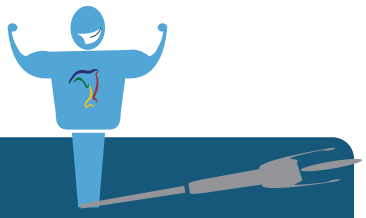
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

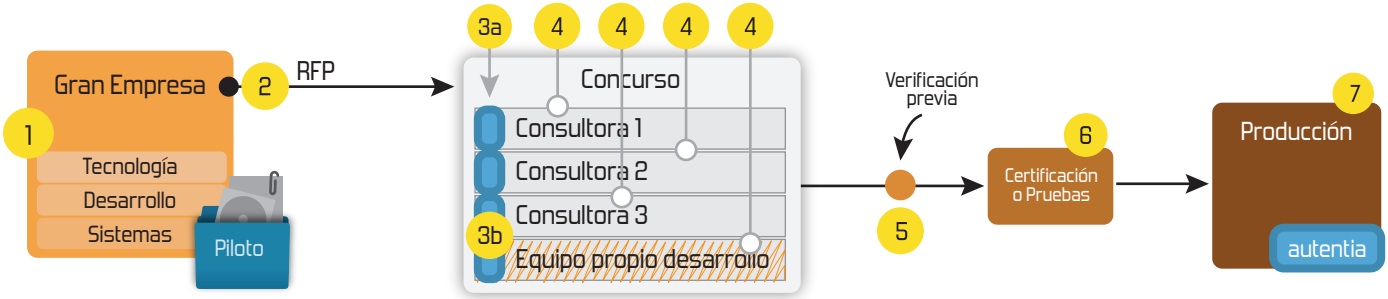
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

- Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery
- Gestor portales (Liferay)
- Gestor de contenidos (Alfresco)
- Aplicaciones híbridas
- Control de autenticación y acceso (Spring Security)
- UDDI
- Web Services
- Rest Services
- Social SSO
- SSO (Cas)
- JPA-Hibernate, MyBatis
- Motor de búsqueda empresarial (Solr)
- ETL (Talend)
- Dirección de Proyectos Informáticos.
- Metodologías ágiles
- Patrones de diseño
- TDD
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conociemiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Miguel Arlandy Rodríguez

Consultor tecnológico de desarrollo de proyectos informáticos.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/JEE

Ver todos los tutoriales del autor

Fecha de publicación del tutorial: 2009-02-26 Tutorial visitado 3 veces Descargar en PDF

WebSockets con Stomp y ActiveMQ: ¿chateamos?.

Índice de contenidos

1. Introducción

Como vimos en el tutorial WebSockets con Java y Tomcat 7, los WebSockets son una técnica excelente para mantener una comunicación bidireccional entre cliente y servidor. Esta comunicación bidireccional es muy útil para crear "aplicaciones en tiempo real" (chat's, sistemas de monitorización, videojuegos multijugador...).

Existen diferentes implementaciones de la especificación WebSocket en el lado del servidor además de la que ofrece Tomcat 7. Una de ellas es la de Apache ActiveMQ, del que también hablamos en anteriores tutoriales. A diferencia de Tomcat 7, ActiveMQ no es un contenedor de Servlets sino un intermediario de mensajes (con sus topics y colas de mensajería).

En este tutorial veremos el soporte a WebSockets que nos ofrece ActiveMQ y enviaremos y consumiremos mensajes directamente desde nuestro navegador (Javascript). Para ello crearemos un chat.

2. Entorno.

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil MacBook Pro 15' (2.2 Ghz Intel Core I7, 8GB DDR3).
- Sistema Operativo: Mac OS Mountain Lion 10.8
- Entorno de desarrollo: Intellij Idea 11.1 Ultimate.
- Apache ActiveMQ 5.8
- jQuery 1.9.1
- jQuery-UI 1.8.22
- Google Chrome 25
- Mozilla Firefox 19

3. ¿Qué vamos a hacer?

El propósito de este tutorial es dar a conocer el soporte a la especificación WebSocket que ofrece Apache ActiveMQ, un excelente gestor de mensajes open source del que ya hemos hablado en anteriores tutoriales.

Últimas Noticias

» Atención, APLAZADO Estreno último capítulo de Terrakas

» Vendedor: Soy inseguro, filtra o elige por mi: si quieres que te compre.

