写在前面：

2019 年寒假，大年初一到初十的时间看了这本书的将近一半内容。具体是第 1-5 章，第 11-14 章。我看的这第一部分主要是先概述了一下 HTTP 是什么，然后讲解 URL 的结构，接着解析 HTTP 报文，最后说了 HTTP 链接的管理以及 web 服务器那边做的一些特殊的处理。第二部分主要就是认证机制和 HTTPs

一、HTTP 概述：

1. 所有能提供 web 内容的东西都是 web 资源，包括静态文件也包括根据需要生成内容的软件程序、搜索引擎等。
2. Web 服务器会为所有 HTTP 对象数据附加一个 MIME 类型：

   MIME 类型是一种文本标记，表示一种主要的对象类型和一个特定的子类型，中间由一条斜杠来分隔。

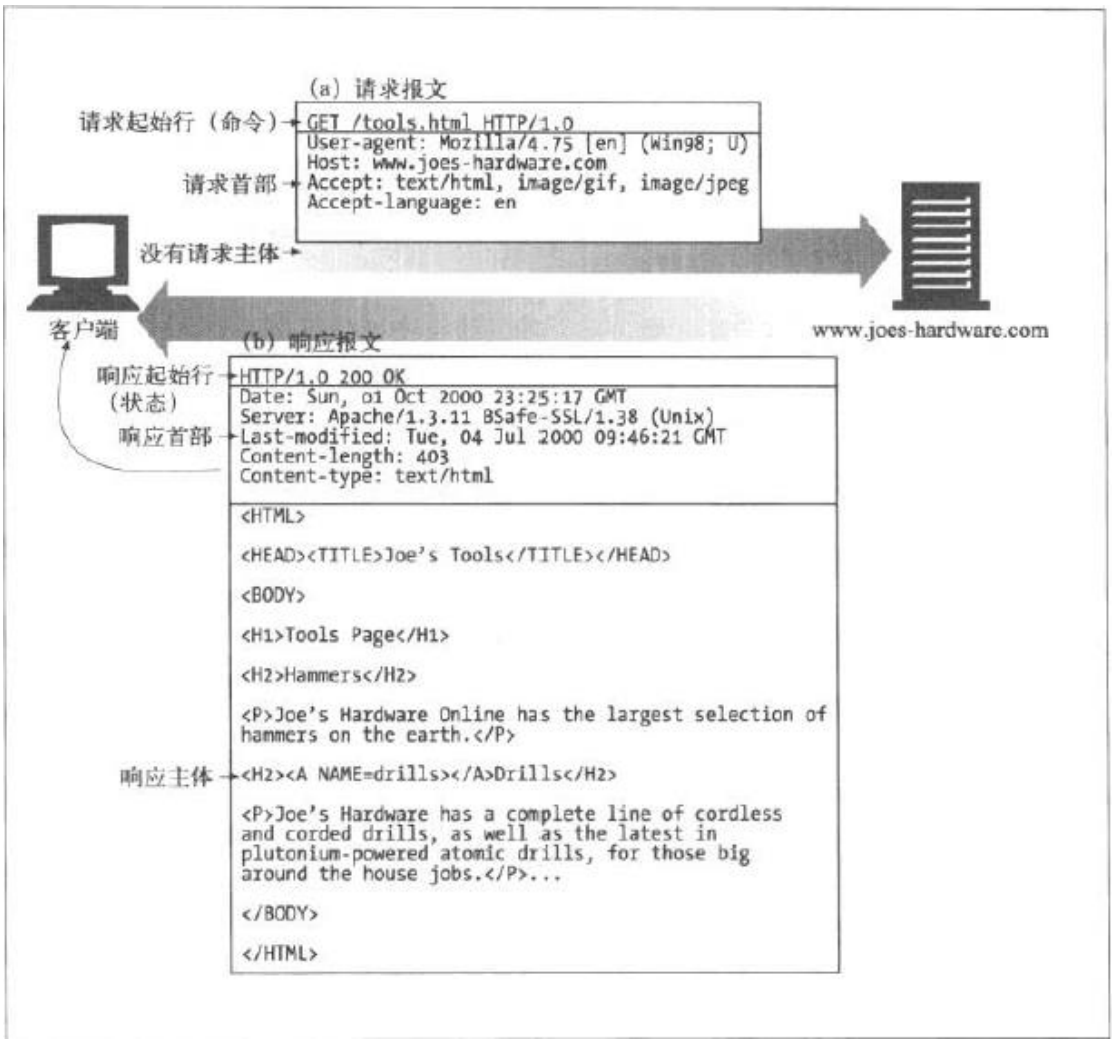
   - HTML 格式的文本文档由 text/html 类型来标记。
   - 普通的 ASCII 文本文档由 text/plain 类型来标记。
   - JPEG 版本的图片为 image/jpeg 类型。
   - GIF 格式的图片为 image/gif 类型。
   - Apple 的 QuickTime 电影为 video/quicktime 类型。
   - 微软的 PowerPoint 演示文件为 application/vnd.ms-powerpoint 类型。

3. URI、URL 与 URN：
   a) URI：统一资源标识符
   b) URL：统一资源定位符
   c) URN：统一资源名
4. 常见的 HTTP 方法（后文会深入探究）：

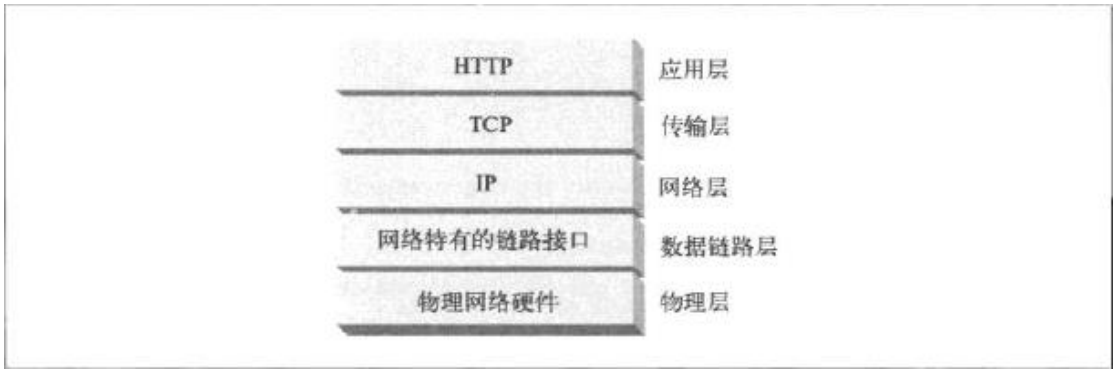
   表1-2 一些常见的HTTP方法

HTTP方法	描　述
GET	从服务器向客户端发送命名资源
PUT	将来自客户端的数据存储到一个命名的服务器资源中去
DELETE	从服务器中删除命名资源
POST	将客户端数据发送到一个服务器网关应用程序
HEAD	仅发送命名资源响应中的 HTTP 首部

5. HTTP 状态码（后面会深入探究）
6. HTTP 报文的结构：
   起始行、首部、主体（返回内容）



7. TCP/IP：
   a) HTTP 是一个应用层协议，局域 TCP 的上方。它把联网的细节都交给了通用、可靠的因特网传输协议 TCP/IP
   b) TCP 提供无差错的数据传输、按序传输、未分段的数据流
   c) 四层模型的结构：



   d) 浏览器是如何通过 HTTP 显示位于远端服务器中的某个简单 HTML 资源的：

(a) 浏览器从 URL 中解析出服务器的主机名；

(b) 浏览器将服务器的主机名转换成服务器的 IP 地址；

(c) 浏览器将端口号（如果有的话）从 URL 中解析出来；

(d) 浏览器建立一条与 Web 服务器的 TCP 连接；

(e) 浏览器向服务器发送一条 HTTP 请求报文；

(f) 服务器向浏览器回送一条 HTTP 响应报文；

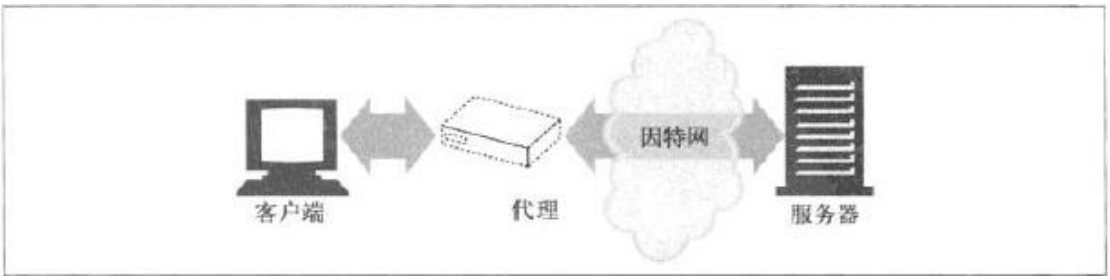
(g) 关闭连接，浏览器显示文档。

8. Telnet:

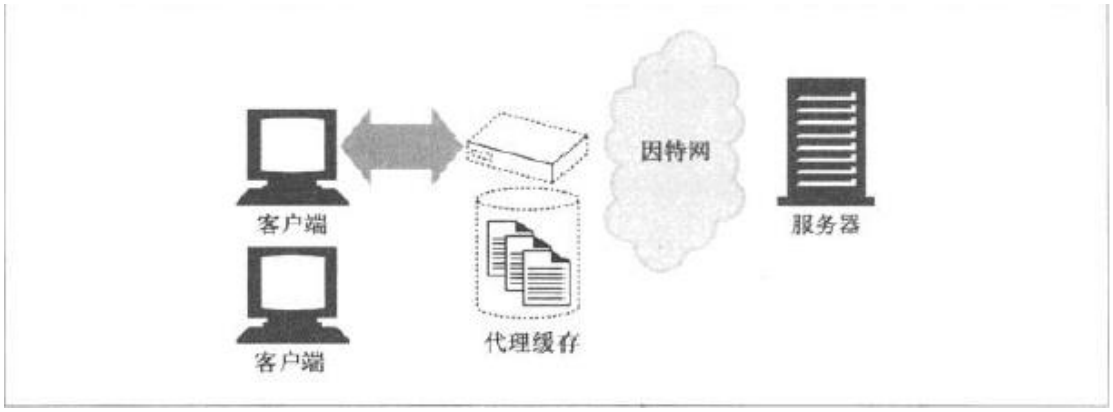
   Telnet 可以直接连接到某个目标 TCP 端口，并将输出回送到显示屏上。(其实就是在模拟 HTTP 客户端)

