PL/SQL-Logik Beine machen: mobil durch Webservices

Daniel Kampf, PITSS GmbH

Wer seine Business-Logik in der Datenbank organisiert hat, stellt sich häufig die Frage, wie diese für mobile Anwendungen genutzt werden kann. Der Artikel gibt einen Überblick über die Möglichkeiten der Webservice-Generierung zur Erschließung von PL/SQL-Logik.

Um das User Interface unabhängiger von Technologie-Wechseln zu machen, haben viele Unternehmen ihre Business-Logik in die Datenbank überführt. Damit dieser Code anderen Anwendungen zur Verfügung steht, eignen sich besonders Webservices als universelles Werkzeug. Sowohl Enterprise-Anwendungen als auch mobile Apps kommen als deren Konsumenten infrage. Dieser Artikel zeigt, wie PL/SQL von einem Webservice aus aufgerufen und dieser Service wiederum in den Oracle Mobile Cloud Service eingebunden wird.

Webservice ist nicht gleich Webservice

Zunächst ein Blick auf die verschiedenen Möglichkeiten zur Implementierung eines Webservice. Abhängig von der Ziel-Anwendung ist es wichtig, den richtigen Webservice zu wählen. Es gibt aktuell im Prinzip zwei verschiedene Typen von Webservices: den etwas in die Jahre gekommenen SOAP- und den RESTful-Webservice. SOAP zeichnet sich durch seine XML-Struktur aus und ist durch eine Vielzahl von Meta-Informationen schnell aufgebläht. Vorteil ist jedoch, dass der Verwender eines SOAP-Webservice über die WSDL-Seite gut einschätzen kann, in welcher Form die Daten vorliegen. Dies ist aber auch der größte Nachteil: Die Payload, also die Daten, die beim Aufruf eines Webservice übermittelt werden, ist recht groß.

Bei RESTful-Webservices ist dieses Problem nicht vorhanden – sie sind deutlich leichtgewichtiger. Hier ist die Beziehung zwischen Service-Anbieter und -Konsument sehr lose: Eine WSDL-Datei, die den Service und alle Methoden beschreibt, ist in Form einer WADL nur rudimentär vorhanden. Diese Eigenschaften machen den RESTful-Service zur ersten Wahl, wenn es um die Entwicklung von mobilen Lösungen geht, da er seine Vorteile bei der Leichtgewichtigkeit der Datenübertragung ausspielt.

Zugriff auf die Datenbank

Neben der Wahl des Webservice für die Anbindung der PL/SQL-Logik stellt sich auch die Frage, mit welchen Tools und auf welcher Plattform der Webservice erzeugt beziehungsweise betrieben werden soll. Oracle bietet zwei grundsätzliche

