

基于 WCF 的 SOA 框架设计与实现

郭丹

(北京邮电大学计算机科学与技术学院, 北京 100876)

摘要: 随着 SOA 标准的成熟以及支持 SOA 实现技术的不断发展, SOA 在各企业的信息化建设及其部门业务整合上具有广泛的应用。WCF 作为 .NET 近几年推出的分布式开发技术, 融合了之前 .Net Remoting, Asp.net WebService 等相关技术, 无疑是 .NET 平台实现 SOA 框架的制胜利器。本文首先简要介绍了 SOA 的概念以及发展现状, 探讨了目前 SOA 的实现方式, 并分析在 JAVA 平台与 .NET 平台实现 SOA 的相关技术。在此基础上, 介绍了 WCF 技术相关知识, 并提出一种基于 WCF 的 SOA 框架, 最后给出其设计和实现。

关键词: SOA; WCF; .NET; 框架

中图分类号: TP311.1

Design and Implementation of an SOA Architecture Based on WCF

Guo Dan

(Computer Science And Technology School, Beijing University of Post And Telecommunication, Beijing 100876)

Abstract: With the maturity of SOA standards and the continuous development of SOA implementation technologies, SOA in the enterprise and departmental business information integration has a wide application. WCF, introduced in .NET platform in recent years, is a distributed development technology, integrated of the previous .Net Remoting, Asp.net WebService, and other related technologies. There is no doubt that WCF is the winning weapon for developing .NET platform SOA framework. In this paper, we firstly introduce the concept of SOA and the development status of the current SOA implementations, and then analyze SOA-related technologies in JAVA platform and .NET platform, describe the WCF technology, and finally propose a SOA framework based on WCF and give its design and implementation.

Key words: SOA; WCF; .NET; Framework

0 引言

距离 1996 年 Gartner 提出 SOA 到现在已十年有余^[1], SOA 已从一种单纯的概念模型, 转化为多种形态的技术实现。越来越多的企业和组织, 在面临复杂的企业应用整合或业务流程开发时选择使用 SOA 框架实现。对于跨语言, 跨平台的系统应用开发, SOA 无疑是一种首选的解决方案。本文讨论的 SOA 架构实现便是在这样的需求中诞生: 企业内部积累的业务逻辑封装都是基于 .net 平台, 而业务发展和扩张, 使得现在使用多语言多平台进行应用开发, 这些应用都不同程度地依赖基础业务逻辑。

1 SOA 及相关技术

1.1 SOA 简介

SOA 是英文词语 "Service Oriented Architecture" 的缩写, 中文有多种翻译, 如 "面向服务的体系结构"、"以服务为中心的体系结构" 和 "面向服务的架构", 其中 "面向服务的架构" 比较常见。SOA 有很多定义, 但基本上可以分为两类: 一类认为 SOA 主要是一种架构风格; 另

作者简介: 郭丹, 网络信息处理. E-mail: frankccm@hotmail.com

一类认为 SOA 是包含运行环境、编程模型、架构风格和相关方法论等在内的一整套新的分布式软件系统构造方法和环境，涵盖服务的整个生命周期：建模-开发-整合-部署-运行-管理^[2]。

简单地说，SOA 是整合异构数据、逻辑乃至业务流程的一种模式。它不是具体的软件产品技术，而是系统架构的一种方法和思想^[3]。一般情况下我们认为，SOA 是一种组件模型，将应用程序的不同功能单元（称为服务，Service）通过这些服务之间定义的良好接口 (Interface)和契约(Contract)联系起来。而接口应该采用中立的方式进行定义，独立于实现服务的硬件平台、操作系统和编程语言。从而使得构建在各种系统中的服务，可以以一种统一和通用的方式进行交互。

1.2 SOA 发展现状

从 2005 年开始，SOA 推广和普及工作开始加速。不仅专家学者，几乎所有关心软件行业发展的人士都开始把目光投向 SOA。各大厂商也逐渐放弃成见，通过建立厂商间的协作组织共同努力制定中立的 SOA 标准。时至今日，SOA 已在许多大厂商产品的产品线上占有重要地位，而诸多企业的软件计划，也都围绕 SOA 进行。世界各地、各行业的一些企业已经实施了一些项目，并总结出成功的经验和失败的教训，形成丰富的最佳实践指导。业务咨询商与客户一起合作，制定出规范的 SOA 监管架构与管理组织模型，成熟的理论被采纳到 SOA 实践中，而且不断地完善 SOA 领域的新技术。

产业界已经开发出相对成熟和可用的相关软件平台，并形成完整的体系。国内外各大厂商也在近几年也相应的推出 SOA 解决方案，例如 IBM 的 Smart SOA，金蝶的 Apusic SOA，Oracle 的 SOA Suit 等。目前大多数厂商的 SOA 解决方案都是基于 J2EE，虽然 .NET 平台还没有商用的 SOA 成套的解决方案，但微软对 SOA 的支持从未停止过。从 .NET framework 最初的 ASP.NET Webservice 到现在的 WCF 和 WPF，.NET 平台涉猎 SOA 领域的优势可见一斑。

