

附录 B 在 Spring 中开发 Web Service



Spring 支持各种远程访问技术，包括 RMI、JAXRPC、Hessian、Burlap、XFire 以及 Spring 自身提供的 HTTP Invoker。本章将重点介绍基于 Spring-WS 应用，读者可以从 Spring 的帮助手册中获取其他远程技术的知识。之所以这样安排，一方面是由于篇幅所限，不可能面面俱到，与其泛泛而谈，不如选择重点深入剖析；另一方面是由于相对于其他远程技术，Web Service 比较复杂，涉及的内容很多，对于实际的应用来说，Web Service 是使用最广、功能最全、标准化最高的远程技术。

Spring-WS 采用契约优先的 Web 服务设计理念，以文档驱动来构建 Web 服务，相对于传统以代码驱动的 Web 服务，Spring-WS 拥有更多的优势，比如可以做到服务契约与内部服务接口的松耦合，可以快速响应业务需求变化，实现多版本数据契约的共存，使用 XML/XSD 定义服务可以解决不同语言数据类型的互通性问题。

本章主要内容：

- ◆ Web Service 基础知识
- ◆ Spring-WS 简介
- ◆ 构建基于文档驱动的 Web 服务
- ◆ 编写客户端调用代码
- ◆ Web Service 的测试
- ◆ 使用 WS-Security 安全策略

本章亮点：

- ◆ 简明扼要地介绍了 Spring-WS 体系结构
- ◆ 详细讲解基于文档驱动的 Web 服务构建过程

B.1 Web Service 简介

Web Service 是建立可互操作的分布式应用程序的技术平台，它提供了一系列标准，定义了应用程序如何在 Web 上进行互操作的规范。开发者可以使用自己喜欢的编程语言，在各种不同的操作系统平台上编写 Web Service 应用。

B.1.1 Web Service 相关概念

Web Service 是单一的、构件化的程序功能实体，能够通过网络，特别是万维网来描述、发布、定位及调用。Web Service 的体系结构描述了三个角色（服务提供者、服务请求者和服务中介者）及三个操作（发布、查找和绑定）。

服务提供者通过在服务中介处注册并发布服务，服务请求者通过查找服务中介发现并定位到服务，服务请求者绑定服务提供者并使用特定的服务。

在 Web Service 的世界里，这三个操作分别通过对应的技术规范完成，特别是 SOAP、WSDL 和 UDDI 三剑客。发布服务使用 UDDI（Universal Description, Discovery and Integration：统一描述、发现和集成）；查找服务使用 UDDI 和 WSDL（Web Services Description Language：Web Service 描述语言）；而绑定服务使用 WSDL 和 SOAP（Simple Object Access Protocol：简单对象访问协议），如图 B-1 所示。

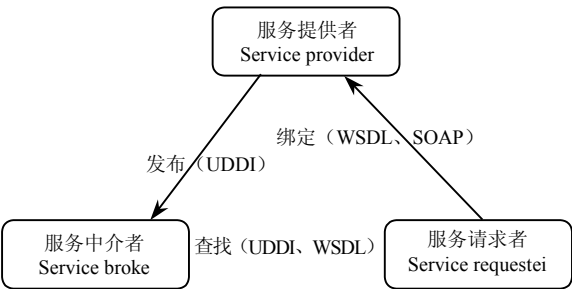


图 B-1 Web Service 三个关键角色的关系

在这三个操作中，绑定操作是最重要的，它代表了服务的最终使用，这也是最终发生交互操作的地方。正是由于服务提供者和服务请求者对 SOAP 规范的全力支持才解决了这些问题，并实现了无缝交互操作。

当开发人员开发新的应用时，可以登录服务中介者所提供 UDDI 搜索引擎的 Web 界面，并在 UDDI 注册表中查找自己所需要的 Web Service，通过 UDDI 注册表中的连接找到 Web Service 的调用细节，具体调用细节采用 WSDL 描述。开发人员可以使用开发工具或通过手工方式还原调用细节，然后在自己的应用程序中根据 WSDL 的描述开发 Web Service 的客户端调用程序——这样开发出的应用就可以通过 SOAP 调用指定的 Web Service 了。如果 Web Service 应用仅在特定组织机构中使用，服务调用者完全可以通过其他途径获取 Web Service 的 WSDL，这样就无须服务发布中介了。



轻松一刻

当某个事物由三个重要部分组成时，我们常常称之为“三剑客”，如我们说 Dreamweaver、Firework 和 Flash 是网页制作的“三剑客”；SOAP、WSDL 和 UUID 是 Web Service 的“三剑客”。“三剑客”源于大仲马的《三剑客》一书，本书通过主人公达达尼昂及三个火枪手的种种奇遇，描写了上层封建统治阶级内部的矛盾和斗争。小说主人公达达尼昂是一个贵族子弟，遵从父命效力于国王路易十三的火枪队队长，与其手下的三个火枪手结成好友。达达尼昂钟情于王后的贴身侍女波纳雪夫人，愿为她而替王后赴汤蹈火。王后与英国首相白金汉公爵的私情为红衣主教黎世留觉察。黎世留得知王后将国王赠给她的那串金刚钻坠子送给了白金汉公爵后，让国王举行盛大舞会，要王后戴上坠子出席，以此让王后丢脸，来发泄他们之间争权夺利的宿怨。达达尼昂与三个火枪手凭着自己的勇气和智谋，闯过黎世留设下的重重阻碍，从英国取回坠子，在舞会开始的前一刻将坠子送到王后手中。



B.1.2 SOAP——简单对象传输协议

SOAP 是一种基于 XML 的不依赖传输层的应用层协议，用于在应用程序之间以对象的形式交换数据。SOAP 所使用的传输协议，可以是 HTTP、SMTP、POP3、JMS 等协议，还可以是为一些应用而专门设计的特殊通信协议。但最常使用的还是 HTTP，这是因为任何可以使用 Web 浏览器的机器都支持 HTTP，同时，当前许多防火墙也配置为只允许 HTTP 连接。

SOAP 以 XML 形式提供了一个简单、轻量级的用于在分布应用环境下交换对象信息的协议。SOAP 本身并没有定义任何应用程序语义，如编程模型或特定语义的实现，这使得 SOAP 能够以消息的形式传递到各种远程系统中。

SOAP 包括以下三个部分。

- SOAP 封装结构：定义了一个整体框架，以表示消息中包含什么内容、谁来处理这些内容及这些内容是可选还是必需的。
- SOAP 编码规则：定义了用以交换应用程序特定数据类型的一系列机制。
- SOAP RPC 表示：定义了一个用来表示远程过程调用和应答的协定。

在 SOAP 封装、SOAP 编码规则和 SOAP RPC 协定之外，这个规范还定义了两个协议的绑定，描述了 HTTP 扩展框架存在或不存在的条件下，SOAP 消息如何包含在 HTTP 报文中。

B.1.3 WSDL——Web Service 描述语言

随着通信协议和消息格式在 Web 中的标准化，以某种格式化的方法描述通信变得越来越重要，其实现的可能性也越来越大。用 WSDL 定义的一套 XML 语法描述的网络服务方

式满足了这种需求。WSDL 把网络服务定义成一个能交换消息的通信端点集。WSDL 为分布式系统提供了“在线帮助”。

一个 WSDL 文档将服务定义为一个网络端点或端口（End Point）的集合。在 WSDL 里，端点及消息的抽象定义与它们具体的网络实现和参数格式绑定是分离的。这样就可以重用这些抽象定义：消息——需要交换数据的抽象描述；端口类型——操作的抽象集合。针对一个特定端口类型的具体协议和数据格式规范构成一个可重用的绑定。一个端口定义成网络地址和可重用绑定的连接，端口的集合定义为服务。因此，一个完整的 WSDL 在定义网络服务时包含如下元素。

- 类型：使用某种类型系统（如 XSD）定义数据类型。
- 消息：通信数据抽象的类型定义。
- 操作：服务所支持动作的抽象描述。
- 端口类型：一个操作的抽象集合，该操作由一个或多个端点支持。
- 绑定：针对一个特定端口类型的具体协议规范和数据格式规范。
- 端口：一个单一的端点，定义成一个绑定和一个网络地址的连接。
- 服务：相关端点的集合。

B.1.4 UDDI——统一描述、发现和集成

UDDI 是一套 Web Service 信息注册标准规范，Web Service 信息注册中心通过实现这套规范开放 Web Service 注册、查询的服务。

UDDI 的核心组件是 UDDI 业务注册，它使用一个 XML 文档来描述企业及其提供的 Web Service。从概念上来说，UDDI 业务注册所提供的信息包含以下三部分。

- 白页（White Page）：包括地址、联系方法和企业标识。
- 黄页（Yellow Page）：包括基于标准分类法的行业类别。
- 绿页（Green Page）：包括该企业所提供的 Web Service 的技术信息，其形式可能是一些指向文件地址或 URL 的指示器，而这些文件地址或 URL 是为服务发现机制服务的。

所有 UDDI 信息注册信息都存储在 UDDI 信息注册中心。通过使用 UDDI 提供的注册服务，企业可以注册那些希望被别的企业发现的 Web Service。企业可以通过 UDDI 商业注册中心的 Web 界面，或使用实现了“UDDI Programmer’s API 标准”的工具，将信息加入到 UDDI 的信息注册中心。UDDI 的注册信息在逻辑上是集中的，在物理上是分布式的，由多个根节点组成，相互之间按一定复制规则进行数据同步。当一个企业在 UDDI 商业注册中心的一个实例中实施注册后，其注册信息会被自动复制到其他 UDDI 根节点，于是就能被任何希望使用这些 Web Service 的人所发现。

B.2 Spring-WS 简介

Spring Web Services（以下简称 Spring-WS）框架是 Springframework 东家 SpringSource 公司旗下的一个子项目，是“专注于基于文档驱动（document-driven）的 Web Services，

并帮助推动契约优先（contract-first）的 SOAP 服务开发，允许利用操作 XML Payloads 的多种方法之一来创建灵活的 Web Services，目前的最新版本是 2.0.2，最新版本可以从 springsource.org 网站下载。用户可以在随书光盘 `resources/springws` 目录下获取 Spring-WS 2.0.2 的发布包。

B.2.1 Spring-WS 特性

Spring-WS 是应用文档驱动（契约优先）的思想来构建灵活 Web Services 的推动者之一，它从传统基于代码优先的 Web Service 开发模式中吸收各种教训，如异构服务互通性、服务契约的多版本等问题，提供了契约和实现之间的松耦合，没有强制将服务契约直接连接到一个业务类，而是用你擅长的任何方式（DOM、SAX、StAX，甚至 XML 映射组件如 JAXB、Castor、JIBX 或者 XMLBeans）来处理 Web 服务报文请求，并通过服务端点调用服务业务逻辑。此外，获取请求映射到服务端点的方式，完全由开发人员自己决定，默认情况下，Spring 提供基于消息内容的映射。总之，它具有以下一些特性：

- Spring 框架的全面支持，你可以使用 Spring 的所有特性；
- Spring-WS 使用契约优先（Contract First）的设计模式，不支持通过代码导出服务；
- 支持几乎所有的 XML API，处理传入 XML 消息的时候不限于 JAXP（Java API for XML Processing），可以选择你所擅长任意的 XML API；
- 灵活的 XML 编组（Marshalling），支持各种 OXM 技术，如 JAXB、XMLBean、Castor 等；
- Security 支持，集成了 Acegi Security，实现 Web Services 认证；
- 提供一个简单易用的 Web 服务模板操作类 `WebServiceTemplate`。

B.2.2 Web Services 开发模式

在开始 Spring-WS 之旅前，有必要回顾一下目前 Web Services 开发的两种模式：一种为代码优先的开发模式（Contract Last）；另一种为契约优先的开发模式（Contract First）。

代码优先的开发模式

代码优先是传统 Web Services 开发世界中最常见、最容易上手的一种开发模式，也是大多数开发人员喜欢的一种开发模式。其开发过程相对比较简单，之所以简单，是因为它迎合了开发人员的惰性。代码驱动 Web 服务构建，是通过相关工具类库（如 CXF 等）直接导出服务类的 Web Service 调用接口，绕过编写数据契约（XSD）及 SOAP 的烦琐过程。

Spring 提供了很多服务导出器，可以将现有的 Bean 导出基于 RMI 或 HTTP 远程调用服务，但还没有提供一个直接将 Bean 导出为 Web Services 的导出器，然而我们可以结合一个优秀的第三方 SOAP 开源框架（如 CXF），就可将 Bean 导出为相应的 Web 服务。本书不再对代码优先的 Web Service 进行讲解，感兴趣的读者可以参阅《Spring 2.x——企业应用开发详解》的第 16 章。

契约优先的开发模式

契约优先的开发模式，是目前 Web Services 开发领域中的最佳实践，也是 Spring 极力倡导的。所谓的契约优先，先要定义 Web 服务的数据契约，也就是服务交互过程中的数据类型及请求响应消息的数据契约，然后再编写相应服务端点的业务处理逻辑。

契约优先与代码优先最大的不同就是自己要编写服务的数据契约，这也是许多沉浸在传统开发模式中的开发人员所不愿做的事。对他们来讲，这是一个“多余”的步骤，又是一个比较“难”入手的步骤。因为自己编写数据契约，必须要去学习 XML 及 XSD 的知识，而且还要去学习 XML 处理的知识，如 OXM 等。而如果采用代码优先，这些过程都无须关注了，要改变这个观念需要一个过程。之所以契约优先的开发模式在业界为被认为是一种最佳实践，是基于以下一些考量：

- 能够提供服务契约多版本；
- 真正做到跨语言、跨平台的服务；
- 提高解析 WSDL 性能；
- 解决对象与 XML 映射阻抗问题；
- 服务契约不受实现接口的影响，实现了服务契约与实现接口的松耦合；
- 能够快速响应业务多变引发服务契约的变化。

B.2.3 Spring-WS 体系及重要 API

WebServiceMessage

WebServiceMessage 是 Spring-WS 中一个核心接口，代表一个与具体传输协议无关的 XML 消息，该接口提供了访问有效负载(Payload)消息的方法，如通过 getPayloadSource()方法获取消息中的 Source 内容，通过 getPayloadResult()获取消息中的 Result 内容，其中 javax.xml.transform.Source 及 javax.xml.transform.Result 表示 XML 的一组输入输出接口，对于不同的 XML 解析模型，都有其相应的实现类，如表 B-1 所示。

