Java™

ORACLE®
甲骨文

Building WebSocket Application in Java using JSR 356

Shing Wai Chan (陳成威)
Servlet 3.1 Specification Lead
java.net/blog/swchan2

MAKE THE FUTURE JAVA

Java™

Agenda

- Primer on WebSocket
- JSR 356: Java API for WebSocket
- Summary
- Resources

Primer on WebSocket

Interactive Web Sites

- HTTP is half-duplex
- HTTP is verbose
- Hacks for Server Push
  - Polling
  - Long Polling
  - Comet/Ajax
- Complex, Inefficient, Wasteful

Primer on WebSocket

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, 2012-09-20

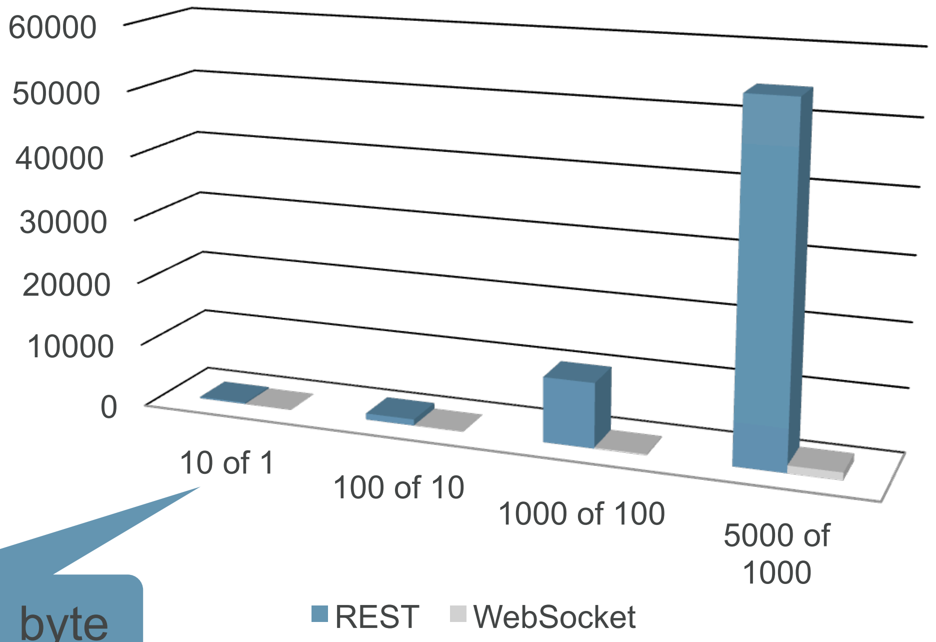
Primer on WebSocket

What’s the Basic idea

- Upgrade HTTP to upgrade to WebSocket
  - Single TCP connection
  - Transparent to proxies, firewalls, and routers
- Send data frames in both direction (Bi-directional)
  - No headers, cookies, authentication
  - No security overhead
  - “ping”/”pong” frames for keep-alive
- Send message independent of each other (Full Duplex)
- End the connection

