Em conexões persistentes, o servidor deixa a conexão TCP aberta após enviar resposta. Requisições e respostas subsequentes entre os mesmos cliente e servidor podem ser enviadas por meio da mesma conexão. Em particular, uma página Web inteira (no exemplo anterior, o arquivo-base HTML e as dez imagens) pode ser enviada mediante uma única conexão TCP persistente. Além do mais, várias páginas residindo no mesmo servidor podem ser enviadas ao mesmo cliente por uma única conexão TCP persistentes. Essas requisições por objetos podem ser feitas em sequência sem ter de esperar por respostas a requisições pendentes (paralelismo). Em geral, o servidor HTTP fecha uma conexão quando ela não é usada durante certo tempo (um intervalo de pausa configurável). Quando o servidor recebe requisições consecutivas, os objetos são enviados de forma ininterrupta. O modo *default* do HTTP usa conexões persistentes com paralelismo. Faremos uma comparação quantitativa entre os desempenhos de conexões persistentes e não persistentes nos exercícios de fixação dos Capítulos 2 e 3. Aconselhamos o leitor interessado a consultar Heidemann [1997] e Nielsen [1997].

2.2.3 Formato da mensagem HTTP

As especificações do HTTP [RFC 1945; RFC 2616] incluem as definições dos formatos das mensagens HTTP. Há dois tipos delas: de requisição e de resposta, ambas discutidas a seguir.

Mensagem de requisição HTTP

Apresentamos a seguir uma mensagem de requisição HTTP típica:

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

Podemos aprender bastante examinando essa simples mensagem de requisição. Primeiro, vemos que ela está escrita em texto ASCII comum, de modo que pode ser lida por qualquer um que conheça computadores. Segundo, vemos que ela é constituída de cinco linhas, cada qual seguida de um '*carriage return*' e '*line feed*' (fim de linha). A última linha é seguida de um comando adicional de '*carriage return*' e '*line feed*'. Embora essa tenha cinco linhas, uma mensagem de requisição pode ter muito mais e também menos do que isso, podendo conter até mesmo uma única linha. A primeira linha é denominada **linha de requisição**; as subsequentes são denominadas **linhas de cabeçalho**. A linha de requisição tem três campos: o do método, o do URL e o da versão do HTTP. O campo do método pode assumir vários valores diferentes, entre eles `GET`, `POST`, `HEAD`, `PUT` e `DELETE`. A grande maioria das mensagens de requisição HTTP emprega o método GET, o qual é usado quando o navegador requisita um objeto e este é identificado no campo do URL. Nesse exemplo, o navegador está requisitando o objeto `/somedir/page.html`. A versão é autoexplicativa. Nesse exemplo, o navegador executa a versão HTTP/1.1.

Vamos agora examinar as linhas de cabeçalho do exemplo. A linha de cabeçalho Host:www.some-school.edu especifica o hospedeiro no qual o objeto reside. Talvez você ache que ela é desnecessária, pois já existe uma conexão TCP para o hospedeiro. Mas, como veremos na Seção 2.2.5, a informação fornecida pela linha de cabeçalho do hospedeiro é exigida por *caches proxy* da Web. Ao incluir a linha de cabeçalho `Connection: close`, o navegador está dizendo ao servidor que não quer usar conexões persistentes; quer que o servidor feche a conexão após o envio do objeto requisitado. A linha de cabeçalho `User-agent`: especifica o agente de usuário, isto é, o tipo de navegador que está fazendo a requisição ao servidor. Neste caso, o agente de usuário é o Mozilla/5.0, um navegador Firefox. Essa linha de cabeçalho é útil porque, na verdade, o servidor pode enviar versões diferentes do mesmo objeto a tipos diferentes de agentes de usuário. (Cada versão é endereçada pelo mesmo URL.) Por fim, o cabeçalho `Accept-language`: mostra que o usuário prefere receber uma versão em francês do objeto se este existir no servidor; se não existir, o servidor deve enviar a versão *default*. O cabeçalho `Accept-language`: é apenas um dos muitos de negociação de conteúdo disponíveis no HTTP.

