Developing JAX-RS Web Application Utilizing SSE and WebSocket

Romain Grecourt
Jitendra Kotamraju
Marek Potociar

Oracle

@gf_jersey

Agenda

- About This Lab
- Intro to the Used Technologies
- Lab Exercises
- Getting Started
- Resources
- Hands on …

Bootstrapping The Lab

- Get the…
  - HOL content
    - https://github.com/jersey/hol-sse-websocket
  - Maven 3.0.4
  - GlassFish 4.0-b57
  - JDK 7u09
  - NetBeans 7.2.1
    - Follow Appendix in lab-guide.pdf to register GlassFish in NetBeans

Once Bootstrapped…

- DO
  - Follow the lab guide – exercises are self-paced
  - Raise your hand if you get stuck – we are here to help
  - try to understand the code
- DON’T
  - just blindly copy-paste

Technologies Used…

- Jersey / JAX-RS 2.0
  - Server-sent events
- Tyrus / Java API for WebSocket
- JSON Processing

JAX-RS / Jersey

JAX-RS

- Java API for RESTful Web Services
  - Annotation-based API for exposing RESTful web services
  - Maps HTTP requests to Java method invocations
  - Part of Java EE
- New in JAX-RS 2.0
  - Client API
  - Filters & Interceptors
  - Asynchronous processing
  - Server-side content negotiation, …

JAX-RS Resources

```
public class OrderResource {

    public List<Order> getOrders() { … }



    public Order getOrder(String orderId,
                          String from) { … }


    public CustomerResource customer(…) { … }

    …
}
```

JAX-RS Resources

```
@Path("orders")
@Produces("application/xml")
@Consumes("application/xml")
public class OrderResource {
    @GET
    public List<Order> getOrders() { … }

    @GET
    @Path("{id}")
    public Order getOrder(@PathParam("id") String orderId,
                          @HeaderParam("From") @Default("unknown") String from) { … }

    @Path("{id}/customer")
    public CustomerResource customer(…) { … }

    …
}
```

More in JAX-RS 1.x

- Injectable information
  - Request, HttpHeaders, UriInfo, …
- Advanced HTTP response construction
  - Response, ResponseBuilder
- Message content handlers (a.k.a entity providers)
  - MessageBodyReader & MessageBodyWriter
- Error handlers
  - ExceptionMapper
- Other APIs aiding HTTP request/response processing

JAX-RS 2.0 Client API

- Accessing HTTP-based (RESTful) services from Java
- Low-level, fluent API
- Synchronous & Asynchronous (Future & callback) programming models
- Re-using server-side components and concepts
  - Response
  - Message content handlers
  - Reader/Writer Interceptors
  - Filters (conceptually)

Main Client API Components

- ClientFactory
  - Bootstrapping (analogy of RuntimeDeleate)
- Client
  - Main API entry point
  - Connection management, configurable
- Web Target
  - URI, URI template abstraction (“glorified URI”)
  - Configurable

JAX-RS 2.0 Client API

```
Client client = ClientFactory.newClient();

WebTarget ordersTarget = client.target("http://example.com/eshop/orders");

WebTarget orderTarget = ordersTarget.path("{id}");

Order order = orderTarget.resolveTemplate("id", "1234")
                         .request("application/xml")
                         .get(Order.class);
```

```
Order newOrder = new Order(…);
Response response = ordersTarget.request("text/plain")
                                 .post(Entity.xml(newOrder));



if (response.getStatus() == 200) {
    String orderId = response.readEntity(String.class);
    Link paymentLink = response.getLink("payment");

    client.target(paymentLink).request()…
}
```

Server-Sent Events

```
: an example of a SSE event
id: 1
event: text-message
data: Hello, this is a
data: multi-line message.
<blank line>
```

Server-sent Events in Jersey

- Server side
  - OutboundEvent
  - EventChannel
  - SseBroadcaster
  - BroadcasterListener
- Client side
  - InboundEvent
  - EventSource
  - EventListener

SSE Server-side

```
@Path("message/stream")
public class MessageStreamResource {
    private static SseBroadcaster broadcaster = new SseBroadcaster();

    @GET
    @Produces(SseFeature.SERVER_SENT_EVENTS)
    public EventOutput getMessageStream() {
        final EventOutput eventOutput = new EventOutput();
        broadcaster.add(eventOutput);
        return eventOutput;
    }

    …
```

