CAPÍTULO 14

Apêndice - Otimizações de front-end

"A esperança...: um sonho feito de despertares"
— Aristóteles

Estudos de diversas empresas ao redor do mundo já provaram que sites rápidos, além de mais prazerosos de usar, trazem mais conversões, vendas e usuários felizes.

A Amazon, por exemplo, descobriu que cada 100ms de melhora na velocidade de carregamento da página trazia um crescimento de 1% em seu faturamento.

O Yahoo! provou que cada 400ms de melhora em sua homepage provoca um aumento de 9% no tráfego dos usuários.

A Mozilla tornou suas páginas 2.2s mais rápidas e, com isso, causou um aumento de 15% nos downloads do Firefox. São 60 milhões de downloads a mais por ano.

O Google descobriu que aumentar o tempo de carregamento da página de busca de 0.4s para 0.9s ao subir o número de resultados de 10 para 30 diminuía o tráfego em 20%.

Até a Caelum já fez experimentos em seu Site. No nosso caso, uma página que tinha tempo de carregamento de 6s em comparação com uma de 2s causava uma perda de 18% na taxa de conversões.

Otimização é coisa de programadores

Um dos maiores erros a se cometer com relação à performance é pensar que é um problema exclusivo da programação da parte server-side do projeto. Certamente um código ruim no servidor pode causar imensos gargalos de performance. Uma busca mal feita no banco de dados, um algoritmo pesado de executar etc.

Na esmagadora maioria das situações, a realidade é que o processamento server-side é responsável por menos de 10% do tempo total de carregamento de uma página. A maior parte dos gargalos de performance está em práticas client-side!

Um estudo que fizemos com 100 Sites de participantes de uma conferência de tecnologia de alto nível técnico - a QCon SP de 2011 - trouxe dados interessantes. Nessa amostra, o tempo médio de carregamento da página era de 9s (com casos bem graves, demorando mais de 30s). Mesmo assim, 75% dos Sites executavam em menos de 400ms no servidor. Os outros 8s são gastos em outros pontos, muito ligados a práticas client-side que veremos.

14.1 - HTML e HTTP - Como funciona a World Wide Web?

Apesar de que conhecer profundamente o funcionamento do protocolo HTTP não seja estritamente necessário para a criação de páginas Web, entender como as coisas funcionam internamente nos ajuda a entender uma série de técnicas e conceitos, resultando em maior qualidade na criação de páginas, além de contribuir para a confiança do programador ao enfrentar um novo desafio.

A primeira coisa que devemos levar em consideração ao conhecer o ciclo de comunicação entre o navegador (cliente) e o servidor, é que o cliente deve conhecer a localização da página (recurso) que ele deseja obter e exibir ao usuário final. O cliente deve ser informado de qual o endereço do recurso necessário em determinado momento, normalmente o usuário final provê essa informação entrando um endereço na barra de endereços do navegador, ou clicando em um link.

Esse endereço é conhecido como URL (Universal Resource Locator), por exemplo:

```
http://209.85.227.121:80/index.html
```

O endereço exemplificado é composto por 4 partes básicas. A primeira delas é o protocolo de comunicação a ser utilizado para a obtenção do recurso. No exemplo acima o protocolo requerido é o HTTP. A comunicação entre um cliente (geralmente o navegador) e um servidor pode ser feita com o uso de diversos protocolos, por exemplo o FTP (File Transfer Protocol) para a transferência de arquivos, ou o protocolo *file*, de leitura direta de arquivo quando desejamos obter um recurso acessível diretamente pelo computador sem utilizar uma conexão com um servidor Web.

Hoje em dia, os navegadores não precisam que explicitemos o protocolo HTTP em sua barra de endereços, sendo ele o padrão para as requisições de páginas Web.

A segunda e terceira partes, entre `//` e `/`, são o endereço IP do servidor (onde está hospedado o recurso que queremos) e a porta de comunicação com o servidor. Os servidores Web utilizam a porta 80 por padrão, então no exemplo poderíamos ter omitido essa informação que a comunicação seria feita com sucesso.

O endereço IP é um código composto de 4 octetos representados em formato decimal separados por um ponto, é um número um tanto difícil de ser memorizado, (a próxima geração de endereços IP, criada para evitar o fim dos endereços disponíveis é formada por 8 grupos de 4 dígitos hexadecimais separados por ":", por exemplo: `2001:0db8:85a3:08d3:1319:8a2e:0370:7344`) por isso a Web utiliza servidores de nomeação de domínios (DNS), para que o usuário final possa informar um nome em vez de um número e uma porta, por exemplo `www.caelum.com.br`.

A quarta parte é o caminho do recurso que desejamos obter dentro do servidor. No nosso exemplo estamos solicitando o arquivo `index.html`, que é o nome padrão de arquivo para a página inicial de um Site, e, nesse caso, também poderia ser omitido. A adoção desses valores padrões permite que para obtermos a página inicial do site da Caelum, por exemplo, os usuários finais digitem somente **www.caelum.com.br** na barra de endereços de seu navegador.

