

第18章 实例研究4——SOAP

提起这件事我就痛苦。很久以前（实际上，大约18个月之前），我写了一本名为《Professional DCOM Application Development》的书（ISBN 1-861001-31-2）。虽然这本书是以DCOM命名的，但它不仅仅是关于DCOM的。事实上，它从非常广泛的角度讲述了如何利用 Microsoft在这几年推出的各种技术构造企业级应用程序，这些技术包括 MTS、MSMQ、ADSI及Microsoft Clusters等。当然还包括DCOM。

问题是书的封面上的“DCOM”格外醒目（请注意，封面上还有我的大照片，坦白地说——这使情况变得更加糟糕）。如果我遇到的某个人恰巧知道我写了这本书，通常随之而来的就是一场争论：DCOM的可行性到底如何。偶尔，有人会客气地询问我目前为他们做的项目为什么不使用DCOM。

当然，问题的根源在于虽然从理论角度讲 DCOM是个很巧妙的概念（正如“嘿，我们为什么不建立一个系统，然后在不做任何修改的情况下在各种环境中实施”），但是它不像最初看起来那样是个通用的万能药。首先，DCOM是重负载的有线协议（wire protocol），尽管在非Windows平台上也存在着相应的版本，可是它必然滞后于 Windows版本。最关键的，如果不争得网络管理员的同意，DCOM无法通过防火墙。

因此，这就是为什么当我听说 SOAP之后感到非常兴奋的原因。最重要的，SOAP是一个纯粹的有线协议，它不要求使用任何 ORB技术（Microsoft或其他厂商和组织）。它能够很好地处理防火墙问题。第11章详细介绍了SOAP，在本章我们将看看它的实际应用。我打算用以前书中的一个例子为例，使用SOAP重新实现它，并加以说明。在本章的实例研究中，我们将使用本地的COM和SOAP实现面向全球的订单输入应用程序。我们将使用包含 ATL 3.0的Visual C++ 6.0编写COM对象；如果你对此不熟悉，我推荐你阅读 Richard Grimes的《Beginning ATL 3 COM Programming》（ISBN 1-861001-20-7）和《Professional ATL COM Programming》（ISBN 1-861001-40-1）。我们将建立两个可选的服务器——一个在Windows NT上（使用Visual Studio C++ 6.0和MFC），另一个在Linux上（使用标准C）——以说明它是真正跨平台的。如果可能的话，我们将使用DOM解析SOAP XML内容，否则我们将自行解析XML内容。

18.1 追根溯源

如果你看过《Professional DCOM Application Development》（如果你还没看过，为什么不买一本呢？），可能还记得一个很奇怪的例子，居住在 Ulaanbaatar的唱片收藏者从 Tierra del Fuego 的一台主机上订购六十年代迷幻乐的慢转密纹唱片，如图 18-1所示。

这个例子实际上有三个版本（也许我记错了，应该是四个版本），它们分别采用以下三种方式：基本的逐字段远程验证订单输入项，根据值排列输入项进行本地验证，以及使用 MSMQ通过可靠的消息队列发送订单对象。现在，我们准备构造使用 SOAP的版本。

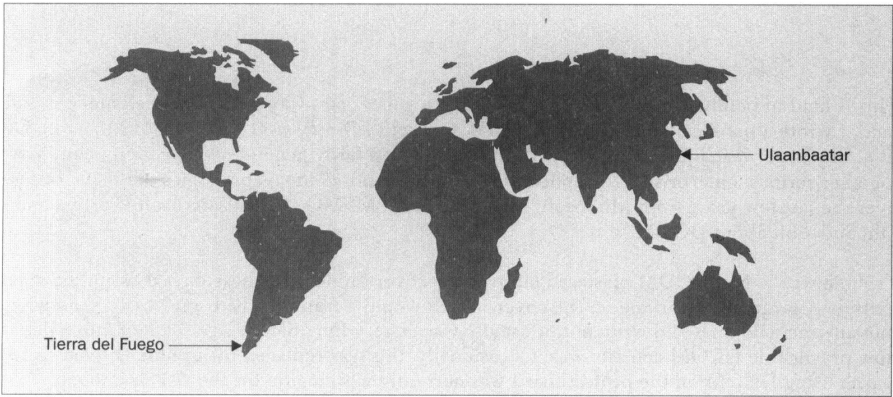


图 18-1

我们的订单输入和验证 COM对象Order只有一个接口 IOrder，该接口提供了五个属性和一个方法。每个属性对应于订单中的一个字段，它们的验证需求略有不同，参见表 18-1。

表 18-1

字 段	验 证
CustomerID	必需的，只能包含字母
Artist	必需的，只能包含字母
Title	必需的，只能包含字母和数字
Label	可以为空；如果存在，只能包含字母
Price	必须是有效的美元价格

接口提供的唯一方法 Submit()用于将订单的详细信息写入结构化文件（ c:\order.dat ）。项目中相应的代码称为 SimpleOrder，我在此不打算详细介绍它。如果你感兴趣，可参见《 Professional DCOM Application Development》一书。

以下代码实现了接口的五个属性和一个方法：

程序清单 18-1

```
STDMETHODIMP COrder::put_CustomerID(BSTR newVal)
{
    HRESULT hResult = MandatoryAlphabetic(newVal);
    if (FAILED(hResult))
        return hResult;

    m_bstrCustomerID = newVal;
    return S_OK;
}

STDMETHODIMP COrder::put_Artist(BSTR newVal)
{
    HRESULT hResult = MandatoryAlphabetic(newVal);
    if (FAILED(hResult))
        return hResult;
}
```

```

    m_bstrArtist = newVal;
    return S_OK;
}

STDMETHODIMP COrder::put_Title(BSTR newVal)
{
    HRESULT hResult = MandatoryAlphanumeric(newVal);
    if (FAILED(hResult))
        return hResult;

    m_bstrTitle = newVal;
    return S_OK;
}

STDMETHODIMP COrder::put_Label(BSTR newVal)
{
    HRESULT hResult = OptionalAlphabetic(newVal);
    if (FAILED(hResult))
        return hResult;

    m_bstrLabel = newVal;
    return S_OK;
}

STDMETHODIMP COrder::put_Price(BSTR newVal)
{
    HRESULT hResult = MandatoryPrice(newVal);
    if (FAILED(hResult))
        return hResult;

    m_bstrPrice = newVal;
    return S_OK;
}

STDMETHODIMP COrder::Submit()
{
    CComPtr<IStorage> pStorage;
    HRESULT hResult = StgCreateDocfile(L"C:\\\\Order.dat",
        STGM_SIMPLE | STGM_CREATE | STGM_READWRITE |
        STGM_SHARE_EXCLUSIVE, 0, &pStorage);
    if (FAILED(hResult))
        return hResult;

    Write(pStorage, L"CustomerID", m_bstrCustomerID);
    Write(pStorage, L"Artist", m_bstrArtist);
    Write(pStorage, L"Title", m_bstrTitle);
    Write(pStorage, L"Label", m_bstrLabel);
    Write(pStorage, L"Price", m_bstrPrice);

    return S_OK;
}

```

正如你所看到的，每个属性定义了订单中的一个有效字段，它们被保存在以下本地成员变量中：

程序清单 18-2

```

private:
    CComBSTR m_bstrCustomerID;
    CComBSTR m_bstrArtist;

```

```
CComBSTR m_bstrTitle;  
CComBSTR m_bstrLabel;  
CComBSTR m_bstrPrice;
```

这是在 Order.h 中定义的。

然后，Submit() 方法将这些属性写入订单文件。除此之外还有一些辅助方法，它们用于处理各种有效性验证选项（参见表 18-2）。

表 18-2

方 法	功 能
MandatoryAlphabetic()	检查值不为空，而且只包含字母
OptionalAlphabetic()	检查值可以为空，或者只包含字母
MandatoryAlphanumeric()	检查值不为空，而且只包含字母和数字
OptionalAlphanumeric()	检查值可以为空，或者只包含字母和数字
MandatoryPrice()	检查值不为空，而且是格式正确的价格

以下是这些方法的代码：

程序清单 18-3

```
HRESULT COrder::MandatoryAlphabetic(BSTR bstrValue)  
{  
    int length = SysStringLen(bstrValue);  
    if (length == 0)  
        return E_INVALIDARG;  
  
    return OptionalAlphabetic(bstrValue);  
}  
  
HRESULT COrder::OptionalAlphabetic(BSTR bstrValue)  
{  
    int length = SysStringLen(bstrValue);  
  
    for (int iChar = 0; iChar < length; iChar++)  
    {  
        if (!iswspace(bstrValue[iChar]) &&  
            !iswalpha(bstrValue[iChar]))  
            return E_INVALIDARG;  
    }  
  
    return S_OK;  
}  
  
HRESULT COrder::MandatoryAlphanumeric(BSTR bstrValue)  
{  
    int length = SysStringLen(bstrValue);  
    if (length == 0)  
        return E_INVALIDARG;  
  
    return OptionalAlphanumeric(bstrValue);  
}  
  
HRESULT COrder::OptionalAlphanumeric(BSTR bstrValue)  
{
```

```

int length = SysStringLen(bstrValue);

for (int iChar = 0; iChar < length; iChar++)
{
    if (!isspace(bstrValue[iChar]) &&
        !iswalnum(bstrValue[iChar]))
        return E_INVALIDARG;
}

return S_OK;
}

HRESULT COrder::MandatoryPrice(BSTR bstrValue)
{
    // Get the locale ID for the US with default sorting
    LCID lcid = MAKELCID(MAKELANGID(LANG_ENGLISH, SUBLANG_ENGLISH_US),
        SORT_DEFAULT);

    // Define a variable to hold the returned currency
    CURRENCY cy = {0};

    // Convert the string to a currency value
    HRESULT hr = VarCyFromStr(bstrValue, lcid, 0, &cy);

    // If the function failed, bstrValue is not a valid currency
    if (FAILED(hr))
        return E_INVALIDARG;

    // Finally, check that the currency value is positive
    if (cy.int64 < 0)
        return E_INVALIDARG;

    return S_OK;
}

```

最后一个辅助方法 Write() 的功能是将一个字段写入文件：

程序清单 18-4

```

void COrder::Write(IStorage* pStorage, LPOLESTR lpszField, CComBSTR& bstr)
{
    CComPtr<IStream> pStream;
    pStorage->CreateStream(lpszField, 'STGM_READWRITE |
        STGM_SHARE_EXCLUSIVE, 0, 0, &pStream);

    bstr.WriteToStream(pStream);
}