SOA 发展至今，相关的技术标准已经成熟并形成完整的体系，工业界已经广泛地采纳这些标准。SOA 涉及到的技术标准如图 1.2-1 所示。

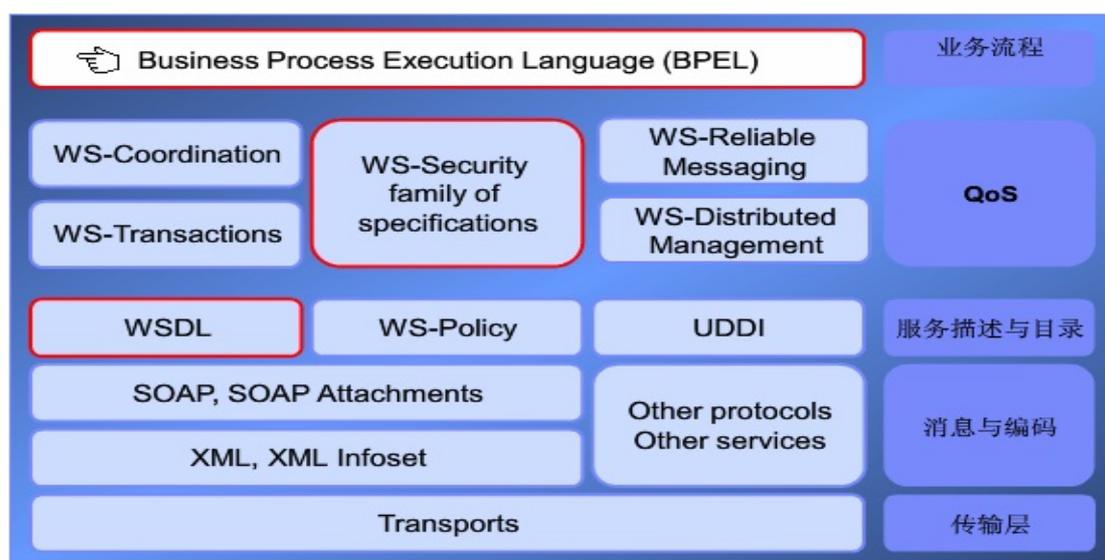


图 1.2-1 SOA 相关标准

产业界已经开发出相对成熟和可用的相关软件平台，并形成完整的体系，如 JAVA 的

JBI, .NET 的 WCF 等。世界各地、各行业的一些企业已经实施了一些项目，并总结出成功的经验和失败的教训，形成丰富的最佳实践指导。业务咨询商与客户一起合作，制定出规范的 SOA 监管架构与管理组织模型，成熟的理论被采纳到 SOA 实践中，而且不断地完善 SOA 领域的新技术。行业内 SOA 的参考模型如图 1.2-2 所示。

