JavaOne™

Programming Java WebSockets

David Delabassee - @delabassee
Java EE PM
Oracle

Agenda

- Introduction to WebSockets
- What are they for ?
- Delving into the APIs

Interactive Web Sites

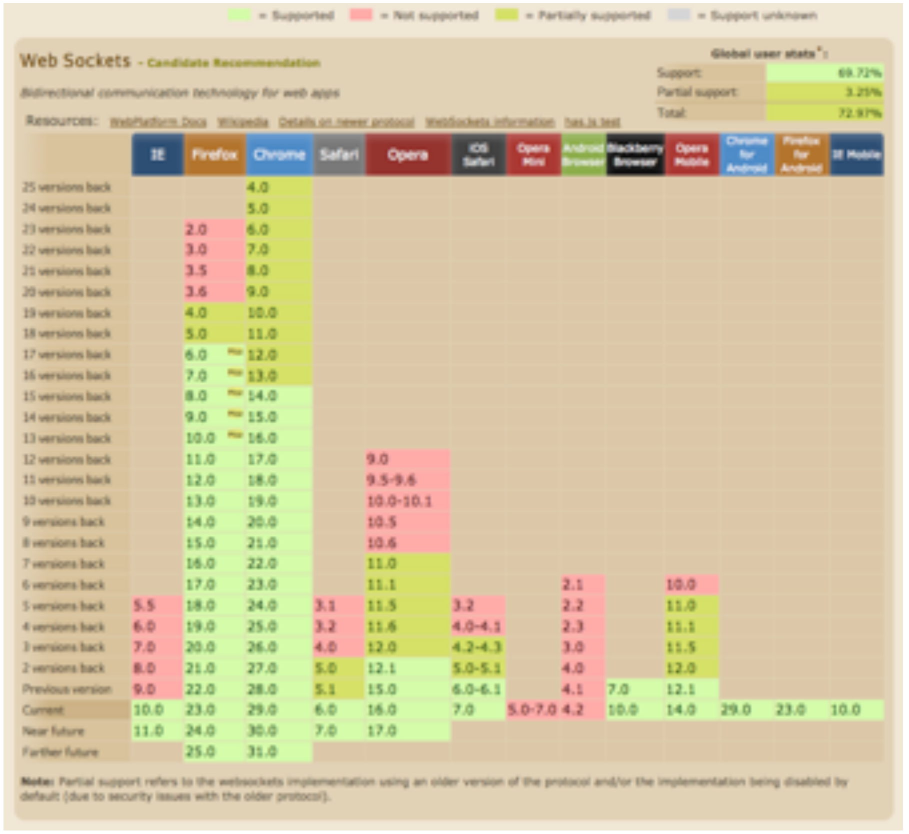
- Flavors of Server Push
  - Polling
  - Long polling
  - Comet/AJAX

- Complex, inefficient, wasteful

WebSockets to the rescue

- TCP based, bi-directional, full-duplex messaging
- IETF defined protocol: RFC 6455
- Part of HTML5
- W3C defined JavaScript API

But can I use WebSockets?



🟩 = Supported 🟥 = Not supported 🟨 = Partially supported ⬜ = Support unknown

Web Sockets - Candidate Recommendation

Bidirectional communication technology for web apps

Resources: WebPlatform Docs Wikipedia Details on newer protocol WebSockets information has.js test

Global user stats*:

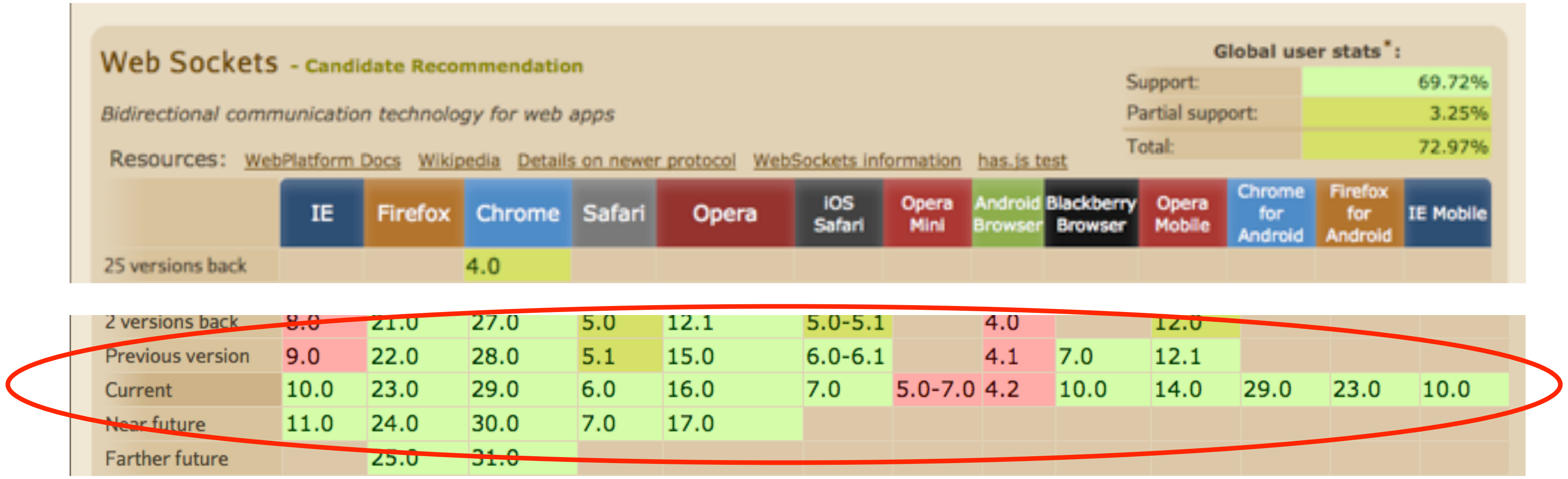
Support:	69.72%
Partial support:	3.25%
Total:	72.97%