```

在原来最初的例子中，我有一个简单的 Visual Basic 客户，它允许用户创建订单对象的远程实例，以有效的方式获取字段值，然后提交它。在本例中，我计划将客户代码建立在以前的代码的基础上，主要有以下两点原因：(1) 我希望说明 SOAP 不仅仅可以用于基于 Web 的应用程序，(2) 我比较懒惰，不愿让代码闲置。

那么，如果我们准备使用 SOAP，我们还需要什么？

18.2 SOAP Opera

我们需要依靠某种基础结构来管理这些，并且要为其命名（这是我真正喜欢的）。正如你所想象的，SOAP迅速成为缩写词的滋生地，由此产生了许多关双语。例如，我最初考虑的名称SUDS（代表Simple Usurper of DCOM using SOAP）很快就被排除掉了，因为它容易与SOAP Uniform Description Semantics（SOAP统一描述语义）相混淆。因此，当我将SOAP OPERA公诸于世时，心里不免战战兢兢。不管怎样，OPERA可能代表“Object Protocol Enabling Remote Access（支持远程访问的对象协议）”，但是，这仅仅是个命名而已，不是吗？

如果我们使用纯DCOM，图18-2显示了系统的简化结构图：

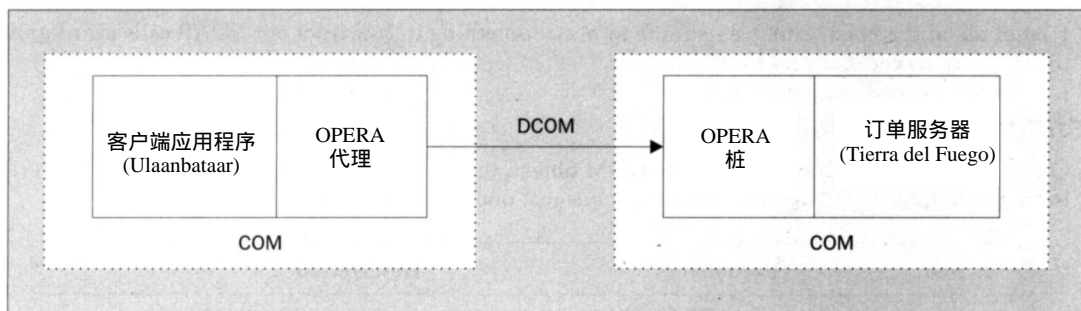


图 18-2

如果我们使用SOAP，系统结构图参见图18-3。

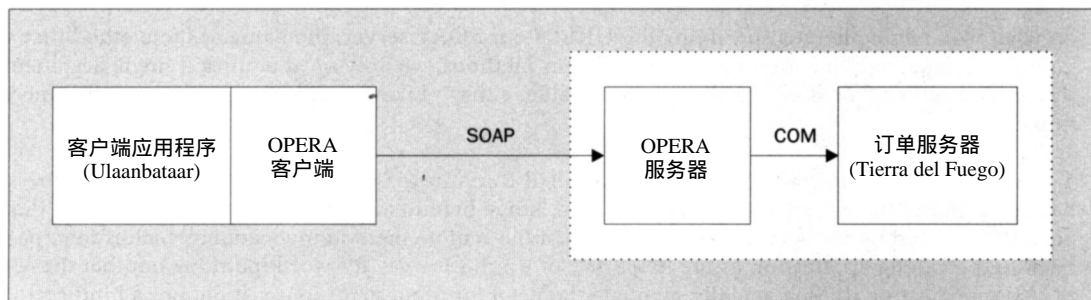


图 18-3

为了使用OPERA客户端，我们必须略微修改一下客户应用程序（当然，我们也可以为它提供一个实现了IOrder接口的COM对象，这样就能在不改动任何代码的情况下使用它）。OPERA客户是进程内COM对象，它实现了新的接口IOpera。我们将详细说明一下这个接口。它与位于Tierra del Fuego的OPERA服务器交换SOAP调用。然后，OPERA服务器实例化COM订单对象，并通过本地COM调用它们的方法。本例中的订单服务器与DCOM例子中的服务器完全相同，但是在此它接收的都是本地访问。

比较以上两个结构图可以看出，OPERA客户取代了COM代理，有限协议由DCOM变为SOAP，

OPERA服务器取代了COM stub（严格地说，OPERA服务器仍然使用COM代理/stub访问订单服务器，但是这主要是出于代码重用的考虑。假如我们重新实现订单服务器，情况就会有所改变）。

最后有一点要说明。对于本例，经验丰富的COM程序员可能会坚持认为应该使用“定制排列（Custom Marshaling）”。当然，我们没有理由不在订单对象中实现IMarshal，以便每个IOrder方法调用都会作为SOAP调用发送给服务器，它对应用程序来说是完全透明的。然而，定制排列不在本例讨论范围之内，如果读者对此感兴趣，可以参考《Professional DCOM Application Development》一书的第3章，其中深入介绍了有关内容。

下面我们来详细介绍该系统。

18.2.1 OPERA客户端

首先，在客户端我们需要构造SOAP调用，并将它发送给服务器。

1. 定义接口

我们的客户端是非常简单的COM对象Opera，要将它伪装为完全通用的。它只有一个接口IOpera，其中包含一个方法（参见表18-3）。

表 18-3

方 法	参 数	功 能
Exchange()	[in] BSTR URL [in] BSTR Method [in] VARIANT Argument	与服务器交换SOAP调用

因此，我们需要指定SOAP服务器的URL，要调用的方法名称，以及一个参数。对于所有的方法，只需要至多一个参数，所以这就足够了。另外，我们不需要返回值（正如我所说，它不是特别通用的实现）。

你可能会疑问，既然方法没有返回值，为什么称为Exchange()。实际上，确实存在着请求/应答交换；然而，服务器唯一返回的是远程方法调用的HRESULT，然后它会将作为Exchange()的HRESULT返回给产生调用的应用程序。需要特别指出的是，1.0版的SOAP规范并不强制要求请求/应答方式。它定义了与HTTP的绑定，然而真正的协议是独立于传输的。你绝对没有理由不使用SOAP进行单向方法调用——它有可能为MSMQ提供实现的基础。

2. 建立客户端

让我们详细分析一下内部的代码。我们将使用ATL，并将它实现为进程内COM对象。我们使用ATL AppWizard创建项目，使用Object Wizard添加一个简单对象，并且确保选中了标有“Support ISupportErrorInfo”的复选框。然后，使用ClassView添加Exchange()方法。

下面是Opera.h头文件：

程序清单 18-5

```
// Opera.h : Declaration of the COpera  
  
#ifndef __OPERA_H_  
#define __OPERA_H_
```



```

#include "resource.h"           // main symbols
#import "msxml.dll"

////////////////////////////////////
// COpera
class ATL_NO_VTABLE COpera :
    public CComObjectRootEx<CComSingleThreadModel>,
    public CComCoClass<COpera, &CLSID_Opera>,
    public ISupportErrorInfo,
    public IDispatchImpl<IOpera, &IID_IOpera, &LIBID_OPERACLIENLib>
{
protected:
    int m_sock;

    HRESULT ConnectToHost(LPSTR lpszURL);
    HRESULT DisconnectFromHost();
    HRESULT ProcessContent(int nChar, LPSTR lpszContent);

public:
    COpera() : m_sock(-1)
    {
    }

    ~COpera()
    {
        if (m_sock != -1)
            DisconnectFromHost();
    }

DECLARE_REGISTRY_RESOURCEID(IDR_OPERA)

DECLARE_PROTECT_FINAL_CONSTRUCT()

BEGIN_COM_MAP(COpera)
    COM_INTERFACE_ENTRY(IOpera)
    COM_INTERFACE_ENTRY(IDispatch)
    COM_INTERFACE_ENTRY(ISupportErrorInfo)
END_COM_MAP()

// ISupportsErrorInfo
STDMETHOD(InterfaceSupportsErrorInfo)(REFIID riid);

// IOpera
public:
    STDMETHOD(Exchange) (/*[in]*/ BSTR URL, /*[in]*/ BSTR Method,
                        /*[in]*/ VARIANT Argument);
};

#endif // __OPERA_H_

```

向导和ClassView会自动生成一些代码，上面突出显示的代码是你要添加的。正如你所看到的，我已经增加了一两个辅助方法，以及一个 int 型的 m_sock 用于存放 TCP/IP 套接字标识符，并且引用了 MSXML 库，以便使用 Microsoft 实现的 DOM。另外，我们还要在 OperaClient.cpp 文件的 DllMain 中添加几行代码：

程序清单 18-6

```
extern "C"
BOOL WINAPI DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID /*lpReserved*/)
{
    if (dwReason == DLL_PROCESS_ATTACH)
    {
        _Module.Init(ObjectMap, hInstance, &LIBID_OPERACLIENTLib);
        DisableThreadLibraryCalls(hInstance);
        WSADATA wsaData;
        if (WSAStartup(0x0101, &wsaData))
            return FALSE;
    }
    else if (dwReason == DLL_PROCESS_DETACH)
    {
        _Module.Term();
        WSACleanup();
    }
    return TRUE;    // ok
}
```

以上突出显示的代码是为了保证 TCP/IP 的正常工作。当客户端应用程序开始使用 DLL 时，我们要发布命令，启动 WinSock（指定版本 1.1），当它断开时清除 WinSock。如果愿意的话，你也可以用 WinInet 替代 WinSock。

如前所述，Exchange() 方法执行与服务器的 SOAP 交换。典型的交换（比如：设置 CustomerID 属性）可能包含以下从客户到服务器的信息：

程序清单 18-7

```
POST /StockQuote HTTP/1.1
Host: www.jpassoc.co.uk
Content-Type: text/xml
Content-Length: 300
SOAPMethodName: www.jpassoc.co.uk/opera#put_CustomerID

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
  <SOAP:Body>
    <m:put_CustomerID xmlns:m="www.jpassoc.co.uk/opera">
      <argument>JMP</argument>
    </m:put_CustomerID>
  </SOAP:Body>
</SOAP:Envelope>
```

以及从服务器到客户的信息：

程序清单 18-8

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 252

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
  <SOAP:Body>
    <m:put_CustomerIDResponse xmlns:m="www.jpassoc.co.uk/opera">
      <return>0</return>
    </m:put_CustomerIDResponse>
  </SOAP:Body>
</SOAP:Envelope>
```

3. Exchange()方法

下面我们来介绍Exchange()方法：

程序清单 18-9