» Comentando el libro: El arte de pensar, de Rolf Dobelli

» Ya está a la venta mi segundo libro: Planifica tu éxito, de aprendiz a empresario

» Ya esta disponible en eBook mi primer libro: Informática Profesional

Histórico de noticias

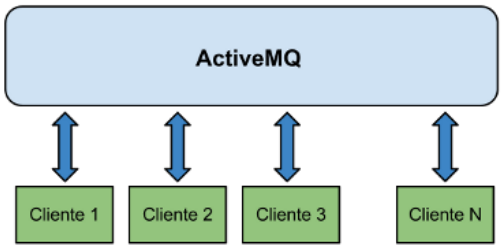
Últimos Tutoriales

» Análisis de los sentimientos en twitter con el soporte de Apicultur.

» Creación de presentaciones con PowToon

» jBPM Form Builder: generación de formularios para jBPM5 y su integración en Guvnor.

Para conseguir nuestro objetivo vamos a ver cómo implementar un chat donde todos los mensajes se enviarán a colas de mensajería de ActiveMQ. Vamos a hacer exactamente el mismo ejemplo que vimos en el tutorial de WebSockets con Tomcat 7. La diferencia es que ahora reemplazaremos Tomcat por ActiveMQ, por lo que no se desarrollará nada en Java (100% Javascript). Mismo problema, distintas soluciones :)



4. El protocolo STOMP

STOMP es un sencillo protocolo diseñado para la comunicación asíncrona entre clientes a través de un mediador de mensajes (en nuestro caso, ActiveMQ). El protocolo está basado en frames. Los frames no son más que un **comando** (u operación), un **mensaje** (o body) y unas **cabeceras** del mensaje (headers).

La especificación STOMP se compone de una serie de operaciones (comandos) para la interacción entre cliente e intermediario de mensajes. A nosotros únicamente nos harán falta cuatro de estas operaciones.

- **connect**: Establece conexión con el broker de mensajería.
- **subscribe**: El cliente se suscribe a un destino del broker (una cola o un topic).
- **send**: El cliente envía un mensaje a un destino del broker (cola o topic).
- **disconnect**: Cierra la conexión con el broker de mensajería.

Este es el listado completo de comandos de la especificación 1.2

Como en nuestro ejemplo el cliente será el navegador (Javascript), necesitaremos una implementación de la especificación STOMP en Javascript. Podemos encontrarla dentro de la distribución de ActiveMQ, en concreto en **${ACTIVEMQ_HOME}/webapps-demo/demo/websocket/stomp.js**.

5. Creando el chat

A continuación veremos los pasos necesarios para implementar nuestro chat.

5.1 Paso 1: habilitar el soporte a WebSockets de ActiveMQ.

Lo primero que debemos hacer es habilitar el soporte a WebSockets que nos ofrece ActiveMQ. En el tutorial Introducción a ActiveMQ vimos algunas de las múltiples opciones de configuración que proporciona esta herramienta, aunque no cómo habilitar WebSockets. Para ello nos vamos al fichero de configuración **${ACTIVEMQ_HOME}/conf/activemq.xml"** y en el apartado "transportConnectors" añadimos lo siguiente:

```
<transportconnectors>
    <transportConnector name="openwire" uri="tcp://0.0.0.0:61616"/>
    <transportConnector name="ws" uri="ws://0.0.0.0:61614"/>
</transportconnectors>
```

5.2 Paso 2: establecer la conexión con ActiveMQ.

Ahora veremos cómo conectar con ActiveMQ vía WebSockets y la ayuda de STOMP. Como dijimos en el apartado anterior, necesitamos una implementación de STOMP para poder interactuar con el broker. La librería está en ${ACTIVEMQ_HOME}/webapps-demo/demo/websocket/stomp.js.

```
<script type="text/javascript" src="js/stomp.js"></script>
```

Una vez que ya tenemos nuestra dependencia, conectar es muy sencillo. Obtenemos un cliente y nos conectamos al broker tal y como se aprecia en el siguiente código:

```
var user = null;
var password = null;
var client = Stomp.client("ws://localhost:61614"); // configurar!!!
client.connect(user, password, onconnect, onerror);

function onconnect() {
    // hacemos lo que sea cuando se establezca la conexión
}

function onerror() {
    // manejamos el error
}
```

5.3 Paso 3: suscribiéndonos a la cola de mensajes entrantes.

Lo siguiente que haremos, una vez estemos conectados, será suscribirnos a la cola de mensajería donde se enviarán los mensajes que recibirá un usuario concreto. Cada usuario tiene su cola de mensajes. Todo aquel que quiera enviar mensajes a un usuario deberá hacerlo en su cola correspondiente.