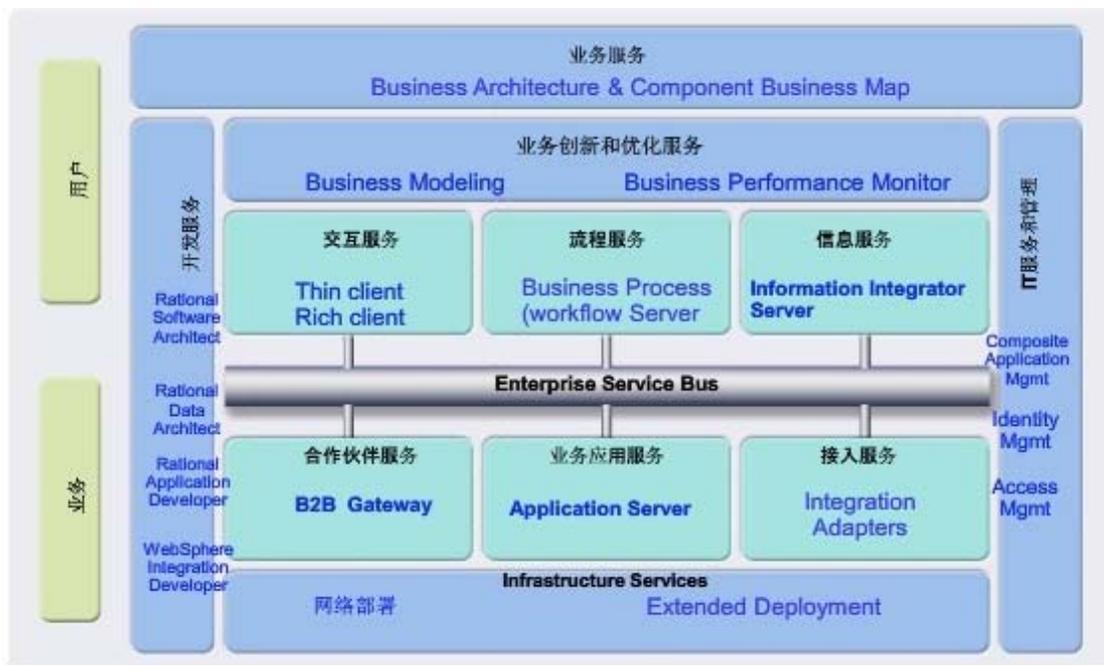


图 1.2-2 SOA 参考模型

SOA 最常见的实现方式为 Web service，所以经常有人将 Web service 与 SOA 混为一谈。实际上 SOA 有多种实现方式，它只是一种思想，Web Service 不过是其中一种实现方法。

除了 Web Service，还有另外两种常用实现方式，分别为：企业级别的利用已有的消息中间件实现 SOA 框架和部门级别的使用 REST 和 Web2.0 的实现^[4]。

下面我们要讨论的主要是 JAVA 平台和 .NET 平台下 Web service 或消息中间件的 SOA 实方式。

1.3 SOA 实现相关技术

JAVA 平台下实现 SOA 相关的技术有：JMS(Java Message Service)，EJB (sun 的服务器端组件模型)，JCA(J2EE Connector Architecture，也缩写为，J2C， J2CA)，RMI (Remote Method Invocation，远程方法调用) 以及 JBI (Java 业务集成) 模型等。

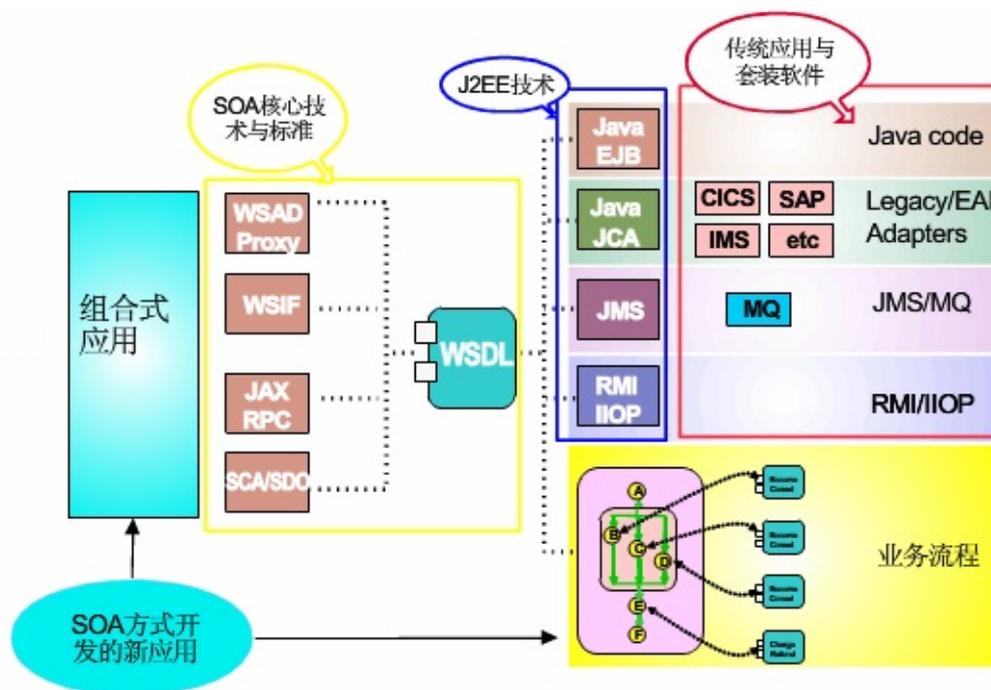


图 1.3 实现 SOA 相关的 JAVA 技术

JMS(Java Message Service)是访问企业消息系统的标准 API，定义了 Java 中访问消息中间件的接口，但 JMS 只是接口，并没有给予实现，实现 JMS 接口的消息中间件称为 JMS 提供者(JMS Provider)。EJB 是 sun 的服务器端组件模型，最大的用处是部署分布式应用程序当然,还有许多方式可以实现分布式应用。用 EJB 技术部署的分布式系统可以不限于特定的平台。JCA(J2EE Connector Architecture, 也缩写为, J2C, J2CA), 是J2EE平台上连接传统系统的一个技术规范。JCA 是 J2EE 体系架构的一部分，为开发人员提供了一套连接各种异类的企业信息系统(EIS, 包括 ERP、SCM、CRM 等,这些系统可能是历史遗留下来非 JAVA 语言编写的系统)的体系架构。RMI (Remote Method Invocation, 远程方法调用)是开发百分之百纯Java 的网络分布式应用系统的核心解决方案之一。其实它可以被看作是 RPC 的 Java 版本。但是传统 RPC 并不能很好地应用于分布式对象系统。而 Java RMI 则支持存储于不同地址空间的程序级对象之间彼此进行通信，实现远程对象之间的无缝远程调用。

