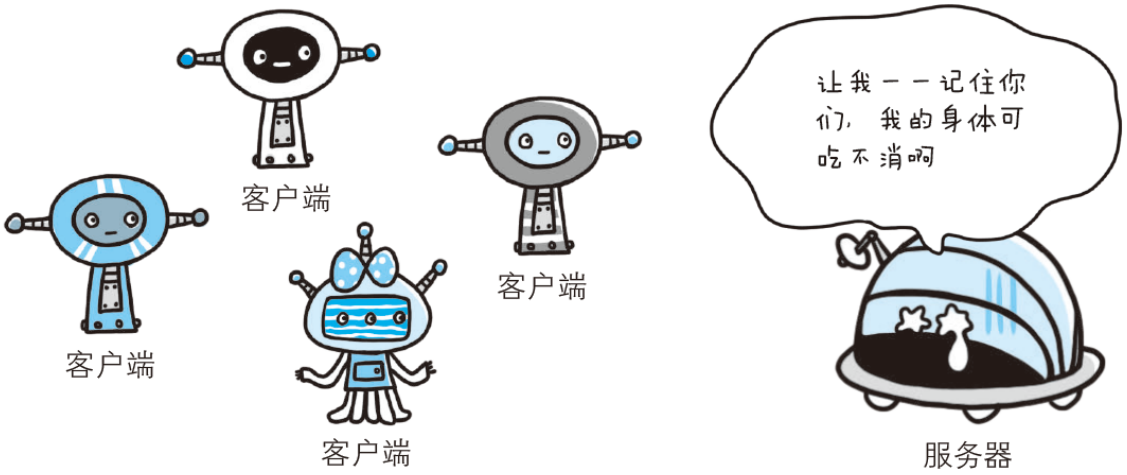
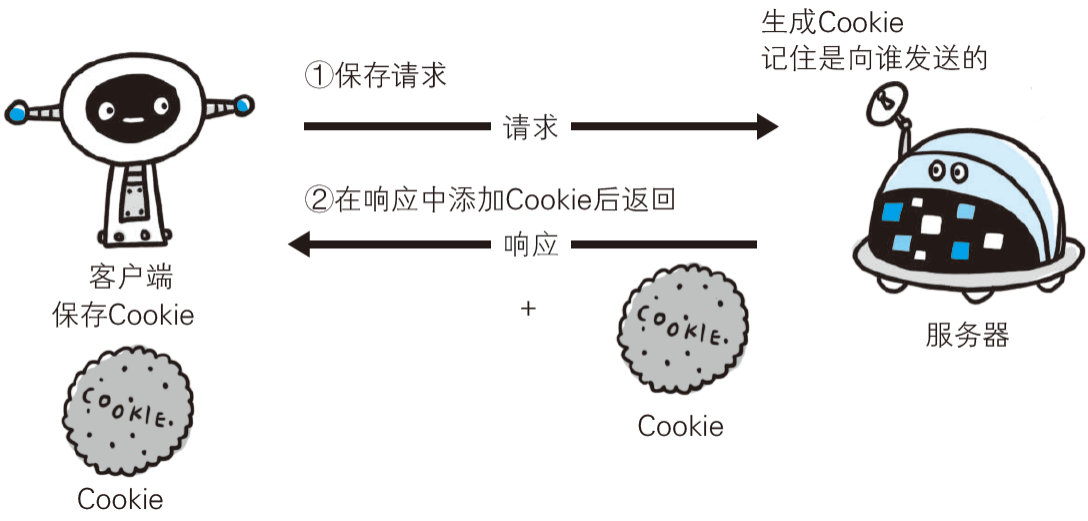
1. 使用Cookie来管理状态

HTTP 是无状态协议，说明它不能以状态来区分和管理请求和响应。也就是说，无法根据之前的状态进行本次的请求处理。
不可否认，无状态协议当然也有它的优点。由于不必保存状态，自然可减少服务器的CPU 及内存资源的消耗。从另一侧面来说，也正是因为HTTP 协议本身是非常简单的，所以才会被应用在各种场景里。

