Socket to them!

Tutorial: Pushing browser updates using WebSockets in Glassfish

April 22, 2013 SteveMillidge

#api (http://jaxenter.com/tag/api) #c2b2 (http://jaxenter.com/tag/c2b2) #glassfish (http://jaxenter.com/tag/glassfish) #grizzly (http://jaxenter.com/tag/grizzly) #highcharts (http://jaxenter.com/tag/highcharts) #html5 (http://jaxenter.com/tag/html5) #java (http://jaxenter.com/tag/java-2) #java ee (http://jaxenter.com/tag/java-ee) #javascript (http://jaxenter.com/tag/javascript) #jboss (http://jaxenter.com/tag/jboss) #jcp (http://jaxenter.com/tag/jcp) #jee 7 (http://jaxenter.com/tag/jee-7) #json (http://jaxenter.com/tag/json) #jsr (http://jaxenter.com/tag/jsr) #oracle (http://jaxenter.com/tag/oracle) #pojo (http://jaxenter.com/tag/pojo) #updates (http://jaxenter.com/tag/updates) #websockets (http://jaxenter.com/tag/websockets)

Steve Millidge gives us a WebSocket taster using a combination of languages and a popular Java server.

Steve Millidge gives us a WebSocket taster using a combination of languages and a popular Java server, in this tutorial from February's JAXMagazine.

Websockets are revolutionising HTML5 and provide the capability to establish a full duplex connection between the web server and the web browser. This means for the first time we can write applications to push updates to the browser directly from the server without having to use complex hacks like long polling, Comet or third party plugins like Flash.

In this tutorial I'll demonstrate pushing stock "updates" to a browser over Websockets to asynchronously update a stock price graph purely using the push capabilities inherent in Websockets. I'll also use GlassFish in this tutorial as this has out of the box Websockets support in the latest production GlassFish 3.1.2.2 and can therefore be built and deployed now. However WebSocket support is not enabled out of the box. WebSocket support can be enabled via the administration console, but the simplest way is to use an asadmin command;

```
asadmin set configs.config.server-config.network-
config.protocols.protocol.http-listener-1.http.websockets-
support-enabled=true
```

To keep things short, in this tutorial, our application will just spawn a thread to create random updates to the Stock price. However in a real application, it would be simple to hook our application to a data feed via JMS or some other mechanism.

Side bar?

The Java API for Websockets is being standardised in the JCP under JSR 356 (http://www.jcp.org/en/jsr/detail?id=356). Currently application servers use a proprietary api to unlock Websockets functionality and the GlassFish api here is specific to GlassFish. Tomcat and other servers have a different API. If you are interested in the proposed JEE7 Websockets API head over to the JCP page to take a look.

Websocket is supported in GlassFish thanks to the Grizzly library. The key classes in the Grizzly Websocket API we need are shown in **Figure 1.**

Figure 1: Key classes in the Grizzly Websocket API

StockSocket class

Working from the bottom up, first we must create a derived class of the Grizzly WebSocket class. This class will implement the protocol between the browser and GlassFish. As we'll see later one instance of this class is created for each client browser. In our class, we will implement Runnable, spawn a Thread and send updates to the browser. In this code (shown in **Listing 1**), we will take advantage of the Grizzly provided DefaultWebSocket class. This implements all the methods of the WebSocket interface with no-ops, so we can just override the methods we are interested in.

Listing 1

```
public class StockSocket extends DefaultWebSocket implements Runnable
{
        private Thread myThread;
        private boolean connected = false;
public StockSocket(ProtocolHandler protocolHandler, WebSocketListener... listeners) {
        super(protocolHandler, listeners);
```

In the onConnect method (which is called when a client browser connects to the server) we need to create a new Thread and pass the Websocket instance as the Runnable, as shown in **Listing 2.**

Listing 2: onConnect

```
@Override     public void onConnect() {         myThread = new Thread(this);        conne
cted = true;  myThread.start();         super.onConnect();     }
```

In the run method (**Listing 3**), we will periodically call our custom sendUpdate method using a random value for a Stock. Our Stock class is a simple Serializable POJO DTO with three attributes, name, description and price.

