JavaOne™

MAKE THE
FUTURE
JAVA

JMS and WebSocket for Lightweight and Efficient Messaging

Ed Bratt
Senior Development Manager, Oracle

Amy Kang
Consulting Member Technical Staff, Oracle

ORACLE®

"Safe Harbor Statement"

… please note

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract.
It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

Today's Agenda

- About JMS and JMS 2.0
- Web Scale Messaging with JMS?
- Components for WebSocket and JMS
- Extending Open MQ for WebSocket
- Messaging via WebSocket - Client Code Samples
- Pulling it all together

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

Java Message Service (JMS) and Messaging

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

JMS 101

JMS – Java Message Service

- JMS is a Java API for message exchange between systems and applications
  - Provides point to point, as well as publish and subscribe distribution
  - Messaging is asynchronous between producers and consumers
  - Producers and consumers are loosely coupled
  - Provides a rich array of service quality options
    - From “Fire and Forget” to Guaranteed, once and only once
    - And a range of options in between
- API is defined by a published standard for Java
  - Can be emulated for other languages

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JMS 101

… continued

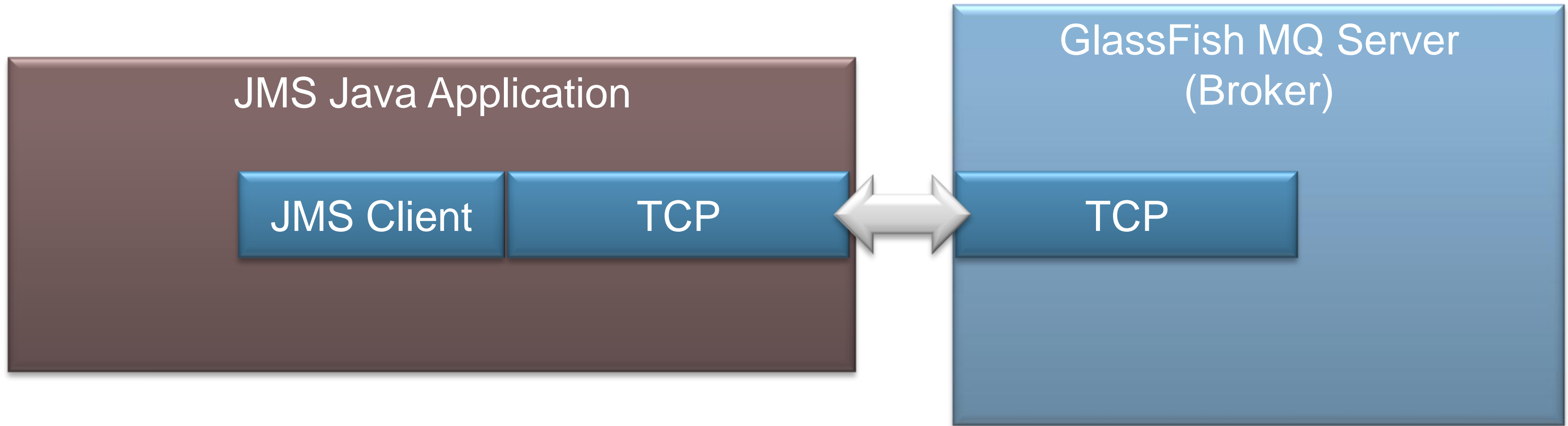
- Java clients send and receive messages using a JMS run-time
- The JMS Runtime coordinates with the JMS Server (or Broker)
- JMS Servers provide
  - Destination management
  - Message routing
    - With or without selectors
  - Persistence
  - Some offer enhanced availability
  - An array of additional features

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne ORACLE

Java Client and JMS Server

An N-Tier client – server technology



 Copyright © 2013, Oracle and/or its affiliates. All rights reserved. | CON-7777, JMS and WebSocket for Lightweight and Efficient Messaging

JavaOne ORACLE

JMS 2.0

First update to JMS API since version 1.1 in 2001

- Simplified Programming Interface (API)
  - An easier-to-use upgrade from the JMS 1.1 API
  - Introduces JMSContext, JMSProducer, JMSConsumer
  - @Inject JMSContext (Java Enterprise Edition, only)
- Additional new features, including
  - Asynchronous send
  - Delivery delay
  - Shared topic subscriptions
- More in the JMS 2.0 Spec, or in the Java EE 7 tutorial
- Nigel Deakin's talk on Tuesday (CON5919)

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

HTML5 – New Demands on Clients

Allows for rich web pages – client applications

- Clients now pull multiple streams onto a single page
  - Client pages look more and more like a full fledged applications
- These clients may benefit from messaging
- HTML5 adds WebSocket and a JavaScript interface for WebSocket
- Support in browsers is approaching ubiquity
  - Most browsers already support WebSocket
  - … including mobile browsers (though not everywhere)
- Many talks here on HTML5 – take your pick

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Web-Scale Messaging With JMS

