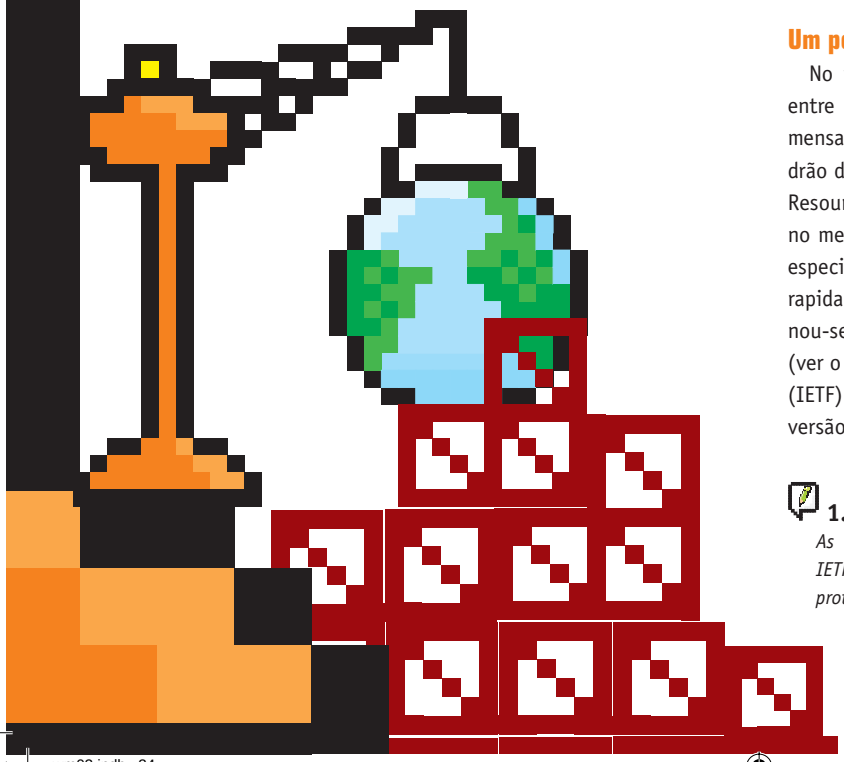
Infra-estrutura web

por Marcelo Morgade



O início do aprendizado de uma linguagem ou framework de programação web é sempre uma experiência gratificante. Todos se sentem realizados ao sair da "velha" e "ultrapassada" arquitetura cliente-servidor para o novo e promissor mundo dos sistemas com interface web. Porém, na ânsia de ganhar experiência rapidamente neste novo campo, alguns programadores queimam etapas importantes do processo de aprendizagem e encaram os recursos de programação para web simplesmente como uma nova biblioteca qualquer de uma linguagem que este já está familiarizado. Ele usa objetos para criar interfaces, mas não possui muito conhecimento sobre a infra-estrutura que estes objetos representam e que torna a internet possível.

Certamente esta visão limitada em relação às adaptações necessárias para a utilização de uma linguagem de programação para web é um dos problemas que mais causam dificuldades para quem está começando. O desenvolvedor descobre que deve manipular objetos de nomes estranhos como request, response e session, e começa a utilizá-los sem conhecer o significado e o funcionamento desses novos objetos. A partir daí, ele se torna uma figura conhecida como "colador de código".

"Colar código" parece funcionar no início da utilização deste novo paradigma de desenvolvimento, mas logo o programador começa a enfrentar requisitos um pouco mais complexos. Neste momento o desenvolvedor irá se perguntar desesperado qual parte do estudo pulou. Ele percebe, assim, que conhecer um pouco do funcionamento da web em um nível um pouco mais baixo é essencial para que possa evoluir de um simples "colador de código" para um programador eficiente.

Ao longo deste artigo, iremos descrever os principais conceitos relacionados ao desenvolvimento de uma aplicação web, e como estes conceitos estão presentes em uma linguagem de programação.

