第8章

集成点：网关、隧道及中继

事实证明，Web 是一种强大的内容发布工具。随着时间的流逝，人们已经从只在网上发送静态的在线文档，发展到共享更复杂的资源，比如数据库内容或动态生成的 HTML 页面。Web 浏览器这样的 HTTP 应用程序为用户提供了一种统一的方式来访问因特网上的内容。

HTTP 也已成为应用程序开发者的一种基本构造模块，开发者们可以在 HTTP 上捎回其他的协议内容（比如，可以将其他协议的流量包裹在 HTTP 中，用 HTTP 通过隧道或中继方式将这些流量传过公司的防火墙）。Web 上所有的资源都可以使用 HTTP 协议，而且其他应用程序和应用程序协议也可以利用 HTTP 来完成它们的任务。

本章简要介绍了一些开发者用 HTTP 访问不同资源的方法，展示了开发者如何将 HTTP 作为框架启动其他协议和应用程序通信。

本章会讨论：

- 在 HTTP 和其他协议及应用程序之间起到接口作用的网关；
- 允许不同类型的 Web 应用程序互相通信的应用程序接口；
- 允许用户在 HTTP 连接上发送非 HTTP 流量的隧道；
- 作为一种简化的 HTTP 代理，一次将数据转发一跳的中继。

8.1 网关

HTTP 扩展和接口的发展是由用户需求驱动的。要在 Web 上发布更复杂资源的需求
197 出现时，人们很快就明确了一点：单个应用程序无法处理所有这些能想到的资源。

为了解决这个问题，开发者提出了网关（gateway）的概念，网关可以作为某种翻译器使用，它抽象出了一种能够到达资源的方法。网关是资源和应用程序之间的粘合剂。应用程序可以（通过 HTTP 或其他已定义的接口）请求网关来处理某条请求，网关可以提供一条响应。网关可以向数据库发送查询语句，或者生成动态的内容，就像一个门一样：进去一条请求，出来一个响应。

图 8-1 显示的是一种资源网关。在这里，Joe 的五金商店服务器就是作为连接数据库内容的网关使用的——注意，客户端只是在通过 HTTP 请求资源，而 Joe 的五金商店的服务器在与网关进行交互以获取资源。

有些网关会自动将 HTTP 流量转换为其他协议，这样 HTTP 客户端无需了解其他协议，就可以与其他应用程序进行交互了（参见图 8-2）。

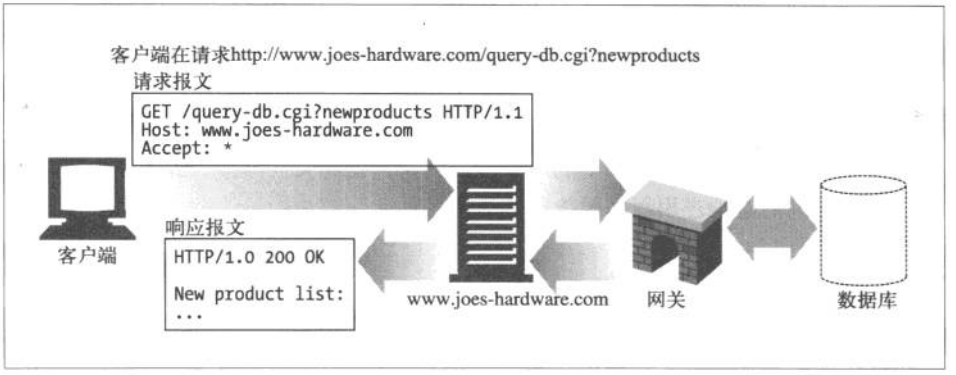


图 8-1　网关的魔力

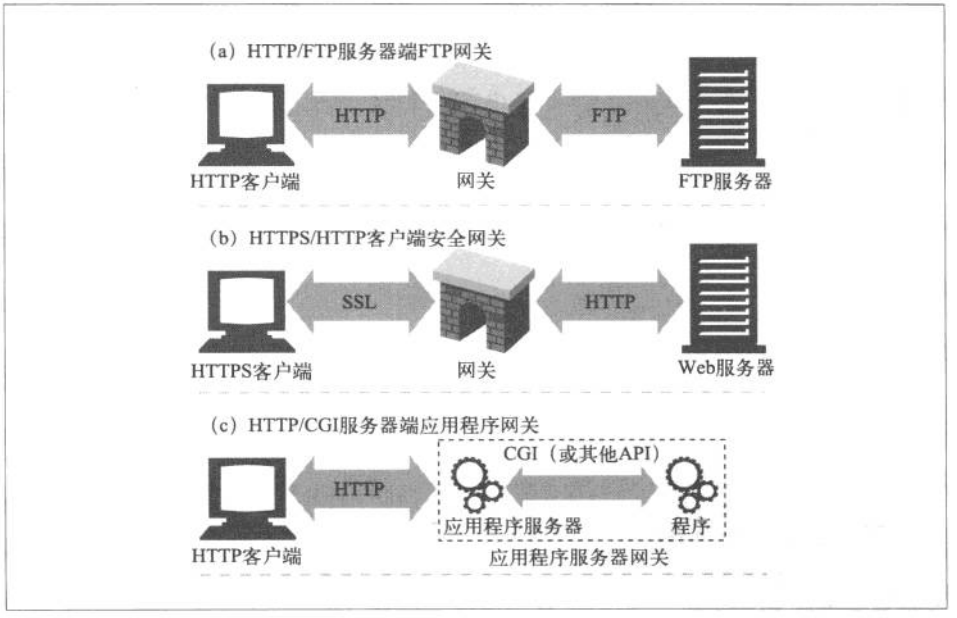


图 8-2　三个 Web 网关实例

图 8-2 显示了三个网关的示例。

- 在图 8-2a 中，网关收到了对 FTP URL 的 HTTP 请求。然后网关打开 FTP 连接，并向 FTP 服务器发布适当的命令。然后将文档和正确的 HTTP 首部通过 HTTP 回送。
- 在图 8-2b 中，网关通过 SSL 收到了一条加密的 Web 请求，网关会对请求进行解

密，[1]然后向目标服务器转发一条普通的 HTTP 请求。可以将这些安全加速器直
198 接放在（通常处于同一场所的）Web 服务器前面，以便为原始服务器提供高性能的加密机制。