9. 代理、缓存和网关:

   a) 代理:

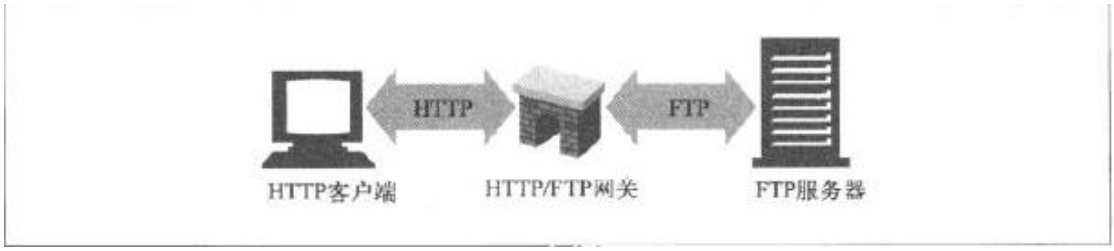


   b) 缓存:



   c) 网关:

   网关是用来转换协议的:



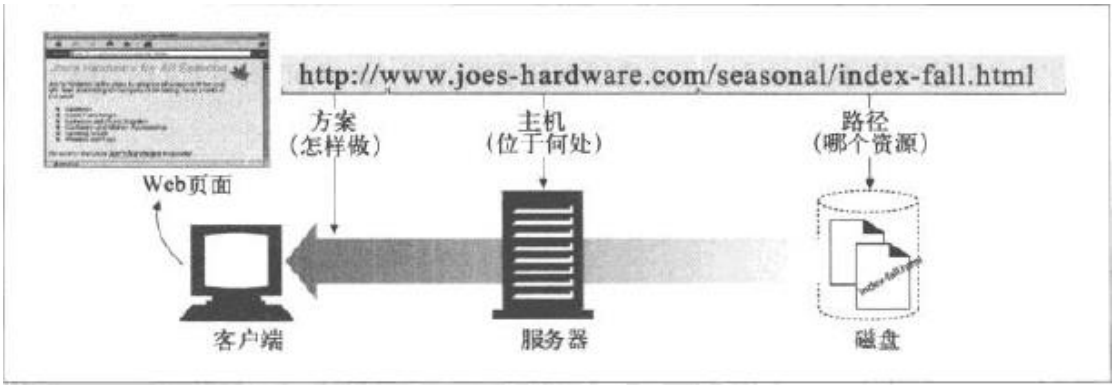
   d)

10.

二、URL 与资源

1. URL 的 3 部分：

   URL 方案、主机、路径

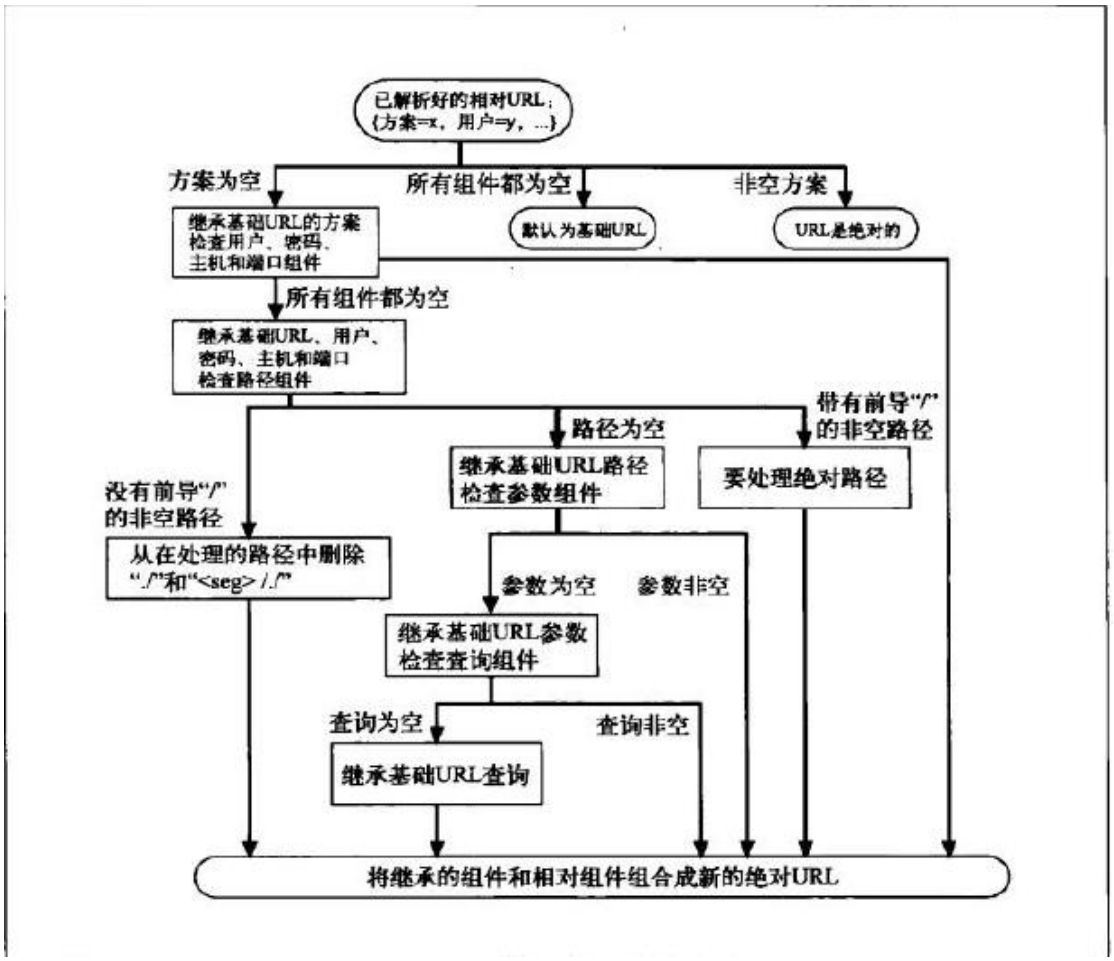


   完整的 URL 语法包括 9 个组件：

```
<scheme>://<user>:<password>@<host>:<port>/<path>;<params>?<query>#<frag>
```

2. URL 的快捷方式：

   a) 相对 URL 转绝对 URL 算法：



   举个例子：

(1) 路径为 ./hammers.html，基础 URL 为 http://www.joes-hardware.com/tools.html。
(2) 方案为空，沿着图表的左半边向下处理，继承基础 URL 方案（HTTP）。
(3) 至少一个组件非空，一直处理到底端，继承主机和端口组件。
(4) 将来自相对 URL（路径：./hammers.html）的组件与我们继承来的组件（方案：http，主机：www.joes-hardware.com，端口：80）合并起来，得到新的绝对 URL：http ://www.joes-hardware.com/hammers.html。

三、HTTP 报文：

1. 报文流：
   报文都是向下游流动的。只是请求报文和响应报文的下游对象不同
2. 报文的组成部分：
   前面已经说过，是起始行、首部、主体三个部分。

a) 起始行里的方法类型：

方　法	描　述	是否包含主体
GET	从服务器获取一份文档	否
HEAD	只从服务器获取文档的首部	否
POST	向服务器发送需要处理的数据	是
PUT	将请求的主体部分存储在服务器上	是
TRACE	对可能经过代理服务器传送到服务器上去的报文进行追踪	否
OPTIONS	决定可以在服务器上执行哪些方法	否
DELETE	从服务器上删除一份文档	否

b) 起始行的状态码：

整体范围	已定义范围	分　类
100 ~ 199	100 ~ 101	信息提示
200 ~ 299	200 ~ 206	成功
300 ~ 399	300 ~ 305	重定向
400 ~ 499	400 ~ 415	客户端错误
500 ~ 599	500 ~ 505	服务器错误

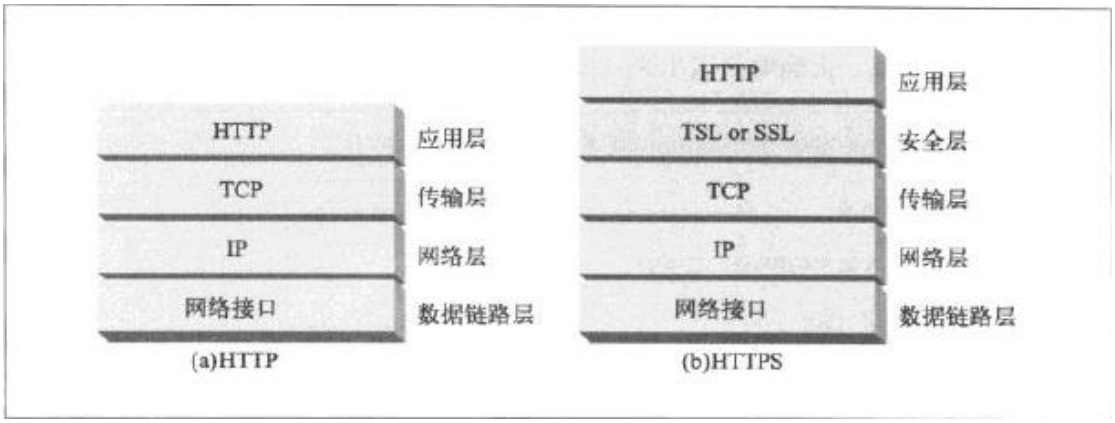
常见的几个状态码：
200 OK 表示成功请求
401 unauthorized 表示未授权，需要输入用户名和密码
404 Not Found 表示服务器无法找到所请求 URL 对应的资源

