WebAPI是建立在MVC和WCF的基础上的，原来微软老是喜欢封装的很多，这次终于愿意将http编程模型的相关细节暴露给我们了。在之前的介绍中，基本上都基于.NET 4.5之后版本，其System.Net.Http程序集非常的丰富，而老版本的则相对较弱。在WebAPI v1.0(和ASP.NET MVC4在一起的版本)很多的类和接口并不存在，同时对Task异步编程（ApiController默认提供异步执行方法）的支持还有一些欠缺（缺少不少方便的扩展方法），在使用时会有一些需要注意的地方，由于一些老的项目用的.NET 4.0的程序集，无法升级和使用一些新的dll，因而部分功能需要自己来考虑，本文旨在将自己遇到的一些困难分享给大家。

1. 路由设置

在Global.asax文件中需要注意WebApi的路由要先于MVC的路由进行注册，不然会出现路由无效的情况。

前端路由地址的提供，使用@Url.HttpRouteUrl("AddedApi", new { controller = "SMSCenterApi", action = "MassTexting" })来生成路由，与MVC的方式有一些差异，需要注意。

1. 参数绑定

包括ModelBinder和MediaTypeFormatter两种方式，与MVC不同（MVC均使用ModelBinder进行绑定）。前者包括针对数组、集合、字典、简单和复杂类型的绑定器，后者其实就是一个序列化器，默认包括3中：Json.NET的json序列化器（用的最多）；DataContractSerializer和XMLSerializer用于序列化XML；最后一种解码表单URL，编码主体数据。这些格式化器均在System.Net.Http.Formatting命名空间中。

相关的特性包括：ModelBindingAttribute，默认绑定逻辑；FormUriAttribute，只从Uri获取值；FromBodyAtrribute，使用MediaTypeFormatter媒体格式化器，也是我们在WebAPi最常用的，再次提醒一下，一定要提供contentType哦，比如”application/json”。

Tip:模型绑定常见问题，WebAPI的格式化器Formatter需要提供相应的contentType才会起作用，返回值通过dataType设置（默认为XML），一定不能忘记内容协商,需要注意内容协商，附上一个ajax调用的例子，我在这也吃了很大的亏，默认formatter其实做了很多事情哦。

这儿强烈提醒的是dataType表示返回值类型，contentType为请求体的类型，熊二你个二货，内容协商是必须的，不然别人哪知道怎么做!此外，这个的dataType=’json’最终反应到http请求体中为**Accept: application/json**,

这个对于你使用过滤器拦截并新建httpMessageResponse的HttpContent时非常有用，最后的例子会涉及这部分内容。

1. 过滤请求

过去我们常常将一些验证逻辑和异常处理逻辑放在Controller中，极大的增加了Controller的复杂性，完全可以通过面向切面（AOP）来处理，在.NET 4.0提供的相关基类和接口如下所示：

|  |  |
| --- | --- |
| 异步接口和同步基类 | 用途 |
| IAuthenticationFilter  AuthorizationFilterAttribute | 认证过滤器可以在参数绑定发生以前运行，它们计划过滤没有正确认证且请求争议操作的请求  认证过滤器先于操作过滤器运行，应用场景为验证客户身份，例如去Cookie或HttpHead中获取相关验证信息 |
| IActionFilter  ActionFilterAttribute | 操作过滤器在参数绑定时发生，并封装API操作方法调用之后运行，允许在调度操作之前，完成执行之后拦截。操作过滤器的目标时允许开发人员增加和替换操作的输入值和输出结果。如果说自定义绑定器或格式化器是用于扩展正常状态下解析数据的话，那么过滤器可以用在一些特殊情况下 |
| IExceptionFilter  ExceptionFilterAttribute | 当调用操作抛出异常时，就会调用异常过滤器，可以检查异常，并采取一些操作，例如记录日志、提供新的响应对象来处理异常等 |

