Server Push and Client-Server

CS Slides #1

The Chat Problem

How to build a Web Application Chat?

- Messages broadcasted to all clients …
- … but HTTP “can’t do” that

How to get the server push?

<u>All this should run GlassFish!</u>

Real-time Web

"The real-time web is a set of technologies and practices that enable users to receive information **as soon** as it is published by its authors, rather than requiring that they or their software check a source periodically for updates [polling]"

//Wikipedia

This is **soft** real time (**as soon** ~ in a few seconds)

Real Time Techniques

History
- Java Applets
- Polling (let JavaScript continuously call server)

Comet
- Long-Polling (tweaking HTTP to it's limits, a "hack")
- HTTP Streaming

Now
- Server Sent Events
- WebSockets

Server Sent Events

To enable servers to push data to Web pages over HTTP or using dedicated server-push protocols
// W3C

Basically
- Plain old HTTP (others possible)
- MIME: text/event-stream
- API
- Not supported by IE, other's browsers: late versions
- Java Server Side support by Jersey (JAX-RS)

WebSocket

To enable Web applications to maintain [full duplex] bidirectional communications with server-side processes // W3C

Basically

- Not like TCP Sockets
- Message based (like UDP) reliable (like TCP)
- Initial HTTP-handshake then switch to websocket protocol (ws://.. and wss://)
- W3C WebSocket interface (client side)
- Server side depends on platform
- Some comments

Websocket Client Side

Create socket, attach handlers, send via socket

```
// Use WebSocket API from JavaScript
var host = "ws://localhost:8000/socket/server/...";
var socket = new WebSocket(host);

// Handlers
socket.onopen = function(){
     // Check socket.readyState
}
socket.onmessage = function(msg){
     // Incoming from server handled here
}
socket.onclose = function(){
     // Check socket.readyState
}

// Send
socket.send('… some data… ');
```

Websocket Server Side

Java API for WebSocket (new in JEE 7)

Basically
- javax.websocket.*
- Web socket endpoints (Classes)
- Annotation and path, @ServerEndpoint("/echo")
- Life cycle methods handling lifecycle events (open, close, error, message)
- Sending messages to clients using a RemoteEndpoint object
- Send and receive text and binary messages
- Send ping-frames, receive pong-frames

WebSocket EndPoint

Websocket lifecycle methods

```
@ServerEndpoint(value = "/wschat", encoders = {ChatMessageEncoder.class}, decoders =
                                                        {ChatMessageDecoder.class})
public class WSChatEndpoint {
    private Chat chat;
    @Inject
    public WSChatEndpoint(Chat chat) { this.chat = chat; }

    @OnOpen
    public void onOpen(Session session) { … }

    @OnClose
    public void onClose(Session session) { … }

    @OnMessage
    public void onMessage(InMessage inMsg, Session session) throws Exception {
    // Call modell, chat.doSomething()

    @OnError
    public void onError(Session session) { … }

}
```

WebSocket Session

The Session object
- Represents a conversation between this endpoint and the remote endpoint
- Available as a parameter lifecycle methods
- If incoming (i.e. @OnMessage method) use Session object for response, see below
- If not a response, store Session object as instance variable for later use (get it in @OnOpen method).
- Retrieve RemoteEndpoint object from Session, use to send
    - Session.getBasicRemote, Session.getAsyncRemote (returns RemoteEndpoint)
    - void RemoteEndpoint.Basic.sendText(String text)
    - void RemoteEndpoint.Basic.sendBinary(ByteBuffer data)
    - void RemoteEndpoint.Basic.sendObject(Object o) // Encode/decoder!
    - void RemoteEndpoint.sendPing(ByteBuffer appData)
    - void RemoteEndpoint.sendPong(ByteBuffer appData)

WebSocket Sending

```
// Echo message to sender
@OnMessage
public void onMessage(Session session, String msg) throws Exception  {
     session.getBasicRemote().sendText(msg);
}


// Broadcast message to all
@OnMessage
private void onMessage(InMessage inMsg, Session session) throws Exception {
   BasicMessage outMsg = new BasicMessage(inMsg);
   for (Session s : session.getOpenSessions()) {
         s.getBasicRemote().sendObject(outMsg);
   }
}
```