c) 报文的首部：
HTTP 首部可以分为以下几类：
通用首部、请求首部、相应首部、实体首部、扩展首部。
常见的首部：

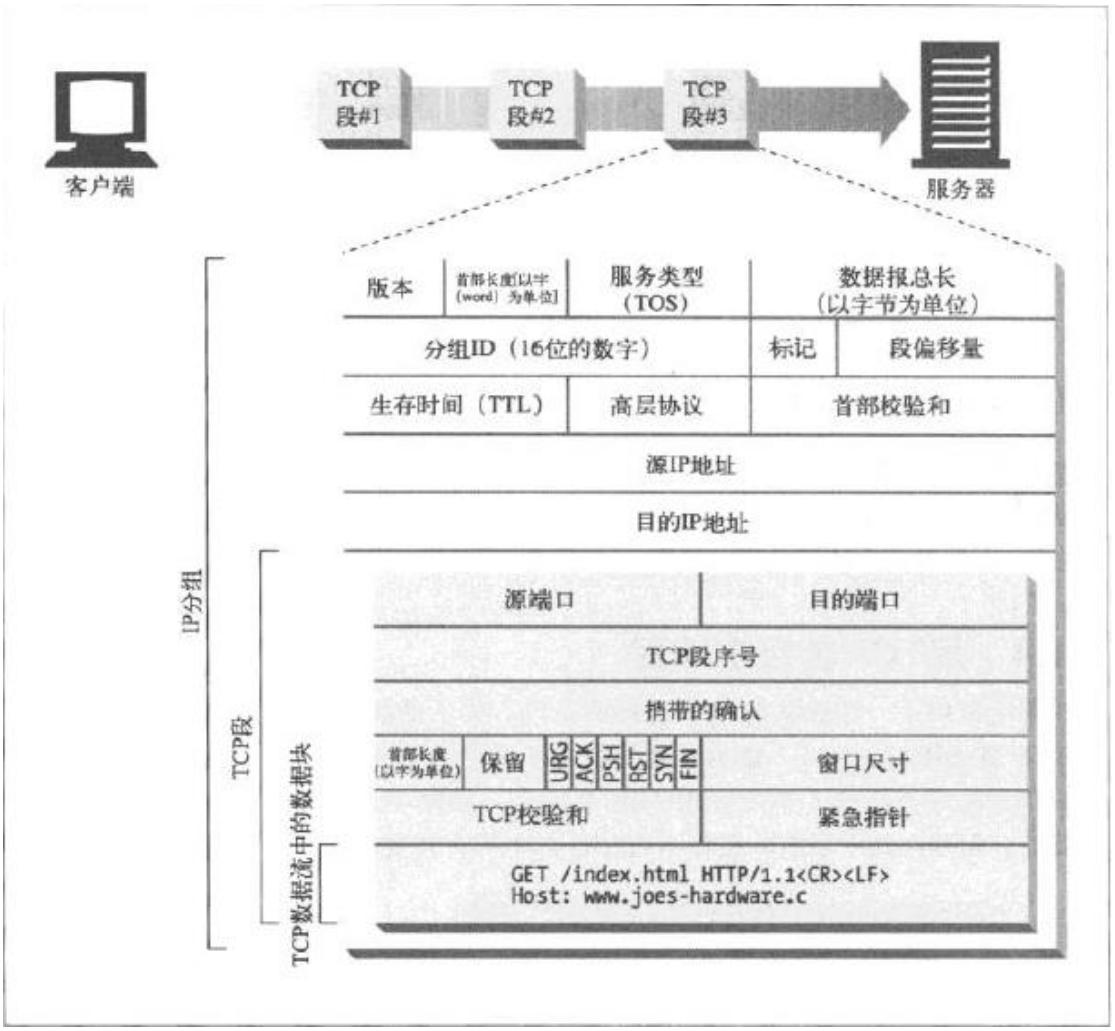
首部实例	描　述
Date:Tue,3Oct 1997 02:16:03 GMT	服务器产生响应的日期
Content-length:15040	实体的主体部分包含了 15 040 字节的数据
Content-type:image/gif	实体的主体部分是一个 GIF 图片
Accept: image/gif, image/jpeg, text/html	客户端可以接收 GIF 图片和 JPEG 图片以及 HTML

四、连接管理

1. TCP 连接：
   a) TCP 流是分段的、由 IP 分组传送
   b) HTTP 和 HTTPs 网络协议栈：



   c) TCP 报文的具体内容可以看本科网络课的 PPT。大概是下图这样：

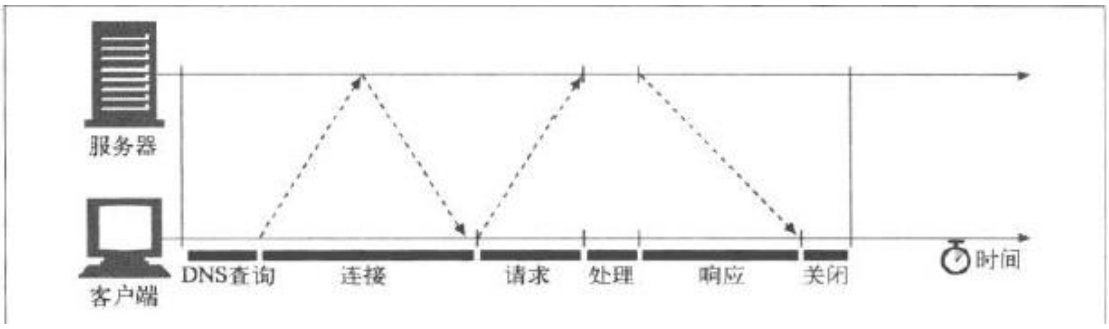


   d) Java 的 socket 编程就是在 TCP 层面上编程。具体可以参见自己的 Java 基础的笔记和敲的 demo 代码

2. 对 TCP 性能的考虑：
   a) HTTP 事务的时延原因：

i. DNS 解析的时延
ii. TCP 连接的时延
iii. 因特网传输的时间
iv. Web 服务器回送 HTTP 响应的时间

如图所示:



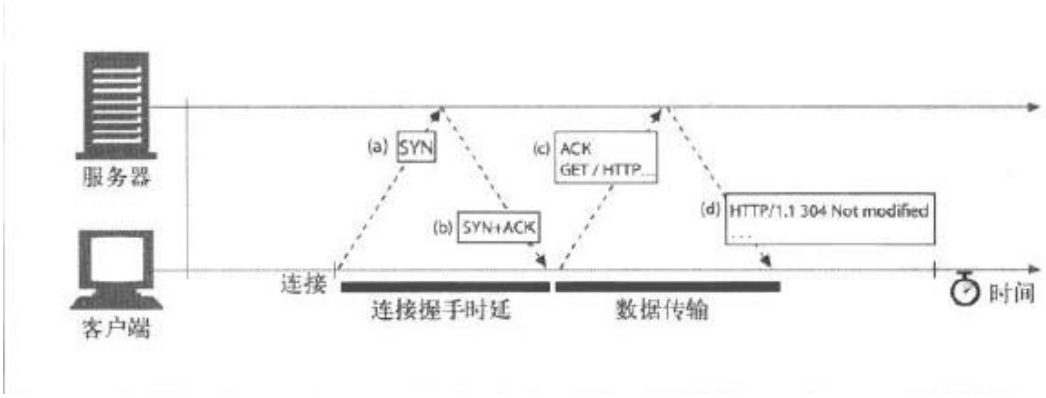
b) 性能聚焦区域是在 TCP 相关时延，其中包括:
i. TCP 连接建立握手
1. 3 次握手,步骤和示意图:

TCP 连接握手需要经过以下几个步骤。

(1) 请求新的 TCP 连接时，客户端要向服务器发送一个小的 TCP 分组（通常是 40
60 个字节）。这个分组中设置了一个特殊的 SYN 标记，说明这是一个连接请求
（参见图 4-8a）。

(2) 如果服务器接受了连接，就会对一些连接参数进行计算，并向客户端回送一
TCP 分组，这个分组中的 SYN 和 ACK 标记都被置位，说明连接请求已被接
（参见图 4-8b）。

(3) 最后，客户端向服务器回送一条确认信息，通知它连接已成功建立（参见图 4-8c
现代的 TCP 栈都允许客户端在这个确认分组中发送数据。



ii. TCP 慢启动拥塞控制

4.2.5 TCP慢启动

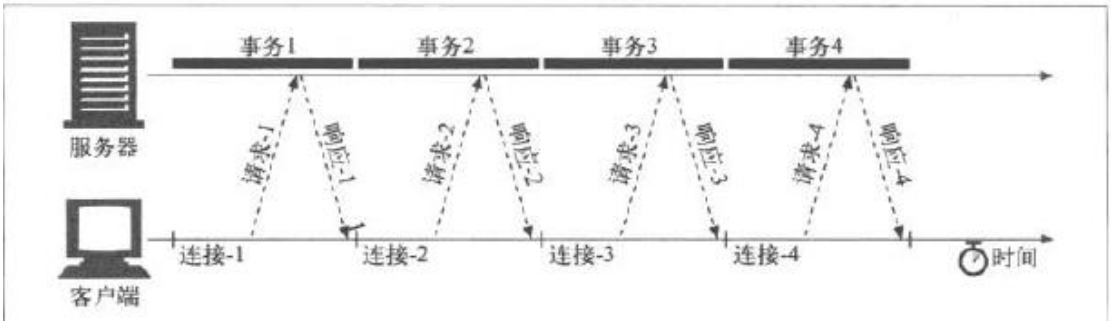
TCP 数据传输的性能还取决于 TCP 连接的使用期（age）。TCP 连接会随着时间进行自我“调谐”，起初会限制连接的最大速度，如果数据成功传输，会随着时间的推移提高传输的速度。这种调谐被称为 TCP 慢启动（slow start），用于防止因特网的突然过载和拥塞。

TCP 慢启动限制了一个 TCP 端点在任意时刻可以传输的分组数。简单来说，每成功接收一个分组，发送端就有了发送另外两个分组的权限。如果某个 HTTP 事务有大量数据要发送，是不能一次将所有分组都发送出去的。必须发送一个分组，等待确认；然后可以发送两个分组，每个分组都必须被确认，这样就可以发送四个分组了，以此类推。这种方式被称为“打开拥塞窗口”。

由于存在这种拥塞控制特性，所以新连接的传输速度会比已经交换过一定量数据的、“已调谐”连接慢一些。由于已调谐连接要更快一些，所以 HTTP 中有一些可以重用现存连接的工具。本章稍后会介绍这些 HTTP“持久连接”。

