http://www.egovframe.go.kr/wiki/doku.php?id=egovframework:rte3.5:ptl:websocket

WebSocket

Outline

Being one specific type of Spring Technology, WebSocket is allowed Socket Communication under the HTTP environment. Spring basically uses WebSocket sub-protocol called STOMP.

(RFC6455 defines WebSocket protocol as its newest web application function. With full-duplex supported between the server and client, WebSocket protocol effectively replaces the conventional means such as java applet, XMLHttpRequest, Flash and ActiveX intended to make the web-based envoronment more interactive.)

While HTTP is used for the initial handshake (Protocol upgrade or switch. Returns 101 when server consents), the TCP socket is open by the upgrade request for HTTP when handshake succeeds.

- Comprising a brand-new spring-websocket module, Spring Framework 4.0 supports much more complex WebSocket, in which case the Java WebSocket gets compatible with Java WebSocket API (JSR356) to further the scope of functions it offers.

Spring Framework 4.0 now supports WebSocket

The minimum requirements for WebSocket are as follows:

- Java EE 7, JDK 7
- Servlet 3.1 (Container : Apache Tomcat 7.0.47+, Eclipse Jetty 9.0+, GlassFish 4.0+)
- Spring MVC 4.0 or newer (included in eGov 3.5)
- Web browser compatible with websocket websocket browser test page

1. WebSocket Fallback Options

Implemented by Servlet 3.1 functions, WebSocket is not available in some browsers (IE 9 or lower is not compatible with WebSocket). Keep in mind you're browser must be compatible with WebSocket and some proxies may force your link down (As a preventive means, SockJS protocol of Spring framework is worth consideration as the fallback option is available there - All you need to do is to activate Spring configurations)

2. Messaging Architecture

Contrary to the conventional REST using a plenty of URLs (nouns), some HTTP methods (verbs) and status-irrelevant architectures in development of web application,

WebSocket application is developed based on a single URL for the initial HTTP handshake, followed by messaging via TCP, signifying that the asynchronous, event-driven and messaging architects resembling the conventional messaging application (JMS, AMQP) are used.

Spring Framework 4.0 functions based upon the spring-messaging module that offers abstract notions such as Message, MessageChannel and MessageHandler, comprising the messaging architecture hand-in-hand. The module comprises several annotations to resemble programing models based upon Spring MVC annotations.

3. Sub-Protocol Support in WebSocket

While implying messaging architectures, WebSocket does not force a user to apply a specific messaging protocol. Being a "very thin layer" spread on TCP, WebSocket converts a series of byte streams, leaving interpretation of messages to the Application.

HTTP (being an application level protocol) is fairly different from WebSocket in that the WebSocket protocol does not contain the specific routing information, which is why WebSocket RFC has defined how to use sub-protocols in it.

While handshake is in use, the server and client are accessible to Sec-WebSocket-Protocol headers, through which they come to terms on a single sub-protocol for communication. This constitutes the duly agreed messaging format between the server and client as WebSocket does not force stringent definition of a sub-protocol.

Spring in compatible with STOMP, a widely applied and specific type of protocol similar to HTTP for use in a script.

4. When using WebSocket is advised?

WebSocket is the best-fit for a solution where the server and client exchange events very frequently. For the web applications such as financial services, game and collaboration services are vulnerable to time delay and involve frequent messaging, WebSocket is worth consideration.

On the contrary, the web applications intended for social communication and news feed need no more than polling as latency is not a major concern. For those web applications vulnerable to latency but involving communication of short messages (e.g. network failure check), they may take to failure check instead of WebSocket.

While WebSocket is the best-fit for the web applications requiring low latency and involving high frequency, some specific type of server and client might prefer combination of WebSocket and REST API, in which case the specific messages are to be sent out to the entirety of WebSocket clients by way of call for REST.

Spring Framework uses Classes @Controller and @RestController accompanied by Handling Methods HTTP and WebSocket. The Handling Method Spring MVC Request easily disseminates the messages to either the entirety of or specific clients.

WebSocket API

Spring Framework has been designed to fit various WebSocket Engines such as runtime-based Tomcat (7.0.47+), GlassFish (4.0+), WildFly (8.0+) JSR-356 or the native WebSocket such as Jetty (9.1+).