» Promesas: Organiza tu código Javascript/Coffeescript

» jBPM5 Console Server and Human Task Server: instalación y configuración

Últimos Tutoriales del Autor

» AngularJS: primeros pasos.

» Sonar y Javascript: obteniendo la cobertura de nuestro código

» Sonar y Total Quality: midiendo la calidad total de nuestros proyectos

» Servicios REST documentados y probados con Swagger

» Creación de plantillas DSL con Drools

Últimas ofertas de empleo

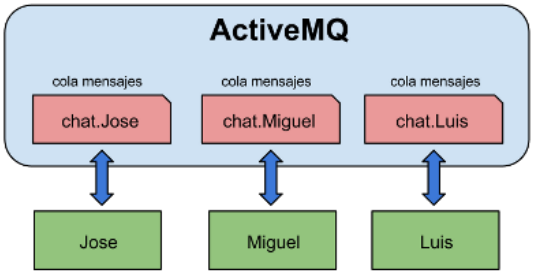
2011-09-08
Comercial - Ventas - MADRID.

2011-09-03
Comercial - Ventas - VALENCIA.

2011-08-19
Comercial - Compras - ALICANTE.

2011-07-12
Otras Sin catalogar - MADRID.

2011-07-06
Otras Sin catalogar - LUGO.

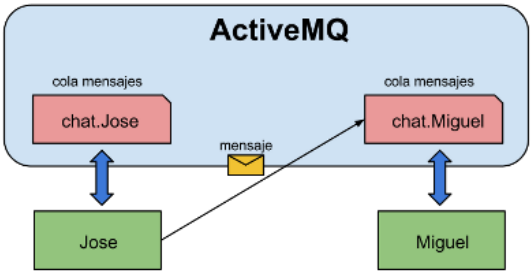


Para suscribirnos a nuestra cola de mensajes entrantes y procesarlos haremos lo siguiente:

```
var userName = "nombreDeUsuario";
function subscribeMessageQueue() {
    client.subscribe('chat.' + userName, processMessage);
}

function processMessage(message) {
    var sender = message.headers.from; // quien ha enviado el mensaje
    var messageBody = message.body;
    // hacemos lo que sea con el emisor del mensaje y el mensaje
}
```

5.4 Paso 4: enviando mensajes a un destinatario.

Muy bien, ya podemos recibir mensajes de otros usuarios. Vamos a ver cómo podemos enviarlos. El proceso es sencillo, basta con mandar un mensaje a la cola del usuario receptor.



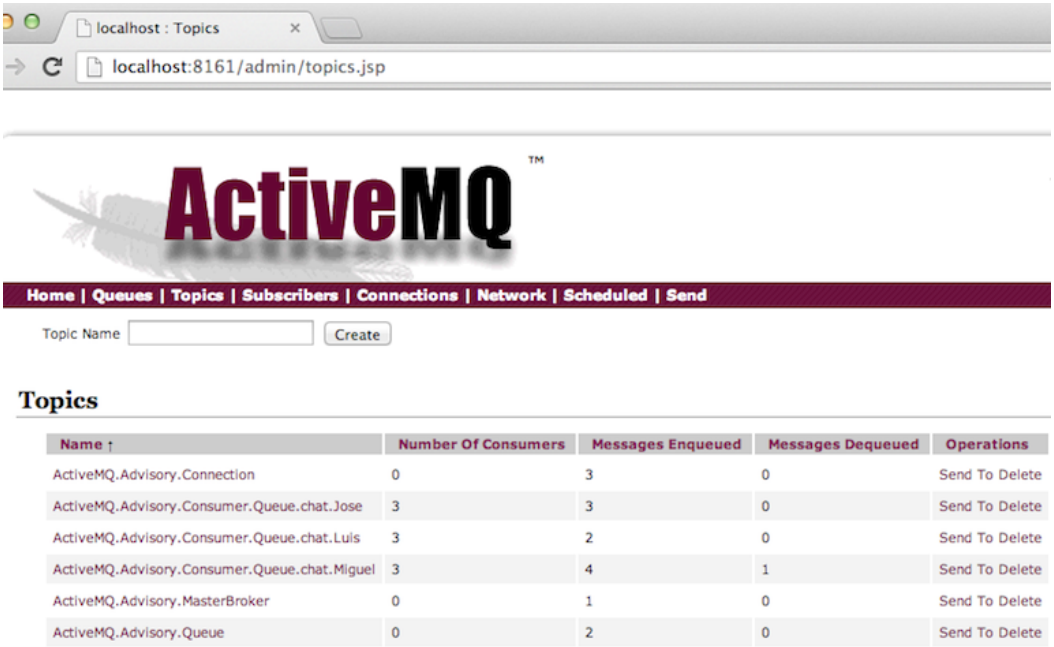
El código es sencillo. Obsérvese que añadimos una cabecera "from" al mensaje para que se sepa quien lo envió.

```
function sendMessage(sender, receiver, message) {
    client.send('chat.' + receiver, {from: sender}, message);
}
```