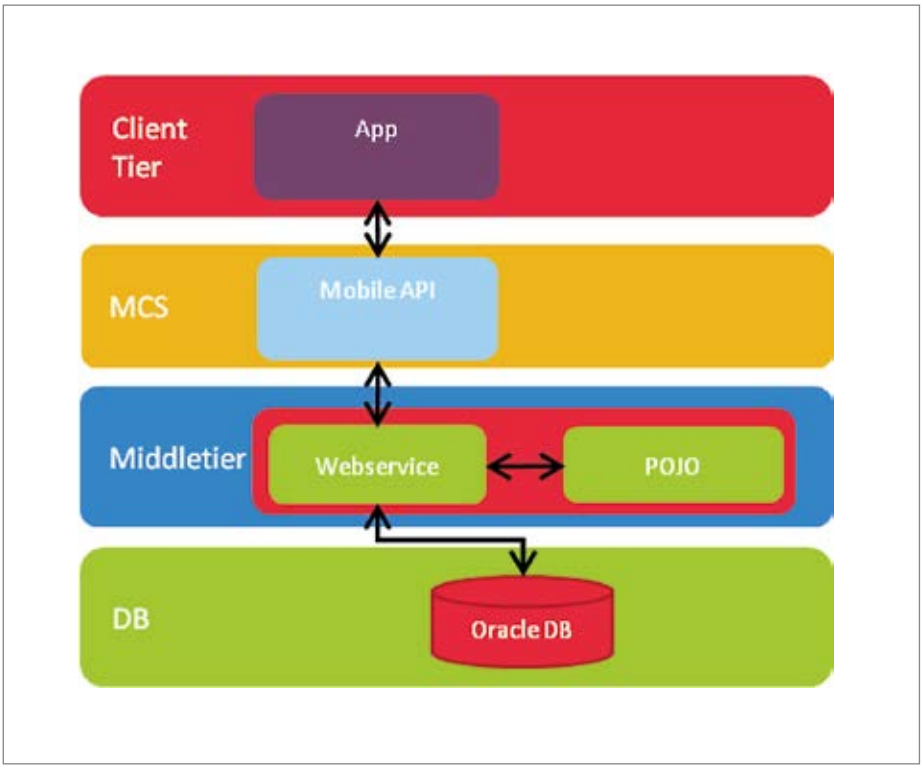

Abbildung 1: Szenario einer mobilen Lösung

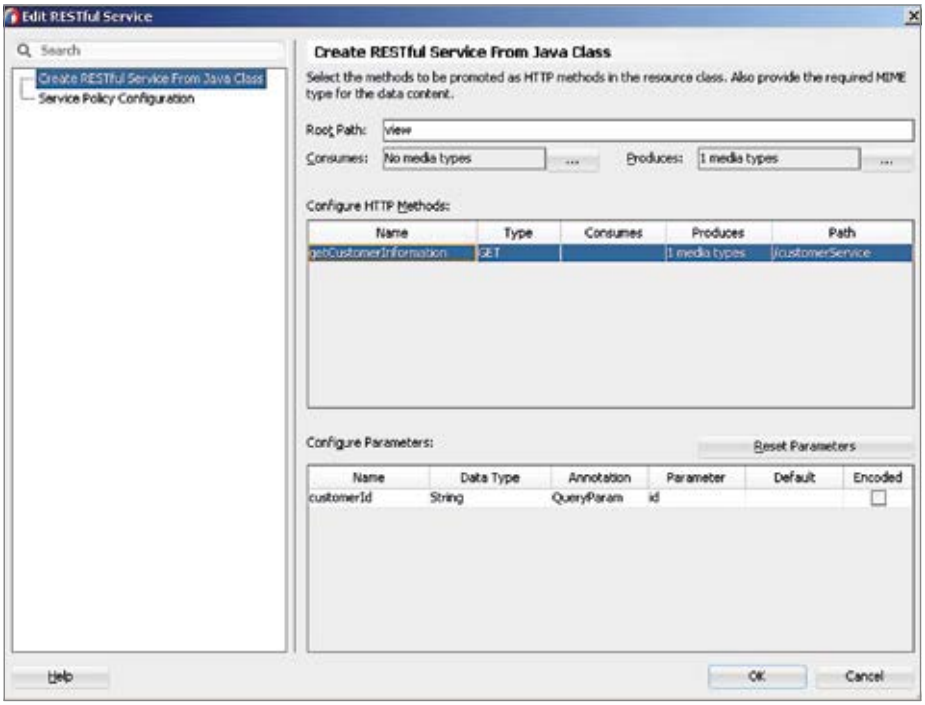

Abbildung 2: Optionen für die Webservice-Generierung

```
27 public class CustomerInformation {
28
29     @GET
30     @Produces("application/json")
31     @Path("/customer")
32     public Customer getCustomerInformation(@QueryParam("id") String customerId){
33
```

Abbildung 3: Annotationen im Java-Code

Möglichkeiten an. Auf der einen Seite lässt sich die Generierung mithilfe des JDeveloper durchführen. Die andere Alternative ist der Oracle REST Data Service (ORDS). Er unterstützt, wie der Name schon sagt, nur die schlankeren RESTful-Webservices, wohingegen im JDeveloper beide Varianten möglich sind. In diesem Artikel liegt der Fokus auf der Implementierung eines Java-Webservice, basierend auf JAX-RS und dem WebLogic-Server in der Version 12.2.1.0.0 als Plattform *(siehe Abbildung 1)*.