Listing 3: Run

```
public void run() {
while(connected) {
           Stock stock = new Stock("C2B2","C2B2",Math.random() * 100.0);
           int sleepTime = (int)(500*Math.random() + 500);               Thread.currentTh
read().sleep(sleepTime);
sendUpdate(stock);
  }
}
```

In the sendUpdate method (**Listing 4**), we serialize the Stock object using the Jackson library into a JSON string. We then send this to the browser over WebSockets by calling the Grizzly base class' send method which writes the JSON string down to the browser.

Listing 4: sendUpdate

```
public void sendUpdate(Stock stock) {
       // CONVERT to JSON
       ObjectMapper mapper = new ObjectMapper();
       StringWriter writer = new StringWriter();
       try {
           mapper.writeValue(writer, stock);
       } catch (IOException ex) {
       }
       String jsonStr = writer.toString();
       // SEND down the Websocket
       send(jsonStr);
    }
```

Finally in our onClose method (**Listing 5**) we will notify the thread to stop by setting connected to false.

Listing 5: onClose

```
@Override
    public void onClose(DataFrame frame) {
        connected = false;
        super.onClose(frame);
    }
}
```

StockApplication class

To hook our derived StockSocket class into the GlassFish server, we need to create a derived WebsocketApplication class, shown below;

```
public class StockApplication extends WebSocketApplication
{
```

In this class there are two methods we need to override. The first is isApplicationRequest which is called by GlassFish when a client browser connects to GlassFish over the Websocket protocol (**Listing 6**). Our application needs to check the request and decide whether it wants to accept the connection. In this case, we will check whether the context path of the Websocket request contains the string "/stocks". If so we need to tell GlassFish that the request is for us by returning true.

Listing 6:isApplicationRequest

```
  @Override
    public boolean isApplicationRequest(Request request)
    {
        if (request.requestURI().toString().endsWith("/stocks"))
        {
            return true;
        }
        else
        {
            return false;
        }
    }
```

The second method is createWebSocket (**Listing 7**). Here, we need to create and return an instance of our StockSocket class described above. The createWebSocket method is called by GlassFish when a client browser connects to our application and we have accepted the request.

Listing 7: createWebSocket

```
  @Override
    public WebSocket createWebSocket(ProtocolHandler protocolHandler, WebSocketListener..
. listeners)
    {
        return new StockSocket(protocolHandler, listeners);
    }
```

StockServlet Class

The final class we need to write is a simple servlet. This servlet only exists to ensure we register our derived StockApplication class with Grizzly's WebSocketEngine.

Listing 8: StockServlet

```
@WebServlet(name = "StockServlet", urlPatterns =

{

"/stocks"

}, loadOnStartup = 1)

public class StockServlet extends HttpServlet {

private StockApplication pushApp;
```

We do this in the init method of our servlet and to ensure our servlet is created on deployment, we must specify that an instance should be created on startup, in the annotations as shown above.

```
@Override
public void init(ServletConfig config) throws ServletException {
super.init(config);
pushApp = new StockApplication();
WebSocketEngine.getEngine().register(pushApp);
}
```

We also need to override the destroy method to ensure our StockApplication is unregistered from the WebSocketEngine on undeployment.

```
@Override

public void destroy()

{

super.destroy();

WebSocketEngine.getEngine().unregister(pushApp);

}

}
```

HTML5 & Javascript

Once we have written our Java servlet and the classes to use Grizzly's WebSocket API, we need to turn our mind to the HTML and Javascript code. For our demonstration we are going to use a Javascript library called Highcharts (http://www.highcharts.com/) which is free for non-commercial use. This Javascript library can render very sexy Charts purely by using HTML5.

For our browser we will create a simple JSP page which uses the Websocket Javascript API to connect to GlassFish over the Websocket protocol and then receives our JSON stock updates which it feeds to HighCharts to graph.

