

O guia amigo do seu cérebro

Use a Cabeça!

(Head First)

Programação em HTML5



Aprenda os segredos
do guru da HTML5



Descubra por que
tudo o que seus
amigos sabem sobre
vídeo provavelmente
esteja errado

Evite problemas
constrangedores
de suporte a
navegadores



Um guia de
aprendizagem para
criar aplicativos web
com JavaScript



Carregue HTML5 e
JavaScript direto
no seu cérebro



Fique atento a
armadilhas comuns
de navegadores

Primeiros Elogios ao *Use a Cabeça! Programação em HTML5*

“HTML5 é o ‘futuro da web’. Quantas vezes você ouviu isso? Se realmente quiser entender a família de tecnologias que constituem HTML5, leia este livro! O *Use a Cabeça! Programação em HTML5* é o livro definitivo sobre HTML5 para todos, de iniciantes a desenvolvedores experientes.”

— **Aaron LaBerge, CEO, Fanzter, Inc.**

“Este livro é uma viagem agradável pelos novos territórios selvagens de HTML5, onde todos provavelmente teremos de lutar com escorpiões por anos. Ele guia-nos através dos conceitos básicos, de modo que possamos entender os objetivos do projeto da HTML5, e também através de cada área, para conhecermos os arredores. Da mesma forma que todos os livros da série Use a Cabeça!, ele substitui a recitação árida por explosões vivas e memoráveis de conhecimento cheio de fatos. Sempre terei o site da especificação do HTML5 para propósito de referência, mas preferiria *aprender* animadamente.”

— **Ken Arnold, Design/Build Hub, Peak Impact, Inc.**

“Um livro obrigatório sobre HTML5, que continua a tradição da série Use a Cabeça! de ser espirituoso, divertido, cheio de exemplos e muito inteligente!”

— **Danny Mavromatis, Sr Software Architect, ABC Television Group**

“*Use a Cabeça! Programação em HTML5* faz um ótimo trabalho ao explicar muitos dos aspectos-chave de HTML5 de uma forma divertida e fácil de entender. Com seu estilo altamente visual e numerosos exemplos de código, conceitos complexos como o canvas e a programação assíncrona são simplificados e ilustrados, tornando-os diretos e interessantes.”

— **Michael S. Scherotter, Principal Architect Evangelist, Microsoft Corporation**

“HTML5 é um bolo com muitas camadas de tecnologias. *Use a Cabeça! Programação em HTML5* assa esse bolo e depois joga-o em sua cara. Você consumirá essa delícia e ficará feliz.”

— **Josh Rhoades, cofundador da BrightHalf**

Com *Use a Cabeça! Programação em HTML5*, a multiplicidade do HTML5 é abordada de múltiplas formas, o que torna divertido o trabalho árduo de aprender.

— **Ward Cunningham, inventor do Wiki**

“HTML5 é a tecnologia nova mais impressionante para o desenvolvimento de sites. Os desenvolvedores mal podem esperar para usá-lo para criar páginas flexíveis e de mídia rica, que também funcionem bem em tablets e smartphones. *Use a Cabeça! Programação em HTML5* é a melhor e mais divertida forma de colocar esta tecnologia nova e interessante no seu cérebro. Eu o recomendo enfaticamente!”

— **Marianne Marck, SVP Technology, Blue Nile Inc.**

Primeiros Elogios ao *Use a Cabeça! Programação em HTML5*

“Direto, informativo e divertido, *Use a Cabeça! Programação em HTML5* é obrigatório para qualquer pessoa que queira iniciar em HTML5 ou apenas refrescar sua memória. A série Use a Cabeça! ajuda a manter meu conhecimento atualizado, permitindo-me apoiar melhor meus desenvolvedores e projetos.”

— **Todd Guill, Project Manager, AllRecipes.com**

“Não é o DHTML do seu avô! *Use a Cabeça! Programação em HTML5* pinta um cenário promissor e confiante da web através da HTML5, enquanto lhe habilita a codificar a sua própria entrada lá. Se você estiver procurando um guia definitivo, acessível e às vezes bastante divertido para este padrão, não procure mais.”

— **Manny Otto, Web Producer and Creative**

“Os autores acertaram na mosca — conhecimento em JavaScript é a chave para o HTML5. Mesmo se você nunca tiver escrito um programa em JavaScript antes, eles lhe iniciarão através de projetos divertidos e práticos.”

— **David Powers, autor de *PHP Solutions: Dynamic Web Design Made Easy***

Elogios a Outros Livros de Eric Freeman e Elisabeth Robson

“A admirável clareza, humor e doses substanciais de inteligência deste livro o tornam o tipo que ajuda até mesmo aos não programadores a pensar bem sobre a resolução de problemas.”

— **Cory Doctorow, coeditor de *Boing Boing* e autor de *Down and Out in the Magic Kingdom* e *Someone Comes to Town, Someone Leaves Town***

“Sinto-me como se quinhentos quilos de livros tivessem sido tirados de cima da minha cabeça.”

— **Ward Cunningham, inventor do Wiki e fundador do Hillside Group**

“Este livro está próximo da perfeição, devido à forma pela qual combina conhecimento e legibilidade. Ele fala com autoridade e é lido maravilhosamente. É um dos poucos livros de software que já li que me parece indispensável. (Eu colocaria talvez dez livros nesta categoria, no máximo.)”

— **David Gelernter, professor de Ciência da Computação, Yale University e autor de *Mirror Worlds* e *Machinery Beauty***

“Eu literalmente amo este livro. Na verdade, bejei este livro na frente da minha esposa.”

— **Satish Kumar**

“Tome cuidado. Se você for alguém que lê à noite antes de dormir, terá de restringir o *Use a Cabeça! Programação em HTML com CSS & XHTML* à leitura diurna. Este livro acorda seu cérebro.”

— **Pauline McNamara, Center for New Technologies and Education, Fribourg University, Suíça**

“*Use a Cabeça Programação em HTML com CSS & XHTML* é uma introdução completamente coerente às práticas de antevi看过 na marcação e apresentação de páginas web. Antecipa corretamente as dúvidas dos leitores e lida com elas na hora. A abordagem altamente gráfica e incremental simula com precisão a melhor forma de aprender este assunto: fazendo uma pequena alteração e vendo-a no navegador para entender o que cada novo item significa.”

— **Danny Goodman, autor de *Dynamic HTML: The Definitive Guide***

“A web seria um lugar muito melhor, se cada autor de HTML começasse lendo este livro.”

— **L. David Baron, Technical Lead, Layout & CSS, Mozilla Corporation
<http://dbaron.org/>**

“O *Use a Cabeça! Programação em HTML com CSS & XHTML* lhe ensina como fazer as coisas certas desde o início, sem fazer com que todo o processo pareça massacrante. HTML, quando explicado apropriadamente, não é mais complicado do que a própria língua portuguesa, e eles fazem um trabalho excelente mantendo cada conceito ao alcance de seus olhos.”

— **Mike Davidson, President & CEO, Newsvine, Inc.**

Outros livros da série Use a Cabeça!

Use a Cabeça! Ajax
Use a Cabeça! Ajax Profissional
Use a Cabeça! Álgebra
Use a Cabeça! Análise e Projeto Orientado ao Objeto
Use a Cabeça! Análise de Dados
Use a Cabeça! C#, Tradução da 2^a Edição
Use a Cabeça! Desenvolvimento de Software
Use a Cabeça! Desenvolvendo para iPhone
Use a Cabeça! Estatística
Use a Cabeça! Excel
Use a Cabeça! Física
Use a Cabeça! Geometria 2D
Use a Cabeça! HTML com CSS & XHTML, Tradução da 2^a Edição
Use a Cabeça! Java 2^a Edição
Use a Cabeça! JavaScript
Use a Cabeça! Padrões de Projetos 2^a Edição
Use a Cabeça! Programação
Use a Cabeça! PHP & MySQL
Use a Cabeça! PMP
Use a Cabeça! Python
Use a Cabeça! Rails, Tradução da 2^a Edição
Use a Cabeça! Redes de Computadores
Use a Cabeça! Servlets & JSP 2^a Edição
Use a Cabeça! SQL
Use a Cabeça! Web Design

A compra deste conteúdo não prevê o atendimento e fornecimento de suporte técnico operacional, instalação ou configuração do sistema de leitor de ebooks. Em alguns casos, e dependendo da plataforma, o suporte poderá ser obtido com o fabricante do equipamento e/ou loja de comércio de ebooks.

Use a Cabeça!

Programação em HTML 5

desenvolvendo aplicativos para web com JavaScript



Não seria um sonho, se houvesse um livro de HTML5 que não presumisse que você já saiba o que é DOM, eventos e APIs logo na terceira página? Isso provavelmente é apenas uma fantasia...

Eric Freeman
Elisabeth Robson

Use a Cabeça! Programação em HTML5

Copyright © 2014 da Starlin Alta Editora e Consultoria Eireli.

Translated from original Head First HTML5 Programming © 2011 by Eric Freeman and Elisabeth Robson. ISBN 978-1-449-39054-9. This translation is published and sold by permission O'Reilly Media, Inc. The owner of all rights to publish and sell the same. PORTUGUESE language edition published by Starlin Alta Editora e Consultoria Eireli. Copyright © 2014 by Starlin Alta Editora e Consultoria Eireli.

Todos os direitos reservados e protegidos por Lei. Nenhuma parte deste livro, sem autorização prévia por escrito da editora, poderá ser reproduzida ou transmitida.

Erratas: No site da editora relatamos, com a devida correção, qualquer erro encontrado em nossos livros. Procure pelo título do livro.

Marcas Registradas: Todos os termos mencionados e reconhecidos como Marca Registrada e/ou Comercial são de responsabilidade de seus proprietários. A Editora informa não estar associada a nenhum produto e/ou fornecedor apresentado no livro.

Impresso no Brasil — 1^a Edição, 2014

Vedada, nos termos da lei, a reprodução total ou parcial deste livro.

| | | | | |
|---|--|---|--|--|
| Produção Editorial Editora Alta Books | Supervisão Gráfica Angel Cabeza | Design Editorial Auleriano Messias | Captação e Contratação de Obras Nacionais Cristiane Santos Marco Pace J. A. Rugeri autoria@altabooks.com.br | Vendas Atacado e Varejo Daniele Fonseca Viviane Paiva comercial@altabooks.com.br |
| Gerência Editorial Anderson Vieira | Supervisão de Qualidade Editorial | Aurélio Silva | | |
| Editoria Informática Thiê Alves | Supervisão de Texto Sergio Luiz de Souza Jaciara Lima | | | Marketing e Promoção marketing@altabooks.com.br |
| | | | | Ouvidoria ouvidoria@altabooks.com.br |
| Equipe Editorial | Claudia Braga Hannah Carriello Letícia Vitória | Livia Brazil Marcelo Vieira Mayara Coelho | Milena Lepsch Milena Souza Natália Gonçalves | Rodrigo Araujo |
| Tradução Leonardo Castilhone | Copidesque Alessandra Santos | Revisão Técnica Flávio Augusto Silveira <i>Analista de Tecnologia</i> | Revisão Gramatical Glória Melgarejo Milena Dias | Diagramação Lucia Quaresma |

Dados Internacionais de Catalogação na Publicação (CIP)

| | |
|-------|--|
| F855u | Freeman, Eric. Use a cabeça! : programação em HTML 5 : desenvolvendo aplicativos para web com JavaScript / Eric Freeman, Elisabeth Robson. – Rio de Janeiro, RJ : Alta Books, 2014. 608 p. : il. ; 24 cm. – (Use a cabeça!) Inclui índice e apêndice. Tradução de: Head First HTML5 Programming : building web Apps with Javascript. ISBN 978-85-7608-845-5 |
| | 1. HTML (Linguagem de marcação de documento). 2. JavaScript (Linguagem de programação de computador). 3. Programação para Internet. 4. Sites da Web - Desenvolvimento. I. Robson, Elisabeth. II. Título. III. Série. |

CDU 004.738.52
CDD 006.74

Índice para catálogo sistemático:

1. Linguagem de marcação de documento : HTML5 004.738.52

(Bibliotecária responsável: Sabrina Leal Araujo – CRB 10/1507)



Rua Viúva Cláudio, 291 – Bairro Industrial do Jacaré
CEP: 20970-031 – Rio de Janeiro – Tels.: (21) 3278-8069/8419
www.altabooks.com.br – e-mail: altabooks@altabooks.com.br
www.facebook.com/altabooks – www.twitter.com/alta_books

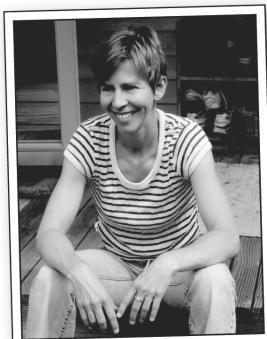
Para Steve Jobs, que promoveu a HTML5 a tal ponto
que este livro deverá vender um zilhão de cópias...

E para Steve Jobs, porque ele é nosso herói.

Autores de Use a Cabeça! Programação em HTML5



← Eric Freeman



↓ Elisabeth
Robson

Eric é descrito pela coautora da série *Use a Cabeça!*, Kathy Sierra, como “um daqueles raros indivíduos fluentes na linguagem, prática e cultura de múltiplos domínios: hackeamento moderno, vice-presidência corporativa, engenheiro e grupos de pesquisa.”

Profissionalmente, Eric concluiu, há pouco tempo, cerca de uma década como executivo de empresa de mídia — tendo ocupado o cargo de CTO da Disney Online & Disney.com na The Walt Disney Company. Eric dedica agora seu tempo para a WickedlySmart, uma empresa jovem cocriada com Elisabeth.

Com relação a sua formação, Eric é cientista da computação, estudou com o erudito da indústria David Gelernter, durante seu trabalho em Ph.D. na Yale University. Sua dissertação é considerada como o trabalho de referência em alternativas à metáfora do desktop, e também como a primeira implementação de ramos de atividade, um conceito que ele e o Dr. Gelernter desenvolveram.

Em seu tempo livre, Eric é profundamente envolvido com a música; você encontrará o último projeto de Eric, uma colaboração com o pioneiro da música ambiente, Steve Roach, disponível na App Store para o iPhone, sob o nome de Immersion Station.

Eric mora com sua esposa e com sua pequena filha em Bainbridge Island. Sua filha é uma visitante constante do estúdio de Eric, onde ela adora ligar e desligar os botões de seus sintetizadores e efeitos de som. Eric também é apaixonado por educação e nutrição para crianças e busca maneiras de aprimorá-las.

Escreva para Eric, eric@wickedlysmart.com ou visite seu site <http://ericfreeman.com>.

Elisabeth é uma engenheira de software, escritora e treinadora. Ela é apaixonada por tecnologia desde seu tempo como estudante na Yale University, onde concluiu seu Mestrado em Ciência da Computação, ao mesmo tempo em que desenvolveu uma linguagem de programação visual e arquitetura de software.

Elisabeth está envolvida com a internet desde muito tempo; ela ajudou a criar o site vencedor de prêmios, The Ada Project, um dos primeiros desenhados para auxiliar as mulheres da Ciência da Computação a encontrarem informações sobre carreira e orientação online.

Ela é atualmente cofundadora da WickedlySmart, uma experiência de educação online centrada em tecnologias web, onde cria livros, artigos, vídeos e muito mais. Anteriormente, como diretora de Projetos Especiais na O'Reilly Media, Elisabeth produziu workshops ao vivo e cursos online numa variedade de tópicos técnicos e desenvolveu sua paixão por criar experiências de aprendizado para ajudar pessoas a entenderem a tecnologia. Antes de seu trabalho com a O'Reilly, Elisabeth passava seu tempo jogando pó mágico na The Walt Disney Company, onde liderou pesquisas na área de desenvolvimento em mídia digital.

Quando não está na frente de seu computador, Elisabeth está escalando, pedalando ou andando de caiaque por aí, com sua câmera pendurada no pescoço, ou cozinhando refeições vegetarianas.

Mande um e-mail para ela em beth@wickedlysmart.com ou visite seu blog <http://elisabethrobson.com>.

Conteúdo (Sumário)

| | | |
|----|---|-----|
| | Introdução | xxi |
| 1 | Conhecendo a HTML5: <i>Bem-vindo a Webville</i> | 1 |
| 2 | Apresentando JavaScript e DOM: <i>Um Pouco de Código</i> | 35 |
| 3 | eventos, manipuladores e todo esse balanço: <i>Um Pouco de Interação</i> | 85 |
| 4 | Funções e objetos em JavaScript: <i>JavaScript Sério</i> | 113 |
| 5 | Tornando ativa sua localização html: <i>Geolocalização</i> | 165 |
| 6 | Falando com a web: <i>Aplicativos Extrovertidos</i> | 213 |
| 7 | Descobrindo seu artista interior: <i>O Canvas</i> | 281 |
| 8 | Não é a TV de seu pai: <i>Vídeo ... com a participação especial da estrela “Canvas”</i> | 349 |
| 9 | Armazenando coisas localmente: <i>Web Storage</i> | 413 |
| 10 | Pondo o javascript para funcionar: <i>Web Workers</i> | 473 |
| | Apêndice: Os dez tópicos mais importantes (que não vimos) | 531 |
| | Índice | 549 |

Conteúdo (A Coisa Real)

Introdução

Seu cérebro em Programação HTML5. Aqui está você tentando aprender algo, enquanto seu cérebro está lhe fazendo um favor se certificando de que o aprendizado não tenha sucesso. Seu cérebro está pensando: “Melhor deixar mais espaço para coisas mais importantes, como quais animais selvagens evitar e se é uma má ideia esquiar na neve pelado”. Então, como enganar seu cérebro para que ele pense que sua vida depende de saber HTML5 e JavaScript?

| | |
|---|-------|
| A quem se destina este livro? | xxii |
| Sabemos o que você está pensando. | xxiii |
| E sabemos o que o seu <i>cérebro</i> está pensando. | xxiii |
| Metacognição: pensando sobre pensar | xxv |
| Revisão Técnica | xxx |
| Agradecimentos | xxxii |

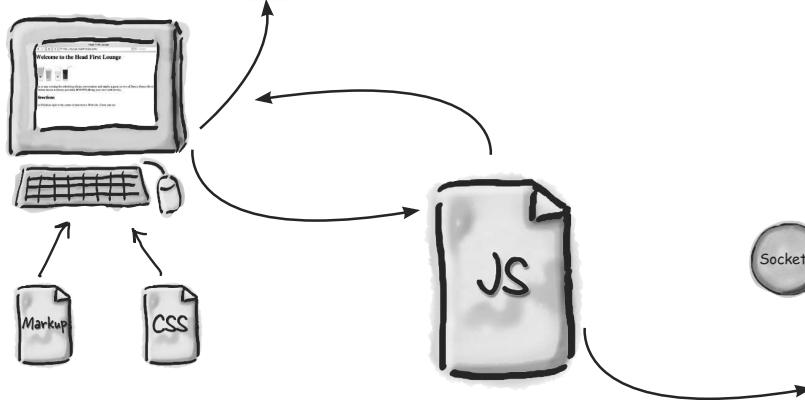
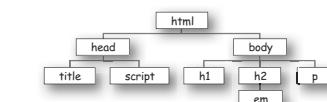
conhecendo a HTML5

1

Bem-vindo a Webville

A HTML tem estado em uma corrida louca. É claro, começou como uma simples linguagem de marcação, porém, mais recentemente, tem desenvolvido mais músculos. Agora, temos uma linguagem ajustada para criar verdadeiros aplicativos web com armazenamento local, desenho 2D, suporte offline, sockets, threads e mais. A história da HTML nem sempre foi bela e é cheia de dramas (falaremos sobre isso tudo), mas, neste capítulo, primeiro daremos uma pequena volta por Webville para ter uma ideia de tudo o que se relaciona com “HTML5”. Vamos lá, suba, estamos indo para Webville, e começaremos indo de zero a HTML5 em 3,8 páginas.

| | |
|--|----|
| Atualize para HTML5 hoje! | 2 |
| Apresentando o HTML-o-mático , atualize sua HTML agora! | 4 |
| Você está mais próximo da marcação em HTML5 do que imagina! | 7 |
| HTML5 Exposto: Confissões da versão mais nova de HTML | 11 |
| A VERDADEIRA HTML5, por favor, levante-se ... | 12 |
| Como HTML5 realmente funciona... | 14 |
| Quem Faz o Quê? | 16 |
| Sua Primeira Missão: Reconhecimento de Navegadores | 17 |
| O que você pode fazer com JavaScript? | 22 |
| Escrevendo JavaScript Seriamente | 25 |
| Escrevendo JavaScript Seriamente Revisto... | 26 |
| Pontos de Bala | 31 |
| Exercício — Solução | 33 |



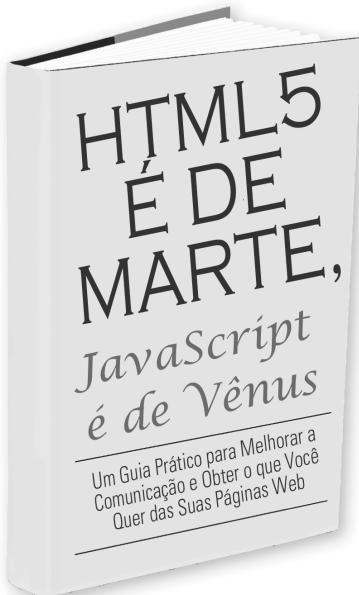
2

apresentando JavaScript e DOM

Um Pouco de Código

JavaScript lhe levará a novos lugares. Você já sabe tudo

sobre marcação HTML (conhecida como estrutura) e sabe tudo sobre estilo CSS (conhecido como apresentação), mas ainda está faltando o JavaScript (conhecido como comportamento). Se tudo o que você souber for relacionado à estrutura e apresentação, certamente poderá criar algumas páginas com ótima aparência, mas elas ainda serão apenas páginas. Quando você adiciona comportamento com JavaScript, pode criar uma experiência interativa ou, melhor ainda, pode criar aplicativos web completos. Apronte-se para adicionar a habilidade mais interessante e versátil ao seu kit de ferramentas web: JavaScript e programação!



| | |
|---|----|
| Como JavaScript funciona | 36 |
| O que você pode fazer com JavaScript? | 37 |
| Declarando uma variável | 38 |
| Como dar nomes às suas variáveis | 40 |
| Tornando-se Expressivo | 43 |
| Fazendo coisas repetidamente... | 46 |
| Tome decisões com JavaScript | 49 |
| Tomando mais decisões... e adicionando um default | 50 |
| Como e onde adicionar JavaScript às suas páginas | 53 |
| Como JavaScript interage com sua página | 54 |
| Como cozinar seu próprio DOM | 55 |
| Uma primeira prova de DOM | 56 |
| HTML5 é de Marte, JavaScript é de Vênus | 58 |
| Faça um test drive nos planetas | 62 |
| Você não pode mexer com o DOM até a página estar inteiramente carregada | 64 |
| Então, em que mais o DOM é bom, afinal? | 66 |
| Podemos conversar sobre JavaScript novamente? | 67 |
| Ou, como armazenar múltiplos valores em JavaScript | 71 |
| O Phrase-o-Matic | 75 |
| Pontos de Bala | 75 |
| Exercício — Solução | 77 |

3

eventos, manipuladores e todo esse balanço

Um Pouco de Interação

Você ainda não alcançou seu usuário. Você aprendeu os conceitos básicos sobre JavaScript, mas será que consegue interação com seus usuários? Quando as páginas respondem a entradas deles, não são mais apenas documentos; são aplicativos vivos e interativos. Neste capítulo, você aprenderá a lidar com uma forma de entrada de usuário e a conectar um elemento `<form>` antigo de HTML a código real. Talvez soe perigoso, mas é poderoso. Aperte seu cinto de segurança. Este é um capítulo que vai diretamente ao ponto, em que vamos de zero a aplicativos interativos em um segundo.

| | |
|--|-----|
| Apronte-se para o Webville Tunes | 86 |
| Iniciando... | 87 |
| Mas nada acontece quando clico em “Adicionar Música” | 88 |
| Lidando com Eventos | 89 |
| Criando um Plano... | 90 |
| Obtendo acesso ao botão “Adicionar Música” | 90 |
| Dando um manipulador de clique para o botão | 91 |
| Um olhar mais próximo no que acabou de acontecer... | 92 |
| Obtendo o nome da música | 94 |
| Como adicionamos uma música à página? | 97 |
| Como criar um novo elemento | 99 |
| Adicionando um elemento ao DOM | 100 |
| Junte tudo... | 101 |
| ... e faça um test drive | 101 |
| Revisão — o que acabamos de fazer | 102 |
| Como adicionar o Código Pronto para Assar... | 105 |
| Integrando seu Código Pronto para Assar | 106 |
| Pontos de Bala | 108 |
| Exercício — Solução | 110 |

funções e objetos em JavaScript

4

JavaScript Sério

Você já pode se chamar de scripter? Provavelmente, você já sabe fazer bastante coisa em JavaScript. No entanto, quem quer ser um scripter quando pode ser um programador? É hora de ficarmos sérios e acelerar um pouco — é hora de você aprender sobre **funções e objetos**. Eles são a chave para escrever um código que seja mais poderoso, melhor organizado e de mais fácil manutenção. Eles também são amplamente usados pelas APIs JavaScript HTML5, de modo que, quanto melhor você entendê-los, mais rapidamente poderá passar para uma nova API e começar a lidar com ela. Prepare-se: este capítulo irá requerer sua atenção exclusiva...



| | |
|---|-----|
| Expandindo seu vocabulário | 114 |
| Como adicionar suas próprias funções | 115 |
| Como uma função funciona | 116 |
| A Anatomia de uma Função | 121 |
| Variáveis Locais e Globais | 123 |
| Conhecendo o escopo das suas variáveis locais e globais | 124 |
| Ah, mencionamos que funções também são valores? | 128 |
| Alguém disse “Objetos”? | 131 |
| Como criar um objeto em JavaScript | 132 |
| Algumas coisas que você pode fazer com objetos | 133 |
| Vamos falar sobre passar objetos para funções | 136 |
| Objetos podem ter comportamento também... | 142 |
| Enquanto isso, de volta ao Cine Webville... | 143 |
| Adicionando a palavra-chave “this” | 145 |
| Como criar um construtor | 147 |
| Como “this” realmente funciona? | 149 |
| Faça o test drive do seu construtor direto do chão da fábrica | 153 |
| O que é o objeto window mesmo? | 155 |
| Um exame mais de perto em window.onload | 156 |
| Outro exame no objeto document | 157 |
| Um exame mais próximo em document.getElementById | 157 |
| Mais um objeto sobre o qual se pensar: seus elementos objetos | 158 |
| Pontos de Bala | 160 |

tornando ativa sua localização html

5

Geolocalização

Aonde quer que vá, lá está você. Às vezes, saber onde se está faz toda a diferença (principalmente para um aplicativo web). Neste capítulo, vamos mostrar como criar páginas da web que são localizáveis — em alguns casos, será possível apontar o local exato dos usuários e, em outros, será apenas possível determinar a área da cidade em que se encontram (mas ainda assim saberemos qual a cidade!). Infelizmente, às vezes não será possível determinar nada em relação a sua localização, o que pode se dar por motivos técnicos ou apenas por não querer ninguém se metendo em suas vidas. Vá descobrir sozinho. De qualquer modo, neste capítulo, vamos explorar uma API JavaScript: Geolocalização. Arranje o melhor dispositivo de localização que puder (mesmo que seja seu computador pessoal) e vamos começar.



| | |
|---|-----|
| Localização, Localização, Localização | 166 |
| A Latitude e a Longitude da Questão... | 167 |
| Como a API de Geolocalização determina sua localização | 168 |
| Mas diz aí: onde você está? | 172 |
| Como tudo se encaixa | 176 |
| Revelando nossa localização secreta... | 179 |
| Escrevendo o código para encontrar a distância | 181 |
| Como adicionar um mapa à sua página | 183 |
| Colocando um alfinete... | 186 |
| Outras coisas legais que você pode fazer com a API do Google Maps | 188 |
| Podemos falar sobre sua precisão? | 191 |
| “Onde quer que vá, lá está você” | 192 |
| Inicializando o aplicativo | 193 |
| Retrabalhando nosso antigo código... | 194 |
| Hora de seguir em frente! | 196 |
| Você tem algumas opções... | 198 |
| O mundo das timeouts e maximum age... | 199 |
| ► Tente Isso em Casa (Levando a Geolocalização até o Limite) | 202 |
| Vamos finalizar este aplicativo! | 204 |
| Integrando nossa nova função | 205 |
| Pontos de Bala | 207 |
| Exercício — Solução | 209 |

falando com a web

6

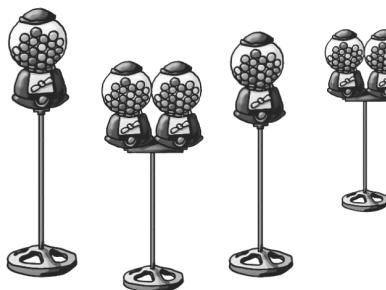
Aplicativos Extrovertidos

Você esteve na sua página por muito tempo. Está na

hora de sair um pouquinho, conversar com outros serviços web, recolher informações e trazer isso tudo de volta para poder construir ótimas experiências, mesclando o que há de melhor nelas. Essa é uma parte muito importante, quando escrevemos aplicativos HTML5 modernos, mas, para isso, é preciso saber como conversar com os serviços web. Neste capítulo, vamos fazer exatamente isso e incorporar alguns dados, a partir de um serviço web real, bem na sua página. Depois de ter aprendido como fazer isso, você será capaz de se aproximar e tocar qualquer serviço web que quiser. Vamos inclusive apresentá-lo à nova e mais badalada linguagem utilizada para se falar com os serviços web. Então, venha. Você usará mais algumas APIs: as APIs de comunicação.



Cuidado com o suspense
neste capítulo!



| | |
|---|-----|
| Mighty Gumball quer um aplicativo web | 214 |
| Um pouco mais de informação sobre a Mighty Gumball | 216 |
| Então como fazemos requisições aos serviços web? | 219 |
| Como fazer uma requisição a partir do JavaScript | 220 |
| Deixando um pouco o XML, conhecendo JSON | 226 |
| Escrevendo uma função de handler onload | 229 |
| Mostrando os dados das vendas de chicletes | 230 |
| Como configurar seu próprio Servidor Web | 231 |
| Retrabalhando nosso código para fazer uso do JSON | 236 |
| Seguindo para o Servidor em Tempo Real | 237 |
| Que suspense! | 239 |
| Lembra que deixamos você com um suspense? Um bug. | 242 |
| O que é Política de Segurança do Browser? | 244 |
| Então, quais são nossas opções? | 247 |
| Conheça o JSONP | 252 |
| Mas o que é o “P” de JSON? | 253 |
| Vamos atualizar o aplicativo web Mighty Gumball | 256 |
| Passo 1: Dando um jeito no elemento script... | 264 |
| Passo 2: Agora é hora do timer | 265 |
| Passo 3: Reimplementando o JSONP | 267 |
| Quase esquecemos: cuidado com a terrível cache do browser | 272 |
| Como remover relatórios de vendas duplicados | 273 |
| Atualizando a URL JSON para incluir o lastreporttime | 275 |
| Pontos de Bala | 277 |

descobrindo seu artista interior



O Canvas

A HTML deixou de ser apenas uma linguagem

“marcação”. Com o novo elemento canvas da HTML5, você tem o poder de criar, manipular e destruir pixels, bem nas suas mãos. Neste capítulo, usaremos o elemento canvas para descobrir o artista que vive em você — chega de falar que o HTML é só semântico e sem apresentação; com o canvas, vamos pintar e desenhar com cores. Agora, é tudo uma questão de apresentação. Vamos aprender como posicionar um canvas em suas páginas, como desenhar texto e gráficos (usando JavaScript, é claro) e até mesmo como lidar com browsers que não suportam o elemento canvas. Canvas não é apenas uma maravilha que só serve para uma coisa: você verá muito mais em outros capítulos neste livro.

Um novo projeto HTML5
está esperando por você
para ser tirado do papel!



| | |
|---|-----|
| Nossa nova partida: TweetShirt | 282 |
| Verificando os “esboços” | 283 |
| Como pôr um canvas em sua página | 286 |
| Como ver seu canvas | 288 |
| Desenhando no Canvas | 290 |
| Fracassando graciosamente | 295 |
| TweetShirt: Visão Geral | 297 |
| Primeiro, vamos organizar o HTML | 300 |
| Agora, vamos adicionar o <form> | 301 |
| Hora de ficar computacional com JavaScript | 302 |
| Escrevendo a função drawSquare | 304 |
| Adicione a chamada para fillBackgroundColor | 307 |
| Enquanto isso, lá na TweetShirt.com... | 309 |
| Desenhando com Nerds | 311 |
| Desvendando o método arc | 314 |
| Provando um pouco do uso do arco | 316 |
| Eu digo grau, você diz radiano | 317 |
| Voltando a escrever o código dos círculos da TweetShirt | 318 |
| Escrevendo a função drawCircle... | 319 |
| Obtendo seus tuítes | 323 |
| Texto no Canvas de Perto | 328 |
| Vá testar o drawText | 330 |
| Completando a função drawText | 331 |
| Pontos de Bala | 338 |
| Exercício — Solução | 341 |



não é a TV de seu paí

8

Vídeo

Não precisamos de plugin. Afinal, vídeo é agora um membro de primeira-classe da família HTML — é só jogar um elemento <video> em sua página e o terá instantaneamente, mesmo em vários dispositivos. O vídeo, no entanto, é muito mais que apenas um elemento; é também uma API JavaScript que nos permite controlar a reprodução, criar nossas próprias interfaces de vídeo personalizadas e integrá-lo com o resto do HTML de maneiras completamente novas. Falando em integração... Lembre-se de que existe aquela conexão entre vídeo e canvas que falamos a respeito — você verá que unir vídeo e canvas nos dá uma poderosa nova forma de processar vídeo em tempo real. Neste capítulo, vamos começar por rodar vídeos numa página e, então, veremos o que a API JavaScript tem de melhor. Venha. Você ficará encantado ao ver o que é possível fazer com algumas tags, marcação, JavaScript e vídeo & canvas.



↑ Sintonize na TV Webville...

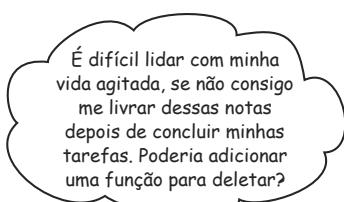
| | |
|--|-----|
| Conheça a TV Webville | 350 |
| O HTML, vamos botar pra quebrar... | 351 |
| Como funciona o elemento vídeo? | 353 |
| Inspecionando de perto os atributos do vídeo... | 354 |
| O que você precisa saber sobre formatos de vídeo | 356 |
| Como manejlar todos aqueles formatos... | 358 |
| Não me disseram que haveria APIs? | 363 |
| Chame esses Métodos | 363 |
| Um pouco de conteúdo de “programação” na TV Webville | 364 |
| Então, o que há com aquele código do event handler? | 367 |
| Como funciona o método canPlayType | 369 |
| Descompactando a Unidade Demo | 375 |
| Investigando o resto do código de origem | 376 |
| Os handlers setEffect e setVideo | 378 |
| Implementando seus controles de vídeo | 384 |
| Alternando vídeos de teste | 387 |
| Está na hora de alguns efeitos especiais | 389 |
| Como funciona o processamento de vídeo | 392 |
| Como processar o vídeo usando um scratch buffer | 393 |
| Implementando um scratch buffer com canvas | 395 |
| Agora precisamos escrever alguns efeitos | 399 |
| Como usar error events | 406 |
| Pontos de Bala | 408 |
| Exercício — Solução | 410 |

armazenando coisas localmente

9

Web Storage

Cansado de empurrar os dados de seu cliente para dentro de um pequeno armário cookie? Isso foi divertido nos anos 1990, mas temos necessidades muito maiores hoje em dia com os aplicativos web. E se disséssemos que poderíamos conseguir cinco megabytes para você no navegador do usuário? Provavelmente, olharia para nós como se estivéssemos tentando vender uma ponte para você no Brooklyn. Bem, não precisa ser cético — a API Web Storage HTML5 faz exatamente isso! Neste capítulo, vamos mostrar tudo que você precisa para armazenar qualquer objeto localmente, no dispositivo de seu usuário, e se aproveitar disso na experiência web.



| | |
|---|-----|
| Como funciona o armazenamento do navegador (1995 — 2010) | 414 |
| Como o Web Storage HTML5 funciona | 417 |
| Nota para si mesmo... | 418 |
| O Local Storage e o Array foram separados na maternidade? | 424 |
| Criando a interface | 429 |
| Agora, vamos adicionar JavaScript | 430 |
| Completando a interface do usuário | 431 |
| Precisamos parar para um reparo na agenda | 434 |
| Manutenção Faça-Você-Mesmo | 435 |
| Nós temos a tecnologia | 439 |
| Retrabalhando nosso aplicativo para usar um array | 440 |
| Convertendo createSticky para usar um array | 441 |
| Deletando sticky notes | 446 |
| A função deleteSticky | 449 |
| Como selecionar uma nota para ser deletada? | 450 |
| Como fazer a nota ser deletada a partir do evento | 451 |
| Delete a nota do DOM, também | 452 |
| Atualize a interface do usuário para que possamos especificar uma cor | 453 |
| JSON.stringify não é só para Arrays | 454 |
| Usando o novo stickyObj | 455 |
| X Não Tente Isto em Casa (ou Jogará pelos Ares seus 5 Megabytes) | 458 |
| Agora que você conhece o localStorage, como irá usá-lo? | 462 |
| Pontos de Bala | 464 |
| Exercício — Solução | 466 |

10

pondendo o JavaScript para funcionar

Web Workers (os Trabalhadores da Web)

Script lento — você quer continuar executando-o? Se você já gastou tempo suficiente com o JavaScript ou pesquisando na web, provavelmente já viu a mensagem “script lento” (slow script). Com todos esses processadores multi núcleo fazendo parte da sua máquina novinha, como pode um script estar rodando tão lentamente? Isso acontece porque o JavaScript só pode fazer uma coisa por vez. Contudo, com o HTML5 e os Web Workers, tudo isso muda. Você agora tem a habilidade de gerar seus próprios trabalhadores JavaScript para obter maior volume de trabalho feito. Se estiver apenas tentando desenhar um aplicativo mais responsivo, ou só quiser exigir o máximo que o CPU de sua máquina pode trabalhar, os Web Workers estão aqui para ajudar. Ponha seu chapéu de gerente JavaScript e vamos pôr alguns peões para trabalhar!

JavaScript Thread



| | |
|--|-----|
| O terrível script lento | 474 |
| Como o JavaScript gasta seu tempo | 474 |
| Quando o single-thread fica RUIM | 475 |
| Adicionando outro thread de controle para ajudar | 476 |
| Como trabalham os Web Workers | 478 |
| Seu primeiro Web Worker... | 483 |
| Escrevendo Manager.js | 484 |
| Recebendo mensagens do worker | 485 |
| Agora vamos escrever o worker | 486 |
| Expropriação Virtual | 494 |
| Como calcular um Mandelbrot Set | 496 |
| Como usar múltiplos workers | 497 |
| Vamos construir o aplicativo Fractal Explorer | 503 |
| Código Pronto para Assar | 504 |
| Criando workers e dando-lhes tarefas... | 508 |
| Escrevendo o código | 509 |
| Pondo os workers para começar | 510 |
| Implementando o worker | 511 |
| De volta ao código: como processar os resultados dos workers | 514 |
| Encaixando o canvas na janela do navegador | 517 |
| O codificador <code>chef</code> do Sr. Explicadinho | 518 |
| No Laboratório | 520 |
| Pontos de Bala | 524 |
| Exercício — Solução | 526 |

Apendice: Os dez tópicos mais importantes (que não vimos)

Vimos muita coisa e você já praticamente terminou este livro.

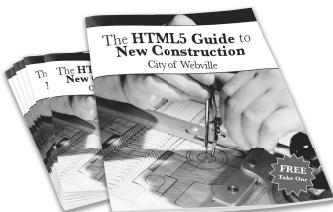
Sentiremos sua falta, mas, antes de deixarmos você ir, queremos lhe oferecer pelo mundo um pouco mais de preparação.

Claro que não há como ensinar tudo o que você precisa saber nesse, relativamente, pequeno capítulo.

Na verdade, nós, originalmente, incluímos tudo o que você precisava saber sobre HTML5, reduzindo o tamanho da fonte para .00004. Tudo caberia, mas ninguém conseguiria ler. Então, jogamos a maior parte fora e mantivemos o melhor para este apêndice dos Dez Mais.



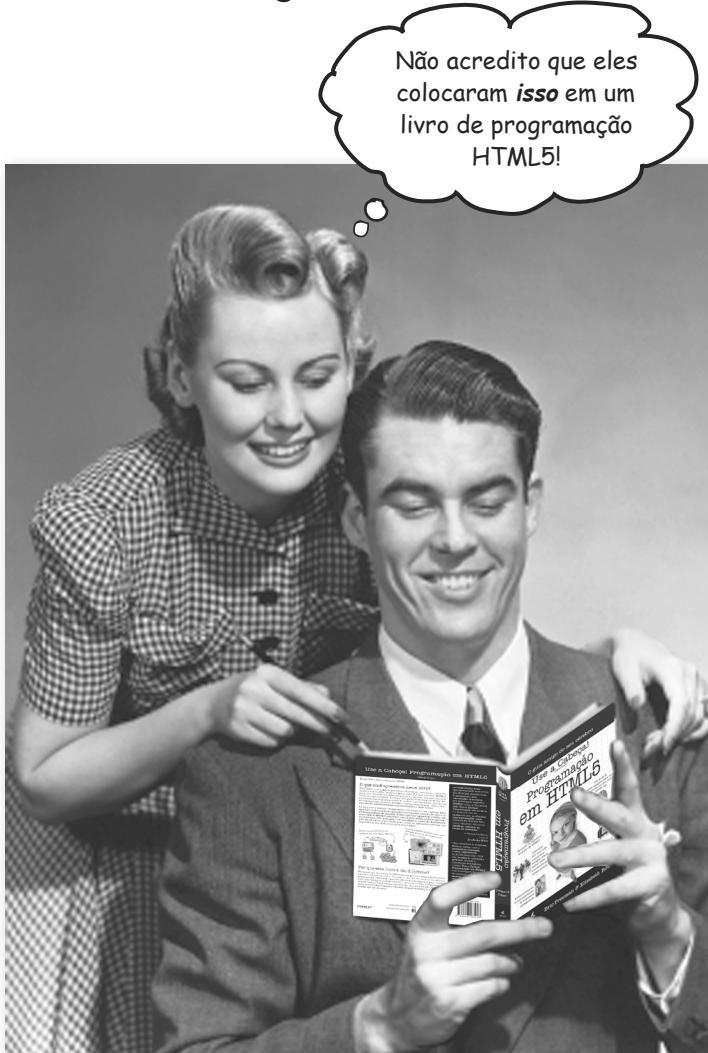
| | |
|---|------------|
| Nº 1 Modernizr | 532 |
| Nº 2 Áudio | 533 |
| Nº 3 jQuery | 534 |
| Nº 4 XHTML está morto, vida longa ao XHTML | 536 |
| Nº 5 SVG | 537 |
| Nº 6 Aplicativos web offline | 538 |
| Nº 7 Web Sockets | 539 |
| Nº 8 Mais API canvas | 540 |
| Nº 9 Seletores API | 542 |
| Nº 10 Tem muito mais! | 543 |
| O Guia HTML5 para Nova Construção | 545 |
| Guia Webville para Elementos Semânticos HTLM5 | 546 |
| Guia Webville para Propriedades CSS3 | 548 |
| Índice | 549 |



í Índice

como usar este livro

Introdução



Nesta seção, respondemos à questão premente:
"Então, por que eles colocaram ISSO em um livro sobre HTML5?"

A quem se destina este livro?

Se você responder “sim” a todas estas perguntas:

- ① Você tem um computador com um **navegador web** e um **editor de texto**?
- ② Você quer **aprender, entender, lembrar** e **criar** aplicativos web usando as melhores técnicas e os padrões mais recentes?
- ③ Você prefere uma **conversa estimulante em um jantar festivo** a **palestras áridas, desinteressantes e acadêmicas**?

este livro é para você.

Quem provavelmente deve fugir deste livro?

Se você responder “sim” a algumas destas perguntas:

- ① Você é **completamente novo** na escrita de páginas web?
- ② Você já está desenvolvendo aplicativos web e procurando um **livro de referência** sobre HTML5?
- ③ Você tem **medo de experimentar algo diferente**?
Você preferiria fazer um tratamento dentário de canal a misturar listras com xadrez? Você acredita que um livro técnico não pode ser sério se filmes educativos ruins da década de 1950 e APIs JavaScript antropomorfizadas estiverem nele?

Veja em *Use a Cabeça! HTML com CSS e XHTML*
uma excelente introdução ao
desenvolvimento web e depois
volte e junte-se a nós.



este livro não é para você.

[Observação do marketing: este
livro é para qualquer pessoa com um
cartão de crédito. Dinheiro vivo
também é bom — Dan]

Sabemos o que você está pensando.

“Como *este* pode ser um livro de programação HTML5 sério?”

“O que aconteceu com todos os gráficos?”

“Eu posso realmente *aprender* desta maneira?”

*Seu cérebro acha que
ISTO é importante.*

E sabemos o que o seu cérebro está pensando.

Seu cérebro precisa de novidades. Está sempre buscando, examinando, esperando por algo diferente. Ele foi criado assim e isso ajuda você a permanecer vivo.

Então, o que o seu cérebro faz com todas as coisas rotineiras, comuns e normais que você encontra? Tudo o que ele *puder* para impedir que elas interfiram na função *real* dele: gravar coisas que *importam*. Ele não se dá ao trabalho de gravar as coisas desinteressantes; elas nunca passam pelo filtro “isto obviamente não é importante”.

Como o seu cérebro *sabe* o que é importante? Suponha que você vá caminhar ao ar livre e um tigre pule na sua frente. O que acontece dentro da sua cabeça e corpo?



Os neurônios disparam. As emoções são acionadas. As substâncias químicas irrompem.

É assim que o seu cérebro sabe...

Isto deve ser importante! Não se esqueça!

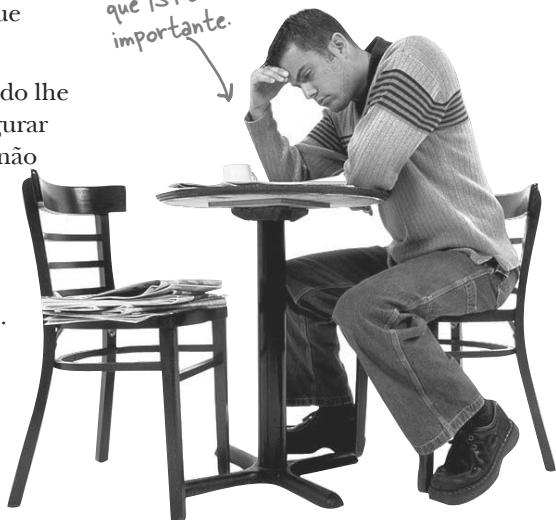
Agora, imagine que você esteja em casa ou em uma biblioteca. É uma região segura, aquecida e sem tigres. Você está estudando ou tentando aprender algum tópico técnico difícil, que o seu chefe acha que levará uma semana, dez dias, no máximo.

Apenas um problema. Seu cérebro está tentando lhe fazer um grande favor. Ele está tentando assegurar que este conteúdo *obviamente não* importante não atravance recursos escassos. Recursos que são melhor gastos armazenando as coisas realmente *importantes*. Assim como os tigres. Como o perigo do fogo. Como você nunca deve esquiar na neve de bermudas novamente.

Não há uma forma simples de dizer ao seu cérebro: “Ei, cérebro, muito obrigado, mas não importa o quanto chato este livro seja, e o quanto pouco estou marcando na escala Richter emocional agora, eu *realmente* quero que você grave esse assunto.”.

Ótimo. Só mais 640 páginas áridas, desinteressantes e chatas.

Seu cérebro acha que ISTO não é importante.



Entendemos que o leitor de um livro da série "Use a Cabeça!" é um aprendiz.

Então, o que é necessário para aprender alguma coisa? Em primeiro lugar, você tem que entender, depois se assegurar que não esquecerá. Não é empurrar fatos na sua cabeça. Baseado na pesquisa mais recente em ciência cognitiva, neurobiologia e psicologia educacional, a aprendizagem precisa de muito mais do que texto em uma página. Sabemos o que liga seu cérebro.

Alguns dos princípios de aprendizagem do Use a Cabeça!:



Faça ser visual. Imagens são muito mais recordáveis do que apenas palavras e tornam a aprendizagem muito mais efetiva (uma melhoria de até 89% em estudos de transferência e lembrança). Também torna as coisas mais compreensíveis.

Coloque as palavras dentro ou próximas de gráficos aos quais estejam relacionadas e os estudantes terão uma probabilidade **até duas vezes maior** de resolver problemas relacionados ao conteúdo.

Use um estilo coloquial e pessoal. Em estudos recentes, alunos tiveram um desempenho até 40% melhor em testes posteriores, se o conteúdo falasse diretamente ao leitor, em primeira pessoa e na forma coloquial, em vez de em tom formal. Conte histórias em vez de palestrar. Não se leve a sério demais. A que você prestaria mais atenção: uma

companhia interessante em um jantar festivo ou uma palestra?

Faça com que o aprendiz pense com mais profundidade.

Em outras palavras, a menos que você exerça ativamente seus neurônios, não acontece muita coisa na sua cabeça. Um leitor tem que ser motivado, incentivado, curioso e inspirado para resolver problemas, chegar a conclusões e gerar novo conhecimento. Para isso, você precisa de desafios, exercícios e questões que provoquem o raciocínio, além de atividades que envolvam ambos os lados do cérebro e múltiplos sentidos.



Obtenha — e mantenha — a atenção do leitor. Todos já tivemos a experiência "realmente quero aprender isso, mas não consigo permanecer acordado após a primeira página". Seu cérebro presta atenção a coisas que são fora do comum, interessantes, diferentes, chamativas, inesperadas. Aprender um novo tópico técnico difícil não tem que ser tedioso. Seu cérebro aprenderá muito mais rapidamente se não for.

Toque nas emoções deles. Agora sabemos que a sua capacidade de lembrar algo é muito dependente do conteúdo emocional. Você lembra daquilo a que dá importância. Você lembra quando sente algo. Não, não estamos falando de histórias tristes sobre um garoto e seu cachorro. Estamos falando de emoções como surpresa, curiosidade, diversão, "mas o que...?" e o sentimento de "Sou o máximo!" que vem quando você resolve um problema, aprende o que todas as outras pessoas acham que é difícil ou percebe que sabe algo que o Bob "sou mais técnico do que você" da engenharia não sabe.

Metacognição: pensando sobre pensar

Se você realmente quiser aprender, e quiser aprender mais rápido e profundamente, preste atenção em como você presta atenção. Pense em como você pensa.

Aprenda como você aprende.

A maioria de nós não fez cursos sobre metacognição ou teoria de aprendizagem quando estávamos crescendo. As pessoas *esperavam* que aprendêssemos, mas raramente nos *ensinavam* a aprender.

Supomos porém que, se você está segurando este livro, realmente quer aprender sobre desenvolvimento em HTML5, provavelmente não quer gastar muito tempo e, já que criará mais aplicativos no futuro, precisa *lembra*r do que lê. Para isso, tem que *entender*. Para tirar o maior proveito deste livro, ou de *qualquer* livro ou experiência de aprendizagem, responsabilize-se pelo seu cérebro. Seu cérebro e *este* conteúdo.

O truque é fazer com que seu cérebro veja o material novo que você está aprendendo como Realmente Importante. Crucial para o seu bem-estar. Tão importante quanto um tigre. Caso contrário, você ficará em uma batalha constante, com seu cérebro fazendo seu melhor para impedir que o novo conteúdo permaneça.

Então, como é que você faz para que seu cérebro ache que esse HTML5 (e JavaScript) é um tigre faminto?

Existe a forma lenta e tediosa, ou a mais rápida e eficaz. A forma lenta tem a ver com repetições. Você obviamente sabe que *pode* aprender e lembrar, mesmo os tópicos mais desinteressantes, se ficar repetindo a mesma coisa no seu cérebro. Com repetições suficientes, seu cérebro diz: “Isto não *parece* importante para ele, mas ele fica olhando a mesma coisa *várias vezes*, então acho que deve ser.”

A forma mais rápida é fazer *qualquer coisa que aumente a atividade do cérebro*, especialmente *tipos* diferentes de atividade cerebral. As coisas na página anterior são uma parte importante da solução, e são todas coisas que foram provadas como eficazes na ajuda para que seu cérebro trabalhe a seu favor. Por exemplo, estudos mostram que colocar palavras *dentro* das imagens que elas descrevem (em vez de em algum outro lugar na página, como um cabeçalho ou no texto do corpo) faz com que seu cérebro tente encontrar sentido no modo em que as palavras e a imagem se relacionam, e isto faz com que mais neurônios disparem. Mais neurônios disparando = mais chances do seu cérebro *entender* que isso é algo ao qual vale a pena prestar atenção, e possivelmente gravar.

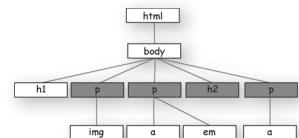
Um estilo coloquial ajuda porque as pessoas tendem a prestar mais atenção quando percebem que estão em uma conversa, já que devem seguir o assunto e esperar seu final. O incrível é que seu cérebro não se *importa* necessariamente que a “conversa” seja entre você e um livro! Por outro lado, se o estilo do texto for formal e árido, seu cérebro percebe da mesma maneira que quando você assiste a uma palestra, sentado em uma sala cheia de ouvintes passivos. Não é preciso permanecer acordado.

Imagens e estilo coloquial, no entanto, são apenas o início.



Aqui está o que NÓS fazemos:

Usamos *imagens*, porque seu cérebro é ligado em imagens, não em textos. No que diz respeito ao seu cérebro, uma figura realmente *vale* mil palavras. Quando texto e imagens trabalham juntos, inserimos o texto *na figura* porque seu cérebro funciona mais efetivamente quando o texto está *dentro* da coisa a qual ele se refere, diferentemente de em um título ou enterrado no texto em algum lugar.



Usamos *redundância*, dizendo a mesma coisa de formas *diferentes*, porque seu cérebro é ligado por novidades, e usamos imagens e ideias com pelo menos o *mesmo conteúdo emocional*, porque seu cérebro é ajustado para prestar atenção à bioquímica das emoções. Isso faz com que você *sinta* que algo tem maior probabilidade de ser lembrado, mesmo que esse sentimento não seja nada mais que um pouco de *humor, surpresa* ou *interesse*.



Sinta-se como
o Navegador

Usamos um *estilo coloquial* pessoal, porque seu cérebro é ajustado para prestar mais atenção quando acredita que você está em uma conversa do que se achar que você está ouvindo passivamente uma apresentação. Seu cérebro faz isso mesmo quando você está *lendo*.



PONTO DE BALA

Incluímos muitas *atividades*, porque seu cérebro é ajustado para aprender e lembrar mais quando você *faz* coisas do que quando *lê* sobre elas. Tornamos os exercícios desafiadores, mas viáveis, porque é o que a maioria das pessoas prefere.

Usamos *múltiplos estilos de aprendizagem*, porque você talvez prefira procedimentos passo a passo, ou entender o contexto geral primeiro, ou ainda só quer ver um exemplo. Independentemente, porém de sua preferência de aprendizagem, *todos* se beneficiam vendo o mesmo conteúdo representado de múltiplas formas.

Incluímos conteúdo para *os dois lados do seu cérebro*, porque, quanto mais do seu cérebro você envolve, maior a probabilidade de aprender e lembrar, e mais tempo consegue permanecer focado. Já que trabalhar um lado do cérebro muitas vezes significa dar ao outro lado uma chance de descansar, você pode ser mais produtivo aprendendo por um período mais longo.

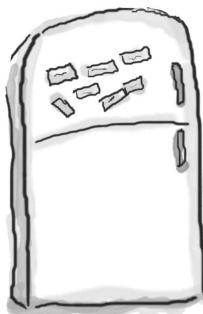


Incluímos *histórias* e exercícios que apresentam *mais de um ponto de vista*, porque seu cérebro é adaptado para aprender mais profundamente, quando é forçado a fazer avaliações e julgamentos.



Incluímos *desafios*, com exercícios, fazendo *perguntas* que nem sempre têm uma resposta direta, porque seu cérebro é adaptado para aprender e lembrar quando tem de *trabalhar* em algo. Pense nisso — você não consegue colocar seu *corpo* em forma apenas *observando* pessoas na ginástica. Fizemos o nosso melhor para assegurar, que quando você estiver trabalhando com afinco, esteja fazendo pelas coisas *certas*. E que você *não está gastando um dentrito extra* processando um exemplo difícil de entender, ou analisando um texto conciso demais, cheio de jargões ou difícil.

Usamos *pessoas*. Em histórias, exemplos, imagens etc, porque, bem, porque você é uma pessoa, e seu cérebro presta mais atenção a *pessoas* do que a *coisas*.



Corte isto e cole
na sua geladeira.

Veja o que fazer para que seu cérebro se curve em sinal de submissão

Então, fizemos a nossa parte. O resto é com você. Estas dicas são um ponto de partida; ouça seu cérebro e descubra o que funciona para você e o que não. Experimente coisas novas.

1 Vá devagar. Quanto mais você entende, menos tem de memorizar.

Não *leia* apenas. Pare e pense. Quando o livro lhe fizer uma pergunta, não pule para a resposta. Imagine que alguém realmente *esteja* lhe fazendo a pergunta. Quanto mais profundamente você forçar seu cérebro a pensar, maior a chance de aprender e lembrar.

2 Faça os exercícios. Faça suas próprias anotações.

Nós os colocamos, mas, se os fizéssemos por você, seria como se outra pessoa fizesse os seus exercícios físicos. Não *olhe* apenas os exercícios. Use um lápis. Há muita evidência de que atividade física *ao* aprender pode aumentar o aprendizado.

3 Leia as seções “Não existem perguntas idiotas”.

Isso significa todas elas. Não são quadros laterais opcionais — *são parte do conteúdo central!* Não os pule.

4 Que isso seja a última coisa que você leia antes de ir dormir. Ou pelo menos a última coisa desafiadora.

Parte do aprendizado (especialmente a transferência para a memória de longo prazo) acontece *após* você fechar o livro. Seu cérebro precisa de tempo sozinho, para processar mais. Se você colocar algo novo durante esse tempo de processamento, algo do que você acabou de aprender será perdido.

5 Beba água. Em grande quantidade.

Seu cérebro trabalha melhor com um belo banho de fluídios. A desidratação (que pode acontecer antes de você sentir sede) diminui a função cognitiva.

6 converse sobre o que está lendo. Em voz alta.

Falar ativa uma parte diferente do cérebro. Se você estiver tentando entender algo, ou aumentar sua chance de lembrar disso mais tarde, fale em voz alta. Melhor ainda, tente explicar isso em voz alta para outra pessoa. Você aprenderá mais rapidamente, e talvez descubra ideias que não sabia que estavam lá enquanto lidava com isso.

7 Ouça seu cérebro.

Preste atenção se o seu cérebro está sendo sobrecarregado. Se você estiver começando a apenas ler por cima e esquecer o que leu, é hora de um intervalo. Assim que você passar de um determinado ponto, não aprenderá mais rapidamente tentando ler mais, e pode até prejudicar o processo.

8 Sinta algo!

Seu cérebro precisa saber que isto é *importante*. Envolve-se com as histórias. Crie seus próprios títulos para as fotos. Suspirar por causa de uma piada ruim *ainda* é melhor do que não sentir nada.

9 Crie algo!

Aplique isto ao seu trabalho diário; use o que você está aprendendo para tomar decisões sobre seus projetos. Faça algo para obter alguma experiência além dos exercícios e atividades deste livro. Tudo o que você precisa é de um lápis e um problema para resolver... um problema que possa se beneficiar do uso de ferramentas e técnicas que você estiver usando para a prova.

Leia-me

Isto é uma prática de aprendizado, não um livro de referência. Nós tiramos deliberadamente tudo que poderia atrapalhar o aprendizado do que quer que estejamos trabalhando nesse ponto do livro. Na primeira vez, você precisa começar do início, porque o livro faz suposições sobre o que você já viu e aprendeu.

Esperamos que você conheça HTML e CSS.

Se você não conhecer marcação HTML (ou seja, tudo sobre documentos HTML, incluindo elementos, atributos, estrutura de propriedades, estrutura versus apresentação), então pegue o livro *Use a Cabeça! HTML com CSS e XHTML* antes de começar este livro. Em caso contrário, deve estar pronto para ler.

Alguma experiência ajuda, mas não esperamos que você conheça JavaScript.

Se você tiver alguma experiência em programação ou scripting (mesmo se não for com JavaScript), ela lhe ajudará. Não esperamos, porém, que você já conheça JavaScript; na verdade, este livro é projetado para seguir o *Use a Cabeça! HTML com CSS e XHTML*, que não tem scripting.

Nós o incentivamos a usar mais de um navegador neste livro.

Incentivamos você a testar as páginas e aplicativos web em diversos navegadores. Isto lhe dará a experiência de ver as diferenças entre navegadores e possibilitará a criação de páginas que funcionem bem em uma diversidade deles. Recomendamos enfaticamente o Google Chrome e o Apple Safari para uso com este livro, como são, de modo geral, os mais atualizados com os padrões correntes. Recomendamos, no entanto, que você também experimente as versões mais recentes dos outros principais navegadores, incluindo o Internet Explorer, Firefox e Opera, assim como navegadores móveis em dispositivos com iOS e Android.

As atividades NÃO são opcionais.

Os exercícios e atividades não são complementos; eles fazem parte do conteúdo central do livro. Alguns deles são para auxiliar a memória, alguns para entender e alguns lhe ajudarão a aplicar o que aprendeu. *Não pule os exercícios.* Até palavras cruzadas são importantes — elas lhe ajudarão a gravar os conceitos no seu cérebro. Mais importante, são boas para dar ao seu cérebro uma chance de pensar nas palavras e termos que você aprendeu em um contexto diferente.

A redundância é intencional e importante.

Uma diferença clara de um livro da série Use a Cabeça! é que queremos que você *realmente* grave. Queremos que você termine o livro lembrando do que aprendeu. A maioria dos livros de referência não tem retenção e lembrança como objetivo, mas este livro é sobre *aprender*, de modo que você verá alguns dos mesmos conceitos aparecerem mais de uma vez.

Os exercícios do Poder do Cérebro não têm respostas.

Para alguns deles, não há uma resposta certa e, para outros, parte da experiência de aprendizado das atividades do Poder do Cérebro é para você decidir se e quando suas respostas estão corretas. Em alguns desses exercícios, você encontrará dicas para lhe indicar o caminho certo.

Requisitos de software

Para escrever código em HTML5 e JavaScript, você precisa de um editor de textos, um navegador e, às vezes, um servidor web (pode estar hospedado localmente no seu desktop pessoal).

Os editores de textos que recomendamos para o Windows são o PSPad, TextPad ou EditPlus (mas você pode usar o Bloco de Notas, se precisar). Se estiver em um sistema Linux, tem muitos editores de textos internos, e acreditamos que você não precise que nós falemos sobre eles.

Esperamos que você tenha instalado pelo menos dois navegadores (veja a página anterior). Em caso contrário, faça isso agora. Também vale a pena gastar seu tempo aprendendo sobre como usar as ferramentas de desenvolvedor do navegador; cada um dos principais navegadores tem ferramentas internas, que você pode usar para inspecionar o console JavaScript (você pode ver erros, assim como mostrar resultados usando o `console.log`, uma alternativa útil a `alert`), uso do armazenamento web, o DOM, o estilo de CSS aplicado aos elementos e muito mais. Alguns navegadores têm até plugins para ferramentas adicionais de desenvolvedor. Você não precisa dessas ferramentas para o livro, mas, se desejar gastar o tempo investigando como usá-las, o desenvolvimento será mais fácil.

Alguns recursos HTML5 e APIs JavaScript requerem que você forneça arquivos a partir de um servidor web real em vez de carregando o arquivo (i.e., sua URL começará com `http://` em vez de `file://`). Identificamos em quais exemplos você precisará de um servidor nos lugares apropriados do livro, mas, se você estiver motivado, recomendamos que siga em frente e instale um servidor no seu computador agora. Para Mac e Linux, o Apache já está incluso, de modo que você só precisa assegurar-se de que sabe acessá-lo e sabe onde colocar seus arquivos para que possa fornecê-los usando seu servidor local. Para Windows, você precisará instalar o Apache ou o IIS; se você for usar o Apache, há muitas ferramentas *open source*, como o WAMP e XAMPP, que são razoavelmente fáceis de instalar.

É isso! Divirta-se...

Revisão Técnica

Paul Barry



Não apenas um revisor; Paul é um autor *Use a Cabeça!* experiente, tendo escrito *Use a Cabeça! Python* e *Use a Cabeça! Programação*.

Lou Barr



Tentamos dizer a ela que só precisaria nos ajudar com os gráficos, mas ela não pôde evitar e também foi uma revisora técnica eminente.

Nossos revisores:

Somos extremamente gratos à nossa revisão técnica. A equipe inteira provou o quanto precisávamos de seu conhecimento técnico e atenção a detalhes. **David Powers, Rebeca Dunn-Krhan, Trevor Farlow, Paul Berry, Louise Barr** e **Bert Bates** não deixaram nada sem exame na sua revisão e o livro é muito melhor por causa disso. Vocês são ótimos!

↓ David Powers



Nosso Revisor Técnico Master

↓ Bert Bates



Não é um simples revisor aqui, ele também é o criador da série! Cara, bota pressão nisso...

↓ Trevor Farlow



Nosso revisor dos 110% de esforço. Ele até correu no meio da noite de pijamas pra testar nosso código geográfico.

↓ Rebeca Dunn-Kahn



Rebeca atuou como segundo par de olhos; nos salvou a pele em detalhes de código que ninguém mais viu (inclusive nós!)

Agradecimentos

Mais revisores técnicos ainda:

Este está se tornando um tema recorrente nos nossos livros, mas queríamos dar mais uma reverência a **David Powers**, nosso estimado revisor técnico e autor de muitos livros, incluindo *PHP Solutions: Dynamic Web Developing Made Easy*. Os comentários de David sempre resultam em melhorias significativas no texto e dormimos melhor à noite sabendo que passou pelo David, então alcançamos a marca técnica. Obrigado novamente, David.

Nota para o Editor:
poderíamos prender
este cara para os
nossos próximos três
livros? Ver se podemos
torná-lo exclusivo.

Na O'Reilly:

Courtney Nash recebeu a difícil tarefa de gerenciar não apenas o livro *Use a Cabeça! Programação em HTML5*, mas também a *nós!* Courtney não apenas clareou todos os caminhos para nós, como também aplicou a pressão delicada que cada editor precisava para que o livro saísse. Mais importante que tudo, porém, Courtney forneceu feedback extremamente valioso sobre o livro e seu conteúdo, o que resultou em alguns trabalhos significativos no mesmo. Este livro é muito melhor devido ao esforço de Courtney. Obrigado.



Courtney Nash ↗



Lou Barr também foi parte essencial deste livro e contribuiu de muitas formas — de revisora, projetista de produção, projetista web, até a luta no Photoshop. Obrigado Lou, não poderíamos tê-lo feito sem você!

↖ Lou Barr, de novo! (E o Toby).

E obrigado a algumas outras pessoas que ajudaram a fazer isso acontecer:

Gostaria de agradecer o resto da equipe da O'Reilly pelo apoio de uma centena de formas diferentes. Essa equipe inclui **Mike Hendrickson, Mike Loukides, Laurel Ruma, Karen Shaner, Sanders Kleinfeld, Kristen Borg, Karen Montgomery, Rachel Monaghan, Julie Hawks e Nancy Reinhardt**.

Mais agradecimentos ainda!*

E obrigado a algumas outras pessoas:

James Henstridge escreveu o código original que se tornou o visualizador de fractais do Capítulo 10, o qual formatamos para nossos propósitos para uso no livro. Desculpe algum código que tenhamos introduzido e que possa não ter sido tão elegante quanto a sua versão original. O ator e artista **Laurence Zankowski**, eternamente estereotipado como CEO Starbuzz, reapareceu generosamente neste livro e auxiliou a testar o aplicativo de vídeo do Capítulo 8 (imperdível). A **Bainbridge Island Downtown Association** gentilmente nos permitiu usar seu excelente logotipo, projetado por Denise Harris, para o WickedlySmart Headquarters. Obrigado **Anthony Vizzari** e A&A Studios por nos permitir usar uma foto do seu fabuloso quiosque de fotos. Nossa exemplo inicial de TweetShirt usa alguns dos belos ícones do **Internet Archive**, no qual ficam os filmes que usamos para a Webville TV. E obrigado a **Daniel Steinberg** por estar sempre disponível para ajudar.

E, finalmente, obrigado Kathy e Bert

E, por fim, mas não menos importante, a **Kathy Sierra** e **Bert Bates**, nossos cúmplices e os CÉREBROS que criaram a série. Esperamos, mais uma vez, ter feito justiça a ela.



Ele está de
voooooooooooooalta!!!!



Pesquisando para o Use a
Cabeça! Método Parelli. ↗

*O grande número de agradecimentos é porque estamos testando a teoria de que todos os mencionados em um agradecimento de livro comprarão pelo menos uma cópia, provavelmente mais, considerando parentes e tudo mais. Se você gostaria de estar nos agradecimentos do nosso próximo livro, e tem uma família grande, escreva para nós.

1 conhecendo a HTML5

* **Bem-vindo a Webville** *



A HTML tem estado em uma corrida

louca. É claro, começou como uma simples linguagem de marcação, porém, mais recentemente, tem desenvolvido mais músculos. Agora, temos uma linguagem ajustada para criar verdadeiros aplicativos web com armazenamento local, desenho 2D, suporte offline, sockets, threads e mais. A história da HTML nem sempre foi bela e é cheia de dramas (falaremos sobre isso tudo), mas, neste capítulo, primeiro daremos uma pequena volta por Webville para ter uma ideia de tudo o que se relaciona com “HTML5”. Vamos lá, suba, estamos indo para Webville, e começaremos indo de zero a HTML5 em 3,8 páginas.

Atenção: A XHTML
recebeu uma carta de
despedida em 2009
e a visitaremos mais
tarde no segmento
“Onde eles estão agora”.



Atualize para HTML5 hoje!
Por que esperar?

Use o meu **HTML5-O-MÁTICO** e faça isso em
apenas **TRÊS ETAPAS SÍMPLES**

Vá agora mesmo! Por tempo limitado, pegaremos
aquele sua página HTML velha e molambenta e, em
apenas **TRÊS ETAPAS SÍMPLES**, a atualizaremos
para HTML5.

Pode mesmo ser assim tão fácil?

Pode apostar; na verdade, já temos uma demonstração
preparada para você.

Veja esta HTML cansada, gasta e que já teve dias
melhores; nós a transformaremos em HTML5 bem na
frente dos seus olhos:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
  <head>  
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">  
    <title>Head First Lounge</title>  
    <link type="text/css" rel="stylesheet" href="lounge.css">  
    <script type="text/javascript" src="lounge.js"></script>  
  </head>  
  <body>  
    <h1>Welcome to Head First Lounge</h1>  
    <p>  
        
    </p>  
    <p>  
      Join us any evening for refreshing <a href="elixirs.html">elixirs</a>,  
      conversation and maybe a game or two of Tap Tap Revolution.  
      Wireless access is always provided; BYOWS (Bring Your Own Web Server).  
    </p>  
  </body>  
</html>
```

Isto tudo é HTML 4.01 normal da
Head First Lounge, que você talvez
se lembre (se não lembrar, não se
preocupe, você não precisa lembrar).



Veja o quão fácil é escrever HTML5

Comece revisando este HTML, que está escrito em HTML 4.01 (a versão anterior a
HTML5). Examine com atenção cada linha e refresque sua memória sobre o que cada
parte faz. Sinta-se à vontade para fazer anotações na página. Veremos como passar
isso para HTML5 nas próximas páginas.



Aponte o seu lápis

Após examinar com cuidado a HTML da página 2, você consegue ver alguma marcação que possa mudar com HTML5? Ou que você gostaria que mudasse? Apontaremos uma para você: a definição doctype:

Este é o doctype para "html", estamos no lugar certo!

Isto só significa que este padrão está disponível publicamente.

Esta parte diz que estamos usando a versão 4.01 de HTML e que esta marcação está escrita em Inglês.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

Isto aponta para um arquivo que identifica este padrão.

Lembre-se de que a definição doctype deve ficar no topo do seu arquivo HTML e informa ao navegador o tipo do seu documento; neste caso, HTML 4.01. Usando um doctype, o navegador consegue ser mais preciso no modo como interpreta e renderiza suas páginas, portanto, ele é altamente recomendável.

Então, usando seus poderes de dedução, como você acha que a definição de doctype para HTML5 se parecerá? Escreva aqui (você pode voltar para ver sua resposta em um segundo após abordarmos esse assunto):

Sua resposta vai aqui. ↗

Apresentando o ~~HTML-o-mático~~, atualize sua HTML agora!



O Passo 1 te deixará impressionado: siga-nos, começaremos no topo do Head First Lounge HTML e atualizaremos o doctype para dar a ele esse novo brilho de HTML5.

Aqui está a versão antiga HTML 4.01 do doctype:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"  
      "http://www.w3.org/TR/html4/strict.dtd">
```

Agora, você talvez tenha imaginado que substituiremos cada menção de "4" por "5" no doctype, certo? Ah, não. Aqui está a parte incrível: o novo doctype para HTML5 é simplesmente:

```
<!doctype html>
```

Não há mais necessidade de pesquisar no Google para lembrar como é o doctype, ou copiar e colar de outro arquivo; este doctype é tão simples que você simplesmente conseguirá se lembrar dele.

Tem mais...

Este doctype não é apenas para HTML5; ele é para *todas as versões futuras* de HTML. Em outras palavras, nunca mais mudará. Além disso, ele funcionará em navegadores mais抗igos também.

Os caras dos W3C HTML Standards nos
asseguraram que desta vez será assim mesmo.



Se você for fã dos programas de TV *Extreme Makeovers* ou *The Biggest Loser*, irá gostar do Passo 2. Neste passo, temos a tag meta de conteúdo... Aqui, veja as imagens do antes e depois:

```
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
```

```
<meta charset="utf-8">
```

↑ ANTES (HTML 4)

DEPOIS (HTML5)

Sim, a nova tag meta ~~emagreceu~~ está muito mais simples. Quando você especificar a tag na HTML5, apenas faça isso junto com um caractere de codificação. Acredite ou não, todos os navegadores (antigos e novos) já entendem esta descrição meta, de modo que você pode usá-la em qualquer página e *funciona*.



Agora, vamos ao Passo 3, aquele que junta tudo. Aqui também iremos focar o elemento `<head>` e atualizar a tag `link`. Aqui está o que temos agora: um link do tipo `text/css` que aponta para uma planilha:

```
<link type="text/css" rel="stylesheet" href="lounge.css"> ↵
Modo antigo
```

Para atualizar isto para HTML5, só precisamos remover o atributo do tipo. Por quê? Porque CSS foi declarado como o padrão, e estilo default, para HTML5. Assim, após removermos o atributo de tipo, o novo link se parece com o seguinte:

```
<link rel="stylesheet" href="lounge.css"> ↵
HTML5
```



Por você ter agido rapidamente, temos um bônus especial. Tornaremos sua vida ainda mais fácil simplificando a tag de script. Com HTML5, JavaScript é agora a linguagem de script padrão e default, de modo que você pode remover o atributo de tipo das suas tags de script também. Aqui está como a nova tag de script fica sem o atributo `type`:

```
<script src="lounge.js"></script> ↵
Não se preocupe se você não
saber muito sobre a tag de
script ainda. Chegaremos lá...
```

Ou, se você tiver algum código inline, pode simplesmente escrever seu script desta forma:

```
<script>
  var youRock = true;
</script> ↵
Todo o seu código
JavaScript vai aqui. ↵
Falaremos mais
sobre JavaScript
em breve.
```



Parabéns, você agora está qualificado para atualizar qualquer HTML para HTML5!

Como um usuário HTML5-o-Mático treinado, você tem as ferramentas que precisa para pegar qualquer página HTML válida e atualizá-la para HTML5. Agora, está na hora de colocar sua certificação em prática!

Espere um momento. Todo esse rebuliço em torno de HTML5 e isso é tudo o que eu preciso fazer? Sobre o que é o resto do livro?

OK, OK, você nos pegou. Até aqui, temos falado sobre atualizar suas páginas HTML mais antigas, de modo que estejam prontas para aproveitar tudo que HTML5 tem a oferecer. Como você pode ver, se estiver familiarizado com HTML 4.01, então está em ótima forma, porque HTML5 é um superconjunto de HTML 4.01 (o que significa que praticamente tudo nela ainda é suportado em HTML5) e tudo o que você precisa é saber como especificar seu `doctype` e o resto das tags no elemento `<head>` para iniciar-se em HTML5.

Contudo, você está certo. Estábamos brincando. É claro que há mais em HTML5 do que apenas a atualização de alguns elementos. Na verdade, o que deixa todos entusiasmados é a capacidade de criar páginas interativas e ricas (ou até aplicativos web sofisticados) e, para suportar isso, o HTML5 fornece uma família inteira de tecnologias, que funcionam junto com a linguagem de marcação HTML5.

No entanto, espere um pouco; antes de chegarmos lá, temos um pouco mais de trabalho a fazer para assegurar que estamos prontos com nossa marcação.





Aponte o seu lápis

Você está mais próximo da marcação em HTML5 do que imagina!

Aqui está um pouco de HTML antiga que precisa de atualização. Trabalhe pelo processo HTML5-o-Mático e atualize esta HTML para HTML5. Siga em frente e rabisque no livro, faça um esboço sobre o código de marcação existente e adicione qualquer código de marcação novo que precisar. Ajudamos um pouco destacando as áreas que precisam de alteração.

Quando tiver terminado, digite (ou pegue os arquivos do exercício e faça suas alterações, se preferir), carregue no seu navegador, sente-se e aprecie seu primeiro HTML5. Ah, e você encontrará nossas respostas na próxima página.



Para baixar todo o código e arquivos de exemplo deste livro, por favor visite <http://wickedlysmart.com/hfhtml5>.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">

<html>
  <head>
    <title>Head First Lounge</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <link type="text/css" rel="stylesheet" href="lounge.css">
    <script type="text/javascript" src="lounge.js"></script>
  </head>
  <body>
    <h1>Welcome to Head First Lounge</h1>
    <p>
      
    </p>
    <p>
      Join us any evening for refreshing <a href="elixirs.html">elixirs</a>,
      conversation and maybe a game or two of Tap Tap Revolution.
      Wireless access is always provided; BYOWS (Bring Your Own Web Server).
    </p>
  </body>
</html>
```



Aponte o seu lápis

Solução Você está mais próximo da marcação em HTML5 do que imagina!

Aqui está um pouco de HTML antiga que precisa de atualização. Trabalhe pelo processo HTML5-o-Mático e atualize esta HTML para HTML5. Siga em frente e rabisque no livro, faça um esboço sobre o código de marcação existente e adicione qualquer código de marcação novo que precisar. Ajudamos um pouco destacando as áreas que precisam de alteração.

Aqui está a nossa solução.

Aqui está o código atualizado:

```
<!doctype html> ← O doctype
<html>
  <head>
    <title>Head First Lounge</title>
    <meta charset="utf-8"> ← ... a tag meta...
    <link rel="stylesheet" href="lounge.css"> ← ... a tag link...
    <script src="lounge.js"></script> ← ... e a tag de script...
  </head>
  <body>
    <h1>Welcome to Head First Lounge</h1>
    <p>
      
    </p>
    <p>
      Join us any evening for refreshing <a href="elixirs.html">elixirs</a>,
      conversation and maybe a game or two of Tap Tap Revolution.
      Wireless access is always provided; BYOWS (Bring Your Own Web Server).
    </p>
  </body>
</html>
```



Aqui estão quatro linhas que alteramos para tornar a nossa página HTML do Head First Lounge oficialmente HTML5.

Não acredita em nós?
Experimente <http://validator.w3.org/> e verá
– ela é validada como
HTML5. De verdade!

Perguntas Idiotas

P: Como isto funciona nos navegadores antigos? O novo doctype, meta e assim por diante... de alguma forma os navegadores mais抗igos trabalham com esta nova sintaxe?

R: Sim, através de um pouco de inteligência e sorte. Veja os atributos de tipo nas tags de link e script; faz sentido eliminar este atributo com HTML5 porque CSS e JavaScript agora são os padrões (e certamente são as tecnologias default para definição de estilo e scripting). Todavia, acabou que os navegadores já supunham os defaults de CSS e JavaScript. Assim, as estrelas se alinharam e o novo padrão de marcação já é suportado nos navegadores há anos. O mesmo é verdade sobre o doctype e a tag meta.

P: E o novo doctype? Ele parece simples demais agora; ele nem tem uma versão do DTD.

R: Sim, parece um pouco mágico que, após anos de uso de doctypes complexos, agora podemos simplificá-lo para "estamos usando HTML". O que houve foi o seguinte: HTML era baseada em um padrão chamado SGML, e esse padrão requeria a forma complexa do doctype e DTD. O novo padrão se afastou de SGML como uma forma de simplificar a linguagem HTML e torná-la mais flexível. Assim, não precisamos mais da forma complexa. Além disso, conforme dissemos anteriormente, houve um pouco de sorte no fato de que quase todos os navegadores só procuram HTML no doctype para assegurar que estão analisando um documento HTML.

P: Você estava brincando quanto a ela nunca mais mudar novamente? Eu achava que o uso de versões era muito importante para os navegadores. Por que não usar `<!doctype html5>`? Não significa que não haverá uma HTML6. Certo?

R: O uso do doctype evoluiu com os desenvolvedores de navegadores usando-o para fazer com que estes renderizassem as coisas no seu próprio "modo padrão". Agora que temos um padrão mais real, o doctype de HTML5 informa o navegador que este documento é HTML padrão, seja da versão 5, 6 ou qual for.

P: Bem, suponho que diferentes navegadores terão capacidades diferentes em um determinado momento. Como lido com isso?

R: É verdade, especialmente até que HTML5 seja suportada 100%. Examinaremos estas duas questões neste capítulo e por todo o livro.

P: Por que isto importa? Eu acabei de digitar uma página sem um doctype e tag meta e ela funcionou bem. Por que eu preciso me preocupar se isso está totalmente correto?

R: Os navegadores são ótimos ao desconsiderar pequenos erros em arquivos HTML, porém, incluindo o doctype e tags meta corretos, você assegurará que os navegadores saibam exatamente o que quer, em vez de terem de imaginar. Além disso, eles usarão o modo padrão, que é o que você quer. Lembre-se de que o modo padrão é aquele no qual o navegador supõe que você está escrevendo em HTML em conformidade com um padrão, então, usa essas regras para interpretar sua página. Se você não especificar um doctype, alguns navegadores podem entrar em um modo "estranho" e supor que a sua página web é escrita para navegadores mais抗igos, quando o padrão não era bem esse, interpretando-a de forma incorreta (ou supondo que tenha sido escrita incorretamente).

P: O que aconteceu com XHTML? Parece que há alguns anos era o futuro?

R: Sim, era, mas aí a flexibilidade venceu a sintaxe estrita e, no processo, XHTML (XHTML 2, para ser preciso) morreu e HTML5 nasceu para ser mais tolerante na forma pela qual as pessoas escrevem páginas web (e na forma pela qual os navegadores as exibem). Dito isso, não se preocupe, porque saber XHTML só irá lhe tornar um autor melhor de conteúdo HTML5 (e você apreciará HTML5 muito mais). A propósito, se você realmente gostar de XML, ainda há uma forma de escrever seu HTML5 da forma estrita. Falaremos mais sobre isso posteriormente...

P: O que é UTF-8?

R: UTF-8 é uma codificação de caracteres que possui suporte para muitos alfabetos, incluindo não ocidentais. Você provavelmente já viu outros conjuntos de caracteres sendo usados, mas UTF-8 está sendo promovido como o novo padrão. É bem mais curto e fácil de lembrar do que as codificações anteriores de caracteres.

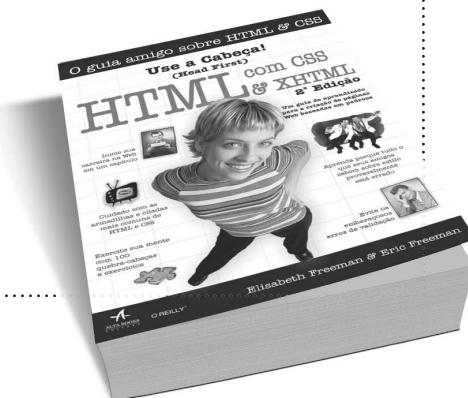


Não esperamos que você já conheça HTML5, ainda.

Se você nunca viu HTML5 antes, não tem problema, mas já deve ter trabalhado com HTML, e há alguns fundamentos que deve conhecer sobre elementos, tags, atributos, aninhamento, a diferença entre marcação semântica e adição de estilo, e assim por diante.

Se você não estiver familiarizado com tudo isso, faremos uma pequena sugestão (e uma propaganda sem-vergonha): há outro livro que precede este, *Use a Cabeça! HTML com CSS e XHTML - 2ª Edição*, e você deveria lê-lo. Se já estiver um pouco familiarizado com linguagens de marcação, talvez queira lê-lo por cima ou usá-lo como referência enquanto lê este livro.

Também incluímos um pequeno guia da marcação HTML5 e CSS no apêndice. Se você só quiser uma visão geral rápida sobre as novas adições, dê uma lida nelas no final do livro.





HTML5 Exposta

A entrevista desta semana:

Confissões da versão mais nova de HTML

Use a Cabeça: Bem-vinda, HTML5. Toda a web está falando sobre você. Para nós, você se parece muito com a HTML 4. Por que todos estão tão entusiasmados?

HTML5: Todos estão entusiasmados porque estou possibilitando uma nova geração de aplicativos e experiências web.

Use a Cabeça: Certo, mas por que HTML 4 ou até mesmo a promessa de “XHTML” não fizeram isso?

HTML5: XHTML 2 ficou em um beco sem saída. Todos que escreviam páginas web reais a detestaram. XHTML reinventou a forma pela qual escrevemos marcação para uma página web e teria tornado obsoletas todas as páginas que já estavam lá. Eu disse “Ei, espere um minuto, posso fazer coisas novas e adotar tudo o que já existe lá”. Quero dizer, se algo funciona, por que reinventar a roda? Esta é a minha filosofia.

Use a Cabeça: Parece estar funcionando, mas, você sabe, alguns dos caras da padronização ainda estão dizendo que a web ficaria melhor seguindo seus padrões “puros”.

HTML5: Sabe, eu não me importo. Eu ouço as pessoas por aí escrevendo páginas web reais — como elas estão me usando e como eu posso ajudar? Em segundo lugar, na minha lista estão os desenvolvedores criando os navegadores web. Em último lugar na minha lista, estão os caras da padronização. Eu os ouvirei, mas não se isso discordar do que os verdadeiros usuários estão fazendo.

Use a Cabeça: Por que não?

HTML5: Porque, se os usuários e os criadores de navegadores discordarem dos caras da padronização, este será um ponto irrelevante. Felizmente as pessoas que trabalham na especificação HTML5 concordam totalmente comigo, e esta é a nossa filosofia.

Use a Cabeça: Voltando à versão anterior de HTML, você disse que é um superconjunto de HTML 4.01. Isto significa que você tem compatibilidade retroativa, certo? Quer dizer que teremos que continuar lidando com todos os projetos ruins do passado?

HTML5: Prometo que farei o melhor possível para lidar com qualquer coisa do passado que seja jogada em mim. Isso não significa, porém, que esta seja a forma de me tratar. Eu quero que os autores de páginas web sejam treinados no padrão mais recente e me usem da melhor maneira possível. Desta forma, eles poderão realmente me levar até os meus limites. De qualquer forma, exibirei uma página antiga da melhor forma que puder, se não estiver atualizada.

Use a Cabeça: Minha próxima pergunta é ...

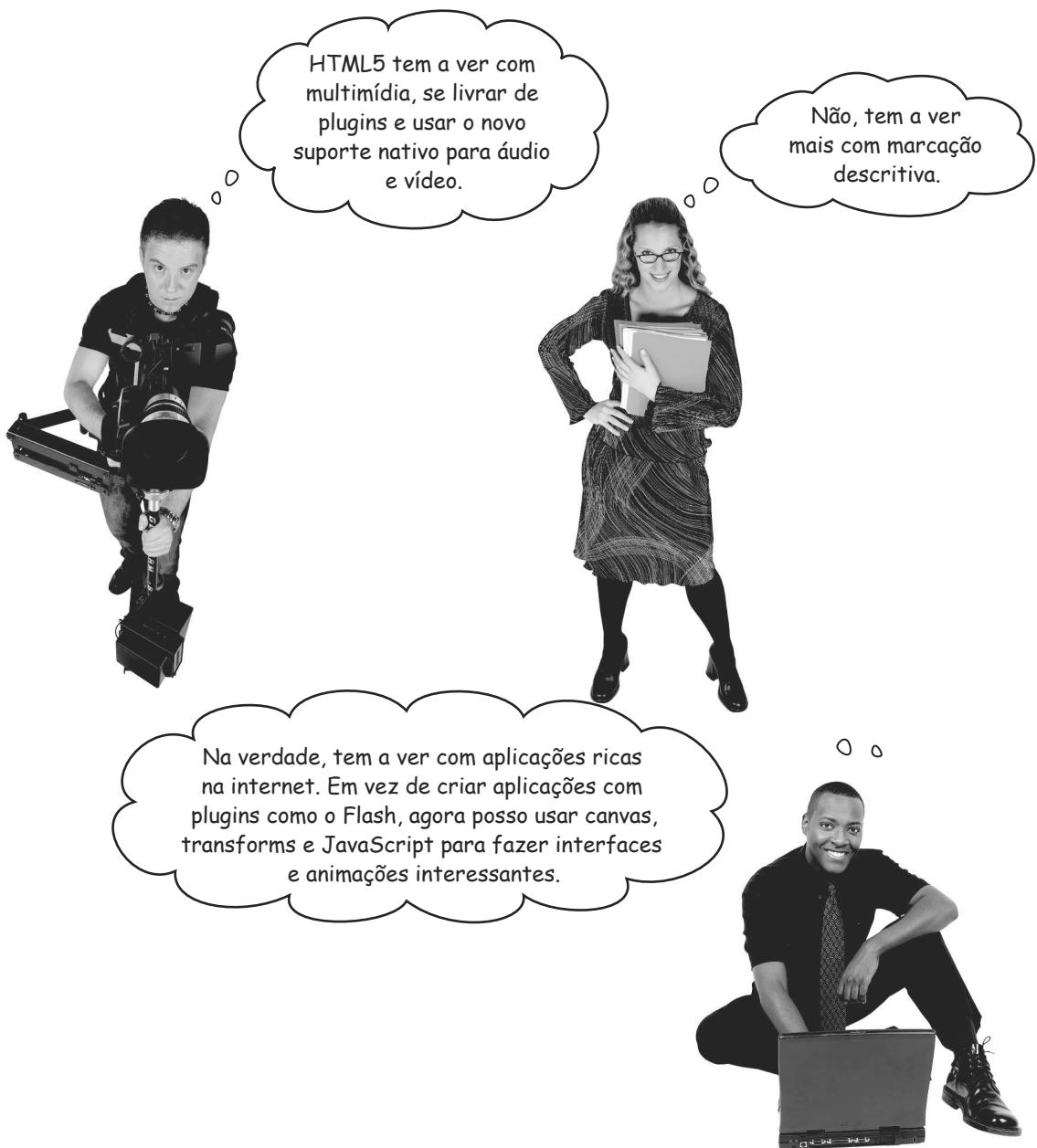
HTML5: Espere um pouco! Com todas essas perguntas sobre o meu passado, não estamos falando sobre o que é importante. No que diz respeito à minha marcação, minha missão pessoal é adotar a web como ela é, adicionar alguns elementos estruturados novos que tornem mais fácil a vida dos autores web e auxiliar todos os implementadores de navegadores a suportar semântica consistente em torno da minha marcação. Estou aqui realmente para mostrar minha nova proposta: aplicativos web...

Use a Cabeça: ... Desculpa, HTML5, só temos tempo para isto. Obrigado. Certamente falaremos sobre qualquer coisa que você queira na próxima entrevista.

HTML5: Argh, eu odeio quando isso acontece!!!

A VERDADEIRA HTML5, por favor, levante-se ...

Ok, você nos aguentou com humor na sátira do “HTML5-o-Mático” e temos certeza que já imaginou que há muito mais em HTML5 do que isso. O que se diz é que HTML5 remove a necessidade de plugins, pode ser usada para tudo, desde páginas simples até jogos do tipo Quake e é uma cobertura para sobremesas. HTML5 parece ser algo diferente para cada pessoa...





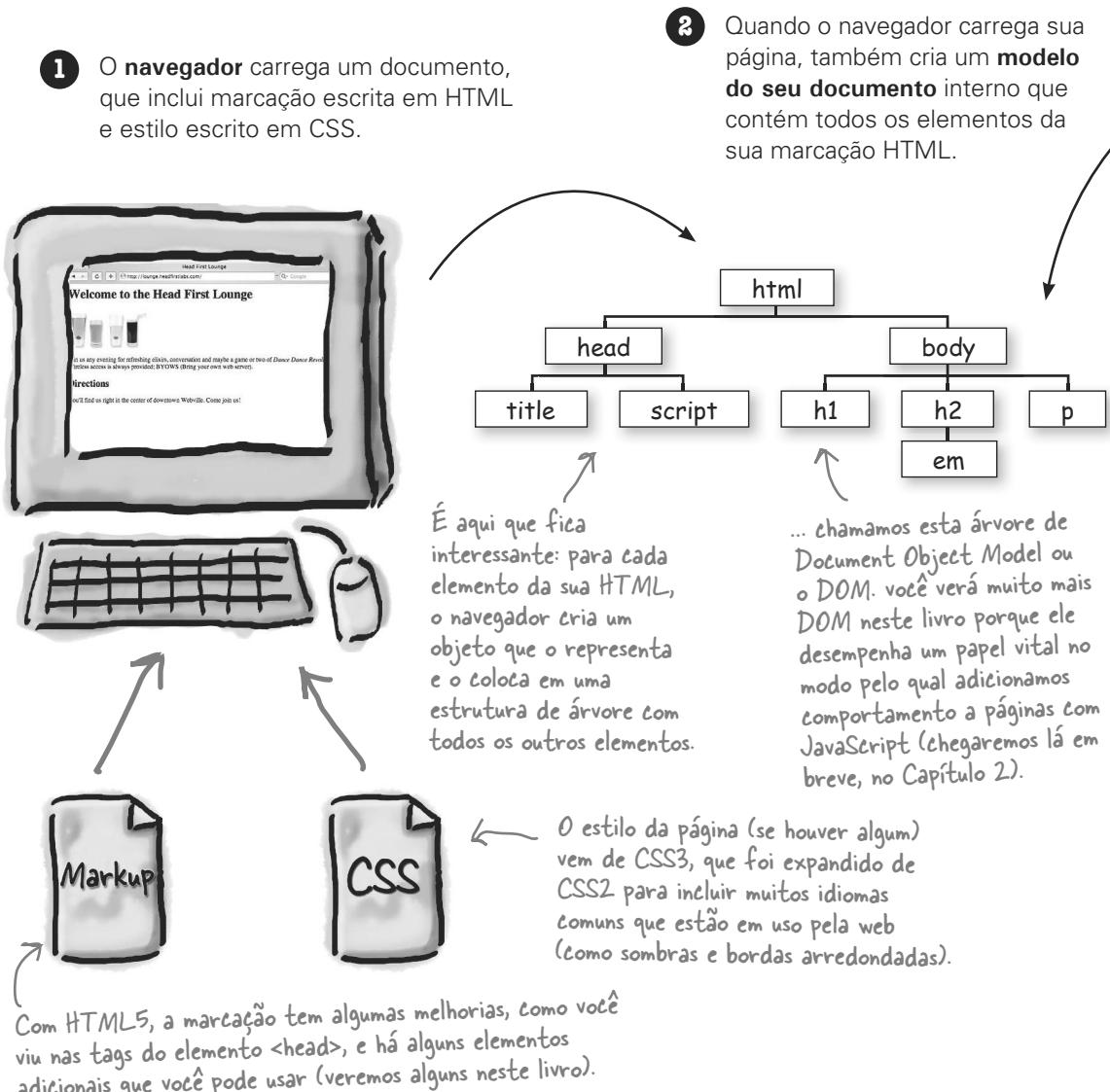
A boa notícia é que HTML5 é todas essas coisas. Quando as pessoas falam em HTML5, querem dizer uma *família de tecnologias* que, quando combinadas, lhe dão uma nova palheta para criar páginas e aplicativos web.

Como HTML5 realmente funciona...

Então nós dissemos que HTML5 é constituída por uma família de tecnologias, mas o que isso significa? Bem, você já sabe que existe a marcação HTML por si só, que se expandiu para incluir alguns elementos novos; também há muitos acréscimos a CSS e CSS3 que lhe dão mais poder ainda para definir o estilo das suas páginas. Depois, há o turbo charger: *JavaScript*, e um conjunto novo de APIs JavaScript que estão disponíveis para você.

Você encontra um belo guia Webville da nova marcação HTML5 e propriedades CSS3 no apêndice.

Vamos examinar o segundo plano e ver como tudo isso se junta:

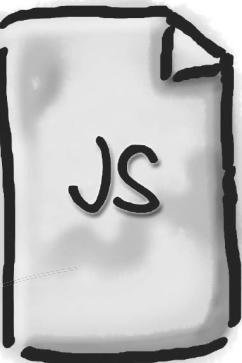




Nos Bastidores

- 3** Enquanto o navegador está carregando sua página, também está carregando seu **código JavaScript**, que geralmente começa a ser executado apenas depois que a página é carregada.

↑
Usando JavaScript, você pode interagir com sua página manipulando o DOM, reagir a eventos gerados pelo usuário ou pelo navegador, ou usar todas as suas novas APIs.



JavaScript interage com sua página através do DOM.

- 4** As **APIs** lhe dão acesso a áudio, vídeo, desenhos em 2D no canvas, armazenamento local e outras ótimas tecnologias necessárias para criar aplicativos. Lembre-se de que, para usar todas estas APIs, precisamos de JavaScript.

APIs, também conhecidas como Application Programming Interfaces, expõem um conjunto de objetos, métodos e propriedades que podemos usar para acessar toda a funcionalidade destas tecnologias. Examinaremos muitas dessas APIs neste livro.

Conheça as APIs JavaScript



QUEM FAZ O QUÉ?

Já falamos tanto sobre a “família de tecnologias” que achamos que elas são... bem, uma família. Ocorre que ainda não a conhecemos bem. Então, será que não está na hora? Você encontrará a maior parte da família abaixo. Vá em frente e socialize-se, veja se consegue descobrir quem é quem. Nós nos adiantamos e já descobrimos uma para você. *Não se preocupe. Sabemos que este é o seu primeiro encontro com os membros da família HTML5, então as respostas estão no final do capítulo.*

CSS3

Usando-me, você pode desenhar direto na sua página web. Comigo, você consegue desenhar texto, imagens, retas, círculos, retângulos, padrões e gradientes.

Web Workers

Você talvez já tenha me usado em HTML 4 para digitar informações, mas estou melhor ainda em HTML5. Posso requerer que você preencha todos os campos e posso verificar mais facilmente se você digitou um e-mail, URL ou número de telefone onde deveria.

Formulários

Você precisava de um plugin para nós, mas agora somos membros de primeira classe da família de elementos HTML. Quer assistir ou ouvir algo? Precisa de nós.

Aplicativos Web Offline

Estamos aqui para ajudar com a estrutura e significado semântico da sua página, incluindo novas formas de criar seções, cabeçalhos, rodapés e navegação.

Áudio e Vídeo

Sou o mais estiloso da família. Você provavelmente já me usou antes, mas sabia que agora posso animar seus elementos, dar a eles bordas arredondadas e até mesmo sombras?

Nova Marcação

Use-me como um pouco de armazenamento local no navegador de cada usuário. Precisa armazenar novas preferências, alguns itens de carrinhos de compras, ou talvez até mesmo um grande cache para aumentar a eficiência? Eu sou a sua API.

Armazenamento Local

Precisa de aplicativos que funcionem mesmo quando não estiver conectado à rede? Posso ajudar.

Canvas

Sou a API que pode lhe informar onde você está, e me saio bem com o Google Maps.

Geolocalização

Você irá me querer sempre que precisar que diversos scripts sejam executados concorrentemente e em segundo plano, de modo que sua interface de usuário continue responsiva.

SUA MISSÃO...

... se você aceitá-la, é fazer o reconhecimento de todos os navegadores HTML. Temos certeza de que você ouviu que alguns navegadores estão prontos para HTML5 e que alguns não estão. Precisamos que você veja bem de perto, porque a verdade está lá fora...



SUA PRIMEIRA MISSÃO: RECONHECIMENTO DE NAVEGADORES

SAIA E DESCUBRA [REDACTED] O NÍVEL ATUAL DE SUPORTE EM CADA NAVEGADOR ABAIXO (DICA: VÁ [REDACTED] AQUI PARA ENCONTRAR ALGUNS RECURSOS QUE ESTÃO ATUALIZADOS COM ESSAS COISAS: [HTTP://WWW.WICKEDLYSMART.COM/HFHTML5/BROWSERSUPPORT.HTML](http://www.wickedlysmart.com/hfhtml5/browsersupport.html), [REDACTED]). CONSIDERE A VERSÃO MAIS RECENTE DO NAVEGADOR. PARA CADA NAVEGADOR/RECURSO, COLOQUE UMA MARCA; SE FOR SUPORTADO, DÊ A ELE SUA PRÓPRIA NOTA SUBJETIVA SOBRE O QUANTO ELE SUPORTA HTML5 [REDACTED]. QUANDO RETORNAR, VENHA RAPIDAMENTE PEGAR SUA PRÓXIMA MISSÃO!

| Navegador \ Recurso | Vídeo | Áudio | Canvas | Armazenamento Web | Geolocalização | Web Workers | Aplicativos Web Offline |
|---------------------|-------|-------|--------|-------------------|----------------|-------------|-------------------------|
| Navegador | | | | | | | |
| Firefox | | | | | | | |
| Safari | | | | | | | |
| Chrome | | | | | | | |
| Mobile WebKit | | | | | | | |
| Opera | | | | | | | |
| IE 6, 7 | | | | | | | |
| IE 8 | | | | | | | |
| IE 9 | | | | | | | |

Dispositivos
iOS e
Android
(entre
outros)

SUA PRIMEIRA MISSÃO: RECONHECIMENTO DE NAVEGADORES SOLUÇÃO

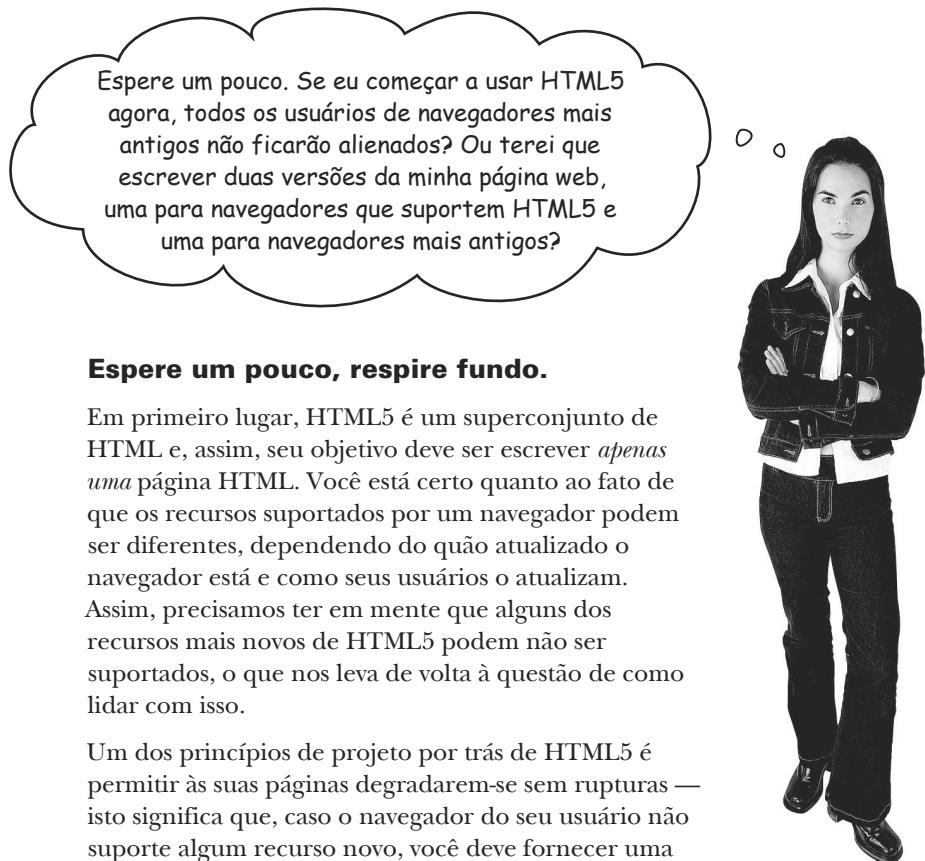
Trapaceámos nas nossas respostas e as preenchemos para 2015. As suas devem refletir o momento em que estiver lendo o livro, mas achamos que você gostaria de ver o futuro.

ARQUIVO DO CASO: HTML 5

| Navegador \ Recurso | Vídeo | Áudio | Canvas | Armazenamento Web | Geolocalização | Web Workers | Aplicativos Web Offline |
|---------------------|-------|-------|--------|-------------------|----------------|-------------|-------------------------|
| Navegador | | | | | | | |
| Firefox | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Safari | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Chrome | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Mobile Webkit | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Opera | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| IE 6, 7 | | | | ✓ | | | |
| IE 8 | | | | ✓ | | | |
| IE 9 | ✓ | ✓ | ✓ | ✓ | ✓ | | |

Embora demore um pouco até que o padrão esteja a pleno vapor, você usará navegadores que suportam integralmente HTML5 muito antes disso. Na verdade, em navegadores modernos, muitos recursos já são suportados. É por isso que é uma boa ideia começar a usar HTML5 agora. Além disso, começando agora, você poderá impressionar seus amigos e colegas de trabalho com todo o seu conhecimento de ponta.

→ E ganhar aquele aumento em breve!



Espere um pouco, respire fundo.

Em primeiro lugar, HTML5 é um superconjunto de HTML e, assim, seu objetivo deve ser escrever *apenas uma* página HTML. Você está certo quanto ao fato de que os recursos suportados por um navegador podem ser diferentes, dependendo do quanto atualizado o navegador está e como seus usuários o atualizam. Assim, precisamos ter em mente que alguns dos recursos mais novos de HTML5 podem não ser suportados, o que nos leva de volta à questão de como lidar com isso.

Um dos princípios de projeto por trás de HTML5 é permitir às suas páginas degradarem-se sem rupturas — isto significa que, caso o navegador do seu usuário não suporte algum recurso novo, você deve fornecer uma alternativa significativa. Neste livro, mostraremos a você como escrever suas páginas para fazer isso.

A boa notícia é que todos os navegadores estão indo na direção do padrão HTML5 e tecnologias relacionadas (até mesmo os navegadores de dispositivos móveis) e assim, com o tempo, a degradação sem rupturas será mais uma exceção do que a regra (embora você sempre vá querer fazer o que pode para dar aos seus usuários uma experiência significativa, não importa o navegador que estejam usando).

não existem Perguntas Idiotas

P: Ouvi que o padrão HTML5 não será uma recomendação final até 2022! É verdade? Se for, por que estamos nos importando?

R: O W3C é o corpo de padronização que recomenda formalmente o padrão HTML5, e o que você precisa saber sobre o W3C é que são conservadores, tão conservadores que prefeririam esperar até que algumas gerações de navegadores HTML5 tivessem surgido e ido embora antes de darem suas assinaturas. Não há problema: o padrão deve ser terminado nos próximos anos e os criadores de navegadores estão no caminho de implementá-lo. Desta forma, sim, pode demorar um pouco até que HTML5 seja uma “recomendação final”, mas deve ser um padrão estável em 2014, e para todos os propósitos práticos você deve iniciar-se agora em HTML5.

P: O que acontece após HTML5 ganhar sua versão final?

R: HTML6? Não temos ideia, mas talvez o que quer que seja virá com carros voadores, roupas com foguetes e jantar em uma pílula. Lembre-se de que, mesmo se adotarmos HTML6, o doctype não mudará. Supondo que o W3C mantenha sua promessa e versões futuras de HTML permaneçam compatíveis com versões mais antigas, estaremos em boa forma para o que quer que venha a seguir.

P: Chrome, Safari, Firefox e muitos navegadores de dispositivos móveis... o mundo não está ficando pior? Como asseguraremos que nossas páginas funcionem em todos esses navegadores?

R: Embora haja muita competição saudável no mercado de navegadores (desktop e dispositivos móveis), na verdade, muitos deles são baseados em alguns mecanismos HTML comuns. Por exemplo, o Chrome, o Safari e os navegadores de dispositivos móveis no Android e iPhone são todos baseados no WebKit, um mecanismo de navegador open source. Assim, na sua maioria, suas páginas funcionarão em múltiplos navegadores sem muito esforço.

P: Por que não simplesmente usar Flash para resolver questões de múltiplos navegadores?

R: Flash é uma ótima ferramenta para muitas aplicações e certamente no desktop é difundido em sistemas operacionais e navegadores. HTML5 e sua família de tecnologias estão tentando permitir que você faça com padrões abertos muitas das mesmas coisas que Flash pode fazer. Assim, que caminho você deveria tomar? Uma coisa para se pensar é a quantidade de investimento indo para tecnologias HTML5 dentro do Google, Apple, Microsoft e outros. A longo prazo, HTML5 será um player imenso, e no espaço móvel já é. Assim, embora a escolha seja sua, e temos certeza de que ambos vão existir por muito tempo, a indústria está indo na direção de padrões abertos.



Arqueologia HTML



Fizemos algumas escavações e descobrimos um código inserido em uma página HTML. Esperamos que você possa nos ajudar a entendê-lo para sabermos o que significa. Não se preocupe; não esperamos que você entenda este código. Estamos apenas tentando esquentar seu cérebro com um pouco de argumentação dedutiva...

```
<script>
    var walksLike = "duck"; ↴
    var soundsLike = document.getElementById("soundslike");
    if (walksLike == "dog") {
        soundsLike.innerHTML = "Woof! Woof!";
    } else if (walksLike == "duck") {
        soundsLike.innerHTML = "Quack, Quack";
    } else {
        soundsLike.innerHTML = "Crickets...";
    }
</script>
```

Uma dica: o documento representa a página HTML inteira, e getElementById provavelmente tenha a ver com ids e elementos HTML.

Estou dizendo, se você quiser mesmo criar aplicativos web usando HTML5, tem algo que você precisa saber sobre JavaScript...



Temos que conversar.

Se você está conosco desde o *Use a Cabeça! HTML com CSS e XHTML* (ou leu o livro até aqui sem pensar em usá-lo para acender a lareira), sabemos que provavelmente tenha uma boa compreensão do uso de linguagens de marcação e folhas de estilo para criar páginas web de ótima aparência. Conhecer essas duas tecnologias pode levar você longe...

Com HTML5, porém, as coisas estão mudando: páginas web estão se tornando experiências ricas (e aplicativos completos), que têm comportamento, são atualizadas durante a execução e interagem com o usuário. Criar estes tipos de páginas requer um pouco de programação e, se você for escrever código para o navegador, há apenas um jogo na cidade: *JavaScript*.

Agora, se você já programou ou escreveu scripts simples, estará bem: JavaScript (apesar de alguns rumores) é uma linguagem fantástica e lhe guiaremos por tudo o que você precisa saber para escrever os aplicativos deste livro. Se você não tiver programado antes, faremos tudo que pudermos para lhe ajudar pelo caminho. Em qualquer caso, um dos grandes benefícios de JavaScript é o quanto acessível ela é para programadores novatos.

Então, o que faremos agora? Vamos apresentar rapidamente um pouco de JavaScript e depois mergulharemos realmente fundo no Capítulo 2. Na verdade, por enquanto, não se preocupe demais com cada detalhe — só queremos que você sinta um *gostinho de JavaScript*.

Nós não podemos pensar em uma melhor ou mais divertida maneira de aprender a programar!

O que você pode fazer com JavaScript?

JavaScript abre um novo universo inteiro de expressão e funcionalidade para suas páginas web. Vejamos apenas algumas coisas que você pode querer fazer com JavaScript e HTML5...

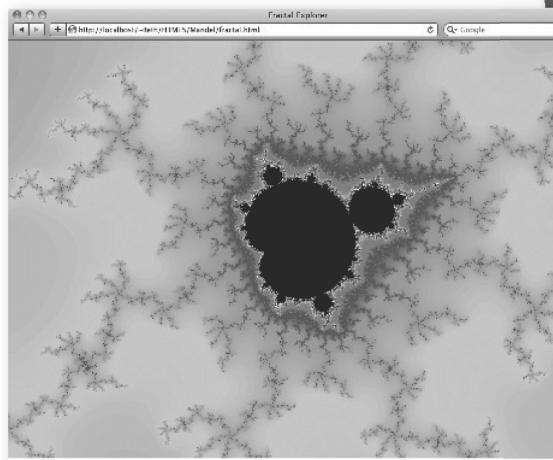
Com HTML5 e JavaScript, você pode criar uma superfície 2D desenhável na sua página, sem necessidade de plugins.

Faça com que suas páginas percebam a localização de seus usuários para mostrar a eles o que está próximo, levá-los a uma caça ao tesouro, dar-lhes coordenadas ou juntar pessoas com interesses comuns na mesma área.



Coloque dados em cache localmente usando armazenamento no navegador para acelerar aplicativos móveis.

Integre suas páginas com o Google Maps e até deixe seus usuários registrarem sua movimentação em tempo real.

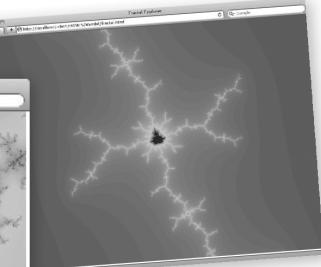


Use web workers para turbinar seu código JavaScript e executar algumas computações complexas ou tornar seu aplicativo mais responsivo. Você pode até fazer um uso melhor do processador multicore do seu usuário!

Acesse qualquer serviço web e traga os dados para seu aplicativo, quase em tempo real.

Não há necessidade de plugins para executar vídeos.

Crie seus próprios controles de execução de vídeo usando HTML e JavaScript.



Diga adeus aos cookies de navegadores e use o armazenamento local com base no navegador.

Usando JavaScript, você pode armazenar localmente muitas preferências e dados para seus usuários, no navegador, e até disponibilizá-los para acesso offline.

O navegador claramente não é mais apenas para documentos tediosos. Com JavaScript, você pode desenhar pixels diretamente nele.

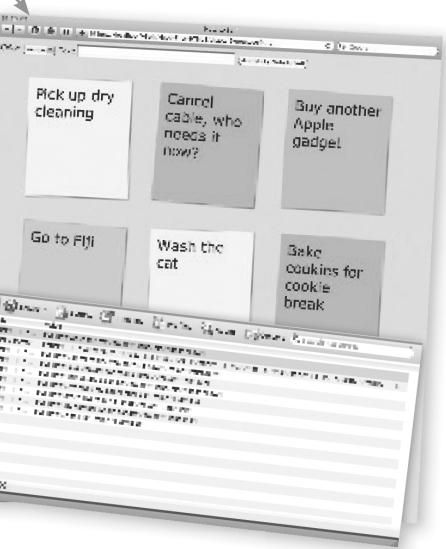
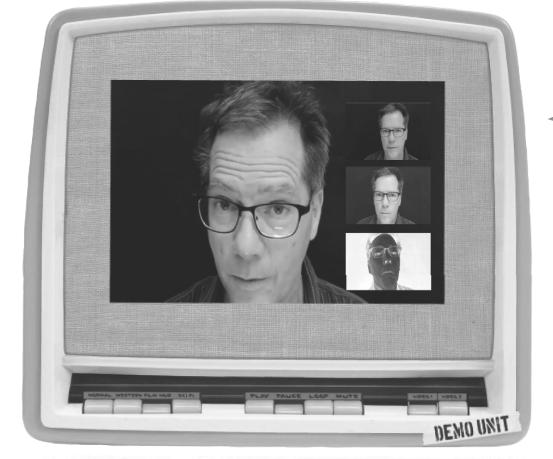


Turbe seus formulários com JavaScript para fornecer interatividade real.



Construa experiências em vídeo completas que incorporem vídeos de modos novos.

Use o poder de JavaScript para criar processamento completo de vídeo no seu navegador. Crie efeitos especiais e até manipule diretamente pixels de vídeo.



Você provavelmente deve estar pensando que pesquisamos a web exaustivamente para descobrir os exemplos mais interessantes que podíamos, certo? Não. Tudo o que fizemos foi pegar as capturas de telas dos exemplos do resto deste livro. Não é interessante? Então agora que você está em Webville, está na hora de aprender o jargão local: JavaScript. Vamos lá começar.



JavaScript Exposto

A entrevista desta semana:

Confissões de uma Linguagem de Scripting

Use a Cabeça: Bem-vinda, JavaScript. Ficamos felizes por você ter conseguido nos inserir na sua agenda ocupada. Deixe-nos fazer uma colocação: HTML5 está se tornando uma celebridade — o que você acha disso?

JavaScript: Não sou alguém que busca o estrelato, sou um tipo de pessoa que gosta de ficar em segundo plano. Ou seja, muito do crédito que vai para o HTML5 deveria ser meu.

Use a Cabeça: Por que você diz isso?

JavaScript: Há uma família inteira de tecnologias que fazem o trabalho “HTML5”, como o canvas 2D, armazenamento local, web workers, esse tipo de coisa. A verdade é que eu, JavaScript, sou necessária para fazer uso delas. Certamente HTML5 lhe dá um local para juntar e apresentar tudo, porém, sem mim, você não teria uma experiência interessante. Está tudo bem, mas poder para o HTML5, vou continuar simplesmente fazendo meu trabalho.

Use a Cabeça: Qual o seu conselho para os novos autores de HTML5?

JavaScript: É fácil. Se você realmente quiser dominar HTML5, gaste seu tempo em JavaScript e em todas as bibliotecas que trabalham com ela.

Use a Cabeça: Sabe, você nem sempre teve a melhor reputação. Citarei uma resenha de 1998: “JavaScript é no máximo uma linguagem de scripting fraca e feita pela metade.”

JavaScript: Isso magoa. Posso não ter começado a vida no ambiente claro e acadêmico de muitas linguagens de programação, mas me tornei uma das mais amplamente usadas de todos os tempos, de modo que não me diminuiria tão rapidamente. Não apenas isso, mas enormes quantidades de recursos têm sido colocados para me tornar robusta e extremamente rápida. Estou pelo menos 100 vezes mais rápida do que há uma década.

Use a Cabeça: Isto é impressionante.

JavaScript: Ah, e caso você não tenha ouvido, o pessoal da padronização acabou de me contar que agora sou a linguagem de scripting padrão para HTML5. Assim, estou aqui para ficar. Na verdade, você nem tem que dizer mais “JavaScript” na sua tag <script>. Então, eles podem ter me chamado de fraca em 1998, mas onde estão agora jScript, VBScript, Java Applets e todas as tentativas fracassadas de linguagens de navegadores?

Use a Cabeça: Bom, certamente parece que você é a chave para a criação de ótimas experiências em HTML5. Você tem a reputação de ser uma linguagem confusa.

JavaScript: Sou uma linguagem muito poderosa, apesar de alguns rumores, de modo que você realmente deve gastar algum tempo aprendendo a me usar bem. Por outro lado, sou popular porque sou muito fácil de começar a usar. O melhor de ambos os mundos, não acha?

Use a Cabeça: Parece ser isso mesmo! Obrigado, JavaScript, por ter vindo aqui hoje.

JavaScript: O prazer foi meu, estou às ordens.

Escrevendo JavaScript Seriamente

Com toda essa conversa sobre JavaScript, apostamos que você está pronto para ver do que se trata. Eles não chamam isto *Use a Cabeça* sem motivo. Temos um aplicativo de negócio super sério abaixo, que mostraremos para você. Por enquanto, comece examinando o código para ter uma ideia sobre ele. Escreva o que você acha que cada linha faz. Não se preocupe, não esperamos que você já entenda tudo, mas apostamos que você consegue fazer algumas suposições realmente boas sobre o que o código faz. Quando tiver terminado, vire a página e veja o quanto perto você ficou ...

```
var drink = "Energy Drink";           Substitua pelo nome da sua bebida favorita.
var lyrics = "";
var cans = 99;

while (cans > 0) {
    lyrics = lyrics + cans + " cans of "
        + drink + " on the wall <br>";
    lyrics = lyrics + cans + " cans of "
        + drink + "<br>";
    lyrics = lyrics + "Take one down, pass it around,<br>";

    if (cans > 1) {
        lyrics = lyrics + (cans-1) + " cans of "
            + drink + " on the wall <br>";
    }
    else {
        lyrics = lyrics + "No more cans of "
            + drink + " on the wall <br>";
    }
    cans = cans - 1;
}

document.write(lyrics);
```



 Escreva suas
respostas aqui.

Escrevendo JavaScript Seriamente Revisto...

Passe pelo código novamente e veja se você acertou. Neste momento, você deve ter uma ideia sobre o código; passaremos por tudo detalhadamente em breve.

```
var drink = "Energy Drink";  
  
var lyrics = "";  
  
var cans = 99;  
  
while (cans > 0) {  
  
    lyrics = lyrics + cans + " cans of "  
        + drink + " on the wall <br>";  
  
    lyrics = lyrics + cans + " cans of "  
        + drink + "<br>";  
  
    lyrics = lyrics + "Take one down, pass it around,<br>";  
  
    if (cans > 1) {  
  
        lyrics = lyrics + (cans-1) + " cans of "  
            + drink + " on the wall <br>";  
    }  
    else {  
  
        lyrics = lyrics + "No more cans of "  
            + drink + " on the wall <br>";  
    }  
    cans = cans - 1;  
  
}  
  
document.write(lyrics);
```



Declare uma variável e atribua a ela um valor de "Energy Drink".

Declare outra variável e atribua a ela um valor de string vazia.

Declare outra variável e atribua a ela um valor numérico, 99.

Este é um loop while. Ele diz que, enquanto o número de latas for maior que 0, faça tudo entre as chaves. Pare quando não houver latas sobrando.

Adicione o próximo verso da música à variável lyrics, usando o operador de concatenação de strings "+".

Termine a linha com uma quebra de linha HTML.

Faça isso novamente — afinal é assim que uma música é, certo?

Adicione o próximo verso, usando concatenação novamente.

Se ainda houver uma lata sobrando (ou seja, o valor de latas é maior que 1)...

... adicione a última linha.

caso contrário, não há mais latas sobrando...

... então, adicione "Não há mais latas" no final da letra da música.

Reduza para 1 o número de latas

Armazenamos todos os versos para a música na variável lyrics, então agora fazemos com que a página web a escreva, o que significa apenas que a string é adicionada à página de forma que você possa ver a música.



Você não achou que faria todo esse trabalho pesado no exercício sem experimentar JavaScript, pensou? O que você precisará fazer é pegar o código da página anterior e digitá-lo (junto com o código HTML abaixo) em um arquivo (como index.html) e então carregá-lo no seu navegador. Você verá nossos resultados abaixo:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My First JavaScript</title>
  </head>
  <body>
    <script>
      </script>
    </body>
</html>
```

Digite isto.

Lembre-se de que para baixar o código e arquivos de exemplo deste livro, por favor visite <http://wickedlysmart.com/hfhtml5>.

As tags `<script>` e `</script>` envolvem o código JavaScript. Elas informam à página que o que há nelas é JavaScript, não HTML.

E digite o código JavaScript da página anterior aqui.

Aqui está a execução de teste deste código. O código cria a letra inteira das 99 garrafas latas de cerveja energética no muro e escreve o texto no documento do navegador.



Pnão existemerguntas Idiotas

P: Por que não havia nada no corpo desse código HTML exceto o script?

R: Decidimos começar com um corpo vazio porque criamos todo o conteúdo desta página usando código em JavaScript, mas, certamente, poderíamos ter apenas digitado a letra da música diretamente no elemento do corpo (e seria muita digitação) ou poderíamos fazer o código executar todo o trabalho árduo por nós (o que fizemos) e depois fazer o código inserir a letra na página com `document.write`.

Tenha em mente que estamos apenas iniciando aqui; gastaremos muito mais tempo neste livro vendo como podemos pegar uma página e preenchê-la dinamicamente no seu conteúdo com código.

P: Entendo que construímos a letra inteira da música, mas exatamente o que o `document.write` fez e como o texto apareceu no documento?

R: Bom, `document.write` recebe uma string de texto e a insere no documento; na verdade, ele coloca a string exatamente onde a tag de script estiver localizada. Assim, neste caso, `document.write` coloca a string direto no corpo da página.

Você verá em breve formas mais sofisticadas de alterar o texto de um documento ao vivo com JavaScript, mas este exemplo deve lhe dar uma ideia de como o código pode alterar dinamicamente uma página.

P: Você tem usado os termos página web e aplicativo web; eles são duas coisas diferentes? O que torna algo um aplicativo web?

R: Esta é uma ótima pergunta, porque estamos usando os termos livremente. Não há diferença técnica entre os dois; em outras palavras, não há nada especial que você faça para transformar uma página escrita em HTML, JavaScript e/ou CSS em um aplicativo web. A distinção é mais de perspectiva.

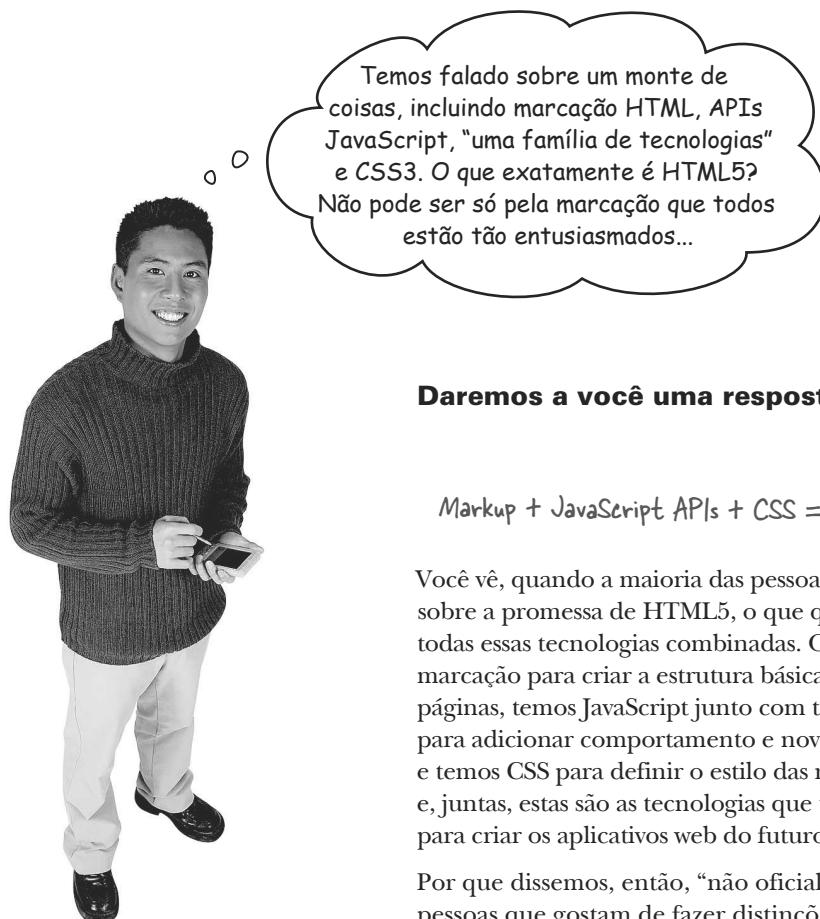
Quando temos uma página que está agindo mais como um aplicativo do que apenas um documento estático, então começamos a pensar nela como um aplicativo web e menos como uma página web. Pensamos em aplicativos como tendo um número de qualidades, como armazenando muitos estados, gerenciando interações complexas com o usuário, exibindo dados atualizados dinâmica e constantemente sem uma atualização da página, ou até mesmo executando tarefas ou cálculos mais complexos.

P: Ei, todo esse JavaScript é ótimo, mas e CSS?

Estou com vontade de aproveitar algumas das novidades de CSS3 sobre as quais tenho ouvido para melhorar a aparência das minhas páginas.

R: Sim, CSS percorreu um longo caminho e estamos impressionados por funcionar tão bem com HTML5. Embora este livro não seja sobre CSS3, você pode ter certeza de que aproveitaremos integralmente alguns dos seus novos recursos. Conforme você talvez já saiba, muitos dos truques que costumávamos fazer para adicionar bordas arredondadas, sombras com imagens em HTML e animação simples com JavaScript, agora podem ser feitos facilmente com CSS3.

Então, sim, faremos uso do poder de CSS3 neste livro, e mostraremos quando estivermos fazendo isso.

**Daremos a você uma resposta não oficial:**

HTML5
↑
Markup + JavaScript APIs + CSS = Crazy Delicious

Você vê, quando a maioria das pessoas está falando sobre a promessa de HTML5, o que querem dizer são todas essas tecnologias combinadas. Ou seja, temos marcação para criar a estrutura básica das nossas páginas, temos JavaScript junto com todas as suas APIs para adicionar comportamento e nova funcionalidade, e temos CSS para definir o estilo das nossas páginas — e, juntas, estas são as tecnologias que todos usaremos para criar os aplicativos web do futuro.

Por que dissemos, então, “não oficial”? Bom, há pessoas que gostam de fazer distinções estritas entre estas tecnologias e a quais padrões elas pertencem. Isso é bom e tem sua utilidade. Entretanto, o que nos importa é o seguinte: quais tecnologias estão disponíveis no navegador e elas estão prontas para usarmos na criação de nossas páginas e aplicativos? Então, dizemos que HTML5 é marcação + APIs JavaScript + CSS e achamos que é o que a maioria das pessoas geralmente quer dizer quando fala sobre HTML5 como uma tecnologia.

↑ Se você estiver realmente interessado em como essas tecnologias se ajustam como um conjunto de padrões (e todos devemos estar), então nós o incentivamos a visitar w3.org para obter mais informações.

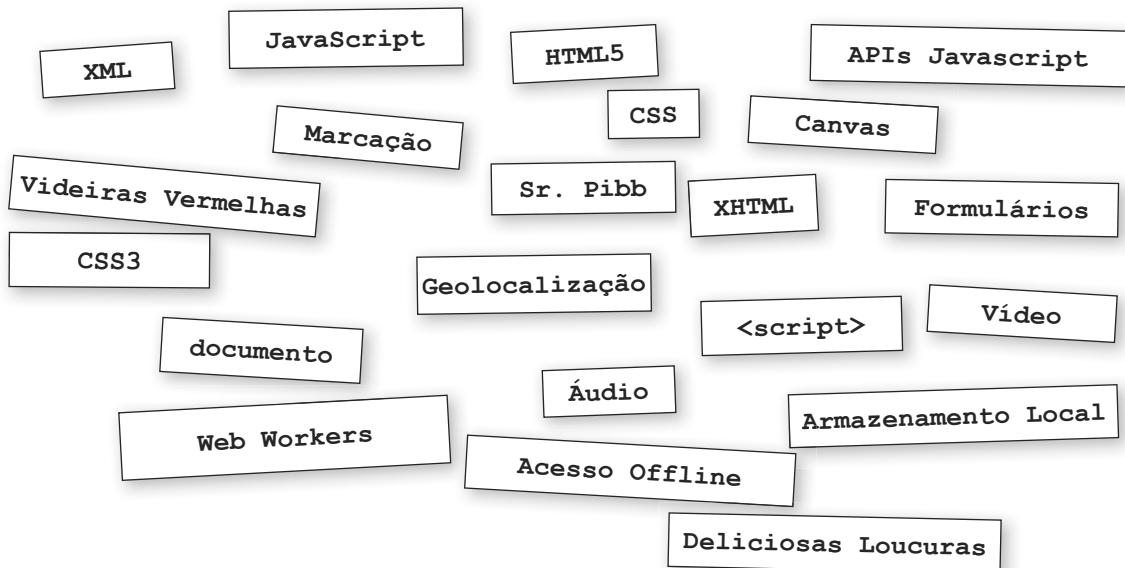


Parabéns, você terminou o Capítulo 1 e escreveu seu primeiro HTML5!

Antes que você corra para o próximo capítulo, temos mais uma tarefa para você assimilar tudo. Use os ímãs abaixo para preencher a fórmula que resolve a equação de “o que é HTML5?”. Tome cuidado agora, pois há algumas distrações misturadas à pilha de ímãs. Assim que você tiver resolvido, descanse um pouco e refresque-se antes de passar para o Capítulo 2.

E o seu primeiro
código JavaScript!

+ + =





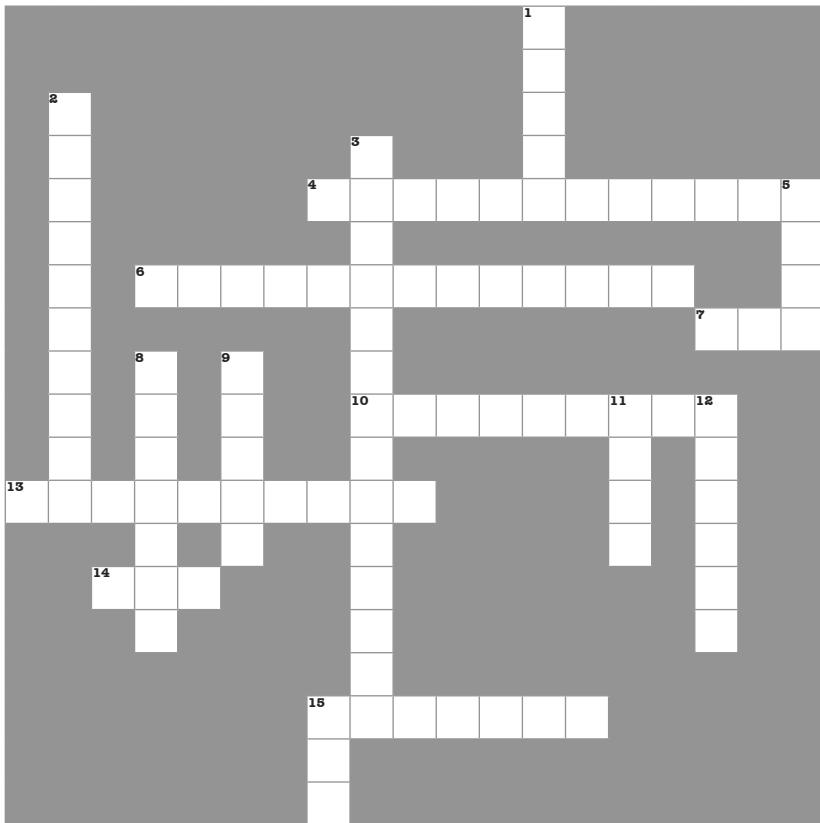
PONTOS DE BALA

- HTML5 é a versão mais recente de HTML. Ela introduz tags simplificadas, nova semântica e elementos de mídia, e se baseia em um conjunto de bibliotecas JavaScript que permite aplicativos web.
- XHTML não é mais o padrão para páginas web. Os desenvolvedores e o W3C decidiram continuar estendendo e melhorando a HTML.
- O doctype HTML5 novo e mais simples é suportado por navegadores mais antigos — eles usam o modo padrão quando veem este doctype.
- O atributo de tipo não é mais necessário na tag <script> ou em um link de stylesheet para CSS. JavaScript e CSS agora são os padrões.
- A tag <meta>, usada para especificar o conjunto de caracteres, foi simplificada para incluir apenas a codificação dos caracteres.
- UTF-8 é agora o conjunto padrão de caracteres em uso na web.
- Fazer alterações no doctype e tag <meta> não estragará suas páginas em navegadores mais antigos.
- Os novos elementos de HTML5 são um superconjunto de elementos HTML 4, o que significa que páginas mais antigas continuarão a funcionar em navegadores modernos.
- O padrão HTML5 não estará oficialmente completo até 2014, mas a maioria dos navegadores modernos o suportará muito antes disso (muitos o suportam hoje!).
- HTML5 introduz elementos que adicionam nova semântica às suas páginas, dando a você mais opções para a criação da estrutura da página web do que tínhamos com HTML 4.01. Não examinaremos essas opções neste livro, mas temos um pequeno guia para elas no apêndice.
- Muitos dos novos recursos em HTML5 requerem JavaScript para serem aproveitados ao máximo.
- Usando JavaScript, você pode interagir com o DOM — o Document Object Model.
- O DOM é a representação interna do navegador de uma página web. Usando JavaScript, você pode acessar e alterar elementos e adicionar novos elementos ao DOM.
- Uma API JavaScript é uma “Application Programming Interface”. APIs possibilitam controlar todos os aspectos de HTML5, como o desenho 2D, execução de vídeos e mais.
- JavaScript é uma das linguagens mais populares do mundo. As implementações de JavaScript têm melhorado drasticamente nos últimos anos.
- Você poderá detectar se um novo recurso é suportado em um navegador e degradar graciosamente a experiência em caso contrário.
- CSS é o padrão de definição de estilos para HTML5; muitas pessoas incluem CSS quando usam o termo “HTML5” para descrever a família de tecnologias usadas para criar aplicativos web.



Palavras cruzadas HTML5

É hora de dar ao lado direito do seu cérebro um descanso e colocar o esquerdo ao trabalho. Todas estas palavras são relacionadas a HTML e a este capítulo.



Horizontais

4. Propaganda ____ também poderia ser chamada de spam.
6. Queremos que nossas experiências web degradem ____.
7. O padrão de estilo oficial para HTML5.
10. Novos ____ em HTML adicionam semântica e estrutura.
13. Produto que limpa seu HTML5 em três etapas.
14. JavaScript é ____ vezes mais rápida do que há uma década.
15. Muito mais simples do que a versão HTML 4.01.

Verticais

1. Use um loop ____ para imprimir versos de uma música.
2. A linguagem de scripting padrão de HTML5.
3. Sua missão era _____ de navegador.
5. O poder real de HTML5 são as ____ JavaScript.
8. A versão de HTML antes de HTML5.
9. Recebeu uma carta de despedida.
11. Este atributo do link e tags de script não é mais necessário em HTML5.
12. A tag <____> informa ao navegador que o que se segue é JavaScript, não HTML.
15. O ____ é uma representação interna de uma página web.

QUEM FAZ O QUÉ?

SOLUÇÃO

Já falamos tanto sobre a “família de tecnologias” que achamos que elas são... bem, uma família. No entanto, de novo, ainda não a conhecemos bem. Então, será que não está na hora? Você encontrará a maior parte da família abaixo. Vá em frente e socialize-se; veja se consegue descobrir quem é quem. Nós nos adiantamos e já descobrimos uma para você. Não se preocupe, sabemos que este é o seu primeiro encontro com os membros da família HTML5; então, aqui estão as respostas.

CSS3

Usando-me, você pode desenhar direto na sua página web. Comigo, você consegue desenhar texto, imagens, retas, círculos, retângulos, padrões e gradientes.

Web Workers

Você talvez já tenha me usado em HTML 4 para digitar informações, mas estou melhor ainda em HTML5. Posso requerer que você preencha todos os campos e posso verificar mais facilmente se você digitou um email, URL, ou número de telefone onde deveria.

Formulários

Você precisava de um plugin para nós, mas agora somos membros de primeira classe da família de elementos HTML. Quer assistir ou ouvir algo? Precisa de nós.

Aplicativos Web Offline

Estamos aqui para ajudar com a estrutura e significado semântico da sua página, incluindo novas formas de criar seções, cabeçalhos, rodapés e navegação.

Áudio e Vídeo

Sou o mais estiloso da família. Você provavelmente já me usou antes, mas sabia que agora posso animar seus elementos, dar a eles bordas arredondadas e até mesmo sombras?

Nova Marcação

Use-me como um pouco de armazenamento local no navegador de cada usuário. Precisa armazenar novas preferências, alguns itens de carrinhos de compras, ou talvez até mesmo um grande cache para aumentar a eficiência? Eu sou a sua API.

Armazenamento Local

Precisa de aplicativos que funcionem mesmo quando não estiver conectado à rede? Posso ajudar.

Canvas

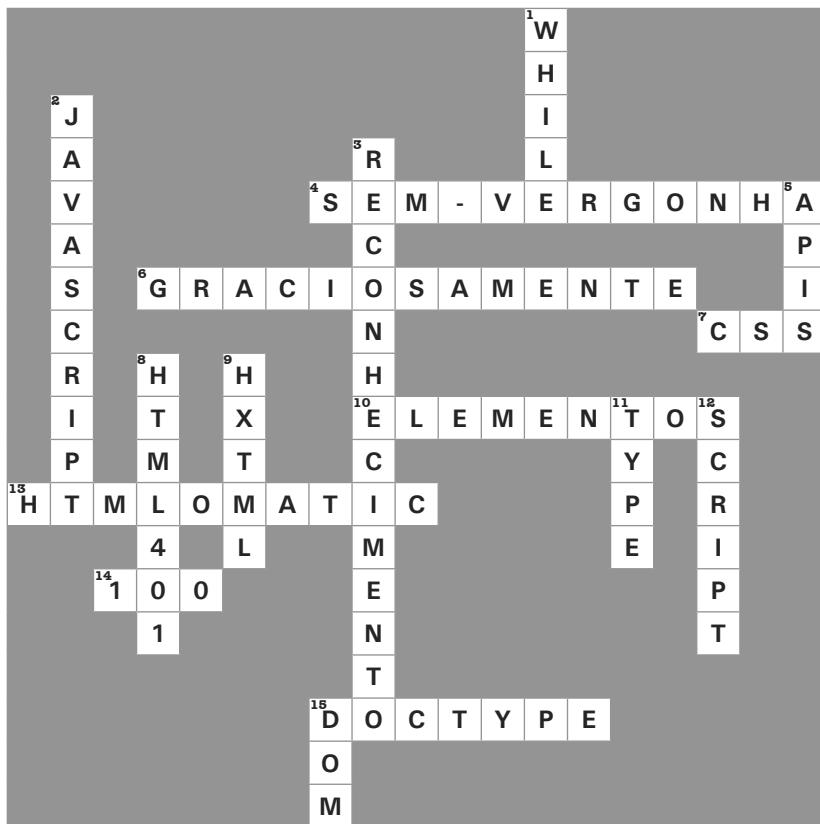
Sou a API que pode lhe informar onde você está, e me saio bem com o Google Maps.

Geolocalização

Você irá me querer sempre que precisar que diversos scripts sejam executados concorrentemente e em segundo plano, de modo que sua interface de usuário continue responsiva.



Palavras Cruzadas HTML5 – Solução



2 apresentando JavaScript e DOM



Um Pouco de Código

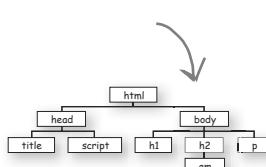
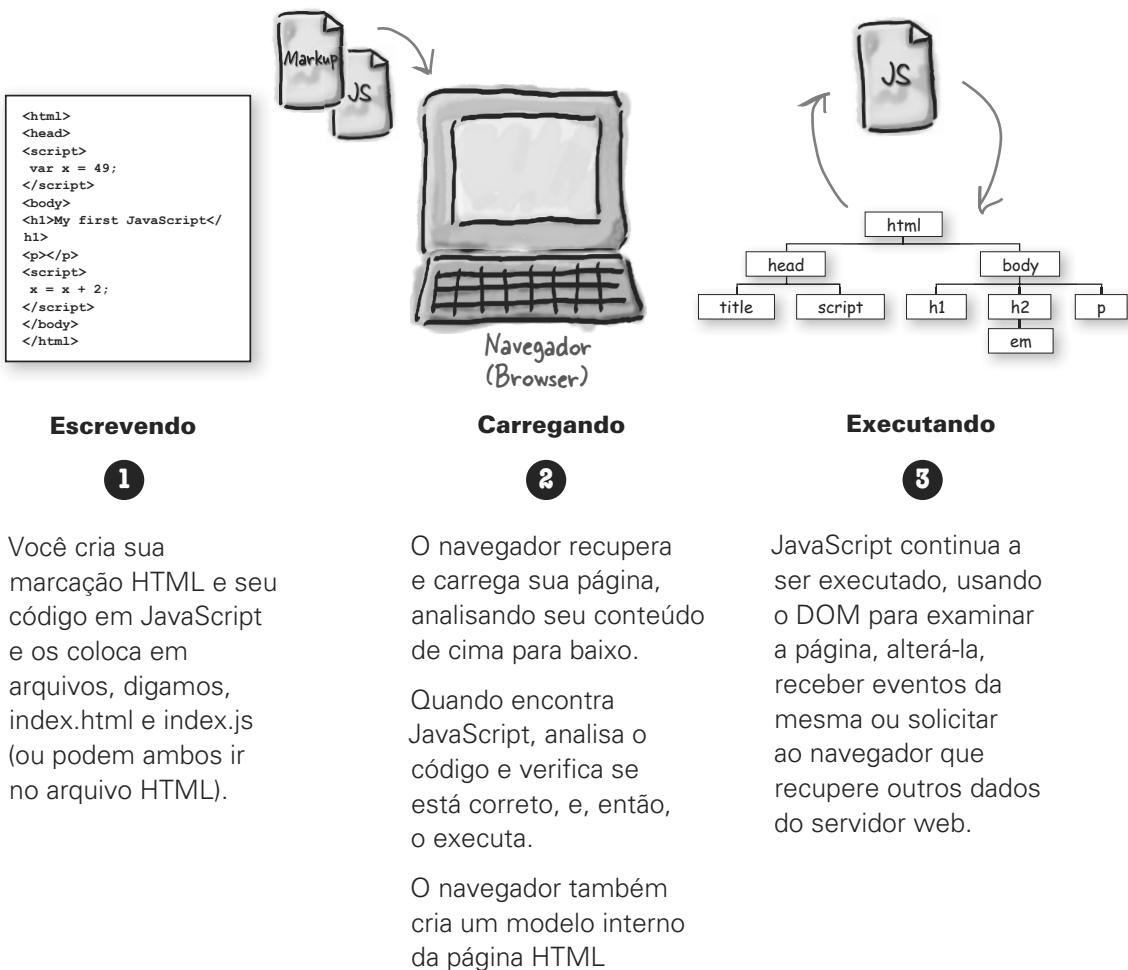


JavaScript lhe levará a novos lugares. Você já sabe tudo sobre marcação HTML (conhecida como *estrutura*) e sabe tudo sobre estilo CSS (conhecido como *apresentação*), mas ainda está faltando o JavaScript (conhecido como *comportamento*). Se tudo o que você souber for relacionado à estrutura e apresentação, certamente poderá criar algumas páginas com ótima aparência, mas elas ainda serão *apenas* páginas. Quando você adiciona comportamento com JavaScript, pode criar uma experiência interativa ou, melhor ainda, pode criar aplicativos web completos. Apronte-se para adicionar a habilidade mais interessante e versátil ao seu kit de ferramentas web: JavaScript e programação!

E, se você precisar de
mais motivação, a mais
lucrativa também!

Como JavaScript funciona

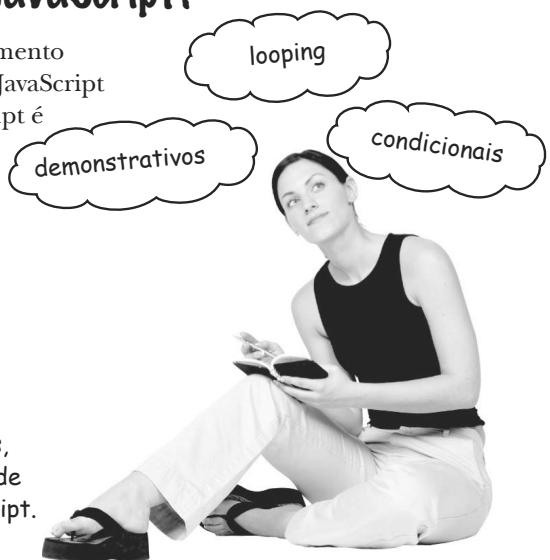
Nosso objetivo é escrever código em JavaScript que seja executado no navegador, quando sua página web for carregada — esse código poderia responder a ações do usuário, atualizar ou alterar a página, comunicar-se com serviços web e, de modo geral, tornar sua página mais parecida com um aplicativo do que com um documento. Vejamos como tudo isso funciona:



O que você pode fazer com JavaScript?

Assim que você tiver uma página com um elemento <script> (ou uma referência a um arquivo JavaScript separado), pode começar a codificar. JavaScript é uma linguagem de programação madura e você pode fazer com ela quase qualquer coisa que faria com outras linguagens, e até mais, porque estamos programando dentro de uma página web!

Você pode mandar JavaScript:



1 criar um comando

Crie uma variável e atribua valores, adicione coisas, use a funcionalidade interna de uma biblioteca JavaScript.

```
var temp = 98.6;
var beanCounter = 4;
var reallyCool = true;
var motto = "I Rule";
temp = (temp - 32) * 5 / 9;
motto = motto + " and so do you!";
var pos = Math.random();
```

2 fazer coisas mais de uma vez

Execute comandos repetidamente, tantas vezes quantas você precisar.

```
while (beanCounter > 0) {
    processBeans();
    beanCounter = beanCounter - 1;
}
```

3 tomar decisões

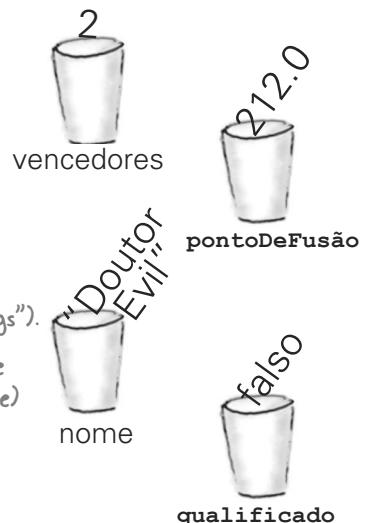
Escreva código que seja condicional, dependendo do estado do seu aplicativo.

```
if (isReallyCool) {
    invite = "You're invited!";
} else {
    invite = "Sorry, we're at capacity.";
}
```

Declarando uma variável

Variáveis armazenam coisas. Com JavaScript, elas podem armazenar muitas coisas diferentes. Vamos ver algumas dessas variáveis que armazenam coisas:

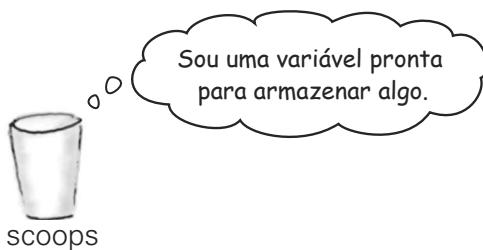
```
var winners = 2;           ← Valor numérico inteiro.  
var boilingPt = 212.0;     ← Ou valor de ponto flutuante.  
var name = "Dr. Evil";    ← Ou strings de caracteres  
                           (chamámos apenas de "strings").  
var isEligible = false;   ← Ou um valor booleano, que  
                           pode ser verdadeiro (true)  
                           ou falso (false).
```



Três passos na criação de uma variável

① →
 var scoops = 10;
 ③ ↓ ② ↓

- ① O primeiro passo é declarar sua variável, neste caso, *scoops* (conchas). Perceba que JavaScript, diferentemente de algumas linguagens, não precisa de um tipo para a variável, ele simplesmente cria um contêiner genérico que pode armazenar muitas coisas:

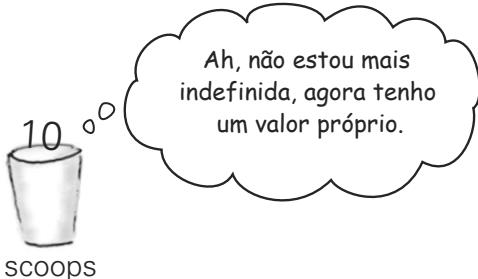


- ② A seguir, precisamos de um valor para colocar na variável. Podemos especificar um valor de algumas maneiras:

Seu valor pode ser um literal,
como um número ou uma string.
 var scoops = 10; ← Ou pode ser o resultado
var scoops = totalScoops / people; de uma expressão.
 var scoops = Math.random() * 10; ↑ Ou usar uma das funções
 das bibliotecas internas de
 JavaScript, como um gerador
 de números aleatórios, para
 criar um valor. Posteriormente,
 veremos mais sobre isto e suas
 próprias funções.

Variáveis são contêineres para armazenar valores. Variáveis JavaScript não têm tipos estritos, de forma que qualquer variável pode armazenar um número, uma string ou um valor booleano.

- 3** Finalmente, temos uma variável e um valor (um valor literal, como 10, ou o resultado da avaliação de uma expressão (como `totalScoops / people`)) e tudo o que precisamos fazer é atribuir esse valor à variável.



É claro que assim que você tiver uma variável criada, poderá alterar seu valor a qualquer momento, ou até mesmo alterá-la para um valor de um tipo diferente. Aqui estão alguns exemplos:

```
scoops = 5;           ← Podemos reinicializar scoops com
                      outro valor inteiro.
scoops = scoops * 10; ← Ou até mesmo usar a própria
                      variável em uma expressão
                      que altere seu valor. Neste
                      caso, scoops valerá 50.
scoops = "Tired of being an integer";
```

```
scoops = null;
```

↑

Há um valor em JavaScript chamado `null`, que significa "valor nenhum". Veremos como isso é usado posteriormente.



Diversão com a Sintaxe

- Cada comando termina com um ponto-e-vírgula.
`x = x + 1;`
- Um comentário de linha única começa com duas barras. Comentários são apenas notas para você ou outros desenvolvedores sobre o código. Eles não são avaliados.
`// Eu sou um comentário`
- Espaços em branco não contam (em quase todos os lugares).
`x = 2233;`
- Coloque aspas duplas em torno das strings.
`"Você é demais!"`
- Variáveis são declaradas usando `var` e um nome. Não é necessário informar um tipo, diferentemente de outras linguagens.
`var width;`
- Não use aspas em valores booleanos `true` e `false`.
`rockin = true;`
- Variáveis não precisam receber um valor quando são declaradas:
`var width;`

→ Perguntas Idiotas

P: Qual é o valor da minha variável quando escrevo:

`var vencedor;`

R: Após este comando ser executado, a variável `vencedor` receberá um valor `undefined` (indefinido), que é outro valor e tipo JavaScript. Veremos onde e como usar isto posteriormente.

P: Tenho visto outras linguagens de programação em que variáveis são declaradas com um tipo, como `int x` ou `String y`. JavaScript não possui tipos?

R: JavaScript tem tipos, mas, ao contrário de outras linguagens que você talvez tenha usado anteriormente, possui tipagem dinâmica, o que significa que você não precisa especificar um tipo, pois o interpretador JavaScript descobrirá que tipo usar quando seu código estiver sendo executado.

Como dar nomes às suas variáveis

Você talvez esteja imaginando como escolher nomes para suas variáveis. Se você estiver acostumado a dar nomes a ids nos seus elementos HTML, achará as variáveis muito semelhantes. Há apenas algumas regras para a criação de nomes.

Regra #1: Comece suas variáveis com uma letra, um caractere de sublinhado ou um sinal de cifrão.

Você quer começar a conhecer bem a nomenclatura de variáveis, não apenas as tornando significativas, mas também usando uma letra (minúscula ou maiúscula), um caractere de sublinhado ou um cifrão. Aqui estão alguns exemplos:



```
var thisIsNotAJoke;
var _myVariable;
var $importantVar;
```

Faça isto...

Começa por número, não é bom.

Começa com símbolos (%) e (^) que não são permitidos.

```
→ var 3zip;
→ var %entage;
→ var ~approx;
```

... não isto.



Regra #2: Depois você pode usar qualquer número de letras, dígitos numéricos, sublinhados e cifrões.

Continue usando letras, cifrões e sublinhados para criar seu nome de variável. Após o primeiro caractere, você também pode colocar números, caso queira:



```
var my3sons;
var cost$;
var vitaminB12;
```

Faça isto...

Contém um espaço, o que não é permitido

Contém sinais -, +. Não são permitidos e confundirão muito o JavaScript.

```
→ var zip code;
```

```
→ var first-name;
```

```
→ var to+do;
```

... não isto.



Regra #3: Assegure-se de evitar todas as palavras reservadas de JavaScript

JavaScript contém um número de palavras que são reservadas, tais como if, while e for (para citar apenas algumas), e JavaScript não é tão gentil para você tentar usar estas palavras reservadas para o nome de suas variáveis. Aqui está uma lista de palavras reservadas do JavaScript. Você não precisa memorizá-las; vai desenvolver um senso do que elas são, enquanto aprende JavaScript. Cada vez que ficar perplexo com a JavaScript reclamando de como você declarou suas variáveis, pense, “Hmm, esta palavra que estou tentando usar é uma palavra reservada?”

| | | | | |
|----------|----------|------------|--------------|-----------|
| abstract | delete | goto | null | throws |
| as | do | if | package | transient |
| boolean | double | implements | private | true |
| break | else | import | protected | try |
| byte | enum | in | public | typeof |
| case | export | instanceof | return | use |
| catch | extends | int | short | var |
| char | false | interface | static | void |
| class | final | is | super | volatile |
| continue | finally | long | switch | while |
| const | float | namespace | synchronized | with |
| debugger | for | native | this | |
| default | function | new | throw | |



Evite estes nomes variáveis

→ não existem Perguntas Idiotas

P: E se eu usasse uma palavra reservada como parte do meu nome de variável? Por exemplo, eu posso ter uma variável com o nome de ifOnly (ou seja, uma variável que contenha a palavra reservada “if”)?

R: Certamente pode, apenas não a crie exatamente igual à palavra reservada. Também é bom escrever código claro. Você não deve usar algo como elze, que pode ser confundido com else.

P: JavaScript diferencia maiúsculas de minúsculas?
Em outras palavras, minhavariavel e MinhaVariavel são a mesma coisa?

R: Se você estiver acostumado com a marcação HTML, pode também estar com linguagens que não diferenciam maiúsculas de minúsculas, afinal <head> e <HEAD> são tratadas da mesma forma pelo navegador. Com JavaScript, entretanto, existe essa diferença e minhavariavel e MinhaVariavel são duas variáveis diferentes.

P: Entendo que JavaScript pode atribuir um valor a uma variável a qualquer momento (número, string e assim por diante). O que acontece, então, quando adiciono duas variáveis e uma é numérica e a outra é uma string de caracteres?

R: JavaScript tenta ser inteligente quanto à conversão de tipos quando necessário. Por exemplo, se você adicionar uma string a um número, ele geralmente tenta converter o número para uma string e concatenar as duas. Embora em alguns casos isso seja ótimo, às vezes não é o que você queria. Não esqueça esta questão, porque voltaremos em breve a este assunto.

O Guia da Webville para uma Boa Nomenclatura

Você tem muita flexibilidade na escolha dos seus nomes de variáveis, então queremos lhe dar algumas dicas para tornar sua nomenclatura mais fácil:



Escolha nomes que signifiquem algo.

Nomes de variáveis como `_m`, `r` e `foo` podem significar algo para você, mas geralmente não são aprovados na Webville. Você provavelmente não apenas os esquecerá com o tempo, como o seu código será muito mais legível com nomes como `ângulo`, `pressãoAtual` e `aprovado`.

Use a primeira letra em maiúscula ao criar nomes de variáveis com mais de uma palavra.

Em algum momento, você terá de decidir como nomeará uma variável que representa, digamos, um dragão de duas cabeças com fogo. Como? Use as primeiras letras de cada palavra em maiúsculas, excetuando a primeira:

`dragãoDeDuasCabeçasComFogo`. Esta é uma formatação fácil, muito usada em Webville e lhe dá suficiente flexibilidade para criar um nome de variável tão específico quanto você precisar. Há outros esquemas também, mas este é o mais comumente usado (mesmo fora de JavaScript).

Use variáveis que começem com `_` e `$` apenas por um motivo muito bom.

Variáveis que começam com `$` geralmente são reservadas para bibliotecas JavaScript e, embora alguns autores usem variáveis começando com `_` por diversas convenções, elas não são amplamente usadas e recomendamos que você evite-as, a menos que tenha um motivo muito bom (você saberá se tiver).

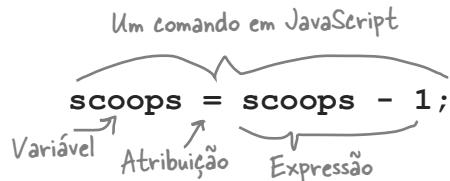
Não se arrisque.

Não se arrisque na sua nomenclatura de variáveis; veremos mais algumas dicas para isso posteriormente no livro, mas, por enquanto, seja claro, evite palavras reservadas e sempre use `var` ao declarar uma variável.



Tornando-se Expressivo

Já vimos alguns comandos em JavaScript que se parecem com



Examinemos, porém, mais de perto as expressões, como a deste comando. Elas estão em todos os lugares em JavaScript, de modo que é importante conhecer os tipos de coisas que você pode expressar. Aqui estão alguns...