RESTful Java-Webservice mit JAX-RS

Alle modernen IDEs sind in der Lage, auf Basis einer Java-Klasse einen Webservice zu generieren – so auch der JDeveloper. Zunächst sind ein neuer Workspace und ein neues Java-Projekt erforderlich. In diesem Beispiel wurde danach im JDeveloper eine Java-Klasse mit dem Namen „CustomerInformation.java“ erstellt, die über den Connection-Pool des Middleware-Servers eine Datenbank-Verbindung zugewiesen bekommt und mit deren Hilfe das Datenbank-Package „pkg_customer“ aufruft *(siehe Abbildung 2)*. Die Prozedur „get_customer_information“ liefert Kundendaten anhand der Kunden-ID.

Daneben wurde ein Plain old Java Object (POJO) angelegt, um die Struktur der Daten vorzugeben. Mit diesen zwei simplen Klassen sind an und für sich schon alle Voraussetzungen gegeben, um einen Java-basierten RESTful-Webservice zu generieren. Mit einem Rechtsklick auf „CustomerInformation.java“ lässt sich der Kontextmenüeintrag „Create RESTful Service“ auswählen.

Es öffnet sich ein weiteres Fenster, in dem verschiedene Optionen ausgewählt werden können; unter anderem, welche HTTP-Operationen der Webservice bereitstellt (GET, POST, PUT und DELETE) und welche Form der MIME-Type für den Dateninhalt hat. In diesem Fall wurde die Option „application/json“ unter „Produces“ ausgewählt, um die Daten in Form von JSON zu strukturieren.

Im unteren Teil des Fensters lassen sich gegebenenfalls Parameter definieren. Es wurde ein Query-Parameter hinzugefügt, um die entsprechenden Informationen für die übergebene ID zurückgeben zu können. Wählt man „OK“ und schaut sich

