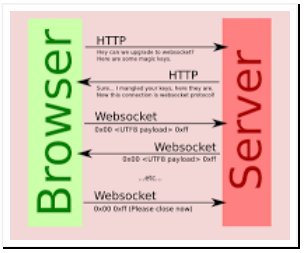
Apuntes de Java

Visita http://kenai.com/projects/apuntes, http://java.net/projects/apuntes
Google Groups: Apuntes de Java

miércoles, 22 de mayo de 2013

WebSockets en Java EE 7 (JSR 356)



Los WebSockets son una manera de poder comunicarse vía web entre un cliente y un servidor. A diferencia con otras tecnologías parecidas como los RESTful WebService, es que esta tecnología es bidireccional. El RESTful tiene que constantemente pedir al servidor para ver si hay un cambio, y con algunas técnicas "push" se puede simular una comunicación bidireccional. Con WebSockets, la comunicación es nativa.

Ya que estamos cerca del lanzamiento de Java EE 7 implementado en GlassFish 4.0, veremos un pequeño esbozo de esta tecnología.

WebSockets no es nuevo. Ya existe una norma en W3C que regula su implementación (aunque aún está en revisión, creo que su uso será como el HTML5.. todos lo usan asumiendo que está aprobado). Existen plugins en JQuery para consumirlos, Dojo Toolkit ya lo tiene implementado en su extensión Dojox; en los navegadores desde las versiones Chrome 4, Firefox 8, y Safari 5 ya está implementado (no responderé sobre MSIE)... ahora nos falta el lado del servidor, y esto es lo que veremos en este post. En la versión Java EE 7 vendrá como API para poder montar nuestro servicio WebSockets.

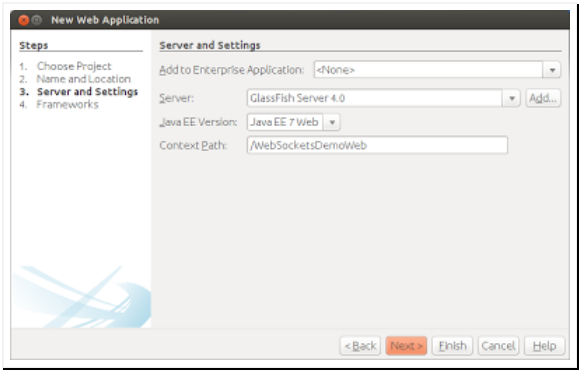
Preparando el Software

A la fecha de este post, he utilizado la versión desarrollo de NetBeans. (Cuando ya sea oficial, pueden descargar la versión completa). Debemos descargar la versión completa para la plataforma, es decir, la de Windows (si usamos Windows), la de Linux (si usamos Linux), etc.. pero no la versión .zip, ya que en esta no viene incluido el GlassFish.

Después de instalarlo, nos preparamos para crear nuestro primer proyecto Web.

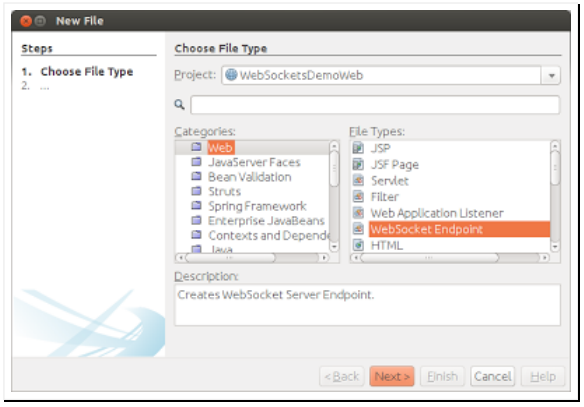
Proyecto Web

Creamos nuestro proyecto llamado **WebSocketsDemoWeb** y usamos el servidor "GlassFish Server 4.0" y elegimos la versión Java EE: Java EE 7 Web.



... y le damos clic en "Finish".