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

Messaging, Direct to Web Clients

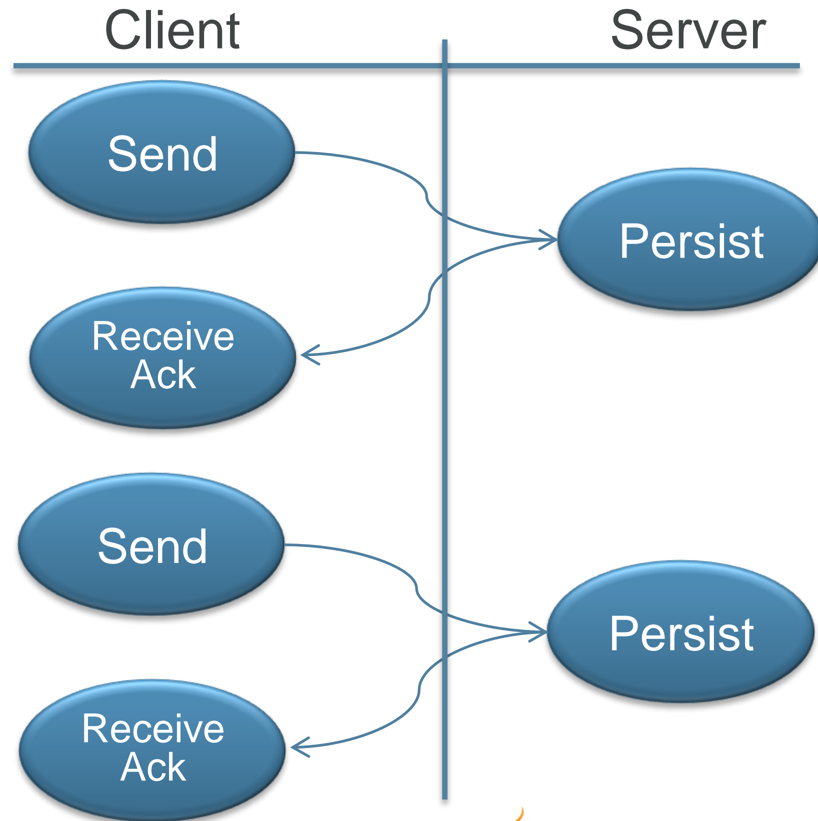
- JMS is a rich API
  - Using JMS can be a complex development effort ... Albeit less so, with the introduction of JMS 2.0
- HTTP is the most common protocol for moving data across the web
  - However it is unidirectional
- Bi-directionality can be simulated by polling
  - Which is expensive and adds complexity to clients and servers
- This can be a challenge to scaling out JMS deployments to web clients

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

Most use-cases require bi-directional communication

Client and server exchange messages and control information across their connection



 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

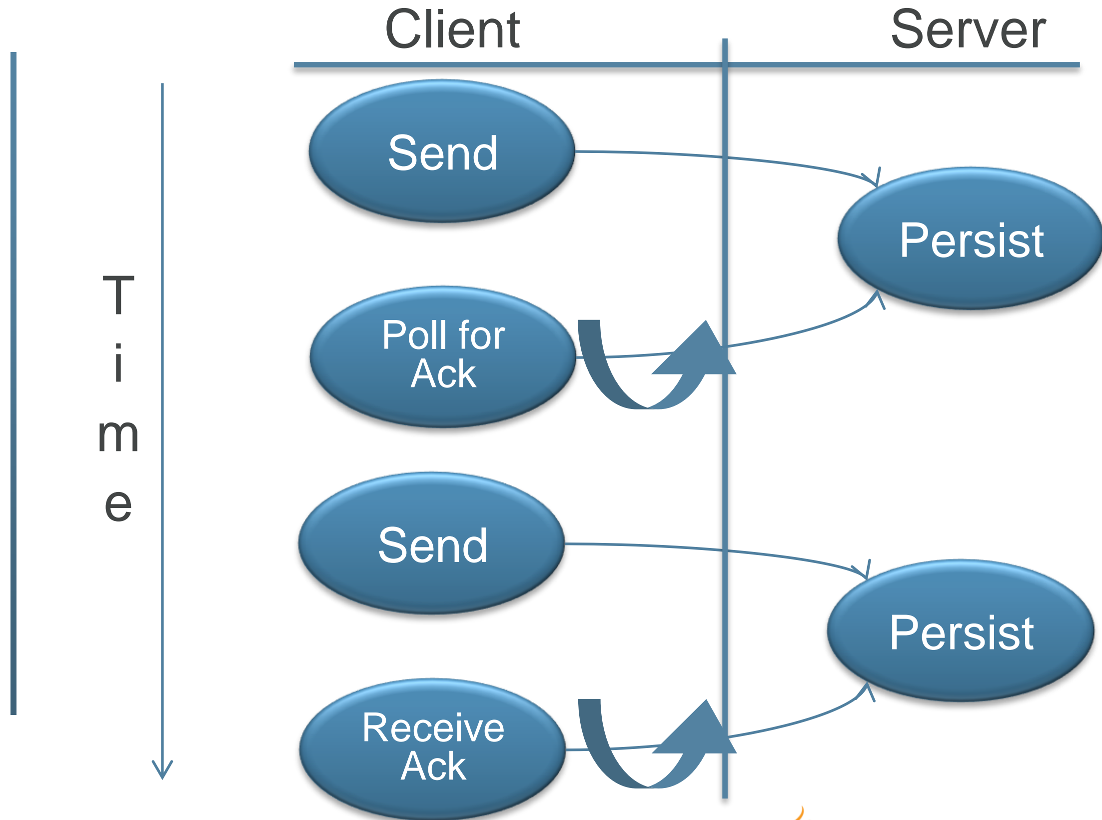
JavaOne ORACLE

Two-Way Simulation Via HTTP

Clients must poll to determine when the server has completed the request.

This takes resources of both the client and server.

This has serious scaling implications.

