

## 第11章 服务器到服务器

传统的服务器和分布式通信对组件模式系统的依赖性很强，例如 COM或者CORBA。这些技术并不真正适合于Internet，它们有一个依赖程度和/或平台问题。实际上，甚至在 Intranet中，可以发现你将得到这样一些信息，即远程对象被写入怎样的组件结构，它们存在于哪些平台。在并不知道任何关于共享的分布对象结构的条件下，能够更容易地写出客户应用程序吗？在Internet环境中恐怕你别无选择。

使用XML作为一个分布组件模式，我们能够克服任何所遭遇的异构结构和平台问题。XML从根本上说，就是一种文本，而且它能够被任何平台和现有的语言所理解，能够传送请求消息和响应于任何环境。

这一章我们将主要关注用于服务器到服务器通信的技术和技巧，介绍一些使用它们的例子，并且介绍它们是怎样服务于所举的例子的。

### 11.1 XML的传送

首先，让我们关注一下有关用于XML传输所使用方式的大体情况，其中一些是传统的，另一些是新的，我们将考虑使用FTP、MSMQ、HTTP和SMTP来传送XML。

下面这幅图将描述这样一种情景：一个中心书籍发行商拥有成千上万的大量书籍，一个个体书籍零售商每周将从发行商处取得几百个书目，而消费者将从零售商处得到自己需要的书目（参见图11-1）。

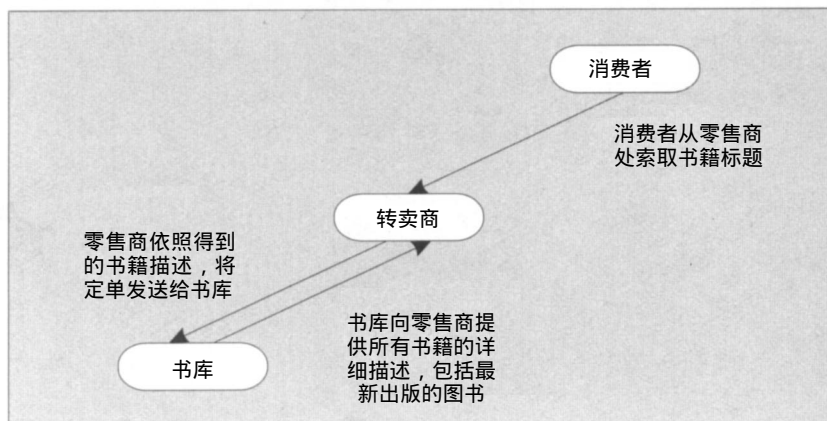


图 11-1

现在我们已经讨论了在这个链中包含的有关过程，但是XML将作用于何处呢？下面这个图表将强调在不同服务器之间传输消息所使用的方式，随后的文字将解释在其中每一步里是如何

利用XML的（参见图 11-2）。

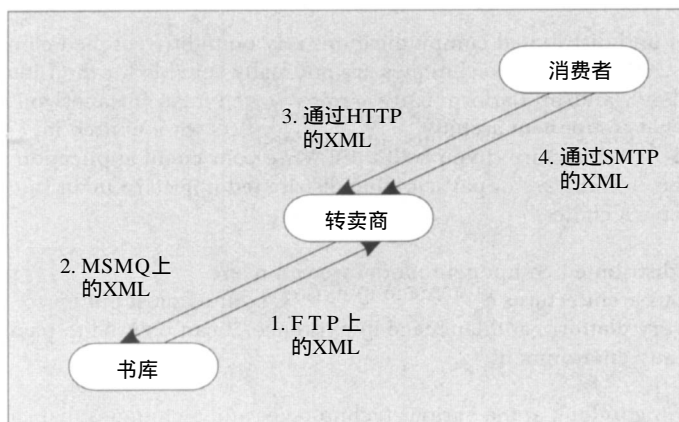


图 11-2

### 11.1.1 基于FTP的XML传送

在书库中有一个常规性的处理过程即送 XML文档到零售商，此 XML文档包括数据库中所有最近出版的书籍的标题信息，例如目录、主题、内容摘要和价格等。显然，经过一段时间将会有成千上万的标题信息，因此 XML文档将变得很大。这就决定了通过 FTP传送这些文档是最有效的一种方式，因为这种方式适合于成块的数据传送。

这是一个后台自动、异步的过程，同时，我们必须在零售商的目的服务器运行一个进程，它对到达的XML文件扫描目标目录，并且进行适当的工作，如索引某条消息，并将它插入到商务书籍数据库中。

图11-3将显示该工作过程。

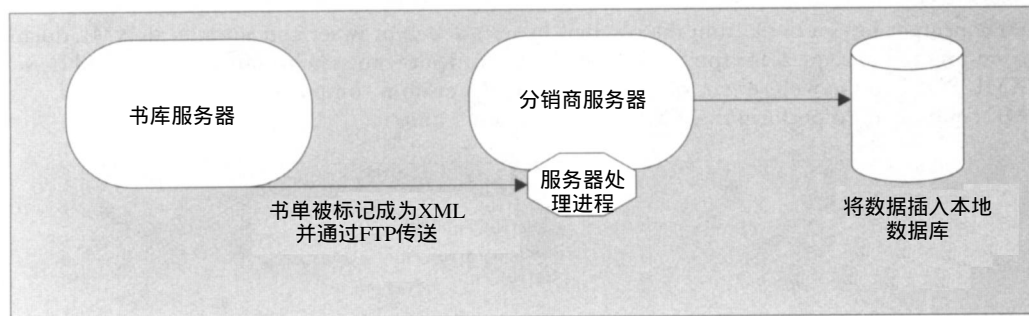


图 11-3

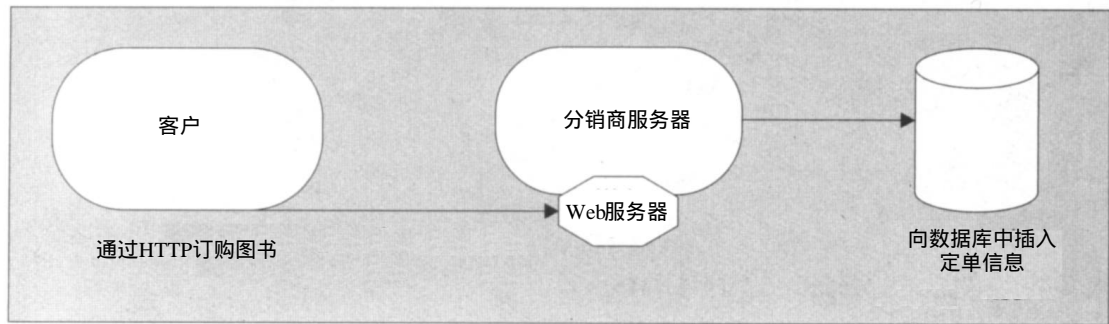
### 11.1.2 XML和消息队列

零售商订购相关所需书籍的过程也是异步的，不过比起通过 FTP来传送XML文档，这个过

The diagram illustrates the initial step of the order processing flow. On the left, a rounded rectangle represents the '分销商服务器' (Distributor Server). An arrow points from this server to a cylinder representing the '书库服务器' (Library Server). Below the arrow, the text '将定单放入队列' (Put the order in the queue) is written. To the right of the library server, the text '进程订单' (Process order) is displayed.

订单消息并不是直接送至服务器处理，而是被排成一个消息队列并迟一些时间再处理。订单在XML中被传递，该XML将从使用标记订购书籍的XML文档中得到，这个文档就是此前已经通过FTP传出的。

当一个消费者从零售商处购买一本书时，他使用网络浏览器并提交一个关于该书的 XML 文档，这个请求将被同步处理，并且能作为一个 MSXML 被呈送到 Web 服务器。或者，如果我们开发一个定制的组件，就能在其中使用 MSXML 组件去实现 XML HTTP 请求（参见图 11-5）。



#### 11.1.4 基于SMTP的XML

消费者订购图书能够采用的最后一种方式是通过 SMTP (e-mail)。其中, 一个 XML 文档将被传送, 必须符合以下两点, 即遵照一个有效的框架, 并有一个检查特定的 POP3 邮箱的过程, 这个邮箱是专供有效的 XML 书籍定单使用, 这些定单是参照订购计划制订的, 在消费者与零售商之间的订购允许异步 (参见图 11-6)。

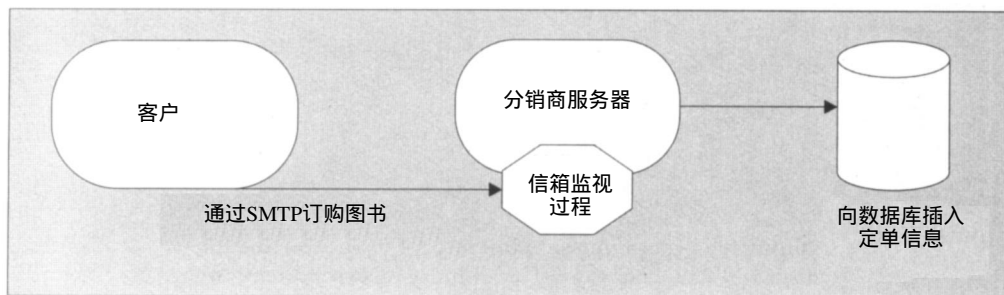


图 11-6

## 11.2 作为一种分布式组件模式的XML

前面已经演示，XML作为一种消息在系统间传送有很多种方式，下面我们将关注针对 XML 消息传送而出现的一些流行的方式和标准。

### 11.2.1 消息传送和串行化

典型的XML文档被构造成描述性结构、人可读数据，并可对此进行一些处理。我们可以将此概念扩展到机读数据，并使用数据串行化按照一个明确的标准来传送消息。这样既减少了平台的依赖性，又通过使用代码简化了操作。

消息传送包括一个称作串行化的过程和它的反转过程即逆串行化。通过串行化，一个数据条目被构造成简单的一维数组在线传送，例如数据“Hello, World!”能够被串行化为这种格式：

```
<string>Hello, World!</string>
```

同样一个包含有数据的记录集如表 11-1所示。

表 11-1

列 名	值
FirstName	Steven
LastName	Livingstone
City	Glasgow

它的串行化如下所示。

程序清单 11-1

```
<recordset>
  <column>
    <columnName>
      FirstName
    </columnName>
    <columnValue>
      Steven
    </columnValue>
  </column>
```

```
<column>
  <columnName>
    LastName
  </columnName>
  <columnValue>
    Livingstone
  </columnValue>
</column>
<column>
  <columnName>
    City
  </columnName>
  <columnValue>
    Glasgow
  </columnValue>
</column>
</recordset>
```

一旦我们使数据串行化，就能够使它作为一种简单的文本在不同的系统中传送，在目的地又使用逆串行化使之恢复到合乎需要的系统格式。有许多协议能够用于传送串行数据，最流行的是HTTP、FTP和SMTP（该协议用于传送e-mail）。

### 1. 使用XML-RPC和SOAP的串行化

这一章里的主要焦点是XML-RPC（XML远程过程调用），我们将看到它将串行化一些简单和比较复杂的数据形成XML并在Internet上传送。执行上它将允许许多语言类型被串行化（或逆串行化），并有一个广阔的范围，非常适合Internet。SOAP即简单对象访问协议，是一个更复杂和更成熟的XML-RPC，我们将在后面作一些描述，二者都是完全开放的（在编写本书时），并且可以被许多平台所运用。

XML-RPC可能是最流行的、最有用的基于XML的分布式通信协议。SOAP在较长时间后可能成为其继任者——对更为复杂的系统尤其如此。

### 2. Coins（XML和Java的混合）

由于有对Java串行化的依赖，一些人并不认为Javabeans是在Java中写组件的最好方法。Coins混合了XML和Java，并用XML代替了串行化，并且不保存对象的状态。它的目标是使程序之间的消息传送变得简单。

而且，由于XML是简单自然语言（它没有Javabeans那么严格），使得调试过程得到了改进。在处理过程中，也少了许多限制——这是由于在XML格式里，数据能够很容易地在不同应用程序中改变。如果参考外部组件，那么Coins也能够再还原为XML文档，这是因为Coins支持独立文档中元素之间的链接（相反，传统的Javabeans并不支持超链接且很少支持组件划分）。

Coins需要在XML文件与Java对象间建立密切的联系，而XML-RPC在XML中是一个静态的接口。这就导致了交换消息的程序之间的依赖性。数据元素能够通过一个更新程序被加入，而不需要对另一个程序同步更新。

关于Coins的进一步信息可参考此网站：<http://www.jxml.com/coins/index.htm/>。

### 3. WDDX的串行化过程

基于以上二者之间的是WDDX，即分布式Web数据交换，它是在程序之间交换复杂数据结

构的一种技术。它基本上是语言独立的 XML 数据的描述标准，它还包括一套串行化 / 逆串行化的模块。模块被 Allaire 所发展。目前，它已经被 JavaScript 1.x、ColdFusion 4.0、COM、Perl 和 Java 所包含。WDDX 使用 XML 描述分布式对象，但并不像 XML-RPC 那样激活一个远程调用。

可以从 <http://www.wddx.org/> 网站中下载 WDDX DK 并能得到关于 WDDX 的更进一步的描述。

WDDX 的应用程序将通过串行化把对本地对象的描述打包，然后传送给远程服务器，在那里，一个应用程序将使用逆串行化过程解包，并产生有用的相关信息。

WDDX 并不关心数据如何被发送，只关心数据已被提交到一个网页，而该网页能够从请求对象中访问被串行化的 XML，一个包含有所需书籍标题名字的简单 WDDX 数据包，如下例所示：

程序清单 11-2

```
<!DOCTYPE wddxPacket SYSTEM "wddx.dtd">
<wddxPacket version='0.9'>
  <header/>
  <data>
    <var name="title">
      <string>Professional XML</string>
    </var>
  </data>
</wddxPacket>
```

我们能够使用脚本并通过消息的串行化创建数据包，下面就是一个 Perl 的例子：

Perl WDDX 库在可下载的 SDK 中。

程序清单 11-3

```
# Include WDDX library
use wddx;

# Include HTML-Encoding functions
use HTML::Entities;

# Set the $Title variable to the name of the book
$title = "Professional XML";

# Create a new serializer "object"
$serObj = new wddx();

# Serialize the $Title variable into a WDDX Packet
$bookPacket = $serObj->cfwddx
(
  "action" => 'perl2wddx',
  "input"  => {"string" => \" $title, }
);
```

下面是一个 JavaScript 的例子，功能和上面的例子相同：



## 程序清单 11-4

```
<!-- Include JavaScript / WDDX functionality -->
<SCRIPT LANGUAGE="JavaScript" SRC="Wddx.js"></SCRIPT>

<!-- Create a custom function that serializes our message ---->
<SCRIPT LANGUAGE="JavaScript">
    function SerializeMsg()
    {
        var Book = new Object;
        Book.title = "Professional XML"

        // Create a new serializer "object"
        SerObj = new WddxSerializer;

        // Serialize the Message variable into a WDDX Packet
        BookPacket = SerObj.serialize(Book);

        // Return the new WDDX packet
        return BookPacket;
    }
</SCRIPT>
```

然后，我们将这个已串行化的 WDDX消息存储在一个 HIDDEN元素中，而这个元素允许消息从一个表单区提交来——我们称这个 HIDDEN区域为 WddxContent。当内容被提交到 WddxContent后，将使用逆串行化过程得到原始消息，使用 WDDX逆串行化对象去进行此项工作。

## 程序清单 11-5

```
<%@ LANGUAGE="Javascript" %>
<%
    strWDDXPacket = Request.Form("WDDXContent");

    // Create a new serializer "object"
    var ObjDeser = Server.CreateObject("WDDX.Deserializer.1");

    // Serialize the Message variable into a WDDX Packet
    Book = ObjDeser.deserialize(strWDDXPacket);

    //writes out our book title
    Response.write(Book.getProp("Title"));
%>
```

WDDX是一个非常用的规范，并能同许多语言协同工作。实际上，它已经被 Allaire（它的发明者）用于一些产品的制造，但是，比起 XML-RPC和SOAP来，它是一个较为封闭的规范，并与这些规范的要求不一样。同时，它比起 SOAP来有较多限制，但对于简单分布式通信而言它是胜任的。

#### 4. XMOP的串行化过程

XML Metadata Object Persistence即XML元数据对象一致性，它的目标是使如 COM、Java和 CORBA等技术能交互运行，而这是通过提供形式对象串行化的机制来实现，这种机制与特定的系统对象并无联系。因此对象能够在 COM和Java之间配置，甚至能够在不同的 Java的虚拟机上

配置。

值得注意的是XMOP是XML-RPC或者SOAP的补充（例如在SOAP方法调用中作为一种界面参数串行化的方式）。

XMOP使用SODL（简单对象定义语言）（SODL可参见<http://jabr.ne.mediaone.net/documents/sodl.htm>）即XML IDL DTD，它允许在COM和CORBA中通过使用IDL使得对对象描述保持兼容性，XMOP实际上使用SODL的DTD 1.0版本。

更多的信息请访问：<http://jabr.ne.mediaone.net/documents/xmop.htm>。

## 5. KOALA的串行化

另一方面，KOML（Koala对象标记语言）是一种Java解决方案，它描述如何串行化和逆串行化一个XML文档中的Java对象，并遵循XML 1.0的标准和SAX 1.0标准。一些人可能发现它是有用的，因为它是一个100%的Java解决方案，但是因为它仅能服务于Java对象，这就限制了它的使用，在Internet上更是如此。

Koala方案使用JavaSerializable接口类实现集成。

一个Koala可串行化对象必须实现接口java.io.Serializable，因此用Java描述的一个叫做Book的类有下面的代码：

程序清单 11-6

```
public class Book implements Serializable
{
    int id = 1642;
}
```