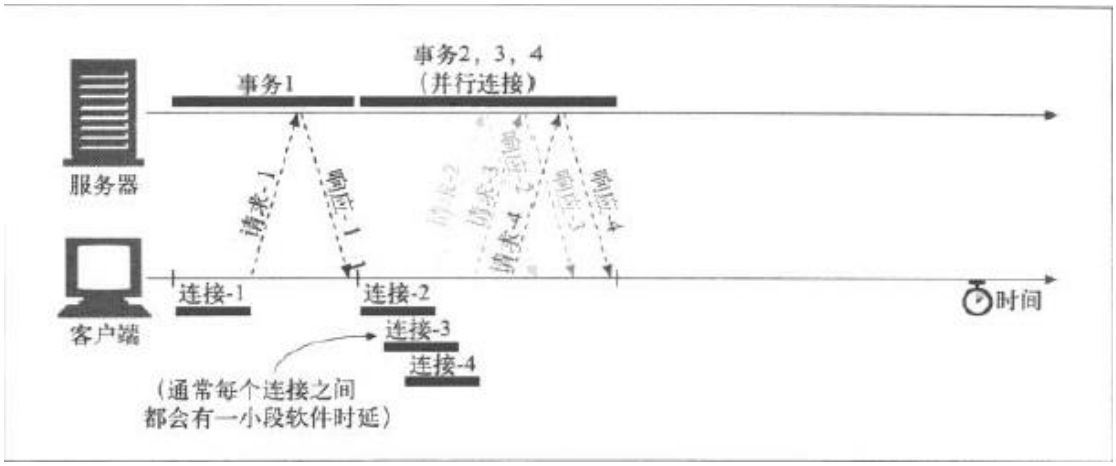
iii. 数据聚集的 Nagle 算法
iv. 用于捎带确认的 TCP 延迟确认算法
v. TIME_WAIT 时延和端口耗尽

3. HTTP 连接的处理：
如果只对连接进行简单的管理，TCP 的性能时延可能会叠加起来。这就是串行事务处理时延，例如：



那么解决这个问题的方法有以下几种：
并行连接：通过多条 TCP 连接发起并发的 HTTP 请求
持久连接：重用 TCP 连接，以消除连接及关闭时延
管道化连接：通过共享的 TCP 连接发起的 HTTP 请求

a) 并行连接：
并行连接可能会提高页面的加载速度，像这样：



但是很显然并行连接不一定能更快：

一个连接就有可能耗尽所有可用的 modem 带宽了。如果并行加载多个对象，每个对象都会去竞争这有限的带宽，每个对象都会以较慢的速度按比例加载，这样带来的性能提升就很小，甚至没什么提升。

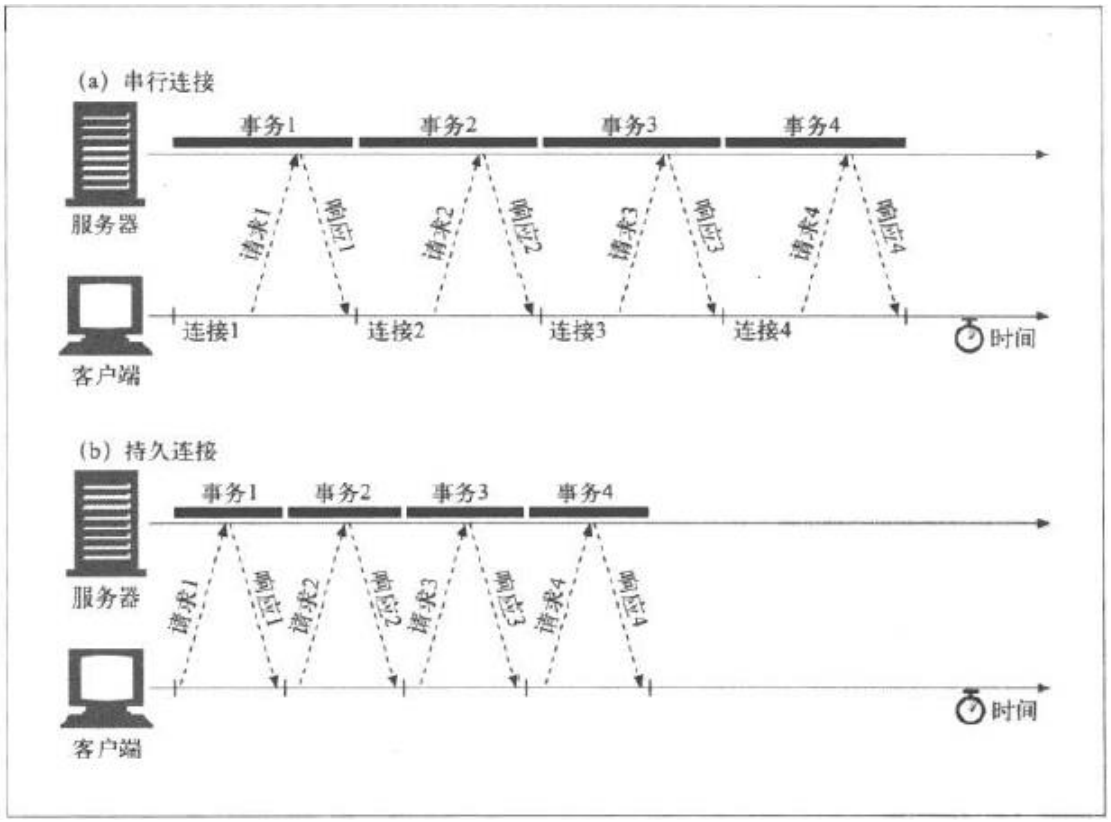
更要命的是，打开大量连接会消耗很多内存资源，从而引发自身的性能问题。

但是，有趣的是，同时加载多个连接会使人“感觉”好像更快一些。毕竟是有反馈的嘛，比一直空白，等着等着，忽然全部跳出来要好。

b) 持久连接：

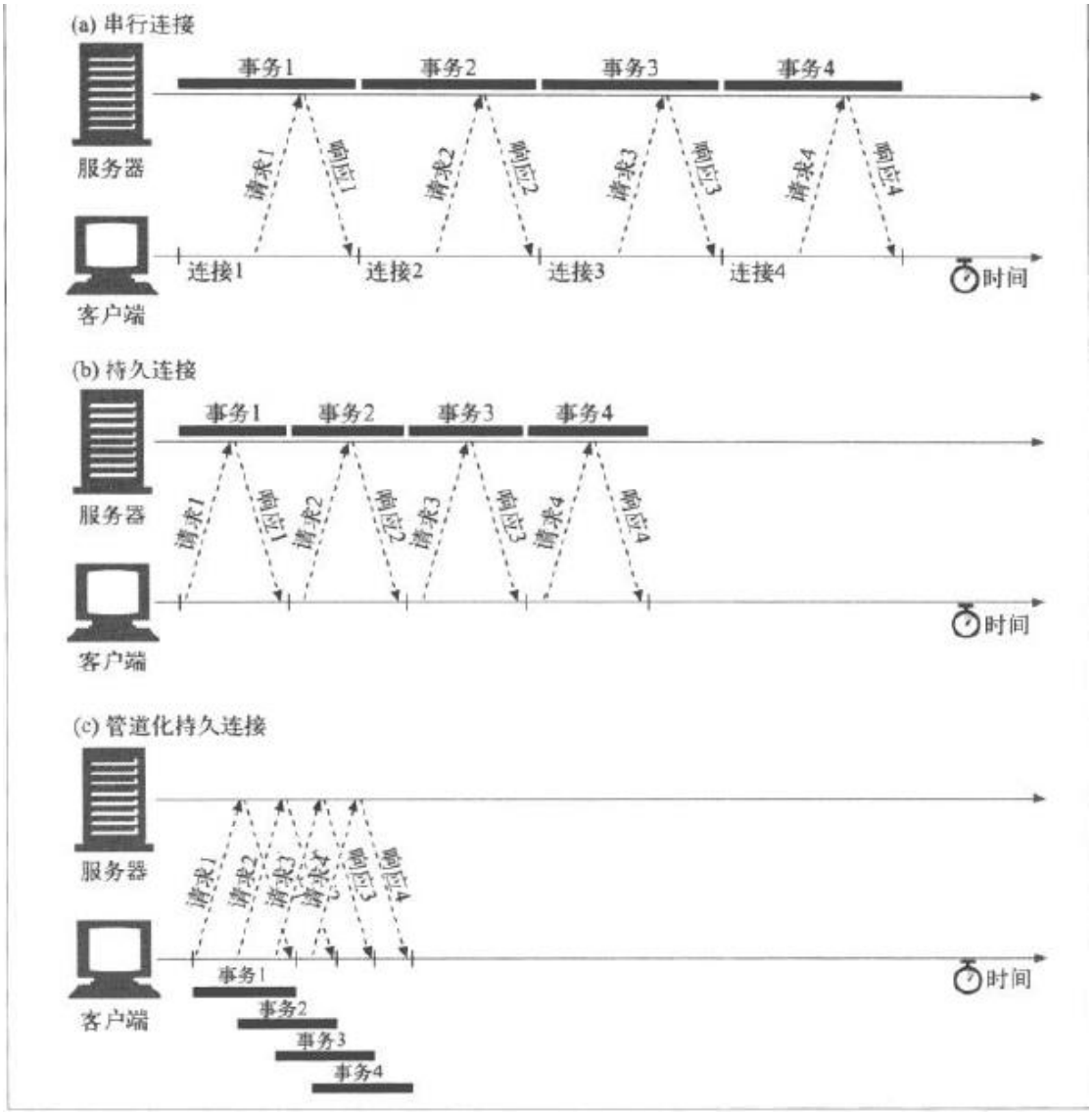
持久连接降低了时延和连接建立的开销，将连接保存在一调谐状态，而且减少了打开连接的潜在数量。但是，管理持久连接时要特别小心，不然就会累积出大量的空闲连接，耗费本地客户端和服务器上的资源。

串行和持久连接的对比：



这里 keep-alive 连接的时候会出现一个哑代理的问题，搞的不是很懂。

c) 管道连接：



五、Web 服务器

1. 一般的商业服务器做了什么：
   a) 建立连接
   b) 接受请求
   c) 处理请求
   d) 访问资源
   e) 构建响应
   f) 发送响应
   g) 记录事务处理过程