在图 8-2c 中，网关通过应用程序服务器网关 API，将 HTTP 客户端连接到服务器端的应用程序上去。在网上的电子商店购物、查看天气预报，或者获取股票报价时，访问的就是应用程序服务器网关。

客户端和服务器端网关

Web 网关在一侧使用 HTTP 协议，在另一侧使用另一种协议。[2]

可以用一个斜杠来分隔客户端和服务器端协议，并以此对网关进行描述：

< 客户端协议 >/< 服务器端协议 >

因此，将 HTTP 客户端连接到 NNTP 新闻服务器的网关就是一个 HTTP/NNTP 网关。我们用术语*服务器端网关*和*客户端网关*来说明对话是在网关的哪一侧进行的。

- *服务器端网关*（server-side gateway）通过 HTTP 与客户端对话，通过其他协议与服务器通信（HTTP/*）。
- *客户端网关*（client-side gateway）通过其他协议与客户端对话，通过 HTTP 与服
199 务器通信（*/HTTP）。

8.2 协议网关

将 HTTP 流量导向网关时所使用的方式与将流量导向代理的方式相同。最常见的方式是，显式地配置浏览器使用网关，对流量进行透明的拦截，或者将网关配置为替代者（反向代理）。

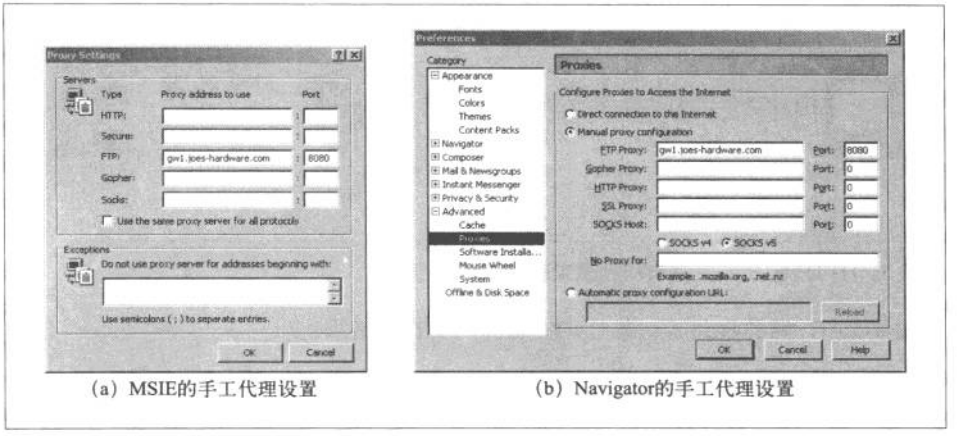
图 8-3 显示了配置浏览器使用服务器端 FTP 网关的对话框。在图中显示的配置中，配置浏览器将 gw1.joes-hardware.com 作为所有 FTP URL 的 HTTP/FTP 网关。浏览器没有将 FTP 命令发送给 FTP 服务器，而是将 HTTP 命令发送给端口 8080 上的 HTTP/FTP 网关 gw1.joes-hardware.com。

图 8-4 给出了这种网关配置的结果。一般的 HTTP 流量不受影响，会继续流入原始服务器。但对 FTP URL 的请求则被放在 HTTP 请求中发送给网关 gw1.joes-

注 1：网关上要安装适当的服务器证书。

注 2：在不同 HTTP 版本之间进行转换的 Web 代理就像网关一样，它们会执行复杂的逻辑，以便在各个端点之间进行沟通。但因为它们在两侧使用的都是 HTTP，所以从技术上来讲，它们还是代理。