Após examinar um exemplo, vamos agora analisar o formato geral de uma mensagem de requisição, ilustrado na Figura 2.8. Vemos que tal formato é muito parecido com nosso exemplo anterior. Contudo, você provavelmente notou que, após as linhas de cabeçalho (e após a linha adicional com "*carriage return*" e "*line feed*"), há um "corpo de entidade". O corpo de entidade fica vazio com o método `GET`, mas é utilizado com o método `POST`. Um cliente HTTP em geral usa o método `POST` quando o usuário preenche um formulário — por exemplo, quando fornece palavras de busca a um site buscador. Com uma mensagem `POST`, o usuário continua solicitando uma página Web ao servidor, mas seu conteúdo depende do que ele escreveu nos campos do formulário. Se o valor do campo de método for `POST`, então o corpo de entidade conterá o que o usuário digitou nos campos do formulário.

Seríamos omissos se não mencionássemos que uma requisição gerada com um formulário não utiliza necessariamente o método `POST`. Ao contrário, formulários HTML costumam empregar o método `GET` e incluem os dados digitados (nos campos do formulário) no URL requisitado. Por exemplo, se um formulário usar o método `GET`, tiver dois campos e as entradas desses dois campos forem `monkeys` e `bananas`, então a estrutura do URL será `www.somesite.com/animalsearch?monkeys&bananas`. Ao navegar normalmente pela Web, você talvez já tenha notado URLs extensos como esse.

O método `HEAD` é semelhante ao `GET`. Quando um servidor recebe uma requisição com o método `HEAD`, responde com uma mensagem HTTP, mas deixa de fora o objeto requisitado. Esse método é usado com frequência pelos programadores de aplicação para depuração. O método `PUT` é muito usado junto com ferramentas de edição da Web. Permite que um usuário carregue um objeto para um caminho (diretório) específico em um servidor Web específico. O método `PUT` também é usado por aplicações que precisam carregar objetos para servidores Web. O método `DELETE` permite que um usuário, ou uma aplicação, elimine um objeto em um servidor Web.

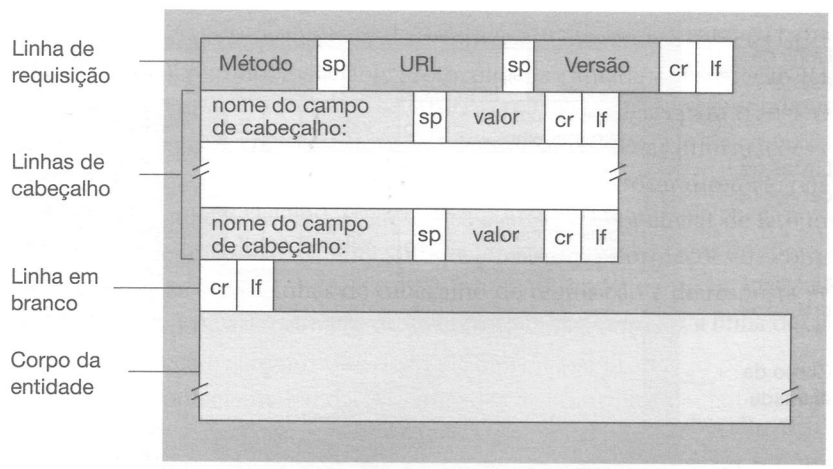
Mensagem de resposta HTTP

Apresentamos a seguir uma mensagem de resposta HTTP típica. Essa mensagem poderia ser a resposta ao exemplo de mensagem de requisição que acabamos de discutir.

