→ 2., aktualisierte und erweiterte Auflage

Marcus Schießer · Martin Schmollinger

Workshop Java EE 7

Ein praktischer Einstieg in die
Java Enterprise Edition mit dem Web Profile

dpunkt.verlag

Marcus Schießer ist freiberuflicher Softwarearchitekt und Consultant aus Frankfurt. Seit seiner Diplomarbeit im Jahr 2002 verwendet er die Java Enterprise Edition, damals noch in der Version 1.3. Unter *www.javaee7.de* bietet er Schulungen und Beratungsdienstleistungen rund um Java EE 7 an.

Martin Schmollinger ist Professor für Informatik an der Hochschule Reutlingen. Er lehrt in den Studiengängen Wirtschaftsinformatik der Fakultät Informatik. Zu seinen Lehr- und Forschungsgebieten zählen Java EE und Business Process Management.

Marcus Schießer · Martin Schmollinger

Workshop Java EE 7

Ein praktischer Einstieg in die Java Enterprise Edition mit dem Web Profile

2., aktualisierte und erweiterte Auflage

dpunkt.verlag

Marcus Schießer
marcus.schiesser@javaee7.de

Martin Schmollinger
martin.schmollinger@javaee7.de

Lektorat: René Schönfeldt
Copy-Editing: Annette Schwarz, Ditzingen
Satz: Birgit Bäuerlein
Herstellung: Susanne Bröckelmann
Umschlaggestaltung: Helmut Kraus, www.exclam.de
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISBN:
Buch 978-3-86490-195-9
PDF 978-3-86491-595-6
ePub 978-3-86491-596-3

2., aktualisierte und erweiterte Auflage 2015

Wieblinger Weg 17
69123 Heidelberg

5 4 3 2 1 0

Vorwort

Für wen ist das Buch?

Das vorliegende Buch beinhaltet einen Workshop, der Ihnen eine Einführung in die *Java Enterprise Edition 7* (Java EE 7) bietet. Den Einstieg in dieses komplexe Thema schaffen wir über das bereits mit Java EE 6 eingeführte *Web Profile*, eine standardisierte Teilmenge der Java-EE-APIs speziell für Webapplikationen. Im Rahmen des Buches entwickeln wir Schritt für Schritt eine Webanwendung mit den Technologien des Web Profile. Zum Ende der Entwicklung kommen darüber hinaus APIs zum Einsatz, die nicht im Web Profile enthalten sind, aber eine hohe Praxisrelevanz für professionelle Anwendungen haben.

Auch wenn Sie im Buch viel über die Neuerungen bei Java EE 7 im Vergleich zur Vorversion erfahren, so ist es nicht vorrangig für erfahrene Java-EE-Entwickler gedacht, die ihr Wissen auf den neusten Stand bringen möchten. Zielgruppe sind vielmehr Einsteiger, die eine ganzheitliche Einführung in Java EE in der aktuellen Version 7 suchen.

Damit das Vorhaben gelingt, setzt das Buch einige allgemeine und einige speziellere Kenntnisse voraus.

Zu den allgemeinen Voraussetzungen zählen wir:

- gute Kenntnisse in der objektorientierten Programmierung mit Java
- Grundkenntnisse der Java-SE-API
- grundlegendes Wissen über Webtechnologien (HTTP, HTML, Webserver)
- grundlegendes Wissen über Datenbanken und XML
- Grundkenntnisse zu verteilten Systemen
- grundlegende Betriebssystemkenntnisse

Zu den speziellen Voraussetzungen gehören:

- Grundkenntnisse zu Java-Annotationen
- Grundkenntnisse einer integrierten Entwicklungsumgebung für Java
- Kenntnisse über die Kommandozeile des verwendeten Betriebssystems
- Grundlagen von SQL

Falls Sie die speziellen Voraussetzungen nicht mitbringen, ist dies für die Lektüre unproblematisch, denn Sie können das entsprechende Wissen recht schnell parallel zum Workshop nachholen. Die allgemeinen Voraussetzungen sind dagegen etwas umfassender und in der Regel nur über einen längeren Zeitraum zu erlernen.