Um pouco de história

No protocolo HTTP temos definidas as fases de comunicação entre clientes (os browsers) e servidores web, os formatos das mensagens trocadas entre estes dois programas, além de um padrão de endereçamento de documentos chamado hoje de Uniform Resource Locator (URL). O HTTP passou a ser usado amplamente no meio acadêmico para troca de informação científica e, com a especificação da Hyper-Text Markup Language (HTML), sagrou-se rapidamente como o serviço mais popular da internet. O HTTP tornou-se um padrão de fato em 1996 com a publicação da RFC1945 (ver o que são RFCs na **1**) pelo Internet Engineering Task Force (IETF). Este documento descreve todos os detalhes da primeira versão do protocolo de aplicação utilizado pela web.

1. Request for Comments.

As RFCs são documentos publicados por grupos de trabalho do IETF com o objetivo de padronizar a implementação de serviços, protocolos ou procedimentos (existe, por exemplo, uma RFC que cria

um padrão de escrita de RFCs!). Elas são extremamente importantes, já que o estabelecimento de padrões de comunicação foi um dos passos básicos para o crescimento da internet. As RFCs são a fonte de informações primordial dos desenvolvedores de programas, componentes de comunicação ou equipamentos que fazem uso da internet.

Há algum tempo atrás, esta RFC era a única fonte de estudo de um programador que estivesse entrando no mundo web (*www.ietf.org/rfc/*). Mas, atualmente não é mais necessário saber toda esta especificação. Hoje temos linguagens e frameworks direcionados para o desenvolvimento de aplicações web que já implementam tudo aquilo que está padronizado pelo HTTP. Aqueles objetos de nomes estranhos como request, response e session nada mais são do que pacotes de software através dos quais podemos interagir com o HTTP. É por isso que para deixar de ser somente um "colador de código", o programador deve conhecer um pouco sobre o funcionamento do HTTP, pois é com ele que estará sempre interagindo e é ele quem determinará o que é possível fazer numa aplicação web.

O Hyper-Text Transfer Protocol

Logo no seu início, a RFC descreve o HTTP de modo geral como um protocolo baseado no paradigma requisição/resposta que não mantém informações sobre estado. Estas informações de estado poderiam envolver dados sobre o tempo total que alguém leva num site, quantos links alguém clicou ou qual a seqüência de páginas acessada por um determinado usuário. A aquisição destes dados foi simplesmente ignorada, com o objetivo de tornar o HTTP algo extremamente simples. Assim, a principal função do HTTP é simplesmente gerenciar pedidos simples de requisição de dados e resposta.

Pode parecer pouco, mas este conhecimento sobre o HTTP envolve as informações mais importantes para entendermos como funciona a web. Isso permite entender o que acontece quando um usuário digita um endereço em um browser e pressiona a tecla <enter> ou quando clica em um link qualquer. Eis o que acontece:

1. O browser processa a parte inicial da URL de forma a descobrir o endereço do servidor. Ele então se conecta com este servidor usando o protocolo TCP/IP e envia uma mensagem (requisição) baseada na parte final da URL junto com outros parâmetros.

2. O servidor web recebe e processa esta mensagem e seus parâmetros. A depender da requisição, envia uma resposta diferente contendo um status, outros parâmetros e um conteúdo qualquer (a resposta).

O HTTP se resume a estes dois passos acima. Ele sai de cena depois que o browser recebe a resposta, cabendo a este fazer o processamento dos parâmetros e conteúdo da resposta do servidor.

Toda esta simplicidade ajuda no desempenho do servidor, uma vez que ele não precisa armazenar em sua memória informações sobre cada uma das requisições anteriores feitas por um browser e não precisa manter a conexão aberta após responder a uma requisição. Existem protocolos mais complexos como o FTP (File Transfer Protocol), que exige o armazenamento na memória do servidor de uma série de informações sobre cada cliente (como login, senha, e diretório do usuário), além de necessitar de um fluxo contínuo de troca de mensagens e uma conexão ininterrupta entre cliente e servidor. Isso faz com que servidores FTP aceitem apenas uma quantidade limitada de usuários simultaneamente. A simplicidade do HTTP permite que muito mais usuários se comuniquem com o servidor ao mesmo tempo.

Observaremos a partir de agora como aqueles objetos estranhos presentes nas linguagens começam a fazer sentido quando olhamos as características básicas do HTTP.