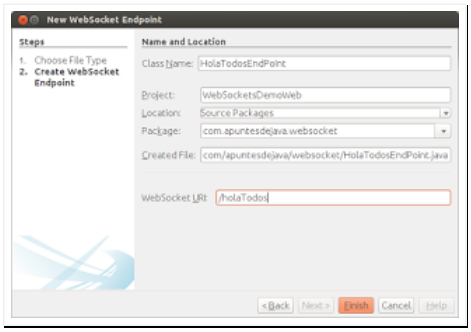
Ahora, crearemos nuestra clase que será el punto de acceso al WebSocket. Creamos nuevo archivo desde File > New (Ctrl+N), y de la categoría "Web" seleccionamos "WebSocket Endpoint"



... clic en "Next".

Ahora debemos declarar las característica que tendrá nuestro WebSocket, como el nombre de la clase, la ruta del websocket, etc.

nombre de clase: **HolaTodosEndPoint**
paquete: **com.apuntesdejava.websocket**
WebSocket URI_: **/holaTodos**



(El "Hola mundo" y "Hola todos" nunca fallan)

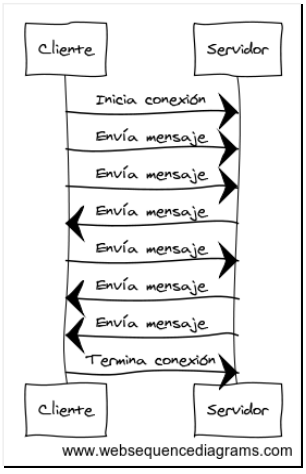
Y listo, la clase está creada. Como ahora vienen las clases de java se pueden declarar sin más archivos de configuración (algún archivo de despliegue o xml que configure los websockets) entonces se pudo haber creado una clase simple y a ella colocarle las anotaciones necesarias.

Ciclo de vida de una conexión WebSocket
Las conexiones AJAX con RESTful o SOAP (y toda aplicación web) tienen esta particularidad:

1. El cliente se conecta al servidor,
2. se establece la comunicación,
3. el cliente hace el requerimiento,
4. el servidor responde
5. y terminó la conexión.

Si se tiene que hacer otro requerimiento, se vuelve a realizar todos los pasos anteriores. De ahí el uso de sesiones en cookies, variables locales y demás artilugios para asegurar que la siguiente petición se refiere al mismo cliente.

Con el WebSocket esto se hace más simple.



Es decir:

1. El cliente inicia la conexión con el servidor
2. Ambos se comunican. El cliente al servidor o el servidor al cliente.
3. Se termina la conexión.

Ya se parece a una aplicación Desktop (por qué no lo inventaron desde un inicio!? El modelo MVC2 se puede ir al tacho!... bueno, es lo que yo opino)

Implementando el WebSocket

Ahora nos toca escribir el código que estará del lado del servidor.
Escribiremos este código co
n algunas anotaciones del WebSocket:

Data hosted with ♥ by Pastebin.com - Download Raw - See Original

```java
package com.apuntesdejava.websocket;

import java.util.logging.Logger;
import javax.websocket.OnClose;
import javax.websocket.OnMessage;
import javax.websocket.OnOpen;
import javax.websocket.server.ServerEndpoint;

@ServerEndpoint("/holaTodos")
public class HolaTodosEndPoint {

    static final Logger LOGGER = Logger.getLogger(HolaTodosEndPoint.class.getName());

    @OnOpen
    public void iniciaConexion() {
        LOGGER.info("Iniciando la conexion");

    }

    @OnClose
    public void finConexion() {
        LOGGER.info("Terminando la conexion");

    }

    @OnMessage
    public String onMessage(String mensaje) {
        LOGGER.log(Level.INFO, "Recibiendo mensaje:{0}", mensaje);
        return "Hola a todos con este mensaje:" + mensaje;
    }

}
```

Código:http://pastebin.com/9bbPPhUw

Notemos que - gracias a las anotaciones - ya no necesitamos implementar ni extender alguna clase. Basta con describir la anotación, Java sabrá que hacer con ese método.

