

Aplicações web com Yeoman e Polymer
Conheça os principais recursos dos
dois frameworks na prática

BOOTSTRAP E jQUERY

CONSTRUINDO UM PORTAL WEB RESPONSIVO



Jogo HTML5
Use Canvas para desenvolver
um jogo multiplataforma

Web Components
Aprenda a criar componentes
personalizados na web

MVP

R\$ 1.000.000,00
INVESTIDOS EM CONTEÚDO
NOS ÚLTIMOS 12 MESES.

APLIQUE ESSE INVESTIMENTO
NA SUA CARREIRA...

E MOSTRE AO MERCADO
QUANTO VOCÊ VALE!

CONFIRA TODO O MATERIAL
QUE VOCÊ TERÁ ACESSO:

- + de **9.000** video-aulas
- + de **290** cursos online
- + de **13.000** artigos
- DEVMEDIA API's consumido + de **500.000** vezes

POR APENAS
R\$ 69,90* mensais

*Tempo mínimo de assinatura: 12 meses.



PRA QUEM QUER EXIGIR
MAIS DO MERCADO!

 **DEVMEDIA**

EXPEDIENTE

Editor

Diogo Souza (diogosouzac@gmail.com)

Consultor Técnico

Daniella Costa (daniella.devmedia@gmail.com)

Produção

Jornalista Responsável Kaline Dolabella - JP24185

Capa e Diagramação Romulo Araujo

Atendimento ao leitor

A DevMedia possui uma Central de Atendimento on-line, onde você pode tirar suas dúvidas sobre serviços, enviar críticas e sugestões e falar com um de nossos atendentes. Através da nossa central também é possível alterar dados cadastrais, consultar o status de assinaturas e conferir a data de envio de suas revistas. Acesse www.devmedia.com.br/central, ou se preferir entre em contato conosco através do telefone 21 3382-5038.

Publicidade

publicidade@devmedia.com.br – 21 3382-5038

Anúncios – Anunciando nas publicações e nos sites do Grupo DevMedia, você divulga sua marca ou produto para mais de 100 mil desenvolvedores de todo o Brasil, em mais de 200 cidades. Solicite nossos Media Kits, com detalhes sobre preços e formatos de anúncios.

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique à vontade para entrar em contato com os editores e dar a sua sugestão!

Se você estiver interessado em publicar um artigo na revista ou no site Java Magazine, entre em contato com o editor, informando o título e mini-resumo do tema que você gostaria de publicar:



DIOGO SOUZA

diogosouzac@gmail.com

Analista de Sistemas Java na Indra Company e já trabalhou em empresas como Instituto Atlântico e Ebix L.A. É instrutor Android, palestrante em eventos sobre Java e o mundo mobile e consultor DevMedia. Conhecimentos e experiências em diversas linguagens e ferramentas de programação e manipulação de dados, bem como metodologias úteis no desenvolvimento de Sistemas diversificados.

Sumário

Artigo no estilo Curso

04 – Criando um Portal Web com Bootstrap e jQuery - Parte1

[*Madson Aguiar*]

Conteúdo sobre Boas Práticas

16 – Construindo aplicações web com Yeoman e Polymer

[*Julio Sampaio*]

Artigo no estilo Curso

28 – Jogos em HTML5: Criando um jogo 2D com Canvas – Parte 2

[*Julio Sampaio*]

Conteúdo sobre Boas Práticas

41 – Web Components na prática

[*Gustavo Corrêa Alves*]

Criando um Portal Web com Bootstrap e jQuery

- Parte 1

Domine os recursos do Bootstrap com jQuery implementando um portal web completo

ESTE ARTIGO FAZ PARTE DE UM CURSO

É sabido que o desenvolvimento web envolto pelas “n” tecnologias, frameworks e ferramentas acarreta uma necessidade de atualização deveras constante. Seja em projetos pequenos ou de grande porte, o desenvolvedor front-end na maioria dos casos deve ter um conhecimento bem aprofundado em tecnologias como HTML5, CSS3, JavaScript e Web Services, por exemplo, além de saber como utilizar frameworks como Bootstrap, jQuery, jQueryUI, dentre outros, levando em consideração que tais frameworks estão em constante atualização, desta forma o desenvolvedor tem que saber a diferença entre as versões já em uso e buscar se atualizar para as próximas, seja uma versão beta ou apenas notas sobre as novidades das que virão em um arquivo README qualquer.

Outro ponto importante é saber o que usar em um projeto e qual versão de determinado framework será inserido no mesmo, pois em se tratando de mobilidade existe a preocupação de tornar o website responsivo, ou seja, permitir que o layout de seu site se adeque ao tamanho da tela do dispositivo em que o usuário esteja usando, seja um smartphone, tablet ou mesmo um PC. Logo, é importante saber, antes de adotar um framework para um projeto, se o mesmo trata os recursos da responsividade, o que pode ser observado juntamente à documentação do framework. Veja na **Figura 1** o portal da DevMedia proporcionando a responsividade de layout para diferentes dispositivos de seus usuários e assinantes MVP.

Fique por dentro

Neste artigo será realizado um embasamento teórico e prático sobre as tecnologias utilizadas no front-end, neste caso Bootstrap e jQuery, permitindo que o leitor se adapte a situações de âmbito real, através da construção de um PDV (portal) Web completo, em conjunto com as tecnologias PHP para o back-end da aplicação, e MySQL para lidar com o salvamento das informações no banco dados. Exploraremos ao máximo os recursos de ambos os frameworks com exemplos os mais próximos da realidade possível, tratando de focar sempre na integração entre todas as referidas tecnologias.

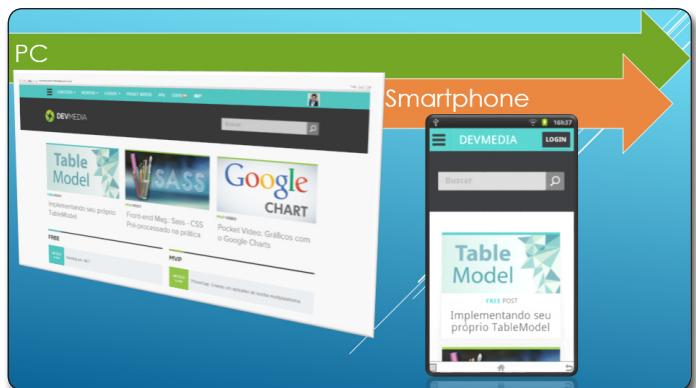


Figura 1. Portal DevMedia com layout responsivo.

Bootstrap

O Bootstrap, ou Twitter Bootstrap, é um dos frameworks open source para front-end mais conhecidos e usados em projetos web em todo o mundo, conquista essa devida principalmente à sua produtividade. Ele foi lançado em 2011 e ganhou popularidade

pela facilidade de uso para criar interfaces robustas para a web e por ser de fácil manutenção.

O Bootstrap foi idealizado e criado por membros da equipe de desenvolvedores do Twitter, especificamente encabeçando o projeto Mark Otto e Jacob Thornton. Cansados de utilizar várias bibliotecas em projetos internos que acarretaram em várias inconsistências e inúmeras horas de manutenção, eles resolveram então criar um framework que fornecesse um kit de ferramentas para projetos web de forma fácil e produtiva, garantindo assim um padrão nos projetos web de interface com elementos HTML bem estruturados e tipograficamente elegantes e agradáveis, ou seja, em conjunto com os elementos HTML e estrutura DOM são aplicados vários estilos CSS e até mesmo recursos em JavaScript.

No Bootstrap temos uma série de classes CSS que podemos utilizar facilmente com elementos divs e outros elementos da HTML. Além disso, o Bootstrap também pode ser considerado uma coleção de ferramentas que integra componentes do jQuery, facilitando, desta forma, o uso das classes CSS do Bootstrap em tags HTML para melhor organizar a estrutura do site, melhorar visualmente os componentes da página, usar recursos de responsividade e JavaScript em conjunto com CSS para criar efeitos elegantes e antenados com a web 2.0. Tudo isso é apenas parte do Bootstrap, mais adiante veremos sua estrutura e como utilizar seus componentes em um projeto web.

A equipe do Twitter abraçou o mundo mobile no Bootstrap, desta forma a partir da versão 2 já existiam várias funcionalidades do Bootstrap para o projeto de Web Design Responsivo. Veja na **Figura 2** o website do Bootstrap onde é possível realizar o download do framework.

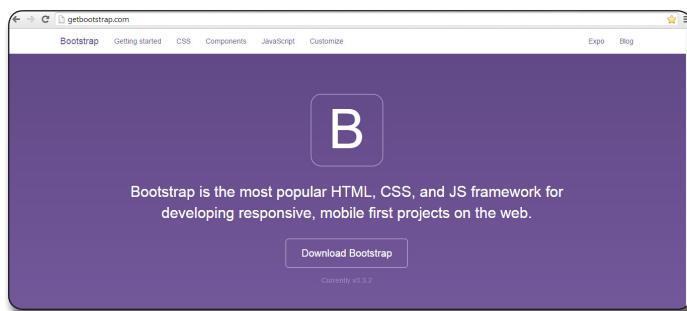


Figura 2. Website oficial do Bootstrap

O website do projeto contempla desde um tour pelos componentes e bibliotecas do framework, até templates prontos, tutoriais e o blog oficial. Um destaque especial vai para a opção "Customize" que apresenta vários componentes reutilizáveis construídos para fornecer iconografia, menus suspensos com dropdowns, botões agrupados, navegação, mensagens de alertas, grupos de inputs, paginação, badges, barras de progresso, painéis dentre outros componentes prontos para aplicar em contextos diversos de um projeto web.

Caso você prefira ter o primeiro contato com o Bootstrap no idioma português, poderá visitar a página de tradução do Bootstrap

feita pela Globo.com (seção **Links**). Só atenha-se ao fato da versão ser uma anterior à que usaremos neste artigo, mas será o suficiente para lidar com as implementações que faremos.

Criando um projeto com Bootstrap

Os templates do Bootstrap são uma mão na roda quando se trata de iniciar uma aplicação do zero e quando não temos nenhum modelo pronto de design em mente. Até mesmo os web designers que começam o desenvolvimento de um site usam o estilo padrão fornecido pelo Bootstrap para ter uma ideia mais rápida e prática de como o mesmo irá se comportar. No menu **Expo** do site oficial do framework, o leitor poderá encontrar links para sites prontos que fizeram uso do Bootstrap em suas construções. Em alguns deles é possível inclusive o reuso do estilo e estrutura usados.

Um fator importante na construção de um projeto front-end é criar uma espécie de mapa com a quantidade de páginas do seu website juntamente com a estrutura de layouts e elementos que irão compor o mesmo, pois a partir deste ponto você saberá se pode aproveitar um template pronto do Bootstrap ou se vai precisar criar seu próprio modelo, definindo a quantidade de containers bem como o sistema de grids a utilizar. A seguir veja os passos para iniciar um novo projeto com o Bootstrap.

Bootstrap CDN

Conforme apresentado na **Figura 2** você pode facilmente realizar o download da última versão do Bootstrap via site, porém existe outra forma de utilizar o mesmo em um projeto sem precisar ter os arquivos do framework fisicamente no servidor de hospedagem. Para isso, seria necessário fazer uso do **Bootstrap CDN** que é uma forma de referência aos arquivos do projeto através do MaxCDN (*Max Content Delivery Network*), um provedor de conteúdo via rede muito usado. Veja na **Listagem 1** como realizar este procedimento em seu projeto.

Listagem 1. Referências CDN para o projeto com Bootstrap.

```
01 <!DOCTYPE html>
02 <html lang="pt-BR">
03   <head>
04     <meta charset="utf-8">
05     <meta name="viewport" content="width=device-width, initial-scale=1">
06     <title>Bootstrap CDN</title>
07
08   <!-- Bootstrap links CDN-->
09   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css/bootstrap.min.css" />
10   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css/bootstrap-theme.min.css" />
11 </head>
12 <body>
13   <h1>Bootstrap CDN!</h1>
14
15   <!-- Dependência jQuery-->
16   <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
17   <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2/js/bootstrap.min.js"></script>
18 </body>
19 </html>
```

Vejamos alguns detalhes importantes presentes na listagem:

- **Linha 1:** Perceba a presença da tag DOCTYPE, que indica que esta página deve ser renderizada pelo browser seguindo os padrões da HTML5. Isso é importante pois o Bootstrap irá trabalhar com elementos dessa versão da HTML em específico.
- **Linha 5:** Aqui foi definido na tag meta o viewport, que, conforme documentação do Bootstrap, é necessário para tornar o layout do projeto responsivo.
- **Linhas 9 e 10:** Foi informado o CDN para a versão minificada dos arquivos CSS do Bootstrap através do MaxCDN.
- **Linha 16:** Nesta linha temos uma referência à versão minificada da biblioteca JS do jQuery que é uma dependência do Bootstrap.
- **Linha 17:** Referência CDN à biblioteca JS minificada do Bootstrap.

A utilização do CDN tem várias vantagens, como o download dos arquivos CSS e JS do Bootstrap que não serão realizados pelo seu servidor de hospedagem, auxiliando na diminuição do consumo de banda quando seu website for visualizado por uma quantidade relativamente alta de usuários. Uma desvantagem seria o fato de que se em algum momento você ficar sem internet no ambiente de desenvolvimento, não seria possível fazer os testes com o Bootstrap.

Configurando o Bootstrap

O primeiro passo é realizar o download do Bootstrap conforme link de download presente na seção **Links**. A versão que usaremos é a v3.3.2, ou seja, a versão mais recente no momento da escrita deste artigo. Após clicado no link para download, você será redirecionado para outra página que possibilita a escolha das seguintes opções conforme **Figura 3**, a saber:

- **Bootstrap:** A primeira opção possibilita realizar o download dos arquivos CSS e JavaScript compilados e minificados, ou seja, uma forma enxuta de usar o Bootstrap, uma vez que isso também ajudaria muito a reduzir o tráfego de dados via browser cliente e a sua aplicação hospedada em produção. Isso é possível graças à remoção dos espaços em branco e quebras de linha, o que torna o mesmo bem pequeno. É importante salientar que não serão incluídos quaisquer arquivos com documentação para consulta. Esta é a opção que usaremos no projeto deste artigo.

- **Source code:** Se o leitor preferir entender como foi estruturado e arquitetado o projeto a nível de código fonte, pode escolher esta opção para download, pois aqui você terá os arquivos de fontes e documentação do projeto juntos, porém é necessário um compilador para usar o projeto final. Neste caso, a equipe do Twitter usa o Grunt.js, um executor de tarefas automatizadas que gera builds através do código de projetos prontos, e cujo link para download você pode encontrar na seção **Links** deste artigo ou no próprio site do Bootstrap.

- **Sass:** Aqui você pode baixar o Twitter Bootstrap convertido de Less para Sass (**BOX 1**) e pronto, já pode usá-lo, inclusive com pré-compiladores em Rails, Compass e projetos Sass-only.

Download

Bootstrap (currently v3.3.2) has a few easy ways to quickly get started, each one appealing to a different skill level and use case. Read through to see what suits your particular needs.

Bootstrap

Compiled and minified CSS, JavaScript, and font files, along with our docs. [Requires a Less compiler and some setup.](#)

[Download Bootstrap](#)

Source code

Source Less, JavaScript, and font files, along with our docs. [Requires a Less compiler and some setup.](#)

[Download source](#)

Sass

Bootstrap ported from Less to Sass for easy inclusion in Rails, Compass, or Sass-only projects.

[Download Sass](#)

Figura 3. Opções de download do Bootstrap

BOX 1. Sass x Less

Ambos os frameworks podem ser entendidos como uma nova forma de escrever código em CSS, ou seja, com o uso de pré-processadores e frameworks de folhas de estilo para dar mais produtividade no desenvolvimento do código CSS principalmente no que diz respeito a repetição de uma mesma ação diversas vezes. Também pode ser feito o uso de variáveis em valores, propriedades e seletores, uma vez que você irá escrever arquivos com a extensão .sass e o mesmo será compilado para .css, para que você possa subir para o servidor.

Depois de realizar o download do Bootstrap e descompactá-lo, você terá uma estrutura de diretórios conforme a apresentada na **Figura 4**. Perceba que existe uma pasta chamada “css” para armazenar os arquivos de CSS com os nomes Bootstrap e Bootstrap-theme com estilo pré-definido para os componentes e temas, respectivamente, porém existem algumas diferenças entre eles: Bootstrap.css e Bootstrap-theme.css são arquivos css simples, Bootstrap.min.css e Bootstrap-theme.min.css são os arquivos CSS minificados, com tamanho reduzido, e os arquivos Bootstrap.css.map e Bootstrap-theme.css.map são usados em pré-processadores e editores Less; outra pasta chamada “js” para arquivos JavaScript e outra chamada “fonts” que armazena algumas fontes utilizadas pelo Bootstrap por padrão.

```
bootstrap/
└── css/
    ├── bootstrap.css
    ├── bootstrap.css.map
    ├── bootstrap.min.css
    ├── bootstrap-theme.css
    ├── bootstrap-theme.css.map
    └── bootstrap-theme.min.css
└── js/
    ├── bootstrap.js
    └── bootstrap.min.js
└── fonts/
    ├── glyphicons-halflings-regular.eot
    ├── glyphicons-halflings-regular.svg
    ├── glyphicons-halflings-regular.ttf
    ├── glyphicons-halflings-regular.woff
    └── glyphicons-halflings-regular.woff2
```

Figura 4. Estrutura de arquivos do Bootstrap

Bootstrap Containers

Para iniciar o uso do Bootstrap na prática você deverá criar um diretório para salvar as páginas HTML e os recursos necessários para o website. Chamaremos a pasta de "www_Bootstrap" e dentro da mesma devem ser extraídos os arquivos do Bootstrap contidos no zip baixado anteriormente. Feito isso, deve ser criado um novo arquivo HTML na raiz do projeto conforme apresentado na **Listagem 2** com o nome de Home.html. Veja na **Figura 5** a hierarquia de diretórios que você obedecer, em detrimento da pasta www do WampServer.

Listagem 2. Código HTML referente à página Home com uso de containers.

```
01 <!DOCTYPE html>
02 <html lang="pt-BR">
03   <head>
04     <meta charset="utf-8">
05     <meta name="viewport" content="width=device-width, initial-scale=1">
06     <title>Website com Bootstrap</title>
07     <link rel="stylesheet" href="Bootstrap-3.3.2-dist/css/Bootstrap.min.css">
08     <link rel="stylesheet" href="Bootstrap-3.3.2-dist/css/Bootstrap-theme_min.css">
09   </head>
10   <body>
11     <h3>h3 fora da class Container, somente dentro de body.</h3>
12     <div class="container">
13       <h3>h3 dentro da class Container.</h3>
14     </div>
15     <div class="jumbotron">
16       <h3>h3 dentro class Jumbotron.</h3>
17     </div>
18     <div class="container">
19       <div class="jumbotron">
20         <h3>h3 dentro class Jumbotron, e Jumbotron dentro de Container.</h3>
21       </div>
22     </div>
23     <div class="container-fluid">
24       <div class="jumbotron">
25         <h3>h3 dentro class Jumbotron, e Jumbotron dentro de
26           Container-fluid.</h3>
27       </div>
28     </div>
29     <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/
30       jquery.min.js"></script>
31     <script src="Bootstrap-3.3.2-dist/js/Bootstrap.min.js"></script>
32   </body>
33 </html>
```

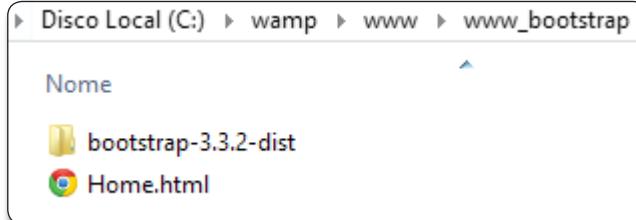


Figura 5. Estrutura de diretórios dentro do WampServer

Nós usaremos o aplicativo WampServer para hospedar o website em ambiente de desenvolvimento e também pelo fato de que iremos precisar do mesmo para interpretar os códigos PHP que serão usados na segunda parte deste artigo.

O WampServer é um pacote tudo em um, ou seja, você instala e junto vem o PHP, o servidor Apache, o PHPAdmin, o banco de dados MySQL, dentre outros aplicativos.

Já em relação ao conteúdo da listagem, temos algumas configurações novas a quem ainda não conhece o framework, a saber:

- **Linhas 7 e 8:** Referências aos arquivos CSS do Bootstrap.
- **Linha 11:** Foi inserido um elemento h3 para que se possa ter uma noção de como fica um elemento HTML solto dentro do body sem uso de um container.
- **Linhas 12 à 14:** É definida uma div onde é atribuída a propriedade class (a classe CSS container) e dentro da mesma é inserido outro h3 para realizar uma comparação no layout em relação à linha 11.
- **Linhas 15 à 17:** Aqui é definida uma div com o uso da classe jumbotron interna ao framework. Veja que ela diferente da classe container pois nela algumas propriedades do CSS como padding e background-color tem valores distintos, possibilitando a cor cinza no fundo e toda a largura da tela. Por fim, dentro desta mesma div foi colocado outro h3 para uso futuro.
- **Linhas 18 à 22:** Temos a presença de uma div de classe jumbotron dentro de outra div de classe container, e um h3 para mostrar o resultado e o efeito em relação às margens, então dessa forma você pode facilmente agrupar um container dentro de outro para obter o resultado esperado.
- **Linhas 23 à 27:** Nestas linhas foi utilizado outro container, neste caso o container-fluid e dentro do mesmo uma jumbotron e uma h3. Veja a diferença em relação ao agrupamento das linhas 18 à 22.
- **Linhas 28 e 29:** Aqui nós inserimos as referências aos arquivos JavaScript do Bootstrap e jQuery. Perceba que estas foram feitas apenas no final do documento em vez de dentro da tag head, isso porque com a referência ao final do documento obtemos uma melhor performance de carregamento dos elementos HTML. Além disso, como o browser já deve ter renderizado toda a página a essa altura, podemos acessar qualquer elemento HTML já carregado via JavaScript.

Quando se utiliza dos recursos do Bootstrap em um website é de extrema importância usar containers para agrupar conjuntos de elementos HTML, ou seja, através da tag div da HTML você pode definir na propriedade "class" o que foi apontado na classe de estilos CSS do Bootstrap para servir de containers de elementos. Veja a seguir algumas classes que podem ser utilizadas neste sentido:

- **Container:** A classe container do Bootstrap funciona como uma forma de recipiente para os elementos HTML do seu layout, tendo sua importância na estruturação do documento e se comportando de acordo com o tipo de dispositivo que está renderizando a página através do uso das Media Queries (**BOX 2**) do CSS, que definem a largura da tela. Conforme o trecho de código da **Listagem 3** (retirado do arquivo Bootstrap.css), pode ser vista parte da formação CSS pertencente à classe container.
- **Container-fluid:** A classe container-fluid pode ser considerada uma extensão de container para layouts fluidos, onde o próprio código é reorganizado por si só.

Criando um Portal Web com Bootstrap e jQuery - Parte1

Listagem 3. Techo de código retirado do arquivo Bootstrap.css que representa a classe container.

```
1564 .container {  
1565   padding-right: 15px;  
1566   padding-left: 15px;  
1567   margin-right: auto;  
1568   margin-left: auto;  
1569 }  
1570 @media (min-width: 768px) {  
1571   .container {  
1572     width: 750px;  
1573   }  
1574 }  
1575 @media (min-width: 992px) {  
1576   .container {  
1577     width: 970px;  
1578   }  
1579 }  
1580 @media (min-width: 1200px) {  
1581   .container {  
1582     width: 1170px;  
1583   }  
1584 }
```

BOX 2. Media Queries

As Media Queries são um recurso utilizado para identificar características do dispositivo que está sendo utilizado para renderizar as páginas HTML. Dessa forma, através do uso de Media Types podemos obter informações como a resolução do dispositivo, por exemplo, e em cima disso fazer com que o layout se comporte de acordo com o tamanho da tela, aplicando código CSS conforme a necessidade. Esta tecnologia proporciona parte dos recursos que precisamos para impor o responsive web design.

Nas **Figuras 6 e 7** é exibido o resultado das execuções da referida listagem em um navegador de PC e dispositivo móvel, respectivamente.

Sistema de Grids do Bootstrap

O novo sistema de grids do Bootstrap 3 teve mudanças significativas em relação à versão 2, principalmente pelo upgrade que tivemos com foco no desenvolvimento de websites responsivos, isto é, o Bootstrap agora idealiza o foco no desenvolvimento mobile primeiro para depois focar na adaptação para dispositivos com maior resolução.

O sistema de grids nada mais é que um recurso do Bootstrap para que você possa construir seu layout, uma forma de organizar os elementos HTML dentro de linhas e colunas. Antigamente eram usadas tabelas para a construção de layouts, porém com a padronização da Web e da estrutura de documentos HTML, elas hoje são usadas apenas para exibir conteúdo tabular.

Para entender o sistema de grids do Bootstrap você precisa pensar na janela de um browser como sendo uma estrutura de tabela, linhas e colunas (como no Excel, por exemplo), sendo possível adicionar várias linhas uma abaixo da outra e que cada linha possa ser dividida em uma ou várias colunas. Todavia, o Bootstrap trabalha com um total de 12 colunas por linha, caso você adicione mais que isso a linha sofrerá uma quebra para a próxima logo abaixo quebrando o padrão do seu layout.