```
STDMETHODIMP COpera::Exchange(BSTR URL, BSTR Method, VARIANT Argument)
{
    size_t length = SysStringLen(URL);
    LPSTR lpszURL = new char[length + 1];
    wcstombs(lpszURL, URL, length);
    lpszURL[length] = 0;

    if (m_sock == -1)
    {
        HRESULT hResult = ConnectToHost(lpszURL);
        if (hResult != S_OK)
        {
            delete[] lpszURL;
            return hResult;
        }
    }
}
```

如果尚未与服务器建立连接，我们将通过一个辅助方法建立连接。稍后将介绍该方法。现在，让我们看看应用程序引用了哪个方法调用，然后从 VARIANT 中提取参数，构造 MessageType 头：

程序清单 18-10

```
length = SysStringLen(Method);
LPSTR lpszMethod = new char[length + 1];
wcstombs(lpszMethod, Method, length);
lpszMethod[length] = 0;

char szCall[2000];

sprintf(szCall,
        "SOAPMethodName: www.jpassoc.co.uk/opera#%s\r\n\r\n", lpszMethod);
strcat(szCall,
        "<SOAP:Envelope xmlns:SOAP=\"urn:schemas-xmlsoap-org:soap.v1\">\r\n");
strcat(szCall, "<SOAP:Body>\r\n");

length = strlen(szCall);
sprintf(&szCall[length],
        "      <m:%s xmlns:m=\"www.jpassoc.co.uk/opera\">\r\n", lpszMethod);

char szArgument[100];

switch (Argument.vt)
{
    case VT_EMPTY:
        break;

    case VT_BSTR:
    {
        length = SysStringLen(Argument.bstrVal);
        wcstombs(szArgument, Argument.bstrVal, length);
        szArgument[length] = 0;
    }
}
```

```

length = strlen(szCall);
sprintf(&szCall[length],
        "                <argument>%s</argument>\r\n", szArgument);

break;
}

default:
delete[] lpszURL;
delete[] lpszMethod;

return AtlReportError(CLSID_Opera, "Variant type not supported",
                      IID_IOpera, E_NOTIMPL);
}

```

你并不指望有一个完整的实现，不是吗？空白和字符串就足以说明问题。我们将添加其他头（包括最重要的Content-Type头），完成上传操作：

程序清单 18-11

```

length = strlen(szCall);
sprintf(&szCall[length],
        "                </m:%s>\r\n                </SOAP:Body>\r\n</SOAP:Envelope>",
        lpszMethod);

length = strlen(szCall);

char szRequest[2000];
sprintf(szRequest, "POST /OperaServer HTTP/1.1\r\nHost: %s\r\nContent-Type: "
        "text/xml\r\nContent-Length: %d\r\n%s", lpszURL, length, szCall);

delete[] lpszURL;
delete[] lpszMethod;

```

下面我们来交换请求和应答。值得注意的是，如果这是一个实际的应用程序，必须设置计时器，以防与服务器之间的连接中途中断。

程序清单 18-12

```

send(m_sock, szRequest, strlen(szRequest), 0);

char szResponse[20000];
int nChar = recv(m_sock, szResponse, sizeof(szResponse), 0);

```

检查套接字有没有被服务器关闭：

程序清单 18-13

```

if (nChar <= 0)
{
    DisconnectFromHost();
    return AtlReportError(CLSID_Opera, "Connection to server lost",
                          IID_IOpera, E_FAIL);
}

```

现在，分析一下应答。首先去除第一行，我们对此不是特别感兴趣：

程序清单 18-14

```
szResponse[nChar] = 0;
LPSTR lpszReturn = strstr(szResponse, "\r\n");

if (!lpszReturn)
    return AtlReportError(CLSID_Opera, "Incomplete response from server",
        IID_IOpera, E_FAIL);

*lpszReturn = 0;

if (strcmp(szResponse, "HTTP/1.1 200 OK"))
    return AtlReportError(CLSID_Opera, "Bad response from server",
        IID_IOpera, E_FAIL);
```

下面是两个标记。仅当我们获得了一个有效的 SOAP调用应答必需具备的所有头，才设置第一个标记。如果服务器返回“Connection: close”头，则设置第二个标记。如果这个标记没有被置位，将保持连接处于打开状态。这是因为我们在 Patagonia生成订单，而且不希望失去它。那里保存了状态信息。实际上，我不得不承认这种方式不太恰当，我也曾试图修改应用程序，使得它能够在客户端生成订单，在交易的最后才发送给服务器。这样，就能够在断线的情况下完成所有操作。然而，我的目标是尽可能仿效原来的 DCOM应用程序，除了要剔除 DCOM有线协议所带来的额外负担。我将这项任务作为练习留给你，希望你能够使应用程序在完全无连接的情况下执行操作。如果读者强烈要求，我会在我的 Web 站点 www.jpassoc.co.uk 上发布无连接版本。

程序清单 18-15

```
bool bSoap = false;
bool bClose = true;

HRESULT hResult = S_OK;

while (1)
{
    LPSTR lpszSegment = lpszReturn + 2;
    lpszReturn = strstr(lpszSegment, "\r\n");

    if (!lpszReturn)
    {
        hResult = AtlReportError(CLSID_Opera, "Non-SOAP response from server",
            IID_IOpera, E_FAIL);
        break;
    }

    *lpszReturn = 0;

    if (!strncmp(lpszSegment, "Content-Type: ", 14))
    {
        if (!strcmp(&lpszSegment[14], "text/xml"))
            bSoap = true;
        else
        {
            hResult = AtlReportError(CLSID_Opera, "Non-SOAP response from server",
                IID_IOpera, E_FAIL);
        }
    }
}
```

```

        break;
    }
}
else if (!strcmp(lpszSegment, "Connection: ", 12))
{
    if (!strcmp(&lpszSegment[12], "Keep-Alive"))
        bClose = false;
}
else if (!strcmp(lpszSegment, "Content-Length: ", 16))
{
    if (!bSoap)
        hResult = AtlReportError(CLSID_Opera, "Non-SOAP response from server",
                                IID_IOpera, E_FAIL);
    else
    {

```

现在，我们知道已经获得了所需的内容，得到了真正的 SOAP应答，因此下面将调用一个辅助方法处理它：

程序清单 18-16

```

    int nChar = atoi(&lpszSegment[16]);
    hResult = ProcessContent(nChar, lpszReturn + 2);
}

break;
}
}

```

如果已经完成订单，将关闭套接字；否则，保持其打开状态，留待下一个调用使用（如果释放对象，套接字也将关闭）。

程序清单 18-17

```

if (bClose)
    DisconnectFromHost();

return hResult;
}

```

4. 辅助方法

下面我们将介绍几个辅助方法。首先是完成连接功能的方法：

程序清单 18-18

```

HRESULT COpera::ConnectToHost(LPSTR lpszURL)
{
    m_sock = socket(AF_INET, SOCK_STREAM, 0);

    int on = 1;
    setsockopt(m_sock, SOL_SOCKET, SO_REUSEADDR, reinterpret_cast<char*> (&on),
               sizeof(on));

    struct sockaddr_in server;
    memset(&server, 0, sizeof(server));

```

```

server.sin_family = AF_INET;
server.sin_port = htons(80);

struct hostent *pHost = gethostbyname(lpszURL);
if (pHost)
    memcpy(&server.sin_addr, pHost->h_addr, pHost->h_length);
else
{
    server.sin_addr.s_addr = inet_addr(lpszURL);

    if (server.sin_addr.s_addr == 0)
    {
        DisconnectFromHost();
        return AtlReportError(CLSID_Opera, "Failed to translate host address",
                               IID_IOpera, E_FAIL);
    }
}

if (connect(m_sock, reinterpret_cast<struct sockaddr *> (&server),
            sizeof(server)) == -1)
{
    char szError[256];
    sprintf(szError, "Failed to connect to host, error = %d\n",
            WSAGetLastError());

    DisconnectFromHost();

    return AtlReportError(CLSID_Opera, szError, IID_IOpera, E_FAIL);
}

return S_OK;
}

```

熟悉TCP/IP编程的人对于以上代码不会感到陌生。其中大部分代码用于将 URL和端口号转换为适于连接的服务器结构。注意，我们连接到服务器的 80端口——SOAP通过这种方式欺骗防火墙，使之认为我们正在进行 Web操作。当SOAP被广泛接受之后，它很可能有自己的特定的端口，以便网络管理人员能够区分出它，但是目前，我们仍然使用端口 80。

断线方法更加简单：

程序清单 18-19

```

HRESULT COpera::DisconnectFromHost()
{
    closesocket(m_sock);
    m_sock = -1;

    return S_OK;
}

```

在结束对 OPERA 客户的讨论之前，我们还要分析另一个辅助方法 ProcessContent()。它负责处理来自服务器的应答。

毫无疑问，我没有必要亲自解析应答，因为 Microsoft 的 XML DOM 提供了许多相当出色的

COM对象。在开始处理之前，首先看一下应答可能具有的形式。以下是来自 OPERA服务器的应答的例子：

程序清单 18-20

```
<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
  <SOAP:Body>
    <m:put_CustomerIDResponse xmlns:m="www.jpassoc.co.uk/opera">
      <return>0</return>
    </m:put_CustomerIDResponse>
  </SOAP:Body>
</SOAP:Envelope>
```

现在，我们开始解析它：

程序清单 18-21

```
HRESULT COpera::ProcessContent(int nChar, LPSTR lpszContent)
{
    CComPtr<MSXML::IXMLDOMDocument> pXML;
    pXML.CoCreateInstance(__uuidof(MSXML::DOMDocument));

    CComBSTR bstrXML = lpszContent;
    VARIANT_BOOL success;
    success = pXML->loadXML(bstrXML.m_str);

    if (success != VARIANT_TRUE)
        return AtlReportError(CLSID_Opera, "Failed to parse XML", IID_IOpera,
                               E_FAIL);

    CComPtr<MSXML::IXMLDOMElement> pElem;
    XML->get_documentElement(&pElem);
```

顺便说一下，如果你对性能非常关注，要考虑避免使用 DOM，特别是考虑到我们在此使用的XML结构不是相当复杂。如果你对采用非 DOM方式感兴趣，可以参考后面介绍的Linux实现。

以下代码用于提取方法名称（虽然在本例中我们并不需要使用它）：

