HTML5, CSS e Javascript

O que é JavaScript?

Controlando o comportamento do HTML e o CSS

JavaScript não é Java

A primeira coisa que você precisa saber: JavaScript não tem nada a ver com Java. Java é uma linguagem server-side, como PHP, Ruby, Python e tantas outras. A única coisa parecida entre eles é o nome.

Sabendo disso, quero que saiba que JavaScript é uma linguagem de programação client-side. Ela é utilizada para controlar o HTML e o CSS para manipular comportamentos na página. Me pergunte agora: "Como assim comportamento?".

Agora eu respondo: um comportamento comum, por exemplo, é um submenu. Sabe quando você passa o mouse em um ítem do menu, e aparece um submenu com vários outros ítens? Pois é. A obrigação de fazer aparecer esse submenu é do JavaScript. O submenu estava escondido, e quando passamos o mouse no ítem, o submenu aparece. Todo esse comportamento quem vai executar é o JavaScript.

Quem criou o JavaScript?

O JavaScript não foi criado pelo W3C, como muitos pensam. Na verdade, ele foi criado por um cara chamado Brendan Eich na Netscape (um dos precursores dos navegadores web). Ele se chamava LiveScript, mas logo seu nome foi mudado para JavaScript. Mesmo assim o nome original é ECMAScript, por que o JavaScript é mantido pela European Computer Manufacturer's Association. Ou seja, chame de JavaScript mesmo, que é como todo mundo chama.

Voltando ao assunto principal: o JavaScript não é mantido pelo W3C, ele é uma linguagem criada e mantida pela ECMA. Eles mantêm uma documentação da linguagem no site deles, mas a melhor documentação ainda são os materiais que você pode encontrar na web mesmo.

Camada de Comportamento

Você já deve ter lido a parte que fala sobre o desenvolvimento separando em camadas, onde explicamos que existem três camadas básicas no desenvolvimento para Web: a informação que fica com o HTML, a formatação, que fica com o CSS e o comportamento, que fica com o JavaScript.

O JavaScript é a terceira camada de desenvolvimento por que ele manipula as duas primeiras camadas, isto é: HTML e CSS. Imagine que você precise de um Slider de imagens. Toda a movimentação, ações de cliques nas setinhas e etc, é o JavaScript que vai cuidar. É isso que chamamos de comportamento.

Orientado a Objeto

Talvez seja cedo demais para falar sobre orientação a objetos em linguagens de programação, mas você precisa saber, pelo menos, que o JavaScript é uma linguagem com Orientação a Objetos. Não vamos entrar em detalhes agora, não queremos que você confunda as bolas. Mas saiba que um objeto na programação é um conjunto de informações. Objeto é um grupo de dados. Mas por hora, fique apenas com essas informações. Vamos nos aprofundar em momento oportuno.

Hello World!

Vamos fazer seu primeiro Hello World com JavaScript?

Primeiro, escreva a estrutura básica do HTML:

```
<!DOCTYPE html>

<html lang="pt-br">

 <head>
```

```
 <title>Título</title>
 <meta charset="utf-8">
 </head>
 <body>


 </body>
</html>
```

Agora, antes do </body> coloque este código:

```
<script type="text/javascript">
 alert('Hello World!');
</script>
```

O código final fica assim:

```
<!DOCTYPE html>
<html lang="pt-br">
 <head>
 <title>Título</title>
 <meta charset="utf-8">
 </head>
 <body>


 <script type="text/javascript">
 alert('Hello World!');
 </script>
 </body>
</html>
```

E pronto, você já escreveu seu primeiro código JavaScript. Abra esse documento HTML no browser e você vai ver uma janela de alerta aparecendo.

HTML5

HTML5 é a mais recente evolução do padrão que define o HTML. O termo representa dois conceitos diferentes:

É uma nova versão da linguagem HTML, com novos elementos, atributos e comportamentos

e um conjunto maior de tecnologias que permite o desenvolvimento de aplicações e web sites mais diversos e poderosos. Este conjunto é chamado HTML5 & friends e muitas vezes abreviado apenas como HTML5.

Criada para ser utilizável por todos os desenvolvedores da Web Aberta, essa página de referências faz ligações a inúmeros recursos do HTML5, classificados em diversos grupos, baseando-se em suas funções;

Semântica: permite você descrever mais precisamente o seu conteúdo.

Conectividade: permite uma comunicação com o servidor de formas modernas e inovadoras.

Offline e armazenamento: Permite que páginas web armazenem dados localmente do lado do cliente e opere de forma offline mais eficientemente.

Multimídia: Viabiliza a utilização de áudio e vídeo de forma primorosa na Web Aberta.

Gráficos e efeitos 2D/3D: viabiliza um leque diversificado de opções de representação gráfica.

Performace e integração: fornece grande otimização de velocidade e melhor utilização do hardware do computador.

Acesso ao dispositivo: viabiliza a utilização de diversos métodos e dispositivos de entrada e saída.

Estilização: permite aos autores a escrita de temas mais sofistificados.

Semântica

Seções e Estruturas em HTML

Uma visão geral sobre as novas estruturas e novos elementos de seção do HTML5: <section>, <article>, <nav>, <header>, <footer> e <aside>

Utilizando Áudio e Vídeo com HTML5

Os elementos <audio> e <video> incorporam e permitem manipulação de novos conteúdos multimídia.

Formulários em HTML5

Uma visão geral sobre as melhorias dos formulários web com o HTML5:

A API de validação de restrição, novos valores para o atributo type dos <input> e o novo elemento <output>.

Novos Elementos Semânticos

Seções laterais, mídia e elementos de formulário: há diversos novos elementos, como <mark>, <figure>, <figcaption>, <data>, <time>, <output>, <progress>, ou <meter> e <main>, incrementando o montante de elementos válidos do HTML5.

Melhorias no <iframe>

Usando os atributos sandbox, seamless, e srcdoc , autores podem ser precisos sobre o nível de segurança e a renderização desejada de um elemento <iframe>.

MathML

Viabiliza a inserção direta de fórmulas matemáticas no código HTML5.

Introdução ao HTML5

Este artigo introduz como indicar para o navegador que você está usando HTML5 em sua página ou aplicação web.

HTML5 parser compatível

O parser, que torna os bytes de um HTML em DOM, foi extendido e agora define precisamente o comportamento em todos os casos, mesmo quando se depara com código HTML inválido.

Isso viabiliza uma grandiosa previsibilidade e interoperabilidade entre navegadores compatíveis com o HTML5.

Conectividade

Web Sockets

Permite a criação de uma conexão permanente entre a página e o servidor para que estes possam trocar dados através desta ligação.