Objetos request e requisições HTTP

Uma requisição ocorre cada vez que um link é clicado, um formulário é submetido, uma imagem é carregada ou qualquer outra informação presente no servidor é necessária. Cada arquivo que compõe uma página que você visualiza na web veio de uma requisição diferente do seu browser ao servidor.

Estas requisições HTTP são nada mais que mensagens em texto ASCII enviadas ao servidor que contêm um nome de arquivo a ser devolvido e um conjunto de parâmetros. A RFC1945 define o formato destas mensagens e o papel de alguns parâmetros. Através da definição destes formatos, todos os browsers e servidores podem processar corretamente requisições e respostas, não importando se rodam em Linux, Windows ou em um Palmtop. Observe um exemplo de uma mensagem de requisição na **1**.

Nesta requisição, o browser informa que o servidor deve processar o arquivo rfc.html (**1 - Linha 1**), além de informar sua versão (**1 - Linha 1**), sistema operacional (**1 - Linha 2**) e ainda qual o idioma preferido do usuário (**1 - Linha 3**). Um teste interessante é simularmos um browser através do telnet. Para isto, conecte-se via telnet no servidor do IETF na porta 80 (a porta padrão dos servidores web). Na maioria dos sistemas operacionais você pode fazer isso usando o seguinte comando:

```
telnet www.ietf.org 80
```

Uma vez conectado, digite exatamente o texto da requisição apresentada na **1** sem errar nada (se errar algo, o servidor não reconhece a mensagem). Depois da última linha, aperte a tecla <enter> duas vezes. Se você fez tudo certo, o servidor mandará uma resposta contendo alguns parâmetros e o conteúdo do arquivo rfc.html. Você acaba de ver na prática como funciona a comunicação browser-servidor web.

Analisando a primeira linha do exemplo apresentado (**1**), observa-se a palavra GET. A primeira palavra de uma requisição é chamada de método. O método define o que o servidor deve

fazer com a requisição. Na RFC é possível encontrar a definição de diversos métodos (GET, POST, PUT, DELETE, HEAD). Porém, os browsers enviam na maioria das vezes apenas os métodos GET, POST e HEAD. Os dois primeiros requisitam o conteúdo do arquivo pedido, porém passam parâmetros de maneiras diferentes. O HEAD algumas vezes é usado para checar se certo arquivo foi modificado ou se um cache local pode ser usado. Observe que os objetos request das linguagens web normalmente disponibilizam a informação sobre qual método de requisição foi usado.

Logo após o método, o browser deve passar o caminho para o arquivo requisitado no servidor. É neste trecho da requisição que o servidor deve decidir se irá enviar o conteúdo de um arquivo HTML, uma imagem, ou ainda se deverá executar algum script ou programa que gerará o conteúdo a ser devolvido na resposta. Para o browser, pouco importa como o conteúdo da resposta foi construído. A única preocupação do browser nesse momento é mostrar na tela de forma correta o conteúdo desta resposta.

No final da primeira linha, é informada qual a versão do HTTP é usada. A versão 1.1 do HTTP é especificada na RFC2068 e define algumas novas extensões. Alguns cabeçalhos muito úteis são especificados e outras funcionalidades adicionadas, mas nada que modifique a natureza simples do protocolo. Novamente, os objetos request das linguagens web também disponibilizam qual versão do HTTP está sendo utilizada.

Após a primeira linha da requisição (**1 – Linhas 2 e 3**), é enviado um conjunto de dados chamado de cabeçalhos (headers) HTTP. Nos cabeçalhos podemos recuperar informações muito úteis sobre o cliente que fez a requisição. Observe que podem existir diversas linhas de cabeçalho, sempre seguindo o formato:

```
<Nome do Cabeçalho>: <valor>
```

Existem muitos cabeçalhos de requisição definidos no HTTP, cada um com a função de enviar informações ou preferências do cliente ao servidor. São estas informações que permitem certas aplicações web fazerem "mágicas" como mostrar o conteúdo em inglês ou português a depender do idioma preferido pelo usuário. Observe que esta "mágica" pode ser feita apenas testando o valor do cabeçalho Accept-Language que os browsers passam nas requisições.