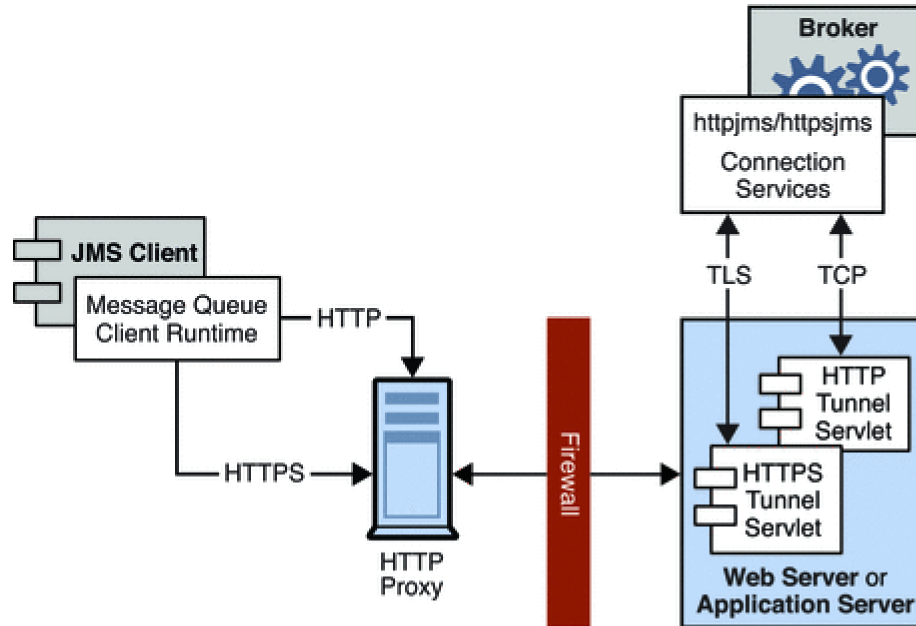


 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

HTTP Tunnel Servlet

As delivered in GlassFish Message Queue



 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

Components for WebSocket and JMS

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

WebSocket

Some basics

- "WebSocket is a web technology providing full-duplex communications channels over a single TCP connection. The WebSocket protocol was standardized by the IETF …" – *Wikipedia, August 2013*
- A WebSocket connection handshake starts as an HTTP upgrade request, then the half-duplex HTTP connection is upgraded to a full-duplex connection via a standardized protocol
- WebSocket provides a mechanism for two-way communication between web clients and servers that does not require multiple HTTP connections

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

WebSocket is also part of Java EE 7

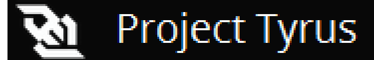
JSR356 Java API for WebSocket

- JSR356 provides an API to
  - Create and configure server and client endpoints
  - Create either programmatic endpoints as well as annotated endpoints.
  - The client API enables any Java application to access remote WebSocket endpoints (server).
- Examples for Java EE 7 WebSocket integration were given at the Hand’s on Lab HOL2147 (Tuesday … Sorry if you missed it)
  - Can download from http://glassfish.org/hol

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Tyrus

The Reference Implementation for JSR 356



- Project Tyrus is the Reference Implementation for JSR 356
- Reference Implementation of WebSocket for Java EE 7
  - https://tyrus.java.net
- Provides support in Java EE 7 Reference Implementation (GlassFish)
- Also as a stand-alone component for custom components
  - https://tyrus.java.net/documentation/1.2.1/index/getting-started.html

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

Grizzly

Provides scalable framework, utilizing Java NIO

Core Framework					HTTP Services		
HTTP	AJP	SPDY	WebSocket				
SSL/TLS	GZIP/LZMA	Thrift	Memcached			Comet	OSGi HTTP Service
Port Unification					Async HTTP Client	HTTP Multi-Part Support	JAXWS
Event Loop / FilterChain and Filters					Client	Server	
TCP/UDP Transports	Memory Mangement/ Zero-Copy Buffers	JMX Monitoring			Core Framework		

Copyright © 2013, Oracle and/or its affiliates. All rights reserved. | CON-7777, JMS and WebSocket for Lightweight and Efficient Messaging

JavaOne™ ORACLE®

Grizzly

- “The Grizzly NIO and Web framework has been designed to help developers to take advantage of the Java™ NIO API. Grizzly's goal is to help developers to build scalable and robust servers using NIO”
- Grizzly is used in
  – GlassFish Enterprise Server
  – GlassFish Message Queue (Open MQ)
  – Tyrus
  – Others …

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne ORACLE

Extending Open MQ for WebSocket – Putting It Together

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

Extending to WebSocket

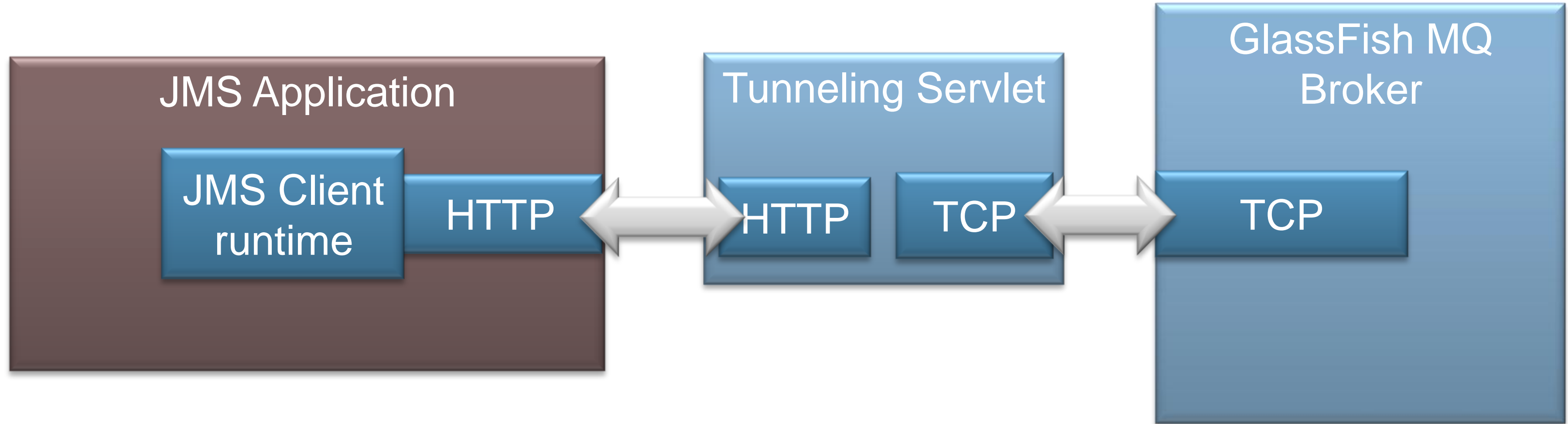
- Existing JMS client runtime and JMS broker communicates via TCP
- Grizzly provides many transports including
  - TCP/TLS
  - WebSocket (ws) or Secure WebSocket (wss)
- Tyrus provides Java API for implementation WebSocket
- We can use Grizzly and Tyrus to simplify the task of adding support for WebSocket

