Realtime Web Anwendungen mit APEX

Johannes Mangold
Trivadis AG
Basel

Schlüsselworte
APEX, HTML5 WebSockets, Realtime Web, Living Web, JMS, Advanced Queuing

Einleitung

APEX, das Rapid Web Application Entwicklungs-Framework der Oracle Datenbank bringt im aktuellen und auch zukünftigen Release zahlreiche Neuerungen mit, die dieses Framework auch weiterhin für moderne Webanwendungen im Geschäftsumfeld attraktiv machen. Im aktuellen Release 4.1 werden auch erstmals Optimierungen für die Browser von mobilen Endgeräten angeboten. Da moderne Web Anwendungen zunehmend auch Echtzeitverhalten und Interaktivität fordern, wäre es wünschenswert, diesen Anforderungen auch mit APEX gerecht werden zu können. Der Vortrag soll eine Möglichkeit aufzeigen, wie man mit APEX neben dem üblichen Request-Response Paradigma auch asynchrone Full-Duplex Kommunikation zwischen Backend und Frontend realisieren kann, um den Web Anwendungen Realtime-Verhalten einzuhauchen.

Warum Realtime in Webanwendungen

Das Web wächst unaufhaltsam zu einer interaktiven Anwendungsplattform heran. Facebook, Twitter, Google – um nur einige prominente Anbieter zu erwähnen – stellen ihren Anwendern Webanwendungen zur Verfügung, die teilweise zum festen Bestandteil der täglichen Internetnutzung geworden sind. Bei der Nutzung dieser Webanwendungen ist einem aufgrund der gebotenen User Experience kaum noch bewusst, dass sie in einer Browserumgebung ablaufen und über die Standard Web-Architektur kommunizieren. Willkommen „Living Web".

Wir können davon ausgehen, dass Anwender den Komfort und die User Experience, die sie in den täglich genutzten interaktiven und dynamischen Webanwendungen wie Google Mail oder Facebook erfahren auch in den Unternehmens-Webanwendungen fordern werden. Durch die starke Verbreitung von vielfältigen Internetfähigen mobilen Endgeräten wie Smartphones und Tablets im Privatbereich wird auch der Bedarf an mobilem Zugriff auf die Unternehmensanwendungen und Geschäftsprozesse stark ansteigen. Dies stellt Unternehmen vor verschiedene Herausforderungen.

Ursprünglich für den Austausch von statischen Dokumenten konzipiert, basiert die Webarchitektur seit der Entstehung bis heute prinzipiell auf einem zustandslosen Request-Response Paradigma und bietet somit keine ideale Basis für interaktive Anwendungen. Interaktivität und Dynamik in Webanwendungen wird bislang durch Workarounds erreicht, welche die Defizite der zustandslosen Request-Response Architektur ausgleichen. Als Hilfsmittel werden oft komplett eigene Laufzeitumgebungen im Browser über Plugins wie zum Beispiel Flash oder Silverlight eingebettet. Realtime-fähigkeit wird in modernen Webanwendungen durch Workarounds wie Ajax Polling oder Long Polling simuliert. Der Preis für diese Workarounds sind zum Teil sehr hohe Komplexität in der Entwicklung auf Client- und Serverseite, sowie schlechte Skalierbarkeit aufgrund großem Overhead und hohem Lastaufkommen auf Serverseite. Mit stetig steigender Anzahl der Webclients und nicht zuletzt durch den starken Zuwachs an internetfähigen mobilen Endgeräten wird diese Tatsache zu einem großen Problem für die Entwicklung und den Betrieb von modernen und lebendigen Webanwendungen.

Die Defizite und Probleme wurden zum einen von der WHATWG und später auch dem W3C Konsortium bereits 2004 in Angriff genommen. Die beiden Gremien haben sich zum Ziel gesetzt die

Standards des Webs so weiterzuentwickeln, dass das Web eine solide Basis für die nächste Generation von Webanwendungen bieten kann: HTML5.

