



BONS ESTUDOS!

HTML5, JavaScript e CSS3

Este artigo apresentará alguns dos recursos trazidos pelas **tecnologias HTML5, JavaScript e CSS3**. Dentre os recursos abordados estão: campos de dados, [validação de formulário](#), armazenamento de dados com JavaScript, efeitos de texto e [bordas redondas com CSS3](#).

Este tema é útil a todos os desenvolvedores que desejam conhecer melhor as novidades das tecnologias client-side que vão revolucionar a maneira de desenvolver aplicações atraentes e interativas para a web.

Este artigo introduzirá os recursos das tecnologias [HTML](#), [JavaScript](#) e [CSS](#) sendo estudados numa abordagem prática. O objetivo das versões das tecnologias tratadas é facilitar a compreensão e manutenção do código, dar suporte a conteúdos que anteriormente só eram apresentados com plug-ins adicionais (vídeos, por exemplo) e auxiliar o desenvolvimento web baseado em padrões compatíveis com todos os tipos de dispositivos.

Através dos anos a internet passou por inúmeras mudanças. Os primeiros web sites que surgiram continham apenas conteúdo estático, ou seja, eram constituídos por páginas HTML expondo somente textos, imagens e links (sem muita interação com o usuário).

A necessidade de mais dinamismo nas páginas web culminou na **criação do JavaScript**, que chegou ao mercado em 1996. Com esta tecnologia foi possível mais interação nas páginas, uma vez que o **JavaScript possui recursos como manipulação de conteúdos, elementos e eventos HTML**.

Também em 1996 surgiu a primeira versão do CSS, uma linguagem de estilo utilizada para definir a apresentação de documentos HTML e XML. Seu principal benefício é separar a aparência do conteúdo de um documento. Assim, ao invés da formatação ficar dentro do documento HTML, neste haverá apenas uma ligação para o arquivo CSS, que contém os estilos. Deste modo, várias páginas podem utilizar o mesmo arquivo de formatação, e quando o programador quiser alterar a aparência do site, é necessário modificar apenas um arquivo.

Apesar dos vários recursos e **funcionalidades do JavaScript** e pelo CSS, ainda havia algumas deficiências. Por exemplo, devido às limitações das primeiras versões do HTML, não existia o suporte à apresentação de vídeos, e criar aplicações visualmente ricas ainda não era possível. Para suprir tais deficiências, comumente se faz uso de plug-ins como o Adobe Flash. Porém, com o intuito de não depender de plug-ins externos para tais tarefas, facilitar a manutenção e compreensão, e auxiliar o desenvolvimento web baseado em padrões compatíveis com todos os dispositivos, surgiu o **HTML5 e o CSS3**.

O HTML5 inclui novas tags que fornecem suporte a arquivos de vídeo e áudio, tipos de campos de dados, validação de formulários e outros recursos que veremos no decorrer do artigo. Também há novidades em sua **API JavaScript**, dentre elas, a possibilidade de salvar dados no computador do usuário de forma mais segura e rápida do que usando *cookies*. Já o CSS 3 possui uma série de recursos visuais para incrementar a aparência de suas aplicações.

Apesar do HTML5 e do CSS3 ainda estarem em desenvolvimento, as versões recentes dos navegadores mais populares já incorporam muitas das novas funcionalidades. Para testar o suporte de seu navegador, você pode acessar o site: <http://html5test.com>.

Com a finalidade de demonstrar um pouco dessas novidades, no decorrer deste artigo serão abordados **recursos do HTML5, JavaScript e CSS3**. Para isso, o conteúdo foi dividido quatro seções: a primeira seção trata de **HTML5 e JavaScript**, a segunda aborda CSS3, a terceira demonstra como configurar o ambiente de desenvolvimento para uma aplicação Java para web e, a última seção **apresenta o HTML5 integrado em um sistema web Java**.

As tecnologias client-side

Como este artigo é focado nas tecnologias utilizadas no desenvolvimento web do lado cliente (parte da aplicação que roda no navegador), é importante conhecer melhor o papel de cada uma para um melhor entendimento dos assuntos tratados nas próximas seções.

Essas tecnologias se dividem em:

- **Marcação:** trata-se da linguagem de marcação HTML, responsável por definir o que representa cada conteúdo em uma página e criar as ligações (links) para outros conteúdos;
- **Apresentação:** refere-se à **tecnologia CSS**, que é uma linguagem de estilo utilizada para definir a aparência de documentos escritos em HTML. Como já mencionado, seu principal benefício é prover a separação entre o formato visual e o conteúdo de um documento;
- **Comportamento:** trata-se do JavaScript, utilizado para trazer mais dinamismo às páginas web através de recursos como controle de eventos, [manipulação de elementos DOM](#) e seus conteúdos, entre outras funcionalidades.

Novidades do HTML5 e de sua API JavaScript

O HTML5 vem para revolucionar a maneira de se programar para web, com novos recursos que antes só eram disponíveis utilizando outras tecnologias. Sua finalidade é trazer suporte para os mais recentes tipos de multimídia e mantendo uma sintaxe legível, consistente e também bem compreendida por computadores e outros dispositivos.

Para testar cada exemplo apresentado nesta primeira seção, foi usado o navegador Google Chrome, que possui muitos dos novos recursos HTML já implementados.

Se você optar por não utilizar o Google Chrome para rodar os exemplos, ou utilizar navegadores em versões mais antigas, poderá não obter um correto funcionamento dos códigos apresentados.

Novas tags para estruturação de documentos

Novos elementos para estruturação do conteúdo da página foram **introduzidos no HTML5**. Estes elementos foram criados para representar semanticamente o conteúdo da página, ou seja, dividir melhor o conteúdo da página (cabeçalho, rodapé, menu principal) de acordo com sua finalidade e também facilitar a interpretação da página pelos mais variados tipos de navegadores e parsers. Observe um exemplo de uso dessas novas tags na **Listagem 1**.

Listagem 1. Código com novas tags HTML para estruturação.

```
<html ng-app>
<head>
  <script src="angular.js"></script>
  <script src="controllers.js"></script>
</head>
<body>
  <div ng-controller='HelloController'>
    <input ng-model='greeting.text'>
    <p>{{greeting.text}}, World</p>
  </div>
</body>
</html>
```

Estudando o código, notamos o uso do elemento `<header>` para definir um cabeçalho da página e o cabeçalho de uma seção. Este elemento pode ser usado, por exemplo, para agrupar índices de conteúdo, campos de busca e o cabeçalho do site com título ou logo.

O elemento `<section>` define cada seção de conteúdo, ou seja, cada divisão de conteúdo. Por exemplo, a página inicial de um site pode ser dividida entre os assuntos: destaques, novidades e últimas notícias. Desta forma, o ideal seria ter uma tag `<section>` para cada um desses temas, organizando o conteúdo do site por assunto. Como já mencionado, essas tags facilitam a interpretação do site por navegadores e parsers.

O `<article>`, por sua vez, trata-se de cada item de conteúdo dentro da página ou da seção, como posts, artigos e outros textos. Por exemplo, se estamos dentro de uma seção de notícias, cada notícia pode ser marcada pelo elemento `<article>`.

`<footer>` é o rodapé da página ou de uma seção e `<nav>` é o conjunto de links que formam a navegação, ou seja, o menu principal do site ou links relacionados ao conteúdo da página.

Já a tag `<aside>` é usada para dispor links e informações relacionados ao conteúdo principal, como a coluna lateral presente em blogs, por exemplo.

Hoje em dia, para definir as regiões da página onde ficam cabeçalhos, rodapés, menus e outros, é comum o uso da tag `<div>` para cada um deles, identificando-os por meio de um **id** específico para cada parte da página. Ao utilizar as novas tags, a página torna-se semanticamente melhor organizada. Um navegador, ao interpretá-la, saberá que o conteúdo presente na tag `<nav>` trata-se do menu principal do site. Ao passo que nos sites de hoje em dia, esse menu pode estar presente em qualquer tag.

