接下来进入的是俺在ASP.NET学习中最重要的WebAPI部分，在现在流行的互联网场景下，WebAPI可以和HTML5、单页应用程序SPA等技术和理念很好的结合在一起。所谓ASP.NET WebAPI，其核心概念就是构建REST风格的Web服务，把一起数据视为资源，无论是服务请求或者是数据操作，与以前的SOAP和XML-RPC架构风格有很大不同。说道这，很多读者可能想到WCF中不是早都有了REST风格的服务么，为什么还需要这个WebAPI？确实如此，不过WCF中的该类型服务显得比较复杂，因为其通信管道的构成由于集成了多种不同的通信协议，自然的其基础程序集就显得非常的庞大臃肿。

简单来说，WebAPI就是简单高效，“你值得拥有”！让我们通过临摹蒋老师的例子对它有个初步的了解，后端代码如下：

|  |
| --- |
| public class ContactsController : ApiController  {  private static IList<Contact> contacts = new List<Contact>  {  new Contact{  Id="001",  Name="Xixi",  PhoneNo="12132432",  EmailAddress="xixi@gmail.com"  },  new Contact{  Id="002",  Name="XiongEr",  PhoneNo="312",  EmailAddress="XiongEr@gmail.com"  }  };  public IEnumerable<Contact> Get()  {  return contacts;  }  public Contact Get(string id)  {  return contacts.FirstOrDefault(c => c.Id == id);  }  public void Put(Contact contact)  {  contact.Id = Guid.NewGuid().ToString();  contacts.Add(contact);  }  public void Post(Contact contact)  {  Delete(contact.Id);  contacts.Add(contact);  }  public void Delete(string id)  {  Contact tempContact = contacts.FirstOrDefault(c => c.Id == id);  contacts.Remove(tempContact);  }  } |

前端代码如下：

|  |
| --- |
| <html>  <head>  <title>联系人管理</title>  <script type="text/javascript" src="~/Scripts/jquery-1.8.2.js"></script>  <script type="text/javascript" src="~/Scripts/knockout-2.2.0.js"></script>  </head>  <body>  <div id="contacts">  <table>  <tr>  <th>姓名</th>  <th>电话号码</th>  <th>Email地址</th>  <th></th>  </tr>  <tbody>  <!-- ko foreach: allContacts -->  <tr>  <td data-bind="text: Name" />  <td data-bind="text: PhoneNo" />  <td>  <input type="text" class="textbox long" data-bind="value: EmailAddress" />  </td>  <td>  <a href="#" data-bind="click: $root.updateContact">修改</a>  <a href="#" data-bind="click: $root.deleteContact">删除</a>  </td>  </tr>  <!-- /ko -->  <tr data-bind="with: addedContact">  <td>  <input type="text" class="textbox" data-bind="value: Name" /></td>  <td>  <input type="text" class="textbox" data-bind="value: PhoneNo" /></td>  <td>  <input type="text" class="textbox long" data-bind="value: EmailAddress" /></td>  <td><a href="#" data-bind="click: $root.addContact" />添加</td>  </tr>  </tbody>  </table>  </div>  <script type="text/javascript">  function ContactViewModel() {  self = this;  self.allContacts = ko.observableArray();  self.addedContact = ko.observable();  //加载联系人列表  self.loadContacts = function () {  $.get("/api/contacts", null, function (data) {  self.allContacts(data);  var emptyContact = { Id: "", Name: "", PhoneNo: "", EmailAddress: "" };  self.addedContact(emptyContact);  });  }  //添加联系人  self.addContact = function (data) {  if (!self.validate(data)) {  return;  }  $.ajax({  url: "/api/contacts/",  data: self.addedContact(),  type: "PUT",  success: self.loadContacts  });  };  //修改联系人  self.updateContact = function (data) {  $.ajax({  url: "/api/contacts/",  data: data,  type: "POST",  success: self.loadContacts  });  };  //删除联系人  self.deleteContact = function (data) {  $.ajax({  url: "/api/contacts/" + data.Id,  type: "DELETE",  success: self.loadContacts  });  };  self.validate = function (data) {  if (data.Name && data.PhoneNo && data.EmailAddress) {  return true;  }  alert("请输入完整联系人信息！");  return false;  }  self.loadContacts();  }  ko.applyBindings(new ContactViewModel());  </script>  </body>  </html> |

这个像补充的是，蒋老师在这用的是自带的knockoutJS作为MVVM风格的部分前端框架。关于这一块，有一个问题困扰了我很久，就是KnockoutJS和AngularJS谁的适用性更强，其实它们没有可比性，KnockoutJS只提供了部分的工作。以下链接是对此问题的解释，结论是我将学习并使用AngularJS。

