Novelties in Java™ EE 7: Java API for WebSocket (JSR-356) & Java API for JSON Processing (JSR-353)

Trayan Iliev

IPT – Intellectual Products & Technologies
e-mail: tiliev@iproduct.org
web: http://www.iproduct.org

Agenda (1)

- Problems with traditional request-response model of HTTP
- Bidirectional data streams are required by certain applications
- Server-push alternatives: polling, long-polling, and streaming
- IETF WebSocket protocol (RFC 6455)
- HTTP upgrade & WebSocket protocol details
- WebSocket security and extensibility – Origin validation, sub-protocols and extensions negotiation
- W3C JavaScript WebSocket API
- Java™ EE 7 WebSocket support: Java API for WebSocket (JSR-356)

Agenda (2)

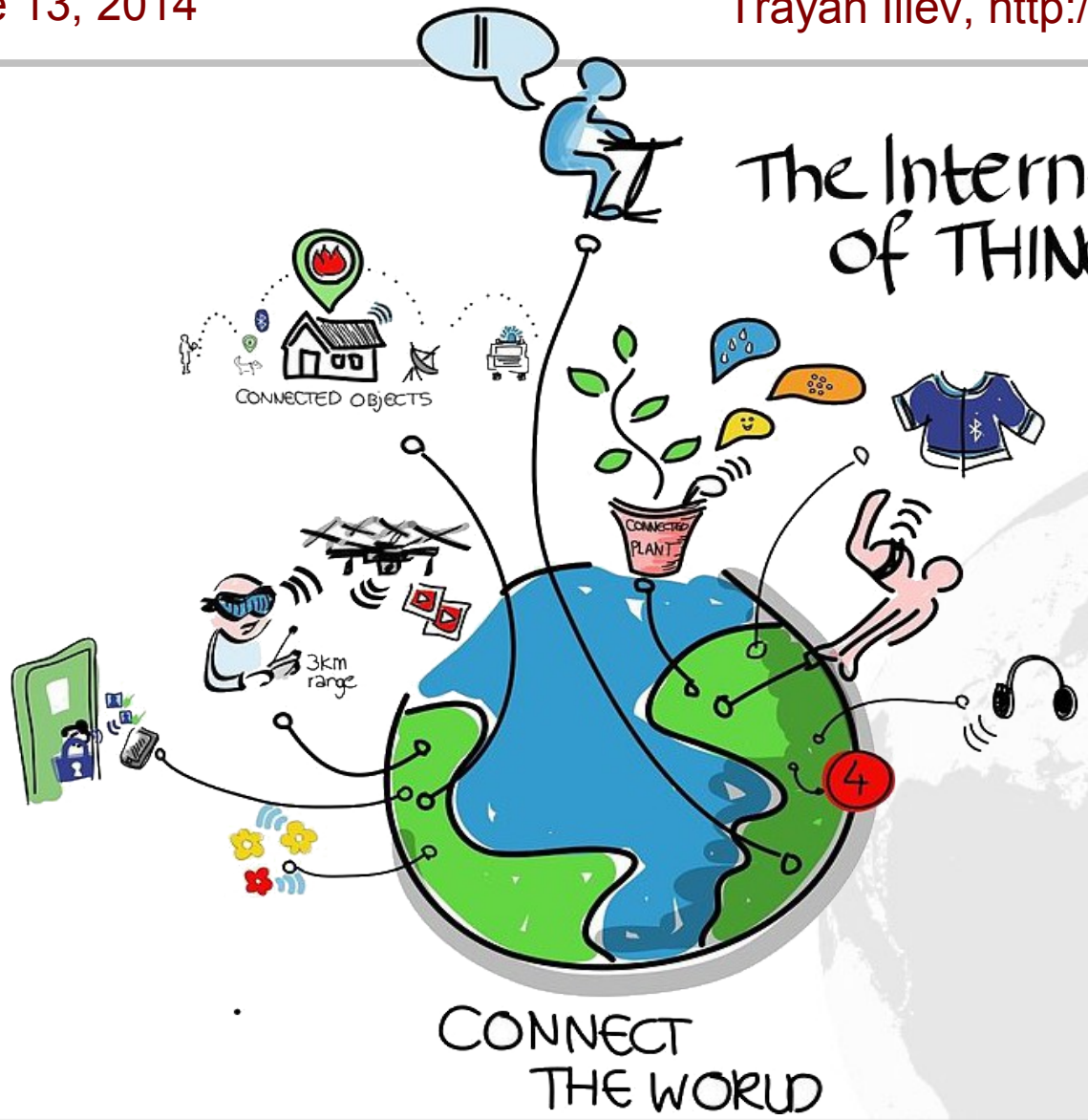
- Practical implementation of WebSocket client and server endpoints – programmatic & using annotations
- Main annotations and their use: @ServerEndpoint, @ClientEndpoint, @OnOpen, @OnClose, @OnMessage, @OnError, @PathParam
- Synchronous and asynchronous processing of messages
- Using custom sub-protocols and message types
- Customization of handshake – sub-protocols, extensions, Origin validation, controlling initialization of endpoint instances
- Sending and receiving text, binary and ping/pong messages

