Web Sockets

olsen
software ltd

Contents

1. Overview of Web sockets
2. Defining a Web Sockets server
3. Defining a Web Sockets client

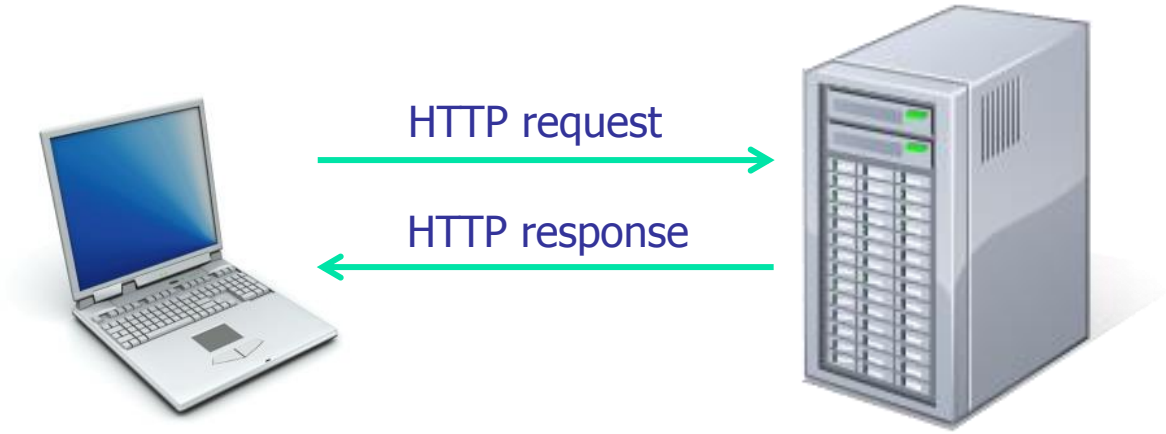
Demos folder:
Demos-11-WebSockets

1. Overview of Web Sockets

- Traditional HTTP Communication
- Polling
- Long Polling
- Limitations of HTTP
- Introducing Web Sockets
- How Web Sockets work
- Web Socket enabled servers

Traditional HTTP Communication

- In traditional HTTP:
  - A browser sends an HTTP request to a web server
  - The web server sends an HTTP response, e.g. an HTML page



- The Web page is an expression of data at one moment
  - E.g. stock prices, news reports, ticket sales, etc.

- There are several ways to keep the page up to date:
  - Polling, long polling

Polling

- This is how polling works in traditional HTTP:
  - The browser sends AJAX requests to the server at regular intervals
  - The server responds immediately, possibly with updated data
  - The process repeats every few seconds

- Remarks:
  - Real-time data is often not that predictable, so the browser will probably make unnecessary requests
  - HTTP connections will probably be opened and closed needlessly in low-message-rate situations
  - Each Ajax request is a separate HTTP request, and HTTP is quite an inefficient protocol

Long Polling

- This is how long polling works in traditional HTTP:
  - The browser connects to the server, setting a connection timeout to a very big value (e.g. hours)
  - The browser then sends a request to the server, to get new data
  - The server returns new data when available, or connection times out
  - The process then repeats

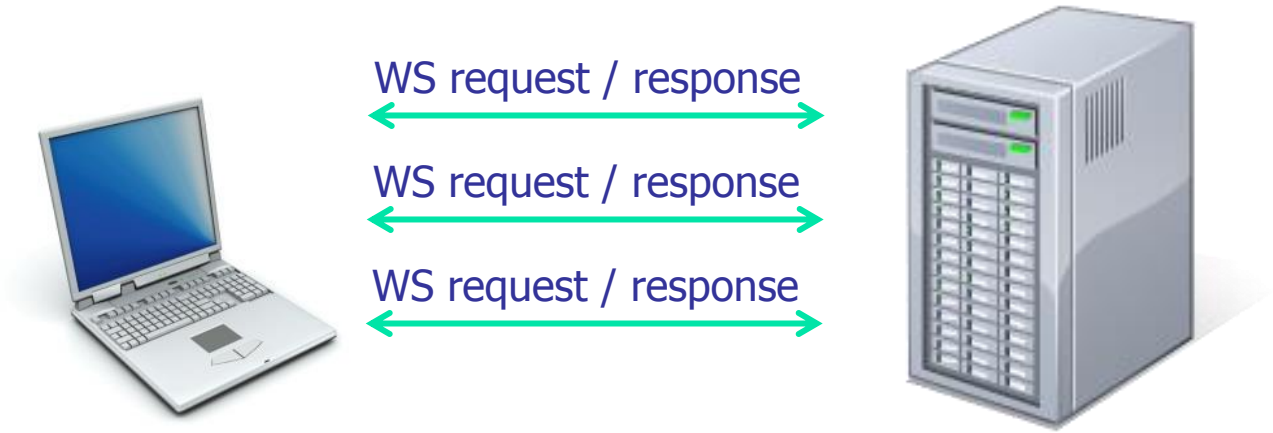
- Remarks:
  - Better than polling, because fewer connections opened/closed
  - But there is an overhead in maintaining an open connection
  - Also the server might only support a limited number of connections