To use WebSocket API for development of application, you'll need to design all those details down the hierarchy. It is thus advised to have the application be compatible with sub-protocols for message formatting or message routing via annotation, which is what Spring provides the user with (STOMP over WebSocket)..

Take a brief look on what Spring WebSocket offers

1. How to configure WebSocketHandler

Spring implements WebSocketHandler to build a WebSocket server. WebSocketHandler comprises TextWebSocketHandler and BinaryWebSocketHandler.

```
import org.springframework.web.socket.WebSocketHandler;
import org.springframework.web.socket.WebSocketSession;
import org.springframework.web.socket.TextMessage;

public class MyHandler extends TextWebSocketHandler {

	@Override
	public void handleTextMessage(WebSocketSession session, TextMessage message) {
		// ...
	}

}
```

How to configure Java-Config for WebSocketHandler in the specific URL

```
import org.springframework.web.socket.config.annotation.EnableWebSocket;
import org.springframework.web.socket.config.annotation.WebSocketConfigurer;
import org.springframework.web.socket.config.annotation.WebSocketHandlerRegistry;
```

```
@Configuration
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {

	@Override
	public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
		registry.addHandler(myHandler(), "/myHandler");
	}

	@Bean
	public WebSocketHandler myHandler() {
		return new MyHandler();
	}

}
```

How to configure XML Config accordingly

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:websocket="http://www.springframework.org/schema/websocket"
	xsi:schemaLocation="
		http://www.springframework.org/schema/beans
		http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/websocket
		http://www.springframework.org/schema/websocket/spring-websocket-4.0.xsd">

	<websocket:handlers>
		<websocket:mapping path="/myHandler" handler="myHandler"/>
	</websocket:handlers>

	<bean id="myHandler" class="org.springframework.samples.MyHandler"/>

</beans>
```

Note the foregoing codings assumes that DispatcherServlet of Spring is used and Spring supports WebSocketHttpRequestHandler so that WebSocketHandler can be used in other web-development environments.

2. Customizing WebSocket Handshaking

Use HandshakeInterceptor to easily customize the initial WebSockethandshake, by coding for processings before and after handshake and making the specific attributes available just like that. Refer to the following example on how to have the interceptor provided by Spring sends http session attributes to WebSocketSession:

```
@Configuration
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {

	@Override
	public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
		registry.addHandler(new MyHandler(), "/myHandler")
			.addInterceptors(new HttpSessionHandshakeInterceptor());
	}

}
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:websocket="http://www.springframework.org/schema/websocket"
	xsi:schemaLocation="
		http://www.springframework.org/schema/beans
```

```
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/websocket
        http://www.springframework.org/schema/websocket/spring-websocket-4.0.xsd">

    <websocket:handlers>
        <websocket:mapping path="/myHandler" handler="myHandler"/>
        <websocket:handshake-interceptors>
            <bean class="org.springframework.web.socket.server.support.HttpSessionHandshakeInterceptor"/>
        </websocket:handshake-interceptors>
    </websocket:handlers>