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android	IE Mobile
25 versions back			4.0										
24 versions back			5.0										
23 versions back		2.0	6.0										
22 versions back		3.0	7.0										
21 versions back		3.5	8.0										
20 versions back		3.6	9.0										
19 versions back		4.0	10.0										
18 versions back		5.0	11.0										
17 versions back		6.0	12.0										
16 versions back		7.0	13.0										
15 versions back		8.0	14.0										
14 versions back		9.0	15.0										
13 versions back		10.0	16.0										
12 versions back		11.0	17.0		9.0								
11 versions back		12.0	18.0		9.5-9.6								
10 versions back		13.0	19.0		10.0-10.1								
9 versions back		14.0	20.0		10.5								
8 versions back		15.0	21.0		10.6								
7 versions back		16.0	22.0		11.0								
6 versions back		17.0	23.0		11.1			2.1		10.0			
5 versions back	5.5	18.0	24.0	3.1	11.5	3.2		2.2		11.0			
4 versions back	6.0	19.0	25.0	3.2	11.6	4.0-4.1		2.3		11.1			
3 versions back	7.0	20.0	26.0	4.0	12.0	4.2-4.3		3.0		11.5			
2 versions back	8.0	21.0	27.0	5.0	12.1	5.0-5.1		4.0		12.0			
Previous version	9.0	22.0	28.0	5.1	15.0	6.0-6.1		4.1	7.0	12.1			
Current	10.0	23.0	29.0	6.0	16.0	7.0	5.0-7.0	4.2	10.0	14.0	29.0	23.0	10.0
Near future	11.0	24.0	30.0	7.0	17.0								
Farther future		25.0	31.0										

Note: Partial support refers to the websockets implementation using an older version of the protocol and/or the implementation being disabled by default (due to security issues with the older protocol).

http://caniuse.com/websockets

Browsers and WebSockets



Web Sockets - Candidate Recommendation

Bidirectional communication technology for web apps

Resources: WebPlatform Docs Wikipedia Details on newer protocol WebSockets information has.js test

Global user stats*:

Support:	69.72%
Partial support:	3.25%
Total:	72.97%

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android	IE Mobile
25 versions back			4.0										
2 versions back	8.0	21.0	27.0	5.0	12.1	5.0-5.1		4.0		12.0			
Previous version	9.0	22.0	28.0	5.1	15.0	6.0-6.1		4.1	7.0	12.1			
Current	10.0	23.0	29.0	6.0	16.0	7.0	5.0-7.0	4.2	10.0	14.0	29.0	23.0	10.0
Near future	11.0	24.0	30.0	7.0	17.0								
Farther future		25.0	31.0										

http://caniuse.com/websockets

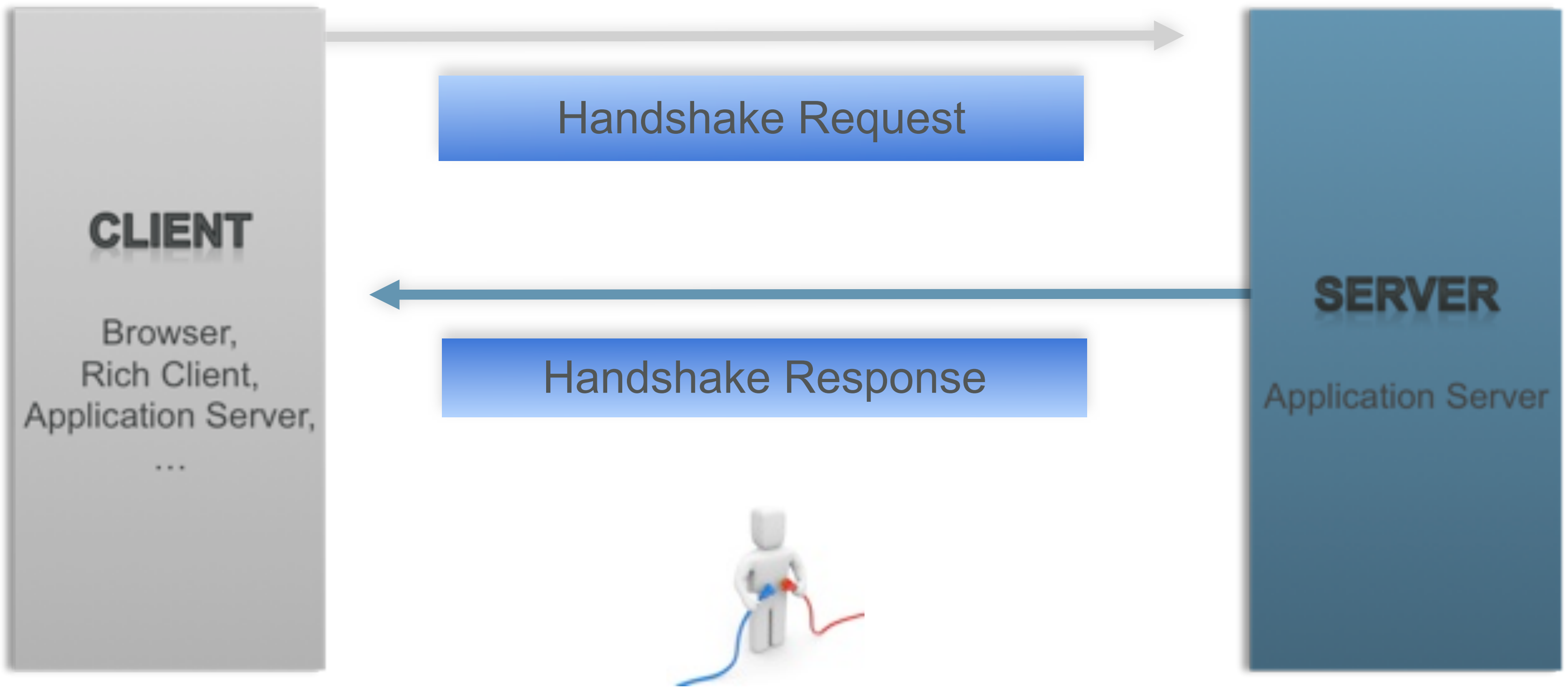
JSR 356 - Java API for WebSocket

- WebSocket Java client & server API
- Part of Java EE 7
  - Inc. in Web Profile!
- Under consideration for Java SE 9
- Reference Implementation
  - http://tyrus.java.net
- Supported in Glassfish 4.0

What’s the basic idea ?