```
CComBSTR bstrMethod;
pElem->get_tagName(&bstrMethod);
```

现在我们提取第一个子节点：

```
CComPtr<IXMLDOMNode> pChild;
pElem->get_firstChild(&pChild);
```

我们对于标记不感兴趣（因为要假设它是 return），因此直接获取其中的文本：

```
CComBSTR bstrReturn;
pChild->get_text(&bstrReturn);
```

要返回给发出调用的应用程序的结果就是从 XML应答中获得的内容：

```
HRESULT hResult = _wtol(bstrReturn);
```

```
return hResult;
```

```
}
```

以上就是我们的客户COM对象。在讨论产生调用的应用程序之前，先看一下服务器的操作。

18.2.2 OPERA服务器

我们的服务器必须接收和解释通过 TCP/IP 发送的 HTTP 命令。简而言之，它应该看上去像个 Web 服务器。因此，我们要编写一个 Web 服务器，是吗？我认为每个人一生之中至少应该有一次这样的经历，因为它有助于理解许多 Web 底层的技术。准备好了吗？我们开始吧！

1. 创建项目

实际上，我们不需要编写一个 Web 服务器。我们要编写的是一个类似于 Web 服务器的东西。我们将使用老式的 MFC，因为我恰巧喜欢 MFC 的套接字类，而且它需要某种形式的用户界面；另外，你也可以使用 Visual Basic 的 WinSock 控件，或者 ATL 的纯 WinSock API。然而，在开始之前，我们要确认已经关闭了 IIS 或个人 Web 服务器，以及其他能够优先访问端口 80 的程序。否则，我们自己的服务器无法监听端口 80。如前所述，将来 SOAP 可能有自己的端口，这样 Web 服务器和 SOAP 就能够一起运行。如果你不愿意关闭或重新设置 Web 服务器，可以为 SOAP 选择其他端口。你要使用 MFC AppWizard 创建基于对话框的项目，并确认选择了套接字复选框。

我们需要两个与套接字相关的类：“主”类和“客户”类。“主”类 COperaMainSocket 派生出的对象用于监听 80 端口，接收进入服务器的调用。当它接收到调用时，它创建 COperaClientSocket 对象。实际上，完全可以将与套接字相关的所有内容都放在一个类中，但是我个人认为这种方式能够提高程序的可读性。这两个类都继承了标准的 MFC 类 CAsyncSocket。

2. 主套接字类

“主”套接口类重载了一个标准方法：OnAccept()。以下代码是类定义。你需要将突出显示的行是要添加到 MFC AppWizard 生成的代码中：

程序清单 18-22

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// COperaMainSocket command target

class COperaMainSocket : public CAsyncSocket
{
// Attributes
public:

// Operations
public:
    COperaMainSocket();
    virtual ~COperaMainSocket();

// Overrides
public:
    // ClassWizard generated virtual function overrides
    //{AFX_VIRTUAL(COperaMainSocket)
    //}AFX_VIRTUAL

    // Generated message map functions
    //{AFX_MSG(COperaMainSocket)
    // NOTE - the ClassWizard will add and remove member functions here.
    //}AFX_MSG

// Implementation

```

```
protected:
    CObArray m_sockArray;

public:
    void Check();
};
```

添加的新成员变量是一个对象数组，我们用它保存指向客户套接字对象的指针。以下是该类的构造方法和析构方法：

程序清单 18-23

```
COperaMainSocket::COperaMainSocket()
{
}

COperaMainSocket::~COperaMainSocket()
{
    int nSock = m_sockArray.GetSize();

    for (int sock = 0; sock < nSock; sock++)
    {
        COperaClientSocket *pSocket =
            static_cast<COperaClientSocket *> (m_sockArray.GetAt(sock));
        delete pSocket;
    }

    m_sockArray.RemoveAll();
}
```

我们所要做的是将信息写入客户套接字数组。真正的操作是在重载的 OnAccept()方法中执行的，可以通过MFC Class Wizard添加该方法：

程序清单 18-24

```
void COperaMainSocket::OnAccept(int nErrorCode)
{
    CWinApp *pWinApp = AfxGetApp();
    COperaServerDlg *pDialog = static_cast<COperaServerDlg *> (pWinApp->m_pMainWnd);

    int length = sizeof(SOCKADDR_IN);

    COperaClientSocket *pSocket = new COperaClientSocket();
    SOCKADDR_IN sockAddr;
    Accept(*pSocket, reinterpret_cast<SOCKADDR *> (&sockAddr), &length);

    m_sockArray.Add(pSocket);

    CString csAddress;
    unsigned int port;
    pSocket->GetPeerName(csAddress, port);

    CString csStatus;
    csStatus.Format("Accepting connection from %s", csAddress);
    pDialog->Status(csStatus);

    CAsyncSocket::OnAccept(nErrorCode);
}
```

以上代码创建处理通信过程的客户套接字，接收进入服务器的 TCP/IP调用，并将它添加到数组中。后半部分代码（从 csStatus 声明向前开始）用于向主对话框 COperaServerDlg 输入某些状态信息，我们稍后将介绍该对话框。

主套接字类中还有一个方法：

程序清单 18-25

```
void COperaMainSocket::Check()
{
    int nSock = m_sockArray.GetSize();

    for (int sock = 0; sock < nSock; sock++)
    {
        COperaClientSocket *pSocket =
            static_cast<COperaClientSocket *> (m_sockArray.GetAt (sock));

        if (!pSocket->m_bActive)
        {
            delete pSocket;
            m_sockArray.RemoveAt (sock);

            sock--;
            nSock--;
        }
    }
}
```

该方法检查客户套接字是否关闭。如果某个套接字关闭，它将从数组中被删除。由于这是客户套接字，我们需要知道它是否关闭，因此必须执行该操作。实际上，该方法是在主对话框的一个计时器中调用的。同样，我们将在介绍 COperaServerDlg 的代码时，进一步描述有关内容。

3. 客户套接字类

现在，让我们看看客户套接字类。该类用于处理与一个客户的交互。你需要重载两个方法：OnClose() 和 OnReceive()。当套接字关闭时，调用 OnClose()；当套接字接收到数据时，调用 OnReceive()。以下是类定义：

程序清单 18-26

```
#include <comdef.h>

#define SMARTPTR_TYPEDEF(x) \
    typedef _com_ptr_t<_com_IID<x, &IID_#x> > x##Ptr;

#include "..\SimpleOrder\SimpleOrder.h"

SMARTPTR_TYPEDEF(IOrder);

#import "msxml.dll"

////////////////////////////////////
// COperaClientSocket command target

class COperaClientSocket : public CAsyncSocket
```

```

{
// Attributes
public:

// Operations
public:
    COperaClientSocket();
    virtual ~COperaClientSocket();

// Overrides
public:
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(COperaClientSocket)
    virtual void OnClose(int nErrorCode);
    virtual void OnReceive(int nErrorCode);
    //}}AFX_VIRTUAL

    // Generated message map functions
    //{{AFX_MSG(COperaClientSocket)
    // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG

// Implementation
protected:
    IOrderPtr m_pOrder;

    void ReplyUnsupported();
    void ReplyBadRequest();
    void ProcessContent(int nChar, LPSTR lpszContent);

public:
    bool m_bActive;
};

```

请注意定义前面的智能指针部分（#define SMARTPTR_TYPEDEF(x)等）——我们可以利用它在不跟踪引用计数的情况下访问订单COM对象。由于这是ATL的特征，而不是MFC自动提供的，因此使用它要略微复杂一些。另外，#import行建立了访问MSXML COM对象所需的智能指针。

客户套接字类的构造方法和析构方法非常普通：

程序清单 18-27

```

COperaClientSocket::COperaClientSocket() : m_bActive(true)
{
}

COperaClientSocket::~COperaClientSocket()
{
}

```

重载的OnClose()方法也非常简单：

程序清单 18-28

```

void COperaClientSocket::OnClose(int nErrorCode)
{
    CWinApp *pWinApp = AfxGetApp();
    COperaServerDlg *pDialog =

```

```

        static_cast<COperaServerDlg *> (pWinApp->m_pMainWnd);

        CString csStatus;

        if (nErrorCode)
            csStatus.Format("Closed with reason %d", nErrorCode);
        else
            csStatus.Format("Closed gracefully");

        pDialog->Status(csStatus);

        m_bActive = false;

        CAsyncSocket::OnClose(nErrorCode);
    }

```

它向主对话框输出适当的消息，然后将激活标志设置为 false，这样主套接口的 Check() 方法下次检查时就能够发现该套接口关闭了。

重载的 OnReceive() 方法比较复杂：

程序清单 18-29

```

void COperaClientSocket::OnReceive(int nErrorCode)
{
    CWinApp *pWinApp = AfxGetApp();
    COperaServerDlg *pDialog =
        static_cast<COperaServerDlg *> (pWinApp->m_pMainWnd);

    CString csStatus;

    if (nErrorCode)
    {
        csStatus.Format("OnReceive called with error code %d", nErrorCode);
        pDialog->Status(csStatus);
    }
    else
    {

```

如果发现错误状态，我们要将请求的内容读入缓存。严格意义上讲，我们应该将它拷贝至循环缓存，当接收完一行之后再进行处理。TCP/IP 不介意消息的边界，它可能先接收半个消息，或者一个半消息。然而，为了简化程序，我们假设它仅接收一个完整的消息。消息解析过程与 OperaClient COM 对象中的解析过程极其类似：

程序清单 18-30

```

char szRequest[20000];
int nChar = Receive(szRequest, sizeof(szRequest), 0);

if (nChar > 0)
{
    szRequest[nChar] = 0;
    pDialog->Status("Data received:");

    LPSTR lpszReturn = strstr(szRequest, "\r\n");

    if (!lpszReturn)
    {