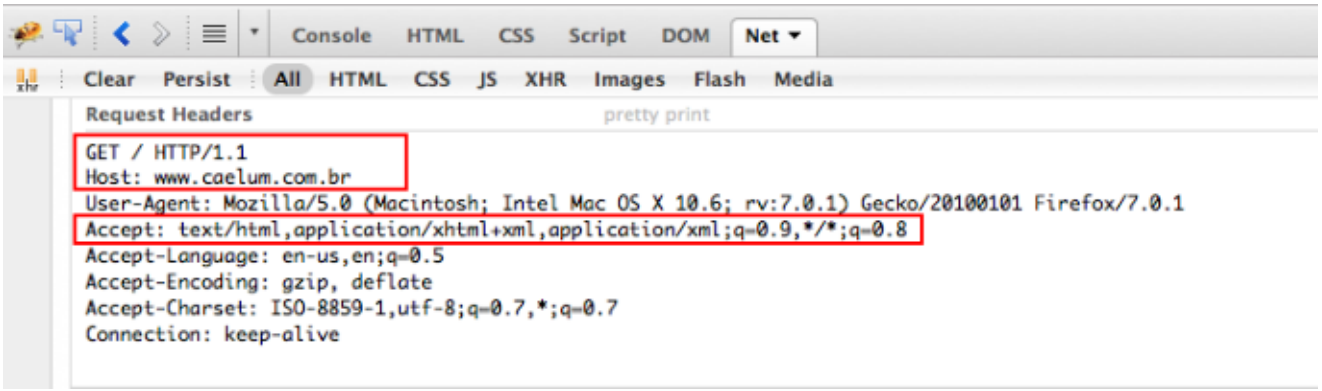
O ciclo HTTP

Conhecemos um pouco sobre URLs, e as utilizaremos quando formos adicionar links e recursos externos em nossos documentos, mas o que acontece quando clicamos em um link ou digitamos um endereço no navegador e clicamos em "ir"? Essas ações disparam uma chamada, dando início ao ciclo HTTP. Essa chamada é o que chamamos de **request** (requisição).

A correta comunicação entre os dois lados do ciclo depende de uma série de informações. Um HTTP request leva consigo todos os dados necessários para que o lado do servidor tome a decisão correta sobre o que fazer. Existem algumas ferramentas que permitem que observemos quais são essas informações.

O protocolo HTTP pode ser utilizado por uma série de aplicações, para uma série de finalidades. Nosso foco é no uso do HTTP para páginas da Web que podemos acessar de um navegador. Alguns navegadores incluem ferramentas de inspeção da página em exibição, e a maioria dessas ferramentas consegue nos mostrar o

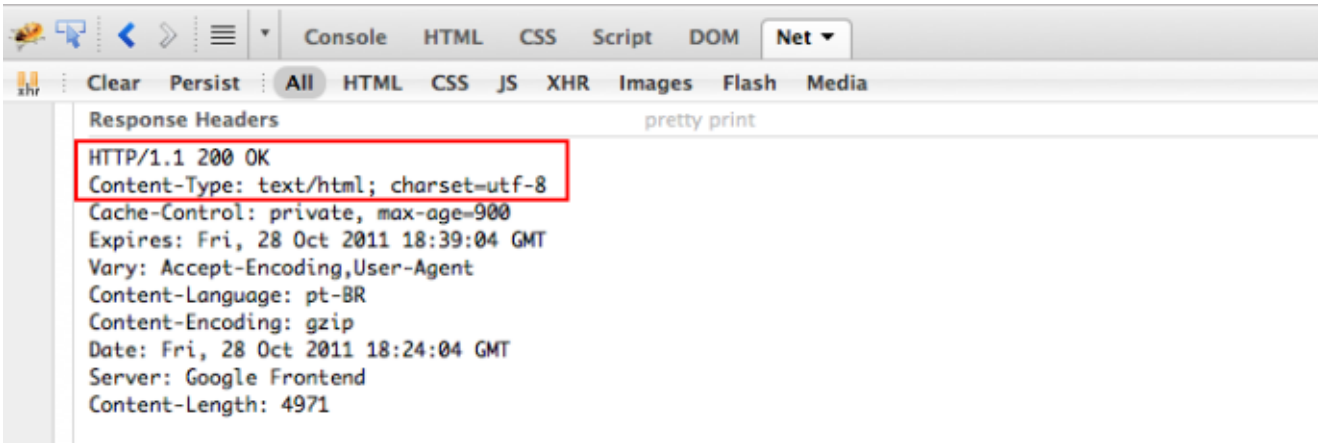
conteúdo da requisição HTTP. Uma dessas ferramentas é o complemento **Firebug**, disponível para o navegador Firefox.



Na imagem estão destacadas as informações mais relevantes da requisição. A primeira delas é a palavra **GET**. GET é um dos métodos suportados pelo protocolo HTTP para realizar a comunicação, e ele deve ser utilizado quando queremos **obter** um recurso que o servidor tem acesso. Caso o servidor encontre o recurso que queremos, ele retorna para o cliente uma **response** (resposta) contendo o recurso que desejamos.

