SOAP and/or REST and/or WebSockets

BOF6984 : Harold Carr

Outline

1. Intro
2. top-level use-cases
3. WebSockets
4. SOAP and WebSockets
5. REST and WebSockets

Topic

1. Intro
2. top-level use-cases
3. WebSockets
4. SOAP and WebSockets
5. REST and WebSockets

BOF6984 : SOAP and/or REST and/or WebSockets

- Harold Carr
  - architect of SOAP Web Services Technology at Oracle

disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

JavaOne / Oracle

other WebSockets presentations

- BOF11880 - Server-Sent Events, Async Servlet, WebSocket, and JSON: Born to Work Together
- CON7365 - Extending Application Protocols and Java APIs over the Web with HTML5 WebSocket
- HOL4461 - Developing JAX-RS Web Applications Utilizing Server-Sent Events and WebSocket
- BOF6336 - Data Grids and WebSocket: Delivering Real-Time Push at Scale
- BOF7007 - Remoting Retrospective: DCE, COM, CORBA, RMI, RMI/IIOP, SOAP, REST, WebSockets

other WebSockets presentations

- CON3852 - WebSocket and Java EE: A State of the Union
- CON7001 - HTML5 WebSocket and Java
- CON8074 - HTTP, WebSocket, and SPDY: Evolution of Web Protocols
- TUT6361 - Building WebSocket Applications with GlassFish and Grizzly
- CON7042 - Building HTML5 Web Applications with Avatar

other SOAP presentations

- BOF4309 - What's Hot in Metro Web Services
- BOF6984 - SOAP and/or REST and/or WebSockets
- BOF4315 - JAX-WS.Next: Progress and Feedback
- BOF6649 - What's New in the Java Transaction API for Java EE 7 and Beyond
- BOF7007 - Remoting Retrospective: DCE, COM, CORBA, RMI, RMI/IIOP, SOAP, REST, WebSockets

other REST presentations

- CON7573 - How RESTful Is Your REST?
- CON3383 - Effective HATEOAS with JAX-RS
- CON4244 - JAX-RS-ME: A New JSR for RESTful Web Clients on Java ME 7
- CON6946 - Building a Web API Platform with Open Source oAuth 2.0, REST, and NoSQL
- BOF6984 - SOAP and/or REST and/or WebSockets
- CON12853 - I Got 99 Problems, but REST Ain't One
- TUT4673 - Beautiful REST + JSON APIs with JAX-RS and Jersey

other REST presentations

- CON4435 - JAX-RS 2.0: New and Noteworthy in the RESTful Web Services API
- CON4804 - Designing a Heavily Loaded REST Application with Java, Take 2
- BOF7007 - Remoting Retrospective: DCE, COM, CORBA, RMI, RMI/IIOP, SOAP, REST, WebSockets
- CON3566 - JSR 353: Java API for JSON Processing
- CON7114 - Pimp My RESTful Java Applications

discussion points

- When to use which?
- Should SOAP or REST use WebSockets as a transport?
- Should SOAP or REST be used in a complementary way where request/response is handled with SOAP or REST while push/pubsub is provided by WebSockets?
- Can REST and WebSockets coexists when REST is stateless (scalability via caching), whereas WebSockets are stateful (scalability via load-balancing)?
- Can WebSockets provide the ability to do efficient binary encodings of SOAP and/or REST?

discussion points

- Are WebSocket apps creating siloed applications instead of REST-like addressable, resources?
- If you do REST over WebSockets, how do you represent HTTP methods, URIs, URI parameters, headers?
- If you did SOAP over WebSockets, would you use binary XML encoding?
- Perhaps SOAP and REST would not leverage full capabilities of WebSockets since you don't expect endpoints or resources to initiate conversations.

discussion points

- Does it make sense to talk about SOAP, REST and WebSockets when:
  - SOAP is
    - stateless (unless doing SecureConversation ReliableMessaging or AtomicTransactions)
    - request/response or request-only
    - async (e.g, ReplyTo=NonAnon)
  - REST is
    - stateless
    - sync : request/response.

discussion points

- WebSockets is like conversation between two people.
  - full duplex: both sides can talk at the same time
  - both sides have to listen at same time, while talking
- Are SOAP and/or REST for
  - interoperable services
- whereas WebSockets for
  - private interactions?