在KOML中将被表现为如下的XML文档：

程序清单 11-7

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE koml SYSTEM "http://www.inria.fr/koala/XML/koml12.dtd">
<koml version='1.2'>
  <classes>
    <class name='Book' uid='-5510978188925784084'>
      <field name='i' type='int'/>
    </class>
  </classes>
  <object class='Book' id='i1'>
    <value type='int' name='id'>1642</value>
  </object>
</koml>
```

一个KOML文档的命名空间将指向URL <http://www.inria.fr/koala/XML/koml11.dtd>，关于KOML的进一步信息，能在如下站点找到：<http://www.inria.fr/koala/XML/serialization/>。

### 11.2.2 紧耦合系统和松耦合系统

下面我们将讨论XML所提供的松耦合系统的优点和缺点，松耦合系统是传统的紧耦合系



统比较而言的。当今的许多应用程序仍提供紧耦合系统，它要求理解在其中传递的消息格式，例如是否这些消息使用了 DCOM 或 CORBA (IIOP)。另一方面，松散耦合系统并不需要系统间如此严格的理解，至于 XML，它仅是一个在其间传送的简单文本。

### 1. 松耦合系统的缺点

对于一个分布式组件解决方案来讲，使用松耦合的 XML 的两个最大可能的缺点是带宽和速度。例如，尽管 XML 消息的创建是简单的，但是在它们被传送到应用程序前必须通过一个 XML 解析器而使之有效，这个过程是非常重要的。同时，将一个标记文本加到数据中去组成一个 XML 文档也是很重要的，通过传送增加了网络的通信量（尽管使用压缩技术可以减少网络的通信量）。

紧耦合系统了解其间传输的消息的格式。与 XML 消息比较而言，它能节省网络成本，因为 XML 的消息必须经过解析。当然，传统的紧耦合系统经常需要“握手”，并且如果服务器升级为另一个新的格式，将很容易导致这种“握手”失败。

### 2. 松散耦合系统的优点

事实上，基于 XML 的组件框架是能够被升级的，多种方法的调用能够被打包在一个请求里，这与传统的客户-服务器模式相反，在这种模式里，每个被调用的方法都必须产生一个请求。

图 11-7 显示了传统的远程方法调用的过程：



图 11-7

图 11-8 将显示远程方法调用的 XML 过程：

对于一个非常可靠的应用程序而言，它要求一系列的调用被处理成一个事务。这个技术保证了调用在一种“都做或者都不做”的方式下被完成，如果应用程序希望使用事务，并且你使用 XML-RPC 协议，那么你将不得不在应用程序中加入事务处理的能力。但如果需要的话，更新的 SOAP（和一些商务的 XML 服务器产品）有能力去实现基于事务的应用程序。

在一个分布式应用程序中，能够在不同的服务器间只使用一种版本控制。因为每一个 XML “文档”都能够被合法化为一个合适的 DTD 或者模式，这样能扩充功能和减少冲突。在一个服务器发现冲突时，将会执行一个良好的错误处理程序。只要 XML 标记没有被删除，所有的功能能够有效地保持在被有效隔离的分布式组件间。事实上，随着传送语言使用 XML，只要包括必要

的数据，客户结构甚至并不必是一种固定的 XML 格式。

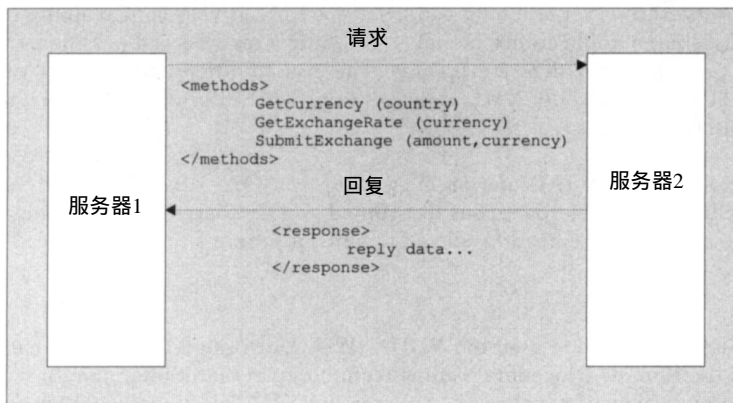


图 11-8

松耦合系统中，从相互发送消息的不同协议规范中创建文件是相对简单的。例如，我们要讨论使用 XML-RPC 和 SOAP 协议的 XML 分布式通信。这些系统是基于不同的 XML 标准的，但是一些方法如 XSLT 在 XML 文档的特定版本的不同规范之间不能被用于“翻译”，这是毫无理由的（事实上，这正在被讨论）。

### 11.2.3 通信方式

使用 XML 进行相互通信的服务器能够使用许多协议和标准，在分布式组件间建立彼此联系的桥梁。最流行的是 HTTP，因为它是使用中最普通的标准，尽管 SMTP 和 FTP 与之非常相似，并能提供独立传送 XML 消息的平台。但是，更多的比较特殊的网络接口，例如：CORBA/IIOP，Java RMI (Remote Method Invocation) 和 Microsoft DCOM/COM 能够被用于一些动态 XML 应用程序服务器。

这一部分将关注 HTTP 作为分布式组件的通信协议，因为它是可用的最简单和最独立的协议。我们在最后也将讨论其他一些可用的协议，这些协议能够实现一些可变化的传输方法。

#### 1. 远程调用

使用 XML 的服务器到服务器通信的首要方法与传统的调用分布式应用程序的方法非常相近，即通过调用一个方法并传送参数到该方法，而该方法提供一个返回值，细微的不同在于所有这些调用都在 HTTP 上运行，并仅使用 XML 消息去提供调用远程网络服务器那个方法的信息。

在本节的后面，我们将关注 XML-RPC 和 SOAP，SOAP 提供功能性的描述，它对于异构的分布式应用程序是非常有效的——它也可能成为服务器通信发展过程中的亮点。

#### 2. 数据交换

数据交换是一种用于 WDDX (Web Distributed Data Exchange, Web 分布式数据交换) 的方法，它与远程过程调用的不同在于它并不说明特定的协议，且仅仅对创建一个关于数据的基于 XML 的消息，数据交换并不像分布式机制将方法和参数封装起来。纯粹的数据交换是一个主动的过程，在此过程中，远程服务器的某一个特定页被激活，并且数据以 XML 形式被发送到该页。

另外一些方法允许通过COM、CORBA、SMTP和另外一些别的传送方式支持数据交换。

### 11.3 XML-RPC

XML-RPC即XML远程过程调用，是一个相对较新的方法，这种方法激活在分布式机器上的方法并使消息返回。它使用XML来传送结构化的消息，而其中封装的功能调用的执行依赖于远程系统，这样我们就能够使远程系统与本地系统结合起来。事实上，当XML-RPC运行在一个纯粹的HTTP上，并使用XML（格式文本）来传送消息时，全部的标准是语言独立的。

你可能对平台的独立性产生怀疑，但这个疑虑将很快被打消，即如果考虑到HTTP和XML这两个主要的组件是简单而有效的标准，并被全部的行业所接受，且在所有的平台上运行良好这个事实。请相信，虽然XML-RPC是简单的，但是它将是在未来的20年中XML技术领域中最有效的一个。

目前的XML-RPC标准（参见<http://www.xmlrpc.com>），允许我们得到一个对于特定方法的返回值，而该方法使用了一套由远程服务器指定的参数。我们能够直接从一客户或者从一个服务器使用XML-RPC而调用在远程服务器上的方法。在关于服务器通信的讨论中，我们将看到更多此类情况。

在XML-RPC世界中，访问一些站点将会使你收获颇丰，例如<http://www.mailtothefuture.com>，它允许你使用一个远程服务器的界面在未来的某个日期发送一个e-mail给你自己，这有点像“提醒”系统，你能够远程地加入一条新消息，删除一条以前的消息，或者像一些其他方法一样，在使用者的队列中得到一些消息。所有这些都可以在浏览这些站点和不使用它们的接口下被实现。图11-9将显示这些工作：

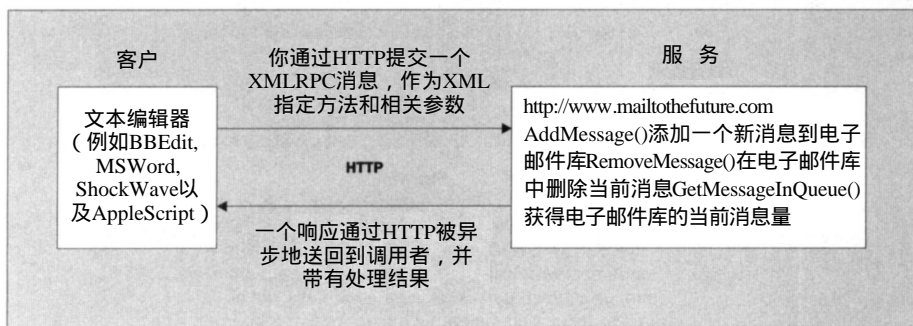


图 11-9

近来，我们在一些平台，例如Frontier、AppleMacs、Unix和Windows等上已经看到允许使用者在远程讨论组上加入和修改一条消息。这能够通过网络浏览器或者客户应用程序来实现（例如Microsoft Word）。在这样的界面里，你能够创建一个并不基于浏览器的工作站编辑器，它允许使用文本编辑器去创建或者编辑消息，而该文本编辑器比嵌在浏览器里的编辑器更为先进。

XML-RPC允许你向讨论论坛粘贴消息，编辑已经存在于论坛里的消息，保存自己的已经存在的消息。这些工作能够在任何系统里完成，只要该系统支持HTTP协议和各种文本编辑器。

可能性每周都在增加，在<http://www.xmlrpc.com>/网站，Dave Winer正不断地把它推向前进。

### 11.3.1 为什么使用XML-RPC

传统的远程通信方法包括 COM和CORBA技术（例如DCOM或IIOP），它们仍然被发展着的通信技术所支持，尽管这些技术在紧密集成的系统里毫无疑问是流行的，但在分布式通信领域仍然强调传统的方法。与之相反的是，XML-RPC系统被广泛的分布式网络所更多地采纳。例如，XML系统并不要求在一个方法被调用前必须知道服务器的标识，相反，服务器能够在一个调用里被标识，或者在使用一个动态的负载平衡后被决定。

传统上，我们必须知道相关的许多技术问题，包括平台形式，它们使用了COM还是CORBA，但如果它们的系统在防火墙之后（许多系统有这样的系统），另一些问题也将被考虑。

图11-10将解释在防火墙问题上，XML-RPC为什么没有问题，而传统的方法将遇到困难，特别在Internet中。

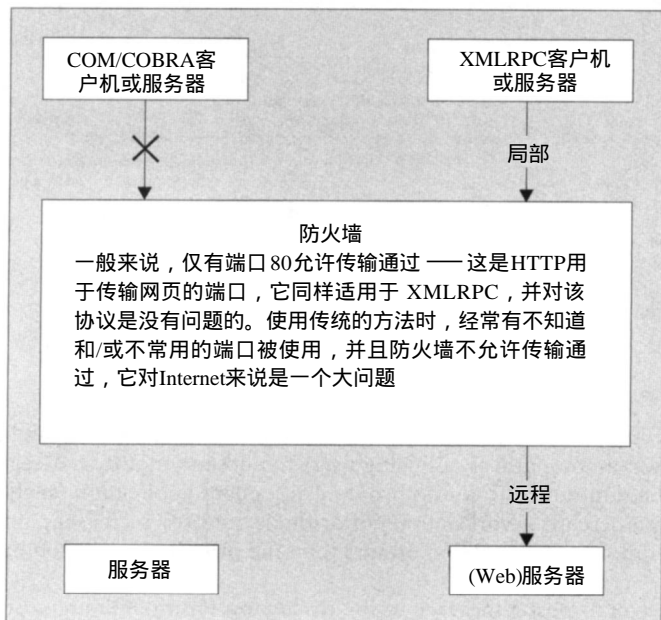


图 11-10

使用XML-RPC协议，所有的通信都是在 HTTP上工作的，因此，消息传送通过防火墙毫无问题（几乎所有的防火墙都允许 HTTP通过），进而，随着 XML对界面以及对数据的描述，与传统的比较固定的系统相比，将会减少很多的集成问题。事实上，所有我们需要的是 URL、界面描述和期望的返回值——这是个简单的问题。

在XML-RPC.com站点最近的一些工作可能是最好的解释。Userland.com站点（运行在Frontier上）展示了一些修改讨论组消息的方法。这些允许开发者远程调用一些方法去执行某些动作，如在讨论板上创建一个新的消息和修改一个已经存在的消息——所有这些都从任何一个远程客户系统通过XML-RPC来实现，而该客户系统是支持HTTP协议的。

Userland对每一个相对较短的文件提供一个方法名称，它的期望参数，它的返回值和对该



方法的一个简短描述，两天之内，人们就可以在平台上例如 Macs，Windows 和 Unix 拥有一个包括多种编程语言的编辑器，这些语言包括 ASP，COM，PHP，Lingo，Java 和 Applescript 等。

这一切都毫无问题。事实上，从 Dave Winer 发来的一封 e-mail 十分令人感兴趣，其大意是他想到一个问题，因为在讨论组标准发布 24 小时后没有一条“成功”的消息被提交。我怀疑是否每一位读者都在综合了多个平台和传统技术领域里使用的语言后，又考虑 24 小时长的时间。XML-RPC 的强大功能是毫无疑问的。

最重要的是要记住 XML-RPC 对于分布式计算并不总是正确的选择，它比传统的方法要慢，传统的方法由于上述的理由被最普遍地用于封闭式网络。但是，Internet 比绝大多数的封闭式网络要慢，因此在使用 XML-RPC 时并不真感受到等待时间的问题（因为这是网络延迟，而不是导致绝大多数延迟的处理时间），如果你在一个相同的共用网络里调用一个方法并对速度更看重，并且也不介意将应用程序界面暴露给远程服务器，最好的选择可能是使用 COM 或者 CORBA 技术，例如 DCOM 或 IIOP。像你所见到的那样，当在一个网络服务器上使用 HTTP 打包并通过 XML-RPC 传送时，将仍然会暴露一些在 COM/CORBA 应用程序中使用的方法。

图 11-11 将显示一个使用 XML-RPC 的分布式系统的可能的结构，这个图表将解释任何一个 XML-RPC 授权的客户如何调用一个在 XML-RPC 服务器上的方法，而这些服务器又是在不同的平台上并运行不同的 XML-RPC 服务程序，可能有许多可能的结构。能够看到服务器能够在另外的远程服务器上激活一个方法，并向客户应用程序提供服务。事实上，一个客户激活一个调用和一个服务器激活该调用二者并没有区别。

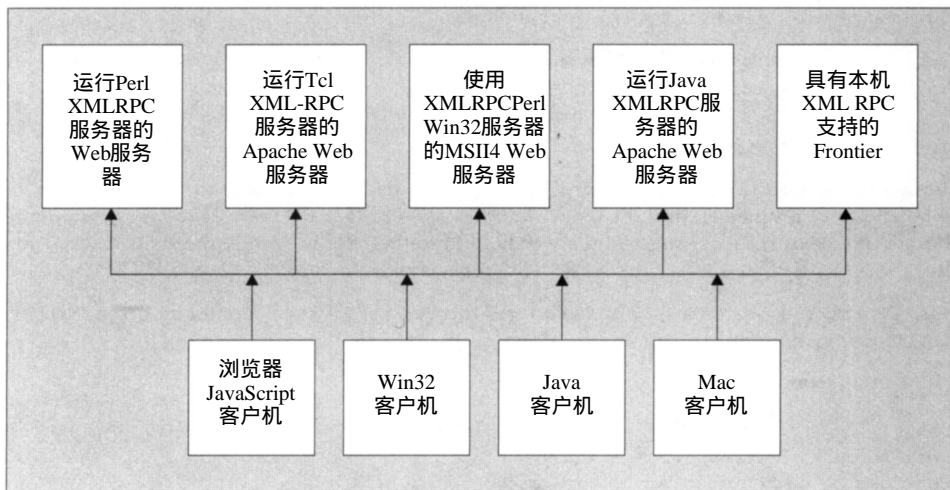


图 11-11

### 11.3.2 XML-RPC 适用于何处

前面几章已经描述了 XML 和它的一些相关技术，因此当你听到这样一些事情时或许会感到

惊讶，即 XML-RPC 目前仅仅使用了最基本的一些功能，即 XML 1.0 标准和 XML 文档对象模型 (XML Document Object Model.)

这些 XML 的一些基本特征在实际中几乎被每一个平台所支持，也保证了 XML-RPC 规范能够被广泛地支持。将来，我们将看到另外一些 XML 特征加入到 XML-RPC 规范。

### 11.3.3 XML-RPC 规范——技术全瞻

XML-RPC 的实现是基于 HTTP POST 方法的，消息体使用 XML 标记，其包含着在远程服务器上执行的方法和这些方法所使用的全部参数。返回的响应也是基于 XML 的。

一个过程的参数和返回值也可以是简单的数据形式，例如：逻辑型、整型、字符串、浮点型和日期型，也可以是较复杂的数据结构，例如通过数组和结构实现的复杂的记录和列表结构，事实上，二进制数同样能被 Base64 编码表示，并且允许，例如，对于一个方法调用返回一个图像。

Base64 编码用于将二进制数据转换成 ASCII 文本，由于通过 SMTP 服务器和另外一些文本协议传送，所以没有数据被误解。Base64 编码也是通过 XML-RPC 传送二进制编码消息的标准，例如 GIF 和 JPEG。在本章最后，将做更多的描述。

这个标准的另一个重要部分是必须有一个返回值，它可能是一个结构，并由许多结果值组成。

该标准包含了 HTTP 报头要求、XML-RPC 请求、有效负载格式、XML-RPC 响应和错误处理。

#### 1. 报头要求

URI (Universal Resource Identifier) 格式中报头的第一行是不指定的，如果服务器仅处理 XML-RPC 调用，它能为空或者是一个单斜线。但是如果服务器处理一个混合的 HTTP 请求，那么我们将使用 URI 去指定指向代码的路径，该代码处理 XML-RPC 请求。XML-RPC 请求必须是通过 HTTP 标准的 POST 方法。

一个用户代理 (User-Agent) 和主机 (Host) 必须被指定，内容形式 (Content-Type) 将是 text/xml。最后，正确的内容长度 (Content-length) 应被给出。

让我们看一个实现报头要求的例子：

程序清单 11-8

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181
```

从运行在 Windows NT 上的 Frontier 客户调用在 userland.com 中名叫 betty 的服务器，上面这个例子指定了 RPC2 作为 XML-RPC 请求的响应。Content-Type (内容形式) 被正确的指定为 text/xml，Content-length (内容长度) 为 181，这是一个合法的 XML-RPC 请求。

#### 2. 请求格式



该请求指定了结构为XML并且使用XML标准1.0版。全部的请求应包括在一个<methodCall>元素中，它必须包含一个子元素<methodName>，该子元素包含了一个字符串，它描述了一个将被激活的远程方法的名字。对于服务器，它正确地标明了一个方法的名字，合法字符为 A~Z、0~9下划线、逗号和斜线（XML经常对第一个字符加以限制）。

如果这个方法含有参数，<methodCall>元素必须包含一个<params>元素，该元素依次可以包含一个或者多个<param>元素，每一个均有一个<value>主题，它们的顺序被该方法接口所定义，下面是一个请求的模板：

程序清单 11-9

```
<?xml version="1.0"?>
<methodCall>
  <methodName>mymethodname</methodName>
  <params>
    <param>
      <value><string>myParameter</string></value>
    </param>
  </params>
</methodCall>
```

### 3. 参数指定

被指定作为参数的值被分为标量、数组和结构，而标量形式有：

- 整型。
- 逻辑型。
- 字符串型。
- 双精度型。
- 日期/时间型。
- Base 64。

我们将依次介绍各种形式。

#### (1) 整型

在XML-RPC中的整型是一个四字节有符号整数，由<i4>或<int>元素来指明，0只有一位，并且不允许有空白，例如：

程序清单 11-10

```
<?xml version="1.0"?>
<methodCall>
  <methodName>GetCelcius</methodName>
  <params>
    <param>
      <value><int>-5</int></value>
    </param>
  </params>
</methodCall>
```

你可能发现一个奇怪的现象：数据形式没有表示成一种属性，可能在普通的XML-RPC规

范里缺乏属性。这只是因为标准的作者并没有使用。

属性真的是比元素更有用处吗？在XML-RPC领域里有许多相关的讨论，但是到目前为止并没有动议使用它们。

## (2) 布尔型（逻辑型）

布尔型在<boolean>元素里被指明，“错误”用“0”表示，“正确”用“1”表示，例如：

程序清单 11-11

```
<?xml version="1.0"?>
<methodCall>
  <methodName>SetProductAvailability</methodName>
  <params>
    <param>
      <value><boolean>1</boolean></value>
    </param>
  </params>
</methodCall>
```

## (3) 字符串型

字符串被封闭在一个<string>元素里，必须由合法的ASCII值组成，但是由于XML-RPC必须是合法的XML特殊字符，所以例如“<”和“&”必须被编码成“&lt”和“&amp”。目前在XML-RPC里编码没有被指定，因此解析器将使用默认的编码。在XML-RPC中，编码类形并不需要被指定，尽管这是一个在XML-RPC中被多次讨论的问题。因此，XML默认的UTF-8编码将被使用，它对于绝大多数的消息都是有效的，下面是一个典型的例子：

程序清单 11-12

```
<?xml version="1.0"?>
<methodCall>
  <methodName>GetCompanyNumber</methodName>
  <params>
    <param>
      <value><string>Wrox Press</string></value>
    </param>
  </params>
</methodCall>
```

## (4) 双精度型

双精度型在XML中定义为双精度带符号浮点数，在<double>元素里被指明。双精度型不能包含空格，它包含一个正号或负号，一连串的数字符号，一个小数点和任意位小数（范围取决于使用情况，并没有被指定），该数据类型的使用情况如下面这个例子：

程序清单 11-13

```
<?xml version="1.0"?>
<methodCall>
  <methodName>SetMyCurrentBankBalance</methodName>
  <params>
    <param>
      <value><double>-23435.87</double></value>
```

```
        </param>
    </params>
</methodCall>
```

#### (5) 日期/时间型

日期和时间在<dateTime.iso8601>中被指明，是iso8601标准的一个子集，该类型对于任何Internet商务系统都是非常重要的，它的格式是：YYYYMMDDTHH24:MM:SS，通过XML-RPC传送特定的日期和时间消息是很容易的。有关时区的消息将被服务器或文档所指定，在该元素里并无描述。下面是一个例子：

程序清单 11-14

```
<?xml version="1.0"?>
<methodCall>
  <methodName>GetInventoryLevel</methodName>
  <params>
    <param>
      <value>
        <dateTime.iso8601>19990912T13:54:12</dateTime.iso8601>
      </value>
    </param>
  </params>
</methodCall>
```