SSE Server-side

```
    …

    private static AtomicLong nextMessageId = new AtomicLong(0);

    @PUT
    @Consumes(MediaType.APPLICATION_JSON)
    public void putMessage(Message message) {
        OutboundEvent event = new OutboundEvent.Builder()
                .id(String.valueOf(nextMessageId.getAndIncrement()))
                .mediaType(MediaType.APPLICATION_JSON_TYPE)
                .data(Message.class, message)
                .build();

        broadcaster.broadcast(event);
    }
}
```

SSE Client-side

```
EventSource events = new EventSource(target.path("message/stream")) {
    @Override
    public void onEvent(InboundEvent event) {
        String name = event.getName();
        Message message = event.getData(Message.class);
        display(name, message);
    }
};

…

events.close();
```

JAX-RS / Jersey

- More at Devoxx
  - The Present and the Future of JAX-RS and Jersey (Thu 16:40, Room 8)
  - JAX-RS 2.0 Status & Directions (Thu 19:00, BOF 2)
- On the Web
  - Specification project: http://jax-rs-spec.java.net
  - Implementation project: http://jersey.java.net
  - Twitter: @gf_jersey

WebSocket / Tyrus

Interactive Web Sites

- Chat, Streaming quotes, games, …
- HTTP is request/response
- Forcing persistence on HTTP
  - Just to know server has data
  - Long Polling, Streaming, Comet/Ajax ..
- Complex, Inefficient, Wasteful