hardware.com。网关代表客户端执行FTP事务，并通过HTTP将结果回送给客户端。



(a) MSIE的手工代理设置　　(b) Navigator的手工代理设置

图8-3　配置一个HTTP/FTP网关

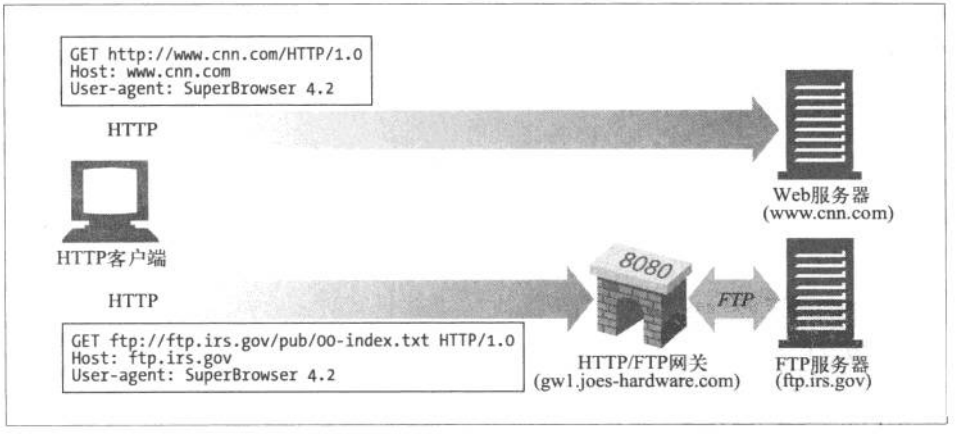


图8-4　浏览器可以通过配置，让特定的协议使用特定的网关

后面的小节会介绍各种常见网关类型：服务器协议转换器、服务器端安全网关、客户端安全网关以及应用程序服务器。

8.2.1　HTTP/*：服务器端Web网关

请求流入原始服务器时，服务器端Web网关会将客户端HTTP请求转换为其他协议（参见图8-5）。

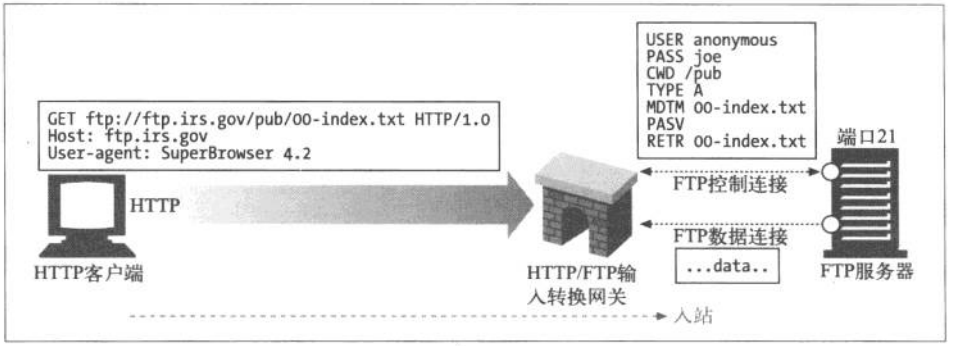


图 8-5 HTTP/FTP 网关将 HTTP 请求转换成 FTP 请求

200 在图 8-5 中，网关收到了一条对 FTP 资源的 HTTP 请求：

ftp://ftp.irs.gov/pub/00-index.txt

网关会打开一条到原始服务器 FTP 端口（端口 21）的 FTP 连接，通过 FTP 协议获取对象。网关会做下列事情：

- 发送 USER 和 PASS 命令登录到服务器上去；
- 发布 CWD 命令，转移到服务器上合适的目录中去；
- 将下载类型设置为 ASCII；
- 用 MDTM 获取文档的最后修改时间；
- 用 PASV 告诉服务器将有被动数据获取请求到达；
- 用 RETR 请求进行对象获取；
- 打开到 FTP 服务器的数据连接，服务器端口由控制信道返回；一旦数据信道打开了，就将对象内容回送给网关。

201

完成获取之后，会将对象放在一条 HTTP 响应中回送给客户端。

8.2.2 HTTP/HTTPS：服务器端安全网关

一个组织可以通过网关对所有的输入 Web 请求加密，以提供额外的隐私和安全性保护。客户端可以用普通的 HTTP 浏览 Web 内容，但网关会自动加密用户的对话（参见图 8-6）。

8.2.3 HTTPS/HTTP客户端安全加速器网关

最近，将 HTTPS/HTTP 网关作为安全加速器使用的情况是越来越多了。这些 HTTPS/HTTP 网关位于 Web 服务器之前，通常作为不可见的拦截网关或反向代理

使用。它们接收安全的 HTTPS 流量，对安全流量进行解密，并向 Web 服务器发送普通的 HTTP 请求（参见图 8-7）。

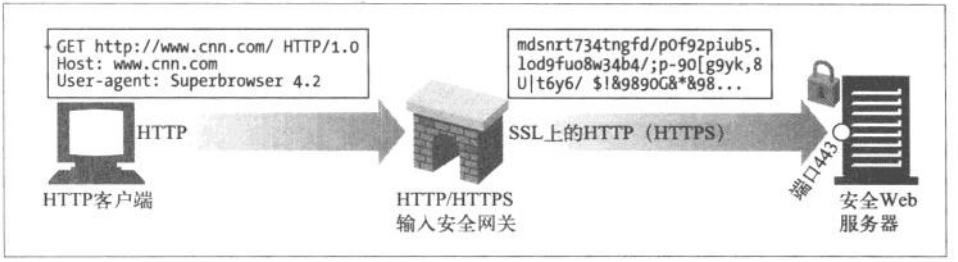


图 8-6　输入 HTTP/HTTPS 安全网关

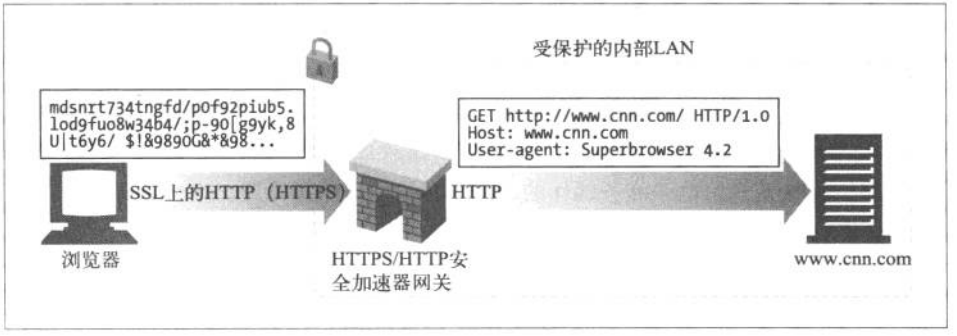


图 8-7　HTTPS/HTTP 安全加速器网关

这些网关中通常都包含专用的解密硬件，以比原始服务器有效得多的方式来解密安全流量，以减轻原始服务器的负荷。这些网关在网关和原始服务器之间发送的是未加密的流量，所以，要谨慎使用，确保网关和原始服务器之间的网络是安全的。 202

8.3　资源网关

到目前为止，我们一直在讨论通过网络连接客户端和服务器的网关。但最常见的网关，应用程序服务器，会将目标服务器与网关结合在一个服务器中实现。应用程序服务器是服务器端网关，与客户端通过 HTTP 进行通信，并与服务器端的应用程序相连（参见图 8-8）。

在图 8-8 中，两个客户端是通过 HTTP 连接到应用程序服务器的。但应用程序服务器并没有回送文件，而是将请求通过一个网关应用编程接口（Application Programming Interface，API）发送给运行在服务器上的应用程序。

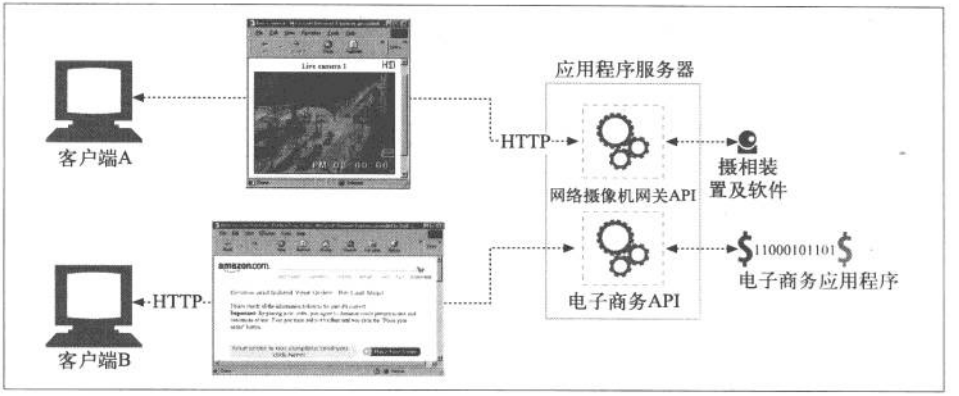


图 8-8　应用程序服务器可以将 HTTP 客户端连接任意后台应用程序

- 收到客户端 A 的请求，根据 URI 将其通过 API 发送给一个数码摄相机应用程序。将得到的图片绑定到一条 HTTP 响应报文中，再回送给客户端，在客户端的浏览器中显示。
- 客户端 B 的 URI 请求的是一个电子商务应用程序。客户端 B 的请求是通过服务器网关 API 发送给电子商务软件的，结果会被回送给浏览器。电子商务软件与客户端进行交互，引导用户通过一系列 HTML 页面来完成购物。

