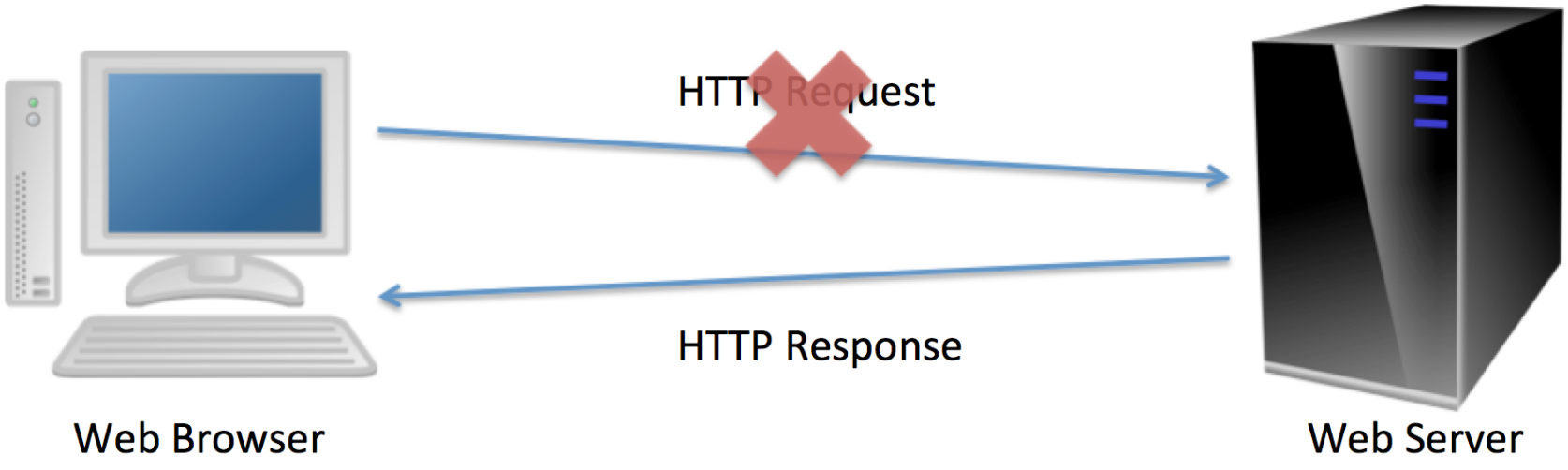
The WebSocket Protocol

Sistemas Distribuídos 2014/2015

How to send data in real-time from the server to the client?



Two-way communication on the Web

- How may a server send content to a browser without the client making an explicit request?
- WebSockets provide two-way communication through a single TCP socket over the Web.
- This is an advance over conventional HTTP for real-time applications (*e.g.*, instant messaging, games).
- Replaces technologies such as COMET.

How it was done before

Historically, creating web applications that need bidirectional communication required HTTP polling of the server.

Problem 1: Server connections

The server must hold a number of different TCP connections (typically one for sending information to the client and a new one for each incoming message).

Problem 2: Header overhead

There is a lot of overhead related to the HTTP headers on each message.

Problem 3: Client-side bookkeeping

The client must maintain a mapping between outgoing connections and the incoming connection, in order to keep track of replies.

Old-school solutions

Polling The client periodically makes a request to the server, to check for updates, for example by setting `<meta http-equiv="refresh" content="5">`

Long Polling The server tries to "hold open" (without replying to) each HTTP request, responding only when there are events to deliver. In this way, there is always a pending request to which the server can reply for the purpose of delivering events as they occur.

Streaming The server keeps a request open indefinitely, that is, it never terminates the request or closes the connection, even after it pushes data to the client.

WebSockets

A simpler solution is to use a single TCP connection for traffic in both directions. The WebSocket protocol provides this.

Initial handshake

The handshake from the client:

```
GET /chat HTTP/1.1
Host: server.myamazingservice.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://myamazingservice.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

The handshake from the server:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

Each side may now, independently, send data to the other side.

Design philosophy

- Conceptually, WebSocket is built on top of regular TCP, while adding naming, framing, handshakes and addressing.
- Otherwise, nothing is added by the WebSocket protocol. The primary goal is to expose raw TCP as closely as possible (send and receive messages) while addressing the constraints of the Web.
- The only relation to HTTP is that the initial handshake is taken by HTTP servers as a request for upgrading to a distinct protocol.
- By default, it uses the same ports as HTTP (80 for regular WebSockets and 443 for WebSockets over TLS).

javax.websocket.server.ServerEndpoint

```
import javax.websocket.*;
import javax.websocket.server.ServerEndpoint;

