Java and WebSockets

Petr Křemen, Martin Ledvinka

KBSS

Winter Term 2020

Contents

Basics

XMLHttpRequest

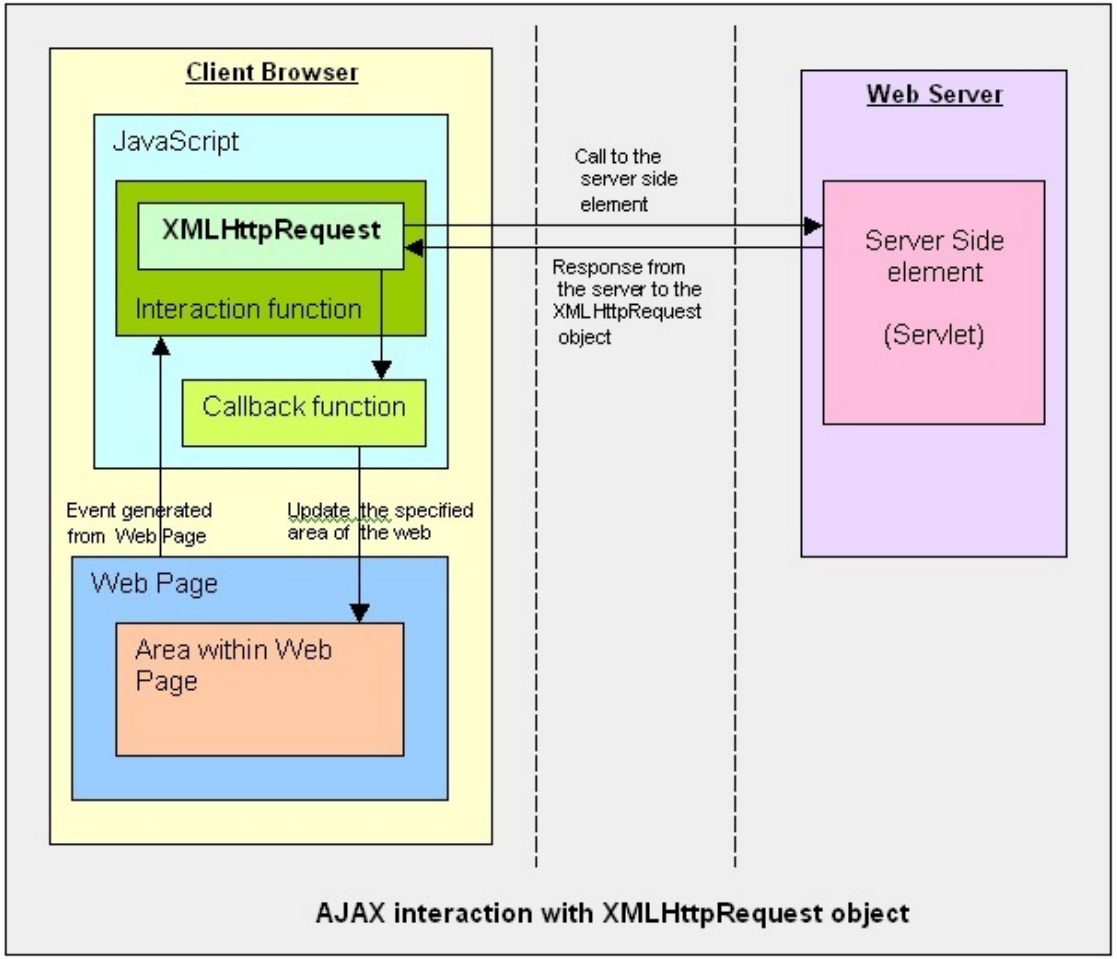


Figure: Source: `https://devcentral.f5.com/articles/social-media-abcs-x-is-for-xmlhttprequest`