Ein fast revolutionärer Teil der HTML5 Spezifikationsbereiche ermöglicht den Bruch des Request-Response Paradigmas durch das Einführen von WebSockets. Mit WebSockets ist es erstmals möglich auf Basis der vorhandenen Webinfrastruktur bidirektionale Full-Duplex Socketverbindungen zwischen Client (z.B. Internetbrowser) und einem Backendsystem herzustellen: TCP Sockets für das Web. Browser werden zu vollwertigen Netzwerkteilnehmern, die Realtime-fähigkeit eröffnet viele neue Möglichkeiten in der Realisierung nachrichtenbasierter Webarchitekturen und kann Webentwickler von Workarounds und Komplexität befreien.

HTML5 WebSockets
Die WebSockets Spezifikation des HTML5 Communikations Bereichs definiert eine einfache API um WebSocket Verbindungen aufzubauen und zu schließen, sowie um Nachrichten zu senden und zu empfangen. Außerdem gibt es verschiedene Events, die beim Öffnen der Verbindung, beim Eintreffen einer neuen Nachricht, bei einem Fehler oder beim Unterbruch oder Schließen der WebSocket verbindung publiziert und werden.

Auf Protokollebene sendet der Client beim Aufbau einer WebSocket Verbindung einen initialen HTTP Request mit einem Request-Header, der den Server um ein Upgrade auf eine permanente WebSocket Verbindung bittet. War der Handshake erfolgreich, wird die WebSocket Verbindung hergestellt.

Einmal aufgebaut, können über diese permanente Verbindung Client und Server im Full-Duplex Modus WebSocket Nachrichten versenden und empfangen. Der Overhead pro Nachricht ist dabei lediglich ein 2-4 Byte grosser Header-Frame. Der Header-Overhead eines HTTP Request-Response Roundtrips liegen im Vergleich leicht bei 1000 Byte, also ca. bei 200 - 500 mal mehr wie bei einer WebSocket Nachricht.

Auch in Bezug auf Latenz bringt WebSocket Kommunikation enorme Vorteile, da Nachrichten jederzeit in beide Richtungen gesendet und empfangen werden können. Bei Ajax Polling kann jeweils immer nur eine Seite eine Nachricht senden. Liegen auf dem Server zwei Nachrichten innerhalb 5 ms zum Versand an den Client bereit, gibt es bei einem angenommenen Polling-Intervall von 500ms für die Zweite Nachricht eine Latenz von mindestens 500ms im schlechtesten Fall fast 1000ms.

Durch die drastische Reduktion an Overhead und starker Verringerung der Latenz pro Nachricht lassen sich mit WebSockets hochskalierbare und Realtime-fähige Webanwendungen entwickeln, basierend auf einer asynchronen, nachrichtenbasierten Architektur.

WebSockets mit APEX
WebSockets basieren auf Web-Standards. Dasselbe gilt für APEX Anwendungen. Aus diesem Grund lässt sich die neue Technologie sehr einfach in eine APEX Anwendung integrieren.

Nahezu alle modernen Browser können bereits heute schon mit WebSockets umgehen (Für ältere Browser und Versionen ohne WebSocket Unterstützung gibt es je nach WebSocket Server transparente Fallbackstrategien, z.B. auf Flash Sockets oder Ajax Long Polling).

So kann in der APEX Anwendung einfach mittels JavaScript eine WebSocket Verbindung zu einem WebSocket-fähigen Server aufgebaut werden:

```
var wsUri = "ws://websocket.trivadis.com/";

function initWebSocket() {
        websocket = new WebSocket(wsUri);
        websocket.onopen = function(evt) { onOpen(evt) };
        websocket.onclose = function(evt) { onClose(evt) };
        websocket.onmessage = function(evt) { onMessage(evt) };
        websocket.onerror = function(evt) { onError(evt) };
}
```

Abb. 1: WebSockets JavaScript API

Das APEX Backend, bzw. der HTTP Listener der Oracle Datenbank oder der Apache Webserver unterstützen von Haus aus noch keine WebSockets. Deshalb wird hier noch ein WebSocket Gateway benötigt, welches WebSocket Verbindungen aufbauen und WebSocket Nachrichten empfangen und weitergeben kann. Es gibt mittlerweile schon eine gute Auswahl an WebSocket fähigen Serversystemen vor allem auch im OpenSource Bereich. Ein kommerzielles, sehr ausgereiftes und für den Unternehmenseinsatz entworfenes Produkt ist der Kaazing WebSocket Gateway.