第一个流行的应用程序网关 API 就是*通用网关接口*（Common Gateway Interface，CGI）。CGI 是一个标准接口集，Web 服务器可以用它来装载程序以响应对特定 URL 的 HTTP 请求，并收集程序的输出数据，将其放在 HTTP 响应中回送。在过去的几年中，商业 Web 服务器提供了一些更复杂的接口，以便将 Web 服务器连接到
203 应用程序上去。

早期的 Web 服务器是相当简单的，在网关接口的实现过程中采用的简单方式一直持续到了今天。

请求需要使用网关的资源时，服务器会请辅助应用程序来处理请求。服务器会将辅助应用程序所需的数据传送给它。通常就是整条请求，或者用户想在数据库上运行的请求（来自 URL 的请求字符串，参见第 2 章）之类的东西。

然后，它会向服务器返回一条响应或响应数据，服务器则会将其转发给客户端。服务器和网关是相互独立的应用程序，因此，它们的责任是分得很清楚的。图 8-9 显示了服务器与网关应用程序之间交互的基本运行机制。

这个简单的协议（输入请求，转交，响应）就是最古老，也最常用的服务器扩展接口 CGI 的本质。

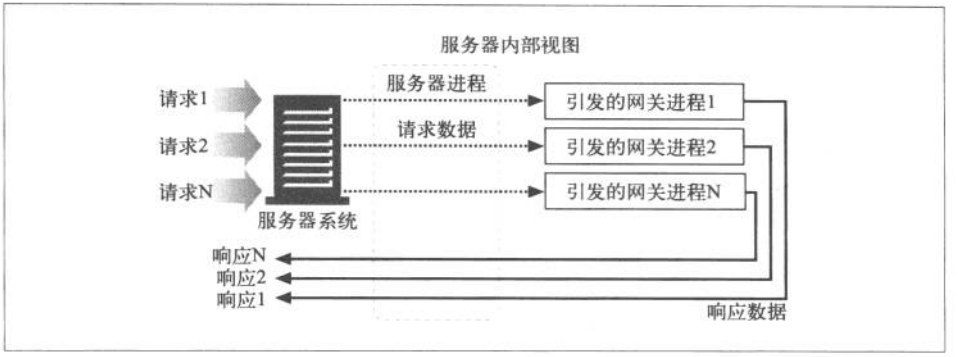


图 8-9 服务器网关应用程序机制

8.3.1 CGI

CGI 是第一个，可能仍然是得到最广泛使用的服务器扩展。在 Web 上广泛用于动态 HTML、信用卡处理以及数据库查询等任务。

CGI 应用程序是独立于服务器的，所以，几乎可以用任意语言来实现，包括 Perl、Tcl、C 和各种 shell 语言。CGI 很简单，几乎所有的 HTTP 服务器都支持它。图 8-9 显示了 CGI 模型的基本运行机制。

CGI 的处理对用户来说是不可见的。从客户端的角度来看，就像发起一个普通请求一样。它完全不清楚服务器和 CGI 应用程序之间的转接过程。URL 中出现字符 cgi 和可能出现的“?”是客户端发现使用了 CGI 应用程序的唯一线索。 204

看来 CGI 是很棒的，对吧？嗯，好吧，既是也不是。它在服务器和众多的资源类型之间提供了一种简单的、函数形式的粘合方式，用来处理各种需要的转换。这个接口还能很好地保护服务器，防止一些糟糕的扩展对它造成的破坏（如果这些扩展直接与服务器相连，造成的错误可能会引发服务器崩溃）。

但是，这种分离会造成性能的耗费。为每条 CGI 请求引发一个新进程的开销是很高的，会限制那些使用 CGI 的服务器的性能，并且会加重服务端机器资源的负担。为了解决这个问题，人们开发了一种新型 CGI——并将其恰当地称为*快速* CGI。这个接口模拟了 CGI，但它是作为持久守护进程运行的，消除了为每个请求建立或拆除新进程所带来的性能损耗。

8.3.2 服务器扩展API

CGI 协议为外部翻译器与现有的 HTTP 服务器提供了一种简洁的接口方式，但如

果想要改变服务器自身的行为，或者只是想尽可能地提升能从服务器上获得的性能呢？服务器开发者为这两种需求提供了几种服务器扩展 API，为 Web 开发者提供了强大的接口，以便他们将自己的模块与 HTTP 服务器直接相连。扩展 API 允许程序员将自己的代码嫁接到服务器上，或者用自己的代码将服务器的一个组件完整地替换出来。

大多数流行的服务器都会为开发者提供一个或多个扩展 API。这些扩展通常都会绑定在服务器自身的结构上，所以，大多数都是某种服务器类型特有的。微软、网景、Apache 和其他服务器都提供了一些 API 接口，允许开发者通过这些接口改变服务器的行为，或者为不同的资源提供一些定制的接口。这些定制接口为开发者提供了强大的接口方式。

微软的 FPSE（FrontPage 服务器端扩展）就是服务器扩展的一个实例，它为使用 FrontPage 的作者进行 Web 发布提供支持。FPSE 能够对 FrontPage 客户端发送的 RPC（remote procedure call，远程过程调用）命令进行解释。这些命令会在 HTTP 中（具体来说，就是在 HTTP POST 方法上）捎回。细节请参见 19.1 节。

8.4 应用程序接口和Web服务

我们已经讨论过可以将资源网关作为 Web 服务器与应用程序的通信方式使用。更广泛地说，随着 Web 应用程序提供的服务类型越来越多，有一点变得越来越清晰了：HTTP 可以作为一种连接应用程序的基础软件来使用。在将应用程序连接起来的过程中，一个更为棘手的问题是在两个应用程序之间进行协议接口的协商，以便这些
205 应用程序可以进行数据的交换——这通常都是针对具体应用程序的个案进行的。

