**[认识HTML5的WebSocket](http://www.cnblogs.com/wei2yi/archive/2011/03/23/1992830.html)**

在HTML5规范中，我最喜欢的Web技术就是正迅速变得流行的WebSocket API。WebSocket提供了一个受欢迎的技术，以替代我们过去几年一直在用的Ajax技术。这个新的API提供了一个方法，从客户端使用简单的语法有效地推动消息到服务器。让我们看一看HTML5的WebSocket API：它可用于客户端、服务器端。而且有一个优秀的第三方API，名为Socket.IO。  
  
**一、什么是WebSocket API?**  
  
WebSocket API是下一代客户端-服务器的异步通信方法。该通信取代了单个的TCP套接字，使用ws或wss协议，可用于任意的客户端和服务器程序。WebSocket目前由W3C进行标准化。WebSocket已经受到Firefox 4、Chrome 4、Opera 10.70以及Safari 5等浏览器的支持。  
  
WebSocket API最伟大之处在于服务器和客户端可以在给定的时间范围内的任意时刻，相互推送信息。WebSocket并不限于以Ajax(或XHR)方式通信，因为Ajax技术需要客户端发起请求，而WebSocket服务器和客户端可以彼此相互推送信息；XHR受到域的限制，而WebSocket允许跨域通信。  
  
Ajax技术很聪明的一点是没有设计要使用的方式。WebSocket为指定目标创建，用于双向推送消息。  
  
**二、WebSocket API的用法**  
  
只专注于客户端的API，因为每个服务器端语言有自己的API。下面的代码片段是打开一个连接，为连接创建事件监听器，断开连接，消息时间，发送消息返回到服务器，关闭连接。

[[Copy to clipboard]](http://www.itpub.net/thread-1373652-1-1.html###) [[ - ]](http://www.itpub.net/thread-1373652-1-1.html###)

CODE:

// 创建一个Socket实例  
var socket = new WebSocket('ws://localhost:8080');   
  
// 打开Socket   
socket.onopen = function(event) {   
  
  // 发送一个初始化消息  
  socket.send('I am the client and I\'m listening!');   
  
  // 监听消息  
  socket.onmessage = function(event) {   
    console.log('Client received a message',event);   
  };   
  
  // 监听Socket的关闭  
  socket.onclose = function(event) {   
    console.log('Client notified socket has closed',event);   
  };   
  
  // 关闭Socket....   
  //socket.close()   
};

让我们来看看上面的初始化片段。参数为URL，ws表示WebSocket协议。onopen、onclose和onmessage方法把事件连接到Socket实例上。每个方法都提供了一个事件，以表示Socket的状态。  
  
onmessage事件提供了一个data属性，它可以包含消息的Body部分。消息的Body部分必须是一个字符串，可以进行序列化/反序列化操作，以便传递更多的数据。  
  
WebSocket的语法非常简单，使用WebSockets是难以置信的容易……除非客户端不支持WebSocket。IE浏览器目前不支持WebSocket通信。如果你的客户端不支持WebSocket通信，下面有几个后备方案供你使用：  
  
**Flash技术** —— Flash可以提供一个简单的替换。 使用Flash最明显的缺点是并非所有客户端都安装了Flash，而且某些客户端，如iPhone/iPad，不支持Flash。  
  
**AJAX Long-Polling技术** —— 用AJAX的long-polling来模拟WebSocket在业界已经有一段时间了。它是一个可行的技术，但它不能优化发送的信息。也就是说，它是一个解决方案，但不是最佳的技术方案。  
  
由于目前的IE等浏览器不支持WebSocket，要提供WebSocket的事件处理、返回传输、在服务器端使用一个统一的API，那么该怎么办呢？幸运的是，Guillermo Rauch创建了一个Socket.IO技术。  
  
**三、带Socket.IO的WebSocket**  
  
Socket.IO是Guillermo Rauch创建的WebSocket API，Guillermo Rauch是LearnBoost公司的首席技术官以及LearnBoost实验室的首席科学家。Socket.IO使用检测功能来判断是否建立WebSocket连接，或者是AJAX long-polling连接，或Flash等。可快速创建实时的应用程序。Socket.IO还提供了一个NodeJS API，它看起来非常像客户端API。  
建立客户端Socket.IO  
  
Socket.IO可以从GitHub下载，可以把socket.io.js文件包含到页面中：

[[Copy to clipboard]](http://www.itpub.net/thread-1373652-1-1.html###) [[ - ]](http://www.itpub.net/thread-1373652-1-1.html###)

CODE:

<script src="http://cdn.socket.io/stable/socket.io.js"></script>  
[/code  
  
此时，Socket.IO在此页面上是有效的，是时候创建Socket了：  
  
[code]  
// 创建Socket.IO实例，建立连接  
var socket= new io.Socket('localhost',{   
  port: 8080   
});   
socket.connect();   
  
// 添加一个连接监听器  
socket.on('connect',function() {   
  console.log('Client has connected to the server!');   
});  
  
// 添加一个连接监听器  
socket.on('message',function(data) {   
  console.log('Received a message from the server!',data);   
});  
  