Agenda (3)

- Building custom Encoder/Decoder classes for easy marshalling/unmarshalling of arbitrary Java objects
- Integration with Java API for JSON Processing (JSR-353) for JavaScript™ friendly JSON serialization of data between client and server
- Let's try it (IPT Mobile Presentation demo)

Question

How many of you have heard that Internet of things (IoT) is the next major revolution comparable in effect with the Internet itself?



Source: http://commons.wikimedia.org/wiki/File:Internet_of_things_signed_by_the_author.jpg, Author: Wilgengebroed on Flickr, License: Creative Commons Attribution 2.0 Generic

Internet of Things (IoT)

Today computers—and, therefore, the Internet—are almost wholly dependent on human beings for information. Nearly all of the roughly 50 petabytes (a petabyte is 1,024 terabytes) of data available on the Internet were first captured and created by human beings—by typing, pressing a record button, taking a digital picture, or scanning a bar code. ... The problem is, people have limited time, attention and accuracy—all of which means they are not very good at capturing data about things in the real world. ... We're physical, and so is our environment ... If we had computers that knew everything there was to know about things ... we would be able to track and count everything, and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best. The Internet of Things has the potential to change the world, just as the Internet did. Maybe even more so.

— Kevin Ashton, 'That 'Internet of Things' Thing', RFID Journal, July 22, 2009

Internet of Things (IoT) Perspectives

- According to Gartner, there will be nearly 26 billion devices on the Internet of Things by 2020.
  [Gartner, 2013-12-12, http://www.gartner.com/newsroom/id/2636073]
- According to ABI Research, more than 30 billion devices will be wirelessly connected to the Internet of Things by 2020 (Internet of Everything)
  [ABI Research, https://www.abiresearch.com/press/more-than-30-billion-devices-will-wirelessly-conne]
- It's expected to be a 19 Trillion USD market
  [John Chambers, Cisco CEO, http://www.bloomberg.com/news/2014-01-08/cisco-ceo-pegs-internet-of-things-as-19-trillion-market.html]