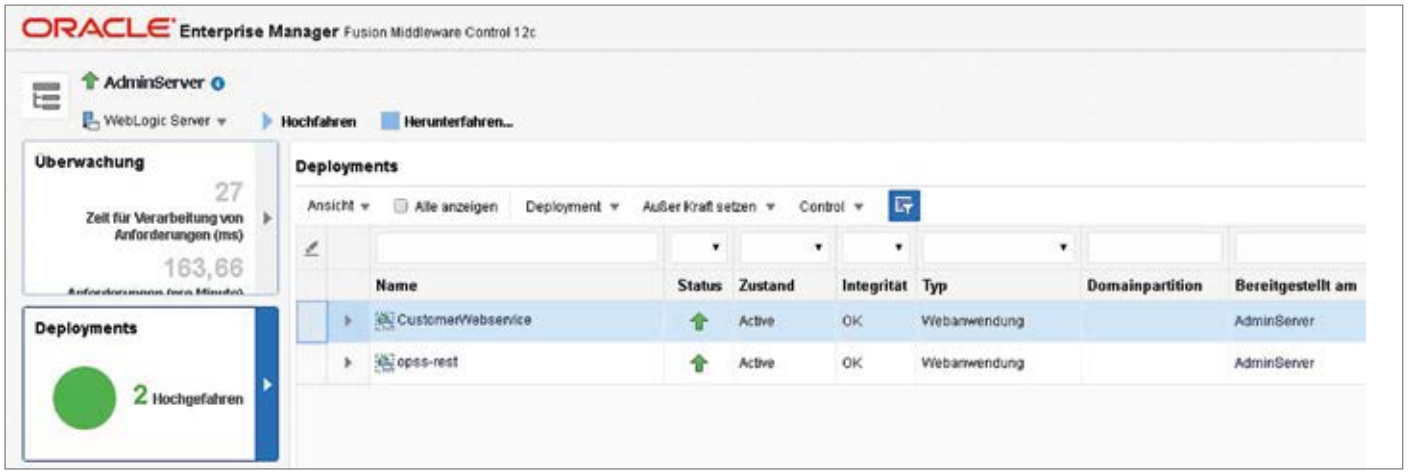

Abbildung 4: „CustomerWebservice“ auf WebLogic-Server eingerichtet

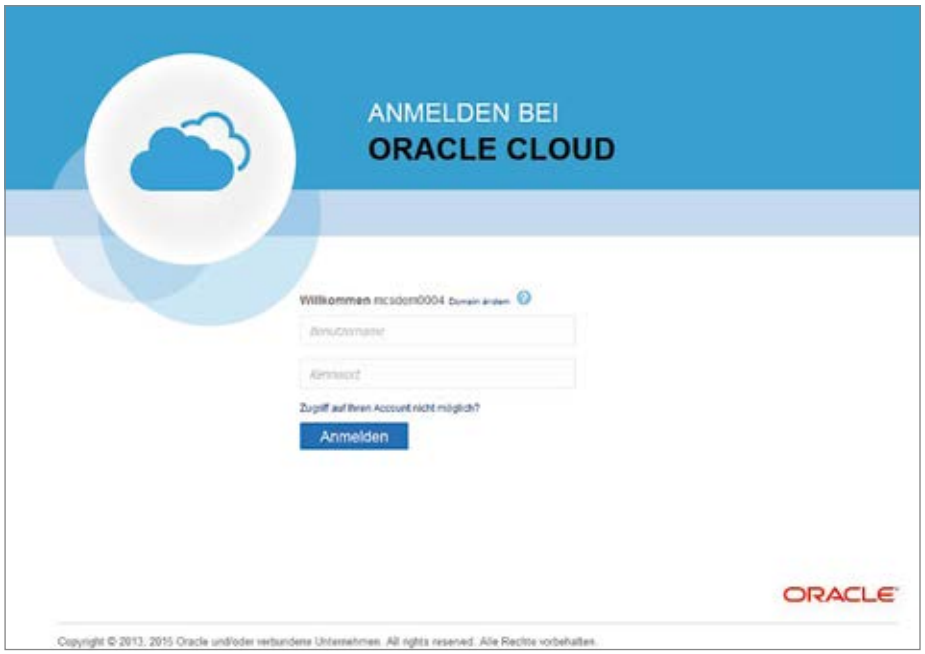

Abbildung 5: Login des Mobile Cloud Service

danach die Java-Klasse an, stellt man fest, dass JDeveloper automatisch alle notwendigen Annotationen in der Java-Klasse hinzugefügt hat *(siehe Abbildung 3)*.

Nach der Generierung des Webservice lässt sich dieser direkt aus dem JDeveloper heraus testen. Dafür kann mit einem erneuten Rechtsklick auf die Java-Klasse „CustomerInformation“ geklickt und „Test Webservice“ auswählt werden. Der Webservice wird dann auf dem internen WebLogic-Server eingerichtet und anschließend der HTTP-Tester des JDeveloper geöffnet *(siehe Abbildung 4)*. Im HTTP-Tester lassen sich unter „Parameter“ die Übergabeparameter einstellen und mit einem Klick auf „Send Request“ erfolgt ein Test des Webservice.

Ist der Test erfolgreich, liefert er den HTTP-Status „200“ und die Kunden-Informationen zurück. Jetzt kann der Webservice auf einem eigenständigen Webserver eingerichtet werden. In diesem Fall kommt der Oracle WebLogic Server in der Version 12.2.1.0.0 zum Einsatz. Dies kann direkt aus dem JDeveloper heraus geschehen, es muss hierfür lediglich ein Deployment-Profil angelegt und der entsprechende WebLogic-Server ausgewählt werden. Nachdem der Webservice auf dem Server eingerichtet worden ist, kann er von außen angesprochen werden, die PL/SQL-Logik ist jetzt erreichbar.

Zum Testen von Webservices gibt es neben dem HTTP-Tester des JDeveloper unzählige kostenfreie Werkzeuge. Exemplarisch sei hier die Browser-Erweiterung Postman für Google Chrome erwähnt.