Portanto, com os novos elementos para estruturação, o papel de cada conteúdo passa a ser mais bem

definido. Desta forma, essas novas tags também são uma solução para o acesso a partir de dispositivos móveis, cujo navegador pode interpretar cada parte da página de maneira a aproveitar melhor o espaço gráfico do aparelho.

Suporte a vídeos

O HTML5 inclui suporte a vídeos. Sendo assim, para apresentar este tipo de mídia nas páginas web, não é mais necessário o uso de plug-ins externos, que é a forma mais utilizada atualmente. A Listagem 2 demonstra um exemplo de uma página que exibe um vídeo.

Para rodar este exemplo é necessário que exista um vídeo de nome *movie* do tipo *mp4*, *webm* ou *ogv*, no mesmo diretório que o arquivo HTML. Exemplos de vídeos podem ser baixados nos endereços: <http://html5demos.com/assets/dizzy.mp4> ou <http://www.w3schools.com/html/movie.mp4>.

```
title>Exemplo com videos</title>
</head>
<body>
    <video width="320" height="240" controls="controls">
        <source src="movie.mp4" type="video/mp4" />
        <source src="movie.webm" type="video/webm" />
        <source src="movie.ogv" type="video/ogv" />
        Seu navegador não suporta a tag video
    </video>
</body>
</html>
```

Note que dentro da tag `<body>` foi criada a marcação `<video>` para dispor um arquivo de vídeo. Os atributos `width` e `height` são respectivamente a largura e a altura da região da página usada para apresentar a mídia. Já o atributo `controls` é utilizado para disponibilizar controles de reprodução.

Para finalizar, as várias tags `<source>` informam os diferentes formatos para o mesmo vídeo. Assim, se o navegador não suportar um formato, ele possui outras versões do mesmo vídeo para poder exibir. Já o texto contido na tag `<video>` somente será exibido caso o navegador não possua o suporte para esta tag. Veja na **Figura 1** o exemplo em funcionamento.

Gráficos com o elemento canvas

O canvas é um novo elemento introduzido pelo HTML5. Com ele, é possível, de maneira fácil, criar gráficos, composições de fotos e [animações usando JavaScript](#). Para tanto, basta definir a tag `<canvas>` dentro do documento HTML. Esta tag definirá a região utilizada pelo canvas e, depois, **usar JavaScript** para montar o desenho.

Para demonstrar a criação de elementos gráficos com este componente, execute o código apresentado na **Listagem 3**.

```

<html>
  <body>
    <canvas id="canvasExample" width="400px" height="200px">
      Seu navegador não suporta o elemento canvas.
    </canvas>
    <script type="text/javascript">
      var canvas=document.getElementById('canvasExample');
      var ctx=canvas.getContext('2d');
      ctx.fillStyle='rgba(255,0,0,0.5)';
      ctx.fillRect(20, 20, 100, 100);
      ctx.fillStyle='rgba(0,255,0,0.5)';
      ctx.fillRect(40, 40, 100, 100);
      ctx.fillStyle='rgba(0,0,255,0.5)';
      ctx.fillRect(60, 60, 100, 100);
      ctx.fillStyle='#000';
      ctx.fillText('Exemplo Usando Canvas', 15, 15);
    </script>
  </body>
</html>

```

Observe no código que na tag <canvas> foram definidas a largura (width) e a altura (height) da região onde o gráfico será desenhado. Logo abaixo, na tag <script>, há o código responsável por montar o desenho nesta região.

No início do código temos a busca pelo objeto canvas por meio método getElementById(). Na sequência, usou-se o método canvas.getContext() para acessar o objeto que contém os métodos para desenhar linhas, círculos, retângulos, entre outros.

Depois, nas configurações de estilo do canvas, é definida a cor vermelha através de seu código rgba com um nível de transparência de 50% (último argumento da função rgba). Essas configurações são setadas na propriedade ctx.fillStyle do canvas. Na linha seguinte, um quadrado de dimensões 100x100 e posição x=20 e y=20 é desenhado usando as configurações de cor anteriormente informadas.

Logo na sequência, uma nova configuração de cor é setada na propriedade ctx.fillStyle – verde com transparência de 50%. A seguir, um novo quadrado de mesma dimensão é desenhado, mas desta vez na posição x=40 e y=40, ou seja, um pouco mais abaixo e à direita do primeiro.

Depois, um terceiro quadrado, com a cor azul, é desenhado na posição x=60 e y=60. Na penúltima linha de código, utilizou-se a propriedade ctx.fillStyle para setar a cor preta (#000) no estilo de preenchimento e, então, foi usado o método fillText() para escrever um título na posição x=15 e y=15 da região do canvas. Observe na **Figura 2** o resultado deste script.

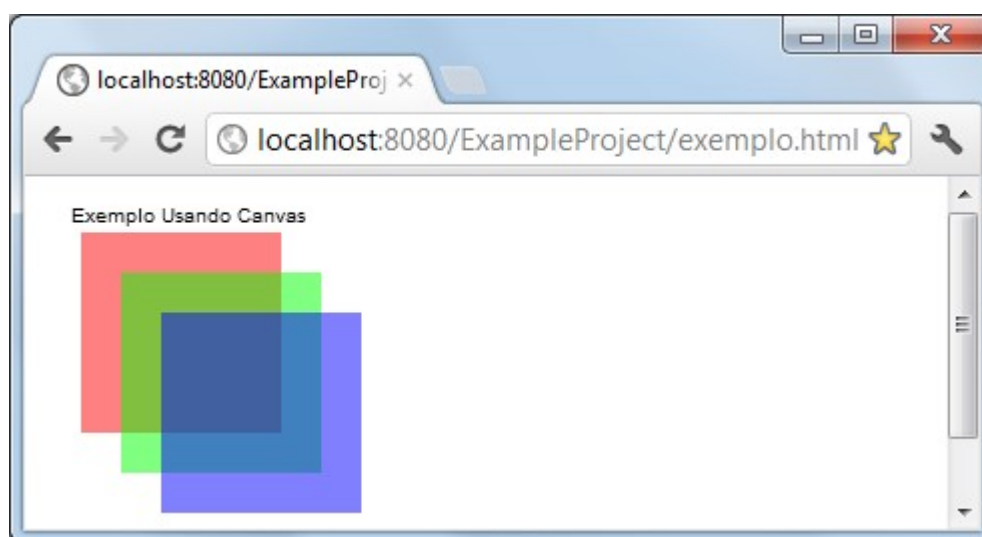
Observe no código que na tag <canvas> foram definidas a largura (width) e a altura (height) da região onde o gráfico será desenhado. Logo abaixo, na tag <script>, há o código responsável por montar o desenho nesta região.

No início do código temos a busca pelo objeto canvas por meio método `getElementById()`. Na sequência, usou-se o método `canvas.getContext()` para acessar o objeto que contém os métodos para desenhar linhas, círculos, retângulos, entre outros.

Depois, nas configurações de estilo do canvas, é definida a cor vermelha através de seu código `rgba` com um nível de transparência de 50% (último argumento da função `rgba`). Essas configurações são setadas na propriedade `ctx.fillStyle` do canvas. Na linha seguinte, um quadrado de dimensões 100x100 e posição `x=20` e `y=20` é desenhado usando as configurações de cor anteriormente informadas.

Logo na sequência, uma nova configuração de cor é setada na propriedade `ctx.fillStyle` – verde com transparência de 50%. A seguir, um novo quadrado de mesma dimensão é desenhado, mas desta vez na posição `x=40` e `y=40`, ou seja, um pouco mais abaixo e à direita do primeiro.

Depois, um terceiro quadrado, com a cor azul, é desenhado na posição `x=60` e `y=60`. Na penúltima linha de código, utilizou-se a propriedade `ctx.fillStyle` para setar a cor preta (`#000`) no estilo de preenchimento e, então, foi usado o método `fillText()` para escrever um título na posição `x=15` e `y=15` da região do canvas. Observe na **Figura 2** o resultado deste script.