表 B-1 Source/Result 实现类

Source/Result 实现类	XML 的输入输出
javax.xml.transform.dom.DOMSource	org.w3c.dom.Node
javax.xml.transform.dom.DOMResult	org.w3c.dom.Node
javax.xml.transform.sax.SAXSource	org.xml.sax.InputSource 和 org.xml.sax.XMLReader
javax.xml.transform.sax.SAXResult	org.xml.sax.ContentHandler
javax.xml.transform.stream.StreamSource	java.io.File、java.io.InputStream 或 java.io.Reader
javax.xml.transform.stream.StreamResult	java.io.File、java.io.OutputStream 或 java.io.Writer

SoapMessage

SoapMessage 也是 Spring-WS 中的另一个核心接口，代表一个基于 SOAP 协议 XML 的消息，它扩展自 MimeMessage 接口，MimeMessage 又扩展自 WebServiceMessage。MimeMessage 提供了多用途网际邮件扩充协议（MIME）操作接口方法，如添加消息附件 MimeMessage.addAttachment() 及访问消息附件 MimeMessage.getAttachment()。

SoapMessage 提供了 SOAP 操作接口方法，如获取 SOAP 头信息 SoapMessage.getHeader()、获取 SOAP 消息体信息 SoapMessage.getSoapBody() 及设置消息动作 SoapMessage.setSoapAction (String soapAction) 等，如果只是想获取消息中的 Source/Result，在代码中尽量使用 WebServiceMessage 接口，避免与具体的协议（MIME、SOAP）产生依赖。

SaajSoapMessageFactory

SaajSoapMessageFactory 是 Spring-WS 提供的一个基于 SAAJ 消息工厂类，它实现自 WebServiceMessageFactory 消息工厂接口，使用带附件的 SOAP API（SAAJ）来创建 SoapMessage。SAAJ 是在松散耦合软件系统中利用 SOAP 协议实现的基于 XML 消息传递的 API 规范，是 Java EE 1.4 规范的一部分，所以它需要在一些高级的应用服务器下才能运行，表 B-2 列出了一个常见的应用服务器支持的 SAAJ 版本。

表 B-2 应用服务器支持的 SAAJ 版本

Application Server 应用服务器	SAAJ 的版本
BEA WebLogic 8 BEA 公司的 WebLogic 8	1
BEA WebLogic 9 BEA 公司的 WebLogic 9	1.1/1.2a
IBM WebSphere 6 6 的 IBM WebSphere	1.2
SUN Glassfish	1.3

Java SE 6 已经包括 SAAJ 的 1.3 版本，实例化一个 SaajSoapMessageFactory 简单，只要在 Spring 上下文中配置一个 SaajSoapMessageFactory 的 Bean 即可。

使用 SaajSoapMessageFactory 如下：

```
<bean id="messageFactory"
      class="org.springframework.ws.soap.saaj.SaajSoapMessageFactory" />
```

SAAJ 基于文档对象模型（DOM），所以它会把整个 SOAP 消息全部读入到内存中，对于一些比较大的 SOAP 消息，会对性能产生一定的负面影响，如果在对性能要求比较高的场合，推荐使用 AxiomSoapMessageFactory 这个消息工厂类，下文将对其进行介绍。

AxiomSoapMessageFactory

AxiomSoapMessageFactory 是 Spring-WS 提供的另一个基于 AXIs 的消息工厂类，它也是实现自 WebServiceMessageFactory 消息工厂接口，使用新一代的 SOAP 引擎（Axis 2）来创建 SoapMessage，AXIs 对象模型（AXIOM）是一个 XML 对象模型，设计用于提高 XML 处理期间的内存使用率和性能，基于 Pull 解析。通过使用 Streaming API for XML (StAX) Pull 解析器，AXIOM（也称为 OM）可以控制解析过程，以提供延迟构建支持。

为了提高 AxiomSoapMessageFactory 读取 SOAP 消息的性能，你可以把 payloadCaching 属性值设置为 false（默认为 true），让其直接从 Socket 中读取 SOAP 消息。需要注意的是，如果设置 payloadCaching 为 false，只能向前读取 SOAP 消息内容，读取过的信息内容无法再次读取，因此在读取 SOAP 消息内容的过程中，要防止其他程序去读取

SOAP 消息，如日志记录等。

使用 AxiomSoapMessageFactory 如下：

```
<bean id="messageFactory"
      class="org.springframework.ws.soap.axiom.AxiomSoapMessageFactory">
  <property name="payloadCaching" value="false"/>
</bean>
```

AXIOM 可以通过 StreamingWebServiceMessage 读取完整的流消息，可直接把消息流设置在响应消息中，而不需要写入到一个文档对象模型树或缓冲区。

MessageContext

MessageContext 由两个部分组成：请求和响应（request 和 response），它代表 C/S 的一组 Q&A，也就是说每一次请求都会新建这样一个对象来持有当前会话中的一组 Q&A。实际上，在客户端向服务端发送 SOAP 消息请求时，所传递的对象（request）就是一个 MessageContext 对象，直到服务端点处理这个对象，才会将 response 注入到 MessageContext 中。

WebServiceConnection:

WebServiceConnection 是 Spring-WS 提供的一个 Web 服务连接接口，对不同的传输协议都有相应的实现类，如 HTTP、JMS、MAIL 等传输协议对应的实现类：HttpURLConnection、HttpServletConnection、MailSenderConnection、MailReceiverConnection、JmsSenderConnection、JmsReceiverConnection。每种传输协议都对应一个发送与接收的实现类，除了基于 HTTP 传输协议的实现类外，我们通过名称就能很容易地区分它们的功能，其中 HttpURLConnection 表示用于创建发送请求 Web 服务连接，HttpServletConnection 表示用于创建接收请求 Web 服务连接。WebServiceConnection 表示客户端向服务端发送一个 Web 服务消息的点到点的连接，它包括两个动作：接收从客户端发来的消息和发送给客户端响应消息。

TransportContext

传输协议上下文，它关联一个 Web 服务连接接口 WebServiceConnection，可以通过 TransportContextHolder 拿到本地线程（ThreadLocal）的传输协议上下文，从而获得 WebServiceConnection 中的一些信息，如基于 HTTP 传输协议，可以获取 HttpServletRequest：

```
...
TransportContext context= TransportContextHolder.getTransportContext();
HttpServletConnection connection= (HttpServletConnection) context.getConnection();
HttpServletRequest request= connection.getHttpServletRequest();
String ipAddress = request.getRemoteAddr();
...
```

XPath

XPath 是第四代的声明语言，XPath 是在 1999 年 11 月 16 日和 XSLT 一起成为正式标准的。XPath 是用作 XSLT 和 XPointer 的对 XML 文档各部分进行定位的语言。它给 XSLT

和 Xpointer（XML 文件内部链接语言）提供一个共同、整合的定位语法，用来定位 XML 文件中各个元素和属性。XPath 除了提供一套定位语法之外，还包括一些函数，它们提供基本的数字运算、布尔运算和字符串处理功能。

XPath 使用一个紧凑的、非 XML 的语法来方便实现 XPath 在 XML 属性值中的使用，它基于 XML 文档的逻辑结构，在该结构中进行导航。除了用于定位，XPath 自身还有一个子集用于匹配，它能验证一个节点是否匹配某个模式。XPath 把一个 XML 文档看成一个树或节点的模型。节点的类型可以有多种，包括元素节点、属性节点和文本节点。

XPath 的基本语法由表达式构成。在计算表达式的值之后产生一个对象，这种对象有以下四种基本类型：节点集合、布尔型、数字型和字符串型，一个完整的 XPath 表达式编码为一个位置路径（location path），它包含一个或多个位置步进（location step），根据需要通过正斜杠（/字符）来定界。因此，一个简单位置路径与定位在某个文件系统上的一个文件的路径相类似。

XPathExpression

XPathExpression 对象是一个 XPath 查询编译过的表现形式，例如 javax.xml.xpath.XPathExpression、Jaxen 的 XPath 类。我们可以使用 Spring 提供的 XPath 表达式 Bean 工厂 XPathExpressionFactoryBean，快速在应用上下文创建相应的 XPathExpression 实例，下面我们以解析查询论坛精华帖子数 SOAP 请求消息报文为例，说明其使用方法。

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
  <bean id="springXPath" class="com.baobaotao.ws.xpath.SpringXPath"
    p:xpathExpression-ref="xpathExpression"/>
  <bean id="xpathExpression"
    class="org.springframework.xml.xpath.XPathExpressionFactoryBean">
    <property name="expression"
      value="/SOAP-ENV:Envelope/v1:getRefinedTopicCountRequest"/>
    <property name="namespaces">
      <map>
        <entry key="SOAP-ENV"> ① 根名称空间
          <value>http://schemas.xmlsoap.org/soap/envelope/</value>
        </entry>
        <entry key="v1">
          <value>http://www.baobaotao.com/ws/server/springws/schema/messages/v1</value>
        </entry> ② 消息元素命名空间
      </map>
    </property>
  </bean>
</beans>
```

上面声明的 `xpathExpression` 表达式使用两个命名空间，分别为报文根命名空间及消息元素命名空间，并把 `xpathExpression` 注入到 `SpringXPath` 中，其中 `SpringXPath` 为解析报文的主类，演示如何应用 Spring 的 `XPathExpression` 表达式解析应用报文过程，为了提高解析的效率，Spring 为我们提供了一个非常灵活的节点元素映射类 `NodeMapper`，该类秉承了 Spring JDBC 中的 `RowMapper` 处理风格，提供了一种简洁的映射处理方式。

```
...
public class SpringXPath {
    private XPathExpression xpathExpression;
    public void doXPath(Node document) {
        xpathExpression.evaluateAsObject(document,
            new NodeMapper() {
                public Object mapNode(Node node, int nodeNum) throws DOMException {
                    Element element = (Element) node;
                    Element startDate = (Element) element.getChildNodes().item(1);
                    Element endDate = (Element) element.getChildNodes().item(3);
                    System.out.println("开始日期: " + startDate.getTextContent());
                    System.out.println("结束日期: " + endDate.getTextContent());
                    return new Object();
                }
            });
    }
    ...
}
```

类似于 Spring JDBC 中 `RowMapper` 的映射行用法，每个匹配的节点映射使用一个匿名内部类，以一种便捷的方式将匹配元素信息映射到目标对象中。

B.2.4 Spring-WS 类库说明

Spring-WS 需要使用到大量第三方依赖类库，了解不同场合需要使用哪些依赖类库会给具体的应用开发带来帮助，表 B-3 对这些类库进行了说明。

表 B-3 Spring-WS 类库

类 库	基本功能	基本功能+XML 配置	说 明
spring-ws-core	√	√	Spring-WS 的核心类库
spring-ws-security	√	√	Spring-WS 的 安全类库
spring-xml	√	√	解析 XML 相关类库
spring-oxm	√	√	对象编组与反编组类库
spring-core	√	√	Spring 的核心类库
xws-security	√	√	Sun 发布的安全类库
stax-api	√	√	以流的方式处理 XML，用该类库提高 XML 的处理性能
wsdl4j	√	√	用于解析一个 WSDL 消息
spring-test	×	×	Spring 测试类库
spring-ws-test	×	×	Java-crumbs 提供测试 spring-ws 类库
spring-jms	×	×	使用 JMS 传输协议进行通信时使用

B.3 应用 Spring-WS 实现 Web 服务

应用 Spring-WS 开发 Web 服务首先我们需要编写服务相应的契约，与 Spring MVC 类似，需要在 web.xml 文件中配置 SOAP 请求信息处理转发器 `MessageDispatcherServlet`，它继承于 `DispatcherServlet`，主要负责将 SOAP 请求信息转发到相应的端点进行处理，接下来就编写相应的服务端点（endpoint），处理输入的 SOAP 请求消息，所谓的端点就是一个使用了 `@Endpoint` 注解的 POJO。

B.3.1 定义 Web 服务契约

Web 服务契约包括数据契约和服务契约，契约以一种独立于具体语言和平台的方式存在，采用 XML 进行描述定义，Spring-WS 通过我们定义的数据契约可以自动生成相应的服务契约（WSDL），为此我们关注的核心是怎样来定义数据契约。到底什么是数据契约，所谓的数据契约，就是用来描述服务交互过程中的数据类型和请求响应消息数据类型结构，采用标准 XSD 进行定义。

虽然手工编写 XSD 不是很困难，但也是一件枯燥无味的工作，我们可以通过相关工具（XMLSpy 或 Trang），从 XML 样本信息直接生成相应的 XSD 文件，虽然这样为我们节省了很多工作，但还是需要我们做一些后续的修饰工作。

我们以查询宝宝淘论坛精华帖子数服务为例，一步步展开契约优先的 Web 服务开发过程，首先我们从查询在一段时间内的精华帖子数 XML 样本信息创建其对应的数据契约 XSD 文件。

代码清单 B-1 BbtForumServiceData.xml：XML 样本信息

```
<getRefinedTopicCountRequest
  xmlns="http://www.baobaotao.com/ws/server/springws/schema/messages/v1">
  <startDate>2011-02-01</startDate>
  <endDate>2011-02-28</endDate>
</getRefinedTopicCountRequest>
```

有了这个样本信息 XML 文件，我们就可以通过 Trang 小工具（附带光盘中提供的 Trang.jar），来生成数据契约 XSD 文件，在 Window 命令窗口中运行 trang.jar，命令的格式如下所示：

```
D:\masterSpring\tools> java -jar trang.jar -I xml -O xsd BbtForumServiceData.xml
                        BbtForumServiceV1.xsd
```

其中，-I 参数表示输入文件的格式，如示例中的 xml；-O 参数表示输出文件的格式，如实例中的 xsd。执行上述命令，在 D:\masterSpring\tools\ 目录中生成了一个 BbtForumServiceV1.xsd 数据契约文件，其内容如下所示：

代码清单 B-2 BbtForumService V1.xsd：数据契约文件

```
<?xml version="1.0" encoding="UTF-8"?>
```

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.baobaotao.com/ws/server/springws/schema/messages/v1"
  elementFormDefault="qualified" ... >
  <xs:element name="getRefinedTopicCountRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="v1:startDate"/> ① 引用定义时间数据类型
        <xs:element ref="v1:endDate"/> 类型
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="startDate" type="xs:NMTOKEN"/> ② 将 xs:NMTOKEN 更改为正确的
  <xs:element name="endDate" type="xs:NMTOKEN"/> 时间类型: xs:date

  <xs:element name="getRefinedTopicCountResponse" type="int"/> ③ 手工添加服务响应
  契约定义
</xs:schema>