Aus unserer Sicht spricht das Buch deshalb besonders die folgenden Gruppen an (ohne Anspruch auf Vollständigkeit):

- Webentwickler, die Java-Kenntnisse haben, aber bisher hauptsächlich mit alternativen Webprogrammiersprachen (z. B. PHP, ASP.NET) gearbeitet haben
- Java-Entwickler, die bisher für andere Plattformen entwickelt haben (z. B. Desktop oder Mobilgeräte)
- (*-)Informatik/Wirtschaftsinformatik-Studenten ab den höheren Bachelor-Semestern

Bei Java EE handelt es sich um eine mächtige Programmierplattform, in der sich Anfänger leicht verirren können. Ziel des Buches ist es, die Komplexität wo immer möglich abzuschütteln und dem Einsteiger den Blick auf das Wesentliche zu verschaffen. Damit soll der Start erleichtert und die Motivation hoch gehalten werden.

Sollten dennoch Java-EE-Experten dieses Buch lesen, so bitten wir bereits jetzt um Verständnis, dass die Vereinfachung von Sachverhalten zwangsläufig auch dazu führt, dass die Vollständigkeit nicht immer gegeben ist. Dies nehmen wir aus didaktischen Gründen bewusst in Kauf. Für die vielen Feinheiten und Alternativwege bleibt – nach einem guten Einstieg in das Thema – immer noch Zeit. Das Buch liefert außerdem die Basis, sich schnell in neue Aspekte der Java EE einarbeiten zu können.

Zur zweiten Auflage

Java EE liegt seit dem zweiten Quartal 2013 in der Version 7 vor. Oracles Referenzimplementierung ist der Glassfish-Applikationsserver in der Version 4. Die Zeit zwischen der Veröffentlichung einer finalen Java-EE-Spezifikation und dem Release marktrelevanter Java-EE-Applikationsserver kann, nicht nur für Autoren, sehr lang sein. Bei der ersten Auflage war Glassfish 4 der einzige Java-EE-7-Applikationsserver. Inzwischen gibt es aber mit WildFly 8 (dem Nachfolger von JBoss AS 7) eine weitere prominente Implementierung. Nach unserer Meinung hat er für Einsteiger aufgrund der Verbreitung des JBoss AS 7 die höchste Praxisrelevanz, weshalb er bei einem Großteil der Entwicklung im Buch zum Einsatz kommt. Darüber hinaus setzen wir in der letzten Iteration der Softwareentwicklung auch den Glassfish-4-Applikationsserver ein, um die Werkzeugkompetenz des Einsteigers zu verbreitern. Abgerundet werden die Ausführungsumgebungen durch die Verwendung des PaaS-Angebots openshift von Red Hat, das es möglich macht, Java-EE-7-Anwendungen in die Cloud zu deployen.

Die neue Version 7 des Standards enthält viele Neuerungen in den maßgeblichen Teilspezifikationen (JSF, CDI, JPA, EJB), von denen einige jedoch für Einsteiger nicht so gravierend sind. Das Buch hat deshalb auch in der zweiten Auflage nicht den Anspruch, alle diese Neuerungen praktisch zu verwenden, sondern möchte einen fundierten, ganzheitlichen Einstieg in das Thema Java EE bieten. Dennoch wenden wir die wichtigsten und spannendsten Innovationen des neuen Standards wie z. B. die neue REST-API und WebSockets mit HTML5 an. Zusätzlich ist am Ende einer jeden Iteration ein Theorieteil angehängt, der nicht praktisch verwendete Themen und Neuerungen von Java EE 7 anspricht.

Neu in der zweiten Auflage sind die praktische Anwendung einiger Features von Java EE 7 im Bereich der JavaServer Faces, wie z. B. *Resource Library Contracts* und die HTML-5-Unterstützung. Darüber hinaus haben wir in der zweiten Auflage zusätzlich die Internationalisierung von Java-EE-Anwendung aufgenommen und ein neues Kapitel über funktionale Tests hinzugefügt.

Wir finden das Konzept des Buches für Java-EE-Einsteiger ideal, da der Leser einen breiten, aber geführten Einstieg in Java EE erhält, die neuen Möglichkeiten von Java EE 7 kennenlernt, die Höhepunkte daraus selbst ausprobiert und mit den am weitesten verbreiteten Open-Source-Produkten arbeiten kann.