Novo método de acesso a elementos

Antes, com JavaScript era possível acessar elementos HTML através do seu id, usando o método `document.getElementById()`, e pela tag do elemento, usando o método `document.getElementsByTagName()`. Com a nova API JavaScript também podemos acessar os componentes HTML a partir da propriedade `class` de cada um.

Esta nova funcionalidade traz mais facilidade para o desenvolvimento, pois em muitos momentos pode ser necessário manipular todos os elementos que possuem uma mesma `class` CSS. E utilizando este novo método, isto pode ser realizado de maneira rápida. Na **Listagem 4** temos um código para demonstrar o uso deste recurso.

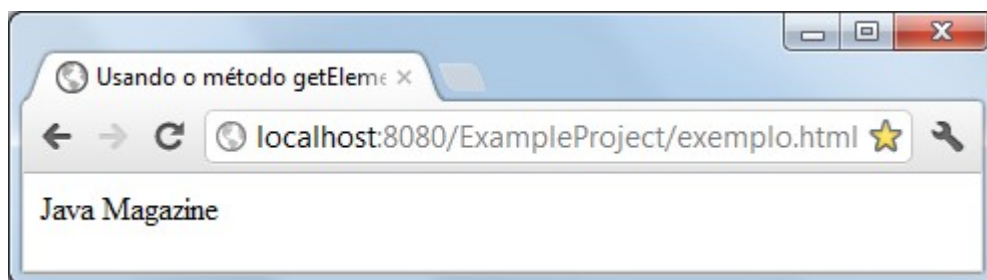
Listagem 4. Código de exemplo usando o método `getElementsByClassName()`.

```
<html>
  <head><title>Usando o método getElementsByClassName</title>
  <script>
    window.onload = function(){
      document.getElementsByClassName("test")[0].innerHTML = "Java Magazine";
    }
  </script>
```

```
</head>
<body>
    <p class="test"></p>
</body>
</html>
```

No código, definimos uma função JavaScript que foi atribuída ao evento `window.onload` e, desta forma, somente será chamada no momento em que a página for carregada.

Uma vez a página montada, o evento `window.onload` é disparado, executando a função. Nesta, temos a chamada do método `getElementsByClassName()`, que busca os elementos que possuem a class com o nome `test` passado como parâmetro e retorna um array com esses elementos. Como neste caso sabe-se que há apenas um elemento com class `test`, acessamos o elemento de índice 0 do array através do atributo `innerHTML`, inserindo o conteúdo “Java Magazine” no parágrafo `p` antes vazio. Veja a **Figura 3**.



Armazenamento de dados com JavaScript

Com o HTML5, as páginas web podem armazenar dados localmente no navegador do usuário. Anteriormente isto era feito através do [uso de cookies](#), porém, utilizando os novos objetos de armazenamento, essa tarefa é realizada de maneira mais rápida e segura, uma vez que utilizando cookies os dados são transitados a cada requisição efetuada. Para exemplificar esta funcionalidade, execute o código apresentado na **Listagem 5**.

Listagem 5. Código que salva dado localmente.

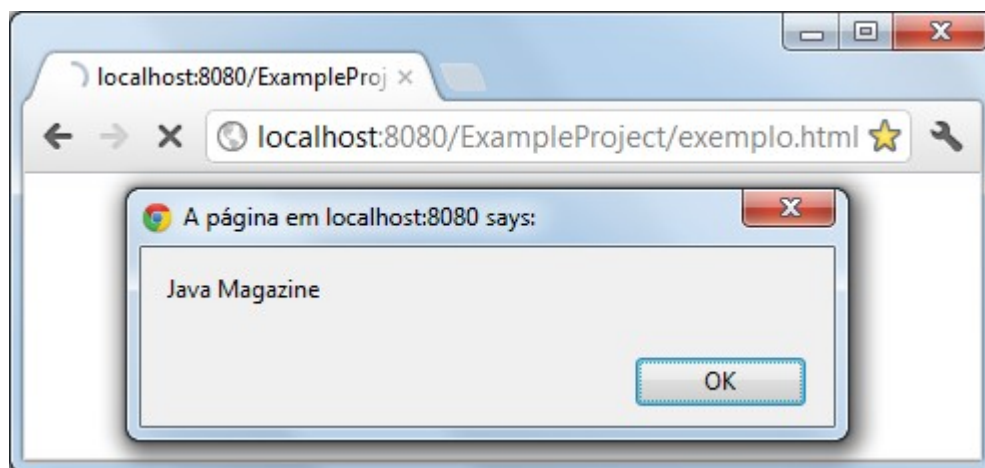
```
<html>
  <script>
    localStorage.mykey = "Java Magazine";
    alert("Dado armazenado.");
  </script>
</html>
```

Ao testar o código, o objeto `localStorage` é chamado para armazenar a String “Java Magazine” na chave de nome `mykey`, para posteriormente podermos acessar a informação contida na String utilizando-se do índice `mykey`. Os dados são sempre armazenados em pares (chave-valor). Neste caso, `mykey` foi o nome dado à chave que contém o valor “Java Magazine”.

Um trecho de código para acessar e exibir o conteúdo armazenado pelo exemplo anterior é mostrado na **Listagem 6**. Ao executá-lo, o dado é acessado por meio da sua propriedade `localStorage.mykey` e apresentado na tela pela função `alert()`. Veja o resultado na **Figura 4**.

Listagem 6. Código que recupera dado salvo localmente.

```
<html>
  <script>
    alert(localStorage.mykey);
  </script>
</html>
```



Este exemplo mostrou o uso do objeto `localStorage`, em que o dado, uma vez salvo, pode ser recuperado a qualquer momento. Mas há também o objeto `sessionStorage`, em que o dado fica armazenado apenas enquanto durar a sessão com o usuário.

Recursos do CSS 3

Criar páginas visualmente ricas ficou muito mais fácil com as novidades trazidas pelo CSS 3. A principal função do CSS 3 é agregar recursos visuais como transparência, transições e efeitos para criar animações de vários tipos. Por exemplo, é possível criar um relógio de ponteiros, usando somente recursos do CSS 3.

Bordas arredondadas com CSS

Para deixar a aparência das páginas mais agradável, é muito comum o uso de bordas redondas nas várias divs que usualmente dispõem o conteúdo do site. Atualmente, programadores utilizam de vários artifícios para obter esse efeito gráfico. Alguns utilizam imagens, outros utilizam scripts, mas com o CSS 3 isto não será mais necessário, já que foi incluída uma propriedade para criação de bordas arredondadas.

Saiba mais sobre CSS ;)

- [CSS3 Border: Trabalhando com bordas em CSS3](#):
Veja neste artigo as propriedades (`border-radius`, `box-shadow` e `border-image`) das CSS3

- para aplicar efeitos visuais às bordas dos elementos das páginas web.
- [Curso de CSS Gratuito](#):
Neste curso de CSS conheça como importar e incorporá-lo no HTML. Além disso você aprenderá a escrever seus primeiros códigos e conhecer as principais propriedades e atributos.
- [Curso de Pseudo classes do CSS](#):
Nesse curso aprenderemos a aplicar estilo baseado no estado do elemento com as pseudo classes do CSS. Veremos como aplicar o efeito de zebado em uma tabela, assim como selecionar o primeiro elemento em um parent.
- [Criando uma estrutura Front-end de navegação](#):
Neste curso criamos uma estrutura de um layout onde abordamos três características: a responsividade sem a utilização de frameworks como o bootstrap; um menu expandable; e também veremos uma barra lateral com scroll independente do conteúdo do site.

A **Listagem 7** apresenta o código HTML do exemplo. Estudando o código, podemos perceber na tag <body> a definição de uma <div> com id mydiv contendo um pequeno texto, e no início da página temos a importação do arquivo CSS, feita com a tag <link>.

