最后一章将进行WCF扩展和新特性的学习，这部分内容有一定深度，有一个基本的了解即可，当需要自定义一个完整的SOA框架时，可以再进行细致的学习和实践。

服务端架构体系的构建主要包含接下来的几个要素：服务描述、终结点分发器选择机制、信道分发器、终结点分发器、分发运行时、分发操作。

ServiceDescription服务描述包含一组终结点和**服务行为**列表，而终结点包含服务地址、绑定和契约信息，契约中则包含操作和**契约行为**，操作信息中包含**操作行为**。

终结点分发器具有两个消息筛选器，一个是地址筛选器，另一个是契约筛选器，均继承自MessageFilter，信道分发器通过他们来选择某个**终结点分发器**来接受请求消息，此外还有一个FilterPriority来决定顺序。在WCF中，终结点ServiceEndpoint和其分发器EndpointDispatcher一一对应。

**信道分发器**ChannelDispatcher包含信息监听器、错误处理器、进行流量控制的服务限流器和信道初始化器，这部分可以扩展的属性包括异常细节信息的传播、手工寻址、最大挂起消息数、同步/异步消息接受和事物控制。

分发运行时DispatcherRuntime是整个WCF服务端运行的核心，其包含很多扩展组件，例如关于安全的ServiceAuthenticationManager、ServiceAuthorizationManager、RoleProvider和AuthorizationPolicy，与服务激活相关的实例提供者、上下文提供者、单例实例上下文和实例上下文初始化器，与并发控制有关的同步上下文，可以对接受的请求消息和回复消息进行相应操作的消息检验器，运行时操作列表和用于选择操作的选择器，自己在项目构建中很多时候可以参考这样的方式。此外，还包括以下的扩展属性，授权、安全审核、事务和会话、未处理的操作、SOAP报头验证、并发控制。

分发操作是在Servicehostk开启后，有OperationDescription转化来的，附加在其上的可扩展组件包括调用上下文初始化器CallContextInitializer，参数检验器ParameterInspector,用于消息序列化和反序列化的消息格式化器DispatchMessageFormatter，用于操作方法的执行的操作调用器OperationInvoker。

客户端架构体系的构建则包含如下几个要素：创建ChannelFactory<TChannel>、客户端运行时、客户端操作、服务代理和服务调用。在通过调用构造函数创建一个ChannelFactory<TChannel>对象后，WCF会根据指定的终结点创建一个ServiceEndpoint对象，其整个结构如下图示。



通过定义四种行为对WCF扩展，这4中行为分别是服务行为、终结点行为、契约行为和操作行为，它们具有相同的4个方法Validate、AddBindingParameters、ApplyDispatchBehavior和ApplyClientBehavior。原书还包含一个关于本地化的扩展例子，需要时可以查阅，注意Properties中的资源文件。

ServiceHost对WCF的扩展一般是通过继承自ServiceHost的自定义类来实现，但需要注意，对Description、分发运行时的定制是无效的，此外可以通过自定义类来集成Unity。

简化开发体验内容包括：默认终结点（标准终结点）、默认绑定配置、默认行为配置和无.svc文件服务激活等。这部分默认的配置就不一一介绍了，唯一需要介绍的就是WCF提供的标准终结点，如下表所示。

|  |  |
| --- | --- |
| 标准终结点 | 描述 |
| mexEndpoint | 用于公开服务元数据的标准终结点 |
| dynamicEndpoint | 使用WS-Discovery在运行时动态查找终结点地址的标准终结点 |
| discoveryEndpoint | 发送/接收发现消息的标准终结点 |
| UdpDiscoveryEndpoint | 通过UDP多播方式发送/接收发现消息的标准终结点 |
| announcementEndpoint | 由服务用于发送公告消息的标准终结点 |
| udpAnnouncementEndpoint | 由服务用于通过UDP绑定发送公告消息的标准终结点 |
| workflowControlEndpoint | 可用于对工作流实例调用控制操作的标准终结点 |
| webHttpEndpoint | 带有自动添加webHttpBehavior行为的WebHttpBinding绑定的标准终结点 |
| webScriptEndpoint | 带有自动添加webScriptEnablingBehavior行为的WebHttpBinding绑定的标准终结点 |