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

JMS over HTTP Implementation

Servlet Tunneling

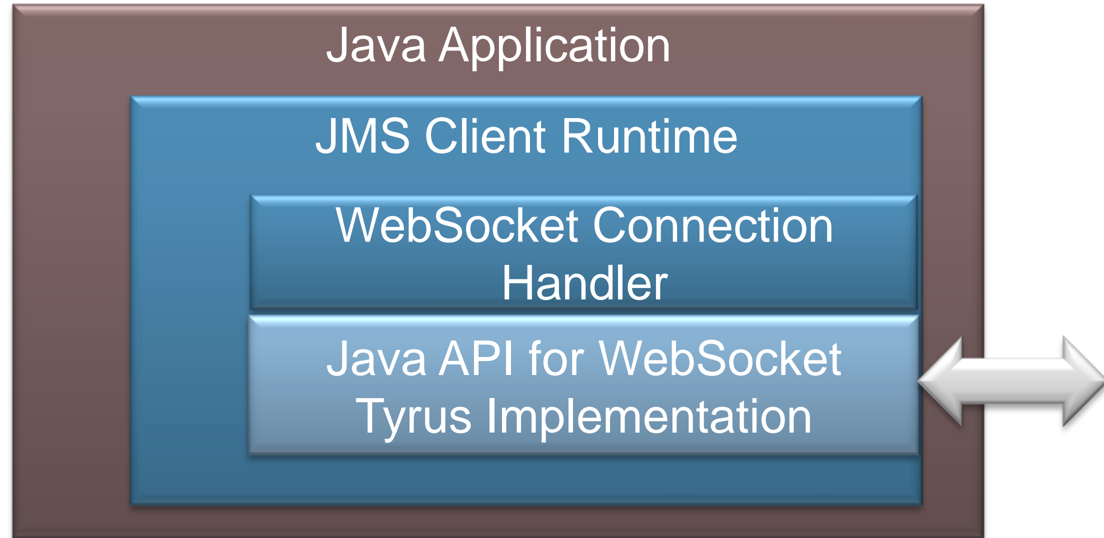


Copyright © 2013, Oracle and/or its affiliates. All rights reserved. | CON-7777, JMS and WebSocket for Lightweight and Efficient Messaging

JavaOne ORACLE

Extending the JMS Client Runtime

- GlassFish MQ JMS Runtime provides complete JMS API
  - … and includes the Tyrus provider as well



 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne ORACLE

Extending the Server

JMS Broker

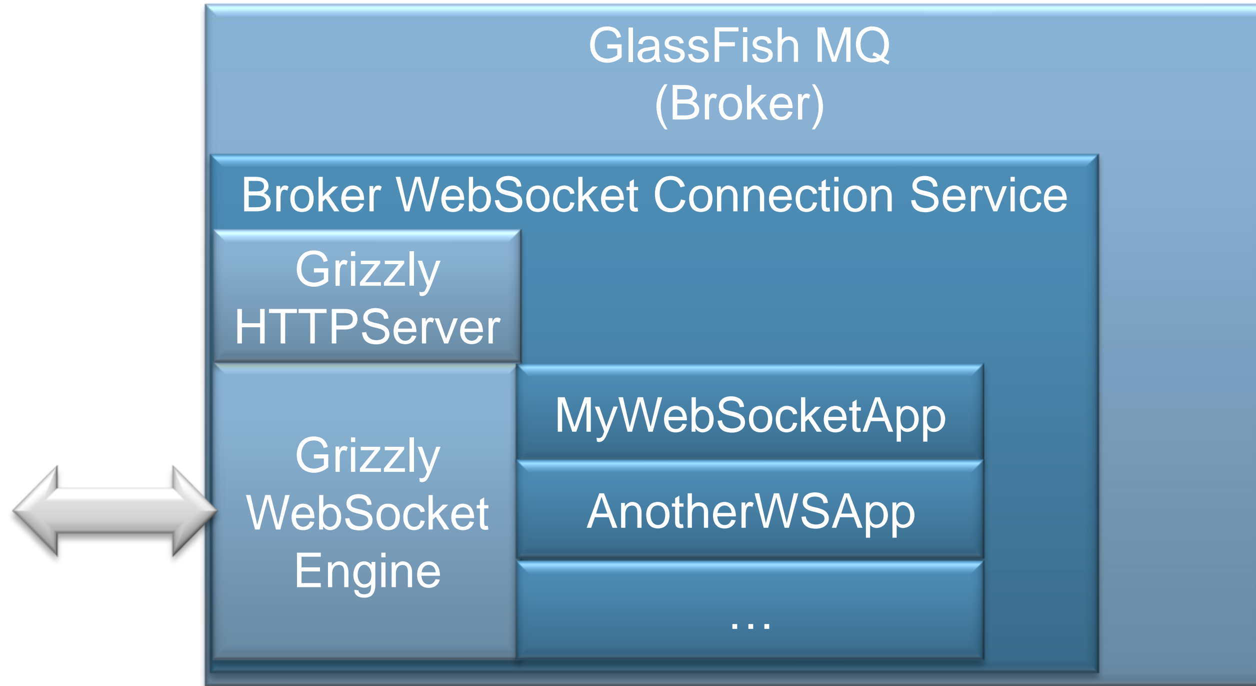
- Grizzly provides a WebSocket API
  - Send data to a remote end-point
  - Listen for events occurring on a WebSocket instance
- Grizzly defines
  - WebSocketApplication
    - for creating/implementing a server-side WebSocket application
  - WebSocketEngine
    - to register/unregister WebSocketApplication
- JMS Broker is extended to provide WebSocket connection service(s)

