Nr.02 | Sommer 2012 | www.ijug.eu

Java aktuell

Das Magazin der Java-Community

Java aktuell

Java im Aufwind

VisualVM
Unbekannte Kostbarkeiten des SDK

Grails
Die Suche ist vorbei

Microsoft und Java
Frei verfügbare Angebote für Software-Entwickler

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

Sonderdruck

iJUG Verbund

Interview mit John Duimovich, Distinguished Engineer in der IBM Software Group mit Schwerpunkt „Java Virtual Machines & Embedded Java", IBM Canada, Seite 16

„Java besitzt immer noch ein enormes Potenzial …" Stefan Koospal, Vorsitzender der Sun User Group Deutschland, im Gespräch mit Java aktuell, Seite 42

Der Sport ist in manchen Bereichen hilfreich für die Entwicklung von Software. Was wir vom America's Cup lernen können, Seite 44

JSFUnit

Bernd Müller und Boris Wickner, Ostfalia, Hochschule für angewandte Wissenschaften

JSFUnit ist ein Werkzeug für den Komponenten- und Integrationstest von JSF-Anwendungen. Es basiert auf Arquillian und ist wie Arquillian ein JBoss-Projekt. Der Artikel charakterisiert JSFUnit und stellt den Einsatz beispielhaft vor.

JSFUnit wird bereits seit mehreren Jahren für Komponenten- und Integrationstests von JSF-Anwendungen eingesetzt. Erste Quellen im Internet datieren von 2007. JSFUnit basierte auf Cactus [1], einem Apache-Projekt, das die Möglichkeit von Komponententests für serverseitigen Code wie Servlets und EJBs zum Ziel hatte. Cactus wird mittlerweile nicht mehr weiterentwickelt und ist in den Attic-Bereich von Apache gezogen. Durch diese Entwicklung von Cactus, vor allem aber durch das neue, hausinterne JBoss-Projekt Arquillian [2], das zukünftig einen zentralen Platz bei den Test-Werkzeugen von JBoss einnehmen wird, stand eine Überarbeitung von JSFUnit an. Die aktuelle und von den Autoren verwendete Version ist 2.0.0.Beta2.

```
<h:form id="form">
 Vorname: <h:inputText id="vorname"
          value="#{kundeView.vorname}" /> <br/>
 Nachname: <h:inputText id="nachname"
          value="#{kundeView.nachname}" /> <br/>
 <h:commandButton id="speichern"
    action="#{kundeView.speichern}"
    value="Speichern" />
</h:form>
```

Listing 1

```
@ManagedBean
public class KundeView {

  private String vorname;
  private String nachname;

  public KundeView() { }

  public String speichern() {
    return "speichern-ok.xhtml";
  }
...
}
```

Listing 2

```
<h:outputText id="ok" value="Kunde #{kundeView.
vorname}
          #{kundeView.nachname} gespeichert" />
```

Listing 3

JSFUnit-Tests werden im Container ausgeführt und erlauben den vollständigen und transparenten Zugriff auf JSF-Komponenten wie Managed Beans, EL-Ausdrücke, UI-Komponenten und die JSF-Konfiguration. Durch die Möglichkeit, einerseits clientseitige Benutzereingaben zu simulieren sowie andererseits auf JSF Managed Beans und über Arquillian auf EJBs zugreifen zu können, ist das Erstellen von Integrationstests über alle Schichten einer Java-EE-Anwendung leicht möglich.

Die Beispiel-Anwendung

Um das Ganze nicht zu abstrakt zu halten, entwickeln wir eine Anwendung, die Kundendaten persistiert. Wir verwenden JBoss AS 7, da dieser als JBoss-Produkt die beste Unterstützung durch JSFUnit und Arquillian genießt. Beginnen wir mit der JSF-Seite, die wie alle folgenden Code-Beispiele möglichst einfach gehalten ist und deren wichtigste Zeilen im Folgenden dargestellt sind (siehe Listing 1).

