SpringSource Advanced Pack für die Oracle Datenbank

Autor: Stefan Glase, OPITZ CONSULTING GmbH

Der Oracle Datenbank-Server hat sich unter anderem durch umfangreiche Unterstützung kritischer Funktionen wie Replikation, Clustering und komplexer Datentypen als Markt- und Technologieführer etabliert. Das Spring Framework ist ein Applikations-Framework für die Java-Plattform auf Open-Source-Basis und verfolgt das Ziel, die Entwicklung von Java-Enterprise-Anwendungen zu vereinfachen und gute Programmierpraktiken zu fördern. Im Vordergrund steht hierbei die Unterstützung für das Zusammenspiel und die Austauschbarkeit lose gekoppelter Komponenten.

Aus Java-Applikationen erfolgt der Zugriff auf relationale Datenbanken über die JDBC-Programmierschnittstellen. JDBC steht für Java Database Connectivity. Für die Verwendung von Funktionen, die über den Standard hinausgehen, stellt Oracle eine erweiterte JDBC-Implementierung bereit, deren Einsatz jedoch häufig Änderungen im Code bedingt. Hier tritt das SpringSource Advanced Pack for Oracle Database auf den Plan und erlaubt Applikationen, die auf dem Spring Framework aufbauen, einen vereinfachten Zugriff auf spezifische Funktionen der Oracle Datenbank, ohne dass Anpassungen in der Implementierung der Datenzugriffsschicht notwendig sind.

Als Teil der SpringSource Performance Suite gehört es in den kommerziellen Stack der SpringSource Enterprise Subscription des Unternehmens SpringSource. Diese Subscription beinhaltet neben der Performance Suite die Spring Enterprise Edition und den SpringSource Support und dient der Sicherstellung von produktiven und hochverfügbaren Spring-getriebenen Anwendungen.

Das dem Unternehmen SpringSource namensgebende Open-Source-Produkt Spring Framework in der derzeit aktuellen Version 2.5 adressiert bereits eine Vielzahl der Funktionen der JDBC-Implementierung von Oracle. Das SpringSource Advanced Pack for Oracle Database setzt an den Stellen an, wo die Unterstützung des Spring Frameworks für über den Standard hinausgehende Funktionen aufhört, und bietet mehrere Funktionen.

XML-Typen und Oracle Advanced Data Types

Oracle Datenbanken sind in der Lage, XML-Daten in der Datenbank zu verwalten und auch über JDBC mittels spezieller Klassen verfügbar zu machen. Die durch das Spring Framework bereitgestellten JDBC-Funktionen erlauben bereits mit den Interfaces `SqlTypeValue` und `SqlReturnType` die Verarbeitung von XML-Daten. Darüber hinaus kennen sowohl die Oracle Datenbank als auch die Sprache PL/SQL weitere proprietäre Daten-Typen, auf welche allerdings nicht über die JDBC API zugegriffen werden kann. Erst über Erweiterungen und entsprechende APIs ist die Verwendung von Daten-Typen wie `ARRAY` und `STRUCT` möglich.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:orcl="http://www.springsource.com/schema/orcl"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/sche-
ma/beans/spring-beans.xsd        http://www.springsource.com/schema/orcl http://www.springsource.com/
schema/orcl/ap-oracle-db-1.0.xsd">

    <orcl:pooling-datasource id="poolingDataSource"
        url="jdbc:oracle:thin:@(description=(address_list=(address=(host=rac1)(protocol=tcp)(port=1521))
(address=(host=rac2)(protocol=tcp)(port=1521)))(connect_data=(service_name=racdb1)))"
        properties-location="classpath:orcl.properties" fast-connection-failover-enabled="true"
        ONS-configuration="rac1:6200,rac2:6200" />

    <bean id="transactionManager"
        class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="poolingDataSource" />
    </bean>

</beans>
```

Listing 1: Konfiguration der DataSource für Spring FCF im RAC

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:orcl="http://www.springsource.com/schema/orcl"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
      http://www.springframework.org/schema/aop
      http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
      http://www.springsource.com/schema/orcl
      http://www.springsource.com/schema/orcl/ap-oracle-db-1.0.xsd
      http://www.springframework.org/schema/tx
      http://www.springframework.org/schema/tx/spring-                    tx.xsd">

  <aop:config>
      <aop:advisor
          pointcut="@annotation(org.springframework.transaction.annotation.Transac-
tional)"
          advice-ref="racFailoverInterceptor" order="1" />
    </aop:config>

    <orcl:rac-failover-interceptor id="racFailoverInterceptor" />

    <tx:annotation-driven />

</beans>
```

Listing 2: Konfiguration des Failover Interceptors für Spring FCF im RAC

SpringSource bietet in diesem Bereich mit dem SpringSource Advanced Pack for Oracle Database eine tiefer gehende Dokumentation und stellt Beispiele und Best Practices bereit. Dies ist sicherlich bei der einen oder anderen Problemstellung sehr hilfreich, zu den wirklich interessanten Leistungsmerkmalen des Advanced Pack zählen aber verschiedene Funktionen.

