

Server-Sent Events (SSE) in Java EE 8

Santiago Pericas-Geertsen

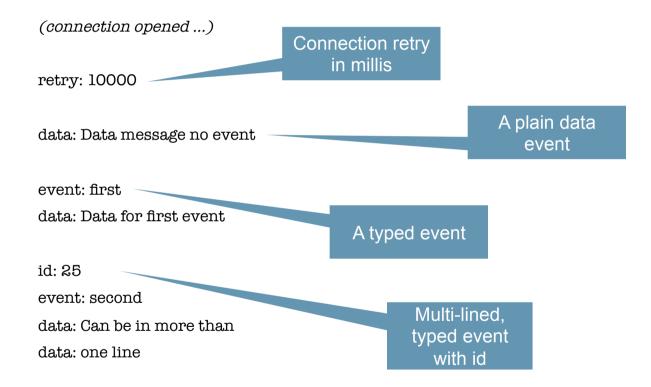


### What is SSE?

- Standardized "Comet"
- Server-to-client streaming of text data
- Long-lived HTTP connection
- Client message notifications as DOM events
- Per-event data streams over same connection
- Browsers handle connection management and stream parsing
- Mime type is text/event-stream



## **Event Stream Example**





### **Goals of Presentation**

Proposed location in the Java EE 8 landscape for an SSE API



# **JSR Options Evaluated**

- 1. Servlet
- 2. WebSocket
- 3. Standalone
- 4. JAX-RS



#### **Servlet**

- √The common denominator for HTTP in EE
- ✓ SSE runs over an HTTP connection
- Async connections already supported
- √SSE is HTTP streaming
- X Right level of abstraction for HTML5 developers?
- X Application scope vs. connection scope
- X Will focus on HTTP/2
  - x Server push in HTTP/2 not for streaming data
- X Operates at HTTP not resource level



#### WebSocket

- ✓ API similarities
  - ✓ Lifecycle events and scopes (connection scope)
  - ✓ One-way vs. two-way
- ✓ Interface and annotation sharing
  - Even if they come from javax.websocket?
- ✓ Related being part of the "HTML5 stack"
- What to do with javax.websocket packages?
- X Relative complexity of API:
  - x Programmatic vs. declarative endpoints
  - x Synchronous and asynchronous messaging
- X Unclear semantics for @OnClose and @OnError
- Why bring WS implementation if only SSE is needed?



## WebSocket Example (Annotated)

```
@ServerEndpoint(sse=true)
public class StockTicker {

@OnOpen
public void onOpen(Session s) {
    new StockThread(s).start();
    }
}
```



Extends existing

annotation

### **StockThread Class**

```
class StockThread extends Thread {
    private Session ss;
    private AtomicBoolean ab =
        new AtomicBoolean(true);

    public StockThread(Session ss) {
        this.ss = ss;
    }

    public void terminate() {
        ab.set(false);
    }
```

```
@Override
public void run() {
    while (ab.get()) {
        try {
            ss.send(new StockQuote("..."));
            // ...
        } catch (IOException e) {
            return;
        }
     }
}
```



## **WebSocket Client Example**

```
@ClientEndpoint
class StockTickerClient {
  @OnMessage
  public void onMessage(StockQuote sq) {
    // ...
ClientManager client = ClientManager.createClient();
client.connect(StockTickerClient.class,
              new URI("http://example.com/tickers"));
```



#### **Standalone**

- √Small specification, small runtime footprint
  - ✓ No need for a large runtime just for SSE
- ✓ Declarative API using own set of annotations
  - ✓ @OnOpen, @OnClose, etc.
- x Yet another JSR
- X Resourcing, additional team
- X Potential duplication of annotations and interfaces
- X New client API from scratch?
- X Unclear semantics for @OnClose



### **Standalone Example**

```
@SseServerEndpoint
public class StockTicker {
 @OnOpen
 public void onOpen(SseSession s) {
   new StockThread(s).start();
```



### **JAX-RS**

- ✓ Ease of implementation Simplicity
  - ✓ SSE is already supported in Jersey
  - ✓ JAX-RS supports async connections
- ✓ Combining regular HTTP and SSE connections
- ✓ Popularity of JAX-RS
- ✓ Streaming a REST resource with special media type
- ✓ Sharing client API concepts like configuration, targets, etc.
- x Make JAX-RS even larger
- SSE as streaming resources may look "unrestful"
- Why bring JAX-RS implementation just for SSE
  - ✓ Many applications will use JAX-RS anyway!



## **JAX-RS Example (Jersey)**

```
@Path("tickers")
public class StockTickerResource {
  @GET @Produces("text/event-stream")
  public EventOutput getQts() {
    EventOutput eo = new EventOutput();
    new StockThread(eo).start();
    return eo;
```



## **JAX-RS Client Example (Jersey)**

```
WebTarget target = client.target("http://example.com/tickers");
EventSource eventSource = new EventSource(target) {
     @Override
     public void onEvent(InboundEvent inboundEvent) {
         StockQuote sq = inboundEvent.readData(StockQuote.class);
          // ...
     }
     };
eventSource.open();
```



### **Recommendation: JAX-RS**

#### Why?

- SSE is streaming HTTP resources with special media type
  - Already supported in JAX-RS
- JAX-RS already popular for HTML5 applications
  - Footprint is a non-issue in practice!
  - Angular JS + JAX-RS becoming quite popular
- Small extension
  - Server API: new media type, EventOutput (Broadcaster)
  - Client API: new handler for SSE events
- Convenience of mixing other HTTP operations with SSE GET's