Einbindung des Webservice Oracle MCS

Wenn der Test des Webservice erfolgreich war, lässt er sich in den Oracle Mobile Cloud Service einbinden. Hierfür muss man sich an der entsprechenden Domain des MCS mit den Zugangsdaten anmelden *(siehe Abbildung 5)*. Nach der Anmeldung wird ein Dashboard angezeigt, das dem Benutzer die abonnierten Cloud-Services anzeigt – mit einem Klick auf einen Service lassen sich Details wie der Service-Status einsehen.

Die Auswahl des assoziierten Service öffnet auf der nächsten Seite die Service-Konsole, in der die einzelnen Komponenten des MCS verwaltet werden *(siehe Abbildung 6)*. In der Übersicht wird unter „Erste Schritte“ die App-Entwicklung mithilfe des MCS in aller Kürze erläutert. Um den generierten Webservice für die App bereitzustellen, ist zunächst ein sogenanntes „Mobile Backend“ zu erzeugen. Es dient der Server-seitigen Verwaltung der App.

Erstellen eines Connectors

Nachdem das Mobile Backend angelegt ist, können wir einen Connector einrichten. Dieser bietet die Möglichkeit, Backend-Systeme einzubinden, so wie in dem hier vorliegenden Fall, oder aber auch externe Webservices wie Google Webservices. Ein Connector kann dann später in einem sogenannten „Custom-