- Establish a WebSocket connection
- Send messages backwards and forwards
- End the connection

Establish a connection



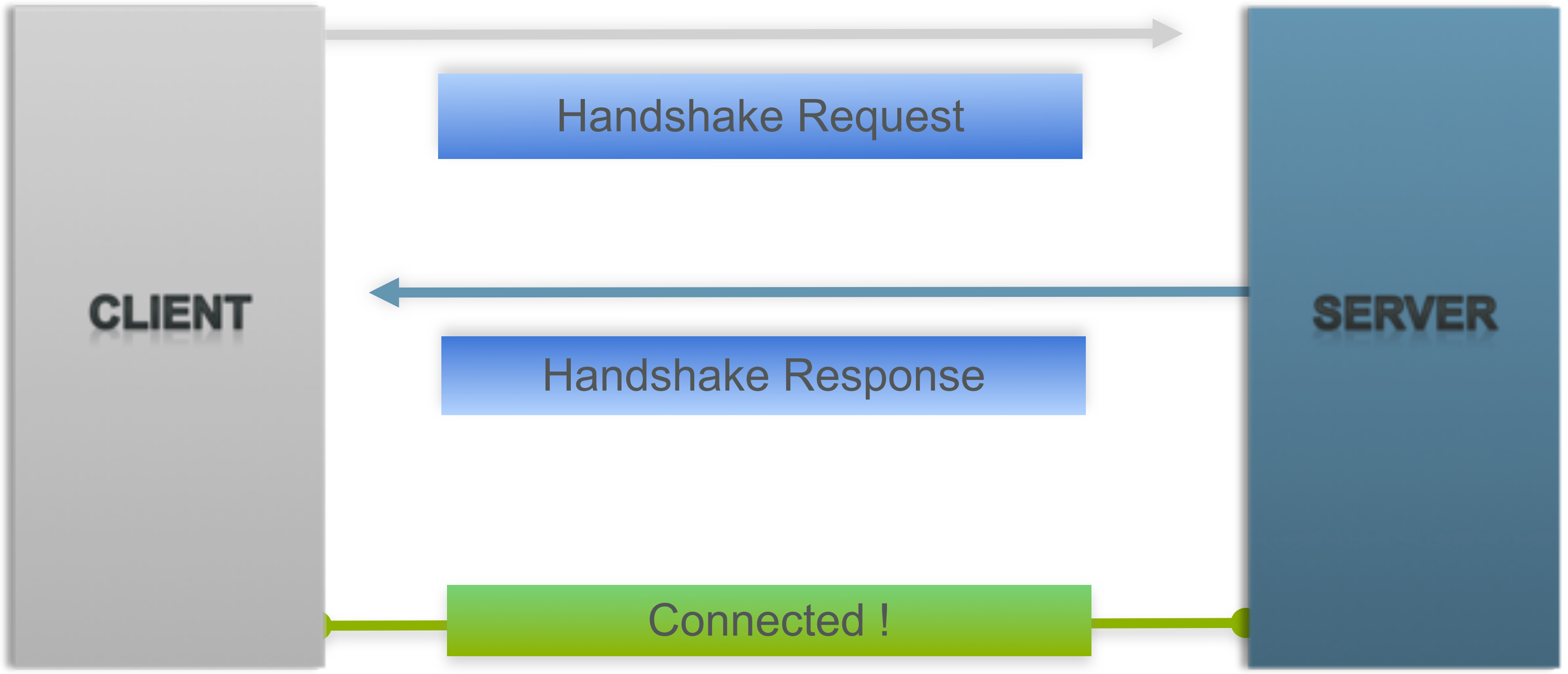
Handshake Request

```
Http Request
GET /mychat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: megachat, chat
Sec-WebSocket-Extensions : compress, mux
Sec-WebSocket-Version: 13
Origin: http://example.com
```