Eventos do servidor

Permite que o servidor envie eventos para um cliente, ao contrário do paradigma clássico onde o servidor pode enviar apenas dados em resposta às requests do cliente.

WebRTC

WebRTC (Comunicação em tempo real), permite conexões entre usuários e controle de videoconferência diretamente no browser, sem necessidade de um plugin ou aplicação externa.

Offline e armazenamento

Recursos offline: cache de aplicação

Firefox possui suporte completo às especificações dos recursos offlines do HTML5. A maioria dos outros navegadores suportam apenas parte deste recurso.

Eventos online e offline

Firefox 3 dá suporte aos eventos WHATWG online e offline, o que permite que aplicações e extensões percebam quando há conexão de Internet, assim como permite a detecção das oscilações de conexão.

WHATWG

Sessão client-side e armazenamento persistente permite que aplicações web armazenem dados de forma estruturada no lado do cliente.

IndexedDB

É um padrão web para armazenamento de quantidades significativas de dados estruturados no navegador e para alta performace de busca nestes dados, usando índices.

Uso de arquivos de aplicações web

Foi adicionado ao Gecko o suporte à nova API de arquivos do HTML5, tornando possível o acesso de arquivos locais pelas aplicações. Isso inclui suporte para seleções de múltiplos arquivos usando <input> do novo tipo file do HTML5. Há também o FileReader.

Multimídia

Utilizando áudio e vídeo com HTML5

Os elementos <audio> e <video> incluem e permitem a manipulação de novos conteúdos multimídia.

WebRTC

Permite conexões entre usuários e controle de videoconferência diretamente no browser, sem necessidade de um plugin ou aplicação externa.

API da câmera

Permite o uso, manipulação e armazenamento de uma imagem diretamente da câmera do computador.

Track e WebVTT

O elemento <track> permite legendas e capítulos. WebVTT é o formato de texto do track <track>.

Gráficos e efeitos 3D

Canvas

Aprenda sobre o novo elemento <canvas> e como utilizá-lo para desenhar gráficos e objetos no Firefox.

API de texto para <canvas>

O elemento <canvas> agora dá suporte à API de texto do HTML5.

WebGL

WebGL traz gráficos 3D à Web, introduzindo uma API que se aproxima bastante à OpenGL ES 2.0, que pode ser usada em elementos <canvas>.

SVG

Um formato de imagens vetoriais baseada em XML que pode ser diretamente embutido no HTML5.

Performance e integração

Web Workers

Permite a delegação da evolução do JavaScript para threads em segundo plano, permitindo que essas atividades sejam prevenidas e assim não deixando as interações dos eventos lentas.

XMLHttpRequest level 2

Permite buscar de forma assíncrona algumas partes da página, permitindo apresentar na tela conteúdo dinâmico, variando de acordo com o tempo e ações do usuário. Está é a tecnologia por trás do Ajax.

Motor JIT-compiling para JavaScript

A nova e poderosa geração de motores JavaScript é muito mais poderosa, levando para uma maior performance.

History API

Permite a manipulação do histórico do navegador. Isso é especialmente útil para páginas que carregam novas informações interativas.

O HTML5 padronizou o atributo contentEditable. Saiba mais sobre este recurso.

Arrastar e soltar

A API do HTML5 permite suportar o recurso de arrastar e soltar (dragging and dropping) items dentro e entre sites da web. Isso também fornece uma simples API para fazer o uso de extensões e aplicações baseadas na Mozilla.

Foco de gestão em HTML

O novo HTML5 activeElement e hasFocus são atributos suportados.

Manipuladores de protocolos beseados na web

Agora você pode registrar aplicações web com manipuladores de protocolos utilizando o método the-navigator.registerProtocolHandler.

requestAnimationFrame

Permite o controle de animações de renderização para obter a performance ideal.

API Fullscreen

Controla o uso de toda a tela para uma página web ou aplicação, sem mostrar a interface de UI do navegador.

API bloqueio de ponteiro

Permite o bloqueio do ponteiro para o conteúdo, para jogos e aplicações semelhantes para não perder o foco quanto o ponteiro atinge o limite da janela.

Eventos online e offline

A fim de construir uma boa aplicação web offline, você precisa saber quando sua aplicação é realmente offline. Aliás, você também precisa saber quando sua aplicação foi retornada por um status online novamente.

Acesso à dispositivos

Usando a API da câmera

É permitido o uso, manipulação, e armazenar imagens através câmeras dos computadores.

Eventos touch

Manipuladores para reagir a eventos criados por um usuário ao pressionar em telas sensíveis ao toque (touch screens).

Utilizando geolocalização

Deixa que os navegadores localizem a posição do usuário utilizando a geolocalização.

Detectando a orientaçåo do dispositivo

Coleta a informação quando o dispositivo em que o browser está rodando muda sua orientação de tela. Isto pode ser utilizado como um dispositivo de entrada (por exemplo, para fazer jogos que utiliza à posiçao do dispositivo) ou para adaptar o layout de uma pagina para a orientaçao da tela (vertical ou horizontal).

Pointer Lock API

Permite que o cursor fique limitado às medidas do conteúdo da aplicação, assim, jogos e outras aplicações não perdem o foto quando o cursos ultrapassa os limites do conteúdo.

Estilização

CSS foi estendido para ser capaz de estilo elementos de uma forma muito mais complexa. Sua extensão, também conhecido como CSS3, mas, como o CSS não segue uma especificação padrão, alguns módulos podem não estar necessariamente na versão 3,. Alguns estão na versão 3 e outros até na 1. Chamar de CSS3 é apenas uma convenção.

Novas caracteristicas dos estilos de background

Agora é possível determinar uma sombra à um elemento, usando a propriedade box-shadow e também podemos definir diversos backgrounds para um elemento.

More fancy borders

Também é possível utilizar imagens para estilizar bordas, usando a propriedade border-image. Bordas arredondadas são suportadas através da propriedade border-radius.

Animating your style

Utilizando transition para animar diferentes estágios de determinadas propriedades ou usando animation para animar trechos da página sem precisar usar o JavaScript com algum evento vinculado, permitindo total controle sobre movimentação de elementos.

Using CSS Transitions to animate between different states or using CSS Animationsto animate parts of the page without a triggering event, you can now control mobile elements on your page.

Typography improvement

Authors have better control to reach better typography. Eles podem controlar text-overflow e hyphenation, mas tambem pode adicionar um shadow a ele ou controlar mais precisamente a sua decorations. Tipos de letras personalizadas podem ser baixadas e aplicadas gracas a nova@font-face at-rule.

Novos layouts de apresentaçoes

A fim de melhorar a flexibilidade dos modelos, foram adicionados, dois novos esquemas: o CSS multi-column layouts e CSS flexible box layout.