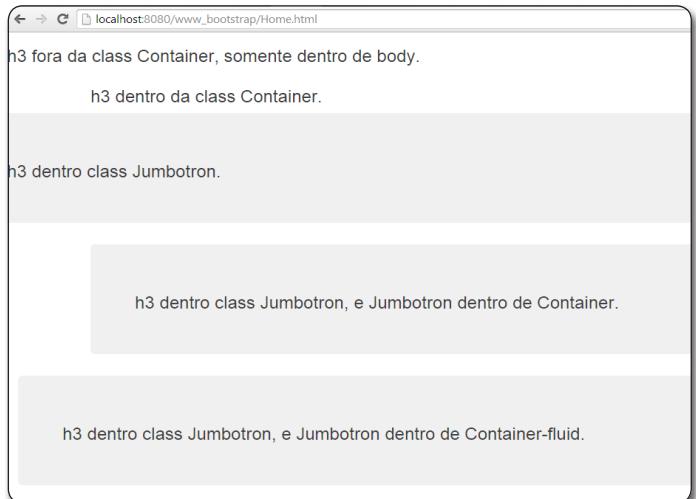


Figura 6. Resultado da execução visualizado pelo navegador em um PC

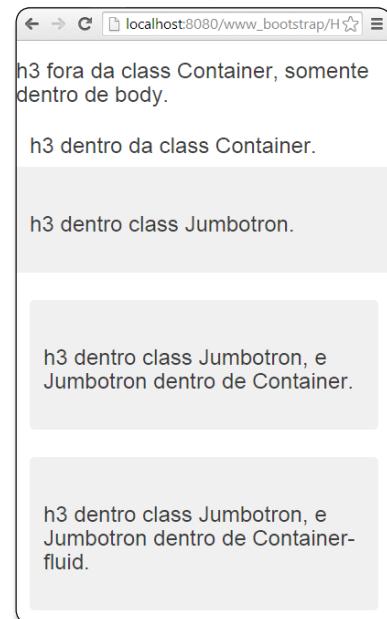


Figura 7. Resultado da execução visualizado em um dispositivo móvel

No fim, veremos que existem quatro classes CSS no Bootstrap responsáveis por dividir as colunas de uma linha observando o tipo de dispositivo através das media queries. Vejamos a seguir os passos para isso:

1. Primeiramente você precisa criar um container usando uma div com as classes container ou container-fluid que vimos para armazenar seu layout.
2. Com o container já criado, você pode ir inserindo a quantidade de linhas que o seu layout necessita, ou pode optar por ir criando uma linha de cada vez para ir acompanhando o resultado no browser. Para criar uma linha basta criar uma div e definir para esta a classe CSS **row** do Bootstrap, e dentro da linha dela dividir as colunas num total de 12.
3. Agora só precisa dividir as colunas pertencentes à linha criada no passo anterior. Para isso é importante imaginar dimensões

invisíveis que você acha que caberão no seu modelo. A conta é mais ou menos assim: imagine uma linha dividia em 12 pedaços na horizontal, então se você deseja criar um menu à esquerda e outro à direita, deverá dividir esses pedaços por 3 (o que chamamos de mescla), o que dará um total de 6 partes restantes para criar a área central do layout. Para fazer isso, precisamos usar umas das seguintes classes específicas do Bootstrap:

- a. **.col-xs-**: Esta classe deve ser utilizada quando se deseja personalizar as divisões de uma linha para dispositivos com resolução muito pequena (*xs - extra small*, ou super pequena) ou menor que 750px.
- b. **.col-sm-**: Usando esta classe você dividirá as colunas de uma linha para dispositivos pequenos (*sm - small*), com resolução entre 750px e 970px.
- c. **.col-md-**: Essa classe terá efeito em dispositivos médios (*md - medium*) com resolução entre 970 e 1170px.
- d. **.col-lg-**: Nesta, o resultado será apenas para resoluções acima de 1170px. Por exemplo: para dividir uma linha em duas partes usamos `<div class="col-lg-4">4 partes</div>` e `<div class="col-lg-8">8 partes</div>`, somando as 12 colunas em duas divisões com 4 e 8 partes mescladas. Perceba que após o hífen no final do nome da classe de divisão de coluna é onde você deve informar a quantidade de partes (ou colunas) que esta divisão vai ocupar na linha, uma espécie de mescla como é efeito em colunas do Excel, por exemplo.

Para uma melhor compreensão de quando cada classe para divisão de colunas deve ser utilizada, veja o conteúdo da tabela presente na **Figura 8**.

Para fixar os conceitos acerca do sistema de grids do Bootstrap vamos criar um layout utilizando uma grid que se comporte diferente de acordo com a resolução do dispositivo em que esteja sendo visualizada a página HTML, desta forma poderemos notar a diferença nas divisões de colunas usando as classes `col-md` e `col-sm` conforme apresentado nas **Figuras 9, 10 e 11**. Veja na **Listagem 4** o código que exemplifica o layout em questão.

A listagem retrata o layout fluido simples, porém com alguns detalhes que merecem destaque, a saber:

- **Linhas 9 à 10:** Definimos a classe `estilo1` com propriedades simples para alterar a cor de fundo e borda.
- **Linha 13:** Criamos um container, pois precisaremos dele para estipular as propriedades responsivas citadas.
- **Linhas 14, 18, 22 e 27:** Perceba que foram criadas várias divs e utilizada a classe `row` do Bootstrap várias vezes, pois estes elementos representam linhas no layout. Veja ainda que todas estão dentro do container e é dentro dele que você pode criar divs usando a classe `row` para assim dividir as colunas.
- **Linhas 15 e 16:** Como estas linhas estão contidas dentro de uma `row`, criamos duas divs para representar a divisão das colunas, neste caso em duas partes: na primeira div foram definidas na propriedade `class` as classes com os valores `col-md-2`, `col-sm-6`

Sistemas de grids - divisão de colunas em linhas.	Dispositivos muito pequenos Phones	Dispositivos Pequenos Tablets	Dispositivos Médios Desktops	Dispositivos muito grande PC/TV
Resolução	<750px	750px até 970px	970px até 1170px	> 1170px
Classe css	<code>.col-XS-</code>	<code>.col-sm</code>	<code>.col-Md-</code>	<code>.col-LG-</code>
Exemplos das classes em divs	<code>class="col-XS-4 col-XS-8"</code>	<code>class="col-sm-4 col-sm-4 col-sm-4"</code>	<code>class="col-Md-6 col-Md-6"</code>	<code>class="col-LG-12"</code>
Nº de colunas			12	
A largura da coluna	Automático	~ 62px	~ 81px	~ 97px

Figura 8. Classe CSS do Bootstrap para divisão de colunas por resolução e dispositivo

Listagem 4. Exemplo de layout utilizando o sistema de grids do Bootstrap.

```

01 <!DOCTYPE html>
02 <html lang="pt-BR">
03 <head>
04   <meta charset="utf-8">
05   <meta name="viewport" content="width=device-width, initial-scale=1">
06   <title>Sistema de grids</title>
07   <link rel="stylesheet" href="Bootstrap-3.3.2-dist/css/Bootstrap.min.css">
08   <link rel="stylesheet" href="Bootstrap-3.3.2-dist/css/Bootstrap-theme.min.css">
09   <style rel="stylesheet" type="text/css">
10     .estilo1{ background-color: lightgreen; border: 1px solid; } </style>
11 </head>
12 <body>
13   <div class="container">
14     <div class="row">
15       <div class="col-md-2 col-sm-6 estilo1">LOGO </div>
16       <div class="col-md-10 col-sm-6 estilo1">MENU</div>
17     </div>
18     <div class="row">
19       <div class="col-md-6 col-sm-4 estilo1">SLIDESHOW <br><br><br>
20       </div>
21       <div class="col-md-6 col-sm-8 estilo1">BANNER <br><br><br>
22     </div>
23     <div class="row">
24       <div class="col-md-3 col-sm-4 estilo1">
25         ESQUEDA <br><br><br><br><br>
26       <div class="col-md-6 col-sm-6 estilo1">
27         CENTRAL <br><br><br><br><br>
28       <div class="col-md-3 col-sm-2 estilo1">
29         DIREITA <br><br><br><br><br>
30     </div>
31   <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
32   <script src="Bootstrap-3.3.2-dist/js/Bootstrap.min.js"></script>
33 </body>
34 </html>

```

que representam o formato e quantidade de colunas destinadas a ocupar duas e seis colunas, respectivamente; e `estilo1` que na realidade faz parte da divisão da mesma linha, o que muda é a quantidade de colunas utilizadas na visualização dos menus. Para entender melhor como irá se comportar a linha veja que na **Figura 9** a logo assume menos colunas que o menu, já na **Figura 10** tanto a logo quanto o menu tem a mesma quantidade de colunas.

• **Linhas 19 e 20:** Perceba que estas linhas seguem o mesmo raciocínio utilizado nas linhas 15 e 16, o que muda agora é o fato de ser

uma nova linha para separar o slideshow e banner, a quantidade de colunas em ambos muda para cada tipo de dispositivo.

- **Linhas 23 a 25:** Aqui o interessante é que uma linha foi dividida em três divs, possibilitando ter dois painéis, um à esquerda e outro à direita, os dois com o mesmo tamanho. Por fim, temos a área central com uma div definindo col-md-6, col-sm-6 e estilo1 na propriedade class.

- **Linhas 28:** Para finalizar este exemplo de layout, nesta linha foi definido o rodapé com apenas uma div setando o valor na class col-md-12 e estilo1, ou seja, esta divisão ocupa toda a largura da tela com 12 colunas.

É interessante notar que a tela da **Figura 11** não tem um comportamento específico, pois não foi definido nada para a classe col-xs-.

Sendo assim, temos apenas o comportamento do layout fluido do Bootstrap que coloca uma divisão abaixo da outra para obter uma melhor visualização do layout em dispositivos menores.

Tipografia do Bootstrap

A tipografia no Bootstrap é tudo que está relacionado com elementos textuais de um website, ou seja, títulos, textos, parágrafos, listas dentro de listas, etc. O Bootstrap por padrão já zera o CSS colocado pelos browsers no HTML, assim ele aplica outra formatação a elementos como as tags `<body>` e `<p>` que recebem outro tamanho de fonte e margem. Isso já ocorre quando são realizadas as referências ao arquivo CSS do Bootstrap, porém os ganhos não param por aí: existem várias classes CSS que podem ser utilizadas juntamente com elementos HTML para formar textos e obter uma melhor visibilidade e legibilidade. Na **Listagem 5** serão apresentados vários elementos presentes na tipografia do Bootstrap.

Sobre a listagem, podemos defini-la da seguinte forma:

- **Linhas 12 à 17:** Aqui temos a definição de títulos fazendo uso dos diferentes tamanhos de texto com `h1` ao `h6`, perceba ainda que está sendo utilizada a tag `<small>` em cada um dos seis títulos para exibir um texto secundário um pouco menor que o título principal.

- **Linha 18:** Esta linha apresenta um parágrafo utilizando a classe `lead` do Bootstrap para dar um destaque a mais neste texto; você poderia fazer uso da classe `lead` para exibir um “Leia mais”, por exemplo. Perceba também que foi usado o elemento `<mark>` para destacar parte do texto deixando a cor de fundo em amarelo.

- **Linha 19:** Quando for necessário excluir parte do texto de um parágrafo, você pode fazer uso da tag ``, desta forma o texto fica em forma de taxado.

- **Linha 20:** Esta linha é o inverso da linha 19, ou seja, você irá usar tag `<ins>` quando desejar inserir algo a mais em um texto. Veja o resultado deste exemplo na **Figura 12**, o texto fica sublinhado, porém o `ins` não serve apenas para colocar um



Figura 9. Visualização em dispositivo acima de 1170px

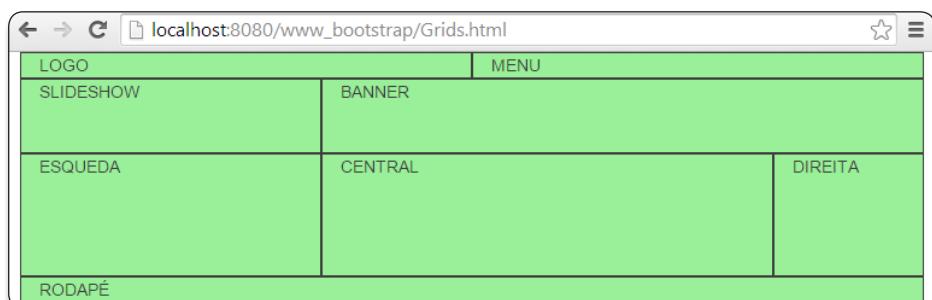


Figura 10. Visualização em dispositivo entre 750px e 970px

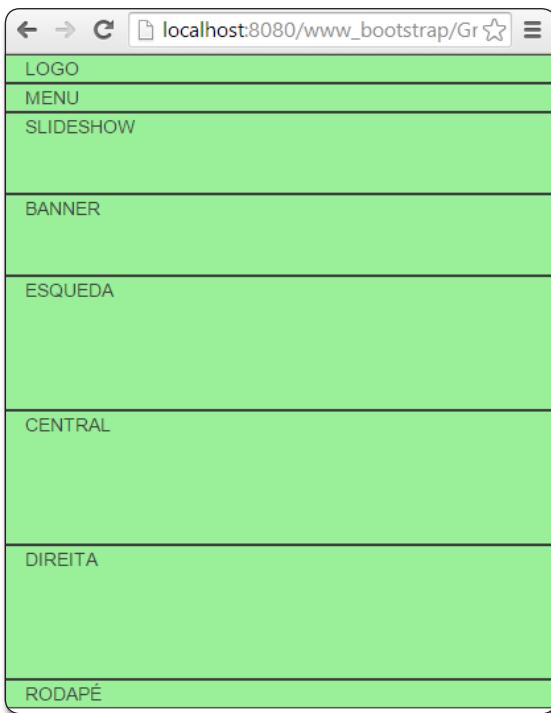


Figura 11. Visualização em dispositivo menor que 750px

Listagem 5. Exemplos de uso da tipografia com classe CSS do Bootstrap e elementos HTML.

```

01 <!DOCTYPE html>
02 <html lang="pt-BR">
03 <head>
04 <meta charset="utf-8">
05 <meta name="viewport" content="width=device-width, initial-scale=1">
06 <title>Tipografia Bootstrap</title>
07 <link rel="stylesheet" href="Bootstrap-3.3.2-dist/css/Bootstrap.min.css">
08 <link rel="stylesheet" href="Bootstrap-3.3.2-dist/css/Bootstrap-theme.min.css">
09 </head>
10 <body>
11   <div class="container">
12     <h1>Título com h1 <small>Texto secundário.</small></h1>
13     <h2>Título com h2 <small>Texto secundário.</small></h2>
14     <h3>Título com h3 <small>Texto secundário.</small></h3>
15     <h4>Título com h4 <small>Texto secundário.</small></h4>
16     <h5>Título com h5 <small>Texto secundário.</small></h5>
17     <h6>Título com h6 <small>Texto secundário.</small></h6>
18     <p class="lead">Aqui temos um parágrafo com texto destacado com uso
19       da classe <mark>"lead e elemento mark"</mark>.</p>
20     <p>Excluindo parte de um texto: <del>Este texto foi excluído usando o
21       elemento "del".</del></p>
22   </div>
23 </body>
24 </html>

```

sublinhado num texto, mas também é usado em mecanismos de busca para destacar esta inclusão.

- **Linhas 21 à 25:** Aqui as classes CSS text-left, text-justify, text-center, text-right e textnowrap são utilizadas em parágrafos para centralizar o texto em diferentes posições.
- **Linhas 26 à 28:** Caso precise deixar um texto por completo em caixa alta, baixa ou apenas com as primeiras letras em maiúsculo você pode usar as classes text-uppercase, text-lowercase e text-capitalize, respectivamente.

Configurando o jQuery

O jQuery constitui um pequeno arquivo que pode ser facilmente baixado no endereço presente na seção **Links** deste artigo.

Nesta página é possível baixar as versões 1.x e 2.x do jQuery, assim como utilizar o CDN conforme foi demonstrado com o Bootstrap, ou até mesmo utilizar as referências do servidor do Google também demonstrado no mesmo exemplo. Selecione a versão 2.x com a opção de arquivo não minificado para permitir uma melhor análise da própria biblioteca. Observe que ao clicar no link de download é provável que o código seja exibido direto no browser, desta forma você pode copiar e salvar um arquivo com extensão .js conforme apresentado na **Figura 13**, que mostra a pasta do projeto jQuery dentro do Bootstrap.

jQuery na prática

Antes de iniciar a prática com o jQuery é importante saber como utilizá-lo, pois existem formas diferentes de recuperar os elementos HTML e executar as funções JavaScript da biblioteca, dados os diferentes ambientes e frameworks onde o mesmo irá atuar. Podemos optar por duas formas:

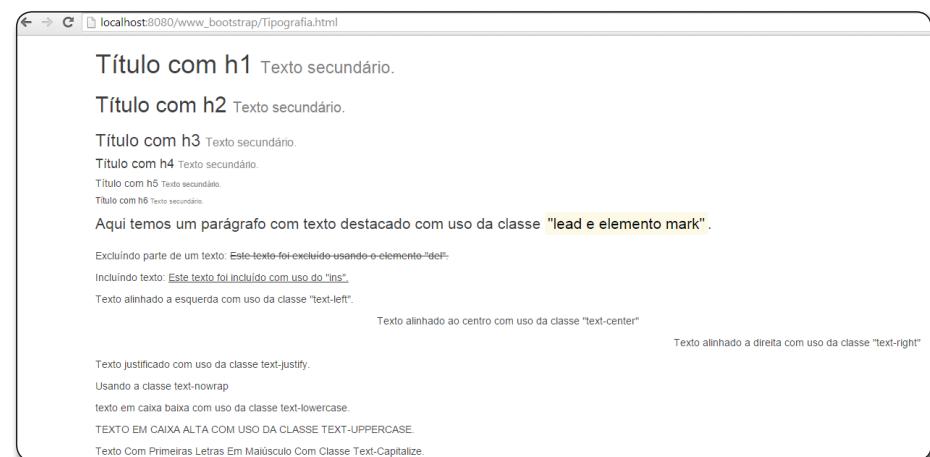


Figura 12. Resultado execução do exemplo sobre tipografia Bootstrap

Disco Local (C:) ▶ wamp ▶ www ▶ www_bootstrap ▶ jquery		
Nome	Tipo	Tamanho
jquery-2.1.3.js	Arquivo JS	251 KB
jquery-2.1.3.min.js	Arquivo JS	83 KB

Figura 13. Arquivos js do jQuery no Bootstrap

1. Usar `$(nome_elemento_html)`: construtor padrão do framework;
2. Usar `jQuery(nome_elemento_html)`: construtor sobre carregado explicitamente.

Analisemos, portanto, o exemplo apresentado na **Listagem 6**. Vejamos os detalhes presentes nesta codificação:

- **Linha 7:** Nesta linha é realizada a referência direta à biblioteca do jQuery, ou seja, os arquivos js. Você também pode substituir pela referência ao CDN aqui nesta linha se tiver conexão com a internet.

Criando um Portal Web com Bootstrap e jQuery - Parte1

- **Linhas 8 e 12:** Temos a definição de um bloco JavaScript dentro do documento HTML, o que não é boa prática. A partir da HTML5 a definição de atributos do tipo type, por exemplo, não é mais obrigatória.
- **Linha 9:** Aqui é usado o operador \$() do jQuery, passando como parâmetro o objeto document (referência da página HTML por completo – DOM - com todos os seus elementos). Depois temos a chamada à função **ready**, que diz que o script só deve ser executado com determinado trecho de código JavaScript apenas quando o browser tiver renderizado todo o documento HTML (semelhante à função load da tag body). Por último, é criada uma função anônima através do comando function() e, dentro da mesma, a exibição de uma mensagem de alerta simples. Veja o resultado da execução na **Figura 14**.

Listagem 6. Exemplo de uso da biblioteca jQuery

```
01 <!DOCTYPE html>
02 <html lang="pt-BR">
03 <head>
04   <meta charset="utf-8">
05   <meta name="viewport" content="width=device-width, initial-scale=1">
06   <title>Exemplo jQuery</title>
07   <script type="text/javascript" src="jquery/jquery-2.1.3.min.js"></script>

08   <script type="text/javascript">
09     $(document).ready(function() {
10       alert("Mensagem de alerta fazendo uso da jQuery");
11     });
12   </script>
13 </head>
14 <body>
15 </body>
16 </html>
```

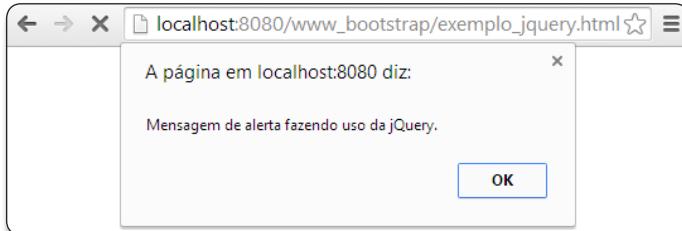


Figura 14. Resultado de execução do comando simples de alerta

Funções de estilo CSS com jQuery

No nosso projeto, faremos uso constante de funções de estilo para impor design aos elementos HTML de forma dinâmica e muitas vezes por intermédio de uma linguagem de programação *server side*. Ações como aplicar formatação, cor, alteração de margens, dentre outras opções, podem ser realizadas via funções de estilo. Para isso, podemos usar algumas funções do jQuery como css(), addClass(), add(), removeClass() e toggleClass(). A seguir na **Listagem 7** é demonstrado como utilizar algumas destas funções.

Vejamos alguns detalhes sobre este trecho de código:

- **Linhas 8 à 12:** Aqui foi inserida a classe estilo1 criada anteriormente no exemplo de uso do Bootstrap. Perceba que este código

apenas altera a cor de fundo e aplica uma borda. Além disso, a partir daqui começamos a mesclar ambos os frameworks para ver o poder dos dois juntos.

- **Linha 23:** Nesta linha é definido um parágrafo e informado o valor “paragrafo_1” para seu atributo id.
- **Linha 24:** É criado um botão para que possamos capturar o evento click (também via funções jQuery) e em cima disso realizar as formatações.
- **Linha 16:** Aqui é selecionado o elemento HTML pelo nome informado no atributo id, foi utilizado o # antes do nome e logo após é chamada a função click para monitorar os cliques no botão btn_alterar.
- **Linha 17:** Nesta linha é selecionado o elemento que se deseja alterar a formatação, neste caso o paragrafo_1, logo em seguida é chamada a função css() e passado como parâmetro o par de chave e valor com as propriedades de estilo que se deseja alterar.
- **Linha 18:** Aqui temos uso da função addClass(), que aplica ao objeto selecionado todo a formatação presente em uma classe CSS, esta que deve ser passada por parâmetro na mesma função.

Listagem 7. Aplicando formatação CSS a elementos com jQuery.

```
01 <!DOCTYPE html>
02 <html lang="pt-BR">
03 <head>
04   <meta charset="utf-8">
05   <meta name="viewport" content="width=device-width, initial-scale=1">
06   <title>Exemplo CSS jQuery</title>
07   <script type="text/javascript" src="jquery/jquery-2.1.3.min.js"></script>
08   <style rel="stylesheet" type="text/css">
09     .estilo1{
10       background-color: lightgreen;
11       border: 1px solid;
12     }
13   </style>
14   <script type="text/javascript">
15     $(document).ready(function() {
16       $("#btn_alterar").click(function(){
17         $("#paragrafo_1").css("color","blue")
18         .addClass("estilo1");
19       });
20     </script>
21   </head>
22   <body>
23     <p id="paragrafo_1">Estilos CSS com jQuery.</p>
24     <button id="btn_alterar">Altera cor de fundo.</button>
25   </body>
26 </html>
```

O leitor deve ter percebido que as divisões de responsabilidades para cada framework até agora já tomam uma certa forma, onde temos o Bootstrap focando principalmente na estilização e componentização dos elementos HTML, enquanto o jQuery foca em dinamizar todo esse conteúdo, inclusive o próprio CSS. Veja na **Figura 15** o resultado da execução dessa listagem, isto é, antes e depois do clique no botão btn_alterar.



Figura 15. Aplicando estilos CSS com jQuery

Ocultando e exibindo elementos HTML com jQuery

O ocultamento de elementos HTML em páginas web é algo muito útil quando desejamos carregar todo o conteúdo de uma requisição, mas não exibir tudo de uma vez. Isso serve, inclusive, para diminuir o tráfego de dados na rede e aumentar a performance da aplicação para fins de usabilidade, SEO, etc.

Existem duas funções bastante úteis no jQuery para controlar a visibilidade de elementos na janela do browser: `hide()` para ocultar um elemento e `show()` para tornar o mesmo visível. Então, é possível controlar a visibilidade dinamicamente no lado do cliente sem ser preciso enviar uma requisição ao servidor e ter que renderizar novamente a página, poupando tempo e consumo de banda.

Veja na **Listagem 8** como é simples ocultar e exibir elementos utilizando a biblioteca jQuery e tenha uma prévia de como faremos isso no nosso projeto com a **Figura 16**.

Na listagem, podemos perceber dentre outras coisas:

- **Linha 31:** Aqui foi inserido um parágrafo exibindo uma mensagem de confirmação de cadastro, este elemento é o que iremos controlar para exibição de uma forma geral.
- **Linhas 32 e 33:** Foram definidos dois botões, um para exibir a mensagem da linha 31 e outro para ocultar a mesma.
- **Linhas 17 à 19:** É utilizada a função `hide()` para ocultar o botão `btn_ocultar` e o parágrafo “mensagem”, logo após é exibido o botão `btn_exibir` através do `show()`. Por fim, nas linhas 23 à 25 é realizado o mesmo processo, porém de forma invertida, ou seja, exibir o botão `btn_exibir` e a mensagem novamente.

HTML dinâmico com jQuery e Bootstrap

O jQuery possibilita gerar elementos HTML dinamicamente e inseri-los em uma página. Para exemplificar, vejamos como realizar este processo fazendo uso também do Bootstrap para criar um pequeno cadastro de clientes que será apresentado apenas quando o usuário clicar no botão cadastrar. Veja na **Listagem 9** todo o código necessário para criar este formulário e os comentários sobre o mesmo em seguida.

Veja a seguir alguns detalhes importantes a serem citados em relação à listagem:

- **Linha 7 a 10:** Temos as referências aos arquivos CSS e JS do Bootstrap e da biblioteca jQuery (a ordem não altera os resultados).
- **Linhas 13 à 30:** É definido o JavaScript que faz uso do jQuery responsável por gerar dinamicamente todo o código HTML.