Você pode escrever expressões que resultem em números...

Expressões numéricas

```
(9 / 5) * tempC + 32
x - 1
Math.random() * 10
2.123 + 3.2
```

... e você pode escrever expressões que resultem em strings.

Expressões string

```
"super" + "cali" + youKnowTheRest
"March" + "21" + "st"
p.innerHTML
phoneNumber.substring(0, 3)
```

Fique de olho nas expressões das próximas páginas (para não dizer do resto do livro) e verá como são usadas para calcular coisas, realizar tarefas repetidamente e tomar decisões no seu código.

Você pode escrever expressões que resultem em valores booleanos verdadeiros ou falsos (estas são, obviamente, expressões booleanas).

Expressões booleanas

```
2 > 3
tempF < 75
pet == "Duck"
startTime > now
level == 4
```

Há outros tipos de expressões também; nós as veremos em breve.

Outras Expressões

```
function () {...}
document.getElementById("pink")
new Array(10)
```



Expresse-se!

Você viu os diferentes tipos de expressões que pode usar em JavaScript; agora é hora de colocar esse conhecimento para trabalhar na avaliação de algumas expressões. Verifique suas respostas no final do capítulo.

`(9 / 5) * tempC + 32`

Qual o resultado, quando `tempC` for 10? _____

`"Number" + " " + "2"`

Qual a string resultante? _____

`level >= 5`

Qual o resultado, quando `level` (nível) for 10? _____

E quando for 5? _____

`color != "pink"`

← Dica: ! significa não.

Qual o resultado, se `color` (cor) for "blue" ("azul")? _____

`(2 * Math.PI) * r`

Qual o resultado, se `r` for 3? _____

Dica: `Math.PI` lhe dá o valor de pi (você sabe, 3.14...)



→ Não este tipo de expressão

Aponte o seu lápis

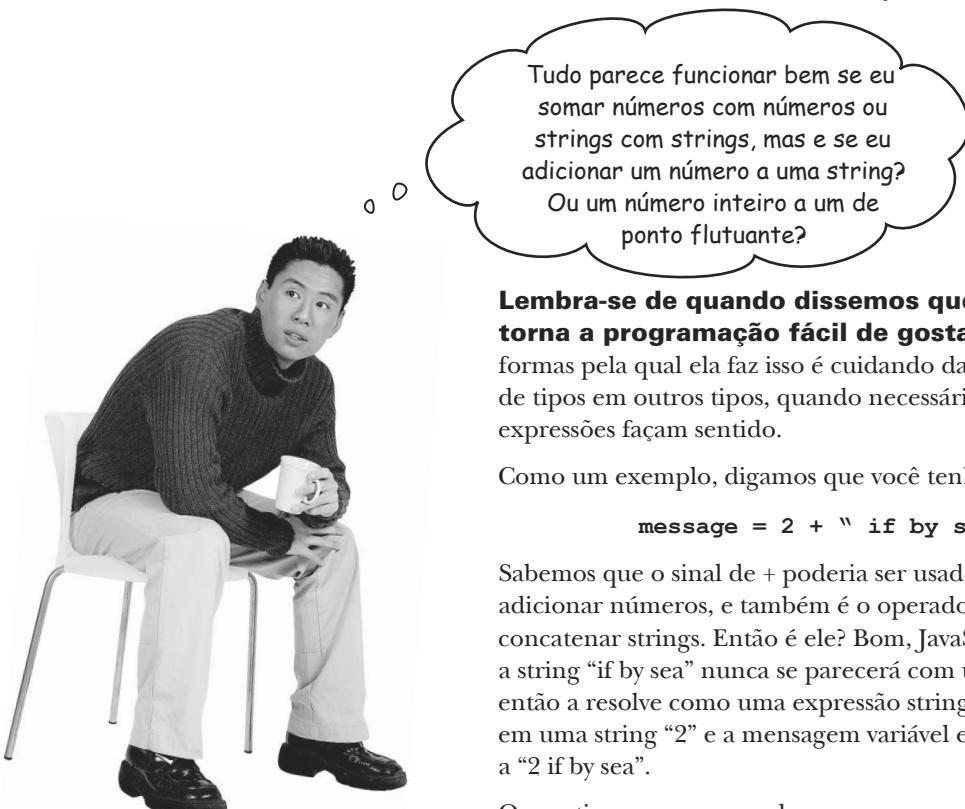
Baseado no que você sabe até agora sobre variáveis, expressões e comandos em JavaScript, veja se consegue descobrir qual destas são legais e quais poderiam gerar um erro.

Da lista a seguir, circule os comandos que são *legais*.

```

var x = 1138;
var y = 3/8;
var s = "3-8";
x = y;
var n = 3 - "one";
var t = "one" + "two";
var 3po = true;
var level_ = 11;
var highNoon = false;
var $ = 21.30;
var z = 2000;
var isBig = y > z;
z = z + 1;
z--;
z y;
x = z * t;
while (highNoon) {
    z--;
}

```



Lembra-se de quando dissemos que JavaScript torna a programação fácil de gostar? Uma das formas pela qual ela faz isso é cuidando da conversão de tipos em outros tipos, quando necessário para que as expressões façam sentido.

Como um exemplo, digamos que você tenha a expressão:

```
message = 2 + " if by sea";
```

Sabemos que o sinal de + poderia ser usado para adicionar números, e também é o operador usado para concatenar strings. Então é ele? Bom, JavaScript sabe que a string “if by sea” nunca se parecerá com um número, então a resolve como uma expressão string, converte o 2 em uma string “2” e a mensagem variável está atribuída a “2 if by sea”.

Ou, se tivermos o comando:

```
value = 2 * 3.1;
```

JavaScript converte o número inteiro 2 em um número de ponto flutuante e o resultado é 6.2.

Como você pode imaginar, porém, JavaScript nem sempre faz o que você quer e, em alguns casos, precisa de uma pequena ajuda nas conversões. Voltaremos a esse tópico daqui a pouco.

PODER DO CÉREBRO

Como JavaScript avalia os seguintes comandos?

```
numORString1 = "3" + "4"  
numORString2 = "3" * "4"
```

E por quê?



```
while (juggling) {  
    keepBallsInAir();  
}
```

Fazendo coisas repetidamente...

Se fizéssemos as coisas apenas uma vez em um programa JavaScript, ele provavelmente seria bem entediante. Você faz muitas coisas repetidamente - enxágua, faz espuma, repete, até que seu cabelo fique limpo, ou continua dirigindo até chegar ao seu destino, ou segue tomando seu sorvete até ele acabar — e, para lidar com estas situações, JavaScript lhe fornece algumas maneiras de iterar por blocos de código.

Você pode usar o loop `while` de JavaScript para executar algo até que uma condição seja satisfeita:

```
var scoops = 10;  
  
while (scoops > 0) {  
    alert("More icecream!");  
    scoops = scoops - 1;  
}  
  
alert("life without ice cream isn't the same");
```

Temos um pote de sorvete e ainda há dez conchas (scoops) nele. Aqui está uma variável declarada e inicializada como dez.

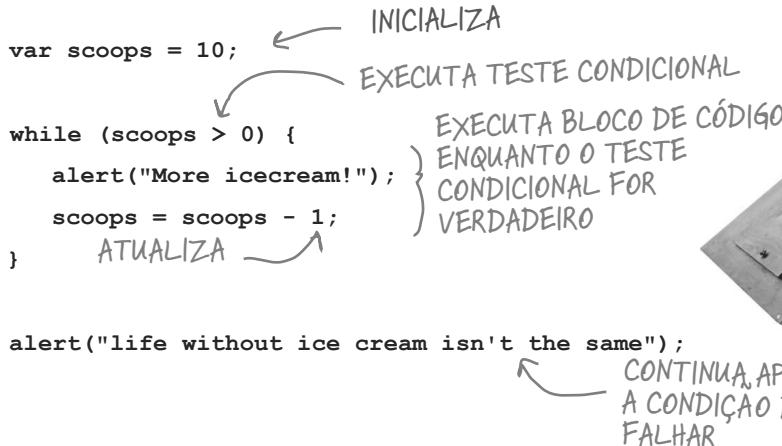
While usa uma expressão booleana que é avaliada em verdadeiro ou falso. Se for verdadeira, o código depois dela é executado.

Embora haja mais de zero scoops sobrando, continuaremos fazendo tudo que houver neste bloco de código.

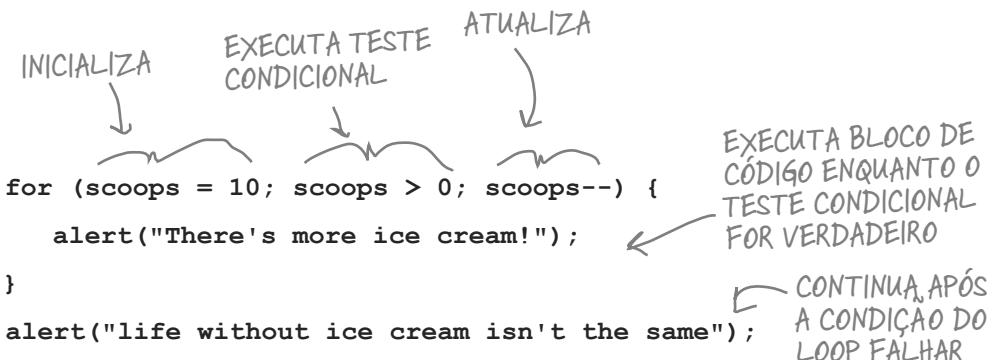
Cada vez que passamos por um loop while, alertamos o usuário de que ainda há sorvete, e então retiramos uma concha, subtraindo um do número de conchas.

Quando a condição (`scoops > 0`) for falsa, o loop termina, e a execução do código continua daqui, qualquer que seja a próxima linha do seu programa

Assim, se você pensar no loop `while`, estamos *inicializando* alguns valores, digamos, o número de conchas de sorvete que restam, que o loop `while` *testa* e, se verdadeiro, *executamos* um bloco de código. Este bloco executa algum trabalho que, em algum momento, *atualiza* o valor envolvido no teste condicional, de modo que a *condição falhe* e o loop termine.



JavaScript também fornece um loop `for`, que formaliza esta estrutura um pouco mais. Aqui está o nosso código do sorvete com um loop `for`:



Pnão existemerguntas Idiotas

P: Os loops `while` e `for` parecem a mesma coisa para mim. Quando eu uso qual?

R: De modo geral, você pode fazer as mesmas coisas com um `for` ou um `while`; entretanto, como você pode ver no exemplo do sorvete, o loop `for` é um pouco mais compacto, e você poderia argumentar que o loop `while` é mais legível. Assim, é realmente uma questão de qual se adapta melhor a uma determinada situação. De modo geral, loops `for` são usados para iterar por um determinado número de valores (digamos, pelos itens de um carrinho de compras), e loops `while` são usados mais para serem executados até que uma condição seja satisfeita (digamos, fazer um teste com o usuário até que ele acerte).



Sinta-se como o Navegador

Cada um dos trechos em JavaScript nesta página são códigos separados. Sua tarefa é dar uma de navegador, avaliar cada trecho de código e responder a uma pergunta sobre o resultado. Escreva suas respostas abaixo do código.

Cheque suas respostas no final do capítulo.

Trecho 1

```
var count = 0;
for (var i = 0; i < 5; i++) {
    count = count + i;
}
alert("count is " + count);
```

Trecho 1

```
var count = 0;
for (var i = 0; i < 5; i++) {
    count = count + i;
}
alert("count is " + count);
```

Que count o alerta apresenta?

}

Quanto tempo você vê o alerta: "Top is spinning!"?

Trecho 2

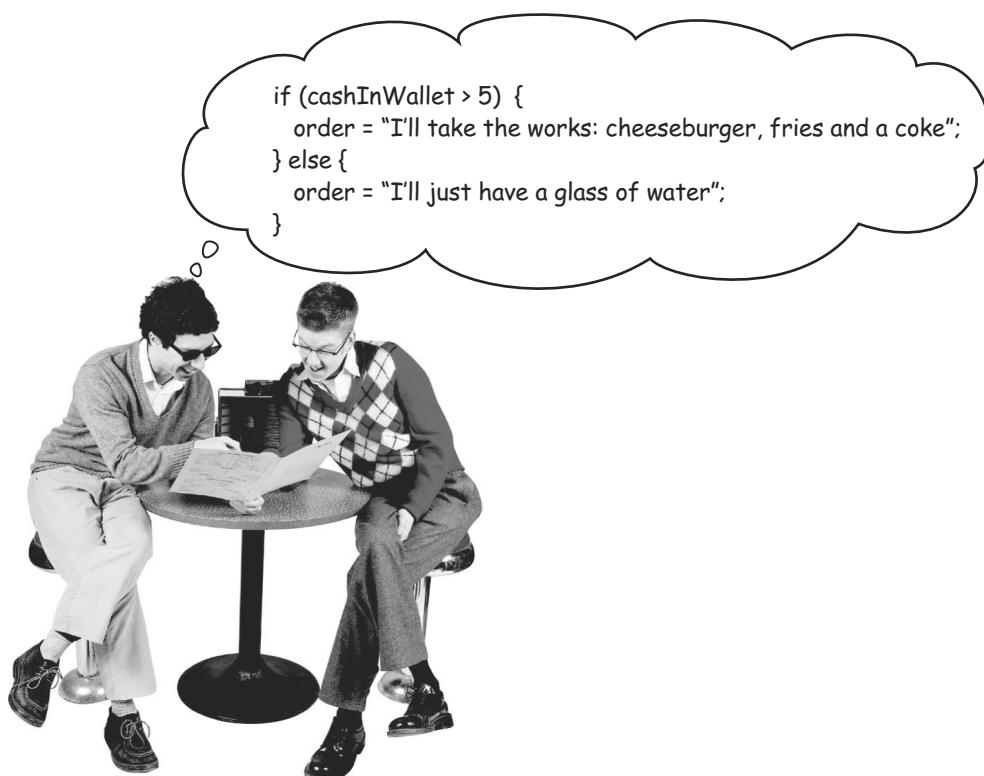
```
var tops = 5;
while (tops > 0) {
    for (var spins = 0; spins < 3; spins++) {
        alert("Top is spinning!");
    }
    tops = tops - 1;
}
```

Trecho 3

```
for (var berries = 5; berries > 0; berries--) {
    alert("Eating a berry");
}
```

Quantos berries você comeu? → _____

← Quantas conchas de sorvete você comeu?



Tome decisões com JavaScript

Temos usado expressões booleanas em comandos `for` e `while` como um teste condicional para decidir se devemos ou não continuar a execução do loop. Você também pode usá-las para tomar decisões em JavaScript. Aqui está um exemplo:

```

if (scoops < 3) {
    alert("Ice cream is running low!");
}

```

Aqui está nossa expressão booleana, teste para ver quantos scoops estão à esquerda.

Se há < 3 scoops à esquerda, executamos o código block.

Podemos juntar mais de um teste também:

```

if (scoops < 3) {
    alert("Ice cream is running low!");
} else if (scoops > 9) {
    alert("Eat faster, the ice cream is going to melt!");
}

```

Adicione quantos testes com "else if" precisar, cada um com seu bloco de código associado que será executado quando a condição for verdadeira.

Tomando mais decisões... e adicionando um default

Você pode fornecer um default para seus comandos if — um else final que é executado se todas as outras condições falharem. Adicionaremos mais alguns if/elses e um default:

```
if (scoops == 3) {
    alert("Ice cream is running low!");
} else if (scoops > 9) {
    alert("Eat faster, the ice cream is going to melt!");
} else if (scoops == 2) {
    alert("Going once!");
} else if (scoops == 1) {
    alert("Going twice!");
} else if (scoops == 0) {
    alert("Gone!");
} else {
    alert("Still lots of ice cream left, come and get it.");
}
```

Perceba que alteramos isto para que só aconteça quando o número de bolas de sorvete for exatamente 3.

Adicionamos condições para uma contagem regressiva.

Aqui está o nosso default; se nenhuma das condições acima for verdadeira, então este bloco certamente será executado.



Pegue o código acima e insira-o no loop while abaixo. Percorra o loop while e escreva os alertas na sequência em que ocorrem. Verifique sua resposta no final do capítulo.

Exercício

```
var scoops = 10;

while (scoops >= 0) {
    scoops = scoops - 1;
}

alert("life without ice cream isn't the same");
```

} { Insira o código acima aqui...

Escreva a saída aqui. ↗



Ímãs de Geladeira

Este código imprime um palíndromo em um alerta. O problema é que uma parte do código estava em ímãs de geladeira e caiu no chão. É sua tarefa juntar esse código novamente para que o palíndromo funcione. Cuidado: já havia alguns ímãs no chão que não fazem parte do código e você terá que usar alguns mais de uma vez! Verifique sua resposta no final do capítulo antes de seguir em frente.

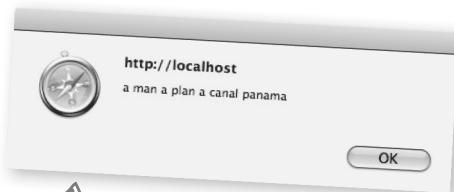
```

var word1 = "a";
var word2 = "nam";
var word3 = "nal p";
var word4 = "lan a c";
var word5 = "a man a p";

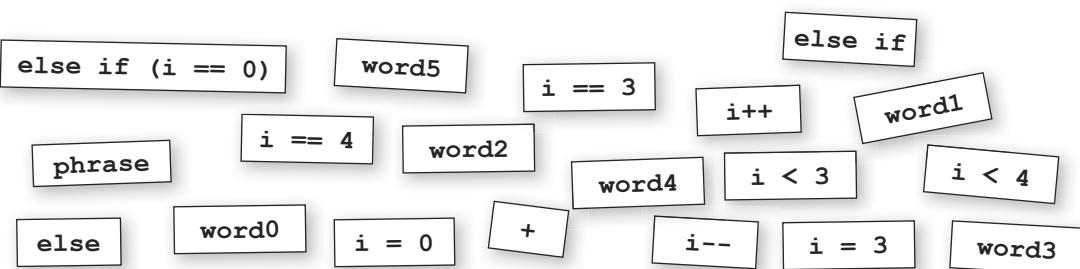
var phrase = "";

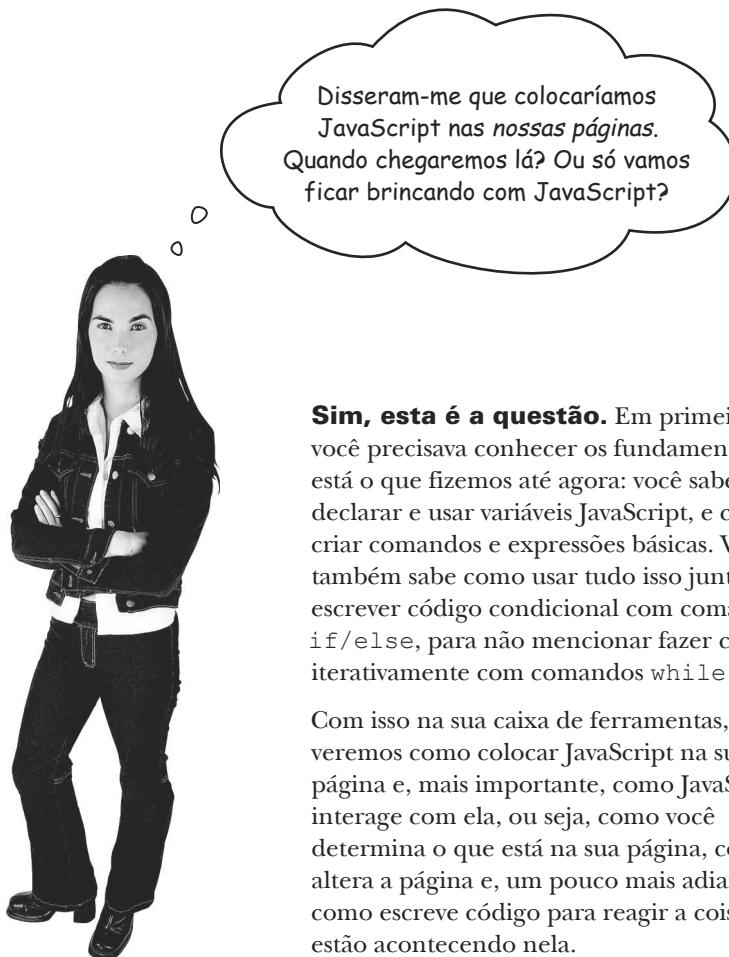
for (var i = 0; _____; _____) {
    if (i == 0) {
        phrase = _____;
    }
    else if (i == 1) {
        phrase = _____ + word4;
    }
    _____ (i == 2) {
        _____ = phrase + word1 + word3;
    }
    _____ (_____) {
        phrase = phrase + _____ + word2 + word1;
    }
}
alert(phrase);

```



↑ Um palíndromo é uma frase que pode ser lida da mesma forma de trás para frente ou de frente para trás! Aqui está o palíndromo que você deve ver, se os ímãs estiverem todos nos lugares certos.





Disseram-me que colocaríamos
JavaScript nas *nossas páginas*.
Quando chegaremos lá? Ou só vamos
ficar brincando com JavaScript?

Sim, esta é a questão. Em primeiro lugar, você precisava conhecer os fundamentos. Aqui está o que fizemos até agora: você sabe como declarar e usar variáveis JavaScript, e como criar comandos e expressões básicas. Você também sabe como usar tudo isso junto para escrever código condicional com comandos `if/else`, para não mencionar fazer coisas iterativamente com comandos `while` e `for`.

Com isso na sua caixa de ferramentas, agora veremos como colocar JavaScript na sua página e, mais importante, como JavaScript interage com ela, ou seja, como você determina o que está na sua página, como altera a página e, um pouco mais adiante, como escreve código para reagir a coisas que estão acontecendo nela.

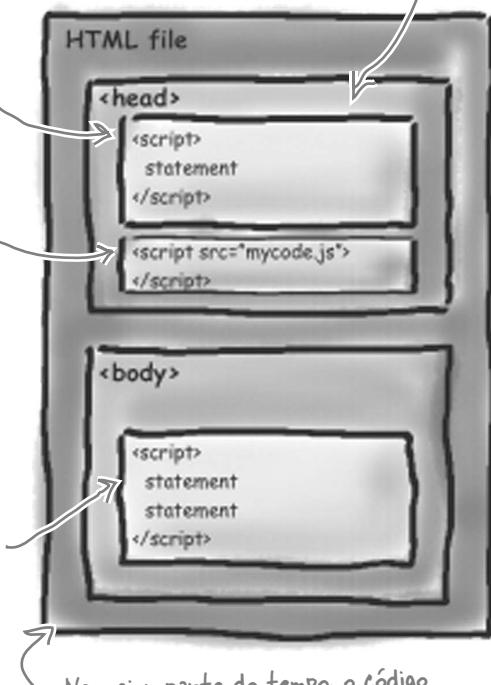
Então, embora ainda não tenhamos terminado com JavaScript, sua espera terminou; está na hora de vermos como a marcação e o comportamento trabalham juntos...

Como e onde adicionar JavaScript às suas páginas

Para usar JavaScript, você tem que adicioná-la a uma página web, mas onde e como? Você já sabe que há um elemento `<script>`, então vejamos onde podemos usá-lo e como ele afeta a forma pela qual podemos executá-lo dentro das suas páginas. Aqui estão três formas diferentes, através das quais você poderia adicionar código à sua página:

Coloque os elementos `<script>` no `<head>` do seu HTML para fazer com que eles sejam executados antes que a página seja carregada.

Você pode digitar seu código direto na página web, ou referenciar um arquivo JavaScript separado usando o atributo `src` da tag do script.



Na maior parte do tempo, o código é adicionado ao cabeçalho da página. Há algumas pequenas vantagens no desempenho ao acrescentar seu código no final do corpo, mas apenas se realmente for necessário superotimizar o desempenho da sua página.

Coloque seu script inline no elemento `<head>`.

A forma mais comum de adicionar código às suas páginas é colocando um elemento `<script>` no cabeçalho delas. Quando você adiciona JavaScript no elemento `<head>`, sua execução acontece assim que o navegador analisa o cabeçalho (o que ele faz primeiro!), antes de ter analisado o resto da página.

Adicione seu script referenciando um arquivo JavaScript separado.

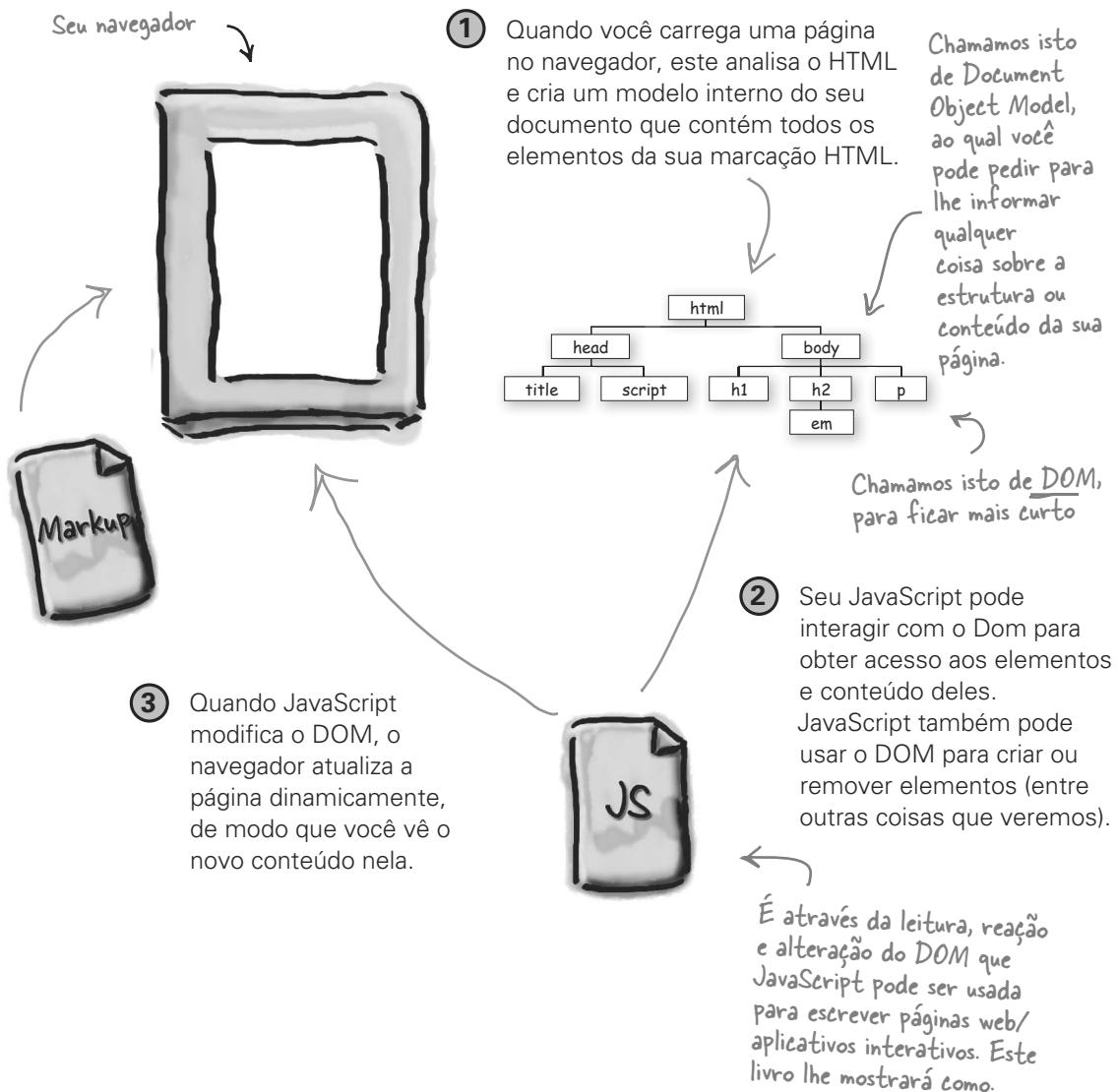
Você também pode conectar um arquivo separado contendo código JavaScript. Coloque a URL do arquivo no atributo `src` do `<script>` de abertura e assegure-se de fechar o elemento do script com `</script>`. Se você estiver conectando a um arquivo no mesmo diretório, pode usar apenas o nome deste arquivo.

Adicione seu código no corpo do documento, seja inline ou como um link para um arquivo separado.

Ou você pode colocar seu código diretamente no corpo do seu HTML. Envolva seu código JavaScript no elemento `<script>` (ou referecie um arquivo separado no atributo `src`). O JavaScript no corpo da sua página é executado, quando o navegador analisa o corpo (o que ele geralmente faz de cima para baixo).

Como JavaScript interage com sua página

JavaScript e HTML são duas coisas diferentes. HTML é marcação e JavaScript é código. Então, como você faz JavaScript interagir com a marcação na sua página? Você usa o Document Object Model.



Como cozinhar seu próprio DOM

Vamos pegar um pouco de marcação e criar um DOM para ela. Aqui está uma receita simples para fazer isso:

Ingredientes

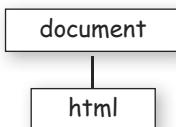
Uma página HTML5 bem formada
Um ou mais navegadores web

Instruções

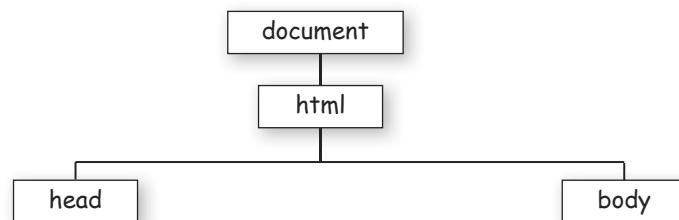
1. Comece criando um documento como raiz no topo.



2. A seguir, pegue o elemento do topo da sua página HTML, no nosso caso o elemento <html>, chame-o de elemento corrente e adicione-o como ramificação do documento.

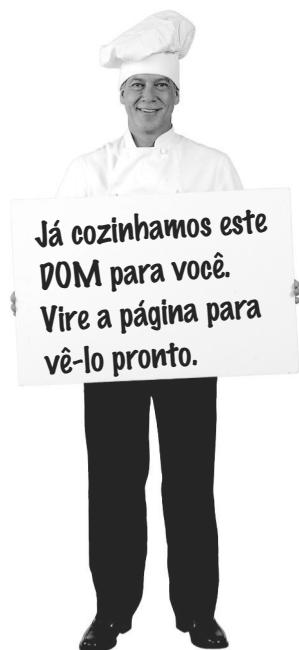


3. Adicione cada elemento aninhado no elemento corrente como filho do elemento documento no DOM.



4. Volte para (3) para cada elemento que você acabou de adicionar e repita até que não haja mais elementos.

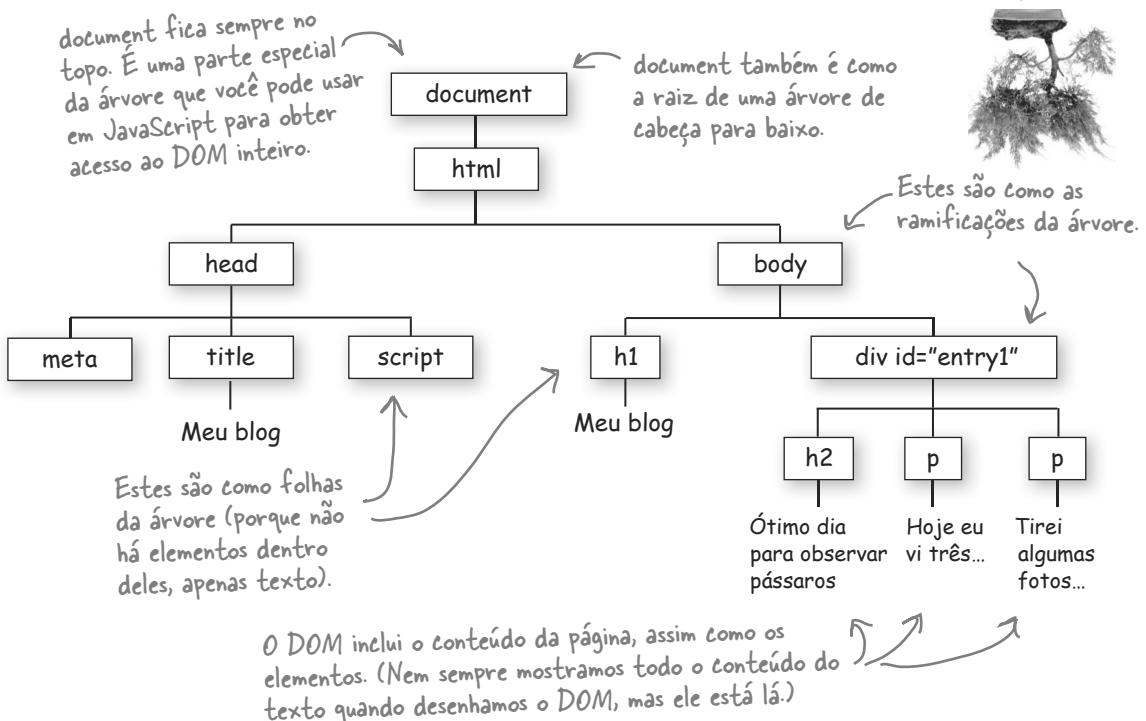
```
<!doctype html>
<html lang="en">
<head>
  <title>My blog</title>
  <meta charset="utf-8">
  <script src="blog.js"></script>
</head>
<body>
  <h1>My blog</h1>
  <div id="entry1">
    <h2>Great day bird watching</h2>
    <p>
      Today I saw three ducks!
      I named them
      Huey, Louie, and Dewey.
    </p>
    <p>
      I took a couple of photos...
    </p>
  </div>
</body>
</html>
```



Uma primeira prova de DOM

A beleza do Document Object Model é que ele nos apresenta formas consistentes, por todos os navegadores, de obter acesso à estrutura e ao conteúdo do HTML a partir do código. Isto é importante. Veremos como tudo isso funciona em breve...

Voltando ao nosso exemplo, se você seguiu a receita da criação de um DOM, acabou com uma estrutura como a apresentada abaixo. Cada DOM possui um objeto documento no topo e depois uma árvore completa com ramificações e nodos de folhas para cada um dos elementos da marcação HTML. Vejamos isso mais de perto.



Comparamos esta estrutura a de uma árvore, porque uma "árvore" é uma estrutura de dados que vem da ciência da computação.



Estes são como as ramificações da árvore.

Ótimo dia para observar pássaros
Hoje eu vi três...
Tirei algumas fotos...

O DOM inclui o conteúdo da página, assim como os elementos. (Nem sempre mostramos todo o conteúdo do texto quando desenhamos o DOM, mas ele está lá.)

Agora que temos um DOM, podemos examinar ou alterá-lo de qualquer maneira que quisermos.



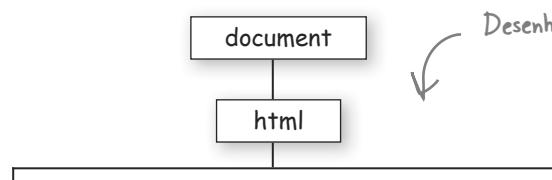
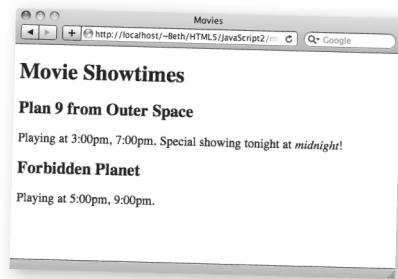


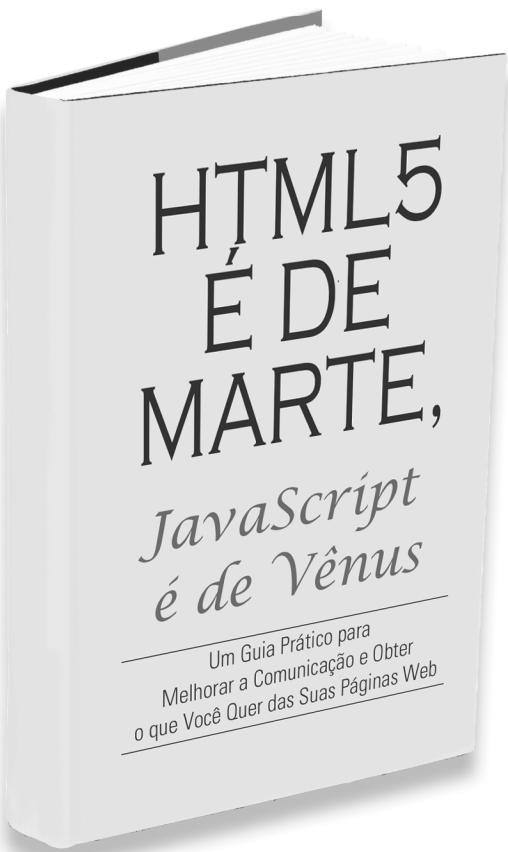
Sinta-se como o Navegador

Sua tarefa é agir como se fosse o navegador. Você precisa analisar o HTML e criar seu próprio DOM a partir dele. Siga em frente e analise o HTML à direita e desenhe seu DOM abaixo. Já começamos para você.

Compare sua resposta com nossa solução no final do capítulo antes de seguir adiante.

```
<!doctype html>
<html lang="en">
  <head>
    <title>Movies</title>
  </head>
  <body>
    <h1>Movie Showtimes</h1>
    <h2 id="movie1" >Plan 9 from Outer Space</h2>
    <p>Playing at 3:00pm, 7:00pm.
      <span>
        Special showing tonight at <em>midnight</em>!
      </span>
    </p>
    <h2 id="movie2">Forbidden Planet</h2>
    <p>Playing at 5:00pm, 9:00pm.</p>
  </body>
</html>
```





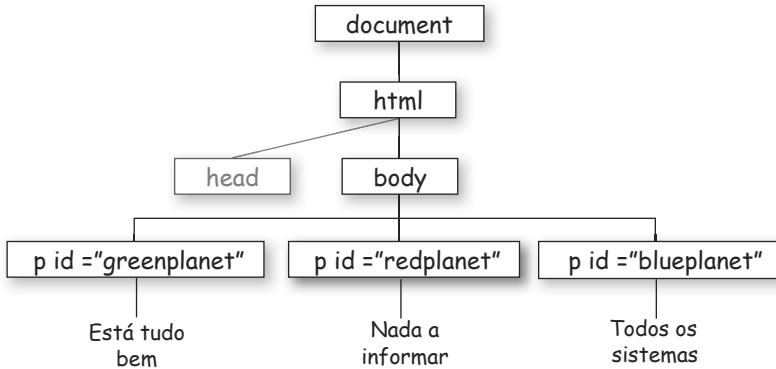
Ou como duas tecnologias totalmente diferentes se conectaram.

HTML e JavaScript são dois planetas completamente diferentes. A prova? O DNA do HTML é feito de marcação declarativa, que permite a você descrever um conjunto de elementos aninhados que constituem a sua página. JavaScript, por outro lado, é constituído de material genético de algoritmo puro, feito para descrever computações.

Eles são tão diferentes que não podem nem se comunicar? É claro que não, porque têm algo em comum: o DOM. Através do DOM, JavaScript se comunica com a sua página, e vice-versa. Há algumas formas de fazer isso acontecer, mas por enquanto nos concentraremos em uma — é um pequeno buraco que permite a JavaScript acessar qualquer elemento, chamado de `getElementById`.

Vejamos como ele funciona...

Começaremos com um DOM. Aqui está um DOM simples; ele possui alguns parágrafos, cada um com uma id identificando-o como planeta verde (green planet), vermelho (red planet) ou azul (blue planet). Cada parágrafo também possui algum texto. É claro que há um elemento <head> também, mas deixamos os detalhes de fora para manter as coisas mais simples.



Agora usaremos JavaScript para tornar as coisas mais interessantes. Digamos que queremos alterar o texto do planeta verde de “Está tudo bem” para “Alerta Vermelho: atingido por fogo de phaser!” (“Red Alert: hit by phaser fire”). Mais adiante, você poderá querer fazer algo como isto, baseado nas ações de um usuário ou até mesmo em dados de um serviço web. Chegaremos lá; por enquanto, simplesmente atualizaremos o texto do planeta verde. Para fazer isso, precisamos do elemento com uma `id` igual a `greenplanet`. Aqui está o código que faz isso:

Lembre-se de que o documento representa a página inteira no seu navegador e contém o DOM completo, de modo que podemos solicitar a ele que faça coisas como encontrar um elemento com uma id específica.

Aqui estamos solicitando ao documento para nos trazer um elemento, encontrando o que tenha um determinado id.

```
document.getElementById("greenplanet");
```

`getElementById("greenplanet")`
retorna o elemento de parágrafo que corresponde a “greenplanet” ...

... e depois o código JavaScript pode fazer todos os tipos de coisas interessantes com ele...

usando getElementById

Assim que `getElementById` lhe apresentar um elemento, você estará pronto para fazer algo com ele (como alterar seu texto para "Alerta Vermelho: atingido por fogo de phaser!"). Para fazer isso, geralmente atribuímos o elemento a uma variável, de modo que possamos nos referir ao elemento por todo o nosso código; faremos isso e depois alteraremos o texto:

Estamos atribuindo o elemento a uma variável chamada `planet`.

Aqui está nossa chamada a `getElementById`, que procura o elemento "greenplanet" e o retorna.

```
var planet = document.getElementById("greenplanet");
```

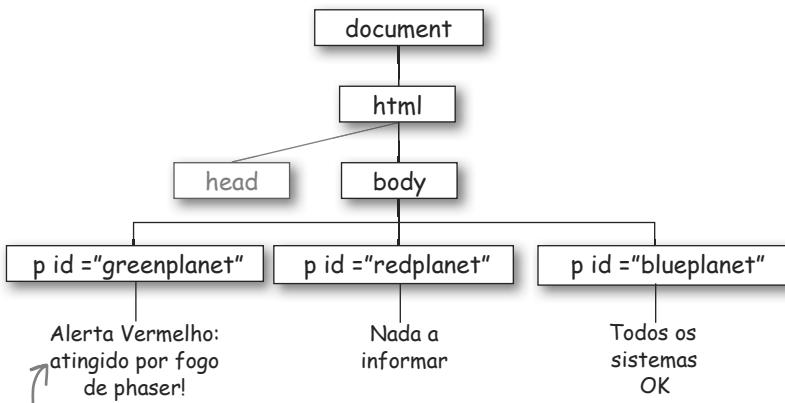
E no nosso código podemos agora simplesmente usar a variável `planet` (`planet`) para nos referir ao nosso elemento.

```
planet.innerHTML = "Red Alert: hit by phaser fire!";
```

Podemos usar a propriedade `innerHTML` do nosso elemento `planet` (`planet`) para alterar o conteúdo do mesmo.

Alteraremos o conteúdo do elemento `greenplanet` para o nosso novo texto... o que resulta no DOM (e sua página) sendo atualizado com o novo texto.

Falaremos mais sobre propriedades de elementos em breve...



Quaisquer alterações no Dom são refletidas na renderização da página do navegador, de modo que você verá o parágrafo mudar para o novo conteúdo!

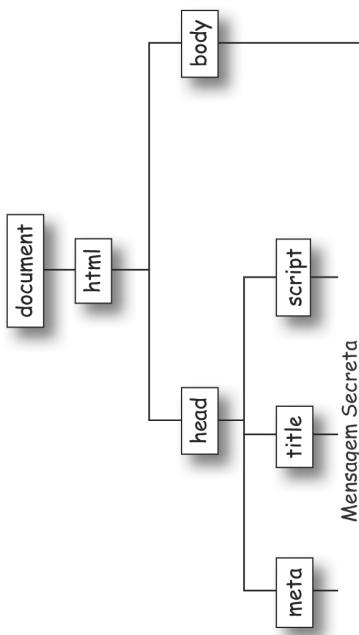
Aponte o seu lápis



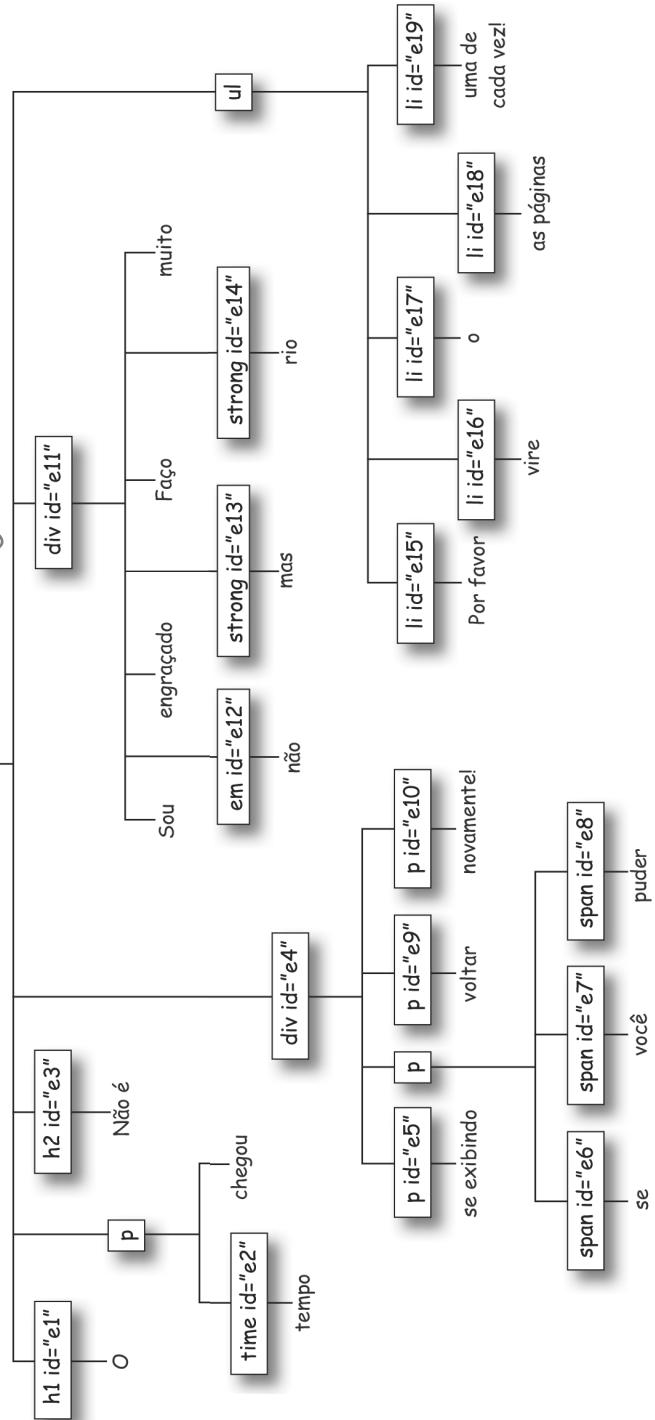
Aqui está um DOM com uma mensagem secreta escondida. Analise o código abaixo para revelar o segredo! A resposta está de cabeça para baixo nesta página.

```
document.getElementById("e7")
document.getElementById("e8")
document.getElementById("e16")
document.getElementById("e9")
document.getElementById("e18")
document.getElementById("e13")
document.getElementById("e12")
document.getElementById("e2")
```

Escreva o elemento que cada linha de código seleciona, assim como o conteúdo do elemento, para revelar a mensagem secreta!



Mensagem Secreta



Resposta: "Você pode voltar novamente as páginas mas não o tempo"

Faça um test drive nos planetas



Você viu como usar `document.getElementById` para obter acesso a um elemento e como usar `innerHTML` para alterar o conteúdo desse elemento. Agora faremos isso pra valer.

Aqui está o HTML dos planetas; temos um elemento `<script>` no cabeçalho, no qual colocaremos o código e três parágrafos para os planetas verde, vermelho e azul. Se você ainda não fez, siga em frente e digite o HTML e JavaScript para atualizar o DOM:

```
<!doctype html>
<html lang="en">
<head>
  <title>Planets</title>
  <meta charset="utf-8">
  <script>
    var planet = document.getElementById("greenplanet");
    planet.innerHTML = "Red alert: hit by phaser fire!";
  </script>
</head>
<body>
  <h1>Green Planet</h1>
  <p id="greenplanet">all is well</p>
  <h1>Red Planet</h1>
  <p id="redplanet">Nothing to report</p>
  <h1>Blue Planet</h1>
  <p id="blueplanet">all systems a-Ok</p>
</body>
</html>
```

Adicionamos o JavaScript no cabeçalho da página.

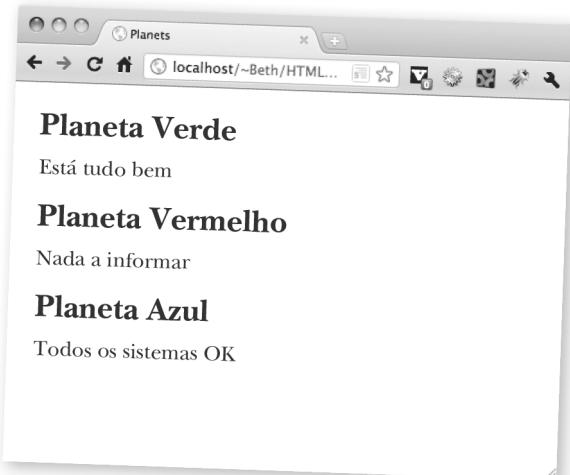
Assim como você fez antes, estamos obtendo o elemento `<p>` com a id "greenplanet" e alterando seu conteúdo.

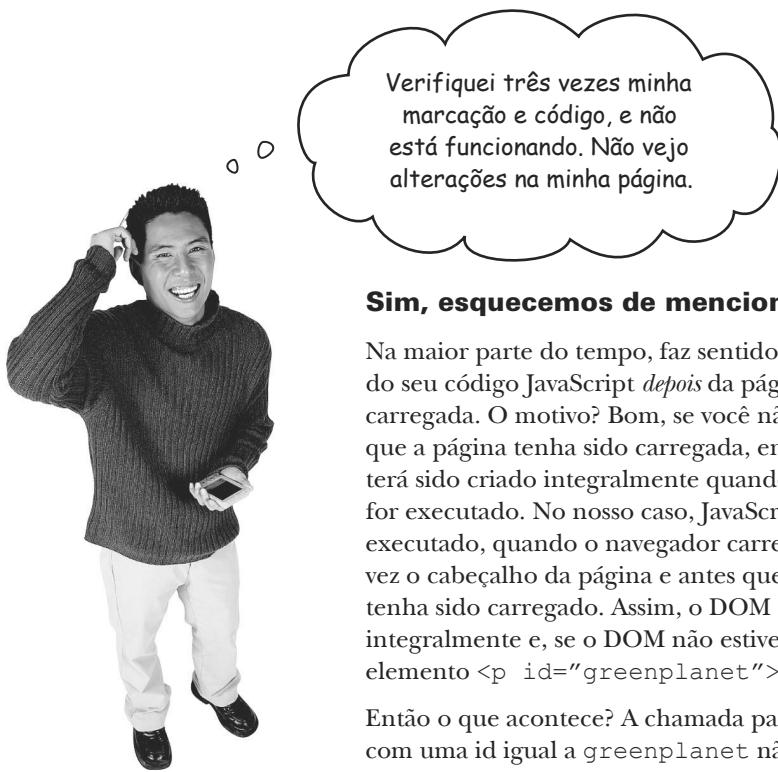
Aqui está o elemento `<p>` que você alterará com JavaScript.



Após você ter digitado isso, siga em frente e carregue a página no seu navegador e veja a mágica DOM acontecer no planeta verde.

OPA! Houston, temos um problema, o planeta verde ainda mostra "Está tudo bem". O que está errado?





Verifiquei três vezes minha marcação e código, e não está funcionando. Não vejo alterações na minha página.

Sim, esquecemos de mencionar uma coisa.

Na maior parte do tempo, faz sentido iniciar a execução do seu código JavaScript *depois* da página estar totalmente carregada. O motivo? Bom, se você não esperar até que a página tenha sido carregada, então o DOM não terá sido criado integralmente quando seu código for executado. No nosso caso, JavaScript está sendo executado, quando o navegador carrega pela primeira vez o cabeçalho da página e antes que o resto da página tenha sido carregado. Assim, o DOM ainda não foi criado integralmente e, se o DOM não estiver criado, então o elemento `<p id="greenplanet">` ainda não existe!

Então o que acontece? A chamada para obter o elemento com uma id igual a `greenplanet` não retornará nada porque não há elemento correspondente, e então o navegador segue em frente e renderiza a página após seu código ter sido executado. Assim, você verá a página renderizada, mas o texto no planeta verde não será alterado pelo código.

O que precisamos é de uma forma para fazer com que o navegador “execute o código após ter carregado totalmente a página e criado o DOM”. Veremos como fazer isso a seguir.

Você não pode mexer com o DOM até a página estar inteiramente carregada

Como fazer para que o navegador execute seu código apenas *após* ser carregado?

Para fazer com que o navegador espere antes de executar código, usaremos duas partes de JavaScript que você ainda não viu muito: o objeto `window` e uma função. Chegaremos nos detalhes de ambos posteriormente, mas siga conosco para ver o código trabalhar.

Atualize seu código JavaScript desta forma:

```
<script>
  function init() {
    var planet = document.getElementById("greenplanet");
    planet.innerHTML = "Red alert: hit by phaser fire!";
  }
<script>
  window.onload = init;
```

Primeiro, crie uma função chamada `init` e coloque nela seu código já existente.

Perceba que seu código fica entre as chaves de abertura { e fechamento}.

Aqui, estamos configurando o valor da propriedade `window.onload` com o nome da função.

Isto diz que, quando a página estiver completamente carregada, execute o código que está em `init`.

Recarregue a página

Siga em frente, recarregue a página e veja se você tem a resposta:



OK! Agora vemos o novo conteúdo no elemento `<p>` do planeta verde. Não é ótimo?

Bom, o que É ótimo é que agora você sabe como fazer com que o navegador espere até que o DOM tenha sido completamente carregado antes de executar o código que acessa elementos.

Aponte o seu lápis

← Aqui está o HTML para a página.

```
<!doctype html>
<html lang="en">
```

<head>

```
    <title>My Playlist</title>
```

```
    <meta charset="utf-8">
```

<script>

```
        addSongs() {
```

```
            var song1 = document._____ ("_____");
```

```
            var _____ = _____ ("_____");
```

```
            var _____ = _____ .getElementById("_____");
```

Aqui está nosso script. Este código deve preencher a lista de músicas abaixo, no .

Preencha os espaços com o código que falta para completar a lista de execução.

```
            _____ .innerHTML = "Blue Suede Strings, by Elvis Pagely";
```

```
            _____ = "Great Objects on Fire, by Jerry JSON Lewis";
```

```
            song3._____ = "I Code the Line, by Johnny JavaScript";
```

}

```
        window._____ = _____;
```

</script>

</head>

<body>

```
    <h1>My awesome playlist</h1>
```

```
    <ul id="playlist">
```

```
        <li id="song1"></li>
```

```
        <li id="song2"></li>
```

```
        <li id="song3"></li>
```


</body>

</html>

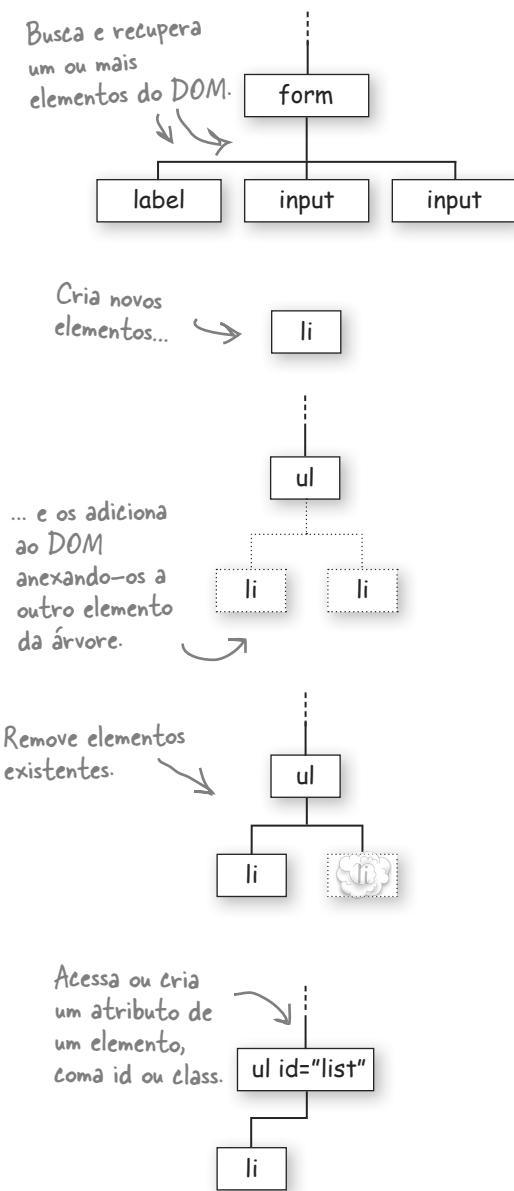
Aqui está a lista vazia das músicas. O código acima deve adicionar conteúdo a cada da lista de execução.

Quando você fizer JavaScript funcionar, é assim que a página web se parecerá após tê-la carregado.



Então, em que mais o DOM é bom, afinal?

O DOM pode fazer mais do que vimos até aqui e usaremos muitas das suas outras funcionalidades, à medida em que seguiremos pelo livro, mas, por enquanto, o examinaremos brevemente de modo que você não se esqueça dele:



Obtenha elementos do DOM.

É claro que você já sabe isso, porque já usamos `document.getElementById`, mas também há outras formas de obter elementos; na verdade, você pode usar nomes de tags, nomes de classes e atributos para recuperar não apenas um elemento, mas um conjunto inteiro deles (por exemplo todos os elementos da classe "em_promoção"). Você pode obter ainda valores de formulários que o usuário digitou, como o texto de um elemento de entrada.

Crie ou adicione elementos ao DOM.

Você pode criar novos elementos e também pode adicioná-los ao DOM. É claro que quaisquer alterações que você fizer no DOM aparecerão imediatamente, quando ele for renderizado pelo navegador (o que é uma coisa boa!).

Remova elementos do DOM.

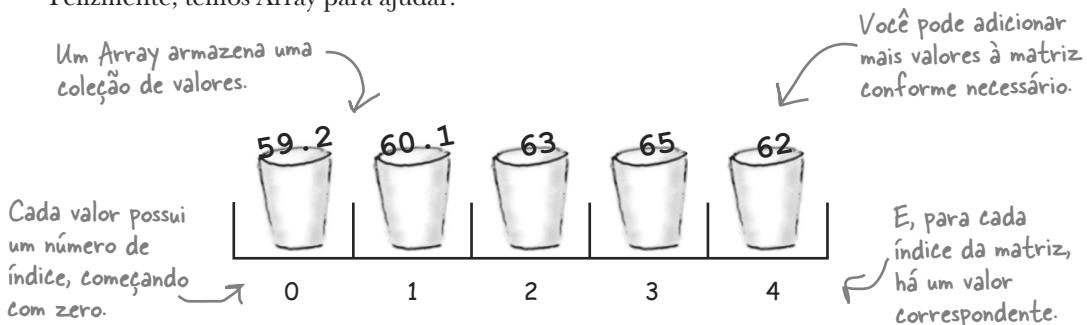
Você também pode remover elementos do DOM, pegando um elemento pai e removendo algum dos seus filhos. Assim, você verá o elemento removido na janela do seu navegador, quando ele for apagado do DOM.

Obtenha e configure atributos de elementos.

Até aqui, você acessou apenas o conteúdo de texto de um elemento, mas pode acessar atributos também. Por exemplo, você poderia saber qual a classe a que um elemento pertence e depois alterá-la durante a execução.

Podemos conversar sobre JavaScript novamente? Ou, como armazenar múltiplos valores em JavaScript

Você está em contato com JavaScript e DOM e, antes de deixá-lo descansar, queríamos lhe contar sobre mais um tipo de JavaScript, que você usará o tempo todo — o Array. Digamos que você quisesse armazenar os nomes de 32 sabores de sorvete, ou os números de todos os itens do carrinho de compras do seu usuário, ou talvez a temperatura externa a cada hora. Fazer isso com variáveis ficaria inadequado, especialmente se você precisar armazenar dezenas, centenas ou milhares de valores. Felizmente, temos Array para ajudar.



Como criar uma matriz

Precisamos criar uma matriz antes de usá-la e atribuir a própria matriz a uma variável, de modo que tenhamos algo para nos referirmos a ela no nosso código. Criaremos a matriz acima com temperaturas horárias:

Aqui está a nossa variável para a matriz...

```
var tempByHour = new array();
tempByHour[0] = 59.2;
tempByHour[1] = 60.1;
tempByHour[2] = 63;
tempByHour[3] = 65;
tempByHour[4] = 62;
```

... e aqui está como realmente criamos uma nova matriz vazia.

Voltaremos a esta sintaxe no Capítulo 4, mas, por enquanto, saiba apenas que ela cria uma nova matriz.

O índice

Para adicionar novos valores à matriz, simplesmente referenciamos o número do índice do item da matriz e damos um valor a ele.

Da mesma forma que uma variável em JavaScript, você pode atribuir qualquer valor (ou tipo de valor) a um índice de matriz

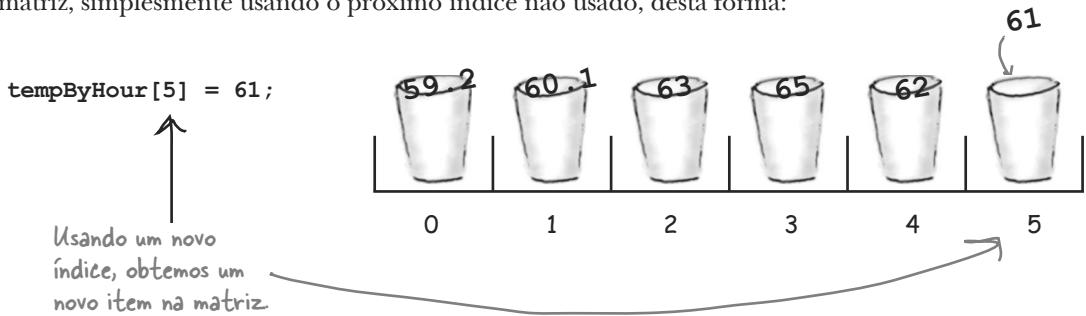
Ou, se você estiver realmente com pressa, JavaScript lhe dá um atalho para digitar uma matriz (o que chamamos de “matriz literal”), para criá-la e inicializá-la com valores:

```
var tempByHour = [59.2, 60.1, 63, 65, 62];
```

Isto cria a mesma matriz acima, mas com muito menos código.

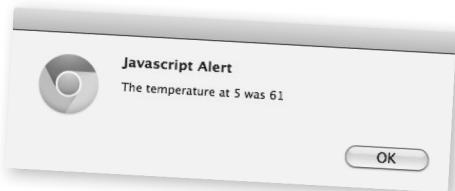
Adicionando outro item à matriz

A qualquer momento, você pode continuar adicionando novos itens à sua matriz, simplesmente usando o próximo índice não usado, desta forma:



Usando seus itens de matriz

Você pode obter o valor de um item de matriz, referenciando a variável da matriz com um índice, desta forma:



```
var message = "The temperature at 5 was " + tempByHour[5];  
alert(message);
```

↑
Para acessar o valor da temperatura no índice 5, apenas referenciamos a matriz nele.

Saiba o tamanho da sua matriz, senão

Você pode obter facilmente o tamanho da sua matriz, referenciando uma propriedade da mesma chamada `length`:

```
var numItems = tempByHour.length;
```

Falaremos mais sobre propriedades no próximo capítulo; por enquanto, saiba apenas que cada matriz possui sua propriedade `length`, que lhe informa o número de itens na matriz.

Agora que sabemos como obter o comprimento de uma matriz, vejamos se podemos combinar o que você sabe sobre loops com matrizes...

Aponte o seu lápis

Abaixo, você encontrará uma página web com uma lista de itens vazios para seu JavaScript preencher com temperaturas. Já demos a você a maior parte do código; é sua tarefa terminá-lo, de modo que ele grave o conteúdo da temperatura correspondente a partir da matriz (por exemplo, o item com id = "temp0" receberá a temperatura no índice 0 da matriz, e assim por diante). Assim, no item com id = "temp3", se lerá "A temperatura em 3 era 65". Para obter créditos extras, veja se descobre como fazer com que a temperatura em 0 mostre "meio-dia" em vez de 0.

```

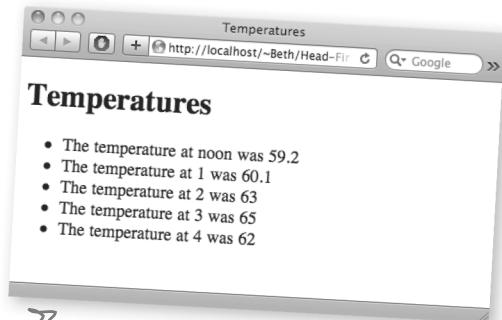
<!doctype html>
<html lang="en">
<head>
<title>Temperatures</title>
<meta charset="utf-8">
<script>
function showTemps() {
    var tempByHour = new _____;
    tempByHour[0] = 59.2;
    tempByHour[1] = 60.1;
    tempByHour[2] = 63;
    tempByHour[3] = 65;
    tempByHour[4] = 62;
    for (var i = 0; i < _____; ____) {
        var theTemp = _____[i];
        var id = "_____ " + i;
        var li = document._____ (id);
        if (i == ____ ) {
            li._____ = "The temperature at noon was " + theTemp;
        } else {
            li.innerHTML = "The temperature at " + _____ + " was " + _____;
        }
    }
    window.onload = showTemps;
}
</script>
</head>
<body>
<h1>Temperatures</h1>
<ul>
    <li id="temp0"></li>
    <li id="temp1"></li>
    <li id="temp2"></li>
    <li id="temp3"></li>
    <li id="temp4"></li>
</ul>
</body>
</html>

```

← Aqui está o HTML

Aqui é onde estamos combinando loops e matrizes. Você consegue ver como estamos acessando cada item da matriz usando um índice de variável?

O código acima preencherá cada item da lista com uma expressão com a temperatura.



exemplo phrase-o-matic

Analise este código para o novo aplicativo Phrase-o-Matic e veja se consegue descobrir o que ele faz antes de seguir em frente...

Experimente o meu novo Phrase-o-Matic e você se tornará habilidoso para conversar, como o seu chefe ou aqueles caras do marketing.



Você não imaginava que nossa aplicação para negócios sérios do Capítulo 1 fosse séria o suficiente? Bom. Tente esta, se você precisa mostrar algo para seu chefe.

```
<!doctype html>
<html lang="en">
<head>
    <title>Phrase-o-matic</title>
<meta charset="utf-8">
<style>
body {
    font-family: verdana, Helvetica, sans-serif;
}
</style>
<script>
function makePhrases() {
    var words1 = ["24/7", "multi-Tier", "30,000 foot", "B-to-B", "win-win"];
    var words2 = ["empowered", "value-added", "oriented", "focused", "aligned"];
    var words3 = ["process", "solution", "tipping-point", "strategy", "vision"];

    var rand1 = Math.floor(Math.random() * words1.length);
    var rand2 = Math.floor(Math.random() * words2.length);
    var rand3 = Math.floor(Math.random() * words3.length);

    var phrase = words1[rand1] + " " + words2[rand2] + " " + words3[rand3];
    var phraseElement = document.getElementById("phrase");
    phraseElement.innerHTML = phrase;
}
window.onload = makePhrases;
</script>
</head>
<body>
    <h1>Phrase-o-Matic says:</h1>
    <p id="phrase"></p>
</body>
</html>
```

O Phrase-o-Matic

Esperamos que você tenha descoberto que este código é a ferramenta perfeita para criar seu slogan de marketing. Ela criou vencedores como “Solução de valor agregado em que todos ganham” e “Processo habilitado 24 horas por dia 7 dias por semana” e temos muita esperança de que mais vencedores surjam no futuro. Vejamos como isto realmente funciona:

- Em primeiro lugar, definimos a função makePhrases, que podemos executar após a página ter sido completamente carregada, de modo a garantir o acesso ao DOM com segurança:

```
function makePhrases() {  
}  
window.onload = makePhrases;
```

Estamos definindo uma função chamada makePhrases, que poderemos chamar posteriormente.

Todo este código de makePhrases vai aqui, chegaremos lá em um segundo...

Executamos makePhrases assim que a página termina de ser carregada.

- Com isso pronto, podemos escrever o código da função makePhrases. Começaremos configurando três matrizes. Cada uma armazenará palavras que usaremos para criar as expressões.

Criamos uma variável chamada words1, que podemos usar para referenciar a primeira matriz

```
var words1 = ["24/7", "multi-Tier", "30,000 foot", "B-to-B", "win-win"];
```

Estamos colocando cinco strings na matriz. Sinta-se livre para alterá-las com as palavras que estão mais na moda.

```
var words2 = ["empowered", "value-added", "oriented", "focused", "aligned"];  
var words3 = ["process", "solution", "tipping-point", "strategy", "vision"];
```

E há mais duas matrizes de palavras, atribuídas a duas novas variáveis, words2 e words3.

como funciona phrase-o-matic

- 3 OK, temos três matrizes novas com palavras da moda; o que faremos agora é escolher aleatoriamente uma palavra de cada e depois juntá-las para criar uma expressão.

Aqui está como selecionamos uma palavra de cada matriz:

```
Geramos um número aleatório para cada matriz e o atribuímos  
a uma nova variável (rand1, rand2 e rand3, respectivamente).  
  
var rand1 = Math.floor(Math.random() * words1.length);  
var rand2 = Math.floor(Math.random() * words2.length);  
var rand3 = Math.floor(Math.random() * words3.length);
```

Este código gera um número aleatório baseado no número de itens em cada matriz (no nosso caso, cinco, mas sinta-se livre para adicionar mais a qualquer matriz que ainda vai funcionar).

- 4 Agora criamos a expressão de marketing, pegando palavras selecionadas aleatoriamente e concatenando-as, com um espaço entre elas por motivo de legibilidade:

Definimos outra variável para armazenar a expressão.

Usamos cada número aleatório para indexar as matrizes de palavras...

```
var phrase = words1[rand1] + " " + words2[rand2] + " " + words3[rand3];
```

- 5 Estamos quase terminando. Temos a expressão, agora só precisamos exibi-la. Neste ponto, você já sabe: usaremos getElementById para localizar nosso elemento de parágrafo e depois usaremos sua innerHTML para colocar a nova expressão lá.

```
var phraseElement = document.getElementById("phrase");  
phraseElement.innerHTML = phrase;
```

Obtemos o elemento <p> com a id "phrase".

E depois configuramos o conteúdo do elemento <p> com a expressão.

- 6** OK, termine essa última linha de código, dê mais uma olhada geral e sinta a sensação de dever cumprido antes de carregá-la no seu navegador. Faça um test drive e aproveite as expressões.

Aqui está como o
nossa se parece!
→



↑ Recarregue a página para infinitas possibilidades (OK, não infinitas, mas coopere conosco, estamos tentando tornar interessante este código simples!).

Perguntas Idiotas

P: O que exatamente é Math, e o que Math.random e Math.floor fazem?

R: Math é uma biblioteca interna de JavaScript que possui funções relacionadas à matemática. Math.random gera um número aleatório entre 0 e 1. Multiplicamos isso pelo número de itens da matriz (que obtemos usando a propriedade length da mesma) para obter um número entre 0 e o comprimento da matriz. O resultado provavelmente será um número de ponto flutuante, como 3.2, de modo que usamos Math.floor para assegurarmos a obtenção de um número inteiro que possamos usar como índice na matriz para selecionarmos a palavra aleatória. Tudo que Math.floor faz é eliminar os números depois do ponto decimal em um número de ponto flutuante. Por exemplo, Math.floor(3.2) é 3.

P: Onde eu posso encontrar documentação sobre coisas como Math?

R: Uma ótima referência sobre JavaScript é *JavaScript: The Definitive Guide*, de David Flanagan (O'Reilly).

P: Anteriormente, você disse que pode armazenar primitivas (como números, strings e booleanos) em variáveis ou objetos. Estamos, porém, armazenando matrizes em variáveis. Então, o que é uma matriz, uma primitiva ou um objeto?

R: Boa pergunta! Uma matriz é um tipo especial de objeto interno de JavaScript. É especial porque você pode usar índices numéricos para acessar os valores armazenados na matriz, algo que não pode fazer com outros objetos que não sejam matrizes, ou objetos que você mesmo tenha criado. Você aprenderá a criar seus próprios objetos no Capítulo 4.

P: O que acontece se eu tentar acessar um índice de matriz que não existe? Como se eu tivesse 5 palavras armazenadas em minhasPalavras e tentasse acessar minhasPalavras[10].

R: Você fica indefinido, o que, se você se lembra, é o valor de uma variável que ainda não tenha recebido um valor.

P: Eu posso remover um item de uma Array? Caso positivo, o que acontece com o índice dos outros elementos?

R: Você pode remover um item de uma Array, e pode fazê-lo de algumas formas diferentes. Poderia configurar o valor da matriz no índice como nulo; por exemplo minhaMatriz[2] = null. O comprimento da Array permaneceria o mesmo. Você pode também remover completamente o item (usando a função splice). Neste caso, os índices dos itens que vierem depois daquele que você remover, diminuirão em 1. Assim, se minhaMatriz[2] = "cachorro" e minhaMatriz[3] = "gato", se você remover "cachorro", então minhaMatriz[2] = "gato" e o comprimento da sua matriz é 1 a menos do que era antes da remoção.

Aprender uma linguagem é trabalho duro e requer que você não apenas trabalhe com seu cérebro, mas que também o descance. Assim, após este capítulo, descanse um pouco, coma alguma coisa por nossa conta, mas, antes de ir, verifique os Pontos de Bala e faça as palavras cruzadas para realmente fixar tudo.

O
O



Não descobrimos
como converter
de digital para
análogo ainda,
então você precisará
fornecer suas
próprias refeições.



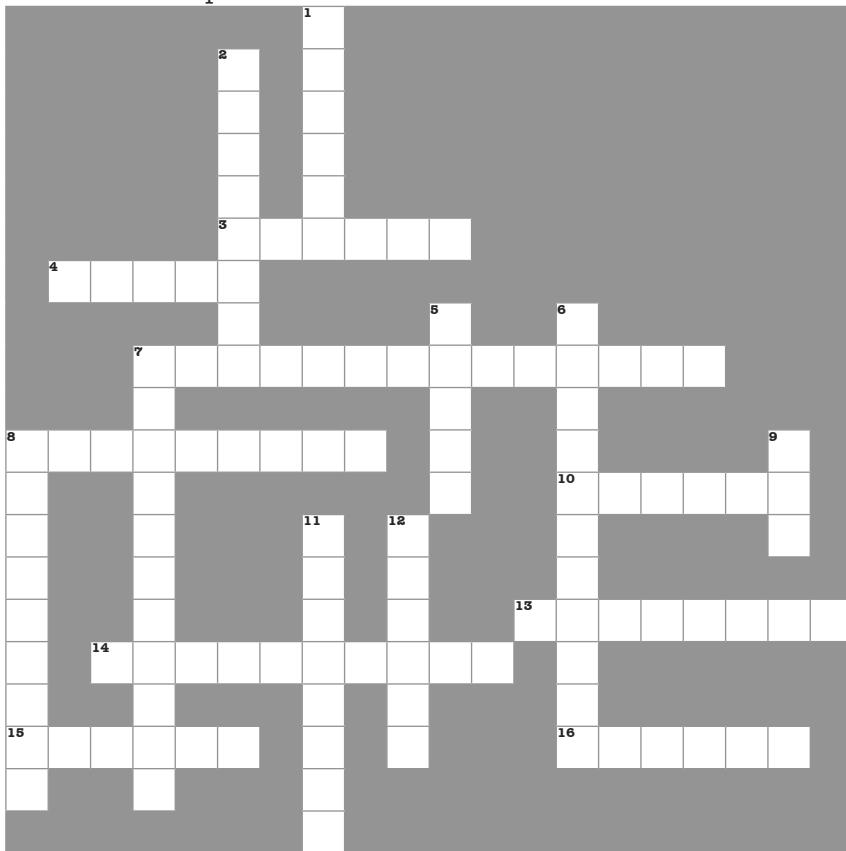
PONTOS DE BALA

- Declare uma variável JavaScript usando var.
- Numéricos, booleanos e strings são tipos primitivos.
- Valores booleanos são verdadeiros ou falsos.
- Números podem ser inteiros ou de ponto flutuante.
- Uma variável não atribuída possui valor indefinido.
- Indefinido e nulo são dois valores diferentes. Indefinido significa que uma variável não teve atribuído a si um valor; nulo significa que a variável não tem valor.
- Expressões numéricas, booleanas e string resultam em um valor numérico, booleano ou de string, respectivamente.
- Para repetir blocos de código, use um loop for ou while.
- Loops for e while fazem a mesma coisa; use o que funcionar melhor para a situação.
- Para terminar um loop for ou while, o teste condicional deve resultar em falso em algum momento.
- Use comandos if/else para tomar uma decisão baseada em um teste condicional.
- Testes condicionais são expressões booleanas.
- Você pode adicionar JavaScript ao cabeçalho ou corpo da sua página web, ou colocá-lo em um arquivo separado e conectá-lo a partir da sua página web.
- Feche seu JavaScript (ou conexão para ele) com o elemento <script>.
- Quando o navegador carrega uma página web, cria um DOM, que é uma representação interna dessa página.
- Você torna sua página web interativa examinando e alterando o DOM usando JavaScript.
- Obtenha acesso a um elemento na sua página web, usando document.getElementById.
- document.getElementById usa a id de um elemento para encontrá-lo no DOM.
- Use a propriedade innerHTML de um elemento para alterar seu conteúdo.
- Se você tentar acessar ou alterar elementos antes que a página web tenha sido integralmente carregada, receberá um erro de JavaScript e seu código não funcionará.
- Atribua uma função à propriedade window.onload para executar o código dessa função após o navegador ter terminado de carregar a página web.
- Use uma matriz para armazenar mais de um valor.
- Para acessar um valor em uma matriz, use um índice. Um índice é um número inteiro que indica a posição do item na matriz (começando em 0).
- A propriedade length de uma matriz lhe informa quantos itens há nela.
- Combinando loops e matrizes, você pode acessar sequencialmente cada item de uma matriz.
- Math é uma biblioteca JavaScript com diversas funções relacionadas à matemática.
- Math.random retorna um número de ponto flutuante entre 0 e 1 (mas nunca precisamente 1).
- Math.floor converte um número de ponto flutuante em um inteiro, eliminando os dígitos após a casa decimal.



Palavras Cruzadas HTML5

Hora de trabalhar uma área diferente do seu cérebro com palavras cruzadas. Divirta-se!



Horizontal

3. Armazene todos os seus sabores de sorvete juntos em uma _____.
4. Execute coisas várias vezes com um loop _____.
7. document._____ é como você obtém um elemento do DOM em JavaScript.
8. Você pode adicionar seu JavaScript ao _____ ou ao corpo do seu HTML.
10. Escolha bons nomes e use _____ para nomes longos.
13. Loops while e for usam uma expressão _____ como teste condicional.
14. Adicione isto para tornar sua página web interativa.
15. Se você estiver quase terminando, beba chá, _____ se não estiver nem próximo disso, continue trabalhando!

16. Você sabe quantos itens há em uma matriz se verificar a propriedade _____.

Verticais

1. Feche seu JavaScript com uma tag <____> se estiver em uma página HTML.
2. _____ é o raiz da árvore DOM.
5. Variáveis começam com uma _____, \$, ou ___.
6. O navegador cria um Document _____ quando carrega a página.
7. a id do planeta atingido pelo fogo do phaser.
8. Se você escrever 3 + "Patetas", JavaScript irá _____ 3 em uma string.
9. Não mexa no _____ até que a página tenha sido totalmente carregada.
11. O DOM é uma representação interna da _____.
12. Use um _____ para obter um valor de uma matriz.



Exercício Solução

Expresse-se!

Você viu os diferentes tipos de expressões que pode usar em JavaScript. Agora é hora de colocar esse conhecimento para trabalhar, avaliando você mesmo algumas expressões. Aqui está nossa solução.

`(9 / 5) * tempC + 32`

Qual o resultado, quando tempC for 10? 50

`"Number" + " " + "2"`

Qual a string resultante? Número 2

`level >= 5`

Qual o resultado, quando level (nível) for 10? true “^{maior ou} igual a”
E quando for 5? true “igual a”

`color != "pink"`

Qual o resultado, se color (cor) for “blue” (“azul”)? true cor “não é igual” a rosa

`(2 * Math.PI) * r`

Qual o resultado, se r for 3? 18.84 aproximadamente!

Math.PI lhe dá o valor de pi (você sabe, 3.14...)



Não este tipo de expressão

Aponte o seu lápis

Baseado no que você sabe até agora sobre variáveis, expressões e comandos em JavaScript, veja se consegue descobrir qual destas são legais e quais poderiam gerar um erro.

Da lista a seguir, circule os comandos que são **legais**.

`var x = 1138;`

`var y = 3/8;`

`var s = "3-8";`

`x = y;`

`var n = 3 - "one";`

→ Tecnicamente, isto é legal, mas resulta em um valor que você não pode usar.

`var t = "one" + "two";`

`var 3po = true; ilegal!`

`var level_ = 11;`

`var highNoon = false;`

`var $ = 21.30;`

`var z = 2000;`

`var isBig = y > z;`

`z = z + 1;`

`z--;`

`z y; ilegal!`

`x = z * t;`

`while (highNoon) {`

`z--;`

`}`



Sinta-se como o Navegador – Solução

Cada um dos trechos em JavaScript nesta página são códigos separados. Sua tarefa é dar uma de navegador, avaliar cada trecho de código e responder a uma pergunta sobre o resultado. Escreva suas respostas abaixo do código.

Trecho 1

Trecho 1

```
var count = 0;
for (var i = 0; i < 5; i++)
{
    count = count + i;
}
alert("count is " + count);
```

Que count o alerta apresenta?

10

Cada vez que executamos o loop, adicionamos o valor de i ao contador, e i está aumentando, de modo que não estamos apenas adicionando 1 ao contador a cada vez, mas sim 0, 1, 2, 3 e 4.

Trecho 2

```
var tops = 5;
while (tops > 0) {
    for (var spins = 0; spins < 3; spins++) {
        alert("Top is spinning!");
    }
    tops = tops - 1;
}
```

15

Quanto tempo você vê o alerta: "Top is spinning!"?

Aqui, estamos começando em 5 e executando o loop até que o número de frutas seja 0, diminuindo a cada vez (em vez de aumentar).

O loop while externo é executado 5 vezes e o loop interno for 3 vezes cada vez que o externo é executado, de modo que o total é $5 * 3$, ou seja, 15!

Trecho 3

```
for (var berries = 5; berries > 0; berries--) {
    alert("Eating a berry");
}
```

Quantos berries você comeu?

5

Trecho 4

```
for (scoops = 0; scoops < 10; scoop++) {
    alert("There's more ice cream!");
}
alert("life without ice cream isn't the same");
```

Uma fácil; apenas executamos o loop 10 vezes de modo que comemos 10 bolas!

10 Quantas bolas de sorvete você comeu?



Pegue o código anterior e insira-o no loop while abaixo. Percorra o loop while e escreva os alertas na sequência em que ocorrem. Aqui está a nossa solução.

```

var scoops = 10;

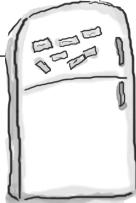
while (scoops >= 0) {
    if (scoops == 3) { Código inserido
        alert("Ice cream is running low!"); ← Isto acontece uma vez,
    } else if (scoops > 9) { ← Isto também
        alert("Eat faster, the ice cream is going to melt!"); ← acontece uma
    } else if (scoops == 2) { ← vez, quando scoops (bolas de
        alert("Going once!"); ← sorvete) vale 10.
    } else if (scoops == 1) { ← Cada um destes acontece
        alert("Going twice!"); ← uma vez, quando o número
    } else if (scoops == 0) { ← de scoops for 2, 1 e 0.
        alert("Gone!");
    } else {
        alert("Still lots of ice cream left, come and get it.");
    }

    scoops = scoops - 1; ← Subtraímos 1 scoops cada vez que o loop é executado.
}

alert("Life without ice cream isn't the same."); ← Isto é executado após
                                                o loop terminar.

Os alertas: → Coma mais rápido, o sorvete vai derreter!
Ainda há muito sorvete, venha pegá-lo.
O sorvete está acabando!
Dou-lhe uma!
Dou-lhe duas!
Terminou!
A vida sem sorvete não é a mesma coisa.

```



Ímãs de Geladeira – Solução

Este código imprime um palíndromo em um alerta. O problema é que uma parte do código estava em ímãs de geladeira e caiu no chão. É sua tarefa juntar esse código novamente para que o palíndromo funcione. Cuidado: já havia alguns ímãs no chão que não fazem parte do código e você terá que usar alguns mais de uma vez! Aqui está a nossa solução.

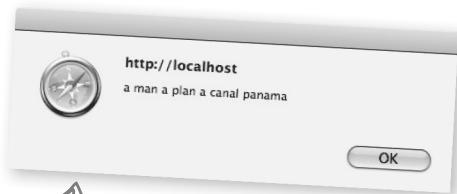
```

var word1 = "a";
var word2 = "nam";
var word3 = "nal p";
var word4 = "lan a c";
var word5 = "a man a p";

var phrase = "";

for (var i = 0; i < 4; i++)
    if (i == 0) {
        phrase = word5;
    }
    else if (i == 1) {
        phrase = phrase + word4;
    }
    else if (i == 2) {
        phrase = phrase + word1 + word3;
    }
    else if (i == 3) {
        phrase = phrase + word1 + word2 + word1;
    }
}
alert(phrase);

```

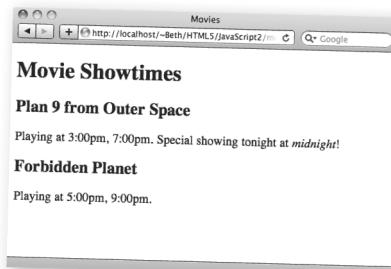


Um palíndromo é uma frase que pode ser lida da mesma forma de trás para frente ou de frente para trás! Aqui está o palíndromo que você deve ver, se os ímãs estiverem todos nos lugares certos.

else if (i == 0)

↳ Ímãs que sobraram

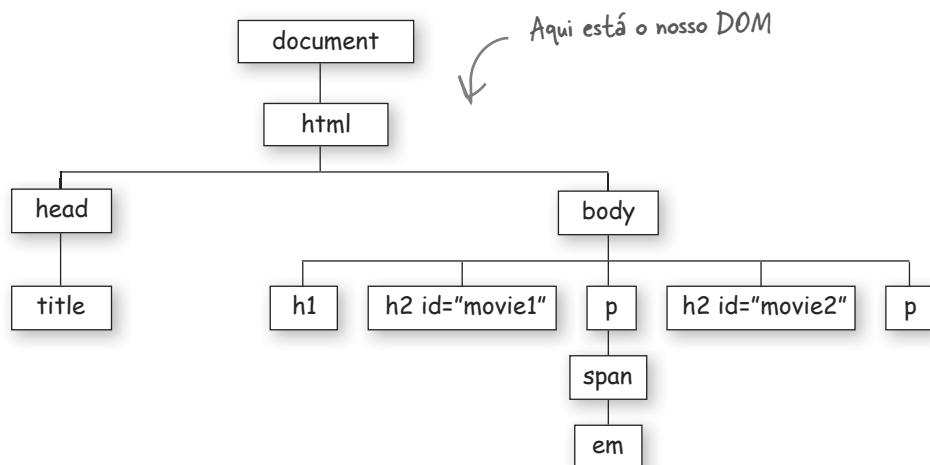
i == 4 word2 word4 i < 3
 else word0 i = 0 + i-- i = 3 word3



Sinta-se como o Navegador – Solução

Sua tarefa é agir como se fosse o navegador. Você precisa analisar o HTML e criar seu próprio DOM a partir dele. Siga em frente e analise o HTML à direita, e desenhe seu DOM abaixo. Já começamos para você.

```
<!doctype html>
<html lang="en">
  <head>
    <title>Movies</title>
  </head>
  <body>
    <h1>Movie Showtimes</h1>
    <h2 id="movie1" >Plan 9 from Outer Space</h2>
    <p>Playing at 3:00pm, 7:00pm.
      <span>
        Special showing tonight at <em>midnight</em>!
      </span>
    </p>
    <h2 id="movie2">Forbidden Planet</h2>
    <p>Playing at 5:00pm, 9:00pm.</p>
  </body>
</html>
```





Aponte o seu lápis Solução

Aqui está o HTML para uma lista de execução de músicas, mas ela está vazia. É sua tarefa completar o JavaScript abaixo para adicionar as músicas à lista. Preencha os espaços com o JavaScript que fará isso. Nossa solução está a seguir.

```
<!doctype html>
<html lang="en">
<head>
  <title>My Playlist</title>
  <meta charset="utf-8">
  <script>
    function addSongs() {
      var song1 = document.getElementById("song1");
      var song2 = document.getElementById("song2");
      var song3 = document.getElementById("song3");

      song1.innerHTML = "Blue Suede Strings, by Elvis Pagely";
      song2.innerHTML = "Great Objects on Fire, by Jerry JSON Lewis";
      song3.innerHTML = "I Code the Line, by Johnny JavaScript";
    }
    window.onload = addSongs;
  </script>
</head>
<body>
  <h1>My awesome playlist</h1>
  <ul id="playlist">
    <li id="song1"></li>
    <li id="song2"></li>
    <li id="song3"></li>
  </ul>
</body>
</html>
```



My awesome playlist

- Blue Suede Strings, by Elvis Pagely
- Great Objects on Fire, by Jerry JSON Lewis
- I Code the Line, by Johnny JavaScript

Se você fizer o JavaScript funcionar, é assim que a página web se parecerá após ser carregada.

Aqui está o código que fará a lista de músicas funcionar.

Sinta-se livre para substituir por suas músicas favoritas

O código acima configura o conteúdo destes elementos `` pegando cada elemento do DOM e configurando `innerHTML` com o nome da música.



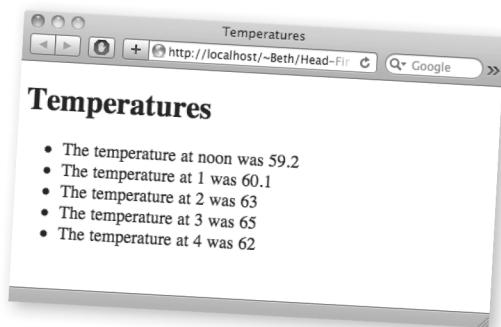
Aponte o seu lápis Solução

```

<!doctype html>
<html lang="en">
<head>
<title>Temperatures</title>
<meta charset="utf-8">
<script>
function showTemps() {
    var tempByHour = new Array();           ↗ Aqui estamos criando uma nova Array
    tempByHour[0] = 59.2;                      para armazenar as temperaturas.
    tempByHour[1] = 60.1;
    tempByHour[2] = 63;
    tempByHour[3] = 65;
    tempByHour[4] = 62;
    for (var i = 0; i < tempByHour.length; i++) { ← Aqui é onde estamos
        var theTemp = tempByHour [i];
        var id = "temp" + i;
        var li = document.getElementById (id);
        if (i == 0) {
            li.innerHTML = "The temperature at noon was " + theTemp;
        } else {
            li.innerHTML = "The temperature at " + i + " was " + theTemp;
        }
    }
    window.onload = showTemps;
}
</script>
</head>
<body>
<h1>Temperatures</h1>
<ul>
    <li id="temp0"></li>
    <li id="temp1"></li>
    <li id="temp2"></li>
    <li id="temp3"></li>
    <li id="temp4"></li>
</ul>
</body>
</html>

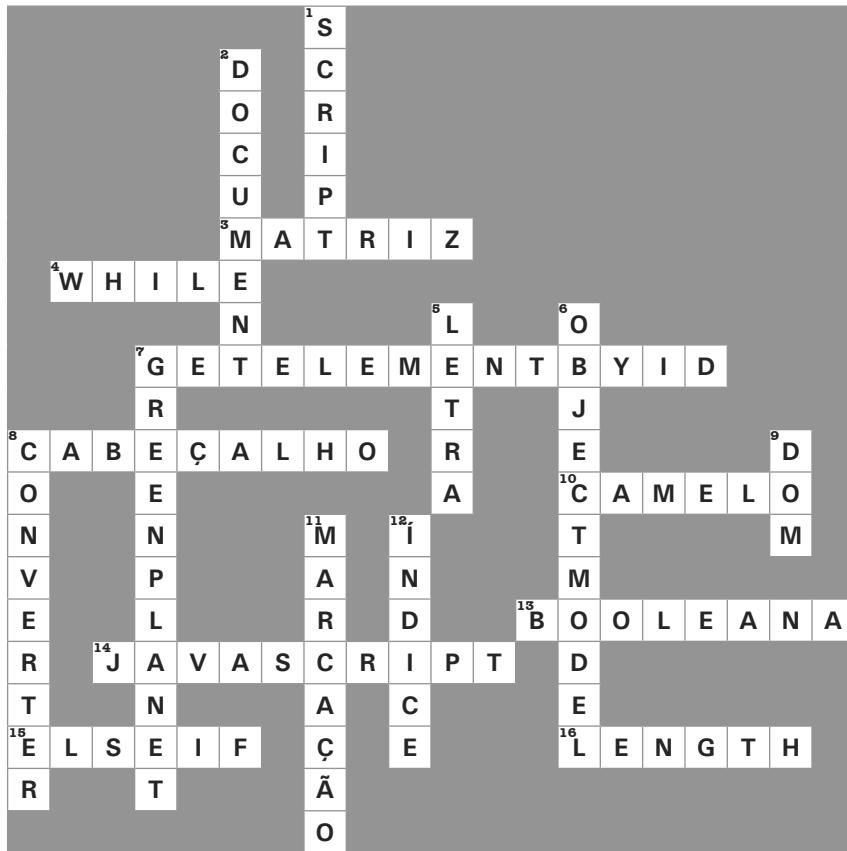
```

→ E nossos resultados!





Solução das Palavras Cruzadas HTML5



3 eventos, manipuladores e todo esse balanço

* Um Pouco de Interação *



Você ainda não alcançou seu usuário. Você aprendeu os conceitos básicos sobre JavaScript, mas será que consegue interação com seus usuários? Quando as páginas respondem a entradas deles, não são mais apenas documentos; são aplicativos vivos e interativos. Neste capítulo, você aprenderá a lidar com uma forma de entrada de usuário e a conectar um elemento `<form>` antigo de HTML a código real. Talvez seja perigoso, mas é poderoso. Aperte seu cinto de segurança. Este é um capítulo que vai diretamente ao ponto, em que vamos de zero a aplicativos interativos em um segundo.

Apronte-se para o Webville Tunes

OK, guiamos você através de muitos conceitos básicos de JavaScript até aqui neste livro e, embora tenhamos falado muito sobre a criação de aplicativos web, não temos muito a mostrar sobre isso ainda. Então agora ficaremos sérios (não de verdade! Queremos dizer apenas desta vez!) e criaremos algo no mundo real.

Que tal gerenciar uma playlist? Vamos chamá-la de algo original, como... hmm, Webville Tunes.



Adiciona novas músicas a qualquer momento.

Exibe todas suas músicas favoritas do Webville Tunes, diretamente no navegador.

O que iremos criar.

↑ Totalmente baseado no navegador. Não há necessidade de código no lado servidor.



Dado o que você sabe sobre este código:

```
window.onload = init;
```

Consegue imaginar o que este código pode fazer?

```
button.onclick = handleButtonClick;
```

Iniciando...

Não precisamos criar uma página web grande e complexa logo de cara. Na verdade, podemos começar de forma muito simples. Criaremos um documento HTML5 com um formulário e um elemento lista para armazenar a lista de músicas:

```

<!doctype html>           Apenas seu cabeçalho e corpo HTML5 padrão.
<html lang="en">
<head>
  <title>Webville Tunes</title>
  <meta charset="utf-8">
  <script src="playlist.js"></script>
  <link rel="stylesheet" href="playlist.css">
</head>                     Todo o que precisamos é um formulário simples. Aqui está ele com um campo
<body>                      de texto para digitar suas músicas. Estamos usando o atributo placeholder
  <form>                      HTML5 que mostra um exemplo do que digitar no campo de entrada.
    <input type="text" id="songTextInput" size="40" placeholder="Song name">
    <input type="button" id="addButton" value="Add Song">
  </form>
  <ul id="playlist">
    </ul>                         Usaremos uma lista para as
</body>                      músicas. Por enquanto ela está
</html>                      vazia, mas alteraremos isso com
                                código JavaScript em um segundo...

```

Colocaremos todo o seu JavaScript no arquivo da lista de músicas.

Incluímos uma folha de estilo para dar ao seu aplicativo de lista de músicas uma ótima aparência *.

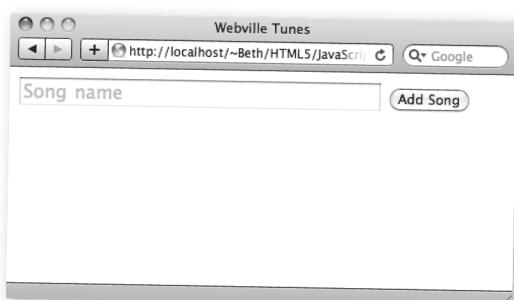
E temos um botão com uma id " addButton" para submeter suas novas adições à lista de músicas.

Faça um Test Drive



Siga em frente e digite o código acima, carregue-o no seu navegador favorito e use-o antes de passar para a próxima página.

Aqui está o que você deverá ver.



*Lembre-se de que você pode baixar a folha de estilo (e todo o código) em <http://wickedlysmart.com/hfhtml5>

Mas nada acontece quando clico em “Adicionar Música”

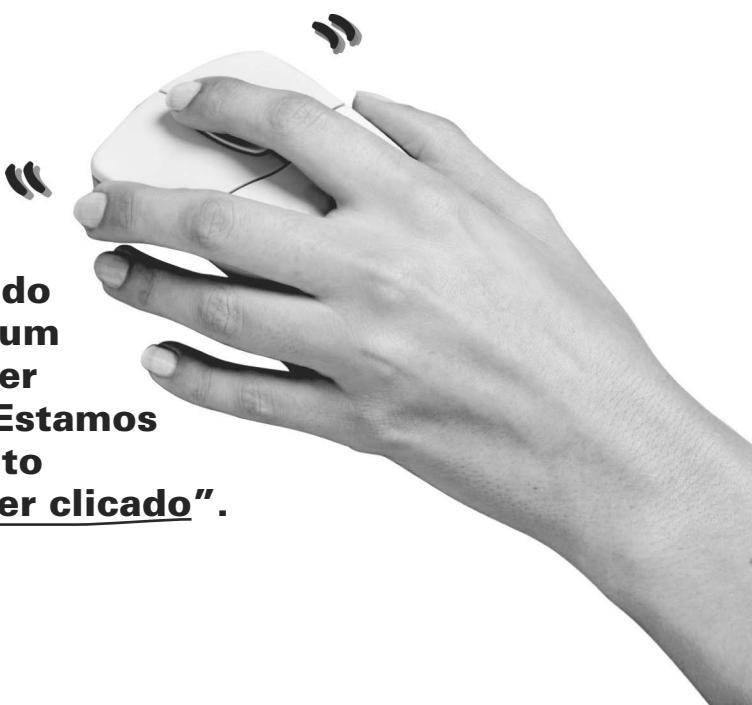
Bom, sim e não. Nada *parece* acontecer, mas o seu navegador sabe que você clicou no botão (dependendo do navegador, você também verá o botão pressionado).

A questão real é como fazemos com que o botão execute alguma coisa quando você clica. O que faremos para que o código em JavaScript seja chamado quando você clica no botão.

Precisamos de duas coisas:

- ① Precisamos de um pouco de código JavaScript que seja avaliado, quando o usuário clicar no botão “Adicionar Música”. Este código (assim que tiver sido escrito) adiciona uma música à sua lista de músicas.
- ② Precisamos de uma forma de conectar esse código, de modo que, quando o botão for clicado, JavaScript saiba executar o seu código de “adição de músicas”.

Quando o usuário clicar (ou tocar em um dispositivo baseado em movimentos) em um botão, queremos saber que isso aconteceu. Estamos interessados no evento “o botão acabou de ser clicado”.





Lidando com Eventos

Você verá que muitas coisas estão acontecendo no navegador enquanto sua página está sendo exibida — botões estão sendo clicados, dados adicionais que seu código solicitou da rede podem estar chegando, temporizadores podem estar sendo disparados (chegaremos em tudo isso). Todas estas coisas fazem com que eventos aconteçam: um evento de clique em botão, um evento de dados disponíveis, um evento de tempo expirado e assim por diante (há muitos mais).

Sempre que houver um evento, há uma oportunidade para seu código *manipulá-lo*, ou seja, fornecer algum código que será chamado quando o evento ocorrer. Você não precisa manipular qualquer um destes eventos, mas precisará fazê-lo se quiser que coisas interessantes aconteçam quando eles ocorrerem — como, por exemplo, quando o evento clique no botão acontecer, adicionar uma nova música à lista de execução; quando novos dados chegarem, processá-los e exibi-los na sua página; quando um temporizador disparar, informar ao usuário que seus ingressos para a primeira fila irão expirar, e assim por diante.

Assim, vejamos o que fazer para lidar com o evento de clique no botão.

Criando um Plano...

Vamos voltar um momento, antes de nos perdermos em manipuladores e eventos. O objetivo aqui é clicar em “Adicionar Música” e fazer com que uma música seja adicionada à uma lista de execução na página. Abordaremos esta tarefa desta maneira:

1. **Configurar um manipulador para lidar com um clique de usuário no botão “Adicionar música”.**
2. **Escrever o manipulador para obter o nome da música que o usuário digitou e depois...**
3. **Criar um novo elemento para armazenar a nova música e...**
4. **Adicionar o elemento ao DOM da página.**

Se estes passos não estiverem claros para você, não se preocupe, explicaremos à medida em que seguirmos em frente... Por enquanto, apenas acostume-se com os passos e siga-os enquanto escrevemos esse manipulador. Vá em frente e abra um novo arquivo `playlist.js` para todo o seu código em JavaScript.

Obtendo acesso ao botão “Adicionar Música”

Para solicitar ao botão que nos informe quando um evento ocorrer, primeiro precisamos obter acesso a ele. Felizmente, criamos o botão usando marcação HTML e isso significa... Você acertou. Ele está representado no DOM e você já sabe como obter elementos do DOM. Se você olhar mais atrás no HTML, verá que demos ao botão uma id de `addButton`. Assim, usaremos `getElementById` para obtermos uma referência a ele:

```
var button = document.getElementById(" addButton");
```

Agora só precisamos dar ao botão um código para chamar quando ocorrer um clique. Para fazer isso, criaremos uma função, chamada `handleButtonClick`, que lidará com o evento. Chegaremos em funções em breve; por enquanto, aqui está a função:

```
A função é chamada de handleButtonClick; chegaremos às especificidades da sintaxe daqui a pouco.  
function handleButtonClick() {  
    alert("Button was clicked!");  
}
```

Uma função lhe dá uma forma de empacotar código. Você pode dar a ela um nome e reusar o código onde quiser.

Colocamos todo o código que queremos executar quando a função for chamada dentro das chaves.

Agora só exibiremos um alerta quando esta função for chamada.

Dando um manipulador de clique para o botão

OK, temos um botão e uma função que atuará como manipulador, `handleButtonClick`, então vamos juntar tudo. Para fazer isso, usaremos uma propriedade do botão, `onclick`. Configuramos a propriedade `onclick` desta forma:

```
var button = document.getElementById("addButton");
button.onclick = handleButtonClick;
```

Com um botão disponível, após chamar `getElementById`, configuramos a propriedade `onclick` na função que queremos que seja chamada quando ocorrer um clique no botão.

Você talvez lembre que fizemos algo semelhante quando usamos a propriedade `window.onload` para chamar uma função após a janela ter sido carregada. Neste caso, porém, chamaremos a função quando o botão for clicado. Agora juntaremos tudo:

```
window.onload = init;
function init() {
    var button = document.getElementById("addButton");
    button.onclick = handleButtonClick;
}
function handleButtonClick() {
    alert("Button was clicked!");
}
```

Da mesma forma que fizemos no capítulo anterior, estamos usando uma função `init` que não será chamada e executada até que a página tenha sido totalmente carregada.

Após a página ser carregada, pegaremos o botão e configuraremos seu manipulador `onclick`.

E o manipulador de clique exibirá um alerta quando clicarmos no botão.

Colocando isso em um teste...

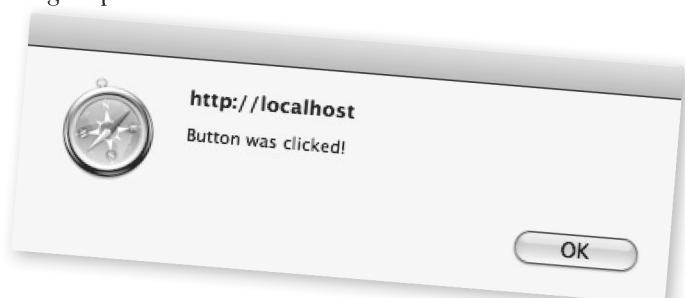


Siga em frente, digite o código acima (no seu arquivo `playlist.js`), carregue a página, clique nesse botão o quanto quiser e verá um alerta a cada vez.

Após ter terminado de testar seu novo manipulador de clique em botões, volte, estude o código e pense em como tudo isso funciona.

Quando você achar que tem tudo na mente, vire a página e nós veremos os detalhes para assegurar que tudo realmente permaneça.

1. Configurar um manipulador para lidar com um clique de usuário no botão "Adicionar música"
2. Escrever o manipulador para obter o nome da música
3. Criar um novo elemento para armazenar a nova música
4. Adicionar o elemento ao DOM da página



Um olhar mais próximo no que acabou de acontecer...

Acabamos de apresentar muitas ideias novas nas últimas páginas; vamos percorrer o código novamente e assegurar que o entendemos claramente. Aqui vamos nós:

- 1 A primeira coisa que você fez foi colocar um botão no seu formulário HTML. Com ele posicionado, você precisava de uma forma de capturar o clique de um usuário nesse botão, de modo que pudesse fazer com que algum código fosse executado. Para fazer isso, criamos um manipulador e o atribuímos à propriedade onclick do seu botão.

```
function init() {  
    var button = document.getElementById("addButton");  
    button.onclick = handleButtonClick;  
}
```

O objeto button tem uma propriedade onclick que configuraremos com a função handleButtonClick.

Configuramos o manipulador do evento “clique no botão” na função init (ou seja, após a página ter sido carregada).

Adicionar Música

Add Song

Não se preocupe. Se eu for clicado, você será literalmente o primeiro a saber.

```
function handleButtonClick() {  
    alert("Button was clicked!");  
}
```

Quando o usuário clicar no botão, o evento clique será disparado e a função handleButtonClick chamada.

- 2 Você também escreveu um manipulador simples que apenas alerta o usuário para o fato de o botão ser clicado, mas este código funciona bem para o teste.



O manipulador com seu código

- 3 Com o código escrito, a página é carregada e exibida pelo navegador, o manipulador é instalado... tudo depende do usuário agora...

Vamos lá... clique no botão... clique ...

- 4 Finalmente, o usuário clica no seu botão, ele entra em ação, percebe que possui um manipulador e o chama...



Hora de acordar,
tem um clique de
usuário.

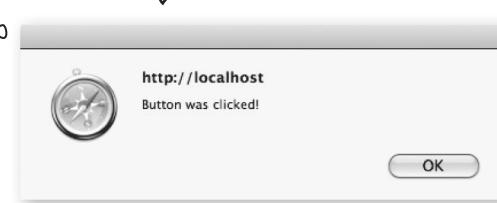
Vejo que tenho um
manipulador para isso,
é melhor avisá-lo.

Sim! Alguém clicou no botão.
Vou executar a função
handlerButtonClick.

Add Song

```
function handleButtonClick() {
    alert("Button was clicked!");
}
```

Pediram-me para alertar você
de que o botão foi clicado... Eu
sei, para uma caixa de diálogo
de alerta é um pouco exagerado,
mas, de qualquer maneira, estou
apenas fazendo meu trabalho.

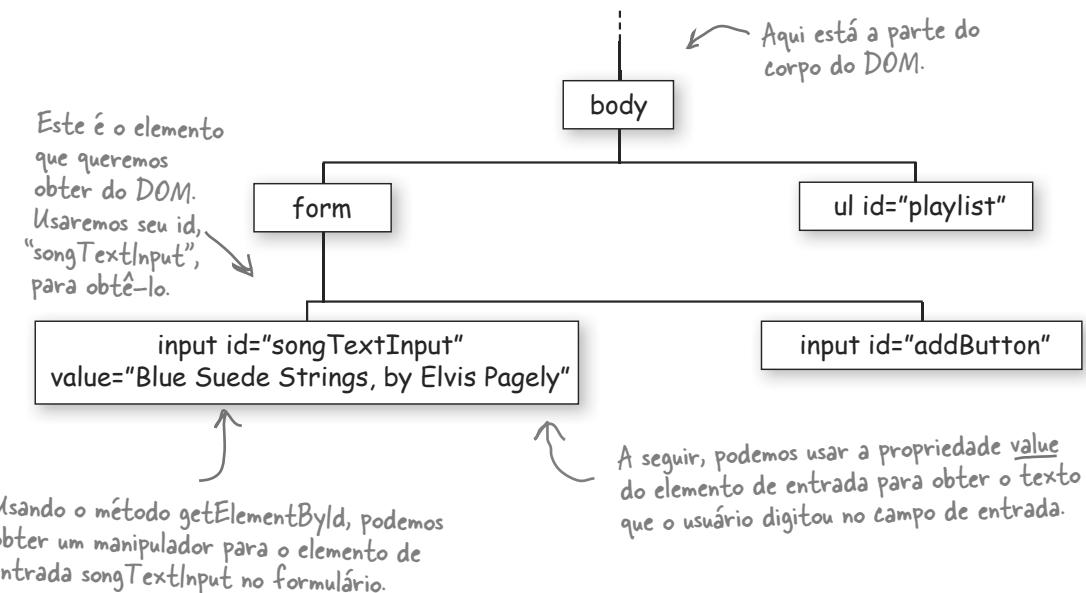


Obtendo o nome da música

Estamos prontos para seguir para o segundo passo da nossa tarefa: obter o nome da música na qual o usuário clicou. Assim que fizermos isso, poderemos pensar em como exibiremos a lista de execução no navegador.

Como, então, obteremos o nome da música? Isto é algo que o usuário digitou, certo? Ah, mas qualquer coisa que acontecer na página web é refletida no DOM, de modo que o texto que o usuário digitou deve estar lá também.

Para obter o texto de dentro de um elemento de entrada de um formulário de texto, primeiro é preciso obter o elemento de entrada a partir do DOM, e você sabe como fazer isso: `getElementById`. Assim que você tiver terminado, pode usar a propriedade `value` do elemento de entrada de texto para acessar o texto que o usuário digitou no campo do formulário. Aqui está como:



Aponte o seu lápis

Retrabalhe a função `handleButtonClick` abaixo para obter o nome da música que o usuário digitou no elemento de entrada do usuário. Verifique a resposta na solução da página 96.

```
function handleButtonClick() {  
  var textInput = document.getElementById(".....");  
  var songName = ..... .value;  
  alert("Adding " + .....);  
}
```



Aponte o seu lápis

BÔNUS

E se você quisesse testar para garantir que o usuário realmente digitou algum texto antes de clicar no botão? Como você poderia fazer isso? (Encontre a solução na página 96)

.....
.....

Perguntas Idiotas não existem

P: Qual o valor da propriedade value da entrada de texto se o usuário não tiver digitado nada? O valor é nulo? Ou o botão “Add Song” não chama o manipulador se o usuário não tiver digitado nada?

R: O botão “Add Song” não é tão inteligente. Determinar se o usuário digitou alguma coisa, depende do seu código. Para saber se a entrada de texto está vazia (ou seja, o usuário não digitou nada), você pode verificar se o seu valor é igual à string sem conteúdo, conhecida como string vazia, que é escrita como “”, ou duas aspas duplas sem nada entre elas. Entendemos o porquê de você pensar que poderia ser nula, mas, da perspectiva do campo de entrada de texto, ele não está armazenando nada. Está com uma string ainda sem nada dentro. Vá entender. ;-)

P: Achei que o “valor” de entrada do texto era um atributo. Você o está chamando de propriedade. Por quê?

R: Você está certo, value é um atributo do elemento de entrada de texto HTML. Você pode inicializar o valor de um elemento de entrada de texto usando o atributo value. Em JavaScript, porém, para acessar um valor que um usuário tenha digitado, você precisa usar a propriedade value do elemento de entrada que obtemos do DOM.

P: Que outros tipos de eventos eu posso gerenciar em JavaScript além de cliques em botões?

R: Há um grande número de outros eventos de mouse que você pode manipular. Por exemplo, você pode detectar e manipular um pressionar de teclas, uma movimentação do mouse, o arrasto do mouse, até mesmo o pressionar e segurar do mouse (diferente de um clique de mouse). Também há muitos outros tipos de eventos que mencionei de passagem, como eventos quando dados ficam disponíveis, eventos de temporizadores, relacionados com a janela do navegador, e assim por diante. Você verá alguns outros tipos de manipulação de eventos no resto do livro; assim que você souber como executar um, pode fazê-lo com praticamente todos eles!

P: O que JavaScript está fazendo enquanto espera por eventos?

R: A menos que você tenha programado seu JavaScript para fazer alguma coisa, ele fica ocioso até que algo aconteça (o usuário interaja com a interface, dados cheguem da web, um temporizador dispare, e assim por diante). Isto é uma coisa boa; significa que o poder de processamento do seu computador fará outras coisas, como tornar seu navegador responsivo. Posteriormente, neste livro, você aprenderá como criar tarefas que sejam executadas em segundo plano, de modo que seu navegador possa executar o código da tarefa e responder a eventos ao mesmo tempo.



Aponte o seu lápis

Solução

Retrabalhe a função handleClick abaixo para obter o nome da música que o usuário digitou no elemento de entrada do formulário. Aqui está a nossa solução:



Primeiro precisamos obter uma referência ao elemento de entrada de texto no formulário. Recebemos este elemento em uma id de "songTextInput", de modo que podemos usá-lo com getElementById para obter uma referência.

```
function handleClick() {
    var textInput = document.getElementById("songTextInput");
    var songName = textInput.value;
    alert("Adding " + songName);
}
```

E agora mostraremos uma caixa de alerta, que deve exibir "Adding" e o nome da música.

A propriedade value do elemento de entrada de texto armazena o que for digitado no texto, que é apenas uma string. Aqui estamos atribuindo esse texto à variável songName.



Aponte o seu lápis

Solução

BÔNUS

E se você quisesse testar para garantir que o usuário realmente digitou algum texto antes de clicar no botão? Como você poderia fazer isso? Aqui está a nossa solução:

```
function handleClick() {
    var textInput = document.getElementById("songTextInput");
    var songName = textInput.value;
    if (songName == "") {
        alert("Please enter a song");
    } else {
        alert("Adding " + songName);
    }
}
```

Podemos ver um comando if e comparar a string songName com uma string vazia para assegurar que o usuário digitou algo. Se ele não tiver digitado nada, nós o alertaremos para digitar uma string.

Como adicionamos uma música à página?

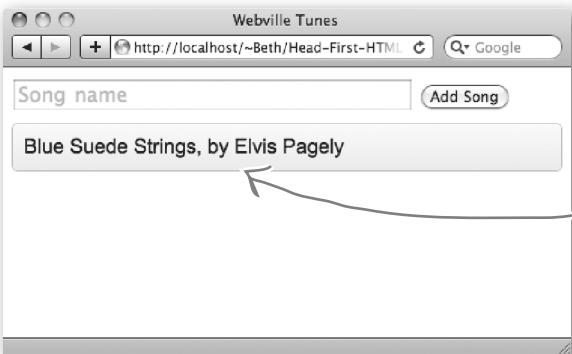
Já fizemos muita coisa! Você pode digitar um nome de música no formulário, clicar no botão Add Song e obter o texto que você digitou no formulário, *tudo dentro do seu código*. Agora exibiremos a lista de músicas na própria página. Aqui está como isso deve ficar:

1. Configurar um manipulador para lidar com um clique de usuário no botão “Adicionar música”

2. Escrever o manipulador para obter o nome da música

3. Criar um novo elemento para armazenar a nova música

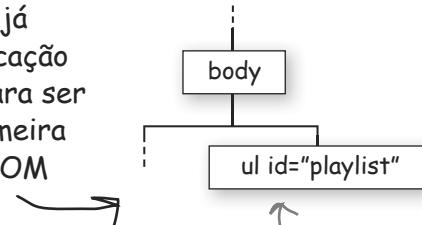
4. Adicionar o elemento ao DOM da página



Quando você clica em “Add Song”, seu JavaScript adicionará a música a uma lista de músicas na página.

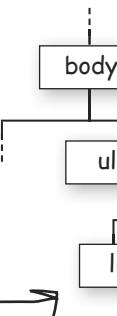
Aqui está o que precisamos fazer:

- 1) Você talvez tenha percebido que já colocamos uma lista vazia na marcação HTML (um elemento `` vazio, para ser exato) quando digitamos pela primeira vez. Com isso, aqui está como o DOM se parece agora.



Aqui está a lista no DOM. Neste momento está vazia.

- 2) Cada vez que você digitar uma nova música, adicionaremos um novo item a uma lista não ordenada. Para fazer isso, criaremos um novo elemento `` que armazenará o nome da música. A seguir, pegaremos o novo elemento `` e o adicionaremos ao `` no DOM. Assim que fizermos isso, o navegador executará sua função e você verá a atualização da página, como se o `` estivesse lá desde sempre. É claro, faremos isso tudo neste código. Verifique o DOM mais uma vez e assegure-se de ter entendido o que precisamos fazer.

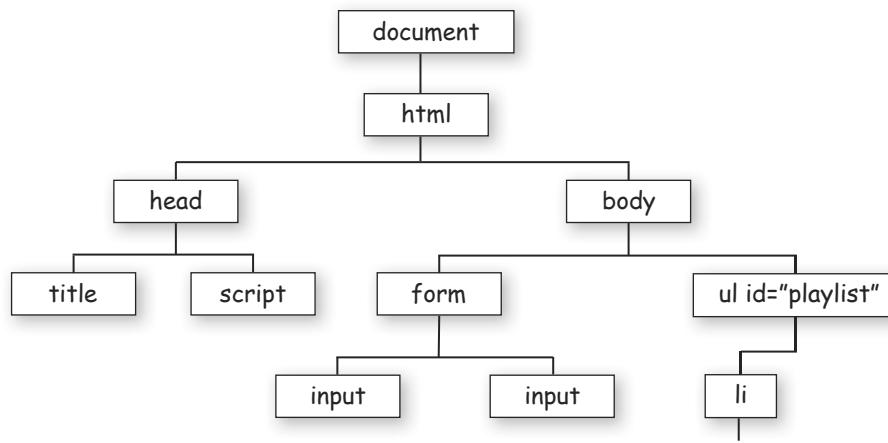


Quando você digitar uma música, criaremos um novo item de lista (elemento ``) e o adicionaremos à lista ``.



Exercício

Para a lista de execução mostrada a seguir, desenhe como o DOM ficará, após você ter adicionado todas essas músicas. Observe a ordem na qual as músicas são adicionadas na página e assegure-se de que os elementos estejam nos lugares corretos no DOM também. Já fizemos isso para você. Verifique a solução no final do capítulo antes de seguir em frente.



Blue Suede Strings,
by Elvis Pagely



Desenhe o resto do DOM aqui
para a lista de execução acima.

Você teve de fazer suposições sobre a ordem na qual os elementos `` são adicionados ao pai?

Como criar um novo elemento

Você já viu como obter acesso a *elementos existentes* através do DOM. Você também pode usar o DOM para criar *novos elementos* (e, depois, como uma segunda etapa, *adicionar*los ao DOM. Chegaremos lá em um segundo).

Digamos que queremos criar um elemento ``. Aqui está como fazemos isso:

Use `document.createElement` para criar um novo elemento. Uma referência ao novo elemento é retornada.

```
var li = document.createElement("li");
```

Aqui estamos atribuindo o novo elemento à variável `li`.

`li`

Passe o tipo de elemento que quiser criar como uma string para `createElement`.

`createElement` cria um novo elemento. Observe que ele não é inserido no DOM ainda. Agora ele é apenas um elemento livre, flutuando e precisando de um lugar no DOM.

Então, agora temos um novo elemento `` sem nada. Você já conhece uma maneira de colocar texto em um elemento:

```
li.innerHTML = songName;
```

Nossa variável `li`:

Isto considera o conceito do `` ao título da música.

`li`

Blue Suede Strings,
by Elvis Pagely

Aqui está o nosso novo objeto de elemento `li` pronto para ir, mas ainda não fazendo parte do DOM.

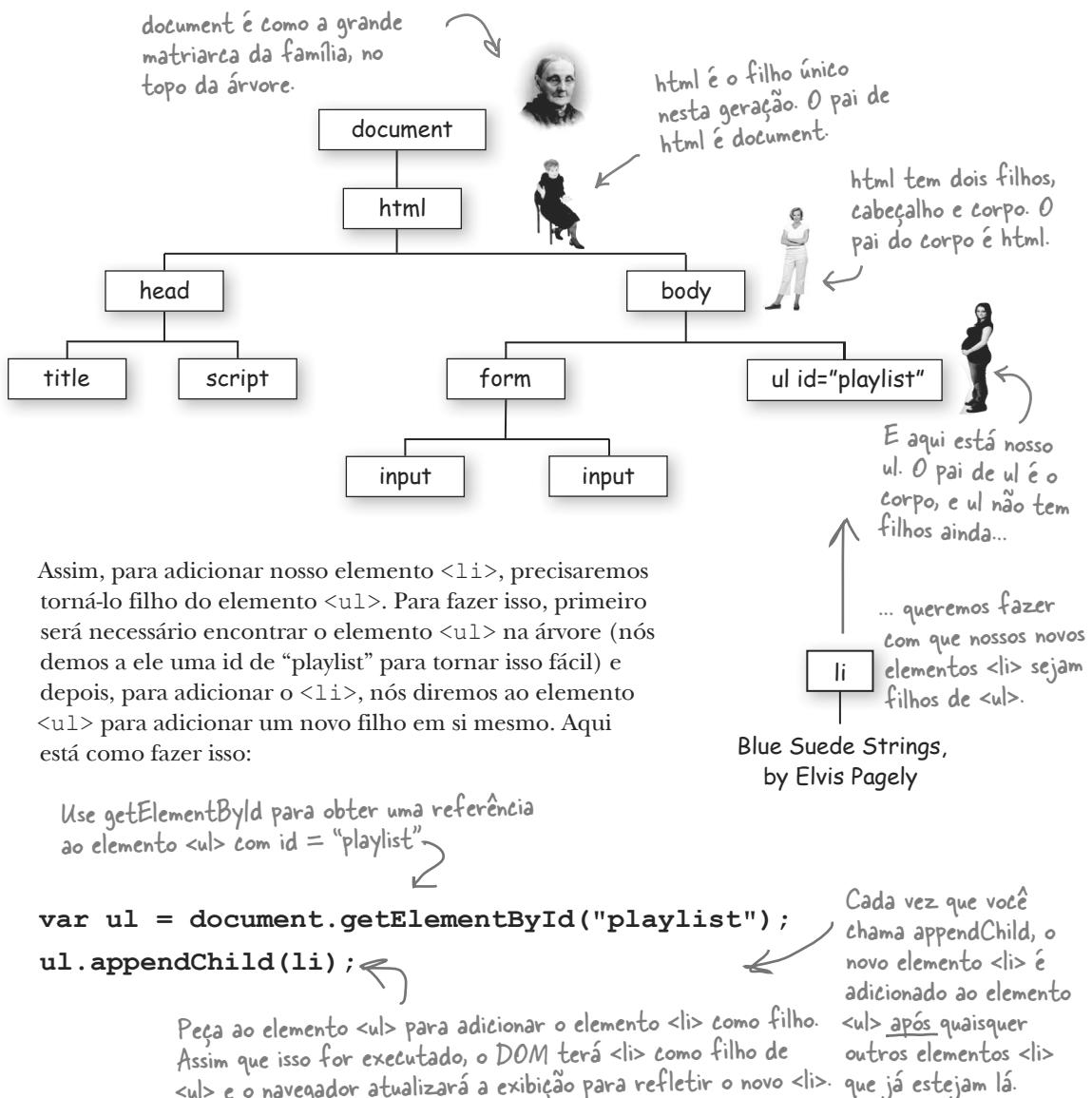


É melhor começarmos a trabalhar na criação destes elementos, Betty. Eles estão atualizando o DOM novamente.

Adicionando um elemento ao DOM

Para adicionar um novo elemento ao DOM, você tem que saber onde quer colocá-lo. Bem, nós sabemos onde colocá-lo: colocaremos o elemento `` no elemento ``. Como faremos isso? Vejamos novamente o DOM. Você se lembra de que dissemos que ele era como uma árvore? Pense em *árvore genealógica*.

1. Configurar um manipulador para lidar com um clique de usuário no botão "Adicionar música"
2. Escrever o manipulador para obter o nome da música
3. Criar um novo elemento para armazenar a nova música
- 4. Adicionar o elemento ao DOM da página**



Junte tudo...

Vamos juntar todo o código e adicioná-lo à função handleClick. Siga em frente e digite-o, caso ainda não o tenha feito, de modo que possa testá-lo.

```
function handleClick() {
    var textInput = document.getElementById("songTextInput");
    var songName = textInput.value;
    var li = document.createElement("li");
    li.innerHTML = songName;
    var ul = document.getElementById("playlist");
    ul.appendChild(li);
}
```

Observe que pedimos ao elemento pai, `ul`, para adicionar `li` como um novo filho.

Primeiro, crie o novo elemento ``, no qual irá o nome da música.

A seguir, configure o conteúdo desse elemento com o nome da música.

O `` com a id "playlist" é o elemento pai do nosso novo ``. Assim, obtemos ele a seguir.

Depois, adicione o objeto `li` ao `ul` usando `appendChild`.

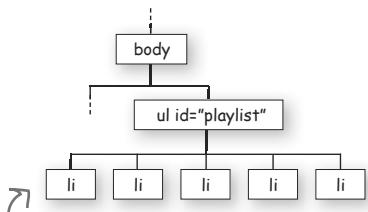
... e faça um test drive

Coloque o Webville Tunes à prova, adicionando algumas músicas. Aqui estão nossos resultados:



The screenshot shows a web browser window titled "Webville Tunes". The address bar shows the URL "http://localhost/~Beth/Head-First-HTML5/c". The page has a header with a "Song name" input field and an "Add Song" button. Below the input field is a list of songs:

- Blue Suede Strings, by Elvis Pagely
- Great Objects on Fire, by Jerry JSON Lewis
- I Code the Line, by Johnny JavaScript
- That'll be the Data, by Buddy Bitly and the Variables
- Your Random Heart, by Hank "Math" Williams



E aqui está como o DOM se parece agora que adicionamos todos aqueles novos elementos ``.



Agora, quando digitamos um nome de música e clicamos em adicionar, a música é adicionada ao DOM, assim vemos mudança na página e a nova música na lista.



Revisão — o que acabamos de fazer

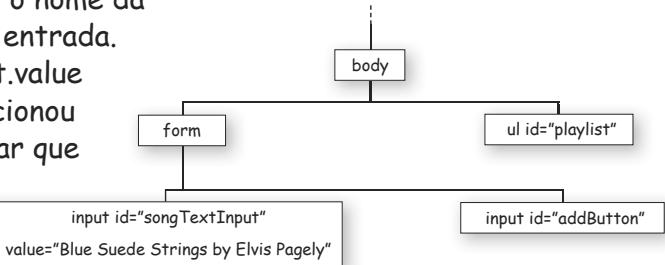
Você fez bastante neste capítulo (e em pouco tempo!). Criou um aplicativo de lista de execução que pode usar para digitar uma música, clicar em um botão e adicionar essa música à lista da página, tudo usando código JavaScript.

- 1 A primeira coisa que você fez foi configurar um manipulador de eventos para lidar com o clique do usuário no botão "Add Song". Você criou uma função, handleButtonClick, e configurou a propriedade onclick do botão "AddSong" com esta função.

Quando o usuário clicar no botão "Add Song", seu manipulador handleButtonClick será chamado.

- 2 A seguir, escreveu código para o manipulador de clique de botão para obter o nome da música do campo de texto de entrada. Você usou a propriedade input.value para obter o texto, e até adicionou uma verificação para assegurar que o usuário digitou uma música. Se não tiver digitado, você o alerta.

Em handleButtonClick, você está obtendo o nome da música que o usuário digitou, usando a propriedade input.value para obter o texto do DOM.

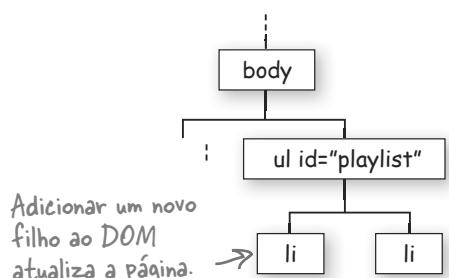


- 3 Para adicionar a música à lista de execução, você criou então um elemento usando document.createElement e configurou o conteúdo do elemento com o nome da música usando innerHTML.

Você cria um novo elemento e configura o conteúdo do elemento com o nome da música.



- 4 Finalmente, você adicionou o novo elemento ao DOM adicionando-o como filho ao elemento pai. Você fez isso usando appendChild, dizendo para o elemento "inserir o elemento como filho", que o adicionou ao DOM. Quando o elemento é adicionado ao DOM, o navegador atualiza a página que o usuário vê e a lista de execução passa a mostrar a música.



Espere um segundo. Consigo interagir com o DOM e tudo o mais, mas como isto é um aplicativo web real? Se eu fechar meu navegador, todas as minhas músicas desaparecem. Minha lista de exibição não deveria permanecer se isto fosse realmente um aplicativo?



Concordamos, a lista de execução deve ser persistente; afinal, qual o sentido de digitar todas essas músicas se elas não permanecerem? Há ainda muitas outras funcionalidades que você poderia querer adicionar também. Você poderia, por exemplo, querer adicionar uma interface de áudio usando a API de áudio e vídeo, de modo que pudesse realmente ouvir e compartilhar músicas com amigos usando um serviço web (como o Facebook e o Twitter), encontrar pessoas perto de você que gostem dos mesmos artistas (usando as APIs de geolocalização) e temos certeza de que poderia imaginar muitas outras funcionalidades.

Voltando, porém, à lista de execução... A ideia era lhe iniciar construindo um pequeno aplicativo web interativo e a lista de exibição fez um bom trabalho nisso. Além disso, armazenar as músicas requer a API Web Storage de HTML5, que está a alguns capítulos de distância.

Por outro lado, realmente não queremos fazer coisas de menos aqui...

Mude a Página





Já pré-assamos algum código para que você não tenha de fazer isso.

Seguimos em frente e assamos um pouco de código por você para gravar suas listas de execuções. Por enquanto, você só precisa digitar e fazer duas pequenas alterações no seu código já existente e terá uma lista de execução baseada em HTML5.

Examinaremos as especificidades de armazenamento de coisas localmente no seu navegador no capítulo Armazenamento Web, mas, por enquanto, você pode fazer sua lista de execução funcionar.

É claro que não faz mal examinar o código de Pronto para Assar. Você talvez se surpreenda com o quanto que já sabe, para não mencionar o quanto você pode descobrir se não souber.



O Código Pronto para Assar não funcionará no IE6 ou 7.

Versões do IE 6 ou 7 não suportam localStorage. Assim, se você estiver usando o IE, assegure-se de estar usando a versão 8 ou posterior.



Veja bem!

O Código Pronto para Assar não funcionará em alguns navegadores, se estiver servindo suas páginas a partir de file:// em vez de um servidor de arquivos como localhost:// ou um servidor hospedado localmente.

Vamos lidar mais com esta situação nos próximos capítulos (que com bastante frequência aparece com novos recursos do HTML5). Por ora, se você não quiser executar um servidor ou copiar os arquivos para um servidor online hospedado, tente usar o Safari ou Chrome.

Como adicionar o Código Pronto para Assar...



Aqui está o código Pronto para Assar para você adicionar ao seu aplicativo Webville Tunes, de modo que você possa gravar essa lista de execução fabulosa que criou. Tudo o que você tem de fazer é criar um arquivo novo, `playlist_store.js`, digitar o código abaixo e depois fazer algumas alterações no seu código já existente (na próxima página).

```

function save(item) {
    var playlistarray = getStorearray("playlist");
    playlistarray.push(item);
    localStorage.setItem("playlist", JSON.stringify(playlistarray));
}

function loadPlaylist() {
    var playlistarray = getSavedSongs();
    var ul = document.getElementById("playlist");
    if (playlistarray != null) {
        for (var i = 0; i < playlistarray.length; i++) {
            var li = document.createElement("li");
            li.innerHTML = playlistarray[i];
            ul.appendChild(li);
        }
    }
}

function getSavedSongs() {
    return getStorearray("playlist");
}

function getStorearray(key) {
    var playlistarray = localStorage.getItem(key);
    if (playlistarray == null || playlistarray == "") {
        playlistarray = new array();
    }
    else {
        playlistarray = JSON.parse(playlistarray);
    }
    return playlistarray;
}

```

Digite isto dentro de "playlist_store.js".

Integrando seu Código Pronto para Assar

Precisamos fazer algumas alterações para integrar o código de armazenamento. Primeiro, adicione uma referência a `playlist_store.js` ao seu elemento `<head>` em `playlist.html`:



Código PRONTO
PARA ASSAR

```
<script src="playlist_store.js"></script>
<script src="playlist.js"></script>
```

Adicione isto logo acima
do seu link para `playlist.
js`. Ele carrega o código
Pronto para Assar.

Agora, você só precisa adicionar duas linhas ao seu código, em `playlist.js`, que carregará e gravará a lista de execução:

```
function init() {
    var button = document.getElementById("addButton");
    button.onclick = handleButtonClick;
    loadPlaylist(); ← Isto carrega as músicas gravadas
}                                a partir do localStorage quando
function handleButtonClick() { ← carregar sua página, de modo que você
    var textInput = document.getElementById("songTextInp
    ut");
    var songName = textInput.value;
    var li = document.createElement("li");
    li.innerHTML = songName;
    var ul = document.getElementById("list");
    ul.appendChild(li);
    save(songName); ← E isto grava sua música cada vez que
}                                você adicionar uma à lista de execução.
```

Faça um test drive com as músicas gravadas

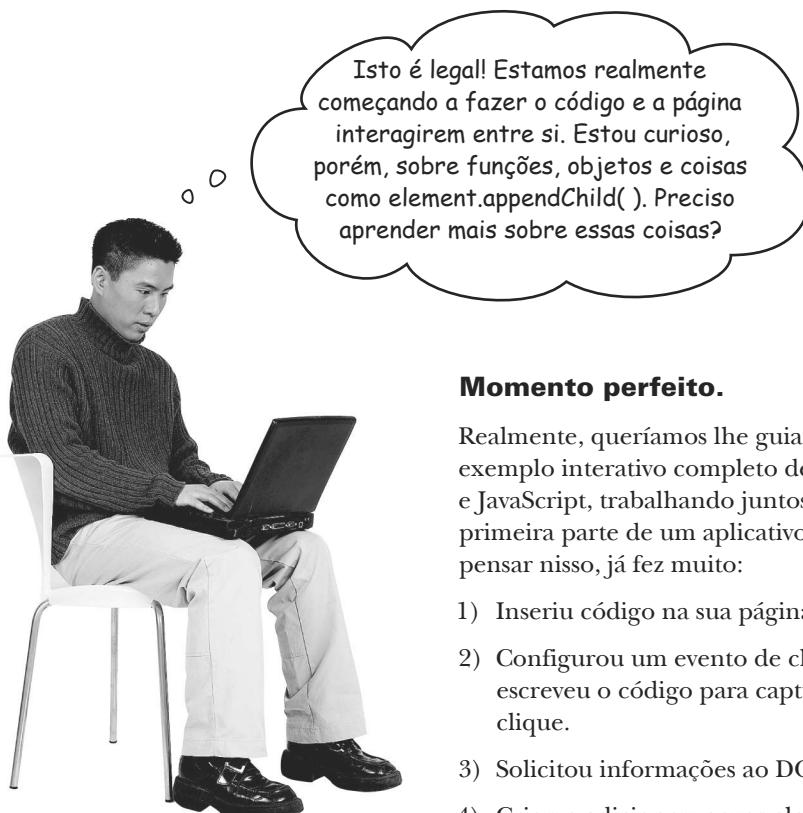


OK, recarregue a página e digite algumas músicas. Feche o navegador. Abra o navegador e carregue a página novamente. Você deve ver todas as músicas armazenadas com segurança na sua lista de execução.

OK, você está cansado da sua lista de execução e quer apagá-la? Terá que verificar o capítulo do Armazenamento Web!



Adicionamos todas
estas músicas,
fechamos o
navegador, depois
o reabrimos,
carregamos a página
e lá estão elas.



Momento perfeito.

Realmente, queríamos lhe guiar através de um exemplo interativo completo de marcação HTML e JavaScript, trabalhando juntos para criar a primeira parte de um aplicativo web. Se você pensar nisso, já fez muito:

- 1) Inseriu código na sua página.
- 2) Configurou um evento de clique em botão e escreveu o código para capturar e lidar com o clique.
- 3) Solicitou informações ao DOM.
- 4) Criou e adicionou novos elementos ao DOM.

Nada mal! Agora, que você tem um pouco de sentido intuitivo sobre como tudo isso funciona junto, vamos fazer um pequeno desvio pela Avenida JavaScript para ver como coisas como funções e objetos *realmente funcionam*.

Esta não será uma excursão comum; não mesmo. Levantaremos as tampas de bueiros e obteremos uma visão rara de como Webville funciona.

Interessado? Venha, junte-se a nós no Capítulo 4...



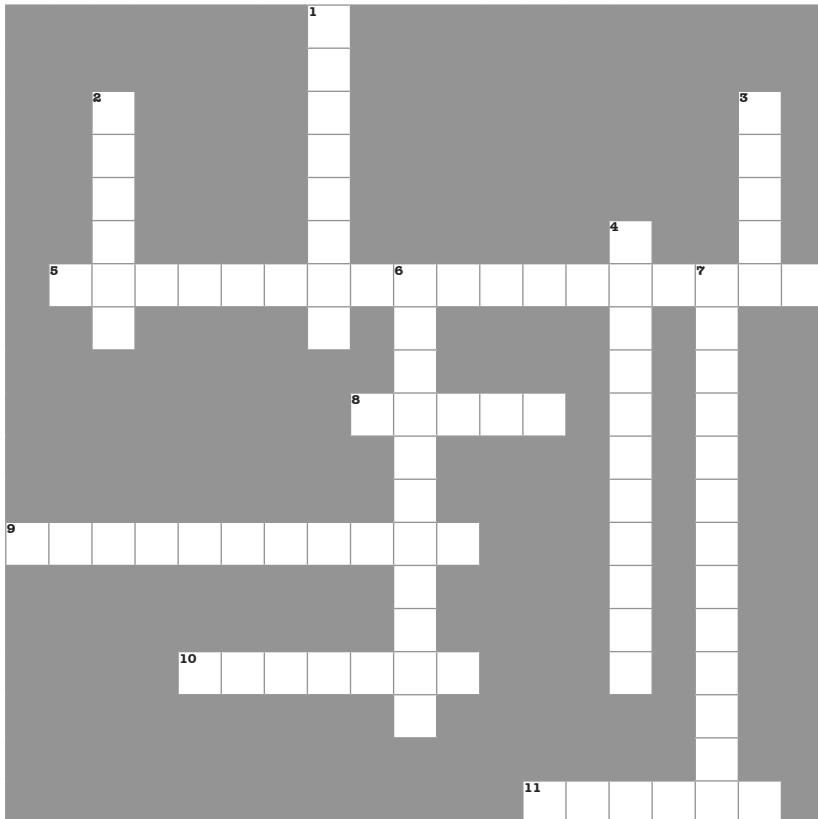
PONTOS DE BALA

- Há muitos eventos acontecendo no seu navegador o tempo todo. Se você quiser responder a estes eventos, precisa manipulá-los com manipuladores de eventos.
- Um evento de clique em botão é disparado quando você clica em um botão em uma página web.
- Você manipula um evento de clique no botão, registrando uma função para manipular o evento. Você faz isso escrevendo uma função e configurando a propriedade onclick do botão com o nome da função.
- Se um manipulador de eventos de clique em botão estiver registrado, essa função será chamada quando você clicar no botão.
- Você escreve código na função manipuladora para responder ao evento de clique no botão. Você pode alertar o usuário ou atualizar a página ou fazer alguma outra coisa.
- Para obter o valor do texto que foi digitado em um campo de texto de entrada em formulário, você usa a propriedade value da entrada.
- Se um usuário não tiver digitado nada em um campo de texto de entrada de formulário, o valor do campo será a string vazia ("").
- Você pode comparar uma variável com a string vazia usando um teste if e ==.
- Para adicionar um novo elemento ao DOM, você primeiro precisa criar o elemento e depois adicioná-lo como filho de um elemento.
- Use document.createElement para criar um novo elemento. Passe o nome da tag (por exemplo "li") para a chamada da função para indicar que elemento criar.
- Para adicionar um elemento como filho de um outro elemento no DOM, obtenha uma referência ao pai e chame appendChild sobre ele, passando no elemento filho que você estiver adicionando.
- Se você adicionar múltiplos elementos a um pai usando appendChild, cada novo filho é inserido após os outros filhos, de modo que aparecem após ou abaixo dos outros filhos na página (supondo que você não altere o layout com CSS).
- Você pode usar a API Web Storage (localStorage) para armazenar dados no navegador de um usuário.
- Usamos localStorage para gravar músicas em listas de execução, usando o código Pronto para Assar.



Palavras Cruzadas HTML5

Dê a si mesmo algum tempo para entender as interações entre HTML e JavaScript. Pense em como tudo isso funciona junto. Enquanto você estiver fazendo isso, misture tudo um pouco fazendo essas palavras cruzadas. Todas as palavras são deste capítulo.



Horizontais

5. Usado em Pronto para assar para habilitar o armazenamento.
8. Insira novos elementos como um ____.
9. O método de DOM para adicionar novos elementos.
10. O que há a frente? Funções e ____.
11. Um clique em botão é um ____.

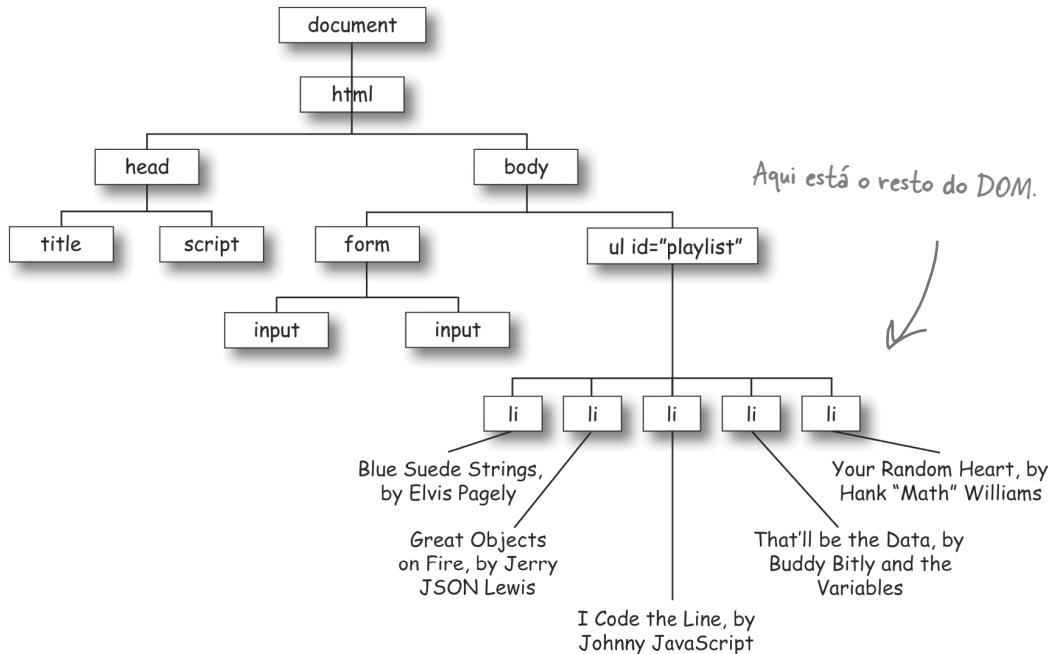
Verticais

1. A grande matriarca da árvore DOM.
2. O DOM é como uma ____ de família.
3. O valor default de um elemento de entrada de formulário se o usuário não digitar nada é a string ____.
4. Artista usado na nossa música de exemplo.
6. Código que cuida dos eventos.
7. O método de DOM para criar novos elementos.



Exercício Solução

Para a lista de execução mostrada a seguir, desenhe como o DOM ficará, após você ter adicionado todas essas músicas. Observe a ordem na qual as músicas são adicionadas na página e assegure-se de que os elementos estejam nos lugares corretos no DOM também. Aqui está a nossa solução.

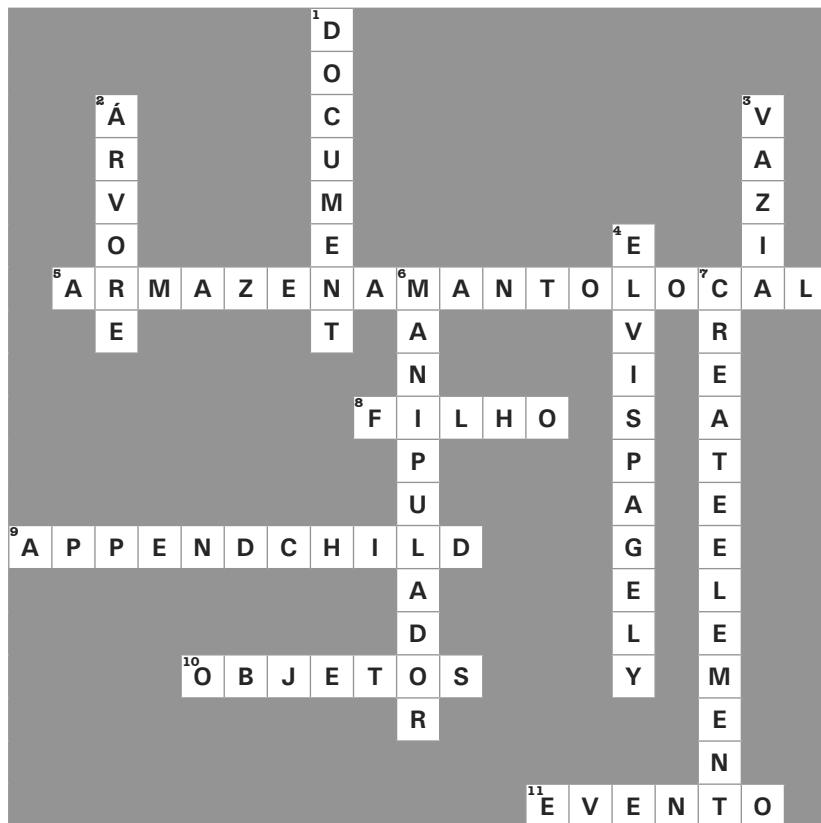


Você teve que fazer alguma suposição sobre a ordem na qual os elementos são adicionados ao pai?

Sim, porque isto afeta a ordem de exibição das músicas na página. appendChild sempre insere o novo elemento após os filhos já existentes



Solução das Palavras Cruzadas HTML5



4 funções e objetos em JavaScript



JavaScript Sério



Você já pode se chamar de scripter? Provavelmente, você já sabe fazer bastante coisa em JavaScript. No entanto, quem quer ser um scripter quando pode ser um programador? É hora de ficarmos sérios e acelerar um pouco — é hora de você aprender sobre **funções** e **objetos**. Eles são a chave para escrever um código que seja mais poderoso, melhor organizado e de mais fácil manutenção. Eles também são amplamente usados pelas APIs JavaScript HTML5, de modo que, quanto melhor você entendê-los, mais rapidamente poderá passar para uma nova API e começar a lidar com ela. Prepare-se: este capítulo irá requerer sua atenção exclusiva...

Expandindo seu vocabulário

Você já pode fazer muito com JavaScript. Vamos examinar algumas das coisas que você sabe fazer:

```
<script>  
var guessInput = document.getElementById("guess");  
var guess = guessInput.value;  
var answer = null;  
  
var answers = [ "red",  
                "green",  
                "blue" ];  
  
var index = Math.floor(Math.random() * answers.length);  
  
if (guess == answers[index]) {  
    answer = "You're right! I was thinking of " + answers[index];  
} else {  
    answer = "Sorry, I was thinking of " + answers[index];  
}  
alert(answer);  
</script>
```

Pegar um elemento do modelo de objetos do documento.

Obter o valor de uma caixa de entrada de texto em formulário.

Criar uma nova matriz preenchida com strings.

Usar bibliotecas de funções.

Tomar decisões baseadas em condições.

Obter uma propriedade de uma matriz, como length.

Usar os elementos de uma matriz

Usar funções de navegador, como alert.



Até aqui, porém, muito do seu conhecimento é informal — claro, você pode obter um elemento do DOM e atribuir HTML novo a ele, mas se você for solicitado a explicar exatamente o que document.getElementById é tecnicamente, bom, isso poderia ser um pouco mais desafiador. Não se preocupe; quando você terminar este capítulo, já estará preparado.

Para lhe conduzir, não começaremos com uma análise profunda e técnica de getElementById, não, faremos algo *um pouco mais interessante*. Estenderemos o vocabulário de JavaScript e faremos com que ele faça coisas novas.

Como adicionar suas próprias funções

Você tem usado funções internas, como `alert`, ou mesmo `Math.random`, mas e se quisesse adicionar as suas próprias? Digamos que você quisesse escrever algum código como este:

```
var guessInput = document.getElementById("guess");
var guess = guessInput.value;
var answer = checkGuess(guess);
alert(answer);
```

... mas, em vez de ter todo o resto do código da página anterior como parte do código principal, preferimos apenas fazer uma boa função "checkGuess", que possamos chamar e que faça a mesma coisa.

Estamos pegando o palpite (`guess`) do usuário da mesma forma com que fizemos na página anterior...

Crie uma função `checkGuess`

- Para criar uma função, use a palavra-chave `function` e coloque um nome após, com "`checkGuess`".

```
function checkGuess(guess) {
    var answers = [ "red",
                    "green",
                    "blue" ];
    var index = Math.floor(Math.random() * answers.length);

    if (guess == answers[index]) {
        answer = "You're right! I was thinking of " + answers[index];
    } else {
        answer = "Sorry, I was thinking of " + answers[index];
    }
    return answer;
}
```

2 Dê à sua função zero ou mais parâmetros. Use parâmetros para passar valores à sua função. Só precisamos de um parâmetro aqui: o palpite (`guess`) do usuário.

- Opcionalmente, retorne um valor como resultado da chamada à função. Aqui estamos retornando uma string com uma mensagem.

- Escreva um corpo para sua função, que vai entre as chaves. O corpo contém todo o código que executa o trabalho da função. Para o corpo aqui, reutilizaremos nosso código da página anterior.

Como uma função funciona

Então, como tudo isso funciona? O que acontece quando nós *realmente chamamos* uma função? Aqui está uma visão geral:

OK, primeiro precisamos de uma função

Digamos que você tenha acabado de escrever sua nova função bark (latir), que tem dois parâmetros, dogName (nome do cachorro) e dogWeight (peso do cachorro), e também um trecho de código muito impressionante que retorna o latido de um cachorro, dependendo do peso dele, é claro.

```
function bark(dogName, dogWeight) {  
    if (dogWeight <= 10) {  
        return dogName + " says Yip";  
    } else {  
        return dogName + " says Woof";  
    }  
}
```

Aqui está nossa útil função bark (latir).

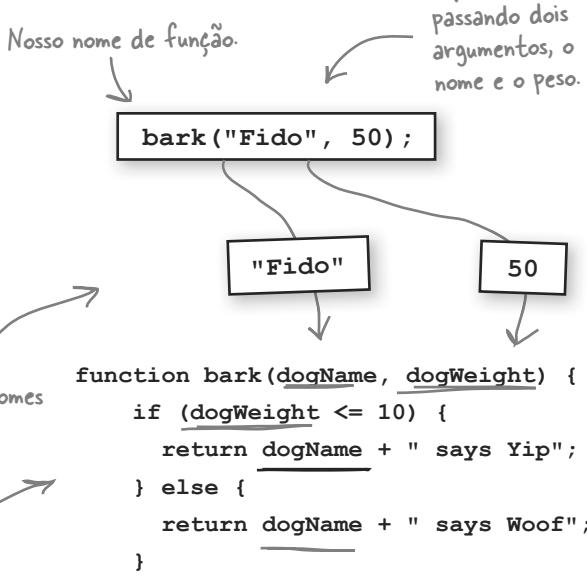
Agora vamos chamá-la!

Você já sabe como chamar uma função: apenas use seu nome e dê a ela os argumentos que ela precisa. Neste caso, precisamos de dois: uma string com o nome do cachorro e um peso, que é um número inteiro.

Vamos fazer essa chamada e ver como isto funciona:

Quando chamamos bark, os argumentos são atribuídos aos nomes dos parâmetros na função bark.

E sempre que os parâmetros aparecerem na função, os valores que passamos serão usados.



E deixamos o corpo da função fazer seu trabalho.

Após termos atribuído o valor de cada argumento ao seu parâmetro correspondente na função — como “Fido” a dogName (nome do cachorro) e o valor inteiro 50 a dogWeight (peso do cachorro) — estamos prontos para avaliar todos os comandos no corpo da função.

Os comandos são avaliados de cima para baixo, da mesma forma que todos os outros códigos que temos escrito. O que é diferente é que estamos fazendo isso em um ambiente onde os nomes de parâmetros dogName (nome do cachorro) e dogWeight (peso do cachorro) são atribuídos aos argumentos que você passou para a função.

```
function bark(dogName, dogWeight) {
  if (dogWeight <= 10) {
    return dogName + " says Yip";
  } else {
    return dogName + " says Woof";
  }
}
```

Aqui avaliamos todo o código do corpo.

Opcionalmente, podemos ter comandos de retorno no corpo...

... e é onde retornamos um valor para o código que faz a chamada. Vejamos como isso funciona:

Andando pela função, dogWeight (peso do cachorro) não é menor ou igual a 10, então usamos a cláusula else e retornamos “Fido says Woof” (“Fido diz Au”) como um valor string.

```
function bark(dogName, dogWeight) {
  if (dogWeight <= 10) {
    return dogName + " says Yip";
  } else {
    return dogName + " says Woof";
  }
}
```

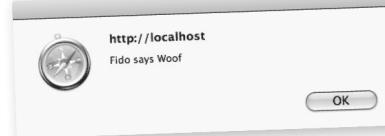
Lembre-se de que funções não precisam retornar um valor. Neste caso, a função bark (latir) retorna um valor.

A string “Fido says Woof” (“Fido diz Au”) é retornada para o código que chama (é o código que chamou a função bark (latir)).

“Fido says Woof”

```
var sound = bark("Fido", 50);
alert(sound);
```

E quando a string é retornada, é atribuída à variável sound, que é então passada para alert, resultando no diálogo.



Eu continuo dizendo, todas
as APIs HTML5 são cheias de
funções, objetos e todas aquelas
coisas avançadas de JavaScript...



Se pudéssemos ter outro momento para conversar...

Sabemos que, lá pelo Capítulo 4, você achou que estaria voando em um jato HTML5, e chegaremos lá. Antes, porém, de chegarmos a esse ponto, você realmente precisa entender os *fundamentos* das APIs JavaScript HTML5. Faremos isso neste capítulo.

Assim, o que são esses fundamentos? Pense nas APIs JavaScript HTML5 como constituídas por objetos, métodos (em outros casos conhecidos como funções) e propriedades. Para realmente entender e dominar estas APIs, você precisa entender essas coisas muito bem. Claro, você poderia tentar seguir em frente sem conhecê-las, mas sempre estará adivinhando como usar as APIs e não conseguindo usá-las integralmente (para não falar nos muitos erros e código cheio de falhas).

Então, só queríamos lhe dizer, antes que você vá a fundo neste capítulo, o que aprenderá a fazer. Aqui, a grande coisa está muito melhor relatada do que em torno de 98% dos scripters JavaScript por aí. Sério.



A Função Exposta

A entrevista desta semana:
algumas coisas que você não sabia...

Use a Cabeça!: Bem-vinda, Função!
Estamos muito interessados em saber o que você é e o que faz.

Função: Estou feliz em estar aqui.

Use a Cabeça!: Estamos percebendo agora que muitas pessoas que são novas em JavaScript não tendem a usar você muito. Elas apenas escrevem seu código, linha a linha, de cima para baixo. Por que deveriam examinar você?

Função: Sim, e isto é uma pena, porque sou poderosa. Pense em mim da seguinte maneira: eu lhe dou uma forma de pegar o código, escrevê-lo uma vez e depois reutilizá-lo várias vezes.

Use a Cabeça!: Bom, desculpe-me por dizer isso, mas se você estiver dando às pessoas apenas a capacidade de fazer a mesma coisa diversas vezes... é um pouco entediente, não?

Função: Não, funções são parametrizadas — em outras palavras, cada vez que você usa a função, passa a ela argumentos, de modo que pode obter resultados que variam, dependendo do que passar.

Use a Cabeça!: Exemplos?

Função: Digamos que você precise informar aos seus usuários quanto os itens no seu carrinho de compras custarão, de modo que escreve uma função `computeShoppingCartTotal`. A seguir, você passa a essa função diferentes carrinhos de compras que pertencem a diferentes usuários e cada vez obtém o custo apropriado do carrinho de compras.

... A propósito, voltando ao seu comentário sobre novos codificadores que não estão usando funções, isto simplesmente não

é verdade, eles as usam todo o tempo: `alert`, `document.getElementById`, `Math.random`. Eles apenas não estão definindo suas *próprias* funções.

Use a Cabeça!: Bom, certo, `alert` faz sentido, mas as outras duas não se parecem muito com funções.

Função: Ah, elas são funções, veja... espere um segundo...

... ah, acabaram de me dizer que os leitores não aprenderam sobre estes tipos de funções ainda, mas chegarão lá em algumas páginas. De qualquer forma, funções estão em todos os lugares.

Use a Cabeça!: Então, uma coisa que uma função tem que fazer é retornar um valor, certo? Quero dizer, e se eu não tiver um valor que queira retornar?

Função: Muitas funções retornam valores, mas uma função não tem que fazê-lo. Diversas funções só fazem algo como atualizar o DOM e depois retornar sem nenhum valor, e já está bom.

Use a Cabeça!: Então nessas funções eu não tenho um comando de retorno?

Função: Isso mesmo.

Use a Cabeça!: Bom, e quanto a dar nomes para as suas funções; ouvi que você não tem que fazer isso também, se não quiser.

Função: OK, não vamos enervar muito a audiência. Que tal voltarmos a esse tópico após eles conhecerem um pouco mais sobre mim?

Use a Cabeça!: Desde que você me dê uma entrevista exclusiva...

Função: Conversaremos...

Não tenho certeza se entendo
a diferença entre um parâmetro
e um argumento - são apenas
dois nomes para a mesma coisa?

Não, são diferentes.

Quando você define uma função, pode *defini-la* com
um ou mais *parâmetros*.

Aqui estamos definindo
três parâmetros: *degrees*,
mode e *duration*.

```
function cook(degrees, mode, duration) {  
    // your code here  
}
```



Quando você chama uma função, a *chama* com *argumentos*:

```
cook(425.0, "bake", 45);
```



Estes são argumentos. Há três argumentos,
um número de ponto flutuante, uma string
e um número inteiro.

```
cook(350.0, "broil", 10);
```



Assim, você só definirá seus parâmetros uma vez, mas
provavelmente chamará suas funções com muitos
argumentos diferentes.

