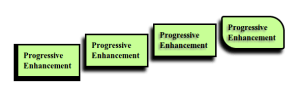
**Aplicando o progressive enhancement**

Postado dia 31/10/2012 por Flávio Almeida em [Front-end](https://blog.caelum.com.br/category/front-end/) [16](https://blog.caelum.com.br/aplicando-o-progressive-enhancement/#comments)

Sempre desejou utilizar recursos modernos do HTML5 e do CSS3, mas era impedido por ter que suportar navegadores mais antigos? Seu site para de funcionar com JavaScript desabilitado?

[](https://i2.wp.com/blog.caelum.com.br/wp-content/uploads/2012/10/pe.png?ssl=1)

Neste post, mostrarei maneiras de aplicar o conceito de progressive enhancement atacando a tríade estrutura, estilo e comportamento para ajudá-lo na difícil tarefa de agradar gregos e troianos que chegam em seu website.

**Uma analogia simples**

Havia um grande rio que cortava duas cidades e dois concorrentes no negócio de travessia. O primeiro utilizava canoa e o segundo jet ski. O primeiro, para **melhorar a experiência de seus usuários**, adicionou um motor à canoa.

Tudo funcionava perfeitamente até que houve escassez de combustível. Sem energia, o jet ski deixou de funcionar e nenhuma travessia era feita. Com a canoa, ainda era possível **navegar**, mesmo sem o motor do concorrente, **permitindo que os usuários ainda acessem este recurso**.

**E o Progressive enhancement?**

O conceito de progressive enhancement define que a construção de uma página parte de uma **base comum e garantida de executar nos mais diversos navegadores** para depois acrescentar pequenas melhorias mesmo que só funcionem em navegadores modernos.

Se alguma dessas melhorias não for suportada pelo navegador, o usuário ainda assim conseguirá acessar o website, mesmo que tenha sua experiência reduzida.

**Este conceito não se aplica uniformemente numa página** e deve ser pensando isoladamente para a estrutura, estilo e comportamento. Cada ponto da tríade se comporta diferentemente quando degradado, isto é, quando não é suportado pelo navegador.

Uma maneira de pensar em cada ponto é através do **critério fail-safe**.

**O critério fail-safe**

O critério *fail-safe* diz que, se um determinado recurso não é suportado pelo navegador, isso não deve resultar em erro, até mesmo sem haver a necessidade de tratamento especial pelo programador. Estão incluídos neste critério o  HTML e o CSS.

No HTML, quando usamos alguma tag desconhecida pelo navegador, nenhum erro é gerado, porque **a tag é simplesmente ignorada**.

Um exemplo prático disso é a utilização da tag <nav> do HTML5. Podemos utilizá-la em nossa marcação visando melhorar a semântica de nossa página, mas se o navegador não a suportar, a lista com os links de navegação ainda continuará acessível:

|  |
| --- |
| <nav>  <ul>    <li><a href="”#”">Produtos</a></li>    <li><a href="”#”">Promoções</a></li>    <li><a href="”#”">Contato</a></li>  </ul>  </nav> |

Este comportamento existe desde que web é web e seu benefício se coaduna com o progressive enhancement de maneira *out of the box*, sem a intervenção do programador.

O programador front-end pode utilizar tags mais modernas com a certeza de que navegadores que não as suportarem as ignorarão. Como nenhum erro é gerado, o usuário consegue acessar o recurso desejado.

Tags não suportadas são ignoradas, logo, não podem ser estilizadas. isto é um problema, principalmente se você precisa estilizar uma tag contêiner como <section> do HTML5. Todas as versões do Internet Explorer inferiores a versão 9 sofrem deste problema.

Este problema é resolvido através do hack [html5shiv](http://code.google.com/p/html5shiv/), que torna estilizáveis as tags do HTML5 nessas versões do IE. Este hack é carregado através de um **comentário condicional**.

**Comentário Condicional**

Um [comentário condicional](http://www.quirksmode.org/css/condcom.html) é uma maneira de carregar scripts e estilos condicionados à versão do Internet Explorer.