Outras informações importantes são o endereço do recurso que desejamos obter (**Host**) e o tipo de recurso que o cliente espera obter (**Accept**). No exemplo, esperamos encontrar uma página HTML pelo endereço `www.caelum.com.br`.

Com essas informações, o servidor processa o pedido e prepara uma **response** (resposta). Essa resposta contém uma série de informações importantes para que o cliente possa tomar as decisões necessárias e, no caso de sucesso na comunicação, exibir o recurso para o usuário.



As informações mais importantes, no nosso caso, são o código da resposta e o tipo do recurso encontrado (**Content-Type**). No nosso exemplo, o código **200** indica que o recurso foi localizado com sucesso e incluído na resposta.

Todas essas informações que vimos até aqui fazem parte dos **cabeçalhos** da

requisição e da resposta. São informações irrelevantes para o usuário, porém essenciais para o correto tratamento das informações solicitadas. As informações que serão exibidas no navegador para o usuário estão contidas no **corpo** da resposta. No nosso exemplo, assim como em toda requisição solicitando uma página Web, o corpo da resposta contém as informações marcadas para exibição correta no navegador.

14.2 – Princípios de programação distribuída

Uma página Web é uma aplicação distribuída. Isso significa que há uma comunicação via rede entre dois pontos. No caso, o navegador e o servidor da página.

E, como toda aplicação distribuída, há alguns princípios básicos de performance. Quando há comunicação remota envolvida, em geral, queremos:

Diminuir o volume de dados trafegados entre as partes.

e

Diminuir o número de chamadas remotas.

14.3 – Ferramentas de diagnóstico – YSlow e PageSpeed

O primeiro passo é saber o que melhorar. Há diversas boas práticas pregadas na literatura de performance Web. Melhor ainda é a existência ferramentas automatizadas para diagnóstico que analisam sua página e dão dicas sobre o quê e

como melhorar. Há até uma nota de 0 a 100 para você saber o quão bem está nas práticas de otimização.

As mais famosas ferramentas são duas extensões, a **YSlow** feita pelo pessoal do Yahoo! e a **PageSpeed** feita pelo Google. Ambas são extensões para o Firefox e para Chrome - mas rodam melhor no Firefox. Para instalá-las, primeiramente, você vai precisar do Firebug, depois é só baixá-las nos respectivos sites:

http://developer.yahoo.com/yslow/ http://code.google.com/speed/page-speed/download.html

O PageSpeed tem até uma versão online que analisa suas páginas que já estejam publicadas em algum endereço na internet:

https://developers.google.com/pagespeed/

YSlow

Abra o Firebug e clique no YSlow. Ele te mostra uma nota para as otimizações da página e sugestões do que melhorar.

Dê uma olhada nas regras. Note que há algumas que envolvem programação no servidor, como configurar compressão GZIP ou acertar os headers HTTP. É uma boa conversar com os programadores do projeto para também fazerem esses acertos no servidor, além do que você já vai fazer no client-side.

Várias regras dizem respeito a otimizações que já podemos implementar como comprimir/minificar nossos CSS e JavaScript, algo que veremos no tópico seguinte.

PageSpeed

Abra o Firebug e vá na aba do PageSpeed. Ele lhe mostra uma nota para a página e diversas práticas de otimização. Aquelas que estão em vermelho são as que você deveria fazer mas não fez. Amarelos são algumas sugestões e verdes são as que você está bem - mas às vezes há mais sugestões até nessas.

Para saber mais: WebPageTest.org

Outra ferramenta interessante e online é a WebPageTest.org. Ela também nos dá notas e sugestões de melhoria. Um diferencial bem interessante é que ela executa a página em navegadores reais em diversos lugares do mundo e nos dá

métricas de tempo de carregamento, e até um vídeo mostrando a performance de acordo com o tempo.

Se você já tem a página publicada em algum endereço, é bem interessante testar essa ferramenta também.

14.4 - Compressão e minificação de CSS e JavaScript

Durante todo o curso, aprendemos diversas boas práticas de codificação CSS e JavaScript. E, como toda programação, um ponto importante é manter a legibilidade do código.

Dar bons nomes a variáveis, escrever bons comentários, escrever código identado, com bom espaçamento visual etc.

Entretanto, nada disso é necessário no momento do navegador renderizar a página. Um comentário no código é completamente inútil na hora da execução. Assim como espaços, identação ou nomes de variáveis legíveis.

Mais que isso, todas essas práticas adicionam bytes e mais bytes aos arquivos CSS e JavaScript. Arquivos esses que vão ser baixados pelos navegadores de todos os nossos usuários com o único objetivo de executá-los. Porque então gastar dados trafegando comentários e outras coisas inúteis?

Uma otimização muito importante e extremamente fácil de se implementar com as ferramentas atuais é o que chamamos de **minificação dos arquivos CSS e JavaScript**.

Rodamos um programa compressor nos nossos arquivos para tirar todos esses bytes desnecessários para simples execução. O resultado são arquivos CSS e JavaScript completamente idênticos em funcionalidade mas sem bytes de comentários, espaços etc. Até variáveis longas são reescritas com nomes mais curtos - como apenas 'a', 'b' etc.