Primer on WebSocket

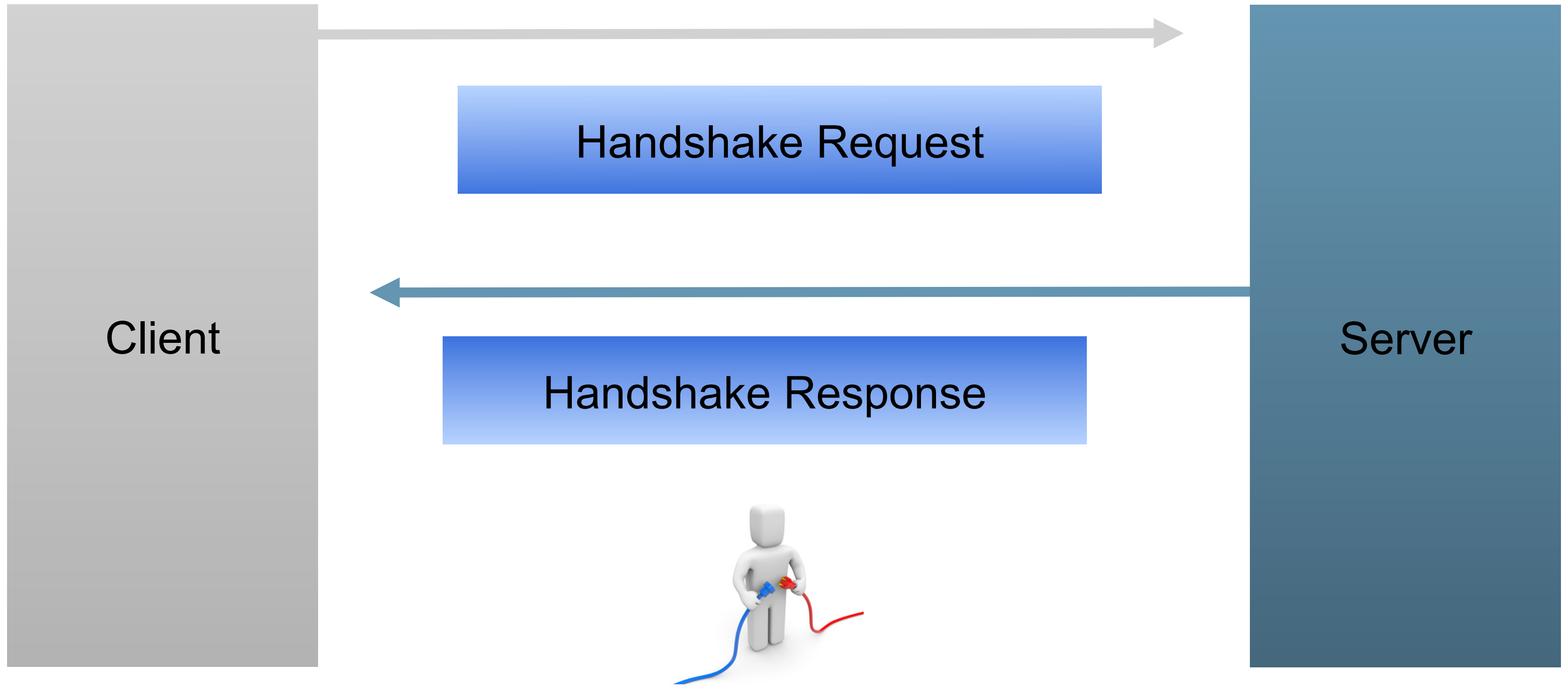
REST vs WebSocket



10 messages of 1 byte

Primer on WebSocket

Establish a Connection: Handshake



Primer on WebSocket

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
```

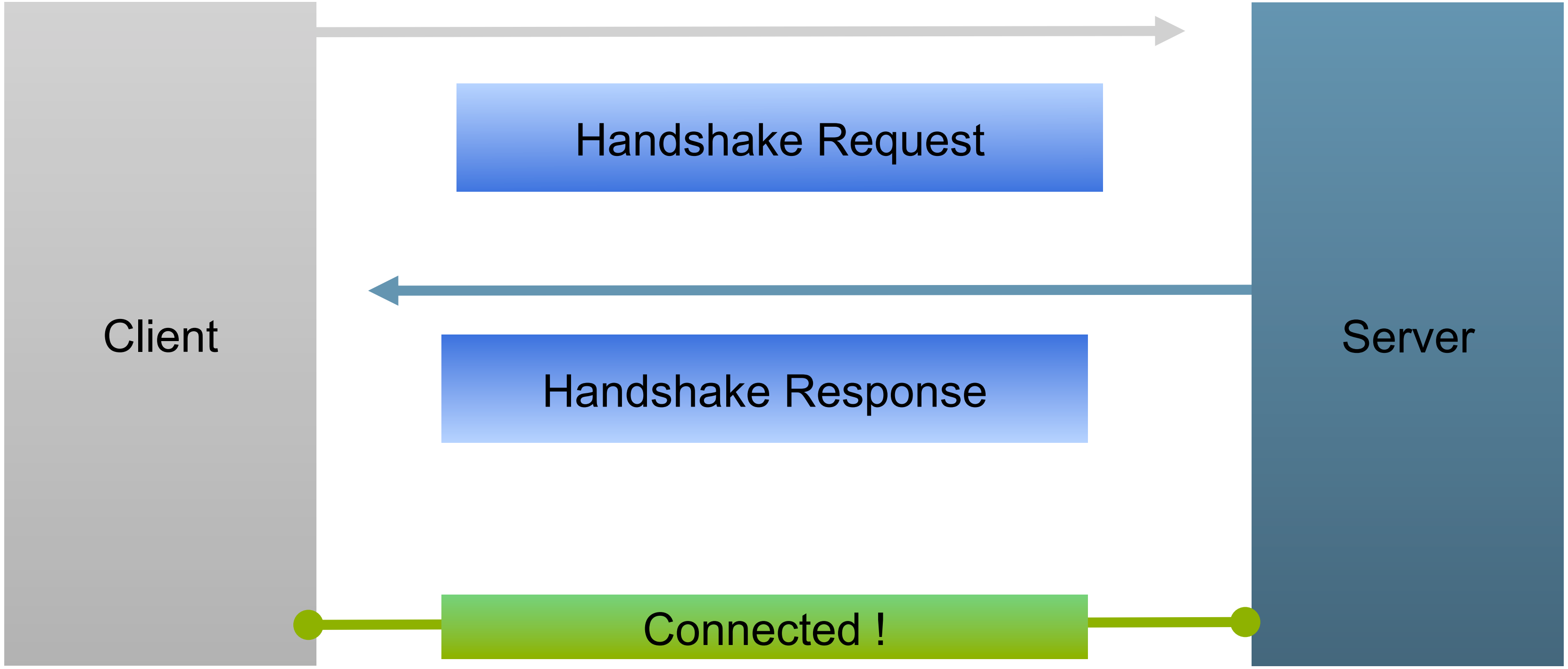
Primer on WebSocket

Handshake Response