Tip: 在MVC4中，推荐使用同步基类，在以后的版本中推荐使用异步接口对应用程序进行扩展。

此外，需要注意过滤器的使用范围，包括：全局，在FilterConfig中添加；类级别过滤器，通过添加特性的方式；方法级别过滤器。

默认提供AuthorizeAttribute完成基础验证，AllowAnonymousAttribute提供匿名验证的情况。此外还提供一个关于OData的第三方解决方案，包括可以自动支持OData查询语法的QueryableAttribute（如$top和$filter等）。

1. 其他小知识点

WebAPI的托管，包括通过System.Web.Http.WebHost.dll的IIS托管，配置对象为GlobalConfiguration；自托管的配置，通过Mocrosoft.AspNet.WebApi.Selfhost。

启动依赖注入，13章学习

可以通过HttpConfiguration.Service获取IApiExplorer服务，即全领域搜索可用服务。

通过ITraceWriter来跟踪应用程序，可以很方便的和ETW、Log4net、ELMAH等跟踪服务集成。

1. 示例

Controller:

|  |
| --- |
| public class SMSCenterApiController : ApiController  {  [HttpPost]  [CheckPermissionFilter]  [ApiExceptionFilter]  public WebApiResult MassTexting([FromBody]SMSCenterViewModel model)  {  var result = new WebApiResult { Status = WebApiResultStatus.Fail, Message = string.Empty };  if (ModelState.IsValid)  {  int sendedNum = SMSCenterBL.Instance.MassTexting(model.SMSContent, model.PhoneList, "xionger");  result.Status = WebApiResultStatus.Success;  result.Message = sendedNum.ToString();  }  else  {  result.Message = "输入消息检验未通过！";  }  return result;  } |

Jquery调用:

|  |
| --- |
| jQuery.ajax({  type: 'POST',  url: url,  contentType: "application/json",  dataType: 'json',  data: postData,  beforeSend: function (request) {  request.setRequestHeader("smsToken", smsToken);  },  success: function (data) {  if (data.Succ == 1) {  var msg = "发送结束。成功{0} --- 共{1}";  msg = msg.format(data.Count, data.Count);  alert(msg);  }  else {  alert("发送失败。");  }  },  error: function (data) {  alert("发送失败。");  }  }); |

CheckPremissionFilter:

|  |
| --- |
| public class CheckPermissionFilterAttribute : AuthorizationFilterAttribute  {  #region 验证权限  /// <summary>  /// 验证权限  /// </summary>  public override void OnAuthorization(HttpActionContext actionContext)  {  var token = GetHttpToken(actionContext.Request.Headers);  if (SMSTokenHelper.CheckToken(token))  {  base.OnAuthorization(actionContext);  }  else  {  throw new BizException("当前请求没有访问权限！");  }  }  #endregion  #region 辅助方法  /// <summary>  /// 获得请求头中的token信息  /// </summary>  private string GetHttpToken(HttpRequestHeaders headers)  {  IEnumerable<string> tokenCollection;  if (headers.TryGetValues(ConfigHelper.SMSCENTER\_TOKEN\_NAME, out tokenCollection))  {  var token = tokenCollection.FirstOrDefault();  return token;  }  return null;  }  #endregion  } |

