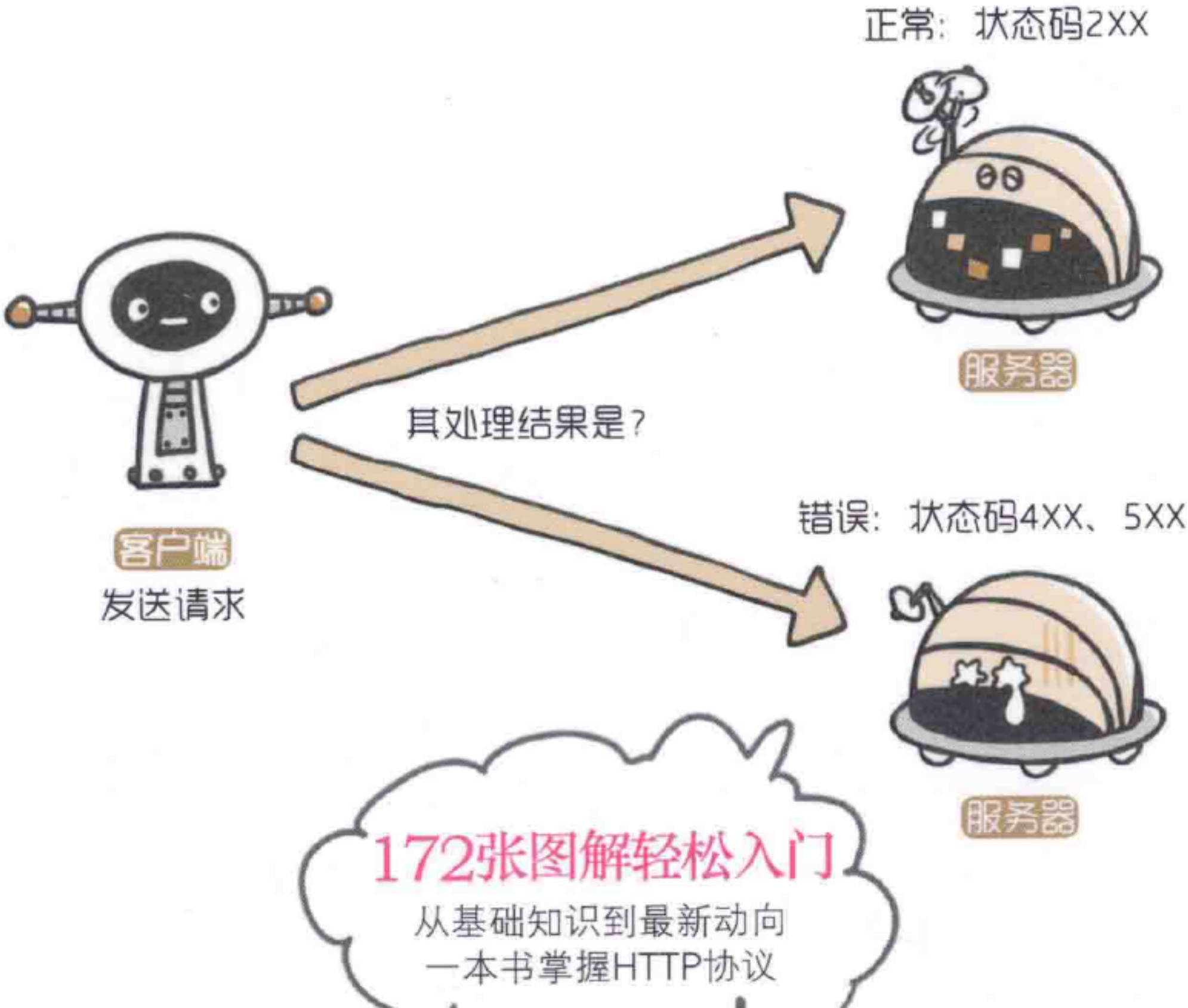
TURING
图灵程序
设计丛书

图解HTTP

OWASP
日本分会主席 【日】上野 宣 著
于均良 译



172张图解轻松入门
从基础知识到最新动向
一本书掌握HTTP协议

人民邮电出版社
POSTS & TELECOM PRESS



图解HTTP

- 172张图解穿插文中，掌握HTTP协议完全不枯燥！
- Web前端开发者必备，从基础知识到最新动向一网打尽！

本书前半部分由HTTP的成长发展史娓娓道来，基于HTTP 1.1 标准讲解通信过程，包括HTTP方法、协议格式、报文结构、首部字段、状态码等的具体含义，还分别讲解HTTP通信过程中代理、网关、隧道等的作用。接着介绍SPDY、WebSocket、WebDAV等HTTP的扩展功能。作者还从细节方面举例，让读者更好地理解何为无状态（stateless）、301和302重定向的区别哪、缓存机制，等等。本书后半部分的重心放在Web安全上，涵盖HTTPS、SSL、证书认证、加密机制、Web攻击手段等内容。

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议 计算机/网络通信

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-35153-1

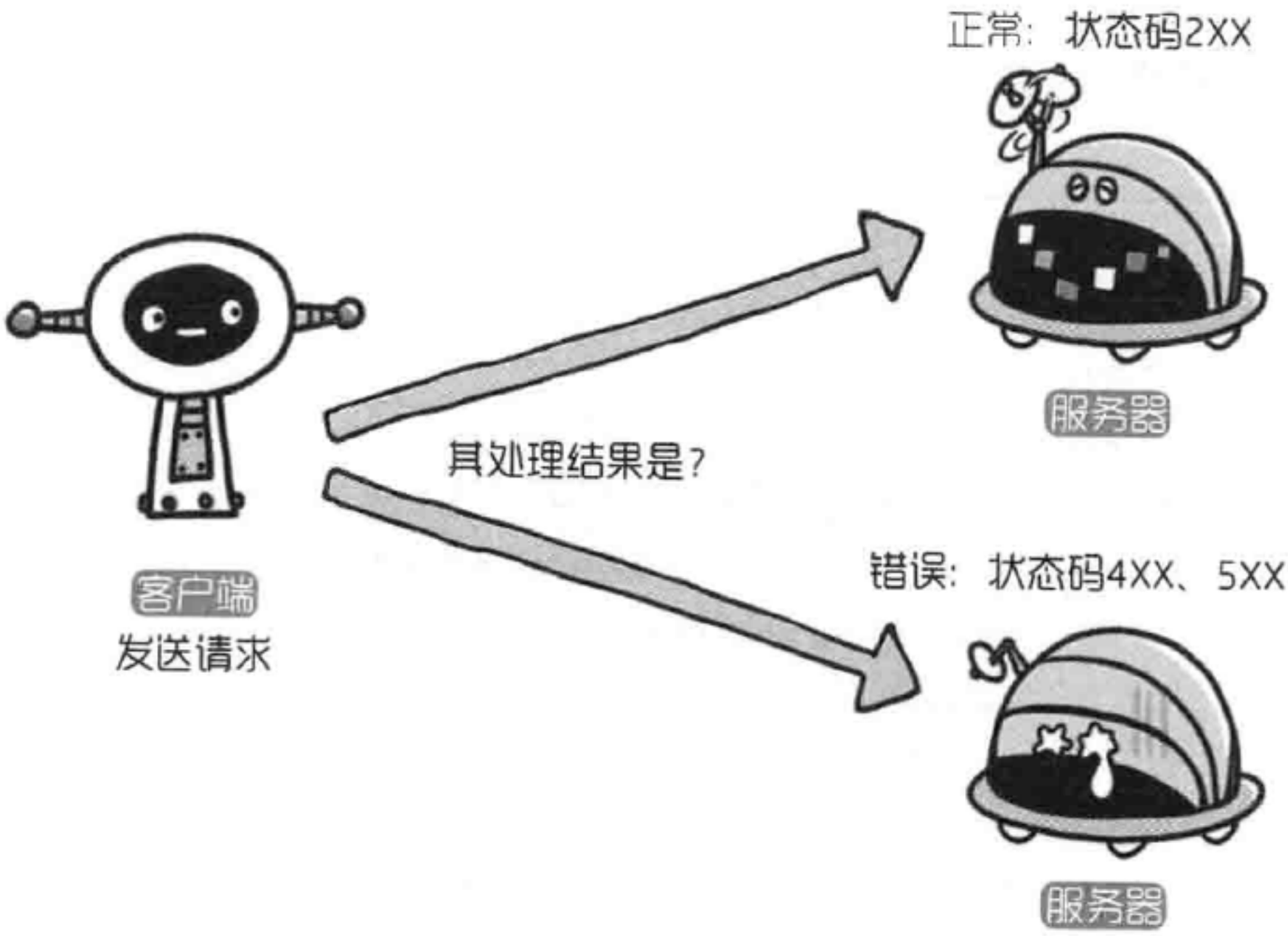
ISBN 978-7-115-35153-1

定价：49.00元

TURING
图灵程序
设计丛书

图解HTTP

OWASP
日本分会主席 【日】上野 宣 著
于均良 译



人民邮电出版社
北 京

图书在版编目（CIP）数据

图解HTTP /（日）上野宣著；于均良译. -- 北京：
人民邮电出版社，2014.5
（图灵程序设计丛书）
ISBN 978-7-115-35153-1

Ⅰ. ①图… Ⅱ. ①上… ②于… Ⅲ. ①计算机网络—
通信协议 Ⅳ. ①TN915.04

中国版本图书馆CIP数据核字（2014）第055492号

内 容 提 要

本书对互联网基盘——HTTP协议进行了全面系统的介绍。作者由HTTP协议的发展历史娓娓道来，严谨细致地剖析了HTTP协议的结构，列举诸多常见通信场景及实战案例，最后延伸到Web安全、最新技术动向等方面。本书的特色为在讲解的同时，辅以大量生动形象的通信图例，更好地帮助读者深刻理解HTTP通信过程中客户端与服务器之间的交互情况。读者可通过本书快速了解并掌握HTTP协议的基础，前端工程师分析抓包数据，后端工程师实现REST API、实现自己的HTTP服务器等过程中所需的HTTP相关知识点本书均有介绍。

本书适合Web开发工程师，以及对HTTP协议感兴趣的各层次读者。

◆ 著　　[日] 上野 宣
译　　于均良
责任编辑　徐　骞
责任印制　焦志炜
◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京天宇星印刷厂印刷
◆ 开本：880×1230　1/32
印张：8.25
字数：220千字　　2014年5月第1版
印数：1–4000册　　2014年5月北京第1次印刷
著作权合同登记号　图字：01-2013-8984号

定价：49.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京崇工商广字第0021号

版权声明

HTTP の教科書（ISBN 978-4-7981-2625-8）
Copyright © 2013 by SEN UENO.
Original Japanese edition published by SHOEISHA Co., Ltd
Simplified Chinese Character translation rights arranged with SHOEISHA Co., Ltd
through CREEK & RIVER Co., Ltd
Simplified Chinese Character translation copyright © 2014 by Posts & Telecom Press

本书中文简体字版由 SHOEISHA Co., Ltd. 授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。
版权所有，侵权必究。

（图灵公司感谢李典对本书的审读。）

译者序

目前，国内讲解 HTTP 协议的书实在太少了。

在我的印象中，讲解网络协议的书仅有两本。一本是《HTTP 权威指南》，但其厚度令人望而生畏；另一本是《TCP/IP 详解，卷 1》，内容艰涩难懂，学习难度较大。这两本书都是被读者们奉为“圣经”的经典之作，大师们的授道自然无可挑剔，但关键是它们对初学者都不那么友好，大家的学习信心很容易受到打击，阅读中途或将束之高阁。本书的出现及时缓解了该问题。

HTTP 协议本身并不复杂，理解起来也不会花费太多学习成本，但纯概念式的学习稍显单调。前端工程师也许对各种具有炫酷效果的页面的实现技巧、赏心悦目的 UI 框架更感兴趣，但因此常常忽视了 HTTP 协议这部分基础内容。实际上，如果想要在专业技术道路上走得更坚实，绝对不能绕开学习 HTTP 协议这一环节。对基础及核心部分的深入学习是成为一名专业技术人员的前提，以不变应万变才是立足之本。

我在学习 Web 开发的过程中，曾接触到编写网络爬虫程序、分析抓包数据、实现 HTTP 服务器、提供网站 REST API、修改后端定制框架等方面，它们无一例外，都会用到 HTTP 协议的各方面知识，并且某些细节无法通过查阅资料立即领会到，还需依靠扎实的基础及平日里的积累。

本书作者的写作手法平实易懂，内容讲解透彻到位。前半部分由 HTTP 的成长发展史娓娓道来，基于 HTTP 1.1 标准讲解通信过程，包括 HTTP 方法、协议格式、报文结构、首部字段、状态码等的具体含义，还分别讲解 HTTP 通信过程中代理、网关、隧道等的作用。接着介绍 SPDY、WebSocket、WebDAV 等 HTTP 的扩展功能。作者还从细节方面举例，让读者更好地理解何为无状态（stateless）、301 和 302 重定向的区别在哪、缓存机制，等等。本书后半部分的重心放在 Web 安全上，涵盖 HTTPS、SSL、证书认证、加密机制、Web 攻击手段等内容。

旨在让读者对 HTTP 协议形成一个整体概念，明确设计 HTTP 的目的及意义所在，了解 HTTP 的工作机制，掌握报文中常用的首部字段，返回结果状态码的作用，对各种客户端与服务器的通信交互场景的细节等都做到了然于心，从而在平时的开发工作中独立思考，迅速准确地定位分析由 HTTP 引发的问题，并辅以适当的方法加以解决。

本书图文并茂，大量图片穿插文中，生动形象地向读者介绍每一个应用案例，减少了读者阅读时的枯燥感。借助一张张配图，读者们不仅会加深视觉记忆，在轻松愉悦的氛围中，还可以更深刻地理解通信机制等背后的工作原理。正所谓一图胜千文。

在本书即将付梓之际，感谢 EMC 首席工程师高博学长、IBM 工程师李亚舟（Fleuria）、豆瓣运维工程师钱龙、全栈工程师缪思源（Aveline Swan）以及姜莹等好友。他们在繁重的工作之余，牺牲个人闲暇时间，耐心地帮我扫清技术疑点，修正翻译疏漏，在此谨表示衷心的感谢。

最后祝大家阅读愉快！

于均良

2014年1月

前言

本书的上一版是2004年出版的《今夜わかるHTTP》(中文译名：今晚我们一起学习HTTP，翔泳社)。和当时一样，现在互联网的主流仍是Web，但人们对Web的要求却不断地发生变化。Google在2005年推出了地图服务Google Maps，很多人看到这一Web应用程序的界面后感到十分震惊。因为在此之前，我们只能借助桌面应用程序或Flash等方式，实现流畅滚动及视角放大缩小等功能，如今这些功能仅需一个Web浏览器就能呈现了。也许正是由于Google Maps的出现，人们对Web的要求才开始变得多了起来。发送请求、等待响应，这些HTTP中稀松平常的功能已经无法满足人们的需求了。于是，Web不再停留在HTTP/1.1版本，在保持HTTP简洁的同时，也开始开发新的功能。我之所以要撰写《今夜わかるHTTP》一书，是因为我发现多数Web应用程序开发者并不了解支撑Web基础的HTTP协议。我坚信通过学习协议，大家能更深刻地理解Web开发。即使是在本书撰写完成后的今天，我的这一想法仍未改变，肯定还有很多开发者尚未了解HTTP协议。

对HTTP协议有了更深入的理解后，也许你会从中得到一些启发。不再囿于HTTP/1.1版本的制约，你也能开发出Google Maps那样的应用程序。

本书不仅面向Web应用程序的开发者，还面向使用Web的软件开发者、Web风险评估的安全工程师、前端工程师以及Web使用者等与Web相关的所有读者，希望这本书能对大家有所帮助。

写于华盛顿DC的酒店

2013年1月吉日

TRICORDER株式会社 上野宣

致谢

Masato Kinugawa先生

感谢您细致的检查核对，并站在读者的角度提出宝贵建议。当您指出我书中的 MD5 hash 值不是 abcd，而是 abc 时，我感到十分惊讶，感叹您不愧是世界级权威专家。能和您一起工作不胜荣幸。

山崎圭吾先生

感谢您及时帮我检查并提出真知灼见。每当我对稿件有所改动，您总是立即着手帮我核对。我曾担心这样是否会对您的本职工作造成影响，不过我就不在此和您客气了，总之十分感谢您的帮助。

Netagent株式会社 长谷川阳介先生

您曾对我说过："不感兴趣的领域我不了解，但我擅长的领域就放心交给我吧！"感谢您以自己独特的方式对本书进行核查。作为安全领域里首屈一指的权威专家，您能协助本书进行审读工作，我倍感荣幸。

我身边的朋友们

在我撰写本书时，我常常顾不上工作，有时又疏于写作，给大家添了很多麻烦。正是得到了诸位的理解与支持，本书才得以顺利出版。在此，我要特别感谢本书的编辑野村先生。

目录

第1章 了解Web及网络基础 001

1.1 使用HTTP协议访问Web ……002
1.2 HTTP的诞生 ……003
1.2.1 为知识共享而规划Web …… 003
1.2.2 Web成长时代 …… 004
1.2.3 驻足不前的HTTP …… 005
1.3 网络基础TCP/IP ……006
1.3.1 TCP/IP协议族 …… 006
1.3.2 TCP/IP的分层管理 …… 007
1.3.3 TCP/IP通信传输流 …… 009
1.4 与HTTP关系密切的协议：IP、TCP和DNS ……010
1.4.1 负责传输的IP协议 …… 011
1.4.2 确保可靠性的TCP协议 …… 012
1.5 负责域名解析的DNS服务 ……013
1.6 各种协议与HTTP协议的关系 ……014
1.7 URI和URL ……016
1.7.1 统一资源标识符 …… 016
1.7.2 URI格式 …… 017

第2章 简单的HTTP协议 021

2.1 HTTP协议用于客户端和服务器端之间的通信 ……022
2.2 通过请求和响应的交换达成通信 ……022
2.3 HTTP是不保存状态的协议 ……025
2.4 请求URI定位资源 ……026
2.5 告知服务器意图的HTTP方法 ……027
2.6 使用方法下达命令 ……033
2.7 持久连接节省通信量 ……034
2.7.1 持久连接 …… 036
2.7.2 管线化 …… 037
2.8 使用Cookie的状态管理 ……037

第3章 HTTP报文内的HTTP信息 041

3.1 HTTP报文 ……042

3.2 请求报文及响应报文的结构……042
3.3 编码提升传输速率……044
3.3.1 报文主体和实体主体的差异 …… 044
3.3.2 压缩传输的内容编码 …… 044
3.3.3 分割发送的分块传输编码 …… 045
3.4 发送多种数据的多部分对象集合……046
3.5 获取部分内容的范围请求……048
3.6 内容协商返回最合适的内容……050

第4章 返回结果的HTTP状态码 053

4.1 状态码告知从服务器端返回的请求结果……054
4.2 2XX 成功 ……055
4.2.1 200 OK …… 055
4.2.2 204 No Content …… 056
4.2.3 206 Partial Content …… 056
4.3 3XX 重定向 ……056
4.3.1 301 Moved Permanently …… 057
4.3.2 302 Found …… 057
4.3.3 303 See Other …… 058
4.3.4 304 Not Modified …… 059
4.3.5 307 Temporary Redirect …… 059
4.4 4XX 客户端错误 ……060
4.4.1 400 Bad Request …… 060
4.4.2 401 Unauthorized …… 060
4.4.3 403 Forbidden …… 061
4.4.4 404 Not Found …… 061
4.5 5XX 服务器错误 ……062
4.5.1 500 Internal Server Error …… 062
4.5.2 503 Service Unavailable …… 062

第5章 与HTTP协作的Web服务器 065

5.1 用单台虚拟主机实现多个域名……066
5.2 通信数据转发程序：代理、网关、隧道……067
5.2.1 代理 …… 068
5.2.2 网关 …… 070
5.2.3 隧道 …… 070
5.3 保存资源的缓存……071
5.3.1 缓存的有效期限 …… 072

5.3.2 客户端的缓存 …… 072

第6章 HTTP首部 075

6.1 HTTP报文首部 ……076
6.2 HTTP首部字段 ……078
6.2.1 HTTP首部字段传递重要信息 …… 078
6.2.2 HTTP首部字段结构 …… 078
6.2.3 4种HTTP首部字段类型 …… 079
6.2.4 HTTP/1.1 首部字段一览 …… 080
6.2.5 非HTTP/1.1首部字段 …… 082
6.2.6 End-to-end首部和Hop-by-hop首部 …… 083
6.3 HTTP/1.1 通用首部字段 ……083
6.3.1 Cache-Control …… 084
6.3.2 Connection …… 091
6.3.3 Date …… 093
6.3.4 Pragma …… 094
6.3.5 Trailer …… 095
6.3.6 Transfer-Encoding …… 096
6.3.7 Upgrade …… 097
6.3.8 Via …… 098
6.3.9 Warning …… 099
6.4 请求首部字段……100
6.4.1 Accept …… 101
6.4.2 Accept-Charset …… 102
6.4.3 Accept-Encoding …… 103
6.4.4 Accept-Language …… 104
6.4.5 Authorization …… 105
6.4.6 Expect …… 106
6.4.7 From …… 107
6.4.8 Host …… 107
6.4.9 If-Match …… 108
6.4.10 If-Modified-Since …… 110
6.4.11 If-None-Match …… 111
6.4.12 If-Range …… 112
6.4.13 If-Unmodified-Since …… 113
6.4.14 Max-Forwards …… 114
6.4.15 Proxy-Authorization …… 115
6.4.16 Range …… 116
6.4.17 Referer …… 116
6.4.18 TE …… 117

6.4.19 User-Agent ····· 118
6.5 响应首部字段 ····· 119
6.5.1 Accept-Ranges ····· 119
6.5.2 Age ····· 120
6.5.3 ETag ····· 120
6.5.4 Location ····· 122
6.5.5 Proxy-Authenticate ····· 123
6.5.6 Retry-After ····· 123
6.5.7 Server ····· 124
6.5.8 Vary ····· 125
6.5.9 WWW-Authenticate ····· 125
6.6 实体首部字段 ····· 126
6.6.1 Allow ····· 126
6.6.2 Content-Encoding ····· 127
6.6.3 Content-Language ····· 128
6.6.4 Content-Length ····· 128
6.6.5 Content-Location ····· 129
6.6.6 Content-MD5 ····· 129
6.6.7 Content-Range ····· 130
6.6.8 Content-Type ····· 131
6.6.9 Expires ····· 131
6.6.10 Last-Modified ····· 132
6.7 为Cookie服务的首部字段 ····· 132
6.7.1 Set-Cookie ····· 134
6.7.2 Cookie ····· 136
6.8 其他首部字段 ····· 137
6.8.1 X-Frame-Options ····· 137
6.8.2 X-XSS-Protection ····· 138
6.8.3 DNT ····· 138
6.8.4 P3P ····· 139

第7章 确保Web安全的HTTPS 141
7.1 HTTP的缺点 ····· 142
7.1.1 通信使用明文可能会被窃听 ····· 142
7.1.2 不验证通信方的身份就可能遭遇伪装 ····· 146
7.1.3 无法证明报文完整性，可能已遭篡改 ····· 148
7.2 HTTP+加密+认证+完整性保护=HTTPS ····· 150
7.2.1 HTTP加上加密处理和认证以及完整性保护后即是HTTPS ····· 150
7.2.2 HTTPS是身披SSL外壳的HTTP ····· 151

7.2.3 相互交换密钥的公开密钥加密技术 …… 152
7.2.4 证明公开密钥正确性的证书 …… 155
7.2.5 HTTPS的安全通信机制 …… 161

第8章 确认访问用户身份的认证 167

8.1 何为认证 …… 168
8.2 BASIC认证 …… 169
8.3 DIGEST认证 …… 171
8.4 SSL客户端认证 …… 173
8.4.1 SSL客户端认证的认证步骤 …… 174
8.4.2 SSL客户端认证采用双因素认证 …… 175
8.4.3 SSL客户端认证必要的费用 …… 175
8.5 基于表单认证 …… 175
8.5.1 认证多半为基于表单认证 …… 176
8.5.2 Session管理及Cookie应用 …… 177

第9章 基于HTTP的功能追加协议 179

9.1 基于HTTP的协议 …… 180
9.2 消除HTTP瓶颈的SPDY …… 180
9.2.1 HTTP的瓶颈 …… 180
9.2.2 SPDY的设计与功能 …… 184
9.2.3 SPDY消除Web瓶颈了吗 …… 185
9.3 使用浏览器进行全双工通信的WebSocket …… 186
9.3.1 WebSocket的设计与功能 …… 186
9.3.2 WebSocket协议 …… 186
9.4 期盼已久的HTTP/2.0 …… 189
9.5 Web服务器管理文件的WebDAV …… 190
9.5.1 扩展HTTP/1.1的WebDAV …… 191
9.5.2 WebDAV内新增的方法及状态码 …… 192

第10章 构建Web内容的技术 195

10.1 HTML …… 196
10.1.1 Web页面几乎全由HTML构建 …… 196
10.1.2 HTML的版本 …… 197
10.1.3 设计应用CSS …… 198
10.2 动态HTML …… 198
10.2.1 让Web页面动起来的动态HTML …… 198

10.2.2 更易控制HTML的DOM …… 198
10.3 Web应用 ……200
10.3.1 通过Web提供功能的Web应用 …… 200
10.3.2 与Web服务器及程序协作的CGI …… 200
10.3.3 因Java而普及的Servlet …… 201
10.4 数据发布的格式及语言……203
10.4.1 可扩展标记语言 …… 203
10.4.2 发布更新信息的RSS/Atom …… 204
10.4.3 JavaScript衍生的轻量级易用JSON…… 206

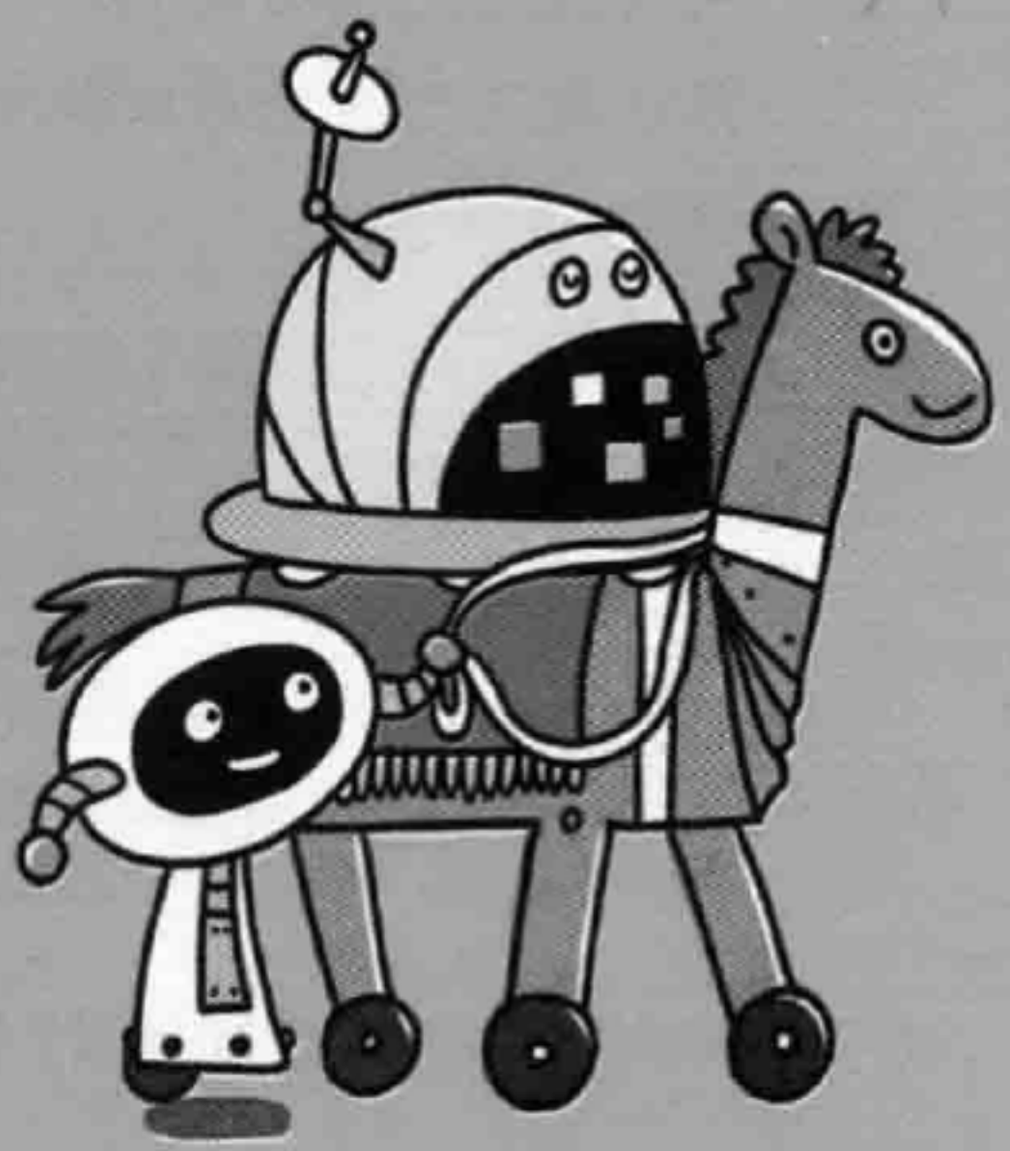
第11章 Web的攻击技术 207

11.1 针对Web的攻击技术 ……208
11.1.1 HTTP不具备必要的安全功能 …… 208
11.1.2 在客户端即可篡改请求 …… 209
11.1.3 针对Web应用的攻击模式…… 210
11.2 因输出值转义不完全引发的安全漏洞……212
11.2.1 跨站脚本攻击 …… 213
11.2.2 SQL注入攻击 …… 218
11.2.3 OS命令注入攻击 …… 223
11.2.4 HTTP首部注入攻击 …… 225
11.2.5 邮件首部注入攻击 …… 228
11.2.6 目录遍历攻击 …… 229
11.2.7 远程文件包含漏洞 …… 230
11.3 因设置或设计上的缺陷引发的安全漏洞……232
11.3.1 强制浏览 …… 232
11.3.2 不正确的错误消息处理 …… 234
11.3.3 开放重定向 …… 237
11.4 因会话管理疏忽引发的安全漏洞……237
11.4.1 会话劫持 …… 238
11.4.2 会话固定攻击 …… 239
11.4.3 跨站点请求伪造 …… 241
11.5 其他安全漏洞……242
11.5.1 密码破解 …… 242
11.5.2 点击劫持 …… 247
11.5.3 DoS攻击 …… 249
11.5.4 后门程序 …… 250

Chapter1

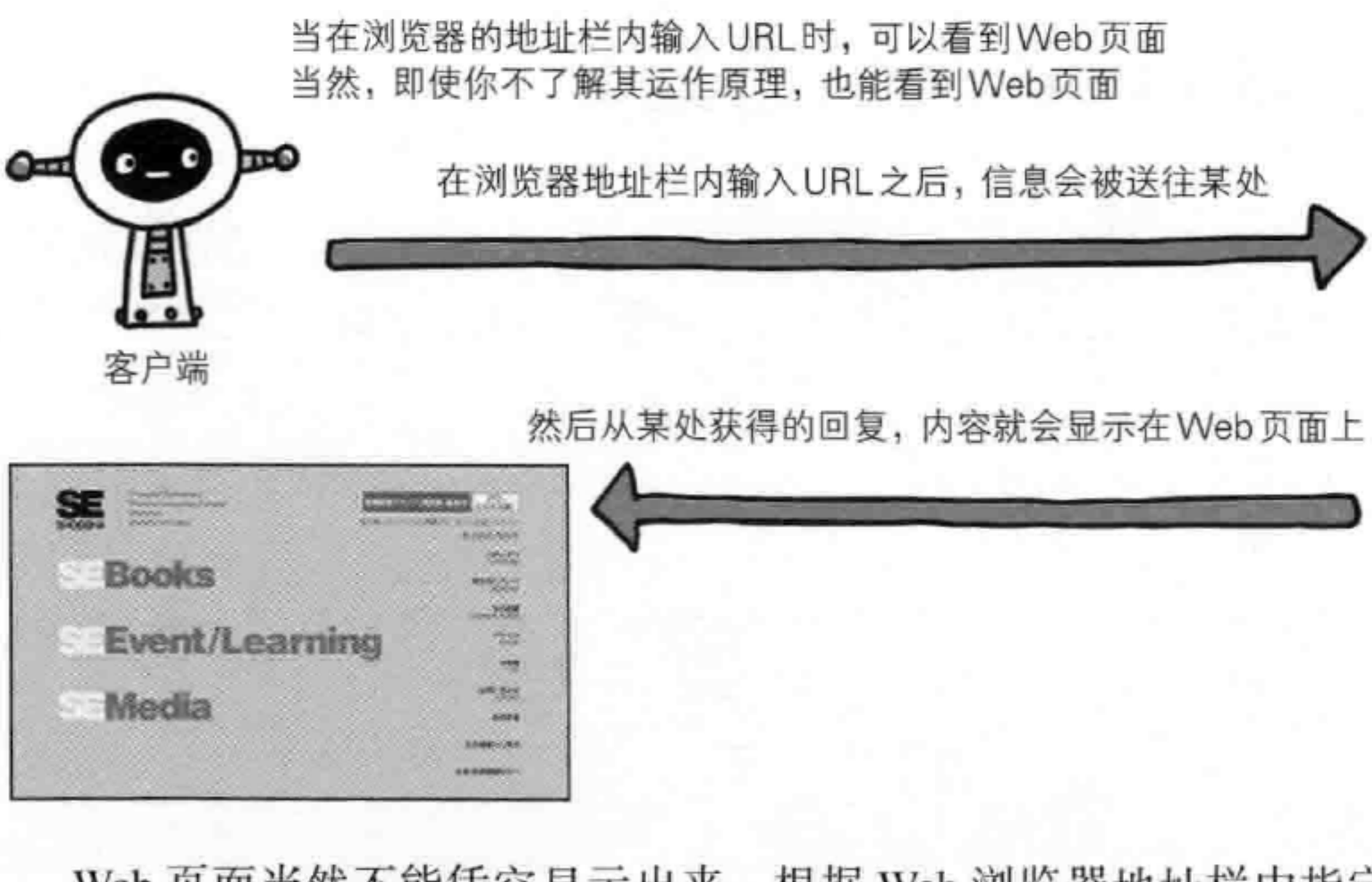
第1章
了解Web及网络基础

本章概述了Web是建立在何种技术之上，以及HTTP协议是如何诞生并发展的。我们从其背景着手，来深入了解这部分内容。



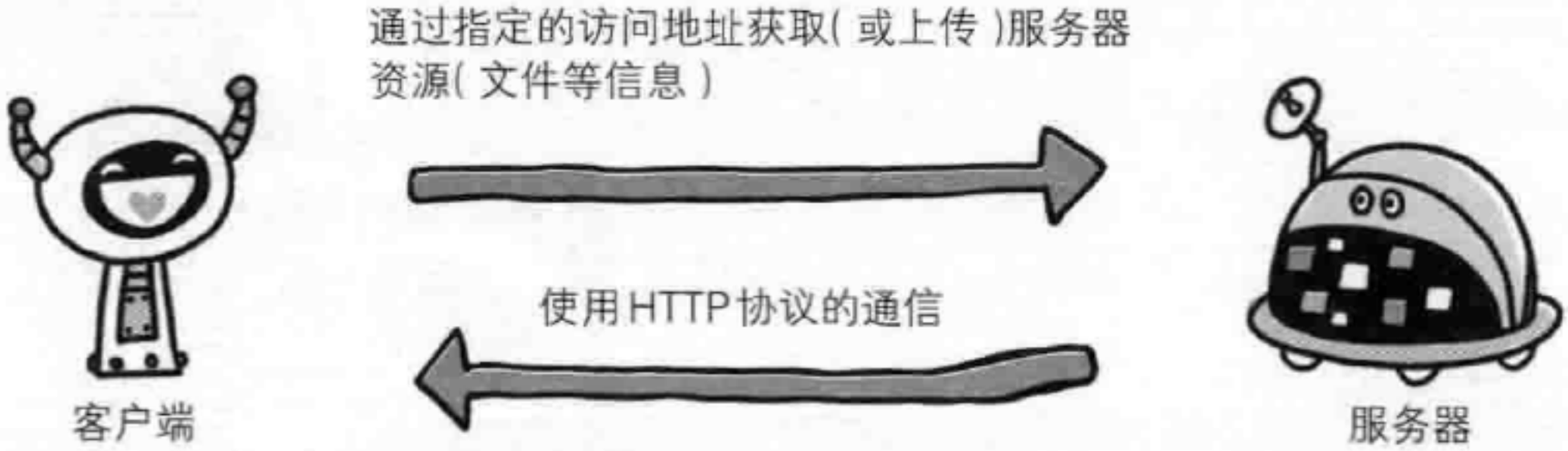
1.1 使用 HTTP 协议访问 Web

你知道当我们在网页浏览器（Web browser）的地址栏中输入 URL 时，Web 页面是如何呈现的吗？



Web 页面当然不能凭空显示出来。根据 Web 浏览器地址栏中指定的 URL，Web 浏览器从 Web 服务器端获取文件资源（resource）等信息，从而显示出 Web 页面。

