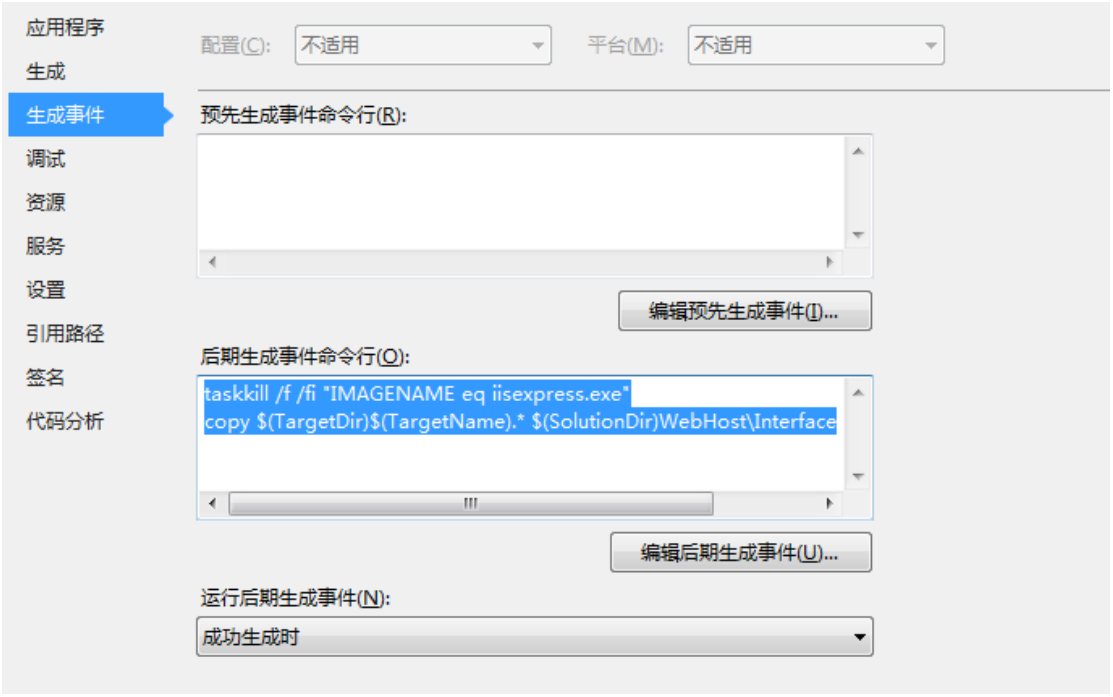


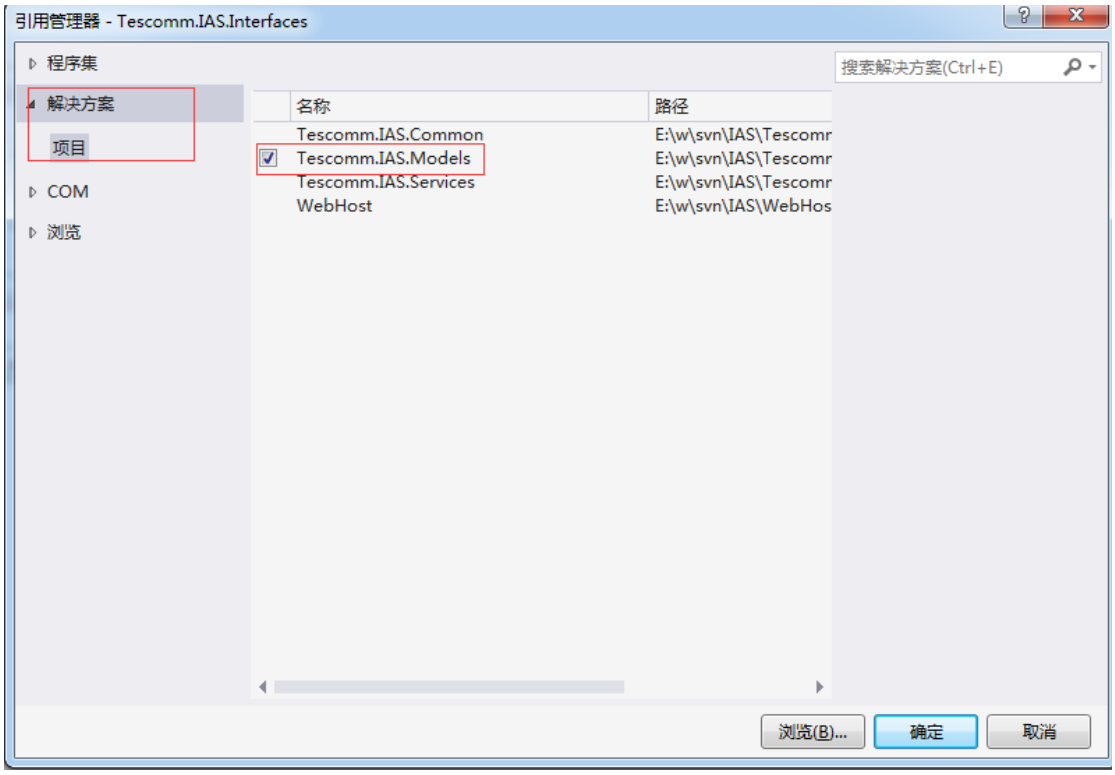
设置生成事件，将 dll 输出到 webhost 下的 interfaces 目录下

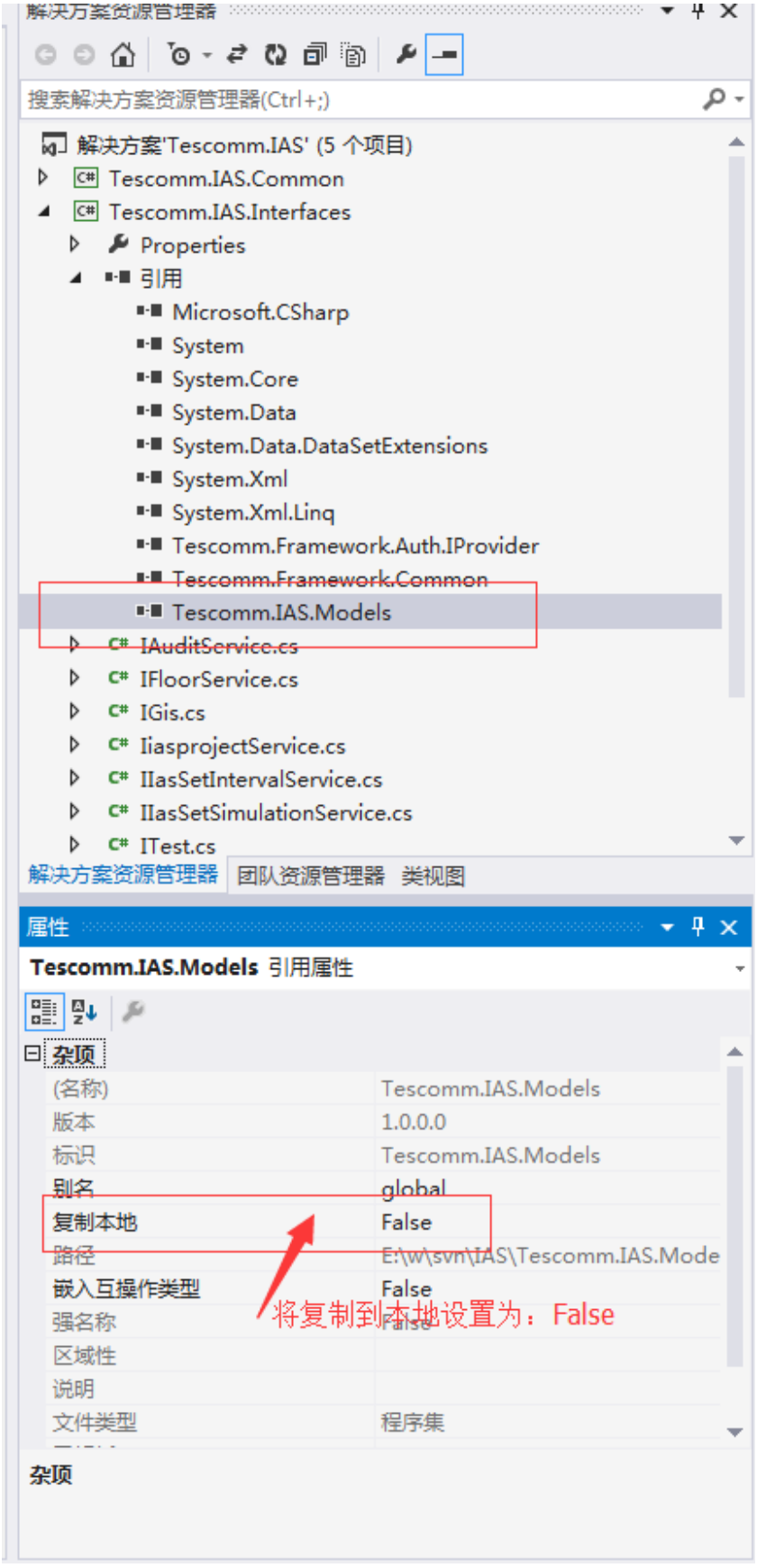
执行命令行：

预先生成事件命令：taskkill /f /fi "IMAGENAME eq iisexpress.exe"

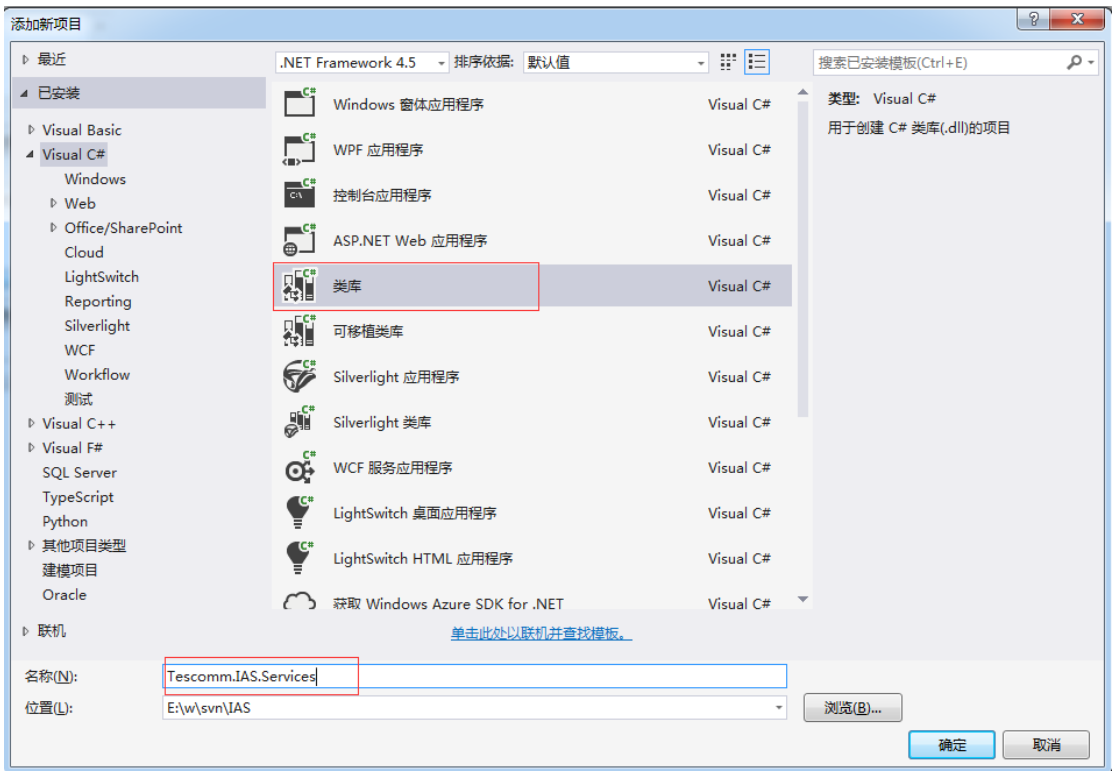
后期生成事件：copy $(TargetDir)$(TargetName).* $(SolutionDir)WebHost\Interfaces\