<http://blog.darkthread.net/post-2014-06-07-go-to-angularjs.aspx>

说到这，我还想到了学习中的一个困惑，那么多的IOC框架到底哪个相对更好一些？结论是Autofac，它以被使用在Orchard开源的CMS系统中，顺道提一嘴，nopCommerce的.net开源电商系统也不错哦。之前IOC框架对比的详情请见如下链接，李平老师做了最好的解释：

<http://www.cnblogs.com/liping13599168/archive/2011/07/17/2108734.html>

接下来，介绍ASP.NET WebAPI的服务器管道，这一块和之前学习的ASP.NET MVC管道很相似，但也有一些差异，不过个人感觉这个管道更加的像J2EE的管道了。由于很多内容比较相似，将进行简单的介绍，不过框架中异步编程模型用的很多，值得学习参考。下图简单的表述了框架对请求的处理过程：



框架通过单例提供HttpControllerHandler对象，多个HttpWebRoute共享对象，并且它将创建右侧的ASP.NET Web API处理管道，通过调用BeginProcessRequest方法激活管道运转。该管道其实就是HttpMessgaeHandler链，HttpServer和HttpControllerDispatcher可以看做两个特殊的HttpMessageHandler，接下来通过表格的形式对相关类型进行简单的介绍:

|  |  |
| --- | --- |
| 类型 | 简介 |
| HttpMessageHandler | 核心类，针对请求的处理实现在SendAsync中，针对响应的处理通过返回类型Task<HttpResponseMessage>完成 |
| HttpRequestMessage | Content属性封装Http主体信息 |
| HttpResponseMessage | StatusCode、ReasonPhrase属性表示响应状态码与描述 |
| DelegatingHandler | 用于构建处理链，通过InnerHanlder属性进行传递，是责任链模式的实现？ |
| HttpServer | Dispatcher属性指向最终的分发器对象，Configuration属性包含了所有的配置信息。 |
| HttpConfiguration | DependencyResolver  Filters: AuthorizationFilter, ActionFilter, ExceptionFilter  Formatters:返回格式化器列表  IncludeErrorDetailPolicy:客户端异常显示策略  Properties  Services: 返回ServiceContainer，一个简易IocR容器，默认实现为DefaultServices，很常用。 |

HttpControllerHandler以延迟加载的方式来创建HttpServer，字典属性Properties以Key为“MS\_HttpContext”、“MS\_HttpRouteData”的形式传递相关数据。HttpControllerDispatcher负责最后对请求做最后的处理，包括对ApiController的激活和目标Action的执行等操作，用下表简述该过程：

|  |  |
| --- | --- |
| 行为 | 简介 |
| HttpController的激活 | 借助HttpControllerDescriptor，完成HttpController类型解析、选择、创建等操作，可以通过自定义DependencyResolver或HttpControllerActivator来实现基于IOC的HttpController的激活。 |
| HttpController的执行 | 通过ExecuteAsync方法，参数为HttpControllerContext，注意UrlHelper中Link代表绝对地址，Route相对地址 |
| Action的选择 | HttpActionDescriptor的ExecuteAsync方法实现Action的执行，Action支持7中不同的HTTP方法，默认为POST。通过HttpActionSelector组件实现对目标Action的选择，方法GetActionMapping的返回值为一个ILookup<string, HttpActionDescriptor>类型 |
| Model元数据的解析 | 与MVC基本一致 |
| Action参数绑定 | 借助HttpParameterDescriptor、HttpActionBinding，通过HttpParameterBinding对象的ExecuteBindingAsync完成绑定，具体的实现类有:  CancellationTokenParameterBinding  ErrorParameterBinding  **FomatterParameterBinding**：消息主体,html,json,xml  HttpRequestParameterBinding：HttpRequestMessage  **ModelBinderParameterBinding**：查询字符串，路由数据 |
| Model的验证 | 包括DataAnnotationModelValidator  RequiredMemberModelValidator  ValidatableObjectAdapter  ErrorModelValidator等验证器，需要注意的是该框架中验证过程是递归的，与MVC有点不同。 |
| Action的执行与结果的响应 | 通过HttpActionInvoker的InvokerActionAsync方法激活Action，通过ActionResultConverter将Action的返回值转换为HttpResponseMessage，转换器包括：  ResponseMessageResultConverter  ValueResultConverter<T>  VoidResultConverter  3个内置Filter筛选器的作用与MVC中的类似 |