像这种通过发送请求获取服务器资源的 Web 浏览器等，都可称为客户端（client）。



Web 使用一种名为 HTTP（HyperText Transfer Protocol，超文本传输

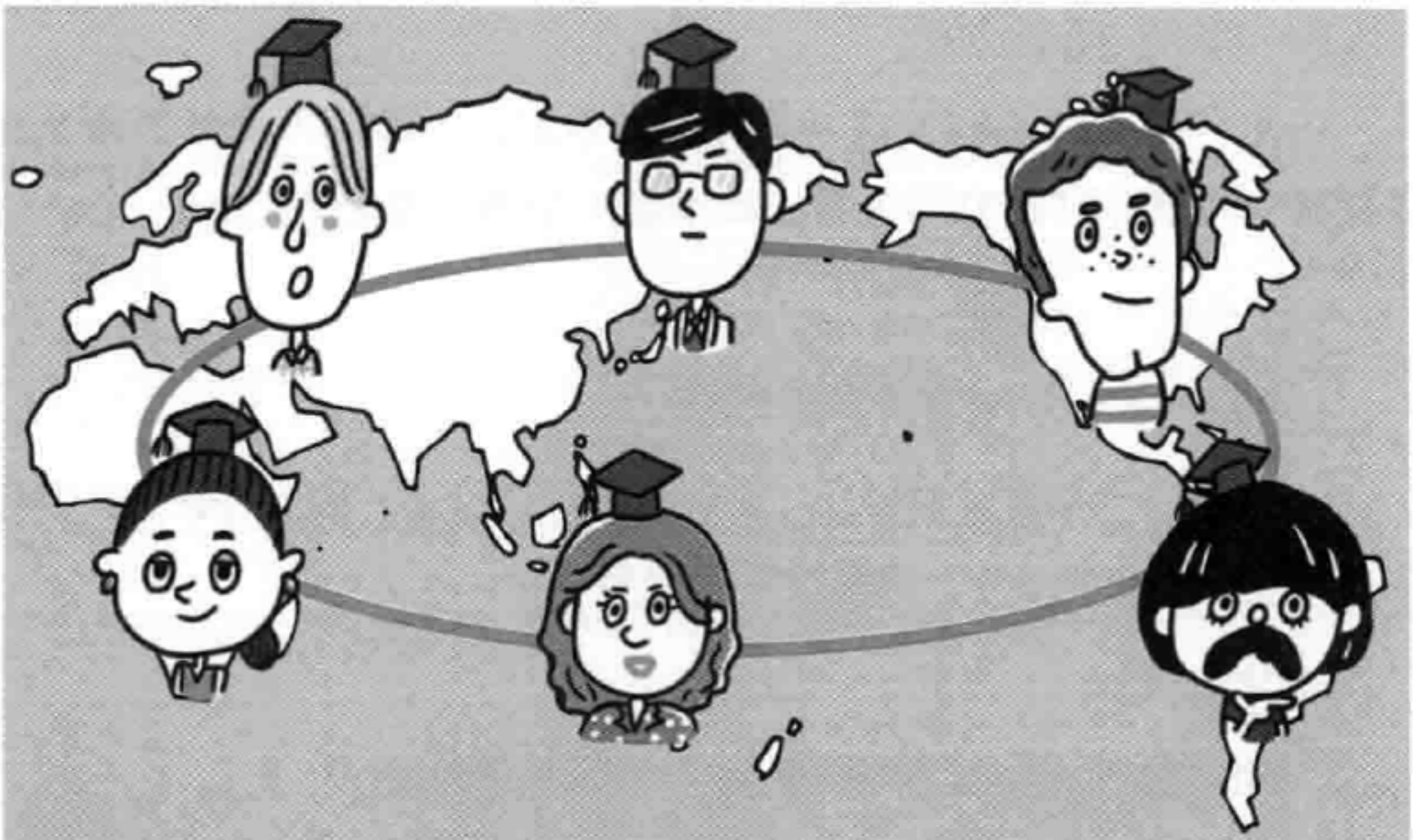
协议[①]）的协议作为规范，完成从客户端到服务器端等一系列运作流程。而协议是指规则的约定。可以说，Web 是建立在 HTTP 协议上通信的。

1.2　HTTP 的诞生

在深入学习 HTTP 之前，我们先来介绍一下 HTTP 诞生的背景。了解背景的同时也能了解当初制定 HTTP 的初衷，这样有助于我们更好地理解。

1.2.1　为知识共享而规划 Web

1989 年 3 月，互联网还只属于少数人。在这一互联网的黎明期，HTTP 诞生了。



WWW这一提议是致力于全世界的研究者们进行知识共享。

① HTTP通常被译为超文本传输协议，但这种译法并不严谨。严谨的译名应该为“超文本转移协议”。但是前一译法已约定俗成，本书将会沿用。有兴趣的读者可参考图灵社区的相关讨论：http://www.ituring.com.cn/article/1817。——译者注

CERN（欧洲核子研究组织）的蒂姆·伯纳斯－李（Tim Berners-Lee）博士提出了一种能让远隔两地的研究者们共享知识的设想。

最初设想的基本理念是：借助多文档之间相互关联形成的超文本（HyperText），连成可相互参阅的 WWW（World Wide Web，万维网）。

现在已提出了 3 项 WWW 构建技术，分别是：把 SGML（Standard Generalized Markup Language，标准通用标记语言）作为页面的文本标记语言的 HTML（HyperText Markup Language，超文本标记语言）；作为文档传递协议的 HTTP；指定文档所在地址的 URL（Uniform Resource Locator，统一资源定位符）。

WWW 这一名称，是 Web 浏览器当年用来浏览超文本的客户端应用程序时的名称。现在则用来表示这一系列的集合，也可简称为 Web。

1.2.2 Web 成长时代

1990 年 11 月，CERN 成功研发了世界上第一台 Web 服务器和 Web 浏览器。两年后的 1992 年 9 月，日本第一个网站的主页上线了。

- 日本第一个主页

  http://www.ibarakiken.gr.jp/www/

1990 年，大家针对 HTML 1.0 草案进行了讨论，因 HTML 1.0 中存在多处模糊不清的部分，草案被直接废弃了。

- HTML1.0

  http://www.w3.org/MarkUp/draft-ietf-iiir-html-01.txt

1993 年 1 月，现代浏览器的祖先 NCSA（National Center for Supercomputer Applications，美国国家超级计算机应用中心）研发的 Mosaic 问世了。它以 in-line（内联）等形式显示 HTML 的图像，在图像方面出色的表现使它迅速在世界范围内流行开来。

同年秋天，Mosaic 的 Windows 版和 Macintosh 版面世。使用 CGI 技术的 NCSA Web 服务器、NCSA HTTPd 1.0 也差不多是在这个时期出现的。

- NCSA Mosaic bounce page

  http://archive.ncsa.illinois.edu/mosaic.html

- The NCSA HTTPd Home Page（存档）

  http://web.archive.org/web/20090426182129/http://hoohoo.ncsa.illinois.edu/（原址已失效）

1994 年的 12 月，网景通信公司发布了 Netscape Navigator 1.0，1995 年微软公司发布 Internet Explorer 1.0 和 2.0。

紧随其后的是现在已然成为 Web 服务器标准之一的 Apache，当时它以 Apache 0.2 的姿态出现在世人眼前。而 HTML 也发布了 2.0 版本。那一年，Web 技术的发展突飞猛进。

时光流转，从 1995 年左右起，微软公司与网景通信公司之间爆发的浏览器大战愈演愈烈。两家公司都各自对 HTML 做了扩展，于是导致在写 HTML 页面时，必须考虑兼容他们两家公司的浏览器。时至今日，这个问题仍令那些写前端页面的工程师感到棘手。

在这场浏览器供应商之间的竞争中，他们不仅对当时发展中的各种 Web 标准化视而不见，还屡次出现新增功能没有对应说明文档的情况。

2000 年前后，这场浏览器战争随着网景通信公司的衰落而暂告一段落。但就在 2004 年，Mozilla 基金会发布了 Firefox 浏览器，第二次浏览器大战随即爆发。

Internet Explorer 浏览器的版本从 6 升到 7 前后花费了 5 年时间。之后接连不断地发布了 8、9、10 版本。另外，Chrome、Opera、Safari 等浏览器也纷纷抢占市场份额。

1.2.3　驻足不前的 HTTP

HTTP/0.9

HTTP 于 1990 年问世。那时的 HTTP 并没有作为正式的标准被建立。现在的 HTTP 其实含有 HTTP1.0 之前版本的意思，因此被称为 HTTP/0.9。

HTTP/1.0

HTTP 正式作为标准被公布是在 1996 年的 5 月，版本被命名为 HTTP/1.0，并记载于 RFC1945。虽说是初期标准，但该协议标准至今仍被广泛使用在服务器端。

- RFC1945 - Hypertext Transfer Protocol -- HTTP/1.0

  http://www.ietf.org/rfc/rfc1945.txt

HTTP/1.1

1997 年 1 月公布的 HTTP/1.1 是目前主流的 HTTP 协议版本。当初的标准是 RFC2068，之后发布的修订版 RFC2616 就是当前的最新版本。

- RFC2616 - Hypertext Transfer Protocol -- HTTP/1.1

  http://www.ietf.org/rfc/rfc2616.txt

可见，作为 Web 文档传输协议的 HTTP，它的版本几乎没有更新。新一代 HTTP/2.0 正在制订中，但要达到较高的使用覆盖率，仍需假以时日。

当年 HTTP 协议的出现主要是为了解决文本传输的难题。由于协议本身非常简单，于是在此基础上设想了很多应用方法并投入了实际使用。现在 HTTP 协议已经超出了 Web 这个框架的局限，被运用到了各种场景里。

1.3 网络基础 TCP/IP

为了理解 HTTP，我们有必要事先了解一下 TCP/IP 协议族。

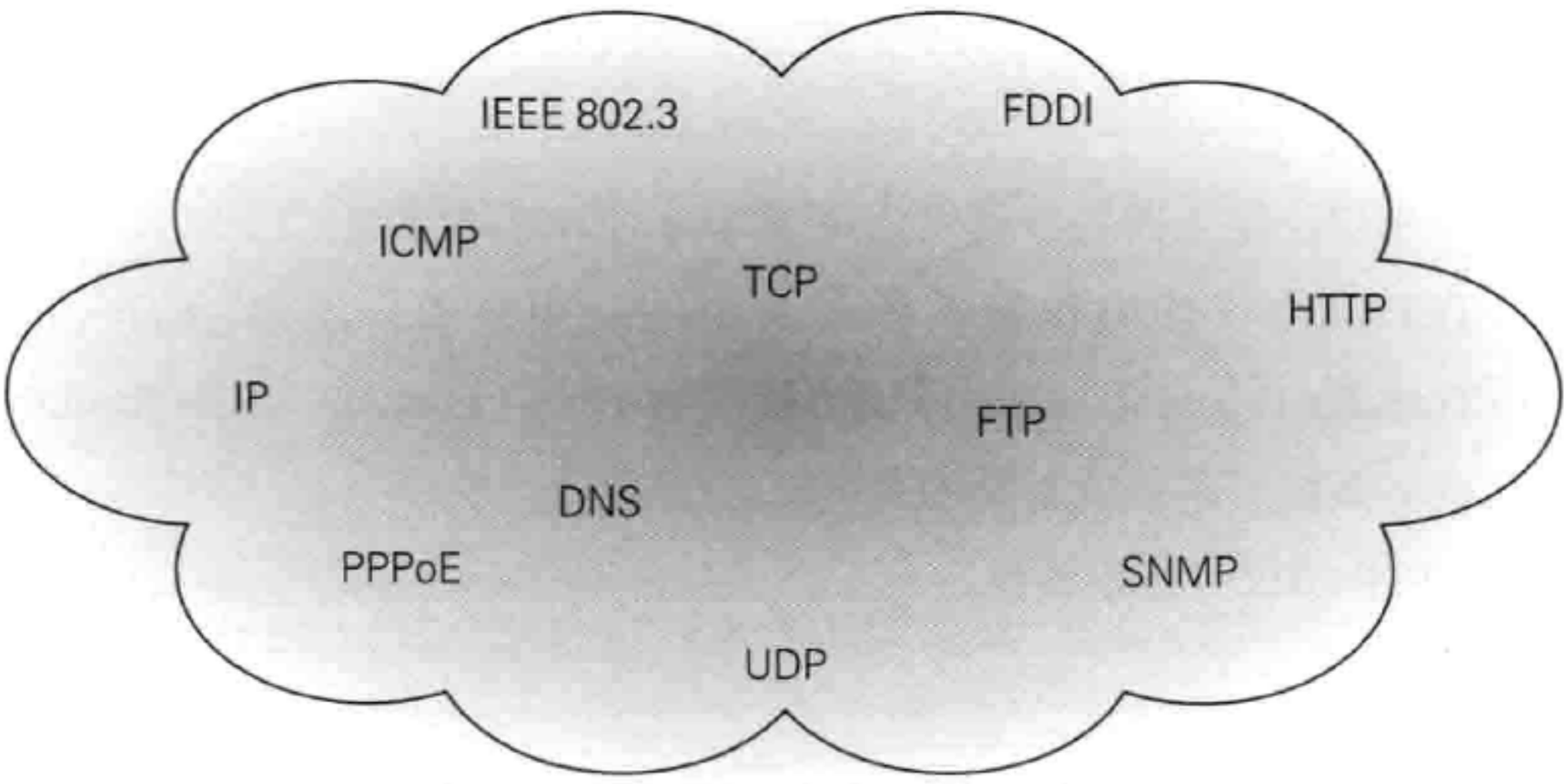
通常使用的网络（包括互联网）是在 TCP/IP 协议族的基础上运作的。而 HTTP 属于它内部的一个子集。

接下来，我们仅介绍理解 HTTP 所需掌握的 TCP/IP 协议族的概要。若想进一步学习有关 TCP/IP 的知识，请参考其他讲解 TCP/IP 的专业书籍。

1.3.1 TCP/IP 协议族

计算机与网络设备要相互通信，双方就必须基于相同的方法。比

如，如何探测到通信目标、由哪一边先发起通信、使用哪种语言进行通信、怎样结束通信等规则都需要事先确定。不同的硬件、操作系统之间的通信，所有的这一切都需要一种规则。而我们就把这种规则称为协议（protocol）。



图：TCP/IP 是互联网相关的各类协议族的总称

协议中存在各式各样的内容。从电缆的规格到 IP 地址的选定方法、寻找异地用户的方法、双方建立通信的顺序，以及 Web 页面显示需要处理的步骤，等等。

像这样把与互联网相关联的协议集合起来总称为 TCP/IP。也有说法认为，TCP/IP 是指 TCP 和 IP 这两种协议。还有一种说法认为，TCP/IP 是在 IP 协议的通信过程中，使用到的协议族的统称。

1.3.2　TCP/IP 的分层管理

TCP/IP 协议族里重要的一点就是分层。TCP/IP 协议族按层次分别分为以下 4 层：应用层、传输层、网络层和数据链路层。

把 TCP/IP 层次化是有好处的。比如，如果互联网只由一个协议统筹，某个地方需要改变设计时，就必须把所有部分整体替换掉。而分层之后只需把变动的层替换掉即可。把各层之间的接口部分规划好之后，

每个层次内部的设计就能够自由改动了。

值得一提的是，层次化之后，设计也变得相对简单了。处于应用层上的应用可以只考虑分派给自己的任务，而不需要弄清对方在地球上哪个地方、对方的传输路线是怎样的、是否能确保传输送达等问题。

TCP/IP 协议族各层的作用如下。

应用层

应用层决定了向用户提供应用服务时通信的活动。

TCP/IP 协议族内预存了各类通用的应用服务。比如，FTP（File Transfer Protocol，文件传输协议）和 DNS（Domain Name System，域名系统）服务就是其中两类。

HTTP 协议也处于该层。

传输层

传输层对上层应用层，提供处于网络连接中的两台计算机之间的数据传输。

在传输层有两个性质不同的协议：TCP（Transmission Control Protocol，传输控制协议）和 UDP（User Data Protocol，用户数据报协议）。

网络层（又名网络互连层）

网络层用来处理在网络上流动的数据包。数据包是网络传输的最小数据单位。该层规定了通过怎样的路径（所谓的传输路线）到达对方计算机，并把数据包传送给对方。

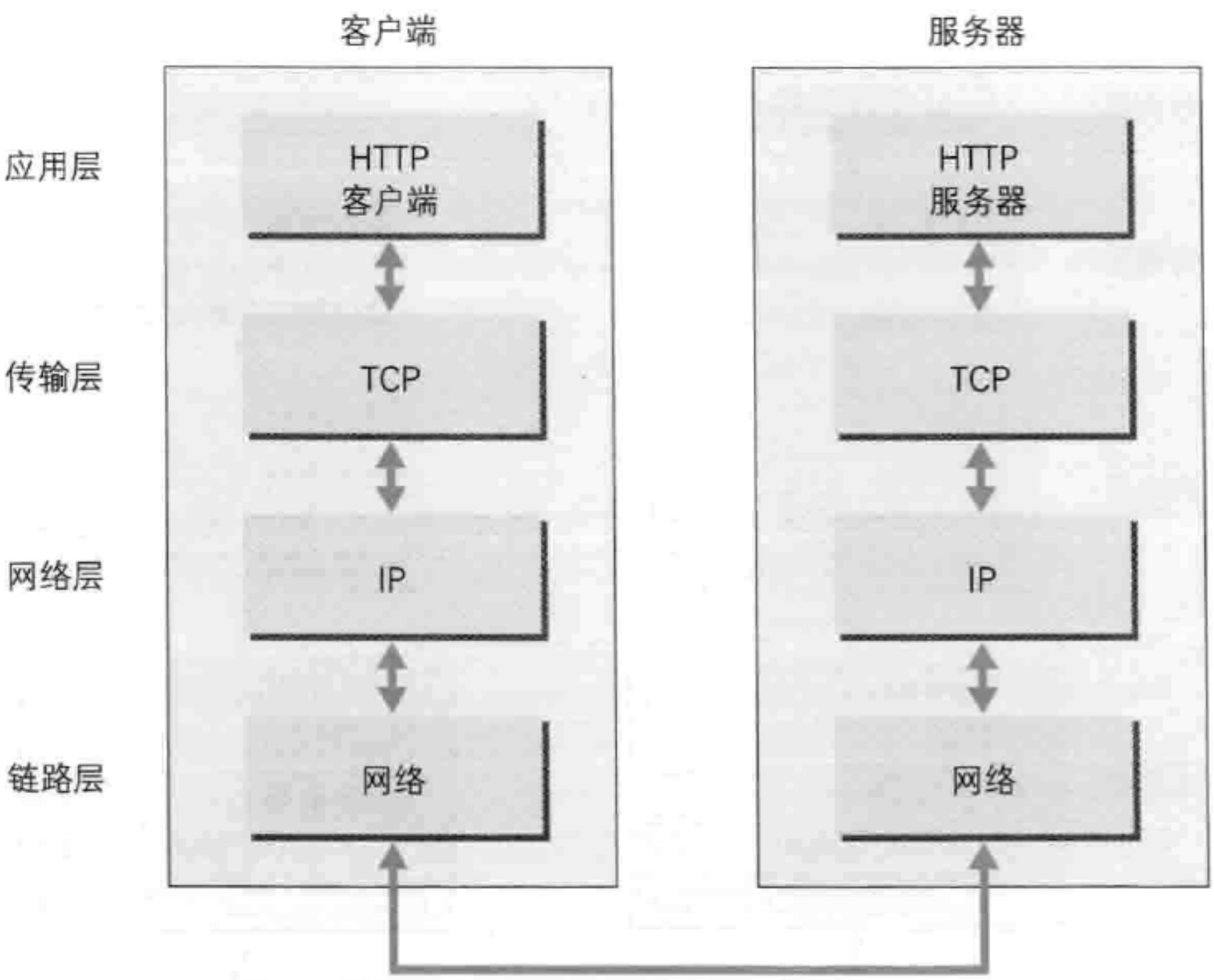
与对方计算机之间通过多台计算机或网络设备进行传输时，网络层所起的作用就是在众多的选项内选择一条传输路线。

链路层（又名数据链路层，网络接口层）

用来处理连接网络的硬件部分。包括控制操作系统、硬件的设备驱动、NIC（Network Interface Card，网络适配器，即网卡），及光纤

等物理可见部分（还包括连接器等一切传输媒介）。硬件上的范畴均在链路层的作用范围之内。

1.3.3　TCP/IP 通信传输流



利用 TCP/IP 协议族进行网络通信时，会通过分层顺序与对方进行通信。发送端从应用层往下走，接收端则往应用层往上走。

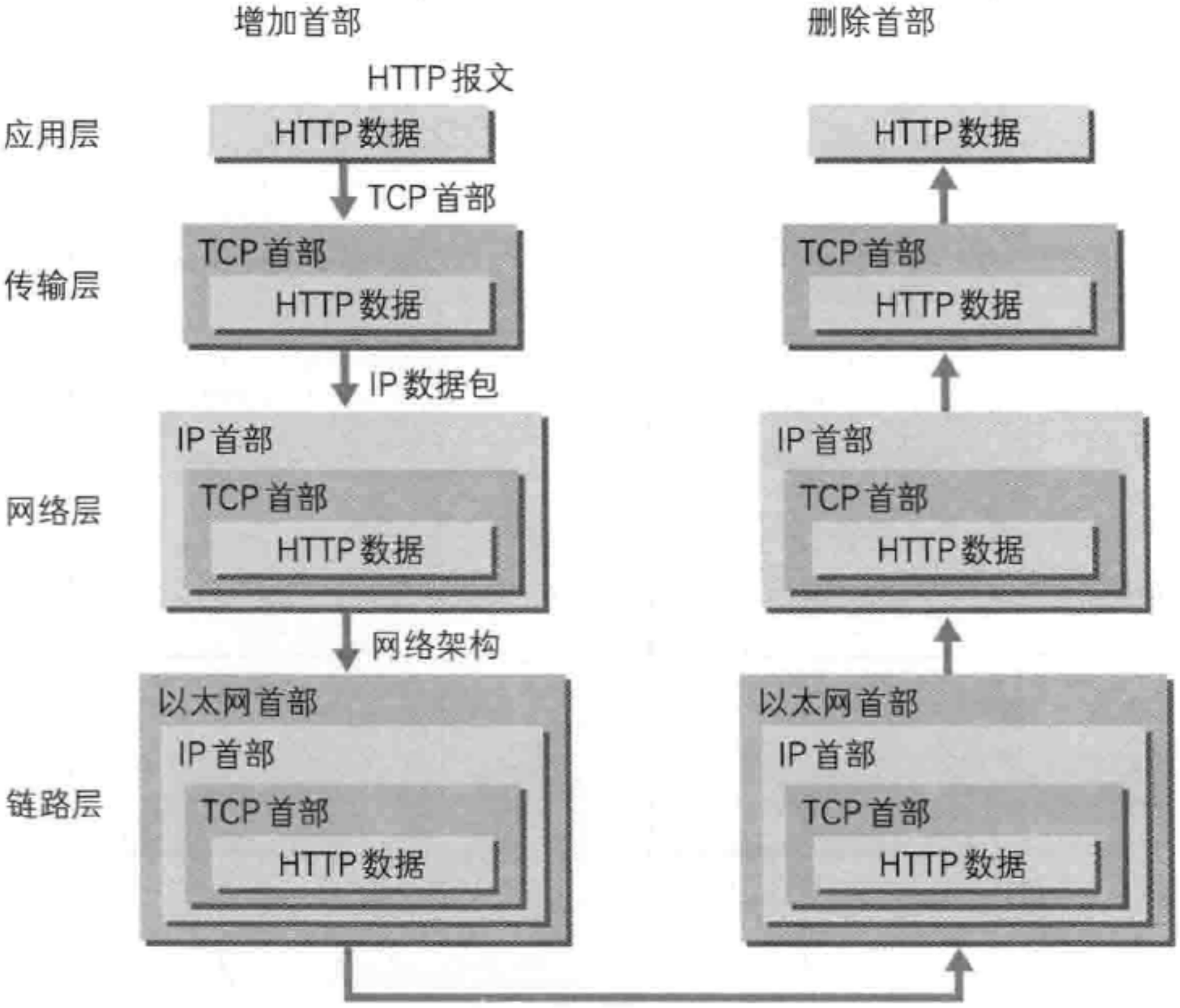
我们用 HTTP 举例来说明，首先作为发送端的客户端在应用层（HTTP 协议）发出一个想看某个 Web 页面的 HTTP 请求。

接着，为了传输方便，在传输层（TCP 协议）把从应用层处收到的数据（HTTP 请求报文）进行分割，并在各个报文上打上标记序号及端口号后转发给网络层。

在网络层（IP 协议），增加作为通信目的地的 MAC 地址后转发给链路层。这样一来，发往网络的通信请求就准备齐全了。

接收端的服务器在链路层接收到数据，按序往上层发送，一直到应

用层。当传输到应用层，才能算真正接收到由客户端发送过来的 HTTP 请求。



发送端在层与层之间传输数据时，每经过一层时必定会被打上一个该层所属的首部信息。反之，接收端在层与层传输数据时，每经过一层时会把对应的首部消去。

这种把数据信息包装起来的做法称为封装（ encapsulate ）。

1.4 与 HTTP 关系密切的协议：IP、TCP 和 DNS

下面我们分别针对在 TCP/IP 协议族中与 HTTP 密不可分的 3 个协议（ IP、TCP 和 DNS ）进行说明。

1.4.1　负责传输的IP协议

按层次分，IP（Internet Protocol）网际协议位于网络层。Internet Protocol这个名称可能听起来有点夸张，但事实正是如此，因为几乎所有使用网络的系统都会用到IP协议。TCP/IP协议族中的IP指的就是网际协议，协议名称中占据了一半位置，其重要性可见一斑。可能有人会把“IP”和“IP地址”搞混，“IP”其实是一种协议的名称。

IP协议的作用是把各种数据包传送给对方。而要保证确实传送到对方那里，则需要满足各类条件。其中两个重要的条件是IP地址和MAC地址（Media Access Control Address）。

IP地址指明了节点被分配到的地址，MAC地址是指网卡所属的固定地址。IP地址可以和MAC地址进行配对。IP地址可变换，但MAC地址基本上不会更改。

使用ARP协议凭借MAC地址进行通信

IP间的通信依赖MAC地址。在网络上，通信的双方在同一局域网（LAN）内的情况是很少的，通常是经过多台计算机和网络设备中转才能连接到对方。而在进行中转时，会利用下一站中转设备的MAC地址来搜索下一个中转目标。这时，会采用ARP协议（Address Resolution Protocol）。ARP是一种用以解析地址的协议，根据通信方的IP地址就可以反查出对应的MAC地址。

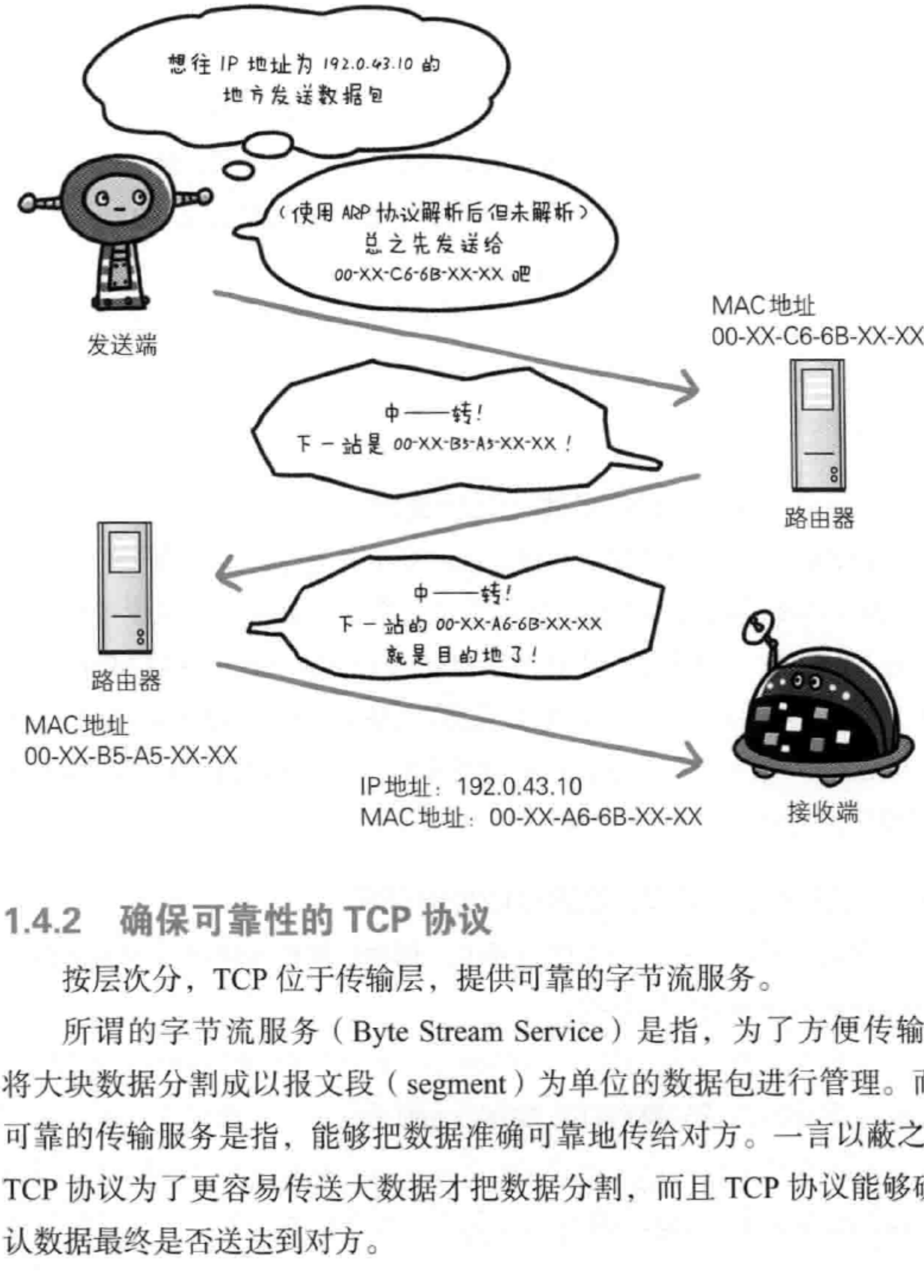
没有人能够全面掌握互联网中的传输状况

在到达通信目标前的中转过程中，那些计算机和路由器等网络设备只能获悉很粗略的传输路线。

这种机制称为路由选择（routing），有点像快递公司的送货过程。想要寄快递的人，只要将自己的货物送到集散中心，就可以知道快递公司是否肯收件发货，该快递公司的集散中心检查货物的送达地址，明确下站该送往哪个区域的集散中心。接着，那个区域的集散中心自会判断

是否能送到对方的家中。

我们是想通过这个比喻说明，无论哪台计算机、哪台网络设备，它们都无法全面掌握互联网中的细节。



1.4.2 确保可靠性的 TCP 协议

按层次分，TCP 位于传输层，提供可靠的字节流服务。

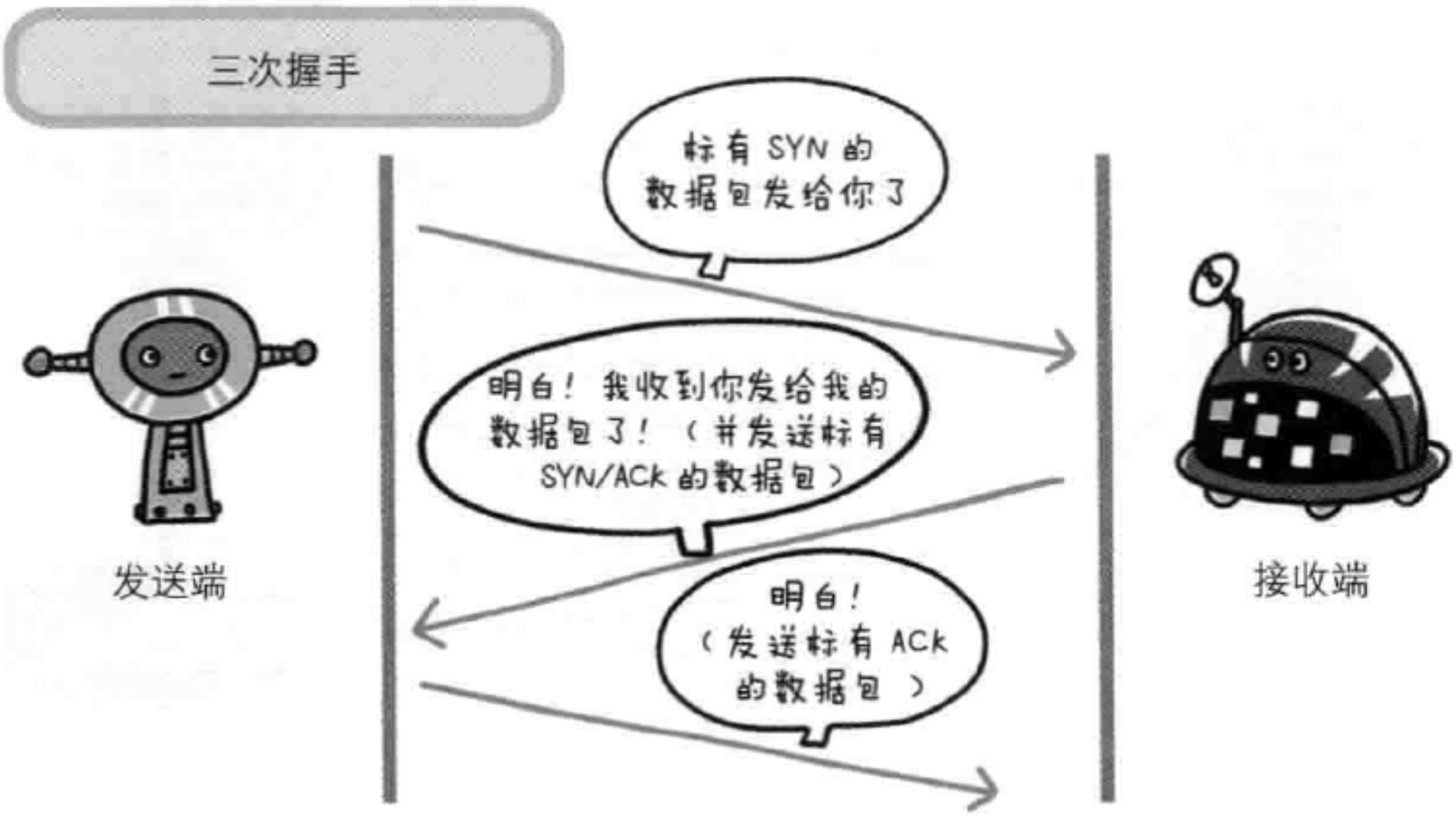
所谓的字节流服务（Byte Stream Service）是指，为了方便传输，将大块数据分割成以报文段（segment）为单位的数据包进行管理。而可靠的传输服务是指，能够把数据准确可靠地传给对方。一言以蔽之，TCP 协议为了更容易传送大数据才把数据分割，而且 TCP 协议能够确认数据最终是否送达到对方。

确保数据能到达目标

为了准确无误地将数据送达目标处，TCP 协议采用了三次握手（three-way handshaking）策略。用 TCP 协议把数据包送出去后，TCP 不会对传送后的情况置之不理，它一定会向对方确认是否成功送达。握手过程中使用了 TCP 的标志（flag）——SYN（synchronize）和 ACK（acknowledgement）。