```

通过 Trang 工具很快就为我们生成了数据契约的骨架，细心的读者就会发现生成的这个 XSD 文件不是很完美，一是 XSD 文件中②处描述的时间数据类型明显不是我们想要的，需要手工调整为我们目标的数据类型，把 xs:NMTOKEN 更改为时间数据类型 xs:date；二是只为我们生成请求契约定义，没有响应的契约定义，所以③处添加一个服务响应的契约定义。

为了更好说明 Spring-WS 基于契约优先带来的价值，我们模拟业务需求发生变化，要求更改我们服务契约，如果采用传统代码优先模式，我们通常有两种办法，一是直接更改代码服务接口，这会导致所有客户端也要随之更改，二是添加一个服务接口及实现方法，但这这就要求更改服务接口的方法签名，因为同一份服务契约中的操作签名不能相同。

假设宝宝淘论坛现在需要向外提供一个查询在一段时间内某种类型的论坛帖子数，这就要求在原有服务契约中添加一个帖子类型，基于 Spring-WS 我们不需要为上述问题感到头痛，只需发布一个新版本的数据契约就可以了。

代码清单 B-3 BbtForumServiceV2.xsd：在数据契约 V1 基础上创建新数据契约 V2

```

<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.baobaotao.com/ws/server/springws/schema/messages/v2" ①
  elementFormDefault="qualified" ... >
  <element name="getRefinedTopicCountRequest">
    <complexType>
      <sequence>
        <element ref="v1:topicType"/> ② 添加一个帖子类型
        <element ref="v1:startDate"/>
        <element ref="v1:endDate"/>
      </sequence>
    </complexType>
  </element>
  <element name="startDate" type="date"/>

```

注意，更改目标命名空间

```
<element name="endDate" type="date" />
<element name="topicType" type="topicDataType" />
<simpleType name="topicDataType">
  <restriction base="string">
    <pattern value="[A-Z][0-9][0-9][0-9]" /> ③
  </restriction>
</simpleType>
<element name="getRefinedTopicCountResponse" type="int" />
</schema>
```

帖子类型的数据类型及编码规则，当前规则表示只能以大写字母开头，后三位是数据，且长度只能为 4 位，如“A001”

从原来数据契约版本 V1 复制一份数据契约，重命名为 BbtForumServiceV2.xsd，在①处更改目标命名空间，这个命名空间很重要，我们在服务端点处理中，需要用这个命名空间作为标识，在②处定义一个帖子类型，在③处定义帖子类型的数据类型及编码规则。为了简化数据契约，我们在版本 V2 中采用默认的命名空间，到此我们完成两个版本的数据契约创建工作。我们在后续的章节中，将会详细介绍 Spring-WS 如何处理多个契约版本服务请求。

B.3.2 创建并配置服务端点

Spring-WS 是基于 Spring MVC 的，在 Spring MVC 中，所有请求都由 DispatcherServlet 操作，这个 Servlet 会将请求分配给处理请求的控制器。类似的，Spring-WS 也提供了一个前端控制器 MessageDispatcherServlet 处理来自客户端的请求。它是 DispatcherServlet 的子类，用于将 SOAP 请求颁发给相应的服务端点处理。Spring-WS 的服务端点秉承了 Spring MVC 中的基于注解控制器简洁、灵活的风格，可以在一个端点中编写多个处理 SOAP 消息请求方法，为一个端点处理多版本的数据契约创造了条件，而且通过这个服务端点我们还可以做到服务契约与内部服务接口之间的一种松耦合关系。

首先在 web.xml 文件中配置 SOAP 请求颁发处理器，下面是配置片段：

代码清单 B-4 在 web.xml 中配置消息颁发器

```
<servlet>
  ...
  <servlet-name>spring-ws</servlet-name>
  <servlet-class>
    org.springframework.ws.transport.http.MessageDispatcherServlet ①
  </servlet-class>
  <servlet-mapping>
    <servlet-name>ws</servlet-name> ②
    <url-pattern>/service/* </url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>ws</servlet-name> ③
    <url-pattern>*.wsdl</url-pattern>
  </servlet-mapping>
  ...
</servlet>
```

负责 SOAP 请求信息的颁发处理 Servlet

在这个 URI 下开放 Web Service 服务

配置获取相应的服务契约请求映射

在 Spring-WS 中，要编写相应的端点来处理输入的 SOAP 消息。一个端点就是一个使用了 `@Endpoint` 注解的 POJO。本例中编写了 `BbtForumEndpoint.java` 端点类，负责接收处理相应的服务请求，并在该端点类里编写一个 `getRefinedTopicCount` 方法来处理查询在指定时间段内的精华帖子数，如何将客户端发送的 SOAP 消息映射到 `getRefinedTopicCount` 方法来处理。在 Spring-WS2.0 之后，Spring-WS 为我们提供一个非常灵活处理映射请求的注解模型，只要打上几个简单的注解，就能把一个普通的方法赋予处理 SOAP 消息请求响应的能力，本例中查询精华帖子数的 `getRefinedTopicCount` 方法如下所示：

代码清单 B-5 BbtForumEndpoint.java 论坛服务端点类

```

@PayloadRoot(localPart = "getRefinedTopicCountRequest", namespace = MES_NS_V1) ①
@Namespace(prefix = "m", uri = MES_NS_V1) ②
@ResponsePayload ③
public Element getRefinedTopicCount(
    @XPathParam("//m:startDate") String startDate, ④ ← 把 SOAP 请求元素映射
    @XPathParam("//m:endDate") String endDate       到相应的方法参数
) throws Exception {

    LocalDate localStartDate = new LocalDate(startDate); ⑤ ← 把请求日期转换为
    LocalDate localEndDate = new LocalDate(endDate);     本地的日期格式

    int topicCount = bbtForumService.getRefinedTopicCount(localStartDate, localEndDate); ⑥

    DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();
    Document document = documentBuilder.newDocument();
    Element response = document.createElementNS(MES_NS_V1, ⑦ ← 创建响应
        "getRefinedTopicCountResponse");                SOAP 信息
    response.setTextContent(Integer.toString(topicCount));
    return response;
}

```

在①处为查询精华帖子数方法签名打上 `@PayloadRoot` 注解，用它就可以指定哪种类型的消息该方法能够处理，其中的 `localPart` 指定有效负载根元素本地名称，也就是在上文数据契约中定义的请求元素 `<xs:element name="getRefinedTopicCountRequest">` 的名称，`namespace` 指定了有效负载根元素的命名空间的全称，要与上文数据契约中定义的目标命名空间 `targetNamespace` 一致。在本例中的 `MES_NS_V1` 为一个静态常量，其值就是数据契约中定义的目标命名空间。在②处配置了命名空间的相关信息，在③处配置了当前方法的返回响应信息的格式，在④处配置了 SOAP 请求信息与当前方法的参数绑定机制，其中 `@XPathParam` 采用 XPath 表达式将相应的请求元素绑定到当前参数中。在⑤处将请求日期字符串转换为本地日期格式。在⑥处我们通过内部的服务接口，获取指定时间段内的精华帖子数，这与传统的开发模式有很大的不同，在基于代码优先的模式中，我们直接将内部这个服务接口导出为相应的服务契约，这就造成了我们内部的服务接口与服务契约的紧耦合。在⑦处通过标准 `w3c` 创建 SOAP 响应的信息，创建过程很简单，首先通过文档构建器创建一个新的文档，然后根据在契约文档中定义的命名空间及响应远程名称，为当前文档创建一个新的元素，最后设置相应的响应值（如实例中的 `topicCount`），就完成了服务响

应信息的创建工作。为了更清晰地了解整个开发过程，在这里我们暂不采用数据绑定组件（如 Castor、XMLBeans 等），在下面的实例中，我们将会采用数据绑定组件来简化服务响应信息创建工作。

B.3.3 发布 WSDL 文件

到此，我们已经完成了数据契约的定义、服务请求到服务端点的映射配置及服务端点编写，离完成创建 Web 服务只剩一步，那就是创建服务契约，即对外发布 WSDL 文件。在 Spring-WS 中，创建服务契约有两种方式：一是通过手工来创建 WSDL 文件；二是通过 DynamicWsdll1Definition Bean 动态创建 WSDL 文件。在本实例中我们采用动态创建方式，只需在 Web 应用程序上下文中声明一个 DefaultWsdll1Definition Bean 即可，Spring 消息颁发器（MessageDispatcherServlet）通过 WsdllDefinition 接口会自动找到我们定义的 Wsdll，如实例中定义的 BbtForumService Bean。

代码清单 B-6 在 ws-servlet.xml 中定义 DefaultWsdll1Definition

```
<bean id=" BbtForumService "
  class="org.springframework.ws.wsdll.wsdll11.DefaultWsdll1Definition">| ① Spring-WS
                                                                2.0 提供

    <!-- <property name="schemaCollection" >
      <bean id="forumSchemaCollection"
class="org.springframework.xml.xsd.commons.CommonsXsdSchemaCollection">
        <property name="xsds">
          <list>
            <value>/resources/v1/forum_messages.xsd</value>| ②
                                                                加载多个数
                                                                据契约
          </list>
        </property>
        <property name="inline" value="true"/>
      </bean>
    </property>-->
    <property name="schema">
      <bean class="org.springframework.xml.xsd.SimpleXsdSchema">| ③
                                                                加载单个数
                                                                据契约
        <property name="xsd"
          value="/resources/v1/forum_messages.xsd" />
      </bean>
    </property>
    <property name="portTypeName" value=" BbtForumServicePortType"/>
    <property name="locationUri" value="http://localhost:8080/baobaotao/service"/>
    <property name="targetNamespace"
      value="http://www.baobaotao.com/ws/springws/server/definitions"/>
  </bean>
```

Spring-WS 2.0 之后，不再支持 DynamicWsdll1Definition 来定义动态创建 WSDL 文件，而被 DefaultWsdll1Definition 所替代，如实例中①处定义的 DefaultWsdll1Definition，不再需要配置 WSDL 构建器，直接配置一个数据契约的模式加载器即可。Spring-WS 为我们提供两个加载器的实现：一个是 CommonsXsdSchemaCollection 用于加载多个数据契约的模式文件；一个是 SimpleXsdSchema 用于加载单个数据契约的模式文件，如实例中的②、③

处定义。其他的属性配置如 `portTypeName`、`locationUri`、`targetNamespace` 与 1.5 版本配置方式一致，其中 `portTypeName` 用于指定 WSDL 端口类型，`locationUri` 用于指定 Web 服务的 URI 地址。为了 `locationUri` 指定的地址在产品发布的时候便于修改，一种好的编程习惯就是把 “`http://localhost:8080/baobaotao/service`” 配置在属性文件中，利用 Spring 的 `PropertyPlaceholderConfigurer` 从属性文件中读取。

B.3.4 部署 Web 服务

创建一个 Tomcat 配置文件 `baobaotao.xml`，并编写一行映射配置（这里使用 Tomcat 5.5），这种方式无须将 Web 应用打包成 WAR，方便开发测试：

```
<Context path="/baobaotao" docBase="D:/masterSpring/chapter19/webapp"/>
```

将 `baobaotao.xml` 放置到 `<TOMCAT_HOME>/conf/Catalina/localhost` 目录下，启动 Tomcat 服务，键入 `http://localhost:8080/baobaotao/service/BbtForumService.wsdl`，用户将可以看到 `BbtForumService` 对应的 WSDL，如图 B-2 所示。

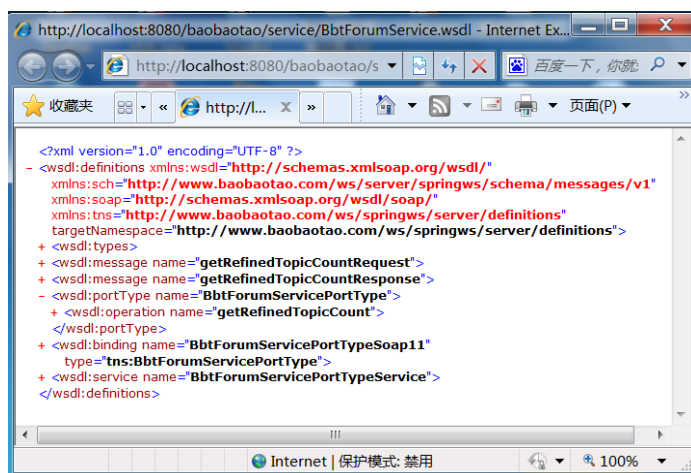


图 B-2 BbtForumService 的 WSDL

阅读这个 WSDL 文档，我们可以知道契约文档中定义的 `getRefinedTopicCount` 已经成功地发布为 Web Service 了。只要拿到这个 WSDL 就可以开发相应的客户端调程序了。

细心的读者可能会发现，我们获取 WSDL 的 URL 与我们采用 XFire 方式获取的有所不同。在 XFire 中，我们通过 “`BbtForumService?wsdl`” 方式，而在 Spring-WS 中通过 “`BbtForumService.wsdl`” 方式，Spring-WS 目前还不支持 “`BbtForumService?wsdl`” 方式，对习惯于 XFire 方式的读者来说，可能会感觉有点怪异。

B.3.5 多版本的数据契约

Spring-WS 倡导基于文档驱动来构建 Web 服务，其主要目的之一是为了能够让我们的 Web 服务适应多变的业务需求，也就是业务需求的变化导致 Web 服务变化过程应该是一个自然、平滑的过程，而不是一个受到各方制约无法平滑过渡的 Web 服务。传统的开发模式

是通过暴露内部的服务接口来导出相应的 Web 服务，导致了服务接口与服务契约紧耦合，这样就很难快速响应多变的业务需求。Spring-WS 为我们提供了一个很好的解决方案，服务契约不再由内部服务接口导出，而是根据我们定义的数据契约来导出，数据契约来源于我们实际的业务需求。因此可以清晰、明确地表示我们实际的业务，如果业务需求发生变化，只要根据当前的数据契约，发布一个新版本的数据契约即可，不但保证了原有服务接口的稳定，还保存了新服务及时响应。

根据代码清单 B-3 中的 BbtForumServiceV2.xsd 数据契约，编写服务端点的处理方法，利用 Spring-WS 创建一个新端点的处理方法很简单，无须创建一个新的端点类，只需在原来的端点类中新增一个处理方法即可，如代码清单 B-7 所示：

