WEBSOCKETS

50

50

“Real-time” interaction

- Update client
  - Chat
  - Sports
  - Stocks
  - Gaming
  - etc

51

51

Real-time interaction

- Polling
  - Good if updates at known intervals
  - Much unnecessary connection opening/closing

52

52

Real-time interaction

- Long-Polling
  - Server keeps request open for set period
  - If event, send to client
  - Close connection otherwise
  - Problem: Could be worse than polling
  - Problem: Latency
  - Problem: HTTP header overhead
    - Client requests

53

53

Real-time interaction

- HTTP Streaming
  - Client sends request
  - Server “streams” open response
  - Problem: buffering proxies
    - Latency
  - Problem: complexity
    - Mapping between client requests and streaming events
  - Good for heavy server-to-client traffic

54

54

WebSockets

- 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

55

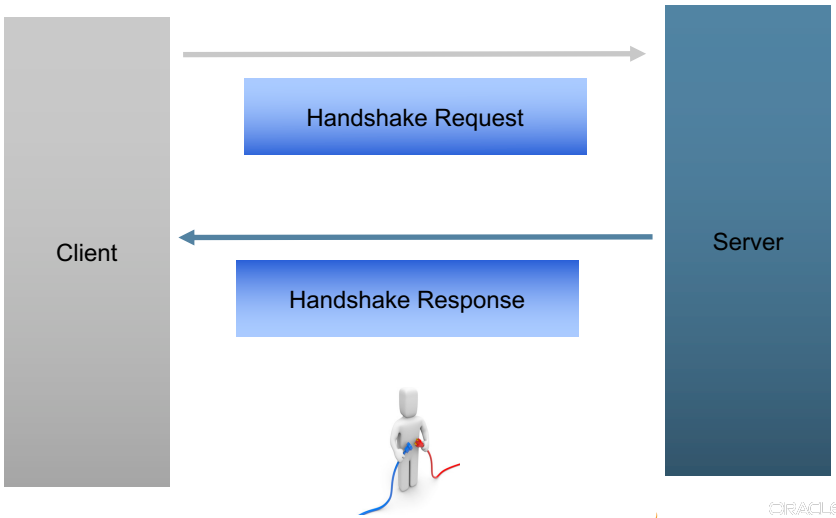
55

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

56

Establish a connection



57

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
```

58

58

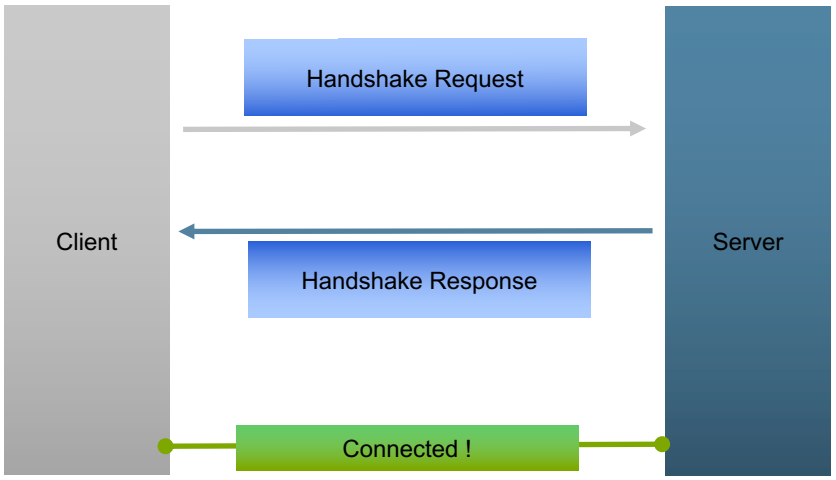
Handshake Response

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

59

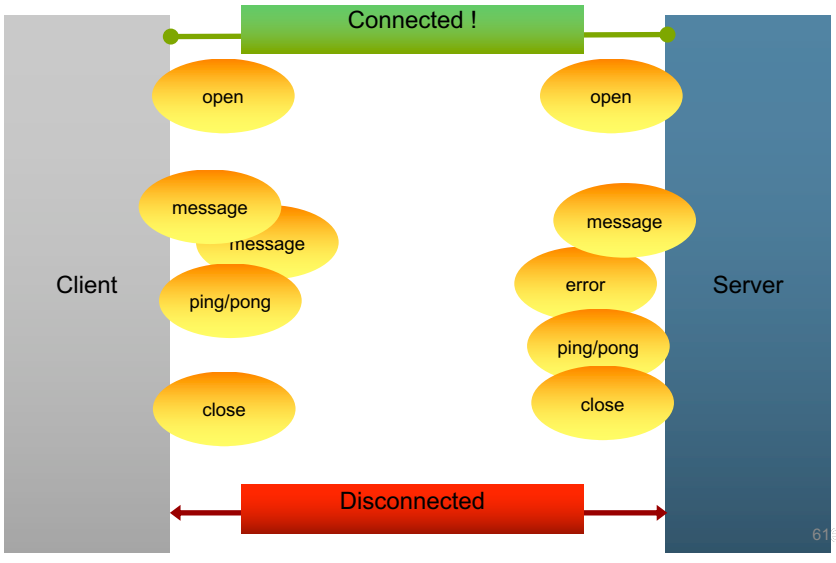
59

Establishing a Connection



60

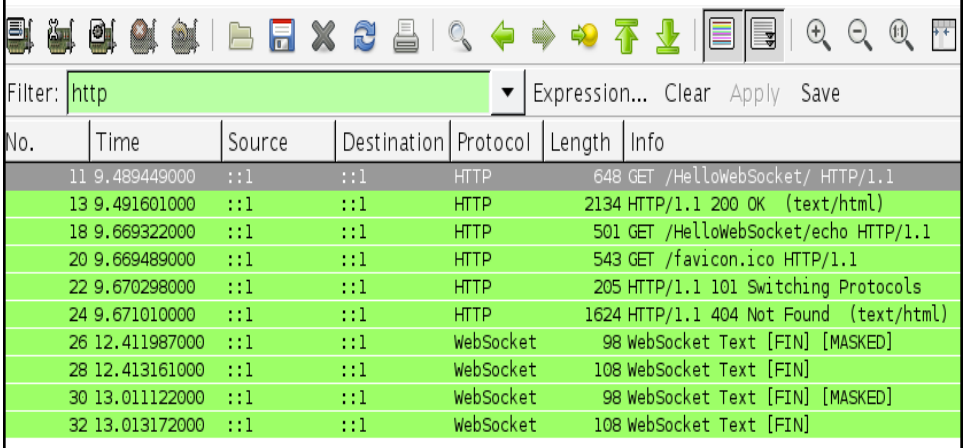
WebSocket Lifecycle



61

WebSocket wire messages

- Capture traffic on loopback



62

62

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

63

63

Advantages over TCP

- Upgrades from HTTP
- Origin domain
  - Use tokens to prevent CSRF attacks
- Proxy connection without losing info
  - HTTP CONNECT
- Message-oriented
- Messages obfuscated by client
  - 32-bit mask, xor-ed with message
- Control frame for closing connection
- Built-in heartbeat support

64

64

Advantages over TCP

- Upgrades from HTTP
- Origin domain
  - Use tokens to prevent CSRF attacks
- Proxy connection without losing info