应用程序
生成
生成事件
调试
资源
服务
设置
引用路径
签名
代码分析
配置(C): 不适用
平台(M): 不适用
预先生成事件命令行(R):
编辑预先生成事件(I)...
后期生成事件命令行(O):
taskkill /f /fi "IMAGENAME eq iisexpress.exe"
copy $(TargetDir)$(TargetName).* $(SolutionDir)WebHost\Interface
编辑后期生成事件(U)...
运行后期生成事件(N):
成功生成时


引用管理器 - Tescomm.IAS.Interfaces
程序集
解决方案
项目
COM
浏览
搜索解决方案(Ctrl+E)
名称
路径
Tescomm.IAS.Common
Tescomm.IAS.Models
Tescomm.IAS.Services
WebHost
浏览(B)...
确定
取消



5、新建应用服务实现项目，和设置输出路径及项目引用

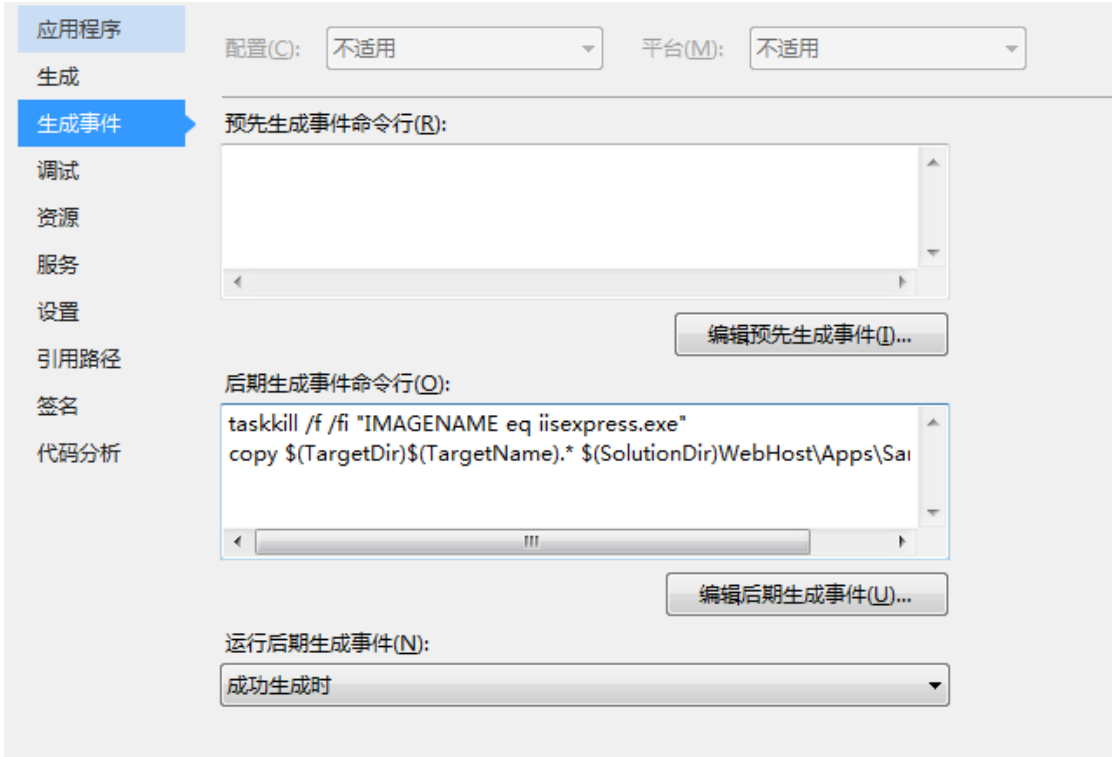


设置输出目录，建服务程序集 dll 输出到 webhost 下的 app 目录中对应的模块文件夹中，命令行：

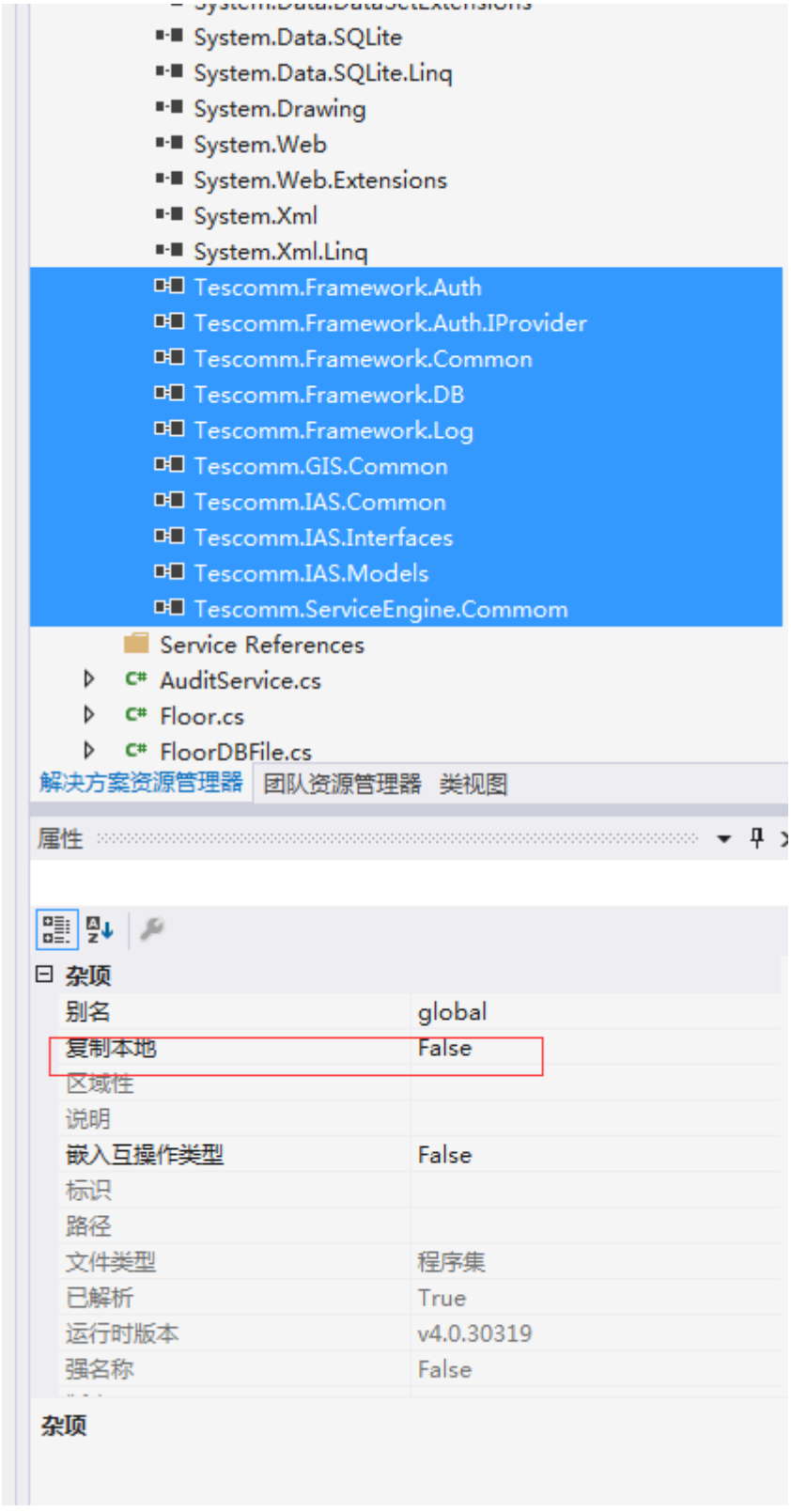
预先生成事件命令：taskkill /f /fi "IMAGENAME eq iisexpress.exe"

后期生成事件：copy $(TargetDir)$(TargetName).* $(SolutionDir)WebHost\Apps\Sample\V1\

备注：Sample 为模块名称，V1 为该模块对应的版本号，默认为 1，如有多个版本请建立对应目录。



本项目模块名称对应 Service.json 文件里配置的模块名称，版本号默认为 1



宿主网站目录说明

Apps:各项目 Servcie 结尾的程序集 dll，根据 Service.json 配置建立对应的模块目录

Common：项目通用程序集存放目录，一般对应为.Common 结尾的 dll

Interfaces：服务接口程序集 dll，

Models：业务模型实体程序集 dll

Framework：基础框架程序集 dll

Static：项目的静态资源目录，所有的 html、js、css 及图片等文件均存放于此。

配置文件说明

Initialize.config：项目启动时应用初始化的配置信息

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="initializeConfiguration" type="Tescomm.Framework.Common.InitializeConfiguration,Tescomm.Framework.Common"/>
  </configSections>
  <initializeConfiguration>
    <initializers>
      <initializer name="dbConfig" type="Tescomm.Framework.DB.DBManager,Tescomm.Framework.DB" loader="Load" >
        <parameters>
          <parameter name="path" type="string" value="\DB.config"/>
        </parameters>
      </initializer>
      <initializer name="logConfig" type="Tescomm.Framework.Log.LogManager,Tescomm.Framework.Log" loader="LoadConfig" >
        <parameters>
          <parameter name="file" type="string" value="\Log.config"/>
        </parameters>
      </initializer>
      <initializer name="cacheConfig" type="Tescomm.Framework.Cache.CacheManager,Tescomm.Framework.Cache" loader="Load" >
        <parameters>
          <parameter name="file" type="string" value="\Cache.config"/>
        </parameters>
      </initializer>
    </initializers>

  </initializeConfiguration>
</configuration>
```

初始化项ID　待初始化的程序集及名称空间　初始化方法名称　初始化的参数配置　参数名称　参数值

Db.config:数据组件配置信息

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="dbConfiguration" type="Tescomm.Framework.DB.DBConfiguration,Tescomm.Framework.DB"/>
  </configSections>
  <dbConfiguration>
    <connectionStrings>
      <add name="mysqlConn" connectionString="server=192.168." providerName="Tescomm.Framewo">...</add>
    ...
      <add name="conn1" connectionString="Data Source=ctestsrc;User ID=ctuni_oauth;Password=ctunioauth;"
           providerName="Tescomm.Framework.DB.Oracle.OracleDataProvider,Tescomm.Framework.DB.Oracle"/>
    </connectionStrings>
    <files>
      <file name="Permission" src="\Oracle\Permission.config" mappingType="Tescomm.Framework.DB.CommandTemplateCollection,Tescomm.Framework.DB" />
      <file name="PermissionModel" src="\Oracle\Permission.config" mappingType="Tescomm.Framework.DB.ModelTemplateCollection,Tescomm.Framework.DB" />
      <file name="OAuth" src="\Oracle\OAuth_db.config" mappingType="Tescomm.Framework.DB.CommandTemplateCollection,Tescomm.Framework.DB" />
      <file name="OAuthModel" src="\Oracle\OAuth_db.config" mappingType="Tescomm.Framework.DB.ModelTemplateCollection,Tescomm.Framework.DB" />
      <file name="FileService" src="\Oracle\FileService.config" mappingType="Tescomm.Framework.DB.ModelTemplateCollection,Tescomm.Framework.DB" />
      <file name="FileServiceCMD" src="\Oracle\FileService.config" mappingType="Tescomm.Framework.DB.CommandTemplateCollection,Tescomm.Framework.DB" />
    </files>
    <dbCommands>
      <dbCommand id="cmd1" db="conn1" >
        <!--动态条件示例-->
        <text>
          <![CDATA[      SELECT sysdate from dual      ]]>
        </text>
        <!--动态条件语句定义
        dynamic:动态条件节点
        prepend: 预置值，当有条件成立是附加到Text包含的文本之后
      -->
        <dynamic prepend=" where ">...</dynamic>
      </dbCommand>
    </dbCommands>
    <models>
      ...
      ...<!--字段配置，
              属性说明：type:实体对应数据类型
              id: 实体属性名称
              name: 数据表中对应字段名称
              isKey: 主键标识，
              visiable: select时是否显示该字段，如果实体属性不存在数据表字段中，则需建该值设置为“false”-->...
      <model id="log_model" db="mysqlConn" table="sys_log">
        <fields>
          <field id="ID" name="id" type="String" isKey="true" identity="@@IDENTITY"  />
        </fields>
      </model>
    </models>
  </dbConfiguration>

</configuration>
```