Um ponto forte desta técnica é que apenas o IE consegue entendê-lo, sendo ignorado pelos demais browsers. No exemplo abaixo, serão realizados ajustes específicos para cada versão do IE:

|  |
| --- |
| <!--[if IE 6]>  <link rel="stylesheet" type="text/css" href="apenas-para-ie6.css">  <![endif]-->  <!--[if IE 7]>  <link rel="stylesheet" type="text/css" href="apenas-para-ie7.css">  <![endif]--> |

O Internet Explorer 10 [não suportará comentários condicionais](http://www.sitepoint.com/microsoft-drop-ie10-conditional-comments/).

**CSS e progressive enhancement**

Assim como o HTML, o CSS também é *fail-safe*: se alguma propriedade não existir, ela será ignorada sem comprometer o acesso ao recurso pelo usuário. Além disso, essa característica pode proporcionar ao mesmo tempo resultados louváveis.

Um exemplo clássico de design progressivo:

|  |
| --- |
| h1{    background-color: rgb(127, 214, 110);    background-color: rgba(127, 214, 110, .1);  } |

No exemplo acima, o seletor aplica à propriedade **background-color** uma cor através de **rgb**. Esta propriedade repete-se logo em seguida, mas recebendo a mesma cor através de  **rbga** com suporte à  transparência. O primeiro possui ampla cobertura pelos navegadores, já o segundo, nem tanto.

Acontece que, após a primeira propriedade ter sido atribuída, ela será sobrescrita com o valor da segunda, apenas se o navegador a suportar. Sendo assim, aquele desprovido de transparência terá seu fundo estanque. Nenhum erro ocorrerá, consagrando a natureza *fail-safe* do CSS.

Você pode encontrar mais exemplos no excelente post [css3 e progressive enhancement](https://blog.caelum.com.br/css3-e-progressive-enhancement/).

**Estilos conflitantes**

Diferente do HTML, um design progressivo com CSS demanda do programador front-end um pouco mais de atenção:

|  |
| --- |
| h1 {   display: inline-block;   border-style:solid;   border-width: 1px 1px 4px 4px;   box-shadow: -3px 3px 2px;  } |

Para criar um efeito de sombra, foi utilizada a propriedade **border-width** com suporte consistente em diversos navegadores. Logo em seguida, a propriedade **box-shadow do CSS3**, com suporte mais reduzido do que o primeiro, mas com uma experiência visual aprimorada.

Caso o browser não suporte box-shadow, a propriedade border-width será aplicada, mas se a primeira também for suportada, teremos duas propriedades funcionando concomitantemente, o que é um problema.

Repare que aqui não é uma questão de *fail-safe*, apesar dele também poder fazer parte da equação, mas **uma questão condicional**, ou seja, aplicação de estilos condicionada ao suporte ou não de deterninado recurso.

**JavaScript não-obstrusivo e progressive enhancement**

**O JavaScript em sua natureza não é fail-safe**, sendo o ponto da tríade mais problemático.  
No lugar de aplicar o conceito *fail-safe*, aplica-se a técnica de **JavaScript não-obstrusivo**.

[**A técnica de JavaScript não-obstrusivo**](https://blog.caelum.com.br/boas-praticas-com-javascript-e-jquery-codigo-nao-obstrusivo/)**, a grosso modo, parte da premissa na qual usuários sem suporte à JavaScript conseguirão consumir a página, pois o não funcionamento de scripts não bloqueará o acesso ao conteúdo**.

Esta técnica muda a maneira pela qual o JavaScript é visto, considerando-o como um **“plus” e nunca um recurso fundamental para o funcionamento da página**.

É devido a essas características que esta técnica se coaduna com o conceito de progressive enhancement, pois uma base sólida e garantida de funcionar nos mais diversos navegadores é uma base sem JavaScript. Um exemplo:

|  |
| --- |
| window.localStorage.cep = "XXXXX-YYY"; |

O código acima guarda o CEP digitado pelo usuário para que ele não tenha que digitá-lo toda vez, mas **não funcionará e gerará um erro** caso o browser não suporte localStorage do HTML5, como é o caso do IE 7.0 entre outros.

A ausência desse recurso tornará menos agradável a experiência do usuário (ter que digitar toda vez), mas ele poderá continuar comprando, por exemplo.

**Feature Detection**

Uma das formas de contornar o problema acima é utilizar a técnica de **feature detection**, que consiste em testar a existência de determinada *feature* do navegador, permitindo que o programador decida o que fazer. Assim temos:

|  |
| --- |
| if (window.localStorage){    window.localStorage.cep = “XXXXX-YYY”;  } |

**Polyfill**

O programador, após ter elaborado sua lógica de detecção, no lugar de simplesmente deixar de utilizar a *feature* não suportada, pode criar seu próprio “tapa buraco” ou apelar para bibliotecas de terceiros que mimetizem a *feature* original.