(Alguns outros artigos relacionados com HTML5.)

Introdução ao HTML5

Introdução ao HTML5

Este artigo introduz como utilizar HTML5 no desenho de site ou de sua aplicação.

Elementos do HTML5

Lista de tags / elementos do HTML5

Esta página contém uma tabela com todos os elementos (tags) baseado no rascunho atual das especificações do HTML5.

Utilizando audio e video

Adicionando suporte aos elementos do HTML5 <audio> e <video> ao Firefox 3.5.

Formulários em HTML5

Veja as melhorias para formulários web em HTML5: a API de validação de restrição, vários novos atributos, novos valores para <input> atributo type e os novo elemento <output>.

Seções e esboços em HTML5

Veja os novos elementos para delinear e
seccionar em HTML5: <section>, <article>, <nav>, <header>, <footer>, <aside> and <hgroup>.

O elemento <mark>

Este elemento é usado para marcar em destaque um texto de especial relevância.

O elemento <figure> e <figcaption>

Este elemento permite adicionar figuras e ilustrações, com uma eventual legenda, colocado abaixo do texto principal.

Suporte Canvas

Tutorial Canvas

Apreda sobre o novo elemento <canvas> e como desenhar gráficos e outros objetos no Firefox.

HTML5 API texto para elemento <canvas>

HTML5 API texto agora é suportado pelo <canvas>.

Recursos de aplicações web

Recursos Offline

O Firefox suporta completamente as especificações de HTML5 para recurso offline. A maioria dos outros navegadores tem algum nível de suporte aos recursos offline.

Eventos online e offline

O Firefox 3 suporta WHATWG eventos online e offline, que permitem que aplicações e extensões detectem se há ou não uma conexão ativa com Internet, bem como detecta quando a conexão conecta e desconecta.

Sessão WHATWG do lado cliente e armazenamento persistente (aka DOM Storage)

A sessão do lado cliente e o armazenamento persistente permitem que as aplicações web armazenem dados estruturados no lado cliente.

O atributo contentEditable: transforma seu website em um wiki!

O HTML5 tem um atributo padronizado contentEditable. Saiba mais sobre este recurso.

Usando arquivos de aplicações web

Suporta para a nova HTML5 API de arquivo foi adicionada ao Gecko, tornando possível as aplicações web para acessarem arquivos locais selecionados pelo usuário. Isso inclui suportepara selecionar vários arquivos usando o novo elemento HTML <input> do type arquivo de multiplos atributos.

Recursos DOM

getElementsByClassName

O método getElementsByClassName no Document e Element nodes são suportados. Estes métodos permitem encontrar elementos de uma classe ou de uma lista de classes.

Arrastar e soltar

A HTML5 API drag and drop permite suporte para arrastar e soltar itens dentro e entre web sites. Isto também proporciona uma API simples para uso de extensões e aplicativos baseados em Mozilla.

Foco na gestão do HTML

Os novos activeElement e hasFocus são atributos suportados pelo HTML5..

Manipuladores de protocolo baseado em web

Agora você pode registrar uma aplicação web como um manipulador de protocolo usando o método navigator.registerProtocolHandler.

HTML parser

O Gecko é compatível com HTML5 parser—que transforma os bytes de documento HTML em um DOM—foi ativado por padrão a partir de maio de 2010.

(Note que a versão do HTML5 parser que foi incluída no Gecko 1.9.2 / Firefox 3.6 tem bastante erros e não é recomendado para uso real.)

Alterações Adicionais

localName e namespaceURI em documentos HTML agora se comportam como em documentos XML: localName retorna em minúsculas e namespaceURI para elementos HTML é "http://www.w3.org/1999/xhtml"

Quando a URI da página muda o identificador de fragmento de documento (a parte depois do caracter "#" (hash)), um novo evento hashchange é enviado para a página. Veja window.onhashchange para mais informação.

Suporte para element.classList para facilitar o manuseio de atributo de classe.

evnto de documento pronto document.onreadystatechange e document.readyState são propriedades suportadas.

Cores em atributos de apresentação são interpretados de acordo com o HTML5.

Tecnologias muitas vezes chamado de parte do HTML5 que não são

Estas são muitas vezes consideradas em conjunto com um uso amplo termo de "HTML5", mas não são realmente parte do W3C HTML5 Spec.

WebGL

FileReader

XMLHttpRequest

querySelector

Geolocation

ECMAScript5

CSS3

XBL2

Web Workers

Web Sockets

Faster JavaScript

Veja também

Alterações nas versões do Firefox que afetam os desenvolvedores da Web:

Firefox 12

Firefox 11

Firefox 10

Firefox 9

Firefox 8

Firefox 7

Firefox 6

Firefox 5

Firefox 4

Firefox 3.6

Firefox 3.5

Firefox 3

Firefox 2

Firefox 1.5

CSS3 e o futuro da Web

O mundo do desenvolvimento Web passa atualmente por grandes mudanças. Com as aplicações Web exigindo cada vez mais dos navegadores, uma verdadeira guerra está sendo travada no client-side da Web.

O até então onipresente Adobe Flash vem recebendo críticas e mais críticas. Com suas próprias alternativas, Microsoft, Oracle e Google tentam entrar nesse mercado. E, ainda parecendo velhos e ultrapassados em comparação a tudo isso, estão HTML, CSS e JavaScript.

Mas, se depender dos novos HTML5 e CSS3, o cenário será bem diferente no futuro. Com o grande apelo de serem padrões abertos implementados nativamente nos navegadores, essas novas especificações prometem simplificar bastante o desenvolvimento de aplicações ricas no client-sidesem o uso de plugins.

Só para falar do CSS3, assunto desse post, podemos citar entre suas novas capacidades de formatação visual, as queridas bordas redondas, o uso de sombras, múltiplos backgrounds, layouts multi-colunas, novos seletores, uso de fontes externas, transformações 2D e 3D e animações. São recursos que facilitam bastante a tarefa do WebDesigner ao criar interfaces ricas e bonitas.

Embora não seja tão recente assim – o CSS3 data de antes de 2001 -, o suporte nos navegadores ainda é bem precário. Firefox, Safari, Chrome e Opera lideram o movimento, mas ainda estão longe de implementar todos os recursos. O IE tem algumas poucas coisas implementadas em sua versão 8e uma promessa dos desenvolvedores de melhorar um pouco mais na futura versão 9.