连接字符串配置

包含文件节点

文件对应节点解析器

Log.config：日志文件配置信息

对应 Log4net 配置文件格式

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

<configuration>
  <configSections>
    <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,log4net"/>
  </configSections>
  <log4net>
    <root>
      <appender-ref ref="WarningLogFileAppender"/>
      <appender-ref ref="ErrorLogFileAppender"/>
      <appender-ref ref="DebugLogFileAppender"/>
      <appender-ref ref="InfoLogFileAppender"/>
      <appender-ref ref="ColoredConsoleAppender"/>
    </root>
    <logger name="ServiceEnine">
      <level value="ALL"/>
      <appender-ref ref="WarningLogFileAppender"/>
      <appender-ref ref="ErrorLogFileAppender"/>
      <appender-ref ref="DebugLogFileAppender"/>
      <appender-ref ref="InfoLogFileAppender"/>
    </logger>

    <logger name="DB_Logger">

      <level value="ALL" />

      <!--<appender-ref ref="AdoNetAppender_MySql" />-->
      <appender-ref ref="AdoNetAppender_Oracle" />
    </logger>
    <appender name="AdoNetAppender_" type="log4net.Appende">...</appender>
    <appender name="AdoNetAppender_" type="log4net.Appende">...</appender>

    <appender name="WarningLogFileA" type="log4net.Appende">...</appender>
    <appender name="ErrorLogFileApp" type="log4net.Appende">...</appender>
    <appender name="DebugLogFileApp" type="log4net.Appende">...</appender>
    <appender name="InfoLogFileAppe" type="log4net.Appende">...</appender>


    <!--调试模式，输出日志到控制台-->
    <appender name="ColoredConsoleA" type="log4net.Appende">...</appender>
  </log4net>

</configuration>
```

Cache.confg:缓存组件配置信息

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="cacheSetting" type="Tescomm.Framework.Cache.CacheConfiguration,Tescomm.Framework.Cache"/>
  </configSections>
  <cacheSetting>
    <caches>                    redis服务配置文件                                基于redis实现的缓存实现程序集
      <add name="redis" file="\redis.config" provider="Tescomm.Framework.Cache.Redis.CacheProvider,Tescomm.Framework.Cache.Redis"  />
    </caches>
  </cacheSetting>
</configuration>
```

REST 服务配置

在 WebHost 项目中的 Config 目录下打开 Service.json 文件，增加如下配置节

```
{
    "Apps":{
        "Sample":{
            "Namespace":"Tescomm.Open.Services"
            ,"Serializer":{
                            "Request":"Tescomm.Framework.Serializer.JSONSerializer,Tescomm.Framework.Serializer"
                            ,"Response":"Tescomm.Framework.Serializer.JSONSerializer,Tescomm.Framework.Serializer"
                        }
            ,"ConfigFile":""
            ,"Auth":false
        },
        "Manage":{
        "Namespace":"Tescomm.Manage.Services"
            ,"Serializer":{
                            "Request":"Tescomm.Framework.Serializer.JSONSerializer,Tescomm.Framework.Serializer"
                            ,"Response":"Tescomm.Framework.Serializer.JSONSerializer,Tescomm.Framework.Serializer"
                        }
            ,"ConfigFile":"Manage.config"
            ,"Auth":false
        }
    }
    ,"Cache":{"Provider":"Tescomm.Framework.Cache.Redis.CacheProvider,Tescomm.Framework.Cache.Redis","Config":"cache.config"}
    ,"AppRoot":"Apps"
    ,"Prefix":""
    ,"ClientHost":"http://localhost:16648,http://localhost:16646"
    ,LoginUrl:"/Sample/Sample/ShowPhoto"
    ,AuthorizeURL:"/Open/OAuth/Authorize"
    ,"ClientID":"Manage"
    ,"Auth":{"Provider":"Tescomm.Framework.Auth.Provider.SharkProvider,Tescomm.Framework.Auth.Provider","Config":""}
    ,"SpecialRoutes":{"/":"GoDefault:Tescomm.ServiceEngine.SpecialRequst","/_authorizecallback":"AuthorizeCallback:Tescomm.ServiceEngine.SpecialRequst","/__a
                      ,"/__Service":"ShowService:Tescomm.ServiceEngine.SpecialRequst"}
}
```

Apps：服务模块配置信息

Sample、Manage：自定义的模块服务名称，配置文件内唯一，

Namespace：服务接口实现所在的程序集名称空间，

Serializer：请求和输出参数的序列化器，保持默认即可。

ConfingFile：模块启动初始是所读取的配置文件信息，如果没有则为空。

应用 Auth：设置该模块是否使用鉴权服务，如果启用，则访问模块里的所有方法都必须通过鉴权认证后才能执行；如果未鉴权通过将自动跳转到鉴权服务的登录页，登录成功并获得授权后方能继续调用该服务。

Cache：配置缓存模块的具体实现方式

AppRoot：配置各个模块服务的 dll 所存放的目录，

Prefix：定义请求 url 路径的前缀，如无则留空。

ClientHost：预先进行跨域请的网站域名或 IP。

LoginUrl：默认访问宿主网站的一个页面路径，该路径将不受鉴权约束

AuthorizeURL：鉴权服务 URL，如果访问某个服务时需要认证则浏览器将自动跳转到路径进行用户权限的认证。默认设置为本机的鉴权服务(“/Open/OAuth/Authorize”),如有其他独立部署的鉴权服务请修改到对应的请求路径。

ClientID：鉴权服务分配给本项目 ID，对应项目管理的的 AppID。

Auth：鉴权组件实现配置，保持默认即可

SpecialRouters：服务引擎对特殊 URL 响应方法，键内容为请求的 URL，值为响方法:名称空间,程序集

PublicAction:服务公开方法，在此列表中的方法只要当前的令牌有效则可以配访问到，而不会在进一步检查是否有授权。

编写服务方法

返回字符串

说明：服务方法接收参数名为 Name 的参数并返回字符串。

1、定义接口方法

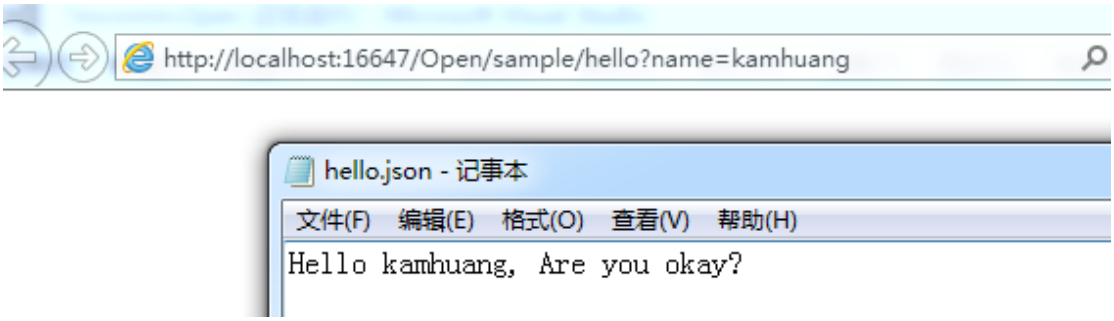
```
namespace Tescomm.Open.Interfaces
{
    0 个引用
    public interface ISample
    {
        0 个引用
        string Hello(string name);

    }
}
```

2、实现接口方法

```
[Serializable]
0 个引用
public class Sample:ISample
{
    1 个引用
    public string Hello(string name)
    {
        return "Hello " + name + ", Are you okay?";
    }
}
```

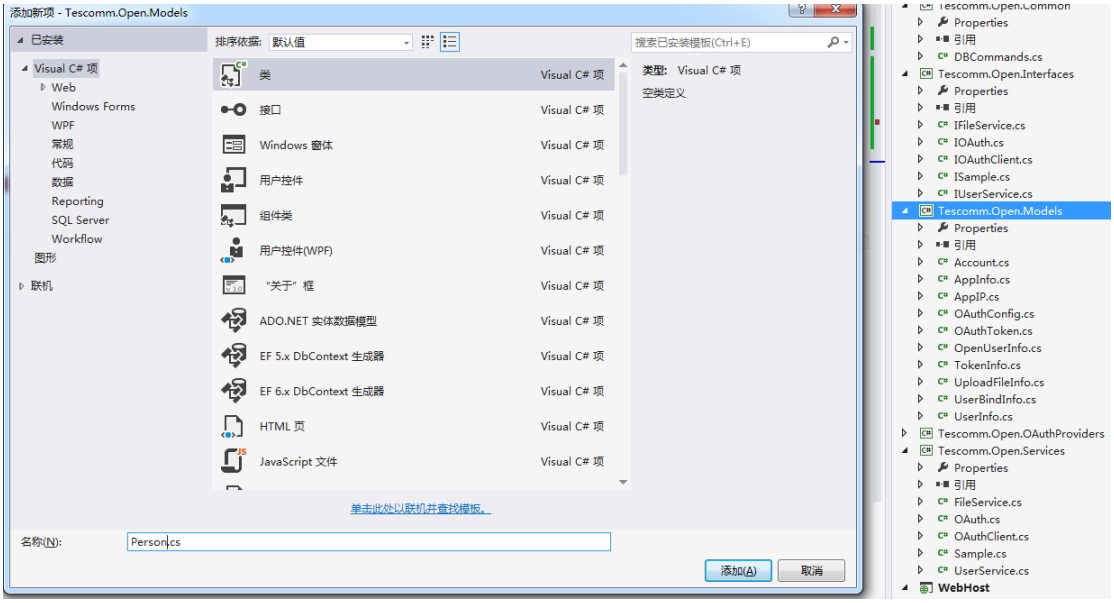
将服务实现类标记为可序列化！！

3、编译查看结果



返回业务模型

说明：将接收的参数 Name 和 Age 赋值给 Person 实例并返回。
在 Models 项目中定义业务模型 Person

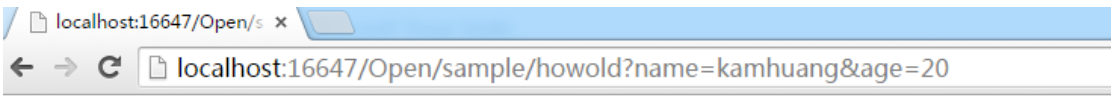



```
namespace Tescomm.Open.Models
{
    0 个引用
    public class Person
    {
        0 个引用
        public string Name { get; set; }
        0 个引用
        public int Age { get; set; }
    }
}
```

定义服务接口方法

```
[Serializable]
0 个引用
public class Sample:ISample
{
    1 个引用
    public string Hello(string name)
    {
        return "Hello " + name + ", Are you okay?";
    }
    1 个引用
    public Person HowOld(string name, int age)
    {
        return new Person() { Name = name, Age = age };
    }
}
```

编译查看请求结果




```
{
  "Name": "kamhuang",
  "Age": 20
}
```

返回流

说明：将图片流返回到浏览器。

定义服务方法

```
using Tescomm.Open.Models;
using System.IO;
namespace Tescomm.Open.Interfaces
{
    1 个引用
    public interface ISample
    {
        1 个引用
        string Hello(string name);

        1 个引用
        Person HowOld(string name, int age);

        1 个引用
        Stream ShowPhoto();
    }
}
```

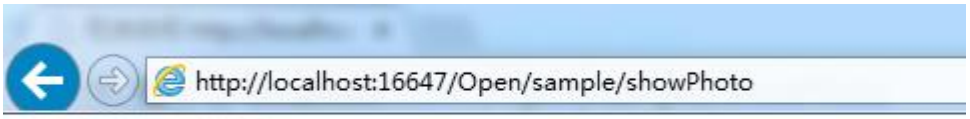
实现方法，读取文件。（如果为下载文件类型，这将 Image 替换为 File 即可）