发送端首先发送一个带 SYN 标志的数据包给对方。接收端收到后，回传一个带有 SYN/ACK 标志的数据包以示传达确认信息。最后，发送端再回传一个带 ACK 标志的数据包，代表“握手”结束。

若在握手过程中某个阶段莫名中断，TCP 协议会再次以相同的顺序发送相同的数据包。



除了上述三次握手，TCP 协议还有其他各种手段来保证通信的可靠性。

1.5　负责域名解析的 DNS 服务

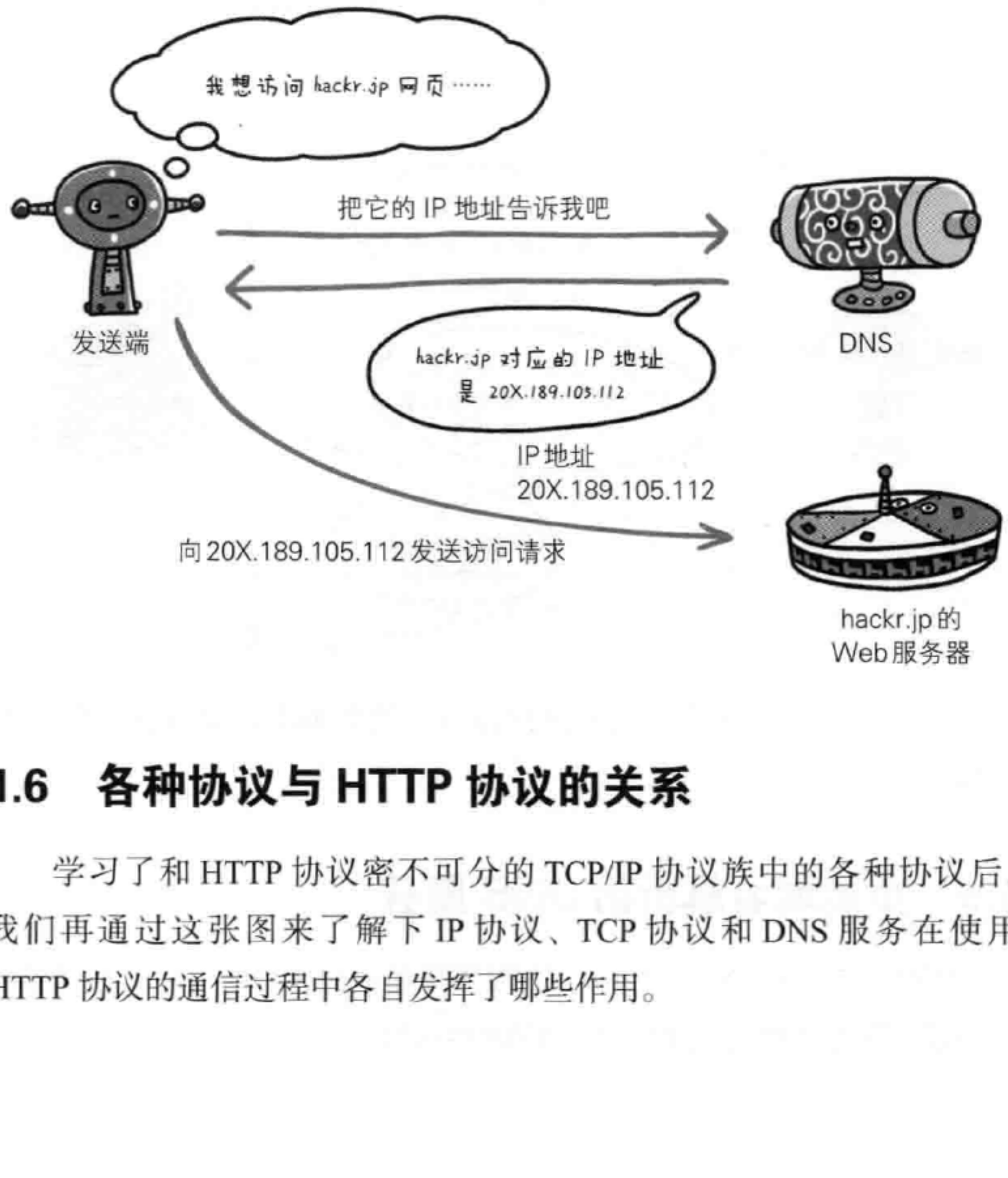
DNS（Domain Name System）服务是和 HTTP 协议一样位于应用层的协议。它提供域名到 IP 地址之间的解析服务。

计算机既可以被赋予 IP 地址，也可以被赋予主机名和域名。比如 www.hackr.jp。

用户通常使用主机名或域名来访问对方的计算机，而不是直接通过 IP 地址访问。因为与 IP 地址的一组纯数字相比，用字母配合数字的表示形式来指定计算机名更符合人类的记忆习惯。

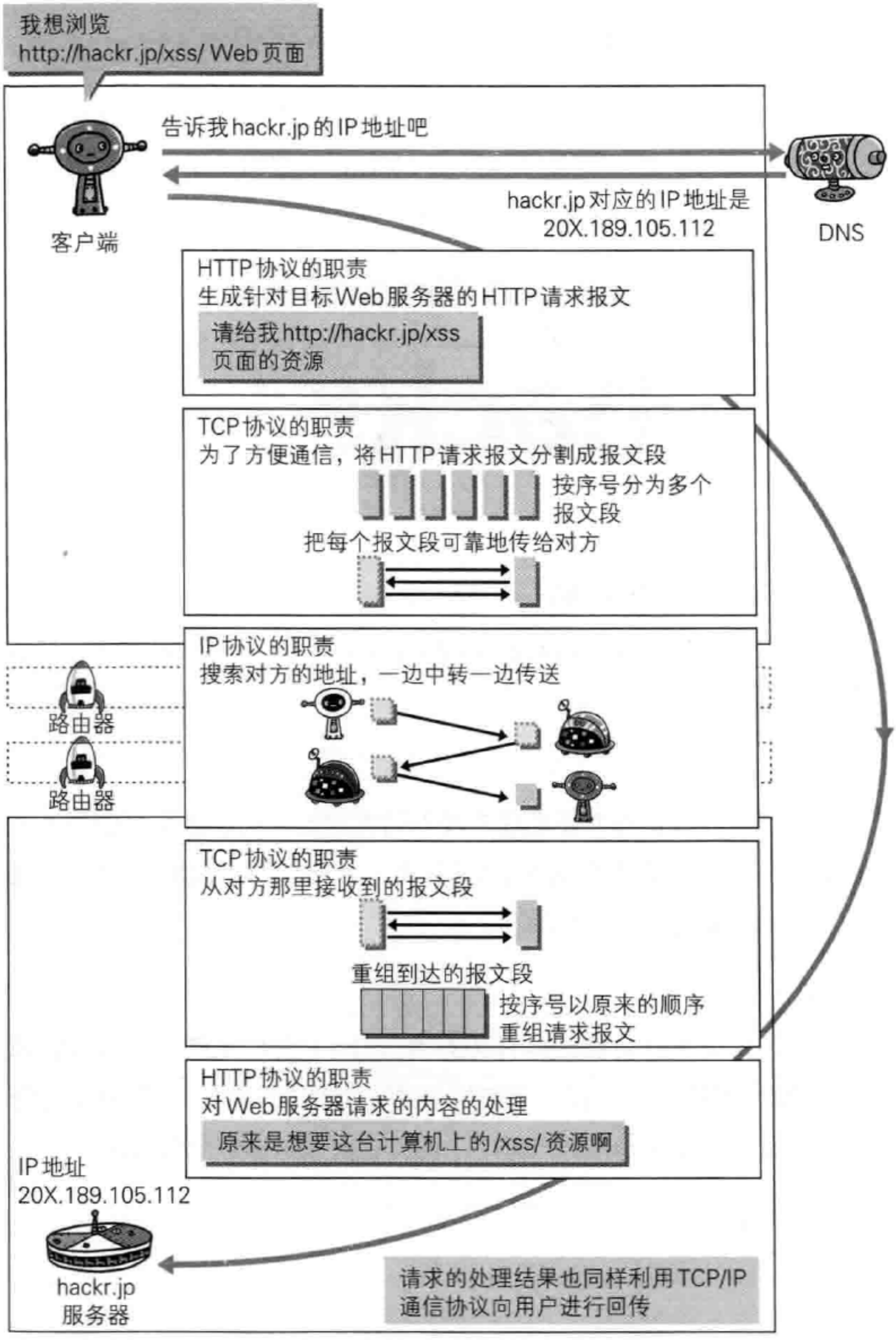
但要让计算机去理解名称，相对而言就变得困难了。因为计算机更擅长处理一长串数字。

为了解决上述的问题，DNS 服务应运而生。DNS 协议提供通过域名查找 IP 地址，或逆向从 IP 地址反查域名的服务。



1.6 各种协议与 HTTP 协议的关系

学习了和 HTTP 协议密不可分的 TCP/IP 协议族中的各种协议后，我们再通过这张图来了解下 IP 协议、TCP 协议和 DNS 服务在使用 HTTP 协议的通信过程中各自发挥了哪些作用。


我想浏览
http://hackr.jp/xss/ Web页面
客户端
告诉我hackr.jp的IP地址吧
hackr.jp对应的IP地址是
20X.189.105.112
DNS
HTTP协议的职责
生成针对目标Web服务器的HTTP请求报文
请给我http://hackr.jp/xss
页面的资源
TCP协议的职责
为了方便通信，将HTTP请求报文分割成报文段
按序号分为多个
报文段
把每个报文段可靠地传给对方
路由器
路由器
IP协议的职责
搜索对方的地址，一边中转一边传送
TCP协议的职责
从对方那里接收到的报文段
重组到达的报文段
按序号以原来的顺序
重组请求报文
HTTP协议的职责
对Web服务器请求的内容的处理
原来是想要这台计算机上的/xss/资源啊
IP地址
20X.189.105.112
hackr.jp
服务器
请求的处理结果也同样利用TCP/IP
通信协议向用户进行回传

1.7 URI 和 URL

与 URI（统一资源标识符）相比，我们更熟悉 URL（Uniform Resource Locator，统一资源定位符）。URL 正是使用 Web 浏览器等访问 Web 页面时需要输入的网页地址。比如，下图的 http://hackr.jp/ 就是 URL。



1.7.1 统一资源标识符

URI 是 Uniform Resource Identifier 的缩写。RFC2396 分别对这 3 个单词进行了如下定义。

Uniform

规定统一的格式可方便处理多种不同类型的资源，而不用根据上下文环境来识别资源指定的访问方式。另外，加入新增的协议方案（如 http: 或 ftp:）也更容易。

Resource

资源的定义是“可标识的任何东西”。除了文档文件、图像或服务（例如当天的天气预报）等能够区别于其他类型的，全都可作为资源。另外，资源不仅可以是单一的，也可以是多数的集合体。

Identifier

表示可标识的对象。也称为标识符。

综上所述，URI 就是由某个协议方案表示的资源的定位标识符。协

议方案是指访问资源所使用的协议类型名称。

采用HTTP协议时，协议方案就是http。除此之外，还有ftp、mailto、telnet、file等。标准的URI协议方案有30种左右，由隶属于国际互联网资源管理的非营利社团ICANN（Internet Corporation for Assigned Names and Numbers，互联网名称与数字地址分配机构）的IANA（Internet Assigned Numbers Authority，互联网号码分配局）管理颁布。

- IANA - Uniform Resource Identifier (URI) SCHEMES（统一资源标识符方案）

  http://www.iana.org/assignments/uri-schemes

URI用字符串标识某一互联网资源，而URL表示资源的地点（互联网上所处的位置）。可见URL是URI的子集。

“RFC3986：统一资源标识符（URI）通用语法”中列举了几种URI例子，如下所示。


```
ftp://ftp.is.co.za/rfc/rfc1808.txt
http://www.ietf.org/rfc/rfc2396.txt
ldap://[2001:db8::7]/c=GB?objectClass?one
mailto:John.Doe@example.com
news:comp.infosystems.www.servers.unix
tel:+1-816-555-1212
telnet://192.0.2.16:80/
urn:oasis:names:specification:docbook:dtd:xml:4.1.2
```

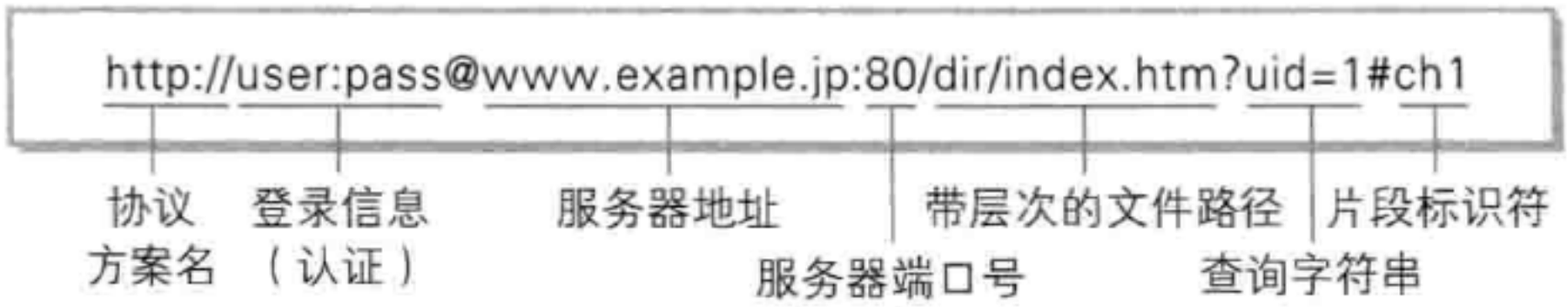
本书接下来的章节中会频繁出现URI这个术语，在充分理解的基础上，也可用URL替换URI。

1.7.2　URI格式

表示指定的URI，要使用涵盖全部必要信息的绝对URI、绝对URL以及相对URL。相对URL，是指从浏览器中基本URI处指定的URL，

形如 /image/logo.gif。

让我们先来了解一下绝对 URI 的格式。



使用 http: 或 https: 等协议方案名获取访问资源时要指定协议类型。不区分字母大小写，最后附一个冒号（:）。

也可使用 data: 或 javascript: 这类指定数据或脚本程序的方案名。

登录信息（认证）

指定用户名和密码作为从服务器端获取资源时必要的登录信息（身份认证）。此项是可选项。

服务器地址

使用绝对 URI 必须指定待访问的服务器地址。地址可以是类似 hackr.jp 这种 DNS 可解析的名称，或是 192.168.1.1 这类 IPv4 地址名，还可以是 [0:0:0:0:0:0:0:1] 这样用方括号括起来的 IPv6 地址名。

服务器端口号

指定服务器连接的网络端口号。此项也是可选项，若用户省略则自动使用默认端口号。

带层次的文件路径

指定服务器上的文件路径来定位特指的资源。这与 UNIX 系统的文件目录结构相似。

查询字符串

针对已指定的文件路径内的资源，可以使用查询字符串传入任意参数。此项可选。

片段标识符

使用片段标识符通常可标记出已获取资源中的子资源（文档内的某个位置）。但在 RFC 中并没有明确规定其使用方法。该项也为可选项。

并不是所有的应用程序都符合 RFC

有一些用来制定 HTTP 协议技术标准的文档，它们被称为 RFC（Request for Comments，征求修正意见书）。

通常，应用程序会遵照由 RFC 确定的标准实现。可以说，RFC 是互联网的设计文档，要是不按照 RFC 标准执行，就有可能导致无法通信的状况。比如，有一台 Web 服务器内的应用服务没有遵照 RFC 的标准实现，那 Web 浏览器就很可能无法访问这台服务器了。

由于不遵照 RFC 标准实现就无法进行 HTTP 协议通信，所以基本上客户端和服务器端都会以 RFC 为标准来实现 HTTP 协议。但也存在某些应用程序因客户端或服务器端的不同，而未遵照 RFC 标准，反而将自成一套的“标准”扩展的情况。

不按 RFC 标准来实现，当然也不必劳心费力让自己的“标准”符合其他所有的客户端和服务器端。但设想一下，如果这款应用程序的使用者非常多，那会发生什么情况？不难想象，其他的客户端或服务器端必然都不得不去配合它。

实际在互联网上，已经实现了 HTTP 协议的一些服务器端和客户端里就存在上述情况。说不定它们会与本书介绍的 HTTP 协议的实现情况不一样。

本书接下来要介绍的 HTTP 协议内容，除去部分例外，基本上都以 RFC 的标准为准。

Chapter2

第2章
简单的HTTP协议

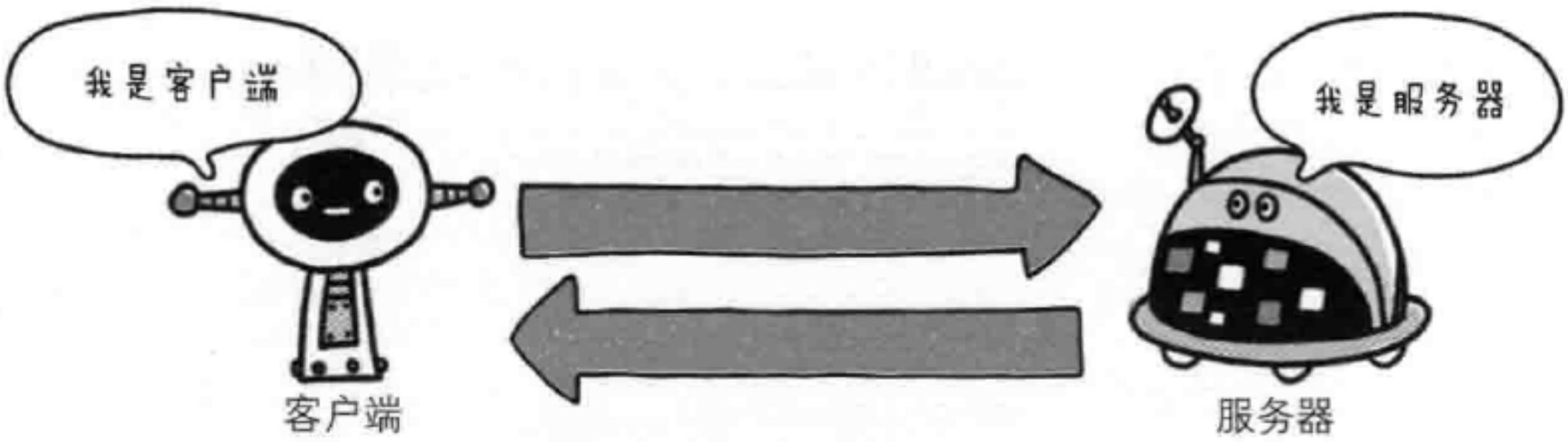
本章将针对HTTP协议结构进行讲解，主要使用HTTP/1.1版本。学完这章，想必大家就能理解HTTP协议的基础了。



2.1 HTTP 协议用于客户端和服务器端之间的通信

HTTP 协议和 TCP/IP 协议族内的其他众多的协议相同，用于客户端和服务器之间的通信。

请求访问文本或图像等资源的一端称为客户端，而提供资源响应的一端称为服务器端。

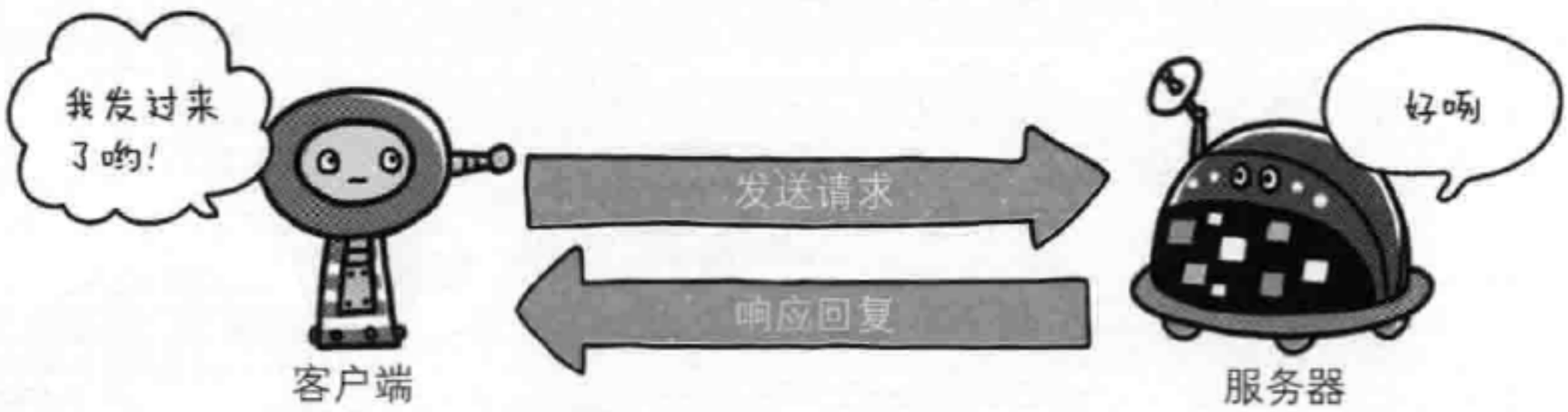


图：应用 HTTP 协议时，必定是一端担任客户端角色，另一端担任服务器端角色

在两台计算机之间使用 HTTP 协议通信时，在一条通信线路上必定有一端是客户端，另一端则是服务器端。

有时候，按实际情况，两台计算机作为客户端和服务器端的角色有可能会互换。但就仅从一条通信路线来说，服务器端和客户端的角色是确定的，而用 HTTP 协议能够明确区分哪端是客户端，哪端是服务器端。

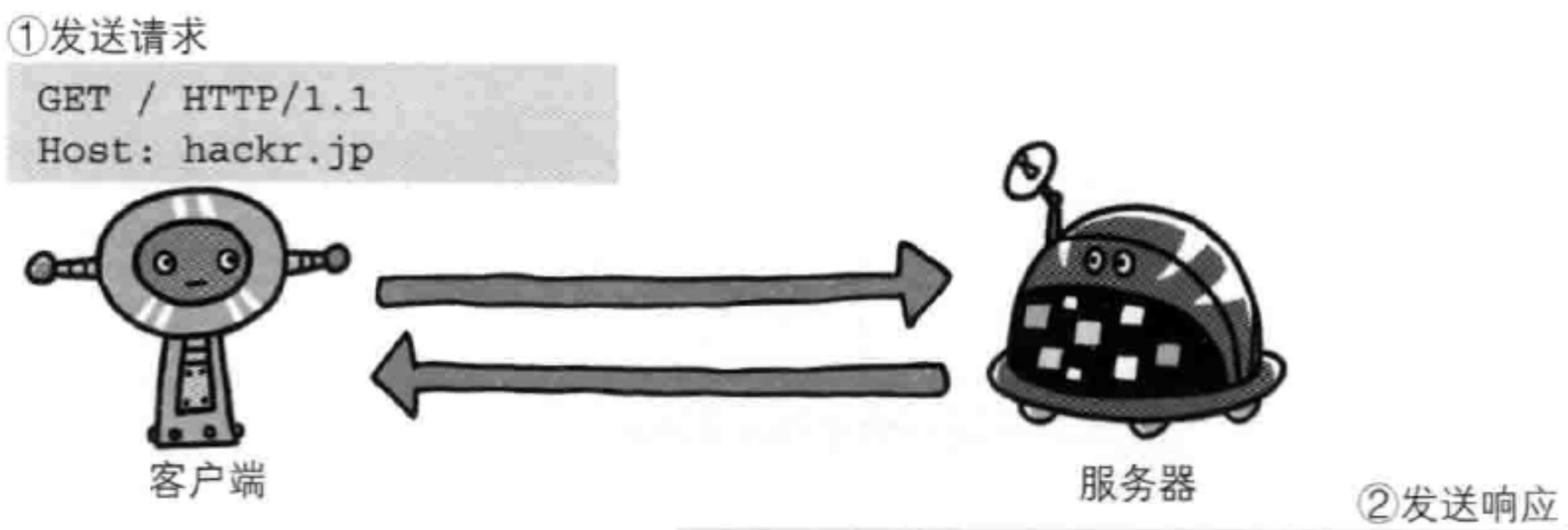
2.2 通过请求和响应的交换达成通信



图：请求必定由客户端发出，而服务器端回复响应

HTTP 协议规定，请求从客户端发出，最后服务器端响应该请求并返回。换句话说，肯定是先从客户端开始建立通信的，服务器端在没有接收到请求之前不会发送响应。

下面，我们来看一个具体的示例。



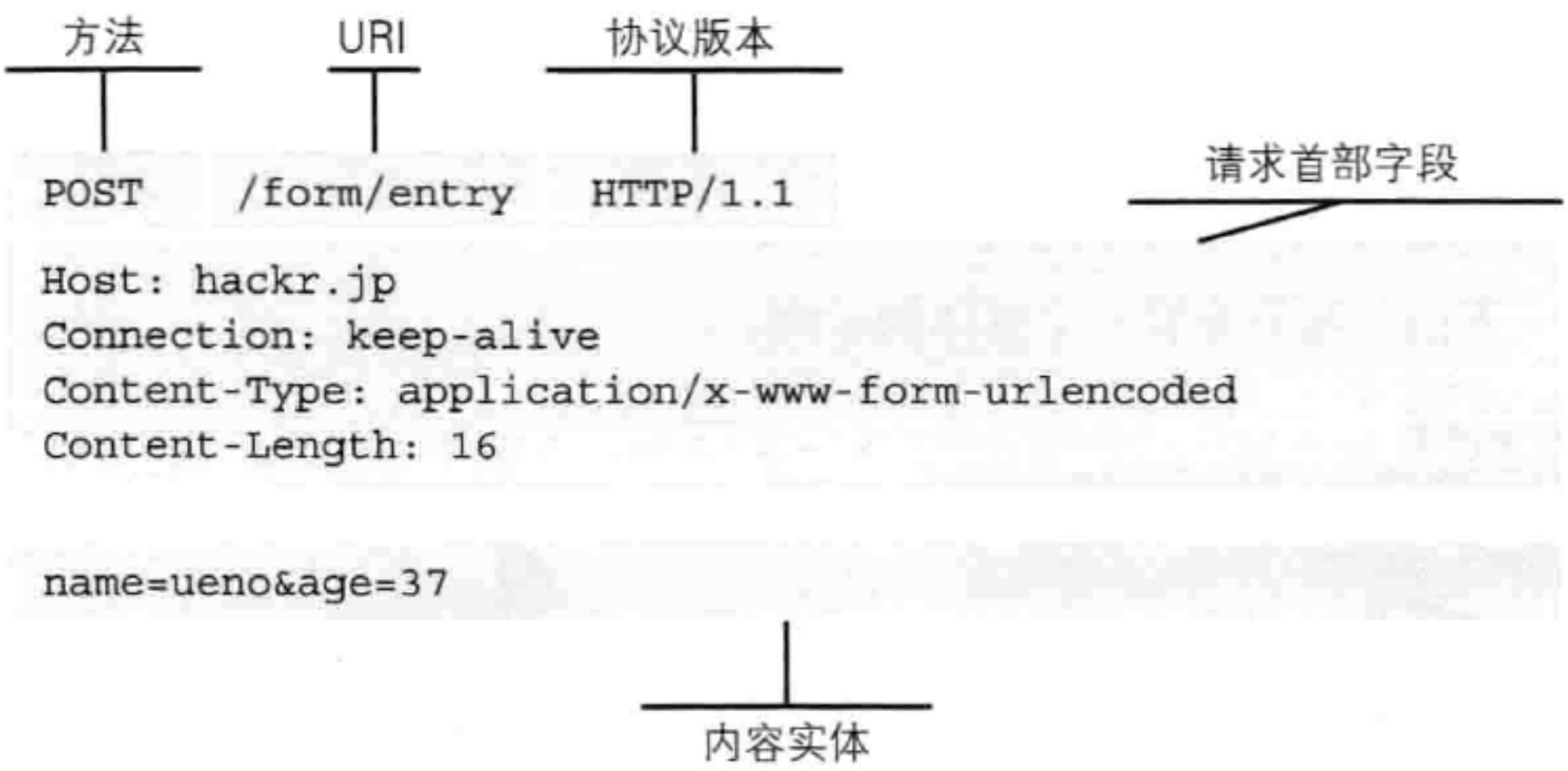
下面则是从客户端发送给某个 HTTP 服务器端的请求报文中的内容。

```
GET /index.htm HTTP/1.1
Host: hackr.jp
```

起始行开头的 GET 表示请求访问服务器的类型，称为方法（method）。随后的字符串 /index.htm 指明了请求访问的资源对象，也叫做请求 URI（request-URI）。最后的 HTTP/1.1，即 HTTP 的版本号，用来提示客户端使用的 HTTP 协议功能。

综合来看，这段请求内容的意思是：请求访问某台 HTTP 服务器上的 /index.htm 页面资源。

请求报文是由请求方法、请求 URI、协议版本、可选的请求首部字段和内容实体构成的。



图：请求报文的构成

请求首部字段及内容实体稍后会作详细说明。接下来，我们继续讲解。接收到请求的服务器，会将请求内容的处理结果以响应的形式返回。


```
HTTP/1.1 200 OK
Date: Tue, 10 Jul 2012 06:50:15 GMT
Content-Length: 362
Content-Type: text/html

<html>
......
```

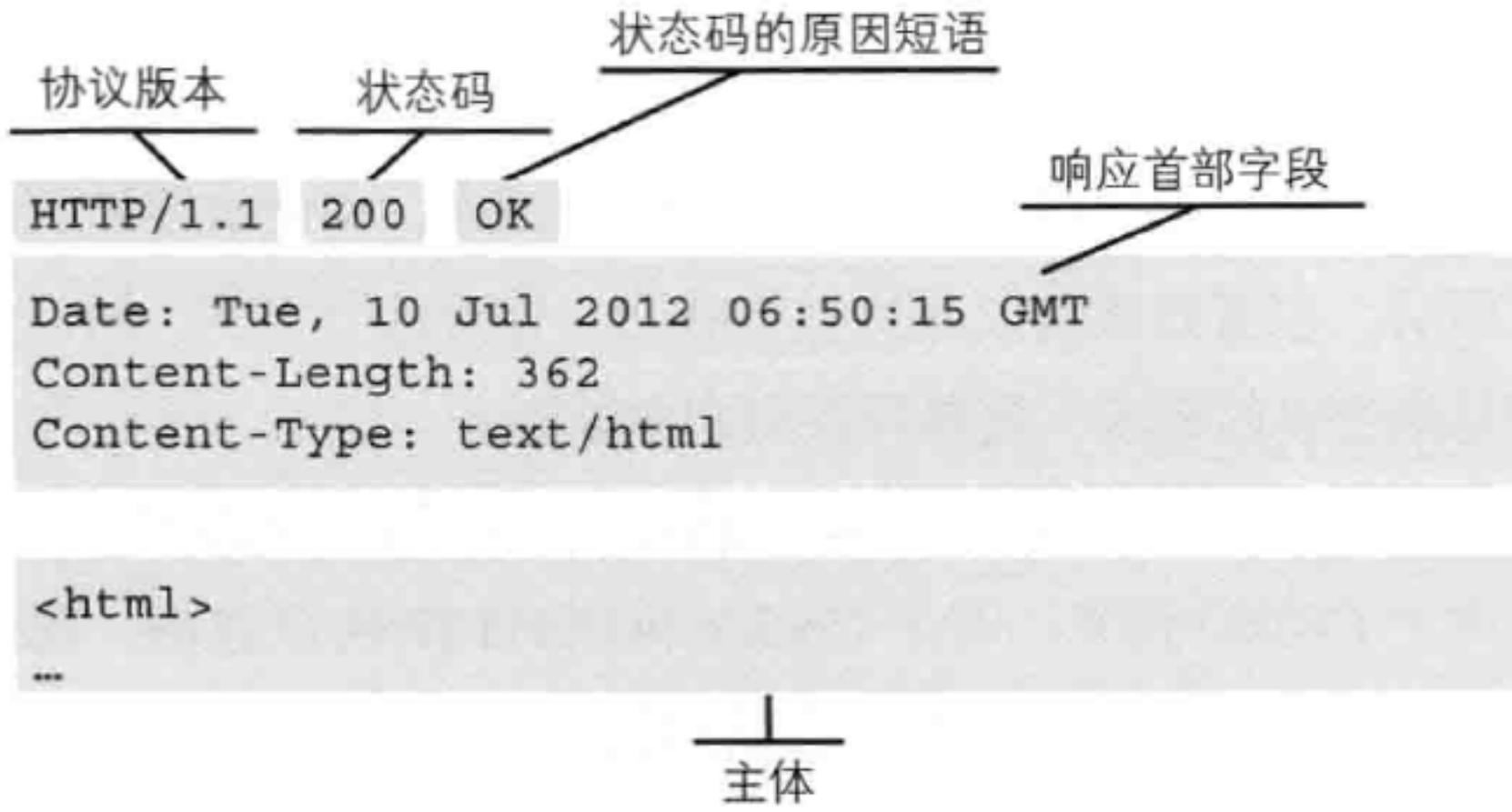
在起始行开头的 HTTP/1.1 表示服务器对应的 HTTP 版本。

紧挨着的 200 OK 表示请求的处理结果的状态码（ status code ）和原因短语（ reason-phrase ）。下一行显示了创建响应的日期时间，是首部字段（ header field ）内的一个属性。

接着以一空行分隔，之后的内容称为资源实体的主体（ entity body ）。

响应报文基本上由协议版本、状态码（表示请求成功或失败的数字代码）、用以解释状态码的原因短语、可选的响应首部字段以及实体主

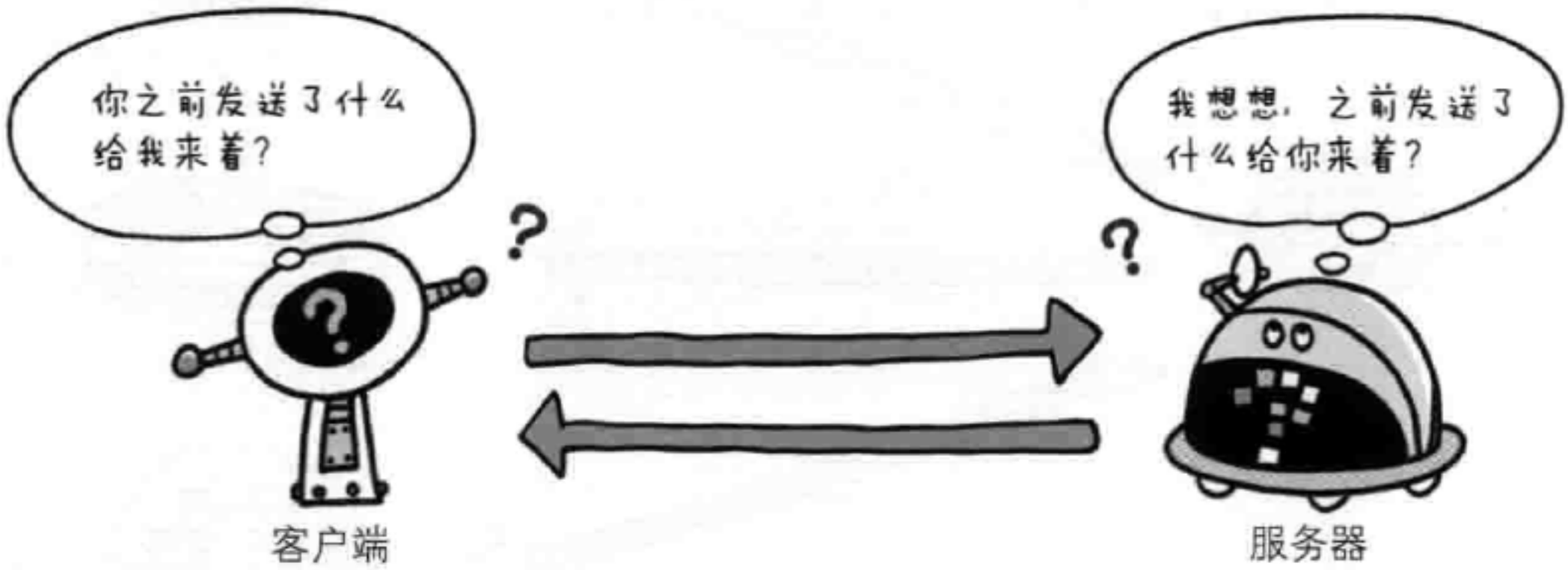
体构成。稍后我们会对这些内容进行详细说明。



图：响应报文的构成

2.3　HTTP 是不保存状态的协议

HTTP 是一种不保存状态，即无状态（stateless）协议。HTTP 协议自身不对请求和响应之间的通信状态进行保存。也就是说在 HTTP 这个级别，协议对于发送过的请求或响应都不做持久化处理。