Para demonstrar o uso de alguns recursos do CSS3, recriamos o logotipo da Caelum inteiramente usando recursos do CSS3, como border-radius, pseudo-elementos :after e :before e @font-face. O HTML da página é simplesmente:

```
<div class="caelum">
<strong>Caelum</strong>
<em>Ensino e Inovação</em>
</div>
```

E, sem usar nenhuma imagem (nem JavaScript/canvas/SVG), desenhamos o logo da Caelum. Veja a demonstração usando alguma versão recente do Firefox (3.5+), Safari/Chrome ou Opera (10.50+) e veja também o código CSS. E, por ser tudo CSS, sem imagens, podemos mudar o tamanho do logo a vontade, como mostra esse outro exemplo.

CSS3, HTML5 e outros tópicos da “nova Web” são também assuntos do novo treinamento da Caelum, WD-43: Desenvolvimento Web com HTML, CSS e JavaScript. O CSS3 é peça importante no futuro da Web. Seus diversos recursos mudarão a forma como fazemos WebDesign.

CSS3

CSS3 é a segunda mais nova versão das famosas Cascading Style Sheets (ou simplesmente CSS), onde se define estilos para páginas web com efeitos de transição, imagem, e outros, que dão um estilo novo às páginas Web 2.0 em todos os aspectos de design do layout.

A principal função do CSS3 é abolir as imagens de plano de fundo, bordas arredondadas, apresentar transições e efeitos para criar animações de vários tipos, como um simples relógio de ponteiros.

Isso se deve aos novos browsers que estão chegando, com suporte à essa linguagem, como o Google Chrome, Opera, Internet Explorer 9, Safari e Mozilla Firefox. Assim, o CSS3 facilitará o trabalho dos que trabalham com web e também dos usuários, pela variedade de transformações na apresentação de um website.

Diferenciações

elemento

-webkit-border-radius:10px;

-moz-border-radius:10px;

-o-border-radius:10px;

-khtml-border-radius:10px;

Como pode-se ver, foram necessárias quatro linhas de código para produzir bordas arredondadas num elemento HTML. Cada prefixo da propriedade border-radius serve para uma plataforma de browsers. A -webkit, para browsers como Chrome e Safari, -moz para o Firefox, -o para Opera e -khtml para Konqueror. Obs.: há browsers que não aceitam todas as propriedades CSS3. Para testar, utilize Teste de Seletores do CSS3 .Info.

Seletores

Veja uma lista dos principais seletores (propriedades) CSS3 e seus exemplos:

border-radius:[tamanho]; /* bordas arredondadas */

border-radius:5px;

box-shadow:[topo] [esquerda] [borrão] [cor]; /* sombra */

box-shadow:2px 2px 2px #000;

text-shadow:[topo] [esquerda] [borrão] [cor]; /* sombra em letras */

text-shadow:2px 2px 2px #000;

opacity:[valor]; /* transparência */

opacity:0.5;

word-wrap:[definição]; /* quebra de palavra (quando ela ultrapassa o tamanho do elemento) */

word-wrap:break-word;

background-size:[largura] [altura]; /* especifica o tamanho do plano de fundo */

background-size:100px 80px;

Novidades do CSS3

O CSS3 é extremamente capaz de construir animações que impressionam o mais avançado desenvolvedor web, tanto em 2 como em 3 dimensões. Os mais comuns são os efeitos de rotação, movimento e transição.

Existem, na web, empresas fazendo propaganda utilizando a criatividade e o poder dessa nova era de estilos.

WebSocket

WebSocket é uma tecnologia que permite a comunicação bidirecional por canais full-duplex sobre um único soquete Transmission Control Protocol (TCP). Ele é projetado para ser executado em browsers e servidores web que suportem o HTML5, mas pode ser usado por qualquer cliente ou servidor de aplicativos. A API WebSocket está sendo padronizada pelo W3C e o protocolo WebSocket está sendo padronizado pelo IETF.

Websocket foi desenvolvido para ser implementado em browsers web e servidores web, mas pode ser usado por qualquer cliente ou aplicação servidor. O protocolo Websocket é um protocolo independente baseado em TCP. Sua única relação com o HTTP é que seu handshake é interpretado por servidores HTTP como uma requisição de Upgrade.

Esquema de URL

A especificação do protocolo WebSocket define dois tipos de esquemas de URL, ws: e wss:, para conexões não criptografadas e criptografadas respectivamente. Além do esquema de nomes, o resto dos componentes da URL são definidos para usar a síntaxe genérica de URI.

Suporte

Todos os browsers mais atuais com exceção do browser Android suportam a ultima especificação do protocolo Websocket (RFC 6455). Uma suite de testes detalhados para protocolo lista a conformidade destes browsers aos aspectos específicos do protocolo.

Internet Explorer 10+

Mozilla Firefox 4+

Safari 5+

Google Chrome 4+

Opera 11+

Implementation status								
Protocol	**Draft date**	**Internet Explorer**	**Firefox (PC)**	**Firefox (Android)**	**Chrome (PC, Mobile)**	**Safari (Mac, iOS)**	**Opera (PC, Mobile)**	**Android Browser**
hixie-75	February 4, 2010				4	5.0.0		
hixie-76 hybi-00	May 6, 2010 May 23, 2010		4.0 (disabled)		6	5.0.1	11.00 (disabled)	
7 hybi-07	April 22, 2011		6[1]					
8 hybi-10	July 11, 2011		7[1]	7	14			
13 RFC 6455	December, 2011	10	11	11	16	6	12.10	

Exemplos de WebSockets

Node.js

Socket.IO

WebSocket-Node

ws

Java

Jetty

Ruby

EventMachine

Python

pywebsocket

Tornado

Erlang

Shirasu

C++

libwebsockets

.NET

SuperWebSocket

Apresentando Websockets: Trazendo Soquetes Para A Web

O problema: conexões de baixa latência de cliente-servidor e servidor-cliente

A web tem sido construída com base no conhecido paradigma de solicitação/resposta de HTTP. Um cliente carrega uma página da web e, em seguida, nada acontece até que o usuário clique na próxima página. Por volta de 2005, o AJAX começou a deixar a web mais dinâmica.

Mesmo assim, toda a comunicação HTTP era direcionada pelo cliente, o que exigia interação do usuário ou sondagem periódica para carregar novos dados do servidor.

As tecnologias que permitem que o servidor envie dados ao cliente no mesmo momento em que constata que novos dados estão disponíveis foram usadas por algum tempo. Elas eram conhecidas por nomes como "Push" ou "Comet".

Um dos problemas mais comuns para criar a ilusão de uma conexão iniciada pelo servidor é a chamada sondagem longa. Com a sondagem longa, o cliente abre uma conexão HTTP com o servidor que permanece aberta até que a resposta seja enviada.