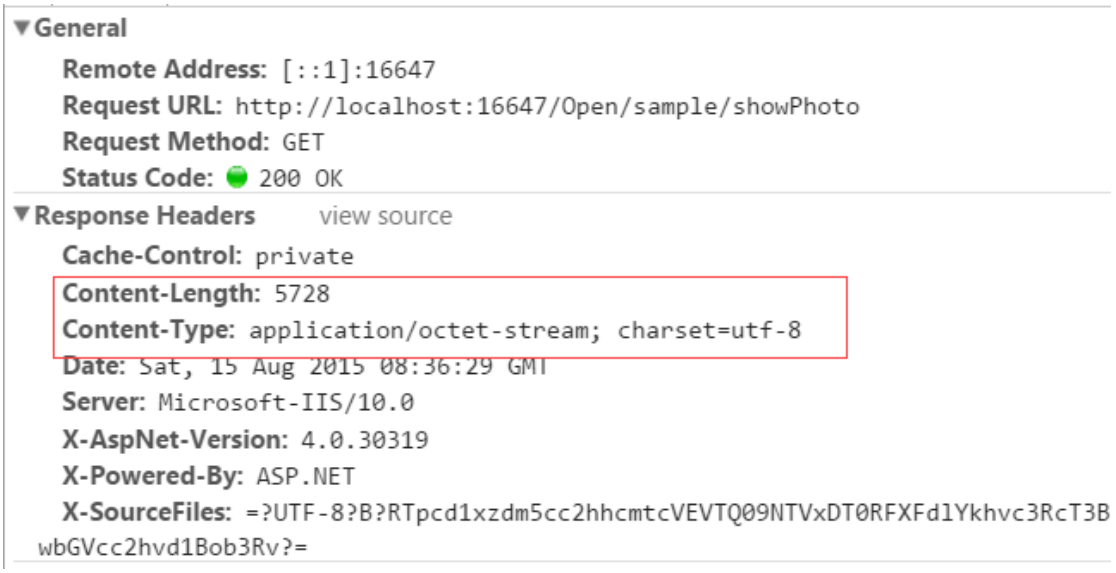
```
using Tescomm.ServiceEngine.Common;
namespace Tescomm.Open.Services
{
    /// <summary>
    /// 请求服务实例
    /// </summary>
    [Serializable]
    0 个引用
    public class Sample:ISample
    {
        /// <summary>
        /// 返回字符串实例
        /// </summary>
        /// <param name="name">参数名称</param>
        /// <returns>返回内容</returns>
        1 个引用
        public string Hello(string name)
        {
            return "Hello " + name + ", Are you okay?";
        }
        /// <summary>
        /// 返回实体实例
        /// </summary>
        /// <param name="name">名称</param>
        /// <param name="age">年龄，数字类型</param>
        /// <returns>Person实例</returns>
        1 个引用
        public Person HowOld(string name, int age)
        {
            return new Person() { Name = name, Age = age };
        }
        /// <summary>
        /// 返回图片流
        /// </summary>
        /// <returns></returns>
        [Action("ShowPhoto", "GET,Post", "Image")]
        1 个引用
        public Stream ShowPhoto()
        {
            return File.OpenRead(Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "static", "imgs", "user-128x128.jpg"));
        }
    }
}
```

Action标签所在程序集

添加标签，并设置ContentType=Image

编译查看输出结果





浏览器跳转

说明：将浏览器重新定向新的目标地址。

1、定义服务接口

```
using Tescomm.Open.Models;
using System.IO;
namespace Tescomm.Open.Interfaces
{
    1 个引用
    public interface ISample
    {
        1 个引用
        string Hello(string name);

        1 个引用
        Person HowOld(string name, int age);

        1 个引用
        Stream ShowPhoto();

        1 个引用
        string GotoBaidu();
    }
}
```

2、实现服务方法，添加 Action 标签，将 ContentType 设置为 location，函数返回等于新的跳转地址。

```
        /// <param name="name">参数名称</param>
        /// <returns>返回内容</returns>
        1 个引用
        public string Hello(string name)
        {
            return "Hello " + name + ", Are you okay?";
        }
        /// <summary>
        /// 返回实体实例
        /// </summary>
        /// <param name="name">名称</param>
        /// <param name="age">年龄，数字类型</param>
        /// <returns>Person实例</returns>
        1 个引用
        public Person HowOld(string name, int age)
        {
            return new Person() { Name = name, Age = age };
        }
        /// <summary>
        /// 返回图片流
        /// </summary>
        /// <returns></returns>
        [Action("ShowPhoto", "GET,Post", "Image")]
        1 个引用
        public Stream ShowPhoto()
        {
            return File.OpenRead(Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "st
        }

        /// <summary>
        /// 浏览器跳转
        /// </summary>
        /// <returns></returns>
        [Action("GotoBaidu","GET","location")]
        1 个引用
        public string GotoBaidu()
        {
            return "http://baidu.com";
        }
    }
}
```

3、编译,浏览结果

URL	协议	方法	结果	类型	已接收	已花费	发起程序
http://baidu.com/	HTTP	GET	302	text/html	290 B	46 ms	导航

框架组件使用说明

服务引擎

负责 URL 解析，方法激活、响应输出。实现服务方法属性标签。
方法参数：支持值参数和复杂参数
返回值：值类型、类实例、流、字符串、url 字符串

方法属性标签:请求方式、ContentType 限定

程序集名称：Tescomm.ServiceEngine ， Tescomm.ServiceEngine.Common

```
/// <summary>
/// 返回图片流
/// </summary>
/// <returns></returns>
[Action("ShowPhoto", "GET,Post", "Image")]
1 个引用
public Stream ShowPhoto()
{
    return File.OpenRead(Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "static", "imgs", "user-128x128.jpg"));
}
```


ContentType 类型：

```
0 个引用
public static string HTML { get { return "text/html"; } }
0 个引用
public static string XML { get { return "application/xml"; } }

/// <summary>
/// json字符串
/// </summary>
0 个引用
public static string JSON { get { return "application/json"; } }

/// <summary>
/// 图像
/// </summary>
0 个引用
public static string Image { get { return "application/octet-stream"; } }
/// <summary>
/// 文件
/// </summary>
0 个引用
public static string File { get { return "application/octet-stream"; } }

/// <summary>
/// 客户端url跳转
/// </summary>
0 个引用
public static string Location { get { return "Location"; } }
```

数据访问组件

提供对数据库访问能力的支持，隔离业务与具体数据引擎依赖，通过配置实现不同类型数据库间的无缝切换。

程序集名称：Tescomm.Framework.DB，

组件名称：DBManager

公开方法：

模型方法

List：返回模型实例列表

Delete：删除记录

Get：查询符合条件的第一条实例记录

GetToInstance:查询符合条件的记录，并将查询结果回填到传入实体实例中

Update：更新记录

Insert：插入记录，成功返回 true，否则 false

Insert4Identity：插入记录，并返回插入的主键值

定义：

```
/// <summary>
/// 根据模板类型返回具体实体列表
/// </summary>
/// <typeparam name="T">实体模型类型</typeparam>
/// <param name="modelId">模型关系定义ID</param>
/// <param name="paramData">查询条件值，可以是字典实例，
/// 模型实例或匿名类实例</param>
/// <returns>实体模型列表</returns>
0 个引用
public static IList<T> List<T>(string modelId ,dynamic paramData) where T : new()


/// <summary>
/// 使用实体模型关系定义删除数据
/// </summary>
/// <typeparam name="T">实体模型类型</typeparam>
/// <param name="modelID">模型关系定义ID</param>
/// <param name="paramData">删除条件值，可以是字典实例，模型实例或匿名类实例</p
/// <returns></returns>
2 个引用
public static bool Delete<T>(string modelID, dynamic paramData) where T : new()

/// <summary>
/// 获取定义模型或脚本对应的记录数
/// </summary>
/// <typeparam name="T"></typeparam>
/// <param name="modelId"></param>
/// <param name="param"></param>
/// <returns></returns>
0 个引用
public static int Count<T>(string modelId, object param) where  T:new()


/// <summary>
/// 分页返回查询结果
/// </summary>
/// <typeparam name="T">实体类型</typeparam>
/// <param name="modelId">实体关系配置ID或dbCommand配置ID</param>
/// <param name="param">查询参数</param>
/// <param name="index">页索引号，从1开始</param>
/// <param name="size">每页大小</param>
/// <returns>查询的分页数据列表</returns>
0 个引用
public static IList<T> PageList<T>(string modelId ,object param ,int index,int size) where  T:new()
```

```
/// <summary>
/// 插入记录并返回主键值
/// </summary>
/// <typeparam name="T">实体模型类型</typeparam>
/// <param name="modelId">模型关系ID</param>
/// <param name="param">插入的数据参数实例</param>
/// <returns>插入记录的主键值</returns>
13 个引用
public static object Insert4Identity<T>(string modelId, dynamic param) where T : new()
```

使用该方法时必须在定义上添加“identity”属性，mysql 设置为：“@@IDENTITY”，orcale 则为序列名称。

```
<field id="token_id" name="token_id" type="String" identity="@@IDENTITY" isKey="true" />
```

```
/// <summary>
/// 使用模型定义插入记录
/// </summary>
/// <typeparam name="T">实体模型类型</typeparam>
/// <param name="modelId">config中定义的模型映射关系ID</param>
/// <param name="paramData">需要插入的模型实例值，可以是字典，实体模型实例或匿名类实例</param>
/// <returns>记录插入成功返回True，否则False</returns>
1 个引用
public static bool Insert<T>(string modelId, dynamic paramData) where T : new()
```

```
/// <summary>
/// 更新数据
/// </summary>
/// <typeparam name="T">实体模型类型</typeparam>
/// <param name="modelId">模型关系定义ID</param>
/// <param name="paramData">更新的参数实例</param>
/// <returns>数据更新成功返回true，否则false</returns>
13 个引用
public static bool Update<T>(string modelId, dynamic paramData) where T : new()
```

```
/// <summary>
/// 查询第一条数据
/// </summary>
/// <typeparam name="T">实体类型</typeparam>
/// <param name="modelId">模型定义关系ID</param>
/// <param name="paramData">查询参数值，可以是字典类型，实体实例或匿名类实例</param>
/// <returns>返回第一条符合条件的记录</returns>
1 个引用
public static T Get<T>(string modelId, dynamic paramData) where T : new()
```

```
/// <summary>
/// 将查询的到的字段信息赋值到传入实例的属性中
/// </summary>
/// <typeparam name="T">拥有空构造函数的类名</typeparam>
/// <param name="modelId">模型定义的id</param>
/// <param name="paramData">查询参数</param>
/// <param name="instance">需要赋值的对象实例</param>
0 个引用
public static void GetToInstance<T>(string modelId, dynamic paramData,ref T instance  ) where T : new()
```

示例：返回应用状态标识为 1 的所有应用记录。

1、 定义模型关系

2、 定义模型 ID 映射常量，方便查看模型使用的数量

```
/// <summary>
/// 项目Model模板
/// </summary>
5 个引用
public static string AppModel { get { return "app"; } }
```

3、 在 DBManager 里的模型操作方法中，使用模型定义

```
var modelList = DBManager.List<AppInfo>(DBCommands.AppModel, new { Status = 1 });
```

插入记录

```
AppInfo model = new AppInfo();
model.Id = Guid.NewGuid().GetGuid();
model.Create_Date = DateTime.Now;
result = DBManager.Insert<AppInfo>(DBCommands.AppModel, model);
```

更新记录

```
AppInfo model = new AppInfo();
model.Id = "dd";
model.Modify_Date = DateTime.Now;
result = DBManager.Update<AppInfo>(DBCommands.AppModel, model);
```

删除记录

```
bool result = true;
result = DBManager.Delete<AppInfo>(DBCommands.AppModel, new { Id = id, Status = 0 });
```

插入记录并返回自增主键

1、 定义模型关系，指定自增主键，identity 属性在 mysql 使用“@@IDENTITY”，rocale 使用序列名称

```
<model id="log_model" db="mysqlConn" table="sys_log">
  <fields>
    <field id="ID" name="id" type="String" isKey="true" identity="@@IDENTITY" />
  </fields>
</model>
```