JB1 的提出是基于面向服务体系提倡的方法和原则，为了解决 EAI 和 B2B 若干问题的 Java 标准。商业和开源界都欢迎 JB1 成为他们 ESB 产品的集成标准。JB1 定义了基于插件方式的架构，以便服务能融入“JB1 运行时”环境。JB1 提供了详细的接口，使服务能与“JB1 运行时”环境交互。这些服务要为“JB1 运行时”环境暴露接口，以便“JB1 运行时”环境为服务路由消息。“JB1 运行时”环境在部署在 SOA 环境中的服务间扮演仲裁者的角色^[5]。

利用上述技术结合，可以在 JAVA 平台下开发出灵活而强大的 SOA 架构。

微软对 SOA 的支撑技术有 ASP.net Web Service, WCF 等等。

SOA 的实现方式之一是通过 HTTP 的 Web Service 来实现。在 Microsoft.NET 的早期版本中实现基于 HTTP 的 Web 服务有两种根本不同的方法。第一种也是较低级的一种技术是编写一个定制的 IHttpHandler 类并把它嵌入到 HTTP 管道中。另一种更高效的方法是使用 Microsoft ASP.NET 的 WebMethods 框架。ASP.NET 为 .asmx 终结点(叫作 WebServiceHandler) 装载了一个专门的 IHttpHandler 类，它为你的需要提供了 XML、XSD、SOAP 和 WSDL 的

功能性样板。因为 WebMethod 框架使你从底层 XML 技术的复杂性中解脱出来，可以将精力集中到一些紧要的业务问题。

WCF (Windows Communication Foundation) 是由微软发展的一组数据通信的应用程序开发接口，它是 .NET 框架的一部分，由 .NET Framework 3.0 开始引入。在 .NET Framework 2.0 以及前版本中，微软发展了 Web Service (SOAP with HTTP communication)，.NET Remoting (TCP/HTTP/Pipeline communication) 以及基础的 Winsock 等通信支持，由于各个通信方法的设计方法不同，而且彼此之间也有相互的重叠性（例如 .NET Remoting 可以开发 SOAP, HTTP 通信），对于开发人员来说，不同的选择会有不同的程序设计模型，而且必须要重新学习，让开发人员在使用中有许多不便。同时，面向服务架构 (Service-Oriented Architecture) 也开始盛行于软件工业中，因此微软重新查看了这些通信方法，并设计了一个统一的程序开发模型，对于数据通信提供了最基本最有弹性的支持，这就是 Windows Communication Foundation。WCF 无疑是 .NET 平台实现 SOA 的利器。

2 WCF 技术

2.1 WCF 技术简介

从功能的角度来看，WCF 完全可以看作是 ASMX，.Net Remoting，Enterprise Service，WSE，MSMQ 等技术的并集。（注：这种说法仅仅是从功能的角度。事实上 WCF 远非简单的并集这样简单，它是真正面向服务的产品，它已经改变了通常的开发模式。）因此，对于上述汽车预约服务系统的例子，利用 WCF，就可以解决包括安全、可信赖、互操作、跨平台通信等等需求。开发者再也不用去分别了解 .Net Remoting，ASMX 等各种技术了^[6]。

2.2 WCF 开发基础

2.2.1 地址

WCF 的每一个服务都具有一个唯一的地址 (Address)。地址服务位置与传输协议，或者是用于服务通信的传输样式。

服务位置包括目标机器名、站点或网络、通信端口、管道或队列，以及一个可选的特定路径或者 URI。

WCF 支持的传输样式：HTTP，TCP，Peer network (对等网)，IPC (基于命名管道的内部进程通信)，MSMQ。

地址格式一般如下：[传输协议]://[机器名称或域名][:可选端口]/[可选的 URI]，例如：

http://localhost:8001/myservice

net.tcp://localhost:8001/myservice

2.2.2 契约

WCF 的所有服务都会公开为契约 (Contract)。契约与平台无关，是描述服务功能的标准方式。分为：服务契约，数据契约，操作契约，错误契约和消息契约。

服务契约以特性的方式 (Attribute)，可以应用到接口的定义或者类上。

关于契约的定义和使用，在下面的例子中可以看到。

2.2.3 托管 (HOST)

