这部分内容的学习，已经放了大半年时间了，果断补充上，尽早将过去遗留的老技术坑都补上。首先将介绍服务幂等性的概念和相关解决方案，这部分也将是本文的理解难点，由于WebAPI是一种Restful风格服务的实现方式，其遵循HTTP标准方法，因此理解好这部分概念，对于提供良好的业务服务显得非常重要。之后则将介绍Signal这一长连接通讯的集成解决方案的概念和实践，这部分在交互式的Web场景中非常有效。最后将补充Owin、IOC、EnterpriseLibrary等相关知识，这些也都是.NET程序员比较容易忽视的知识点，这些知识在快速搭建Solution上有很大的帮助。

最早接触这个概念还是在一次面试的过程中，当时记得自己只能是通过卖萌将这一概念一笔带过，由于当时的项目实践相对较少，且被微软便捷的服务搭建所欺骗，以为搭建一个webService只用在IDE中添加有一个.asmx就万事大吉了，其他项目通过一个服务引用就算是SOA实现了（WCF额外需要配置一下终结点），其实对于整个服务的概念完全是个门外汉。

那么在实际中，搭建一个服务需要注意那些问题呢？接下来通过一个简单的表格来描述。

|  |  |
| --- | --- |
| 关注因素 | 诠释 |
| 通讯协议的选取 | 例如常见的TCP、HTTP、SOAP等，实际上任何协议都可以作为服务的载体，只要适合相应的场景即可 |
| URL | 服务的地址，服务的消费者可以通过这个地址请求服务 |
| 安全性 | 只有通过认证的请求才能获得服务，不同的服务方法需要不同权限控制 |
| 幂等性 | 对于同一个服务方法来说，相同的请求参数无论请求几次，都将获得相同的结果（简化版的解释，不太完备） |
| 其他 | 与服务相关的概念非常多，比如：在架构层次流行的微服务，用于解耦应用；服务的监控、限流；分布式服务的治理、扩容等。 |

接下来将详细介绍服务幂等性的概念，相关解决方案和基于HTTP协议的服务幂等性等知识。

**基础概念**：从抽象代数的角度，幂等Idempotence就是f(f(x)) = f(x)。也就是说对同一个服务的1次或多次调用，返回的结果相同，且对服务系统的影响相同，接下来通过一个非常简单的图来描述该概念在服务请求场景下的意义。



在上图中，第一次扣款请求成功，但返回丢失，这是Client重发扣款请求，之后成功。在这样的场景下，如果不控制服务的幂等性，就会出现重复扣款的情况出现。

**解决方案**：上例标准的解决方案是，客户端的这个操作需要两个请求，首先需要向服务端申请一个ticket进行扣款操作，之后将该ticket作为参数的一部分发送给Server请求扣款。服务端首先检验该ticket是否已经被使用，若被使用，直接返回成功；若未被使用，则进行相应扣款操作。逻辑上很简单，不过在实践时有几点需要注意：

|  |  |
| --- | --- |
| 注意事项 | 诠释 |
| 幂等的时效性 | 在实际项目中，幂等是具有时效性的，不同的业务需求会有不同的时效性要求。一般来说，对于重要业务操作，通过是与Money有关的操作，要求持久的服务幂等性，这是就需要选用数据库来实现幂等控制，将ticket（流水号，GUID等）保存起来，其特点是安全、低效；对于一般的数据，可以选用缓存来控制，其特点是高效、不稳定。 |
| 数据库实现幂等 | 比如SQL SERVER, 在Read Committed隔离级别下，，建立一张专门的幂等表，通过重复insert的异常来实现幂等，比较规范；也可以通过NoLock读的方式，更搞笑，但存在隐患，推荐前者。Tip:通过SqlException, Number = 2601 |
| 缓存实现幂等 | 比如Redis，通过与key相关的操作，Exists Key |
| CRUD操作的幂等级别 | SELECT最高，为只读级别；UPDATE、DELETE次之，为幂等级别；INSERT为不能幂等级别。也就是说，SELECT操作不管是一次操作还是多次操作，均不改变目标的状态；UPDATE、DELETE只在第一次调用时会改变状态，之后不会；而INSERT则每次均改变状态。 |
| CAS操作 | 之前一直有个疑惑，就是关于UPDATE操作，比如a++的情况下，这个操作不是幂等的，但实际上，这不是一个原子操作，其涉及一次查询和一次修改，在很多语言中，都支持称为CAS(CompareAndSet)的原子操作。 |
| Ticket的生成 | 根据不同的场景，可以是客户端生成，也可以是服务端生成，当然，最好的方案是设置好指定规则，然后由客户端生成ticket，比如GUID组合客户端的标识的方式，因为这样可以减少服务端的压力，无论是CPU还是网络。 |