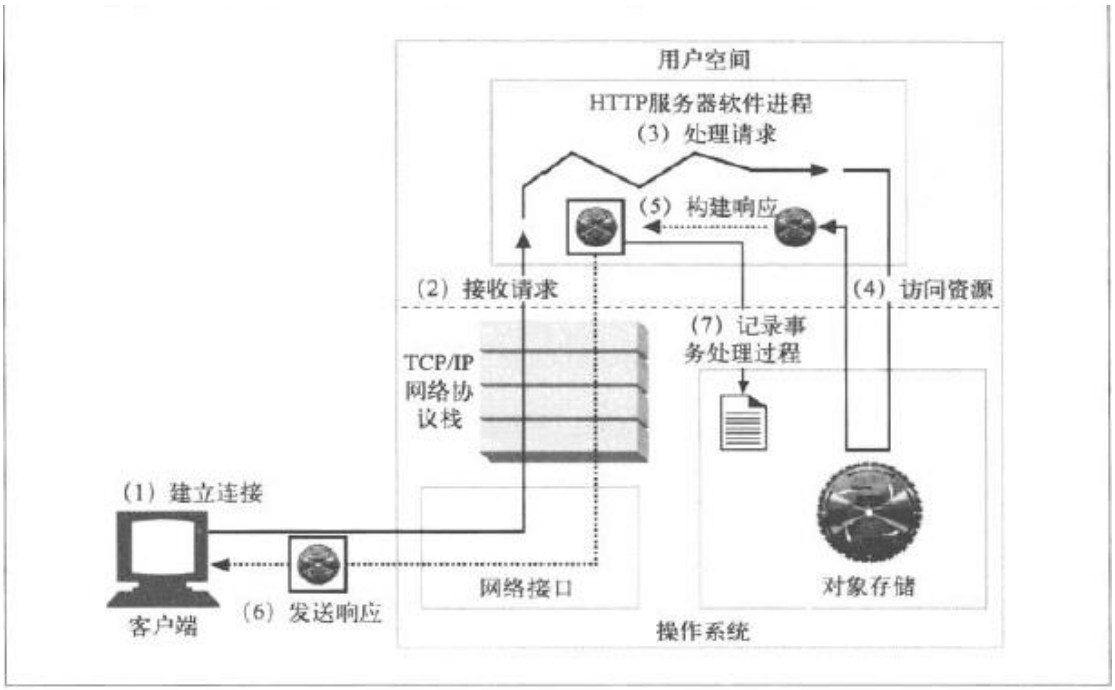


图 5-3 基本 Web 服务器请求的步骤

六、客户端识别与 Cookie 机制

1. HTTP 本身是一个匿名的、无状态的请求/响应协议。无法做记录。这本书里给出了几种用户识别机制：

   承载用户身份信息的 HTTP 首部

   客户端 IP 地址跟踪，通过用户的 IP 地址对其进行识别

   用户登录，用认证方式来识别用户

   胖 URL，一种在 URL 中嵌入识别信息的技术

2. 上述几个方法都很挫。。我感觉现在已经都见不到了。所以书里着重说了 cookie 机制。Cookie 可以大概分为会话 cookie 和持久 cookie。前者是一种临时 cookie，它记录了用户访问站点是的设置和偏好。用户退出浏览器的时候，它就被删除了。持久 cookie 是存在硬盘上的。
3. Cookie 工作的机制：

   Cookie 就像服务器给客户端贴的贴纸。

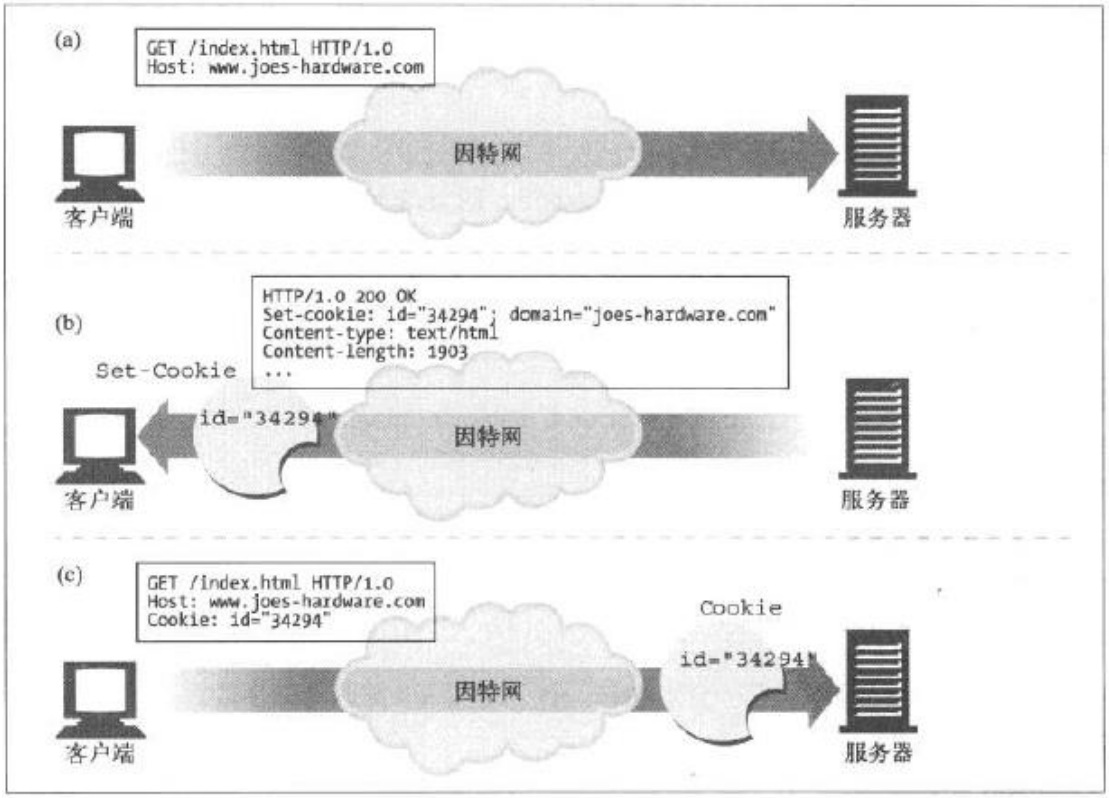


图 11-3　给用户贴一个 cookie

4. 使用 cookie 的流程：

步骤：

- **图 11-5a——浏览器首次请求 Amazon.com 根页面。**
- **图 11-5b——服务器将客户端重定向到一个电子商务软件的 URL 上。**
- **图 11-5c——客户端对重定向的 URL 发起一个请求。**
- **图 11-5d——服务器在响应上贴上两个会话 cookie，并将用户重定向到另一个 URL，这样客户端就会用这些附加的 cookie 再次发出请求。这个新的 URL 是个胖 URL，也就是说有些状态嵌入到 URL 中去了。如果客户端禁止了 cookie，只要用户一直跟随着 Amazon.com 产生的胖 URL 链接，不离开网站，仍然可以实现一些基本的标识功能。**
- **图 11-5e——客户端请求新的 URL，但现在会传送两个附加的 cookie。**
- **图 11-5f——服务器重定向到 home.html 页面，并附加另外两个 cookie。**
- **图 11-5g——客户端获取 home.html 页面并将所有四个 cookie 都发送出去。**
- **图 11-5h——服务器回送内容。**