Listagem 8. Aplicando funções `hide()` e `show()` com jQuery.

```

01 <!DOCTYPE html>
02 <html lang="pt-BR">
03   <head>
04     <meta charset="utf-8">
05     <meta name="viewport" content="width=device-width, initial-scale=1">
06     <title>Exemplo visibilidade com jQuery</title>
07     <script type="text/javascript" src="jquery/jquery-2.1.3.min.js"></script>
08     <style rel="stylesheet" type="text/css">
09       .estilo1{
10         background-color: lightgreen;
11         border: 1px solid;
12       }
13     </style>
14     <script type="text/javascript">
15       $(document).ready(function(){
16         $("#btn_ocultar").click(function(){
17           $("#mensagem").hide();
18           $("#btn_ocultar").hide();
19           $("#btn_exibir").show();
20         });
21
22         $("#btn_exibir").click(function(){
23           $("#mensagem").show();
24           $("#btn_ocultar").show();
25           $("#btn_exibir").hide();
26         });
27       });
28     </script>
29   </head>
30   <body>
31     <p id="mensagem">Cadastro realizado com sucesso.</p>
32     <button id="btn_ocultar">Ocultar mensagem</button>
33     <button id="btn_exibir">Exibir mensagem</button>
34   </body>
35 </html>
```

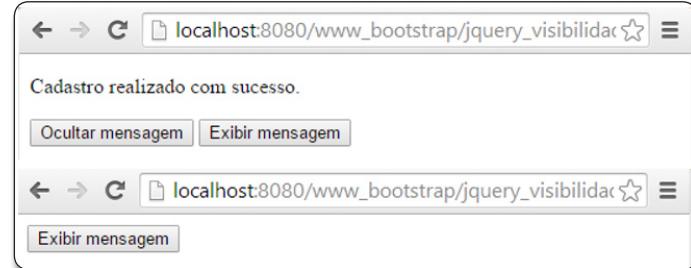


Figura 16. Resultado da execução das funções `show()` e `hide()`

- **Linha 33:** Foi criado um botão que será utilizado para disparar o evento de click para gerar o formulário de cadastro de cliente dinamicamente na página.
- **Linhas 34 e 35:** Nestas linhas temos as tags de abertura e fechamento do formulário HTML que receberá os elementos para o usuário preencher.
- **Linha 14:** Definição da função `ready` que aguarda a renderização por completo da página e aguarda o clique do usuário no botão cadastrar.
- **Linha 15:** Seleção do botão `btn_formulario` para recuperar o evento click e em seguida gerar o formulário.

Criando um Portal Web com Bootstrap e jQuery - Parte1

Listagem 9. Gerando código HTML dinamicamente com jQuery.

```
01 <!DOCTYPE html>
02 <html lang="pt-BR">
03 <head>
04   <meta charset="utf-8">
05   <meta name="viewport" content="width=device-width, initial-scale=1">
06   <title>Gerando elementos html com jQuery</title>
07   <link rel="stylesheet" href="Bootstrap-3.3.2-dist/css/Bootstrap.min.css" >
08   <link rel="stylesheet" href="Bootstrap-3.3.2-dist/css/
      Bootstrap-theme.min.css">
09   <script type="text/javascript" src="jquery/jquery-2.1.3.min.js"></script>
10   <script type="text/javascript" src="Bootstrap-3.3.2-dist/js/
      Bootstrap.min.js"></script>
11
12
13   <script type="text/javascript">
14     $(document).ready(function() {
15       $("#btn_formulaio").click(function(){
16         $("form").append("<div class='container'>" + "<div class='jumbotron'>" +
17           "<div id='div_nome' class='form-group'></div>" + "</div>" +
18           "</div>");
19
20         $("#div_nome").append("<br><br>");
21         $("#div_nome").append("<label>Nome</label>");
22         var inputNome = $("<input />", {id:"inputNome", class:"form-control"});
23         $("#div_nome").append(inputNome);
24
25         $("#div_nome").append("<label>CPF</label>");
26         var inputCPF = $("<input />", {id:"inputCPF", class:"form-control"});
27         $("#div_nome").append(inputCPF);
28
29         $("#div_nome").append("<br><br><button type='submit'
            class='btn btn-default'>Cadastrar</button>");
30       });
31     });
32   </body>
33   <button id="btn_formulaio" class="btn btn-default">Cadastro cliente
34   </button><br><br>
35 </form>
36 </body>
37 </html>
```

FÓRUM DEVMEDIA

O lugar perfeito para você ficar por dentro
de tudo o que acontece nas tecnologias do
mercado atual

No fórum da DevMedia você irá encontrar uma equipe disponível
e altamente qualificada com consultores e colaboradores prontos
para te ajudar a qualquer hora e sobre qualquer assunto.
Temos as salas de Java, .NET, Delphi, Banco de Dados,
Engenharia de Software, PHP, Java Script, Web Design,
Automação comercial, Ruby on Rails e muito mais!



ACESSE AGORA
www.devmedia.com.br/forum

- **Linhas 16 e 17:** Aqui, a geração dos elementos HTML é iniciada e são chamadas as classes CSS para gerar o formulário com aparência de alguns componentes originais do Bootstrap. Perceba que foi selecionado o form que está dentro do body através do \$("form"), em seguida é chamada a função append que recebe como parâmetro um código HTML a ser inserido conforme o elemento selecionado, neste caso o form. Também é interessante observar que foram inseridas três divs usando as seguintes classes do Bootstrap: container, jumbotron e form-group, que serão responsáveis pelo alinhamento.

- **Linhas 19 e 20:** Aqui também é gerado mais código HTML, neste caso os elementos propriamente ditos do formulário, como a label de título, etc. Veja que foi utilizada a função append e selecionada a div com o id 'div_nome' gerada anteriormente, dessa forma podemos notar que embora o JavaScript não tenha sido executado por completo já é possível usar os elementos gerados na instrução anterior através desta função.

- **Linha 21:** Aqui é gerado um elemento do tipo input, porém foi criada primeiro uma variável **inputNome** e em seguida gerado o elemento com dois parâmetros: o tipo do elemento e o <input />. Em seguida, finalizamos com as configurações finais do Bootstrap.

- **Linha 22:** Mesmo processo das linhas 24 à 26, para gerar o elemento de CPF.

- **Linha 28:** Por fim, é criado o botão de cadastro de fato com as classes btn btn-default de formatação do botão de input do Bootstrap.

Veja na **Figura 17** o resultado da execução do código, observando os efeitos de usar ambos os frameworks juntos. Pressione a tecla F12 no seu navegador e observe se algum log de erro JavaScript foi impresso no console de depuração do mesmo.

Esta primeira parte do artigo teve como objetivo realizar uma breve introdução ao Bootstrap e à biblioteca jQuery, mas ainda tem muito a ser explorado sobre os mesmos. Além disso, é de suma importante que você tenha uma boa noção de tecnologias como HTML5, CSS3 e JavaScript para melhor compreensão do artigo e, dessa forma, possa ganhar experiência e produtividade em projetos que focam nas mesmas.

Figura 17. Tela de cadastro dinâmica com jQuery e Bootstrap

Links:

Website Bootstrap

<http://getbootstrap.com/>

Download Bootstrap

<http://getbootstrap.com/getting-started/#download>

Compilador Grunt usado no Bootstrap

<http://gruntjs.com/>

Tradução Bootstrap realizada pela Globo.com

<http://globocom.github.io/Bootstrap/index.html>

Download WampServer

<http://www.wampserver.com/en/>

Download biblioteca jQuery

<http://jquery.com/>

Autor



Madson Aguiar Rodrigues



Formação acadêmica em Análise e Desenvolvimento de Sistemas pela UNOPAR, pós-graduação em Engenharia de Sistemas pela ESAB e especialista em Tecnologias para aplicações Web pela UNOPAR. Trabalha com desenvolvimento de software há sete anos com uso a da plataforma .NET, JAVA e Mobile com Android. Atua no mercado com prestação de serviços e consultoria em TI, fornecendo soluções em software, tutor do curso de graduação em Análise e Desenvolvimento de Sistemas na UNOPAR e autor artigos e vídeo no portal DEVMEDIA.

Construindo aplicações web com Yeoman e Polymer

Aplicações responsivas e customizáveis usando HTML5 e frameworks JavaScript

Construir aplicações web modernas requer muita configuração e um uso demasiadamente excessivo de ferramentas e frameworks. Isso inclui pré-processadores, frameworks JavaScript das mais diversas formas e finalidades, ferramentas de teste e muito mais. E, à medida que a complexidade desses aplicativos cresce, também é também a amplitude de ferramentas e serviços necessários para gerenciá-los.

Além disso, quando falamos sobre manutenção de aplicações desse estilo, temos inúmeros profissionais envolvidos, processos mal estabelecidos e gerenciados, sem falar no código que vai se tornando cada vez maior, cheio de integrações, dependências e acoplamentos. Para ter uma ideia de como tudo isso funciona na prática, analisemos a **Figura 1** que traz basicamente a representação final (HTML) do conteúdo de páginas em aplicações famosas como Google Gmail, Facebook e Amazon.

Perceba que logo de cara temos bastante conteúdo, e todo ele particionado em div's e span's, o que nos leva a questionar se essa estrutura característica realmente se faz tão necessária. Isso provavelmente também o levará a pensar como será toda a manutenção desse tipo de código dentro do projeto nessas empresas, e se as mesmas mantêm alguma estrutura de automatização ou usam frameworks para facilitar esse gerenciamento.

Muito tem sido discutido acerca dos valores semânticos que as tags da HTML5 trouxeram às nossas aplicações contemporâneas (e que também visavam a diminuição significativa de código HTML nas páginas), porém elas ainda estão longe de proporcionar uma “margem de lucro programática” significativa, principalmente por duas razões, a saber:

Fique por dentro

Este artigo fará uma abordagem bem completa acerca de dois dos frameworks front-end mais usados pela comunidade web: o Polymer.js e o Yeoman, através da criação de um exemplo prático de blog com posts, páginas e avatar, totalmente responsivo. Exploraremos recursos como automação de tarefas usando o Grunt.js, gerenciamento automático de pacotes com o projeto Yo e atualização de bibliotecas via Bower, além de entender como configurar cada uma das tecnologias usadas e suas dependências.

Na construção do aplicativo de teste, faremos uso da Google Spreadsheets API para fornecer uma fonte de dados real, assim como usaremos elementos da customelements.io (biblioteca de projetos prontos do Polymer) para fornecer funcionalidades mais completas ao projeto.

- Quando temos dois componentes muito similares em nossa página web que caem dentro da mesma estrutura semântica, para distingui-los um do outro, nós usamos classes, ids, ou outros atributos;
- A lista disponível de tags semânticas é simplesmente insuficiente para atingir a grande variedade de componentes que constituem nosso design. Como resultado, nós voltamos novamente para as tags tradicionais como div, span, etc.

A W3C pretende ir de encontro a esse problema ao introduzir o conceito de componentização web. Os componentes web são nada mais que uma coleção de especificações que possibilitam que desenvolvedores criem suas próprias aplicações web como um conjunto de componentes reusáveis. Através disso, eles proveem uma mudança de paradigma em relação à abordagem tradicional

The image shows three separate browser windows side-by-side. Each window has a search bar at the top and displays the raw HTML code for a different website. The left window is labeled 'GMAIL' and shows the source code for the Gmail inbox. The middle window is labeled 'FACEBOOK' and shows the source code for the Facebook homepage. The right window is labeled 'AMAZON' and shows the source code for the Amazon homepage. The code is presented in a monospaced font, showing various HTML tags, CSS styles, and JavaScript snippets.

Figura 1. Conteúdo HTML das páginas do Gmail, Facebook e Amazon, respectivamente

ao mudar fundamentalmente a forma como construímos e concebemos as aplicações web.

Cada componente vive em sua autodefinida unidade de encapsulamento com estilo e lógica correspondentes. Estes mesmos componentes não só podem ser compartilhados via aplicação web comum, como também distribuídos na web para uso de outrem. Basicamente, eles são constituídos de quatro especificações:

- **Elementos customizados:** Permitem que desenvolvedores criem seus próprios elementos (que são relevantes para o design) como parte da estrutura DOM com a habilidade de estilizar/importar scripts a eles como qualquer outra tag HTML.
- **Templates HTML:** Define fragmentos de marcação que permanecem consistentes sobre as páginas web com a habilidade de injetar conteúdo dinâmico usando somente JavaScript.
- **Shadow DOM:** Designado para abstrair todas as complexidades da marcação de tags através da definição de limites funcionais entre a árvore DOM e as sub-árvores escondidas atrás de uma shadow raiz.
- **Importação de HTML:** Similar à importação de arquivos CSS dentro deles mesmos, permite que você inclua e reuse documentos HTML em outros documentos HTML.

Resumindo, os componentes web conseguem resolver todos os principais problemas de complexidade na forma como criamos novos elementos que incutem funcionalidades ricas sem a necessidade de bibliotecas externas.

Existem vários frameworks e empresas que já lidam com esse conceito na prática. O **Polymer.js** foi criado pelo Google e, por intermédio dele, nós podemos usar todos estes recursos de forma integrada em um só lugar, com excelente suporte da

empresa e comunidade, e totalmente adaptado aos browsers modernos.

Um outro problema recorrente nesse universo de desenvolvimento front-end se refere à forma como integramos as coisas, isto é, como podemos gerenciar todas essas ferramentas, componentes web, frameworks, HTML, etc. em um só lugar, de forma fácil, rápida e segura? A resposta é **Yeoman**. O Yeoman funciona como uma coleção composta por três ferramentas principais: Yo, Grunt e Bower. Combinados, estes três projetos fornecem tudo que você pode precisar em um projeto desse tipo.

Neste artigo faremos uso dos referidos frameworks para entender como criar aplicações web de forma mais fácil e produtiva. Construiremos uma aplicação de blog na web para que o leitor entenda como as ferramentas atuam em conjunto, bem como funcionam os processos de configuração, instalação e adaptação das mesmas aos diferentes ambientes de software.

0 que é o Yeoman?

O Yeoman por si só, nada mais é do que um empacotamento de três projetos distintos, cada um com um objetivo pré-definido:

- **Yo:** É uma ferramenta que é usada para automatizar o seu build e gerenciar dependências no seu projeto front-end. Ela pode criar ativos de projeto comuns, como folhas de estilo e arquivos JavaScript. Basicamente, o Yo fornece uma maneira de pré-criar todo o código clichê que você precisa para começar um projeto.

- **Grunt:** O Grunt é um executor de tarefas que pode automatizar coisas como a compilação dos arquivos Sass ou a otimização de imagens. Quando usado como parte do pacote do Yeoman, o Grunt lida com a construção, testes e pré-visualização do seu projeto.

Construindo aplicações web com Yeoman e Polymer

- **Bower:** É um gerenciador de pacotes para a web. Ele ajuda a gerenciar as dependências necessárias para a sua aplicação, nos livrando da responsabilidade de ter que baixá-las manualmente.

Cada uma dessas ferramentas é desenvolvida de forma independente, mas todas têm grandes comunidades empurrando seu avanço contínuo.

O nome Yeoman é usado em alusão ao termo inglês que se refere a um fazendeiro que cultivava sua própria terra e tinha pequenas propriedades na antiga Inglaterra, assim como alguns direitos políticos. Por isso a mascote do framework é usada em referência ao mesmo personagem inglês característico (Figura 2).

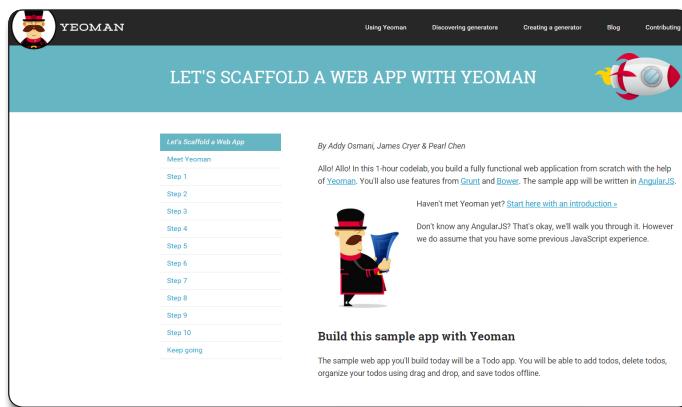


Figura 2. Página oficial do Yeoman, em alusão ao fazendeiro proprietário da história inglesa

Grunt e Bower

O Grunt.js é um executor de tarefas que pode aumentar a produtividade no fluxo de desenvolvimento front-end substancialmente. Baseado em linha de comando, ele pode lidar com tarefas dos mais diversos tipos, tais como execução de testes unitários, ferramentas de detecção de erros no código, otimização de imagens, compressão de arquivos JavaScript e Sass, atualização automática de binários quando da mudança em seus respectivos fontes, etc.

Com isso, o foco do desenvolvedor sai da área de configuração e instalação e se volta para o código em si. Além disso, ele também conta com uma série de plugins que podem ser estendidos rapidamente via linha de comando, adicionando novos recursos ao seu ambiente.

Já o Bower é um gerenciador de pacotes que organiza todas as dependências do projeto. Ele funciona de forma semelhante a ferramentas como o Maven no Java, ou o Nuget para o .NET, por exemplo, porém não usamos arquivos XML para efetuar as configurações e sim uma interface totalmente baseado na linha de comando.

O que é o Polymer?

Em poucas palavras, o Polymer é um framework que define, cria e renderiza elementos customizados complexos de uma forma simplista e elegante em relação à forma como fazímos usando a antiga HTML. Isso é possível através do forte encapsulamento

da maior parte do código e estrutura do projeto, permitindo que desenvolvedores usem uma convenção simples de estilos de tags; assim como da suíte de elementos UI pré-definidos para extensão.

Em outras palavras, o que esta biblioteca pode fazer é nos permitir criar componentes reutilizáveis que funcionam como verdadeiros elementos DOM, ajudando a minimizar a nossa dependência de JavaScript para fazer manipulações mais complexas e rendendo mais resultados na UI como um todo.

O código a seguir ilustra rapidamente como é fácil usar o Polymer. Digamos que queiramos implementar um relógio funcionando na nossa página, como o que temos na barra de programas do Windows. Isso normalmente implicaria em pesados códigos JavaScript, mas usando o Polymer podemos simplesmente fazer o seguinte:

```
<polymer-ui-clock></polymer-ui-clock>
```

Isto se parece exatamente com a sintaxe de tags HTML que todos conhecemos e é muito mais fácil de implementar, ler e manter do que um código JavaScript complexo. O resultado final se parece com o ilustrado na Figura 3.

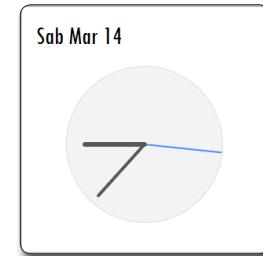


Figura 3. Exemplo de uso do Polymer para criar relógio simples

Como o componente gerado nada mais é do que um elemento comum do DOM em forma de HTML, podemos estilizá-lo normalmente como faríamos com as divs, títulos, etc. Veja na Listagem 1 a representação CSS do exemplo em questão.

Listagem 1. Código CSS do relógio exemplo no Polymer.

```
01 polymer-ui-clock {  
02   width: 320px;  
03   height: 320px;  
04   display: inline-block;  
05   background: url("../img/relogio.png") no-repeat;  
06   background-size: cover;  
07   border: 4px solid rgba(32, 32, 32, 0.3);  
08 }
```

Perceba que a utilização do componente não traz um estilo padrão em relação ao plano de fundo do relógio e isso pode ser facilmente manipulado pelo desenvolvedor, tornando o processo todo bem dinâmico. Todavia, o ponto central da implementação é provar que, longe das regras de CSS básicas ou definições de

estilo de quaisquer tipos, usar o Polymer é fácil e que o mesmo é poderoso na geração de componentes.

No site oficial do projeto (seção **Links**) você encontrará as opções de download, assim como tutoriais e a própria documentação do framework, tal como na **Figura 4**. Além disso, demos, blog e outros tópicos complementares à tecnologia são sempre atualizados pela comunidade e pelo Google na mesma página. O projeto se encontra atualmente na versão 0.5 e é a que utilizaremos no artigo.

Arquitetura do Polymer

O Polymer é dividido em quatro camadas, a saber:

- **Native**: Funcionalidades necessárias que estão disponíveis na maioria dos browsers atualmente.

- **Foundation**: Elementos que implementam as funcionalidades necessárias dos browsers e que ainda não estão nativamente disponíveis nos mesmos. A ideia é que essa camada desapareça com o tempo, à medida que os browsers forem adicionando suporte nativo aos referidos recursos.

- **Core**: Representa a infraestrutura necessária para os elementos do Polymer poderem adicionar as capacidades providas pelas camadas anteriores.

- **Elements**: Um conjunto básico de elementos que servem como blocos de construção e que podem te ajudar a criar uma aplicação inteira. Incluem elementos que proveem funcionalidades básicas como Ajax, animações, layouts Flex e gestos. Também encapsula APIs e layouts CSS complexos dos browsers, bem como contém renderizadores de componentes UI, tais como acordeões, sidebars, etc.

Veja na **Figura 5** uma ilustração resumida da real arquitetura dessas camadas.

Polymer x Angular.js

O Angular é um framework completo para construção de aplicações web, já o Polymer é uma biblioteca para criar componentes web. Os conceitos diferem, mas apesar disso ambos podem ser usados para construir uma webapp.

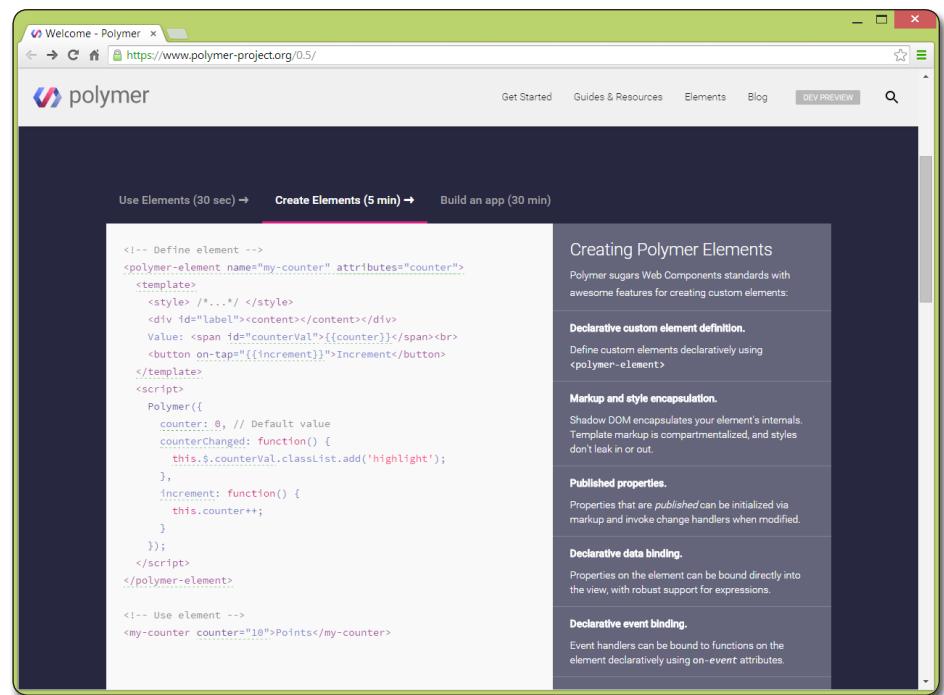


Figura 4. Site oficial do Polymer na parte de tutoriais rápidos

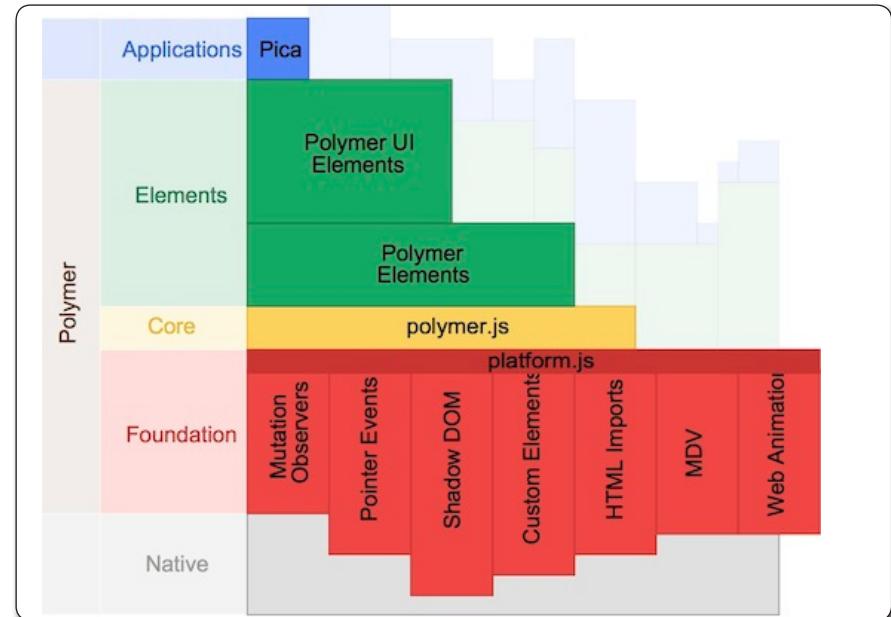


Figura 5. Representação arquitetural das camadas do Polymer

O Angular tem APIs de alto nível para coisas como serviços, roteamento, comunicação entre servidores, etc. O Polymer, por outro lado, não provê esse tipo de recurso exceto o que separa componentes web de suas bibliotecas núcleo. Em vez disso, ele foca em permitir a criação de conteúdo rico e reusável que pode ser usado para

construir aplicações que façam uso do Angular ao mesmo tempo.

É provável que o Google invista em projetos de integração entre estes frameworks visto que ambos pertencem à empresa. Apesar de tudo, existem similaridades entre os dois, como as diretivas dos elementos que são muito semelhantes aos

componentes web, resultando em uma comparação que seria feita basicamente entre os elementos customizados do Polymer e as diretivas do Angular. Isto é, enquanto o Polymer provê a habilidade de compor JavaScript, CSS e HTML encapsulados como elementos customizados, o Angular tem uma implementação extremamente semelhante quando se trata de suas diretivas de elementos. A maior diferença está no fato do Angular não precisar fazer uso das APIs de componentes web que o Polymer faz. Ao mesmo tempo, podemos dizer que as diretivas do Angular são “um caminho” para construir elementos customizados, enquanto o Polymer e a especificação dos componentes web são o “caminho padronizado” para fazer isso.

Da mesma forma que o Angular, os elementos do Polymer proveem a funcionalidade de templates e associação de dados bidirecionais. Outra diferença está no fato de que o Angular não reconhece quaisquer formas de encapsulamento de CSS, algo que planeja-se embutir na tecnologia em breve.

Configurando o ambiente

Antes de seguir para os frameworks citados no artigo, devemos antes instalar o Node.js, visto que todos eles têm dependência direta no **npm** (gerenciador de downloads e pacotes do Node.js). Para isso, baixe a última versão do Node.js (vide seção [Links](#)) e instale-o sem alterar nenhuma das opções que já vêm marcadas por padrão.

Após a instalação, para verificar se tudo ocorreu direito, basta abrir o prompt de comandos do Windows (ou do respectivo Sistema Operacional) e executar o seguinte comando:

```
node -v
```

O resultado deverá ser v0.12.0 (última versão na época da escrita deste artigo). Caso apareça algo como “node não é reconhecido como um comando interno” basta refazer os passos, desinstalando o Node.js e instalando-o novamente.