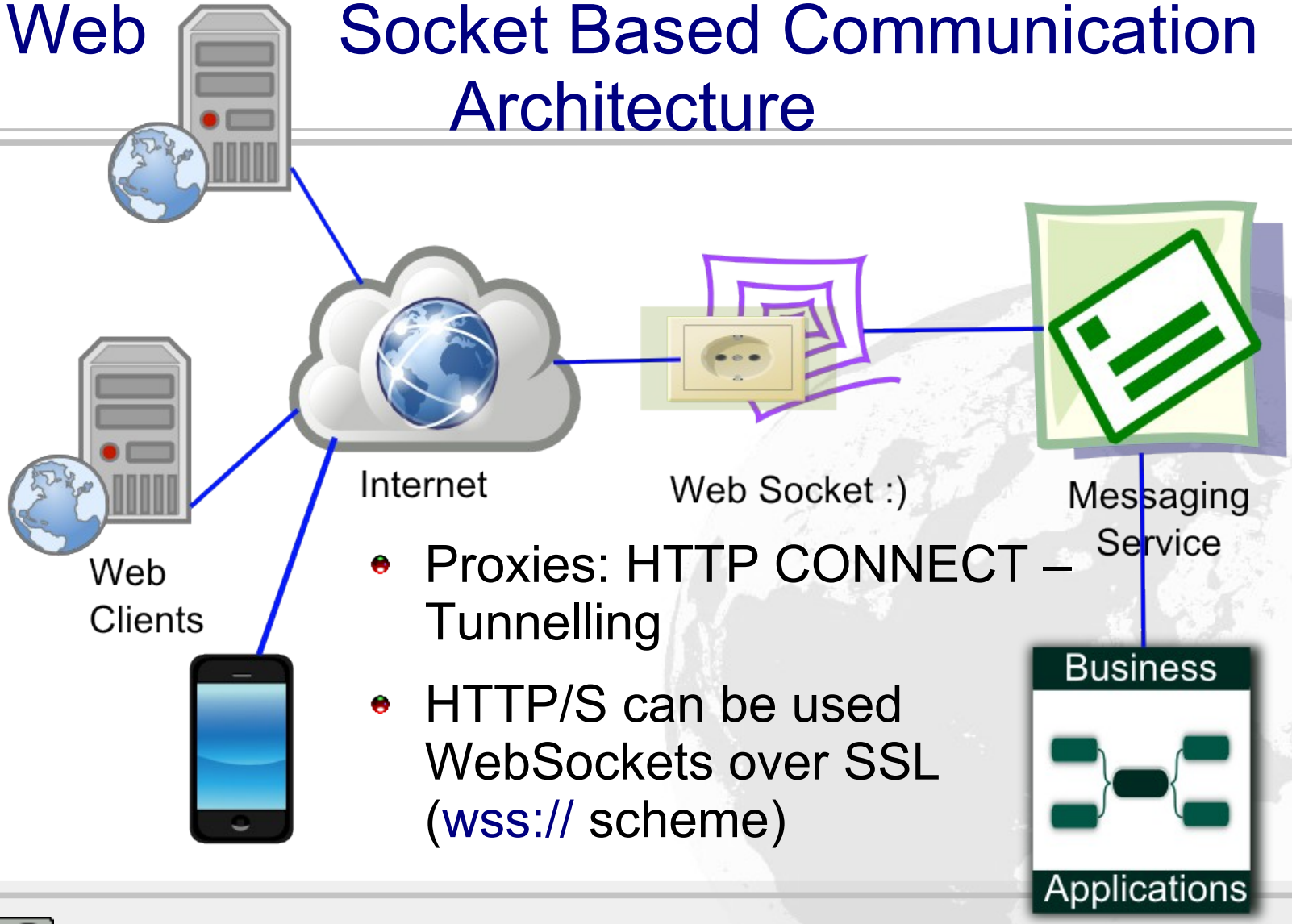
IoT and Web Sockets

"Basket of remotes" problem – we'll have hundreds of applications to interface with hundreds of devices that don't share protocols for speaking with one another

[Jean-Louis Gassée, Apple initial alumni team, and BeOS co-founder, http://www.mondaynote.com/2014/01/12/internet-of-things-the-basket-of-remotes-problem/]

- Only IPv6 addresses are not enough – IoT devices should be also easily and directly accessible for users and [their] agents
- In read/write mode
- Preferably using a standard web browser
- Even behind firewalls

Web Socket Based Communication Architecture



- Proxies: HTTP CONNECT – Tunnelling
- HTTP/S can be used WebSockets over SSL (wss:// scheme)