Zusatzinformationen zum Buch

Sie finden den kompletten Sourcecode der im Buch entwickelten Anwendungen My-Aktion und My-Aktion-Monitor auf der bei Entwicklern beliebten Kooperationsplattform GitHub:

```
https://github.com/marcusschiesser/my-aktion-2nd
```

In Abschnitt A.2 des Anhangs finden Sie Informationen, wie Sie Git verwenden müssen, um Zugriff auf den Sourcecode zu bekommen.

Darüber hinaus gibt es eine Website zum Buch, auf der Sie Ergänzungen zum Buch und zum Einsatz von Java-EE-Technologien erhalten.

```
www.javaee7.de/tutorial
```

Aufbau des Buches

In Kapitel 1 erhalten Sie eine kurze Übersicht zu Java EE. Dabei wird eine Zielarchitektur entwickelt, die uns durch den ganzen Workshop begleitet.

In Kapitel 2 widmen wir uns zunächst der Entwicklungs- und Ausführungsumgebung für unser Softwareprojekt. Sie erfahren, wie Sie WildFly installieren und ein erstes Java-EE-Projekt mithilfe von Maven, einem weitverbreiteten Werkzeug zur Verwaltung von Projektkonfigurationen, erstellen und auf WildFly deployen.

Der Anhang enthält einen zusätzlichen Abschnitt A.1, der dem interessierten Leser parallel zu Kapitel 1 und 2 ein schnelleres Verständnis von Java-EE-Anwendungen auf Quellcodeebene vermitteln soll.

Das Kapitel 3 beschreibt die zu erstellende Applikation. Dabei werden die fachlichen Anwendungsfälle (engl. *Use Cases*) und ihre Abläufe, die Fachklassen und die zugehörigen grafischen Oberflächen im Zentrum der Beschreibung stehen.

Die Entwicklung der Software soll in mehreren Iterationen durchgeführt werden. Jedes Kapitel realisiert eine Iteration und schließt mit einer Reihe von Aufgaben. Die erste Iteration beginnt in Kapitel 4 mit der Weboberfläche. Nach der in Kapitel 8 beschriebenen Iteration ist die Anwendung voll funktionsfähig. In Kapitel 9 wird ein weiterer Applikationsserver, Glassfish 4, verwendet, um den letzten Anwendungsfall zu realisieren.

Kapitel 10 widmet sich dem Thema »Java EE 7 und die Cloud«. Zwar soll eine Standardisierung des Themas erst mit Java EE 8 kommen, aber bereits heute gibt es einige proprietäre Cloud-Angebote für Java-EE-Anwendungen. Wir zeigen das Deployment der entwickelten Anwendung My-Aktion für den Cloud-Anbieter *OpenShift*. Letzterer bietet WildFly-Instanzen in der Cloud an und ermöglicht so die Verbreitung von Java-EE-Applikationen im Web ohne den Betrieb eines eigenen Servers.

Danksagung

Wir möchten uns ganz herzlich bei allen bedanken, die die erste Auflage des Buches durchgearbeitet und manchmal auch durchlitten haben. Die vielen Rückmeldungen mit Lob, Kritik, Verbesserungsmöglichkeiten, aber auch Fehlern haben uns zu der vorliegenden zweiten Auflage motiviert. Insbesondere möchten wir den folgenden Personen für ihre Hinweise danken: Michael Behr, Vanesco Böhm, Muammer Cakir, Nils Faupel, Yangyang Gao, Patrick Hargens, Stefan Höfler, Sebastian Krome, Jens Krückel, Fetullah Misir, Alex Petri, Peter Pfeiffer, Georg Ruzicka und Markus Wetzka.