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

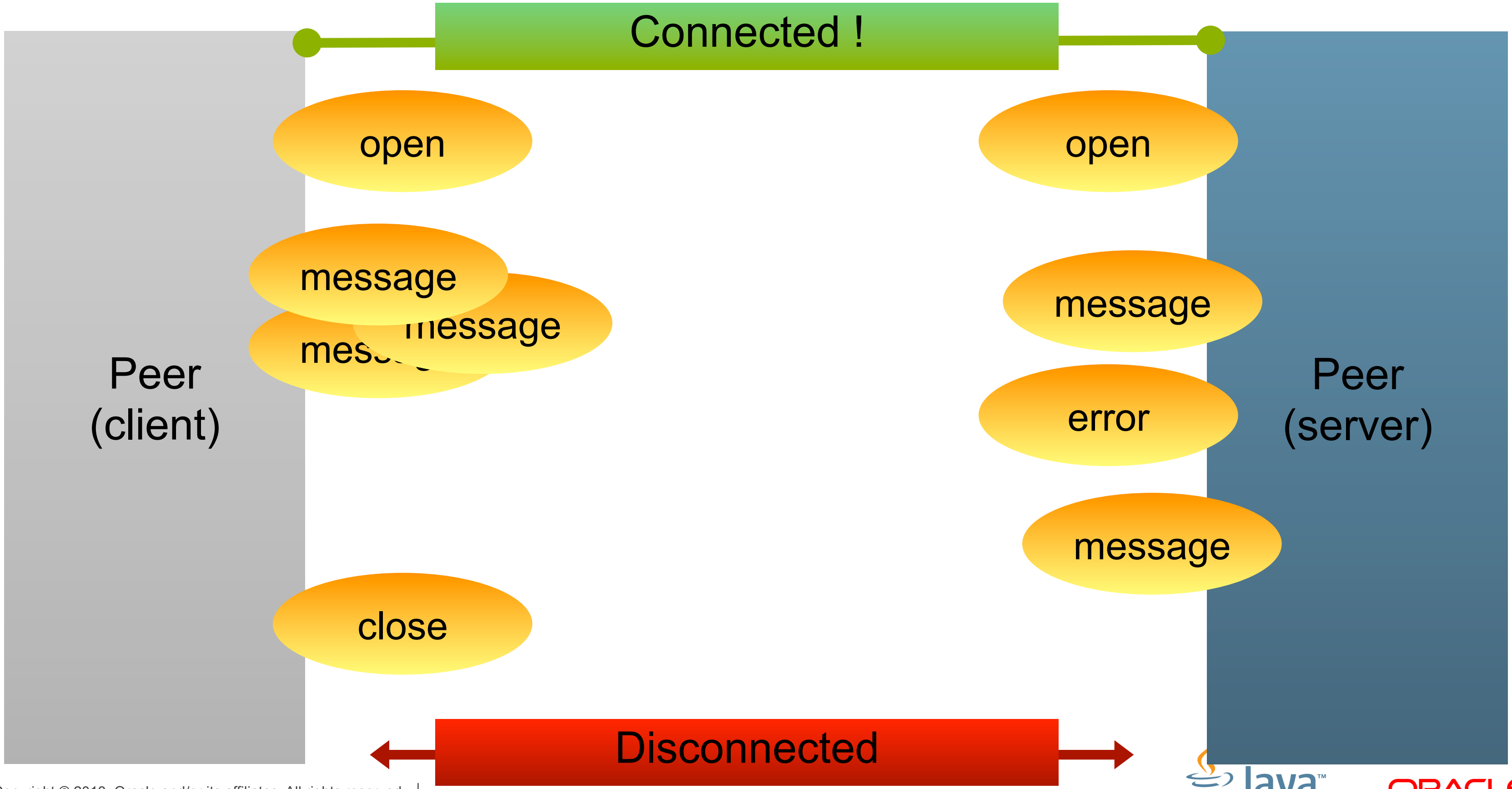
Primer on WebSocket

Establishing a WebSocket Connection



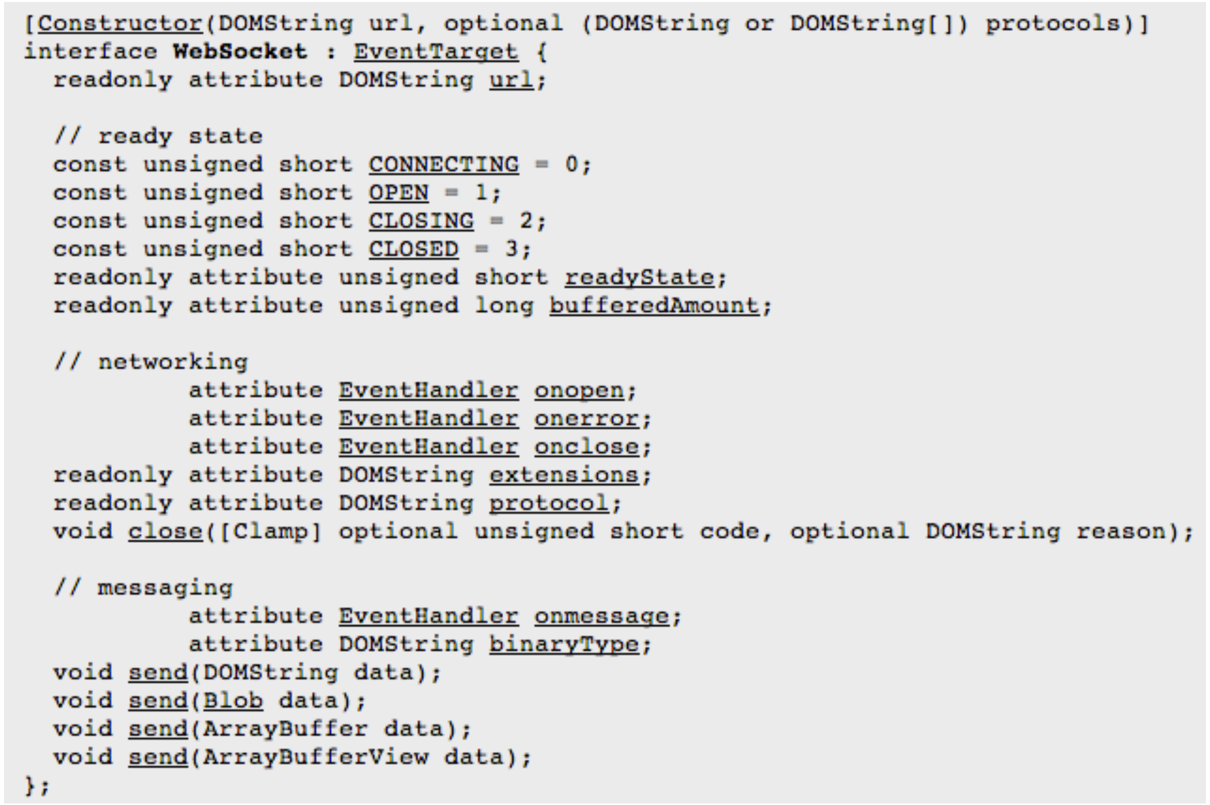
Primer on WebSocket

WebSocket Lifecycle



Primer on WebSocket

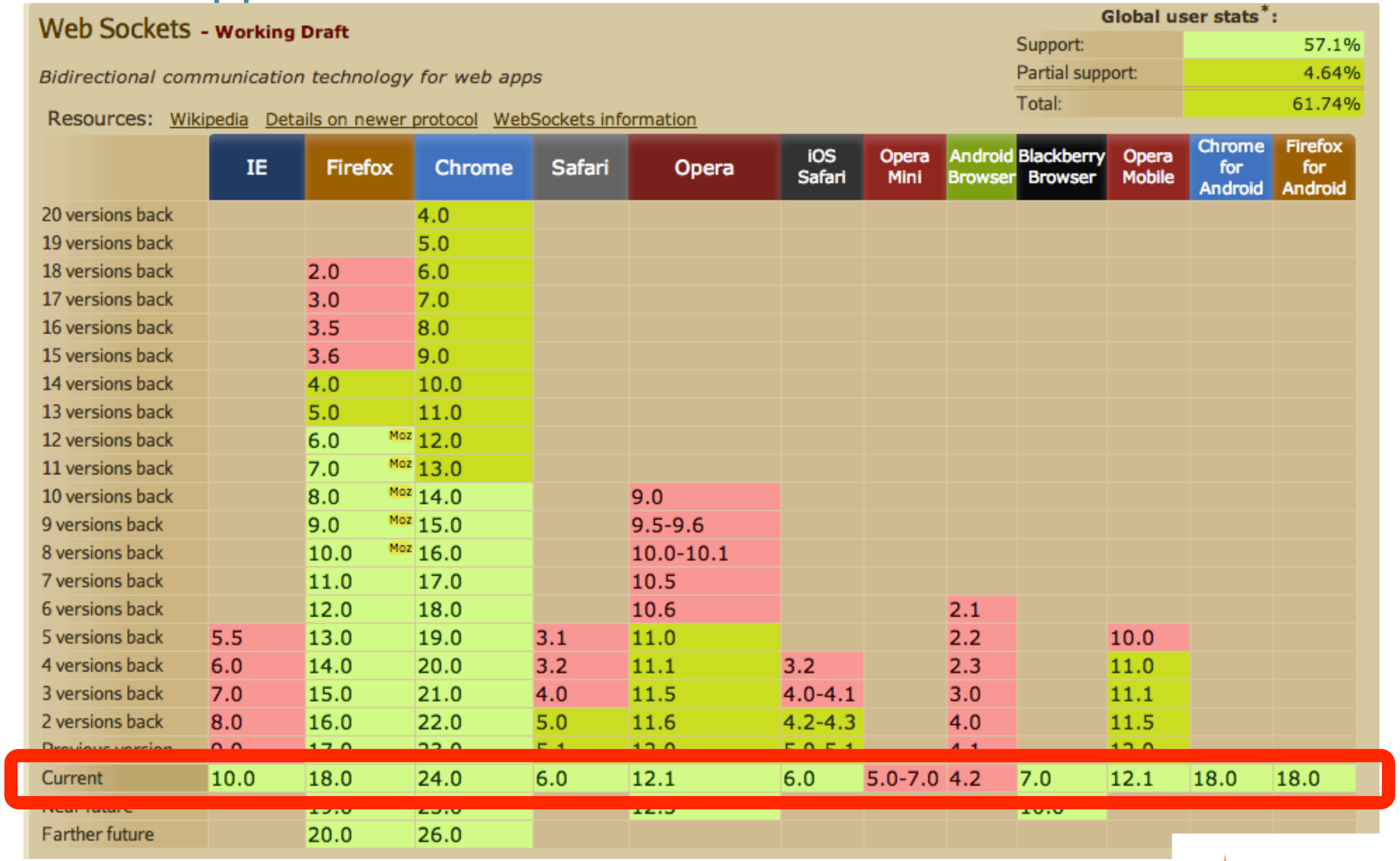
WebSocket API: www.w3.org/TR/websockets