Uma vez com o Node.js configurado, agora podemos executar o comando para instalar o Yeoman. Para isso, crie uma pasta em um diretório de preferência para o projeto, tal como “proj-yeoman-polymer” e navegue até o mesmo via cmd. Após isso, execute o seguinte comando:

```
npm install -g yo
```

Aguarde um pouco até o processo finalizar, e você verá algo parecido com o conteúdo da **Listagem 2**.

Perceba que o npm irá alocar os recursos necessários para se trabalhar com o Yeoman (e seus projetos dependentes) na pasta **AppData** do diretório do seu usuário do sistema. Isso significa que essa é uma configuração feita por usuário, logo se precisar acessar os recursos do Yeoman com outros usuários terá de refazer os mesmos passos. Na linha 2 temos uma mensagem de alerta que informa sobre a versão que estamos usando do Node.js, npm e a versão do Yo que será instalada. Não se preocupe, isso é apenas

para alertar que é sempre recomendável usar o que há de mais recente, uma vez que o npm busca os pacotes de instalação e faz o download no repositório via conexão web.

Listagem 2. Resultado da instalação do Yeoman.js.

```
01 D:\tests\yeoman-polymer>npm install -g yo
02 npm WARN engine yo@1.4.6: wanted: {"node":">=0.10.0","npm":">=2.1.0"}
  (current:
03 {"node":"0.10.35","npm":"1.4.28"})
04 C:\Users\Julio\AppData\Roaming\npm\yo -> C:\Users\Julio\AppData\Roam
05 ing\npm\node_modules\yo\lib\cli.js
06
07 > yo@1.4.6 postinstall C:\Users\Julio\AppData\Roaming\npm\node_modules\yo
08 > yodoctor
09
10
11 Yeoman Doctor
12 Running sanity checks on your system
13
14 V Global configuration file is valid
15 V NODE_PATH matches the npm root
16 V No .bowerrc file in home directory
17 V No .yo-rc.json file in home directory
18
19 Everything looks all right!
```

Esse processo já irá efetuar o download do Bower e Grunt devidamente, adicionar as referências nas variáveis de ambiente, uma vez que precisamos disso para que os frameworks sejam reconhecidos a nível de ambiente Windows e possam ter seus comandos executados via cmd. Tudo estará ok quando a mensagem da linha 19 aparecer na tela. Após essa linha, várias outras mensagens aparecerão em detrimento da configuração das dependências do projeto. Para conferir se deu tudo certo, execute os seguintes comandos no prompt:

```
bower -version
grunt -version
```

Os comandos deverão imprimir, respectivamente, as versões do Bower e Grunt instalados no processo todo. Com isso, nosso Yeoman estará instalado e pronto para funcionar.

ENOGIT git is not installed or not in the PATH

Esse tipo de erro é comum quando se está tentando fazer a instalação do Yeoman especialmente pelo uso do Bower. O que acontece é que o Bower tem dependências diretas no Git, e de duas uma: ou você não tem o Git instalado na sua máquina, ou ele foi instalado sem ser adicionado à variável de ambiente PATH.

Se estivermos lidando com a primeira situação, basta efetuar o download do Git no site oficial (seção [Links](#)) e instalá-lo. Quando executar o instalador, siga o passo a passo sem mudar nenhuma opção padrão, exceto pela representada na **Figura 6**, onde podemos ver a wizard que pergunta se desejamos configurar o mesmo na referida variável. A segunda opção irá adicionar todas as referências necessárias e deixar o Bower funcional.

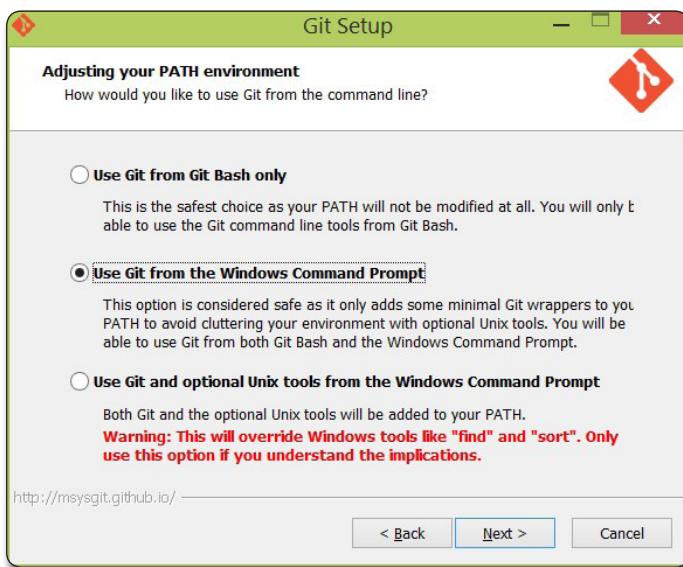


Figura 6. Seleção da opção para adicionar o Git na variável de ambiente PATH

Generators

O Yeoman trabalha ainda com o conceito de **generators**, que são componentes que definem a estrutura da aplicação bem como quaisquer dependências que a mesma venha a precisar. Os generators estão disponíveis para uma grande quantidade de aplicações, desde uma aplicação web padrão até uma aplicação que faça uso do Angular.js ou do Backbone.js, por exemplo. Veja na **Tabela 1** uma lista dos principais generators da comunidade e seus respectivos ids no Github.

Nome	Generator	Github Id
Web App	generator-webapp	github.com/yeoman/generator-webapp
Backbone	generator-backbone	github.com/yeoman/generator-backbone
Ember.js	generator-ember	github.com/yeoman/generator-ember
Angular.js	generator-angular	github.com/yeoman/generator-angular
Polymer	generator-polymer	github.com/yeoman/generator-polymer
jQuery	generator-jquery	github.com/yeoman/generator-jquery

Tabela 1. Lista de generators do Yeoman mais famosos da comunidade

Você pode encontrar uma lista mais completa dos generators disponíveis na página do projeto Yeoman no Github (seção **Links**). Os generators também são instalados via npm. Para instalar o generator do Polymer, basta executar o seguinte comando:

```
npm install generator-polymer -g
```

Aguarde até que o processo termine. Isso irá permitir que possamos customizar elementos do Polymer via linha de comando e importá-los usando imports HTML comuns dentro das tags link do cabeçalho da página, o que nos economiza tempo por não termos de criar código clichê de inicialização de páginas web.

Esse generator tem algumas propriedades importantes que serão usadas no projeto, a saber:

- **polymer:element** é usado para extrair novos elementos individuais do Polymer. Por exemplo:

```
yo polymer:element carousel
```

- **polymer:app** é usado para extrair o conteúdo inicial da index.html, um arquivo Gruntfile.js contendo configurações de tempo de execução para o projeto, assim como tarefas do Grunt e uma estrutura de pastas recomendada para o projeto. Isso irá te dar também a opção de usar o Sass Bootstrap nos estilos do seu projeto.

Criando uma aplicação Polymer

Como caso de uso para entendimento do Polymer em conjunto com o Yeoman, vamos construir uma página de blog simples, fazendo uso extensivo dos elementos de ambos os frameworks assim como do nosso generator.

Para iniciar, acesse a pasta do projeto via terminal de linha de comando e execute o seguinte comando:

```
yo polymer
```

O Yo irá te perguntar se deseja fazer uso do Boostrap como opção ao estilo e componentização do projeto, para tanto basta digitar Y e dar um enter. O resultado será semelhante ao da **Figura 7**.

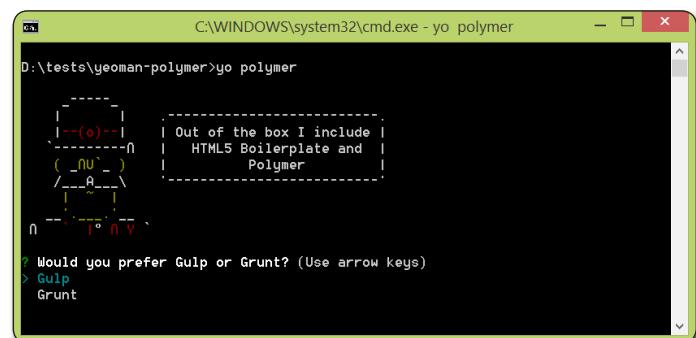


Figura 7. Resultado de execução do comando para criar projeto no Yo

Perceba como a ferramenta lida de forma extrovertida com as configurações feitas. Note que nessa tela estamos sendo perguntados se desejamos usar o Grunt ou o Gulp como ferramenta de automação de tarefas. Conforme decidimos antes, vamos optar pelo Grunt, então selecione a opção e dê um enter.

Após isso, algumas outras questões vão sendo feitas pelo framework para adicionar ou remover recursos do projeto. Veja na **Figura 8** quais perguntas e respostas selecionaremos para o nosso projeto.

Aguarde até que o processo termine, o que pode tomar um bom tempo. Na tela, o Yo informa quais arquivos, módulos e dependências serão instalados. Ao mesmo tempo, veja que no diretório que escolhemos para o projeto, vários arquivos vão sendo criados (**Figura 9**).

Note que o mesmo automaticamente já adiciona suporte ao Git, onde você pode facilmente adicionar ou clonar o projeto para

Construindo aplicações web com Yeoman e Polymer

um repositório público como o Github, por exemplo. Além disso, temos todas as dependências de arquivos do Bower, JShint (para análise de erros de código), Yo e do Grunt (Gruntfile.js).



```
D:\tests\yeoman-polymer>yo polymer
-----
[ -o- ] Out of the box I include
        HTML5 Boilerplate and
        Polymer
[ -A- ]
[ ~ ]
[ -o- ] Out of the box I include
        HTML5 Boilerplate and
        Polymer

? Would you prefer Gulp or Grunt? Grunt
? Would you like to include core-elements? Yes
? Would you like to include paper-elements? Yes
? Would you like to use SASS/SCSS for element styles? Yes
? Would you like to include web-component-tester? No
  create bower.json
  create package.json
  create .gitignore
  create .gitattributes
  create .bowerrc
  create wct.conf.js
  create .jshintrc
  create .editorconfig
  create Gruntfile.js
  create app\404.html
  create app\favicon.ico
  create app\robots.txt
  create app\styles\main.scss
  create app\scripts\app.js
  create app\.htaccess
  create app\elements\elements.html
  create app\elements\yo-list\yo-list.html
  create app\elements\yo-list\yo-list.scss
  create app\elements\yo-greeting\yo-greeting.html
  create app\elements\yo-greeting\yo-greeting.scss
  create app\index.html

I'm all done. Running npm install & bower install for you to install the required dependencies. If this fails, try running the command yourself.

npm WARN package.json yeoman-polymer@0.0.0 No description
npm WARN package.json yeoman-polymer@0.0.0 No repository field.
npm WARN package.json yeoman-polymer@0.0.0 No README data
npm WARN engine imagemin@3.1.0: wanted: {"node":">=0.10.0", "npm":">=2.1.5"} (cur-
```

Figura 8. Próximas perguntas do Yo para geração de arquivos do projeto

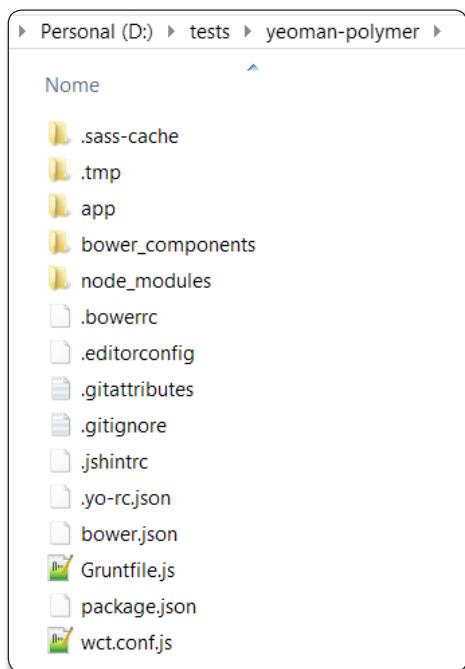


Figura 9. Estrutura de diretórios criada para o projeto pelo Yo

A pasta **app** é a principal do projeto guardando os elementos visuais como imagens, scripts, arquivos de CSS, HTML, etc., sendo nela onde será criado o arquivo index.html. Já a pasta **node_modules** configura os módulos dependentes que definimos respondendo a todas aquelas questões, e a pasta **bower_components** irá se encarregar de gerenciar todas as dependências núcleo do projeto, tais como componentes do Bootstrap, Sass, Less, etc.

Após todo esse processo, teremos a última versão do Polymer configurado, assim como o arquivo index.html preenchido com o conteúdo apresentado na **Listagem 3**.

Logo nas duas primeiras linhas podemos reparar que estamos lidando com a geração automática de arquivos HTML5, pelas tags simplificadas geradas. Note como não temos de nos preocupar com processos simples como importar arquivos de scripts/estilo, ou definir o conteúdo da tag head. Até mesmo a inclusão do favicon é feita de forma automática, basta que coloquemos a imagem desejada no diretório raiz das imagens (linha 9).

Das linhas 11 à 21 temos a importação dos arquivos CSS, JavaScript e HTML sendo feitas, basta que alteremos agora as estruturas dos arquivos de origem. Com fortes definições de templates, podemos ver na linha 25 o uso direto da tag também autogerada, com os elementos **core-drawer-panel** (linha 27) criando o cabeçalho da página com a inclusão de um menu e submenus e **core-header-panel** (linha 45) para conter o conteúdo dessa página em específico. Finalmente, na linha 64 vemos a importação de mais um arquivo de scripts, dessa vez para a aplicação.

Veja como o framework abstrai inclusive os nomes de atributos HTML, reduzindo a quantidade deles como o **type** das tags de importação de scripts e arquivos de estilo. Para ver o resultado, você pode simplesmente executar o arquivo no navegador, mas vamos aprender um jeito mais interessante de fazer isso.

O Grunt traz consigo uma implementação de servidor (atrelado ao Node.js) que serve como um servidor web, porém para o lado cliente. Ele executa o arquivo index.html como página home da aplicação e fica ouvindo as alterações que fazemos nos arquivos diretamente ou via linha de comando, atualizando a página no browser sem que sequer precisemos dar um refresh na mesma, tudo isso graças à task “watch” que vem configurada por padrão no arquivo Gruntfile.js. Então, execute o seguinte comando no prompt cmd:

```
grunt serve
```

Isso irá fazer com que o grunt chame as tasks para gerar os executáveis de cada arquivo e minificar os mesmos. Aguarde o processo acabar e ele irá automaticamente abrir a página do seu navegador padrão, no endereço *localhost:9000*, tal como demonstrado na **Figura 10**. Lembre-se que este comando deve ser executado sempre dentro do diretório raiz do projeto criado.

No exemplo em questão, temos uma caixa de texto simples que irá atualizar o texto do título da página automaticamente após mudarmos seu valor. Esse efeito é feito via bibliotecas de Ajax importadas.

Listagem 3. Conteúdo inicial da página index.html.

```

01 <!doctype html>
02 <html lang="">
03
04 <head>
05   <meta charset="utf-8">
06   <meta name="description" content="">
07   <meta name="viewport" content="width=device-width, initial-scale=1">
08   <title>poly foo</title>
09   <!-- Place favicon.ico and apple-touch-icon.png in the root directory -->
10
11  <!-- build:css styles/main.css -->
12  <link rel="stylesheet" href="styles/main.css">
13  <!-- endbuild-->
14
15  <!-- build:js bower_components/webcomponentsjs/webcomponents.min.js -->
16  <script src="bower_components/webcomponentsjs/webcomponents.js">
17    </script>
18  <!-- endbuild -->
19
20  <!-- will be replaced with elements/elements.vulcanized.html -->
21  <link rel="import" href="elements/elements.html">
22  <!-- endreplace-->
23
24 <body unresolved fullbleed layout vertical>
25  <template is="auto-binding" id="app">
26
27    <core-drawer-panel>
28
29      <!-- Drawer -->
30      <core-header-panel drawer>
31
32        <!-- Drawer Toolbar -->
33        <core-toolbar>Menu</core-toolbar>
34
35        <!-- Drawer Content -->
36        <core-menu selected="0">
37          <core-item label="Yo"></core-item>
38          <core-item label="Polymer"></core-item>
39          <core-item label="App"></core-item>
40        </core-menu>
41
42      </core-header-panel>
43
44      <!-- Main -->
45      <core-header-panel main>
46
47        <!-- Main Toolbar -->
48        <core-toolbar>
49          <paper-icon-button icon="menu" core-drawer-toggle></paper-icon-button>
50          <span>{{appName}}</span>
51        </core-toolbar>
52
53        <!-- Main Content -->
54        <div class="hero-unit">
55          <yo-greeting></yo-greeting>
56          <p>You now have</p>
57          <yo-list></yo-list>
58        </div>
59      </core-header-panel>
60    </core-drawer-panel>
61
62  </template>
63
64  <!-- build:js scripts/app.js -->
65  <script src="scripts/app.js"></script>
66  <!-- endbuild-->
67 </body>
68
69 </html>
```

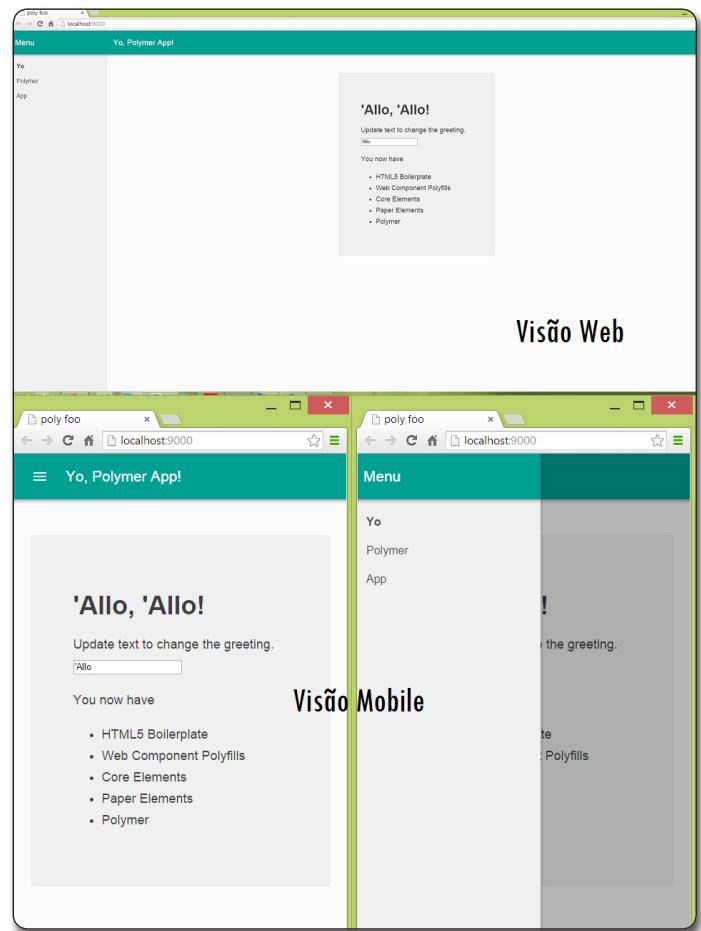


Figura 10. Página index.html executada através do comando grunt serve

Veja como a página assume características responsivas em função da mudança de tamanhos de telas, logo não precisamos perder tempo nos preocupando com essa implementação no projeto que precisa de responsividade.

O servidor, por sua vez, possui suporte a *LiveReload*, isto é, você pode abrir um editor de texto, editar um elemento customizável e o browser irá recarregar sozinho. Isso também economiza tempo na hora de efetuar os testes da aplicação.

Nota

Existe uma task chamada server do Grunt que pode ser usada em vez da serve, que usamos no exemplo. Entretanto, ela está depreciada e irá gerar a mensagem "The `server` task has been deprecated. Use `grunt serve` to start a server." no console. Além disso, para não estar o tempo todo reiniciando o servidor, abra uma janela do prompt só para essa task e outra para executar os comandos no projeto.

Outra coisa interessante sobre o servidor do Grunt é que ele gera informações estatísticas ao final quando o paramos. Na **Figura 11** é possível observar isso, o que traz mais poder de controle sobre o que estamos fazendo na aplicação e como os diversos frameworks estão se comportando. Para parar o servidor basta pressionar as teclas Ctrl + C.

Construindo aplicações web com Yeoman e Polymer

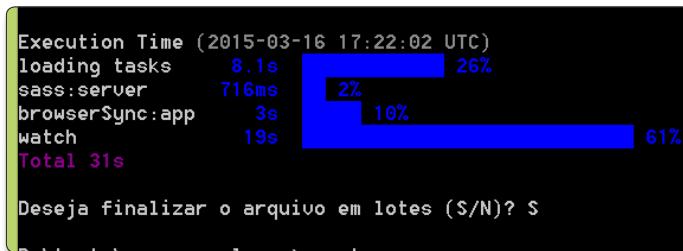


Figura 11. Dados estatísticos de uso do servidor Grunt

Agora, vamos criar um novo elemento do Polymer que representará um post do blog. O comando para gerar isso seria algo como:

```
yo polymer:element post
```

Mas isso acabaria por gerar um erro como “*Error: Element name must contain a dash “-”*”. Isso porque os elementos dos componentes web do Polymer necessitam ter um hífen separando um pseudonome de uma pseudo-categoria. Isso faz parte da especificação HTML5. Portanto, mudemos nossos comandos a partir de agora para conter sempre algum sufixo, tal como:

```
yo polymer:element post-julio
```

Após isso, a interface de linha de comando irá fazer duas perguntas, a primeira relacionada a se você deseja criar um arquivo externo pra guardar o CSS da página, que respondemos Y (yes); e a segunda sobre a importação do arquivo no elements.html, que também dissemos Y. No final, dois novos arquivos (post-julio.html e post-julio.scss) foram gerados no caminho relativo “yeoman-polymer\app\elements\post-julio”. O código do arquivo conterá a declaração do elemento customizado, permitindo que o use no DOM como um elemento HTML comum.

Trabalhando com dados reais

Nosso exemplo de blog precisará de um local para salvar e ler as informações. Para demonstrar o funcionamento com um serviço de armazenamento real, vamos usar o Google Apps Spreadsheets API, que permite facilmente a leitura de conteúdo de qualquer planilha do Google Docs.

Para isso, abra o link de planilha pronta na seção **Links** e dê uma olhada no seu conteúdo. Ela contém o necessário para fazermos nossos posts funcionarem (se desejar mudar o conteúdo crie uma cópia e depois recopie a URL através da opção *File > Publish to the web*). Para o nosso caso, precisaremos converter essa planilha em JSON, o que é possível através do domínio spreadsheets.google.com via segunda URL também da seção **Links**.

A Google Spreadsheets API imprime cada um dos campos com um prefixo especial **post.gsxs\$**. Quando iterarmos cada linha do nosso JSON, vamos referenciar estes campos para recuperar cada informação relevante para cada post. Você pode agora editar o documento html de post criado para lidar com esses atributos da

API Google. Veja na **Listagem 4** o código para isso, e perceba que incluímos logo de início dois valores ao atributo **attributes**:

- **post**: que lerá a informação do título do post, autor, conteúdo e os outros campos que criamos.
- **selected**: que será usado para somente exibir o post se o usuário navegar até a página correta.

Listagem 4. Alteração da página de post para incluir os parâmetros do Google API.

```
01 <polymer-element name="post-julio" attributes="post selected">
02   <template>
03     <link rel="stylesheet" href="post-julio.css">
04
05     <div class="col-lg-4">
06       <template if="{{post.gsxs$atalho.$t === selected}}>
07
08         <h2>
09           <a href="#{{post.gsxs$atalho.$t}}">
10             {{post.gsxs$titulo.$t }}
11           </a>
12         </h2>
13
14         <p>Por {{post.gsxs$autor.$t}}</p>
15
16         <p>{{post.gsxs$conteudo.$t}}</p>
17
18         <p>Publicado em: {{post.gsxs$data.$t}}</p>
19
20         <small>Palavras-chave: {{post.gsxs$keywords.$t}}</small>
21     </template>
22   </div>
23 </template>
24 <script>
25   (function () {
26     Polymer('post-julio', {
27       created: function() {},
28       enteredView: function() {},
29       leftView: function() {},
30       attributeChanged: function(attrName, oldVal, newVal) {}
31     });
32   })();
33 </script>
34 </polymer-element>
```

Agora que temos todos os valores da planilha inseridos na página, podemos criar a página que lidará com o blog de uma forma geral. Para isso execute o seguinte comando:

```
yo polymer:element blog-element
```

Responda às mesmas perguntas feitas antes e isso irá gerar a página blog-element.html no diretório ao lado. Vamos adicionar conteúdo à mesma também, para isso edite o arquivo com o conteúdo da **Listagem 5**.

Nessa listagem temos a demonstração da importação da página de post através da tag **import**, assim como a inclusão de um conteúdo básico, sem nada dinâmico, apenas para ver a página funcionando. Após fazer isso, perceba que o Grunt irá atualizar os arquivos dinamicamente e inserir as referências aos dois arquivos novos dentro do arquivo elements.html, tal como podemos conferir na **Figura 12**.

Vamos adicionar agora mais um generator para incutir a funcionalidade de leitura de JSON no projeto. Esse projeto se chama Polymer JSONP e pode ser instalado rodando o seguinte comando:

```
bower install polymer-elements
```

Durante o processo de instalação, pode ser que o Bower não encontre a versão padrão de alguns componentes, basta selecionar sempre a última neste caso. No final, teremos o componente desejado configurado e precisamos adicionar uma chamada à página HTML dele de dentro da nossa blog-element.html:

```
<link rel="import" href="../../bower_components/polymer-jsonp/polymer-jsonp.html">
```

Isso irá importar todo o conteúdo da página. Após, precisamos adicionar o elemento de tag provindo dessa importação ao corpo da nossa página de blog, apontando para a URL de JSON da nossa planilha, tal como a seguir:

```
<polymer-jsonp auto url="https://spreadsheets.google.com/feeds/list/0Asi6vfhU4mmfdFNrNWNsN0twSzBYWFhnMTh3Y0ZGZ2c/o6/public/values?alt=json-in-script&callback=" response="{{posts}}></polymer-jsonp>
```

Por fim, precisamos fazer referência aos atributos que adicionamos à tag polymer-element da página post-julio.html. Veja na **Listagem 6** o que precisamos adicionar à página de blog.

Na linha 3 temos logo a representação de um loop que irá iterar sobre os elementos de lista que virão do JSON. Cada post será representado por um item de uma lista HTML simples com um link para cada post original. Logo após, temos a declaração do conteúdo do post em si, uma iteração na qual guardará cada elemento de post e se o mesmo está selecionado ou não. Note que aqui estamos fazendo uso da tag post-julio que criamos no arquivo referente.