#### (6) 64进制

我们能通过XML-RPC使用Base 64标准来传送二进制编码消息，例如 GIF和JPEG，它必须被指定为二进制文件（如下面这个例子）。<base64>元素用于指定编码数据，该例没有使用真正的编码图像。

程序清单 11-15

```
<?xml version="1.0"?>
<methodCall>
  <methodName>InsertCompanySharesGraph</methodName>
  <params>
    <param>
      <value>
        <base64>Wk964jkf0skamllp970kmk</base64>
      </value>
    </param>
  </params>
</methodCall>
```

#### (7) 数组

在XML-RPC中，一个<array>包含唯一的<data>元素，而该元素又包含一些内含参数的<value>元素。不同的类型在一个数组中能够被混合。如果一个<array>元素包含一个数组或结构，它们甚至能被递归。数组适用于两种情况：一是将传送的数据（例如关于一个对象的状态消息）由一系列的参数所组成，而这些参数在设计时并不知道；二是参数很多，而你希望以数组形式传送它们而不是采取分散的个体形式，下面是一个例子：

## 程序清单 11-16

```

<?xml version="1.0"?>
<methodCall>
  <methodName>InsertCompanyInfo</methodName>
  <params>
    <param>
      <array>
        <data>
          <value><string>Deltabiz Inc</string></value>
          <value><int>8878</int></value>
          <value><boolean>1</boolean></value>
          <value>
            <array>
              <data>
                <value><int>674</int></value>
                <value><string>NoWhere Rd</string></value>
                <value><string>Glasgow</string></value>
              </data>
            </array>
          </value>
        </data>
      </array>
    </param>
  </params>
</methodCall>

```

## (8) 结构

结构允许你对于值的顺序采取一些智能的办法，它不像数组，数据有一个简单的顺序，结构在<struct>元素中指定，它包含一个或几个<member>元素（并无数量限制）。其中，依次序有<name>和<value>元素保存数据，像数组一样，结构是可递归的，并且也能包含数组，下面是一个有关XML-RPC结构的例子：

## 程序清单 11-17

```

<?xml version="1.0"?>
<methodCall>
  <methodName>CreatePurchase</methodName>
  <params>
    <param>
      <struct>
        <member>
          <name>Product</name>
          <value><string>Silver Clock</string></value>
        </member>
        <member>
          <name>Cost</name>
          <value><i4>87</i4></value>
        </member>
        <member>
          <name>Purchase Date</name>
          <value>
            <dateTime.iso8601>19990912T02:53:02</dateTime.iso8601>
          </value>
        </member>
      </struct>
    </param>
  </params>
</methodCall>

```

```

    <member>
      <name>Purchase Order</name>
      <value><double>384793</double></value>
    </member>
  </struct>
</param>
</params>
</methodCall>

```

看了上面这些代码，你可能想到 XML 方法能够减少许多冗余，如：

程序清单 11-18

```

<struct>
  <Product><string>Silver Clock</string></Product>
  <Cost><i4>87</i4></Cost>
  ...
</struct>

```

对于 XML 规范，有许多理由使之没有以这样的方式进行定义。其中之一是 XML-RPC 结构定义的方式与传统编程语言里的结构（或者目录对象）十分相似，在传统的编程语言里我们有明确的名字/值对。

#### 4. 响应的报头

响应的第一行将是一个合法的 200 OK 的 HTTP 的报头（除非服务器自身有一个低级错误），下面是 text/xml 的 Content-Type（内容形式）和一个正确的 Content-length（内容长度），下面是一个例子（Date 和 Server 是随意的）：

程序清单 11-19

```

HTTP/1.0 200 OK
Connection: close
Content-length: 342
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: Userland Frontier/5.1.2-WinNT

```

#### 5. 响应的格式

响应的有效负载必须有一个结构。结构中，<methodResponse> 元素能够包含一个唯一的 <params> 元素，该元素又封装一个唯一的 <param> 元素，该 <param> 元素又包含唯一的 <value> 元素。

程序清单 11-20

```

<?xml version="1.0"?>
<methodResponse>
  <params>
    <value>
      <array>
        <data>
          <value><string>www.deltabiz.com</string></value>
          <value><double>-76570</double></value>
        </data>
      </array>
    </value>
  </params>
</methodResponse>

```

```
        </array>
      </value>
    </params>
  </methodResponse>
```

如果有错误，响应也能包含出错信息，我们看一下这个方面的有关问题。

#### 6. 出错响应

既然可以进行成功响应，那响应也能够返回一个出错消息。在 `<method Response>` 元素里包含一个 `<fault>` 元素，该 `<fault>` 元素又包含一个 `<value>` 元素，`<value>` 元素包含一个含有两个数据的 `<struct>` 元素，一个用 `<int>` 值调用错误代码（`faultCode`）；另一个字符串叫做 `faultString` 来指明出错理由。

下面是一个例子：

程序清单 11-21

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>873</int></value>
        </member>
        <member>
          <name>faultString</name>
          <value><string>An error message</string></value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

### 11.3.4 XML-RPC的实现

XML-RPC被分成两个组件，叫做客户和服务，客户独立于服务器。因此一个运行在 Windows NT 服务器上的 COM 客户能够调用在 Unix Apache 服务器上的 Perl 编写的方法。

XML-RPC 可通过 Python、Java、Perl、Tcl、ASP、COM 和 PHP 实现。对于每一种方式的进一步描述参见 XML-RPC 部分的“到哪里去”一节。

#### 1. 一个简单的例子

下面这个例子将对以上某些语言加以说明并显示如何调用一个远程的方法，该方法返回了储蓄账户的余额。该例将分成“XML 请求”、“客户的实现”、“XML 响应”、“服务器的实现”等几个方面。

这个例子是基于 ASP 和 MS IIS 的，虽然能使用另外的技术编写服务器并使用上述的客户机制进行调用——只要接口定义保持不变，就没有问题。就是一个漂亮的 XML-RPC。

在开始这个例子之前，在 Web 服务器上，把所有的例子文件放到目录 `<root>/xmlrpc/client/` 下，并且从指定的 URL 上下载 ASP 客户/服务器文件（URL 在下面的“到哪里去”部分给出），其中要



保证xmlrpc.asp文件也在目录<root>/xmlrpc/client/下。

同样，在同一个URL下，使用RegSvr32实用程序注册COM组件。保证安装指令（装载IE5等）使你能成功地装载RPC工具。

注意注册这个组件，打开一个命令行窗口，输入

```
RegSvr32 <component name>.dll
```

将会弹出一个消息窗口告知这个组件已被成功注册。

#### (1) XML请求

对于该例，下面这段代码将描述了 XML-RPC的请求，在此我们将调用一个称为 GetCurrentBalance ( ) 的本地方法，该方法接受一个账号作为参数。

程序清单 11-22

```
<?xml version="1.0"?>
<methodCall>
  <methodName>GetCurrentBalance</methodName>
  <params>
    <param>
      <value>
        <double>873214</double>
      </value>
    </param>
  </params>
</methodCall>
```

我们看到希望调用的方法称为 GetCurrentBalance ( )，并将账号定义为双精度型。

在目标平台上使用 XML-RPC方法，开发商仅仅需要提供参数和方法。实际的XML代码将被扩展成起始方法调用的一部分，HTTP的POST方法将用来代表用户。让我们看一看怎样通过使用四种语言即ASP、COM(VB)、Java和PHP实现该功能的。

如果想按自己的方法试验一个例子，可以参见“到哪里去”一节，并下载合适的 XML-RPC工具。

#### (2) ASP

程序清单 11-23

```
<!--#include file="xmlrpc.asp" -->
<%
  ReDim paramList(1)
  paramList(0)=873214

  myresp = xmlRPC("http://localhost /xmlrpc/server.asp", _
    "GetCurrentBalance", paramList)

  response.write(myresp & "<p>")
%>
```

#### (3) COM ( Visual Basic )

## 程序清单 11-24

```

Dim obj As deltabiz.XMLRPCClient
Set obj = CreateObject("deltabiz.xmlrpcClient")

ReDim param(1)
Dim strArr
param(0) = 873214

retval = obj.xmlRPC("http://localhost/xmlrpc/server.asp", "", _
    "", "", "GetCurrentBalance", param)

MsgBox retval

```

## (4) Java语言

## 程序清单 11-25

```

XmlRpcClient xmlrpc =
    new XmlRpcClient("http://www.localhost.com/xmlrpc/server.asp");
Vector params = new Vector();

params.addElement(873214);
Integer retVal = (Integer) xmlrpc.execute("GetCurrentBalance", params);

```

## (5) PHP

## 程序清单 11-26

```

$xmlclient=new xmlrpc_client("/xmlrpc/server.asp", .
    "localhost",80);
$ret=$xmlclient->send(new xmlrpcmsg("GetCurrentBalance", .
    array(new xmlrpcval("873214","double"))));

$returnval=$ret->value();
$retVal=$returnval->scalarval();

```

## (6) XML响应

XML对于上面请求的响应有如下结构：

## 程序清单 11-27

```

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <int>7635</int>
      </value>
    </param>
  </params>
</methodResponse>

```

如果在调用中存在问题，将收到一个如下的响应。下面的 XML说明在与 XML-RPC服务器联系时出现一个问题：

## 程序清单 11-28

```
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
          <value><int>345</int></value>
        </member>
        <member>
          <name>faultString</name>
          <value>
            <string>The server name or address could not be resolved.</string>
          </value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

因为以上的错误代码和错误串是一个 Win32 错误，因此你将把调用打包在一个错误处理程序中，应该对正在使用的特定的 XML-RPC 工具参照错误描述进行检查。

对于以上功能调用的一个例子如下：

## 程序清单 11-29

```
<!--#include file="xmlrpc.asp" -->
<%
  ReDim paramList(1)
  paramList(0)=876

  On Error Resume Next

  myresp = xmlRPC("http://localhost/xmlrpc/server.asp", _
    "GetCurrentBalance", paramList)

  If Err.Number<>0 Then
    response.write("Error Number: " & Err.Number & "<BR/>")
    response.write("Error Detail: " & Err.Description & "<P>")
  Else
    response.write(myresp & "<p>")
  End If
%>
```

让我们看看调用这些方法的服务器代码。再次强调服务器实现并不需要使用与客户相同的语言（甚至平台）。但是我们将得到相同的功效，就像为客户做的那样。

## (7) ASP

## 程序清单 11-30

```
<!--#include file="xmlrpc.asp" -->
<%
  rpcserver
```

```
%>

<SCRIPT LANGUAGE=JAVASCRIPT RUNAT=SERVER>
  function GetCurrentBalance (AccNumber)
  {
    var intBalance=7635;

    //return the amount to the client
    return intBalance;
  }
</SCRIPT>
```

---

#### (8) Java

程序清单 11-31

```
XmlRpcServer xmlrpc = new XmlRpcServer();
xmlrpc.addHandler("GetCurrentBalance", new BalanceHandler());

Integer result = xmlrpc.execute(request.getInputStream());
response.setContentType("text/xml");
response.setContentLength(result.length());
PrintWriter writer = response.getWriter();
writer.write(result);
writer.flush();
```

---

#### (9) PHP

程序清单 11-32

```
<?php
include("xmlrpc.inc"); include("xmlrpcs.inc");

function GetCurrentBalanceImpl($params)
{
    return new xmlrpcresp(new rpcval("873214","integer"));
}

$s = new xmlrpc_server(array("GetCurrentBalance" =>
                             "GetCurrentBalanceImpl"));

?>
```

---

### 11.3.5 书籍应用例子

下面的这个应用的例子（Wrox网站上有一些相关的本书没有显示的代码）表明如何激活一个远程程序来接收消息和更新远程的信息库。Wrox Books已经决定在世界五个不同的地方安装五个远程服务器。在那里，书籍信息以XML文件的方式存储。尽管这些文件也能从在伯明翰的本地服务器上直接更新（假设书籍目录的维护是在该站点进行的），使用者将能够选择咨询地点并得到一个主题表单，当这个表单被返回，使用者能选择一个主题以及查询的区域。这样，就能在这些地点得到有关该主题的信息。

让我们看看表示相关过程的图表（参见图 11-12）。

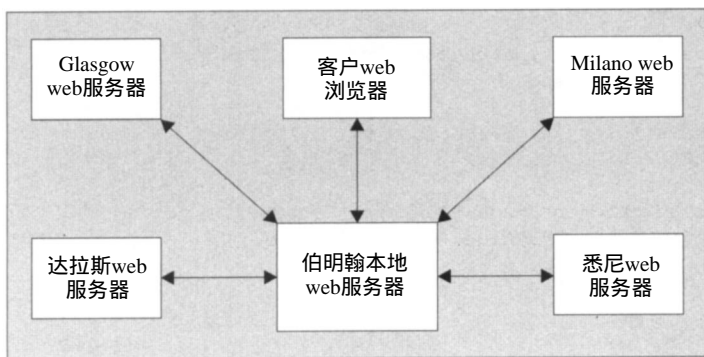


图 11-12

一个想获得一个或更多服务器上书籍信息的使用者，将通过显示器来选择站点，进而通过该站点获得一个书籍主题的列表（参见图 11-13）。

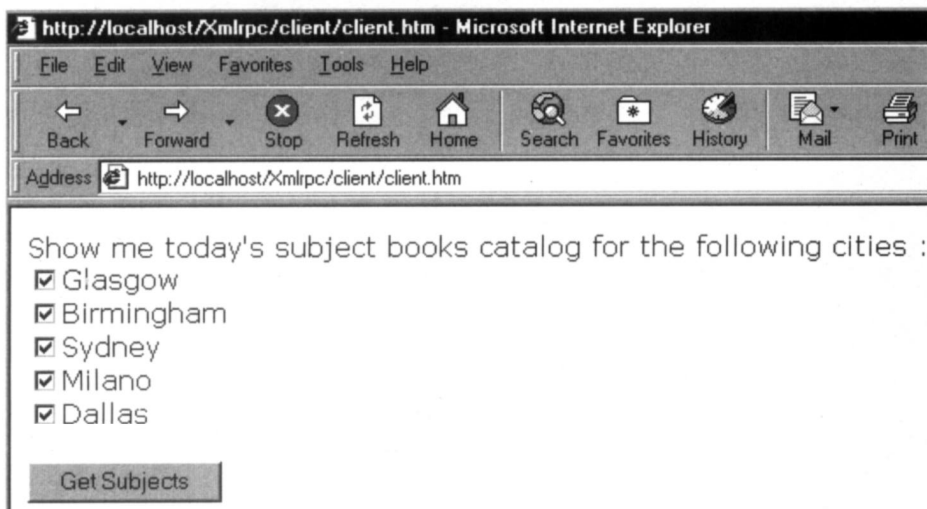


图 11-13

产生显示的HTML如下所示，它在文件client.htm里（参见程序清单 11-33）：

程序清单 11-33

```
<HTML>
<HEAD>
  <META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
  <form method="post" action="Subjects.asp" name=form1>
    Show me today's subject books catalog for the following cities :
    <BR/>
    <INPUT name="city" type=checkbox value="Glasgow"
      CHECKED>Glasgow
    <BR/>
```

```
<INPUT name="city" type="checkbox" value="Birmingham"
CHECKED>Birmingham
<BR/>
<INPUT name="city" type="checkbox" value="Sydney"
CHECKED>Sydney
<BR/>
<INPUT name="city" type="checkbox" value="Milano"
CHECKED>Milano
<BR/>
<INPUT name="city" type="checkbox" value="Dallas"
CHECKED>Dallas
<P/>
<input type="hidden" name="Function" VALUE="GetSubjects">
<input type="submit" value="Get Subjects">
</form>
</BODY>
</HTML>
```

响应返回了一个城市的列表，以及一个书籍主题的下拉式列表（参见图 11-14）。

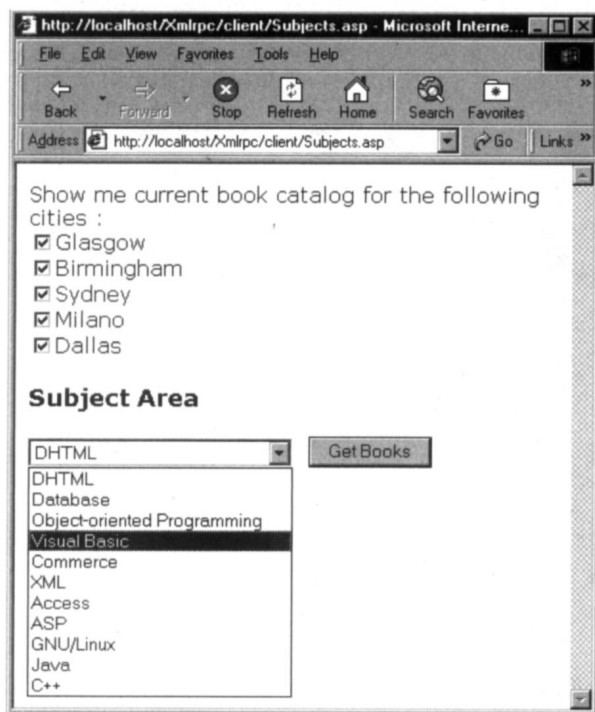


图 11-14

在特定的远程站点，相关书籍的数据在被存入到一个 XML 文件里。这个文件被称作 CityNameBooks.xml，每个文件是否被查询取决于选择的城市。因此，所有城市的 XML 数据文件都是不同的，并被该站点更新。

这个例子说明了 XML-RPC 的强大，每一个 XML 库均能存在世界上不同的远程服务器上。



同样,使用这个例子,不仅能够将所有的XML库放进网络服务器的一个目录里,还能够通过在不同端口创建Web实现多个虚拟服务器。可以通过如 `http://myserver:port/<citystore>.xml` 的方式实现访问,位置对客户是完全隐藏的。

关于在何处指定XML库位置的进一步描述将在本章给出。

下面的XML文件是为Glasgow书店建立的。这里有许多书,但每本书必须包含一个书的ID、出版日期、书名、作者以及书的类属。出版日期是UK日期格式,所有的处理将在伯明翰UK Web服务器上。很明显,如果该XML文件要在一些国际化的服务器上处理,使用类似于ISO 8601(该格式被XML-RPC规范内部使用)时间格式将更有好处。

#### 程序清单 11-34

```
<?xml version="1.0"?>
<books>
  <book bookid="1" pubdate="01/03/1999">
    <title>
      Instant Netscape Dynamic HTML Programmer's Reference NC4
      Edition
    </title>
    <authors>
      <author>Alex Homer</author>
      <author>Chris Ullman</author>
    </authors>
    <subject>DHTML</subject>
  </book>
  <book bookid="2" pubdate="11/10/2000">
    <title>Professional Visual Basic 6 Distributed Objects</title>
    <authors>
      <author>Rockford Lhotka</author>
    </authors>
    <subject>Visual Basic</subject>
  </book>
  <book bookid="3" pubdate="11/12/1999">
    <title>Beginning Active Server Pages 2.0</title>
    <authors>
      <author>Brian Francis</author>
      <author>John Kauffman</author>
      <author>Juan T Llibre</author>
      <author>David Sussman</author>
      <author>Chris Ullman</author>
    </authors>
    <subject>ASP</subject>
  </book>
  <book bookid="4" pubdate="11/10/2000">
    <title>Beginner's Guide to Access 2.0</title>
    <authors>
      <author>Wrox Author Team</author>
    </authors>
    <subject>Access</subject>
  </book>
  <book bookid="5" pubdate="11/12/1999">
    <title>Beginning Java 2</title>
    <authors>
      <author>Ivor Horton</author>
```