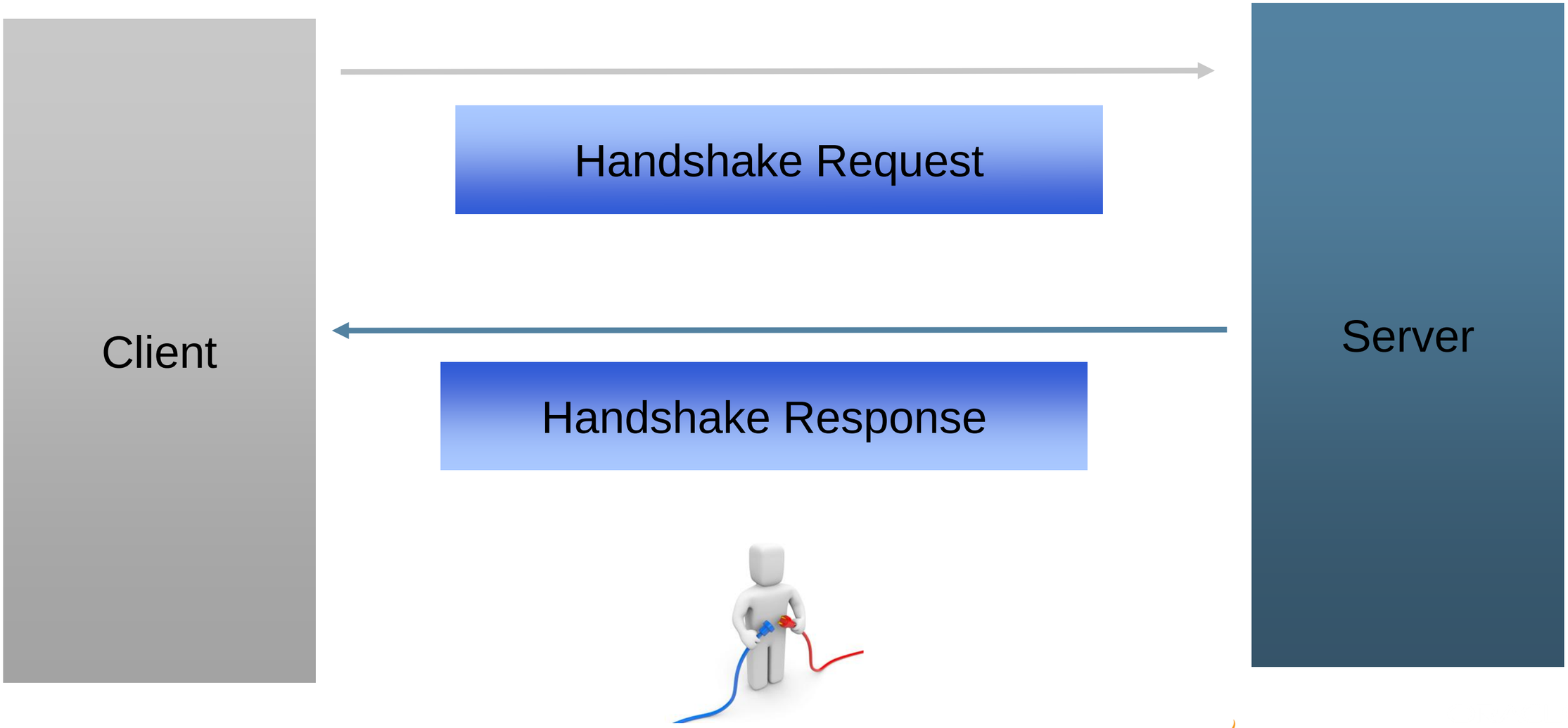
WebSocket to the Rescue

- TCP based, bi-directional, full-duplex messaging
- Originally proposed as part of HTML5
- IETF-defined **Protocol**: RFC 6455
  - Handshake
  - Data Transfer
- W3C defined **JavaScript API**
  - Candidate Recommendation

What’s the basic idea ?

- Establish a connection (Single TCP connection)
- Send messages in both directions (Bi-directional)
- Send message independent of each other (Full Duplex)
- End the connection

