之前介绍了基于SOAP的Web服务，接下来将介绍基于REST的轻量级的Web服务。

REST(Representational State Transfer)与技术无关，代表一种软件架构风格，可以成为ROA面向资源的架构，之前Web服务的架构风格主要是SOAP和XML-RPC。REST从资源的角度来观察整个网络，分布在各处的资源有URI来标识，而客户端通过URI来获取资源的表征，获得这些表征使得应用程序转变了状态。作者是这样解释的，“设计良好的网络应用表现为一系列的网页，这些网页可以看做是虚拟的状态机，用户选择这些链接导致下一网页传输给客户端展现给使用的人，而这正代表了状态的改变”。一般来说，REST是建立在HTTP、URI、XML、JSON等概念的基础之上的，其特点是：一切数据都是资源，所有的资源均可被你唯一标识，采用统一而简单的接口，基于表征的通信，无状态服务调用。

在Web Http编程模型中，包含的主要的类型有：WebHttpBinding, WebHttpBehavior, WebGetAttribute/WebInvokeAttribute和WebServiceHost等。其中值得一提的是WebHttpSecurityMode：None表示请求未使用任何安全性；Transport表示请求使用传输级安全；TransportCredentialOnly表示仅提供基于HTTP客户端身份验证。这儿可以看到由于WebHttpBinding不是基于SOAP协议，因此WS-\*协议簇均无法使用。在消息内容上，可以通过设置相关属性进行，例如RequestFormat=WebMessageFormat.Xml,ResponseFormat=WebMessageFormat.Json，BodyStyle=WebMessgaeBodyStyle.Bare。

对于SOAP协议来说，操作的选择是通过<Action>来决定的，而在这儿时通过UriTemplate属性表示的一个URI模板来决定的，常见的路由例子如接下来的，/filename.{ext}/， /{filename}.jpg/， /{filename}.{ext}/, /{a}.{b}someLiteral{c}{d}/等多种通配符方式，和ASP.NET一样由一个通过注册一个静态的路由表，之后通过路由表来路由请求。

接下来，介绍几个比较有趣的概念，分别是输出缓存、条件获取和更新。前者由于涉及到ASP.NET的CacheProfile的使用，需要使用ASP.NET的兼容模式，不太推荐，可以考虑使用其他的缓存方式进行缓存，比如Redis。后者涉及一个http协议中的请求头ETag，通过对其的判断来决定内容是否已经被更新，比较有实际意思，例子的代码如下。

Interface

|  |
| --- |
| namespace Sory.CoreFramework.Interface  {  [ServiceContract(Namespace = "http://www.sory.com")]  public interface IEmployees  {  [WebGet(UriTemplate="all")]  IEnumerable<Employee> GetAll();  [WebGet(UriTemplate = "{id}")]  Employee Get (String id);  [WebInvoke(UriTemplate = "/", Method = "POST")]  void Create(Employee employee);  [WebInvoke(UriTemplate = "/", Method = "PUT")]  void Update(Employee employee);  [WebInvoke(UriTemplate = "{id}", Method = "DELETE")]  void Delete(string id);  }  } |

Service的实现

|  |
| --- |
| public class EmployeesService : IEmployees  {  private static IList<Employee> \_employees = new List<Employee>  {  new Employee {Id="001", Name="xiongda", Department="xiaowei", Grade="T13"},  new Employee {Id="002", Name="xionger", Department="xiaowei", Grade="T15"}  };  public IEnumerable<Interface.Entities.Employee> GetAll()  {  var hashCode = \_employees.GetHashCode();  WebOperationContext.Current.IncomingRequest.CheckConditionalRetrieve(hashCode);  WebOperationContext.Current.OutgoingResponse.SetETag(hashCode);  return \_employees;  }  public Interface.Entities.Employee Get(string id)  {  var employee = \_employees.FirstOrDefault(e => e.Id == id);  if (null == employee)  {  //WebOperationContext.Current.OutgoingResponse.StatusCode = HttpStatusCode.NotFound;  throw new WebFaultException(HttpStatusCode.NotFound);  }  WebOperationContext.Current.OutgoingResponse.SetETag(employee.GetHashCode());  return employee;  }  public void Create(Interface.Entities.Employee employee)  {  \_employees.Add(employee);  }  public void Update(Interface.Entities.Employee employee)  {  var existing = \_employees.FirstOrDefault(e => e.Id == employee.Id);  if (null == existing)  {  throw new WebFaultException(HttpStatusCode.NotFound);  }  existing.Name += Guid.NewGuid().ToString();  WebOperationContext.Current.IncomingRequest.CheckConditionalUpdate(existing.GetHashCode());  Delete(employee.Id);  \_employees.Add(employee);  WebOperationContext.Current.OutgoingResponse.SetETag(employee.GetHashCode());  }  public void Delete(string id)  {  var employee = this.Get(id);  if (null != employee)  {  \_employees.Remove(employee);  }  }  } |

Server

|  |
| --- |
| public static class ServiceHost  {  public static void Start()  {  using (WebServiceHost host = new WebServiceHost(typeof(EmployeesService)))  {  host.Open();  Console.Read();  }  }  }  配置文件  <system.serviceModel>  <services>  <service name ="Sory.CoreFramework.Service.EmployeesService">  <endpoint address="http://127.0.0.1:3721/employees" binding="webHttpBinding"  contract="Sory.CoreFramework.Interface.IEmployees"/>  </service>  </services>  </system.serviceModel> |

Client

|  |
| --- |
| public class CheckDemo  {  public static void Test()  {  using (ChannelFactory<IEmployees> channelFactory = new ChannelFactory<IEmployees>("employeeService"))  {  var proxy = channelFactory.CreateChannel();  Array.ForEach<Employee>(proxy.GetAll().ToArray(), emp => Console.WriteLine(emp));  }  }  }.  配置文件  <system.serviceModel>  <behaviors>  <endpointBehaviors>  <behavior name="webBehavior">  <webHttp/>  </behavior>  </endpointBehaviors>  </behaviors>  <client>  <endpoint name="employeeService" address="http://127.0.0.1:3721/employees" behaviorConfiguration="webBehavior"  binding="webHttpBinding" contract="Sory.CoreFramework.Interface.IEmployees">  </endpoint>  </client>  </system.serviceModel> |

参考资料：

[1]蒋金楠. WCF全面解析[M]. 上海:电子工业出版社, 2012.