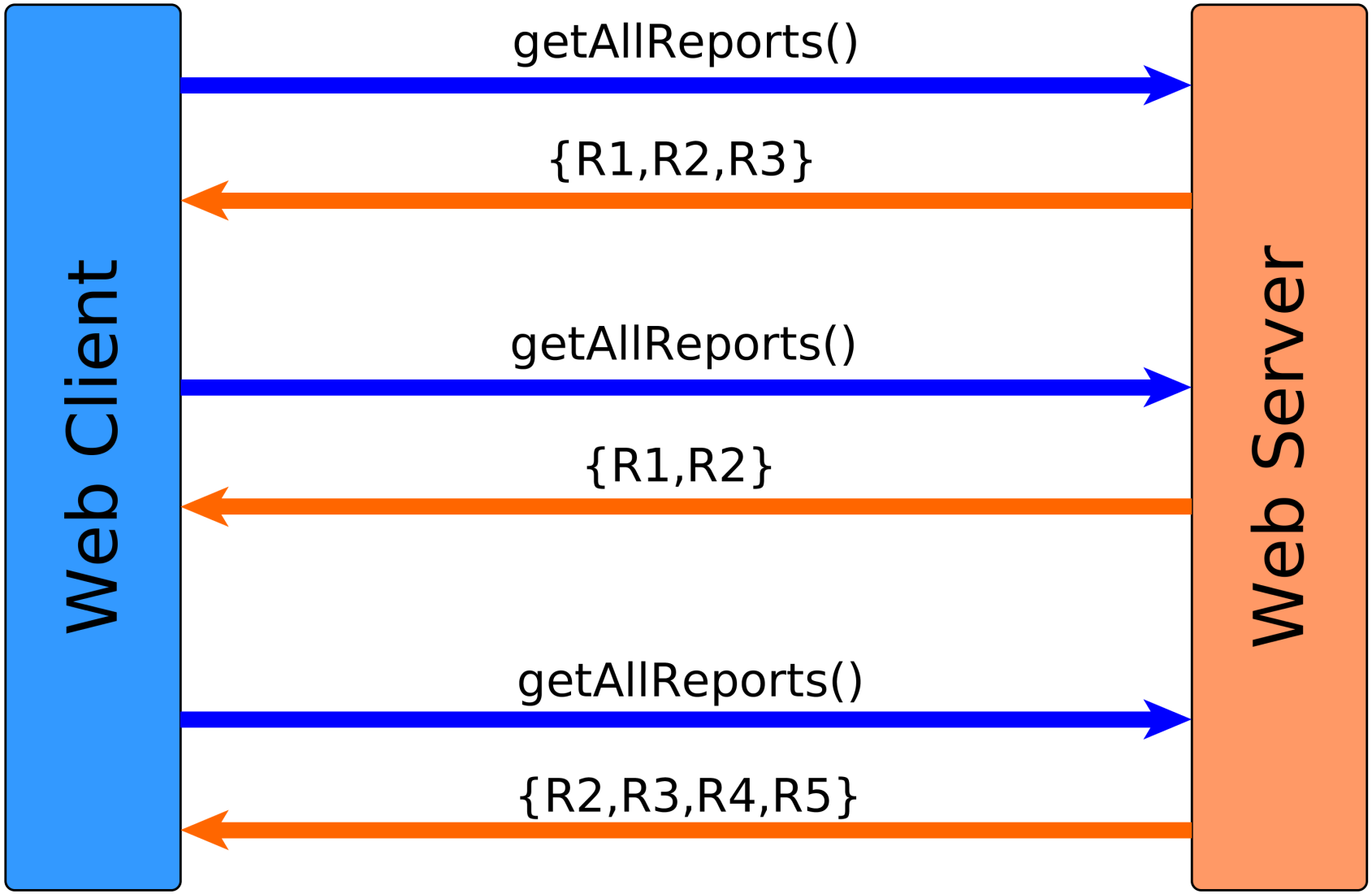
The Story so Far

- We have learned technologies to create an application on server-side
- To communicate with a client we use exclusively the HTTP(S) protocol, typically through REST

Problem

What to do when new data on server appear and the client does not know?

Simple Solution Using HTTP



...