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

Extending the Server

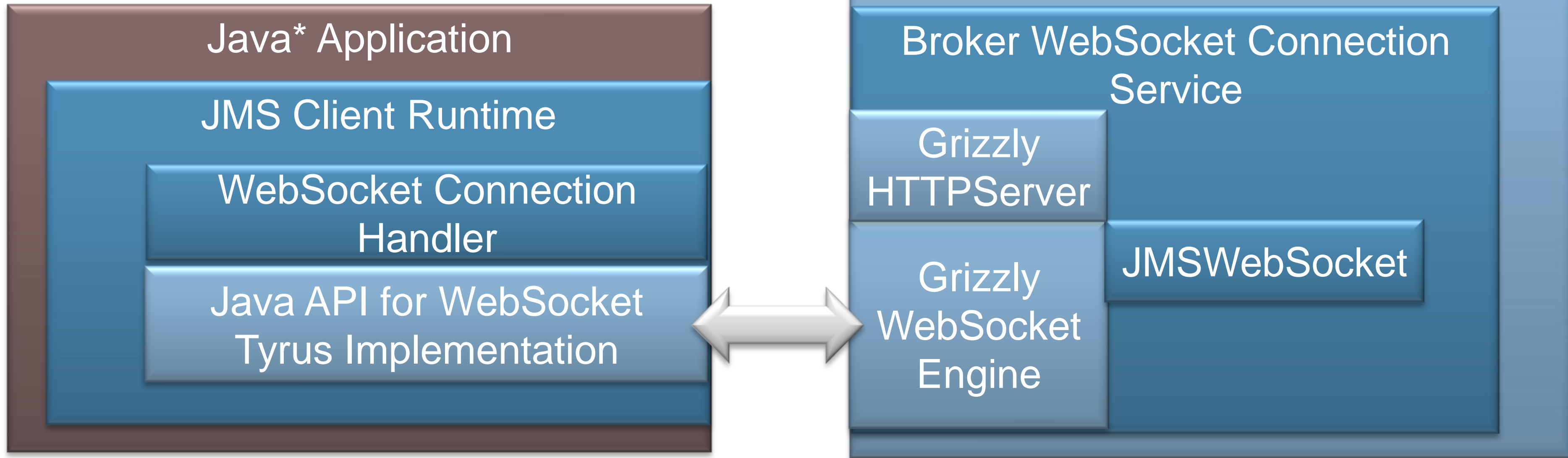
JMS Broker - WebSocket Architecture



 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

JMS over WebSocket



*Including JavaFX applications

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

STOMP over WebSocket

Streaming **T**ext **O**riented **M**essaging **P**rotocol

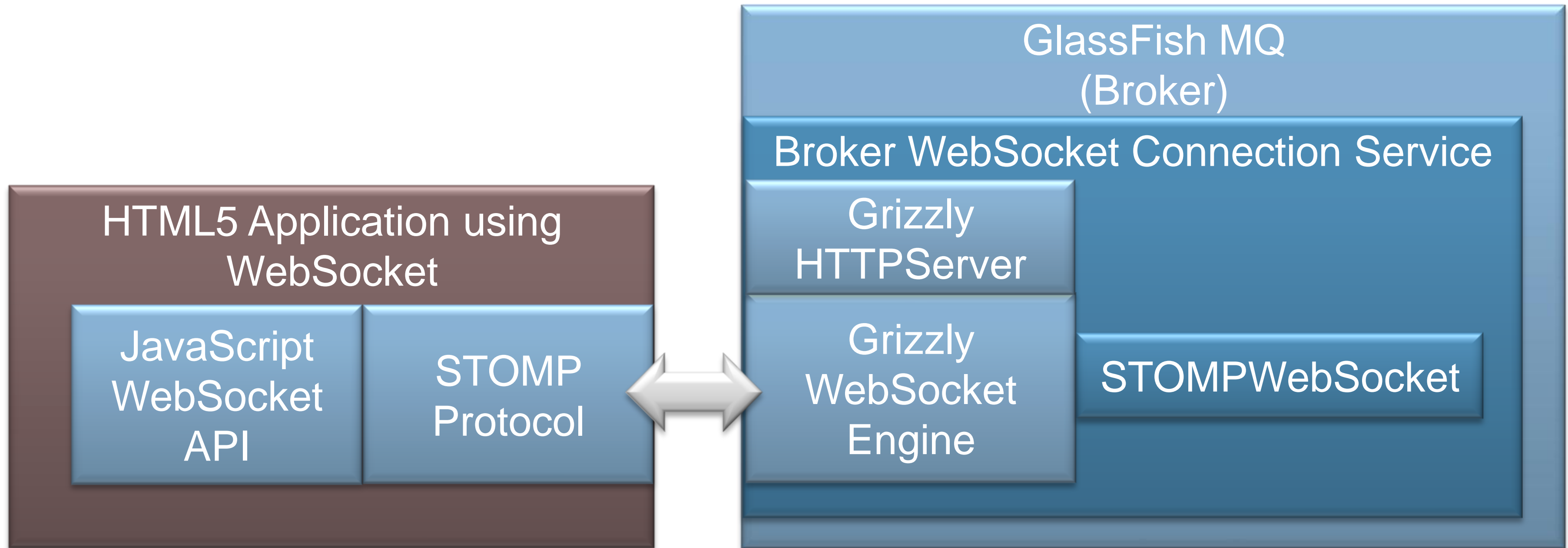
- STOMP is a simplified message exchange protocol