Listagem 7. Código HTML do exemplo com a propriedade de bordas arredondadas.

```
<html>
  <head>
    <title>Exemplo com Bordas Arredondadas</title>
    <link rel="stylesheet" type="text/css" href="style.css" />
  </head>
  <body>
    <div id="mydiv">
      Exemplo de div com bordas arredondadas.
    </div>
  </body>
</html>
```

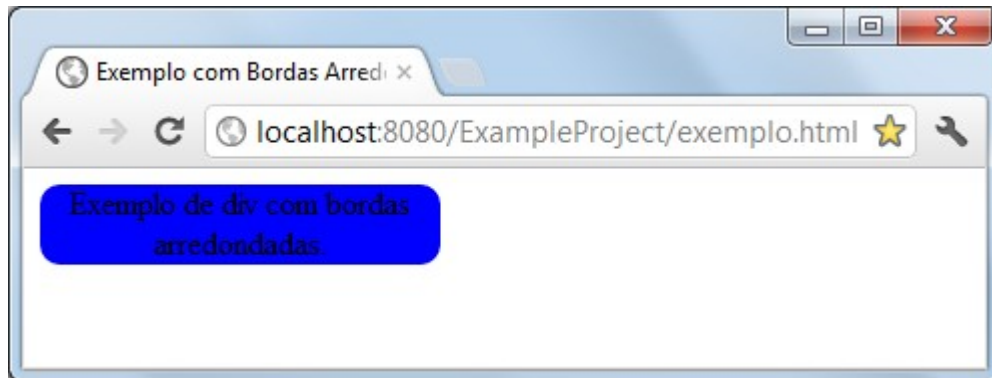
A **Listagem 8** exhibe o código CSS deste exemplo. Salve este código em um arquivo com o nome style.css e coloque-o na mesma pasta que o arquivo criado anteriormente. Veja no código CSS que algumas propriedades são configuradas para mydiv. Entre essas configurações, setamos que o texto presente neste elemento terá um alinhamento centralizado (text-align: center), aplicamos um fundo azul (background: blue) para facilitar a visualização das bordas, configuramos uma largura de 200 pixels (width: 200px) e, por fim, com a propriedade border-radius, definimos que o raio da borda arredondada será de 10 pixels.

A **Figura 5** ilustra este exemplo em funcionamento.

Listagem 8. Código CSS do exemplo com a propriedade de bordas arredondadas.

```
#mydiv {
  text-align: center;
  background: blue;
  width: 200px;
```

```
} border-radius: 10px;
```



Também é possível arredondar a borda de cada canto separadamente, através das seguintes propriedades CSS 3:

- `border-radius-topleft` – para o canto superior esquerdo;
- `border-radius-topright` – para o canto superior direito;
- `border-radius-bottomright` – para o canto inferior direito;
- `border-radius-bottomleft` – para o canto inferior esquerdo.

Ou ainda podemos passar todos os valores numa única declaração, assim como outras propriedades CSS permitem:

```
border-radius: 10px 20px 30px 40px;
```

Efeito de sombra de caixa

Outro efeito gráfico que pode ajudar a enriquecer o visual de uma aplicação web é o efeito de sombra de caixa. Este efeito exibe uma sombra para o elemento ao qual foi adicionado, como mostra a **Figura 6**. Para demonstrar seu funcionamento, este recurso foi aplicado em um elemento div. O código apresentado na **Listagem 9** pode ser usado para testar esta funcionalidade.



Observando o código, veja que na tag <body> da página é criada uma <div> e na tag <style> definimos a propriedade box-shadow, responsável por aplicar o efeito de sombra. Os argumentos passados à propriedade são respectivamente: a distância horizontal da sombra, a distância vertical, o nível de desfocagem e a cor da sombra.

Listagem 9. Código da página usando a propriedade box-shadow.

```
<html>
  <head>
    <style type="text/css">
      div {
        width: 300px;
        height: 100px;
        background-color: yellow;
        box-shadow: 10px 10px 5px #888888;
      }
    </style>
  </head>
  <body>
    <div></div>
  </body>
</html>
```

Efeito de sombra de texto

O CSS 3 também tem algumas novidades com relação a efeitos em textos, suprimindo necessidades há muito tempo conhecidas por programadores web que, comumente, fazem uso de imagens e até mesmo plug-ins externos quando é necessário a criação de textos com um visual mais elaborado.

Com a finalidade de apresentar um dos novos recursos, na **Listagem 10** temos o código para a demonstração do efeito de sombra de texto. Ao executar o exemplo, vamos obter o resultado demonstrado pela **Figura 7**.

Listagem 10. Código do exemplo com efeito de sombra de texto.

```
<html>
  <head>
    <title>Efeitos de texto com CSS 3</title>
    <style type="text/css">
      h1 {
        text-shadow: 5px 5px 5px #FF0000;
      }
    </style>
  </head>
  <body>
    <h1></h1>
  </body>
</html>
```

```

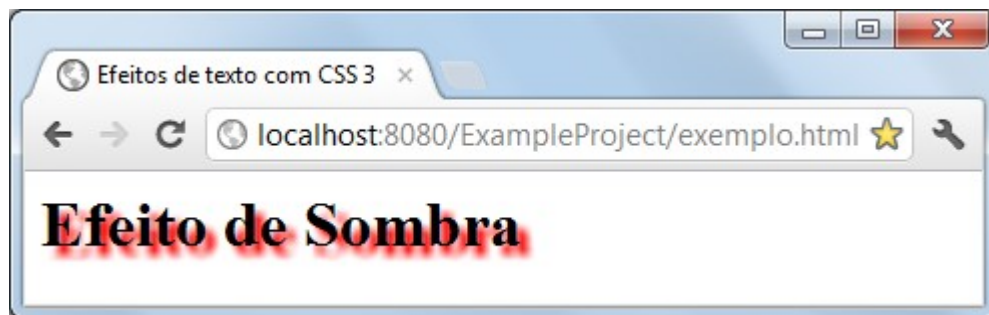
}
</style>
</head>
<body>
    <h1>Efeito de Sombra</h1>
</body>
</html>

```

No código, foi criado um título (elemento `<h1>`) dentro do `<body>` da página, e no início da mesma foi definida a tag `<style>` na qual atribuímos a propriedade CSS `text-shadow` para o elemento `<h1>` citado anteriormente.

`Text-shadow` é a propriedade responsável por aplicar o efeito de sombra de texto. Neste caso, utilizamos a propriedade para aplicar este efeito ao título. Veja que é possível definir uma cor para a sombra e também a sua distância em relação ao texto.

Observando o código CSS, note que os argumentos passados depois da propriedade são exatamente os mesmos do exemplo anterior.



Quebra de linha para palavras grandes em espaços curtos

Em uma página web, às vezes, uma palavra pode ser muito grande para uma determinada área e, em consequência, se expandir para fora dos limites do elemento HTML ao qual está inserida. Uma situação que muitas vezes presenciamos ao utilizar a internet no nosso dia-a-dia. Para entendermos melhor esse problema, execute o código da **Listagem 11**.

Listagem 11. Código com palavra muito longa.

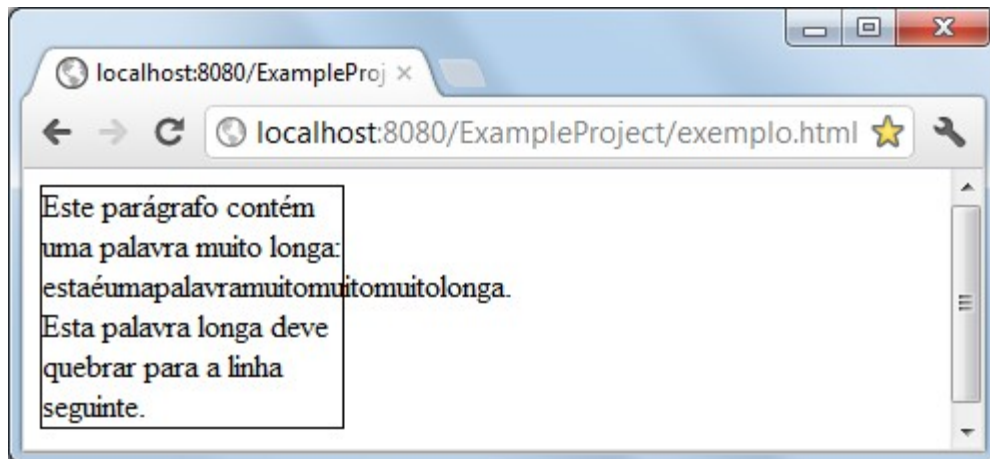
```

<html>
  <head>
    <style type="text/css">
      #test {
        width: 150px;
        border: 1px solid #000000;
      }
    </style>
  </head>
  <body>
    <p id="test"> Este parágrafo contém uma palavra muito longa:

```

```
estaéumapalavramuitomuitomuitolonga. Esta palavra longa  
deve quebrar para a linha seguinte.</p>  
</body>  
</html>
```

Ao abrir a página no navegador (veja a **Figura 8**), note que uma palavra ultrapassou os limites do elemento `<p>`. Isto porque a palavra toda ocupa uma extensão maior do que a largura do componente, que neste caso é de 150px.