代码清单 B-7 BbtForumEndpoint.java 论坛服务端点类

```
@PayloadRoot(localPart = "getRefinedTopicCountRequest", namespace = MES_NS_V2)
@Namespace(prefix = "m", uri = MES_NS_V2)
@ResponsePayload
public GetRefinedTopicCountResponse getRefinedTopicCount(
    @XPathParam("/m:topicType") String topicType,
    @XPathParam("/m:startDate") String startDate,
    @XPathParam("/m:endDate") String endDate)throws Exception {
    LocalDate localStartDate = new LocalDate(startDate.substring(0,10));
    LocalDate localEndDate = new LocalDate(endDate.substring(0,10));
    int topicCount = bbtForumService.getRefinedTopicCount(topicType,
        localStartDate, localEndDate);
    GetRefinedTopicCountResponse response = new GetRefinedTopicCountResponse();
    response.setResult(topicCount);
    return response;
}
```

① ↖ JAXB 创建 SOAP 响应消息封装对象

我们新增版本端点处理方法的方法签名“getRefinedTopicCount”及 SOAP 消息的端口名称“getRefinedTopicCountRequest”都保持不变，Spring-WS 将根据请求 SOAP 消息中的端口指定命名空间映射到相应端点方法进行处理。如实例中，如果客户端发送查询论坛帖子数 SOAP 消息中的端口命名空间是 MES_NS_V1 对应的值时，就映射到旧版本的端点方法进行处理；如果命名空间是 MES_NS_V2 对应的值时，就映射新版本的端点方法进行处理。对于客户端来讲，访问 Web 服务的方法保持一个平滑的过渡。细心的读者会发现，新版本的方法简洁了很多，那是因为本实例利用 JAXB 编组功能，帮助我们快速创建响应 SOAP 消息。在实例中端点方法返回一个 JAXB 创建 SOAP 响应消息封装对象 GetRefinedTopicCountResponse，如何将该对象编组成一个 SOAP 消息？Spring-WS 2.0 版本提供一个端点方法适配器 DefaultMethodEndpointAdapter，用于端点方法返回对象的编组工作，只要在 Spring 容器中声明它即可，Spring-WS 1.5 版本提供一个端点方法编组适配器 GenericMarshallingMethodEndpointAdapter，但在 2.0 版本中已被作废，而且配置比较麻烦，读者可以与下面的代码进行比较：

```
...
<bean
```

```

class="org.springframework.ws.server.endpoint.adapter.DefaultMethodEndpointAdapter">

<!-- spring-WS 1.5声明方式，在spring-WS 2.0中已作废-- -->
<!--
<bean
  class="org.springframework.ws.server.endpoint.adapter.GenericMarshallingMethodEndpointAd
  apter">
  <constructor-arg>
    <bean class="org.springframework.oxm.jaxb.Jaxb2Marshaller">
      <property name="contextPath"
value="com.baobaotao.ws.server.springws.schema.v2" />
    </bean>
  </constructor-arg>
</bean>
-->
...

```

特别当响应消息比较复杂时，Spring OXM 可以帮助我们减少一大部分工作量，Spring-WS 对 Spring OXM 中的数据绑定组件是透明的，因此可以任意选用你熟悉的一种数据绑定组件如 XStream、Castor、XMLBeans、JAXB 等。

B.4 各种客户端调用方式

Spring-WS 提供一个访问服务端 Web Service 的统一操作接口 `WebServiceOperations`，并提供了相应的实现模板类 `WebServiceTemplate`，这个模板类与 JDBC 数据访问模板类 `JdbcTemplate` 风格一致，用来定义发送和接收服务请求及响应消息的模板方法。除此之外，我们还可以采用其他多种方式来访问，如 Saaj、Axis、JAX-WS 等。具体要选用哪种访问方式，要根据开发环境及客户的实际需求而定，如果客户端是基于 Spring 开发，建议采用 Spring-WS 提供的模板类，加上 Spring OXM 的支持，可以大大缩减开发工作量。

B.4.1 使用 WebServiceTemplate

通过 Spring-WS 提供的服务访问模板类 `WebServiceTemplate` 来查询论坛精华帖子数，采用 XMLBeans（其用法，读者可以参阅 15 章）来创建请求和响应的 SOAP 消息映射对象。在本实例中，利用 XMLBeans 组件提供的工具，根据服务定义的数据契约创建 SOAP 消息对应的映射对象，如 `GetRefinedTopicCountRequestDocument.java`、`GetRefinedTopicCount ResponseDocument.java` 等，最后通过 Spring OXM 提供的 `XmlBeansMarshaller` 编组器，可以轻松对 SOAP 消息进行编组和反编组操作，客户端代码如代码清单 B-8 所示。

代码清单 B-8 BbtForumServiceClient.java 查询精华帖子数客户端

```

package com.baobaotao.ws.client.springws;
import org.springframework.ws.client.core.WebServiceOperations;
import java.util.Calendar;
...
public class BbtForumServiceClient {

```

```

private WebServiceOperations webServiceTemplate;
public int getRefinedTopicCount() {
    GetRefinedTopicCountRequestDocument requestDocument =
        GetRefinedTopicCountRequestDocument.Factory.newInstance(); ①
    GetRefinedTopicCountRequestDocument.GetRefinedTopicCountRequest request =
        requestDocument.addNewGetRefinedTopicCountRequest();
    Calendar startDate = Calendar.getInstance();
    startDate.clear();
    startDate.set(2010, Calendar.MAY, 1);
    Calendar endDate = Calendar.getInstance();
    endDate.clear();
    endDate.set(2011, Calendar.MAY, 31);
    request.setStartDate(startDate);
    request.setEndDate(endDate);
    GetRefinedTopicCountResponseDocument getRefinedTopicCountResponseDocument =
        (GetRefinedTopicCountResponseDocument)
        getWebServiceTemplate().marshalSendAndReceive(requestDocument);
    int count = getRefinedTopicCountResponseDocument.getGetRefinedTopicCountResponse();
    return count;
}
...
}

```

通过 `GetRefinedTopicCountRequestDocument` 创建 SOAP 请求消息映射对象实例，设置请求参数，如实例中的开始日期及结束日期。通过模板类中的 `marshalSendAndReceive()` 方法，向服务发起查询请求及接收响应信息，这个方法以 SOAP 请求消息的映射对象为参数，并返回 SOAP 响应消息的映射对象。

完成了客户端 `BbtForumServiceClient` 类的编写之后，还需要在客户端 Bean 配置文件中声明并注入一个 `WebServiceTemplate` 实例来发送和接收消息。

代码清单 B-9 springws-client-context.xml 客户端 Bean 配置文件

```

...
<bean id="marshaller" class="org.springframework.oxm.xmlbeans.XmlBeansMarshaller"/> ①
<bean id="messageSender"
    class="org.springframework.ws.transport.http.HttpURLConnectionMessageSender"/> ②

<!-- 声明基于jms消息发送器-->
<!--
<bean id="connectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory"
    p:brokerURL="tcp://localhost:61616"/>
<bean id="messageSender" class="org.springframework.ws.transport.jms.JmsMessageSender"
    p:connectionFactory-ref="connectionFactory"/>
-->

<bean id="webServiceTemplate" class="org.springframework.ws.client.core.WebServiceTemplate"
    p:messageSender-ref="messageSender"
    p:marshaller-ref="marshaller" ③
    p:unmarshaller-ref="marshaller"
    p:defaultUri="http://localhost:8088/baobaotao/service/"

```

```

/>
<!--
    p:defaultUri="jms:RequestQueue"
-->
<bean id="bbsForumServiceClient" class="com.baobaotao.ws.client.springws.BbsForumServiceClient"
    p:webServiceTemplate-ref="webServiceTemplate"
/>
...

```

④ 基于 JMS 协议 URI 设置

在①处声明一个 `XmlBeansMarshaller` 编组器，用于 SOAP 消息的编组与反编组操作。在②处声明一个基于 `http` 的消息发送者，实例中利用 `URLConnectionMessageSender` 进行消息的传输，Spring-WS 还有其他多种可选择的传输方式（`.jms`、`mail`、`xmpp` 等），每一种传输方式都提供相应的消息发送与接收处理类（`JmsMessageSender`、`JmsMessageReceiver`、`MailMessageSender`、`MailMessageReceiver`、`XmppMessageSender`、`XmppMessageReceiver` 等）。在③处声明一个服务模板操作类，其中 `defaultUri` 设置值为远程 Web 服务 URI。`messageSender` 设置值为传输消息方式，`marshaller` 及 `unmarshaller` 设置值分别为编组与反编组器。

如果不想显式注入 `WebServiceTemplate` 实例，客户端可以扩展 Spring-WS 提供的 `WebServiceGatewaySupport`，与 DAO 类可以扩展 Spring JDBC 中的 `JdbcDaoSupport` 类风格一致，客户端代码如代码清单 B-10 所示。

代码清单 B-10 BbsForumServiceGatewayClient.java 查询精华帖子数客户端

```

package com.baobaotao.ws.client.springws;

import org.springframework.ws.client.core.support.WebServiceGatewaySupport;
import java.util.Calendar;
...
public class BbsForumServiceGatewayClient extends WebServiceGatewaySupport { ①
    public int getRefinedTopicCount() {
        GetRefinedTopicCountRequestDocument requestDocument =
            GetRefinedTopicCountRequestDocument.Factory.newInstance();
        ...
        getWebServiceTemplate().marshalSendAndReceive(requestDocument);
        int count = getRefinedTopicCountResponseDocument().getGetRefinedTopicCountResponse();
        return count;
    }
}

```

代码清单 B-11 springws-client-context.xml 客户端 Bean 配置文件

```

...
<bean id="bbsForumServiceGatewayClient"
    class="com.baobaotao.ws.client.springws.BbsForumServiceGatewayClient"
    p:defaultUri="http://localhost:8088/baobaotao/service/" ②
    p:marshaller-ref="marshaller" p:unmarshaller-ref="marshaller" />
...

```

从 WebServiceGatewaySupport 源码可以看出，它提供两个构造函数：一个是不带参数的构造函数；一个是带 WebServiceMessageFactory 参数的构造函数。如果没有显示为客户端声明 WebServiceTemplate Bean 或 WebServiceMessageFactory Bean，就必须直接设置一个 defaultUri，如实例中的②所示，为默认 WebServiceTemplate 设置远程服务的 URI，否则 Web 服务模板就无法定位远程服务的资源。



实战经验

Web Service 客户端对服务端进行调用时，请求和响应都使用 SOAP 报文进行通信。在开发和测试时，常常查看 SOAP 报文的内容，以便进行分析和调试。TcpTrace 是一款比较小巧的工具，可以让我们截获 TCP/IP 协议上的报文，因为 HTTP、JMS、STMP 等协议都构建在 TCP/IP 基础上，所以可以很容易地截获 Web Service 的 SOAP 请求和响应报文。我们实例中的 Web Service 运行于 8080 端口，可以让 TcpTrace 在 8088 端口上监听，并将 8088 端口监听的报文转发到 8080 端口上。通过这样的设置，TcpTrace 就可以截获请求和响应的 SOAP 报文。

用户可以按以下步骤进行具体的设置。

- 1. 启动 TcpTrace 并设置监听端口和目标地址及端口，如图 B-3 所示。

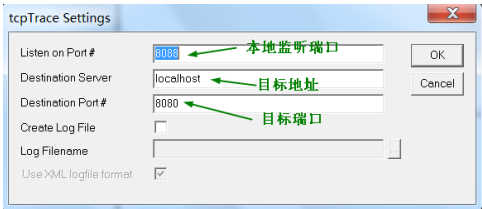


图 B-3 TcpTrace 设置

- 2. 调整客户端程序的 URL 地址指向监听端口：

serviceURL = "http://localhost:8088/baobaotao/service/BbtForumService"

- 3. 重新运行客户端程序，用户将可以在 TcpTrace 中看到 SOAP 的请求和响应报文，如图 B-4 所示。

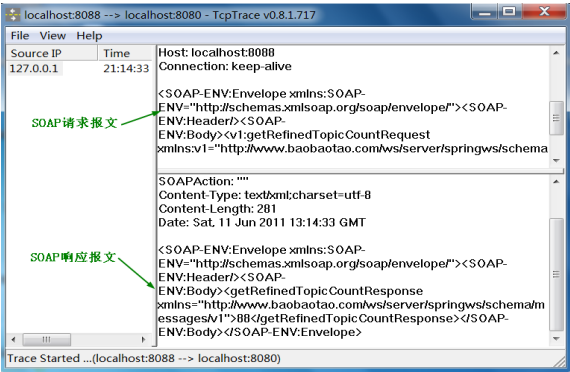


图 B-4 TcpTrace 截取的 SOAP 请求和响应报文

B.4.2 使用 SAAJ 构造客户端程序

虽然 Spring-WS 提供一种便捷的 Web 服务模板操作类，但也要求客户端必须引用 Spring-WS 类库，如果客户端程序不想使用 Spring，Spring-WS 允许我们用其他方式构造客户端访问程序，如可以采用 SaaJ、Axis、JAX-WS 等方式。为了减少篇幅，本例选用其中的 SAAJ（SOAP with Attachments API for Java）方式来演示构造客户端访问程序过程。SAAJ 是利用 SOAP 协议实现的基于 XML 消息传递的 API 规范，扩展了对文档风格的 Web Service 通信的自然支持，支持基于标准接口上的 XML 消息传递，并且得到了供应商的广泛支持。下面，我们来演示如何应用 SAAJ 来构造客户端访问程序，如代码清单 B-12 所示。

代码清单 B-12 通过 SAAJ 构造客户端程序

```
import javax.xml.soap.*;
import javax.xml.transform.TransformerException;
...

public class BbtForumServiceClient {
    public static final String NAMESPACE_URI_V1 =
        "http://www.baobaotao.com/ws/server/springws/schema/messages/v1";
    public static final String PREFIX = "forum";
    private SOAPConnectionFactory connectionFactory;
    private MessageFactory messageFactory;
    private URL url;

    //①客户端构造函数，url为远程Web服务地址
    public BbtForumServiceClient(String url) throws SOAPException,... {
        connectionFactory = SOAPConnectionFactory.newInstance();
        messageFactory = MessageFactory.newInstance();
        this.url = new URL(url);
    }

    //②创建SOAP请求消息报文
    private SOAPMessage createSOAPMessageRequest() throws SOAPException {
        SOAPMessage message = messageFactory.createMessage();
        SOAPEnvelope envelope = message.getSOAPPart().getEnvelope();
        Name requestName = envelope.createName("getRefinedTopicCountRequest", PREFIX,
            NAMESPACE_URI_V1);
        SOAPBodyElement requestElement =
            message.getSOAPBody().addBodyElement(requestName);
        Name startDate = envelope.createName("startDate", PREFIX, NAMESPACE_URI_V1);
        SOAPElement startDateElement = requestElement.addChildElement(startDate);
        startDateElement.setValue("2011-01-01");
        Name endDate = envelope.createName("endDate", PREFIX, NAMESPACE_URI_V1);
        SOAPElement endDateElement = requestElement.addChildElement(endDate);
        endDateElement.setValue("2011-05-01");
        return message;
    }
}
```