// 添加一个关闭连接的监听器  
socket.on('disconnect',function() {   
  console.log('The client has disconnected!');   
});   
  
// 通过Socket发送一条消息到服务器  
function sendMessageToServer(message) {   
  socket.send(message);   
}

Socket.IO简化了WebSocket API，统一了返回运输的API。传输包括：  
WebSocket  
Flash Socket  
AJAX long-polling  
AJAX multipart streaming  
IFrame  
JSONP polling  
  
你还可以设置任意的Socket.IO构造器的第二个选项，选项包括：  
  
port - 待连接的端口  
transports - 一个数组，包含不同的传输类型  
transportOptions - 传输的参数使用的对象，带附加属性  
  
Socket.IO还提供了由本地WebSocket API提供的普通连接、断开连接、消息事件。Socket还提供了封装每个事件类型的方法。  
  
**四、NodeJS和Socket.IO联合开发**  
  
Socket.IO提供的服务器端解决方案，允许统一的客户端和服务器端的API。使用Node，你可以创建一个典型的HTTP服务器，然后把服务器的实例传递到Socket.IO。从这里，你创建连接、断开连接、建立消息监听器，跟在客户端一样。  
  
一个简单的服务器端脚本看起来如下：

[[Copy to clipboard]](http://www.itpub.net/thread-1373652-1-1.html###) [[ - ]](http://www.itpub.net/thread-1373652-1-1.html###)

CODE:

// 需要HTTP 模块来启动服务器和Socket.IO  
var http= require('http'), io= require('socket.io');   
  
// 在8080端口启动服务器  
var server= http.createServer(function(req, res){   
  // 发送HTML的headers和message  
  res.writeHead(200,{ 'Content-Type': 'text/html' });   
  res.end('<h1>Hello Socket Lover!</h1>');   
});   
server.listen(8080);   
  
// 创建一个Socket.IO实例，把它传递给服务器  
var socket= io.listen(server);   
  
// 添加一个连接监听器  
socket.on('connection', function(client){   
  
  // 成功！现在开始监听接收到的消息  
  client.on('message',function(event){   
    console.log('Received message from client!',event);   
  });   
  client.on('disconnect',function(){   
    clearInterval(interval);   
    console.log('Server has disconnected');   
  });   
});

你可以运行服务器部分，假定已安装了NodeJS，从命令行执行：

[[Copy to clipboard]](http://www.itpub.net/thread-1373652-1-1.html###) [[ - ]](http://www.itpub.net/thread-1373652-1-1.html###)

CODE:

node socket-server.js

现在客户端和服务器都能来回推送消息了！在NodeJS脚本内，可以使用简单的JavaScript创建一个定期消息发送器：

[[Copy to clipboard]](http://www.itpub.net/thread-1373652-1-1.html###) [[ - ]](http://www.itpub.net/thread-1373652-1-1.html###)

CODE:

// 创建一个定期（每5秒）发送消息到客户端的发送器  
var interval= setInterval(function() {   
  client.send('This is a message from the server! ' + new Date().getTime());   
},5000);

服务器端将会每5秒推送消息到客户端！  
  
**五、dojox.Socket和Socket.IO**  
  
Persevere的创建者Kris Zyp创建了dojox.Socket。dojox.Socket以Dojo库一致的方式封装了WebSocket API，用于在客户端不支持WebSocket时，使用long-polling替代。  
  
下面是怎样在客户端使用dojox.Socket和在服务器端使用Socket.IO的例子：

[[Copy to clipboard]](http://www.itpub.net/thread-1373652-1-1.html###) [[ - ]](http://www.itpub.net/thread-1373652-1-1.html###)

CODE:

var args, ws= typeof WebSocket!= 'undefined';   
var socket= dojox.socket(args= {   
  url: ws? '/socket.io/websocket' : '/socket.io/xhr-polling',   
  headers:{   
    'Content-Type':'application/x-www-urlencoded'   
  },   
  transport: function(args, message){   
    args.content = message; // use URL-encoding to send the message instead of a raw body   
    dojo.xhrPost(args);   
  };   
});   
var sessionId;   
socket.on('message', function(){   
  if (!sessionId){   
    sessionId= message;   
    args.url += '/' + sessionId;   
  }else if(message.substr(0, 3) == '~h~'){   
   // a heartbeat   
  }   
});

dojox.socket.Reconnect还创建了在套接字失去连接时自动重连。期待包含dojox.Socket的Dojo 1.6版本早日发布。  
  
**六、实际应用和WebSocket资源**  
  
有很多WebSocke的实际应用。WebSocket对于大多数客户机-服务器的异步通信是理想的，在浏览器内聊天是最突出的应用。WebSocket由于其高效率，被大多数公司所使用。  
  
**WebSocket资源**  
Socket.IO站点：[http://socket.io/](http://socket.io/" \t "_blank)  
WebSocket的Wikipedia：[http://en.wikipedia.org/wiki/WebSockets](http://en.wikipedia.org/wiki/WebSockets" \t "_blank)  
WebSockets.org站点：[http://www.websockets.org/](http://www.websockets.org/" \t "_blank)  
Dojo WebSocket站点：<http://www.sitepen.com/blog/2010/10/31/dojo-websocket/>