  “STOMP provides an interoperable wire format so that STOMP clients can communicate with any STOMP message broker to provide easy and widespread messaging interoperability among many languages, platforms and brokers.” – *stomp.github.io, Aug. 2013*

- Stomp is a protocol which can be implemented via any transport, which can support the required semantics
- Available from: http://stomp.github.io/
  - About 16 languages listed http://stomp.github.io/implementations.html
- Note: STOMP is not JMS but many JMS providers support STOMP

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

STOMP over WebSocket



 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne ORACLE

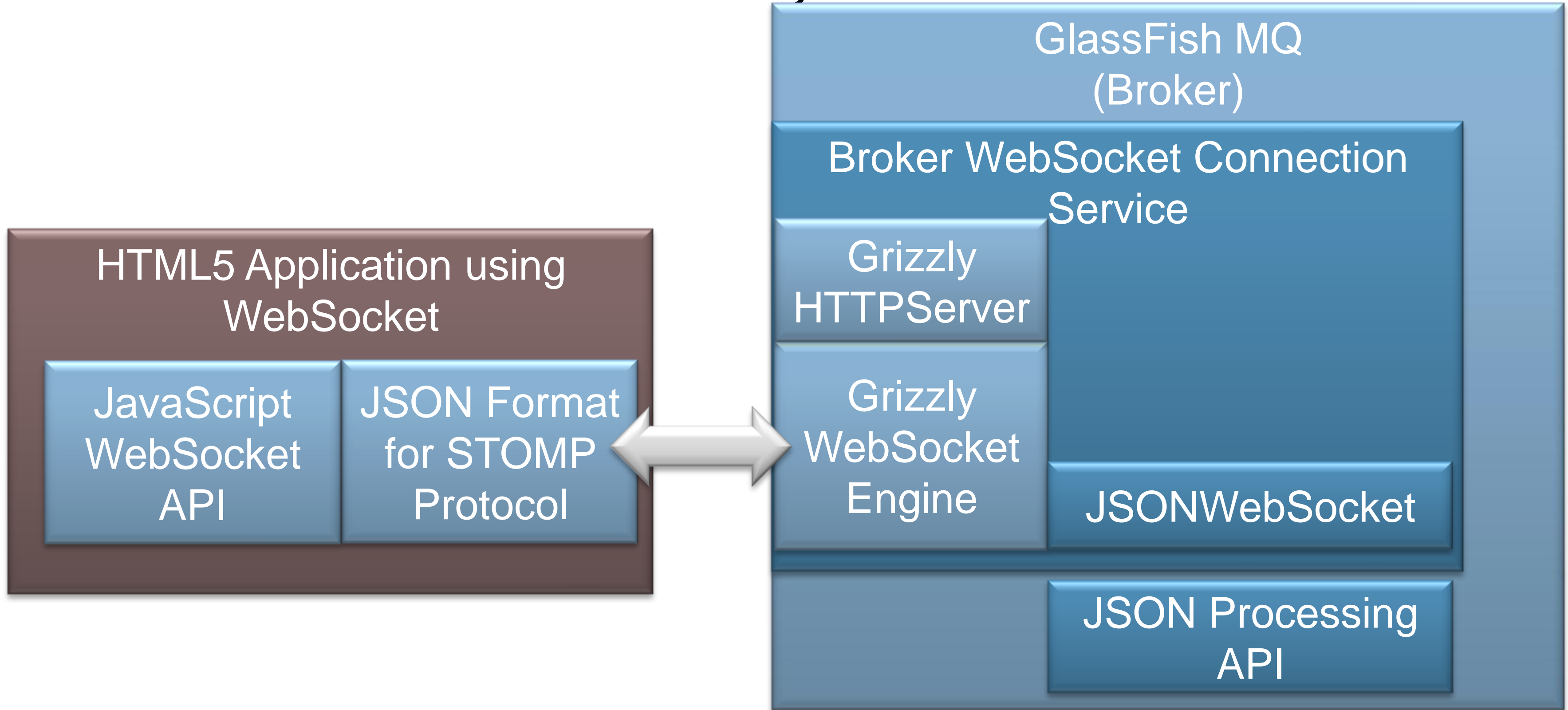
JSON and Messaging over WebSocket

"JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999" – *JSON.org, Aug. 2013*

- Language independent
  - Supported in C/C++, Perl, Python, Etc.
  - 58 languages listed at json.org

