**<https://www.jianshu.com/p/8242215cd8c0>**

**Web API项目中使用Area对业务进行分类管理**

[](https://www.jianshu.com/u/5c9663046ca3)

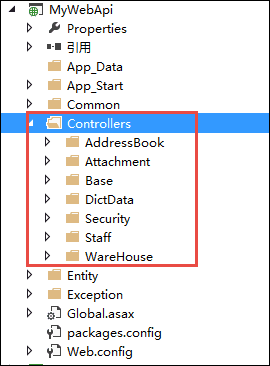
[伍华聪\_开发框架](https://www.jianshu.com/u/5c9663046ca3)

2016.08.31 23:42\* 字数 1943 阅读 1048评论 1喜欢 11

在之前开发的很多Web API项目中，为了方便以及快速开发，往往把整个Web API的控制器放在基目录的Controllers目录中，但随着业务越来越复杂，这样Controllers目录中的文件就增加很快，难以管理，而且如果有不同业务模块有重复的控制器名的话，还需要尽量避免。引入Area的作用就是把控制器按照不同的业务模块进行区分，方便管理，而且控制器名称可以重名。

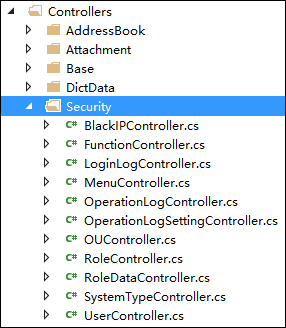
**1、Web API项目引入Area进行分类**

Area在项目中可以称之为区域，每个Area代表应用程序的不同功能模块，Area 使每个功能模块都有各自的文件夹，文件夹中有自己的Controller、View和Model，但对于管理也增加了一定的难度。如果是Web API项目，我们可以把不必要的目录移除即可，简化对目录的管理。  
引入Area可以是我们不同的业务模块可以重名，而且各个业务模块管理起来也更加方便，在原先的Web API项目里面，它们的目录是这样的。



虽然我们把它们的目录归类，但是它们还是存放在一个命名空间下的。

namespace MyWebApi.Controllers



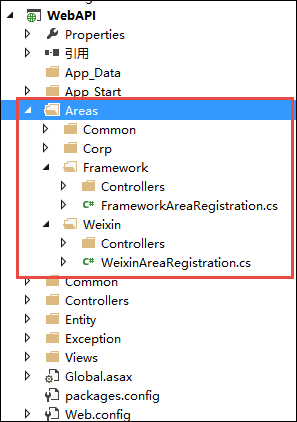
这样使用虽然也没有什么问题，但是还是存在一些弊端，因此引入Area的方式对不同业务模块的控制器进行管理，以达到我们分类管理的目的。  
引入Area前，我们的API路径如下所示

[http://localhost:9001/api/User](https://link.jianshu.com?t=http:/localhost:9001/api/User)

引入Area后，我们把常规的权限管理、字典管理等基础模块放到Framework的Area里面，那么这个时候API路径和具体的Area相关，地址则变成了如下：

[http://localhost:9001/api/Framework/User](https://link.jianshu.com?t=http:/localhost:9001/api/Framework/User)

我们再来看看具体的项目目录，Web API项目中使用Area后，Controller的目录如下所示。

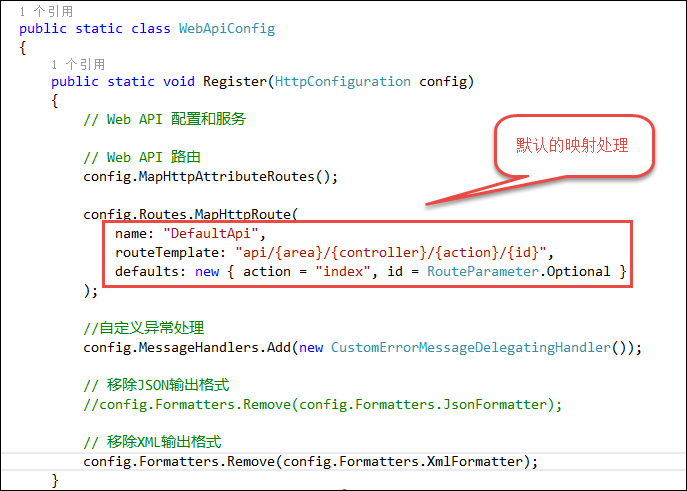


除了在各个不同Area下有不同的控制器，而且也增加了一个\*\*\*\*AreaRegistration.cs的文件，如对应Framework的Area，有一个FrameworkAreaRegistration.cs文件  
这样对应下面的控制器，它的命名空间如下所示。