//③获取论坛精华帖子数

```

public int getRefinedTopicCount() throws SOAPException, ... {
    SOAPMessage request = createSOAPMessageRequest();
    SOAPConnection connection = connectionFactory.createConnection();
    SOAPMessage response = connection.call(request, url);
    if (!response.getSOAPBody().hasFault()) {
        SOAPBodyElement mileage = (SOAPBodyElement)
            response.getSOAPBody().getChildElements().next();
        return Integer.valueOf(mileage.getValue());
    } else {
        SOAPFault fault = response.getSOAPBody().getFault();
        throw new SOAPException(fault.getFaultString());
    }
}
}

```

在①处通过构造函数创建两个工厂实例，分别是消息连接工厂（`connectionFactory`）和消息工厂（`messageFactory`）。在②处创建 SOAP 请求消息实例对象，通过消息工厂对象创建一个 SOAP 消息实例对象 `message`，并设置消息头及消息主体内容，在创建消息主体时，必须设置相应的 Web 服务目标命名空间，如示例中 `NAMESPACE_URI_V1`。在③处获取论坛精华帖子方法中，通过消息连接工厂对象创建一个消息连接，并调用当前连接的 `call` 方法，向远程 Web 服务发起查询请求，其中 `request` 为②中创建的请求消息对象，`url` 为远程 Web 服务地址。

B.5 Web Service 的测试

在 B.4 节中，我们学习了客户端调用服务端中 Web Service 的方法。在实际应用中，在开放 Web Service 之前需要进行严格的测试，以保证功能的正确性。在一般的框架中，测试 Web Service 往往是一个炼狱般痛苦的过程。Spring-WS 通过 Java-crumbs 提供测试框架（<http://javacrumbs.net/spring-ws-test/>）极大地简化了 Web Service 的测试。允许我们模拟客户端请求及服务端的响应消息，在 SOAP 报文层面开展对服务端代码的测试。Java-crumbs 为 Spring 环境下进行 Web Service 测试提供了一系列的测试工具类，如 `DefaultWsTestHelper`、`WsMockControl` 等，仅通过启用 Spring 容器就可以完成 Web Service 的测试。通过 Java-crumbs 精心设计的测试工具类，对 Web Service 的测试工作便成为一项可以轻松应对的工作。

如果用户在编写服务端 Web Service 的同时，还需要编写客户端调用程序，这时不可避免会希望从客户端角度对 Web Service 进行测试。由于客户端程序需要访问真实的 Web Service，所以需要开启 Web 服务器，让服务端的 Web Service 能够提供服务供客户端访问调用。如果客户端和服务端都在同一个项目中开发，Java-crumbs 允许用户在不启动 Web 服务器的情况下测试客户端程序，其原理是让 Web Service 运行于 JVM 模式下。

B.5.1 基于 SOAP 报文的服务端服务测试

`WsTestHelper` 是 Java-crumbs 提供的一个便于测试 Spring-WS 服务端点的接口，

DefaultWsTestHelper 是其默认的实现类，为我们提供一系列便捷的测试实现方法，如 receiveMessage() 模拟接收客户端 SOAP 请求消息报文，createMessageValidator() 验证服务端点返回的 SOAP 响应消息报文是否与预期一致。

为了测试 Web Service，我们必须准备一个 SOAP 请求报文，用户可以简单地手工编写一个，或通过 SOAP 报文截取工具（如前面我们介绍的 TcpTrace、SOAPScope、Apache Axis 的 TCPMon 等）获得一些可用的 SOAP 请求报文。代码清单 B-13 是一个访问 BbtForumServiceEndpoint 服务端点的请求 SOAP 报文：

代码清单 B-13 request_soap.xml: SOAP 请求消息报文

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <forum:getRefinedTopicCountRequest
      xmlns:forum="http://www.baobaotao.com/ws/server/springws/schema/messages/v1">
      <forum:startDate>2011-05-01</forum:startDate>
      <forum:endDate>2011-05-31</forum:endDate>
    </forum:getRefinedTopicCountRequest>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

代码清单 B-14 request_soap_payload.xml: SOAP 请求消息报文有效负载

```
<forum:getRefinedTopicCountRequest
  xmlns:forum="http://www.baobaotao.com/ws/server/springws/schema/messages/v1">
  <forum:startDate>2011-05-01</forum:startDate>
  <forum:endDate>2011-05-31</forum:endDate>
</forum:getRefinedTopicCountRequest>
```

我们分别将它们保存在 request_soap.xml 和 request_soap_payload.xml 文件中，放置在类路径 com/baobaotao/ws/server/springws/ 下。当该 SOAP 请求报文发送给 BbtForumServiceEndpoint 服务端点后，我们预计它应该返回如代码清单 B-15 所示的正确的 SOAP 响应报文。

代码清单 B-15 response_soap.xml SOAP 响应报文

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <getRefinedTopicCountResponse
      xmlns="http://www.baobaotao.com/ws/server/springws/schema/messages/v1">
      88
    </getRefinedTopicCountResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

我们将其保存在 response_soap.xml 文件中，放置在类路径 com/baobaotao/ws/server/springws/ 下，最后我们着手编写测试 BbtForumServiceEndpoint 端点服务的用例，以验证实际服务端点接收 SOAP 请求和响应报文的正确性。

代码清单 B-16 BbtForumServiceEndpointTest 服务端点测试类

```

import net.javacrumbs.springws.test.helper.DefaultWsTestHelper;
import net.javacrumbs.springws.test.helper.WsTestHelper;
import org.junit.Test;
import static org.junit.Assert.assertNotNull;
...
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {
    "classpath:com/baobaotao/ws/server/springws/applicationContext.xml",
    "classpath:com/baobaotao/ws/server/springws/springws-server-context.xml",
    DefaultWsTestHelper.DEFAULT_CONFIG_PATH})
public class BbtForumServiceEndpointTest {
    @Autowired
    private WsTestHelper wsTestHelper; | ① ← 注入服务端点测试助手类
    private String xmlPath = "com/baobaotao/ws/server/springws/";

    //模拟SOAP 请求消息报文的有效负载请求
    @Test
    public void testSoapPayload() throws Exception{

        MessageContext message =wsTestHelper.receiveMessage(xmlPath+
                                                                "request_soap_payload.xml"); | ②

        assertNotNull(message);
        assertNotNull(message.getResponse().getPayloadSource());
        wsTestHelper.createMessageValidator(message.getResponse()).compare(xmlPath+
                                                                "response_soap.xml"); | ③
    }

    //模拟SOAP 请求消息报文
    @Test
    public void testSoap() throws Exception{
        MessageContext message = wsTestHelper.receiveMessage(xmlPath+"request_soap.xml"); | ④
        assertNotNull(message);
        assertNotNull(message.getResponse().getPayloadSource());
        wsTestHelper.createMessageValidator(message.getResponse()).compare(xmlPath+
                                                                "response_soap.xml"); | ⑤
    }

    ...
}

```

使用 WsTestHelper 测试服务端点首先要加载服务端点及 DefaultWsTestHelper 配置信息到 Spring 容器中，通过 receiveMessage()方法接收 SOAP 请求消息报文并创建响应的消息上下文 MessageContext，如②处及④处所示，WsTestHelper 不仅可以接收模拟的 SOAP 消息请求报文，也可以接收 SOAP 消息请求报文的有效负载。第二步验证响应消息是否为

空，最后通过 `createMessageValidator()` 方法验证响应消息报文与预期的消息报文是否一致，如③处及⑤处所示。

B.5.2 基于 SOAP 报文的客户端服务测试

WsMockControl 是 Java-crumbs 提供的一个便于测试 Spring-WS 客户端的模拟控制器，在不需要服务端支持的情况下就可以很便捷地测试客户端代码的正确性。WsMockControl 模拟控制器与 EasyMock 类似，可以生成相应的 Mock 对象，设定 Mock 对象的预期行为和输出，对 Mock 对象的行为进行验证，如示例中我们利用 WsMockControl 生成 WebServiceMessageSender 的 Mock 对象，模拟向服务端点发送 SOAP 请求消息报文。

代码清单 B-17 BbtForumServiceClientTest 客户端点测试类

```
import com.baobaotao.ws.client.springws.BbtForumServiceClient;
import net.javacrumbs.springws.test.simple.WsMockControl;
...

public class BbtForumServiceClientTest{
    private BbtForumServiceClient bbtForumServiceClient;
    private WebServiceTemplate webServiceTemplate;

    //①初始化服务客户端服务操作模板
    @Before
    public void setUp() throws Exception
    {
        bbtForumServiceClient = new BbtForumServiceClient();
        webServiceTemplate = new WebServiceTemplate();
        XmlBeansMarshaller marshaller = new XmlBeansMarshaller();
        webServiceTemplate.setMarshaller(marshaller);
        webServiceTemplate.setUnmarshaller(marshaller);
        bbtForumServiceClient.setWebServiceTemplate(webServiceTemplate);
    }

    //②模拟测试客户端
    @Test
    public void getRefinedTopicCount(){
        WsMockControl mockControl = new WsMockControl();
        WebServiceMessageSender mockMessageSender = mockControl
            .validateSchema("xsd/v1/forum_messages.xsd")
            .expectRequest("com/baobaotao/ws/server/springws/request_soap.xml")
            .returnResponse("com/baobaotao/ws/server/springws/response_soap.xml")
            .createMock();

        webServiceTemplate.setMessageSender(mockMessageSender);
        Calendar startDate = Calendar.getInstance();
        startDate.clear();
        startDate.set(2011, Calendar.MAY, 1);
        Calendar endDate = Calendar.getInstance();
        endDate.clear();
        endDate.set(2011, Calendar.MAY, 31);
        int count = bbtForumServiceClient.getRefinedTopicCount(startDate,endDate);
        assertEquals(88, count);
        mockControl.verify();
    }
}
```

```

    }
    ...
}

```

使用 WsMockControl 测试客户端首先要创建服务操作模板 webServiceTemplate，如果客户端有使用 OXM 编组器，则需要为 webServiceTemplate 设置相应的编组器，如示例中使用 XmlBeansMarshaller 编组器，最后把服务操作模板 webServiceTemplate 设置到我们的客户端，如①处所示。

第二步我们需要创建一个 Mock 对象，模拟向 Web Service 发送一个 SOAP 请求报文及设置 SOAP 响应报文，在实例中，首先创建一个 WsMockControl 对象，设置请求响应消息验证模式框架（即前文中创建的服务数据契约），然后设置希望的请求报文 request_soap.xml 及响应报文 response_soap.xml，调用模拟器对象 createMock 方法，创建一个消息发送器 WebServiceMessageSender 的 Mock 对象，并把消息发送器 Mock 对象设置到服务操作模板 webServiceTemplate 中，最后可以进行调用客户端服务调用验证操作，如②处所示。

B.5.3 在同一 JVM 模式下 Web 服务集成测试

在不启动 Web 服务器的情况下，通过客户端程序测试 Web Service 的功能，这一崭新的测试方法对于开发人员来说一定颇具吸引力。因为，这意味着用户可以完全在 IDE 环境中运行测试，不需要外部环境的支持。不过应用必须保证客户端和服务端的 Web Service 都位于同一 JVM 中，这时请求报文和响应报文直接在 JVM 内部通道中传输。当使用 JVM 内部通道传输请求和响应的 SOAP 报文时，我们只需要调整服务的消息发送器就可以了，Java-crumbs 为我们提供了一个内存服务消息发送器 InMemoryWebServiceMessageSender。

代码清单 B-18 memory-message-sender.xml 声明基于内存的消息发送器

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
  <bean id="messageSender"
        class="net.javacrumbs.springws.test.helper.InMemoryWebServiceMessageSender"/>
</beans>

```

代码清单 B-19 BbtForumServiceTest JVM 模式测试

```

package com.baobaotao.ws.server.springws;
import com.baobaotao.ws.client.springws.BbtForumServiceGatewayClient;
...
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations =
{"classpath:com/baobaotao/ws/server/springws/applicationContext.xml",
 "classpath:com/baobaotao/ws/server/springws/springws-server-context.xml",
 "classpath:com/baobaotao/ws/client/springws/springws-client-context.xml",
 "classpath:com/baobaotao/ws/client/springws/memory-message-sender.xml" })
public class BbtForumServiceTest {

```