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

WebSocket and JSON, JMS

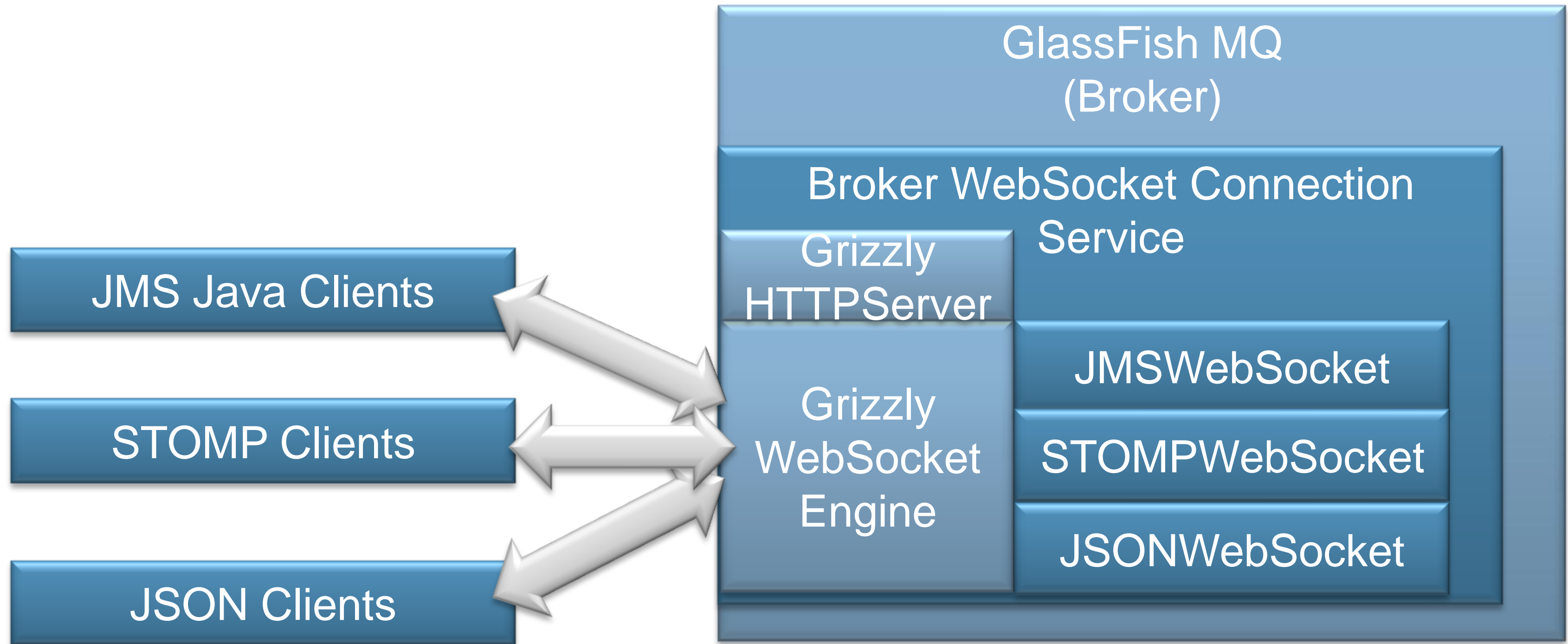


Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne ORACLE

WebSocket Architecture

Provides an array of WebSocket messaging options



 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

WebSocket Benefits

Compared to HTTP

- Greater throughput
  - At least 2x improvement

  … based on simple HTTP / SOAP comparison tests on developer class systems
- Less to manage
  - No intermediary
- Fewer resources
  - No polling … all the way around

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

Messaging via WebSocket – Client Code Samples

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

Java JMS over WebSocket

Java programs are relatively unchanged

```
ConnectionFactory cf = new com.sun.messaging.ConnectionFactory();
cf.setProperty(ConnectionConfiguration.imqAddressList,
"mqws://mybrokerhost:7670/wsjms")
JMSContext ctx  = cf.createContext();

//Send a message to 'myQueue'
JMSProducer producer = ctx.createProducer();
producer.send(ctx.createQueue("myQueue"), ctx.createTextMessage(msg));

//Receive a message from 'myQueue'
JMSConsumer consumer = ctx.createConsumer(ctx.createQueue("myQueue"));
TextMessage textMessage = (TextMessage)consumer.receive();
String payload = textMessage.getText();
System.out.println("Received message: "+payload);

//Close the JMSContext
ctx.close();
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

JavaScript for STOMP over WebSocket

- Open a WebSocket connection and setup event handlers

```
var wsUri = "ws://myServer.com:7670/wsjms/mqstomp";
websocket = new WebSocket(wsUri);
websocket.binaryType="blob";
websocket.onopen = function(evt) { onOpen(evt) };
websocket.onclose = function(evt) { onClose(evt) };
websocket.onmessage = function(evt) { onMessage(evt) };
websocket.onerror = function(evt) { onError(evt) };
```

- onOpen handler (etc.)

```
function onOpen(evt) {
     // Process Open event (e.g. write message to screen)
 }
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

JavaScript for STOMP over WebSocket

- Send a STOMP CONNECT frame to broker:
```
var NULL = '\x00';
websocket.send("CONNECT\nlogin:guest\npasscode:guest\n
accept-version:1.2\n\n"+NULL);
```
- Send a message to broker on Queue 'myQueue' using STOMP SEND frame
```
websocket.send("SEND\ndestination:/queue/myQueue\n\n
This is a message from websocket client. #"+nmsg+"\n"
+NULL);
```
- Create a subscriber on Queue 'myQueue' to receive messages from broker using STOMP SUBSCRIBE frame
```
sendData("SUBSCRIBE\ndestination:/queue/myQueue\n
id:mysubid\n\n"+NULL);
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

JavaScript for STOMP over WebSocket

- Using WebSocket onMessage() to listen for data sent from broker

```
function onMessage(evt) {
    //process evt.data
}
```

- Unsubscribe the subscriber and disconnect:

```
websocket.send("UNSUBSCRIBE\ndestination:/queue/myQueue\n
id:mysubid\n\n"+NULL);
websocket.send("DISCONNECT\n\n"+NULL);
websocket.close()
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