```
[Constructor(DOMString url, optional (DOMString or DOMString[]) protocols)]
interface WebSocket : EventTarget {
  readonly attribute DOMString url;

  // ready state
  const unsigned short CONNECTING = 0;
  const unsigned short OPEN = 1;
  const unsigned short CLOSING = 2;
  const unsigned short CLOSED = 3;
  readonly attribute unsigned short readyState;
  readonly attribute unsigned long bufferedAmount;

  // networking
           attribute EventHandler onopen;
           attribute EventHandler onerror;
           attribute EventHandler onclose;
  readonly attribute DOMString extensions;
  readonly attribute DOMString protocol;
  void close([Clamp] optional unsigned short code, optional DOMString reason);

  // messaging
           attribute EventHandler onmessage;
           attribute DOMString binaryType;
  void send(DOMString data);
  void send(Blob data);
  void send(ArrayBuffer data);
  void send(ArrayBufferView data);
};
```

Primer on WebSocket

Browser Support



Web Sockets - **Working Draft**

Bidirectional communication technology for web apps

Resources: Wikipedia Details on newer protocol WebSockets information

Global user stats*:

Support:	57.1%
Partial support:	4.64%
Total:	61.74%

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android
20 versions back			4.0									
19 versions back			5.0									
18 versions back		2.0	6.0									
17 versions back		3.0	7.0									
16 versions back		3.5	8.0									
15 versions back		3.6	9.0									
14 versions back		4.0	10.0									
13 versions back		5.0	11.0									
12 versions back		6.0 Moz	12.0									
11 versions back		7.0 Moz	13.0									
10 versions back		8.0 Moz	14.0		9.0							
9 versions back		9.0 Moz	15.0		9.5-9.6							
8 versions back		10.0 Moz	16.0		10.0-10.1							
7 versions back		11.0	17.0		10.5							
6 versions back		12.0	18.0		10.6			2.1				
5 versions back	5.5	13.0	19.0	3.1	11.0			2.2		10.0		
4 versions back	6.0	14.0	20.0	3.2	11.1	3.2		2.3		11.0		
3 versions back	7.0	15.0	21.0	4.0	11.5	4.0-4.1		3.0		11.1		
2 versions back	8.0	16.0	22.0	5.0	11.6	4.2-4.3		4.0		11.5		
Previous version	9.0	17.0	23.0	5.1	12.0	5.0-5.1		4.1		12.0		
Current	10.0	18.0	24.0	6.0	12.1	6.0	5.0-7.0	4.2	7.0	12.1	18.0	18.0
Near future		19.0	25.0		12.5				10.0			
Farther future		20.0	26.0									

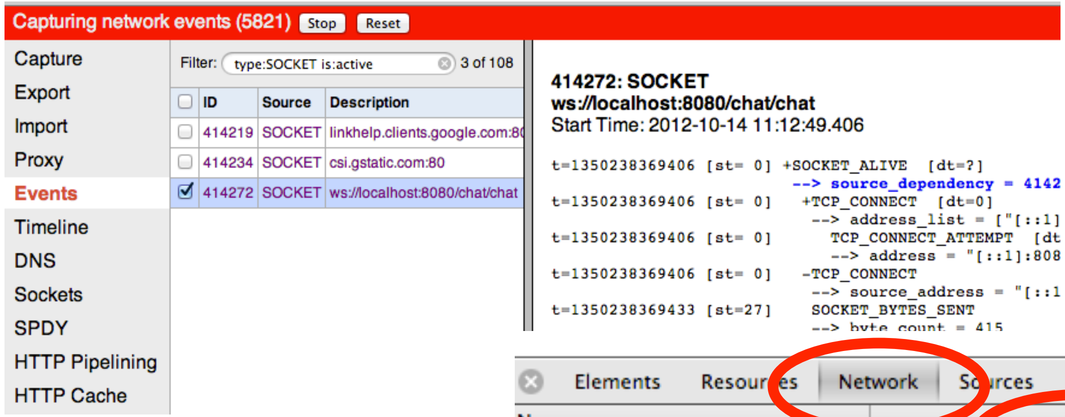
http://caniuse.com/websockets

JavaOne ORACLE

Primer on WebSocket

How to view WebSocket messages?

chrome://net-internals -> Sockets -> View live sockets



Capturing network events (5821) Stop Reset

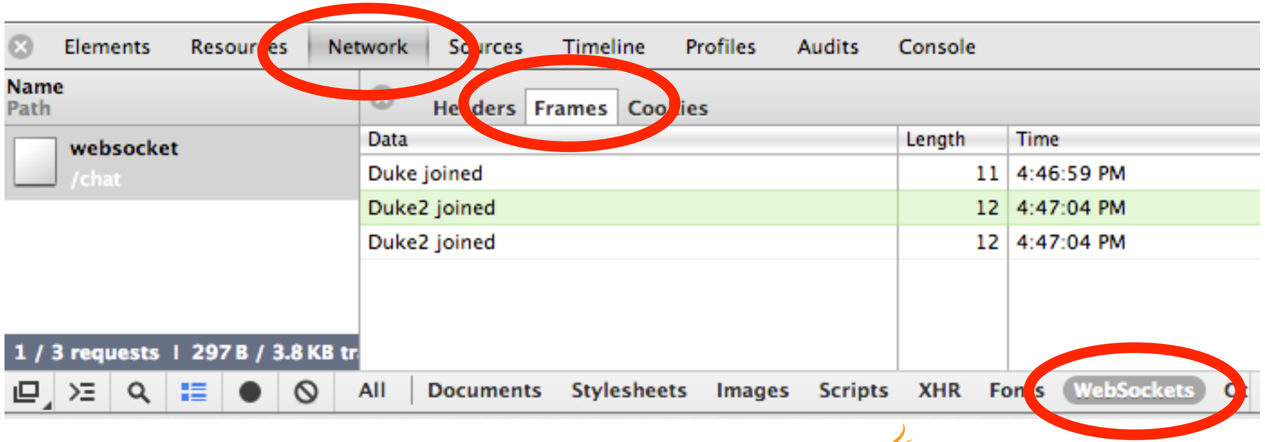
Capture
Export
Import
Proxy
Events
Timeline
DNS
Sockets
SPDY
HTTP Pipelining
HTTP Cache

Filter: type:SOCKET is:active 3 of 108

	ID	Source	Description
☐	414219	SOCKET	linkhelp.clients.google.com:8
☐	414234	SOCKET	csi.gstatic.com:80
☑	414272	SOCKET	ws://localhost:8080/chat/chat

414272: SOCKET
ws://localhost:8080/chat/chat
Start Time: 2012-10-14 11:12:49.406