Agora que você sabe como a "mágica" é feita, certamente já sabe onde deve buscar essa informação na sua linguagem web. Em seu objeto request deve existir uma maneira de descobrir o valor de um determinado cabeçalho HTTP e assim utilizar estas informações para tomar decisões que a princípio não pareciam possíveis. Você pode também, por exemplo, obter o valor do cabeçalho Referer via o objeto request para descobrir de qual página o usuário de sua aplicação veio. A RFC2068 é uma boa fonte de pesquisa para descobrir outros cabeçalhos de requisição com informações que podem ser úteis à sua aplicação.

1. Exemplo de mensagem e requisição HTTP – Método GET

```
1. GET /rfc.html HTTP/1.0
2. User-Agent: Mozilla/4.0 Linux
3. Accept-Language: pt-br
```

2. Exemplo de mensagem de requisição HTTP – Método POST.

```
POST /servlet/respostaDoForm HTTP/1.0
User-Agent: Mozilla/4.0 Linux
Accept-Language: pt-br

param1=valor1&param2=valor2
```

O HTTP também define a forma como os parâmetros de requisição são passados, através de uma string de consulta (query string) após o arquivo: "/arquivo?param1=valor1&param2=valor2..." (usada com o método GET). Outra forma de passar parâmetros é através do método POST, que codifica estes dados uma linha após a lista de cabeçalhos da requisição. No exemplo da **2**, observamos uma requisição feita através do método POST, provavelmente feita a partir de um formulário HTML que contém campos de dados chamados param1 e param2 cujos valores eram valor1 e valor2 no momento da requisição.

A recuperação dos parâmetros de requisição é, normalmente, o recurso mais usado dos objetos request das linguagens web, uma vez que é o método mais comum para recuperar dados de entrada enviados pelos usuários através de links ou formulários HTML.

Objetos response e respostas HTTP

Ao receber uma requisição HTTP, o servidor web deve processar todos os seus dados ou deve entregá-los a uma aplicação web empacotados na forma de um objeto request. O servidor ou a aplicação deve então devolver ao browser uma resposta apropriada para a requisição. Se você conseguiu simular a requisição de browser via telnet, observou como é o formato de uma resposta HTTP (ver **3**).

Observe na linha 1 (**3**), que o servidor informa novamente a versão do HTTP e logo após passa um número chamado de status de resposta. É através deste número que o servidor informa se a requisição obteve sucesso ou se ocorreu algum problema na requisição. O status 200 informa que a requisição foi atendida e o conteúdo da resposta virá em seguida. Alguns outros números de status são bem conhecidos, como o 404 (informa que a página ou aplicação não foi encontrada) e o 500 (informa que houve um erro no servidor ou na aplicação web). Um número de status que

3. Exemplo de mensagem de resposta HTTP.

```
1.  HTTP/1.1 200 OK
2.  Date: Mon, 30 Aug 2004 20:58:29 GMT
3.  Server: Apache/2.0.46 (Red Hat)
4.  Last-Modified: Fri, 02 May 2003 19:13:31 GMT
5.  ETag: "414159-cf2-35634cc0"
6.  Accept-Ranges: bytes
7.  Content-Length: 3314
8.  Connection: close
9.  Content-Type: text/html; charset=UTF-8

10. <HTML>
    ... conteúdo da página
11. </HTML>
```

muitos utilizam sem saber é o 302, que informa ao browser para redirecionar a requisição para outro arquivo. Normalmente, o número de status é configurado automaticamente pelo servidor web. Mas, também podemos manipulá-lo em nossas aplicações. Os objetos response geralmente possibilitam configurar manualmente o status da sua resposta.

Após a primeira linha da resposta (**3 – Linhas 2 a 9**), é enviado um conjunto de cabeçalhos HTTP no mesmo formato visto durante a requisição. Desta vez, estes cabeçalhos vão fornecer ao browser informações adicionais sobre o servidor e a resposta recebida. Usando o seu objeto response você pode setar o valor de alguns cabeçalhos da sua resposta para que o browser efetue funcionalidades específicas. Você pode, por exemplo, pedir que o browser refaça a requisição de tempos em tempos configurando o cabeçalho:

```
Refresh: 3
```

Você também pode dizer ao browser que a resposta que sua aplicação envia não é HTML, mas sim um arquivo de dados separado por vírgula que deve ser salvo no disco e não mostrado para o usuário. Para isso, basta definir os cabeçalhos:

```
Content-Type: text/comma-separated-values
Content-Disposition: inline; filename=meuarquivo.csv
```

Mais uma vez, uma consulta às RFCs é a fonte mais completa de informações sobre o que é possível definir nos cabeçalhos de resposta HTTP.