应用程序之间要配合工作，所要交互的信息比 HTTP 首部所能表达的信息要复杂得多。第 19 章描述了几个用于交换定制信息的扩展 HTTP 或 HTTP 上层协议实例。19.1 节介绍的是在 HTTP POST 报文之上建立 RPC 层，19.2 节介绍的是向 HTTP 首部添加 XML 的问题。

因特网委员会开发了一组允许 Web 应用程序之间相互通信的标准和协议。尽管 Web 服务（Web service）可以用来表示独立的 Web 应用程序（构造模块），这里我们还是宽松地用这个术语来表示这些标准。Web 服务的引入并不新鲜，但这是应用程序共享信息的一种新机制。Web 服务是构建在标准的 Web 技术（比如 HTTP）之上的。

Web 服务可以用 XML 通过 SOAP 来交换信息。XML（Extensible Markup Language，扩展标记语言）提供了一种创建数据对象的定制信息，并对其进行解释的方法。SOAP（Simple Object Access Protocol，简单对象访问协议）是向 HTTP 报文中添加

XML 信息的标准方式。[3]

8.5 隧道

我们已经讨论了几种不同的方式，通过这些方式可以用 HTTP 对不同类型的资源进行访问（通过网关），或者是用 HTTP 来启动应用程序到应用程序的通信。在本节中，我们要看看 HTTP 的另一种用法——Web 隧道（Web tunnel），这种方式可以通过 HTTP 应用程序访问使用非 HTTP 协议的应用程序。

Web 隧道允许用户通过 HTTP 连接发送非 HTTP 流量，这样就可以在 HTTP 上捎带其他协议数据了。使用 Web 隧道最常见的原因就是要在 HTTP 连接中嵌入非 HTTP 流量，这样，这类流量就可以穿过只允许 Web 流量通过的防火墙了。

8.5.1 用CONNECT建立HTTP隧道

Web 隧道是用 HTTP 的 CONNECT 方法建立起来的。CONNECT 方法并不是 HTTP/1.1 核心规范的一部分，[4]但却是一种得到广泛应用的扩展。可以在 Ari Luotonen 的过期因特网草案规范“Tunneling TCP based protocols through Web proxy servers”（“通过 Web 代理服务器用隧道方式传输基于 TCP 的协议”），或他的著作 *Web Proxy Servers* 中找到这些技术规范，本章末尾引用了这两份资源。 206

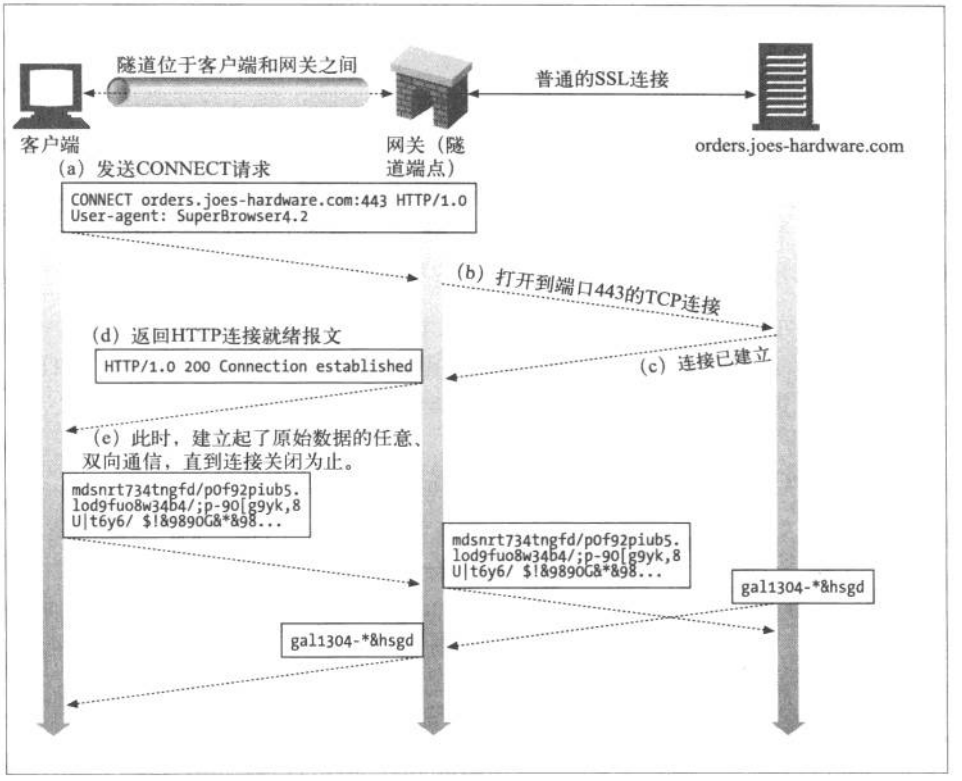
CONNECT 方法请求隧道网关创建一条到达任意目的服务器和端口的 TCP 连接，并对客户端和服务器之间的后继数据进行盲转发。

图 8-10 显示了 CONNECT 方法如何建立起一条到达网关的隧道。

- 在图 8-10a 中，客户端发送了一条 CONNECT 请求给隧道网关。客户端的 CONNECT 方法请求隧道网关打开一条 TCP 连接（在这里，打开的是到主机 orders.joes-hardware.com 的标准 SSL 端口 443 的连接）。
- 在图 8-10b 和图 8-10c 中创建了 TCP 连接。
- 一旦建立了 TCP 连接，网关就会发送一条 HTTP 200 Connection Established 响应来通知客户端（参见图 8-10d）。
- 此时，隧道就建立起来了。客户端通过 HTTP 隧道发送的所有数据都会被直接转发给输出 TCP 连接，服务器发送的所有数据都会通过 HTTP 隧道转发给客户端。

注 3：更多信息，请参见 http://www.w3.org/TR/2001/WD-soap12-part0-20011217/。Doug Tidwell、James Snell 和 Pavel Kulchenko 编写的 *Programming Web Services with SOAP*（SOAP Web 服务开发）一书（O'Reilly）也是非常好的 SOAP 协议信息资源。