Sempre que tem novos dados, o servidor envia a resposta (outras técnicas envolvem Flash, solicitações XHR multipart e os chamados htmlfiles). A sondagem longa e as outras técnicas funcionam muito bem. Você as utiliza todos os dias em aplicativos como o chat do Gmail.

No entanto, todas essas soluções compartilham um problema: elas carregam a sobrecarga de HTTP, que não é adequada para aplicativos de baixa latência. Pense em jogos com vários jogadores no navegador ou em qualquer outro jogo on-line com um componente em tempo real.

Apresentando Websocket: Trazendo Soquetes Para A Web

A especificação WebSocket define uma API que estabelece conexões de "soquete" entre um navegador da web e um servidor.

Em outras palavras, há uma conexão persistente entre o cliente e o servidor e ambas as partes podem começar a enviar dados a qualquer momento.

Primeiros Passos

Para abrir uma conexão WebSocket, basta chamar o construtor WebSocket:

```
var connection = new WebSocket('ws://html5rocks.websocket.org/echo', ['soap', 'xmpp']);
```

Observe ws:. Há um novo esquema de URL para conexões WebSocket. Existe também wss: para uma conexão WebSocket segura e, do mesmo modo que https:, é usado para conexões HTTP seguras.

A associação imediata de alguns manipuladores de eventos à conexão permite identificar quando a conexão está aberta, quando há mensagens recebidas ou quando há um erro.

O segundo argumento aceito subprotocolos opcionais. Pode ser uma string ou uma matriz de strings. Cada string deve representar um nome de subprotocolo, e o servidor aceita somente um dos subprotocolos transmitidos na matriz. Para determinar o subprotocolo aceito, acesse a propriedade protocol do objeto WebSocket.

Os nomes de subprotocolos devem ser um dos nomes registrados no registro IANA. Atualmente, há somente um nome de subprotocolo (soap), registrado em fevereiro de 2012.

```
// When the connection is open, send some data to the server

connection.onopen = function

connection.send('Ping'); // Send the message 'Ping' to the server


// Log errors

connection.onerror = function (error) {

console.log('WebSocket Error ' + error);

// Log messages from the server

connection.onmessage = function (e) {

console.log('Server: ' + e.data);
```

Comunicação Com O Servidor

Assim que houver uma conexão com o servidor (quando o evento open for acionado), poderemos começar a enviar dados para o servidor usando o método send('your message') no objeto de conexão. Ele é usado para suportar somente strings, mas, a partir da última especificação, agora também pode enviar mensagens binárias. Para enviar dados binários, use o objeto Blob ou ArrayBuffer.

```
// Sending String

connection.send('your message');

// Sending canvas ImageData as ArrayBuffer

var img = canvas_context.getImageData(0, 0, 400, 320);

var binary = new Uint8Array(img.data.length);

for (var i = 0; i < img.data.length; i++) {

binary[i] = img.data[i];

connection.send(binary.buffer);
```

```
// Sending file as Blob

var file = document.querySelector('input[type="file"]').files[0];

connection.send(file);
```

Da mesma forma, o servidor pode nos enviar mensagens a qualquer momento. Sempre que isso acontece, o retorno de chamada onmessage é acionado. O retorno de chamada recebe um objeto de evento, e a mensagem real pode ser acessada por meio da propriedade data.

WebSocket também pode receber mensagens binárias na última especificação. Os quadros binários podem ser recebidos no formato Blob ou ArrayBuffer. Para especificar o formato do binário recebido, defina a propriedade binaryType do objeto WebSocket como 'blob' ou 'arraybuffer'. O formato padrão é 'blob'. Não é necessário alinhar o parâmetro binaryType durante o envio.

```
// Setting binaryType to accept received binary as either 'blob' or 'arraybuffer'

connection.binaryType = 'arraybuffer';

connection.onmessage = function(e) {

console.log(e.data.byteLength); // ArrayBuffer object if binary
```

As extensões são outro recurso recente do WebSocket. Usando as extensões, será possível enviar quadros compactados, multiplexados etc. Para localizar as extensões aceitas pelo servidor, examine a propriedade extensions do objeto WebSocket depois do evento open. Não há nenhuma especificação de extensão publicada oficialmente até fevereiro de 2012.

```
// Determining accepted extensions

console.log(connection.extensions);
```

Comunicação De Origem Cruzada

Sendo um protocolo moderno, a comunicação de origem cruzada está integrada diretamente no WebSocket. Embora você ainda deva se comunicar somente com clientes e servidores confiáveis, o WebSocket permite a comunicação entre partes de qualquer domínio. O servidor decide se disponibilizará seu serviço para todos os clientes ou somente para os que residem em um conjunto de domínios bem definidos.

Servidores Proxy

Cada nova tecnologia tem um novo conjunto de problemas. No caso do WebSocket, é a compatibilidade com os servidores proxy que faz a mediação das conexões HTTP na maioria das redes corporativas. O protocolo WebSocket usa o sistema de upgrade HTTP (que normalmente é usado para HTTP/SSL) para fazer "upgrade" de uma conexão HTTP para uma conexão WebSocket. Alguns servidores proxy não gostam disso e abandonarão a conexão. Assim, mesmo se um determinado cliente usar o protocolo WebSocket, talvez não seja possível estabelecer uma conexão. Isso torna a próxima seção ainda mais importante :)

Use O Websockets Hoje Mesmo

O WebSocket ainda é uma tecnologia jovem e não foi completamente implementada em todos os navegadores. No entanto, você pode usar o WebSocket hoje com as bibliotecas que usam um dos fallbacks mencionados acima sempre que o WebSocket não estiver disponível.

Uma biblioteca que se tornou muito popular nesse domínio é a socket.io, que é fornecida com um cliente e uma implementação de servidor do protocolo e inclui fallbacks (o socket.io não oferece suporte para mensagens binárias ainda, segundo dados de fevereiro de 2012). Há também soluções comerciais como PusherApp que podem ser facilmente integradas em qualquer ambiente da web fornecendo uma API HTTP para enviar mensagens WebSocket aos clientes. Devido à solicitação HTTP extra, sempre haverá sobrecarga adicional em comparação com o WebSocket puro.

O Lado Do Servidor

O uso de WebSocket cria um padrão de uso totalmente novo para aplicativos de servidor. Embora pilhas de servidor tradicionais como LAMP sejam desenvolvidas com base no ciclo de solicitação/resposta HTTP, elas geralmente não lidam bem com um grande número de conexões WebSocket abertas.

Manter um grande número de conexões abertas ao mesmo tempo requer uma arquitetura que receba alta concorrência com baixo desempenho. Essas arquiteturas normalmente são desenvolvidas com base em encadeamento ou no chamado IO sem bloqueio.