@ServerEndpoint(value = "/chat")
public class ChatWebSocket {
    ...
}
```

This class level annotation declares that the `ChatWebSocket` class is a web socket endpoint that will be deployed and made available in the URI-space of a web socket server (no need to do it in `web.xml`).

Listening to messages (and events)

Annotate a method as `@OnMessage`:

```
@OnMessage
public void incoming(String message) {
    // we should never trust the client, and sensitive HTML
    // should be replaced with &lt; &gt; &quot; &amp;
    doSomethingWith(message);
}
```

The method is invoked whenever there's a message from a client.

Use also annotations `@OnOpen` and `@OnClose`:

```
@OnOpen
public void start(Session session) {
    this.session = session;
    connections.add(this);
    doSomethingWhenOpen();
}

@OnClose
public void end() {
    connections.remove(this);
    doSomethingWhenClosed();
}
```

Pushing content to the client

The `javax.websocket.Session` interface contains the following method:

```
getBasicRemote();
```

This method returns a `RemoteEndpoint` object, representing the peer of this conversation, that is able to send messages to the peer.

```
try {
    synchronized (client) {
        client.session.getBasicRemote().sendText(message);
    }
} catch (IOException e) {
    ...
}
```

Implement the client (HTML+JavaScript)

Clients (*i.e.*, browsers) take HTML, as usual, with some JavaScript to handle the WebSocket. A simple chat page:

```
<!DOCTYPE html>
<html>
<head>
    <title>WebSocket Chat</title>
    <link rel="stylesheet" type="text/css" href="style.css">
    <script type="text/javascript">
        // JavaScript to handle the websocket, the input, and the output
    </script>
</head>
<body>
<noscript>JavaScript must be enabled for WebSockets to work.</noscript>
<div>
    <div id="container"><div id="history"></div></div>
    <p><input type="text" placeholder="type␣to␣chat" id="chat"></p>
</div>
</body>
</html>
```

Minimal JavaScript for handling WebSockets (1/3)

```
var websocket = null;

window.onload = function() { // URI = ws://10.16.0.165:8080/chat/chat
    connect('ws://' + window.location.host + '/chat/chat');
    document.getElementById("chat").focus();
}

function connect(host) { // connect to the host websocket servlet
    if ('WebSocket' in window)
        websocket = new WebSocket(host);
    else if ('MozWebSocket' in window)
        websocket = new MozWebSocket(host);
    else {
        writeToHistory('Get a real browser which supports WebSocket.');
        return;
    }

    websocket.onopen    = onOpen; // set the event listeners below
    websocket.onclose   = onClose;
    websocket.onmessage = onMessage;
    websocket.onerror   = onError;
}
```

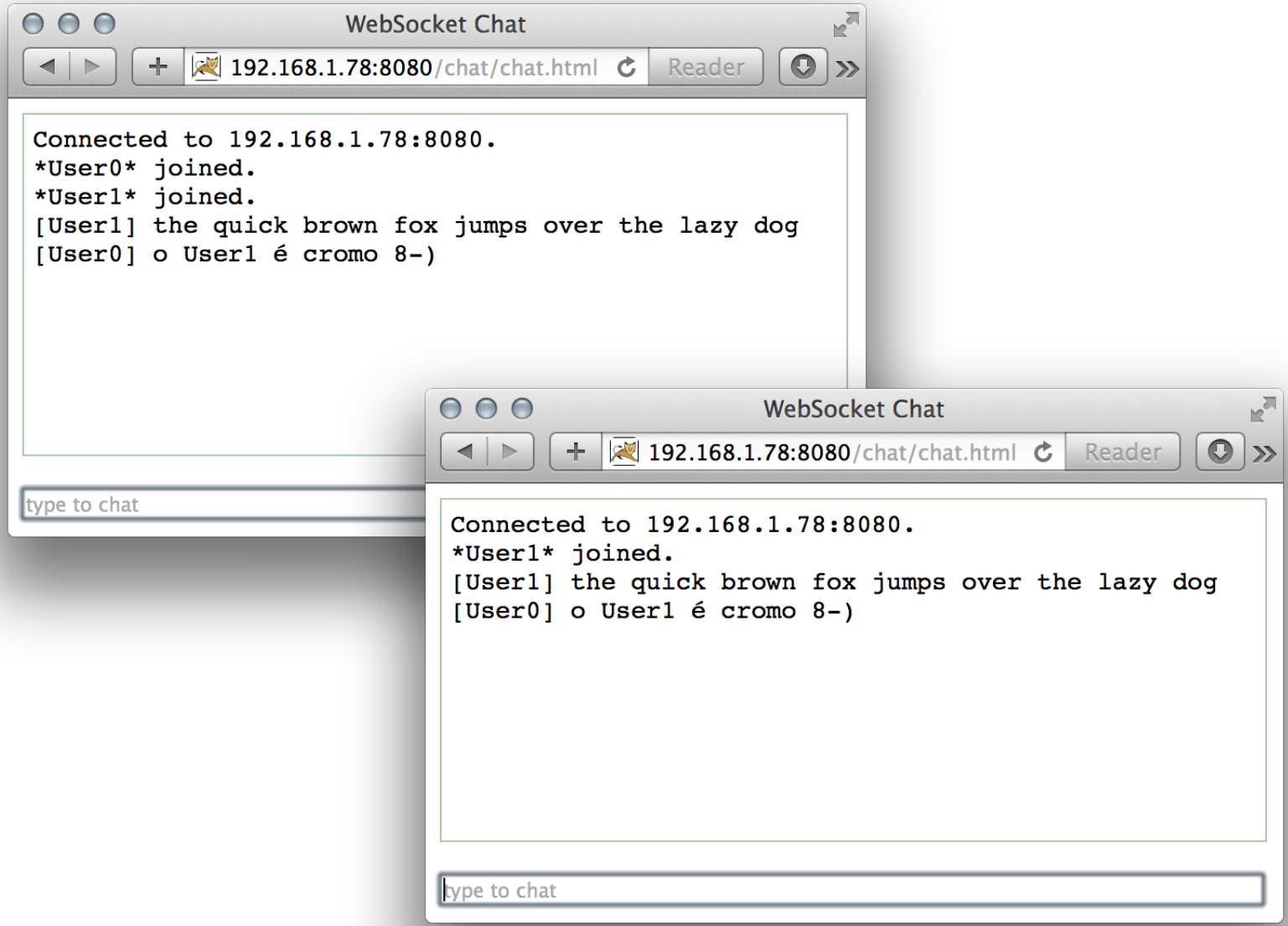
Minimal JavaScript for handling WebSockets (2/3)