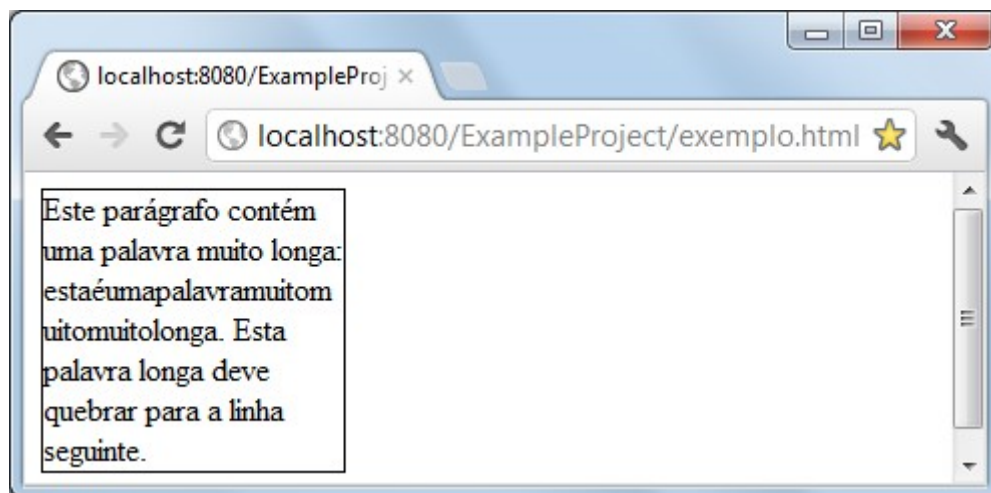


Para evitar que este tipo de situação indesejada aconteça, o CSS 3 traz a propriedade `word-wrap`. Com o seu uso podemos garantir que todo o texto esteja envolvido pelo elemento HTML, mesmo que, às vezes, isso signifique que a palavra seja quebrada ao meio. Para vermos o uso desta propriedade, substitua o conteúdo da tag `<style>` pelo código da **Listagem 12**.

Listagem 12. Código usando propriedade `word-wrap`.

```
<style type="text/css">  
  #test {  
    width: 150px;  
    border: 1px solid #000000;  
    word-wrap: break-word;  
  }  
</style>
```

Ao executar novamente a página, vamos ter o resultado mostrado na **Figura 9**. Agora sem o problema de a palavra exceder os limites do componente, pois ao utilizar a propriedade `word-wrap` recebendo `break-word` como argumento, se a palavra for longa de tal forma a ultrapassar a região da tag, esta palavra é quebrada (`break-word`), continuando na próxima linha.



Alterando tamanho, forma e posição de elementos HTML

Um novo recurso também foi adicionado com o objetivo de permitir alterar a posição, forma e tamanho de elementos HTML. Essas mudanças são feitas através do atributo CSS `transform`, que além de permitir a criação de aplicações com um visual mais rico, pode ser usado em conjunto com scripts para a criação de efeitos animados.

No Google Chrome, navegador que está sendo usado para as demonstrações deste artigo, para o uso da propriedade `transform` é necessário acrescentar o prefixo `-webkit-`.

Com o objetivo de ilustrar o uso dessa funcionalidade, vamos executar um exemplo que rotaciona a posição de uma `div`. O código é apresentado na **Listagem 13**.

Listagem 13. Exemplo rotacionando uma `div`.

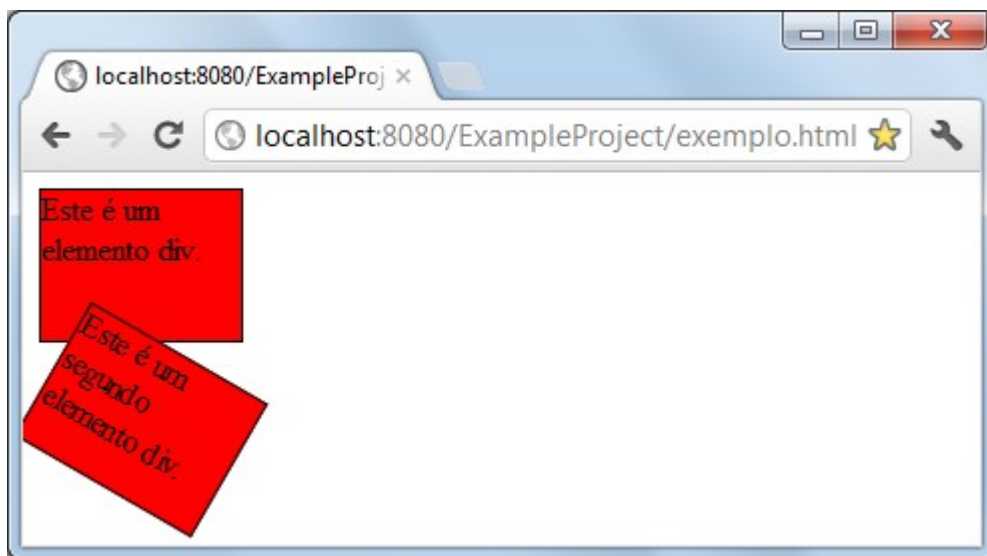
```
<html>
  <head>
    <style type="text/css">
      div
      {
        width:100px;
```

```

height:75px;
background-color:red;
border:1px solid black;
}
#div2
{
    -webkit-transform:rotate(30deg);
}
</style>
</head>
<body>
    <div>Este é um elemento div.</div>
    <div id="div2">Este é um segundo elemento div.</div>
</body>
</html>

```

Observe no código que duas divs foram criadas no corpo da página. Na tag `<style>` foram definidas configurações CSS gerais para estes dois elementos. No entanto, somente para a div2 temos a definição da propriedade `-webkit-transform` recebendo como parâmetro `rotate(30deg)`, que indica que a div2 deverá ser rotacionada em 30 graus. O resultado desta configuração é exibido na **Figura 10**.



O atributo transform possui muitos outros recursos que não serão abordados, como por exemplo: `scale` – responsável por redimensionar o elemento; `translate` – responsável por alterar a posição; `skew` – que faz uma transformação alterando os ângulos do elemento; e `matrix` – que altera a posição, ângulos e rotaciona o elemento (usando uma mesma propriedade). Caso tenha se interessado pelo assunto, visite o tutorial sobre CSS 3 que se encontra na seção **Links**.