```

```

    ReplyBadRequest();
    CAsyncSocket::OnReceive(nErrorCode);
    return;
}

*lpzReturn = 0;

csStatus.Format(" %s", szRequest);
pDialog->Status(csStatus);

if (strcmp(szRequest, "POST /OperaServer HTTP/1.1"))
{
    ReplyUnsupported();
    CAsyncSocket::OnReceive(nErrorCode);
    return;
}

bool bSoap = false;

while (1)
{
    LPSTR lpzSegment = lpzReturn + 2;
    lpzReturn = strstr(lpzSegment, "\r\n");

    if (!lpzReturn)
    {
        ReplyUnsupported();
        break;
    }

    *lpzReturn = 0;

    csStatus.Format(" %s", lpzSegment);
    pDialog->Status(csStatus);

    if (!strcmp(lpzSegment, "Content-Type: ", 14))
    {
        if (!strcmp(&lpzSegment[14], "text/xml"))
            bSoap = true;
        else
        {
            ReplyUnsupported();
            break;
        }
    }
    else if (!strcmp(lpzSegment, "Content-Length: ", 16))
    {
        if (!bSoap)
        {
            ReplyBadRequest();
            break;
        }
        int nChar = atoi(&lpzSegment[16]);
        ProcessContent(nChar, lpzReturn + 2);

        break;
    }
}
}

```

```

    }
}

CAsyncSocket::OnReceive(nErrorCode);
}

```

以上代码与 OperaClient COM对象中相应代码的主要区别体现在以下两方面：以上代码使用 Receive()方法，而不是 recv() API调用；以上代码通过将消息发送回发出调用的客户进行错误处理，而不是使用 AltReportError()报告给客户。

与COM对象类似，真正的核心是 ProcessContent()方法：

程序清单 18-31

```

void COperaClientSocket::ProcessContent(int nChar, LPSTR lpszContent)
{
    if (strncmp(lpszContent, "SOAPMethodName: www.jpassoc.co.uk/opera#", 40))
    {
        ReplyBadRequest();
        return;
    }

    LPSTR lpszMethod = &lpszContent[40];
    LPSTR lpszReturn = strchr(lpszMethod, '\r');

    if (!lpszReturn)
    {
        ReplyBadRequest();
        return;
    }

    *lpszReturn = NULL;
}

```

此处的代码与 OperaClient COM对象在需求上存在差别：如果不存在订单对象，我们需要创建一个新的订单对象：

程序清单 18-32

```

if (m_pOrder == NULL)
    CoCreateInstance(CLSID_Order, NULL, CLSCTX_LOCAL_SERVER, IID_IOrder,
        (void **) &m_pOrder);

```

然后，我们可以通过类似的方式利用 XML DOM解析内容：

程序清单 18-33

```

MSXML::IXMLDOMDocumentPtr pXML;
CoCreateInstance(__uuidof(MSXML::DOMDocument), NULL, CLSCTX_INPROC_SERVER,
    __uuidof(MSXML::IXMLDOMDocument), (void **) &pXML);

CString csXML = lpszReturn + 1;
BSTR bstrXML = csXML.AllocSysString();

VARIANT_BOOL success;
success = pXML->loadXML(bstrXML);

::SysFreeString(bstrXML);

```



```
MSXML::IXMLDOMElementPtr pElem;
pXML->get_documentElement(&pElem);

bool bClose = false;

HRESULT hResult = S_OK;
```

然而，这次我们真正要调用方法了。首先是一个无参数的方法：

程序清单 18-34

```
if (!strcmp(lpszMethod, "Submit"))
{
    hResult = m_pOrder->Submit();
    bClose = true;
}
else
{
    MSXML::IXMLDOMNodePtr pChild;
    pElem->get_firstChild(&pChild);

    BSTR bstrArg;
    pChild->get_text(&bstrArg);
```

然后是含一个参数的方法：

程序清单 18-35

```
if (!strcmp(lpszMethod, "put_CustomerID"))
    hResult = m_pOrder->put_CustomerID(bstrArg);
else if (!strcmp(lpszMethod, "put_Title"))
    hResult = m_pOrder->put_Title(bstrArg);
else if (!strcmp(lpszMethod, "put_Artist"))
    hResult = m_pOrder->put_Artist(bstrArg);
else if (!strcmp(lpszMethod, "put_Label"))
    hResult = m_pOrder->put_Label(bstrArg);
else if (!strcmp(lpszMethod, "put_Price"))
    hResult = m_pOrder->put_Price(bstrArg);

SysFreeString(bstrArg);
}
```

最后，我们要构造包含结果的SOAP应答，并发送它：

程序清单 18-36

```
char szCallResponse[2000];

strcpy(szCallResponse,
    "<SOAP:Envelope xmlns:SOAP=\"urn:schemas-xmlsoap-org:soap.v1\">\r\n");
strcat(szCallResponse, "<SOAP:Body>\r\n");

size_t length = strlen(szCallResponse);
sprintf(&szCallResponse[length],
    "<m:%sResponse xmlns:m=\"www.jpassoc.co.uk/opera\">\r\n",
    lpszMethod);

length = strlen(szCallResponse);
sprintf(&szCallResponse[length], "        <return>%d</return>\r\n",
```

```

        hResult);

length = strlen(szCallResponse);
sprintf(&szCallResponse[length], "                </m:%sResponse>\r\n"
        "                </SOAP:Body>\r\n    </SOAP:Envelope>", lpszMethod);

length = strlen(szCallResponse);

char szResponse[2000];

if (bClose)
    sprintf(szResponse, "HTTP/1.1 200 OK\r\nConnection: close\r\n"
        "Content-Type: text/xml\r\nContent-Length: %d\r\n%s",
        length, szCallResponse);
else
    sprintf(szResponse, "HTTP/1.1 200 OK\r\nConnection: Keep-Alive\r\n"
        "Content-Type: text/xml\r\nContent-Length: %d\r\n%s",
        length, szCallResponse);

Send(szResponse, strlen(szResponse), 0);
}

```

另外，还有两个辅助方法，它的功能非常清晰：

程序清单 18-37

```

void COperaClientSocket::ReplyUnsupported()
{
    CWinApp *pWinApp = AfxGetApp();
    COperaServerDlg *pDialog =
        static_cast<COperaServerDlg *> (pWinApp->m_pMainWnd);
    pDialog->Status ("Replying \"Not supported\"");

    Send("HTTP/1.1 501 Not supported", 25, 0);
}

void COperaClientSocket::ReplyBadRequest()
{
    CWinApp *pWinApp = AfxGetApp();
    COperaServerDlg *pDialog =
        static_cast<COperaServerDlg *> (pWinApp->m_pMainWnd);
    pDialog->Status("Replying \"Bad request\"");

    Send("HTTP/1.1 400 Bad request", 23, 0);
}

```

需要特别说明的是，你可以构造一个更复杂的 OPERA 服务器，它将过滤并处理相关的 SOAP 调用，而将其他请求发送给另一台运行真正的 Web 服务器的机器。

4. 主对话框

最后，我们快速介绍一下主对话框。除了 OnInitDialog() 之外，还有一个重载方法：OnTimer()。

对话框本身非常简单，它包含一个无序列表框 IDC_STATUS，以及三个按钮：IDC_START，IDC_STOP 和 IDOK（参见图 18-4）。

与前两个按钮相关的 _CLICKED 方法分别称为：OnStart() 和 OnStop()。可以通过 ClassView 添加这两个方法，除此之外，还要添加与主对话框相关的 WM_TIMER 处理器（OnTimer()）。以下是类定义：



图 18-4

程序清单 18-38

```

#define MAX_STATUS 1000

class COperaMainSocket;

////////////////////////////////////
// COperaServerDlg dialog

class COperaServerDlg : public CDialog
{
// Construction
public:
    COperaServerDlg(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
//{{AFX_DATA(COperaServerDlg)
enum { IDD = IDD_OPERASERVER_DIALOG };
// NOTE: the ClassWizard will add data members here
//}}AFX_DATA

// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(COperaServerDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
public:
    void Status(CString csStatus);

protected:
    HICON m_hIcon;

    COperaMainSocket *m_pSocket;

    void Enable(int nID, BOOL bEnable);

// Generated message map functions

```

```

//{{AFX_MSG(COperaServerDlg)
virtual BOOL OnInitDialog();
afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
afx_msg void OnPaint();
afx_msg HCURSOR OnQueryDragIcon();
afx_msg void OnStart();
afx_msg void OnStop();
afx_msg void OnTimer(UINT nIDEvent);
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

```

以下是重载的OnInitDialog()方法：

程序清单 18-39

```

BOOL COperaServerDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // Set the icon for this dialog. The framework does this automatically
    // when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);        // Set big icon
    SetIcon(m_hIcon, FALSE);       // Set small icon

    CoInitialize(NULL);

    SetTimer(1, 1000, NULL);

    return TRUE; // return TRUE unless you set the focus to a control
}

```

在以上代码中，我们初始化 COM 对象，设置计时器，该计时器将用于检查套接字是否仍然激活。你应该记得，在 COperaMainSocket 代码中，我们利用一个方法检查客户套接字是否仍然激活。检查套接字的方法是在 COperaServerDlg 的 OnTimer() 重载方法中调用的：

程序清单 18-40

```

void COperaServerDlg::OnTimer(UINT nIDEvent)
{
    if (m_pSocket)
        m_pSocket->Check();
}

```

```
    CDialog::OnTimer(nIDEvent);  
}
```

当点击Start按钮，启动OPERA服务器后，会执行哪些操作？以下是 OnStart()方法：

程序清单 18-41

```
void COperaServerDlg::OnStart()  
{  
    CString csStatus;  
  
    m_pSocket = new COperaMainSocket();  
    if (!m_pSocket->Create(80))  
    {  
        csStatus.Format("Failed to create socket, reason code %d",  
                        m_pSocket->GetLastError());  
        Status(csStatus);  
  
        delete m_pSocket;  
        m_pSocket = NULL;  
  
        return;  
    }  
  
    if (!m_pSocket->Listen())  
    {  
        csStatus.Format("Failed to create socket, reason code %d",  
                        m_pSocket->GetLastError());  
        Status(csStatus);  
  
        delete m_pSocket;  
        m_pSocket = NULL;  
  
        return;  
    }  
  
    Enable(IDC_START, FALSE);  
    Enable(IDC_STOP, TRUE);  
}
```

同样，以上方法非常简明，它在端口 80创建主套接字，并监听该端口。此后，每当新的客户连接时，将调用 COperaMainSocket的OnAccept()重载方法。OnStop()方法更加简单：

程序清单 18-42

```
void COperaServerDlg::OnStop()  
{  
    delete m_pSocket;  
    m_pSocket = NULL;  
  
    Enable(IDC_START, TRUE);  
    Enable(IDC_STOP, FALSE);  
}
```

另外，我们还要确保在适当时刻只有恰当的按钮可用：

程序清单 18-43

```
void COperaServerDlg::Enable(int nID, BOOL bEnable)
{
    CWnd *pWnd = GetDlgItem(nID);
    pWnd->EnableWindow(bEnable);
}
```