图：HTTP 协议自身不具备保存之前发送过的请求或响应的功能

使用 HTTP 协议，每当有新的请求发送时，就会有对应的新响应产生。协议本身并不保留之前一切的请求或响应报文的信息。这是为了更

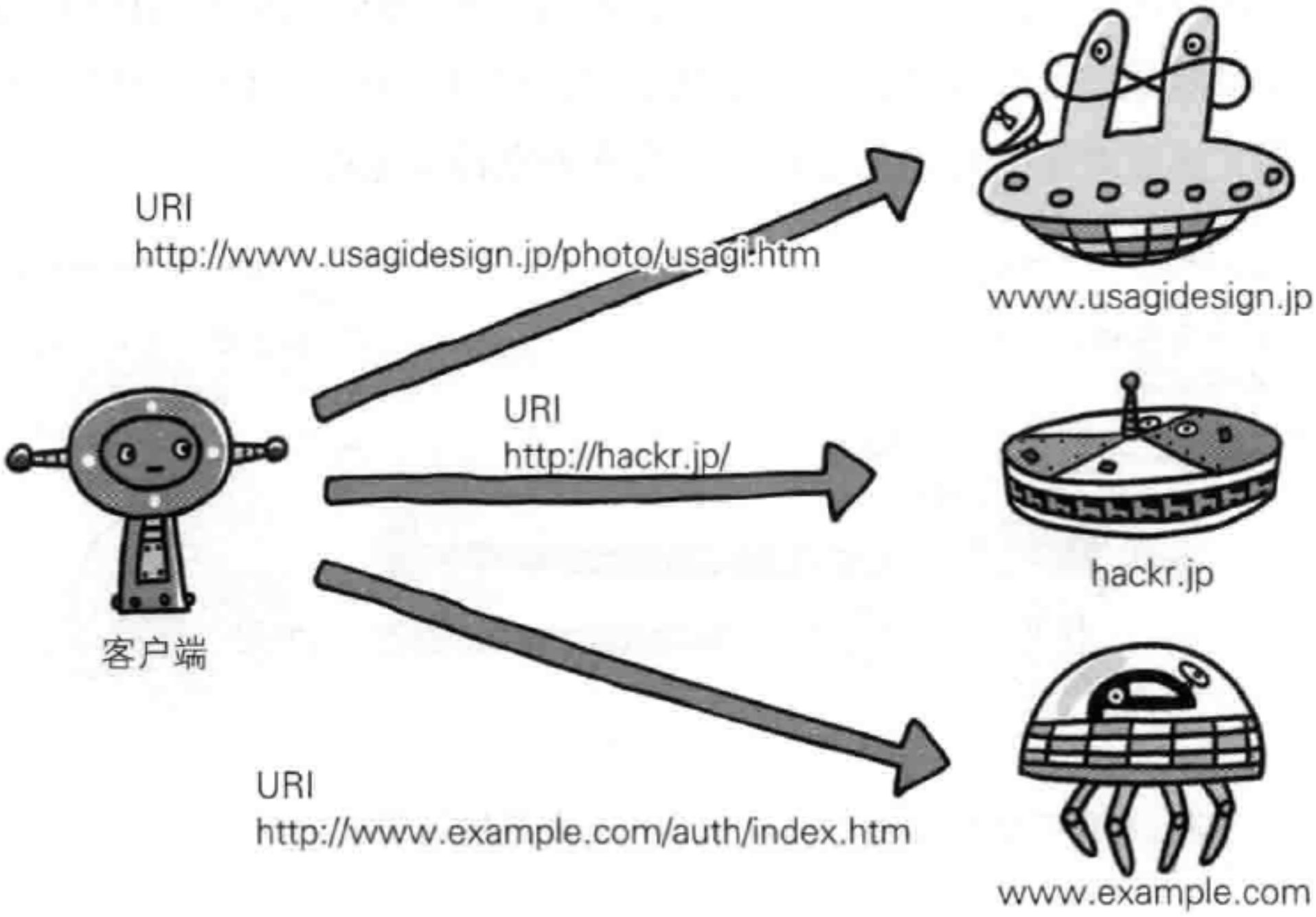
快地处理大量事务，确保协议的可伸缩性，而特意把 HTTP 协议设计成如此简单的。

可是，随着 Web 的不断发展，因无状态而导致业务处理变得棘手的情况增多了。比如，用户登录到一家购物网站，即使他跳转到该站的其他页面后，也需要能继续保持登录状态。针对这个实例，网站为了能够掌握是谁送出的请求，需要保存用户的状态。

HTTP/1.1 虽然是无状态协议，但为了实现期望的保持状态功能，于是引入了 Cookie 技术。有了 Cookie 再用 HTTP 协议通信，就可以管理状态了。有关 Cookie 的详细内容稍后讲解。

2.4 请求 URI 定位资源

HTTP 协议使用 URI 定位互联网上的资源。正是因为 URI 的特定功能，在互联网上任意位置的资源都能访问到。



图：HTTP 协议使用 URI 让客户端定位到资源

当客户端请求访问资源而发送请求时，URI需要将作为请求报文中的请求URI包含在内。指定请求URI的方式有很多。

- URI为完整的请求URI

```
GET http://hackr.jp/index.htm HTTP/1.1
```

请求URI

- 在首部字段Host中写明网络域名或IP地址

```
GET /index.htm HTTP/1.1
Host: hackr.jp
```

图：以 http://hackr.jp/index.htm 作为请求的例子

除此之外，如果不是访问特定资源而是对服务器本身发起请求，可以用一个 * 来代替请求URI。下面这个例子是查询HTTP服务器端支持的HTTP方法种类。


```
OPTIONS * HTTP/1.1
```

2.5　告知服务器意图的HTTP方法

下面，我们介绍HTTP/1.1中可使用的方法。

GET：获取资源

GET方法用来请求访问已被URI识别的资源。指定的资源经服务器端解析后返回响应内容。也就是说，如果请求的资源是文本，那就保持原样返回；如果是像CGI（Common Gateway Interface，通用网关接口）那样的程序，则返回经过执行后的输出结果。



客户端　　服务器

使用 GET 方法的请求・响应的例子

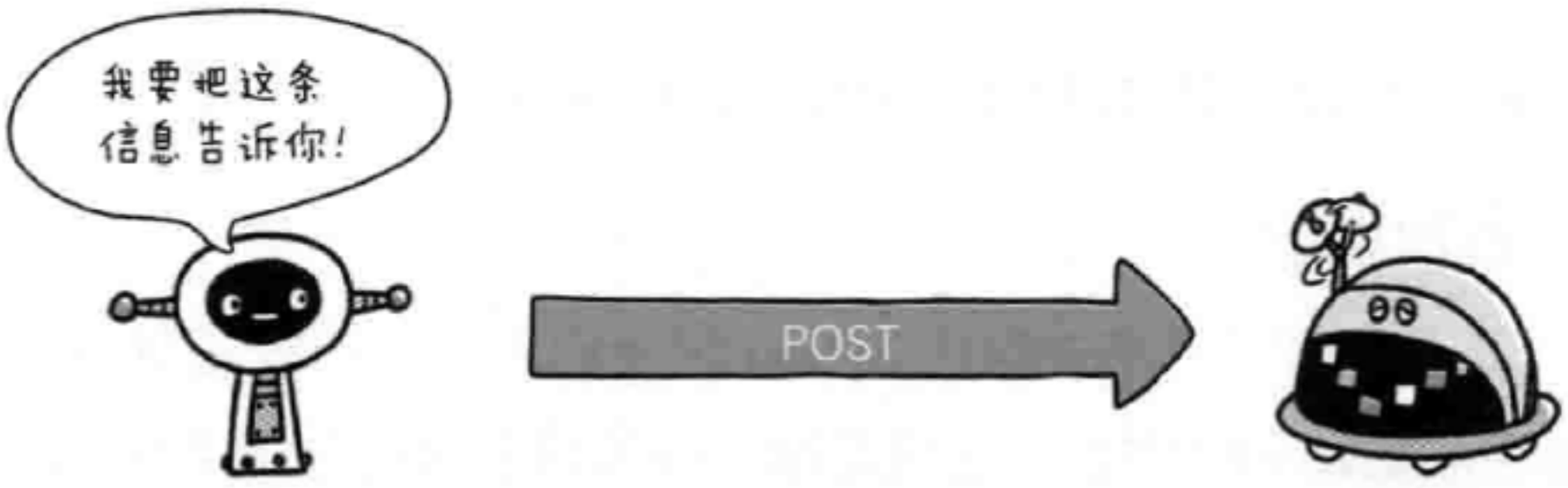
请求	GET /index.html HTTP/1.1 Host: www.hackr.jp
响应	返回 index.html 的页面资源

请求	GET /index.html HTTP/1.1 Host: www.hackr.jp If-Modified-Since: Thu, 12 Jul 2012 07:30:00 GMT
响应	仅返回2012年7月12日7点30分以后更新过的index.html页面资源。 如果未有内容更新，则以状态码304 Not Modified作为响应返回

POST：传输实体主体

POST 方法用来传输实体的主体。

虽然用 GET 方法也可以传输实体的主体，但一般不用 GET 方法进行传输，而是用 POST 方法。虽说 POST 的功能与 GET 很相似，但 POST 的主要目的并不是获取响应的主体内容。



客户端　　服务器

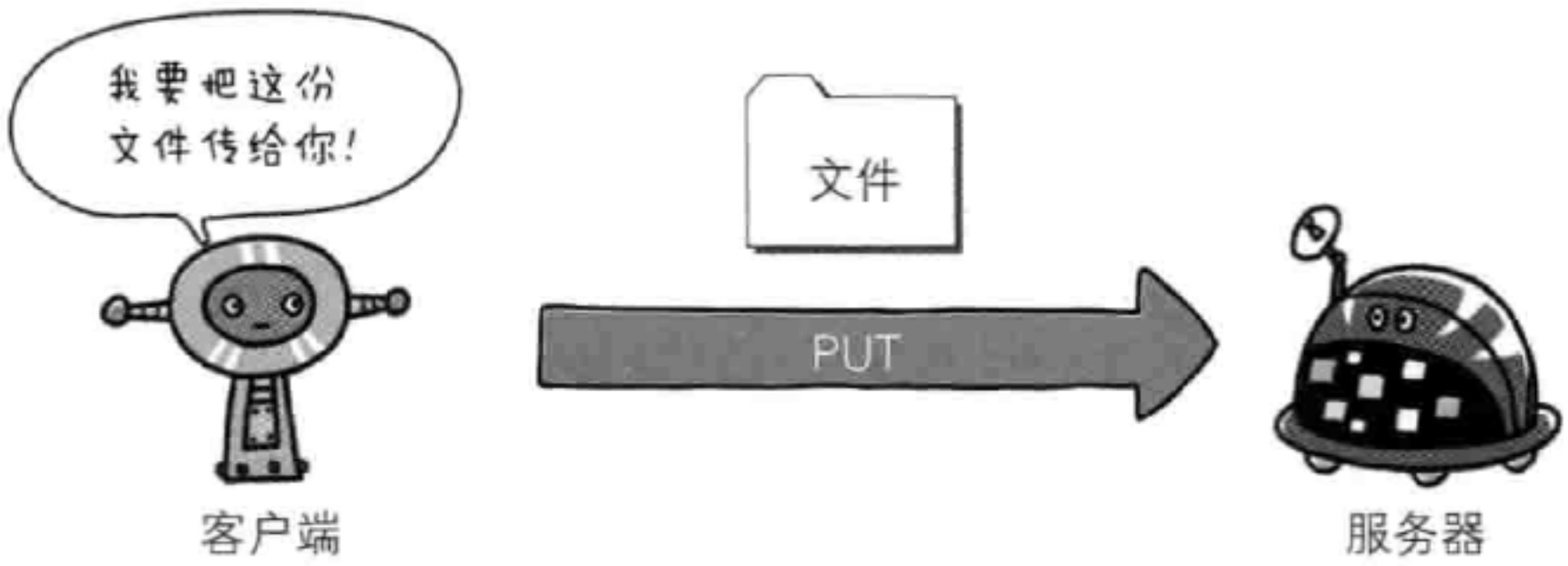
使用 POST 方法的请求・响应的例子

请求	POST /submit.cgi HTTP/1.1 Host: www.hackr.jp Content-Length: 1560（1560字节的数据）
响应	返回submit.cgi接收数据的处理结果

PUT：传输文件

PUT 方法用来传输文件。就像 FTP 协议的文件上传一样，要求在请求报文的主体中包含文件内容，然后保存到请求 URI 指定的位置。

但是，鉴于 HTTP/1.1 的 PUT 方法自身不带验证机制，任何人都可以上传文件，存在安全性问题，因此一般的 Web 网站不使用该方法。若配合 Web 应用程序的验证机制，或架构设计采用 REST（REpresentational State Transfer，表征状态转移）标准的同类 Web 网站，就可能会开放使用 PUT 方法。



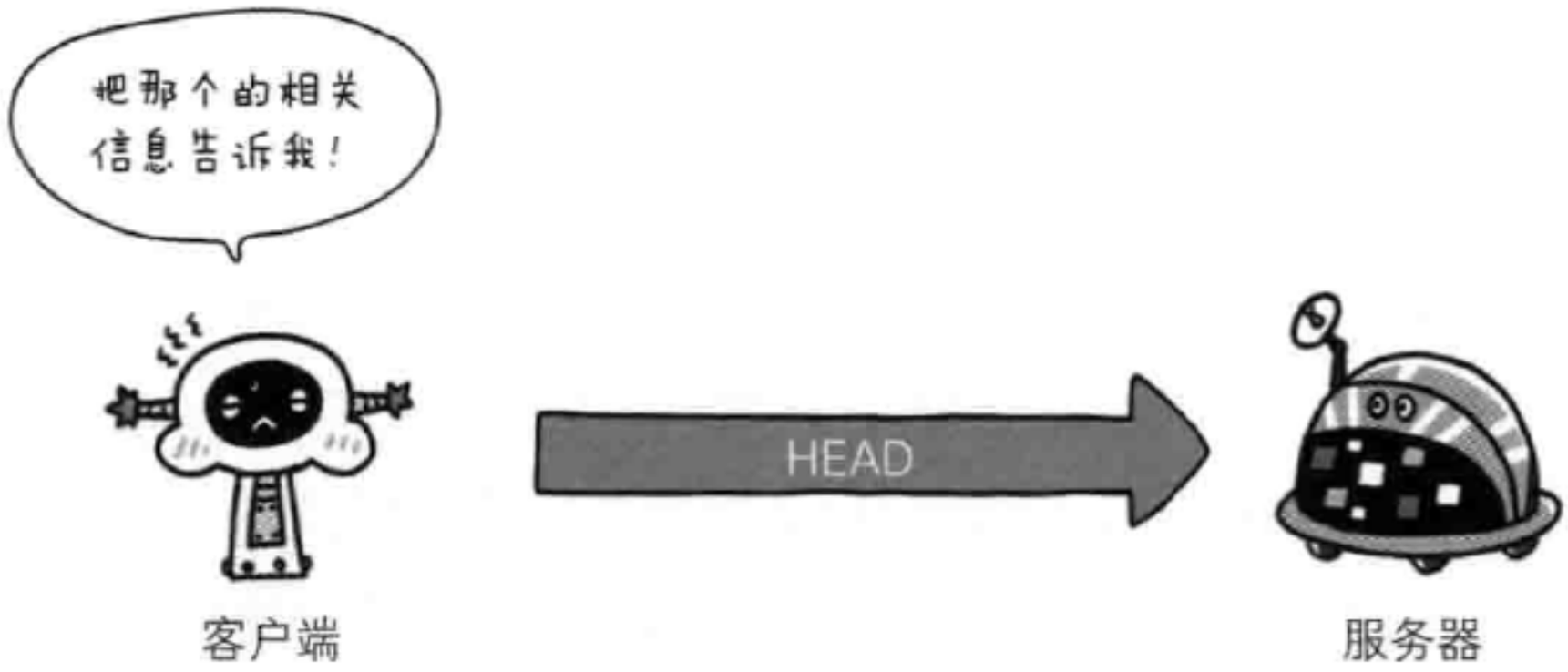
使用 PUT 方法的请求・响应的例子

请求	PUT /example.html HTTP/1.1 Host: www.hackr.jp Content-Type: text/html Content-Length: 1560（1560字节的数据）
响应①	响应返回状态码204 No Content（比如：该html已存在于服务器上）

① 响应的意思其实是请求执行成功了，但无数据返回。——译者注

HEAD：获得报文首部

HEAD 方法和 GET 方法一样，只是不返回报文主体部分。用于确认 URI 的有效性及资源更新的日期时间等。



图：和 GET 一样，但不返回报文主体

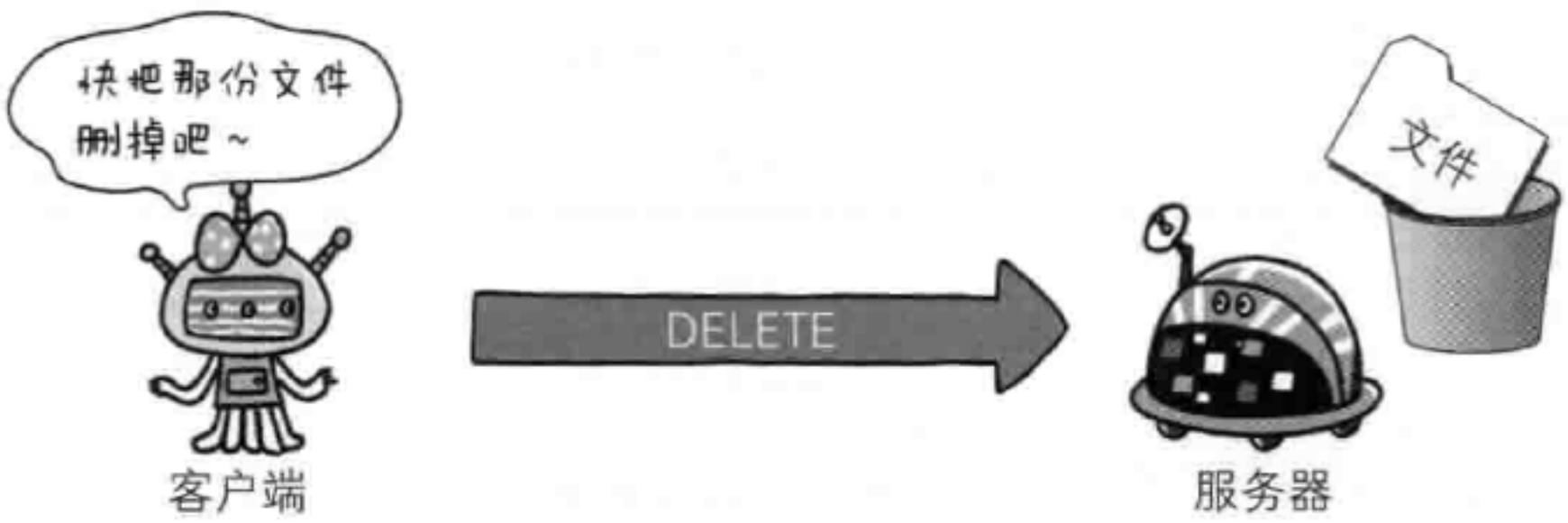
使用 HEAD 方法的请求・响应的例子

请求	HEAD /index.html HTTP/1.1 Host: www.hackr.jp
响应	返回 index.html 有关的响应首部

DELETE：删除文件

DELETE 方法用来删除文件，是与 PUT 相反的方法。DELETE 方法按请求 URI 删除指定的资源。

但是，HTTP/1.1 的 DELETE 方法本身和 PUT 方法一样不带验证机制，所以一般的 Web 网站也不使用 DELETE 方法。当配合 Web 应用程序的验证机制，或遵守 REST 标准时还是有可能会开放使用的。

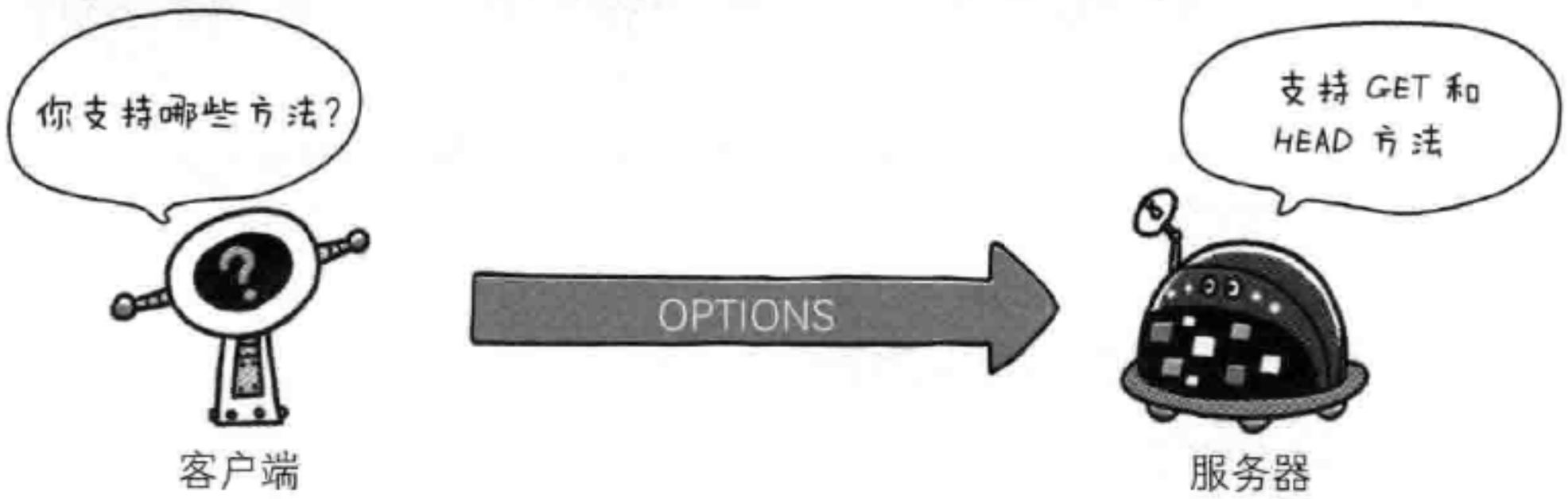


使用 DELETE 方法的请求・响应的例子

请求	DELETE /example.html HTTP/1.1 Host: www.hackr.jp
响应	响应返回状态码204 No Content（比如：该html已从该服务器上删除）

OPTIONS：询问支持的方法

OPTIONS 方法用来查询针对请求 URI 指定的资源支持的方法。



使用 OPTIONS 方法的请求・响应的例子

请求	OPTIONS * HTTP/1.1 Host: www.hackr.jp
响应	HTTP/1.1 200 OK Allow: GET, POST, HEAD, OPTIONS （返回服务器支持的方法）

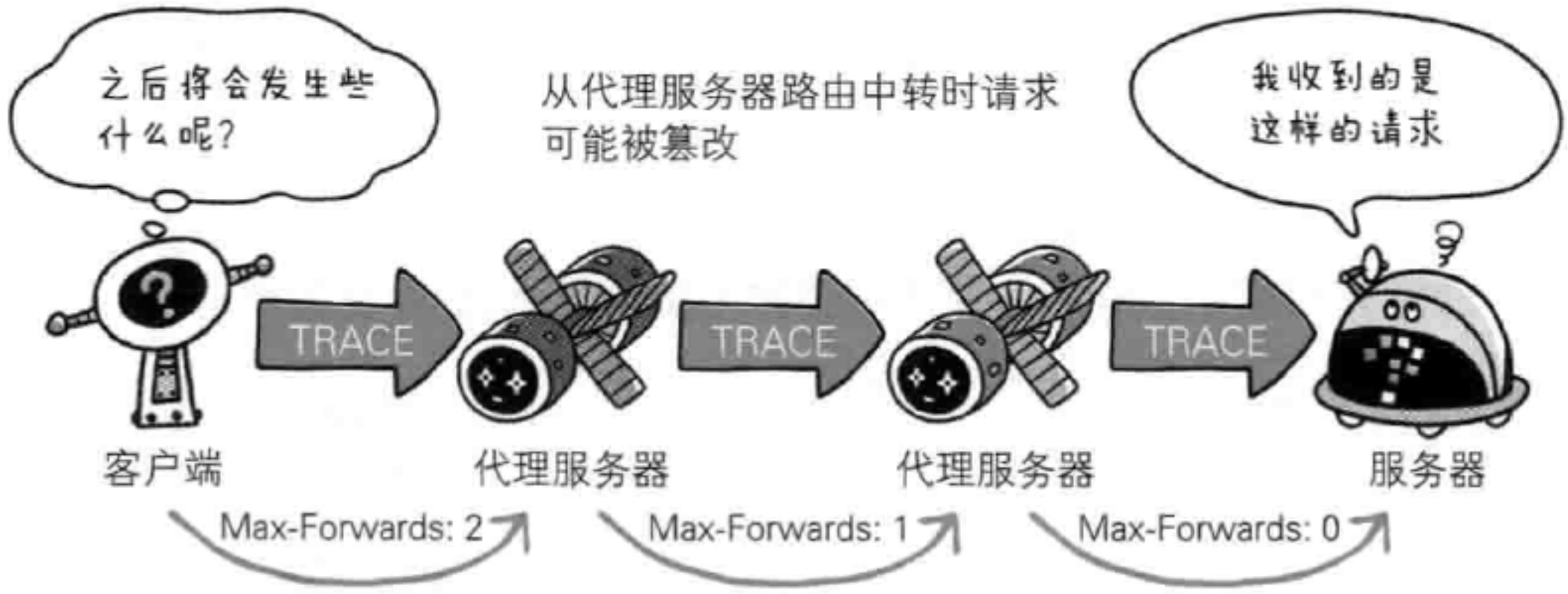
TRACE：追踪路径

TRACE 方法是让 Web 服务器端将之前的请求通信环回给客户端的方法。

发送请求时，在 Max-Forwards 首部字段中填入数值，每经过一个服务器端就将该数字减 1，当数值刚好减到 0 时，就停止继续传输，最后接收到请求的服务器端则返回状态码 200 OK 的响应。

客户端通过 TRACE 方法可以查询发送出去的请求是怎样被加工修改 / 篡改的。这是因为，请求想要连接到源目标服务器可能会通过代理中转，TRACE 方法就是用来确认连接过程中发生的一系列操作。

但是，TRACE 方法本来就不怎么常用，再加上它容易引发 XST（Cross-Site Tracing，跨站追踪）攻击，通常就更不会用到了。



使用 TRACE 方法的请求・响应的例子

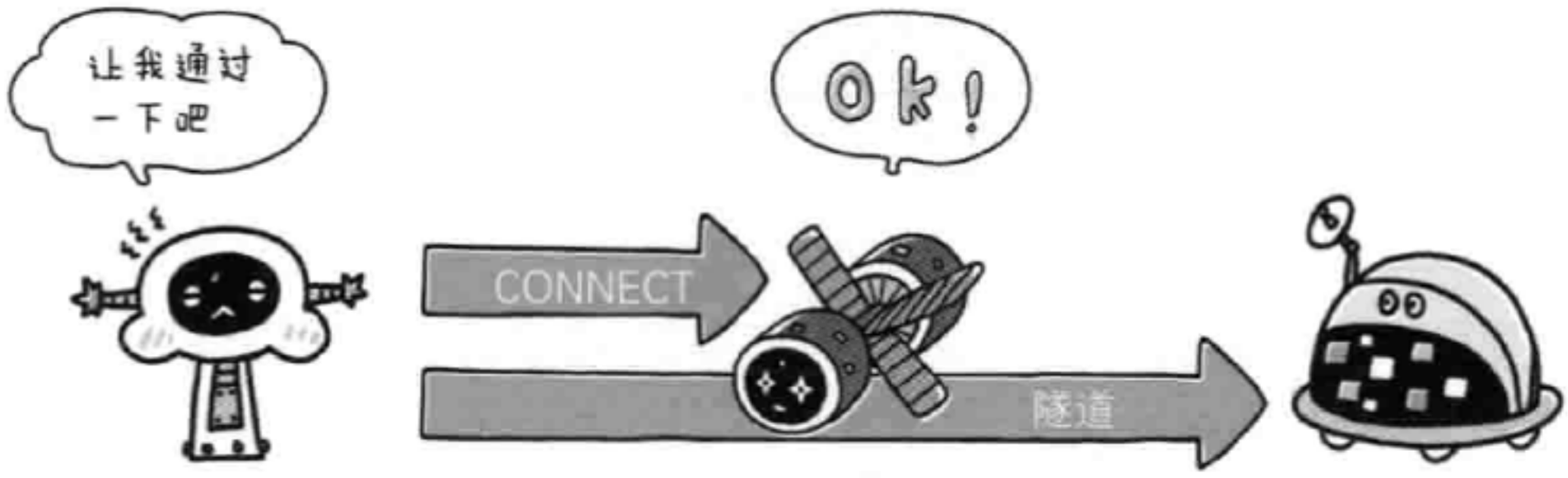
请求	TRACE / HTTP/1.1 Host: hackr.jp Max-Forwards: 2
响应	HTTP/1.1 200 OK Content-Type: message/http Content-Length: 1024 TRACE / HTTP/1.1 Host: hackr.jp Max-Forwards: 2（返回响应包含请求内容）

CONNECT：要求用隧道协议连接代理

CONNECT 方法要求在与代理服务器通信时建立隧道，实现用隧道协议进行 TCP 通信。主要使用 SSL（Secure Sockets Layer，安全套接层）和 TLS（Transport Layer Security，传输层安全）协议把通信内容加密后经网络隧道传输。

CONNECT 方法的格式如下所示。

```
CONNECT 代理服务器名：端口号 HTTP版本
```



使用 CONNECT 方法的请求・响应的例子

请求	CONNECT proxy.hackr.jp:8080 HTTP/1.1 Host: proxy.hackr.jp
响应	HTTP/1.1 200 OK（之后进入网络隧道）

2.6　使用方法下达命令

向请求 URI 指定的资源发送请求报文时，采用称为方法的命令。

方法的作用在于，可以指定请求的资源按期望产生某种行为。方法中有 GET、POST 和 HEAD 等。



图：使用方法给服务器下达命令

下表列出了 HTTP/1.0 和 HTTP/1.1 支持的方法。另外，方法名区分大小写，注意要用大写字母。

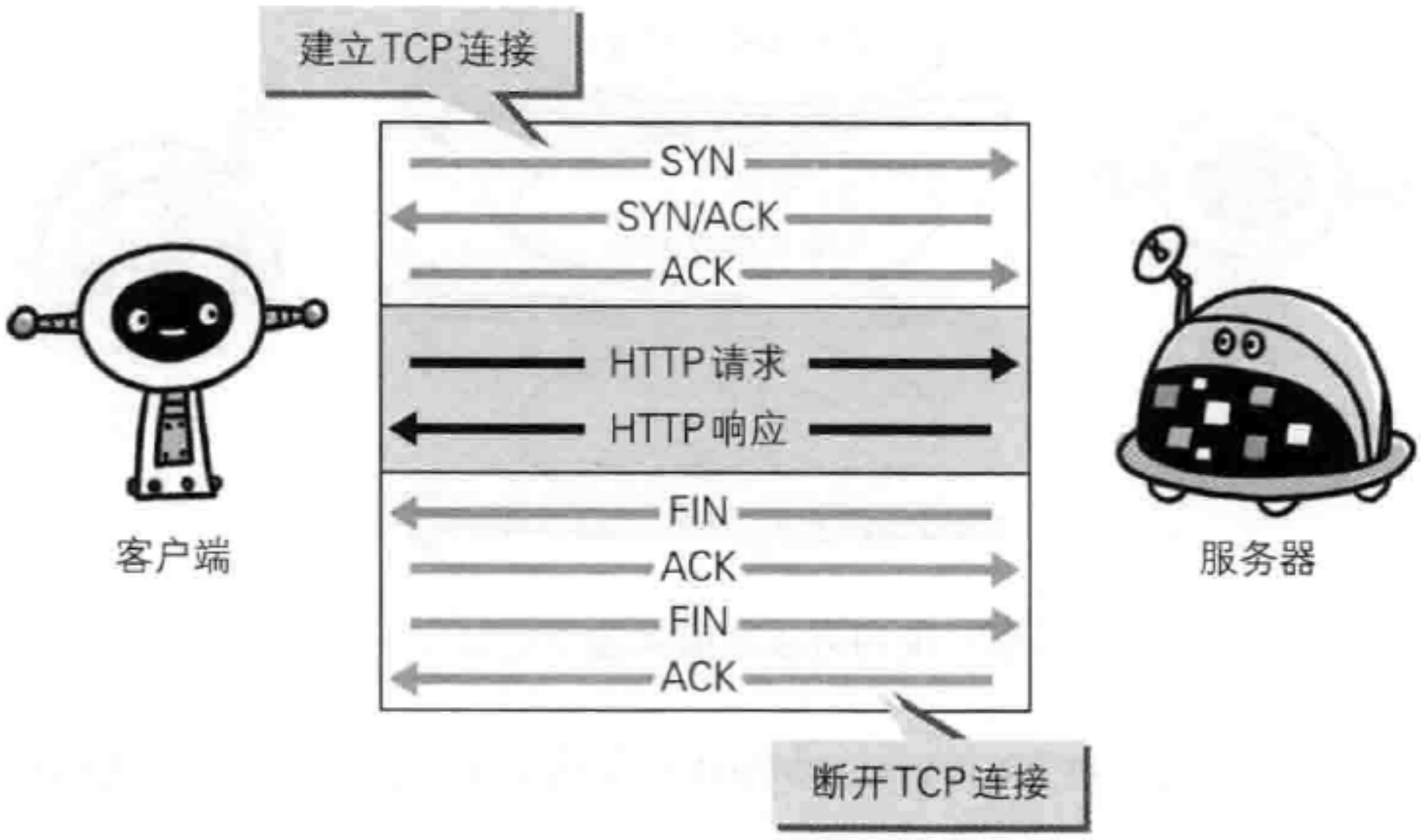
表 2-1：HTTP/1.0 和 HTTP/1.1 支持的方法

方法	说明	支持的HTTP协议版本
GET	获取资源	1.0、1.1
POST	传输实体主体	1.0、1.1
PUT	传输文件	1.0、1.1
HEAD	获得报文首部	1.0、1.1
DELETE	删除文件	1.0、1.1
OPTIONS	询问支持的方法	1.1
TRACE	追踪路径	1.1
CONNECT	要求用隧道协议连接代理	1.1
LINK	建立和资源之间的联系	1.0
UNLINE	断开连接关系	1.0