Inhaltsübersicht

1	Ein Einstieg mit Profil	1
2	Aufsetzen der Entwicklungsumgebung	19
3	Fachlichkeit der Beispielanwendungen »My-Aktion« und »My-Aktion-Monitor«	39
4	Iteration Nr. 1 – JavaServer Faces	55
5	Iteration Nr. 2 – Funktionale Tests	147
6	Iteration Nr. 3 – CDI	167
7	Iteration Nr. 4 – Java Persistence API	197
8	Iteration Nr. 4 – Enterprise JavaBeans	243
9	Iteration Nr. 5 – Kommunikation und HTML5	291
10	Java EE 7 und die Cloud	355
A	Anhang	371
	Index	383

Inhaltsverzeichnis

1 Ein Einstieg mit Profil 1

1.1 Java EE 7 – der Standard für Enterprise Java 1
1.1.1 Struktur einer Enterprise-Java-Anwendung 1
1.1.2 Die Java Enterprise Edition (Java EE) 2
1.1.3 Anatomie einer Java-EE-Anwendung 3
1.2 Die Komponentenarchitektur von Java EE 7 4
1.2.1 Die Komponenten der Webschicht 5
1.2.2 CDI – ein Komponententyp mit viel Potenzial 8
1.2.3 Enterprise JavaBeans 9
1.2.4 Java Persistence API (JPA) 10
1.2.5 Die Java-EE-Profile 11
1.2.6 Und ab geht's auf den Applikationsserver! 13
1.3 Die Zielarchitektur 15
1.4 Wie geht es jetzt weiter? 16
1.5 Weiterführende Literatur 18

2 Aufsetzen der Entwicklungsumgebung 19

2.1 Werkzeuge und Server für Java EE 19
2.2 Installation der nötigen Softwarepakete 20
2.2.1 Installation von Java SE 8 21
2.2.2 Installation von Maven 3 22
2.2.3 Installation von WildFly 8.1.0 23
2.3 Vom Projekt zur ausgeführten Webanwendung 24
2.3.1 Anlegen eines leeren Java-EE-7-Webprojektes mit Maven . 24
2.3.2 Der erste Build und das erste Deployment 28
2.3.3 Test des Deployments 30

2.4 Eclipse Luna – Mehr als ein Texteditor 31
2.4.1 Installation von Eclipse 31
2.4.2 Eclipse-Plug-in m2e 32
2.4.3 Import von Maven-Projekten 33
2.4.4 Das Deployment 35
2.5 Weitere Werkzeuge des Workshops 37
2.6 Weiterführende Literatur 37

3 Fachlichkeit der Beispielanwendungen »My-Aktion« und »My-Aktion-Monitor« 39

3.1 Einleitung 39
3.2 Übersicht der Anwendungsfälle 39
3.3 Fachklassen 41
3.4 Anwendungsfälle 43
3.4.1 Anmeldung notwendig 44
3.4.2 Aktionen anzeigen und bearbeiten 44
3.4.3 Spendenformular bearbeiten 45
3.4.4 Spendenliste anzeigen 46
3.4.5 Aktion bearbeiten 47
3.4.6 Aktion löschen 47
3.4.7 Neue Aktion anlegen 48
3.4.8 Aktionsdaten editieren 48
3.4.9 Geld spenden 50
3.4.10 Reduzierte Spendenliste anzeigen 52
3.5 Weiterführende Literatur 53

4 Iteration Nr. 1 – JavaServer Faces 55

4.1 Einleitung 55
4.2 Fachklassen als JavaBeans 56
4.2.1 Konto 56
4.2.2 Spende 57
4.2.3 Aktion 58
4.3 Internationalisierung 60
4.4 Das Vorlagensystem 63
4.4.1 Templating mit Facelets 63
4.4.2 Ein Template für My-Aktion 64
4.4.3 Resource Library Contracts 67