Implementado el cliente

En el lado del cliente lo haremos usando HTML5, así que funcionará con FF, GC y Safari (sorry MSIE). El código es bastante simple, y está comentado para no perder el hilo.

Data hosted with ♥ by Pastebin.com - Download Raw - See Original

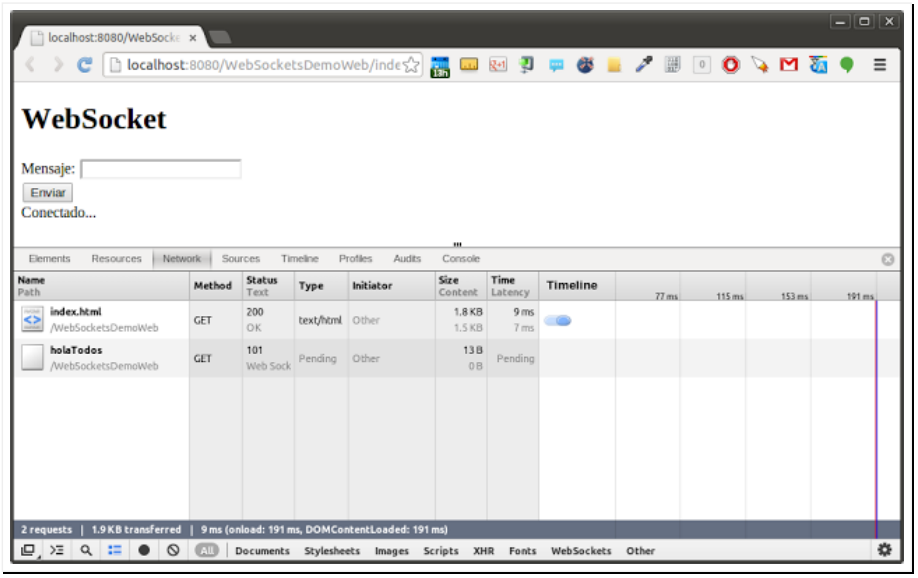
```html
<!DOCTYPE html>
<html>
    <head>
        <title></title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <script type="text/javascript">
            //notar el protocolo.. es 'ws' y no 'http'
            var wsUri = "ws://localhost:8080/WebSocketsDemoWeb/holaTodos";
            var websocket = new WebSocket(wsUri); //creamos el socket
            websocket.onopen = function(evt) { //manejamos los eventos...
                log("Conectado..."); //... y aparecerá en la pantalla
            };
            websocket.onmessage = function(evt) { // cuando se recibe un mensaje
                log("Mensaje recibido:" + evt.data);
```

```
15.             };
16.             websocket.onerror = function(evt) {
17.                 log("oho!.. error:" + evt.data);
18.             };
19.
20.
21.             function enviarMensaje() {
22.                 websocket.send(mensajeTXT.value);
23.                 log("Enviando:" + mensajeTXT.value);
24.             }
25.             function log(mensaje) { //aqui mostrará el LOG de lo que está haciendo el WebSocket
26.                 var logDiv = document.getElementById("log");
27.                 logDiv.innerHTML += (mensaje + '<br/>');
28.             }
29.
30.         </script>
31.     </head>
32.     <body>
33.         <h1>WebSocket</h1>
34.         <form >
35.             <label for="mensajeTXT">Mensaje:</label>
36.             <input id='mensajeTXT' name='mensajeTXT'/><br/>
37.             <button type="button" onclick="enviarMensaje()">Enviar</button>
38.         </form>
39.         <div id="log">
40.
41.         </div>
42.     </body>
43. </html>
```

Código: http://pastebin.com/CAgYxzqX

Probando la aplicación
Bastará con ejecutar la aplicación.

Si estamos con Chrome activemos la consola de desarrollo con la tecla F12. Veremos en la pestaña de NetWork solo ha tenido dos peticiones, la primera es la misma HTML y la segunda es la conexión al WebSocket... y tiene estado pendiente!