Short Polling

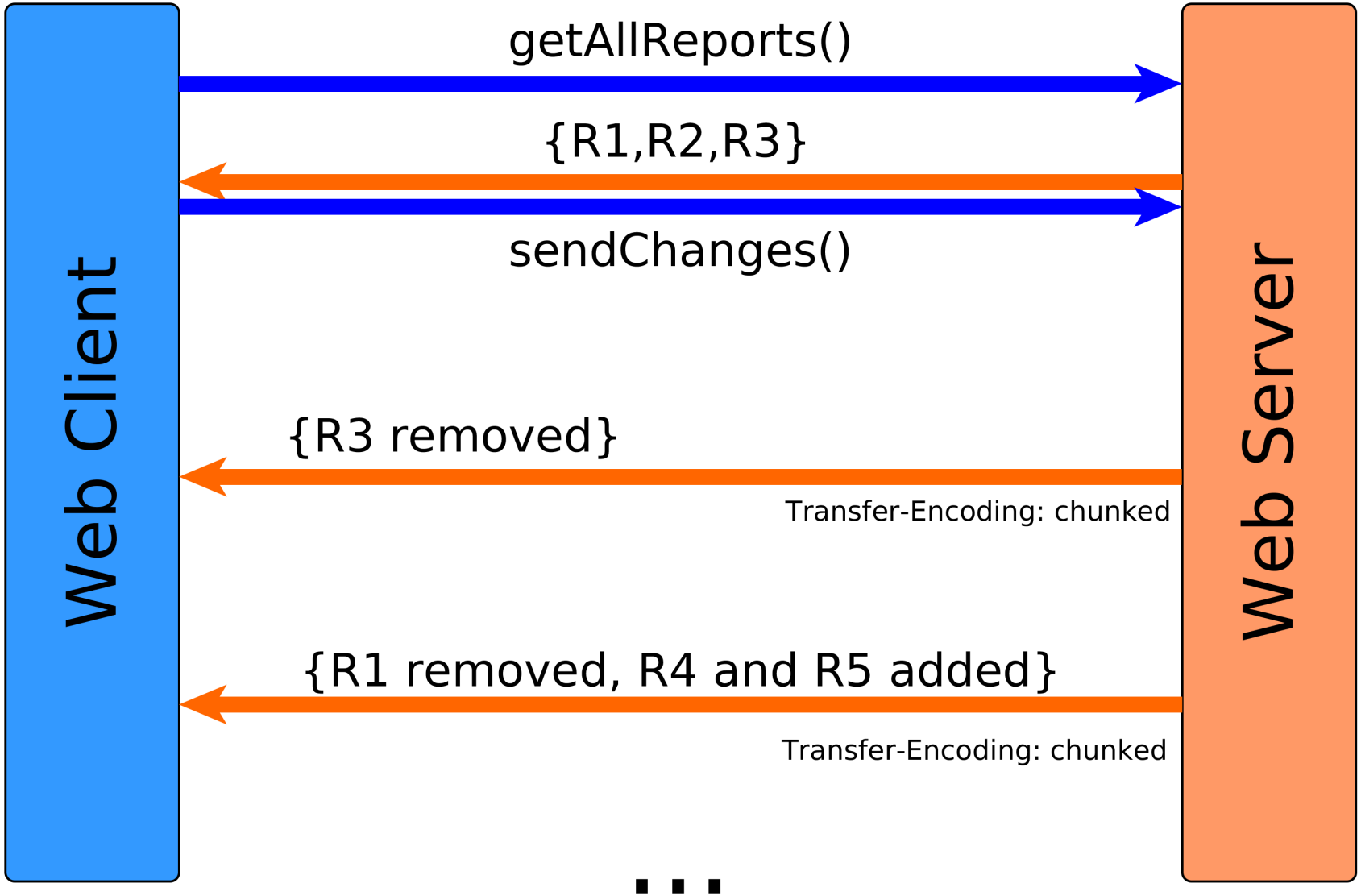
- Very easy to implement
  - Just repeat a request with a certain interval
- Resource intensive (a lot of back and forth communication)
- Extremely poor scalability

Better Solution Using HTTP – AJAX push (Long Poll)

Long Polling

- Based on the *Comet* programming model
- Client opens a request with a long timeout
- Server delays the response until it has data to respond with (or a timeout occurs)
- Simple on client side
- Difficult on server side – need to keep track of open connections
- But less HTTP communication overhead