Establish a connection



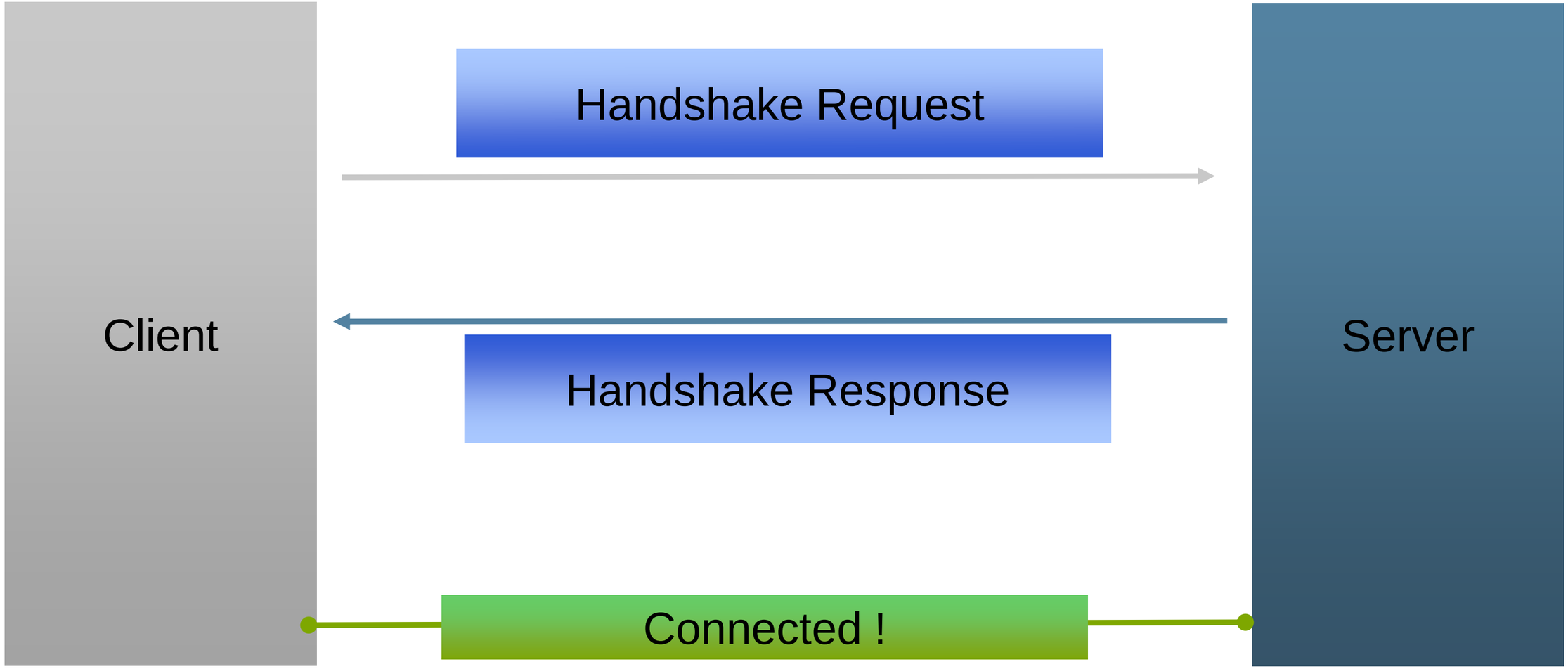
Handshake Request

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

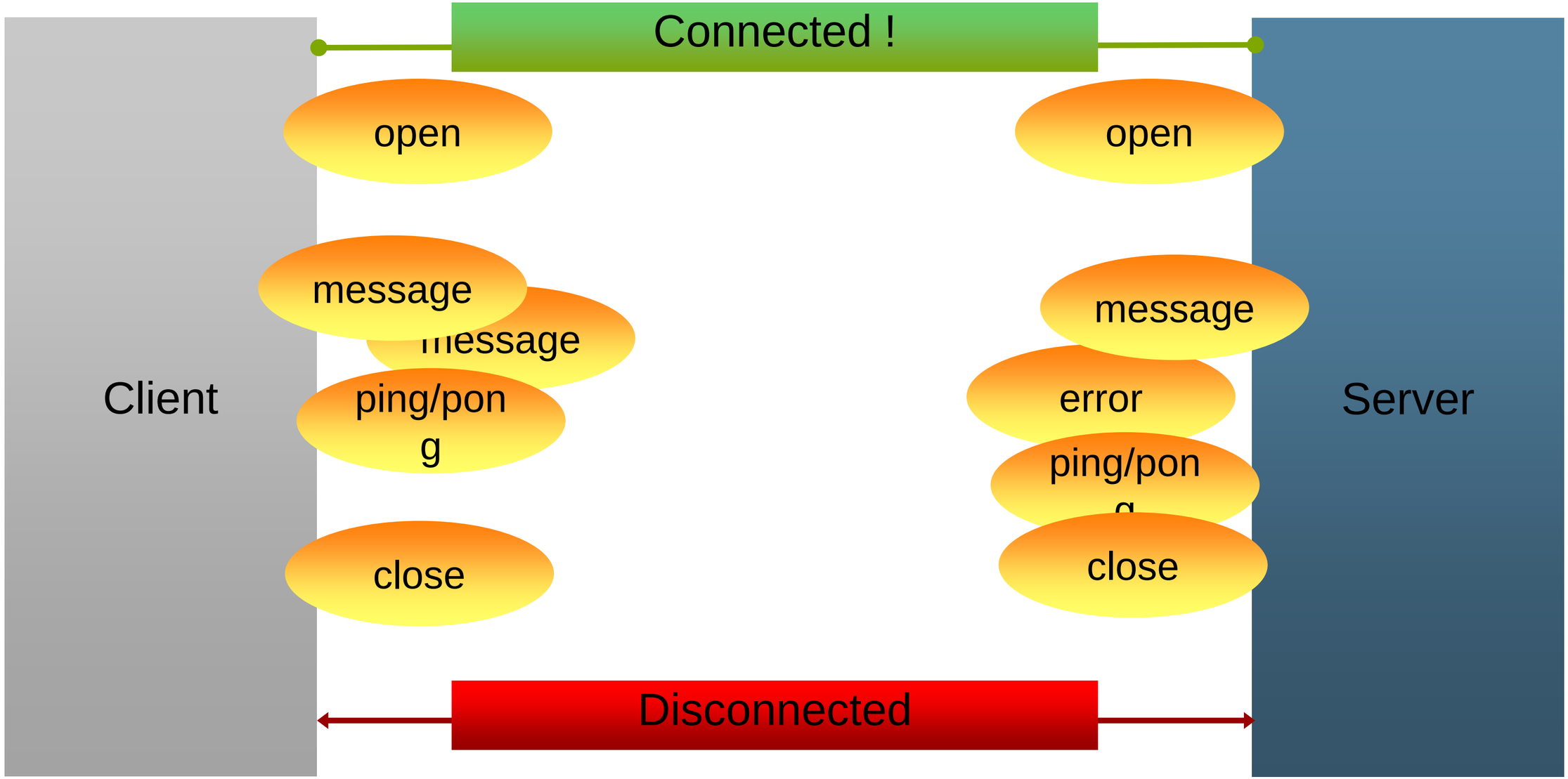
Handshake Response

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

Establishing a Connection



WebSocket Lifecycle



WebSocket wire messages

- Capture traffic on loopback

Filter: http

Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
11	9.489449000	::1	::1	HTTP	648	GET /HelloWebSocket/ HTTP/1.1
13	9.491601000	::1	::1	HTTP	2134	HTTP/1.1 200 OK (text/html)
18	9.669322000	::1	::1	HTTP	501	GET /HelloWebSocket/echo HTTP/1.1
20	9.669489000	::1	::1	HTTP	543	GET /favicon.ico HTTP/1.1
22	9.670298000	::1	::1	HTTP	205	HTTP/1.1 101 Switching Protocols
24	9.671010000	::1	::1	HTTP	1624	HTTP/1.1 404 Not Found (text/html)
26	12.411987000	::1	::1	WebSocket	98	WebSocket Text [FIN] [MASKED]
28	12.413161000	::1	::1	WebSocket	108	WebSocket Text [FIN]
30	13.011122000	::1	::1	WebSocket	98	WebSocket Text [FIN] [MASKED]
32	13.013172000	::1	::1	WebSocket	108	WebSocket Text [FIN]

WebSocket Protocol Summary

- Starts with HTTP handshake
- Data transfer
  - Text/Binary frames
  - Ping/Pong control frames for keep-alive
  - Data frames don’t have HTTP overhead
    - No headers/cookies/security/metadata
  - Close frame
- Full duplex and bi-directional

The Web Sockets API

- Web Sockets API defines WebSocket javascript interface
- Event handlers for onopen(), onmessage(), onclose(), onerror()
- Send a String, Blob, ArrayBuffer using send()
- Supports sub protocols

JSR 356 Specification

- Standard Java API for creating WebSocket Applications
- Transparent Expert Group
  - jcp.org/en/jsr/detail?id=356
  - java.net/projects/websocket-spec
- Now: Early Draft Review
- December: Public Draft Review
- Will be in Java EE 7

JSR 356: Reference Implementation

- Tyrus: java.net/projects/tyrus
- Originated from WebSocket SDK
  - java.net/projects/websocket-sdk
- Works in stand alone, Java EE deployments
- Integrated in GlassFish 4 Builds

API Features

- Create WebSocket Client/Endpoints
  - Annotation-driven (@ServerEndpoint)
  - Interface-driven (Endpoint)
- Integration with Java EE Web container
  - CDI, Security, HttpSession etc.