注 4：HTTP/1.1 规范保留了 CONNECT 方法，但没有对其功能进行描述。



207 图 8-10 用 CONNECT 建立一条 SSL 隧道

图 8-10 中的例子描述了一条 SSL 隧道，其中的 SSL 流量是在一条 HTTP 连接上发送的，但是通过 CONNECT 方法可以与使用任意协议的任意服务器建立 TCP 连接的。

1. CONNECT请求

除了起始行之外，CONNECT 的语法与其他 HTTP 方法类似。一个后面跟着冒号和端口号的主机名取代了请求 URI。主机和端口都必须指定：

```
CONNECT home.netscape.com:443 HTTP/1.0
User-agent: Mozilla/4.0
```

和其他 HTTP 报文一样，起始行之后，有零个或多个 HTTP 请求首部字段。这些行照例以 CRLF 结尾，首部列表以一个空的 CRLF 结束。

2. CONNECT响应

发送了请求之后，客户端会等待来自网关的响应。和普通 HTTP 报文一样，响应码 200 表示成功。按照惯例，响应中的原因短语通常被设置为“Connection Established”：

```
HTTP/1.0 200 Connection Established
Proxy-agent: Netscape-Proxy/1.1
```

与普通 HTTP 响应不同，这个响应并不需要包含 Content-Type 首部。此时连接只是对原始字节进行转接，不再是报文的承载者，所以不需要使用内容类型了。[5]

8.5.2 数据隧道、定时及连接管理

管道化数据对网关是不透明的，所以网关不能对分组的顺序和分组流作任何假设。一旦隧道建立起来了，数据就可以在任意时间流向任意方向了。[6]

作为一种性能优化方法，允许客户端在发送了 CONNECT 请求之后，接收响应之前，发送隧道数据。这样可以更快地将数据发送给服务器，但这就意味着网关必须能够正确处理跟在请求之后的数据。尤其是，网关不能假设网络 I/O 请求只会返回首部数据，网关必须确保在连接准备就绪时，将与首部一同读进来的数据发送给服务器。在请求之后以管道方式发送数据的客户端，如果发现回送的响应是认证请求，或者其他非 200 但不致命的错误状态，就必须做好重发请求数据的准备。[7]

如果在任意时刻，隧道的任意一个端点断开了连接，那个端点发出的所有未传输数据都会被传送给另一个端点，之后，到另一个端点的连接也会被代理终止。如果还有数据要传输给关闭连接的端点，数据会被丢弃。

8.5.3 SSL隧道

最初开发 Web 隧道是为了通过防火墙来传输加密的 SSL 流量。很多组织都会将所有流量通过分组过滤路由器和代理服务器以隧道方式传输，以提升安全性。但有些协议，比如加密 SSL，其信息是加密的，无法通过传统的代理服务器转发。隧道会通过一条 HTTP 连接来传输 SSL 流量，以穿过端口 80 的 HTTP 防火墙（参见图 8-11）。

注 5：为了实现一致性，今后的规范可能会为隧道定义一个媒体类型（比如 application/tunnel）。

注 6：隧道的两端（客户端和网关）必须做好在任意时刻接收来自连接任一端分组的准备，而且必须将数据立即转发出去。由于隧道化协议中可能包含了数据的依赖关系，所以隧道的任一端都不能忽略输入数据。隧道一端对数据的消耗不足可能会将隧道另一端的数据生产者挂起，造成死锁。

注 7：传送的数据不要超过请求 TCP 分组的剩余容量。如果在收到所有管道化传输的 TCP 分组之前，网关关闭了连接，那么，管道化传输的多余数据就会使客户端 TCP 重置。TCP 重置会使客户端丢失收到的网关响应，这样客户端就无法分辨错误是由于网络错误、访问控制，还是认证请求造成的了。

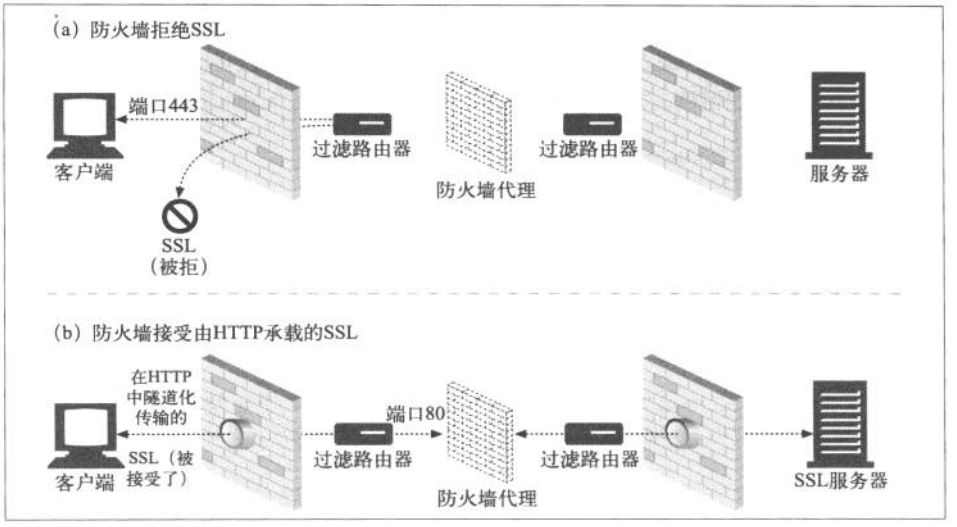


图 8-11　隧道可以经由 HTTP 连接传输非 HTTP 流量

为了让 SSL 流量经现存的代理防火墙进行传输，HTTP 中添加了一项隧道特性，在此特性中，可以将原始的加密数据放在 HTTP 报文中，通过普通的 HTTP 信道传送（参见图 8-12）。

209

在图 8-12a 中，SSL 流量被直接发送给了一个（SSL 端口 443 上的）安全 Web 服务器。在图 8-12b 中，SSL 流量被封装到一条 HTTP 报文中，并通过 HTTP 端口 80 上的连接发送，最后被解封装为普通的 SSL 连接。

通常会用隧道将非 HTTP 流量传过端口过滤防火墙。这一点可以得到很好的利用，比如，通过防火墙传输安全 SSL 流量。但是，这项特性可能会被滥用，使得恶意协议通过 HTTP 隧道流入某个组织内部。