  - HTTP CONNECT
- Message-oriented
- Messages obfuscated by client
  - 32-bit mask, xor-ed with message
- Control frame for closing connection
- Built-in heartbeat support

65

65

WEBSOCKET API

66

66

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 subprotocols

67

67

JSR 356 Specification

- Standard Java API for creating WebSocket Applications
- Specification (JSR 356): https://projects.eclipse.org/projects/ee4j.websocket
- Reference Implementation: https://projects.eclipse.org/projects/ee4j.tyrus

68

68

API Features

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

69

69

Hello World Server

```
public class HelloServer extends Endpoint {
  @Override
  public void onOpen(Session session,
                     EndpointConfig configuration) {
    session.addMessageHandler(
     new MessageHandler.Whole<String>() {
      public void onMessage(String name) {
        try {
          session.getBasicRemote().sendText("Hello " + name);
        } catch (IOException ioe) {
          // Handle failure.
        }
      }
    });
  }
}
```


70

Hello World Client

```
public class HelloClient extends Endpoint {
  @Override
  public void onOpen(Session session,
      EndpointConfig configuration) {
    try {
      session.getBasicRemote().sendText("Hello you!");
    } catch (IOException ioe) {
      . . .
    }
  }
}
```


71

Client Server Configuration

```
ServerContainer serverContainer =
    (ServerContainer) servletContext.getAttribute(
        “javax.websocket.server.ServerContainer”);

ServerEndpointConfig serverConfiguration =
    ServerEndpointConfig.Builder.create(
        HelloServer.class, "/hello").build();

serverContainer.addEndpoint(serverConfiguration);
...
```


Client Server Configuration

```
URI clientURI =
      new URI("ws://myserver.com/websockets/hello");

WebSocketContainer container =
    ContainerProvider.getWebSocketContainer();

