Der Pfadfinder – Referenzierung mit JNDI im WebLogic-Server

Anton Frank
esentri AG
Ettlingen

Schlüsselworte
JNDI, WebLogic, Lookup, Java EE, EJB, CDI

Einleitung
Hier folgt nun Ihre Einleitung als Fließtext mit einem einfachen Zeilenabstand, als Blocksatz, in Times New Roman, 11 Punkt. Bitte beachten Sie auch folgende Seitenränder:
Oben: 3,0 cm, Unten: 4,0 cm, Links: 2,5 cm, Rechts: 2,5 cm
Dies ist der Vorlauftext Ihres ausgearbeiteten Vortragsmanuskriptes, der in wenigen Sätzen beschreibt, welche Themen Ihr Vortrag behandelt.
Beim Deployment einer Applikation im WebLogic-Server wird eine EJB mit einem Namen im Naming Service registriert. Um auf eine Instanz dieser EJB zugreifen zu können, wird mit dem JNDI-Interface auf den Naming Service zugegriffen – dieser Vorgang wird als Lookup bezeichnet. Dafür wird ein Überblick über die Möglichkeiten eines Lookups auf EJB nach Java-EE-5- und Java-EE-6-Konventionen geboten, welche Namensräume sich mit JNDI bieten und wie diese anhand von Beispielen verwendet werden können. Ferner wird JNDI im Zusammenhang mit Load Balancing und Failover-Szenarien gebracht. Best Practices vermitteln abschließend zu einem soliden Umgang mit Referenzen via JNDI im Java EE Umfeld.

1 – EJBs und Referenzierung von EJBs
Wenn eine Enterprise Java Bean auf dem Server deployed wird, so wird ihre Position mit einem Namen im *Naming Service* registriert. Über das *Java Naming and Directory Interface* ist es dem Client dann möglich, den Aufenthaltsort der EJB zu ermitteln. Das JNDI greift hierbei auf ein Factory Object zu, eine Art Fabrik für die aufgerufene EJB. Diese Fabrik generiert eine Instanz der EJB, sodass der Client über RMI auf deren Funktionalitäten und die darunterliegenden Daten zugreifen kann.
Die Java EE Umgebung definiert eine Reihe von Schnittstellen, welche bestimmte Funktionsbereiche – z.B. Webanwendungen, Datenbankzugriff etc. – umfassen. Ein Application Server muss diese Schnittstellen konform zur Spezifikation implementieren. Diese Schnittstellen werden als Service Provider Interface bezeichnet. Das JNDI ist ebenfalls ein SPI, weshalb sich der Einsatz des JNDI in Funktionsumfang oder Funktionsweise unter den verschiedenen Providern unterscheidet.

Um die angebotene Funktionalität einer EJB verwenden zu können, stellt die Bean ein Interface zur Verfügung. Dieses Interface kann – abhängig von der Nutzung und dem Deployment der EJB – als lokal oder remote deklariert werden.

Neben den gängigen Annotationen um den Zugriff auf das Bean Interface als remote oder lokal zu deklarieren bieten sich dem Entwickler weitere Möglichkeiten.

@EJB
Die EJB Annotation wird verwendet, um eine Abhängigkeit zu einer anderen EJB anzugeben. Die referenzierte EJB kann über den lokalen Kontext der referenzierenden EJB aufgerufen werden.

Parameter	Beschreibung
name	Gibt den Namen an unter welchem JNDI Namen die Bean im Komponentennamensraum (java:comp/env) gefunden werden kann.
beanInterface	Spezifiziert das Interface der referenzierten Bean (notwendig bei mehreren implementierten Interfaces); Standardwert ist `Object.class`.
beanName	Gibt den Namen der referenzierten EJB an. Entspricht dem `name` Attribut der @Stateful/@Stateless Annotation.
mappedName	Die EJB lässt sich über ihren globalen JNDI mappedName Namen referenzieren. Dieses Attribut ist produktspezifisch.
description	Eine Beschreibung der EJB Referenz.

WebLogic initialisiert automatisch die annotierte Variable mit der Referenz über *Dependency Injection*.