Hello World

```
import javax.net.websocket.annotations.*;

@ServerEndpoint("/hello")
public class HelloBean {

    @OnMessage
    public void hello(String str) {
        // Receiving a "Hello World"
    }
}
```

Hello World

```
@ServerEndpoint("/hello")
public class HelloBean {

    @OnMessage
    public String hello(String str) {
        // Echoing "Hello World”
        return str;
    }
}
```

Custom Objects

```
@ServerEndpoint(
      value     = "/hello”,
      encoders = {ServerHello.class},
      decoders = {ClientHello.class})
public class HelloBean {
    @OnMessage
    public ServerHello hello(ClientHello str) {
        …
    }
}
```

Lifecycle Events

```
@ServerEndpoint("/chat")
public class ChatBean {

    @OnOpen
    public void xxx(Session peer) {
    }

    @OnClose
    public void yyy(Session peer) {
    }


    @OnError
    public void zzz(Session peer) {
    }
```

Custom Decoders

- Binary(ByteBuffer/InputStream), Text (String/Reader) messages

```
public class X implements Decoder.TextStream<JsonObject> {

    public JsonObject decode(Reader io) {
        return new JsonReader(io).readObject());
    }

    public boolean willDecode(String string) {
        return true;    // Only if can process the payload
    }

}
```

Custom Encoders

- Binary(ByteBuffer/OutputStream), Text (String/Writer) messages

```
public class X implements Encoder.TextStream<JsonObject> {

    public void encode(JsonObject obj, Writer io) {
        new JsonWriter(io).write(obj);
    }
}
```

Tyrus / WebSocket

- More at Devox
  - JSR 356 : Java API for WebSocket (Fri 10:45 am, Room 5)
- On The Web
  - Specification Project: http://websocket-spec.java.net
  - Implementation: http://tyrus.java.net

JSON Processing API

Standard JSON API

- Parsing/Processing JSON
- Data binding : JSON text <-> Java Objects
- Two JSRs (similar to JAXP and JAXB)
  - Processing/Parsing – Java EE 7
  - Binding – Java EE 8

Java API for Processing JSON

- Streaming API to produce/consume JSON
  - Similar to StAX API in XML world
- Object model API to represent JSON
  - Similar to DOM API in XML world
- Aligns with Java EE 7 schedules
  - PR in Dec
- EG (Oracle, RedHat, Twitter, 3 individual members)
  - Also, user community !