Finalmente, depois dos cabeçalhos vem o conteúdo da resposta. Neste ponto normalmente se usa algum recurso dos objetos response que permita mandar dados seqüencialmente para o browser. Em linguagens ou frameworks Web baseados em scriptlets (ASP, PHP, JSP), o HTML é misturado com o código fonte do programa e assim se perde a noção de que o objeto response é o verdadeiro responsável por administrar este conteúdo da resposta.

Um detalhe importante no formato da mensagem de resposta HTTP é o fato do conteúdo da resposta ser mandado sempre depois dos cabeçalhos e do status. Isso é a causa de um erro comum cometido pelos programadores, que tentam configurar o status ou valores de cabeçalhos após já terem mandado parte do conteúdo da resposta. É por este motivo que só se pode redirecionar o browser para outra página se nenhum conteúdo de resposta tiver sido enviado.

Cookies

Agora que você conhece os recursos básicos do HTTP, fica mais claro entender outros recursos das linguagens web. O envio de cookies é um destes recursos que se tornou um mito, pois sempre foi visto como uma brecha de segurança para os browsers. Mas os cookies são apenas mais um conjunto de cabeçalhos que pode ser enviado nas requisições e respostas HTTP. Este novo conjunto de cabeçalhos foi especificado na RFC2109, que define mecanismos capazes de implementar gerenciamento de estado através do HTTP. Esse gerenciamento de estados implementado usando cookies permite saber, por exemplo, quantas vezes uma determinada pessoa esteve em seu site, gravando esta informação no computador cliente.

Quando você pede para seu objeto response enviar um cookie com o número de visitas para ser gravado no browser, o objeto apenas adiciona um cabeçalho na resposta HTTP com o seguinte padrão:

```
Set-Cookie: numerodevisitas=129;expires=Friday, 31-Dec-2006
23:59:59 GMT; path=/minhaAplicacaoWeb
```

Se o browser que receber este cabeçalho tiver o recebimento de cookies habilitado, esta informação sobre o número de visitas será gravada em disco e associada ao seu site e aplicação. Da próxima vez que o browser fizer uma requisição qualquer para sua aplicação (mesmo que seja uma semana depois da última visita), ele adicionará o seguinte cabeçalho:

```
Cookie: numerodevisitas="129"; $Path="/minhaAplicacaoWeb"
```

Quando sua aplicação recebe uma requisição com este cabeçalho, você pode pedir ao objeto request o valor do cookie "número de visitas" e terá como resposta o valor 129. É neste mecanismo simples de troca de cabeçalhos que os cookies são baseados. Observe que existem outros detalhes que você pode pesquisar na RFC como a data de expiração dos cookies, comentários, opções de segurança, etc. Tudo isso é implementado nos cabeçalhos Cookie e Set-Cookie.

Os objetos session

Um programador iniciando no mundo web descobre a importância dos objetos session logo no início do seu estudo. O objetivo principal de existência desses objetos é semelhante ao objetivo dos cookies, ou seja, implementar o gerenciamento de estados que falta no HTTP. Você já deve também ter ouvido falar que os objetos session são implementados através de cookies (ou como vimos, apenas mais cabeçalhos HTTP).

A principal diferença entre estes dois recursos está no local onde as informações de estado são guardadas. Vimos que usando cookies, dados podem ser enviados diretamente para o browser a fim de serem armazenados na máquina cliente. Utilizando os objetos session, estes dados relacionados a um browser são guardados por um tempo na memória do servidor, e não nos discos dos seus usuários. Os cookies são usados neste caso apenas para identificar os browsers e associá-los ao objeto session correspondente na memória do servidor. Estes cookies são chamados cookies de sessão e não são gravados no disco do cliente. O browser apenas os guarda na memória enquanto está executando e simplesmente os descarta depois que são fechados.