Beispiel:

```
@EJB(name="Friend", beanName="Red", mappedName="redbean")
public class BlueBean implements Friend{
```

Im Falle von Unklarheiten können die Attribute `beanName`, `beanInterface` oder `mappedName` verwendet werden um die abhängige EJB explizit zu benennen. Die Verwendung von `mappedName` sollte mit Bedacht erfolgen, da dieses Attribut herstellerspezifisch ist.

@Stateful & @Stateless
Mit diesen Annotationen wird angegeben, ob eine Session Bean *stateful* oder *stateless* ist. Diese Annotationen markieren die EJB als Session Bean, sodass der EJB Container diese Bean entsprechend konfiguriert und sich um das Transaktionsmanagement kümmert.

Parameter	Beschreibung
name	Gibt den Namen an unter welchem die Bean im JNDI hinterlegt ist
mappedName	Definiert im WebLogic JNDI root einen globalen JNDI Namen namens „bluesclues“, unter welchem die Bean gefunden werden kann
description	Eine Beschreibung der Session Bean

Beispiel:

```
@Stateless(name="Blue", mappedName="bluebean")
public class BlueBean implements Friend{
```

Im oberen Beispiel wurde die EJB unter dem Namen „Blue“ im JNDI Tree hinterlegt. Die Bean kann im Namensraum unter diesem Namen gefunden werden, z.B. `java:global/appName/moduleName/Blue`.
Da ein *mappedName* angegeben wurde, lässt sich die Bean auch direkt über diesen Namen im JNDI Tree zu finden. Es gilt hier zu beachten dass `mappedName` stets providerspezifisch ist.
Nicht alle Applikationsserver unterstützen jedoch den Gebrauch dieses Parameters und bieten untereinander wenig Portabilität. In WebLogic wird ein Lookup auf ein mit `mappedName` gebundenes Objekt über ein *Hashtag*, gefolgt von der vollständigen Interfacebezeichnung, ausgeführt. Die Form des Lookup folgt dabei einer strengen Richtlinie und setzt sich folgendermaßen zusammen:

```
<mappedName>#<fully.qualified.interface.name>
```

Beispiel: Der `mappedName` ist „*bluesclues*“, das verwendete Interface namens *BlueManGroupCall* liegt im Package *com.acme.blue*, dann wäre der Lookup:

```
Object obj = ctx.lookup(„bluesclues#com.acme.blue.BlueManGroupCall“);
```

2 – JNDI

Das Java Naming and Directory Interface bietet auf Java basierenden Applikationen ein einheitliches Interface an, über welches mehrere Namens- und Verzeichnisservices verwendet werden können. Es ermöglicht einen einheitlichen Zugriff auf Services wie LDAP, DNS oder CORBA. Das JNDI ist auf für Java EE als SPI verfügbar, wobei der Provider für eine Java EE konforme Implementierung zuständig.
Das JNDI wird in diesem Dokument verwendet, um Lookups auf EJB im Java EE Umfeld durchzuführen. Alle hier behandelten Themen werden hier spezifisch im Bezug auf Java EE und EJB betrachtet.
Die Klassen für die Nutzung von JNDI befinden sich im *javax.naming.** Paket.
In das JNDI werden Werte zusammen mit einem Namen abgelegt. Über ein Lookup wird das Objekt zurückgegeben, welches im JNDI unter dem gesuchten Namen abgelegt wurde. Ist ein Name bei der Ablage bereits vergeben, so wird eine *NameAlreadyExistException* ausgegeben. Wird beim Aufruf eine *NameNotFoundException* ausgelöst, so wurde im JNDI kein Objekt unter dem gesuchten Namen abgelegt. Eine *NotFoundException* hingegen könnte Hinweise darauf geben dass ein Eintrag unter besagtem Namen zwar existiert, das dazugehörige Objekt aber nicht mehr existiert und womöglich das JNDI Verzeichnis nicht aktualisiert wurde.