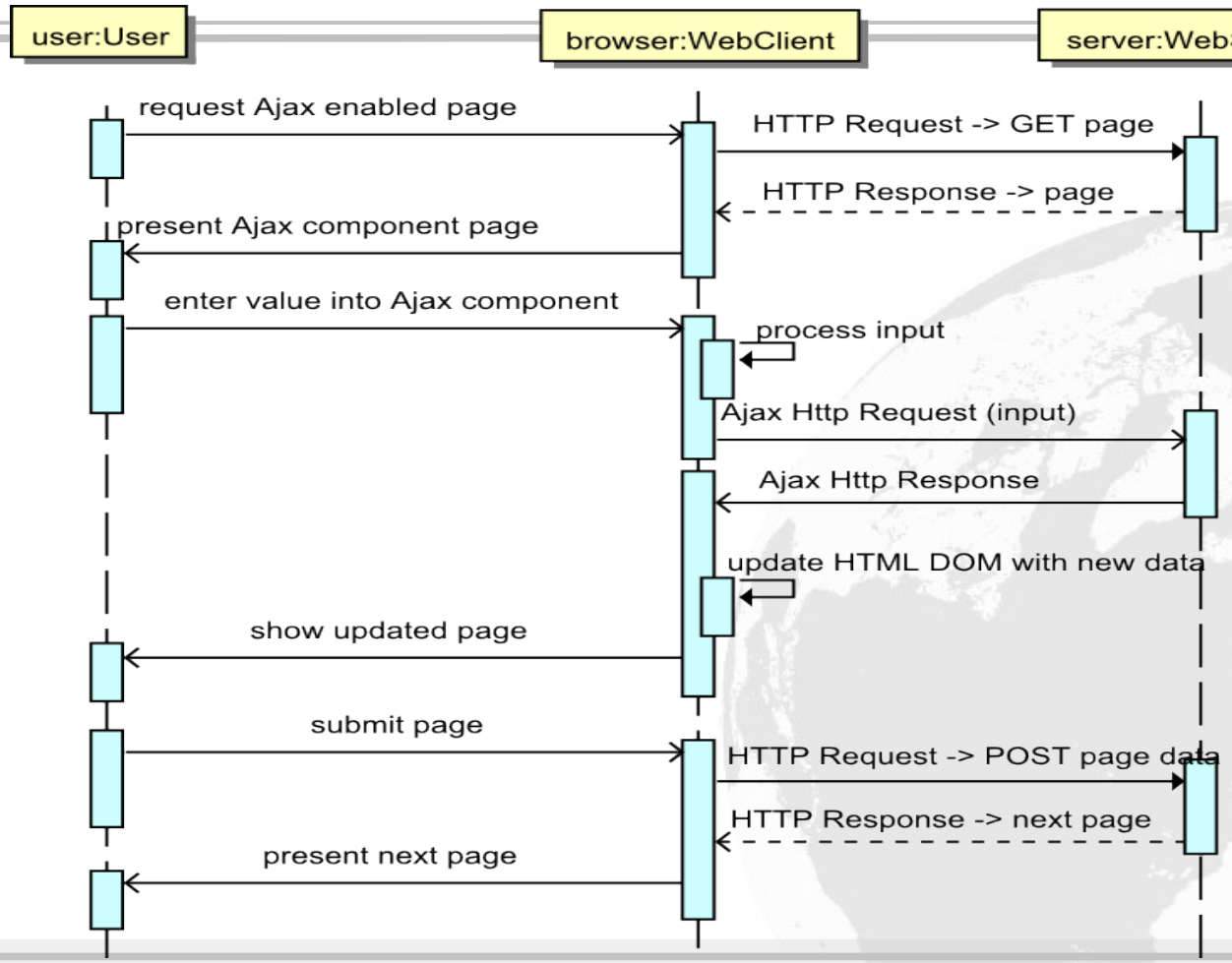
WebSocket Main Applicatin Areas

- Massively multiplayer online role-playing game (MMORPG)
- Online trading – large scale auctions, stock tickers
- Interactive synchronous communication – chat, audio- & video-conferencing
- Collaborative authoring, groupware & social applications - including modelling and art
- Dynamic data monitoring and control – e.g. management dashboards presenting SLA, KPI and BI data in real time
- Remote observation and control of devices and services – e.g. remote monitoring of home security, energy consumption, data center services and devices performance visualizations

Other Alternatives for Bidirectional Communication over HTTP

- Ajax polling
- Long polling - Comet, HTTP server push, Reverse Ajax
- HTTP Streaming - Comet, chunked HTTP responses
- Bayeux protocol by Dojo Foundation – channels, publish/subscribe model
- Extensible Messaging and Presence Protocol (XMPP) over Bidirectional-streams Over Synchronous HTTP (BOSH)
- Drawbacks – buffering the streamed response, HTTP request and response headers on each message, two connections – coordination overhead and increased complexity that does not scale. HTTP wasn't designed for real-time, full-duplex comm.

Asynchronous JS and XML (AJAX) over HTTP



Why Not HTTP - HTTP Request Structure

GET */context/Servlet* **HTTP/1.1**

Host: *Client_Host_Name*

Header2: *Header2_Data*

...

HeaderN: *HeaderN_Data*

<Празен ред>

POST */context/Servlet* **HTTP/1.1**

Host: *Client_Host_Name*

Header2: *Header2_Data*

...

HeaderN: *HeaderN_Data*

<Празен ред>

POST_Data

Why Not HTTP - HTTP Response Structure

HTTP/1.1 200 OK

Content-Type: application/json

Header2: *Header2_Data*

...