The first thing you need to do in the WebSockets Javascript API is to connect to the GlassFish server, using the Websocket protocol. To do this, we need to create a URL of the form ws://<host>:<port>/<context> and pass this to the constructor of the WebSocket class.

```
<script type="text/javascript">

var wsUri = "ws://" + location.host +
"${pageContext.request.contextPath}/stocks";

websocket = new WebSocket(wsUri);
```

Once we have our Websocket object, we then must set up the call back functions. These Javascript functions are called by the browser when Websocket events occur, for example, when the socket is opened (onOpen), closed (onClose) or there is an error (onError). For simplicity, we will set these as empty functions.

```
websocket.onopen = function(event) { };

websocket.onclose = function(event) { };

websocket.onerror = function(event) { };
```

The most important callback is onmessage. This is triggered when the browser receives data from the server over the Websocket, and in our case will be called when we receive the JSON string representing the stock object. So we will parse the JSON string and create a new datapoint in HighCharts for this Stock price update.

```
websocket.onmessage = function(event) {

var object = JSON.parse(event.data);

var x = (new Date()).getTime();

var y = object.price;

document.chart.series[0].addPoint([x,y],true,true,false);

}

</script>
```

The initialisation of the HighCharts chart is done in the head of the document, a snippet of which is shown below.

Listing 9: HighCharts

```
<script type="text/javascript">
$(document).ready(function() {
        Highcharts.setOptions({
                global: {
                        useUTC: false
                }
        });
        var chart;
        document.chart = new Highcharts.Chart({
       …
});
</script>
```

The JSP page should be packaged up into a war file, with the servlet and Java Grizzly code shown above and deployed to your GlassFish server in the usual way.

Final View

Once the code is deployed successfully, you can navigate to it using your usual browser and you should see an updating chart.

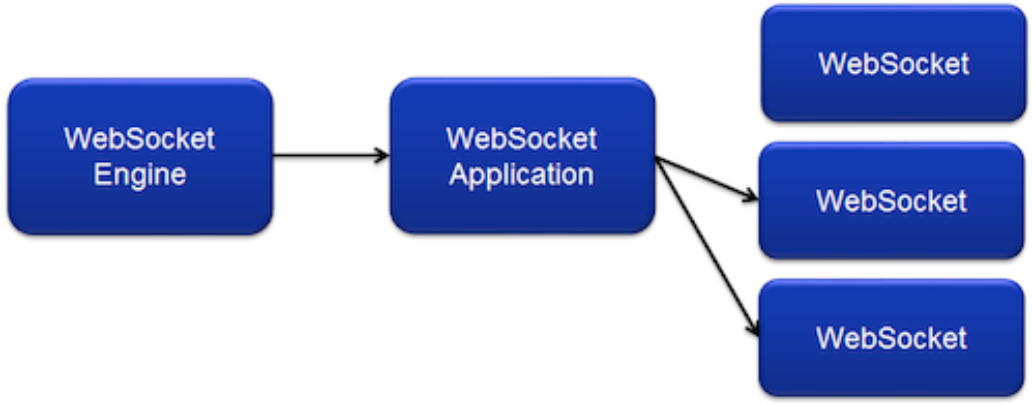

Figure 2: Updating Chart

Building push applications using the standard WebSocket Javascript API and modern application servers like GlassFish is very easy to do. Hopefully this tutorial has whetted your appetite and inspired you to explore WebSockets in your applications.

Steve Millidge is the director and founder of C2B2 Consulting Limited, he has used Java extensively since pre1.0 and has been a field based professional service consultant for over 10 years. Through C2B2 he now focuses on the configuration of JEE and SOA infrastructure for maximum Scalability, Performance, Availability, Recoverability, Manageability and Security. Having worked for and on behalf of Oracle, BEA and Red Hat professional services he has extensive experience of deploying large scale production systems. Steve has spoken at a number of events including Java One, Jax London, UK Oracle User Group Conference, The Server SOA, Cloud & Service Technology Symposium, JBoss World; he is the main organiser of the London JBoss User Group and regularly presents brown bag technical sessions for C2B2's customer base.