最后一个方法用于将信息输出到状态列表框：

程序清单 18-44

```
void COperaServerDlg::Status(CString csStatus)
{
    CListBox *pStatus = static_cast<CListBox *> (GetDlgItem(IDC_STATUS));

    int count = pStatus->GetCount();
    if (count >= MAX_STATUS)
        pStatus->DeleteString(0);

    time_t now = time(NULL);
    struct tm *pTm = gmtime(&now);

    CString csFullStatus;
    csFullStatus.Format("%02d:%02d:%02d %s", pTm->tm_hour, pTm->tm_min,
        pTm->tm_sec, static_cast<LPCSTR> (csStatus));

    int index = pStatus->AddString(csFullStatus);
    pStatus->SetTopIndex(index);
}
```

5. 快速的测试

在我们连接客户端之前，首先单独测试一下服务器，运行它，并使用Internet Explorer 5连接它。为此，只需要点击Start按钮，服务器启动之后，通过浏览器访问http://localhost/test（参见图18-5）。

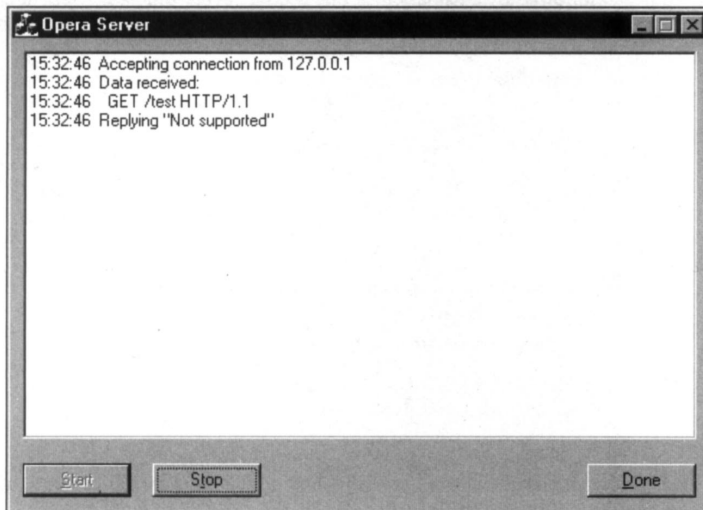


图 18-5

Internet Explorer将与本地Web服务器（换句话说，OperaServer）建立连接，并向它发送用于测试的GET请求。以上就是我们期望看到的结果，因此我们现在可以测试 OPERA客户端。我们将保持服务器继续处在运行状态，并准备接收任何真正的调用请求。

18.2.3 订单输入应用程序

对于我们真正的订单输入程序，将采用《Professional DCOM Application Development》中的程序，并做一些小的改动。它是用 Visual Basic 创建的。图18-6是唯一的一个表单frmOrderEntry。

我们有五个文本字段：txtCustomer，txtTitle，txtArtist，txtLabel和txtPrice。另外，还有一个命名按钮cmdConfirm和一个（位于表单底部隐含的）标签lblError。

以下是 Visual Basic 代码，其中突出显示的行是在原来的面向DCOM的版本基础上修改的：

图 18-6

程序清单 18-45

```
Option Explicit
Const conHost = "laa-laa"
'Dim m_order As SIMPLEORDERLib.Order
Dim m_opera As OPERACLIENLib.Opera

Private Sub HandleError(ctrl As MSForms.Control)
    If (Err.Number = 5) Then
        lblError.Caption = "Invalid input - please re-enter"
    ElseIf (Err.Number = &H80004005) Then
        lblError.Caption = "This field is mandatory"
    Else
        lblError.Caption = Err.Description
    End If
    ctrl.Tag = False
End Sub

Private Sub HandleSuccess(ctrl As MSForms.Control)
    lblError.Caption = ""
    ctrl.Tag = True
End Sub

Private Sub UserForm_Initialize()
    ' Set m_order = New SIMPLEORDERLib.Order
    Set m_opera = New OPERACLIENLib.Opera
End Sub

Private Sub txtCustomerID_Exit(ByVal Cancel As MSForms.ReturnBoolean)
On Error GoTo TryAgain
    ' m_order.CustomerID = txtCustomerID.Text
    Call m_opera.Exchange(conHost, "put_CustomerID", txtCustomerID.Text)
    HandleSuccess txtCustomerID
Exit Sub
TryAgain:

```



```

TryAgain:
    HandleError txtCustomerID
    Cancel = True
End Sub

Private Sub txtTitle_Exit(ByVal Cancel As MSForms.ReturnBoolean)
On Error GoTo TryAgain
'    m_order.Title = txtTitle.Text
    Call m_opera.Exchange(conHost, "put_Title", txtTitle.Text)
    HandleSuccess txtTitle
Exit Sub

TryAgain:
    HandleError txtTitle
    Cancel = True
End Sub

Private Sub txtArtist_Exit(ByVal Cancel As MSForms.ReturnBoolean)
On Error GoTo TryAgain
'    m_order.Artist = txtArtist.Text
    Call m_opera.Exchange(conHost, "put_Artist", txtArtist.Text)
    HandleSuccess txtArtist
Exit Sub

TryAgain:
    HandleError txtArtist
    Cancel = True
End Sub

Private Sub txtLabel_Exit(ByVal Cancel As MSForms.ReturnBoolean)
On Error GoTo TryAgain
'    m_order.Label = txtLabel.Text
    Call m_opera.Exchange(conHost, "put_Label", txtLabel.Text)
    HandleSuccess txtLabel
Exit Sub

TryAgain:
    HandleError txtLabel
    Cancel = True
End Sub

Private Sub txtPrice_Exit(ByVal Cancel As MSForms.ReturnBoolean)
On Error GoTo TryAgain
'    m_order.Price = txtPrice.Text
    Call m_opera.Exchange(conHost, "put_Price", txtPrice.Text)
    HandleSuccess txtPrice
Exit Sub

TryAgain:
    HandleError txtPrice
    Cancel = True
End Sub

Private Sub cmdConfirm_Click()

' Make sure that all mandatory controls have been filled
Dim ctrl As MSForms.Control
For Each ctrl In Me.Controls
    If InStr(1, ctrl.Name, "txt", vbBinaryCompare) = 1 Then
        If ctrl.Tag <> "True" Then
            lblError = "This field is mandatory"
            ctrl.SetFocus
            Exit Sub
        End If
    End If
End Sub

```

```
End If
Next ctrl

' m_order.Submit
Call m_opera.Exchange(conHost, "Submit", Empty)
Unload Me
End Sub
```

显然，你需要修改 conHost 的值，使之匹配运行服务器的主机名称（或者将服务器名称改为 laa-laa）。

正如你所看到的，我们已经将所有显式的 DCOM 调用替换为通过 OPERA 客户的 SOAP 调用（我们已经实现了本章开始处讨论的定制排列，虽然我们不一定非要实现它）。假设服务器已经在 Tierra del Fuego 运行，当我们的客户在 Ulaanbataar 启动后，看看会出现什么情况？正如我们所知，蒙古的流行趋势已经发生变化，六十年代后期的迷幻乐不再流行。要想获得有限版本的 CD，只能加入 King Crimson Collectors 俱乐部（参见图 18-7）。