Limitations of HTTP

- HTTP was not designed for real-time full-duplex communication
  - The methods described earlier involve lots of HTTP request/response headers, which causes high latency
- Furthermore, in an attempt to simulate full-duplex communication over half-duplex HTTP…
  - The traditional approaches have typically used two separate connections (one for upstream, one for downstream)
  - Maintaining and coordinating these two connections consumes extra resources and adds complexity

Introducing Web Sockets

- Web Sockets defines a full-duplex socket-based communication channel between browser and server
  - Simultaneous two-way data exchange between client and server
  - A large advance in HTTP capabilities
  - Extremely useful for real-time, event-driven Web applications



How Web Sockets Work

- To establish a Web Socket connection between a client and a server:
  - The client sends an HTTP or HTTPS request, containing an "upgrade to WebSocket" HTTP header
  - If the server supports Web Sockets, it returns an HTTP response containing an "OK to upgrade to WebSocket" HTTP header
  - A persistent two-way socket connection is established
- Once a Web Socket connection has been established:
  - The client and server can send and receive messages over the open connection simultaneously
  - The format of the data can be anything that both parties are happy with

Web Socket Enabled Servers

- Microsoft
  - Internet Information Services 8 and later
- Node.js
  - http://nodejs.org
- PHP
  - http://code.google.com/p/phpwebsocket/
- Apache HTTP Server extension
  - http://code.google.com/p/pywebsocket/
- Java
  - Java EE 7
- Ruby
  - https://github.com/gimite/web-socket-ruby

2. Defining a Web Sockets Server

- Overview
- Configuring a server for Web Sockets
- Creating an IIS Web Application
- Understanding the server code
- Publishing the project

Overview

- In this section we're going to see how to implement a Web Sockets server
  - We'll implement the server as an ASP.NET HTTP handler
  - An ASP.NET HTTP handler is an object called by ASP.NET to handle requests, usually associated with a `.ashx` file extension

- The demo code is located in the `HelloWebSockets` folder
  - Server file: `SocketServer.cs` (discussed in this Section)
  - Client page: `SocketClient.html` (discussed in next Section)

Configuring a Server for Web Sockets

- IIS 8 supports Web Sockets, but you have to explicitly enable this feature on the computer first
  - This is a one-off task at the server, and it will probably already be taken care of by the system administrators at your place of work

- Follow these steps:
  - Open Control Panel and click Programs and Features.
  - In the Programs and Features window, click "Turn Windows features on or off"
  - In the Windows Features dialog box, enable the following features:
    - Expand the .NET Framework 4.5 Advanced Services node, and enable the ASP.NET 4.5 option
    - Expand the Internet Information Services node, and enable the Web Management Tools and World Wide Web Services options
    - Expand the World Wide Web Services option itself, and enable the ASP.NET 4.5 and WebSocket Protocol options

Creating an IIS Web Application

- The demo project must run in IIS 8
  - You can't just run it using Ctrl+F5 in Visual Studio!

- Follow these steps, to create a Web Application in IIS 8:
  - Open File Explorer and go to `C:\inetpub\wwwroot` folder
  - Create a new sub-folder named `SocketServer`
  - Start the Internet Information Services Manager console (the easiest way to do this is to search for `inetmgr` in Windows)
  - In the Internet Information Services Manager console:
    - In the left-hand side of the console, expand the Sites node. Right-click the Default Web Site node, and then click Add Application
    - In the Add Application dialog box, create an alias named `SocketServer` for the `c:\inetpub\wwwroot\SocketServer` folder

Understanding the Server Code

- Run Visual Studio as an Administrator
  - Then open `HelloWebSockets.sln`
- Take a look at `SocketServer.cs`
  - `SocketServer` is an HTTP handler class (i.e. it implements the `IHttpHandler interface`)
  - There's a lot of detail - the comments explain what's happening
  - Feel free to skim-read!
- Also take a look at `Web.config`
  - Associates the class: `SocketServer`
  - With the following URL: `SocketServer/SocketServer.ashx`