This article first appeared in **JAX Magazine: Socket to them!** in February 2013. Download that issue and others here. (http://jaxenter.com/jax-magazine/)

Image courtesy of cote (http://www.flickr.com/photos/cote/)

Be the first to share this article with your network!

Author

SteveMillidge
All Posts by SteveMillidge (http://jaxenter.com/author/stevemillidge)

Steve Millidge is the director and founder of C2B2 Consulting Limited, he has used Java extensively since pre1.0 and has been a field based professional service consultant for over 10 years. Through C2B2 he now

focuses on the configuration of JEE and SOA infrastructure for maximum Scalability, Performance, Availability, Recoverability, Manageability and Security. Having worked for and on behalf of Oracle, BEA and Red Hat professional services he has extensive experience of deploying large scale production systems. Steve has spoken at a number of events including Java One, Jax London, UK Oracle User Group Conference, The Server SOA, Cloud & Service Technology Symposium, JBoss World; he is the main organiser of the London JBoss User Group and regularly presents brown bag technical sessions for C2B2's customer base.

Comments

YOLO Driven Development (and other hilarious IT methodologies) (http://jaxenter.com/yolo-driven-development-methodology-113469.html)

JAX Finance – the European conference for financial technology (http://jaxenter.com/jax-finance-european-conference-financial-technology-113406.html)

Why agile doesn't scale (and what you can do about it) (http://jaxenter.com/agile-doesnt-scale-113177.html)

JAX EVENTS

(http://jax-finance.com/2015/call-for-paper/?utm_medium=banner&utm_source=jaxenter&utm_campaign=jaxfinance_sands)

TRENDING POSTS

(http://jaxenter.com/eclipse-netbeans-or-intellij-which-is-the-best-java-ide-107980.html)

Eclipse, NetBeans or IntelliJ: Which is the best Java IDE? (http://jaxenter.com/eclipse-netbeans-or-intellij-which-is-the-best-java-ide-107980.html)

(http://jaxenter.com/motivates-developers-hint-probably-money-113347.html)

What motivates developers? (Hint: probably not money) (http://jaxenter.com/motivates-developers-hint-probably-money-

113347.html)

A sneak peek at the radically new Angular 2.0 (http://jaxenter.com/angular-2-0-112094.html)

Why is my database slowing down? (http://jaxenter.com/why-is-my-database-slowing-down-107919.html)

Topics

Agile (http://jaxenter.com/news/agile)

Architecture (http://jaxenter.com/news/architecture)

Big Data (http://jaxenter.com/news/big-data-news)

Career (http://jaxenter.com/news/career-news)

Cloud (http://jaxenter.com/news/cloud)

Eclipse (http://jaxenter.com/news/eclipse)

Editor's Pick (http://jaxenter.com/news/editors-pick)

IoT (http://jaxenter.com/news/iot-news)

Java (http://jaxenter.com/news/java)

Web (http://jaxenter.com/news/web)

Pages

Authors (http://jaxenter.com/authors)

Contact (http://jaxenter.com/contact)

Found a bug? (http://jaxenter.com/found-bug)

Advertise (http://jaxenter.com/advertise)

Privacy Policy (http://jaxenter.com/privacy-policy)

Terms of Use (http://jaxenter.com/terms)

Imprint (http://jaxenter.com/imprint)

Follow JAXenter

Twitter (https://twitter.com/jaxentercom)

Facebook (https://www.facebook.com/pages/JAXentercom/123294781032857?fref=ts)

Google+ (https://plus.google.com/108137535157006873567/)

RSS (http://http://jaxenter.com/feed)

S&S Media

JAXenter.de (http://jaxenter.de/)

JAX Finance (http://jax-finance.com/)

JAX London (http://http://jaxlondon.com/)

JAX Germany (http://jax.de/)

Developer.Press (http://developerpress.com/)

International PHP Conference (http://phpconference.com/2014/en)

Webinale (http://webinale.de/2015/)

WebMagazin (http://webmagazin.de/en)

S&S Media (http://sandsmedia.com/en)