Spring Fast Connection Failover

Mit Oracle RAC (Real Application Cluster) lässt sich eine Datenbank zur Verbesserung der Ausfallsicherheit auf mehreren Knoten eines Rechnerverbundes betreiben. Jeder Knoten kann Anfragen bedienen, im Fehlerfall übernehmen andere Knoten im Rechnerverbund seine Funktionalität. Wird ein Fehler erkannt, so findet ein Rollback statt.

Eine neue Transaktion wird mittels Fast Connection Failover (FCF) auf einem anderen Knoten initiiert. Die direkte Verwendung dieser Funktion erfordert – ohne die Unterstützung des SpringSource Advanced Packs for Oracle Database – Änderungen an der Datenzugriffsschicht.

Das Spring Fast Connection Failover erkennt nun einen Transaktionsfehler auf einem Knoten und stößt eine erneute Ausführung der gesamten Transaktion an.

Im Erfolgsfall der zweiten Transaktion wird dieser Prozess komplett transparent für den anfragenden Client ablaufen und sich lediglich in einer leichten Verzögerung der Antwort bemerkbar machen. Für die Konfiguration ist eine `DataSource` für RAC und die Einbindung eines AOP Advisors als Failover Interceptor zur Wiederholung der Versuche notwendig.

Listing 1 zeigt die Konfiguration der Datenquelle für die Verwendung eines Real Application Clusters in einer Spring-Applikation.

Vereinfachend wirkt sich dabei die Verwendung des `orcl`-Namensraumes aus. Neben der Adresse (URL) des thin-Treibers finden sich hier Attribute zum Laden notwendiger Verbindungsdaten (Benutzername und Passwort), zum Aktivieren des Fast Connection Failovers und ein weiteres Attribut, das die Konfiguration des Oracle Notification Service (ONS) steuert.

In Listing 2 wird mittels aspektorientierter Programmierung ein Aspekt um die Ausführung der transaktionalen Methoden gewoben. Dieser Aspekt versucht, fehlgeschlagene Versuche nach einem Rollback auf einem anderen Knoten im RAC in einer neuen Transaktion auszuführen. Das Listing geht davon aus, dass Transaktionsvorschriften anhand von Annotationen in der Spring-Applikation definiert wurden. Ein Listing für die Konfiguration bei Verzicht auf Transactional-Annotationen existiert ebenfalls in der Dokumentation des SpringSource Advanced Pack for Oracle Database.

Oracle Streams Advanced Queueing

Oracle Streams dient der Weitergabe von Informationen, Transaktionsdaten und Events innerhalb eines einzigen Datenstromes. Die Weitergabe kann innerhalb einer einzelnen Datenbank stattfinden oder von einer Datenbank an andere Datenbanken erfolgen. Mit Oracle Streams lassen sich Daten rep-

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:orcl="http://www.springsource.com/schema/orcl"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springsource.com/schema/orcl
        http://www.springsource.com/schema/orcl/ap-oracle-db-1.0.xsd">

    <orcl:pooling-datasource id="dataSource" />

    <orcl:aq-jms-connection-factory id="connectionFactory" data-source="dataSource"
/>

    <bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate">
        <property name="connectionFactory" ref="connectionFactory" />
    </bean>

</beans>
```

Listing 3: Bereitstellung des JmsTemplates

```
@Service("shoppingService")
public class ShoppingServiceImpl implements ShoppingService {

    private JmsTemplate jmsTemplate;

    @Resource
    public void setJmsTemplate(JmsTemplate jmsTemplate) {
        this.jmsTemplate = jmsTemplate;
    }

    @Transactional(propagation = Propagation.REQUIRED)
    public Order enqueueTrackingNumber(Order order) {
        this.jmsTemplate.convertAndSend("oe.jms_text_que", order.getTracking-
Number());
        return order;
    }
}
```

Listing 4: Verwendung des Jms Templates zum Versand von Nachrichten

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:orcl="http://www.springsource.com/schema/orcl"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springsource.com/schema/orcl
        http://www.springsource.com/schema/orcl/ap-oracle-db-1.0.xsd">

    <orcl:pooling-datasource id="dataSource" url="${jdbc.url}"
username="${jdbc.username}"
        password="${jdbc.password}">
        <orcl:username-connection-proxy connection-context-
provider="usernameProvider" />
    </orcl:pooling-datasource>

    <bean id="usernameProvider" class="de.opitzconsulting.opitztoys.
db.CurrentUsernameProvider" />

</beans>
```

Listing 5: Konfiguration der Datenquelle für einen Proxy-Nutzer

```
public class CurrentUsernameProvider implements ConnectionUsernameProvider {
    private UserBean userBean;
    @Resource
    public void setUserBean(UserBean userBean) {
        this.userBean = userBean;
    }
    /* Methode aus dem Interface ConnectionUsernameProvider */
    public String getUserName() {
        if (userBean.getUser() == null)
            return „readOnlyUser“;
        return userBean.getUserAsDbUser();
    }
}
```

Listing 6: Implementierung des CurrentUsernameProviders

lizieren, Message Queueing anstoßen, Daten in ein Data Warehouse übertragen oder die Datensicherung in Hochverfügbarkeitssystemen durchführen.