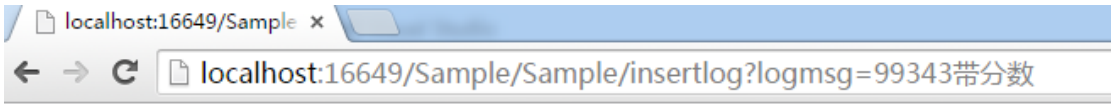
2、 定义实体模型

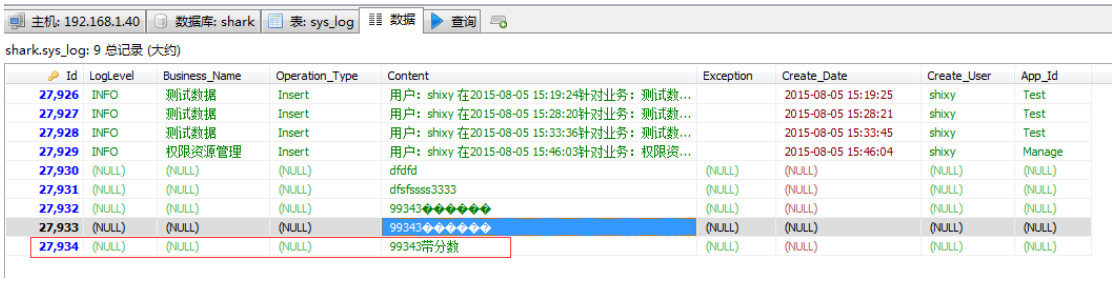
```
3 个引用
public class log
{
    1 个引用
    public log() { }
    0 个引用
    public int ID { get; set; }
    1 个引用
    public string Content { get; set; }
}
```

3、 调用 Insert4Identity 方法

```
public string InsertLog(string logmsg)
{
    return DBManager.Insert4Identity<log>("log_model", new log() { Content = logmsg }).ToString();
}
```

4、 查看输出结果





Id	LogLevel	Business_Name	Operation_Type	Content	Exception	Create_Date	Create_User	App_Id
27,926	INFO	测试数据	Insert	用户: shixy 在2015-08-05 15:19:24针对业务: 测试数...		2015-08-05 15:19:25	shixy	Test
27,927	INFO	测试数据	Insert	用户: shixy 在2015-08-05 15:28:20针对业务: 测试数...		2015-08-05 15:28:21	shixy	Test
27,928	INFO	测试数据	Insert	用户: shixy 在2015-08-05 15:33:36针对业务: 测试数...		2015-08-05 15:33:45	shixy	Test
27,929	INFO	权限资源管理	Insert	用户: shixy 在2015-08-05 15:46:03针对业务: 权限资...		2015-08-05 15:46:04	shixy	Manage
27,930	(NULL)	(NULL)	(NULL)	dfdfd	(NULL)	(NULL)	(NULL)	(NULL)
27,931	(NULL)	(NULL)	(NULL)	dfsfssss3333	(NULL)	(NULL)	(NULL)	(NULL)
27,932	(NULL)	(NULL)	(NULL)	99343□□□□□□	(NULL)	(NULL)	(NULL)	(NULL)
27,933	(NULL)	(NULL)	(NULL)	99343□□□□□□	(NULL)	(NULL)	(NULL)	(NULL)
27,934	(NULL)	(NULL)	(NULL)	99343带分数	(NULL)	(NULL)	(NULL)	(NULL)

SQL 脚本方法

ExecuteList：执行 dbCommand 定义的 sql 脚本，返回所有查询到的记录列表

ExecuteTransaction：开启事务操作，在方法回调中执行数据库的操作。

ExecuteJsonObject：执行 dbCommand 定义的 sql 脚本，返回第一条件记录。

Execute：执行 dbCommand 定义的 sql 脚本，返回影响数。

ExecuteDict：执行 dbCommand 中的 sql 脚本，返回多个列表记录，orcale 不支持同时执行多个 sql 语句，故该方法在 orcale 配置环境下不能使用。

定义

```
/// <summary>
/// 执行dbCommand脚本，并返回JsonObject列表
/// </summary>
/// <param name="cmdID">命令ID</param>
/// <param name="paramData">参数集合</param>
/// <returns>查询结果</returns>
29 个引用
public static IList<JsonObject> ExecuteList(string cmdID, object paramData = null)
```

```
/// <summary>
/// 启用事务执行数据库操作方法
/// </summary>
/// <param name="callback">包含有数据库操作的回调函数,回调中调用的数据库操作方法将自动附近上当前事务
/// ,参数：事务名称，返回值（bool），true：提交事务，false：回滚</param>
0 个引用
public static void ExecuteTransaction(Func<string, bool> callback)


/// <summary>
/// 执行dbCommand定义的sql脚本，返回第一条件记录
/// </summary>
/// <param name="cmdID">dbCommand定义ID</param>
/// <param name="paramData">参数</param>
/// <returns></returns>
1 个引用 | 0/1 通过
public static JsonObject ExecuteJsonObject(string cmdID, object paramData = null)


/// <summary>
/// 执行dbCommand定义的sql脚本，返回执行后的影响记录数
/// </summary>
/// <param name="cmdID">dbCommand定义ID</param>
/// <param name="paramData">参数值，可以是字典或实体、泛型类实例</param>
/// <returns>影响数</returns>
 2 个引用
 public static int Execute(string cmdID, object paramData = null)

/// <summary>
///执行脚本模板 返回多个记录集
/// </summary>
/// <param name="cmdID">模板ID</param>
/// <param name="paramData">模板所需参数</param>
/// <returns></returns>
0 个引用
public static IDictionary<string, IList<JsonObject>> ExecuteDict(string cmdID, object paramData = null)


/// <summary>
/// 迭代参数列表，执行多次dbCommand定义sql脚本
/// </summary>
/// <param name="cmdID">dbCommand定义ID</param>
/// <param name="paramList">参数集合，继承IEnumerable接口的实例</param>
/// <returns></returns>
0 个引用
public static bool Execute(string cmdID, IEnumerable<IDictionary<string, object>> paramList)
```

示例

获取日志列表，传递参数，操作类型，开始、结束时间

1、定义 Sql 脚本

```
<dbCommand id="GetLogs" db="conn1">
  <text>
    <![CDATA[
  SELECT * FROM CTUNI_OAUTH.Sys_Log WHERE ({Type:String} is null or Operation_Type={Type:String}) AND Create_Date>={Begin:DateTime} AND Create_Date<={End:DateTime}
]]>
  </text>
</dbCommand>
```

2、调用脚本

```
var list = DBManager.ExecuteList(DBCommands.GetLogs, new { Type = operationType, Begin = begin, End = end });
```

执行事务操作，回调函数中执行的数据库组件方法将会自动带上事务属性，无需显式传入事务参数。

```
DBManager.ExectueTrancation((n) =>
{
    DBManager.Insert4Identity<log>("log_m", new log() { Content = "2" + logmsg }).ToString();
    return true;
});
```

根据用户 ID 获取用户信息

```
<dbCommand id="User_Get" db="conn1">
  <text>
    <!--AND a.App_Id={App_Id:String} AND a.App_Id={App_Id:String} -->
    <![CDATA[
  SELECT a.*,b.Bind_Account AS QQ,c.Bind_Account AS WeChat FROM CTUNI_OAUTH.Sys_User a LEFT JOIN CTUNI_OAUTH.Sys_LoginAccount b ON a.Id=b.User_Id AND b.Bind_Type=0
  LEFT JOIN CTUNI_OAUTH.Sys_LoginAccount c ON a.Id=c.User_Id AND c.Bind_Type=1 WHERE STATUS=1 AND Id={Id:STRING}
]]>
  </text>
</dbCommand>
```

```
DBManager.ExecuteJsonObject(DBCommands.User_Get, new { Id = id })
```

保存用户与项目关系

```
<dbCommand id="User_SaveApp" db="conn1">
  <text>
    <![CDATA[
  insert into CTUNI_OAUTH.Sys_UserApp (User_Id,App_Id) values ({Relation_Id:String},{App_Id:String})
]]>
  </text>
</dbCommand>
```

```
IDictionary<string, object> userOrg = new Dictionary<string, object>();

userOrg.Add("Relation_Id", model.Id);

userOrg.Add("App_Id", CommonData.App_Id);
DBManager.Execute("User_SaveApp", userOrg);
```

使用动态条件

定义脚本

```
<dbCommand id="cmd1" db="conn1" >
<!--动态条件示例-->
  <text>
    <![CDATA[      SELECT sysdate from dual      ]]>
  </text>
  <!--动态条件语句定义
    dynamic:动态条件节点
    prepend: 预置值，当有条件成立是附加到Text包含的文本之后
  -->
  <dynamic prepend=" where ">
    <statements>
      <!--条件定义，id: 条件唯一标识，prepend: 预置置，条件成立是附近到文本之后
      type: 条件比较类型，
      parameter: 执行时传入的参数名称
      text:条件成立时附加到父节点Text之后的sql脚本
      value:与参数比较的值
      before:当条件成立时添加到本段Text前面，在prepend后
      after: 当条件成立时添加到本地Text后面
      itemPrepend: 当为Iterate类型时，迭代输出时自动添加到每次循环Text前。
      -->
      <!--参数值不为Null-->
      <statement id="c1" type="NotNull" prepend=" and " parameter="p1">
        <text>
          <![CDATA[     {p1:String} ={p1:String}      ]]>
        </text>
      </statement>
      <!--参数值不为空-->
      <statement id="c2" type="NotEmpty" prepend=" and " parameter="p1">
        <text>              <![CDATA[ {p1:String} ={p1:String} ]]>            </text>
      </statement>
      <!--参数值与比较值相等-->
      <statement id="c3" type="Equals" prepend=" and  " parameter="p1" value="009">
        <text>              <![CDATA[     {p1:String} ={p1:String}      ]]>            </text>
      </statement>
      <!--参数值小于等于比较值-->
        <statement id="c4" type="LesserAndEquals" prepend=" and  " before="(  after=")" parameter="p1" value="19">
          <text>                <![CDATA[     19 =19      ]]>              </text>
      </statement>
      <!--参数值大于等于比较值-->
      <statement id="c5" type="GreaterAndEquals" prepend=" and  " before="(  after=")" parameter="p1" value="1">
        <text>
          <![CDATA[     1 =1      ]]>
        </text>
      </statement>
      <!--迭代参数值输出-->
      <statement id="c6" type="Iterate" prepend=" and  " before="(  after=")" itemPrepend=" or " parameter="p2">
        <text>              <![CDATA[     {p2}={p2}      ]]>            </text>
      </statement>
      <!--迭代参数值输出，如果text里包含的参数名为子项属性名则会被替换成对应值-->
      <statement id="c7" type="Iterate" prepend=" and  " before="(  after=")" itemPrepend=" or " parameter="p3">
        <text>              <![CDATA[     '{p31}'='{p31}' and {p32}={p32}      ]]>            </text>
      </statement>
    </statements>
  </dynamic>
  </dbCommand>
```

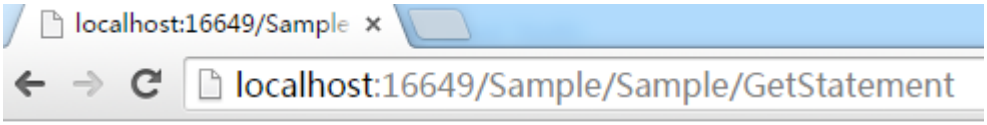
传入执行参数

```
0 个引用
public string GetStatement()
{
    List<int> ll = new List<int>();
    ll.Add(8);
    ll.Add(81);
    ll.Add(82);
    return DBManager.ExecuteJsonObject("cmd1",
        //匿名类
        new
    {
        p1 = 009,//值参数
        p2 = ll ,//列表参数，
        p3 = new[] { new { p31 = "322dd", p32 = 2233 }, new { p31 = "111dd", p32 = 33 } }//匿名类数组
    })[0].ToString();
}
```

解析后的 sql

SELECT sysdate from dual where :p1 =:p1 and :p1 =:p1 and (19 =19) and (1 =1) and (8=8 or 81=81 or 82=82) and ('322dd'='322dd' and 2233=2233 or '111dd'='111dd' and 33=33)