Es fällt zunächst die Verwendung des „id"-Attributs bei allen JSF-Tags auf. Um die interne Funktionsweise von JSF zu garantieren, muss jede Komponente eine eindeutige Id besitzen. Wird diese nicht explizit angegeben, generiert JSF eine solche Id. Diese ist jedoch wesentlich schwieriger in JSFUnit zu verwenden, sodass beim Einsatz von JSFUnit die Grundregel gilt, dass jede Komponente eine explizite Id erhalten sollte. Die für die JSF-Seite erzeugte HTML-Seite enthält diese Ids ebenfalls und das von JSFUnit intern verwendete HtmlUnit [3] nutzt diese, um Daten einzugeben und auszulesen sowie um Formulare abzuschicken.

Kommen wir nun zur verwendeten Managed Bean „KundeView". Diese ist zunächst ein einfaches POJO mit der @ManagedBean-Annotation (siehe Listing 2).

Das Navigationsziel, die JSF-Seite „speichern-ok.xhtml", enthält als einfache Nachricht an den Benutzer die Bestätigung der Kundenspeicherung (siehe Listing 3).

Mit den beiden JSF-Seiten und der Managed Bean sind wir nun in der Lage, unseren ersten JSFUnit-Test zu schreiben. Wir wählen JUnit als Testwerkzeug und geben Arquillian als eigentlichen Test-Runner mit der Annotation „@RunWith" an (siehe Listing 4). Für eine Einführung in Arquillian verweisen wir auf den Artikel von Frederik Mortensen in der Java aktuell 4/2011 [4].

Mit „@Deployment" wird die Methode annotiert, die das Deployment-Archiv erzeugt. Arquillian baut hier auf Shrinkwrap [5] auf, ebenfalls ein JBoss-Projekt. Es werden der Reihe nach definiert: die Art des Archivs (hier ein WAR), die zu verwendende web.xml, die zu deployenden Packages, die beiden JSF-Seiten sowie die „faces-config.xml", die in diesem Fall als leere Datei erzeugt wird. Wir verwenden ein von Eclipse erzeugtes Dynamic-Web-Project, was die Verwendung von „WebContent" als Präfix der JSF-Seiten erklärt. Auch die Benutzung verschiedener Verzeichnisse

Abbildung 1: Ausschnitt des Eclipse Project Explorers

für die Anwendung und die Tests sind kein Problem. Abbildung 1 zeigt den entsprechenden Ausschnitt des Project Explorers von Eclipse. Die noch nicht verwendeten Klassen und die „persistence.xml" kommen im nächsten Beispiel dazu. Der eigentliche Test wird durch JUnits @Test-Annotation markiert (siehe Listing 5).

Die initiale Seite ist mit der Annotation „@InitialPage" definiert. Die Test-Methode bekommt als Parameter ein „JSFServerSession"- und ein „JSFClientSession"-Objekt übergeben, die die entsprechenden Sessions repräsentieren. Mit „client.setValue()" erfolgt eine Benutzereingabe in die entsprechende Komponente. Wir haben zu Demonstrationszwecken beide Versionen, einmal mit und einmal ohne Formular-Präfix, gewählt. Die Methode „client.click()" schickt das Formular ab. Mit „server.getManagedBeanValue()" kann ein EL-Ausdruck evaluiert werden. „client.getElement()" liefert die entsprechende HTML-Komponente in der Antwort.

Beispiel-Anwendung mit EJB und JPA

Wir erweitern nun unsere Anwendung, um die Persistenz der Kundendaten zu realisieren. Zunächst machen wir aus der JSF-Bean eine CDI-Bean, da CDI auch für die Injektion der EJB verwendet wird. Wir geben nur die geänderten Teile der Bean wieder (siehe Listing 6).

Das JPA-Entity ist ebenfalls sehr minimalistisch gehalten, bekommt aber eine Named-Query, um alle Kunden zu selektieren (siehe Listing 7).

Die EJB besitzt Methoden, um einen Kunden zu speichern und alle Kunden zu lesen (siehe Listing 8).

Damit die CDI-Bean und die Injektion korrekt verwendet werden, muss das Deployment den CDI-Deployment-Deskriptor „beans.xml" enthalten. Wir verzichten auf eine erneute Darstellung der Deployment-Methode. Ein Integrationstest mit Eingabe der Kundendaten, Abschicken des Formulars, Einfügen in die Datenbank und anschließendem Prüfen der Datenbank-Inhalte ist nun leicht möglich. Wir beschränken uns hier auf einen rein quantitativen Test. Da wir in der „persistence.xml" Hibernate anweisen, alle Tabellen neu zu erzeugen, können wir von einer leeren Kundentabelle ausgehen. Nach dem Test muss daher genau ein Kunde existieren (siehe Listing 9).