补上IOC实现的代码和HttpParameterBinding的流程图：

|  |
| --- |
| public class NinjectDependencyResolver : IDependencyResolver  {  private List<IDisposable> disposableServices = new List<IDisposable>();  public IKernel Kernel { get; private set; }  public NinjectDependencyResolver(NinjectDependencyResolver parent)  {  this.Kernel = parent.Kernel;  }  public NinjectDependencyResolver()  {  this.Kernel = new StandardKernel();  }  public void Register<TFrom, TTo>() where TTo : TFrom  {  this.Kernel.Bind<TFrom>().To<TTo>();  }  public IDependencyScope BeginScope()  {  return new NinjectDependencyResolver(this);  }  public object GetService(Type serviceType)  {  var service = this.Kernel.TryGet(serviceType);  this.AddDisposableService(service);  return service;  }  public IEnumerable<object> GetServices(Type serviceType)  {  foreach (var service in this.Kernel.GetAll(serviceType))  {  this.AddDisposableService(service);  yield return service;  }  }  public void Dispose()  {  foreach (var disposable in disposableServices)  {  disposable.Dispose();  }  }  private void AddDisposableService(object service)  {  IDisposable disposable = service as IDisposable;  if (null != disposable && !disposableServices.Contains(disposable))  {  disposableServices.Add(disposable);  }  }  } |
| public class WebApiApplication : System.Web.HttpApplication  {  protected void Application\_Start()  {  //自定义操作  NinjectDependencyResolver dependencyResolver = new NinjectDependencyResolver();  dependencyResolver.Register<IContactRepository, DefaultContactRepository>();  GlobalConfiguration.Configuration.DependencyResolver = dependencyResolver;  }  } |

HttpParameterBinding流程图：



最后介绍与WebAPI客户端调用相关的内容，提到调用大家第一反应就是在Web页面中通过javascript进行Ajax调用，获取数据并呈现，服务的消费者是前端页面，这只是调用的主要方式之一。另外一种就是通过HttpClient来进行调用，这和Web Service调用很相似，服务的消费者是一般应用程序。HttpClient类继承之抽象类HttpMessageInvoker，核心方法SendAsync包括HttpRequestMessage的参数和HttpResponseMessage的返回类型，和之前服务器端的HttpMessageHandler类型一样，实际上HttpClient就是一个该类的封装。HttpCompletionOption用于设置响应完成的标志，包括读完消息头和读完消息体。属性BaseAddress用于指定WebAPI基地址，DefaultRequestHeader用于添加任意的报头，MaxResponseContentBufferSize表示读取缓存区的大小，默认2G，Timeout表示超时时限，默认100s。GetAsync, GetByteArrayAsync, GetStreamAsync, GetStringAsync用于HTTP-GET请求，其他方法也有相似定义。下面通过一个服务器端自我寄宿，客户端一般调用的例子完成学习，需要注意通过Nuget添加SelfHost和Client的库，代码如下所示：

|  |
| --- |
| //服务器端  class Program  {  static void Main(string[] args)  {  var config = new HttpSelfHostConfiguration("http://127.0.0.1:3721");  config.Routes.MapHttpRoute("DefaultApi", "api/{controller}/{id}", new { id = RouteParameter.Optional });  using (var httpServer = new HttpSelfHostServer(config))  {  httpServer.OpenAsync().Wait();  Console.WriteLine("按任意键关闭WebAPI");  Console.Read();  }  }  } |
| //客户端  class Program  {  static void Main(string[] args)  {  Uri baseAddress = new Uri("http://127.0.0.1:3721");  var httpClient = new HttpClient { BaseAddress = baseAddress };  IEnumerable<Contact> contacts = httpClient.GetAsync("api/contacts").Result.Content.ReadAsAsync<IEnumerable<Contact>>().Result;  Console.WriteLine("当前联系人列表：");  ListContacts(contacts);  var contact = new Contact { Id = "003", Name = "qiuzi", EmailAddress = "qiuqiu@gmail.com", PhoneNo = "95580" };  Console.WriteLine("\n添加联系人003: ");  httpClient.PutAsync<Contact>("/api/contacts", contact, new JsonMediaTypeFormatter()).Wait();  contacts = httpClient.GetAsync("api/contacts").Result.Content.ReadAsAsync<IEnumerable<Contact>>().Result;  ListContacts(contacts);  contact = new Contact { Id = "003", Name = "qiuzi", EmailAddress = "zhaoyun@outlook.com", PhoneNo = "123" };  Console.WriteLine("\n修改联系人003: ");  httpClient.PostAsync<Contact>("/api/contacts", contact, new XmlMediaTypeFormatter()).Wait();  contacts = httpClient.GetAsync("api/contacts").Result.Content.ReadAsAsync<IEnumerable<Contact>>().Result;  ListContacts(contacts);  Console.WriteLine("\n删除联系人003: ");  httpClient.DeleteAsync("/api/contacts/003").Wait();  contacts = httpClient.GetAsync("api/contacts").Result.Content.ReadAsAsync<IEnumerable<Contact>>().Result;  ListContacts(contacts);  Console.Read();  }  private static void ListContacts(IEnumerable<Contact> contacts)  {  foreach (var contact in contacts)  {  Console.WriteLine("{0, -6}{1, -6}{2, -20}{3, -10}", contact.Id, contact.Name, contact.EmailAddress, contact.PhoneNo);  }  }  } |

注：本文主要供自己学习，不妥之处望见谅。

参考资料：

[1]蒋金楠. ASP.NET MVC4框架揭秘[M]. 上海:电子工业出版社, 2012. 445-526