5.5 Paso 5: actualizando la lista de usuarios conectados.

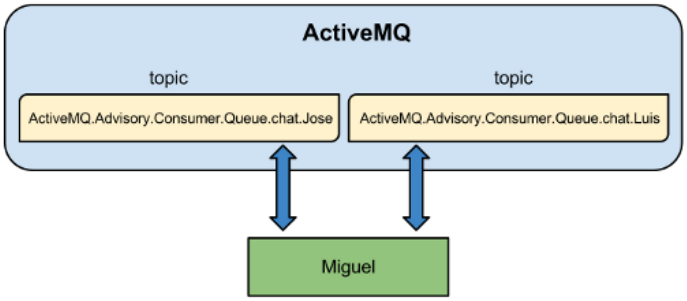
Pues lo último que nos queda es actualizar nuestra lista de usuarios conectados al chat. Probablemente este es el punto más complejo.

ActiveMQ cuenta con unos topics especiales donde se envían los mensajes de los consumidores de colas o topics de mensajería. A estos topics llega la información relativa a las conexiones o desconexiones a los destinos del broker. Dichos topics son de la forma **"ActiveMQ.Advisory.Consumer.>"**. Puede verse mejor en la consola de administración.



Lo que haremos para conocer los usuarios conectados al chat será suscribirnos a estos topics para que tengamos la

información de qué usuarios se conectan o desconectan.



Cuando nos suscribamos a este tipo de topics, recibiremos dos tipos de mensajes con la información relativa a las conexiones y desconexiones que se producen.

Cuando un **usuario se conecta** a una cola de mensajes, en el topic "Advisory" correspondiente a esa cola recibiremos un mensaje como el siguiente:

```
{"ConsumerInfo": {
  "commandId": 3,
  "responseRequired": false,
  "consumerId": {
    "connectionId": "identificador de conexión de Luis",
    "sessionId": -1,
    "value": 1
  },
  "destination": [
    "chat.Luis",
    {}
  ],
  "prefetchSize": 1000,
  "maximumPendingMessageLimit": 0,
  "browser": false,
  "dispatchAsync": true,
  "noLocal": false,
  "exclusive": false,
  "retroactive": false,
  "priority": 0,
  "optimizedAcknowledge": false,
  "noRangeAcks": false
}}
```

Nos interesan dos valores de este mensaje (JSON) que son: **connectionId** y el primer valor del **array destination**. El primer valor nos dará el identificador de la conexión del usuario que está suscrito a los mensajes de la cola cuyo nombre es el primer valor del array. Con esto tendremos el nombre del usuario conectado. Ahora actualizaríamos la lista de usuarios conectados con el nombre de dicho usuario.

Cuando un **cliente se desconecta**, en el topic "Advisory" correspondiente a la cola a la que estaba suscrito recibiremos un mensaje como el siguiente:

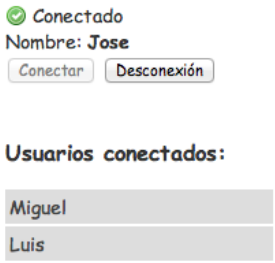
```
{"RemoveInfo": {
  "commandId": 0,
  "responseRequired": false,
  "objectId": {
    "@class": "ConsumerId",
    "connectionId": "identificador de conexión de Luis",
    "sessionId": -1,
    "value": 1
  },
  "lastDeliveredSequenceId": 0
}}
```

El valor que nos interesa es **connectionId** que es el mismo que teníamos asociado al usuario cuando se conectó, por lo que bastaría actualizar la lista de usuarios conectados eliminando el usuario que se corresponde con dicho id.

