Carnegie Mellon University

95-733 Internet of Things
Websockets in Java
Week Three

Week Three

- Learning objectives:
- Consider two data formats: XML and JSON
- Understand Java and Websockets
- Introduce Websockets and MQTT for IOT

XML Or JSON ? (1)

```
<Person firstName="John" lastName="Smith">
    <Address>
        <streetAddress>21 2nd Street</streetAddress>
        <city>New York</city>
        <state>NY</state>
        <postalCode>10021</postalCode>
    </Address>
    <phoneNumber type="home"> 212 555-1234 </phoneNumber>
    <phoneNumber type="fax"> 646 555-4567 </phoneNumber>
</Person>
```

From Wikipedia.org

XML Or JSON ? (2)

```
{
  "firstName": "John", "lastName": "Smith",
  "address":
    { "streetAddress": "21 2nd Street",
      "city": "New York", "state": "NY",
      "postalCode": "10021"
    },
   "phoneNumbers":
    [ { "type": "home", "number": "212 555-1234" },
      { "type": "fax", "number": "646 555-4567" } ]
}
```

If this text is in the variable coolText, then we can create a JavaScript object with the line of code:
var x = eval("(" + coolText + ")");
and access the data with x.phoneNumbers[0].
Better to use JSON.parse(text) and JSON.stringify(obj).

Which To Use?

Both formats are textual, coarse grained, and interoperable.

JSON has some advantages over XML:

- JSON messages are smaller and therefore faster.
- No namespaces (but see JSON-LD)
- JSON is simpler and simple is good (but see JSON-LD)
- For message signing and encryption, see JSON Web Token (JWT)

XML has some advantages over JSON:

- Namespaces built in
- Languages adopted by many industries
- XSLT can be used to transform a response into an appropriate format.
- XML Signature and XML Encryption for signing and encryption

Bottom line: JSON is more popular. Note: Both textual formats are slow.

jQuery AJAX Example

```
<!DOCTYPE html>
<html>
<head>
<script src="jquery.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("div").load('test1.txt');
  });
});
</script>
</head>
<body>

<div><h2>Let AJAX change this text</h2></div>
<button>Change Content</button>

</body>
</html>
```

The syntax is: ***$(selector).action()***
A $ sign to define/access jQuery
A (*selector*) to "query (or find)" HTML elements
A jQuery *action*() to be performed on the element(s)

Document ready before running jQuery.

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_ajax_load

Dojo Ajax Example

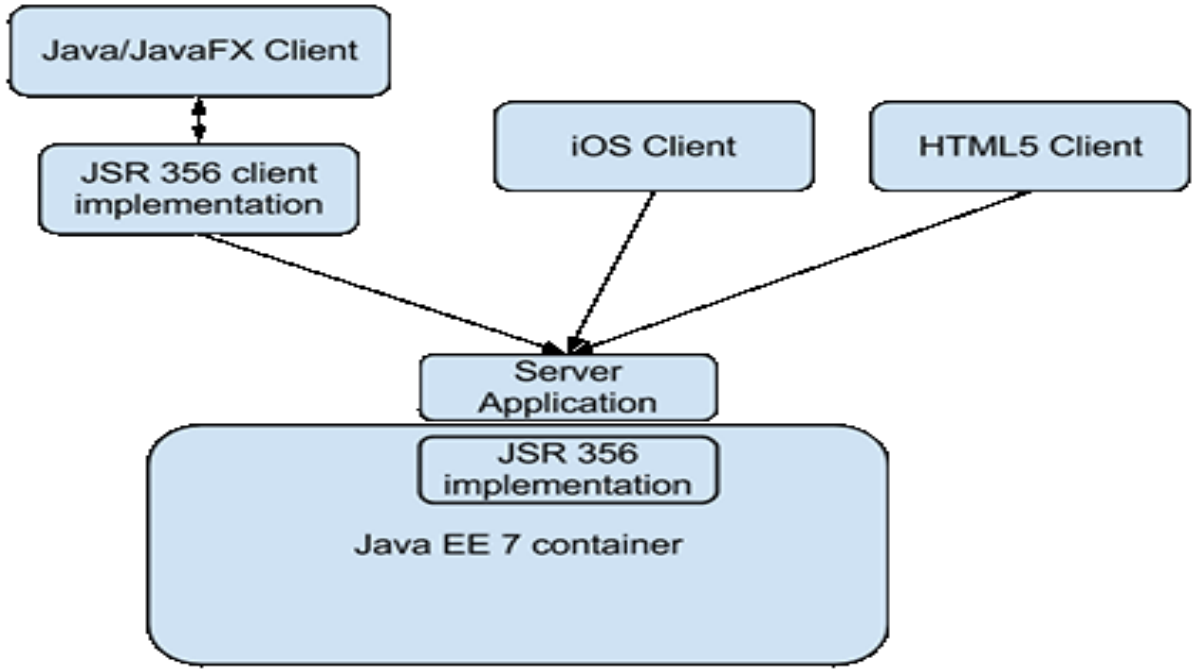
```
dojo.io.bind ( {
   url : “getCityState.php?zip=“ + zip,
   load: function( type, data, evt ) {
                  var place = data.split( ‘, ’ );
                  if(dojo.byID(“city”).value = “”
                        dojo.byID(“city”).value = place[0];
                  if(dojo.byID(“state”).value = “”
                        dojo.byID(“state”).value = place[1];
          },
   error: function(type, data, evt ) {
              alert(“Error in request, returned data:”+data);
          },
   method: “GET”,
   mimetype: “text/plain”
   }
);
```