此外，无.svc文件服务的配置如下所示。

|  |
| --- |
| <system.serviceModel>  <serviceHostingEnvironment>  <serviceActivations>  <add service="Sory.CoreFramework.Service.EmployeeService" relativeAddress="EmployeeService.svc"/>  </serviceActivations>  </serviceHostingEnvironment>  </system.serviceModel> |

**路由服务**实际上就是一个WCF服务，当端到端通信而不是点对点通信非常有用，可以将请求转发。RoutingService包括4个服务契约接口，ISimplexDatagramRouter、ISimplexSessionRouter、IRequestReplyRouter和IDuplexSessionRouter，实际上是同ProcessRequest/ProcessMessage两个方法来工作，此外其OperationBehaviorAttribute特性的Impersonation设置为Allowed即允许身份模拟。其路由策略涉及如下几个要素，RoutingBehavior服务行为、消息筛选器和筛选器表。

**服务发现**

之前的内容都是服务提供方和需求方直接沟通场景，那么现在有一个问题，当企业应用很多，需要统一管理服务建立SOA体系时，如何发现服务。这儿就涉及WS-Discovery服务发现的知识了，其包含两种基本的操作模式，Ad-Hoc和Managed。前者客户端在一定的网络范围内以广播的形式发送探测Probe消息搜索目标服务，在该探测消息中，包含相应的搜寻条件，不过感觉管理性很差，不推荐。后者Managed模式通过维护一个所有可用目标服务的中心发现代理（中介者模式），客户端只需要将探测消息发给该中心，即可得到目标服务信息，接下来着重介绍Managed模式。

Managed模式，可用服务都注册在发现代理中，其服务发现过程如下图所示，和想象中的基本一致，服务是去中心的，但服务发现即其目录是中心化的。



接下来通过蒋大师的一个自定义的发现代理服务来彻底了解一个SOA治理中心基本构建方法，其基本步骤包括：创建自定义发现代理服务；寄宿发现代理服务和目标服务；服务的动态调用。