**基于HTTP协议的服务幂等性**：在之前的表格中，已介绍过CRUD操作所对应的幂等级别，那么对应到HTTP的操作呢？很简单，GET表示查询操作，PUT和DELETE表示更新和删除操作，POST表示插入操作，因此POST操作需要添加幂等控制的。当然了，在WebAPI的实际设计中，接口的URL格式和http报文中Body的参数值会需要进一步的思考。

此外，大家也可以查阅博主Todd Wei的博文http的幂等性<http://www.cnblogs.com/weidagang2046/archive/2011/06/04/idempotence.html>

SignalR这个名字，咋一看还挺高大上的，实际上和WCF、AJAX类似，并不是什么新技术，而是对已有技术的一种整合，集成了客户端和服务端的库。不知道大家还记不记的，大学时学习的Windows网络编程，当时通过WinSocket搭建了一个聊天室，其实这里的SignalR也一样，最常见的应用仍然是聊天室场景，不过变成浏览器和服务器之间，而不是过去的Client与服务器之间。那么它与H5的WebSocket有什么区别么？准确来说，SignalR整合了WebSocket，在浏览器支持H5的情况下就使用WebSocket，若不支持，就通过长轮训的方式，算是一种兼容性的整体解决方案。

简单来说，记住一点就好，SignalR支持双向通信的长连接，其是对http请求-响应模式的有力补充。其提供一个简单的API用于创建服务端到客户端的远程过程调用(RPC),以便从服务器端.NET代码中调用客户端浏览器中的js代码。

SignalR的API包含两种客户端和服务器之间进行通信的模型：永久连接和Hubs。。接下来通过来通过一段代码，走进SignalR的世界。

前端页面

|  |
| --- |
| 前端页面  @{  ViewBag.Title = "Chat";  }  <h2>Chat</h2>  <div class="container">  <input type="text" id="message" />  <input type="button" id="sendmessage" value="Send" />  <input type="hidden" id="displayname" />  <ul id="discussion"></ul>  </div>  @section scripts{  <script src="~/Scripts/jquery.signalR-2.2.0.min.js"></script>  <script src="~/signalr/hubs"></script>  <script>  $(function () {  //reference the auto-generated proxy for the hub  var chat = $.connection.chatHub;  chat.client.addNewMessageToPage = function (name, message) {  //add the message to the page  $('#discussion').append('<li><strong>' + htmlEncode(name) + '</strong>: ' + htmlEncode(message) + '</li>');  };  //get the user name and store it to prepend to messages.  $('#displayname').val(prompt('Enter your name:', ''));  //set initial focus to message input box  $('#message').focus();  //start the connection  $.connection.hub.start().done(function () {  $('#sendmessage').click(function () {  chat.server.send($('#displayname').val(), $('#message').val());  $('#message').val('').focus();  });  });  });  //this optional function html-encodes messages for display in the page  function htmlEncode(value) {  var encodedValue = $('<div/>').text(value).html();  return encodedValue;  }  </script>  }  Startup文件  [assembly: OwinStartup(typeof(Sory.Framework.SignalRDemo.Startup))]  namespace Sory.Framework.SignalRDemo  {  public class Startup  {  public void Configuration(IAppBuilder app)  {  //Any connection or hub wire up and configuration should go here  app.MapSignalR();  }  }  }  Hub文件  public class ChatHub : Hub  {  public void Send(string name, string message)  {  Clients.All.addNewMessageToPage(name, message);  }  } |

相关学习可以参见张善友大神的4年前的博文SingalR QuickStart, <http://www.cnblogs.com/shanyou/archive/2012/07/28/2613693.html>

另外还有博主[will\_晓柠](http://home.cnblogs.com/u/vance/)的博文<http://www.cnblogs.com/vance/p/SignalR.html>，相对版本更新一些，此外其翻译的Signal入门非常的赞，必须顶<http://files.cnblogs.com/files/wanliwang01/SignalR2.0.pdf>

简单来说，类似J2EE中javaWeb的相关标准（servlet），之后各种不同的容器厂商均可以针对该接口提供自己的实现，比如Tomcat、Weblogic等。随着微软慢慢走向开发，也提出了相应的接口标准，这个标准就是Owin，我们常见的IIS其实就是该标准的一个官方实现。记得身边的一位大牛说过，JAVA就是先自己开发，其他的厂商也模仿着开发，之后为了统一就建立标准，之后的版本大家都按照这个标准来，类似于实践推导出理论，理论再来指导实践的过程。

在Owin中，将不再使用ASP.NET管道处理请求，而是使用Owin管道来处理请求，其通过一个Dictionary来传递上下文信息，其信息如下表所示。

|  |  |  |
| --- | --- | --- |
| Key Name | 类型 | 描述 |
| Owin.RequestBody | Stream | Http请求体 |
| Owin.RequestHeaders | IDictionary<string, string[]> | Http请求头 |
| Owin.RequestMethod | String | 请求方法, get, post等 |
| Owin.RequestPathBase | String | URL根 |
| Owin.RequestPath | String | URL路径，根后面的部分 |
| Owin.RequestProtocol | String | 协议名称和版本, http/1.1 |
| Owin.RequestQueryString | String | 查询字符串 |
| Owin.RequestSchema | String | http或者https |

