Deployen? Natuurlijk!

Tackle de interferentie met de klassen

In de huidige wereld van informatievoorziening is het niet deployen van een informatiesysteem ondenkbaar. Verschillende applicatieserves zijn tegenwoordig beschikbaar met allemaal hun eigenaardigheden. Elke applicatieserver heeft bovendien eigen systeemklassen die kunnen interfereren met klassen van een gebruikt framework.

De OC4J en de WebLogic zijn beide Java EE servers. Een Java EE server biedt voor componenten zoals servlets of enterprise beans onderliggende services aan in de vorm van een container. Een container is de interface tussen de component en de platform specifieke functionaliteit die de component ondersteunt. Voordat een webcomponent of een enterprise bean component uitgevoerd kan worden, moet deze eerst geïnstalleerd worden in een Java EE module en gedeployd naar de juiste container. Bij het installatieproces moeten de componentinstellingen van de container gespecificeerd worden. Deze instellingen passen de onderliggende voorzieningen van de Java EE server aan. Dit zijn services zoals beveiliging, transactiemanagement, JNDI-lookups, en remote connectivity. Bijvoorbeeld

- Met het beveiligingsmodel kunnen we een bepaalde component zodanig configureren dat bepaalde resources alleen door geautoriseerde gebruikers kunnen worden benaderd.
- Met het transactiemodel kunnen we relaties tussen methoden specificeren die tezamen één transactie voorstellen, zodat alle methoden als een eenheid gezien worden.
- De JNDI-lookup service is een interface om resources geconfigureerd in de applicatieserver te benaderen vanuit de applicatie.
- Het remote connectivity model zorgt voor de communicatie tussen enterprise beans en een client, met als gevolg dat de client de enterprise bean kan benaderen alsof deze op dezelfde virtuele machine draait.

Doordat de Java EE architectuur configureerbare services biedt, kunnen componenten binnen dezelfde applicatie, afhankelijk waar de componenten gedeployd zijn, een ander gedrag vertonen. Bijvoorbeeld een enterprise bean kan beveiligingsinstellingen hebben die in de ene productieomgeving wel bepaalde data kan benaderen en in een andere productieomgeving niet. De container onderhoudt bovendien services die niet configureerbaar zijn, zoals lifecycles van servlets en enterprise beans, data persistentie en toegang tot de Java EE API's. We onderscheiden in dit artikel de volgende containertypen:

- Java EE server - het runtime gedeelte van een Java EE product. Een Java EE server biedt een EJB container en een web container.
- Enterprise JavaBeans (EJB) container - verzorgt voor Java EE applicaties het uitvoeren van enterprise beans.
- Web container - verzorgt voor Java EE applicaties het uitvoeren van JSP componenten en servlet componenten.

We gaan verder met het deployen van de Java persistentie implementatie TopLink Essentials op een WebLogic server. Daarna gaan we in op het gebruik van Enterprise JavaBeans (EJB's). De JavaServer Faces componentenbibliotheek Trinidad, die gebruik maakt van AJAX, is het volgende punt van aandacht. Het partial page rendering mechanisme waarvan Trinidad gebruikt maakt, kan problemen geven op de WebLogic server. Met behulp van een voorbeeld introduceren we een oplossing voor dit probleem. Vervolgens kijken we naar de problematiek omtrent het classloading mechanisme, met name de bibliotheek ANTLR. Het besproken voorbeeld maakt gebruik van Hibernate als object/relationeel framework dat een eigen ANTLR implementatie heeft.

Java Persistentie

Java persistentie is een Java standaard voor persistentie. De persistentie maakt gebruik van een object/relationele mapping om het gat te dichten tussen een object georiënteerd model en een relationele database. Een veel gebruikte Java persistentie implementatie is TopLink Essentials. Een applicatie die gebruikt maakt van TopLink Essentials kan niet gedeployd worden op een WebLogic server door de volgende exceptie:

```
java.lang.IllegalArgumentException: URI is not hierarchical
at java.io.File.<init>(File.java:335)
at oracle.toplink.essentials.ejb.cmp3.persistence.ArchiveFactoryImpl.
createArchive
(ArchiveFactoryImpl.java:104)
```