在这里列举的众多方法中，LINK 和 UNLINK 已被 HTTP/1.1 废弃，不再支持。

2.7 持久连接节省通信量

HTTP 协议的初始版本中，每进行一次 HTTP 通信就要断开一次 TCP 连接。

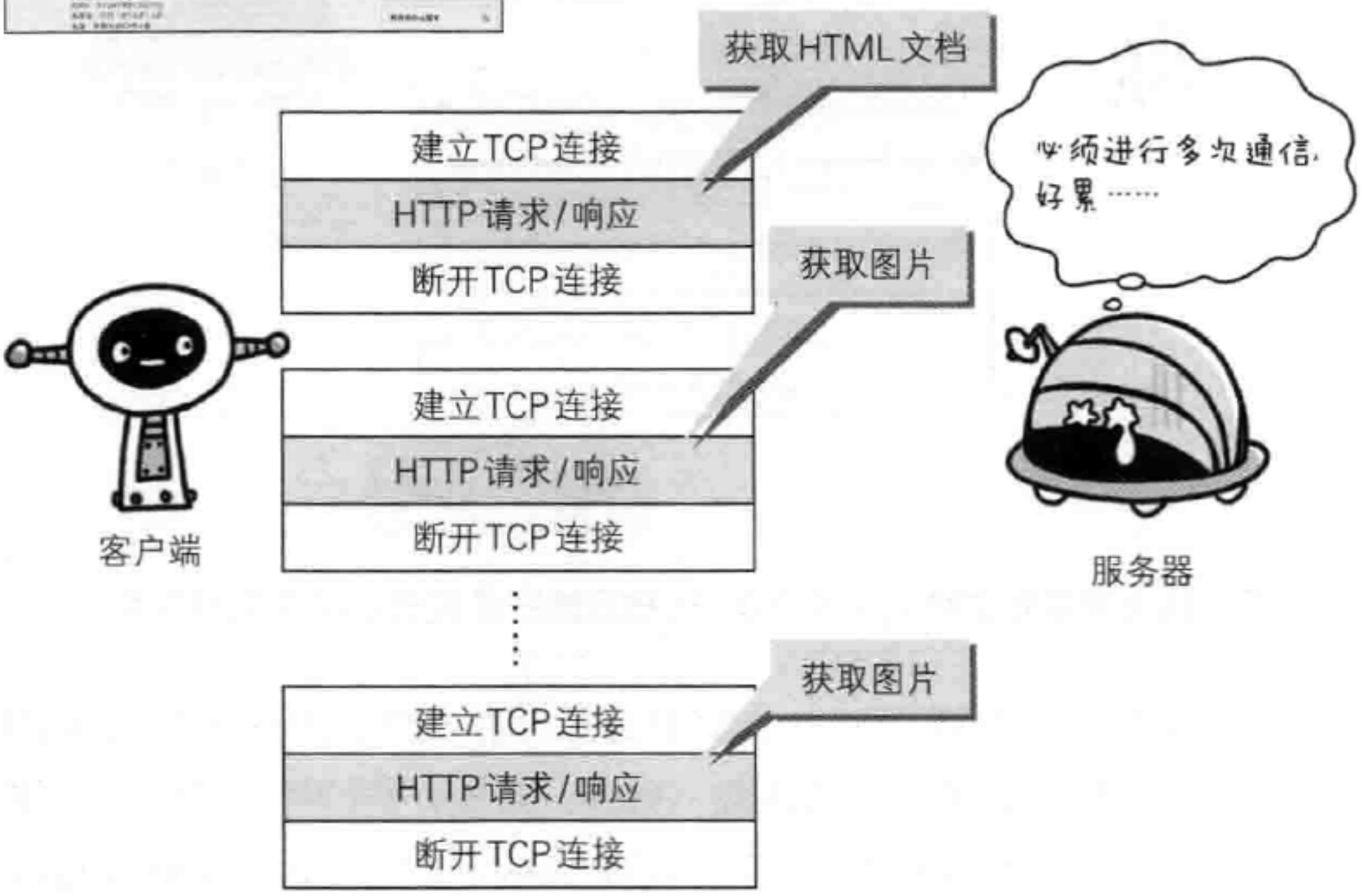


以当年的通信情况来说，因为都是些容量很小的文本传输，所以即使这样也没有多大问题。可随着 HTTP 的普及，文档中包含大量图片的情况多了起来。

比如，使用浏览器浏览一个包含多张图片的 HTML 页面时，在发送请求访问 HTML 页面资源的同时，也会请求该 HTML 页面里包含的其他资源。因此，每次的请求都会造成无谓的 TCP 连接建立和断开，增加通信量的开销。

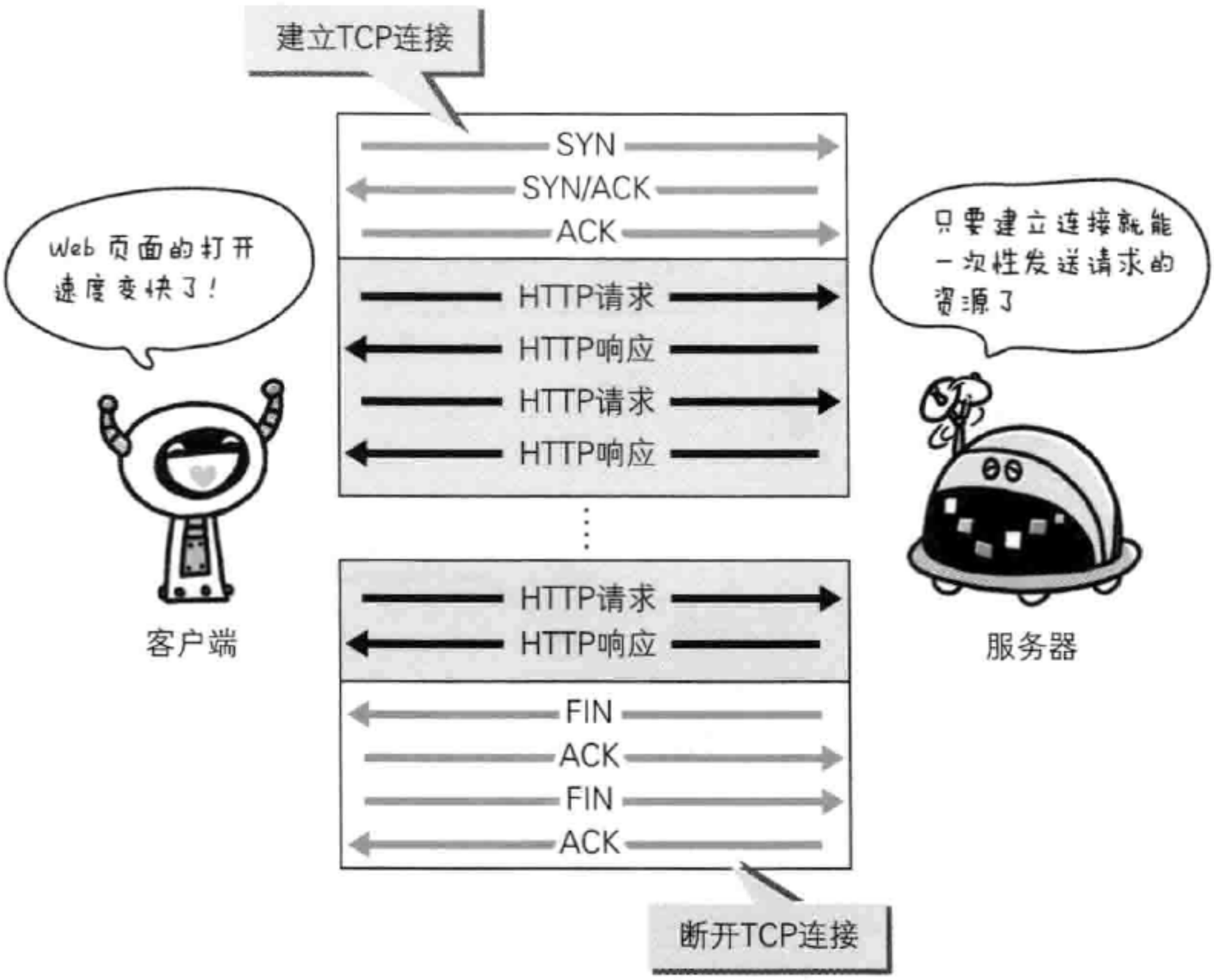


发送请求一份包含多张图片的HTML文档对应的Web页面，会产生大量的通信开销。



2.7.1 持久连接

为解决上述 TCP 连接的问题，HTTP/1.1 和一部分的 HTTP/1.0 想出了持久连接（HTTP Persistent Connections，也称为 HTTP keep-alive 或 HTTP connection reuse）的方法。持久连接的特点是，只要任意一端没有明确提出断开连接，则保持 TCP 连接状态。



图：持久连接旨在建立 1 次 TCP 连接后进行多次请求和响应的交互

持久连接的好处在于减少了 TCP 连接的重复建立和断开所造成的额外开销，减轻了服务器端的负载。另外，减少开销的那部分时间，使 HTTP 请求和响应能够更早地结束，这样 Web 页面的显示速度也就相应提高了。

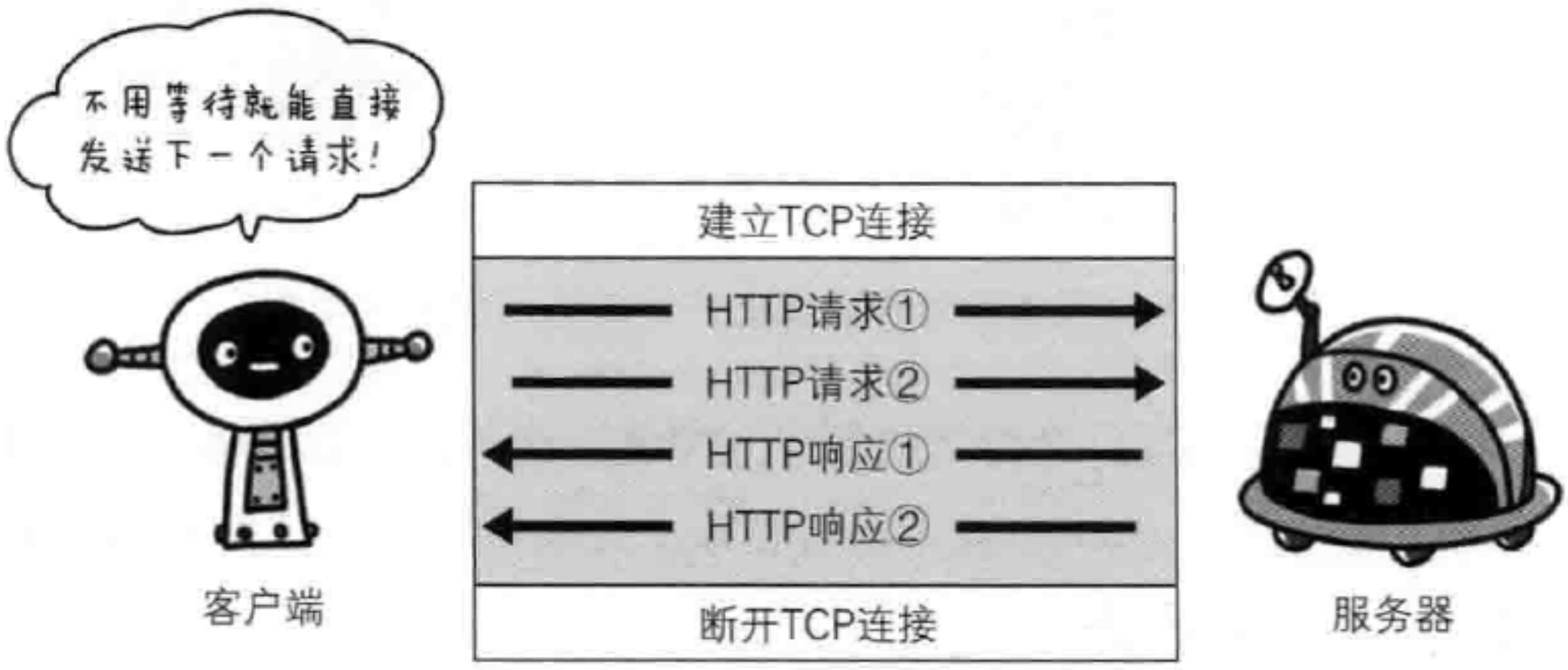
在 HTTP/1.1 中，所有的连接默认都是持久连接，但在 HTTP/1.0 内并未标准化。虽然有一部分服务器通过非标准的手段实现了持久连接，

但服务器端不一定能够支持持久连接。毫无疑问，除了服务器端，客户端也需要支持持久连接。

2.7.2　管线化

持久连接使得多数请求以管线化（pipelining）方式发送成为可能。从前发送请求后需等待并收到响应，才能发送下一个请求。管线化技术出现后，不用等待响应亦可直接发送下一个请求。

这样就能够做到同时并行发送多个请求，而不需要一个接一个地等待响应了。



图：不等待响应，直接发送下一个请求

比如，当请求一个包含 10 张图片的 HTML Web 页面，与挨个连接相比，用持久连接可以让请求更快结束。而管线化技术则比持久连接还要快。请求数越多，时间差就越明显。

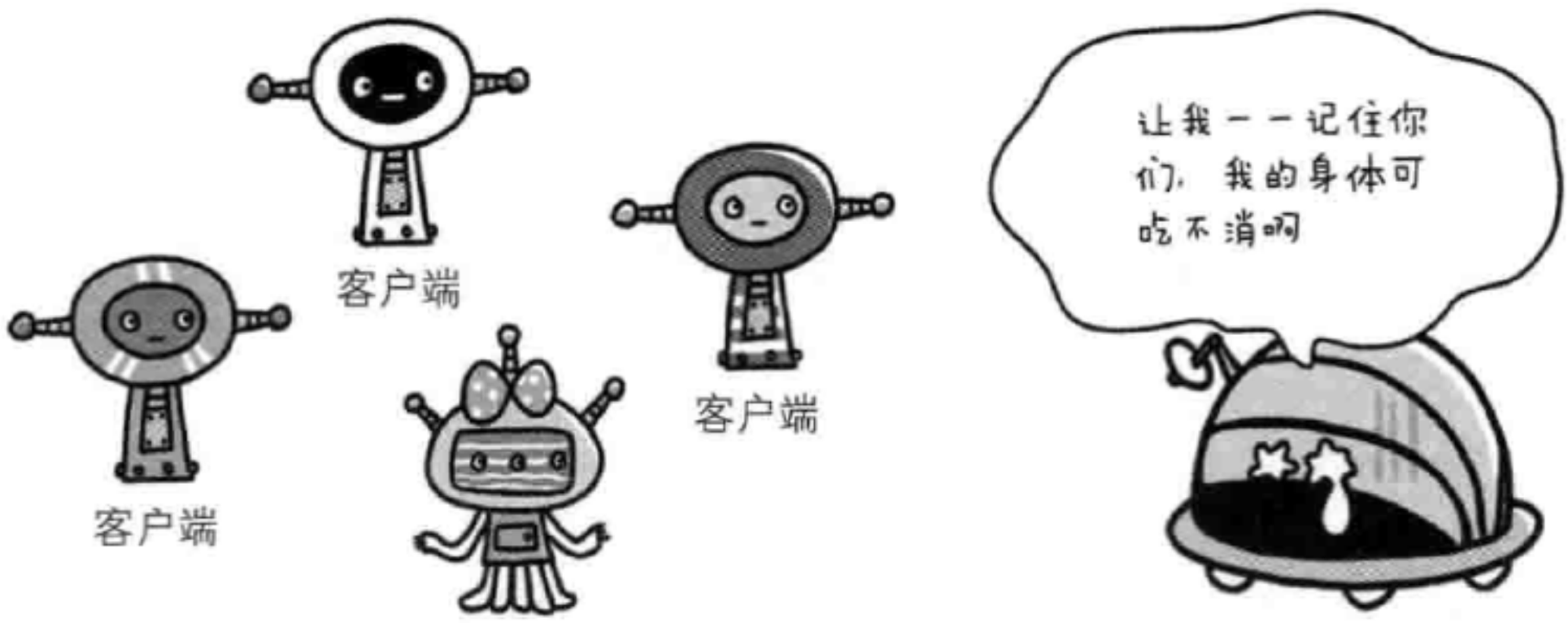
2.8　使用 Cookie 的状态管理

HTTP 是无状态协议，它不对之前发生过的请求和响应的状态进行管理。也就是说，无法根据之前的状态进行本次的请求处理。

假设要求登录认证的 Web 页面本身无法进行状态的管理（不记录已登录的状态），那么每次跳转新页面不是要再次登录，就是要在每次

请求报文中附加参数来管理登录状态。

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。



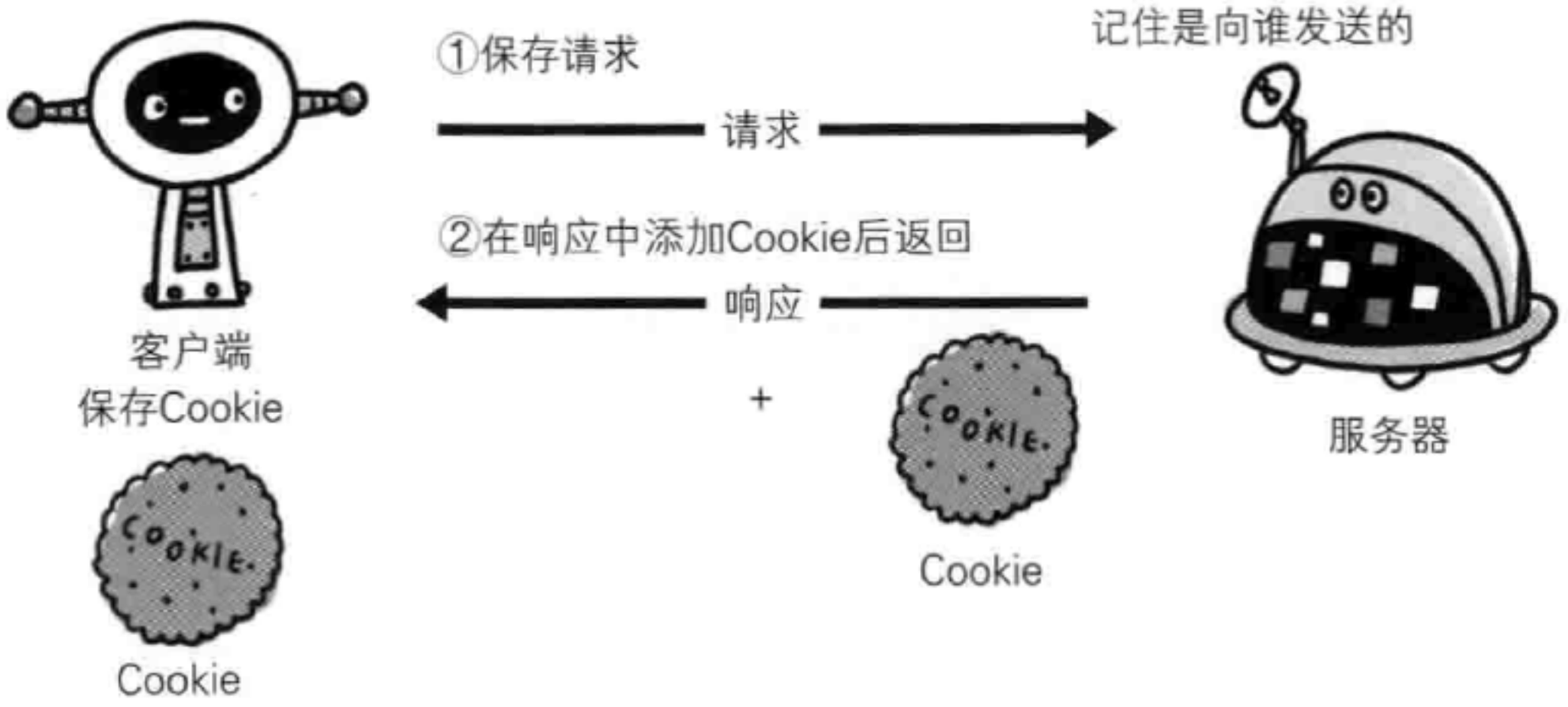
图：如果让服务器管理全部客户端状态则会成为负担

保留无状态协议这个特征的同时又要解决类似的矛盾问题，于是引入了 Cookie 技术。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。

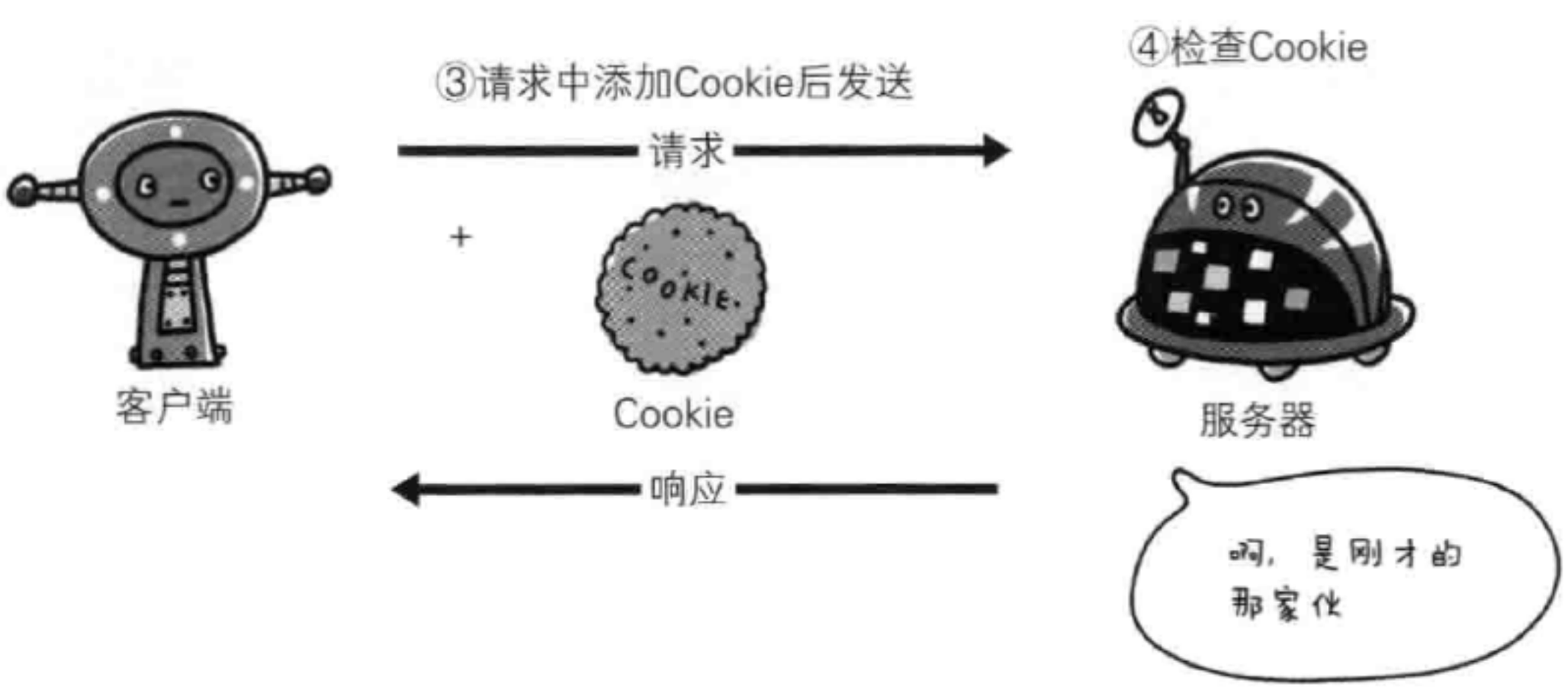
Cookie 会根据从服务器端发送的响应报文内的一个叫做 Set-Cookie 的首部字段信息，通知客户端保存 Cookie。当下次客户端再往该服务器发送请求时，客户端会自动在请求报文中加入 Cookie 值后发送出去。

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。

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）

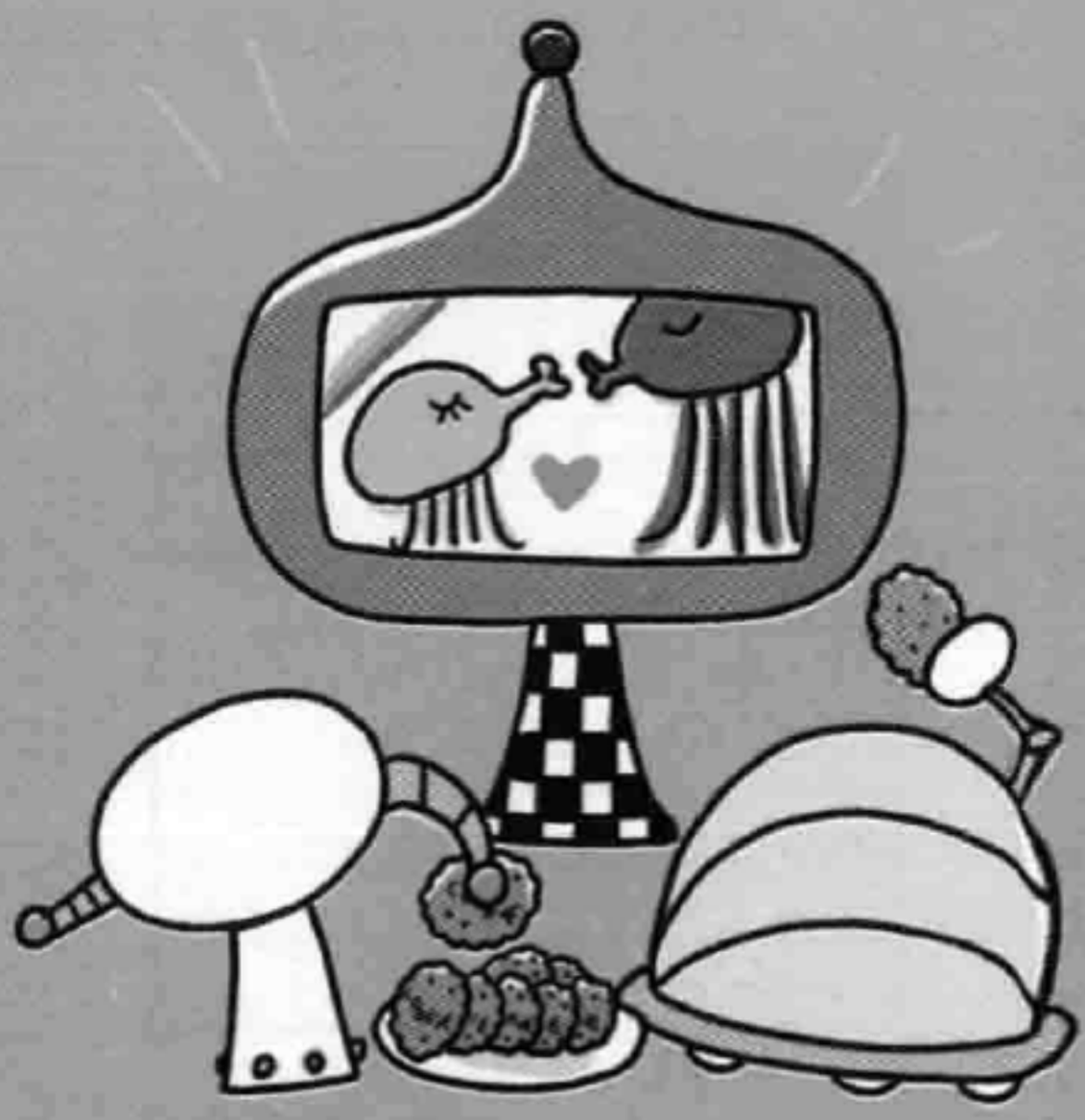
```
GET /image/ HTTP/1.1
Host: hackr.jp
Cookie: sid=1342077140226724
```


有关请求报文和响应报文内 Cookie 对应的首部字段，请参考之后的章节。

Chapter3

第3章
HTTP报文内的HTTP信息

HTTP通信过程包括从客户端发往服务器端的请求及从服务器端返回客户端的响应。本章就让我们来了解一下请求和响应是怎样运作的。



3.1 HTTP 报文

用于 HTTP 协议交互的信息被称为 HTTP 报文。请求端（客户端）的 HTTP 报文叫做请求报文，响应端（服务器端）的叫做响应报文。HTTP 报文本身是由多行（用 CR+LF 作换行符）数据构成的字符串文本。

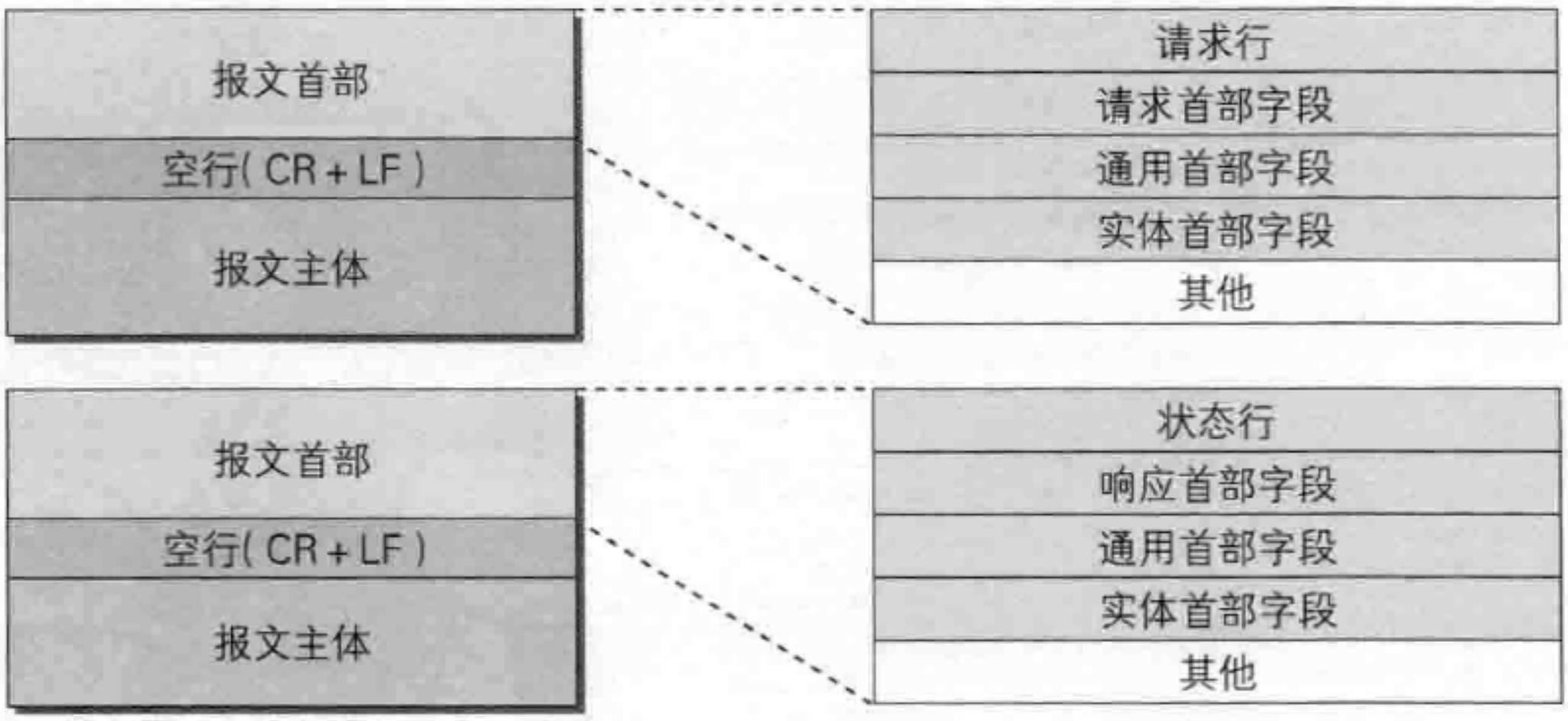
HTTP 报文大致可分为报文首部和报文主体两块。两者由最初出现的空行（CR+LF）来划分。通常，并不一定要有报文主体。



图：HTTP 报文的结构

3.2 请求报文及响应报文的结构

我们来看一下请求报文和响应报文的结构。



图：请求报文（上）和响应报文（下）的结构

```
GET / HTTP/1.1                                                    请求行
Host: hackr.jp
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:13.0) Gecko/20100101 Firefox/13.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: ja,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache                                           各种首部字段
空行(CR+LF)

HTTP/1.1 200 OK                                                   状态行
Date: Fri, 13 Jul 2012 02:45:26 GMT
Server: Apache
Last-Modified: Fri, 31 Aug 2007 02:02:20 GMT
ETag: "45bae1-16a-46d776ac"
Accept-Ranges: bytes
Content-Length: 362
Connection: close
Content-Type: text/html                                           各种首部字段
空行(CR+LF)
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>hackr.jp</title>
</head>
<body>
<img src="hackr.gif" alt="hackr.jp" width="240" height="84" />
</body>
</html>                                                           报文主体
```

图：请求报文（上）和响应报文（下）的实例

请求报文和响应报文的首部内容由以下数据组成。现在出现的各种首部字段及状态码稍后会进行阐述。

请求行

包含用于请求的方法，请求 URI 和 HTTP 版本。

状态行

包含表明响应结果的状态码，原因短语和 HTTP 版本。

首部字段

包含表示请求和响应的各种条件和属性的各类首部。

一般有 4 种首部，分别是：通用首部、请求首部、响应首部和实体首部。

其他

可能包含 HTTP 的 RFC 里未定义的首部（Cookie 等）。

3.3 编码提升传输速率

HTTP 在传输数据时可以按照数据原貌直接传输，但也可以在传输过程中通过编码提升传输速率。通过在传输时编码，能有效地处理大量的访问请求。但是，编码的操作需要计算机来完成，因此会消耗更多的 CPU 等资源。