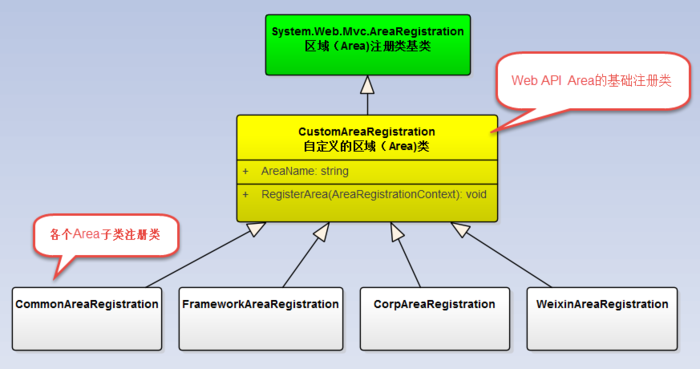
namespace WebAPI.Areas.Framework.Controllers

**2、Web API项目对Area控制器的路径映射**

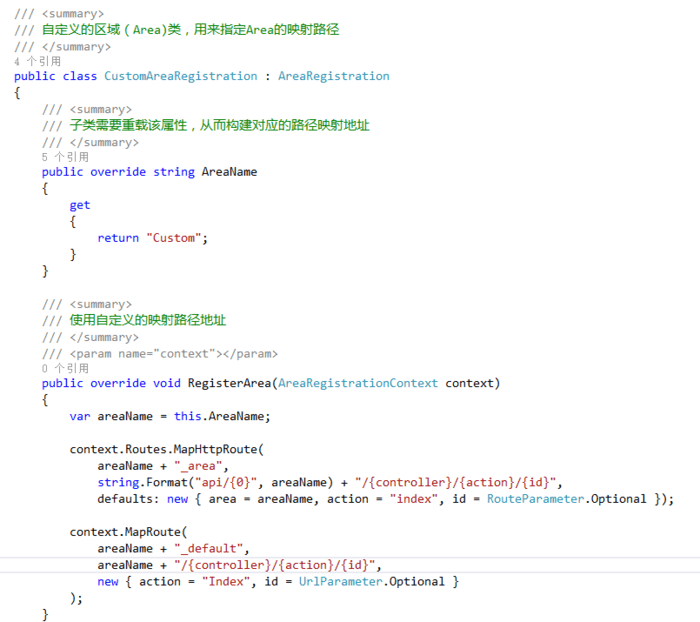
上面小节介绍了使用Area来对Web API控制器的分类管理，并介绍了引入Area后对控制器位置、命名空间、Web API的URL等方面的不同。这样如果我们要解析对应地址的Web API，那么也需要做一定的处理，否则是无法找到对应的控制器，从而出现错误信息。  
首先我们需要修改Web API里面WebApiConfig的配置信息，如下所示。



上面指定了默认的Web API映射，并指定结果只做JSON格式的输出（移除XML输出）。  
为了对不同的Area实现API的地址处理，我们先设计一个基类，然后让不同的Area注册类继承它，方便统一处理。



其中基类Area注册类的CustomAreaRegistration类代码如下所示。



有了上面的基类映射 RegisterArea函数，我们只需要在子类设置对应的AreaName基类实现不同Area子类的正确映射API路径处理了。

/// <summary>

/// 框架基础Area的注册类

/// </summary>

public class FrameworkAreaRegistration : CustomAreaRegistration

{

public override string AreaName

{

get

{

return "Framework";

}

}

}

当然为了实现对Area的Web API控制器的URL正确解析，获取属于Action、Controller、以及对应命名空间的对象，那么还需要在global.asa.cs里面添加一行代码，如下所示。

//对Web API的Area进行支持

GlobalConfiguration.Configuration.Services.Replace(typeof(IHttpControllerSelector), new AreaHttpControllerSelector(GlobalConfiguration.Configuration));

其中AreaHttpControllerSelector是我们自定义的HTTP控制器地址解析器，需要根据我们的地址提取出具体的控制器、Area名称、程序集类型等，方便构建对应的解析器。

private HttpControllerDescriptor GetApiController(HttpRequestMessage request)

{

var controllerName = base.GetControllerName(request);

var areaName = GetAreaName(request);

if (string.IsNullOrEmpty(areaName))

{

return null;

}

var type = GetControllerTypeByArea(areaName, controllerName);

if (type == null)

{

return null;

}

return new HttpControllerDescriptor(\_configuration, controllerName, type);

}

有了这些基础的管理，我们就可以定义好我们所需要Area，然后构建具体业务范畴下的控制器接口即可。

**3、Web API在客户端的接口调用**

所有的Web API地址，都是与具体的Area有关系，例如在Framework业务下的字典模块，它们Web API配置的地址如下所示。

<add key="DictType" value="http://localhost:27206/api/Framework/DictType"/>  
<add key="DictData" value="http://localhost:27206/api/Framework/DictData"/>  
<add key="CorpDictData" value="http://localhost:27206/api/Framework/CorpDictData"/>  
<add key="City" value="http://localhost:27206/api/Framework/City"/>  
<add key="District" value="http://localhost:27206/api/Framework/District"/>  
<add key="Province" value="http://localhost:27206/api/Framework/Province"/>  
<add key="UserParameter" value="http://localhost:27206/api/Framework/UserParameter"/>

