# WebSocket and Java EE: A State of the Union

Justin Lee @evanchooly
Senior Software Engineer
Squarespace.com

# Introduction

- Java developer since 1996
- Former member of the GlassFish/Grizzly team
  - Implemented the WebSocket support for GlassFish 3.1.x and 3.2
- Currently Sr. Software Engineer at Squarespace.com
- JSR 356 (WebSocket) Expert Group Member

# Motivation

- HTML and HTTP started off as simple document exchange mechanisms
  - academics and government
- HTTP is stateless
- Once commercial interests came along, demands increased
- Simple web pages at first and then increasingly complex

# Motivation

- Heavyweight plugins: applets, flash, silverlight was designed for simple document transfer
- Asynchronous interactions are "required" in today's app environment
- Firewall friendly

# Motivation - Options

- AJAX
- COMET
- Servlet 3
- Flash/Silverlight/Applets

# Enter Websockets

- Websockets is an official IETF protocol now
  - http://tools.ietf.org/html/rfc6455
- W3C is defining the javascript API
  - http://www.w3.org/TR/websockets/
- JSR 356
  - http://www.jcp.org/en/jsr/detail?id=356

# Brief Overview

- Piggybacks on top of HTTP
- Connection upgrade request and handshake
- Asynchronous, bidirectional
- Masked transfers so it's no longer HTTP
- Extensions
  - Multiplexing
  - Compression
- Explicit subprotocols on top of websocket transport

# Client Side Options

- Built-in to most browsers
  - http://caniuse.com/websockets

| IE | Firefox | Chrome | Safari | Android | iOS |
|---|---|---|---|---|---|
| 10.0 | since 6.0 | since 14.0 | since 6.0 | native no<br>FF 15.0<br>Chrome 18.0 | 6.0 |

# Client Side Options

- Of course, not all clients need to be browsers, but still
- Development options
  - Roll your own mini-framework
  - socket.io
  - Atmosphere -- https://github.com/Atmosphere/atmosphere

# Server Side Options

jWebSocket

Autobahn WebSocket

Webbit

KAAZING

jetty://

Netty project

Atmosphere

# Server Side Options

jWebSocket

Autobahn

WebSocket

Webbit

KAAZING

jetty://

Netty project

Atmosphere

# Coming Attractions

- Look at some examples of each
- Take a look at the current JSR proposal
- Brief look at the future of websocket
  - protocol updates
  - SPDY, HTTP 2.0

# Grizzly and GlassFish

- WebSocketEngine
- WebSocketApplication
- WebSocket
- WebSocketClient
- WebSocketListener  (optional)
- And a servlet if you're on GlassFish

```
public abstract class WebSocketApplication extends
WebSocketAdapter {
    public WebSocket createSocket(ProtocolHandler handler,
        HttpRequestPacket packet, WebSocketListener... listeners)
    public void onClose(WebSocket socket, DataFrame frame)
    public void onConnect(WebSocket socket)
    public abstract boolean
        isApplicationRequest(HttpRequestPacket req)
    protected void handshake(HandShake hs)
        throws HandshakeException
    protected boolean onError(WebSocket webSocket,
        Throwable t)
}
```

```
public interface WebSocket {
    GrizzlyFuture<DataFrame> send(String/byte[] data);
    GrizzlyFuture<DataFrame> sendPing(byte[] data);
    GrizzlyFuture<DataFrame> sendPong(byte[] data);
    GrizzlyFuture<DataFrame> stream(boolean last, String fragment);
    GrizzlyFuture<DataFrame> stream(boolean last, byte[] fragment, int off,
int len);
    void close();
    boolean isConnected();
    void onConnect();
    void onMessage(String/byte[] text);
    void onFragment(boolean last, String/byte[] payload);
    void onClose(DataFrame frame);
    void onPing(DataFrame frame);
    void onPong(DataFrame frame);
    boolean add(WebSocketListener listener);
    boolean remove(WebSocketListener listener);
}
```

```
WebSocketServer server = new WebSocketServer("0.0.0.0", new PortRange(PORT));
server.register("/ping", new WebSocketApplication() {
                public boolean isApplicationRequest(HttpRequestPacket request) {
                        return (request.getRequestURI().startsWith("/ping"));
                }
                public void onPing(WebSocket socket, byte[] bytes) {
                        latch.countDown();
                }
        });

WebSocketClient client = new WebSocketClient( "ws://localhost:" + PORT + "/ping",
      new WebSocketAdapter() {
            public void onPong(WebSocket socket, byte[] bytes) {
                  latch.countDown();
            }
      });

server.start();
client.connect(5, TimeUnit.SECONDS);
client.sendPing("ping".getBytes(Charsets.UTF8_CHARSET));
```