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 09 Aug 2011 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(dados dados dados dados dados ...)
```

FIGURA 2.8 FORMATO GERAL DE UMA MENSAGEM DE REQUISIÇÃO HTTP



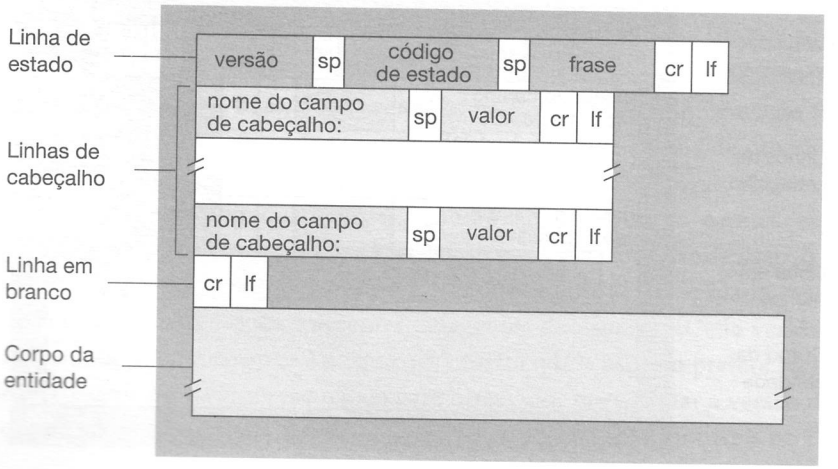
Vamos examinar com cuidado essa mensagem de resposta. Ela tem três seções: uma linha inicial ou **linha de estado**, seis **linhas de cabeçalho** e, em seguida, o **corpo da entidade**, que é a parte principal da mensagem — contém o objeto solicitado (representado por `dados dados dados dados dados` ...). A linha de estado tem três campos: o de versão do protocolo, um código de estado e uma mensagem de estado correspondente. Neste exemplo, ela mostra que o servidor está usando o HTTP/1.1 e que está tudo OK (isto é, o servidor encontrou e está enviando o objeto solicitado).

Agora, vamos ver as linhas de cabeçalho. O servidor usa `Connection: close` para informar ao cliente que fechará a conexão TCP após enviar a mensagem. A linha de cabeçalho `Date`: indica a hora e a data em que a resposta HTTP foi criada e enviada pelo servidor. Note que esse não é o horário em que o objeto foi criado nem o de sua modificação mais recente; é a hora em que o servidor extraiu o objeto de seu sistema de arquivos, inseriu-o na mensagem de resposta e a enviou. A linha de cabeçalho `Server:` mostra que a mensagem foi gerada por um servidor Web Apache, semelhante à linha de cabeçalho `User-agent`: na mensagem de requisição HTTP. A linha de cabeçalho `Last-Modified`: indica a hora e a data em que o objeto foi criado ou sofreu a última modificação. Esse cabeçalho, que logo estudaremos em mais detalhes, é fundamental para fazer *cache* do objeto, tanto no cliente local quanto em servidores de *cache* da rede (também conhecidos como servidores *proxy*). A linha de cabeçalho `Content-Length`: indica o número de bytes do objeto que está sendo enviado e a linha `Content-Type`: mostra que o objeto presente no corpo da mensagem é um texto HTML. (O tipo do objeto é oficialmente indicado pelo cabeçalho `Content-Type`: e não pela extensão do arquivo.)

Após examinar um exemplo, vamos agora analisar o formato geral de uma mensagem de resposta, ilustrado na Figura 2.9. Esse formato geral de mensagem de resposta condiz com o exemplo anterior de uma mensagem de resposta. Mas falemos um pouco mais sobre códigos de estado e suas frases, que indicam o resultado da requisição. Eis alguns códigos de estado e frases associadas comuns:

- `200 OK`: requisição bem-sucedida e a informação é entregue com a resposta.
- `301 Moved Permanently`: objeto requisitado foi removido em definitivo; novo URL é especificado no cabeçalho Location: da mensagem de resposta. O software do cliente recuperará automaticamente o novo URL.
- `400 Bad Request`: código genérico de erro que indica que a requisição não pôde ser entendida pelo servidor.
- `404 Not Found`: o documento requisitado não existe no servidor.
- `505 HTTP Version Not Supported`: a versão do protocolo HTTP requisitada não é suportada pelo servidor.

FIGURA 2.9 FORMATO GERAL DE UMA MENSAGEM DE RESPOSTA HTTP



R19. É possível que o servidor Web e o servidor de correio de uma organização tenham exatamente o mesmo apelido para um nome de hospedeiro (por exemplo, foo.com)? Qual seria o tipo de RR que contém o nome de hospedeiro do servidor de correio?

R20. Examine seus e-mails recebidos e veja o cabeçalho de uma mensagem enviada de um usuário com um endereço de correio eletrônico .edu. É possível determinar, pelo cabeçalho, o endereço IP do hospedeiro do qual a mensagem foi enviada? Faça o mesmo para uma mensagem enviada de uma conta do gmail.

SEÇÃO 2.6

R21. No BitTorrent, suponha que Alice forneça blocos para Bob durante um intervalo de 30 s. Bob retornará, necessariamente, o favor e fornecerá blocos para Alice no mesmo intervalo? Por quê?

R22. Considere um novo par, Alice, que entra no BitTorrent sem possuir nenhum bloco. Sem qualquer bloco, ela não pode se tornar uma das quatro melhores exportadoras de dados para qualquer dos outros pares, visto que ela não possui nada para enviar. Então, como Alice obterá seu primeiro bloco?

R23. O que é uma rede de sobreposição? Ela inclui roteadores? O que são as arestas da rede de sobreposição?

R24. Considere um DHT com uma topologia da rede de sobreposição (ou seja, cada par rastreia todos os pares no sistema). Quais são as vantagens e desvantagens de um DHT circular (sem atalhos)?

R25. Relacione pelo menos quatro diferentes aplicações que são apropriadas naturalmente para arquiteturas P2P. (*Dica*: Distribuição de arquivo e mensagem instantânea são duas.)

SEÇÃO 2.7

R26. O servidor UDP descrito na Seção 2.7 precisava de um *socket* apenas, ao passo que o servidor TCP precisava de dois. Por quê? Se um servidor TCP tivesse de suportar n conexões simultâneas, cada uma de um hospedeiro cliente diferente, de quantos *sockets* precisaria?

R27. Para a aplicação cliente-servidor por TCP descrita na Seção 2.7, por que o programa servidor deve ser executado antes do programa cliente? Para a aplicação cliente-servidor por UDP, por que o programa cliente pode ser executado antes do programa servidor?

PROBLEMAS

P1. Falso ou verdadeiro?

a. Um usuário requisita uma página Web que consiste em algum texto e três imagens. Para essa página, o cliente enviará uma mensagem de requisição e receberá quatro mensagens de resposta.

b. Duas páginas Web distintas (por exemplo, `www.mit.edu/research.html` e `www.mit.edu/students.html`) podem ser enviadas pela mesma conexão persistente.

c. Com conexões não persistentes entre navegador e servidor de origem, é possível que um único segmento TCP transporte duas mensagens distintas de requisição HTTP.

d. O cabeçalho Date: na mensagem de resposta HTTP indica a última vez que o objeto da resposta foi modificado.

e. As mensagens de resposta HTTP nunca possuem um corpo de mensagem vazio.

P2. Leia o RFC 959 para FTP. Relacione todos os comandos de cliente que são suportados pelo RFC.

P3. Considere um cliente HTTP que queira obter um documento Web em um dado URL. Inicialmente, o endereço IP do servidor HTTP é desconhecido. Nesse cenário, quais protocolos de transporte e de camada de aplicação são necessários, além do HTTP?

P4. Considere a seguinte cadeia de caracteres ASCII capturada pelo Wireshark quando o navegador enviou uma mensagem HTTP GET (ou seja, o conteúdo real de uma mensagem HTTP GET). Os caracteres <*cr*><*lf*> são

retorno de carro e avanço de linha (ou seja, a cadeia de caracteres em itálico <cr> no texto abaixo representa o caractere único retorno de carro que estava contido, naquele momento, no cabeçalho HTTP). Responda às seguintes questões, indicando onde está a resposta na mensagem HTTP GET a seguir.