3.3.1 报文主体和实体主体的差异

- 报文（message）

  是 HTTP 通信中的基本单位，由 8 位组字节流（octet sequence，其中 octet 为 8 个比特）组成，通过 HTTP 通信传输。

- 实体（entity）

  作为请求或响应的有效载荷数据（补充项）被传输，其内容由实体首部和实体主体组成。

HTTP 报文的主体用于传输请求或响应的实体主体。

通常，报文主体等于实体主体。只有当传输中进行编码操作时，实体主体的内容发生变化，才导致它和报文主体产生差异。

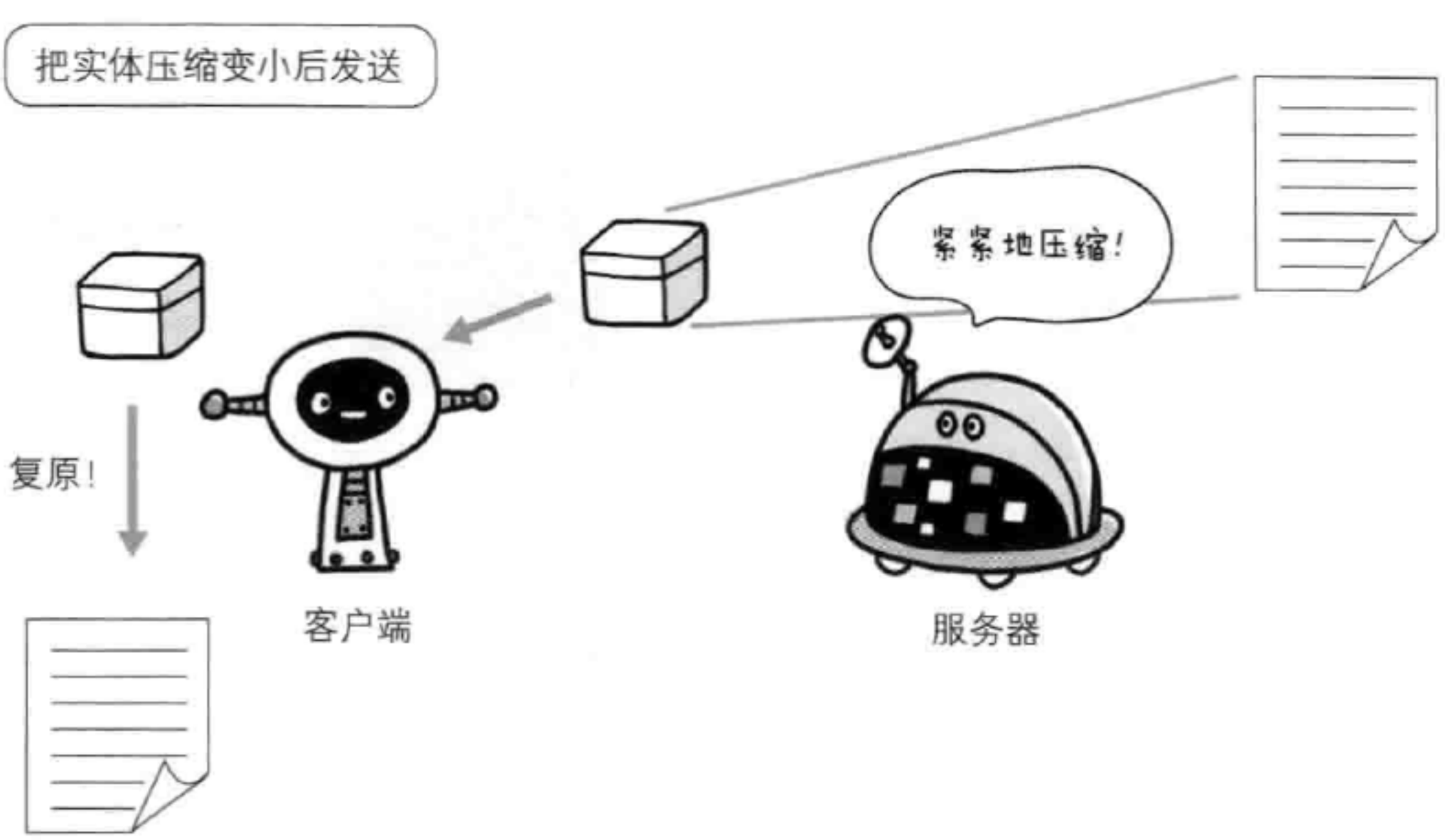
报文和实体这两个术语在之后会经常出现，请事先理解两者的差异。

3.3.2 压缩传输的内容编码

向待发送邮件内增加附件时，为了使邮件容量变小，我们会先用 ZIP 压缩文件之后再添加附件发送。HTTP 协议中有一种被称为内容编

码的功能也能进行类似的操作。

内容编码指明应用在实体内容上的编码格式，并保持实体信息原样压缩。内容编码后的实体由客户端接收并负责解码。



图：内容编码

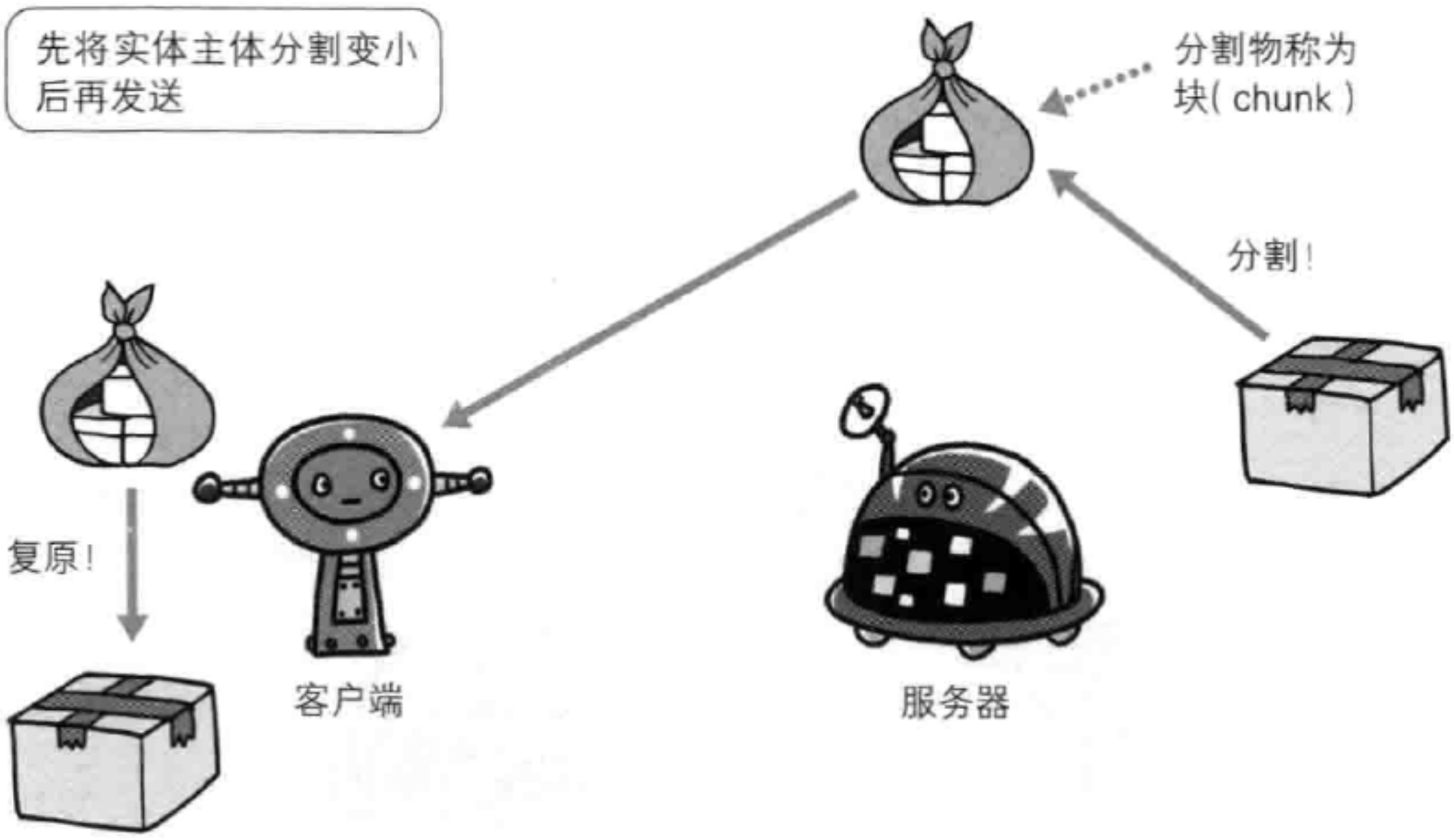
常用的内容编码有以下几种。

- gzip（GNU zip）
- compress（UNIX 系统的标准压缩）
- deflate（zlib）
- identity（不进行编码）

3.3.3　分割发送的分块传输编码

在 HTTP 通信过程中，请求的编码实体资源尚未全部传输完成之前，浏览器无法显示请求页面。在传输大容量数据时，通过把数据分割成多块，能够让浏览器逐步显示页面。

这种把实体主体分块的功能称为分块传输编码（Chunked Transfer Coding）。



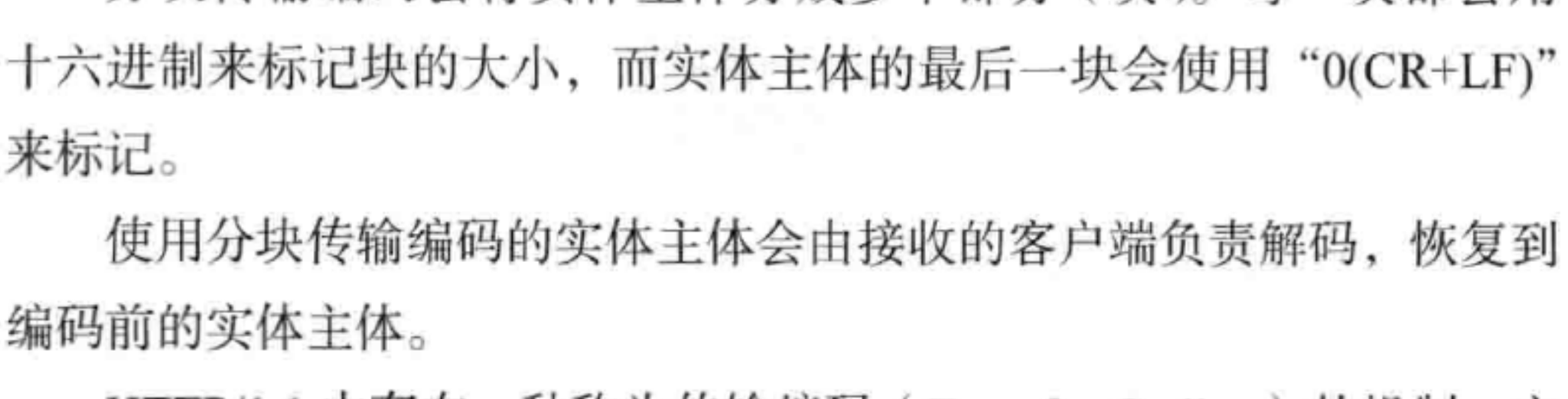
图：分块传输编码

分块传输编码会将实体主体分成多个部分（块）。每一块都会用十六进制来标记块的大小，而实体主体的最后一块会使用“0(CR+LF)”来标记。

使用分块传输编码的实体主体会由接收的客户端负责解码，恢复到编码前的实体主体。

HTTP/1.1 中存在一种称为传输编码（Transfer Coding）的机制，它可以在通信时按某种编码方式传输，但只定义作用于分块传输编码中。

3.4 发送多种数据的多部分对象集合



发送邮件时，我们可以在邮件里写入文字并添加多份附件。这是因

为采用了 MIME（Multipurpose Internet Mail Extensions，多用途因特网邮件扩展）机制，它允许邮件处理文本、图片、视频等多个不同类型的数据。例如，图片等二进制数据以 ASCII 码字符串编码的方式指明，就是利用 MIME 来描述标记数据类型。而在 MIME 扩展中会使用一种称为多部分对象集合（Multipart）的方法，来容纳多份不同类型的数据。

相应地，HTTP 协议中也采纳了多部分对象集合，发送的一份报文主体内可含有多类型实体。通常是在图片或文本文件等上传时使用。

多部分对象集合包含的对象如下。

- multipart/form-data

  在 Web 表单文件上传时使用。

- multipart/byteranges

  状态码 206（Partial Content，部分内容）响应报文包含了多个范围的内容时使用。

- multipart/form-data

```
Content-Type: multipart/form-data; boundary=AaB03x

--AaB03x
Content-Disposition: form-data; name="field1"

Joe Blow
--AaB03x
Content-Disposition: form-data; name="pics"; filename="file1.txt"
Content-Type: text/plain

...（file1.txt的数据）...
--AaB03x--
```

- multipart/byteranges

```
HTTP/1.1 206 Partial Content
```

```
Date: Fri, 13 Jul 2012 02:45:26 GMT
Last-Modified: Fri, 31 Aug 2007 02:02:20 GMT
Content-Type: multipart/byteranges; boundary=THIS_STRING_SEPARATES

--THIS_STRING_SEPARATES
Content-Type: application/pdf
Content-Range: bytes 500-999/8000

...（范围指定的数据）...
--THIS_STRING_SEPARATES
Content-Type: application/pdf
Content-Range: bytes 7000-7999/8000

...（范围指定的数据）...
--THIS_STRING_SEPARATES--
```

在 HTTP 报文中使用多部分对象集合时，需要在首部字段里加上 Content-type。有关这个首部字段，我们稍后讲解。

使用 boundary 字符串来划分多部分对象集合指明的各类实体。在 boundary 字符串指定的各个实体的起始行之前插入“--”标记（例如：--AaB03x、--THIS_STRING_SEPARATES），而在多部分对象集合对应的字符串的最后插入“--”标记（例如：--AaB03x--、--THIS_STRING_SEPARATES--）作为结束。

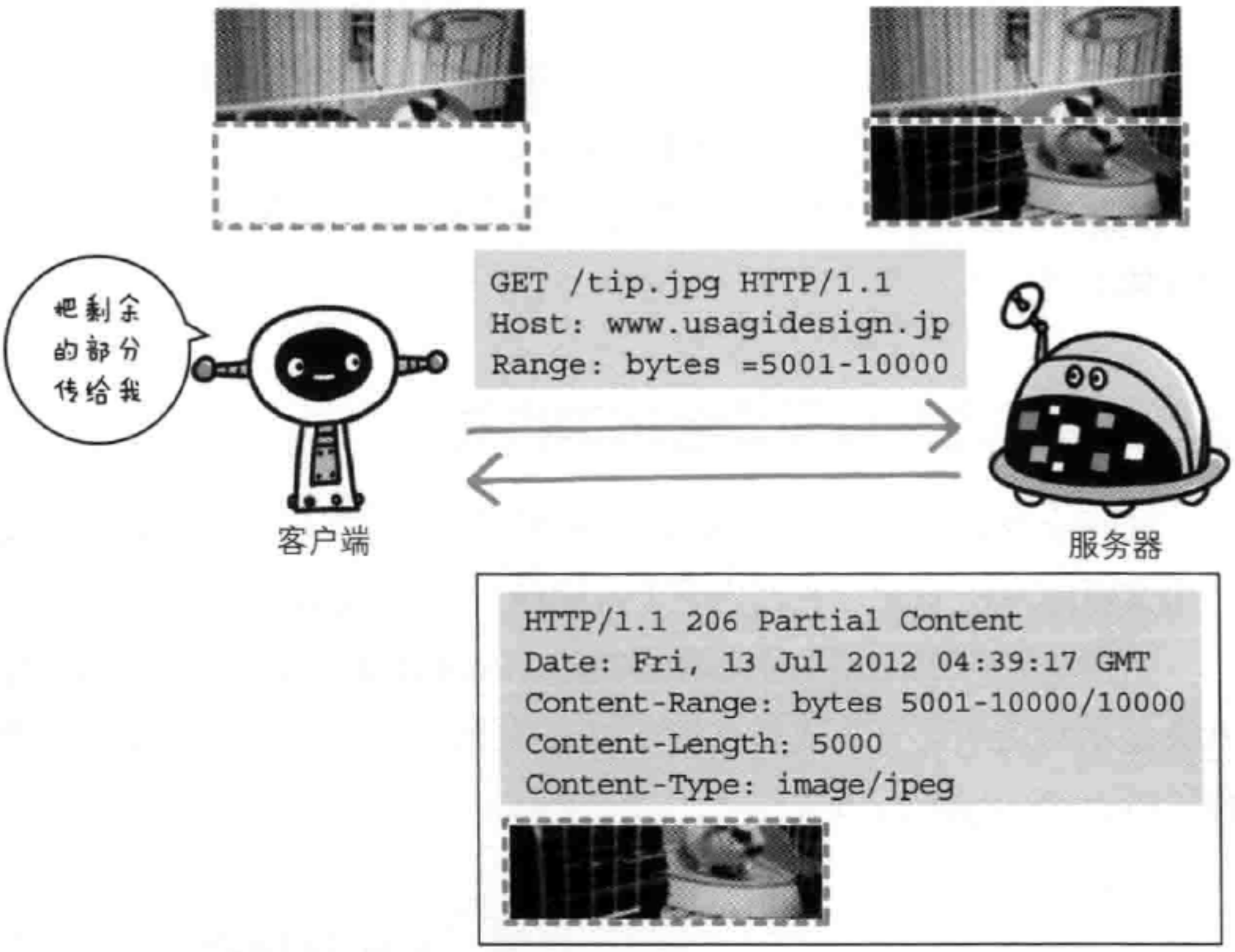
多部分对象集合的每个部分类型中，都可以含有首部字段。另外，可以在某个部分中嵌套使用多部分对象集合。有关多部分对象集合更详细的解释，请参考 RFC2046。

3.5 获取部分内容的范围请求

以前，用户不能使用现在这种高速的带宽访问互联网，当时，下载一个尺寸稍大的图片或文件就已经很吃力了。如果下载过程中遇到网络中断的情况，那就必须重头开始。为了解决上述问题，需要一种可恢复的机制。所谓恢复是指能从之前下载中断处恢复下载。

要实现该功能需要指定下载的实体范围。像这样，指定范围发送的请求叫做范围请求（Range Request）。

对一份 10 000 字节大小的资源，如果使用范围请求，可以只请求 5001~10 000 字节内的资源。



执行范围请求时，会用到首部字段 Range 来指定资源的 byte 范围。byte 范围的指定形式如下。

- 5001~10 000 字节

```
Range: bytes=5001-10000
```

- 从 5001 字节之后全部的

```
Range: bytes=5001-
```



- 从一开始到 3000 字节和 5000~7000 字节的多重范围

```
Range: bytes=-3000, 5000-7000
```

针对范围请求，响应会返回状态码为 206 Partial Content 的响应报文。另外，对于多重范围的范围请求，响应会在首部字段 Content-Type 标明 multipart/byteranges 后返回响应报文。

如果服务器端无法响应范围请求，则会返回状态码 200 OK 和完整的实体内容。

3.6 内容协商返回最合适的内容

同一个 Web 网站有可能存在着多份相同内容的页面。比如英语版和中文版的 Web 页面，它们内容上虽相同，但使用的语言却不同。

当浏览器的默认语言为英语或中文，访问相同 URI 的 Web 页面时，则会显示对应的英语版或中文版的 Web 页面。这样的机制称为内容协商（Content Negotiation）。

图：访问 http://www.google.com/

内容协商机制是指客户端和服务器端就响应的资源内容进行交涉，然后提供给客户端最为适合的资源。内容协商会以响应资源的语言、字符集、编码方式等作为判断的基准。

包含在请求报文中的某些首部字段（如下）就是判断的基准。这些首部字段的详细说明请参考下一章。

- Accept
- Accept-Charset
- Accept-Encoding
- Accept-Language
- Content-Language

内容协商技术有以下 3 种类型。

服务器驱动协商（Server-driven Negotiation）

由服务器端进行内容协商。以请求的首部字段为参考，在服务器端自动处理。但对用户来说，以浏览器发送的信息作为判定的依据，并不一定能筛选出最优内容。

客户端驱动协商（Agent-driven Negotiation）

由客户端进行内容协商的方式。用户从浏览器显示的可选项列表中手动选择。还可以利用 JavaScript 脚本在 Web 页面上自动进行上述选择。比如按 OS 的类型或浏览器类型，自行切换成 PC 版页面或手机版页面。

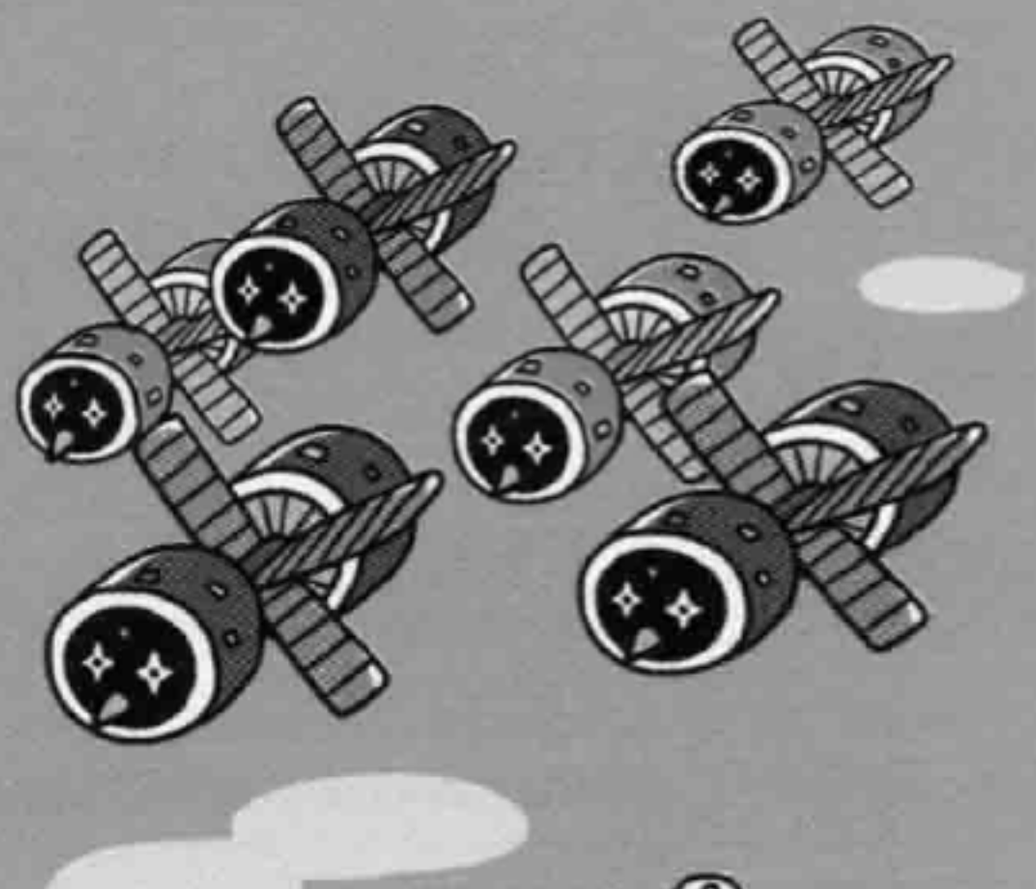
透明协商（Transparent Negotiation）

是服务器驱动和客户端驱动的结合体，是由服务器端和客户端各自进行内容协商的一种方法。

Chapter4

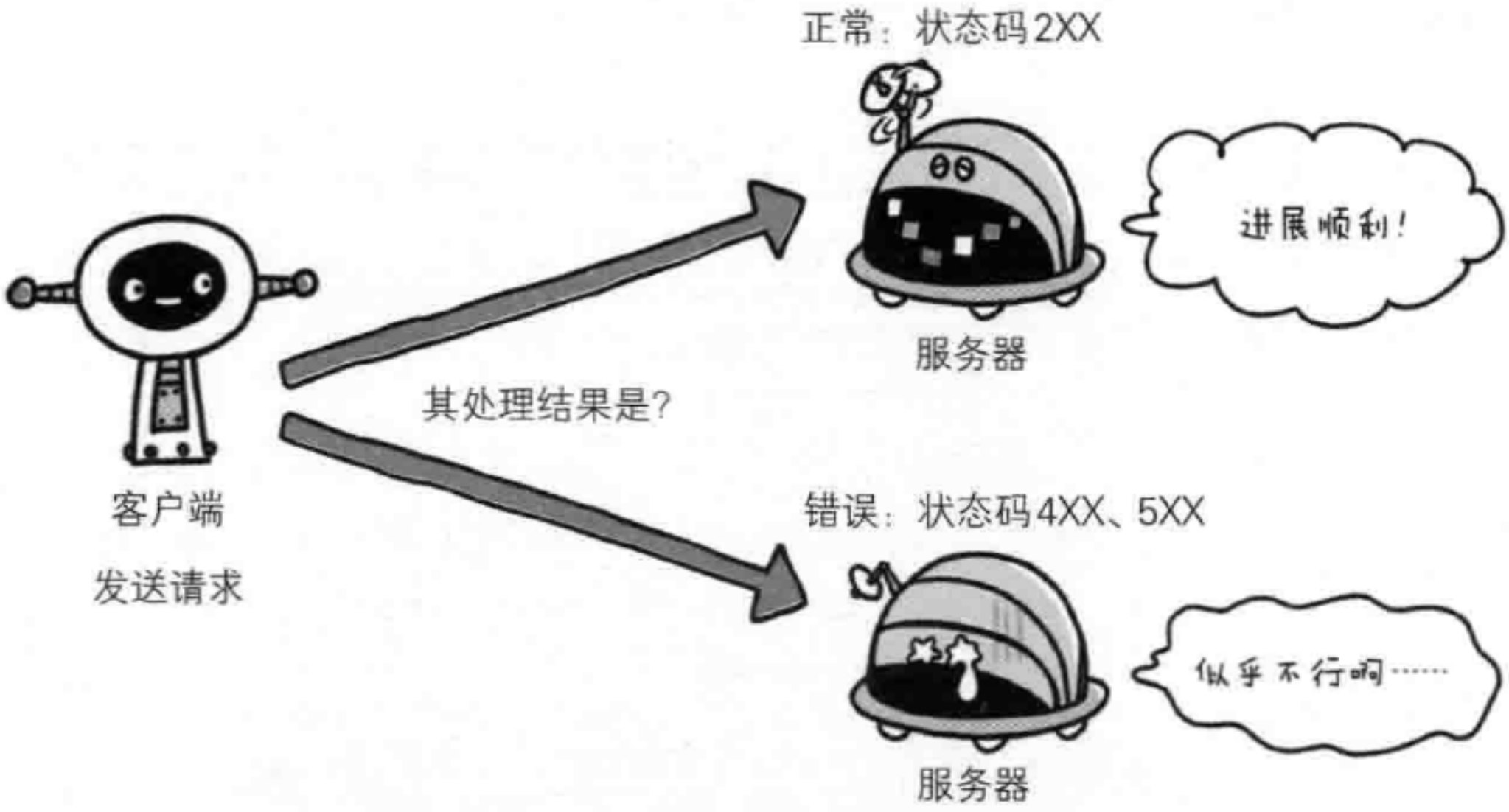
第4章
返回结果的HTTP状态码

HTTP状态码负责表示客户端HTTP请求的返回结果、标记服务器端的处理是否正常、通知出现的错误等工作。让我们通过本章的学习，好好了解一下状态码的工作机制。



4.1 状态码告知从服务器端返回的请求结果

状态码的职责是当客户端向服务器端发送请求时，描述返回的请求结果。借助状态码，用户可以知道服务器端是正常处理了请求，还是出现了错误。



图：响应的状态码可描述请求的处理结果

状态码如 200 OK，以 3 位数字和原因短语组成。

数字中的第一位指定了响应类别，后两位无分类。响应类别有以下 5 种。

表 4-1：状态码的类别

	类别	原因短语
1XX	Informational（信息性状态码）	接收的请求正在处理
2XX	Success（成功状态码）	请求正常处理完毕
3XX	Redirection（重定向状态码）	需要进行附加操作以完成请求
4XX	Client Error（客户端错误状态码）	服务器无法处理请求
5XX	Server Error（服务器错误状态码）	服务器处理请求出错

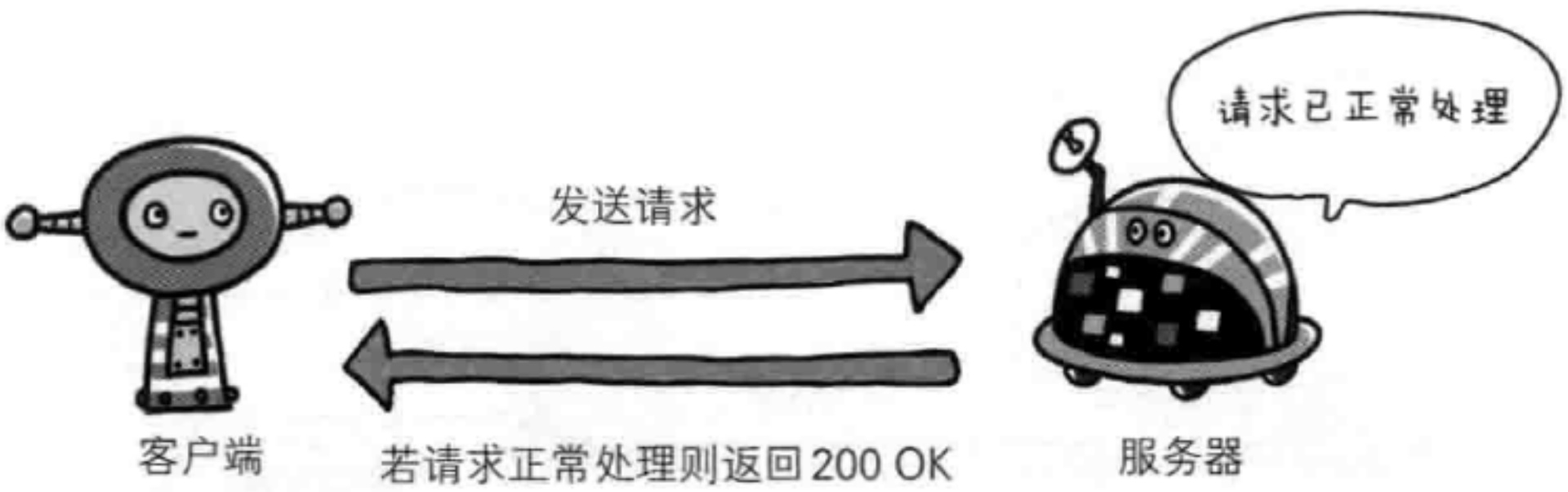
只要遵守状态码类别的定义，即使改变RFC2616中定义的状态码，或服务器端自行创建状态码都没问题。

仅记录在RFC2616上的HTTP状态码就达40种，若再加上WebDAV（Web-based Distributed Authoring and Versioning，基于万维网的分布式创作和版本控制）（RFC4918、5842）和附加HTTP状态码（RFC6585）等扩展，数量就达60余种。别看种类繁多，实际上经常使用的大概只有14种。接下来，我们就介绍一下这些具有代表性的14个状态码。

4.2　2XX 成功

2XX的响应结果表明请求被正常处理了。

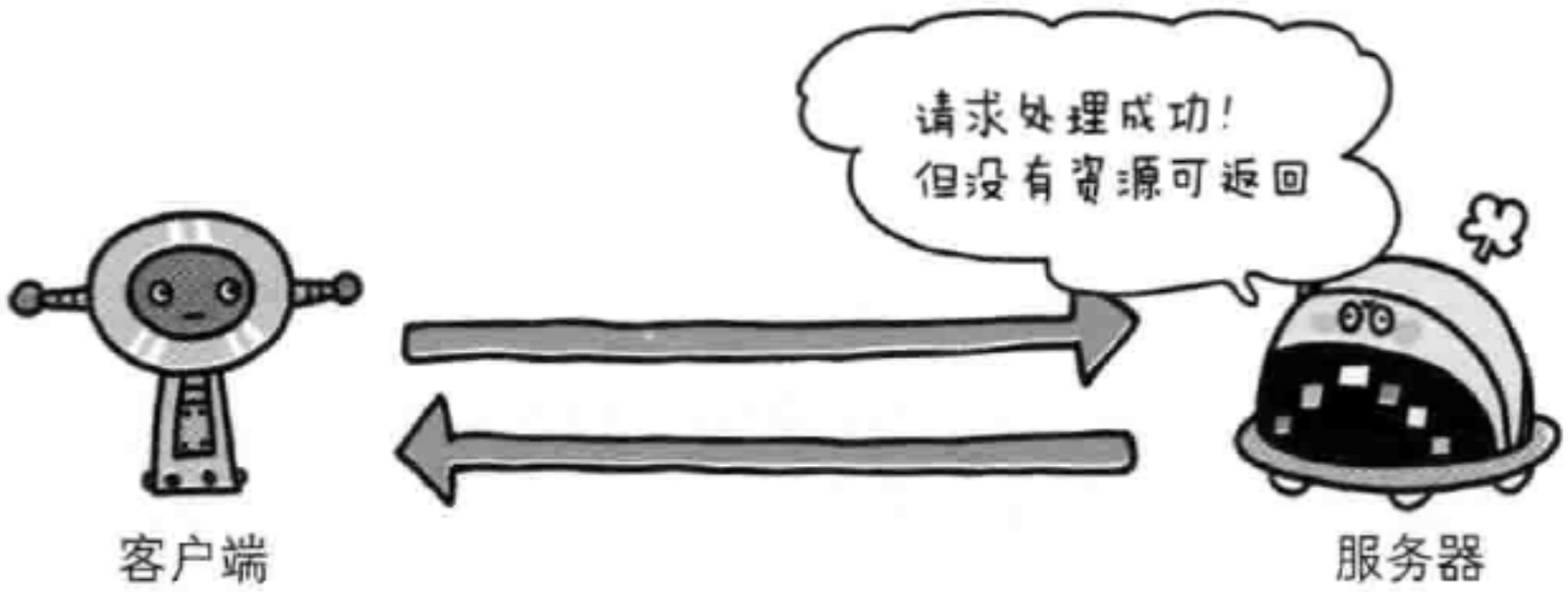
4.2.1　200 OK



表示从客户端发来的请求在服务器端被正常处理了。

在响应报文内，随状态码一起返回的信息会因方法的不同而发生改变。比如，使用GET方法时，对应请求资源的实体会作为响应返回；而使用HEAD方法时，对应请求资源的实体首部不随报文主体作为响应返回（即在响应中只返回首部，不会返回实体的主体部分）。

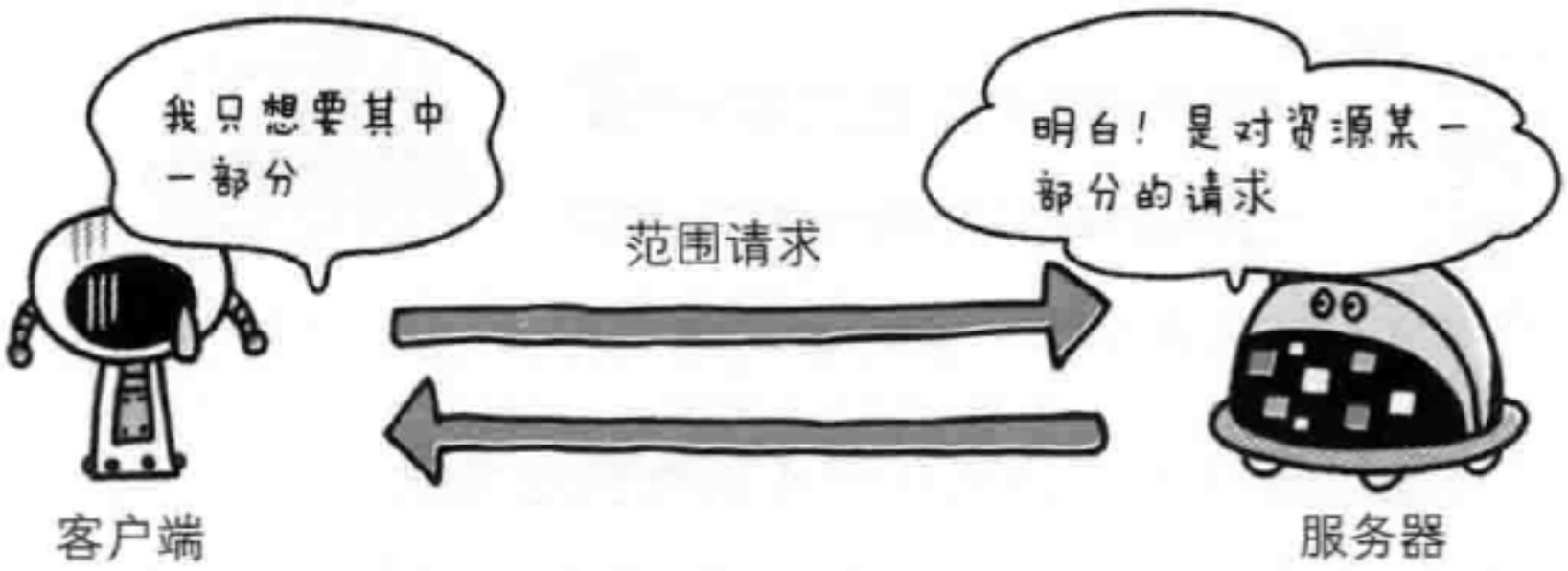
4.2.2 204 No Content



该状态码代表服务器接收的请求已成功处理，但在返回的响应报文中不含实体的主体部分。另外，也不允许返回任何实体的主体。比如，当从浏览器发出请求处理后，返回 204 响应，那么浏览器显示的页面不发生更新。