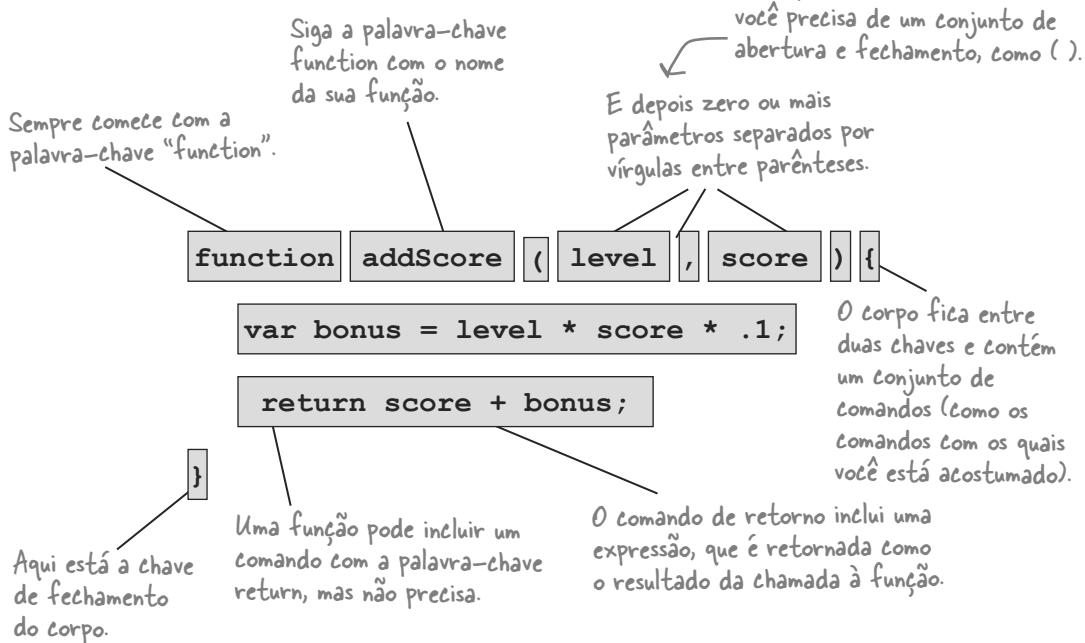
Você ficaria surpreso com o número de
pessoas que entendem isso errado – até
livros entendem errado, de modo que, se
você ler isso diferentemente em algum
outro lugar, agora você sabe melhor...

**Você define uma função com parâmetros, chama
uma função com argumentos.**



A Anatomia de uma Função

Agora que você sabe como definir e chamar uma função, vamos nos assegurar de que temos a sintaxe correta. Aqui estão todas as partes da anatomia de uma função:



não existem Perguntas Idiotas

P: Por que os nomes de parâmetros têm var na frente? Um parâmetro é uma nova variável, certo?

R: Efetivamente, sim. A função faz todo o trabalho de instanciar a variável para você, de modo que não precise fornecer a palavra-chave `var` na frente dos seus nomes de parâmetros.

P: Quais são as regras para os nomes de funções?

R: As regras para dar nome a uma função são as mesmas para dar nomes a uma variável.

P: Estou passando uma variável para minha função — se eu alterar o valor do parâmetro correspondente na minha função, isso também altera a minha variável original?

R: Não. Quando eu passar um valor primitivo, ele é copiado para o parâmetro. Chamamos isto de "passagem por valor". Assim, se você alterar o valor do parâmetro no corpo da sua função, isto não tem efeito no valor do seu argumento inicial. A exceção é a passagem de uma matriz ou objeto, e chegaremos lá em breve.

P: Então, como eu altero valores em uma função?

R: Você só pode alterar os valores de variáveis globais (as definidas fora das funções), ou variáveis que você tenha definido explicitamente na sua função. Falaremos sobre isso com um pouco mais de detalhes em breve.

P: O que uma função retorna se não tiver um comando de retorno?

R: Uma função sem comando de retorno retorna `undefined`.



Aponte o seu lápis

Use seu conhecimento sobre funções e passagem de argumentos para parâmetros para avaliar o código abaixo. Após passar pelo código, escreva o valor de cada variável abaixo. Verifique suas respostas com a solução no final do capítulo antes de seguir em frente.

```
function dogsAge(age) {  
    return age * 7;  
}  
  
var myDogsAge = dogsAge(4);  
  
function rectangleArea(width, height) {  
    var area = width * height;  
    return area;  
}  
var rectArea = rectangleArea(3, 4);  
  
function addUp(numArray) {  
    var total = 0;  
    for (var i = 0; i < numArray.length; i++)  
    {  
        total += numArray[i];  
    }  
    return total;  
}  
  
var theTotal = addUp([1, 5, 3, 9]);  
  
function getAvatar(points) {  
    var avatar;  
    if (points < 100) {  
        avatar = "Mouse";  
    } else if (points > 100 && points < 1000)  
    {  
        avatar = "Cat";  
    } else {  
        avatar = "Ape";  
    }  
    return avatar;  
}  
var myAvatar = getAvatar(335);
```

Escreva o
valor de cada
variável aqui...

myDogsAge =
rectArea =
theTotal =
myAvatar =



Variáveis Locais e Globais

Conheça a diferença ou arrisque-se à humilhação

Você já sabe que pode declarar uma variável usando a palavra-chave `var` e um nome em qualquer lugar do seu script:

```
var avatar;
var levelThreshold = 1000;
```

Estas são variáveis globais,
elas são acessíveis em qualquer
lugar do seu código JavaScript.

E sabe que também pode declarar variáveis dentro de uma função:

```
function getScore(points) {
  var score;
  for (var i = 0; i < levelThreshold; i++) {
    //code here
  }
  return score;
}
Mesmo se usarmos levelThreshold dentro da função, ela é global porque é declarada fora da função.
```

As variáveis `points`, `score` e `i` são todas declaradas dentro de uma função.

Nós as chamamos de variáveis locais porque só são conhecidas localmente dentro da própria função.

Se uma variável for declarada fora de uma função, é GLOBAL. Se for declarada dentro de uma função, é LOCAL.

O que importa isso? Variáveis são variáveis, certo? Bom, *onde* você declara suas variáveis determina *o quanto visíveis* elas são para as outras partes do seu código e, posteriormente, entender como estes dois tipos de variáveis operam lhe ajudará a escrever um código mais sustentável (para não mencionar que lhe ajudará a entender o código de outros).

Conhecendo o escopo das suas variáveis locais e globais

Onde você definir suas variáveis determinará seu *escopo*, ou seja, onde elas estão definidas e onde não estão, onde estão visíveis para o seu código e onde não estão.

Vejamos um exemplo de variáveis de escopo local e global — lembre-se de que as variáveis que você define fora de uma função têm escopo global, e as variáveis de função têm escopo local.

```

var avatar = "generic";
var skill = 1.0;
var pointsPerLevel = 1000;
var userPoints = 2008;

function getAvatar(points) {
    var level = points / pointsPerLevel;

    if (level == 0) {
        return "Teddy bear";
    } else if (level == 1) {
        return "Cat";
    } else if (level >= 2) {
        return "Gorilla";
    }
}

function updatePoints(bonus, newPoints) {
    for (var i = 0; i < bonus; i++) {
        newPoints += skill * bonus;
    }
    return newPoints + userPoints;
}

userPoints = updatePoints(2, 100);
avatar = getAvatar(2112);

```

Estas quatro variáveis têm escopo global. Isto significa que são definidas e visíveis em todo o código abaixo.

Observe que, se você conectar com scripts adicionais na sua página, eles verão estas variáveis globais também!

A variável `level` aqui é local e visível apenas para o código dentro da função `getAvatar`. Isto significa que apenas esta função pode acessar a variável `level`.

E não esqueçamos o parâmetro `points`, que também tem escopo local na função `getAvatar`.

Observe que `getAvatar` faz uso da variável global `pointsPerLevel` também!

Em `updatePoints` temos uma variável local `i`. Esta variável é visível para todo o código em `updatePoints`.

`bonus` e `newPoints` também são locais em `updatePoints`, enquanto que `userPoints` é global.

E aqui no nosso código podemos usar apenas as variáveis globais, não temos acesso a qualquer variável dentro das funções porque elas não ficam visíveis no escopo global.

As vidas curtas das variáveis

Quando você é uma variável, trabalha duro e a vida pode ser curta. Quer dizer, a menos que você seja uma variável global, mas mesmo para estas a vida tem seus limites. O que, então, determina a vida de uma variável? Pense nisso da seguinte maneira:

Variáveis globais vivem tanto tempo quanto a página.

Uma variável global começa quando seu JavaScript é carregado na página. A vida dela termina quando a página vai embora. Mesmo se você recarregar a mesma página, todas as suas variáveis globais são destruídas e depois recriadas na página recém-carregada.

Variáveis locais geralmente desaparecem quando sua função termina.

Variáveis locais são criadas quando sua função é chamada e vivem até que ela retorne (com um valor ou não). Isto dito, você pode pegar os valores das suas variáveis locais e retorná-los a partir da função antes que elas encontrem seu criador digital.

Assim, NÃO há escapatória da página, há? Se você for uma variável local, sua vida vem e vai rapidamente e, se for suficientemente sortudo para ser uma global, só vive enquanto o navegador não recarregar a sua página.

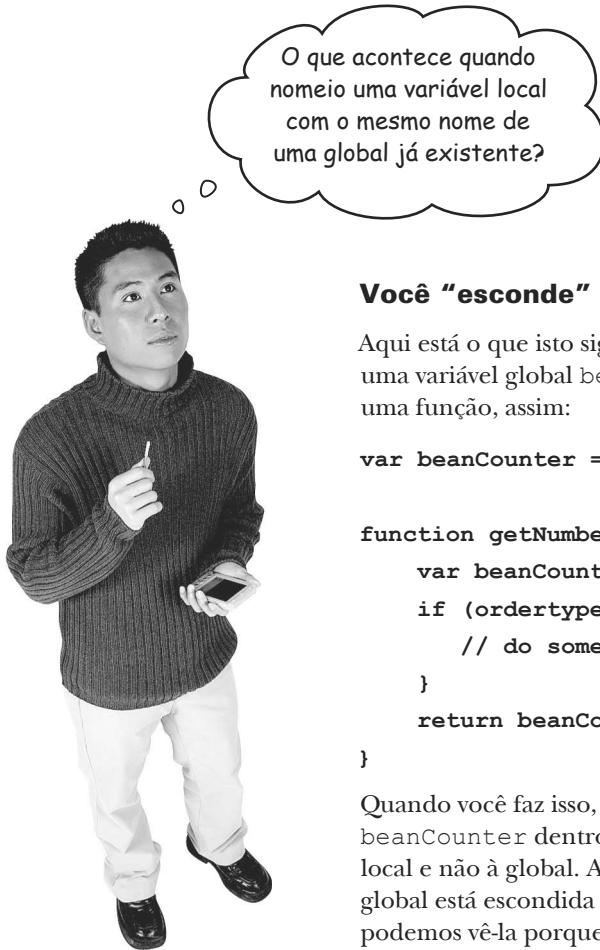
Tem que haver um meio de escapar da página! Podemos encontrar um! Não podemos?

Eu poderia jurar que a variável estava bem atrás de mim, mas, quando me virei, ela tinha... sumido...



Dizemos “geralmente” porque há algumas formas avançadas de reter as locais um pouco mais tempo, mas não nos preocuparemos com elas agora

Junte-se a nós no capítulo Armazenamento Web, em que ajudaremos nossos dados a escapar da terrível recarga de página.



Você “esconde” a sua global.

Aqui está o que isto significa: digamos que você tenha uma variável global `beanCounter` e depois declare uma função, assim:

```
var beanCounter = 10;  
  
function getNumberOfItems(ordertype) {  
    var beanCounter = 0;  
    if (ordertype == "order") {  
        // do some stuff with beanCounter...  
    }  
    return beanCounter;  
}
```

← Temos uma
global e uma
local!

Quando você faz isso, quaisquer referências a `beanCounter` dentro da função são para a variável local e não à global. Assim, dizemos que a variável global está escondida pela local (em outras palavras, não podemos vê-la porque a versão local está na frente).

↑ Observe que as variáveis local e global não têm efeito uma na outra: se você alterar uma, isto não tem efeito na outra. Elas são variáveis independentes.

Perguntas Ídiotas



Continuaríamos neste tópico se este fosse um livro para se aprofundar em programação JavaScript, mas dado que é o *Use a Cabeça! Programação em HTML5*, vamos apenas sugerir que você explore este tema no futuro para melhorar a qualidade do seu código!

P: Guardar os escopos de todas estas locais e globais é confuso, então por que não ficar apenas com globais? É o que eu sempre fiz.

R: Se você estiver escrevendo um código que seja complexo ou que precise ser mantido por um período longo de tempo, então realmente tem que ver como gerencia suas variáveis. Quando se é super zeloso na criação de variáveis globais, torna-se difícil registrar quando suas variáveis estão sendo usadas (e onde está fazendo alterações nos valores delas) e isso pode levar a um código com erros. Tudo isso se torna ainda mais importante quando você está escrevendo um código com outros colegas ou está usando bibliotecas de terceiros (embora se essas bibliotecas forem bem escritas, devem estar estruturadas para evitar esses problemas).

Assim, use globais onde fizer sentido, mas use-as com moderação e, sempre que possível, torne suas variáveis locais. À medida em que ganhar mais experiência com JavaScript, poderá investigar técnicas adicionais para estruturar código de modo que seja mais sustentável.

P: Tenho variáveis globais na minha página, mas estou carregando em outro arquivo JavaScript também. Esses arquivos têm conjuntos separados de variáveis globais?

R: Existe apenas um escopo global, de modo que todo arquivo que você carrega vê o mesmo conjunto de variáveis (e cria globais no mesmo espaço). É por isso que é tão importante que você tenha cautela com o uso de variáveis, para evitar colisões (e reduzir ou eliminar variáveis globais quando puder).

P: Vi códigos nos quais as pessoas não usam a palavra-chave `var` ao atribuir um valor a um novo nome de variável. Como isso funciona?

R: Sim, isso pode ser feito; quando você atribui um valor a um nome de variável que não tenha sido declarado anteriormente, ele é tratado como uma nova variável global. Tenha cuidado, então; se você fizer isso dentro de uma função, estará criando uma variável global. Observe que não recomendamos esta prática de codificação; ela é não apenas potencialmente confusa ao se ler o

código, como também algumas pessoas acham que este comportamento pode mudar algum dia nas implementações de JavaScript (o que provavelmente estragaria seu código).

P: Eu preciso definir uma função antes de usá-la, ou ela pode aparecer em qualquer lugar do meu script?

R: Declarações de função podem aparecer em qualquer lugar do seu script. Você pode declarar uma função abaixo de onde você a usa, se quiser. Isto funciona porque, quando você carrega pela primeira vez sua página, o navegador analisa todo o JavaScript na página (ou no arquivo externo) e vê a declaração da função antes de começar a executar o código. Você também pode colocar suas declarações de variáveis globais em qualquer lugar do seu script, embora recomendemos que declare todas as suas variáveis globais no topo dos seus arquivos, de modo que sejam fáceis de localizar.

Uma coisa a ter em mente ao usar mais de um arquivo JavaScript externo é que, se você tiver duas funções em diferentes arquivos com o mesmo nome, a função que o navegador vê por último é a que é usada.

P: Todos parecem reclamar do excesso de uso de variáveis globais em JavaScript. Por que isso? A linguagem foi mal projetada ou as pessoas não sabem o que estão fazendo? E o que fazer a respeito?

R: Globais são, muitas vezes, usadas em excesso em JavaScript. Um pouco disso é porque a linguagem facilita começar a codificar logo — isto é uma coisa boa — porque JavaScript não impõe muita estrutura ou overhead a você. A desvantagem é quando as pessoas escrevem código importante desta forma e ele tem que ser alterado e mantido a longo prazo (e isso mais ou menos descreve todas as páginas web). Tendo dito isto, JavaScript é uma linguagem poderosa e inclui recursos como objetos que você pode usar para organizar seu código de uma forma modular. Muitos livros foram escritos sobre este tópico, e lhe daremos uma pequena ideia sobre objetos na segunda metade deste capítulo (o que está a apenas algumas páginas de distância).

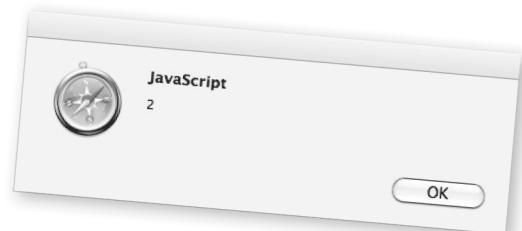
Ah, mencionamos que funções também são valores?

OK, você usa variáveis para armazenar números, valores booleanos, strings, matrizes, todos os tipos de coisas. Mencionamos que você também pode atribuir uma função a uma variável? Veja isto:

```
function addOne(num) {           Vamos definir uma função simples que
    return num + 1;             ↙ adiciona o número um ao seu argumento.
}
var plusOne = addOne;           ↙ Agora vamos fazer algo novo. Usaremos
                                o nome da função addOne e atribuiremos
                                addOne a uma nova variável, plusOne.
                                ↙ Observe que não está sendo chamada a função com
                                addOne(), estamos apenas usando a função name.
var result = plusOne(1);       ↙ plusOne é atribuída a uma função, de
                                modo que podemos chamar-la com um
                                argumento integer igual a 1.
                                ↙ Após esta chamada,
                                result vale 2.
```

Bom, nós não apenas não mencionamos este pequeno detalhe sobre funções antes, mas também não fomos totalmente francos quando lhe contamos sobre a anatomia de uma função — na verdade, você nem precisa dar um nome a ela. Isso mesmo: sua função pode ser *anônima*. O que isto significa e por que você iria querer tal coisa? Primeiro, vejamos como você cria uma função sem nome:

```
function(num) {               ↙ Aqui estamos criando uma função e não usando um nome...
    return num + 1;             ↙ hmm... mas como fazemos alguma coisa com ela?
}
var f = function(num) {         ↙ Vamos fazer de novo e, desta vez, atribuí-la a uma variável.
    return num + 1;
}
var result = f(1);             ↙ E depois podemos usá-la para chamar a função.
alert(result);                ↙
                                ↙ Após esta chamada,
                                o resultado é 2.
```





Dê uma olhada neste código: o que você acha que está acontecendo?

```
var element = document.getElementById("button");
element.onclick = function () {
    alert("clicked!");
}
```

Isto deve começar a parecer um pouco mais compreensível com o que já vimos...

→ Não se preocupe se ainda não tiver entendido 100% disto, chegaremos lá...

O que você pode fazer com funções como valores?

Então, o que é importante? Por que isto é útil? Bom, o importante não é tanto que podemos atribuir uma função a uma variável. Esta é apenas a nossa forma de mostrar a você que uma função *realmente é um valor*. Você sabe que pode armazenar valores em variáveis ou matrizes, pode passá-los como argumentos para funções ou, como veremos em breve, pode atribuí-los a propriedades de objetos. Em vez de lhe guiar pelas maneiras através das quais as funções anônimas são úteis, vamos apenas examinar uma das muitas formas de quando usar funções como valores começa a ficar interessante:

```
function init() {
    alert("you rule!");
}
window.onload = init;
```

→ Aqui está uma função init simples.

→ Aqui estamos atribuindo a função que definimos ao manipulador onload.

→ Veja, já estávamos usando funções como valores!

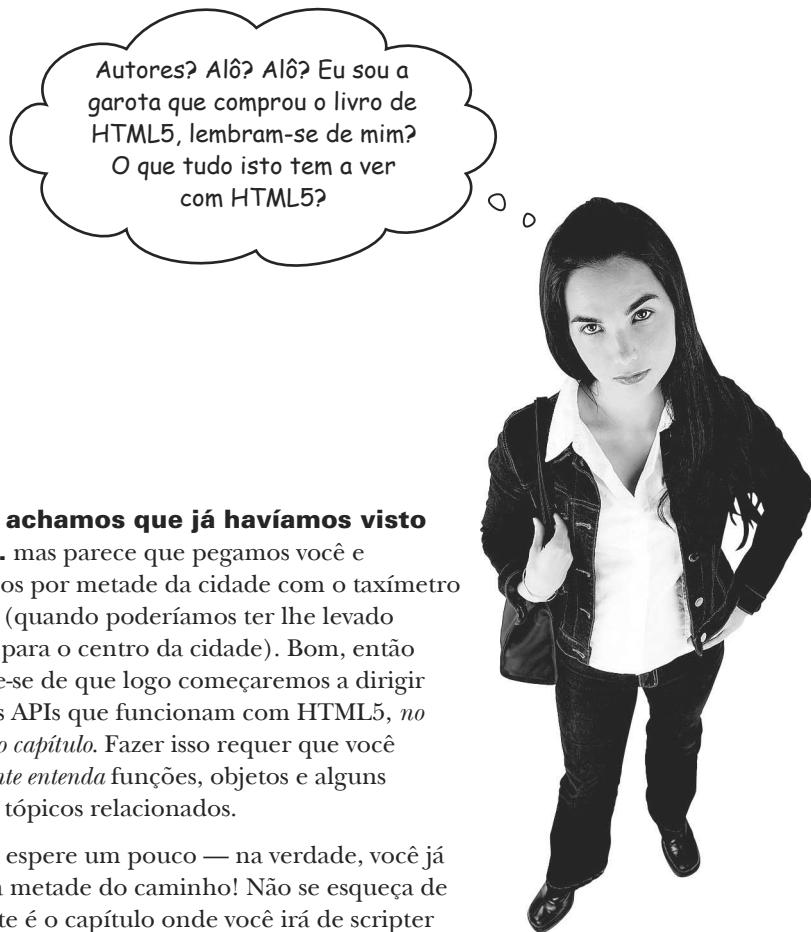
→ Aqui estamos criando uma função sem nome explícito e depois atribuindo seu valor diretamente à propriedade window.onload.

Ou melhor ainda:

→ Puxa, não é mais simples e legível?

→ Não se preocupe se window.onload ainda não estiver muito clara, em breve examinaremos isso tudo.

Você talvez esteja começando a ver que funções podem fazer algumas coisas além de apenas empacotar código para reuso; para lhe dar uma ideia melhor de como tirar vantagem integral de funções, veremos como elas se adaptam a JavaScript e depois juntaremos tudo.



Bom, achamos que já havíamos visto isso... mas parece que pegamos você e rodamos por metade da cidade com o taxímetro ligado (quando poderíamos ter lhe levado direto para o centro da cidade). Bom, então lembre-se de que logo começaremos a dirigir para as APIs que funcionam com HTML5, *no próximo capítulo*. Fazer isso requer que você *realmente entenda* funções, objetos e alguns outros tópicos relacionados.

Então, espere um pouco — na verdade, você já está na metade do caminho! Não se esqueça de que este é o capítulo onde você irá de scripter a programador, de um jóquei HTML/CSS para alguém capaz de criar aplicativos reais.

↑ Nós já mencionamos que isto provavelmente lhe trará muito mais dinheiro também?

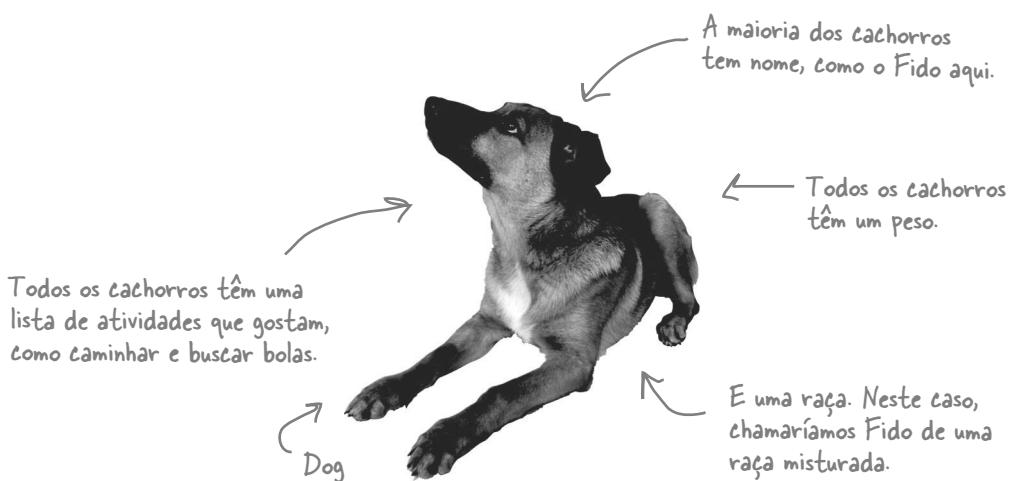
Com objetos, o futuro é tão brilhante que temos de usar óculos escuros...



Alguém disse “Objetos”?!

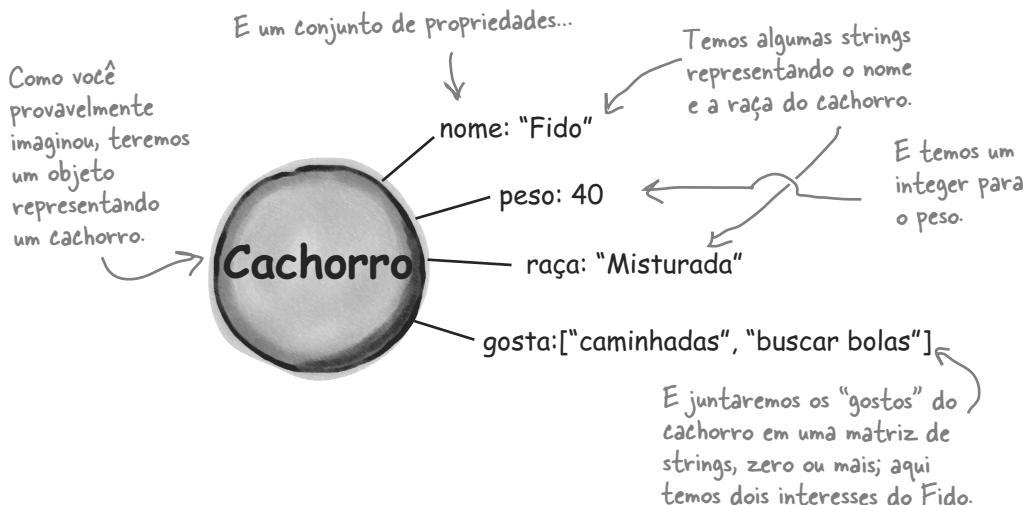
Ah, o nosso tópico favorito! Objetos levantarão suas habilidades de programação em JavaScript a um nível acima — eles são a chave para gerenciar código complexo, para entender o DOM, organizar seus dados e são até a forma fundamental na qual as APIs JavaScript HTML5 são empacotadas (e esta é apenas nossa lista curta!). Dito isso, objetos são um tópico difícil, certo? Ha! Vamos usar a cabeça primeiro e você os usará rapidamente.

Aqui está o segredo dos objetos em JavaScript: eles são apenas uma coleção de propriedades. Vamos pegar um exemplo, digamos, um cachorro. Um cachorro tem propriedades:



Pense em propriedades...

É claro que o Fido seria o primeiro a admitir que há muito mais nele do que apenas algumas propriedades, mas, para este exemplo, estas serão as que precisamos capturar em software. Vamos pensar nestas propriedades em termos de tipos de dados JavaScript:



Como criar um objeto em JavaScript

Assim, temos um objeto com algumas propriedades; como criamos isto usando JavaScript? Aqui está como:

```
var fido = {
    name: "Fido",
    weight: 40,
    breed: "Mixed",
    loves: ["walks", "fetching balls"]
};
```

Atribuiremos nosso objeto à variável fido.

Inicie um objeto com apenas a chave esquerda, depois todas as propriedades vão dentro.

Perceba que cada propriedade é separada por uma vírgula. NÃO um ponto e vírgula!

Este objeto possui quatro propriedades: nome, peso, raça e gostos.

Perceba que o valor do peso é um número, 40, e os valores de raça e nome são strings.

E, é claro, temos uma matriz para armazenar os gostos do cachorro.

Algumas coisas que você pode fazer com objetos

1 Acessar propriedades de objetos com a notação de "ponto":

```
if (fido.weight > 25) {
    alert("WOOF");
} else {
    alert("yip");
}
```

Use o objeto com um ":" e um nome de propriedade para acessar o valor dessa propriedade.

Aqui está o objeto... ... e, a seguir, o nome da propriedade.

fido.weight

2 Acessar propriedades usando uma string com a notação []:

```
var breed = fido["breed"];
if (breed == "mixed") {
    alert("Best in show");
}
```

Use o objeto, junto com o nome da propriedade entre aspas e em colchetes para acessar o valor dessa propriedade.

Agora use [] em torno do nome da propriedade. ↴

fido["weight"]

Aqui está o objeto... ... e o nome da propriedade entre aspas.

Achamos a notação de pontos a mais legível das duas.

3 Alterar o valor de uma propriedade:

```
fido.weight = 27;
fido.breed = "Chawalla/Great Dane mix";
fido.loves.push("chewing bones");
```

Estamos alterando o peso do Fido...

... sua raça...

... e adicionando um novo item à sua matriz de gostos.

push simplesmente adiciona um novo item ao final de uma matriz.

4 Enumerar todas as propriedades de um objeto:

```
var prop;
```

Para enumerar as propriedades, usaremos um for-in loop.

```
for (prop in fido) {
    alert("Fido has a " + prop + " property ");
    if (prop == "name") {
        alert("This is " + fido[prop]);
    }
}
```

Enumerar é passar por todas as propriedades do objeto.

Cada vez que percorremos o loop, a variável prop obtém o valor da string do nome da próxima propriedade.

E usamos a notação [] para acessar o valor dessa propriedade.

Observe que a ordem das propriedades é arbitrária, então não conte com uma determinada ordenação.

5 Divertir-se com uma matriz de objetos:

```
var likes = fido.loves;           ← Aqui, estamos atribuindo o valor dos
var likesString = "Fido likes";   gostos de fido à variável likes.

for (var i = 0; i < likes.length; i++) {
    likesString += " " + likes[i];
}

alert(likesString);             ← Podemos percorrer a matriz de
                                gostos e criar uma likesString
                                de todos os interesses de fido.

                                ← E podemos usar a string em um alert.
```

6 Passar um objeto para uma função:

```
function bark(dog) {           ← Podemos passar um objeto para
    if (dog.weight > 25) {     uma função da mesma forma que
        alert("WOOF");         qualquer outra variável.
    } else {
        alert("yip");
    }
}

bark(fido);                  ← E, na função, podemos acessar as
                            → propriedades do objeto normalmente,
                            → usando o nome do parâmetro para o
                            → objeto, claro.
```

Estamos passando fido como nosso argumento para a função bark (latir), que espera um objeto dog (cachorro).

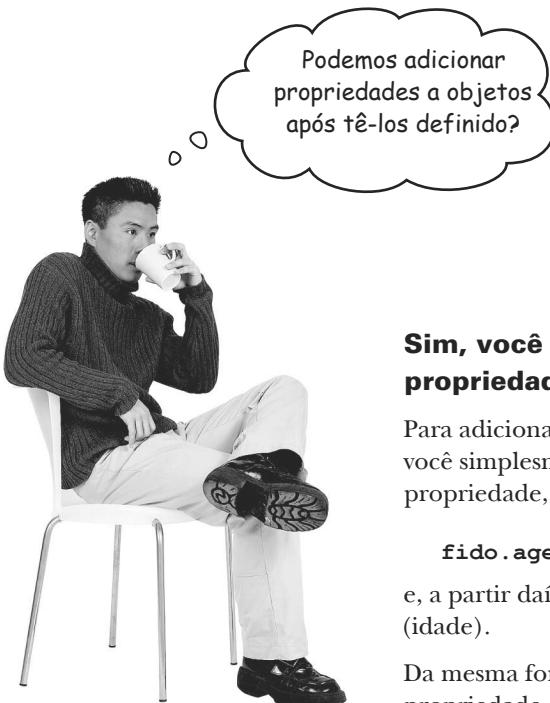


O Operador Ponto .

O operador ponto (.) lhe dá acesso às propriedades de um objeto. De modo geral, ele é mais fácil de ler do que a notação ["string"].

- fido.weight é o tamanho de fido.
- fido.breed é a raça de fido.
- fido.name é o nome de fido.
- fido.loves é uma matriz contendo os interesses de fido.





Sim, você pode adicionar ou excluir propriedades a qualquer momento.

Para adicionar uma propriedade a um objeto, você simplesmente atribui uma valor a uma nova propriedade, desta maneira:

```
fido.age = 5;
```

e, a partir daí, fido terá uma nova propriedade: age (idade).

Da mesma forma, você pode excluir qualquer propriedade com a palavra-chave delete, desta forma:

```
delete fido.age;
```

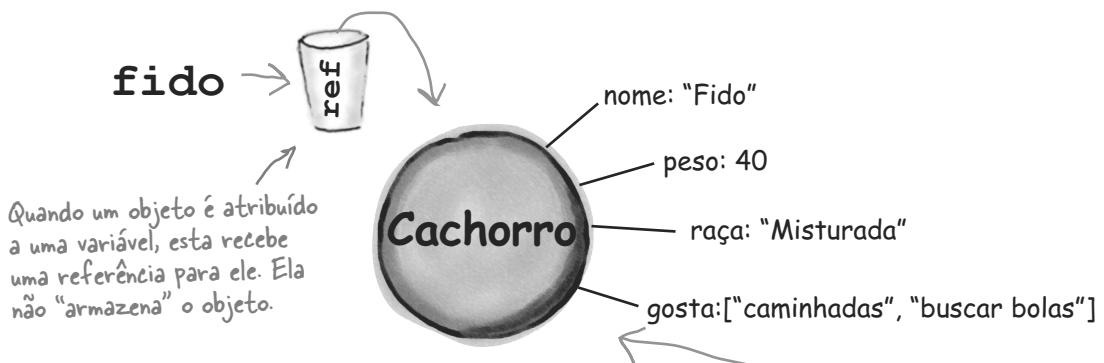
Quando você exclui uma propriedade, não está excluindo apenas o valor da mesma, está excluindo a própria propriedade. Na verdade, se você usar fido.age após excluí-la, ela será avaliada como undefined.

A expressão delete retorna true se a propriedade for excluída com sucesso (ou se você excluir uma propriedade que não existe, ou se o que você estiver tentando excluir não for uma propriedade de um objeto).

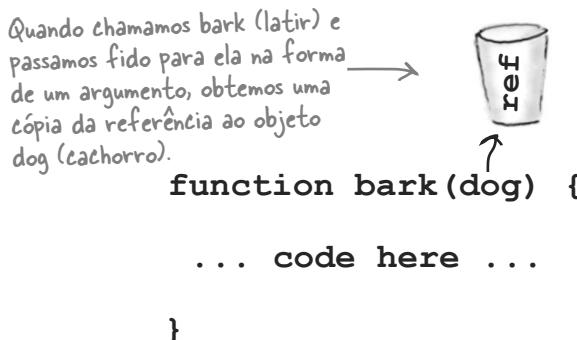
Vamos falar sobre passar objetos para funções

Já falamos um pouco sobre como argumentos são passados para funções — argumentos são passados *por valor*, de modo que, se você passar um integer, o parâmetro correspondente da função receberá uma *cópia* do *valor* desse integer para seu uso na função. As mesmas regras são verdadeiras para objetos, *porém*, temos que examinar um pouco mais de perto o que a variável armazena quando é atribuída a um objeto para sabermos o que isto significa.

Quando um objeto é atribuído a uma variável, esta armazena uma *referência* ao objeto, não o próprio objeto. Pense em uma referência como um ponteiro para um objeto.



Assim, quando você chamar uma função e passar um objeto para ela, estará passando uma referência para o objeto - não o próprio objeto, apenas um “ponteiro” para ele. Uma cópia da referência é passada para o parâmetro, o qual então aponta para o objeto original.



Então, o que tudo isso significa? Bom, quando você altera uma propriedade do objeto, está alterando a propriedade do objeto *original*, não uma cópia e, assim, verá todas as alterações que fizer em um objeto dentro e fora da sua função. Passaremos por um exemplo usando uma função `loseWeight` (perder peso) para dogs (cachorros)...

Colocando Fido em dieta...

Vamos ver o que acontece quando passamos fido para loseWeight (perder peso) e alteramos a propriedade dog.weight (peso de cachorro).

Nos
Bastidores



- Definimos um objeto, fido, e estamos passando-o para uma função, loseWeight.

fido é uma referência a um objeto, o que significa que ele não está na variável fido, mas é apontado por ela.

`fido.weight = 48;`

`...
loseWeight(fido);`

Quando passamos fido para uma função, estamos passando a referência para o objeto.



- O parâmetro dog da função loseWeight recebe uma cópia da referência a fido. Assim, quaisquer alterações nas propriedades do parâmetro afetam o objeto que foi passado.

Quando passamos fido para loseWeight, o que é atribuído ao parâmetro dog é uma cópia da referência, não do objeto. Assim, fido e dog apontam para o mesmo objeto.

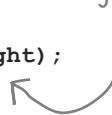
```
function loseWeight(dog) {
  dog.weight = dog.weight - 10;
}
```

```
alert(fido.name + " now weighs " + fido.weight);
```

A referência a dog é uma cópia da referência a fido.



Assim, quando subtraímos 10 de dog.weight, estamos alterando o valor de fido.weight.



EM EXIBIÇÃO NO CINE WEBVILLE

O Cine Webville procurou nossa ajuda com sua API JavaScript. Vamos começar de forma simples e projetar o objeto do filme para eles. O que precisamos é de alguns objetos filme, que incluem, cada um, título, gênero, avaliação e os horários de exibição. Siga em frente e esboce o seu projeto de objeto filme aqui (você pode usar nosso objeto dog como modelo). Aqui estão alguns dados de exemplo, que você pode usar para colocar nos seus objetos:

Plan 9 from Outer Space, às 15h, 19h e 23h; é do gênero “cult classic”; e tem avaliação de 2 estrelas.

Forbidden Planet, às 17h e às 21h ; seu gênero é “ficção científica clássica”; avaliação 5 estrelas.

A solução está na próxima página, mas não olhe até ter terminado o exercício. Sério.

Projete os seus objetos aqui.



Sinta-se livre para adicionar os seus favoritos em vez destes.



EM EXIBIÇÃO NO CINE WEBVILLE

SOLUÇÃO

Como foi a criação do seu objeto filme?

Aqui está a nossa solução:

Criamos dois objetos, movie1 (filme1) e movie2 (filme2), para os dois filmes.

```
var movie1 = {
  title: "Plan 9 from Outer Space",
  genre: "Cult Classic",
  rating: 5,
  showtimes: ["3:00pm", "7:00pm", "11:00pm"]
};
```

movie1 tem quatro propriedades: title (título), genre (gênero), rating (nota) e showtime (horário de exibição).

title e genre são strings.

rating é um número.

E showtime é uma matriz que contém os horários do filme na forma de strings.

```
var movie2 = {
  title: "Forbidden Planet",
  genre: "Classic Sci-fi",
  rating: 5,
  showtimes: ["5:00pm", "9:00pm"]
};
```

movie2 também tem quatro propriedades: title (título), genre (gênero), rating (nota) e showtime (horário de exibição).

Lembre-se de separar suas propriedades com vírgulas.

Usamos os mesmos nomes de propriedade, mas diferentes valores de movie1.



Nossa próxima exibição é às...

Já tivemos uma ideia do que significa misturar objetos e funções. Vamos mais a fundo, escrevendo um pouco de código que nos informe quando será a próxima exibição de um filme. Nossa função receberá um filme como argumento e retornará uma string contendo o próximo horário, baseada no horário atual.



Aqui está nossa nova função, que recebe um objeto movie.

```
function getNextShowing(movie) {
    var now = new Date().getTime();
```

Estamos pegando o horário corrente usando o objeto Date de JavaScript. Não nos preocuparemos sobre os detalhes dele ainda, mas já sabemos que retorna o horário corrente em milissegundos.

```
for (var i = 0; i < movie.showtimes.length; i++) {
    var showtime = getTimeFromString(movie.showtimes[i]);
    if ((showtime - now) > 0) {
```

Agora use a matriz de movie, showtimes, e itere pelos horários de exibição.

```
        return "Next showing of " + movie.title + " is " + movie.showtimes[i];
    }
}
```

Para cada horário de exibição, obtemos seu horário em milissegundos e depois compáramos.

```
return null;
}
```

Se não houver nenhuma exibição, apenas retornamos nulo.

```
function getTimeFromString(timeString) {
    var theTime = new Date();
    var time = timeString.match(/(\d+)(?::(\d\d))?\s*(p?)/);
    theTime.setHours( parseInt(time[1]) + (time[3] ? 12 : 0) );
    theTime.setMinutes( parseInt(time[2]) || 0 );
    return theTime.getTime();
}
```

Não se preocupe com este código; ele usa expressões regulares, o que você aprenderá posteriormente no seu estudo em JavaScript. Por enquanto, apenas use-as!

```
var nextShowing = getNextShowing(movie1);
alert(nextShowing);
nextShowing = getNextShowing(movie2);
alert(nextShowing);
```



Código Pronto para Assar

Aqui está um pouco de código pronto para assar que recebe uma string com o formato como 1am ou 3pm e a converte para um horário em milissegundos.

Agora usamos a função chamando getNextShowing (pegar próxima exibição) e usamos a string que ela retorna em um alert

E fazemos isso de novo com movie2.



Como o "Encadeamento" funciona ...

Você entendeu isto no código anterior?

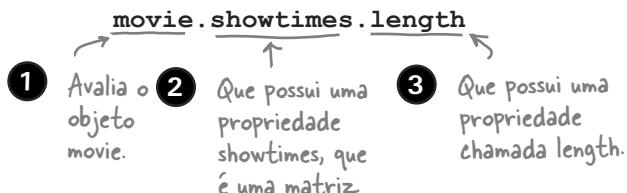
```
movie.showtimes.length
```

Isto não se parece com nada que tenhamos visto antes. É, na verdade, apenas um atalho para uma série de passos que poderíamos ter dado para obter o comprimento da matriz showtimes dos objetos movie. Poderíamos ter escrito o seguinte:

```
var showtimesArray = movie.showtimes;
var len = showtimesArray.length;
```

← Primeiro, pegamos a matriz showtimes.
← Depois, a usamos para acessar a propriedade length.

Podemos, porém, fazer tudo isso de uma só vez encadeando as expressões. Vamos ver como isso funciona:

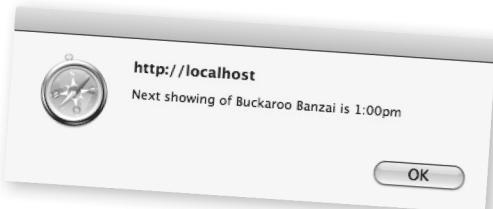


Testando no drive-in

Digite o código da página anterior e faça um teste com ele. Você verá que a função getNextShowing recebe qualquer filme e descobre seu próximo horário de exibição. Sinta-se à vontade para criar alguns objetos movie novos e testá-los também. Nós fizemos isso, no nosso horário local de 12:30pm:

```
var banzaiMovie = {
  title: "Buckaroo Banzai",
  genre: "Cult classic",
  rating: 5,
  showtimes: ["1:00pm", "5:00pm", "7:00pm"]
}

var nextShowing = getNextShowing(banzaiMovie);
alert(nextShowing);
```



↑ ↗ Observação: nosso código não tem muita qualidade de "código de produção"; se você executá-lo após a última exibição do filme, ele retornará nulo. Tente novamente amanhã. ☺

Objetos podem ter comportamento também...

Você não achou que objetos servissem apenas para armazenar números, strings e matrizes, achou? Objetos são ativos, podem fazer coisas. Cachorros não ficam apenas sentados: eles latem, correm, brincam de pegar. Por isso, um objeto cachorro deveria ser ativo também! Dado tudo o que você aprendeu neste capítulo, está pronto para adicionar comportamento aos seus objetos. Aqui está como fazemos isso:

```
var fido = {  
    name: "Fido",  
    weight: 40,  
    breed: "Mixed",  
    loves: ["walks", "fetching balls"]  
    bark: function() {  
        alert("Woof woof!");  
    }  
};
```

↑

Em vez de dizer que isto é uma “função no objeto”, apenas dizemos que é um método. Eles são a mesma coisa, mas todos se referem a funções de objetos como métodos.

↑

Perceba que estamos usando uma função anônima e atribuindo-a à propriedade bark do objeto.

←

Podemos adicionar uma função diretamente ao nosso objeto, desta maneira.

Quando um
objeto possui
uma função,
dizemos que
esse objeto
possui um
método.

Para chamar um método em um objeto, usamos o nome do objeto junto com o do método com a nossa notação de ponto e fornecemos os argumentos necessários.

fido.bark();

Dizemos para um objeto fazer alguma coisa, chamando métodos nele. Neste caso, estamos chamando o método bark do fido.



Enquanto isso, de volta ao Cine Webville...

Agora que o seu conhecimento sobre objetos está se expandindo, podemos voltar e melhorar o código do cinema. Já escrevemos uma função `getNextShowing`, que recebe um filme como argumento, mas poderíamos, em vez disso, fazer dela uma parte do objeto `movie`, tornando-a um método. Vamos fazer isso:

```
var movie1 = {
    title: "Plan 9 from Outer Space",
    genre: "Cult Classic",
    rating: 5,
    showtimes: ["3:00pm", "7:00pm", "11:00pm"],
    getNextShowing: function(movie) {
        var now = new Date().getTime();

        for (var i = 0; i < movie.showtimes.length; i++) {
            var showtime = getTimeFromString(movie.showtimes[i]);
            if ((showtime - now) > 0) {
                return "Next showing of " + movie.title + " is " +
                    movie.showtimes[i];
            }
        }
        return null;
    }
};
```



Pegamos nosso código e o colocamos em um método do objeto `movie1` com o nome de propriedade `getNextShowing`.

Sabemos, porém, que isso pode não estar muito certo...

Na verdade, não podemos simplesmente jogar a função neste objeto porque `getNextShowing` recebe um argumento `movie`, e o que realmente queremos é chamar `getNextShowing` desta forma:

```
var nextShowing = movie1.getNextShowing();
```

Nenhum argumento deve ser necessário aqui; está claro de qual filme queremos a próxima exibição, ou seja, queremos `movie1`.

Tudo bem. Então como consertaremos isso? Temos que remover o parâmetro da definição do método `getNextShowing`, no entanto precisamos fazer algo com todas as referências a `movie.showtimes` no código, pois, assim que removermos o parâmetro, `movie` não existirá mais como uma variável. Vejamos...

Vamos tirar o parâmetro do filme...

Tomamos a liberdade de remover o parâmetro movie e todas as suas referências, o que nos deixa com este código:

```
var movie1 = {  
    title: "Plan 9 from Outer Space",  
    genre: "Cult Classic",  
    rating: 5,  
    showtimes: ["3:00pm", "7:00pm", "11:00pm"],  
    getShowing: function() {  
        var now = new Date().getTime();  
  
        for (var i = 0; i < showtimes.length; i++) {  
            var showtime = getTimeFromString(showtimes[i]);  
            if ((showtime - now) > 0) {  
                return "Next showing of " + title + " is " + showtimes[i];  
            }  
        }  
        return null;  
    }  
};
```

← Destacamos as alterações abaixo...

↑ Tudo isso parece muito razoável, mas precisamos pensar em como o método `getNextShowing` usará a propriedade `showtimes`...
... estamos acostumados a variáveis locais (que `showtimes` não é) e globais (que `showtimes` não é). Hmm...

↑ Ah, e aqui está outra, a propriedade `title`.

E agora?

Tudo bem. Aqui está a charada: temos estas referências às propriedades `showtimes` e `title`. Normalmente em uma função, estamos referenciando uma variável local, uma variável global ou um parâmetro da função, mas `showtimes` e `title` são *propriedades* do objeto `movie1`. Bom, talvez isto funcione... Será que JavaScript pode ser suficientemente esperta para descobrir?

Não. Não funciona. Sinta-se à vontade para testar; JavaScript lhe informará que as variáveis `showtimes` e `title` estão indefinidas. Como pode ser?

OK, aqui está o que faremos: estas variáveis são propriedades de um objeto, mas não estão informando a JavaScript de qual objeto. Você talvez esteja dizendo para si mesmo “Bom, obviamente queremos dizer ESTE objeto, este bem aqui! Como poderia haver confusão a respeito?” Queremos ainda as propriedades deste objeto. Na verdade, há uma palavra-chave em JavaScript chamada `this` (isto) e é exatamente com ela que você informará à JavaScript. *This* quer dizer *o objeto no qual estamos*.

A situação, na verdade, é um pouco mais complicada do que parece, (chegaremos lá em um segundo), mas, por enquanto, adicionaremos a palavra-chave `this` e faremos este código funcionar.

Adicionando a palavra-chave “this”

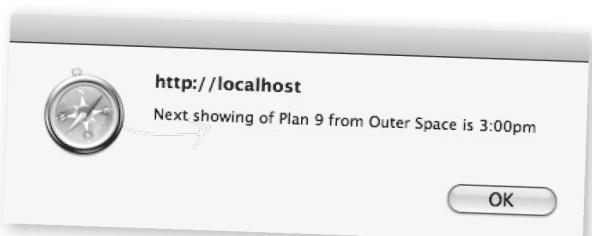
Adicionaremos `this` a cada local em que especificarmos uma propriedade. Deste modo, informaremos à JavaScript que queremos a propriedade *deste* objeto:

```
var movie1 = {
    title: "Plan 9 from Outer Space",
    genre: "Cult Classic",
    rating: 5,
    showtimes: ["3:00pm", "7:00pm", "11:00pm"], Aqui adicionamos uma
                                                palavra-chave this antes
                                                de cada propriedade para
                                                dizer que queremos a
                                                referência ao objeto movie1.
    getNextShowing: function() {
        var now = new Date().getTime(); ↙
        for (var i = 0; i < this.showtimes.length; i++) {
            var showtime = getTimeFromString(this.showtimes[i]);
            if ((showtime - now) > 0) {
                return "Next showing of " + this.title + " is " +
                    this.showtimes[i];
            }
        }
        return null;
    }
};
```

Um test drive com “this”

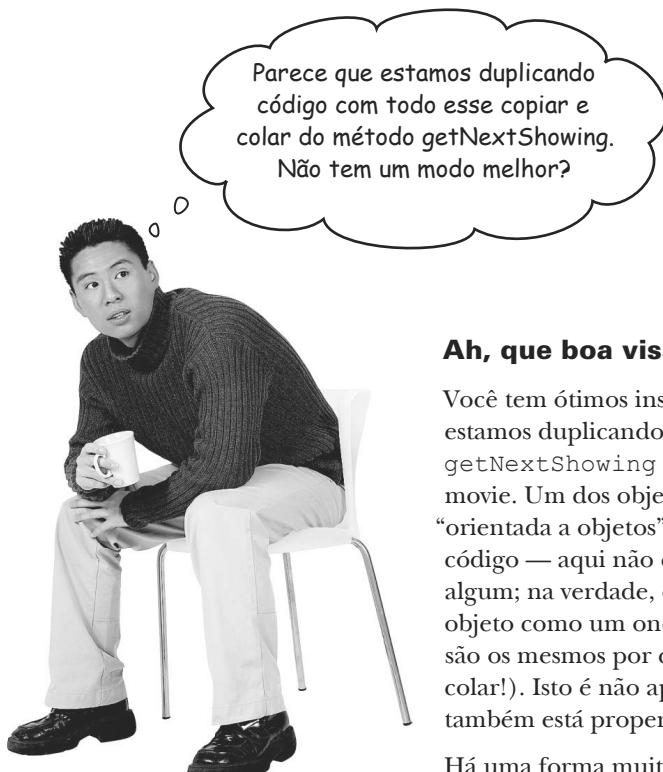


Siga em frente, digite o código acima e adicione a função `getNextShowing` ao seu objeto `movie2` (apenas copie e cole). Depois, faça as alterações abaixo no seu código de teste anterior. Execute-o! Aqui está o que obteremos:



```
var nextShowing = movie1.getNextShowing();
alert(nextShowing);
nextShowing = movie2.getNextShowing();
alert(nextShowing);
```

*↑ Observe que agora estamos chamando
getNextShowing NO objeto. Faz mais sentido, não?*



Ah, que boa visão.

Você tem ótimos instintos se percebeu que estamos duplicando código, quando copiamos `getNextShowing` em mais de um objeto `movie`. Um dos objetivos da programação “orientada a objetos” é maximizar o reuso de código — aqui não estamos reusando código algum; na verdade, estamos criando cada objeto como um one-off e nossos objetos `movie` são os mesmos por convenção (e por copiar e colar!). Isto é não apenas um desperdício, como também está propenso a erros.

Há uma forma muito melhor de fazer isso usando um *construtor*. O que é um construtor? É apenas uma função especial que escreveremos e que pode criar objetos para nós, tornando-os os mesmos. Pense nisso como uma pequena fábrica, que recebe valores de propriedades que você quer configurar no seu objeto e depois lhe devolve um novo e belo objeto com todas as propriedades e métodos certos.

Vamos criar um construtor...

Como criar um construtor

Vamos criar um construtor para dogs. Já sabemos como queremos que nossos objetos dogs se pareçam: eles têm propriedades name, breed e weight, e têm um método bark. Assim, o que nosso construtor precisa é receber valores de propriedades como parâmetros e depois nos dar um objeto dog pronto para latir. Aqui está o código:

```

function Dog(name, breed, weight) {
    this.name = name;
    this.breed = breed;
    this.weight = weight;
    this.bark = function() {
        if (this.weight > 25) {
            alert(this.name + " says Woof!");
        } else {
            alert(this.name + " says Yip!");
        }
    };
}
  
```

Uma função construtora se parece muito com uma normal. Por convenção, porém, dâmos ao nome da função uma letra maiúscula.

Os nomes de propriedade e os nomes de parâmetros não têm que ser os mesmos, mas muitas vezes são — por convenção.

Os parâmetros do construtor recebem valores para as propriedades que queremos que nossos objetos tenham.

Aqui, estamos inicializando as propriedades do objeto com os valores que foram passados para o construtor.

Podemos incluir o método bark no objeto que estamos criando, inicializando a propriedade bark com um valor de função, como temos feito.

Precisamos usar "this.weight" e "this.name" no método para nos referir às propriedades do objeto, como fizemos antes.

Perceba como a sintaxe difere da sintaxe de objetos. Estes são comandos, de modo que precisamos terminar cada um deles com um ";", da mesma forma que normalmente fazemos em uma função.

Assim, vamos examinar isto novamente para assegurar que esteja tudo certo. Dog é uma função construtora e recebe um conjunto de argumentos, que são os valores iniciais das propriedades que queremos: name, breed e weight. Assim que tiver esses valores, ela os atribui às propriedades usando a palavra-chave this. Também define nosso método bark. O resultado de tudo isso? O construtor Dog retorna um objeto novo. Vejamos como realmente usar o construtor.

Não se preocupe em
criar todos esses
objetos; nós os
construiremos para você.



Agora usaremos nosso construtor

Agora que temos a nossa fábrica, podemos usá-la para criar alguns dogs. Só há uma coisa que não lhe dissemos: que você precisa chamar uma função construtora de uma forma especial, colocando a palavra-chave `new` (novo) antes da chamada. Aqui estão alguns exemplos:

Para criar um dog, usamos a palavra-chave `new` com o construtor.

```
var fido = new Dog("Fido", "Mixed", 38);
var tiny = new Dog("Tiny", "Chawalla", 8);
var clifford = new Dog("Clifford", "Bloodhound", 65);
```

E depois o chamamos da mesma forma que qualquer função.

```
fido.bark();
tiny.bark();
clifford.bark();
```

Assim que temos os objetos,
podemos chamar seus
métodos `bark` para fazer
cada Cachorro latir.

Estamos criando
três objetos `Dog`
diferentes, passando
diferentes argumentos
para personalizar cada
cachorro.

Vamos revisar o que está acontecendo aqui mais uma vez: estamos criando três objetos `Dog`, cada um com suas próprias propriedades, usando a palavra-chave `new` com o construtor `Dog` que criamos. O construtor retorna um objeto `Dog` personalizado com os argumentos passados.

A seguir, chamamos o método `bark` em cada um — perceba que estamos compartilhando o mesmo método `bark` em todos os `Dog`s e, quando cada cachorro late, `this` aponta para o objeto `Dog` que fez a chamada. Assim, se chamarmos o método `bark` em `fido`, então, no método `bark`, `this` é configurado para o objeto `fido`. Vejamos um pouco mais de perto como isso acontece.



Como "this" realmente funciona?

Sempre que colocamos `this` no código de um método, ele será interpretado como uma referência ao objeto no qual o método foi chamado. Assim, se chamarmos `fido.bark`, então `this` referenciará `fido`. Ou, se o chamarmos no nosso objeto `dog tiny`, então `this` referenciará `tiny` dentro dessa chamada de método. Como `this` sabe qual objeto está representando? Vejamos:

Nos
Bastidores



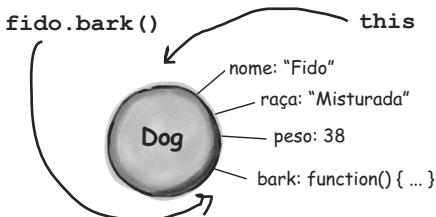
- Digamos que temos um objeto `dog` atribuído a `fido`:

```
fido = new Dog("Fido", "Mixed", 38);
```



Aqui está o nosso novo objeto `dog` instanciado com os valores de propriedades que queremos.

- E chamamos `bark()` em `fido`:

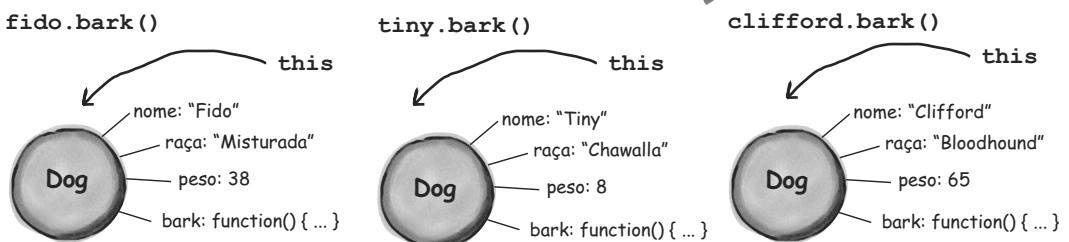


Sempre que chamamos um método em um objeto, JavaScript configura `this` para apontar para o próprio objeto. Assim, aqui, `this` aponta para `fido`.

E assim, quando nos referimos a `this.name`, sabemos que o nome é "Fido".

- Assim, "this" sempre se refere ao objeto no qual o método foi chamado, não importa quantos dogs tenhamos criado para latir:

Você pode chamar `bark` em qualquer objeto `dog` e `this` será atribuído ao `dog` específico antes que o seu código do corpo seja executado.





Ímãs de Geladeira

Uma função construtora Movie funcionando estava na geladeira, mas alguns dos ímãs caíram no chão. Você pode ajudar a juntar tudo? Tenha cuidado, pois alguns ímãs extras já estavam no chão e podem lhe distrair.

```
function _____(_____, _____, rating, showtimes) {  
    this.title = _____;  
    this.genre = genre;  
    this._____ = rating;  
    this.showtimes = _____;  
    this.getNextShowing = function() {  
        var now = new Date().getTime();  
        for (var i = 0; i < _____.length; i++) {  
            var showtime = getTimeFromString(this._____ [i]);  
            if ((showtime - now) > 0) {  
                return "Next showing of " + _____ + " is " + this.  
showtimes[i];  
            }  
        }  
    }  
}
```

✓ Usamos estes ímãs para
completar o código.

title

function

Movie

Woof

rating

showtimes

this.showtimes

this.title

bark()

genre

this

;

,

Perguntas Idiotas

não existem

P: Qual a diferença real entre uma função e um método? Afinal, se eles são a mesma coisa, por que chamá-los de modo diferente?

R: Por convenção, se um objeto tiver uma função, chamamos de método. Ambos funcionam da mesma forma, exceto quando você chama o método de um objeto, usando o operador ponto, e um método pode usar `this` para acessar o objeto, no qual o método é chamado. Pense em uma função como uma peça independente de código, que você pode chamar, e um método como comportamento, que é anexado a um objeto específico.

P: Então, quando eu crio objetos com um construtor e esses objetos têm um método, todos esses objetos compartilham o mesmo código para esse método?

R: É verdade, e esta é uma das vantagens da programação orientada a objetos: você pode criar o código para essa classe de objetos (digamos todos os seus objetos `dog`) em um lugar e todos os `dogs` o compartilham. A forma pela qual você especifica para cada `dog` é com suas propriedades e usando `this` para acessar essas propriedades.

P: Posso configurar `this` com um valor da minha escolha e, se o fizer, isso vai bagunçar tudo?

R: Não, você não pode configurar `this` para nada. Lembre-se de que `this` é uma palavra-chave, não uma variável! Ela se parece e age como uma, mas não é uma variável.

P: `this` tem um valor fora do seu método de objeto?

R: Não, se você estiver chamando um método de objeto, então `this` é indefinido.

P: Então a forma de pensar em `this` é quando chamo um método em um objeto. O valor de `this` é configurado com esse objeto o tempo inteiro em que o método estiver sendo avaliado?

R: Dentro do corpo do objeto, sim, `this` sempre será o próprio objeto. Há alguns casos avançados em que isso pode não ser verdadeiro; por exemplo, as coisas ficam mais complicadas quando você tem objetos dentro de objetos, e se você começar a fazer isso, precisará examinar a semântica, mas esta é uma boa regra geral.

P: Ouvi que, na programação orientada a objetos, posso ter classes de objetos e elas podem herdar umas das outras. Eu poderia ter uma classe `mamíferos`, da qual `gatos` e `cachorros` herdassem. Eu posso fazer isso em JavaScript?

R: Pode. JavaScript usa algo chamado herança de protótipo, que é mais poderosa ainda do que os modelos baseados estritamente em classes. Chegar à herança de protótipos está um pouco além do escopo deste livro, mas, quem sabe, não possamos vir a ser convencidos a escrever mais sobre JavaScript?

P: Então, quando dizemos `new Date()`, estamos usando um construtor, certo?

R: Sim, é isso mesmo! `Date` é um construtor interno em JavaScript. Quando você diz `new Date()`, obtém um objeto `Date` com alguns métodos úteis que pode usar para manipular a data.

P: Qual a diferença entre objetos que escrevemos nós mesmos e os que criamos com construtores?

R: A principal diferença é como você os cria. Objetos que você mesmo escreve, usando chaves e propriedades separadas por vírgulas, são conhecidos como “literais de objetos”. Você literalmente os digita no seu código! Se você quiser outro como esse, tem que digitá-lo você mesmo e assegurar que tenha as mesmas propriedades. Objetos criados por um construtor são criados usando `new` e uma função construtora, que retorna o objeto. Você pode usar a função construtora para criar muitos objetos que tenham as mesmas propriedades, mas valores diferentes nelas, se quiser.



Ímãs de Geladeira

Uma função construtora Movie funcionando estava na geladeira, mas alguns dos ímãs caíram no chão. Você pode ajudar a juntar tudo? Tenha cuidado, pois alguns ímãs extras já estavam no chão e podem lhe distrair.

Este é um construtor, então estamos usando "Movie" para o nome.

```

function Movie(title, genre, rating, showtimes) {
    this.title = title;
    this.genre = genre;
    this.rating = rating; ← Passamos valores para as propriedades que queremos customizar: title, genre, rating e showtimes...
    this.showtimes = showtimes;
    this.getNextShowing = function() { ← ... e inicializar as propriedades.
        var now = new Date().getTime();
        for (var i = 0; i < this.showtimes.length; i++) {
            var showtime = getTimeFromString(this.showtimes[i]);
            if ((showtime - now) > 0) {
                return "Next showing of " + this.title + " is " +
                    this.showtimes[i];
            }
        }
    } ← Não se esqueça de acabar esta declaração com um ponto e vírgula.
}

```

↙ Sobras de imãs.

function this.showtimes Woof
 , bark() this

Faça o test drive do seu construtor direto do chão da fábrica



Agora que você tem um construtor para Movie, está na hora de criar alguns objetos Movie! Vá em frente. Digite a função construtora de Movie, adicione o código abaixo e execute seu construtor. Achamos que você concordará que esta é uma forma muito mais fácil de criar objetos.

```
var banzaiMovie = new Movie("Buckaroo Banzai",
                            "Cult Classic",
                            5,
                            ["1:00pm", "5:00pm", "7:00pm", "11:00pm"]);
```

Observe que podemos colocar os valores da matriz com os horários de exibição diretamente na chamada da função.

```
var plan9Movie = new Movie("Plan 9 from Outer Space",
                           "Cult Classic",
                           2,
                           ["3:00pm", "7:00pm", "11:00pm"]);
```

← A seguir, Plan 9 from Outer Space...

```
var forbiddenPlanetMovie = new Movie("Forbidden Planet",
                                      "Classic Sci-fi",
                                      5,
                                      ["5:00pm", "9:00pm"]);
```

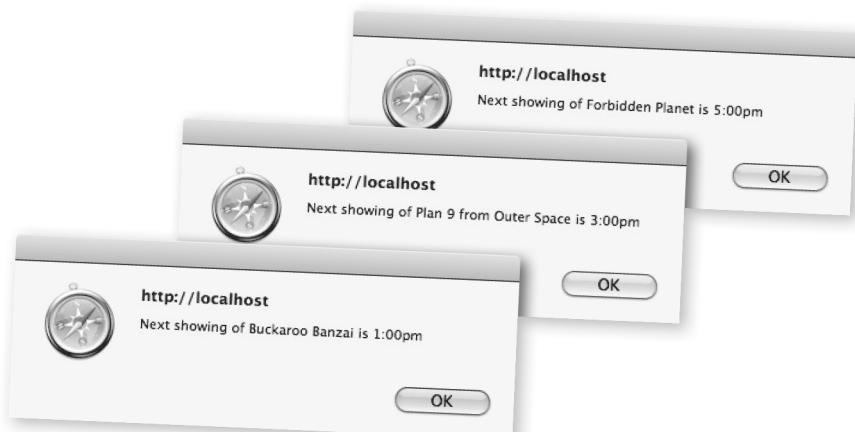
```
alert(banzaiMovie.getNextShowing());
alert(plan9Movie.getNextShowing());
alert(forbiddenPlanetMovie.getNextShowing());
```

Primeiro criaremos um objeto movie para o filme Buckaroo Banzai (um dos nossos clássicos cult favoritos). Passaremos valores para os parâmetros.

← A seguir, Plan 9 from Outer Space...

← E, é claro, Forbidden Planet...

← Assim que você tiver todos os seus objetos criados, poderá chamar o método getNextShowing e alertar o usuário sobre os próximos horários de exibição.





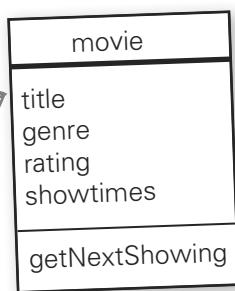
Parabéns, você passou pelas funções e objetos! Agora que sabe tudo sobre eles, e antes de terminarmos o capítulo, vamos gastar um pouco de tempo verificando objetos JavaScript no seu ambiente, ou seja, no seu habitat natural, o navegador!

Agora, você talvez tenha começado a perceber...

... que objetos estão por toda parte. Por exemplo, document e window são objetos, assim como os elementos que obtemos de volta de document.getElementById. Estes são apenas alguns dos muitos objetos que encontraremos — quando chegarmos às APIs HTML5, veremos objetos em todos os lugares!

Vamos ver novamente alguns dos objetos que você tem usado neste livro:

Aqui está o nosso próprio objeto movie.



Desenhamos objetos como este para mostrar propriedades no topo...

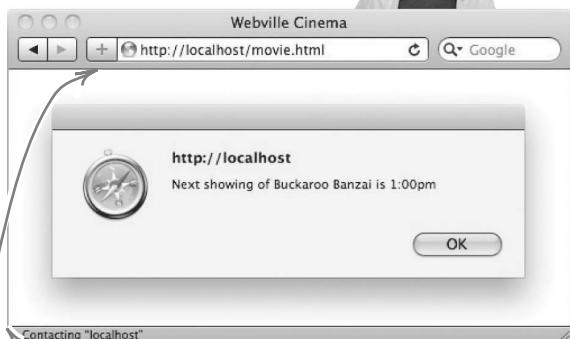
... e métodos na parte de baixo. Assim, você obtém um resumo do objeto, suas propriedades e métodos em um instante.

Alguns dos objetos com os quais já nos deparamos.

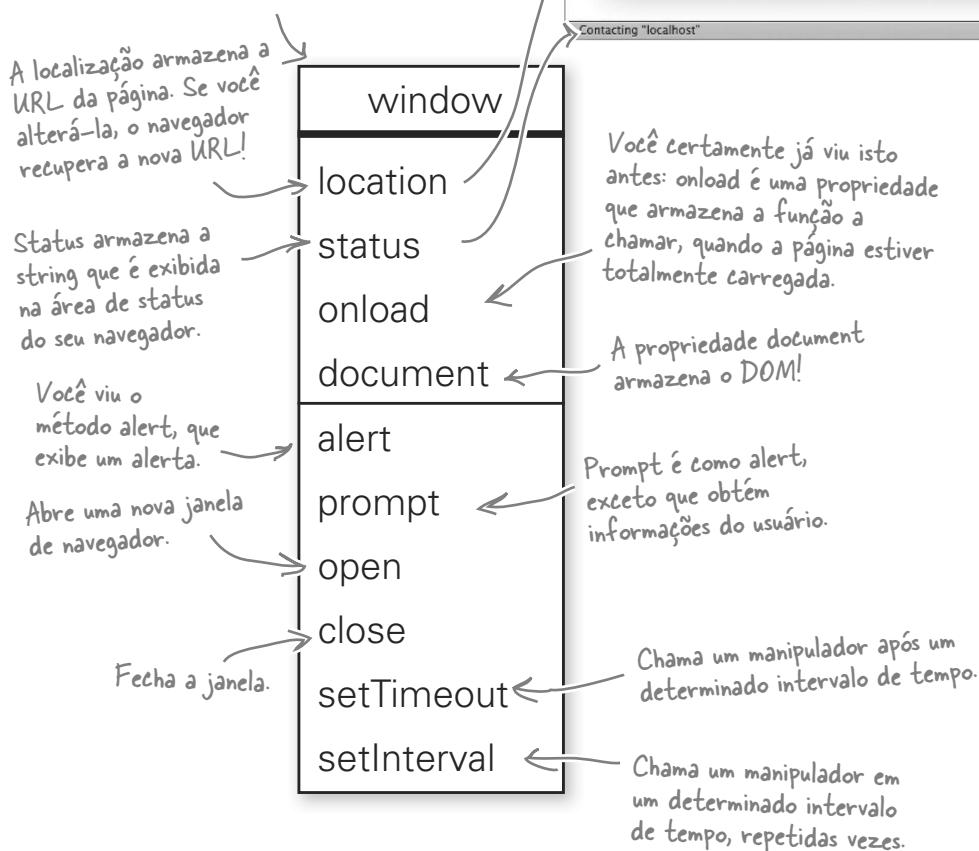


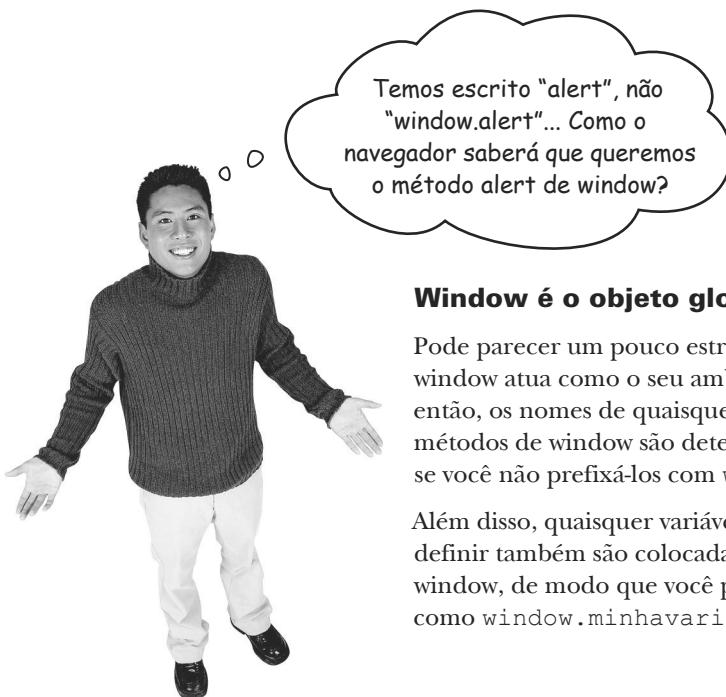
O que é o objeto window mesmo?

Quando você estiver escrevendo código para o navegador, o objeto window sempre fará parte da sua vida. O objeto window representa tanto o ambiente global para seus programas JavaScript quanto a janela principal do seu aplicativo e, como tal, contém muitas propriedades e métodos centrais. Vamos examiná-lo:



Aqui está nosso objeto window com algumas propriedades e métodos importantes que você irá querer conhecer. Há muitos outros...





Temos escrito "alert", não
"window.alert"... Como o
navegador saberá que queremos
o método alert de window?

Window é o objeto global.

Pode parecer um pouco estranho, mas o objeto window atua como o seu ambiente global, então, os nomes de quaisquer propriedades ou métodos de window são determinados, mesmo se você não prefixá-los com window.

Além disso, quaisquer variáveis globais que você definir também são colocadas no namespace window, de modo que você pode referenciá-las como `window.minhavariavel`.

Um exame mais de perto em `window.onload`

Uma coisa que usamos frequentemente até aqui neste livro é um manipulador de eventos `window.onload`. Atribuindo uma função à propriedade `window.onload`, podemos assegurar que nosso código não seja executado até que a página seja carregada e o DOM esteja completamente configurado. Há muita coisa acontecendo no comando `window.onload`. Então, o examinaremos novamente e as peças começarão a se encaixar:

Aqui está nosso
objeto window global.

`onload` é uma propriedade
do objeto window.

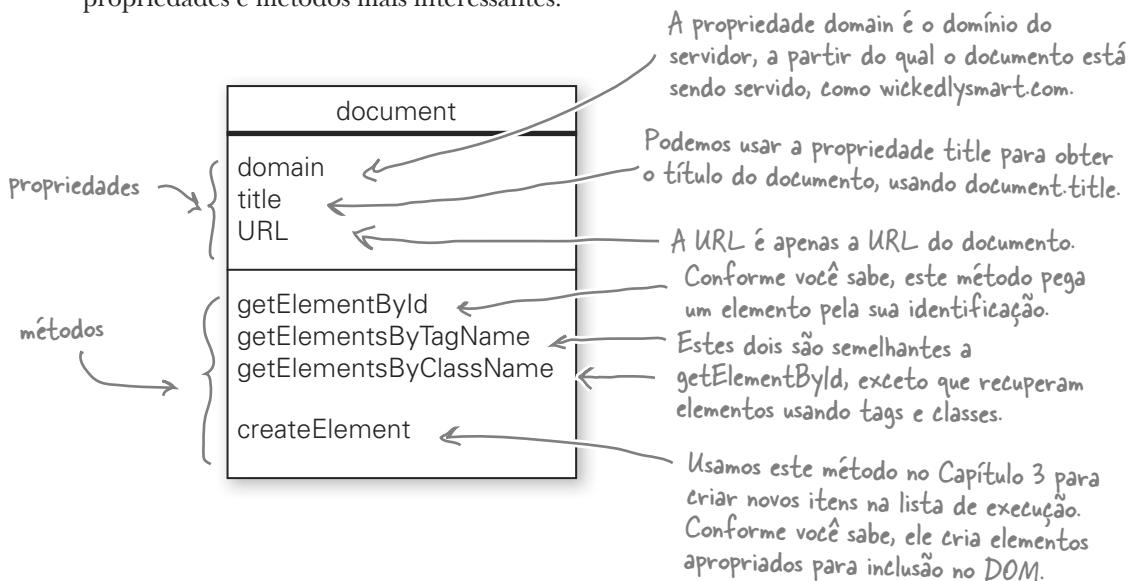
Esta é uma função anônima
atribuída à propriedade `onload`.

```
↳ window.onload = function() {  
    ↳ // code here ↳  
    ↳ } ;
```

E é claro que o corpo da função é executado,
assim que a janela carrega totalmente a
página e chama nossa função anônima!

Outro exame no objeto document

O objeto document é outro rosto familiar; é o objeto que temos usado para acessar o DOM. Como acabamos de ver, é na verdade uma propriedade do objeto window. É claro que não o usamos como `window.document` porque não precisamos. Vamos fazer um exame breve sob a superfície para ver suas propriedades e métodos mais interessantes:



Um exame mais próximo em `document.getElementById`

Prometemos no início deste capítulo que você entenderia `document.getElementById` até o final dele. Bom, você passou por funções, objetos e métodos, e agora está pronto! Veja:

↓

document é o objeto `document`, um objeto interno JavaScript que lhe dá acesso ao DOM.

```
var div = document.getElementById("myDiv");
```

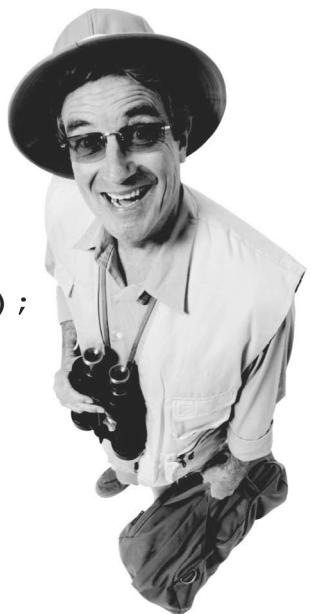
↑

getElementById é um método que...

↑

... recebe um argumento, a id de um elemento <div> e retorna um objeto do elemento.

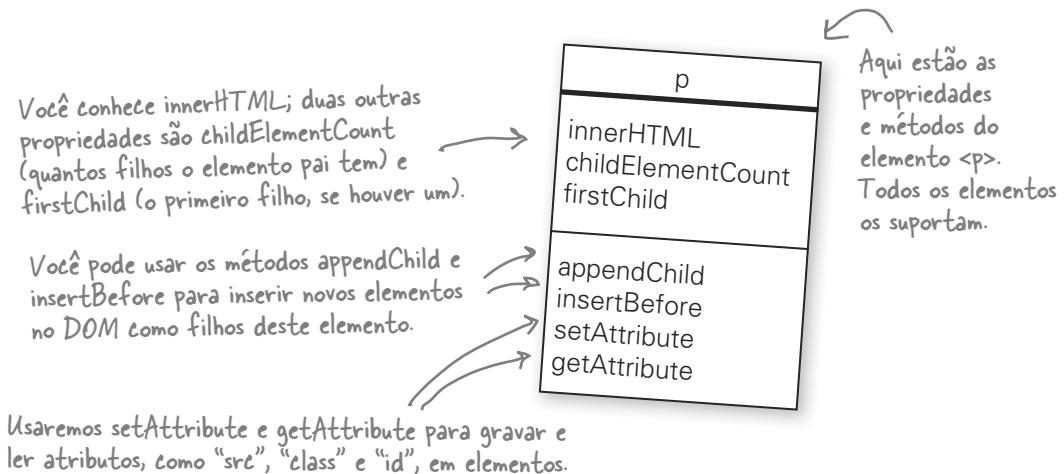
O que era uma string de aparência e sintaxe confusa agora tem muito mais significado, certo? Essa variável `div` também é um objeto: um objeto de elemento. Vamos dar uma olhada mais de perto nele também.



Mais um objeto sobre o qual se pensar: seus elementos objetos

Não devemos esquecer, quando estamos trabalhando com métodos como `getElementById`, que os elementos que eles retornam também são objetos! OK, você pode não ter percebido isto, mas, agora que sabe, talvez esteja começando a pensar que tudo em JavaScript é um objeto e, bom, está quase certo.

Você já viu alguma evidência de propriedades de elementos, como a propriedade `innerHTML`; vamos examinar alguns dos métodos e propriedades mais importantes:



Pnão existemerguntas Idiotas

P: Já que `window` é o objeto global, isso significa que posso usar suas propriedades e todos os seus métodos sem especificar `window` primeiro, certo?

R: Certo. Prefixar as propriedades e métodos do objeto `window` com `window` depende de você. Para coisas como `alert`, todos sabem o que é, e ninguém usa `window` com ele. Por outro lado, se estiver usando propriedades ou métodos menos conhecidos, talvez queira tornar seu código mais fácil de entender, e usar `window`.

P: Então, eu poderia escrever `onload = init` em vez de `window.onload = init`, certo?

R: Sim, mas não recomendamos isso neste caso específico, porque há muitos objetos que têm propriedades `onload`. Seu código ficará muito mais claro se usar `window` na frente de `onload`.

P: O motivo pelo qual não dizemos `window`. `onload = init()` é porque isso chamaria a função, em vez de usar seu valor?

R: Certo. Quando você usa parênteses após o nome da função, como `init()`, está dizendo que quer chamar a função `init`. Se você usar seu nome sem parênteses, então estará atribuindo o valor da função à propriedade `onload`. É uma diferença sutil quando você está digitando, mas as ramificações são grandes, então preste bastante atenção.

P: Qual das duas formas de criar um manipulador window.onload é melhor, usando um nome de função ou uma função anônima?

R: Uma não é melhor que a outra, ambas fazem basicamente a mesma coisa: configuram o valor de window.onload para uma função que será executada quando sua página for carregada. Se por algum motivo você precisar chamar init a partir de outra função posteriormente no seu programa, então precisará definir uma função init. Em caso contrário, não importa qual forma utilizar.

P: Qual a diferença entre objetos internos, como window e document, e os que criamos?

R: Uma diferença é que objetos internos seguem as diretrizes estabelecidas por especificações. Assim, você pode ver as especificações W3C para entender suas propriedades e métodos. Além disso, muitos dos objetos internos, como String, podem ter propriedades que são imutáveis e não podem ser alteradas. Além do mais, objetos são objetos. O bom dos objetos internos é que já estão criados para você.

Sim, String é um objeto! Veja uma boa referência de JavaScript para obter todos os detalhes de suas propriedades e métodos.

Você está deixando este capítulo, sabendo mais sobre objetos e funções do que muitas pessoas por aí. É claro que sempre poderá aprender mais e o incentivamos a explorar (após ter terminado este livro)!

Parabéns! Você completou nossa excursão pelos objetos e passou por vários capítulos de treino em JavaScript. Agora está na hora de usar todo o conhecimento para programar com HTML5 e todas as novas APIs JavaScript, começando no próximo capítulo!

Então, descance um pouco depois deste capítulo, mas, antes que vá, examine brevemente os pontos de bala e faça as palavras cruzadas para fixar tudo.





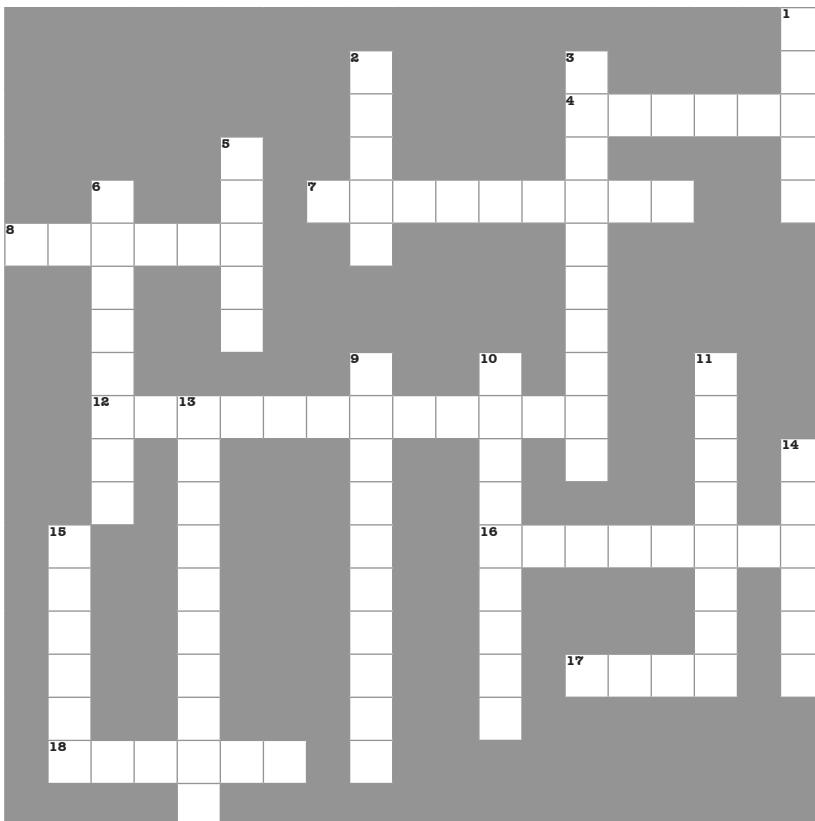
PONTOS DE BALA

- Para criar uma função, use a palavra-chave `function` com parênteses para os parâmetros, se houver algum.
- Funções podem ter nome ou ser anônimas.
- Regras de nomenclatura de funções são as mesmas usadas para variáveis.
- O corpo de uma função fica entre chaves e contém comandos que executam o trabalho da função.
- Uma função pode retornar um valor com o comando `return`.
- Para chamar uma função, use seu nome e passe os argumentos que ela precisar.
- JavaScript usa a passagem de parâmetros por valor.
- Quando você passa um objeto, como um `dog`, como parâmetro para uma função, o parâmetro recebe uma cópia da referência ao objeto.
- Variáveis definidas em funções, incluindo parâmetros, são conhecidas como variáveis locais.
- Variáveis definidas fora das funções são conhecidas como variáveis globais.
- Variáveis locais não são visíveis fora da função na qual são definidas. Isto é conhecido como o escopo de uma variável.
- Se você declarar uma variável local com o mesmo nome de uma global, a local esconde a global.
- Quando você conecta múltiplos arquivos JavaScript a partir da sua página, todas as variáveis globais são definidas no mesmo espaço global.
- Se você atribuir uma nova variável sem usar a palavra-chave `var`, essa variável será global, mesmo se você a estiver atribuindo primeiro a uma função.
- Funções são valores que podem ser atribuídos a variáveis, passados para outras funções, armazenados em matrizes e atribuídos a propriedades de objetos.
- Objetos são coleções de propriedades.
- Você acessa as propriedades de um objeto usando a notação de pontos ou a de `[]`.
- Se usar a notação `[]`, coloque o nome da propriedade entre aspas; por exemplo, `meuObjeto["nome"]`.
- Você pode alterar o valor de uma propriedade, apagar ou adicionar novas propriedades a um objeto.
- Você pode enumerar as propriedades de um objeto, usando um loop `for-in`.
- Uma função atribuída a uma propriedade de objeto é referenciada como um método.
- Um método pode usar uma palavra-chave especial, `this`, para se referir ao objeto no qual foi chamado.
- Um construtor é uma função que cria objetos.
- O trabalho de um construtor é criar um novo objeto e iniciar suas propriedades.
- Para chamar um construtor para criar um objeto, use a palavra-chave `new`. Por exemplo, `new Dog()`.
- Já temos usado diversos objetos neste livro, incluindo `document`, `window` e diversos objetos `element`.
- O objeto `window` é o objeto global.
- O objeto `document` é uma das propriedades de `window`.
- O método `document.getElementById` retorna um objeto `element`.



Palavras Cruzadas HTML5

Este foi um capítulo cheio de funções, objetos, propriedades e métodos — então, há muita coisa para lembrar. Sente-se, relaxe e trabalhe o resto do seu cérebro um pouco. Aqui estão as palavras cruzadas do Capítulo 4.



Horizontal

4. Funções podem ou não incluir este tipo de comando.
7. Por convenção, construtores têm um nome com uma primeira letra _____.
8. Escopo de variável que é visível em qualquer lugar.
12. Juntar propriedades e chamadas de funções com o operador “ponto”.
16. Use esta palavra-chave para iniciar uma definição de função.
17. Refere-se ao objeto corrente em um método de objeto.
18. Uma propriedade em window a qual atribuímos uma função de manipulação.

Vertical

1. O operador _____ permite a você acessar as propriedades e métodos de um objeto.
2. Estas variáveis só estão disponíveis em funções.
3. O que você fornece na sua chamada de função.
5. Argumentos são passados por _____.
6. O objeto _____ representa o DOM.
9. O que você fornece na sua definição de função.
10. Funções sem comandos de retorno retornam isto.
11. Funções sem um nome.
13. Este tipo de função cria objetos.
14. O verdadeiro objeto global.
15. Uma função em um objeto.



Aponte o seu lápis Solução

Use seu conhecimento sobre funções e passagem de argumentos para parâmetros para avaliar o código abaixo. Após ter passado pelo código, escreva o valor de cada variável abaixo. Aqui está a nossa solução.

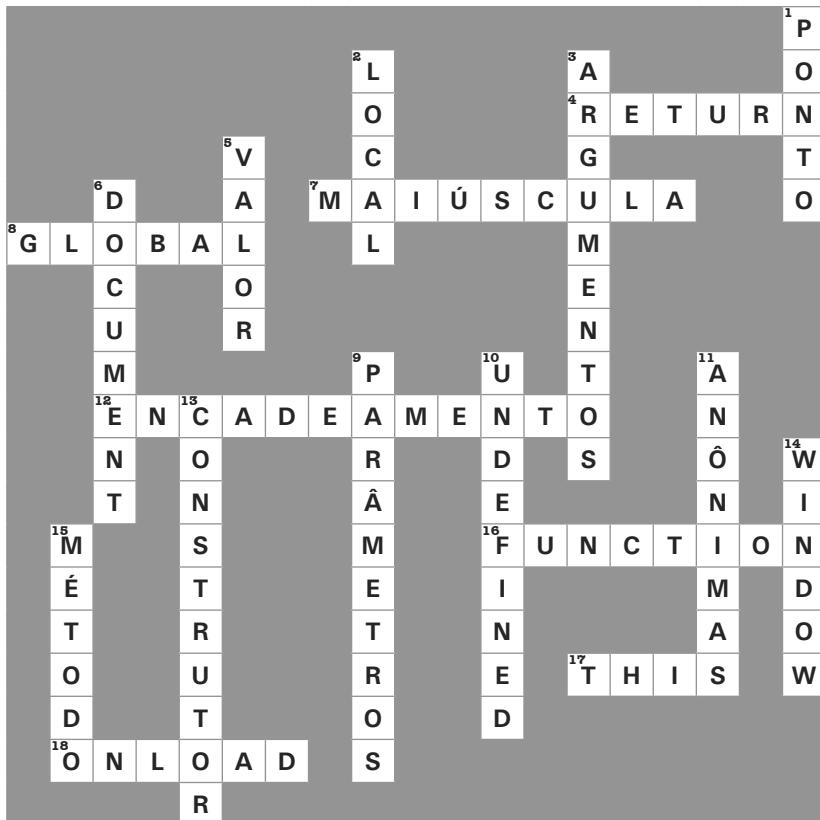
```
function dogsAge(age) {  
    return age * 7;  
}  
  
var myDogsAge = dogsAge(4);  
  
function rectangleArea(width, height) {  
    var area = width * height;  
    return area;  
}  
var rectArea = rectangleArea(3, 4);  
  
function addUp(numArray) {  
    var total = 0;  
    for (var i = 0; i < numArray.length; i++)  
    {  
        total += numArray[i];  
    }  
    return total;  
}  
  
var theTotal = addUp([1, 5, 3, 9]);  
  
function getAvatar(points) {  
    var avatar;  
    if (points < 100) {  
        avatar = "Mouse";  
    } else if (points > 100 && points < 1000)  
    {  
        avatar = "Cat";  
    } else {  
        avatar = "Ape";  
    }  
    return avatar;  
}  
var myAvatar = getAvatar(335);
```

Escreva o valor de cada variável aqui...

myDogsAge = 28
rectArea = 12
theTotal = 18
myAvatar = Cat.....



Solução das Palavras Cruzadas HTML5



5 tornando ativa sua localização html



Geolocalização



Aonde quer que vá, lá está você. Às vezes, saber onde se está faz toda a diferença (principalmente para um aplicativo web). Neste capítulo, vamos mostrar como criar páginas da web que são **localizáveis** — em alguns casos, será possível apontar o local exato dos usuários e, em outros, será apenas possível determinar a área da cidade em que se encontram (mas ainda assim saberemos qual a cidade!). Infelizmente, às vezes não será possível determinar nada em relação a sua localização, o que pode se dar por motivos técnicos ou apenas por não querer ninguém se metendo em suas vidas. Vá descobrir sozinho. De qualquer modo, neste capítulo, vamos explorar uma API JavaScript: Geolocalização. Arranje o melhor dispositivo de localização que puder (mesmo que seja seu computador pessoal) e vamos começar.

Seus usuários agora estão em movimento com seus dispositivos móveis localizáveis. Os melhores aplicativos serão aqueles que podem aprimorar as experiências desses usuários utilizando localização.

Localização, Localização, Localização

Saber a localização de seus usuários pode acrescentar, e muito, à experiência da web: você poderá oferecer direções, fazer sugestões de lugares que eles poderão visitar; é possível saber se está chovendo e indicar atividades em lugares fechados; pode-se permitir aos usuários saber quem mais nos arredores está querendo fazer alguma atividade. Realmente, não há fim para as maneiras com que se pode utilizar a informação de localização.

Com HTML5 (e a API JavaScript de Geolocalização), pode-se, facilmente, acessar informações de localização em suas páginas. Dito isto, há algumas coisinhas a saber a respeito da localização antes de começarmos. Vamos conferir?



Pnão existemerguntas Idiotas

P: Ouvi falar que a Geolocalização não é uma API de verdade...

R: A Geolocalização não é considerada um membro da primeira classe dentro do padrão existente de HTML5, mas, mesmo assim, é um padrão para o W3C, amplamente suportado e quase todo mundo inclui a Geolocalização em suas listas de APIs HTML5 importantes. É mais provável que ela seja, sim, uma **verdadeira** API JavaScript!

P: A API de Geolocalização é a mesma API do Google Maps?

R: Não. São APIs completamente diferentes. A API de Geolocalização é focada unicamente na obtenção de informação a respeito do posicionamento na Terra. A API do Google Maps é uma biblioteca JavaScript oferecida pela Google que proporciona acesso a todas as funcionalidades do Google Maps. Portanto, se precisar mostrar a localização

de seus usuários num mapa, a API do Google oferece uma maneira conveniente de implementar tal funcionalidade.

P: Não existe uma preocupação com a questão da privacidade, já que meu dispositivo revela minha localização?

R: A especificação da Geolocalização determina que qualquer browser deve ter a permissão expressa do usuário para fazer uso de sua localização. Por isso, se seu código utiliza a API de Geolocalização, a primeira coisa que o browser fará será se certificar de que o usuário concorda com o compartilhamento de sua localização.

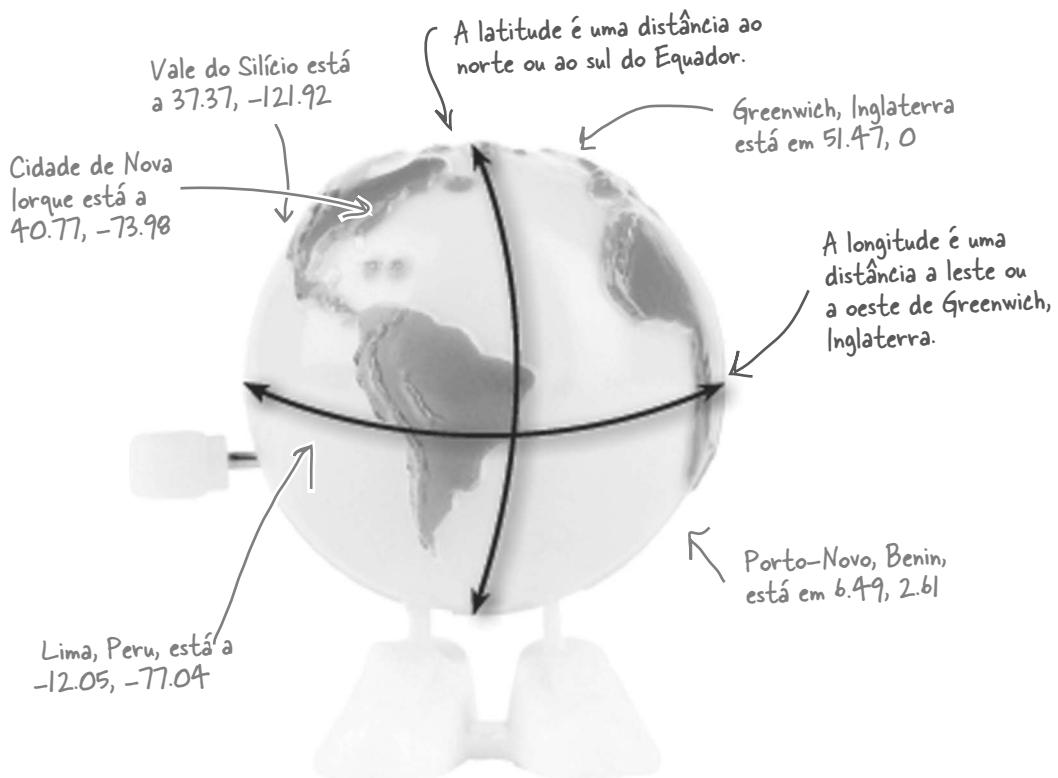
P: A Geolocalização tem bom suporte?

R: Certamente; de fato, está disponível em praticamente todos os browsers modernos, tanto os de desktop quanto os mobile. É preciso saber ao certo se está usando a última versão de seu navegador; se estiver, então provavelmente você terá êxito.

A Latitude e a Longitude da Questão...

Para você saber onde está, é necessário um sistema de coordenadas, e você vai precisar de um para a superfície da Terra. Por sorte, nós temos algo do gênero, que usa latitude e longitude juntos como um sistema de coordenadas. A latitude especifica um ponto no sentido norte/sul do planeta e a longitude, um ponto leste/oeste. A latitude é medida a partir do Equador e a longitude a partir de Greenwich, na Inglaterra. O trabalho da API de geolocalização é nos dar coordenadas de onde estamos a qualquer momento, usando-as assim:

O Real
Observatório
de Greenwich,
para ser mais
preciso.



Latitude/Longitude de Perto

Você provavelmente já viu as especificações da latitude e da longitude tanto em graus/minutos/segundos ($47^{\circ}38'34''$, $122^{\circ}32'32''$), quanto em valores decimais (47.64, -122.54). Com a API de Geolocalização sempre usamos valores decimais. Se precisar converter graus/minutos/segundos em decimais, utilize esta função:

```
function degreesToDecimal(degrees, minutes, seconds) {  
    return degrees + (minutes / 60.0) + (seconds / 3600.0);  
}
```

Perceba também que a longitude Oeste e a latitude Sul são representadas por valores negativos.

Como a API de Geolocalização determina sua localização

Você não precisa ter o mais recente smartphone para deter a capacidade da localização. Mesmo browsers de desktop estão se juntando ao grupo. Talvez você pergunte: como um browser de desktop determina sua localização se não possui GPS ou qualquer outra tecnologia avançada de localização? Bem, todos os navegadores (em dispositivos e em seu computador) estão se utilizando de formas diferentes para determinar sua exata localização, mesmo que algumas sejam mais precisas do que outras. Vamos dar uma olhada:



GPS

Global Positioning System (Sistema de Posicionamento Global), suportado por diversos dispositivos atuais, fornece informação de localização extremamente apurada, baseada em satélites. Dados de localização podem incluir informações de altitude, velocidade e direção. Para usá-lo, no entanto, seu dispositivo precisa ter livre acesso ao céu, e pode levar um bom tempo para obter a localização. O GPS também pode gastar muita bateria.

Endereço IP

A informação de localização baseada em seu endereço IP usa uma database externa para rastrear o IP até uma localização física. A vantagem desta abordagem é que ela funciona em qualquer lugar; no entanto, volta e meia os endereços IP indicam localizações, tais como o provedor de internet de seu escritório. Pense neste método como sendo confiável apenas para estabelecer limites de cidades ou vizinhanças.





Meu telefone é dos antigos.
Nenhum GPS nele, mas, através da
triangulação de torres de celular,
meu telefone consegue ter uma
boa ideia de onde me encontro, e o
navegador faz uso disso.

Telefone Celular

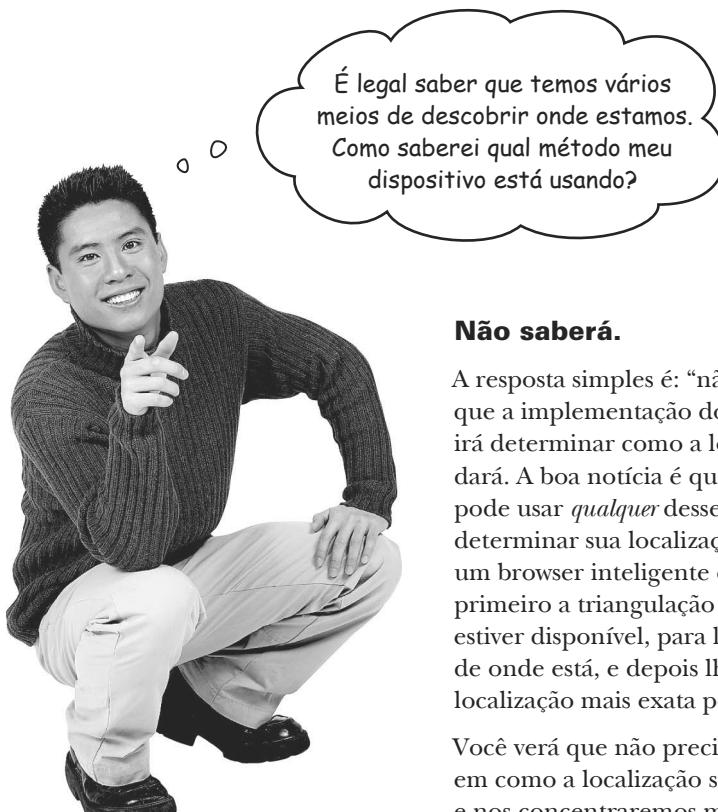
A triangulação de telefones celulares descobre sua localização baseada na distância em que você se encontra de uma ou mais torres de celular (obviamente que quanto mais torres, mais precisa será sua localização). Este método pode ser bem funcional e opera mesmo dentro de ambientes fechados (diferentemente do GPS); também é bem mais rápido que o GPS. Então, novamente, se estiver no meio do nada com apenas uma torre de celular, sua precisão irá sofrer.



Vou de cafeteria em cafeteria com meu laptop e minhas assinaturas WiFi. Você saberá onde estou ao triangular todos os hotspots. Parece funcionar muito bem.

Wi-Fi

O posicionamento por Wi-Fi usa um ou mais pontos de acesso Wi-Fi para triangular sua localização. Este método pode ser bem preciso, funciona dentro de ambientes fechados e é rápido. Claro que requer que esteja *meio que* parado (talvez bebendo um chá gelado numa cafeteria).



É legal saber que temos vários meios de descobrir onde estamos. Como saberei qual método meu dispositivo está usando?

Não saberá.

A resposta simples é: “não saberá”, já que a implementação do navegador irá determinar como a localização se dará. A boa notícia é que o browser pode usar *qualquer* desses meios para determinar sua localização. De fato, um browser inteligente deverá utilizar primeiro a triangulação de torres, se estiver disponível, para lhe dar uma ideia de onde está, e depois lhe dará uma localização mais exata por WiFi ou GPS.

Você verá que não precisa se preocupar em como a localização será determinada, e nos concentraremos mais na precisão do que na localização. Baseado na precisão, pode-se determinar quanto útil será a localização para você. Fique ligado — voltaremos à precisão logo mais.



Aponte o seu lápis

Pense a respeito de suas páginas e aplicativos HTML existentes (ou alguns que quer criar); como será possível incorporar informações de localização neles?

- Permitir que meus usuários façam compartilhamentos com outros nos arredores.
- Permitir que meus usuários encontrem, mais facilmente, recursos e serviços.
- Rastrear onde meu usuário faz algo.
- Dar direções aos meus usuários a partir de onde estão.
- Usar localização para determinar outros dados demográficos de meus usuários.
-
-
-
-



Suas ideias aqui!

Mas diz aí: onde você está?

Bem, claro que *você* sabe onde está, mas vejamos onde seu *browser* acha que você está. Para sabê-lo, vamos criar uma pequena HTML:

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Where am I?</title>
  <script src="myLoc.js"></script>
  <link rel="stylesheet" href="myLoc.css">
</head>
<body>
  <div id="location">
    Your location will go here.
  </div>
</body>
</html>
```

Todas as coisas de sempre no início, incluindo um link para o arquivo em que poremos nosso JavaScript, myLoc.js, e uma folha de estilo, myLoc.css para deixar tudo bem bonito.

Vamos escrever nosso código de geolocalização em myLoc.js.

E você usará esta <div> para gerar sua localização.

Ponha toda esta HTML num arquivo chamado myLoc.html.



Agora vamos criar myLoc.js e escrever um pequeno código; faremos isso rapidamente para depois voltar e analisar. Adicione ao seu arquivo myLoc.js o que segue:

```
window.onload = getMyLocation;
function getMyLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(displayLocation);
  } else {
    alert("Oops, no geolocation support");
  }
}
```

Chamamos a função getMyLocation assim que o navegador carregar a página.

É assim que verificamos para nos certificarmos de que o browser suporta a API de Geolocalização; se o objeto navigator.geolocation existir, então conseguimos!

Se existir, então acionamos o método getCurrentPosition e o passamos por uma função handler, displayLocation. Vamos implementar isso em um segundo.

A função displayLocation é o manipulador que conseguirá ter controle sobre a localização.

Se o browser NÃO suportar geolocalização, então vamos apenas criar um alerta pop up ao usuário.

Eis nosso manipulador, que irá disparar quando o browser tiver uma localização.

```
function displayLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;

    var div = document.getElementById("location");
    div.innerHTML = "You are at Latitude: " + latitude + ", Longitude:
    " + longitude;
}
```

Então pegamos nossa <div> da HTML...

O manipulador de `getCurrentPosition` recebe uma posição que contém a latitude e a longitude de sua localização (junto com algumas informações de precisão que veremos adiante).

Pegamos a latitude e a longitude de sua localização do objeto `position-coords`.

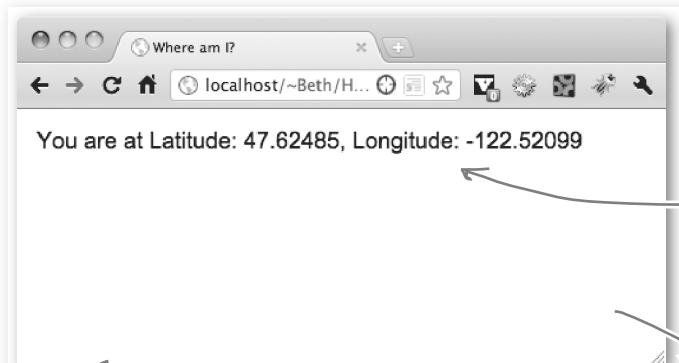
...e, por ora, vamos apenas ajustar o conteúdo da <div> de localização para sua localização usando `innerHTML`.

Test drive de sua localização



Digite seu código e leve sua nova página de localização para um test drive.

Quando você rodar um aplicativo web de Geolocalização pela primeira vez, notará uma solicitação no browser, pedindo sua permissão para usar sua localização. Esta é uma checagem de segurança do navegador, à qual você poderá dizer não. Supondo, porém, que talvez queira testar este aplicativo web, você terá que clicar em Permitir ou Sim. Quando o fizer, o aplicativo deverá mostrar sua localização, como exemplificado abaixo:



Lembre-se sempre de que conseguir sua localização não é muito instantâneo; pode levar um tempinho...



A solicitação para permissão pode parecer um pouco diferente, dependendo do browser que estiver usando, mas vai ficar mais ou menos assim.

Eis sua localização! Claro que será diferente da nossa (caso contrário, ficaremos muito preocupados contigo).

Se sua localização não está aparecendo, e supondo que você verificou duas vezes por erros de digitação e coisas do tipo, aguente as pontas por algumas páginas e lhe daremos alguns códigos para tirar esses tipos de bugs...

Se seu browser suporta a API de Geolocalização, você encontrará uma propriedade de geolocalização no objeto navigator.



O que acabamos de fazer...

Agora que temos alguns códigos de geolocalização rodando perfeitamente (e, novamente, se ainda não estiver visualizando nenhuma localização, espere um pouco, pois já estamos chegando a algumas técnicas de eliminação de bugs), vamos dar um passeio pelo código, porém com mais alguns detalhes:

- 1 A primeira coisa que precisa saber, se for escrever código de geolocalização, é: "este browser dá suporte?" Para fazer isso, utilizamos o fato de que os navegadores possuem uma propriedade de geolocalização em seu objeto de navegação apenas quando a geolocalização é suportada.

Portanto, podemos testar para ver se a propriedade de geolocalização existe e, se existir, fazer uso dela. Do contrário, deixaremos o usuário saber:

```
if (navigator.geolocation) {  
    ...  
} else {  
    alert("Oops, no geolocation support");  
}
```

Podemos usar um teste simples para ver se a geolocalização está lá (se não estiver, então o navigator.geolocation avalia em null e a condição irá falhar).

Se estiver lá, podemos fazer uso dele, e, se não, faremos com que o usuário saiba.

- 2 Agora, se houver uma propriedade navigator.geolocation, faremos mais uso dela. De fato, a propriedade navigator.geolocation é um objeto que contém a API de geolocalização inteira. O método principal que a API suporta é o `getCurrentPosition`, que faz o trabalho de conseguir a localização do browser. Vamos dar uma olhada mais de perto neste método, que possui três parâmetros, dois dos quais são opcionais:

O `successHandler` é uma função que é chamada se o browser for capaz de determinar, com sucesso, sua localização.

O `errorHandler` é outra função que é chamada se algo der errado e o browser não puder determinar sua localização.

```
getCurrentPosition(successHandler, errorHandler, options)
```

Esses dois parâmetros são opcionais. Por isso não precisamos deles antes.

O parâmetro `options` permite-lhe personalizar a maneira com que a geolocalização trabalha.

- 3 Agora, vamos dar uma olhada em nossa chamada para o método `getCurrentPosition`. Por ora, forneceremos apenas o argumento `successHandler` para manipular uma tentativa bem-sucedida de obter a localização do browser. Veremos o caso em que o browser falha em encontrar uma localização em um instante.

```
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(displayLocation);
}
```

E estamos chamando o método `getCurrentPosition` do objeto `geolocation` com um argumento: o `success callback*`.

Você se lembra do encadeamento* do Capítulo 4? Estamos usando o objeto `navigator*` para ter acesso ao objeto `geolocation`, que é simplesmente uma propriedade de `navigator`.

Se e quando a geolocalização determinar sua localização, ela vai chamar `displayLocation`.

Você percebeu que estamos passando de uma função para outra aqui? Lembre-se de que vimos no Capítulo 4 que funções são valores, então podemos fazer isso sem problemas.

- 4 Agora, vamos ver o `success handler*`, `displayLocation`. Quando `displayLocation` é chamado, a API de geolocalização passa-lhe um objeto `position` que contém informação sobre a localização do browser, incluindo um objeto `coordinates` que possui a latitude e a longitude (assim como alguns outros valores que falaremos mais tarde).

`position` é um objeto que é passado dentro de seu `success handler` pela API de geolocalização.

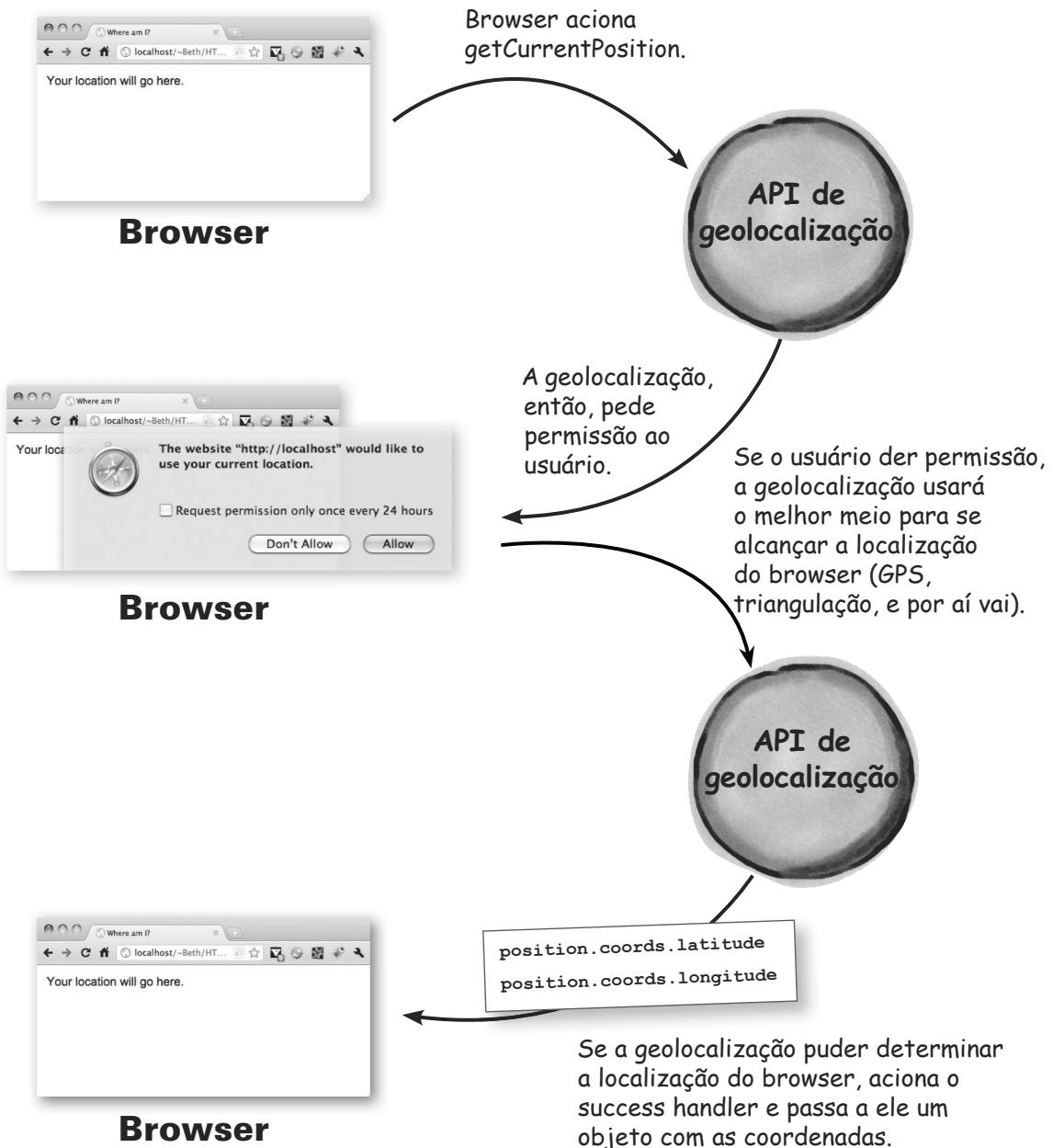
```
function displayLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var div = document.getElementById("location");
    div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: "
+ longitude;
}
```

O objeto `position` possui uma propriedade `coords` que tem uma referência em relação ao objeto `coordinates`...
... e o objeto `coordinates` tem sua latitude e sua longitude.

E esta parte temos certeza de que você pode fazer até dormindo: estamos apenas pegando a informação de coordenadas e mostrando-a em uma `<div>` dentro da página.

Como tudo se encaixa

Agora que já vimos o código, vejamos como tudo funciona:



Test Drive – Diagnósticos



Quando lidamos com Geolocalização, nem todos os testes serão bem sucedidos. Mesmo que o primeiro teste tenha dado certo, algo vai dar errado em algum momento. Para ajudar, criamos um pequeno teste de diagnóstico para que você possa acrescentar em seu código. Então, se estiver tendo problemas, eis sua resposta; e mesmo que não esteja, algum de seus usuários terá algum percalço e você vai querer saber como lidar com ele no seu código. Portanto, adicione o código abaixo e, se estiver com algum problema, preencha cuidadosamente o formulário de diagnóstico do fim, uma vez que o tenha diagnosticado:

Para criar o teste de diagnóstico, vamos adicionar um error handler ao acionamento do método `getCurrentPosition`. Este handler será acionado a qualquer momento que a API de Geolocalização encontrar um problema em determinar sua localização. Veja como fazemos para adicioná-lo:

Acrecente um segundo argumento em seu `getCurrentPosition`, chamado `displayError`. Esta é uma função que será acionada quando a geolocalização falhar na detecção de uma localização.

)

```
navigator.geolocation.getCurrentPosition(displayLocation, displayError);
```

Agora precisamos escrever o error handler. Para fazê-lo, é preciso saber que a geolocalização passa um objeto error para seu handler, que contém um código numérico descrevendo a razão pela qual não foi possível determinar a localização de seu browser. Dependendo do código, poderá também fornecer uma mensagem dando informações posteriores sobre o erro. Segue abaixo como podemos usar o objeto error no handler*:

Aqui está o nosso novo handler, no qual é passado um erro pela API de Geolocalização.

```
function displayError(error) { ← Aqui está o nosso novo handler, no qual é
    var errorTypes = { ← passado um erro pela API de Geolocalização.
        0: "Unknown error", ← O objeto error contém uma propriedade do código que possui
        1: "Permission denied by user", ← um número de 0 a 3. Esta é uma maneira legal de associar
        2: "Position is not available", ← uma mensagem de erro com cada código em JavaScript.
        3: "Request timed out" ← Criamos um objeto com três propriedades
    }; ← nomeadas de 0 a 3. Essas propriedades
    var errorMessage = errorTypes[error.code]; ← são strings com uma mensagem de erro
    if (error.code == 0 || error.code == 2) { ← que queremos associar com cada código.
        errorMessage = errorMessage + " " + error.message; ← E, usando a propriedade
    } ← error-code, assinalamos
    var div = document.getElementById("location"); ← uma daquelas strings a uma
    div.innerHTML = errorMessage; ← nova variável, errorMessage.
} ← No caso de erros zero e dois,
    ← há, ocasionalmente, informações
    ← adicionais na propriedade error-
    ← message, então adicionamo-la a
    ← nossa string errorMessage.
    Então, adicionamos a mensagem à página
    para que o usuário saiba.
```



Antes de rodarmos o teste, vamos dar uma olhada mais de perto nos tipos de erros que podemos encontrar.

```
var errorTypes = {  
    0: "Unknown error",  
    1: "Permission denied by user",  
    2: "Position is not available",  
    3: "Request timed out"  
};
```

Finalmente, a geolocalização possui uma configuração interna de tempo limite, a qual, se excedida antes de uma localização ser determinada, causa este erro.

Este é um erro genérico, que é usado quando nenhum dos outros faz sentido. Dê uma olhada na propriedade `error.message` para mais informações.

Isto significa que o usuário negou a solicitação de usar sua informação de localização.

Isto significa que o browser tentou, mas falhou ao obter sua localização. Novamente, consulte `error.message` para mais informações.

Veremos como mudar o tempo limite padrão da geolocalização mais a frente, neste capítulo.



Quando tiver digitado o teste de diagnóstico, vá em frente e experimente. Obviamente que se você recebeu uma localização, então tudo funcionará e não verá nenhum desses erros. Você poderá forçar um erro, negando a solicitação do browser para usar sua localização. Ou você pode ser criativo e, digamos, andar até um lugar fechado com seu telefone com GPS, enquanto desativa sua rede. No pior dos casos, se esperar por um longo período sem obter uma localização ou uma mensagem de erro, é mais provável que você espere por um longo valor de tempo limite para, bem, atingi-lo. Veremos como diminuir a duração do tempo limite um pouco mais para frente.

Aqui estão os resultados do diagnóstico

- Não dei permissão para que minha localização seja usada.
- Minha posição não estava disponível.
- Após alguns segundos, recebi uma mensagem indicando que havia uma solicitação de tempo limite.
- Nada aconteceu; nenhuma localização e nenhum alerta de erro.
- Outros _____



Para testar seu código de geolocalização num dispositivo móvel, você vai querer um servidor.

A menos que tenha um meio de carregar seu HTML, JavaScript e arquivos CSS diretamente em seu dispositivo móvel, a maneira mais fácil para testá-lo é colocá-lo num servidor (dê uma olhada no próximo capítulo para ver como configurar seu próprio servidor, se quiser) e acessá-lo lá. Se você tiver um servidor e quiser fazer isso, nós o encorajamos. Por outro lado, se isso não funcionar para você, informamos que o código está disponível nos servidores Wickedly Smart, podendo assim testar em seus dispositivos móveis. Dito isto, damos força para que siga com o código em seu desktop, e uma vez que ele esteja funcionando lá, teste em seu dispositivo móvel usando o servidor (o seu ou o Wickedly Smart).

Para o primeiro test drive (incluindo o diagnóstico de erro), use seu dispositivo para ir até o endereço <http://wickedlysmart.com/hfhtml5/chapter5/latlong/myLoc.html>.

Revelando nossa localização secreta...

Agora que tiramos o básico da frente, vamos fazer algo mais interessante com localização. Que tal vermos quanto longe você está de nossa localização secreta em Wickedly Smart QG? Para fazer isso, precisamos das coordenadas QG e precisamos saber como calcular a distância entre duas coordenadas. Primeiro, vamos acrescentar outra <div> para usar no HTML:

```

<body>
  <div id="location">
    Your location will go here.
  </div>
  <div id="distance">
    Distance from WickedlySmart HQ will go here.
  </div>
</body>
</html>
```

↑ Acrescente esta nova <div> em seu HTML.

Não existem Perguntas Idiotas

P: A latitude e a longitude informadas pelo aplicativo para minha localização não batem. Por quê?

R: Há uma porção de maneiras para que seu dispositivo e o serviço provedor de localização calcule sua posição, e alguns são mais precisos que outros. O GPS é normalmente o mais preciso. Vamos ver um jeito de determinar a estimativa de precisão que o serviço de localização retorna como parte do objeto position, então você poderá saber quanto esperar de precisão em relação aos dados de localização.



Quartel-general da Wickedly Smart está em 47.62405, -122.52099.



Alguns Códigos PRONTOS para ASSAR: computando distância

Sempre quis saber como computar a distância entre dois pontos numa esfera? Você achará os detalhes fascinantes, mas eles estão um pouco fora de contexto neste capítulo. Então, vamos lhe dar alguns *códigos prontos para assar* que fazem exatamente isto. Para computar a distância entre duas coordenadas, a maioria das pessoas usa a equação Haversine; você a encontrará implementada logo abaixo. Sinta-se à vontade para usá-la onde quer que precise para saber a distância entre duas coordenadas:

Esta função considera duas coordenadas, uma de partida e outra de destino, e oferece a distância em quilômetros entre elas.

```
function computeDistance(startCoords, destCoords) {  
    var startLatRads = degreesToRadians(startCoords.latitude);  
    var startLongRads = degreesToRadians(startCoords.longitude);  
    var destLatRads = degreesToRadians(destCoords.latitude);  
    var destLongRads = degreesToRadians(destCoords.longitude);  
  
    var Radius = 6371; // radius of the Earth in km  
    var distance = Math.acos(Math.sin(startLatRads) * Math.sin(destLatRads) +  
        Math.cos(startLatRads) * Math.cos(destLatRads) *  
        Math.cos(startLongRads - destLongRads)) * Radius;  
  
    return distance;  
}  
  
function degreesToRadians(degrees) {  
    var radians = (degrees * Math.PI)/180;  
    return radians;  
}
```

Veremos mais dessa função no capítulo Canvas.

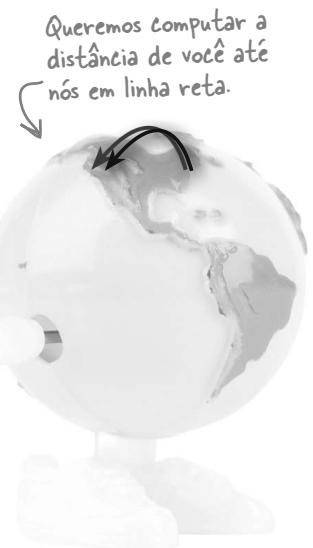
Arcense este em seu arquivo myLoc.js

Escrevendo o código para encontrar a distância

Agora que temos uma função para computar a distância entre duas coordenadas, vamos definir nossa localização (isto é, a do autor) aqui no QG da Wickedly Smart (vá em frente e digite isto também):

```
var ourCoords = {
    latitude: 47.624851,
    longitude: -122.52099
};
```

Aqui vamos definir um objeto literal para as coordenadas de nossa localização no QG da Wickedly Smart. Adicione isto como uma variável global no início de seu arquivo myLoc.js.



Agora vamos escrever o código: tudo que precisamos fazer é passar as coordenadas de sua localização e de nossa localização à função computeDistance:

```
function displayLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;

    var div = document.getElementById("location");
    div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: " + longitude;

    var km = computeDistance(position.coords, ourCoords);
    var distance = document.getElementById("distance");
    distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";
}
```

Aqui estamos passando as coordenadas de sua posição e também nossas coordenadas para computeDistance.

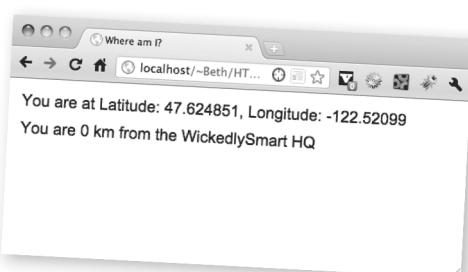
Então pegamos os resultados e atualizamos os conteúdos da distância <div>.

Test drive da ativação da localização



Agora, vamos levar este novo código para um test drive. Vamos em frente. Termine de adicionar o código em myLoc.js e então recarregue myLoc.html em seu browser. Confira sua localização e também sua distância de nós.

Sua localização e distância →
serão, logicamente,
diferentes, dependendo de
onde estiver no mundo.



Tente online <http://wickedlysmart.com/hfhtml5/chapter5/distance/myLoc.html>



Mapeando sua posição

Como lhe dissemos anteriormente, a API de Geolocalização é bem simples — fornece-lhe uma maneira de saber onde você está (e, como verá mais a frente, rastrear também), mas não lhe fornece quaisquer ferramentas para visualizar sua localização. Para fazer isso, precisamos confiar numa ferramenta terceirizada, e como você deve imaginar, o Google Maps é de longe a ferramenta mais popular para tanto. Obviamente, o Google Maps não faz parte das especificações HTML5, mas os dois interagem bem, portanto, não nos importamos com um pouco de diversidade aqui e ali para lhe mostrar como integrá-lo com a API de Geolocalização. Se quiser diversificar, pode começar adicionando isto ao cabeçalho de seu documento HTML e depois vamos trabalhar para adicionar um mapa à sua página:

```
<script src="http://maps.google.com/maps/api/js?sensor=true"></script>
```

↑
Esta é a localização da API
JavaScript do Google Maps.

↑ Certifique-se de digitar isto exatamente
como está, incluindo o parâmetro sensor
query* (a API não vai funcionar sem isso).
Estamos usando sensor=true, pois nosso
código está usando sua localização. Se
estivéssemos usando apenas o mapa sem a
localização, digitariamnos sensor=false.

Como adicionar um mapa à sua página

Agora que você criou um link com a API do Google Maps, toda a funcionalidade do Google Maps está disponível por meio do JavaScript. Precisamos, porém, de um lugar para pôr nosso Google Maps, e, para fazer isso, precisamos definir um elemento que irá fixá-lo.

```

:
:
<body>
  <div id="location">
    Your location will go here.
  </div>
  <div id="distance">
    Distance from WickedlySmart HQ will go here.
  </div>
  <div id="map"> ← Aqui está a <div>. Perceba que definimos um
    </div> estilo em myLoc.css que ajusta o mapa <div> em
  </body> 400px por 400px com uma margem preta.
</html>
```

Preparando-se para criar um mapa...

Precisamos de duas coisas para criar o mapa: latitude e longitude (já sabemos como consegui-las) e de um conjunto de opções que descrevam como queremos criá-lo. Comecemos com a latitude e a longitude. Já sabemos como consegui-las com a API de Geolocalização, mas a API do Google gosta delas amarradas em seu próprio objeto. Para criar um destes objetos, podemos usar um construtor fornecido pelo Google:

Não se esqueça!
Construtores
começam com uma
letra maiúscula.

```
var googleLatAndLong = new google.maps.LatLng(latitude, longitude);
```

google.maps precede todos os métodos da API do Google Maps. ↑ Eis o construtor, que pega nossa latitude e longitude e entrega um novo objeto que fixa os dois.

O Google nos dá algumas opções de ajustes para controlar como o mapa será criado. Por exemplo, podemos controlar a intensidade do zoom com que o mapa é visto, onde o mapa é centralizado e o tipo de mapa, como um mapa de estradas, uma vista de satélite, ou ambos. Segue como criamos as opções:

A opção de zoom pode ser especificada de 0 a 21. Experimente o zoom: números maiores correspondem a mais zoom (então poderá ver mais detalhes). 10 é meio que do tamanho de "cidades".

```
var mapOptions = {
  zoom: 10, ← Aqui está nosso novo objeto que acabamos de criar.
  center: googleLatAndLong, ← Queremos que o mapa esteja centrado nesta localização.
  mapTypeId: google.maps.MapTypeId.ROADMAP ← Você pode experimentar
}; ; também SATELLITE e HYBRID como opções aqui.
```



Mostrando o Mapa

Vamos pôr tudo junto numa nova função, showMap, que pega um conjunto de coordenadas e mostra um mapa em sua página:

```
var map; ← Estamos declarando um mapa de variável global, que irá fixar o mapa do Google depois de o criarmos. Você verá como isso será útil num instante.

function showMap(coords) {
  var googleLatAndLong =
    new google.maps.LatLng(coords.latitude,
                           coords.longitude); ← Usamos nossa latitude e
                                         nossa longitude a partir
                                         do objeto coords...
  var mapOptions = { ← ...e usamos elas para
                     zoom: 10, ← criar um objeto
                     center: googleLatAndLong, ← googlemapsLatLng.
                     mapTypeId: google.maps.MapTypeId.ROADMAP ← Criamos o objeto mapOptions
  }; ← com as opções que queremos
       ← ajustar em nosso mapa.

  var mapDiv = document.getElementById("map"); ← Finalmente, pegamos o mapa
  map = new google.maps.Map(mapDiv, mapOptions); ← <div> do DOM e o passamos
                                                 ← com o mapOptions para o
                                                 ← construtor do Mapa para criar
                                                 ← o objeto google.maps.Map. Isso
                                                 ← mostra o mapa em nossa página.

} ← Estamos assinalando o novo
     ← objeto Map para nosso
     ← mapa de variável global. ← Aqui está outro construtor da API do
                               ← Google, que pega um elemento e nossas
                               ← opções e cria e devolve um objeto map.
```

Vá em frente e acrescente este código ao seu arquivo no fim. Agora, precisamos apenas amarrá-lo a seu código existente. Façamos isso ao editar a função displayLocation:

```
function displayLocation(position) {
  var latitude = position.coords.latitude;
  var longitude = position.coords.longitude;

  var div = document.getElementById("location");
  div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: " + longitude;

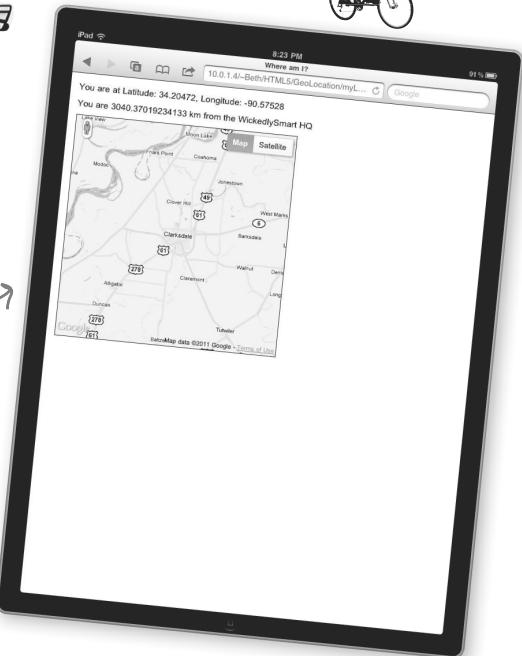
  var km = computeDistance(position.coords, ourCoords);
  var div = document.getElementById("distance");
  distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";

  showMap(position.coords); ← Acionaremos showMap a partir de displayLocation
} ← depois de ter atualizado as outras <div>s na página.
```

Test drive de seu novo heads-up display

Certifique-se de ter adicionado todo o novo código da página anterior e também adicionado a nova <div> do mapa em seu HTML; então, recarregue sua página e, se o browser puder determinar sua localização, você verá um mapa.

Eis nosso novo Google Map!
 Estamos mostrando a localização do ciclista a 34.20472, -90.57528; claro que você provavelmente está em outro lugar.



Legal! Existe alguma forma de vermos nossa exata localização no mapa? Com alguns desses ajustes malucos de vocês?

Você realmente quer um desses perto de sua bicicleta?

Visite: <http://wickedlysmart.com/hfhtml5/chapter5/map/myLoc.html>



Colocando um alfinete...

Seria mais útil ainda, se pudesse ver exatamente onde está localizado no mapa. Se você já usou o Google Maps, então provavelmente já está familiarizado com os alfinetes que apontam a localização de itens que está procurando. Por exemplo, se procurar pelo Obelisco Espacial (Space Needle), em Seattle, no Estado de Washington, você terá um mapa com um alfinete próximo à área do Obelisco na cidade e, se clicar nele, você verá uma janela de informação com mais detalhes sobre o item. Bem, esses alfinetes são chamados de marcadores e eles são uma das inúmeras coisas oferecidas pela API do Google Maps.

Adicionar um marcador com uma janela pop-up de informação requer um pouco de código, pois você terá que criar o marcador, a janela de informação e um handler para o click event* do marcador (que abre a janela de informação). Dado que estamos em uma distração, vamos passar pelo assunto bem rápido, mas, neste ponto do livro, você tem tudo que precisa para estar à altura!

- Comece criando uma nova função, addMarker, e em seguida use a API do Google para criar um marcador:

A função addMarker pega um mapa, latitude e longitude no estilo Google, um título para o marcador e também algum conteúdo para a janela de informação.

```
function addMarker(map, latlong, title, content) {
```

```
    var markerOptions = {
        position: latlong,
        map: map,
        title: title,
        clickable: true
    };
```

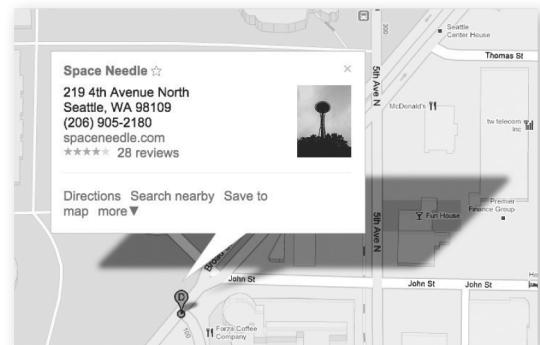
```
    var marker = new google.maps.Marker(markerOptions);
```

```
}
```

Criamos um objeto options com a latitude e longitude, o mapa, o título e se queremos ou não que o marcador seja clicável...

...ajustamos para true aqui, pois queremos que seja possível mostrar uma janela de informação quando for clicado.

Então, criamos o objeto marker, usando outro construtor da API do Google, e o passamos ao objeto markerOptions que criamos.



Quando você procura por um item no Google Maps, vê um alfinete vermelho assinalando o local do resultado da busca.



- ② Em seguida, criaremos a janela de informação, definindo algumas opções específicas para, então, criar um novo objeto InfoWindow com a API do Google. Adicione o código abaixo em sua função addMarker:

```
function addMarker(map, latlong, title, content) {
```

← Nosso outro código ainda está aqui,
estamos apenas salvando algumas árvores...

```
var infoWindowOptions = {  
    content: content,  
    position: latlong  
};
```

Agora vamos definir algumas opções para a janela de informação

← Precisamos do conteúdo...

← ...e da latitude e da longitude.

`var infoWindow = new google.maps.InfoWindow(infoWindowOptions);`

Com isso criamos a janela de informação.

```
google.maps.event.addListener(marker, "click", function() {  
    infoWindow.open(map);  
});
```

Quando o marcador é clicado, esta função é chamada e InfoWindow abre no mapa.

Passamos ao ouvinte uma função que é chamada, quando o usuário clica no marcador.

Depois, usaremos o método addListener do Google Maps para adicionar um "listener" para o click event*. Um listener é exatamente como um handler, como onload e onclick, que você já viu anteriormente.

- ③ Agora, tudo o que resta fazer é chamar a função `addMarker` da `showMap`, certificando-se de passar todos os argumentos corretos para os quatro parâmetros. Acrescente isso no final de sua função `showMap`:

```
var title = "Your Location";  
  
var content = "You are here: " + coords.latitude + ", " + coords.longitude;  
  
addMarker(map, googleLatLong, title, content);
```

Passamos o mapa e os objetos googleLatAndLong que criamos usando a API do Google Maps...

... e uma string de título e outra de conteúdo para o marcador.



Testando o marcador



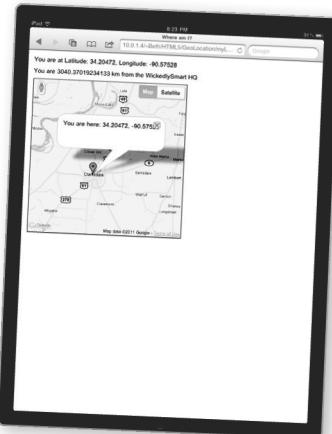
Pegue todo o código adicionado para addMarker, atualize o showMap para acionar o addMarker e recarregar a página.

Você verá um mapa com um marcador e sua localização nele.

Experimente clicar no marcador. Verá, logo em seguida, uma janela pop-up com sua latitude e longitude.

Isso é ótimo, pois agora sabe exatamente onde está (caso estivesse perdido ou algo do gênero...).

Eis como parece nosso mapa
com o marcador e a janela
pop-up de informações.



Vá até <http://wickedlysmart.com/hfhtml5/chapter5/marker/myLoc.html>

Outras coisas legais que você pode fazer com a API do Google Maps



Apenas tocamos a ponta do iceberg de todas as coisas que podemos fazer com a API do Google Maps e, embora esta API esteja muito longe do assunto principal deste livro, você já está bem treinado para ser capaz de fuçar nela por conta própria. Aqui estão algumas coisas para se considerar utilizará-la e alguns pontos de partida.

Controles: Por padrão, seu mapa do Google inclui muitos controles, como o controle de zoom, o controle panorâmico, um controle para alternar entre visualização de mapa ou de satélite e até mesmo um controle para street view (o bonequinho acima do controle do zoom). Você poderá acessar estes controles programaticamente a partir do JavaScript para usá-los em seus aplicativos.

Serviços: Sempre faz uma visita às direções do Google Maps? Se assim o for, então você já usou o serviço Directions (direções). Você tem acesso a direções, assim como outros serviços, como distância e street view, por meio das APIs de serviços do Google Maps.

Camadas: as camadas fornecem outra vista superior do mapa pelo Google; digamos, uma camada de temperatura. Se estiver comutando, você poderá verificar o trânsito com tal camada. É possível criar camadas personalizadas, como marcadores personalizados, suas fotos e, basicamente, tudo o mais que puder imaginar usando as APIs de camada do Google Maps.

Tudo isto está disponível por meio da API JavaScript do Google Maps. Para prosseguir em suas experiências, dê uma olhada na seguinte documentação:

<http://code.google.com/apis/maps/documentation/javascript/>



Geolocalização Exposta

Entrevista da semana:

Uma conversa com uma prensa API HTML5

Use a Cabeça!: Bem-vinda, Geolocalização. Já vou dizendo de antemão que estou um pouco surpreso de vê-la aqui.

Geolocalização: Por quê?

Use a Cabeça: Você nem mesmo faz parte “oficialmente” das especificações do HTML5, e olha você aí, a primeira API a receber um capítulo! Como se sente?

Geolocalização: Bem, você está certo quanto a eu ser definida em uma especificação à parte da HTML5, mas *sou* uma especificação oficial do W3C. É só olhar em volta e verá que qualquer dispositivo móvel que “vale o que come” já me possui implementada em seu navegador. Digo, quão bom é um aplicativo web para dispositivo móvel sem mim?

Use a Cabeça: Então, quais os tipos de aplicativos que a utilizam?

Geolocalização: Na verdade, a maioria dos aplicativos que as pessoas usam; desde aplicativos que permitem atualizar seu status e incluir informação de geolocalização, até aplicativos de câmera que gravam onde as fotos são tiradas, ou mesmo aplicativos sociais que encontram amigos por perto ou permitem “ingressar” em vários locais. Caramba, as pessoas estão me usando até para gravar por onde elas vão de bicicleta, ou correm, ou comem, ou para saber aonde estão indo.

Use a Cabeça: Sua API parece meio simplista, quero dizer, você tem o quê? Umas duas propriedades ou métodos no total?

Geolocalização: Pequeno e simples é poderoso. Quantas reclamações sobre mim você vê por aí afora? Nenhuma. Tenho o que todo desenvolvedor precisa e os aplicativos com localização estão falhando às dúzias pelo mercado. Além disso, menor

é igual a rápido e fácil de aprender, certo? Talvez seja por isso que eu seja a primeira API com seu próprio capítulo.

Use a Cabeça: Vamos falar sobre suporte.

Geolocalização: É um tópico curto, pois tenho suporte em quase todos os navegadores, de desktop ou mobile.

Use a Cabeça: Ok, então uma coisa que sempre quis perguntá-la: quão boa você é num dispositivo que não possua GPS?

Geolocalização: Há uma concepção errônea de que sou, de certa forma, dependente de GPS. Existem outras formas de se determinar a localização hoje, por meio da triangulação de torres de celular utilizando endereços IP, entre outros. Se você possui GPS, ótimo, pois realmente poderei ajudá-lo ainda mais; mas, se não, há uma porção de formas de obter a localização.

Use a Cabeça: Ajudar ainda mais?

Geolocalização: Se você tiver um dispositivo móvel bom o bastante, posso lhe oferecer altitude, direção, velocidade, todo o tipo de coisas.

Use a Cabeça: Digamos que nenhum desses métodos funcione, ou seja, GPS, endereço IP, triangulação; então para que você servirá?

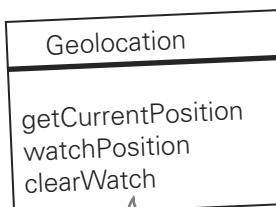
Geolocalização: Bem, não posso garantir que você sempre conseguirá uma localização, mas está tudo bem, pois ofereço uma forma boa de lidar com falhas. Tudo o que precisará fazer é me dar um error handler e poderei acioná-lo se tiver algum problema.

Use a Cabeça: Bom saber. Bem, nosso tempo está acabando. Obrigado, Geolocalização, por estar aqui conosco e parabéns por ter sido promovida a uma verdadeira W3C standard.

Enquanto isso, de volta à API de Geolocalização...

Já trilhamos uma boa distância com a API de Geolocalização: determinamos nossa localização, computamos distâncias a outras localidades, lidamos com situações de erro da API e até acrescentamos um mapa usando a API do Google Maps. Ainda não está, porém, na hora de descansar, pois estamos no ponto de nos familiarizarmos com as partes interessantes da API. Também chegamos ao ponto entre saber sobre a API e ter pleno domínio sobre ela; portanto, vamos continuar!

Uma coisa que precisamos fazer antes de seguir em frente é dar uma olhada mais de perto na API em si. Falamos bastante sobre ela, mas nunca *olhamos para ela* de fato. Como vínhamos falando, a API é, realmente, bem simples, possuindo apenas três métodos: `getCurrentPosition` (o qual você já sabe alguma coisa a respeito), `watchPosition` (que você entenderá em pouco tempo) e `clearWatch` (que, você adivinhou, está relacionado com o anterior). Antes de chegar nesses dois novos métodos, vamos dar outra olhada no `getCurrentPosition` e em alguns objetos relacionados, como os objetos `Position` e `Coordinates`. Você encontrará algumas coisas novas lá que não conhecia.



Os métodos que
são parte da API
de Geolocalização.

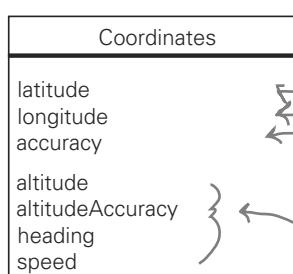
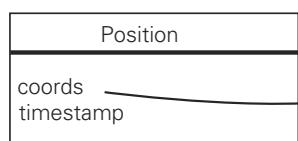
O error handler é acionado quando
o browser não pode determinar sua
localização. Como vimos, há diversas
razões possíveis para isso.

`getCurrentPosition(successHandler, errorHandler, positionOptions)`

Lembre-se de que o success handler (ou callback) é acionado quando uma localização é determinada, e é passado um objeto `position`.

E temos outro parâmetro que ainda não usamos que nos permite arrumar o comportamento da geolocalização.

Sabemos sobre latitude e longitude, mas existem outras propriedades no objeto `coordinates`.



Sabemos a respeito da propriedade `coords`, mas há também uma propriedade `timestamp` em posição que contém o tempo em que foi criado o objeto `position`. Isso pode ser útil para saber a idade da localização.

É garantido que três estejam lá: lat, long e accuracy. O resto pode ou não ter suporte, dependendo de seu dispositivo.

Podemos falar sobre sua precisão?

Encontrar sua localização não é exatamente uma ciência. Dependendo do método que o browser utilizar, você poderá apenas saber qual o Estado, cidade ou quadra em que você está. Portanto, novamente, com dispositivos mais avançados, você poderá descobrir sua localização num raio de 10 metros, associada com velocidade, direção e altitude.

Então, como escrevemos código, dada esta situação? Os designers da API de Geolocalização fizeram um pequeno contrato conosco: toda vez que eles nos derem uma localização, também nos darão a precisão da localização em metros, com cerca de 95% de grau de confiança. Portanto, por exemplo, podemos saber nossa localização com precisão de 500 metros, o que significa que podemos contar com certeza com a localização, contanto que incluamos um raio de 500 metros. Para 500 metros, estariamos seguros, por exemplo, para indicar recomendações de cidades ou vizinhanças, porém talvez não queiramos fornecer direções rua a rua. Em qualquer um dos casos, a decisão será de seu aplicativo sobre como ele vai querer usar a precisão dos dados.

Chega de falação, vamos descobrir como está sua precisão na localização atual. Como acabou de ver, a informação de precisão é parte do objeto coordinates. Vamos tirá-la de lá e usá-la na função displayLocation.

```
function displayLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var div = document.getElementById("location");

    div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: " + longitude;
    div.innerHTML += " (with " + position.coords.accuracy + " meters accuracy)";

    var km = computeDistance(position.coords, ourCoords);
    var div = document.getElementById("distance");
    distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";
    showMap(position.coords);
}
```

↑ Aqui usamos a propriedade de posição accuracy e anexamos no fim da innerHTML da <div>.

Teste de Precisão



Certifique-se de ter acrescentado esta linha em seu código e carregue a página. Agora poderá ver quão precisa é sua localização. Garanta que experimentará em qualquer dispositivo que tiver.

Experimente: <http://wickedlysmart.com/hfhtml5/chapter5/accuracy/myLoc.html>



"Onde quer que vá, lá está você"

A origem desta frase foi arduamente debatida. Alguns dizem que sua primeira menção real foi no filme *Buckaroo Banzai*, outros remontam suas origens ao texto Zen Budista, e ainda outros citam diversos livros, filmes e canções populares.

Não importa a fonte, chegou para ficar, e mais ainda depois desse capítulo, pois vamos transformá-la num pequeno aplicativo web chamado "Onde quer que vá, lá está você". Sim, há um aplicativo para isso!

Vamos precisar, porém, de um pouco de sua participação, o leitor, porque para esse aqui você terá de (desculpe-nos o termo) levantar o traseiro e se mexer um pouco.

O que vamos fazer é estender nosso código atual de forma que ele rastreie seus movimentos em tempo real. Para fazer isso, vamos juntar tudo, incluindo os últimos dois métodos da API de Geolocalização, e criar um aplicativo que o rastreie em tempo quase real.



Onde você se encaixa nesse debate? O dito é um produto do Banzai Institute ou possui origens na literatura Zen?

Como vamos rastrear seus movimentos

Você já foi informado de que a API de Geolocalização tem um método `watchPosition`. Este método faz exatamente o que diz: presta atenção em seus movimentos e reporta sua localização de acordo com as mudanças da mesma. O método `watchPosition`, na verdade, parece justamente com o método `getCurrentPosition`, porém se comporta um pouco diferente: aciona seu `success` handler repetidamente cada vez que sua posição muda. Vejamos como funciona:



Browser

- ① Seu aplicativo aciona `watchPosition`, passando uma função `success` handler.

- ② `watchPosition` fica no background e constantemente monitora sua posição.



`position.coords.latitude`
`position.coords.longitude`



- ④ `watchPosition` continua a monitorar sua posição (e reportá-la a seu `success` handler) até que limpe-a, acionando `clearWatch`.

- ③ Quando sua posição muda, `watchPosition` aciona sua função `success` handler para reportar sua nova posição.



Inicializando o aplicativo

Usaremos nosso código anterior como um ponto de partida; primeiro, vamos adicionar alguns botões ao HTML de forma que possamos começar e parar o rastreamento de sua localização. Por que precisamos dos botões? Bem, primeiro de tudo, os usuários não querem ser rastreados toda hora e, normalmente, eles querem ter controle sobre isso. Contudo, há uma outra razão: checar sua posição constantemente é uma operação de gasto de energia intenso para um dispositivo móvel e, se deixá-lo acionado a todo tempo, diminuirá a vida útil de sua bateria. Portanto, antes de tudo, vamos atualizar o HTML para adicionar um formulário e dois botões: um para começar a ver sua posição e outro para parar.

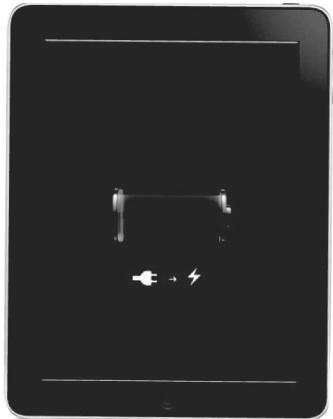
```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Onde quer que vá, lá está você</title>
  <script src="myLoc.js"></script>
  <link rel="stylesheet" href="myLoc.css">
</head>
<body>
  <form>
    <input type="button" id="watch" value="Watch me">
    <input type="button" id="clearWatch" value="Clear watch">
  </form>
  <div id="location">
    Your location will go here.
  </div>
  <div id="distance">
    Distance from WickedlySmart HQ will go here.
  </div>
  <div id="map">
  </div>
</body>
</html>
```

Rastrear um usuário em tempo real pode sugar mesmo sua bateria. Certifique-se de informar seu usuário a respeito do rastreamento, e dar a ele certo controle sobre isso também.

Estamos acrescentando um elemento de formulário com dois botões, um para começar a localização, com uma id de "watch", e outro para parar a localização, com uma id de "clearWatch".

Vamos reutilizar nossas velhas `<div>`s para enviar informação sobre localização em tempo real.

Já voltaremos a nos preocupar com o mapa do Google.



Retrabalhando nosso antigo código...

Agora, precisamos adicionar button click handlers para os dois botões. Iremos adicioná-los à função `getMyLocation` apenas se houver suporte para geolocalização. Já que vamos controlar todo o rastreamento de geolocalização usando os dois botões, removeremos o acionamento existente para `getCurrentPosition` de `getMyLocation`. Vamos em frente remover aquele código e adicionar dois handlers: `watchLocation` para o botão de início da localização e `clearWatch` para o botão de limpeza:

```
function getMyLocation() {  
    if (navigator.geolocation) {  
        navigator.geolocation.getCurrentPosition(displayLocation, displayError);  
        var watchButton = document.getElementById("watch");  
        watchButton.onclick = watchLocation;  
        var clearWatchButton = document.getElementById("clearWatch");  
        clearWatchButton.onclick = clearWatch;  
    }  
    else {  
        alert("Oops, no geolocation support");  
    }  
}
```

Se o browser der suporte à geolocalização, vamos adicionar nossos button click handlers, mas não há sentido em adicioná-los se a geolocalização não for suportada.

Vamos acionar `watchLocation` para iniciar e `clearWatch` para limpar.

Escrevendo o handler `watchLocation`

Neste ponto, eis o que estamos tentando fazer: quando o usuário clicar no botão de início, significa que ele quer começar a rastrear sua posição. Portanto, usaremos o método `geolocation.watchPosition` para iniciar sua localização. O método `geolocation.watchPosition` possui dois parâmetros: um success handler e um error handler; assim, vamos reutilizar aqueles que já temos. Ele também retorna um `watchId`, que pode ser usado a qualquer tempo para cancelar o comportamento de localização. Vamos guardar o `watchId` numa variável global, que será usada quando escrevermos o click handler para o botão de limpar. Abaixo, segue o código para a função `watchLocation` e para o `watchId`. Vá em frente e acrescente este código em `myLoc.js`:

```
var watchId = null;  
function watchLocation() {  
    watchId = navigator.geolocation.watchPosition(displayLocation,  
                                                   displayError);  
}  
Estamos acionando o método watchPosition, passando o success handler que já  
escrevemos, o displayLocation e nosso error handler existente, displayError.
```

Acrescente `watchId` no topo de seu arquivo como uma variável global. Vamos inicializá-lo como null. Precisaremos dele mais tarde para limpar a localização.

Escrevendo o handler clearWatch

Agora vamos escrever o handler para limpar a atividade de localização. Para fazer isso, precisamos pegar o watchId e passá-lo no método geolocation.clearWatch.

```
function clearWatch() { ← Certifique-se de haver um watchId e depois...
    if (watchId) {
        navigator.geolocation.clearWatch(watchId); ← ... acione o método
        watchId = null;                            geolocation.
    }                                              clearWatch, passando
}                                              o watchId. Isso faz
                                                a localização parar.
```

Ainda precisamos fazer uma pequena atualização para o displayLocation...

Há uma pequena mudança que precisamos fazer e esta envolve o código do Google Maps que escrevemos antes. Neste código, acionamos showMap para mostrar o mapa do Google. Agora, o showMap cria um novo mapa em sua página e isso é uma coisa que você só vai querer fazer uma vez. Lembre-se, porém, que quando começar a pôr para funcionar sua localização com o watchPosition, o displayLocation será chamado todas as vezes que houver uma atualização de sua posição.

Para nos certificarmos de chamar o showMap apenas uma vez, vamos primeiro testar para ver se o mapa existe, então, se não existir, chamaremos o showMap. Caso contrário, o showMap já foi chamado (e já criou o mapa) e não precisamos chamá-lo novamente.

```
function displayLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;

    var div = document.getElementById("location");
    div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: " + longitude;
    div.innerHTML += " (with " + position.coords.accuracy + " meters accuracy)";

    var km = computeDistance(position.coords, ourCoords);
    var distance = document.getElementById("distance");
    distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";

    if (map == null) { ← Se ainda não chamou o showMap, chame-o
        showMap(position.coords); → agora; caso contrário, não precisaremos chamá-lo toda vez que displayLocation for chamado.
    }
}
```

Hora de seguir em frente!



Certifique-se de que o código novo esteja todo digitado e recarregue sua página, myLoc.html. Agora, para verdadeiramente testar esta página você precisará “relocalizar” para ter sua posição atualizada. Então, vá dar uma caminhada, pegue sua bicicleta, entre no carro, ou use qualquer que seja seu meio de transporte favorito.

Não é necessário que se diga que, se você estiver rodando no desktop, este aplicativo será bem chato (já que você não poderá levá-lo com você). Sendo assim, realmente é necessário utilizar um dispositivo móvel para este teste. Se precisar de ajuda para iniciar uma versão hospedada com seu dispositivo móvel, deixamos uma cópia deste código em:

<http://wickedlysmart.com/hfhtml5/chapter5/watchme/myLoc.html>.

Aqui está nosso teste...

Estes números
vão se atualizar
à medida que
você se move.

↗



Perceba que o mapa
vai se centralizar
em sua localização
inicial, por ora...

Vá até: <http://wickedlysmart.com/hfhtml5/chapter5/watchme/myLoc.html>

Perguntas Idiotas

P: Como posso controlar as taxas com que o browser fornece atualizações de minha localização quando usar watchPosition?

R: Não pode. O browser determina qual é a taxa de atualização otimizada e decide quando você mudou de posição.

P: Por que minha localização muda algumas vezes, quando eu carrego a página pela primeira vez, mesmo eu estando parado?

R: Você se lembra que dissemos que o browser pode usar alguns métodos para determinar sua localização? Dependendo do método (ou métodos) que o browser estiver usando, a precisão da localização pode se modificar de tempos em tempos. Normalmente, a precisão melhora, mas, às vezes (digamos, você estava dirigindo em uma área rural com apenas uma torre de celular), pode piorar. Você pode utilizar sempre a propriedade accuracy no objeto position.coords para ficar atento à precisão (accuracy).

P: Posso usar as propriedades altitude e altitudeAccuracy do objeto coordinates?

R: Essas propriedades não possuem garantias de terem suporte (e, obviamente, só terão suporte em dispositivos móveis de última geração); portanto, você precisa ter certeza de que seu código pode lidar com tal situação adversa.

P: O que são heading e speed?

R: Heading é a direção para a qual você está viajando e speed é quanto rápido está indo. Pense no caso de estar num carro em direção ao norte, na Interestadual 5, a 90 km/h. Sua heading é norte e sua speed é 90 km/h. Se estiver em seu carro, no estacionamento do Starbuzz Coffee, então sua speed será 0 e você não possuirá heading (pois não está se movendo).

P: Quando traço a distância do meu local para o seu, é muito mais longe que o acusado no aplicativo. Por quê?

R: Lembre-se de que nossa função distance está computando a distância "em linha reta". Sua ferramenta de traçar rotas provavelmente dará a distância ao dirigir.

Aponte o seu lápis



Abaixo, você encontrará uma implementação alternativa para o displayLocation. Consegue adivinhar o que isso faz? Dê uma olhada e escreva sua resposta abaixo. Se você é um aventureiro, vá em frente!