```
GET /cs453/index.html HTTP/1.1<cr><lf>Host: gai
a.cs.umass.edu<cr><lf>User-Agent: Mozilla/5.0 (
Windows;U; Windows NT 5.1; en-US; rv:1.7.2) Gec
ko/20040804 Netscape/7.2 (ax) <cr><lf>Accept:ex
t/xml, application/xml, application/xhtml+xml, text
/html;q=0.9, text/plain;q=0.8,image/png,*/*;q=0.5
<cr><lf>Accept-Language: en-us,en;q=0.5<cr><lf>Accept-
Encoding: zip,deflate<cr><lf>Accept-Charset: ISO
-8859-1,utf-8;q=0.7,*;q=0.7<cr><lf>Keep-Alive: 300<cr>
<lf>Connection:keep-alive<cr><lf><cr><lf>
```

a. Qual é a URL do documento requisitado pelo navegador?

b. Qual versão do HTTP o navegador está rodando?

c. O navegador requisita uma conexão não persistente ou persistente?

d. Qual é o endereço IP do hospedeiro no qual o navegador está rodando?

e. Que tipo de navegador inicia essa mensagem? Por que é necessário o tipo de navegador em uma mensagem de requisição HTTP?

P5. O texto a seguir mostra a resposta enviada do servidor em reação à mensagem HTTP GET na questão anterior. Responda às seguintes questões, indicando onde está a resposta na mensagem.