Publishing the Project

- You must publish the project to IIS
  - We've already configured the project's deployment details, to deploy the app to the `SocketServer` folder in IIS
- To publish for the project, follow these steps:
  - Right-click the project in Visual Studio
  - … and select the Publish option in the popup menu

3. Defining a Web Sockets Client

- The WebSocket interface
- Checking for Web Sockets support
- Opening a connection
- Handling Web Sockets events
- Sending data to the server
- Receiving data from the server
- Closing a connection
- Running the demo client

The WebSocket Interface

- The W3C defines a client-side `webSocket` interface

```
interface WebSocket
{

  readonly attribute DOMString URL;

  const unsigned short CONNECTING = 0;
  const unsigned short OPEN = 1;
  const unsigned short CLOSED = 2;
  readonly attribute unsigned short readyState;
  readonly attribute unsigned long bufferedAmount;

  attribute Function onopen;
  attribute Function onmessage;
  attribute Function onclose;
  attribute Function onerror;

  void close();
  void send(in DOMString data);
};
```

Checking for Web Sockets Support

- The first step in a client application is to check whether the browser supports Web Sockets
  - Test for the existence of the `WebSocket` property on the window object

- Example:

```
if (window.WebSocket) {
  // Web Sockets are supported …

}
```

Opening a Connection

- To open Web Sockets connection to a server:
  - Create a `webSocket` object, specifying the URL to connect to
- For an unsecure Web Socket connection:
  - Use the `ws://` protocol
  - Default port is 80
- For a secure Web Socket connection:
  - Use the `wss://` protocol
  - Default port is 443
- Example:

```
var aSocket = new WebSocket("ws://myserver.com/myapp");
```

Handling Web Sockets Events

- The Web Sockets API is asynchronous, because:
  - Establishing a connection is relatively time-consuming
  - Once a connection has been established, data can arrive at any time over the connection

- The client should handle events as follows:

```
aSocket.onopen    = function(e) { … };
```

```
aSocket.onmessage = function(e) { … };
```

```
aSocket.onclose   = function(e) { … };
```

```
aSocket.onerror   = function(e) { … };
```

Sending Data to the Server

- The client can send data to the server at any time
  - Invoke `send()` on the `webSocket` object

```
aSocket.send(someData);
```

- Types of data supported:
  - Text
  - Binary data
  - An array

Receiving Data from the Server

- The client can receive data from the server at any time
  - Handle the `message` event
- The event argument has two properties: `type` and `data`
  - If the type is `"binary"`, the web socket object has a `binaryType` property that indicates whether the data is a `"blob"` or `"arrayBuffer"`

```
aSocket.onmessage = function(e) {
  if (e.type == "text") {
    … handle text data …
  }
  else {
    if (aSocket.binaryType == "blob") …
    else if (aSocket.binaryType == "arrayBuffer") …
  }
};
```

Closing the Connection

- The client should close the Web Sockets connection when it wants to terminate its conversation with the server
  - Invoke `close()` on the `WebSocket` object
  - You can specify optional parameters: code and reason

```
aSocket.close(42, "Client closed connection normally");
```

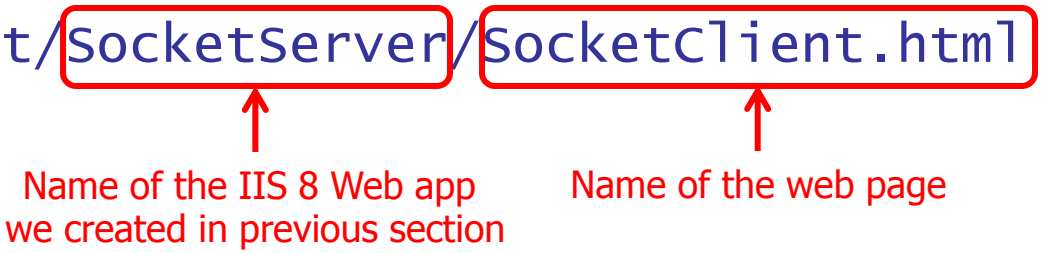
- When the connection has been closed, the `close` event occurs
  - The event argument has 3 properties: `wasClean`, `code`, `reason`

```
aSocket.onclose = function(e) {
  if (!e.wasClean) {
    alert("Connection closed with code " + e.code +
          " [reason: " + e.reason + "]");
  }
};
```

Running the Demo Client

- Our demo has a client web page, `SocketClient.html`
  - Open it in a browser using the following URL:

`http://localhost/SocketServer/SocketClient.html`



- The client should successfully establish a web socket connection with the server
  - You should be able to send data to the server, and see it echoed back to you

- Open another browser window and repeat the above steps
  - The server should echo messages to all active clients

Any Questions?