Efeitos de transição

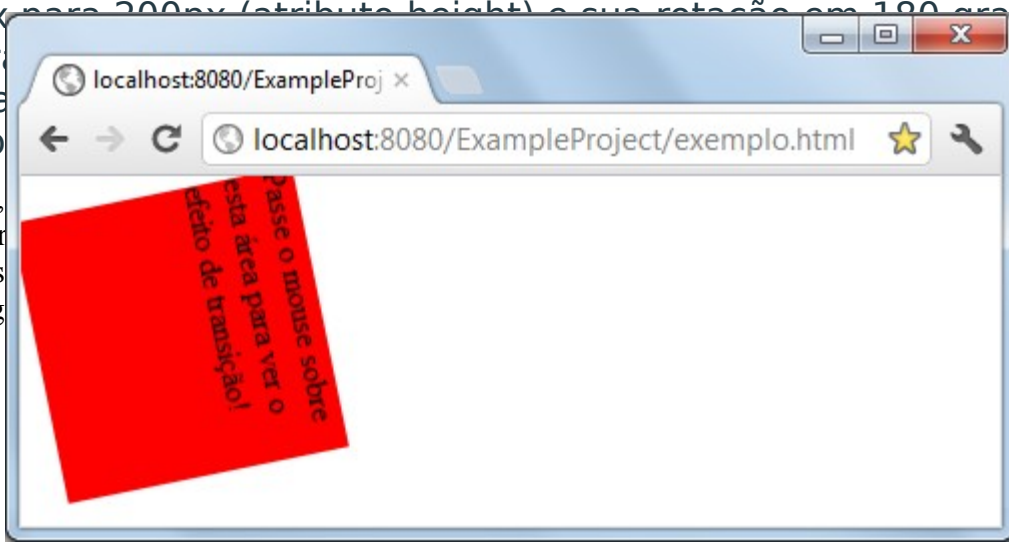
A última funcionalidade CSS 3 que será demonstrada são os efeitos de transição (transition). Esta propriedade permite alterar gradualmente valores de outros atributos CSS. Assim como no exemplo anterior, é necessária a inclusão do prefixo -webkit- para que este recurso funcione no Google Chrome. Na **Listagem 14** encontra-se o código do exemplo.

Listagem 14. Código do exemplo com efeito de transição.

```
<html>
  <head>
    <style type="text/css">
      div {
        width: 100px;
        height: 100px;
        background: red;
        -webkit-transition: width 2s, height 2s, -webkit-transform 2s;
      }
      div:hover {
        width: 200px;
        height: 200px;
        -webkit-transform: rotate(180deg);
      }
    </style>
  </head>
  <body>
    <div>Passe o mouse sobre esta área para ver o efeito de
transição!</div>
  </body>
</html>
```

Na tag <style> do exemplo especificamos através do div:hover que, ao ser sobreposta pelo cursor, a div irá alterar sua largura de 100px para 200px (atributo width), sua altura de 100px para 200px (atributo height) e sua rotação em 180 graus (-webkit-transform: rotate(180deg)).

Desta forma, segundos após o mouse passar sobre esta área para ver o efeito de transição!



dois
as
ra

Agora que já estudamos alguns recursos importantes do CSS 3, HTML5 e JavaScript, chegou o momento de utilizarmos esses conhecimentos em uma [aplicação Java para web](#).

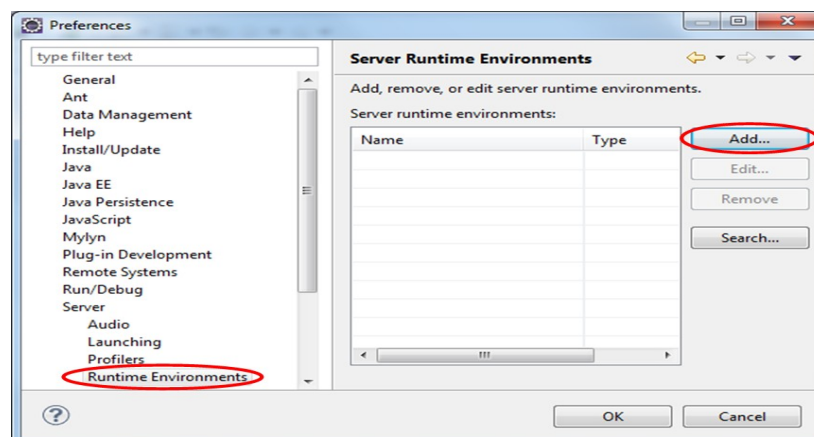
Preparando o ambiente de trabalho

Para o desenvolvimento do aplicativo que será apresentado na seção final deste artigo, precisamos antes montar o ambiente de trabalho. Deste modo, esta seção irá demonstrar os passos para a configuração deste ambiente.

Dentre as ferramentas que vamos utilizar, estão a [IDE Eclipse](#) Indigo para [Java EE](#), o JRE (*Java Runtime Environment*) e o Apache Tomcat versão 7.0.26. Os endereços para baixar estas ferramentas podem ser encontrados na seção **Links**.

Configurando o Tomcat

Para iniciarmos o projeto exemplo, devemos primeiramente configurar o [Tomcat](#) para rodar nosso aplicativo. Para isso, no Eclipse, selecione *Window > Preferences*. Na tela de preferências, ilustrada pela **Figura 12**, clique em *Server > Runtime Environments*, e então, no botão *Add*, localizado no canto direito superior da tela.



Na nova janela, selecione a versão 7 do Tomcat e clique em *Next*. Feito isso, na próxima tela clique em *Browse* para informar o diretório onde o mesmo foi instalado e depois em *Finish*. Terminados esses passos, o servidor estará configurado no Eclipse. Agora podemos criar nosso projeto.

Criando o projeto para desenvolver os exemplos

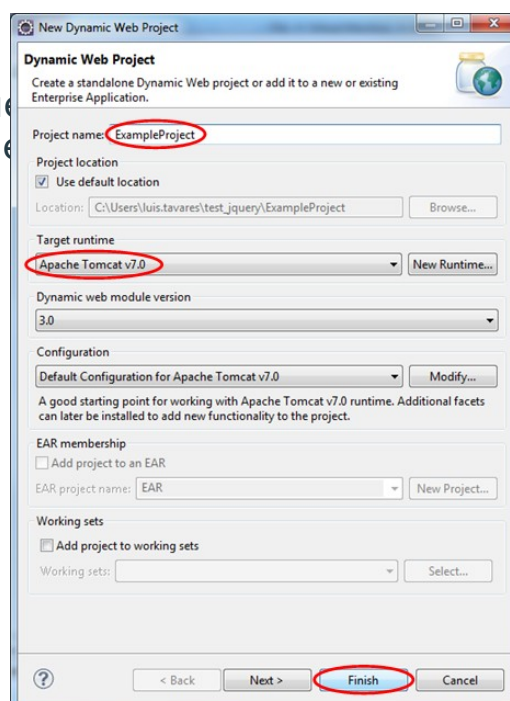
No Eclipse, crie o projeto onde iremos implementar o exemplo. Para isso, dê um clique com o botão direito dentro da visão *Project Explorer*, acesse a opção *New* e depois clique em *Dynamic Web Project*, conforme mostra a **Figura 13**.

Na nova janela, selecione a versão 7 do Tomcat e clique em *Next*. Feito isso, na próxima tela clique em *Browse* para informar o diretório onde o mesmo foi instalado e depois em *Finish*. Terminados esses passos, o servidor estará configurado no Eclipse. Agora podemos criar nosso projeto.

Criando o projeto para desenvolver os exemplos

No Eclipse, crie o projeto onde iremos implementar o exemplo. Para isso, dê um clique com o botão direito dentro da visão *Project Explorer*, acesse a opção *New* e depois clique em *Dynamic Web Project*, conforme mostra a **Figura 13**.

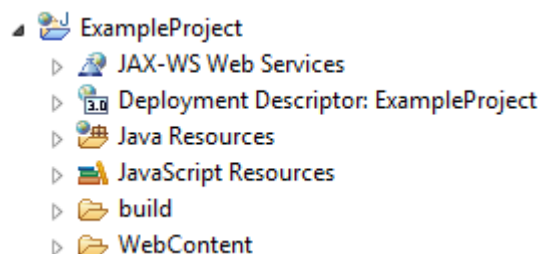
Você pode dar qualquer nome para o projeto, desde que o *Target runtime* seja configurado de acordo com a **Figura 13**.



é importante que o *Target runtime* seja configurado de acordo com a **Figura 13**.

Após criar o projeto, devemos obter a estrutura apresentada na **Figura 15**. Com o ambiente de trabalho pronto, podemos codificar os arquivos necessários e desenvolver o aplicativo exemplo.