Note the use of JS object to describe an AJAX call.

From Pg. 422 Of Sebesta

Web Sockets and Java

Javascript API defined by W3C – Available on many browsers.
Java client side API and server side API - see JSR 356 part of JEE 7



From Oracle

Web Sockets

Lifecycle Events

The typical lifecycle event of a WebSocket interaction goes as follows:

One peer (a client) initiates the connection by sending an HTTP handshake request.
The other peer (the server) replies with a handshake response.
The connection is established. From now on, the connection is completely symmetrical.
Both peers send and receive messages.
One of the peers closes the connection.
Most of the WebSocket lifecycle events can be mapped to Java methods, both in the annotation-driven and interface-driven approaches.

Note that Web Sockets don't describe the conversation that will take place. This is left to the layer above the web socket layer.

From Oracle

Client side Web Sockets (1)

```
// Establish the socket in Javascript:

// Open a new WebSocket connection (ws), use wss for secure
// connections over SSL.

var wsUri = "ws://" + document.location.host +
            document.location.pathname +
            "whiteboardendpoint";
var websocket = new WebSocket(wsUri);
```

Client side Web Sockets (2)

```
// Prepare to send and receive data.
// A callback function is invoked for each new message from
// the server
websocket.onmessage = function(evt) {
   onMessage(evt)
};
function onMessage(evt) {       // Called for us when a message
     drawImageText(evt.data); // arrives.
}
function sendText(json) {       // We call this when we want.
    websocket.send(json);
}
```

Server side Web Sockets in Java (1)

```
// The Decode is used for messages arriving from the peer.
// The Encoder is used for messages being sent to the peer.
@ServerEndpoint(value="/whiteboardendpoint",
      encoders = {FigureEncoder.class},
      decoders = {FigureDecoder.class})

public class MyWhiteboard {

// create a synchronized (thread safe) set of sessions in a hashset
// names this collection peers
private static Set<Session> peers =
                  Collections.synchronizedSet(new HashSet<Session>());
```

Server side Web Sockets in Java (2)

```
// when a message arrives from the decoder, call the peers
@OnMessage
public void broadcastFigure(Figure figure, Session session)
                                        throws IOException, EncodeException {
System.out.println("broadcastFigure: " + figure);
// for each peer in the peers set of sessions
for (Session peer : peers) {
    if (!peer.equals(session)) {
        // this is not being sent back to the caller
        peer.getBasicRemote().sendObject(figure);
     }
  }
}
// when sendObject is called, the Encoder is executed.
```

Decoder Goes on Arrival

```
public class FigureDecoder implements Decoder.Text<Figure> {

    @Override
    public Figure decode(String string) throws DecodeException {
        System.out.println("decoding: " + string);
        JsonObject jsonObject = Json.createReader(
                new StringReader(string)).readObject();
        return new Figure(jsonObject);
    }
```

Decode a message and create a Java object.
The Java object involved is of type Figure.
So, from Text we get a Figure.