Para verificar o funcionamento, basta adicionar uma tag blog-element vazia no corpo da página index.html e esperar a página recarregar no browser (**Figura 13**).

Nós conseguimos até agora listar os posts diretamente da planilha online e exibir somente os títulos. Para buscar o restante do conteúdo ao clicar em um dos links de cada post, precisamos primeiro preencher o objeto route da mesma listagem. Para isso, faremos uso de uma biblioteca chamada **Flatiron** que associará o valor da variável independente quando o hash da URL mudar. O projeto está depreciado no Github, mas temos algumas fontes que ainda o mantêm.

Acesse o arquivo de download do fonte desse projeto e dentro do caminho relativo “yeoman-polymer\app\elements\blog-element” copie a pasta “flatiron-director” diretamente para o seu projeto no mesmo diretório.

```

1 <link rel="import" href="../bower_components/core-drawer-panel/core-drawer-panel.html">
2 <link rel="import" href="../bower_components/core-header-panel/core-header-panel.html">
3 <link rel="import" href="../bower_components/core-toolbar/core-toolbar.html">
4 <link rel="import" href="../bower_components/core-icons/core-icons.html">
5 <link rel="import" href="../bower_components/paper-icon-button/paper-icon-button.html">
6 <link rel="import" href="../bower_components/core-menu/core-menu.html">
7 <link rel="import" href="../bower_components/core-item/core-item.html">
8 <link rel="import" href="yo-list/yo-list.html">
9 <link rel="import" href="yo-greeting/yo-greeting.html">
10 <!-- Add your elements here -->
11 <link rel="import" href="post-julio/post-julio.html">
12 <link rel="import" href="blog-element/blog-element.html">
13
14

```

Figura 12. Inclusão automática dos arquivos novos à página elements.html

Listagem 5. Alteração da página de blog para escrever o texto de um post.

```

01 <link rel="import" href="../post-julio/post-julio.html">
02
03 <polymer-element name="blog-element" attributes="">
04 <template>
05   <style>
06     @host { :scope {display: block;} }
07   </style>
08   <span>Eu sou um <b>blog-element</b>. Esse é o meu Shadow DOM.
09   </span>
10 </template>
11
12 <script>
13   Polymer('blog-element', {
14     created: function() {},
15     enteredView: function() {},
16     leftView: function() {},
17     attributeChanged: function(attrName, oldVal, newVal) {}
18   });
19 </script>
20 </polymer-element>

```

Listagem 6. Conteúdo para iteração dos posts na página de blog.

```

01 <!-- Conteúdo da iteração -->
02 <ul>
03 <template repeat="{{post in posts.feed.entry}}>
04   <li><a href="#{{post.gsx$atalho.$t}}">{{post.gsx$titulo.$t}}</a></li>
05 </template>
06 </ul>
07
08 <!-- Conteúdo do post -->
09 <template repeat="{{post in posts.feed.entry}}>
10 <post-julio post="{{post}}" selected="{{route}}></post-julio>
11 </template>

```

Yo, Polymer App!

Eu sou um blog-element. Esse é o meu Shadow DOM.

- [Tudo sobre Political](#)
- [A vida na Austrália](#)
- [Receitas de Bolo](#)
- [Melhores livros do ano](#)

Figura 13. Resultado da iteração dos títulos de cada post no blog

Construindo aplicações web com Yeoman e Polymer

Figura 14. Resultado final do blog

Após isso, faça as mudanças sugeridas na **Listagem 7** ao arquivo de blog-element para que as alterações surtam efeito.

A primeira instrução diz para buscar o arquivo do generator que deve estar corretamente referenciado aqui. A segunda simplesmente cria o elemento e o popula, tornando-o disponível à tag post-julio definida em sequência.

Listagem 7. Alterações no blog-element para adição do route.

```
01 // No início do arquivo
02 <link rel="import" href="flatiron-director/flatiron-director.html">
03
04 // Logo após a tag h1
05 <flatiron-director route="{{route}}" autoHash></flatiron-director>1
```

Depois disso, basta salvar o arquivo e o Grunt irá atualizar a página gerando o resultado visto na **Figura 14**.

Basta clicar em cada link e o conteúdo será atualizado automaticamente, em todas as seções correspondentes com cada coluna da nossa planilha. Como adendo, o leitor pode testar alterando o conteúdo da planilha se a tiver copiado e analisando como a tela se comporta ante às mudanças; assim como acrescentar mais elementos do Polymer e ver como eles vão se integrando junto ao Yeoman.

Adicionando elementos de terceiros

O ecossistema em volta dos componentes web tem crescido muito nos últimos tempos com galerias de componentes como a customelements.io (Figura 15), por exemplo, que guardam os projetos open source de inúmeros profissionais da comunidade.

Na base da página tem uma caixa de texto para busca de projetos do site. Digitando a palavra gravatar você irá encontrar vários projetos que acessam o gravatar.com para universalização de imagens de perfil.

Figura 15. Página oficial do Custom Elements IO

Dentre eles, temos o projeto **gravatar-element** que usaremos com essa finalidade. Efetue o download do arquivo zip do projeto na sua página do Github e adicione a pasta ao nosso projeto. Adicionalmente, você pode copiar diretamente do nosso arquivo de código fonte do artigo. Após isso, faça as mudanças indicadas na **Listagem 8** para configurar o arquivo HTML.

A primeira alteração consiste em adicionar a importação do arquivo de HTML da extensão gravatar-element, e a segunda adiciona o próprio elemento dentro do corpo da página post. O atributo size da linha 5 define o tamanho que a imagem terá quando for impressa. O resultado será semelhante ao da **Figura 16**.

Apesar de termos visto muitos componentes e recursos usando essas duas tecnologias, os componentes web ainda vivem a fase de aprimoramento contínuo aliados aos inúmeros desenvolvedores

que usam boa parte de seu tempo para colocar os projetos para frente. Mas isso já é o suficiente para que você se aventure mais ainda no universo do Polymer e do Yeoman.

Em relação ao nosso projeto, você pode evoluí-lo adicionando mais recursos. Crie uma seção (div) para comentários, cruze as tecnologias com uma linguagem de programação *server side* para permitir o login de usuários, crie páginas de blogs, como Contato, Quem somos, etc.

Tudo sobre Política!

Por Cameron Dias



Lorem Ipsum is simply dummy text of the printing and typesetting industry. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing versions of *Lo*rem *Ip*sum.

Publicado em: 10/1/2015

Palavras-chave: abc

Figura 16. Resultado de inclusão do gravatar

Listagem 8. Inclusão do gravatar-element na página de post.

```
01 // No início da página
02 <link rel="import" href="gravatar-element/src/gravatar.html">
03
04 // Após o elemento de autor
05 <gravatar-element username="{{post.gsx$email.$t}}" size="100">
   </gravatar-element>
```

Ou teste seus conhecimentos criando uma nova aplicação com objetivos diferentes dos aqui apresentados. Adicionalmente, você pode fazer pesquisas no Github para ver projetos prontos de outrem como forma de aprender recursos mais avançados da tecnologia já prontos e testados.

Autor



Júlio Sampaio

É analista de sistema e entusiasta da área de Tecnologia da Informação. Atualmente é consultor na empresa Visagio, trabalhando em projetos de desenvolvimento de sistemas estratégicos, é também instrutor JAVA. Possui conhecimentos e experiência em áreas como Engenharia de Software e Gerenciamento de Projetos, tem também interesse por tecnologias relacionadas ao front-end web.



Links:

Página oficial do projeto Polymer.js.
<https://www.polymer-project.org/0.5/>

Página de download oficial do Node.js.
<https://nodejs.org/download/>

Página de download do Git.
<http://git-scm.com/downloads/>

Página de generators do Yeoman no Github.
<https://github.com/yeoman/>

Planilha de exemplo no Google Docs.
<https://docs.google.com/spreadsheet/pub?key=0Asi6vfhU4mmfdFNrNWNsN0twSzBYWFhnMTh3Y0ZGZ2c&output=html>

URL do conteúdo JSON gerado.
<https://spreadsheets.google.com/feeds/list/0Asi6vfhU4mmfdFNrNWNsN0twSzBYWFhnMTh3Y0ZGZ2c/od6/public/values?alt=json-in-script>

Jogos em HTML5: Criando um jogo 2D com Canvas – Parte 2

Crie um jogo 2D completo usando os recursos do HTML5, Canvas e JavaScript

ESTE ARTIGO FAZ PARTE DE UM CURSO

Na primeira parte deste artigo nós abordamos conceitos iniciais acerca do desenvolvimento do jogo usando Canvas, atrelado à HTML5, JavaScript e CSS3. Essa introdução foi muito importante para que o leitor consiga se adaptar à forma como o Canvas não só impõe sua arquitetura de camadas, mas também a como ele lida com as funções de script para expor o conteúdo e gerenciar o “desenho” gráfico das diversas formas do jogo.

Até o momento criamos três atores principais: o plano de fundo (e seu efeito de movimento, dando a ideia de cenário infinito), o player e os combos de ataque. Além disso, alguns conceitos físicos foram aplicados, muitos deles abstraídos pela própria linguagem, e continuarão sendo abordados até o fim da implementação, haja vista a necessidade de incutir tais efeitos em quase todos os personagens.

Na segunda parte do artigo focaremos no restante do desenvolvimento, que consiste em criar os inimigos e adicionar os efeitos de animação, identificar a colisão dos combos de ataque entre ambos os atores, adicionar sonorização, implementar as funções que irão randomizar a exibição dos inimigos e players, etc. A partir de agora, faremos referência às figuras do pool de objetos e repositórios de forma mais frequente, uma vez que quanto mais objetos precisarem ser criados, maior o processamento que ocorrerá na instanciação desse tipo de estrutura de dados.

Fique por dentro

Este artigo abordará a finalização do jogo 2D usando HTML5 e seu recurso nativo Canvas. O jogo segue o estilo shoot em up, um gênero em que o personagem jogador é controlado na horizontal e tem como objetivo destruir os inimigos por meio de combos ou ataques de poder. Abordaremos desde a criação dos inimigos com seus respectivos efeitos, até a detecção de colisões através do uso da estrutura de dados quadtree, muito famosa por herdar conceitos de árvores binárias em algoritmos. Por fim, adicionaremos sons e efeitos randômicos ao jogo, entendendo como as APIs da HTML5 funcionam para cada um dos casos.

Criando os inimigos

A primeira mudança significativa para criar os inimigos, assim como fizemos para os demais objetos Image, é inserir as referências aos objetos e instanciá-los. Lembre-se que o objeto **repositorio** funciona como uma espécie de contêiner em memória que pré-carrega as variáveis e objetos antes que eles sejam requisitados pelas respectivas funções e exibidos na tela. Isso alivia o processamento e impede que travamentos e lentidão aconteçam em tempo de jogo. Veja na **Listagem 1** o código que usaremos para efetuar essa mudança.

As configurações previstas inicialmente para os demais objetos permanecem. Nas linhas 10 e 11 declaramos e instanciamos os objetos **inimigo** e **comboInimigo**, respectivamente, que servirão para guardar a referência à imagem física de cada um deles. Posteriormente, estes objetos serão usados para receber o valor randômico que implementaremos. A linha 13 traz a variável **numImagens** que já havia sido criada e é incrementada de acordo com a quantidade de objetos que foram carregados no pool **repositorio**.

Listagem 1. Adicionando os objetos de inimigo e combo no objeto repositorio.

```
01 /**
02 * Define um objeto para manter todas as nossas imagens do jogo para
03 * evitar que elas sejam criadas mais de uma vez.
04 */
05 var repositorio = new function() {
06   // Define os objetos de imagens
07   this.plano fundo = new Image();
08   this.player = new Image();
09   this.combo = new Image();
10   this.inimigo = new Image();
11   this.combolinimigo = new Image();
12
13   var numImagens = 5;
14   var numCarregados = 0;
15   function imgCarregada() {
16     numCarregados++;
17     if (numCarregados === numImagens) {
18       window.iniciar();
19     }
20   }
21   this.plano fundo.onload = function() {
22     imgCarregada();
23   }
24   this.player.onload = function() {
25     imgCarregada();
26   }
27   this.combo.onload = function() {
28     imgCarregada();
29   }
30   this.inimigo.onload = function() {
31     imgCarregada();
32   }
33   this.combolinimigo.onload = function() {
34     imgCarregada();
35   }
36
37 // Configura os caminhos (src) das imagens
38 this.plano fundo.src = "imgs/pf.png";
39 this.player.src = "imgs/player1.png";
40 this.combo.src = "imgs/combo1.png";
41 this.inimigo.src = "imgs/enemy1.png";
42 this.combolinimigo.src = "imgs/combo2.png";
43 }
```

Caso a informação não case exatamente com a quantidade de objetos criados dentro dessa classe, você receberá um erro na tela do console JavaScript.

Da linha 30 à 35 temos a adição das funções que efetuarão o pré-carregamento de fato. Veja como o reaproveitamento de código é feito a partir do uso da função **imgCarregada()** que irá lidar com quaisquer tipos de objetos desenháveis. Por último temos a configuração dos atributos src dos mesmos objetos que apontam o caminho relativo das imagens. As imagens informadas podem ser encontradas no arquivo de download do código fonte deste artigo, mas o leitor pode ficar à vontade para usar suas próprias imagens de jogadores e inimigos e ver como o jogo se adapta às mesmas. Não esqueça de verificar sempre os tamanhos de cada arquivo para que os personagens não apareçam cortados.

Esse código ainda não surtirá nenhum efeito, porque precisamos que as demais configurações dos objetos sejam criadas em suas respectivas funções. Para isso, atualizemos a classe Combo, como mostra a **Listagem 2**.

Nota

O objeto de pool do nosso código serve não só para pré-inicializar objetos do tipo Image (que são sempre a maioria em jogos), mas quaisquer objetos que se qualifiquem como “desenháveis”, ou seja, que tenham uma representação gráfica, como animações, gráficos, grafos, etc.

Listagem 2. Atualizando as funções na classe Combo.

```
01 function Combo(objeto) {
02   this.vivo = false; // Será marcado como true se o combo estiver em uso
03   var self = objeto;
04
05 /*
06  * Valores dos combos
07  */
08 this.configurar = function(x, y, velocidade) {
09   this.x = x;
10   this.y = y;
11   this.velocidade = velocidade;
12   this.vivo = true;
13 };
14
15 /*
16  * Função que desenha os combos
17  */
18 this.desenhar = function() {
19   this.context.clearRect(this.x-1, this.y-1, this.largura+1, this.largura+1);
20   //this.context.clearRect(this.x, this.y, this.largura, this.altura);
21   this.x += this.velocidade;
22   if (self === "combo" && this.x <= 0 - this.largura) {
23     return true;
24   } else if (self === "combolinimigo" && this.y >= this.alturaCanvas) {
25     return true;
26   } else {
27     if (self === "combo") {
28       this.context.drawImage(repositorio.combo, this.x, this.y);
29     } else if (self === "combolinimigo") {
30       this.context.drawImage(repositorio.combolinimigo, this.x, this.y);
31     }
32     return false;
33   }
34 };
35
36 /*
37  * Reinicia as propriedades do combo
38  */
39 this.limpar = function() {
40   this.x = 0;
41   this.y = 0;
42   this.velocidade = 0;
43   this.vivo = false;
44 };
45 }
46 Combo.prototype = new Desenhavel();
```

Nessa listagem podemos observar dois comportamentos que serão distintos dos combos que já criamos antes:

- Primeiro, os inimigos irão lançar combos diferentes em relação aos do player, e esses mesmos combos também têm comportamentos diferentes dos demais, uma vez que durarão menos tempo e serão criados de forma aleatória, sem o disparo de um comando via evento JavaScript;
- E segundo que o objeto Inimigo precisará de seu próprio pool de combos para lidar com o lançamento deles de forma definitiva, até que seja destruído no jogo; logo, precisamos atualizar o objeto

Pool para se adaptar a essa definição. Além disso, o mesmo pool será responsável por guardar o objeto de combo do inimigo em si, o que será útil quando quisermos, por exemplo, criar uma nova horda de inimigos quando a atual for destruída.

Na linha 1 nós recebemos por parâmetro a string **objeto** que será responsável por definir qual o “tipo” do objeto combo que está sendo criado, isto é, se será um combo do player ou do inimigo. Na linha 3 nós adicionamos a referência a uma variável local que será usada no método **desenhar()** da linha 18. Perceba que nesse mesmo método, nós determinamos se o combo já passou do limite da tela (apagou-se) e, caso positivo, desenhamos a imagem apropriada no Canvas. A partir da linha 21, iniciamos os testes para verificar qual o tipo do objeto, bem como se devemos desenhá-lo ou não, ou passá-lo para a próxima função com o valor booleano.

Já em relação ao objeto Pool, na **Listagem 3** podemos observar que a única mudança necessária se faz no método **iniciar()** que receberá também as operações condicionais de verificação do tipo de objeto a ser criado, uma vez que isso implicará no tipo de referência que teremos em memória para os diferentes vetores de pool. Veja que logo na linha 4 a declaração da função já recebe um novo argumento que representará o tipo do objeto e será atrelado à variável “tamanho” já criada, para guardar a quantidade de objetos de cada pool (que deverá variar no momento da instanciação do mesmo).

Listagem 3. Modificando a função **iniciar()** do objeto Pool.

```
01 /*
02 * Popula o vetor de pool com objetos de combo
03 */
04 this.iniciar = function(objeto) {
05   if (objeto == "combo") {
06     for (var i = 0; i < tamanho; i++) {
07       // Inicializa o objeto de combo
08       var combo = new Combo("combo");
09       combo.iniciar(0, 0, repositorio.combo.width, repositorio.combo.height);
10       pool[i] = combo;
11     }
12   } else if (objeto == "inimigo") {
13     for (var i = 0; i < tamanho; i++) {
14       // Inicializa o objeto de inimigo
15       var inimigo = new Inimigo();
16       inimigo.iniciar(0, 0, repositorio.inimigo.width, repositorio.inimigo.height);
17       pool[i] = inimigo;
18     }
19   } else if (objeto == "combolnimigo") {
20     for (var i = 0; i < tamanho; i++) {
21       // Inicializa o objeto de combo inimigo
22       var combo = new Combo("combolnimigo");
23       combo.iniciar(0, 0, repositorio.combolnimigo.width,
24                     repositorio.combolnimigo.height);
25       pool[i] = combo;
26   }
27};
```

Perceba também que o tipo do objeto de inimigo mudou em relação aos demais, mas ainda não criamos tal classe. Nela, os inimigos imitarão o movimento de OVNIs comuns em outros jogos do estilo, se movendo de cima para baixo e vice versa em conjunto e sincronizados. Além disso, eles não terão movimentos horizontais (que podem ser facilmente adicionados pelo leitor se desejar) e não poderão ultrapassar o limite do topo e da base da tela do jogo que iremos estipular. Finalmente, nós os faremos descer no início do jogo ou quando cada nova horda de inimigos aparecer, com um efeito de animação. Veja na **Listagem 4** o código para criar a classe Inimigo (adicone-o logo abaixo da classe Player).

Os objetos inimigos terão uma chance pequena de atirar um combo de ataque por frame. Ao fazer isso, nós fazemos com que o jogo adquira uma característica mais inesperada em vez de usar padrões fixos que se tornam fáceis de memorizar com o tempo. Na linha 5 nós criamos a variável **percTiros**, que irá guardar o percentual de combos que deverão ser lançados por segundo. O valor .01 pode parecer muito pequeno, mas quando considerarmos que eles se movem 60 vezes por segundo, o objetivo é que eles atirem ao menos uma vez a cada dois ou três segundos.

Uma vez que cada inimigo estará se movendo em grupo para cima e para baixo o tempo todo, cada um precisa saber o quanto longe pode ir antes de mudar de direção. Essa mudança é relativa a cada inimigo individual, então nós não podemos simplesmente configurar um limite para todos os inimigos e esperar que funcione. Sendo assim, a função **configurar()** na linha 11 recebe os valores das posições x e y do inimigo e então lhe dá três limites básicos: uma borda no topo, outra na base (que será importante para a animação inicial de descida) e outra borda na esquerda. Logo, cada inimigo poderá se mover dentro do limite estipulado de 190px. O restante das variáveis da função segue o mesmo princípio de uso que os demais objetos.

Na linha 27 damos início à declaração da função **desenhar()**, que configura a direção e o exato momento em que o player deverá mudar a rota (linhas 33 a 37). Após tocar a borda da base invisível, a velocidade do inimigo será levemente reduzida. Na linha 40 nós usamos a função **Math.random()** para gerar um número aleatório entre 1 e 100 com o objetivo de dizer se o inimigo irá atirar nesse frame ou não. Se o leitor desejar que o inimigo demore mais, basta aumentar o intervalo aleatório. Já na linha 48 temos a função **atirar()** que será chamada pela função anterior e lidará com o acesso direto ao pool de combos. Note que o último parâmetro passado à função **get()** para recuperar o combo é referente à velocidade com que as balas irão trafegar e você pode ajustar esse valor para inibir dificuldade ao jogo, por exemplo. Finalmente, na linha 54 temos a função **limpar()** que reiniciará todos os valores dos atributos do objeto em questão.

Você talvez se pergunte por que o objeto Inimigo não tem um pool próprio como fizemos com o objeto Player. A razão para isso é novamente para nos manter à frente. Se cada player tem seu próprio pool de combos, então nós iremos animar somente

Listagem 4. Criando a classe Inimigo para representar os objetos alienígenas inimigos.

```
01 /**
02 * Cria o objeto de Inimigo.
03 */
04 function Inimigo() {
05     var perctTiros = .01;
06     var chance = 0;
07     this.vivo = false;
08     /*
09      * Configura os valores dos inimigos
10     */
11     this.configurar = function(x, y, velocidade) {
12         this.x = x;
13         this.y = y;
14         this.velocidade = velocidade;
15         this.velocidadeX = 0;
16         this.velocidadeY = velocidade;
17         this.vivo = true;
18         this.bordaEsquerda = this.x - 90;
19         this.bordaTopo = this.y + 50;
20         this.bordaBase = this.y + 190;
21         console.log('Set');
22         // debugger
23     };
24     /*
25      * Move o inimigo
26     */
27     this.desenhar = function() {
28         this.context.clearRect(this.x-1, this.y, this.largura+1, this.altura);
29         this.x += this.velocidadeX;
30         this.y += this.velocidadeY;
31         console.log(this.x + ' - ' + this.y);
32         // debugger
33     if (this.y >= this.bordaBase) {
34         this.velocidadeY = -this.velocidade;
35     } else if (this.y <= this.bordaTopo) {
36         this.velocidadeY = this.velocidade;
37     }
38     this.context.drawImage(repositorio.inimigo, this.x, this.y);
39     // Inimigo tem a chance de atirar a cada instante
40     chance = Math.floor(Math.random()*101);
41     if (chance/100 < perctTiros) {
42         this.atirar();
43     }
44    };
45    /*
46     * Atira um combo
47     */
48    this.atirar = function() {
49        jogo.combolInimigoPool.get(this.x+this.largura/2, this.y+this.altura, -2);
50    };
51    /*
52     * Reinicia os valores do inimigo
53     */
54    this.limpar = function() {
55        this.x = 0;
56        this.y = 0;
57        this.velocidade = 0;
58        this.velocidadeX = 0;
59        this.velocidadeY = 0;
60        this.vivo = false;
61    };
62}
63 Inimigo.prototype = new Desenhavel();
```

aqueles combos o tempo em que o player estiver vivo. Se o player morrer, o pool de objetos do inimigo poderá removê-lo de lá, limpando-o e depois movendo-o novamente para o mesmo pool. Isso faria com que todas os combos que o inimigo atirou e ainda estivessem na tela simplesmente parassem de se mover e permanecessem lá. É o que chamamos de forte acoplamento.

Para as balas se moverem, elas dependem “fortemente” do inimigo ainda estar vivo, uma vez que perder o inimigo implicará em consequentemente perder os combos dele. O que queremos de fato é um sistema de baixo acoplamento onde se o player morre os combos continuam existindo na tela. Para fazer isso, nós teremos de criar um pool de objetos que guarde todos os combos para todos os inimigos e que deverá ser controlado pelo objeto Jogo (veja a **Listagem 5** para isso).

Perceba que o uso de Prototypes volta a acontecer nessa listagem dentro da função iniciar() que irá lidar com a configuração genérica dos atributos de contexto, largura e altura do canvas. Após isso, temos a inicialização do pool de inimigos e do pool de combo dos inimigos, respectivamente. Veja que eles são declarados em ordem e quantidade equivalentes, e que suas definições de largura, altura, e onde eles irão aparecer no plano cartesiano devem ser definidas logo nessa função para uso posterior dentro do próprio objeto.

Nós teremos nove inimigos, cuja quantidade pode ser alterada na iteração da linha 53 em detrimento da divisão feita no if da linha 56. A fórmula funciona assim: a variável **espaco** irá

calcular o espaço entre cada um dos inimigos na tela e será usada em seguida para determinar qual a posição y do próximo inimigo, acrescida do seu valor. O primeiro for irá configurar a quantidade de inimigos e buscará cada objeto diretamente de dentro do objeto **poolInimigos** (linha 54). Uma vez com o inimigo em mãos, a função calcula a distância x de forma fixa adicionando o valor 15 (que também pode ser alterado para aumentar ou diminuir a distância) à largura do objeto. Se você deseja mudar a quantidade de inimigos, então ela deverá ser um exponencial do número referenciado na operação de **mod** da linha 54, isso porque os objetos serão exibidos em fileiras uma abaixo da outra. O valor 600 configurado à variável x deve ser o mesmo antes e depois do for, uma vez que será usado para definir onde os objetos começarão a ser impressos na tela. No final, iniciamos o pool de combos do inimigo passando a String de identificação do mesmo (linha 62).

Além disso, veja que como queremos que os inimigos começem fora da tela, nós iniciaremos a linha da base logo acima da borda do topo (y = -altura), deixando a maioria dos inimigos com 100px.

Nota

As imagens do jogo já vieram com tamanhos pré-definidos para as dimensões de tela que estamos usando. Caso deseje, você pode aumentar a imagem fisicamente e o jogo irá se adaptar automaticamente.