Mas repare que não estamos defendendo que você deva *escrever seu código* retirando comentários, identação etc. A boa prática continua sendo escrever código legível e bem documentado. Queremos apenas que, antes de colocar o site no ar, os arquivos sejam minificados. E, com as ferramentas automáticas de hoje em dia, é muito fácil fazer isso.

YUI Compressor

Há diversas ferramentas para compressão de CSS e JavaScript. Uma das mais famosas é o **YUI Compressor** do Yahoo!. Por ser em Java, é multiplataforma e fácil de se usar. Ele comprime tanto código CSS quanto código JavaScript.

Você pode baixá-lo em http://developer.yahoo.com/yui/compressor/

Ele é uma ferramenta de linha de comando, o que torna muito fácil automatizar sua execução antes do site ser colocado no ar, por exemplo.

Usá-lo é bem simples:

```
java -jar yuicompressor-x.y.z.jar script.js -o script-min.js
```

Este comando vai ler o arquivo `script.js`, minificar seu conteúdo e gravar o resultado em `script-min.js`. O mesmo poderia ser feito com arquivos CSS.

> **Testando online**
>
> Diversos sites oferecem uma interface Web para o YUI - e outros compressores. São úteis para você testar e ver logo o impacto sem instalar nada, mas são mais chatos para se automatizar.
>
> http://refresh-sf.com/yui/

14.5 - Compressão de imagens

Imagens são também fortes candidatas a otimizações. Quão importante será fazer isso?

O **HTTPArchive.org** armazena informações históricas coletadas mensalmente sobre os 17 mil sites mais acessados da Internet mundial. Com base nelas, compila alguns gráficos e dados interessantes.

E, com relação a imagens, os estudos mostram que mais de 60% do peso de uma página está nelas:

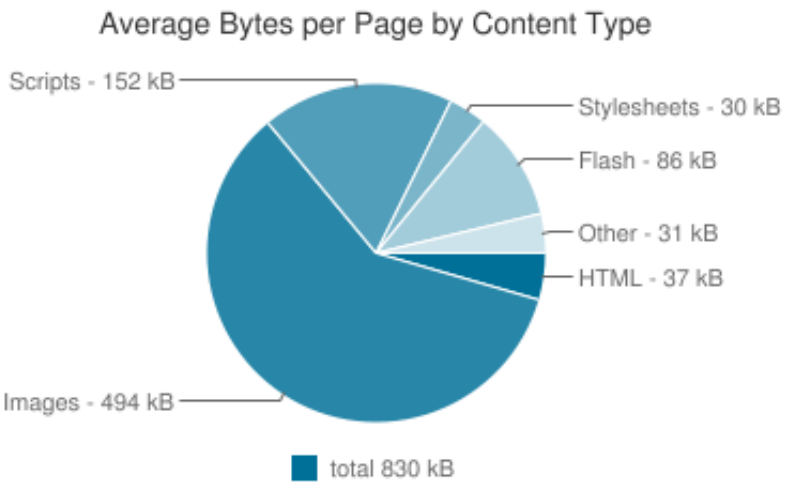


Figura 14.3: Participação de cada tipo de arquivo no tamanho total das páginas

Se conseguir otimizar um pouco as imagens, o resultado final será de grande impacto para a página!

Otimizações lossy

A otimização mais direta relacionada a isso é diminuir a qualidade das imagens. Quando você salva um JPG, pode escolher o grau de compressão, obtendo imagens menores mas sacrificando um pouco a qualidade. Chamamos esse tipo de otimização de *lossy*, pois há perda de qualidade.

É preciso avaliar até que ponto se pode sacrificar a qualidade do design em prol da performance. No entanto, tenha em mente que muitas otimizações podem acabar sendo imperceptíveis para o olho humano - ainda mais o olho ágil e desatento dos usuários Web que varrem as páginas rapidamente e podem nem perceber que uma imagem está com menos definição.

Você pode otimizar fotos JPG manualmente no seu editor de imagens favorito e chegar a um meio termo entre qualidade e tamanho final. Ou então, pode tentar o excelente serviço online **JPEGMini** que promete diminuir o tamanho das suas imagens diminuindo a qualidade de maneira praticamente imperceptível. Eles usam um algoritmo que promete simular as características da percepção do olho humano, o que lhes permite piorar a qualidade da imagem apenas em pontos que são pouco percebidos pelo nosso olhar.

http://www.jpegmini.com/

Design otimizado

Outra estratégia boa é pensar bem na hora de fazer o design - ou convencer o designer a pensar bem. Será que realmente precisamos daquele monte de imagens na página? Será que aquele ícone precisa ser truecolor ou podia ser salvo em grayscale?

Um ponto importante é que o crescimento e adoção de CSS3 tem trazido novas possibilidades de design em CSS puro que antes só eram possíveis com imagens. Bordas redondas, gradientes, sombras, etc. Usando CSS, conseguimos o mesmo efeito evitando colocar mais imagens na página.