Das JNDI speichert seine Einträge in einer Baumstruktur, welche auch als *JNDI Tree* bezeichnet wird. In WebLogic kann der JNDI Tree über die WebLogic Server Administration Console eingesehen werden. Die Einträge enthalten jeweils

- Einen Binding Name unter dem das Objekt verzeichnet ist
- Den Klassennamen des Objektes
- Einen Hashcode (dezimal) sowie
- Das toString() Ergebnis, welcher sich wie folgt zusammenbaut:

```
<ClassName>@<HashCode(Hexadecimal)>
```

Der Lookup mit JNDI war im Java EE Umfeld bis Version 5 stark kontextuell abhängig und es existierten keine globalen Einträge für Ressourcen (z.B. EJB). Die Ressource wurde abhängig von der Position des Aufrufs mit einem unterschiedlichen JNDI Namen aufgerufen. Um auf eine EJB außerhalb der Applikation zugreifen zu können musste über RMI das Home Interface der gewünschten Bean herangezogen werden um anschließend eine Instanz davon auf dem Client zu erzeugen, über welche dann die Methodenaufrufe der EJB gingen. Da nur der komponentenweite Namensraum existierte musste man für jede Komponente auf welche man von der Bean aus lokal zugreifen wollte eine Referenz über den DD erstellen.
Auf diese Problematik wurde in Java EE 6 eingegangen und durch das Hinzufügen drei neuer Namensräume vereinfacht. Dies ermöglichte den Zugriff über JNDI für jede auf dem Server abgelegte Komponente unter einem eindeutigen Namen, womit auch der Zugriff auf Komponenten auf anderen Servern vereinfacht wurde. Auch lokal wurde eine Referenz nicht mehr benötigt, da nun über modulweite (JAR/WAR) als auch applikationsweite (EAR) Namensräume ein direkter JNDI Lookup ausgeführt werden konnte.

3 – Namensräume

Das Wissen um die unterschiedlichen Namensräume mit JNDI ist unerlässlich. Das JNDI arbeitet in einem bestimmten Kontext - das heißt dass der Lookup stets vom *Client*, der aufrufenden Komponente, abhängig ist. Die Namensräume werden von *comp* über *module* und *app* bis zum *global* immer größer. Da ein größerer Namensraum einen größeren Overhead verursacht wird empfohlen möglichst auf einer feingranularen Ebene zu beginnen.

java:comp/env
Bis inklusive Java EE 5 existierte nur der java:comp Namensraum. Das comp ist eine Abkürzung für Components und bezeichnet den für eine Komponente gültigen Namensraum. Wurde an dieser Komponente (manuell) eine Referenz zu einer anderen Komponente hergestellt, so konnte auf diese über den Namensraum

java:comp/env/

zugegriffen werden. Dieser Namensraum ist stark Kontextabhängig, da jede Komponente ihren eigenen java:comp/env Namensraum besitzt. Ist die referenzierte Komponente hierbei eine EJB, so wird diese in einem Unterordner namens *ejb* in diesem Namensraum abgelegt. Selbiges gilt für Datenquellen mit *jdbc*.

Beispiele:
Java:comp/env/ejb/OrdersEntityBean
Java:comp/env/jdbc/Salary

Bei einem Lookup wird für gewöhnlich der komplette Name der Komponente (siehe obiges Beispiel) angegeben. Im WebLogic ist es bereits ausreichend, den Verzeichnispfad nach java:comp/env anzugeben (z.B. ejb/Orders). Zu diesem Namensraum sind folgende Aspekte zu beachten:

- Jede Komponente besitzt ihren eigenen java:comp/env Namensraum
- Einträge im Namensraum müssen explizit im DD angegeben werden
- Wenn EJB A eine Referenz auf EJB B deklariert, so hat A in seinem Namensraum eine Referenz auf EJB B

Dies bedeutet dass jede Komponente welche auf eine andere Komponente zugreifen wollte diese als Referenz deklarieren musste. Hat man Java EE 5 im Einsatz so sollte man bedenken dass Servlets diesen Namensraum anders interpretieren.
Während der java:comp/env Namensraum bei EJB für jede Bean einzeln existiert, so teilen sich alle Servlets innerhalb einer WAR untereinander diesen Namensraum.
Dies bedeutet, dass Servlets ohne Angabe von Referenzen andere Servlets über diesen Namensraum erreichen können. Eine EJB kann jedoch nicht auf den privaten komponentenweiten Namensraum einer anderen EJB zugreifen.