HeaderN: *HeaderN_Data*

<Празен ред>

```
[{  "id":1,
    "name":"Novelties in Java EE 7 ...",
    "description":"The presentation is ...",
    "created":"2014-05-10T12:37:59",
    "modified":"2014-05-10T13:50:02",
},
{  "id":2,
    "name":"Mobile Apps with HTML5 ...",
    "description":"Building Mobile ...",
    "created":"2014-05-10T12:40:01",
    "modified":"2014-05-10T12:40:01",
}]
```

Sample HTTP Request Headrs – Much Overhead

GET JSON | 304 Not Modified | en.wikipedia.org | 28.3 KB | 91.198.174.192:80 | 90ms

Headers | Response | HTML | Cache | Cookies

Response Headers view source

Accept-Ranges	bytes
Age	33614
Cache-Control	private, s-maxage=0, max-age=0, must-revalidate
Connection	keep-alive
Content-Encoding	gzip
Content-Language	en
Content-Type	text/html; charset=UTF-8
Date	Fri, 13 Jun 2014 08:14:24 GMT
Last-Modified	Thu, 12 Jun 2014 22:54:07 GMT
Server	Apache
Vary	Accept-Encoding,Cookie
Via	1.1 varnish, 1.1 varnish, 1.1 varnish
X-Cache	cp1053 hit (10), amssq52 hit (95), amssq56 frontend miss (0)
X-Content-Type-Options	nosniff
X-Varnish	2755970074 2722575545, 1855856392 1851900024, 1030168116
x-ua-compatible	IE=Edge

Request Headers view source

Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding	gzip, deflate
Accept-Language	en,bg;q=0.7,en-us;q=0.3
Cache-Control	max-age=0
Connection	keep-alive
Cookie	centralnotice_bannercount_fr12=3; centralnotice_bannercount_fr12-wait=3%7C1401359244420%7C0; enwiki-gettingStartedUserId=1CkDb9sdZ2Cjzf4n1mGNHWuTY6WGxVvE; GeoIP=BG:Sofia:42.6833:23.3167:v4; uls-previous-languages=%5B%22en%22%5D; mediaWiki.user.sessionId=nQViPsuMNGYnKGD3BcfCMtUP7i28jlm3; mw_hidetoc=1; centralnotice_bucket=1-4.2
Host	en.wikipedia.org
If-Modified-Since	Thu, 12 Jun 2014 22:54:07 GMT
Referer	http://www.google.bg/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0CCcQFjAB&url=http%3A%2F%2Fen.wikipedia.org%2Fwiki%2FJSON&ei=4ayaU6rmK8fB0QWD4YGoAg&usg=AFQjCNHfk8CeJn25-S_gvF4dnY6ZaKxg4g&sig2=z2kN3sEMPh_SeBz8cKK-gw&bvm=bv.68911936,d.d2k
User-Agent	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:29.0) Gecko/20100101 Firefox/29.0

Response Headers From Cache

Accept-Ranges bytes

IETF WebSocket protocol (RFC 6455) (1)

- Official IETF standard - RFC 6455
- TCP-based full-duplex protocol
- Starts as standard HTTP /HTTPS connection to web server port **80/443** (**handshake phase**) – easy firewall and proxy traversal without the overhead connected with polling
- Uses HTTP protocol upgrade mechanism (Upgrade: websocket + Connection: Upgrade) – communication is immediately upgraded to more efficient WebSocket protocol (**data transfer phase**)
- Allows bidirectional streaming of data (partial messages)

IETF WebSocket protocol (RFC 6455) (2)

- Designed with security and extensibility in mind: Origin validation, sub-protocols & extensions negotiation (through standardized HTTP headers exchanged in handshake phase)
- **WebSocket API in Web IDL** is being standardized by **W3C**
- Supported by latest versions of all major web browsers –

Implementation status

Protocol	Draft date	Internet Explorer	Firefox[17] (PC)	Firefox (Android)	Chrome (PC, Mobile)	Safari (Mac, iOS)	Opera (PC, Mobile)	Android Browser
hixie-75	February 4, 2010				4	5.0.0		
hixie-76 hybi-00	May 6, 2010 May 23, 2010		4.0 (disabled)		6	5.0.1	11.00 (disabled)	
7 hybi-07	April 22, 2011		6[18]1					
8 hybi-10	July 11, 2011		7[19]1	7	14[20]			
13 RFC 6455	December, 2011	10[21]	11	11	16[22]	6	12.10[23]	4.4

[1] Gecko-based browsers versions 6–10 implement the WebSocket object as "MozWebSocket",[24] requiring extra code to integrate with existing WebSocket-enabled code.