Hacemos clic en ese nodo para ver el detalle. Podremos ver la cabecera de conexión...

Diego Silva's favorite books »

RSS interesantes

Planet NetBeans
APIDesign - Blogs: Gradle belongs to Stone Age!
Hace 13 horas

Avbravo

Método onblur
Hace 1 día

The Aquarium
Java EE 7 Hands-on-Lab Updated for You to Use!
Hace 2 días

Oracle Technology Network's Blog
Best of OTN Week of March 9th
Hace 2 días

Delicious/OracleTechnologyNetwork/otntecharticle
Oracle Identity Manager 11G R2 PS2 Catalog Cart Items Customization
Hace 2 días

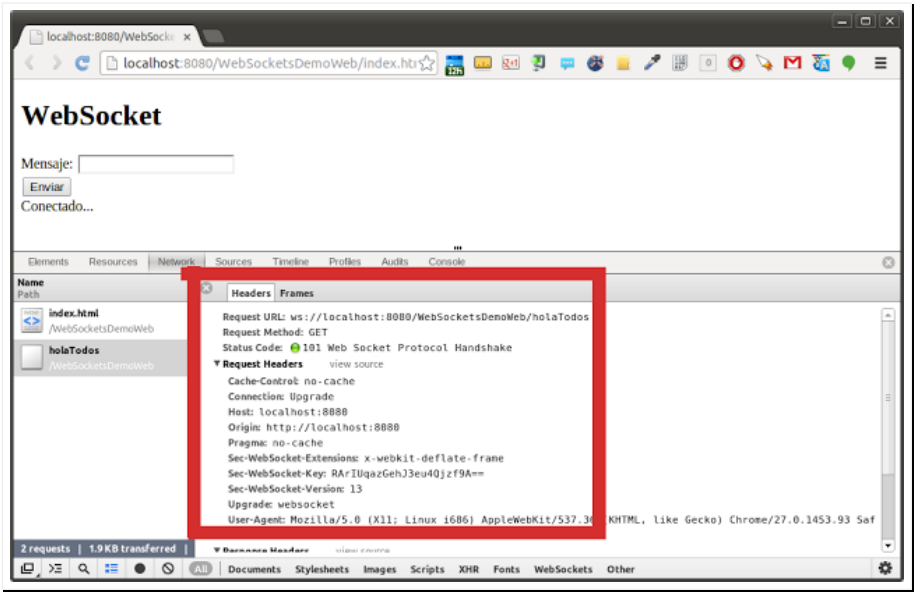
NetBeans Highlights
Free Oracle Book: "Developing Applications with NetBeans IDE 8.0"
Hace 5 días

Delicious/OracleTechnologyNetwork/java
Using File Based Loader for Fusion Product Hub
Hace 6 días

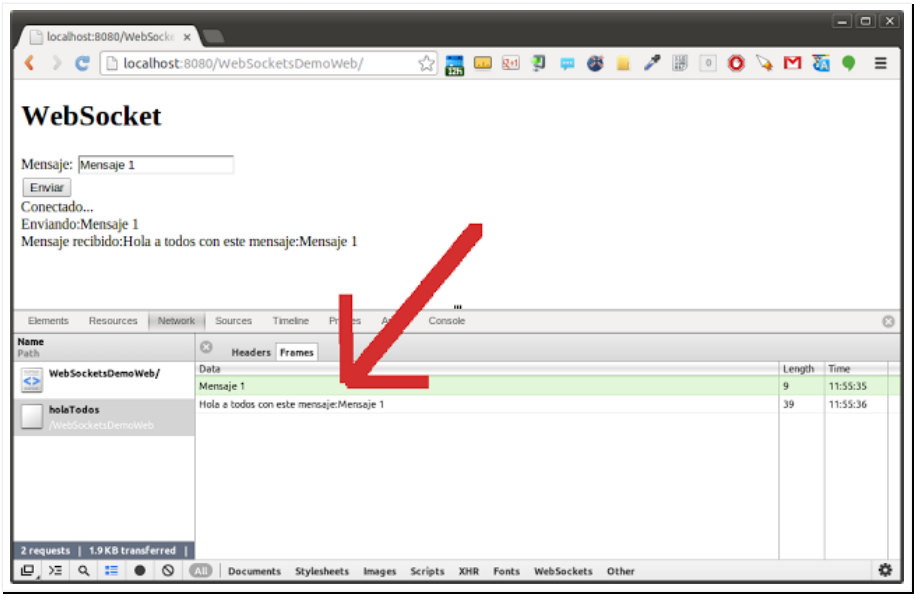
Java Evangelist John Yeary's Blog
JSF 2.x Tip of the Day: Implementing a ViewMapListener
Hace 3 semanas