Weitere Möglichkeiten von JSFUnit

Die Autoren konnten in diesem Artikel die Möglichkeiten von JSFUnit nur anreißen. Mit JSFUnit können praktisch alle für JSF relevanten Konstrukte getestet werden. Auch der anspruchsvolle Umgang mit generierten Ids für ein „<h:dataTable>" ist möglich, wie sie in ihrem JSF-Buch [6] für JSFUnit 1.2 gezeigt haben.

JSFUnit 2.0 bietet neben der verwendeten „@InitialPage"-Annotation eine Reihe weiterer Annotationen an, um die Erstellung von Tests zu erleichtern. Zu diesen gehören „@BasicAuthentication", „@FormAutentication", „@Proxy", „@Browser" und „@Cookies" mit der jeweiligen offensichtlichen Bedeutung.

Wo Licht ist, ist auch Schatten

Es wurde bisher nur beschrieben, was mit JSFUnit möglich ist. JSFUnit ist jedoch im Augenblick noch in einem Zustand, der auch einiges an Kritik erlaubt. Derzeit existiert keine Distribution für JSFUnit, sondern lediglich die Möglichkeit einer Maven-basierten Installation. Diese resultiert in 151 Jars mit einer Größe von insgesamt 41 MB. Dies erscheint den Autoren nicht angemessen.

Nach dem Wechsel von JBoss 7.0.1.Final auf 7.0.2.Final misslangen alle Tests ohne erkennbaren Grund. Die Verwendung von GlassFish 3.1.1 scheiterte komplett. Der Zugriff auf CDI-Beans über „server.getManagedBeanValue()" ist ebenfalls nicht

```
@RunWith(Arquillian.class)
public class KundeTest {

  @Deployment
  public static WebArchive createDeployment() {
    WebArchive war = ShrinkWrap
      .create(WebArchive.class, „test.war“)
      .setWebXML(new File(„WebContent/WEB-INF/
web.xml“))
      .addPackage(Package.getPackage(„de.pdbm.app“))
      .addPackage(Package.getPackage(„de.pdbm.test“))
      .addAsWebResource(new File(„WebContent/kunde.
xhtml“))
      .addAsWebResource(
        new File(„WebContent/speichern-ok.xhtml“))
      .addAsWebInfResource(EmptyAsset.INSTANCE,
                    „faces-config.xml“);
    return war;
  }
...
}
```

Listing 4

```
@Test
@InitialPage(„/kunde.jsf“)
public void testKunde(JSFServerSession server,
                JSFClientSession client) {
  String vorname = “Max”;
  String nachname = “Mustermann”;
Assert.assertEquals(“/kunde.xhtml”,
                server.getCurrentViewID());
  client.setValue(„form:vorname“, vorname);
  client.setValue(„nachname“, nachname);
  client.click(„speichern“);
  Assert.assertEquals(vorname,
    server.getManagedBeanValue(„#{kundeView.vorname}“));
  Assert.assertEquals(nachname,
    server.getManagedBeanValue(„#{kundeView.nachname}“));
  Assert.assertEquals(„Kunde „ + vorname + „ „
                + nachname + “ gespeichert”,
      client.getElement(“ok”).getTextContent());
  Assert.assertEquals(“/speichern-ok.xhtml”,
       server.getCurrentViewID());
  }
```

Listing 5

```
@Named
@RequestScoped
public class KundeView {
  ...  @Inject
  KundeService kundeService;
  ...
  public String speichern() {
    kundeService.speichern(new Kunde(vorname,
     nachname));
    return “speichern-ok.xhtml”;
  }
  ...
}
```

Listing 6

```
@Entity
@NamedQuery(name = „Kunde.alleKunden“,
        query = „SELECT k from Kunde k“)
public class Kunde {

        @Id @GeneratedValue
        private Integer id;
        private String vorname;
        private String nachname;
  ...
}
```