一般在只需要从客户端往服务器发送信息，而对客户端不需要发送新信息内容的情况下使用。

4.2.3 206 Partial Content

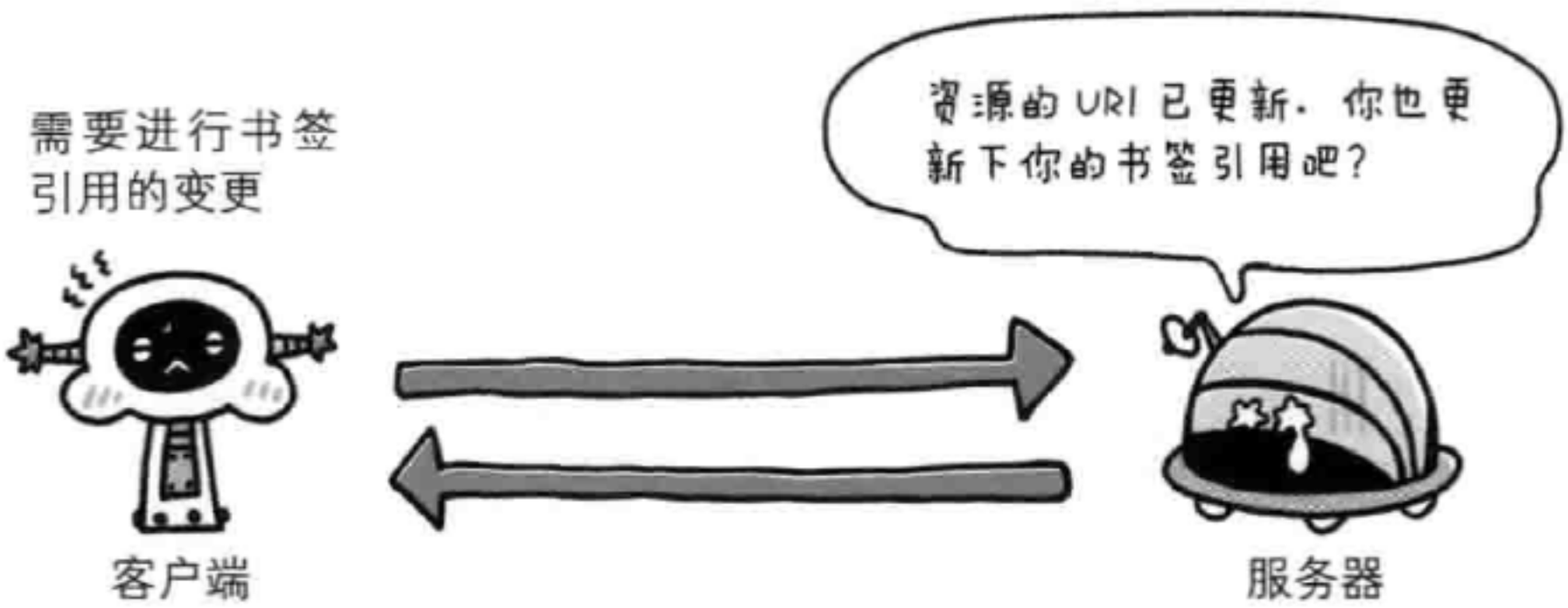


该状态码表示客户端进行了范围请求，而服务器成功执行了这部分的 GET 请求。响应报文中包含由 Content-Range 指定范围的实体内容。

4.3 3XX 重定向

3XX 响应结果表明浏览器需要执行某些特殊的处理以正确处理请求。

4.3.1　301 Moved Permanently

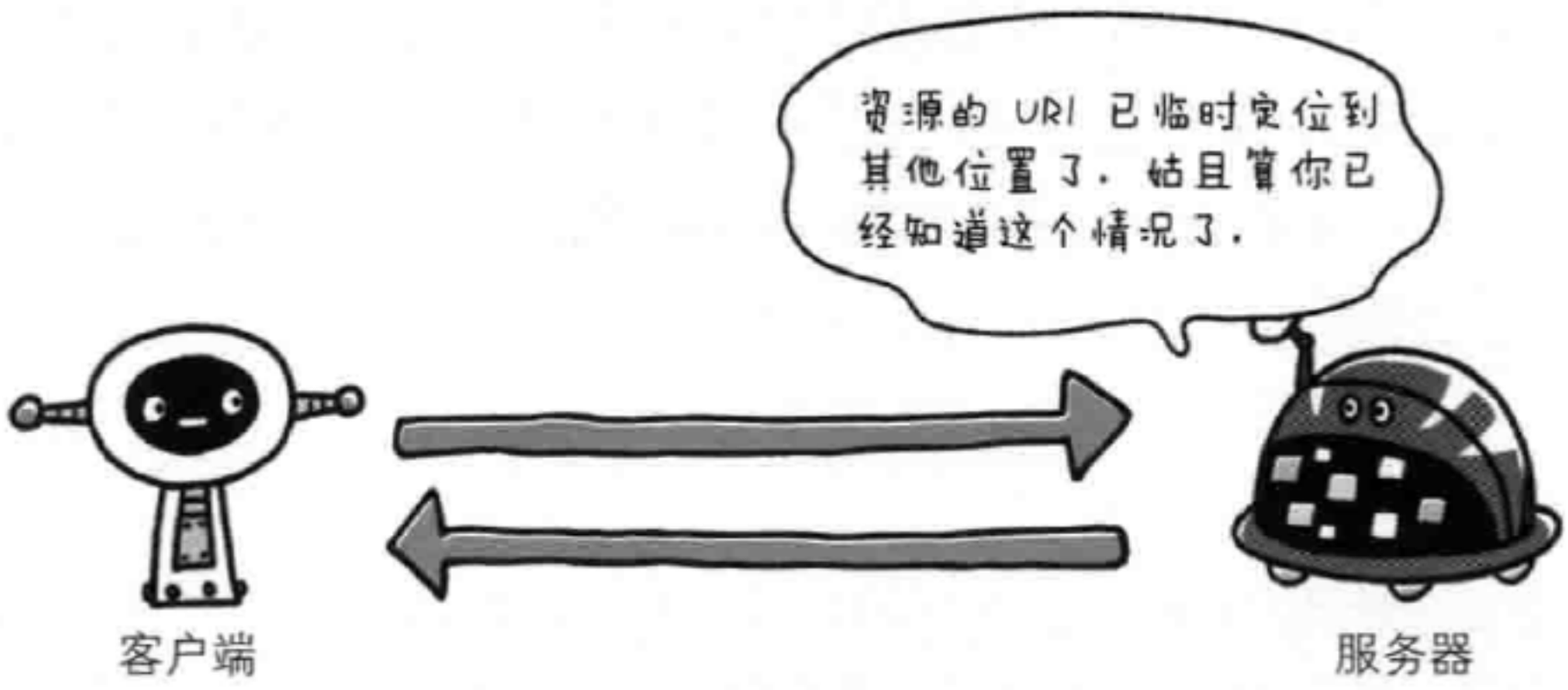


永久性重定向。该状态码表示请求的资源已被分配了新的 URI，以后应使用资源现在所指的 URI。也就是说，如果已经把资源对应的 URI 保存为书签了，这时应该按 Location 首部字段提示的 URI 重新保存。

像下方给出的请求 URI，当指定资源路径的最后忘记添加斜杠“/”，就会产生 301 状态码。


```
http://example.com/sample
```

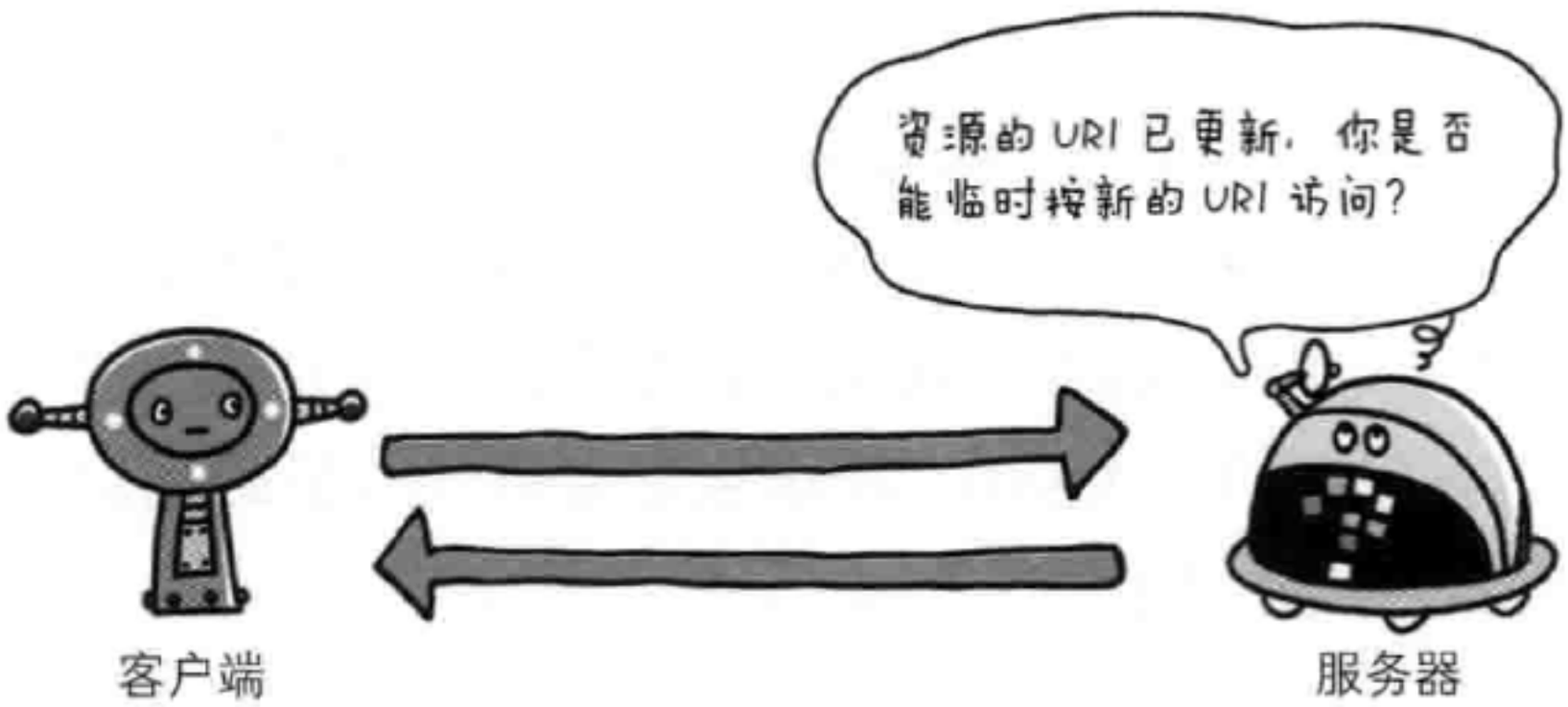
4.3.2　302 Found



临时性重定向。该状态码表示请求的资源已被分配了新的 URI，希望用户（本次）能使用新的 URI 访问。

和 301 Moved Permanently 状态码相似，但 302 状态码代表的资源不是被永久移动，只是临时性质的。换句话说，已移动的资源对应的 URI 将来还有可能发生改变。比如，用户把 URI 保存成书签，但不会像 301 状态码出现时那样去更新书签，而是仍旧保留返回 302 状态码的页面对应的 URI。

4.3.3 303 See Other



该状态码表示由于请求对应的资源存在着另一个 URI，应使用 GET 方法定向获取请求的资源。

303 状态码和 302 Found 状态码有着相同的功能，但 303 状态码明确表示客户端应当采用 GET 方法获取资源，这点与 302 状态码有区别。

比如，当使用 POST 方法访问 CGI 程序，其执行后的处理结果是希望客户端能以 GET 方法重定向到另一个 URI 上去时，返回 303 状态码。虽然 302 Found 状态码也可以实现相同的功能，但这里使用 303 状态码是最理想的。[①]

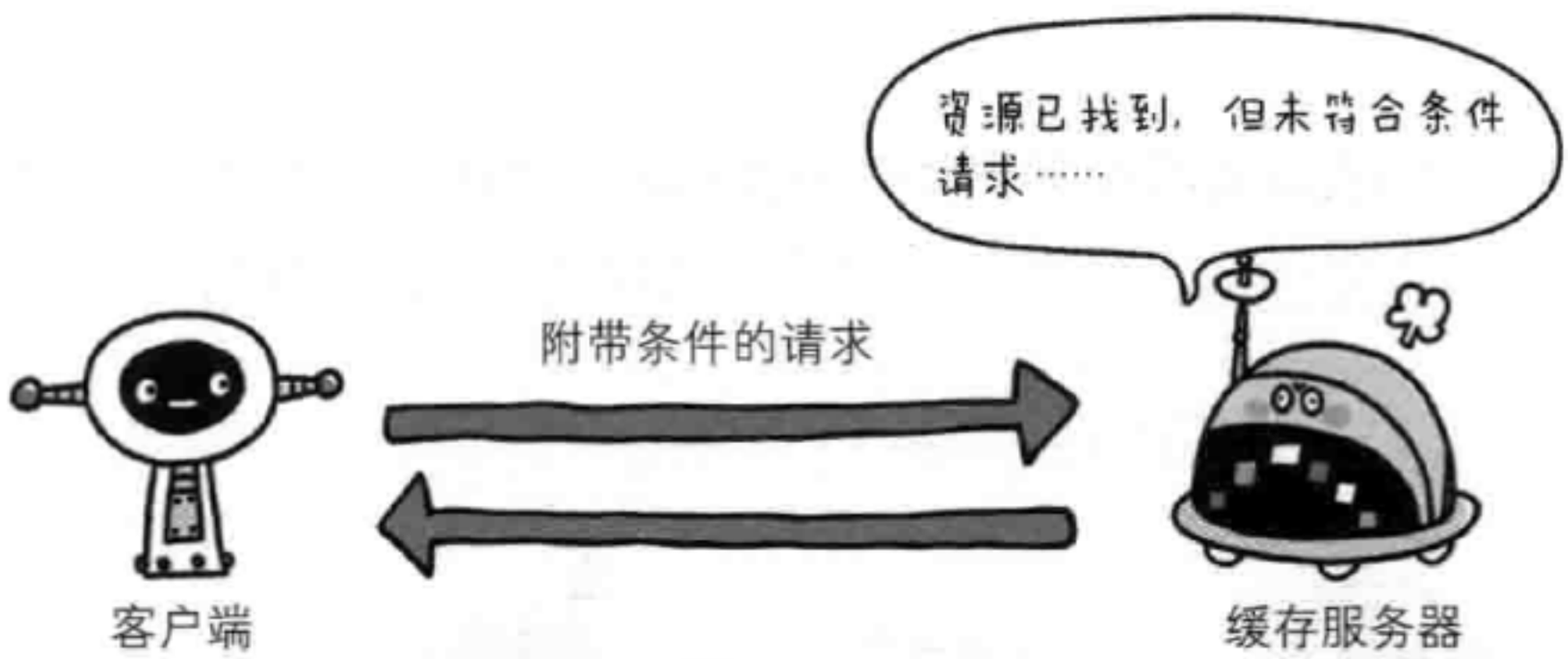
---

① 本书采用的是HTTP/1.1，而许多HTTP/1.1版以前的浏览器不能正确理解303状态码。虽然RFC 1945和RFC 2068规范不允许客户端在重定向时改变请求的方法，但是很多现存的浏览器将302响应视为303响应，并且使用GET方式访问在Location中规定的URI，而无视原先请求的方法。所以作者说这里使用303是最理想的。——译者注

当 301、302、303 响应状态码返回时，几乎所有的浏览器都会把 POST 改成 GET，并删除请求报文内的主体，之后请求会自动再次发送。

301、302 标准是禁止将 POST 方法改变成 GET 方法的，但实际使用时大家都会这么做。

4.3.4　304 Not Modified



该状态码表示客户端发送附带条件的请求[①]时，服务器端允许请求访问资源，但未满足条件的情况。304 状态码返回时，不包含任何响应的主体部分。304 虽然被划分在 3XX 类别中，但是和重定向没有关系。

4.3.5　307 Temporary Redirect

临时重定向。该状态码与 302 Found 有着相同的含义。尽管 302 标准禁止 POST 变换成 GET，但实际使用时大家并不遵守。

307 会遵照浏览器标准，不会从 POST 变成 GET。但是，对于处理响应时的行为，每种浏览器有可能出现不同的情况。

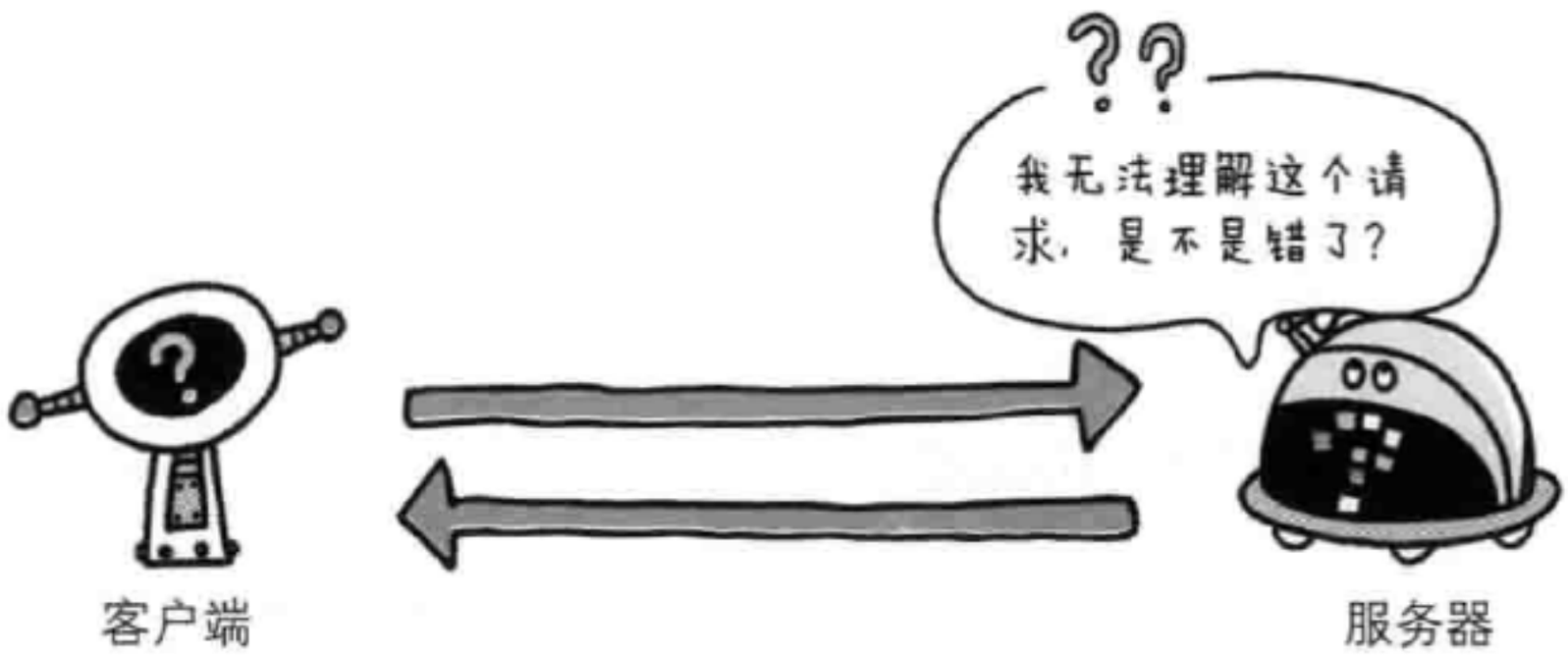
---

① 附带条件的请求是指采用GET方法的请求报文中包含If-Match，If-Modified-Since，If-None-Match，If-Range，If-Unmodified-Since中任一首部。

4.4 4XX 客户端错误

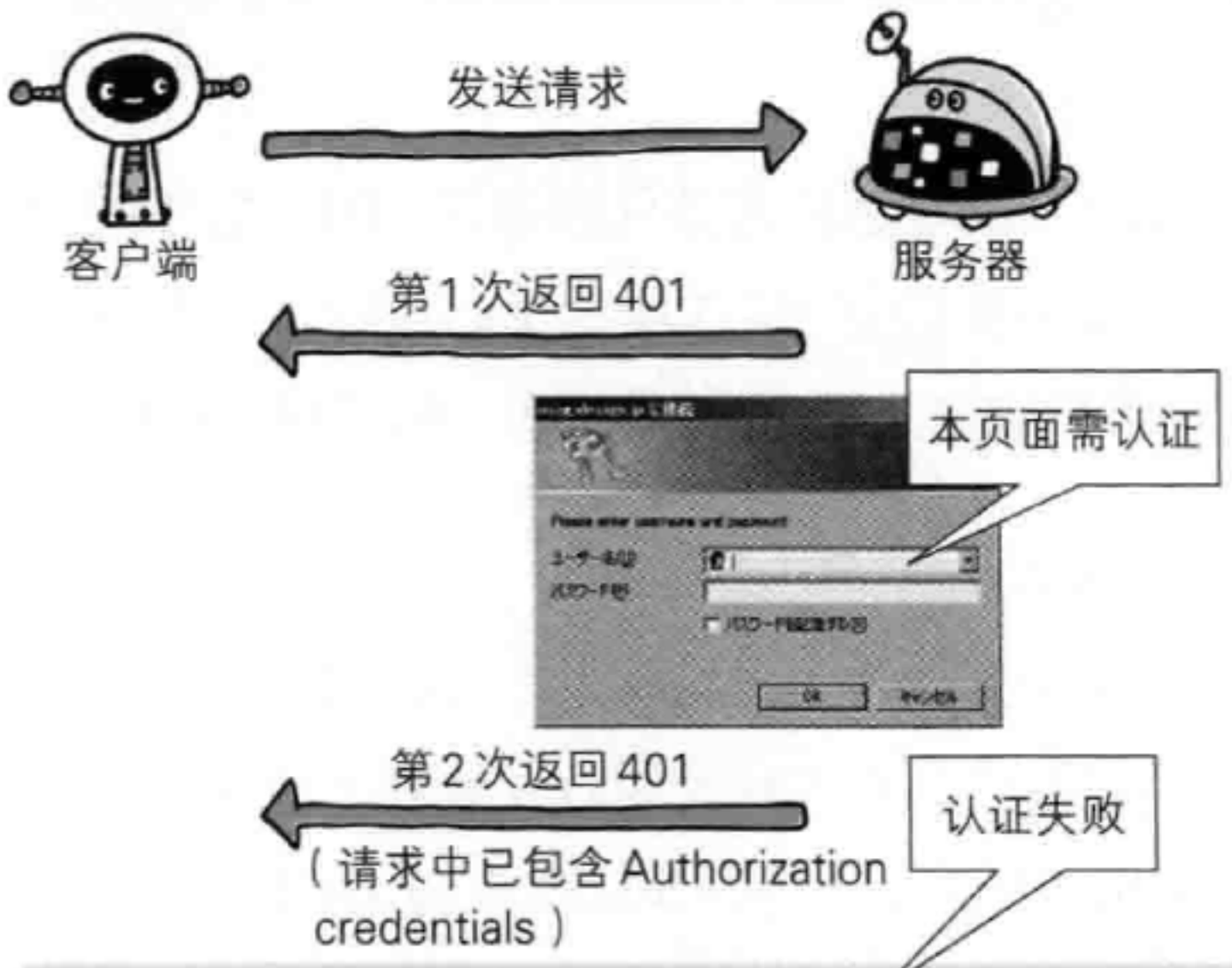
4XX 的响应结果表明客户端是发生错误的原因所在。

4.4.1 400 Bad Request



该状态码表示请求报文中存在语法错误。当错误发生时，需修改请求的内容后再次发送请求。另外，浏览器会像 200 OK 一样对待该状态码。

4.4.2 401 Unauthorized

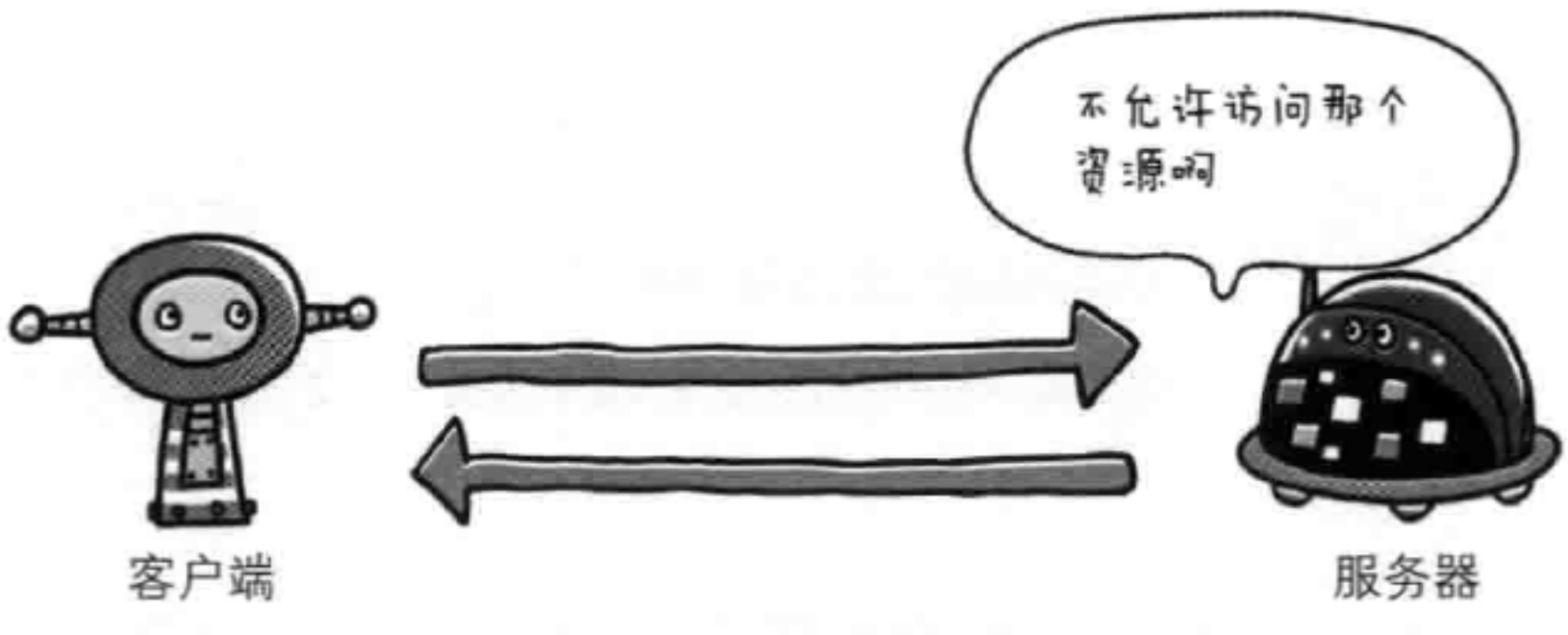



```
Authorization Required
This server could not verify that you are authorized to
access the document requested. Either you supplied the
wrong credentials (e.g., bad password), or your browser
doesn't understand how to supply the credentials required.
```

该状态码表示发送的请求需要有通过 HTTP 认证（BASIC 认证、DIGEST 认证）的认证信息。另外若之前已进行过 1 次请求，则表示用户认证失败。

返回含有 401 的响应必须包含一个适用于被请求资源的 WWW-Authenticate 首部用以质询（challenge）用户信息。当浏览器初次接收到 401 响应，会弹出认证用的对话窗口。

4.4.3　403 Forbidden



该状态码表明对请求资源的访问被服务器拒绝了。服务器端没有必要给出拒绝的详细理由，但如果想作说明的话，可以在实体的主体部分对原因进行描述，这样就能让用户看到了。

未获得文件系统的访问授权，访问权限出现某些问题（从未授权的发送源 IP 地址试图访问）等列举的情况都可能是发生 403 的原因。

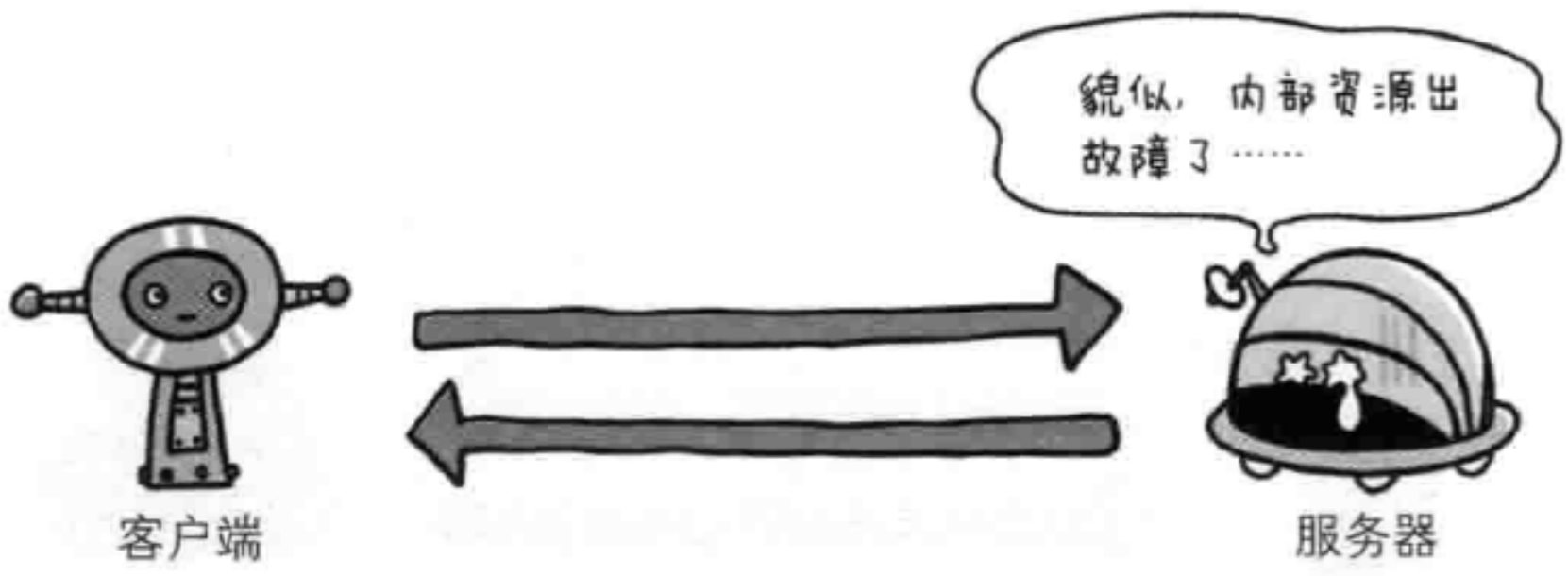
4.4.4　404 Not Found

该状态码表明服务器上无法找到请求的资源。除此之外，也可以在服务器端拒绝请求且不想说明理由时使用。

4.5 5XX 服务器错误

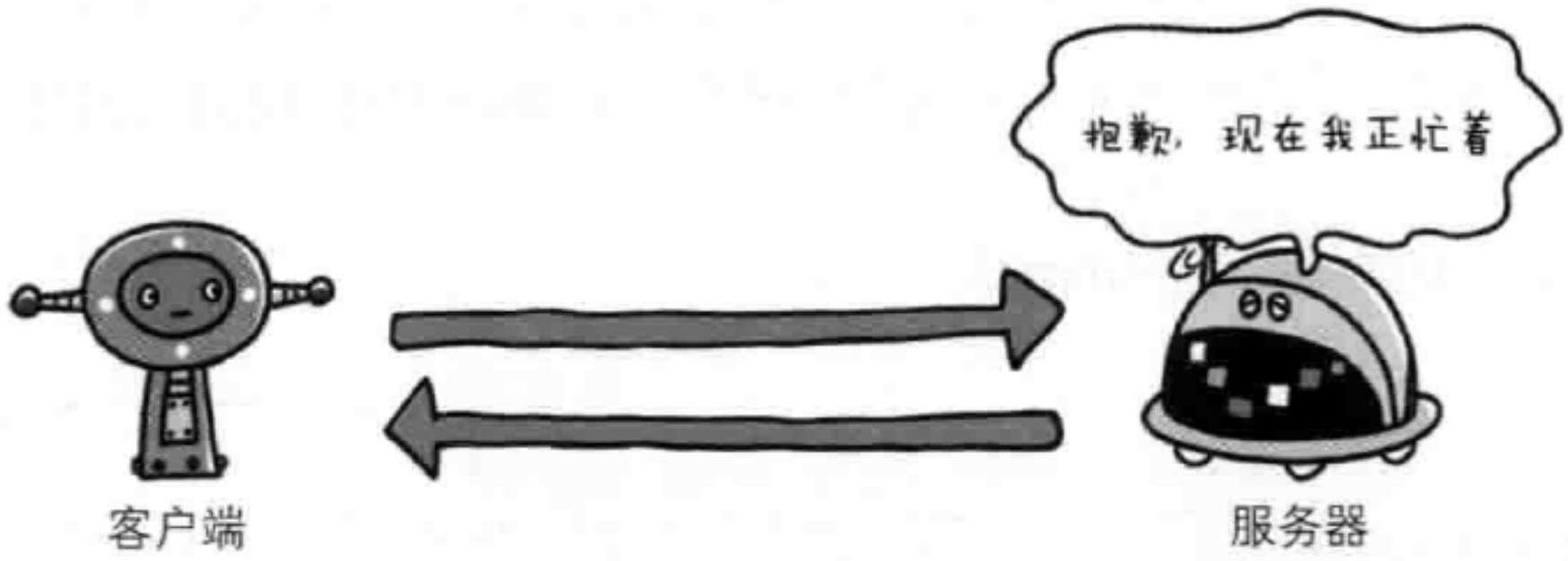
5XX 的响应结果表明服务器本身发生错误。

4.5.1 500 Internal Server Error



该状态码表明服务器端在执行请求时发生了错误。也有可能是 Web 应用存在的 bug 或某些临时的故障。

4.5.2 503 Service Unavailable



该状态码表明服务器暂时处于超负载或正在进行停机维护，现在无法处理请求。如果事先得知解除以上状况需要的时间，最好写入 Retry-After 首部字段再返回给客户端。

状态码和状况的不一致

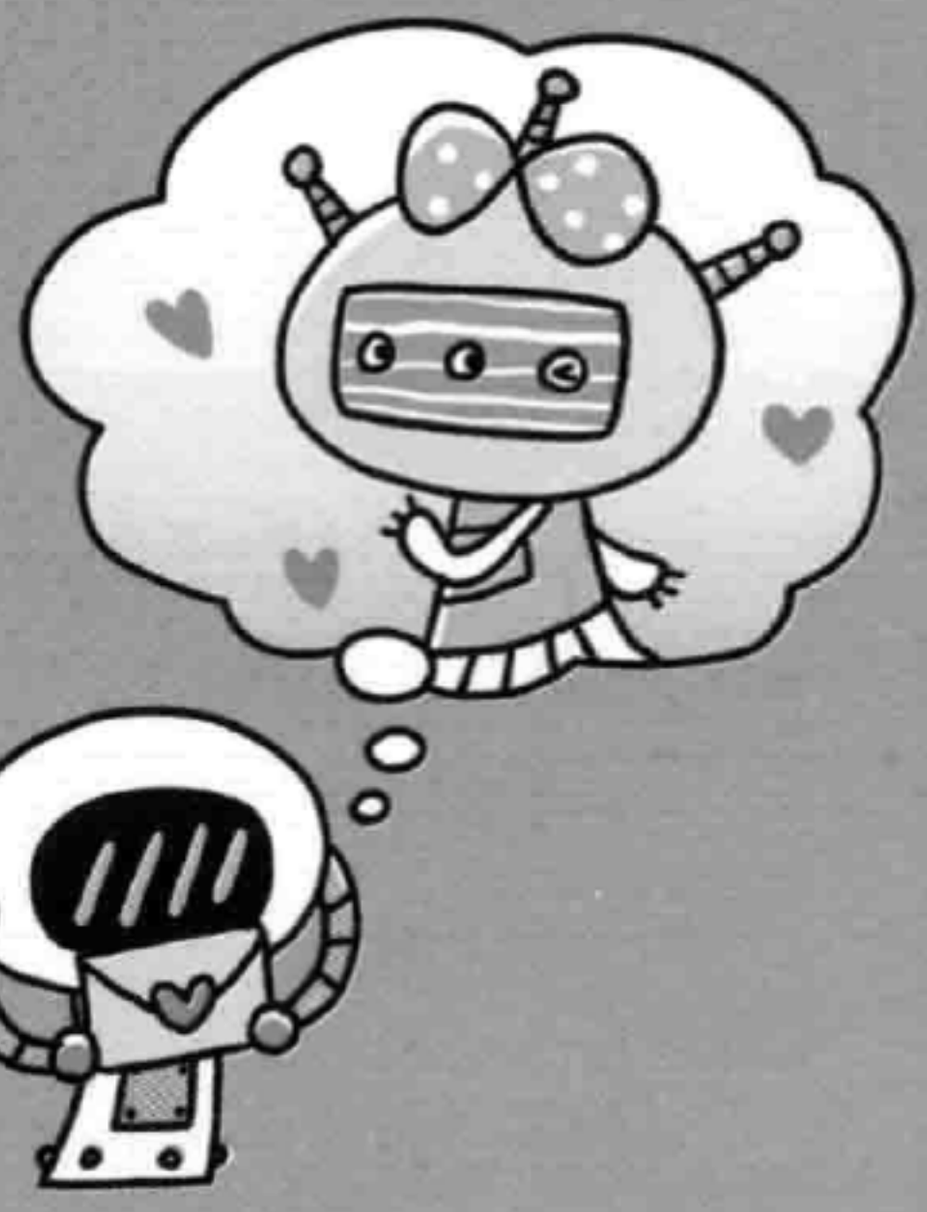
不少返回的状态码响应都是错误的，但是用户可能察觉不到这点。比如Web应用程序内部发生错误，状态码依然返回200 OK，这种情况也经常遇到。