JavaScript for JSON over WebSocket

- Open a WebSocket connection and setup event handlers:

```
var wsUri = "ws://myServer.com:7670/wsjms/mqjsonstomp";
websocket = new WebSocket(wsUri);
websocket.onopen = function(evt) { onOpen(evt) };
websocket.onclose = function(evt) { onClose(evt) };
websocket.onmessage = function(evt) { onMessage(evt) };
websocket.onerror = function(evt) { onError(evt) };
```

- onOpen handler (etc.):

```
function onOpen(evt) {
        //Process Open event (e.g. write to screen)
}
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

JavaScript for JSON over WebSocket

- Send a STOMP CONNECT frame to broker:

```
var jmsframe = {};
jmsframe.command = "CONNECT";
jmsframe.headers = {"login":"guest“,
   "passcode":"guest", "accept-version":"1.2"};
var data = JSON.stringify(jmsframe)
websocket.send(data);
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

JavaScript for JSON over WebSocket

- Send a message to broker on Queue 'myQueue' using STOMP SEND frame:

```
var jmsframe = {}
jmsframe.command = "SEND";
jmsframe.headers = {"destination":"/queue/myQueue"};
jmsframe.body = {"text": "This is a message from
websocket json client"};
var data = JSON.stringify(jmsframe);
websocket.send(data);
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

JavaScript for JSON over WebSocket

- Create a subscriber on Queue 'myQueue' to receive messages from broker using STOMP SUBSCRIBE frame:

```
var jmsframe = {};
jmsframe.command = "SUBSCRIBE";
jmsframe.headers =
  {"destination":"/queue/myQueue", "id":"mysubid"};
var data = JSON.stringify(jmsframe);
websocket.send(data);
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaScript for JSON over WebSocket

- Using WebSocket onMessage() to listen for data sent from broker:

```
function onMessage(evt) {
    var jmsframe = JSON.parse(evt.data);
    //process jmsframe.command
    //process jmsframe.headers
    //process jmsframe.body
}
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

JavaScript for JSON over WebSocket

- Unsubscribe the subscriber and disconnect:

```
var jmsframe = {};
jmsframe.command = "UNSUBSCRIBE";
jmsframe.headers = {"destination":"/queue/myQueue", "id":"mysubid"};
var data = JSON.stringify(jmsframe);
websocket.send(data);

jmsframe = {};
jmsframe.command = "DISCONNECT";
jmsframe.headers = {"receipt-id":"mydisconnectid"};
data = JSON.stringify(jmsframe);
websocket.send(data);
websocket.close()
```

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

Pulling it all together

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Which to use?

- JMS Java API provides the richest control
  - However, the client must be a Java application or use the product specific protocol
  - Requires a JMS client runtime
- STOMP – Easy integration with many different languages and different messaging servers
- JSON – Probably the most natural for direct implementation in HTML5 clients
  - Has the most language supported

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

Try it

- Download the latest promoted build of Open MQ

https://mq.java.net/5.0.1

- Instructions for using WebSocket with Java, STOMP, and JSON-STOMP are provided

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Open Message Queue

New features, unrelated to JMS 2.0

- New shared thread-pool implementation in broker
  - As before, set threadpool_model=shared to enable
    - Uses Grizzly NIO framework
    - Supports SSL (previously only for threadpool_model=dedicated)
    - Shared threads more scalable at a cost to performance
- Improved support for DB reconnection with JDBC databases
  - In the Message Queue JDBC Connection Pool
- C Client extended with some new JMS 2.0 features
  - Shared subscriptions (durable & non-durable), Delivery delay

Copyright © 2013, Oracle and/or its affiliates. All rights reserved. | CON-7777, JMS and WebSocket for Lightweight and Efficient Messaging
JavaOne ORACLE

Summary

- JMS provides a rich Java API for systems to use for coordination
- JMS 2 extends the API by providing a simplified API as well as new features
- WebSocket extends HTTP and provides full-duplex connection providing a better throughput, over HTTP, to a variety of clients including HTML5
- Grizzly simplifies the system side coding
- Tyrus provides reference implementation for the standard Java API for WebSocket
- WebSocket provides direct integration with your JMS server – with JMS over WebSocket; STOMP over WebSocket, and STOMP via JSON over WebSocket

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

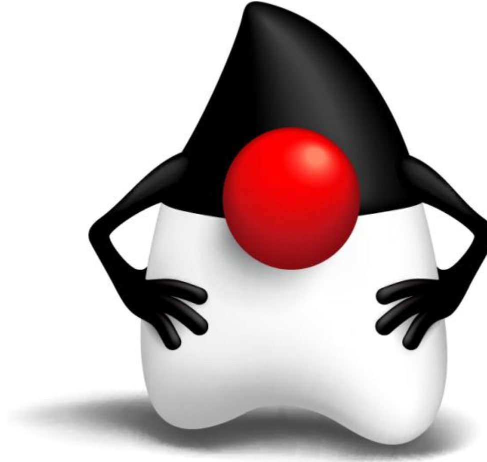
JavaOne™ ORACLE®

Where to go for more information

- Open MQ Project – https://mq.java.net
  – Download Open MQ 5.0.1 Milestone and try using JMS over WebSocket
- GlassFish – http://glassfish.org
  – Java EE Tutorial: http://docs.oracle.com/javaee/7/tutorial/doc/home.htm
- Grizzly Project – https://grizzly.java.net
- Tyrus Project – https://tyrus.java.net
- STOMP – http://stomp.github.io
- JSON – http://json.org

 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

Questions?



 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

JavaOne™ ORACLE®

MAKE THE FUTURE JAVA

ORACLE®

JavaOne™