Latest Updates from MySQL AB
Oracle Linux Now Hosted on the Docker Hub Registry
Hace 5 semanas

Java and everything
Oracle Linux Docker Base Image on Ubuntu 14.10
Hace 3 meses



... y los *frames* de comunicación. Cada vez que escribimos un mensaje en el input veremos cómo se envían los frames entre el servidor y el cliente.



Broadcast a todos los conectados

Este ejemplo es bastante interesante, ya que la idea es enviar un mensaje a todos los usuarios conectados. A diferencia del ejemplo anterior - donde el cliente espera una respuesta de la petición que ha hecho - este ejemplo solo espera que el servidor le diga algo. Notaremos que el cliente no tiene que hacer ni nada, ni tampoco estará consultando cada cierto tiempo al servidor si hay mensajes nuevos.

Lo que haremos es colocar en el servidor un EJB que cada cierto tiempo envíe una comunicación a los clientes. Para ello haremos que nuestro ServerEndPoint sea un ejb singleton, y colocamos un método programado para que envíe cada 10 segundos un mensaje. Ahora, para saber quienes están conectados, debemos crear una lista de todas las sesiones. Cada vez que se ejecuta el evento `@OnOpen` se recibirá como parámetro la sesión y se agregará a una lista; y cada vez que se llame al evento `@OnClose`, haremos que cierra la conexión y quitamos el evento de la lista.