Topic

1. Intro
2. top-level use-cases
3. WebSockets
4. SOAP and WebSockets
5. REST and WebSockets

top-level use-cases

SOAP

- enterprise
- service description
- static typing
- end-to-end security
- third-party authentication
- interoperability

REST

- public APIs
- small footprint
- loose-coupling
- interoperability

top-level use-cases

WebSockets

- browsers as universal clients for event oriented apps (full duplex low latency)
  - multiplayer games, chat, live tickers (e.g., sport/stock), social streams
- service push; pub/sub
  - alternative to AJAX, Comet, long polling, HTML5/SSE, ...
- async, frame-oriented and state-oriented app protocols layered on top
  - e.g. AMQP, XMPP, JMS, STOMP, VNC (RFB),
- private subprotocols (e.g., small, binary)
  - potential performance improvement
- tight-coupling

Topic

1. Intro
2. top-level use-cases
3. WebSockets
4. SOAP and WebSockets
5. REST and WebSockets

Motivation: high penetration / server push

- faking HTTP request to establish a (SSL/TLS) connection
  - then tunnel through it passing through HTTP proxies and firewalls
- including ENTERPRISE proxies ones that only let port 80 and 443 open
- bi-directional connection enables server push
- WebSockets is not something "more" - it's just TCP

timeline

- 1994 Web
  - HTTP: client request/service response
  - then nothing until next click
- 2005 AJAX
  - “feel” more dynamic
  - still all comm initiated by client
    - requires user interaction or periodic polling

timeline

- 2006 Comet, long-polling, HTTP streaming ("reverse AJAX")
  - illusion of server initiated connection
  - client opens connection, server keeps it open until event/response
  - HTTP overhead still a problem
    - not suited for low latency apps
    - e.g., multiplayer first person shooter games, collaboration
- 2010/2012 WebSockets
  - persistent client/server connection
  - both can send data at any time
  - cross origin comm supported (you'd better trust parties)

timeline details

- HTTP
  - strict request-response model
  - client sends request, waits until response received
- AJAX/XMLHttpRequest
  - pulls data "requests" from server
  - XMLHttpRequest just does HTTP request in background (no UI blockage) with callback handling response
  - polling: apps periodically do AJAX request to ask if server has an "event"
  - event latency depends on polling period
    - can increase polling rate
    - but wastes resources and scales poorly (most polls return empty)

timeline details

- COMET/LONG-POLLING ("reverse AJAX")
  - client does request; service maintains connection
  - fakes notifications by sending "event"/response when ready
  - then client send request again, and wait for next event
  - always a pending request
- HTTP STREAMING
  - service keeps response message open
  - response body never closed after sending event
  - new events written to same stream
  - response message body represents a unidirectional event stream to the client
  - requires proxies to forward "partial" responses

timeline details

- HTML5 SERVER-SENT EVENTS (SSE)
  - standardizes COMET
  - client API to open HTTP conn for receiving push notification
  - `EventSource` represents client endpoint to receive events
  - mime `text/event-stream` for event framing format
  - `EventSource` impl can do connection management to share connections
    - period reconnections under the covers
  - service should send periodic "keep-alive" comments
    - to avoid proxies killing the connection

client SSE example

```
var source = new EventSource('Events');
source.onmessage = function (event) { ...  }
```

Comparison between WebSockets and Comet

- major difference is in the upstream link
- Comet downstream link is efficient/reliable
- but real streaming upstream not possible with HTTP
- most realtime Web app, downstream more important
- but collaborative apps need good upstream link

WebSockets properties

- server can send info to client anytime: update latency reduced
- server handles fewer connection requests
  - some requests transformed into updates to WebSockets
  - but then connection management
- client heartbeats be required so server can cleanup dead connections
- service heartbeat not required
  - service pushes shows it is alive
  - but might need to reset connection if no updates

WebSockets properties

- support WebSockets but fall back to HTTP Streaming and HTTP Long Polling dynamically
- no connection reliability
  - no reconnect handling
  - no guaranteed message delivery (like Server-Sent Event does)
  - reliability has to be in app (same as “keep alive” message in SSE)
  - sharing WebSocket between different pages must lock writes/reads
- no same origin policy limitation
- enabling siloed apps

client WebSockets example

```
var ws = new WebSocket('ws://localhost:8876/Channel',
                       'mySubprotocol.example.org');
ws.onmessage = function (message) {
    var messages = document.getElementById('messages');
    messages.innerHTML += "<br>[in] " + message.data;
};

sendmsg = function() {
    var message = document.getElementById('send').value
    document.getElementById('message_to_send').value = ''
    ws.send(message);
    var messages = document.getElementById('messages');
    messages.innerHTML += "<br>[out] " + message;
}
```

WebSockets negatives

- introduces new failure modes
- needs another protocol layered on top
- WISH: web socket identify protocol in use
  - so if intermediary understands, it can get involved
- little potential for protocol evolution
  - the client JavaScript and server implementation have to match
  - no means to negotiate protocol

WebSockets negatives

- app protocol between app and server . . . just like TCP sockets
- no TCP because
  - server firewall won't allow connections to ports other than 80 and 443
  - or the client firewall
- So WebSockets is generic proxy/tunnel
- reinventing: all we already have for TCP using WebSockets
- need new generation of firewalls that can inspect WebSockets traffic
- all this effort to avoid configuring firewalls to **allow** connections to ports other than 80 and 443?