WCF 服务类必须托管，也称宿主。它可以托管到 IIS 服务，自托管到 Windows 进程或

者托管到 Windows，在 Vista 系统中，可以托管到 WAS（Windows 激活服务）。

2.2.4 绑定

服务之前的通信方式是多种多样的，有多种可能的通信模式。包括同步，异步，即时消息或队列消息等，消息的传输协议和编码格式，以及安全机制都有多种组合方式。

WCF 引入了绑定（Binding）技术将这些通信特征组合在一起。一个绑定封装了诸如传输协议、消息编码、通信模式、可靠性、安全性、事务传播以及互操作性等相关选项的集合，使得它们保持一致。WCF 定义了 9 种标准绑定：BasicHttpBinding，NetTcpBinding，NetPeerTcpBinding，NetNamedPipeBinding，WSHttpBinding，WSFederationHttpBinding，WSDualHttpBinding，NetMsmqBinding，MsmqIntergrationBinding。开发者也可以根据自己需要，编写自己的定制绑定。

2.2.5 终结点

服务与地址、绑定以及契约有关，每个终结点都包含这三个元素，而宿主负责公开终结点。终结点相当于服务的接口，每个服务至少必须公开一个业务终结点，每个终结点有且只能有一个契约。

2.2.6 元数据交换

服务有两种方式可以发布自己的元数据。一种是基于 HTTP-Get 协议提供元数据，需要显式地添加服务行为(Behavior)；另一种是以专门的终结点方式。

2.2.7 客户端

在 WCF 中有两种不同的方法可以用于创建客户端服务对象,他们分别为:

1. 代理构造法

可以使用 Visual Studio 2005 或者 SvcUtil 工具来导入服务元数据并且生成一个代理。如果服务是自宿主的，那么首先要启动该服务，然后从客户端工程的上下文菜单中选择"Add Service Reference..."。客户端需要知道服务所在地并使用与它的服务相同的绑定；当然，也要以代理的形式导入服务合同。实质上，这与在服务的端点处捕获的信息完全一致。为了反映这一信息，客户端配置文件包含关于目标端点的信息并且甚至使用与宿主相同的模式。

2. 通道工厂法

可以直接使用通道来调用服务上的操作而甚至不必依赖于一个 SvcUtil 生成的代理。ChannelFactory<T>类能够使你任意地创建一个代理。你需要提供给它的构造器端点信息-或者是来自配置文件的端点名，或者是绑定和地址对象，或者是一个端点对象。然后，使用 CreateChannel()方法以获得一个到代理的参考（顶级通道）并且使用它的方法。

3 基于 WCF 的 SOA 架构设计与实现

SOA 框架设计目标以统一性，简洁性，透明性为基础，力求达到统一服务调用，统一服务发布，统一服务配置的功能。宿主进程为一个 Windows 服务，之所以没有让 SOA 服务宿主到 IIS 或者其他进程，是为了不占用 IIS 更多资源，并且提供更好的性能，当然也有安全性方面的考虑。

SOA 框架的设计目标是，提供类插件式服务架构。添加或修改服务接口和方法时，只需要修改相应的 Class 和 Interface，并将编译好的 DLL 部署到框架系统内，框架可以自动生成相应的服务终结点供客户端调用。如图所示：