图解：

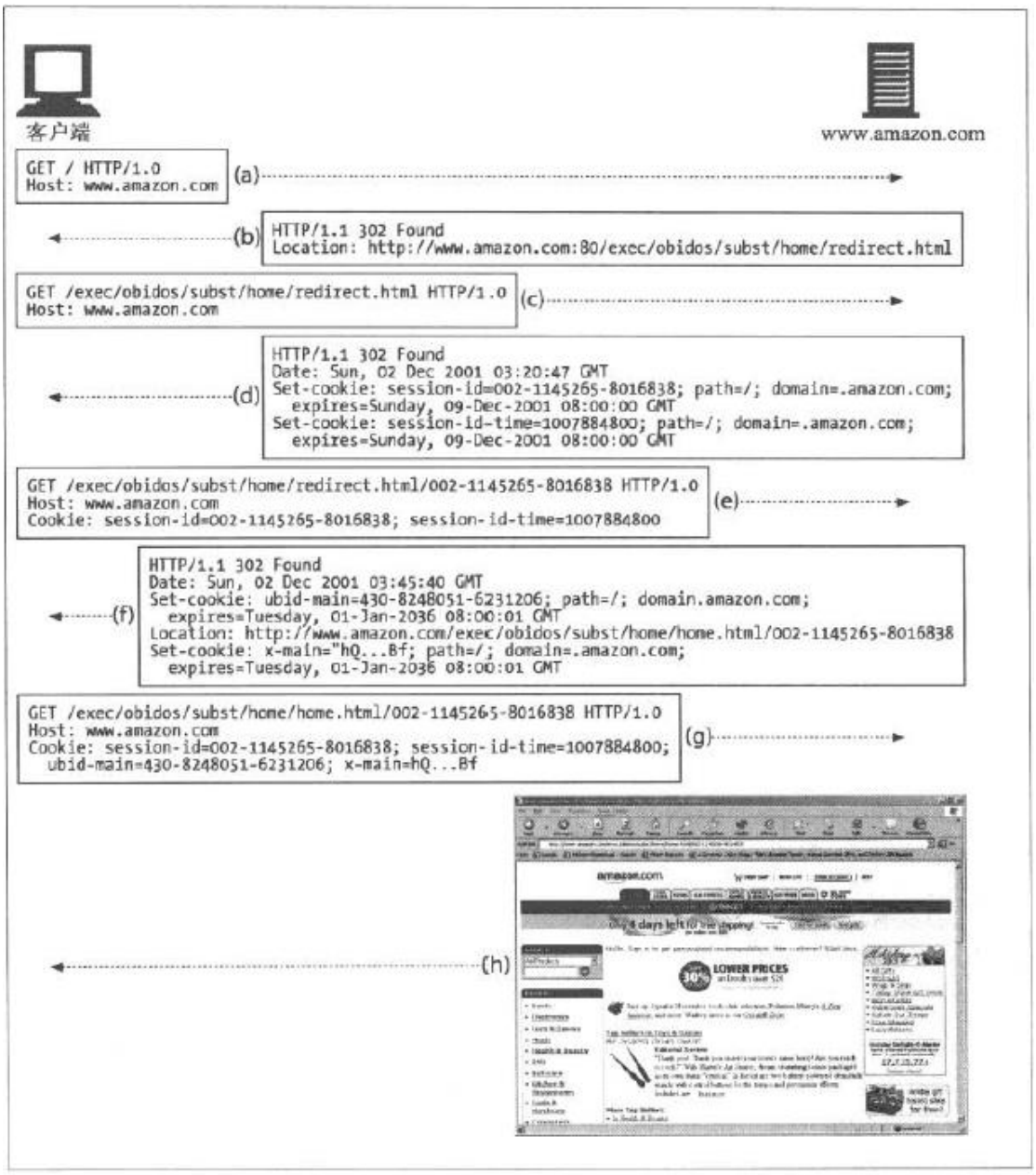
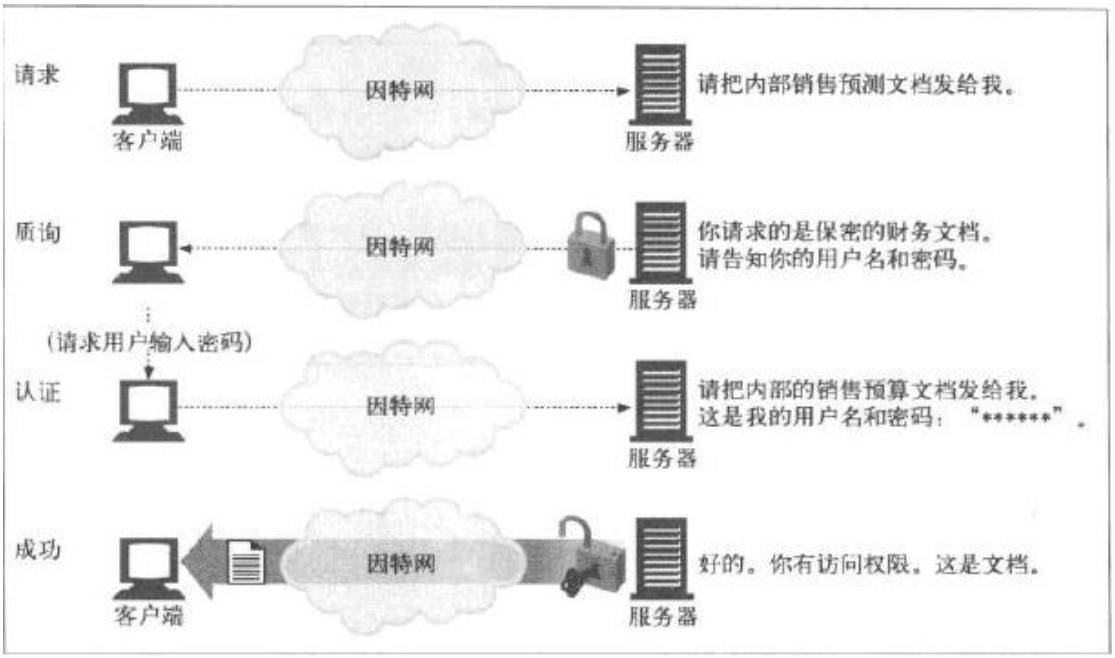


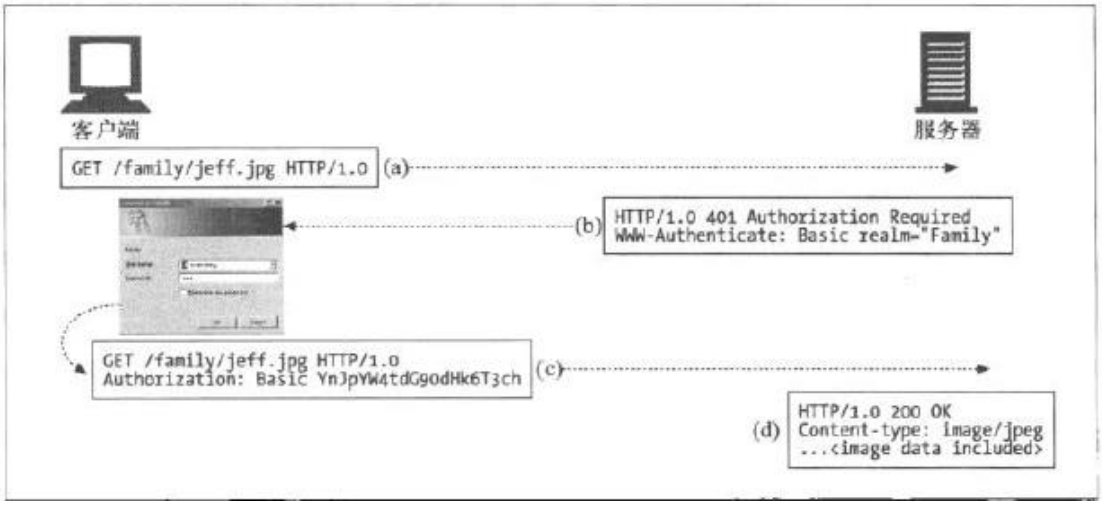
图 11-5　Amazon.com 网站用会话 cookie 来跟踪用户

七、基本认证机制

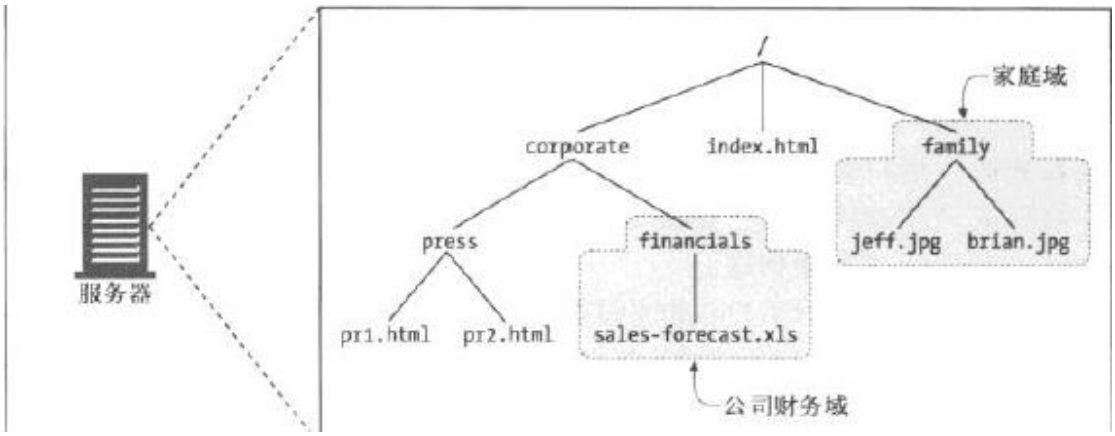
1. HTTP 的质询/响应认证框架：



图解认证步骤：



2. Web 服务器上的额安全域：允许服务器为不同的资源使用不同的访问权限。
   例子:



3. 基本认证的基本机制就是上面那样，下面来看一个例子：

- 在图 12-2a 中，用户请求了私人家庭相片 /family/jeff.jpg。
- 在图 12-2b 中，服务器回送一条 401 Authorization Required，对私人家庭相片进行密码质询，同时回送的还有 WWW-Authenticate 首部。这个首部请求对 Family 域进行基本认证。
- 在图 12-2c 中，浏览器收到了 401 质询，弹出对话框，询问 Family 域的用户名和密码。用户输入用户名和密码时，浏览器会用一个冒号将其连接起来，编码成“经过扰码的”Base-64 表示形式（下节介绍），然后将其放在 Authorization 首部中回送。
- 在图 12-2d 中，服务器对用户名和密码进行解码，验证它们的正确性，然后用一条 HTTP 200 OK 报文返回所请求的报文。

图解：

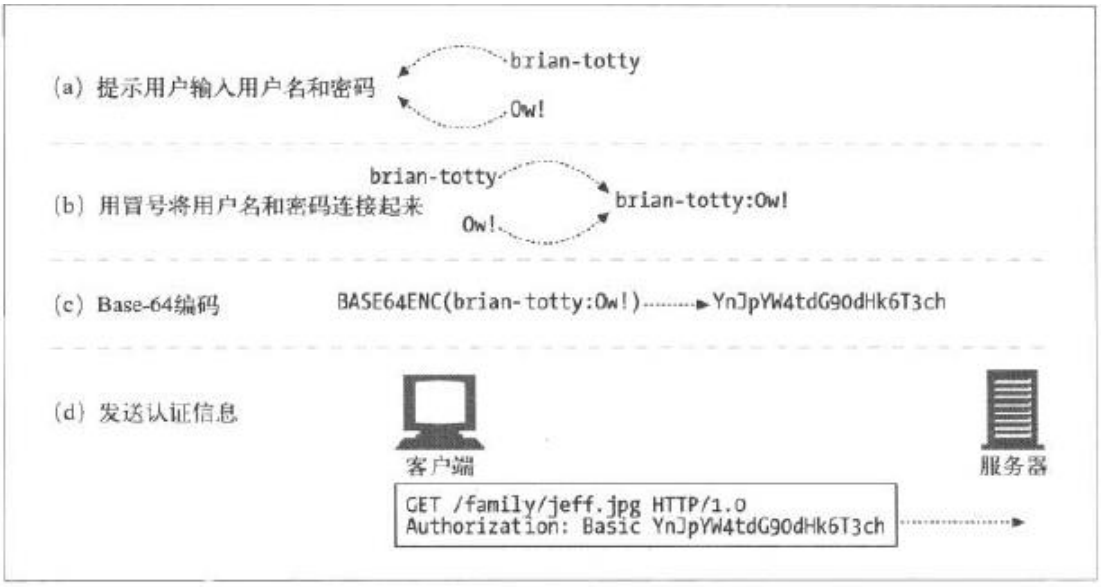


图 12-4　从用户名和密码中生成一个基本认证首部

八、摘要认证

1. 摘要认证对基本认证的改进：
   - 永远不会以明文方式在网络上发送密码。
   - 可以防止恶意用户捕获并重放认证的握手过程。
   - 可以有选择地防止对报文内容的篡改。
   - 防范其他几种常见的攻击方式。
2. 摘要认证就是不是让客户端发送密码而是一个“摘要”，就像“指纹”一样。
3. 摘要认证的工作原理：