Encoder Goes Before Sending to Peer

```
public class FigureEncoder implements Encoder.Text<Figure> {

    @Override
    public String encode(Figure figure) throws EncodeException {
        System.out.println("Encoding ");
        return figure.getJson().toString();
    }
```

Encode a Java object.
The Java object involved is of type Figure.
So, from Figure we encode to Text.

Sending JSON data

```
// Microsoft example: Send text to all users through the server
function sendText()
  { // Construct a msg object containing the data the server
    // needs to process the message from the chat client.
    var msg = { type: "message",
                  text: document.getElementById("text").value,
                  id: clientID,
                  date: Date.now()
                };
   // Send the msg object as a JSON-formatted string.
   exampleSocket.send(JSON.stringify(msg));
   // Blank the text input element, ready to receive the next line of text
   // from the user.
   document.getElementById("text").value = ""; }
```

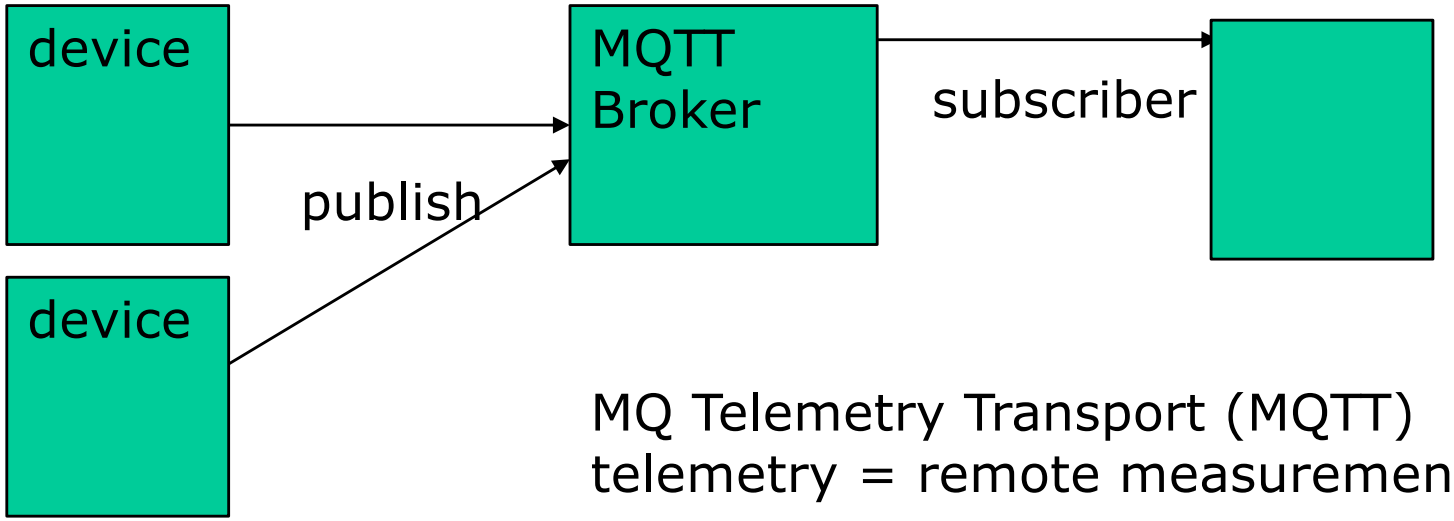
WebSockets using JEE

A message arrives from a peer – it needs to be decoded into a Java object. Then, it may be processed by the Java code.

A message is being sent to a peer. It needs to be encoded before being placed on the wire.

Image from http://www.mastertheboss.com/javaee/websockets/websockets-using-encoders-and-decoders

Web Sockets and IoT



MQ Telemetry Transport (MQTT)
telemetry = remote measurements.

MQTT runs on top of TCP. Can be carried by web sockets for, e.g., real time dashboards.

MQTT-SN for sensor networks. No TCP or IP. Can use e.g. ZigBee or BlueTooth.

MQTT is an established publish subscribe messaging protocol for intermittent networks.

With MQTT over WebSockets, a browser can become an MQTT device – subscribing to or publishing messages.