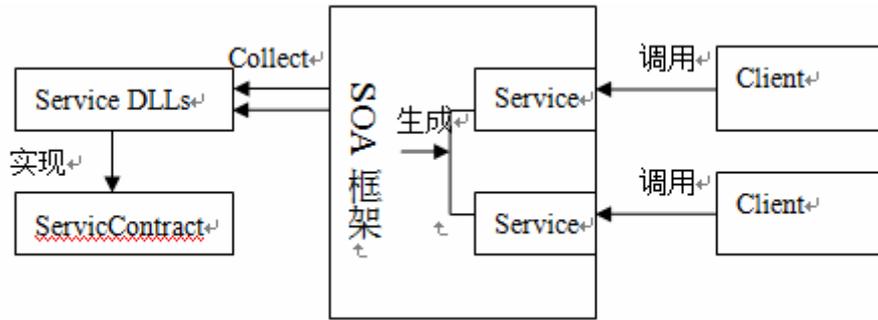


图 3-1 插件式 SOA 框架

下面是各模块的具体实现，涉及到代码部分，只给出类的概要信息。

3.1 SOA 服务端模块及实现

服务器端主要包括服务宿主模块，服务通信模块，服务生成模块，服务调用契约模块，服务扩展模块等。

服务器端模块关系图：

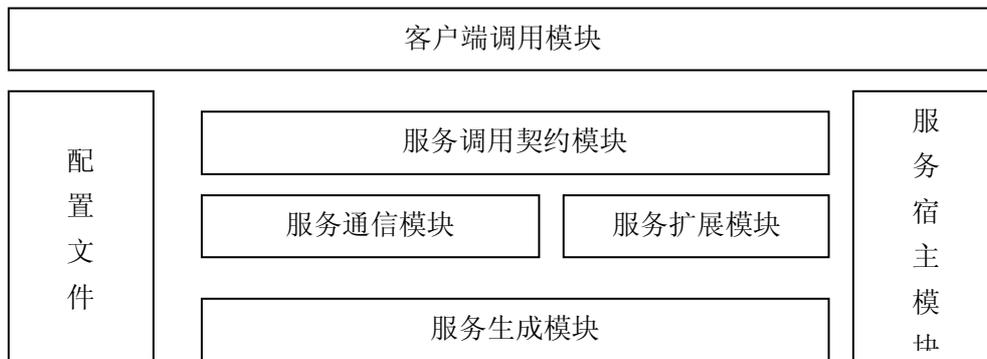


图 3-2 SOA 服务模块层次图

3.1.1 服务调用契约模块

负责数据的交换格式及服务的接口提供。此模块负责要交换的数据契约的定义，及待公布的服务契约的定义，在整个 SOA 框架中，每次新增加服务接口，需要在此模块中增加接口返回数据的定义，及接口的契约和接口功能的定义，并编译此模块，部署到服务器端 LocalRepository 即可。这个模块是 SOA 框架中，新增或修改服务时唯一需要变动的模块（包括配置文件）。

以增加一个返回用户基本信息的服务接口为例，首先需要定义数据契约：

```
[DataContract]
public class CurrentUser : BaseDataContract
{
    private int _Id;
    [DataMember]
    public int Id { get { return _Id; } set { _Id = value; } }
    private string _Name;
    [DataMember]
    public string Name { get { return _Name; } set { _Name = value; } }
    .....
}
```

```
}
```

然后定义服务契约：

```
[OperationContract(Action = "GetUser", IsOneWay = false)]  
[WebGet(UriTemplate = "json/getuser?userId={userId}", ResponseFormat =  
WebMessageFormat.Json)]  
    CurrentUser GetUser(int userId);
```

实现服务契约：

```
public CurrentUser GetUser(int userId)  
{  
    try {  
        if (userId <= 0)  
            return null;  
        .....  
        return currentUser;  
    }  
    catch (Exception ex)  
    {  
        LogException(ex, " ApiWcf.Users");  
        return null;  
    }  
}
```

将服务契约定义在 `public partial class ApiWcf` 类下，编译后部署到服务器中相应目录下，以便自动生成服务模块能够找到该服务接口，并发布服务。

3.1.2 服务通信模块

该模块主要定义绑定模式，服务行为模式，自定义序列化反序列化格式，以及自定义消息传输通道等。模块提供了很多自定义扩展，但最常用的是自定义绑定和服务行为。

自定义绑定：

```
public class CustomEncodingBindingSection : BindingElementExtensionElement  
{  
    private const string EncoderTypeConfigKey = "encoderType";  
    [ConfigurationProperty(EncoderTypeConfigKey)]  
    public string EncoderType {}  
    public override Type BindingElementType {}  
    protected override BindingElement CreateBindingElement() {}  
    public override void ApplyConfiguration(BindingElement bindingElement) {}  
}
```

自定义服务行为：

```
public class CustomWebBehavior : IEndpointBehavior  
{  
    #region IEndpointBehavior Members  
    public void AddBindingParameters(ServiceEndpoint endpoint, BindingParameterCollection  
bindingParameters) {}  
    public void ApplyClientBehavior(ServiceEndpoint endpoint, ClientRuntime  
clientRuntime) {}  
    public void ApplyDispatchBehavior(ServiceEndpoint endpoint, EndpointDispatcher
```

```
endpointDispatcher) {}  
    public void Validate(ServiceEndpoint endpoint) {}  
    #endregion  
}
```

配置:

```
<extensions>  
  <behaviorExtensions>  
    <add name="customWebBehavior" type="PlatformServices.Communication.CustomWebSection,  
PlatformServices.Communication, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />  
    <add name="msgFormatterInserter"  
type="PlatformServices.Communication.MessageFormatterInserter,  
PlatformServices.Communication, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />  
  </behaviorExtensions>  
  
  <bindingElementExtensions>  
    <add name="customMessageEncoding"  
type="PlatformServices.Communication.CustomEncodingBindingSection,  
PlatformServices.Communication, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />  
    <add name="customTransport"  
type="PlatformServices.Communication.CustomTransportBindingSection,  
PlatformServices.Communication, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" />  
  </bindingElementExtensions>  
</extensions>
```

3.1.3 服务扩展模块

该模块目前主要进行安全验证，以后可以增加其他方面的扩展，例如统计等。

安全验证主要根据客户端的 IP 查看是否在允许调用服务的范围内：

```
public class MsgCheckerInspector : IDispatchMessageInspector, IClientMessageInspector  
{  
    #region IStubMessageInspector  
    public object AfterReceiveRequest(ref Message request, IClientChannel channel,  
InstanceContext instanceContext) {}  
    public void BeforeSendReply(ref Message reply, object correlationState) {}  
    #endregion  
    #region IProxyMessageInspector  
    public void AfterReceiveReply(ref Message reply, object correlationState) {}  
    public object BeforeSendRequest(ref Message request, IClientChannel channel) {}  
    #endregion  
}
```