```
distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";
if (km < 0.1) {
    distance.innerHTML = "You're on fire!";
} else {
    if (prevKm < km) {
        distance.innerHTML = "You're getting hotter!";
    } else {
        distance.innerHTML = "You're getting colder...";
    }
}
prevKm = km;
```

Escreva aqui o que você acha que essa coisa faz



.....

Você tem algumas opções...

Até então, estivemos distantes do terceiro parâmetro `getCurrentPosition` (e `watchPosition`): o parâmetro `positionOptions`. Com esse parâmetro, podemos controlar como a geolocalização computa seus valores. Vamos dar uma olhada nos três parâmetros, ao mesmo tempo em que vemos seus valores padrão:

```
var positionOptions = {  
    enableHighAccuracy: false,  
    timeout: Infinity,  
    maximumAge: 0  
}
```

Primeiro, temos uma propriedade que ativa a alta precisão (vamos falar sobre o que isso significa em alguns instantes).

A opção `timeout` controla quanto tempo o browser levará para determinar sua localização. Por padrão, ela é ajustada para infinito, significando que o browser consegue todo o tempo que precisar.

Você pode reiniciar isto para um valor em milissegundos, digamos `10000`, dando ao browser dez segundos para encontrar uma localização, senão o error handler é chamado.

Finalmente, a opção `maximumAge` ajusta a idade máxima que uma localização pode ter, antes do browser ter de recalcular a localização. Por padrão, o valor será zero, significando que o browser sempre terá que recalcular sua localização (toda vez que `getCurrentPosition` for chamado).

Podemos falar sobre sua precisão novamente?

Já vimos que cada posição passada para nós pela API da Geolocalização possui uma propriedade `accuracy` (precisão). É possível, porém, também dizer à API da Geolocalização que queremos apenas o resultado mais preciso a que chegar. Agora, isso é apenas considerado como uma sugestão pelo browser e, de fato, implementações diferentes podem fazer coisas diferentes com a sugestão. Além disso, enquanto que esta opção não parece ser grande coisa, possui várias implicações. Por exemplo, se você não se importa com o fato de que seus resultados não sejam super precisos — você pode não ligar para o fato de que seu usuário está em Baltimore — a API pode ser capaz de dizer-lhe isso de forma bem rápida e barata (em termos de consumo de energia). Se, por outro lado, você precisa saber a rua na qual se encontra seu usuário, tudo bem, mas a API talvez tenha que ativar o GPS, utilizando muita energia para extrair tal informação. Com a opção `enableHighAccuracy`, você está dizendo à API que você precisa da localização mais precisa que puder, mesmo tendo seus custos. Tenha, no entanto, sempre em mente que usar esta opção não garante que o browser possa lhe oferecer uma localização mais apurada.



O mundo das timeouts e maximum age...

Revisemos mais uma vez o que são as opções `timeout` e `maximumAge`:

timeout: Esta opção diz ao browser quanto **tempo** leva para determinar a localização do usuário. Perceba que, se o usuário for avisado para aprovar a solicitação de localização, o timeout não é iniciado até que seja aceito. Se o browser não puder determinar uma nova localização dentro do número de milissegundos especificado no timeout, o error handler é chamado. *Por padrão, esta opção é definida como infinito (Infinity).*

maximumAge: Esta opção diz ao browser qual a **idade** que a localização pode ter. Portanto, se o browser possuir uma localização determinada em sessenta segundos e a `maximumAge` for ajustada a 90000 (90 segundos), uma chamada para `getCurrentPosition` retornará uma posição existente e armazenada (o browser não vai querer uma nova). Se, porém, a `maximumAge` estiver ajustada para 30 segundos, o browser será forçado a determinar uma nova posição.



Portanto, ao usar `maximumAge`, posso realmente ajustar a frequência com a qual meu browser recalcula ou determina minha posição. Vejo que usar isso pode tornar meu aplicativo mais rápido e mais econômico para a bateria. E quanto à `timeout`? Como posso usá-la para melhorar as coisas?

Você está certo em relação a essa história da `maximumAge`. Pense na `timeout` assim: quando estiver usando `maximumAge`, tendo assim um resultado (armazenado) ultrapassado, desde que o resultado seja mais novo que o `maximumAge` que você especificou, funciona muito bem para otimizar a performance de seu aplicativo. O que acontece, no entanto, quando a idade da posição exceder a `maximumAge`? Bem, o browser sai para procurar outra. Mas, e se você não se importa *tanto assim* — digamos que você fosse a uma nova localização, se houvesse, mas, ao contrário, você não precisasse disso agora. Bem, você poderia ajustar a `timeout` para 0 e, se houvesse um resultado que passasse o teste `maximumAge`, ótimo, prontinho; do contrário, a chamada falharia imediatamente e chamaria seu error handler (com um erro de código `TIMEOUT`). Esse é apenas um exemplo dos jeitos criativos com que você pode usar `timeout` e `maximumAge` para ajustar o comportamento de seu aplicativo.



QUEM FAZ O QUÊ?

Abaixo, você verá algumas opções para a API da Geolocalização. Para cada opção, existe um comportamento correlato.

`{maximumAge: 600000}`

Quero apenas posições armazenadas com menos de 10 minutos. Se não houver nenhuma posição com esse tempo, pedirei uma nova posição, mas apenas se conseguir uma em 1 segundo ou menos.

`{timeout: 1000, maximumAge: 600000}`

Usarei uma posição armazenada se o browser encontrar uma que tenha menos de 10 minutos, ou então vou querer uma posição novinha em folha.

`{timeout: 0, maximumAge: Infinity}`

Quero apenas posições novas. O browser pode demorar o quanto for para arranjá-las para mim.

`{timeout: Infinity, maximumAge: 0}`

Quero apenas posições armazenadas. Vou pegar uma de qualquer idade. Se não houver quaisquer posições, então chamarei o error handler. Sem novas posições para mim! Sou para uso offline.

Como especificar opções

Uma das coisas legais a respeito do JavaScript é que, se quisermos especificar um conjunto de opções num objeto, podemos apenas digitar um objeto literal, exatamente no meio de nossa chamada de método. É assim que se faz: digamos que o objetivo seja habilitar a precisão e também ajustar a idade máxima da localização para 60 segundos (ou 60.000 milissegundos). Poderíamos criar opções como essa:

```
var options = {enableHighAccuracy: true, maximumAge: 60000};
```

Então, passe options tanto para getCurrentPosition quanto para watchPosition, assim:

```
navigator.geolocation.getCurrentPosition(  
    displayLocation,  
    displayError,  
    options);
```

Aqui estamos apenas passando nossas opções adiante usando a variável de opções.

Você está começando a perceber que o JavaScript bota pra quebrar? Bem, pelo menos nós achamos que sim. ☺

Ou, poderíamos apenas escrever as opções de objetos enfileiradas, assim:

```
navigator.geolocation.getCurrentPosition(  
    displayLocation,  
    displayError,  
    {enableHighAccuracy: true, maximumAge:  
     60000});
```

Você verá muito esta técnica sendo usada no código JavaScript.

Aqui estão as opções, escritas como um objeto literal, bem na chamada de função! Alguns diriam que isto é mais fácil e mais legível como código.

Agora que você conhece as opções, o que elas fazem e como especificá-las, deveríamos usá-las. Faremos isso, mas lembre-se de que elas são feitas para ajustar seu aplicativo, que terá seus próprios requisitos exclusivos. Estas opções também são afetadas pelo seu dispositivo, pela implementação do browser e pela rede; portanto, você precisará saber mexer sozinho para explorar todas as capacidades.



Test Drive do Check-up dos Diagnósticos

Quando você rodou antes os diagnósticos, entendeu o caso-teste em que esperou e esperou e nada aconteceu? É bem provável que isso tenha acontecido por causa do timeout infinito. Em outras palavras, o browser vai esperar para sempre para obter a localização, desde que não encontre alguma condição de erro. Certo, agora você sabe como consertar isso, pois podemos forçar a API de Geolocalização para ser um pouco mais expediente, ajustando seu valor de timeout. Veja abaixo como fazer isso:

```
function watchLocation() {  
    watchId = navigator.geolocation.watchPosition(  
        displayLocation,  
        displayError,  
        {timeout:5000});
```

} Sinta-se à vontade e tente ajustar os valores de opção.

Ao ajustar o timeout para 5000 milissegundos (5 segundos), você está se certificando de que o browser não pare e fique esperando para sempre tentando achar uma localização.

~~NÃO~~ TENTE ISSO EM CASA (LEVANDO A GEOLOCALIZAÇÃO ATÉ O LIMITE)

Não seria divertido ver quão rápido seu browser consegue encontrar sua localização? Poderíamos tornar isso o mais difícil possível para ele:

- vamos pedir-lhe que ative precisão máxima,
- não vamos permitir que ele use cache (ajustando maximumAge para 0)
- vamos cronometrá-lo, ajustando a opção timeout para 100, e então aumentar o timeout toda vez que falhar.

Cuidado: não sabemos se todos os dispositivos e suas baterias estão prontos para isso. Portanto, é por sua conta e risco!

Segue abaixo como poderão aparecer as opções iniciais:

```
{enableHighAccuracy: true, timeout:100, maximumAge:0}
```

Começaremos
aqui...

```
{enableHighAccuracy: true, timeout:200, maximumAge:0}
```

e se falhar, dê-
lhe mais tempo...

```
{enableHighAccuracy: true, timeout:300, maximumAge:0}
```

e assim por
diante...

Agora verifique o código na próxima página, pois você o achará bem interessante. Vá em frente e digite-o — simplesmente adicione-o ao seu JavaScript em myLoc.js. Experimente-o em seus vários dispositivos e anote os resultados aqui:

aqui o dispositivo ↗

↓ aqui o tempo

NO _____ ENCONTRADO EM _____ Milissegundos



Experimente online: [http://wickedlysmart.com/hfhtml5/
chapter5/speedtest/speedtest.html](http://wickedlysmart.com/hfhtml5/chapter5/speedtest/speedtest.html)





```

var options = { enableHighAccuracy: true, timeout:100, maximumAge: 0 };
window.onload = getMyLocation;
function getMyLocation() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(
            displayLocation,
            displayError,
            options);
    } else {
        alert("Oops, no geolocation support");
    }
}
function displayError(error) {
    var errorTypes = {
        0: "Unknown error",
        1: "Permission denied",
        2: "Position is not available",
        3: "Request timeout"
    };
    var errorMessage = errorTypes[error.code];
    if (error.code == 0 || error.code == 2) {
        errorMessage = errorMessage + " " + error.message;
    }
    var div = document.getElementById("location");
    div.innerHTML = errorMessage;
    options.timeout += 100; ←
    navigator.geolocation.getCurrentPosition(
        displayLocation,
        displayError,
        options);
    div.innerHTML += "... checking again with timeout=" + options.timeout;
}
function displayLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var div = document.getElementById("location");
    div.innerHTML = "You are at Latitude: " + latitude +
        ", Longitude: " + longitude;
    div.innerHTML += " (found in " + options.timeout + " milliseconds)";
}

```

Comece inicializando nossas opções com uma timeout de 100 e maximumAge de 0.

Faça o de sempre aqui, com displayLocation e displayError como nossos success e error handlers, e passando a opções como o terceiro parâmetro.

Faremos o error handler primeiro.

Este código aqui é o mesmo...

Mas, no caso de uma falha, vamos aumentar a opção timeout para 100ms e tentar novamente. Vamos deixar que o usuário saiba que também estamos tentando novamente.

Quando o browser captar sua posição com sucesso, permitiremos ao usuário saber quanto tempo ele levou para isso.

Vamos finalizar este aplicativo!

Se parar e pensar a respeito, verá que, com apenas um pouco de HTML e JavaScript, você criou um aplicativo web que não apenas pode determinar sua localização, mas também pode rastreá-lo e mostrá-lo em *tempo quase real*. Nossa! O HTML amadureceu (assim como nossas habilidades!).

Falando do aplicativo, você não acha que precisa dar uma polida para terminá-lo? Por exemplo, poderíamos mostrar sua posição no mapa à medida que se move e ir um pouco mais à frente, mostrando onde você esteve também, para criar um caminho pelo mapa.

Vamos escrever uma função para manter o mapa centrado em sua localização à medida que se move e colocar um novo marcador a cada vez que tivermos uma nova posição:

Ok, vamos chamar esta função scrollMapToPosition e vamos passar para ela umas coordenadas da posição.

As coordenadas serão sua última nova posição, portanto vamos centrar o mapa naquela localização e pôr um marcador ali também.

```
function scrollMapToPosition(coords) {
    var latitude = coords.latitude;
    var longitude = coords.longitude;
    var latlong = new google.maps.LatLng(latitude, longitude);
    map.panTo(latlong);
    addMarker(map, latlong, "Your new location", "You moved to: " +
        latitude + ", " + longitude);
}
```

Primeiro, vamos pegar a nova lat e long e criar um google.maps. Objeto LatLong neles!

O método panTo do mapa leva o objeto LatLong e faz a rolagem do mapa, de maneira que sua nova localização fique no centro dele.

Finalmente, vamos adicionar um marcador para sua nova localização, usando a função addMarker que escrevemos mais cedo, passando para o mapa o objeto LatLong, um título e algum conteúdo para o novo marcador.



Integrando nossa nova função

Agora, tudo que precisamos é atualizar a função `displayLocation` para chamar `scrollMapToPosition` cada vez que sua posição mudar. Lembre-se de que, da primeira vez que `displayLocation` for chamada, vamos chamar `showMap` para criar o mapa e mostrar um marcador para sua localização inicial. Toda vez depois disso, precisaremos apenas chamar `scrollMapToPosition` para adicionar um novo marcador e recentralizar o mapa.

```
function displayLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var div = document.getElementById("location");
    div.innerHTML = "You are at Latitude: " + latitude
                    + ", Longitude: " + longitude;
    div.innerHTML += " (with " + position.coords.accuracy + " meters accuracy)";
    var km = computeDistance(position.coords, ourCoords);
    var distance = document.getElementById("distance");
    distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";
    if (map == null) {
        showMap(position.coords);
    } else {
        scrollMapToPosition(position.coords);
    }
}
```

Na primeira vez que `displayLocation` for chamada, precisaremos desenhar o mapa e adicionar o primeiro marcador.

Depois disso, tudo o que precisamos fazer é adicionar um novo marcador ao mapa existente.

}

E mais uma vez...



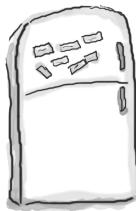
Recarregue sua página e comece a se mover... Seu mapa está seguindo você? Você deverá ver uma trilha de marcadores sendo adicionados à medida que se move (a menos que esteja parado em frente ao seu computador!).

Então, apresentamos este aplicativo como prova concreta de que “onde quer que vá, lá está você”.

Vá até <http://wickedlysmart.com/hfhtml5/chapter/watchmepan/myLoc.html>



Nossa trilha de marcadores numa viagem recente a partir do QG da Wickedly Smart até o secreto covil subterrâneo — oh, espere, não deveríamos ter dito isso...



Ímãs de Geladeira

Antes de concluirmos este capítulo, achamos que você realmente quer polir este aplicativo. Percebeu (sob algumas circunstâncias) que existem alguns marcadores em excesso sendo adicionados ao mapa quando observa sua posição?

O que está acontecendo é que `watchPosition` está detectando seu movimento frequentemente, portanto, está chamando o `success` handler do `displayLocation` toda vez que dá um passo. Uma maneira de consertar isso é adicionar um pouco de código, de forma que tenhamos que nos mover bem, digamos uns 20 metros para finalidades de teste, antes de criarmos um novo marcador.

Já temos uma função que irá computar a distância entre duas coordenadas (`computeDistance`). Tudo que temos de fazer é salvar nossa posição toda vez que `displayLocation` for chamado e verificar se a distância entre a posição anterior e a nova é maior que 20 metros antes de chamar `scrollMapToPosition`. Você encontrará um pouco do código para isso logo abaixo; terminar é sua tarefa. Cuidado, você terá que usar alguns ímãs mais de uma vez!





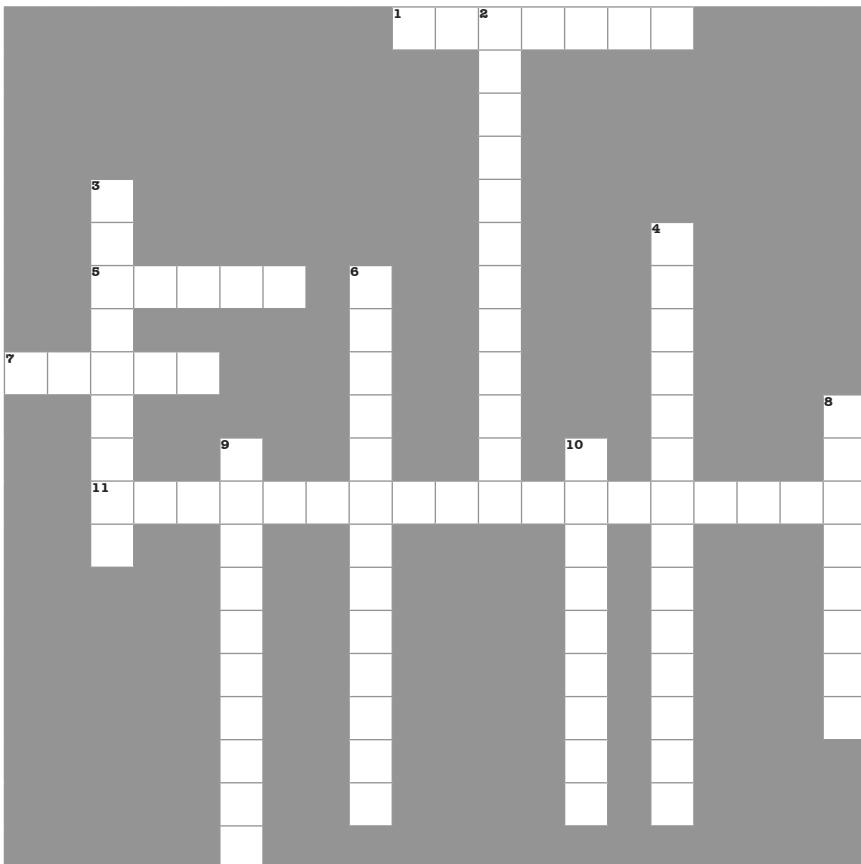
PONTOS DE BALA

- A Geolocalização não é “oficialmente” parte da especificação HTML5, mas é considerada parte da “família” de especificações HTML5.
- Há uma porção de maneiras para se determinar sua localização, dependendo do dispositivo que tiver.
- GPS é um método mais apurado para se obter sua localização do que a partir de triangulação de torres de celular ou rede IP.
- Dispositivos móveis sem GPS podem usar a triangulação de torres de celular para determinar sua localização.
- A API de Geolocalização possui três métodos e algumas propriedades.
- O método primário na API de Geolocalização é getCurrentPosition, um método do objeto navigator.geolocation.
- getCurrentPosition possui um parâmetro necessário, o success handler, e dois opcionais, o error handler e as opções.
- Um objeto position é passado para o success handler com informação a respeito de sua localização, incluindo sua latitude e longitude.
- O objeto position contém uma propriedade coords, que é um objeto coordinates.
- O objeto coordinates tem propriedades: latitude, longitude e accuracy.
- Alguns dispositivos podem dar suporte às outras propriedades coordinates: altitude, altitudeAccuracy, heading (direção) e speed (velocidade).
- Utilize a propriedade accuracy (precisão) para determinar a precisão de sua localização em metros.
- Quando getCurrentPosition for chamado, seu browser deverá verificar se você deu permissão para compartilhar sua localização.
- watchPosition é um método do objeto geolocalização, que monitora sua localização e chama um success handler quando sua localização muda.
- Assim como getCurrentPosition, watchPosition possui um parâmetro necessário, um success handler e dois parâmetros opcionais, um error handler e opções.
- Use clearWatch para parar o monitoramento de sua localização.
- Quando watchPosition é usado, seu dispositivo passa a requerer mais energia; portanto, a vida útil de sua bateria deverá diminuir.
- O terceiro parâmetro, opções, para getCurrentPosition e watchPosition, é um objeto com propriedades que você ajusta para controlar o comportamento da API da Geolocalização.
- A propriedade maximumAge determina se o getCurrentPosition usará uma posição armazenada e, se assim o for, quão velha pode ser aquela posição antes de uma nova ser requisitada.
- A propriedade timeout determina quanto tempo o getCurrentPosition tem para obter uma nova posição antes do error handler ser chamado.
- A propriedade enableHighAccuracy dá uma sugestão aos dispositivos, para que eles se esforcem mais para obter uma localização o mais apurada possível.
- Você pode usar a API de Geolocalização com a API do Google Maps para identificar sua localização no mapa.



Palavras Cruzadas HTML5

Você viajou bastante neste capítulo com sua primeira API JavaScript. Faça-a combinar com esta palavra cruzada.

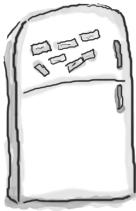


HORIZONTAIS

1. A precisão possui implicações em seu aplicativo, pois pode afetar a vida da _____.
_____.
5. Se disser não quando seu browser pedir-lhe para compartilhar sua localização, seu error handler será chamado com um _____ código de 1.
7. Centralize novamente seu mapa usando o método _____.
11. A latitude e a longitude do(a) _____ é 40.77, -73.98.

VERTICIAIS

2. Dispositivos抗igos sem GPS usam _____ por torres de celular para determinar sua localização.
3. A longitude é medida a partir de _____, Inglaterra.
4. "Onde quer que vá, lá está você" foi mencionado no filme _____.
6. A localização secreta do QG _____ é 47.62485, -122.52099.
8. Não dê direções a alguém se suas coordenadas não possuírem um(a) bom(boa) _____.
_____.
9. Você nunca vai obter uma localização armazenada se ajustar o(a) _____ para 0.
10. Você poderá usar a equação _____ para encontrar a distância entre duas coordenadas.



Ímãs de Geladeira

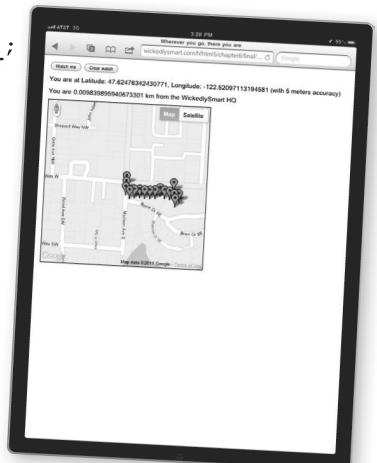
É seu trabalho terminar o código abaixo, no qual é indicado que um novo marcador só deve ser mostrado se viajarmos mais que 20 metros desde o último marcador adicionado. Utilize os ímãs de geladeira para completar o código. Cuidado, pois você terá de usar alguns mais de uma vez! Aqui está a solução.

```

var prevCoords = null;
function displayLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var div = document.getElementById("location");
    div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: " + longitude;
    div.innerHTML += " (with " + position.coords.accuracy + " meters accuracy)";
    var km = computeDistance(position.coords, ourCoords);
    var distance = document.getElementById("distance");
    distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";
    if (map == null) {
        showMap(position.coords);
        prevCoords = position.coords;
    }
    else {
        var meters = computeDistance(position.coords, prevCoords) * 1000;
        if (meters > 20) {
            scrollMapToPosition(position.coords);
            prevCoords = position.coords;
        }
    }
}

```

Muito melhor!



Tente online: <http://wickedlysmart.com/hfhtml5/chapterb/final/myLoc.html>



Aponte o seu lápis Solução

Abaixo você encontrará uma implementação alternativa para `displayLocation`. Você pode adivinhar o que ela faz? Dê uma olhada e escreva sua resposta abaixo. Se você estiver empolgado, experimente! Aqui está nossa solução.

```
distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";
if (km < 0.1) {
    distance.innerHTML = "You're on fire!";
} else {
    if (prevKm < km) {
        distance.innerHTML = "You're getting hotter!";
    } else {
        distance.innerHTML = "You're getting colder...";
    }
}
prevKm = km;
```

Escreva o que você
pensa que faz aqui.

Este código transforma nosso aplicativo num jogo de quente/frio. Ele mostra uma mensagem de “está ficando quente” se estiver se aproximando ao QG da Wickedly Smart, ou “está ficando frio” se estiver se afastando. Se estiver a menos de 0.1km do QG, então a mensagem será: “Está pegando fogo!”



Nós testamos, aqui está o resultado!

QUEM FAZ O QUÉ? SOLUÇÃO

Abaixo, você verá algumas opções para a API de Geolocalização.
Para cada opção, há uma combinação com seu comportamento.

`{maximumAge: 600000}`

Quero apenas posições armazenadas com menos de 10 minutos. Se não houver nenhuma posição com esse tempo, pedirei uma nova posição, mas apenas se conseguir uma em 1 segundo ou menos.

`{timeout:1000, maximumAge: 600000}`

Usarei uma posição armazenada se o browser encontrar uma que tenha menos de 10 minutos, ou então vou querer uma posição novinha em folha.

`{timeout:0, maximumAge:Infinity}`

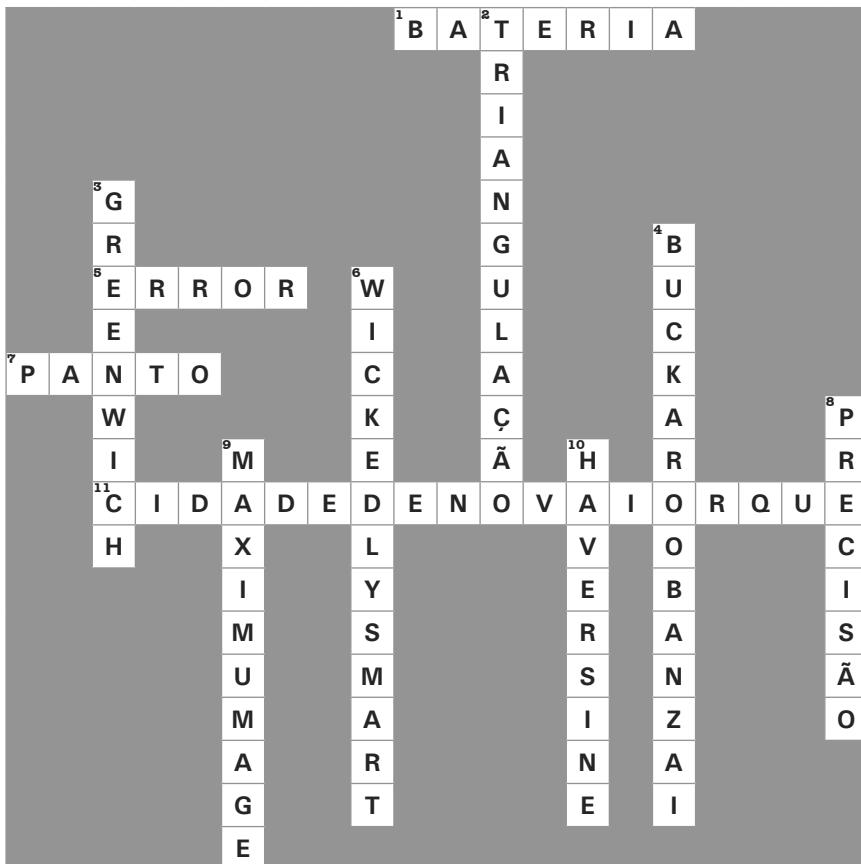
Quero apenas posições novas. O browser pode demorar o quanto for para arranjá-las para mim.

`{timeout:Infinity, maximumAge:0}`

Quero apenas posições armazenadas. Vou pegar uma de qualquer idade. Se não houver quaisquer posições, então chamarei o error handler. Sem novas posições para mim! Sou para uso offline.



Solução das Palavras Cruzadas HTML5



6 falando com a web

Aplicativos Extrovertidos



Você esteve na sua página por muito tempo. Está na hora de sair um pouquinho, conversar com outros serviços web, recolher informações e trazer isso tudo de volta para poder construir ótimas experiências, mesclando o que há de melhor nelas. Essa é uma parte muito importante, quando escrevemos aplicativos HTML5 modernos, mas, para isso, é preciso *saber como* conversar com os serviços web. Neste capítulo, vamos fazer exatamente isso e incorporar alguns dados, a partir de um serviço web real, bem na sua página. Depois de ter aprendido como fazer isso, você será capaz de se aproximar e tocar qualquer serviço web que quiser. Vamos inclusive apresentá-lo à nova e mais badalada linguagem utilizada para se falar com os serviços web. Então, venha. Você usará mais algumas APIs: as APIs de comunicação.

Mighty Gumball quer um aplicativo web

O caso é esse: Mighty Gumball, Inc., uma empresa inovadora que constrói e distribui máquinas de bala de chiclete de verdade, contatou-nos buscando ajuda. Se você não os conhece, eles recentemente ativaram na rede suas máquinas de chiclete para rastrear vendas quase em tempo real.

Agora, dá para perceber que a Mighty Gumball é uma expert em balas de chiclete, não desenvolvedora de software e, por isso, pediu nossa ajuda para construir um aplicativo que a ajudasse a monitorar suas vendas de balas de chiclete.

Eis o que eles nos enviaram:

Deem uma olhada na nova máquina de chiclete na rede MG2200. Isso irá revolucionar os negócios.

Talvez você se lembre deles de nosso livro Use a Cabeça! Padrões de Design, quando o ajudamos a desenhar o código no lado servidor deles.



Diretor executivo da Mighty Gumball

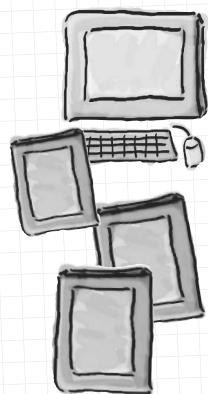


Mighty Gumball, Inc.

Onde a Máquina de Chiclete Nunca Está Meio Vazia

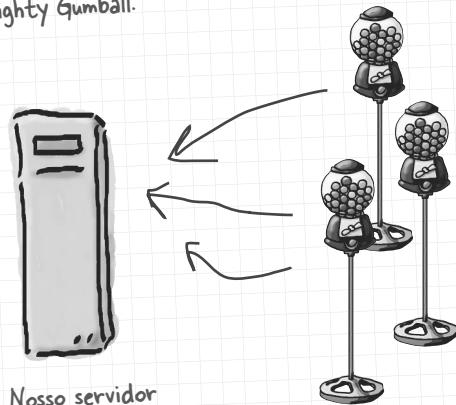
Obrigado por nos ajudar! Segue a maneira que achamos que a ferramenta de vendas em tempo real da máquina de chiclete deveria funcionar e esperamos que vocês possam implementá-la para nós! Diga-nos se tiverem quaisquer dúvidas!

Ah, enviaremos algumas especificações para o serviço web em breve.
- Engenheiros da Mighty Gumball.



Dispositivos móveis e desktops captam as vendas de um servidor em tempo real por meio de um serviço web.

Queremos que escrevam esta parte, usando HTML5, é claro!



Nossa servidor na web

Todas as nossas máquinas de chiclete reportando para o servidor central



Antes de começarmos, pense um pouco sobre como você poderia desenhar um aplicativo que resgata dados de um serviço web e mantém uma página atualizada baseada nos mesmos. Não se preocupe se você ainda não sabe recuperar os dados, apenas pense em algo de bom nível. Faça um desenho, ponha etiquetas, escreva pseudocódigos para qualquer código que precisar. Pense nisso como um aquecimento, só para preparar seu cérebro.



Mighty Gumball, Inc.

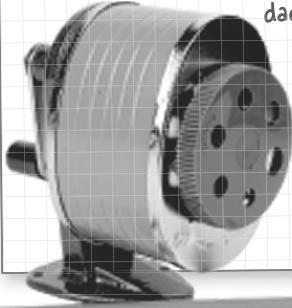
Onde a Máquina de Chiclete
Never Gets Empty

Notas dos Engenheiros

Como resgatamos os
dados do serviço web
para nossa página?

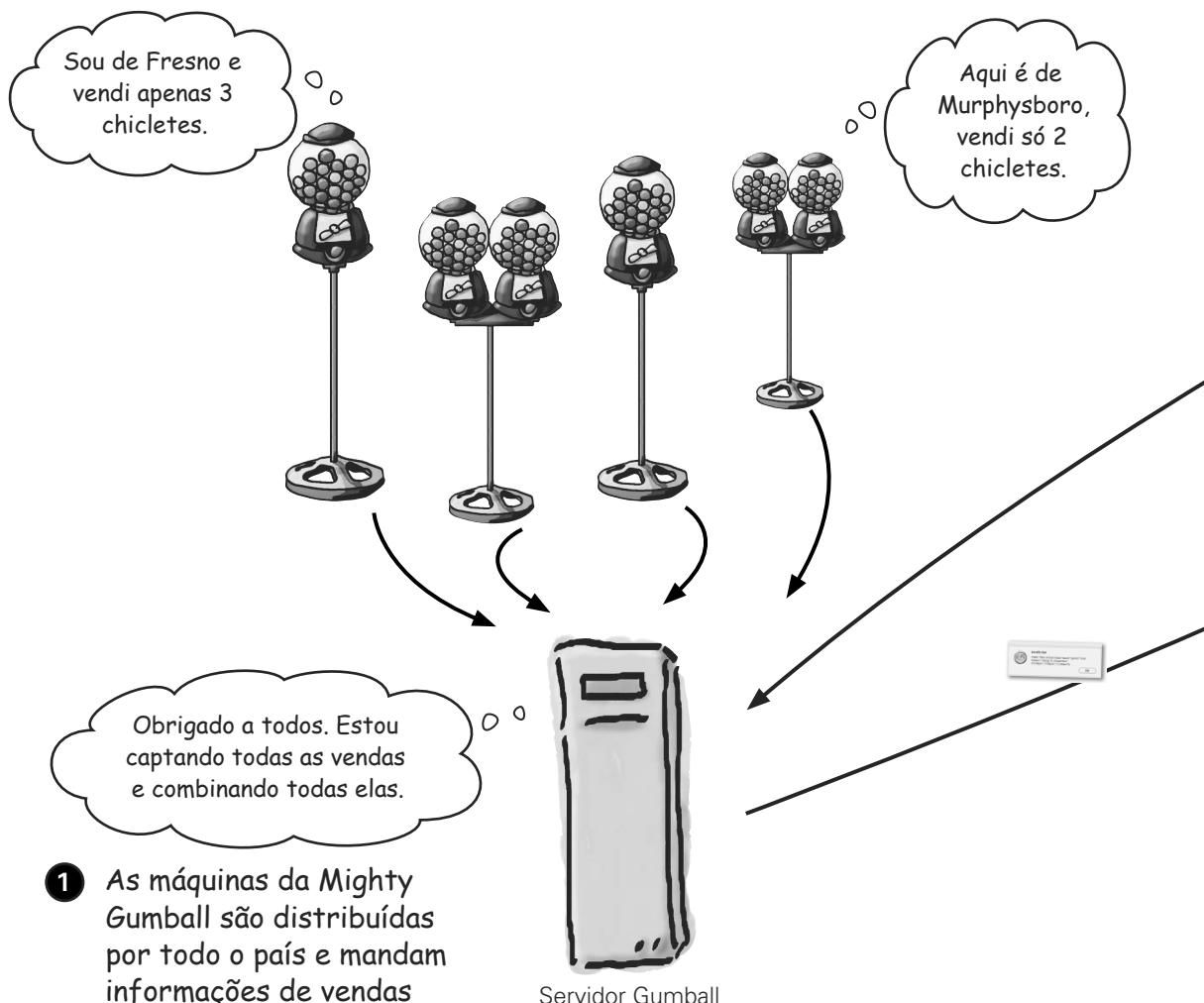
Uma vez que pegamos
os dados, como atualizamos
a página?

Que tipo de problemas
podemos ter ao resgatar
dados de um servidor remoto?



Um pouco mais de informação sobre a Mighty Gumball

Provavelmente, você precisa de um pouco mais de informação, além daquela pequena nota da Mighty Gumball. Eis o que nós temos: primeiro, eles têm máquinas de chiclete com pedidos de vendas por todo o país direcionados ao servidor da Mighty Gumball, que combina todos aqueles relatórios e torna-os disponíveis por intermédio de um serviço web. Segundo, eles estão nos pedindo para construir um aplicativo web que mostre as vendas num browser para a equipe de vendas da Gumball. É provável que eles queiram este relatório atualizado à medida que as vendas mudem com o passar do tempo. Eis a vista a 10.000 pés acima:



- 1 As máquinas da Mighty Gumball são distribuídas por todo o país e mandam informações de vendas para os servidores centrais da Gumball. O servidor agrupa todas juntas e as deixa disponíveis por meio de um serviço web.

- 2 O browser carrega o aplicativo web Mighty Gumball, incluindo a marcação HTML, o CSS e o JavaScript.



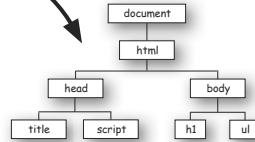
- 3 O aplicativo faz uma solicitação pela web para resgatar as vendas agregadas do servidor Gumball.



- 4 O aplicativo recebe dados de volta do servidor Gumball.



- 5 O aplicativo dá uma olhada nos dados e então atualiza o DOM da página para refletir quaisquer novos dados de vendas.



- 7 O aplicativo volta ao passo 3 e continuamente pergunta por novos dados. Como resultado, a página aparece para ser atualizada quase em tempo real.



- 6 O browser atualiza a página baseada no DOM e seus usuários veem os resultados.

Apenas um rápido início...

Enquanto estamos esperando por aquelas especificações da Mighty Gumball, vamos agilizar um pouco de HTML.

Você provavelmente está entendendo que não precisamos de um monte de marcação HTML para pôr para funcionar um aplicativo web. Tudo que precisamos é de um lugar para colocar nossos relatórios de vendas à medida que forem chegando e deixarmos o JavaScript fazer o resto. Vá em frente, comece a digitar e vamos dar uma olhada em como resgatar as coisas via web.

```
<!doctype html>
<html lang="en">
<head>
<title>Mighty Gumball (JSON)</title>
<meta charset="utf-8">
<script src="mightygumball.js"></script>
<link rel="stylesheet" href="mightygumball.css">
</head>
<body>
<h1>Mighty Gumball Sales</h1>
<div id="sales">
</div>
</body>
</html>
```

Apenas seu cabeçalho e corpo HTML5 padrão.

Fomos em frente e criamos um link a um arquivo JavaScript, sabendo que vamos escrever um pouco de JavaScript em breve!

E criamos nosso CSS para estilizar o relatório de vendas Mighty Gumball de forma que fique bonito para o CEO.

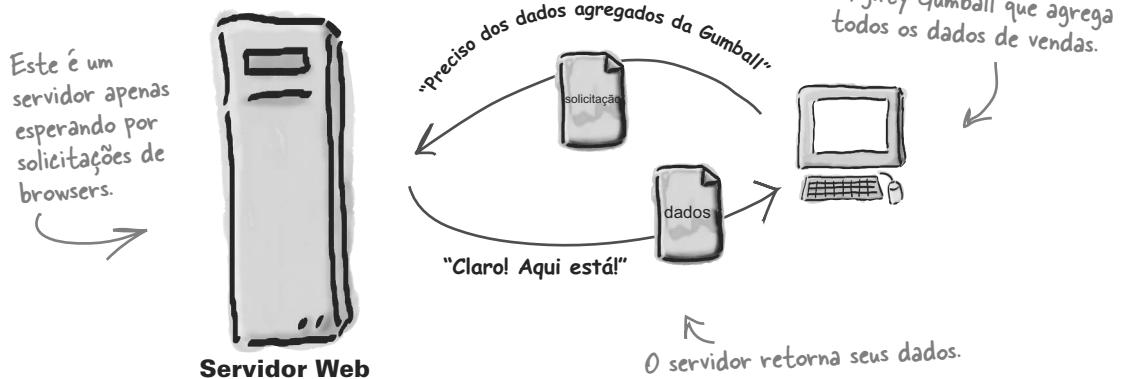
Aqui fica um placeholder onde vamos pôr os dados de vendas. Cada item de venda será adicionado como uma <div> aqui.

Desligue o motor...

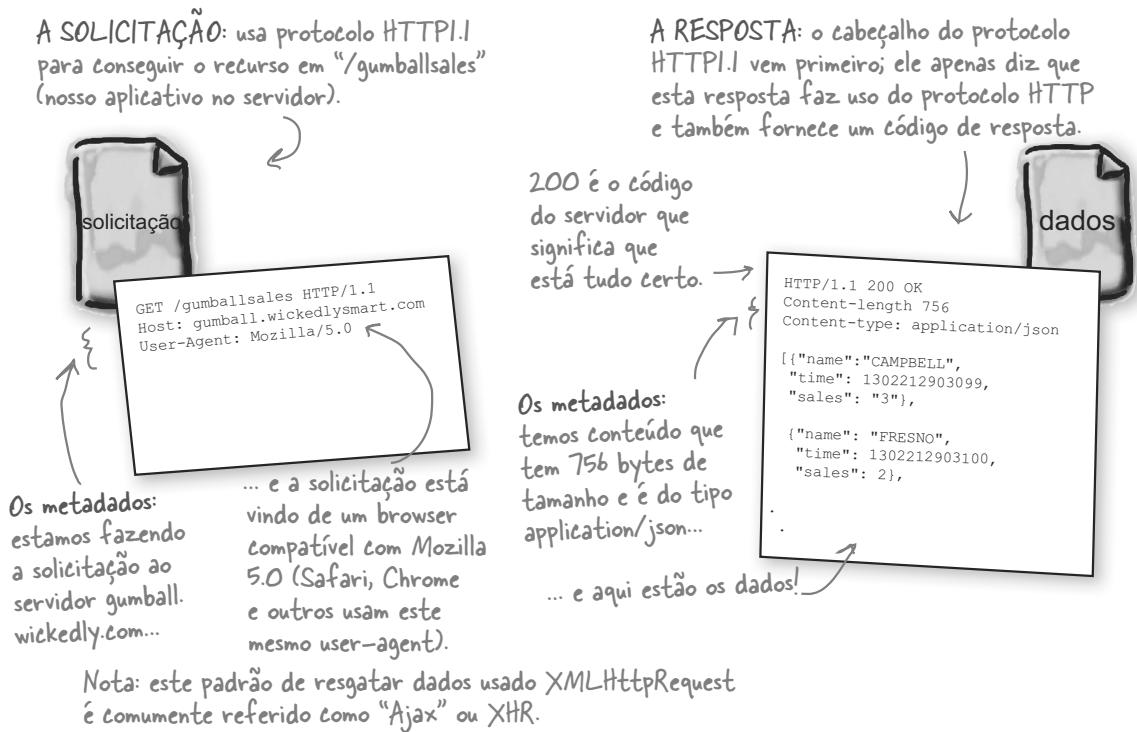
Vá e digite o código acima, carregue-o em seu browser favorito e experimente-o antes de continuar. Lembre-se: você pode fazer o download do CSS (e de outros códigos para este capítulo) em <http://wickedlysmart.com/hfhtml5>.

Então como fazemos requisições aos serviços web?

Vamos voltar um pouquinho... Você já sabe como um browser solicita uma página de um servidor web — ele faz uma solicitação HTTP para o servidor, o qual retorna a página junto com outros metadados que (normalmente) somente o browser vê. O que talvez você não saiba é que o browser pode também *resgatar dados* com HTTP a partir de um servidor web da mesma maneira. Veja como funciona:



Isso ajuda a olhar um pouco mais de perto para a solicitação que fazemos ao servidor e para a resposta que volta. A solicitação se preocupa em dizer ao servidor quais dados estamos procurando (aos quais às vezes nos referimos como “recurso” que estamos procurando), enquanto a resposta contém metadados e, se tudo der certo, os dados que solicitamos:



Como fazer uma requisição a partir do JavaScript

Tudo bem, já sabemos que é possível resgatar dados com o HTTP, mas como? Vamos escrever um pouco de código para criar uma verdadeira solicitação HTTP e então pedir ao browser para fazer a solicitação por nós. Depois de feita a solicitação, o browser então nos devolverá os dados recebidos. Vamos passo a passo fazer a solicitação HTTP:

- Para dar o pontapé inicial, vamos começar com uma **URL**.

Afinal de contas, precisamos dizer ao browser **onde** buscar os dados que estamos procurando:

Aqui está nossa URL em `someserver.com`.
`var url = "http://someserver.com/data.json";`

O ".json" significa um formato para troca de dados, mas já voltaremos a ele em instantes.

E vamos esconder a URL numa variável, `url`, que será útil em breve.

- Depois, vamos criar um **objeto request**, como este:

`var request = new XMLHttpRequest();`
 ↑
 Estamos designando o objeto request para a variável de solicitação.
 ↑
 E usamos o construtor XMLHttpRequest para criar um novo objeto de solicitação. Falaremos a respeito da parte "XML" em breve.



- Em seguida, precisaremos dizer ao objeto request qual a URL que queremos resgatar, junto com o tipo de solicitação que deveria usar (usaremos a solicitação padrão HTTP GET, como vimos na página anterior). Para fazer isso, utilizaremos o método `open` do objeto de solicitação. Agora, "open" parece com um método que não apenas define esses valores no objeto request, mas também abre a conexão e resgata os dados. Não é bem assim. Apesar do nome, `open` apenas cria a solicitação com uma URL e diz ao objeto request o tipo de solicitação a ser usada, de forma que o XMLHttpRequest possa verificar a conexão.

Segue como devemos chamar o método `open`:

`request.open("GET", url);`

Isto cria uma solicitação para nós, usando uma solicitação HTTP GET, que é o meio padrão de se resgatar dados HTTP.

E também cria a solicitação para usar a URL armazenada em nossa variável `url`.

O objeto XMLHttpRequest atualizado que sabe onde está indo



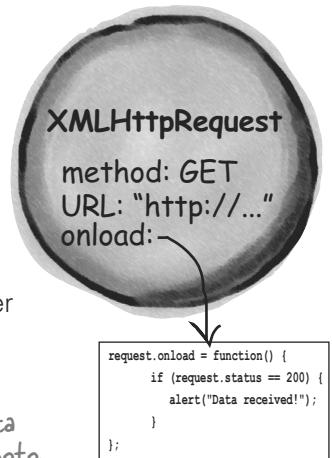
- 4 Ok, agora a parte importante e o truque de como o XMLHttpRequest funciona: quando finalmente pedirmos ao objeto XMLHttpRequest para resgatar dados, ele vai por conta própria atrás dos dados. Deve levar uns 90 milissegundos (um tempo e tanto, em termos de computador), ou, num dia lento, pode levar até dez segundos (uma eternidade nos mesmos termos). Portanto, em vez de ficar esperando pelos dados, vamos fornecer um handler que seja chamado quando os dados chegarem. Veja como criar o handler (isto deverá parecer bem familiar):

Nosso objeto request

```
request.onload = function() {
    if (request.status == 200) {
        alert("Data received!");
    }
};
```

Quando o browser recebe uma resposta do serviço web remoto, ele chama essa função.

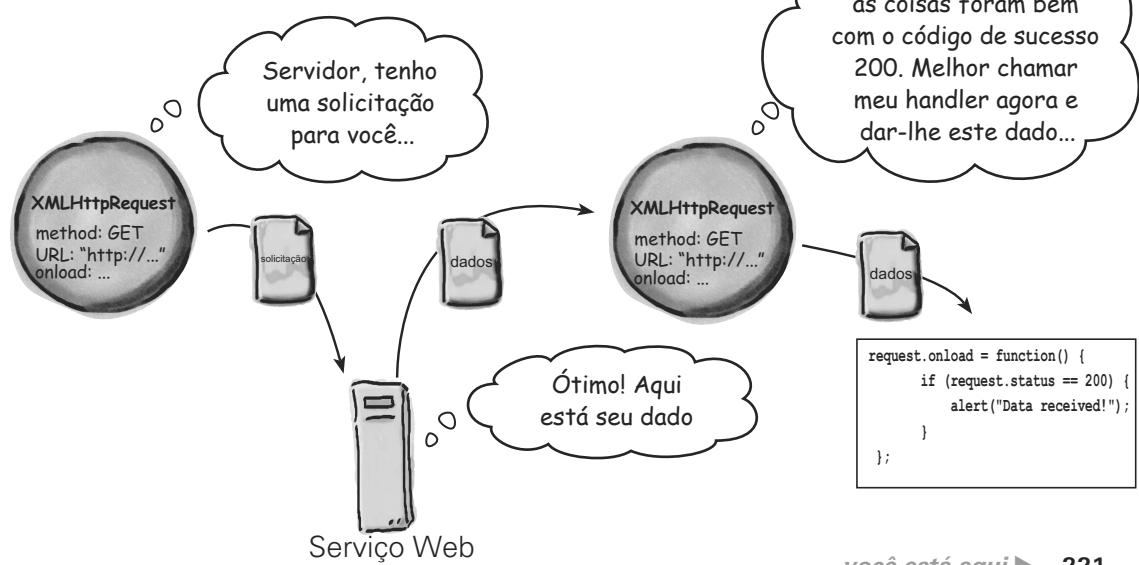
O handler precisa antes verificar se o código de retorno é 200 ou "OK", e então poderá fazer algo com os dados. Por ora, apenas vamos alertar o usuário de que os dados estão aqui. Vamos preencher tudo isso com mais códigos importantes em breve.

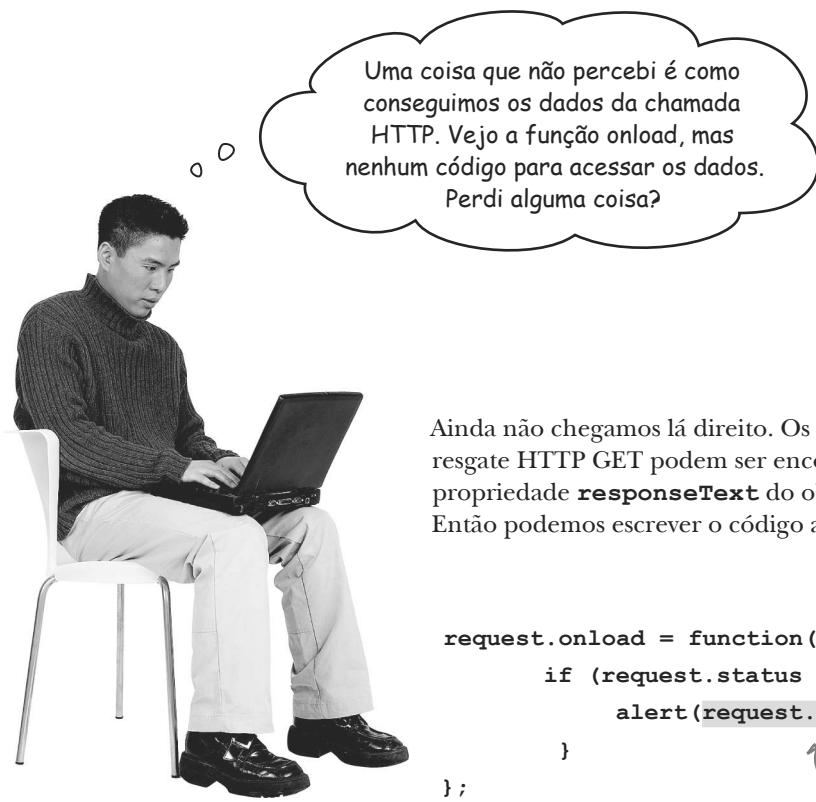


- 5 Apenas mais um último passo: ainda precisamos dizer à solicitação para ir buscar os dados e, para isso, usaremos o método `send`:

`request.send(null);` ↗ Isso envia a solicitação ao servidor. Mandamos null se não estivermos enviando nenhum dado ao serviço remoto (o que não estamos).

Então, para revisar: criamos um objeto XMLHttpRequest, carregamo-lo com uma URL e um tipo de solicitação HTTP, junto com um handler. Depois, enviamos a solicitação e esperamos que os dados cheguem. Quando isso acontece, o handler é chamado.





Uma coisa que não percebi é como conseguimos os dados da chamada HTTP. Vejo a função onload, mas nenhum código para acessar os dados.
Perdi alguma coisa?

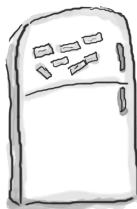
Ainda não chegamos lá direito. Os dados do resgate HTTP GET podem ser encontrados na propriedade **responseText** do objeto **request**. Então podemos escrever o código assim:

```
request.onload = function() {  
    if (request.status == 200) {  
        alert(request.responseText);  
    }  
};
```

Esta função é
chamada quando
a solicitação
recebeu uma
resposta.

↑ Podemos conseguir a
resposta da propriedade
responseText do objeto
request.

Espere um pouco, já estamos chegando ao ponto de escrever alguns códigos de verdade que usam **request.responseText**.



Ímãs de Geladeira

Um novo serviço web em <http://wickedlysmart.com/ifeelluckytoday> retorna tanto “unlucky” (sem sorte) quanto “lucky” (com sorte) toda vez que você o ativa. A lógica é baseada num antigo e secreto algoritmo que não podemos revelar, mas é um grande serviço para que os usuários saibam se estão ou não com sorte num determinado dia.

Precisamos de sua ajuda para criar uma implementação de referência para mostrar aos outros como poderão incluir isso em seus sites. Você encontrará o esqueleto do código abaixo; ajude-nos a preencher os detalhes usando os ímãs. Seja cuidadoso, talvez você não precise de todos os ímãs. Já fizemos um para você.

```
window.onload = function () {
    var url = "http://wickedlysmart.com/ifeelluckytoday";
    var request = _____
    _____ {
        if (_____) {
            displayLuck(_____);
        }
    };
    _____
}

function displayLuck(luck) {
    var p = document._____("luck");
    p._____ = "Today you are " + luck;
}
```

Seus ímãs vão aqui!

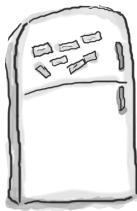
Sente-se sortudo
hoje? Quer ter
certeza? Use o
serviço!



```

new TextHttpRequest();           request.create("GET",
                                         url);
var i = 0;                     request.responseText
request.send(null);           request.open("GET", url);
request.onload = function()   new XMLHttpRequest();
myLuckyText                   getElementById
request.status == 200

```



Ímãs de Geladeira - Solução

Um novo serviço web em <http://wickedlysmart.com/ifeelluckytoday> retorna tanto “unlucky” (sem sorte) quanto “lucky” (com sorte) toda vez que você o ativa. A lógica é baseada num antigo e secreto algoritmo que não podemos revelar, mas é um grande serviço para que os usuários saibam se estão ou não com sorte num determinado dia.

Precisamos de sua ajuda para criar uma implementação de referência para mostrar aos outros como poderão incluir isso em seus sites. Você encontrará o esqueleto do código abaixo; ajude-nos a preencher os detalhes usando os ímãs. Seja cuidadoso, talvez você não precise de todos os ímãs. Segue nossa solução.

```

window.onload = function () {

    var url = "http://wickedlysmart.com/ifeelluckytoday";

    var request = new XMLHttpRequest();

    request.onload = function() {
        if (request.status == 200) {
            displayLuck(request.responseText);
        }
    };

    request.open("GET", url);
    request.send(null);

}

function displayLuck(luck) {
    var p = document.getElementById("luck");
    p.innerHTML = "Today you are " + luck;
}

var i = 0;
myLuckyText = new TextHttpRequest();
request.create("GET", url);
    
```

Sente-se sortudo hoje? Quer ter certeza? Use o serviço!

Seus ímãs vão aqui!

Imãs restantes



XMLHttpRequest Exposto

Entrevista da semana:

Confissões de um Objeto HTTP Request

Use a Cabeça!: Bem-vindo, XMLHttpRequest. Ficamos felizes que você tenha podido nos encaixar em sua agenda ocupada. Diga-nos onde você se encaixa na construção de aplicativos web.

XMLHttpRequest: Comecei essa empreitada ao trazer dados externos para dentro de sua página. Já ouviu falar de Google Maps? GMail? Tudo eu. De fato, nada disso seria possível sem mim.

Use a Cabeça!: Como assim?

XMLHttpRequest: Antes de eu aparecer, as pessoas construíam uma página no lado servidor e iam jogando tudo lá dentro dela à medida que criavam. Eu permito que você vá e consiga dados *depois* de a página estar construída. Pense no Google Maps: ele atualiza o que está na página toda vez que você ajusta sua localização no mapa, sem ter que recarregar a página toda.

Use a Cabeça!: Então, você é um cara bem-sucedido. Qual o seu segredo?

XMLHttpRequest: Sou humilde, ah, e simples. Dê-me uma URL e vou atrás dos dados para você. Nada mais do que isso.

Use a Cabeça!: É só isso?

XMLHttpRequest: Bem, você tem que me dizer o que fazer com o dado depois de eu tê-lo resgatado para você. Você pode me dar apenas uma função handler — um tipo de callback — e quando conseguir o dado, jogo ele em seu handler para que ele faça o que quiser com o dado.

Use a Cabeça!: Quais tipos de dados estamos falando aqui?

XMLHttpRequest: A web é cheia de dados hoje em dia; clima, mapas, dados sociais sobre pessoas e amigos, dados de geolocalização sobre o que está por perto... Sério, quase qualquer dado que você possa imaginar está trilhando seu caminho na web num formulário que funciona comigo.

Use a Cabeça!: E tudo isso se trata de XML, certo? Digo, seu nome é XML.

XMLHttpRequest: Sério? Você é um profissional e é por aí que você quer levar esta entrevista? Você fez seu trabalho de casa e tudo o que você tem a dizer é “tudo se resume em XML, certo?” Permita-me deixar algumas coisas claras por aqui. Claro, havia um tempo em que eu, principalmente, recuperava XML, mas o mundo está evoluindo. Hoje em dia, resgato todos os tipos de dados. Claro, algum XML de vez em quando, mas cada vez mais recebo solicitações para JSON.

Use a Cabeça!: Sério? O que é JSON e por que isto está se tornando tão popular?

XMLHttpRequest: JSON é JavaScript Object Notation e possui muitas vantagens — tamanho, leitura, o fato de que é nativo para a linguagem de programação mais popular na web: meu amigo JavaScript, é claro.

Use a Cabeça!: Mas não é aí que o formato não deveria ser tão importante assim para você? Os usuários deveriam poder solicitar XML ou JSON ou teletipo, pois você pode isso tudo. Não?

XMLHttpRequest: <silêncio>

Use a Cabeça!: Bem, parece que peguei num ponto crítico. Tudo bem, temos que ir para o intervalo... Então, XMLHttpRequest, acho que teremos um pouco mais de tempo com você, mais tarde, neste capítulo?

XMLHttpRequest: Sim, infelizmente, isto está em minha agenda...

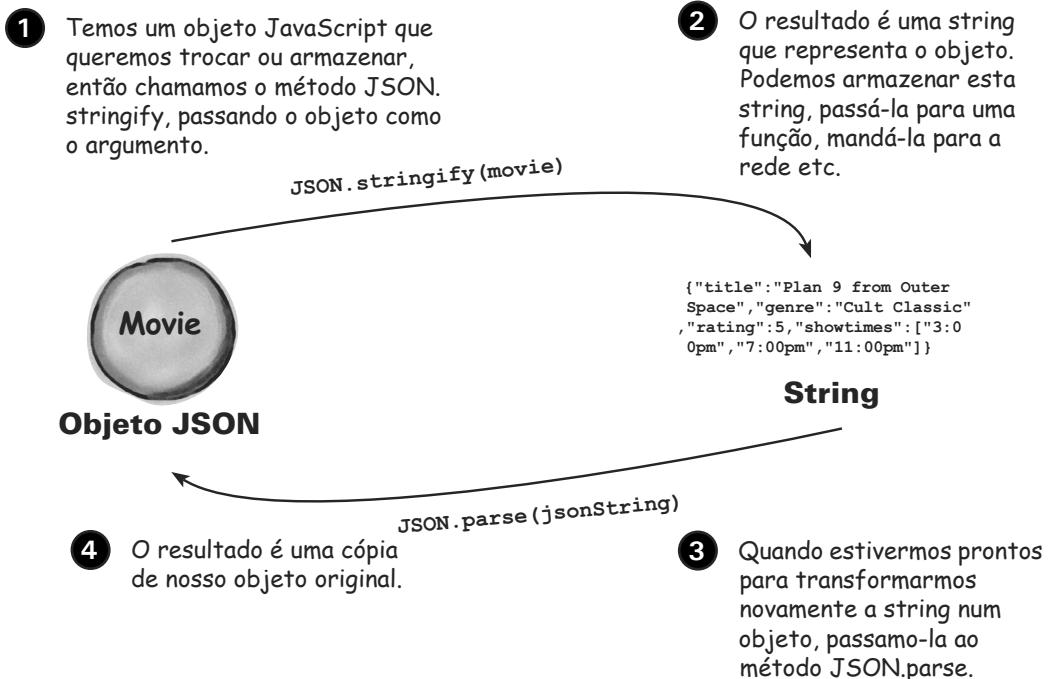
Deixando um pouco o XML, conhecendo JSON

Talvez você se lembre (talvez não) que o XML salvava a nós todos — um formato de dados que era legível para humanos e analisável por máquinas, um formato de dados que daria suporte a todos os dados necessários do mundo. Quando o XMLHttpRequest foi desenvolvido, o XML era, de fato, a maneira com a qual todos trocávamos dados (portanto, o nome XMLHttpRequest).

Bem, com o passar do tempo, XML, pelo visto, escorregou numa casca de banana jogada pelo JSON. Quem é JSON? Apenas o melhor e mais atual formato de dados, nascido do JavaScript, sendo adotado por toda a web no browser e no lado servidor. Tornou-se rapidamente o *formato preferido* para os aplicativos HTML5.

O que há de tão bom no JSON? Bem, é legível pra caramba para humanos, e pode ser analisado rápida e facilmente diretamente nos valores e objetos JavaScript. Contrariamente ao XML, ele é tão bonitinho... Pois é, dá para ver que gostamos dele só um pouquinho, né? Você verá um monte de JSON neste livro. Vamos usá-lo para trocar dados JavaScript pela rede, para armazenar dados num armazenamento local com a API Web Storage e como parte de outra maneira de acessar dados da web (em breve falaremos mais sobre isso).

Espere, porém, um segundo, um segundo. Formatos de troca de dados na rede... formatos de armazenamento... coisas complicadas, né? Não se preocupe, ~~nas próximas dez páginas vamos torná-lo um expert~~ você já sabe praticamente tudo o que precisa saber sobre JSON. Para usar JSON, você só precisa entender de objetos JavaScript (o que você sabe, e muito) e duas simples chamadas de método. Veja como tudo isso funciona:



Um rápido exemplo usando JSON

- ① Vamos percorrer um exemplo rápido de conversão de um objeto em seu formato de string JSON. Começaremos por um objeto que você já entende, o objeto Movie (Filme), do Capítulo 4. Nem tudo pode ser convertido numa string JSON — por exemplo, métodos — mas todos os tipos básicos, como números, strings e arrays possuem suporte. Vamos criar um objeto e, então, colocá-lo numa string:

```
var plan9Movie = new Movie("Plan 9 from Outer Space","Cult Classic", 2,
                           ["3:00pm", "7:00pm", "11:00pm"]);
```

Veja um belo objeto movie completo
com strings, números e um array.

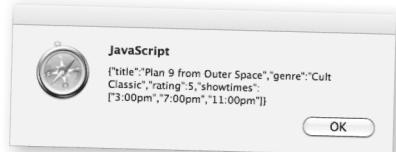
- ② Uma vez que já tenha um objeto, você pode convertê-lo num formato de string JSON com o método `JSON.stringify`. Vejamos como isso funciona... (fique à vontade para experimentar, abrindo seu código movie do Capítulo 4 lá atrás e adicionando o seguinte código no fim de seu script):

```
var jsonString = JSON.stringify(plan9Movie);
alert(jsonString);
```

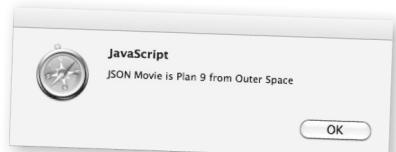
- ③ Agora, temos uma string JSON que representa nosso objeto movie. Neste momento, poderíamos pegar essa string e fazer qualquer coisa com ela, como enviar por HTTP para um servidor. Também poderíamos receber uma string JSON de outro servidor. Digamos que um servidor nos deu esta string; como fariam para torná-la novamente um objeto com o qual poderíamos fazer algo? É só usar o método irmão `JSON.stringify`: `JSON.parse`. Assim:

```
var jsonMovieObject = JSON.parse(jsonString);
alert("JSON Movie is " + jsonMovieObject.title);
```

Existem, na verdade,
algumas outras
restrições, mas não
nos preocuparemos
com elas agora.



↑
Este é o resultado: uma
versão string do objeto
mostrado no alerta.



↑
Ah, e agora usamos isso
como um objeto real que
acessa suas propriedades.

PODER DO CÉREBRO

Experimente esta URL. O que você vê?

<http://search.twitter.com/search.json?q=hfhtml5>

Nota: o Firefox pedirá para abrir ou salvar
um arquivo. Você pode abrir com TextEdit,
Notepad ou qualquer editor de texto básico.

Ei! As
especificações
já chegaram!
Vire a folha!



As especificações
acabaram de chegar!



Mighty Gumball, Inc.
Onde a Máquina de Chiclete
Nunca Está Meio Vazia

Especificações do Servidor Gumball

Obrigado por assumirem esta tarefa!!!

Temos todas as vendas das máquinas agregadas e sendo servidas
pelo servidor central em:
<http://gumball.wickedlysmart.com/>

Escolhemos JSON para ser nosso formato de dados. Se vocês acessarem a URL acima,
receberão um array dos objetos JSON, que se parecem com isso:

```
[ { "name": "CaMPBELL", ←  
  "time": 1302212903099, ←  
  "sales": 3 }, ←  
  { "name": "FRESNO", ←  
  "time": 1302212903100, ←  
  "sales": 2 }, ←  
  . . . ←  
 ] ← E mais cidades ficarão aqui...
```

O nome da cidade; estamos apenas testando Califórnia agora.

O tempo em milissegundos quando este relatório chegou.

Número de chicletes vendidos desde o último relatório.

Uma segunda cidade, Fresno.

Vão em frente e digitem esta URL em seu browser para ver os valores retornando.

→ Você deverão ver um ou mais destes objetos num array.

Apenas especifiquem
um tempo em milissegundos.

Você também podem adicionar um parâmetro `lastreporttime` no fim da URL;
desta forma terão apenas os relatórios desde aquele tempo. Usem-no assim:

<http://gumball.wickedlysmart.com/?lastreporttime=1302212903099>

Certifique-se de fazer isso!

Temos centenas de máquinas de chiclete reportando agora mesmo, de fato você deverá ver relatórios a cada cerca de 5 a 8 segundos, em média. Dito isso, este é o nosso servidor de produção, portanto teste seu código localmente, antes de tudo!

Obrigado novamente pela ajuda de vocês! E lembre-se: "a máquina de chicletes nunca está meio vazia", como diz nosso CEO.

- Engenheiros da Mighty Gumball

Vamos ao trabalho!

Já temos nossas especificações da Mighty Gumball e você já fez seu treinamento em XMLHttpRequest e JSON. Já deve estar preparado para escrever um pouco de código. O objetivo é ter uma ideia de como ficará o Gumball App.

Lembre-se de que já temos um pouco de HTML estruturado para trabalhar, e que ele possui um link para um arquivo chamado `mightygumball.js`. É daí que vamos começar a escrever nosso código agora.

Lembre-se também de que já deixamos um lugar vazio no HTML onde vamos pôr os dados das vendas de chicletes, bem na `<div>` que rotulamos com uma id “vendas” (sales). Então, vamos colocar tudo junto e escrever um pouco de código.

```
<!doctype html>
<html lang="en">
  <head>
    <title>Mighty Gumball (JSON)</title>
    <meta charset="utf-8">
    <script src="mightygumball.js"></script>
    <link rel="stylesheet" href="mightygumball.css">
  </head>
  <body>
    <h1>Mighty Gumball Sales</h1>
    <div id="sales">
    </div>
  </body>
</html>
```

Escrevendo uma função de handler onload

Temos certeza de que tudo isso já está manjado pra você, mas vamos escrever um handler onload que é invocado quando o HTML é carregado por completo; também vamos em frente disparar uma solicitação HTTP para obter os dados de vendas. Quando os dados voltarem, pediremos ao XMLHttpRequest para chamar a função `updateSales` (a qual escreveremos num instante):

```
window.onload = function() {
  var url = "http://localhost/sales.json";
  var request = new XMLHttpRequest();
  request.open("GET", url);
  request.onload = function() {
    if (request.status == 200) {
      updateSales(request.responseText);
    }
  };
  request.send(null); Finalmente, enviamos a solicitação.
}
```

Testaremos num arquivo local primeiro (como os engenheiros da Mighty Gumball sugeriram!) para nos certificarmos de que tudo está funcionando. Falaremos mais a respeito num segundo...

Desenvolvemos o XMLHttpRequest ao criar o objeto, chamando o método aberto com nossa URL e então ajustando a propriedade `onload` a uma função.

Checkamos para certificar de que tudo está certo e, então...
... quando o dado terminar de carregar, esta função é chamada.



Veja bem!

Se estiver usando Opera ou IE 8 ou anterior, recomendamos que teste com outro browser. Falaremos sobre como dar suporte a eles mais tarde.

Mostrando os dados das vendas de chicletes

Agora, precisamos escrever o handler updateSales. Vamos facilitar e usar a implementação mais simples possível, pois sempre poderemos melhorar depois:

```
function updateSales(responseText) {  
  var salesDiv = document.getElementById("sales");  
  salesDiv.innerHTML = responseText;  
}
```

Já vamos pegar a <div>, pô-la no HTML e usá-la como um lugar para os dados.

Configure o conteúdo da div de acordo com toda aquela porção de dados. Já, já vamos analisar isso... Primeiro vamos testar.

Cuidado, desvio adiante!



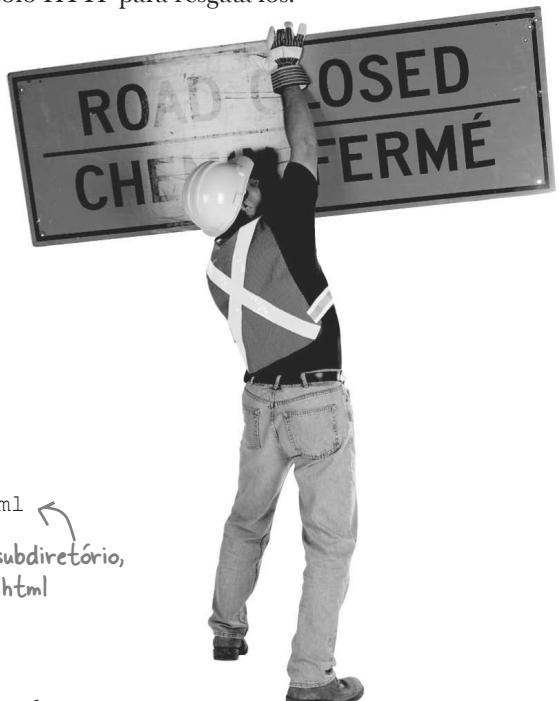
Está na hora de outro test drive, mas temos um pequeno desvio pela frente. Os engenheiros da Mighty Gumball nos pediram para testar localmente antes de acessar seu servidor de produção, o que é uma boa ideia. Para isso, porém, precisamos dos dados num servidor para que o XMLHttpRequest possa usar o protocolo HTTP para resgatá-los.

Em termos de servidores, você tem algumas opções:

- Se sua empresa possui servidores que estão disponíveis para teste, use-os.
- Ou você pode usar um serviço de hospedagem terceirizado, como GoDaddy, Dreamhost, ou um dos muitos outros serviços similares.
- Finalmente, você pode configurar um servidor direto de sua própria máquina. Neste caso, suas URLs vão parecer mais ou menos assim:

`http://localhost/mightygumball.html`

Os arquivos também podem ser dispostos num subdiretório, como `http://localhost/gumball/mightygumball.html`



Dê uma olhada na próxima página para dicas e direcionamentos. Tenha em mente que ambientes de hospedagem diferem bastante. Então, não dá para escrevermos um guia geral para eles. Portanto, que a força esteja com você, e, se você não tem fácil acesso a um servidor, configurar um em sua máquina local pode ser sua melhor escolha!

Como configurar seu próprio Servidor Web

A maneira com a qual você vai configurar uma hospedagem local realmente depende de que tipo de sistema operacional está usando. Verifique as dicas abaixo para OS X (ou mais conhecido como Mac), PC e Linux. Você encontrará outras opções na próxima página.



Eu sou um Mac



Configurar um servidor no Mac é fácil. Vá até > System Preferences, então escolha Sharing. No painel da esquerda, certifique-se de que Web Sharing está marcado:



Quando tiver ativado Web Sharing (ou se já estiver ativo), você verá alguma informação sobre como acessar seu servidor local. Você deverá ser capaz de usar o localhost em vez do endereço IP (que tende a mudar se estiver usando um roteador DHCP, assim o localhost funcionará melhor para você). Por padrão, seus arquivos são servidos do `http://localhost/~YOUR_USERNAME/`, que serve arquivos de sua pasta `YOUR_USERNAME/Sites/`, então você provavelmente vai querer configurar uma subpasta lá para Mighty Gumball.

Eu sou um PC



Instalar seu próprio servidor no Windows é mais fácil do que costumava ser, graças ao Microsoft Web Platform Installer (também conhecido por Web PI). A versão atual está disponível para Windows 7, Windows Vista SP2, Windows XP SP3+, Windows Server 2003 SP2+, Windows Server 2008 e Windows Server 2008 R2, e você pode baixá-la aqui: <http://microsoft.com/web/downloads/platform.aspx>.

Outra opção é instalar a fonte aberta WampServer, que vem com o Apache, PHP e MySQL para desenvolvimento de aplicativos web. É fácil de instalar e gerenciar.

Você pode fazer o download em:
<http://www.wampserver.com/en/>.

Há mais uma porção de soluções de fonte aberta por aí se você procurar, portanto, você tem muitas opções.

Sou um Nerd completo uma Distribuição Linux

Venhamos e convenhamos, você já sabe o que está fazendo. Certo? O Apache normalmente já está instalado por padrão. Então verifique sua documentação de distribuição.

Como configurar seu próprio Servidor Web, continuação

Ah, você *realmente* quer hospedar suas páginas? Excelente! Não há nada melhor do que ter suas páginas hospedadas na web de verdade. Cheque as dicas abaixo e divirta-se!

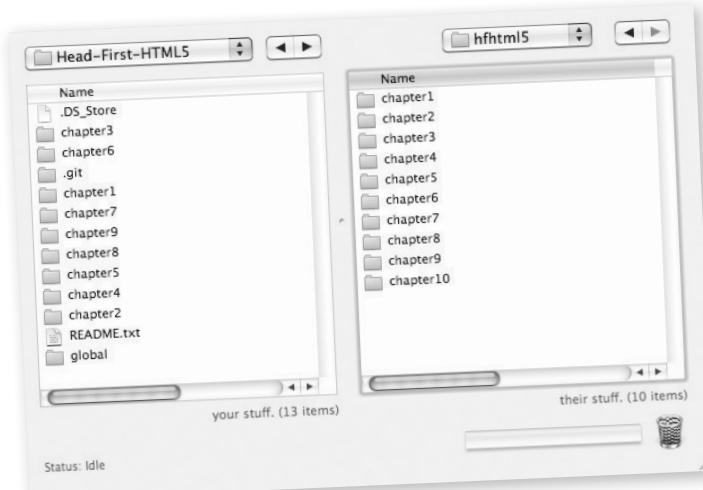


Desvio

Hospedagem terceirizada...

Se não quiser configurar seu próprio servidor, você sempre poderá usar um servidor remoto, mas precisará hospedar seu HTML, o JavaScript e o CSS, assim como o arquivo JSON, todos no mesmo servidor (falaremos depois sobre o porquê disto ser crucial) para continuar com este exemplo.

A maioria dos serviços de hospedagem irá lhe dar acesso FTP a uma pasta onde poderá pôr todos esses arquivos. Se tiver acesso a um servidor como esse, faça o upload de todos os arquivos e substitua seu nome de servidor onde quer que veja localhost nas páginas seguintes.



Você pode usar um programa FTP como o Transmit, o Cyberduck ou o WinSCP para fazer o upload de seus arquivos se não quiser usar linhas de comando FTP.

Fizemos uma lista de provedores de hospedagem, caso precise de uma recomendação, mas eles são fáceis de achar; procure apenas por “hospedagem web” e encontrará vários para escolher. Nossa lista está em <http://wickedlysmart.com/hfhtml5/hosting/hosting.html>. Conte para a gente se você puser um site HTML5 online; nós adoraríamos ver!

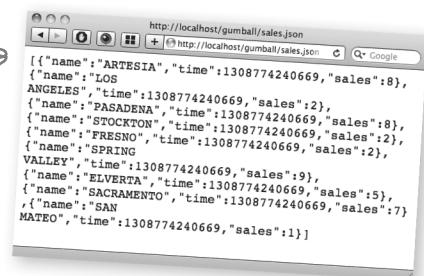
De volta ao código

Neste ponto, esperamos que você já tenha seu próprio servidor em pleno funcionamento — isso poderia ser um servidor rodando em sua máquina local (o que estamos fazendo) ou um servidor em algum outro lugar a que você tenha acesso. Em qualquer um dos casos, você irá pôr seu HTML e seus arquivos JavaScript no servidor e então apontar seu browser ao arquivo HTML. Você também vai precisar do arquivo de teste dos dados de vendas da Mighty Gumball lá também, então lhe daremos um simples arquivo de dados para pôr em seu servidor. Para seu aplicativo, parecerá exatamente como se estivesse sendo gerado pelo servidor da Mighty Gumball em tempo quase real. Isso permitirá que você teste seu código sem acessar o servidor da Mighty Gumball. Eis como o arquivo se parece: é chamado de `sales.json` e está incluído no código para o livro (ou você pode digitar, se preferir esse tipo de coisa):

```
[{"name": "ARTESIA", "time": 1308774240669, "sales": 8},
 {"name": "LOS ANGELES", "time": 1308774240669, "sales": 2},
 {"name": "PASADENA", "time": 1308774240669, "sales": 8},
 {"name": "STOCKTON", "time": 1308774240669, "sales": 2},
 {"name": "FRESNO", "time": 1308774240669, "sales": 2},
 {"name": "SPRING VALLEY", "time": 1308774240669, "sales": 9},
 {"name": "ELVERTA", "time": 1308774240669, "sales": 5},
 {"name": "SACRAMENTO", "time": 1308774240669, "sales": 7},
 {"name": "SAN MATEO", "time": 1308774240669, "sales": 1}]
```

Vá em frente e ponha este arquivo em seu servidor. Certifique-se de atualizar seu JavaScript para a URL desse arquivo. A nossa é `http://localhost/gumball/sales.json`:

Ajuda se você testar primeiro esta URL no seu browser para ter certeza que funciona.



```
window.onload = function() {
    var url = "http://localhost/gumball/sales.json";
    var request = new XMLHttpRequest();
    request.open("GET", url);
    request.onload = function() {
        if (request.status == 200) {
            updateSales(request.responseText);
        }
    };
    request.send(null);
}
```

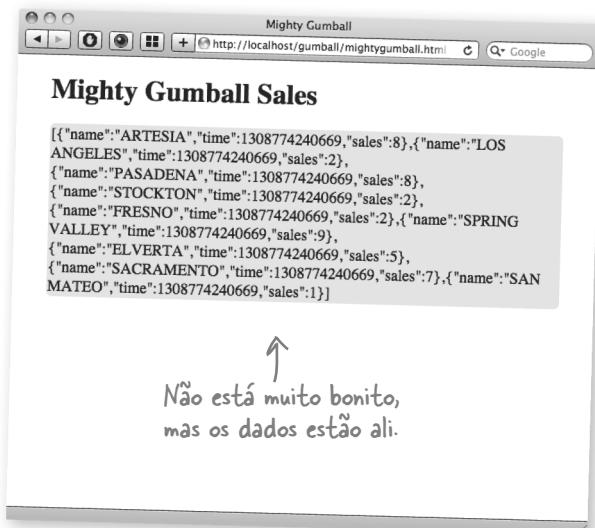
Certifique-se de que esteja apontando para a URL correta.

Já vamos testar!



Tem sido uma jornada e tanto, mas finalmente estamos prontos para testar este código!

Apenas se certifique de ter os arquivos HTML, JavaScript, JSON — e não se esqueça de seu CSS — no servidor. Vá em frente e entre na URL de seu arquivo HTML dentro de seu browser (a nossa é <http://localhost/gumball/mightygumball.html>), aperte Enter...



```
[{"name":"ARTESIA","time":1308774240669,"sales":8}, {"name":"LOS ANGELES","time":1308774240669,"sales":2}, {"name":"PASADENA","time":1308774240669,"sales":8}, {"name":"STOCKTON","time":1308774240669,"sales":2}, {"name":"FRESNO","time":1308774240669,"sales":2}, {"name":"SPRING VALLEY","time":1308774240669,"sales":9}, {"name":"ELVERTA","time":1308774240669,"sales":5}, {"name":"SACRAMENTO","time":1308774240669,"sales":7}, {"name":"SAN MATEO","time":1308774240669,"sales":1}]
```

Lembre-se de que estamos enviando uma solicitação HTTP para conseguir os dados em sales.json, os quais estamos apenas jogando para dentro da <div> por ora. Parece que funcionou!

Se estiver tendo problemas, verifique cada arquivo independentemente por meio de seu browser e certifique-se de que eles sejam acessíveis. Então, clique duas vezes em suas URLs.



Impressionando seu cliente...

Fizemos um esforço danado para colocar este aplicativo em funcionamento, e está ótimo, mas a Mighty Gumball ficará muito mais impressionada se ficar bonito também. É isso que vamos fazer agora...

O que temos



No momento, estamos apenas jogando um array JSON para dentro do browser. É, de certa forma, efetivo, mas feio. Que desperdício! Há toda uma estrutura de dados só esperando para ser usada com mais efetividade!

O que queremos



Aqui usamos o array JSON e criamos um bom display a partir dele. Esses são os últimos 10% que podem fazer a diferença entre amadores e profissionais, você não acha?

Eis o que precisamos fazer para melhorar nosso display:

- ① Primeiro, precisamos pegar de volta o dado que temos de nosso objeto XMLHttpRequest (que é apenas uma string JSON) e convertê-lo num verdadeiro objeto JavaScript.
- ② Então, poderemos dar um jeito no array resultante e adicionar novos elementos ao DOM, um para cada item sales do array.

Retrabalhando nosso código para fazer uso do JSON

Vamos seguir esses dois passos para pôr em forma os códigos:

- ① Primeiro, precisamos pegar o dado que temos do objeto XMLHttpRequest (que não é uma string JSON) e convertê-lo num verdadeiro objeto JavaScript.

Para isso, vamos atualizar a função `updateSales`, primeiro deletando a linha que liga o conteúdo `<div>` à string `responseText`, e convertendo o `responseText` de uma string a seu equivalente JavaScript usando `JSON.parse`.

```
function updateSales(responseText) {  
    var salesDiv = document.getElementById("sales");  
    salesDiv.innerHTML = responseText; ↗  
    var sales = JSON.parse(responseText); ↗  
}  
↑  
Pegue a resposta e use JSON.parse para  
convertê-la num objeto JavaScript (neste caso,  
será um array), e designe-a à variável sales.  
Não precisamos  
mais desta linha.
```

- ② Agora, vamos dar um jeito no array resultante e adicionar novos elementos ao DOM, um para cada item `sales` no array. Neste caso, vamos criar uma nova `<div>` para cada item:

```
function updateSales(responseText) {  
    var salesDiv = document.getElementById("sales");  
    var sales = JSON.parse(responseText);  
    for (var i = 0; i < sales.length; i++) { ↗ Alterar em cada item no array.  
        var sale = sales[i];  
        var div = document.createElement("div"); ↗ Para cada item, criar uma  
        div.setAttribute("class", "saleItem"); ↗ <div> e dar-lhe a classe  
        div.innerHTML = sale.name + " sold " + sale.sales + " gumballs"; ↗ "saleItem" (usada pelo CSS).  
        salesDiv.appendChild(div); ↗ Ajuste os conteúdos <div> com  
    } ↗ innerHTML e então acrescente-o  
} como um filho da <div> sales.
```

A reta final...

Você já sabe como isso aqui vai se parecer, mas vá em frente e faça essas mudanças. Dê mais uma olhada cuidadosa no código na página anterior e certifique-se de que tudo está certinho. Então, vá e carregue novamente aquela página.

Viu, dissemos que se pareceria com isto!



O teste correu bem. Você está pronto para usar os servidores de produção em tempo real da Mighty Gumball. Boa sorte!



Ajay, o cara do Controle de Qualidade.

Seguindo para o Servidor em Tempo Real

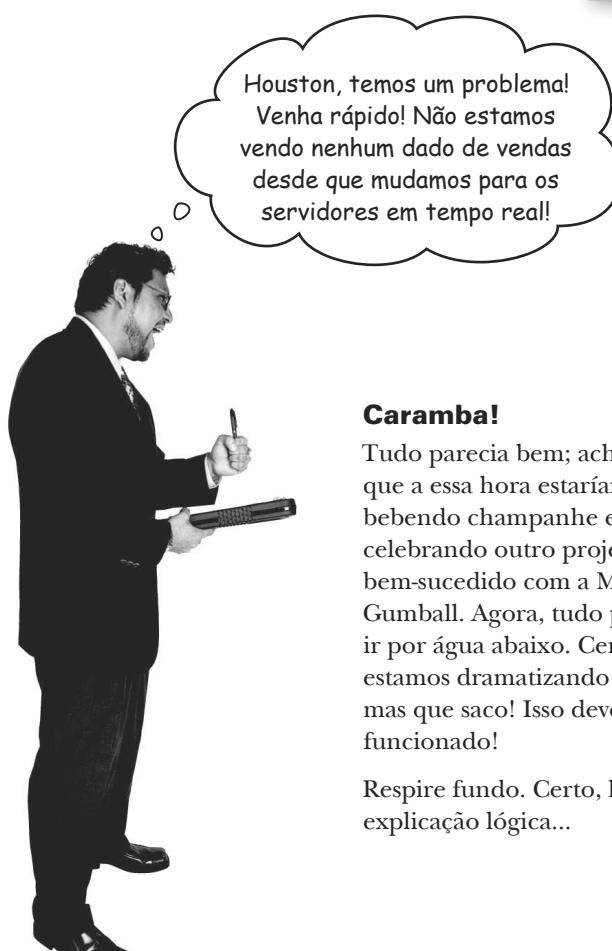
A Mighty Gumball nos pediu para testar localmente e assim o fizemos. Agora, estamos prontos para seguir para os testes com o servidor real. Desta vez, em vez de resgatar um arquivo de dados JSON estático, vamos resgatar o JSON que é gerado dinamicamente dos servidores Mighty Gumball. Precisamos atualizar a URL que aquele XMLHttpRequest está usando e mudá-la para apontar para a Mighty Gumball. Vamos fazer isso:

```
window.onload = function() {
    var url = "http://gumball.wickedlysmart.com";
    var request = new XMLHttpRequest();
    request.open("GET", url);
    request.onload = function() {
        if (request.status == 200) {
            updateSales(request.responseText);
        }
    };
    request.send(null);
}
```

Aqui está o servidor URL. Mude isso e certifique-se de que esteja salvo.

Test Drive em Tempo Real...

Certifique-se de que sua mudança na URL esteja salva em seu arquivo `mightygumball.js` no seu servidor, se quiser continuar resgatando seu HTML de lá, ou para seu HD local, se estiver usando localhost. De lá, você sabe o que fazer: aponte seu browser para seu arquivo HTML e veja os belos e verdadeiros dados, em tempo real, de todas aquelas pessoas ao redor do mundo comprando Mighty Gumballs!



Houston, temos um problema!
Venha rápido! Não estamos
vendo nenhum dado de vendas
desde que mudamos para os
servidores em tempo real!

O quê?! Não estamos
vendo nenhum dado!

Caramba!

Tudo parecia bem; achávamos que a essa hora estaríamos bebendo champanhe e celebrando outro projeto bem-sucedido com a Mighty Gumball. Agora, tudo poderá ir por água abaixo. Certo, estamos dramatizando demais, mas que saco! Isso deveria ter funcionado!

Respire fundo. Certo, há uma explicação lógica...

Nota ao Editor: nós, na verdade, pensávamos que estaríamos recebendo um cheque gordo de adiantamento para enviar este livro! Agora, temos de dar um jeito nessa bela confusão!

Ajay, o cara do Controle de Qualidade bem irritado.

Que suspense!

Não estamos vendendo nenhum dado em nossa página. Estava tudo funcionando perfeitamente até que passamos para o servidor em tempo real...

Será que **acharemos** o problema?

Vamos **consertá-lo**?

Fique ligado... Responderemos a essas questões e mais...

Enquanto isso, veja se consegue bolar algumas ideias sobre o que deu errado e como poderemos resolver.



PONTOS DE BALA

- Para conseguir arquivos ou dados HTML de um servidor, o browser envia uma solicitação HTTP.
- Uma resposta HTTP inclui um código de resposta que indica se houve um erro com a solicitação.
- O código resposta HTTP 200 significa que a solicitação não teve erros.
- Para enviar uma solicitação HTTP do JavaScript, use o objeto XMLHttpRequest.
- O handler onload do objeto XMLHttpRequest lida com a obtenção da resposta do servidor.
- A resposta JSON para um XMLHttpRequest é colocada na propriedade responseText da solicitação.
- Para converter a string responseText para JSON, use o método JSON.parse.
- O XMLHttpRequest é usado em aplicativos para atualizar conteúdo, tais como mapas e e-mail, sem solicitar um novo carregamento da página.
- O XMLHttpRequest pode ser usado para resgatar qualquer tipo de conteúdo de texto, como XML, JSON e mais.
- O XMLHttpRequest Level 2 é a mais recente versão de XMLHttpRequest, mas o padrão ainda está em desenvolvimento.
- Para usar XMLHttpRequest, você deve servir arquivos e solicitar dados de um servidor. Você pode criar um servidor local em sua própria máquina para testar ou usar uma solução de hospedagem.
- A propriedade onload XMLHttpRequest não é suportada por browsers mais antigos, como o IE8 e o Opera 10 ou anteriores. Você pode escrever código para checar a versão do browser e fornecer uma alternativa para browsers mais antigos.



XMLHttpRequest Exposto Parte 2

Entrevista da semana:

Internet Explorer e “Você disse JSON?”

Use a Cabeça!: Bem-vindo à segunda parte da entrevista, XMLHttpRequest. Queria lhe perguntar sobre suporte a navegadores — você está disponível apenas nos browsers mais novos?

XMLHttpRequest: Os caras não me chamam de “velhinho” por nada; venho tendo suporte de browsers desde 2004. Nos anos da internet, sou um idoso.

Use a Cabeça!: Bem, e quanto ao desuso, você se preocupa com isso?

XMLHttpRequest: Sou alguém que se reinventa, mais ou menos, a cada década. No momento, estamos todos trabalhando na segunda versão do XMLHttpRequest, conhecida como Level 2 (nível 2). De fato, os browsers mais modernos já têm suporte para o Level 2.

Use a Cabeça!: Impressionante. O que há de diferente no Level 2?

XMLHttpRequest: Bem, primeiramente, suporte para mais tipos de eventos. Pode-se fazer coisas como rastrear o progresso de uma solicitação e escrever códigos mais elegantes (na minha opinião).

Use a Cabeça!: Falando em suporte a browser...

XMLHttpRequest: Certo, lá vem...

Use a Cabeça!: Disseram-nos as más línguas que você e o IE não se dão muito bem...

XMLHttpRequest: Aí está. Se você quer a resposta para isso, tudo o que tem de fazer é ler todas as entrevistas que já dei. Aparentemente, você perdeu todas elas. Você está de brincadeira comigo? Toda essa história de XMLHttpRequest começou com o IE.

Use a Cabeça!: Sim, mas e o ActiveXObject e o XDomainRequest? Já ouviu falar desses nomes?

XMLHttpRequest: Esses são meus apelidos! É como eles me chamam lá na Microsoft! Certo, concordo que seja chato nós termos diferentes nomes para mim, mas todos eles fazem a mesma coisa. É facilmente resolvido com um pouco mais de código e em termos dos recentes browsers da Microsoft, versão 9 e posteriores, todas vão bem. Se isso é novidade para seus leitores, ficarei feliz em ficar após a entrevista para me certificar de que o código deles funcione em versões prévias do IE.

Use a Cabeça!: Isso é muito gentil! Garanto que encontraremos um lugar para isso neste capítulo. —————

XMLHttpRequest: Ei, eu sou um cara legal, nunca deixaria seus leitores na mão.

Use a Cabeça!: Acreditamos em você. Outra pergunta: você mencionou o JSON e que você é um grande fã dele. Você se preocupa, de alguma forma, com o JSONP?

XMLHttpRequest: O quê? Eu? Preocupar?

Use a Cabeça!: As pessoas têm dito por aí que ele está substituindo você.

XMLHttpRequest: Certo, claro, com o JSONP você pode resgatar dados, mas é apenas um jeito inteligente de invadir. Digo, pense no código truncado que precisa ser escrito, e quanto à segurança?

Use a Cabeça!: Ei, não estou sendo técnico. Tudo que sei é que as pessoas dizem que ele ajuda a resolver problemas que você não pode. De qualquer forma, só temos tempo para isso.

XMLHttpRequest: É, pelo menos você tem a parte “nem tão técnica” correta.



A propriedade `onload` de XMLHttpRequest não tem suporte para versões mais antigas de browsers, mas existe uma alternativa fácil.

Temos usado `request.onload` para definir uma função que é chamada quando a solicitação termina de pegar dados do servidor. Isso é uma característica de XMLHttpRequest Level 2 (pense nisso como uma "versão 2"). XMLHttpRequest Level 2 é bem novo, portanto, muitos usuários podem ainda estar usando browsers que não dão suporte. Particularmente, IE 8 (e anteriores) e Opera 10 (e anteriores) suportam apenas XMLHttpRequest Level 1. As boas notícias são que as novidades do XMLHttpRequest Level 2 são melhorias e, por isso, você pode continuar a usar apenas as características da versão 1 em todos os browsers sem quaisquer problemas; isso só significa que seu código não é tão elegante. Segue abaixo o código para usar o XMLHttpRequest Level 1:

```
function init() {
    var url = "http://localhost/gumball/sales.json";
    var request = new XMLHttpRequest();
    request.onreadystatechange = function() {
        if (request.readyState == 4 && request.status == 200) {
            updateSales(request.responseText);
        }
    };
    request.open("GET", url);
    request.send(null);
}

A maioria do código para
se usar XMLHttpRequest
Level 1 é o mesmo...
... mas não há
propriedade request.
onload no Level
2, portanto, você
precisará usar
a propriedade
onreadystatechange
no lugar.

Então, cheque o
readyState para se
certificar de que os
dados terminaram
de carregar. Se o
readyState for
4, saberá que está
completo.

Você também
poderá procurar
por outros
valores de status
e readyState, se
quiser verificar
vários erros.

Todo o resto é
praticamente
a mesma coisa.
```

Lembra que deixamos você com um suspense? Um bug.

Tínhamos o código funcionando perfeitamente usando nosso servidor local, mas, assim que o deslocamos para o servidor em tempo real na web, falhou!

O que esperávamos:

Mighty Gumball

Mighty Gumball Sales

```
ARTESIA sold 8 gumballs
LOS ANGELES sold 2 gumballs
PASADENA sold 8 gumballs
STOCKTON sold 2 gumballs
FRESNO sold 2 gumballs
SPRING VALLEY sold 9 gumballs
ELVERTA sold 5 gumballs
SACRAMENTO sold 7 gumballs
SAN MATEO sold 1 gumballs
```

Aqui se encontra como nossa página aparece, quando rodamos o código, usando nosso servidor local para servir os dados de vendas a partir de `http://localhost/gumball/sales.json`.

O que temos:

Mighty Gumball

Mighty Gumball Sales

The page is completely blank.

E aqui nós vemos como nossa página aparenta, quando rodamos o código, usando o servidor Mighty Gumball em tempo real para servir os dados de vendas a partir de `http://gumball.wickedlysmart.com`.

Então, o que fazemos agora?!

Por quê? Vamos fazer o que sempre fazemos: reunir pessoas para uma rápida conversa. Temos certeza de que juntos, todos nós (incluindo alguns personagens fictícios), podemos resolver isso! Frank? Jim? Joe? Onde estão vocês? Ah, aí estão! Na próxima página...



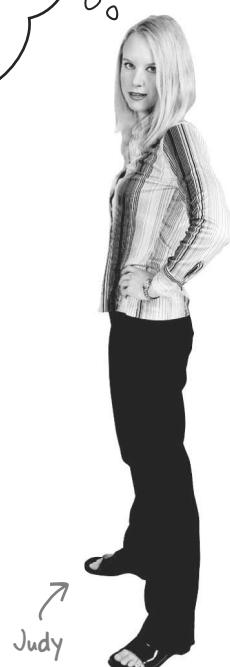
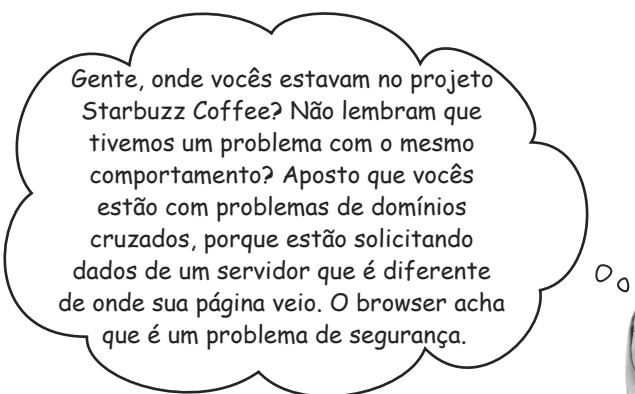


Jim: Você tem a URL correta?

Frank: Sim, e digitei-a dentro do browser para ter certeza de que veria os dados das vendas que esperávamos e funcionou perfeitamente. Eu não entendo...

Joe: Eu dei uma olhada no console JavaScript do Chrome e vi algo sobre controle de acesso e origens ou domínios.

Frank: Errrrr?



O que é Política de Segurança do Browser?

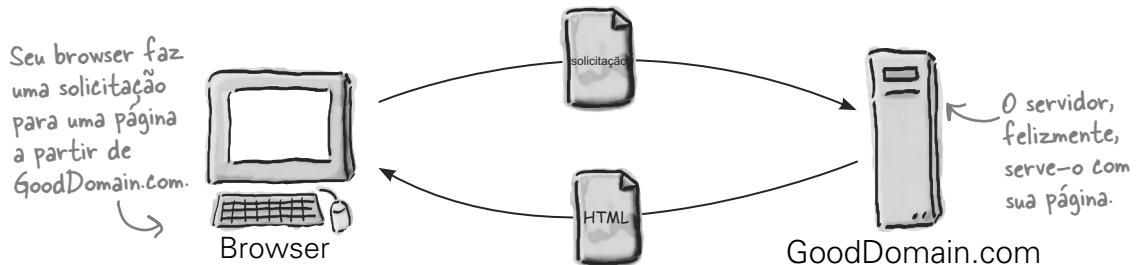
Tudo bem, é embracoso encontrar obstáculos — só de pensar na posição em que estamos pondo vocês leitores — mas a Judy está certa. O browser aplica certa segurança em suas solicitações HTTP XMLHttpRequest e isso pode causar alguns problemas.

Então, o que é essa política? Bem, é uma política de browser e ela diz que você não pode resgatar dados de um domínio que seja diferente do em que a própria página era servida. Digamos que você esteja rodando o site para DaddyWarBuckBank.com e alguém invada seus sistemas e insira um pouco de JavaScript, que acessa informação pessoal do usuário, fazendo todo o tipo de coisas interessantes com ela ao se comunicar com o servidor HackersNeedMoreMoney.com. Parece ruim, certo? Bem, para parar com esse tipo de coisa, o browser evita que você faça solicitações XMLHttpRequest para domínios diferentes do original, do qual a página pertencia.

Vamos dar uma olhada no que está certo e no que não está:

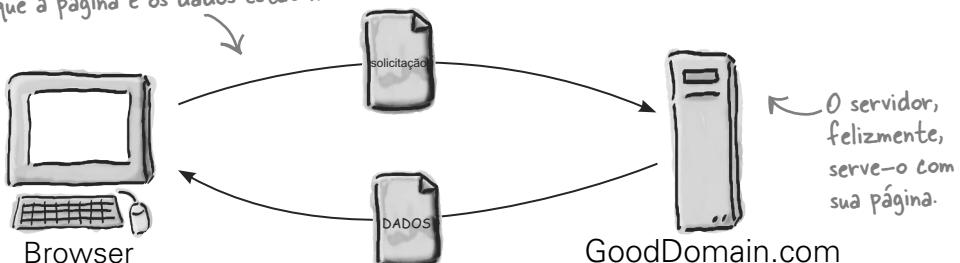
Comportamento aceitável para código JavaScript:

- 1 Primeiro o usuário (por meio do browser) faz uma solicitação para uma página HTML (e, claro, qualquer associado JavaScript e CSS):



- 2 A página precisa de alguns dados de GoodDomain.com, fazendo assim uma solicitação de dados para XMLHttpRequest:

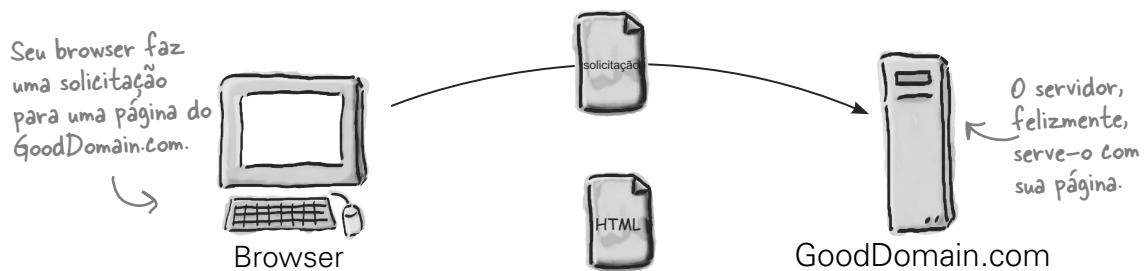
Esta solicitação para conseguir dados de GoodDomain.com é bem-sucedida porque a página e os dados estão no mesmo domínio.



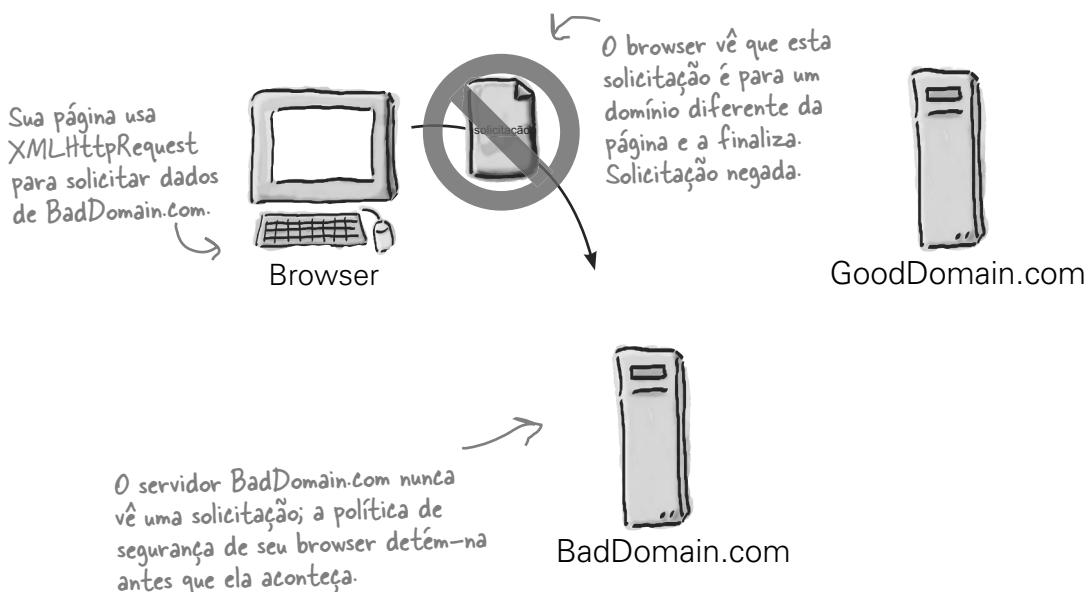
Comportamento inaceitável para código JavaScript:

Agora, vejamos o que acontece quando sua página hospedada em GoodDomain.com tenta fazer uma solicitação de dados usando XMLHttpRequest ao BadDomain.com, no lugar.

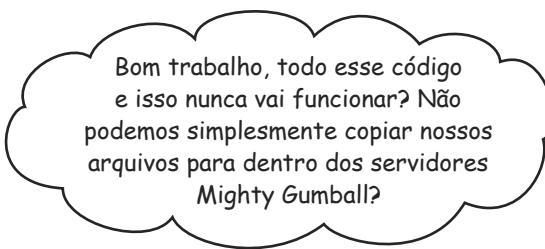
- 1 Assim como antes, o browser faz uma solicitação a uma página em GoodDomain.com. Isso pode incluir arquivos CSS e JavaScript que também estão hospedados em GoodDomain.com.



- 2 Agora, temos código que quer dados de outra fonte, isto é, BadDomain.com. Vejamos o que acontece quando a página solicita aqueles dados usando XMLHttpRequest:



Pelo menos não no orçamento
que o editor nos deu!



Normalmente, a resposta é sim.

Digamos que você fosse um desenvolvedor trabalhando no código para a Mighty Gumball. Você habitualmente teria acesso a seus servidores (ou a pessoas que poderiam distribuir arquivos aos servidores por você). Poderia colocar todos os seus arquivos lá e evitar quaisquer problemas de domínios cruzados. Neste caso, porém (e nós detestamos acabar com suas esperanças), você *não* está, efetivamente, trabalhando para a Mighty Gumball. Você é leitor deste livro e não conseguimos pensar numa maneira de ter centenas de milhares de pessoas (todos os leitores) copiando seus arquivos para dentro dos servidores da Mighty Gumball.

Então, onde isso nos deixa? Chegamos a um beco sem saída? Não, ainda temos algumas opções. Vamos até elas...



Então, quais são nossas opções?

Temos de ser sinceros com você. Todo esse tempo nós já sabíamos que o cruzamento de origem da solicitação XMLHttpRequest falharia. Como acabamos de dizer, quando se está construindo aplicativos, deve-se possuir acesso ao servidor, mas, isso não será um problema (e se estiver construindo aplicativos na dependência de seus próprios dados, usar o XMLHttpRequest será, normalmente, a melhor maneira de fazê-lo).

Neste ponto, podemos ouvi-lo dizer “isso é ótimo, mas como podemos pôr esse código para funcionar?”. Bem, temos algumas formas para fazer isso acontecer:

① **Plano 1: usar nossos arquivos hospedados.**

Já pusemos arquivos em nosso servidor para você, deixando-os em:

<http://gumball.wickedlysmart.com/gumball/gumball.html>

Vá até lá e tente apontar seu browser para esta URL. Você será capaz de ver o mesmo código que digitou até agora, em ação e funcionando.

② **Plano 2: usar outra maneira para conseguir os dados.**

O XMLHttpRequest é uma maneira ótima de se inserir dados em seus aplicativos, quando eles estiverem hospedados no mesmo domínio, mas e se você precisar realmente pegar dados de um terceiro? Digamos que você precise de dados do Google ou Twitter, por exemplo. Nestes casos, precisaremos realmente pôr um fim ao problema e encontrar outra abordagem.

Acontece que existe outra forma, baseada no JSON, conhecida como JSONP (se estiver curioso, é a abreviatura, em inglês, de “JSON com revestimento”; concordamos que parece estranho, mas nós enfrentaremos isso num segundo). Pegue seu *jetpack*, porque isso parece vir “de outro planeta”, se é que você nos entende.



Joe: É mesmo! Mas, o que é isso?

Jim: Parece que é outra forma de conseguir dados de serviços web para dentro de nossos aplicativos.

Frank: Sou inútil aqui. Sou só o cara criativo.

Jim: Frank, não acho que isso seja ruim. Eu rapidamente pesquisei no Google a respeito do JSONP e, basicamente, é uma forma de forçar a tag `<script>` a fazer o trabalho de resgatar os dados.

Joe: Mmm, e isso é legal?

Jim: Completamente — muitos serviços grandes estão dando suporte a isso, como o Twitter.

Frank: Parece uma invasão.

Joe: Bem, sim, é onde eu estava chegando. Quero dizer, como ficar usando uma tag `<script>` pode ser uma maneira correta de se conseguir dados? Nem sei como isso funcionaria.

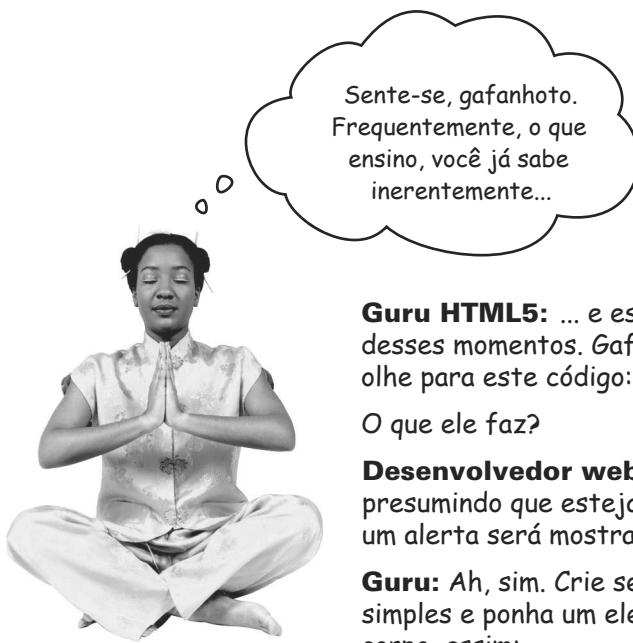
Jim: Estou começando a entender. Mas pense nisso assim: quando você usa um elemento `<script>`, ele está resgatando código para você, certo?

Joe: Certo...

Jim: Bem, e se puser dados naquele código?

Joe: Ok, estou chegando lá...

Frank: Sim, a passos de tartaruga...



Guru HTML5: ... e este é um desses momentos. Gafanhoto, olhe para este código:

```
alert("woof");
```

O que ele faz?

Desenvolvedor web: Quando você avalia, presumindo que esteja rodando num browser, um alerta será mostrado dizendo "woof".

Guru: Ah, sim. Crie seu próprio arquivo HTML simples e ponha um elemento <script> nele, no corpo, assim:

Este código está localizado nesta URL.

```
<script src="http://wickedlysmart.com/hfhtml5/chapter6/dog.js">
</script>
```

Guru: O que isso faz?

Desenvolvedor web: Carrega a página, que carrega o JavaScript a partir de dog.js que está em wickedlysmart.com, o qual chama a função alerta e eu vejo um alerta com "woof" no display, por intermédio do browser.

Guru: Então um arquivo JavaScript, servido por outro domínio, pode chamar uma função dentro do seu browser?

Desenvolvedor web: Bem, pensando assim, sim Guru, acho que é isso que está acontecendo. O arquivo dog.js do wickedlysmart.com, uma vez resgatado, chama o alerta em meu browser.

Guru: Você encontrará outro arquivo em: <http://wickedlysmart.com/hfhtml5/chapter5/dog2.js> com o JavaScript:

```
animalSays("dog", "woof");
```

Guru: O que isso faz?

Desenvolvedor web: É parecido com dog.js, mas chama uma função animalSays. Tem também dois argumentos, não apenas um: o tipo de animal e o som do animal.

Guru: Escreva a função animalSays e acrescente-a num elemento <script> no cabeçalho de seu arquivo HTML, acima do elemento <script> que aponta para wickedlysmart.

Desenvolvedor web: Como é isso?

```
function animalSays(type, sound) {  
    alert(type + " says " + sound);  
}
```

Guru: Muito bem, você está entendendo bem. Agora, mude sua outra referência <script>, aquela que aponta para dog.js, para apontar para dog2.js e carregue novamente a página em seu browser.

Desenvolvedor web: Eu obtenho um alerta que diz "dog says woof".

Guru: Dê uma olhada em <http://wickedlysmart.com/hfhtml6/chapter5/cat2.js>, mude sua referência <script> para apontar para cat2.js e tente isso.

```
animalSays("cat", "meow");
```

Desenvolvedor web: Obtenho um alerta que diz "cat says meow".

Guru: Portanto, não apenas o arquivo JavaScript que foi servido por outro domínio pode chamar qualquer função que quiser em seu código, mas também pode passar-nos qualquer dado que quiser?

Desenvolvedor web: Não vejo qualquer dado, na verdade, apenas dois argumentos.

Guru: E argumentos não são dados? E se mudarmos os argumentos para parecerem assim:

```
var animal = {"type": "cat", "sound": "meow"};  
animalSays(animal);
```

← cat3.js

Desenvolvedor web: Agora a função animalSays está passando um argumento que, pelo visto, é um objeto. Mmm, posso ver claramente como aquele objeto começa a se parecer com dado.

Guru: Você pode reescrever animalSays de forma que ele use o novo objeto?

Desenvolvedor web: Vou tentar...

Desenvolvedor web: Que tal isso?

```
function animalSays(animal) {
    alert(animal.type + " says " + animal.sound);
}
```

Guru: Muito bem. Mude sua referência para <http://wickedlysmart.com/hfhtml5/chapter6/dog3.js> e experimente-o. Tente também <http://wickedlysmart.com/hfhtml5/chapter6/cat3.js>.

Desenvolvedor web: Sim, ambos funcionam como esperado com minha nova função.

Guru: E se você mudar o nome de `animalSays` para `updateSales`?

Desenvolvedor web: Guru, não vejo como animais estão relacionados às vendas da Gumball...

Guru: Funcionam comigo aqui. E se renomearmos `dog3.js` para `sales.js` e reescrevermos isso assim:

```
var sales = [{"name": "ARTESIA", "time": 1308774240669, "sales": 8},
             {"name": "LOS ANGELES", "time": 1308774240669, "sales": 2}];
updateSales(sales);
```

Desenvolvedor web: Acho que estou começando a entender. Estamos passando dados por intermédio do arquivo JavaScript a que fazemos referência, em vez de usar XMLHttpRequest para resgatá-los sozinhos.

Guru: Sim, Gafanhoto. Não se apegue, porém, aos detalhes quando se pode ver o todo. Não estamos também pegando isso de outro domínio? Algo que é proibido pelo XMLHttpRequest.

Desenvolvedor web: Sim, parece que sim. Isso se parece mais com mágica.

Guru: Não há mágica. O elemento `<script>` sempre se comportou assim. A resposta sempre esteve dentro de você. Agora, por favor, vá meditar sobre como isso funciona para que ele seja aceito.

Desenvolvedor web: Sim, mestre. "Fazer com que ele seja aceito"... Sei que essa frase parece tão familiar, mas não consigo comprehendê-la.



MOMENTO ZEN

Usar JavaScript para resgatar dados é algo com o qual você precisa se fundir. Pegue uma folha de papel ou use a capa interna deste livro. Desenhe um servidor que hospeda seus arquivos HTML e JavaScript. Faça também um desenho de um servidor em outro domínio que possua os arquivos `dog3.js` e `cat3.js`. Agora percorra os passos que o browser utiliza para conseguir e usar o objeto em cada arquivo. Quando achar que conseguiu, vamos fazer isso tudo novamente juntos.

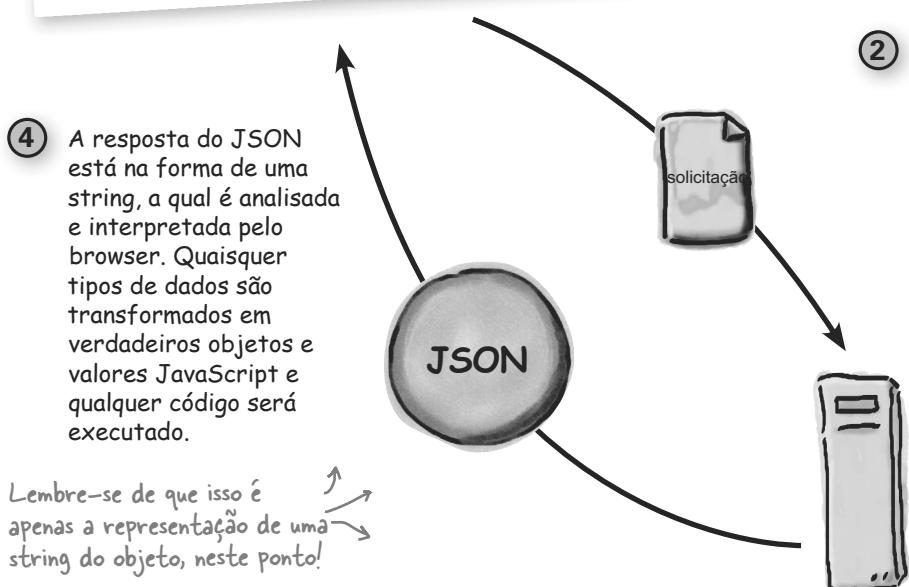
Conheça o JSONP

Você provavelmente descobriu que o JSONP é uma maneira de resgatar os objetos JSON ao usar a tag `<script>`. É também uma maneira de resgatar dados (novamente, sob a forma de objetos JSON), que evita os problemas de segurança de mesma origem que vimos com o XMLHttpRequest.

Vamos desvendar como o JSONP funciona nas próximas páginas:

```
!doctype html
<html lang="en">
...
<body>
  <h1>Mighty Gumball Sales</h1>
  <div id="sales">
    </div>
    <script src="http://gumball.wickedlysmart.com/"></script>
</body>
</html>
```

- 1 Em nosso HTML, nós incluímos um elemento `<script>`. A fonte para esse script é, na verdade, a URL de um serviço web que vai nos suprir com JSON para nossos dados, como nossos dados de vendas da Mighty Gumball.

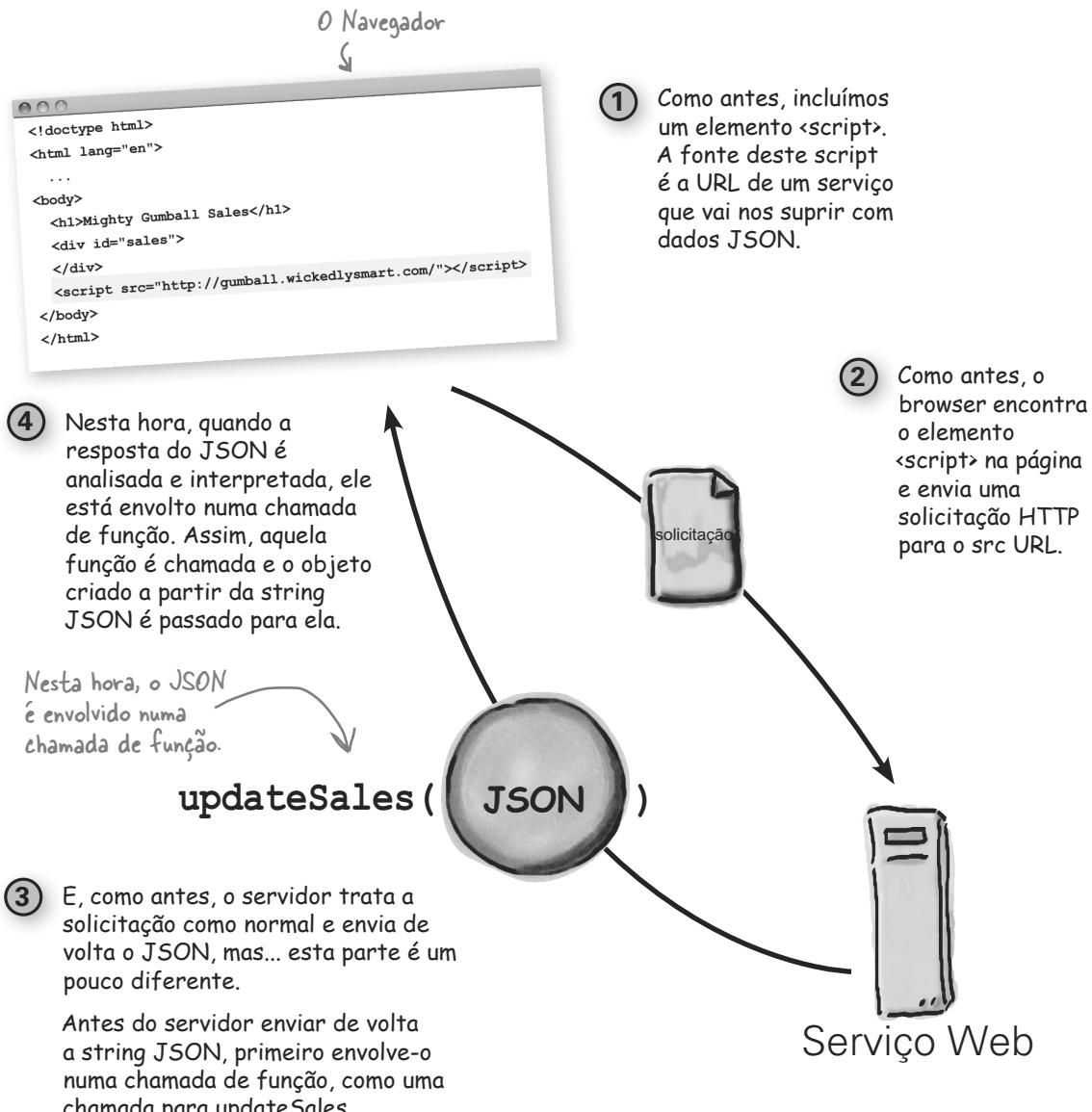


Serviço Web

Mas o que é o “P” de JSON?

Ok, a primeira coisa que você precisa saber sobre o JSONP é que ele possui um nome idiota e pouco óbvio: “JSON with Padding” (JSON com Revestimento). Se tivéssemos que nomeá-lo, iríamos chamá-lo de algo como “JSON com um Callback” ou “traga-me um JSON e execute-o quando voltar” ou, vejamos, realmente qualquer outra coisa serviria que não fosse JSON com revestimento.

O revestimento, porém, equivale a envolver uma função em torno do JSON, antes que ele volte na solicitação. Veja como funciona:





Entendo como usar a tag <script> para fazer com que o browser vá resgatar o JavaScript e como o servidor consegue pôr seus dados naquele JavaScript. E quanto ao nome da função? Como que o serviço web poderá saber o nome certo da função? Tipo, como o serviço web Mighty Gumball sabe chamar o updateSales? E se eu tiver outro serviço e quiser chamá-lo de, sei lá, updateScore, ou alerta, ou qualquer outra coisa?

Os serviços web permitem-nos especificar uma função callback.

Em geral, os serviços web deixam que você especifique o nome que quiser para a função. Embora não tenhamos dito, a Mighty Gumball já dá suporte a uma maneira de se fazer isso. Veja como funciona: quando você especificar sua URL, adicione um parâmetro no fim, assim:

`http://gumball.wickedlysmart.com/?callback=updateSales`

↑
Esta é a URL
que temos usado.

E aqui adicionamos um parâmetro URL, `callback`, que diz para usar a função `updateSales` quando o JavaScript for gerado.

A Mighty Gumball vai então usar a `updateSales` para envolver o objeto formatado JSON antes de lhe enviar de volta. Comumente, os serviços web nomeiam o parâmetro `callback`, mas verifique na documentação de seu serviço web para ter certeza o que estão utilizando.

PODER DO CÉREBRO

Experimente essas URLs: o que você vê na resposta?

`http://search.twitter.com/search.json?q=hfhtml5&callback=myCallback`

`http://search.twitter.com/search.json?q=hfhtml5&callback=justDoIt`

`http://search.twitter.com/search.json?q=hfhtml5&callback=updateTweets`

Nota: o Firefox irá pedir que abra ou salve um arquivo.
Você pode abri-lo com oTextEdit, Notepad, ou
qualquer editor de texto.

Gente, conseguimos. Tomou-nos um bom tempo para chegarmos a um consenso e usarmos um elemento <script> para acessarmos um serviço web, mas agora até parece mais fácil que usar XMLHttpRequest.



Jim: Bem, quase.

Joe: Acho que isso realmente nos permite deletar um pouco do código.

Frank: Estou pronto para deixar isso tudo bem bonito quando você tiver terminado.

Jim: Então, Joe, gênio dos códigos, o que tem em mente?

Joe: Com o XMLHttpRequest estávamos resgatando uma string. Ao usar JSONP, a tag script vai analisar e avaliar o código que está voltando. Portanto, na hora que pusermos as mãos nos dados, eles serão um objeto JavaScript.

Jim: Certo, e com o XMLHttpRequest estávamos usando JSON.parse para converter a string para um objeto. Podemos simplesmente nos livrar disso?

Joe: Sim, sim. É nisso que acredito!

Jim: E o que mais?

Joe: Bem, obviamente que precisamos inserir o elemento <script>.

Jim: Estava pensando sobre isso. Onde o colocaremos?

Joe: Bem, o browser vai controlar quando ele carrega, e nós queremos que a página seja carregada primeiro, para assim atualizarmos o DOM quando o updateSales for chamado. A única maneira que penso em lidar com isso é pondo <script> no fim da página, no corpo do HTML.

Jim: Sim, parece um bom palpite. Deveríamos olhar para isso com mais carinho. Para início de conversa, vamos tentar isso.

Joe: Ok, quero ver esse código funcionando! Vamos inseri-lo!

Frank: É melhor vocês se apressarem, pessoal. Aposto que ela já tem sua própria versão em atividade.

Vamos atualizar o aplicativo web Mighty Gumball

Está na hora de atualizar seu código Mighty Gumball com JSONP. A não ser pelo fato de remover o código existente que lida com a chamada XMLHttpRequest, todas as outras mudanças são menores. Vamos fazê-las agora:

O que precisamos fazer:

- ① Remova nosso código XMLHttpRequest.
- ② Certifique-se de que a função updateSales esteja pronta para receber um objeto, não uma string (como era com o XMLHttpRequest).
- ③ Adicione o elemento <script> para efetuar o resgate real dos dados.

-
- ① Todo o código em nossa função onload foi envolto em código no XMLHttpRequest, então podemos apenas deletá-lo. Manteremos a função onload por perto, caso precisemos dela mais tarde. Por ora, ela não fará nada. Abra seu arquivo `mightygumball.js` e faça essas mudanças:

```
window.onload = function() {  
    var url = "http://gumball.wickedlysmart.com";  
    var request = new XMLHttpRequest();  
    request.open("GET", url);  
    request.onload = function() {  
        if (request.status == 200) {  
            updateSales(request.responseText);  
        }  
    };  
    request.send(null);  
}
```

Por ora, delete todo código desta função.

- ② Depois, lembre-se que, quando usamos o elemento <script>, estamos dizendo ao browser que ele precisa resgatar o JavaScript e, assim, o browser resgata-o, analisa-o e o avalia. Isso significa que, na hora que chegar a sua função updateSales, o JSON não mais estará em sua forma string, mas será um objeto JavaScript de primeira classe. Quando usamos XMLHttpRequest, os dados voltam na forma de string. Neste momento, a updateSales presume que é uma string; então, vamos mudar isso de forma que ela manipule um objeto, não uma string:

```
function updateSales(responseText) {
  function updateSales(sales) {
    var salesDiv = document.getElementById("sales");
    var sales = JSON.parse(responseText); ← Remova o responseText e
    for (var i = 0; i < sales.length; i++) { reescreva a linha com um
      var sale = sales[i]; parâmetro chamado sales.
      var div = document.createElement("div");
      div.setAttribute("class", "saleItem");
      div.innerHTML = sale.name + " sold " + sale.sales + " gumballs";
      salesDiv.appendChild(div);
    }
  }
}
```

Podemos deletar a chamada JSON.parse também.

↑ E é isso: temos agora uma função pronta para lidar com nossos dados.

- ③ Finalmente, vamos adicionar o elemento <script> para fazer o verdadeiro resgate de dados.

```
<!doctype html>
<html lang="en">
<head>
  <title>Mighty Gumball</title>
  <meta charset="utf-8">
  <script src="mightygumball.js"></script>
  <link rel="stylesheet" href="mightygumball.css">
</head>
<body>
  <h1>Mighty Gumball Sales</h1>
  <div id="sales">
  </div>
  <script src="http://gumball.wickedlysmart.com/?callback=updateSales"></script>
</body>
</html>
```

Este é o link para o serviço web Mighty Gumball. Estamos usando o parâmetro callback e especificando nossa função, updateSales, de maneira que o serviço web envolva o JSON numa chamada de função para updateSales.

Test drive do seu novo código carregado de JSON



Se você fez todas as suas mudanças, está na hora de um test drive. Carregue novamente `mightygumball.html` em seu browser. Você está agora carregando os dados de vendas da Mighty Gumball, usando seu aplicativo web e o JSONP. A página deverá parecer igual a quando estava pegando os dados de vendas do arquivo local, mas você sabe que está usando um método completamente diferente de obter dados.

Aqui está o que vemos quando recarregamos a página Mighty Gumball. Você terá cidades e vendas diferentes, porque estes são dados reais.



| Cidade | Vendas |
|---------------|------------|
| IMPERIAL | 4 gumballs |
| SAN FRANCISCO | 5 gumballs |
| CHINO HILLS | 8 gumballs |
| STANFORD | 3 gumballs |
| CARSON | 9 gumballs |
| GOLETA | 6 gumballs |
| BRADLEY | 5 gumballs |
| CAPAY | 7 gumballs |
| LANCASTER | 8 gumballs |
| SAN QUENTIN | 5 gumballs |



O JSONP parece mais
um grande buraco de
segurança para mim!

**Não é nem mais nem menos seguro
que usar <script> para carregar o
JavaScript.**

É verdade: se fizer uma solicitação JSONP para um serviço web malicioso, a resposta poderá incluir código JavaScript que você pode não estar esperando e o browser o executará.

Não é, no entanto, nada diferente do que incluir JavaScript criando um link com bibliotecas hospedadas em outros servidores. Em qualquer momento que você fizer um link com o JavaScript, não importa se for com a biblioteca no `<head>` de seu documento, ou usando JSONP, você precisa ter certeza de que confia naquele serviço. Se estiver escrevendo um aplicativo web que usa autenticação para dar ao usuário acesso a dados sensíveis, talvez seja melhor não usar bibliotecas de terceiros ou dados JSON hospedados em qualquer outro servidor.

Portanto, escolha com cuidado os serviços web com os quais você se conecta. Se estiver usando uma API como Google, Twitter, Facebook ou qualquer um dos muitos serviços web conhecidos por aí, você estará seguro. Caso contrário, aconselhamos tomar precaução.

Em nosso caso, conhecemos pessoalmente os engenheiros da Mighty Gumball e sabemos que eles nunca poriam qualquer conteúdo malicioso em seus dados JSON. Então, você pode ficar tranquilo para prosseguir.



Conversa Informal



Bate-papo desta noite: **XMLHttpRequest e JSONP**

Esta noite, temos dois métodos populares de resgatar dados de seu browser.

XMLHttpRequest:

Sem ofensas, mas você não é meio que um invasor? Quero dizer, seu propósito é resgatar códigos e as pessoas estão usando você para fazer solicitações de dados.

Tudo que você está fazendo é juntar dados com código. Não há como fazer suas solicitações diretamente do código JavaScript; você tem que usar um elemento `<script>` HTML. Parece-me muito confuso para seus usuários.

Ei, XML ainda é bastante usado, não vem com essa. E você pode resgatar JSON muito bem comigo.

Pelo menos comigo você tem controle de quais dados são analisados no JavaScript. Com você, isso simplesmente acontece...

Bem, você pode ir em frente e usar um hack, como o JSON com revestimento — heh, nome estúpido — ou pode usar a coisa certa, o XMLHttpRequest e crescer com ele à medida que vai evoluindo. Afinal, as pessoas têm trabalhado para me tornar mais flexível com segurança.

JSONP:

Invadir? Eu chamaria isso de elegância. Podemos usar os mesmos meios de resgatar código e dados. Por que ter duas maneiras de se fazer isso?

Ei, isso funciona, e ainda permite que as pessoas escrevam códigos que resgatem JSON de serviços como o Twitter ou Google e muitos outros. Como você faria isso com o XMLHttpRequest, dadas as suas restrições de segurança? Digo, você ainda está preso na antiguidade, “XML”, hâ...

Claro, se quiser sempre analisar o resultado com `JSON.parse`.

Isso é uma vantagem — no momento em que meus usuários obtêm seus dados, é tudo muito bem analisado para eles. Veja, tenho muito respeito por você, que tornou toda essa história de escrever aplicativos possível, mas o problema é que você é muito restrito. Hoje em dia, neste mundo de serviços web, precisamos ser capazes de fazer solicitações a outros domínios.

Claro que as pessoas estão desenvolvendo novas maneiras, mas meus usuários possuem reais necessidades hoje em dia — eles não podem ficar esperando você resolver todos os seus problemas de domínios cruzados.

XMLHttpRequest:

Não tenho nada a ver com o nome Ajax, portanto não me pergunte! A propósito, você nunca disse como pode ser seguro...

Tudo que posso dizer é que, se você não precisa obter o dado de outro lugar, como do Twitter ou do Google, e estiver escrevendo seu próprio serviço web e cliente, fique comigo. Sou mais seguro e mais direto no uso.

Sei, sei... bobagem.

Qual é?! Não é preciso tanto código para me dar suporte em browsers como o IE5...

Sei, bem, existem outras coisas mais importantes... Você já tentou fazer algo iterativo, no qual você precise resgatar coisas repetidas vezes? Como aquele negócio da Mighty Gumball em que eles têm trabalhado. Como eles vão fazer isso funcionar?

Eis a minha impressão de seus leitores tendo acabado de ouvir o que você disse: “Como é que é?”

JSONP:

Não há nada de estúpido com “revestimento”. Significa apenas que, quando um usuário faz uma solicitação a um serviço web, ele também lhe pede para adicionar um pequeno prefixo, como “updateSales()”, no resultado. E como eles vinham te chamando por um tempo? Ajax? É algum tipo de limpador de banheiro?

Os codificadores sempre precisaram ser cuidadosos. Se estiver resgatando códigos de outro servidor, sim, você precisa saber o que está fazendo. A resposta, porém, não é apenas dizer “não faça isso”.

Alôô? Ninguém está escrevendo serviços que não usam dados externos. Já ouviu falar no nome “mashup”?

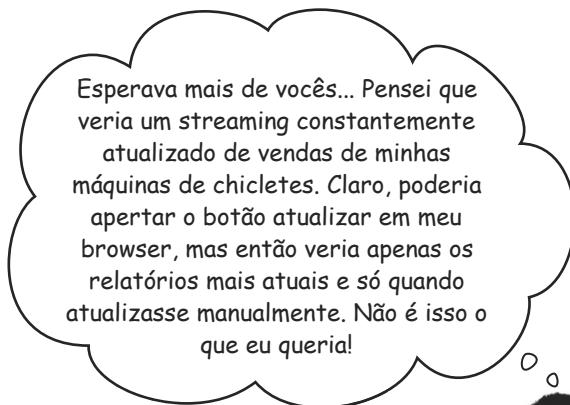
E... pelo menos eu sou suportado em qualquer lugar. Eu odeio ter de escrever código XMLHttpRequest que funcione em antigod navegadores

Haha, para mim só precisa de ZERO código. Apenas uma simples tag HTML.

Ei, não é tão ruim assim. Você só precisa escrever um novo elemento <script> dentro do DOM para fazer outra solicitação.

Use a Cabeça:

Obrigado rapazes! Receio que nosso tempo acabou!

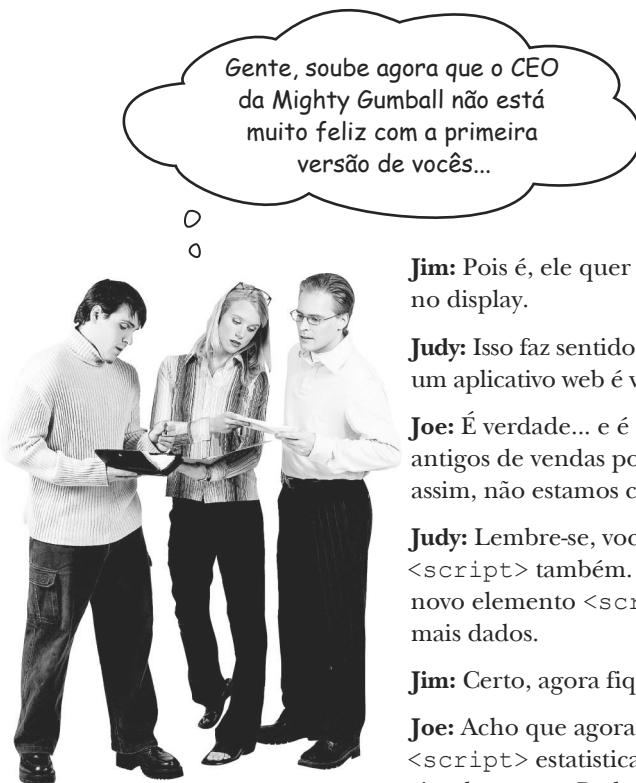


Esperava mais de vocês... Pensei que veria um streaming constantemente atualizado de vendas de minhas máquinas de chicletes. Claro, poderia apertar o botão atualizar em meu browser, mas então veria apenas os relatórios mais atuais e só quando atualizasse manualmente. Não é isso o que eu queria!

PODER DO CÉREBRO

Ele está certo. Precisamos mudar nosso aplicativo para que atualize o display com novas vendas em intervalos regulares (digamos, a cada dez segundos). No momento, estamos apenas pondo um elemento `<script>` dentro da página que inicia a solicitação ao servidor apenas uma vez. Consegue pensar em alguma maneira de utilizar o JSONP para resgatar continuamente novos relatórios de vendas?

Dica: usando o DOM podemos inserir um novo elemento `<script>` dentro da página. Será que funciona?



Jim: Pois é, ele quer os dados sendo continuamente atualizados no display.

Judy: Isso faz sentido. Quero dizer, uma grande vantagem para um aplicativo web é você não ter de atualizá-lo como uma página.

Joe: É verdade... e é óbvio que sabemos como substituir dados antigos de vendas por novos na página usando o DOM. Ainda assim, não estamos certos quanto à parte do JSONP.

Judy: Lembre-se, você não pode usar o DOM com o elemento `<script>` também. Em outras palavras, você pode criar um novo elemento `<script>` no DOM sempre que quiser resgatar mais dados.

Jim: Certo, agora fiquei enrolado. Pode repetir, por favor?

Joe: Acho que agora entendi. Estamos pondo o elemento `<script>` estatisticamente no HTML digitando-o simplesmente. Poderíamos, em vez disso, criar um novo elemento `<script>` com código JavaScript e adicioná-lo ao DOM. A única parte que não tenho certeza é se o browser fará outro resgate quando criarmos o novo elemento `<script>`...

Judy: Fará com certeza.

Jim: Entendo. Então, criamos um novo elemento `<script>` qualquer hora que quisermos que o browser faça uma operação do tipo JSONP para nós.

Judy: Certo! Parece que você está entendendo. E você sabe como fará isso outras vezes?

Jim: Bem, mmm, ainda não temos certeza, pois estávamos pensando a respeito do JSONP.

Judy: Você sabe tudo sobre funções handler por ora, sabe coisas tipo `onload` ou `onclick`. Pode criar um timer para chamar uma função handler num intervalo específico usando o método `setInterval` no JavaScript.

Joe: Então, vamos criar esse negócio e pôr o JSONP dinâmico para funcionar o quanto antes para o CEO da Gumball.

Jim: Ah, é tudo o que você precisa? É melhor nos apressarmos!

Melhorando o Mighty Gumball

Como pode ver, temos um pouco mais de trabalho a fazer, mas não será tão ruim assim. Basicamente, escrevemos nossa primeira versão obtendo os últimos relatórios de vendas da Mighty Gumball, mostrando-os apenas *uma vez*. Falha nossa, pois quase qualquer aplicativo web, hoje em dia, deveria monitorar continuamente os dados e atualizar o aplicativo em tempo (quase) real.

Eis o que precisamos fazer:

- ① Vamos remover o elemento `<script>` do JSONP do HTML da Mighty Gumball, pois não o usaremos mais.
- ② Precisamos criar um handler para lidar com a execução da solicitação do JSONP a cada poucos segundos. Levaremos em conta o conselho da Judy e usaremos o método `setInterval` do JavaScript.
- ③ Então, precisamos implementar nosso código JSONP no handler, de forma que, toda hora em que ele for chamado, faça uma solicitação para obter os últimos relatórios de vendas da Mighty Gumball.

Passo 1: Dando um jeito no elemento script...

Vamos usar um novo jeito de invocar nossas solicitações JSONP, portanto, vamos começar a remover o elemento `<script>` de nosso HTML.

```
<!doctype html>
<html lang="en">
<head>
  <title>Mighty Gumball</title>
  <meta charset="utf-8">
  <script src="mightygumball.js"></script>
  <link rel="stylesheet" href="mightygumball.css">
</head>
<body>
  <h1>Mighty Gumball Sales</h1>
  <div id="sales">
  </div>
  <script src="http://gumball.wickedlysmart.com/?callback=updateSales"></script>
</body>
</html>
```

Vá em frente e delete este elemento de seu arquivo HTML.

Passo 2: Agora é hora do timer

Ok, estamos progredindo de resgatar os relatórios de vendas uma vez para resgatá-los mais vezes, digamos a cada três segundos. Isso pode ser muito rápido ou lento dependendo do aplicativo, mas para a Mighty Gumball vamos começar com três segundos.

Agora, para fazer algo a cada três segundos, precisamos ter uma função que poderemos chamar a cada três segundos. Como Judy mencionou, podemos usar o método `setInterval` no objeto `window` para fazer isso; veja como fica:



```
setInterval(handleRefresh, 3000);
```

O método `setInterval` funciona como um handler e um intervalo de tempo.

Aqui está nossa função handler que vamos definir já, já.

E aqui está nosso intervalo de tempo, expresso em milissegundos. 3000 milissegundos = 3 segundos.

Portanto, a cada 3.000 milissegundos o JavaScript invocará seu handler, neste caso, a função `handleRefresh`. Vamos escrever um simples handler e fazer uma tentativa:

```
function handleRefresh() {
    alert("I'm alive");
}
```

Toda vez em que isso for chamado (e será a cada 3 segundos), vamos jogar o alerta "I'm alive" (Estou vivo).

Agora, só precisamos de um pouco de código para criar a chamada `setInterval`, a qual será adicionada à função `.onload`, para que assim ela seja estabelecida logo após a página inteira ser carregada:

```
window.onload = function() {
    setInterval(handleRefresh, 3000);
}
```

Esta é nossa antiga função `.onload` que não tinha nada nela após termos deletado o código `XMLHttpRequest`.



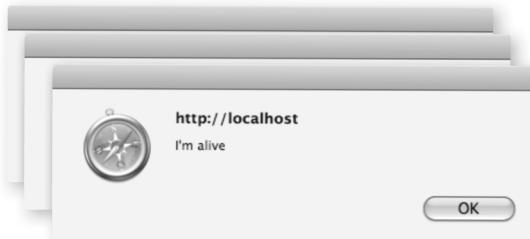
Tudo o que precisamos fazer é adicionar nossa chamada para `setInterval`, a qual, quando a função `init` estiver rodando, dará a partida num timer que dispara a cada três segundos e chama nossa função `handleRefresh`.

Vamos experimentar isso e, então, quando soubermos que está funcionando — isto é, quando virmos nosso handler sendo acionado a cada três segundos —, implementaremos o código JSONP.

Um test drive guiado pelo tempo



Isso vai ser divertido. Certifique-se de ter digitado a função handleRefresh e também de ter feito as mudanças no handler onload. Salve tudo e carregue em seu browser. Você verá uma sequência de alertas e terá de fechar a janela de seu browser para pará-la!



Aponte o seu lápis

Agora que você conhece o setInterval (sem falar no XMLHttpRequest e no JSONP), pense em maneiras para usá-los em outros aplicativos da web. Liste-as aqui:

Cheque e atualize o progresso de uma tarefa e mostre-a.

Veja se algum novo comentário foi postado em algum tópico.

Atualize um mapa se qualquer amigo tiver aparecido nas redondezas.



Passo 3: Reimplementando o JSONP

Ainda queremos usar o JSONP para resgatar nossos dados, mas precisamos de uma maneira de fazer isso sempre que nosso handler refresh for chamado, não apenas durante o carregamento da página. É aí que entra o DOM — a melhor coisa no DOM é que podemos inserir novos elementos para dentro dele *a qualquer momento*, mesmo nos elementos `<script>`. Portanto, deveríamos ser capazes de inserir um novo elemento `<script>` a qualquer hora que quiséssemos fazer uma chamada JSONP. Vamos trabalhar um pouco com códigos, usando tudo que sabemos sobre o DOM e o JSONP para fazer isso.

Primeiro, vamos criar a URL JSONP

Esta é a mesma URL que usamos com nosso elemento script anterior. Aqui, vamos designá-la a uma variável para uso posterior. Delete o alerta de seu handler e adicione este código:

```
Voltamos a nossa
função handleRefresh.
↓
function handleRefresh() {
    var url = "http://gumball.wickedlysmart.com?callback=updateSales";
}
```

Aqui, estamos criando a URL JSONP e designando-a para a url variável.

Depois, vamos criar um novo elemento script

Agora, em vez de ter o elemento `<script>` em nosso HTML, vamos construir um elemento `<script>` usando JavaScript. Precisamos criar o elemento para então ajustar seus atributos `src` e `id`:

```
function handleRefresh() {
    var url = "http://gumball.wickedlysmart.com?callback=updateSales";

    var newScriptElement = document.createElement("script");
    newScriptElement.setAttribute("src", url);
    newScriptElement.setAttribute("id", "jsonp");
}

O método setAttribute pode parecer novo
para você (apenas o mencionamos de passagem,
até agora), mas é bem fácil de ver o que ele
faz. O método setAttribute permite-lhe
ajustar os atributos de um elemento HTML,
como os atributos src e id ou mais uma
porção de coisas, como classe, href etc.
↑
```

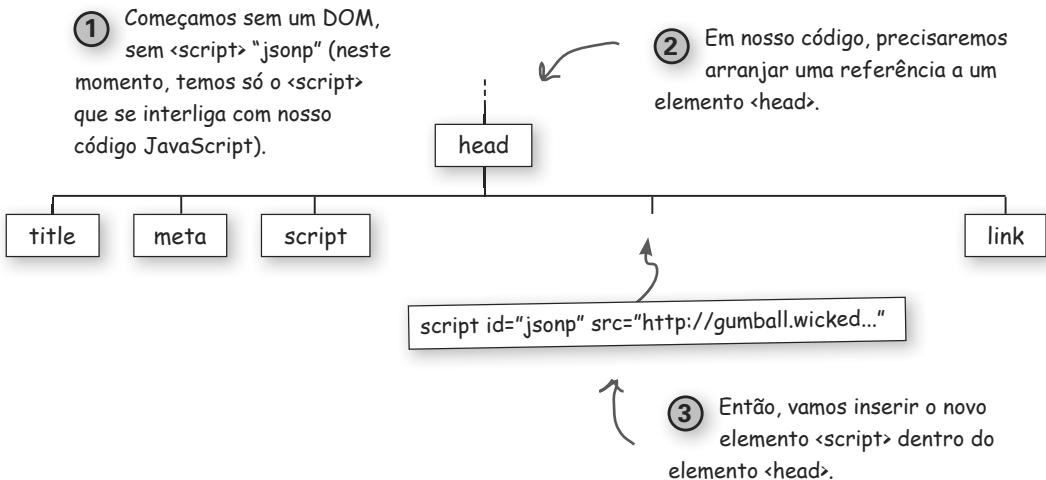
Primeiro, criamos um novo elemento script...

... e ajustamos o atributo `src` do elemento para nossa URL JSONP.

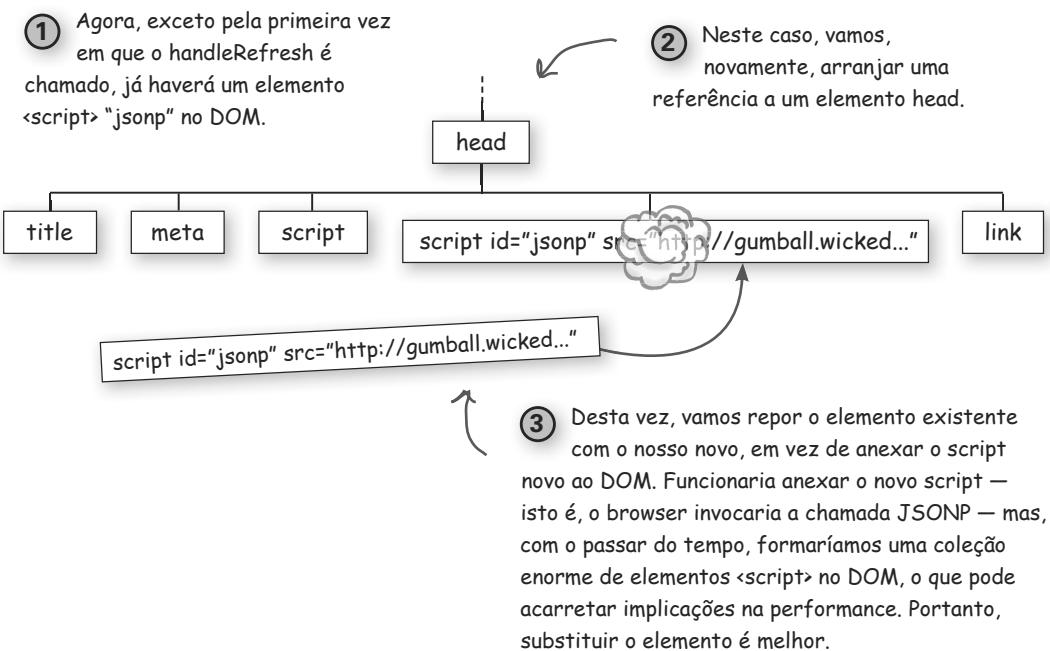
Vamos dar a este script uma id, de forma que possamos facilmente obtê-la novamente, o que precisaremos, como você verá.

Como inserimos o script para dentro do DOM?

Estamos quase lá. Precisamos apenas inserir nosso elemento script recém-criado. Uma vez feito isso, o browser o verá e fará o resto, fazendo com que a solicitação JSONP seja efetuada. Agora, para inserir o script, requer-se um pouco de planejamento e antecipação; vejamos como isso funcionará:



Uma vez tendo o script inserido, o browser verá o novo script no DOM e vai resgatar o que está na URL, no atributo src. Agora, temos um segundo caso. Vamos ver:



Vamos escrever o código para inserir o script dentro do DOM

Agora, que sabemos os passos, vamos verificar o código. Faremos isso em dois passos: primeiro, mostraremos o código para adicionarmos um novo script, então o código para substituí-lo:

```
function handleRefresh() {
    var url = "http://gumball.wickedlysmart.com?callback=updateSales";
    var newScriptElement = document.createElement("script");
    newScriptElement.setAttribute("src", url);
    newScriptElement.setAttribute("id", "jsonp");
    var oldScriptElement = document.getElementById("jsonp");
    var head = document.getElementsByTagName("head")[0];
    if (oldScriptElement == null) {
        head.appendChild(newScriptElement);
    }
}
```



Agora que temos uma referência ao elemento `<head>`, verificaremos para ver se já existe um elemento `<script>` e, se não houver (se sua referência for `null`), então seguiremos em frente e anexaremos o novo elemento `<script>` ao cabeçalho.

Primeiro, vamos arranjar a referência ao elemento `<script>`. Se não existir, retornaremos `null`. Perceba que estamos contando que isso tenha a id "jsonp".

Depois, vamos pegar uma referência ao elemento `<head>`, utilizando um novo método `document.getElementsByTagName`. Voltaremos para ver como isso funciona, mas por ora basta saber que ele arranja uma referência para o elemento `<head>`.

Ok, vamos verificar o código que substitui o elemento `script`, se ele já existir. Vamos apenas mostrar o comando condicional `se`, que é onde está todo o código novo:

Aqui está nossa condicional novamente, lembrando que ela está apenas checando para ver se um elemento `<script>` já existe no DOM.

```
if (oldScriptElement == null) {
    head.appendChild(newScriptElement);
} else {
    head.replaceChild(newScriptElement, oldScriptElement);
}
```

Se já existe um elemento `<script>` no cabeçalho, então apenas o substituimos. Você poderá usar o método `replaceChild` no elemento `<head>` e passá-lo aos scripts velhos e novos para fazer isso. Veremos um pouco mais de perto esse novo método já, já.



getElementsByName de perto

Esta é a primeira vez que você vê o método `getElementsByTagname`; portanto, o veremos rapidamente mais de perto. É parecido com o `getDocumentById`, exceto pelo fato de que ele retorna um array de elementos que combina com um determinado nome de uma tag.

O `getElementsByTagname` retorna todos os elementos do DOM com esta tag.

```
var arrayOfHeadElements = document.getElementsByTagName("head");
```

Neste caso, ele retorna um array de elementos head.

Uma vez que você tem o array, pode obter o primeiro item dentro dele usando o índice 0:

```
var head = arrayOfHeadElements[0];
```

Retorna o primeiro elemento head do array (e deveria haver apenas um, certo?).

Agora, podemos combinar essas duas linhas, assim:

```
var head = document.getElementsByTagName("head")[0];
```

Pegamos o array e então colocamos o índice dentro dele para obter o primeiro item em um passo.

Em nosso código de exemplo, sempre usamos o primeiro elemento `<head>`, mas você pode usar este método em qualquer tag, como `<p>`, `<div>` e assim por diante. Normalmente, você terá de volta mais de um daqueles no array.



replaceChild de Perto

Vamos ver também o método `replaceChild`, porque você não viu antes. Chame o método `replaceChild` no elemento em que quiser substituir uma tag filha, passando as referências tanto às novas quanto às antigas tag filhas. O método simplesmente substitui a tag filha antiga pela nova.

O método `replaceChild` diz ao elemento `<head>` para substituir uma de suas tag filhas, `oldScriptElement`, com uma nova tag filha, `newScriptElement`.

Nosso novo elemento `<script>`

O `<script>` que já está na página.

```
head.replaceChild(newScriptElement, oldScriptElement);
```

Perguntas Idiotas

P: Por que não posso simplesmente substituir os dados no atributo src em vez de substituir o elemento <script> por inteiro?

R: Se você substituir apenas o atributo src pela nova URL, o browser não o verá como um novo script, e assim não fará a solicitação para resgatar o JSONP. Para forçar o browser a fazer a solicitação, temos de criar este novo script por completo. Esta é uma técnica conhecida como “script injection”.

P: O que acontece à child antiga quando a substituo?

R: Ela é removida do DOM. O que acontece a partir dali, depende de você: se ainda tiver uma referência a ela numa variável em algum lugar, você poderá continuar a usá-la (de qualquer maneira poderá ter sentido). No entanto, se não tiver, o tempo de execução do JavaScript poderá, eventualmente, reclamar a memória que o elemento está ocupando em seu browser.

P: E se houver mais de um elemento <head>? Seu código parece depender da existência de apenas um cabeçalho quando você indexa a zero o array retornado por getElementByTag...

R: Por definição, um arquivo HTML possui apenas um elemento <head>. Dito isto, claro, alguém pode digitar dois dentro de um arquivo HTML. Neste caso, seus resultados podem variar (é isso o que consegue por não validar seu HTML!), mas, como sempre, o browser fará o seu melhor para fazer a coisa certa (o que será, vai depender do browser).

P: Posso parar o timer de intervalos após iniciá-lo?

R: Certamente. O método setInterval, na verdade, retorna uma id que identifica o timer. Ao armazenar a id numa variável, você pode então passá-la ao método clearInterval a qualquer momento para parar o timer. Fechar seu browser também interrompe a contagem dele.

P: Como posso saber quais os parâmetros que um serviço web utiliza e se ele dá suporte a JSON e JSONP?

R: A maioria dos serviços web publica uma API pública que inclui as maneiras com as quais você pode acessar o serviço, assim como todas as coisas que poderá fazer com tal serviço. Se estiver usando uma API comercial, talvez precise arrumar essa documentação diretamente com o provedor. Para muitas APIs públicas, é mais provável que você encontre a informação necessária por intermédio de uma busca na web ou na área do desenvolvedor da organização dos sites delas. Também é bom visitar sites como programtheweb.com, que documenta a crescente lista de APIs por aí.

P: O XMLHttpRequest é claramente mais antigo que o HTML5, mas e o JSON e JSONP? Eles pertencem ao HTML5? Preciso do HTML para usá-los?

R: Podemos chamar o JSON e o JSONP de contemporâneos do HTML5. Apesar de nenhum deles ser definido por uma especificação HTML5, são amplamente utilizados por aplicativos HTML5, sendo parte fundamental na construção de aplicativos web. Portanto, quando dizemos HTML=Marcação + APIs JavaScript + CSS, bem, JSON e JSONP são, sem dúvida, partes integrantes desta equação (assim como as solicitações usando HTTP com XMLHttpRequest).

P: As pessoas ainda usam XML ou é tudo JSON agora?

R: Um truismo na indústria dos computadores é que nada morre, e assim temos certeza de que ainda teremos o XML por muito tempo. Dito isto, poderíamos dizer também que o JSON vive seu momento agora e, por isso, muitos serviços têm sido criados usando esta tecnologia. Você encontrará frequentemente muitos serviços web que dão suporte a uma variedade de formatos de dados, incluindo XML, JSON e muitos outros (como RSS). O JSON possui a vantagem de que é baseado diretamente no JavaScript e o JSONP resolve nossos problemas de domínio cruzado.

Quase esquecemos: cuidado com a terrível cache do browser

Estamos quase prontos para seguirmos em frente, mas existe um pequeno detalhe que precisamos tomar cuidado, e é um daqueles problemas do tipo “se nunca fez isso antes, como poderia saber que precisa tratar dele”.

A maioria dos browsers têm uma propriedade interessante neles: se você resgatar a mesma URL repetidas vezes (como nossa solicitação JSONP fará), o browser acaba armazenando-a para ter mais eficiência, obtendo, então, sempre o mesmo arquivo (ou dado) armazenado. Não é o que queremos.

Felizmente, há uma cura fácil e “tão velha quanto a web” para isso. Tudo o que faremos é acrescentar um número aleatório ao fim da URL, e então o browser é levado a pensar que é uma URL nunca vista antes por ele. Vamos consertar nosso código, mudando a linha URL acima para:

Você encontrará este código no topo de sua função handleRefresh.