8.5.4　SSL隧道与HTTP/HTTPS网关的对比

可以像其他协议一样，对 HTTPS 协议（SSL 上的 HTTP）进行网关操作：由网关（而不是客户端）初始化与远端 HTTPS 服务器的 SSL 会话，然后代表客户端执行 HTTPS 事务。响应会由代理接收并解密，然后通过（不安全的）HTTP 传送给客户端。这是网关处理 FTP 的方式。但这种方式有几个缺点：

- 客户端到网关之间的连接是普通的非安全 HTTP；
- 尽管代理是已认证主体，但客户端无法对远端服务器执行 SSL 客户端认证（基于 X509 证书的认证）；

- 网关要支持完整的 SSL 实现。

210

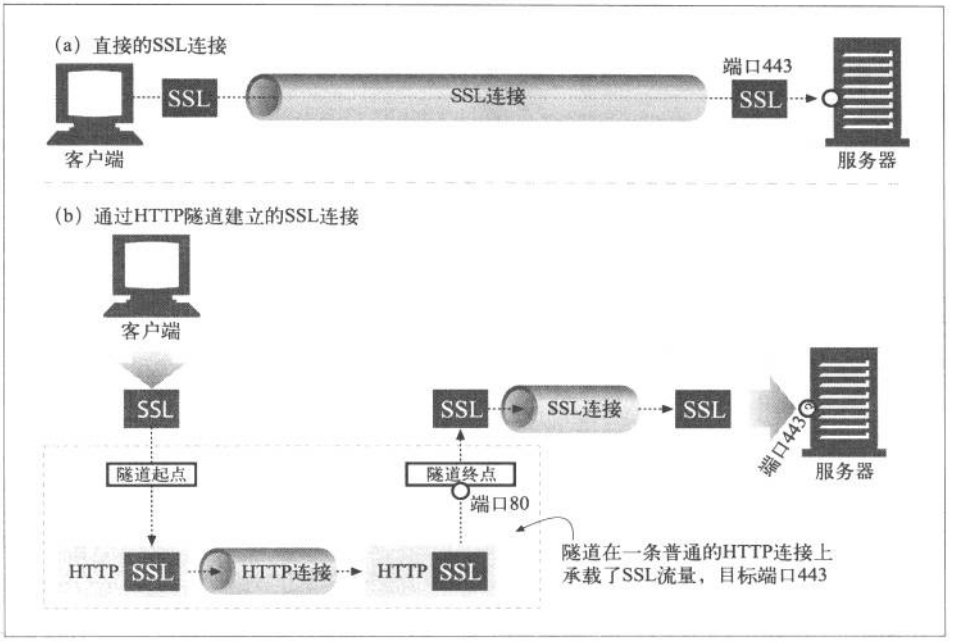


图 8-12　直接的 SSL 连接与隧道化 SSL 连接的对比

注意，对于 SSL 隧道机制来说，无需在代理中实现 SSL。SSL 会话是建立在产生请求的客户端和目的（安全的）Web 服务器之间的，中间的代理服务器只是将加密数据经过隧道传输，并不会在安全事务中扮演其他的角色。

8.5.5　隧道认证

在适当的情况下，也可以将 HTTP 的其他特性与隧道配合使用。尤其是，可以将代理的认证支持与隧道配合使用，对客户端使用隧道的权利进行认证（参见图 8-13）。

8.5.6　隧道的安全性考虑

总的来说，隧道网关无法验证目前使用的协议是否就是它原本打算经过隧道传输的协议。因此，比如说，一些喜欢捣乱的用户可能会通过本打算用于 SSL 的隧道，越过公司防火墙传递因特网游戏流量，而恶意用户可能会用隧道打开 Telnet 会话，或用隧道绕过公司的 E-mail 扫描器来发送 E-mail。

211

为了降低对隧道的滥用，网关应该只为特定的知名端口，比如 HTTPS 的端口 443，打开隧道。

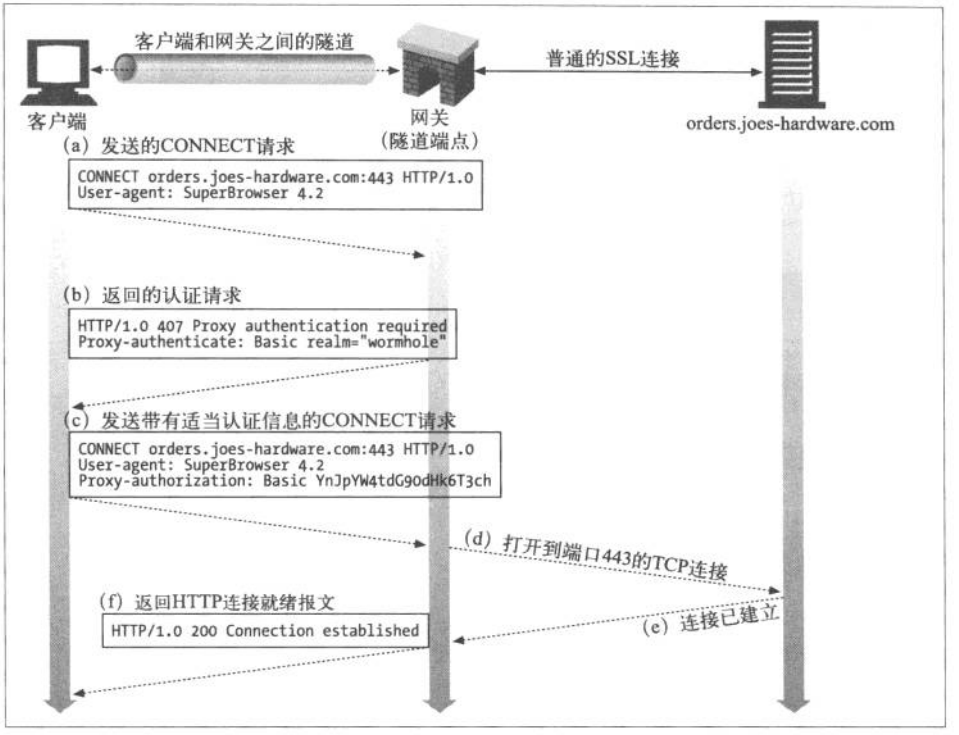


图 8-13　网关允许某客户端使用隧道之前，可以对其进行代理认证

8.6　中继

HTTP 中继（relay）是没有完全遵循 HTTP 规范的简单 HTTP 代理。中继负责处理 HTTP 中建立连接的部分，然后对字节进行盲转发。

HTTP 很复杂，所以实现基本的代理功能并对流量进行盲转发，而且不执行任何首部和方法逻辑，有时是很有用的。盲中继很容易实现，所以有时会提供简单的过滤、诊断或内容转换功能。但这种方式可能潜在严重的互操作问题，所以部署的时候要特别小心。