Pense bem no seu design e na forma como o codifica. Ele pode ser um fator de peso na performance da sua página.

Otimizações lossless

A otimização mais simples e eficaz com imagens é o que chamamos de compressão *lossless*. É diminuir o tamanho da imagem sem perder absolutamente nada na qualidade.

Isso é possível porque os formatos da Web (JPEG, PNG, GIF) em geral guardam em seus arquivos mais informações do que as necessárias para renderizar a imagem. Uma foto JPEG por exemplo tem diversos metadados embutidos como horário da foto e até coordenadas de GPS, se a câmera suportar essa funcionalidade. Ou ainda PNG exportados pelos editores como Photoshop levam diversos metadados extra e muitas vezes até uma miniatura da imagem embutida no mesmo arquivo.

Tudo isso pode ser interessante para se organizar os arquivos pessoais, montar seus álbuns etc. Mas são completamente irrelevantes para a renderização na página.

Podemos usar ferramentas automáticas para retirar esses bytes extra de imagens sem perda alguma de qualidade. A ferramenta mais famosa é o **Smush.it** do Yahoo:

http://smush.it

E o próprio JPEGMini já faz isso também para nossas fotos JPEG.

E offline?

Usar o Smush.it é uma das formas mais simples e eficientes de se otimizar as imagens. Caso você queira usar algo direto no computador, recomendamos dois programas com interfaces gráficas locais:

MAC: http://imageoptim.pornel.net/ Windows: http://luci.criosweb.ro/riot/

Se o objetivo é automatizar a otimização, você provavelmente vai querer ferramentas de linha de comando. E há várias delas disponíveis (inclusive as usadas pelo Smush.it). Procure por: optipng, pngout, pngcrush, advpng, jpegoptim, gifsicle.

14.6 - Diminuir o número de requests

Todas as práticas que vimos até agora tinham como objetivo diminuir o tamanho das requisições, o volume do tráfego de dados. Há ainda outro ponto que levantamos sobre aplicações distribuídas: *diminuir o número total de requests*.

No YSlow, na aba *Components*, você pode ver todos os componentes de página que acabou de fazer. Cada imagem, arquivo JavaScript, CSS e até vídeos e Flash são requisições feitas ao servidor. Isso sem pensar no próprio HTML da página e em possíveis requests extras numa aplicação Ajax.

Cada requisição envolve uma chamada para o servidor o que gera um overhead bastante grande. A maior parte do tempo de um request é gasto em tarefas de rede (DNS, SSL, TCP/IP etc). Se você já otimizou o tamanho dos requests, verá que uma

pequena parte apenas é gasta no download dos bytes.

Fora o próprio gargalo de rede, existe uma limitação no número de requisições que um navegador faz simultaneamente a um mesmo servidor. Esse número varia bastante, mas chega a ser bem baixo em navegadores antigos (apenas 2 conexões). Hoje, nos navegadores mais modernos, gira em torno de 6 a 8 conexões. Parece um número alto - e realmente foi uma grande evolução -, mas se você começar a contar todos os arquivos externos que está usando, vai ver que há chances de uma página mediana fazer dezenas de requests.

O HTTPArchive reporta uma média de mais de 80 requests sendo feitos na página:

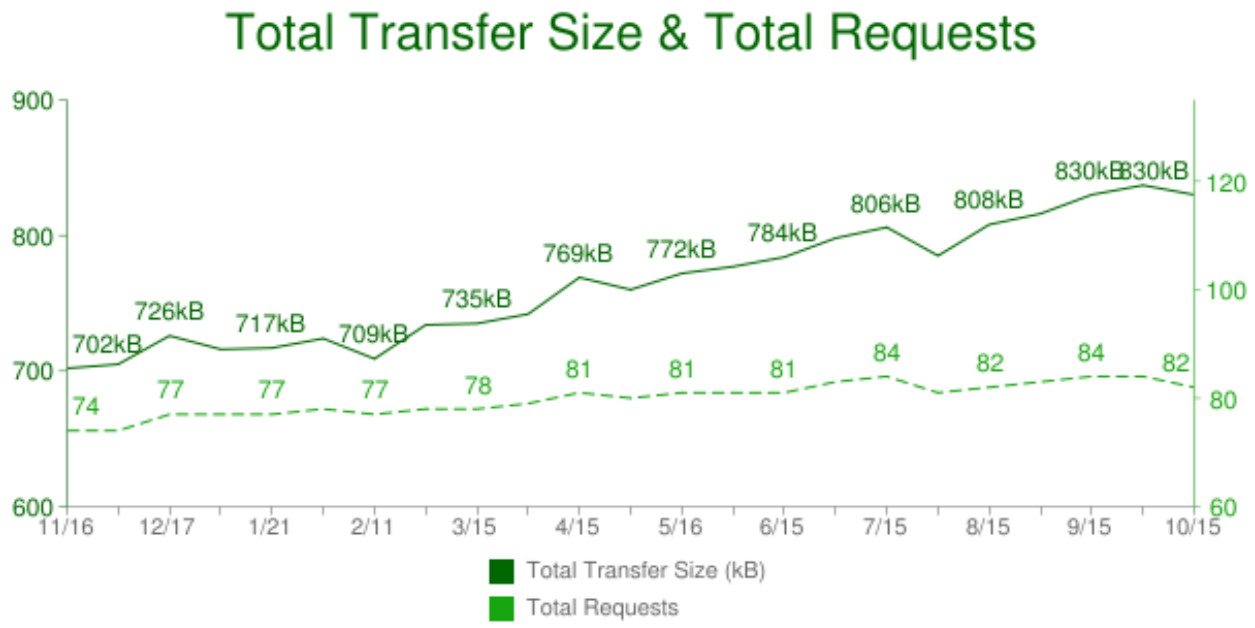


Figura 14.4: Número e volume médio de requests

Ou seja, mesmo com 8 conexões simultâneas, o volume de requests é bem maior o que vai atrasar o carregamento total da página. Em alguns casos (como arquivo JS), o navegador fica travado até que todos os downloads terminem.

Diminuir o número de requisições é essencial.

14.7 - Juntar arquivos CSS e JS

No caso de arquivos CSS e JavaScript, a boa prática é juntar diversos arquivos em um número menor para minimizar requisições.

Ao usar jQuery, por exemplo, importamos a biblioteca em si e, em geral, criamos um script a mais da nossa aplicação que vai usá-lo. É muito comum também (como veremos no curso WD-47) usarmos diversos plugins com o jQuery para fazer várias tarefas avançadas. São todos requests a mais que vão sobrecarregar nossa página.

Se vamos usar, por exemplo, jQuery com 3 plugins e mais 2 arquivos da aplicação, poderíamos simplesmente juntar todos em 1 ou 2 arquivos. Novamente, a boa prática não é escrever um código todo misturado de uma vez só em um arquivo de milhares de linhas. Faça seu código com boas práticas e execute alguma rotina que junte automaticamente esses arquivos pra você antes de subir no servidor.

O próprio jQuery faz isso. Usamos um arquivo chamado *jquery.js* mas se você for olhar o projeto oficial, verá que esse é apenas um arquivo gerado a partir de outros 22 arquivos separados, que foram criados de maneira independente para melhor organizar e encapsular as funcionalidades do projeto.

A mesma dica vale para arquivos CSS. Você pode organizar seu código em arquivos diferentes (por exemplo, um com tipografia, outro com estrutura de layout e outro com estilos visuais), porém, na hora de subir a aplicação no ar, a boa prática é diminuir o número de arquivos e juntar quantos puder.

14.8 - Image Sprites

Juntar arquivos texto como JS e CSS é muito fácil, basta copiar os conteúdos seguidamente. E as imagens? Já vimos que elas costumam ser o componente mais pesado das páginas. Em designs complexos, são responsáveis também por um grande número de requisições.

Conseguimos juntar imagens?

É possível, e essa técnica é chamada de **sprites**, que é um processo muito mais complicado. Juntar imagens consiste em criar um arquivo só e posicionar diversas imagens dentro. Depois, usando CSS, "recorta-se" os pedaços da imagem que devem ser mostradas em cada parte da página.

Diversos grandes sites usam essa técnica. O Google por exemplo, tem uma única imagem em sua home:

Criar a imagem é o primeiro passo, e um dos mais chatos. Há algumas ferramentas que tentam automatizar (como o sprite.me), mas em geral o processo é bastante manual. Abre-se um editor de imagens e se posicionam as imagens para obter o resultado final.

E, principalmente, nesse processo de juntar as imagens no editor, devemos prestar bastante atenção no posicionamento que precisa ser anotado precisamente. A posição (X,Y) de cada imagem dentro da sprite, além do tamanho (width, height) de cada uma. Essas informações serão valiosas para escrever o CSS.

Utilizamos a propriedade background do CSS do elemento como na técnica de Image Replacement, mas é preciso especificar a posição da imagem em relação ao ponto inicial do elemento.

Para saber mais - Data URI

Ainda pensando em minimizar o número de requisições para imagens, há uma outra técnica conhecida como **Data URIs**. A ideia é que você pode embutir conteúdos binários (como imagens) dentro de arquivos textos (como páginas HTML ou arquivos CSS). Basta converter os bytes da imagem para uma string comum que segue o formato de codificação chamado *Base 64*.

Esse processo de conversão é feito por algum programa que converta para base64 ou diretamente no servidor por meio de código Java, PHP etc. Há alguns serviços online que ajudam nessa tarefa, também.