State Handling

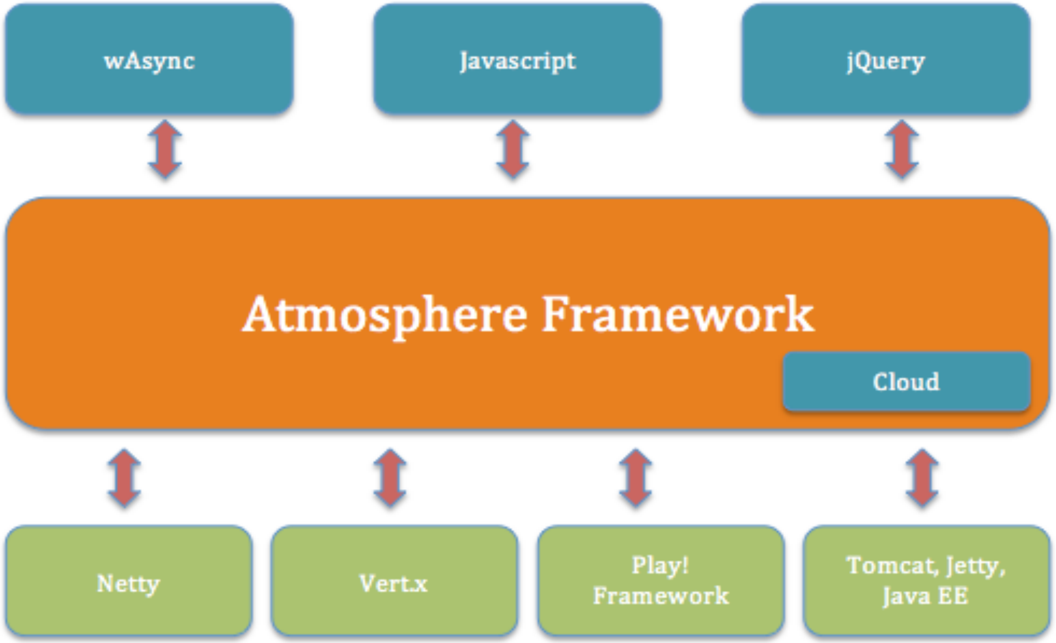
Instance (class) variables to store client state information

- Instance of endpoint class <u>for every connection</u>
- Session.getUserProperties, map for user properties
- Information common to all connected clients, use class variables (must ensure thread-safe access)

Atmosphere

Atmosphere : Realtime Client Server Framework

-Transparently supports WebSockets, Server Side Events (SSE), Long-Polling, HTTP Streaming and JSONP



Some version problems, use 2.1.5 or better

Primefaces Push

Remainder: PrimeFaces a higher level JSF component suite

- PrimeFaces Push part of framework
- Built on top of Atmosphere
- Versions differ, 3.5 vs 5.0 code rather different (both as code samples)

Primefaces 5.0 Push

```
// Element in page and JS snippet to display messages
<p:socket onMessage="handleMessage" channel="/notify" />

<script type="text/javascript">
   function handleMessage(facesmessage) {
       facesmessage.severity = 'info';
       PF('growl').show([facesmessage]); // PF = PrimeFaces object
    }
</script>
```

Also other possibilities, see code samples

Primefaces 5.0 Push, cont

```
// An endpoint with life cycle methods
@PushEndpoint("/notify")
public class NotifyResource {
    ...
    // Send to all on channel
   @OnMessage(encoders = {JSONEncoder.class})
    public FacesMessage onMessage(FacesMessage message) {
         return message;
    }
}

// CDI Bean to send
public void send() {
   EventBus eventBus = EventBusFactory.getDefault().eventBus();
   eventBus.publish(CHANNEL, new FacesMessage(
                 StringEscapeUtils.escapeHtml(summary),
                 StringEscapeUtils.escapeHtml(detail)));
}
```