# Tomcat

- Primarily a Tomcat *7* option
  - backported to 6
- StreamInbound/MessageInbound
- WsOutbound
- Servlet-based approach -- WebSocketServlet

```
public abstract class MessageInbound extends StreamInbound {
  void onBinaryData(InputStream is)
  void onTextData(Reader r)

  abstract void onBinaryMessage(ByteBuffer message)
  abstract void onTextMessage(CharBuffer message)
}
```

```
public class WsOutbound {
  void close(int status, ByteBuffer data);
  void close(WsFrame frame);
  void pong(ByteBuffer data);
  void writeBinaryData(int b);
  void writeBinaryMessage(ByteBuffer msgBb);
  void writeTextData(char c);
  void writeTextMessage(CharBuffer msgCb);
}
```

```
public abstract class WebSocketServlet extends HttpServlet {
  StreamInbound createWebSocketInbound(String subProtocol,
    HttpServletRequest req)
  void doGet(HttpServletRequest req,
    HttpServletResponse resp)
  String selectSubProtocol(List<String> subProtocols)
  boolean verifyOrigin(String origin)
}
```

```
public class EchoMessage extends WebSocketServlet {
  protected StreamInbound createWebSocketInbound(
    String subProtocol, HttpServletRequest request) {
    return new EchoMessageInbound(2097152,2097152);
  }
}
```

```
class EchoMessageInbound extends MessageInbound {
  EchoMessageInbound(int byteSize, int charSize) {
    super();
    setByteBufferMaxSize(byteSize);
    setCharBufferMaxSize(charSize);
  }

  void onBinaryMessage(ByteBuffer message)
    throws IOException {
    getWsOutbound().writeBinaryMessage(message);
  }

  void onTextMessage(CharBuffer message)
    throws IOException {
    getWsOutbound().writeTextMessage(message);
  }
}
```

# Jetty


- WebSocketServlet based as well
- All hand-wavy about using it directly
- WebSocketHandler
- WebSocket
  - OnTextMessage, OnBinaryMessage, OnFrame, OnControl
- No Java client

```
new WebSocketHandler() {
  public WebSocket doWebSocketConnect(HttpServletRequest req,
    String pro){
    if ("org.ietf.websocket.test-echo".equals(pro)
      || "echo".equals(pro)
      || "lws-mirror-protocol".equals(pro))
      _websocket = new TestEchoWebSocket();
    else if ("org.ietf.websocket.test-echo-broadcast".equals(pro))
      _websocket = new TestEchoBroadcastWebSocket();
    else if ("org.ietf.websocket.test-echo-assemble".equals(pro))
      _websocket = new TestEchoAssembleWebSocket();
    else if ("org.ietf.websocket.test-echo-fragment".equals(pro))
      _websocket = new TestEchoFragmentWebSocket();
    else if (pro ==null)
      _websocket = new TestWebSocket();
    return _websocket;
  }
};
```

```
public interface WebSocket {
  void onOpen(Connection connection);
  void onClose(int closeCode, String message);
  interface OnTextMessage extends WebSocket {
    void onMessage(String data);
  }

  interface OnBinaryMessage extends WebSocket {
    void onMessage(byte[] data, int offset, int length);
  }

  interface OnControl extends WebSocket {
    boolean onControl(byte controlCode, byte[] data,
      int offset, int length);
  }

  interface OnFrame extends WebSocket {
    boolean onFrame(byte flags,byte opcode, byte[] data,
      int offset, int length);
    void onHandshake(FrameConnection connection);
  }
}
```

```
class TestEchoWebSocket extends TestWebSocket {
  @Override
  public void onOpen(Connection connection) {
    super.onOpen(connection);
    connection.setMaxTextMessageSize(-1);
    connection.setMaxBinaryMessageSize(-1);
  }

  @Override
  public boolean onFrame(byte flags, byte opcode, byte[] data,
    int offset, int length) {
    super.onFrame(flags,opcode,data,offset,length);
    try {
      if (!getConnection().isControl(opcode))
      getConnection().sendFrame(flags,opcode,data,offset,length);
    } catch (IOException e) {
      e.printStackTrace();
    }

    return false;
  }
}
```