Handshake Response

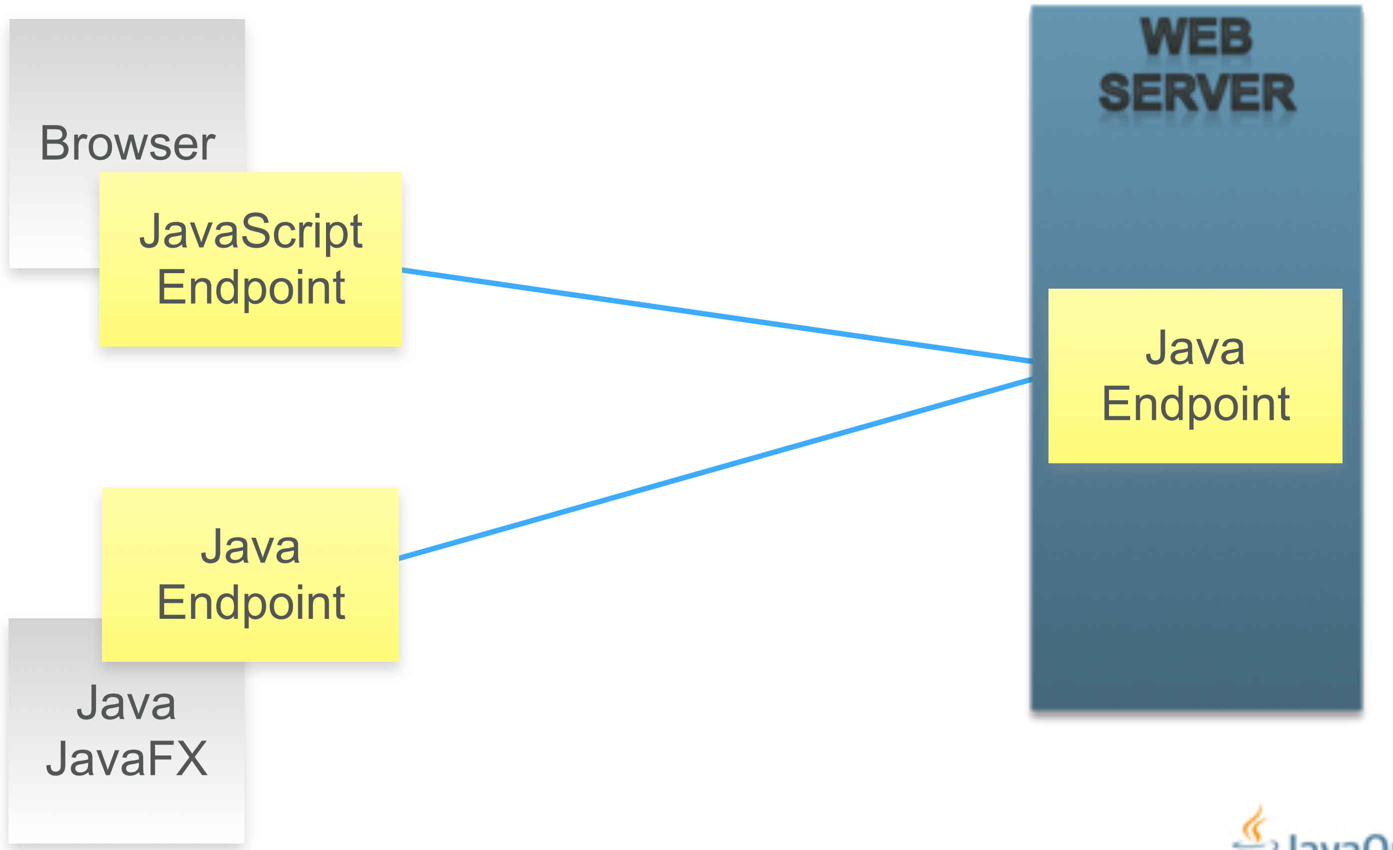
```
Http Response
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Extensions: compress, mux
```

Establishing a connection

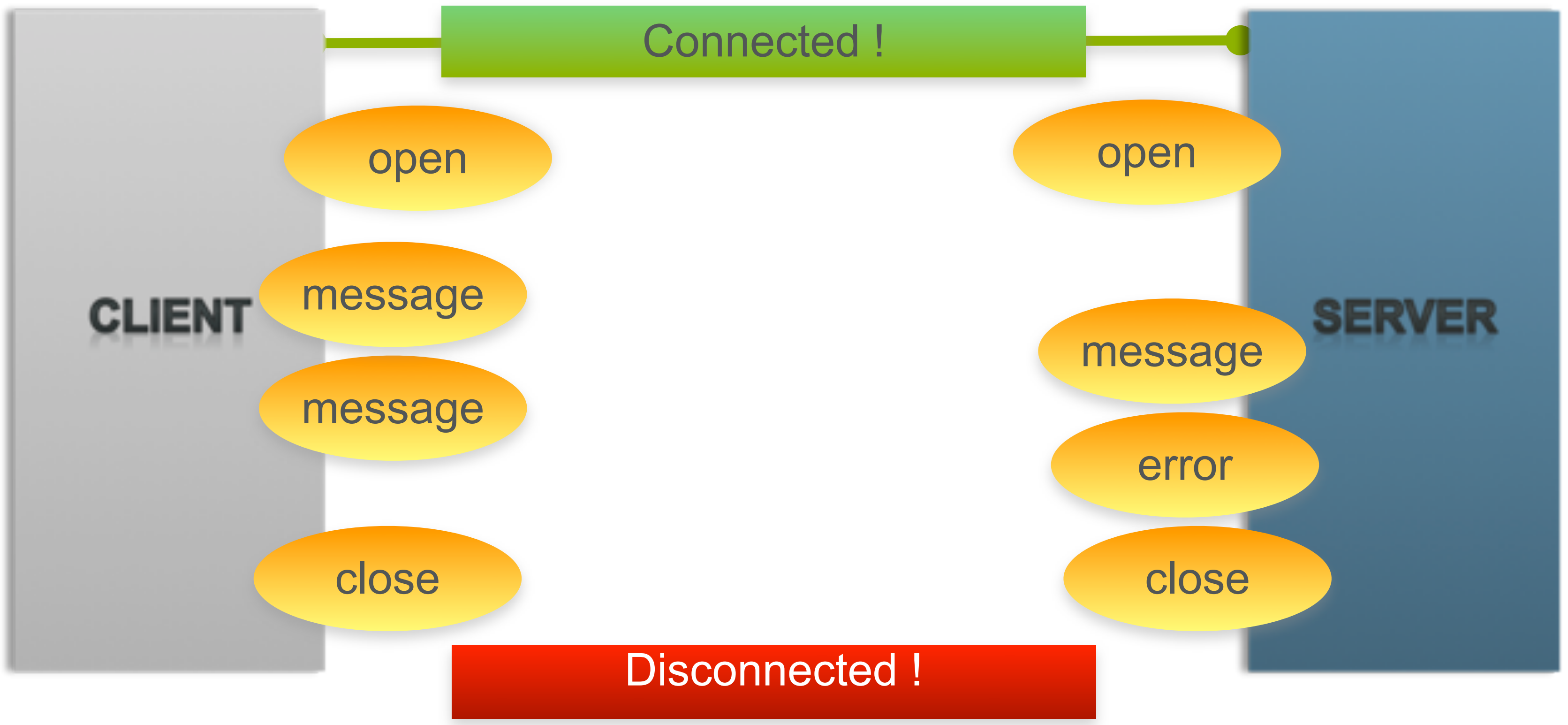


Fundamentals

Endpoints


Browser
JavaScript Endpoint
Java Endpoint
Java JavaFX
WEB SERVER
Java Endpoint

Lifecycle


Connected !
CLIENT
SERVER
open
open
message
message
message
error
close
close
Disconnected !