Aquí el código del Servidor:


```
1. package com.apuntesdejava.websocket;
2.
3. import java.io.IOException;
4. import java.util.ArrayList;
5. import java.util.Date;
6. import java.util.List;
7. import java.util.logging.Level;
8. import java.util.logging.Logger;
```

```
9.  import javax.ejb.Schedule;
10. import javax.ejb.Singleton;
11. import javax.websocket.OnClose;
12. import javax.websocket.OnMessage;
13. import javax.websocket.OnOpen;
14. import javax.websocket.RemoteEndpoint;
15. import javax.websocket.Session;
16. import javax.websocket.server.ServerEndpoint;
17.
18. @ServerEndpoint("/broadcast")
19. @Singleton //es - a la vez- nuestro EJB para programar eventos
20. public class BroadcastUsuariosEndPoint {
21.
22.     static final Logger LOGGER = Logger.getLogger(BroadcastUsuariosEndPoint.class.getName());
23.     //La lista de conexiones realizadas
24.     static final List<Session> conexiones = new ArrayList<>();
25.
26.     /**
27.      * Evento que se ejecuta cuando un cliente se conecta
28.      *
29.      * @param session La sesion del cliente
30.      */
31.     @OnOpen
32.     public void iniciaSesion(Session session) {
33.         LOGGER.log(Level.INFO, "Iniciando la conexion de {0}", session.getId());
34.         conexiones.add(session); //Simplemente, lo agregamos a la lista
35.
36.     }
37.
38.     /**
39.      * Evento que se ejecuta cuando se pierde una conexion.
40.      *
41.      * @param session La sesion del cliente
42.      */
43.     @OnClose
44.     public void finConexion(Session session) {
45.         LOGGER.info("Terminando la conexion");
46.         if (conexiones.contains(session)) { // se averigua si está en la colección
47.             try {
48.                 LOGGER.log(Level.INFO, "Terminando la conexion de {0}", session.getId());
49.                 session.close(); //se cierra la conexión
50.                 conexiones.remove(session); // se retira de la lista
51.             } catch (IOException ex) {
52.                 LOGGER.log(Level.SEVERE, null, ex);
53.             }
54.         }
55.     }
56.
57.     /**
58.      * Enviaremos un mensaje a todos los conectados
59.      */
60.     @Schedule(second = "*/10", minute = "*", hour = "*", persistent = false)
61.     public void notificar() {
62.         LOGGER.log(Level.INFO, "Enviando notificacion a {0} conectados", conexiones.size());
63.         String mensaje = "Son las " + (new Date()) + " y hay " + conexiones.size() + " conectados
";  // el mensaje a enviar
64.         for (Session sesion : conexiones) { //recorro toda la lista de conectados
65.             RemoteEndpoint.Basic remote = sesion.getBasicRemote(); //tomo la conexion remota con
el cliente...
66.             try {
67.                 remote.sendText(mensaje); //... y envío el mensajue
68.             } catch (IOException ex) {
69.                 LOGGER.log(Level.WARNING, null, ex);
70.             }
71.         }
72.
```

```
    }

    /**
     * Solo es un metodo que atiende las peticiones
     *
     * @param mensaje
     * @param sesion
     */
    @OnMessage
    public void onMessage(String mensaje, Session sesion) {
        LOGGER.info("Se recibe un mensaje, aunque no se hace nada");
    }

}
```

Código: http://pastebin.com/6UFi1gHv

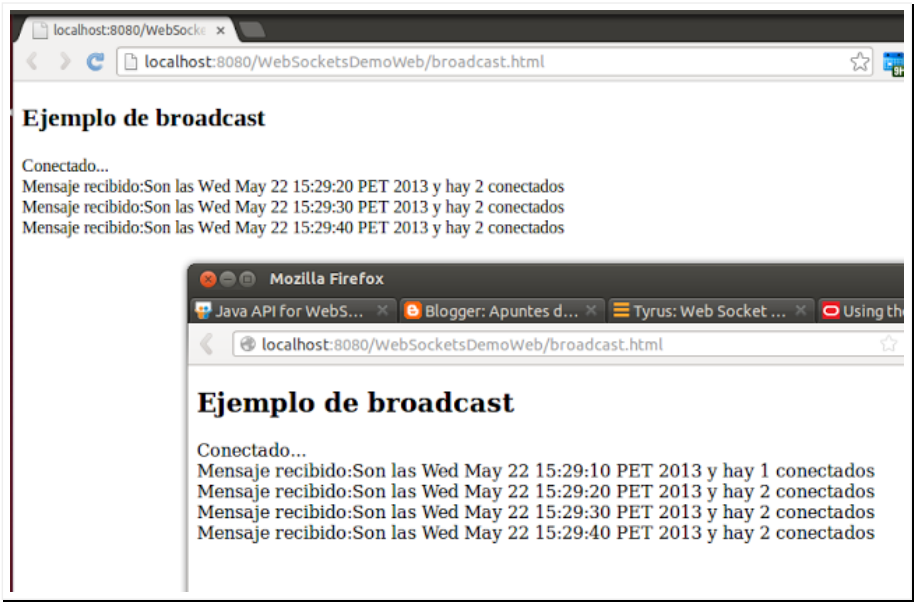
Y el lado del cliente es lo más simple que puede haber:

Data hosted with ♥ by Pastebin.com - Download Raw - See Original

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <script type="text/javascript">
            var wsUri = "ws://localhost:8080/WebSocketsDemoWeb/broadcast";
            var websocket = new WebSocket(wsUri); //creamos el socket
            websocket.onopen = function(evt) { //manejamos los eventos...
                log("Conectado..."); //... y aparecerá en la pantalla
            };
            websocket.onmessage = function(evt) { // cuando se recibe un mensaje
                log("Mensaje recibido:" + evt.data);
            };
            websocket.onerror = function(evt) {
                log("oho!.. error:" + evt.data);
            };


            function log(mensaje) { //aqui mostrará el LOG de lo que está haciendo el WebSocket
                var logDiv = document.getElementById("log");
                logDiv.innerHTML += (mensaje + '<br/>');
            }

        </script>
    </head>
    <body>
        <h2>Ejemplo de broadcast</h2>
        <div id="log">

        </div>
    </body>
</html>
```

Código: http://pastebin.com/ZuSZBkL9

Ahora lo ejecutamos, abrimos desde un navegador, luego abrimos el mismo enlace desde otro navegador y veremos el efecto.



No se qué opinan ustedes, pero para mi es la mejor solución Cliente/Servidor que haya visto para Web.

Conclusión

En este post solo se vió algo muy simple de implementar, pero podemos ir más allá. Por ejemplo, podemos notificar alertas a los clientes - sea web o mobil (que podríamos verlo en otro post) - enviar objetos (que también será tema de otro post), etc.

Mi opinión es que esta tecnología debería enseñarse a la nueva generación de desarrolladores web, y que AJAX sea tomado como un tema de historia.

Referencias

Como yo no me invento estas cosas, naturalmente las he tomado de alguien. Arun Gupta (@arungupta) ha prestando varias presentaciones sobre Java EE 7 que es muy conveniente revisarlas.
Además, me he basado de otros links del mismo autor y otros ejemplos.

- WebSocket and Java EE 7 - Getting Ready for JSR 356 (TOTD #181) (Referencia antigua, pero válida para comprender el concepto)
- Building WebSocket Apps in Java using JSR 356
- Ejemplos de Tyrus, biblioteca que implementa el WebSocket API en GlassFish 4.0

Código fuente

Y por si las dudas, aquí el código fuente del proyecto. Recuerden utilizar GlassFish 4.0 y que el IDE sea compatible con Java EE .

https://java.net/projects/apuntes/downloads/download/web/WebSocketsDemoWeb.tar.gz

Luego subiré el código en el Mercurial de java.net.

Buen día y bendiciones a todos!

Publicado por Diego Silva a las 16:29:00 +13 Recomendar esto en Google

Etiquetas: glassfish, glassfish v4, html5, javaee, javaee7, netbeans, netbeans 7.4, tutorial, web, webservices, websockets

¿Qué te parece?: divertido (0) interesante (0) excelente (0) aburrido (0)

12 comentarios

Añadir un comentario como Javier Vbno

Mejores comentarios

Apuntes de Java a través de Google+ hace 1 año · Compartido públicamente

Apuntes de Java a través de Google+ hace 1 año - Compartido públicamente.
Un nuevo post, ahora de **#WebSockets** sobre **#JavaEE7** con **+NetBeans** y **+GlassFish**

+1 1 · Responder

Edwin Ilovares hace 1 año
¿Cuantos post tienes planeado hacer?

Diego Silva hace 1 año
Casi tantos como los de RESTful :) con un objeto, con un arreglo, un CRUD, seguridad, usando móviles. La cuestión es el tiempo :S Aunque por ahora quiero centrarme más en publicar todo lo nuevo de **#JavaEE7** .. y espero que el tiempo me dé.