Figura 15. Estrutura do projeto para criação da aplicação Java.



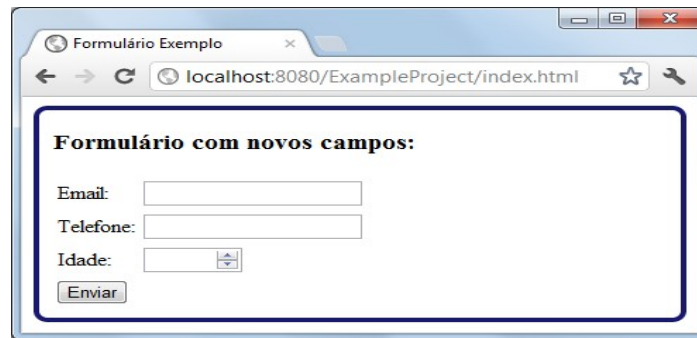
Utilizando HTML5 em uma aplicação Java

Para por em prática o conteúdo até aqui apresentado e ainda conhecer alguns novos tipos de campos de dados para formulários, iremos implementar uma aplicação web que vai simular a realização de um cadastro simples. O objetivo deste exemplo é apresentar a integração de alguns dos novos elementos do HTML5 em um sistema web Java e também demonstrar como desenvolver uma

validação de um <form> HTML sem a necessidade de nenhum script adicional para este fim.

A aplicação trata-se de um formulário web com três campos distintos: e-mail, telefone e idade. No momento da submissão do formulário, será realizada a validação dos campos de acordo com os atributos que especificamos na criação do form. Uma vez que a validação tenha sido realizada com sucesso, os dados serão submetidos para a parte servidor da aplicação, representada por um [Servlet](#). Caso ocorram erros durante a validação, mensagens serão exibidas apresentando estes erros.

A **Figura 16** demonstra o formulário em execução.



Desenvolvendo e entendendo o exemplo

Para desenvolver o nosso exemplo, localize o projeto criado na seção anterior e dentro da pasta WebContent crie a página index.html. Nesta página, adicione o código apresentado na **Listagem 15**.

Listagem 15. Código da página do formulário.

```
<html>
  <head>
    <title>Formulário Exemplo</title>
    <link rel="stylesheet" type="text/css" href="style.css" />
  </head>
  <body>
    <div>
      <form method="post" action="MyServlet">
        <h3>Formulário com novos campos:</h3>
        <table>
          <tr>
            <td><label>Email: </label></td>
            <td><input type="email" name="email" required/></td>
          </tr>
          <tr>
            <td><label>Telefone: </label></td>
            <td><input type="tel" pattern="\(\d\d\) \d\d\d\d-\d\d\d\d"
              title="(xx) xxxx-xxxx" name="phone" required/></td>
          </tr>
          <tr>
```

```

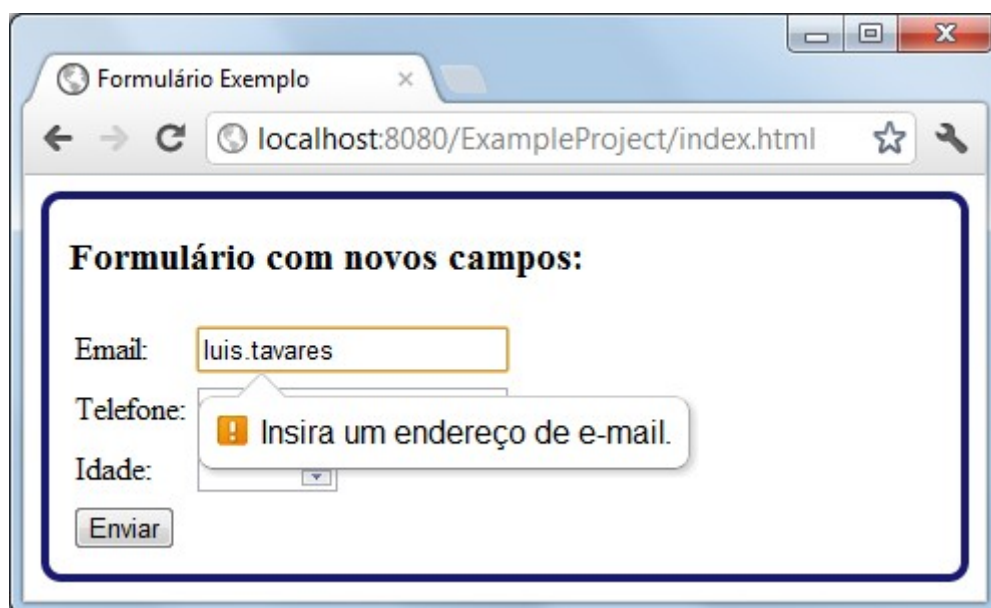
        <td><label>Idade: </label></td>
        <td><input type="number" name="age" required min="18"
max="100"/></td>
    </tr>
</table>
<button type="submit">Enviar</button>
</form>
</div>
</body>
</html>

```

Analizando o exemplo, na tag <body> foi criado um <form> e em seu interior uma tabela foi definida para dispor três labels e três inputs. Nestes três campos declarados, repare que foram utilizados novos valores para a propriedade type e também alguns novos atributos.

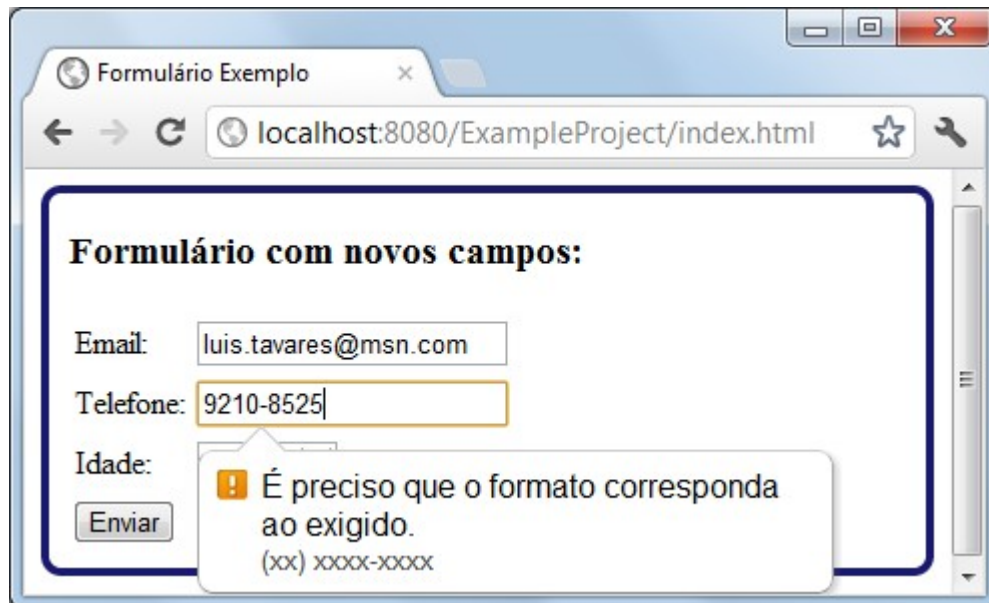
Foram definidos um campo do tipo email, um campo do tipo tel (para telefone) e um último como sendo do tipo number (para idade). Você pode notar que a aparência do campo de tipo number foi alterada para facilitar a inserção e modificação de valores numéricos (observe novamente a **Figura 16**). Neste mesmo campo também é possível estabelecer um valor máximo (propriedade max) e mínimo (propriedade min) que o input pode receber. Por se tratar de um tipo number, apenas valores numéricos serão válidos. Já o campo **email** somente será válido caso seja digitado um endereço de e-mail no formato correto.

Nota-se também, na definição de cada input, o atributo required, que torna o campo de preenchimento obrigatório. Portanto, o HTML já inclui a validação de campos, funcionalidade que já é suportada nas versões atuais de alguns navegadores. Veja a validação em funcionamento na **Figura 17**.