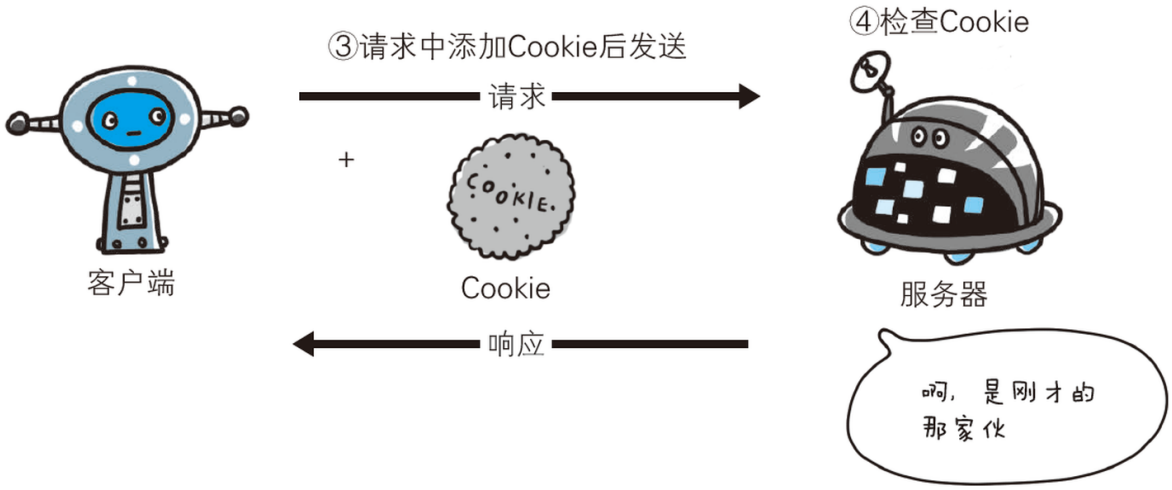


我们登录淘宝的时候首先要登录，我们看到了一个商品点进去，进行了页面跳转/刷新，按照HTTP的无状态协议岂不是又要登录一次?
所以为了解决这个问题，Cookie诞生了，在保留无状态协议这个特征的同时又要解决类似记录状态的矛盾问题。Cookie 技术通过在请求和响应报文中写入Cookie 信息来控制客户端的状态。
Cookie 会根据从服务器端发送的响应报文内的一个叫做Set-Cookie的首部字段信息，通知客户端保存Cookie。当下次客户端再往该服务器发送请求时，客户端会自动在请求报文中加入Cookie 值后发送出去。
服务器端发现客户端发送过来的Cookie 后，会去检查究竟是从哪一个客户端发来的连接请求，然后对比服务器上的记录，最后得到之前的状态信息。

- 没有Cookie信息状态下的请求



- 第2次以后（存有Cookie信息状态）的请求



上图很清晰地展示了发生Cookie 交互的情景。
HTTP 请求报文和响应报文的内容如下（数字和图中对应）。
①请求报文（没有Cookie 信息的状态）

```
GET /reader/ HTTP/1.1
Host: hackr.jp
*首部字段内没有Cookie的相关信息
```

②响应报文（服务器端生成Cookie 信息）

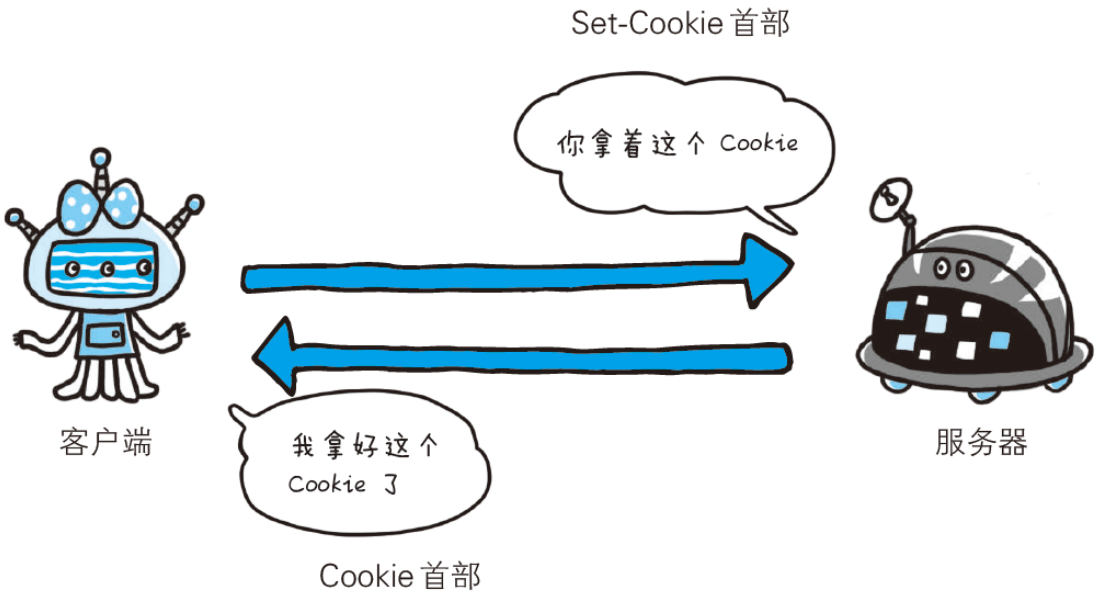
```
HTTP/1.1 200 OK
Date: Thu, 12 Jul 2012 07:12:20 GMT
Server: Apache
<Set-Cookie: sid=1342077140226724; path=/; expires=Wed,10-Oct-12 07:12:20 GMT>
Content-Type: text/plain; charset=UTF-8
```