Diego Silva a través de Google+ hace 1 año - Compartido públicamente.
Amigo José.. podemos hacer un hangout (si es que aun no han hablado del tema) sobre esto.

> **Apuntes de Java** es el autor original de esta publicación.
> Un nuevo post, ahora de **#WebSockets** sobre **#JavaEE7** con **+NetBeans** y **+GlassFish**

1 · Responder

Ver las 3 respuestas

Jose Diaz hace 1 año
Hola, si hay que programar de nuevo los hangouts. Que dia les acomoda mas? horario?

Edwin Ilovares hace 1 año
Domingos :3, mañana, tarde, noche.

Julio Cuevas hace 1 año - Compartido públicamente.
Muy buen aporte... Gracias

+1 1 · Responder

Daniel Giancarlo hace 1 año - Compartido públicamente.
Francamente mis felicitaciones por tan interesante post de lo ultimo de java EE 7. Estare esperando por mas de este tema. Gracias.

1 · Responder

Christian G. Adam hace 1 año - Compartido públicamente.
Muy bueno tu post. Sabes algo de la implementacion de JSR 356 pero para desarrollar aplicaciones clientes con JSE? Estuve viendo tyrus pero tiene dependencias con clases de J2EE. Necesito alguna implementacion simple que pueda embeber en una app JSE. Gracias!

1 · Responder

Sergio Andrés Ñustes hace 1 año
Hola amigo he probado este (http://java-websocket.org/) y me ha funcionado perfectamente como servidor y cliente en Java, y con javascript consumiendo sin problemas, no lo he probado con Android pero supongo que debe ir bien como en la web explican pues igual básicamente es un Jar que se agrega como librería y que utiliza el ServerSocket como base.

Christian Loza hace 1 año - Compartido públicamente.
si quiero enviar un dato para que sea calculado como lo haría?

1

Entrada más reciente Página principal Entrada antigua

Suscribirse a: Enviar comentarios (Atom)

Nube de etiquetas

actualizacion (1) adf (1) ajax (11) ant (1) aop (2) apache (9) articulos (2) base de datos (4) bug (1) bugzilla (1) cdi (3) certificación (1) chiste (4) colaboración (1) comentarios (11) commons (1) concurso (3) conferencias (5) configuración (2) curso (5) dao (2) datasource (4) desktop (2) documentacion (7) eclipse (3) ejb (10) ejb 3.1 (6) encuesta (3) error (1) errores (1) eventos (1) facelets (3) firefox (2) formateo (3) glassfish (30) glassfish v3 (9) glassfish v4 (8) groovy (2) html5 (3) ibatis (3) ICEfaces (6) internet (2) ireport (4) jasperreports (4) java (74) java 7 (3) java ee (39) java ee 6 (18) java ee 7 (2) java ee 8 (1) java se6 (1) java8 (2) javaee (7) javaee7 (13) javaee8 (1) javafx (8) javascript (7) jdbc (6) jdk (3) jdk 7 (1) JEE7 (3) jersey (1) jfreechart (1) jmx (1) jpa (10) jpa 2.0 (2) jquery (1) jsf (31) jsf 2.0 (6) jsf 2.2 (12) json (3) jsp (4) jstl (1) juegos (2) junit (1) jvisualvm (1) libraries (2) libros (11) liferay (19) linux (1) log4j (2) logging (2) mapeo (1) maven (1) mvc (3) myfaces (1) mysql (7) netbeans (143) netbeans 6.1 (14) netbeans 6.5 (24) netbeans 6.7 (9) netbeans 6.8 (14) netbeans 6.9 (20) netbeans 7.0 (6) netbeans 7.3 (4) netbeans 7.4 (3) NetBeans Platform (2) netbeans8 (2) nojava (1) noticias (35) noticias NetBeans (1) off topic (38) opinion (25) opnj (1) oracle (8) packtpub (1) php (9) plugins (7) portalpack (4) portlets (11) presentacion (2) programador (2) RCP (1) regex (1) rendimiento (1) reportes (3) restful (12) scjp (2) seguridad (8) senna (1) server (2) servlets (3) spring (10) sql (3) struts (5) sun (9) swing (8) tiles (2) tips (40) tomcat (15) trucos (24) tutorial (62) tutorial java (2) ubuntu (2) video (12) web (68) webinar (1) weblogic (2) webservices (9) websockets (2) wiki (1) wildfly (1) win8 (1) windows (7) xml (4) youtube (1)