```
t=1350238369406 [st= 0] +SOCKET_ALIVE  [dt=?]
                         --> source_dependency = 4142
t=1350238369406 [st= 0]   +TCP_CONNECT  [dt=0]
                           --> address_list = ["[::1]
t=1350238369406 [st= 0]      TCP_CONNECT_ATTEMPT  [dt
                             --> address = "[::1]:808
t=1350238369406 [st= 0]   -TCP_CONNECT
                           --> source_address = "[::1
t=1350238369433 [st=27]    SOCKET_BYTES_SENT
                           --> byte_count = 415
```



Elements Resources Network Sources Timeline Profiles Audits Console

Name
Path

websocket
/chat

Headers Frames Cookies

Data	Length	Time
Duke joined	11	4:46:59 PM
Duke2 joined	12	4:47:04 PM
Duke2 joined	12	4:47:04 PM

1 / 3 requests | 297 B / 3.8 KB tr

All Documents Stylesheets Images Scripts XHR Fonts WebSockets O

Agenda

- Primer on WebSocket
- JSR 356: Java API for WebSocket
- Summary
- Resources

JSR 356: Java API for WebSocket

JSR 356 Specification

- **FINAL**: New for Java EE 7
- Standard Java API for creating WebSocket Applications
- Transparent Expert Group
  - jcp.org/en/jsr/detail?id=356
  - java.net/projects/websocket-spec

JSR 356: Java API for WebSocket

JSR 356: Reference Implementation

- Tyrus: java.net/projects/tyrus
- Open source and transparent
- Integrated in GlassFish 4 Builds
  - download.java.net/glassfish/4.0/release

JSR 356: Java API for WebSocket

Overview

- Based on the rapidly adopted WebSocket protocol
- Standard Java API for creating WebSocket Applications
- Gives Web applications the ability to push data
- Enables richly interactive web applications
- Reaches any device with an HTML5 browser

JSR 356: Java API for WebSocket

Features Summary

- API for WebSocket Server and Client Endpoints
  - Annotated: `@ServerEndpoint`, `@ClientEndpoint`
  - Programmatic: `Endpoint`
    - WebSocket opening handshake negotiation
- Wide variety of message protocol facilities
- Integration into Java EE 7 Web container

JSR 356: Java API for WebSocket

Flexible Message Processing

- Send or receive text and binary messages
  - As complete messages
  - As sequence of partial messages
  - Using traditional blocking I/O
- Send or receive WebSocket messages as any Java object
  - Using pluggable encoder/decoder components
  - Encoders and decoders for Java primitives built in
- Send messages synchronously or asynchronously

JSR 356: Java API for WebSocket

WebSocket Server and Client

- Typical (1:1) : one instance per WebSocket Session/Client
  - Single threaded callbacks
- Untypical (1:n) : one shared instance per application, for multiple WebSocket Sessions/Clients
  - Concurrent callbacks

Hello World and Basics
POJO

JSR 356: Java API for WebSocket

Hello World Example

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

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

    @OnMessage
    public String sayHello(String name) {
        return “Hello “ + name;
    }
}
```

JSR 356: Java API for WebSocket

WebSocket Annotations

Annotation	Level	Purpose
@ServerEndpoint	class	Declare a Server Endpoint
@ClientEndpoint	class	Declare a Client Endpoint
@OnOpen	method	Declare this method handles WebSocket Open events
@OnMessage	method	Declare this method handles WebSocket Messages
@OnClose	method	Declare this method handles WebSocket Close events
@OnError	method	Declare this method handles errors
@PathParam	method parameter	Declare this parameter matches a path segment of a URI-template

JSR 356: Java API for WebSocket

@ServerEndpoint attributes

value	Relative URI or URI template e.g. “/hello” or “/chat/{subscriber-level}”
decoders	list of message decoder classes
encoders	list of message encoder classes
subprotocols	list of the names of the supported subprotocols
configurator	optional custom configurator class to configure new endpoint instances

JSR 356: Java API for WebSocket

ServerEndpointConfig.Configurator

- `boolean checkOrigin(String originHeaderValue)`
- `<T> T getEndpointInstance(Class<T> endpointClass)`
- `List<Extension> getNegotiatedExtension(List<Extension> installed, List<Extension> required)`
- `String getNegotiatedSubprotocol(List<String> supported, List<String> request)`
- `void modifyHandshake(ServerEndpointConfig sec, HandshakeRequest request, HandshakeResponse response)`

JSR 356: Java API for WebSocket

Custom Payloads

```
@ServerEndpoint(
    value="/hello",
    decoders={MyMessageDecoder.class},
    encoders={MyMessageEncoder.class}
)
public class MyEndpoint {
    . . .
}
```

JSR 356: Java API for WebSocket

Custom Payloads – Text Decoder

```
public class MyMessageDecoder implements Decoder.Text<MyMessage> {

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


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

  . . .
}
```

JSR 356: Java API for WebSocket

Custom Payloads – Text Encoder

```
public class MyMessageEncoder implements Encoder.Text<MyMessage> {

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

  . . .

}
```