WebSockets Endpoints

- Create Java classes that are WebSocket endpoints
- The endpoint implements all the WebSocket functionality
- Endpoints can be
  - Annotated : POJO + Annotations
  - Programmatic : Java classes that extend WebSocket API class
  - Clients : Always connect to one server
  - Servers : Many clients can connect

Hello World Server (annotated)

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

@ServerEndpoint("/echo")
public class EchoServer {

    @OnMessage
    public String echo(String incomingMessage) {
            return ″Hello " + incomingMessage;
    }

}
```

Hello World Server (programmatic)

```
public class ProgrammaticEchoServer extends Endpoint {

@Override
public void onOpen(Session session, EndpointConfig endpointConfig {

final Session mySession = session;

mySession.addMessageHandler(new MessageHandler.Whole<String>() {

     @Override
      public void onMessage(String text) {
      try {
         mySession.getBasicRemote().sendText("I got " + text);
      } catch (IOException ioe) {
          System.out.println(”Oops " + ioe.getMessage());
      }}});
}

}
```

Hello Client (annotated)

```
import java.io.IOException;
import javax.websocket.*;

@ClientEndpoint
public class HelloClient {

@OnOpen
public void init(Session session) {
    try {
        session.getBasicRemote().sendText("Hello you !");
    } catch (IOException ioe) {
        // oops
    }
}
}
```

Hello Client (programmatic)

```
public class MyClient extends Endpoint {
  @Override
  public void onOpen(final Session session, EndpointConfig ec) {
    try {
         session.getBasicRemote().sendText(“Hello Duke”);
    } catch (IOException ex) {
         // Ooops
    }
}
```

Hello Client - Bootstrap

```
WebSocketContainer container =
                      ContainerProvider.getWebSocketContainer();
String uri = "ws://localhost:8080"
                      + request.getContextPath() + "/websocket";
container.connectToServer(MyClient.class, null, URI.create(uri));
```

Lifecycle

Intercepting WebSocket events

	Annotated Endpoints	**Programmatic Endpoints** `javax.websocket.Endpoint` **method to override**
Open event	`@OnOpen`	`public void onOpen(Session session, EndpointConfig config)`
Message event	`@OnMessage`	Create and register an instance of a `javax.websocket.MessageHandler`
Error event	`@OnError`	`public void onError(Session session, Throwable thrw)`
Close event	`@OnClose`	`public void onClose(Session session, CloseReason reason)`

Basic Messaging

Message

- Payload
  - Text
  - Binary
- Synchronous
  - Whole message / Sequence of partial messages
  - Ping-Pong
- Asynchronous
- No delivery guarantee!

Sending messages

Use the **`RemoteEndpoint`** interface, obtained from the **`Session`** object

```
mySession.getBasicRemote().sendText(”Hello”);
```

Return a value from your **`@OnMessage`** method (convenience)

```
@OnMessage
public String echo(String incomingMessage) {
    return(“Hello”);
}
```

Sending messages

Means of sending	RemoteEndpoint method to use
as whole string	`sendText(String message)`
as binary data	`sendBinary(ByteBuffer message)`
in string fragments	`sendText(String part, boolean last)`
in binary data fragments	`sendBinary(ByteBuffer part, boolean last)`
as a blocking stream of text	`Writer getSendWriter())`
as a blocking stream of binary data	`OutputStream getSendStream()`
as a custom object of type T	`sendObject(T customObject)`

Receiving messages

Annotation - Annotated a suitable method with **`@OnMessage`**

```
@OnMessage
public void whenGettingAText(String message)

@OnMessage
public void whenGettingPartialText(String message, boolean isLast)
```

Programmatic - Implement a **`MessageHandler`**, add it to the **`Session`**

```
MyTextMessageHandler implements MessageHandler.Whole<String>...
...
session.addMessageHandler(new MyTextMessageHandler());
```

Receive a message - Annotation

	`@OnMessage` **method**

Text	`public void handleText(String message)`
	`public void handleReader(Reader r)`
	`public void handleTextPieces(String message, boolean isLast)`

Binary	`public void handleBinary(ByteBuffer bb)`
	`public void handleStream(InputStream is)`
	`public void handleBinaryPieces(ByteBuffer bb, boolean isLast)`

Any object	`public void handleObject(CustomObject co)`

Receive a message - Programmatically

	Annotation	Example

Text	`@OnMessage`	`MessageHandler.Whole<String>`
		`MessageHandler.Partial<String>`
		`MessageHandler.Whole<Reader>`

Binary	`@OnMessage`	`MessageHandler.Whole<ByteBuffer>`
		`MessageHandler.Partial<ByteBuffer>`
		`MessageHandler.Whole<InputStream>`

Any object	`@OnError`	`MessageHandler.Whole<CustomObject>`

```
session.addMessageHandler(…);
```

Sending Java Object

Java primitive

- Text
- `x.toString()`

Java Object

- Custom `javax.websocket.Encoder` implementation
- Text or binary

Encoders and Decoders

Encoder

- Object to Binary: `Encoder.Binary<T>`, `Encoder.BinaryStream<T>`
- Object to Text: `Encoder.Text<T>`, `Encoder.TextStream<T>`

Decoder

- Text to Object: `Decoder.Text<T>`, `Decoder.TextStream<T>`
- Binary to Object: `Decoder.Binary<T>`, `Decoder.BinaryStream<T>`

Lifecycle

- `init()` and `destroy()` methods
- `willDecode ()`

Custom Payload - Text

```
public class MyMessage implements Decoder.Text<MyMessage>,
                                  Encoder.Text<MyMessage> {
  private JsonObject jsonObject;

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

  public MyMessage decode(String s) {
    jsonObject = new JsonReader(new StringReader(s)).readObject();
    return this;
  }

  public String encode(MyMessage myMessage) {
    return myMessage.jsonObject.toString();
  }
```

Custom Payload - Binary