JSR-353: Java API for Processing JSON

- JsonReader – reads JsonObject/JsonArray from i/o

```
try(JsonReader reader = new JsonReader(io)) {
    JsonObject jsonObj = reader.readObject();
}
```

- JsonWriter – writes JsonObject/JsonArray to i/o

```
try(JsonWriter writer = new JsonWriter(io)) {
    writer.writeObject(jsonObj);
}
```

JSR-353: Java API for Processing JSON

- JsonBuilder – builds JSON object/array from scratch

```
JsonArray arr = new JsonBuilder()
    .beginArray()
        .beginObject()
            .add("type", "home”)
            .add("number", "212 555-1234")
        .endObject()
        .beginObject()
            .add("type", "fax”)
            .add("number", "646 555-4567")
        .endObject()
    .endArray()
.build();
```

```
[
    {
        "type": "home”,
        "number": "212 555-1234"
    },
    {
        "type": "fax”,
        "number": "646 555-4567"
    }
]
```

JSON Processing

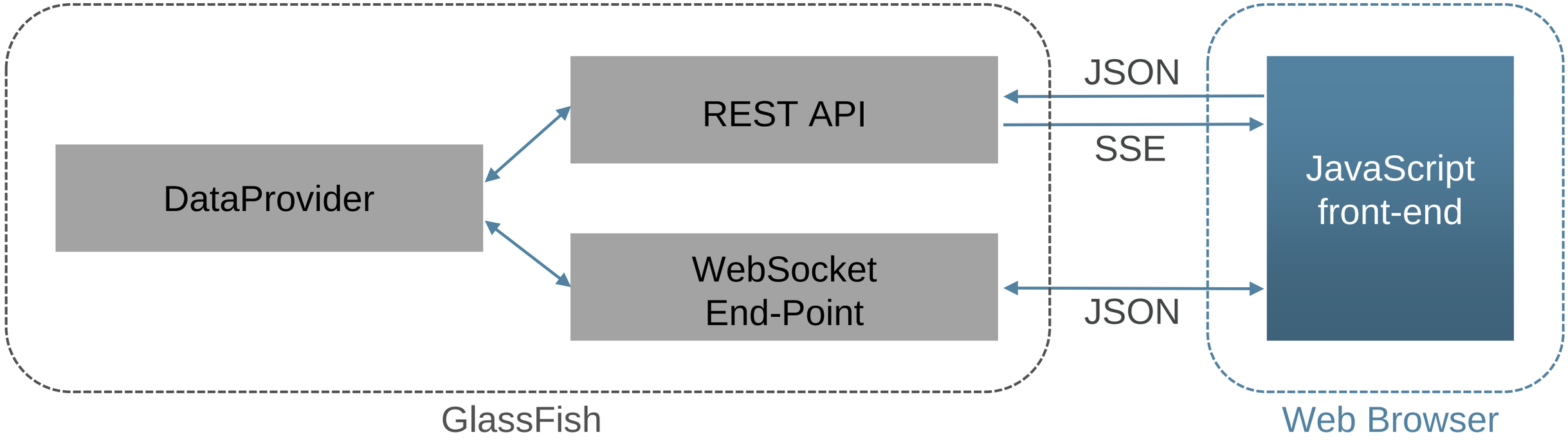
- More at Devoxx
  - JSR 353: Java API for JSON Processing (Thu, 3:10 pm, Room 8)
- Projects
  - Specification Project - http://json-processing-spec.java.net
  - RI Project - http://jsonp.java.net
- Latest Javadoc
  - http://json-processing-spec.java.net/nonav/releases/1.0/edr/javadocs/index.html

Lab Overview

Lab Exercises

- Drawing Board web application:
  - Exercise 1: Exposing RESTful API
  - Exercise 2: Adding Server-Sent Events
  - Exercise 3: Adding Web Sockets
- Simple Drawing Board client:
  - Exercise 4: Implementing a Simple Java SSE Client

Drawing Board Application



Getting Started

- Get the HOL content from github:
  - https://github.com/jersey/hol-sse-websocket
- Open lab-guide.pdf
  - Look at Appendix to configure GlassFish in NetBeans
- Follow the instructions
  - If you get stuck, raise your hand

Additional Resouces

- Follow **@gf_jersey** on Twitter
- Jersey – http://jersey.java.net
  - Mailing list: users@jersey.java.net
  - Fork Jersey on GitHub: http://github.com/jersey
- Tyrus – http://tyrus.java.net
- JSON Processing – http://jsonp.java.net