Better Solution Using HTTP – AJAX push (Streaming)



Streaming

- Based on the *Comet* programming model
- Two main techniques:
  - Forever iframe – `iframe` with `src` pointing to the server, server responds with new `script` tag every time a new event occurs
  - XMLHttpRequest – Multipart request, each new event fires a *onreadystatechange* event on the request on client
- Server side - need to keep track of open connections
- Virtually no error handling for forever iframes
- Multipart XMLHttpRequest non-standard and obsolete nowadays

... but Still Not Perfect

- The client has to create a new HTTP connection for each communication type (getReports, getUsers, chat)
- HTTP headers have to be sent back and forth for each client side
- The client has to understand/parse low-level HTTP chunks

WebSockets

Represent a systematic solution to HTTP client-server peculiarities and provide a symmetric model for client-server communication.

WebSockets

WebSocket Protocol

- TCP-based protocol with minimum overhead
- HTTP handshake for compatibility
- `ws://` (port 80) or `wss://` (port 443)
- Text or binary messages
- Full duplex

WebSocket vs. HTTP

HTTP

- Designed for "web pages" not "interactive web applications"
- Traditional request-response model
- Intensive client-server communication – significant overhead (HTTP headers)

Web Sockets

- Bi-directional, full-duplex, real-time,
- Low-latency client/server communications on top of TCP/IP
- $\in$ Java EE 7

Web Socket Handshake

```
GET http://server.org/wsendpoint
  HTTP/1.1
Host: server.org
Connection: Upgrade
Upgrade: websocket
Origin: http://server.org
Sec-WebSocket-Version: 13
Sec-WebSocket-Key:
  GhkZiCk+0/91FXIbUuRlVQ==
Sec-WebSocket-Extensions:
  permessage-deflate;
  client_max_window_bits
```

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: upgrade
Sec-WebSocket-Accept:
  jpwu9a/SXDrsoRR26Oa3JUEFchY=
Sec-WebSocket-Extensions:
  permessage-deflate;
  client_max_window_bits=15
...
```

Web Sockets in Java

Java API for WebSocket (JSR-356)

Annotations on POJOs to interact with WebSocket lifecycle events

Interfaces to implement to interact with WebSocket lifecycle events

Integration with other Java EE technologies – EJB, CDI

JSR-356 Example

```
@ServerEndpoint("/actions")
public class WebSocketServer {

    @OnOpen
    public void open(Session session) { ... }

    @OnClose
    public void close(Session session) { ... }

    @OnError
    public void onError(Throwable error) { ... }

    @OnMessage
    public void handleMessage(String message, Session session) {
        // actual message processing
    }
}
```

JavaScript Side Example

```
const socket = new WebSocket("ws://server.org/
    wsendpoint");
socket.onmessage = onMessage;

function onMessage(event) {
    const data = JSON.parse(event.data);
    if (data.action === "addMessage") {
        ...
        // actual message processing
    }
    if (data.action === "removeMessage") {
        ...
        // actual message processing
    }
}
```

Other Options

- Spring has wide support through custom annotations - `spring-websocket` module
- ReactJS has `react-websocket` module (listener to WebSocket Events)

Sample Application – Chat

https://goo.gl/MQMWBf

Sample Application – Chat Monitoring

- Open Developer Tools in Google Chrome
- Navigate to the web site
- Open tab "Network" and select the request "actions" (chat)
- Select the subtab "Messages" and you can track the WebSocket communication

The End

Thank You

References

Reverse Ajax https://www.ibm.com/developerworks/library/wa-reverseajax1/index.html

RFC 6455 - The WebSocket Protocol https://tools.ietf.org/html/rfc6455

JSR 356: Java API for WebSocket https://jcp.org/en/jsr/detail?id=356

Java EE 7: Building Web Applications with WebSocket, JavaScript and HTM http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/HomeWebsocket/WebsocketHome.html

Spring Support for WebSocket http://docs.spring.io/spring/docs/current/spring-framework-reference/html/websocket.html