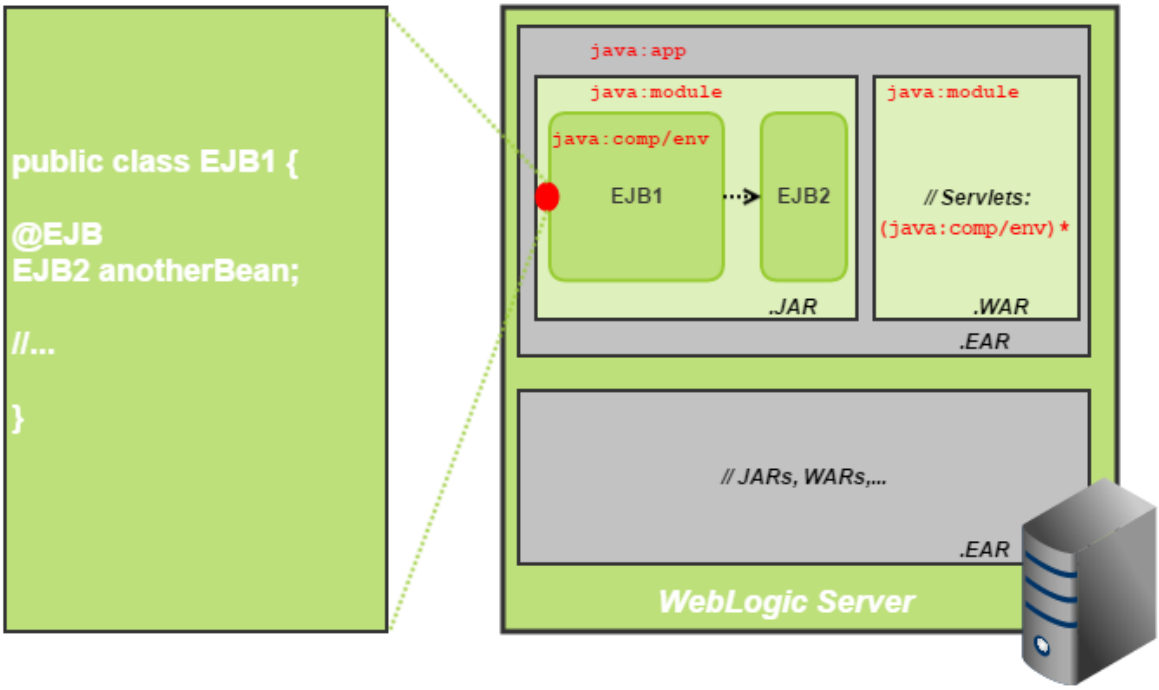

Abb. 1: java:comp/env Namensraum der EJB1 mit Referenz auf EJB2

Der java:comp Namensraum sollte für EJB nur dann verwendet werden, wenn diese eine Referenz auf die Zielkomponente besitzt. Beispielsweise lassen sich für Servlets entfernte Ressourcen (resource-ref) in der web.xml angeben, welche dann über den komponentenweiten Namensraum verwendet werden können. Hierbei wird der Name angegeben unter welchem man vom Servlet aus auf die Ressource zugreift sowie den Namen der Ressource unter welcher sie im JNDI hinterlegt ist.

