WebSocket

This page summarizes the current status of WebSocket support in CXF. This feature is currently only available in trunk (CXF 3.0.0-SNAPSHOT).

Using the WebSocket transport

The intention for providing the WebSocket transport in CXF is to turn CXF endpoint services capable of reading or writing data over the socket that is connected to the client. For example, a JAXRS service using the WebSocket transport can continuously push or write data (e.g., status changes, events, etc) to the client once the client establishes a WebSocket connection to the service.

Current Status

Short summary

- The code resides in rt/transport/http-websocket and it enables cxf services to be invoked over websockets.
- It supports both the embedded mode and the servlet container mode. The former can be used in the standalone setup that uses an embedded jetty server and the latter can be used in the container setup that uses the servlet container provided by the container.
- Some test cases are located in systests/jaxrs/src/test/java/org/apache/cxf/systest/jaxrs/websocket and systests/jaxws/src/test/java/org/apache/cxf/systest/jaxws/websocket. There is also a browser based demo at distribution/src/main/release/samples/jax_rs/websocket.
- Several additional libraries are required to enable this feature depending on the usage. Concretely, jetty-websocket is required for the embedded standalone mode (i.e., the endpoint is given with the host and port part as in address="ws://host:port/path"). For the servlet mode (i.e., the endpoint is given using a relative path (i.e., address="/path"), it requires either jetty-websocket to use jetty or atmosphere-runtime and a specific websocket implementation such as jetty-websocket or tomcat-websocket to use that specific container's websocket capability.

Usage Patterns

We have the following message exchange patterns to use this websocket transport for cxf service invocation.

1. the client opens a websocket to some URL hosting a cxf service (e.g. ws://localhost:8080/cxf/myapp/books)
2. the client can subsequently invoke an operation provided by this service (e.g., the book order operation at /myapp/books/order/123) by sending a request message that a simplified HTTP request message.

Request Syntax

```
Request = Request-Line CRLF
             *(( header ) CRLF)
             CRLF
             [ body ]

Request-Line  = Method SP Request-URI
```

Method uses the HTTP methods.

header may include any headers that are required by CXF to determine the operation and expected by the operation itself (e.g., Content-Type).

3. the response message returned by the the service is sent back to the client as

Request Syntax

```
Response = [ Status-Line CRLF ]
           *(( header ) CRLF)
           CRLF
           [ body ]

Status-Line = Status-Code
```

Status-Code uses the HTTP status codes.

4. the service can subsequently push further data using jaxrs StreamingOuptut or writing to the injected servlet response's output stream. In this case, the message is simply returned to the client without Status-Line and may or may not include headers.

(Note: need for discussing the above binding syntax).

Further details

- the text encoding is fixed when text is sent as bytes. Currently, when the text is sent in the binary mode, its encoding is assumed to be utf-8.
- if the client and the service want to explicitly exchange a sequence of requests and responses, they are responsible in passing some id and ref-id in the exchanged message's headers so that each response can be correlated to its request.
- for correlating requests and their responses that are send back asynchronously, the request message may include the RequestId header and the corresponding response may set this value in its ResponseId header.
- asynchronous web service calls that are performed using a decoupled endpoint using the http transport can be performed using the websocket transport without using a decoupled endpoint.

Open questions

- the content-length header is not used to determine the length of the message, as the message end can be determined for a websocket message using other means (i.e., either its length when used a block-mode or the fragment's indicator). So it is not clear whether we can filter out the content-length.
- the protocol format needs to be clarified (see above regarding the protocol format used currently) and possibly to several variations binding variations.

TODOs of immediate interests

- allow setting a buffer size to switch to the fragment transfer mode over websockets

Currently, what is written over out.write(byte[] b, int offset, int length) will be written as a websocket message (e.g., triggering a single onMessage event to the client). We should allow sending back a sequence of fragments and terminates the transmission at the flushing step (i.e., out.flush() or of some sort).

- native string message transfer

Currently, strings are converted into utf-8 bytes and sent using the byte transfer mode of websocket. We could add the native string/text transfer as websocket supports it. However, this should be allowed only when the entity body is absent or of string/text.

- add more browser based sample client applications

we can update the logbrowser sample. There is a browser based demo in the samples collection (see above). This demo uses the current default binding which is not javascript friendly. To support browser based applications, we need different bindings.

TODOs of future interests

- supporting a direct output stream connection from the client application to the service application. Currently, from the service application to the client, a direct output stream can be kept open after the initial call from the client to the service takes place. Thus, the service can continue to push data directly to the client over this output stream. Similarly, it would be interesting to establish a direct output stream from the client to the service so that the client can continue to push data directly to the service.