```

@Autowired
private BbtForumServiceGatewayClient bbtForumServiceClient;
@Test
public void getRefinedTopicCount() throws Exception {
    Calendar startDate = Calendar.getInstance();
    startDate.clear();
    startDate.set(2010, Calendar.MAY, 1);
    Calendar endDate = Calendar.getInstance();
    endDate.clear();
    endDate.set(2011, Calendar.MAY, 31);
    assertEquals(88, bbtForumServiceClient.getRefinedTopicCount(startDate, endDate));
}
}

```

以上代码中，我们采用注入了基于内存的消息发送器替换实际运行环境中的基于 Http 的发送器 **URLConnectionMessageSender**，就可轻松完成客户端与服务端的集成测试。

B.6 Web Service 安全

Web Service 是安全的吗？鉴于安全性涉及诸多方面（例如身份验证和授权、数据隐私和完整性等），而 SOAP 规范中根本没有提及安全性这一事实，我们不难理解人们为什么认为答案是否定的。

很少有不需要某种形式的安全性保证的企业级系统。在 Web Service 中，处理 Web Service 安全的过程比处理其他领域应用的安全问题更为复杂，因为 Web Service 具有分布式、无状态的特性。WS-Security 是解决 Web Service 安全问题的规范，大多数商业的 Java EE 应用服务器都实现了 WS-Security 规范，Apache WSS4J 是 WS-Security 的开源实现，WSS4J 通过 SOAP 中 WS-Security 相关的信息对 SOAP 报文进行验证和签名，Spring-WS 通过 WSS4J 对 WS-Security 提供了支持。用户可以从 <http://ws.apache.org/wss4j> 了解更多关于 WSS4J 的信息。

B.6.1 认识 WS-Security

2002 年 4 月，IBM、Microsoft 和 VeriSign 在他们的 Web 站点上提议建立 Web Services Security (WS-Security) 规范。此规范包括安全凭证、XML 签名和 XML 加密的安全性问题，此规范还定义了用户名凭证和已编码的二进制安全性凭证的格式。

2002 年 6 月，OASIS 从 IBM、Microsoft 和 Verisign 接收到了提议的 WS-Security 安全性规范。不久之后就在 OASIS 成立了“Web Service 安全性技术委员会 (WSS TC)”。

2006 年 2 月 15 日，OASIS 通过了 WS-Security 1.1 安全规范，该规范突出了对安全权标支持、消息附件和权限管理的增强。该规范包括 WS-Security 核心规范及用户名权标规范 1.1、X.509 权标规范 1.1、Kerberos 权标规范 1.1、SAML 权标规范 1.1、权限表达 (REL) 权标规范 1.1、带附件的 SOAP (SWA) 规范 1.1 和模式 1.1。

也许读者会提出这样的问题：SSL 也具有完整性和机密性，为什么还需要另外一个 WS-Security 规范呢？之所以要制定 WS-Security 规范，是因为 SSL 存在以下的问题：

- 端对端的通信，脱离传输层就无法保证安全性；
- 消息必须全部加密/签名，而不能针对消息的某部分，没有考虑 XML 处理。

SSL 对应 OSI 的传输层，前面我们提到过 Web Service 是和传输层无关的，将安全问题依附在 SSL 上就违反了 Web Service 与传输层无关的原则。而 WS-Security 对应于 OSI 的应用层，建立在消息层 SOAP 之上。

针对不同领域的细分问题，OASIS 在 WS-Security 的基础上又制定了几十个规范，其中包括 WS-SecureConversation、WS-Federation、WS-Authorisation、WS-Policy、WS-Trust、WS-Privacy 等，形成了一个庞大的 Web Service 安全性协议家族。

有了 WS-Security 规范，用户就拥有在 Web Service 应用中实施完整性、机密性和身份验证等安全需求的规范方法，如图 B-5 所示。

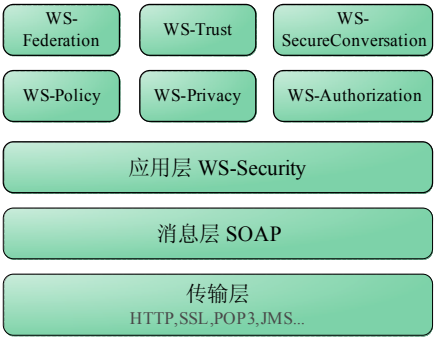


图 B-5 WS-Security 所在位置

B.6.2 Spring-WS 应用 WS-Security 的总体方案

Spring-WS 对 Apache 的 WSS4J 及 SUN 的 XWS-Security 这两种 WS-Security 方式都提供支持，Spring-WS 为 WSS4J、XWSS 实现相应的拦截处理器 Wss4jSecurityInterceptor、XwsSecurityInterceptor。Spring-WS 在发送和接收 SOAP 报文前都可以注册拦截器，对 SOAP 报文进行相应的加工操作。Spring-WS 通过拦截器实施 WSS4J 或 XWSS，当发送 SOAP 报文时，通过注册一系列拦截器，对 SOAP 报文进行加密、签名、添加用户身份信息等处理操作。而在接收 SOAP 报文时，则通过注册一系列的拦截器对 SOAP 进行解密、验证签名、用户身份认证等前置操作，其过程如图 B-6 所示。

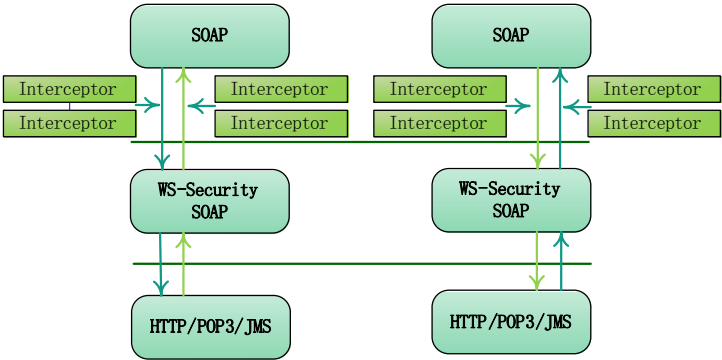


图 B-6 Spring-WS Security 的方案

请求和响应的 SOAP 在发送之前可以通过设置注册拦截器的加密动作策略进行加工处理, 让 SOAP 转换为 WS-Security 的保护格式。而服务端和客户端在接收 SOAP 报文之前, 可以设置注册拦截器的验证动作策略, 将 WS-Security 格式的 SOAP 转换为正常的 SOAP 进行处理。

由于 Spring-WS 在 SOAP 收发过程中定义了多个不同的生命阶段, 所以可以在发送前和接收前完成 SOAP 报文的安全处理工作, 这些操作完全独立于业务处理逻辑, 实施 WS-Security 对于 Web Service 的业务操作是透明的。

B.7 使用 WS-Security

在前面内容中, 我们应用 Spring-WS 技术的各种方式构建了一个查询宝宝淘精华帖子数的 Web 服务, 但单个 Web Service 是没有任何安全性可言的, 任何拿到 WSDL 的人都可以轻松地构造客户端程序访问我们的 Web Service 服务。在本节中, 我们将解决这个问题, 对例子中的 Web 服务添加不同的安全功能。

B.7.1 准备工作

在应用 Spring-WS Security 之前, 必须做一些准备性的工作, 包括搭建安全环境、创建密钥对和证书等内容。

安装 Java 策略文件

策略文件被 JDK 使用, 用以控制加密的强度和算法。确认已经安装对应 JDK 版本的 Unlimited Strength Jurisdiction 策略文件, 这是一个无限制的安全控制文件。用户可以从 <http://java.sun.com/j2se/1.5.0/download.jsp> 或 <http://java.sun.com/j2se/1.4.2/download.html> 页面的底部找到下载的连接。否则在使用 WS-Security 时, 可能会抛出 `java.security.InvalidKeyException: Illegal key size` 的错误信息。

策略文件包括 `local_policy.jar` 和 `US_export_policy.jar` 文件, 将其复制到 `<JAVA_HOME>/jre/lib/security` 目录下。用户可以在光盘 `resources/jce_policy-1_5_0` 找到该策略文件。

读者也许会问: 为何 Sun 不把它集成到 JDK 中去, 而单独“损人不利己”地弄一个链接出来给人下载? 这是因为每个国家, 尤其是美国, 对涉及密码的软件产品控制得非常严格, 在美国国内, 很多密码算法长度都作了限制, 而且某些算法在某些国家没有申请专利, 可以随意使用, 而在某些国家却做了明确限制, 不准使用, 因此 Sun 必须按照惯例行事。

安装 SecurityProvider

WSS4J 使用了 BouncyCastle 的 SecurityProvider, 所以需要事先在 `java.security` 文件中进行配置, 否则运行加密模式的 Spring-WS 认证时, 会抛出以下的出错信息:

`org.apache.ws.security.WSSecurityException: An unsupported signature or encryption algorithm was used unsupported key`

在 `java.security` 文件中（位于 `<JAVA_HOME>/jre/lib/security` 目录中）添加 BouncyCastle Provider 的配置：

```
...
security.provider.1=sun.security.provider.Sun
security.provider.2=sun.security.rsa.SunRsaSign
security.provider.3=com.sun.net.ssl.internal.ssl.Provider
security.provider.4=com.sun.crypto.provider.SunJCE
security.provider.5=sun.security.jgss.SunProvider
security.provider.6=com.sun.security.sasl.Provider
security.provider.7=org.bouncycastle.jce.provider.BouncyCastleProvider
...
```

Spring-WS 发布包中包含了 BouncyCastle SecurityProvider 的类包，位于 `<XFIRE_HOME>/lib/bcprov-jdk15-133.jar` 下，必须将 `bcprov-jdk15-133.jar` 加入到类路径中。用户可以在 <http://docs.safehaus.org/display/PENROSE/Installing+Security+Provider> 中获取更多关于安装 BouncyCastle SecurityProvider 的帮助。

创建密钥对和数字证书

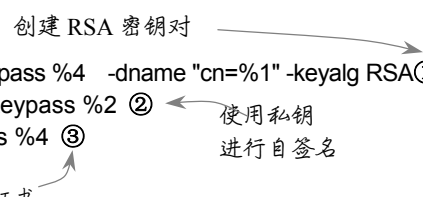
签名和加密需要使用到数字证书和密钥对，可以使用 JDK 提供的 KeyTool 工具创建密钥对和数字证书。我们分别为服务端和客户端创建 RSA 密钥对，并生成各自的 X509 数字证书（包含公钥和数字签名）。服务端和客户端拥有各自的密钥库 JKS 文件，服务端的密钥库保存服务端的密钥对和客户端的数字证书，而客户端的密钥库保存客户端的密钥对和服务端的数字证书。

下面，我们来完成创建服务端和客户端密钥库的工作。

- ① 编写一个用于创建 RSA 密钥对的 `generateKeyPair.bat` 批处理文件。

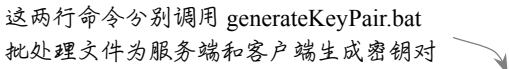
```
rem @echo off
#接受参数
echo alias %1
echo keypass %2
echo keystoreName %3
echo KeyStorePass %4
echo keyName %5

keytool -genkey -alias %1 -keypass %2 -keystore %3 -storepass %4 -dname "cn=%1" -keyalg RSA①
keytool -selfcert -alias %1 -keystore %3 -storepass %4 -keypass %2 ②
keytool -export -alias %1 -file %5 -keystore %3 -storepass %4 ③
```



- ② 编写一个使用 `generateKeyPair.bat` 创建服务端和客户端密钥库的 `generateKeyStore.bat` 批处理文件。

```
call generateKeyPair.bat server serverpass serverStore.jks storepass serverKey.rsa ①
call generateKeyPair.bat client clientpass clientStore.jks storepass clientKey.rsa
```



```
keytool -import -alias server -file serverKey.rsa -keystore clientStore.jks -storepass storepass
-noprompt②
```

← 将服务端的数字证书导入客户端的密钥库

```
keytool -import -alias client -file clientKey.rsa -keystore serverStore.jks -storepass storepass -noprompt
```


③ ← 将客户端的数字证书导入服务端的密钥库

运行该批处理文件后，将分别为服务端和客户端生成一个 Java 密钥库文件，它们分别拥有一个自己的密钥对和对方的数字证书。我们通过表 B-4 对两者密钥库文件的内容进行说明。

表 B-4 密钥库说明

	服务端 Java 密钥库	客户端 Java 密钥库
对应密钥库文件	serverStore.jks	clientStore.jks
密钥库密码	storepass	storepass
库中包含的内容	server 密钥对、client 数字证书	client 密钥对、server 数字证书
密钥对别名	server	client
密钥对私钥的保护密码	serverpass	clientpass

③ 将 serverStore.jks 和 clientStore.jks 复制到 chapter19/src/META-INF/springws 目录下，以便后面的实例代码使用。



提示

我们使用 KeyTool 创建密钥对和数字证书仅是为了开发演示，在实际的应用系统中，需要由 CA 中心产生密钥对，颁发数字证书，且数字证书的签名使用 CA 机构的私钥签名，而非使用某一用户的私钥进行自签名。

B.7.2 使用用户名/密码进行身份认证

对 SOAP 报文进行身份认证的方式很多，不过都是通过在 SOAP 报文头中添加一些安全凭证（Security Token）信息来完成的，主要包括以下一些身份凭证：

- 用户名/密码；
- X.509 证书；
- Kerberos 票据和认证者；
- SIM 卡的移动设备安全性凭证。

其中用户名/密码是最简单的身份认证方式，它不需要密钥、数字证书，所以也就不需要 CA，部署实施简单易行。下面我们就通过例子讲解如何进行基于用户名/密码的 SOAP 认证。这个实例让客户端提供用户名/密码，服务端验证客户端的身份，而客户端按正常方式接收 SOAP 响应报文。

服务端

在服务端的配置文件 springws-server-context.xml 中配置服务安全拦截器，让指定的服务端点拥有用户名/密码的认证功能。

代码清单 B-20 springws-server-context.xml: 身份认证

```
...
<sws:interceptors>
  <sws:payloadRoot
    localPart="getRefinedTopicCountRequest"
    namespaceUri="http://www.baobaotao.com/ws/server/spring
    ws/schema/messages/v2">
    <bean class="org.springframework.ws.soap.security.xwss.XwsSecurityInterceptor">
      <property name="secureResponse" value="false"/>
      <property name="policyConfiguration"
        value="classpath:com/baobaotao/ws/server/springws/security/securityPolicy.xml"/>
      <property name="callbackHandler">
        <bean class="org.springframework.ws.soap.security.xwss.callback.
          SpringDigestPasswordValidationCallbackHandler">
          <property name="userDetailsService" ref="securityService"/>
        </bean>
      </property>
    </bean>
  </sws:payloadRoot>
</sws:interceptors>
<security:global-method-security secured-annotations="enabled"/>
<security:authentication-manager>
  <security:authentication-provider user-service-ref="securityService"/>
</security:authentication-manager>
<bean id="securityService"
  class="com.baobaotao.ws.server.springws.security.ForumUserDetailsService">
</bean>
...
```

① 指定要实施安全的有效负载根元素的本地名称及命名空间

② 基于 XWSS 拦截器

③ 指定安全策略

④ 指定 Spring 密码回调实现类

⑤ 配置查询和认证服务

由于需要对指定的 SOAP 消息报文进行拦截处理，所以必须设置有效负载根元素的本地名称及命名空间，如实例中本地名称“getRefinedTopicCountRequest”及命名空间“http://www.baobaotao.com/...”。在②处指定一个基于 XWSS 的 XwsSecurityInterceptor 拦截器，用于对拦截接收的 SOAP 报文进行安全认证处理。在③处配置了一个安全策略文件，用于对安全策略类型进行配置。

代码清单 B-21 securityPolicy.xml: 安全策略