Advanced Queuing als JMS Provider
Um den Kreis zu schließen wird noch ein Ziel benötigt, welches die Nachrichten vom Client weiterverarbeiten, sowie selbst Nachrichten asynchron an den Client versenden kann. Eine sehr interessante Option hierfür ist die Verwendung des in der Oracle Datenbank verfügbare Advanced Queuing als Message Broker. Advanced Queing kann als vollwertiger JMS Provider angesprochen werden und lässt sich so wunderbar als Messagingsystem vom Kaazing WebSocket Gateway JMS Edition anbinden. Verwendet man dann noch clientseitig die Kaazing JavaScript JMS Client Library, eröffnen sich spannende neue Möglichkeiten für das Design von datengetriebenen APEX Anwendungen besonders auch für mobile Clients. Per JavaScript kann die Anwendung dann via JMS asynchron nachrichtenorientiert und transaktional mit den AQ Queues und Topics nach dem Point-to-Point bzw. Publish/Subscribe Pattern kommunizieren. Ohne Workarounds, in Realtime.

So können beispielweise über JMS Messages über WebSockets PL/SQL Prozeduren aufgerufen werden. Ergebnisse von PL/SQL Prozeduren können vom Server aus asynchron zum Client gesendet werden. Auch ein interessanter Anwendungsfall wäre die Anwendung von Insert/Update/Delete Trigger um alle Clients in Echtzeit über Datenänderungen in Datenbanktabellen zu informieren.

Gesamtarchitektur

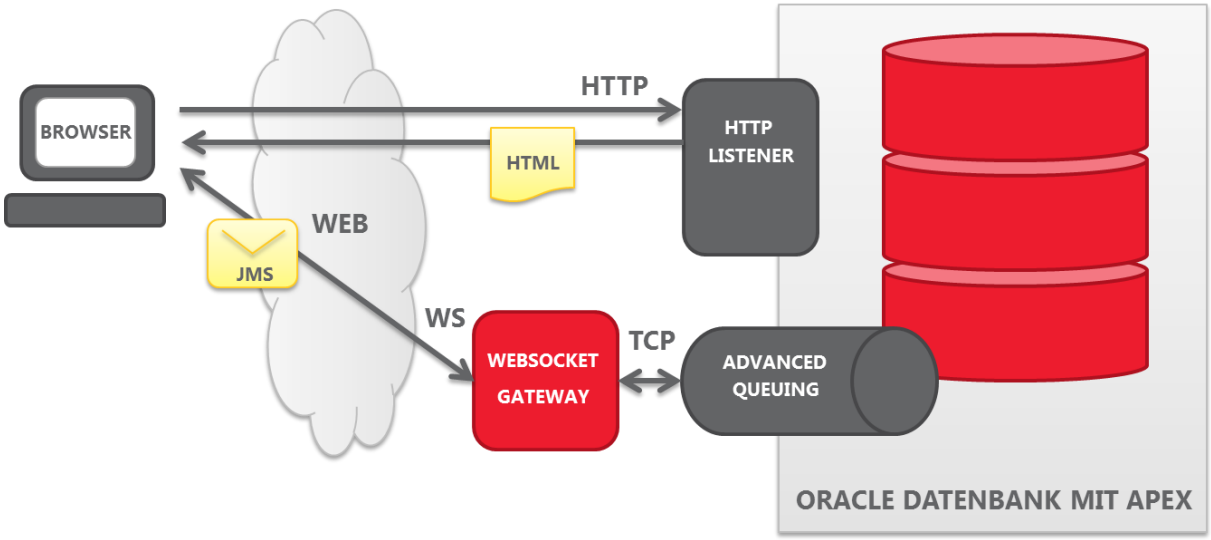

Abb. 2: Architektur APEX mit Realtime-fähigkeit

Live Demo
In einer interaktiven Live-Demonstration wird ein Showcase einer Realtime-fähige APEX Anwendung gezeigt, welche nach der oben beschriebenen Architektur entwickelt wurde.

Kontaktadresse:

Johannes Mangold

Senior Consultant
Application Development

Trivadis AG
Elisabethenanlage 9
CH-4051 Basel

Telefon:	+41-61-279 97 55
Fax:	+41-61-279 97 56
E-Mail	johannes.mangold@trivadis.com
Internet:	www.trivadis.com