Beispiel:

```
<resource-ref>
                    <res-ref-name>myUsedResourceName</res-ref-name>
                    <jndi-name>RealJNDIResourceName</jndi-name>
</resource-ref>
```

Eine Referenz über Dependency Injection sorgt ferner dafür dass die injizierte Ressource am komponentenweiten Namensraum eingetragen wird. Über DI injizierte Ressourcen sind somit ebenfalls stets über diesen Namensraum aufrufbar.

java:module/

Der Namensraum des java:module wird verwendet um (lokale) EJB innerhalb eines Moduls zu addressieren. Die Syntax für den java:module Scope lautet wie folgt:

```
java:module/<bean-name>[!<fully-qualified-interface-name>]
```

Der Name des Interface wird nur dann benötigt, wenn die EJB mehrere Interfaces implementiert. Der Namensraum beschränkt sich auf das Modul der Applikation. Ein Modul ist hierbei ein JAR oder ein WAR innerhalb einem EAR und beherbergt mehrere Komponenten (z.B. EJBs).

java:app/

Der java:app Scope wird verwendet, um einen Lookup auf lokale EJB zu betreiben welche innerhalb derselben Applikation gepackt sind. Dieser Scope wird verwendet wenn eine EJB in einer EAR gepackt ist welche mehrere Java EE Module enthält. JNDI Adressen im java:app Namensraum besitzen folgende Form:
java:app/<module-name>/<bean-name>[!<fully-qualified-interface-name>]
Der Name des Interface wird nur dann benötigt, wenn die EJB mehrere Interfaces implementiert.

java:global/

Der java:global Namensraum bietet die meisten Möglichkeiten um auf EJB innerhalb der Applikation zuzugreifen. Innerhalb dieses Namensraumes können applikationsweit remote EJB addressiert werden. Klassischer Aufbau:

java:global[/<app-name>]/<module-name>/<bean-name>[!<fully-qualified-interface-name>]

Die Parameter <app-name> sowie <module-name> sind entsprechend ihrer Paketnamen (ohne die Dateiendungen). Dabei ist <app-name> nur dann erforderlich, wenn die Ressource sich in einem EAR befindet. Der Name des Interface wird nur dann benötigt, wenn die EJB mehrere Interfaces implementiert. Der Container der EJB muss bei der Verwendung des *global* Namensraumes im JNDI einen Eintrag für jedes implementierte Interface – *lokal* wie *remote* – erstellen.
Das folgende Schaubild demonstriert in abstrakter Weise wie JAR, WAR und EAR in einer Enterprise Applikation gepackt werden und wie der Lookup auf unterschiedlichen Namensräumen zu interpretieren ist.