De exceptie kan worden herleid naar het feit dat WebLogic verkeerde URL's naar TopLink Essentials stuurt. Het probleem is dat de URL's de vorm jar:file:c:/iets.jar hebben. Volgens de URI specificatie is dit een invalide URI omdat er geen '/' achter het schema (jar:file:) staat. In alle gevallen wordt de URI als 'opaque' gezien; wat onacceptabel is voor de constructor java.io.File(URI), met als gevolg de bovenstaande exceptie.
Een mogelijke oplossing is de volgende regel code in createArchive(URL):

```
File f = new File(uri);
```

te vervangen door:

```
File f;
if (!uri.isOpaque()) {
    f = new File(uri);
} else {
    f = new File(url.getPath()):
}
```

Om TopLink Essentials opnieuw te compileren zijn de bibliotheken ant.jar en jta.jar nodig. Zonder de bovengenoemde oplossing is het niet mogelijk TopLink Essentials te specificeren als de JPA provider op een WebLogic server. Om DML operaties te kunnen uitvoeren hebben we een transactie controller nodig, zoals:

```
import javax.transaction.TransactionManager;
import oracle.toplink.essentials.transaction.JTATransactionController;

public class WebLogicTransactionController extends
JTATransactionController {

    private static final String JNDI_TRANSACTION_MANAGER_NAME =
                                        "javax.transaction.
TransactionManager";

    public WebLogicTransactionController() {
    }

    protected TransactionManager acquireTransactionManager() throws
Exception {
        return (TransactionManager)jndiLookup(JNDI_TRANSACTION_MANAGER_
NAME);
    }
}

</code >

Deze klasse moet geregistreerd worden in de file persistence.xml

<code>

<?xml version="1.0" encoding="windows-1252" ?>
<persistence ...>
    <persistence-unit name="Optimize">
        <provider>oracle.toplink.essentials.PersistenceProvider</provi-
der>
        ...
        <properties>
            <property name="toplink.logging.level" value="FINE"/>
            <property name="toplink.target-database" value="Oracle"/>
            <property name="toplink.cache.shared.default"
value="false"/>
            <property name="toplink.target-server"
                      value="model.utils.
WebLogicTransactionController"/>
        </properties>
    </persistence-unit>
</persistence>
```

De bovenstaande file definieert een persistentie unit, die benaderbaar is onder de naam 'Optimize'. Bovendien wordt een provider vastgelegd, in dit geval TopLink Essentials, en worden een aantal configuratie eigenschappen ingesteld. De eigenschap toplink.target-server zegt dat we TopLink Essentials op een bepaalde server gaan deployen.

Een aantal opmerkingen is nog wel op zijn plaats. Elke applicatie met een persistente toestand moet op één of andere manier interactie hebben met de persistentie provider, wanneer een bepaalde toestand in het geheugen naar de database gepropageerd moet worden (of omgekeerd). Met andere woorden, we moeten de interface van de persistentie provider gebruiken om objecten te laden en op te slaan. Deze interface is de zogenaamde EntityManager. Elke EntityManager is geassocieerd met een persistentie context. Een persistentie context is een soort cache die bijhoudt welke objecten zijn veranderd in een bepaalde werkeenheid (transactie). De persistentie context is niet iets dat zichtbaar is binnen de applicatie; het is geen API die aangeroepen kan worden.

TopLink Essentials maakt standaard gebruik van een cache die gedeeld wordt door clients gekoppeld aan een bepaalde sessie. Bijvoorbeeld als een client een object uit de database haalt of een object naar de database schrijft, wordt een kopie van het object opgeslagen in de cache van de applicatieserver. Het object kan vervolgens door andere clients uit de cache gehaald worden. Een volgend probleem doet zich nu voor: stel we voegen een entiteit toe, verwijderen deze vervolgens en voegen deze vervolgens weer toe. Het laatste toevoegen heeft tot gevolg dat er een exceptie gegooid wordt met de melding dat de entiteit al bestaat. Om dit te verhelpen moet de eigenschap toplink.cache.shared.default op false gezet worden. Deze eigenschap zegt dat de cache gedeeld wordt door meerdere clients. Door de eigenschap op false te zetten wordt de cache exclusief gebruikt door één bepaalde client. Deze client kan nog steeds aan objecten refereren binnen een gedeelde cache, maar andere clients kunnen niet aan objecten refereren binnen de exclusieve cache.
Als laatste stap moeten de klassen van TopLink Essentials toegevoegd worden aan het klassenpad van de WebLogic server. Een WebLogic Server werkt met zogenaamde domeinen. Binnen zo'n domein bevinden zich allerlei configuratiefiles, waaronder een file om bepaalde klassen te laden tijdens het opstarten van het domein. Deze file bevindt zich in de directory <domain-home>/bin en is genaamd setDomainEnv. Om ervoor te zorgen dat de

