HTML5 Websockets with ruby and rails

Saurabh Bhatia, CEO, Safew Labs

Meet the humble "Websocket"

1. ## What is it ?

   WebSocket is a technology providing for bi-directional, full-duplex communications channels, over a single TCP socket.

   The aim is to provide a standard W3C standard complient API that runs on standard ports, facilitating easy implementation and wider acceptance.

2. ## Where is it found ?

   Browser - Safari 5 , Google Chrome 4,Firefox 4 , Opera 11.

   Mobile Browsers - Safari for iOS 4.2, Blackberry Browser

3. ## Similar Existing Stuff

   Comet , Reverse HTTP , Webhooks

The Poll that matters

Polling is a technique that sends out a poll request to the server at regular intervals (cron job) , so the freshness of data depends upon the frequency at which polling happens.

- Simple HTTP Request - Works with all routers, firewalls etc
- Huge Number of Polling Requests - High Bandwidth, More Infrastructure Required, Hence cost is high. Large buffer required for this.
- PointCast network first implemented this and ended up choking their network. **#fail**

Enter Long Polling

Why is Long Polling a #fail

- **How it works?**

  Browser sends a request to the Server

  Server Keeps it Open for a Particular time.

  If a response notification is received within the period,it is forwarded to the client

  If a response is not received within the period, the request is terminated and a fresh request is sent

- **The #fail Part**

  Large Number of requests - No improvement over polling

  Large number of http headers - Becomes a bandwidth hog

  Larger Cache requirements for the server

  What happens if the data is generated when the server is holding the poll requests ?

  HTTP 1.1 , Specification Section 8.1.4 - Browser should not hold more than two simultaneous requests open to the same server at the same time.

Is Streaming an alternative ?

- ### How it works?

  Browser sends a complete request

  Server keeps an Open Connection from it's side and keeps updating it.

  A request is kept Open infinitely

  Server sends a message but does not close the request. It keeps it open for future messages.

- ### Good Part

  Comparitively less bandwidth usage because of sending of data-only packets and not multiple HTTP headers.

- ### Bad Part

  The request is still encapsulated as an HTTP request, so intervening HTTP proxies may choose to buffer the response ,reducing the latency of message delivery.

Enter HTML5 Websocket

- ### What is it ?

  An attempt at standardization of sockets through a standard HTML API.

  A Full duplex means of communication

- ### Good Part

  No HTTP headers used, so very lightweight to use.

- ### Bad Part

  Still a work in progress

Specifications

- ### URL Basics

```
var connection = new WebSocket('ws://your.websocketurl.com');
```

ws - a new url schema specific to HTML

wss - SSL based web socket url schema similar to https

runs on port 81

- ### Event handlers

Open Connection

```
// When the connection is open, send some data to the server
connection.onopen = function () {
  connection.send('Ping'); // Send the message 'Ping' to the server
};
```

Error Logging

```
// Log errors
connection.onerror = function (error) {
  console.log('WebSocket Error ' + error);
};
```

A Real Websocket Connection

```
var socket = new WebSocket('ws://echo.websocket.org');
socket.onopen = function(event) {
  socket.send('Hello, WebSocket');
};
socket.onmessage = function(event) { alert(event.data); }
socket.onclose = function(event) { alert('closed'); }
```

Full-duplex, bi-directional communication over the Web: Both the server and client can send data at any time, or even at the same time. Only the data itself is sent, without the overhead of HTTP headers, dramatically reducing bandwidth.

Location:

ws://echo.websocket.org

☐ Use secure WebSocket (TLS/SSL)

Connect Disconnect

Message:

Hello, WebSocket Send

Output:

Clear log

Websockets - The Ruby Way

- **eventmachine** - EventMachine is an event-driven I/O and lightweight concurrency library for Ruby. It provides event-driven I/O using the Reactor pattern
- **cool.io**- event driven programming library for ruby
- **em-websockets** - An event machine based websocket server

eventmachine

Event Driven I/O Library for Ruby..

```
gem install eventmachine
```

```
require 'rubygems'
require 'eventmachine'
require 'em-http-request'

EventMachine.run {
  http = EventMachine::HttpRequest.new('ws://echo.websockets.org').get :timeout => 0

  http.callback {
    puts "WebSocket connected!"
    http.send("Hello client")
  }

  http.stream { |msg|
    puts "Recieved: #{msg}"
    http.send "Pong: #{msg}"
  }
}
```

cool.io

A Cool pure ruby websocket

```
gem install cool.io
```

```
require 'rubygems'
require 'cool.io'
HOST = 'localhost'
PORT = 4321

class EchoServerConnection < Cool.io::TCPSocket
  def on_connect
    puts "#{remote_addr}:#{remote_port} connected"
  end

  def on_close
    puts "#{remote_addr}:#{remote_port} disconnected"
  end

  def on_read(data)
    write data
  end
end

server = Cool.io::TCPServer.new(HOST, PORT, EchoServerConnection)
server.attach(Cool.io::Loop.default)

puts "Echo server listening on #{HOST}:#{PORT}"
```

Reactor Pattern

Is a design pattern for programming such that service requests are delivered concurrently to a service handler by one or more inputs.The service handler then demultiplexes the incoming requests and dispatches them synchronously to the associated request handlers.

Performance wise, events driven is better than forking.

```
require 'rubygems'
require 'eventmachine'
require 'evma_httpserver'

class Handler  < EventMachine::Connection
  include EventMachine::HttpServer

  def process_http_request
    resp = EventMachine::DelegatedHttpResponse.new( self )

    sleep 2 # Simulate a long running request

    resp.status = 200
    resp.content = "Hello World!"
    resp.send_response
  end
end

EventMachine::run {
  EventMachine.epoll
  EventMachine::start_server("0.0.0.0", 8080, Handler)
  puts "Listening..."
}
```

EM Websocket

Websocket server based on Event Machine

```
gem install em-websocket
```

```
require 'rubygems'
require 'em-websocket'

EventMachine.run {

    EventMachine::WebSocket.start(:host => "0.0.0.0", :port => 8080) do |ws|
        ws.onopen {
          puts "WebSocket connection open"

          # publish message to the client
          ws.send "Hello Client"
        }

        ws.onclose { puts "Connection closed" }
        ws.onmessage { |msg|
          puts "Recieved message: #{msg}"
          ws.send "Pong: #{msg}"
        }
    end
}
```

Wrapping these in a Rails app

Thank You