Jogos em HTML5: Criando um jogo 2D com Canvas – Parte 2

Listagem 5. Alterando a classe Jogo para receber os inimigos.

```
01 /**
02 * Cria um objeto mais genérico que se encarregará de lidar com os dados do jogo.
03 */
04 function Jogo() {
05     this.iniciar = function() {
06         // Recupera o elemento canvas
07         this.pfCanvas = document.getElementById('planoDeFundo');
08         this.playerCanvas = document.getElementById('player_mov');
09         this.principalCanvas = document.getElementById('principal');
10         var retorno = false;
11
12         // Testa para verificar se o canvas é suportado
13         if(this.pfCanvas.getContext) {
14             // inicializa o objeto de plano de fundo
15             this.planofundo = new PlanoFundo();
16             this.planofundo.iniciar(0,0); // Inicia no ponto 0,0
17
18             this.pfContext = this.pfCanvas.getContext('2d');
19             this.playerContext = this.playerCanvas.getContext('2d');
20             this.principalContext = this.principalCanvas.getContext('2d');
21
22             // inicializa os objetos configurando as propriedades em questão
23             PlanoFundo.prototype.context = this.pfContext;
24             PlanoFundo.prototype.larguraCanvas = this.pfCanvas.width;
25             PlanoFundo.prototype.alturaCanvas = this.pfCanvas.height;
26
27             Player.prototype.context = this.playerContext;
28             Player.prototype.larguraCanvas = this.playerCanvas.width;
29             Player.prototype.alturaCanvas = this.playerCanvas.height;
30
31             Combo.prototype.context = this.principalContext;
32             Combo.prototype.larguraCanvas = this.principalCanvas.width;
33             Combo.prototype.alturaCanvas = this.principalCanvas.height;
34
35             Inimigo.prototype.context = this.principalContext;
36             Inimigo.prototype.larguraCanvas = this.principalCanvas.width;
37             Inimigo.prototype.alturaCanvas = this.principalCanvas.height;
38
39         // Inicializa o objeto player
40         this.player = new Player();
41         // Configura o player para aparecer no meio da tela à esquerda
42         var playerIniX = repositorio.player.width / 4;
43         var playerIniY = this.playerCanvas.height / 3 + repositorio.player.height;
44         this.player.iniciar(playerIniX, playerIniY, repositorio.player.width,
45                             repositorio.player.height);
46
47         this.poolInimigos = new Pool(30);
48         this.poolInimigos.iniciar("inimigo");
49         var altura = repositorio.inimigo.height;
50         var largura = repositorio.inimigo.width;
51         var x = 600;
52         var y = -altura;
53         var espaco = 120 * 0.7;
54         for (var i = 1; i <= 9; i++) {
55             this.poolInimigos.get(x,y,1);
56             x += largura + 15;
57             if (i % 3 == 0) {
58                 x = 600;
59                 y += espaco
60             }
61         this.combolInimigoPool = new Pool(50);
62         this.combolInimigoPool.iniciar("combolInimigo");
63
64         retorno = true;
65     }
66     return retorno;
67 };
68
69 // Inicia o loop de animação
70 this.jogar = function() {
71     this.player.desenhar();
72     animar();
73 };
74 }
```

Após isso, nós também precisamos mudar a forma como inicializamos o objeto de pool de combos na classe Player. Para isso modifique a chamada logo no início da classe para a seguinte:

```
this.poolCombos.iniciar("combo");
```

Finalmente, vamos modificar a função externa `animar()` que se encontra logo após a declaração da classe Jogo (**Listagem 6**).

A listagem basicamente mostra a adição das chamadas às funções `animar()` dos dois pools criados. Agora é só salvar o código e executar a página `index.html` no navegador e o resultado deverá ser semelhante ao ilustrado na **Figura 1**.

Identificando colisões

Existem inúmeros algoritmos para detectar quando dois objetos se tocam em um universo 2D. Para o nosso jogo, não precisaremos implementar um algoritmo complexo para lidar com toda a física intrínseca a esse tipo de recurso, mas sim um teste de caixa preta para fazer isso por nós:

```
if (objeto1.x < objeto2.x + objeto2.largura && objeto1.x + objeto1.largura > objeto2.x &&
    objeto1.y < objeto2.y + objeto2.altura && objeto1.y + objeto1.altura > objeto2.y {
```



Figura 1. Configuração de nosso projeto

Listagem 6. Adicionando as funções de animação à função externa `animar()`.

```
01 function animar() {
02     getFrameAnimacao( animar );
03     jogo.planofundo.desenhar();
04     jogo.player.mover();
05     jogo.player.poolCombos.animar();
06     jogo.poolInimigos.animar();
07     jogo.combolInimigoPool.animar();
08 }
```

Esse tipo de algoritmo de teste considera dois objetos e verifica se os limites do primeiro estão em contato com os limites do segundo. Isso requer quatro checagens, uma para cada borda da caixa invisível que envolve o objeto. Para implementar a detecção de colisão nós poderíamos simplesmente colocar esse algoritmo em um loop e verificar cada objeto em relação aos outros. Entretanto, se pensarmos na quantidade de objetos que teremos na tela por vez, isso pode ficar muito custoso, gerando cerca de cinco mil verificações para apenas uma checagem. Além disso, nós também precisaríamos executar esse algoritmo a cada frame, o que definitivamente pode deixar nosso jogo muito lento.

Para resolver esse problema, devemos reduzir a quantidade de objetos sendo verificados e, para isso, faremos uso de uma técnica conhecida como “particionamento espacial”. Esse tipo de técnica pode ser facilmente substituído pela iteração sobre a lista de objetos e verificação da colisão no jogo, mas faremos uso dela aqui para que o leitor tenha todos os subsídios necessários para lidar com o mesmo quando mais personagens, fases, desafios, etc. forem adicionados.

Ela basicamente divide um espaço confinado (neste caso a tela do jogo) em espaços menores. Ao fazer isso, nós podemos rapidamente determinar quais objetos pertencem à mesma área e só temos de executar a detecção de colisão sobre estes objetos. Qualquer objeto que não pertença ao mesmo espaço não pode colidir, portanto economizamos tempo descartando-o. O número de detecções cai para aproximadamente 100/frame, ou seja, um ganho enorme em performance. Para implementar essa técnica, nós usaremos uma estrutura de dados chamada **quadtree**.

O que é um Quadtree?

Um quadtree é uma estrutura de dados usada para dividir uma região 2D em partes gerenciáveis menores. É nada mais que uma extensão de uma árvore binária, mas em vez de dois nós filhos ele tem quatro. Na **Figura 2** temos um exemplo de imagem que representa um espaço 2D particionado por um algoritmo de quadtree. Observe que ele criará os nós em ordem anti-horária.

Cada objeto adicionado ao quadtree é inserido em um nó específico. Isso se assemelha à forma como o algoritmo de busca binária atua, reduzindo o escopo de busca na metade a cada nova pesquisa. Para o quadtree, à medida que mais objetos são adicionados a um nó, o mesmo será partitionado novamente em mais quatro subnós (**Figura 3**).

Para criar o nosso objeto quadtree, vamos declarar primeiramente a classe e criar os atributos obrigatórios para lidar com o controle dos demais objetos. Para isso, adicione o código da **Listagem 7** ao final do nosso arquivo de scripts.

Esse ainda não é o objeto completo (ele será o mais complexo do jogo). Vamos detalhando função por função para ficar melhor o entendimento. Na linha 4 declaramos o construtor que receberá o argumento **limiteCaixa**, que servirá basicamente para guardar as informações do eixo cartesiano mais as dimensões do objeto. Repare que mesmo não tendo tipificação no JavaScript, esse objeto se assemelha ao tipo Rectangle encontrado em algumas linguagens

de programação, e deverá ser sempre trafegado com todas as informações preenchidas. Os demais atributos consistem em listas e vetores que usaremos mais à frente para controlar o nível, posição e outros valores do quadtree.

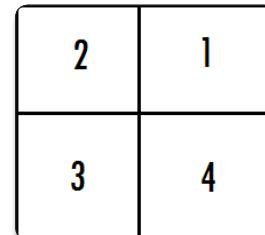


Figura 2. Exemplo de imagem 2D cortada pelo quadtree

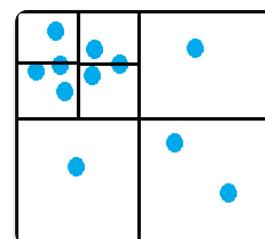


Figura 3. Exemplo de subnó gerado pelo quadtree devido ao aumento de objetos

Listagem 7. Criando a classe quadtree.

```

01 /**
02 * Objeto ArvoreQuadrante.
03 */
04 function ArvoreQuadrante(limiteCaixa, nvl) {
05     var maxObjetos = 10;
06     this.limites = limiteCaixa || {
07         x: 0,
08         y: 0,
09         width: 0,
10         height: 0
11     };
12     var objetos = [];
13     this.nos = [];
14     var nivel = nvl || 0;
15     var maxNiveis = 5;
16     /*
17      * Limpa a arvoreQuadrante e todos os nós dos objetos
18     */
19     this.limpar = function() {
20         objetos = [];
21         for (var i = 0; i < this.nos.length; i++) {
22             this.nos[i].limpar();
23         }
24         this.nos = [];
25     };
26     /*
27      * Recupera todos os objetos na arvoreQuadrante
28     */
29     this.getTodosObjetos = function(objRetornados) {
30         for (var i = 0; i < this.nos.length; i++) {
31             this.nos[i].getTodosObjetos(objRetornados);
32         }
33         for (var i = 0, len = objetos.length; i < len; i++) {
34             objRetornados.push(objetos[i]);
35         }
36         return objRetornados;
37     };
38 }
```

Jogos em HTML5: Criando um jogo 2D com Canvas – Parte 2

Listagem 8. Criando funções CRUD para auxiliar nas operações quadtree.

```
01 /*  
02 * Retorna todos os objetos que obj pode colidir  
03 */  
04 this.procurarObjs = function(objRetornados, obj) {  
05   if (typeof obj === "undefined") {  
06     console.log("OBJETO UNDEFINED");  
07     return;  
08   }  
09   var indice = this.getIndice(obj);  
10  if (indice != -1 && this.nos.length) {  
11    this.nos[indice].procurarObjs(objRetornados, obj);  
12  }  
13  for (var i = 0, len = objetos.length; i < len; i++) {  
14    objRetornados.push(objetos[i]);  
15  }  
16  return objRetornados;  
17};  
18  
19 this.inserir = function(obj) {  
20  if (typeof obj === "undefined") {  
21    return;  
22  }  
23  if (obj instanceof Array) {  
24    for (var i = 0, len = obj.length; i < len; i++) {  
25      this.inserir(obj[i]);  
26    }  
27  }  
28};  
29 if (this.nos.length) {  
30   var indice = this.getIndice(obj);  
31   // Só add o obj ao subnó se ele encaixar completamente  
32   if (indice != -1) {  
33     this.nos[indice].inserir(obj);  
34     return;  
35   }  
36 }  
37 objetos.push(obj);  
38 // Previne divisão infinita  
39 if (objetos.length > maxObjetos && nível < maxNíveis) {  
40   if (this.nos[0] == null) {  
41     this.dividir();  
42   }  
43   var i = 0;  
44   while (i < objetos.length) {  
45     var indice = this.getIndice(objetos[i]);  
46     if (indice != -1) {  
47       this.nos[indice].inserir([objetos.splice(i, 1)[0]]);  
48     }  
49     else {  
50       i++;  
51     }  
52   }  
53 }  
54};
```

A função limpar() da linha 19 é bem simples e só varre todos os nós do objeto, chamando o método limpar de cada um dos objetos (herdados de Desenhavel) e zerando o vetor dos nós ao final. Já a função **getTodosObjetos()** na linha 29 realiza a mesma iteração sobre os nós modificando o valor do mesmo objeto recebido por parâmetro.

Em seguida, adicione o código da **Listagem 8** logo após os métodos recém criados. Eles serão úteis para efetuar operações de CRUD dentro dos objetos quadtree.

Essa listagem traz duas funções do tipo CRUD muito importantes que tem funções bem definidas, a saber:

- A linha 4 declara a função **procurarObjs()** que fará uma varredura em todos os objetos (sem considerar os nós) e retornará os que estão elegíveis a sofrer colisões.

- Na linha 5 temos um teste básico que retornará a execução do código à função chamadora se o objeto não for válido;

- Das linhas 9 à 12 temos uma recursividade que buscará todos os índices dos objetos em questão para assim adicioná-los ao retorno do método;

- A linha 14 faz referência a um método da API JavaScript que simula os conceitos de pilhas e outras estruturas de dados, adicionando todos os objetos encontrados ao retorno.

- A função **inserir()** (linha 19), por sua vez, faz as mesmas verificações iniciais que a função anterior e recebe somente o objeto que vai ser inserido por parâmetro.

- A linha 23 explicita o uso do operador instanceof para verificar se estamos lidando de fato com um vetor;

- Mais uma vez recuperamos o índice de cada objeto na linha 30, só que agora para inseri-lo. Caso a inserção seja feita, a execução do método para aí mesmo;

- O bloco if da linha 39 se encarrega de efetuar as divisões de nós. Além disso, ele se certificará de não criar divisões infinitas dadas as condições estabelecidas e a quantidade de objetos por região.

Até agora, vimos o uso da função **getIndice()** várias vezes nas funções criadas. Veja na **Listagem 9** o código de criação desse método.

Essa função irá atuar principalmente no rastreamento de objetos via limites do quadrante. Veja que as cinco primeiras linhas dela lidam exclusivamente com a partição da tela que virá como parâmetro e seus respectivos limites de bordas. Após isso, dois testes simples são feitos:

- O primeiro, na linha 10, verifica se o objeto está antes da linha vertical média que corta o quadrante. Caso positivo, verificamos se está acima ou abaixo da linha horizontal.
- O segundo, na linha 20, verifica exatamente o contrário.

Observe que, de acordo com os testes, o atributo **índice** é atualizado sempre respeitando a ordem anti-horária do algoritmo.

Por último, precisamos somente criar o método que fará a divisão de fato dos quadrantes. Para isso, adicione o código da **Listagem 10** após todas essas funções na classe quadtree.

Essa função não traz muitas novidades, exceto pela criação dos objetos quadtree dentro da própria classe, associando para cada um dos quatro nós um objeto de mesmo tipo. Note que os valores de Rectangle de cada nó são recuperados recursivamente através do próprio objeto Rectangle, uma vez que o mesmo começa zerado mas é incrementado ao longo da execução do jogo. Além disso, sempre que um novo nó precisar ser quebrado em mais quatro, a

Listagem 9. Adicionando o método getIndice() para recuperação dos índices no quadtree.

```
01 this.getIndice = function(obj) {  
02   var indice = -1;  
03   var pontoMedioVertical = this.limites.x + this.limites.width / 2;  
04   var pontoMedioHorizontal = this.limites.y + this.limites.height / 2;  
05   // Objeto pode caber completamente dentro do quadrante do topo  
06   var quadranteTopo = (obj.y < pontoMedioHorizontal && obj.y + obj.height  
   < pontoMedioHorizontal);  
07   var quadranteBase = (obj.y > pontoMedioHorizontal);  
08   // Objeto pode caber completamente dentro do quadrante esquerdo  
09   if (obj.x < pontoMedioVertical &&  
10     obj.x + obj.width < pontoMedioVertical) {  
11     if (quadranteTopo) {  
12       indice = 1;  
13     }  
14   }  
15   else if (quadranteBase) {  
16     indice = 2;  
17   }  
18 }  
19 // Objeto pode caber completamente dentro do quadrant direito  
20 else if (obj.x > pontoMedioVertical) {  
21   if (quadranteTopo) {  
22     indice = 0;  
23   }  
24   else if (quadranteBase) {  
25     indice = 3;  
26   }  
27 }  
28 return indice;  
29};
```

Listagem 10. Adicionando o método getIndice() para recuperação dos índices no quadtree.

```
01 this.dividir = function() {  
02   // Uso do Bitwise  
03   var subLargura = (this.limites.width / 2) | 0;  
04   var subAltura = (this.limites.height / 2) | 0;  
05   this.nos[0] = new ArvoreQuadrante({  
06     x: this.limites.x + subLargura,  
07     y: this.limites.y,  
08     width: subLargura,  
09     height: subAltura  
10   }, nivel+1);  
11   this.nos[1] = new ArvoreQuadrante({  
12     x: this.limites.x,  
13     y: this.limites.y,  
14     width: subLargura,  
15     height: subAltura  
16   }, nivel+1);  
17   this.nos[2] = new ArvoreQuadrante({  
18     x: this.limites.x,  
19     y: this.limites.y + subAltura,  
20     width: subLargura,  
21     height: subAltura  
22   }, nivel+1);  
23   this.nos[3] = new ArvoreQuadrante({  
24     x: this.limites.x + subLargura,  
25     y: this.limites.y + subAltura,  
26     width: subLargura,  
27     height: subAltura  
28   }, nivel+1);  
29};
```

mesma implementação pode ser reaproveitada mudando apenas a referência ao vetor de nós.

Com todos os nós e subnós indexados, agora é hora de adaptar o restante dos objetos para receber essa mudança. Entretanto, ainda falta um fator para finalizar a detecção de colisão 2D: nem todos os objetos poderão colidir com todos os outros objetos. Por exemplo, um player não pode colidir com seus próprios combos, e a mesma coisa para os inimigos. Então, precisamos checar se dois objetos podem colidir antes de checar se eles estão colidindo de fato. Isso significa que cada objeto irá guardar uma lista de objetos que podem colidir com eles mesmos e o algoritmo de colisão irá comparar a lista através dos tipos de cada um.

Listagem 11. Adicionando atributos e funções para detecção na classe Desenhavel.

```
01 this.colidivelCom = "";  
02 this.isColidindo = false;  
03 this.tipo = "";  
04  
05 // ...  
06  
07 this.isColidivelCom = function(obj) {  
08   return (this.colidivelCom === obj.tipo);  
09};
```

Para iniciar essa implementação, vamos editar a classe Desenhavel para guardar mais alguns atributos (**Listagem 11**).

As variáveis declaradas servirão para controlar com quem cada objeto desenhável poderá colidir e verificar se o mesmo está colidindo no frame de teste. Além disso, a função **isColidivelCom()** irá retornar uma comparação simples entre o objeto que está requisitando tal resposta, nos dizendo quem não deverá ser reimpresso no canvas. Através dessa mudança, cada objeto que herde da classe Desenhavel terá que obrigatoriamente lidar com essa lista de objetos que podem colidir uns com os outros.

Comecemos portanto pela classe Combo. Adicione as mudanças sugeridas pela **Listagem 12** para tornar o objeto apto a lidar com isso.

Listagem 12. Efetuando as mudanças na classe Combo.

```
01 if (this.isColidindo) {  
02   return true;  
03 } if (self == "combo" && this.x <= 0 - this.largura) { ...  
04  
05 // E na função limpar  
06 this.isColidindo = false;
```

Apenas as funções desenhar() e limpar() sofrerão alterações. A primeira adiciona uma checagem para ver se o objeto está colidindo. Caso positivo, retorna true para o pool de objetos para que ele saiba que o combo pode ser reusado. A função limpar() apenas zera a propriedade booleana.

Jogos em HTML5: Criando um jogo 2D com Canvas – Parte 2

A próxima classe que deverá ser adaptada é Player. Vejamos na **Listagem 13** as mudanças que devem ser feitas.

Na primeira linha temos a definição do tipo de objeto que pode colidir com os players. Após isso, definimos qual o tipo do objeto em si e nas linhas 5 à 7 envolvemos a chamada à função desenhar() dentro de uma condição que verifica se o objeto não está colidindo. Essa verificação será mais comum daqui pra frente e é necessária para não efetuar ações quando o objeto não for mais exibido. Ao fim, adicionamos mais uma condição ao if da linha 11 que também só irá disparar combos se o objeto não estiver colidindo.

Dando sequência, vamos modificar agora a classe Inimigo para o que está exibido na **Listagem 14**.

Listagem 13. Efetuando as mudanças na classe Player.

```
01 this.colidivelCom = "combolinimigo";
02 this.tipo = "player";
03
04 // E na função mover
05 if (!this.isColidindo) {
06   this.desenhar();
07 }
08
09 // ...
10
11 if (STATUS_CHAVES.space && cont >= intervaloTiros && !this.isColidindo) { ... }
```

Listagem 14. Efetuando as mudanças na classe Inimigo.

```
01 this.colidivelCom = "combo";
02 this.tipo = "inimigo";
03
04 if (!this.isColidindo) {
05   this.context.drawImage(repositorio.inimigo, this.x, this.y);
06
07   chance = Math.floor(Math.random()*101);
08   if (chance/100 < perctTiros) {
09     this.atirar();
10   }
11   return false;
12 } else {
13   return true;
14 }
15
16 // E na função limpar
17 this.isColidindo = false;
```

Essa listagem faz o mesmo que as outras em relação ao tipo do objeto. Na linha 4 também verificamos se o objeto não está em colisão para assim realizar a regra que já existia. A função agora terá retorno booleano, haja vista a necessidade de controlar esse fluxo. No final, também resetamos a variável isColidindo na função limpar() da classe.

A próxima classe para alterar é Pool. Veja na **Listagem 15** o que precisamos mudar nela.

As mudanças nessa classe se darão essencialmente na função iniciar(). No início, temos as mesmas configurações de tipo e objeto de colisão. Na linha 9 vemos a inclusão de um novo método,

getPool(), que deverá ser adicionado após a função iniciar(). Ela será responsável por retornar todos os objetos vivos no pool em forma de vetor que serão inseridos dentro do quadtree.

Para implementar o quadtree em nosso jogo, nós precisamos instanciar um novo objeto dentro da classe Jogo, logo antes da linha de retorno:

```
this.arvoreQuadrante = new ArvoreQuadrante({x:0,y:0,width:this.principalCanvas.width,height:this.principalCanvas.height});
```

Listagem 15. Efetuando as mudanças na classe Pool.

```
01 // No primeiro
02 combo.colidivelCom = "inimigo";
03 combo.tipo = "combo";
04
05 // No terceiro for
06 combo.colidivelCom = "player";
07 combo.tipo = "combolinimigo";
08
09 this.getPool = function() {
10   var obj = [];
11   for (var i = 0; i < tamanho; i++) {
12     if (pool[i].vivo) {
13       obj.push(pool[i]);
14     }
15   }
16   return obj;
17 }
```

Isso criará uma instância com os valores do rectangle preenchidos a partir do nosso canvas principal. Essa configuração é importante pois definirá as dimensões do jogo e quando mudarmos a largura ou altura dele, o quadtree irá se ajustar automaticamente. Para finalizar, adicionemos o código da **Listagem 16** ao nosso arquivo.

A primeira coisa que fazemos é limpar o quadtree e adicionar todos os objetos a ele. Uma vez adicionados, podemos executar o algoritmo de detecção de colisão por meio da função **detectarColisao()**. Logo após, movemos e animamos todos os objetos para garantir que qualquer objeto que colidir em um frame não seja redesenhadado no fim do mesmo. Na linha 10 criamos a dita função, que recupera todos os objetos no quadtree e os salva em um vetor simples. Então, percorremos o vetor e testamos cada um dos objetos que podem ser colididos. No final, rodamos o algoritmo para verificar se algum objeto colidiu no frame atual. Isso nos assegurará uma checagem de colisões em 60 fps.

Agora basta executar o jogo novamente e o resultado será semelhante ao da **Figura 4**. Perceba que alguns dos inimigos sumiram, bem como o próprio player, isso porque eles foram todos atingidos no exemplo.

Randomizando os desenháveis

Para randomizar a impressão dos objetos desenháveis (plano de fundo, player e inimigos), basta gerar um número randômico

através da classe Math com a quantidade de opções disponíveis e verificar por meio de uma cláusula switch qual imagem associar. Para isso, no objeto repositorio modifique a associação aos caminhos relativos de cada imagem tal como demostrado na **Listagem 17**.

Listagem 16. Código final para detecção de colisões.

```

01 // No início da função externa animar()
02 jogo.arvoreQuadrante.limpar();
03 jogo.arvoreQuadrante.inserir(jogo.player);
04 jogo.arvoreQuadrante.inserir(jogo.player.poolCombos.getPool());
05 jogo.arvoreQuadrante.inserir(jogo.poolInimigos.getPool());
06 jogo.arvoreQuadrante.inserir(jogo.comboInimigoPool.getPool());
07 detectarColisao();
08
09 // Após a função animar()
10 function detectarColisao() {
11     var objetos = [];
12     jogo.arvoreQuadrante.getTodosObjetos(objetos);
13     for (var x = 0, len = objetos.length; x < len; x++) {
14         jogo.arvoreQuadrante.procurarObjs(obj = [], objetos[x]);
15         for (y = 0, length = obj.length; y < length; y++) {
16             // Algoritmo de detecção de colisão
17             if (objetos[x].colidivelCom === obj[y].tipo &&
18                 (objetos[x].x < obj[y].x + obj[y].largura &&
19                  objetos[x].x + objetos[x].largura > obj[y].x &&
20                  objetos[x].y < obj[y].y + obj[y].altura &&
21                  objetos[x].y + objetos[x].altura > obj[y].y)) {
22                 objetos[x].isColidindo = true;
23                 obj[y].isColidindo = true;
24             }
25         }
26     }
27 };

```

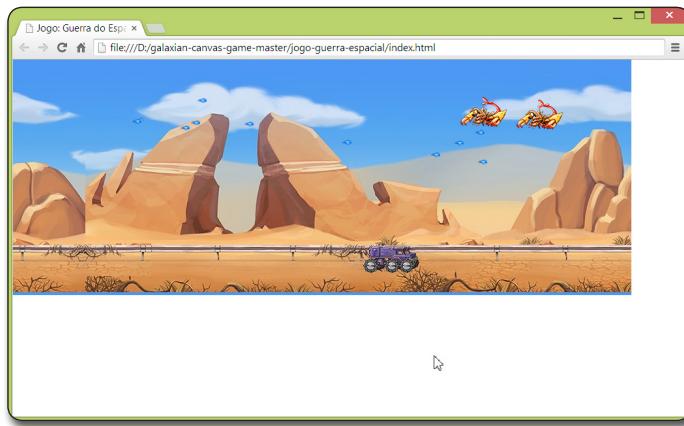


Figura 4. Resultado da execução do jogo com detecção de colisão

Perceba que as estruturas se equiparam, mudando apenas a quantidade, ordem e valores das imagens associadas. O resultado será semelhante ao da **Figura 5**.

Exibindo a pontuação

Para implementar um placar de pontuação, basta fazer algumas alterações simples. Como será impresso na HTML, precisamos modificar o conteúdo da index.html tal como na **Listagem 18**.