```
var url = "http://gumball.wickedlysmart.com/?callback=updateSales" +  
    "&random=" + (new Date()).getTime();
```

↑
Estamos adicionando um novo parâmetro “bobo” no fim da URL. O servidor web vai apenas ignorá-lo, mas será o suficiente para enganar o browser.

Mude sua declaração URL acima para se parecer com isso.

↑
Criamos um novo objeto Date, usamos o método getTime do objeto Date para obter o tempo em milissegundos, então acrescentamos o tempo no final da URL.

Com este novo código, a URL gerada parecerá assim:

Esta parte deverá parecer familiar...

<http://gumball.wickedlysmart.com/?callback=updateSales&random=1309217501707>

E aqui está o parâmetro aleatório.

↑
Esta parte mudará toda hora para acabar com o armazenamento.

Vá em frente e substitua a declaração variável url em sua função handleRefresh pelo código e então estaremos prontos para um test drive!

Mais uma vez, o test drive



Ótimo, com certeza pensamos em tudo desta vez. Deveríamos estar preparados. Certifique-se de já ter todo o código do último test drive e recarregue a página. Nossa, vemos contínuas atualizações!

Espere um minuto... você está vendendo o que estamos vendendo? O que são essas duplicidades? Isso não é bom. Mmm. Será que estamos resgatando muito rápido e obtendo relatórios que já resgatamos?

Duplicidades!



Como remover relatórios de vendas duplicados

Se der uma olhada rápida às Especificações da Gumball na página 228, você verá que pode especificar um último parâmetro `lastreporttime` na solicitação URL, assim:

Você também pode adicionar um parâmetro `lastreporttime` no fim da URL e terá apenas os relatórios desde aquele tempo. Use-o assim:

`http://gumball.wickedlysmart.com/?lastreporttime=1302212903099`

Apenas especifique um tempo em milissegundos.

Isso é ótimo, mas como saberemos a hora do último relatório resgatado?

Vamos ver o formato dos relatórios de vendas novamente:

```
[{"name": "LOS ANGELES", "time": 1309208126092, "sales": 2},
 {"name": "PASADENA", "time": 1309208128219, "sales": 8},
 {"name": "DAVIS CREEK", "time": 1309214414505, "sales": 8}
 ...]
```

Cada relatório de vendas possui a hora em que foi lançado.



Estou percebendo onde você está querendo chegar; podemos rastrear a hora do último relatório e então usar isso quando fizermos a próxima solicitação, de forma que o servidor não nos dê relatórios que já recebemos, certo?

Certo.

E para rastrear os últimos relatórios de vendas recebidos, vamos precisar acrescentar algumas coisas à função updateSales, onde todo o processamento dos dados das vendas acontece. Primeiro, no entanto, deveríamos declarar uma variável para marcar o tempo do relatório mais recente:

```
var lastReportTime = 0;
```

O tempo não pode ser menor que zero, portanto, vamos apenas ajustá-lo para 0, para os iniciantes.

Ponha isso no topo de seu arquivo JavaScript, frente a qualquer função.

Vamos pegar o tempo da venda mais recente em updateSales:

```
function updateSales(sales) {
    var salesDiv = document.getElementById("sales");
    for (var i = 0; i < sales.length; i++) {
        var sale = sales[i];
        var div = document.createElement("div");
        div.setAttribute("class", "saleItem");
        div.innerHTML = sale.name + " sold " + sale.sales +
            " gumballs";
        salesDiv.appendChild(div);
    }
    if (sales.length > 0) {
        lastReportTime = sales[sales.length-1].time;
    }
}
```

Se você olhar para o array de vendas, verá que a venda mais recente é a última do array. Portanto, aqui vamos designar isso para nossa variável lastReportTime.

Precisamos nos certificar, porém, que HAJA um array; se não existissem vendas novas, então resgatariamos um array vazio e nosso código aqui causaria uma exceção.

Atualizando a URL JSON para incluir o lastReportTime

Agora que estamos rastreando o último tempo relatado das vendas, precisamos ter certeza de que estamos enviando-o para a Mighty Gumball como parte da solicitação JSON. Para isso, editaremos a função handleRefresh e adicionaremos o parâmetro query lastReportTime assim:

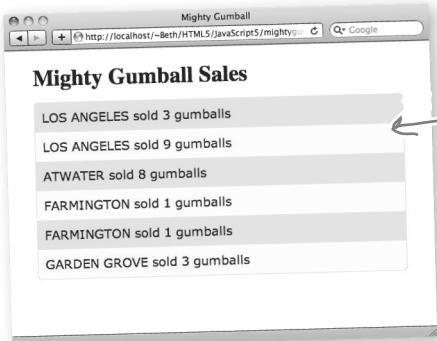
```
function handleRefresh() {
    var url = "http://gumball.wickedlysmart.com" +
              "?callback=updateSales" +
              "&lastReportTime=" + lastReportTime +
              "&random=" + (new Date()).getTime();

    var newScriptElement = document.createElement("script");
    newScriptElement.setAttribute("src", url);
    newScriptElement.setAttribute("id", "jsonp");
    var oldScriptElement = document.getElementById("jsonp");
    var head = document.getElementsByTagName("head")[0];
    if (oldScriptElement == null) {
        head.appendChild(newScriptElement);
    } else {
        head.replaceChild(newScriptElement, oldScriptElement);
    }
}
```

Dividimos a URL em várias strings que estávamos concatenando...
... e aqui está o parâmetro lastReportTime com seu novo valor.

Test drive lastReportTime

Vamos levar o lastReportTime parâmetro query para um test drive e ver se isso resolve nosso problema de relatórios de vendas duplicadas. Certifique-se de digitar o novo código, recarregue a página e clique no botão de atualizar.



Legal! Agora temos apenas novos relatórios de vendas. Todas as duplicidades sumiram!





Vocês se superaram! Funciona que é uma maravilha e agora posso ficar de olho nas vendas da minha mesa ou quando estiver no celular. Estou realmente começando a gostar desses aplicativos web. Pensem no que poderemos ser capazes de fazer com as máquinas de chicletes, JSON e todas essas APIs HTML5!





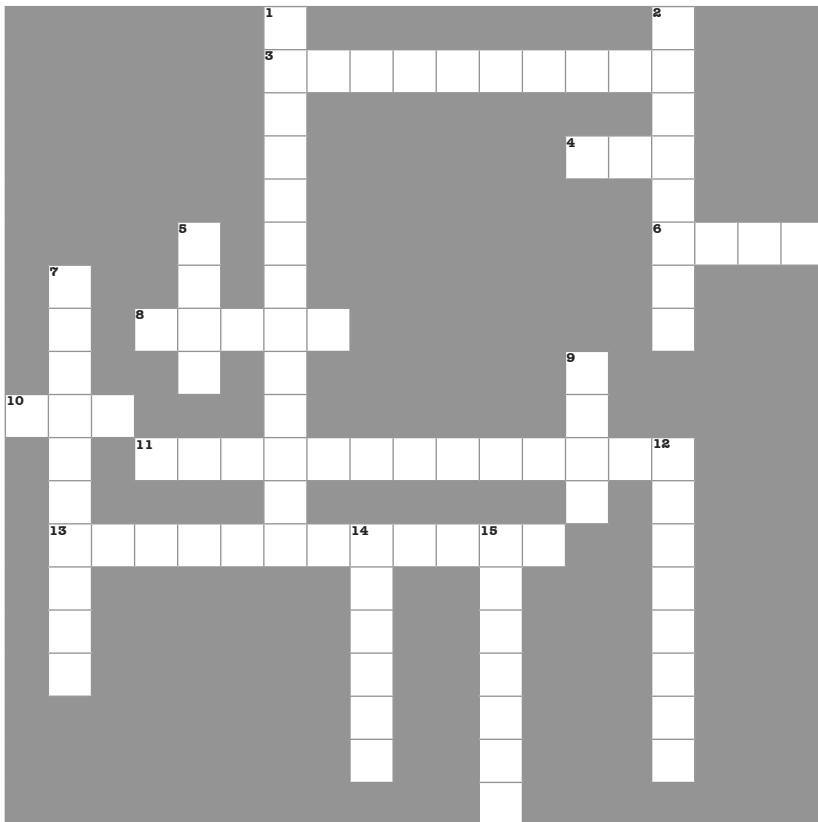
PONTOS DE BALA

- O XMLHttpRequest não lhe permite solicitar dados de um servidor diferente daquele que seu HTML e JavaScript estão se servindo. Esta é uma política de segurança de browsers feita para prevenir JavaScripts maliciosos de acessarem suas páginas e os cookies de um usuário.
- Uma alternativa para o XMLHttpRequest acessar dados hospedados por serviços web é o JSONP.
- Use o XMLHttpRequest quando seu HTML e seu JavaScript estiverem hospedados na mesma máquina que seus dados.
- Use o JSONP quando precisar acessar os dados hospedados por um serviço web num servidor remoto (presumindo que o serviço web suporte JSONP). Um serviço web é uma API web que é acessada por HTTP.
- JSONP é um método de resgatar dados usando o elemento <script>.
- JSONP são os dados JSON envoltos por um JavaScript; normalmente, uma chamada de função.
- As chamadas de função que envolvem os dados JSON tornando-os em JSONP são referidas como um "callback".
- Especifique a função callback como um parâmetro query de URL numa solicitação JSONP.
- O JSONP não é mais (nem menos) seguro que ligar bibliotecas JavaScript usando o elemento <script>. Tenha cuidado sempre que criar links a JavaScript terceirizado.
- Especifique o elemento <script> para fazer a solicitação JSONP, acrescentando-o diretamente a seu HTML, ou escrevendo o elemento <script> ao DOM utilizando JavaScript.
- Use um número aleatório no fim da URL da sua solicitação JSONP se estiver fazendo a solicitação diversas vezes; assim, o browser não armazenará as respostas.
- O método replaceChild substitui um elemento no DOM por outro elemento.
- O setInterval é um timer que chama a função num intervalo específico. Você pode usar o setInterval para fazer repetidas solicitações JSONP a um servidor para resgatar novos dados.



Palavras Cruzadas HTML5

Nossa, você fez seus aplicativos conversarem com a web neste capítulo! Hora de um pouco de atividade para o lado esquerdo do cérebro para ajudá-lo a memorizar tudo.



HORIZONTAIS

3. A Mighty Gumball está testando o MG2200 em ____.
4. O formato que todos pensamos que salvaria o mundo.
6. ___, o cara do CQ, ficou irritado quando a solicitação ao servidor de produção da Gumball falhou.
8. O Guru ensina ao Gafanhoto que os argumentos de função são também ____.
10. É fácil criar um servidor local num(a) ____.
11. Ficamos ___ por passarmos vinte e cinco páginas do capítulo sem descobrir a política de segurança do browser.
13. JSONP é abreviatura de JSON com ____.

VERTICIAIS

1. Um dos apelidos do XMLHttpRequest na Microsoft.
2. O JSONP usa um(a) ____.
5. O padrão para usar o XMLHttpRequest para obter dados de servidores é às vezes chamado ____.
7. O JSONP usa esses tipos de objetos.
9. ___ lembrou Frank, Jim e Joe a respeito de problemas de segurança de domínio cruzado com o XMLHttpRequest.
12. Este capítulo teve um(a) desses(as) no meio.
14. ___ é a última máquina de chicletes habilitada para web da Mighty Gumball.
15. ___ possui um serviço web JSONP.

Uma Mensagem Especial do Capítulo 7...



Embora tenhamos feito
você pensar sobre o JSONP,
adoraríamos ter sua ajuda no
capítulo *Canvas*.

Estamos trabalhando com a API
JSONP do Twitter e construindo um
serviço que lhe permite inserir qualquer
tweet numa camiseta.

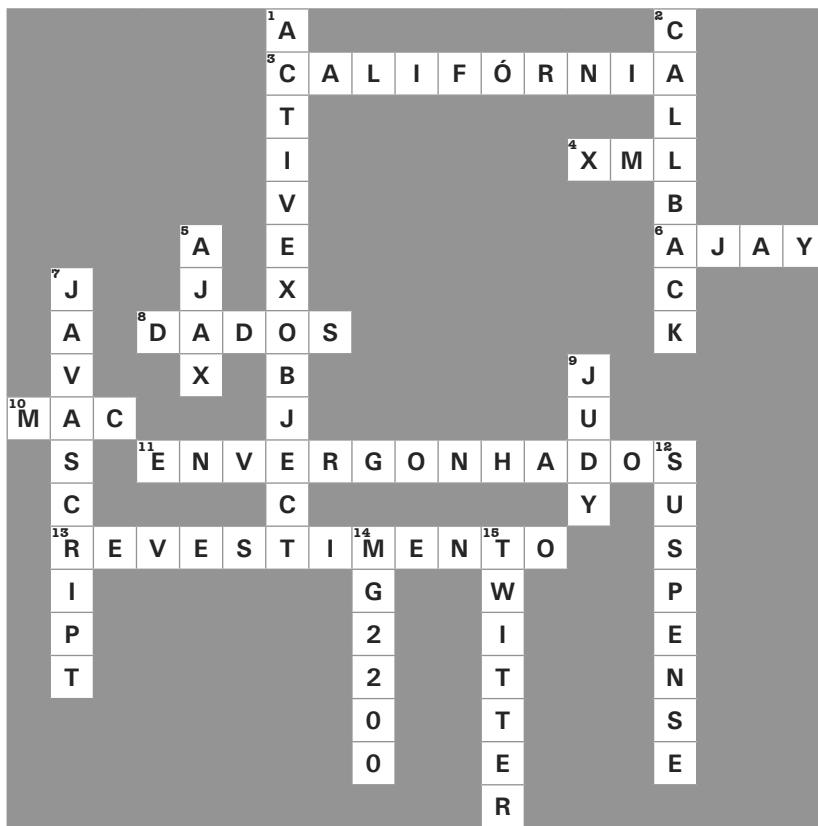
Gostaríamos de dizer "se
vale a pena tuitar, vale
a pena imprimir numa
camiseta".



Fundadora da
TweetShirt.com



Solução das Palavras Cruzadas HTML5



7 descobrindo seu artista interior

O Canvas



Sim, claro, marcação é legal e tal, mas não há nada como pôr as mãos na massa e pintar com frescos e puros pixels.

A HTML deixou de ser apenas uma linguagem “marcação”.

Com o novo elemento canvas da HTML5, você tem o poder de criar, manipular e destruir *pixels*, bem nas suas mãos. Neste capítulo, usaremos o elemento canvas para descobrir o artista que vive em você — chega de falar que o HTML é só semântico e sem apresentação; com o canvas, vamos pintar e desenhar com cores. Agora, é *tudo* uma questão de apresentação. Vamos aprender como posicionar um canvas em suas páginas, como desenhar texto e gráficos (usando JavaScript, é claro) e até mesmo como lidar com browsers que não suportam o elemento canvas. Canvas não é apenas uma maravilha que só serve para uma coisa: você verá muito mais em outros capítulos neste livro.

←
Ok,
“destruir”
talvez seja
um pouco
demais...

↑
De fato, sabemos que <canvas> e <video> têm
compartilhado mais do que apenas páginas... vamos
descobrir os detalhes mais legais só mais tarde.

Nossa nova partida: TweetShirt

Nosso slogan é “se vale a pena escrever no Twitter, vale a pena imprimir numa camiseta”.

Afinal, metade da batalha de chamar a si mesmo de jornalista é estar disposto a colocar suas próprias palavras impressas. Portanto, qual lugar seria melhor do que o peito de outra pessoa? Pelo menos é esse o nosso ponto de partida e estamos nos atendo a ele.

Agora, há apenas uma coisa que fica no caminho de conseguir que esse plano decole: precisamos de um aplicativo web legal que permita que os clientes criem um design customizado de camiseta, aplicando um de seus tweets recentes.



↑
Você provavelmente está pensando consigo mesmo “quer saber de uma coisa? Isso não é uma má ideia”. Bem, vamos nessa então. Vamos fazer esse negócio decolar até o fim do capítulo. Ah, e se você de fato for adiante com isso e fizer algum dinheiro, não vamos reclamar quaisquer direitos de propriedade intelectual ou algo assim, mas, pelo menos, mande-nos uma camiseta de presente!

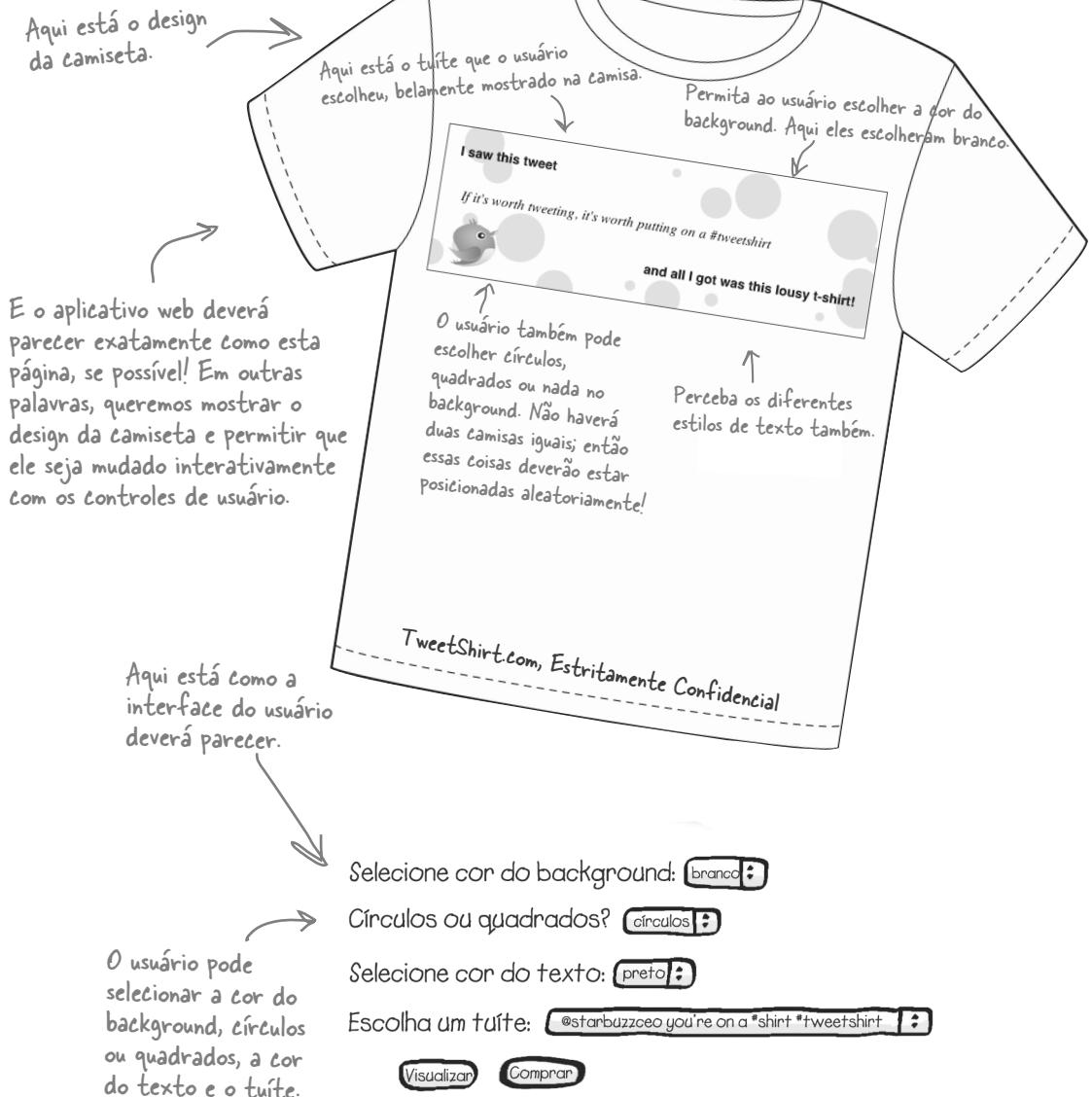


Verificando os “esboços”

Após um design iterativo exaustivo e testes extensivos com grupos-alvo, chegamos a um esboço (também conhecido como um design visual inicial) pronto para você analisar.

Vamos dar uma olhada:

Tá bom, nós somos iniciantes, fizemos isso num guardanapo no Starbuzz Coffee.



Poder do Cérebro

Veja novamente os requerimentos na página anterior. Como você acha que pode realizar isso usando HTML5? Lembre-se: um requerimento é se certificar de que seu site funciona em tantos formatos e tamanhos de dispositivos quanto possível.

Verifique todas as possibilidades abaixo (e então circule a melhor resposta):

- Apenas use Flash; funciona na maioria dos browsers.
- Dê uma olhada no HTML5 e veja se existe alguma nova tecnologia que possa ajudar (dica: deve existir alguma chamada canvas).
- Escreva um aplicativo customizado para cada dispositivo. Desta forma, você saberá a exata experiência obtida.
- Apenas compute a imagem no lado servidor e entregue uma imagem customizada de volta ao browser.

P^{não existem}erguntas idiotas

P: Vai, sério, por que não o Flash ou um aplicativo personalizado para esse caso?

R: Flash é uma grande tecnologia e você certamente poderia usá-la. No momento, porém, a indústria está indo em direção ao HTML5 e, enquanto escrevemos isso, você deve estar tendo problemas em rodar seu aplicativo Flash em todos os dispositivos, incluindo alguns bem populares.

Um aplicativo pode ser uma boa escolha, se você realmente precisar de uma experiência que seja totalmente personalizada para o dispositivo. Tenha em mente que desenvolver um aplicativo personalizado para vários dispositivos é caro.

Com o HTML5, você obtém o maior suporte para dispositivos móveis e desktop e poderá criar mais vezes um aplicativo, usando uma única solução tecnológica.

P: Gosto da ideia de criar a imagem no servidor. Dessa forma, posso escrever um pedaço de código e as imagens funcionam em todos os dispositivos. Conheço um pouco de PHP, então talvez eu chegue lá.

R: Essa seria outra forma de dar certo, mas as desvantagens são que, se você tiver um zilhão de usuários, terá de se preocupar com o dimensionamento de todos os servidores para cumprir com a demanda (versus cada cliente tendo de lidar com a geração de uma pré-visualização da camiseta). Você também terá que possuir uma experiência muito mais interativa e harmoniosa, se, no lugar disso escrever para o browser.

Como? Bem, que bom que você perguntou...

Vamos falar com o pessoal da TweetShirt...

Você já sabe dos requerimentos e já tem um design básico para a experiência do usuário. Agora, vem a parte difícil: fazer funcionar. Vamos bisbilhotar um pouco e ver aonde tudo isso vai levar...

Joe: Pensei que seria simples até ver aqueles círculos no background.

Frank: O que quer dizer? É apenas uma imagem...

Judy: Não, não, a fundadora quer que aqueles círculos sejam aleatórios; assim, os círculos na minha camiseta serão diferentes da sua. O mesmo se aplica aos quadrados.

Frank: Tudo bem, já fizemos isso no passado gerando uma imagem no lado servidor.

Joe: Sim, eu sei, mas aquilo não funcionou muito bem; lembra que tivemos de pagar todas aquelas taxas para a Amazon?

Frank: Ah... é... esquece...

Joe: E, de qualquer jeito, queremos que esse negócio seja instantaneamente gratificante, sabe, sem longas viagens de volta ao servidor. Então, vamos fazer tudo no lado cliente, se pudermos.

Judy: Gente, acho que podemos... Dei uma olhada naquela coisa de canvas no HTML5.

Frank: Canvas? Lembra que sou só o cara do design... Preciso me inteirar.

Judy: Você deve ter ouvido falar de canvas, Frank — é um novo elemento do HTML5 que cria uma região desenhável para formas em 2D, texto e imagens em bitmap.

Frank: Está parecendo com uma tag . Apenas a posicionamos na página com uma largura e altura e o browser faz o resto.

Judy: Não foi uma má comparação. Nós realmente definimos uma altura e uma largura para o canvas, mas nesse caso o que é desenhado nele é especificado com código JavaScript.

Joe: Bem, onde fica a marcação nisso tudo? Podemos dizer ao canvas no JavaScript para “por este elemento <h1> aqui”?

Judy: Não, após posicionar o canvas na página, você deixa o mundo de marcação para trás; no JavaScript nós colocamos pontos, linhas, caminhos, imagens e texto. É realmente uma API de baixo nível.

Joe: Bem, contanto que consigamos dar um jeito naqueles círculos, estou dentro. Ok, chega de conversa, vamos dar uma olhada nisso!



Frank, Judy e Joe

Como pôr um canvas em sua página

Frank estava certo. De certa forma, um canvas é como um elemento . Você adiciona um canvas assim:

O elemento canvas é um elemento HTML normal que inicia com uma tag de abertura <canvas>

O atributo width (largura) define quantos pixels horizontais ele ocupa numa página.

Da mesma forma, o atributo height (altura) define a área vertical da página que ele ocupa, aqui de 200 pixels.

```
<canvas id="lookwhatIdrew" width="600" height="200"></canvas>
```

Adicionamos uma id para que possamos identificar o canvas. Você verá como isso funciona num instante...

Aqui a largura está ajustada para 600 pixels.

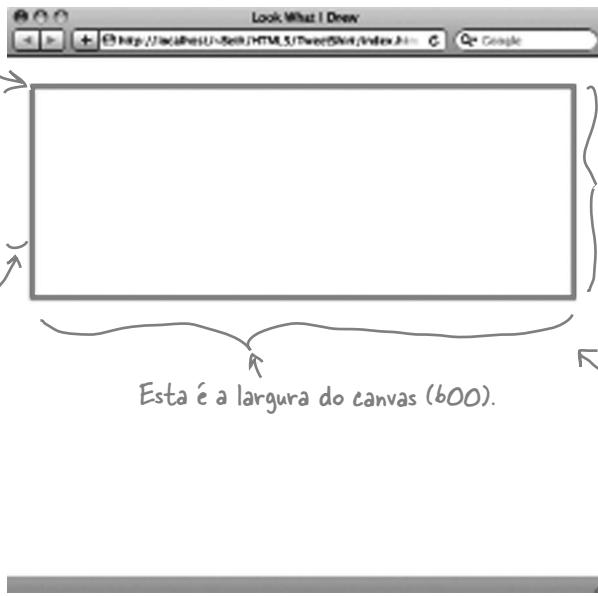
E aqui está a tag de fechamento.

E o browser aloca um espaço na página para o canvas, com a largura e altura especificadas.

Nesse caso, uma largura de 600 e uma altura de 200.

Aqui é o canto superior esquerdo do canvas. Vamos usar este ponto para mensurar todo o resto no canvas (como você já verá).

O canvas tem um pouco de espaço em torno — esta é a margem padrão do elemento body.



Esta é a altura do canvas (200, no nosso caso).

Esta é a largura do canvas (600).

Seu HTML de todo dia pode fluir pelo canvas. O canvas é como qualquer outro elemento (como uma imagem etc.).

Test drive de seu novo canvas



Está na hora de você pôr isso para funcionar, em sua própria página. Vá em frente e digite o código abaixo dentro de um novo arquivo para carregá-lo em seu browser:

```
<!doctype html>
<html lang="en">
<head>
    <title>Look What I Drew</title>
    <meta charset="utf-8">
</head>
<body>

<canvas id="lookwhatIdrew" width="600" height="200"></canvas>

</body>
</html>
```

Digite isto e
experimente.

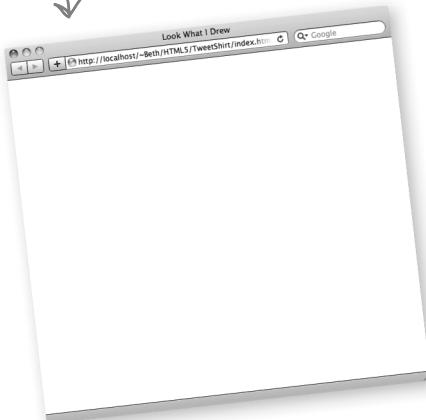
Experimentar o
quê? Minha página
está branca!

O que ela vê...

... e o que você
provavelmente está
vendo também!

Desenhamos estas linhas
para explicar como o canvas
se ajusta numa página, para
efeito ilustrativo apenas.
Elas não estão realmente lá
(a menos que você mesmo as
tenha desenhado).

Vire a página para descobrir mais...



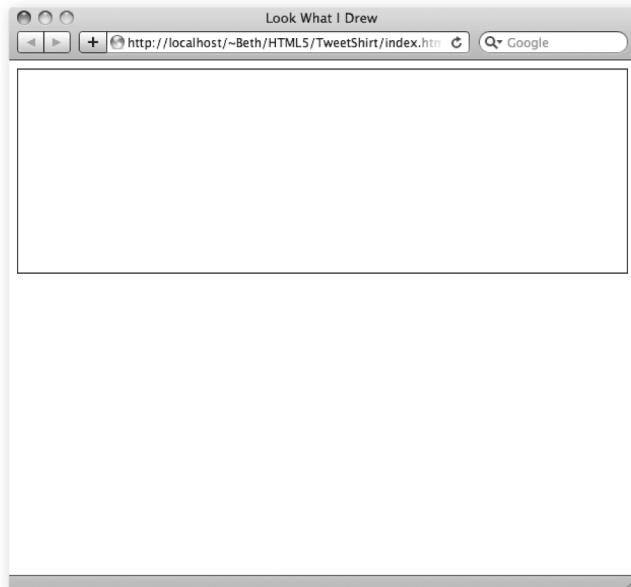
Como ver seu canvas

A menos que você tenha desenhado algo no canvas, você não o verá. É simplesmente um espaço na janela do browser para você poder desenhar. Vamos desenhar no canvas em breve, mas, por ora, o que realmente queremos é evidência de que o canvas está, de fato, em nossa página.

Há uma outra maneira de podermos ver o canvas... Se usarmos CSS para estilizar o elemento <canvas> e assim podermos ver a borda, seremos capazes de vê-lo na página. Vamos acrescentar um estilo simples, que adiciona uma borda preta de 1 pixel de largura ao canvas.

```
<!doctype html>
<html lang="en">
<head>
    <title>Look What I Drew</title>
    <meta charset="utf-8">
    <style>
        canvas {
            border: 1px solid black;
        }
    </style>
</head>
<body>
<canvas id="lookwhatIdrew" width="600" height="200"></canvas>
</body>
</html>
```

← Adicionamos um estilo ao canvas que só põe uma borda preta de 1px nele; assim podemos vê-lo na página.



não existem Perguntas Idiotas

P: Posso ter apenas um canvas por página?

R: Não, você pode ter quantos quiser (ou com quantos seu browser ou seus usuários puderem lidar). Apenas dê uma id única a cada um e poderá desenhar em todos eles em separado. Você verá num instante como usar a id do canvas.

P: O canvas é transparente?

R: Por padrão, sim, o canvas é transparente. Você pode desenhar no canvas para preenchê-lo com pixels coloridos; você vai saber como fazer isso mais para frente, ainda neste capítulo.

P: Se é transparente, isso significa que eu poderia posicioná-lo no topo de outro elemento para, digamos, desenhar algo em cima de uma imagem ou qualquer outra coisa na página, certo?

R: Isso mesmo! Essa é uma das coisas legais sobre o canvas. Ele lhe dá a habilidade para adicionar gráficos em qualquer lugar da página.

P: Posso usar CSS para ajustar a largura e a altura do canvas em vez dos atributos width e height no elemento?

R: Você pode, mas vai funcionar um pouco diferente de como espera ver. Por padrão, um elemento canvas tem 300 px de largura e 150 px de altura. Se não especificar os atributos width e height na tag canvas, é isso que irá aparecer. Se, porém, especificar um tamanho no CSS, digamos 600px por 200px, o canvas de 300 x 150 é dimensionado para caber naquele tamanho, e assim é que funciona com tudo que for desenhado no canvas. É mais ou menos como dimensionar uma imagem especificando uma nova largura e altura que seja maior ou menor que a real largura e altura da imagem. Se aumentar, você terá um pixelamento na imagem, certo?

O mesmo acontece com o canvas. Um canvas com 300px de largura que se transforma em 600px possui o mesmo número de pixels alongados em duas vezes o seu tamanho, parecendo meio amontoado. No entanto, se usar os atributos width e height no elemento, você estará ajustando as dimensões do canvas para maiores (ou menores) que 300 x 150 e tudo desenhado naquele canvas será desenhado normalmente. Portanto, recomendamos especificar a largura e a altura nos atributos da tag, e não ajustando aquelas propriedades no CSS, a menos que realmente queira dimensionar o canvas.



Você deve ter percebido que o elemento canvas não possuía nenhum conteúdo dentro dele. Se tivesse que pôr texto entre as tags, o que você acha que o browser faria quando a página fosse carregada?

<canvas>

?

</canvas>

Desenhando no Canvas

No momento, temos um canvas em branco nos encarando. Em vez de ficar sentado aqui com um estojo de blocos de letras de JavaScript, vamos em frente pondo um belo retângulo preenchido de preto em nosso canvas. Para fazer isso, temos de decidir onde será colocado e qual seu tamanho. E se pusermos na localização x=10 pixels e y=10 pixels e deixá-lo com 100 pixels de largura e altura? Funcionará para nós.

Agora, vamos dar uma checada num código que faz isso?

```

<!doctype html>
<html lang="en">
<head>
    <title>Look What I Drew</title>
    <meta charset="utf-8" />
    <style>
        canvas { border: 1px solid black; }
    </style>
    <script>
        window.onload = function() {
            var canvas = document.getElementById("tshirtCanvas");
            var context = canvas.getContext("2d");
            context.fillRect(10, 10, 100, 100);
        };
    </script>
</head>
<body>
    <canvas width="600" height="200" id="tshirtCanvas"></canvas>
</body>
</html>

```



Vamos começar pelo nosso HTML5 padrão.

Manteremos nossa borda CSS por ora.

Aqui é o nosso handler onload; começaremos a desenhar após o carregamento completo da página.

Para desenhar no canvas, precisamos de uma referência a ele. Vamos usar getElementById para obtê-lo do DOM.

Mmm, isto é interessante, aparentemente precisamos de um contexto "2d" do canvas para começar a desenhar de fato...

Estamos usando o contexto 2d para desenhar um retângulo preenchido no canvas.

Estes números são a posição x, y do retângulo no canvas.

E também temos a largura e a altura (em pixels).

Interessante também que um método rectangle fill não leva uma cor de preenchimento... mais disso num segundo.

Ah, e não podemos nos esquecer de nosso elemento canvas. Estamos especificando um canvas que tem 600 pixels de largura e 200 pixels de altura; e possui uma id de "tshirtCanvas".

Test drive de um pequeno Canvas...

Vá em frente e digite este código (ou pegue-o em <http://wickedlysmart.com/hfhtml5>), carregando-o em seu browser. Imaginamos que esteja utilizando um browser moderno, portanto, você deve estar vendo algo assim:

Aqui está nosso retângulo 100×100 posicionado em $10, 10$ no canvas.



Vendo melhor o código

Esse foi um ótimo ensaio, mas vamos mergulhar mais fundo:

- Em nossa marcação, definimos um canvas e lhe demos uma id, usando a tag <canvas>. A primeira coisa que precisa fazer para desenhar naquele canvas é colocar um handle no objeto canvas do DOM. Como sempre, faremos isso com o método getElementById:

```
var canvas = document.getElementById("tshirtCanvas");
```

- 2 Com uma referência ao elemento canvas, designado para a variável canvas, temos agora que seguir alguns “protocolos” antes de desenhar. Precisamos pedir ao canvas para nos dar um contexto onde desenhar. Neste caso, queremos principalmente um contexto 2D. O contexto devolvido pelo canvas está designado para a variável context:

```
var context = canvas.getContext("2d");
```

Isso é meio que um protocolo que temos de seguir antes de podermos iniciar nosso desenho no canvas.

- 3 Agora, com o contexto em mãos, podemos usá-lo para desenhar no canvas, o que faremos ao chamar o método fillRect. Este método cria uma retângulo começando na posição x, y de 10, 10 e isso significa 100 pixels de largura e de altura.

Perceba que estamos chamando o método fillRect no contexto, não no canvas em si.

```
context.fillRect(10, 10, 100, 100);
```

Experimente isso e deverá aparecer um retângulo preto. Tente mudar os valores x,y e largura, altura e veja o que acontece.



Consegue imaginar alguma forma de usar um elemento canvas, se seu browser suportá-lo, e, se não, de apenas mostrar uma mensagem tipo “Ei, você, sim, você, faça um upgrade em seu browser!!”?

não existem Perguntas Idiotas

P: Como o canvas saberá como fazer o retângulo preto?

R: Preto é a cor de preenchimento padrão para um canvas. Claro que você pode mudar isso usando uma propriedade `fillStyle`, como vamos mostrar em instantes.

P: E se eu quisesse um contorno em forma de retângulo, não um preenchido?

R: Para ter apenas o contorno do retângulo, você deverá usar a função `strokeRect` em vez de `fillRect`. Você aprenderá mais a respeito de traços mais tarde, ainda neste capítulo.

P: O que é um contexto 2d, e por que não posso apenas desenhar direto no canvas?

R: O canvas é a área gráfica mostrada na página. O contexto é um objeto associado com o canvas, que define um conjunto de propriedades e métodos que você pode usar para desenhar no canvas. Você pode até salvar o estado do contexto e restaurá-lo mais tarde, o que é bem útil às vezes. Você verá muitas dessas propriedades e métodos até o fim do capítulo.

O canvas foi feito para suportar mais do que uma interface, 2d, 3d e outras que ainda nem temos a menor ideia. Utilizando um contexto, podemos trabalhar com diferentes interfaces dentro do mesmo elemento, canvas. Você não pode desenhar direto no canvas, porque precisa especificar qual interface está usando ao escolher um contexto.

P: Isso significa que existe um contexto "3d" também?

R: Ainda não. Existem algumas normas concorrentes que vêm surgindo para isso, mas nada que pareça muito promissor ainda. Fique ligado nisso. Enquanto isso, dê uma olhada no WebGL e nas bibliotecas que o usam, como SpiderGL, SceneJS e three.js.

Codificação Séria

Está matutando como você pode detectar se seu browser suporta ou não canvas em código?

É claro que pode, e devemos salientar que, até agora, apenas presumimos que nosso browser suporta canvas. Em qualquer produção de códigos, você deve testar para ter certeza de que tem suporte.

Tudo o que tem de fazer é testar para ver se o método `getContext` existe em seu objeto canvas (aquele que você recebe do `getElementById`):

Primeiro, pegamos uma referência ao elemento canvas na página.

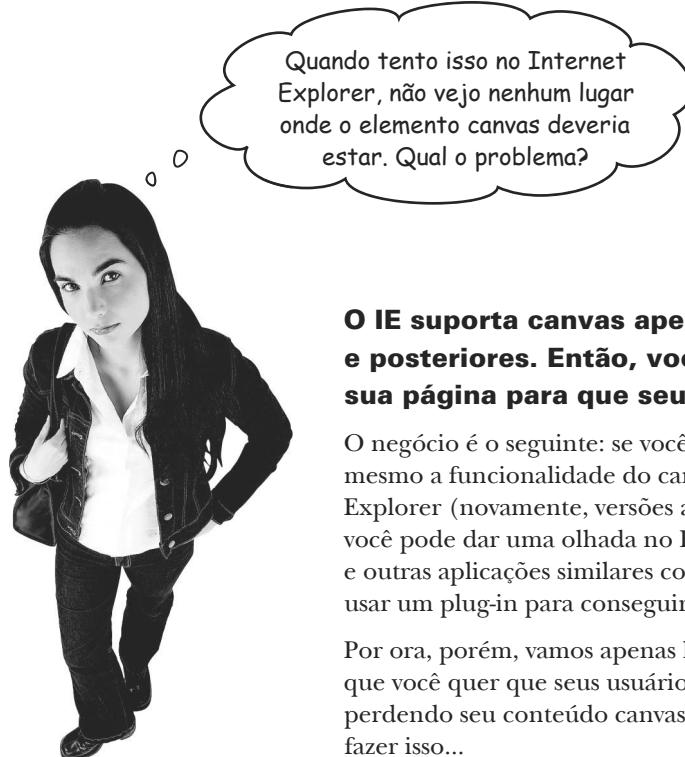
```
var canvas =
  document.getElementById("tshirtCanvas");
if (canvas.getContext) {
  // you have canvas
} else {
  // sorry no canvas API support
}
```

Então, verificamos a existência do método `getContext`. Note que não o estamos chamando, apenas vendo se possui algum valor.

Se quiser testar o suporte a canvas sem ainda ter um canvas em sua marcação, você pode criar um elemento canvas imediatamente, usando todas as técnicas que já sabe, assim:

```
var canvas =
  document.
createElement("canvas");
```

Certifique-se de verificar o apêndice para informações sobre uma biblioteca de fonte aberta que você pode usar para testar todas as funcionalidades em HTML5 de forma consistente.



Quando tento isso no Internet Explorer, não vejo nenhum lugar onde o elemento canvas deveria estar. Qual o problema?

O IE suporta canvas apenas nas versões 9 e posteriores. Então, você deve codificar sua página para que seus usuários saibam.

O negócio é o seguinte: se você precisa suportar mesmo a funcionalidade do canvas no Internet Explorer (novamente, versões anteriores à 9), então você pode dar uma olhada no Explorer Canvas Project e outras aplicações similares como uma maneira de usar um plug-in para conseguir tal funcionalidade.

Por ora, porém, vamos apenas levar em consideração que você quer que seus usuários saibam que eles estão perdendo seu conteúdo canvas incrível. Vejamos como fazer isso...

E talvez você possa sugerir a eles que atualizem para IE9!

Fracassando graciosamente

A verdade é que, por aí, em algum lugar, em outro espaço-tempo, um usuário vai visitar seu site sem ter suporte para o elemento canvas. Você gostaria de lhe mandar uma mensagem simpática, dizendo que ele deveria se atualizar? Eis como:

```

    ↘ Apenas o seu típico e cotidiano elemento canvas.
<canvas id="awesomecontent">
    Hey you, yes YOU, upgrade your browser!!
</canvas>
    ↗ Ponha a mensagem que quiser ver no
        display para seus usuários que não têm
        um browser compatível com canvas.

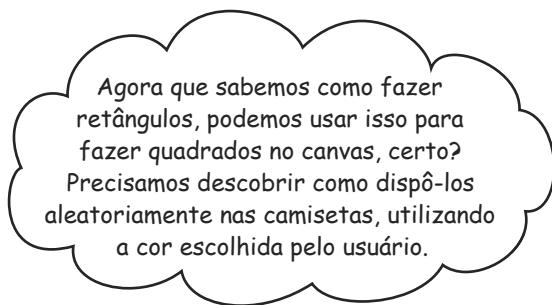
```

Como isso funciona? Bem, todas as vezes em que o browser ver um elemento que não reconhece, ele mostrará qualquer texto contido dentro dele como um comportamento padrão. Então, quando browsers não-compatíveis veem o elemento `<canvas>`, eles apenas mostram “Ei você, sim, VOCÊ, atualize seu browser!!” Browsers compatíveis, por outro lado, ignoram, convenientemente, qualquer texto entre as tags canvas e não o mostram.

↘ Obrigado ao pessoal da padronização HTML5 por ter tornado tudo mais fácil!



Como vocês já sabem, outra forma de lidar com browser que não suporta canvas é usar o JavaScript para detectar se ele conhece o elemento. Isso lhe dá maior flexibilidade para oferecer a seus usuários uma experiência diferente, no caso de seus browsers não darem suporte a ele; por exemplo, você poderia redirecioná-los a uma página diferente ou mostrar uma imagem no lugar.



Frank: Claro, mas também precisamos da interface de usuário para que ele especifique tudo isso. Quero dizer, já temos o mock-up, mas precisamos implementá-lo.

Judy: Você está certo, Frank. Não há por que continuar sem a interface.

Joe: Não é só uma questão de HTML?

Frank: Sim, acho que sim, mas, dado que estamos tentando fazer isso tudo no lado cliente, como vai funcionar? Por exemplo, onde é que o formulário poderá ser submetido? Não estou bem certo de que entendi como isso tudo se encaixa.

Joe: Frank, podemos apenas chamar a função JavaScript quando o usuário clicar no botão de pré-visualização e, então, mostrar o design da camisa no canvas.

Frank: Faz sentido, mas como acessaremos os valores do formulário, se está tudo no lado cliente?

Judy: Da mesma maneira que sempre acessamos o DOM; podemos usar `document.getElementById` para conseguir os valores do formulário. Você já fez isso antes.

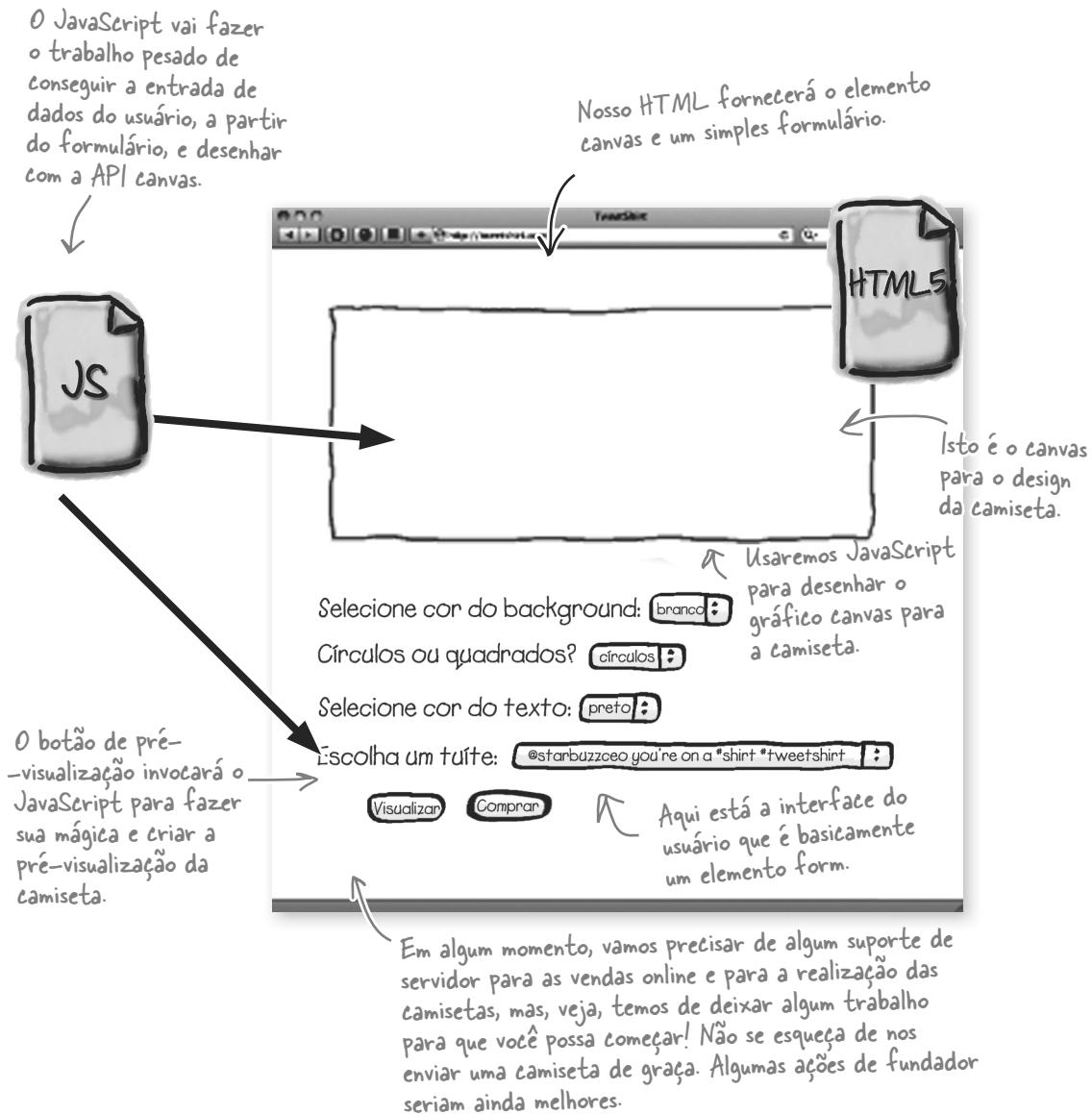
Frank: Gente, já me perdi lá atrás...

Joe: Tudo bem, vamos juntos nessa. Vamos começar com uma visão geral.

TweetShirt: Visão Geral

Antes de mergulharmos de cabeça num grande trabalho de implementação, vamos recuar um pouco e olhar o todo. Vamos construir este aplicativo web fora de um *elemento canvas*, junto com alguns elementos do formulário que agem como a interface do usuário, e, por trás disso, vamos fazer tudo acontecer com o JavaScript e a *API canvas*.

Veja como vai ficar:



Sinta-se como o Navegador



Abaixo, você encontrará o formulário para a interface da camiseta. Seu trabalho é fingir ser o browser e gerar a interface.

Depois que terminar,
compare-a com a da página
anterior para checar se fez
tudo corretamente.

```
<form>


<label for="backgroundColor">Select background color:</label>
    <select id="backgroundColor">
        <option value="white" selected="selected">White</option>
        <option value="black">Black</option>
    </select>



<label for="shape">Circles or squares?</label>
    <select id="shape">
        <option value="none" selected="selected">Neither</option>
        <option value="circles">Circles</option>
        <option value="squares">Squares</option>
    </select>



<label for="foregroundColor">Select text color:</label>
    <select id="foregroundColor">
        <option value="black" selected="selected">Black</option>
        <option value="white">White</option>
    </select>



<label for="tweets">Pick a tweet:</label>
    <select id="tweets">
    </select>



<input type="button" id="previewButton" value="Preview">


</form>
```

Gere sua interface aqui.
Desenhe a página como ela vai
aparecer com os elementos de
formulário à esquerda.

Imagine que usou a
interface para conseguir
valores para sua camiseta.

Sinta-se como o Navegador, novamente

Agora que possui uma interface,
execute essas afirmações
JavaScript e escreva o
valor para cada elemento
da interface. Verifique
sua resposta com nossa
solução no fim do capítulo.



```
var selectObj = document.getElementById("backgroundColor");  
var index = selectObj.selectedIndex;  
var bgColor = selectObj[index].value;  
  
.....  
  
var selectObj = document.getElementById("shape");  
var index = selectObj.selectedIndex;  
var shape = selectObj[index].value;  
  
.....  
  
var selectObj = document.getElementById("foregroundColor");  
var index = selectObj.selectedIndex;  
var fgColor = selectObj[index].value;
```

Primeiro, vamos organizar o HTML

Chega de falação! Vamos construir esse negócio. Antes de fazer qualquer outra coisa, precisamos apenas de uma simples página HTML. Atualize seu arquivo index.html para que se pareça com isso:

```
<!doctype html>           ↗ Um belo HTML5...
<html lang="en">
<head>                   ↗ arquivo compatível, sim!
    <title>TweetShirt</title>
    <meta charset="utf-8" />
    <style>
        canvas {border: 1px solid black;}
    </style>
    <script src="tweetshirt.js"></script>           ↗ Perceba que mudamos o
                                                        título para "TweetShirt"
</head>                   ↗ Vamos pôr todo nosso
<body>                   código JavaScript
    <h1>TweetShirt</h1>                         num arquivo separado,
    <canvas width="600" height="200" id="tshirtCanvas">       de maneira que fique
        <p>Please upgrade your browser to use TweetShirt!</p>   um pouco mais fácil
    </canvas>                                         de gerenciar.
    <form>           ↗ Aqui está o canvas!
        </form>           ↗ E este é o formulário que irá
                           amarrar todos os controles para o
                           aplicativo tweetshirt. Chegaremos
                           lá na próxima página...
    </body>
</html>
```



O que mais você precisa saber para substituir a borda do CSS em seu canvas por uma borda desenhada no canvas usando JavaScript? E, se puder, qual seu método preferido (CSS versus JavaScript) e por quê?

Agora, vamos adicionar o <form>

Ok, agora, vamos adicionar a interface do usuário para que possamos começar a escrever algum código que crie camisetas. Você já viu esse código antes, mas acrescentamos algumas anotações só para deixar tudo bem claro; enquanto digita o código, certifique-se de verificar nossas anotações:

```

<form>
  <p>
    <label for="backgroundColor">Select background color:</label>
    <select id="backgroundColor">
      <option value="white" selected="selected">White</option>
      <option value="black">Black</option>
    </select>
  </p>
  <p>
    <label for="shape">Circles or squares?</label>
    <select id="shape">
      <option value="none" selected="selected">Neither</option>
      <option value="circles">Circles</option>
      <option value="squares">Squares</option>
    </select>
  </p>
  <p>
    <label for="foregroundColor">Select text color:</label>
    <select id="foregroundColor">
      <option value="black" selected="selected">Black</option>
      <option value="white">White</option>
    </select>
  </p>
  <p>
    <label for="tweets">Pick a tweet:</label>
    <select id="tweets">
    </select>
  </p>
  <p>
    <input type="button" id="previewButton" value="Preview">
  </p>
</form>

```

Todo este código vai entre as tags <form> que você criou na página anterior.

Aqui é onde o usuário seleciona a cor de background para o design da camiseta com o tuite. As escolhas são preta ou branca. Fique à vontade para adicionar suas próprias cores.

Estamos usando outro controle de seleção aqui para escolher entre círculos e quadrados para personalizar o design. O usuário também pode escolher nenhum deles (o que deverá resultar num background plano).

Outra seleção para escolher a cor do texto. Novamente, apenas preto ou branco.

Aqui é onde todos os tuítes vão. Então, por que está vazio? Ah, vamos complementar este detalhe mais tarde (dica: precisamos deixá-los ligados ao Twitter, afinal de contas, isto é um aplicativo web, certo?!).

Se estiver acostumado a formulários, talvez tenha percebido que este não possui um atributo action (o que significa que o botão não fará nada quando clicado). Vamos falar sobre tudo isto num instante...

E, por último, um botão para visualizar a camisa.

Hora de ficar computacional com JavaScript

Marcação é ótimo, mas é o JavaScript que faz acontecer o aplicativo web TweetShirt. Vamos inserir um pouco de código em `tweetshirt.js`. No momento, queremos dar passinhos de bebê e apenas pôr quadrados aleatórios na camiseta. Antes mesmo, porém, de chegarmos lá, precisamos habilitar nosso botão de visualização para que ele chame uma função JavaScript quando clicado.

Crie um arquivo `tweetshirt.js` e adicione isto.

```
window.onload = function() {
    var button = document.getElementById("previewButton");
    button.onclick = previewHandler;
};
```

Comece pelo elemento `previewButton`.

↑
Adicione um click handler para este botão, de forma que, quando ele for clicado (ou tocado, no caso de um dispositivo móvel), a função `previewHandler` seja chamada.

Então, agora, quando o botão de visualização for clicado, a função `previewHandler` será chamada, e esta é nossa chance de atualizar o canvas para representar a camisa que o usuário está desenhando. Vamos começar a escrever `previewHandler`:

```
function previewHandler() {
```

Comece pelo elemento `canvas` e pergunte por seu contexto de desenho em 2d.

```
    var canvas = document.getElementById("tshirtCanvas");
    var context = canvas.getContext("2d");
```

Agora precisamos ver qual forma você escolheu na interface. Primeiro, pegamos o elemento com a id de "shape".

```
    var selectObj = document.getElementById("shape");
    var index = selectObj.selectedIndex;
    var shape = selectObj[index].value;
```

Então, descobrimos qual item será selecionado (quadrados ou círculos), obtendo o index do item selecionado e designando seu valor à variável `shape`.

```
    if (shape == "squares") {
```

E se o valor de `shape` é "squares" (quadrados), então precisaremos desenhar alguns quadrados. Que tal uns 20 deles?

```
        for (var squares = 0; squares < 20; squares++) {
            drawSquare(canvas, context);
        }
    }
```

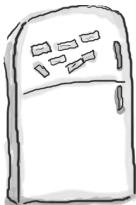
↑
Para desenhar cada quadrado, estamos nos baseando numa nova função chamada `drawSquare`, a qual teremos que escrever. Entenda que estamos passando tanto o canvas quanto o contexto para `drawSquare`. Você já vai ver como os utilizamos.

não existem Perguntas Idiotas

P: Como o selectedIndex funciona?

R: A propriedade selectedIndex de um controle de formulário de seleção retorna o número da opção que escolheu a partir do menu "pulldown". Toda a lista de opções é transformada num array e cada opção fica no array, em ordem*. Portanto, digamos que você tenha uma lista de seleção com essas opções: "pizza", "doughnut" e "granola bar". Se você selecionou "doughnut", o selectedIndex seria de 1 (lembre-se de que os arrays do JavaScript começam no 0).

Agora, você provavelmente não quer apenas o index; quer o valor da opção daquele index (em nosso caso, "doughnut"). Para obter o valor da opção, você primeiro usa o index para obter o elemento do array, que retorna um *objeto* option. Para conseguir o valor daquele objeto, você usa a propriedade *value*, a qual retorna a string no atributo value da opção.



Pseudo-ímãs de Geladeira

Use seus poderes de código pseudomágicos para arrumar o pseudocódigo para a função drawSquare. Esta função pega um canvas e um contexto e desenha um quadrado aleatoriamente dimensionado no canvas. Verifique sua resposta no fim do capítulo antes de olhar para trás.

```
function drawSquare (
```

context

Fizemos esse para você.

```
}
```

canvas

desenhe um quadrado na
posição x, com a largura w

Seus ímãs vão aqui!

calcule uma posição y aleatória
para o quadrado dentro do canvas

"lightblue" (azul claro) é
a cor dos quadrados em
nossa esboço do design.

ajuste fillStyle
para "lightblue"

calcule uma largura
aleatória para o quadrado

calcule uma posição x aleatória
para o quadrado dentro do canvas

Escrevendo a função drawSquare

Agora que você fez todo o trabalho árduo de descobrir o pseudocódigo, vamos usar o que já sabemos para escrever o drawSquare:

Aqui, precisamos de largura e posições x e y aleatórias para o quadrado.

```
function drawSquare(canvas, context) {
    var w = Math.floor(Math.random() * 40);
    var x = Math.floor(Math.random() * canvas.width);
    var y = Math.floor(Math.random() * canvas.height);
    context.fillStyle = "lightblue";
    context.fillRect(x, y, w, w);
}
```

Aqui está nossa função, que tem dois parâmetros: canvas e context.

Estamos usando Math.random() para criar números aleatórios para a largura e a posição x, y do quadrado. Falaremos mais disso...

Escolhemos 40 como o maior tamanho para que os quadrados não fiquem muito grandes.

As coordenadas x e y são baseadas na largura e na altura do canvas. Escolhemos um número aleatório entre 0 e a largura e a altura, respectivamente.

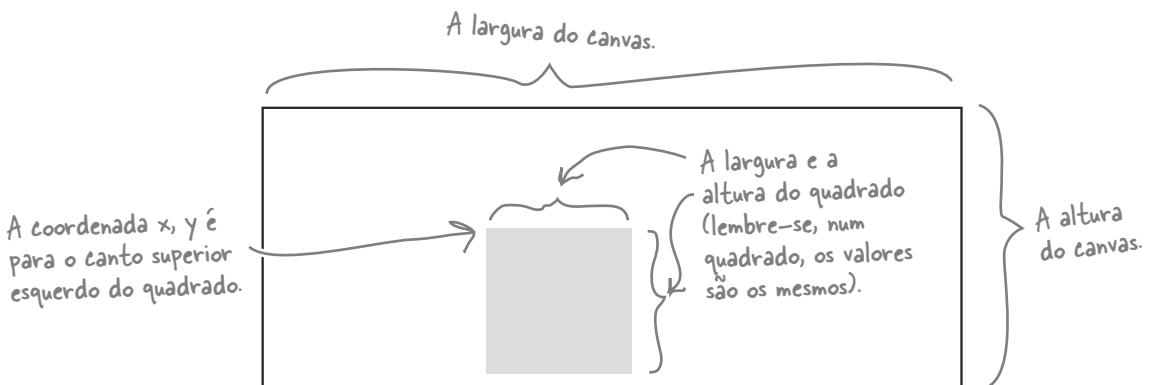
Vamos fazer os quadrados com um belo azul claro, utilizando o método fillStyle; vamos ver este método mais a fundo num segundo...

E, finalmente, desenhamos o quadrado real com fillRect.

Use a Cabeça! HTML com CSS é XHTML tem um bom capítulo sobre cores, se precisar refrescar a memória.

Como descobrimos quais números deveríamos multiplicar por cada valor Math.random para conseguirmos a largura e as posições x, y de nosso quadrado? No caso da largura do retângulo, escolhemos 40 porque é um bom tamanho, pequeno em relação ao tamanho do canvas. Pelo fato de ser um quadrado, usamos o mesmo valor para a altura. Escolhemos a largura e altura do canvas como a base para a escolha dos valores x e y. Assim nosso quadrado permanece dentro dos limites do canvas.

Sinta-se à vontade para especificar outro valor que não seja 40 em seu próprio código!

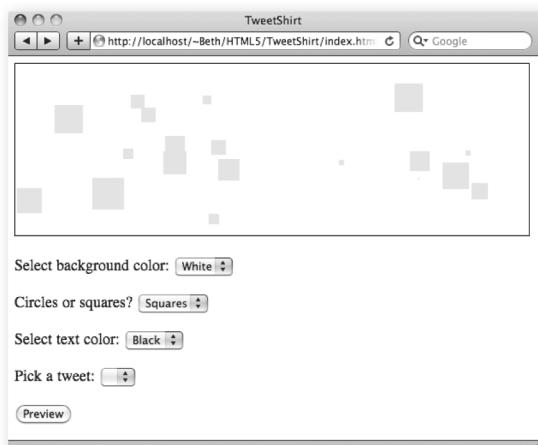


Hora do test drive!

Ok, depois de tanto digitar, vamos testar todo esse código. Vá em frente e abra seu arquivo index.html do TweetShirt em seu browser. Pressione pré-visualizar e você deverá ver quadrados azuis aleatórios.

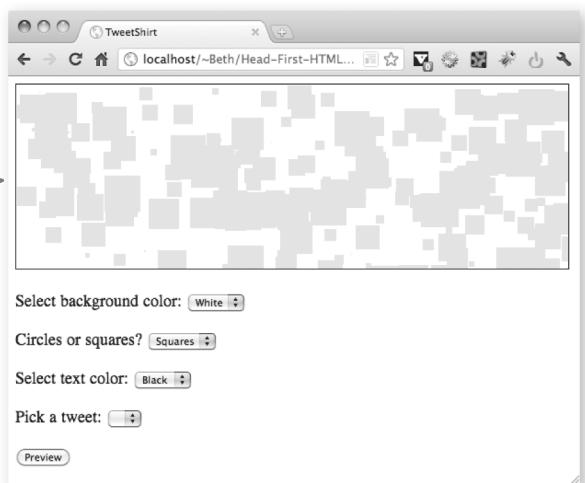
Eis o que vemos:

Legal! Exatamente como queríamos!



Mmm, espere um segundo. Se continuar pressionando o botão de pré-visualização, você terá MUITOS quadrados. Isso não está certo!

Ele está certo.
Temos um pequeno
problema. Pressione
seu botão de pré-
visualização algumas
vezes e verá algo
parecido com isso.



Por que estamos vendo os antigos e os novos quadrados quando pré-visualizamos?

Este é, na verdade, meio que um efeito legal... mas não é o que queríamos. Queremos que os novos quadrados *substituam* os quadrados antigos, toda vez que pressionarmos a tecla preview (assim como também vamos querer que o novo tuíte substitua o velho, quando fizermos os tuítes funcionarem).

A chave aqui é lembrar que estamos pintando com pixels no canvas. Quando pressiona a pré-visualização, você tem o canvas e os quadrados do desenho nele. O que quer que já esteja no canvas será exatamente pintado em cima dos novos pixels!

Não se preocupe. Você já sabe tudo o que precisa saber para consertar isso agora mesmo. Eis o que vamos fazer:

- ① Pegue a cor selecionada do background do objeto de seleção "backgroundColor".
- ② Preencha a cor do background do canvas, usando `fillStyle` e `fillRect`, todas as vezes antes de começarmos a desenhar quadrados.



Aponte o seu lápis

Para nos certificarmos de que apenas veremos novos quadrados no canvas, cada vez que clicarmos em pré-visualização, precisamos preencher o background do canvas com a cor do background que o usuário selecionou no menu de seleção "backgroundColor". Primeiro, vamos implementar uma função para preencher o canvas com aquela cor. Preencha os espaços em branco abaixo para completar o código. Confira sua resposta com nossa solução ao fim do capítulo, antes de continuar.

```
function fillBackgroundColor(canvas, context) {  
    var selectObj = document.getElementById("_____");  
    var index = selectObj.selectedIndex;  
    var bgColor = selectObj.options[index].value;  
    context.fillStyle = _____;  
    context.fillRect(0, 0, _____, _____);  
}
```

Dica: o que você obterá da opção selecionada será uma string color que poderá ser usada assim como você usou "lightblue" para os quadrados.

Dica: queremos preencher o canvas **INTEIRO** com a cor!

Adicione a chamada para fillBackgroundColor

Você já tem prontinha a função `fillBackgroundColor`; agora, precisamos apenas nos certificar de que a chamaremos a partir de `previewHandler`. Vamos acrescentar isso como a primeira coisa a ser feita, pois assim conseguiremos um background limpo e bonito, antes de começarmos a acrescentar qualquer outra coisa ao canvas.

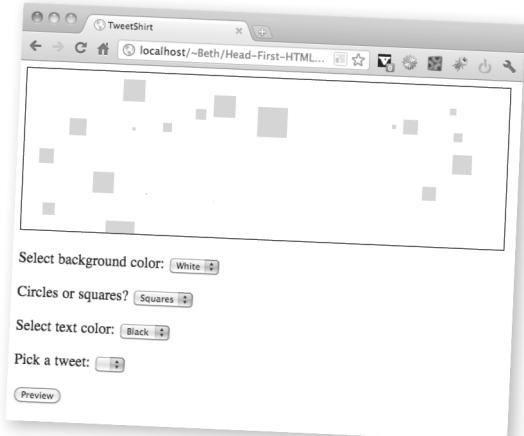
```
function previewHandler() {
    var canvas = document.getElementById("tshirtCanvas");
    var context = canvas.getContext("2d");
    fillBackgroundColor(canvas, context); ← Estamos adicionando a chamada
                                            para fillBackgroundColor antes
                                            de desenharmos nossos quadrados.
                                            Isso irá cobrir os desenhos
                                            anteriores e nos dará um
                                            background limpo para os novos.

    var selectObj = document.getElementById("shape");
    var index = selectObj.selectedIndex;
    var shape = selectObj[index].value;

    if (shape == "squares") {
        for (var squares = 0; squares < 20; squares++) {
            drawSquare(canvas, context);
        }
    }
}
```


Outro test drive rápido para ter certeza de que nossa função `fillBackgroundColor` funciona...

Adicione o novo código ao seu arquivo `tweetshirt.js`, recarregue seu browser, selecione uma cor de background, selecione quadrados e clique em preview. Clique novamente. Desta vez, você deverá ver apenas novos quadrados cada vez que clicar na pré-visualização.



↑ Agora, temos apenas os novos quadrados para cada pré-visualização.



Conte os quadrados em algumas visualizações diferentes. Você sempre vê menos de 20 quadrados? Talvez.

Por que isso acontece? O que você poderia fazer para resolver este problema? (Afinal de contas, você não quer passar a perna em seus clientes oferecendo só 20 quadrados, quer?)



JavaScript de Perto

Vamos olhar mais de perto o `fillStyle`, já que é a primeira vez que o vê. `fillStyle` é uma propriedade do contexto, que mantém a cor para qualquer desenho que fizer no canvas.

Igual ao `fillRect`, `fillStyle` é algo que controlamos pelo contexto.

Mas, ao contrário de `fillRect`, `fillStyle` é uma propriedade, não um método. Portanto, nós o ajustamos em vez de chamá-lo.

```
context.fillStyle = "lightblue";
```

E o que ajustamos é uma cor. Você pode usar os mesmos formatos de cor que usar no CSS. Então você pode usar nomes de cor, como `lightblue`, ou valores como `#ccccff` ou `rgb(0, 173, 239)`. Experimente!

não existem perguntas idiotas

Perceba que, ao contrário do CSS, você deve por entre aspas o valor, se não for usar uma variável.

P: Pensei que ajustaríamos a cor de `background` dos quadrados e do canvas passando um valor de cor para `fillRect`. Não entendi direito como o `fillStyle` funciona. Como isso afeta o que o `fillRect` faz?

R: Boa pergunta. Isso é um pouco diferente de como você talvez pense as coisas. Lembre-se, que o contexto é um objeto que controla o acesso ao canvas. O que deve ser feito com `fillStyle` e `fillRect` é, primeiro, ajustar uma propriedade que diz ao canvas, “o que quer que desenhe depois, deverá ser nessa cor”. Portanto, qualquer coisa que você preencher com cor (como com `fillRect`), após ajustar o `fillStyle`, usará aquela cor até que você mude-a novamente, ajustando o `fillStyle` a uma cor diferente.

P: Por que a cor precisa estar entre aspas, quando os valores da propriedade no CSS

não precisam? Por exemplo, eu não uso as aspas quando estou ajustando a cor do `background` de um elemento.

R: Bem, CSS é uma linguagem diferente do JavaScript, e o CSS não precisa de aspas. Se você, porém, não puser a cor entre aspas, o JavaScript achará que o nome da cor é uma variável em vez de uma string, e tentará usar o valor da variável para a cor.

Digamos que tenha uma variável:

```
fgColor = "black". Você poderia escrever context.fillStyle = fgColor, e funcionaria porque o valor de fgColor é "black" (preto).
```

No entanto, `context.`

```
fillStyle = black
```

não vai funcionar porque `black` (preto) não é uma variável (a menos que você a configure assim, o que será um pouco confuso). Você saberá que cometeu este engano, pois receberá um erro de JavaScript que diz algo como “Não pode achar variável”.

`black`” (não se preocupe, todos já cometemos esse engano, pelo menos uma vez na vida).

P: Ok, desisto. Por que estávamos vendo menos de 20 quadrados, às vezes?

R: O `x`, `y` e a largura dos quadrados eram todos aleatórios. Alguns quadrados podem obliterar outros. Um quadrado pode ter uma posição `x`, `y` de 599, 199. Neste caso, só um pixel dele poderá ser visto (porque o resto do quadrado estará fora do canvas). Alguns quadrados podem ter 1 pixel de largura, e alguns até mesmo 0 pixels, pois o método `Math.random` pode retornar 0. Você pode ainda gerar dois quadrados de exatamente mesmo tamanho e localização.

Para este aplicativo, porém, tudo faz parte da aleatoriedade, então achamos que está tudo bem. Para outro aplicativo, talvez precisássemos garantir que isso não acontecesse.

Enquanto isso, lá na TweetShirt.com...



Jim: Estou impressionado com quão pouco código foi necessário. Pense só se tivéssemos feito tudo da velha maneira no lado servidor. Ainda estaríamos ligando nosso servidor.

Frank: E me parece que está fácil para acertarmos as coisas com os círculos do design também; afinal, eles são exatamente como os quadrados.

Jim: Concordo. Onde está a Judy? Ela provavelmente já sabe a API para os círculos. Novamente, é provável que seja apenas uma questão de chamar o método `fillCircle`.

Frank: Também acho que sim! Quem precisa da Judy? Nós resolveremos isso!



E, algumas horas depois...

Frank: Não sei o que está acontecendo. Chequei tudo duas vezes, mas, não importa o que eu faça, quando chamo `fillCircle`, nada aparece no canvas.

Judy: Bem, mostre-me seu método `fillCircle`.

Frank: O que quer dizer por meu método? Não tenho um. Estou usando o método na API do canvas, diretamente.

Judy: A API do canvas não tem um método `fillCircle`.

Frank: Mmm, eu *achei* que tivesse porque tínhamos um `fillRect`...

Judy: Sim, você sabe bem o que o achismo pode fazer pela gente, né?! Venha, abra um browser — sempre é possível achar a API em: <http://dev.w3.org/html5/2dcontext/>.

... Ainda mais, desenhar um círculo é um pouco mais complicado do que chamar um método só. Você precisa aprender sobre caminhos e arcos primeiro.

Jim, entrando: Judy, o Frank te contou como acertamos na mosca com o negócio dos círculos?

Frank: É... Jim, *enoughway ithway ethay irclehay!*

↑ Recomendamos os serviços de tradução de piglatin.bavetta.com

Desenhando com Nerds

Antes de mergulharmos nos círculos, precisamos falar a respeito das trajetórias e dos arcos. Vamos começar pelos caminhos e desenhar alguns triângulos. Se quisermos desenhar um triângulo no canvas, não haverá método `fillTriangle`, mas podemos criar um triângulo mesmo assim, criando, primeiro, um *caminho* (*path*) no formato de um e, então, *contornando-o* para desenhar um triângulo no canvas.

O que isso significa? Bem, digamos que você quisesse ser realmente cuidadoso ao desenhar no canvas. Pegaria um lápis e traçaria suavemente o formato (vamos apenas chamá-lo de caminho) no canvas. O traço é tão fraco que só você pode vê-lo. Em seguida, após estar satisfeito com seu caminho, você pegaria uma caneta (com a grossura e cor de sua escolha) e a usaria para contornar o caminho para que todos pudessem ver seu triângulo (ou qualquer que tenha sido a forma traçada pelo lápis).

É assim que funciona o desenhar de formas com linhas. Vamos desenhar um triângulo e ver como é isso:

Usamos o método `beginPath` para dizer ao canvas que estamos iniciando um novo caminho.

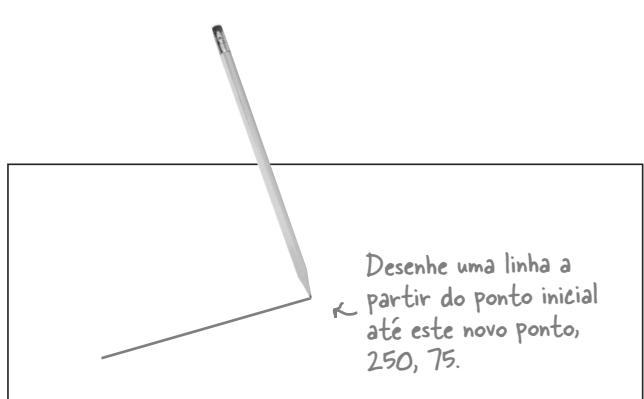
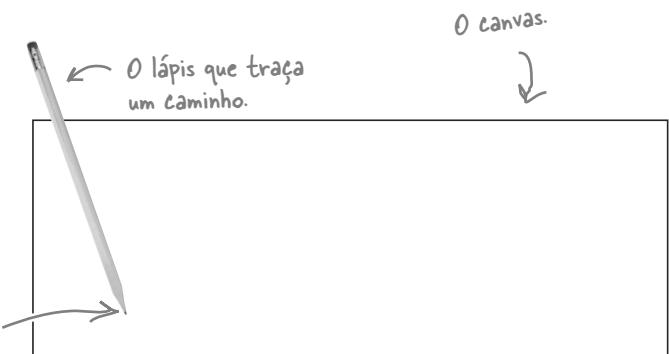
```
context.beginPath();
context.moveTo(100, 150);
```

Usamos o método `moveTo` para mover o "lápis" a um ponto específico no canvas. Você pode imaginar que o lápis está sendo pousado neste ponto.

O método `lineTo` traça um caminho a partir da atual localização do lápis até outro ponto no canvas.

```
context.lineTo(250, 75);
```

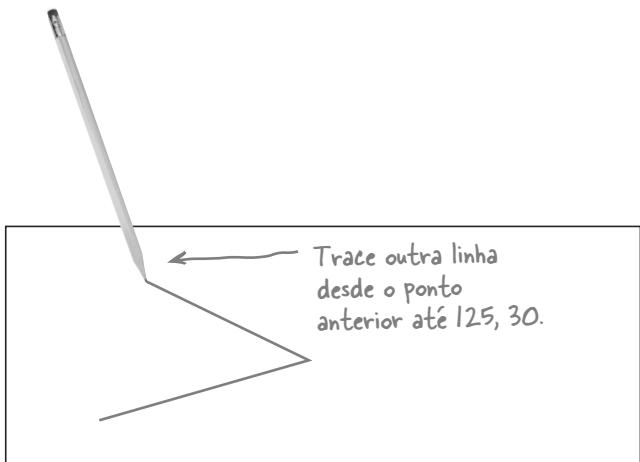
O lápis estava em $x=100, y=150$, e aqui estamos estendendo o caminho de lá até o ponto $x=250, y=75$.



Já temos o primeiro lado do triângulo; agora, precisamos de mais dois. Vamos usar novamente o `lineTo` para o segundo lado:

```
context.lineTo(125, 30);
```

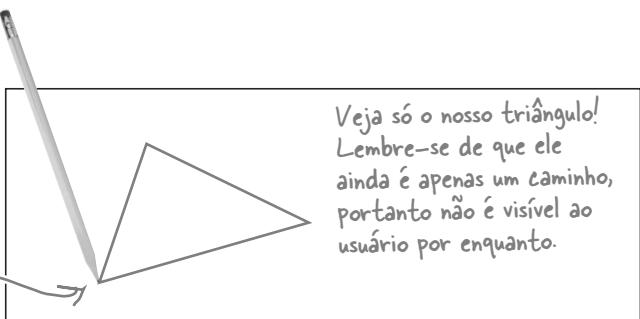
Aqui estamos traçando a partir da atual posição do lápis ($250, 75$) em direção à nova posição, $x=125$, $y=30$. Assim, completámos a nossa segunda linha.



Estamos chegando lá! Tudo o que precisamos fazer agora é traçar mais uma linha para finalizar o triângulo. Para isso, vamos apenas fechar o caminho com o método `closePath`.

```
context.closePath();
```

O método `closePath` conecta o ponto inicial do caminho ($100, 150$) ao último ponto do atual caminho ($125, 30$).



Exercício

Quer dizer que você tem um caminho! E agora?

Você utiliza o caminho para desenhar linhas e preencher sua forma com cor, é claro! Vá em frente e crie uma simples página HTML5 com um elemento canvas e digite todo o código feito até agora. Faça um teste também.

```
context.beginPath();
context.moveTo(100, 150);
context.lineTo(250, 75);
context.lineTo(125, 30);
context.closePath();

context.lineWidth = 5;

context.stroke();

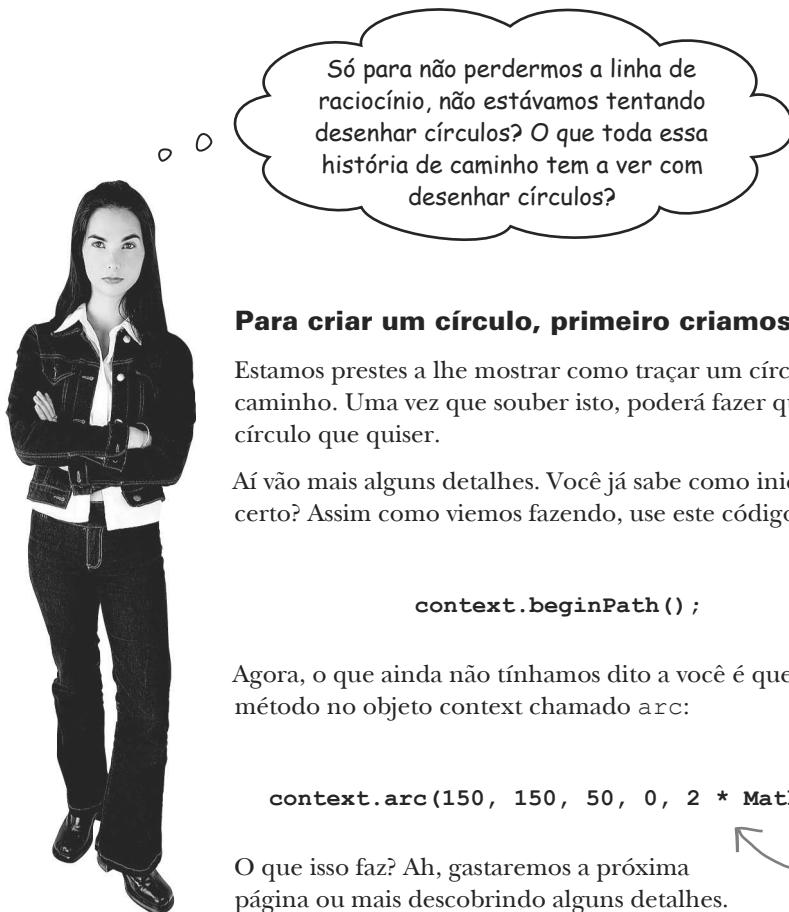
context.fillStyle = "red";

context.fill();
```

}

Aqui está o código que fizemos até agora.

E aqui temos um novo código. Vá em frente e faça anotações daí para que você acha que ele faz. Carregue a página. Você estava certo? Verifique suas respostas no fim do capítulo.



Para criar um círculo, primeiro criamos um caminho.

Estamos prestes a lhe mostrar como traçar um círculo como um caminho. Uma vez que souber isto, poderá fazer qualquer tipo de círculo que quiser.

Aí vão mais alguns detalhes. Você já sabe como iniciar um caminho, certo? Assim como viemos fazendo, use este código:

```
context.beginPath();
```

Agora, o que ainda não tínhamos dito a você é que existe outro método no objeto context chamado arc:

```
context.arc(150, 150, 50, 0, 2 * Math.PI, true);
```

O que isso faz? Ah, gastaremos a próxima página ou mais descobrindo alguns detalhes. Como você deve imaginar, ele traça um caminho junto ao círculo.

↑ Será que você se lembra das aulas de geometria nas quais se dizia que a circunferência de um círculo = $2\pi r$? Guarde isto na memória por um segundo...

Desvendando o método arc

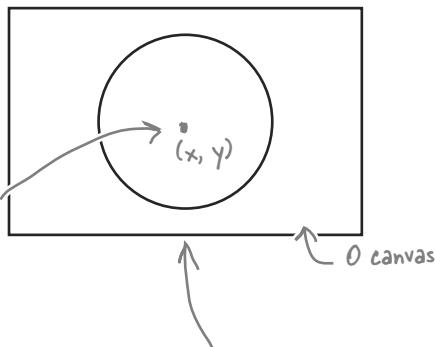
Vamos mergulhar fundo no método arc e verificar seus parâmetros.

```
context.arc(x, y, radius, startAngle, endAngle, direction)
```

A questão principal do método arc é especificar como queremos traçar um caminho ao longo de um círculo. Vejamos como cada parâmetro contribui com isso:

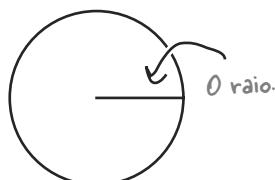
x, y Os parâmetros x e y determinam onde será localizado o centro de seu círculo no canvas.

Esta é a posição x, y do centro de seu círculo.



context.arc(x, y, radius,

raio (radius) Este parâmetro é usado para especificar 1/2 da largura do círculo.



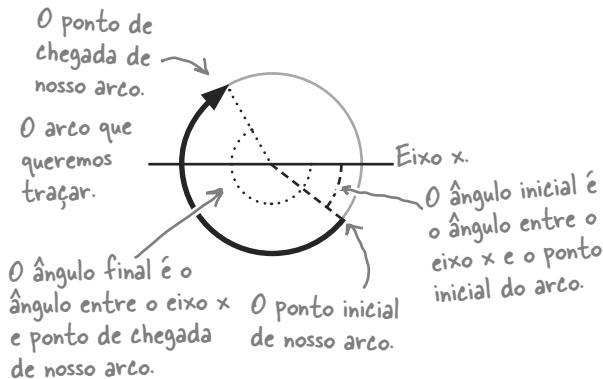
direction

Determina se estamos criando o caminho do arco no sentido anti-horário ou horário. Se a direção for verdadeira, giramos no sentido anti-horário; se for falsa, giramos no sentido horário.



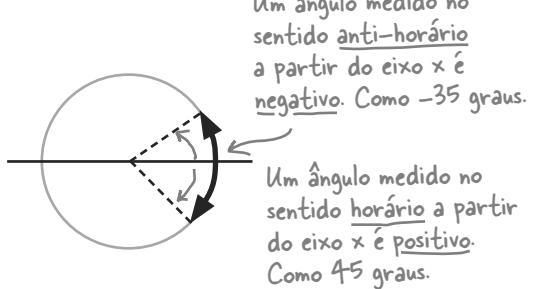
startAngle, endAngle, direction)

startAngle, endAngle O ângulo inicial e o final do arco determinam onde o caminho de seu arco começa e onde termina no círculo.



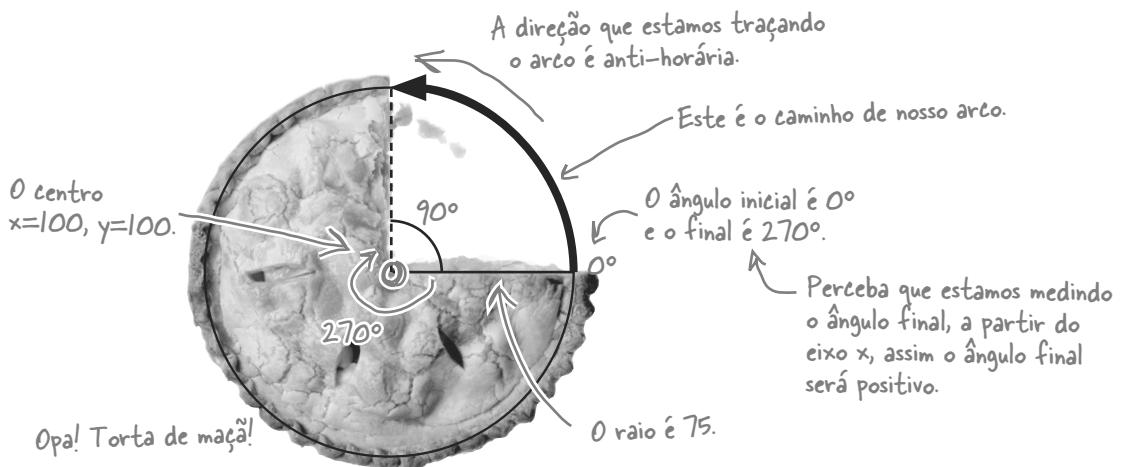
Detalhe importante logo abaixo!

Não se esqueça disso. Os ângulos podem ser medidos na direção negativa (anti-horário a partir do eixo x) ou na direção positiva (horário a partir do eixo x). Isso *não* é o mesmo que o parâmetro de direção para o caminho do arco! (Você verá na próxima página.)



Provando um pouco do uso do arco

O que precisamos agora é de um bom exemplo. Digamos que você queira traçar um arco sobre um círculo que está centrado em $x=100$, $y=100$, deseje que o círculo tenha 150 pixels de largura (ou um raio de 75) e que o caminho traçado seja de apenas 1/4 do círculo, assim:



Agora vamos criar uma chamada de método arc que trace este caminho:

- Começamos pelo ponto x, y no centro do círculo: 100, 100.

```
context.arc(100, 100, __, __, _____, __);
```

- Depois, precisamos do raio do círculo, 75.

```
context.arc(100, 100, 75, __, _____, __);
```

- E quanto aos nossos ângulos inicial e final? Bem, o ângulo inicial é 0, porque o ponto inicial está em 0 graus do eixo x. O ângulo final é o ângulo entre o eixo x e o ponto final de nosso arco. Já que nosso arco é um arco de 90 graus, nosso ângulo final é de 270 graus ($90+270=360$). (Note que, se medíssemos no negativo, ou na direção anti-horária, em vez disso, nosso ângulo final seria de -90 graus.)

```
context.arc(100, 100, 75, 0, degreesToRadians(270), __);
```

- Finalmente, estamos traçando o arco numa direção anti-horária, portanto usamos true.

```
context.arc(100, 100, 75, 0, degreesToRadians(270), true);
```

Voltaremos nisso já, já. Isso só converte graus (com os quais já estamos acostumados) em radianos (o que o context parece preferir).

Eu digo grau, você diz radiano

Todos falamos de ângulos de círculos todos os dias: "Que 360 da hora, cara", ou "Estava indo em direção àquele caminho, aí dei um 180 completo", ou... bem, você entendeu. O único problema é que nós pensamos em *graus*, mas o contexto do canvas pensa em *radianos*.

Agora, podemos dizer que:

$$360 \text{ degrees} = 2\pi \text{ radians}$$

Um radiano é só outra medida de um ângulo. Um radiano é igual a $180/3,14159265\dots$ (ou 180 dividido por π).



e você está pronto, se quiser, para calcular radianos de cabeça daqui por diante. Ou, se por alguma razão não quiser calcular de cabeça, eis uma função bem útil que fará este trabalho para você:

```
function degreesToRadians(degrees) {  
    return (degrees * Math.PI) / 180;  
}
```

talvez se lembre de ter visto isto rapidamente no capítulo sobre Geolocalização.

Para obter radianos a partir de graus, você multiplica por π e divide por 180.

Use esta função sempre que quiser pensar em graus, mas obter radianos para desenhar um arco.

Na página 313 você nos viu usar $2 * Math.PI$ para especificar o ângulo final de um círculo. Você poderia fazer isso... ou apenas use `degreesToRadians(360)`.

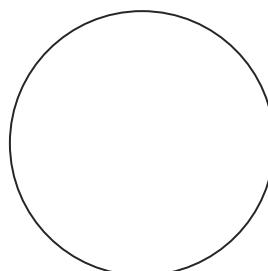
Sinta-se como o Navegador

Interprete a chamada ao método `arc` e rascunhe todos os valores no círculo, inclusive o caminho que o método cria.

```
context.arc(100, 100, 75, degreesToRadians(270),  
0, true);
```



Anote este círculo com todos os argumentos e então desenhe o caminho que esta chamada de método cria.



Dica: o que sobra da torta depois de comer esta parte?



Voltando a escrever o código dos círculos da TweetShirt

Agora que você sabe como desenhar círculos, está na hora de voltar ao TweetShirt e adicionar uma nova função, `drawCircle`. Queremos desenhar 20 círculos aleatórios, assim como fizemos com os quadrados. Para desenhar aqueles círculos, precisamos, primeiramente, determinar que o usuário selecionou círculos a partir do menu shape (formas). Vamos adicionar isso à função `previewHandler`.

Edita seu arquivo `tweetshirt.js` e adicione o novo código abaixo.

```
function previewHandler() {  
    var canvas = document.getElementById("tshirtCanvas");  
    var context = canvas.getContext("2d");  
    fillBackgroundColor(canvas, context);  
  
    var selectObj = document.getElementById("shape");  
    var index = selectObj.selectedIndex;  
    var shape = selectObj[index].value;  
  
    if (shape == "squares") {  
        for (var squares = 0; squares < 20; squares++) {  
            drawSquare(canvas, context);  
        }  
    } else if (shape == "circles") {  
        for (var circles = 0; circles < 20; circles++) {  
            drawCircle(canvas, context);  
        }  
    }  
}
```

Este código parece quase idêntico ao código para testar quadrados. Se o usuário escolheu círculos em vez de quadrados, então desenharemos 20 círculos com a função `drawCircle` (a qual, agora, precisamos escrever).

Estamos passando a função `drawCircle` ao `canvas` e ao `contexto`, assim como fizemos com `drawSquares`.



Qual ângulo inicial e final você usa para desenhar um círculo completo?

Qual a direção que usa: anti-horária ou horária? Isso importa?

A: Você desenha um círculo com um ângulo inicial de 0° e um ângulo final de 360° . Não importa qual a direção que vai usar, contanto que esteja desenhando um círculo completo.

Escrevendo a função drawCircle...

Agora vamos escrever a função drawCircle. Lembre-se: aqui só precisaremos desenhar um círculo aleatório. O outro código já está providenciando a chamada desta função 20 vezes.

```
function drawCircle(canvas, context) {
    var radius = Math.floor(Math.random() * 40);
    var x = Math.floor(Math.random() * canvas.width);
    var y = Math.floor(Math.random() * canvas.height);

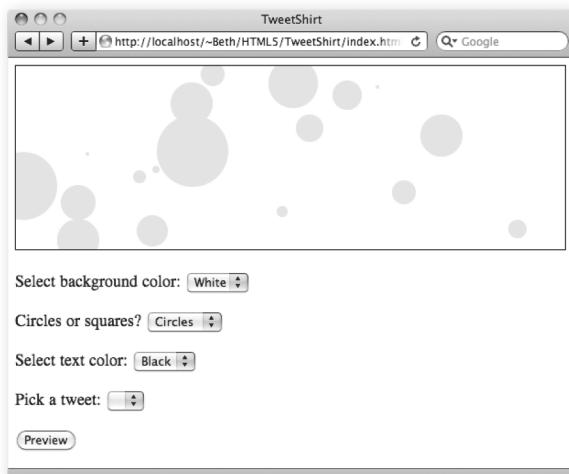
    context.beginPath();
    context.arc(x, y, radius, 0, degreesToRadians(360), true);

    context.fillStyle = "lightblue";
    context.fill();
```

} ← Estamos usando "lightblue" como fillStyle, novamente, e então preenchendo o caminho com context.fill().

... e hora do test drive! 

Então, vá nessa e digite tudo isso (e não se esqueça de adicionar a função degreesToRadians também). Salve e carregue em seu browser. Aí está o que vemos (dado que esses são círculos aleatórios — o seu deve ficar um pouco diferente):



Assim como fizemos para os quadrados, estamos usando 40 para o tamanho máximo do raio para evitar que nossos círculos fiquem grandes demais.

E, novamente, as coordenadas x e y do centro do círculo são baseadas na largura e na altura do canvas. Escolhemos números aleatórios entre 0 e a largura e a altura, respectivamente.

Usamos um ângulo final de 360° para obter um círculo completo. Desenhamos no sentido anti-horário em torno do círculo, mas, para um círculo, não importa qual a direção utilizada.



Intervalo



Uma pequena pausa para um cookie

Ufa! Esse foi um belo calhamço de páginas que acabamos de estudar. Não sabemos quanto a você, mas estamos prontos para alguns cookies. Que tal darmos uma pequena pausa para uns cookies? Não fique achando, porém, que não vamos dar algo legal para você fazer enquanto estamos comendo (dê uma checada no exercício da direita).

Então, sente-se, descance bem e dê uma beliscada nisso enquanto você dá ao seu cérebro e ao seu estômago outra coisa para fazer por um instante. Depois, volte aqui e terminaremos o código TweetShirt!

À direita, você encontrará um rosto sorridente (ou um cookie com um rosto sorridente, se preferir). O código abaixo para desenhar a carinha feliz está quase terminado; apenas precisamos de sua ajuda. Com todo o trabalho que já fez nesse capítulo, você tem tudo o que precisa para concluir-lo. Você pode sempre verificar sua resposta no fim do capítulo quando tiver terminado.

```
function drawSmileyFace() {
    var canvas = document.getElementById("smiley");
    var context = canvas.getContext("2d");

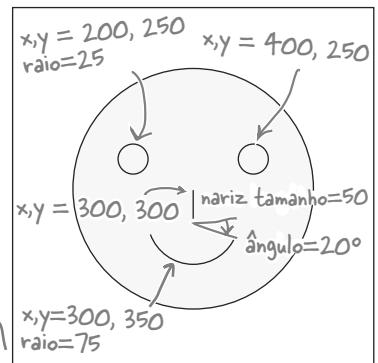
    context.beginPath();
    context.arc(300, 300, 200, 0, degreesToRadians(360), true);
    context.fillStyle = "#ffffcc";
    context.fill();
    context.stroke();

    context.beginPath();
    context.arc(___, ___, 25, ___, _____, true); ← Este é o olho esquerdo.
    context.stroke();

    context.beginPath();
    context.arc(400, ___, ___, ___, _____, _____); ← Este é o olho direito.
    context.stroke();

    context.beginPath();
    context._____(___, ___); ← Este é o nariz.
    context._____(___, ___);
    context._____();

    context.beginPath();
    context._____(300, 350, ___, degreesToRadians(___), degreesToRadians(___), ___);
    context.stroke();
}
```



Isso é o que estamos procurando. Você talvez queira fazer alguns cookies de verdade enquanto estuda isso...

O círculo do rosto. Fizemos esse para você. Perceba que o preenchemos com amarelo.

Este é o olho esquerdo.

Este é o olho direito.

Este é o nariz.

E aqui está a boca. Esta é a mais complicada!

Bem-vindo de volta...

Você voltou, descansou, revigorou-se e estamos na reta final de pormos isso para funcionar.

Quando você olha para todo o trabalho que fizemos, tudo o que realmente deixamos de lado foi mostrar os tuítes e o outro texto na pré-visualização do canvas.

Agora, para colocar um tuíte no canvas, primeiro precisaremos de seus tuítes recentes para escolhermos algum. Vamos usar o JSONP para inseri-los. Se você se lembra do Capítulo 6, já sabe como proceder. Se precisar, volte ao Capítulo 6 para dar uma recordada. Tudo o que faremos é:

- 1 Adicione um `<script>` no fim do arquivo `tweetshirt.html` para fazer uma chamada à API JSONP do Twitter. Vamos requisitar o mais recente status de atualizações de um usuário específico.
- 2 Implemente um callback para obter os tuítes que o Twitter está nos enviando. Usaremos o nome deste callback na URL para o `<script>` do Passo 1.

Eis nosso arquivo HTML para a TweetShirt.

```
<html>
  ...
  <body>
    <form>
      ...
      </form>
    </body>
</html>
```

Imagine seu elemento head aqui e seu formulário aqui (queríamos salvar algumas árvores).

Aqui está a nossa chamada JSONP; funciona resgatando o JSON criado por chamar a URL Twitter e, então, passando aquele JSON para a função callback (a qual definiremos num instante).

Aqui está a chamada API do Twitter. Estamos solicitando a linha do tempo de um usuário, que nos dará status recentes.

Mude isso para o seu nome de usuário, ou para o de outrem, se preferir.

```
<script src="http://twitter.com/statuses/user_timeline/wickedsmartly.json?callback=updateTweets">
  </script>
```

E aqui está a função callback para onde o JSON será passado de volta.

Tem muita coisa acontecendo aqui. Se isto for apenas vagamente familiar, por favor, dê uma boa olhada em como funciona o JSONP, no Capítulo 6.

Digite isto tudo em apenas uma linha em seu arquivo de texto (é muito longa para caber em uma linha do livro).



Obtendo seus tuítes

Já fizemos o trabalho duro, que é obter os tuítes do Twitter.

Agora, precisamos adicioná-los ao elemento `<select>` dos tuítes no `<form>` de nossa página. Só para recordar: quando a função callback é chamada (em nosso caso, a função `updateTweets`), o Twitter envia uma resposta que

A resposta do Twitter é um array de tuítes. Cada tuíte tem uma tonelada de dados dentro; a parte que vamos usar é o texto do tuíte.

Edita seu arquivo `tweetshirt.js` e adicione a função `updateTweets` ao final. Aqui está o código:

Aqui está nossa callback.

A qual recebe uma resposta contendo os tuítes da linha do tempo do usuário como um array de tuítes.

Pegamos uma referência à seleção dos tuítes a partir do formulário.

```
function updateTweets(tweets) {
```

```
    var tweetsSelection = document.getElementById("tweets");
```

Para cada tuíte no array, nós:

```
    for (var i = 0; i < tweets.length; i++) {
```

Pegamos um tuíte do array.

```
        tweet = tweets[i];
```

Criamos um novo

```
        var option = document.createElement("option");
```

elemento option.

```
        option.text = tweet.text;
```

Definimos seu texto ao tuíte.

```
        option.value = tweet.text.replace("\\"", "\"");
```

E definimos seu valor

ao mesmo texto, apenas processamos a string um pouco para substituir as aspas duplas por aspas simples (assim podemos evitar problemas de formatação no HTML).

```
        tweetsSelection.options.add(option);
```

Em seguida, pegamos a nova opção e adicionamo-la à seleção de tuíte no formulário.

```
}
```

```
    tweetsSelection.selectedIndex = 0;
```

Depois de termos feito isto para cada tuíte, teremos um elemento `<select>` que contém uma opção para cada tuíte.

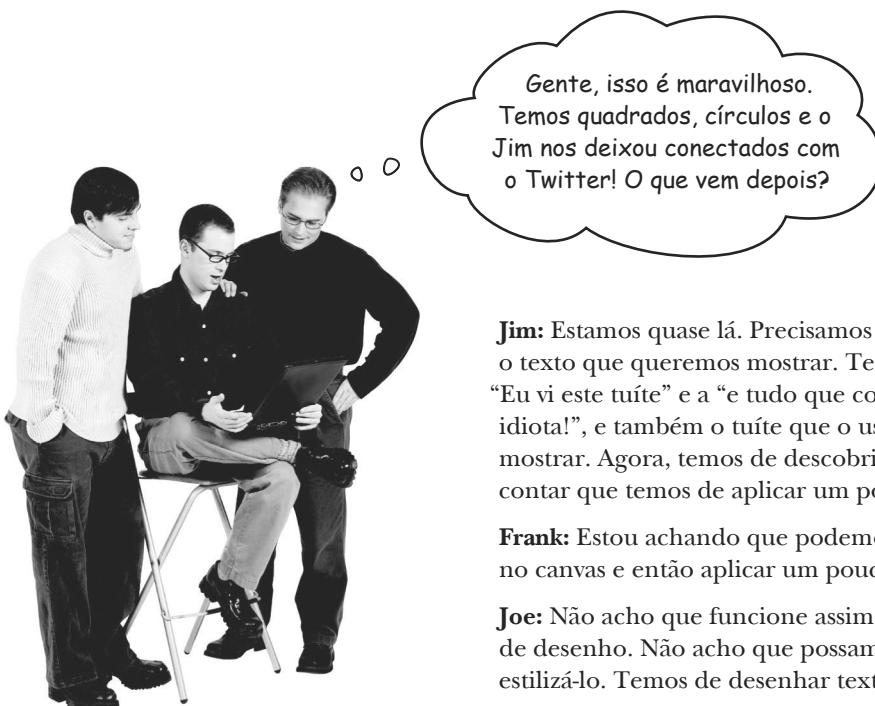
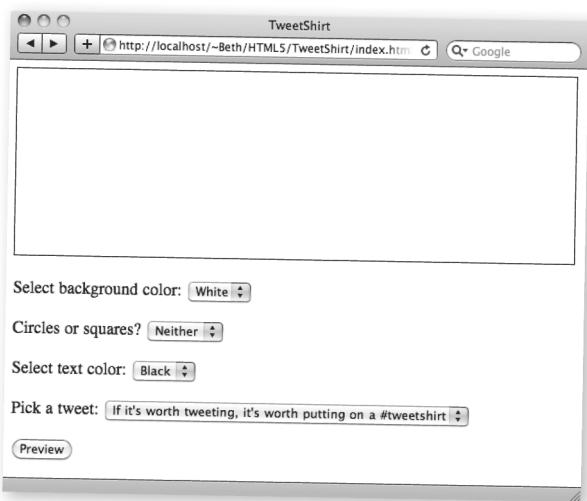
```
}
```

E, finalmente, certificamo-nos de que o primeiro tuíte é o selecionado, ao ajustar o `selectedIndex` do `<select>` para 0 (o primeiro elemento `option` contido nele).

Test drive de tuítes

Façamos um rápido test drive. Verifique se adicionou todo o código ao tweetshirt.js e ao index.html. Certifique-se também de que está usando um nome de usuário do Twitter, que possui tuítes recentes em seu script src URL (isso garantirá que verá alguns tuítes!). Carregue a página e clique na seleção de tuítes. Isto é o que veremos:

Aqui está o menu de tuítes
com tuítes REAIS nele. Legal!



Tablet do Frank

Jim: Estamos quase lá. Precisamos acertar em cheio todo o texto que queremos mostrar. Temos duas mensagens: a “Eu vi este tuíte” e a “e tudo que consegui foi esta camiseta idiota!”, e também o tuíte que o usuário escolheu para mostrar. Agora, temos de descobrir como mostrá-lo, sem contar que temos de aplicar um pouco de estilo ao texto.

Frank: Estou achando que podemos jogar alguns textos no canvas e então aplicar um pouco de CSS neles, certo?

Joe: Não acho que funcione assim. O canvas é uma área de desenho. Não acho que possamos colocar texto e estilizá-lo. Temos de desenhar texto nele.

Jim: Bem, dessa vez aprendi minha lição e já verifiquei a API para texto.

Joe: Bom... eu não vi ainda; como é?

Jim: Lembra-se daquele método arc? Temos de fazer um desenho personalizado por todo nosso texto usando ele.

Frank: Está brincando? Acho que vou trabalhar o fim de semana inteiro agora.

Jim: Te peguei! Não falei sério. Existe um método `fillText` que pega o texto para desenhar no canvas junto com a posição x, y, onde o desenha.

Joe: Isso parece bem direto. E quanto às diferenças no estilo? Se eu lembrar do esboço, o tuíte será itálico e o resto do texto negrito.

Jim: Precisamos procurar um pouco mais. Existem vários métodos para configurar o alinhamento, as fontes e os estilos de preenchimento, mas não estou muito certo sobre como usá-los.

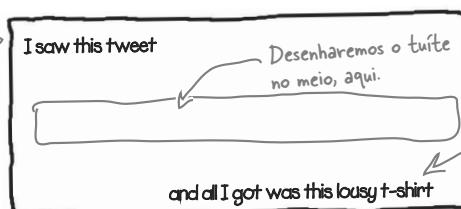
Frank: E pensar que talvez eu pudesse ter ajudado, mas sem CSS?

Jim: Desculpe, como disse o Joe, isto é uma API para desenhar no canvas, não fazendo uso de HTML ou estilização CSS de qualquer tipo.

Joe: Bem, vamos nos esforçar na API e dar uma olhada. Daí poderemos tentar colocar o texto "Eu vi um tuíte" no canvas. Venha Frank, junte-se a nós. Não vai ser tão ruim e tenho certeza de que poderemos usar seu conhecimento de fontes, estilos e tudo mais.

Frank: Claro, estou aqui para ajudá-los!

Precisamos escrever o texto "Eu vi este tuíte" acima do tuíte verdadeiro, no canto superior esquerdo.



Select background color:

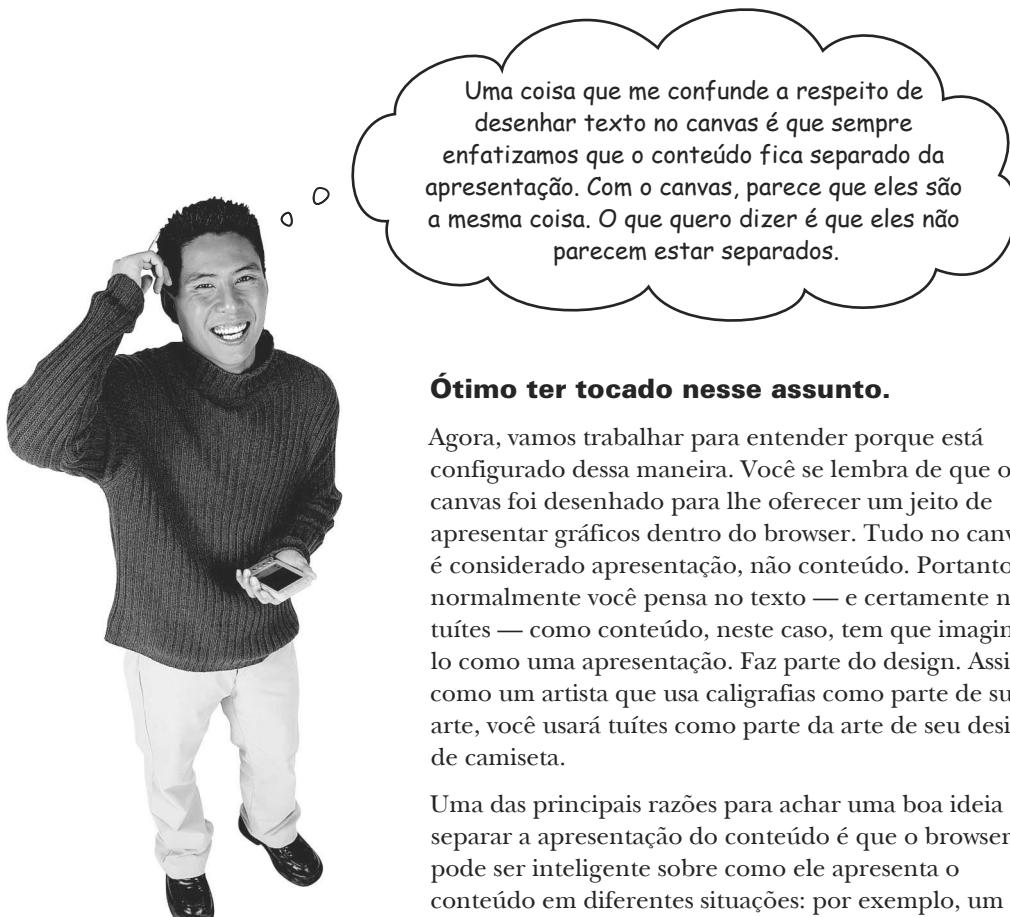
Circles or Squares?

Select text color:

Pick a tweet:

Pegaremos a seleção de cor de primeiro plano para usar como a cor do texto.

Já pegamos os tuítes no menu tweet.



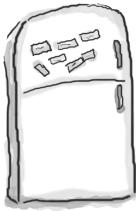
Ótimo ter tocado nesse assunto.

Agora, vamos trabalhar para entender porque está configurado dessa maneira. Você se lembra de que o canvas foi desenhado para lhe oferecer um jeito de apresentar gráficos dentro do browser. Tudo no canvas é considerado apresentação, não conteúdo. Portanto, se normalmente você pensa no texto — e certamente nos tuítes — como conteúdo, neste caso, tem que imaginá-lo como uma apresentação. Faz parte do design. Assim como um artista que usa caligrafias como parte de sua arte, você usará tuítes como parte da arte de seu design de camiseta.

Uma das principais razões para achar uma boa ideia separar a apresentação do conteúdo é que o browser pode ser inteligente sobre como ele apresenta o conteúdo em diferentes situações: por exemplo, um artigo de um site de notícias é apresentado numa tela grande, de uma maneira, e em seu telefone, de outra.

Para o design de nossa camiseta, queremos que o que está no canvas seja mais como uma imagem: deveria ser mostrado da mesma maneira, não importando como está sendo visualizado.

Então, vamos colocar o texto no canvas e pôr logo isso para funcionar!



Ímãs de Geladeira

Está na hora de seu primeiro experimento com texto em canvas. Abaixo, começamos o código para `drawText`, o método que chamaremos para desenhar todo o texto na pré-visualização do canvas. Veja se consegue terminar o código para desenhar "Eu vi esse tuíte" e "e tudo que consegui foi essa porcaria de camiseta!" no canvas, pois salvaremos o desenho do tuíte real para depois. Certifique-se de conferir sua resposta com a solução no fim do capítulo, antes de partir para a prática.

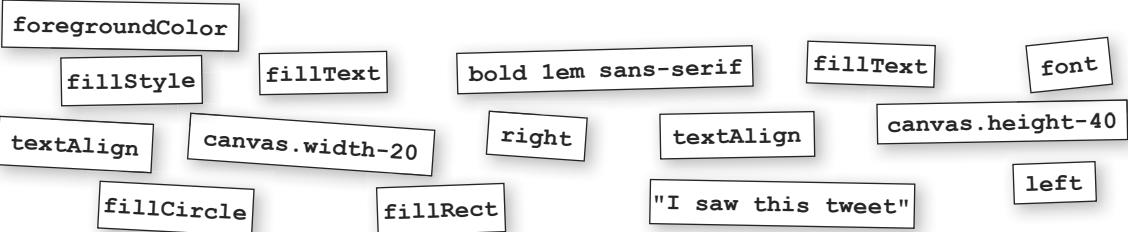
```
function drawText(canvas, context) {
    var selectObj = document.getElementById("_____");
    var index = selectObj.selectedIndex;
    var fgColor = selectObj[index].value;
    context._____ = fgColor;
    context._____ = "bold 1em sans-serif";
    context._____ = "left";
    context._____(_("_____"), 20, 40);
    // Get the selected tweet from the tweets menu
    // Draw the tweet
    context.font = "_____";
    context._____ = "_____";
    context._____ ("and all I got was this lousy t-shirt!",
    _____, _____);
}
```

Dica: esta é a posição x, y para o texto "eu vi esse tuíte".

Dica: usaremos uma fonte com serifa itálica para o tuíte, mas queremos que essa aqui seja serif-sans em negrito.

Dica: queremos posicionar o texto no canto inferior direito.

Por ora,
estamos
apenas pondo
os comentários
no código.





Texto no Canvas de Perto

Agora que você teve uma chance de desenhar seu primeiro texto no canvas, está na hora de dar uma olhada mais de perto nos métodos e propriedades de texto disponíveis na API do canvas. Como já descobriu no exercício, esta é uma API considerada de baixo-nível — você tem de dizer ao contexto qual texto desenhar, qual posição ou qual fonte utilizar etc.

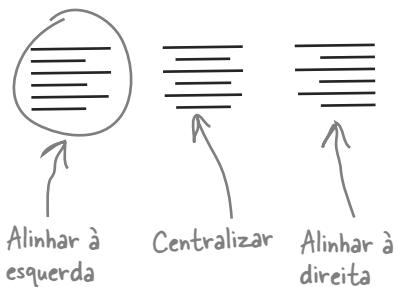
Neste segmento, examinaremos as propriedades do alinhamento, da fonte e da baseline, assim como os métodos de preencher e contornar em detalhes, de forma que você se torne um expert de texto em canvas, assim que virar a página!

alinhamento

A propriedade `textAlign` especifica onde está o ponto âncora para o texto. “start” é o padrão.

```
context.textAlign = "left";
```

Valores possíveis são: start (início), end (fim), left (esquerda), right (direita) e center (centro). Start e end significam o mesmo que left e right nos idiomas que são escritos da esquerda para a direita, como o inglês ou português, e são invertidos nos idiomas escritos da direita para a esquerda, como o Hebraico.



preenchimento e contorno

Assim como para os retângulos, podemos contornar e preencher o texto. Fornecemos o texto para desenhar; a posição x, y e um parâmetro opcional: largura máxima, que faz com que o texto seja dimensionado, se for maior que a largura máxima.

```
context.fillText("Dog", 100, 100, 200);
```

```
context.strokeText("Cat", 100, 150, 200);
```

Texto preenchido

Dog ↗

Cat ↗

Texto contornado

Se o texto ficar maior que 200, será automaticamente dimensionado para caber.

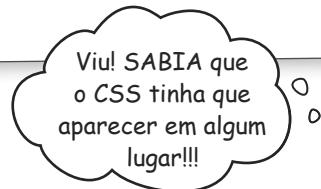
fonte

Para ajustar as propriedades de fonte, você pode usar o mesmo formato que está acostumado a usar no CSS, o que é muito útil. Se especificar todos os valores de propriedade, você incluirá: estilo de fonte, peso, tamanho e família, nesta ordem.

```
context.font = "2em Lucida Grande";           → Tea
context.fillText("Tea", 100, 100);

context.font = "italic bold 1.5em Times, serif"; → Coffee
context.fillText("Coffee", 100, 150);
```

As especificações recomendam que use apenas fontes vetorizadas (fontes bitmap podem não ficar bem).



baseline

A propriedade `textBaseline` ajusta os pontos de alinhamento da fonte e determina a linha em que suas letras se estabelecem. Para ver como a linha afeta seu texto, tente desenhá-la no mesmo ponto x, y que você desenha um texto.

```
context.beginPath();
context.moveTo(100, 100);
context.lineTo(250, 100);
context.stroke();
context.textBaseline = "middle";
context.fillText("Alfabeto", 100, 100);
```

Alfabeto ↪ *alphabetic*

Alfabeto ↪ *bottom*

Alfabeto ↪ *middle*

Alfabeto ↪ *top*

Os valores possíveis são: top, hanging, middle, alphabetic, ideographic e bottom. O padrão é alphabetic. Experimente os diferentes valores para encontrar o que precisa (e verifique as especificações para mais detalhes).

Vá testar o drawText

Agora que você sabe um pouco mais sobre a API, vá em frente e digite aquele código que criou no exercício de Ímãs de Geladeira — aqui estão os ímãs traduzidos em código:

```

function drawText(canvas, context) {
    var selectObj = document.getElementById("foregroundColor");
    var index = selectObj.selectedIndex;
    var fgColor = selectObj[index].value;

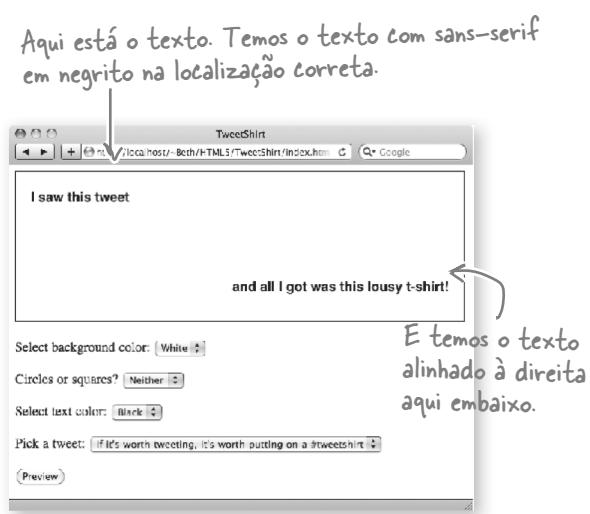
    context.fillStyle = fgColor;
    context.font = "bold 1em sans-serif";
    context.textAlign = "left";
    context.fillText("I saw this tweet", 20, 40);

    context.font = "bold 1em sans-serif";
    context.textAlign = "right";
    context.fillText("and all I got was this lousy t-shirt!",
        canvas.width-20, canvas.height-40);
}

```

Vamos pôr o código que desenha o texto do tuíte aqui já, já.

Depois de ter digitado, atualize sua função previewHandler para chamar a função drawText, e faça um test drive, carregando tudo em seu browser. Você deverá ver algo como o que temos:



Tente você completar a função drawText. É preciso obter o tuíte selecionado; ajustar a fonte itálica com serifa, que é um pouco (1.2em) maior que a do padrão; certifique-se de que o texto esteja alinhado à esquerda e posicione-o a x=30, y=100. Este é o último passo antes de vermos a TweetShirt!

Escrive seu código acima
e não olhe a próxima
página! (Mesmo!)

Completando a função drawText

Aqui está nossa solução do código. Como está em comparação com a sua? Se você ainda não digitou seu código, vá e digite o código abaixo (ou sua versão, se preferir) e recarregue seu index.html. Nós lhe mostraremos nosso test drive na próxima página.

```
function drawText(canvas, context) {
    var selectObj = document.getElementById("foregroundColor");
    var index = selectObj.selectedIndex;
    var fgColor = selectObj[index].value;

    context.fillStyle = fgColor;
    context.font = "bold 1em sans-serif";
    context.textAlign = "left";
    context.fillText("I saw this tweet", 20, 40);

    selectObj = document.getElementById("tweets");
    index = selectObj.selectedIndex;
    var tweet = selectObj[index].value;
    context.font = "italic 1.2em serif";
    context.fillText(tweet, 30, 100);

    context.font = "bold 1em sans-serif";
    context.textAlign = "right";
    context.fillText("and all I got was this lousy t-shirt!",
        canvas.width-20, canvas.height-40);
}
```

Não precisamos alinhar o texto do tuíte à esquerda; o alinhamento ainda é ajustado daqui de cima.

Pegamos a opção selecionada do menu do tuíte.

Ajuste a fonte para itálica com serif, só um pouquinho maior... ... e desenhe-a na posição 30, 100.

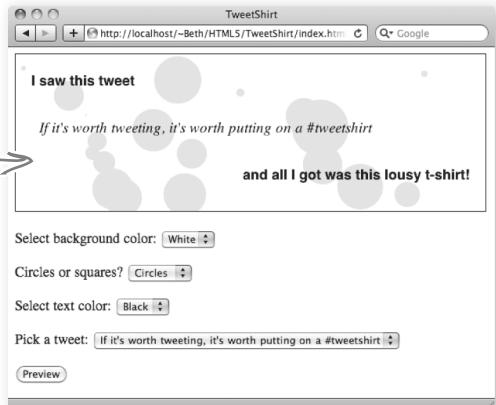


Um pequeno test drive e então ~~almoco~~ vamos nessa!

Esperamos que você esteja vendendo o que estamos vendendo! Legal, né?! Dê à interface um pouco de garantia de qualidade testando: experimente todas as combinações de cores e formas, ou troque o nome do usuário por outro que goste.

Está com a sensação de que está pronto para rodá-lo de verdade? Vamos nessa!

Veja o tuíte na pre-visualização da camiseta.
Legal!





- 1 A primeira coisa que precisamos é de uma imagem. Pusemos uma imagem chamada `twitterBird.png` na pasta TweetShirt. Para colocá-la no canvas, primeiro precisamos de um objeto de imagem JavaScript. Veja como faremos isso:

```
var twitterBird = new Image();
```

Crie um novo objeto de imagem.
E ajuste sua fonte
`twitterBird.src = "twitterBird.png";` para que seja a imagem do pássaro do Twitter.

- 2 A próxima parte deverá parecer bem tranquila agora; precisamos desenhar a imagem no canvas usando um método context chamado, você já adivinhou, `drawImage`.

```
context.drawImage(twitterBird, 20, 120, 70, 70);
```

Usando o método drawImage ↑ Aqui está ↑ E especificamos a localização x, y, largura e altura. de imagem.

- 3 Só mais uma coisa que você deve saber sobre imagens: elas nem sempre carregam imediatamente. Então, você precisa se certificar de que a imagem está completamente carregada antes de desenhá-la. Como esperaremos até algo estar carregado antes de darmos o próximo passo? Implementaremos um handler `onload`:

```
twitterBird.onload = function() {
```

Aqui, estamos dizendo: quando a imagem tiver carregado, então execute esta função.

```
    context.drawImage(twitterBird, 20, 120, 70, 70);
```

}; Desenhamos a imagem no canvas usando o método `drawImage` do contexto.



Veja se consegue montar a função drawBird com todas as peças que a Judy nos deu. A função drawBird pega um canvas e um contexto para depois desenhar o pássaro dentro do canvas. Você pode imaginar que, com esta função, estamos colocando "twitterBird.png" na localização x=20, y=120, com uma largura e altura de 70. Escrevemos a declaração do método e a primeira linha para você. Encontre nossa solução no fim do capítulo.

```
function drawBird(canvas, context) {
    var twitterBird = new Image();
```

Seu código →
aqui.

}

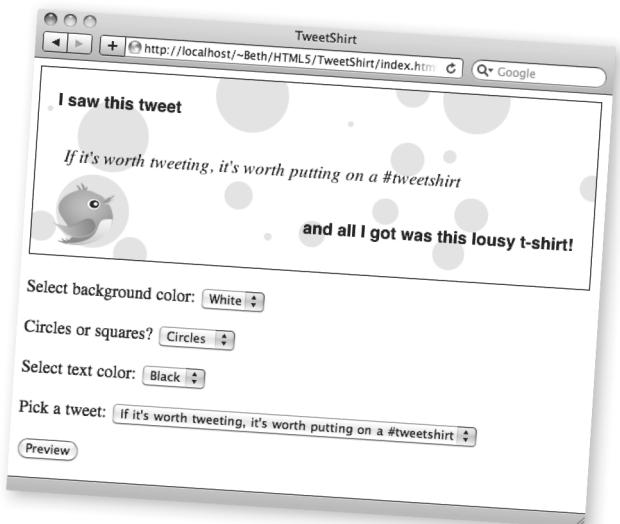
Certifique-se de
adicionar uma chamada
à função drawBird na
função previewHandler.

Mais um test drive

Cheque duas vezes seu código e
faça outro test drive! Nossa, agora
está ótimo.

Teste-o algumas vezes; experimente
círculos ou quadrados. Você
perceberá que usamos um png com
um background transparente, de
forma que os círculos e quadrados
apareçam, se estiverem por detrás
do pássaro.

Isso está muito legal e estamos
progredindo nessa história
de desenvolver um aplicativo
bacana, mas como dissemos,
estamos contando contigo para
implementar o e-commerce, a
concretização e tudo mais...



Perguntas Idiotas

P: Não vimos o objeto Image antes. Você o utilizou quando adicionou uma imagem no canvas. O que é isso? Por que não criamos com document.createElement("img")?

R: Boa sacada. Ambos os métodos que mencionou criam objetos de imagem. O construtor JavaScript Image é possivelmente uma maneira mais direta para se criar imagens a partir do JavaScript e nos dá um maior controle sobre o processo de carregamento (como a habilidade de facilmente usar um handler para ser notificado quando a imagem é carregada).

Portanto, nossa meta aqui é criar uma imagem e ter certeza de que ela será carregada antes de desenhá-la no canvas. O objeto Image nos dá o melhor caminho para fazer isso.

P: Canvas é legal... mas também meio cansativo, se comparado ao HTML. Algo mais complicado do que formas básicas deve ser meio difícil de fazer.

R: Não tenha dúvida, pois você está desenhando códigos de gráficos, quando está programando canvas. É diferente do browser, que dá conta de vários detalhes por você, como elementos de fluidez nas páginas, para que não tenha de se preocupar em desenhar tudo sozinho. No canvas, você precisa dizer onde pôr tudo.

O canvas, porém, lhe dá o poder de fazer quase qualquer tipo de gráfico (atualmente, 2D) que puder imaginar e ainda estamos no princípio do canvas. É provável que bibliotecas de código JavaScript tornarão mais fácil escrever os gráficos no futuro.

P: Percebi que, para tuítes muito longos, o tuíte simplesmente desaparece nos limites do canvas. Como faço para consertar isso?

R: Uma maneira é verificar o número de caracteres que o tuíte contém e, se for maior que determinado número, divida-o em várias linhas e desenhe cada uma separadamente no canvas. Incluímos uma função, splitIntoLines, no código em [wickedlysmart](#), com a qual você poderá fazer exatamente isso.

P: Também percebi que alguns tuítes possuem entidades HTML neles, como " e & . O que é isso?

R: É a API do Twitter que estamos usando para obter os tuítes, à medida que o JSON converte os caracteres que as pessoas postam em seus tuítes para as entidades HTML. É, na verdade, uma coisa boa, porque qualquer caractere especial, ou mesmo citações, que estragariam nossa habilidade de obter

os tuítes apropriadamente do JSON, são representadas com entidades. Se estivéssemos mostrando os tuítes em HTML, aquelas entidades seriam mostradas no browser como os caracteres que você gostaria de ver, assim como as entidades que adicionou em sua própria página são mostradas corretamente nos browser. No entanto, como você viu, no canvas elas não ficam tão boas. Infelizmente, ainda não existe qualquer função na API do canvas, que converta aquelas entidades de volta aos seus caracteres; então, será necessário que o faça sozinho.

P: É possível fazer alguma coisa sofisticada, como pôr sombras no texto ou formas?

R: Sim! Há muitas coisas sofisticadas que você pode fazer com o canvas, e sombra é certamente uma delas. Como deve imaginar, você cria uma sombra ajustando as propriedades no contexto. Por exemplo: para definir o tamanho do borrão da sombra, você deve definir context.shadowBlur. Você pode definir a posição da sombra com context.shadowOffsetX e context.shadowOffsetY, e a cor com context.shadowColor.

Outras coisas que você pode fazer no canvas, que talvez queira saber, são do tipo desenhar gradientes, rotação de formas e pôr cantos arredondados nos retângulos.

P: Quais outras coisas interessantes posso fazer com o canvas?

R: Várias! Vamos falar mais sobre o uso do canvas em capítulos posteriores e, com certeza, você vai querer dar uma olhada na API do canvas para saber mais em: <http://dev.w3.org/html5/2dcontext/>.

P: Todo esse negócio de canvas vai funcionar no meu dispositivo móvel também? Ou terei de reescrever tudo para usuários mobile?

R: Se seu dispositivo móvel tiver um browser moderno (dispositivos como Android, iPhone e iPad, todos servem), então vai funcionar direitinho (o tamanho da página talvez fique meio estranho, mas a funcionalidade estará perfeita). O legal do canvas é que, pelo fato de você estar desenhando com pixels básicos, o que desenhar parecerá igual em qualquer lugar (ou, em qualquer browser que suportar canvas). Felizmente, dispositivos inteligentes modernos, como o Android, iPhone e iPad, possuem browsers sofisticados que apresentam a maioria das funcionalidades dos browsers de desktop.



O
O

Pensei que seria legal ser possível
salvar uma camiseta e a localização e
posição de todos os seus quadrados.
Existe algum método de salvamento no
canvas para isso?

Não, isso requer um pouco de trabalho extra.

O canvas foi realmente feito para simplesmente desenhar superfícies. Quando você desenha uma forma, o canvas apenas a vê em pixels. O canvas não tem ciência das especificidades daquilo que é desenhado e não rastreia quaisquer formas. Ele simplesmente cria os pixels que você pede para serem criados. (Se você estiver familiarizado com os termos gráficos desenho “bitmap” e “vetorizado”, você reconhecerá o que o canvas está fazendo como desenho “bitmap”).

Se quiser tratar os retângulos em seu canvas como um conjunto de objetos que você pode salvar e talvez até mover ou manipular, é preciso manter as informações a respeito das formas e dos caminhos quando os criou no canvas. Pode-se armazenar estes dados nos objetos JavaScript. Por exemplo: se estiver rastreando os círculos aleatórios que desenhamos no canvas TweetShirt, precisará salvar a localização x, y, o raio do círculo e a cor, para que seja capaz de recriar aquele círculo.

Esse parece ser um bom projeto para você...

Parabéns, equipe! Vocês conseguiram! Até funciona em meu iPad; então, é perfeito para clientes em movimento. Estou empolgada. Estamos organizando uma festa de lançamento da TweetShirt. Venham participar.



A fundadora da TweetShirt também queria transmitir que está feliz por ver o aplicativo funcionando no iPad e até no iPhone dela! Se ela está feliz, nós também estamos.





PONTOS DE BALA

- Canvas é um elemento que você coloca em sua página para criar um espaço de desenho.
- O canvas não tem estilo ou conteúdo padrão até que você forneça (então você não o verá na página até que desenhe algo nele ou adicione uma borda com CSS).
- Você pode ter mais de um canvas em sua página. É claro, será preciso dar a cada um uma id única para acessar cada JavaScript em uso.
- Para especificar o tamanho do elemento canvas, use os atributos de largura e altura no elemento.
- Tudo que você põe no canvas é desenhado com o auxílio do JavaScript.
- Para desenhar no canvas, primeiro precisa criar um contexto. Atualmente, um contexto 2D é sua única opção, embora outros tipos de contexto possam vir a existir no futuro.
- Um contexto é necessário para desenhar no canvas, pois fornece um tipo específico de interface (i.e., 2D versus 3D). Você será capaz de escolher mais de um tipo de interface para desenhar no canvas.
- Você acessa o canvas usando propriedades e métodos de contexto.
- Para desenhar um retângulo no canvas, use o método context.fillRect. Isso cria um retângulo preenchido com cor.
- Para criar um contorno de retângulo, use strokeRect em vez de fillRect.
- Use fillStyle e strokeStyle para mudar a cor padrão de preenchimento e contorno, que é preta.
- Você pode especificar cores usando o mesmo formato que usa com o CSS (i.e., "black", "#000000", "rgb(0, 0, 0)"). Lembre-se de pôr aspas em torno do valor do fillStyle.
- Não há método fillCircle. Para desenhar um círculo no canvas, é preciso desenhar um arco.
- Para criar formas ou arcos arbitrários, primeiro crie um caminho.
- Um caminho é uma linha ou forma invisível, que você cria para definir uma linha ou área no canvas. Você não verá o caminho até que o contorne ou preencha.
- Para criar um triângulo, crie um caminho usando beginPath; então, use moveTo e lineTo para desenhar o caminho. Use closePath para unir as duas pontas do caminho.
- Para desenhar um círculo, crie um arco de 360 graus. Seu ângulo inicial é 0 e o final é 360 graus.
- Ângulos são especificados no canvas usando radianos, não graus. Será preciso converter de graus em radianos para especificar seus ângulos inicial e final.
- $360 \text{ graus} = 2\pi \text{ radianos}$.
- Para desenhar texto no canvas, use o método fillText.
- Quando você desenha texto no canvas, é preciso especificar a posição, estilo e outras propriedades usando propriedades de contexto.
- Quando você define uma propriedade de contexto, ela se aplica a todos os desenhos seguintes, até que você mude a propriedade novamente. Exemplo: mudar o fillStyle afetará a cor das formas e dos textos que desenhar após configurar o fillStyle.
- Acrescente uma imagem ao seu canvas com o método drawImage.
- Para acrescentar uma imagem, você primeiro precisa criar um objeto de imagem e certificar-se de que ele está completamente carregado.
- Desenhar no canvas é como fazer desenhos "bitmap" em programas de gráficos.

O WEBVILLE INQUIRIDOR

Temos um furo de reportagem: **<canvas>** e **<video>** são itens, afinal de contas!

Webville – Você lerá primeiro aqui

Após uma entrevista exclusiva, podemos anunciar que **<canvas>** e **<video>** têm feito mais do que apenas compartilhar as mesmas páginas. Sim, é isso mesmo... Vamos apenas dizer que eles têm misturado seus conteúdos.

Por Troy Armstrong
ESCRITOR INQUIRIDOR DA STAFF

<Video> diz, "É verdade, construímos uma relação íntima. Entenda, sou um rapaz bem simples; sei como mostrar vídeos, e sei muito bem. Mas é basicamente tudo o que faço. Com **<canvas>**, tudo mudou. Estou me vestindo com controles personalizados, filtrando meu conteúdo de vídeo, mostrando vídeos múltiplos ao mesmo tempo."

Pedimos a **<canvas>** que comentasse. Será que ela é a mulher por trás da tag **<video>?** **<Canvas>** nos disse, "Bem, o **<video>** faz tudo muito bem sozinho, sabe, decodificando todos aqueles codecs, mantendo seus frames-por-segundo, tudo mais, é um grande trabalho e eu nunca conseguiria fazer isso. Mas comigo ele tem a chance de fugir da comum, ouso dizer, maneira "chata" de rodar vídeos. Dou a ele meios de explorar todos os tipos de possibilidades criativas de mesclar vídeos na experiência web."

Bem, quem poderia imaginar? Acho que temos algumas coisas interessantes além do relacionamento **<canvas>+<video>**!

Podemos esperar a consequência desta revelação para continuarmos bem até o capítulo de vídeo, quando a fluorescente relação será exposta à opinião pública.

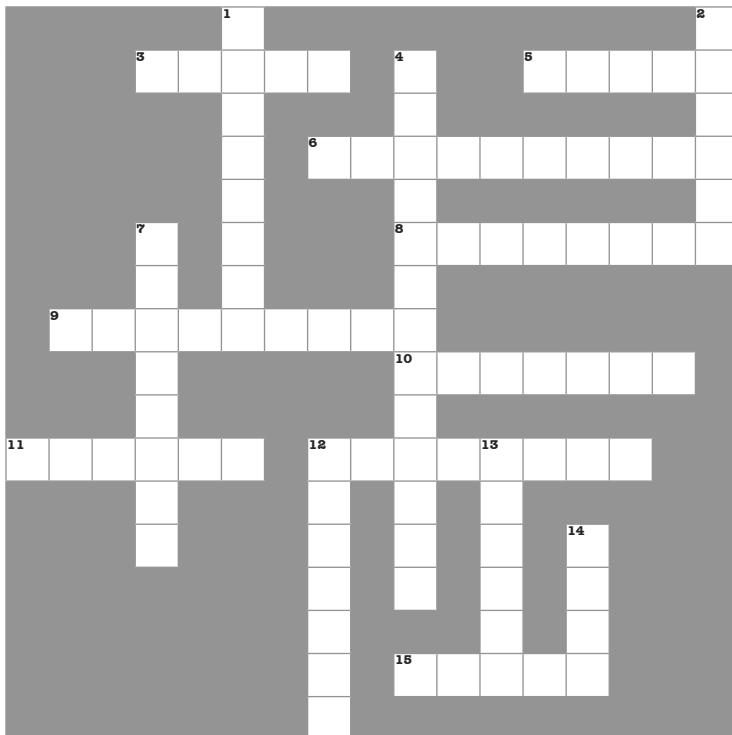


Moradora local Heidi Musgrove chocou-se em saber a verdade a respeito dos dois elementos.



Cruzada HTML5

Estamos ansiosos em saber do furo de reportagem sobre <canvas> e <video> no próximo capítulo. Enquanto isso, solidifique seus novos conhecimentos sobre o canvas com uma rápida palavra cruzada e talvez uma xícara de chá.



HORIZONTAIS

3. Há 360 ____ num círculo.
5. O melhor lugar para um bom tuíte.
6. Use este método para desenhar texto no canvas.
8. Um objeto com métodos e propriedades para desenhar num canvas.
9. A propriedade que definimos para preencher uma forma com uma cor.
10. Nós alinhamos ____ o texto “e tudo que consegui foi essa camiseta idiota!”.
11. Tudo no canvas é ____.
12. Como fazemos o caminho de uma forma visível.
15. Canvas e ____ ficam bem juntos.

VERTICIAIS

1. Pensamos em graus, canvas pensa em ____.
2. Para mover seu lápis-caminho para o ponto 100, 100 use ____(100, 100).
4. Quer saber qual opção é selecionada? Talvez queira essa propriedade.
7. Este método context cria um retângulo.
12. Uma linha invisível que você criou para desenhar uma forma.
13. Você pode dizer quando algo terminou de carregar usando um handler ____.
14. Desenhar um círculo com um(a) ____.

Sinta-se como o Navegador – Solução

Digamos que você tenha usado a interface para escolher os valores para sua camiseta.

Agora que você tem uma interface, execute essas afirmações JavaScript e escreva no valor para cada elemento de interface.



```
var selectObj = document.getElementById("backgroundColor");
var index = selectObj.selectedIndex;
var bgColor = selectObj[index].value; ..... branco
```

```
var selectObj = document.getElementById("shape");
var index = selectObj.selectedIndex;
var shape = selectObj[index].value; ..... círculos
```

```
var selectObj = document.getElementById("foregroundColor");
var index = selectObj.selectedIndex;
var fgColor = selectObj[index].value; ..... preto
```

Perceba que, para cada valor menu option, temos o select element em que a opção está contida, encontramos a opção selecionada com a propriedade selectedIndex, e o get the value da opção selecionada.

Lembre-se de que o valor da opção pode ser diferente do texto que você vê nos controles; em nossa situação, é só o caso das primeiras letras do texto.

Aqui estão os valores que escolhemos na interface TweetShirt para criar as respostas acima.

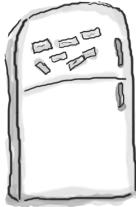
Select background color: White ▾

Circles or squares? Neither ▾

Select text color: Black ▾

Pick a tweet: ▾

Preview



Pseudo-ímãs de Geladeira – Solução

Use seus poderes de código pseudomágicos para arrumar o pseudocódigo para a função drawSquare. Esta função pega um canvas e um contexto e desenha um quadrado aleatoriamente dimensionado no canvas. Aqui está nossa solução.

```
function drawSquare ( canvas , context ) {  
    calcule uma largura  
    aleatória para o quadrado  
  
    calcule uma posição x aleatória  
    para o quadrado dentro do canvas  
  
    calcule uma posição y aleatória  
    para o quadrado dentro do canvas  
  
    ajuste fillStyle para "lightblue"  
  
    desenhe um quadrado na  
    posição x , com a largura w  
}
```

Fizemos esse aqui para você.

Seus ímãs vão aqui!



Aponte o seu lápis Solução

Para nos certificarmos de que apenas veremos novos quadrados no canvas cada vez que clicarmos em pré-visualização, precisamos preencher o background do canvas com a cor do background que o usuário selecionou no menu de seleção “backgroundColor”. Primeiro, vamos implementar uma função para preencher o canvas com aquela cor. Preencha os espaços em branco abaixo para completar o código. Aqui está nossa solução.

```
function fillBackgroundColor(canvas, context) {  
    var selectObj = document.getElementById("backgroundColor");  
    var index = selectObj.selectedIndex;  
    var bgColor = selectObj.options[index].value;  
    context.fillStyle = bgColor;  
    context.fillRect(0, 0, canvas.width, canvas.height);  
}
```

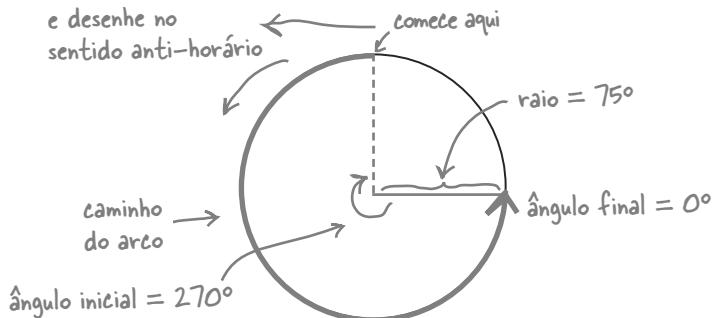
Tudo que faremos para criar uma cor de background é desenhar um retângulo que preencha o canvas inteiro com uma cor.

Sinta-se como o Navegador

Interprete a chamada ao método arco e rascunhe todos os valores no círculo, inclusive o caminho que o método cria.



```
context.arc(100, 100, 75, degreesToRadians(270),
0, true);
```



Exercício Solução

Quer dizer que você tem um caminho! E agora?

Você utiliza o caminho para desenhar linhas e preencher sua forma com cor, é claro! Vá em frente e crie uma simples página HTML5 com um elemento canvas e digite todo o código feito até agora. Faça um teste também.

```
context.beginPath();
context.moveTo(100, 150);
context.lineTo(250, 75);
context.lineTo(125, 30);
context.closePath();

context.lineWidth = 5;
context.stroke();
context.fillStyle = "red";
context.fill();
```

}

Aqui está o código que fizemos até agora.

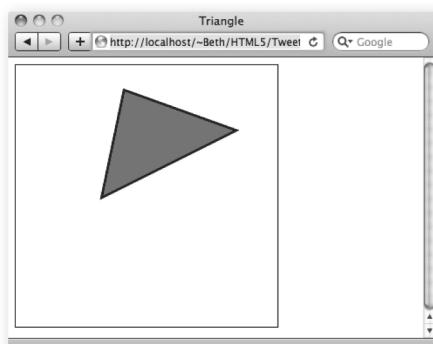
Defina a largura da linha para desenhar sobre o caminho.

Desenhe sobre o caminho com a linha.

Defina a cor para preencher o triângulo com vermelho.

Preencha o triângulo com vermelho.

Quando carregamos nossa página de triângulo, é isso que vemos (fizemos um canvas de 300 x 300 para desenhar).

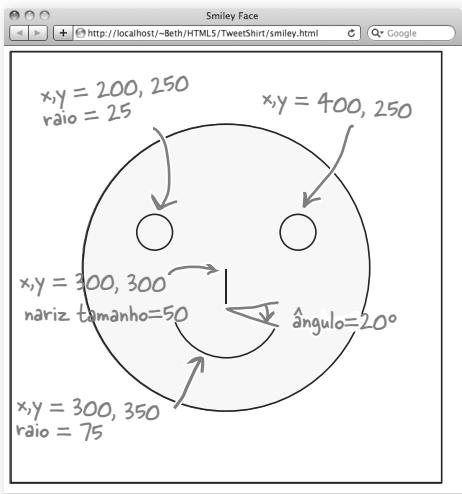




Intervalo – Solução

Hora de praticar suas novas habilidades de desenhar arcos e caminhos para criar uma carinha feliz. Preencha os espaços em branco com o código necessário para completar a carinha feliz. Demos-lhe algumas dicas de onde deveriam estar os olhos, nariz e boca no diagrama.

Veja nossa solução:



```
function drawSmileyFace() {
    var canvas = document.getElementById("smiley");
    var context = canvas.getContext("2d");

    context.beginPath();
    context.arc(300, 300, 200, 0, degreesToRadians(360), true);
    context.fillStyle = "#ffffcc";
    context.fill();
    context.stroke();

    context.beginPath();
    context.arc(200, 250, 25, 0, degreesToRadians(360), true);
    context.stroke();

    context.beginPath();
    context.arc(400, 250, 25, 0, degreesToRadians(360), true);
    context.stroke();

    context.beginPath();
    context.moveTo(300, 300);
    context.lineTo(300, 350);
    context.arc(300, 350, 75, degreesToRadians(20), degreesToRadians(160), false);
    context.stroke();
}
```

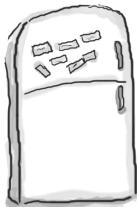
O círculo do rosto.
Fizemos esse para
você. Perceba que
o preenchemos

Este é o olho esquerdo.
O centro do círculo
está em $x=200$,
 $y=250$, o raio é 25,
o ângulo inicial é 0 e
o final é $\text{Math.Pi} * 2$
radianos (360 graus).
Contornamos o caminho
para obter o círculo
(mas sem preenchimento).

Este é o olho direito.
Assim como o olho
esquerdo, exceto por
estar em $x=400$. Usamos
sentido anti-horário
(true) para a direção
(não importa para um
círculo completo).

Para obtermos um sorriso mais realista, começamos
e terminamos a borda da boca em 20 graus
abaixo do eixo x. Isso significa que o ângulo inicial
é de 20° e o final é de 160°.

A direção é horária (false), porque queremos
a boca num sorriso. (Lembre-se de que o
ponto inicial é à direita do centro da boca.)



Pseudo-ímãs de Geladeira – Solução

Está na hora de seu primeiro experimento com texto em canvas. Abaixo, começamos o código para `drawText`, o método que chamaremos para desenhar todo o texto na pré-visualização do canvas. Veja se consegue terminar o código para desenhar “Eu vi esse tuíte” e “e tudo que consegui foi essa porcaria de camiseta!” no canvas, pois salvaremos o desenho do tuíte real para depois. Aqui está nossa solução.

```
function drawText(canvas, context) {
    var selectObj = document.getElementById("foregroundColor");
    var index = selectObj.selectedIndex;
    var fgColor = selectObj[index].value;
    context.fillStyle = fgColor;
    context.font = "bold 1em sans-serif";
    context.textAlign = "left";
    context.fillText("I saw this tweet", 20, 40);
```

Dica: esta é a posição x, y para o texto “eu vi esse tuíte”.

Por ora,
estamos
apenas os
comentários
para
visualizar o
código `drawText`.

```
// Get the selected tweet from the tweets menu
// Draw the tweet

context.font = "bold 1em sans-serif";
context.textAlign = "right";
context.fillText("and all I got was this lousy t-shirt!",
    canvas.width-20, canvas.height-40);
```

Dica: usaremos uma fonte com serif itálica para o tuíte, mas queremos essa aqui como Helvetica negritada.

Dica: queremos posicionar o texto no canto inferior direito.

↑ Queremos desenhar este texto em 20 a partir do lado direito do canvas e 40 a partir da parte inferior do canvas, balanceando assim a primeira linha do texto. ↓

Ímãs restantes.

`fillCircle`

`fillRect`

`left`



Exercício Solução

Veja se consegue montar a função drawBird com todas as peças que a Judy nos deu. A função drawBird pega um canvas e um contexto para depois desenhar o pássaro dentro do canvas. Você pode imaginar que, com esta função, estamos colocando "twitterBird.png" na localização x=20, y=120, com uma largura e altura de 70. Escrevemos a declaração do método e a primeira linha para você. Aqui está nossa solução.

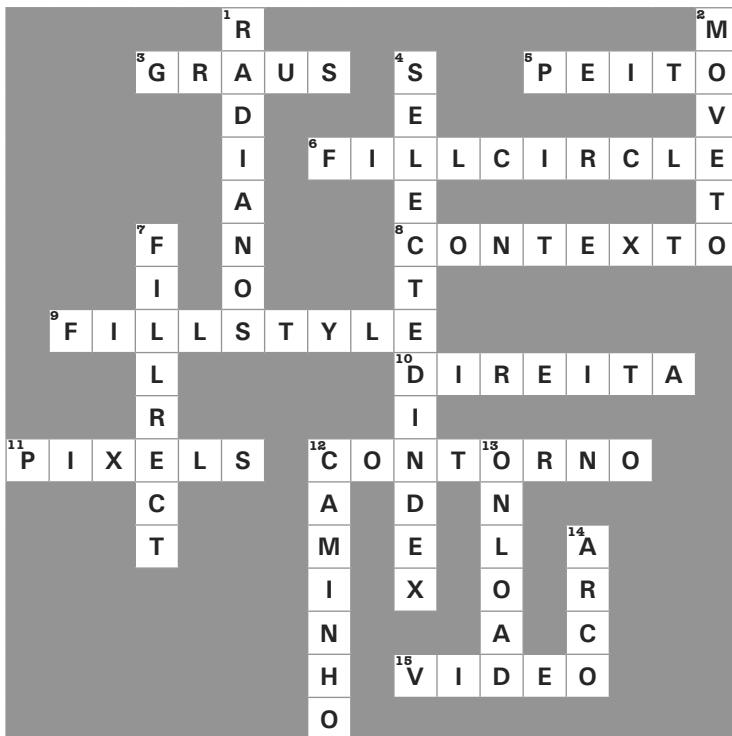
Seu código aqui:→

```
function drawBird(canvas, context) {
    var twitterBird = new Image();
    twitterBird.src = "twitterBird.png";
    twitterBird.onload = function() {
        context.drawImage(twitterBird, 20, 120, 70, 70);
    };
}
```

Não se esqueça
de adicionar uma
chamada à drawBird
na previewHandler!



Cruzada HTML5 – Solução



Ovo de Páscoa TweetShirt

Você fez a pré-visualização perfeita para a TweetShirt — e agora? Bem, se quiser fazer uma camiseta diferente de seu design, você pode! Como? Aí vai um pequeno bônus extra para adicionar ao seu código — um “ovo de páscoa” TweetShirt, se preferir — que criará uma imagem diferente de seu design. Tudo pronto para que você faça o upload para um site que imprimirá uma imagem numa camiseta para você (há uma porção delas na Web).

Como podemos fazer isso? É simples! Podemos usar o método `toDataURL` do objeto **canvas**. Dê uma olhada:

```
function makeImage() {
    var canvas = document.getElementById("tshirtCanvas");
    canvas.onclick = function () {
        window.location = canvas.toDataURL("image/png");
    };
}

↑ Definimos a localização da
janela do browser para a
imagem que será gerada; assim,
verá uma página do browser
com apenas a imagem nela.
```

Fizemos uma nova função, `makeImage`, para adicionar esta funcionalidade.

```
↑ Estamos pedindo ao
canvas para criar uma
imagem png dos pixels
desenhados no canvas.
```

Pegamos o
objeto canvas...

E adicionamos um
event handler para
que, quando você
clique no canvas, ele
crie uma imagem.

Note que png é o
único formato que
deve ser suportado
pelos browsers; então,
recomendamos seu uso.

Agora, só adicione uma chamada para `makeImage` na função `onload` da janela e seu canvas estará habilitado para fazer uma imagem quando clicar sobre ele. Experimente e conte para nós se fizer uma camiseta!

```
window.onload = function() {
    var button = document.getElementById("previewButton");
    button.onclick = previewHandler;
    makeImage();
}
```

Chame `makeImage` para adicionar o
handler do click event ao canvas e
seu ovo de páscoa estará completo.



Alguns browsers não permitem que você pegue imagens do canvas, se estiver rodando o código a partir de file://.

Rode este código a partir de `localhost://` ou de um servidor hospedado, se quiser tê-lo funcionando em outros browsers.



8 não é a TV de seu pai

Vídeo

... com a participação especial da estrela "Canvas"



Não precisamos de plugin. Afinal, vídeo é agora um membro de primeira-classe da família HTML — é só jogar um elemento <video> em sua página e o terá instantaneamente, mesmo em vários dispositivos. O vídeo, no entanto, é *muito mais* que *apenas um elemento*; é também uma API JavaScript que nos permite controlar a reprodução, criar nossas próprias interfaces de vídeo personalizadas e integrá-lo com o resto do HTML de maneiras completamente novas. Falando em *integração*... Lembre-se de que existe aquela *conexão entre vídeo e canvas* que falamos a respeito — você verá que unir vídeo e canvas nos dá uma poderosa nova forma de *processar* vídeo em tempo real. Neste capítulo, vamos começar por rodar vídeos numa página e, então, veremos o que a API JavaScript tem de melhor. Venha. Você ficará encantado ao ver o que é possível fazer com algumas tags, marcação, JavaScript e vídeo & canvas.

Conheça a TV Webville

A TV Webville — todo o conteúdo pelo qual você tem esperado, como *Destination Earth*¹, *O Ataque da Mulher de 15 metros*, *O Monstro do Ártico*, *A Bolha Assassina*, e não estaria longe de nós passar alguns filmes educacionais dos anos 1950. O que mais você esperaria de Webville? Isso, porém, é só o conteúdo. No lado da tecnologia, você esperaria algo menos do que vídeo HTML5?

É claro, essa é só a visão. Temos de construir a TV Webville se quisermos torná-la realidade. Pelas próximas páginas, vamos construí-la desde os primórdios, usando marcação HTML5, o elemento vídeo e um pouco de JavaScript aqui e ali.

A TV Webville
com 100% de
tecnologia HTML5.



Em breve,
num
browser mais
perto de você!

¹ N. E.: Sem título no Brasil, curto desenho que promovia a indústria petroleira e sinalizava como ela enriquecia (e enriquece) a população americana. Marcou sua cultura na década de 1950, em plena Guerra Fria.

O HTML, vamos botar pra quebrar...

Ei, este é o Capítulo 8! Chega de ficar sem fazer nada! Vamos arregaçar as mangas e criar um pouco de HTML.

```
<!doctype html>           ↗ Bem padrão HTML5.
<html lang="en">
<head>
  <title>Webville TV</title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="webvilletv.css">
</head>
<body>
  <div id="tv">
    <div id="tvConsole">
      <div id="highlight">
        
      </div>
      <div id="videoDiv">
        <video controls autoplay src="video/preroll.mp4" width="480"
height="360"
          poster="images/prerollposter.jpg" id="video">
        </video>
      </div>
    </div>
  </div>
</body>
</html>
```

↖ Não se esqueça do arquivo CSS para deixá-lo bonito.

↓ Só um pouco de imagem para ajudar a deixá-lo como um arranjo de televisão.

↑ E aqui está nosso elemento <video> para reproduzir nosso vídeo. Vamos dar uma olhada mais cuidadosa num instante...

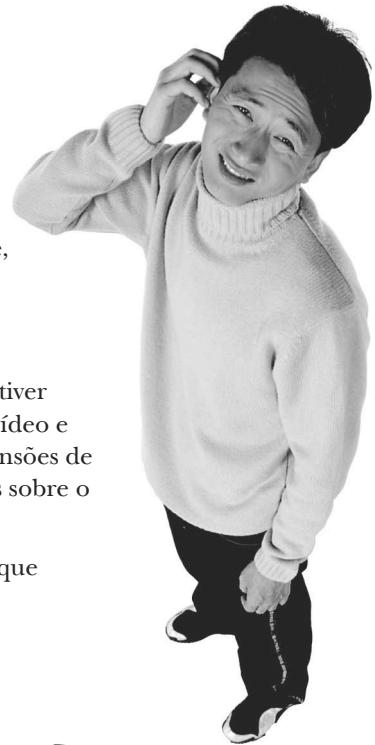
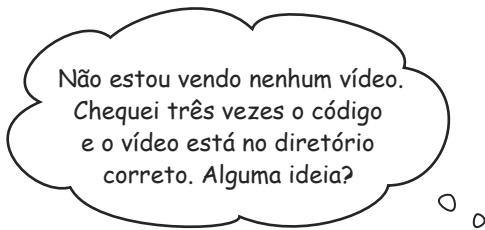
Junte tudo e experimente...

Você precisa se certificar de algumas coisas: primeiro, verifique se o código acima está digitado num arquivo chamado webvilletv.html; segundo, verifique se baixou o arquivo CSS e, finalmente, certifique-se também de ter baixado os arquivos de vídeo e posto eles num diretório chamado video. Feito isto, carregue a página e encoste-se para assistir.

↑ Eis o que vemos. Perceba que, se você passar o ponteiro de seu mouse sobre a tela, verá um conjunto de controles com os quais poderá dar pause, play, ajustar o áudio ou buscar cenas.



Baixe tudo em <http://wickedlysmart.com/hfhtml5>



Na época em que você ler isso, talvez esses formatos sejam mais amplamente suportados pelos browsers. Então, se seus vídeos estiverem funcionando, ótimo. Sempre cheque a web para ver as novidades deste tópico. Voltaremos a ele para falar mais sobre o assunto daqui a pouco.

Sim, é provavelmente o formato do vídeo.

Embora os construtores de browsers concordem em como o elemento <video> e a API aparecerão no HTML5, nem todos concordam no *formato real* dos próprios arquivos de vídeo. Por exemplo: se estiver no Safari, o formato H.264 é melhor; se estiver no Chrome, o WebM é mais favorecido; e assim por diante.

No código que acabamos de escrever, presumimos o H.264 como sendo um formato que funciona no Safari, Mobile Safari e IE9+. Se estiver usando outro browser, então vá até seu diretório vídeo e verá três diferentes tipos, com três diferentes extensões de arquivo: ".mp4", ".ogv" e ".webm" (falaremos mais sobre o que isso significa num segundo).

Para o Safari, você já deveria estar usando .mp4 (que contém H.264).

Para o Google Chrome, use o formato .webm, substituindo seu atributo src por:

`src="video/preroll.webm"`

Se estiver usando Firefox ou Opera, então substitua seu atributo src por:

`src="video/preroll.ogv"`

E se estiver usando IE8 ou anterior, você está sem sorte — espere um segundo, este é o Capítulo 8! Como você ainda está usando o IE8 ou anterior? Atualize! Se precisar saber como suprir o conteúdo para seus usuários IE8, espere aí. Já vamos chegar lá.

Experimente para seguirmos em frente, pois voltaremos a isso num instante.

Como funciona o elemento vídeo?

A esta altura você já tem um vídeo funcionando a todo vapor em sua página, mas, antes de continuarmos, vamos voltar um pouco e checar o elemento vídeo que usamos em nosso marcação:

Se presente, o atributo controls faz com que o player forneça controles para a reprodução do vídeo e do áudio.

<video controls

autoplay O atributo autoplay faz com que o vídeo comece a reprodução durante o carregamento da página.

src="video/preroll.mp4" A localização fonte do vídeo.

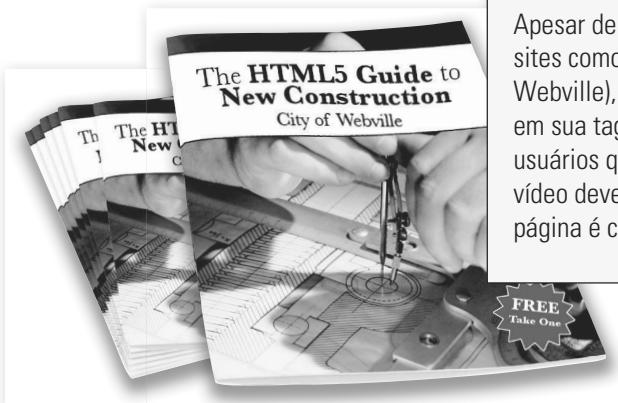
width="480" height="360" A largura e a altura do vídeo na página.

poster="images/prerollposter.jpg" Uma imagem de pôster para mostrar quando o filme não está rodando.

id="video"> Uma id para o elemento vídeo, para que possamos acessá-lo mais tarde a partir do JavaScript.

</video>

Outra dica útil do Guia HTML5 da cidade Webville.

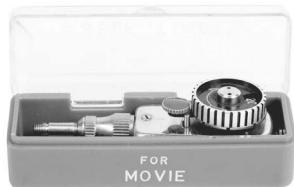


Etiqueta Bom Vídeo: A propriedade autoplay

Apesar de autoplay ser a melhor opção para sites como YouTube e Vimeo (aliás, para a TV Webville), pense duas vezes antes de defini-lo em sua tag <video>. Frequentemente, os usuários querem participar da decisão de se o vídeo deve ou não ser rodado, enquanto sua página é carregada.

Inspecionando de perto os atributos do vídeo...

Vamos dar uma olhada mais de perto em alguns atributos mais importantes do vídeo:



controls

O atributo controls é um atributo booleano. Ou está lá ou não está. Se estiver, então o navegador adicionará seus controles embutidos ao display do vídeo. Isso varia de navegador para navegador. Portanto, verifique cada um para ver como eles ficam. Aqui está sua aparência no Safari.



autoplay

O atributo booleano autoplay diz ao navegador para iniciar o vídeo assim que tiver dados suficientes. Para os vídeos que estamos usando como demonstração, você provavelmente os verá começando quase que imediatamente.

poster

O browser normalmente mostrará um frame do vídeo como uma imagem “pôster” para representá-lo. Se remover o atributo autoplay, você verá esta imagem no display antes de clicar no play. É o navegador que escolhe a imagem a ser mostrada; geralmente, o navegador vai apenas mostrar o primeiro frame do vídeo... que deve ser preto. Se quiser mostrar uma imagem específica, então cabe a você criar uma para exibição e especificá-la usando o atributo poster.

loop

Outro atributo booleano, o loop automaticamente reinicia o vídeo após o fim de sua reprodução.

src é qual arquivo de vídeo que é usado aqui.

src

O atributo src é exatamente como o src do elemento — é uma URL que diz ao elemento vídeo onde encontrar o arquivo fonte. Neste caso, a fonte é video/preroll.mp4. (Se você baixou o código para este capítulo, encontrará este vídeo e dois outros no diretório video.)



O player de vídeo

preload

O atributo booleano preload é tipicamente usado para controle aprimorado sobre como o vídeo carrega para propósitos de otimização. Na maior parte das vezes, o navegador escolhe quanto carregar do vídeo, baseado em coisas como se o autoplay está definido de acordo com a taxa de transferência do usuário. Você pode desprezar isso, definindo o preload para none (nada do vídeo é baixado até que o usuário aperte “play”), metadata (o vídeo metadata é baixado, mas nenhum conteúdo do vídeo) e auto para permitir que o navegador tome a decisão.

width, height

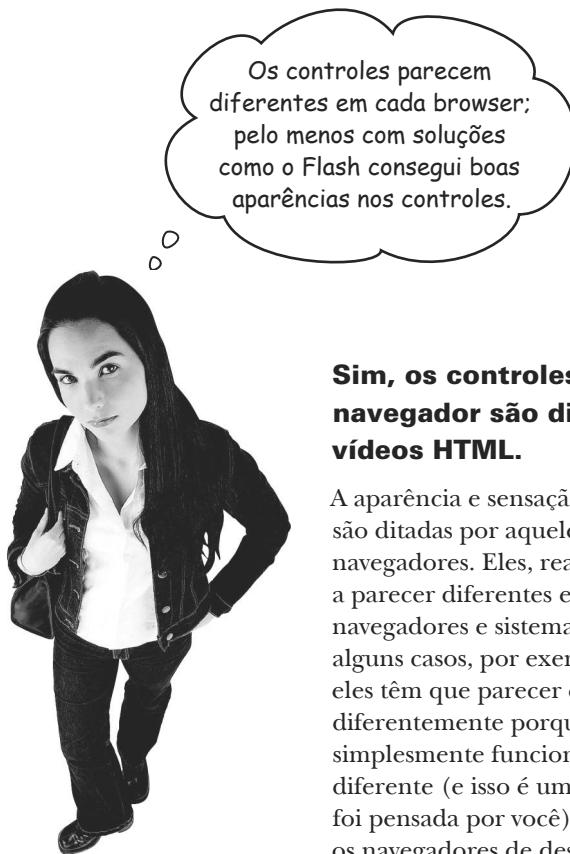
Os atributos width e height ajustam a largura e a altura da área do vídeo, respectivamente (também conhecidos como “viewport”). Se você especificar um pôster, a imagem será dimensionada para a largura e altura que especificar. O vídeo também será dimensionado, mas manterá sua proporção (i.e., 4:3 ou 16:9). Assim, se houver espaço sobrando dos lados, ou no topo e no fim, o vídeo será esticado ou comprimido para caber no tamanho da área de exibição. Deve-se tentar ajustar as dimensões nativas do vídeo se quiser a melhor performance (assim, o navegador não terá que dimensionar em tempo real).

Esticado



Comprimido





Os controles parecem diferentes em cada browser; pelo menos com soluções como o Flash consegui boas aparências nos controles.

Sim, os controles em cada navegador são diferentes com vídeos HTML.

A aparência e sensação de seus controles são ditadas por aqueles que implementam navegadores. Eles, realmente, tendem a parecer diferentes em variados navegadores e sistemas operacionais. Em alguns casos, por exemplo, num tablet, eles têm que parecer e se comportar diferentemente porque o dispositivo simplesmente funciona de forma diferente (e isso é uma coisa boa que já foi pensada por você). Entre, digamos, os navegadores de desktop, seria legal ter controles similares, mas esta não é uma parte formal das especificações HTML5 e, em alguns casos, um método que funcione em um SO pode entrar em colapso com as diretrizes UI de outro sistema operacional. Portanto, saiba apenas que os controles podem diferir, e, se você realmente se sentir motivado, implemente controles personalizados para seus aplicativos.

↑ Faremos isso mais tarde...

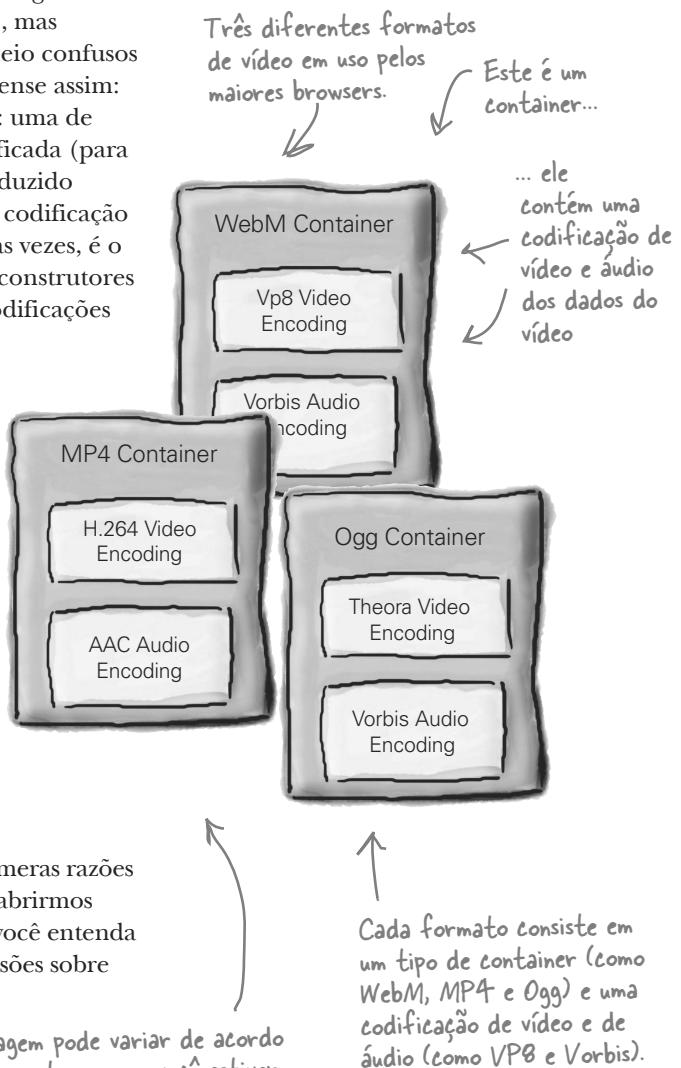
O que você precisa saber sobre formatos de vídeo

Gostaríamos que tudo fosse tão simples e organizado quanto o elemento video e seus atributos, mas acontece que os formatos de vídeo são meio confusos na web. O que é um formato de vídeo? Pense assim: um arquivo de vídeo contém duas partes: uma de vídeo e outra de áudio. Cada uma é codificada (para reduzir o tamanho e permiti-lo ser reproduzido mais eficientemente) usando um tipo de codificação específico. Tal codificação, na maioria das vezes, é o ponto que ninguém concorda — alguns construtores de navegadores são apaixonados pelas codificações H.264, outros realmente gostam de VP8; e ainda outros, como a alternativa de fonte aberta, Theora. Para tornar tudo isso *ainda mais* complicado, o arquivo que contém o vídeo e o áudio codificados (que é conhecido como “container”) possui seu próprio formato com seu próprio nome. Portanto, estamos realmente fazendo uma sopa de letrinhas por aqui.

De qualquer forma, apesar de que o mundo poderia ser muito mais feliz se todos os construtores de navegadores concordassem com apenas um formato para ser usado pela web, bem, isso não parece ser provável por inúmeras razões técnicas, políticas e filosóficas. Em vez de abrirmos este debate aqui, vamos nos garantir que você entenda o tópico para que tome suas próprias decisões sobre como dar suporte ao seu público.

Vamos dar uma olhada nas codificações mais populares por aí; no momento, existem três candidatos tentando mandar no mundo (da web)...

Sua milhagem pode variar de acordo com o momento em que você estiver lendo este livro, à medida que as codificações favoritas tendem a mudar com o passar do tempo.



A especificação HTML5 permite qualquer formato de vídeo. É a implementação do navegador que determina quais formatos são de fato suportados.

Os candidatos

A realidade é que, se você vai servir conteúdo a um amplo espectro de usuários, terá de suprir mais de um formato. Se, por outro lado, tudo com que você se importa é, digamos, o iPad da Apple, é possível se dar bem com apenas um. Atualmente, temos três principais candidatos — vamos observar cada um deles:

Container MP4 com Vídeo H.264 e Áudio AAC

O H.264 é licenciado pelo grupo **MPEG-LA**.

Há mais de um tipo de H.264; cada um é conhecido como um “profile”.

MP4/H.264 é suportado pelo **Safari** e **IE9+**. Talvez encontre suporte em outras versões do **Chrome**.

Container WebM com Vídeo VP8 e Áudio Vorbis

O WebM foi desenvolvido pelo Google para funcionar com vídeos codificados com VP8.

O WebM é suportado pelo **Firefox**, **Chrome** e **Opera**.

Você encontrará vídeos formatados pelo WebM com a extensão **.webm**.

Container Ogg com Vídeo Theora e Áudio Vorbis

Theora é um **codec de fonte aberta**.

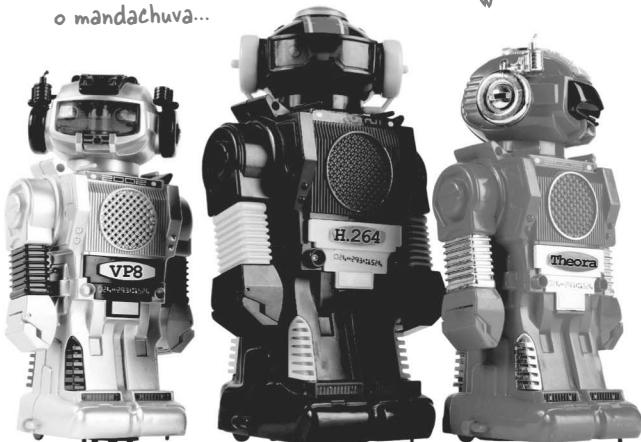
O vídeo codificado pelo Theora é normalmente contido num arquivo Ogg, com a extensão **.ogg**.

Ogg/Theora é suportado pelo **Firefox**, **Chrome** e **Opera**.

O H.264, o queridinho da indústria, mas não o mandachuva...

Theora. A alternativa de fonte aberta.

VP8, o candidato apoiado pelo Google, suportado por outros e chegando forte...



Como manejar todos aqueles formatos...

Bom, já sabemos como o mundo é confuso em relação a formatos de vídeo, mas o que fazer? Dependendo do seu público, você pode decidir fornecer apenas um formato para seu vídeo, ou vários. Em todo caso, você pode usar um elemento `<source>` (não confundir com o atributo `src`) por formato dentro de um elemento `<video>`, para fornecer um conjunto de vídeos, cada um com seu próprio formato, e permitir que o browser escolha o primeiro que der suporte. Desta forma:

The diagram shows the HTML code for a video element with three source tags. Annotations explain the removal of the src attribute from the video tag, the addition of three source tags with their own attributes, and the logic of the browser trying different formats. A juggling clown on a unicycle is used as a metaphor for the browser's process.