```

    </authors>
    <subject>Java</subject>
  </book>
</books>

```

## 1. 获得主题列表

图11-15将表明从XML数据文件里获得一个主题列表的过程。

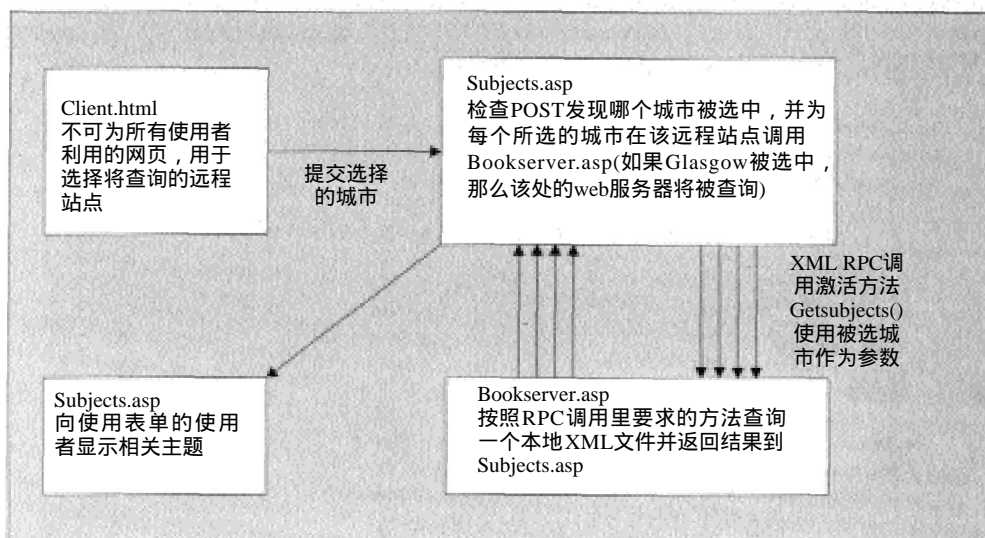


图 11-15

表单被提交到一个称作 subject.asp的页面上，它是用 VBScript写的。我们必须首先将 XML-RPC客户/服务器的文件包含进来，该文件包含了一个功能即将你的方法调用转换成标准定义（就ASP而言，目前使用的是VBScript）。

```
<!--#include file="xmlrpc.asp"-->
```

然后可以使用HTML在屏幕显示城市及其复选框，这个文件称作 proxy.asp，当按照选择的主题查询书籍时它将被使用。

程序清单 11-35

```

<HTML>
<HEAD>
  <META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
  <form method="post" action="proxy.asp" id=form1 name=form1>
    Show me current book catalog for the following cities :
    <BR/>
    <INPUT name="city" type=checkbox value="Glasgow"
      CHECKED>Glasgow
    <BR/>
    <INPUT name="city" type=checkbox value="Birmingham"
      CHECKED>Birmingham
  </form>

```

```

<BR/>
<INPUT name="city" type=checkbox value="Sydney"
CHECKED>Sydney
<BR/>
<INPUT name="city" type=checkbox value="Milano"
CHECKED>Milano
<BR/>
<INPUT name="city" type=checkbox value="Dallas"
CHECKED>Dallas
<P/>

```

然后我们创建一个目录对象的实例来保存每一个返回的主题的名字。将使用函数 CheckDuplicate ( ), 这个函数决定是否一个主题的实例已经被找到。如果没有, 它将被去掉——这样就可以避免用户在下拉列表中看到重复主题。

程序清单 11-36

```

<%
Dim objDict
Set objDict = Server.CreateObject("Scripting.Dictionary")

Function CheckDuplicate(strValue)
If objDict.Exists(strValue) Then
CheckDuplicate="TRUE"
Else
objDict.Add strValue, strValue
CheckDuplicate="FALSE"
End If
End Function

```

然后我们引用在前页 client.htm 中的隐藏元素 Function, 它将用于定义被激活的 XML-RPC 方法。我们想调用的这个方法称作 GetSubjects ( ), 被选城市将被作为一个参数传送, 因此我们声明一个数组 ( 该参数必须以数组形式传送 ) 称作 paramlist, 有一个元素。

程序清单 11-37

```

strFunction=Request.Form("Function")
Dim paramList(1)
Dim intNumCities
intNumCities=0
Dim intCounter
intCounter=0

```

这些值被插入到一个断开的记录集中去, 因为有记录集对象的 save ( ) 方法, 因而保存分布式系统数据是简单的。记录集设置了用户游标来许可记录集的断开。

程序清单 11-38

```

'Create a local ADO RecordSet to store our data for manipulation
Set objRS = Server.CreateObject("ADODB.RecordSet")
objRS.CursorLocation=3

```

随后, 在记录集对象里创建一个区域。我们想通过标题的地点和它在本地的唯一 ID 来识别

每一个标题。因此，我们在记录集里附属三个标题，ID作为一个整型来定义（整型在ADO记录集里以“3”来定义），标题和城市作为字符串（字符串型在ADO记录集里以“8”来定义），然后使用Open（）打开记录集进行操作。

---

程序清单 11-39

---

```
'Create the Fields
objRS.Fields.Append "ID",3
objRS.Fields.Append "Title",8
objRS.Fields.Append "City",8
objRS.Open
```

因为在每一个城市都有一个服务器，我们产生XML-RPC调用来得到一个相关主题的列表。首先检查客户是否已经查看了Glasgow数据库，做这个工作是通过检查FORM POST，而FORM POST是由使用Instr（）函数的用户产生的。如果第一个指定的串包含一个第二个串的结果（“Glasgow”在第一个交易里），该函数将返回一个大于零的数。在这种情况下，那么将参数定义为Glasgow（因为GetSubject（）期望一个参数）。然后，XML-RPC使用如下格式产生远程服务器调用。

```
returnValue = xmlRPC(URL,MethodName,ParamaterArray)
```

显然，该应用程序适合于我们的那些单一服务器，服务器查询以默认方式在本地服务器上生成。但是，在XML-RPC调用里的URL能够被转换到任何远程服务器，只要该服务器支持对于该应用程序的XML-RPC接口。

在单一服务上测试它的简单方法是在5个不同的端口上设置5个测试站点，并且在XML-RPC调用里更改URL，来反映此功能。

对XML-RPC调用的返回值是一个数组，该数组是一个被查询地点的相关主题使用逗号定界的数组，因此，使用split（）方法来得到主题的一个一维数组。然后再次通过这个数组将一个ID、主题标题和城市描述加到记录集中，我们对每一个被用户选定的分布地区重复XML-RPC调用，这些地区都是在请求对象表单里被指定的。

---

程序清单 11-40

---

```
If Instr(1,Request.Form("City"),"Glasgow")>0 Then
    paramList(0)="Glasgow"
    strGlas = _
        xmlRPC("http://LOCALHOST/xmlrpc/client/BookServer.asp", _
            "GetSubjects", paramList)
    arrTemp=split(strGlas,",")

    For k=0 To UBOUND(arrTemp)
        objRS.AddNew
        objRS.Fields("ID").Value=k
        objRS.Fields("Title").Value=arrTemp(k)
        objRS.Fields("City").Value=paramList(0)
        objRS.Update
        intCounter=intCounter+1
    Next
End If
```

```
If Instr(1,Request.Form("City"),"Birmingham")>0 Then
    paramList(0)="Birmingham"
    strBirm = _
        xmlRPC("http://LOCALHOST/xmlrpc/client/BookServer.asp", _
            strFunction, paramList)
    arrTemp=split(strBirm,",")
    For k=0 To UBOUND(arrTemp)
        objRS.AddNew
        objRS.Fields("ID").Value=k
        objRS.Fields("Title").Value=arrTemp(k)
        objRS.Fields("City").Value=paramList(0)
        objRS.Update
        intCounter=intCounter+1
    Next
End If

If Instr(1,Request.Form("City"),"Sydney")>0 Then
    paramList(0)="Sydney"
    strSyd = _
        xmlRPC("http://LOCALHOST/xmlrpc/client/BookServer.asp", _
            strFunction, paramList)
    arrTemp=split(strSyd,",")

    For k=0 To UBOUND(arrTemp)
        objRS.AddNew
        objRS.Fields("ID").Value=k
        objRS.Fields("Title").Value=arrTemp(k)
        objRS.Fields("City").Value=paramList(0)
        objRS.Update
        intCounter=intCounter+1
    Next
End If

If Instr(1,Request.Form("City"),"Milano")>0 Then
    paramList(0)="Milano"
    strMil = _
        xmlRPC("http://LOCALHOST/xmlrpc/client/BookServer.asp", _
            strFunction, paramList)
    arrTemp=split(strMil,",")

    For k=0 To UBOUND(arrTemp)
        objRS.AddNew
        objRS.Fields("ID").Value=k
        objRS.Fields("Title").Value=arrTemp(k)
        objRS.Fields("City").Value=paramList(0)
        objRS.Update
        intCounter=intCounter+1
    Next
End If

If Instr(1,Request.Form("City"),"Dallas")>0 Then
    paramList(0)="Dallas"
    strDal = _
        xmlRPC("http://LOCALHOST/xmlrpc/client/BookServer.asp", _
            strFunction, paramList)
    arrTemp=split(strDal,",")

    For k=0 To UBOUND(arrTemp)
```

```

objRS.AddNew
objRS.Fields("ID").Value=k
objRS.Fields("Title").Value=arrTemp(k)
objRS.Fields("City").Value=paramList(0)

objRS.Update
intCounter=intCounter+1
Next
End If

```

当重复每一个被选的主题，我们将从所有选中的地点得到一个完全断开的书籍主题描述的记录集。如果这些记录集不空，并且使一个主题列表返回，就把它们写进 HTML SELECT 元素，这也是通过重复记录设备对象来实现的。在这个过程中，我们调用了 CheckDuplicate ( ) 函数。它保证了当所有城市的所有主题集中在一起时且某个主题多次出现，不会将同样的主题写两次。在下一节将对这一功能做更详细描述。如果无主题可用的（这意味着被选城市没有当前的书本储备），那么一个适当的消息将提示给用户。

#### 程序清单 11-41

```

'get the values in Ascending order
objRS.MoveFirst

Response.Write "<h3>Subject Area</h3>"

If NOT objRS.EOF Then
    Response.Write "<SELECT Name='Subject'>"
    Dim strValue
    strValue=""

    Do While NOT objRS.EOF
        If CheckDuplicate(objRS.Fields(1).Value)="FALSE" Then
            Response.Write "<OPTION VALUE='" & objRS.Fields(1).Value _
                & "'>" & objRS.Fields(1).Value & vbNewLine
        End If

        objRS.MoveNext
    Loop
    Response.Write "</SELECT>"
Else
    Response.Write "<H4>There are currently no subject(s) at " & _
        "your chosen locations(s).</H4>"
End If

objRS.Close
Set objRS = Nothing
%>

```

最后，我们将建立一个函数，当用户从 SELECT 选择一个主题并点击了“GET BOOKS”按钮，提交了名为 proxy.asp 文件后，该函数将被调用，在后面我们将作描述。

#### 程序清单 11-42

```

<input type="hidden" name="Function" VALUE="GetBooks">
<input type="submit" value="Get Books" id=submit1 name=submit1>
</form>

```



```
</BODY>  
</HTML>
```

## 2. 检索书籍信息

既然已经从选中的城市得到了相关主题，那么我们想选择主题中的一个并得到该主题相关的所有标题，包括书籍的作者。下面是实现该功能的一个例子。在该例中用户选中了 Glasgow 和 Birmingham，并选中 Commerce（商务）作为主题，并点击了“GET BOOK”按钮（参见图 11-16）。

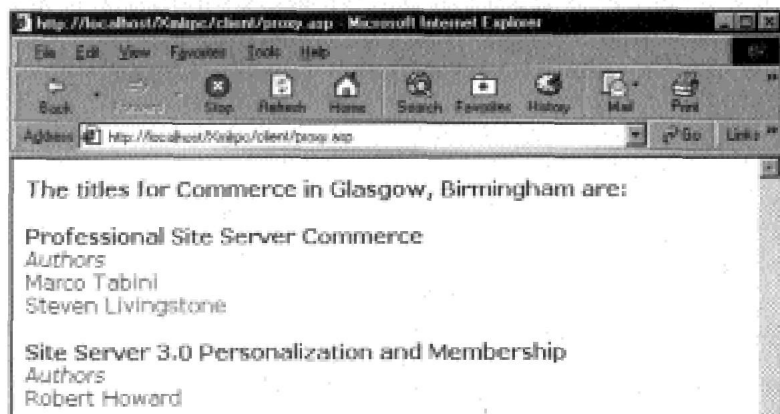


图 11-16

应保证你已经遵循了本章开始处的指令并将所有 Wrox 下载文件拷贝到你的 Web 服务器的下述目录：<root>/xmlrpc/client/。

图 11-17 说明系统的这个部分是如何链接到前一部分的，在前一部分里，用户已经选择了一个主题。

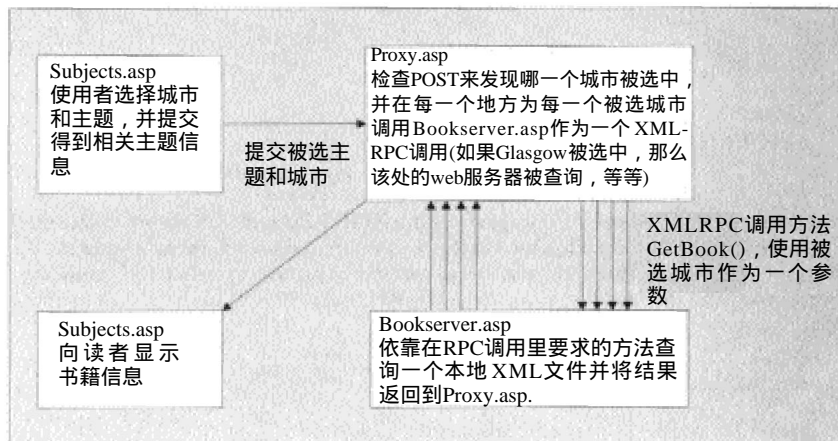


图 11-17

创建上页 ( proxy.asp ) 的脚本基本上使用 JavaScript。但是针对 ASP，当前使用的工具都是用 VBScript 来实现的，因此，必须有一个使用 VBScript 的程序，用它创建一个 VBArray 对象并发送到 XML-RPC 方法。这是需要的，因为在 XML-RPC 方法调用里使用的参数表单将上面的请求描述为一个参数数组 ( 必须是一个 VBArray )。下面是 Proxy.asp 文件的第一部分：

程序清单 11-43

```
<!--#include file="xmlrpc.asp"-->

<SCRIPT LANGUAGE=vbscript RUNAT=Server>
    Function CreateVBArray(a)
        Dim i, j, k
        Dim arr(2)
        arr(0)=Request.Form("Subject")
        arr(1)=a
        CreateVBArray = arr
    End Function
</SCRIPT>
```

然后，转到使用 JavaScript 上。首先，再次使用函数 checkDuplicate ( )，早些时候它在 Subject.asp 中已经用过。该函数的基本功能是把值加到 Dictionary 对象，并检查该值是否已经存在 ( 使用对象的 Exists ( ) 方法 ) —— 如果不存在，它将被加上。

程序清单 11-44

```
<SCRIPT LANGUAGE=JAVASCRIPT RUNAT=SERVER>
    var objDict = Server.CreateObject("Scripting.Dictionary")

    function CheckDuplicate(strValue)
    {
        var strRet="";

        if (objDict.Exists(strValue))
            strRet="TRUE";
        else
        {
            objDict.Add(strValue,strValue);
            strRet="FALSE";
        }

        return strRet;
    }
}
```

函数 DrawScreen ( ) 封装了 XML-RPC 代码，该代码将创建 HTML 来显示搜索的结果。我们再次声明参数数组和保持着书籍数据的目录对象。同时创建断开的记录集并创建一个域将把 XML-RPC 调用的响应加进来。下面已经将它列出用以同 VBScript 比较：

程序清单 11-45

```
function DrawScreen()
{
    var strFunction=Request.Form("Function");
    var paramList = new Array(2);
```

```

var intNumCities=0;
var arrBookList = new Array(1);
var intCounter=0;
var objDictList=Server.CreateObject("Scripting.Dictionary");

//Create a local ADO RecordSet to store our data for manipulation
var objRS = Server.CreateObject("ADODB.RecordSet");
objRS.CursorLocation=3;

//Create the Fields
objRS.Fields.Append("ID",3);
objRS.Fields.Append("Title",8);
objRS.Fields.Append("City",8);
objRS.Open();

```

激活远程方法的部分有一点细微的不同。首先检查至少一个城市被选中，如果如此，那么该城市将被引用：

程序清单 11-46

```

if (Request.Form("City").Count>0 &&
    Request.Form("City").Item(1).indexOf("Glasgow")!=-1)
{

```

然后使用CreateVBAArray方法返回一个VBAArray类型的对象，VBAArray保持了该方法的参数。然后，一个XML-RPC调用被执行，即使用在对应URL中的BookServer.asp文件调用（GetBooks()）函数，VBAArray列出参数被发送到这个调用（其中含有城市和主题），然后将返回每本书和它的描述（一个记录），并用逗号隔开（如：BookRecord1, BookRecord2, BookRecord3,等等），结果被附加到记录集。

每个ID被唯一的索引k所定义，该索引标识一个包含书的标题和书的作者的记录。

程序清单 11-47

```

paramList=CreateVBAArray(Request.Form("City").Item(1));
strGlas =
    xmlRPC("http://LOCALHOST/xmlrpc/client/BookServer.asp",
        strFunction, paramList);
arrTemp = strGlas.split(",");

for (k=0;k<arrTemp.length;k++)
{
    objRS.AddNew();
    objRS.Fields("ID").Value=k;
    objRS.Fields("Title").Value=arrTemp[k];
    objRS.Fields("City").Value=Request.Form("City").Item(1);
    objRS.Update();
    intCounter++;
}
}

```

// code repeating this for each city (as before) not shown.

创建一个报头来声明主题和被查询的城市。然后重复记录集，对每一个记录都有一个条目

(不是“no title”，它的返回是该指定主题内无此书)，我们保证它仍然未被写过。如果它仍然未被写过，分裂函数创建一个数组，使用“#”作为分隔。这是因为在RPC调用期间当BookRecord被创建时，书籍标题同它的作者被“#”号所隔开。因此，当分裂函数创建数组时，第一个元素将总是保持书籍的标题，剩下的元素将是书籍作者。

程序清单 11-48

```
Response.Write("<h4>The titles for " + Request.Form("Subject") +
    " in " + Request.Form("City") + " is : </h4>");

//get the values in Ascending order
if (!objRS.EOF)
{
    while (!objRS.EOF)
    {
        if (objRS.Fields(1).Value!="no titles")
        { //we should write it out
            if (CheckDuplicate(objRS.Fields(1).Value)=="FALSE")
            {
                str = objRS.Fields(1).Value.split("#");
                Response.Write("<P><B>" + str[0] + "</B>");
                Response.Write("<BR><I>Authors</I><BR>");
                for (i=1;i<str.length;i++)
                    Response.Write(str[i] + "<BR>");
            }
        }
        j++;
        objRS.MoveNext();
    }
}
```

如果没有相关的书籍，那么将给出一个提示消息，记录集资源将被释放。

程序清单 11-49

```
else
    Response.Write("<h4>There are currently no books for this "
        + "subject at your chosen locations(s).</H4>");

objRS.Close();
var objRS = null;
}
</SCRIPT>

<HTML>
<HEAD>
</HEAD>

<%DrawScreen()%>

</HTML>
```

### 3. 目录的更新

这种分布式系统的一个重要表现是它有能力从中心服务器更新条目，并独立于任何正在运

行的分布服务器。更新条目的 HTML 界面如下所示。在下载的例子文件为 AddNew.asp 文件（参见图 11-18）。

在这个部分，你将必须作为一名允许写文件的用户被登录，因为你将直接更新 XML 文件。一种方法是进行响应鉴定和对于客户目录的匿名用户失效。你能在 IIS Management Console 中通过右击客户目录和选择特性做到这些。然后，选择 Directory Securing（目录安全）标记并点击 Edit（编辑）键，保证 Allow Anonymous Access（允许匿名登录）不被选择，而 Windows NT Challenge / Response 被选中。



图 11-18

图 11-19 显示目录更新是如何工作的：

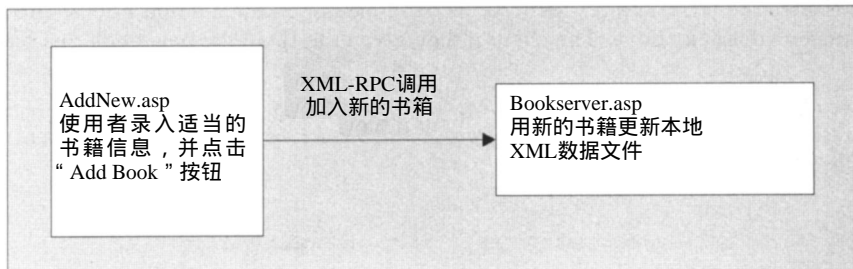


图 11-19

被称作 AddNew.asp 的文件包含以上的 HTML 和实现 XML-RPC 所需要的脚本。我们想激活的

XML-RPC方法有如下接口（参见表 11-2）。

表 11-2

方法名称	参 数
AddNewBook()	city——城市名字 PubDate——出版日期 Title——书籍标题 Authors——书籍作者列表 Subject——书籍的主题领域

在客户页，AddNew.asp以常用的PRC include文件和HTML报头开始：

程序清单 11-50