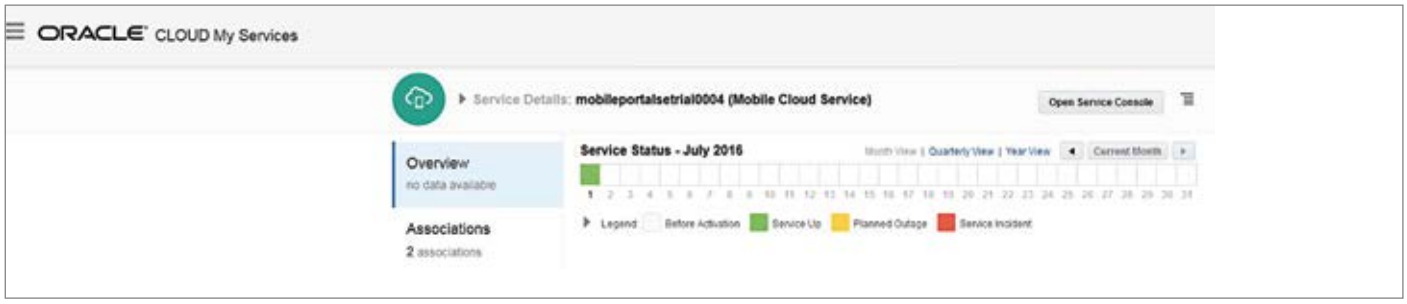

Abbildung 6: Service Console öffnen

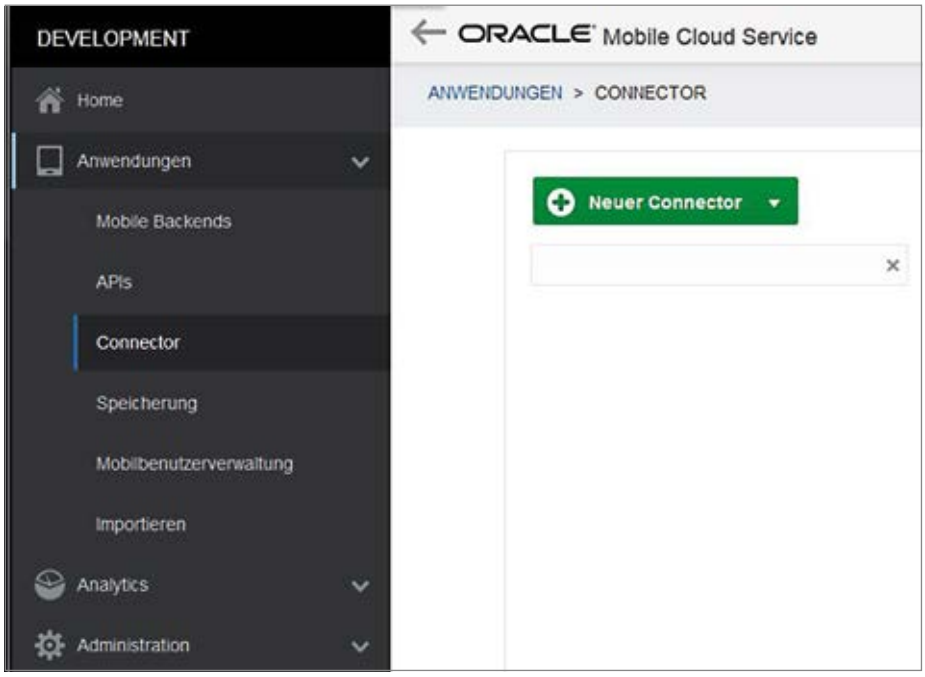

Abbildung 7: Erstellen eines neuen Connectors

API" verwendet werden. Ein Klick auf „Neuer Connector" zeigt eine Auswahl an möglichen Connectoren an: REST, SOAP und Integration Cloud Service.

Nach der Auswahl des zu erzeugenden Service wird eine Eingabemaske geöffnet. Hier müssen folgende Pflichtfelder gefüllt werden: API-Anzeigename, API-Name, Remote-URL und eine Beschreibung. Nachdem diese Felder ausgefüllt sind, lässt sich das API konfigurieren und testen. In den allgemeinen Einstellungen können die Namen für das API nochmals überarbeitet werden. Ebenfalls kann hier ein HTTP-Lese- und Verbindungs-Timeout gesetzt werden.

Im folgenden Menüpunkt „Regeln" lassen sich Standard-Parameter und die HTTP-Methoden für den Connector vorkonfigurieren. Unter dem Menüpunkt „Sicherheit" lassen sich verschiedene Sicherheits-Policies wie zum Beispiel die Authentifizierung über SSL durchsetzen.

Zu guter Letzt lässt sich der Connector testen. Im vorliegenden Fall soll getestet werden, ob für den Kunden mit der ID=„203" Daten vorliegen. An die Service-URL wird hierfür lediglich „?id=203" angehängt und auf „Test-Endpunkt" geklickt. Oracle MCS führt die Anforderung aus und zeigt im unteren Teil der Webseite das Ergebnis an *(siehe Abbildung 7)*.

Nach dem Einbinden des Webservice in den MCS ist es möglich, diesen in der weiteren Entwicklung der App aufzurufen. Der hier exemplarisch eingebundene Webservice für die Abfrage von Kunden-Informationen könnte je nach Anforderung weitere Methoden für das Abfragen aller Kunden oder das Anlegen neuer Kunden erhalten. Sind bereits entsprechende Programm-Einheiten in der Datenbank vorhanden, müssen lediglich der Webservice um den Aufruf dieser erweitert und entsprechende HTTP-Methoden zugewiesen werden.

Fazit

Egal, ob selbstentwickelte Java-Klasse im JDeveloper oder ORDS – viele Wege führen zum Ziel, PL/SQL-Logik auf der Datenbank für andere Anwendungen oder mobile Apps verfügbar zu machen. Mit den richtigen Werkzeugen und ein wenig Know-how lässt sich dies bereits umsetzen. Mit dem Mobile Cloud Service bietet Oracle darüber hinaus die Möglichkeit, APIs basierend auf Webservices in eine App mit einzubinden, was dem App-Entwickler eine Vielzahl von Aufgaben abnimmt. Aber auch ohne MCS lassen sich solche Services gut in mobile Apps integrieren. Erwähnt seien an dieser Stelle das Oracle Mobile Application Framework (MAF) und Apex Mobile.

Weiterführender Link

- Oracle MCS Tutorial:
  http://www.oracle.com/webfolder/technetwork/tutorials/cloud/ocms/OCMS_MBE_OBE_tutorial.html

Daniel Kampf
dkampf@pitss.com