ClientEndpointConfig clientConfiguration =
    ClientEndpointConfig.Builder.create().build();

container.connectToServer(HelloClient.class,
    clientConfiguration, clientURI);
```

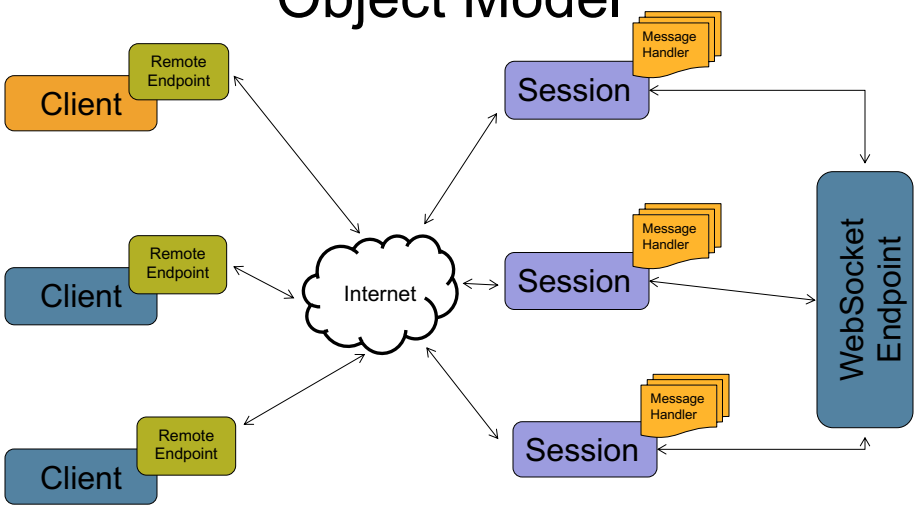
Main API Classes: javax.websocket.*

- **Endpoint**: Intercepts WebSocket lifecycle events
- **MessageHandler**: Handles all incoming messages for an Endpoint
- **RemoteEndpoint**: Represents the 'other end' of this conversation
- **Session**: Represents the active conversation

74

74

Object Model



75

75

Sending the Message

Whole string *	`RemoteEndpoint.Basic`	`sendText(String message)`
Binary data *	`RemoteEndpoint.Basic`	`sendBinary(ByteBuffer message)`
String fragments	`RemoteEndpoint.Basic`	`sendText(String part, boolean last)`
Binary data fragments	`RemoteEndpoint.Basic`	`sendBinary(ByteBuffer part, boolean last)`
Blocking stream of text	`RemoteEndpoint.Basic`	`Writer getSendWriter())`
Blocking stream of binary data	`RemoteEndpoint.Basic`	`OutputStream getSendStream()`
Custom object	`RemoteEndpoint.Basic`	`sendObject(Object customObject)`

* additional flavors: by completion, by future

76

Receiving the Message

Whole string	`MessageHandler.Whole<String>`	`onMessage(String message)`
Binary data	`MessageHandler.Whole<ByteBuffer>`	`onMessage(ByteBuffer message)`
String fragments	`MessageHandler.Partial<String>`	`onMessage(String part, boolean last)`
Binary data fragments	`MessageHandler.Partial<ByteBuffer>`	`onMessage(ByteBuffer part, boolean last)`
Blocking stream of text	`MessageHandler.Whole<Reader>`	`onMessage(Reader r)`
Blocking stream of binary data	`MessageHandler.Whole<InputStream>`	`onMessage(InputStream r)`
Custom object of type T	`MessageHandler.Whole<T>`	`onMessage(T customObject)`

77

ANNOTATION-DRIVEN API

78

78

Hello World

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

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

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

ORACLE

79

79

WebSocket Annotations

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

80

Hello World

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

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

  @OnMessage
  public ServerHello hello(ClientHello str) {
    return str;
  }
}
```


81

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) { ... }
```

82

82

Custom Payloads – Text

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

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

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

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

83

83

Custom Payloads – Binary

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

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

84

84

Chat Sample

```
@ServerEndpoint("/chat")
public class ChatBean {
    Set<Session> peers = Collections.synchronizedSet(…);

    @OnOpen
    public void onOpen(Session peer) {
        peers.add(peer);
    }

    @OnClose
    public void onClose(Session peer) {
        peers.remove(peer);
    }
    ...
```

85

85

Chat Sample (Continued)

```
    . . .

    @OnMessage
    public void message(String message, Session client) {
        for (Session peer : peers) {
            peer.getRemote().sendObject(message);
        }
    }
}
```

86

86

URI Template Matching

```
@ServerEndpoint("/orders/{order-id}")
public class MyEndpoint {
  @OnMessage
  public void processOrder(
    @PathParam("order-id") String orderId) {
    ...
  }
}
```

87

87

@OnMessage Methods

- A parameter type that can be decoded in incoming message
  - String, primitive, Reader, ByteBuffer, byte[], InputStream, or any type for which there is a decoder
- An optional Session parameter
- Boolean partial flag
- 0..n String parameters annotated with `@PathParameter`
- A return type that can be encoded in outgoing message
  - String, primitive, Writer, ByteBuffer, byte[], OutputStream, or any type for which there is an encoder

Jakarta Websocket

- Specification (JSR 356): https://projects.eclipse.org/projects/ee4j.websocket
- Reference Implementation: https://projects.eclipse.org/projects/ee4j.tyrus