# JSR 356

- Still under development. Likely to change though pace is slowing
- Endpoint
- Sessions
- Configuration - client and server
- MessageHandler

```
public abstract class Endpoint {
  public abstract void onOpen(Session session);

  public void onClose(Session session,
    CloseReason reason) {}

  public void onError(Throwable thr, Session s) {}
}
```

```
public interface EndpointConfiguration {
  List<Encoder> getEncoders();
  List<Decoder> getDecoders();
}

public interface ClientEndpointConfiguration
  extends EndpointConfiguration {
  List<String> getPreferredSubprotocols();
  List<Extension> getExtensions();
  void beforeRequest(Map<String, List<String>>
headers);
  void afterResponse(HandshakeResponse hr);
}
```

```
public interface ServerEndpointConfiguration
  extends EndpointConfiguration {
  Class<? extends Endpoint> getEndpointClass();
  String getNegotiatedSubprotocol(List<String>
    requestedSubprotocols);
  List<Extension> getNegotiatedExtensions(
    List<Extension> requestedExtensions);
  boolean checkOrigin(String originHeaderValue);
  boolean matchesURI(URI uri);
  void modifyHandshake(HandshakeRequest request,
    HandshakeResponse response);
  String getPath();
}
```

```
public interface MessageHandler {
  /**
   * The allowed types for T:
   * java.lang.String and java.io.Reader for text messages
   * java.nio.ByteBuffer, byte[] java.io.InputStream for binary
   * PongMessage for representing pongs
   * any developer object that has a Decoder configured
   */
  interface Basic<T> extends MessageHandler {
    void onMessage(T message);
  }

  /**
   * The allowed types for T:
   * java.lang.String and java.io.Reader for text messages
   * java.nio.ByteBuffer, byte[] java.io.InputStream for binary
   */
  interface Async<T> extends MessageHandler {
    void onMessage(T partialMessage, boolean last);
  }
}
```

```
public class HelloMain {
  public static void main(String args[]) throws Exception {
    // create a server
    ServerEndpointConfiguration serverConfig =
      new DefaultServerConfiguration(HelloEndpoint.class,
      "/hello");
    ServerContainer serverContainer = /* mumble mumble */

    // create a client
    ContainerProvider.createClientContainer()
      .connectToServer(HelloClient.class,
      new DefaultClientConfiguration(), “/hello);
  }
}
```

```
public class HelloServer extends Endpoint {
  @Override
  public void onOpen(Session session,
    EndpointConfiguration config) {
    final RemoteEndpoint remote =
      session.getRemote();
    session.addMessageHandler(
      new MessageHandler.Basic<String>() {
        public void onMessage(String text) {
          try {
            remote.sendString("Got your message ("
              + text + "). Thanks !");
          } catch (IOException ioe) {
            ioe.printStackTrace();
          }
        }
      });
  }
}
```

```
public class HelloClient extends Endpoint {
  public void onOpen(Session session) {
    session.addMessageHandler(
      new MessageHandler.Text() {
      public void onMessage(String text) {
        System.out.println("Received: " + text);
      }
    });
    try {
      session.getRemote().sendString("Oh, you!");
    } catch (IOException ioe) {
      ioe.printStackTrace();
    }
  }
}
```

```
public class CustomMessageReceive extends Endpoint {
  public void onOpen(Session session,
    EndpointConfiguration config) {
    config.getDecoders().add(
      new Decoder.Text<Person>() {
      Person decode(String s) throws DecodeException{
        return JSON.parse(s);
      }
    });
  }

  public void onError(Session session, Throwable e) {
    if (e instanceof DecodeException) {
      // diagnose translation error
    } else {
      // diagnose ioexception (wire snipped...)
    }
  }
}
```

# But wait there's more!

- Annotations
  - @WebSocketEndpoint
  - @WebSocketOpen
  - @WebSocketClose
  - @WebSocketMessage
  - @WebSocketError
- Encoding APIs

```
public class HelloServer extends Endpoint {
  public void onOpen(Session session) {
    final RemoteEndpoint remote =
      session.getRemote();
    session.addMessageHandler(
      new MessageHandler.Text() {
        public void onMessage(String text) {
          try {
            remote.sendString("Got your message ("
              + text + "). Thanks !");
          } catch (IOException ioe) {
            ioe.printStackTrace();
          }
        }
      });
  }
}
```

```
@WebSocketEndpoint(“/hello/{message}”)
public class MyHelloServer {
  @WebSocketMessage
  public String doListen(
    @WebSocketPathParam("message") msg) {
    return "Got your message (" + msg
      + "). Thanks !";
  }
}
```

# Coming up

- Extension support
  - Custom framing support
  - Already in the API but hasn't seen much discussion yet
- Multiplexing
- Compression

# Resources

- tools.ietf.org/html/rfc6455
- http://jcp.org/en/jsr/detail?id=356
- http://java.net/projects/websocket-spec
- http://www.websocket.org
- http://grizzly.java.net
- http://tomcat.apache.org
- http://eclipse.org/jetty

# WebSocket and Java EE: A State of the Union

Justin Lee @evanchooly
http://antwerkz.com
http://github.com/evanchooly
Squarespace.com