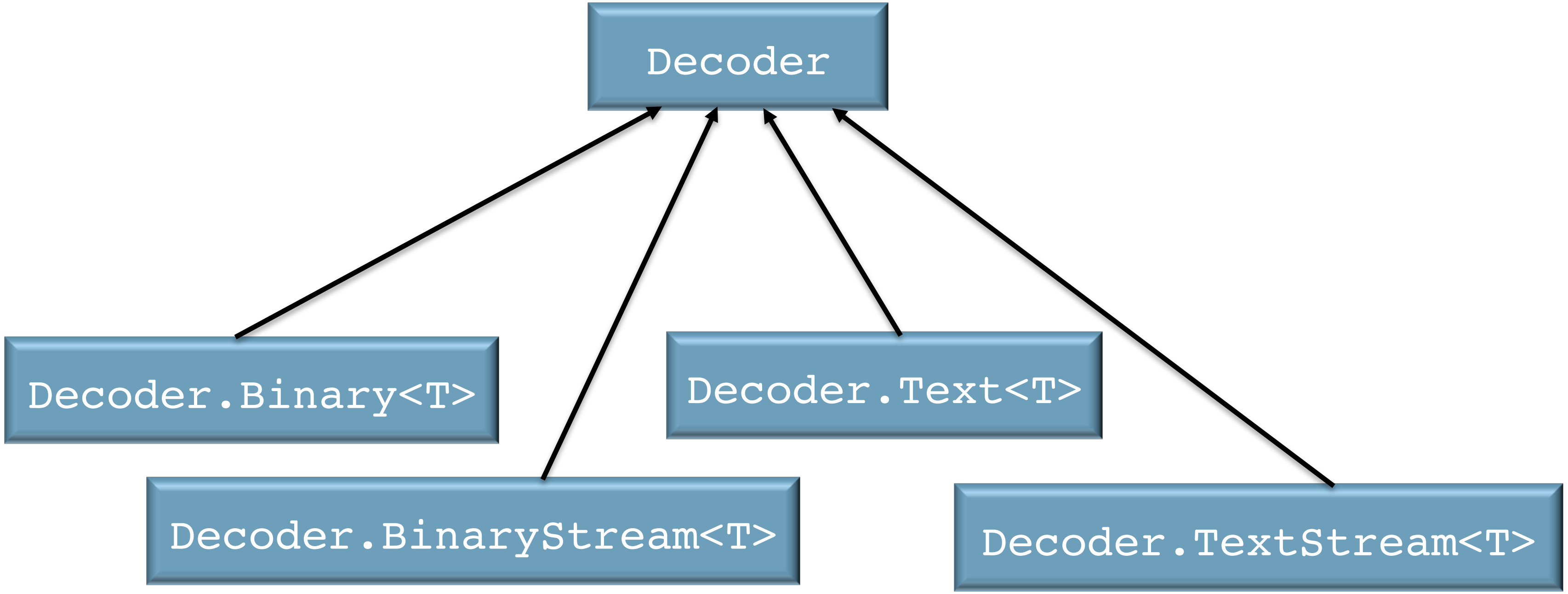
JSR 356: Java API for WebSocket

Custom Payloads – Binary Decoder

```
public class MyMessageDecoder implements Decoder.Binary<MyMessage> {

  public MyMessage decode(ByteBuffer bytes) {
    . . .
    return myMessage;

  }
  public boolean willDecode(ByteBuffer bytes) {
    . . .
    return true;  // Only if can process the payload
  }

  . . .
}
```

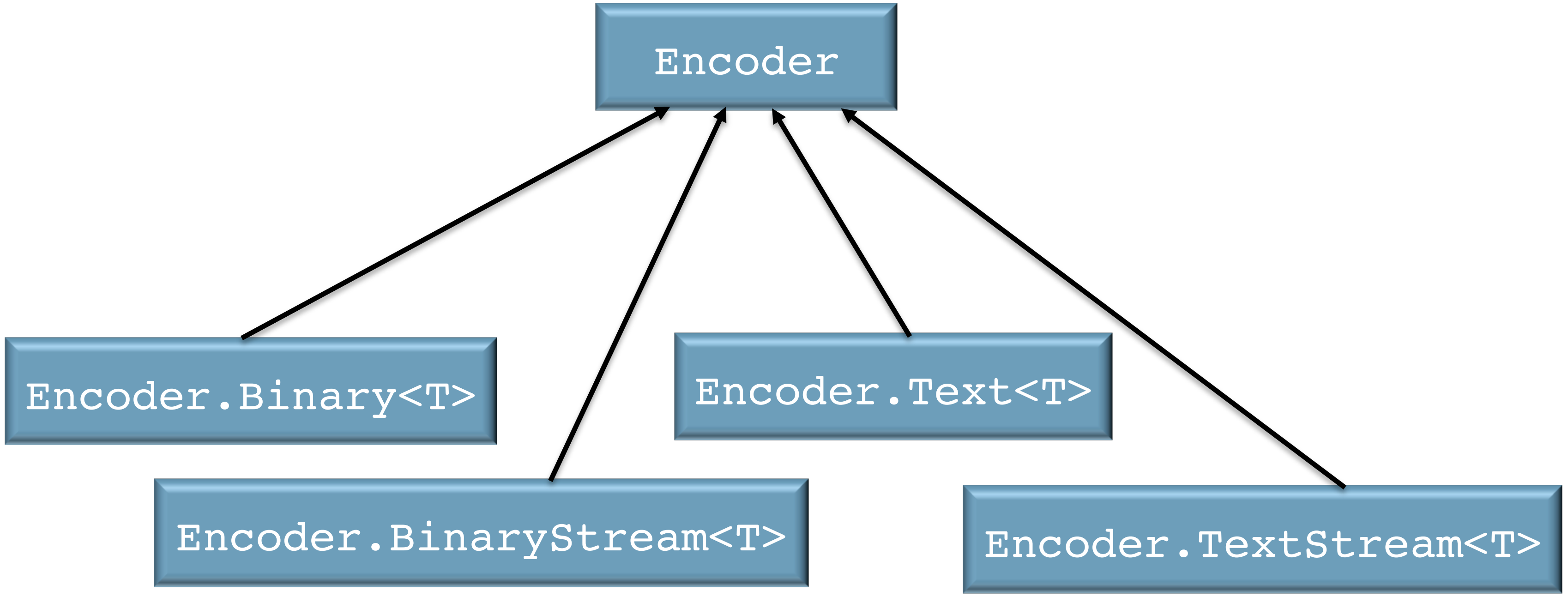
JSR 356: Java API for WebSocket

javax.websocket.Decoder



JSR 356: Java API for WebSocket

javax.websocket.Encoder



JSR 356: Java API for WebSocket

URI Template Matching

- Level 1 only

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

JSR 356: Java API for WebSocket

Which methods can be @OnMessage?

- Exactly one of the following
  - Text: `String`, Java primitive or equivalent class, `String` and `boolean`, `Reader`, any type for which there is a decoder
  - Binary: `byte[]`, `ByteBuffer`, `byte[]` and `boolean`, `ByteBuffer` and `boolean`, `InputStream`, any type for which there is a decoder
  - Pong messages: `PongMessage`
- An optional `Session` parameter
- 0..n `String` parameters annotated with `@PathParam`
- Return type: `String`, `byte[]`, `ByteBuffer`, Java primitive or class equivalent or any type for which there is a encoder

JSR 356: Java API for WebSocket

Quiz: Which methods can be @OnMessage?

- `void m(String s);`
- `void m(Float f, @PathParam(“id”)int id);`
- `Product m(Reader reader, Session s);`
- `void m(byte[] b);` or `void m(ByteBuffer b);`
- `Book m(int i, Session s, @PathParam(“isbn”)String isbn, @PathParam(“store”)String store);`
- `void m(int i, int j, int k);`

JSR 356: Java API for WebSocket

Quiz

Is the following valid?

```
@OnError
void m(int i, @PathParam(“id”)int id);
```

- `@OnError`
  - `Session?, Throwable, @PathParam String (0..n)`
- `@OnOpen`
  - `Session?, EndpointConfig?, @PathParam String (0..n)`
- `@OnClose`
  - `Session?, CloseReason?, @PathParam String (0..n)`

JSR 356: Java API for WebSocket

`javax.websocket.Session`

- `extends Closeable`
- `void addMessageHandler(MessageHandler handler)`
- `RemoteEndpoint.Async getAsyncRemote()`
- `RemoteEndpoint.Basic getBasicRemote()`
- `Set<Session> getOpenSessions()`
- `Principal getUserPrincipal()`
- `URI getRequestURI()`
- …

JSR 356: Java API for WebSocket

Example: Chat Server

```
@ServerEndpoint(”/chat")
public class ChatBean {
    …
    @OnMessage
    public void message(String message, Session session) {
        for (Session s : session.getOpenSessions()) {
            if (s.isOpen()) {
                s.getBasicRemote().sendObject(message);
            }
        }
    }
}
```