Source: Wikipedia, [http://en.wikipedia.org/wiki/WebSockets]

WebSocket Request Example

[Request URL: **ws://localhost:8080/ipt-present/ws**]:
GET /ipt-present/ws HTTP/1.1
Host: localhost:8080
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: 3RhAwlJCs7wbj3xUdeDTXA==
Sec-WebSocket-Protocol: epresentation, ipt_present
Sec-WebSocket-Version: 13
Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits, x-webkit-deflate-frame
Origin: http://localhost:8080

WebSocket Response Example

[Request URL: **ws://localhost:8080/ipt-present/ws**]:

HTTP/1.1 **101 Switching Protocols**

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: QIMZj0lbIv1TM+JMx/JsoSKwYb8=

Sec-WebSocket-Protocol: epresentation

Server: GlassFish Server Open Source Edition 4.0

X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition 4.0 Java/Oracle Corporation/1.7)

W3C JavaScript WebSocket API [Web IDL]

```
WebSocket WebSocket(
  in DOMString url,
  in optional DOMString protocols
);

OR

WebSocket WebSocket(
  in DOMString url,
  in optional DOMString[] protocols
);
```

Attributes

Attribute	Type	Description
`binaryType`	`DOMString`	A string indicating the type of binary data being transmitted by the connection. This should be either "blob" if DOM `Blob` objects are being used or "arraybuffer" if `ArrayBuffer` objects are being used.
`bufferedAmount`	`unsigned long`	The number of bytes of data that have been queued using calls to `send()` but not yet transmitted to the network. This value does not reset to zero when the connection is closed; if you keep calling `send()`, this will continue to climb. **Read only.**
`extensions`	`DOMString`	The extensions selected by the server. This is currently only the empty string or a list of extensions as negotiated by the connection.
`onclose`	`EventListener`	An event listener to be called when the WebSocket connection's `readyState` changes to `CLOSED`. The listener receives a `CloseEvent` named "close".
`onerror`	`EventListener`	An event listener to be called when an error occurs. This is a simple event named "error".
`onmessage`	`EventListener`	An event listener to be called when a message is received from the server. The listener receives a `MessageEvent` named "message".
`onopen`	`EventListener`	An event listener to be called when the WebSocket connection's `readyState` changes to `OPEN`; this indicates that the connection is ready to send and receive data. The event is a simple one with the name "open".
`protocol`	`DOMString`	A string indicating the name of the sub-protocol the server selected; this will be one of the strings specified in the `protocols` parameter when creating the WebSocket object.
`readyState`	`unsigned short`	The current state of the connection; this is one of the Ready state constants. **Read only.**
`url`	`DOMString`	The URL as resolved by the constructor. This is always an absolute URL. **Read only.**

Source: Mozilla Developer Network (MDN), [https://developer.mozilla.org/en-US/docs/WebSockets/Writing_WebSocket_client_applications]

W3C WebSocket API - Methods [Web IDL]

```
void send(
  in DOMString data
);

void send(
  in ArrayBuffer data
);

void send(
  in Blob data
);
```

```
void close(
  in optional unsigned short code,
  in optional DOMString reason
);
```

Constant	Value	Description
CONNECTING	0	Connection is not open yet
OPEN	1	Connection is open and ready to communicate
CLOSING	2	Connection is in the process of closing
CLOSED	3	Connection is closed / can not be opened

WebSocket JS API Example (1)

- Example 1:

```
connection = new WebSocket('ws://h2j.org/echo', ['soap', 'xmpp']);
```

- Example 2:

```
var rootWsUri = "ws://" + (document.location.hostname == "" ?
    "localhost" : document.location.hostname) + ":" +
    (document.location.port == "" ? "8080" :
            document.location.port) + "/ipt-present/ws";

var websocket = new WebSocket( rootWsUri );
```

WebSocket JS API Example (2)

```
websocket.onopen = function (event) {
    onOpen(event);
};
websocket.onmessage = function (event) {
    onMessage(event)
};
websocket.onerror = function (event) {
    onError(event)
};
```

WebSocket JS API Example (3)

```
function onMessage(evt) {
        var jso = JSON.parse(evt.data);
        switch(jso.type){
          case "login-resp":
            conversationId = jso.cid;
            $(".logged-name").html(" - " + userName);
            $("#button-login").hide();
            $("#button-logout").show();
            showToster(jso.data.message, "info");
            break;
          ( - continues in next slide - )
```

WebSocket JS API Example (4)

```
 case "logout-resp":
    conversationId = ""; userName = "";
    $(".logged-name").html("");
    $("#button-logout").hide();
    $("#button-login").show();
    showToster(jso.data.message, "info");
    break;
case "online-resp":
case "offline-resp":
    showToster(jso.data.message, "info");
    break;  ( - continues in next slide - )
```