IP 安全配置:

```
<IPSecurityConfig>  
  <IPSet Math="localhost;10.129.60.0.11">  
    <AllowMethodList>  
      getuser;getfriendlist;  
    </AllowMethodList>  
  </IPSet>  
</IPSecurityConfig>
```

Behavior 配置:

```
<extensions>
  <behaviorExtensions>
    <add name="msgChecker"
type="PlatformServices.Extensions.Behaviors.MsgCheckerBehaviorSection,
PlatformServices.Extensions, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"/>
  </behaviorExtensions>
</extensions>
<behaviors>
  <serviceBehaviors>
    <behavior name="PlatformServiceBehaviors">
      <msgChecker IPFilter="true" /> //可以根据需求, 启动或取消IP安全检查
    </behavior>
  </serviceBehaviors>
</behaviors>
```

3.1.4 服务生成模块

此模块是整个框架的核心模块, 该模块从宿主进程启动开始, 一直监视 LocalRepository 服务目录, 如果有文件发生了变化, 自动重新加载服务所依赖的所有 DLL, 生成新的服务。生成服务流程图:

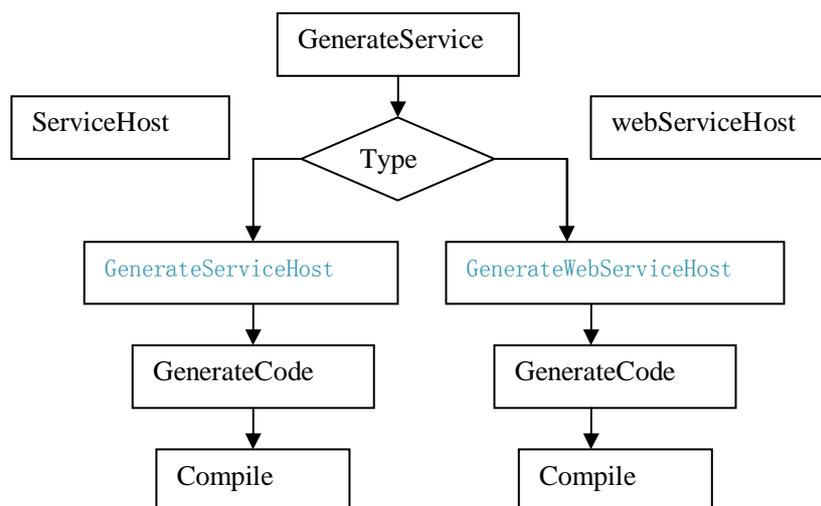


图 3-3 服务生成模块流程图

根据配置文件中指定的类型生成服务, 如果在配置文件中指定为 generateWcfServiceType="ServiceHost"则调用 GenerateServiceHost, 主要通过 TCP 进行消息通信, 并使用客户端进行调用。指定为 WebServiceHost, 则生成 WebService 服务, 主要通过 Http 进行消息通信, 并提供 Get 和 Post 调用方式。

3.1.5 服务宿主模块

服务宿主模块是一个 Windows 服务, 该服务承载了配置文件指定的几种服务的 Host, 包括上面提到的 ServiceHost 和 WebServiceHost。

3.1.6 配置文件

以上各模块描述时已提到相关的配置信息。实际上框架最主要的配置文件有四个。App.config, 提供框架基本配置信息, 及所包含的服务, 最关键的是<LocalRepository>配置部分, 它包含了整个框架要提供的服务信息。

```
<LocalRepository enable="true" endpointName="boot">
  <HostMetadata hostName="API_PlatformServices" assemblyNames="PlatformServices.Core">
    <Services>
      <add name="BasicHttp.ApiWcf"
        serviceType="PlatformServices.Library.ApiWcf"
        appDomainHostName="PlatformServices.BasicHttpAPI"
        serviceNamespace="__basichttp"
        generateWcfServiceType="ServiceHost"
        config="LocalRepository\BasicHttpApiWcf\BasicHttp.ApiWcf.config"
        assemblyFolderName="LocalRepository\BasicHttpApiWcf"
        assemblyNames="PlatformServices.Library"
      />
      <add name="NetTcp.ApiWcf"
        serviceType="PlatformServices.Library.ApiWcf"
        appDomainHostName="PlatformServices.NetTcpAPI"
        serviceNamespace="__nettcp"
        generateWcfServiceType="ServiceHost"
        config="LocalRepository\NetTcpApiWcf\NetTcp.ApiWcf.config"
        assemblyFolderName="LocalRepository\NetTcpApiWcf"
        assemblyNames="PlatformServices.Library"
      />
      <add name="WebHttp.ApiWcf"
        serviceType="PlatformServices.Library.ApiWcf"
        appDomainHostName="PlatformServices.WebHttpAPI"
        serviceNamespace="__webhttp"
        generateWcfServiceType="WebServiceHost"
        config="LocalRepository\WebHttpApiWcf\WebHttp.ApiWcf.config"
        assemblyFolderName="LocalRepository\WebHttpApiWcf"
        assemblyNames="PlatformServices.Library"
      />
    </Services>
  </HostMetadata>
</LocalRepository>
```