Topic

1. Intro
2. top-level use-cases
3. WebSockets
4. SOAP and WebSockets
5. REST and WebSockets

SOAP/WebSocket potential benefits

- can tunnel out of a firewall
  - use instead of WS-MakeConnection
- connection-based
  - use instead of WS-ReliableMessaging
- multi-message communication

MS-SWSB : SOAP Over WebSocket Protocol Binding Specification

```
<wsdl:definitions ...> ...
 <wsdl:binding name="MyBinding" type="MyPortType"> ...
  <soap12:binding
   transport="http://schemas.microsoft.com/soap/websocket
   <wsdl:operation name="MyOp"> ... </wsdl:operation>
 </wsdl:binding>
 <wsdl:service name="MyService">
  <wsdl:port name="MyPort" binding="MyBinding">
   <soap12:address location=" ws://myHost/myService/" />
  </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

SOAP/WebSocket protocol example

```
GET http://myHost/myService HTTP/1.1
Connection: Upgrade,Keep-Alive
Upgrade: websocket
Sec-WebSocket-Key: ROOw9dYOJkStW2nx5r1k9w==
Sec-WebSocket-Version: 13
Sec-WebSocket-Protocol: soap
soap-content-type: application/soap+msbinsession1
microsoft-binary-transfer-mode: Buffered
Accept-Encoding: gzip, deflate
```

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: soap
```

SOAP easily done on WebSockets

- WSDL enables
  - different transports
  - different address representations
- SOAP/XML messages are self-contained
  - for interop : avoid binary reps
- main work
  - WebSocket connection management
  - message dispatching (i.e., no servlet)?

Topic

1. Intro
2. top-level use-cases
3. WebSockets
4. SOAP and WebSockets
5. REST and WebSockets

REST/WebSockets potential benefits

- multi-message communication
- existing apps use REST/WebSockets lib
  - reduced overhead of single connection without refactoring code
- Transactional-REST
  - beginTransaction; REST/WebSocket calls, . . . ; endTransaction;
  - if connection drops before commit then rollback TX

REST/WebSocket naysayers

- you don't expect resources to initiate conversations
  - counter: resource send change notification
- WebSocket "resources" not addressable
  - since private subprotocol
- Glenn Block
  - "Would it be viable to have a REST resources that can communicate changes (events) through web sockets?"
  -
  - Roy Fielding : "It would be a different style of interaction. Generally speaking, REST is designed to avoid tying a server's connection-level resources to a single client using an opaque protocol that is indistinguishable from a denial of service attack. Go figure." - July 8, 2010

REST/WebSocket naysayers

- WebSockets : two-way socket connection : not an app protocol
  - rolled back 20 years of protocol consolidation
  - now back to a free-for-all of everybody's pet protocol
    - having every system roll their own protocol is retrograde
  - OK if client/server are tightly coupled, non-reusable UI app
  - BAD if service is supposed to be reusable by unknown heterogeneous clients
    - have to negotiate media type AND application protocol too

WebSocket apps are not providing addressable, reusable resources to world.

Web Sockets break REST constraints (compared to polling)

- Stateless
  - connection management needed on client and service side
  - servers use more memory
  - single server going down has more impact
    - but client may be able to reestablish connection
- Cache
  - can't share updates between different clients
    - cache with standard protocols visible to intermediaries, so can participate in notification distribution networks

Web Sockets break REST constraints (compared to polling)

- Uniform Contract
  - no uniform contract
  - intermediaries do not know what is going on, so can't help/hinder
  - firewalls can't enforce policy decisions
- Layered System
  - proxies/gateways cannot be transparently inserted
  - instead direct connection to server itself
  - potentially negative security and performance impacts

REST/WebSockets representations

```
{
    "identity" : uuid_generated_by_the_server,
    "requests" : [{
            "uuid" : 0,
            "method" : "POST",
            "path" : "/any_url/",
            "headers" : [{
                    "name" : "Content-Type",
                    "value" : "test/plain"
            }],
            "queryStrings" : [{
                    "name" : "foo2",
                    "value" : "bar2"
            }],
            "messageBody" : "Swagger Socket Protocol"
```