    <bean id="myHandler" class="org.springframework.samples.MyHandler"/>

</beans>
```

In order to apply the advanced options, you'll need to expand DefaultHandshakeHandler comprising, without limitation, validating, client origin and sub-protocol. You may also have to configure the customized RequestUpgradeStrategy to make it compatible with the WebSocket Server Engine or versions non-compatible.

3. WebSocketHandler Decoration

Spring provides you with the basic class of WebSocketHandlerDecorator, most commonly used to decorate the activities supplementary to WebSocketHandler. While implementation of Logging, Exception Handling and WebSocketHandlerDecorator is made feasible by way of Java-config or XML config, you can also use ExceptionWebSocketHandlerDecorator to catch the exceptions thrown off WebSocketHandler, eventuating the Status Code 1011 representing server error.

4. Broadcasting Considerations

Integration of Spring WebSocket API and Spring MVC application is much easier than expected as the Dispatcher Servlet serves not only HTTP WebSocket Handshake but also HTTP request. By using WebSocketHttpRequestHandler, you can easily have other HTTP processing scenarios (Web Frameworks not falling under Spring MVC) combined, save that you'll need to consider the following affairs for JSR-356 Runtime to be made available:

A couple of broadcasting mechanisms are available for Java WebSocket API (JSR-356), being Servlet Container Classpath Scan (being part of Servlet 3) and Registration API for use in default Servlet Container, none of which is capable of the single front controller to process an HTTP request. In other words, DispatcherServlet cannot be used alongside WebSocket Handshake and for other HTTP requests.

By providing the server-specific RequestUpgradeStrategy, you'll need to support WebSocket of Spring in JSR-356 Runtime environment. With the current edition of Spring compatible with Tomcat 7.0.47+, Jetty 9.1+, GlassFish4.0+ and WildFly8.0+, you can experience wider compatibility when WebSocket Runtime becomes available.

One last thing to consider is to have the Servlet Container compatible with JSR-356 work on SCI Scan that makes an application run slow (or very slow when specific conditions are met). If you can experience the obvious change upon upgrade into the Servlet Container into JSR-356-compatible edition, you can have the web fragments (SCI scanning) either activated or deactivated as you desire (in other words, make absolute-ordering from web.xml available or not). Use of absolute-ordering in web.xml is feasible for the sequence of web fragment to be obvious (in other words, to have scanning unused).

5. Configuring WebSocket Engine

You can set configuration properties such as message buffer size, idle timeout and other runtime properties for each WebSocket Engine. You can simply add ServletServerContainerFactoryBean in the WebSocket java config.

```
@Configuration
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {
```

```
	@Bean
	public ServletServerContainerFactoryBean createWebSocketContainer() {
		ServletServerContainerFactoryBean container = new ServletServerContainerFactoryBean();
		container.setMaxTextMessageBufferSize(8192);
		container.setMaxBinaryMessageBufferSize(8192);
		return container;
	}

}
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:websocket="http://www.springframework.org/schema/websocket"
	xsi:schemaLocation="
		http://www.springframework.org/schema/beans
		http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/websocket
		http://www.springframework.org/schema/websocket/spring-websocket.xsd">

	<bean class="org.springframework...ServletServerContainerFactoryBean">
		<property name="maxTextMessageBufferSize" value="8192"/>
		<property name="maxBinaryMessageBufferSize" value="8192"/>
	</bean>

</beans>
```

To configure client -side WebSocket , use WebSocketContainerFactoryBean(XML) or ContainerProvider.getWebSocketContainer() ..

If you intend to use Jetty, you'll need to work on pre-configured WebSocketServerFactory added on to DefaultHandshakeHandler of Spring via WebSocket java config.

```
@Configuration
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {

	@Override
	public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {
		registry.addHandler(echoWebSocketHandler(),
			"/echo").setHandshakeHandler(handshakeHandler());
	}

	@Bean
	public DefaultHandshakeHandler handshakeHandler() {

		WebSocketPolicy policy = new WebSocketPolicy(WebSocketBehavior.SERVER);
		policy.setInputBufferSize(8192);
		policy.setIdleTimeout(600000);

		return new DefaultHandshakeHandler(
				new JettyRequestUpgradeStrategy(new WebSocketServerFactory(policy)));
	}

}
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:websocket="http://www.springframework.org/schema/websocket"
	xsi:schemaLocation="
		http://www.springframework.org/schema/beans
		http://www.springframework.org/schema/beans/spring-beans.xsd
```

```
        http://www.springframework.org/schema/websocket
        http://www.springframework.org/schema/websocket/spring-websocket.xsd">

    <websocket:handlers>
        <websocket:mapping path="/echo" handler="echoHandler"/>
        <websocket:handshake-handler ref="handshakeHandler"/>
    </websocket:handlers>

    <bean id="handshakeHandler" class="org.springframework...DefaultHandshakeHandler">
        <constructor-arg ref="upgradeStrategy"/>
    </bean>

    <bean id="upgradeStrategy" class="org.springframework...JettyRequestUpgradeStrategy">
        <constructor-arg ref="serverFactory"/>
    </bean>

    <bean id="serverFactory" class="org.eclipse.jetty...WebSocketServerFactory">
        <constructor-arg>
            <bean class="org.eclipse.jetty...WebSocketPolicy">
                <constructor-arg value="SERVER"/>
                <property name="inputBufferSize" value="8092"/>
                <property name="idleTimeout" value="600000"/>
            </bean>
        </constructor-arg>
    </bean>

</beans>
```