输出结果



缓存组件

程序集：Tescomm.Framework.Cache.dll
名称空间：Tescomm.Framework.Cache
类名：CacheManager
方法：

```
/// <summary>
/// 获取对应缓存实例
/// </summary>
/// <param name="name">CacheSettings节下节点的 name</param>
/// <returns></returns>
4 个引用
private static ICache Get(string name)...
/// <summary>
/// 重指定缓存服务中提取key对应的值
/// </summary>
/// <typeparam name="T">缓存数据的类型</typeparam>
/// <param name="name">缓存服务名称，对应配置文件中的 cache节点中的name</param>
/// <param name="key">提取缓存的键名</param>
/// <returns>缓存数据</returns>
2 个引用 | 0/1 通过
public static T Get<T>(string name, string key)...

/// <summary>
/// 将对应数据提交到缓存服务
/// </summary>
/// <typeparam name="T">缓存数据类型</typeparam>
/// <param name="name">缓存服务配置名称</param>
/// <param name="key">键名</param>
/// <param name="v">缓存数据</param>
/// <param name="expire">缓存过期时间，单位：秒</param>
1 个引用 | 0/1 通过
public static void Put<T>(string name, string key, T v,int expire)...
/// <summary>
/// 删除对应键值的缓存数据
/// </summary>
/// <param name="name">缓存服务配置名称</param>
/// <param name="key">键名称</param>
0 个引用
public static void Remove(string name, string key)...
/// <summary>
/// 清除缓存服务中的所有数据
/// </summary>
/// <param name="name"></param>
0 个引用
public static void Clear(string name)...
```

示例：

```
0 个引用
public string GetCache()
{

    //获取缓存数据 sample1数据
    return Tescomm.Framework.Cache.CacheManager.Get<string>("redis", "sample1");
}
0 个引用
public string PutCache(string v)
{
    //缓存 v 参数里的值10秒
    Tescomm.Framework.Cache.CacheManager.Put<string>("redis", "sample1", v, 10);
    return "success";
}
```

日志组件

程序集：Tescomm.Framework.Log.dll
名称空间：Tescomm.Framework.Log
类名：LogManager
方法：

```
public static void Debug(string logger, object message, Exception exception)
{
    ILog lger = CreateLog(logger);
    lger.Debug(message, exception);
}
0 个引用
public static void DebugFormat(string logger, string format, object arg0)
{
    ILog lger = CreateLog(logger);
    lger.DebugFormat(format, arg0);
}
0 个引用
public static void DebugFormat(string logger, string format, params object[] args)
{
    ILog lger = CreateLog(logger);
    lger.DebugFormat(format, args);
}
0 个引用
public static void DebugFormat(string logger, IFormatProvider provider, string format, params object[] args)
{
    ILog lger = CreateLog(logger);
    lger.DebugFormat(provider, format, args);
}
0 个引用
public static void DebugFormat(string logger, string format, object arg0, object arg1)
{
    ILog lger = CreateLog(logger);
    lger.DebugFormat(format, arg0, arg1);
}
0 个引用
public static void DebugFormat(string logger, string format, object arg0, object arg1, object arg2)
{
    ILog lger = CreateLog(logger);
    lger.DebugFormat(format, arg0, arg1, arg2);
}
```

```
public static void Error(string logger, object message)...
1 个引用
public static void Error(string logger, object message, Exception exception)...
0 个引用
public static void ErrorFormat(string logger, string format, object arg0)...
0 个引用
public static void ErrorFormat(string logger, string format, params object[] args)...
0 个引用
public static void ErrorFormat(string logger, IFormatProvider provider, string format, params object[] args)...
2 个引用
public static void ErrorFormat(string logger, string format, object arg0, object arg1)...

1 个引用
public static void ErrorFormat(string logger, string format, object arg0, object arg1, object arg2)...

  public static void Info(string logger, object message)...
  0 个引用
  public static void Info(string logger, object message, Exception exception)...
  0 个引用
  public static void InfoFormat(string logger, string format, object arg0)...
  0 个引用
  public static void InfoFormat(string logger, string format, params object[] args)...
  0 个引用
  public static void InfoFormat(string logger, IFormatProvider provider, string format, params object[] args)...
  0 个引用
  public static void InfoFormat(string logger, string format, object arg0, object arg1)...
  1 个引用
  public static void InfoFormat(string logger, string format, object arg0, object arg1, object arg2)...
public static void Warn(string logger, object message)...
0 个引用
public static void Warn(string logger, object message, Exception exception)...
0 个引用
public static void WarnFormat(string logger, string format, object arg0)...
0 个引用
public static void WarnFormat(string logger, string format, params object[] args)...
0 个引用
public static void WarnFormat(string logger, IFormatProvider provider, string format, params object[] args)...
0 个引用
public static void WarnFormat(string logger, string format, object arg0, object arg1)...
0 个引用
public static void WarnFormat(string logger, string format, object arg0, object arg1, object arg2)...

/// <summary>
/// 日志输出到数据库
/// </summary>
/// <param name="logMsg">日志对象</param>
1 个引用
public static void Log2DB(LogMsg logMsg)...
```

示例：

将错误信息既然到错误日志中

```
string LOGGER_NAME = "ServiceEngine";
LogManager.ErrorFormat(LOGGER_NAME,"错误信息：{0}\r\n 异常信息:{1}", msg,ex);
```

鉴权组件

程序集：Tescomm.Framework.Auth.dll

名称空间：Tescomm.Framework.Auth

类名：AuthManager

方法：

```
/// <summary>
/// 取得当前app的鉴权组件实例
/// </summary>
6 个引用
public static IAuth Current...
/// <summary>
/// 取得当前用户的菜单及动作权限
/// </summary>
0 个引用
public static IList<Permission> CurrentUserRights...
/// <summary>
/// 取得当前用户的数据权限信息
/// </summary>
0 个引用
public static IList<DataItem> CurrentUserDataRights...

/// <summary>
/// 取得当前页登录的用户基础信息
/// </summary>
1 个引用
public static UserInfo CurrentUser...
```

示例：

```
/// <summary>
/// 获取当前用户的姓名
/// </summary>
/// <returns></returns>
0 个引用
public string GetUserName()
{
    return Tescomm.Framework.Auth.AuthManager.CurrentUser.Real_Name;
}
```

通用组件库

程序集：Tescomm.Framework.Common.dll
名称空间：Tescomm.Framework.Common

类名：AppHelper

```
/// <summary>
/// 取得当前的应用信息
/// </summary>
2 个引用
public static ApplicationInfo Current...

/// <summary>
/// 当前应用上下文
/// </summary>
11 个引用
public static HttpContext HostContext...
/// <summary>
/// 插入session值
/// </summary>
/// <param name="name"></param>
/// <param name="v"></param>
0 个引用
public static void SetSession(string name, object v)...
/// <summary>
/// 获取session值
/// </summary>
/// <param name="name"></param>
/// <returns></returns>
0 个引用
public static object GetSession(string name)...
/// <summary>
/// 获取当前登录的Token
/// </summary>
3 个引用
public static string CurrentToken...

/// <summary>
///请求的 cookie集合
/// </summary>
0 个引用
public static HttpCookieCollection Cookie...
/// <summary>
/// 当前请求的端口号
/// </summary>
0 个引用
public static int Port...
/// <summary>
/// 当前请求Host
/// </summary>
0 个引用
public static string Host...
/// <summary>
/// 当前请求的协议名称,http或https
/// </summary>
0 个引用
public static string Scheme ...
```

类名：ConfigurationManager

方法：

```
//配置文件存放目录，相对于当前应用目录
public const string CONFIG_DIR = "Config";
/// <summary>
/// 将json配置文件序列化成对应实例
/// </summary>
/// <typeparam name="T"></typeparam>
/// <param name="config"></param>
/// <returns></returns>
2 个引用
public static T CreateJSONCongfig<T>(string config)...

/// <summary>
/// 返回配置文件中对应配置节信息
/// </summary>
/// <param name="configFile">配置文件全路径</param>
/// <param name="name">配置节名称</param>
/// <returns></returns>
6 个引用
public static ConfigurationSection GetConfiguationSection(string configFile,string name)...
/// <summary>
/// 返回配置文件的全路径，
/// </summary>
/// <param name="filename"></param>
/// <returns></returns>
6 个引用
public static string GetCofigurationPath(string filename)...
```

示例：

实例化 service.json 配置信息

```
string rootPath = context.Request.PhysicalApplicationPath;
ServiceConfiguration EngineConfig =
ConfigurationManager.CreateJSONCongfig<ServiceConfiguration>(Path.Combine(root
                        , ConstConfig.CONFIG_DIR, ConstConfig.CONFIG_FILE));
```

实例化缓存配置文件

```
CacheConfiguration config = ConfigurationManager.GetConfiguationSection(
ConfigurationManager.GetCofigurationPath(file), "cacheSetting") as CacheConfiguration;
```

类名：Utility

方法：

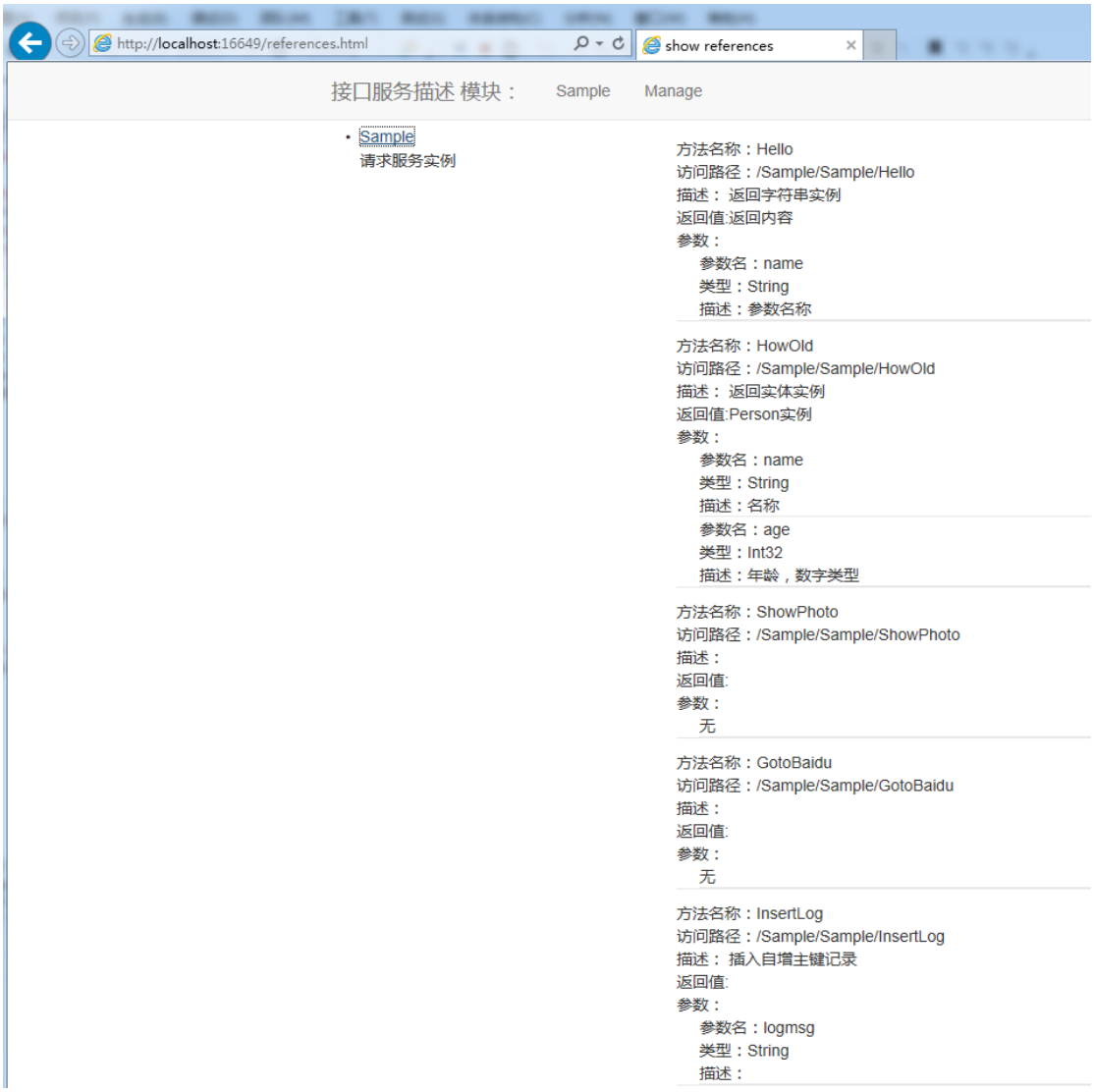
```
/// <summary>
/// 读取流中的文本数据
/// </summary>
/// <param name="s">流实例</param>
/// <returns></returns>
1 个引用
public static string ReadStream2String(Stream s)...
/// <summary>
/// 读取流中的字节数据
/// </summary>
/// <param name="s"></param>
/// <returns></returns>
1 个引用
public static byte[] ReadStream(Stream s)...

/// <summary>
/// 建json字符转换为对应的类实例
/// </summary>
/// <typeparam name="T">目标类型</typeparam>
/// <param name="s">json字符串</param>
/// <returns></returns>
2 个引用
public static T DeserializeJSON<T>(string s)...
/// <summary>
/// 将对象序列化为json字符串
/// </summary>
/// <param name="obj"></param>
/// <returns></returns>
2 个引用
public static string SerializeJSON(object obj)...

/// <summary>
/// 将子目录名称转换成列表
/// </summary>
/// <param name="rootPath"></param>
/// <param name="deep"></param>
0 个引用
public static List<string> GetChildrenPaths(string rootPath, bool deep)...
/// <summary>
/// 将字节数组转换成字符串
/// </summary>
/// <param name="stringBytes">字节数组</param>
/// <returns>字符串</returns>
0 个引用
public string GetString(byte[] stringBytes)...
/// <summary>
```

```
/// <summary>
/// 将字符串编码为字符串数组
/// </summary>
/// <param name="stringValue">字符串</param>
/// <returns>字节数组</returns>
0 个引用
public byte[] GetBytes(string stringValue)...
/// <summary>
/// MD5加密
/// </summary>
/// <param name="input">加密字符串</param>
/// <returns>哈希字符串</returns>
1 个引用
public static string GetMd5Hash(string input)...

/// <summary>
/// 验证字符串和哈希字符串是否匹配
/// </summary>
/// <param name="input">验证字符串</param>
/// <param name="hash">哈希字符串</param>
/// <returns>通过验证返回true，否则为false</returns>
0 个引用
public static bool VerifyMd5Hash(string input, string hash)...

/// <summary>
/// 将字符串转换哈希值
/// </summary>
/// <param name="inputs"></param>
/// <returns></returns>
0 个引用
public static long String2Hash(string inputs)...

/// <summary>
/// 判断url是否为本地url
/// </summary>
/// <param name="url"></param>
/// <returns></returns>
0 个引用
public static bool IsLocalUrl(string url)...
/// <summary>
/// 生成32的guid字符串
/// </summary>
/// <returns></returns>
1 个引用
public static string GetGuid()...
```

```
/// <summary>
///  将url中的参数值转换成字典
/// </summary>
/// <param name="urlParams"></param>
/// <returns></returns>
0 个引用
public static Dictionary<string, string> UrlParameters2Dict(string urlParams)...
/// <summary>
/// 判断字符串是否为数字
/// </summary>
/// <param name="target">数字字符串</param>
/// <returns>数字返回true，否则false</returns>
3 个引用
public static bool IsNumber(string target)...
/// <summary>
/// 判断目标Ip是否在指定范围内
/// </summary>
/// <param name="target">目标IP</param>
/// <param name="start">起始IP</param>
/// <param name="end">结束IP</param>
/// <returns>在范围内返回true，否则false</returns>
2 个引用
public static bool IpIsInRange(string target, string start, string end)...
/// <summary>
///将字符串转换为目标类型
/// </summary>
/// <param name="type">目标类型</param>
/// <param name="value">值字符串</param>
/// <returns>目标类型的值</returns>
5 个引用
public static object ConvertValue(Type type, string value)...

/// <summary>
/// 获取字典项或类实例里的属性值
/// </summary>
/// <param name="obj">类型实例或字典实例</param>
/// <param name="propertyName">属性名称或键</param>
/// <returns>属性值或字典项</returns>
3 个引用
public static object GetPropertyOrDictValue(object obj,string propertyName)...
```