```
<video src="video/preroll.mp4" id="video"
       poster="video/prerollposter.jpg" controls
       width="480" height="360">
  <source src="video/preroll.mp4">
  <source src="video/preroll.webm">
  <source src="video/preroll.ogv">
  <p>Sorry, your browser doesn't support the video element</p>
</video>
```

Perceba que estamos removendo o atributo `src` da tag `<video>`...

... e adicionando três tags `source`, cada uma com seus atributos, cada qual com uma versão do vídeo num formato diferente.

Isso é o que o browser mostrará, se não suportar o vídeo.

O browser começa no topo e vai fazendo seu trabalho por todo o caminho, até que embaixo ele encontra um formato que pode rodar.

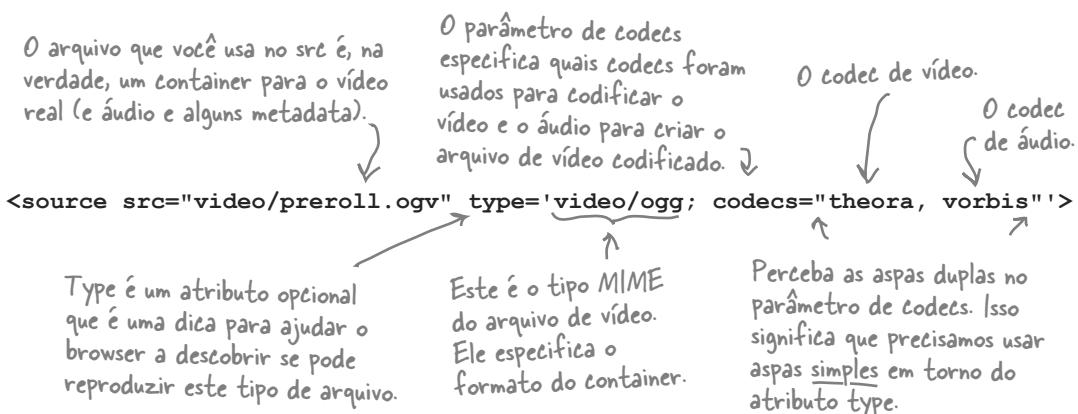
Para cada fonte, o browser carrega o metadata do arquivo de vídeo para ver se ele pode reproduzi-lo (o que pode ser um processo demorado, embora possamos facilitar as coisas no browser... veja a próxima página).

PONTOS DE BALA

- O **container** é o formato de arquivo que é usado para compactar a informação de vídeo, áudio e metadata. Os formatos de container mais comuns são: MP4, WebM, Ogg e Flash Video.
- O **codec** é o software usado para codificar e decodificar uma codificação especial de vídeo ou áudio. Codecs populares na web são: H.264, VP8, Theora, AAC e Vorbis.
- O browser decide qual vídeo pode decodificar e nem todos os construtores de browsers concordam; então, se quiser suportar todos eles, você precisará de codificações múltiplas.

Como ser ainda mais específico com os formatos de vídeo

Dizer ao navegador a localização de seus arquivos de fonte oferece a ele uma seleção de diferentes versões de onde se pode escolher. Contudo, o navegador precisa fazer um trabalho de detetive antes de determinar com certeza se um arquivo é reproduzível. Você pode auxiliar seu navegador ainda mais, dando-lhe mais informação sobre o tipo MIME e (opcionalmente) os codecs de seus arquivos de vídeo:



Podemos atualizar nossos elementos `<source>` para incluir a informação de tipo para todos os três tipos de vídeo que temos, assim:

```
<video id="video" poster="video/prerollposter.jpg" controls width="480" height="360">
  <source src="video/preroll.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
  <source src="video/preroll.webm" type='video/webm; codecs="vp8, vorbis"'>
  <source src="video/preroll.ogv" type='video/ogg; codecs="theora, vorbis"'>
  <p>Sorry, your browser doesn't support the video element</p>
</video>
```

Se você não sabe os parâmetros dos codecs, então pode deixá-los de fora e apenas usar o tipo MIME. Será um pouco menos eficiente, mas, na maioria das vezes, não tem problema.

Os codecs para mp4 são mais complicados que os outros dois, pois o h.264 suporta vários "profiles", codificações diferentes para utilizações diferentes (como alta taxa de transferência vs. baixa taxa de transferência). Portanto, para acertar, você precisará ter mais detalhes sobre como seu vídeo foi codificado.

Se e quando você fizer sua própria codificação de vídeo, precisará saber mais a respeito das várias opções para os parâmetros type para usar em seu elemento source. Você pode obter muito mais informações sobre os parâmetros type em http://wiki.whatwg.org/wiki/Video_type_parameters.

Perguntas ^{não existem} idiotas

P: Existe alguma possibilidade de se chegar a um formato de container ou tipo de codec nos próximos anos? Não é por isso que temos padrões?

R: Provavelmente, não haverá uma codificação que terá o domínio sobre todas as outras num futuro próximo — como dissemos antes, este tópico interage com tantos outros assuntos, desde empresas querendo controlar seus próprios destinos no espaço do vídeo até um complexo conjunto de problemas de propriedade intelectual. O comitê de padrões HTML5 reconhece isso e decidiu não especificar um formato de vídeo apenas para o HTML5. Então, enquanto que em princípio o HTML5 dê suporte (ou, ao menos, seja incerto com relação) a todos esses formatos, cabe aos construtores de navegadores decidirem o que farão e não suportarão.

Fique atento a este tópico, se vídeos forem importantes para você; será, certamente, bem interessante prestar atenção nele nos próximos anos, já que está tudo meio resolvido. Como sempre, tenha em mente o que seu público precisa e tenha certeza de fazer o que pode para lhe dar suporte.

P: Se quiser codificar meu próprio vídeo, por onde começo?

R: Há uma porção de programas de captura e codificação de vídeos por aí e qual você irá escolher vai, realmente, depender de que tipo de vídeo vai capturar e como quer utilizar o produto final. Livros inteiros têm sido escritos sobre codificação de vídeos; portanto, esteja preparado para adentrar um mundo de novos acrônimos e tecnologias. Comece com programas

simples, como iMovie ou Adobe Premiere Elements, que incluem a habilidade de codificar seus vídeos para a web. Se estiver querendo se envolver a sério, trabalhe com Final Cut Pro ou Adobe Premiere, pois esses programas incluem em suas próprias ferramentas de produção. Finalmente, se estiver desenvolvendo seus vídeos a partir de uma Content Delivery Network (CDN), muitas empresas CDN também oferecem serviços de codificação. Então, você tem uma grande variedade de escolhas, dependendo das suas necessidades.

P: Posso reproduzir meu vídeo em tela cheia? Estou surpreso por não haver uma propriedade para isto numa API.

R: Essa funcionalidade ainda não foi padronizada, embora você encontre maneiras para fazer isso com alguns dos navegadores, se vasculhar a web. Alguns dos navegadores oferecem um controle de tela cheia (por exemplo, em tablets), que dá ao elemento vídeo esta capacidade. Note também que, uma vez que consiga usar a tela cheia, o que você conseguir fazer com o vídeo, desconsiderando a reprodução básica, pode ficar limitado por razões de segurança (assim como é com soluções de vídeo plugin hoje em dia).

P: E quanto ao volume do meu vídeo? Posso usar a API para controlar o nível do volume?

R: Certamente que sim. A propriedade volume pode ser definida a um valor com ponto flutuante entre 0.0 (sem som) a 1.0 (som no máximo). Apenas use seu objeto vídeo para definir isto a qualquer tempo:
`video.volume = 0.9;`

SUA PRÓXIMA MISSÃO: RECONHECIMENTO DO VÍDEO **CONFIDENCIAL**

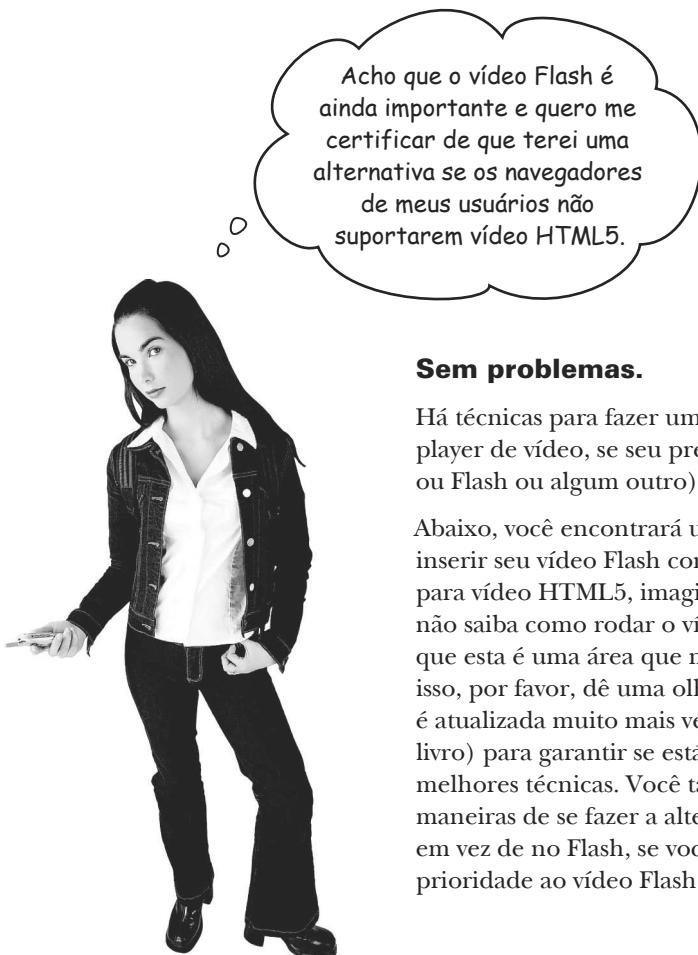
Vá e determine [REDACTED] o atual nível de suporte para vídeo em cada navegador abaixo (dica, aqui estão alguns sites que estão relacionados a tais coisas: http://en.wikipedia.org/wiki/html5_video, <http://caniuse.com/#search=video>). Considere a última versão do navegador. Para cada browser/característica, faça uma marca se tiver suporte. Quando voltar, avise-nos para o encaminharmos à sua próxima missão!

Dispositivos iOS e Android (entre outros)



| Browser | Safari | Chrome | Firefox | Mobile Webkit | Opera | IE9+ | IE8 | IE7 or < |
|------------|--------|--------|---------|---------------|-------|------|-----|----------|
| video | | | | | | | | |
| H.264 | | | | | | | | |
| WebM | | | | | | | | |
| Ogg Theora | | | | | | | | |





Acho que o vídeo Flash é ainda importante e quero me certificar de que terei uma alternativa se os navegadores de meus usuários não suportarem vídeo HTML5.

Sem problemas.

Há técnicas para fazer uma alternativa por outro player de vídeo, se seu preferido (seja HTML5 ou Flash ou algum outro) não for suportado.

Abaixo, você encontrará um exemplo de como inserir seu vídeo Flash como uma alternativa para vídeo HTML5, imaginando que o browser não saiba como rodar o vídeo HTML5. Claro que esta é uma área que muda toda hora. Por isso, por favor, dê uma olhada na web (que é atualizada muito mais vezes do que um livro) para garantir se está usando as últimas e melhores técnicas. Você também encontrará maneiras de se fazer a alternativa no HTML5, em vez de no Flash, se você preferir dar prioridade ao vídeo Flash.

```
<video poster="video.jpg" controls>
  <source src="video.mp4">
  <source src="video.webm">
  <source src="video.ogv">
  <object>...</object>
</video>
```

↑ Insira seu elemento `<object>` dentro do elemento `<video>`. Se o browser não souber do elemento `<video>`, o `<object>` será usado.

Não me disseram que haveria APIs?

Como pode ver, você pode fazer muita coisa utilizando marcação e o elemento <video>. Sendo que este também expõe uma API rica, que pode ser usada para implementar todos os tipos de comportamentos e experiências de vídeo interessantes. Segue um rápido resumo de alguns dos métodos, propriedades e eventos do elemento <video> pelos quais você pode vir a se interessar (e cheque as especificações para uma lista compreensiva):



Chame esses Métodos

- `play` ← ação seu vídeo
 - `pause` ← pausa o vídeo
 - `load` ← carrega seu vídeo
- `canPlayType` ↗ ajuda-o a determinar quais tipos de vídeo podem ser reproduzidos, programaticamente



Use essas Propriedades

| | |
|--------------------------|-------------------------|
| <code>videoWidth</code> | <code>loop</code> |
| <code>videoHeight</code> | <code>muted</code> |
| <code>currentTime</code> | <code>paused</code> |
| <code>duration</code> | <code>readyState</code> |
| <code>ended</code> | <code>seeking</code> |
| <code>error</code> | <code>volume</code> |



Essas são todas as propriedades do objeto elemento <video>. Algumas você pode definir (como o `loop` e `muted`); outras são para somente leitura (como `currentTime` e `error`).



Pegue esses Eventos

| | |
|-----------------------------|-----------------------------|
| <code>play</code> | <code>ended</code> |
| <code>pause</code> | <code>abort</code> |
| <code>progress</code> | <code>waiting</code> |
| <code>error</code> | <code>loadedmetadata</code> |
| <code>loadedmetadata</code> | <code>volumechange</code> |
| <code>timeupdate</code> | |

Esses são todos os eventos que você pode manipular, se quiser, adicionando event handlers, que são chamados quando o evento que estiver procurando ocorrer.

Um pouco de conteúdo de “programação” na TV Webville

Até agora, só tivemos apenas um vídeo em funcionamento na TV Webville. O que realmente gostaríamos é de uma grade de programação, que servisse uma lista de reprodução de vídeos. Como fazer isso?



- 1 Mostre uma pequena apresentação ao público: os anúncios de pipoca e Coca-Cola, a etiqueta da audiência, e assim por diante...



- 2 Mostre nossa primeira atração, intitulada **Você é popular?** Confie na gente. Você vai gostar.



- 3 Então, mostre nossa apresentação destacada, **Destination Earth**, apresentada com todas as cores. Criada pela American Petroleum Institute, o que diabos eles queriam dizer com isso? Assista e descubra.

Aponte o seu lápis



Agora, você deve estar tentado checar as especificações da marcação <video> para ver como se especifica uma lista de reprodução. Para isso, porém, você precisará de código, porque o elemento <video> permite-lhe especificar apenas um vídeo. Se estivesse numa ilha deserta e tivesse que implementar uma lista de reprodução com apenas um navegador, o elemento <video>, a propriedade src, os métodos load e play e o evento ended, como você faria (você pode usar qualquer tipo de dado JavaScript que quiser)?

Só uma dica: o evento ended acontece quando um vídeo alcança o fim e para de rodar. Como qualquer evento, você pode ter um handler chamado quando isso acontecer.

Não é para olhar a resposta!!

Aponte o seu lápis Solução

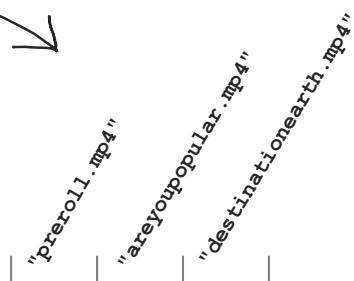
Quando a página carrega, criamos um array de lista de reprodução, iniciamos a reprodução do primeiro vídeo e criamos um event handler para quando parar.



Pseudocódigo da Lista de Reprodução

- Criar array de vídeos de lista de reprodução
- Obter vídeo do DOM
- Definir event handler em vídeo para evento "ended"
-
- Criar posição variável = 0
- Definir fonte de vídeo para posição 0 da lista de reprodução
- Reproduzir o vídeo

Eis como vamos armazenar a lista, como um array. Cada item é um vídeo para ser rodado.



Array da Lista de Reprodução

Este é o nosso handler para lidar com o fim do vídeo.

Ended Event

Toda vez que um vídeo para de reproduzir, o evento ended ocorre...

... o que chama o event handler ended.

Pseudocódigo do Event Handler Ended

- Incrementar a posição por um
- Definir o vídeo para a próxima posição da lista de reprodução
- Reproduzir o próximo vídeo



Quando chegamos ao fim de nossa lista de reprodução, podemos tanto parar quanto acionar um loop até o primeiro vídeo.

Implementando a lista de reprodução da TV Webville

Agora vamos usar o JavaScript e a API de vídeo para implementar a lista de reprodução da TV Webville. Vamos começar adicionando um link a um arquivo JavaScript em webvilletv.html; apenas adicione isto dentro do elemento <head>:

```
<script src="webvilletv.js"></script>
```

E delete este aqui de seu elemento <video>:

```
<video controls autoplay src="video/preroll.mp4" width="480" height="360">
    poster="images/prerollposter.jpg" id="video">
</video> ↗
```

Remova também quaisquer elementos <source>
que você, talvez, esteja experimentando.

Estamos removendo
os atributos `autoplay`
e `src` da tag <video>.

Agora, crie um novo arquivo webvilletv.js e vamos definir algumas variáveis globais e uma função que será chamada quando a página for completamente carregada:

```
var position = 0;
var playlist; ← Precisamos de uma variável para manter o
var video; ← array da lista de reprodução do vídeo.
              ← E também uma variável para manter uma
                  referência ao elemento vídeo.

window.onload = function() {
    playlist = ["video/preroll.mp4", ← Vamos criar nossa lista com três vídeos.
                "video/areyoupopular.mp4",
                "video/destinationearth.mp4"];
    video = document.getElementById("video"); ← Pegue o elemento vídeo.
    video.addEventListener("ended", nextVideo,
    false); ← Adicione um handler para o evento ended do vídeo. Sim,
              parece diferente daquilo com que estamos acostumados —
              espere um pouco, falaremos sobre isso na próxima página.

    video.src = playlist[position]; ← Agora, vamos definir a src para o primeiro vídeo.
    video.load();
    video.play(); ← Depois, carregar o vídeo para reproduzi-lo!
}
```

Então, o que há com aquele código do event handler?

No passado, sempre só designávamos uma função handler para ser chamada, quando um evento ocorria a uma propriedade (como `onload` ou `onclick`), assim:

```
video.onended = nextVideo;
```

No entanto, desta vez, faremos as coisas um pouco diferentes. Por quê? Porque na época em que escrevíamos isso, o suporte para todas as propriedades de evento no objeto vídeo era um pouco escasso. Tudo bem; aquela deficiência irá também nos permitir mostrar a você outra maneira de registrar para eventos: `addEventListener`, que é um método geral suportado por muitos objetos para registrar para vários eventos. Veja como funciona:

Você pode usar o método `addEventListener` para adicionar um event handler.

Esta é a função que vamos chamar quando o evento acontecer.

```
video.addEventListener("ended", nextVideo, false);
```

Este é o objeto no qual estamos ouvindo para o evento.

Este é o evento que estávamos ouvindo. Perceba que não colocamos um "on" antes do nome do evento, como fazemos com os handlers que definimos com propriedades (como `onload`).

O terceiro parâmetro controla alguns métodos avançados de obter eventos se estiver ajustado para `true`. A menos que esteja escrevendo um código avançado, você sempre ajustará isso.

Além do fato do método `addEventListener` ser um pouco mais complicado do que apenas adicionar um handler, ao definir a propriedade a uma função, ele funciona praticamente da mesma maneira. Então, voltemos ao nosso código!

Como escrever o handler “end of video”

Agora, só precisamos escrever o handler para o evento `ended` do vídeo. Este handler será chamado sempre que o player do vídeo atingir o fim do atual arquivo. Veja como escrevemos a função `nextVideo` (adicione-a a `webvilletv.js`):

```
function nextVideo() {
    position++;
    if (position >= playlist.length) {
        position = 0;
    }
    video.src = playlist[position];
    video.load();
    video.play();
}
```

Primeiro, incremente a posição no array da lista de reprodução.

E, se atingirmos o fim da lista, vamos apenas reiniciar o loop novamente ao ajustar a posição para zero.

Agora vamos definir a fonte do player para o próximo vídeo.

E, finalmente, vamos carregar e iniciar a reprodução do vídeo novo.

Note que o handler não será chamado se o usuário pausar o vídeo ou se o vídeo estiver em `looping` (que você pode ativar ajustando a propriedade `loop`).

Outro test drive...



Acredita que já estamos prontos para um test drive? Tudo que fizemos foi usar a API para preparar um vídeo para rodar, então nos certificamos de que tínhamos um event listener pronto para lidar com a situação quando o vídeo finalizasse, o que ele faz iniciando o próximo vídeo na lista de reprodução. Certifique-se de ter feito as modificações em seu arquivo HTML, digite seu novo código JavaScript e faça um test drive.

É isto o que vemos. Sinta-se à vontade para mexer e remexer no vídeo, mudando de um ponto para o outro, sem assistir ao programa inteiro.



Funcional! Como decidimos agora qual formato de vídeo rodar, quando estamos usando o código para carregar a fonte do vídeo?



Boa pergunta.

Quando estamos usando múltiplas tags `source`, podemos contar com o navegador para organizar um ou mais formatos de vídeo e decidir se ele pode tocar algum deles. Agora que estamos usando código, estamos apenas dando ao elemento vídeo uma única opção. Portanto, como testamos para ver o que o browser suporta, para ter certeza se escolhemos o melhor formato?

Fazemos isso usando o método `canPlayType` do objeto `video`. O `canPlayType` leva um formato de vídeo e traz uma string, que representa a confiança do browser em poder tocar aquele tipo de vídeo. Há três níveis de confiança: “probably” (provável), “maybe” (talvez) ou “no confidence” (sem confiança). Vamos dar uma olhada mais de perto e então trabalhar novamente no código da lista de reprodução para usá-lo.

Você está coçando a cabeça, pensando: “Provavelmente? Talvez? Por que ele não retorna `true` ou `false`?”. Nós também pensamos assim, mas veremos o que isso significa num instante...

Como funciona o método canPlayType

O objeto vídeo fornece um método `canPlayType` que pode determinar qual a probabilidade de você ser capaz de reproduzir um formato de vídeo. O método `canPlayType` leva a mesma descrição de formato que você usou com a tag `<source>` e retorna um dos três valores: a string vazia, "maybe" ou "probably". Veja como você chama `canPlayType`:

Se passarmos apenas o formulário resumido de um formato, então poderemos apenas obter "" ou "maybe" como resultado.

```
video.canPlayType("video/ogg")
```

```
video.canPlayType('video/ogg; codecs="theora, vorbis"')
```

Agora, se passarmos o tipo específico com um codec, poderemos então obter "", "maybe" ou "probably" como resposta.

Perceba que o navegador é apenas confiante além de "maybe", se você incluir o parâmetro do codec no tipo. Também note que não há qualquer valor de resposta "I'm absolutely sure" (Tenho total certeza). Mesmo que o browser saiba que pode reproduzir um *tipo* de vídeo, ainda não há garantia que possa rodar o vídeo *em si*; por exemplo, se o bitrate do vídeo for muito alto, o navegador não será capaz de decodificá-lo.

Pondo o canPlayType em uso

Usaremos o `canPlayType` para determinar qual formato de vídeo usar para os vídeos da TV Webville — você já sabe que temos três versões de cada arquivo: MP4, WebM e Ogg e, dependendo de qual browser estiver utilizando, alguns funcionarão, outros não. Vamos criar uma nova função que retorne a extensão de arquivo ("mp4", ".webm" ou ".ogv") que for apropriada para seu browser. Utilizaremos apenas os tipos MIME ("video/mp4", "video/webm" e "video/ogg") e não os codecs; assim, o único retorno possível será "maybe" e a string vazia. Aí vai o código:

```
function getFormatExtension() {
  if (video.canPlayType("video/mp4") != "") {
    return ".mp4";
  } else if (video.canPlayType("video/webm") != "") {
    return ".webm";
  } else if (video.canPlayType("video/ogg") != "") {
    return ".ogg";
}
```

Sabemos que vamos apenas obter "maybe" e string vazia como respostas; então, vamos só nos certificar de que nosso tipo compatível não resulte numa string vazia.

Tentamos cada tipo e retornamos à extensão de arquivo correspondente, se o navegador disser: "Talvez eu possa suportá-lo".

Para a maioria dos casos, se você não souber os codecs, será suficiente a confiança "maybe".

Integrando a função getFormatExtension

Agora, precisamos fazer algumas mudanças nas funções window.onload e nextVideo para usar getFormatExtensions. Primeiro, removeremos as extensões de arquivo dos nomes de arquivo na lista de reprodução (pois iremos calculá-las usando o getFormatExtension no lugar), daí chamaremos getFormatExtension, onde definiremos a propriedade video.src:

```
window.onload = function() {
    playlist = ["video/preroll",
                "video/areyoupopular",
                "video/destinationearth"];
    video = document.getElementById("video");
    video.addEventListener("ended", nextVideo, false);
    video.src = playlist[position] + getFormatExtension();
    video.load();
    video.play();
}
```

← Remova as extensões de arquivo. Vamos calculá-las programaticamente agora.

← E concatenar o resultado de getFormatExtension com o nome de arquivo para a nova video src.

Faça a mesma coisa em nextVideo:

```
function nextVideo() {
    position++;
    if (position >= playlist.length) {
        position = 0;
    }
    video.src = playlist[position] + getFormatExtension();
    video.load();
    video.play();
}
```

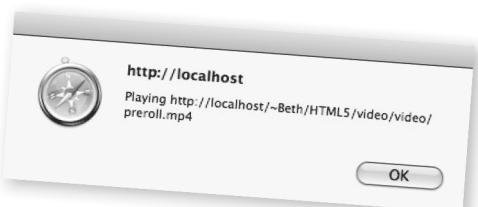
← O mesmo aqui; concatenamos o resultado de getFormatExtension com a video src.

E test drive... 

Adicione a função canPlayType e faça as modificações acima, daí recarregue seu arquivo webvilletv.html. Funciona? Agora seu código está calculando o melhor formato. Se quiser saber qual vídeo o browser escolheu, tente adicionar um alerta nas funções window.onload e nextVideo; adicione-o ao fim de cada função, depois do video.play():

```
alert("Playing " + video.currentSrc);
```

Qual arquivo seu browser reproduz?



não existem Perguntas Idiotas

P: Se estou definindo programaticamente a fonte de meu vídeo e o canPlayType diz que é um “maybe”, mas ainda assim a reprodução falha, como faço para lidar com isso?

R: Existe uma porção de maneiras de abordar isso. Uma delas é pegar o erro e dar ao objeto vídeo outra fonte (falaremos sobre pegar erros no fim deste capítulo). A outra é usar o DOM para escrever múltiplas tags source

dentro do objeto vídeo de uma vez (como se tivesse digitado dentro da sua marcação). Dessa forma, seu objeto vídeo possuirá várias escolhas e você não terá de escrever códigos mais complexos de manipulação de erro. Não faremos isso neste capítulo, mas é uma forma de oferecer ao seu objeto vídeo múltiplas possibilidades e fazê-lo por intermédio do código, não pela marcação, se possuir necessidades avançadas.



Veja bem!

Talvez precise instalar o Quicktime para rodar vídeo mp4 no

Safari.

O Quicktime frequentemente vem instalado por padrão, mas, se não, você pode baixá-lo a partir de <http://www.apple.com/quicktime/download/>.



Veja bem!

O Google Chrome possui restrições extras de segurança.

Essas restrições de segurança irão prevenir que você faça algumas operações vídeo + canvas, se tiver carregado a página como um arquivo (i.e., sua URL mostrará file:/// em vez de http://), como faremos pelo resto do capítulo. Se tentar, o aplicativo não funcionará e não terá nenhuma indicação do porquê.

Portanto, para este capítulo, recomendamos tanto usar um navegador diferente, quanto rodar seu próprio servidor e os exemplos de <http://localhost>.



Veja bem!

Certifique-se de que seu servidor está servindo arquivos de vídeo com o tipo MIME correto.

Se estiver usando seu próprio servidor local, ou rodando um aplicativo usando vídeo de um servidor hospedado, você precisará se certificar de que os vídeos estão sendo servidos corretamente. Do contrário, eles poderão não funcionar apropriadamente.

Se estiver num Mac ou Linux, usando um servidor local, é mais provável que esteja usando o Apache. Pode-se modificar o arquivo `httpd.conf` (se tiver acesso-raiz) ou criar um arquivo `.htaccess` no diretório onde seus arquivos de vídeo são armazenados e adicionar as seguintes linhas:

```
AddType video/ogg .ogv
AddType video/mp4 .mp4
AddType video/webm .webm
```

Isso diz ao servidor como servir arquivos com aquelas extensões.

Você pode instalar o Apache no Windows e fazer o mesmo. Para servidores IIS, recomendamos vasculhar a documentação online da Microsoft para “Configurando tipos MIME em IIS”.

Vivo lhe dizendo que não é só uma questão de JavaScript... Você tem de ver o todo. Construir aplicativos web tem a ver com marcação, CSS e JavaScript com todas as suas APIs.



Em algum momento, teremos de tratá-lo como um verdadeiro desenvolvedor.

Neste livro, nós (com sorte) o ajudamos em cada passo de seu caminho — estivemos lá para segurá-lo antes de cair e ter certeza de que, em seu código, estivessem postos todos os pingos nos i's. Contudo, parte do que é ser um verdadeiro desenvolvedor é se arriscar, ler o código dos outros, desbravar a floresta independentemente das árvores e desvendar a complexidade de como tudo se comunga.

Durante o resto deste capítulo, vamos começar a deixá-lo fazer isso. Em seguida, teremos um exemplo que é o mais próximo de um verdadeiro aplicativo web que já vimos até agora, com suas várias peças, vários usos e códigos API que lidam com vários detalhes reais. Agora, não podemos acompanhá-lo em cada passo, explicando todas as nuances, como normalmente fazemos (ou este livro teria 1200 páginas); e nem queremos isso, pois você também precisa adquirir a habilidade de juntar as peças *sem nós*.

Não se preocupe. Ainda estaremos aqui e vamos lhe dizer o que cada coisa faz, mas queremos que você comece a aprender como fazer o código, lê-lo, analisá-lo e, então, aumentá-lo e alterá-lo para fazer *o que você quiser*. Pelos próximos três capítulos, queremos que mergulhe nesses exemplos, estude-os e tenha os códigos de cabeça. Sério... Você está pronto!

Precisamos da sua ajuda!

Veja só... Temos um contrato para construir o software **Starring You Video** para a nova cabine de vídeo deles. O que diabos é isso? Ah, só a mais recente cabine de vídeo-mensagens habilitada por HTML5 — um cliente entra numa cabine de vídeo fechada e grava sua própria mensagem de vídeo. Eles podem, então, aprimorar seus vídeos utilizando efeitos cinematográficos reais; há um filtro sépia de antigos filmes de faroeste, outro filtro de filmes noir em preto-e-branco, ou até mesmo um filtro alienígena para filmes de ficção científica. Daí, o cliente pode enviar sua mensagem a um amigo. Fomos adiante e nos comprometemos em construir a interface e os efeitos de vídeo, processando sistemas para eles.

Existe, porém, um problema. As cabines de vídeo não estarão disponíveis por cerca de seis semanas e, quando eles chegarem, o código tem que estar pronto. Portanto, nesse intervalo, vamos ter uma unidade demo parcialmente funcional e alguns arquivos de vídeo para teste. Assim, escreveremos todos os nossos códigos. Então, quando tivermos terminado, o pessoal da Starring You poderá apontar o código para o recém-gravado vídeo. Lembre-se, é claro, de que tudo isso tem de ser feito com HTML5.

Esperamos que você esteja dentro, pois já assinamos o contrato!

Entre, grave um vídeo, estilize-o
e envie-o para seus amigos!



Aproxime-se da cabine, vamos dar uma olhada...

Abaixo, você verá nossa unidade demo completa com uma interface de usuário. O que temos é uma tela de vídeo, na qual os usuários verão seus vídeos reproduzidos. Serão capazes de aplicar um filtro, como “faroeste antigo” ou “ficção científica”, verem como fica e, quando estiverem felizes, enviarem-no a um amigo. Nós ainda não temos capacidade para gravação. Temos os vídeos de teste para poder reproduzir. Nosso primeiro trabalho será conectar tudo para que os botões funcionem e, então, escrever os filtros de vídeo. Antes de começarmos, verifique a interface:

Aqui está a interface da unidade demo. Tem um player de vídeo bem no meio para a visualização dos vídeos.



Aplique seu efeito favorito: faroeste antigo (sépia), filme noir (bem escuro) ou ficção científica (vídeo invertido).

Os controles play, pause, loop e mute.

Escolha um vídeo teste. Nossa unidade demo possui dois para escolher.

Descompactando a Unidade Demo

A unidade demo acabou de chegar pelo correio e está na hora de desempacotá-la. Parece que temos uma unidade em funcionamento, com um pouco de marcação HTML & JavaScript escrito até agora. Vamos dar uma olhada no HTML primeiro (`videobooth.html`). A propósito, sente-se. Temos algumas páginas de código de origem para checar, para depois arrasarmos no código *de verdade*.

```

<!doctype html>
<html lang="en">
<head>
    <title>Starring YOU Video Booth</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="videobooth.css">
    <script src="videobooth.js"></script>
</head>
<body>
    <div id="booth">
        <div id="console">
            <div id="videoDiv">
                <video id="video" width="720" height="480"></video>
            </div>
            <div id="dashboard">
                <div id="effects">
                    <a class="effect" id="normal"></a>
                    <a class="effect" id="western"></a>
                    <a class="effect" id="noir"></a>
                    <a class="effect" id="scifi"></a>
                </div>
                <div id="controls">
                    <a class="control" id="play"></a>
                    <a class="control" id="pause"></a>
                    <a class="control" id="loop"></a>
                    <a class="control" id="mute"></a>
                </div>
                <div id="videoSelection">
                    <a class="videoSelection" id="video1"></a>
                    <a class="videoSelection" id="video2"></a>
                </div>
            </div>
        </div>
    </div>
</body>
</html>

```

HTML5, é claro.

E toda a estilização está pronta para nós! Olha aqui o arquivo CSS.

E aqui está o arquivo JavaScript que vamos precisar para escrever a maior parte disso. Vamos olhar mais profundamente, mas parece que eles acabaram de escrever o código para controlar os botões na interface, até agora...

Aqui está a interface principal; temos o console em si, o qual parece que está dividido entre o display e o painel do vídeo, com três arranjos de botões agrupados em "effects" (efeitos), "controls" (controles) e "videoSelection".

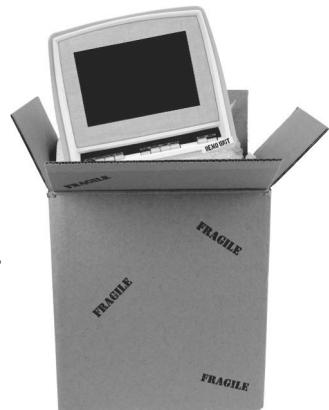
Eles até têm um videoplayer instalado... ótimo, vamos precisar disso.

Aqui estão todos os efeitos.

Essas são apenas âncoras HTML. Veremos como nos prendemos a elas num segundo.

E os controles do player.

E os dois vídeos demo para se testar.



Investigando o resto do código de origem

Vamos agora dar uma olhada em todo o código JavaScript que chegou da fábrica (origem), incluindo o código que cria os botões (que acabamos de ver no HTML) e o código para cada handler de botões (os quais, no momento, só asseguram que os botões corretos sejam pressionados). Revisaremos tudo antes de começarmos a adicionar nosso novo código.



E agora o JavaScript...

Então, vamos abrir o JavaScript (`videobooth.js`). Parece que todos os botões de interface funcionam. Eles não fazem nada de interessante ainda, porém é importante que nós entendamos como são criados, porque os botões invocarão o código que temos de escrever (por exemplo reproduzir um vídeo ou visualizá-lo com um efeito de filtro).

Abaixo, você encontrará a função que é invocada quando a página é carregada. Para cada conjunto de botões (efeitos, controles e a seleção de vídeo), o código perambula pelos botões e designa click handlers aos links âncora. Vamos dar uma olhada:

```
window.onload = function() {
    var controlLinks = document.querySelectorAll("a.control");
    for (var i = 0; i < controlLinks.length; i++) {
        controlLinks[i].onclick = handleControl;
    }

    var effectLinks = document.querySelectorAll("a.effect");
    for (var i = 0; i < effectLinks.length; i++) {
        effectLinks[i].onclick = setEffect;
    }

    var videoLinks = document.querySelectorAll("a.videoSelection");
    for (var i = 0; i < videoLinks.length; i++) {
        videoLinks[i].onclick = setVideo;
    }

    pushUnpushButtons("video1", []);
    pushUnpushButtons("normal", []);
}

function handleControl() {
    // ...
}

function setEffect() {
    // ...
}

function setVideo() {
    // ...
}
```

Esta é a função invocada, quando a página é completamente carregada. Cada um, pois a indicação passa pelos elementos de um grupo de botões.

Os handlers onclick, para cada botão dos controles do player, são definidos ao handler handleControl.

E o handler para efeitos está definido como setEffect.

E, finalmente, o handler para seleção de vídeo é definido como setVideo.

Uma vez que fizemos todo esse trabalho de base, usamos uma função de auxílio para pressionar visualmente o botão "video1", e o botão "normal" (sem filtro) na interface.

Você não viu `document.querySelectorAll` antes; é semelhante a `document.getElementByIdByName` só que você seleciona elementos que combinam um seletor CSS. O método retorna um array de objetos elemento que combinam o argumento seletor de CSS.

```
var elementArray = document.querySelectorAll("selector");
```

Vamos ver os handlers de botões

Ok, até aqui o código JavaScript deu conta de criar todos os botões de forma que, se clicados, o handler apropriado seja chamado. Em seguida, vamos dar uma olhada nos handlers reais, começando pelo handler para os botões do player (play, pause, loop e mute), para ver o que eles estão fazendo:



A primeira coisa que faremos neste handler é ver quem nos chamou, resgatando a id do elemento que invocou o handler.

```
function handleControl(e) {
    var id = e.target.getAttribute("id");

    if (id == "play") {
        pushUnpushButtons("play", ["pause"]);
    } else if (id == "pause") {
        pushUnpushButtons("pause", ["play"]);
    } else if (id == "loop") {
        if (isButtonPushed("loop")) {
            pushUnpushButtons("", ["loop"]);
        } else {
            pushUnpushButtons("loop", []);
        }
    } else if (id == "mute") {
        if (isButtonPushed("mute")) {
            pushUnpushButtons("", ["mute"]);
        } else {
            pushUnpushButtons("mute", []);
        }
    }
}
```

↑
Todo o código até agora foi pura perfumaria: só modifica a aparência dos botões quando pressionados ou não. Não há código para fazer qualquer coisa real, como reproduzir um vídeo. Isso é o que teremos de escrever.

Uma vez que sabemos a id, sabemos se o elemento foi play, pause, loop ou mute.

Dependendo de qual botão foi pressionado, alteraremos a interface para refleti-lo. Por exemplo, se o pause foi pressionado, então o play não deve ser.

Estamos usando uma função de auxílio para ter certeza de que os estados do botão onscreen estejam sendo cuidados. Ela se chama pushUnpushButtons e leva um botão pressionado com um array de botões não pressionados e atualiza a interface para refletir aquele estado.

Vários botões têm diferentes semânticas. Por exemplo, play e pause são como botões de rádio (apertando um, faz levantar o outro), enquanto que mute e loop são como botões de alternância.

Agora está tudo ótimo e tal, mas onde entra *nossa código*? Vamos analisar isso: quando um botão, como o play, é pressionado, não será nossa única preocupação atualizar a interface (o que o código já faz). Vamos adicionar também um pouco de código que realmente *faça alguma coisa*, como o vídeo começar a tocar. Vamos seguir em frente, observar os outros dois handlers (para definir os efeitos do vídeo e para definir o vídeo de testes), e deverá ser meio óbvio (se já não é) onde nosso código ficará...

Os handlers `setEffect` e `setVideo`

Vamos dar uma olhada nos outros dois handlers. O handler `setEffect` manipula sua escolha de efeito, como sem efeito (normal), faroeste, filme noir ou ficção científica. Da mesma forma, o handler `setVideo` manipula sua escolha de vídeo de teste um ou dois. Seguem abaixo:



```
function setEffect(e) {
    var id = e.target.getAttribute("id");
    if (id == "normal") {
        pushUnpushButtons("normal", ["western", "noir", "scifi"]);
    } else if (id == "western") {
        pushUnpushButtons("western", ["normal", "noir", "scifi"]);
    } else if (id == "noir") {
        pushUnpushButtons("noir", ["normal", "western", "scifi"]);
    } else if (id == "scifi") {
        pushUnpushButtons("scifi", ["normal", "western", "noir"]);
    }
}
```

Este funciona igual ao handler `handleControl`: pegaremos a id do elemento que nos chamou (o botão que foi clicado) e então atualizaremos a interface de acordo.

E aqui será onde nosso código ficará.

Adicionaremos código para cada caso que manipular a implementação do filtro de efeito especial apropriado.

E o mesmo acontecerá com `setVideo`; veremos qual botão foi pressionado e atualizaremos a interface.

```
function setVideo(e) {
    var id = e.target.getAttribute("id");
    if (id == "video1") {
        pushUnpushButtons("video1", ["video2"]);
    } else if (id == "video2") {
        pushUnpushButtons("video2", ["video1"]);
    }
}
```

Adicionaremos código também aqui para implementar a alternância entre os dois vídeos de teste.

E aqui estão as funções de auxílio

E para que haja completude (ou se você estiver num voo de 11 horas para Fiji, sem internet e realmente quiser digitar isso tudo):

`pushUnpushButtons` preocupa-se com os estados do botão.

Os argumentos são as ids de um botão para pressionar e de um ou mais botões para despressionar um array.

```
function pushUnpushButtons(idToPush, idArrayToUnpush) {
    if (idToPush != "") {
        var anchor = document.getElementById(idToPush);
        var theClass = anchor.getAttribute("class");
        if (!theClass.indexOf("selected") >= 0) {
            theClass = theClass + " selected";
            anchor.setAttribute("class", theClass);
        }
        var newImage = "url(images/" + idToPush + "pressed.png)";
        anchor.style.backgroundImage = newImage;
    }
    for (var i = 0; i < idArrayToUnpush.length; i++) {
        anchor = document.getElementById(idArrayToUnpush[i]);
        theClass = anchor.getAttribute("class");
        if (theClass.indexOf("selected") >= 0) {
            theClass = theClass.replace("selected", "");
            anchor.setAttribute("class", theClass);
        }
        anchor.style.backgroundImage = "";
    }
}

function isButtonPushed(id) {
    var anchor = document.getElementById(id);
    var theClass = anchor.getAttribute("class");
    return (theClass.indexOf("selected") >= 0);
}
```

E lembre-se de que se não quiser digitar, você pode conseguir todo o código em <http://wickedlysmart.com/hfhtml5>.

Primeiro, cheque para verificar se a id do botão de pressionar não está vazia.

Pegue o elemento âncora usando aquela id...

... e obtenha o atributo class.

Nós "pressionamos" o botão, adicionando a classe "selected" à âncora, e...

Para despressionar os botões, nós passamos pelo array de ids para despressionar, pegando cada âncora...

... certifique-se de que o botão está realmente pressionado (se estiver, terá uma classe "selected")...

... remova "selected" da classe...

... e remova a imagem de background; assim, veremos o botão despressionado.

isButtonPushed simplesmente verifica para ver se um botão está pressionado. Pega a id de uma âncora...

... pega a âncora...

... obtém a classe daquela âncora...

... e retorna true (verdadeiro) se a âncora tiver a classe "selected".

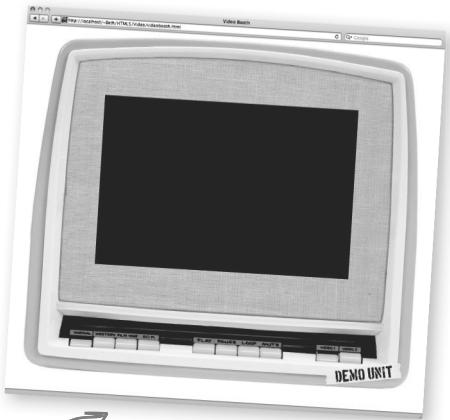


Aquele cheiro de máquina demo nova... hora do test drive!



Ainda não escrevemos muito código, mas estamos lendo e entendendo ele, e isso é tão bom quanto. Então, carregue o arquivo `videobooth.js` dentro de seu browser e verifique os botões. Teste-os bem. Para melhorar, adicione alguns alertas dentro das funções do handler. Tenha uma boa sensação por saber como isso está funcionando. Quando voltar, começaremos a escrever um pouco de código para fazer com que os botões comecem a funcionar de verdade.

Experimente todos os botões, percebendo que alguns são como botões de rádio, outros como botões de alternância.

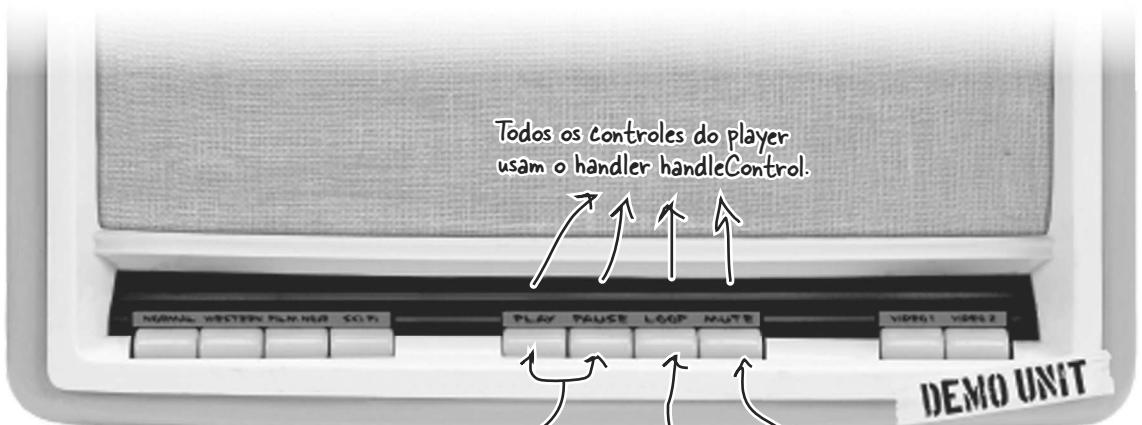


Aponte o seu lápis



Você encontrará a solução em algumas páginas...

Marque os botões abaixo para distinguir se são de alternância (independentes) ou de rádio (apertando um, salta o outro). Também anote cada botão com seu click handler correspondente. Fizemos alguns para você:



Os botões de rádio play e pause não podem ser selecionados ao mesmo tempo.

Loop e Mute são botões de alternância, portanto podem ser usados independentemente de quaisquer outros botões.

Acho que perdi alguma coisa...
Como você saiu de <div>s com
tags âncora para chegar a ter
botões na interface?

Este é o poder do CSS.

É uma pena este livro não ser *Use a Cabeça! Programação HTML5 com JavaScript & CSS*, mas, se assim fosse, teria 1.400 páginas, não é mesmo? É claro, poderíamos receber o convite para escrevermos um livro de CSS avançado...

Sem brincadeiras, este é o poder da marcação para estrutura e CSS para apresentação (e se esse é um tópico novo para você, verifique *Use a Cabeça! HTML com CSS & XHTML*).

O que estamos fazendo não é tão complexo; aqui está, resumidamente, para matar a curiosidade:

Ajustamos a imagem do background do console <div> para o console da cabine (sem botões).

O elemento <video> está numa <div> que é posicionada em relação ao console. Então, o elemento <video> é absolutamente posicionado de forma que fique no meio do console.

Posicionamos o painel <div> em relação ao console e, depois, posicionamos a <div> para cada grupo de botões, relativamente ao painel.

Cada <div> de grupo de botão fica com uma imagem de background para todos os botões não pressionados.

Cada “botão” âncora é posicionado dentro da <div> para o grupo e recebe uma largura e altura para combinar com seu botão correspondente. Quando você clica num “botão”, dá àquela âncora a imagem de background de um botão pressionado para cobrir o não pressionado.

Se quiser, verifique o CSS em detalhes: `videobooth.css`.





Aponte o seu lápis Solução

Marque os botões abaixo para distinguir se são de alternância (independentes) ou de rádio (apertando um, salta o outro). Também anote cada botão com seu click handler correspondente. Veja nossa solução:

The diagram shows a 'DEMO UNIT' with several rows of controls. Arrows point from text labels to specific buttons.

- Todos os botões de efeito usam o handler setEffect.** (Top row, four buttons)
- Todos os controles do player usam o handler handleControl** (Second row, four buttons)
- Todos os botões de seleção de vídeo usam o handler setVideo.** (Third row, two buttons)
- Todos esses são botões de rádio; permitimos apenas um efeito por vez para ser aplicado ao vídeo.** (Row below first, four buttons)
- Botões de rádio, play e pause não podem ser selecionados ao mesmo tempo.** (Row below second, three buttons)
- Loop e Mute são botões de alternância; eles podem ser usados independentemente de quaisquer outros botões.** (Row below third, two buttons)
- Os seletores de vídeo são botões de rádio também; podemos assistir apenas um vídeo por vez.** (Bottom right, two buttons)

Preparando nossos vídeos demo...

Antes de implementarmos os controles de botões, precisamos do vídeo para testá-lo e, como você pode ver a partir dos botões, a Starring You Video mandou-nos dois demos. Vamos logo criar um objeto para deixarmos dois vídeos e depois adicionaremos um pouco de código para nosso handler onload, para criar a fonte do objeto vídeo (assim como fizemos para a TV Webville).

Criaremos este objeto para deixarmos os dois demos. Voltaremos para explicar isso em breve.

```
var videos = {video1: "video/demovideo1", video2: "video/demovideo2"};
```

```
window.onload = function() {
```

```
    var video = document.getElementById("video");
    video.src = videos.video1 + getFormatExtension();
    video.load();
```

Aqui, estamos obtendo o elemento vídeo e definindo sua fonte ao primeiro vídeo no array com uma extensão executável.

Depois, vamos em frente e carregamos o vídeo para que, se o usuário clicar em play, esteja pronto para rodar.

```
    var controlLinks = document.querySelectorAll("a.control");
```

```
    for (var i = 0; i < controlLinks.length; i++) {
        controlLinks[i].onclick = handleControl;
    }
```

```
    var effectLinks = document.querySelectorAll("a.effect");
    for (var i = 0; i < effectLinks.length; i++) {
        effectLinks[i].onclick = setEffect;
    }
```

```
    var videoLinks = document.querySelectorAll("a.videoSelection");
    for (var i = 0; i < videoLinks.length; i++) {
        videoLinks[i].onclick = setVideo;
    }
```

```
    pushUnpushButtons("video1", []);
    pushUnpushButtons("normal", []);
}
```

Antes, porém, de ficar relaxado, lembre-se de que a função `getFormatExtension` está na TV Webville, não neste código! Então, abra `webvilletv.js` e copie e cole a função dentro do seu código da cabine de vídeo. Outra coisa: no código da cabine de vídeo, não há um objeto de vídeo global. Acrescente esta linha ao topo de sua função `getFormatExtension` para compensar isso:

```
var video = document.getElementById("video");
```

LEIA ISTO
CUIDADOSAMENTE!

Acrescente esta linha no topo de sua função `getFormatExtension`.

Implementando seus controles de vídeo

Certo, hora de trabalhar com aqueles botões! Agora, é importante ressaltar que, para este projeto, vamos implementar *nossos próprios* controles de vídeo. Isto é, em vez de usar controles embutidos, vamos controlar a experiência por nós mesmos — quando os usuários precisarem dar play, pause ou mute no vídeo, ou mesmo dar um loop na reprodução, eles usarão nossos botões personalizados, não os embutidos. Também significa que vamos fazer tudo isso *programaticamente* pela API. Contudo, não vamos fazer todo o processo, o que significaria implementar nosso próprio video scrubber ou, talvez, botões de avançar e retroceder, porque não fariam sentido nesse aplicativo, *mas poderíamos, se precisássemos*. Você verá que, só por implementar nosso pequeno painel de controle, entenderá o cerne da questão, estando em perfeita forma para dar prosseguimento nisso, se quiser.

Portanto, vamos começar. Que tal iniciar pelo botão play e ir para a direita (para o pause, depois loop e então mute)? Então, encontre o handler handleControl e adicione este código:

```
function handleControl(e) {
    var id = e.target.getAttribute("id");
    var video = document.getElementById("video"); ← Precisamos de uma referência ao objeto vídeo.

    if (id == "play") {
        pushUnpushButtons("play", ["pause"]);
        if (video.ended) {
            video.load();
        }
        video.play(); ← Este deve ser bem simples.
    } else if (id == "pause") { ← Se o usuário pressionar play,
        pushUnpushButtons("pause", ["play"]); ← então chame o método play
                                                no objeto vídeo.

    } else if (id == "loop") {
        if (isButtonPushed("loop")) {
            pushUnpushButtons("", ["loop"]);
        } else {
            pushUnpushButtons("loop", []);
        }
    } else if (id == "mute") {
        if (isButtonPushed("mute")) {
            pushUnpushButtons("", ["mute"]);
        } else {
            pushUnpushButtons("mute", []);
        }
    }
}
```

Há uma situação duvidosa aqui prestes a nos pegar e precisamos dar conta dela: se tivéssemos reproduzido um vídeo e o deixado tocar até o fim, para tocá-lo novamente desde o início, teríamos de carregá-lo de novo, em primeiro lugar. Checamos para ter certeza de que o vídeo foi até o fim (e não estava simplesmente pausado), pois só vamos querer carregá-lo novamente naquele caso. Se estiver pausado, podemos apenas reproduzir sem carregar.



Agora, vamos implementar todos esses botões.

Implementando o resto dos controles do vídeo

Vamos dar um jeito logo no resto dos controles — eles são tão diretos que praticamente se escrevem sozinhos:

```
function handleControl(e) {
    var id = e.target.getAttribute("id");
    var video = document.getElementById("video");

    if (id == "play") {
        pushUnpushButtons("play", ["pause"]);
        video.load();
        video.play();
    } else if (id == "pause") {
        pushUnpushButtons("pause", ["play"]);
        video.pause();
    } else if (id == "loop") {
        if (isButtonPushed("loop")) {
            pushUnpushButtons("", ["loop"]);
        } else {
            pushUnpushButtons("loop", []);
        }
        video.loop = !video.loop;
    } else if (id == "mute") {
        if (isButtonPushed("mute")) {
            pushUnpushButtons("", ["mute"]);
        } else {
            pushUnpushButtons("mute", []);
        }
        video.muted = !video.muted;
    }
}
```

Se o usuário pausa o vídeo, então use o método pause do objeto vídeo.

Para o loop, temos uma propriedade booleana chamada loop no objeto vídeo. Apenas a ajustamos apropriadamente...

... e, para, isso vamos mantê-lo desperto usando o operador booleano "!" (not), que lança os valores booleanos para nós.

E o mute funciona da mesma maneira: somente lançamos o valor atual da propriedade mute, quando o botão for pressionado.

Outro test drive!

Certifique-se de que todas as mudanças no código foram digitadas. Carregue o `videobooth.html` em seu navegador e teste seus botões de controle. Você deverá ver o vídeo começar a reproduzir, ser capaz de pausá-lo, emudecê-lo, ou mesmo pô-lo num loop. Claro que você não pode selecionar outro vídeo demo ainda, ou adicionar um efeito, mas já vamos chegar lá!



Resolvendo um probleminha...

Há um pequeno detalhe que precisamos resolver para, realmente, fazer com que esses botões funcionem como deveriam. Veja um caso real: digamos que você esteja rodando um vídeo e o loop não está selecionado. Daí o vídeo vai até o fim e para. Como temos coisas implementadas agora, o botão play permanecerá na posição de pressionado. Não seria melhor se ele saltasse novamente, pronto para ser pressionado?

Usando eventos podemos facilmente fazer isso. Comecemos por adicionar um listener para o evento ended. Adicione este código ao fim de seu handler onload:

```
video.addEventListener("ended", endedHandler, false);
```

Agora, vamos escrever o handler que será chamado qualquer hora em que o vídeo pare sua reprodução ao chegar ao fim:

```
function endedHandler() {  
    pushUnpushButtons("", ["play"]);  
}
```



Nosso botão play
não está 100%
certo ainda...

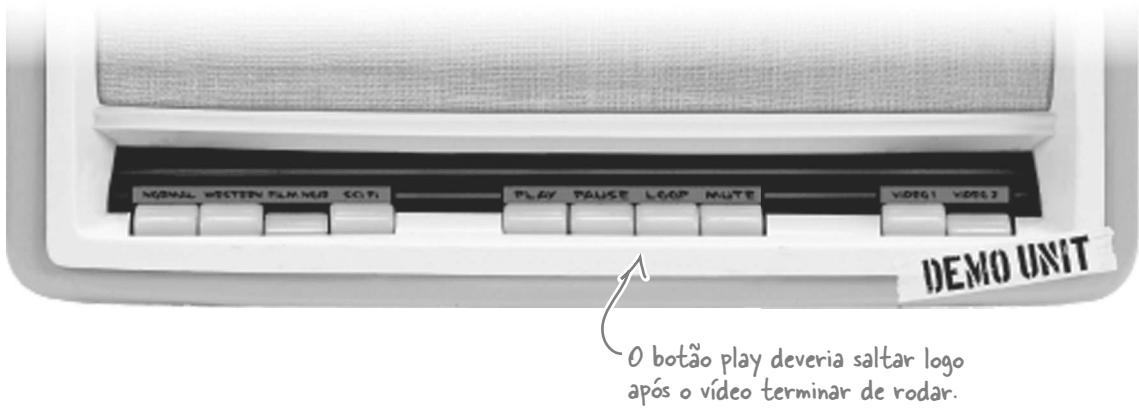
Defina um handler
para o evento "ended",
que é chamado quando
a reprodução do vídeo
termina (mas NÃO
quando é pausada!).

Tudo o que
precisamos fazer
é "despressionar" o
botão play. Pronto!

E outro...



Ok, faça as mudanças, salve o código e recarregue. Agora, comece um vídeo e deixe-o rodar até terminar, sem o botão loop estar pressionado e, ao fim, você deverá ver o botão play levantar sozinho.



O botão play deveria saltar logo
após o vídeo terminar de rodar.

Alternando vídeos de teste

Já acrescentamos um objeto para segurar nossos dois vídeos de teste e até temos dois botões para selecionar entre eles. Cada botão está designado ao handler `setVideo`. Vamos continuar nosso trabalho de escrever isso agora, para que possamos alternar entre nossos vídeos:



Este é o nosso objeto com os dois vídeos.

Estamos mostrando novamente como um lembrete para que você veja como vamos usá-lo...

```
var videos = {video1: "video/demovideo1", video2: "video/demovideo2"};
```

```
function setVideo(e) {
```

← E aqui é o handler de novo.

```
var id = e.target.getAttribute("id");
```

De novo, precisamos de uma referência ao objeto vídeo.

```
var video = document.getElementById("video");
```

```
if (id == "video1") {
```

Depois, ainda atualizamos os botões da mesma maneira que estavam, sem mudanças aí.

```
pushUnpushButtons("video1", ["video2"]);
```

```
    } else if (id == "video3") {
```

```
pushUnpushButtons("video2", ["video1"]);
```

1

```
video.src = videos[id] + getFormatExtension();
```

```
video.load();
```

```
video.play();
```

```
pushUnpushButtons ("play", ["pause"]);
```

Então, usamos a id fonte do botão (o atributo id da âncora) para pegar o correto nome do arquivo do vídeo e adicioná-lo à nossa extensão amigável ao navegador.

Percoba que estamos usando a notação [] com nosso objeto vídeos, utilizando a string id como nome da propriedade.

Uma vez que temos o caminho do vídeo e nome do arquivo corretos, carregamos e executamos o vídeo.

E nos certificamos de que o botão play seja pressionado, pois começamos a execução do vídeo quando o usuário clica numa nova seleção.

Mude de motorista e test drive!



Faça essas mudanças na sua função `setVideo`, e carregue sua página novamente. Você deverá ser capaz agora de alternar facilmente entre as fontes de vídeo.





Confidencial HTML5

Entrevista da semana:

Confissões do Elemento Vídeo

Use a Cabeça!: Bem-vindo, Vídeo. Vou direto ao assunto que está na cabeça de todo mundo, que, no caso, seria VOCÊ e o elemento Canvas.

Vídeo: Isso quer dizer o quê?

Use a Cabeça!: Parece que foram vistos à noite pela cidade, tomaram café da manhã juntos. Preciso dizer mais?

Vídeo: O que há para dizer? Canvas e eu temos um ótimo relacionamento. Ela mostra o conteúdo de uma maneira muito, digamos, visualmente atraente, e eu sou um burro de carga de vídeos. Saio mastigando os codecs e levo o conteúdo do vídeo ao navegador.

Use a Cabeça!: Bem, não é onde eu queria chegar, mas vamos por aí. Ok, ela é ótima num display 2D, você é ótimo num display de vídeo. Então? Qual a verdadeira relação?

Vídeo: Como qualquer relacionamento, quando você junta duas coisas e obtém mais do que a soma das partes, aí é quando vê que existe algo especial.

Use a Cabeça!: Ok, bem, consegue ser mais direto?

Vídeo: É um conceito simples: se quiser fazer algo além da simples reprodução do vídeo — isto é, se quiser processar qualquer coisa em seu vídeo, ou personalizar overlays

ou mostrar múltiplos vídeos de uma só vez — então você vai querer usar a Canvas.

Use a Cabeça!: Tudo isso parece ótimo, mas o vídeo requer um processamento pesado, digo, é uma grande quantidade de dados. Como o JavaScript, uma linguagem de script, vai fazer qualquer coisa real? Escrever JavaScript não é como escrever numa língua nativa.

Vídeo: Ah, você ficaria surpreso... Você já viu as últimas melhorias no JavaScript? Já são rápidos e ficam mais rápidos a cada dia que passa. Os mais brilhantes jóqueis de máquinas virtuais da indústria já estão resolvendo os problemas e chutando alguns traseiros por aí.

Use a Cabeça!: Sim, mas vídeo? Sério?

Vídeo: Sério.

Use a Cabeça!: Pode nos dar alguns exemplos de coisas que vocês podem fazer com JavaScript, canvas e vídeo?

Vídeo: Claro, você pode processar vídeo em tempo real, inspecionar as suas características, obter dados a partir de frames de vídeo e alterar os dados do vídeo, digamos... rodando-o, dimensionando-o ou, até mesmo, modificando seus pixels.

Use a Cabeça!: Pode nos ajudar a entender como isso pode ser feito em código?

Vídeo: Oh, vou ter de deixar essa para a próxima. Estou recebendo uma ligação da Canvas... Tenho que correr...

Está na hora de alguns efeitos especiais

Já não é hora de acrescentarmos aqueles efeitos especiais no filme? O que queremos é pegar nosso vídeo original e poder aplicar efeitos, como filme noir, faroeste ou de ficção científica. Se observar a API do vídeo, você não encontrará qualquer método de efeitos lá, ou quaisquer maneiras de adicioná-los diretamente. Então, como faremos isso?

Fique à vontade para pensar em como poderemos adicionar efeitos aos nossos vídeos. Não se preocupe com o fato de ainda não saber processar vídeo; apenas pense por cima nos designs.



Queremos pegar
nossa vídeo original
e sermos capazes de
aplicar efeitos de
filme noir, faroeste
e ficção científica.

Starring you Video

Notas da Engenharia...

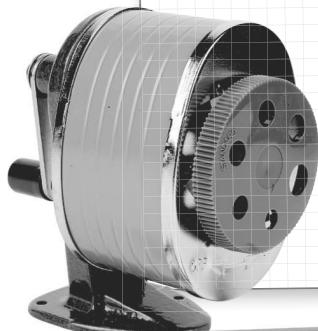
Use esta nota da engenharia para fazer um desenho, etiquetá-lo, ou escrever um pouco de pseudocódigo para qualquer código de seus efeitos do vídeo. Pense nisso como um aquecimento, só para ligar um pouco o cérebro...

Como você colocará as mãos
nos pixels que formam
cada frame do vídeo?

Uma vez que possui os pixels, como
você os processa para aplicar
os efeitos?

Digamos que você fosse escrever
uma função para implementar cada
efeito, como ela pareceria?

Como poderá mostrar o vídeo,
uma vez que já processou todos
os seus pixels para aplicar
os efeitos?



Suas ideias aqui.



O plano FX¹

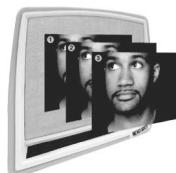
Ainda não sabemos exatamente como implementar os efeitos, mas aí vai um plano de ataque de alto nível:

- 1 Sabemos que ainda precisamos interligar aqueles botões ao controle de efeitos. Então, faremos isso primeiro.



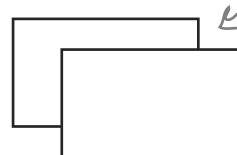
Os botões
ainda precisam
estar ligados.

- 2 Vamos aprender um pouco sobre processamento de vídeo e verificar a técnica “scratch buffer” para adicionar nossos efeitos.
- 3 Vamos implementar o scratch buffer, o qual nos dará a chance de ver vídeo e canvas juntos em ação.



O scratch buffer; isso
parece interessante...

- 4 Vamos implementar uma função para cada efeito; faroeste, filme noir e ficção científica.
- 5 Finalmente, vamos juntar tudo para um teste!



Implemente o
scratch buffer
usando canvas
(aeride se quiser)!

```
function noir(pos, r, g, b, data) {  
    ...  
}
```



Vamos mostrar os
pixels alterados num,
adivinhe, canvas.

PODER DO CÉREBRO

Agora você sabe que vamos implementar uma função, que manipulará cada efeito. Vamos usar filme noir, por exemplo. Como você vai pegar um pixel colorido do vídeo e torná-lo preto-e-branco? Dica: cada pixel possui três componentes: vermelho, verde e azul. Se pudéssemos pôr nossas mãos naquelas peças, o que poderíamos fazer?

¹ N.E.: FX é uma sigla para efeitos especiais.

Hora de pôr aqueles botões de efeitos para funcionar



Tudo bem, primeiro, a parte fácil: vamos colocar aqueles botões de efeitos para funcionar, conectando-os. Vamos começar criando uma variável global

chamada `effectFunction`. Esta variável vai armazenar uma função, que pode pegar dados do vídeo e aplicar um filtro nele. Ou seja, dependendo de qual efeito escolhermos, a variável `effectFunction` vai armazenar uma função que sabe como processar os dados do vídeo e torná-lo preto-e-branco, ou sépia, ou invertido (para ficção científica). Então, acrescente esta variável global no topo de seu arquivo:

```
var effectFunction = null;
```

Agora vamos definir esta variável, para sempre que o botão de efeitos for clicado. Por ora, usaremos os nomes da função como `western`, `noir` e `scifi`, e escreveremos essas funções num instante.

Aqui está nosso handler `setEffect` novamente. Lembre-se de que ele é chamado toda vez que o usuário clica num botão de efeito.

```
function setEffect(e) {
    var id = e.target.getAttribute("id");

    if (id == "normal") {
        pushUnpushButtons("normal", ["western", "noir", "scifi"]);
        effectFunction = null;
    } else if (id == "western") {
        pushUnpushButtons("western", ["normal", "noir", "scifi"]);
        effectFunction = western;
    } else if (id == "noir") {
        pushUnpushButtons("noir", ["normal", "western", "scifi"]);
        effectFunction = noir;
    } else if (id == "scifi") {
        pushUnpushButtons("scifi", ["normal", "western", "noir"]);
        effectFunction = scifi;
    }
}
```

Para cada aperto de botão, definimos a variável `effectFunction` à função determinada (todas as quais ainda precisamos escrever).

Se o efeito é sem efeito, ou normal, apenas usamos `null` como valor.

Do contrário, definimos `effectFunction` a uma função nomeada apropriadamente, que fará o serviço de aplicar o efeito.

Ainda precisamos escrever essas funções de efeito. Portanto, vejamos como processamos vídeo para poder aplicar efeitos nele!

Ok, com isso fora do nosso caminho, vamos aprender sobre o “scratch buffer” e, então, voltar e ver como essas funções se encaixarão e também saber como escrevê-las!

Como funciona o processamento de vídeo

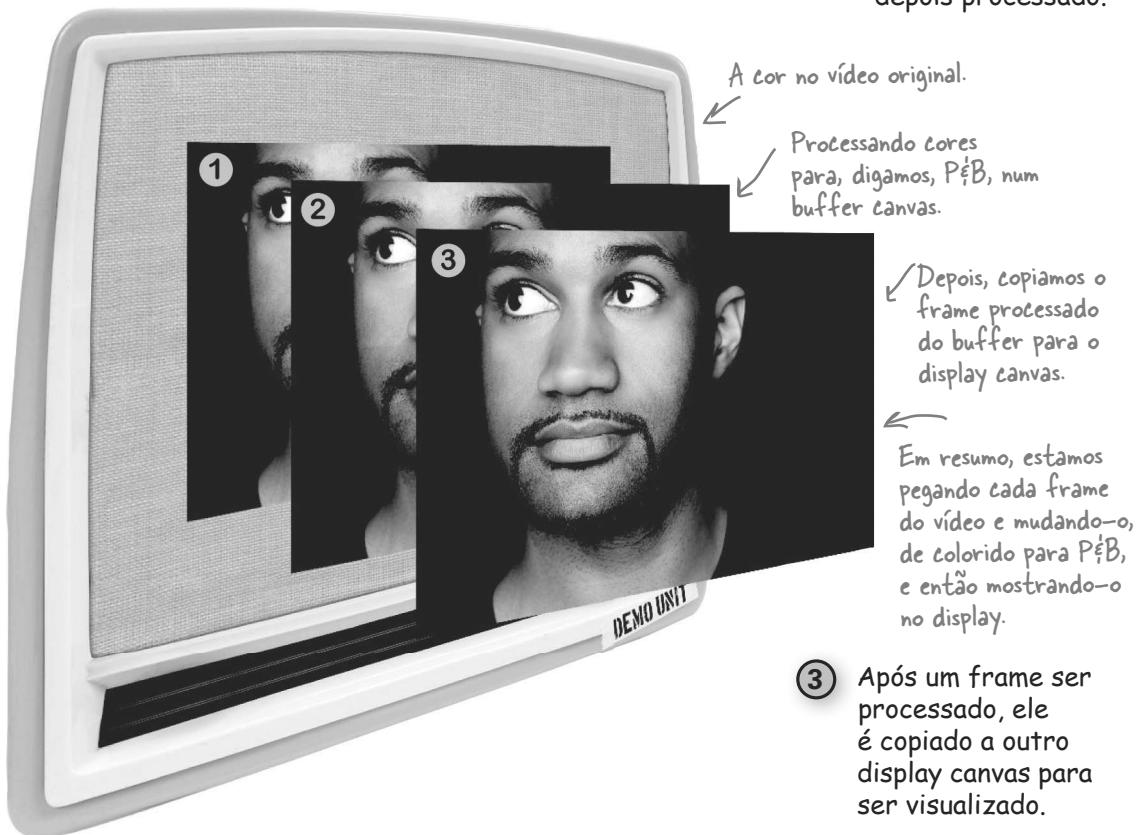
O que fizemos até agora foi fornecer a nós mesmos uma maneira de designar uma função à variável global `effectsFunction` como resultado do clique sobre os botões de efeitos na interface. Por ora, apenas aceite aquele conhecimento e guarde-o no cérebro, pois temos de trabalhar para saber como, realmente, vamos pegar um vídeo e processá-lo em tempo real para adicionar um efeito. Para isso, precisamos pôr nossas mãos nos pixels do vídeo, alterá-los para chegar ao nosso tão ansiado efeito e, então, de alguma maneira, colocá-los de volta na tela.

No entanto, a API do vídeo oferece alguma maneira especial de processar vídeo antes que ele seja mostrado? Não, mas oferece, sim, uma maneira de pegar os pixels. Então só precisaremos de uma forma de processar e mostrá-los. Espere, pixels? Display? Lembra-se do Capítulo 7? O `canvas`! Ah, é verdade, nós mencionamos algo sobre uma “relação especial” que o elemento vídeo e o `canvas` têm. Portanto, vamos conhecer uma das formas com que os elementos `canvas` e `video` podem trabalhar juntos?

Os detalhes do furo finalmente revelados!

- ① O player de vídeo decodifica e reproduz o vídeo por trás das cenas.

- ② O vídeo é copiado frame a frame dentro de um buffer `canvas` (oculto) e depois processado.





Nos
Bastidores

Como processar o vídeo usando um scratch buffer

Agora, talvez você pergunte por que estamos usando dois canvas para processar e mostrar o vídeo. Por que não encontramos, simplesmente, uma maneira de processar o vídeo, como decodificado?

O método que estamos utilizando aqui é uma técnica aprovada para minimizar as falhas visuais durante o processamento intenso de vídeo e imagem: é conhecido como usar um “scratch buffer”. Ao processar um frame de vídeo num buffer e então copiá-lo automaticamente no display canvas, minimizamos problemas visuais.

Vamos investigar como nossa implementação do scratch buffer vai funcionar.

- ① O navegador decodifica o vídeo em uma série de frames. Cada frame é um retângulo de pixels com um snapshot do vídeo em um dado momento.



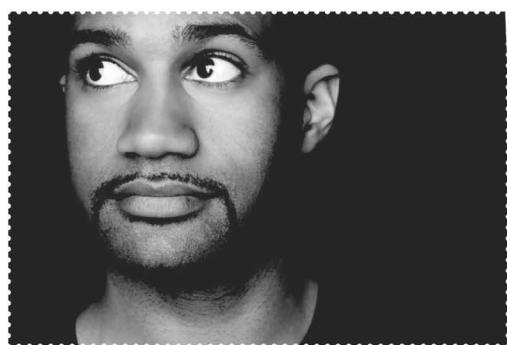
Um frame do vídeo.



- ② Como cada frame é decodificado, os copiamos dentro do canvas que está agindo como um scratch buffer.



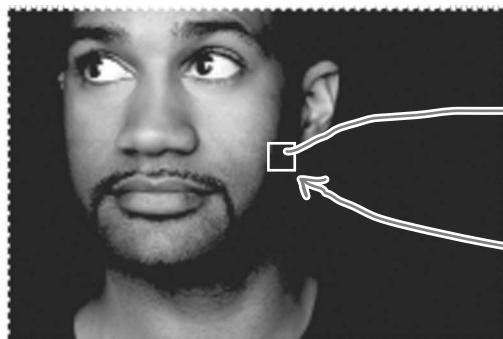
Podemos copiar o frame inteiro dentro do canvas.



↑ Este é o scratch buffer.

usando um scratch buffer

- ③ Iteramos pelo scratch buffer, pixel por pixel, passando cada pixel pela nossa função effects para processamento.



Após obter os dados do pixel do canvas, o acessamos, um pixel por vez, e o processamos, manipulando os valores RGB de cada pixel.

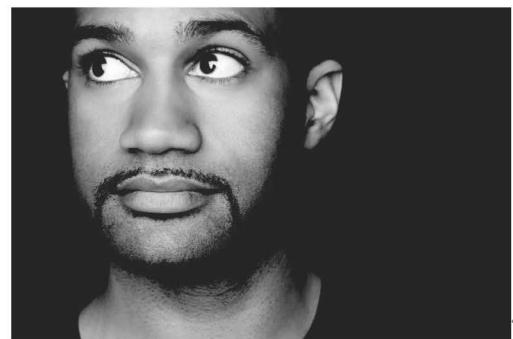
effectFunction()

1 pixel

- ④ Depois que todos os pixels no scratch buffer são processados, os copiamos a partir do scratch buffer canvas para o display canvas.



Uma vez que os dados no scratch buffer forem processados...



... pegamos a imagem do scratch buffer canvas e copiamos a coisa toda para o display canvas.

E, é claro, este é o canvas que você, de fato, vê!

- ⑤ Então, repetimos o processo em cada frame, sendo decodificado aos poucos pelo objeto vídeo.

Implementando um scratch buffer com canvas

Como você já sabe, para implementar um scratch buffer no canvas, precisamos de dois canvas: um para fazer nossos cálculos e outro para mostrar nossos resultados. Para criar aqueles canvas, começaremos do início, desde nosso arquivo HTML `videobooth.html`. Abra este arquivo e encontre a `<div>` com a id “`videoDiv`” e adicione dois elementos canvas abaixo do `<video>`:

```
<div id="videoDiv">
    <video id="video" width="720" height="480"></video>
    <canvas id="buffer" width="720" height="480"></canvas>
    <canvas id="display" width="720" height="480"></canvas>
</div>
```

Estamos adicionando dois elementos canvas, um para o buffer e outro para o display.

Perceba que eles têm exatamente o mesmo tamanho, como o elemento video.

Como posicionar o vídeo e o canvas

Agora, você deve estar imaginando como posicionar esses elementos; vamos posicionar um bem no topo do outro. Então, no fim, estará o elemento vídeo, no topo daquele, estará o buffer e, no topo do outro, o display do elemento canvas. Estamos usando CSS para isso e, embora não falemos muito de CSS neste livro, se abrir `videobooth.css`, você verá o posicionamento para os três elementos:

```
div#videoDiv {
    position: relative;
    width: 720px;
    height: 480px;
    top: 180px;
    left: 190px;
}
video {
    background-color: black;
}
div#videoDiv canvas {
    position: absolute;
    top: 0px;
    left: 0px;
}
```

A `<div>` `videoDiv` é posicionada em relação ao elemento que está lá (o console `<div>`), a 180px do topo e 190px da esquerda, o que a deixa no centro do console. Definimos a largura e a altura iguais à largura e altura do `<video>` e dos dois elementos `<canvas>`.

O `<video>` é o primeiro elemento na `<div>` `videoDiv`, portanto, será automaticamente posicionado no topo, no canto superior à esquerda, na `<div>`. Definimos o `background` para preto, pois se ele for esticado ou comprimido o espaço será preto.

Os dois elementos `<canvas>` na `<div>` `videoDiv` são posicionados absolutamente em relação à `videoDiv` (seu ascendente), portanto, ao posicionar os elementos `<canvas>` a 0px do topo e 0px da esquerda, estarão exatamente na mesma posição que o `<video>` e a `videoDiv`.

Escrever o código para processar o vídeo

Temos um elemento de vídeo, um buffer que é um canvas e um canvas que mostrará os frames finais do vídeo. Também temos todos eles empilhados; assim, vemos somente o topo do display do canvas, que conterá o vídeo com o efeito aplicado. Para processá-lo, vamos usar o evento play do elemento do vídeo, o qual é chamado assim que o vídeo começa a ser reproduzido. Adicione isto ao fim do handler onload:

```
video.addEventListener("play", processFrame, false);
```

Quando o vídeo
começar a rodar, ele
chamará a função
processFrame.

A função processFrame é onde processaremos os pixels do vídeo e inseriremos no canvas para o display. Começaremos nos certificando de que temos acesso a todos os nossos objetos DOM:

```
function processFrame() {  
    var video = document.getElementById("video");  
    if (video.paused || video.ended) {  
        return;  
    }  
    var bufferCanvas = document.getElementById("buffer");  
    var displayCanvas = document.getElementById("display");  
    var buffer = bufferCanvas.getContext("2d");  
    var display = displayCanvas.getContext("2d");  
}
```

Primeiro, pegue o objeto vídeo...
... e verifique se o vídeo ainda
está rodando. Se não estiver,
não haverá qualquer trabalho a
ser feito a não ser retornar.
Pegue uma
referência tanto
aos elementos
canvas quanto aos
seus contextos, pois
precisaremos deles.

Como criar o buffer

Para criar o buffer, precisamos pegar o vídeo frame atual e copiá-lo ao buffer canvas. Uma vez que o temos no canvas, podemos processar dados no frame. Então, para criar aquele buffer, fazemos isso (acrescentar isso no fim do processFrame):

```
buffer.drawImage(video, 0, 0, bufferCanvas.width, bufferCanvas.height);  
var frame = buffer.getImageData(0, 0, bufferCanvas.width, bufferCanvas.height);
```

Ele pega uma imagem e a desenha no canvas a uma
posição x, y para uma dada largura e altura.

Você se lembra do
contexto do método
drawImage do Capítulo 7?

Desta vez, estamos pegando uma imagem do vídeo.
Ao especificá-lo como a fonte, drawImage pega um
frame do vídeo como dados da imagem.

Então, pegamos os dados da imagem do
contexto canvas e os armazenamos numa
variável. Assim, poderemos processá-los.

Aqui estamos apenas dizendo
que queremos todos os dados
da imagem no canvas.

Como processar o buffer

Temos nossas mãos no frame dos dados do vídeo, então vamos processar um pouco! Para fazer isso, vamos passar por cada pixel nos dados do frame e retirar os valores de cor RGB que ficam armazenados neles. Na verdade, cada pixel possui 4 valores RGB e Alpha (a opacidade), mas não usaremos o Alpha. Uma vez que tivermos os valores RGB, chamaremos a `effectFunction` (lembre-se de que é a função que definimos lá na página 392 e lhe pedimos para guardar na parte de trás de seu cérebro!) com a informação RGB e o frame.

Adicione este código ao fim de sua função `processFrame`:

```
buffer.drawImage(video, 0, 0, bufferCanvas.width, displayCanvas.height);
var frame = buffer.getImageData(0, 0, bufferCanvas.width, displayCanvas.height);

var length = frame.data.length / 4; ← Primeiro, descobriremos o comprimento dos dados do frame. Perceba que os dados estão numa propriedade do frame, frame.data, e length (comprimento) é uma propriedade de frame.data. O comprimento é, de fato, quatro vezes maior que o tamanho do canvas, porque cada pixel possui quatro valores: RGBA.

for (var i = 0; i < length; i++) { ← Agora, passaremos por sobre os dados e obteremos os valores RGB para cada pixel. Cada pixel ocupa quatro espaços no array; pegaremos o r da primeira posição, o g da segunda e o b da terceira.

    var r = frame.data[i * 4 + 0];
    var g = frame.data[i * 4 + 1];
    var b = frame.data[i * 4 + 2];
    if (effectFunction) {
        effectFunction(i, r, g, b, frame.data); ← Em seguida, chamaremos a effectFunction (se não for null, o que poderá ser se o botão "Normal" estiver pressionado) passando a posição do pixel, os valores RGB e o array frame.data. A função effect atualizará o array frame.data com novos valores de pixel, processados de acordo com a função filtro designada para effectFunction.
    }
}
display.putImageData(frame, 0, 0);
```

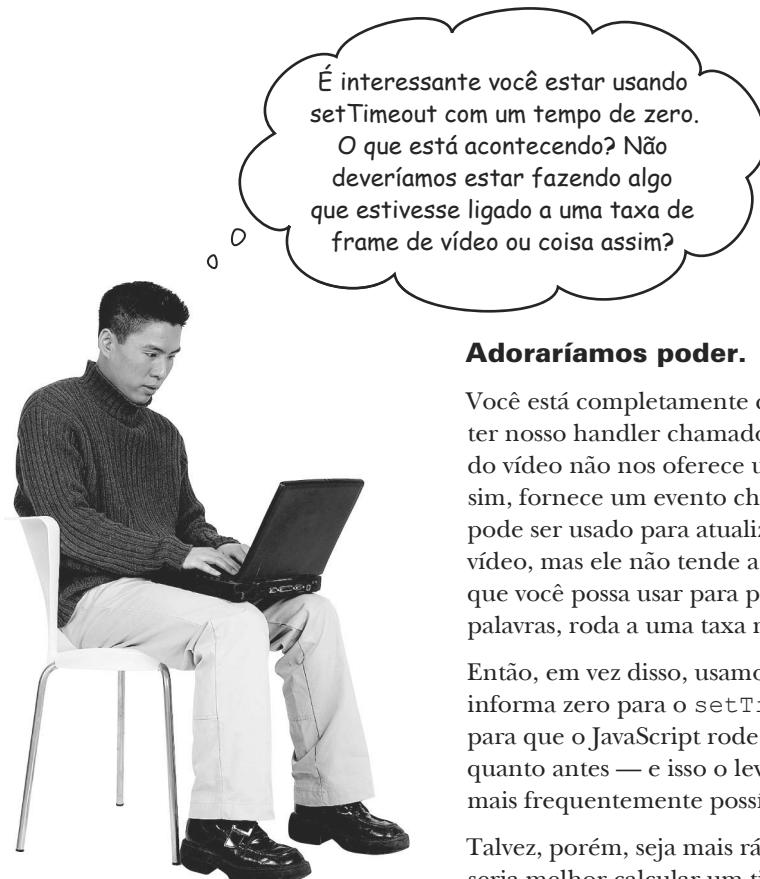
Neste ponto, os dados do frame foram processados. Então, usaremos o método `putImageData` do contexto para pôr os dados dentro do display canvas. Este método aceita os dados do frame e os escreve dentro do canvas na posição x, y especificada.

Processamos um frame, e agora?

Sim, esse foi apenas o primeiro frame a ser processado e queremos continuar processando todos, à medida que o vídeo continuar a rodar. Podemos usar `setTimeout` e fornecer-lhe um valor de zero milissegundos para pedir ao JavaScript para rodar `processFrame` novamente, assim que for possível. O JavaScript não vai, de fato, rodar a função em zero milissegundos, mas nos dará o slot de tempo mais recente que puder conseguir. Para isso, só acrescente isto no fim de sua função `processFrame`:

```
setTimeout(processFrame, 0); ← setTimeOut é igual a setInterval, só que roda apenas uma vez após um determinado tempo em milissegundos.
```

Diz ao JavaScript para rodar `processFrame` novamente, assim que possível!



É interessante você estar usando `setTimeout` com um tempo de zero. O que está acontecendo? Não deveríamos estar fazendo algo que estivesse ligado a uma taxa de frame de vídeo ou coisa assim?

Adoraríamos poder.

Você está completamente certo: o que adoramos fazer é ter nosso handler chamado para cada frame, mas a API do vídeo não nos oferece uma maneira de fazer isso. Ela, sim, fornece um evento chamado `timeupdate`, que pode ser usado para atualizar um time display de seu vídeo, mas ele não tende a atualizar a uma granularidade que você possa usar para processar frames (em outras palavras, roda a uma taxa mais lenta que o vídeo).

Então, em vez disso, usamos `setTimeout`. Quando você informa zero para o `setTimeout`, você está pedindo para que o JavaScript rode seu handler de timeout o quanto antes — e isso o leva ao seu handler, rodando-o mais frequentemente possível.

Talvez, porém, seja mais rápido que o frame rate... Não seria melhor calcular um timeout próximo ao que é necessário para o frame rate? Bem, você poderia, mas é improvável que o handler vá mesmo rodar simultaneamente com os frames de seu vídeo; então, zero é um bom chute. Claro, se estiver procurando por características de melhoria de performance de seu aplicativo, você pode fazer sempre um pouco de profiling e descobrir quais são os valores otimizados. Contudo, até termos uma API mais específica, ainda estaremos limitados.

Agora precisamos escrever alguns efeitos

Finalmente, temos tudo o que precisamos para escrever os efeitos do vídeo: estamos pegando cada frame à medida que eles vão chegando, acessando os dados de frame pixel a pixel e enviando os pixels à nossa função de filtro de efeitos. Vamos ver o filtro *Filme Noir* (que, em nossa versão, é só um nome bonito para preto e branco):

```

A função filtro passa
a posição do pixel...
... os valores vermelho,
verde e azul do pixel...
... e uma referência
ao array dos dados
do frame no canvas.

é um operador
bitwise que
transforma os
bits num valor
numérico para
modificar o
número. Explore
mais isso num livro
de referência
JavaScript.

function noir(pos, r, g, b, data) {
    var brightness = (3*r + 4*g + b) >>> 3;
    if (brightness < 0) brightness = 0;
    data[pos * 4 + 0] = brightness;
    data[pos * 4 + 1] = brightness;
    data[pos * 4 + 2] = brightness;
}

Lembre-se de
que esta função é
chamada uma vez por
pixel no vídeo frame!
... e uma referência
ao array dos dados
do frame no canvas.

Assim, a primeira coisa
que faremos é computar
um valor de brilho para
este pixel baseado
em todos os seus
componentes (r, b e g).

Depois, designaremos
cada componente na
imagem do canvas
para aquele brilho.

Isto tem o efeito de
definir o pixel para um
valor de escala de cinza
que corresponde ao
brilho em geral do pixel.

```

Um test drive do filme noir

Adicione esta função no `videobooth.js` e, em seguida, recarregue sua página. Assim que o vídeo começar a reproduzir, pressione o botão *Filme Noir* e verá uma versão perturbadora dele, em preto e branco. Agora escolha *Normal* novamente. Nada mal, hein?! E tudo em JavaScript, em tempo real!

↑
Meio que incrível, quando
você pensa a respeito.



Aponte o seu lápis



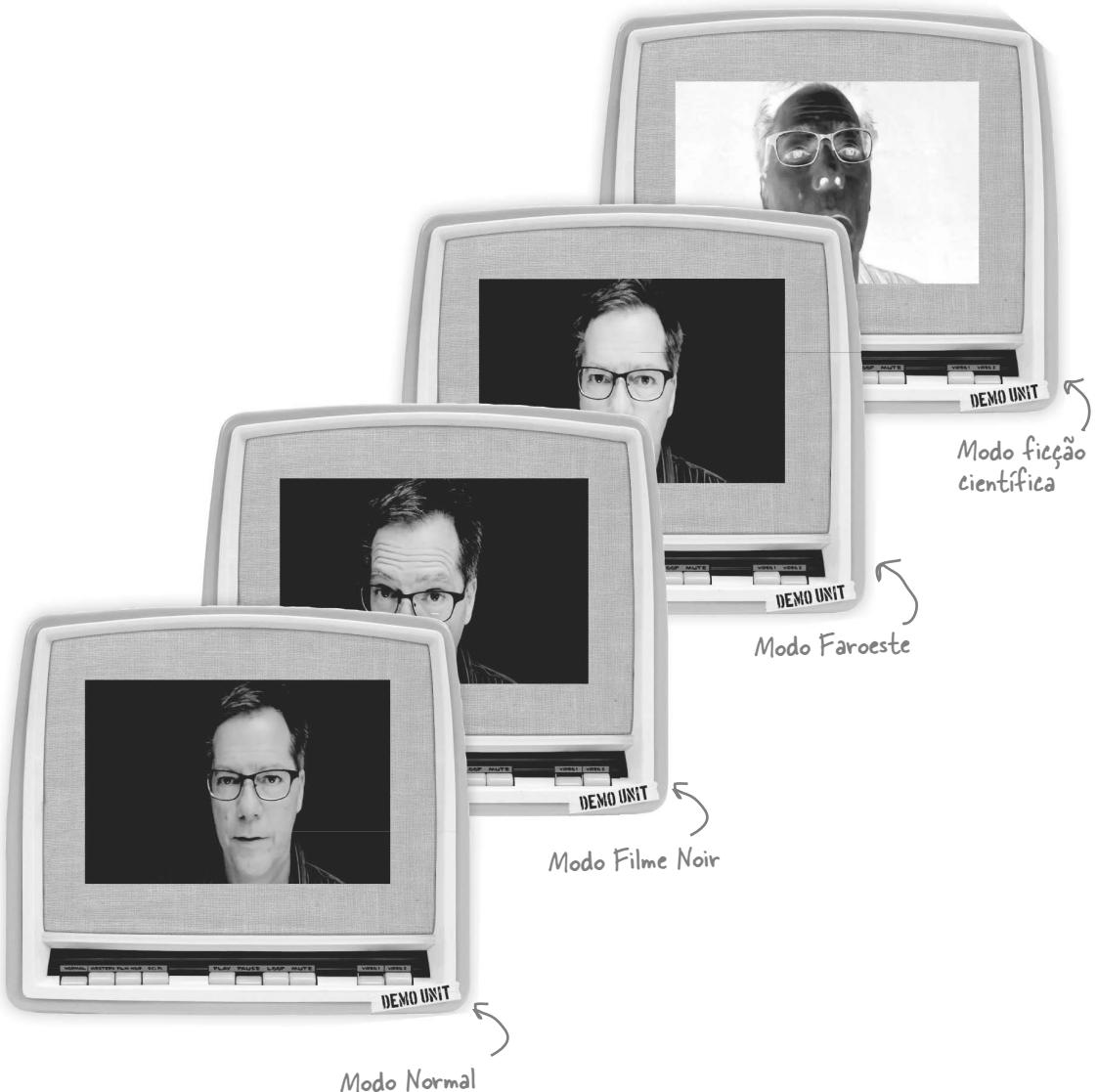
Este livro não é exatamente sobre processamento de vídeo e efeitos, mas com certeza isso é divertido. Abaixo, temos os efeitos faroeste e ficção científica. Dê uma olhada no código e faça anotações à direita sobre como ele funciona. Ah, e acrescentamos um extra — o que ele faz?

```
function western(pos, r, g, b, data) {  
    var brightness = (3*r + 4*g + b) >>> 3;  
    data[pos * 4 + 0] = brightness+40;  
    data[pos * 4 + 1] = brightness+20;  
    data[pos * 4 + 2] = brightness-20;  
}  
  
function scifi(pos, r, g, b, data) {  
    var offset = pos * 4;  
    data[offset] = Math.round(255 - r) ;  
    data[offset+1] = Math.round(255 - g) ;  
    data[offset+2] = Math.round(255 - b) ;  
}  
  
function bwcartoon(pos, r, g, b, outputData) {  
    var offset = pos * 4;  
    if( outputData[offset] < 120 ) {  
        outputData[offset] = 80;  
        outputData[++offset] = 80;  
        outputData[++offset] = 80;  
    } else {  
        outputData[offset] = 255;  
        outputData[++offset] = 255;  
        outputData[++offset] = 255;  
    }  
    outputData[++offset] = 255;  
    ++offset;  
}
```

O Grande Test Drive



É isso! Temos o código prontinho para zarpar para a **Starring You Video**. Vá em frente, cheque duas vezes para conferir se digitou todo o código certinho, salve e carregue `videobooth.html`. Depois, vá se divertir com seu novo aplicativo!



NO LABORATÓRIO

Obviamente, apenas tocamos a superfície em termos de processamento de vídeo e temos certeza de que você pode pensar em efeitos mais criativos do que aqueles que trouxemos à tona. Vá em frente, imagine alguns, implemente-os e documente-os aqui.

Inventou alguma coisa bem legal e a implementou? Conte-nos a respeito em wickedlysmart.com e vamos compartilhar com outros leitores!



Desenho P&B é apenas uma de muitas outras coisas engraçadas que você pode fazer com efeitos.





Olha, eu sei que está quase no fim do capítulo, mas continuo a querer perguntar: estamos carregando vídeo de um arquivo local. Quais modificações devem ser feitas, se meu vídeo estiver hospedado na web?

Claro, é só usar uma URL.

Você pode substituir qualquer uma das fontes que definimos localmente por uma URL. Por exemplo:

```
<video src="http://wickedlysmart.com/myvideo.mp4">
```

Tenha em mente que existe maior possibilidade de coisas ruins acontecerem, quando se trabalha na web (e falaremos sobre como lidar com isso num momento!). Além do mais, o bitrate de seus vídeos começa a ser muito mais importante, quando se desenvolve para um browser ou um dispositivo móvel pela rede. Assim como com os formatos de vídeo, se estiver trilhando este caminho, procure por especialistas e vá aprender.

Ótimo, e só mais uma pergunta:
existe alguma diferença entre
o que estamos fazendo e
streaming de vídeo?

Sim, uma grande diferença.

O termo streaming geralmente é usado como os termos xerox ou kleenex — um termo genérico para se obter vídeo da web para seu navegador. “Vídeo progressivo” e “streaming de vídeo”, porém, são, na verdade, termos técnicos. Neste livro, utilizamos vídeo progressivo, o que significa que, quando resgatamos o vídeo (tanto localmente quanto pela internet), estamos resgatando um arquivo utilizando HTTP, assim como um arquivo HTML ou uma imagem. Tentamos decodificar e reproduzi-lo, à medida que o resgatamos. Streaming de vídeo é gerado utilizando um protocolo, que é altamente relacionado com o fornecimento de vídeo de uma maneira otimizada (talvez até alterando o bitrate do vídeo com o passar do tempo, à medida que a transferência de dados se torna mais ou menos disponível).



O streaming de vídeo provavelmente fornecerá a seu usuário uma experiência melhor (é verdade!), e é, possivelmente, o mais eficiente nos quesitos conexão de seus usuários e cobranças de transferência de dados (também verdade). Acima de tudo, o streaming de vídeo torna mais fáceis coisas como proteger o conteúdo de seu vídeo, se precisar desse tipo de segurança.



Então, existe um padrão para streaming no HTML5?

Não.

Não há padrão para streaming de vídeo com o HTML5. Inclusive, o problema não é o HTML5. Não há, de fato, um padrão suportado para streaming de vídeo em parte alguma — mas existe uma porção de proprietários. Por quê? Existe um número enorme de razões, variando desde o dinheiro que pode ser feito com streaming de vídeo até o fato de que muitas pessoas de fonte aberta não querem trabalhar num protocolo que poderá ser usado por DRM ou outras tecnologias de proteção. Como naquela situação com os formatos de vídeo, estamos num mundo complexo com o streaming de vídeo.

Então, o que eu faço se eu preciso de streaming?

Existem soluções por aí.

Há diversos usos legítimos para tecnologias de streaming de vídeo e, se tiver um grande público, ou conteúdo que pensa ser necessário estar protegido, você deveria dar uma olhada nesses: HTTP Live Streaming da Apple, Smooth Streaming da Microsoft e HTTP Dynamic Streaming da Adobe. São ótimos lugares para se começar.

Há boas notícias no horizonte também: os órgãos de definição de padrão estão começando a olhar mais de perto para o streaming de vídeo baseado em HTTP. Portanto, fique de olho por acontecimentos nesta área.



Se pelo menos fosse um mundo perfeito...

Mas não é: temos todos aqueles problemas chatos de rede, dispositivos e sistemas operacionais incompatíveis e uma crescente chance de asteroides atingirem a Terra. Com relação ao último, não podemos ajudar, mas quanto ao primeiro e ao segundo, na verdade, saber que há um erro é metade do caminho, já que, pelo menos, você pode fazer algo a respeito.

O objeto vídeo possui um error event, que pode ser lançado por diversas razões, ser encontradas na propriedade `video.error`, ou, mais especificamente, na propriedade `video.error.code`. Vamos dar uma olhada nos tipos possíveis de erros que somos capazes de detectar:



Erros

`MEDIA_ERR_ABORTED=1`

Usado a qualquer hora em que o processo de obtenção de vídeo pela rede é abortado pelo navegador (possivelmente, durante uma solicitação de usuário).

`MEDIA_ERR_NETWORK=2`

Usado sempre que um resgate de vídeo na rede é interrompido por um erro de rede.

`MEDIA_ERR_DECODE=3`

Usado sempre que a decodificação de um vídeo falha. Isso pode acontecer porque a codificação usa recursos que o navegador não suporta ou porque o arquivo está corrompido.

`MEDIA_ERR_SRC_NOT_SUPPORTED=4`

Usado quando a fonte do vídeo especificada não pode ser suportada por causa de uma URL ruim ou porque o tipo de fonte não é decodificável pelo navegador.

Cada tipo de erro também tem um número associado que é o código do erro produzido pelo error event. Veremos isso num segundo...

Como usar error events

Lidar com erros é um negócio complexo e a maneira como você irá lidar com eles, dependerá muito de seu aplicativo e do que seria apropriado para ele e seus usuários. Dito isto, podemos, ao menos, iniciá-lo e direcioná-lo para o lugar certo. Vamos pegar a TV Webville e dar a ela habilidade de saber que encontrou um erro — e, se fornecer um, dê ao público uma mensagem de POR FAVOR, AGUARDE.

Queremos ser notificados, quando houver uma mensagem de erro. Então, precisamos adicionar um listener para o error event. Veja como fazemos isso (acrescente este handler onload no webville.js):

```
video.addEventListener("error", errorHandler, false);
```

Agora, precisamos escrever a função errorHandler, que verificará se existe um erro e, em caso positivo, colocará nossa imagem “por favor, aguarde” no display, tornando-a a imagem do pôster:

```
function errorHandler() {  
  var video = document.getElementById("video");  
  if (video.error) {  
    video.poster = "images/technicaldifficulties.jpg";  
    alert(video.error.code);  
  }  
}
```

Quando um erro ocorre, a função errorHandler é chamada.

Se o handler for chamado, vamos nos certificar de que há um erro, checando o video.error e, então, colocamos um pôster no display.

Opcionalmente, adicione esta linha para que seja capaz de ver o código do erro (veja a página anterior, para a íntegra da propriedade armazenada no código).

Test Crash!



Há muitas maneiras de se fazer a reprodução do vídeo falhar e, para testar esse código, você a fará falhar. Eis algumas sugestões:

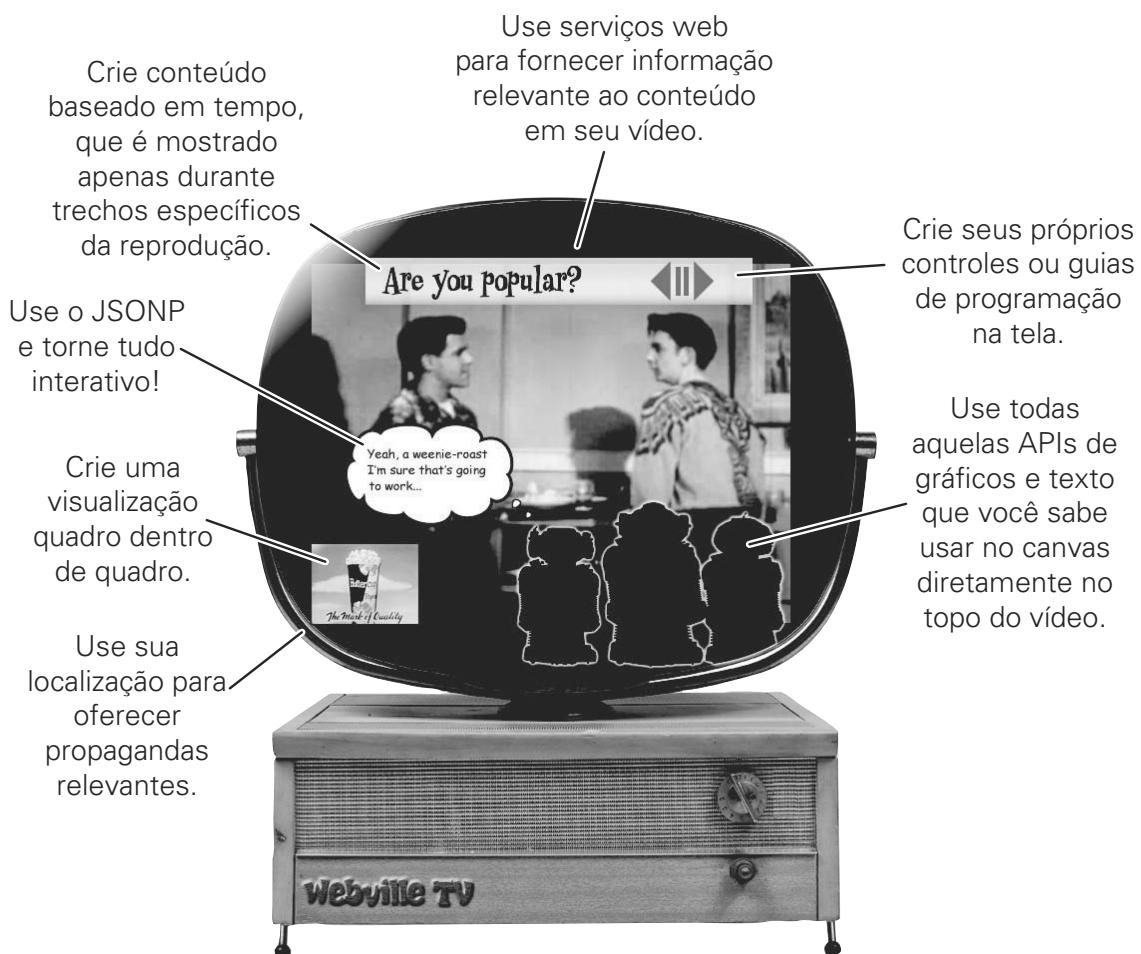
- Desconecte sua rede em diferentes momentos durante a reprodução.
- Dê uma URL ruim para o player.
- Dê ao player uma URL que nem seja um vídeo.
- Use software para reduzir sua transferência de dados (está por aí, é só procurar).

Então, digite este código e comece a testar. Lembre-se de que você pode mapear a íntegra na caixa de diálogo do alerta para um código verdadeiro, olhando os códigos da página 407.



Aonde você pode ir a partir de agora?

Agora é que a coisa fica boa, pois pense em tudo o que você já sabe fazer com marcação HTML, com o elemento vídeo e, é claro, o canvas... sem falar nos serviços web, geolocalização... Nossa! Claro, fizemos alguns processamentos legais de vídeo com o canvas, mas você pode aplicar tudo o que sabe fazer com o canvas no vídeo. Aí vão só algumas ideias que tivemos; mas, por favor, acrescente algumas suas. Dê um tapinha nas suas próprias costas por nós, pois você merece!





PONTOS DE BALA

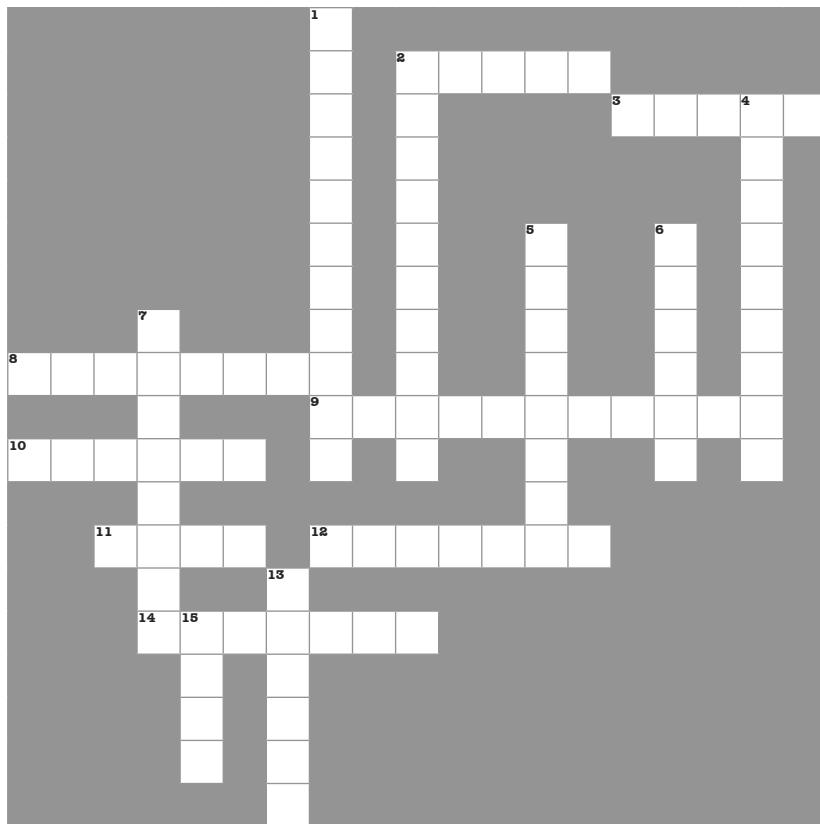
- Você pode reproduzir vídeo usando o elemento <video> com alguns simples atributos.
- O atributo autoplay começa a reprodução no carregamento da página, mas use apenas quando for apropriado.
- O atributo controls faz com que o navegador exponha um conjunto de controles de reprodução.
- A aparência e sensação dos controles diferem entre os navegadores.
- Você pode fornecer sua própria imagem de pôster com o atributo poster.
- O atributo src possui uma URL para que o vídeo seja reproduzido.
- Há muitos “padrões” para formatos de vídeo e áudio.
- Três formatos são de uso comum, WebM, MP4/H.264 e Ogg/Theora.
- Conheça seu público para saber quais os formatos que você precisa suprir.
- Use a tag <source> para especificar formatos alternativos de vídeo.
- Use tipos plenamente especificados em sua tag <source> para poupar tempo e trabalho de seu navegador.
- Você pode continuar a dar suporte a outros frameworks de vídeo, como o Flash, adicionando uma tag alternativa <object> no elemento vídeo.
- O objeto vídeo fornece um rico conjunto de propriedades, métodos e eventos.
- O video suporta os métodos e propriedades play, pause, load, loop e mute para controlar diretamente a reprodução do vídeo.
- O evento ended pode ser usado para saber quando a reprodução do vídeo foi finalizada (por exemplo, para implementar uma lista de reprodução).
- Você pode pedir programaticamente ao objeto video se ele pode reproduzir um formato com canPlayType.
- O método canPlayType retorna string vazia (sem suporte para o formato), maybe (que talvez seja capaz de reproduzir o formato) ou probably (que ele pensa confiar na reprodução do formato).
- O canvas pode ser usado como uma superfície do display para vídeos, para implementar controles personalizados ou outros efeitos com vídeos.
- Você pode usar um scratch buffer para processar o vídeo antes de copiá-lo no display.
- Você pode usar um handler setTimeout para processar os frames do vídeo; ainda que não esteja diretamente ligado a cada frame do vídeo, é o melhor método que temos até agora.
- Você pode usar uma URL como fonte de vídeo para reproduzir vídeos baseados na rede.
- Alguns navegadores fazem cumprir a mesma política de origem de vídeos, para que você precise servir o vídeo a partir da mesma origem que sua página fonte.
- Erros são sempre possíveis, senão prováveis, com video, principalmente quando uma rede está envolvida.
- O error event pode ser usado para notificar um handler, quando erros de resgate do vídeo, decodificação ou reprodução ocorrem.
- O elemento video baseia-se em seu download progressivo. Atualmente, não há padrão HTML5 para streaming, embora alguns órgãos de padronização têm procurado por soluções de streaming para HTTP.
- Não há atualmente uma maneira padrão de proteger vídeos fornecidos pelo elemento video.



Cruzada HTML5

não é a tv de seu pai

Antes de se sentar e assistir a um pouco de TV Webville, faça uma rápida palavra cruzada para gravar tudo em sua mente. Segue abaixo, seu enigma do Capítulo 8.



HORIZONTAIS

2. Quando o programa acaba, este evento é lançado.
3. O que processamos em toda chamada setTimeout.
8. Começa um vídeo assim que pode.
9. Tipos de fornecimento que o elemento video usa para o vídeo.
10. O que você deveria fazer se um asteroide estivesse para atingir a Terra.
11. Propriedade para reproduzir seu vídeo repetidas vezes.
12. Aparenta e dá a sensação que os controles do browser ____.
14. Tipo de buffer para que usamos o canvas.

VERTICIAIS

1. Posso reproduzir este tipo, você pode?
2. Vimos filmes _____ da década de 1950.
4. Para fornecer várias opções de vídeo, use _____ elementos de fonte.
5. Clint Eastwood gostaria desse estilo de efeito.
6. O codec de áudio de fonte aberta.
7. Use _____ se quiser uma forma embutida de controlar vídeos.
13. Usado para mostrar vídeo processado.
15. O CEO da Starbuzz cospe seu(ua) _____.



Aponte o seu lápis Solução

Este livro não é exatamente sobre processamento de vídeo e efeitos, mas com certeza isso é divertido. Abaixo, temos os efeitos faroeste e ficção científica. Dê uma olhada no código e faça anotações à direita sobre como ele funciona. Ah, e acrescentamos um extra — o que ele faz? Eis nossa solução.

```
function western(pos, r, g, b, data) {
    var brightness = (3*r + 4*g + b) >>> 3;
    data[pos * 4 + 0] = brightness+40;
    data[pos * 4 + 1] = brightness+20;
    data[pos * 4 + 2] = brightness-20;
}
```

```
function scifi(pos, r, g, b, data) {
    var offset = pos * 4;
    data[offset] = Math.round(255 - r);
    data[offset+1] = Math.round(255 - g);
    data[offset+2] = Math.round(255 - b);
}
```

```
function bwcartoon(pos, r, g, b, outputData) {
    var offset = pos * 4;
    if( outputData[offset] < 120 ) {
        outputData[offset] = 80;
        outputData[+offset] = 80;
        outputData[+offset] = 80;
    } else {
        outputData[offset] = 255;
        outputData[+offset] = 255;
        outputData[+offset] = 255;
    }
    outputData[+offset] = 255;
    +offset;
}
```

O filtro Faroeste enfatiza os componentes vermelho e verde do pixel, enquanto que diminui a incidência do componente azul, para dar ao vídeo uma tintura amarronzada.

O filtro ficção científica inverte as quantidades de componentes RGB de cada pixel. Assim, se um pixel tiver muito vermelho, agora ele terá pouco. Se tiver pouco verde, terá muito.

O filtro bwcartoon (desenho em P&B) transforma todo pixel com um componente vermelho menor que 120 (de 255) em preto e todos os outros pixels em branco, dando ao vídeo uma estranha aparência de desenho animado em P&B.

RECONHECIMENTO DO VÍDEO **CONFIDENCIAL** SOLUÇÃO

Dispositivos iOS e Android (entre outros)

| video | Browser | Safari | Chrome | Firefox | Mobile Webkit | Opera | IE9+ | IE8 | IE7 or ✓ |
|------------|---------|--------|--------|---------|---------------|-------|------|-----|----------|
| H.264 | ✓ | some | | | ✓ | | ✓ | | |
| WebM | | ✓ | ✓ | | | ✓ | | | |
| Ogg Theora | | ✓ | ✓ | | | ✓ | | | |

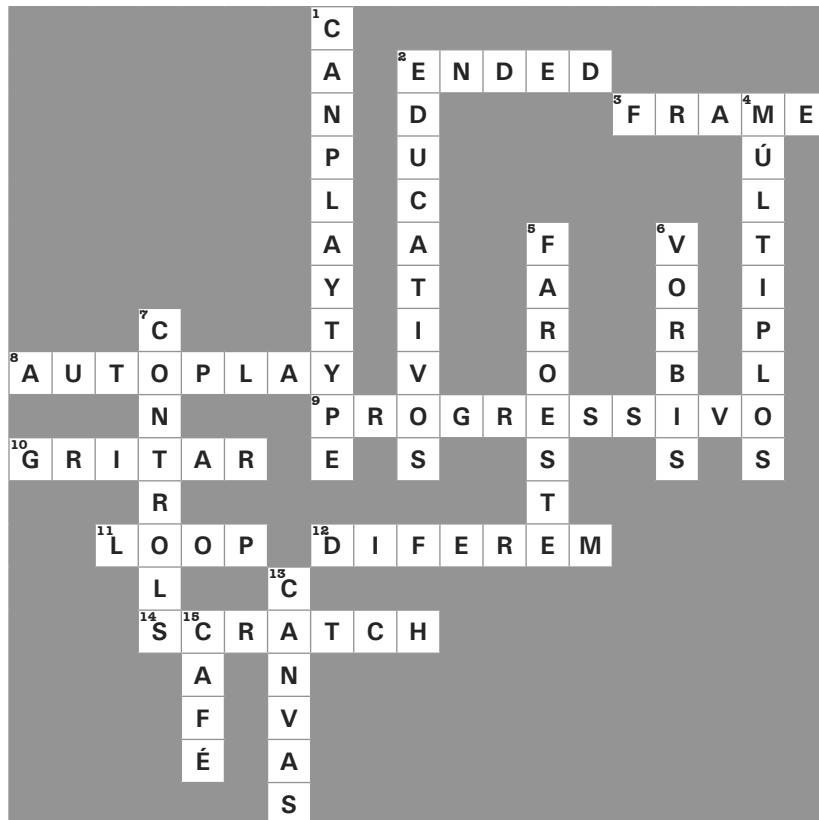


Veja bem!

*Isso vai mudar
rápido! Então
verifique as
novidades de
suporte na web.*



Cruzada HTML5 – Solução



9 armazenando coisas localmente

Web Storage

Estou farta deste armário pequeno
e de ter de vestir sempre esse
mesmo tailleur. Com o HTML5, tenho
armazenamento local suficiente para
vestir uma nova roupa a cada dia!



Cansado de empurrar os dados de seu cliente para dentro de um pequeno armário cookie? Isso foi divertido nos anos 1990, mas temos necessidades muito maiores hoje em dia com os aplicativos web. E se disséssemos que poderíamos conseguir cinco megabytes para você no navegador do usuário? Provavelmente, olharia para nós como se estivéssemos tentando vender uma ponte para você no Brooklyn. Bem, não precisa ser céitico — a API Web Storage HTML5 faz exatamente isso! Neste capítulo, vamos mostrar tudo que você precisa para armazenar qualquer objeto localmente, no dispositivo de seu usuário, e se aproveitar disso na experiência web.

Como funciona o armazenamento do navegador (1995 — 2010)

Construindo um carrinho de compras? Precisa armazenar algumas preferências de usuário para seu site? Ou apenas precisa estocar um pouco de dados que devem ser associados com cada usuário? É aí que o armazenamento do navegador aparece. O armazenamento do navegador oferece uma maneira de armazenar persistentemente dados que poderemos usar na construção de uma experiência web.

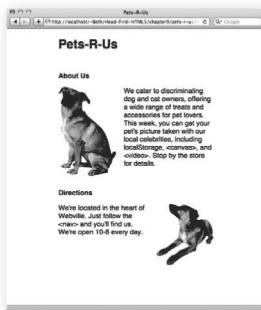
Até agora só havia um jeito disponível — o cookie do navegador — para armazenar informação no navegador. Vejamos como os cookies funcionam:

Nos
Bastidores



- 1 Quando seu navegador resgata uma página, digamos de "pets-R-us.com", o servidor pode enviar um cookie junto com sua resposta. Os cookies contêm um ou mais pares de valor e chave:

Enquanto estou servindo uma página para você, também vou lhe dar alguns pares de valor/chave para armazenar para mim. Da próxima vez que entrar em contato comigo, mande-os para mim, junto com sua solicitação.



Navegador

O navegador salva o cookie localmente e o enviará de volta ao servidor da próxima vez que fizer uma solicitação.

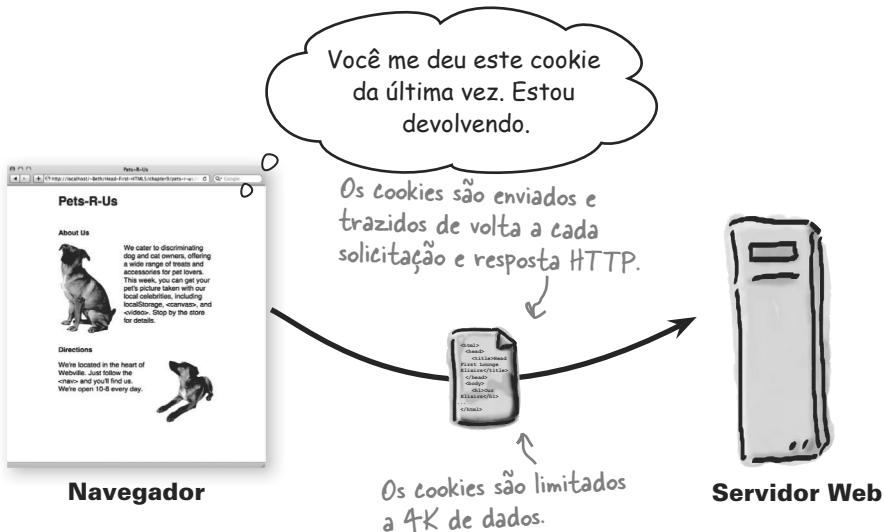


Servidor Web

Cookie: pet=dog; age=5; color=black

Aqui estão alguns pares de valor e chave. Temos uma chave de "pet" com um valor de "dog", e uma chave de "age" com um "5", e assim por diante...

- ② Da próxima vez que o navegador fizer uma solicitação para "pets-R-us.com", ele mandará quaisquer cookies que já tenham sido mandados previamente:



- ③ O servidor pode, então, usar o cookie para personalizar a experiência, neste caso, promovendo os itens relevantes ao usuário, mas há muitas outras maneiras dos cookies serem usados também.



PODER DO CÉREBRO

Os cookies convivem conosco há muito tempo, mas talvez você seja capaz de pensar em algumas maneiras de melhorá-los.

Marque todos os itens abaixo que você acha que podem tornar os cookies problemáticos:

- Há apenas 4k para se trabalhar e meu aplicativo precisa de maior armazenamento que isso.
- Enviar e trazer de volta os cookies toda vez parece realmente ineficiente, principalmente se eu estiver usando um dispositivo móvel com baixa taxa de transferência de dados.
- Eles parecem uma boa maneira de transmitir vírus e outros malwares ao meu navegador.
- Ouvi falar que a maneira com que os pares chave/valor são feitos como parte da solicitação HTTP é muito chata de se lidar em um código.
- Não estamos potencialmente enviando e recebendo dados pessoais toda vez que fazemos uma solicitação?
- Eles não parecem combinar com todos os avanços no lado cliente que vínhamos fazendo. Eles parecem presumir tudo o que está acontecendo no servidor.

Pelo que consta, e apesar de notícias contrárias, os cookies são bem seguros e não um refúgio de desenvolvedores de vírus.



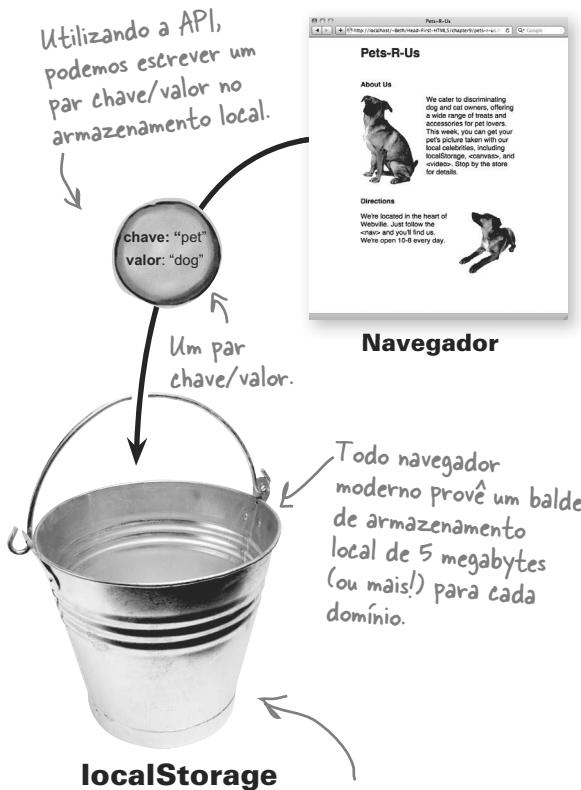
Como o Web Storage HTML5 funciona

Nos
Bastidores

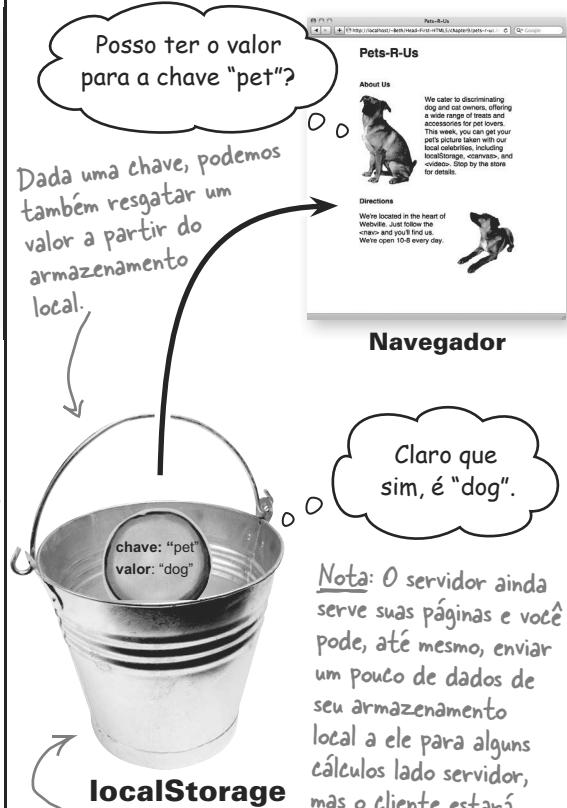


O HTML5 nos dá uma boa e simples API JavaScript no navegador para armazenar pares chave/valor que são persistentes. Você não está limitado a quatro miseráveis kilobytes de armazenamento também; todos os navegadores hoje oferecerão, com prazer, de cinco a dez megabytes de armazenamento em todos os navegadores do usuário. O armazenamento local HTML5 também foi criado com aplicativos web (e aplicativos de dispositivos móveis!) em mente — armazenamento local significa que seu aplicativo pode armazenar dados no navegador para reduzir a comunicação necessária com o servidor. Vamos dar uma olhada em como funciona (e, em seguida, vamos pular para a API):

- 1** Uma página pode armazenar um ou mais pares chave/valor no armazenamento local do navegador.



- 2** Depois, use uma chave para resgatar seu valor correspondente.



Nota: O servidor ainda serve suas páginas e você pode, até mesmo, enviar um pouco de dados de seu armazenamento local a ele para alguns cálculos lado servidor, mas o cliente estará lidando com os detalhes do armazenamento local, não com o servidor (como é comum com cookies).

Como os cookies, sua página pode armazenar e resgatar apenas itens que foram criados por páginas servidas a partir do mesmo domínio. Falaremos mais sobre isso num instante.

Nota para si mesmo...

Precisando de um sistema para conseguir fazer coisas? É difícil melhorar o velho sistema de notas de Post-it (mais comumente conhecido como *stickies*). Você sabe como funciona: você anota algum item “para fazer”, cola em algum lugar e, assim que termina a tarefa, joga o papelzinho no lixo (ou o recicla).

Que tal se construirmos um usando HTML? Vejamos. Precisamos encontrar uma maneira de armazenar todos aqueles papeizinhos. Vamos precisar de um servidor e alguns cookies... ah, espere um segundo, voltando um pouco, podemos fazer isso com a API de Web Storage HTML5!

A API de Web Storage é simples, divertida e instantaneamente gratificante. Nós prometemos!

Buscar roupas
na lavanderia

Ferramenta de
produtividade
high-tech.



Exercício

Sem brincadeirinhas, vamos direto ao assunto e começar a usar o armazenamento local. Para isso, você deverá criar uma simples página html com todo o básico: cabeçalho, corpo e um script (ou apenas use o arquivo iniciante `notetoself.html` nos exemplos de código). Passando para digitação do código em seu elemento `<script>` (digitar ajuda a decorar):

- Não há nada mais para um Post-it do que o texto que você escreverá nele, certo? Então, vamos começar por armazenar uma “sticky” para “Buscar roupas na lavanderia”.

A API de Web Storage está disponível para você através do objeto `localStorage`. Você já o encontrará definido pelo navegador. Quando usá-lo, estará fazendo uso do sistema de armazenamento local subjacente.

Para armazenar algo, usaremos o método `setItem`.

O método `setItem` pega duas strings como argumentos que agem como o par chave/valor.

```
localStorage.setItem("sticky_0", "Pick up dry cleaning");
```

O argumento da primeira string é uma chave em que o item é armazenado. Nomeie-o como quiser, contanto que seja uma string.

Começamos de forma simples, mas antes que você perceba, teremos um aplicativo completo de Stickies funcionando a todo vapor.

Você pode apenas armazenar itens de tipo String. Não é possível armazenar números ou objetos diretamente (mas encontraremos um jeito de superar essa limitação em breve).

A segunda string é o valor que você vai querer armazenar no `localStorage`.

- ② Essa foi fácil; vamos adicionar um segundo item no armazenamento local:

```
localStorage.setItem("sticky_1", "Cancel cable tv, who needs it now?");
```

Outra chave. Como já dissemos, poderá ser usada qualquer chave que quiser, contanto que seja uma string, mas você só pode armazenar um valor por chave.

Um valor para acompanhar nossa nova chave.

- ③ Agora que temos dois valores armazenados de forma segura em nosso armazenamento local do navegador, você pode utilizar uma das chaves para resgatar seu valor correspondente do localStorage. Assim:

Estamos definindo o valor associado com a chave "sticky_0" do armazenamento local...

... e designando-o a uma variável chamada sticky.

```
var sticky = localStorage.getItem("sticky_0");
```

`alert(sticky);` E para tornar isso mais interessante, vamos usar a função alert para fazer pular na tela o valor do Post-it.

Hora do test drive!

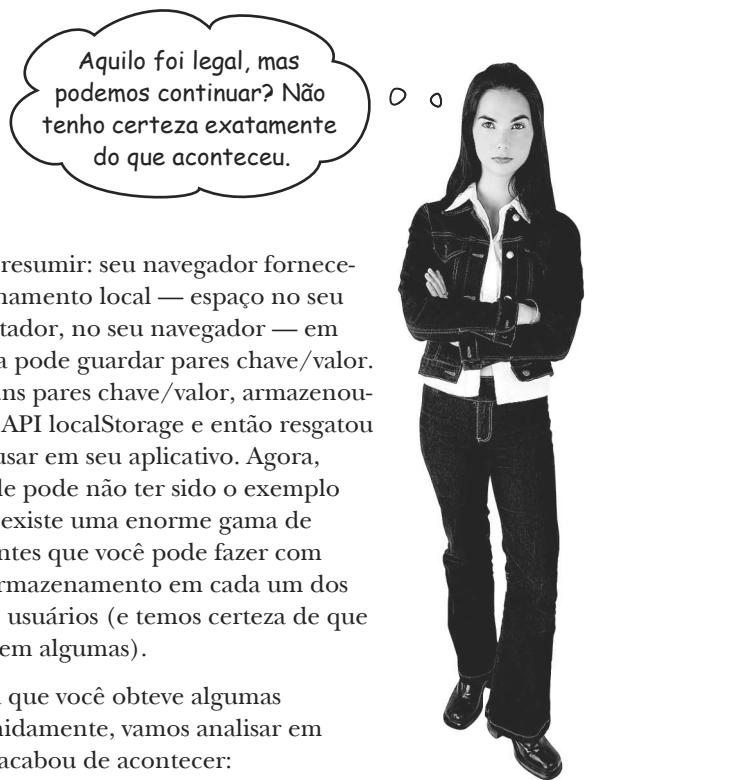
Certifique-se de que todo o código dentro de seu elemento script está digitado e carregue-o dentro dos navegadores. Aqui está o resultado de nosso test drive:



Veja nosso alerta JavaScript, com o valor de sticky_0 como a mensagem de alerta.

O que é legal com relação a isso é que este valor foi armazenado e resgatado do localStorage do navegador! Você poderia desligar seu navegador, sair de férias para Fíki por um mês, voltar e ele ainda estaria lá esperando por você.

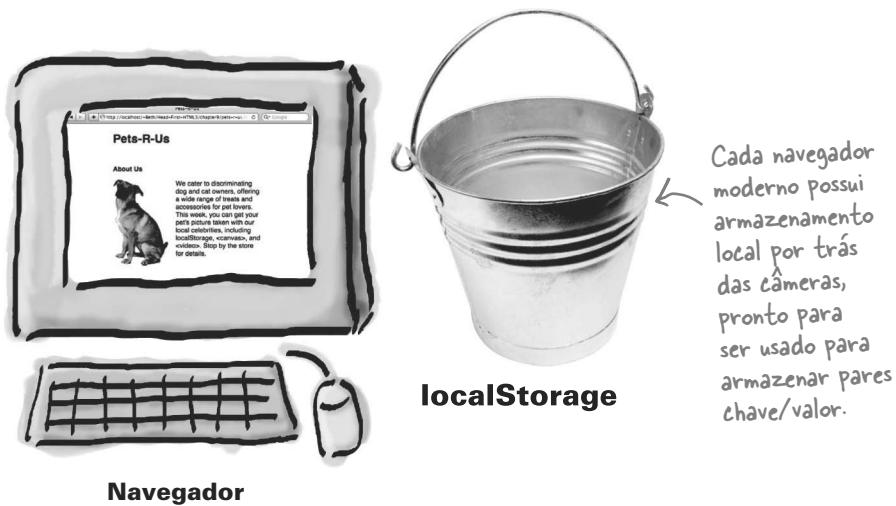
Ok, ok, concordamos que o exemplo poderia ser um pouco mais empolgante, mas vem com a gente. Estamos chegando lá...



Claro. Vamos resumir: seu navegador fornece-lhe um armazenamento local — espaço no seu próprio computador, no seu navegador — em que uma página pode guardar pares chave/valor. Você criou alguns pares chave/valor, armazenou-os usando uma API localStorage e então resgatou um deles para usar em seu aplicativo. Agora, enquanto aquele pode não ter sido o exemplo mais excitante, existe uma enorme gama de coisas interessantes que você pode fazer com um pouco de armazenamento em cada um dos navegadores de usuários (e temos certeza de que você já pensou em algumas).

Portanto, agora que você obteve algumas respostas resumidamente, vamos analisar em detalhes o que acabou de acontecer:

- ① Primeiro, lembre-se de que cada navegador possui um pouco de armazenamento local que você pode usar para armazenar pares chave/valor.

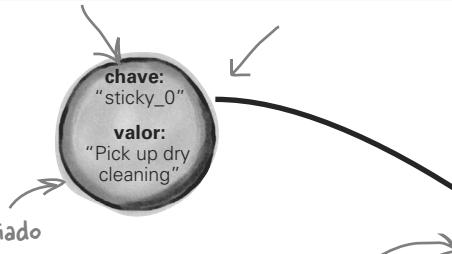


- ② Com aquele armazenamento local, você pode pegar uma chave e um valor (ambos na forma de strings) e armazená-los.

```
localStorage.setItem("sticky_0", "Pick up dry cleaning");
```

Usamos o método `setItem` para armazenar um par chave/valor. A chave é "sticky_0" e o valor é "Buscar a roupa na lavanderia".

O par chave/valor criado pela chamada `setItem`.



Uma vez que você tenha colocado o par chave/valor no `localStorage`, ele será persistentemente armazenado para você, mesmo que feche a janela do navegador, feche o navegador ou reinicie seu computador.

localStorage

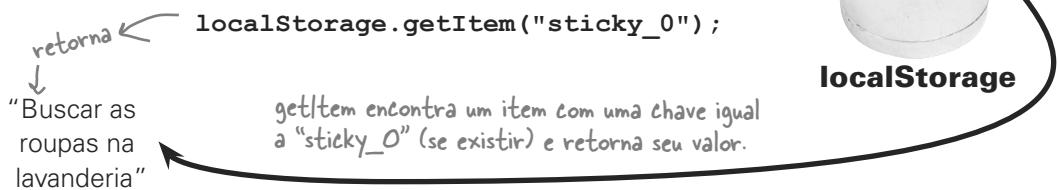
- ③ Então, chamamos `setItem` novamente e armazenamos um segundo par chave/valor, só que dessa vez com uma chave de "sticky_1" e um valor de "Cancel cable tv, who needs it now?" (Cancelar TV à cabo, não preciso agora!).

```
localStorage.setItem("sticky_1", "Cancel cable tv, who needs it now?");
```



Agora existem dois valores armazenados sob duas chaves únicas.

- ④ E, quando chamamos `getItem` com uma chave de "sticky_0", ela retorna o valor do par chave/valor.



`getItem` encontra um item com uma chave igual a "sticky_0" (se existir) e retorna seu valor.

Note que, ao obter um item, não o removemos do armazenamento, ainda está lá. Estamos apenas obtendo o valor para a chave determinada.

Perguntas Idiotas

não existem

P: Primeiro, você disse “Web Storage” e então você começou a falar “local storage”. São a mesma coisa?

R: O padrão web é chamado “Web Storage”, mas a maioria das pessoas simplesmente o chama de local storage (de fato, os navegadores até expõem a API pelo objeto localStorage). Web Storage não é, na verdade, o melhor nome para o padrão (pois os itens são armazenados em seu navegador, em vez de na web). Contudo, não podemos ir mais além. Você nos verá usar o termo local storage mais do que o nome padrão “Web Storage”.

P: Qual a abrangência do suporte da API Web Storage? Posso contar com ela?

R: Sim. De fato, é uma das APIs mais suportadas, desde a época do IE8 até os navegadores mobile mais modernos de hoje. Existem algumas advertências aqui e ali, mas vamos avisá-lo à medida que formos nos deparando com elas. Em termos de contar com o Web Storage, como sempre você deveria testar antes de usar APIs. Veja como pode fazer isso:

```
if (window["localStorage"]) {  
    // your localStorage code here...  
}
```

Note que testamos checando para ver se o objeto window global possui propriedade localStorage. Se estiver lá, saberemos que o navegador suporta localStorage.

P: Bem no começo do capítulo, você mencionou 5MB de armazenamento em cada navegador. São cinco megabytes no total, contando com todos os aplicativos?

R: Não, são realmente cinco megabytes por domínio.

P: Você disse que o servidor não precisava estar envolvido, mas depois você começou a falar sobre domínios.

R: Certo, todo o armazenamento é gerenciado no cliente. O domínio vem à tona, pois cinco megabytes são alocados em todas as páginas a partir do mesmo domínio para armazenamento. Pet-R-Us.com tem cinco, PetEmporium.com mais cinco, e assim por diante, todos na sua máquina.

P: Como isso se compara com Google Gears [ou insira sua tecnologia de armazenamento local proprietária favorita aqui]?

R: Não há nada de errado com outra tecnologia de armazenamento no navegador, mas o local storage do HTML5 é hoje o padrão (e Google, Apple, Microsoft e outros

hoje reconhecem Web Storage como a maneira padrão de armazenar conteúdo localmente no navegador).

P: O que acontece se eu executar um `setItem` na mesma chave múltiplas vezes? Digamos que eu tenha chamado `setItem` duas vezes em “`sticky_1`”, o que acontece? Vou ter dois “`sticky_1`” no armazenamento local?

R: Não. As chaves são únicas no localStorage; portanto, `setItem` irá sobrescrever o primeiro valor no segundo. Aí vai um exemplo; se rodou este código:

```
localStorage.setItem("sticky_1", "Get Milk");  
localStorage.setItem("sticky_1", "Get Almond Milk");  
var sticky = localStorage.getItem("sticky_1");
```

O valor da sticky seria “Get Almond Milk”.

P: Quem pode ver dados em meu armazenamento local?

R: Local storage é gerenciado de acordo com a origem (simplesmente pense na origem como sendo seu domínio) dos dados. Então, por exemplo, toda página em wickedlysmart.com pode ver os itens armazenados por outras páginas naquele site, mas código de outros sites, digamos, google.com, não podem acessar aquele armazenamento (eles podem acessar apenas seus próprios itens no local storage).

P: Quando estou carregando uma página a partir do meu computador, como estamos neste exercício, qual é minha origem?

R: Boa pergunta. Neste caso, sua origem é conhecida como a origem “Local Files”, que é ótima de usar para testes. Se tiver acesso ao servidor, você poderá testar seus arquivos lá também e, assim, chegará na origem do domínio.



Veja bem!

O Local Storage pode não funcionar direito em todos os navegadores, se estiver usando file://.

Este é outro caso em que alguns navegadores requerem que você sirva páginas usando `localhost://` ou um servidor hospedado, em vez de carregar a partir de um arquivo. Portanto, se seus Post-its não estiverem funcionando, tente rodar de um servidor ou tente um navegador diferente.



Joel ↗

Então, posso armazenar strings no localStorage; mas e se eu quiser armazenar um número? Estava pensando se poderia usar localStorage para armazenar itens com números inteiros e preços em decimais para um aplicativo de carrinho de compras que quero escrever. Esta é a tecnologia errada?

Você tem a tecnologia certa.

É verdade. Com o localStorage, você pode usar somente strings como chaves e valores, mas não é tão restrito quanto parece. Digamos que você precise armazenar o número inteiro 5. Você pode armazenar a string “5” no lugar e convertê-la num número inteiro novamente quando resgatá-la do armazenamento local. Vamos dar uma olhada em como fazer isso para inteiros e decimais.

Digamos que você queira armazenar um inteiro com a chave “numitems”. Você escreveria:

```
localStorage.setItem("numitems", 1);
```

O que? Não
acabamos de dizer
que não poderíamos
armazenar inteiros?

Ok, pode parecer que está armazenando inteiros aqui, mas o JavaScript sabe que isto precisa ser uma string. Então, ele faz o valor inteiro virar uma string por você. O que osetItem vê, na verdade, é a string “1”, não um número inteiro. O JavaScript não é tão inteligente quando você resgata um valor com getItem:

```
var numItems = localStorage.getItem("numitems");
```

Neste código, numItems é designado à string “1”, não a um inteiro, como gostaríamos. Para ter certeza de que numItems é um número, você precisa usar a função JavaScript parseInt para converter uma string a um valor inteiro:

Envolvemos o valor numa chamada parseInt,
que converte a string num número inteiro.

```
var numItems = parseInt(localStorage.getItem("numitems"));  
numItems = numItems + 1;
```

Então, o armazenamos
novamente, com o
JavaScript tomando conta
da conversão de novo.

Podemos
adicionar 1
aqui, pois é
um número.

Se estiver armazenando valores decimais, vai querer usar a função parseFloat quando for retirar os itens com preço do localStorage:

A mesma coisa aqui,
armazenamos um decimal que
é convertido numa string.

```
localStorage.setItem("price", 9.99);
```

```
var price = parseFloat(localStorage.getItem("price"));
```

E convertemos de volta para
um decimal com parseFloat.

como o `localStorage` parece com um array

O Local Storage e o Array foram separados na maternidade?

O local storage tem um outro lado que você ainda não viu. Ele não apenas fornece os métodos `getItem` e `setItem`, mas também permite que você trate o objeto `localStorage` como um array associativo. O que isso significa? Bem, em vez de usar o método `setItem`, você pode designar uma chave a um valor no armazenamento, desta forma:

```
localStorage["sticky_0"] = "Pick up dry cleaning";
```

Aqui, a chave parece um index para o array de armazenamento.

E aqui está nosso valor do lado direito de um comando de atribuição.



Podemos também resgatar o valor armazenado numa chave desta forma. Esta é a sintaxe:

```
var sticky = localStorage["sticky_0"];
```

Aqui, designámos nossa variável `sticky`...

... ao valor da chave "sticky_0" no armazenamento local.

Isso funciona exatamente como usar a chamada ao método `getItem`.

Nada mal, hein?! Então, você pode usar ambas as sintaxes, pois são válidas. Se estiver acostumado a usar arrays associativos no JavaScript, esta sintaxe pode ser mais concisa e legível para você.

Espere, tem mais!

A API `localStorage` também fornece duas outras coisas interessantes: uma propriedade, `length`, e um método, `key`. A propriedade `length` detém o número de itens no armazenamento local. Você verá o que o método `key` faz, logo abaixo:

Aqui estamos iterando cada item.

```
for (var i = 0; i < localStorage.length; i++) {  
    var key = localStorage.key(i);  
    var value = localStorage[key];  
    alert(value);  
}
```

Vá em frente e experimente... você recebe um alerta para cada item?

A propriedade `length` nos conta quantos itens estão no `localStorage`.

Então, com o nome da chave, podemos resgatar o valor.

Para cada item no `localStorage`, o método `key` nos dá a chave (como "sticky_0", "sticky_1" e assim por diante).

Visão geral: estamos usando `length` para iterar os conteúdos do `localStorage` (assim como um array) e acessando cada chave (como "sticky_0") à medida que avançamos. Podemos, então, usar aquela chave para extraír seu valor correspondente.

não existem Perguntas Idiotas

P: Quando iterar o localStorage usando localStorage.length e localStorage.key, qual a ordem dos itens? A mesma que escrevi dentro do armazenamento?

R: Na verdade, a ordem dos itens não é definida. O que isso significa? Significa que você verá toda chave/valor no armazenamento ao iterar, mas não deverá contar com qualquer ordem específica em seu código. De fato, diferentes navegadores podem lhe oferecer diferentes ordens para o mesmo código e para os mesmos itens.



Jogo da Concha

Pronto para tentar a sorte? Ou devo dizer habilidade? Temos um jogo para você testar seu comando de localStorage, mas precisará estar atento. Use seu conhecimento de obter e definir pares de chave/valor no localStorage para rastrear a ervilha, à medida que ela muda de concha em concha.

```
function shellGame() {
    localStorage.setItem("shell1", "pea");
    localStorage.setItem("shell2", "empty");
    localStorage.setItem("shell3", "empty");
    localStorage["shell1"] = "empty";
    localStorage["shell2"] = "pea";
    localStorage["shell3"] = "empty";
    var value = localStorage.getItem("shell2");
    localStorage.setItem("shell1", value);
    value = localStorage.getItem("shell3");
    localStorage["shell2"] = value;
    var key = "shell2";
    localStorage[key] = "pea";
    key = "shell1";
    localStorage[key] = "empty";
    key = "shell3";
    localStorage[key] = "empty";

    for (var i = 0; i < localStorage.length; i++) {
        var key = localStorage.key(i);
        var value = localStorage.getItem(key);
        alert(key + ": " + value);
    }
}
```

Você pode digitar isto para verificar sua resposta e ver em qual concha está a ervilha.

Sinta-se à vontade para usar este espaço para rastrear o estado do localStorage.

Qual concha esconde a ervilha? Escreva sua resposta aqui:

| Chave | Valor |
|----------|-------|
| concha 1 | |
| concha 2 | |
| concha 3 | |

Conversa Informal



Bate-papo de hoje: **Cookie e Local Storage**

Esta noite temos a tecnologia incumbente de armazenamento em navegador, o “Cookie”, junto com o novo campeão, Local Storage.

Cookie:

Veja ele, o garoto de ouro, Local Storage.
Estou nesse negócio por mais de uma
década e você pensa que pode chegar como
se soubesse de alguma coisa. Não acha que
está muito convencido?

Você tem alguma ideia de em quantas
páginas eu sou usado? Já olhou para
suas estatísticas?

Ei, sou ubíquo, penetrante, estou em
todos os lugares! Não acho que exista um
navegador no qual você não vá me encontrar
num desktop, dispositivo ou navegador
mobile, não importa qual idade tenham.

Veremos. O que você acha exatamente
que pode oferecer a mais do que eu? Meu
armazenamento funciona muito bem.

Não tenho ideia do que está falando.

Local Storage:

Claro, você pode ver dessa forma, ou
poderia dizer que fui construído baseado
em toda a experiência conquistada a partir
de seus erros.

Dê-me mais alguns anos e dê outra
olhada. A realidade é que estou ajudando
a habilitar toda uma nova geração de
aplicativos web no navegador. Muitas dessas
páginas que você citou são *apenas* páginas.

Estou chegando lá rapidamente. De todas
as tecnologias HTML5, sou uma das mais
bem suportadas.

Bem, não tenho certeza se quero
mencionar isso em público, mas você tem
um probleminha com tamanho.

Olha, você que começou isso tudo, não
eu. Você sabe muito bem que é limitado a
armazenar 4K e eu tenho 1.200 vezes isso!

Cookie:

Sim, sou esguio, ligeiro, pode-se dizer até ágil.

Qual é, sou um livro aberto, sou armazenamento puro para pôr qualquer coisa que quiser.

Ah, e pares chave/valor são alguma inovação grandiosa?

<Snicker> Ah sim, e você armazena tudo como uma string! Belo trabalho! </Snicker>

Sim, sim, fale comigo em dez anos e veremos se suportou o teste do tempo.

Você verá. Ainda irá me ligar chorando quando disserem “Haha, 5 megabytes, só isso que você tem?”

Local Storage:

Há, que ótimo. Você já falou com algum desenvolvedor web? Você é qualquer outra coisa, menos ágil. Dado que você é o Sr. Estatística, tem as estatísticas do número de horas de desenvolvimento perdidas com os erros estúpidos e concepções erradas usando cookies?

O que você realmente quer dizer é que você, essencialmente, não possui qualquer formato de dados, então os desenvolvedores têm de inventar um novo esquema para armazenar dados em cookies.

Não precisamos de uma grande inovação no armazenamento; os pares chave/valor funcionam perfeitamente, são diretos e se adaptam muito bem a muitos aplicativos de computadores.

Você pode tirar vantagem das strings e, se precisar de algo mais complexo, existem maneiras.

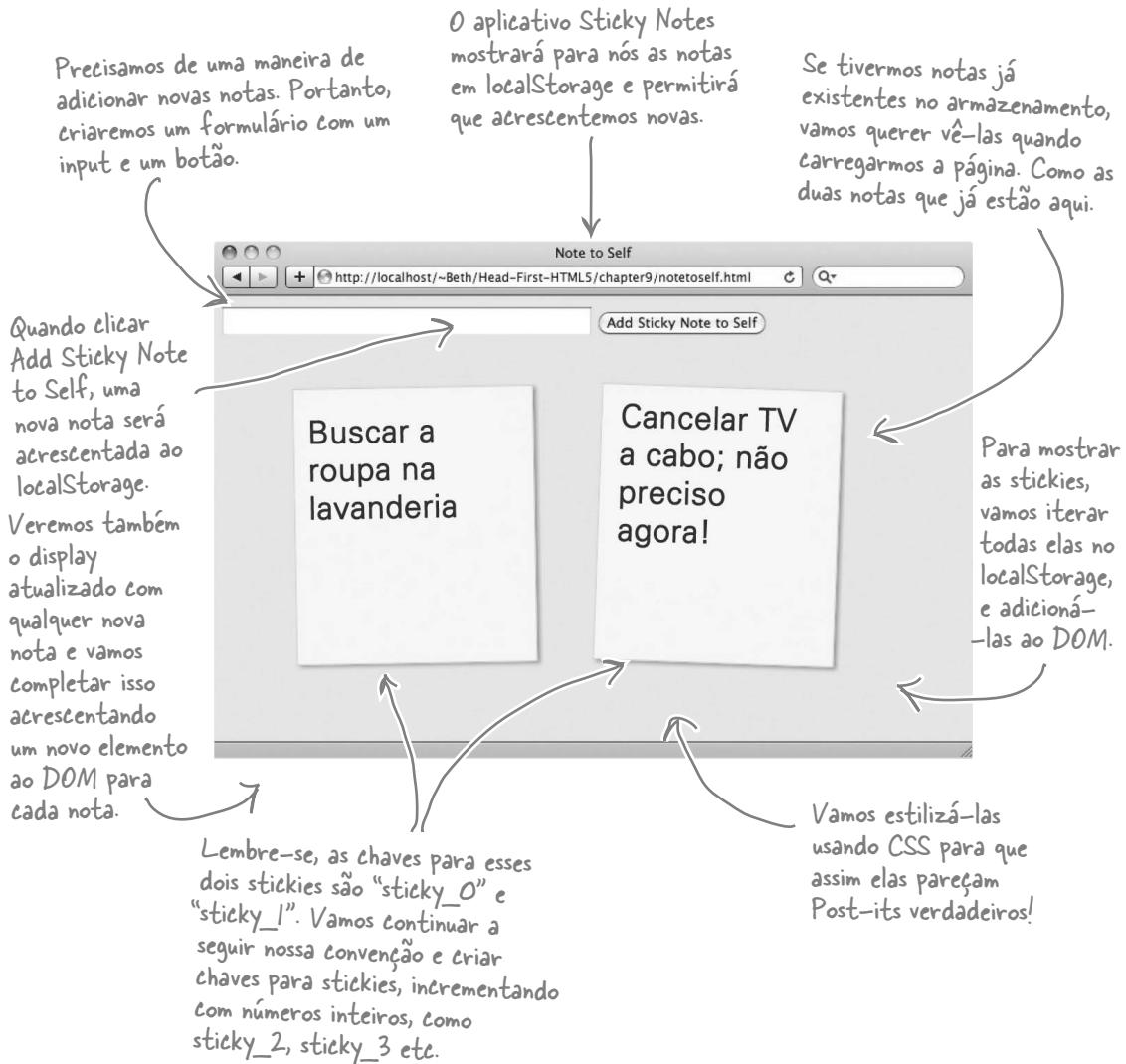
Ah, pode apostar! Seja realista. Você já estava acabado desde o início. Digo, qual é, quem daria aos seus filhos o nome Cookie?

A coisa está ficando séria com as stickies

Agora que você teve um tempo para brincar com o Web Storage, vamos tocar para frente essa implementação. Vamos criar um aplicativo Sticky Notes, de forma que veja suas stickies e adicione outras. Analisaremos o que vamos construir antes de fazê-lo.



localStorage



Criando a interface

Para começar, precisamos de uma maneira de colocar o texto em nossas notas sticky. Seria ótimo se pudéssemos vê-las na página. Então, precisamos de um elemento que mantenha todas as notas nela.

Vamos trabalhar em alguns códigos para fazer isso, começando com a marcação HTML — pegue seu arquivo HTML existente e adicione um elemento `<form>`, o elemento `` e o link do CSS para isso, como segue:



Este é o nosso arquivo HTML principal.

```

<!doctype html>
<html>
<head>
<title>Note to Self</title>
<meta charset="utf-8">
<link rel="stylesheet" href="notetoself.css">
<script src="notetoself.js"></script>
</head>
<body>
<form>
    <input type="text" id="note_text">
    <input type="button" id="add_button" value="Add Sticky Note to Self">
</form>
<ul id="stickies">
</ul>
</body>
</html>

```

Jogamos um pouco de CSS para deixar as coisas um pouco mais reais, como as notas de verdade. Este livro não é sobre CSS, mas fique livre para verificar a fonte!

Vamos mover todo nosso JavaScript para o arquivo "notetoself.js".

Adicionamos um formulário como uma interface de usuário para adicionar novas notas.

E temos que arranjar algum lugar para colocar nossas stickies na interface. Assim, vamos pô-las numa lista desordenada.

O CSS se preocupa em fazer com que cada item da lista pareça um pouco mais com um Post-it real.

Agora, vamos adicionar JavaScript

Temos tudo que precisamos na página agora e algumas notas no localStorage esperando para serem mostradas. Vamos pô-las na página, lendo primeiro a partir do localStorage e, então, as inserindo no elemento unordered list que acabamos de criar. Veja como faremos:

```
Quando a página for carregada,  
vamos chamar a função init...  
↓  
window.onload = init;  
  
function init() {  
    for (var i = 0; i < localStorage.length; i++) {  
        var key = localStorage.key(i);  
        if (key.substring(0, 6) == "sticky") {  
            var value = localStorage.getItem(key);  
            addStickyToDOM(value);  
        }  
    }  
}  
  
... que lê todas as  
notas existentes do  
localStorage e adiciona-  
as à <ul> pelo DOM.  
  
Para isso, iteraremos  
todos os itens no  
armazenamento.  
  
Pegue cada chave.  
E então nos  
certificamos de que  
este item é uma sticky,  
testando para ver  
se sua chave começa  
com "sticky". Por que  
faremos isso? Bem,  
devem existir outros  
itens armazenados no  
localStorage que não  
sejam nossas notas  
(mais sobre isso num  
instante).
```

Se for uma nota,
então pegue seu valor
e adicione-o à nossa
página (via DOM).

Agora, precisamos escrever a função addStickyToDOM, o que irá inserir as notas no elemento :

O texto da sticky note está sendo passado
para nós. Precisamo criar um item de lista
para a unordered list e então inseri-lo.