Implementações De Servidor

Node.js

Socket.IO

WebSocket-Node

ws

Java

Jetty

Ruby

EventMachine

Python

pywebsocket

Tornado

Erlang

Shirasu

C++

libwebsockets

.NET

SuperWebSocket

Versões De Protocolo

O protocolo com fio (um handshake e a transferência de dados entre cliente e servidor) para WebSocket agora é RFC6455.

As últimas versões do Google Chrome e do Google Chrome para Android são totalmente compatíveis com RFC6455, incluindo mensagens binárias. Além disso, o Firefox será compatível na versão 11 e o Internet Explorer na versão 10.

Você ainda poderá usar versões mais antigas do protocolo, mas isso não é recomendado porque sabe-se que elas são vulneráveis.

Se você tiver implementações de servidor para versões mais antigas do protocolo WebSocket, recomendamos que faça o upgrade para a versão mais recente.

Casos de Uso

Use o WebSocket sempre que precisar de uma conexão quase em tempo real de baixa latência entre o cliente e o servidor. Tenha em mente que isso pode envolver a reformulação do modo como você cria os aplicativos de servidor com um novo foco em tecnologias como filas de eventos. Alguns exemplos de casos de uso:

Jogos on-line de vários jogadores

Aplicativos de chat

Links para esportes ao vivo

Atualização em tempo real de redes sociais

Demonstrações

Plink

Paint With Me

Pixelatr

Dashed

Massively multiplayer online crossword

Ping server (usado nos exemplos acima)

Exemplo de HTML5demos

Uso de WebSockets e HTML5

Neste artigo vamos trabalhar com o conceito de websockets, onde este é uma nova adição às especificações do HTML5, este, por sua vez, permite que um servidor web consiga estabelecer uma conexão com o navegador e se comunicar diretamente com ele, sem qualquer atraso. Normalmente, a comunicação regular consiste em uma série de pedidos e respostas entre o navegador e o servidor web, onde, para aplicações web que atuam em tempo real, esta técnica não é bem adequada.

Dessa forma, com a utilização de websockets, podemos estabelecer uma conexão uma única vez e, em seguida, a comunicação entre o servidor e o navegador poderão seguir sem problemas de atrasos ou lags.

Esse é o tipo de tecnologia que podemos precisar utilizar para aplicações que requerem atualizações regulares e rápidas a partir de um webserver, como por exemplo, podemos citar aplicações de jogos multiplayer e aplicações de chat, dentre outros.

A ideia inicial para a construção da web foi de que uma cliente requisita dados e o servidor cumpre essas requisições passando os dados solicitados. Esse foi um paradigma incontestável por anos, até o surgimento do AJAX em torno do ano de 2005, onde muitos exploradores procuraram por novas possibilidades de fazer conexões entre um cliente e servidor de forma bidirecional.

Com o crescimento das aplicações web, muitos dados estavam sendo consumidos de forma exponencial e a forma mais importante de manter essas aplicações era com a utilização do modelo de transações HTTP tradicional iniciado pelo cliente.

Para que pudesse ser superado essa barreira, um grande número de estratégias fora desenvolvido de forma a permitir que os servidores enviassem dados para os clientes, dentre os quais se destacou a estratégia de Log-Polling, onde está tinha por objetivo manter a conexão HTTP aberta até que o servidor tivesse dados disponíveis para passar para o cliente.

O problema é que esse tipo de solução levava o HTTP a uma sobrecarga com o aumento da latência. Então o que poderia ser feito para que houvesse uma conexão persistente para suportar as transações iniciadas entre servidor e cliente e que além disso, possuísse uma baixa latência? É disso que trataremos neste artigo, que é a utilização dos websockets.

Como Trabalhar Com Os Websockets?

Os WebSockets têm por finalidade proporcionar uma ligação persistente entre um cliente e um servidor onde ambas as partes podem ser utilizadas para iniciar o envio de dados a qualquer momento.

Desta forma, o cliente estabelece uma conexão WebSocket através de um processo, no qual o cliente envia um pedido via HTTP para o servidor de forma regular, com isso, um cabeçalho de atualização vai incluso neste pedido que informa ao servidor que o cliente pretende estabelecer uma conexão via WebSocket. Um exemplo de cabeçalho que pode ser enviado é apresentado da seguinte forma:

GET ws://exemplo.websocket.com.br/ HTTP/1.1

Origin: http://exemplo.com

Connection: Upgrade

Host: exemplo.websocket.com

Upgrade: websockets

Como podemos observar, a URL do WebSocket usa o esquema ws, onde além deste, ainda podemos utilizar o WSS que é para conexões WebSocket seguras, além de que esta conexão é equivalente de HTTPS.

No caso do servidor ter suporte a utilização de protocolos WebSockets, ele concorda com a atualização e retorna um novo cabeçalho de atualização na sua resposta, como podemos apresentar a seguir:

HTTP 101 WebSocket Protocol Handshake

Date: Wed, 16 Feb 2015 22:07:34 GMT

Connection: Upgrade

Upgrade: WebSocket

Com a comunicação tendo sido concluída, o HTTP inicial é substituído por uma ligação Websocket que utiliza a mesma ligação TCP/IP subjacente, a partir deste momento, qualquer uma das partes poderá iniciar o envio de dados.

Com os Websockets, a transferência de dados pode ser feita sem a ocorrência de sobrecargas associadas as solicitações HTTP, com isso, os dados são transferidos através de um Websocket como uma mensagem, que consiste em um ou mais frames (quadros) que possui os dados que estarão sendo enviados.

A fim de que a mensagem seja reconstruída de forma adequada, ao chegar no cliente, cada um dos quadros é pré-fixado com um tamanho de 4 a 12 bytes de dados sobre a carga útil.

Com a utilização deste sistema de mensagens baseados em frames há uma redução significativa na latência e na quantidade de dados que seria enviado em um único pacote.

É importante notarmos aqui que o cliente só será notificado sobre a nova mensagem uma vez que todos os quadros tiverem sido recebidos e a carga da mensagem original tiver sido reconstruída.

Para que possamos entender melhor com relação ao conceito de Websockets, nada melhor do que aprendermos realizando o desenvolvimento de um exemplo simples, para isso, apresentaremos a partir deste momento alguns dos itens mais importantes e comuns na utilização do Websocket no HTML5.

Criando um Exemplo Simples de WebSockets

Para começarmos com o nosso exemplo, primeiramente escolheremos a Ide que mais nos adequarmos para a construção do nosso código, neste momento, utilizaremos o Netbeans, mas fiquem a vontade

para desenvolver da forma que acharem melhor. Dito isso, criaremos primeiro a nossa página index.html, de acordo com a apresentada pela Listagem 1.

