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

SockJS Fallback Options

http://docs.spring.io/spring/docs/current/spring-framework-reference/html/websocket.html#websocket-fallback

If your browser is not compatible with webSocket or unavailable due to restricted proxy in the network, you may use the fallback option available in Spring, an based SockJS-based emulation option for WebSocket API.

1. Outline of SockJS

Intended for allowing the application to use WebSocket API, SockJS supports fallback option by which you can use the webSocket API without altering the codes.

Composition of SockJS

- SockJS protocol
- SockJS client - javascript library
- Implementing SockJS Server - Provided by spring-websocket

SockJS is compatible with various sorts of browsers and browser versions thanks to its technical versatility. A total of three transfer types are available. **WebSocket, HTTP Streaming, HTTP Long Polling**. Refer to the following for more details:

In an attempt to obtain the basic server information, the SockJS client calls “GET /info”, followed by determining the type of transfer that SockJS will use. WebSocket is on a priority list, with HTTP Streaming > HTTP (long) polling followed.

Transfer request features the following URL structure.

http://host:port/myApp/myEndpoint/{server-id}/{session-id}/{transport}

- {server-id} - Used solely for routing the request to the cluster.
- {session-id} - Associated with the HTTP request that SockJS session has.
- {transport} - Type of transport and transfer ex> websocket, xhr-streaming, etc.

For transfer via webSocket you need only one HTTP request for webSocket handshaking, followed by which the messages are exchanged via socket.

HTTP transfer requires more requests as Ajax/XHR streaming is used for a single long-running request for messaging from server to client, followed by additional request for HTTP POST used for sending messages from client to server. Note that long polling is similar to XHR streaming, save that the current request is concluded after the server responds to the client.

SockJS also involves the minimal level of message framing as its server sends out an open frame (“o”) at the early stage, with the message transferred by way of JSON-encoded arrays [“message1”,”message2”]. The character “h” (heartbeat frame) is submitted when the message flow is suspended for more than 25 seconds, with the character “c” (close frame) used for closing the concerned session.

You can refer to the browser for more information. With SockJS providing debug flag and fixating the transfer type, you'll need to refer to the each and every type of transfer. You can activate the logging TRACE in “org.springframework.web.socket” to refer to the logs.

2. Activating SockJS

You can simply work the configuration to activate SockJS as follows:

@Configuration

```
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {

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

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

}
<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:sockjs/>
	</websocket:handlers>

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

</beans>
```

Note that the foregoing configuration is intended for Spring MVC and must be included in DispatcherServlet (when application context is established hierarchically). Like the Spring webSocket is available for both SpringMVC and the rest, SockJS serves the additional function as well, via SockJsHttpRequestHandler.

On the browser (client) side, you'll need to use sockjs-client to emulate W3C WebSocket API, by which the server communication is established to auto-select the most optimal type of transfer you can make. Manual configuration is also available for a couple of options.

3. HTTP Streaming (Ajax/XHR vs IFrame) in IE 8 / 9

With IE 8/9 still being the most commonly used browser, SockJS is of importance as a broadly applied protocol.

Supporting Ajax/XHR Streaming via Microsoft XDomainRequest, SockJS is compatible with inter-domain communication. Note, however, that transfer of cookies, being one of the core factors in Java Application, is not supported. With SockJS designed not only for Java but also for varied server types, you need to make sure whether cookies are of importance for your project. The SockJS client thus uses Ajax/XHR Streaming or iframe-based technique.

Being the initial request from the SockJS client, '/info' is a request to choose the type of client transfer. For this particular request, you'll need to check whether the server application depends on cookie or not (for the purpose of authentication or session clustering with stick mode). SockJS in Spring automatically activates the attribute of sessionCookieNeeded, dependent on JSESSIONID cookie. You'll thus need to deactivate the attribute manually to have the SockJS client use xdr-streaming in IE 8/9.