Imagine que queremos colocar o seguinte logo da Caelum em nossa página:

Com HTML normal, faríamos:

```
<img src="logo-caelum.png">
```

Isso vai causar uma requisição para o arquivo `logo-caelum.png`. Usando data URIs, vamos embutir o código base64 da imagem direto na tag HTML:

```
<img
src="
JAUlEQVRo3u3baZBVxRXA8d8wbAO4oiKKC4pGo7hQYpByCcYF3LXigiAhBgWNWmiMUVFBQEstU4UE
Iz
GKC6jjluAeDaWJo5aCREARcIuAiCiMghGBYTj58C6P++C9YWAWBDn3w3u3u2/f7v/te/qc032hicE
mq
xC1fFSYbLAmNnlpYmKtw0sfEzd9iINrB1XzOD5uibdieayU72N07BnC4Gq1o73rParMx2Z5we16O0
Tj
WuxnS93d6sC6QDi5pvCOiKHxeiyLfDI6hMlraUGJa32Ut+5P9VBUS/08Rwhd6wJhjg5svU7wWsRF8
V5
UJd9ndGJV0svsVJ2V5vl8NUVw9A8dYaq5e8eYauP7SQyPhbF2EaIKPTwmW2OZ/tppmLx0XfT3mBVJ
3i
UbBcLd4rV4P5bEa3HdWvEdGi/FiqieVIGwpdeSGhc4K2+JTt4RvtVqIxmF/SIiYnY0rBLf1nFXVEb
1p
SDCYuOSOp/TumDril3myo3kRa7eKOwZ82LdpCDCm5M6f6t+5IegC/eJl2PdpQDCjomeu49NCmFVM/
Kl
sTSi9hD+QwhzbL3JIcx/FMfwWF/Ji7BzUvPJ/CgQtohnI2oX4V1CeJIfBcI2MSmithFmjOkLfhQIO
8S
ciNpGuH9S+yE/AoTtY1FE7SM8JQmFNdnkEbaKmRF1gfBCIUxZz/YeZpC73Ki7HfLml+isvxF+7ySN
qk
BY4md2Xu3atg5Pne3scOzlEnc4x1bZ6452o5udkTijhRGWxFsRdYPwGiE8sZ4In8u2cYGzV8srdoN
lq
V7M1KcAwp9bLpyWc/VE4c3s2ShhqkGWJnWV64COyrO1T9K+CoRF8XhEXSHsJ4R31gtgkX8b540kvr
RU
21TebonPvcIM48xM+nJHEi7LRdhVrIFweg7C0jWolOtrYU7KLNsWRHhzRN0hPFUIy2oUUN3SQCHcl
k1
pYqpQ6U7bJSnHmiWEq2uAcIVb7KyNodmo0UeOtqWjTBXC8AIIe0fUJcIDk/scXCMd3szXwuPZ85uF
yp
SuK9Ldf4Xwne3XG+Ffsuejkwe/d1ZvLhMm5EW4eyyuW4RFvhDC+esBbjeXeshwPTTylPBKkr6X5cI
12
XJHmZD053kH1GAU7pc97yaEZ1Lly4QleRGOjahbhNwrhIfWGeB5FmXb+R/vCP9KcvoIFZqBfT2dlB
mv
S4HppHoIK1PKZh8h3J4q//DKvq0G8ISIukd4THK3dQvpn5FV4h9m27sS4UhhPFoZabkQPnRWzrrL+
iB
cksrdXQiDUin350XYND6qD4S8KoRPNK82wK18LsxNLLd9vZ+D8FXhJTf4VgjzXJxjE9YnwoER9YPw
iO
SOw6qN8HQh9Mue/zwH4X3ZPnxroBZ5rq8fhG3j+/pCyPBkxe6X1UQ4RFhim9Sk9GkK4cXJfDmigM9
SH
```

```
YTzawHhoxH1h7CRN5L7PpB1nvLJKQaCW4XFtkzlfJRCmDGUXl3j9S2E8Egh3JLK30PUGOGOUVGfCG
lt
UnLv2Y7NW6KDV4TQBWcKkXLYDs95kVfO8mNWW7pvrkFehFtZIcxKlX6kFhBeE1G/CGmRNT/CM3pn/
Yo
SHV2gNOsRPIotzBU+S7Z07JrgX4Vwa3MS37tbYoq0NMBX9s+LMKP5wjAt0DYxUGqGsCg+qX+ENHCZ
+S
llstAM0xKzJJJZu3syVnomLtcUE1QIK3IQcmjijYSFXk6cu9C7AMKuKhNt/LUQlptXQ4THRWwIhJn
xc
5t5eX31+S7P8aT7WJLNm+Ci1RCyRdKtVdtLntGhAEIuTkygTNTnGGNriPCJDYcwMxqP8EcvetcCC7
3h
bpfqYos1yu2tv8fd7dcaa+v0PPPvns41zLNGuMSuqfR9dbffqhgfaOVapR7UR2uc5DjbZ/NOcKJdc
uz
S3+ioJJVyvJMyd6jDqWQdEG7UkqwSx2aENUNYuhlhTRHO3oywZgh3iahvhP2zR6cqW3dIjfYWbuvq
TGi
+FqSxgxXbI98utBDn1D/CUKZUqdK1+Mhne74G3d5H2KeWEL7pJXNM1yYvwuEbAuFpa6Q1zf4ryrtd
qTj
HJGmWd6RUhbBpKqco50xOTrqWFkm5Btoq1iyV11hxCuHbGxphmUHeV+Fvyciba7xJ2qCXMnTwmSt8
rVx
PcIDplvhstYX1O80zLWdUr0Q4wIOesswMuyVqZIHFXrGjYtOcAAa4H0PM9aV7NEaxMcr9z1VglHnm
JMt
dDY2x0HdGZsIaoc5swioR3pvowm0w3ccOtJ/lOmtisb64yvXoZzo6CaNsY6gPwFRDNXeTuakwwXne
1Ng
RylPRnJUIb1HpLC1NNATtLXKUnbxqGO5zN5joV3b1thIlvtINV3hNc+0st6dOXtDQDiochN+ZpK1D
zc8
MhBCxIRC+ZayxxtoZ010OPtADE43IBq1WIdwGnVXIuFrbo3XOKuCj7neYQyzVOQ/CF8Fwo3GlceBS
M9D
NHOxgSVZ5/MIyvTEuaVUnLZOXuUfiIo5zHXgqs8K3oRCmX+TpSTR6ur7YV6m5HvHTHIRNk98mzrUg
uW6
BXtk6pvg0eSj5EI4Fw/wdj7sTmRWcxhopd7Ceydrchd4zyhf64hPdU238g2lGWOxUfOtMcJM3fngI
+2X
V+LPmKMqDsKnjhO3QQqXjsnWUGgIOTUW210Q4Fn/1T3C+8kTLXe9hvXC27+yNmfphYrKo2sfu+vvK
jhp
Y4jR8YgB4MKO9K9RBuH+lFPx0J1zuIAc5SOvVEDY0y2G43sd5R2FT21roVi0MsSg1c/czwdZ2tzi1
2Sg
/wh4WOdYuyjwGunrbXFvhahPQUbgIVxpvR0daaid/9iROSx79KFPspbO5+sJkMbrOEBb8gOxrC8z3
pXk
GoCx5ITO/p3jdDO9on5qRv9Ak9Xuybyz1jdNzDJonlfvYOam0dr7QDgOSbfGD3YcG/qTSUm8nkZxG
Pkj
WtNuZapqnXWO8EiVGKleuHzr60LtGu82LaKXMMpUeyOjswWLPGF0HI3EdP2PMlbXtPSzWZpVdltfy
q1q
aF/wcqGXqfyNPprynXD9nu1WhuM0f0xaWZrr6vjoPZvMn3YXkRPe4bG2F/g9S7CNMkQtgRwAAAABJ
RU5
ErkJggg==">
```