```
<xwss:SecurityConfiguration dumpMessages="false"
xmlns:xwss="http://java.sun.com/xml/ns/xwss/config">
  <xwss:RequireUsernameToken passwordDigestRequired="true" nonceRequired="true"/>
</xwss:SecurityConfiguration>
```

Spring 安全框架为我们提供了各种密码回调实现类，根据安全策略的不同，可以选用相应的密码回调实现类。如实例中，安全策略是要求密码采用摘要的方式，则在④处设置摘要密码回调实现类 SpringDigestPasswordValidationCallbackHandler，如果密码采用明文发送，则密码回调实现类要设置为 SpringPlainTextPasswordValidationCallbackHandler。

我们通过密码回调实现类的 userDetailsService 属性指定一个查询用户资料的服务

ForumUserDetailsService。

UserDetailsService 指定的类必须实现 org.springframework.security.core.userdetails.UserDetailsService 接口，返回的用户详细资料对象必须实现 UserDetails 接口，其代码如代码清单 B-22 和 B-23 所示。

代码清单 B-22 ForumUserDetailsService

```
package com.baobaotao.ws.server.springws.security;
import org.springframework.security.core.userdetails.UserDetails;
...
public class ForumSecurityService implements UserDetailsService {
    private static final Map<String,String> pwMockDB = new HashMap();
    static{
        pwMockDB.put("tom", "123456");
        pwMockDB.put("john", "123456");
        pwMockDB.put("katty", "123456");
        pwMockDB.put("mike", "123456");
    }
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException{
        if(pwMockDB.get(username)!=null){
            return new ForumUserDetails(username,pwMockDB.get(username));
        }else{
            throw new UsernameNotFoundException("当前用户: "+username+"未授权! ");
        }
    }
    ...
}
```

① 正确用户/密码的模拟数据

代码清单 B-23 ForumUserDetails 用户详细资料

```
package com.baobaotao.ws.server.springws.security;
...
public class ForumUserDetails implements UserDetails {
    private String userName;
    private String password;
    public static final Collection<GrantedAuthority> GRANTED_AUTHORITIES =
        new ArrayList<GrantedAuthority>();
    {
        GRANTED_AUTHORITIES.add(new GrantedAuthorityImpl("ROLE_QUERY_AUTH"));
    }
    public ForumUserDetails(String userName, String password) {
        this.userName = userName;
        this.password = password;
    }
    public Collection<GrantedAuthority> getAuthorities() {
        return GRANTED_AUTHORITIES;
    }
    ...
}
```

ForumSecurityService 负责根据用户名查询正确的密码，这里，我们通过一个 pwMockDB 模拟存储用户名密码的数据库，如①处所示。必须将正确的密码及用户名设置到 ForumUserDetails，并返回给 Spring 密码回调实现类。

客户端

现在服务端的 Web Service 服务已经需要对请求 SOAP 报文进行用户名/密码的认证了，客户端当然要进行相应的调整，以便在发送 SOAP 请求报文时添加用户名/密码的信息。

与 XFire 安全处理机制不同，我们无须更改客户端访问代码，只需要在客户端的配置文件中为 WebServiceTemplate 服务操作模板配置一个安全拦截器即可。

代码清单 B-24 springws-client-context.xml

```
<bean id="webServiceTemplateEnc"
class="org.springframework.ws.client.core.WebServiceTemplate">
  <property name="defaultUri" value="http://localhost:8088/baobaotao/service/">
  <property name="interceptors">
    <list>
      <ref local="securityInterceptor1"/>
    </list>
  </property>
  <property name="messageSender" ref="messageSender"/>
  <property name="marshaller" ref="marshaller2"/>
  <property name="unmarshaller" ref="marshaller2"/>
</bean>
<bean id="securityInterceptor1"
class="org.springframework.ws.soap.security.wss4j.Wss4jSecurityInterceptor">
  <property name="securementActions" value="UsernameToken"/>
  <property name="securementUsername" value="john"/>
  <property name="securementPassword" value="123456"/>
</bean>
```

① 指定相应的安全拦截器

② 用户名/密码认证

在①处，为客户端服务操作模板配置了安全拦截器 Wss4jSecurityInterceptor。在②处，我们定义了一个 securementActions 属性，它定义了需要处理的动作，UsernameToken 表示进行用户名/密码认证的操作。其他的动作包括 Encrypt、Signature、Timestamp、SamlTokenUnsigned 等，可以同时设置多个动作，多个动作之间用空格分隔。

运行客户端代码，通过类似于 TcpTrace 的工具截取 SOAP 请求报文，用户将可以看到带 WS-Security 报文头的 SOAP 报文。

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header>
  <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
SOAP-ENV:mustUnderstand="1">
    <wsse:UsernameToken xmlns:wsu="..." wsu:Id="UsernameToken-1"
xmlns:wsse="...">
      <wsse:Username>john</wsse:Username>
      <wsse:Password Type="...">
```

```

        HD5Wd7hDELTXKPnc8bIUa/O2faQ=
        </wsse:Password>
        <wsse:Nonce EncodingType="...">
            pM9C6MrGsdmffxt9Yaxi1g==
        </wsse:Nonce>
        <wsu:Created>2011-05-13T19:39:00.858Z</wsu:Created>
    </wsse:UsernameToken>
</wsse:Security>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
    <ns2:getRefinedTopicCountRequest
        xmlns:ns2="http://www.baobaotao.com/ws/server/springws/schema/messages/v2">
        <ns2:topicType>TT8888</ns2:topicType>
        <ns2:startDate>2011-05-14+08:00</ns2:startDate>
        <ns2:endDate>2011-05-14+08:00</ns2:endDate>
    </ns2:getRefinedTopicCountRequest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

报文中粗体的部分为 WS-Security 的信息，为了简洁，我们特意删除了报文中一些命名空间的内容。

<wsse:UsernameToken>元素被导入到 SOAP 报头中以携带身份信息。wsse:Username 用于指定一个用户名，而<wsse:Password>则用于指定密码。有两种类型的密码：PasswordText 和 PasswordDigest，PasswordText 以明文格式表示密码，而 PasswordDigest 则被定义为 Base64 编码的 SHA-1 散列值。还有两个可选元素<wsse:Nonce>和<wsu:Created>：前者是发送方创建的一个随机值，后者则是一个时间戳。如果<wsse:Nonce>和<wsu:Created>两个元素中至少出现了一个，计算 PasswordDigest 的算法如下（每次 PasswordDigest 的值都是不一样的）：

$$\text{PasswordDigest} = \text{Base64}(\text{SHA-1}(\text{nonce} + \text{created} + \text{password}))$$

如果没有使用其他的安全机制，PasswordDigest 是通过非保密渠道发送用户名和口令的最佳方法。即使使用 XML 加密对<wsse:Password>元素进行加密，PasswordText 依然可以使用。

B.7.3 对 SOAP 报文进行数字签名

使用用户名/密码虽然可以验证 SOAP 请求报文发送者的身份实现授权访问，但是服务端却无法保证报文在传输过程中没有被篡改——黑客可以截取使用了 UsernameToken 的 SOAP 报文并在篡改后再发送给服务端，就会使 SOAP 报文的完整性遭受破坏。

此外，仅使用 UsernameToken 的 SOAP，客户端用户可以抵赖自己的操作行为，因为黑客确实可以通过一些手段（如键盘监听、暴力破解等）获取用户的密码。

而数字签名则可以解决以上的问题，保证交易的完整性和不可抵赖性。客户端通过私钥对 SOAP 报文进行数字签名，由于私钥只为人拥有，因此不可抵赖性得到了保证。数字签名其实是使用私钥对报文的摘要进行加密，只有报文在传输过程中不被篡改，接收端在进行数字签名验证时才可能成功，因此完整性又得到了保证。

下面，我们在客户端对请求 SOAP 进行数字签名，而服务端验证客户端签名的合法性。客户端使用 client 私钥进行数字签名，服务端使用 client 的数字证书(包含 client 的公钥)验证客户端的签名。

服务端

服务端在验证客户端的签名时，必须访问 serverStore.jks 中的 client 数字证书，所以需要进行相应的配置，如代码清单 B-25 所示。

代码清单 B-25 springws-server-context.xml: 配置数字签名

```
...
<sws:interceptors>
<sws:payloadRoot localPart="getRefinedTopicCountRequest"
namespaceUri="http://www.baobaotao.com/ws/server/springws/schema/messages/v2">
    <bean id="serverSecurityInterceptor"
class="org.springframework.ws.soap.security.wss4j.Wss4jSecurityInterceptor">
        <property name="validationActions" value="Signature"/> ① 数字签名动作
        <property name="validationSignatureCrypto" ref="serverCrypto"/> ②
        <property name="validationCallbackHandler">
            <bean class="org.springframework....KeyStoreCallbackHandler">
                <property name="keyStore">
                    <bean class="org.springframework....KeyStoreFactoryBean">
                        <property name="password" value="serverpass"/>
                    </bean> ③ 密钥对私钥的保护密码
                </property>
                <property name="privateKeyPassword" value="serverpass"/>
            </bean>
        </property>
    </bean>
</sws:payloadRoot>
</sws:interceptors>
<bean id="serverCrypto"
class="org.springframework.ws.soap.security.wss4j.support.CryptoFactoryBean"> ④
    <property name="keyStorePassword" value="storepass"/>
    <property name="keyStoreLocation" value="classpath:META-INF/springws/serverStore.jks"/>
</bean>
```

在①处配置了一个 validationActions 属性，它定义了需要处理的动作，Signature 表示进行数字签名验证操作。在②处配置一个 validationSignatureCrypto 属性，serverCrypto 表示对数字签名的安全提供者，Spring-ws 为我们提供一个便捷创建相应安全提供者的 Crypto FactoryBean 工厂 Bean。在默认情况下，CryptoFactoryBean 将返回一个 org.apache.ws.security.components.crypto.Merli 的实例，可以通过设置 cryptoProvider 属性，指定其他的安全提供者，具体的配置如④处所示，其中 keyStorePassword 设置访问 serverStore.jks 的密码，keyStoreLocation 设置密钥库文件的所在位置。我们还需要设置一个 validationCallbackHandler 属性，指定一个 KeyStoreCallbackHandler 指向相应的密钥库，并设置密钥对私钥的保护密码。

客户端

客户端必须设置相应的拦截器,对 SOAP 报文进行数字签名,如实例中的 Wss4jSecurityInterceptor,与服务端的拦截器设置有所不同,客户端要设置相应的加密动作,而服务端是设置相应的验证动作。客户端的私钥别名为 client,存储在 clientStore.jks 的密钥库中。访问密钥库和私钥都必须提供密码,因此必须进行相应的设置。

代码清单 B-26 springws-client-context.xml

```
...
<bean id="webServiceTemplateEnc"
      class="org.springframework.ws.client.core.WebServiceTemplate">
    <property name="defaultUri" value="http://localhost:8088/baobaotao/service/" />
    <property name="interceptors">
        <list>
            <ref local="securityInterceptor2" />
        </list>
    </property>
    ...
</bean>
<bean id="securityInterceptor2"
      class="org.springframework.ws.soap.security.wss4j.Wss4jSecurityInterceptor">
    <property name="securementActions" value="Signature" />①
    <property name="securementUsername" value="client" /> ②
    <property name="securementPassword" value="clientpass" /> ③
    <property name="securementSignatureCrypto" ref="clientCrypto"/>④
</bean>
<bean id="clientCrypto"
      class="org.springframework.ws.soap.security.wss4j.support.CryptoFactoryBean">
    <property name="keyStorePassword" value="storepass"/>
    <property name="keyStoreLocation" value="classpath:META-INF/springws/clientStore.jks"/>
</bean>
...
```

密钥库文件所在位置及访问密码

和用户名/密码进行身份认证相似,在进行数字签名时,也需要提供用户名和密码,不过两者的用途是不一样的,后者的用户名为密钥库中密钥对的别名,密码为私钥的访问密钥。在①处设置一个 securementActions 属性,它定义了需要处理的动作,Signature 表示进行数字签名加密操作。在②处和③处我们设置 clientStore.jks 密钥库中客户端密钥对的别名及密码。在④处引用一个数字加密相应的安全提供者 clientCrypto。

运行 BbtForumServiceEncClient 后,查看 SOAP 请求报文,用户将看到报文头拥有一个<ds:Signature>元素,包含了签名信息<ds:SignedInfo>、<ds:SignatureValue>、<ds:KeyInfo>等元素,它们分别代表签名信息、签名值以及签名所用密钥的信息。

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <wsse:Security xmlns:wsse="..." SOAP-ENV:mustUnderstand="1">
      <ds:Signature xmlns:ds="..." Id="Signature-1">
```

```

<ds:SignedInfo>
  <ds:CanonicalizationMethod Algorithm=".."/>
  <ds:SignatureMethod Algorithm="..."/>
  <ds:Reference URI="#id-2">
    <ds:Transforms>
      <ds:Transform Algorithm=".."/>
    </ds:Transforms>
    <ds:DigestMethod Algorithm=".."/>
    <ds:DigestValue>...</ds:DigestValue>
  </ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>
  ....
</ds:SignatureValue>
<ds:KeyInfo Id="KeyId-4B75EA4B8C9BD1796613079951075842">
  <wsse:SecurityTokenReference>
    <ds:X509Data>
      ....
    </ds:X509Data>
  </wsse:SecurityTokenReference>
</ds:KeyInfo>
</ds:Signature>
</wsse:Security>
</SOAP-ENV:Header>
<SOAP:Body Id='Body'>
  ...
</SOAP:Body>
</SOAP-ENV:Envelope>

```

B.7.4 对 SOAP 报文体进行加密

虽然通过数字签名解决了完整性和不可抵赖性的安全问题，但报文体还是以明文的方式进行发送，在传输过程中，报文的内容有可能被监视，保密性得不到保证。如果报文体中包含了一些敏感内容，则发送者希望报文的内容能以加密的方式进行传输，防止窥视。通过对 SOAP 报文体进行加密，即可解决保密性的问题。

客户端使用服务端的公钥对请求 SOAP 报文进行加密，服务端公钥包含在服务端的数字证书中。clientStore.jks 中服务端数字证书的别名为 server，访问服务端数字证书不需要密码。服务端需要使用私钥进行解密，服务端私钥包含在服务端的密钥对中，在 serverStore.jks 中服务端密钥对的别名为 server，访问私钥的密码为 serverpass。

在 Spring-WS 中对 SOAP 报文体进行加密解密需要解决的主要问题就是注册相应的拦截器，并为其提供相应的访问密钥信息。

服务端

服务端处理加密的 SOAP 请求报文前，需要通过拦截器将其解密。解密的操作需要访问 serverStore.jks 的 server 私钥，所以要进行相应的配置，如代码清单 B-17 所示。