我们在客户端，只需要对Web API进行封装即可，这个部分可以使用Database2Sharp代码生成工具进行统一的生成，所有继承关系统一处理好，我们所做的就是进行新增接口的处理即可。  
例如对于字典模块DictData的处理，它对于Web API的封装类如下所示。

/// <summary>

/// DictData, 基于API服务的Facade接口实现类

/// </summary>

public class DictDataCaller : BaseApiService<DictDataInfo>, IDictDataService

这个基类，默认封装了对常规数据表业务Web API接口方式的增删改查以及各种复杂的接口处理。  
如果对于一般的Web API（非数据表业务），那么只需要继承的基类做调整即可。

/// <summary>

/// 基于API服务的Facade接口实现类

/// </summary>

public class TestCaller : NormalApiService, ITestService

这个NormalApiService基类，默认只是封装了对token和签名的读取处理，没有特殊的业务接口，具体特定的接口我们来实现处理。

对于WebAPI客户端的调用，我们主要就是需要构建对应的URL，然后通过GET传递或者POST传递一些参数，并读取HTML结果，把它解析为对应的类型数据即可，如下代码所示。

/// <summary>

/// 根据字典类型名称获取对应的字典记录

/// </summary>

/// <param name="dictTypeName">字典类型名称</param>

/// <returns></returns>

public List<DictDataInfo> FindByDictType(string dictTypeName)

{

var action = System.Reflection.MethodBase.GetCurrentMethod().Name;

string url = GetTokenUrl(action) + string.Format("&dictTypeName={0}", dictTypeName);

List<DictDataInfo> result = JsonHelper<List<DictDataInfo>>.ConvertJson(url);

return result;

}

通过GetTokenUrl(action) 函数获取对应的URL地址，由于传入一个参数，接口这里没有发生数据修改，是GET方式提交参数数据，因此把参数附加在URL即可。  
也就是下面代码实现了完整Web API地址的构建。

string url = GetTokenUrl(action) + string.Format("&dictTypeName={0}", dictTypeName);

构建好这些URL地址后，我们通过获取对应Web API的结果并进行序列号到具体对象即可。如下代码所示。

List<DictDataInfo> result = JsonHelper<List<DictDataInfo>>.ConvertJson(url);

关于Web API接口的设计文章，可以参考我的随笔。

* [Web API接口设计经验总结](https://www.jianshu.com/p/3efc0ff455f1)
* [Web API应用架构设计分析（1）](https://www.jianshu.com/p/e43933cdf582)
* [Web API应用架构设计分析（2）](https://www.jianshu.com/p/c2cd37f063ef)  
  具体的Web API接口的使用，可以参考随笔：
* [Web API应用架构在Winform混合框架中的应用（1）](https://www.jianshu.com/p/db25986eb27e)
* [Web API应用架构在Winform混合框架中的应用（2）--自定义异常结果的处理](https://www.jianshu.com/p/0cdbb7a92021)
* [Web API应用架构在Winform混合框架中的应用（3）--Winfrom界面调用WebAPI的过程分解](https://www.jianshu.com/p/78406e7a2424)
* [Web API应用架构在Winform混合框架中的应用（4）--利用代码生成工具快速开发整套应用](https://www.jianshu.com/p/2cc77246f4f1)
* [Web API应用架构在Winform混合框架中的应用（5）--系统级别字典和公司级别字典并存的处理方式](https://www.jianshu.com/p/930f8faca684)

通过以上的封装处理，那么对于业务表的Web API接口调用，具体使用客户端的代码如下所示。

var dictType = CallerFactory<IDictTypeService>.Instance.GetTree();

Console.WriteLine(dictType.ToJson());

var dictData = CallerFactory<IDictDataService>.Instance.GetAllDict();

Console.WriteLine(dictData.ToJson());

如果对于非数据表业务的Web API接口调用，具体使用客户端的代码如下所示。

var testAction = CallerFactory<ITestService>.Instance.TestAction();

Console.WriteLine(testAction.ToJson());

var test = CallerFactory<ITestService>.Instance.Test("123");

Console.WriteLine(test.ToJson());

这样，不管是在Web项目里面，还是在Winform项目里面，或者在跨平台的IOS项目里面（或者安卓项目），都可以以相同的方式消费Web API，这样我们所有的数据入口在一个地方，可以集中业务接口的统一开发，并且可以有效管理我们的数据提供的性能问题，如统一缓存处理，统一权限处理...  
感谢大家对本文章的细心阅读，希望对您的开发有所启发或帮助。