Listagem 1. Criando a página index.html.

```
<!DOCTYPE html>

<html>

<head>

<title>WebSockets com HTML5</title>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" href="style.css">

</head>

<body>

<div id="page-wrapper">

<h1>Exemplo de utilização de WebSockets</h1>

<div id="status">Conectando a aplicação...</div>

<ul id="mensagens"></ul>

<form id="mensagem-formulario" action="#" method="post">

<textarea id="mensagem" placeholder="Escreva a sua mensagem aqui!" required></textarea>

<button type="submit">Enviar mensagem</button>

<button type="button" id="close">Fechar conexão</button>

</form>

</div>

<script src="aplicacao.js"></script>

</body>

</html>
```

Neste momento, temos alguns elementos-chave criados que serão utilizados pela nossa aplicação, dentre os quais, temos aqui uma responsável pela exibição das mensagens sobre o status da conexão; além dela, temos uma lista que vai ser utilizada para apresentarmos as mensagens enviadas e recebidas a partir do servidor; e também, teremos um formulário para inserirmos as mensagens que serão enviadas. Agora que temos a nossa página index.html criada, percebam que teremos também a criação de um arquivo CSS, que chamamos de style e um arquivo js, que chamamos de aplicacao.js, os quais estão sendo apresentados de acordo com as Listagens 2 e 3.

Listagem 2. Criação do arquivo de estilo Style.css.

```
*, *:before, *:after
```

-moz-box-sizing: border-box;

-webkit-box-sizing: border-box;

box-sizing: border-box;

html

font-family: Helvetica, Arial, sans-serif;

font-size: 100%;

background: #333;

#page-wrapper

width: 650px;

background: #FFF;

padding: 1em;

margin: 1em auto;

border-top: 5px solid #69c773;

box-shadow: 0 2px 10px rgba(0,0,0,0.8);

h1

margin-top: 0;

#status {

font-size: 0.9rem;

margin-bottom: 1rem;

.open

color: green;

.closed

color: red;

ul

list-style: none;

margin: 0;

padding: 0;

font-size: 0.95rem;

ul li

padding: 0.5rem 0.75rem;

border-bottom: 1px solid #EEE;

```
ul li:first-child
border-top: 1px solid #EEE;
ul li span
display: inline-block;
width: 90px;
font-weight: bold;
color: #999;
font-size: 0.7rem;
text-transform: uppercase;
letter-spacing: 1px;
.envia
background-color: #F7F7F7;
.recebida {}
#mensagem-formulario {
margin-top: 1.5rem;
textarea
width: 100%;
padding: 0.5rem;
font-size: 1rem;
border: 1px solid #D9D9D9;
border-radius: 3px;
box-shadow: inset 0 1px 1px rgba(0, 0, 0, 0.1);
min-height: 100px;
margin-bottom: 1rem;
button
display: inline-block;
border-radius: 3px;
border: none;
font-size: 0.9rem;
padding: 0.6rem 1em;
color: white;
```

```
margin: 0 0.25rem;

text-align: center;

background: #BABABA;

border-bottom: 1px solid #999;

button[type="submit"]

background: #86b32d;

border-bottom: 1px solid #5d7d1f;

button:hover

opacity: 0.75;

cursor: pointer;
```

Agora que temos a nossa classe style criada, onde definimos apenas alguns itens para uma melhor visualização do nosso projeto exemplo, vamos partir para a criação do nosso arquivo JavaScript, como apresentado pela Listagem 3.

Listagem 3. Criação do arquivo aplicacao.js.

```
window.onload = function

// pega as referencias dos elementos da página.

var form = document.getElementById('mensagem-formulario');

var mensagemTexto = document.getElementById('mensagem');

var listaMensagem = document.getElementById('mensagens');

var socketStatus = document.getElementById('status');

var btnFechar = document.getElementById('close');
```

Como apresentado, criamos aqui um número de variáveis e as inicializamos para a busca dos elementos-chave na página. Com isso, temos o início da nossa aplicação, onde temos uma aplicação demonstrativa configurada e portanto, podemos começar a aprender sobre a API do WebSocket.

De início, veremos aqui como podemos realizar uma conexão através do WebSocket. Para criarmos uma conexão não será nenhuma dificuldade, onde tudo que precisamos fazer é chamar o construtor WebSocket e passar a URL do servidor que será utilizado, como podemos ver através da seguinte passagem que incluiremos no nosso arquivo aplicacao.js, como apresentado pela Listagem 4.

Listagem 4. Abrindo uma conexão com webSockets.

```
// Criando uma nova conexão WebSocket.

var socket = new WebSocket('ws://echo.websocket.org');
```

Para realizarmos os testes com websockets podemos ver através algumas informações relevantes através do seu site. Uma vez que tivermos a conexão estabelecida, o evento open será disparado para a nossa instância WebSocket. Na nossa aplicação, o que estamos fazendo é a adição de um listener do evento que irá atualizar o status da com uma mensagem, apresentando que a conexão foi estabelecida. Agora que temos a nossa conexão, vamos adicionar mais este trecho de código no nosso arquivo aplicacao.js, como mostra a Listagem 5.

Listagem 5. Apresentando uma mensagem de abertura do Websocket.

```
socket.onopen = function(event) {

socketStatus.innerHTML = 'Conectado com: ' + event.currentTarget.URL;

socketStatus.className = 'open';
```

Com os websockets também podemos recuperar erros que possam ocorrer no decorrer de um Listener (escuta) que parta de um evento error. No nosso caso, por simplificação, estamos apenas apresentando esses erros num console através do código apresentado de acordo com a Listagem 6.

Listagem 6. Código referente a ocorrência de erros.

```
socket.onerror = function(error) {

console.log('Erro do WebSocket: ' + error);
```

Agora que vimos, como realizarmos uma conexão e capturarmos erros, o que mais poderíamos ter com os websockets? O que nos falta no momento é o envio das mensagens, as quais, para serem enviadas através da conexão WebSocket, devemos chamar o método send na nossa instância WebSocket, onde passaremos os dados que serão transferidos, o que apresentamos da seguinte forma:

```
socket.send(data);
```

Com os Websockets, podemos realizar o envio de tanto de textos como dados binários, no nosso exemplo, enviaremos o conteúdo do campo para o servidor quando o formulário for enviado. Para que possamos fazer isso, precisaremos inicialmente ter um evento de Listener configurado no formulário, o que faremos de acordo com a Listagem 7.