```
public class MyMessage implements Decoder.Binary<MyMessage>,
                                          Encoder.Binary<MyMessage> {

  public boolean willDecode(byte[] bytes) {
    ...
    return true;  // Only if can process the payload
  }
  public MyMessage decode(byte[] bytes) {
    ...
    return this;
  }
  public byte[] encode(MyMessage myMessage) {
    ...
```

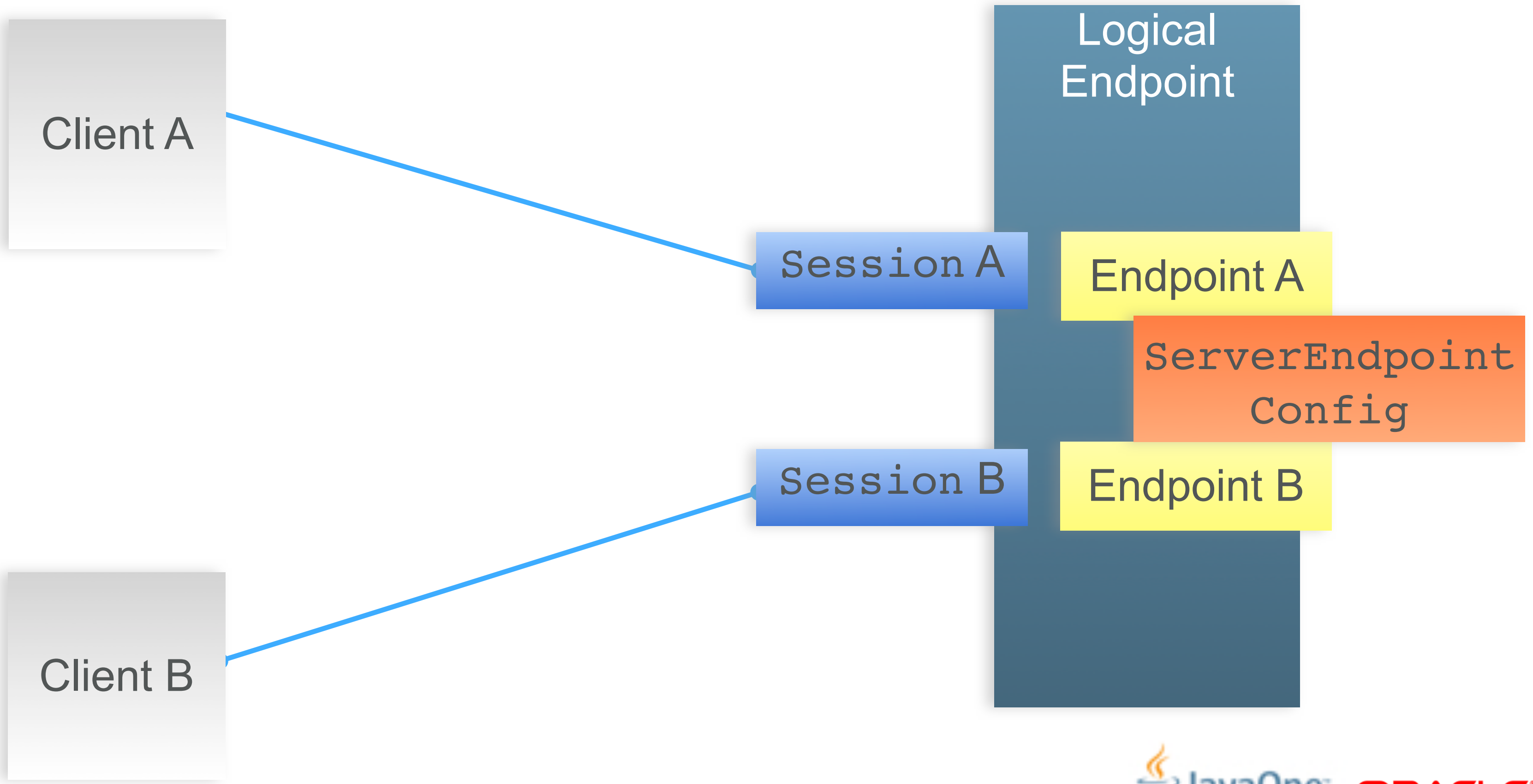
Encoders and Decoders - Configurations

```
@ServerEndpoint(
    decoders = { Foo.class, Bar.class },
    encoders = { Baz.class, Forz.class }
)
```