Chapter5

第5章

与HTTP协作的Web服务器

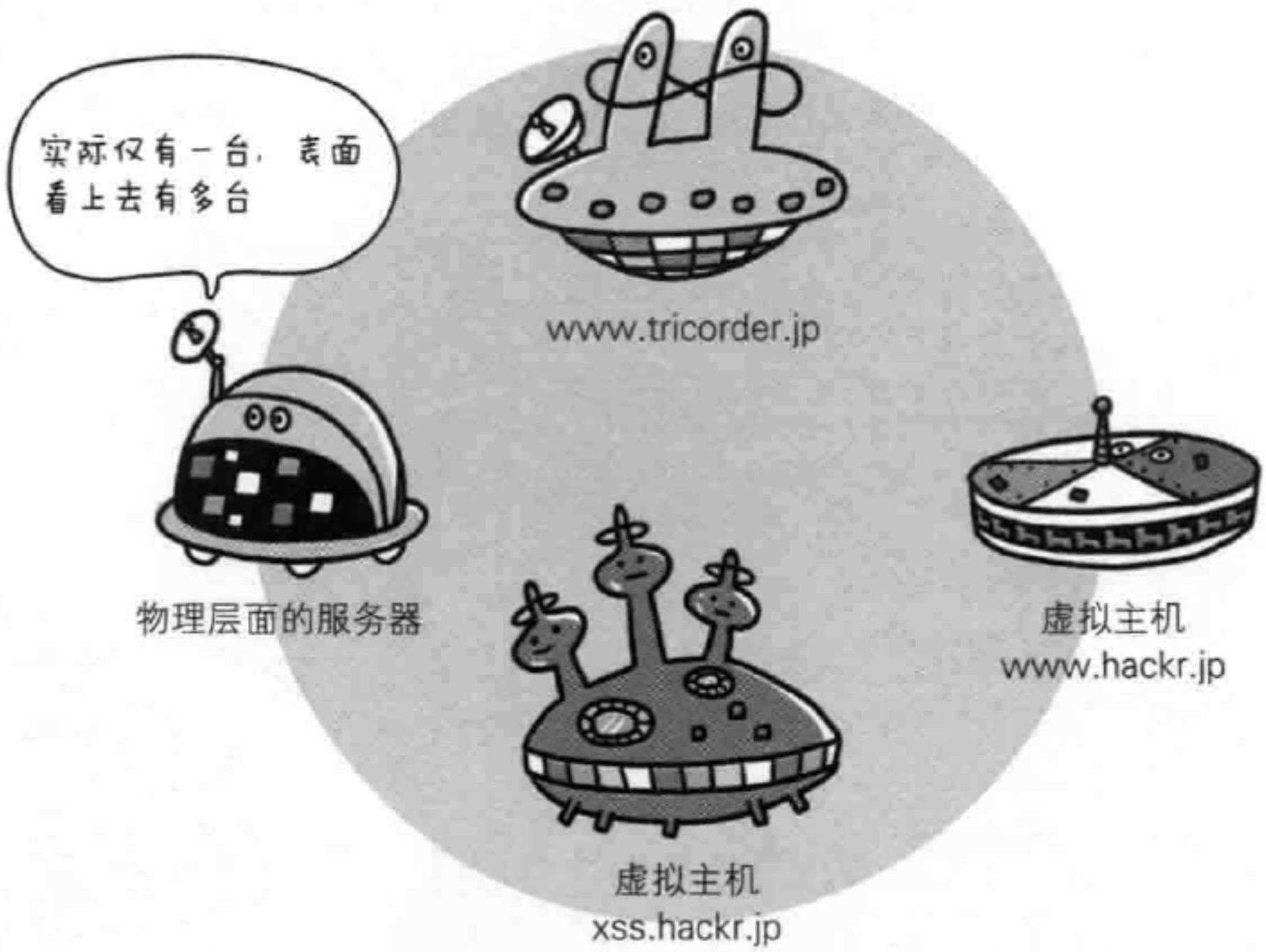
一台Web服务器可搭建多个独立域名的Web网站，也可作为通信路径上的中转服务器提升传输效率。



5.1 用单台虚拟主机实现多个域名

HTTP/1.1 规范允许一台 HTTP 服务器搭建多个 Web 站点。比如，提供 Web 托管服务（Web Hosting Service）的供应商，可以用一台服务器为多位客户服务，也可以以每位客户持有的域名运行各自不同的网站。这是因为利用了虚拟主机（Virtual Host，又称虚拟服务器）的功能。

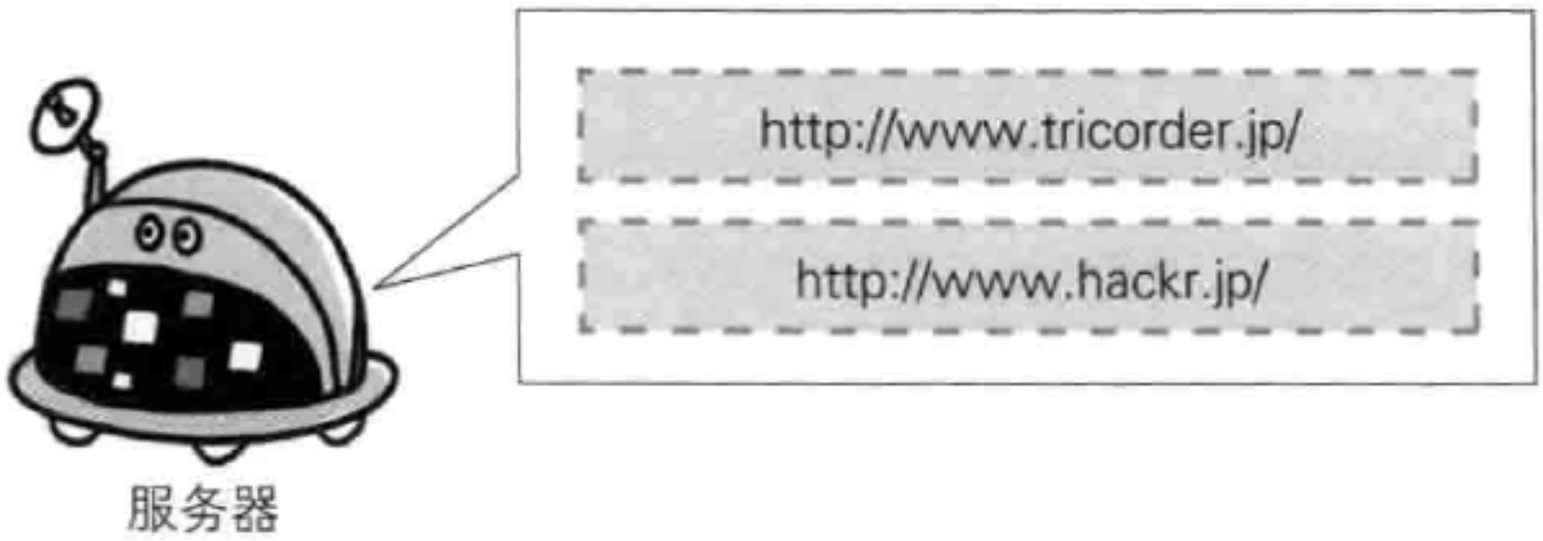
即使物理层面只有一台服务器，但只要使用虚拟主机的功能，则可以假想已具有多台服务器。



客户端使用 HTTP 协议访问服务器时，会经常采用类似 www.hackr.jp 这样的主机名和域名。

在互联网上，域名通过 DNS 服务映射到 IP 地址（域名解析）之后访问目标网站。可见，当请求发送到服务器时，已经是以 IP 地址形式访问了。

所以，如果一台服务器内托管了 www.tricorder.jp 和 www.hackr.jp 这两个域名，当收到请求时就需要弄清楚究竟要访问哪个域名。



若www.tricorder.jp和www.hackr.jp同时部署在同一个服务器上（相同的IP地址），使用DNS服务解析域名后，两者的访问IP地址会相同。

http://203.189.105.112/

在相同的 IP 地址下，由于虚拟主机可以寄存多个不同主机名和域名的 Web 网站，因此在发送 HTTP 请求时，必须在 Host 首部内完整指定主机名或域名的 URI。

5.2　通信数据转发程序：代理、网关、隧道

HTTP 通信时，除客户端和服务器以外，还有一些用于通信数据转发的应用程序，例如代理、网关和隧道。它们可以配合服务器工作。

这些应用程序和服务器可以将请求转发给通信线路上的下一站服务器，并且能接收从那台服务器发送的响应再转发给客户端。

代理

代理是一种有转发功能的应用程序，它扮演了位于服务器和客户端“中间人”的角色，接收由客户端发送的请求并转发给服务器，同时也接收服务器返回的响应并转发给客户端。

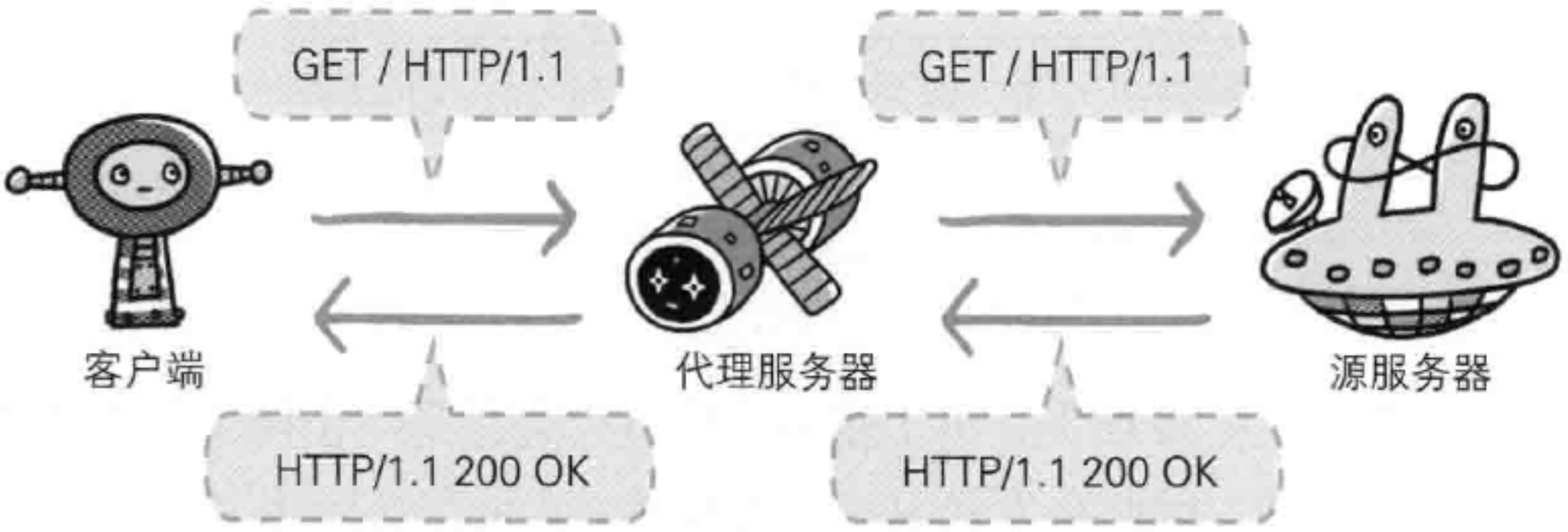
网关

网关是转发其他服务器通信数据的服务器，接收从客户端发送来的请求时，它就像自己拥有资源的源服务器一样对请求进行处理。有时客户端可能都不会察觉，自己的通信目标是一个网关。

隧道

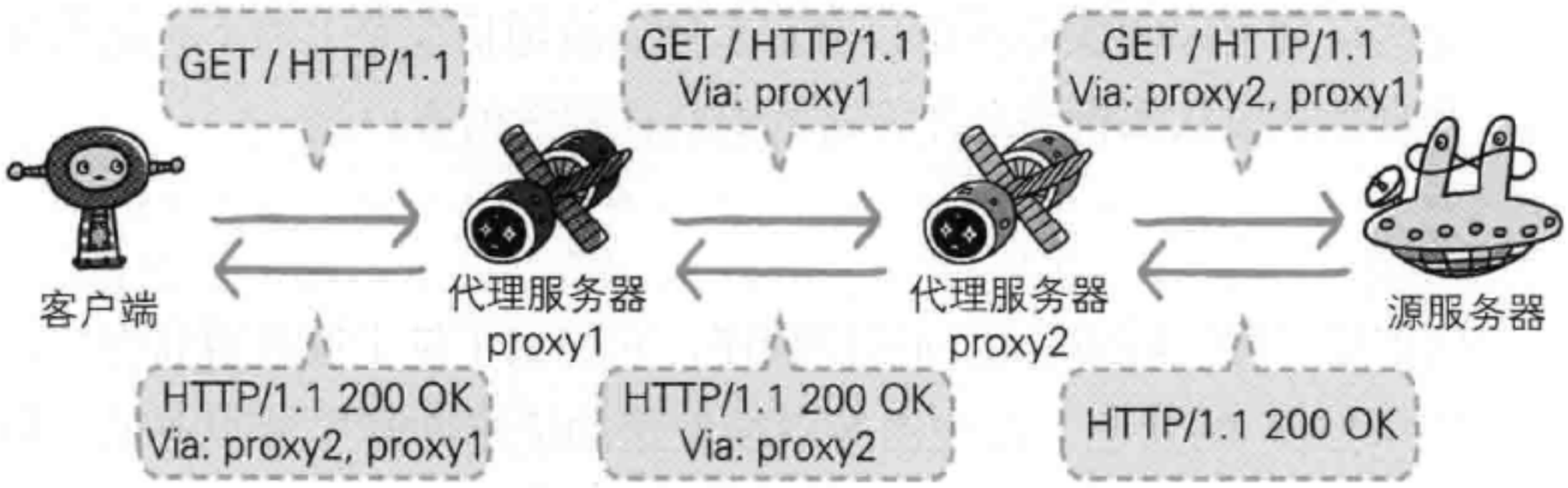
隧道是在相隔甚远的客户端和服务器两者之间进行中转，并保持双方通信连接的应用程序。

5.2.1 代理



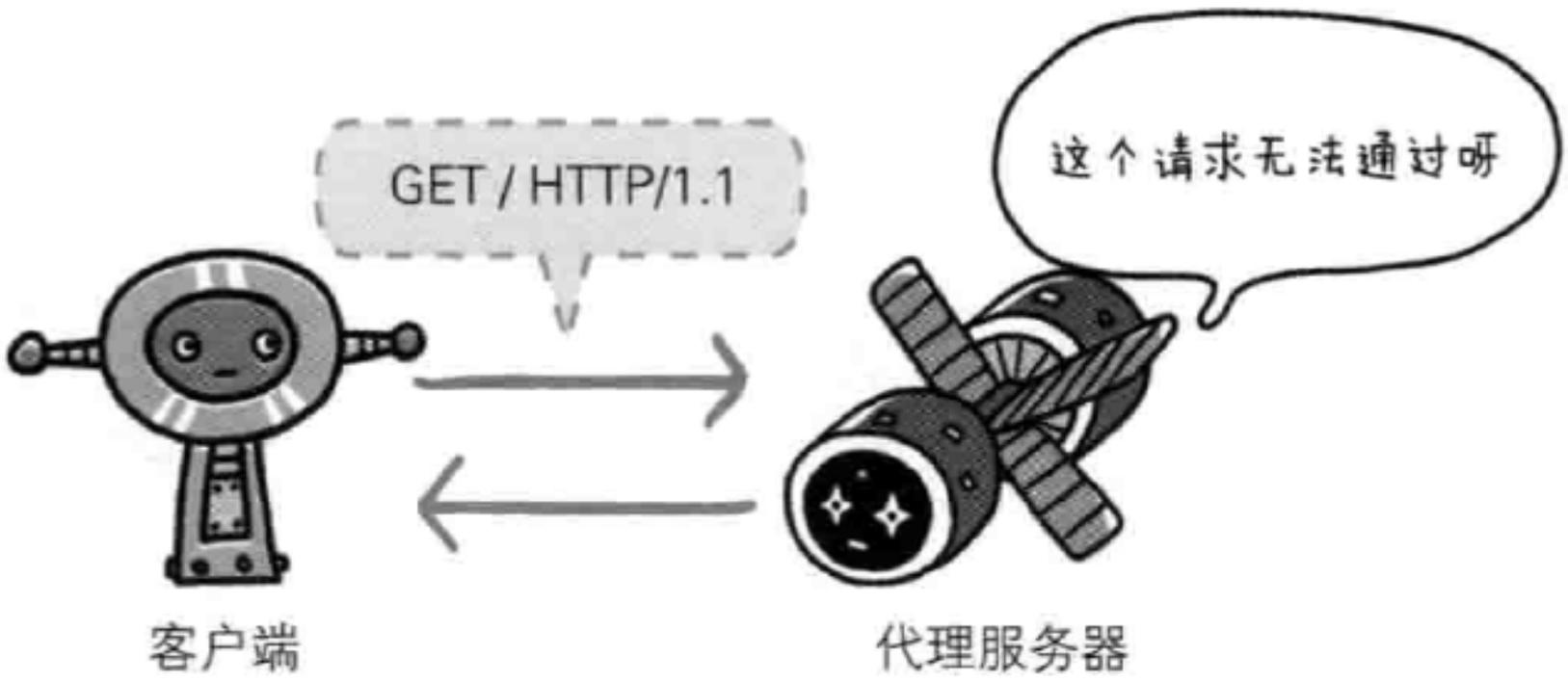
代理服务器的基本行为就是接收客户端发送的请求后转发给其他服务器。代理不改变请求 URI，会直接发送给前方持有资源的目标服务器。

持有资源实体的服务器被称为源服务器。从源服务器返回的响应经过代理服务器后再传给客户端。



图：每次通过代理服务器转发请求或响应时，会追加写入 Via 首部信息

在 HTTP 通信过程中，可级联多台代理服务器。请求和响应的转发会经过数台类似锁链一样连接起来的代理服务器。转发时，需要附加 Via 首部字段以标记出经过的主机信息。



通过设置组织内部的代理服务器可做到针对特定URI访问的控制。

使用代理服务器的理由有：利用缓存技术（稍后讲解）减少网络带宽的流量，组织内部针对特定网站的访问控制，以获取访问日志为主要目的，等等。

代理有多种使用方法，按两种基准分类。一种是是否使用缓存，另一种是是否会修改报文。

缓存代理

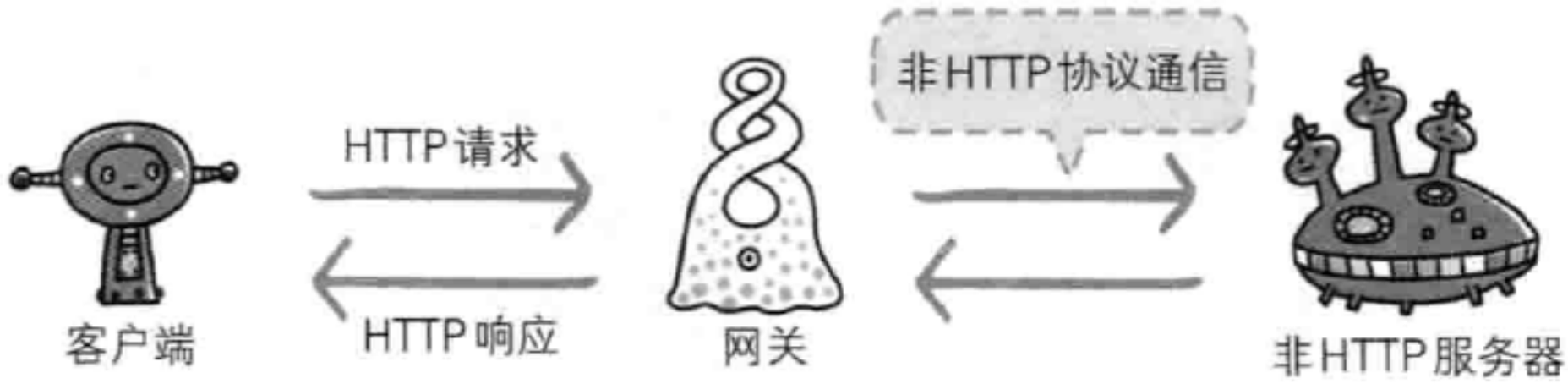
代理转发响应时，缓存代理（Caching Proxy）会预先将资源的副本（缓存）保存在代理服务器上。

当代理再次接收到对相同资源的请求时，就可以不从源服务器那里获取资源，而是将之前缓存的资源作为响应返回。

透明代理

转发请求或响应时，不对报文做任何加工的代理类型被称为透明代理（Transparent Proxy）。反之，对报文内容进行加工的代理被称为非透明代理。

5.2.2 网关



图：利用网关可以由 HTTP 请求转化为其他协议通信

网关的工作机制和代理十分相似。而网关能使通信线路上的服务器提供非 HTTP 协议服务。

利用网关能提高通信的安全性，因为可以在客户端与网关之间的通信线路上加密以确保连接的安全。比如，网关可以连接数据库，使用 SQL 语句查询数据。另外，在 Web 购物网站上进行信用卡结算时，网关可以和信用卡结算系统联动。

5.2.3 隧道

隧道可按要求建立起一条与其他服务器的通信线路，届时使用 SSL 等加密手段进行通信。隧道的目的是确保客户端能与服务器进行安全的通信。

隧道本身不会去解析 HTTP 请求。也就是说，请求保持原样中转给之后的服务器。隧道会在通信双方断开连接时结束。

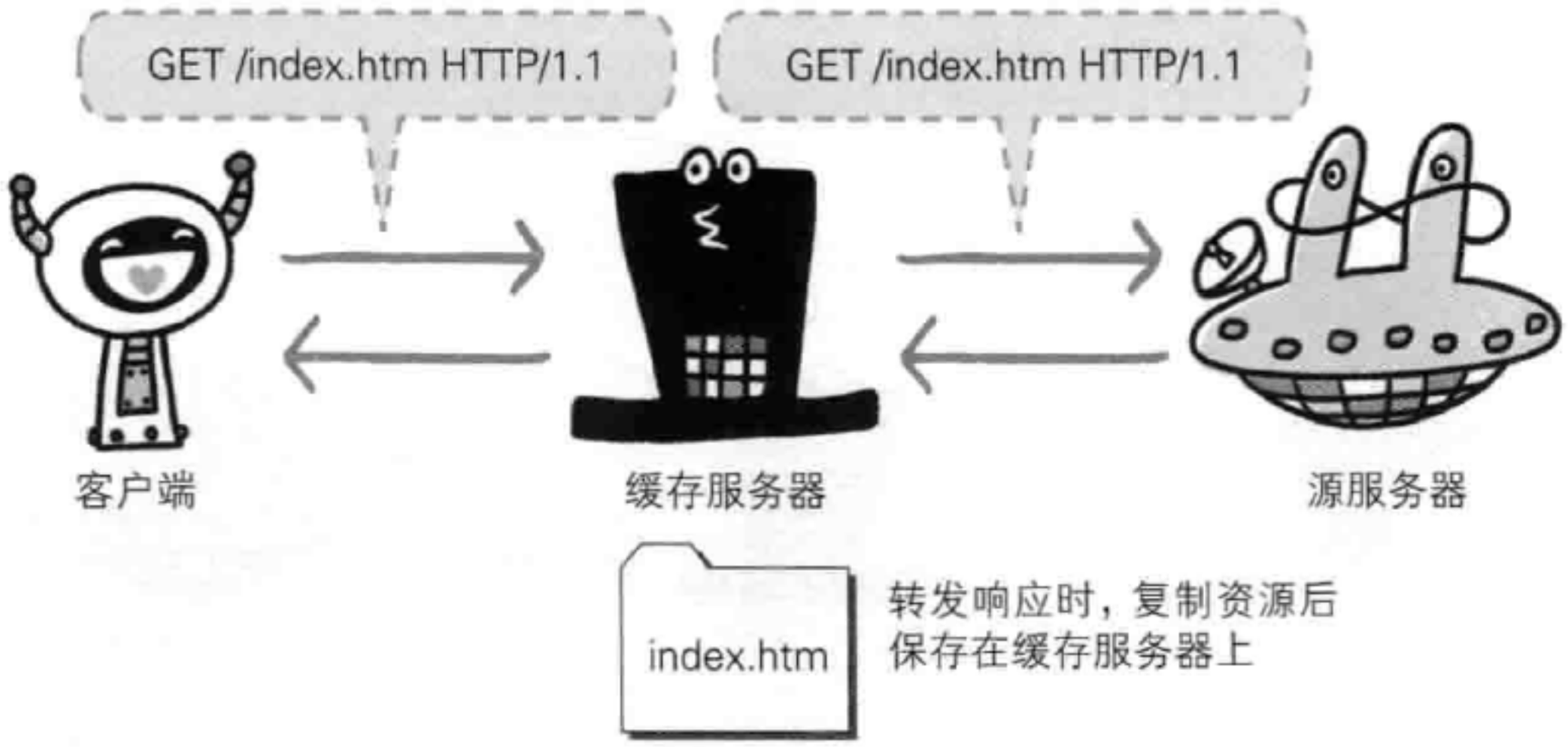


图：通过隧道的传输，可以和远距离的服务器安全通信。隧道本身是透明的，客户端不用在意隧道的存在

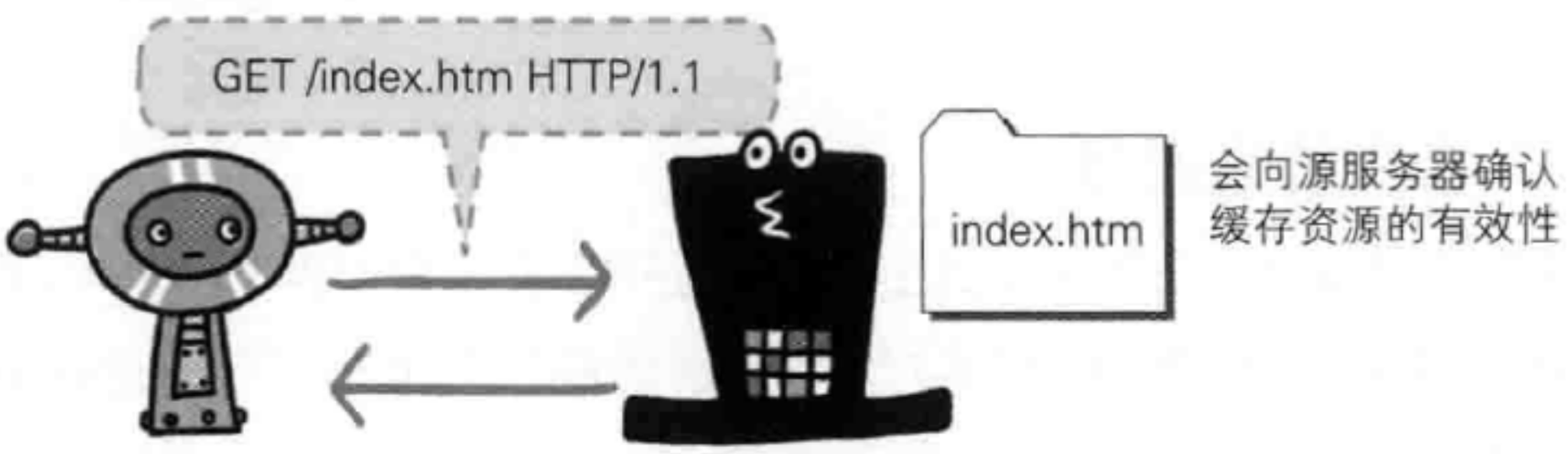
5.3　保存资源的缓存

缓存是指代理服务器或客户端本地磁盘内保存的资源副本。利用缓存可减少对源服务器的访问，因此也就节省了通信流量和通信时间。

缓存服务器是代理服务器的一种，并归类在缓存代理类型中。换句话说，当代理转发从服务器返回的响应时，代理服务器将会保存一份资源的副本。



请求的资源如果已经被缓存则直接由缓存服务器返回给客户端



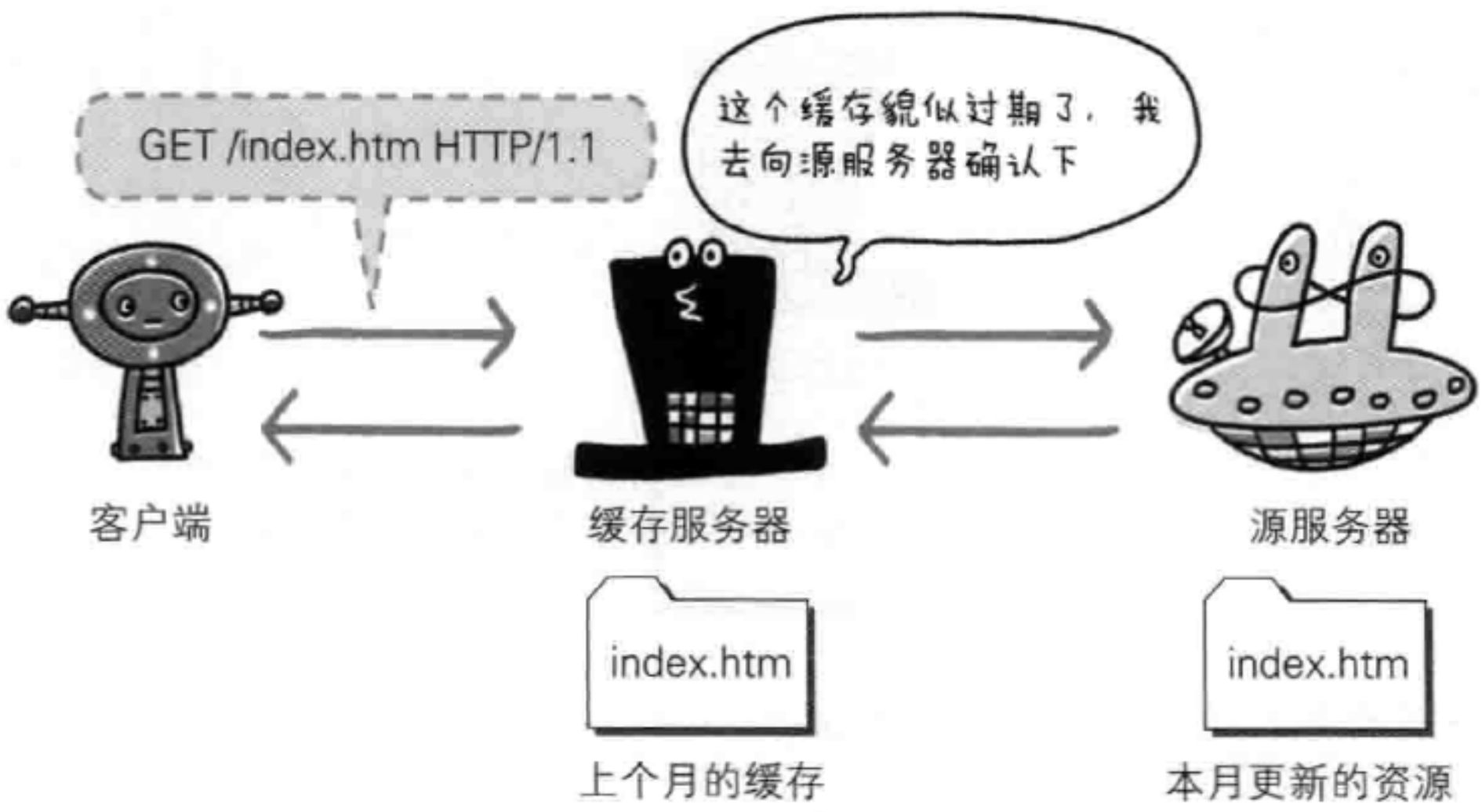
缓存服务器的优势在于利用缓存可避免多次从源服务器转发资源。因此客户端可就近从缓存服务器上获取资源，而源服务器也不必多次处理相同的请求了。

5.3.1 缓存的有效期限

即便缓存服务器内有缓存，也不能保证每次都会返回对同资源的请求。因为这关系到被缓存资源的有效性问题。

当遇上源服务器上的资源更新时，如果还是使用不变的缓存，那就会演变成返回更新前的“旧”资源了。

即使存在缓存，也会因为客户端的要求、缓存的有效期等因素，向源服务器确认资源的有效性。若判断缓存失效，缓存服务器将会再次从源服务器上获取“新”资源。

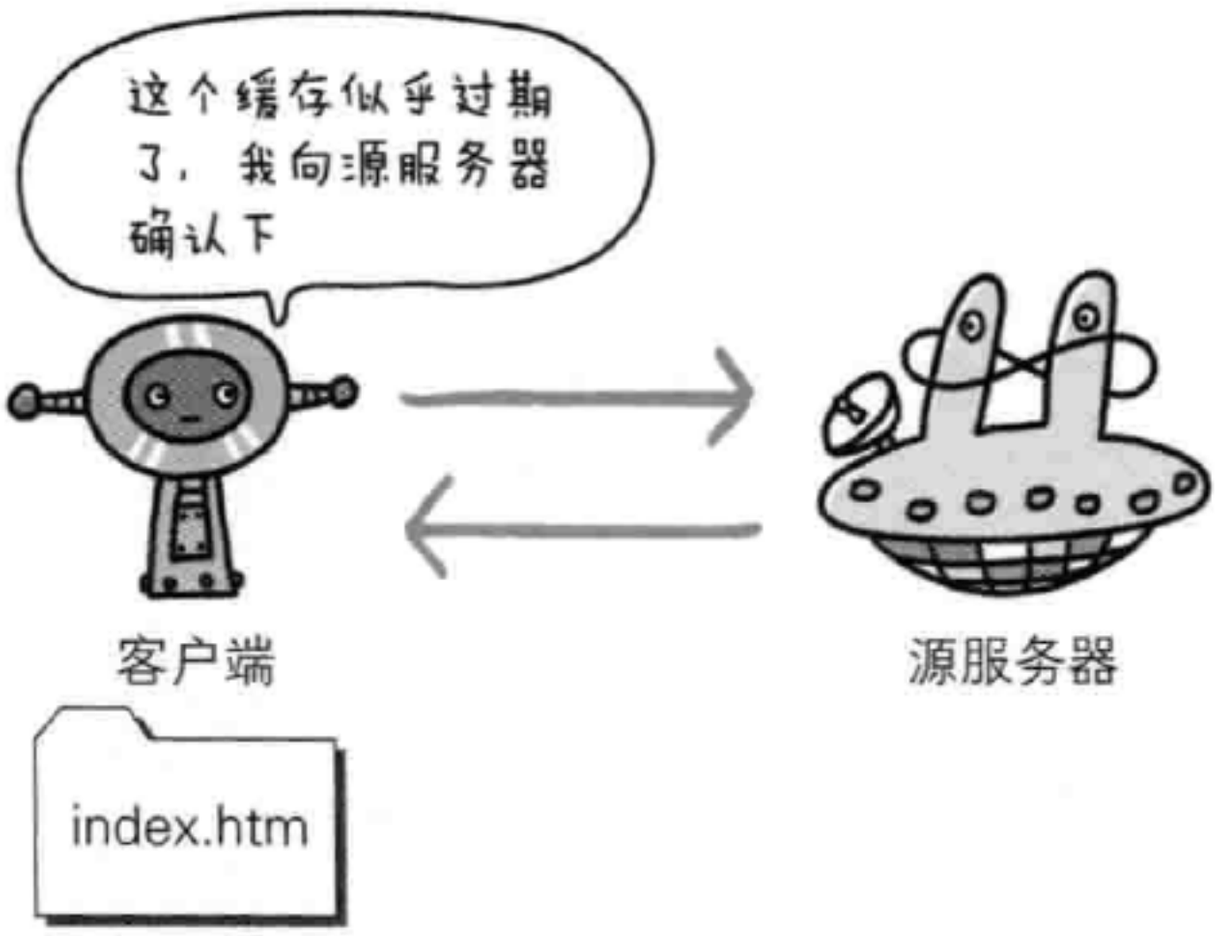


5.3.2 客户端的缓存

缓存不仅可以存在于缓存服务器内，还可以存在客户端浏览器中。以 Internet Explorer 程序为例，把客户端缓存称为临时网络文件（Temporary Internet File）。

浏览器缓存如果有效，就不必再向服务器请求相同的资源了，可以直接从本地磁盘内读取。

另外，和缓存服务器相同的一点是，当判定缓存过期后，会向源服务器确认资源的有效性。若判断浏览器缓存失效，浏览器会再次请求新资源。



在 HTTP 出现之前的协议

在 HTTP 普及之前，也就是从互联网的诞生期至今，曾出现过各式各样的协议。在 HTTP 规范确立之际，制定者们参考了那些协议的功能。也有某些协议现在已经彻底退出了人们的视线。接下来，我们会简单介绍一下这些协议。

FTP (File Transfer Protocol)

传输文件时使用的协议。该协议历史久远，可追溯到 1973 年前后，比 TCP/IP 协议族的出现还要早。虽然它在 1995 年被 HTTP 的流量 (Traffic) 超越，但时至今日，仍被广泛沿用。

NNTP (Network News Transfer Protocol)

用于 NetNews 电子会议室内传送消息的协议。在 1986 年前后出现，属于比较古老的一类协议。现在，利用 Web 交换信息已成主流，所以该协议已经不怎么使用了。

Archie

搜索 anonymous FTP 公开的文件信息的协议。1990 年前后出现，现在已经不常使用。

WAIS (Wide Area Information Servers)

以关键词检索多个数据库使用的协议。1991 年前后出现。由于现在已经被 HTTP 协议替代，也已经不怎么使用了。

Gopher

查找与互联网连接的计算机内信息的协议。1991 年前后出现，由于现在已经被 HTTP 协议替代，也已经不怎么使用了。

Chapter6

第6章
HTTP首部

HTTP协议的请求和响应报文中必定包含HTTP首部，只是我们平时在使用Web的过程中感受不到它。本章我们一起来学习HTTP首部的结构，以及首部中各字段的用法。