```
function addStickyToDOM(value) {  
    var stickies = document.getElementById("stickies");  
    var sticky = document.createElement("li");  
    var span = document.createElement("span");  
    span.setAttribute("class", "sticky");  
    span.innerHTML = value;  
    sticky.appendChild(span);  
    stickies.appendChild(sticky);  
}
```

Vamos pegar o elemento
de lista "stickies".

Crie um elemento de
lista e lhe dê um nome da
classe de "sticky" (para
podermos estilizá-lo).

Defina o conteúdo do span
segurando o texto da sticky note.

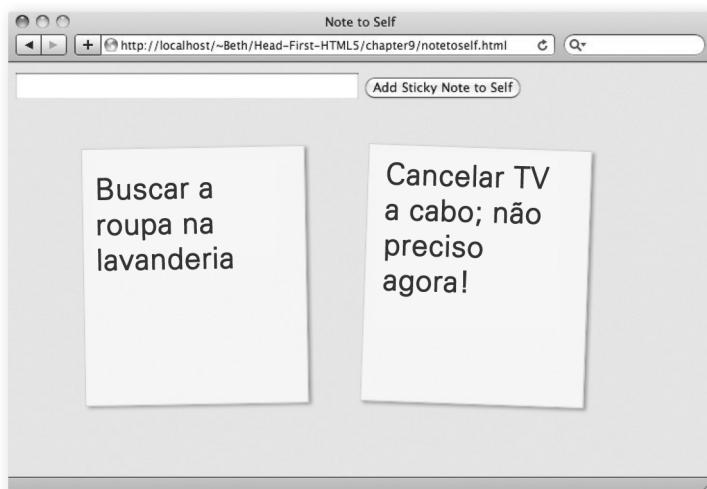
E adicione o span à
"sticky" li, e a li à
"sticky" list.

Hora de outro test drive!



Vá em frente, ponha este código dentro de seu elemento script e carregue-o dentro de seu navegador.

Eis o que temos, quando carregamos a página em nosso navegador:



Completando a interface do usuário

Agora, tudo o que temos a fazer é habilitar o formulário para termos uma maneira de adicionar novas notas. Para tanto, precisamos adicionar um handler para quando o botão “Add Sticky Note to Self” for clicado e também escrever algum código para criar uma nova nota. Veja nosso código para adicionar um handler:

```

Acrecente este novo código
à sua função init: ✓
function init() {
    var button = document.getElementById("add_button");
    button.onclick = createSticky; ←

    // for loop goes here ←
}

O resto do código em init
fica do mesmo jeito. Estamos
salvando algumas árvores por
não repeti-lo aqui.

```

Pegue uma referência ao botão “Add Sticky Note to Self”.

E adicione um handler para quando este for clicado. Vamos chamar o handler createSticky.

criando notas autoadesivas com código

E o código para criar uma nova sticky note;

```
function createSticky() {  
    var value = document.getElementById("note_text").value;  
    var key = "sticky_" + localStorage.length;  
    localStorage.setItem(key, value);  
  
    addStickyToDOM(value);  
}  
  
E, finalmente, acrescentamos  
o novo texto ao DOM para  
representar a sticky.
```

Quando o botão é clicado,
este handler é invocado.

Ele primeiro resgata
o texto no formulário
de caixa de texto.

Depois, precisamos criar
uma única chave para a
sticky. Vamos usar "sticky_"
concatenado com o length do
armazenamento inteiro; vai
continuar aumentando, certo?

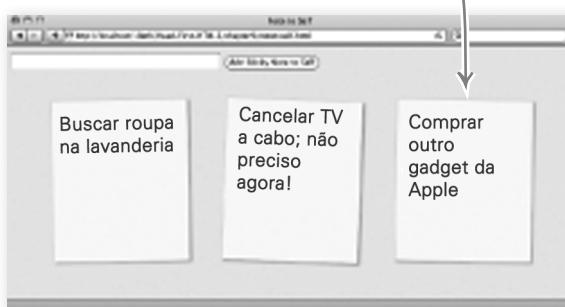
Então, adicionamos
uma nova nota
ao localStorage
usando nossa chave.

Mais um test drive!

Agora estamos verdadeiramente interativos! Carregue este novo código em seu navegador, entre uma nova "sticky note to self" e clique ou toque o botão "Add Sticky Note to Self". Você deverá ver a nova nota aparecer em sua lista de notas. Aqui está o que vemos:

Pode fazer aquela viagem a Fiji agora e, quando voltar,
suas notas ainda estarão esperando por você!

A chave para esta sticky
é "sticky_2", o length
do armazenamento (antes
de o termos adicionado)
concatenado com "sticky_".



Veja nosso teste!
Parece bom!

Certifique-se de tentar fechar sua
janela do navegador e então abrir o
arquivo novamente. Ainda vê as notas?

Perguntas Idiotas

P: Por que testamos para ver se cada chave de item começa com a string "sticky"?

R: Lembre-se de que todas as páginas de um domínio (como apple.com) podem ver cada item armazenado de outras páginas naquele domínio. Isso significa que, se não tivermos cuidado quanto à nomeação de nossas chaves, podemos conflitar com outra página que esteja usando a mesma chave de maneira diferente. Então, esta é nossa maneira de checar para ter certeza de que um item é uma sticky (o que não é dizer um número ordenado ou um nível de jogo) antes de usarmos seu valor para uma sticky note to self.

P: E se existirem vários itens no localStorage, incluindo vários itens que são stickies? Não seria ineficiente iterar o conjunto de itens inteiro?

R: Bem, a menos que você já esteja falando sobre um número bem grande de itens, duvidamos que você note a diferença. Dito isto, você está certo. Não é eficiente e talvez haja maneiras melhores de abordar o gerenciamento de nossas chaves (falaremos sobre algumas delas em breve).

P: Estou pensando a respeito de usar localStorage.length como o número de sticky na chave. Como em

`"sticky_" + localStorage.length`

Por que fizemos isso?

R: Precisamos de uma maneira de criar novas chaves que são únicas. Poderíamos usar algo como o tempo ou gerar um número inteiro aumentariam a cada momento. Ou, como fizemos, podemos usar o length do armazenamento (o qual aumenta a cada vez que adicionamos um item). Se estiver pensando que isso pode ser problemático, voltaremos a isso. E se não pensou, não se preocupe, ainda voltaremos a isso.

P: Criei um monte de notas no Safari e então mudei para o Chrome e agora não vejo nenhuma delas. Por que não?

R: Cada navegador mantém seu próprio local storage. Então, se você criar stickies no Safari, só vaivê-las lá.

P: Acabei de recarregar minha página e agora minhas stickies estão numa ordem diferente!

R: Quando adiciona uma nova nota, o novo item é anexado à lista de notas, sempre indo para o fim dela. Quando carrega a página, as notas são adicionadas na ordem em que são encontradas no localStorage (o que, lembre-se, não é garantido estar em nenhuma ordem em particular). Você deve ter pensado que a ordem seria a mesma em que os itens foram adicionados ao armazenamento, ou alguma outra ordem razoável, contudo. Não conte com isso. Por quê? Bem, uma razão é que as especificações não definem uma ordem. Então, navegadores diferentes podem implementar isso de diferentes formas. Se seu navegador parece retornar itens numa ordem que faz sentido para você, considere-se uma pessoa de sorte, mas não conte com o ordenamento, pois o navegador de seu usuário pode ordenar seus itens de outra maneira.

P: Frequentemente uso o formulário "for in" do for loop. Isso vai funcionar aqui?

R: Claro que sim. Vai ficar assim:

Isto vai iterar
cada chave no
localStorage.
Muito útil.
`for (var key in localStorage) {
 var value = localStorage[key];
}`

P: E se eu não quiser mais determinada sticky?
Posso deletá-la?

R: Sim, podemos deletar itens do localStorage, usando o método localStorage.removeItem. Você também pode remover itens diretamente do localStorage, usando o console do navegador. Vamos mostrar ambos neste capítulo.



Dada a maneira com que as notas são implementadas, haveria um problema com nosso esquema de nomeação, se um usuário pudesse deletar uma nota ao seu bel-prazer. Consegue imaginar qual seria esse problema?

Precisamos parar para um reparo na agenda

Não seria ótimo se houvesse uma ferramenta para visualizar diretamente os itens em seu localStorage? Ou uma ferramenta para deletar itens ou mesmo limpar tudo e começar desde o princípio quando estivesse tirando bugs?

Bem, todos os navegadores de maior importância são enviados com ferramentas de desenvolvedor embutidas, que lhe permitem examinar diretamente seu armazenamento local. Como era de se esperar, essas ferramentas diferem entre navegadores. Sendo assim, em vez de falarmos sobre cada um aqui, vamos apontar na direção certa e, então, você mesmo poderá fuçar e descobrir as especificações de seu navegador. Como exemplo, vamos ver o que o Safari oferece:

Clicamos na aba Recursos para inspecionar o localStorage

Sem contar que novas versões dos navegadores estão surgindo mais rápido do que podemos escrever essas páginas!

Ferramentas de desenvolvedor, como aparecem no Safari.

| Key | Value |
|----------|------------------------------------|
| cat | Oliver |
| sticky_2 | Buy another Apple gadget |
| sticky_0 | Pick up dry cleaning |
| sticky_1 | Cancel cable tv, who needs it now? |

O par de valores da chave para cada item no armazenamento está aqui.

Ao clicar, mostrará o storage associado com esta origem.

Falaremos sobre isso mais tarde.

Ao clicar com o botão direito sobre um dos itens do storage, você pode editar ou deletar o item na própria ferramenta.

Cookies da velha guarda, se você os quiser.

A origem do storage. Aqui estamos usando arquivos locais servidos do `http://localhost`, mas isso pode ser também um nome de domínio, se estiver testando num servidor hospedado.

No Safari, podemos ver essas ferramentas para recarregar a visualização do Storage e deletar um item selecionado.

Para habilitar ou acessar as ferramentas de desenvolvedor, como dissemos, você precisará fazer coisas diferentes para diferentes navegadores. Direcione seu navegador para `http://wickedlysmart.com/hfhtml5/devtools.html` para ver como fazer isso em seu navegador específico.



Manutenção Faça-Você-Mesmo

Há uma outra maneira de fazer uma faxina em seus itens (e, como veremos num segundo, para deletá-las uma a uma), que requer que você mesmo faça uma pequena manutenção, direto do JavaScript. A API localStorage inclui um método útil, `clear`, que deleta todos os itens a partir de seu armazenamento local (pelo menos aqueles de seu domínio). Vamos dar uma olhada em como podemos usar esta chamada em JavaScript ao criar um novo arquivo chamado `maintenance.html`. Uma vez que tenha feito isso, adicione o código abaixo e vamos guiá-lo passo a passo.

```
<!doctype html>
<html>
<head>
<title>Maintenance</title>
<meta charset="utf-8">
<script>
window.onload = function() {
    var clearButton = document.getElementById("clear_button");
    clearButton.onclick = clearStorage;
}

function clearStorage() {
    localStorage.clear();
}
</script>
</head>
<body>
<form>
    <input type="button" id="clear_button" value="Clear storage" />
</form>
</body>
</html>
```

← Esta é uma boa ferramenta para a sua caixa.

← Adicionamos um botão para a página e este código adiciona um click handler para o botão.

← Quando clicar no botão, a função clearStorage é chamada.

← Tudo o que essa função faz é chamar o método localStorage.clear. Use-a com cuidado, já que deletará todos os itens associados com a origem desta página de manutenção!

↑ E aqui está nosso botão. Use este arquivo sempre que precisar apagar tudo no localStorage (bom para testar).

Depois de ter digitado o código, vá em frente e carregue-o em seu navegador. É seguro (com relação ao nosso aplicativo Sticky Notes) continuar a limpar seu localStorage agora, portanto, experimente! Tenha certeza de que descobriu suas ferramentas de desenvolvedor primeiro. Assim poderá observar as mudanças.



Veja bem!

Isso deleta todos os itens de seu domínio

Se você tem um armazenamento local super valioso relacionado a outro projeto no mesmo domínio, você perderá todos os seus itens rodando este código. Só estou falando...



Estou com um problema. Enquanto estava fazendo os exercícios do livro, também usei meu conhecimento para criar o novo carrinho de compras de nossa empresa. Meu aplicativo Sticky Notes parou de funcionar. Quando olhei para localStorage com as ferramentas de desenvolvedor do Safari, vi que a contagem das minhas notas estava toda desarrumada. Tenho "sticky_0", "sticky_1", "sticky_4", "sticky_8", "sticky_15", "sticky_16", "sticky_23", "sticky_42". Estou com um pressentimento de que isto está acontecendo porque estou criando outros itens no localStorage, ao mesmo tempo que as notas. Que diabos está acontecendo?

Ah, você descobriu uma das maiores falhas de design.

Tudo bem, é hora de abrir o jogo: construímos um ótimo pequeno aplicativo até agora, e ele deveria funcionar perfeitamente por anos a fio, *contanto que não sejam introduzidos outros itens* dentro do localStorage (como Joel fez com seu carrinho de compras). Uma vez que faça isso, todo nosso trabalho de rastrear notas não vai funcionar mais ou, pelo menos, não vai mais funcionar bem. Veja o porquê:

Antes de tudo, nossas notas são numeradas de zero ao número de notas (menos um):



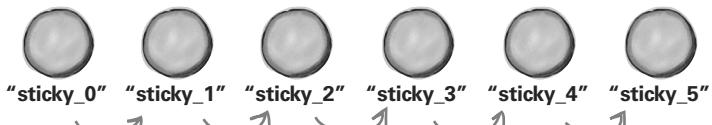
Cinco notas, rotuladas de zero a quatro.

Se estiver disposto a viver com isso, tudo bem; do contrário, é melhor continuar a ler.

Para adicionar uma nova nota, contamos o número de itens no armazenamento local e criamos nossa nova chave a partir daquele número:

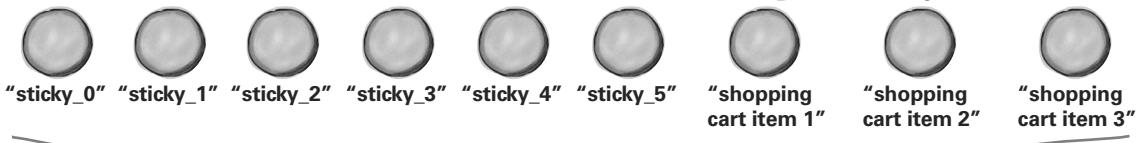
```
var key = "sticky_" + localStorage.length;
```

Para mostrar todas as notas, iteramos de zero à length do armazenamento local (menos um):



Length é agora seis, então iteramos de zero a cinco, mostrando cada nota de "sticky_0" a "sticky_5".

Agora vamos adicionar os itens do carrinho de compras de Joel para o localStorage:



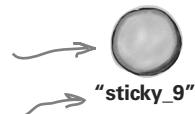
Esses são os itens que Joel está usando em seu código do carrinho de compras.



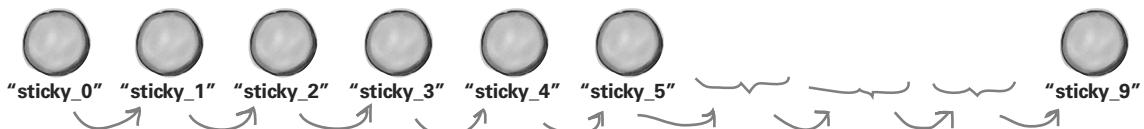
Agora temos nove itens no total no localStorage.

Vamos criar uma nova sticky:

```
var key = "sticky_" + localStorage.length;
O length de nosso armazenamento local é agora nove.
Então, ao criarmos nossa nova nota, ela é chamada
"sticky_9". Mmm, não parece certo.
```



Quando precisamos iterar as notas para mostrá-las, estamos fritos:



O length é agora dez (apenas acrescentamos uma nova nota), assim iteraremos de zero a nove, mostrando cada sticky de "sticky_0" a "sticky_9".

O-oh, não há "sticky_6", "sticky_7" ou "sticky_8".

Aponte o seu lápis

Marque a seguir as maneiras com que nossa implementação atual pode causar problemas:

- Mostrar as notas é inefficiente se houver vários itens em localStorage que não são stickies.
- Uma nota poderá ser sobreescrita por setItem, se o tamanho do localStorage ficar menor, quando outro aplicativo deletar seus próprios itens.
- É difícil dizer rapidamente quantas notas existem; você terá de iterar cada item no localStorage para conseguir todas as stickies.
- Use um cookie. Deve ser mais fácil do que tudo isso!



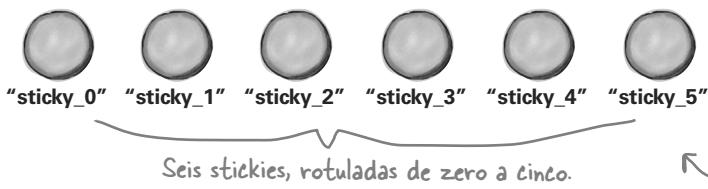
Se pudesse, armazenaria apenas um array num localStorage. Poderíamos usá-lo para manter todas as chaves das notas e, também, para, sempre facilmente, saber o número de notas que estivéssemos armazenando. Todos sabemos, porém, que o localStorage armazena apenas strings. Então, mesmo que um array seja maravilhoso, sei que é apenas uma fantasia...

Nós temos a tecnologia

Não mentimos. É verdade que você só pode armazenar strings como valores dos itens no localStorage, porém essa não é toda a verdade, pois podemos sempre converter um array (ou um objeto) numa string antes de armazená-lo. Claro, parece uma trapaça, mas é uma maneira completamente legítima de armazenar seus tipos de dados não-String no localStorage.

Sabemos que você está morrendo de vontade de pular direto para a parte legal sobre como armazenar arrays, mas, antes disso, vamos primeiro analisar como um array resolveria de fato nosso (e do Joel) problema.

Vamos recapitular e imaginar que temos seis notas no localStorage:



e temos um array no localStorage chamado "stickiesArray":

| | | | | | |
|------------|------------|------------|------------|------------|------------|
| "sticky_0" | "sticky_1" | "sticky_2" | "sticky_3" | "sticky_4" | "sticky_5" |
|------------|------------|------------|------------|------------|------------|

"stickiesArray"

As stickies e os arrays de stickies são todos armazenados no localStorage.

Cada elemento do array de stickies é uma chave para uma sticky no localStorage.

Agora, vamos adicionar uma nova sticky. Chamaremos a sticky "sticky_815". Por que um número tão maluco? Porque não iremos nos importar como será chamada a partir de agora, contanto que seja única. Portanto, para adicionar a sticky, apenas acrescentamos "sticky_815" ao array e então armazenamos um item para ela, exatamente como antes. Assim:

| | | | | | | |
|------------|------------|------------|------------|------------|------------|--------------|
| "sticky_0" | "sticky_1" | "sticky_2" | "sticky_3" | "sticky_4" | "sticky_5" | "sticky_815" |
|------------|------------|------------|------------|------------|------------|--------------|

Temos uma sticky extra no localStorage.

Sete stickies: suas chaves não importam mais, só precisam ser únicas.

| | | | | | | |
|------------|------------|------------|------------|------------|------------|--------------|
| "sticky_0" | "sticky_1" | "sticky_2" | "sticky_3" | "sticky_4" | "sticky_5" | "sticky_815" |
|------------|------------|------------|------------|------------|------------|--------------|

"stickiesArray"

E ampliamos o array de stickies a um valor.

Retrabalhando nosso aplicativo para usar um array

Ok, sabemos como vamos rastrear nossas stickies usando um array, mas vamos um pouco mais à frente e garantir que podemos iterar e mostrar todas as notas. No código atual, nós mostramos todas as notas na função `init`. Podemos reescrevê-la usando um array? Primeiro, vamos procurar o código existente e, depois, ver como ele muda (tomara que para melhor) com um array. Não digite este código ainda; estamos nos focando nas mudanças que precisamos fazer por ora e não tornando este código à prova de balas. Traremos esse negócio de à prova de balas num instante.

Antes...

```
function init() {
    // button code here...
    for (var i = 0; i < localStorage.length; i++) {
        var key = localStorage.key(i);
        if (key.substr(0, 6) == "sticky") {
            var value = localStorage.getItem(key);
            addStickyToDOM(value);
        }
    }
}
```

Este é nosso antigo código que reside nas stickies com nomes específicos, `sticky_0`, `sticky_1` e assim por diante.

Nossa, pensando melhor, isso estava uma confusão.

Como sabemos, isso pode não dar certo porque não podemos depender do fato de que todas as notas estarão lá, se estivermos nomeando-as baseado-nos na contagem de itens do `localStorage`.

Novo e melhorado

```
function init() {
    // button code here...
    var stickiesarray = localStorage["stickiesarray"];
    if (!stickiesarray) {
        stickiesarray = [];
        localStorage.setItem("stickiesarray", stickiesarray);
    }
    for (var i = 0; i < stickiesarray.length; i++) {
        var key = stickiesarray[i];
        var value = localStorage[key];
        addStickyToDOM(value);
    }
}
```

Começamos por tirar o `stickiesArray` do `localStorage`.

Precisamos nos certificar de que haja um array no `localStorage`. Se não houver, então criaremos um vazio.

Estamos iterando o array aqui.

Nota: você ainda não sabe armazenar e resgatar arrays do `localStorage`, portanto, trate isso como um pseudocódigo até lhe mostrarmos. Teremos de fazer um pequeno acréscimo para que isto funcione.

Depois, adicionamos aquele valor ao DOM, assim como já fizemos.

Cada elemento do array é a chave de uma nota, então usaremos isso para resgatar o item correspondente do `localStorage`.



Ainda precisamos descobrir como armazenar efetivamente um array no localStorage.

Você já deve ter imaginado que podemos usar JSON para criar uma representação de string de um array e, se assim o fez, estava certo. Uma vez que a tenha, poderá armazenar no localStorage.

Lembre-se de que existem apenas dois métodos na API JSON: stringify e parse. Vamos pôr esses métodos para funcionar, finalizando a função init (veja a solução no fim do capítulo antes de seguir em frente):

```
function init() {
    // button code here...
    var stickiesarray = localStorage["stickiesarray"];
    if (!stickiesarray) {
        stickiesarray = [];
        localStorage.setItem("stickiesarray", _____(stickiesarray));
    } else {
        stickiesarray = _____(stickiesarray); ← Acrescentamos esta
    }
    for (var i = 0; i < stickiesarray.length; i++) {
        var key = stickiesarray[i];
        var value = localStorage[key];
        addStickyToDOM(value);
    }
}
```

Convertendo createSticky para usar um array

Quase terminamos este aplicativo. Tudo o que precisamos fazer é retrabalhar o método createSticky, o qual, como você deve lembrar, apenas pega o texto para a nota do formulário, armazena-o localmente e depois o mostra. Vamos olhar para a implementação atual antes de mudá-la:

```
function createSticky() {
    var value = document.getElementById("note_text").value;
    var key = "sticky_" + localStorage.length;
    localStorage.setItem(key, value);
    addStickyToDOM(value);
}
```

Em vez de usar o localStorage length para criar uma chave, que vimos poder causar problemas, precisaremos criar uma chave única a mais.

Também vamos precisar adicionar a sticky em nosso array de sticky e salvar o array no localStorage.

O que precisa mudar:

Temos duas coisas que precisam ser mudadas em `createSticky`. Primeiro, precisamos de uma nova forma de gerar uma chave para cada nota que seja única. Também precisamos alterar o código de maneira que armazene a nota no `stickyArray` no `localStorage`.

1 Precisamos criar uma chave única para a sticky

Existem diversos jeitos de criar chaves únicas. Poderíamos usar data e hora, ou criar incríveis números aleatórios de 64 bits, ou conectar nosso aplicativo a uma API de relógio atômico. Mmm, data e hora parece uma maneira boa e fácil de se fazer isso. O JavaScript suporta um objeto `date` que retorna o número de milissegundos desde 1970; isso deve ser único o bastante (a menos que você vá criar suas notas a uma taxa *muito rápida*):

Crie um objeto Date, então obtenha o tempo atual em milissegundos.

```
var currentDate = new Date();
var time = currentDate.getTime();
var key = "sticky_" + time;
```

Então, crie a chave, anexando os milissegundos à string "sticky_".

Nosso novo código para criar uma chave única.

2 Precisamos armazenar a nova nota no array

Agora que temos uma maneira de criar uma chave única, precisamos armazenar o texto da sticky com aquela chave e adicionar a chave ao `stickiesArray`. Vamos ver como fazer isso e, em seguida, juntar todos os códigos.

Primeiro, pegamos o array das stickies.

```
var stickiesArray = getStickiesArray();
localStorage.setItem(key, value);
stickiesArray.push(key);
localStorage.setItem("stickiesArray",
  JSON.stringify(stickiesArray));
```

E armazenamos o array de volta ao `localStorage`, passando o `stringify` nele primeiro.

Em vez de repetir todo o código para obter e verificar o `stickiesArray`, como fizemos na `init` (na página anterior), vamos criar uma nova função para isso. Chegaremos lá num segundo.

Depois, armazenamos a chave com seu valor como sempre fizemos (apenas com nossa nova chave).

Então, usamos o método `array push`, que anexa a chave ao fim do array `stickies`.

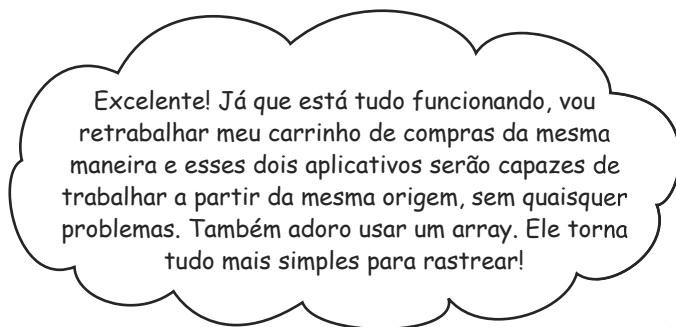
não existem perguntas idiotas

P: O que são milissegundos desde 1970?

R: Você já deve saber que um milissegundo é um milésimo de um segundo, e o método `getTime` retorna uma conta de milissegundos que ocorre desde 1970. Por que 1970? Este comportamento é herdado do sistema operacional Unix, que definiu o tempo dessa maneira. Embora não seja perfeito (por exemplo, representa o tempo antes de 1970 com números negativos), vem a ser bastante útil quando precisamos de um número único ou controlar o tempo em código JavaScript.

P: Esses tipos JSON de parse e `stringify` não são inefficientes? E se meu array ficar muito grande, também não será inefficiente para armazenar?

R: Teoricamente, sim em ambos os casos, mas para tarefas típicas de programação de páginas, normalmente não são um problema. Dito isso, se estiver implementando um aplicativo sério com vários requerimentos de armazenamento de grande porte, você poderá ver alguns problemas usando JSON para converter itens em strings e de strings.



Juntando tudo

Está na hora de integrar todo esse código baseado em arrays, incluindo as funções `init` e `createSticky`.

Para isso, primeiro vamos resumir uma pequena parte do código que é necessária em ambas as funções — é o código que resgata o array `stickies` do `localStorage`. Você o viu em `init` e precisamos dele novamente em `createSticky`.

Vamos pegar esse código e colocá-lo num método chamado `getStickiesArray` — talvez lhe pareça familiar, já que é o código que analisamos antes:

```
function getStickiesArray() {
  var stickiesArray = localStorage.getItem("stickiesArray");
  if (!stickiesArray) {
    stickiesArray = [];
    localStorage.setItem("stickiesArray", JSON.stringify(stickiesArray));
  } else {
    stickiesArray = JSON.parse(stickiesArray);
  }
  return stickiesArray;
}
```

Primeiro, tiramos o item "stickiesArray" do localStorage.

Se esta é a primeira vez que carregamos este aplicativo, talvez não haja um item "stickiesArray".

E se não houver um array ainda, criaremos um vazio e, depois, o armazenaremos de volta no localStorage.

Não se esqueça de passar o stringify primeiro!

Do contrário, encontramos o array no localStorage e precisamos passar o parse para convertê-lo num array JavaScript.

Em todo caso, acabamos com um array e o retornamos.

Juntando tudo, continuação...

Com o `getStickiesArray` escrito, vamos para as versões finais simplificadas das funções `init` e `createSticky`. Vá em frente e digite isto:

```
function init() {
    var button = document.getElementById("add_button");
    button.onclick = createSticky;
}

var stickiesarray = getStickiesarray();

for (var i = 0; i < stickiesarray.length; i++) {
    var key = stickiesarray[i];
    var value = localStorage[key];
    addStickyToDOM(value);
}

}
```

Lembre-se de que também criamos o botão events aqui no método init.

Depois, pegamos o array com as chaves das stickies nele.

Agora, vamos iterar o array de stickies (não os itens localStorage!).

Cada item no array é uma chave para uma sticky. Vamos pegar cada uma delas.

E pegar seu valor de localStorage.

E adicioná-lo ao DOM, do mesmo jeito que vínhamos fazendo.

Com a `init` terminada, apenas sobrou `createSticky`:

```
function createSticky() {
    var stickiesarray = getStickiesarray();
    var currentDate = new Date();
    var key = "sticky_" + currentDate.getTime();
    var value = document.getElementById("note_text").value;
    localStorage.setItem(key, value);
    stickiesarray.push(key);
    localStorage.setItem("stickiesarray", JSON.stringify(stickiesarray));
    addStickyToDOM(value);
}


```

Começamos pegando o array de stickies.

Depois, vamos criar aquela chave única para nossa nova nota.

Adicionamos chave/valor da sticky em localStorage.

E acrescentamos a nova chave ao array de stickies...

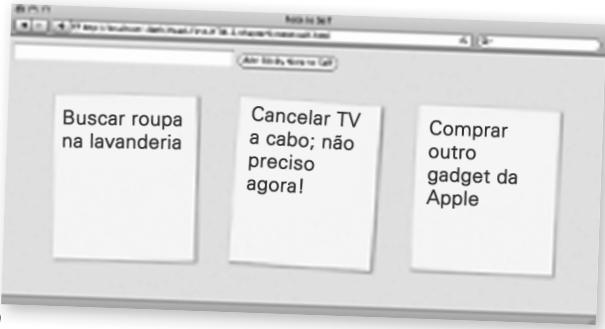
Finalmente, atualizamos a página com a nova sticky, adicionando a sticky no DOM.

Em seguida, passamos o `stringify` no array e o escrevemos de volta para o localStorage.

Test Drive!



Pegue todo esse código e limpe seu localStorage para ter um bom começo. Carregue o código, devendo ter exatamente o mesmo comportamento que da última vez. Joel, você deverá ver seu código funcionando perfeitamente agora!



^{não existem} Perguntas Idiotas

P: Estamos usando “sticky_” como o prefixo para os nossos nomes de item para o localStorage. Há uma convenção para nomear trabalhos com o localStorage?

R: Não há convenção para nomear itens no localStorage. Se seu aplicativo estiver num pequeno site, num domínio sobre o qual você tem poder, então a nomeação não deverá ser um problema para você, já que terá total consciência dos nomes usados por todas as diferentes páginas no site. Achamos que seria uma boa ideia usar um nome que indicasse a página ou o aplicativo web relacionado com aquele item. Portanto, “sticky_” nos ajuda a lembrar que aqueles itens estão relacionados com o aplicativo Sticky Notes.

P: Então, se meu aplicativo Sticky Notes é apenas um de muitos aplicativos num domínio, tenho que me preocupar com potenciais conflitos, certo?

R: Sim. Nesse caso, seria uma boa para você (ou para alguém que gerencie os sites do domínio) criar um plano sobre como nomear os itens.

P: Se possuo diversos stickies, meu stickiesArray vai ficar muito grande. Isso é um problema?

R: A menos que você crie milhares de notas, não deverá ser (e mesmo que crie milhares de notas, queremos que saiba como você é produtivo!). O JavaScript é bem rápido hoje em dia.

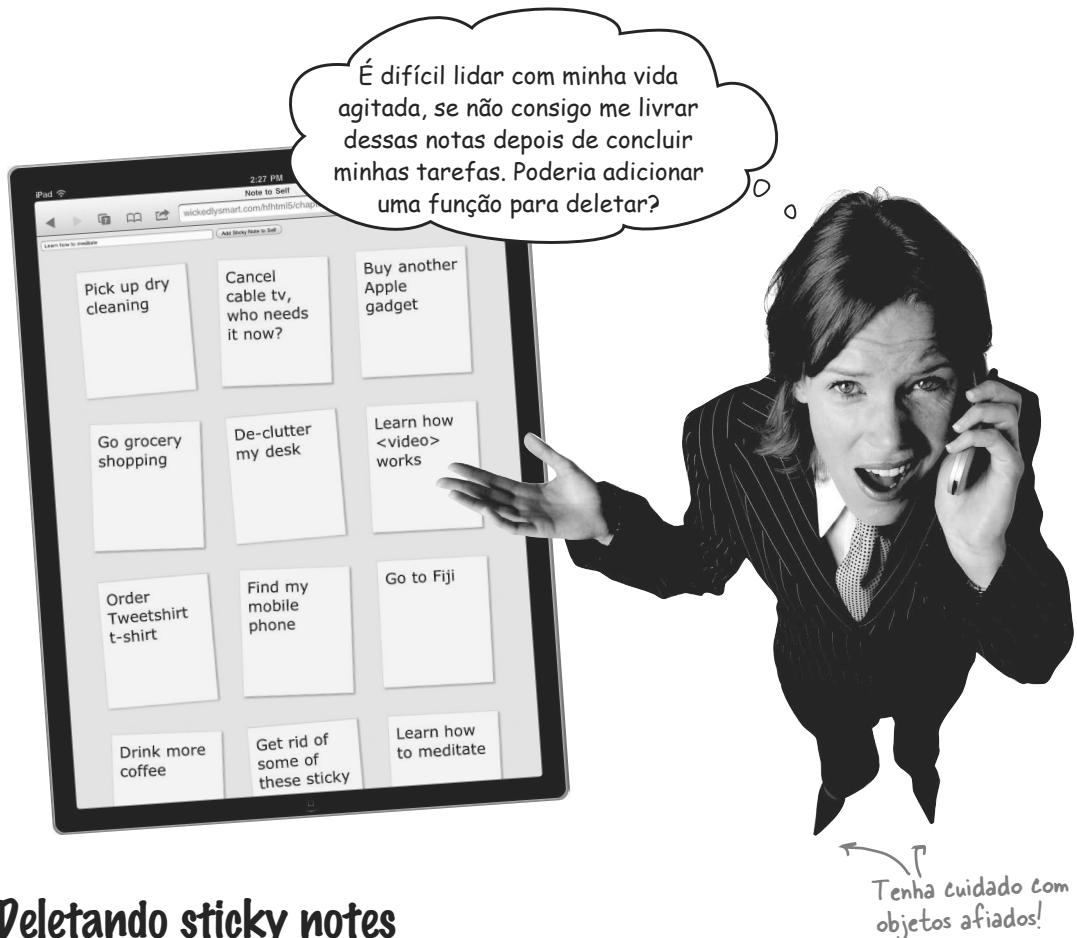
P: Então, só para esclarecer, podemos armazenar qualquer objeto no localStorage só por passar stringify nele antes com JSON?

R: Certo. As strings JSON são versões simplificadas de objetos JavaScript, e os mais simples métodos JavaScript podem ser transformados numa string usando JSON e armazenados no localStorage. Isso inclui arrays (como já viu), assim como objetos contendo nomes de propriedade e valores, como verá em breve.

Escolha nomes de trabalho para seus itens no localStorage, que não conflitarão com aqueles de outros aplicativos no mesmo domínio.

Se precisar armazenar arrays ou objetos no localStorage, use JSON.stringify para criar o valor para armazenar e JSON.parse depois de resgatá-lo.

outro recurso solicitado: deletar



Deletando sticky notes

Ela está certa. Este aplicativo não será muito bem-sucedido se não pudermos remover as notas. Já mencionamos o método `localStorage.removeItem` neste capítulo, mas não falamos direito sobre ele. O método `removeItem` pega a chave de um item e o remove do `localStorage`.

```
localStorage.removeItem(key);
```

Este método remove o item do `localStorage` com a dita chave.

O `removeItem` tem um parâmetro: a chave do item a ser removido.

Isso parece fácil o bastante, não? Ah, mas se pensar melhor sobre isso, há mais para remover uma nota do que chamar o método `removeItem` — também precisamos lidar com o `stickyArray`...



Aponte o seu lápis

Vamos deletar uma nota!

Abaixo, verá os conteúdos de localStorage. Você tem todo o JavaScript que quiser com o método `removeItem`. Usando um lápis, esboce o que precisa fazer para remover `sticky_1304220006342` do localStorage. Depois disso, vá em frente e escreva um pseudocódigo abaixo para mostrar como poderá escrever seu código.



`"sticky_1304294652202"` `"sticky_1304220006342"` `"sticky_1304221683892"` `"sticky_1304221742310"` `"shopping cart item 1"` `"shopping cart item 2"`

`"sticky_1304294652202"` `"sticky_1304220006342"` `"sticky_1304221742310"` `"sticky_1304221683892"`
`"stickiesArray"`

← Seu pseudocódigo aqui



Aponte o seu lápis Solução

Vamos deletar uma nota!

Abaixo, verá os conteúdos de localStorage. Você tem todo o JavaScript que quiser com o método `removeItem`. Usando um lápis, esboce o que precisa fazer para remover `sticky_1304220006342` do localStorage. Depois disso, vá em frente e escreva um pseudocódigo abaixo para mostrar como poderá escrever seu código. Veja nossa solução.

`localStorage.removeItem("sticky_1304220006342");`



`"sticky_1304294652202" "sticky_1304220006342" "sticky_1304221742310" "sticky_1304221683892"
"stickiesArray"`

- (1) Remova a nota com a chave `"sticky_1304220006342"` do localStorage usando o método `localStorage.removeItem`.
- (2) Pegue o `stickiesArray`.
- (3) Remova o elemento com `key="sticky_1304220006342"` do `stickiesArray`.
- (4) Escreva `stickiesArray` de volta no localStorage (primeiro o `stringifying`).
- (5) Encontre `"sticky_1304220006342"` no DOM e remova-o.

A função deleteSticky

Você fez um plano para deletar notas. Então, vamos dar uma olhada na função deleteSticky:

```
function deleteSticky(key) {
    localStorage.removeItem(key);
    var stickiesArray = getStickiesArray();
    if (stickiesArray) {
        for (var i = 0; i < stickiesArray.length; i++) {
            if (key == stickiesArray[i]) {
                stickiesArray.splice(i, 1);
            }
        }
    }
    localStorage.setItem("stickiesArray", JSON.stringify(stickiesArray));
}
```

Primeiro, removemos a sticky note do localStorage usando removeItem, passando a chave da nota para deletar.

Estamos usando a função getStickiesArray para tirar o stickiesArray do localStorage.

Nos certificamos de ter um stickiesArray (só por precaução) e então iteramos o array procurando pela chave que queremos deletar.

Quando encontramos a chave certa, deletamos do array usando splice.

O splice remove os elementos de um array, começando pela localização dada pelo primeiro argumento (i), para tantos elementos quanto forem especificados no segundo argumento (1).

Finalmente, salvamos o stickiesArray (com a chave removida) de volta para o localStorage.



O

Entendi o código, mas não entendo como passamos a chave para deleteSticky. Pensando bem, como o usuário escolhe a nota a ser deletada, antes de qualquer coisa?

Como selecionar uma nota para ser deletada?

Precisamos encontrar uma maneira que permita ao usuário selecionar uma nota a ser deletada. Poderíamos incrementar tudo e adicionar um pequeno ícone para deletar cada nota, mas, para nosso aplicativo Sticky Notes, faremos algo muito mais simples: vamos apenas deletar a nota se o usuário clicar nela. Talvez não seja a melhor implementação em termos de usabilidade, mas é bem direto.

Para isso, primeiro precisamos mudar as stickies, de forma que possamos detectar *quando* uma sticky for clicada e, depois, a passaremos para a função `deleteSticky`.

A maior parte disso precisa acontecer na função `addStickyToDOM`. Vejamos como:

Quadro geral: vamos usar a chave da nota, que, lembre-se, é "sticky_" + o tempo para unicamente identificar a nota. Passaremos essa chave sempre que chamarmos `addStickyToDOM`.

```
function addStickyToDOM(key, value) {  
    var stickies = document.getElementById("stickies");  
    var sticky = document.createElement("li");  
    sticky.setAttribute("id", key); ←  
    var span = document.createElement("span");  
    span.setAttribute("class", "sticky");  
    span.innerHTML = stickyObj.value;  
    sticky.appendChild(span);  
    stickies.appendChild(sticky);  
    sticky.onclick = deleteSticky; ←  
}  
}
```

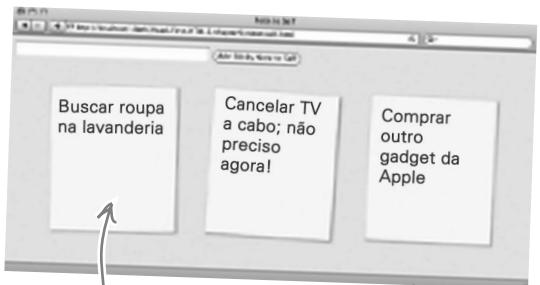
Estamos adicionando uma id única ao elemento `` que representa a nota no DOM. Faremos isso para que `deleteSticky` saiba qual nota foi clicada. Pelo fato de já sabermos que a chave da sticky é única, apenas a usaremos como a id.

Também estamos adicionando um click handler em cada nota. Quando clicar numa nota, a `deleteSticky` será chamada.



Seu trabalho agora é atualizar todo o código para que, de qualquer lugar que chamemos `addStickyToDOM`, passemos a chave, assim como o valor. Você deverá ser capaz de facilmente encontrar esses lugares. Depois de terminar, confira as respostas no fim do capítulo para garantir.

Não pule essa parte ou o test drive que está por vir não funcionará!



Como fazer a nota ser deletada a partir do evento

Temos agora um event handler em cada nota, ouvindo os cliques. Quando clicar numa nota, `deleteSticky` será chamada e um objeto event será passado à `deleteSticky` com informação sobre o evento, como qual elemento foi clicado. Podemos ver o `event.target` para dizer qual nota foi clicada. Vamos dar uma olhada mais de perto no que acontece quando você clica num sticky note.

```

<li id="sticky_1304270008375">
  <span class="sticky">Pick up dry cleaning</span>
</li>

```

← Este é o HTML para a nota que criamos no `addStickyToDOM`.

De qualquer forma, o evento gerado pelo clique passa dentro do `deleteSticky`.

```

function deleteSticky(e) {
  var key = e.target.id;
  if (e.target.tagName.toLowerCase() == "span") {
    key = e.target.parentNode.id;
  }
  localStorage.removeItem(key);
  var stickiesArray = getStickiesArray();
  if (stickiesArray) {
    for (var i = 0; i < stickiesArray.length; i++) {
      if (key == stickiesArray[i]) {
        stickiesArray.splice(i, 1);
      }
    }
    localStorage.setItem("stickiesArray", JSON.stringify(stickiesArray));
    removeStickyFromDOM(key);
  }
}

```

O target é o elemento que você clicou que gerou o evento, e podemos obter a id daquele elemento a partir da propriedade `target`. Se `target` é ``, estará certo.

Se o target é o ``, então precisamos obter a id do elemento parente, o ``. O `` é o elemento com a id que é a chave que precisamos.

Agora podemos usar a chave para removermos o item do `localStorage` e do `stickiesArray`.

Também precisamos remover o `` que está segurando a nota da página, assim desaparecerá quando clicada. Faremos isso em seguida...

Delete a nota do DOM, também

Para terminar de deletar, precisamos implementar a função `removeStickyFromDOM`. Você atualizou a função `addStickyToDOM` antes, para poder adicionar a chave da nota como a id do elemento `` prendendo a nota no DOM, para podermos usar `document.getElementById` para encontrar a nota no DOM. Pegamos o nó pai da nota e usamos o método `removeChild` para deletar a sticky:

Passe a chave (também a id) do elemento sticky que estamos procurando.

```
function removeStickyFromDOM(key) {  
  var sticky = document.getElementById(key);  
  sticky.parentNode.removeChild(sticky);  
}
```

Pegamos o elemento `` do DOM...

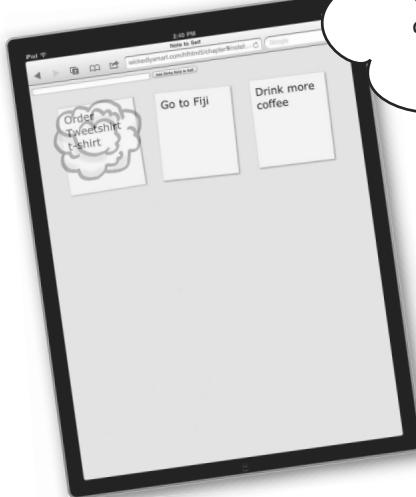
... e o removemos,

Pegando seu `parentNode` antes e então usando o `removeChild` para removê-lo.

Ok, teste... 

Pegue o código, carregue a página, adicione e delete algumas notas. Feche seu navegador, carregue novamente e dê uma bela testada!

Podemos deletar notas agora!



Bom trabalho! Agora, pode me arranjar uma maneira de colorir minhas notas com código? Sabe, amarelo para urgente, azul para ideias, rosa para segundo plano, essas coisas...

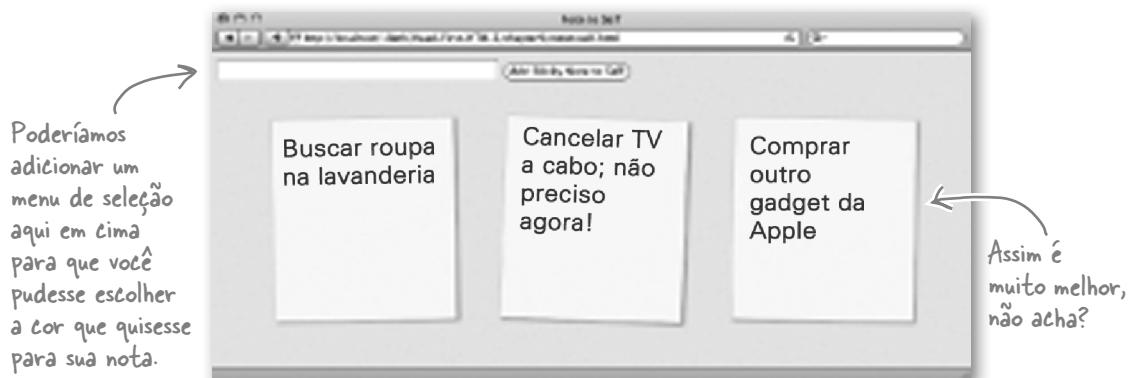


É claro que sim!

Vamos lá! Dado seu nível de experiência com tudo isso, vamos ser capazes de fazer isso tranquilamente. Como faremos? Bem, vamos criar um objeto para armazenar o texto da nota e sua cor e depois armazenar isso tudo como o valor do item sticky, usando `JSON.stringify` para converter isso numa string antes.

Atualize a interface do usuário para que possamos especificar uma cor

No momento, todas as nossas notas são amarelas. Não seria mais legal se pudéssemos ter uma gama maior de cores para as stickies?



Vamos resolver a parte mais fácil primeiro: atualizar o HTML para que tenhamos um menu de seleção de cores para ter de onde escolher. Edite seu arquivo `notetoself.html` e atualize seu formulário para adicionar as cores assim:

```
<html>
...
<form>
    <label for="note_color">Color: </label>
    <select id="note_color">
        <option value="LightGoldenRodYellow">yellow</option>
        <option value="PaleGreen">green</option>
        <option value="LightPink">pink</option>
        <option value="LightBlue">blue</option>
    </select>
    <label for="note_text">Text:</label> <input type="text" id="note_text">
    <input type="button" id="add_button" value="add Sticky Note to Self">
</form>
...
</html>
```

Estamos apenas mudando o formulário, o resto fica igual.

Acrecentaremos um rótulo para o texto das notas, assim o usuário saberá para que serve aquele campo.

Note a id do `<select>`; precisaremos disso para pegar o valor da opção selecionada no JavaScript.

Adicionamos quatro notas coloridas para escolher.

O valor de cada opção é o nome de uma cor que podemos colocar diretamente no estilo de nossas notas.

E o resto do formulário é o mesmo.

Vínhamos usando CSS para definir a cor padrão para as notas. Agora, queremos armazenar uma cor de nota com ela própria. Então, agora a questão é: como vamos armazenar a cor para a sticky note no localStorage?

JSON.stringify não é só para Arrays

Para armazenar a cor da nota com o texto da nota, podemos usar a mesma técnica que usamos para stickiesArray; armazenamos um objeto que contém o texto e a cor como valor para a nota no localStorage.

Color: pink Text: Cancel cable tv, who needs it now? Add Sticky Note to Self



Vamos pegar os valores que o usuário entra para a cor junto com o texto da nota e torná-los um simples objeto.

Assim como stickiesArray, teremos de chamar JSON.stringify no valor sticky antes de chamarmos localStorage.setItem para salvar o valor.

```
var stickyObj = {
  "value": "Cancel cable tv, who needs it now?",
  "color": "LightPink"
};
```

E vamos armazenar isso no localStorage com a chave sticky.

localStorage

Vamos reescrever a função createSticky para armazenar a cor com o texto da nota.

Para representar o texto e a cor, usaremos nosso objeto útil:

```
function createSticky() {
  var stickiesArray = getStickiesArray();
  var currentDate = new Date();
  var colorSelectObj = document.getElementById("note_color");
  var index = colorSelectObj.selectedIndex;
  var color = colorSelectObj[index].value;
  var key = "sticky_" + currentDate.getTime();
  var value = document.getElementById("note_text").value;
  var stickyObj = {
    "value": value,
    "color": color
  };
  localStorage.setItem(key, JSON.stringify(stickyObj));
  stickiesArray.push(key);
  localStorage.setItem("stickiesArray", JSON.stringify(stickiesArray));
  addStickyToDOM(key, stickyObj);
}
```

Faremos o usual para pegar o valor da opção de cor selecionada.

Então, usaremos aquela cor para criar stickyObj: um objeto que contém duas propriedades, o texto da nota e a cor que o usuário selecionou.

E chamaremos JSON.stringify no stickyObj antes de o colocarmos no localStorage.

Agora estamos passando o objeto em vez de uma string de texto para o addStickyToDOM. O que significa que você precisará atualizar addStickyToDOM, certo?

Usando o novo stickyObj

Agora que passamos o stickyObj para addStickyToDOM, precisamos atualizar a função para usar o objeto, em vez da string que estávamos passando anteriormente, e para definir a cor de fundo da nota. É uma mudança bem fácil, no entanto; vamos dar uma olhada:

```
function addStickyToDOM(key, stickyObj) {
    var stickies = document.getElementById("stickies");
    var sticky = document.createElement("li");
    sticky.setAttribute("id", key);
    sticky.style.backgroundColor = stickyObj.color;
    var span = document.createElement("span");
    span.setAttribute("class", "sticky");
    span.innerHTML = stickyObj.value;
    sticky.appendChild(span);
    stickies.appendChild(sticky);
    sticky.onclick = deleteSticky;
}
```

Precisamos mudar o parâmetro aqui para ser stickyObj em vez do valor de texto da nota.

Pegamos a cor do stickyObj que estamos passando ao addStickyToDOM.

Os objetos do elemento HTML têm uma propriedade de estilo que você pode usar para acessar o estilo daquele elemento.

Note que, quando definimos a propriedade da cor do background no JavaScript, a especificamos como backgroundColor, NÃO background-color, como no CSS.

Então, precisamos pegar o valor do texto que vamos usar na sticky note do objeto.

Há mais um lugar em que precisamos atualizar o código. É a função init, de onde estamos pegando as notas do localStorage e passando para addStickyToDOM, quando primeiro carregamos a página.

```
function init() {
    var button = document.getElementById("add_button");
    button.onclick = createSticky;

    var stickiesarray = getStickiesarray();

    for (var i = 0; i < stickiesarray.length; i++) {
        var key = stickiesarray[i];
        var value = JSON.parse(localStorage[key]);
        addStickyToDOM(key, value);
    }
}
```

Agora, quando pegarmos o valor da sticky note do localStorage, precisaremos do JSON.parse, pois é um objeto, não mais uma string.

E vamos passar aquele objeto para addStickyToDOM no lugar da string (o código parece o mesmo, mas o que passamos é diferente).

Test drive das cores das notas



Antes de rodar o aplicativo Note to Self novamente, você vai precisar limpar seu localStorage, pois a versão anterior de nossas stickies não tinha qualquer cor armazenada nela e agora estamos usando um formato diferente para nossos valores de notas. Antes, estávamos usando strings; agora, estamos usando objetos. Portanto, esvazie seu localStorage, recarregue a página e adicione alguns stickies, selecionando cores diferentes para cada um. Aqui estão nossos stickies (e vamos também dar uma olhada no localStorage):

Você pode ver seu arquivo maintenance.html para limpar seu localStorage, ou usar o console.

Escolhemos amarelo, rosa e azul para nossas notas quando fomos adicioná-las.

The screenshot shows a web browser window titled "Note to Self" with the URL "http://localhost/~Beth/Head-First-HTML5/chapter9/notetoself.html". The page displays three sticky notes:

- Color: yellow Text: Buscar roupa na lavanderia
- Color: pink Text: Cancelar TV a cabo; não preciso agora!
- Color: blue Text: Comprar outro gadget da Apple

Below the browser is the Chrome DevTools Resources panel. The localStorage section shows the following data:

| Key | Value |
|----------------------|--|
| sticky_1312150404612 | {"value": "Buy another Apple gadget", "color": "LightBlue"} |
| sticky_1312150374242 | {"value": "Pick up dry cleaning", "color": "LightGoldenRodYellow"} |
| stickiesArray | ["sticky_1312150374242", "sticky_1312150386414", "sticky_1112150404612"] |
| sticky_1312150386414 | {"value": "Cancel cable tv, who needs it now", "color": "LightPink"} |

Cada valor de sticky note é agora um objeto (JSON.stringify), contendo o valor de texto da sticky e a cor da sticky.

Estive pensando... Se podemos armazenar objetos e arrays, por que não simplesmente armazenar todas as notas no próprio array? Por que precisamos de todos esses outros itens? Parece que isso torna complicado, quando poderíamos integrar tudo num só item no localStorage.



Para alguns usuários, isso faz muito sentido.

Sabendo o que já sabemos, certamente poderíamos desenhar as notas de forma a serem objetos integrados num array. Se for além, você poderá decidir fazer exatamente isso. Pode também fazer sentido para seu carrinho de compras. O único ponto negativo é que os métodos `JSON.stringify` e `JSON.parse` exigem muito mais trabalho, sempre que se quer fazer alguma modificação. Por exemplo: para adicionar uma nota, temos de analisar o conjunto inteiro de notas, adicionar a nota e então passar o `stringify` por todas as notas novamente antes de escrevê-las de volta no armazenamento. De qualquer maneira, para a quantidade de dados no Stickies Note, isso não deverá ser um problema em geral (embora é preciso pensar em dispositivos móveis com CPUs limitados e o efeito do uso do CPU na vida da bateria).

Portanto, se quiser juntar tudo num objeto ou array no `localStorage`, realmente depende de quantos itens de dados você precisa armazenar, qual o tamanho de cada um e qual o tipo de processamento que você fará neles.

A nossa implementação pode ser um pouco sacrificante para o número limitado de Stickies, porém esperamos que você concorde que isso nos deu uma ótima oportunidade para pensar na API `localStorage` e como lidar com itens nela.

~~NÃO~~ TENTE ISTO EM CASA (OU JOGARÁ PELOS ARES SEUS 5 MEGABYTES)

Dissemos a você que temos cinco megabytes inteiros de armazenamento em cada navegador de usuário, mas mesmo que cinco megabytes pareça ser muito, lembre-se de que todos os seus dados são armazenados no formulário de uma string, em vez de num formato de dados com eficiência de bytes. Pegue um número grande qualquer, digamos, a dívida pública — quando expressada na forma de decimais, utiliza muitas vezes mais aquela quantidade de memória. Então, dito isto, os cinco megabytes podem não ser tanto quanto você imagina.

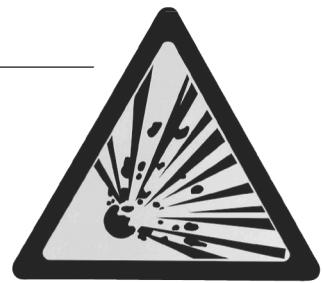
Então, o que acontece quando você usa todos os 5MB? Bem, infelizmente, esse é um daqueles comportamentos que não estão bem definidos pelas especificações do HTML5 e os navegadores podem fazer diversas coisas quando você excede seu limite — o navegador pode perguntar se quer permitir mais armazenamento ou simplesmente lançar uma exceção QUOTA_EXCEEDED_ERR, que você pode resolver assim:

```
Um try/catch captura quaisquer exceções que são lançadas dentro do try block.  
↓  
try {  
    localStorage.setItem(myKey, myValue);  
} catch(e) {  
    if (e == QUOTA_EXCEEDED_ERR) {  
        alert("Out of storage!");  
    }  
}
```

Aqui está uma chamada `setItem` no meio do `try block`; se alguma coisa der errado e o `setItem` lançar uma exceção, o `catch block` será invocado.

Essa é uma área do JavaScript que não vimos, talvez queira adicioná-la a sua lista de coisas para pesquisar.

Nem todos os navegadores estão lançando atualmente a exceção `QUOTA_EXCEEDED_ERR`, mas eles ainda lançam a exceção quando você excede seu limite; então, talvez você queira lidar com o caso geral de uma exceção ocorrendo quando você define um item.



Não vemos qualquer razão para não exigir de seu navegador até o limite, ver do que ele é feito, até onde ele pode ir, ver qual seu comportamento sob pressão. Vamos escrever um pouco de código para forçar seu navegador até seu limite de armazenamento:

```
<html>
<head>
<script>

localStorage.setItem("fuse", "-");
while(true) {
    var fuse = localStorage.getItem("fuse");
    try {
        localStorage.setItem("fuse", fuse + fuse);
    } catch(e) {
        alert("Your navegador blew up at " + fuse.length + " with exception: " + e);
        break;
    }
}
localStorage.removeItem("fuse");
</script>
</head>
<body>
</body>
</html>
```

Vamos começar com uma string de um caractere, com a chave "fuse".

E ir aumentando seu tamanho...

... dobrando a string (concatenando-a com ela mesma).

Então, tentaremos escrevê-la de volta ao localStorage.

Se explodir, acabamos por aqui! Alertaremos o usuário e sairemos desse loop.

E não vamos deixar uma bagunça, então remova o item do localStorage.

Vá em frente e digite isto, diminua o fuse carregando-o e divirta-se! Tente isso em alguns navegadores diferentes.

.....
.....
.....

Se tiver coragem de rodar isso, ponha seus resultados aqui.



Veja bem!

Use por sua conta e risco!

Sério. Este código pode levar seu navegador ao colapso, o que pode levar seu sistema operacional a um estado de infelicidade, o que pode levar você a perder seu trabalho. Use por sua conta e risco!!!



Estive fazendo um teste beta no meu aplicativo de carrinho de compras e os usuários não querem seus carrinhos colados neles pelo navegador. Como posso remover todos os itens do carrinho de compras, quando o usuário fechar o navegador? Escolhi a tecnologia errada?

Não, Luke. Há um outro Skywalker.

Acontece que o localStorage tem uma irmã, chamada sessionStorage. Se você substituir a variável global sessionStorage em todo lugar que usou localStorage, seus itens serão armazenados somente durante a sessão do navegador. Portanto, assim que aquela sessão tiver terminado (em outras palavras, o usuário fechar a janela do navegador), os itens armazenados serão removidos.

O objeto sessionStorage suporta exatamente a mesma API que localStorage. Sendo assim, você já sabe tudo o que precisa.

Experimente!

QUEM FAZ O QUÊ?

A esta altura, você já conhece a API localStorage. Abaixo, você encontrará todos os principais personagens da API sentados com suas máscaras. Veja se pode determinar o que faz o quê. Fomos adiante e fizemos um por você para incentivá-lo a começar.

clear

Use-me para armazenar itens a longo prazo.

sessionStorage

Pego chaves e valores e os escrevo dentro do localStorage. Agora, tenha em mente que, se já houver um item com aquela chave no localStorage, não vou avisá-lo. Simplesmente, sobrescreverei; portanto, é melhor que saiba o que está pedindo.

key

Se você se delongar demais no localStorage e usar muito espaço, receberá uma exceção e ouvirá falar da gente.

setItem

Precisa jogar fora um item? Farei esse trabalho discretamente.

removeItem

Apenas dê-me uma chave e sairei em busca de um item com ela e lhe darei o valor dele.

length

Sou do tipo curto prazo e armazenarei suas coisas somente enquanto seu navegador estiver aberto. Feche seu navegador e puff, todas as coisas se foram.

getItem

Quando estiver farto de todos os itens em seu localStorage, limpo todos eles e jogo-os fora, deixando-o com um fresco e vazio localStorage (tenha em mente que posso apenas limpar minha própria origem).

localStorage

Precisa saber quantos itens estão em seu localStorage? Estou aqui.

QUOTA_EXCEEDED_ERR

Dê-me um index e lhe darei uma chave dele no localStorage.

Agora que você conhece o localStorage, como irá usá-lo?

Há muitas maneiras de se usar o localStorage — o aplicativo Stickies usou-o, portanto, não precisamos de um servidor; mas, mesmo com um servidor, o localStorage pode ser bem útil. Veja algumas maneiras que os desenvolvedores têm usado:

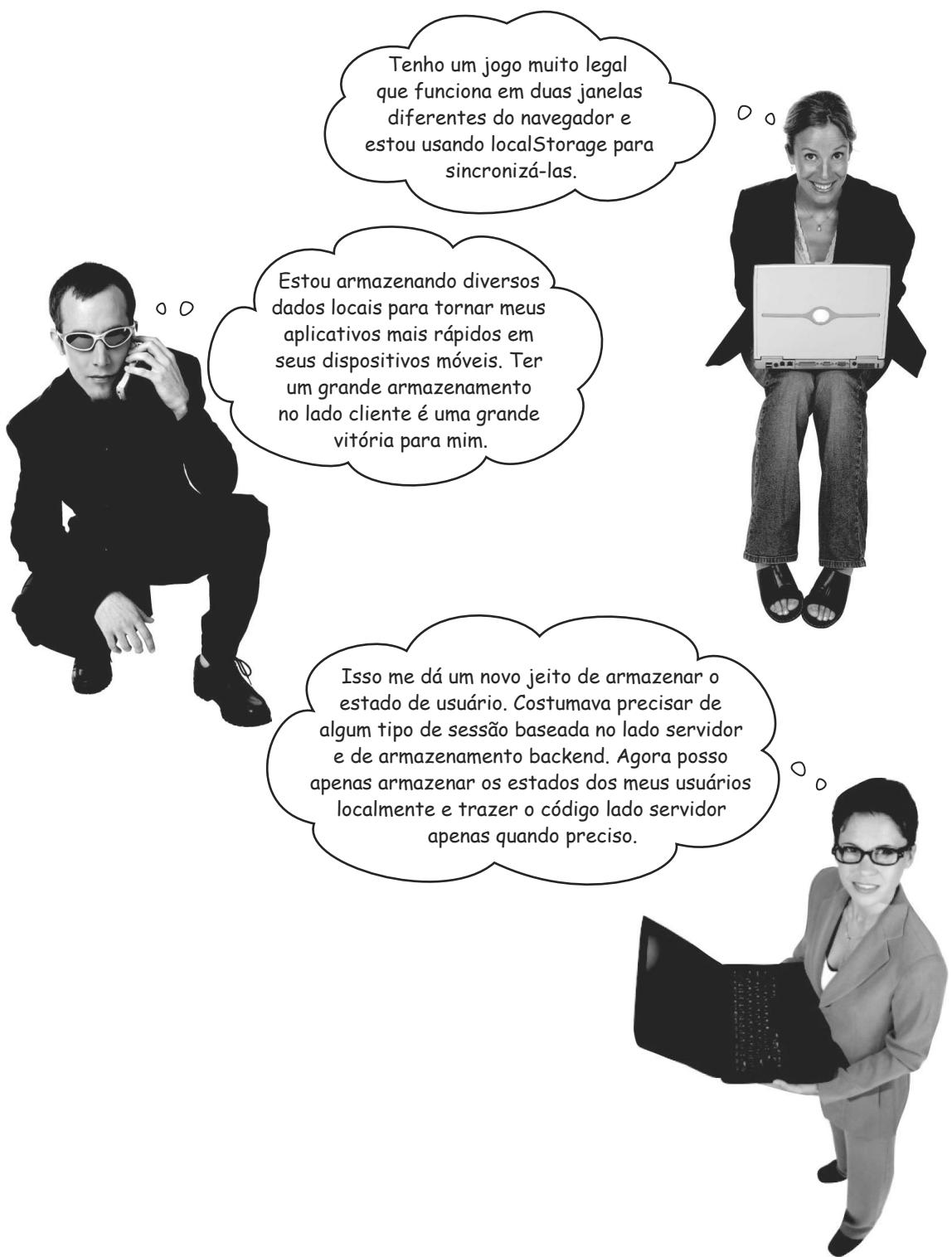


No meu novo cliente Twitter, vou armazenar os resultados das buscas do Twitter para ter eficiência com o localStorage. Quando meus usuários buscam, checo os resultados locais primeiro. Isso poderá ajudar muito meus usuários móveis.

Vou armazenar listas de reprodução com metadata para meus usuários. Eles serão capazes de armazenar seus clipes favoritos junto com o timecode de onde eles deixaram de ver.



Estou usando sessionStorage para o carrinho de compras de minha biblioteca e-commerce. Se o usuário fechar o navegador, quero que o carrinho suma.





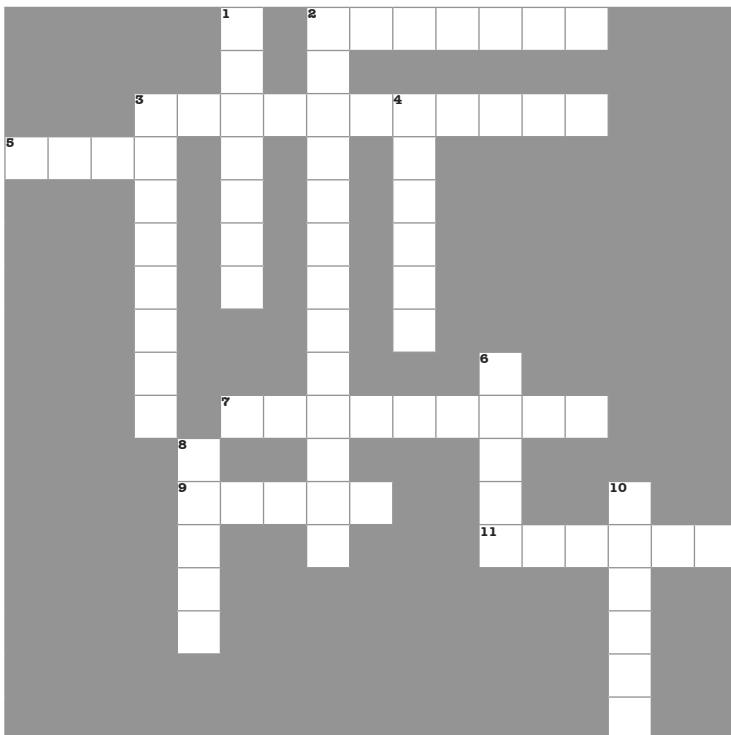
PONTOS DE BALA

- Web Storage é um armazenamento em seu navegador e uma API que pode ser usada para salvar e resgatar itens do armazenamento.
- A maioria dos navegadores fornece pelo menos 5 megabytes de armazenamento por origem.
- O Web Storage consiste em armazenamento local e armazenamento em sessão.
- O armazenamento local é persistente, mesmo que feche sua janela do navegador ou feche o navegador.
- Itens no sessionStorage são removidos quando você fecha suas janelas do navegador ou o fecha. O sessionStorage é bom para itens temporários, sem armazenamento de longo prazo.
- Tanto o localStorage quanto o sessionStorage usam exatamente a mesma API.
- O Web Storage é organizado pela origem (pense em domínio). Uma origem é a localização do documento na web (p.e., wickedlysmart.com ou headfirstlabs.com).
- Cada domínio possui um armazenamento separado. Itens armazenados numa origem não são visíveis para páginas em outras origens.
- Use localStorage.setItem (chave) para adicionar um valor para o armazenamento.
- Use o localStorage.getItem (chave) para resgatar um valor do armazenamento.
- Você pode usar a mesma sintaxe como arrays associativos para definir e resgatar itens para o armazenamento. Use o localStorage[chave] para isso.
- Use o método localStorage.key() para enumerar as chaves no localStorage.
- localStorage.length é o número de itens no localStorage em uma dada origem.
- Use o console em seu navegador para ver e deletar itens no localStorage.
- Você pode deletar itens diretamente do localStorage ao clicar com o botão direito num item e escolher deletá-lo (nota: talvez não funcione em todos os navegadores).
- Você pode deletar itens do localStorage no código, usando o método removeItem (chave) e o método clear. Note que o método clear deleta tudo no localStorage da origem onde fizer a limpeza.
- As chaves para cada item do localStorage devem ser únicas. Se usar a mesma chave de um item existente, você sobreporá o valor daquele item.
- Uma maneira de gerar uma chave única é utilizar o tempo atual em milissegundos desde 1970, usando o método getTime() do objeto Date.
- É importante criar um esquema de nomeação para seu aplicativo, que ainda funcionará se itens forem removidos de seu armazenamento ou se outro aplicativo criar itens no armazenamento.
- O Web Storage atualmente suporta armazenar strings como valores para chaves.
- Você pode converter números armazenados em localStorage como strings de novo em números, usando parseInt ou parseFloat.
- Se precisar armazenar mais dados complexos, você poderá usar objetos JavaScript e convertê-los em strings, antes de armazenar usando JSON.stringify, e de volta a objetos depois de resgatá-los usando JSON.parse.
- O armazenamento local pode ser particularmente útil em dispositivos móveis para reduzir requerimentos de transferência de dados.
- O sessionStorage é como um armazenamento local, exceto pelo fato de que o que estiver salvo no armazenamento do navegador não persiste se a aba, a janela, ou o próprio navegador forem fechados. O sessionStorage é útil para armazenamentos de curto prazo, assim como para sessões de compras.



Cruzada HTML5

Aproveite um tempinho para testar seu próprio armazenamento local.



HORIZONTAIS

2. Armazenamos um item no localStorage com este método.
3. O sessionStorage é quase como o localStorage, exceto que não é _____ se fechar sua janela do browser.
5. A irmã do Luke Skywalker.
7. O cookie tem um(a) problema de _____.
9. Pensamos que seria somente uma fantasia armazenar um(a) _____ no localStorage, mas no fim das contas podemos, com o JSON.
11. Criamos um(a) _____ para armazenar o texto da nota e sua cor num item localStorage.

VERTICIAIS

1. O localStorage pode armazenar somente _____.
2. Usamos o _____ para manter as chaves de todas nossas stickies, assim poderíamos encontrá-las facilmente no localStorage.
3. Use _____ para converter uma string a um número inteiro.
4. Podemos detectar sobre quais notas o usuário clicou olhando o evento _____.
6. Se armazenar algo em seu browser e for a _____, ainda estará lá quando você voltar.
8. Use um try/catch para detectar quota-exceeded errors no localStorage.
10. Quando usamos o(a) _____ do localStorage para criar nomes chave, vamos em direção a um problema: lacunas nos nomes de nossas notas.



Jogo da Concha – Solução

Pronto para tentar a sorte? Ou devo dizer habilidade? Temos um jogo para você testar seu comando de localStorage, mas precisará estar atento. Use seu conhecimento de obter e definir pares de chave/valor no localStorage para rastrear a ervilha, à medida que ela muda de concha em concha. Veja nossa solução:

```
function shellGame() {  
    localStorage.setItem("shell1", "pea");  
    localStorage.setItem("shell2", "empty");  
    localStorage.setItem("shell3", "empty");  
    localStorage["shell1"] = "empty";  
    localStorage["shell2"] = "pea";  
    localStorage["shell3"] = "empty";  
    var value = localStorage.getItem("shell2");  
    localStorage.setItem("shell1", value);  
    value = localStorage.getItem("shell3");  
    localStorage["shell2"] = value;  
    var key = "shell2";  
    localStorage[key] = "pea";  
    key = "shell1";  
    localStorage[key] = "empty";  
    key = "shell3";  
    localStorage[key] = "empty";  
  
    for (var i = 0; i < localStorage.length; i++) {  
        var key = localStorage.key(i);  
        var value = localStorage.getItem(key);  
        alert(key + ": " + value);  
    }  
}
```

Qual concha esconde a ervilha?

| Chave | Valor |
|----------|---------|
| concha 1 | vazia |
| concha 2 | ervilha |
| concha 3 | vazia |

A ervilha está sob a concha 2.



Seu trabalho agora é atualizar todo o código para que, de qualquer lugar que chamemos addStickyToDOM, passemos a chave, assim como o valor.

Você deverá atualizar todas as chamadas a addStickyToDOM na init e createSticky para ficar assim:

```
addStickyToDOM(key, value);
```



Aponte o seu lápis Solução

Marque a seguir as maneiras com que nossa implementação atual pode causar problemas:

- Mostrar as notas é ineficiente se houver vários itens em localStorage que não são stickies.
- Uma nota poderá ser sobreescrita por setItem, se o tamanho do localStorage ficar menor, quando outro aplicativo deletar seus próprios itens.
- É difícil dizer rapidamente quantas notas existem; você terá de iterar cada item no localStorage para conseguir todas as stickies.
- Use um cookie. Deve ser mais fácil do que tudo isso!



Exercício Solução

Ainda precisamos descobrir como armazenar efetivamente um array no localStorage.

Você já deve ter imaginado que podemos usar JSON para criar uma representação de string de um array e, se assim o fez, estava certo. Uma vez que a tenha, poderá armazenar no localStorage.

Lembre-se de que existem apenas dois métodos na API JSON: stringify e parse. Vamos pôr esses métodos para funcionar, finalizando a função init:

```
function init() {
    // button code here...
    var stickiesarray = localStorage["stickiesarray"];
    if (!stickiesarray) {
        stickiesarray = [];
        localStorage.setItem("stickiesarray", JSON.stringify(stickiesarray));
    } else {
        stickiesarray = JSON.parse(stickiesarray);
    }
    for (var i = 0; i < stickiesarray.length; i++) {
        var key = stickiesarray[i];
        var value = localStorage[key];
        addStickyToDOM(value);
    }
}
```

Pegue o array do localStorage. Se não houver um array no localStorage, então criamos um array vazio e designamo-lo a uma variável stickiesArray. Nesse momento, a variável stickiesArray é uma string.

Se tivéssemos que criar um novo array, usariammos o JSON.stringify para criar uma representação string do array, e então o armazenaríamos.

Se o stickiesArray já estiver armazenado no localStorage (como uma string), então precisaremos analisá-lo com JSON. Depois disso, teremos um array de chaves designado para a variável stickiesArray.

Só para esclarecer, estamos levando a string à direção stickiesArray, analisando-a dentro de um array e, então, designando aquele array de volta à variável stickiesArray.

NÃO TENTE ISTO EM CASA (OU JOGARÁ PELOS ARES SEUS 5 MEGABYTES)

Dissemos a você que temos cinco megabytes inteiros de armazenamento em cada navegador de usuário, mas mesmo que cinco megabytes pareça ser muito, lembre-se de que todos os seus dados são armazenados no formulário de uma string, em vez de num formato de dados com eficiência de bytes. Pegue um número grande qualquer, digamos, a dívida pública — quando expressada na forma de decimais, utiliza muitas vezes mais aquela quantidade de memória. Então, os cinco megabytes podem não ser tanto quanto você imagina.

Então, o que acontece quando você usa todos os 5MB? Bem, infelizmente, esse é um daqueles comportamentos que não estão bem definidos pelas especificações do HTML5 e os navegadores podem fazer diversas coisas quando você excede seu limite — o navegador pode perguntar se quer permitir mais armazenamento ou simplesmente lançar uma exceção QUOTA_EXCEEDED_ERR, que você pode resolver assim:

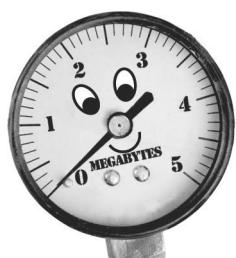
Um try/catch captura quaisquer exceções que são lançadas dentro do try block.

```
try {  
    localStorage.setItem(myKey, myValue);  
} catch(e) {  
    if (e == QUOTA_EXCEEDED_ERR) {  
        alert("Out of storage!");  
    }  
}
```

Aqui está uma chamadasetItem no meio do try block; se alguma coisa der errado e o setItem lançar uma exceção, o catch block será invocado.

Estamos testando para ver se este é um storage quota error (contrário a alguns tipos de exceção). Se assim for, alertaremos o usuário. Você normalmente vai querer fazer algo mais significativo do que apenas alertar.

Nem todos os navegadores estão lançando atualmente a exceção QUOTA_EXCEEDED_ERR, mas eles ainda lançam a exceção quando você excede seu limite, então talvez você queira lidar com o caso geral de uma exceção ocorrendo quando define um item.



Não vemos qualquer razão para não exigir de seu navegador até o limite, ver do que ele é feito, até onde ele pode ir, ver qual seu comportamento sob pressão. Vamos escrever um pouco de código para forçar seu navegador até seu limite de armazenamento:



```

<html>
<head>
<script>

localStorage.setItem("fuse", "-");
while(true) {
    var fuse = localStorage.getItem("fuse");
    try {
        localStorage.setItem("fuse", fuse + fuse);
    } catch(e) {
        alert("Your navegador blew up at" + fuse.length + " with
exception: " + e);
        break;
    }
}
localStorage.removeItem("fuse");
</script>
</head>
<body>
</body>
</html>

```

Vamos começar com uma string de um caractere, com a chave "fuse".

E ir aumentando seu tamanho...

... dobrando a string (concatenando-a com ela mesma).

Então, tentaremos escrevê-la de volta ao localStorage.

E não vamos deixar uma bagunça, então remova o item do localStorage.

Se explodir, acabamos por aqui! Alertaremos o usuário e sairemos desse loop.

Vá em frente, digite isto, diminua o fuse carregando-o e divirta-se! Tente isso em alguns navegadores diferentes.

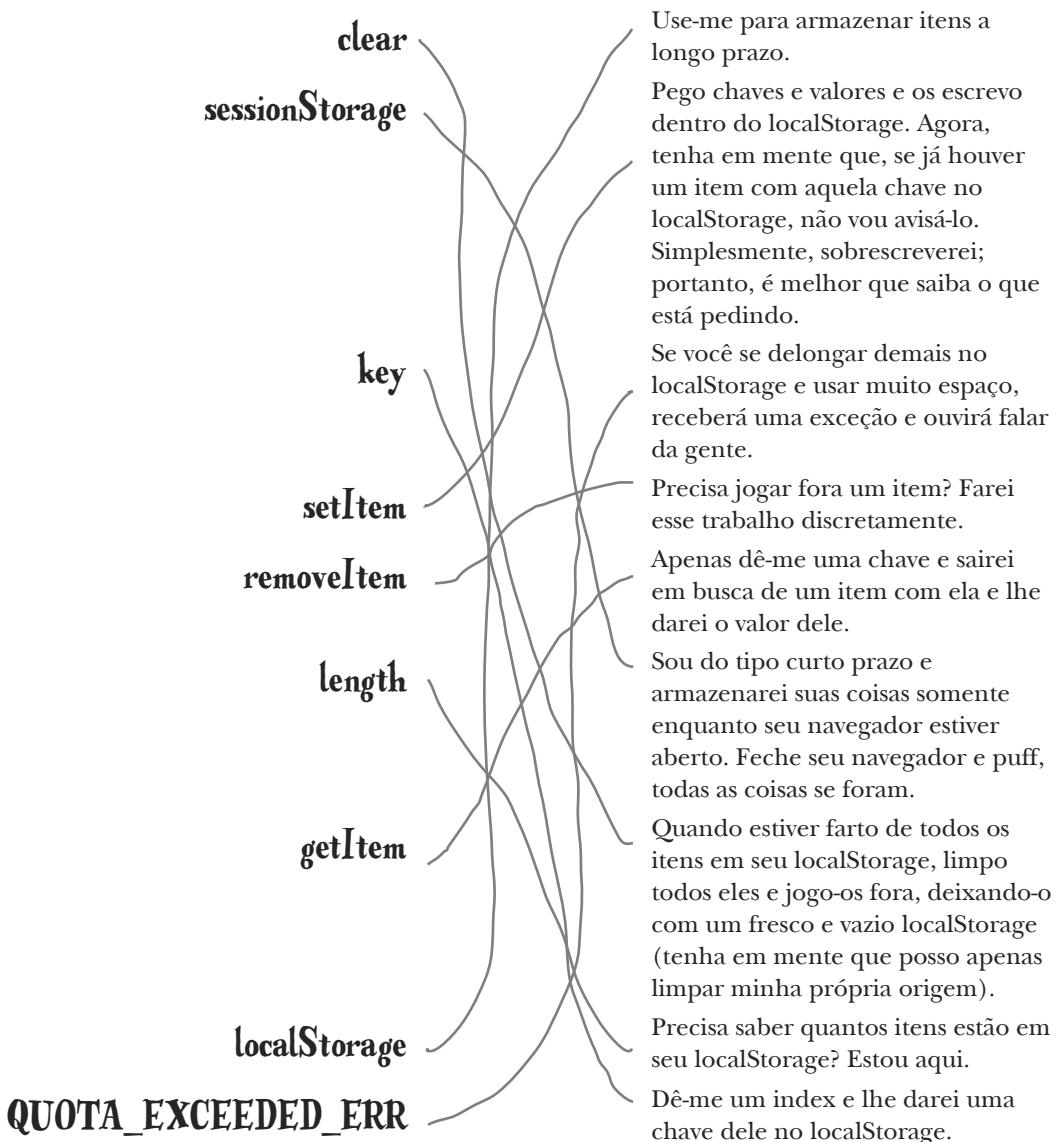
Estes são os nossos resultados do Safari e do Chrome.



QUEM FAZ O QUÊ?

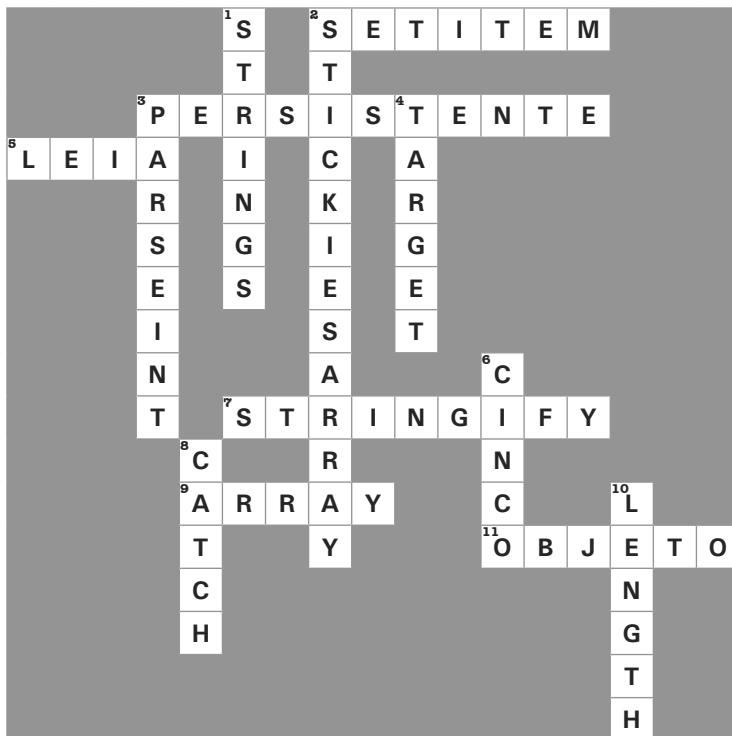
SOLUÇÃO

A esta altura, você já conhece a API localStorage. Abaixo, você encontrará todos os principais personagens da API sentados com suas máscaras. Veja se pode determinar o que faz o quê. Fomos adiante e fizemos um por você para incentivá-lo a começar.





Cruzada HTML5 – Solução



10 Pondo o javascript para funcionar

Web Workers (os Trabalhadores da Web)



Script lento — você quer continuar executando-o? Se você já gastou tempo suficiente com o JavaScript ou pesquisando na web, provavelmente já viu a mensagem “script lento” (slow script). Com todos esses processadores multi núcleo fazendo parte da sua máquina novinha, como pode um script estar rodando *tão lentamente*? Isso acontece porque o JavaScript só pode fazer uma coisa por vez. Contudo, com o HTML5 e os Web Workers, *tudo isso muda*. Você agora tem a habilidade de gerar *seus próprios* trabalhadores JavaScript para obter maior volume de trabalho feito. Se estiver apenas tentando desenhar um aplicativo mais responsivo, ou só quiser exigir o máximo que o CPU de sua máquina pode trabalhar, os Web Workers estão aqui para ajudar. Ponha seu chapéu de gerente JavaScript e vamos pôr alguns peões para trabalhar!

O terrível script lento

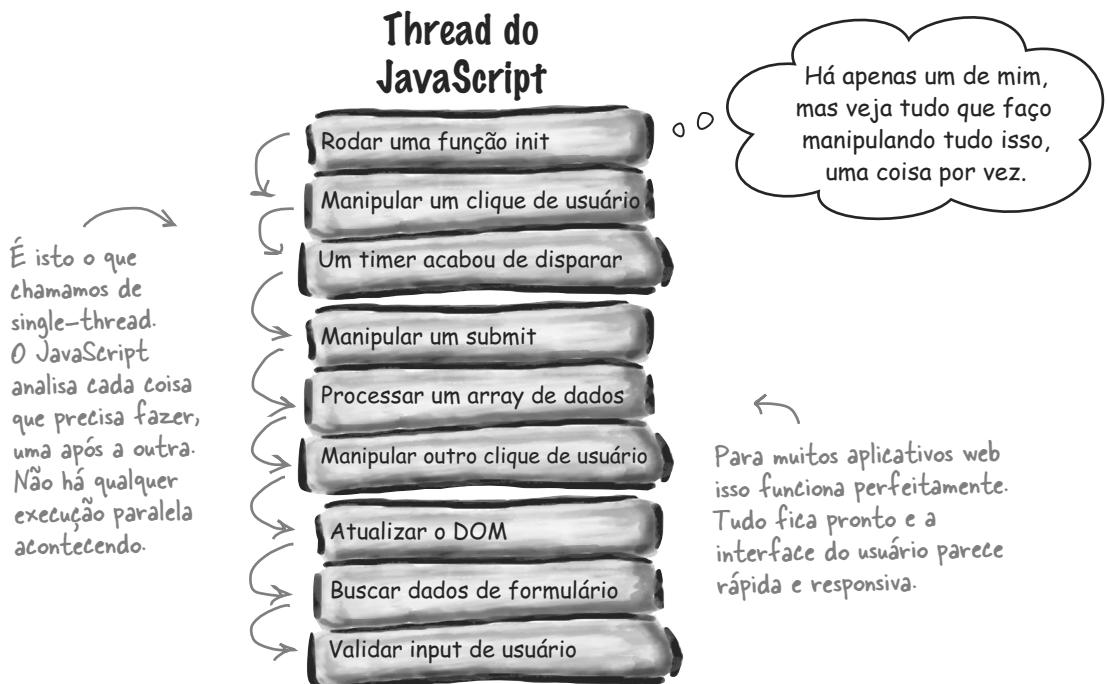
Uma das coisas mais incríveis sobre o JavaScript é que ele faz só uma coisa por vez. É o que gostamos de chamar de “single-thread”. Por que isso é incrível? Porque faz a programação de forma direta. Quando você tem vários threads de execução acontecendo ao mesmo tempo, escrever um programa que funcione corretamente pode se tornar um desafio e tanto.

O lado negativo de ser single-thread é que, se você der muita coisa para fazer a um programa JavaScript, ele pode ficar sobrecarregado e acabamos tendo diálogos “slow script”. A outra consequência de ter apenas um thread é que, se tiver um código JavaScript que estiver trabalhando muito, ele não deixa muita força computacional para a interface de seu usuário ou para as interações do usuário, e seu aplicativo pode parecer meio molenga ou não-responsivo.



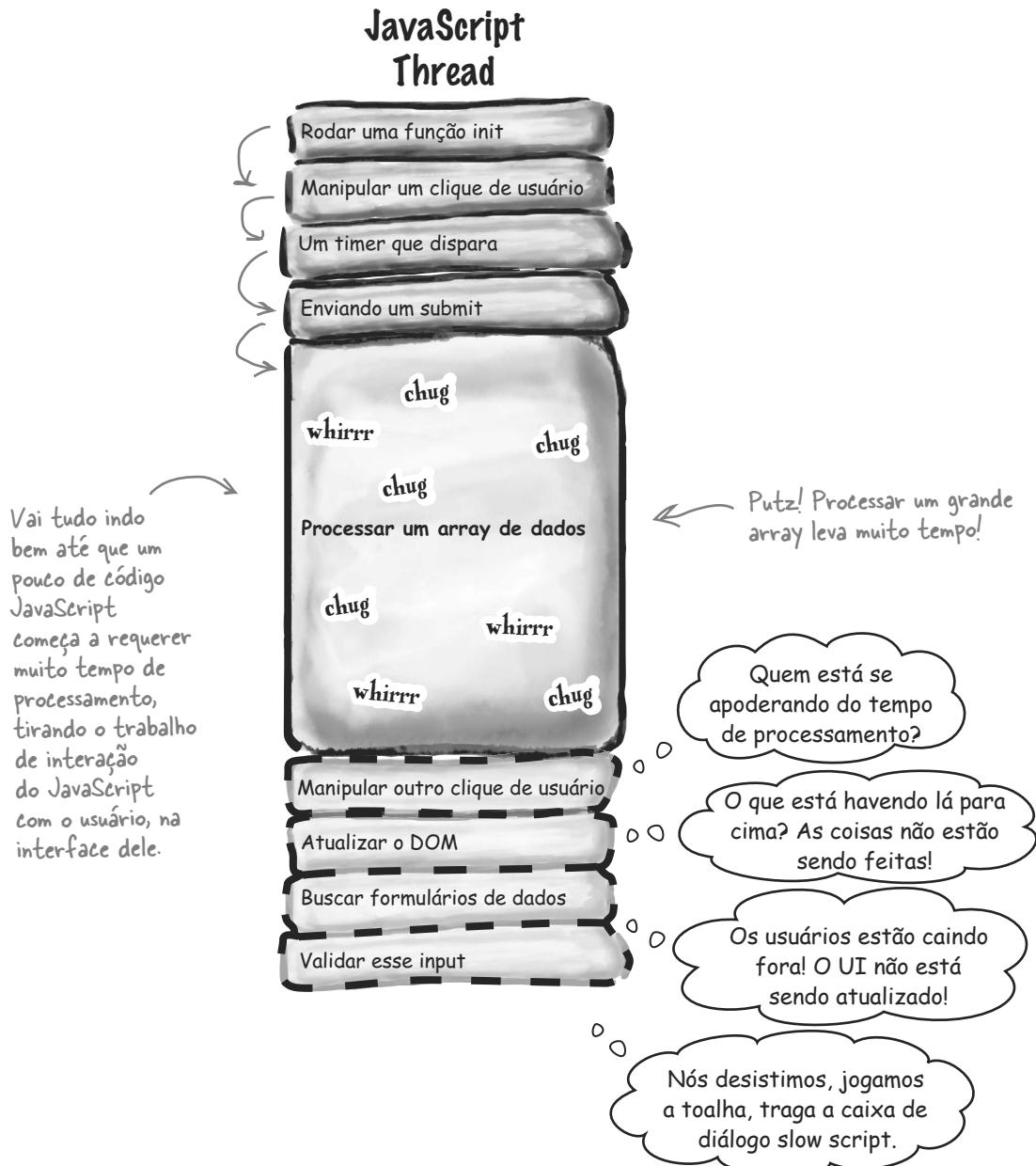
Como o JavaScript gasta seu tempo

Vejamos o que tudo isso significa, dando uma olhada em como o JavaScript manipula as tarefas de uma página típica:



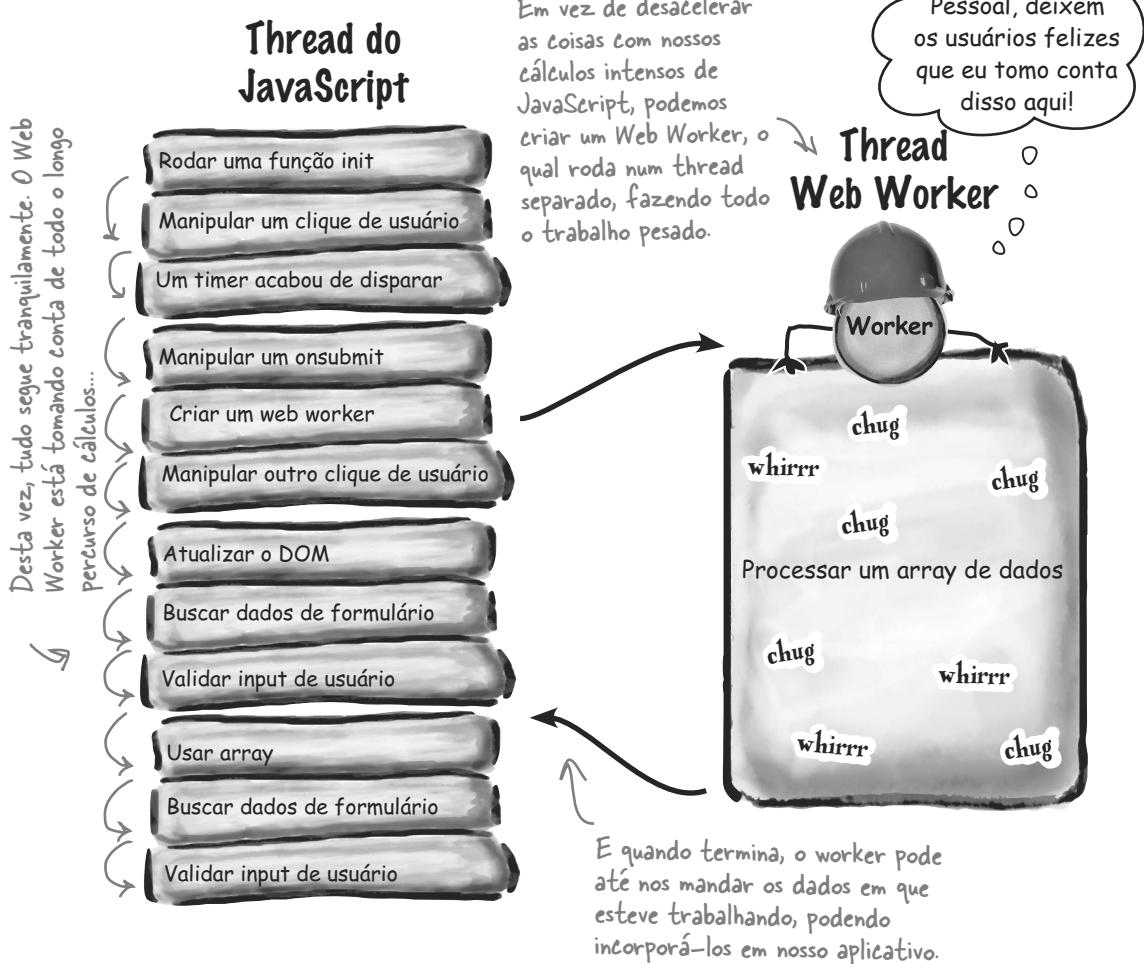
Quando o single-thread fica RUIM

É verdade que, para muitos usos, este modo single-thread de computar do JavaScript funciona perfeitamente e, como dissemos, torna a programação bem direta. Quando se escreve código, porém isso se torna tão “computacionalmente intensivo” que começa a criar impacto na habilidade do JavaScript de lidar com tanto trabalho; o modelo single-thread começa a entrar em colapso.



Adicionando outro thread de controle para ajudar

Antes de HTML5, ficávamos presos a apenas um thread de controle em nossas páginas e aplicativos, mas agora, com os Web Workers, encontramos uma maneira de criar outro thread de controle para nos ajudar. Então, se você possui código que leva muito tempo para ser calculado, crie um Web Worker que irá manipular aquela tarefa enquanto o thread de controle principal do JavaScript estiver se certificando de que está tudo bem com o navegador e o usuário.



Fizemos um escarcéu com o fato de que um thread de controle mantém as coisas simples e fáceis de serem programadas, o que não deixa de ser verdade. Como verá, porém, os Web Workers têm trabalhado cuidadosamente para garantir que tudo permaneça simples, fácil e seguro para os programadores. Veremos em um instante...



JavaScript Exposto (De Novo)

Entrevista da semana:

Onde o JavaScript passa seu tempo

Use a Cabeça!: Bem-vindo de volta, JavaScript. Bom vê-lo novamente.

JavaScript: Fico feliz de estar aqui, contanto que nos atenhamos à minha agenda, pois tenho muito o que fazer.

Use a Cabeça!: É exatamente onde pensei que iríamos focar nosso tempo hoje. Você é um cara de muito sucesso e faz tanta coisa ao mesmo tempo... Como consegue concluir tudo?

JavaScript: Bem, tenho uma filosofia: faço uma coisa por vez e faço isso muito bem.

Use a Cabeça!: Como consegue fazer uma coisa de cada vez? Para nós, parece que você está resgatando dados, mostrando páginas, interagindo com o usuário, gerenciando timers e alertas e assim por diante...

JavaScript: Sim, eu faço tudo isso, mas o que quer que esteja fazendo, faço só aquilo. Então, se eu estiver lidando com o usuário, só vou fazer isso até terminar.

Use a Cabeça!: Como pode ser verdade? E se um timer dispara, ou chegam dados de rede, ou qualquer outra coisa, você não para e vai fazer a outra coisa?

JavaScript: Quando um evento ocorre, como os que você mencionou, ele é adicionado a uma fila. Nem chego a olhar para ele até que tenha terminado a tarefa na qual estou trabalhando. Dessa forma, faço tudo corretamente e de maneira segura e eficiente.

Use a Cabeça!: Você está sempre atrasado para mexer com uma daquelas tarefas na fila?

JavaScript: Ah, isso acontece. Felizmente, eu sou a tecnologia por detrás das páginas do navegador. Então, não deve ser tão ruim se eu me atrasar um pouco, né? Você deveria falar com os caras que têm de rodar código para propulsores de naves espaciais ou controladores de usinas de energia nuclear... Esses, sim, têm de viver sob diferentes circunstâncias — por isso que eles ganham tanta grana.

Use a Cabeça!: Sempre me perguntei o que estava acontecendo quando obtinha a mensagem na caixa de diálogo “Slow script, deseja continuar?” no navegador. É você tirando uma folguinha?

JavaScript: Tirando uma folguinha!? Há... Isso é quando alguém acabou estruturando sua página de maneira a me deixar tão atarefado que acabo não conseguindo dar conta de tudo! Se escrever um pouco de JavaScript que monopolize todo meu tempo, então sua interação com o usuário vai ser sofrida. Não posso fazer milagres.

Use a Cabeça!: Parece que você precisa de um pouco de ajuda.

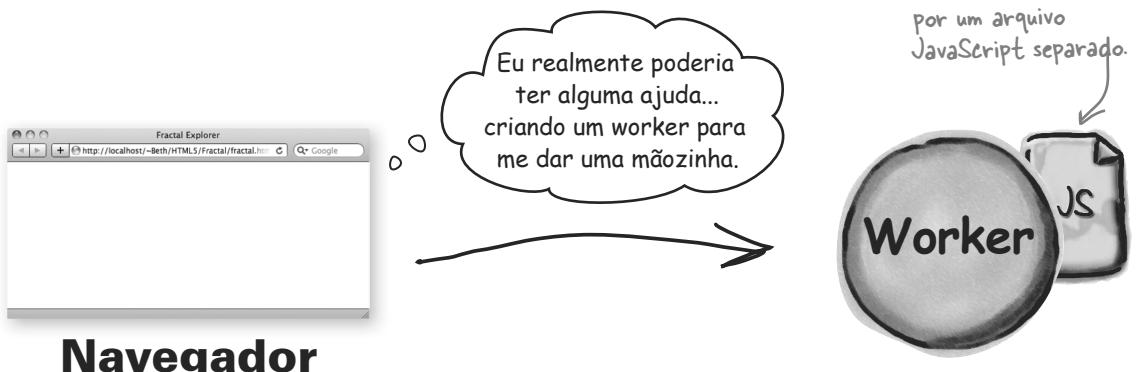
JavaScript: Bem, graças ao HTML5, eu ando tendo alguma ajuda, pois é onde entram os Web Workers. Se você realmente precisar escrever código de cálculos intensivos, use os Web Workers para aliviar um pouco do trabalho — dessa maneira, posso manter meu foco e os trabalhadores podem fazer um pouco do trabalho árduo para mim (sem ficarem no meu pé).

Use a Cabeça!: Interessante, veremos isso. Agora, próxima pergunta... Ah, espere, ele se foi. Parece que partiu para sua próxima tarefa. Cara sério, né?

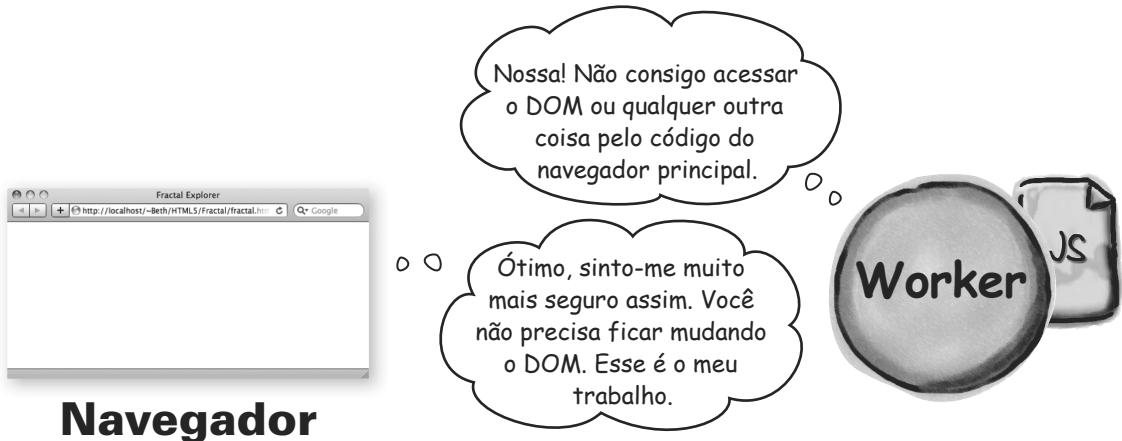
Como trabalham os Web Workers

Vamos dar uma olhada num dia de trabalho de um Web Worker: como eles são criados, como sabem o que fazer e como eles recuperam os resultados para o código de seu navegador principal.

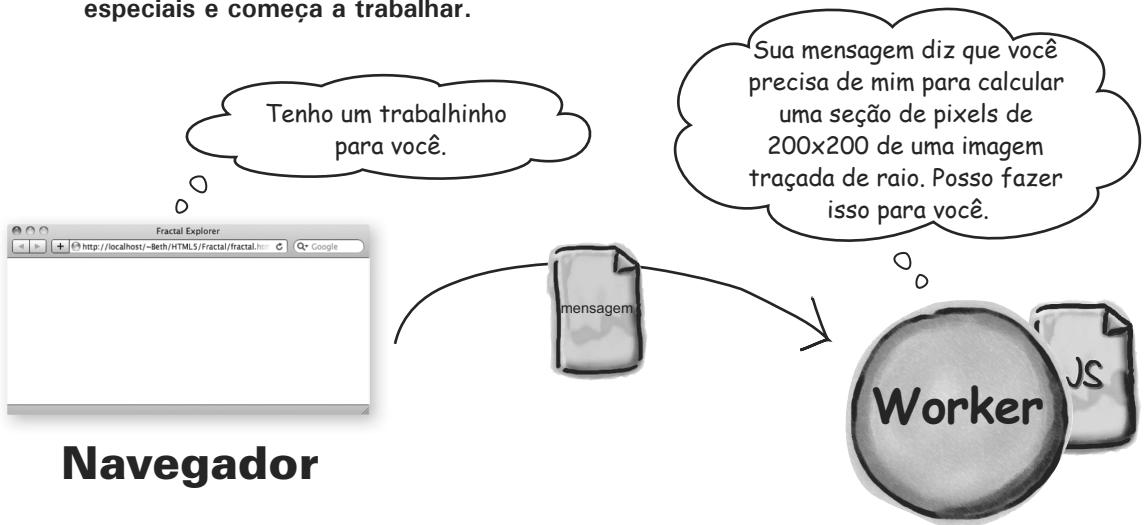
Para usar Web Workers, o navegador tem de primeiro criar um ou mais workers para ajudar nas tarefas de cálculo. Cada worker é definido pelo seu próprio arquivo JavaScript que contém todo o código (ou referências ao código) que precisa para fazer seu trabalho.



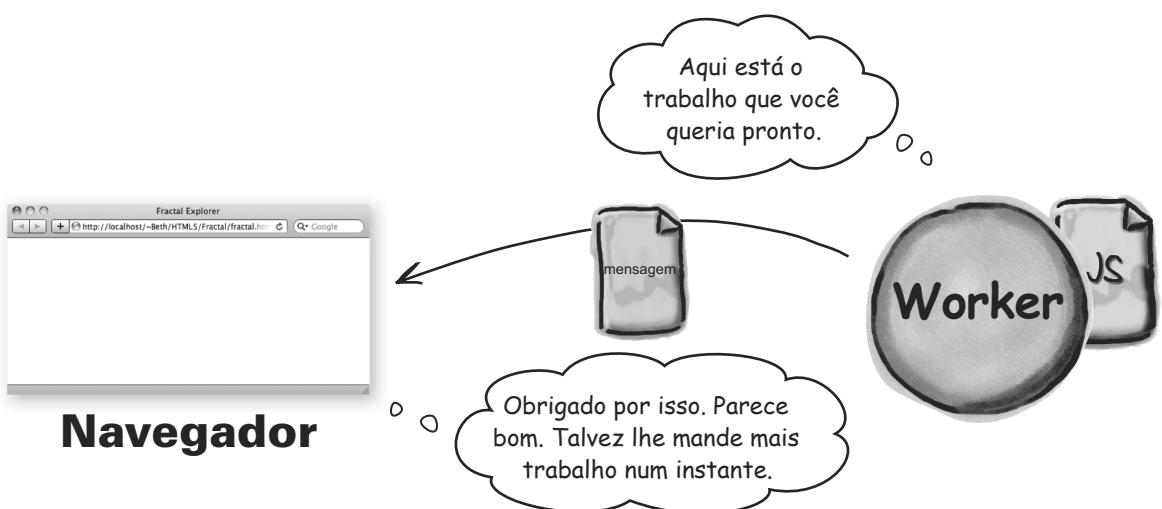
Hoje, os workers vivem num mundo bem restrito; não possuem acesso a muitos objetos runtime que o código de seu navegador principal tem, como o DOM ou qualquer das variáveis ou funções em seu código principal.



Para que um worker comece a trabalhar, o navegador normalmente envia-lhe uma mensagem. O código worker recebe a mensagem, dá uma olhada nela para ver se existem instruções especiais e começa a trabalhar.



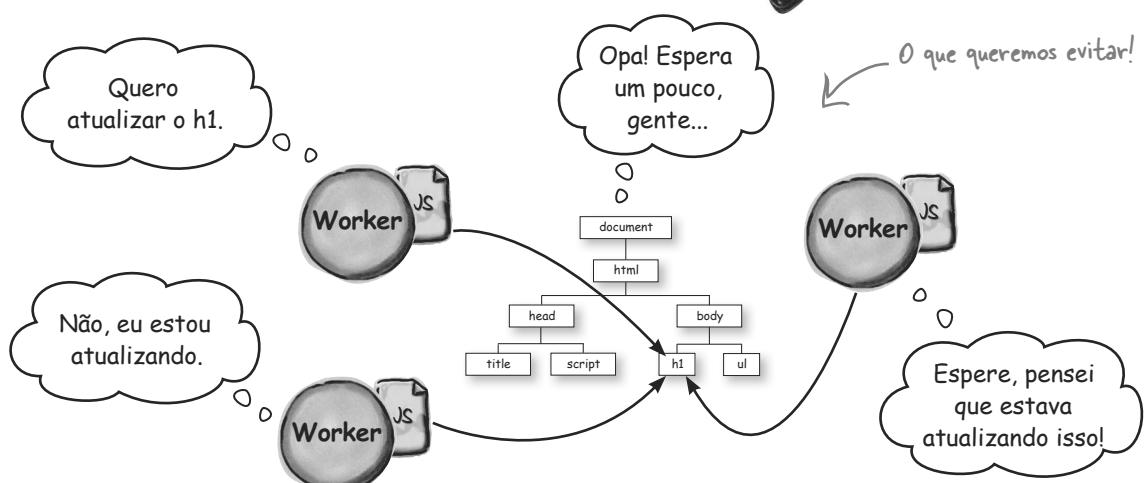
Quando o worker completa seu trabalho, ele então manda uma mensagem de volta, com os resultados finais daquilo que está sendo trabalhado. O código do navegador principal pega esses resultados e os incorpora à página de alguma maneira.



Por que não permitir que os workers accessem o DOM? Digo, parece-me muito trabalhoso enviar e receber mensagens, quando todos esses workers estão passeando no mesmo navegador.

Para manter tudo eficiente.

Uma razão para o DOM e o JavaScript serem tão bem-sucedidos é que somos capazes de otimizar altamente as operações do DOM, porque temos apenas um thread com acesso ao DOM. Se deixarmos múltiplos threads de cálculo mudarem concorrentemente o DOM, criaremos sérios impactos em seu desempenho (e os implementadores dos navegadores teriam de fazer grandes esforços para se certificarem de que as mudanças ao DOM fossem seguras). A verdade é que permitir muitas mudanças ao DOM ao mesmo tempo pode facilmente levar a situações em que ele fique num estado inconsistente, o que seria ruim. Muito ruim.





Aponte o seu lápis

Dê uma olhada em todos os usuários em potencial para workers abaixo. Quais poderiam melhorar o design e a performance de um aplicativo?

- Pegar dados para usar em suas páginas.
- Corrigir a ortografia da página à medida que o usuário digita.
- Processar grandes quantidades de dados em arrays ou grandes respostas JSON de serviços web.
- Opinar nos serviços web e alertar a página principal quando algo acontece.
- Gerenciar uma conexão database, adicionando e removendo os registros, em conjunto, para a página principal.
- Processamento de dados de imagem num canvas.
- Agente de apostas para corridas automatizadas.
- Destacar sintaxe de código ou outra linguagem.
- Analisar vídeo.
- Prévia busca de dados baseada no que seu usuário está fazendo.
- Gerenciar anúncios em página.
-
-
-
-
-
-



Suas ideias aqui!

Todas as respostas são de boa valia, embora você possa debater um pouco: verificá-las de ortografia e verificá-las de sintaxe podem ser feitas melhor no código da página principal. Já aposta em corridas podem não ser. :-)



Veja bem!

O navegador Chrome, do Google, possui algumas restrições de segurança extra que irão evitar que você rode Web Workers diretamente de um arquivo. Se tentar, sua página não vai rodar e não receberá qualquer notificação por causa disso (nem mesmo mensagens de erro, dizendo-lhe o que está errado!).

Então, para esses exemplos, recomendamos tanto o uso de diferentes navegadores, quanto rodar seu próprio servidor e os exemplos de `http://localhost`. Ou ainda pode fazer o upload deles para um servidor hospedado, se tiver acesso a um.

Você também pode usar o switch de runtime do Chrome `--allow-file-access-from-files`, mas não recomendamos esse switch além de apenas testar seu código.



Veja bem!

Quase todos os navegadores modernos dão suporte a Web Workers, mas há uma exceção: Internet Explorer 9. A boa notícia é que, a partir do IE10, você vai poder contar com os Web Workers. Para o IE9 e todos os anteriores, você terá de fornecer uma experiência alternativa.

Bem, em vez de se preocupar a respeito do IE especificamente, veja como você pode facilmente verificar se qualquer navegador suporta Web Workers:

Se os workers forem suportados,
a propriedade `Workers` será
definida no âmbito global, `window`.

```
if (window["Worker"]) {  
    var status = document.getElementById("status");  
    status.innerHTML = "Bummer, no Web Workers";  
}
```

E se `Workers` não estiver
definido, então não temos
suporte no navegador.

↑
Você vai querer manipular essa condição
de uma maneira que seja apropriada
para seu aplicativo. Aqui, apenas estamos
permitindo que o usuário saiba, ao pôr uma
mensagem num elemento com `id="status"`.

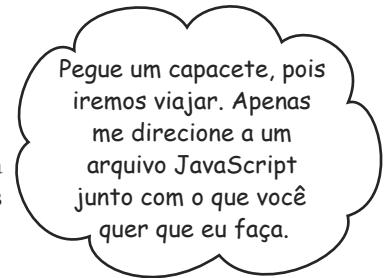
Seu primeiro Web Worker...

Vamos direto ao trabalho de criar um worker para ver como tudo isso funciona. Para isso, precisamos de uma página para pôr tudo dentro. Usaremos a marcação HTML5 mais simples que pudermos; digite isto em pingpong.html:

```
<!doctype html>
<html lang="en">
  <head>
    <title>Ping Pong</title>
    <meta charset="utf-8">
    <script src="manager.js"></script>
  </head>
  <body>
    <p id="output"></p>
  </body>
</html>
```

Este código JavaScript vai criar e gerenciar todos os workers.

E poremos um output do worker aqui.



Como criar um Web Worker

Antes de começarmos a implementar manager.js, vejamos como, de fato, criamos um Web Worker.

Para criar um novo worker,
criamos um novo objeto Worker...

```
var worker = new Worker("worker.js");
```

E designámos o novo worker
a uma variável JavaScript
chamada worker.

... o arquivo JavaScript "worker.js"
contém o código para o worker.

Portanto, é assim que se cria um worker, mas, é claro, você não precisa parar aí; você pode criar quantos workers quiser:

```
var worker2 = new Worker("worker.js");
var worker3 = new Worker("worker.js");
```

Podemos facilmente criar
dois ou mais workers que
façam uso do mesmo código
como nosso primeiro worker.

```
var another_worker = new Worker("another_worker.js");
```

Ou podemos criar outros workers baseados
num arquivo JavaScript diferente.

Veremos como usar
múltiplos workers
juntos num instante...

Escrevendo Manager.js

Agora que você sabe como criar um worker (e quão fácil é), vamos começar a trabalhar em nosso código manager.js. Vamos deixar este código simples e criar apenas um worker por ora. Crie um arquivo chamado manager.js e adicione este código:

```
window.onload = function() {  
    var worker = new Worker("worker.js");  
}
```

Esperemos para que
a página carregue
por completo.

E aí criamos um
novo worker.

Esse é um belo começo, mas agora queremos que o worker realmente faça alguma coisa. Como já discutimos, uma forma de dizer ao worker para fazer alguma coisa é enviando-lhe uma mensagem. Para enviá-la, usamos o método postMessage do objeto worker. Veja como usá-lo:

```
window.onload = function() {  
    var worker = new Worker("worker.js");  
  
    worker.postMessage("ping");  
}
```

O método postMessage é definido
por você na API Web Worker.

E estamos usando o
método postMessage
do worker para lhe
enviar uma mensagem.
Nossa mensagem é a
string simples "ping".

Quer mandar
mensagens mais
complexas?
Veja como...



postMessage de Perto

Você pode mandar mais do que apenas strings em postMessage. Vamos dar uma olhada em tudo o que você pode mandar numa mensagem:

```
worker.postMessage("ping");  
worker.postMessage([1, 2, 3, 5, 11]);  
worker.postMessage({ "message": "ping", "count": 5});
```

Você pode mandar uma string...

... ou mesmo um
objeto JSON.

Você não pode mandar funções:

```
worker.postMessage(updateTheDOM);
```

Você não pode mandar uma função...
Ela pode conter uma referência ao
DOM, permitindo ao worker mudá-lo!

Recebendo mensagens do worker

Ainda não acabamos de trabalhar com nosso código manager.js — precisamos ser capazes de receber uma mensagem do worker, se formos utilizar todo seu potencial de trabalho árduo. Para receber uma mensagem do worker, precisamos definir um handler para a propriedade onmessage do worker. Assim, qualquer hora que uma mensagem chegar do worker, nosso handler será chamado (e terá a mensagem recebida). Veja como fazer isso:

```
window.onload = function() {
    var worker = new Worker("worker.js");
    worker.postMessage("ping");
```

Aqui estamos definindo uma função que será chamada sempre que recebermos uma mensagem deste worker. A mensagem do worker é envolvida num objeto event.

```
worker.onmessage = function (event) {
    var message = "Worker says " + event.data;
    document.getElementById("output").innerHTML = message;
};
```

O objeto event passado a nosso handler possui uma propriedade data que contém os dados da mensagem (o que estávamos atrás) que o worker enviou.

Quando recebemos uma mensagem do worker, devemos enfiá-la num elemento <p> na página HTML.



onMessage de Peito

Vamos dar uma rápida olhada na mensagem que nosso handler onmessage está recebendo do worker. Como dissemos, esta mensagem é envolvida num objeto Event, que possui duas propriedades, nas quais estamos interessados: data e target.

```
worker.onmessage = function (event) {
    var message = event.data;
    var worker = event.target;
```

Este é o objeto que é enviado do worker para o código em sua página, quando o worker posta uma mensagem.

A propriedade data contém a mensagem que o worker enviou (por exemplo, uma string, como "pong").

E target é uma referência ao worker que enviou a mensagem. Isso pode ser bem útil, se você precisar saber de qual worker ela veio. Usaremos isso mais tarde, ainda neste capítulo.

Agora vamos escrever o worker

Para começar com o worker, a primeira coisa que precisamos fazer é ter certeza de que ele pode receber mensagens que são enviadas do `manager.js` — é como o worker recebe suas ordens. Para isso, vamos fazer uso de outro handler `onmessage`, aquele no próprio worker. Todo worker está pronto para receber mensagens, desde que você dê a ele um handler para processá-las. Veja como faremos isso (vá em frente, crie um arquivo `worker.js` e adicione este código):

```
onmessage = pingPong;
```

↑

Estamos designando a propriedade `onmessage` no worker para a função `pingPong`.

↑

Vamos escrever a função `pingPong` para manipular quaisquer mensagens que cheguem.

Escrevendo o handler da mensagem do worker

Vamos escrever o handler da mensagem do worker, `pingPong`, e começar do básico. Veja o que ele irá fazer (você já deve ter imaginado, dado o nome `pingPong`); o worker vai checar toda mensagem que receber para se certificar de que contenha a string “ping” e, se assim o for, enviar uma mensagem de volta que diga “pong”. Então, o trabalho do worker, de fato, é apenas receber um “ping” e responder com um “pong” — não vamos complicar os cálculos por enquanto, mas apenas nos certificar de que o manager e o worker estão se comunicando. Ah, e se a mensagem não disser “ping”, vai apenas ignorá-la.

Portanto, a função `pingPong` recebe uma mensagem e responde com “pong”. Vá em frente e adicione este código ao `worker.js`:

```
onmessage = pingPong;  
  
function pingPong(event) {  
    if (event.data == "ping") {  
        postMessage("pong");  
    }  
}
```

Quando o worker recebe uma mensagem do código principal, a função `pingPong` será chamada e a mensagem será passada.

Se a mensagem contiver uma string que diga “ping”, responderemos com uma mensagem que diga “pong”. A mensagem do worker volta ao código que o criou.

Perceba que o worker usa `postMessage` para enviar mensagens também.

Servindo um test drive



Certifique-se de ter digitado e salvo pingpong.html, manager.js e worker.js. Agora, mantenha esses arquivos abertos, para que você possa revisá-los, e vamos analisar como tudo funciona. Primeiro, o manager.js cria um novo worker, designa um handler de mensagem e então aguarda. Em algum momento, o worker vai receber uma mensagem do manager e, quando isso acontecer, ele verificará se contém “ping” (o qual está lá). Depois o worker faz ~~um monte~~ de muito pouco trabalho árduo e envia uma mensagem “pong” de volta.

Neste ponto, o código do navegador principal recebe uma mensagem do worker, que envia ao handler de mensagem. O handler então simplesmente prefixa “Worker says” (Worker diz) na frente da mensagem e a exibe.

Agora, nossos cálculos aqui dizem que a página deveria mostrar “Worker says pong”... Ok, ok, já sabemos, você não aguenta mais o suspense... Vá em frente e carregue a página!



Espere um instante, pense um pouco... Se sempre criarmos mais de um pong worker, talvez eu tenha de ralar mesmo.



Sinta-se como o Navegador



Está na hora de fingir que você é o navegador avaliando o JavaScript. Para cada parte do código abaixo, aja como se fosse o navegador e escreva seu output nas linhas fornecidas. Presuma que este código está usando o mesmo worker.js que acabamos de escrever:

← Você pode ver as soluções no fim do capítulo.

```
window.onload = function() {  
    var worker = new Worker("worker.js");  
    worker.onmessage = function(event) {  
        alert("Worker says " + event.data);  
    }  
    for (var i = 0; i < 5; i++) {  
        worker.postMessage("ping");  
    }  
}
```

.....
.....
.....
.....
.....

```
window.onload = function() {  
    var worker = new Worker("worker.js");  
    worker.onmessage = function(event) {  
        alert("Worker says " + event.data);  
    }  
    for(var i = 5; i > 0; i--) {  
        worker.postMessage("pong");  
    }  
}
```

.....
.....
.....
.....
.....

```
window.onload = function() {  
    var worker = new Worker("worker.js");  
    worker.onmessage = function(event) {  
        alert("Worker says " + event.data);  
        worker.postMessage("ping");  
    }  
    worker.postMessage("ping");  
}
```



Cuidado ao tentar esses; você
talvez tenha de matar seu
navegador para escapar...



```
window.onload = function() {  
    var worker = new Worker("worker.js");  
    worker.onmessage = function(event) {  
        alert("Worker says " + event.data);  
    }  
  
    setInterval(pinger, 1000);  
  
    function pinger() {  
        worker.postMessage("ping");  
    }  
}
```



Aponte o seu lápis

Embora os workers tipicamente recebam suas ordens de trabalho por meio de mensagens, eles não precisam. Veja esta forma fácil e compacta de ter seu trabalho feito com os workers e com o HTML. Quando souber o que ele faz, descreva abaixo. Você poderá comparar sua solução com a nossa no fim do capítulo.

```
<!doctype html>           quote.html
<html lang="en">          ↙
  <head>
    <title>Quote</title>
    <meta charset="utf-8">
  </head>
  <body>
    <p id="quote"></p>
    <script>
      var worker = new Worker("quote.js");
      worker.onmessage = function(event) {
        document.getElementById("quote").innerHTML = event.data;
      }
    </script>
  </body>
</html>

var quotes = ["I hope life isn't a joke, because I don't get it.",
              "There is a light at the end of every tunnel... just pray it's
not a train!",
              "Do you believe in love at first sight or should I walk by
again?"];
var index = Math.floor(Math.random() * quotes.length);
postMessage(quotes[index]);

Sua descrição aqui:
```

quote.js
↓

Tente digitar e
rodar o código!



Vamos acrescentar alguns workers ao nosso jogo pingPong. Seu trabalho é preencher os campos em branco para completar o código. Assim, teremos três pings enviados aos workers e três pongs devolvidos por eles.

Escreva seu
código nos
campos em
branco.

```
window.onload = function() {
    var numWorkers = 3; ← Estamos criando três
    var workers = []; ← workers e armazenando-
    for (var i = 0; i < .....; i++) {
        var worker = new .....("worker.js");
        worker..... = function(event) {
            alert(event.target + " says "
                + event.....);
        };
        workers.push(worker);
    }
    for (var i = 0; i <.....; i++) {
        workers[i] ....."ping");
    }
}
```

Aqui, estamos adicionando um novo worker para o array de workers.

não existem Perguntas Idiotas

P: Posso apenas passar uma função, em vez de um arquivo JavaScript, quando eu criar o worker? Isso me parece mais fácil e mais consistente com o que o JavaScript normalmente faz.

R: Não, não pode. Entenda porquê: como você sabe, um dos requerimentos de um worker é que ele não possua acesso ao DOM (ou a qualquer estado do thread do navegador principal). Se pudesse passar uma função ao construtor Worker. Então sua função poderia também conter referência ao DOM ou a outras partes de seu código JavaScript principal, o que violaria o requerimento. Então, os designers do Web Workers preferem, em vez disso, que você simplesmente passe uma URL JavaScript para evitar esse problema.

P: Quando envio a um worker um objeto numa mensagem, ele se torna um objeto compartilhado entre minha página principal e o worker?

R: Não, quando você envia um objeto, o worker recebe uma cópia dele. Qualquer mudança que o worker faça não afetará o objeto em sua página principal. O worker está executando num ambiente diferente de sua página principal. Então, não tem acesso aos objetos lá. O mesmo acontece com os objetos que o worker lhe envia: você recebe uma cópia deles.

P: Os workers podem acessar localStorage ou fazer XMLHttpRequests?

R: Sim, os workers podem acessar o localStorage e fazer XMLHttpRequests.



Vamos acrescentar alguns workers ao nosso jogo pingPong. Seu trabalho é preencher os campos em branco para completar o código. Assim, teremos três pings enviados aos workers e três pongs devolvidos por eles. Veja nossa solução.

Usamos numWorkers para iterar três vezes
e criar três workers (fique à vontade para
mudar esta variável e adicionar mais!)

```
window.onload = function() {
    var numWorkers = 3;
    var workers = [];
    for (var i = 0; i < numWorkers; i++) {
        var worker = new Worker("worker.js");
        worker.onmessage = function(event) {
            alert(event.target + " says "
                + event.data);
        };
        workers.push(worker);
    }
    for (var i = 0; i < workers.length; i++) {
        workers[i].postMessage("ping");
    }
}
```

Nós enviamos ping ao worker com postMessage.

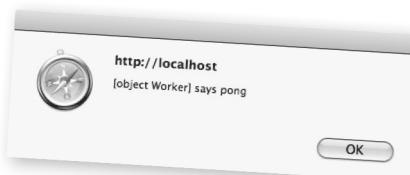
Criamos o handler da mensagem no código de nossa página principal, usando a propriedade onmessage do worker.

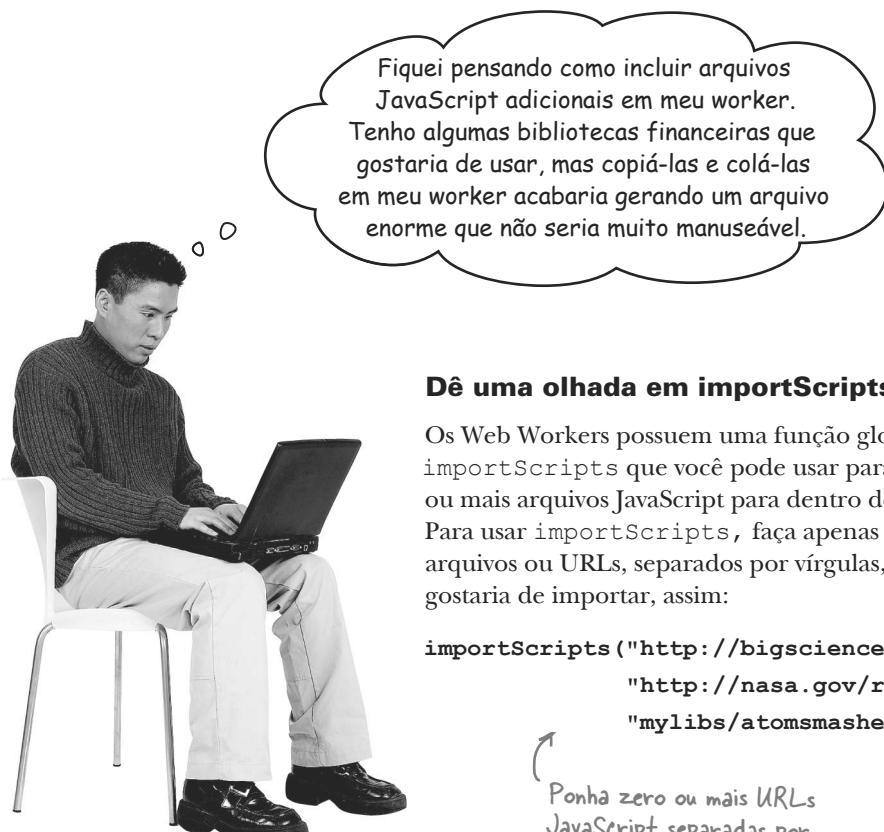
Usamos a propriedade data para receber os conteúdos da mensagem.

Você também poderia usar numWorkers aqui, se quisesse.

Perceba que não são necessárias mudanças ao código do worker. Cada worker faz seu trabalho feliz e independente.

Você verá este alerta 3 vezes.





Dê uma olhada em importScripts.

Os Web Workers possuem uma função global chamada `importScripts` que você pode usar para importar um ou mais arquivos JavaScript para dentro de seu worker. Para usar `importScripts`, faça apenas uma lista de arquivos ou URLs, separados por vírgulas, que você gostaria de importar, assim:

```
importScripts ("http://bigscience.org/nuclear.js",
              "http://nasa.gov/rocket.js",
              "mylibs/atomsmasher.js");
```

Ponha zero ou mais URLs JavaScript separadas por vírgulas em `importScripts`.

Então, quando `importScripts` for invocado, cada URL JavaScript será resgatada e avaliada em ordem.

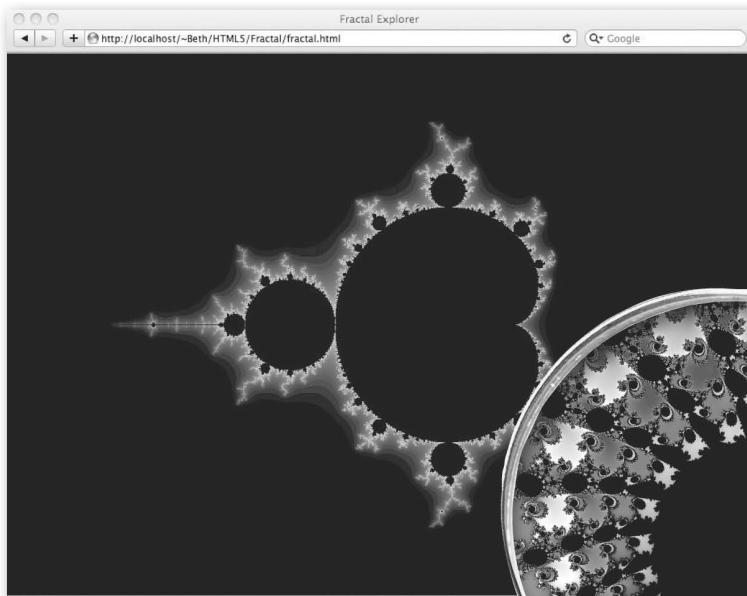
Note que `importScripts` é uma função plena (ao contrário das declarações `import` numa porção de linguagens). Você pode tomar decisões runtime a respeito de importações, assim:

```
if (taskType == "songdetection") {
    importScripts("audio.js");
}
```

Por ser `importScripts` uma função, você pode importar o código como se fosse requerimento da tarefa.

Expropriação Virtual

Exploradores do Mandelbrot Set já tomaram terras do interior virtual, dando-lhes nomes como o amável “Seahorse Valley”, “Rainbow Islands” e o terrível “Black Hole”. Dado o valor das propriedades físicas hoje em dia, a única possibilidade que sobra parece ser nos espaços virtuais. Portanto, vamos construir um explorador para o Mandelbrot Set e pôr isso em ação. De fato, temos de confessar que já o construímos, mas está lento — navegar ao redor de todo o MandelbrotSet levaria muito tempo — então, esperamos que juntos possamos acelerá-lo e temos a sensação de que os Web Workers podem ser a solução.



Dê uma olhada em volta

Vá em frente e dispare <http://wickedlysmart.com/hfhtml5/chapter10/singlethread/fractal.html> e você terá uma visualização do Mandelbrot Set à distância. Clique em qualquer lugar e terá um zoom na área do mapa. Continue clicando para explorar diferentes áreas ou recarregue para começar novamente. Fique atento para áreas com buracos negros, pois elas tendem a sugar. Não sabemos quanto a você, mas embora o cenário seja bonito, nosso visualizador poderia ser um pouquinho mais rápido, não acha? Seria ótimo também ter desempenho suficiente para maximizar a visualização para a janela inteira do navegador! Vamos consertar isso tudo, acrescentando os Web Workers ao Fractal Explorer.



Mandel o quê?

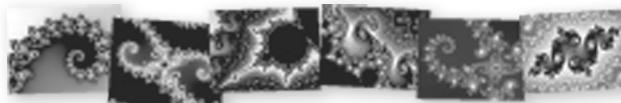
Bem, se você for um matemático,

deve saber que Mandelbrot Set é a equação:

$$z_{n+1} = z_n^2 + c$$

Ela foi descoberta e estudada por Benoit Mandelbrot. Também deve saber que é um conjunto de números complexos (números com uma parte real e uma parte imaginária) gerados pela equação.

Se, por outro lado, você não é um matemático, a melhor maneira de se pensar na Mandelbrot Set é como se ela fosse uma imagem fractal infinitamente complexa — significa que é uma imagem que você pode dar zoom em qualquer nível de intensidade e encontrar interessantes estruturas. Veja só algumas das coisas que você pode encontrar navegando dentro do conjunto:



Por que estamos tão interessados nisso? Bem, o conjunto possui algumas propriedades interessantes. Primeiramente, é gerado por uma equação bem simples (essa logo acima), que pode ser expressa em apenas algumas linhas de código; segundo, gerar Mandelbrot Set leva um número e tanto de ciclos de cálculo, o que a torna um grande exemplo para usar Web Workers. Por fim, é uma viagem legal e um grande aplicativo para se terminar o livro, não acha?



Desejase em paz, Benoit Mandelbrot, que faleceu enquanto este livro estava sendo escrito. Tivemos muita sorte de tê-lo conhecido.

Como calcular um Mandelbrot Set

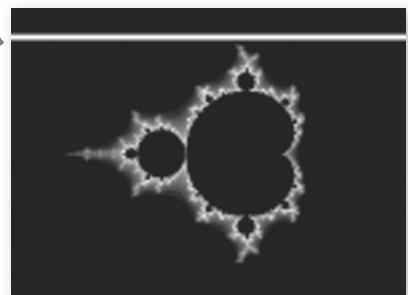
Vamos dar uma olhada em como você normalmente estrutura seu código para calcular um Mandelbrot Set antes de envolvêmos os workers. Não queremos focar muito nos pequenos detalhes de calcular os valores em pixel do Mandelbrot; já demos conta de todo esse código e já vamos passá-lo a você num segundo. Por ora, apenas queremos que você se conscientize da visão geral de como calcular o conjunto:

```
for (i = 0; i < numberOfRows; i++) {
    var row = computeRow(i);
    drawRow(row);
}
```

Depois, desenhamos cada fileira na tela. Provavelmente, você verá no display fileira por fileira, quando rodar o código de teste em seu navegador.

Para computar o Mandelbrot Set, passamos por cada fileira da imagem. E para cada fileira nós calculamos os pixels para ela.

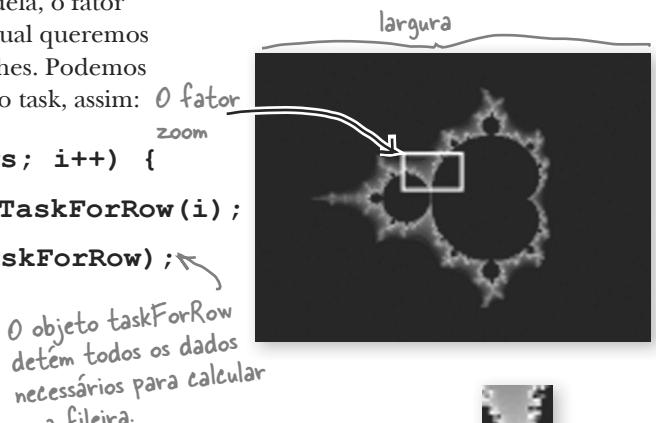
Perceba que nossa meta aqui não é ensiná-lo a ser um analista de números (que pode codificar equações com números complexos), mas sim adaptar um aplicativo de cálculos intensivos a usar os Web Workers. Se estiver interessado nos aspectos numéricos do Mandelbrot Set, a Wikipedia é um excelente lugar para começar.



Agora, este código só servirá como simples pseudocódigo — quando chegar a hora de escrever o código verdadeiro, haverá alguns detalhes a mais que precisaremos ater: por exemplo, para calcular uma fileira, precisamos saber a largura dela, o fator de zoom, a resolução numérica para a qual queremos calcular e alguns outros pequenos detalhes. Podemos capturar todos esses detalhes num objeto task, assim:

```
for (i = 0; i < numberOfRows; i++) {
    var taskForRow = createTaskForRow(i);
    var row = computeRow(taskForRow);
    drawRow(row);
}
```

E passamos taskForRow para dentro de computeRow, que retorna a fileira calculada.



Agora, o truque será pegar isso, retrabalhá-lo para dividir o cálculo entre o número de workers e, então, acrescentar o código, que manipula a distribuição de tasks (tarefas) para os workers e manipula como lidar com os resultados quando os workers concluem suas tarefas.

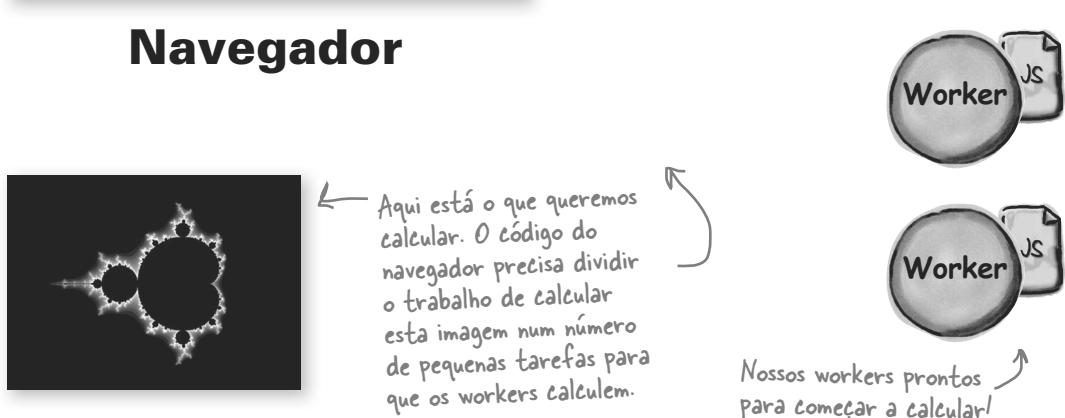
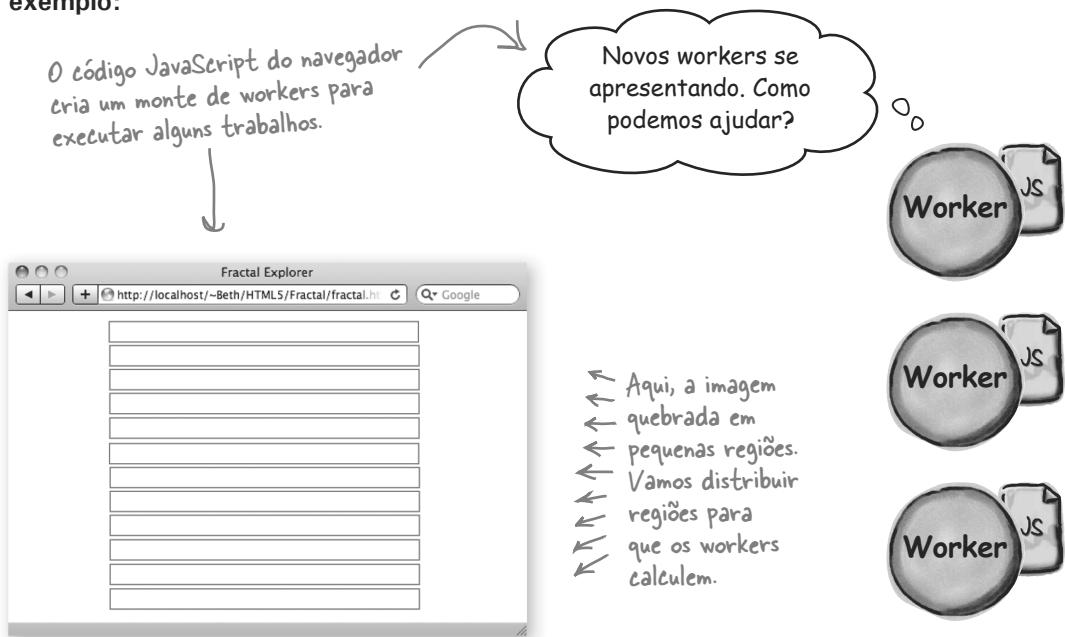
Nível de
precisão
para calcular



Como usar múltiplos workers

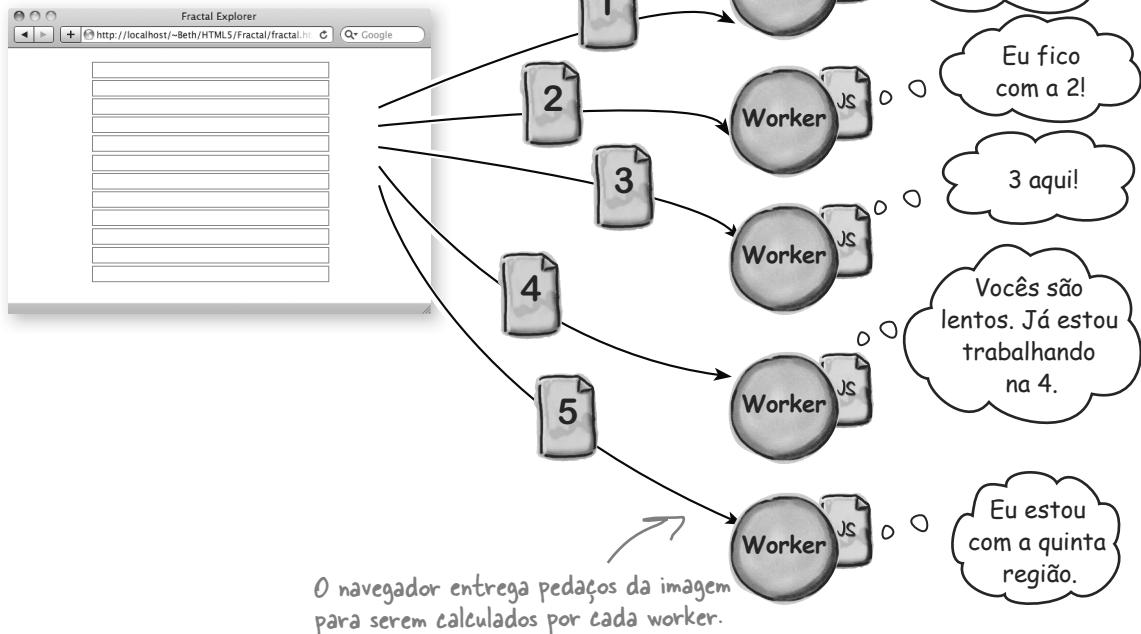
Você já sabe como criar novos workers. Agora, como vai usá-los para que façam algo mais complicado, do tipo calcular as fileiras do Mandelbrot Set? Ou aplicar um efeito tipo Photoshop numa imagem? Ou traçar raios numa cena de filme? Em todos esses casos, podemos dividir o trabalho em pequenas tarefas em que cada worker poderá trabalhar independentemente. Por ora, vamos nos ater a calcular o Mandelbrot Set (mas o padrão que usaremos pode ser aplicado a qualquer desses exemplos).

Para começar, o navegador primeiro cria um monte de workers para ajudar (mas nem tantos — workers podem se tornar bem caros, se criarmos muitos deles — falaremos sobre isso mais tarde). Usaremos apenas cinco workers para este exemplo:

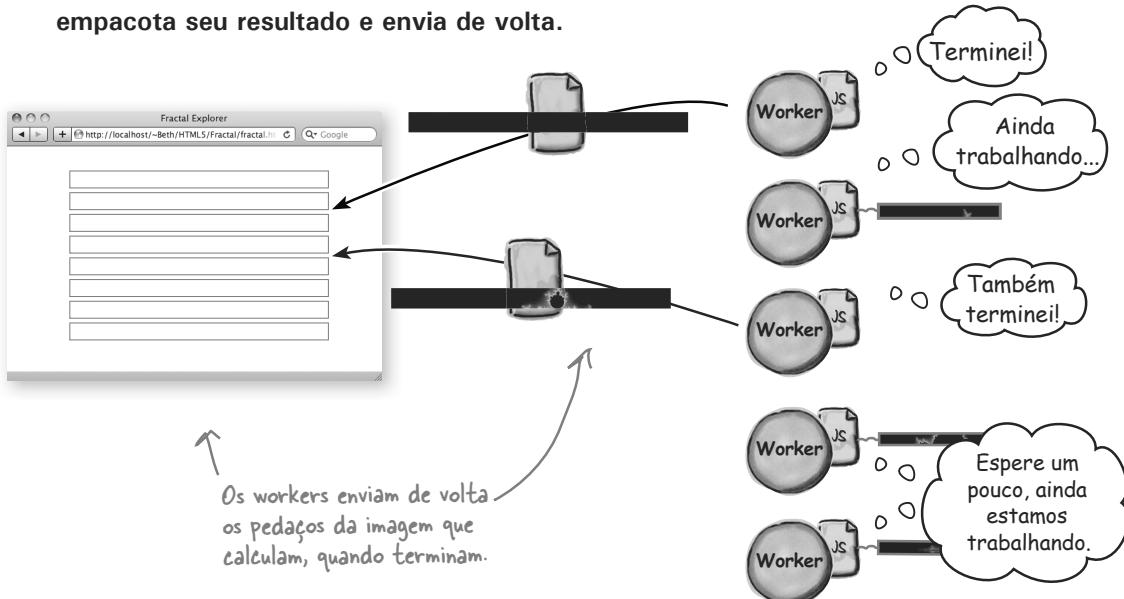


como computar com workers

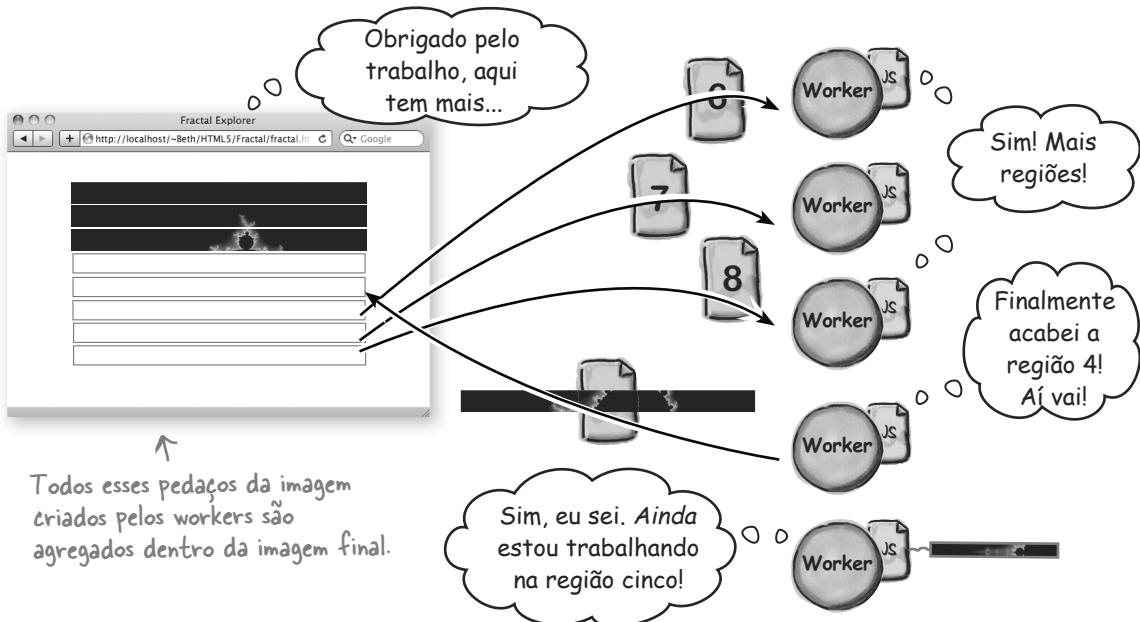
Em seguida, o código do navegador distribui uma parte diferente da imagem para cada worker calcular:



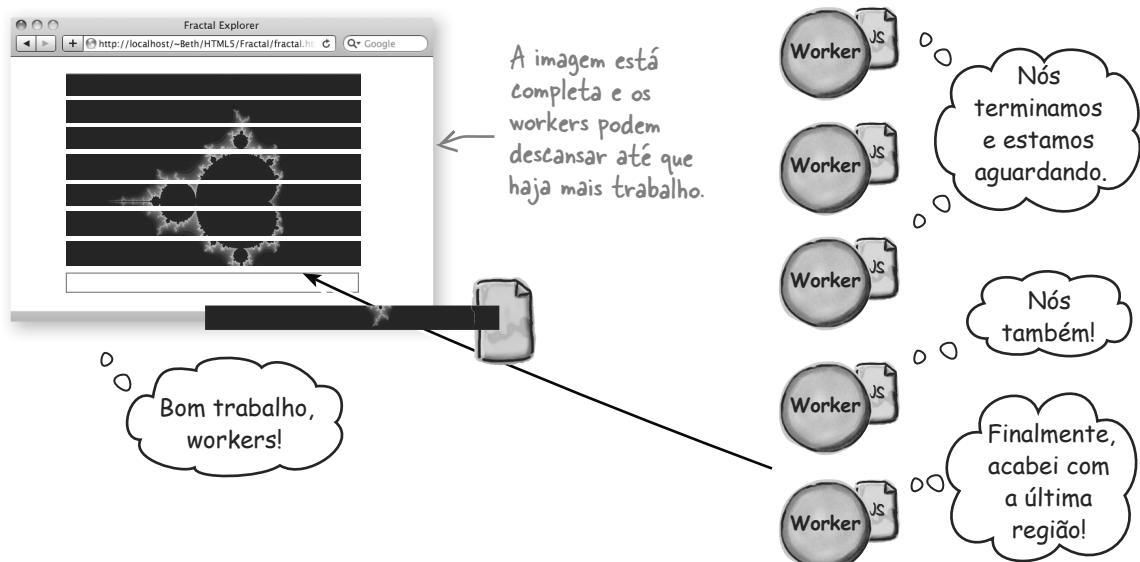
Cada worker trabalha em seu próprio pedaço da imagem, independentemente. À medida que o worker acaba sua tarefa, ele empacota seu resultado e envia de volta.



À medida que os pedaços da imagem vão retornando dos workers, eles são agregados dentro da imagem no navegador e, se houver mais pedaços a serem calculados, novas tarefas serão entregues aos workers que já terminaram.



Com a última parte da imagem computada, ela está completa e os workers descansam até o usuário clicar no zoom in. Então, começa tudo outra vez...



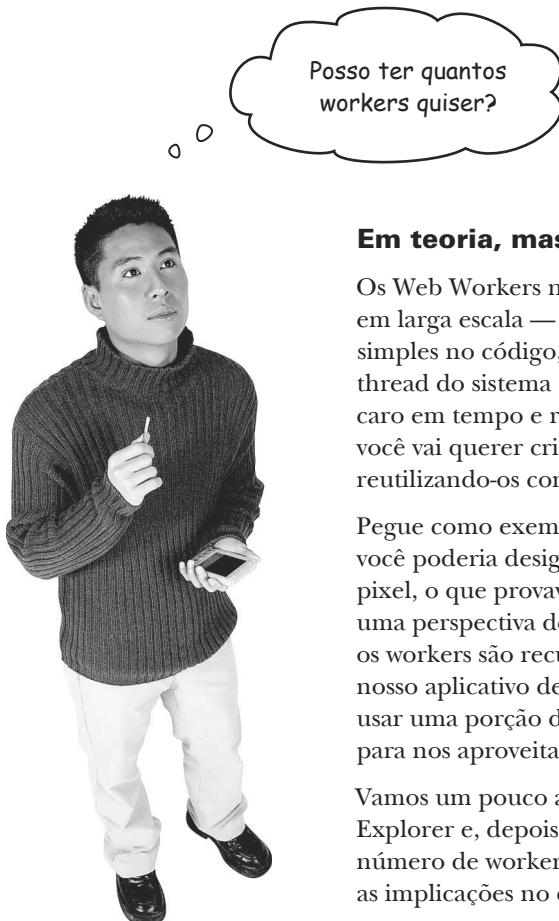
O que importa se eu dividir a tarefa e distribuí-la aos workers? Quero dizer, meu computador ainda possui a mesma CPU. Como o cálculo pode ser mais rápido?

Pode ser mais rápido de duas maneiras...

Primeiro, considere um aplicativo que possui muitos “cálculos” acontecendo e que também precisa ser responsivo ao usuário. Se seu aplicativo estiver monopolizando muito o tempo do JavaScript, seus usuários vão experimentar uma interface molenga (novamente, porque o JavaScript é single-thread). Ao adicionar workers para um aplicativo assim, pode-se melhorar imediatamente a sensação dele para seus usuários. Por quê? Porque o JavaScript tem uma chance de responder à interação do usuário, enquanto obtém resultados dos workers; algo que não acontece, se tudo estiver sendo calculado no thread principal. Então, o UI é mais responsivo... e seu aplicativo vai *dar a impressão* de ser mais rápido (mesmo que, por baixo dos panos, não esteja nem um pouquinho mais veloz). Não acredita em nós? Experimente e ponha alguns usuários reais na frente de seu aplicativo. Pergunte a eles o que acham.

A segunda maneira é *realmente mais rápida*. Quase todos os desktops e dispositivos modernos são feitos com processadores multi núcleo (e talvez até múltiplos processadores). Multi núcleo significa apenas que o processador pode fazer várias tarefas ao mesmo tempo. Com apenas um single thread de controle, o JavaScript no navegador não faz uso de suas cores ou processadores extras. Eles simplesmente são desperdiçados. No entanto, se usar os Web Workers, os workers podem levar vantagem, correndo em suas diferentes cores. Você verá uma real aceleração em seu aplicativo porque terá mais poder de processamento em atividade. Se você tem uma máquina multi núcleo, apenas espere. Verá a diferença já, já.





Em teoria, mas não na prática.

Os Web Workers não foram feitos para serem usados em larga escala — embora criar workers pareça ser simples no código, eles requerem memória extra e um thread do sistema operacional, o que pode se tornar caro em tempo e recursos iniciais. Portanto, em geral, você vai querer criar um número limitado de workers, reutilizando-os com o tempo.

Pegue como exemplo nosso Mandelbrot: na teoria, você poderia designar um worker para calcular cada pixel, o que provavelmente seria muito mais simples de uma perspectiva de design do código, mas, dado que os workers são recursos de peso, nunca desenhariíamos nosso aplicativo dessa forma. Em vez disso, é melhor usar uma porção de workers e estruturar nosso cálculo para nos aproveitarmos deles.

Vamos um pouco a frente no desenho do Fractal Explorer e, depois, voltaremos para brincar com o número de workers que estamos usando, para entender as implicações no desempenho.



Você, agora, com certeza tem uma boa base para construir aplicativos Web Worker, como criar e utilizar workers, um pouco sobre como resolver grandes cálculos ao dividi-los em pequenas tarefas que podem ser calculadas por seus workers e até mesmo sabe um pouco sobre como os conjuntos Mandelbrot são calculados. Tente juntar isso tudo e analise como você pegaria o pseudocódigo abaixo e o reescreveria usando workers. Talvez você primeiramente presumiu que já tem a quantidade de workers necessária (digamos um worker para cada fileira), mas depois adicione: descobrirá um problema possui um número limitado de workers (menos workers que o número de fileiras):

```
for (i = 0; i < numberOfRows; i++) {  
    var taskForRow = createTaskForRow(i);  
    var row = computeRow(taskForRow);  
    drawRow(row);  
}
```

Aqui está nosso pseudocódigo;
agora, o que precisa para
aumentar Web Workers?

Suas notas vão aqui:

Vamos construir o aplicativo Fractal Explorer

Veja o que precisamos fazer:

- Definir nossa página HTML para manter o Mandelbrot App.
- Pegar todos os  Códigos Pronto para Assar inseridos (ou baixados).
- Criar alguns workers e defini-los para calcular.
- Designar os workers para suas tarefas.
- Implementar o código worker.
- Processar os resultados dos workers à medida em que completam suas tarefas.
- Manipular eventos click e resize na interface do usuário.

- | | |
|--------------------------|---|
| <input type="checkbox"/> | Criando uma página HTML |
| <input type="checkbox"/> |  Códigos Pronto para Assar |
| <input type="checkbox"/> | Criando Workers |
| <input type="checkbox"/> | Pondo os workers para começar |
| <input type="checkbox"/> | Implementando os workers |
| <input type="checkbox"/> | Processando os resultados |
| <input type="checkbox"/> | Testando a interação do código de usuário |

Criando a Marcação HTML Fractal Viewer

Primeiro, precisamos definir uma página HTML para manter nosso aplicativo. Você deve criar um arquivo HTML chamado `fractal.html` e adicionar a seguinte marcação. Vamos dar uma olhada:

```
<!doctype html>           Como sempre, um arquivo
<html lang="en">          HTML5 padrão.
  <head>
    <title>Fractal Explorer</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="fractal.css">
    <script src="mandellib.js"></script>
    <script src="mandel.js"></script>
  </head>
  <body>
    <canvas id="fractal" width="800" height="600"></canvas>
  </body>
</html>
```

Este código vai no `fractal.html`

Aqui estão todos os  Códigos Pronto para Assar que temos para você. Eles contêm todo o código numérico, assim como um pouco de código para manipular gráficos.

E aqui está o código JavaScript que vamos escrever... Se está pensando onde vai o código do worker, lembre-se de que não ligamos diretamente com o arquivo JavaScript do worker; fazemos referência a tal arquivo quando criamos o worker em código.

Seja! Nossa amiga `<canvas>` está de volta!!

E o `<body>` possui um elemento `canvas`. Definimo-lo a um tamanho inicial de 800x600 pixels, mas veremos como redimensioná-lo à largura e altura da janela, usando JavaScript. Afinal de contas, queremos o mais próximo de um Mandelbrot que podemos alcançar!



Código Pronto para Assar

Lembrete: você pode baixar todos os códigos de <http://wickedlysmart.com/hfhtml5>

Temos de lhe dizer que tínhamos planejando um capítulo inteiro sobre as maravilhas do Mandelbrot Set... Planejamos explicar em detalhes, incluindo uma história de Benoit Mandelbrot, como ele o descobriu, todas as suas propriedades incríveis, otimizações de pixels, mapas coloridos e assim por diante, mas aí recebemos uma ligação de nosso editor — você sabe, A LIGAÇÃO. Acho que estávamos um pouco atrasados em nosso livro; então, aí vão nossas desculpas, mas vamos ter de lhe dar um pouco de **Códigos Pronto** para fazer cálculos básicos dos gráficos Mandelbrot. Veja, no entanto, o lado bom: poderemos focar em como usar os Web Workers sem gastar os próximos dias em cálculos matemáticos e gráficos. Dito isto, encorajamo-lo a explorar esses tópicos por si só!

De qualquer forma, primeiro, temos o código usado para gerenciar tarefas e desenhar as fileiras para as imagens fractais. Comece por digitar este código dentro de um arquivo chamado "mandellib.js".

```
var canvas;
var ctx;
```



Perceba que nosso canvas e contexto estão aqui.

```
var i_max = 1.5;
var i_min = -1.5;
var r_min = -2.5;
var r_max = 1.5;
```



Essas são as variáveis globais que o código dos gráficos Mandelbrot utilizam para calcular o conjunto e mostrá-lo.

```
var max_iter = 1024;
var escape = 1025;
var palette = [];
```

```
function createTask(row) {
    var task = {
        row: row,
        width: rowData.width,
        generation: generation,
        r_min: r_min,
        r_max: r_max,
        i: i_max + (i_min - i_max) * row / canvas.height,
        max_iter: max_iter,
        escape: escape
    };
    return task;
}
```



Esta função compacta todos os dados necessários para o worker calcular uma fileira de pixels dentro de um objeto. Você verá mais tarde como passámos este objeto para que o worker o utilize.

Este código vai em mandellib.js.



Código PRONTO para ASSAR, Continuação

- Criando uma página HTML
- Código PRONTO para ASSAR
- Criando Workers
- Pondo os workers para começar
- Implementando os workers
- Processando os resultados
- Testando a interação do código de usuário

```

function makePalette() {
    function wrap(x) {
        x = ((x + 256) & 0xffff) - 256;
        if (x < 0) x = -x;
        return x;
    }
    for (i = 0; i <= this.max_iter; i++) {
        palette.push([wrap(7*i), wrap(5*i), wrap(11*i)]);
    }
}

function drawRow(workerResults) {
    var values = workerResults.values;
    var pixelData = rowData.data;
    for (var i = 0; i < rowData.width; i++) {
        var red = i * 4;
        var green = i * 4 + 1;
        var blue = i * 4 + 2;
        var alpha = i * 4 + 3;
        pixelData[alpha] = 255; // set alpha to opaque
        if (values[i] < 0) {
            pixelData[red] = pixelData[green] = pixelData[blue] = 0;
        } else {
            var color = this.palette[values[i]];
            pixelData[red] = color[0];
            pixelData[green] = color[1];
            pixelData[blue] = color[2];
        }
    }
    ctx.putImageData(this.rowData, 0, workerResults.row);
}

```

makePalette mapeia um grande conjunto de números dentro de um array de cores rgb. Usaremos esta palette em drawRow (abaixo) para converter o valor que recebemos de um worker para uma cor para o display do gráfico do conjunto (a imagem fractal).

drawRow pega os resultados do worker e desenhá-los dentro de canvas.

Ele usa esta variável rowData para isso; rowData é um objeto ImageData de fileira única que mantém os pixels verdadeiros para aquela fileira do canvas.

Aqui é onde usamos a palette para mapear os resultados do worker (apenas um número) para uma cor.

E aqui é onde escrevemos os pixels para o objeto ImageData no contexto do canvas!

Este código vai em mandellib.js.



Código PRONTO PARA ASSAR, CONTINUAÇÃO

setUpGraphics cria as variáveis globais usadas por todo o código de desenho de gráficos, assim como o cálculo do Mandelbrot.

```
function setupGraphics() {  
  
    canvas = document.getElementById("fractal");  
    ctx = canvas.getContext("2d");  
  
    canvas.width = window.innerWidth;  
    canvas.height = window.innerHeight;  
  
    var width = ((i_max - i_min) * canvas.width / canvas.height);  
    var r_mid = (r_max + r_min) / 2;  
    r_min = r_mid - width/2;  
    r_max = r_mid + width/2;  
  
    rowData = ctx.createImageData(canvas.width, 1);  
  
    makePalette();  
}  
}
```

← É aqui que pegamos o canvas e o contexto e definimos a largura e altura iniciais do canvas.

← Estas são variáveis usadas para calcular o Mandelbrot Set.

← Aqui, estamos inicializando a variável rowData (usada para escrever os pixels para o canvas).

← E aqui estamos inicializando a palette de cores que estamos usando para desenhar o conjunto como uma imagem fractal.

Este código vai em mandellib.js.



Código PRONTO PARA ASSAR, CONTINUAÇÃO

Este **Código PRONTO** é o que o worker usará para fazer os cálculos matemáticos do Mandelbrot Set. Aqui é onde, realmente, acontece a mágica do cálculo (e se você explorar o Mandelbrot Set mais profundamente, será onde você vai querer se focar). Digite este código dentro de "workerlib.js".