```
List<Encoder> myEncoders = new List<>();
myEncoders.add(Foo.class);
myEncoders.add(Bar.class);
ServerEndpointConfig myConfig =
    ServerEndpointConfig.Builder.create(MyEp.class, "/bar")
            .encoders(myEncoders)
            .build();
```

Configuration and Sessions

Instance diagram of EndpointConfigs and Sessions



User state and endpoint state

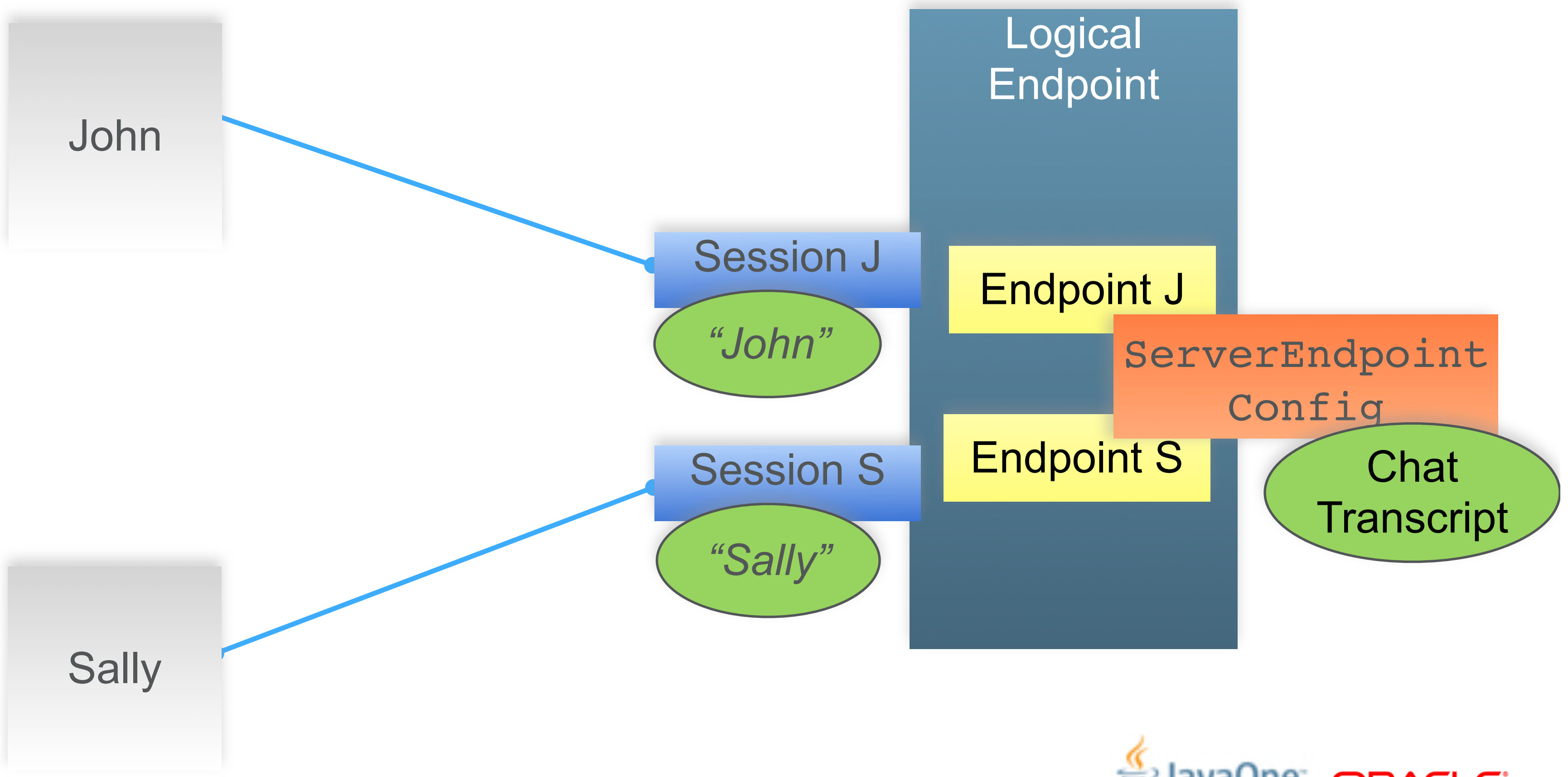
`Session`

- Representation of the WebSocket connection with a peer
- One instance per peer
- Has property bag for per peer endpoint application state

`EndpointConfig`

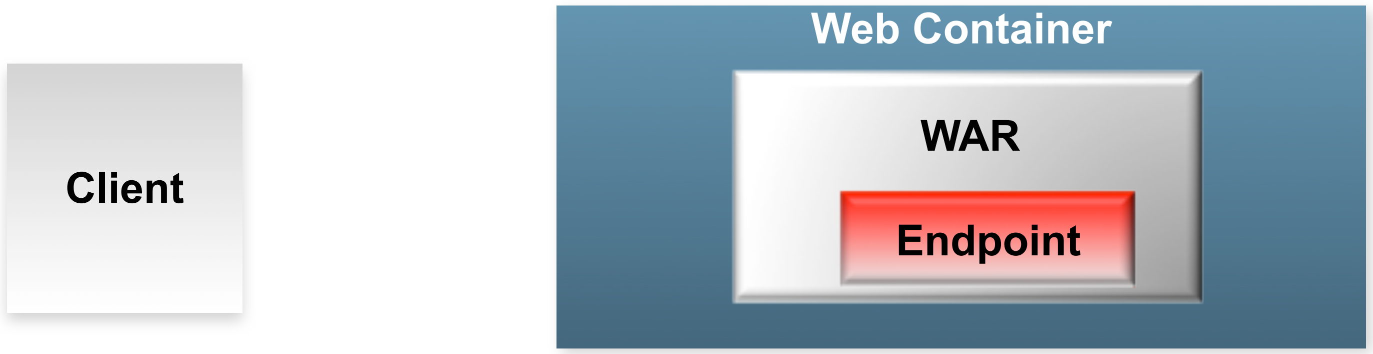
- Holds all the configuration information for a logical endpoint
- One instance per logical endpoint
- Has property bag for per logical endpoint application state
- Can extend **`EndpointConfig`** to add behavior

Storing Chat application state



WebSocket Path Mapping

WebSocket Path Anatomy



endpoint path: **/foo**

+

WAR context root: **mywebapp**

+

Server hostname: **www.myapps.com**

URL Endpoint **ws://www.myapps.com/mywebapp/foo**

Path Types - exact URI

Server endpoints can be mapped to exact URI

- Example `/foo` or `/airlines/seating/updater`

  `/foo` != `/Foo`
- Implementations use an *exact match* policy
- Can’t have 2 endpoints mapped to the same URI in the same app

Path Types - URI Template

URI-Template

- Level 1, i.e. simple string extension
- Example `/travel/{mode}`

  `/travel/{mode}/{class}`
- Match if incoming URI is a valid expansion of the URI template

  `/travel/boat , /travel/plane` ✔

  `/travel , /travel/steam/train` ✘
- Can’t have two equivalent URI-templates in an application

  `/travel/{mode} + /travel/{vip}` ✘
- Expanded parameters can be retrieved by name inside the endpoint

URI Template and Path Parameter

```
@ServerEnpoint("/travel/{mode}")
public class Trip {

@OnOpen
public void Booking ( @PathParam("mode") String travelMode ) {

     switch (travelMode) {
           case "plane" :
           . . .
```

URI Template and Path Parameter

```
@ServerEnpoint("/travel/{mode}/{class}")
public class Trip {

@OnOpen
public void Booking (Session session, EndpointConfig config) {

  Map <String, String> pathParam = session.getPathParameters();
  String travelMode = pathParam.get("mode");
  String travelClass = pathParam.get("class");
  . . .