El código necesario para actualizar la lista de usuarios del chat es el siguiente. Obsérvese que para suscribirnos a un topic debemos anteponer **/topic/** al nombre de nuestro destino:

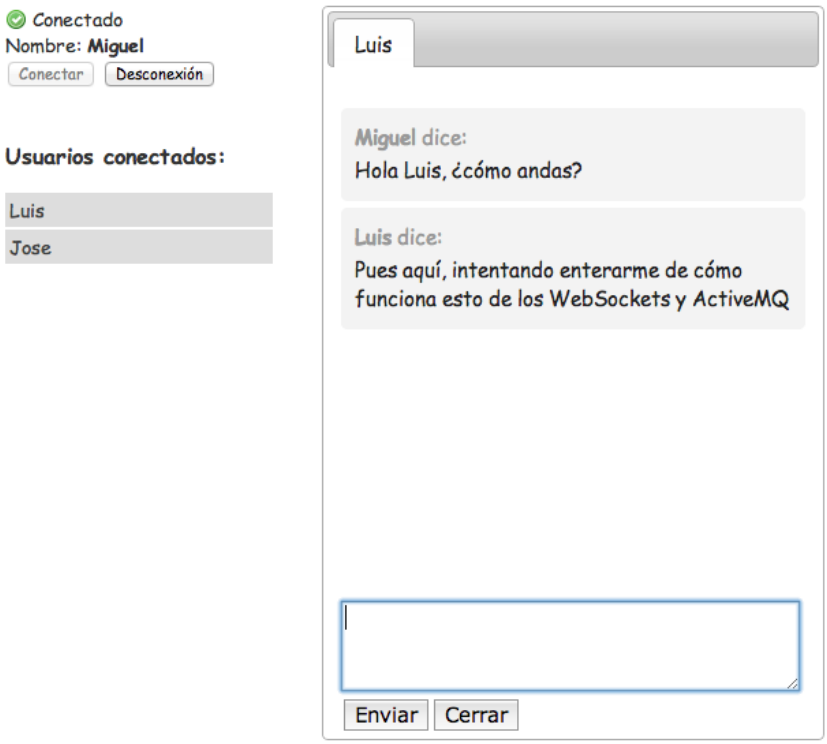
```
function subscribeActiveMQAdvisory() {
    client.subscribe('/topic/ActiveMQ.Advisory.Consumer.Queue.chat.>', function(msg){
        var jsonObject = JSON.parse(msg.body);
        proccessConsumers(jsonObject);
     });
}

function proccessConsumers(msg) {
    if (msg.ConsumerInfo) {
        processConsumerInfo(msg.ConsumerInfo); // si es un nuevo usuario
    } else if (msg.RemoveInfo) {
        processRemoveInfo(msg.RemoveInfo); // si alguien se ha desconectado
    } else {
        console.log('Mensaje desconocido ' + msg);
    }
}
```

El resultado sería algo así:



6. Listos para chatear

Pues con lo que hemos visto y un poquito de Javascript y CSS ya tendríamos nuestro chat implementado con WebSockets, STOMP y ActiveMQ :-D



El **código fuente al completo de este ejemplo** está disponible aquí https://github.com/marlandy/websockets-activemq.

7. Referencias

- API WebSocket
- Protocolo STOMP
- ActiveMQ y Websockets
- Código fuente del ejemplo
- WebSockets con Tomcat 7

8. Conclusiones

En este tutorial hemos visto otra forma de implementar un chat con WebSockets sin necesidad de usar Tomcat 7.

Personalmente, a la hora de implementar chat con WebSockets, me quedo con Tomcat 7 por su facilidad a la hora de gestionar el listado de usuarios que se conectan y desconectan. No obstante, para otro tipo de aplicaciones "en tiempo real" como puede ser un sistema de monitorización, podría ser mejor solución utilizar un broker de mensajería como ActiveMQ. Cuestión de gustos :)

Espero que este tutorial os haya sido de ayuda. Un saludo.

Miguel Arlandy

marlandy@autentia.com

Twitter: @m_arlandy

Por favor, vota +1 o compártelo si te pareció interesante

Share | 0

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

» Registrate y accede a esta y otras ventajas **«**

IMPULSA Impulsores Comunidad ¿Ayuda?

---
sin clicks

0 personas han traído clicks a esta página

+ + + + + + + +

powered by karmacracy