Owin接口微软的官方实现叫做Katana（武士刀）的组件包括：Application, Middleware, Server和Host。在vs2013中的MVC5模板中，添加如下DLL，和在nuget中package添加如下配置。

|  |
| --- |
| <package id=”Owin” version=”1.0” targetFramework=”net451”/>  <package id=”Microsoft.Owin” version=”2.0.0” targetFramework=”net451”/>  <package id=”Microsoft.Owin.Host.SystemWeb” version=”1.0” targetFramework=”net451”/>  <package id=”Microsoft.Owin.Security” version=”1.0” targetFramework=”net451”/>  <package id=”Microsoft.Owin.Security.Cookies” version=”1.0” targetFramework=”net451”/> |

需要提及的是，其通过一个称为claims-based认证方式进行用户的认证，与原有Form认证方式有一定区别，简单来讲，类似于windows的token认证（单登SSO）。

详细信息，可以参考Jesse博主的博文http://www.cnblogs.com/jesse2013/p/aspnet-identity-claims-based-authentication-and-owin.html#what-is-owin

IOC框架用于解耦系统不同层次间的依赖关系，便于系统的扩展。当然其也会在一定程度上增加系统的复杂性，影响系统的效率，那么选择一个高效可靠IOC显得非常重要。IOC控制反转的相关组件非常的多，包括微软的Unity，第三方的Autofac,CastleWindsor, Spring.NET, StructureMap, Ninject等，其中Unity表现的中规中矩，在大量迭代情况下（>1000000），**Autofac**和StructureMap效率最高。就我个人而言，由于公司的组件库支持Unity,那么就不用选了，哈哈。

详情请见Leepy大神的博文：http://www.cnblogs.com/liping13599168/archive/2011/07/17/2108734.html

微软的企业库包含的模块非常多，比如缓存、数据存取、日志、IOC、AOP、异常处理等，由于现代企业往往都提供统一的日志管理系统和DAL系统，因此微软企业库仍然有价值的部分为AOP和IOC等部分，其实都是面向切面的应用（我们常用的WCF、MVC、WebAPI项目由于拦截器的存在，已经实现了AOP），当然在一部分做的最好的应该是Spring。

接下来通过一个示例，来对AOP的应用，有一个基本的了解（通过Unity的Interception扩展）。

|  |
| --- |
| 配置：  <configuration>  <configSections>  <section name="unityInterception" type="Microsoft.Practices.Unity.Configuration.UnityConfigurationSection, Microsoft.Practices.Unity.Configuration"/>  </configSections>  <unityInterception>  <sectionExtension type="Microsoft.Practices.Unity.InterceptionExtension.Configuration.InterceptionConfigurationExtension  , Microsoft.Practices.Unity.InterceptionExtension.Configuration">  <sectionExtension type="Microsoft.Practices.Unity.InterceptionExtension.Configuration.InterceptionConfigurationExtension  , Microsoft.Practices.Unity.Interception.Configuration" />  <container>  <extension type="Interception"/>  <register type="LipinInvoice.BL.AOPDemo, LipinInvoice.BL">  <interceptor type="TransparentProxyInterceptor" />  <policyInjection />  </register>  </container>  </sectionExtension>  </unityInterception>  </configuration>  拦截类  public class ExceptionHandler : ICallHandler  {  public IMethodReturn Invoke(IMethodInvocation input, GetNextHandlerDelegate getNext)  {  IMethodReturn methodReturn = getNext()(input, getNext);  return null;  }  public int Order { get; set; }  }  [AttributeUsage(AttributeTargets.Method | AttributeTargets.Interface | AttributeTargets.Class)]  public class ExceptionHandlerAttribute : HandlerAttribute  {  public override ICallHandler CreateHandler(IUnityContainer container)  {  return new ExceptionHandler() { Order = this.Order };  }  } |

EL6.0官方文档：http://files.cnblogs.com/files/wanliwang01/EL5.0.pdf

EL5.0的详细配置，可以参考博主黄聪的系列教程：<http://www.cnblogs.com/huangcong/archive/2010/06/08/1753988.html>

还可以参考仓储大叔的博文：http://www.cnblogs.com/lori/p/4088889.html

参考资料：

1. 蒋金楠. ASP.NET Web API 2框架揭秘[M]. 北京:电子工业出版社, 2014.
2. （美）加洛韦. ASP.NET MVC 5高级编程(第5版)[M]. 北京:清华大学出版社, 2015.