```
<!--#include file="xmlrpc.asp" -->
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE></TITLE>
</HEAD>
<BODY>
```

然后从上面的网页检查是否有一个作者被提交（它对一本书总是需要的）。如果有，就对每一个提交的值设立参数数组：

程序清单 11-51

```
<P>
<%
    If NOT Request.form("auth") = "" then
        Dim paramList(5)
        paramList(0)=Request.form("city")
        paramList(1)=Request.form("pubdate")
        paramList(2)=Request.form("title")
        paramList(3)=Request.form("auth")
        paramList(4)=Request.form("subject")
```

既然对方法已定义了参数，就能对 XML-RPC使用COM对象来激活它。该XML-RPC将返回一个已经被加入的书的新ID（关于XML-RPC COM工具的更多信息将在“到哪里去”一部分给出）。通过创建xmlrpcClient对象的实例并向AddNewBook（）方法传送参数得以实现。然后客户得到了已经加入的新书的ID。

作为一个练习，你可能希望尝试和扩展该功能，即从BookServer.asp文件里使用一个XML-RPC调用从书库得到一个新的ID，而不是从本地XML文件中得到ID。

程序清单 11-52

```
Set objDB = Server.CreateObject("deltabiz.xmlrpcClient")
myresp = _
    objDB.xmlRPC("http://LOCALHOST/xmlrpc/client/BookServer.asp", _
        "", "", "", "AddNewBook", paramList)
```

```

Response.Write "<P><STRONG>The new book has been added to "
Response.Write Request.form("city") & ".</STRONG></P>"
Response.write(" The book has been given an ID of " & myresp & "<P>")
Else
%>

```

最后是一个实际使用的用于更新书籍所需要的程序段，可输入城市、出版日期、书籍标题、作者和书籍主题。AddBook ( ) 方法作为一个隐藏元素引入。

#### 程序清单 11-53

```

<P><STRONG>Add a new Book</STRONG></P>
<FORM action=AddNew.asp id=form1 method=post name=form1>
Choose a City and enter the appropriate details
<BR>
<INPUT CHECKED name=city type=radio value=Glasgow>Glasgow<BR>
<INPUT name=city type=radio value=Birmingham>Birmingham<BR>
<INPUT name=city type=radio value=Sydney>Sydney<BR>
<INPUT name=city type=radio value=Milano>Milano<BR>
<INPUT name=city type=radio value=Dallas>Dallas</P>
<P>
<P>Publication Date<BR><INPUT name="pubdate"></P>
<P> Title<BR><INPUT name="title"></P>
<P> Authors<BR>
<INPUT name="auth"><BR>
<INPUT name="auth"><BR>
<INPUT name="auth"><BR>
<INPUT name="auth"><BR>
<INPUT name="auth"><BR>
<INPUT name="auth"><BR>
<INPUT name="auth">
</P>
<P>Subject<BR><INPUT name="subject"></P>
<% End If%>
<INPUT name=Function type=hidden value=AddBook>
<INPUT id=submit1 name=submit1 type=submit value="Add Book">
</FORM></P>
</BODY>
</HTML>

```

#### 4. Wrox书籍服务器

在该应用中我们仍然没有处理的部分是由 BookServer.asp所支持的服务器的实现。这是在 Internet上工作所使用的XML-RPC实现工具的典型范例。不需要直接访问服务器代码，但是提供一个界面标准和语义。这一部分对于理解 RPC应用程序的另一侧是非常有用的。如果你自己写RPC服务器应用程序，那么它也会有特别的帮助。

应用程序的这一部分就是实现我们已经在前面的程序中调用的远程方法。

在XML-RPC的ASP版本里，尽管调用过程 rpcserver ( ) 来激活XML服务器，客户和服务器的执行工具有相同的include文件报头部分对于所有XML-RPC服务器的实现都是需要的——它是一个简单的使用数据对服务器的初始化，这些数据都是通过XML-RPC调用从客户处POST的。



## 程序清单 11-54

```
<!--#include file="xmlrpc.asp" -->
<%
    rpcserver
%>
```

除了早些看到的 VBAArray 函数，服务器的剩余部分都使用 JavaScript 实现。

## 程序清单 11-55

```
<SCRIPT LANGUAGE=vbscript RUNAT=Server>
Function CreateVBAArrayAddNewBook(a)
    Dim i, j, k
    Dim arr(1)
    arr(0) = a
    CreateVBAArrayAddNewBook = arr
End Function
</SCRIPT>
```

第一个函数是对于特定的城市得到一个相关主题。创建一个 MSXML DOM 对象的实例并打开该城市的 XML 文件，然后使用 `getElementsByTagName()` 方法得到 `NodeList` 对象，并创建一个相关主题的一维数组（它是书籍元素的第三个子结点）。

## 程序清单 11-56

```
<SCRIPT LANGUAGE=JAVASCRIPT RUNAT=SERVER>
function GetSubjects(city)
{
    var arrTitles = new Array();

    //'load the XML document
    var objXML = Server.CreateObject("Microsoft.XMLDOM");
    objXML.async = false;
    objXML.load(Server.MapPath(city + "books.xml"));

    //'get all the book titles in a Nodelist
    var objNodeList = objXML.getElementsByTagName("book");

    for (var i=0;i<objNodeList.length;i++)
    {
        arrTitles[i]=objNodeList.item(i).childNodes(2).text;
    }

    return arrTitles;
}
```

当集中了所有主题，我们想有一个方法来得到特定城市中被选主题的所有书籍，`GetBooks()` 函数将做此工作，同样也装载该相关城市的 XML 库。

## 程序清单 11-57

```
function GetBooks(subject,city)
{
    var arrTitles = new Array();
```

```

//load the XML document
var objXML = Server.CreateObject("Microsoft.XMLDOM");
objXML.async = false;
objXML.load(Server.MapPath(city+"books.xml"));

```

然后，在一个Nodelist对象里得到所有<book>元素，并遍历每一个元素，检查书籍的主题类型（书籍元素的第三个子结点）是否与从客户发送的一样。如果有一个主题匹配，那么该书的标题被加入，成为一个数组的第一个元素。重复遍历每一个<authors>元素（<book>元素的第二个子结点），每一个<authors>元素可能有多于一个<author>子结点。每个作者被加到数组的第二个元素；该数组就是刚刚将书籍标题加入的那一个，其中每本书的作者被“#”隔开，下一本书被加到数组的下一元素等等，以以下形式直至结束。

```

arrTitles[0]="BookTitleA#Author1"
arrTitles[1]="BookTitleB#Author1# Author2# Author3"
arrTitles[2]="BookTitleC#Author1# Author2"
...

```

实现该功能的代码如下：

程序清单 11-58

```

//get all the book titles in a Nodelist
var objNodeList = objXML.getElementsByTagName("book");

var j=0;
for (var i=0;i<objNodeList.length;i++)
{
    if (objNodeList.item(i).childNodes.length>0)
    {
        if (subject==objNodeList.item(i).childNodes(2).text)
        {
            arrTitles[j]=objNodeList.item(i).childNodes(0).text;

            for (k=0;
                k<objNodeList.item(i).childNodes(1).childNodes.length;
                k++)
            {
                if (k==0)
                    arrTitles[j]+="# " +
objNodeList.item(i).childNodes(1).childNodes(k).text;
                else
                    arrTitles[j]+="# " +
objNodeList.item(i).childNodes(1).childNodes(k).text;
            }

            j++;
        }
    }
}

```

如果对于特定城市无相关标题，那么将简单地返回文本“no titles”(无标题)。

## 程序清单 11-59

```
if (arrTitles.length==0)
{
    arrTitles[0]="no titles";
    return arrTitles;
}
else
{
    return arrTitles;
}
```

为了更新城市的 XML 书库，我们将调用 AddNewBook ( ) 方法，它将把新书标题、一个作者的数组和书籍的主题作为数据加入。我们创建一个 MXSML DOM 的实例，并用 documentElement 作为 XML 文件的根：

## 程序清单 11-60

```
//This function adds a new book to the specified city xml file
//Authors is a list of authors separated by a |
function AddNewBook(city,pubDate,title,Authors,subject)
{
    var ret="";
    var arrTitles = new Array();

    //load the XML document
    var objXML = Server.CreateObject("Microsoft.XMLDOM");
    objXML.async = false;
    objXML.load(Server.MapPath(city+"books.xml"));
    var objRoot = objXML.documentElement;
```

该文件最后的 <book> 元素用 lastChild ( ) 方法找到，将被加入的新元素的书籍 ID 比最后元素的书籍 ID 大得多，并被属性 bookid 定义。

## 程序清单 11-61

```
var objLastBookNode = objRoot.lastChild;

//Get the next available ID number for a book
var intNextNum =
    parseInt(objLastBookNode.getAttribute("bookid"))+1;
```

用 CreateElement ( ) 方法来创建一个新的 <book> 元素并附加到文件后面，然后设置两个属性，即已经得到的 ID 属性和出版日期属性，该属性在 RPC 调用里也被作为一个参数被传送。

<title> 元素也被创建，并被附加成为新的 <book> 元素的子结点，该 <title> 元素的值从 XML-RPC 调用中被传送的参数中获得。

## 程序清单 11-62

```
//Create a new book element and add attributes
var objBookEl = objXML.createElement("book");
```

```
//Add the Book Element Attributes
objBookEl.setAttribute("bookid",intNextNum)
objBookEl.setAttribute("pubdate",pubDate)

//Add the Title Element
var objTitleEl = objXML.createElement("title");
objBookEl.appendChild(objTitleEl);
objTitleEl.text=title;
```

创建一个<authors>元素并加入被作为参数传入的每一个作者（每位作者都以逗号隔开），成为<author>元素。该元素是<authors>的子元素，<authors>元素随后被附加到双亲<book>元素。

---

程序清单 11-63

```
//Add the Authors of the book Element
var objAuthorsEl = objXML.createElement("authors");

var arrAuthList = new Array();
arrAuthList=Authors.split(",");

for (var i=0;i<arrAuthList.length;i++)
{
    if (arrAuthList[i].length>1)
    {
        var objAuthorEl = objXML.createElement("author");
        objAuthorEl.text=arrAuthList[i];
        objAuthorsEl.appendChild(objAuthorEl);
    }
}

objBookEl.appendChild(objAuthorsEl);
```

随后<subject>元素被创建并被附加到<book>元素（该元素早些时候已经创建），主题文本被设置成一个XML-RPC参数来传送。随后，<book>自身代表一本新书被加到文档根目录。

---

程序清单 11-64

```
//Add the Subject Element
var objSubjectEl = objXML.createElement("subject");
objBookEl.appendChild(objSubjectEl);
objSubjectEl.text=subject;

objRoot.appendChild(objBookEl);
```

最后，用Microsoft XML执行工具的save（）方法保存XML文档，用它被更新的结构和返回的被加入书籍的ID作为对XML-RPC调用的响应：

---

程序清单 11-65

```
//persist the new document
var xmldoc = Server.CreateObject("Microsoft.XMLDOM");
xmldoc.async = false;
xmldoc.load(objXML);
xmldoc.save(Server.MapPath(city+"books.xml"));
```

```

var objXML = null;
var xmldoc = null;

return intNextNum;
}
</SCRIPT>

```

### 11.3.6 到哪里去

下面这个清单展示了一些不同的实现工具和得到它们的站点：

- XML-RPC client for Python, 由PythonWare提供：<http://www.pythonware.com/xmlrpc/>
- XML-RPC client/server for Java, 由Hannes Wallnöfer提供：<http://helma.at/hannes/xmlrpc/>
- XML-RPC client for Java, 由Josh Lucas提供：<http://www.stonecottage.com/josh/rpcClient.html>
- XML-RPC client/server for Perl, 由Ken MacLeod提供：<http://bitsko.slc.ut.us/~ken/xml-rpc/>
- XML-RPC in Tcl, 由Steve Ball提供：<http://www.zveno.com/zm.cgi/in-tclxml/in-xmlrpc.tml>
- XML-RPC client/server for ASP, 由David Carter-Tod提供：<http://www.wc.cc.va.us/dtod/XMLRPC/>
- XML-RPC client COM, 由Steven Livingstone提供：<http://www.deltablz.com/xmlrpc/default.asp>
- XML-RPC client/server PHP, 由Useful Int. 公司提供：<http://usefulinc.com/xmlrpc/>

在Frontier和Zope 2.0也有内置的XML-RPC支持(<http://www.Zope.org/Download/Releases/Zope-2.0.01>)。

#### 1. ASP

所有文件, 包括客户或者服务器的, 只要参予了 XML-RPC的方法调用, 均需要在顶端将文件xmlrpc.asp包括进来。通过XML-RPC调用一个方法, 将使用如下语法:

程序清单 11-66

```
xmlRPC("Server URI", "procedure or function name", array of arguments)
```

目前的实现需要 MS Internet Explorer 5 和在 <http://www.alphasierapapa.com/lisDev/Components/> 上的免费软件 ASPTear组件。在服务器上, Alvaro Redondo的免费Base64编码库也必须注册(使用MSDOS上的regsvr32工具)。

#### 2. COM

XML-RPC COM客户实现是本书的作者自己(Steven Livingstone)写的, 目前, 它正在XML-RPC领域测试, DLL可免费从<http://www.deltebiz.com/xmlrpc/default.asp>上得到, 它将使用regsvr32工具注册。

目前, ProgID是deltabiz.xmlrpcClient, 用VBScript可以执行一个XML-RPC调用:

程序清单 11-67

```

Set objDB = Server.CreateObject("deltabiz.xmlrpcClient")
myresp = objDB.xmlRPC(URL, ProxyURL, ProxyUserName, ProxyPassword, _
    methodName, ParameterArray)

```

在表11-3中。

表 11-3

参 数	意 义
URL	XML-RPC服务器的地址
ProxyURL	代理服务器的URL，如果不用代理，可设置为“ ”
ProxyUserName	使用代理的用户名，如无代理，应使用“ ”
ProxyPassword	使用的代理的口令，如无代理，应使用“ ”
methodName	将调用的远程方法的名字
parameterArray	方法的参数

Proxy版需要ASPTear的商业版，其他版本均可用免费版本。

### 3. Java

有两个Java的实现工具，一个客户/服务器程序来自于Hannes Wallnöfer，一个客户程序来自Josh Lucas。

第一个必须内置在HTTP服务器上。一个XML解析器必须能在此系统上使用。因为XML-RPC库使用SAX，你应该使用一个下载区列出的建议使用的解析器，尽管默认的是James Clark的比较流行的XT解析器。这个站点还有API文档和一个邮件列表。

Josh Lucas有一个基于Java的XML-RPC客户程序。下载文档应被包括在你的类路径(classpath)里，且com.barista.\*应该被引入。

也有许多用Python、Perl、Tcl和PHP编写的实现工具——所有这些均可以从<http://www.xmlrpc.com/>这一XML-RPC站点下载。

## 11.4 SOAP

XML-RPC由于其简单性获得了广泛的支持，使用它有事半功倍的效果，其代码的编写简单到几乎可以使用任何语言——在许多平台——在远程服务器上执行。既然许多Web程序员熟悉提交到服务器脚本和网关的应用程序，因此，他们绝大多数都能使用这些有力的技术实现所需的代码。

在这一部分，我们将关注对XML-RPC的一些改进，它们中的一些被提议称为简单对象访问协议(Simple Object Access Protocol, SOAP)。首先，我们将关注什么是SOAP和它形成的理由，这一点与XML-RPC的某些缺点的定位有关。随后，将看到使用ASP描述SOAP的简单工具——尽管与你在XML-RPC看到的风格一样，但一个SOAP界面能使用任何语言来写。最后，将就SOAP所能解决的问题范围和不能解决问题的范围进行讨论。

我们将遵循SOAP的最新标准，它来自于<http://msdn.microsoft.com/xml/general/soapspec-v1.asp>。请注意，许多文章，包括在Microsoft站点的一篇仍然参考0.9版，但是该版本与新版本有很大的不同。

### 11.4.1 XML-RPC++

最初，远程过程调用是通过HTTP的——使用XML来定义那些调用——其灵活而功能强大，因为它建立在两个广泛使用的标准之上。本书将使你相信——只要你需要——XML作为标准之

一，功能是非常强大的，因为它能在许多系统里存储或传送、接收数据。就标准的通用性而言，HTTP无疑是该领域的鼻祖。现在，几乎没有平台不使用针对该协议的服务器软件。

对于XML-RPC的局限性的讨论与它的构筑基础的一些概念无关，这点我们 110%的确信。更深入地对这些概念的实现进行分析，我们将看到需要比 XML-RPC更进一步。我们需要 XML-RPC++，我们需要 SOAP。

当然，你将注意到我们并不是只做了如 Stroustrup对C++的原始定义中所描述的“++”的工作。

那么，XML-RPC在什么地方需要改进呢？第一个方面——尽管包含的范围很广——与在XML-RPC里消息被标记的方式有关，因此，我们将从关注 XML-RPC处理数据的复杂开始，并考察SOAP是如何定位的。第二个方面与控制有关，将关注管理者如何管理什么能和什么不能被传送到他所管理的系统，因此，当关注完数据问题之后，我们将继续考虑 XML-RPC的“全部或没有”的问题是如何去解决的。

你可能会问既然SOAP如此之好，可为什么不将XML-RPC完全淘汰呢？原因是尽管XML-RPC只存在于一个较短的时间，但它已经吸引了大量的爱好者。全球的程序员都与 Dave Winer的Userland的例子交互，这些例子是使用XML-RPC的。并且，正如你在这一部分看到的，XML-RPC在实现上比SOAP简单，因此，它可能要存在一个较长的时间。

但是，如果某些人不能为他们的平台找到或建立一个SOAP模块，那么我将极力建议构造一个仅使用XML-RPC的新RPC系统。行业化的应用程序不可能使用XML-RPC——尽管它有许多爱好者。Microsoft是设计SOAP的关键，也非常热衷于推出它。不久，它可能将使其他一些服务器到服务器的技术黯然失色。

## 1. 数据

在数据传送问题上，XML-RPC的主要薄弱环节是冗长和数据类型。首先，我们将关注在XML-RPC里需要传送消息的数量，然后注意数据被分类的方式——通过这种办法，将看到整型和字符串型在数据传送上的不同。然后，关注SOAP是如何允许复杂结构和数组被传送的。最后，我们对SOAP不同于XML-RPC的特点加以总结。

### (1) 冗长

字典里对冗长的定义是：

冗长——过于详细的说明；使用了许多不需要的词语；啰嗦；词不达意等。

它详细说明了这个XML-RPC问题，要注意利用XML-RPC进行消息传送的核心是内容不应被遗失，即使是使用另外一套（更小的）符号集。

像你在前几段看到的那样，在XML-RPC里，每个值被一个<Value>元素定义。该元素里又存在另一个元素，这个元素指明该数据类型，接下来是它本身的值，例如：

程序清单 11-68

```
<value><int>7</int></value>
```



在这里有两个问题，第一个是从 XML 的角度去考察，数据是没有类型的。只是简单地由一个元素包含另一个元素，而这个元素又依次包含一个字符串，事实是直到开始在 XML-RPC 的层次上处理文档，才知道数据为整型——这可能对 DOM 是无效的。因为我们所拥有的只是一个被称作 `<int>` 的元素。在 XML 中关于数据分类的大量工作最近才开始，因此 XML-RPC 的初始创作者没有使用它。但是，正是因为我们开始使用它，故而我们将迅速地关注这个问题。

第二个问题是虽然对于一条或两条消息这种编码方法可能是较好的选择，但对于大量数据，它将变得十分冗长：

程序清单 11-69

```
<value>
  <array>
    <data>
      <value><int>4</int></value>
      <value><int>5</int></value>
      <value><int>6</int></value>
      <value><int>7</int></value>
    </data>
  </array>
</value>
```

因为可能的元素——`<int>`、`<string>`、`<array>` 等等——只有在高层才会理解，`<value>` 和 `<data>` 元素在此显得效果不佳。例如，数组能够被这样表示：

程序清单 11-70