代码清单 B-27 springws-server-context.xml: 报文加密

```

...
<sws:interceptors>
  <sws:payloadRoot localPart="getRefinedTopicCountRequest"
    namespaceUri="http://www.baobaotao.com/ws/server/springws/schema/messages/v2">
    <bean id="serverSecurityInterceptor"
      class="org.springframework.ws.soap.security.wss4j.Wss4jSecurityInterceptor">
      <property name="validationActions" value="Encrypt"/> ①
      <property name="validationDecryptionCrypto" ref="serverCrypto"/> ②
      <property name="validationCallbackHandler">③
        <bean
          class="org.springframework.ws.soap.security.wss4j.callback.KeyStoreCallbackHandler">
            <property name="keyStore">
              <bean
                class="org.springframework.ws.soap.security.support.KeyStoreFactoryBean">
                  <property name="password" value="serverpass"/>
                </bean>
              </property>
              <property name="privateKeyPassword" value="serverpass"/>
            </bean>
          </property>
        </bean>
      </sws:payloadRoot>
    </sws:interceptors>
    <bean id="serverCrypto"
      class="org.springframework.ws.soap.security.wss4j.support.CryptoFactoryBean">
      <property name="keyStorePassword" value="storepass"/>④
      <property name="keyStoreLocation" value="classpath:META-INF/springws/serverStore.jks"/>⑤
    </bean>
  
```

密钥库访问密码，
密钥库文件位置



在①处配置一个 `validationActions`，它定义了需要处理的动作，`Encrypt` 表示进行加密验证操作。在②处配置一个 `validationDecryptionCrypto` 属性，验证的时候对加密的报文进行解密。在③处配置一个 `validationCallbackHandler` 属性，指定一个 `KeyStoreCallbackHandler` 指向相应的密钥库，并设置密钥对私钥的保护密码。

客户端

客户端通过注册拦截器使用 `server` 数字证书中的公钥对报文体进行加密，`server` 数字证书不需要访问密钥，因此客户端只需要指定访问 `server` 数字证书的必要信息即可。

代码清单 B-28 springws-client-context.xml

```

...
<bean id="webServiceTemplateEnc"
  class="org.springframework.ws.client.core.WebServiceTemplate">
  <property name="defaultUri" value="http://localhost:8088/baobaotao/service"/>
  <property name="interceptors">
    <list>
      <ref local="securityInterceptor3"/>
    
```



```

        </list>
    </property>
    ...
</bean>
<bean id="securityInterceptor3"
    class="org.springframework.ws.soap.security.wss4j.Wss4jSecurityInterceptor">
    <property name="securementActions" value="Encrypt" />①
    <property name="securementUsername" value="client" />②
    <property name="securementPassword" value="clientpass" />③
    <property name="securementEncryptionCrypto" ref="clientCrypto"/>④
    <property name="securementEncryptionParts"
    value="{Content}{http://www.baobaotao.com/ws/server/springws/schema/messages/v2}topicType"
    /> ⑤
    <property name="securementEncryptionUser" value="server"/>
</bean>
<bean id="clientCrypto"
    class="org.springframework.ws.soap.security.wss4j.support.CryptoFactoryBean">
    <property name="keyStorePassword" value="storepass"/>
    <property name="keyStoreLocation" value="classpath:META-INF/springws/clientStore.jks"/>
</bean>
...

```

密钥库访问密码
 钢库文件位置



在①处设置一个 `securementActions` 属性，它定义了需要处理的动作，`Encrypt` 表示进行报文加密操作。在②处和③处我们设置 `clientStore.jks` 密钥库中客户端密钥对的别名及密码，在④处引用一个数字加密相应的安全提供者 `clientCrypto`。在⑤处设置一个 `securementEncryptionParts` 属性指定要加密的报文元素，对匹配的报文消息元素进行加密处理，而报文其他部分则不做加密，如需要对多个报文元素进行加密，则用分号进行分隔，其格式为“`{Content}{命名空间}加密元素;{Element}{命名空间}加密元素`”。

运行 `BbtForumServiceEncClient`，观察加密后的 SOAP 请求报文，其结构如下所示。

```

1 <soap:Envelope>
2   <soap:Header>
3     <wsse:Security>
4       <xenc:EncryptedKey Id="EncKeyId-4B3896...">
5         <xenc:EncryptionMethod
6           Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
7         <ds:KeyInfo>
8           <wsse:SecurityTokenReference>
9             <ds:X509Data>
10               <ds:X509IssuerSerial>
11                 <ds:X509IssuerName>
12                   CN=server
13                 </ds:X509IssuerName>
14                 <ds:X509SerialNumber>
15                   1307003693
16                 </ds:X509SerialNumber>
17               </ds:X509IssuerSerial>
18             </ds:X509Data>
19           </wsse:SecurityTokenReference>

```

```

20         </ds:KeyInfo>
21         <xenc:CipherData>
22             <xenc:CipherValue>
23                 VIKqzDBnJqz...
24             </xenc:CipherValue>
25         </xenc:CipherData>
26         <xenc:ReferenceList>
27             <xenc:DataReference URI="#EncDataId-18687346" />
28         </xenc:ReferenceList>
29     </xenc:EncryptedKey>
30 </wsse:Security>
31 </soap:Header>
32 <soap:Body>
33     <xenc:EncryptedData Id="EncDataId-18687346">
34         <xenc:EncryptionMethod
35             Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
36         <xenc:CipherData>
37             <xenc:CipherValue>
38                 SIEBvI0nJy7ZI3...
39             </xenc:CipherValue>
40         </xenc:CipherData>
41     </xenc:EncryptedData>
42 </soap:Body>
43 </soap:Envelope>

```

BbtForumServiceEncClient 对 SOAP 加密主要完成了以下几个操作：

- soap:Body 的内容被加密（32~42）；
- 使用的算法是 aes128-cbc 对称加密算法(35)；
- 对称密钥被公钥加密后也包含在该消息中传输（21~25），其使用的加密算法是 RSA（6）；
- 使用 server 的数字证书对对称密钥进行 RSA 加密（8-B）。

B.7.5 组合安全策略

以上三节所介绍的都基于单一安全策略机制，此外只在客户端应用了安全策略。有时我们可能需要在 Web Service 的服务端和客户端同时使用安全策略，不但保证客户端请求 SOAP 报文的安全性，同时也保证服务端响应 SOAP 报文的安全性。每一方都可以采用多种组合的安全策略，如使用“数字签名+报文加密”或“身份认证+报文加密”等组合方式。

从请求和服务的角度上看，Web Service 的交互两端分为客户端和服务端，但在实施安全策略时两者是对等的。即如果客户端的请求 SOAP 希望使用 WS-Security，客户端需要注册并配置拦截器的加密动作，服务端则相应地注册并配置拦截器的验证动作。相应的，如果服务端的响应 SOAP 希望使用 WS-Security，服务端需要注册并配置拦截器的加密动作，而客户端需要注册并配置拦截器的验证动作。我们完全可以在前三节的基础上完成所有组合的安全策略，本节以“数字签名+报文加密”的组合方式进行 SOAP 请求及响应消息双向加密进行演示。

服务端

服务端在接收客户端请求 SOAP 报文时，需要使用 server 私钥进行数字签名验证及解密处理，在发送响应 SOAP 报文时需要使用数字证书进行签名及加密处理。

代码清单 B-29 springws-server-context.xml

```

...
<sws:interceptors>
  <sws:payloadRoot localPart="getRefinedTopicCountRequest"
    namespaceUri="http://www.baobaotao.com/ws/server/springws/schema/messages/v2">

    <bean id="serverSecurityInterceptor"
      class="org.springframework.ws.soap.security.wss4j.Wss4jSecurityInterceptor">
        <property name="validationActions" value="Signature Encrypt"/>
        <property name="validationSignatureCrypto" ref="serverCrypto"/>
        <property name="validationDecryptionCrypto" ref="serverCrypto"/>
        <property name="validationCallbackHandler">
          <bean class="org.springframework...KeyStoreCallbackHandler">
            <property name="keyStore">
              <bean class="org.springframework....support.KeyStoreFactoryBean">
                <property name="password" value="serverpass"/>
              </bean>
            </property>
            <property name="privateKeyPassword" value="serverpass"/>
          </bean>
        </property>
        <property name="securementEncryptionUser" value="client"/>
        <property name="securementActions" value="Signature Encrypt" />
        <property name="securementUsername" value="server" />
        <property name="securementPassword" value="serverpass" />
        <property name="securementSignatureCrypto" ref="serverCrypto"/>
        <property name="securementEncryptionCrypto" ref="serverCrypto"/>
        <property name="securementEncryptionParts"
          value="{http://www.baobaotao.com/ws/server/springws/schema/messages/v2}result"/>
      </bean>
    </sws:payloadRoot>
  </sws:interceptors>
  <bean id="serverCrypto"
    class="org.springframework.ws.soap.security.wss4j.support.CryptoFactoryBean">
    <property name="keyStorePassword" value="storepass"/>
    <property name="keyStoreLocation" value="classpath:META-INF/springws/serverStore.jks"/>
  </bean>

```

对请求 SOAP 报文进行解密处理 ①

对响应 SOAP 报文进行加密处理 ③

②

④

密钥库访问密码，
密钥库文件位置 ⑤ ⑥

在①处配置一个 validationActions，它定义了需要处理的动作，Signature 表示进行数字签名验证操作，Encrypt 表示进行加密验证操作，在③处配置对响应 SOAP 报文消息进行数据签名及加密处理，在④处对指定响应 SOAP 消息部分报文进行加密处理。

客户端

客户端需要完成两件事：首先，对 SOAP 请求报文进行数字签名及加密处理，然后接收对 SOAP 响应报文进行数字签名验证及解密。

代码清单 B-30 springws-client-context.xml

```

...
<bean id="webServiceTemplateEnc"
    class="org.springframework.ws.client.core.WebServiceTemplate">
    <property name="defaultUri" value="http://localhost:8088/baobaotao/service/" />
    <property name="interceptors">
        <list><ref local="securityInterceptor4"/></list>
    </property>
    ...
</bean>
<bean id="securityInterceptor4"
    class="org.springframework.ws.soap.security.wss4j.Wss4jSecurityInterceptor">
    <property name="securementActions" value="Signature Encrypt" />
    <property name="securementUsername" value="client" />
    <property name="securementPassword" value="clientpass" />
    <property name="securementSignatureCrypto" ref="clientCrypto" />
    <property name="securementEncryptionCrypto" ref="clientCrypto" />
    <property name="securementEncryptionUser" value="server" />
    <property name="securementEncryptionParts"
value="{Content}{http://www.baobaotao.com/ws/server/springws/schema/messages/v2}topicType" />
    <property name="validationActions" value="Signature Encrypt" />
    <property name="validationSignatureCrypto" ref="clientCrypto" />
    <property name="validationDecryptionCrypto" ref="clientCrypto" />
    <property name="validationCallbackHandler">
        <bean class="org.springframework...KeyStoreCallbackHandler">
            <property name="keyStore">
                <bean class="org.springframework...support.KeyStoreFactoryBean">
                    <property name="password" value="clientpass" />
                </bean>
            </property>
            <property name="privateKeyPassword" value="clientpass" />
        </bean>
    </property>
</bean>
<bean id="clientCrypto"
    class="org.springframework.ws.soap.security.wss4j.support.CryptoFactoryBean">
    <property name="keyStorePassword" value="storepass" />
    <property name="keyStoreLocation" value="classpath:META-INF/springws/clientStore.jks" />
</bean>
...

```

对请求 SOAP 报文消息进行加密处理

①

②

③ 对指定的消息元素进行加密

对返回响应消息进行解密处理

④

⑤ 密钥库访问密码，
密钥库文件位置

⑥

在①处对 SOAP 请求报文进行数字签名及加密处理进行相关配置，值得注意的是组合

动作的设置，多个动作空格分隔，动作的执行顺序和动作的编写顺序一致，如“Signature Encrypt”表示先数字签名再进行加密。

客户端同时使用加密和签名的安全策略后，就可以保证请求 SOAP 报文的保密性、完整性和不可抵赖性，解决信息安全领域三个最关键的问题。

B.8 小结

较之 Axis、CXF 等，Spring-WS 在实施 Web Service 方面更加灵活高效，加上 Spring 框架的全面支持，你可以应用 Spring 的所有特性，支持几乎所有的 XML API，处理传入 XML 消息的时候就不限于 JAXP，可以选择你所擅长的任意 XML API，各种强大 OXM 组件的支持如 XMLBeans、Castor 等，处理 SOAP 请求响应消息变得轻松自然，加上 Spring-WS 基于文档驱动来构建 Web 服务，不再依赖内部的服务接口，使得我们 Web 服务快速响应多变业务需求成为可能。

Spring-WS 支持多种传输协议包括 HTTP、JMS、EMAIL 等，都提供基于相应协议的发送器，如 `URLConnectionMessageSender`、`JmsMessageSender`、`MailMessageSender`，各种传输对于客户端是完全透明，可以根据需要进行切换设置。Spring-WS 为客户端提供一个便捷访问服务的模板操作类 `WebServiceTemplate`，当然用户也可选择其他访问方式，如 SAAJ、AXIS、JAX-WS 等方式。

技术可用性的一个很大的标准是它是否方便测试，Java-crums 与 Spring 测试框架组合为测试 Spring-WS 提供了一流的支持，通过 `WsTestHelper` 工具类，我们可以采用 SOAP 消息报文对服务端点进行测试，通过 `WsMockControl` 模拟控件对客户端进行模拟测试，通过 `InMemoryWebServiceMessageSender` 内存消息发送器，能够在不启动 Web 容器的情况下测试 Web Service，Web Service 的测试工作变得不再像原来那样让人畏惧。

Web Service 是分布式的应用，SOAP 报文可能在不安全的传输层传输，应用安全受到极大的挑战。WS-Security 是专门为解决 Web Service 应用安全而制定的规范。Spring-WS 通过 Apache 的 WSS4J 及 SUN 的 XWS-Security 对 WS-Security 提供了全面的支持。Spring-WS 通过注册一系列的拦截器对接收和发送的 SOAP 报文进行安全处理，Web Service 的业务逻辑对 WS-Security 是透明的。

本章中，我们分别讲解了使用 UsernameToken 进行身份认证、对 SOAP 报文进行数字签名、对 SOAP 报文进行加密的 WS-Security 实施方法，接着我们还讲解了如何在客户端和服务端使用组合的安全策略。相信读者朋友可以在此基础上举一反三，进行各种组合安全策略的应用。