4.5 Der Anwendungsfall »Aktionen anzeigen und bearbeiten« 70
4.5.1 Die Backing Beans 70
4.5.2 JSF-Technologien für den ersten Anwendungsfall 75
4.5.3 Unsere erste View! 81
4.5.4 Der Start – ein erster Meilenstein 83
4.6 Weitere JSF- und Java-EE-Technologien 87
4.6.1 @Inject-Annotation 87
4.6.2 Texteingabe mit Validierung, Konvertierung und Fehlermeldungen 88
4.6.3 Das Rasterlayout von UI-Komponenten 95
4.6.4 Auswahlkomponenten 96
4.6.5 Die GET-Parameterverarbeitung in einer View 97
4.6.6 Ajax – Etwas mehr Dynamik gefällig? 98
4.6.7 HTML5 Friendly Markup 100
4.7 Die Implementierung der restlichen Anwendungsfälle 101
4.7.1 Aktion bearbeiten und Aktion neu anlegen 101
4.7.2 Spendenliste anzeigen 110
4.7.3 Spendenformular bearbeiten 114
4.7.4 Geld spenden 120
4.8 Neue Funktionalitäten in Java EE 7 125
4.8.1 Faces Flows 125
4.8.2 Stateless Views 126
4.9 Exkurs: PrimeFaces 127
4.9.1 Installation über Maven 128
4.9.2 Neue Farbauswahl für die Bearbeitung des Spendenformulars 128
4.9.3 Tabs für das Anlegen und Editieren von Aktionsdaten ... 133
4.9.4 Der Bestätigungsdialog für den Anwendungsfall »Aktion löschen« 137
4.10 Aufgaben 142
4.10.1 Verwendung von PrimeFaces-Komponenten für alle Views (Pflichtaufgabe) 142
4.10.2 Lokalisierung der Schaltfläche »Update URL« 143
4.10.3 Durchgängige Verwendung von Inline-Labels 144
4.10.4 Auslagerung der Eigenschaften des Spendenformulars ... 144
4.11 Weiterführende Literatur 145

5 Iteration Nr. 2 – Funktionale Tests 147

5.1 Einleitung 147
5.2 Testframeworks Arquillian und Graphene konfigurieren 148
5.3 Wiederwendbare Klassen erstellen 150
5.3.1 Testdaten für Aktionen und Spenden bereitstellen 151
5.3.2 Anwendung My-Aktion mit Arquillian deployen 152
5.3.3 Webseiten mit Graphene steuern 153
5.4 Anwendungsfälle testen 156
5.4.1 Den Anwendungsfall »Neue Aktion anlegen« testen 156
5.4.2 Den Anwendungsfall »Geld spenden« testen 161
5.5 Aufgaben 165
5.5.1 SetupDatabase erweitern 166
5.5.2 Testfall für bisher gespendeten Betrag hinzufügen 166
5.5.3 Weitere Testfälle erstellen 166
5.6 Literaturverzeichnis 166

6 Iteration Nr. 3 – CDI 167

6.1 Warum CDI? 167
6.2 Der Laufzeit-Container 168
6.2.1 Der Sichtbarkeitsbereich (Scope) einer Bean 169
6.2.1.1 RequestScope 169
6.2.1.2 SessionScope 170
6.2.1.3 ViewScope 170
6.2.1.4 DependentScope 171
6.2.1.5 Setzen der Scopes für unsere Anwendung 171
6.2.2 Beans referenzieren über Dependency Injection 172
6.2.3 Der Lebenszyklus 174
6.2.3.1 Lebenszyklusmethode PostConstruct 174
6.2.3.2 Lebenszyklusmethode PreDestroy 175
6.2.4 Beliebige Klassen als Beans mit Producer-Methoden 175
6.3 Anwendungsweite Nachrichten 180
6.3.1 Events senden und empfangen 180
6.3.2 Nachrichten gleichen Typs mit Qualifiern unterscheiden . . 181
6.4 Services 184
6.4.1 Die Mock-Methode in eine Serviceklasse auslagern 184
6.4.2 Die verwendete Serviceimplementierung mit Qualifiern auswählen 187

6.5 Was noch nicht behandelt wurde 190
6.5.1 ApplicationScope und ConversationScope 190
6.5.2 Der FlowScope 191
6.5.3 Interzeptoren und Dekoratoren 191
6.5.4 Stereotypes 192
6.5.5 Der BeanManager 192
6.5.6 Erweiterungen 193
6.5.7 Disposer-Methoden 193
6.6 Aufgaben 194
6.6.1 Fachliches und technisches Log unterscheiden 194
6.6.2 Nachricht zur Aktualisierung einer Aktion hinzufügen . . . 194
6.7 Weiterführende Literatur 195