WebSocket JS API Example (5)

```
            case "error-resp":
                showToster(jso.data.message, "error");
                break;
        }
        console.log(jso.data);
}
```

Java™ EE 7 WebSocket support: Java API for WebSocket (JSR-356)

- What about the server-side support?
- Now easier than ever – Java™ EE 7 added two new APIs:
  - JSR-356: Java API for WebSocket
  - JSR-353: Java API for JSON Processing
- The new APIs facilitate rapid developemnt of WebSocket client and server applications (endpoints):
  - programmatic – by extending standard class Endpoint;
  - using annotations – @ServerEndpoint, @ClientEndpoint, @OnOpen, @OnClose, @OnMessage, @OnError, @PathParam

Programmatic Implementation of WebSocket Server Endpoints

```
import javax.websocket.*;
import javax.websocket.server.*;
public class MyEchoEndpoint extends Endpoint {
    @Override
    public void onOpen(final javax.websocket.Session session,
            EndpointConfig config) {
        session.addMessageHandler(new MessageHandler.Whole<String>() {
            @Override
            public void onMessage(String message) {
                try {
                    session.getBasicRemote().sendText(message);
                } catch (IOException e) { }
            }
        });
    }
}
```

Implementing WebSocket Using Annotations (1)

```
@ServerEndpoint(value = "/ws",
            decoders = {PresentationMessageDecoder.class},
           encoders = {PresentationMessageEncoder.class})
public class IPTPresentationEndpoint {

private static Set<Session> sessions =
        Collections.newSetFromMap(
             new ConcurrentHashMap<Session, Boolean>());

   @OnOpen
   public void onOpen(Session session) {
      sessions.add(session);
   }
```

Implementing WebSocket Using Annotations (2)

```
@OnClose
public void onClose(Session session) {
    sessions.remove(session);
}
@OnMessage
public void onMessage(PresentationMessage message, Session session) {
    String name;
    try{
        switch (message.getType()) {
            case LOGIN_ACTION:
                name = message.getPayload().getString("name", "Anonimous");
                session.getUserProperties().put("name", name);
                 ...
```

Implementing WebSocket Using Annotations (2)

```
        session.getBasicRemote().sendObject(
            new PresentationMessage(LOGIN_RESPONSE, session.getId(),
                Json.createObjectBuilder().add("message", "You are logged as "
                    + name).build()));
        sendMessageToOthers(session, USER_ONLINE_RESPONSE, name
                + " is now online.");  //send notification to all users
        break;
    case LOGOUT_ACTION:
        name = (String) (session.getUserProperties().get("name"));
        session.getBasicRemote().sendObject(
            new PresentationMessage(LOGOUT_RESPONSE, session.getId(),
                Json.createObjectBuilder().add("message", " ...").build()));
        sendMessageToOthers(session, USER_OFFLINE_RESPONSE, name +
            " is offline.");  //send notification to all users that current user is offline
        break;
```

Implementing WebSocket Using Annotations (3)

```
    } catch (EncodeException | IOException e ){
        try {
            session.getBasicRemote().sendObject(
                    new PresentationMessage(ERROR_RESPONSE, session.getId(),
                            Json.createObjectBuilder().add("message",
                                                    e.getMessage()).build()));
          Logger.getLogger(IPTPresentationEndpoint.class.getName())
                    .log(Level.SEVERE, null, e);
        } catch (IOException ex) {
            Logger.getLogger(IPTPresentationEndpoint.class.getName())
                    .log(Level.SEVERE, null, ex);
        } catch (EncodeException ex) { ... }
    }
}
```

Implementing WebSocket Using Annotations (4)

```
    private void sendMessageToOthers(Session currentSession, String
        messageType, String message) throws EncodeException, IOException {
        for(Session s: sessions){
            if(!s.getId().equals(currentSession.getId())){
                s.getBasicRemote().sendObject(
                    new PresentationMessage(messageType, s.getId(),
                        Json.createObjectBuilder().add("message", message).build()));
            }
        }
    }
}
```

- Alternative approach – use **session.getOpenSessions()** to access all sessions to the same endpoint

Java API for WebSocket Details (1)

- Types of messages:
  - Text (String, Reader, custom Object decoded using provided Decoder.Text or Decoder.TextStream)
  - Binary (ByteBuffer, byte[], InputStream, custom Object decoded using provided Decoder.Binarry or Decoder.BinarryStream)
  - Ping – echo response handled automatically, no custom handling needed
  - Pong (PongMessage)
- Main annotations: @ServerEndpoint, @ClientEndpoint, @OnOpen, @OnClose, @OnMessage, @OnError, @PathParam