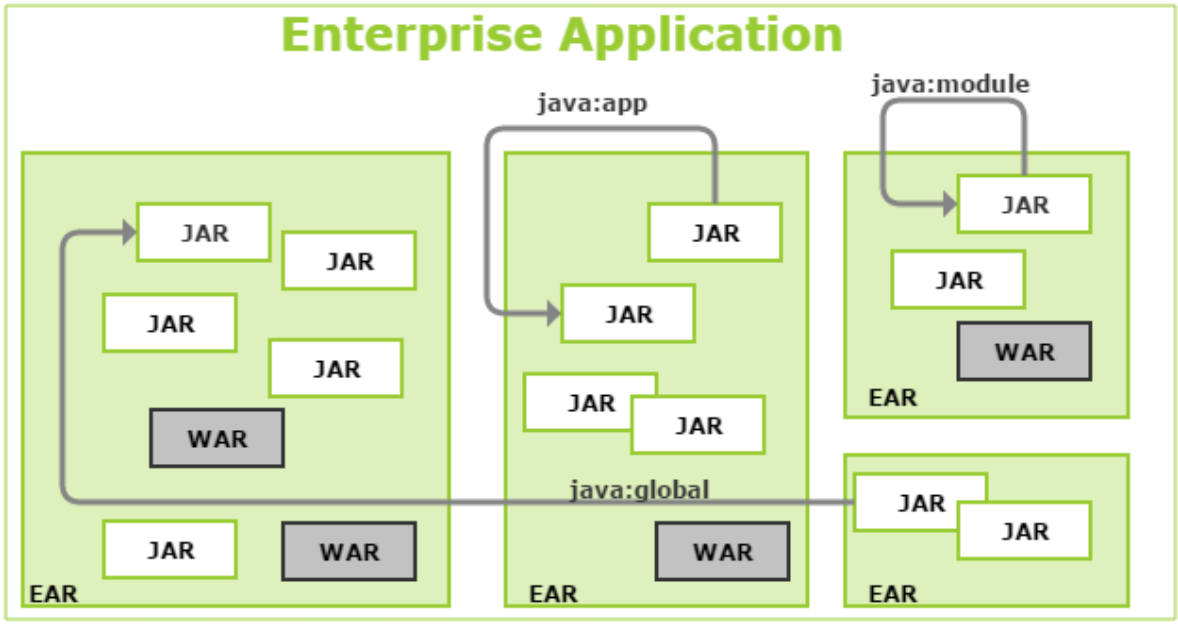

Abb. 2: Enterprise Application über mehrere EAR, Java EE 6 Namensräume

Ein Lookup auf Modulebene würde den Inhalt einer einzelnen JAR betreffen. Hier sind die EJB lokal verpackt. Vom Standpunkt der Applikationsebene kann dort ein Lookup zwischen mehreren JAR innerhalb einer EAR erfolgen. Der globale Zugriff ermöglicht letztendlich einen LookUp über mehrere EAR hinweg. Wichtig zu beachten ist die Verwendung der `@Stateful`, `@Stateless` und `@Singleton` Annotationen in EJBs. Für gewöhnlich entspricht der Name der Bean im JNDI seinem Klassennamen. Werden die eben erwähnten Annotationen jedoch angewandt und mit ihnen ein Name für die Bean deklariert, so ist diese Bean im JNDI über den Namen in der Annotation erreichbar.

4 - Manueller Lookup VS Dependency Injection

- Für einen Zugriff auf entfernte Objekte (anderer Server) muss ein manueller Lookup über den InitialContext betrieben werden, da dies mit DI nicht möglich ist. Die Ressource ist mit einem manuellen Lookup von überall aus erreichbar.
- Die Referenz auf Objekte auf demselben Server ist über DI einfacher, da der Container mit dem Management der Ressource beauftragt wird. Dies spart Code ein (u.a. try-catch Block um den InitialContext) und erfordert nicht die Angabe des vollständigen exakten JNDI Namens.
- Die Session einer Stateful Session Bean ist in beiden Fällen dieselbe Instanz. Führt man aus dieser Session heraus einen Lookup auf die EJB aus oder greift über DI auf diese zu so wird dieselbe Instanz damit angesprochen.
- Mit Java EE 6 ist eine EJB mit JNDI standardmäßig über drei Namensräume verfügbar: modulweit (java:module, JAR/WAR), applikationsweit (java:app, EAR) sowie über mehrere EAR und auch Server hinaus (java:global). Mit DI lassen sich die Referenzen nur innerhalb eines Servers (mit mappedName) ansprechen. Über DI bezogene Ressourcen sind in der referenzierenden Komponente im komponentenweiten Namensraum (java:comp) registriert.
- Ändert sich der Aufenthaltsort einer Ressource, so muss jeder Lookup welcher diese Ressource anpeilt angepasst werden. Da sich bei DI der Container um die Auflösung der Referenz kümmert ist eine lose Kopplung gegeben, der Entwickler muss keine Anpassungen vornehmen.
- Aufgrund des String-basierten Lookup kann erst zur Laufzeit ermittelt werden, ob die Referenz aufgelöst werden kann. Über DI wird die Ressource zur Kompilierungszeit bereits geprüft ob die Ressource erreichbar ist.