klassen van TopLink Essentials geladen worden tijdens het opstarten, moet het volgende worden toegevoegd:

```
@REM SET THE CLASSPATH
set TOPLINK=..\..\..\wlserver_10.3\ADF\lib\toplink-essentials.jar
set TOPLINK=%TOPLINK%;..\..\..\wlserver_10.3\ADF\lib\toplink-essentials-
agent.jar
set CLASSPATH=%TOPLINK%;%PRE_CLASSPATH%...
```

TopLink Essentials is het ingebouwde Java persistentie framework van de OC4J. Op een OC4J zijn de bovenstaande aanpassingen niet nodig. Het ingebouwde Java persistentie framework op de WebLogic server is Kodo (OpenJPA). Als Kodo als persistentie provider wordt gekozen zijn de bovenstaande aanpassingen eveneens niet nodig.

Enterprise JavaBeans

Geschreven in de programmeertaal Java zijn Enterprise JavaBeans de componenten die de businesslogica bevatten. Om een EJB in een client te gebruiken of een resource geconfigureerd op een applicatieserver te gebruiken in een EJB, biedt Java EE de Java Naming and Directory interface (JNDI). JNDI biedt applicaties de mogelijkheid attributen te associëren met objecten en te zoeken naar objecten met behulp van hun attributen. Bijvoorbeeld een applicatie kan door gebruik te maken van JNDI elk type Java object opslaan en ophalen met de gegeven JNDI naam. Als voorbeeld kijken we naar de volgende enterprise bean die we onder de JNDI naam 'ejb/Optimize' registreren

```
@Remote
public interface Optimize {...}

@Stateless(name = "ejb/Optimize", mappedName = "ejb/Optimize")
public class OptimizeBean implements Optimize {...}
```

Deze enterprise bean kan vervolgens in een client worden benaderd met behulp van een JNDI lookup. In het geval van een OC4J wordt gebruik gemaakt van de naam die gekoppeld is aan het attribuut name, de WebLogic server maakt daarentegen gebruik van het attribuut mappedName.

Een container implementeert de omgeving voor een component en biedt deze aan als een JNDI context. Een component kan de omgeving benaderen door gebruik te maken van de JNDI interfaces, dat wil zeggen een component creëert een object InitialContext om de context op te halen. Deze context kan vervolgens gebruikt worden om naar geregistreerde objecten te zoeken zoals de bovenstaande enterprise bean. Als er gebruik gemaakt wordt van een OC4J, kan de enterprise bean in de client worden benaderd met:

```
Context context = new InitialContext();
Optimize optimize = (Optimize)context.lookup("ejb/Optimize");
```

Met een WebLogic server kan de enterprise bean benadert worden met

```
Context context = new InitialContext();
Optimize optimize = (Optimize)context.lookup("ejb/Optimize#datamodel.
logic.Optimize");
```

Voor de # staat de mappedName achter de # staat de package plus de naam van de klasse. Het element name is alleen binnen de applicatie benaderbaar. Om aan resources te refereren binnen de applicatieserver wordt gebruik gemaakt van het element mappedName. De OC4J ondersteunt het mappedName element niet, de WebLogic server wel.