③请求报文（自动发送保存着的Cookie 信息）

```
GET /image/ HTTP/1.1
Host: hackr.jp
Cookie: sid=1342077140226724
```

2. 关于Cookie 的首部字段

首部字段名	说明	首部类型
Set-Cookie	开始状态管理所使用的Cookie信息	响应首部字段
Cookie	服务器接收到的Cookie信息	请求首部字段



2.1 Set-Cookie

```
Set-Cookie: status=enable; expires=Tue, 05 Jul 2011 07:26:31 GMT; ⇒
path=/; domain=.hackr.jp;
```

当服务器准备开始管理客户端的状态时，会事先告知各种信息。下面的表格列举了Set-Cookie 的字段值。

2.1.1Set-Cookie 字段的属性

属性	说明
NAME=VALUE	赋予Cookie的名称和其值（必需项）
expires=DATE	Cookie的有效期（若不明确指定则默认为浏览器关闭前为止）
path=PATH	将服务器上的文件目录作为Cookie的适用对象（若不指定则默认为文档所在的文件目录）
domain=域名	作为Cookie适用对象的域名（若不指定则默认为创建Cookie的服务器的域名）

2.1.2 expires 属性

Cookie 的expires 属性指定浏览器可发送Cookie 的有效期。当省略expires 属性时，Cookie仅在浏览器关闭之前有效。
另外，一旦Cookie 从服务器端发送至客户端，服务器端就不存在可以显式删除Cookie 的方法。但可通过覆盖已过期的Cookie，实现对客户端Cookie 的实质性删除操作。

2.1.3 path 属性

Cookie 的path 属性可用于限制指定Cookie 的发送范围的文件目录。不过另有办法可避开这项限制，看来对其作为安全机制的效果不能抱有期待。

2.1.4 domain 属性

通过Cookie 的domain 属性指定的域名可做到与结尾匹配一致。比如， 当指定http://example.com 后， 除http://example.com 以外，Example Domain或www2.example.com 等都可以发送Cookie。因此，除了针对具体指定的多个域名发送Cookie 之外，不指定domain 属性显得更安全。

2.1.5 secure 属性

Cookie 的secure 属性用于限制Web 页面仅在HTTPS 安全连接时，才可以发送Cookie。发送Cookie 时，指定secure 属性的方法如下所示。
Set-Cookie: name=value; secure
以上例子仅当在https： //Example Domain（HTTPS）安全连接的情况下才会进行Cookie 的回收。也就是说，即使域名相同时http : //Example Domain（HTTP） 也不会发生Cookie 回收行为。当省略secure 属性时，不论HTTP 还是HTTPS，都会对Cookie 进行回收。

2.1.6 HttpOnly 属性

Cookie 的HttpOnly 属性是Cookie 的扩展功能，它使JavaScript 脚本无法获得Cookie。其主要目的为防止跨站脚本攻击（Cross-sitescripting，XSS）对Cookie 的信息窃取。
发送指定HttpOnly 属性的Cookie 的方法如下所示。
Set-Cookie: name=value; HttpOnly
通过上述设置，通常从Web 页面内还可以对Cookie 进行读取操作。但使用JavaScript 的document.cookie 就无法读取附加HttpOnly 属性后的Cookie 的内容了。因此，也就无法在XSS 中利用JavaScript 劫持Cookie 了。
虽然是独立的扩展功能，但Internet Explorer 6 SP1 以上版本等当下的主流浏览器都已经支持该扩展了。另外顺带一提，该扩展并非是为了防止XSS 而开发的。

2.2 Cookie

Cookie: status=enable
首部字段Cookie 会告知服务器，当客户端想获得HTTP 状态管理支持时，就会在请求中包含从服务器接收到的Cookie。接收到多个Cookie 时，同样可以以多个Cookie 形式发送。

3 Session 管理及Cookie 应用

3.1 什么是Session

在计算机中，尤其是在网络应用中，称为“会话控制”。Session 对象存储特定用户会话所需的属性及配置信息。这样，当用户在应用程序的 Web 页之间跳转时，存储在 Session 对象中的变量将不会丢失，而是在整个用户会话中一直存在下去。当用户请求来自应用程序的 Web 页时，如果该用户还没有会话，则 Web 服务器将自动创建一个 Session 对象。当会话过期或被放弃后，服务器将终止该会话。Session 对象最常见的一个用法就是存储用户的首选项。例如，如果用户指明不喜欢查看图形，就可以将该信息存储在 Session 对象中。

3.2 通过Cookie来管理Session

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。

Session 管理及Cookie 状态管理

- 步骤一：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。
- 步骤二：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。
- 步骤三：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）A 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。