- 在图 13-1a 中，客户端请求了某个受保护文档。
- 在图 13-1b 中，在客户端能够证明其知道密码从而确认其身份之前，服务器拒绝提供文档。服务器向客户端发起质询，询问用户名和摘要形式的密码。
- 在图 13-1c 中，客户端传递了密码的摘要，证明它是知道密码的。服务器知道所有用户的密码，[5] 因此可以将客户提供的摘要与服务器自己计算得到的摘要进行比较，以验证用户是否知道密码。另一方在不知道密码的情况下，很难伪造出正确的摘要。
- 在图 13-1d 中，服务器将客户端提供的摘要与服务器内部计算出的摘要进行对比。如果匹配，就说明客户端知道密码（或者很幸运地猜中了！）。可以设置摘要函数，使其产生很多数字，让人不可能幸运地猜中摘要。服务器进行了匹配验证之后，会将文档提供给客户端——整个过程都没有在网络上发送密码。

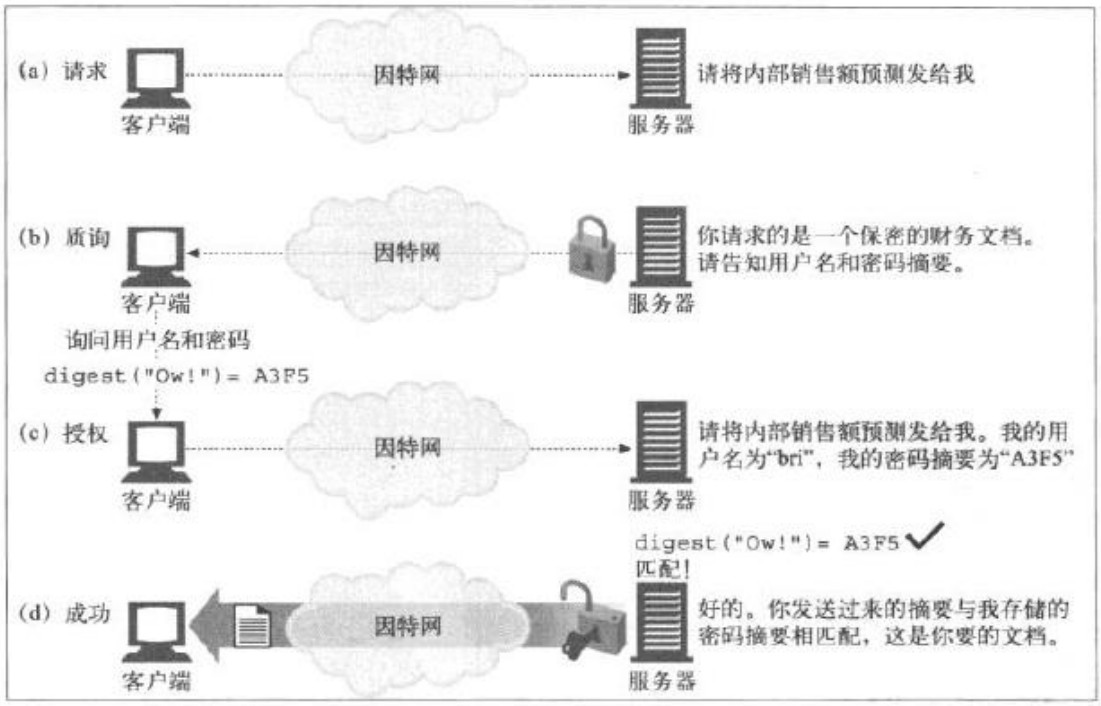


图 13-1　用摘要来实现隐藏密码的认证

4. 所谓摘要是对信息主体的浓缩。摘要是一种单向函数，主要用于将无限的输入值转换为有限的浓缩输出值。常见的摘要函数如 MD5，是将任意长度的字节序列转换为一个 128 的摘要，用 32 个十六进制字符表示出来。
5. 摘要认证的握手机制：

- 在第（1）步中，服务器会计算出一个随机数。在第（2）步中，服务器将这个随机数放在 WWW-Authenticate 质询报文中，与服务器所支持的算法列表一同发往客户端。
- 在第（3）步中，客户端选择一个算法，计算出密码和其他数据的摘要。在第（4）步中，将摘要放在一条 Authorization 报文中发回服务器。如果客户端要对服务器进行认证，可以发送客户端随机数。
- 在第（5）步中，服务器接收摘要、选中的算法以及支撑数据，计算出与客户端相同的摘要。然后服务器将本地生成的摘要与网络传送过来的摘要进行比较，验

注 9：随机数这个词表示“本次”或“临时的”。在计算机安全的概念中，随机数捕获了一个特定的时间点，将其加入到安全计算之中。

证其是否匹配。如果客户端反过来用客户端随机数对服务器进行质询，就会创建客户端摘要。服务器可以预先将下一个随机数计算出来，提前将其传递给客户端，这样下一次客户端就可以预先发送正确的摘要了。

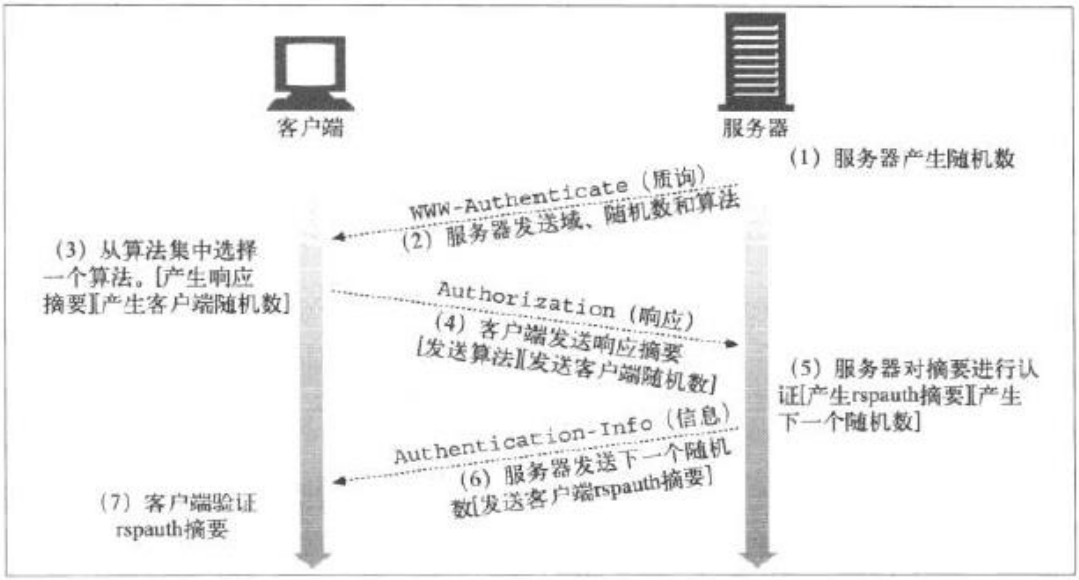


图 13-2 摘要认证的握手机制

九、安全 HTTP（HTTPs）

1. HTTPs 的网络结构：

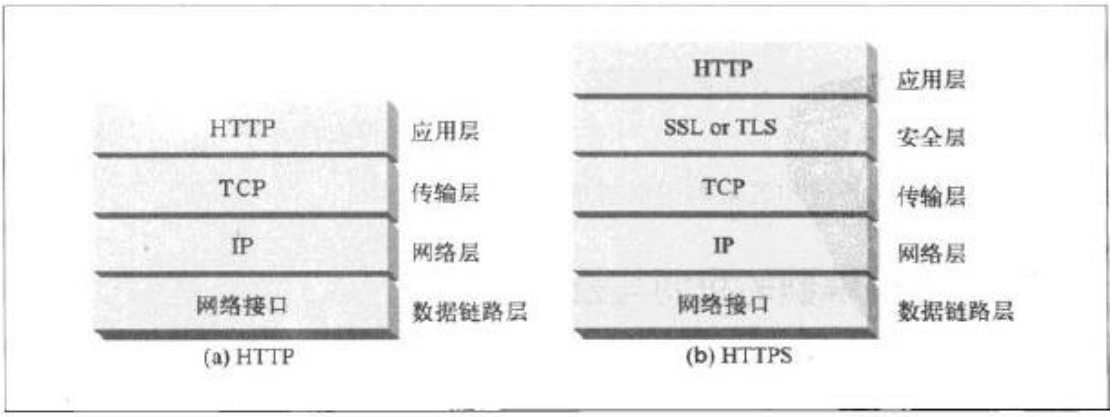


图 14-2　HTTPS 是位于安全层之上的 HTTP，这个安全层位于 TCP 之上

2. 数字加密的概念介绍:

- **密码**

对文本进行编码，使偷窥者无法识别的算法。

- **密钥**

改变密码行为的数字化参数。

- **对称密钥加密系统**

编/解码使用相同密钥的算法。

- **不对称密钥加密系统**

编/解码使用不同密钥的算法。

- **公开密钥加密系统**

一种能够使数百万计算机便捷地发送机密报文的系统。

- **数字签名**

用来验证报文未被伪造或篡改的校验和。

- **数字证书**

由一个可信的组织验证和签发的识别信息。

a) 简单的公开密钥加密技术:

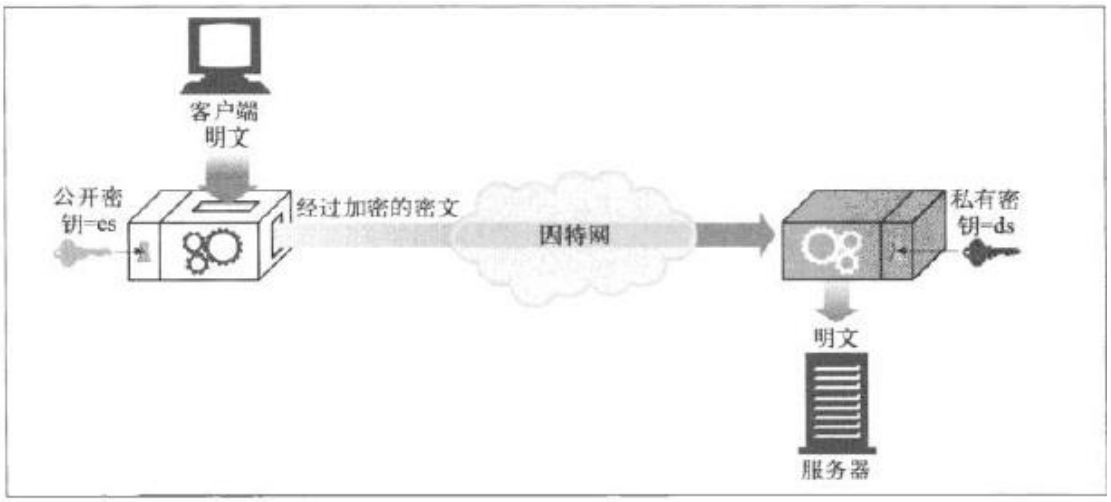


图 14-8　公开密钥加密技术是非对称的，为编码和解码使用了不同的密钥

b)　解密的数字签名：

- 节点 A 将变长报文提取为定长的摘要。
- 节点 A 对摘要应用了一个“签名”函数，这个函数会将用户的私有密钥作为参数。因为只有用户才知道私有密钥，所以正确的签名函数会说明签名者就是其所有者。在图 14-10 中，由于解码函数 D 中包含了用户的私有密钥，所以我们将其作为签名函数使用。[9]
- 一旦计算出签名，节点 A 就将其附加在报文的末尾，并将报文和签名都发送给 B。

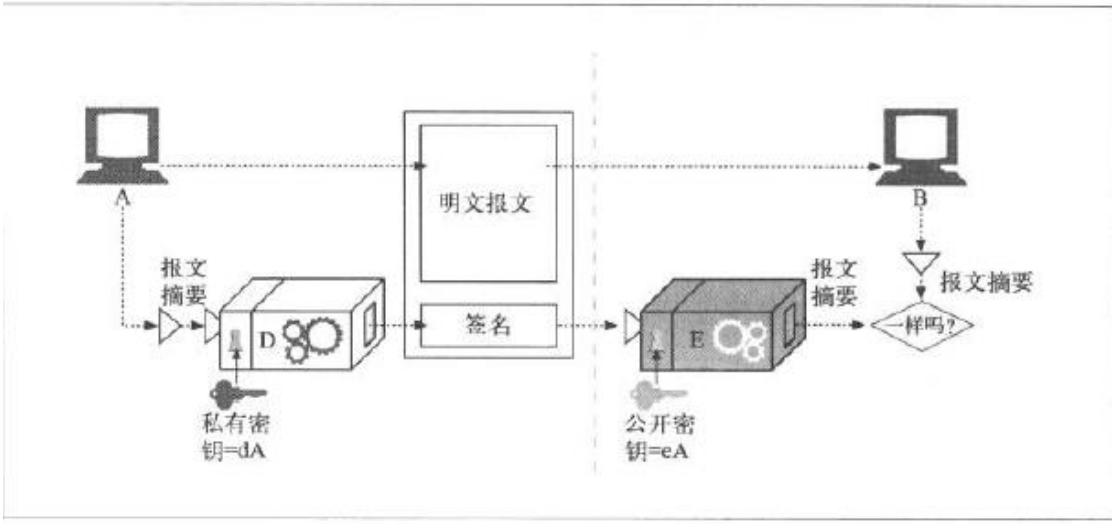


图 14-10　解密的数字签名

c)　典型的数字签名证书的格式和认证：

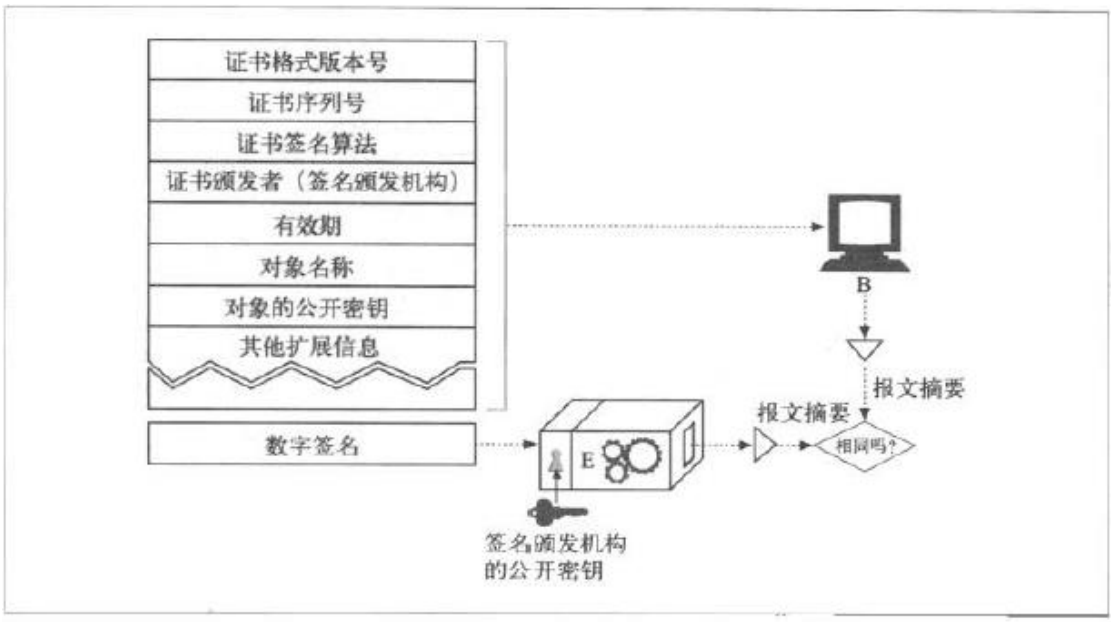


图 14-12　验证签名是真的

3. HTTP 的默认端口是 80，而 HTTPs 的默认端口号 443，HTTP 事务和 HTTPs 的事务是不同的，如下：

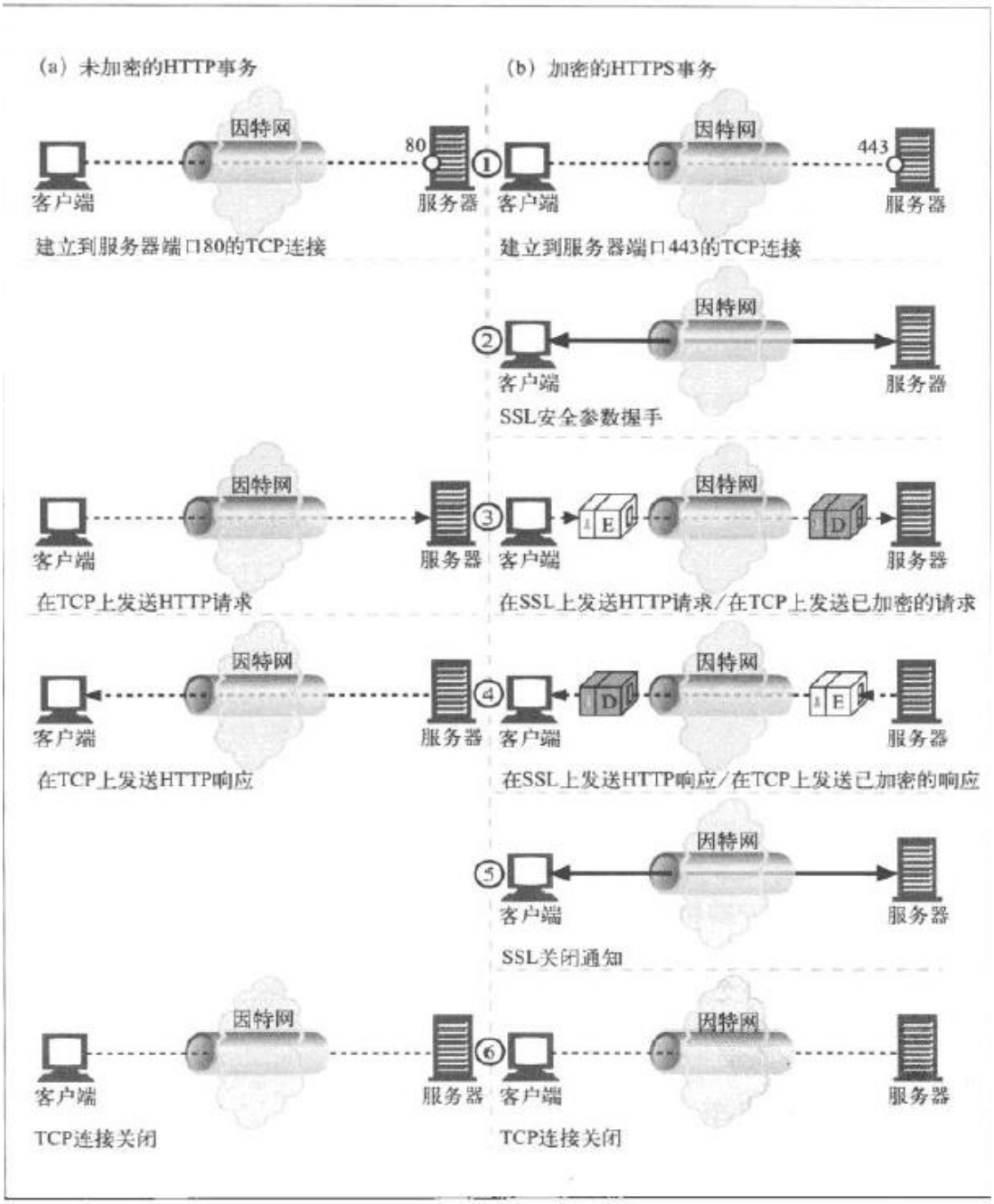


图 14-15　HTTP 和 HTTPS 事务

4. SSL 是支持双向认证的。将服务器证书承载回客户端，再讲客户端的证回送给服务器。

   web 站点证书的有效性：

   日期检测

   签名颁发者可信度检测：任何人都可以生成证书，但是有些证书办法机构（CA）是权威的。

   签名检测

   站点身份检测

5. 其余的虚拟主机什么的就暂且不管了。笔记整理完成于 2019 年 2 月 19 日星期二，祝福自己，希望春招能找到个上海的好实习，秋招的时候能找到好工作！fighting!