In het algemeen is het aan te raden om in een applicatieserver resources te configureren, zoals bijvoorbeeld een Java Messaging Service (JMS). JMS is een messaging standaard die componenten in staat stelt om messages te creëren, te versturen, te ontvangen en te lezen. Om een JMS service op een WebLogic server te configureren moeten we een aantal stappen volgen:

- Creëren van een persistentie store - Een fysieke repository om systeem data op te slaan.
- Creëren van een JMS server - Containers die queues en topics in JMS modules van de JMS server onderhouden.
- Creëren van een JMS module - JMS systeem resources worden geconfigureerd als modules, deze resources zijn bijvoorbeeld queues, topics en connectie factories.
- Creëren van een subdeployment - Een mechanisme waarmee JMS resources gegroepeerd aan een server resource (zoals een JMS server) gekoppeld worden.
- Creëren van JMS resources - Queues, topics en connectie factories. Queues en topics zijn zogenaamde bestemmingen die een client kan specificeren als zijnde het doel respectievelijk de bron van de messages die door de client geproduceerd en respectievelijk geconsumeerd worden. Een connectie factory is het object dat gebruikt wordt om een connectie te maken naar een JMS server.

De persistentie store, JMS server en de JMS module worden toegevoegd aan de file config.xml deze bevindt zich in de directory <domain-home>/config. De JMS resources worden in aparte file geplaatst in de directory <domain-home>/config/jms. In deze file bevinden zich de JNDI namen. Met een OC4J moeten de JMS resources toegevoegd worden aan de file jms.xml deze bevindt zich in de directory <oc4j-home>/config. Resources geconfigureerd op een applicatieserver kunnen sinds Java EE worden benaderd met behulp van resource injectie. Stel we hebben een connectie factory geconfigureerd in de applicatieserver en deze onder de JNDI naam jms/Optimize geregistreerd. Als we een OC4J gebruiken kan deze resource binnen een enterprise bean als volgt benaderd worden:

```
@Resource(name = "jms/Optimize")
private ConnectionFactory optimize;
```

Op een WebLogic Server wordt resource injectie niet vanzelf ondersteund. Als een bepaalde enterprise bean resources wenst te gebruiken moeten deze in de file weblogic-ejb-jar.xml toegevoegd worden. (Deze file, die de beans definieert, kan bepaalde deployment instellingen overriden.) Het is natuurlijk ook mogelijk de resource te benaderen met behulp van een JDNI lookup.

Als een applicatie gedeployd wordt, kan er een deployment plan aangemaakt worden. Bij een WebLogic server bevinden zich in het deployment plan verschillende module-override secties. Hierin wordt een aantal files gespecificeerd waarin we de overrides kunnen plaatsen, bijvoorbeeld:

```
<weblogic-ejb-jar...>
  <weblogic-enterprise-bean>
    <ejb-name>Optimize</ejb-name>
    <enable-call-by-reference>True</enable-call-by-reference>
    <jndi-name>ejb/Optimize</jndi-name>
  </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

Binnen deze file kunnen we eveneens resources, die gebruikt worden door een enterprise bean, specificeren. Buiten het feit dat we resources kunnen configureren die we vervolgens kunnen gebruiken in een applicatie, is het ook mogelijk in enterprise beans:

- transacties uit te voeren die door de container onderhouden worden. De EJB container bakent de transactie af, dat wil zeggen net voordat een EJB methode start wordt een transactie gestart en net voordat een EJB methode stopt eindigt de transactie (met een commit of rollback).
- de lifecycle te gebruiken om resources aan te maken (PostConstruct fase) en te verwijderen (PreDestroy fase). Zo worden op de juiste momenten resources aangemaakt en weer verwijderd. Als we een connectie nodig hebben naar een JMS provider, creëren we deze in de PostConstruct fase en verwijderen we deze in de PreDestroy fase. Als resources zoals bijvoorbeeld JMS connecties niet worden verwijderd kan het voorkomen dat nieuwe connecties naar de JMS provider niet meer worden vrijgegeven.
- Eentity managers door de container te laten onderhouden. Door gebruik te maken van een entity manager, die wordt onderhouden door de container, wordt de persistentie context automatisch gepropageerd naar alle componenten die de entity manager binnen een transactie gebruiken.

Door gebruik te maken van container onderhouden resources wordt veel code afgehandeld door de applicatieserver. Hierdoor kunnen we ons focussen op het specifieke probleem en niet op alle rompslomp eromheen.

Trinidad

Trinidad is een componenten library voor JavaServer Faces. Als een versie van Trinidad gebruikt wordt ouder dan de 10.*.10, geeft dit problemen op de WebLogic server, dat wil zeggen het partial page rendering mechanisme, dat gebruik maakt van AJAX, werkt niet. Trinidad gaat ervan uit dat het contenttype altijd text/xml is. In het geval van de WebLogic server is deze aanname onjuist, dat wil zeggen een AJAX request wordt verstuurd als text/html en door Trinidad ook als zodanig geinterpreteerd, met als gevolg dat het partial page rendering mechanisme niet functioneert. Een oplossing voor dit probleem is het hard coderen van de contenttype in Trinidad, zodat de contenttype altijd text/xml is, zoals:

```
@SuppressWarnings("deprecation")
final class XmlHttpServletResponse extends HttpServletResponseWrapper {
    private String _contentType = null;

    XmlHttpServletResponse(ServletResponse response) {
        super((HttpServletResponse)response);
        _contentType = "text/xml;charset=utf-8";
    }
...

    @Override
    public void setContentType(final String type) {
        super.setContentType(_contentType);
    }
}
```

De setContentType methode override het default gedrag en zet de contenttype altijd op text/xml. Met deze oplossing is het partial page rendering probleem er één voor de geschiedenisboeken. Als de klasse is aangepast moet de trinidad-impl-1.*.*.jar gecreëerd worden. Hiervoor zijn de volgende libraries nodig: javaee.jar, jsf-api.jar (juiste JSF versie), jsf-facelets-1.1.14.jar, servlet.jar, trinidad-api-1.*.*.jar (zelfde versie als de impl jar) en een jar dat de package javax.portlet bevat bijvoorbeeld wsrp-container.jar (deze zit bij een JDeveloper distributie). Met een OC4J ondervindt Trinidad geen problemen. Zoals eerder opgemerkt werkt een WebLogic server met domeinen. Het is mogelijk om bibliotheken voor een domein te installeren, bijvoorbeeld een Trinidad bibliotheek. Hiervoor moeten we de twee jars van Trinidad in een war-file plaatsen. Een bibliotheek kan gedeployd worden als elke andere applicatie, alleen wordt deze dan als een bibliotheek gedeployd en niet als een applicatie. In de configuratiefile (<domain-home>/config/config.xml) wordt de bibliotheek toegevoegd, bijvoorbeeld:

```
<library>
    <name>trinidad</name>
    <target>GeneralServer</target>
    <module-type>war</module-type>
    <source-path>pad\trinidad.war</source-path>
    <deployment-order>1</deployment-order>
    <security-dd-model>DDOnly</security-dd-model>
</library>
```

In de configuratiefile komen alle resources van een domain terecht, die door applicaties kunnen worden gebruikt. Om een applicatie een bepaalde bibliotheek te laten gebruiken, moeten we een referentie aan deze bibliotheek toevoegen in de file weblogic.xml (deze wordt door de WebLogic server aangemaakt als er een deployment plan wordt gemaakt):

```
<library-ref>
    <library-name>trinidad</library-name>
</library-ref>
```

Op een OC4J gaat dit anders in zijn werk. We willen bijvoorbeeld gebruik maken van een TimesTen datasource. Om dit te bewerkstelligen moet de OC4J de locatie weten van de juiste jar files. Er zijn enkele stappen die we moeten doorlopen:

- Voeg een directory shared-lib/TimesTen/5.0 toe aan de home van OC4J.
- Plaats de jar files in deze directory.
- Voeg aan de file server.xml (<oc4j-home>/config) het volgende toe

```
<shared-library name="timesten" version="5.0" library-compatible="true">
    <code-source path="pad/shared-lib/timesten/5.0/ttjdbc5.jar"/>
</shared-library>
```

- Voeg aan de file application.xml (<oc4j-home>/config) het volgende toe