查看服务方法

启动 iisexpress 后，浏览/References.html



常见错误解决方法

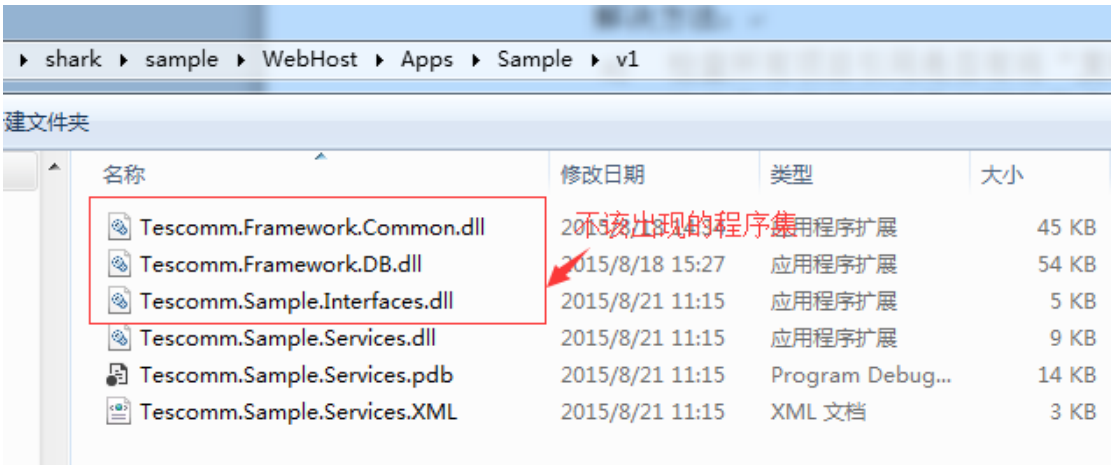
1、 服务方法无进入断点

解决方法：

a) 检查所有项目引用是否有将“复制到本地”项设置为 false；

b) 检查各个类型的目录的程序集是否为本地最新编译的版本，

c) 检查个类型目录下的 dll 是否重复，例如：Models 下的 dll 存在 Interfaces 目录下，Models 目录下是否存在 Common 目录下的 dll 或 framework 下的 dll。



2、 调用数据组件方法成功，但返回数据为空或异常

解决方法：

a) 确认定义的脚本的正确性，在 debug 下，在输出窗口下，将 sql 脚本拷贝到数据库的查看器中执行，检查结果是否正确。

b) 确认定义的脚本模板类型与执行方法是否匹配，dbCommand 定义的脚步只能使用 Execute 开头的方法调用，Model 类型的定义只能使用 Get，Delete，List，Delete,Update 等方法进行调用。