如上所示, 目前框架支持三种服务, 前两种服务只能在.net 平台用客户端进行调用。后一种服务可以进行跨平台的调用。服务中 config 指定了配置文件所在的位置及名称。我们所说的四个最主要的配置文件, 除了 app.config, 剩下的三个就是各个服务所对应的配置文件, 如 Webhttp.ApiWcf 对应的配置文件 Webhttp.ApiWcf.config。它定义了服务的行为模式, 绑定模式, 服务的终结点等。其中终结点包含了服务的基本地址:

```
<service behaviorConfiguration="PlatformServiceBehaviors"
  name="__webhttp.ApiWcf">
  <endpoint address="" binding="jsonpBinding"
bindingConfiguration="JsonpBindingConfiguration"
```

```
behaviorConfiguration="JsonpBindingBehavior"  
  contract="__webhttp.ApiWcf" />  
  <baseAddresses>  
    <add baseAddress="http://10.60.0.13:8002" />  
  </baseAddresses>  
</host>  
</service>
```

3.2 SOA 框架处理流程

SOA 框架的处理流程如图所示：

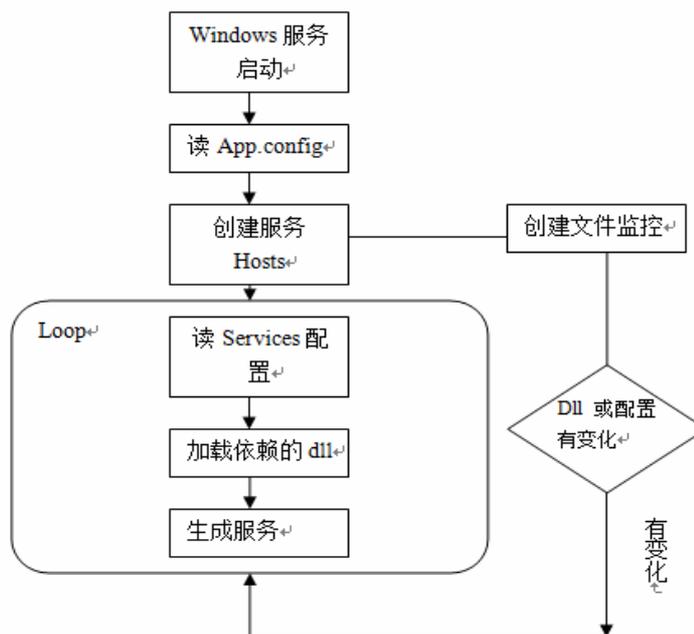


图 3-4 SOA 服务框架流程图

3.3 SOA 客户端调用

SOA 客户端调用分为两种方式：

对于 WebHttp.ApiWcf 服务，可以直接通过 URL 进行 Get 请求或 Post 请求调用，如 `http://10.60.0.13:8002/getuser?userId=1300000001`。另外，还可以通过框架提供的客户端进行调用。

客户端模块主要分为两部分，一部分为调用接口 Provider，一部分为客户端生成器 ClientGenerator。在上一节已经提到 WCF 客户端的编程模式，是通过调用客户端的代理对象来完成的。首先客户端需要生成服务的代理对象，这一过程在我们的框架中由 ClientGenerator 在加载客户端配置文件时自动完成。

客户端调用服务很简单，只需要获得 ClientProvider 的实例，并知道对应服务接口的方法规格就可以了，还是举获得用户信息的例子：`PlatformServiceProvider.Instance.GetUser(1234567890)`。

4 结论

本文在分析 SOA 发展现状及实现技术的基础上，介绍了 .NET 平台下实现 SOA 的技术 WCF，并以 WCF 为基础，设计并实现了一个类插件式的 SOA 框架，并给出了客端调用方

式。

[参考文献] (References)

- [1] 张道海.基于 SOA 的企业系统架构研究[J]. 中国管理信息化, 2008, 24.
- [2] 毛新生.《SOA 原理·方法·实践》[M].电子工业出版社. 2007 年 12 月
- [3] 陈东涛.SOA 在企业信息系统集成中的应用研究[J]. 管理观察,总第 379 期.
- [4] URL: <http://news.csdn.net/n/20070717/106450.html>
- [5] URL: <http://edu.itbulo.com/200710/120241.htm>
- [6] Juval Lowy 著.张逸 徐宁译.《WCF 服务编程》[M].机械工业出版社.2008.1