}
```

URI Template, Path Parameter and Query String

```
ws://masterTrip.com/travel/plane?destination=hawaii


@OnOpen
public void makeOffer(Session session, EndpointConfig epCfg) {

String targetLocation =
       session.getRequestParameterMap().get("destination").get(0);

. . .
```

Misc.

Security

Based on the Servlet model

- WebSocket Endpoints are considered resources via their uri
- Require users be authenticated to access a WebSocket endpoint
- Limit access to a WebSocket endpoint
  - to certain users, via role mapping
  - to an encrypted protocol (wss://)

API to give at runtime a view on the security model

Endpoint Configuration

Control the instantiation and initialization of Endpoint

Access the initial HTTP request

- Eg. Perform custom checks

Modify the handshake response

Choose a subprotocol from those requested by the client

```
ServerEndpointConfig.Builder
                .create(MyEndpoint.class, “/myUri”).build();
```

Wrap up

WebSocket

Concepts

- Session, Endpoints, lifecycle

Java API for WebSocket

- Annotation & Programmatic based
- Part of Java EE 7

Annotations

Annotation	Level	Purpose
`@ServerEndpoint`	class	Turns a POJO into a WebSocket Endpoint
`@ClientEndpoint`	class	Turns a POJO into a WebSocket Client
`@OnMessage`	method	Intercepts WebSocket Message events
`@OnOpen`	method	Intercepts WebSocket Open events
`@OnError`	method	Intercepts errors during a conversation
`@OnClose`	method	Intercepts WebSocket Close events
`@PathParam`	method param	Flags a matched path segment of a URI-template

WebSocket

Advanced

- Security, Asynchronous, Protocols, Batching, etc.

Considerations

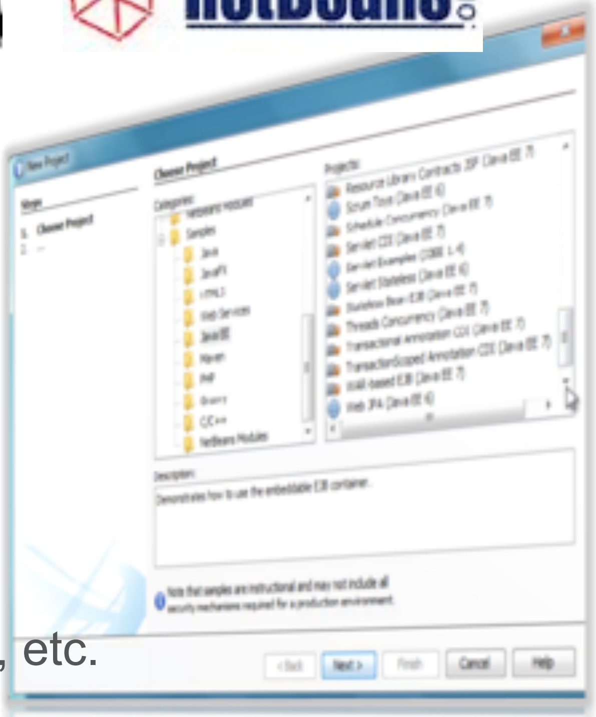
- Firewall
- Tools

Get started now!

NetBeans bundles GlassFish
Incremental compilation; auto-deploy
Complete Java EE 7 support

- All Java EE 7 project types
- HTML5 features available in Java EE projects
- Maven Support
- Advanced Wizards
  - Entity to REST generation, DB to JSF 2.2, etc.

•Embedded WebKit browser


New Project

Steps
1. Choose Project
2. ...

Choose Project

Categories:
- Samples
  - Java
  - JavaFX
  - HTML5
  - Web Services
  - Java EE
  - Maven
  - PHP
  - Groovy
  - C/C++
  - NetBeans Modules

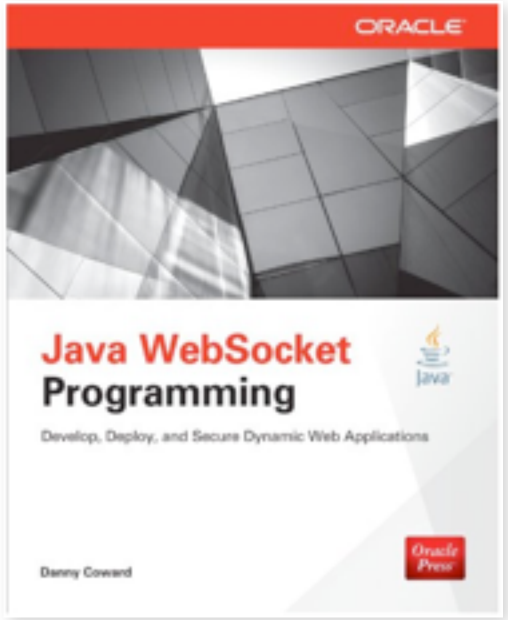
Projects:
- Resource Library Contracts JSF (Java EE 7)
- Scrum Toys (Java EE 6)
- Schedule Concurrency (Java EE 7)
- Servlet CDI (Java EE 7)
- Servlet Examples (J2EE 1.4)
- Servlet Stateless (Java EE 6)
- Stateless Bean EJB (Java EE 7)
- Threads Concurrency (Java EE 7)
- Transactional Annotation CDI (Java EE 7)
- TransactionScoped Annotation CDI (Java EE 7)
- WAR-based EJB (Java EE 7)
- Web JPA (Java EE 6)

Description:
Demonstrates how to use the embeddable EJB container.

Note that samples are instructional and may not include all security mechanisms required for a production environment.

Resources

- •Java EE 7 Tutorial
- •Java WebSocket Programming (Oracle Press)
- •NetBeans 7.4 WebSocket samples
- •http://glassfish.org


ORACLE
Java WebSocket Programming
Develop, Deploy, and Secure Dynamic Web Applications
Danny Coward
Oracle Press

MAKE THE
FUTURE
JAVA

JavaOne™

ORACLE®