假设服务器仍然在主机 laa-laa 上运行。那么，我们在服务器上看到什么？图 18-8 是服务器端的输出：

看起来非常不错。

在继续讨论之前，我们最好看一下订单文件。利用 dfview，我们将看到图 18-9 所示内容：

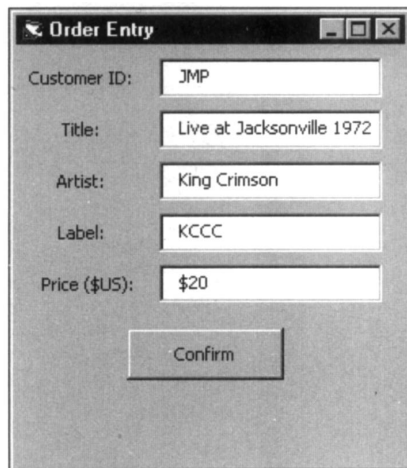


图 18-7

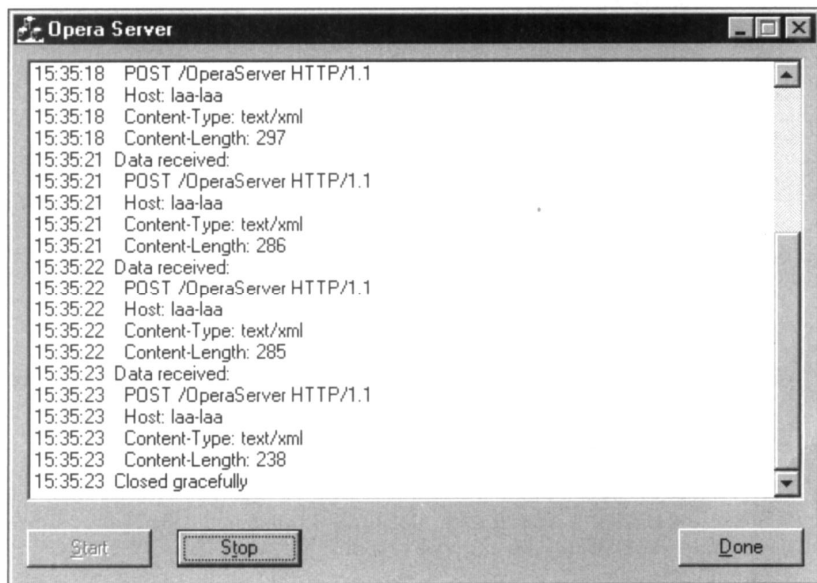


图 18-8

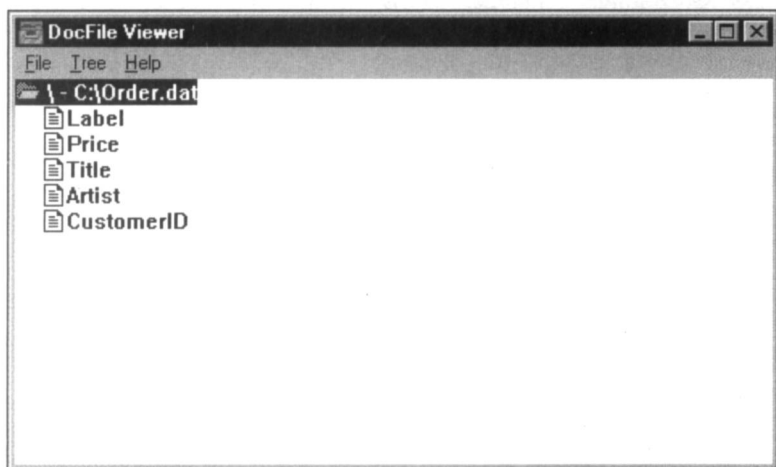


图 18-9

双击标题，你就能够看到订单的内容（参见图 18-10）。

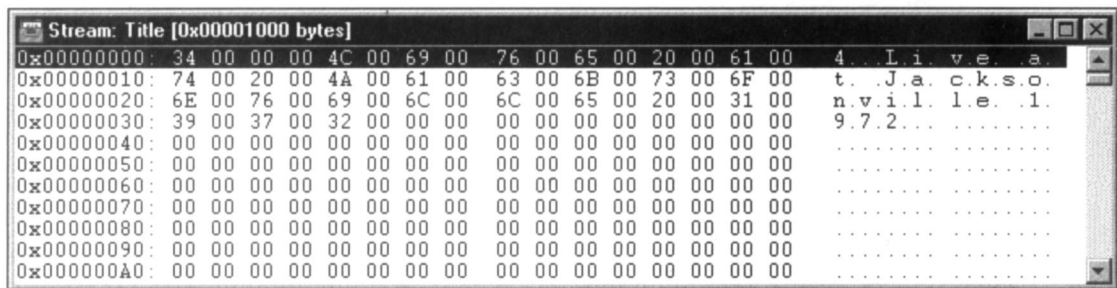


图 18-10

当然，如果从头开始编写，而不是使用旧的应用程序，自然会将 XML 作为文件格式。但是你不需重写所有代码。SOAP 并不会改变你的编程习惯。

这就是 SOAP。但是，还有一个问题……

18.2.4 对象

在 Internet 上，没有人知道你的身份。更有甚者，也没有人知道你的服务器的身份。如果你没有面向对象的 SOAP 服务器怎么办？如果你只有一个不太可靠且只能运行标准 C 的旧 Unix 程序怎么办？如果你仔细考虑一下，SOAP 并不是特别面向对象的。它只是通过防火墙发送函数调用的方法。完全可以不捕获这些 SOAP 调用，并将它们定位到旧的 C 程序的函数调用。

因此，完全可以用 SOAP 整理旧系统。仅仅出于兴趣，我认为我们可以研究一下这种用旧式标准 C 编写的旧系统（不，我不能忍受回到 Fortran 或 Cobol，我担心——这与新千年的第一本书应有的特征相差太远了，新书应该致力于新内容）。然而，我认为标准 C 仍然有重新流行的可能性。在本节中，我将提供 ScumServer。

这个程序将运行在旧的 HP/UX系统上，当你读到本书时，它可能已经无法运行，因为我不能保证它是Y2K兼容的。在HP/UX系统上试验成功后，我又将它移植到相当流行的 Linux系统上，它同样能够运行（当然，我要做一些改动，消除某些编译警告，并且增加了 htons()函数，它用于将端口号从主机标准转化为网络标准）。下面我们进行详细介绍。

我们像往常一样从C的头文件开始，另外还要增加 Unix不提供的几个定义：

程序清单 18-46

```
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define TRUE 1
#define FALSE 0

#define S_OK 0
#define E_INVALIDARG 0x80070057

#define FAILED(result) (result != S_OK)
```

以下结构用于存放与每个进入的客户套接字相关的信息（因此，它等价于COperaClientSocket）。此外，它还要存放当前用户的订单信息，所以它或许可以称为一种会话结构：

程序清单 18-47

```
typedef struct tagSocket {
    struct tagSocket *pNext;
    struct tagSocket *pPrev;
    int clientSock;
    char *customerID;
    char *artist;
    char *title;
    char *label;
    char *price;
} SOCKET;
```

这是套接字结构链接列表的开始：

```
SOCKET sockHead = {NULL, NULL};
```

下面是几个函数原型（实际上，这个特定机器上的编译器不支持成熟的函数原型）：

程序清单 18-48

```
SOCKET* addSocket();
void deleteSocket();
void processRequest();
void processContent();
void replyUnsupported();
void replyBadRequest();
int put_CustomerID();
```

```

int    put_Artist();
int    put_Title();
int    put_Label();
int    put_Price();
int    submit();
int    isAlphabetic();
int    isAlphanumeric();
int    isPrice();

```

以下是主程序：

程序清单 18-49

```

main(argc, argv)
int argc;
char **argv;
{
    int    mainSock;
    int    on;
    struct sockaddr_in address;
    struct timeval timeout;
    int    maxSock;
    fd_set readMask;
    int    addrLen;
    int    clientSock;
    int    nRead;
    char    buffer[2000];
    SOCKET *pSocket;
    SOCKET *pNextSock;

```

我们首先创建套接字，将它绑定到端口 80，并监听进入的调用。注意，函数 `htons()` 用于保证 Linux 有适当格式的端口号：

程序清单 18-50

```

mainSock = socket(AF_INET, SOCK_STREAM, 0);

on = 1;
setsockopt(mainSock, SOL_SOCKET, SO_REUSEADDR, (void *) &on,
           sizeof(on));

address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(80);

if (bind(mainSock, (struct sockaddr *) &address, sizeof(address)) < 0)
{
    perror("Failed to bind to socket");
    exit(0);
}

listen(mainSock, 5);

while (1)
{

```

现在，我们准备最重要的 select 调用。我们寻找主套接字以及与客户相连的套接字上的读通知。主套接字上的读通知相当于进入调用信号。

程序清单 18-51

```
FD_ZERO(&readMask);
FD_SET(mainSock, &readMask);

maxSock = mainSock;

pSocket = sockHead.pNext;

while (pSocket)
{
    FD_SET(pSocket->clientSock, &readMask);

    if (maxSock < pSocket->clientSock)
        maxSock = pSocket->clientSock;

    pSocket = pSocket->pNext;
}

timeout.tv_sec = 0;
timeout.tv_usec = 100;

select(maxSock + 1, &readMask, (fd_set *) 0, (fd_set *) 0,
        &timeout);
```

现在，检查进入的调用。如果有调用，接受它，并创建新的 SOCKET 结构，并使用辅助函数 addSocket 将它添加到链接列表中：

程序清单 18-52

```
if (FD_ISSET(mainSock, &readMask))
{
    addrLen = sizeof(struct sockaddr_in);
    clientSock = accept(mainSock, (struct sockaddr *) &address,
                        &addrLen);

    printf("Connection accepted\n");

    addSocket(clientSock);
}
```

现在，我们检查链接列表中的每个客户套接字的读通知：

程序清单 18-53

```
pSocket = sockHead.pNext;

while (pSocket)
{
    pNextSock = pSocket->pNext;

    if (FD_ISSET(pSocket->clientSock, &readMask))
```

```
{
    nRead = recv (pSocket->clientSock, buffer, sizeof(buffer), 0);
```

如果不存在读通知，即便设置了读信号，套接字也将被关闭，或者产生错误，因此我们将删除套接字结构。否则，我们处理请求：

程序清单 18-54

```
        if (nRead <= 0)
            deleteSocket(pSocket);

        else
        {
            buffer[nRead] = 0;
            processRequest(pSocket, buffer);
        }

        pSocket = pNextSock;
    }
}
```

以上就是main()函数。以下是处理链接列表所需的几个辅助函数：

程序清单 18-55

```
SOCKET* addSocket(clientSock)
int clientSock;
{
    SOCKET *pSocket;

    printf("Adding new connection\n");

    pSocket = (SOCKET *) malloc(sizeof(SOCKET));
    memset(pSocket, 0, sizeof(SOCKET));

    pSocket->clientSock = clientSock;

    if (sockHead.pPrev)
    {
        pSocket->pPrev = sockHead.pPrev;
        (sockHead.pPrev)->pNext = pSocket;
    }
    else
    {
        pSocket->pPrev = &sockHead;
        sockHead.pNext = pSocket;
    }

    sockHead.pPrev = pSocket;

    return pSocket;
}

void deleteSocket(pSocket)
SOCKET *pSocket;
```



```

{
    printf("Deleting connection\n");
    (pSocket->pPrev)->pNext = pSocket->pNext;

    if (pSocket->pNext)
        (pSocket->pNext)->pPrev = pSocket->pPrev;
    else
        sockHead.pPrev = pSocket->pPrev;

    free(pSocket);
}

```

现在来看ScumServer。如果你看过COperaClientSocket的代码，可能觉得以下代码有些眼熟。实际上，它们几乎完全一致：

程序清单 18-56

```

void processRequest(pSocket, request)
SOCKET *pSocket;
char request[];
{
    char *pReturn;
    int bSoap;
    char *pSegment;
    int nChar;

    printf("Data received:\n");

    pReturn = strstr(request, "\r\n");

    if (!pReturn)
    {
        replyBadRequest(pSocket);
        return;
    }

    *pReturn = 0;

    printf(" %s\n", request);

    if (strcmp(request, "POST /OperaServer HTTP/1.1"))
    {
        replyUnsupported(pSocket);
        return;
    }

    bSoap = FALSE;

    while (1)
    {
        pSegment = pReturn + 2;
        pReturn = strstr(pSegment, "\r\n");

        if (!pReturn)
        {
            replyUnsupported(pSocket);

```

```

        return;
    }
    *pReturn = 0;

    printf("  %s\n", pSegment);

    if (!strcmp(pSegment, "Content-Type: ", 14))
    {
        if (!strcmp(&pSegment[14], "text/xml"))
            bSoap = TRUE;
        else
        {
            replyUnsupported(pSocket);
            break;
        }
    }
    else if (!strcmp(pSegment, "Content-Length: ", 16))
    {
        if (!bSoap)
        {
            replyBadRequest(pSocket);
            break;
        }
        nChar = atoi (&pSegment[16]);
        processContent (pSocket, nChar, pReturn + 2);

        break;
    }
}
}
}

```

然而，处理 XML 内容的函数要略微复杂一些，因为我们无法依靠任何 XML DOM。我们不得不自己解析 XML。事实上，情况没有你想象的那么糟，因为我们使用的是 XML 的一个很小的子集，而且我们不处理非常复杂的 SOAP 消息结构。