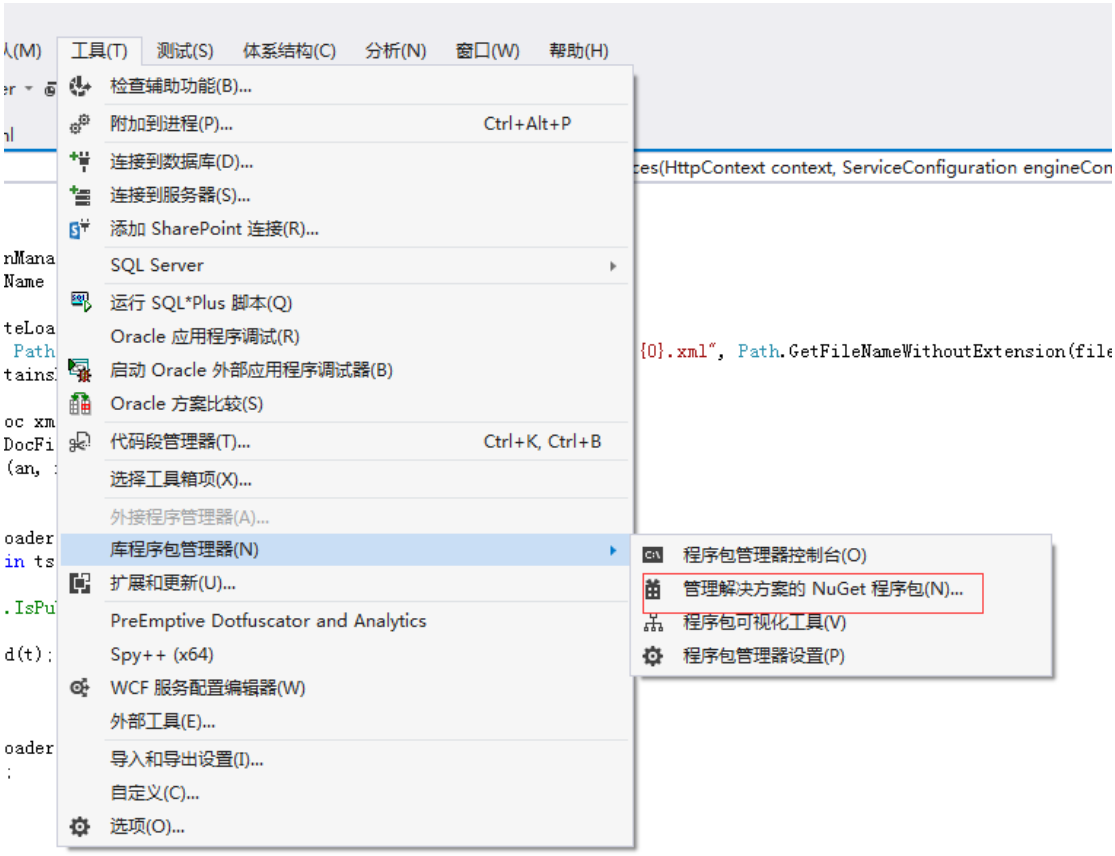
3、 错误日志中提示 orcale 客户端版本过低

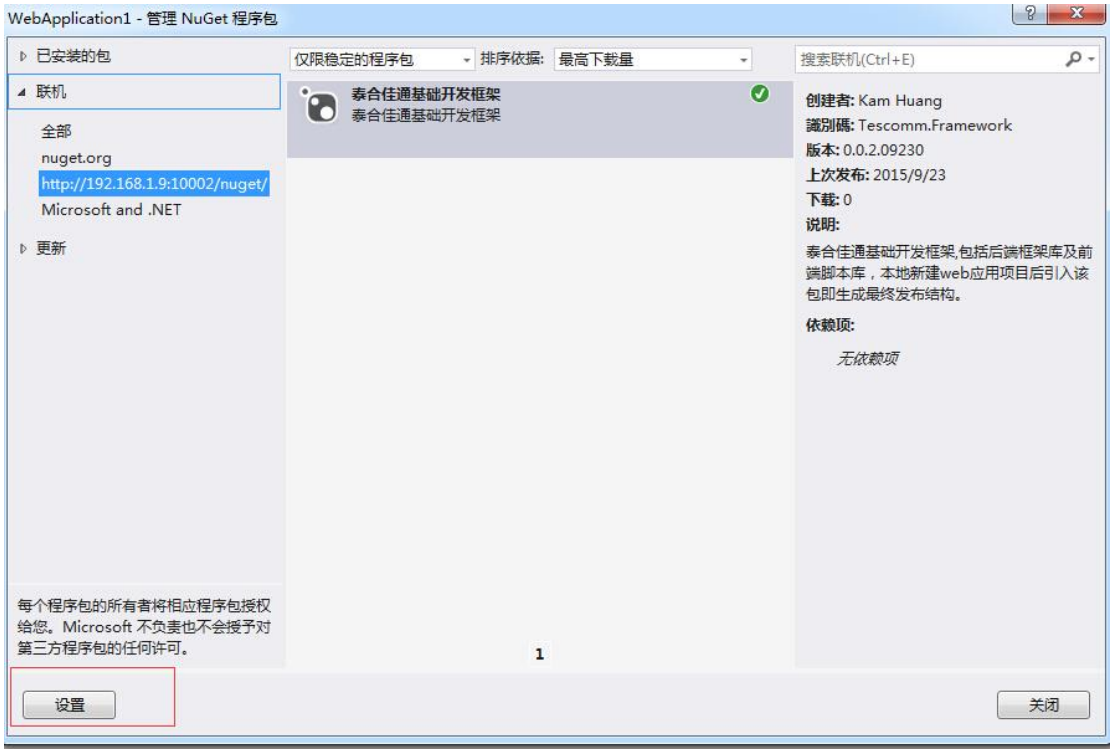
解决方法：

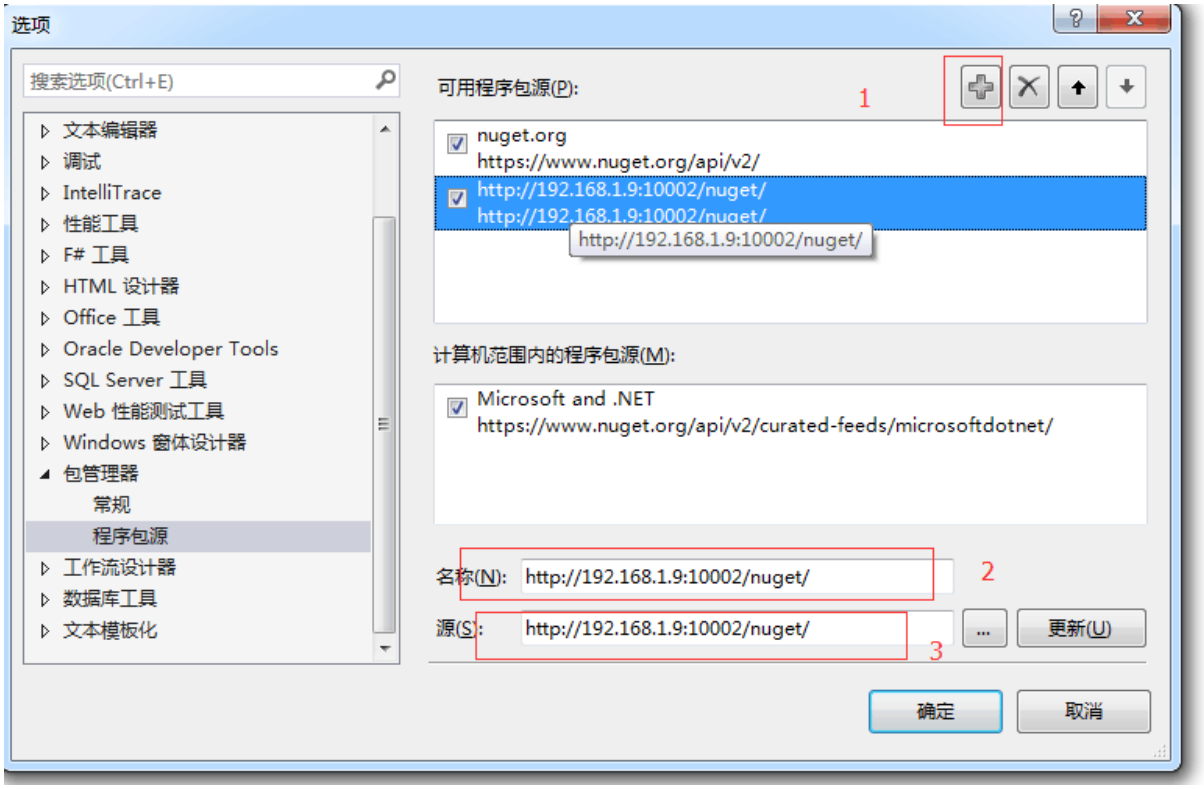
安装 32 位的 orcale 客户端。目前使用的 orcale 链接器为 32 为版本，后续会进行版本适配。

设置 NuGet 源

添加源路径

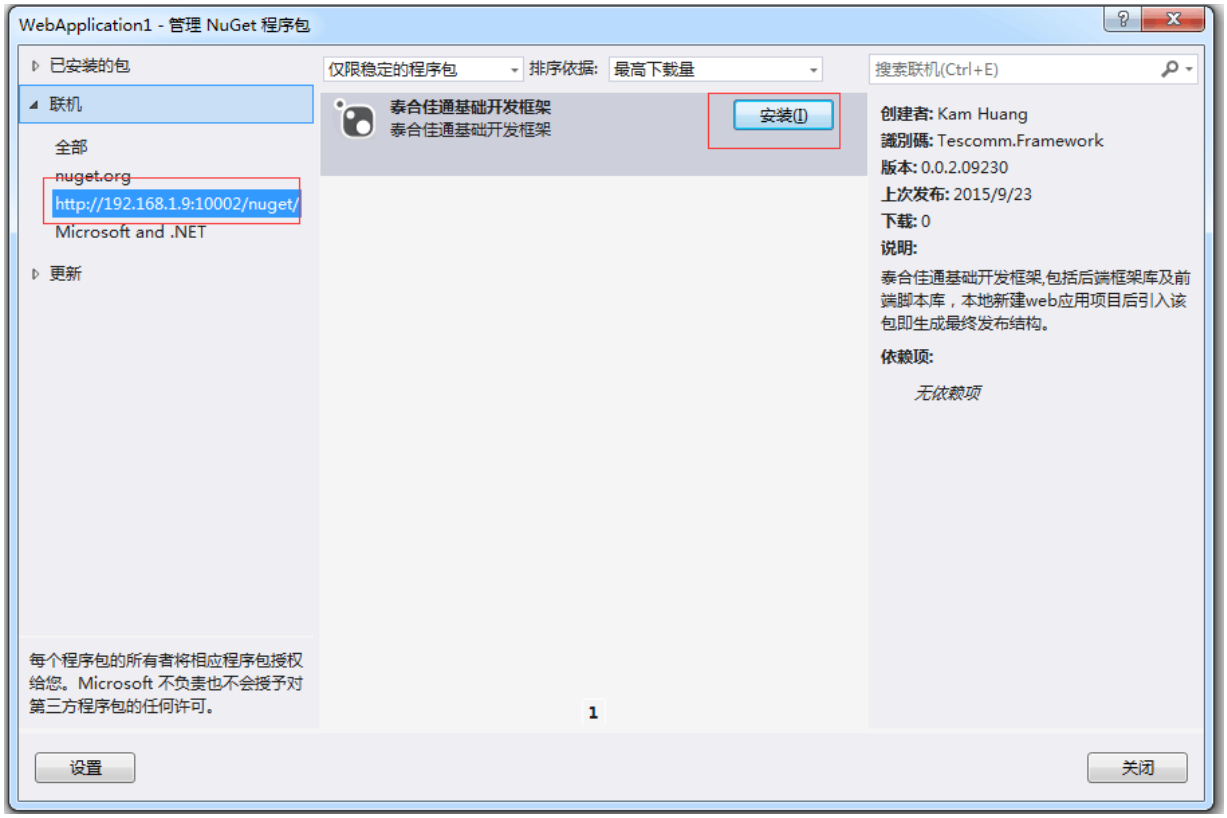

工具(T)
测试(S)
体系结构(C)
分析(N)
窗口(W)
帮助(H)
检查辅助功能(B)...
附加到进程(P)... Ctrl+Alt+P
连接到数据库(D)...
连接到服务器(S)...
添加 SharePoint 连接(R)...
SQL Server
运行 SQL*Plus 脚本(Q)
Oracle 应用程序调试(R)
启动 Oracle 外部应用程序调试器(B)
Oracle 方案比较(S)
代码段管理器(T)... Ctrl+K, Ctrl+B
选择工具箱项(X)...
外接程序管理器(A)...
库程序包管理器(N)
扩展和更新(U)...
PreEmptive Dotfuscator and Analytics
Spy++ (x64)
WCF 服务配置编辑器(W)
外部工具(E)...
导入和导出设置(I)...
自定义(C)...
选项(O)...
程序包管理器控制台(O)
管理解决方案的 NuGet 程序包(N)...
程序包可视化工具(V)
程序包管理器设置(P)


WebApplication1 - 管理 NuGet 程序包
已安装的包
联机
全部
nuget.org
http://192.168.1.9:10002/nuget/
Microsoft and .NET
更新
仅限稳定的程序包
排序依据:
最高下载量
搜索联机(Ctrl+E)
泰合佳通基础开发框架
泰合佳通基础开发框架
创建者: Kam Huang
識別碼: Tescomm.Framework
版本: 0.0.2.09230
上次发布: 2015/9/23
下载: 0
说明:
泰合佳通基础开发框架,包括后端框架库及前端脚本库，本地新建web应用项目后引入该包即生成最终发布结构。
依赖项:
无依赖项
每个程序包的所有者将相应程序包授权给您。Microsoft 不负责也不会授予对第三方程序包的任何许可。
1
设置
关闭

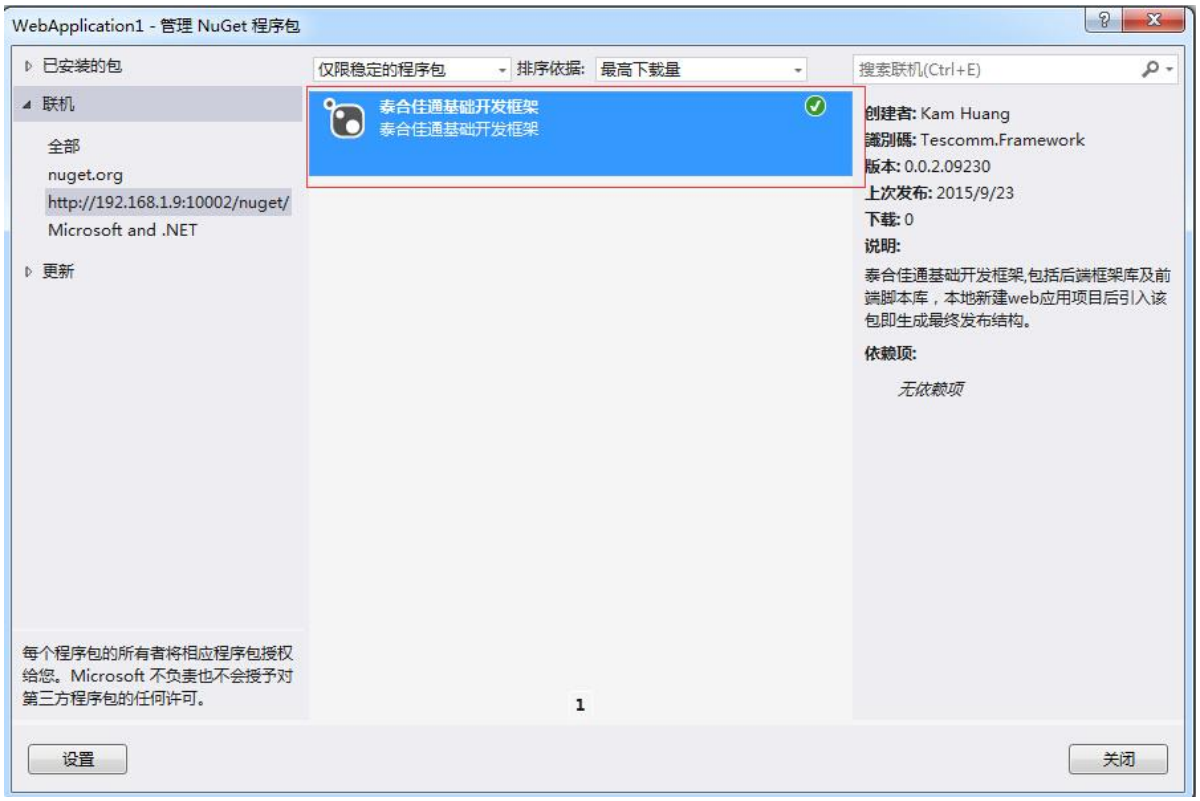


从 NuGet 源中安装包

选择目标源，找到需要安装的包，点击安装



安装成功


WebApplication1 - 管理 NuGet 程序包
已安装的包
联机
全部
nuget.org
http://192.168.1.9:10002/nuget/
Microsoft and .NET
更新
仅限稳定的程序包
排序依据:
最高下载量
搜索联机(Ctrl+E)
泰合佳通基础开发框架
泰合佳通基础开发框架
创建者: Kam Huang
标识码: Tescomm.Framework
版本: 0.0.2.09230
上次发布: 2015/9/23
下载: 0
说明:
泰合佳通基础开发框架,包括后端框架库及前端脚本库，本地新建web应用项目后引入该包即生成最终发布结构。
依赖项:
无依赖项
每个程序包的所有者将相应程序包授权给您。Microsoft 不负责也不会授予对第三方程序包的任何许可。
1
设置
关闭