Listagem 17. Código de randomização dos objetos desenháveis.

```

01 var pfSrc;
02 switch (Math.floor(Math.random()*2)) {
03     case 0:
04         pfSrc = "imgs/pf.png";
05         break;
06     case 1:
07         pfSrc = "imgs/pf2.png";
08         break;
09 }
10 this.planofundo.src = pfSrc;
11 var playerSrc;
12 switch (Math.floor(Math.random()*2)) {
13     case 0:
14         inimSrc = "imgs/player1.png";
15         break;
16     case 1:
17         inimSrc = "imgs/player2.png";
18         break;
19 }
20 this.player.src = inimSrc;
21 var inimSrc;
22 switch (Math.floor(Math.random()*3)) {
23     case 0:
24         inimSrc = "imgs/enemy1.png";
25         break;
26     case 1:
27         inimSrc = "imgs/enemy2.png";
28         break;
29     case 2:
30         inimSrc = "imgs/enemy3.png";
31         break;
32 }
33 this.inimigo.src = inimSrc;

```

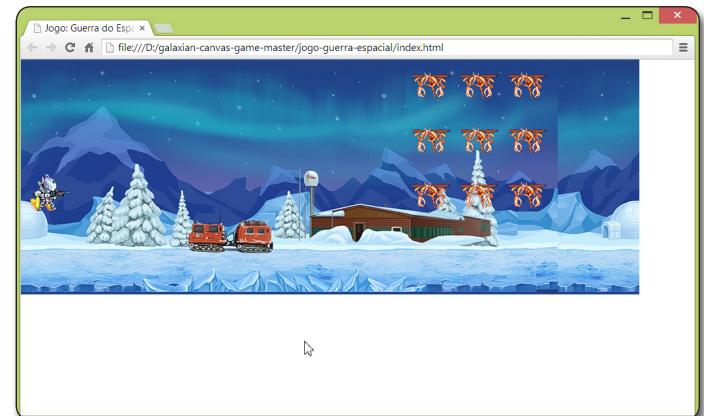


Figura 5. Resultado da randomização dos objetos desenháveis no jogo

A listagem exibe apenas a adição de uma div ao documento HTML para exibir de fato o valor atualizado da pontuação e um CSS para estilizar a mesma. Para dinamizar precisamos modificar algumas funções do arquivo de script, que estão definidas na **Listagem 19**.

A primeira alteração está na classe Jogo, adicionando a variável que irá contar a pontuação. Logo após, alteramos a classe Inimigo na hora de desenhar o código que irá de fato incrementar a pontuação assim que um inimigo for atingido. O mesmo incremento será feito de cinco em cinco.

Jogos em HTML5: Criando um jogo 2D com Canvas – Parte 2

Finalmente, na função externa de animação incutimos o código que irá referenciar a div HTML e modificar seu valor com a pontuação atualizada. O resultado deverá ser semelhante ao da Figura 6.

Listagem 18. Código HTML para placar de pontuação.

```
01 // Após o último canvas
02 <div class="pontuacao">PONTUAÇÃO: <span id="pontuacao"></span></div>
03
04 // No CSS
05 .pontuacao {
06   position: absolute;
07   top: 5px;
08   left: 480px;
09   color: white;
10  font: 15pt Segoe UI, sans-serif;
11  font-weight: bold;
12  cursor: default;
13}
```

Listagem 19. Código JavaScript para placar de pontuação.

```
01 // Na função iniciar() da classe Jogo
02 this.pontuacao = 0;
03
04 // No último else da função desenhar() da classe Inimigo
05 jogo.pontuacao += 5;
06
07 // Na função externa animar()
08 document.getElementById('pontuacao').innerHTML = jogo.pontuacao;
```



Figura 6. Inclusão do placar de pontuação ao jogo

Adicionando sons ao jogo

Para implementar os efeitos de sons às ações dos personagens faremos uso da API de áudio da HTML5. A vantagem de usá-la é que não precisamos nos preocupar com nenhuma implementação de terceiros, já vem tudo integrado no seu navegador. Entretanto, ela tem algumas desvantagens como o fato de não saber nem informar exatamente quando um áudio está 100% carregado e disponível. Em função disso, implementaremos uma função de intervalo que executa de tempos em tempos para verificar se o som está carregado e o carregar apropriadamente caso contrário.

Ao usar a HTML5 Audio API é importante se certificar de criar sempre um objeto Audio para cada som a ser tocado, isso porque eles funcionam de forma independente na árvore DOM da HTML.

O primeiro objeto que vamos criar é o pool de sons (**Listagem 20**). Isso para que quando um som terminar, nós automaticamente reutilizamos o objeto que acabou para tocar novamente, dando a ideia de faixa infinita.

Listagem 20. Código de criação do pool de sons.

```
01 function PoolSons(tamMax) {
02  var tamanho = tamMax; // Max de sons permitido no pool
03  var pool = [];
04  this.pool = pool;
05  var somAtual = 0;
06  /*
07   * Popula o array com o son dado
08   */
09  this.iniciar = function(objeto) {
10    if (objeto == "laser") {
11      for (var i = 0; i < tamanho; i++) {
12        // Inicializa o som
13        laser = new Audio("sons/laser.mp3");
14        laser.volume = .12;
15        laser.load();
16        pool[i] = laser;
17      }
18    }
19    else if (objeto == "explosao") {
20      for (var i = 0; i < tamanho; i++) {
21        var explosao = new Audio("sons/explosao.mp3");
22        explosao.volume = .1;
23        explosao.load();
24        pool[i] = explosao;
25      }
26    }
27  };
28  /*
29   * Toca o som
30   */
31  this.get = function() {
32    if(pool[somAtual].currentTime == 0 || pool[somAtual].ended) {
33      pool[somAtual].play();
34    }
35    somAtual = (somAtual + 1) % tamanho;
36  };
37}
```

Essa classe de pool é ligeiramente diferente das demais porque não precisamos dar push ou pop nos objetos. O elemento de áudio irá tocar até o fim e quando acabar nós setamos sua propriedade **ended** para true. Isso nos permite reexecutar o som somente se seu tempo de execução for 0 ou se ele tiver acabado. Na função **iniciar()** da linha 9 nós checamos se o objeto que recebemos é um som de laser (que será tocado quando dispararmos um combo) ou uma explosão (quando atingirmos um inimigo). Note que dentro de cada um dos testes nós definimos o volume do som através da propriedade **volume** (e você pode ajustar esse valor a bel prazer). Finalmente, a função **get()** toca o som após fazer a dita verificação.

Com tudo isso pronto, nós precisamos ainda adicionar os objetos de áudio físicos à classe Jogo, como mostra a **Listagem 21**. Adicione-os logo após a criação do objeto quadtree.

Essa listagem é bem simples, apenas cria cada um dos objetos Audio, define seus volumes, se tocarão em loop e chama a função

checkEstadoPronto() (**Listagem 22**) que irá se encarregar de implementar o intervalo de checagem que falamos anteriormente. Você irá encontrar os arquivos de som no pacote de download do fonte deste artigo.

Listagem 21. Código de criação dos objetos de sons.

```
01 this.laser = new PoolSons(10);
02 this.laser.iniciar("laser");
03 this.explosao = new PoolSons(20);
04 this.explosao.iniciar("explosao");
05 this.somFundo = new Audio("sons/mus_fundo.mp3");
06 this.somFundo.loop = true;
07 this.somFundo.volume = .25;
08 this.somFundo.load();
09 this.somGameOver = new Audio("sons/game_over.mp3");
10 this.somGameOver.loop = true;
11 this.somGameOver.volume = .25;
12 this.somGameOver.load();
13 this.checkAudio = window.setInterval(function(){checkEstadoPronto()},1000);
```

Listagem 22. Função de checagem do estado dos sons.

```
01 function checkEstadoPronto() {
02   if (jogo.somGameOver.readyState === 4 && jogo.somFundo.readyState ===
03     4) {
04     window.clearInterval(jogo.checkAudio);
05     jogo.jogar();
06   }
}
```

Essa função tem o objetivo de evitar que erros no DOM aconteçam ao checar o estado de nossos arquivos de som maiores: o de game over e o de plano de fundo. Dessa forma, só carregaremos o jogo se eles estiverem ok. Ao fazer isso, nós precisamos remover a função jogar() do método iniciar() externo. Para finalizar, façamos três alterações nas classes Inimigo, Player e Jogo (**Listagem 23**).

Listagem 23. Últimos ajustes para implementar os sons no jogo.

```
01 // No final do método atirar() da classe Player
02 jogo.laser.get();
03
04 // No else do método desenhar() da classe Inimigo
05 jogo.explosao.get();
06
07 // Na função jogar() da classe Jogo
08 this.somFundo.play();
```

A primeira linha força o som de laser a ser tocado quando o player atirar. A segunda faz o mesmo quando o inimigo explodir e a última servirá para tocar o som de fundo quando o jogo iniciar. Isso será o suficiente para ter os efeitos de sons implementados, agora recarregue a página do jogo e confira.

DÊ UM SALTO EM CONHECIMENTO!



Acesse o maior
portal para
desenvolvedores
da América
Latina!

20
mil
posts

430
mil
cadastrados

10
milhões de
page-views
por mês

Jogos em HTML5: Criando um jogo 2D com Canvas – Parte 2

Toques finais

Para finalizar o nosso jogo, vamos apenas implementar a adição de mais uma horda de inimigos quando a que estamos combatendo acabar. Para isso, efetue as mudanças descritas na **Listagem 24** à classe Jogo.

Listagem 24. Criando nova horda de inimigos.

```
01 // Na função iniciar()
02 this.novaHorda();
03
04 // Criar nova função na classe
05 this.novaHorda = function() {
06     var altura = repositorio.inimigo.height;
07     var largura = repositorio.inimigo.width;
08     var x = 600;
09     var y = -altura;
10    var espaco = 120 * 0.7;
11    for (var i = 1; i <= 9; i++) {
12        this.poollnimigos.get(x,y,1);
13        x += largura + 15;
14        if (i % 3 == 0) {
15            x = 600;
16            y += espaco
17        }
18    }
19}
20
21 // Na função animar()
22 if (jogo.poollnimigos.getPool().length === 0) {
23     jogo.novaHorda();
24}
```

Essa listagem não traz tantas novidades, exceto pela função **novaHorda()** que contém o mesmo código de quando criamos a horda inicial. Por último, na função **animar()**, quando é identificado que o pool de inimigos está vazio uma nova horda é criada e adicionada à tela.

Existem muitas outras coisas que o leitor pode acrescentar ao jogo com os conhecimentos adquiridos. Você pode criar uma tela de game over para quando o player for atingido, criar um botão para pausar o jogo, outro para deixá-lo mudo, criar níveis e aumentar a dificuldade à medida que eles forem subindo, e até mesmo chefões para o fim de cada um deles. Efeitos também são importantes então foque sempre nos códigos de animação que mostramos até então.

Além da HTML5 e Canvas, existem outras bibliotecas externas que você pode adicionar ao seu projeto para fornecer mais efeitos e recursos ao jogo, como o Enchant.js, Parallax, etc.

Autor



Júlio Sampaio

É analista de sistema e entusiasta da área de Tecnologia da Informação. Atualmente é consultor na empresa Visagio, trabalhando em projetos de desenvolvimento de sistemas estratégicos, é também instrutor JAVA. Possui conhecimentos e experiência em áreas como Engenharia de Software e Gerenciamento de Projetos, tem também interesse por tecnologias relacionadas ao front-end web.



Conhecimento faz diferença!

As capas das revistas mostram tópicos como:

- Edição 24 :: Aug 2012: Gerência de Configuração: Definição + Ferramentas.
- Edição 28 :: Ano 2: Agilidade: Negociação de contratos em projeto.
- Edição 29 :: Ano 3: Automação de Testes: Definições, preocupações e custo.
- Edição 29 :: Ano 3: Evolução do Software: Processo e levantamento de requisitos de negócios – Parte 2.
- Edição 29 :: Ano 3: Qualidade de Software: Definição, características e importância.

Na parte inferior, uma grande estrela vermelha com destaque para "Mais de 290 vídeos para assinantes".

Faça já sua assinatura digital! | www.devmedia.com.br/es

Faça um *upgrade* em sua carreira

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre outros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software. Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviços); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. **Assine Já!**



DEV MEDIA

Web Components na prática

Conheça o futuro promissor da web através da customização de componentes

AHTML5 nos trouxe inúmeros recursos novos como os Web Components. Os Web Components podem ser divididos em cinco categorias: Custom Elements, Templates, HTML Imports, Shadow DOM e Decorators. Essas tecnologias visam ajudar o desenvolvedor front-end na criação de novos componentes com o auxílio nativo do browser. Apesar dos componentes web ganharem cada vez mais notoriedade, muitas dúvidas e receios ainda costumam surgir entre os desenvolvedores, envolvendo assuntos como acessibilidade, semântica, SEO e performance no client-side. Explorando os assuntos do ponto de vista dos Web Components percebemos que podemos facilmente contornar tais receios utilizando algumas técnicas e tecnologias já consolidadas no desenvolvimento front-end. É possível utilizarmos microdados, WAI-ARIA, e as tags semânticas da HTML5 para aumentarmos a acessibilidade e semântica dos componentes. Buscadores como o Google aperfeiçoam constantemente o algoritmo do sistema de busca e recentemente engenheiros da empresa declararam no blog oficial já ser possível indexar o conteúdo gerado com JavaScript. Essa notícia diminui a preocupação de utilizarmos recursos como templates, uma vez que para tal é necessário a utilização de manipulação do conteúdo via JavaScript.

O único dificultador na utilização da tecnologia de fato é o suporte dos navegadores. Felizmente, os principais navegadores já iniciaram o processo de implementação dos Web Components. Enquanto aguardamos o suporte deles, é possível explorar os polyfills (código que pode ser baixado para prover facilidades que não estão implementadas nos browsers nativamente) que existem no mercado como o Polymer mantido pelo Google, o X-Tags pela Mozilla e o Bosonic.

Diante do grande furor gerado pelo lançamento dessas tecnologias, os mais entusiasmados consideram que os Web Components são o futuro da web. E de fato

Fique por dentro

Este artigo será útil para o entendimento sobre os Web Components, uma tecnologia nova da HTML5 que vem ganhando cada vez mais notoriedade e que visa auxiliar o desenvolvedor front-end na criação de componentes customizados. Além do entendimento sobre a tecnologia, o artigo também visa esclarecer alguns receios comuns da comunidade de desenvolvedores em torno dos Web Components que poderiam impactar negativamente nossos projetos ao utilizá-la. Os principais receios explorados no artigo são: acessibilidade, semântica, indexação e performance.

são, principalmente ao se considerar que a sua adesão ainda não aconteceu plenamente pela comunidade de desenvolvedores e sobretudo pelos principais navegadores do mercado.

Além do apoio que esses projetos fornecem à utilização dos Web Components, profissionais mais experientes e bastante conhecidos na comunidade de desenvolvimento web têm escrito artigos, realizado workshops e palestrado em eventos incentivando o estudo a fim de que os desenvolvedores conheçam e apoiem a nova tecnologia.

É bem verdade que muitas iniciativas de separar aplicações em parte menores, muitas vezes chamadas de componentes, deram origem a muitos projetos que foram adotados por muitos desenvolvedores web. Projetos como Twitter Bootstrap, Zurb Foundation, Pure.css, Semantic UI e tantos outros ganharam reconhecimento e continuam sendo utilizados em larga escala. A maioria possui uma vasta variedade de componentes, auxiliam no desenvolvimento responsivo, foram demasiadamente testados e são suportados pelos principais navegadores. Diante de projetos bem consolidados, muitos podem se perguntar qual o grande diferencial dos Web Components. Resumidamente, os Web Components são a promessa de padronizar e tornar nativo nos navegadores os recursos que até hoje têm sido uma solução criativa e alternativa por parte dos desenvolvedores que criaram substitutos a altura.

O que são os Web Components?

De acordo com a especificação da W3C, os Web Components consistem em “um conjunto de cinco tecnologias: Templates, Shadow DOM, Custom Elements, HTML Imports e Decorators”. Sendo que essa última, diferente das demais, ainda não possui uma especificação e tem sido bastante omitida pela comunidade, por esses motivos os exemplos disponíveis nesse artigo não farão menção aos decoradores. Para facilitar o entendimento do conjunto de tecnologias podemos defini-las da seguinte forma:

- **Templates:** Templates representam uma sub árvore no DOM que permanece inerte e é enxertada na árvore principal. O trecho de código contido no template serve para a criação de componentes

dinâmicos, possibilitando que o desenvolvedor instancie e manipule tal código sob demanda. Devido a inércia dos templates o navegador não realiza a requisição das imagens e não executa os scripts contidos nos mesmos até que sejam enviados para a página via JavaScript.

- **Shadow Dom:** O Shadow DOM realiza o isolamento de elementos do DOM em trechos independentes evitando que estilos CSS e comportamentos JavaScript dos elementos alterem as características uns dos outros.

- **Custom Elements:** O Custom Element permite que o desenvolvedor crie tags diferentes das convencionais. O intuito é encapsular as demais tags numa personalizada, reutilizável, manutenível e manuseável. Essa tecnologia auxilia bastante na diminuição dos

Listagem 1. Arquivo componente-loading.html portador do componente de carregamento (Loading...)

```
01 <html>
02   <head>
03     <script>
04       var NovoComponente = document.registerElement('loading-componente', {
05         prototype: Object.create(HTMLElement.prototype, {
06           createdCallback: {
07             value: function(){
08               var
09                 link = document.querySelector('link[rel=import]'),
10                 t = link.import.querySelector('#tpl-loading'),
11                 clone = document.importNode(t.content, true);
12                 element = this,
13                 text = this.getAttribute('text'),
14                 animates = clone.querySelectorAll('.load-anime div');
15                 clone.querySelector('.load-anime').classList.add("circle");
16                 clone.querySelector('.loading-text').innerHTML = text;
17                 element.createShadowRoot().appendChild(clone);
18             }
19           }
20         })
21       });
22     </script>
23   </head>
24   <body>
25     <template id="tpl-loading">
26       <style>
27         .loading {
28           height: auto;
29           margin: 10px auto;
30           text-align: center;
31           width: auto;
32         }
33         .loading p{
34           font-family:'Arial';
35           font-weight: bold;
36         }
37         .load-anime div{
38           background: #FF5700;
39           display: inline-block;
40         }
41         .circle{
42           height: 10px;
43         }
44         .circle div {
```

```
45           border-radius: 100%;
46           display: inline-block;
47           height: 10px;
48           width: 10px;
49           -webkit-animation: scale 1.2s infinite ease-in-out;
50           animation: scale 1.2s infinite ease-in-out;
51         }
52         .load-anime .load-anime2 {
53           -webkit-animation-delay: -1.1s;
54           animation-delay: -1.1s;
55         }
56         .load-anime .load-anime3 {
57           -webkit-animation-delay: -1.0s;
58           animation-delay: -1.0s;
59         }
60         .load-anime .load-anime4 {
61           -webkit-animation-delay: -0.9s;
62           animation-delay: -0.9s;
63         }
64         .load-anime .load-anime5 {
65           -webkit-animation-delay: -0.8s;
66           animation-delay: -0.8s;
67         }
68         @-webkit-keyframes scale {
69           0%, 40%, 100% { -webkit-transform: scale(0.4) }
70           20% { -webkit-transform: scale(1.0) }
71         }
72         @keyframes scale {
73           0%, 40%, 100% {
74             transform: scale(0.4);
75             -webkit-transform: scale(0.4);
76           }
77           20% {
78             transform: scale(1.0);
79             -webkit-transform: scale(1.0);
80           }
81       </style>
82     <div class="loading">
83       <div class="load-anime">
84         <div class="load-anime1"></div>
85         <div class="load-anime2"></div>
86         <div class="load-anime3"></div>
87         <div class="load-anime4"></div>
88         <div class="load-anime5"></div>
```

enormes aninhamentos de divs, conhecidos como *div hell*. Custom Elements também possibilitam que os elementos personalizados herdem características e propriedades de tags nativas específicas como a tag <button>, por exemplo.

• **HTML Import:** Com o HTML Import podemos importar documentos HTML inteiros para dentro de nossas páginas. De uma maneira semelhante à importação de um arquivo JavaScript ou CSS, o HTML Import cria uma nova requisição ao servidor para realizar o download do arquivo. Essa tecnologia auxilia no encapsulamento dos componentes em arquivos externos possibilitando importá-los posteriormente. Antes do HTML Imports, as poucas alternativas de importarmos documentos HTML eram através de elementos iframes e requisições AJAX, não muito bem vistas do ponto de vista do código limpo.

• **Decorators:** Os decoradores aplicam os templates com base em seletores CSS e JavaScript para criar mudanças visuais e comportamentais. O elemento “content” inserido dentro do template será substituído com o conteúdo do elemento de decoração. E através do atributo “select” é possível especificar a posição exata do elemento particular na marcação que será produzida. Com o CSS em conjunto com as Media Queries seria possível criar uma poderosa ferramenta de Responsive Web Design utilizando decoradores.

Veja nas **Listagens 1 e 2** um exemplo simples e de cunho didático onde podemos visualizar os quatro componentes web: Templates, Shadow DOM, Custom Elements e HTML Imports funcionando em conjunto.

Listagem 2. Arquivo index.html importa e faz uso do componente de carregamento

```
01 <!DOCTYPE html>
02 <html lang="pt-br">
03 <head>
04   <meta charset="utf-8">
05   <title>Web Components</title>
06   <meta name="description" content="Web Components">
07   <link rel="import" href="componente-loading.html">
08 </head>
09 <body>
10   <loading-componente text="Loading"></loading-componente>
11 </body>
12 </html>
```

No primeiro exemplo do nosso artigo criamos um novo elemento de carregamento animado, muito conhecido como “Loading...”. Antes das animações em CSS era muito comum a utilização de imagens GIF, arquivos em Flash e até mesmo JavaScript para a criação de elementos como esse. Agora com o uso da versão três do CSS para animar o elemento e com os componentes web podemos facilmente encapsulá-lo e torná-lo customizável de acordo com cada necessidade do desenvolvedor.

No arquivo *componente-loading.html* criamos o nosso componente propriamente dito. Entre as linhas 4 e 20 manipulamos o DOM para criarmos o componente baseado no template que fora definido entre as linhas 25 e 92. O template é bem simples, ele possui

regras de estilo e algumas “divs” que realizam a animação do elemento, além do parágrafo que receberá o texto do atributo “text” definido no elemento. Perceba também que para criá-lo fazemos uso direto da tag template do próprio framework.

O arquivo *index.html* realiza a importação do componente através da linha 7 e declara o componente de carregamento na linha 10. O componente criado possui um atributo chamado “text” que possui o texto que será exibido junto ao elemento renderizado, nesse caso a palavra “Loading”. O resultado da execução deste exemplo pode ser visualizado na **Figura 1**, assim como o código HTML gerado na **Listagem 3**.