Java API for WebSocket Details – Example (1)

```
@ServerEndpoint("/topic/{id}")
public class MyEchoEndpoint {
    @OnMessage
    public void textMessage(@PathParam ("id") String id, Session
        session, String message) {
        System.out.println("Topic " + id + " - Text message: " + message);
    }
    @OnMessage
    public void binaryMessage(@PathParam ("id") String id, Session
        session, ByteBuffer message) {
        System.out.println("Topic " + id + " - Binary message: " +
            message.toString());
    }
```

Java API for WebSocket Details – Example (2)

```
    @OnMessage
    public void pongMessage(@PathParam ("id") String id, Session
        session, PongMessage message) {
        System.out.println("Topic " + id + " - Pong message: "
                + message.getApplicationData().toString());
    }
}
```

Asynchronous processing of messages - **RemoteEndpoint.Async**

- void setSendTimeout(long timeoutmillis)
- sendText(String text, SendHandler handler)
- Future<Void> sendText(String text)
- void sendBinary(ByteBuffer data, SendHandler handler)
- Future<Void> sendBinary(ByteBuffer data)
- void sendObject(Object data, SendHandler handler)
- Future<Void> sendObject(Object data)

Java API for WebSocket (JSR-356) & Java API for JSON Processing (JSR-353)

- Using custom sub-protocols and message types
- Customization of handshake – sub-protocols, extensions, Origin validation, controlling initialization of endpoint instances
- Building custom Encoder/Decoder classes for easy marshalling/unmarshalling of arbitrary Java objects
- Integration with Java API for JSON Processing (JSR-353) for JavaScript™ friendly JSON serialization of data -client / server
- Building, navigating and streaming JSON object models
- Using pull parsers and generators for efficient JSON data processing

Example: Using Custom Decoder (1)

```
public class PresentationMessageDecoder implements
Decoder.Text<PresentationMessage> {
    @Override
    public PresentationMessage decode(String jsonData) throws
DecodeException {
        JsonObject obj;
        PresentationMessage message = null;
        try (JsonReader jsonReader = Json.createReader(
            new StringReader(jsonData))) {
           obj = jsonReader.readObject();
           if (obj.containsKey("type") && obj.containsKey("sid") &&
               obj.containsKey("data")) {
                    ( - continues in next slide - )
```

Example: Using Custom Decoder (2)

```
            message = new PresentationMessage(
                    obj.getString("type"), obj.getString("sid"),
                    obj.getJsonObject("data"));
        } else {
            throw new DecodeException(jsonData, "Invalid json string");
        }
    }
    return message;
}
```

(- continues in next slide -)

Example: Using Custom Decoder (3)

```
@Override
public boolean willDecode(String jsonData) {
    JsonObject obj;
    try (JsonReader jsonReader = Json.createReader(new
      StringReader(jsonData))) {
       obj = jsonReader.readObject();
       if (obj.containsKey("type") && obj.getString("type").length() > 0) {
          String type = obj.getString("type");
          switch (type) {
             case LOGIN_ACTION:
             Case  … :
                  return true;
          } } }    return false;
}
```

Example: Using Custom Encoder (1)

```
public class PresentationMessageEncoder implements
                                        Encoder.Text<PresentationMessage> {
    @Override
    public String encode(PresentationMessage message) throws
EncodeException {
        JsonObject obj = message.getPayload();
        JsonObject response = Json.createObjectBuilder()
            .add("type", message.getType())
            .add("sid", message.getSessionId())
            .add("data", obj).build();
        return response.toString();
    }
...
}
```

References

- Internet Engineering Task Force (IETF) WebSocket Protocol Specification – http://tools.ietf.org/html/rfc6455
- W3C The Web Sockets API – http://www.w3.org/TR/2009/WD-websockets-20091029/
- Mozilla Developer Network (MDN) – https://developer.mozilla.org/en-US/docs/WebSockets/Writing_WebSocket_client_applications
- Internet of Things (IoT) in Wikipedia – http://en.wikipedia.org/wiki/Internet_of_Things
- JSR 356: JavaTM API for WebSocket – https://jcp.org/en/jsr/detail?id=356
- JSR 353: Java API for JSON Processing - Reference Implementation – https://jsonp.java.net/

Thanks for Your Attention!

Questions?