O resultado é assustador e parece até pior com relação a tamanho, mas lembre-se de que estamos economizando os bytes da imagem. E essa string dentro do HTML, após o GZIP, costuma ter um tamanho muito próximo ao que seria a imagem binária original.

Você pode usar essa técnica também em arquivos CSS, dentro de `background-image` por exemplo.

O maior problema dessa técnica é que ela não é suportada em todos os

navegadores. Todos os mais modernos suportam, já o IE6 e IE7 e mesmo o IE8 possui algumas limitações. Além disso, o processo de geração dessa string base64 costuma exigir um pouco de conhecimento de programação no servidor.

14.9 - Para saber mais

A Caelum tem vários posts no Blog sobre o assunto, sendo o principal:

http://blog.caelum.com.br/por-uma-web-mais-rapida-26-tecnicas-de-otimizacao-de-sites/

14.10 - Exercícios: otimizações Web

1. Rode as ferramentas de análise do PageSpeed Online e do WebPageTest:

https://developers.google.com/pagespeed/

http://webpagetest.org/

Analise o resultado. Veja possíveis pontos de melhora.

2. Nossos banners principais na home são fotografias imensas, mas estão em PNG. O formato ideal para eles é JPG, que traz uma qualidade satisfatória com muito menos kbytes.

Faça a conversão dos banners de PNG para JPG e compare os resultados.

3. Comprima as imagens do projeto para economizarmos no tamanho.

Os PNGs e GIFs podem ser comprimidos sem perdas no Smush.it do Yahoo:

http://smushit.com

Um serviço alternativo é o http://kraken.io

Os JPEGs podem ser comprimidos diminuindo sua qualidade e o seu tamanho sem muita perda de qualidade. Uma ótima ferramenta pra isso é o JPEGMini: http://jpegmini.com

4. Criamos vários arquivos CSS e JavaScript na nossa home. Podemos juntá-los para obter um ganho de performance.

Nos CSS, temos dois arquivos: `estilos.css` e `mobile.css`. Podemos juntá-los num único arquivo. (dica: o mobile.css é importado com media query; para juntá-lo no arquivo principal, você precisará escrever a media query corretamente dentro do CSS)

Nos JavaScripts, podemos, por exemplo, juntar o jQuery e o jQuery UI num único arquivo.

5. Comprima os arquivos CSS e JavaScript.

6. Depois dessas otimizações, teste novamente no PageSpeed e WebPageTest. Houve alguma melhora?

CAPÍTULO ANTERIOR:

Integrações com serviços Web

PRÓXIMO CAPÍTULO:

Apêndice - LESS

Blog Caelum

Cursos Online

Facebook

Newsletter

Casa do Código

Twitter