If your transfer is iframe-based, you may from time to time need to block iframe when the HTTP response header, X-Frame-Option, is configured for DENY, SAMEORIGIN or ALLOW-FROM <origin>. Note that these headers are intended to defend against clickjacking.

In Spring Security 3.2+, the header X-Frame-Options are supported to configure every response, signifying that the Spring Security Java Config has the default value of DENY. Although the headers are not needed for the Spring Security XML configuration, you'll need to set the header default as time passes.

You can refer to Section 7.1. Default Security Headers of Spring Security for more information on the header configuration of X-Frame-Options. You can also refer to SEC-2501 for backgrounds of such configuration.

When you intend to add the X-Frame-Option response header (using Spring Security), you'll need to configure the headers for either SAMEORIGIN or ALLOW-FROM <origin>. You'll also need to refer to the location of the SockJS client, loaded iframe-based. Your default configuration for iframe would download the SockJS client (sockJS.js) from the CDN location, in which case you need to manually configure for the same origins.

You can apply the following coding example for Java Config. You need to access <websocket:sockjs> for XML configuration.

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

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/portfolio").withSockJS()
                .setClientLibraryUrl("http://localhost:8080/myapp/js/sockjs-client.js");
    }

    // ...

}
```

If you are on the early stage of development, make sure "SockJS client devel mode" is activated to keep your browser from caching SockJS requests (such as iframe). Check out SockJS client for more information.

4. Heartbeat Messages

Make sure the SockJS protocol transfers the heartbeat message to prevent the proxies from recognizing connection as a hung. Configurations available for Spring SockJS, representing the attribute of heartbeatTime, are used to set the frequency as the default value of 25-sec, according to IETF recommendation, is applied with no message involved.

When you intend to use STOMP over WebSocket/SockJS, the SockJS heartbeat gets deactivated when the client and server for STOMP agree upon exchange of heartbeat.

For scheduling of heartbeat, Spring SockJS permits manual configuration of TaskScheduler, loaded from the default thread pool determined by the number of processors available. Keep in mind that the configuration should meet the request-specific demands.

5. Servlet 3 Async Requests

SockJS transfer via HTTP streaming and HTTP long polling allows extensive opens throughout the connection. See more for how Servlet 3 Async Requests work.

Servlet 3 async in Servlet Container allows the Servlet thread in process out and record responses from other Servlet thread.

Notable for Servlet API is that it does not alert user in case of sudden client disconnection (See SERVLET_SPEC-44). However, the Servlet container accordingly throws an exception to write the response. With the Spring SockJS supporting the heartbeat message with the interval of 25 seconds, the client disconnect is observed within the interval set (or shorter, if the interval is set accordingly).

As a consequence, network IO failure are frequently occurred upon client disconnects, in which case logs are filled with the stack trace. In identification of the client disconnect, Spring attempts to record the minimum volume of messages (The log is preferred by the log category DISCONNECTED_CLIENT_LOG_CATEGORY defined in AbstractSockJsSession. If you want to check for the stack trace, you need to configure the corresponding log category for TRACE)

6. CORS Headers for SockJS

The SockJS protocol uses the CORS headers in transferring XHR streaming and polling to stay compatible with cross-domain. When left undetected in the response, the CORS headers are to be added automatically. Spring SockJsService skips configuration when the CORS headers has already been configured.

Refer to the following value, containing headers, expected out of SockJS protocol:

“Access-Control-Allow-Origin” - intitialized from the value of the “origin” request header or “*”. “Access-Control-Allow-Credentials” - always set to true. “Access-Control-Request-Headers” - initialized from values from the equivalent request header. “Access-Control-Allow-Methods” - the HTTP methods a transport supports (see TransportType enum). “Access-Control-Max-Age” - set to 31536000 (1 year).

For more implementation examples in detail, find AbstractSockJsService.addCorsHeaders() and TransportType enum from the sources.

If you consider the CORS configuration to exclude the URL corresponding to SockJS endpoint prefix, you may have Spring SockJsService to proceed with.