Figura 1. Componentes de carregamento gerados com Web Components através do Google Chrome

Listagem 3. Código-fonte inspecionado através do Chrome DevTools.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="utf-8">
<title>Web Components</title>
<meta name="description" content="Web Components">
<link rel="import" href="loading-componente.html">
#document
</link>
</head>
<body cz-shortcut-listen="true">
<loading-componente text="Loading">
#shadow-root
<style>...</style>
<div class="loading">
<div class="load-anime circle">
<div class="load-anime1"></div>
<div class="load-anime2"></div>
<div class="load-anime3"></div>
<div class="load-anime4"></div>
<div class="load-anime5"></div>
</div>
<p class="loading-text">Loading</p>
</div>
</loading-componente>
</body>
</html>
```

Exemplos bem mais complexos e elaborados podem ser visualizados através do endereço customelements.io na seção **Links** do artigo, que funciona como um repositório de elementos criado com Web Components e compartilhado pela comunidade de desenvolvedores. A quantidade de elementos tem crescido a cada dia, mostrando que muitos desenvolvedores têm aderido à tecnologia. Também é importante destacar que muitos elementos foram criados baseados nos projetos que já existem, e possuem boilerplates (código clichê) que facilitam na implementação dos

novos elementos. Além dos boilerplates para os projetos citados também podemos usar um boilerplate que utiliza o VanillaJS (seção [Links](#)) para customizar nossos elementos.

Mesmo com tantos benefícios, os Web Components ainda têm preocupado os desenvolvedores mais cautelosos, os que gostam de analisar e ver as coisas acontecerem antes de se alvorocarem. principais receios giram em torno do usuário final e de aspectos como: acessibilidade, semântica, indexação do conteúdo por meio dos mecanismos de busca e performance no *client-side*.

As novas tecnologias que compõem o conjunto dos componentes da web possuem características bem peculiares, e que tem relação direta e indireta com tais aspectos. As novas tecnologias possibilitam importar arquivos HTML inteiros aumentando o número de requisições ao servidor, algo que implicaria na performance da nossa aplicação; encapsular parte do DOM em sub níveis mais profundos podendo dificultar o acesso ao conteúdo por parte de sintetizadores de voz ou robôs de busca, algo que afetaria a acessibilidade ou a indexação de conteúdo prejudicando o SEO; e a criação de novos elementos HTML de forma arbitrária podendo prejudicar a semântica do conteúdo.

Com a finalidade de nos aprofundarmos em tais receios e analisarmos se realmente oferecem algum impedimento para o futuro dos componentes web, vamos averiguar as principais áreas que podem afetar o usuário final. De uma forma macro e dentro das limitações atuais impostas pelos navegadores e pela tecnologia, realizamos uma série de pesquisas e testes que forneceram o embasamento para o artigo.

Acessibilidade

A acessibilidade é um fator importantíssimo do ponto de vista do usuário. Permitir que o mesmo acesse o conteúdo da melhor forma possível é um esforço geral que pode envolver inúmeros profissionais como arquitetos de informação, desenhistas, desenvolvedores e outros. A tecnologia, por sua vez, é o meio pelo qual tais profissionais disponibilizarão a informação para o usuário final.

A W3C considera quatro princípios como a base da web acessível: percepção, operabilidade, compreensão e robustez comumente conhecidos como POUR (*perceivable, operable, understandable, and robust*). Esses são os pilares que regem a web para todos. O objetivo desses princípios é permitir que todos tenham a chance de navegar na internet sem que sejam impedidos por suas limitações visuais, auditivas, cognitivas, motoras ou qualquer outra.

Em palestras sobre Web Components são bastante comuns indagações a respeito da acessibilidade. A preocupação faz sentido quando pensamos na forma de isolamento e customização que os Web Components nos oferece. As principais dúvidas são se o Shadow DOM, com seu forte encapsulamento, ou a criação de novos elementos poderiam afetar de alguma forma a navegação do usuário ou o acesso ao conteúdo.

Tomando como base a biblioteca Polymer, um dos mais famosos projetos relacionados ao assunto e que tem por finalidade tornar a criação de aplicações web mais fáceis e rápidas utilizando Web

Components, foram feitas algumas asserções para averiguar se há algum impacto na acessibilidade quando utilizamos essas tecnologias.

Uma bateria de testes simples que englobaram leitores de telas, navegação por teclado, inversão no esquema de cores, alteração na resolução da tela, uso de dispositivos móveis (smartphones e tablets), sistemas operacionais e navegadores diferentes foi realizada usando alguns dos principais componentes (menu dropdown, menu acordeon, slider, botões, checkboxes, tooltip e outros) do Polymer.

Utilizando leitores de tela como VoiceOver para OSX, NVDA no Windows e o plugin ChromeVox do Google Chrome foi possível perceber que não há muita diferença entre um componente construído de forma convencional com recursos nativos da HTML e JavaScript comparados com os Web Components. O único contraste encontrado durante o teste de navegação assistida ocorreu ao utilizarmos o plugin ChromeVox, uma vez que o mesmo não foi capaz de acessar e ler o conteúdo encapsulado pelo Shadow DOM.

A navegação por teclado não funciona tão bem quanto esperado, não por culpa da tecnologia dos Web Components, mas por falta de aprimoramento nos componentes testados. Alguns elementos não respondem bem à navegação utilizando a tecla "tab", algo que poderia ser facilmente resolvido com o atributo "tabindex" da própria linguagem HTML. Outro fator importante na navegação via teclado é saber a posição que nos encontramos na tela. Alguns dos elementos testados removeram a propriedade CSS outline dificultando a orientação do usuário na página. Além da navegação, é importante que ações dos componentes respondam às interações dos usuários provocadas via teclados, algo que não ocorre muito bem e pode ser melhorado via JavaScript.

Ao alterar a resolução da tela e esquema de cores nenhum problema fora do esperado ocorreu que impactasse na utilização da tecnologia. Os testes nos dispositivos, plataformas e navegadores foram bem razoáveis apesar de o suporte ainda ser bem precário. Entretanto, é muito importante ressaltar que esse tipo de teste só foi possível devido a utilização do Polymer, sem o mesmo não seria possível a realização destes.

Apesar do cenário de teste ter sido baseado em um polyfill e não utilizando a tecnologia final, ficou claro que para tornar um website ou aplicação web acessível dependemos muito mais das pessoas envolvidas na execução do projeto do que da tecnologia em si. Ao contrário do que possa parecer, os componentes web se utilizados de uma maneira inteligente podem ser grandes aliados para enriquecer a acessibilidade de nossas páginas.

Semântica

A versão cinco da HTML ampliou a sua coleção de tags da linguagem. Dentre as finalidades está a de enriquecer a semântica do conteúdo disponibilizado na internet. Muitos desenvolvedores têm usufruído de tags como: <article>, <section>, <nav>, <aside>, <header>, <footer> e tantas outras que nos auxiliam a estruturar e dar mais sentido às nossas páginas.

Além das novas tags semânticas, a HTML5 provê outros recursos para ampliar o significado. Os microdados, por exemplo, destinam-se a dar mais sentido ao conteúdo web para que possam ser lidos por máquinas, como mecanismo de busca, *web crawlers*, navegadores, leitores de tela, e outros, com a finalidade de enriquecer a experiência do usuário. A WAI-ARIA (*Web Accessibility Initiative - Accessible Rich Internet Applications*) é outro excelente recurso que possibilita dar sentido às interações que ocorrem na página transmitindo ao usuário o significado de suas ações.

Sabe-se que, com o advento dos Web Components, o desenvolvedor ganha a liberdade de criar suas próprias tags, podendo uni-las com a coleção de tags nativas da linguagem HTML. A única regra que o desenvolvedor deve atentar é que para criar as novas tags ele precisará respeitar o padrão *dash case*, ou seja, a criação das tags customizadas deverá contemplar o caractere “-” separando as palavras do novo elemento. Nesse caso “<meulemento>” é um nome válido enquanto “<meuElemento>” ou “<meuelemento>”, não. Além do hífen, o desenvolvedor deverá atentar para algumas combinações de palavras reservas como: annotation-xml, color-profile, font-face, font-face-format, font-face-name, font-face-src, font-face-uri e missing-glyph. Essas palavras podem ser encontradas nas especificações SVG e MathML, cujos links estão disponíveis na página oficial do W3C. Outra forma bem simples de verificar se o nome para o novo componente encontra-se disponível é utilizar a ferramenta online mothereff.in, disponibilizada pelo Mathias Bynens e que utiliza o módulo do Node.js `validate-element-name` criado por Sindre Sorhus.

A liberdade concedida pelos Web Components na criação dos nossos próprios elementos não deveria ser algo preocupante do ponto de vista semântico, pois, apesar de toda essa liberdade, é possível que o desenvolvedor mantenha a semântica do seu conteúdo utilizando os recursos citados: Microdados e WAI-ARIA.

O exemplo da **Listagem 4** demonstra a utilização de um componente de navegação por abas que foi enriquecido pelo uso de WAI-ARIA. Perceba que agora fazemos uso de tags antes nunca reconhecidas pela HTML, em quaisquer browsers que executassem. A estrutura dos itens de tabs aliada aos elementos roles (que lembram muito os do Bootstrap) complementam a implementação deixando o entendimento fácil até para quem nunca usou Web Components. Essa estrutura também é muito similar à de frameworks como o jQuery UI, que provê funções JavaScript para produzir o mesmo efeito que temos na listagem, porém exigindo a necessidade de importação de duas bibliotecas externas à aplicação.

Alguns desenvolvedores ainda não sabem, mas ao utilizarmos Custom Elements para a criação de novos elementos podemos estender elementos nativos herdando suas propriedades e semântica. Nesse caso para criarmos um botão baseado na sua versão nativa da HTML, por exemplo, basta definirmos seu protótipo como sendo uma instância do objeto HTMLButtonElement, tal como demonstrado na **Listagem 5**. Isso é possível graças ao uso da função `registerElement` presente no próprio objeto

DOM JavaScript, além disso, esse modelo pode ser usado para definir quaisquer outros elementos que o leitor desejar.

Listagem 4. Componente de painel com abas que utiliza WAI-ARIA

```
01 <!-- Menu em Abas -->
02 <tabs-menu role="tabmenu">
03   <tab-item href="#tab-1" role="tab" aria-selected="true">Custom Elements
04   </tab-item>
05   <tab-item href="#tab-2" role="tab">Templates</tab-item>
06   <tab-item href="#tab-3" role="tab">Shadow DOM</tab-item>
07   <tab-item href="#tab-3" role="tab">HTML Import</tab-item>
08 </tabs-menu>
09 <tab-panel id="tab-1" role="tabpanel" aria-labelledby="customelements">
10   Conteúdo sobre Custom Elements...
11 </tab-panel>
12 <tab-panel id="tab-2" role="tabpanel" aria-labelledby="template">
13   Conteúdo sobre templates...
14 </tab-panel>
15 <tab-panel id="tab-3" role="tabpanel" aria-labelledby="shadowdom">
16   Conteúdo sobre Shadow DOM
17 </tab-panel>
18 <tab-panel id="tab-4" role="tabpanel" aria-labelledby="htmlimport">
19   Conteúdo sobre HTML Import
20 </tab-panel>
```

Listagem 5. Estendendo o elemento nativo button.

```
01 var meuBotao = document.registerElement(meu-botao, {
02   prototype: Object.create(HTMLButtonElement.prototype),
03   extends: 'button'
04 });
```

Outro detalhe importante do ponto de vista semântico, é que além de fornecer sentido ao conteúdo de suas páginas, um bom desenvolvedor front-end deve se preocupar com a semântica do seu código a fim de que outros profissionais possam facilmente entender e dar continuidade ao projeto. Portanto, deve-se tomar cuidado ao escolher o nome do novo marcador evitando encurtar demais o nome dos elementos de forma que perca o significado. Evite simplificar demais o componente como no exemplo da **Listagem 6**, por exemplo. O conceito é o mesmo que já usamos para declaração de variáveis em qualquer linguagem de programação.

Listagem 6. Componente com nome demasiadamente simples.

```
01 <dd-m>
02   <dd-i>Criar</dd-i>
03   <dd-i>Editar</dd-i>
04   <dd-i>Excluir</dd-i>
05 </dd-m>
```

Procure utilizar palavras que auxiliem na identificação do componente e seus elementos. No exemplo da **Listagem 7**, uma vez entendido o que são “dropdowns”, fica fácil perceber como os seus elementos internos se alinham e dependem uns dos outros.

Listagem 7. Componente com nome mais verboso, contudo mais significativo.

```
01 <dropdown-menu>
02 <dropdown-item>Criar</dropdown-item>
03 <dropdown-item>Editar</dropdown-item>
04 <dropdown-item>Excluir</dropdown-item>
05 </dropdown-menu>
```

1. Indexação

Um especialista front-end é um conhecedor assíduo dos principais fatores *on-page* para otimização de sites para os mecanismos de busca. Ele sabe que, acima de tudo, o conteúdo é o principal responsável por atrair as visitas e consequentemente pelo posicionamento de um site nas primeiras posições das SERPs (*Search Engine Result Pages*). O conteúdo precisa ser original, bem escrito, relevante e principalmente estar facilmente acessível aos usuários e buscadores.

Diante disso, muitos desenvolvedores têm se perguntado como os Web Components se comportarão mediante a indexação do conteúdo de suas páginas através dos mecanismos de busca. Para criar novos componentes, manipular templates e encapsular seus elementos o desenvolvedor precisa utilizar JavaScript para manipular o DOM. E dúvidas como “O que aconteceria se um *crawler* como o Googlebot acessasse uma página recheada de Custom Elements e fortemente isolados com Shadow DOM?”, surgem entre muitos desenvolvedores curiosos.

Como se sabe, a maioria dos *web crawlers* navega de uma forma bem limitada, sem renderização de estilos CSS e execução de scripts, baseando-se apenas na estrutura HTML para realizar uma navegação totalmente textual.

O Google está constantemente aprimorando o seu algoritmo para que seus robôs ganhem mais inteligência no processo de indexação de conteúdo web. Recentemente, no blog da empresa, os engenheiros de software Erik Hendriks e Michael Xu e o Webmaster Analista de Tendências Kazushi Nagayama publicaram um artigo reconhecendo que o cenário atual da web mudou bastante desde o início da empresa. Os sites atuais são mais ricos, dinâmicos e utilizam bastante JavaScript. Diante disso, eles anunciaram que a ferramenta de busca já tem sido capaz de executar JavaScript com a finalidade de enxergarem as páginas como os navegadores modernos o fazem.

A informação é bem animadora, porém o artigo é um tanto vago a respeito de como o Googlebot realiza a execução do JavaScript. Os profissionais da empresa também deixam claro que em casos particulares a renderização da página pode não funcionar como o esperado podendo causar um impacto negativo no resultado da pesquisa. Diante disso, os autores e especialistas compartilham através do artigo dicas para evitar armadilhas que possam afetar o seu site. Dentre as dicas a que se destacou foi a ideia de construir páginas que degradem graciosamente, dessa forma o conteúdo estaria acessível à minúscula parcela de usuários que desabilitam o JavaScript de seus navegadores e para rastreadores que ainda não interpretam JavaScript.

Ainda é bem difícil determinar até que ponto a indexação de conteúdo poderá afetar o futuro dos Web Components. Na [página de FAQ](#) do projeto Polymer, a questão “*Crawlers understand custom elements? How does SEO work?*” (Crawlers entendem Custom Elements? Como o SEO funciona?) é respondida com bastante otimismo dizendo que “daqui para frente, é uma suposição razoável que, como o uso de espécies nativas do Shadow DOM aumentam, os fornecedores de motores de busca tentarão se adaptar e entendê-los, assim como eles se adaptaram a outras novas tecnologias web no passado.”

Obviamente, o cenário atual não está propício para a utilização dos Web Components na criação de sites tradicionais que precisam ter o seu conteúdo facilmente indexável e disponibilizado para os resultados de pesquisa. Projetos como o Polymer destacam que muitos desenvolvedores têm obtido sucesso ao utilizarem a biblioteca em suas aplicações que encontram-se em produção, mas que ainda é muito importante ter cautela. Devido a imaturidade da tecnologia, a mesma poderá sofrer muitas mudanças e a realização de um *test drive* é amplamente aconselhada.

No contexto atual da internet é muito difícil imaginarmos quais soluções estariam utilizando caso o JavaScript não existisse. A linguagem mudou completamente o panorama da internet e continua evoluindo muito nesses últimos dias. Graças ao JavaScript, as aplicações web se tornaram mais dinâmicas e com uma interface muito mais rica. O percentual de usuários que navegam com ele desabilitado é muito baixo. É bem provável que tão logo a indexação de conteúdo gerado por JavaScript esteja sanada, mitigando os riscos de usarmos Web Components.

2. Performance

Performance é um fator importantíssimo para qualquer site ou aplicação web. Disponibilizar o conteúdo para o usuário final em tempo hábil é sempre um desafio. Inúmeras pesquisas e estudos de caso comprovam que cada segundo é precioso, podendo impactar em conversões, engajamentos, resultados de busca, e tantos outros pontos.

Diante disso, muitas tecnologias, técnicas e dicas foram compartilhadas entre a comunidade de desenvolvedores. Essas técnicas foram aperfeiçoadas ao decorrer do tempo, e outras novas têm surgido para melhorar a experiência do usuário.

O desenvolvedor front-end se mostrou bastante criativo e um verdadeiro solucionador de problemas, aprendendo a tratar melhor as imagens, diminuir o tamanho dos seus arquivos, configurar o cacheamento dos arquivos estáticos, diminuir os refluxos de renderização das páginas, diminuir a quantidade de requisições para o servidor e muito mais.

No que tange à performance dos recursos dos Web Components de importação de HTML e Shadow DOM, o que temos na verdade são verdadeiras incógnitas. O que sabemos atualmente é que quando importamos documentos HTML geramos novas requisições ao servidor. Além das próprias requisições do documento importado, o mesmo pode conter outras importações e requisições de imagens, folhas de estilo ou arquivos de scripts caindo em um

verdadeiro *request hell*. No exemplo ilustrado na **Listagem 8** é possível ver isso, onde Web Components são importados de várias fontes diferentes, gerando várias consequentes requisições.

Listagem 8. Importações de múltiplos arquivos HTML.

```
01 <!DOCTYPE>
02 <html lang="pt-br">
03 <head>
04 <link rel="import" href="componente-1.html">
05 <link rel="import" href="componente-2.html">
06 <link rel="import" href="componente-3.html">
07 <link rel="import" href="componente-N.html">
08 </head>
09 <body>
10 </body>
11 </html>
```

Isso difere das soluções *server side* utilizadas para incorporar trechos de códigos HTML ao documento principal. Essas soluções inserem os *partials/includes* no arquivo que será entregue ao usuário final sem a necessidade de requisições extras.

Com a finalidade de diminuir o impacto na performance, o time responsável pelo Polymer desenvolveu uma ferramenta em NodeJS chamada **Vulcanize**. O Vulcanize concatena as referências dos arquivos a serem importados em um único arquivo evitando a necessidade de novas requisições. Para instalá-lo basta executar o comando `sudo npm install -g vulcanize` no npm do Node.js e concatenar os arquivos usando `vulcanize index.html`.

O HTTP 2.0 é outra tecnologia que tem sido aguardada com muita expectativa pela comunidade web. A nova versão do protocolo HTTP recebeu inúmeras melhorias visando justamente o desempenho, especificamente para o usuário final, percebido pela latência da rede e uso de recursos do servidor. Uma dessas melhorias é poder paralelizar as requisições com multiplexação, isso significa poder realizar várias requisições ao mesmo tempo e receber suas respectivas respostas conforme ficam prontas, de forma paralela e assíncrona. Utilizando esse recurso, apenas uma conexão basta evitando a necessidade de criarmos outros *hostnames* para superar a limitação do número de conexões por *hostname* dos navegadores.

Mas apesar de ainda não estar disponível, o HTTP 2.0 teve uma influência forte de outra tecnologia já disponível no mercado, o protocolo SPYD (pronuncia-se “SPeeDY”, ou “mais rápido”), implementado pelo Google e já suportado pelos navegadores mais modernos. Além do problema das requisições causadas pelas importações que podem ser contornadas concatenando os arquivos HTML com o Vulcanize, e que provavelmente será solucionada em breve com a utilização do HTTP 2.0, temos um outro dificultador: o isolamento do Shadow DOM.

O isolamento do Shadow DOM é realmente ótimo para evitar que estilos indesejados alterem as características de seus componentes, mas pode se tornar um verdadeiro incômodo quando queremos reaproveitar estilos comuns para nossos componentes nos levando à utilização de estilos incorporados.

A utilização de regras CSS incorporadas diretamente na página é algo considerado ruim tanto para performance quanto para modularização, e vem sendo evitada a bastante tempo por desenvolvedores que têm apreço por desempenho e reutilização de código. Estilos declarados fora do escopo do Shadow DOM não conseguem acessar os elementos devido ao seu isolamento. A solução é declarar regras de estilo direto no escopo do elemento isolado, ou seja, tag `<style></style>` direto no elemento. Veja um exemplo dessa injeção direta de JavaScript no exemplo da **Listagem 9**.

Listagem 9. Exemplo de utilização de estilos incorporados na declaração do Shadow DOM.

```
01 <div class="elemento"></div>
02 <script>
03 var root = document.querySelector('elemento').createShadowRoot();
04 root.innerHTML = '<style>h3{ color: red; }</style>' +
'<h3>Shadow DOM</h3>';
05 </script>
```

Novos seletores têm aparecido e nos permitirão estilizar o conteúdo isolado no Shadow DOM sem que tenhamos que declarar estilos incorporados toda vez que precisarmos estilizar nossos elementos. Os pseudo-elementos `::shadow` (**Listagem 10**) e o `/deep/` (**Listagem 11**) nos permitem penetrar a barreira do Shadow DOM para que possamos acessar os elementos dentro da Shadow Tree.

Listagem 10. Exemplo de utilização do seletor `::shadow`.

```
01 <!DOCTYPE html>
02 <html lang="en">
03 <head>
04 <meta charset="UTF-8">
05 <title>Shadow DOM</title>
06 <style>
07 #host::shadow p {
08   color: red;
09   font-weight: bold;
10 }
11 </style>
12 </head>
13 <body>
14 <div id="host">
15   <p>Conteúdo no DOM</p>
16 </div>
17
18 <script>
19 var host = document.querySelector('#host');
20 var root = host.createShadowRoot();
21 root.innerHTML = "<content></content>" +
"<p>Conteúdo no Shadow DOM</p>";
22 </script>
23 </body>
24 </html>
```

No exemplo, é possível ver que incutimos CSS específico (linhas 7 à 10) somente para o elemento de parágrafo criado dentro da função JavaScript da linha 21. Para recuperar a raiz da Shadow DOM basta usarmos a função `createShadowRoot()` da própria API HTML5.

Os seletores `::shadow` e `/deep/` são semelhantes, sendo que o segundo é bem mais poderoso. Ele ignora completamente as barreiras do Shadow DOM e permite acessar determinado elemento em qualquer nível da árvore que o mesmo se encontre. Podemos utilizar o `/deep/` para customizarmos o player da tag nativa `<video>`, por exemplo, como mostrado na [Listagem 11](#). Veja como a forma de declarar esse tipo de estilo difere um pouco do que estamos acostumados a usar em CSS, parecendo mais com uma espécie de comentário.

Criando seus próprios Web Components

Vamos mudar um pouco as coisas dentro do elemento `<template>` e ver o que acontece. Vejamos o exemplo definido na [Listagem 12](#).

Listagem 11. Utilização do seletor `/deep/` para aplicar estilos na tag nativa `video`.

```
01 <style>
02   video /deep/ input[type="range"] {
03     background: #ff0;
04   }
05   video /deep/ div {
06     color: #00ff00;
07   }
08 </style>
09 <video src="http://techslides.com/demos/sample-videos/small.ogv" controls>
10   Texto de fallback...
11 </video>
```

Listagem 12. Exemplo de criação de biblioteca própria.

```
01 <!-- Carrega a biblioteca do Google Polymer -->
02 <link rel="import" href="..//packages/sample/polymer/polymer.html">
03
04 <!-- Define um novo web component. -->
05 <polymer-element name="meu-componente" noscript>
06   <template>
07     Web components
08     <em><content></content>!</em>
09   </template>
10 </polymer-element>
11
12 <!-- Usa o novo web component. -->
13 <meu-componente>rocha</meu-componente>
14 <meu-componente>são fáceis de usar</meu-componente>
15 <meu-componente>me dê superpoderes web</meu-componente>
```

Enquanto o uso de componentes web apenas requer conhecimentos básicos de HTML, criação de componentes complexos requer o conhecimento de JavaScript, e está fora do escopo deste artigo. Se você é um programador JavaScript, dê uma olhada na documentação do projeto do Google Polymer para obter mais informações sobre como criar seus próprios componentes via JavaScript.

Se você não escreve muito JavaScript mesmo, há ainda uma gama de possibilidades interessantes para a criação de componentes básicos da web. A biblioteca Polymer permite que você crie componentes web simples somente com HTML. A demonstração que fizemos na última listagem usa o Polymer para criar um novo componente sem JavaScript:

- O exemplo define um novo componente chamado `<meu-componente>`.
- O componente inclui um elemento `<template>`. Qualquer coisa dentro do template vai se repetido cada vez que alguém usa o seu componente.
- O template inclui o texto de ações “Web components”.
- O modelo também inclui um elemento `<content>`. Este é mais um novo elemento HTML que funciona como um espaço reservado no template de um componente web. Quando alguém usa o seu componente, se colocar um texto ou outra coisa entre as tag `<meu-componente>` de abertura e a tag `</ meu-componente>` de fechamento, esse texto irá aparecer onde o espaço reservado para `<content>` aparece.

Este tipo de componente web é efetivamente uma “macro” HTML: um pedaço de HTML que você pode usar várias vezes sem constantemente ter que copiar e colar coisas. Há muito mais o que você pode fazer com um componente desse tipo, mas o ponto principal é que você pode criar componentes web simples, mesmo se você não sabe o JavaScript.

Apesar das preocupações que os Web Components podem gerar acerca do desempenho, vimos que inúmeras soluções têm surgido e que muito em breve evitarão os impactos negativos na performance de nossas aplicações web.

Em um mundo extremamente dinâmico como o da web é difícil falarmos sobre o futuro de uma determinada tecnologia. Muitas tecnologias desapareceram antes mesmo de se tornarem conhecidas, outras levaram bastante tempo para serem adotadas. Os Web Components ainda são uma tecnologia muito nova, mesmo assim são apontados como sucesso certo em um futuro próximo. Os receios citados em torno da tecnologia estão todos voltados para o momento da web que estamos vivenciando e certamente serão solucionados a partir do instante em que a adesão dos Web Components finalmente ocorrer.

Já em relação ao impeditivo do suporte dos navegadores, talvez deva ser considerado o principal e único receio quando pensamos em adotar os Web Components em nossos projetos. Felizmente, os principais browsers têm dado início à implementação dos Web Components. Os browsers baseados no motor de renderização Blink, Google Chrome e Opera, têm liderado essa corrida. Eles são os únicos até o momento que implementaram o conjunto de recursos: Templates, Shadow DOM, Custom Elements e HTML Imports de forma estável.

O Firefox, mesmo sendo mais vagaroso, tem feito grandes progressos e implementado várias partes do conjunto de recursos. A Mozilla mantém uma lista de bugs em seu repositório oficial, nos quais tem trabalhado para corrigir. Através da lista podemos acompanhar o progresso e votar nos bugs que consideramos mais importantes para correção. É uma ótima forma de contribuir com a evolução da Mozilla.

A Microsoft também tem feito planos para implementação dos componentes web no Internet Explorer. Através do endereço disponível na seção **Links** é possível acompanhar o status das

funcionalidades cogitadas para fazerem parte do navegador. Os quatro principais recursos ainda estão sob consideração. A versão atual do navegador não possui nenhuma das APIs para Web Components e é muito provável que tais implementações sejam disponibilizadas apenas na próxima versão do navegador que também contará com uma série de melhorias.

O Safari ainda permanece uma grande incógnita. Embora a recente versão do navegador já suporte o recurso de template, a funcionalidade de Shadow DOM foi recentemente removida do Webkit levantando dúvidas sobre o que de fato será implementado no navegador e quando.

Ao contrário do que possa parecer, o futuro dos Web Components depende antes de tudo da comunidade de desenvolvedores. Somos nós os principais responsáveis por utilizar e consequentemente manter viva qualquer tecnologia. É bem verdade que os Web Components e as tecnologias adjacentes ainda hão de amadurecer bastante, mas isso não impede que deixemos nossos receios de lado e possamos confiar mais uma vez na HTML5.

Agora cabe a você se inteirar melhor do assunto, fazendo mais testes e acrescentando funcionalidades aos exemplos desenvolvidos neste artigo. Tente baixar um dos polyfills da seção **Links**, e criar seus próprios Web Components de forma a entender como eles funcionam na prática e como serão abordados quando começarem a ser usados. Leia as documentações, as notas de atualização, e os blogs dos mesmos sites. Ademais, as principais dúvidas dos desenvolvedores podem sempre ser solucionadas/acompanhadas pelos fóruns oficiais de cada projeto e isso te ajudará quando estiver emperrado com alguma implementação específica. Bons estudos!

Autor



Gustavo Corrêa Alves

Desenvolvedor web há 8 anos, se especializou no desenvolvimento front-end: HTML, CSS, JavaScript, AngularJS, NodeJS (Express), SEO, Design Responsivo e Performance. Formado em Análise e Desenvolvimento de Sistemas pela Unicarioca, atualmente trabalha como Front-End Sênior no Hotel Urbano.



Links:

Especificação dos Web Components

<http://www.w3.org/TR/2013/WD-components-intro-20130606/>

WebComponents.org

<http://webcomponents.org/>

CustomElements.io

<http://customelements.io/>

Boilerplate com Vanilla

<https://github.com/webcomponents/element-boilerplate/>

Polymer

<https://www.polymer-project.org/>

X-Tags

<http://www.x-tags.org/>

Página de status do projeto no IE

<https://status.modern.ie/>

Somos tão apaixonados por tecnologia que o nome da empresa diz tudo.

Porta 80 é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia. Somos uma equipe composta de gente que entende e gosta do que faz, assim como você.



Estrutura

100% NACIONAL.
Servidores de primeira linha, links de alta capacidade.

Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.

Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.

1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única.
[Conheça!](#)



Porta 80

WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce |
Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486