```
<array>
  <int>4</int>
  <int>5</int>
  <int>6</int>
  <int>7</int>
</array>
```

## (2) 结构

对于 XML-RPC 的冗长问题，当我们对一个数据结构进行编码时将变得更为突出。回忆一下 XML-RPC 部分，`<struct>` 元素用于传递一个结构，并且每部分都有一个 `<name>` 和 `<value>` 对，例如：

程序清单 11-71

```
<value>
  <struct>
    <member>
      <name>Product</name>
      <value><string>Silver Clock</string></value>
    </member>
    <member>
      <name>Cost</name>
      <value><i4>87</i4></value>
    </member>
  </struct>
</value>
```

```
<name>Purchase Date</name>
<value>
  <dateTime.iso8601>19990912T02:53:02</dateTime.iso8601>
</value>
</member>
<member>
  <name>Purchase Order</name>
  <value><double>384793</double></value>
</member>
</struct>
</value>
```

相同的消息也能被这样传送(假定仍然使用比较冗长的类型消息)：

程序清单 11-72

```
<Order>
  <Product><string>Silver Clock</string></Product>
  <Cost><int>87</int></Cost>

  <PurchaseDate><dateTime.iso8601>19990912T02:53:02</dateTime.iso8601></PurchaseDate>
  <PurchaseOrder><double>384793</double></PurchaseOrder>
</Order>
```

这仍然在使用一些如 `< string >` 和 `< int >` 的元素来指明数据类型——稍后将解决该问题——其优点是它将结构直接映射到了 XML。这意味着如果我们把句法加到 XML-RPC 中去，那么我们能够直接在客户和服务器间传递 XML。这也表明 XML-RPC 如果不通过 `< struct >` 的整理，它无法做到这一点，因为该 `< struct >` 元素“隐藏”了数据的结构。SOAP 将处理这个问题，但是通过看看数据的分类将对这个问题有一个最好的解释。

### (3) 数据分类

SOAP 吸取了最近的 W3C 工作草案——XML 模式的第二部分——来帮助传送更多的将要处理的数据消息。像你在第 7 章里所看到的，该草案提供了大量“内建”的数据类型和一个可以附加新类型的设计。SOAP 使用这个特点加入了一些新的数据类型如变量和数组，很快我们将会提到。

使用 SOAP 以及在工作草案中定义的数据类型，结构的最后设计可能会是以下形式：

程序清单 11-73

```
<Order xmlns="W3C-Schemas-URI">
  <Product xsd:type="string">Silver Clock</Product>
  <Cost xsd:type="integer">87</Cost>
  <PurchaseDate xsd:type="timeInstant">19990912T02:53:02</PurchaseDate>
  <PurchaseOrder xsd:type="integer">384793</PurchaseOrder>
</Order>
```

这可能很像远程过程调用要转换的 XML，但是它比起 XML-RPC 的冗长要优良得多。

但结构也伴随一个问题，即可能没有关于一个对象的名字或者参数的任何信息，或者名字可能毫不相干。SOAP 允许我们使用另外的缩写，或联想 XML-RPC 技术：

程序清单 11-74

```
<Order>
```

```
<string>Silver Clock</string>
<integer>87</integer>
<timeInstant>19990912T02:53:02</timeInstant>
<integer>384793</integer>
</Order>
```

决定何时使用这个技术和何时使用类型属性是非常直接的。绝大多数时间里你能用你喜欢的一个；对于SOAP规范，简单地说，只要某元素的名字不足以唯一指明数据类型，那么就使用类型属性，下面是浮点型的例子：

程序清单 11-75

```
<velocity xsd:type="float">15.5</velocity>
```

而这是一个简单的字符串：

程序清单 11-76

```
<velocity>15.5</velocity>
```

尽管这两个技术——对数据类型设置元素名字或者使用 `xsd:type` 属性——可达到同样的目的，但使用 `xsd:type` 属性的优点在于我们能够从解析器中得到正确的数据类型。例如，如果我们要求解析器来读 `<PurchaseOrder>` 元素中的内容，那么将得到一个时间和日期值，而不是一个简单的字符串，当然这需要XML解析器建立所需元素的类型版本，像第7章讨论的那样。

#### (4) XML模式第二部分的扩展

有效的元素值——例如字符串和整型——与在XML模式第二部分工作草案中所列出的那些相同：SOAP又加入了另一个——变量——来指明可能包含了许多数据类型的元素。尽管Microsoft的Visual Basic广泛使用了变量——并且VBScript除了变量不用其他数据类型——但SOAP的变量仅对这样一些数组有用，即这些数组可能包含了混合的值的类型。下面这个例子中有一个二维数组，其中元素0包含了一个整数，而元素1包含一个字符串：

程序清单 11-77

```
<ArrayOfvariant xsd:type="u:variant[2]">
  <variant xsd:type="int">23</variant>
  <variant xsd:type="string">some string</variant>
</ArrayOfvariant>
```

SOAP也将该草案加以扩展，即加入了数组，事实上，它提供了用于编码数组的两种方法，第一种方法需要将这些数组内容的信息编码成元素的名字，例如：

程序清单 11-78

```
<ArrayOfinteger xsd:type="u:int[4]">
  <integer>4</integer>
  <integer>5</integer>
  <integer>6</integer>
  <integer>7</integer>
</ArrayOfinteger>
```

该例摘自于SOAP规范，但是在XML模式第二部分的工作草案中没有称为int的数据类型，SOAP数据类型均来自于该草案。这就意味着int类型将来自于自定义类型。SOAP标准并没有明确说明这些自定义类型消息如何被传送，明白地讲，SOAP消息包并没有一个模式。也就是说，每一个消息包里，该模式应该被动态创建，因为数据类型可能是复杂的对象。

该元素的名字被设置为ArrayOf，后面是数组里每一个元素的基本类型。该元素也有它的类型属性设置来指明该数组的大小，在该例中，u:int[4]意味着这是一个包含四个整型的数组。

第二种方法是使用一个已命名的元素指明一个数组：

程序清单 11-79

```
<Authors xsd:type="u:author[4]">
  <author>Stephen</author>
  <author>Kathie</author>
  <author>Eric</author>
  <author>Peter</author>
</Authors>
```

这里，数组的每一个元素是一个作者。注意你选择哪种技术并不取决于数组的内容。下面是一个合法的例子：

程序清单 11-80

```
<ArrayOfauthor xsd:type="u:author[4]">
  <author>Stephen</author>
  <author>Kathie</author>
  <author>Eric</author>
  <author>Peter</author>
</ArrayOfauthor>
```

这里使用了ArrayOf处理authors，而不是整型（integers）。使用哪一种方法取决于你使用何种形式将数组放在另外的消息的上下文之中，而这些消息都是正在传送的。因为这也关系另外的消息类型。

在结束对数组讨论之前，我将举两个的例子，从中你能感受到SOAP超出XML - RPC的灵活性。

下面是一个二维字符串数组：

程序清单 11-81

```
<ArrayOfstring xsd:type="u:string[2,3]">
  <string>row 1 column 1</string>
  <string>row 1 column 2</string>
  <string>row 1 column 3</string>
  <string>row 2 column 1</string>
  <string>row 2 column 2</string>
  <string>row 2 column 3</string>
</ArrayOfstring>
```

尽管数组里的所有元素相继排列，但它们应被解释成 2行，每行包括3列。一个这样的二维数组与一个由数组组成的数组是不同的，后者可能为如下形式：

## 程序清单 11-82

```
<ArrayOfArrayOfstring xsd:type="u:string[]" [2] ">
  <ArrayOfstring xsd:type="u:string[3] ">
    <string>one</string>
    <string>two</string>
    <string>three</string>
  </ArrayOfstring>
  <ArrayOfstring xsd:type="u:string[2] ">
    <string>one</string>
    <string>two</string>
  </ArrayOfstring>
</ArrayOfArrayOfstring>
```

注意因为这两个数组是分离的，故而大小不同也是合理的。

数组元素也能指出它们在数组中被安排的位置，在这个例子里，我们能向一个序列里插入一个数组元素：

## 程序清单 11-83

```
<ArrayOfstring xsd:type="u:string[3] ">
  <string position="[1]">two</string>
  <string position="[2]">three</string>
  <string position="[0]">one</string>
</ArrayOfstring>
```

指定位置的能力能够允许一个数组的某个部分被传送，在这个例子里有另一个元素没有发送：

## 程序清单 11-84

```
<ArrayOfstring xsd:type="u:string[3] ">
  <string position="[0]">one</string>
  <string position="[2]">three</string>
</ArrayOfstring>
```

如果要指定一个连续的元素设置，这些元素开始于数组里的某一个位置，但是又不想指定初始元素，那么可以指定一个偏移量，并从这里开始装载这些元素：

## 程序清单 11-85

```
<ArrayOfstring xsd:type="u:string[3] " offset="[1]">
  <string>two</string>
  <string>three</string>
</ArrayOfstring>
```

如果觉得这里有些不对，那么请不要忘记数组元素是从零开始计数的，偏移量也是如此。

#### (5) 其他的改进

SOAP已经解决了数据类型的问题，它已经加入了更好的数组处理，并且现在允许 XML作为XML被传送，而不是被处理成冗长的结构。还有另外的什么吗？

好了，另一个加入到SOAP中的灵巧的特点是它允许一个值被许多地方所引用，传送下面这

个数据结构本质上并没有错误：

程序清单 11-86

```
<Authors>
  <author>
    <name>Steve</name>
    <City>Birmingham</City>
  </author>
  <author>
    <name>Fred</name>
    <City>Birmingham</City>
  </author>
</Authors>
```

你将发现如果有几百个作家碰巧都住在 Birmingham，那将是一个多么庞大的消息。SOAP允许一些被多次涉及的元素以如下形式出现：

程序清单 11-87

```
<Authors>
  <author>
    <name>Steve</name>
    <City href="#1" />
  </author>
  <author>
    <name>Fred</name>
    <City href="#1" />
  </author>
</Authors>

<City id="1">Birmingham</City>
```

如果数据只能被一次引用则被称作单一引用（single - reference），该情况下，任何数据没有一个id属性。多引用（multi - reference）技术是很有用的，例如，前面所出现的一些由数组组成的数组可有如下表达方式：

程序清单 11-88

```
<ArrayOfArrayOfstring xsd:type="u:string[] [2]">
  <ArrayOfstring href="#array-1"/>
  <ArrayOfstring href="#array-2"/>
</ArrayOfArrayOfstring>

<ArrayOfstring id="array-1" xsd:type="u:string[3]">
  <string>one</string>
  <string>two</string>
  <string>three</string>
</ArrayOfstring>

<ArrayOfstring id="array-2" xsd:type="u:string[2]">
  <string>one</string>
  <string>two</string>
</ArrayOfstring>
```

注意这种多引用数组必须使用 `ArrayOf` 语法，它们也必须在顶层。在 SOAP 标准里顶层元素被称作独立元素；而包含于其他元素的元素被称作嵌入元素。像使用 XML - RPC 一样，SOAP 有一个包含信息的包，这些包被传送到一个过程或从此返回。顶层是一个包。

下面这个例子是不允许的，因为它打破了这些规则：

程序清单 11-89

```
<ArrayOfArrayOfstring xsd:type="u:string[]" [2]">
  <ArrayOfstring href="#array-1"/>
  <ArrayOfstring href="#array-2"/>
  <!--This should be at the top level -->
  <ArrayOfstring id="array-1" xsd:type="u:string[3]">
    <string>one</string>
    <string>two</string>
    <string>three</string>
  </ArrayOfstring>
</ArrayOfArrayOfstring>

<!-- This should use ArrayOf -->
<MyArray id="array-2" xsd:type="u:string[2]">
  <string>one</string>
  <string>two</string>
</MyArray>
```

数组 `array - 1` 是错误的，因为它应该在顶层。而数组 `array - 2` 的错误在于，如果它想在其他地方被引用，那么应该使用 `ArrayOf` 语法。

#### (6) 数据处理扩充的小结

SOAP 加入了一些高级的 XML - RPC 数据处理能力。一些可能成为 XML 应用程序的标准的数据类型的加入使 SOAP 有了一个坚实的基础。同样，一些技术，例如稀疏数组和多引用元素，对于减少消息的传送量很有帮助。最后，像 XML 一样进行传送，而不是编码成为名字 / 值对，使数据格式变得更加灵活。

既然我们已经使数据处理能力大为提高，那么就再关注一下调用方法。

#### 2. 调用

SOAP 加入了大量的产生调用的方法。特别是它允许站点管理者很好地控制服务器什么消息能处理和什么不能处理。更多的消息在消息包的头部提供，同样增加了使用 HTTP 动词的灵活性。

对于远程过程调用使用 HTTP 标准作为传输机制存在的一个问题是，对以下两种请求难以区分。即一些请求可能是某些高级通信过程的一部分，而另一请求可能是低级的或是恶意的。像我们使用 XML - RPC 时所看见的，POST 动词用于传送这样一些消息，即该消息将决定哪个过程将被执行。但是 POST 也被用于填写一些表格，它甚至被用于文档的上载。

在某些情况下，对 RPC 使用 POST 动词也许并不麻烦。例如；安全可能不成问题，或者——更可能的情形是——可能有这样一种情况，开发商想在一个服务器上实现 RPC 功能，但是我们没有向该服务器加入新动词或报头消息的能力。因此，我们总是想首先通过 POST 来尝试和传送 RPC 消息。



但是，由于某些原因管理者可能采用防火墙技术阻止 POST 表单到达服务器。他们尽管想允许 RPC 调用通过，但并不想允许任何其他 POST 通过。区别 XML-RPC 请求是一个被提交远程过程调用而不是非 RPC POST 的唯一方法是通过考查实际被传送的数据。但是这意味着防火墙要有更多的工作，因为它需要防火墙能识别有效负载，并解析 XML。如果对于 SOAP 使用防火墙，那么为什么不能使用 BizTalk，WebDAV，或者另外别的什么协议？一方面新的软件不停地被加入防火墙，一方面又设计一些新的标准，这对于维护防火墙是不可能的。在这种状态下，我们将需要一个动词来区分 POST。

XML-RPC 的强大功能在于它使用 HTTP 作为它的传送机制。这是非常重要的，因为这就意味着任何防火墙都将允许我们的过程调用通过——并不需要像使用 Corba 或者 DCOM 那样设置特别的端口。但是，XML-RPC 的一个问题是对自身的描述不够精确。它通过 “text/xml” 内容形式来使用动词 POST，使对粒度的控制变得比较困难。

### (1) HTTP 的扩展

向 HTTP 加入新的动词是不可能也是不希望的，设想一下如果每一个人都能简单地把它们所喜欢的动词随意加入的情况吧！防火墙和服务器管理者将无法知道特定请求的目的，无法提供可靠的过滤。如果两个不同的服务器都想发明一个如 RENAME 的新动词，我们如果区分它们？

即使需要，也不能加入新的动词，应该在报头提供更多的消息。HTTP 扩展模式提供一种机制用于加入附加报头和对于 HTTP 请求加入动词信息，希望能在不分析请求本身的情况下，被传送的消息自然状态能被推断出来。

对于 HTTP 扩展的一个完整描述参见：

<http://www.w3.org/Protocols/HTTP/ietf-http-ext/>

在一条消息被处理前，扩展模式允许一个报头列表被一个服务器或者代理所理解。同样的方法，你能够指定一个功能调用所需要的参数。如果函数能被运行，它可能需要一个起始参数和一个终结参数，两者缺一，即为非法调用。使用扩展模式我们能够指定一些相似的过程。如果服务器不理解任何强制性参数，那么它必须拒绝全部调用。

那么对于指定的消息又如何呢？首先无论你想使用什么 HTTP 动词，必须在它前面加前缀 M——如果出现了任何强制性的报头。第二个要求是这些强制性的报头使用同 XML 命名空间相同的机制被编组。下面扩展模式的例子有助于问题的理解：

### 程序清单 11-90

```
M-PUT /a-resource HTTP/1.1
Man: "http://www.copyright.org/rights-management"; ns=16
16-copyright: http://www.copyright.org/COPYRIGHT.html
16-contributions: http://www.copyright.org/PATCHES.html
Host: www.w3.org
Content-Length: 1203
Content-Type: text/html
```

在我们关注扩展之前，对于 HTTP 标准部分的含义应加以更新：

- 第一行的结尾指明我们遵从 HTTP 1.1 版本的要求。

- 主机报头说明将接收请求的服务器是 `www.w3.org`。
- 第一行的中间有要求的 URL，格式为 `/a-resource`。
- 请求的长度是 1 203 字节（`Content-length`），请求的 MIME 类型是 HTML。

现在让我们看一下附加的扩展模式部分的信息：

- 因为第一行的动词有 M-前缀，那么这肯定是一个 Man：报头。
- 一旦 M-被删除，那么请求将使用 HTTP 动词 PUT。
- Man:报头用一个 URI 来识别一个命名空间和用 16 来命名它（第二行）。
- 16 可以用来定义所有必需的用于 PUT 的报头。
- 我们有两个强制性报头——`copyright` 和 `contributions`。

你所看到这里允许任何人使用 `copyright` 报头为他们自己的目的服务，只要一个唯一的命名空间前缀用来避免混淆另外的 `copyright` 报头和我们在该例中所使用的 `copyright` 报头——它们可能有许多。

## (2) SOAP 的强制性报头

让我们看一下 SOAP 所需要的报头。事实上在最新的规范里目前仅只有一个，你可能比较喜欢这种方式。以前的 SOAP 标准有一个接口名和一个方法名。但是，事实上，方法名前有一个命名空间前缀——在这个例子里为 `Some-Namespace-URI#GetLastTradePrice`——这可能会使 SOAP 作者认为接口名都是多余的。

一个 SOAP 请求可能的报头形式如下：

程序清单 11-91

```
M-POST /StockQuote HTTP/1.1
Host: www.stockquotesserver.com
Content-Type: text/xml
Content-Length: nnnn
Man: "urn:schemas-xmlsoap-org:soap.v1", ns=01
01-SOAPMethodName: Some-Namespace-URI#GetLastTradePrice
```

用 M-前缀来表明已经提供了强制性报头。这将使一个能够理解 HTTP 扩展模式的服务器（或者一个防火墙）来寻找一个称为 Man 的报头。一旦找到这个报头，处理软件将发现一个唯一的命名空间——这个命名空间必须设置为 `urn:schemas-xmlsoap-org:soap.v1`，如该例所述——这个命名空间与前缀 01 相连。因此有一个强制性的称为 `SOAPMethodName` 的报头，它位于命名空间 `urn:schemas-xmlsoap-org:soap.v1`，除非有服务器能识别该报头，否则，服务器将不处理该消息。

现在已经为一个站点管理者提供了一个过滤请求的可能性，如果管理者想阻止任何一个请求到达服务器（依自身权限），那么可以设置不允许在命名空间里报头为 `urn:schemas-xmlsoap-org:soap.v1` 的消息通过。如果需要滤出特定的消息，那么可以设置寻找报头 `urn:schemas-xmlsoap-org:soap.v1:SOAPMethodName`。没有检查 `SOAPMethodName` 的关键在于其他人使用报头来表明完全不同的其他内容。使用命名空间，混淆就消失了。

## (3) XML-PRC 简化了什么

如果我们不介意失去那些使 XML-PRC 变得如此成功的特点——你需要做的仅是实现或处理一个 POST 指令来实现远程的服务器控制。那么，能使用传统的 POST 来执行调用，如果还需要

一个等级划分。像以前的调用现在可有如下表述：

程序清单 11-92

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml
Content-Length: nnnn
SOAPMethodName: Some-Namespace-URI#GetLastTradePrice
```

其中，同在 SOAP 报头里用于告诉我们哪种方法被执行的前缀在这里没有了。

SOAP 标准说明在这里显示的格式，首先应该被客户的软件试验——那是一种使用某方法的 POST，该方法在 SOAPMethodName 报头里被激活。如果 POST 被服务器所接收，那么我们的调用与 XML-RPC 看起来并无不同，所有改变在于有一个报头来提示何种方法并被执行。