5 – Lookups im Cluster

Die WebLogic Implementierung des JNDI ist auch im Cluster verwendbar. Führt ein Client eine Anfrage an den Cluster über einen Kontext aus, so wird dieses nach einem Load Balancing Algorithmus – als Standard das Round Robin Verfahren - von einer Serverinstanz aus dem Cluster zurückgegeben. Dieser Vorgang wird vom WebLogic automatisch verwaltet. Als Vorraussetzung für das Load Balancing ist die Angabe aller Cluster-Knoten in der Provider-URL als kommaseparierte Liste oder ein DNS Name welcher repräsentativ für den Cluster steht. Wird als Provider-URL nur ein Server ausgewählt, so findet kein Load Balancing statt, das InitialContext Objekt wird von diesem Server zurückgegeben.
Wird eine EJB auf mehreren Cluster-Knoten deployed, so enthält der JNDI Tree jedes Servers einen sogenannten *Replica-Aware Stub* für diese EJB. In diesem Stub sind alle Serverinstanzen im Cluster verzeichnet, welche diese EJB hosten. Sollte die Ressource während einer Transaktion nicht erreichbar sein wird ein Failover ausgeführt. Hierbei übernimmt eine Kopie der Ressource welche auf einem anderen Clusterknoten deployed ist die Transaktion. Der WebLogic Server trägt die Verantwortung dafür dass alle Server im Cluster ihren JNDI Tree auf demselben Stand haben.
Das Beispiel demonstriert ein LookUp auf ein Objekt, das auf allen Cluster-Knoten verfügbar ist:

```
Hashtable env = new Hashtable( );
env.put(Context.INITIAL_CONTEXT_FACTORY,
      „weblogic.jndi.WLInitialContextFactory“);
env.put(Context.PROVIDER_URL, „t3://myclusteraddress:7007“);

//Verbindung zum clusterübergreifenden JNDI Tree
Context ctx = new InitialContext(env);
//Ab hier kann der LookUp auf das Objekt erfolgen
```

Im klassischen Fall wird die Adresse über ein DNS Namen repräsentiert. Im oberen Beispiel wird der Wert für die Property `PROVIDER_URL` auf den DNS Namen `mycluster` gesetzt. Dieser ist stellvertretend für jeden Server im Cluster. Anstattdessen ist es ebenfalls möglich eine durch Kommata getrennte Liste von Serveradressen anzugeben.

```
env.put(Context.PROVIDER_URL, „t3://ManagedServer1:7001,
ManagedServer2:7002,ManagedServer3:7003“);
```

Für den WebLogic Server entwickelte Oracle ein eigenes Protokoll für die Kommunikation über RMI Objekte, namentlich das T3 Protokoll. Laufen mehrere Teilnehmer des Clusters unter dem gleichen Port, so kann der Umgebungsvariablenparameter zur Provider URL wie folgt angegeben werden.

```
Env.put(Context.PROVIDER_URL,
„t3://ManagedServer1,ManagedServer2,ManagedServer3:7001“);
```

Zum Aufbau des InitialContext im Cluster ist es nicht zwingend notwendig, alle Server anzugeben. Die Angabe des Cluster DNS ist hierbei ausreichend. Wird nur einer der Server aus dem Cluster als Provider angegeben, so wird explizit zu diesem Server der Kontext aufgebaut, es findet kein Load Balancing statt. Die Methodenaufrufe auf die Ressourcen erfolgen dann stets gegen den explizit gewählten WebLogic Server. Da alle Serverknoten jedoch die gleiche Information miteinander teilen, kann somit dennoch auf alle Ressourcen im Cluster zugegriffen werden. Als Load Balancing Strategie stehen Round Robin, Weight-Based und Random zur Auswahl.

6 – Best Practices & Empfehlungen

Dieses Kapitel fasst abschließend alle in diesem Whitepaper ausgesprochenen Empfehlungen zusammen und ermöglicht einen Überblick für Best Practices in der Referenzierung von EJBs im WebLogic.

@Local statt @Remote

Häufig miteinander kommunizierende Beans sollten sich mit einem lokalen Interface aufrufen können und sollten daher wenn möglich auf derselben JVM hinterlegt sein. Die Verwendung von lokalen Interfaces folgt der pass-by-reference Semantik und ist wesentlich performanter als die remote pass-by-value Semantik, bei welcher Argumente erst kopiert und dann über das Netz übertragen werden müssen.

mappedName wenn möglich vermeiden

Das providerspezifische Attribut mappedName sollte so weit wie möglich vermieden werden. Jede auf dem Server hinterlegte Ressource ist unter einem Namen hinterlegt. Wird mappedName für eine EJB oder eine andere Ressource gesetzt, so muss man dafür Sorge tragen dass jeder Name auf dem Server nur einmal vorhanden ist. Ferner implementieren alle Provider den mappedName auf ihre eigene Art, sodass ein gesetzter mappedName auf einem Applikationsserver nicht zwingend auf anderen Servern akzeptiert wird.
MappedName sollte verwendet werden wenn die Applikation nur auf einem bestimmten Applikationsserver lauffähig sein soll. In allen anderen Fällen sollte darauf verzichtet werden.