```
function onOpen(event) {
    writeToHistory('Connected to ' + window.location.host + '.');
    document.getElementById('chat').onkeydown = function(key) {
        if (key.keyCode == 13)
            doSend(); // call doSend() on enter key
    };
}

function onClose(event) {
    writeToHistory('WebSocket closed.');
    document.getElementById('chat').onkeydown = null;
}

function onMessage(message) { // print the received message
    writeToHistory(message.data);
}

function onError(event) {
    writeToHistory('WebSocket error (' + event.data + ').');
    document.getElementById('chat').onkeydown = null;
}
```

Minimal JavaScript for handling WebSockets (3/3)

```
function doSend() {
    var message = document.getElementById('chat').value;
    if (message != '')
        websocket.send(message); // send the message
    document.getElementById('chat').value = '';
}

function writeToHistory(text) {
    var history = document.getElementById('history');
    var line = document.createElement('p');
    line.style.wordWrap = 'break-word';
    line.innerHTML = text;
    history.appendChild(line);
    history.scrollTop = history.scrollHeight;
}
```

Two clients running



The complete ServerEndpoint (1/3)

```
package chat;

import java.io.IOException;
import java.util.Set;
import java.util.concurrent.CopyOnWriteArraySet;
import java.util.concurrent.atomic.AtomicInteger;

import javax.websocket.*;
import javax.websocket.server.ServerEndpoint;

@ServerEndpoint(value = "/chat")
public class ChatWebSocket {
    private static final String PREFIX = "User";
    private static final AtomicInteger connectionIds
            = new AtomicInteger(0);
    private static final Set<ChatWebSocket> connections
            = new CopyOnWriteArraySet<>();
    private final String nickname;
    private Session session;

    public ChatWebSocket() {
        nickname = PREFIX + connectionIds.getAndIncrement();
    }
```

The complete ServerEndpoint (2/3)

```
@OnOpen
public void start(Session session) {
    this.session = session;
    connections.add(this);
    String message = "*" + nickname + "* joined.";
    broadcast(message);
}

@OnClose
public void end() {
    connections.remove(this);
    String message = "*" + nickname + "* disconnected.";
    broadcast(message);
}

@OnMessage
public void incoming(String message) {
    // we should never trust the client, and sensitive HTML
    // should be replaced with &lt; &gt; &quot; &amp;
    broadcast("[" + nickname + "] " + message);
}
```

The complete ServerEndpoint (3/3)

```
    private static void broadcast(String msg) {
        for (ChatWebSocket client : connections) {
            try {
                synchronized (client) {
                    client.session.getBasicRemote().sendText(msg);
                }
            } catch (IOException e) {
                connections.remove(client);
                try {
                    client.session.close();
                } catch (IOException e1) {
                    // Ignore
                }
                String message = String.format("* %s %s",
                        client.nickname, "has been disconnected.");
                broadcast(message);
            }
        }
    }
}
```

Bibliography

- The WebSocket Protocol, RFC6455, http://tools.ietf.org/html/rfc6455
- The WebSocket API, W3C, http://www.w3.org/TR/websockets