某些简单盲中继实现中存在的一个更常见（也更声名狼藉的）问题是，由于它们无法正确处理 Connection 首部，所以有潜在的挂起 keep-alive 连接的可能。图 8-14 对这种情况进行了说明。

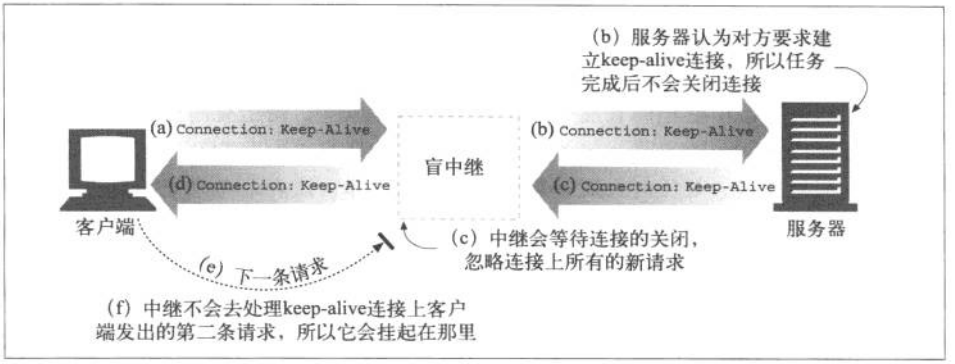


图 8-14　如果简单盲中继是单任务的，且不支持 Connection 首部，就会挂起

这张图中发生的情况如下所述。

- 在图 8-14a 中，Web 客户端向中继发送了一条包含 `Connection: Keep-Alive` 首部的报文，如果可能的话要求建立一条 keep-alive 连接。客户端等待响应，以确定它要求建立 keep-alive 信道的请求是否被认可了。
- 中继收到了这条 HTTP 请求，但它并不理解 `Connection` 首部，因此会将报文一字不漏地沿着链路传递给服务器（参见图 8-14b）。但 `Connection` 首部是个逐跳首部；只适用于单条传输链路，是不应该沿着链路传送下去的。要有不好的事情发生了！ 212
- 在图 8-14b 中，经过中继转发的 HTTP 请求抵达 Web 服务器。当 Web 服务器收到经过代理转发的 `Connection: Keep-Alive` 首部时，会错误地认为中继（对服务器来说，它看起来就和其他客户端一样）要求进行 keep-alive 的对话！这对 Web 服务器来说没什么问题——它同意进行 keep-alive 对话，并在图 8-14c 中回送了一个 `Connection: Keep-Alive` 响应首部。那么，此时，Web 服务器就认为它是在与中继进行 keep-alive 对话，会遵循 keep-alive 对话的规则。但中继对 keep-alive 会话根本就一无所知。
- 在图 8-14d 中，中继将 Web 服务器的响应报文，以及来自 Web 服务器的 `Connection: Keep-Alive` 首部一起发回给客户端。客户端看到这个首部，认为中继同意进行 keep-alive 对话。此时，客户端和服务器都认为它们是在进行 keep-alive 对话，但与它们进行对话的中继却根本不知道什么 keep-alive 对话。
- 中继对持久对话一无所知，所以它会将收到的所有数据都转发给客户端，等待原始服务器关闭连接。但原始服务器认为中继要求服务器将连接保持在活跃状态，所以是不会关闭连接的！这样，中继就会挂起，等待连接的关闭。
- 在图 8-14d 中，当客户端收到回送的响应报文时，它会直接转向第二条请求，在 keep-alive 连接上向中继发送另一条请求（参见图 8-14e）。简单中继通常不会期

待同一条连接上还会有另一条请求到达。浏览器上的圈会不停地转，但没有任何进展。

有一些方法可以使中继稍微智能一些，以消除这些风险，但所有简化的代理都存在着出现互操作性问题的风险。要为某个特定目标构建简单的 HTTP 中继，一定要特别注意其使用方法。对任何大规模部署来说，都要非常认真地考虑使用真正的、完全遵循 HTTP 的代理服务器。

更多与中继和连接管理有关的信息，参见 4.5.6 节。

8.7 更多信息

更多信息，请参见下列参考材料。

- http://www.w3.org/Protocols/rfc2616/rfc2616.txt
  RFC 2616，由 R. Fielding、J. Gettys、J. Mogul、H. Frystyk、L. Mastinter、P. Leach 和 T. Berners-Lee 编写的“Hypertext Transfer Protocol”。

- *Web Proxy Servers*（《Web 代理服务器》）
213 Ari Luotonen，Prentice Hall 出版的计算机图书。

- http://www.alternic.org/drafts/drafts-l-m/draft-luotonen-Web-proxy-tunneling-01.txt
  Ari Luotonen 编写的“Tunneling TCP based protocols through Web proxy servers”（“用隧道方式通过 Web 代理服务器传输基于 TCP 的协议”）。

- http://cgi-spec.golux.com
  通用网关接口——RFC 项目页面。

- http://www.w3.org/TR/2001/WD-soap12-part0-20011217/
  W3C——SOAP 版本 1.2 工作草案。

- *Programming Web Services with Soap*[8]（《Soap Web 服务开发》）
  James Snell、Doug Tidwell 和 Pavel Kulchenko 编写，O'Reilly & Associates 公司出版。

- http://www.w3.org/TR/2002/WD-wsa-reqs-20020429
  W3C——Web 服务的架构要求。

- *Web Services Essentials*[9]（《Web 服务精髓》）
214 Ethan Cerami，O'Reilly & Associates 公司出版。

注 8-9：这二本书的中文版已由中国电力出版社出版。（编者注）