SOAP 实际上并不需要报头，因此你可以像使用 XML-RPC 一样提交一个简单的 POST。这就是为什么防火墙的管理者非常喜欢设置他们的系统使用 POST 动词和 XML 的内容类型来拒绝消息包，如果你所作的全部就是检查 SOAPMethodName 是否作为一个报头而出现，那么你将让一个 SOAP 通过。

如果一个管理员想实施更多的控制，那么他们能够通过检查 SOAPMethodName 的报头前缀来拒绝这些 POST。当然，这样做也有可能拒绝一些完全不相关的或者碰巧也使用 SOAPMethodName 作为报头名字的非 SOAP 请求，但这时系统对此什么也不做。拒绝这些请求的服务器设置应该向客户指明，如果某方法是不允许的那么将以 405 做为一个方法不被允许的返回值。

如果客户接收到这个错误代码，那么并不等于说明 SOAP 不被支持。我们已说过，可能服务器在允许该调用通过之前需得到更多的信息。在一个 405 事件里，客户将再次提交请求，但这一次使用 M-POST 格式——包含所有强制性报头。

如果你熟悉 SOAP 的 0.9 版本，那么将注意到你必须执行 POST 和 M-POST 的顺序已经被颠倒了。

#### (4) 方法调用的小结

我希望通过一些有关请求的简单例子能将所有这些阐述清楚。第一个来自于 SOAP 标准本身，并使用普通的 POST 对服务器产生一个的调用，就像使用 XML-RPC 所做的那样。在下一部分讲解 SOAP 服务器的执行时，将解释通过的数据结构，现在仅讲报头消息：

程序清单 11-93

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml
Content-Length: nnnn

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
  <SOAP:Body>
    <m:GetLastTradePrice xmlns:m="Some-Namespace-URI">
      <symbol>DIS</symbol>
```

```
</m:GetLastTradePrice>
</SOAP:Body>
</SOAP:Envelope>
```

SOAP标准里报头并不一定非要包含方法的名字（尽管应该这样）。遗憾的是，对于防火墙这样做将不能激活方法，因为它并不理解请求的自体。它要么允许所有的方法通过，要么要求提供报头消息。像我们看到的，在 HTTP 扩展模式里 M-XXX 扩展的目标在于提供一种机制来处理那些对一定报头的要求。该模式提供了所有的 HTTP 动词的一个版本，这些 HTTP 动词在正常情况下没有什么不同，但只有在出现强制性的报头消息时，它们才生效。

现在来看一下如果使用一个强制性 POST 或 M-POST 再次提交，我们的请求将会怎样。如果一个服务器或者代理被设置或使用 XML 的内容类型来忽略 POST 时，M-POST 将被使用。我们的第二个请求将如下所示，其中包含了传统的报头消息：

程序清单 11-94

```
M-POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml
Content-Length: nnnn
Man: "urn:schemas-xmlsoap-org:soap.v1", ns=01
01-SOAPMethodName: Some-Namespace-URI#GetLastTradePrice

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
...
</SOAP:Envelope>
```

Man：报头是扩展模式的一部分，指明了一个用于将所有报头编组的前缀，这些报头又是强制性请求的一部分。在这里，我们说任何以“01-”开始的报头均是相同的强制性组的一部分。像我们从前所说的，“01”功能非常像在 XML 中的命名空间前缀。事实上，在 SOAP 有效载荷里，它必须被设置成 SOAP 包中的 SOAP 命名空间 URI 相同的值。

现在防火墙高兴了。它能看到我们将怎样处理请求，也能决定是否允许它通过。通过设置防火墙来拒绝那些有“text/xml”内容类型的 POST，防火墙能迫使 SOAP 用户使他的所有方法调用均使用 M-POST，这些请求保证有对于防火墙来说必需的报头消息，用来作为一个消息是否能够通过的依据。

SOAP 标准并没有完整地描述我在这里所做的每件事情。它只是展示了一个情况，防火墙拒绝了那些有“text/xml”内容类型和有 SOAPMethodName 类型的报头的 POST。但是，因为该标准说明 SOAPMethodName 报头应被提供，而不是必须提供，故而提供它并不是一件有用的事，因为没有 SOAP 方法报头的请求也将允许通过。

这意味着，所有的 XML POST 必须使用 M-POST。将来这会产生一个问题，因为这种结合对于 SOAP 请求来说不是唯一的。例如，一个服务器可能支持 BizTalk 或者 WebDAV，也支持 SOAP，并且它们都能够使用带有“text/xml”内容类型的动词 POST。防火墙通过迫使 SOAP 使用强制性方法以至 WebDAV 请求被完全阻止。

SOAP 的 0.9 版本使用“text/xml-SOAP”内容类型来解决这个混淆，但是这已经很少提及。

作为一个有用的分辨器，我认为它将成为最终正确的选择，因为对内容类型的提出并不是随意的。0.9版本也有另一个解决方案，它对SOAP请求报头进行限制。它可能成为一个对XML-RPC爱好者的让步。即允许一个SOAP请求与一个简单的POST一样，在起始部分什么也没有。但是如我所说，这将无法分辨一个请求是否为SOAP请求。

#### 11.4.2 实现

我们将关注SOAP实现中较XML-RPC的高明之处。现在，不再孤立地定义SOAP规范，将逐步建立了一个SOAP客户和服务。在这个过程中，将介绍该标准中的一些关键点。

这一节的代码包括在网上本书的可下载代码部分。

##### 模块结构

有两个我们必需建立的模块——一个用于客户系统，另一个用于服务器系统。因为有某些函数同时在客户和服务上执行——例如建立有效载荷——那么我们也需要创建一个普通处理模块。

##### (1) 客户模块

我们将从显示一个调用的代码开始。尽管远程调用的思想是在另一个服务器上执行一个函数，我们仍希望尽可能地像一个正常的功能调用那样实现远程调用。例如，一个用来在数据库里得到第一个作者的名字本地调用可能有如下形式：

程序清单 11-95

```
name = getAuthorName(1)
```

那么我们希望远程调用有如下的形式：

```
name = RemoteCall("ServerName", "getAuthorName", 1)
```

如果你使用某种语言如C或者C++（请参见本章后面部分的C++实现例子）来实现SOAP客户和服务，那么将很容易通过参数数量的变化来处理调用。但是，我们将演示用ASP来建立一个SOAP实现工具，因此仅能够传送预先给定数量的参数。为了避免该种限制，我们总是传送一个参数，它是一个数组。该数组将包含真实的参数。这样，参数的多少就可如你所愿。

一个客户请求的例子如下：

程序清单 11-96

```
<!--#include virtual="X-Port/SOAP/SOAP.asp" -->
<!--#include virtual="X-Port/SOAP/SOAPClient.asp" -->

Dim paramList(2), oRet

if not Request.form("iAuthor") = "" then
    paramList(0) = CInt(Request.form("iAuthor"))
    paramList(1) = 1
    iRet = SOAP("http://server/SOAP/myServer.asp", _
        "myNamespace", _
        "getAuthor", _
        paramList, _
```

```

        oRet)
    ' Check the return status and then maybe do something with the returned value
end if

```

在这个例子里我们向远程过程（getAuthor）传送两个参数。从该调用的任何返回值将放在变量oRet里，任何成功或失败的状态信息将放在变量iRet里。该远程过程被函数SOAP（）激活，如果你正在实现自己的系统，那么能够使用任何你认为有意义的名字。

在远程服务器上这个过程做了些什么呢？它返回了一批作者名，他们的ID都是大于或等于第一个参数的。在这个例子里，该参数的值来自于一个可变的表单。至于确切地返回作者的数量，将通过第二个参数设置——在这里用“1”。这个程序将在后面被用到。

你可能注意到这里有两个include文件，并想知道为什么我们不包含SOAPClient.asp中的基本模块SOAP.asp，这样就而只需要包含一个。在绝大多数情况可以这样做，但是，后面我将向你展示这样一种情形，即我们想建立一个既是客户又是服务器的应用程序。在这种情况下，将使基本模块被包含两次。因此，最好将此模块独立于客户和服务器模块之外，是否包含，依需要而定。当然，如果你使用的是一个比ASP更高级的环境，那么你能使用条件包含来避免这个问题。

## (2) 调用

让我们看一看当执行调用时发生了什么。下面这段代码在SOAPClient.asp内：

程序清单 11-97

```

function SOAP(sURL, sNamespace, sMethod, arrParam, ByRef oRet)
On Error Resume Next
' Create and send the XML request
Dim oSOAP : Set oSOAP = doRemoteCall(sURL, sNamespace, sMethod, arrParam)
If Err.Number <> 0 Then
    Dim sTemp : sTemp = Err.Number & ": " & Err.Description
    Err.Clear
    Err.Raise 999, "SOAP", "Failed to call remote server (" & sTemp & ")"
Else
    ...
End If
end function

```

该调用被传送到另一个函数——doCall（）——它做实际的处理过程。SOAP（）过程必须包含它自己，让调用者知道处理是成功还是失败。

那么远程调用实际做了什么呢？这是一个过程的开始，如你所见，首先的任务是准备一个调用的有效载荷：

程序清单 11-98

```

function doRemoteCall(sU, sNS, sM, arP)
On Error Resume Next
' Create the request body from the method name and array of parameters

```



```

Dim oPL : Set oPL = CreateCallPayload(sNS, sM, arP)
If Err.Number <> 0 Then
    Dim sTemp : sTemp = Err.Number & ": " & Err.Description
    Err.Clear
    Err.Raise 999, "SOAP", "Failed to create payload (" & sTemp & ")"
Else
    ...
End If
end function

```

### (3) 有效载荷

创建一个有效载荷是在客户和服务端双方都需要做的工作，因此，有效载荷函数存在于通用模块——SOAP.asp内。

像使用XML-RPC，有效载荷由一些实际的数据组成，这些数据是我们将要传送的。它们被包含在一个非常严格的消息包里以保证这些数据能够被正确地抽取。有效载荷的实际格式随着对有效载荷的不同请求情况而不同。例如，承担一个方法调用消息的有效载荷与传送调用结果的有效载荷不同。SOAP为调用、返回和错误报告提供不同的有效载荷。

尽管有三种形式的有效载荷，但它们还是有一些共同的特征：

- 它们都包含于叫做 < Envelope > 的元素。这个元素必须属于 SOAP命名空间。
- < Envelope > 元素顺序包含一个 < Body > 元素，它也是在 SOAP命名空间内。这些条件意味着总是有如下这种模式：

#### 程序清单 11-99

```

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
  <SOAP:Body>
    ...
  </SOAP:Body>
</SOAP:Envelope>

```

现在能够编码有效载荷函数，它包含任何已经定义过的子元素，不论是一个调用、一个响应或是一个错误：

#### 程序清单 11-100

```

function CreatePayload(oSubElem)
    Dim oPL : Set oPL = Server.CreateObject("Microsoft.XMLDOM")
    oPL.async = false

    Dim oRoot : Set oRoot = oPL.createNode(1, "SOAP:Envelope", NS_SOAP)
    oRoot.setAttribute "xmlns:xsd", NS_DATATYPES
    oRoot.setAttribute "xmlns:dt", NS_DATATYPES

    Dim oBody : Set oBody = oPL.createNode(1, "SOAP:Body", NS_SOAP)

    Dim oPI : Set oPI = oPL.createProcessingInstruction("xml", "version="&"1.0"&"")

    oPL.appendChild(oPI)
    oBody.appendChild(oSubElem)
    oRoot.appendChild(oBody)

```



```
oPL.appendChild(oRoot)

Set CreatePayload = oPL
end function
```

注意 < Envelope > 元素也包含对于数据类型的命名空间定义，像早些时候讨论的那样。正常情况下，仅需要加入 xsd 命名空间，但作为一个过渡性的方法，也为 XML-DR 加了命名空间，因此能从 Microsoft XML DOM（在第7章有全面的解释）得到分类的数据。这个元素可以包含任意数量的命名空间定义，只要你想要。并且它也包含另外的属性，这些属性也满足命名空间的要求。

我们已经有了创建一个有效载荷的函数，该有效载荷又是包含了任意子元素的，因此需要一个一般过程来创建那些子元素。一个用于调用，一个用于对调用的响应，一个用于错误指示。

#### (4) 方法调用和响应

调用和它们的响应使用方法的名字，该名字位于传输的命名空间。响应将 “ Response ” 附加到方法的名字上，因此我们可有如下调用：

程序清单 11-101

```
<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
  <SOAP:Body>
    <m:getAuthor xmlns:m="Some-Namespace">
      ...
    </m:getAuthor>
  </SOAP:Body>
</SOAP:Envelope>
```

得到响应如下：

程序清单 11-102

```
<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
  <SOAP:Body>
    <m:getAuthorResponse xmlns:m="Some-Namespace">
      ...
    </m:getAuthorResponse>
  </SOAP:Body>
</SOAP:Envelope>
```

创建一个调用有效载荷的函数如下：

程序清单 11-103

```
function CreateCallPayload(sNamespace, sMethod, arP)
  Dim oSubElem, i
  Set oSubElem = oDOM.createElement(1, "m:" & sMethod, sNamespace)
  for i = 0 to UBound(arP) - 1
    addChild oSubElem, "", arP(i)
  next
  Set CreateCallPayload = CreatePayload(oSubElem)
end function
```

调用响应函数只需要调用该调用有效载荷函数，使用被改动的方法名字：

程序清单 11-104

```
function CreateCallResponsePayload(sNamespace, sMethod, arP)
  Set CreateCallResponsePayload = CreateCallPayload(sNamespace, _
                                                    sMethod & "Response", arP)
end function
```

一个调用的有效载荷相继拥有所有的参数。addChild()过程在通用模块SOAP.asp中，用于将节点加到输出上。addChild()的第二个参数一般是将被给出节点的名字，但是在本例中，我们把它空起来以表明我们想用数据类型名字作为节点名字（addChild()函数将在后面讨论）。

注意我们使每一个参数“匿名”便于使用，在标准里并不需要这样做。下面这个从标准规范里得来的例子展示了将调用编码的另一种方法，这次使用了被命名的参数：

程序清单 11-105

```
<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
  <SOAP:Body>
    <m:GetLastTradePriceDetails xmlns:m="Some-Namespace-URI">
      <Symbol>DEF</Symbol>
      <Company>DEF Corp</Company>
      <Price>34.1</Price>
    </m:GetLastTradePriceDetails>
  </SOAP:Body>
</SOAP:Envelope>
```

但是，使用一个简单的ASP实现，例如我们正在建造的，比起编码参数名称和其值要麻烦得多（定义和命名接口的系统正向前发展，下面将要讨论）。因此使用数据类型来命名参数是简洁明了的。在我们的系统里，如果有一个包含三个元素的数组，并且有与刚才给出的例子里相同的值得得到：

程序清单 11-106

```
<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1"
  xmlns:xsd="Some-Schema-Namespace-URI">
  <SOAP:Body>
    <m:GetLastTradePriceDetails xmlns:m="Some-Namespace-URI">
      <String xsd:type="string">DEF</String>
      <String xsd:type="string">DEF Corp</String>
      <Float xsd:type="float">34.1</Float>
    </m:GetLastTradePriceDetails>
  </SOAP:Body>
</SOAP:Envelope>
```

## (5) 方法参数

在SOAP规范叙述了参数可以“输入”和“输入/输出”，这就意味着它们能够赋值，并做为信息被传送到一个一般过程，该值也可以用于保存函数的结果。但是，在SOAP里没有把一个函数定义成这样的模式——哪些参数用于输入，哪些参数还能返回数据。

这意味着对于一个客户实现，没有办法发现在一个调用过程中哪些参数是需要的，对于服

务器来说也没有办法在执行方法前检查传送的参数。当然，你能够在自己的每个函数里检查发现是否接收到了所期望的参数。但是理想化的 SOAP 服务器层应该把这些对你隐藏起来。

尽管这种检查是非常有用的，但这并不是 SOAP 一个部分。原因有二，其一是它将使 XML-RPC 爱好者们感到苦恼，你将不再能够快速地将客户或服务模块发布到任何平台，现在将不得不检查参数的名字和那些参数的数据类型。但是如果客户和服务模块均在你的控制之下，那么就可不必如此费心，而可以保证这些参数是正确的。

第二个理由是已经有一些初始化的工作存在指定了一个函数的“足迹”。它们中的一些来自于软件设计领域。那里，对象和它们的方法可以自动地被多种工具所定义。另外一些是新的，专用于如 SOAP 和 XML-RPC 的以 XML 为基础的协议。一个这样的初始化工作是 Component Description Language（组件描述语言：CDL），它来自于 DevelopMentor，在 <http://www.develop.com/soap/cdl.htm>，它允许指定参数的类型以及函数的返回值。

#### (6) 出错响应

最后的有效载荷类型是针对错误的。如果 SOAP 层能够检查出错误，那么一个出错响应就被返回。在各种层次上都有可能发生错误，可能是一个消息包的错误，或者传送的 XML 的错误，或者在有效载荷里方法的名字可能与报头里的不匹配。SOAP 本身并不能总是检查出错误，例如，如果一个错误存在于被传送的 XML 中，或者 M-POST 在没有强制性的参数被传送的情况下被使用，那么直到这个消息包被 SOAP 处理，错误才被发现；另一方面，如果错误的命名空间被用于特定的 SOAP 元素，那么 SOAP 将能够识别该错误，并进行处理。

SOAP 对它能够识别的错误的返回指示如下所示：

程序清单 11-107

```
<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap:v1">
  <SOAP:Body>
    <SOAP:Fault>
      <SOAP:faultcode>200</SOAP:faultcode>
      <SOAP:faultstring>SOAP Must Understand Error</SOAP:faultstring>
      <SOAP:runcode>1</SOAP:runcode>
    </SOAP:Fault>
  </SOAP:Body>
</SOAP:Envelope>
```

这个例子来自于 SOAP 规范本身，仅有的不同在于在 `<faultcode>`、`<faultstring>` 和 `<runcode>` 开始于一个简单的命名空间前缀。该标准只要插入子元素，往往就省略名字前缀——这里是 `<SOAP:Fault>`——但是使用 SOAP 前缀并与 SOAP 有关的一些代码除外。

`<fault>` 结构中几个部分的含义如下：

- `<faultcode>` 错误代码号，如 100 代表调用使用了不被支持的 SOAP 版本。
- `<faultstring>` 是错误的字符串表示，对错误 100 它显示“Version Mismatch”（版本不匹配）。
- `<runcode>` 被用于指示是否——尽管错误被报告——函数被送到了应用程序，值的范围是 0、1 和 2，分别说明“可能”、“没有”和“有”。

这三个元素是必须的，但是也可能有第四个元素——`<detail>`——它包含该应用程序想返回的任何内容，例如关于为什么调用失败的详细描述。例如，如果被调用的应用程序失败，那么

它将返回 SOAP 错误 400（错误代码），但是并没有告诉调用者更多；该应用程序就能通过使用这个参数提供更多的信息：

#### 程序清单 11-108

```
<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
  <SOAP:Body>
    <SOAP:Fault>
      <SOAP:faultcode>400</SOAP:faultcode>
      <SOAP:faultstring>SOAP Application Failed Error</SOAP:faultstring>
      <SOAP:runcode>1</SOAP:runcode>
      <SOAP:detail>
        <errorcode>1001</errorcode>
        <message>No such record</message>
      </SOAP:detail>
    </SOAP:Fault>
  </SOAP:Body>
</SOAP:Envelope>
```

一个出错的有效载荷在如下函数中得到准备：

#### 程序清单 11-109

```
function CreateFaultPayload(iFaultcode, sDetail)
  Dim oSubElem
  Set oSubElem = oDOM.createNode(1, "SOAP:Fault", NS_SOAP)
  addChild oSubElem, "faultcode", iFaultcode
  Select Case iFaultcode
    Case 100
      sFaultstring = "Version Mismatch"
    Case 200
      sFaultstring = "Must Understand"
    Case 300
      sFaultstring = "Invalid Request"
    Case 400
      sFaultstring = "Application Faulted"
  End Select
  addChild oSubElem, "faultstring", sFaultstring
  addChild oSubElem, "runcode", iRuncode
  if sDetail <> "" then
    addChild oSubElem, "detail", sDetail
  end if
  Set CreateFaultPayload = CreatePayload(oSubElem)
end function
```

该函数仅需要一个错误代码和一个描述串来创建一个 <Fault> 结构，出错串从代码中创建，并且运行代码取决于一个全局变量。注意描述串仅在它确实被设置后才能被返回。

当使用所有有效载荷函数时，addChild（）被用于在树中创建节点，我们现在转到这个函数上来。

#### (7) addChild（）

在前面的部分里简要地对 addChild（）进行了说明，在此我们将更加详细地对它进行探讨，因为它正是 SOAP 系统的核心。addChild（）的目的是向一个 XML 树加入节点，节点的使用遵守 SOAP 标准的数据规则，例如，如果有一个值为 7 的整数，可能创建如下代码：

## 程序清单 11-110