No campo de telefone, há o uso da propriedade pattern, na qual

podemos setar uma expressão regular para validar o formato dos dados preenchidos. A **Figura 18** mostra o atributo `pattern` em uso para validar esse input.



Voltando nossa análise para o início da página, podemos verificar a importação do arquivo de estilo CSS (`style.css`). Assim, crie este arquivo também na pasta *WebContent* e digite o código apresentado na **Listagem 16**.

Listagem 16. Código CSS do exemplo.

```
div {  
    border: 4px solid #191970;  
    border-radius: 10px;  
}  
form {  
    margin: 10px;  
}
```

No código CSS, é definida uma borda para a `div` que contém o formulário e, através da [propriedade `border-radius`](#), os cantos da mesma são arredondados. O resultado dessas formatações pode ser observado nas figuras que apresentam o formulário.

Novamente estudando o funcionamento do exemplo, no momento em que os inputs forem preenchidos e o botão *Enviar* for pressionado, o navegador irá realizar a verificação dos dados com

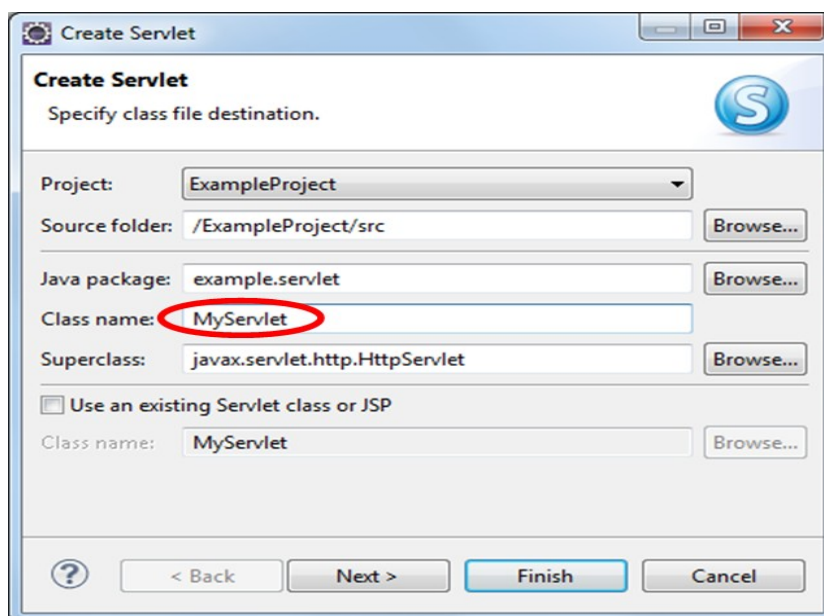
base nos atributos HTML de validação definidos em cada campo e exibir as mensagens com os erros de validação, se houver algum dado inválido. Essas mensagens podem ser observadas nas **Figuras 17 e 18**.

Uma vez que os campos tenham sido preenchidos de forma correta e o botão *Enviar* pressionado, os valores são validados e depois ocorre a submissão dos dados, enviando-os assim para a parte servidor do nosso exemplo – a classe *MyServlet*, que vamos implementar posteriormente.

A validação desenvolvida demonstra que usando os recursos do HTML5 não é mais necessário o uso de scripts adicionais, muitas vezes trabalhosos, para verificar os dados. A validação pode ser feita de maneira simples, apenas inserindo algumas propriedades nos próprios inputs.

Após os inputs serem preenchidos e verificados, os dados serão enviados para *MyServlet*, que vamos implementar neste momento. Para isso, devemos criar um pacote dentro da pasta *src* do projeto. O pacote receberá o nome de *example.servlet*, e dentro deste codificaremos uma classe que será responsável apenas por simular a parte servidor da aplicação Java.

Deste modo, clique com o botão direito sobre o pacote e depois em *New > Servlet*. Em seguida, será apresentada a janela para criação de um Servlet, conforme a **Figura 19**. Dê o nome de “*MyServlet*” para a classe e clique em *Finish*.



A **Listagem 17** exibe o código do nosso Servlet. Observe que ao

receber os parâmetros passados pelo formulário através do método doPost(), os valores são armazenados em variáveis. Depois essas variáveis são impressas no console da aplicação com chamadas a System.out.println() e uma mensagem é retornada para o navegador, através do método response.getWriter().write(), informando que o formulário foi submetido com sucesso. Veja a **Figura 20**.

Listagem 17. Código do Servlet da aplicação.

```
package example.servlet;
```

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

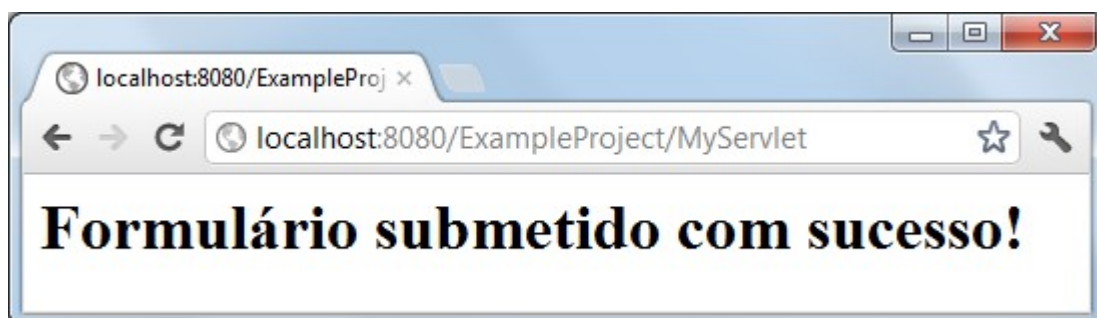
@WebServlet("/MyServlet")
public class MyServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException
    { /* Sem implementação */ }

    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException
    {
        String email = request.getParameter("email");
        String phone = request.getParameter("phone");
        String age = request.getParameter("age");

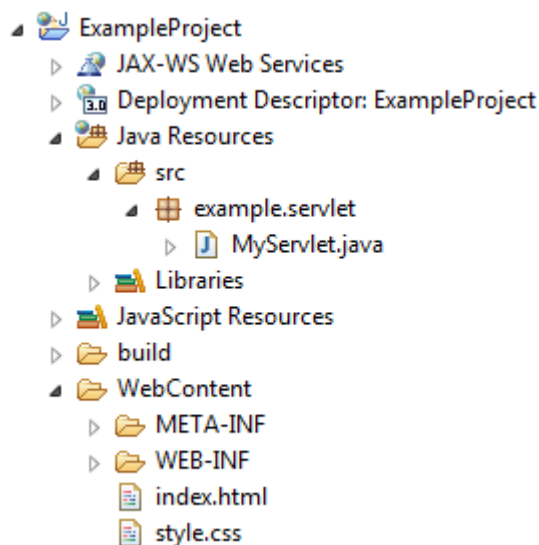
        System.out.println("\nParâmetros recebidos: ");
        System.out.println("email: " + email);
        System.out.println("telefone: " + phone);
        System.out.println("idade: " + age);

        response.getWriter().write("<h1>Formulário submetido com sucesso!</h1>");
        response.getWriter().flush();
    }
}
```



Para executar a aplicação e observar o seu funcionamento, clique com o botão direito do mouse sobre a página *index.html* e selecione *Run As > Run on Server*.

A **Figura 21** apresenta a estrutura de pastas e arquivos criada ao final da implementação do projeto.



Conclusões

Este artigo apresentou uma introdução às novas funcionalidades presentes na versão 5 do HTML, em sua API JavaScript e na versão 3 do CSS. Estas novidades foram estudadas a partir de exemplos práticos, que foram analisados passo-a-passo para facilitar a compreensão.

Além dos vários exemplos nas seções iniciais, no final do artigo foi desenvolvida uma aplicação que demonstrou toda a simplicidade de implementar uma validação de campos usando os novos atributos do HTML5, não sendo mais necessário a criação de scripts adicionais para este fim e nem mesmo a definição de elementos HTML para dispor mensagens de erro.