JSR 356: Java API for WebSocket

WebSocket Client

```
@ClientEndpoint
public class HelloClient {
  @OnMessage
  public void message(String message, Session session) {
    // process message from server
  }
}

WebSocketContainer c = ContainerProvider.getWebSocketContainer();
c.connectToServer(HelloClient.class, “hello”);
```

JSR 356: Java API for WebSocket

Packaging – Java EE Style

- Client side
  - Classes + resources packaged as a JAR
- Web Container
  - Only WAR packaging
  - Classes + resources packaged in `WEB-INF/classes` or `WEB-INF/lib`

Hello World and Basics
Non-POJO

JSR 356: Java API for WebSocket

Interface-driven Endpoint

```
public class MyEndpoint extends Endpoint {

  @Override
  public void onOpen(Session session) {
    session.addMessageHandler(new MessageHandler.Whole<String>()
{
        @Override
        public void onMessage(String name) {
          try {
            session.getBasicRemote().sendText(“Hello “ + name);
          } catch (IOException ex) {
          }
        }
    });
  }
}
```

JSR 356: Java API for WebSocket

Interface-driven Endpoint: Server Packaging

```
public class MyServerApplicationConfig implements
ServerApplicationConfig {
   public Set<ServerEndpointConfig> getEndpointConfigs(Set<Class<?
extends Endpoint>> endpointClasses) {
      …
      ServerEndpointConfig config = ServerEndpointConfig.Builder
            .create(MyEndpoint.class, “/foo”)
            .build();
      configs.add(config);
      return configs;
   }

   public Set<Class<?>> getAnnotatedEndpointClasses(Set<Class<?>>
scanned) { … }
}
```

DEMO

JSR 356: Java API for WebSocket

Server and Client Configuration

- Server
  - URI matching algorithm
  - Subprotocol and extension negotiation
  - Message encoders and decoders
  - Origin check
  - Handshake response
- Client
  - Requested subprotocols and extensions
  - Message encoders and decoders
  - Request URI

JSR 356: Java API for WebSocket

Relationship with Dependency Injection

- Full Dependency Injection support required in endpoints
  - Field, method, constructor injection
  - @ApplicationScoped
- Interceptors permitted too

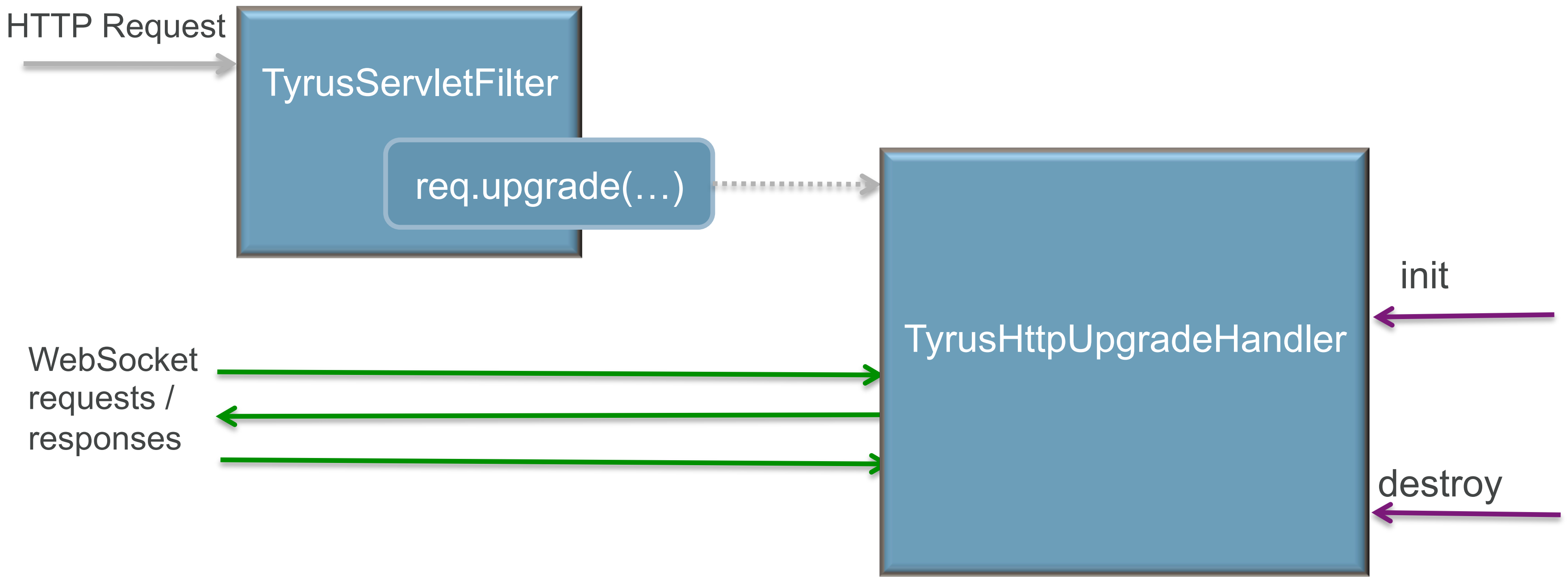
JSR 356: Java API for WebSocket

Relationship with Servlet 3.1

- Allows a portable way to upgrade HTTP request
- New API in `HttpServletRequest`
  - `<T extends HttpUpgradeHandler> T upgrade(Class<T> handlerClass) throws IOException, ServletException`

JSR 356: Java API for WebSocket

HTTP Protocol Upgrade



JSR 356: Java API for WebSocket

Security

- Authenticates using Servlet security mechanism during opening handshake
  - Endpoint mapped by `ws://` is protected using security model defined using the corresponding `http://` URI
- Authorization defined using `<security-constraint>`
- Transport Confidentiality using `wss://`
  - Access allowed over encrypted connection only

Agenda

- Primer on WebSocket
- JSR 356: Java API for WebSocket
- Summary
- Resources

Summary

JSR 356: Java API for WebSocket

- Add WebSocket protocol support to the Java EE Web Container
- API for creating WebSocket endpoints
  - Client and Server
  - Annotation and programmatic
  - Flexible message processing option
  - Integrate into the Java EE programming model