Keine manuellen DD Anpassungen in Java EE 6

Die Java EE 6 Spezifikation wurde so erstellt, dass wenig bis keine manuellen Anpassungen im Deployment Deskriptor mehr notwendig sind. Die Arbeit sollte dem Container überlassen werden um sich die Mühe der Konfiguration zu ersparen. Deployment Deskriptoren sollten erst dann händisch konfiguriert werden, wenn dieselbe Konfiguration nicht durch Annotationen erreicht werden kann.

Neuere Technologie verwenden

Es sollte Java EE 6 oder höher zusammen mit EJB 3.1 verwendet werden. Für EJBs muss nicht mehr explizit RMI spezifischer Code geschrieben werden, es sind weniger Konfigurationen am DD erforderlich und die durch Java EE 6 hinzugekommenen Namensräume ermöglichen eine höhere Flexibilität. Die Verwendung neuerer Versionen bietet einen geringeren Aufwand in der Entwicklung und Wartung von Applikationen.

Dependency Injection statt Lookups innerhalb der Applikation

Referenzen innerhalb einer Applikation (EAR) sollten mit DI gesetzt werden. Dies erfordert wesentlich weniger Programmcode und mögliche Referenzprobleme werden bereits zur Kompilierzeit aufgedeckt. Referenzen außerhalb der EAR können mit Lookups erreicht werden.

Kleinsten Namensraum bei Lookups verwenden

Wird eine Referenz über einen Lookup erstellt, so sollte stets über den kleinstmöglichen Namensraum diese referenziert werden. Für eine Ressource im selben Modul sollte java:module verwendet werden, eine Ressource innerhalb derselben EAR sollte java:app verwendet werden. Darüber hinaus wird java:global eingesetzt, um Referenzen auf Ressourcen außerhalb der EAR und des Servers herzustellen.
Je weitreichender der Namensraum ist, umso höher ist der Overhead. Hier sollte ebenfalls bereits bei der Erstellung der Applikation darauf geachtet werden, dass möglichst oft miteinander kommunizierende Ressourcen beieinander liegen.

Namen logisch für Lookups definieren

Der WebLogic Server bezieht die Namensgebung für das JNDI aus den Namen der Deploymentprofile für die Module und Applikationen. Das Deploymentprofil sollte demnach für Modul wie Applikation einen für das Modul logischen passenden Namen beinhalten.

Clusterrepräsentativen DNS Namen wählen

Damit das Load Balancing richtig funktioniert und die Server die Lastenverteilung optimiert durchführen können wird empfohlen einen DNS Namen repräsentativ für den Cluster anzugeben. In diesem Falle kann das Load Balancing Verfahren eingesetzt werden, bei der Erstellung der Umgebungsvariablen für den InitialContext muss nur diese eine Adresse angegeben werden.

Load Balancing an Hardware anpassen

Über die WebLogic Konsole lässt sich ein Load Balancing Algorithmus einstellen, standardmäßig kommt Round Robin zum Einsatz. Der Algorithmus sollte sich an der Hardware des Clusters anpassen falls Server im Cluster auf unterschiedlichen Maschinen laufen. Sollte der Cluster heterogen sein (d.h. die Maschinen auf welchen die Server laufen unterschiedliche Rechenleistung haben), so sollte ein gewichteter Load Balancing Algorithmus gewählt werden um die Last auf weniger leistungsstarken Maschinen zu verringern. In homogenen Clustern kann Round Robin beibehalten werden. Wenn der Cluster eine hohe Anzahl an Transaktionen verarbeitet kann alternativ ein Zufallsalgorithmus angewandt werden.

Kontaktadresse:
Anton Frank
esentri AG
Pforzheimer Straße 132
D-76275 Ettlingen

Telefon: +49 (0) 7243 / 354 90 61
E-Mail anton.frank@esentri.com
Internet: www.esentri.com