Estas bibliotecas de terceiros são chamada de **polyfills**. Por exemplo, para localStorage é possível utilizar o polyfill <https://gist.github.com/350433>.

**Lembre-se que mesmo que você utilize um polyfil, ele deve ser um “plus”, e não algo sem o qual seu website não funcionará**.

**Feature Detection com Modernizr**

Nem sempre é fácil desenvolver algoritmos de detecção de recurso com no exemplo acima. Além disso, o algoritmo do exemplo é falho, pois a presença de qualquer objeto com o nome localStorage, pode causar falso positivo.

Uma biblioteca que auxilia no processo de detecção é o [Modernizr](http://www.modernizr.com). Há uma série de verificações, inclusive para os recursos mais recentes do HTML5.

Uma vez detectada a ausência de determinado recurso, basta escolher o polyfill (o seu ou de terceiros) de interesse que o Modernizr se encarregará de carregá-lo para você.

|  |
| --- |
| <script type="text/javascript" src="script/modernizr-custom.js"></script>  <script type="text/javascript">    Modernizr.load({    test: Modernizr.localstorage,    nope: ['script/localstorage-polyfill.js']  });  </script> |

É possível personalizar o build do Modernizr com os testes de interesse em seu próprio site, diminuindo assim o tamanho da biblioteca final.

**Aplicação condicional de estilos**

O próprio Modernizr ajuda a resolver o problema do nosso CSS, quando temos duas propriedades que não podem ser aplicadas concomitantemente, aplicando-as condicionalmente. Como?

O Modernizr adiciona automaticamente na tag HTML uma classe para cada recurso que detecta e se o recurso não for suportado, ele receberá o prefixo “no-“.

O exemplo abaixo ilustra o suporte ao localStorage e ausência ao box-shadow:

|  |
| --- |
| <html class="localstorage no-boxshadow"]  <!-- restante do html --> |

Agora, só resta alterar o CSS:

|  |
| --- |
| /\* aplicado apenas se box-shadow for suportado \*/  .boxshadow h1 {     box-shadow: -3px 3px 2px  }    /\* aplicado apenas se box-shadow não for suportado \*/  .no-boxshadow h1 {     border-width: 1px 1px 4px 4px;  } |

O exemplo acima leva em consideração as classes adicionadas automaticamente pelo Modernizr, garantindo a aplicação condicional de estilos.

**E se o JavaScript estiver indisponível?**

O Modernizr possui um **mecanismo de fallback** limitado, porém não menos útil para situações nas quais o suporte ao JavaScript esteja indisponível.

A equipe do Modernizr sugere que o programador front-end adicione a classe “no-js” na tag <html>. Quando o Modernizr é carregado (JavaScript habilitado), ele automaticamente mudará a tag “no-js” para “js”. Esse comportamento possibilita aplicar estilos condicionais baseados na disponibilidade ou não do JavaScript:

|  |
| --- |
| .boxshadow h1 {     box-shadow: -3px 3px 2px  }  /\* classe .no-js adicionada \*/  .no-boxshadow h1, .no-js h1 {      border-width: 1px 1px 4px 4px;  } |

Uma possível crítica para esta solução aparece quando o navegador sem suporte à JavaScript tiver suporte à propriedade box-shadow, pois será aplicado o estilo básico utilizando o truque com bordas, ainda sim haverá conformidade com o conceito de progressive enhancement, garantindo uma base sólida de funcionar nos mais diversos navegadores.

**Conclusão**

Aplicar o conceito de progressive enhancement em cada ponto da tríade estrutura, estilo e comportamento é um quebra-cabeça que demanda ainda mais do programador-front.

Novos problemas necessitam de novas soluções, como é o caso da pluralidade de dispositivos móveis, TV’s e até [geladeiras](http://www.gaz.com.br/noticia/257353-samsung_lanca_geladeira_com_acesso_a_internet.html) que hoje acessam nossos websites.

Existem técnicas e recursos prontos para serem utilizados, o progressive enhancement é uma deles, ainda temos o [responsive design](https://blog.caelum.com.br/flexibilidade-em-paginas-para-dispositivos-moveis-com-media-queries/) entre outros.