|  |
| --- |
| 服务端代码  public class DiscoveryAsyncResult:IAsyncResult  {  public object AsyncState{ get; private set; }  public WaitHandle AsyncWaitHandle { get; private set; }  public bool CompletedSynchronously { get; private set; }  public bool IsCompleted { get; private set; }  public EndpointDiscoveryMetadata Endpoint { get; private set; }  public DiscoveryAsyncResult(AsyncCallback callback, object asyncState)  {  this.AsyncState = asyncState;  this.AsyncWaitHandle = new ManualResetEvent(true);  this.CompletedSynchronously = this.IsCompleted = true;  if(callback != null)  {  callback(this);  }  }  public DiscoveryAsyncResult(AsyncCallback callback, object asyncState, EndpointDiscoveryMetadata endpoint)  :this(callback, asyncState)  {  this.Endpoint = endpoint;  }  }  [ServiceBehavior(InstanceContextMode = InstanceContextMode.Single, ConcurrencyMode = ConcurrencyMode.Multiple)]  public class DiscoveryProxyService : DiscoveryProxy  {  public IDictionary<EndpointAddress, EndpointDiscoveryMetadata> Endpoints { get; private set; }  public DiscoveryProxyService()  {  this.Endpoints = new Dictionary<EndpointAddress, EndpointDiscoveryMetadata>();  }  /// <summary>  /// Find(Probe)  /// </summary>  protected override IAsyncResult OnBeginFind(FindRequestContext findRequestContext, AsyncCallback callback, object state)  {  var endpoints = from item in this.Endpoints  where findRequestContext.Criteria.IsMatch(item.Value)  select item.Value;  foreach (var endpoint in endpoints)  {  findRequestContext.AddMatchingEndpoint(endpoint);  }  return new DiscoveryAsyncResult(callback, state);  }  protected override void OnEndFind(IAsyncResult result) { }  /// <summary>  /// Resolve  /// </summary>  protected override IAsyncResult OnBeginResolve(ResolveCriteria resolveCriteria, AsyncCallback callback, object state)  {  EndpointDiscoveryMetadata endpoint = null;  if (this.Endpoints.ContainsKey(resolveCriteria.Address))  {  endpoint = this.Endpoints[resolveCriteria.Address];  }  return new DiscoveryAsyncResult(callback, state);  }  protected override EndpointDiscoveryMetadata OnEndResolve(IAsyncResult result)  {  return ((DiscoveryAsyncResult)result).Endpoint;  }  /// <summary>  /// Online  /// </summary>  protected override IAsyncResult OnBeginOnlineAnnouncement(DiscoveryMessageSequence messageSequence, EndpointDiscoveryMetadata endpointDiscoveryMetadata, AsyncCallback callback, object state)  {  this.Endpoints[endpointDiscoveryMetadata.Address] = endpointDiscoveryMetadata;  return new DiscoveryAsyncResult(callback, state);  }  protected override void OnEndOnlineAnnouncement(IAsyncResult result) { }  protected override IAsyncResult OnBeginOfflineAnnouncement(DiscoveryMessageSequence messageSequence, EndpointDiscoveryMetadata endpointDiscoveryMetadata, AsyncCallback callback, object state)  {  if (this.Endpoints.ContainsKey(endpointDiscoveryMetadata.Address))  {  this.Endpoints.Remove(endpointDiscoveryMetadata.Address);  }  return new DiscoveryAsyncResult(callback, state);  }  protected override void OnEndOfflineAnnouncement(IAsyncResult result) { }  }  public static void Start()  {  using (var discoveryProxyService = new ServiceHost(typeof(DiscoveryProxyService)))  using(var employeeService = new ServiceHost(typeof(EmployeesService)))  {  discoveryProxyService.Open();  employeeService.Open();  }  }  服务端配置  <system.serviceModel>  <services>  <service name ="Sory.CoreFramework.Service.DiscoveryProxyService">  <endpoint address="net.tcp://127.0.0.1:8866/discoveryproxy/probe" binding="netTcpBinding" kind="discoveryEndpoint" isSystemEndpoint="false" ></endpoint>  <endpoint address="net.tcp://127.0.0.1:8867/discoveryproxy/announcement" binding="netTcpBinding" kind="announcementEndpoint"></endpoint>  </service>  <service name ="Sory.CoreFramework.Service.EmployeesService" behaviorConfiguration="serviceAnnouncement">  <endpoint address="http://127.0.0.1:3721/employees" binding="ws2007HttpBinding"  contract="Sory.CoreFramework.Interface.IEmployees"/>  </service>  </services>  <behaviors>  <serviceBehaviors>  <behavior>  <serviceDiscovery/>  </behavior>  <behavior name="serviceAnnouncement">  <serviceDiscovery>  <announcementEndpoints>  <endpoint kind="announcementEndpoint" address="net.tcp://127.0.0.1:8867/discoveryproxy/announcement" binding="netTcpBinding"></endpoint>  </announcementEndpoints>  </serviceDiscovery>  </behavior>  </serviceBehaviors>  </behaviors>  </system.serviceModel>  客户端代码  using (ChannelFactory<IEmployees> channelFactory = new ChannelFactory<IEmployees>("employeeService"))  {  var proxy = channelFactory.CreateChannel();  Array.ForEach<Employee>(proxy.GetAll().ToArray(), emp => Console.WriteLine(emp));  }  客户端配置  <system.serviceModel>  <client>  <endpoint name="employeeService" kind="dynamicEndpoint" endpointConfiguration="unicastEndpoint"  binding="ws2007HttpBinding" contract="Sory.CoreFramework.Interface.IEmployees">  </endpoint>  </client>  <standardEndpoints>  <dynamicEndpoint>  <standardEndpoint name="unicastEndpoint">  <discoveryClientSettings>  <endpoint kind="discoveryEndpoint" address="net.tcp://127.0.0.1:8888/discoveryproxy/probe" binding="netTcpBinding"></endpoint>  </discoveryClientSettings>  </standardEndpoint>  </dynamicEndpoint>  </standardEndpoints>  </system.serviceModel> |

终于完成WCF最后一块拼图，非常的开心，虽然赶脚只掌握了3成左右，但也基本足够了，需要时再回顾学习了。

参考资料：

[1]蒋金楠. WCF全面解析[M]. 上海:电子工业出版社, 2012.