```
HTTP/1.1 200 OK<cr><lf>Date: Tue, 07 Mar 2008
    12:39:45GMT<cr><lf>Server: Apache/2.0.52 (Fedora)
    <cr><lf>Last-Modified: Sat, 10 Dec2005 18:27:46
    GMT<cr><lf>ETag: "526c3-f22-a88a4c80"<cr><lf>Accept-
    Ranges: bytes<cr><lf>Content-Length: 3874<cr><lf>
    Keep-Alive: timeout=max=100<cr><lf>Connection:
    Keep-Alive<cr><lf>Content-Type: text/html; charset=
    ISO-8859-1<cr><lf><cr><lf><!doctype html public "-
    //w3c//dtd html 4.0 transitional//en"><lf><html><lf>
    <head><lf> <meta http-equiv="Content-Type"
    content="text/html; charset=iso-8859-1"><lf> <meta
    name="GENERATOR" content="Mozilla/4.79 [en] (Windows NT
    5.0; U) Netscape]"><lf> <title>CMPSCI 453 / 591 /
    NTU-ST550A Spring 2005 homepage</title><lf></head><lf>
    <muito mais texto do documento em seguida (não mostrado)>
```

a. O servidor foi capaz de encontrar o documento com sucesso ou não? A que horas foi apresentada a resposta do documento?

b. Quando o documento foi modificado pela última vez?

c. Quantos bytes existem no documento que está retornando?

d. Quais são os 5 primeiros bytes do documento que está retornando? O servidor aceitou uma conexão persistente?

P6. Obtenha a especificação HTTP/1.1 (RFC 2616). Responda às seguintes perguntas:

a. Explique o mecanismo de sinalização que cliente e servidor utilizam para indicar que uma conexão persistente está sendo fechada. O cliente, o servidor, ou ambos, podem sinalizar o encerramento de uma conexão?

b. Que serviços de criptografia são providos pelo HTTP?

c. O cliente é capaz de abrir três ou mais conexões simultâneas com um determinado servidor?

d. Um servidor ou um cliente pode abrir uma conexão de transporte entre eles se um dos dois descobrir que a conexão ficou lenta por um tempo. É possível que um lado comece a encerrar a conexão enquanto o outro está transmitindo dados por meio dessa conexão? Explique.