Agenda

- Primer on WebSocket
- JSR 356: Java API for WebSocket
- Summary
- Resources

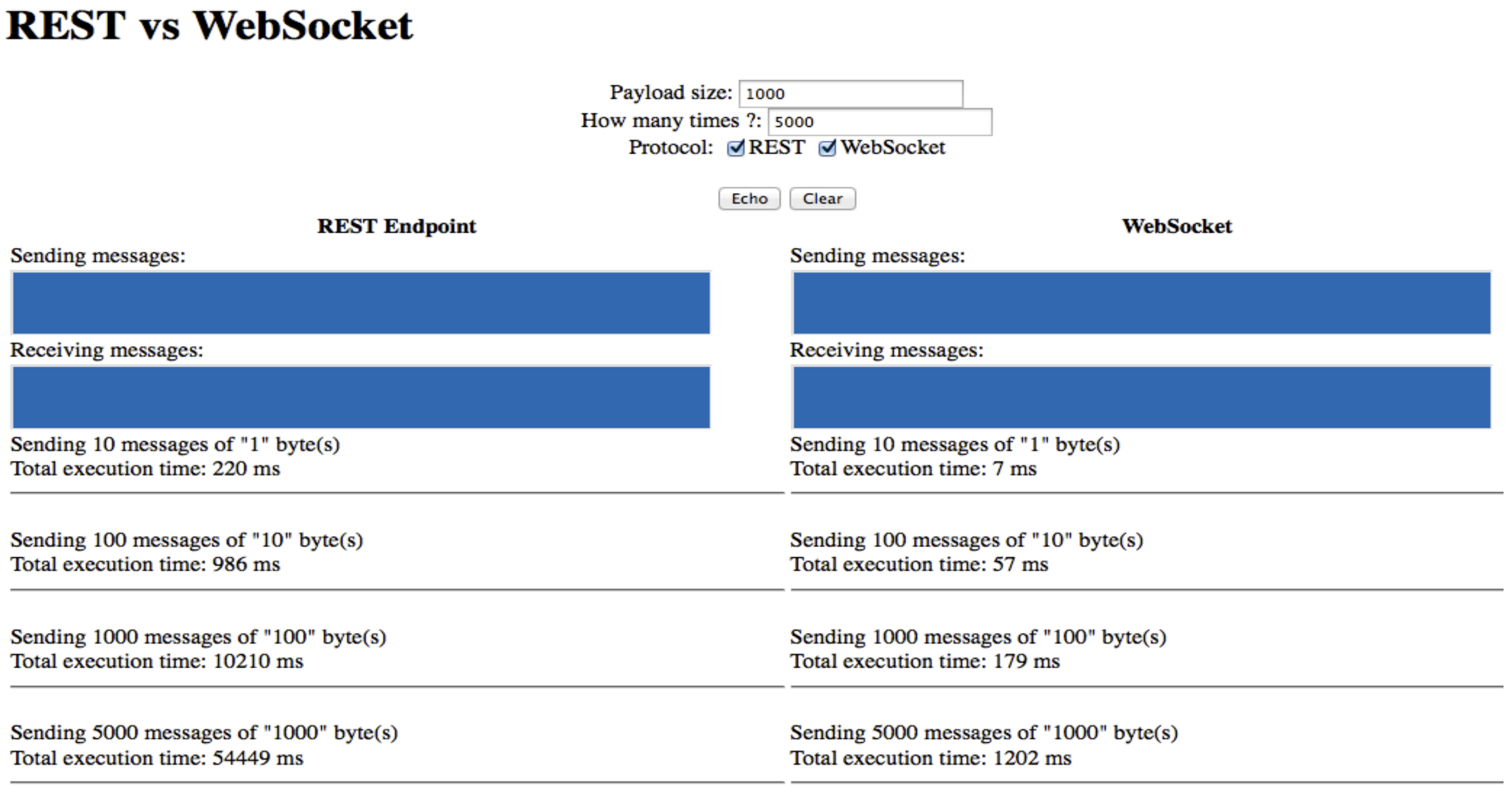
Resources

- Specification
  - JSR: jcp.org/en/jsr/detail?id=356
  - Mailing Lists, JIRA, Archive: java.net/projects/websocket-spec
  - FINAL: Part of Java EE 7
- Reference Implementation
  - Tyrus: java.net/projects/tyrus
  - Integrated in GlassFish 4 builds: glassfish.java.net

Q & A

Java™

Primer on WebSocket



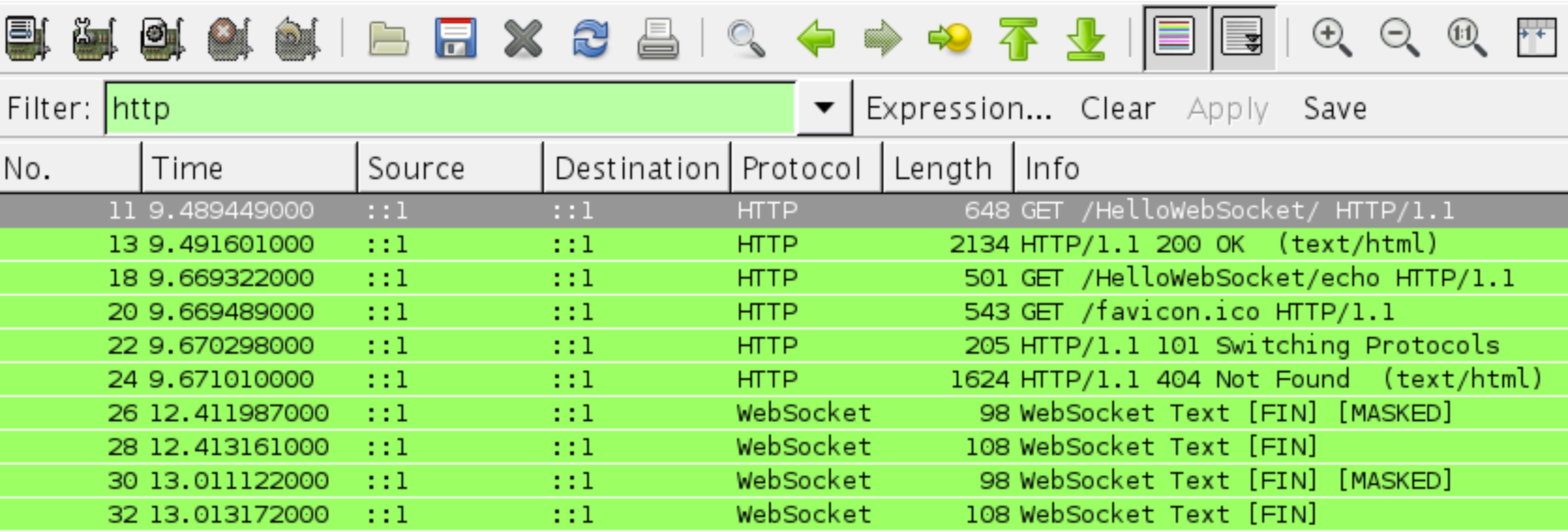
REST vs WebSocket

Payload size: 1000
How many times ?: 5000
Protocol: [✓ REST] [✓ WebSocket]

REST Endpoint	WebSocket
Sending messages:	Sending messages:
Receiving messages:	Receiving messages:
Sending 10 messages of "1" byte(s) Total execution time: 220 ms	Sending 10 messages of "1" byte(s) Total execution time: 7 ms
Sending 100 messages of "10" byte(s) Total execution time: 986 ms	Sending 100 messages of "10" byte(s) Total execution time: 57 ms
Sending 1000 messages of "100" byte(s) Total execution time: 10210 ms	Sending 1000 messages of "100" byte(s) Total execution time: 179 ms
Sending 5000 messages of "1000" byte(s) Total execution time: 54449 ms	Sending 5000 messages of "1000" byte(s) Total execution time: 1202 ms

Primer on WebSocket

How to view WebSocket messages?
Capture traffic on loopback



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]