```

function computeRow(task) {
    var iter = 0;
    var c_i = task.i;
    var max_iter = task.max_iter;
    var escape = task.escape * task.escape;
    task.values = [];
    for (var i = 0; i < task.width; i++) {
        var c_r = task.r_min + (task.r_max - task.r_min) * i / task.width;
        var z_r = 0, z_i = 0;

        for (iter = 0; z_r*z_r + z_i*z_i < escape && iter < max_iter; iter++) {
            // z -> z^2 + c
            var tmp = z_r*z_r - z_i*z_i + c_r;
            z_i = 2 * z_r * z_i + c_i;
            z_r = tmp;
        }
        if (iter == max_iter) {
            iter = -1;
        }
        task.values.push(iter);
    }
    return task;
}

```

computeRow calcula uma fileira de dados do Mandelbrot Set. É dado um objeto com todos os valores compactados necessários para calcular aquela fileira.

Perceba que, para cada fileira do display, estamos fazendo dois loops, um para cada pixel da fileira... É muito cálculo. Ótimo!

... e outro loop para encontrar o valor certo para aquele pixel. Este loop interno é onde a complexidade computacional está, e é por isso que o código roda muito mais rápido quando possui múltiplas cores em seu computador!

O resultado final de todo aquele cálculo é um valor que é adicionado a um array de valores nomeados, que por sua vez é colocado de volta dentro do objeto task, assim o worker pode enviar o resultado de volta ao código principal.

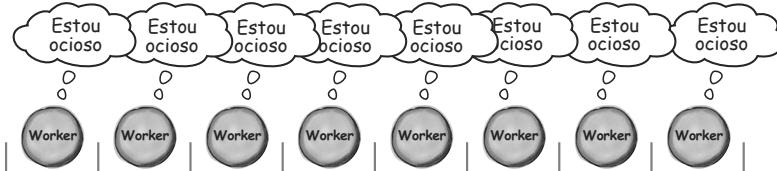
Vamos ver essa parte em mais detalhes num instante.

Este código vai em workerlib.js.

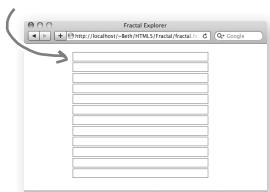
Criando workers e dando-lhes tarefas...

Com o **Código Assado** fora do caminho, vamos agora voltar nossas atenções para escrever o código que vai criar e distribuir tarefas para os workers. Veja como isso vai funcionar:

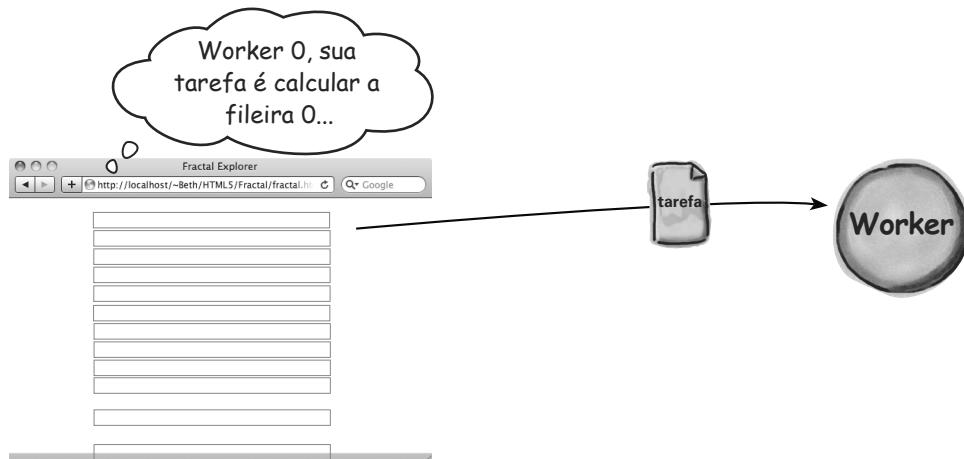
- 1 Criamos um array de workers, inicialmente todos ociosos. E uma imagem com nada calculado (`nextRow = 0`).



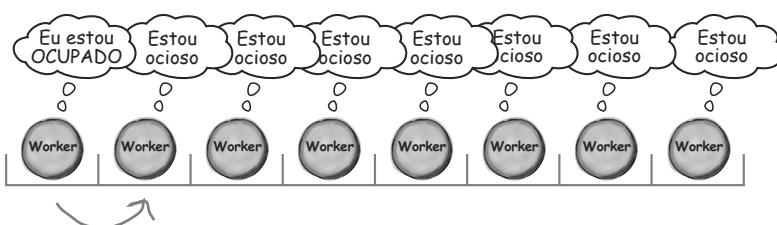
`nextRow = 0`



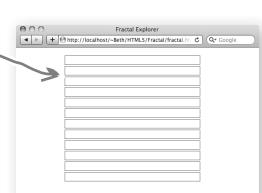
- 2 Iteramos o array e criamos uma tarefa para cada worker ocioso:



- 3 Continuamos iterando, procurando pelo próximo worker ocioso que irá receber uma tarefa. O próximo é `nextRow = 1`. E assim por diante...



`nextRow = 1`



- | | |
|-------------------------------------|---|
| <input checked="" type="checkbox"/> | Criando uma página HTML |
| <input checked="" type="checkbox"/> | Código Pronto para Assar |
| <input type="checkbox"/> | Criando Workers |
| <input type="checkbox"/> | Pondo os workers para começar |
| <input type="checkbox"/> | Implementando os workers |
| <input type="checkbox"/> | Processando os resultados |
| <input type="checkbox"/> | Testando a interação do código de usuário |

Escrevendo o código

Agora que sabemos como criar e gerenciar nossos workers, vamos escrever o código. Realmente, precisamos de uma função inicial para isso; então, vamos criar uma função em `mandel.js` chamada `init` — vamos colocar algumas outras coisas nela também, para pôr o aplicativo em funcionamento (tipo, para ter certeza que deixamos a inicialização de gráficos fora do caminho):

Criando uma página HTML
 Código Pronto para Assar
 Criando Workers
 Pondo os workers para começar
 Implementando os workers
 Processando os resultados
 Testando a interação do código de usuário

Primeiro, vamos definir uma variável que mantém o número de trabalhadores que queremos. Escolheremos o 8, mas sintase à vontade para brincar com isso quando o aplicativo estiver funcionando.

```
var numberOfWorkers = 8;
```

E aqui está um array vazio para manter nossos workers.

```
var workers = [];
```

Vamos criar um handler `onload` que chama `init` quando a página estiver carregada por completo.

```
window.onload = init;
```

Esta função é definida no Código Pronto e manipula a obtenção do contexto canvas, redimensionando-o ao tamanho do seu navegador, e alguns outros detalhes de gráfico.

```
function init() {
```

```
    setupGraphics();
```

Agora, itere sobre o número de workers... ... e crie um novo worker de "worker.js", que ainda não escrevemos.

```
for (var i = 0; i < numberOfWorkers; i++) {
```

```
    var worker = new Worker("worker.js");
```

Em seguida, definimos cada message handler de worker a uma função que chama a função `processWork`, passando-lhe o `event.target` (o worker que acabou de terminar) e o `event.data` (os resultados do worker).

```
    worker.onmessage = function(event) {
```

```
        processWork(event.target, event.data);
```

```
    }
```

Mais uma coisa... lembre-se de que queremos saber quais workers estão trabalhando e quais estão ociosos. Para isso, vamos adicionar uma propriedade "idle" ao worker. Esta é nossa própria propriedade, não faz parte da API Web Worker. No momento, definimo-la para true, já que não demos nada para os workers fazerem.

```
    worker.idle = true;
```

E acrescentamos o worker que acabamos de criar no array de workers.

```
    workers.push(worker);
```

E, finalmente, em algum momento precisaremos fazer com que esses workers começem a trabalhar. Colocaremos esse código numa função chamada `startWorkers`, que precisamos escrever.

Esse código vai em `mandel.js`.

Pondo os workers para começar

Ok, temos algumas coisas para concluir: precisamos que os workers comecem a trabalhar, precisamos escrever a função que pode processar o trabalho que retorna dos workers e, bem, precisamos também escrever o código para o worker. Vamos começar por escrever o código para que os workers comecem:

- | | |
|-------------------------------------|---|
| <input checked="" type="checkbox"/> | Criando uma página HTML |
| <input checked="" type="checkbox"/> | Código Pronto para Assar |
| <input checked="" type="checkbox"/> | Criando Workers |
| <input type="checkbox"/> | Pondo os workers para começar |
| <input type="checkbox"/> | Implementando os workers |
| <input type="checkbox"/> | Processando os resultados |
| <input type="checkbox"/> | Testando a interação do código de usuário |

Estamos adicionando duas variáveis

globais a mais em mandel.js.



A primeira é nextRow, que rastreia em qual fileira estamos, à medida que avançamos na imagem.

```
var nextRow = 0;
var generation = 0;
```



Toda vez que o usuário dá um zoom na imagem Mandelbrot, iniciamos um novo cálculo nela. A variável de geração rastreia quantas vezes fizemos isso. Mais sobre isso já, já.

```
function startWorkers() {
    generation++;
    nextRow = 0;
```



A função startWorkers vai ativar os workers e também reativá-los, se o usuário der zoom na imagem. Então, cada vez que ativarmos os workers, reiniciaremos nextRow para zero e incrementaremos a geração.

```
for (var i = 0; i < workers.length; i++) {
    var worker = workers[i];
```

Como ambos são utilizados, ficará mais claro num instante...

```
if (worker.idle) {
```



Agora, nós passamos o loop em todos os workers no array de workers...

```
    var task = createTask(nextRow);
```

... e verificamos se o worker está ocioso.

```
    worker.idle = false;
```

Se estiver, daremos uma tarefa para o worker fazer. Essa tarefa é calcular uma fileira do Mandelbrot Set. createTask está definido em mandellib.js e ele retorna um objeto task com todos os dados que o worker precisa para calcular aquela fileira.

```
    worker.postMessage(task);
```



Agora, estamos prestes a dar algo para o worker fazer, então definimos a propriedade idle para false (ou seja, ocupado).

```
    nextRow++;
```

E aqui é onde diremos ao worker para começar a trabalhar, enviando-lhe uma mensagem contendo a tarefa. O worker está aguardando uma mensagem, então, quando recebe-la, recomeçará a trabalhar na tarefa.

```
}
```



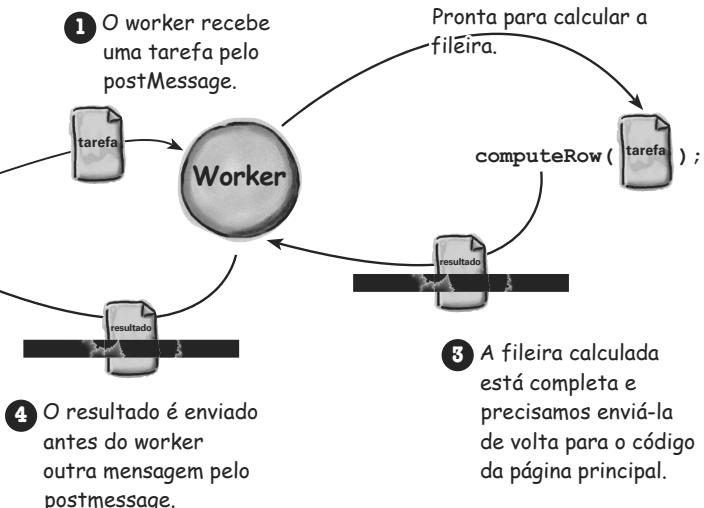
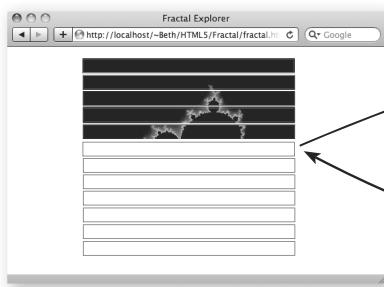
E, finalmente, incrementamos a fileira. Assim, o próximo worker ficará com a próxima fileira.

Este código vai em mandel.js.

Implementando o worker

Agora que temos o código para ativar nossos workers, dando-lhes tarefas, vamos escrever o código worker. Tudo o que precisaremos fazer será voltar e processar os resultados do worker, uma vez que ele calculou sua parte da imagem fractal. Contudo, antes de escrevermos o código vamos revisar rapidamente como ele deve funcionar:

- Criando uma página HTML
- Código Pronto para Assar
- Criando Workers
- Pondo os workers para começar
- Implementando os workers
- Processando os resultados
- Testando a interação do código de usuário



Então, vamos implementar isso: vá em frente e digite o seguinte código dentro de seu arquivo `worker.js`.

Estamos usando `importScripts` para importar Código Pronto de `workerlib.js`; assim, o worker pode chamar a função `computeRow` definida naquele arquivo de biblioteca.

```
importScripts("workerlib.js");
```

Tudo o que o worker faz é criar o handler `onmessage`. Ele não precisa fazer mais nada, pois o que ele faz é esperar por mensagens de `mandel.js` para começar a trabalhar!

```
onmessage = function (task) {
```

```
    var workerResult = computeRow(task.data);
```

```
    postMessage(workerResult);
```

```
}
```

O resultado do cálculo, salvo na variável `workerResult`, é enviado de volta ao JavaScript principal, usando `postMessage`.

Isso obtém os dados da tarefa e os passa à função `computeRow`, que faz o trabalho pesado do cálculo Mandelbrot.

Este código vai para `worker.js`.

Um pequeno pit stop...

Vimos muito código em tão poucas páginas. Vamos fazer uma pequena parada, reabastecer nossos tanques e estômagos.

Acreditamos que você talvez queira dar uma olhada rápida, por trás das câmeras, e ver como se parecem as tarefas e resultados do worker (eles são incrivelmente similares, como veremos). Então, pegue uma garrafa de salsaparrilha e vamos dar uma olhada enquanto descansa...



Tarefas de Perto

Você deu uma olhada na chamada para `createTask` e `postMessage`, que usa a tarefa:

```
var task = createTask(nextRow);
worker.postMessage(task);
```

E deve estar imaginando como se parece aquela tarefa. Bem, é um objeto feito de propriedades e valores. Vamos dar uma olhada:

A tarefa contém todos os valores que o worker precisa para fazer seu cálculo.

```
task = {
  row: 1,
  width: 1024,
  generation: 1,
  r_min: 2.074,
  r_max: -3.074,
  i: -0.252336,
  max_iter: 1024,
  escape: 1025
};
```

Identifica a fileira para a qual estamos criando os valores dos pixels.

Identifica a largura da fileira.

Identifica quantas vezes demos zoom. Veremos como isso é usado num instante...

Estes definem a área do Mandelbrot que estamos calculando.

E estes controlam a precisão daquilo que estamos calculando.



Resultados de Perto

E quanto aos resultados que obtemos do cálculo da fileira no worker?

```
var workerResult = computeRow(task.data);
postMessage(workerResult);
```

Como se parecem? Incrivelmente parecidos com a tarefa:

O worker pega a tarefa passada a ele, então adiciona uma propriedade values que contém os dados necessários para desenhar a fileira no canvas.

```
workerResult = {
  row: 1,
  width: 1024,
  generation: 1,
  r_min: 2.074,
  r_max: -3.074,
  i: -0.252336,
  max_iter: 1024,
  escape: 1025,
  values: [3, 9, 56, ... -1, 22]
};
```

Isso é tudo igual à tarefa.
Isso é ótimo, porque, quando a receberemos do worker, já sabemos tudo a respeito dela.

Ah, mas isso é novo. Esses são os valores de cada pixel, os quais ainda precisam ser mapeados com cores (o que acontece em drawRow).



Hora de voltar para a estrada...

Obrigado por despender um pouco de seu tempo conosco para verificar as tarefas e resultados. É melhor dar o último gole naquela salsaparrilha — estamos voltando para a estrada!



De volta ao código: como processar os resultados dos workers

Agora que você já viu os resultados do trabalho do worker, vejamos o que acontece quando os obtemos de volta. Lembre-se de que, quando criamos nossos workers, designamos um message handler chamado `processWork`:

```
var worker = new Worker("worker.js");

worker.onmessage = function(event) {
    processWork(event.target, event.data);
}
```

- Criando uma página HTML
- Código Pronto para Assar
- Criando Workers
- Pondos os workers para começar
- Implementando os workers
- Processando os resultados
- Testando a interação do código de usuário

Nosso message handler chama `processWork`, passando-lhe os dados do worker, e também do `target`, que é apenas uma referência ao worker que enviou os dados.

Quando um worker posta uma mensagem de volta para nós com seus resultados, é a função `processWork` que vai manipulá-la. Como pode ver, são passadas duas coisas: o target da mensagem, que é apenas uma referência ao worker que a mandou, e os dados da mensagem (é o objeto `task` com os valores para uma fileira da imagem). Portanto, nosso trabalho agora é escrever `processWork` (entre este código em `mandel.js`):

```
function processWork(worker, workerResults) {
    drawRow(workerResults);
    reassignWorker(worker);
}
```

Distribuímos os resultados para `drawRow` para desenhar os pixels no canvas.

Estamos quase lá. Vamos acabar logo com essa `reassignWorker`. Veja como funciona: verificamos a fileira que estamos calculando, usando nossa variável global `nextRow`, e, o que haja mais para calcular (desde que podemos determinar olhando para quantas fileiras estão em nosso canvas), damos ao worker uma nova designação. Do contrário, se não houver mais trabalho a ser feito, simplesmente definimos a propriedade `idle` do worker para `true`. Vá e entre este código em `mandel.js` também:

```
function reassignWorker(worker) {
    var row = nextRow++;
    if (row >= canvas.height) {
        worker.idle = true;
    } else {
        var task = createTask(row);
        worker.idle = false;
        worker.postMessage(task);
    }
}

Vamos dar a este worker a próxima fileira que precisa de cálculo. Obteremos o número de nextRow e incrementaremos nextRow (assim o próximo worker ficará com a próxima).
```

Se a fileira for maior ou igual à altura do canvas, acabou! Preenchemos o canvas inteiro com resultados dos workers do Mandelbrot Set.

Canvas é uma variável global que foi designada quando chamamos `setupGraphics` em nossa função `init`.

Se ainda temos fileiras a fazer, criamos uma nova tarefa para a próxima fileira. Certifique-se de que a propriedade `idle` de nosso worker seja `false` e poste uma mensagem com a nova tarefa para o worker.

Este código fica em `mandel.js`

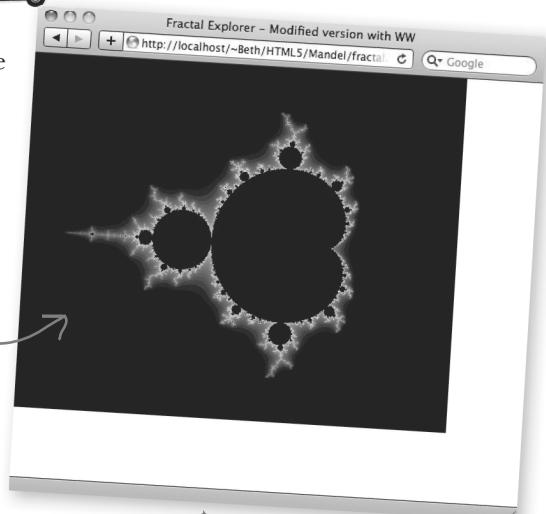
Test drive psicodélico



Chega de códigos! Vamos rodar um teste nesse negócio. Carregue o arquivo `fractal.html` em seu navegador e veja seus workers indo ao trabalho. Dependendo de sua máquina, seu Fractal Explorer deverá ser um pouco mais rápido do que antes.

Não escrevemos qualquer código para manipular o redimensionamento da janela de seu navegador ou, o clique para dar zoom no fractal, por sinal. Então, por ora, você verá a imagem à direita.

Até agora está bom, né?



Manipulando um click event

Nossos workers estão ocupados trabalhando no cálculo do Mandelbrot Set e retornando resultados para nós. Podemos desenhá-los no canvas, mas o que aconteceria se você clicasse para dar zoom? Felizmente, pelo fato de estarmos utilizando workers para o cálculo intenso no background, o UI deve estar esperto em lidar com seu clique. Dito isto, precisamos escrever um pequeno código para, de fato, manipular o clique. Veja como fazemos isso:

↑
Veja só! Que pena que não podemos dar zoom e pena ainda não preencher a janela toda, mas chegaremos lá...

- ① A primeira coisa que precisamos fazer é adicionar um handler para dar conta dos cliques do mouse e, lembre-se, os cliques estão acontecendo em nosso elemento canvas. Para isso, apenas acrescentamos um handler para a propriedade `onclick` do canvas, assim:

```
canvas.onclick = function(event) {
    handleClick(event.clientX, event.clientY);
};
```

- Criando uma página HTML
- Código Pronto para Assar
- Criando Workers
- Pondo os workers para começar
- Implementando os workers
- Processando os resultados
- Testando a interação do código de usuário

Se o canvas for clicado, chamaremos a função `handleClick` com a posição x, y do clique.

Acrescente este código acima para chamar `setUpGraphics` na função `init` de "mandel.js".

- ② Agora precisamos apenas escrever a função `handleClick`. Antes disso, vamos pensar sobre isto um segundo: quando um usuário clica no canvas, significa que ele quer dar zoom da área em que está clicando (você pode voltar à versão single-threaded em <http://wickedlysmart.com/hfhtml5/chapter10/singlethreaded/fractal.html> para observar este comportamento). Então, quando o usuário clicar, precisaremos obter as coordenadas de onde ele quer dar zoom e então pôr todos os workers para trabalhar na criação de uma nova imagem. Recorde também que já temos uma função para designar novo trabalho para qualquer worker ocioso: `startWorkers`. Vamos experimentar...

handleClick será chamada quando o usuário clicar no canvas para dar zoom no fractal.

```
function handleClick(x, y) {
    var width = r_max - r_min;
    var height = i_min - i_max;
    var click_r = r_min + width * x / canvas.width;
    var click_i = i_max + height * y / canvas.height;

    var zoom = 8;
```

Passamos a posição x, y do clique para que saibamos onde ele criou na tela.

Este código redimensiona a área do fractal que estamos calculando com a posição x, y no centro da nova área. Também ratifica que a nova área tem a mesma proporção daquela existente.

```
r_min = click_r - width/zoom;
r_max = click_r + width/zoom;
i_max = click_i - height/zoom;
i_min = click_i + height/zoom;
```

Aqui é onde definimos as variáveis globais que são usadas para criar tarefas para os workers: o nível de zoom determina a qual distância do zoom estamos no fractal, o que determina quais valores do Mandelbrot Set estão sendo calculados.

```
} startWorkers();
```

← Agora estamos prontos para reiniciar os workers.

Este código entra em mandel.js.

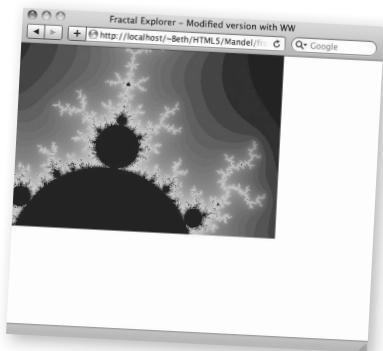
Outro test drive



Vamos experimentar essas mudanças no código. Recarregue `fractal.html` em seu navegador e, desta vez, clique em algum lugar no canvas. Quando o fizer, verá os workers começarem a trabalhar na visualização com zoom.

Ei, você deverá ser capaz de começar a explorar agora! Depois de brincar um pouco, vamos fazer algumas modificações finais para dar sequência a essas implementações.

Legal! Podemos dar zoom, mas ainda precisamos redimensionar o canvas para caber por completo em nossa janela.



Encaixando o canvas na janela do navegador

Queremos a imagem fractal preenchendo a janela do navegador, o que significa que precisamos redimensionar o canvas, se o tamanho da janela mudar. Não apenas isso, mas, se modificarmos o tamanho do canvas, deveremos também distribuir um novo conjunto de tarefas aos workers, para que eles possam redesenhar o fractal para preencher o novo tamanho do canvas. Vamos escrever o código para redimensionar o canvas ao tamanho da janela do navegador e também vamos reiniciar os workers.

- Criando uma página HTML
- Código Pronto para Assar
- Criando Workers
- Pondo os workers para começar
- Implementando os workers
- Processando os resultados
- Testando a interação do código de usuário

```
function resizeToWindow() {
    canvas.width = window.innerWidth;
    canvas.height = window.innerHeight;
    var width = ((i_max - i_min) * canvas.width / canvas.height);
    var r_mid = (r_max + r_min) / 2;
    r_min = r_mid - width/2;
    r_max = r_mid + width/2;
    rowData = ctx.createImageData(canvas.width, 1);

    startWorkers();
}

E, uma vez mais,
reiniciamos os workers.

    
```

resizeToWindow garante que a largura e a altura do canvas estejam definidas de acordo com as novas largura e altura da janela.

Também atualiza os valores que o worker usará para fazer seu cálculo baseado nas novas altura e largura (certificamo-nos de que o fractal sempre caberá no canvas e manterá a proporção da janela).

Há um detalhe administrativo que usa uma variável global de que não falamos a respeito: rowData. rowData é o objeto ImageData que estamos usando para desenhar pixels dentro de uma fileira do canvas. Então, quando redimensionarmos o canvas, precisaremos recriar o objeto rowData para que ele tenha a mesma largura que a nova largura do canvas. Cheque a função drawRow no mandel.js para ver como usamos rowData para desenhar pixels dentro do canvas.

Agora, precisamos fazer mais uma coisa: instalar resizeToWindow como um handler para o evento de redimensionamento da janela do navegador. Veja como faremos isso:

```
window.onresize = function() {
    resizeToWindow();
};
```

Coloque este código na função init de mandel.js, logo abaixo da chamada para setUpGraphics.

Esse código entra em mandel.js.

O codificador ~~chef~~ do Sr. Explicadinho

Há só mais uma coisa, e poderíamos deixar passar batido, mas o código não parece correto sem isso. Analise com a gente: você tem um bando de workers trabalhando felizes em suas fileiras e, de repente, o usuário clica na tela para dar zoom. Bem, isso não é ótimo, porque os workers trabalharam duro em suas fileiras e agora o usuário quer mudar a imagem inteira, tornando todo o trabalho inútil. Pior, os workers não possuem qualquer conhecimento de que o usuário clicou e eles vão retornar seus resultados mesmo assim. Ainda pior, o código na página principal vai, com todo o prazer, receber e mostrar aquela fileira! Sem querer dizer que é o fim do mundo, ou algo assim, temos exatamente o mesmo problema se o usuário redimensionar a janela.

Você provavelmente nunca percebeu nada disso porque não há tantos workers e eles calculam rapidamente as mesmas fileiras para a nova imagem, sobrescrevendo as fileiras incorretas anteriores. Isso, porém, está errado e é tão fácil de consertar que temos de fazê-lo.

Bom, temos uma confissão a fazer: sabíamos que chegariamos aqui. Talvez, você se recorde de uma pequena variável que nos deparamos chamada `generation`. Lembra? Toda vez que reiniciamos nossos workers, aumentamos o valor de `generation`. Lembre-se também do objeto `results` que retorna do worker: todo resultado tem seu “`generation`” como uma propriedade. Então, podemos usar `generation` para saber se temos um resultado da visualização anterior ou atual.

Vamos dar uma olhada no code fix e, então, poderemos conversar a respeito de como isso funciona; edite sua função `processWork` em `mandel.js` e acrescente essas duas linhas:

```
function processWork(worker, workerResults) {  
    if (workerResults.generation == generation) {  
        drawRow(workerResults);  
    }  
    reassignWorker(worker);  
}  
Em todo caso, pegamos o worker e o  
designamos a um novo trabalho!
```

Estamos verificando o resultado do worker para ver se seu `generation` combina com o atual.

Se combinhar, desenhemos a fileira; do contrário, deve ser antiga e a ignoramos.

Portanto, tudo o que estamos fazendo aqui é verificar para ter certeza de que o `generation` atual, no qual estamos trabalhando, combina com o `generation` do resultado, que retorna do worker. Se assim for, ótimo, precisamos desenhar a fileira. Se não, isso talvez signifique que seja velho, vamos apenas ignorar — é uma pena que nosso worker tenha desperdiçado seu tempo, mas não queremos desenhar uma fileira antiga, da imagem anterior, na tela.

Então, está na hora de verificar se digitou as modificações acima e preparar-se para...

Nota ao editor:
Perdão pela fala raivosa aqui, mas, olha, depois de tantas páginas, talvez chegue até você...

Hora do test drive final!



É isso! Você deve estar pronto para continuar com todo seu código. Carregue o arquivo `fractal.html` em seu navegador e veja seus workers indo para o trabalho. Esta versão deverá ser mais rápida e mais responsiva que a original e single-thread; se tiver mais de um core em seu computador, ficará *muito* mais rápido.

Divirta-se... dê zoom in... explore. Conte-nos se você encontrar qualquer “país” desconhecido no Mandelbrot Set (mande-nos um tuíte com screenshots para **#hfhtml5**, se quiser!).

- Criando uma página HTML
- Código Pronto para Assar
- Criando Workers
- Pondo os workers para começar
- Implementando os workers
- Processando os resultados
- Testando a interação do código de usuário

Clique, dê zoom, explore!

Redimensione sua tela em qualquer forma ou tamanho agora!

NO LABORATÓRIO

Se estiver escrevendo código de alta performance, você vai querer verificar como o número de workers pode causar impacto no runtime de seu aplicativo.

Para isso, pode usar o monitor de tarefas tanto no OS X quanto no Windows. Se voltarmos à nossa versão original (aquele single-thread em <http://wickedlysmart.com/hfhtml5/chapter10/singlthread/fractal.html>), nossa performance parecerá com o gráfico à direita.



Temos oito cores em nossa máquina. No Fractal Explorer com os Web Workers, definimos o número de workers para combinar com isso, com `numberOfWorkers = 8`. Você pode observar em nosso monitor de atividade que todos os oito cores estão sendo usados ao máximo.

O que acha que acontecerá se definirmos o número de workers para 2, ou 4, ou 16, ou 32? Ou algo no meio?

Experimente em sua máquina e veja quais valores funcionam melhor para você.



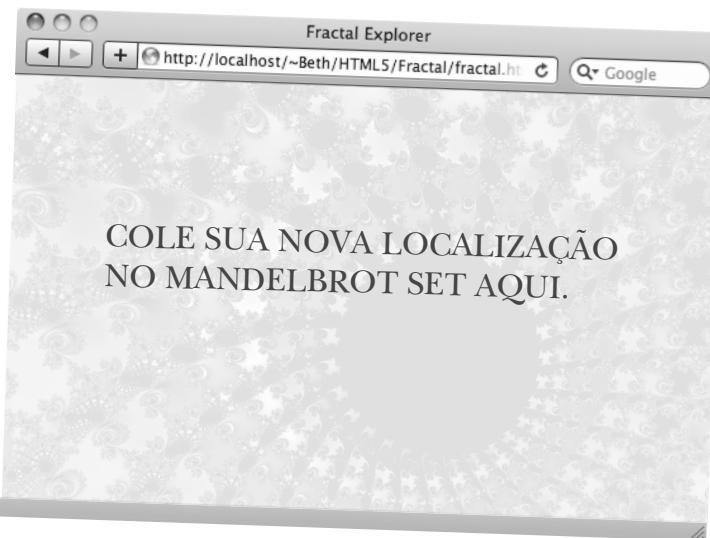
↗ Nossa máquina com oito cores. Um core está no máximo e não pode calcular mais nada. Os outros sete não estão fazendo nada para ajudar.

Agora nossos oito cores estão realmente trabalhando duro e o cálculo de nosso fractal está MUITO mais rápido.



REIVINDIQUE SEUS DIREITOS

Você conseguiu! Você tem um Fractal Explorer funcionando a pleno vapor, que está pronto para explorar o território Mandelbrot Set. Então, o que está esperando? Cave e encontre seu pequeno pedaço de universo virtual. Uma vez encontrado, imprima-o, cole-o aqui e dê um nome à sua nova pequena propriedade.



Nomeie seu novo território: _____



Agora, antes de ir embora, você acreditaria que tem muito mais coisa para conhecer a respeito dos Web Workers? Dê uma olhada nas próximas páginas para ver tudo o que não falamos durante este capítulo.



Finalize um worker

Você criou workers para realizarem uma tarefa, a tarefa está feita, agora você quer se livrar de todos eles (eles realmente ocupam memória valiosa no navegador). Você pode finalizá-los a partir do código em sua página principal, dessa forma:

```
worker.terminate();
```

Se acontecer de o worker ainda estar rodando, o script dele vai abortar. Então, use com cuidado. Uma vez que você tenha finalizado um worker, não mais poderá reutilizá-lo. Terá de criar um novo.

Você pode fazer com que um worker pare sozinho, chamando `close();` (de dentro do worker).

Handle errors nos workers

O que acontece se algo der muito errado num worker? Como você tira os bugs dele? Use o handler `onerror` para capturar quaisquer erros e também obter informação de debug, assim:

```
worker.onerror = function(error) {  
    document.getElementById("output").innerHTML =  
        "There was an error in " + error.filename +  
        " at line number " + error.lineno +  
        ": " + error.message;  
}
```

Use importScripts para fazer uma solicitação JSONP

Você não pode inserir novos elementos <script> para fazer solicitações JSONP a partir dos workers, mas você **pode** usar importScripts para fazer solicitações JSONP, assim:

```
function makeServerRequest() {
    importScripts("http://SomeServer.com?callback=handleRequest");
}

function handleRequest(response) {
    postMessage(response);
}

makeServerRequest();
```

Lembra do JSONP? Inclua sua função callback na query URL e ele será chamado com os resultados JSON passados ao parâmetro response.

Use setInterval em seus workers

Talvez não tenha prestado atenção nisso (passou muito rápido, usamos ele apenas em um exemplo), mas você pode usar setInterval (e setTimeout) em seus workers para executar a mesma tarefa repetidamente. Por exemplo, você poderia atualizar o worker de citações (quote.js) para postar uma citação aleatória a cada três segundos, assim:

```
var quotes = ["I hope life isn't a joke, because I don't get it.",
    "There is a light at the end of every tunnel...just pray it's not a
train!",
    "Do you believe in love at first sight or should I walk by again?"];
function postAQuote() {
    var index = Math.floor(Math.random() * quotes.length);
    postMessage(quotes[index]);
}
postAQuote();
setInterval(postAQuote, 3000);
```

Mova essas duas linhas para uma função postAQuote...

... chame postAQuote para enviar uma citação imediatamente e então defina um intervalo para enviar mais citações a cada três segundos.

Subworkers

Se seu worker precisa de ajuda com sua tarefa, ele pode criar seus próprios workers. Digamos que esteja dando regiões de uma imagem a seus workers para trabalharem. O worker poderia decidir que, se uma região é maior que algum tamanho, ele dividirá entre seus próprios subworkers.

Um worker cria subworker, assim como o código de sua página cria um worker, com:

```
var worker = new Worker("subworker.js");
```

Lembre-se que subworkers, assim como os workers, são bem pesados: eles ocupam memória e são rodados como threads separados. Então, seja cauteloso em relação a quantos subworkers você criará.



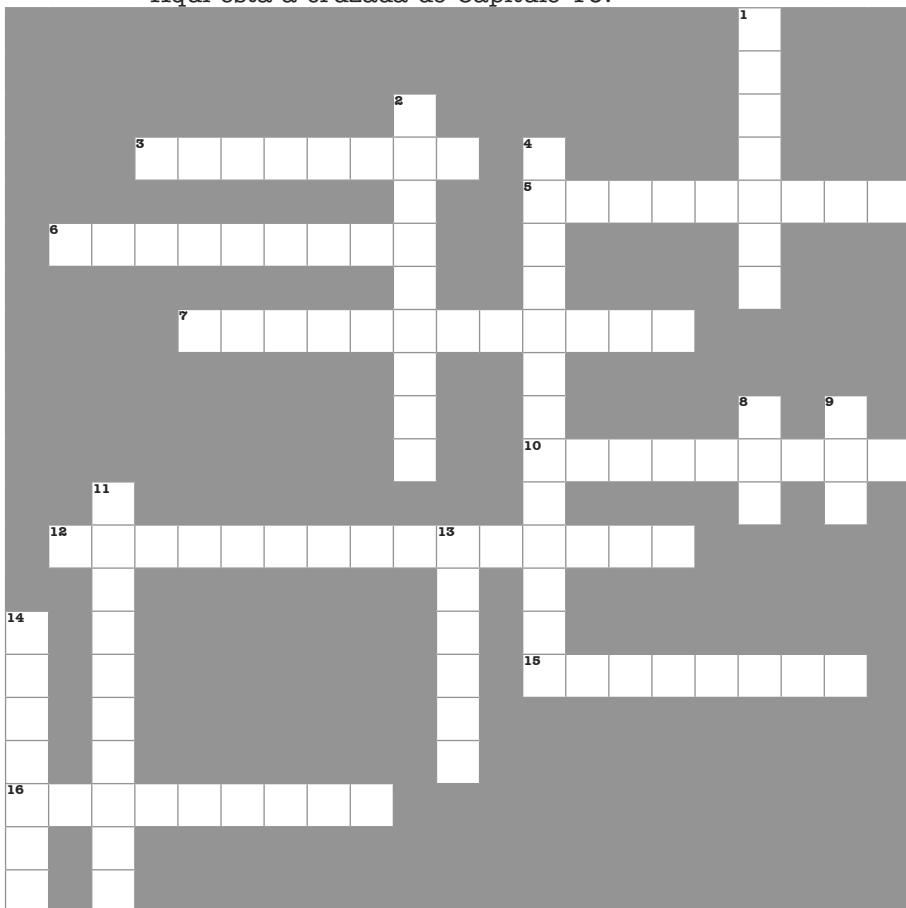
PONTOS DE BALA

- Sem os Web Workers, o JavaScript seria single-thread, o que significa que poderia fazer só uma coisa por vez.
- Se der a um programa JavaScript muita coisa para fazer, talvez veja a tela de diálogo de script lento.
- Os Web Workers manipulam tarefas num thread separado. Assim, o código principal de seu JavaScript pode continuar a rodar e seu UI permanece responsivo.
- O código para um Web Worker fica num arquivo separado do código de sua página.
- Os Web Workers não têm acesso a quaisquer funções do código de sua página ou do DOM.
- O código em sua página e o Web Worker se comunicam via mensagens.
- Para mandar uma mensagem a um worker, use postMessage.
- Você pode enviar strings e objetos para um worker via postMessage. Não se pode mandar funções para um worker.
- Receba de volta mensagens dos workers, definindo a propriedade onmessage do worker para uma função handler.
- Um worker recebe mensagens do código em sua página, definindo sua propriedade onmessage a uma função handler.
- Quando um worker está pronto para enviar de volta um resultado, ele chama postMessage e passa o resultado como o argumento.
- Os resultados do worker são encapsulados num objeto event e colocados na propriedade data.
- Você pode descobrir qual worker enviou a mensagem, usando a propriedade event.target.
- As mensagens são copiadas, não compartilhadas, entre o código de sua página principal e o worker.
- Você pode usar múltiplos workers para longos cálculos, que podem ser divididos em tarefas múltiplas, tal como calcular uma visualização fractal ou traçar o raio de uma imagem.
- Cada worker roda em seu próprio thread; portanto, se seu computador possui um processador multicore, os workers rodarão em paralelo, o que aumentará a velocidade do cálculo.
- Você pode finalizar um worker, chamando worker.terminate() a partir do código de sua página principal. Isto irá abortar o script do worker. Um worker pode também parar-se por conta própria chamando close().
- Os workers também possuem uma propriedade onerror. Você pode defini-la a uma função error handling, que será chamada se seu worker tiver um erro no script.
- Para incluir e usar as bibliotecas JavaScript em seu arquivo worker, use importScripts.
- Você também pode usar importScripts com JSONP. Implemente o callback que passou no query URL do arquivo worker.
- Embora os workers não tenham acesso ao DOM ou a funções em seu código principal, eles podem usar XMLHttpRequest e Local Storage.



Cruzada HTML5

Nossa, Capítulo 10, você conseguiu. Sente-se, relaxe e fixe as informações, trabalhando o resto do seu cérebro um pouco. Aqui está a cruzada do Capítulo 10.



HORIZONTAIS

3. Nossa primeiro exemplo usou este jogo.
5. O gerenciador e os workers se comunicam com isso.
6. Capacidade de um processador para fazer mais que uma coisa ao mesmo tempo.
7. Os workers podem usar XMLHttpRequest e acessar ____.
10. Mandelbrot usa números ____.
12. O cara que escreveu a versão original do Fractal Viewer.
15. Uma área agradável do interior de Mandelbrot é ____ Valley.
16. Como abortar um worker.

Ok, nos nem
chegamos a
dizer-ló, e James
Hensfridgé.

VERTICIAIS

1. ____/worker.
2. A propriedade usada para registrar um handler para receber mensagens.
4. Como importar código adicional para dentro de um worker.
8. Como criar um Worker.
9. Workers não podem ter acesso ao ____.
11. O fractal mais famoso.
13. ____ de execução.
14. Você pode passar ____ aos workers usando postMessage.



Sinta-se como o Navegador – Solução

Está na hora de fingir que você é o navegador avaliando o JavaScript.

```
window.onload = function() {  
    var worker = new Worker("worker.js");  
    worker.onmessage = function(event) {  
        alert("Worker says " + event.data);  
    }  
    for (var i = 0; i < 5; i++) {  
        worker.postMessage("ping");  
    }  
}
```

Este envia cinco mensagens Ping ao worker, que responde com cinco Pong; então, recebemos cinco alertas “Worker says Pong”.

```
window.onload = function() {  
    var worker = new Worker("worker.js");  
    worker.onmessage = function(event) {  
        alert("Worker says " + event.data);  
    }  
    for(var i = 5; i > 0; i--) {  
        worker.postMessage("pong");  
    }  
}
```

Este envia cinco mensagens Pong ao worker, que as ignora, já que não são Pings. Sem resposta.

```
window.onload = function() {  
    var worker = new Worker("worker.js");  
    worker.onmessage = function(event) {  
        alert("Worker says " + event.data);  
        worker.postMessage("ping");  
    }  
    worker.postMessage("ping");  
}
```

Este envia um ping e, então, cada vez que um pong retorna, envia mais um. Temos um loop infinito de alertas.

```
window.onload = function() {  
    var worker = new Worker("worker.js");  
    worker.onmessage = function(event) {  
        alert("Worker says " + event.data);  
    }  
  
    setInterval(pinger, 1000);  
  
    function pinger() {  
        worker.postMessage("ping");  
    }  
}
```

Este envia um ping a cada segundo. Assim recebemos de volta um pong toda vez que ele envia um ping.



Aponte o seu lápis Solução

Embora os workers tipicamente recebam suas ordens de trabalho por meio de mensagens, eles não precisam. Veja esta forma fácil e compacta de ter seu trabalho feito com os workers e com o HTML. Quando souber o que ele faz, descreva abaixo.

```
<!doctype html>
<html lang="en">
  <head>
    <title>Quote</title>
    <meta charset="utf-8">
  </head>
  <body>
    <p id="quote"></p>
    <script>
      var worker = new Worker("quote.js");
      worker.onmessage = function(event) {
        document.getElementById("quote").innerHTML = event.data;
      }
    </script>
  </body>
</html>

var quotes = ["I hope life isn't a joke, because I don't get it.",
              "There is a light at the end of every tunnel...just pray it's not a train!",
              "Do you believe in love at first sight or should I walk by again?"];
var index = Math.floor(Math.random() * quotes.length);

postMessage(quotes[index]);
```

quote.html

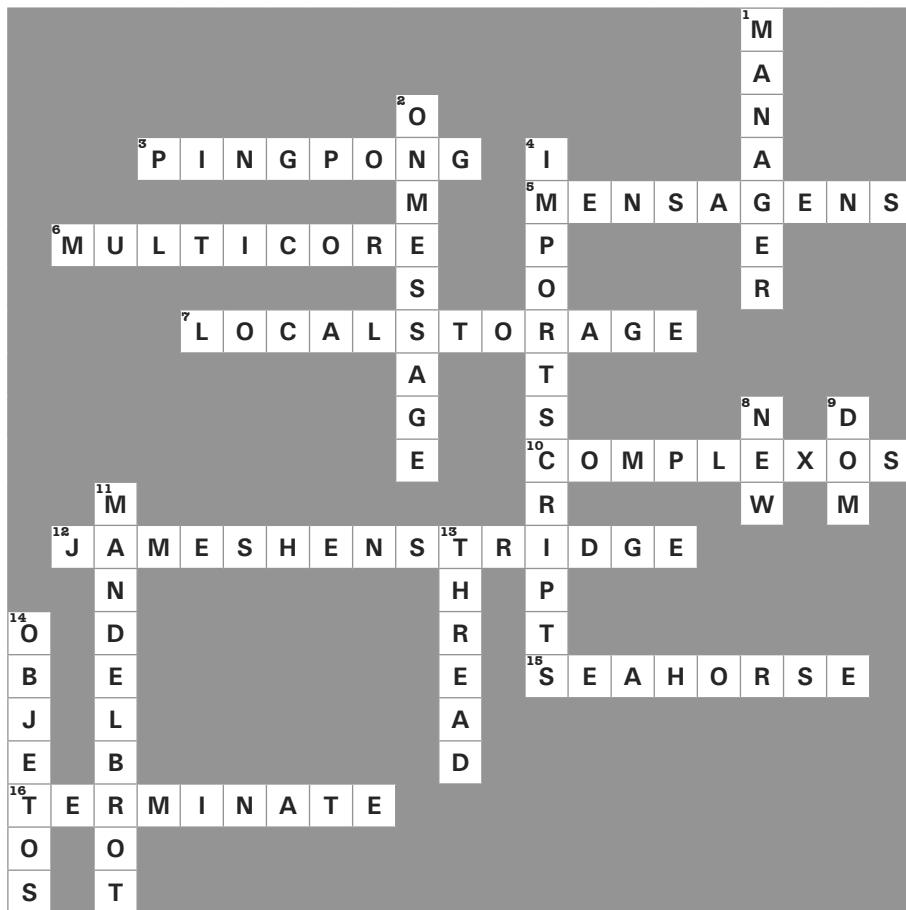
quote.js

Sua resposta aqui:

Em nosso HTML, temos um script que cria um worker, que executa imediatamente. O worker escolhe uma citação aleatoriamente a partir do array quotes e a envia ao código principal, usando postMessage. O código principal obtém a citação a partir de event.data e a adiciona à página no "quote" do elemento <p>.



Cruzada HTML5 – Solução





Não seria um sonho se este
fosse o fim do livro? Se não
houvesse pontos de bala ou
enigmas ou listas de JavaScript
ou qualquer outra coisa? Talvez
seja só uma fantasia...

Parabéns!
Você conseguiu.

É claro, há ainda um apêndice.

E o índice.

E o colofão.

E tem também o site...

Não tem escapatória, sério.

Apêndice: remanescentes

* Os dez tópicos mais importantes (que não vimos)



Vimos muita coisa e você já praticamente terminou este

livro. Sentiremos sua falta, mas, antes de deixarmos você ir, queremos lhe oferecer pelo mundo um pouco mais de preparação. Claro que não há como ensinar tudo o que você precisa saber nesse, relativamente, pequeno capítulo. Na verdade, nós, originalmente, *incluímos* tudo o que você precisava saber sobre HTML5, reduzindo o tamanho da fonte para .00004. Tudo caberia, mas ninguém conseguia ler. Então, jogamos a maior parte fora e mantivemos o melhor para este apêndice dos Dez Mais.

Este realmente é o fim do livro. Exceto pelo índice, é claro (de leitura necessária!).

Nº 1 Modernizr

Uma coisa que você provavelmente percebeu neste livro foi que, quando quiser detectar suporte do navegador para uma API, não há uma maneira uniforme de fazê-lo; de fato, quase toda API é detectada de uma maneira diferente. Para geolocalização, por exemplo, procuramos pelo objeto geolocalização como uma propriedade do objeto navigator; enquanto que para armazenamento web, vemos se o localStorage está definido no objeto window; para vídeo, vemos se podemos criar um elemento vídeo no DOM; e assim por diante. Será que há uma maneira melhor?

Modernizr é uma biblioteca JavaScript de fonte aberta que fornece uma interface uniforme para detectar suporte ao navegador. O Modernizr cuida de todos os detalhes dos diferentes meios de detecção, mesmo nos casos de navegadores mais antigos. Você encontrará a página do Modernizr em <http://www.modernizr.com/>

O Modernizr ganhou muito suporte de desenvolvedores. Por isso, você verá ele sendo usado amplamente pela Web. Recomendamos muito.

Incluindo Modernizr em sua página

Para usar o Modernizr, você precisa carregar a biblioteca JavaScript dentro de sua página. Para tanto, primeiro visite o site Modernizr em <http://www.modernizr.com/download/>, que permite que você configure de forma personalizada uma biblioteca que contém exatamente o código de detecção que precisa (ou pode sempre pegar tudo enquanto estiver lá). Depois disso, armazene a biblioteca num arquivo de sua escolha e carregue-o dentro de sua página (visite o site Modernizr para tutoriais adicionais e documentação para melhor utilização dele).

Como detectar suporte

Uma vez que tenha instalado o Modernizr, detectar elementos HTML5 e APIs JavaScript fica muito mais fácil e mais direto:

Aqui temos um exemplo de detecção para geolocalização, armazenamento web e vídeo, todos de uma maneira consistente.

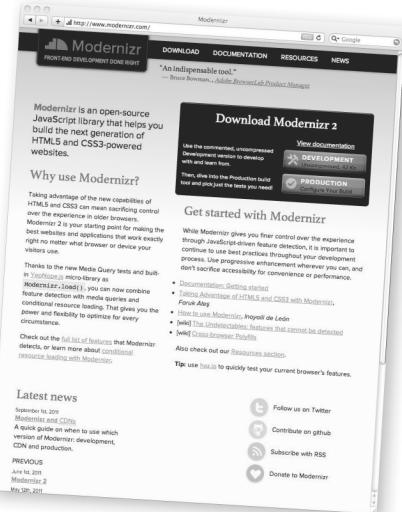
Nota: o Modernizr vai muito além do que simples detecção de API e pode também detectar suporte para recursos CSS, codecs de vídeo e muitos outros. Então, dê uma olhada!



```
if (Modernizr.geolocation) {
    console.log("You have geo!");
}

if (Modernizr.localstorage) {
    console.log("You have web storage!");
}

if (Modernizr.video) {
    console.log("You have video!");
}
```



Nº 2 Áudio

O HTML5 lhe dá uma maneira padrão de reproduzir áudio em suas páginas, sem um plugin, com o elemento `<audio>`:

```
<audio src="song.mp3" id="boombox" controls>
    Sorry but audio is not supported in your navegador.
</audio>
```

Parece-lhe familiar?
Sim, áudio suporta
funcionalidades
similares às do vídeo
(menos o vídeo, óbvio!).

Além do elemento `<audio>`, também há uma API Áudio correspondente que suporta os métodos que você espera, como `play`, `pause` e `load`. Se isso lhe parece familiar, deve ser mesmo, pois a API áudio espelha (onde é apropriado) a API vídeo. Áudio também suporta muitas das propriedades que você viu na API vídeo, como `src`, `currentTime` e `volume`. Aí vai um pouco de código de áudio, para você ter um gostinho de usar a API com um elemento na página:

```
var audioElement =
    document.getElementById("boombox");
audioElement.volume = .5;
audioElement.play();
```

Pegue uma referência ao elemento áudio; então,
abaixe seu volume pela
metade e comece a tocar.

Assim como para vídeo, cada navegador implementa sua própria aparência e sensação para os controles do player (o que, tipicamente, consiste em uma barra progressiva com controles de `play`, `pause` e `volume`).

Apesar da simples funcionalidade, o elemento áudio e a API lhe dão muito controle. Assim como fizemos com vídeo, você pode criar experiências web interessantes, escondendo os controles e gerenciando a reprodução do áudio em seu código. Com o HTML5, você agora pode fazer isso sem a preocupação de ter de usar (e aprender) um plugin.

Um padrão para codificações de áudio

Infelizmente, como o vídeo, não há padrão para codificação do áudio. Três formatos são populares: mp3, wav e Ogg Vorbis. Você descobrirá que o suporte para esses formatos varia de acordo com a paisagem do navegador, com diferentes níveis de suporte para os vários formatos em cada navegador (na época em que este livro foi escrito, o Chrome era o único navegador que suportava os três formatos).



Nº 3 jQuery

jQuery é uma biblioteca JavaScript voltada para reduzir e simplificar muito o código e a sintaxe JavaScript, que é necessária para trabalhar com o DOM. Use Ajax e adicione efeitos visuais às suas páginas. O jQuery é uma biblioteca muito popular, que é amplamente utilizada e expansível para outros modelos plugin.

Agora, não há nada que possa fazer no jQuery que não consiga no JavaScript (como dissemos, o jQuery é apenas uma biblioteca JavaScript). Contudo, possui, sim, o poder de reduzir a quantidade de código que precisa escrever.

A popularidade do jQuery fala por si só, embora demore um pouco para se acostumar, se você for novo com ele. Vamos dar uma olhada em algumas coisas que você pode fazer com o jQuery e nós o encorajamos a dar uma olhada mais de perto, se acha que ele pode ser para você.

Para os iniciantes: você se lembra de todas as funções de window onload que escrevemos neste livro? Tipo:

```
window.onload = function() {
    alert("the page is loaded!");
}
```

Lembre-se que Ajax é apenas um nome para usar o XMLHttpRequest, como fizemos no Capítulo 6.

Um conhecimento

profissional do jQuery é uma boa habilidade hoje em dia no mercado de trabalho e mesmo para a compreensão de outros códigos.

Aqui é a mesma coisa usando o jQuery:

```
$(document).ready(function() { ← Igual à nossa versão, quando o documento
    alert("the page is loaded!");
});
```

Ou você pode diminuir isso ainda mais, para:

```
$(function() { ← Isso é legal, mas como pode
    alert("the page is loaded!");
});
```

Então, que tal obter elementos do DOM? É onde o jQuery brilha. Digamos que você tenha uma âncora em sua página com uma id de “buynow” e queira designar um click handler para o click event naquele elemento (como fizemos algumas vezes nesse livro). Veja como faremos:

```
$(function() { ← O que está acontecendo? Primeiro, estamos criando uma
    $("#buynow").click(function() { função que é chamada quando a página é carregada.
        alert("I want to buy now!"); });
});
```

Depois, pegamos a âncora com uma id “buynow” (perceba que o jQuery usa a sintaxe CSS para selecionar elementos).

Então, chamamos um método jQuery. Clique no resultado para definir o handler onclick.

Isso é só o começo; podemos facilmente definir o click handler em *todas as âncoras* da página:

```
$(function() {
  $("a").click(function() {
    alert("I want to buy now!");
  });
});
```

Para isso, tudo o que precisamos fazer é usar o nome da tag. Compare isso com o código que você escreveria para fazer o mesmo, usando o JavaScript sem o jQuery.

Ou, podemos fazer coisas que são muito mais complexas:

```
$(function() {
  $("#playlist > li").addClass("favorite");
});
```

Como encontrar todos os elementos `` que são filhos do elemento com uma id de lista de reprodução.

Então, adicione-os à classe "favorite".

Na verdade, isso já é o jQuery apenas se aquecendo; o jQuery pode fazer coisas muito mais sofisticadas do que isso.

Há um lado completamente diferente do jQuery que lhe permite fazer transformações de interface interessantes em seus elementos, assim:

```
$(function() {
  $("#specialoffer").toggle(function() {
    $(this).animate({ backgroundColor: "yellow" }, 800);
  }, function() {
    $(this).animate({ backgroundColor: "white" }, 300);
  });
});
```

Isso alterna o elemento com uma id de `specialoffer` entre ser amarelo com 800 pixels de largura, e branco com 300 pixels de largura, e ainda anima a transição entre os dois estados.

Como pode ver, há muita coisa para se fazer com o jQuery e ainda nem falamos sobre como podemos usá-lo para falar com serviços web ou com todos os plugins que funcionam com ele. Se estiver interessado, a melhor coisa a fazer é apontar seu navegador para <http://jquery.com/> e dar uma olhada nos tutoriais e documentações de lá.

E dê uma olhada também no `Use a Cabeça!` jQuery.

Nº 4 XHTML está morto, vida longa ao XHTML

Fomos bem duros com o XHTML neste livro, primeiro com a discussão “XHTML está morto”, depois com “JSON versus XML”. A verdade é que, quando se trata de XHTML, apenas o XHTML 2 e posteriores que morreram e, de fato, você pode escrever seu HTML5 usando estilo XHTML, se quiser. Por que iria querer? Bem, talvez precise validar ou transformar seus documentos para XML ou talvez queira suporte a tecnologias XML, como SVG (ver Nº 5), que funciona com HTML.

Vejamos um simples documento XML e então passaremos pelos pontos principais (não poderíamos nunca passar por tudo que precisa saber neste tópico, assim como todas as coisas do XML; fica complicado, rápido).

```
<!DOCTYPE html>           ← Mesmo doctype!
<html xmlns="http://www.w3.org/1999/xhtml">   ← Este é o XML, precisamos
    <head>                                de um namespace!
        <title>You Rock!</title>
        <meta charset="UTF-8" />           ← Todos os elementos têm de ser
    </head>                                formados; note o caminho /> aqui
    <body>
        <p>I'm kinda liking this XHTML!</p>
        <svg xmlns="http://www.w3.org/2000/svg">
            <rect stroke="black" fill="blue" x="45px" y="45px"
                  width="200px" height="100px" stroke-width="2" />
        </svg>
    </body>
</html>
```

← Podemos embutir XML bem na página! Legal, né?

Estamos usando SVG para desenhar um retângulo dentro de nossa página. Verifique o Nº 5 (próxima página) para mais sobre SVG.

Agora, veja algumas coisas que você precisa considerar para suas páginas XML:

- Sua página precisa estar bem formada com XML.
- Sua página deverá ser servida pelo MIME type application/xhtml+xml e, para isso, você vai precisar garantir que o seu servidor esteja servindo este tipo (ou leia tudo sobre isso ou conte com seu administrador do servidor).
- Certifique-se e inclua o namespace XHTML em seu elemento `<html>` (o que fizemos logo acima).

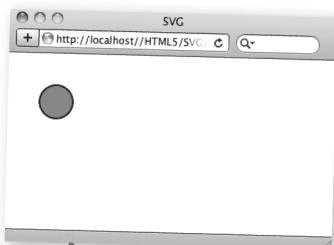
Fechando todos os seus elementos, aspas em volta dos valores do atributo, encaixes válidos de elementos e tudo mais.

Como dissemos, com o XML, há muita coisa para saber e muitas coisas para ficar atento. Como sempre, com o XML, que a força esteja com você...

Nº 5 SVG

Scalable Vector Graphics, ou SVG, é outra maneira — sem contar o canvas — de incluir gráficos de forma nativa em suas páginas. SVG já está por aí há algum tempo (desde mais ou menos 1999) e é hoje suportado em todas as versões atuais dos maiores navegadores, incluindo o IE9 e posteriores.

Diferente do canvas, que, como você sabe, é um elemento que lhe permite desenhar pixels dentro de uma superfície de desenho bitmap em sua página com JavaScript, os gráficos SVG são especificados com XML. “XML?” você diz. Sim, XML! Você cria elementos que representam gráficos e, então, pode combinar esses elementos de formas complexas para fazer cenas com gráficos. Vejamos um exemplo simples SVG:



The screenshot shows a web browser window with the URL `http://localhost//HTML5/SVG`. Inside the browser, there is a single gray circle centered at the top of the page.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>SVG</title>
  <meta charset="utf-8" /> Estamos usando um
</head> elemento <svg> bem
<body> no nosso HTML!
  <div id="svg">
    <svg xmlns="http://www.w3.org/2000/svg">
      <circle id="circle"
        cx="50" cy="50" r="20"
        stroke="#373737" stroke-width="2"
        fill="#7d7d7d" />
    </svg>
  </div>
</body>
</html>
```

Estamos usando o HTML5 estilo XHTML porque estamos usando SVG que é baseado em XML.

Estamos usando um elemento `<svg>` bem no nosso HTML!

Nosso SVG é simples: contém apenas um círculo que é localizado na posição $x=50, y=50$ e possui raio de 20...

... um contorno
que tem 2 pixels
de largura e
colorido com
cinza escuro...

Você pode pegar esse elemento círculo, como qualquer outro elemento do DOM, e fazer coisas com ele, por exemplo:
você poderia adicionar
um click handler e
mudar o atributo de
preenchimento do círculo
para “red”, quando o
usuário clicar no círculo.

O SVG define uma variedade de formas básicas, como círculos, retângulos, polígonos, linhas e assim por diante. Se tiver formas mais complexas para desenhar, você também poderá especificar caminhos com SVG — é claro, à esta altura as coisas ficam mais complicadas (como já viu com os caminhos em canvas). No entanto, existem editores gráficos que irão lhe permitir desenhar uma cena e exportá-la como SVG, poupando dores de cabeça para tentar descobrir todos aqueles caminhos por si só!

O que há de tão legal a respeito do SVG? Bem, um aspecto legal do SVG é que você pode dimensionar seus gráficos como quiser (grandes ou pequenos), que eles não ficam pixelados, como uma imagem jpeg ou png ficaria. Isso os torna fáceis de serem reutilizados em diferentes situações. Por ser o SVG especificado com texto, os arquivos SVG podem ser buscados, indexados, programados e comprimidos.

Mal tocamos a superfície de tudo o que se pode fazer com o SVG. Então, explore mais, se for de seu interesse.

Nº 6 Aplicativos web offline

Se você tem um smartphone ou tablet, provavelmente acessa a internet em movimento e, com as redes WiFi e de celular, você está conectado quase que a todo tempo. No entanto, como faz naquelas horas em que não está? Não seria ótimo se pudesse continuar a usar aqueles incríveis aplicativos HTML5 que você mesmo construiu?

Bem, agora você pode. Aplicativos web offline são suportados por todos os navegadores móveis e de desktop modernos (com uma exceção: IE).

Então, como tornar seu aplicativo web disponível offline? Você cria um arquivo *cache manifest*, que contém uma lista de todos os arquivos que seu aplicativo precisa para funcionar; o navegador fará o download de todos os arquivos e alternará para os arquivos locais, se e quando seu dispositivo ficar offline. Para dizer à sua página que ela tem um arquivo manifest, você simplesmente adicionará o filename do arquivo cache manifest em sua tag <html>, assim:

```
<html manifest="notetoself.manifest">
```

Veja o que o arquivo notetoself.manifest contém:

```
CACHE MANIFEST ← Todo arquivo cache manifest
CACHE:           deve começar com isso.
{               }
notetoself.html }   ← Liste todos os arquivos que quiser
notetoself.css  para armazenar na seção CACHE:
notetoself.js   html, css, JavaScript, imagens etc.
```

Este arquivo diz: quando você visita a página que aponta para este arquivo, baixe todos os arquivos listados na seção CACHE do arquivo. Você também pode adicionar duas outras seções ao arquivo: Fallback e Network. Fallback especifica qual arquivo usar se tentar acessar um arquivo que não esteja armazenado e Network especifica arquivos que nunca deveriam ser armazenados (por exemplo, recursos de rastreamento de visitas).

Agora, antes de sair para brincar com isso, você precisa saber duas coisas: primeiro, precisa ter certeza de que seu servidor web esteja configurado para servir o mime type para arquivos cache manifest corretamente (assim como tivemos de fazer com os arquivos de vídeo no Capítulo 8). Por exemplo, num servidor Apache, adicione esta linha em seu arquivo .htaccess no nível mais alto de seu diretório:

```
AddType text/cache-manifest .manifest
```

Outra coisa que precisa saber é que testar aplicativos web offline é complicado! Recomendamos que dê uma olhada numa boa referência sobre o assunto e leia as especificações de aplicativos web offline para HTML5.

Uma vez que você saiba o básico sobre armazenamento em cache, pode usar JavaScript para ser notificado sobre cache events, tal como quando um arquivo cache manifest é atualizado e o status da cache. Para ser notificado sobre eventos, você deve adicionar event handlers ao objeto window.applicationCache, assim:

```
window.applicationCache.addEventListener("error", errorHandler, false);
```



↑ Com aplicativos web offline, você pode usar seus aplicativos web favoritos quando não estiver conectado!

Implemente o errorhandler para ser notificado se houver algum erro com a cache.

Nº 7 Web Sockets

Vimos duas maneiras de se comunicar neste livro: XMLHttpRequest e JSONP. Em ambos os casos, usamos um modelo solicitação/resposta baseado em HTTP. Isto é, usamos o navegador para fazer uma solicitação à página inicial, CSS e JavaScript, e, cada vez que precisávamos de algo mais, fazíamos outra solicitação XMLHttpRequest ou JSONP. Até fizemos solicitações, quando não havia mais dados novos para nós, o que aconteceu algumas vezes com o exemplo Mighty Gumball.

O Web Sockets é uma nova API que lhe permite manter uma conexão aberta com um serviço web, de forma que, a qualquer hora que um novo dado estiver disponível, o serviço pode simplesmente enviá-lo a você (e seu código pode ser notificado). Pense nisso como uma linha de telefone aberta entre você e o serviço.

Eis uma visão geral de como usá-lo: primeiro, para criar um web socket utilizamos o construtor de web socket:

```
var socket = new WebSocket("ws://yourdomain/yourservice");
```

Perceba que esta URL usa o protocolo ws, não o protocolo http.

Você pode ser notificado assim que o socket estiver aberto com o evento open, para o qual você pode designar um handler:

```
socket.onopen = function() {
    alert("Your socket is now open with the web service");
}
```

E lembre-se de que ou você ou outra pessoa terá de escrever o código do servidor para que tenha com que falar!

Você pode enviar uma mensagem ao serviço web com o método postMessage:

```
socket.postMessage("player moved right");
```

Aqui fornecemos um handler que é chamado quando o socket está plenamente aberto e pronto para comunicação.

E para receber mensagens você registra outro handler, assim:

```
socket.onmessage = function(event) {
    alert("From socket: " + event.data);
};
```

Aqui, somos nós enviando uma string ao servidor; o sistema binário está chegando, mas ainda não é amplamente suportado.

Ao registrar um handler, recebemos todas as mensagens que estão contidas na propriedade data do evento.

Há mais coisa sobre isso, é claro, e você vai querer dar uma olhada em alguns tutoriais online, mas não tem muito mais coisa na API. Esta API meio que ficou para trás em relação ao desenvolvimento de outras API HTML5. Então, verifique os últimos guias de compatibilidade do navegador, antes de usá-la num projeto grande.



Nº 8 Mais API canvas

Nós nos divertimos com o canvas no Capítulo 7, construindo nosso projeto TweetShirt. Há ainda muito mais coisas divertidas que você pode fazer com o canvas e queremos falar sobre algumas aqui.

Mencionamos rapidamente que você pode `save` (salvar) e `restore` (restaurar) o contexto canvas. Por que iria querer fazer isso? Digamos que você defina algumas propriedades do contexto, como `fillStyle`, `strokeStyle`, `lineWidth` e assim por diante, e queira mudar temporariamente esses valores para fazer uma coisa, como desenhar uma forma, mas sem precisar reiniciar todos eles para voltar aos valores da propriedade que tinha anteriormente. Você pode usar os métodos `save` e `restore` para fazer isso:

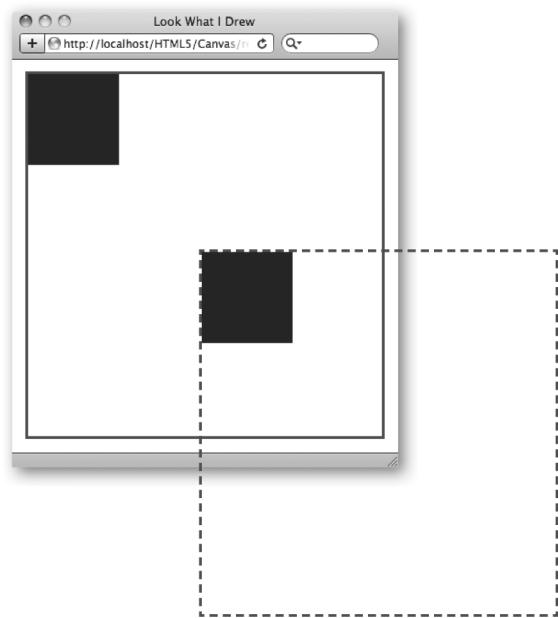
```
context.fillStyle = "lightblue";  
...  
context.save();  
context.fillStyle = "rgba(50, 50, 50, .5)";  
context.fillRect(0, 0, 100, 100);  
context.restore();  
...  
...
```

Criamos uma porção de propriedades no contexto e fizemos alguns desenhos.

Agora, salvamos o contexto. Todas aquelas propriedades são salvas de forma segura. Podemos modificá-las...

... e então colocamos todas elas de volta onde pertenciam quando as salvamos, simplesmente chamando o método `restore`! Neste ponto, todas as nossas propriedades estão como eram antes de as salvarmos.

Estes métodos tornam-se particularmente úteis quando você quer `translate` (traduzir) ou `rotate` (girar) o canvas para desenhar algo e então voltar para sua posição original. O que fazem os métodos `translate` e `rotate`? Vamos dar uma olhada...

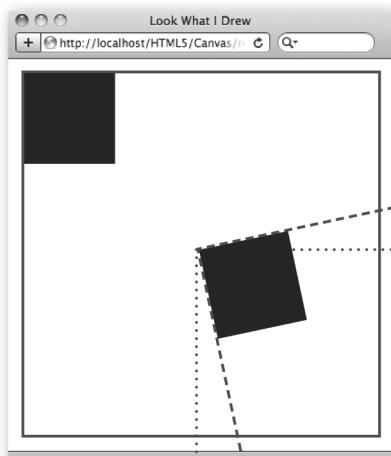


- ① Temos um canvas de 400x400 na página. Se desenharmos um retângulo preto em $x=0, y=0$, o desenho aparecerá no canto esquerdo superior, como esperado.

```
context.fillRect(0, 0, 100, 100);
```

- ② Agora, pegaremos o canvas e vamos movê-lo a 200 pixels para a direita e 200 pixels para baixo. Se desenharmos outro retângulo em $x=0, y=0$, o retângulo será desenhado a 200 pixels à direita e abaixo dos outros retângulos. Acabamos de traduzir o canvas.

```
context.translate(200, 200);  
context.fillRect(0, 0, 100, 100);
```



③ E se girarmos o canvas antes de desenharmos o retângulo? O canvas gira em torno de seu canto superior esquerdo (por padrão) e, já que acabamos de mover o canto superior esquerdo a 200, 200, essa será a posição em que o canvas vai girar.

```
context.translate(200, 200);
context.rotate(degreesToRadians(36));
context.fillRect(0, 0, 100, 100);
```

Quando você traduz ou gira o canvas, ele é movido numa grade que é posicionada em relação ao canto esquerdo superior da janela do navegador. Se você posicionou seu canvas usando CSS, esses valores são levados em conta. Experimente!

Agora vamos juntar tudo! Você pode usar os métodos `translate` e `rotate` juntos para criar alguns efeitos interessantes.

```
var canvas = document.getElementById("canvas");
var context = canvas.getContext("2d");
var degrees = 36;
context.save();
context.translate(200, 200);
context.fillStyle = "rgba(50, 50, 50, .5)";
for (var i = 0; i < 360/degrees; i++) {
    context.fillRect(0, 0, 100, 100);
    context.rotate(degreesToRadians(degrees));
}
context.restore();
```

Estamos salvando o contexto aqui, assim poderemos facilmente restaurá-lo à sua posição normal na grade, depois de terminarmos.

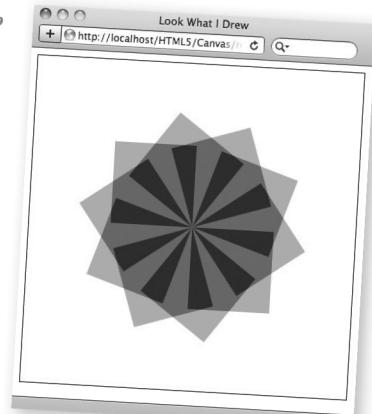
Traduzimos nosso canvas para 200, 200.

Estamos desenhandando 10 retângulos, girando o canvas 36 graus antes de desenhar um retângulo a 0, 0 cada vez, pelo loop.

E aqui está o resultado. Divertido!

Agora nosso canvas está de volta à sua posição original!

Combine essas simples transformações com outros métodos ainda mais poderosos (e complexos!), como compositing e transforms, e as possibilidades de criar gráficos artísticos com canvas serão infinitas.



Nº 9 Seletores API

Você já sabe como selecionar elementos do DOM, usando `document.getElementById`; usamos isso por todo o livro como uma maneira para o HTML e o JavaScript trabalharem juntos. Você também viu como usar `document.getElementsByTagName` (este método retorna um array de todos os elementos que combinam com uma tag) e há até um método `getElementsByClassName` (retornando, você adivinhou, todos os elementos que existem numa dada classe).

Com o HTML5, temos agora uma nova maneira de selecionar elementos do DOM, inspirados pelo jQuery. Você pode agora usar o mesmo seletor que usa no CSS para selecionar elementos de estilização em JavaScript, para selecionar elementos do DOM com o método `document.querySelector`.

Suponhamos que temos este simples HTML:

```
<!doctype html>
<html lang="en">
<head>
  <title>Query selectors</title>
  <meta charset="utf-8">
</head>
<body>
  <div class="content">
    <p id="avatar" class="level5">Gorilla</p>
    <p id="color">Purple</p>
  </div>
</body>
</html>
```

Veja mais de perto a estrutura deste HTML. Vamos usar os seletores API para selecionar elementos da página.



Temos um elemento `<div>` com a classe “`content`” e dois elementos, cada um com suas próprias ids, e um com a classe “`level5`”.

Agora, vamos usar a API `selectors` para pedir o elemento `<p>` “`avatar`”:

```
document.querySelector("#avatar"); ↴
```

É, essencialmente, a mesma coisa que o `document.getElementById("avatar")`. Agora, vamos usar a classe de elemento para selecioná-lo:

```
document.querySelector("p.level5"); ↴ Agora estamos usando o nome da tag e a classe para selecioná-lo.
```

Podemos também selecionar um elemento `<p>`, que é um filho do elemento `<div>`, assim:

```
document.querySelector("div>p");
```

↳ Aqui, estamos usando um seletor filho para selecionar o elemento `<p>` que é um filho da `<div>`. Ele seleciona o primeiro, por padrão.

ou dessa forma:

```
document.querySelector(".content>p"); ↴
```

Se o que realmente queremos são *todos* os elementos `<p>` na `<div>`, podemos usar o outro método na API `selectors`, `querySelectorAll`:

```
document.querySelectorAll("div>p"); ↴ Agora, pegamos todos os elementos filho <p> da <div>!
```

`querySelectorAll` retorna um array de elementos, assim como `getElementsByTagName`. É isso! Esses são os únicos dois métodos na API. A API `selectors` é pequena, mas adiciona uma nova e poderosa funcionalidade para selecionar elementos.

Nº 10 Tem muito mais!

Ok, realmente queríamos deixar isso para as dez coisas que não lhe contamos, mas parece que temos de ir e, em vez de ficarmos entre você e sua leitura do índice, vamos oferecer-lhe muito mais numa única página. Aqui estão eles (tenha em mente que algumas dessas áreas ainda estão em evolução, mas imaginamos que iria querer saber a respeito delas para referências futuras):

Indexed Database API e Web SQL

Se estiver procurando por algo mais industrial que a API Web Storage para armazenar seus dados localmente, fique de olho no espaço de web database. Duas visões concorrentes estão por aí agora mesmo: Web SQL e IndexedDB. Ironicamente, o Web SQL é o mais amplamente suportado dos dois, mas foi recentemente menosprezado pelos órgãos de padronização (o que significa que eles não recomendam adotá-lo como um padrão e, provavelmente, você não deveria basear seu próximo projeto nele!). O IndexedDB, por outro lado, ainda não é amplamente implementado, mas possui suporte do Google e do Firefox. O IndexedDB fornece acesso rápido a uma grande coleção de dados indexados, enquanto que o Web SQL é um pequeno motor SQL que roda no navegador. Fique de olho para onde vão essas tecnologias; elas estão mudando rapidamente!

Arraste e Solte

Os desenvolvedores web têm arrastado e soltado com jQuery já há algum tempo e, agora, essa funcionalidade é nativa no HTML5. Com a API Drag and Drop do HTML5, você especifica algo para arrastar onde possa soltá-lo e os handlers JavaScript são notificados pelos vários eventos que ocorrem, enquanto são arrastados e soltos. Para tornar um elemento arrastável, apenas defina o atributo `draggable` para `true`. Quase todos os elementos podem ser arrastados: imagens, listas, parágrafos etc. Você pode personalizar o comportamento do arrastar, ao ouvir os eventos como `dragstart` e `dragend`, e até mesmo mudar o estilo de um elemento para ficar como você quiser, enquanto estiver sendo arrastado. Pode enviar um pouco de dados com seu elemento arrastado, usando a propriedade `dataTransfer`; acesse isso pelo objeto `event` para saber se, digamos, o elemento está sendo movido ou copiado. Como pode ver, há uma porção de grandes oportunidades para construir novas interações UI com o Drag and Drop do HTML5.

Cross-document Messaging

No capítulo 6, usamos um padrão de comunicação conhecido como JSONP para dar um jeito nos problemas de comunicação interdomínios com o XMLHttpRequest. Há uma outra maneira com a qual você pode se comunicar entre documentos — mesmo documentos de diferentes domínios. A API Cross-Document Messaging especifica que você pode mandar uma mensagem a um documento que tenha carregado, usando um elemento `iframe`. Este documento poderia estar até num domínio diferente! Agora, você não vai querer carregar *qualquer* documento dentro de seu `iframe`; vai querer ter certeza de que vem de um domínio que você confia e configurá-lo para receber suas mensagens. O resultado é que esta é uma maneira de enviar e receber mensagens entre dois documentos HTML.

E poderíamos continuar...

O mais legal sobre o HTML5 é que há muitas novas capacidades sendo desenvolvidas a uma velocidade espantosa; há ainda muito mais que poderíamos colocar nesta página, mas não temos mais espaço. Então, mantenha-se informado conosco em nossa página <http://wickedlysmart.com> (em inglês) para todas as novidades sobre o HTML5!

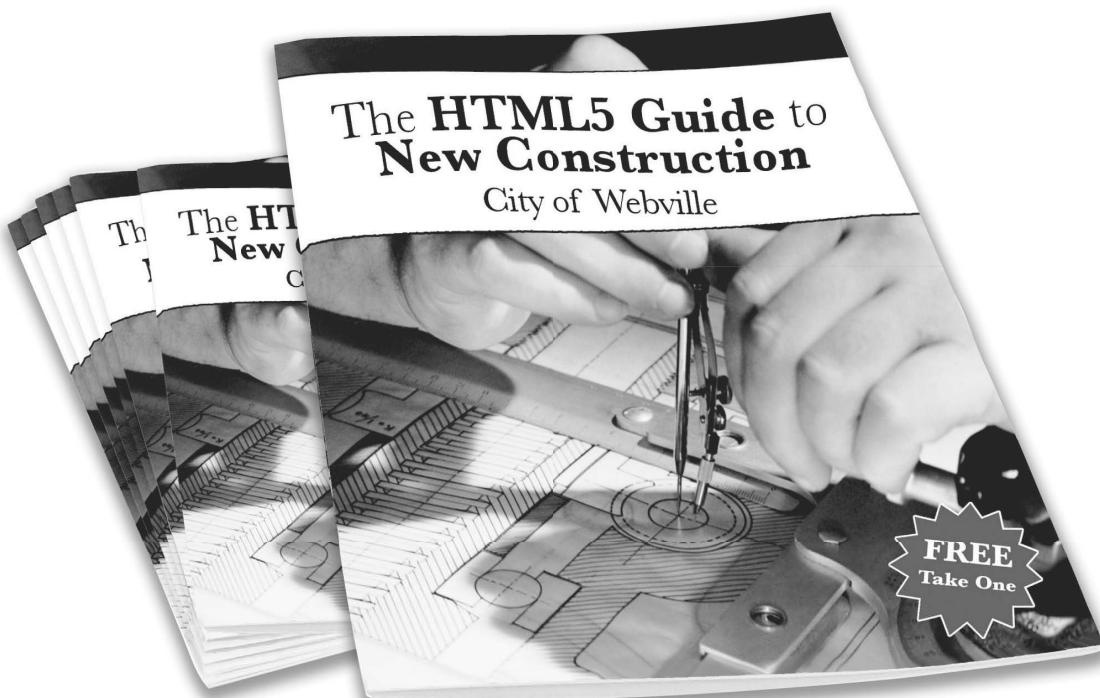




Não acredito que o livro está quase acabando.
Antes de ir, temos um pequeno presente de
despedida para você da Cidade de Webville;
é o guia para os elementos HTML5 (e o que
há de novo em CSS3) que lhe prometemos. A
Webville não é incrível?!

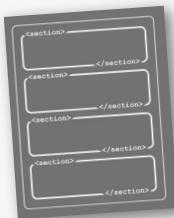
O Guia HTML5 para Nova Construção

Aqui em Webville, nós recentemente fizemos alguns acréscimos aos nossos códigos de construção e preparamos um guia útil para qualquer nova construção que esteja considerando. Particularmente, adicionamos muitos novos elementos de semântica, que lhe dão ainda mais poder para arquitetar suas páginas. Agora, nosso guia não é cansativo; em vez disso, nossa meta aqui é dar a você, construtor experiente, o suficiente para se familiarizar com os novos elementos HTML5 e propriedades CSS3. Desta forma, poderá usá-los nos aplicativos web que você está aprendendo como construir neste livro, quando estiver pronto. Então, se precisar de um rápido tutorial sobre os acréscimos semânticos no HTML5, leve um — eles são gratuitos (por tempo limitado).



Guia Webville para Elementos Semânticos HTML5

Aqui em Webville fizemos algumas mudanças recentes em nosso código de construção e preparamos um guia útil para todas as novas construções. Se você usou `<div>`s para construções comuns como cabeçalhos, navegação, rodapés e artigos de blog, então temos alguns novos blocos de construção para você. Certifique-se de saber os códigos.



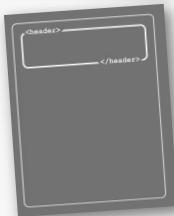
`<section>`

Um `<section>` é um “documento genérico”. Você poderia usar `<section>` para fazer uma marcação, ah, digo um Guia para HTML. Ou para encerrar o HTML para um jogo. Um `<section>` *não* é um contêiner genérico — esse é o trabalho da `<div>`. Lembre-se: use a `<div>` se estiver apenas agrupando elementos para propósitos de estilo.



`<article>`

Um `<article>` é um pedaço independente do conteúdo que talvez você queira compartilhar com outra página ou site (ou mesmo seu cão). Perfeito para postagens de blog e novos artigos.



`<header>`

`<header>` é para a parte superior de elementos como `<section>` e `<article>`. Talvez queira usar `<header>` no topo do corpo para criar o cabeçalho principal para sua página.



`<footer>`

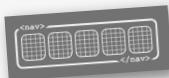
`<footer>` é para os rodapés das coisas. Coisas como `<section>`s, `<article>`s e `<div>`s. Talvez pense que lhe é apenas permitido um por página; de fato, você pode usá-lo sempre que precisar de um conteúdo de rodapé numa seção de sua página (como uma biografia ou referências para um artigo).



`<hgroup>`

Este aqui pode ser complicado. Contrário ao `<header>`, que pode conter quaisquer elementos relacionados ao cabeçalho, `<hgroup>` é especificamente para agrupamento de cabeçalhos (`<h1>...<h6>`) juntamente dentro de um `<header>`. Bom para contornos.

Guia Webville para Elementos Semânticos HTML5



<nav>

<nav> é para navegação e para links, é claro, mas não qualquer link: use <nav> quando tiver um grupo de links, como navegação para seu site, ou um blogroll. Não o utilize para links únicos em parágrafos.



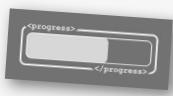
<aside>

<aside> é útil para todos os tipos de coisas que são partes de conteúdo fora do fluxo principal de sua página, como uma sidebar, uma citação pessoal ou uma segunda fase.



<time>

Finalmente! Já era tempo. Você pode marcar suas horas com <time>. Sem pressa; vá no seu ritmo e faça direito — você precisará estudar um pouco sobre os formatos válidos para <time>.



<progress>

Quase lá? Sim, estamos progredindo através desses elementos HTML5... <progress> representa quão longe você está de concluir uma tarefa. Use-o com um pequeno CSS e JavaScript para ter alguns efeitos legais.



<abbr>

Ei Sr., certifique-se de usar uma abreviatura para aquela palavra longa! Ótimo para buscas, pois mecanismos de busca não são sempre tão inteligentes para abreviaturas quanto nós.



<mark>

Use <mark> para marcar palavras, para destacar ou editar, por assim dizer. Uma ótima para usar com resultados de mecanismos de busca.

Adicionando estilo à sua nova construção com CSS3

Guia Webville para Propriedades CSS3

Agora que você tem seus blocos novos de construção no lugar, está na hora de pensar na decoração. Você vai querer que sua nova construção fique bonita, certo?

Novas propriedades

Há algumas novas propriedades no CSS3, muitas das quais fazem o que autores de página têm feito por anos com vários contorcionismos de HTML, imagens e JavaScript. Exemplos:



`opacity: 0.5;` Torna um elemento 50% opaco
`border-radius: 6px;` Cria um efeito arredondado com uma curvatura de 6px em cada canto
`box-shadow: 5px 5px 10px #373737;` Uma sombra de 5px de comprimento, 5px de altura, um borrão de 10px e uma cor cinza escuro.

Novos layouts

Há algumas poderosas novas maneiras de estilizar sua página com CSS, que vão além de posicional e são muito mais fáceis de usar. Exemplos:



`display: table;` } Isso lhe dá um layout de quadro
`display: table-cell;` sem os quadros HTML.

`display: flexbox;` Com flexbox, você tem mais controle
`flex-order: 1;` sobre como o navegador faz as caixas fluírem, como <div>s dentro da página.

Novas animações

Com animações, você pode animar entre valores de propriedade. Por exemplo, você pode fazer algo desaparecer, transitando da opacidade para translúcido:



`transition: opacity 0.5s ease-in-out;`
`opacity: 0;` Ao ajustar a opacidade para 0, digamos num evento hover, podemos criar uma animação

A propriedade transition especifica uma propriedade para transitar para dentro e para fora (neste caso, da opacidade), quanto tempo leva para fazer a transição e a função easing, então é gradual.

Novos seletores

Há um monte de novos seletores, incluindo nth-child, que lhe permitem dirigir elementos child (filho) específicos encerrados num elemento. Finalmente, você pode ajustar a cor do background de fileiras alternantes numa lista, sem ficar maluco.

`ul li:nth-child(2n) { color: gray; }` Isso significa: selecione todos os outros itens da lista e defina a cor do background como cinza.

Índice

Números e símbolos

- _ (underline), começando nomes de variáveis
 - JavaScript 40, 42
- , (vírgula), propriedades de separação de objeto
 - 132
- ; (ponto e vírgula), finalizando afirmações em
 - JavaScript 39
- . (ponto) operador
 - acessar propriedades object 133, 134
 - métodos invoking 151
- “ ” (aspas, duplas)
 - denotar strings vazias 26, 95, 108
 - em torno de codecs, parâmetro de elemento <source> 359
 - em torno de valores de propriedade
 - JavaScript 133, 308
 - envolvendo strings de caracteres em
 - JavaScript 39
- [] (colchetes)
 - denotar strings vazias 26, 95, 108
 - em torno de codecs, parâmetro de elemento <source> 359
 - em torno de valores de propriedade
 - JavaScript 133, 308
 - envolvendo strings de caracteres em
 - JavaScript 39
- { } (chaves)
 - encerramento de blocos de código 26
 - encerramento de propriedades de objetos
 - 132
- // (barras), comentários iniciais sobre
 - JavaScript 39
- + (sinal de mais)
 - operador de adição ou operador de concatenação de string 45
 - operador de concatenação de string 26
- \$ (cifrão)
 - \$() (função jQuery) 534
 - iniciando nomes de variáveis
 - JavaScript 40, 42
- 2D contexto de desenho, canvas 292. *Ver também* Canvas API;

A

- AAC Áudio 357
- AAC e Vorbis codificações 357
- <abbr> (abreviação) elemento 547
- addEventListener, método 367
 - chamar error handler 406
 - listener para ended video event 386
 - popping up botão play após fim do vídeo 386
- addMarker, função (exemplo) 186
- addStickyToDOM, função (exemplo) 430, 432, 440
 - passar chave, assim como o valor toda vez que for chamada 450
 - usando objeto sticky, em vez de string 455
- adicionar elemento <canvas> para página 286
- adicionar janela de informação para marcador Google Maps (exemplo) 187
- adicionar para cabine de vídeo, código 383
- afirmações 37
 - finalizando com ponto e vírgula 39
- afirmações if 49
- altitude e altitudeAccuracy propriedades, coordenadas
- ângulos
 - medidas em graus, converter em radianos 317
 - startAngle e endAngle parâmetros de arco método 315
- animações, novas, em CSS3 548
- Apache
 - dizendo para servidor de arquivos de vídeo com certas extensões de arquivo 371
 - usando em Mac, PC, e Linux 231
- API arraste e solte 543
- API Geolocalização 16. *Ver também* geolocalização
 - componentes de 190
 - entrevista com 189
 - getCurrentPosition método 174, 177, 190, 207
 - posição, opções de 198
 - watchPosition, método 194
- APIs (Aplicativo Programming Interfaces) 15, 31

o índice remissivo

- aplicativo/xhtml+xml MIME, tipo 536
aplicativos web
 APIs para criar 15
 e HTML5 6, 13
 e JavaScript 21, 24
 exemplos 22
 o que é isso 28
 offline 538
- Aponte Seu Lápis exercício
 drawText, função 330
 mostrar apenas novos quadrados no preview 306, 342
- appendChild, método
 elemento, objeto 158
 em addStickyToDOM, função (exemplo) 450
 objeto método ul 101
- apresentação, separar de conteúdo, em canvas 326
- argumentos, função 120
 objetos, como 136
 passando a parâmetros 122, 162
- Arquivos Locais, origem 422
- arrays 67
 adicionar itens 68
 armazenando em local storage 440, 467
 criar e designar para uma variável 67
 de objetos 134, 457
 de workers 508
 enchimento lista, itens de (exemplo) 69
 length de 68
 localStorage, objeto como array associativo 424
 obter valor de itens em 68, 303
 remover itens de 73, 448
 resolução de problemas em local storage 439
 usar para armazenar valores múltiplos 75
 vídeo lista de reprodução 365
- arrays associativos 424
- atributo draggable 543
- atributo id. Ver *também* getElementById, método;
 documento, objeto
 <canvas>, elemento 290
 <script>, elemento 267
 <video>, elemento 353
 acessar elementos por 59
 necessário para deletar sticky note (exemplo) 450
- atributos, obtendo e ajustando 158
 ajuste de atributo de elemento 379
 ajuste de atributo id de sticky note 450
 áudio 16, 533
 autoplay atributo, <video> elemento 353, 354
- ## B
- background tarefas 95
background, cor de
 canvas, preencher antes de desenhar novos quadrados 306, 342
 definir backgroundColor, propriedade para sticky note 455
 definir para fileiras alternadas numa lista 548
barra (/), começo dos comentários JavaScript 39
- beginPath método, canvas context 311, 319
- “bitmap”, desenhar, sobre o canvas 336
- <body>, elementos, adicionando elementos <script> em 53
- booleanos 40
 booleanas, expressões 43
 testes condicionais em para e enquanto afirmações 47, 49
 usar para tomar decisões com JavaScript 49
 valores true e false 39
- border-radius propriedade 548
- botões
 assistindo posição e limpando o relógio 193
 botão objeto, onclick propriedade 154
 click handlers para cabine de vídeo, JavaScript código 377–379
 controlar efeitos em cabine de vídeo 390, 391
 createSticky handler 432
 CSS styling para cabine de vídeo 381
 HTML para botões de cabine de vídeo 375
 implementar para cabine de vídeo 384–386
 JavaScript factory código para cabine de vídeo 376
 limpar local storage 435
 manipulando click event 89, 92, 102, 108
 preview, botão para camiseta design aplicativo 302
 selecionando entre vídeos de teste 387
 sticky note aplicativo 431
 toggle ou botões de rádio 380
- box-shadow propriedade 548
-
 elemento 26

C

cache manifest arquivo para aplicativos web offline 538
 cache, navegador 272, 277
 callbacks 254, 277
 obtendo tuítes enviados do Twitter 322
 camel case em multi-word nomes de variáveis 42
 caminhos e arcos em canvas 310, 338
 arc, método 313–315
 desenhar uma carinha feliz 344
 usar método arc e path 343
 usar método arc para desenhar um círculo 316
 usar método arc para traçar um caminho 316
 usar caminhos para desenhar formas 311
 camiseta 282. *Ver também* TweetShirt App Web
 camiseta design aplicativo web 282
 canPlayType método, vídeo objeto 368–374
 Canvas API 16, 281–348, 540
 resposta “maybe”, mas a reprodução falha 371
 usar para determinar formato do vídeo para seu navegador 369
<canvas>
 adicionar borda, usar CSS 288
 adicionar na página 286
 conversa fechada sobre o elemento <canvas> 285
 parceria com elemento <video> 339, 388
<canvas> elemento vs. gráficos SVG 537
 carácter codificação, UTF-8 9
 chamada para fillBackgroundColor função 307
 character strings, citando em JavaScript 39
 chaves ({ })
 encerramento de código blocks 26
 encerramento propriedades de objeto 132
 childElementCount, propriedade, elemento objeto 158
 Chrome 20. *Ver também* navegadores
 HTML5 suporte 18
 Ogg/Theora vídeo 357
 restrições de segurança no Web Workers 482
 segurança restrições no vídeo+operações
 canvas 371
 WebM/VP8 vídeo 357
 .webm arquivos de vídeo 352

cifrão (\$)
 \$() função em jQuery 534
 nomes de variáveis JavaScript 40, 42
 cinema aplicativo (exemplo) 138
 adicionar comportamento ao objeto Movie com um método 143–145
 criar objetos movie 139
 função Movie construtor 150, 152
 implementar função getNextShowing 140
 usar esta tecla para referência a Movie objeto 145
 usar Movie construtor para criar objetos Movie 153
 círculos, desenhar com canvas 309–317, 338
 arc método 314
 converter medida do ângulo em graus para radianos 317
 criar caminhos 311–313
 class atributo, <anchor> elemento 379
 class, selecting elemento por 542
 clear método, localStorage objeto 435
 clearInterval método 271
 clearStorage função (exemplo) 435
 clearWatch método 190
 click events
 adicionar handler em aplicativo de geolocalização 194
 adicionar handler em canvas aplicativo 302, 347
 adicionar handlers para aplicativo sticky notes 431, 435, 450
 designar handler a elemento usando jQuery 534
 handler alertando usuário de cliques de botão 92
 handlers para aplicativo de vídeo 376
 handling para botão Add Song 89
 handling para botões 108
 click handler para canvas em Fractal Viewer (exemplo) 515
 close(), método, worker objeto 522, 524
 closePath, método, canvas context 312
 codecs
 AAC áudio 357
 codecs, parâmetro de tipo de elemento <source>, atributo 359
 definido 358
 H.264 vídeo 357
 Theora, vídeo 357
 tipos principais de 356
 Vorbis, áudio 357

o índice remissivo

- VP8, vídeo 357
- codecs 358
- codificação, seu próprio vídeo 360
- codificações em arquivos de vídeo 356
- código inline, escrever em HTML5 <script> elemento 5
- código reuse
 - funções e 119
 - métodos e 146
- colchetes ([])
 - acessar objeto propriedades 133
 - criar com fillRect 290, 292, 304
 - criar e indexar arrays 67
 - e arrays associativos 424
 - escrever drawSquare função 304
 - preencher cor de fundo da canvas antes de desenhar novos quadrados 306
 - pseudocódigo para drawSquare função 303
 - usar com localStorage 424
 - x, y, e width aleatórios de quadrados 308
- comentários sobre JavaScript 39
- comprimento, propriedade
 - arrays 68
 - localStorage, objeto 424, 430, 432
- computeDistance função (exemplo) 180
- concatenando strings. *Ver também + (sinal de mais)*, sob Símbolos
 - criar slogans de marketing (exemplo) 72
- condicionais 37
 - testar em while e para loops 47
 - while loops 46
- consciência da localização 165
- construtores 146, 160
 - criar 147
 - embutidos 151
 - LatLong construtor do Google Maps 183
 - Mapa, construtor do Google Maps 184
 - Movie construtor função 150, 152
 - usar 148
 - usar Movie construtor para criar objetos
 - Movie 153
 - WebSocket 539
- containers 356
 - definidos 358
 - em src, atributo de elemento <source> 359
 - MIME type para <source> type atributo 359
 - MP4 container 357
 - Ogg container 357
 - WebM container 357
- Conteúdo Delivery Network (CDN) empresas, codificação serviços 360
- contexto, canvas 292, 293, 504. *Ver também Canvas API*
 - arc, método 313–317
 - beginPath, método 311, 312
 - closePath, método 312
 - definido 293
 - drawImage, método 333
 - fillRect, método 292, 304
 - fillStyle, propriedade 330, 331
 - fillText, método 328, 329, 330, 331
 - font, propriedade 329, 330, 331
 - lineTo, método 311, 312, 329
 - moveTo, método 311, 312, 329
 - obter 302
 - saving e rearmanezar 540
 - stroke, método 329
 - strokeText, método 328
 - textAlign, propriedade 328, 330, 331
 - textBaseline, propriedade 329
 - translate e rotate, métodos 540
- controls atributo, <video> elemento 354
- Conversa Informal
 - Cookie e Local Storage 426
 - XMLHttpRequest e JSONP 260
- cookies 414–416
- Conversa Informal, Cookie e Local Storage 426
- coordenadas
 - calcular distância entre 180
 - latitude e longitude 167
- coordenadas, objeto 175, 207
 - altitude e altitudeAccuracy, propriedades 197
 - latitude e longitude, propriedades 173, 175
 - propriedades 190
- coords objeto, latitude e longitude
 - propriedades 173
- coords, propriedade; posição, objeto 190
- cor de fundo de canvas, preencher antes de desenhar
- cores
 - definir cor de fundo de fileiras alternativas em listas 548
 - definir para fillStyle propriedade de contexto canvas 304, 308
 - escolhendo para sticky notes, em aplicativo stickies 453–456
 - especificar em canvas 338
 - fatores que os tornam problemáticos 416
 - fillBackgroundColor função para canvas context 307
 - fillRect método vs. fillStyle propriedade, canvas context 308

- createElement, método; document, objeto 99, 157, 335, 450
 createSticky função (exemplo) 432
 convertendo para usar um array 441
 reescrevendo para armazenar texto com sticky note 454
 stickies, aplicativo; versão final 444
 createTask, função (exemplo) 512
 cross-domínio, problemas com XMLHttpRequest 243–252
 CSS 31
 estilização para cabine de vídeo 381
 estilizar <canvas>, elemento; adicionar borda 288
 padrão declarado para estilizar 5
 posicionamento vídeo e canvas 395
 propriedade values 308
 seletores 542
 usar para definir atributos width e height de <canvas> 289
 usar para estilizar sticky notes 429
 CSS3 16, 28, 548
 estilização de página 14
 currentTime, propriedade, áudio objeto 533
- ## D
- datatypes
 conversões em JavaScript 41, 45
 dynamic typing em JavaScript 39
 tipos primitivos 40
 variáveis em JavaScript, no strict types 38
 definindo funções com parâmetros 120
 degradação sem rupturas 19
 degreesToRadians, função 180, 317, 319, 344
 deletando objeto, propriedades 135
 deleteSticky, função (exemplo) 449
 event object; target, informação 451
 desenhar no canvas 290–294
 círculos 309–317
 círculos aleatórios para camiseta, design do aplicativo 318
 escrever drawSquare, função para desenhar quadrados 304
 usar caminhos para desenhar formas com linhas 311
 desenhar no canvas 290–294, 338
 arc, método 314
 caminhos e arcs 311–318
 drawBird, função (exemplo) 334, 346
 drawSmileyFace, função (exemplo) 321, 344
 drawSquare, função (exemplo) 302, 342
 pseudocódigo para 303
 escrever 304
 drawText, função (exemplo) 327, 330, 345
 completando 331
 desenhar texto 325–332, 345
 desenhar uma carinha feliz 321, 344
 direção de parâmetro, arc método 315
 displayLocation handler função 173, 175
 alterar para show map somente uma vez 195
 chamadas de watchPosition, controlando 206
 implementação alternativa 197, 210
 mostrando novo marcador apenas depois de viajar mais do que 20 metros 209
 dispositivos móveis
 canvas em 335
 testar código de geolocalização 179
 navegador, suporte para aplicativos web offline 538
 distância
 cálculo e mapeamento de 197
 calcular 180
 controlar acréscimo de novos marcadores no mapa 209
 <doctype> elemento 3
 escrever código para encontrar 181
 mudanças em HTML5 9, 31
 mudando HTML 4.01 doctype para HTML5 4 omitindo 9
 document, objeto 56, 154
 createElement, método 99, 101, 335, 450
 getElementById, método 59, 157
 getElementsByTagName, método 270
 método write 28
 propriedades e métodos 157
 querySelector, método 542
 querySelectorAll, método 376, 542
 documents, cross-document messaging 543
 DOM (Document Objeto Model) 14, 31, 54–65
 adicionar elementos para 100
 adicionar stickies de local storage 428, 430
 analizar HTML e construir DOM de 81
 Aponte Seu Lápis exercício 61
 criar 55
 criar novos elementos <script> para atualizar continuamente dados 263, 267
 deletar sticky note de 452
 desenhar para músicas adicionadas à lista de reprodução 98, 110

o índice remissivo

- estrutura e conteúdo de 56
 - inabilidade de acessar ou mudar antes de a página ser carregada 64
 - inserindo e substituindo JSONP elementos <script> 268
 - interação de JavaScript com 54
 - Não existem perguntas idiotas 271
 - nomes 97
 - obter elementos de, usar jQuery 534
 - obter, criar, adicionar, ou remover elementos 66
 - replaceChild, método 270
 - resumo de pontos importantes 108
 - retornando elementos pelo nome da tag 270
 - selecionando elementos de, usar Selectors API 542
 - vazio, elemento para elementos para manter música workers sem permissão para acessar 480
 - domínio propriedade, document objeto 157
 - domínios
 - cross-origin problemas com XMLHttpRequests 244–253
 - local storage, allocated per domínio 422
 - origem e management de local storage 422
 - drawCircle, função (exemplo) 318
 - escrever 319
 - drawImage, método 333
 - dropshadows, em canvas 335
 - dynamic typing em JavaScript 39
- ## **E**
- efeito de ficção científica para vídeo 374, 400, 410
 - efeito faroeste para vídeo 374, 400, 410
 - efeitos 410
 - aplicar a vídeos 389–391
 - criar usando canvas context, translate e rotate, métodos 541
 - escolha de cabine de vídeo 378
 - escrever efeitos especiais para vídeo 399–404
 - efeitos especiais
 - aplicar para vídeos 389–391
 - funções 399, 410
 - effectFunction
 - chamando para aplicar filtro de vídeo 397
 - usado como variável para manter filtro, função 391
 - elemento <article> 546
 - elemento <aside> 547
- elemento objetos 158
 - returned por getElementById, método 160
 - elementos
 - acessar com getElementById 59
 - adicionar ao DOM 100
 - criar 99
 - definindo atributos com método setAttribute 267
 - obter com método getElementById 114, 157
 - obter com método getElementByTagName 154, 269
 - obter com método getElementsByClassName 154
 - Elementos Adobe Premiere 360
 - elementos filho
 - adicionar um elemento no DOM 108
 - no DOM, estrutura em árvores 100
 - nth-child selector 548
 - replaceChild método 270
 - enableHighAccuracy opção 198, 201
 - encadeamento
 - objetos, propriedades e métodos, geolocalização 175
 - objetos e propriedades, filme exemplo 141
 - encaixando o canvas na janela do navegador em Fractal Viewer (exemplo) 517
 - endedHandler função (exemplo) 386
 - endereço IP, baseado em informações de localização 168
 - entrevistas
 - com Função 119
 - com Geolocalização 189
 - com HTML5 11
 - com JavaScript 24, 477
 - com Vídeo 388
 - com XMLHttpRequest 225, 240
 - enumerar propriedades de um objeto 133
 - error handlers
 - em workers 522
 - Geolocalização API 190, 207
 - para getCurrentPosition 174, 177–179
 - para watchPosition 194
 - para cache errors 538
 - video errors 406
 - error, propriedade, objeto vídeo 405
 - erros
 - navegadores, ignorar pequenos erros nos arquivos HTML 9
 - Geolocalização API
 - timeout, error 200

- tipos de erros 178
- JavaScript, sintaxe 44
- localStorage, quota excedida 458
- manipular erros com video playback 371
- video error, tipos 405
- XMLHttpRequest sem erros, 200 código de resposta 239
- espaço em branco em JavaScript código 39
- estrutura 35, 545
- evento ended, vídeo 365
 - adicionar event listener para 386
 - escrever handler para 367
- evento load
 - e image onload, propriedade 333
 - e window.onload propriedade 64
- evento manipular 89
 - addEventListener, método, registrar event handler 367
 - botão click handler 102
 - clearWatch event handler 195
 - createSticky (exemplo) 431
 - criar handler e designá-lo ao botão onclick
 - deleteSticky (exemplo) 450
 - handler alertando usuário que botão foi clicado 92
 - handler para evento de video ended 386
 - handler para fazer imagem de desenho no canvas 347
 - handleRefresh, função 265
 - handlers para cabine de vídeo botões 377
 - HTTP request handler 221
 - onclick event handler
 - onclick event handler para dar zoom no canvas em Fractal
 - onload event handler função para Mighty Gumball (exemplo) 229
 - onload event handler para imagem do pássaro do Twitter (exemplo) 333
 - onload handler, anonymous, função 156
 - onmessage event handler para Web Sockets 539
 - onmessage event handler para worker 485
 - onopen event handler para Web Sockets 539
 - previewHandler função (exemplo) 302
 - propriedade 91
 - retrabalhando handleButtonClick para obter título de música digitado no formulário pelo usuário 96
 - revisão de pontos importantes 108
 - tipos de eventos manipulados pelo JavaScript 95
 - Visualizador 515
- evento objeto
 - data e target propriedades 485
 - dataTransfer, propriedade 543
 - target, propriedade 451
- events
 - âncora, click event 376
 - botão click event 90, 91, 92, 93
 - cache, notificação de 538
 - canvas, click event 347, 383, 515
 - dragstart e dragend, eventos 543
 - image load, evento 333
 - propriedades para event handlers em objetos 154
 - request load, evento 221, 222, 229
 - vídeo 363
 - video ended, event 367, 386
 - window load, event 64, 129, 155, 156, 158, 159
- exceções, QUOTA_EXCEEDED_ERR 458, 468
- exercício, drawBird, função 334, 346
- exercício, usar caminho para desenhar linhas e fill shape com cor 312, 343
- exercícios
 - Aponte Seu Lápis
 - adicionar títulos de música para lista de reprodução 65, 82
 - canvas, drawText função 330
 - canvas, mostrar somente novos quadrados em preview 306, 342
 - DOM com mensagem secreta 61
 - funções 122, 162
 - geolocalização 171, 197, 210
 - HTML5 marcação 3, 7, 8
 - JavaScript afirmações 44, 77
 - local storage, deletar uma sticky 447, 448
 - local storage, problemas na implementação de stickies 437, 467
 - populando lista de itens de um array 69, 83
 - retrabalhando handleButtonClick, função 94, 96
 - testando para usuário input num formulário 94, 96
 - usar setInterval em aplicativos web 266
 - video control, botões; toggle ou radio 380, 382
 - vídeo, lista de reprodução, implementar 364, 365
 - vídeo, efeitos de faroeste e ficção científica 400, 410
 - Web Workers 481, 490, 527
 - canvas, drawText função 327, 345
 - Expresse-se (JavaScript) 44

o índice remissivo

- geolocalização 209
- HTML5 arqueologia 20
- Ímãs de Geladeira 51, 80
- Jogo da Concha, local storage 425, 466
- lucky/unlucky serviço web 223, 224
- Movie, construtor 150, 152
- Não tente isso em casa
 - local storage, excedendo a quota 458, 468
 - quão rápido o navegador pode encontrar uma localização 202
- “O que é HTML5?” 30
- O que faz o quê?
 - geolocalização opções 200, 211
 - HTML5, família de tecnologias 16, 33
 - localStorage API 461, 470
- Palavras Cruzadas
 - aplicativos web conversando com a web 278, 280
 - canvas 340, 346
 - funções e objetos 161, 163
 - geolocalização 208, 212
 - HTML5 32, 34
 - interações de HTML e JavaScript 109, 111
 - JavaScript 76, 84
 - local storage 465, 471
 - vídeo 409, 411
 - Web Workers 525, 528
- Pseudo-ímãs de Geladeira, drawSquare, função 303, 342
- Sinta-se como o Navegador 48, 78
 - construir o DOM 57, 81
 - interface elemento values 299, 341
 - interpretando chamado para método arc 317, 343
 - rendering interface de usuário 298
 - Web Workers 488, 526
- suspense, movendo para servidor online 239, 242
- expressões 39, 43
 - avaliando 44, 77
 - conversões de tipo 45
- expressões numéricas 43
- extensões de arquivo para vídeo 352, 369
- F**
- false (valor booleano) 39
- família de tecnologias 12, 29
 - função de cada 16, 33
- ferramentas de desenvolvedor embutidas no navegador 434
- fill, método; canvas, context 312
- fillBackgroundColor, função 306, 342
 - chamando 307
 - Aponte Seu Lápis exercício 306, 342
- fillRect método; canvas context 292, 304
 - efeitos de propriedade fillStyle 308
- fillStyle propriedade, canvas context 304
 - análise mais de perto de 308
- fillStyle, propriedade de canvas context 308
- fillText método, canvas context 325, 328
 - usar com texto do tuíte (exemplo) 331
- filme noir, filtro de vídeo 374, 399
- filtro de vídeo bwcartoon 400, 410
- Firefox. Ver *também* navegadores
 - HTML5 suporte 18
 - Ogg/Theora video 357
 - .ogv arquivos de vídeo 352
 - WebM/VP8 video 357
- Flash
 - HTML5 versus 284
 - usar para resolver problemas de cross browser 20
- Flash Video 358
- flexbox layout 548
- font, propriedade; canvas, context 329
 - definindo para texto do tuíte (exemplo) 331
- <footer> elemento 546
- formatos 357, 533
- formulário para camiseta, interface do aplicativo 298
- formulários 16, 85–112
 - adicionar botão a 91
 - adicionar design de formulário de camiseta para página HTML 301
 - adicionar tuítes ao elemento <select> em formulário 323
 - atualizar para add colors 453
 - camiseta aplicativo interface 298
 - documento HTML5 para manter formulário e listar elemento para lado cliente, acessar valores em 296
 - lista de reprodução 87
 - lista de reprodução manager aplicativo 102
 - mostrando lista de reprodução em página HTML 97
 - obter text de input elemento 94, 108
 - sticky note, aplicativo 429
 - tracking posição 193
 - usar JavaScript para real interatividade 23
 - verificando se o usuário digitou texto de entrada 96

- Fractal Explorer aplicativo, construir (exemplo) 494, 503
 código assado para Mandelbrot Set (cálculo) 504–507
 como o número de workers afeta a performance 520
 criar página Fractal Viewer HTML 503
 criar workers e dar tarefas para 508
 escrever o código 509
 gerenciando gerações de fractais 518
 handling click events para zoom em 515
 implementar workers 511
 obter workers iniciados 510
 processando os resultados dos workers 514
 tarefas 512
 test drive final 519
- fractal, imagem, Mandelbrot Set como 495
- FTP programas 232
- fullscreen, reprodução de vídeo 360
- função showMap (exemplo) 184
 certificando que será chamado somente uma vez 195
 criar mapa e mostrar marcador para localização inicial 205
- funções 113–130, 160
 anatomia de 121
 callbacks 254
 como funciona 116
 construtor 147
 criar seu próprio 115
 declarações, disposição de 127
 definindo 71
 definindo efeitos especiais para vídeos 391
 designando para objeto window, onload propriedade 75
 embutido 119
 entrevista com 119
 escopo de local e variáveis globais 124
 inabilidade para passar para construtor Worker 491
 invocando 116
 life span de variáveis 125
 Math biblioteca 75
 métodos versus 151
 Não existem perguntas idiotas 121, 127
 nomeando 121
 objeto passado para, acessar propriedades de 134
 Palavras Cruzadas 161, 163
 parâmetros e argumentos 120
 passando uma função para uma função 175
 passando argumentos para parâmetros 122, 162
 passando objetos para 134, 136
 retornar afirmações no corpo 117
 retrabalhando como métodos 143
 funções anônimas 128
 usando 129
- funções, continuação
 Aponte Seu Lápis exercício 122, 162
 como valores 128
 usar como valores 129
 variáveis definidas em 123, 160
- ## G
- geolocalização 165–212
 adicionar um mapa em sua página 183
 adicionar um marcador Google para seu mapa 186
 Aponte Seu Lápis exercício 171
 clearWatch handler 195
 com marcadores num mapa 204
 como funciona getCurrentPosition 176
 descobrindo quanto rápido seu navegador pode encontrar uma localização 202
 error handler 177
 especificar opções 201
 getCurrentPosition, método 175
 implementação alternativa para displayLocation 197, 210
 mapeando sua posição 182
 mostrar mapa em sua página 184
 Não existem perguntas idiotas 166, 197
 Não tente isto em casa, exercício 202
 O que faz o quê? Exercício 200, 211
 outras utilizações para o Google Maps 188
 Palavras Cruzadas 208, 212
 precisão de localização 191
 resumo de pontos importantes 207
 servidor solicitado para testar código em dispositivos móveis 179
 success handler para getCurrentPosition 174
 timeout e maximumAge, opções 199
 tracking movements 192
 watchLocation handler 194
 watchPosition, método 192
 geolocation, propriedade; navigator objeto 174
 geração, fractal (exemplo) 518
 gerenciador de lista de reprodução, criar 86
 adicionar código para lista de reprodução salva 105

- aplicativo usado para entrar música, clique do botão e adicionar música para código para salvar a lista de reprodução 104
- documento HTML5 para manter formulário e list element para
- DOM depois de títulos de música são adicionados a lista de reprodução 98, 110
- integrando storage, código 106
- lista de reprodução 102
- lista de reprodução 87
- manipulado Add Song botão click events 89
- mostrando lista de reprodução no HTML 97
- obter nome de música de elemento text input 94
- GET request (HTTP) 220
- getAttribute método, elemento objeto 158, 379
- getContext método, canvas objeto 292, 293
- getCurrentPosition método, geolocalização objeto 174, 190, 207
- como funciona 176
- error handler para 177
- getElementById método; document, objeto 58, 72, 157
- usar para localizar elemento e mudar seu conteúdo 59, 60
- getElementsByClassName, método; document, objeto 157
- getElementsByTagName, método; document, objeto 157, 270
- getFormatExtension, função (exemplo) 369, 370
- getItem, método; localStorage, objeto 419, 421
- getMyLocation, função (exemplo) 172
- getNextShowing, função (exemplo) 140
- getStickiesArray, função (exemplo) 443
- getTime, método; Date, objeto 140, 272, 442
- getTimeFromString, função (exemplo) 140
- Google Chrome. *Ver* Chrome
- Google Maps 182
- adicionar marcador em seu mapa 186
- LatLong, construtor 183
- outros usos para 188
- GPS (Global Positioning System) 168
- dispositivos sem, usando API de Geolocalização em 189
- gráficos, SVG 537
- graus
- ângulos medidos em 316
- convertendo em radianos 317
- latitude e longitude em, convertendo em valores decimais 167
- Greenwich, England, longitude medida de 167
- ## H
- H.264, formato de vídeo 352, 356, 357
- handleButtonClick, função (exemplo) 90
- código para criar elemento filho e adicioná-lo ao DOM 101
- designar para botão; onclick, propriedade 91
- retrabalho para obter título de música; digitar formulário para usuário 94, 96
- rodando quando usuário clica no botão 93
- handleClick, função (exemplo) 515, 516
- handleControl, função (exemplo) 377, 384
- implementar resto dos controles de vídeo 385
- handleRefresh, função (exemplo) 265, 267, 272
- adicionar lastreporttime, parâmetro 275
- handleRequest, função (exemplo) 523
- <head>, elemento
- <link> e <script>, elementos em 5
- pondo <script>, elementos em 53
- substituindo <script> elementos filho 270
- <header>, elemento 546
- heading, propriedade; coordinates, objeto 190, 197
- <hgroup> elemento 546
- HTML
- analisar e construir DOM de 57
- interação de JavaScript com markup 54
- HTML, entidades, em tuítes no canvas 335
- HTML5
- Aponte Seu Lápis exercício 3, 7
- como realmente funciona 14
- convertendo HTML 4.01, documento para 2–5
- entrevista com 11
- família de tecnologias 12, 16, 33
- Ímãs de Geladeira - exercício, “o que é HTML5?” 30
- interações de HTML e JavaScript 109, 111
- JavaScript como parte integrante de 21, 118, 130
- JSON e JSONP 271
- manipulando navegadores antigos 19
- marcação, JavaScript APIs e CSS 29
- melhorias na marcação 14
- Mighty Gumball, página do aplicativo (exemplo) 218
- Não existem perguntas idiotas 9, 20, 28, 284

novas capacidades e recursos 12
 novos elementos, referências 545
 o que é isso 12
 O que faz o quê? Exercício 16, 33
 o que você pode fazer com HTML5 e
 JavaScript 22
 página para design da camiseta, aplicativo 300
 Palavras Cruzadas 32, 34
 pré-requisitos para aprendizado 10
 recomendação final de padrão 20
 resumo de pontos importantes 31
 suporte em navegadores 18
 versus usar Flash ou aplicativos
 personalizados 284
 HTTP Dynamic Streaming da Adobe 404
 HTTP Live Streaming da Apple 404
 HTTP usado com XMLHttpRequest
 HTTP, respostas 219, 221, 239
 acessar returned data 222
 HTTP, solicitações 219, 220, 239
 servidor solicitado para uso 230
 HTTP-baseado request/response, modelo 539
 HTTP-baseado video streaming 404

I

IE. Ver Internet Explorer
 if/else afirmações 50
 <iframe>, elemento 543
 IIS servers, configurando MIME types 371
 image, fazendo o design da camiseta desenhado
 em canvas 347
 image, objetos
 criar 333
 Image, construtor 335
 Ímãs de Geladeira exercício 327, 345
 , elemento, <canvas> versus 285
 iMovie, codificação de vídeo com 360
 implementar um scratch buffer 395–398
 importScripts função global, Web Workers
 493, 511
 usar para realizar solicitações JSONP 523
 Indexed Database API 543
 indexes, array 75
 InfoWindow, objeto 187
 init, função 64
 como função anônima 159
 innerHTML propriedade 60
 elemento objeto 158
 usar para mudar conteúdo do elemento 62

insertBefore, método; elemento, objeto 158
 interface transformações em elementos, usar
 jQuery 535
 Internet Explorer. Ver também navegadores
 canvas suporte, versões 9 e later 294
 formato de arquivo de vídeo 352
 HTML5 suporte 18
 MP4/H.264 vídeo, suportado por IE9 357
 versões 6 e 7, sem suporte localStorage 104
 Web Worker sem suporte anterior a IE10 482
 XMLHttpRequest, objeto e 240
 interval timer, parada do 271
 invocando funções 116
 com argumentos 120
 isButtonPushed helper, função (exemplo)
 379, 384
 iterando localStorage 424

J

JavaScript 31, 35–54
 adicionar comportamento com 35
 adicionar em páginas 53
 APIs 15
 Aponte Seu Lápis exercício
 afirmações 44, 77
 displayLocation, implementação 197, 210
 funções 122, 162
 populating lista de reprodução, itens usar
 um array 65, 82
 populating list, itens de um array 69, 83
 reworking handleButtonClick, função 94
 usar setInterval em aplicativos web 266
 arrays 67, 69, 71–73
 armazenar em localStorage 439, 445
 e elemento <select>, opções 303
 e objetos 133
 passando para funções 122
 passando para Web Worker 484
 retornado de querySelectorAll 376
 retornado de getElementsByTagName
 270, 271

arrays associativos 424
 como funciona 36
 como lidar com tarefas de páginas típicas 474
 criar conteúdo de página HTML dinâmico 28
 declarando uma variável 38–40
 desenhar no canvas 285
 e HTML5 21, 22, 118, 130
 entrevista com 24, 477
 escrever 25
 expressões 43

o índice remissivo

- fazendo solicitações HTTP de 220–225
- fazendo uso da família de tecnologias
 - HTML5 24
- funções 113–130, 162
 - resumo de pontos importantes 160
- getElementById 58
- habilitando botão de preview no design da camiseta design, aplicativo 302
- handling events 89
 - revisão de pontos importantes 108
- idioma padrão de script em HTML5 5
- Imagen, construtor 335
- Ímãs de Geladeira, exercício 51, 80
 - canvas, drawText função 327
 - displayLocation handler função 209
 - Movie, construtor 150, 152
- incluindo arquivos adicionais em worker 493
- interação com página por meio do DOM
 - 15, 58
 - interação com sua página 54
- jQuery 534
- linha por linha, análise de código 26
- Modernizr biblioteca 532
- navegador, política de segurança 244–246
- Não existem perguntas idiotas 28, 41, 47, 73
 - eventos e handlers 95
 - funções 121, 127
 - funções e objetos 151
 - objetos 158
 - Web Workers 491
- objetos 113, 131–161
 - resumo de pontos importantes 160
- Palavras Cruzadas 76, 84
 - funções e objetos 161, 163
 - interações com HTML 109, 111
- palavras reservadas 41
- propriedade values em 308
- resumo de pontos importantes 75
- Sinta-se como o Navegador, exercício 48, 78, 81
- single-thread, modelo 474, 477
- sintaxe 39
- tarefas repetitivas, usar loops 46–48
- testar código em página HTML 27
- tomando decisões, usar afirmações condicionais 49
- trabalhando com canvas e vídeo 388
 - usar com HTML5 22
- jQuery 534
 - documentação online e tutoriais 535
- JSON (JavaScript Objeto Notation) 226
 - adicionar suporte a aplicativos web 236
- como dados 249–251
- converter objeto movie para e de formato
 - JSON string (exemplo) 227
- criar string, representação de um array 441, 467
- e JSONP 252
- e XMLHttpRequest 225
- formato (exemplo) 227
- HTML5 e 271
 - Não existem perguntas idiotas 271
- Palavras Cruzadas 278, 280
- performance, problemas para converter para e de strings 442
- tuítes retornados do Twitter (exemplo) 323
- URL para incluir last report time (exemplo) 275
- vendas de chiclete retornaram da Mighty Gumball (exemplo) 233
- XML e 226
- JSON.método parse 226
 - convertendo JSON string back para objeto 227
 - usar em arrays ou objetos resgatados de localStorage 443, 445
 - usar quando objeto é armazenado em localStorage 455
- JSON.stringify método 226
 - armazenar objeto em local storage 454
 - convertendo objeto para JSON formato string 227
 - usar para armazenar arrays ou objeto em localStorage 442, 445
- JSONP (JSON com Padding) 240, 247
 - atualizar aplicativos web com 256–263
 - Conversa na lareira com XMLHttpRequest 260
 - fazendo chamada para Mighty Gumball JSONP API (exemplo) 257
 - fazendo chamada para Twitter JSONP API (exemplo) 322
- HTML5 e 271
- introdução a 252
 - Não existem perguntas idiotas 271
- P em JSONP, definição 253
- Palavras Cruzadas 278, 280
 - resumo de pontos importantes 277
- segurança e 259
- tornando dinâmico 264–271
- usar importScripts para fazer solicitações 523

L

<label> elemento 453

- lastreporttime query, parâmetro (exemplo) 275
- latitude e longitude 167
 - precisão de geolocalização, informação 179
 - latitude e longitude propriedades, coordenadas objeto 184
- layouts, novo, em CSS3 548
- letras maiúsculas e minúsculas em JavaScript 41
- letter-boxing video 354
- lineTo método, canvas context 311
- lineWidth propriedade, canvas context 312
- linhas, desenhar formas em canvas 311
- <link> elementos, dentro do elemento <head>, apontando para CSS folha de estilo 5
- Linux
 - Apache servidor, configurando MIME types 371
 - criando servidor em 231
- listas
 - adicionar canções para lista de reprodução com JavaScript (exemplo) 65, 82
 - criar elementos 99
 - definir cor de background para alternar fileiras 548
 - encontrando todos os elementos filho com id de lista de reprodução, usar JQuery 535
 - lista de reprodução, gerenciador (exemplo) adicionar elemento filho para pai 100, 110
 - elementos para manter nomes das músicas 97, 99
 - elemento para manter lista de reprodução 87, 97
 - preenchendo itens usando array (exemplo) 69
 - stickies, aplicativo (exemplo)
 - criar elemento para sustentar sticky note 430
 - stickies de localStorage inseridas no elemento 430
 - elemento para sustentar stickies 429
- listas de reprodução
 - criar vídeo, lista de reprodução 364
 - implementar para Webville TV (exemplo) 366
 - populando com títulos de música usar JavaScript array 65, 82
- local storage 16, 108, 413–472
 - 5MB limite e domínio 422
 - acesso para os workers 491
- Aponte Seu Lápis exercício
 - deletar uma sticky 447, 448
 - problemas com stickies implementação 437, 467
- armazenar arrays 440
- armazenar non-String, tipos de dados 439
- armazenar números 423
- armazenar objetos 454–457
- arrays associativos 424
- browser storage, história de 414–416
- browser-based, em vez de cookies 23
- código baseado no array, integrando no aplicativo stickies 443
- código para salvar lista de reprodução 104
- como funciona armazenamento web HTML5 417
- como funciona local storage API 420
- Conversa Informal, Cookie e Local Storage 426
- deletar itens 446
- desenhando o armazenamento de seu aplicativo 457
- excedendo a capacidade de 458
- ferramentas dos navegadores para gerenciar 434
- IndexedDB e Web SQL 543
- Jogo da Concha exercício 425, 466
- navegador, problemas com arquivo:// 422
- Não existem perguntas idiotas 422, 425, 433, 442, 445
- Não tente isso em casa - exercício 458, 468
- nomeando as chaves 433, 445
- O que faz o quê? Exercício 461, 470
- Palavras Cruzadas 465, 471
- problemas com usar length para armazenar chaves 436
- resumo de pontos importantes 464
- sessionStorage, objeto 460
- stickies, aplicativo 418, 428
 - usar 462 443
- localStorage objeto
 - suporte para 422
- localStorage objeto 418
 - clear método 435
 - getItem método 419, 421
 - length propriedade 424
 - método lenght 424
 - removeItem método 433, 446, 449
 - setItem método 418, 421
 - tratando como array associativo 424
- localStorage propriedade, janela objeto 422
- loop atributo, <video> elemento 354

o índice remissivo

loop propriedade, vídeo objeto 385
looping 37, 46–48
avaliando enquanto e para loops (exemplo) 48
decidindo entre enquanto e para loops 47
enquanto loops 46
para loops 47
usar arrays com loops 69, 75

M

Mac
Apache servidor, configurando MIME types 371
configurando servidor no 231
monitor de tarefas no OS X 520
makeImage, função (exemplo) 347
makeServerRequest, função (exemplo) 523
Mandelbrot Set. Ver também Fractal Explorer
aplicativo, construir
calcular 496
código assado para calcular 504–507
equação 495
explorer para 494
usar múltiplos workers para calcular 497–500
Mandelbrot, Benoit 495
mapas
adicionar marcadores para 186, 204
adicionar para a página 183
mostrando em sua página 184
testar display do mapa em sua página 185
mapeando sua posição 182
mapOptions objeto 184
marcadores, adicionar ao mapa 186, 204
controlar frequência de novos marcadores 209
otimizar uso do marcador 206
<mark> elemento 547
marcação, nova 16, 533
Math, biblioteca 73, 75
Math.floor função 70, 304, 319
Math.PI 317
Math.random, função 70, 304, 319
x, y, e largura de quadrados desenhados no canvas 308
maximumAge, opção 199, 201
mensagem de slow script 473
mensagem handler, escrever para o worker 486
mensagens
dados que podem ser enviados 484, 491
enviando de Web Worker 486

enviando e recebendo usando Web Sockets 539
enviando para Web Workers 484
recebendo de Web Workers 485
recebendo por Web Worker 486
mensagens cross-document 543
messaging, cross-document 543
<meta> tags 31
especificar em HTML5 4
omitting 9
método arc, contexto canvas 313
desenhando círculos para camiseta, design de aplicativo 319
direção, startAngle e endAngle, parâmetros 315
interpretando chamar para, e esboçando todos os parâmetros em círculo 317, 343
usar para traçar um dado caminho 316
x, y, e raio, parâmetros 314
método drawImage, contexto canvas 333
método key, localStorage objeto 424, 430
método load
áudio, objeto 533
vídeo, objeto 385
método send, XMLHttpRequest objeto 221
métodos 142, 160
código, reutilização e 146
convertendo funções para 143
esta keyword, como funciona 149
funções versus 151
métodos e propriedades de áudio API 533
métodos save e restore, canvas context 540
Microsoft. Ver também Internet Explorer;
Windows systems
Smooth Streaming 404
Web Platform Installer 231
Mighty Gumball, aplicativo (exemplo) 214–218
atualizar código para usar JSONP 256–263
atualizar JSON URL com lastreporttime 275
escrever onload handler, função 229
melhorar o display 235
mostrar vendas 230
movendo para servidor vivo 237–246
navegador cache, prestando atenção para 272
opções para circumvent cross-origin request problemas 247–251
removendo relatórios de vendas duplicados 273
retrabalhando o código para usar JSON 236
revisando as especificações 228

testar localmente 230, 234
 tornando o JSONP dinâmico 264–271
 milissegundos desde 1970 442
MIME types
 aplicativo/xhtml+xml 536
 certificando-se de que o servidor está
 servindo arquivos de vídeo com tipo
 certo 371
 de arquivos de vídeo 359, 369
Modernizr biblioteca, JavaScript 532
moveTo método, canvas context 311
.mp3, áudio 533
.mp4, arquivos de vídeo 352
MP4, container 357
MPEG-LA, group 357
multi-core processadores 500
mute propriedade, vídeo objeto 385

N

namespaces, XHTML 536
Não existem perguntas idiotas
 canvas 289, 293, 308
 eventos e handlers 95
 falando com a web 271
 funções 121, 127
 funções e objetos 151
 geolocalização 166, 197
HTML5 9, 20
 HTML5 aplicativos web 284
JavaScript 41, 47, 73
 local storage 422, 425, 433, 442, 445
 objetos 158
 tecnologias JavaScript e HTML5 28
 vídeo 360, 371
 Web Workers 491
Não existem perguntas idiotas 289, 293, 308, 335
navegadores
 analisar HTML e construir DOM a partir de
 57, 81
 armazenamento de dados usando
 localStorage 108
 áudio, suporte de codificação 533
 background tasks 95
 caching e repeated JSONP solicitações
 272, 277
 capacidade local storage 420
 carregando e mostrando, HTML
 documentos 14
 codificações de vídeo suportado 358
 controles para vídeo HTML 355

criar workers 478
 cross-navegador compatibilidade de páginas
 HTML 20
 detectar geolocalização, suporte 174
 detectar suporte para canvas, em código 293
 detectar suporte, usando Modernizr
 biblioteca 532
 dispositivos móveis, canvas suporte 335
 encaixando canvas na janela em Fractal
 Viewer (exemplo)
 excedendo capacidade de local storage 458
 executando código apenas depois que a
 página está carregada 64
 fallbacks para vídeo suportado 362
 ferramentas de desenvolvedor para gerenciar
 local storage 434
 histórico do navegador storage 414–416
 localStorage não funcionando quando
 carregando do arquivo 422
 mesma política de origem, vídeo 408
 métodos de determinar localização 170
 política de segurança 244
 rodar código armazenado em local
 storage 104
 sem suporte <canvas>, exibindo texto
 contido em 295
 sem suporte para recursos de HTML5,
 fornecendo alternativa para 19
 suporte para aplicativos web offline 538
 suporte para HTML5 17
 suporte para Web Workers 482
 suporte para XMLHttpRequest, onload
 propriedade 239
 testar para suporte de formatos de vídeo
 para vídeo carregado pelo código 368
 vídeo formato de arquivos 352
 vídeo suporte, determinar nível de 361, 411
 Web Storage suporte 422
navegadores móveis 20
 HTML5 suporte 18
navegadores não têm suporte para canvas 295
<nav> elemento 547
navigator objeto, geolocalização propriedade 174
nextVideo handler, função 367
nomes
 de funções 121
 de variáveis 40
 guia para melhores nomes de variáveis 42
 local e variáveis globais com mesmo nome
 126
 localStorage, chaves 433, 445
 novos quadrados 306

o índice remissivo

- nth-child seletor 548
- números
 - armazenar em local storage 423
 - conversões para outros tipos em expressões 45
 - primitive type em JavaScript 40
- números decimais
 - armazenamento em local storage 423
 - conversão de números inteiros em expressões 45
- números inteiros
 - armazenar em localStorage como strings 423
 - conversão de strings 423
 - conversões para números decimais em expressões 45
- O**
 - <objeto> elemento, usar dentro elemento <video> 362
 - objeto 190, 197
 - objeto data, método getTime 140, 272, 442
 - objeto literais 151
 - objetos 40, 113, 131–161
 - adicionar ou deletar propriedades a qualquer tempo 135
 - armazenar em localStorage 445, 454
 - armazenar formas desenhadas no canvas como 336
 - array 73
 - arrays de 457
 - construtores 147
 - convertendo para e de JSON string formato 226, 227
 - criar 132
 - movie, objeto (exemplo) 138
 - usar construtores 148, 153
 - embutido versus criado por usuários 159
 - escrever versus criar com a construtor 151
 - métodos 142
 - Não existem perguntas idiotas 151, 158
 - no navegador 154
 - Palavras Cruzadas 161
 - palavra-chave this 144
 - passando para funções 136
 - propriedades 132
 - resumo de pontos importantes 160
 - usos de 133
 - offline, aplicativos web 16, 538
 - Ogg, container, formato 357
 - Ogg/Theora vídeo codificação 356, 357
 - Ogg/Vorbis áudio codificação 356, 357, 533
 - .ogv, arquivos de vídeo 357
 - onclick propriedade, botão objetos 91, 154
 - adicionar event handler função para 91, 450
 - onerror handler, usar em workers 522
 - onload handler função 64, 229
 - e anonymous funções 129, 156
 - escrever com jQuery 534
 - usar para carregar página antes de acessar o DOM 64
 - onload propriedade
 - image, objeto 333
 - janela, objeto 156
 - designar função para 64, 75, 129, 156, 265
 - XMLHttpRequest objeto 239
 - navegadores sem suporte, trabalhar para 241
 - onmessage event handler 485
 - opacity propriedade 548
 - opacity, transitando de opaco para translúcido 548
 - <opção> elemento, em stickies aplicativo formulário 453
 - opções, geolocalização API 198, 201
 - O que faz o que exercício 200, 211
 - resumo de 207
 - open event, Web Sockets 539
 - Opera. Ver também navegadores
 - .ogv arquivos de vídeo 352
 - HTML5 suporte 18
 - Ogg/Theora video 357
 - sem suporte XMLHttpRequest Level 1 241
 - WebM/VP8 video 357
 - operator de adição (+) 45
 - operador ponto (.)
 - acessar objeto propriedades 133, 134
 - invocando métodos 151
 - outras cláusulas em if afirmações 50
 - overlays, Google Maps 188
- P**
 - <p> (parágrafo) elementos, mudar usando JavaScript 62
 - páginas vs. aplicativos web 28
 - Palavras Cruzadas 340, 346
 - palavra-chave new, usar com construtores 148, 160
 - palíndromos 51
 - panTo método, map objeto 204
 - para loops 47

- avaliando (exemplo) 48
- decidindo entre loops e 47
- if/else*, afirmações em 51
- parâmetros, função 120
 - adicionar elemento filho com appendChild 100, 101,
 - adicionar elementos filho para 108 em DOM 100
 - nomes de 121
 - passando argumentos para parâmetros 116, 122, 162
- pares chave/valor
 - armazenar em um array 439
 - criar chaves únicas 442
 - em string formulário, obter e definir em local storage 419, 421
 - gerenciando chaves no aplicativo stickies 433
 - no local storage do navegador 417
 - passando chave cada vez que uma sticky note é adicionada ao DOM 450
 - unicidade de chaves em local storage 422
 - usar chave para remover item de localStorage e array 451
- parse método. Ver JSON.parse método
- parseFloat função 423
- parseInt função 423
- passando por valor 136
 - passando um objeto referência à função 136
- pause, método
 - áudio, objeto 533
 - vídeo, objeto 385
- PC, criando um servidor no 231
- Phrase-o-Mático aplicativo (exemplo) 70
- pillar-boxing, vídeo 354
- ping-pong Web Workers game (exemplo) 484
 - adicionar workers 491, 492
 - pingPong mensagem handler função para worker 486
- Sinta-se como o Navegador exercício 488, 526
- pixels
 - acessar em vídeo 392
 - como apresentação, não conteúdo 326
 - desenhar no canvas 281, 306
 - em bitmap, desenhar 336
 - processando em scratch buffer do canvas 394, 397
 - processar video pixels e obtê-los dentro do canvas para display 396
- play, botão (exemplo)
 - handler para cabine de vídeo 377
 - surgindo de volta quando o vídeo termina 386
- play, método
 - áudio, objeto 533
 - vídeo, objeto 385
- png, formato de imagem 347
- política de segurança, navegadores 244
- ponto e vírgula (;), ending JavaScript afirmações 39
- position, objeto 175, 207
 - coords e timestamp propriedades 190
- positionOptions, Geolocalização API 190, 198
- postAQuote função (exemplo) 523
- pôster, atributo, <video> elemento 353, 354
- pôster, propriedade, objeto vídeo 406
- postMessage, método
 - Web Sockets 539
 - worker objeto 484, 511, 512
- precisão, informação de localização 191
 - enableHighAccuracy opção 198
- preload, atributo, <video> elemento 354
- preto e branco, convertendo de pixels para 399
- preview em camiseta design aplicativo, problemas com 306
- previewHandler função (exemplo) 302
 - atualizar para chamar drawText função 330
 - chamando fillBackgroundColor função 307
- primitive types 40
- processando video frame em canvas scratch buffer 397
- processFrame função (exemplo) 396
 - rodrar novamente 397
- processWork função (exemplo) 514, 518
- programtheweb.com 271
- <progress> elemento 547
- propriedade dataTransfer; objeto event 543
- propriedade de dados, objeto event 485, 524
- propriedade firstChild, elemento objeto 158
- propriedade precisão, coordenadas do objeto 190
- propriedade src
 - audio objeto 533
 - image objeto 333
 - vídeo objeto 370
- propriedade target, objeto event 451, 485
- propriedade transition 548
- propriedades 132
 - acessar, changing value e enumerating 133
 - adicionar ou deletar a qualquer tempo 135
 - canvas contexto objeto 338
 - fillStyle propriedade 308

propriedades texto 328
document, objeto 154, 157
elemento objeto 158
especificando valores em JavaScript 308
Geolocalização API 190
localStorage, length propriedade 424
new, em CSS3 548
objetos como coleções de 131
vídeo, objeto 363
window, objeto 155
Pseudocódigo, ímãs, exercício 303, 342
pushUnpushButtons, função auxiliar 376, 379, 384
putImageData método, canvas context 397

Q

quadrados, desenhar no canvas 302
quebra de linha em HTML 26
querySelector método, document objeto 542
querySelectorAll método, document objeto 376, 542
Quicktime 371
QUOTA_EXCEEDED_ERR exception 458, 468
quotation marks, double. Ver “ ”, under Symbols

R

radianos 316
convertendo graus para 317
rádio, botões 380, 382
radius, parâmetro de arc método 314
rastreando movimentos 192–198
formulário para começar e parar
rastreamento 193
reassignWorker função (exemplo) 514, 518
referências, objeto 136
removeItem método, localStorage objeto 433, 446
removeStickyFromDOM função (exemplo) 452
repetitive tasks 46
replaceChild método 270
request/response model based on HTTP 539
reserved words em JavaScript 41
resizeToWindow função (exemplo) 517
responseText propriedade, request objeto 222
restore, método, canvas context 540
resultados de cálculos de workers
armazenado em propriedade event.data 485
de Fractal Explorer workers (exemplo) 513

processando em Fractal Explorer
(exemplo) 514
recebendo resultados de workers 485, 498
resumo de pontos importantes 338
retângulos, desenhar em canvas 338
desenhar retângulos preenchidos 292
return, afirmações
em função body 117
funções sem 119
revisar camiseta, design aplicativo
implementação 296
RGB, valores de cores para pixels, processando
frame data de vídeo, 397, 410
rotate método, canvas context 540

S

Safari 20. *Ver também* navegadores
ferramentas de desenvolvedor para local
storage 434
H.264 formato de vídeo 352
HTML5 suporte 18
MP4/H.264 video 357
Quicktime player para mp4 vídeo 371
salvando e restaurando, contexto canvas 540
Scalable Vector Gráficos (SVG) 537
scope, variáveis 124
scratch buffer, processamento de vídeo com
390, 393
implementar buffer com canvas 395–398
<script> elementos 27
adicionar para HTML arquivo para chamar
para Twitter JSONP API
adicionar para HTML em <head> ou
<body> 53
criar e inserir dinamicamente 263, 267–269
especificar em HTML5 5
resgatar dados com 248–251, 257
script, injecção 271
scrollMapToPosition função (exemplo) 204
adicionar para aplicativo 205
<section> elemento 546
segurança, JSONP e 259
<select> elemento 301, 453
selectedIndex, propriedade, seleção de controle
de formulário 302
como funciona 303
Selectors API 542
separação de apresentação e conteúdo 326
selectors, novos, em CSS3 548

- serviços de hospedagem 230, 232
- serviços web 213
 - como funciona, Mighty Gumball (exemplo) 216
 - especificar função callback para 254
 - JSONP problemas de segurança e 259
 - lucky/unlucky serviço 224
 - maneiras para acessar, em API pública 271
 - mantendo uma conexão aberta com, usar Web Sockets para 539
 - parâmetros suportados 271, 274
 - receber JSON data de 233
 - usar JSONP com 253
 - usar XMLHttpRequest com 220
 - XMLHttpRequest, interdomínio problemas de segurança com 244
- servidores 230
 - criando seu próprio servidor web 231
 - movendo para servidor vivo 237
 - problema quando move para servidor vivo 242
- sessionStorage, objeto 460
- setAttribute método, elemento objeto 158, 274, 275
 - definir id sticky para sua chave única 450
 - usar para definir a id atributo 267, 269, 450
 - usar para definir o src atributo 267, 269
 - usar para set the class atributo 236, 257, 379, 430
- setEffect handler função, cabine de vídeo (exemplo) 378, 391
- setInterval método, janela objeto 263, 265
 - usar com Web Workers 523
- setItem método, localStorage objeto 418, 421
- setTimeout método, janela objeto
 - timeout parâmetro de 0 398
 - usar com Web Workers 523
 - usar para processar frame data de vídeo 397
- setupGraphics função (exemplo) 509
- setVideo handler função, cabine de vídeo (exemplo) 378, 387
- SGML 9
- shadowBlur propriedade, canvas context 335
- shadowColor propriedade, canvas context 335
- shadowing variáveis 126
- shadowOffsetX e shadowOffsetY propriedades, canvas context 335
- sinal de mais (+)
 - addition ou string concatenation operator 45
 - string concatenation operator 26
- single-thread modelo, JavaScript 474, 477
- breaking down 475
- Sinta-se como o Navegador – exercício
 - interpretando chamada para o método arc 317, 343
- <source> elemento
 - src atributo 359
 - type atributo 359
 - usar inside <video> elemento para cada formato de vídeo 358
- speed propriedade, coordinates objeto 190, 197
- splice método, Array objeto 449
- SQL, web 543
- src atributo
 - <script> elemento 53, 218, 249
 - atualizar com setAttribute 267
 - <source> elemento 358, 359
 - <video> elemento 353, 354
- startWorkers função (exemplo) 509, 510
- stickies, aplicativo (exemplo) 418, 428
 - adicionar “Add Sticky Note para Self” botão 431
 - adicionar JavaScript código 430
 - atualizar usuário interface para especificar cor 453–456
 - convertendo createSticky para usar um array 441
 - criar interface 429
 - deletar sticky de DOM 452
 - deletar sticky notes 446
 - design flaw 436
 - integrando código baseado em array 443
 - reescrivendo para usar um array 440
 - selecionando sticky note para deletar 450
- streaming de vídeo 403
 - tecnologias para 404
- string concatenation operator (+) 26, 45
- string expressões 43
- stringify método. Ver JSON.stringify método
- strings
 - acessar e enumerar objeto propriedades 133
 - em arrays 71
 - como index de array associativo 424
 - conversões para números em expressões 45
 - converter objetos para JSON string
 - formato 226
 - converter para decimais com parseFloat função 423
 - converter para números inteiros com parseInt função 423
 - criar a representação de string de um array 441, 467

o índice remissivo

- pares chave/valor armazenados em local storage 418
 - como objetos 159
 - type primitiva em JavaScript 40
 - receber de Web Workers com onmessage em event.
 - propriedade dados, 485
 - enviar para Web Workers com postMessage 484
 - strings vazias
 - comparando variáveis para 108
 - designando como valor à variável 26
 - procurando por 95
 - stroke método, canvas context 312
 - strokeText método, canvas context 328
 - <style> elemento
 - adicionar borda para canvas 288
 - CSS é estilo padrão 9, 31
 - style propriedade 455
 - subworkers 523
 - success handler, Geolocalização API 174, 175, 190
 - SVG (Scalable Vector Gráficos) 537
- T**
- table e table-cell layouts 548
 - tarefas, enviar e receber data de Web Workers (Fractal Explorer exemplo) 512
 - task monitor no OS X ou Windows 520
 - terminate método, worker objeto 522
 - text <input> elemento, value propriedade 94
 - verificando se o usuário entrou input 96
 - text, desenhar on canvas 325–332, 338
 - dividindo em linhas 335
 - drawText, função 345
 - mostrar entidades HTML 335
 - text métodos e propriedades em canvas API 328
 - textAlign propriedade, canvas context 328
 - alinhar texto do tuíte em design da camiseta, aplicativo (exemplo) 331
 - textBaseline propriedade, canvas context 329
 - texto, métodos e propriedades 328
 - Theora, formato de vídeo 357
 - third-party hosting services 230, 232
 - this (keyword) 144
 - adicionar para objeto movie (exemplo) 145
 - perguntas e respostas sobre 151
 - usar com construtores 147
 - usar com método calls 149, 151
- threading. *Ver também* Web Workers
- adicionar outro thread de controle 476
 - com Web Workers 478, 524
 - single-thread modelo, JavaScript 474
- time
- Date, objeto; getTime, método 442
 - milissegundos desde 1970 442
 - <time> elemento 547
- timeout opção 199, 201
- timestamp propriedade, posição objeto 190
- timeupdate event 398
- title propriedade, document objeto 157
- toDataURL método, canvas objeto 347
- toggle botões 380, 382
- tornando canvas visível, adicionar borda usando CSS 288
- tracking movements 192–198
- formulário para start e stop tracking 193
- traduzindo ou rotating canvas 540
- translate e rotate métodos, canvas context 540
- triangulação de sinal de celular 169
- triângulos, desenhar no canvas 311
- true e false (valores booleanos) 39
- try/catch afirmações, capturando exceções 458, 468
- TweetShirt App web (exemplo) 282
- adicionar tuítes para <select> elemento em <formulário> 323
- adicionar usuário interface formulário para HTML página 301
- criar aplicativo design 297
 - desenhar círculos 318
 - desenhar quadrados 304
 - desenhar texto 324, 327, 330, 331
 - desenhar uma imagem 333
- fazendo imagem de design para upload e impressão na camiseta 347
- formulário para aplicativo interface 298
- obter tuítes de Twitter 322
- preencher a cor de background 306
 - requerimentos e usuário interface 283
 - revisando plano de implementação 296
- Twitter JSONP API, fazendo chamada para 322
- type atributo
 - remoção de <link> e <script> tags 5
 - <source> elemento 359

U

ul.appendChild método 101
 underline (_), começando nomes de variáveis 40, 42
 updateSales função (exemplo) 230
 updateTuites callback função (exemplo) 323
 URL propriedade, objeto document 157
 URLs
 atualizar JSON URL para include last report time (exemplo) 275
 callback parâmetro 254
 setting up JSONP URL (exemplo) 267
 Web Socket 539
 alternativa para cache do navegador 272
 usar a superfície do display para vídeo 408
 usar elemento <canvas> para Fractal Viewer (exemplo) 503, 514
 UTF-8 9, 31

V

valores indefinidos 73
 retornados por funções sem indicação de retorno 121
 value atributo, text <input> elemento 95
 value propriedade, text <input> elemento 94
 value atributo versus 95
 values
 changing objeto valores de propriedade 133
 funções as 128, 129
 objeto valores de propriedade 132
 var palavra-chave 39
 variáveis
 chaining value de 39, 133, 141
 comparing para empty string 108
 declarando e designando valores 26, 38
 designando funções para 128
 local e global 123, 160
 nomeando 40, 42
 objetos designados para 136
 passando para funções 121
 scope de 124
 sombreamento 126
 vida curta de 125
 variáveis globais 123, 160
 ciclo de vida 125
 overuse em JavaScript 127
 razões para poupar uso de 127
 sombreamento 126
 variáveis locais 123, 160
 ciclo de vida de 125
 como a Geolocalização API determina isso 168
 Geolocalização API em JavaScript 166
 localização
 precisão de 191
 sombreamento de variáveis globais 126
 vector, fontes 329
 vector, gráficos vs. bitmap 336
 Veja bem!
 Código Pronto para assar não funciona em alguns navegadores 104
 deletar todos os itens de local store 435
 garantindo que o servidor está servindo arquivos de vídeo de forma correta
 Internet Explorer sem suporte Web Workers prior para IE10 482
 local storage e navegador problemas com arquivo:// 422
 MIME type 371
 navegadores sem suporte XMLHttpRequest's onload propriedade 229
 páginas servidas de arquivo://, restrições de segurança em Chrome 371
 pegando imagem de canvas, e code run de arquivo:// 347
 pushing navegador over local storage limit 459
 Quicktime para tocar vídeo mp4 em Safari 371
 rapid changes em vídeo suporte por navegadores 411
 restrições de segurança no Chrome
 impedindo execução de Web Workers do arquivo 482
 servidor requisitado para testar código de geolocalização em aparelhos móveis 179
 solução alternativa para navegadores sem suporte XMLHttpRequest's onload propriedade 241
 vídeo 16, 349–412
 adicionar informação de formato no elemento <source> 359
 alternando vídeos de teste 387
 aparência dos controles em diferentes navegadores 355
 Aponte Seu Lápis exercício
 control botões, toggle ou rádio 380, 382
 implementar lista de reprodução 364, 365
 western e ficção científica, efeitos 400, 410

o índice remissivo

- cabine (exemplo) 373
 - adicionar efeitos especiais 389–391
 - código para processar o vídeo 396
 - demo unit 374–376
 - escrever efeitos especiais 399–404
 - funções auxiliares 379
 - implementar controles de vídeo 384–386
 - obter vídeos demo prontos 383
 - processamento de vídeo usar scratch
 - buffer 393
 - setEffect e setVideo handlers 378
 - visão geral de processamento de vídeo 392
- canPlayType método, como funciona 369–375
- codecs 357, 358
 - coisas para se prestar atenção em 371
 - como funciona elemento <video> 353
 - error event, usar 406
 - errors 405
 - formatos 352, 356, 358
 - e possibilidade de padronização 360
 - hospedado na web 403
 - ideias para desenvolvimento posterior 407
 - Não existem perguntas idiotas 360, 371
 - navegador suporte, determinar nível de 361, 411
 - Palavras Cruzadas 409, 411
 - resumo de pontos importantes 408
 - streaming 403
 - testar para navegador suporte quando usar
 - código para carregar vídeo 368
 - usar JavaScript com HTML5 23
 - voltando ao player suportado 362
- Webville TV (exemplo) 350
 - canPlayType 369
 - construir com tecnologia HTML5 350
 - desenhando a lista de reprodução 365
 - handler para ended event para passar ao próximo vídeo 367
 - HTML5 página 351
 - implementar getFormatExtension função com
 - implementar nextVideo função 367
 - implementar lista de reprodução de vídeo 366
 - integrating getFormatExtension função 370
- <video> elemento
 - <objeto> elemento dentro 362
 - <source> elemento dentro 358
 - atributos 354, 408
 - como funciona 353
 - entrevista com 388
- métodos, propriedades e events 363
 - novo HTML5 elemento e API 351
 - parceria com <canvas> elemento 339, 388
- video objeto
 - acessar frame data 396
 - canPlayType método 368–374
 - error propriedade 406
 - load, método 385
 - loop propriedade 385
 - métodos, propriedades, e events 363
 - mute, propriedade 385
 - pause método 385
 - play método 366, 385
 - propriedades, métodos e events 408
 - src propriedade 366, 387
 - volume propriedade 360
 - vídeo progressivo 403
 - vídeo, processar com 392–394
 - viewport 354
 - vírgula (,), separar objeto, propriedades 132
 - volume propriedade
 - objeto áudio 533
 - objeto vídeo 360
 - Vorbis audio codec 357
 - VP8 video codec 357
- W**
- W3C 20
- WampServer 231
- watchId variable (exemplo) 194
- watchLocation função (exemplo) 194
- watchPosition método, geolocalização objeto 190, 192, 194, 207
 - controle de atualizações de localização 197
 - muitas chamadas para displayLocation 206
- wav formato de áudio 533
- .webm formato de arquivo vídeo 352, 357
- Web Platform Installer (Microsoft) 231
- Web Sharing (Mac) 231
- Web Sockets 539
- Web SQL 543
- Web Storage API 108, 418. *Ver também* local storage;
- Web Workers 16, 473–530
 - adicionar outro thread de controle 476
 - adicionar workers para jogo de pingPong (exemplo) 491, 492
- Aponte Seu Lápis exercício

- código assado para workers, cálculo de
 - Mandelbrot Set 504–507
- como eles funcionam 478
- como os workers tornam seus aplicativos mais rápidos 500
- construir explorer para Mandelbrot Set 494
- construir Fractal Explorer aplicativo
 - (exemplo) 503, 509
- criar 483
- criar e distribuir tarefas para 508
- eliminando 522
- enviar a mensagem para o worker 484
- escrever worker, message handler 486
- Explorer (exemplo) 515
- gerenciar gerações de fractais em Fractal Viewer 518
- handling evento de clique para zoom in no canvas em Fractal
- handling errors em workers 522
- implementar em Fractal Explorer aplicativo 511
- importScripts global função 493
- Não existem perguntas idiotas 491
- navegadores, suporte de 482
- número de workers
 - efeitos em performance 520
 - limites no 501
- obter workers iniciados em Fractal Explorer, aplicativo 510
- Palavras Cruzadas 525, 528
- por que os workers não podem acessar o DOM 480
- processando resultados dos workers em Fractal Explorer 514
- receber a mensagem do worker 485
- reescrever pseudocódigo para usar workers 502
- resultados de computações de workers 513
- resumo de pontos importantes 524
- Sinta-se como o Navegador exercício 488, 526
- subworkers 523
- tarefas para workers do Fractal Explorer 512
- tipos de dados que podem ser enviados para os workers 484
- usando workers compactos 490, 527
- usar importScripts para fazer JSONP requests 523
- usar multiple workers para contar Mandelbrot Set 497–500
- usos em potencial para workers 481
- WebKit-based navegadores 20. Ver também navegadores HTML5 suporte 18
- WebM/VP8 formato video container 357
- while loops 46
 - avaliando (exemplo) 48
 - decidindo entre loops e for 47
 - exemplo em JavaScript 26
 - if/else afirmações em 50
- width e height atributos
 - <canvas> elemento 286
 - definindo uso CSS 289
 - <video> elemento 353, 354
- WiFi posicionamento 169
- window, objeto 154
 - como objeto global 156, 158
 - criar onload event handler para 64, 75, 159
 - document objeto propriedade 155
 - localStorage, propriedade 422
 - Location, propriedade 347
 - onload, propriedade 64, 156
 - propriedades e métodos 155
 - setInterval, método 155, 265
 - setTimeout, método 155, 397
- windows
- Windows systems
 - instalando servidor web em 231
 - garantindo que o servidor está servindo vídeo com correção MIME type 371
 - task monitor 520
- worker, objeto. Ver também Web Workers
 - close, método 522
 - criar 483
 - criar e usar multiple 491, 492
 - onerror, propriedade 522
 - onmessage, propriedade 485
 - postMessage, método 484
 - subworkers 523
 - terminate, método 522

X

- XHTML 9, 536
 - problemas com 11
- XML
 - JSON versus 226, 271
 - SVG gráficos 537
 - usos de XHTML para 536
- XMLHttpRequest objeto 220, 239
 - acessar response text 222
 - alternativa para navegadores sem suporte Level 2 241
 - Conversa na lareira com JSONP 260

o índice remissivo

cross-domínio solicitações, problemas de segurança com 244, 277
entrevista com 225, 240
Level 2 240
onload handler, função 229
Palavras Cruzadas 278, 280
quando usar 246, 277

resgatando dados JSONP com 233
solicitações feitas por workers 491
servidor solicitado para uso de 230

Z

zoom in no canvas em Fractal Viewer (exemplo) 515

Colofão*



Todos os layouts do interior foram desenvolvidos por Eric Freeman e Elisabeth Robson.

Kathy Sierra e Bert Bates criaram a aparência e a sensação da série Use a Cabeça!. O livro foi produzido usando Adobe InDesign CS e Adobe Photoshop CS, e composto usando as fontes Uncle Stinky, Mister Frisky (você acha que estamos brincando), Ann Satellite, Baskerville, Comic Sans, Myriad Pro, Skippy Sharp, Savoye LET, Jokerman LET, Courier New e Woodrow.

O design interior e a produção foram feitos exclusivamente em Macintoshes da Apple — dois Mac Pros e dois MacBook Airs, para ser mais preciso.

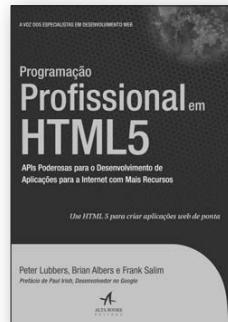
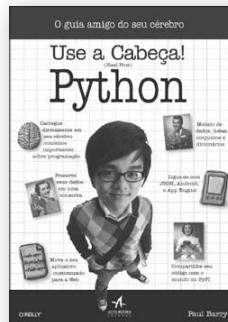
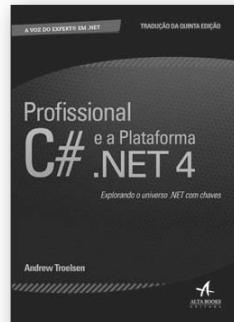
Locais onde foi escrito: Bainbridge Island, Washington; Portland, Oregon; Las Vegas, Nevada; Porto de Ness, Escócia; Seaside, Flórida; Lexington, Kentucky; Tucson, Arizona; e Anaheim, Califórnia. Longos dias de escrita foram abastecidos por cafeína da Honest Tea, GT's Kombucha e as músicas de Sia, Sigur Ros, Tom Waits, OMD, Phillip Glass, Muse, Eno, Krishna Das, Mike Oldfield, Audra Mae, Devo, Steve Roach, Beyman Brothers, Pogo, todo mundo da turntable.fm e muito mais músicas dos anos 1980 que vocês possam imaginar.

Você não conhece o site? Temos as respostas para algumas das questões deste livro, guias sobre como se aprimorar e atualizações diárias no blog dos autores! O conteúdo está em inglês.

Isto não é um adeus
Traga seu cérebro para
wickedlysmart.com



Conheça alguns de nossos outros livros sobre informática

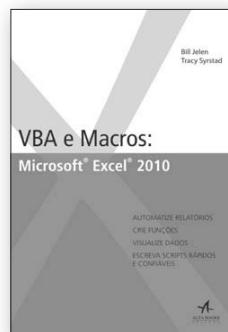


Todas as imagens são meramente ilustrativas



ALTA BOOKS
EDITOR A

- Idiomas
- Culinária
- Informática
- Negócios
- Guias de Viagem
- Interesse Geral



Visite também nosso site para conhecer
lançamentos e futuras publicações!

www.altabooks.com.br



/alta_books



/altabooks

The screenshot shows the Alta Books website. The homepage features a search bar and navigation links like 'Lançamentos', 'Blog', 'Institucional', 'Autôres', 'Fale Conosco', 'Sala de Imprensa', and 'Login'. Below is a book thumbnail for 'Como Avaliar Sua Fazenda' with a brief description. The news section includes a headline about the book being a best-seller in The New York Times.

Seja autor da Alta Books

Todo o custo de produção fica por conta da editora e você ainda recebe direitos autorais pela venda no período de contrato.*

Envie a sua proposta para autoria@altabooks.com.br ou encaminhe o seu texto** para:
Rua Viúva Cláudio 291 - CEP: 20970-031 Rio de Janeiro

*Caso o projeto seja aprovado pelo Conselho Editorial.

**Qualquer material encaminhado à editora não será devolvido.