SMSTokenHelper

|  |
| --- |
| internal class SMSTokenHelper  {  public static string CreateToken(string eid)  {  //1.使用eid，moduleID，当前时间构建认证对象  var token = new SMSToken()  {  EID = eid,  ModuleID = ConfigHelper.SMSCENTER\_MODULE\_ID,  CurrentTime = DateTime.Now.ToString()  };  //2.转化为Json字符串  var tokenString = JsonConvert.SerializeObject(token);  //3.将json字符串加密  var encryptToken = DESHelper.DESEncrypt(tokenString);  return encryptToken;  }  public static bool CheckToken(string token)  {  try  {  //1.解密  var tokenString = DESHelper.DESDecrypt(token);  //2.反序列化为对象  var smstoken = JsonConvert.DeserializeObject<SMSToken>(tokenString);  //3.验证结果  if (ConfigHelper.SMSCENTER\_MODULE\_ID == smstoken.ModuleID)  {  return true;  }  }  catch { }  return false;  }  private class SMSToken  {  public string EID { get; set; }  public string ModuleID { get; set; }  public string CurrentTime { get; set; }  }  #region 辅助类  /// <summary>  /// DES加密解密  /// </summary>  private class DESHelper  {  /// <summary>  /// 获取密钥  /// </summary>  private static string Key  {  get { return @"P@+#wG+Z"; }  }  /// <summary>  /// 获取向量  /// </summary>  private static string IV  {  get { return @"L%n67}G/Mk@k%:~Y"; }  }  /// <summary>  /// DES加密  /// </summary>  /// <param name="plainStr">明文字符串</param>  /// <returns>密文</returns>  public static string DESEncrypt(string plainStr)  {  byte[] bKey = Encoding.UTF8.GetBytes(Key);  byte[] bIV = Encoding.UTF8.GetBytes(IV);  byte[] byteArray = Encoding.UTF8.GetBytes(plainStr);  string encrypt = null;  DESCryptoServiceProvider des = new DESCryptoServiceProvider();  try  {  using (MemoryStream mStream = new MemoryStream())  {  using (CryptoStream cStream = new CryptoStream(mStream, des.CreateEncryptor(bKey, bIV), CryptoStreamMode.Write))  {  cStream.Write(byteArray, 0, byteArray.Length);  cStream.FlushFinalBlock();  encrypt = Convert.ToBase64String(mStream.ToArray());  }  }  }  catch { }  des.Clear();  return encrypt;  }  /// <summary>  /// DES解密  /// </summary>  /// <param name="encryptStr">密文字符串</param>  /// <returns>明文</returns>  public static string DESDecrypt(string encryptStr)  {  byte[] bKey = Encoding.UTF8.GetBytes(Key);  byte[] bIV = Encoding.UTF8.GetBytes(IV);  byte[] byteArray = Convert.FromBase64String(encryptStr);  string decrypt = null;  DESCryptoServiceProvider des = new DESCryptoServiceProvider();  try  {  using (MemoryStream mStream = new MemoryStream())  {  using (CryptoStream cStream = new CryptoStream(mStream, des.CreateDecryptor(bKey, bIV), CryptoStreamMode.Write))  {  cStream.Write(byteArray, 0, byteArray.Length);  cStream.FlushFinalBlock();  decrypt = Encoding.UTF8.GetString(mStream.ToArray());  }  }  }  catch { }  des.Clear();  return decrypt;  }  }  #endregion  }  } |

Tip: DES加密部分借鉴博主IT合伙人文章<http://www.cnblogs.com/IT-haidong/p/4856848.html>

最后，补充一个在MVC4.0下的自定义ModerBinder,非常的简单，但可以帮助实现json数据的绑定，简化使用。当然使用JQuery的form.serialize()，将数据转化为form提交，然后应用默认的绑定器也是ok的。以前一直form提交也没有认真去想想form的区别，其实form是用“**&**”符号来连接数据的。

|  |
| --- |
| [AttributeUsage(AttributeTargets.Parameter)]  public class FromBodyAttribute : CustomModelBinderAttribute  {  private static readonly ILog \_logger = LogManager.GetLogger(typeof(FromBodyAttribute));  public override IModelBinder GetBinder()  {  return new JsonModelBinder();  }  public class JsonModelBinder : IModelBinder  {  public object BindModel(ControllerContext controllerContext,  ModelBindingContext bindingContext)  {  //Verify content-type  if (!controllerContext.HttpContext.Request.ContentType.ToLower().Contains("json"))  {  return null;  }  try  {  var jsonString = GetJsonString(controllerContext.HttpContext.Request.InputStream);  var result = JsonConvert.DeserializeObject(jsonString, bindingContext.ModelType);  return result;  }  catch(Exception ex)  {  \_logger.Warn(ex.Message, ex);  }  return null;  }  private string GetJsonString(Stream stream)  {  var jsonString = string.Empty;  using (var sr = new StreamReader(stream))  {  stream.Position = 0;  jsonString = sr.ReadToEnd();  }  return jsonString;  }  }  } |

参考资料：

1. （美）加洛韦. ASP.NET MVC 4高级编程(第4版)[M]. 北京:清华大学出版社, 2012.