Listing 7

```
@Stateless
public class KundeService {

  @PersistenceContext
  EntityManager em;

  public void speichern(Kunde kunde) {
  em.persist(kunde);
  }

  public List<Kunde> alleKunden() {
    return em.createNamedQuery(“Kunde.alleKunden”,
          Kunde.class).getResultList();
  }
}
```

Listing 8

```
...
Assert.assertEquals(0, kundeService.alleKunden().size());
client.setValue(„vorname“, vorname);
client.setValue(„nachname“, nachname);
client.click(“speichern”);
Assert.assertEquals(1, kundeService.alleKunden().size());
...
```

Listing 9

möglich. Hier können nur JSF-Managed-Beans verwendet werden, obwohl dies nach Foren-Einträgen auch für CDI-Beans möglich sein sollte. Abschließend ist noch die sehr rudimentäre Dokumentation zu bemängeln.

Fazit

JSFUnit wird zurzeit intern auf Arquillian und ein annotationsbasiertes API umgestellt. Die von uns verwendete Version 2.0.0.Beta2 empfiehlt sich im Augenblick noch nicht für den produktiven Einsatz. Wir hoffen, dass es JBoss im finalen Release gelingen wird, das Qualitätsniveau früherer Releases zu erreichen.

Weiterführende Informationen

[1] http://jakarta.apache.org/cactus
[2] http://www.jboss.org/arquillian
[3] http://htmlunit.sourceforge.net
[4] Frederik Mortensen. Arquillian, Java aktuell 4/2011
[5] http://www.jboss.org/shrinkwrap
[6] Bernd Müller, JavaServer Faces 2.0, Hanser, 2010

Bernd Müller
bernd.mueller@ostfalia.de
Boris Wickner
bo.wickner@ostfalia.de

Bernd Müller ist Professor für Software-Technik an der Ostfalia. Er ist Autor des Buches „JavaServer Faces 2.0“ und Mitglied in der Expertengruppe des JSR 344 (JSF 2.2).

Boris Wickner ist Mitarbeiter an der Ostfalia im Bereich Infrastruktur für die Software-Ausbildung und Master-Student an der TU Braunschweig.

Vorschau

Java aktuell – Herbst 2012
Die nächste Ausgabe erscheint am 6. Juni 2012.
Falls Sie einen Artikel darin veröffentlichen möchten, schicken Sie bitte vorab Ihren Themenvorschlag an redaktion@ijug.eu

Redaktionsschluss ist am 4. April 2012

www.ijug.eu

JETZT ABO BESTELLEN

Sichern Sie sich 4 Ausgaben für 18 EUR

Für Oracle-Anwender und Interessierte gibt es das Java aktuell Abonnement auch mit zusätzlich sechs Ausgaben im Jahr der Fachzeitschrift *DOAG News* und vier Ausgaben im Jahr *Business News* zusammen für 75 EUR. Weitere Informationen unter www.doag.org/go/shop

FAXEN SIE DAS AUSGEFÜLLTE FORMULAR AN

0700 11 36 24 39

ODER BESTELLEN SIE ONLINE

go.ijug.eu/go/abo

Interessenverbund der Java User Groups e.V.
Tempelhofer Weg 64
12347 Berlin

Java aktuell

+++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN

Ja, ich bestelle das Abo Java aktuell – das IJUG-Magazin: 4 Ausgaben zu 18 EUR/Jahr

Ja, ich bestelle den kostenfreien Newsletter: Java aktuell – der iJUG-Newsletter

ANSCHRIFT

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

GGF. RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer

X

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

*Allgemeine Geschäftsbedingungen:

Zum Preis von 18 Euro (inkl. MwSt.) pro Kalenderjahr erhalten Sie vier Ausgaben der Zeitschrift "Java aktuell - das iJUG-Magazin" direkt nach Erscheinen per Post zugeschickt. Die Abonnementgebühr wird jeweils im Januar für ein Jahr fällig. Sie erhalten eine entsprechende Rechung. Abonnementverträge, die während eines Jahres beginnen, werden mit 4,90 Euro (inkl. MwSt.) je volles Quartal berechnet. Das Abonnement verlängert sich automatisch um ein weiteres Jahr, wenn es nicht bis zum 31. Oktober eines Jahres schriftlich gekündigt wird. Die Wiederrufsfrist beträgt 14 Tage ab Vertragserklärung in Textform ohne Angabe von Gründen.