```
<imported-shared-libraries>
    <import-shared-library name="timesten"/>
</imported-shared-libraries>
```

Met het gebruik van bibliotheken is het niet nodig deze aan een ear- of war-file toe te voegen.

Classloading

Het probleem classloading demonstreren we aan de hand van een voorbeeld dat gebruik maakt van Hibernate. Hibernate is evenals Java persistentie een object/relationeel framework. Binnen de meeste object/relationele frameworks wordt gebruik gemaakt van bepaalde querytalen. Hierbij wordt er geen gebruik gemaakt van databaseobjecten, maar van Javaobjecten om de query op te stellen. Uiteindelijk moet er een SQL-query worden gegenereerd. Hibernate gebruikt de ANTLR bibliotheek als de queryparser. Helaas gebruikt de WebLogic server een eigen versie van ANTLR in het systeemklassepad dat geladen wordt voordat de applicatieklassen worden geladen. Doordat de WebLogic server geen juiste isolatie voor het laden van klassen heeft, worden de Hibernate klassen in de applicatie context niet herkend. De WebLogic server lost dit op door namen van packages te prefixen. Een nadeel is echter dat de ANTLR versie in de WebLogic server deze prefix niet heeft. Een workaround voor dit probleem moet ervoor zorgen dat de Hibernate-klassen geladen worden, voordat de systeemklassen worden geladen. Dit kan worden bereikt door de eerder genoemde file setDomainEnv aan te passen. Het volgende moet worden toegevoegd:

```
@REM SET THE CLASSPATH
set CLASSPATH=..\..\..\wlserver_10.3\Hibernate\antlr-2.7.6.jar
set CLASSPATH=%CLASSPATH%;..\..\..\wlserver_10.3\Hibernate\asm.jar
set CLASSPATH=%CLASSPATH%;..\..\..\wlserver_10.3\Hibernate\asm-attrs.jar
set CLASSPATH=%CLASSPATH%;..\..\..\wlserver_10.3\Hibernate\
cglib-2.1.3.jar
set CLASSPATH=%CLASSPATH%;..\..\..\wlserver_10.3\Hibernate\commons-col-
lections-2.1.1.jar
set CLASSPATH=%CLASSPATH%;..\..\..\wlserver_10.3\Hibernate\commons-log-
ging-1.0.4.jar
set CLASSPATH=%CLASSPATH%;..\..\..\wlserver_10.3\Hibernate\
dom4j-1.6.1.jar
set CLASSPATH=%CLASSPATH%;..\..\..\wlserver_10.3\Hibernate\ehcache-
1.2.3.jar
set CLASSPATH=%CLASSPATH%;..\..\..\wlserver_10.3\Hibernate\hibernate3.
jar
set CLASSPATH=%CLASSPATH%;..\..\..\wlserver_10.3\Hibernate\jta.jar
set CLASSPATH=%CLASSPATH%;..\..\..\wlserver_10.3\TimesTen\ttjdbc5.jar

set CLASSPATH=%CLASSPATH%;%PRE_CLASSPATH%;%WEBLOGIC_CLASSPATH%;...
```

Als we Hibernate in de OC4J willen gebruiken, moeten we de bibliotheek van TopLink (bevat klassen die eveneens interfereren met Hibernate) verwijderen. In de file application.xml moeten we het volgende toevoegen:

```
<imported-shared-libraries>
    <import-shared-library name="timesten"/>
    <remove-inherited name="oracle.toplink"/>
</imported-shared-libraries>
```

Als alle Hibernate klassen op de juiste plaats staan, werkt Hibernate zonder problemen op de OC4J en de WebLogic server.

Conclusie

Drie complexe stukken software zijn de revue gepasseerd: de applicatieserver, Enterprise JavaBeans en Java persistentie (object/relationeel mapping framework). Bij het maken van applicaties is een gedegen kennis van een applicatieserver zeker zo belangrijk als kennis van de software om applicaties te bouwen. Het delegeren van resource-management naar de applicatieserver leidt ertoe dat de ontwikkeltijd verkort, de complexiteit van de applicatie verkleint en de performance verbetert.

Referenties

- Christian Bauer, Gavin King, Java Persistence with Hibernate, Manning, 2007
- Joshua Bloch, Effective Java: Programming Language Guide, Addison Wesley, 2001
- The Java EE 5 Tutorial, http://java.sun.com/javaee/5/docs/tutorial/doc/

René van Wijk, consultant/trainer bij Transfer Solutions B.V.