Listagem 7. Enviando os dados do formulário.

```
form.onsubmit = function(e) {

e.preventDefault;

// Recuperando a mensagem do textarea.

var mensagem = mensagemTexto.value;

// Enviando a mensagem através do WebSocket.

socket.send(mensagem);

// Adicionando a mensagem numa lista de mensagens enviadas.

listaMensagem.innerHTML += '<li class="envia"><span>Enviado:</span>' + mensagem +

'</li>';

// Limpando o campo da mensagem após o envio.

mensagemTexto.value = '';

return false;
```

Quando o formulário for enviado, o código apresentado pela Listagem 7 tem por finalidade recuperar a mensagem do campo mensagem Texto e em seguida, enviá-la através do WebSocket. A mensagem é então adicionada à lista de mensagens chamada de lista Mensagem e apresentada na página. Para concluirmos nosso exemplo, precisamos obter o valor de mensagem Texto como reposta pronta para que o usuário digite uma nova mensagem.

Recebendo uma mensagem

Quando uma mensagem é recebida o evento mensagem é disparado. Este evento inclui uma propriedade chamada de data que poderá ser usada para acessar o conteúdo da mensagem. Para o nosso exemplo, precisaremos criar um evento Listener que será disparado quando uma nova mensagem for recebida, com isso, o código devera recuperar a mensagem do evento e exibi-lo no listaMensagem. Este método está sendo apresentado de acordo com a Listagem 8.

Listagem 8. Copiando a mensagem que é enviada para o servidor.

```
socket.onmessage = function(event) {

var mensagem = event.data;

listaMensagem.innerHTML += '<li class="recebida"><span>Recebida:</span>' +

mensagem + '</li>';
```

Após termos finalizado a utilização do nosso WebSocket, o que precisamos fazer é encerrar a nossa conexão, onde utilizaremos o método close:

```
socket.close;
```

Após a nossa conexão ter sido fechada, o navegador irá disparar um evento close. Associando um evento de Listener ao evento close, o que nos permite a execução de qualquer limpeza que precisarmos fazer. Para demonstrarmos que a nossa conexão foi fechada, iremos apenas atualizar o status da conexão, de acordo com o código apresentado pela Listagem 9, que será adicionado também ao nosso arquivo aplicacao.js.

Listagem 9. Fechamento da conexão WebSocket.

```
socket.onclose = function(event) {

socketStatus.innerHTML = 'Disconectado do WebSocket.';

socketStatus.className = 'closed';
```

Para finalizarmos, vamos adicionar um evento que será disparado quando o botão de “Fechar conexão” for clicado, o que irá chamar o método close do WebSocket, como podemos apresentar de acordo com a Listagem 10.

Listagem 10. Fechando a conexão WebSocket com o botão.

```
btnFechar.onclick = function(e) {

e.preventDefault;

// Fechando o WebSocket.

socket.close;

return false;
```

Agora que apresentamos cada trecho do nosso aplicacao.js, explicando cada ponto dele. Vejamos o arquivo completo através da Listagem 11.

Listagem 11. Arquivo aplicacao.js completo.

```
window.onload = function {

// Pegando as referências para os elementos da página.
```

```
var form = document.getElementById('mensagem-formulario');
var mensagemTexto = document.getElementById('mensagem');
var listaMensagem = document.getElementById('mensagens');
var socketStatus = document.getElementById('status');
var btnFechar = document.getElementById('close');
// Criando uma nova WebSocket.
var socket = new WebSocket('ws://echo.websocket.org');
// segurando os erros que ocorrerem.
socket.onerror = function(error) {
console.log('erros do WebSocket: ' + error);

// Mostrando a mensagem de Conectado quando o WebSocket for aberto.
socket.onopen = function(event) {
socketStatus.innerHTML = 'Conectado com: ' + event.currentTarget.URL;
socketStatus.className = 'open';
// Pegando as mensagens enviadas pelo servidor.
socket.onmessage = function(event) {
var mensagem = event.data;
listaMensagem.innerHTML += '<li class="recebida"><span>Recebida:</span>' +
mensagem + '</li>';
//Mostrando a mensagem de desconectado quando o websocket for fechado.
socket.onclose = function(event) {
socketStatus.innerHTML = 'Disconectando o WebSocket.';
socketStatus.className = 'closed';
//Enviando uma mensagem quando o formulário for submetido.
form.onsubmit = function(e) {
e.preventDefault;
// Recuperando a mensagem do textarea.
var mensagem = mensagemTexto.value;
// Enviando a mensagem através do WebSocket.
socket.send(mensagem);
```

```
//Adicionando a mensagem para a lista de mensagens.
listaMensagem.innerHTML += '<li class="envia"><span>Enviada:</span>' + mensagem +
'</li>';
// Limpando o campo de mensagem.
mensagemTexto.value = '';
return false;
//Fechando a conexão WebSocket quando o botão for clicado.
btnFechar.onclick = function(e) {
e.preventDefault;
// Fechando o WebSocket.
socket.close;
return false;
```

Ao contrário do que o nome indica, o protocolo WebSocket não é apenas sobre a web! A especificação foi implementada a partir dos sistemas operacionais móveis e tablets, incluindo iOS, Android e Windows. Isso significa que podemos usar o poder e a velocidade do protocolo WebSocket em um aplicativo de smartphone nativamente. Os princípios fundamentais permanecem, independentemente de usarmos JavaScript, ou linguagens de programação, como Objective-C ou mesmo o C#.

O protocolo WebSocket como parte do HTML5, possui uma estrutura robusta para o desenvolvimento e a concepção de aplicações web. Não é apenas uma nova marcação ou alguns novos seletores de estilo, nem é uma nova linguagem de programação.

O HTML5 significa uma coleção de tecnologias, linguagens de programação e ferramentas, cada uma das quais possui um papel discreto e todos juntos realizam uma tarefa específica que é a construção de ricas aplicações web para qualquer tipo de dispositivo. Os principais pilares do HTML5 incluem Markup, CSS3 e Api's JavaScript.

Com isso finalizamos o nosso artigo sobre a utilização de WebSockets, onde vimos que ele tem por finalidade definir uma comunicação persistente bidirecional entre servidores e clientes web, o que significa que ambas as partes podem trocar dados de mensagens ao mesmo tempo. Eles são otimizados para obter alto desempenho e resultados em aplicações web muito mais ágeis e ricas.

Com a utilização dos WebSockets, vimos que podemos substituir os long-pollings, onde este é na verdade, um conceito bastante interessante, no qual, um cliente envia uma solicitação para o servidor, e como este ainda não possui os dados de resposta, ele essencialmente mantém a conexão aberta até ocorrer o refresh, onde os dados estarão atualizados e prontos para serem enviados para o cliente que recebe a informação e envia um novo pedido.

Dentre as vantagens que podemos ter com a utilização de websockets, podemos citar aqui a redução na latência, onde, como já temos uma conexão que foi aberta, não será requerida uma nova conexão.