Die Message-Queueing-Unterstützung wird von Advanced Queueing (AQ) bereit gestellt und integriert sich nahtlos in das Java Message Service API. Rechenintensive 2-Phasen-Commits sind hier kein Thema, da AQ in der Datenbank läuft und somit eine gemeinsame Transaktion für den Datenbankzugriff und das Messaging genutzt wird. Hierfür bietet das Advanced Pack eine Erweiterung der XML-Konfigurationssprache von Spring an, die eine vereinfachte Konfiguration erlaubt.

In Listing 3 wird zuerst analog zum vorherigen Beispiel eine Data Source erzeugt. An dieser Stelle wird die Minimalkonfiguration verwendet: Die Verbindungsdaten liegen im Klassenpfad in der Datei orcl.properties und werden ohne weitere Einstellungen geladen.

Eine JMS-ConnectionFactory für Advanced Queueing lässt sich mithilfe des orcl-Namensraumes ebenfalls sehr einfach konfigurieren. Diese wird durch das JmsTemplate verwendet, welches in der Applikation beim einfachen Versand von Nachrichten hilfreich ist.

Listing 4 zeigt einen vereinfachten Spring-Service mit dem Namen `shoppingService`, welcher sich das in Listing 3 konfigurierte `JmsTemplate` injizieren lässt. Mithilfe des `JmsTemplate` lässt sich in der Methode `enqueueTrackingNumber()` in einfacher Syntax eine Bestellnummer, die für eine Bestellung erzeugt wurde, versenden.

Custom DataSource Connection Preparer

In der Oracle JDBC-Implementierung findet sich die Funktion der Authentifizierung eines Endbenutzers durch einen zwischengeschalteten Proxy-Benutzer mit beschränkten Berechtigungen. Sämtliche Zugriffe auf die Datenbank durch den Endbenutzer werden nicht auf dem Proxy-Benutzer, sondern auf dem Datenbank-Zugang des Endbenutzers ausgeführt. Diese Funktion ist insbesondere für Web-Applikationen in einem geschlossenen Benutzerkreis und die Integration bereits vorhandener Datenbank-Logik interessant.

Mit dem XML-Namensraum „`orcl`“ aus dem Advanced Pack ist die simple Konfiguration eines Connection-Pools für einen Proxy-Benutzer möglich. Um den Namen des tatsächlichen Endbenutzers bekannt zu machen, muss das Interface `ConnectionUsernameProvider` mit der Methode `getUserName` implementiert werden. Das „SpringSource Advanced Pack for Oracle Database“ beinhaltet neben der API für die Konfiguration des Datenbank-Zugangs auch hilfreiche Dokumentation.

Um Aktionen auf der Datenbank unter der Benutzerkennung des Anwenders der Spring-Applikation auszuführen, wird die Data Source um ein Kind-Element `username-connection-proxy` ergänzt. Dieses referenziert die Implementierung des vorgegebenen Interfaces `ConnectionUsernameProvider` zur Bereitstellung des Datenbank-Benutzernamens des aktuell agierenden Applikations-Anwenders. In Listing 6 findet sich eine vereinfachte Darstellung: Anhand des Benutzers in der `UserBean` wird der Datenbank-Benutzer bestimmt. Ist kein Benutzer an der Applikation bekannt, so wird ein Benutzer `readOnlyUser` mit eingeschränkten Zugriffsrechten verwendet.

Fazit

Das SpringSource Advanced Pack for Oracle Database macht Funktionen, die nur in Oracle Datenbanken zur Verfügung stehen, auf unkomplizierte Weise für Spring-Applikationen zugänglich und unterstützt zugleich die Portabilität der Anwendung, da diese Funktionen nicht die Implementierung der Datenzugriffslogik beeinflussen. Die Unterstützung von transparentem Oracle RAC Failover erhöht die Verfügbarkeit der Datenbank und auf Message Queue basierende Anwendungen erfreuen sich einer enormen Vereinfachung durch den Einsatz von Oracle Streams Advanced Queueing. Die Unterstützung von nativem XML und der Oracle Advanced Data Types rundet dieses Paket ab.

Kontakt:
Stefan Glase
stefan.glase@opitz-consulting.de