程序清单 18-57

```

void processContent(pSocket, nChar, content)
SOCKET *pSocket;
int nChar;
char content[];
{
    char *pMethod;
    char *pReturn;
    char *pXML;
    int bClose;
    int result;
    int bArgument;
    char *pElement;
    char *pValue;
    char callResponse[2000];
    size_t length;
    char response[2000];

    if (strcmp (content, "SOAPMethodName: www.jpassoc.co.uk/opera#", 40))
    {
        replyBadRequest (pSocket);
    }
}

```

```

    return;
}

pMethod = &content[40];
pReturn = strchr (pMethod, '\r');

if (!pReturn)
{
    replyBadRequest (pSocket);
    return;
}

*pReturn = NULL;

pXML = pReturn + 1;

```

以上代码都与 COperaClientSocket::ProcessContent 非常类似。然而，从此开始就要体现出两者的区别，因为 OperaServer 要开始处理 XML DOM：

```

bClose = FALSE;
result = S_OK;

```

如果通过 OPERA 调用的方法恰好是 Submit()，我们不需要进行更多的解析操作：

程序清单 18-58

```

if (!strcmp(pMethod, "Submit"))
{
    result = submit(pSocket);
    bClose = TRUE;
}
else
{

```

然而，如果调用了某个 put_ 方法，就需要提取参数值。我们将使用 strtok() 提取 XML 的各种元素。我们将处于以下两种状态之一：等待标记 (“<argument>”)，以及等待参数本身。标志 bArgument 用于确定到底属于哪种状态：

程序清单 18-59

```

bArgument = FALSE;

pElement = strtok(pXML, "<>\r\n");

while (pElement)
{
    if (bArgument)
    {

```

如果是标记，需要提取参数本身，假设它不是空值：

程序清单 18-60

```

if (!strcmp(pElement, "/argument"))
    pValue = "";
else

```

```

-----
pValue = pElement;

```

现在，已经得到了参数值，pValue指向该值。所以，继续调用请求的put_方法：

程序清单 18-61

```

if (!strcmp(pMethod, "put_CustomerID"))
    result = put_CustomerID(pSocket, pValue);
else if (!strcmp(pMethod, "put_Title"))
    result = put_Title(pSocket, pValue);
else if (!strcmp(pMethod, "put_Artist"))
    result = put_Artist(pSocket, pValue);
else if (!strcmp(pMethod, "put_Label"))
    result = put_Label(pSocket, pValue);
else if (!strcmp(pMethod, "put_Price"))
    result = put_Price(pSocket, pValue);
break;
}

```

如果我们仍然等待标记，检查是否获得了标记：

程序清单 18-62

```

else if (!strcmp (pElement, "argument"))
    bArgument = TRUE;

```

现在，我们继续XML中的下一个元素：

程序清单 18-63

```

    pElement = strtok(0, "<>\r\n");
}
}

if (!pElement)
    replyBadRequest (pSocket);

```

这还不错，是吗？从现在开始，我们又回到熟悉的代码。这与 COperaClientSocket::ProcessContent中对等的代码非常类似：

程序清单 18-64

```

strcpy(callResponse,
    " <SOAP:Envelope xmlns:SOAP=\"urn:schemas-xmlsoap-org:soap.v1\">\r\n");
strcat (callResponse, "          <SOAP:Body>\r\n");

length = strlen(callResponse);
sprintf(&callResponse[length],
    "          <m:%sResponse xmlns:m=\"www.jpassoc.co.uk/opera\">\r\n",
    pMethod);

length = strlen(callResponse);
sprintf(&callResponse[length], "          <return>%d</return>\r\n",
    result);

```

```

length = strlen(callResponse);
sprintf(&callResponse[length], "          </m:%sResponse>\r\n"
    "          </SOAP:Body>\r\n    </SOAP:Envelope>",
    pMethod);

length = strlen(callResponse);

if (bClose)
    sprintf(response,
        "HTTP/1.1 200 OK\r\nConnection: close\r\n"
        "Content-Type: text/xml\r\nContent-Length: %d\r\n%s",
        length, callResponse);
else
    sprintf(response,
        "HTTP/1.1 200 OK\r\nConnection: Keep-Alive\r\n"
        "Content-Type: text/xml\r\nContent-Length: %d\r\n%s",
        length, callResponse);

send (pSocket->clientSock, response, strlen (response), 0);
}

```

以下是同样熟悉的辅助函数，它报告返回给发出调用的应用程序的错误情况：

程序清单 18-65

```

void replyUnsupported(pSocket)
SOCKET *pSocket;
{
    send(pSocket->clientSock, "HTTP/1.1 501 Not supported", 25, 0);
}

void replyBadRequest(pSocket)
SOCKET *pSocket;
{
    send(pSocket->clientSock, "HTTP/1.1 400 Bad request", 23, 0);
}

```

现在，我们来看SimpleOrder的等价函数。以下例程用于建立订单中的字段：

程序清单 18-66

```

int put_CustomerID(pSocket, customerID)
SOCKET *pSocket;
char customerID[];
{
    int result;

    result = isAlphabetic(customerID);

    if (FAILED(result))
        return result;

    pSocket->customerID = malloc(strlen(customerID) + 1);
    strcpy(pSocket->customerID, customerID);

    return S_OK;
}

int put_Artist(pSocket, artist)

```

```
SOCKET *pSocket;
char artist[];
{
    int result;

    result = isAlphabetic(artist);

    if (FAILED(result))
        return result;

    pSocket->artist = malloc(strlen(artist) + 1);
    strcpy(pSocket->artist, artist);

    return S_OK;
}
```

```
int put_Title (pSocket, title)
SOCKET *pSocket;
char title[];
{
    int result;

    result = isAlphanumeric(title);

    if (FAILED(result))
        return result;

    pSocket->title = malloc(strlen(title) + 1);
    strcpy(pSocket->title, title);

    return S_OK;
}
```

```
int put_Label(pSocket, label)
SOCKET *pSocket;
char label[];
{
    int result;

    result = isAlphabetic(label);

    if (FAILED(result))
        return result;

    pSocket->label = malloc(strlen(label) + 1);
    strcpy(pSocket->label, label);

    return S_OK;
}
```

```
int put_Price(pSocket, price)
SOCKET *pSocket;
char price[];
{
    int result;

    result = isPrice(price);

    if (FAILED(result))
```

```

        return result;

    pSocket->price = malloc(strlen(price) + 1);
    strcpy(pSocket->price, price);

    return S_OK;
}

```

以下例程用于提交订单（在本示例中，它仅包含将订单的详细内容写入文件）：

程序清单 18-67

```

int submit(pSocket)
SOCKET *pSocket;
{
    FILE *pOrder;

    pOrder = fopen("order.dat", "w+");

    fprintf(pOrder, "CustomerID:\t%s\n", pSocket->customerID);
    fprintf(pOrder, "Artist:\t%s\n", pSocket->artist);
    fprintf(pOrder, "Title:\t%s\n", pSocket->title);
    fprintf(pOrder, "Label:\t%s\n", pSocket->label);
    fprintf(pOrder, "Price:\t%s\n", pSocket->price);

    fclose(pOrder);

    return S_OK;
}

```

最后是由于验证有效性的例程：

程序清单 18-68

```

int isAlphabetic (value)
char value[];
{
    size_t nChar;
    int iChar;

    nChar = strlen(value);
    for (iChar = 0; iChar < nChar; iChar++)
    {
        if (!isspace(value[iChar]) && !isalpha (value[iChar]))
            return E_INVALIDARG;
    }

    return S_OK;
}

int isAlphanumeric(value)
char value[];
{
    size_t nChar;
    int iChar;

    nChar = strlen(value);

```

```
for (iChar = 0; iChar < nChar; iChar++)
{
    if (!isspace (value[iChar]) && !isalnum (value[iChar]))
        return E_INVALIDARG;
}

return S_OK;
}

int isPrice(value)
char value[];
{
    size_t nChar;
    int iChar;

    if (value[0] != '$')
        return E_INVALIDARG;

    nChar = strlen(value);

    for (iChar = 1; iChar < nChar; iChar++)
    {
        if (!isdigit(value[iChar]) && (value[iChar] != '.'))
            return E_INVALIDARG;
    }

    return S_OK;
}
```

运行该程序后你会发现，仅从客户角度看，它与基于 COM 的 OperaServer 并无差别。同样，你需要保证没有其他程序占用端口 80。这就是我们将在客户看到的内容：

程序清单 18-69

```
Connection accepted
Adding new connection
Data received:
POST /OperaServer HTTP/1.1
Host: po
Content-Type: text/xml
Content-Length: 300
Data received:
POST /OperaServer HTTP/1.1
Host: po
Content-Type: text/xml
Content-Length: 306
Data received:
POST /OperaServer HTTP/1.1
Host: po
Content-Type: text/xml
Content-Length: 297
Data received:
POST /OperaServer HTTP/1.1
Host: po
Content-Type: text/xml
Content-Length: 286
```



```
Data received:
POST /OperaServer HTTP/1.1
Host: po
Content-Type: text/xml
Content-Length: 285
Data received:
POST /OperaServer HTTP/1.1
Host: po
Content-Type: text/xml
Content-Length: 238
Deleting connection
```

以下是order.dat文件的内容：

```
CustomerID: JMP
Artist: King Crimson
Title: The VROOOM Sessions 1994
Label: KCCC
Price: $15
```

因此，不要被 SOAP 中的字母“O”误导。在大多数单词中，它意味着对象。但是我们在本节中介绍的代码是用标准 C 编写的，它与对象没有任何瓜葛。

18.3 小结

SOAP 是一种消息协议，它能够封装远程对象的方法调用。它是用 XML 编写的，用 HTTP 作为传输协议（虽然规范并不强制要求这一点，它仅仅定义了与 HTTP 的绑定）。这使得它非常容易通过防火墙，特别是当它使用著名的端口 80。将来它将分配自己的标准端口。

实际上，SOAP 并不要求服务器采用面向对象的通信方式。它甚至不要求服务器将进入的 SOAP 消息看作远程过程调用。它将处理的自由完全留给了服务器。

因此，SOAP 到底有哪些优点？它利用目前广泛采纳的技术解决常见的问题——如何标准化消息协议。既然已经建立了 SOAP，各种类型的实现应用程序将迅速涌现，届时我们将能够实现大家在信息传递方面的各种需求。