```
<integer>7</integer>
```

或：

```
<score xsd:type="int">7</score>
```

这个函数是非常灵活的，可进行处理数组等工作，又因为它是递归的，因此它能够处理数组之间的嵌套和其他的复杂结构，它也能将某些 VB对象转化成XML，例如作为数据库查询结果的记录集，该函数涉及到支持文档，篇幅很长。这里只展示该例程的一部分。

被传送的参数是要加入的节点，包括新节点的名字和将转换成节点的实际条目。如果节点名字为空，那么使用被传送的条目类型作为元素名字：

## 程序清单 11-111

```
Function addChild(oEl, sName, vItem)
    Dim bRet : bRet = False
    If sName = "" Then
        sName = TypeName(vItem)
    End If
```

下面是创建包含数据的节点。注意如果我们有一个变量数组，那么我们 Arrayof语法：

## 程序清单 11-112

```
If sName = "Variant()" Then
    sName = "ArrayOfvariant"
End If
Dim oTemp : Set oTemp = oDOM.createElement(sName)
```

依照数据类型，正确设置新的元素：

## 程序清单 11-113

```
Select Case TypeName(vItem)
    Case "Null"
        oTemp.setAttribute "xsd:null", "1"
    Case "Boolean"
        setTypedData oTemp, "boolean", vItem
    Case "Integer"
        setTypedData oTemp, "integer", vItem
    Case "Long"
        setTypedData oTemp, "integer", vItem
    Case "Single"
        setTypedData oTemp, "float", vItem
    Case "Double"
        setTypedData oTemp, "float", vItem
    Case "String"
        vItem = Replace(vItem, "&", "&amp;", 1, -1, 1)
        vItem = Replace(vItem, "<", "&lt;", 1, -1, 1)
        setTypedData oTemp, "string", vItem
```

将一个记录集作为一个参数的能力使之成为一个功能较强的函数。注意对于每一个领域都

递归调用 addChild ( ), 于是得到了条目的正确数据类型。它的另一项功能是, 通过它也能处理嵌套的记录集:

程序清单 11-114

```
Case "Recordset"
    Do While Not vItem.EOF
        Dim oRec : Set oRec = oDOM.createElement("Record")
        Dim oField, oNodeField
        for each oField in vItem.fields
            Set oNodeField = oDOM.createElement(oField.name)
            addChild oTemp, oField.name, oField.value
        next
        oTemp.appendChild(oRec)
        vItem.MoveNext
    Loop
```

最后, 不匹配的只有参数数组, 或不可识别的形式。如果是一个数组, 我们将把它看作成一个变量数组:

程序清单 11-115

```
Case Else
    If VarType(vItem) > vbArray Then
        Set oTemp = Nothing
        Set oTemp = oDOM.createElement("ArrayOfvariant")
        oTemp.setAttribute "xsd:type", "u:variant[" & UBound(vItem) & "]"
        for i = 0 to UBound(vItem) - 1
            addChild oTemp, "variant", vItem(i)
        next
    Else
        setTypedData oTemp, "string", "Unrecognised: " & TypeName(vItem) & _
            " [" & VarType(vItem) & "]"
    End If
End Select
oEl.appendChild(oTemp)
addChild = bRet
end function
```

有效载荷功能的最后部分是低层次的例程, 它对一个特定数据类型设置了一个节点:

程序清单 11-116

```
function setTypedData(ByRef oNode, sType, vData)
on error resume next
oNode.setAttribute "xsd:type", sType
oNode.dataType = sType
oNode.nodeTypeValue = vData
If Err Then
    oNode.setAttribute "xsd:type", "string"
    oNode.dataType = "string"
    oNode.nodeTypeValue = _
        "Error setting '" & oNode.nodeName & "' " & "(type " & sType & ") to " & _
        "'" & vData & "' " & "(type " & VarTypeText(vData) & "). " & _
        & "[" & err.number & "-" & err.description & "-" & err.source & "]"
```

```

        Err.Clear
    End If
end function

```

为了运用Microsoft XML DOM中的XML-DM工具，我在nodeTypedValue属性内放了一个值，同时也设置了xsd:type值作为XML模式第二部分的请求。我猜想，当Microsoft发布下一个XML DOM版本时，dt:dt语法将被xsd:type语法所取代。在这种语法里，函数的第一行可以不要。

#### (8) 有效载荷的提交

已经阐明了如何建立一个有效载荷，现在回到所关注的远程调用上来。为了不用来回翻看，现把代码写在下面：

#### 程序清单 11-117

```

function doRemoteCall(sU, sNS, sM, arP)
    On Error Resume Next
    '
    ' Create the request body from the method name and array of parameters
    '
    Dim oPL : Set oPL = CreateCallPayload(sNS, sM, arP)
    If Err.Number <> 0 Then
        Dim sTemp : sTemp = Err.Number & ": " & Err.Description
        Err.Clear
        Err.Raise 999, "SOAP", "Failed to create payload (" & sTemp & ")"
    Else
        ...
    End If
end function

```

如果成功调用了有效载荷，那现在将把有效载荷发送到远程服务器上。记住我们在 SOAP 的增强功能叙述里讨论过，必须在进行 M-POST之前先试一个 POST。下面是代码的开始部分：

#### 程序清单 11-118

```

        Err.Raise 999, "SOAP", "Failed to create payload (" & sTemp & ")"
    Else
        sM = sNS & "#" & sM
        Dim oHTTP : Set oHTTP = Server.CreateObject("Microsoft.XMLHTTP")
        iStatus = doPost("POST", oHTTP, sU, sNS, sM, oPL)
        If iStatus = 405 Then
            iStatus = doPost("M-POST", oHTTP, sU, sNS, sM, oPL)
        End If
    End If

```

如果你正在写自己的SOAP模块，那么记住SOAP标准列出了一些必须被处理的HTTP返回代码。例如，重定向代码302必须被实现。处理HTTP请求的一些组件将为你做这些工作，但是对它进行一下检查是有必要的。

在执行了一个POST后（或M-POST），可以检查返回值。如果收到200，那么至少知道HTTP处理了调用——尽管在SOAP的层次上仍可能有一些错误。除了200外的任何值都表示出错，最有可能的错误代码是501——没有执行或者510——没有扩展：



## 程序清单 11-119

```

'
' A 200 means that the HTTP layer is all OK
'
If iStatus <> 200 Then
    Err.Raise 999, "SOAP", "HTTP error: " & iStatus
Else
    Set doRemoteCall = oHTTP.responseXML
End If
End If
end function

```

实现POST或M-POST的函数如下：

## 程序清单 11-120

```

function doPost(sType, oHTTP, sURL, sNS, sMN, oB)
On Error Resume Next
Dim sMethod : sMethod = sMN
If sNS <> "" Then
    sMethod = sNS & "#" & sMethod
End If
oHTTP.open sType, sURL, false
oHTTP.setRequestHeader "Content-Type", "text/xml"
If sType = "M-POST" Then
    oHTTP.setRequestHeader "Man", "" & NS_SOAP & "", ns=01"
    oHTTP.setRequestHeader "01-SOAPMethodName", sMethod
Else
    oHTTP.setRequestHeader "SOAPMethodName", sMethod
End If
oHTTP.send(oB.xml)
doPost = oHTTP.status
end function

```

如果被实现的POST类型是强制性的——一个M-POST——那么要保证出现所有的报头。

需要去做的任务只剩下检查返回的数据，这是在 SOAP函数的顶层。CheckPayload ( ) 函数用于检查一个有效载荷，来确定在报头和有效载荷中的方法名是否一致，并确定命名空间是正确的等等。这个函数在客户端也用于检查从一个调用返回的有效载荷，而在服务器端用于检查激活一个调用的有效载荷。该函数返回在 SOAP里的重要节点，或者在 DOM参数里的一个错误结构：

## 程序清单 11-121

```

Err.Raise 999, "SOAP", "Failed to call remote server (" & sTemp & ")"
Else
Dim dom
Dim iOK : iOK = CheckPayload(oSOAP, dom, sNamespace, sMethod &
    "Response", 1)
If iOK = False Then
    Set oSOAP = dom
End If

```

注意，选择开始于根节点，这是因为在这里要处理两种情况：一是当实际的根节点是一个

<SOAP:Fault>元素，如果远程服务器返回一个<SOAP:Fault>和CheckPayload()，就是这种情形。二是如果在返回的有效载荷里有错误，在这处情况下，将从 CheckPayload() 得到一个错误的有效载荷。此时，根元素将是 <SOAP:Envelope>。如果发现了一个<SOAP:Fault>元素，那么将找出错误代码和另外的信息，并把它显示出来：

程序清单 11-122

```
Dim oTemp : Set oTemp = oSOAP.selectSingleNode("//SOAP:Fault")
If Not IsNull(oTemp) Then
    Set oSOAP = oTemp
    Dim iFC : iFC = oSOAP.selectSingleNode("faultcode").nodeTypedValue
    Dim sFS : sFS = oSOAP.selectSingleNode("faultstring").nodeTypedValue
    Dim sDet : sDet = oSOAP.selectSingleNode("detail").nodeTypedValue
    Err.Clear
    Err.Raise 999, sNamespace & "#" & sMethod, sFS & ": (" & sDet & ")"
Else
```

如果一切OK，那么将使用XMLToValue()把返回的XML转变成普通变量——addChild()函数的反转功能是很有效的：

程序清单 11-123

```
SOAP = True
oRet = XMLToValue(oSOAP)
End If
End If
```

### (9) 服务器模块

现在我们开始构建服务器端。第一个要做是创建一个XML DOM对象，并用提交已给我们发送的数据来装载它：

程序清单 11-124

```
<%
On Error Resume Next
Set oXML = Server.CreateObject("Microsoft.XMLDOM")
oXML.async = false
oXML.load(Request)
```

下一个得到的是调用者使用的方法。如果它是一个M-POST，那么我们需要得到Man报头，它包含了命名空间用以区别强制性报头，如果在命名空间两边有引用的话，那么要把它们删除：

程序清单 11-125

```
sMethod = Request.ServerVariables("REQUEST_METHOD")
Dim oBody
If sMethod = "M-POST" Then
    sMan = Request.ServerVariables("HTTP_Man")
    sMan = Trim(Replace(sMan, "", ""))
```

我们将检查命名空间是否与在SOAP标准里的定义相同，但在此时，什么也没有做：

## 程序清单 11-126

---

```

iPos = Instr(sMan, ";")
If Left(sMan, iPos - 1) <> NS_SOAP Then
End If

```

---

实际的命名空间标识符——与XML里的前缀相似——以“ns=”打头，一旦其被重新得到，就能创建报头名字，进而我们能够得到方法的名字：

## 程序清单 11-127

---

```

iPos = Instr(sMan, "ns=")
sManID = Mid(sMan, iPos + 1)

sMethodName = Request.ServerVariables("HTTP_" & sManID & "-SOAPMethodName")
Else
sMethodName = Request.ServerVariables("HTTP_SOAPMethodName")
End If

```

---

有了方法的名字，我们需要将命名空间分开；因为需要将它与有效载荷的命名空间值核对：

## 程序清单 11-128

---

```

' Get the namespace and method
'
sNamespace = ""
iPos = Instr(sMethodName, "#")
if iPos <> 0 then
    sNamespace = Left(sMethodName, iPos - 1)
    sMethodName = Mid(sMethodName, iPos + 1)
end if

```

---

下面这个片断并不是 SOAP 标准的一部分，但是用于检测目的的有用的代码片断。当方法是 ECHO ( ) 并且命名空间是 SOAP 时，它所做的就是返给用户一个实际传送的内容：

## 程序清单 11-129

---

```

' If the namespace is "SOAP" and the method is "ECHO" then just send it
' all straight back as XML to help testing
'
If sNamespace = "SOAP" and sMethodName = "ECHO" Then
    Response.ContentType = "text/xml"
    Response.Write oXML.xml
Else

```

---

在确定执行需要的函数之前，需要检查有效载荷的一致性：

## 程序清单 11-130

---

```

Dim vRet
Dim domServer
iOK = CheckPayload(oXML, domServer, sNamespace, sMethodName)
If iOK Then

```

---

如果有效载荷 OK，在传送它们到回调例程之前，oXML 将包含参数节点，并把它们转换成变量：

程序清单 11-131

```
param = XMLToValue(oXML)
SOAPCallback(param)
Set oXML = Nothing
```

如果调用成功执行，那么将创建一个响应有效载荷，并且它将被返回到调用者。另一方面，如果有错误，那么创建一个错误有效包：

程序清单 11-132

```
If Err Then
    Set domServer = CreateFaultPayload(400, err.number & ": " & _
        err.description)
    Err.Clear
Else
    Set domServer = CreateCallResponsePayload(sNamespace, sMethodName, vRet)
End If
End If

Response.ContentType = "text/xml"
Response.Write domServer.xml
Set domServer = Nothing
End If
%>
```

最后将被执行的是一系列的实际的例程。回忆一下我们已经创建的调用：

程序清单 11-133

```
if not Request.form("iAuthor") = "" then
    paramList(0) = CInt(Request.form("iAuthor"))
    paramList(1) = 1
    iRet = SOAP("http://server/SOAP/myServer.asp", _
        "myNamespace", _
        "getAuthor", _
        paramList, _
        oRet)
    ' Check the return status and then maybe do something with the returned value
end if
```

现在，需要创建 myServer.asp 模块，SOAP 服务器模块需要包含进来以使我们有权使用所有的有效载荷和数据处理函数。为了创建到你的函数的连接，需要有一个被 SOAP 结构调用的回调。在函数内放入了 case 语句，它将选择命名空间和方法。这里允许在不同的命名空间里相同的方法名字能被使用，注意尽管命名空间和方法值能作为参数被传送，但目前它们是全局变量。函数必须总是将它的值放于 vRet 变量，它也是一个全局变量：

## 程序清单 11-134

```

<!--#include virtual="/X-Port/SOAP/SOAP.asp" -->
<!--#include virtual="/X-Port/SOAP/SOAPServer.asp" -->
<%
'
' The main code calls a user-defined function and then wraps up the results.
' The results are then returned to the caller
'

Function SOAPCallback(param)
'
' First check the namespace
'
Select Case sNamespace
Case "myNamespace"
'
' Can now check the method name
'
Select Case sMethodName
Case "getAuthor"
vRet = getAuthor(param(0), param(1))
Case Else
Err.Raise 1000, "Namespace '" & sNamespace & "'
has no method called '" _
& sMethodName & "'"
End Select
Case Else
Err.Raise 1002, "There is no namespace called '" & sNamespace & "'"
End Select
End Function

```

最后是函数本身，例程简单地查询 Author 表（参见最后一章）以获得 iCount 值。这些作者的 ID 都比 iID 要大，查询的结果被放入到了一个数组，在数据库里的每一行都是一个多层数组：

## 程序清单 11-135

```

function getAuthor(iID, iCount)
Set dbConn = CreateObject("ADODB.Connection")
dbConn.Provider = "MSDataShape"
dbConn.Open "Data Provider=MSDASQL;DSN=SSMembership;user=sa;pwd=1998;"
Dim rs : Set rs = Server.CreateObject("ADODB.Recordset")
Dim sQuery
sQuery = "SELECT * FROM Author WHERE pk_Author > " & iID
rs.Open sQuery, dbConn, 0, 3
If Err = 0 Then
Dim a
a = rs.GetRows(iCount)
Dim aRow()
ReDim aRow(UBound(a, 1) + 1)
Dim aRows()
ReDim aRows(UBound(a, 2) + 1)
for i = 0 to UBound(a, 2)
for j = 0 to UBound(a, 1)
aRow(j) = a(j, i)

```

```
        next
        aRows(i) = aRow
    next
    getAuthor = aRows
End If
end function
%>
```

### 11.4.3 结论

XML-RPC以它的简洁性赢得了广大用户。它是非常简单的，XML-RPC对应用于行业的服务器到服务器的应用程序没有足够的定义来处理，而 SOAP作为一个主要升级产品使之成为一个理想的选择，如果它吸取 XML-RPC中的一些简洁的实现的话。

## 11.5 WebDAV

SOAP为RPC应用程序的创建提供了一个重要的基础。将来在 SOAP的基础上将可能设计出用于不同的目的的其他RPC标准。

但是，当我们对方法转换的数据格式进行标准化的时候，对于方法本身并没有标准化。在某些方面这又是不现实的，应用程序的范围是如此的巨大，以至于你不可能将使用的方法常规化。但是在Internet上有一种类型的应用程序是非常通行的——对于文档的编辑和文档上载。

XML-RPC的流行很大程度上得益于业内人士对于它的富于想象力的使用。他们设计程序的目的之一就是为了让人们在公告板上上载和编辑信息。另外是一种保持消息条目的能力。这些程序使用XML-RPC来允许一个远程使用者来保持他们的数据，而不需要服务器自己关心这些数据怎样被编辑。

使用这个技术存在的一个简单问题是用于保持信息的方法不是标准化的。换句话说，一个用于编辑在Userland站点上信息的客户应用程序不能用于维护其他站点的信息。同样，使用另外的协议的保持信息的应用程序，如 Microsoft FrontPage，不能用于编辑Userland数据。

WebDAV(Web Document Authoring and Versioning)提供了一系列特别的方法来编辑和控制通过HTTP传送的任何类型的数据。WebDAV定义了一些方法和用于加入、删除和浏览的格式。SOAP定义了方法调用如何被传送，WebDAV则定义了方法本身。

当然，公布一系列能够被 SOAP调用的方法是有可能的，同样你也能够为一些软件包公布一个API。提供一个覆盖Internet的系统的可能性正在增加。

尽管WebDAV对于处理远程的数据操作是一个不错的方案，但其执行客户和服务器应用程序并不像XML-RPC和SOAP那样轻松。但是它提供了一种可能性，例如 Microsoft已经在他的网络服务器中加入了一个WebDAV的兼容层，它提供同其FrontPage Server扩展相似的功能，但现在这却是一个标准的途径。这意味着任何理解 WebDAV的客户软件都能用来编辑存储在这样服务器上的文档。许多这样的工具将会出现，但目前 FrontPage和Microsoft的WebFolders（一个Internet Explorer 5的附加组件）能同任何WebDAV服务器通信。Microsoft也声明它们将在下一个发布的Exchange服务器里提供一个WebDAV界面。

在<http://www.ietf.cnri.reston.va.us/rfc/rfc2518.txt>里找到更多的关于WebDAV的信息。另外也

有公司从事有关 WebDAV 的工作。关于该技术使用情况的一些好的例子请浏览 <http://www.sharemation.com/>。

## 11.6 小结

在这章里我们主要看了看一些服务器到服务器的通讯方法，所有这些都包含对 XML 的使用。我们通过关注通信所需的条件来开始本章的，然后简要的介绍了一下提供给开发商的一些可选的实现方式。在这一部分里，我们的重点是 XML-RPC 和 SOAP，也介绍了一下 Coins、WDDX、XMOP 和 KOALA。

其后，我们更加深入地探讨了 XML-RPC——它为什么存在，它能够做什么，它的表达式是什么和在什么地方我们能够找到一些通信的解决方案。我们也关注了一些使用 XML-RPC 有详细描述的例子——一个贯穿始终的有关书籍目录的例子。一个 URL 的列表作为工具和附加读物被提供。

接着我们介绍了对 XML-RPC 进行强化的 SOAP，我们已经看到了这种技术——在写作时该技术尚是初期——是如何改变我们所关注的分布式计算方式的。它的优点是明显的——SOAP 是一种简单的协议，它能使我们利用现存的 HTTP 基础设施实现更为强大的功能，并克服了 XML-RPC 内在的冗长。然后，我们通过一个有教育性的例子描述了这个技术是如何工作的。

最后，我们看了 WEBDAV 是如何标准化更多的希望应用在整个 WEB 上的普通方法调用。WEBDAV 是一个呼之欲出的技术，故而提供了一些链接以使读者获得更多最新的信息资源。