EJB3.0

Unit-Testing Reloaded

Werner Eberling

werner.eberling@mathema.de

www.mathema.de

Automatisiertes Testen

- EJBs sind Backend-Komponenten ohne Oberfläche
- D.h. ideale Kandidaten für automatisiertes Testen
- Eine gängige Technik ist das "Unit-Testen"
  - Einzelne "Einheiten" werden isoliert für sich getestet
  - Der Test soll jederzeit ad-hoc und schnell ausführbar sein
  - Referenzierte Einheiten werden nach Bedarf durch Dummies ersetzt
    - verbesserte Performance
    - sauberer Separation
    - vereinfachte Herstellung spezieller Testkonstellationen

Automatisiertes Testen von EJBs

- Testen ohne Application-Server
  - Keine Notwendigkeit von Deployments
  - Bringt die geforderte Adhoc-Ausführbarkeit
  - Erlaubt einfaches, lokales Debugging
- Ersetzen von EJB-Implementierungen durch Mocks
  - Erfüllen dasselbe Interface wie die originale Implementierung
  - Erlauben isolierte Betrachtung der zu testenden EJBs
  - Z.B. Entkopplung von kritischen externen Systemen
  - Im einfachsten Fall:
    - Ableitung mit „kurzgeschlossenen“ Funktionen

Stolpersteine beim Testen mit EJB 2.x

- EJB-Klassen implementieren EJB-Interfaces nicht
  - new() ist nicht ohne Weiteres möglich
  - Instanziierung erfordert generierte Proxyklasse
  - Eigentlich nur für einen Container korrekt durchführbar
- Lifecycle-Callbacks müssen aufgerufen werden
- Referenzauflösung über JNDI-Kontexte
  - Isolierte Instanziierung ist schwierig
- Enge Kopplung der Persistenz an Komponentenmodell
  - Leistungsschwach und außerhalb JEE nicht anwendbar
  - Dadurch kein Markt für Lösungen ohne Application-Server

Stolpersteine beim Testen mit EJB 2.x - Beispiel


```
public class KundenVerwaltungImpl implements SessionBean {
    SessionContext sc;
    KundenDAO dao;

    void setSessionContext(SessionContext sc) {
        this.sc = sc;
        KundenDAOHome home = (KundenDAOHome)
            new InitialContext().lookup("ejb/dao");
        dao = home.create();
    }
    //... activate(), passivate(), ...

    void aktualisiereKunde(String kundenNr, String name) {
      if (!sc.getCallerPrincipal().getName().equals(kundenNr))
        throw new SecurityException("Unerlaubter Datenzugriff");
      Kunde kunde = dao.findeKunde(kundenNr);
      kunde.setName(name);
      dao.auditEintrag(kundenNr, "Name geändert: " + name);
    }
}
```

EJB 3.0 - Die Lösung ?

- EJB-Klassen implementieren ihre Interfaces
  - new() ist möglich
- Referenzauflösung über Dependency-Injection
  - Injection kann zu Testzwecken selbst implementiert werden
- Vollständig neues Persistenz-Management JPA
  - Per Spezifikation auch ohne Application-Server anwendbar
- Also: dem Unit-Testing steht nichts mehr im Wege
  - Oder doch ???

EJB 3.0 - Die Lösung ? - Beispiel


```
public class KundenVerwaltungImpl implements KundenVerwaltung {
    @Resource SessionContext sc;
    @EJB KundenDAO dao;

    // ... @PostConstruct, @PreDestroy, ...

    void aktualisiereKunde(String kundenNr, String name) {
      if (!sc.getCallerPrincipal().getName().equals(kundenNr))
        throw new SecurityException("Unerlaubter Datenzugriff");
      Kunde kunde = dao.findeKunde(kundenNr);
      kunde.setName(name);
      dao.auditEintrag(kundenNr, "Name geändert: " + name);
    }
}
```

Die Sache mit new()

- Der vom Container bereitgestellte Proxy entfällt
- Es fehlen
  - Sicherheits-Checks
  - Transaktionen
  - Methodeninterceptoren & Lifecycle-Callbacks
- Evtl. Verlust der Call-By-Value-Semantik
- Nur eine Lösung für stark isolierte Tests
  - Fokus auf die Fachlichkeit
  - Kein Test von Kollaboration oder technischen Aspekten

Das Beispiel noch einmal genauer betrachtet


```
public class KundenVerwaltungImpl implements KundenVerwaltung {
    @Resource SessionContext sc;
    @EJB KundenDAO dao;

    // ... @PostConstruct, @PreDestroy, ...

    void aktualisiereKunde(String kundenNr, String name) {
      if (!sc.getCallerPrincipal().getName().equals(kundenNr))
        throw new SecurityException("Unerlaubter Datenzugriff");
      Kunde kunde = dao.findeKunde(kundenNr);
      kunde.setName(name);
      dao.auditEintrag(kundenNr, "Name geändert: " + name);
    }
}
```