Para um servidor implementar um objeto session, ele precisa enviar um cookie de sessão na primeira requisição que o browser faz à sua aplicação. O servidor cria uma área na sua memória para um objeto session nesta primeira requisição e define um valor chamado SESSIONID que é o identificador deste objeto na memória. Este SESSIONID é o único dado passado ao browser em forma de cookie:

```
Set-Cookie: SESSIONID=To1O1OmC9615692976201S3At;Path=/
```

Nas requisições seguintes, o browser incluirá este cookie com o SESSIONID recebido no cabeçalho HTTP. Com esta informação, o servidor pode recuperar o objeto session específico do browser criado nas requisições anteriores e recuperar todas as informações armazenadas pela aplicação que estão relacionadas ao usuário "dono" da sessão. Isso permite implementar funcionalidades interessantes, como os famosos carrinhos de compra encontrados nos sites de comércio eletrônico. Estes "carrinhos" são módulos responsáveis por guardar a lista de produtos que você escolhe durante a visita a um site de compras. Quando você clica para comprar um produto, ele é adicionado no seu carrinho de compras e você pode continuar buscando por outros produtos no site. Na hora do pagamento, o site é capaz de mostrar todos os produtos que você colocou no carrinho durante suas compras. Tudo isso é possível com o uso do objeto session, que pode armazenar de forma segura as informações sobre os produtos escolhidos por cada usuário durante uma compra.

Observe que os dados obtidos de um objeto session são muito mais confiáveis que os dados vindos de um cookie. Naquele exemplo do número de visitas, um usuário pode abrir o arquivo do cookie gravado em sua máquina em modificar o dado manualmente. Usando o session, o máximo que o usuário pode fazer é modificar o valor do seu SESSIONID. Isso só o levaria a perder todos os dados da sua sessão no servidor (a não ser que ele tenha muita sorte e acerte o valor do SESSIONID de um objeto criado para outro browser).

Objetos session podem ocupar muita memória no servidor e por isso precisam ser bem gerenciados. Como o HTTP é baseado no esquema de requisição-resposta, o servidor não tem como saber se um cliente associado a um determinado objeto session saiu do site ou ainda está simplesmente lendo o conteúdo da resposta. Devido a essa característica, normalmente se trabalha com um tempo de limite de sessão. Cada objeto session na memória do servidor tem um prazo de duração, que é renovado sempre que o browser dono do objeto faz uma requisição. Se o cliente fica muito tempo sem fazer uma requisição e o tempo do seu objeto se esgota, o servidor assume que o cliente não interagirá novamente e remove o objeto da memória. Neste caso, normalmente diz-se que a sessão expirou. Este tempo normalmente é configurável através do objeto session, e deve ser bem estudado de acordo com a aplicação. Um tempo de sessão muito curto pode irritar os usuários mais "lentos", que podem perder suas sessões por levarem muito tempo lendo uma página. Um tempo muito longo pode fazer com que o servidor fique com a memória sobrecarregada de objetos session desnecessários, diminuindo o desempenho da sua aplicação.

Conclusão

Essa visão rápida do HTTP serve para mostrar de forma geral como os recursos das linguagens e frameworks web são implementados. Com este conhecimento, você obterá uma visão mais ampla e apurada sobre como sua aplicação funciona e quais recursos que ela utiliza, permitindo solucionar os desafios técnicos do dia-a-dia com muito mais facilidade e te dando segurança para poder dizer ao seu cliente ou gerente de projeto o que é e o que não é possível fazer em matéria de interfaces para sistemas web. ■

Marcelo Burgos Morgade Cortizo (*morgade@fja.edu.br*) é Mestre em Redes de Computadores pela Universidade Salvador e Bacharel em Ciência da Computação pela Universidade Salvador. Atualmente é professor de Técnicas e Linguagens de Programação e Sistemas Distribuídos do curso de Sistemas de Informação das Faculdades Jorge Amado (Salvador – BA) e Analista de Sistemas da Empresa Baiana de Águas e Saneamento (EMBASA).