Die Sache mit der Dependency Injection

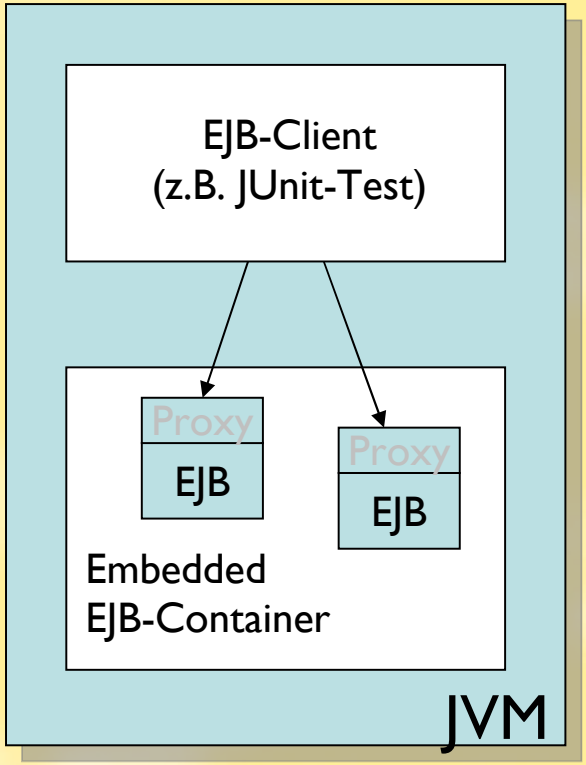
- Alternative A: Vornehmlich Injektion von Mocks
  - Mindestens eine Mock-Implementierung pro EJB
  - Sehr aufwendig
  - Nur eine Lösung für stark isolierte Tests
- Alternative B: Vornehmlich Injektion echter EJBs
  - Kaskadierung der Injektionen ist zu implementieren
  - Nachbildung der Containerarbeit
  - Fehleranfällig und ebenfalls relativ aufwendig
- Besonders unangenehm: @Resource
  - z.B. SessionContext
  - JEE-Klassen müssen nachimplementiert werden

Die Sache mit der Persistenz

- JPA grundsätzlich ohne Container möglich
- ABER: Verlust der Containerdienste
  - Injection
  - Tx-Management
- Nachbildung dieser Dienste erforderlich
  - Aufwändig
  - Verfälscht den Testfall

Ohne Container geht's leider immer noch nicht

- Auch für Tests ist ein Container notwendig
  - Innerhalb des selben Prozesses
  - Ohne Overhead eines App'Servers
- Lösung
  - Sog. Embedded-Container
- Dafür gibt es verschiedene Ansätze
  - JBoss Embedded Container
  - OpenEJB
  - Pitchfork
  - CUBA



JBoss Embedded Container

- EJB-Container von JBoss ist ohne Server verwendbar

- Vorteil:
  - hohe Vergleich-barkeit zum Serverbetrieb

- Nachteil:
  - In Entwicklung
  - Kein Support
  - Zeitaufwendiger Bootstrap

```
public class Client {
  public static void main(String[] args)
    throws Exception{
    EJB3StandaloneBootstrap.boot(null);
    EJB3StandaloneBootstrap.scanClasspath();

    Hashtable props = new Hashtable();
    props.put("java.naming.factory.initial",
"org.jnp.interfaces.LocalOnlyContextFactory");
    props.put("java.naming.factory.url.pkgs",
"org.jboss.naming:org.jnp.interfaces");
    Context ctx = new InitialContext(props);

    // EJB-Aufrufe

    EJB3StandaloneBootstrap.shutdown();
  }
}
```

JBoss a division of Red Hat

http://docs.jboss.org/ejb3/embedded/embedded.html

OpenEJB

- OpenSource EJB-Container
- Integrierbar in verschiedene Middleware-Server
  - Z.B. Apache Geronimo
- Ist auch allein remote und embedded nutzbar
- Vorteil:
  - 100'ige Vergleichbarkeit in allen Umgebungen
- Nachteil:
  - Noch kein Support für EJB 3
  - Produktionstauglichkeit ?

http://incubator.apache.org/openejb/

Pitchfork

- Verbindet EJB3 und Spring
- EJB3-Beans im Spring-Container betreibbar
- Embedded-Container des Bea-Weblogic
- Vorteil
  - Leichtgewichtiger Ansatz
- Nachteil
  - Fragliche Vergleichbarkeit mit Produktionsumgebung
  - Bootstrapping-Aufwand ?

http://interface21.com/pitchfork/

CUBA

- Eigenes Modell, equivalent zu EJB 3 SessionBeans
- Adapter- und Deskriptorgenerator ermöglicht Betrieb
  - als EJB in beliebigem EJB2 oder EJB3-Container
  - als AXIS-Webservice
  - im eigenem Embedded-Container
- Vorteil:
  - Sehr klein - im EJB-Modus nur schmaler Kompatibilitätslayer
- Nachteil:
  - Kein „echtes“ EJB 3

CUBA

http://cuba.sourceforge.net/

Fazit (1/2)

- EJB 3 ist keine Voraussetzung für Testen ohne Application-Server
  - siehe OpenEJB und CUBA
- Ein echter Vorteil wäre die Forderung gewesen:
  - Jeder EJB-Container muss ohne Application-Server nutzbar sein
- Ohne diese Forderung muss Test und Produktion u.U. in verschiedenen Containern erfolgen
  - Das ist unproblematisch für SessionBeans
  - Evtl. ein Problem bei JPA
    (spez. bei der Verwendung von Vendor-Extensions)

Fazit (2/2)

- EJB 3 bringt viele attraktive Vereinfachungen und Erweiterungen mit
  - Relevanz für automatisiertes Testen ist sehr gering
  - Für HelloWorld-Bean sieht es gut aus, sonst nicht
- Größtenteils die gleichen Maßnahmen notwendig wie für EJB 2
- Embedded-Container ist notwendig
- Persistence-Management sollte separat gelöst werden
- Neu ist das breite Bewusstsein, dass automatisiertes testen von EJBs sinnvoll ist und funktioniert

Vielen Dank ! - Noch Fragen ?

- Dann am entweder hier und jetzt
  - oder gleich am Stand der MATHEMA
    - oder später zum Nachlesen
      - Mit Extra-Kapitel zum Thema Testen
      - Ab sofort im Handel
      - ISBN 3-446-41085-6