7 Iteration Nr. 4 – Java Persistence API 197

7.1 Einleitung 197
7.2 Entitäten annotieren 198
7.2.1 Als Entität markieren und Primärschlüssel festlegen 198
7.2.2 Relationen der Entitäten festlegen 200
7.2.3 Abhängige Entitäten einbinden 202
7.3 Services um Datenzugriffe erweitern 203
7.3.1 EntityManager einbinden 203
7.3.2 Datenbankabfragen hinzufügen 204
7.3.3 CRUD-Operationen ausführen 206
7.3.4 CampaignListProducer um Servicedelegation erweitern . . 208
7.4 Daten transaktional speichern 210
7.4.1 Datenquelle konfigurieren 210
7.4.2 Services um Transaktionen erweitern 213
7.5 Spenden persistieren und anzeigen 214
7.5.1 Service zur Bearbeitung von Spenden erstellen 214
7.5.2 Anwendungsfall *Geld spenden* finalisieren 216
7.5.3 Den bisher gespendeten Betrag berechnen 217
7.5.4 Spenden in der Spendenliste anzeigen 219
7.6 Eingaben über Bean Validation überprüfen 221
7.6.1 Entitäten mit Bedingungen annotieren 222
7.6.2 Validatoren aus Views entfernen 226
7.6.3 Zugeordnete Spenden beim Löschen einer Aktion automatisch entfernen 227
7.7 Eigene In-Memory-Datasource für Tests benutzen 229

7.8 Was noch nicht behandelt wurde 231
7.8.1 CriteriaQuery als Alternative zur NamedQuery 231
7.8.2 LazyInitialisationException über FetchType vermeiden . . . 233
7.8.3 Object Locking 234
7.8.3.1 Optimistische Sperre 234
7.8.3.2 Pessimistischen Sperre 235
7.8.4 Lebenszyklusmethoden bei Zustandsänderungen 236
7.9 Neue Funktionalitäten in Java EE 7 238
7.9.1 Bean Validation auf Methodenebene 238
7.9.2 Erweiterte CDI-Unterstützung 238
7.9.3 Aufruf gespeicherter Prozeduren und Funktionen 239
7.9.4 Schreibende Massenoperationen mit der Criteria API 240
7.10 Zusammenfassung 240
7.11 Aufgaben 241
7.11.1 JPQL-Abfragen mit Criteria API ersetzen 241
7.11.2 Zeitstempel der Aktualisierung hinzufügen 241
7.11.3 Minimalen Spendenbetrag der Aktion überprüfen 242
7.11.4 Organisator über Spendenziel informieren 242
7.12 Weiterführende Literatur 242

8 Iteration Nr. 4 – Enterprise JavaBeans 243

8.1 Einleitung 243
8.2 Sicherheit 244
8.2.1 Organisatoren speichern 244
8.2.2 Security-Domain für WildFly anlegen 250
8.2.3 Benutzeroberfläche absichern 252
8.2.4 Eigene Aktionen für jeden Organisator 254
8.2.5 Services über Annotationen absichern 256
8.2.6 Mehr Sicherheit durch SSL 259
8.2.7 Eigenen Anmeldebildschirm festlegen 261
8.2.8 Automatisches Login für die Tests 264
8.3 Transaktionssteuerung 266
8.3.1 Transaktionssteuerung durch den Container 266
8.3.2 Transaktionssteuerung über die Bean 269
8.3.3 Transaktionssteuerung über Interzeptoren 270
8.4 Zeitgesteuerte Abläufe realisieren 274
8.4.1 Spenden überweisen 274
8.4.2 Hintergrundjob zeitlich gesteuert aufrufen 275

8.5 Vergleich EJBs und CDI 277
8.5.1 Unterschiedlicher Lebenszyklus 277
8.5.2 Vorteile der jeweiligen Technologie 279
8.6 Was nicht behandelt wurde 280
8.6.1 Stateful Session Beans 280
8.6.2 Entfernter Methodenaufruf 282
8.6.3 Nebenläufigkeit 283
8.6.4 Asynchroner Methodenaufruf 285
8.6.5 Message-Driven Beans 286
8.6.6 Entity Beans 287
8.7 Aufgaben 287
8.7.1 Registrierungsformular für Organisatoren erstellen 287
8.7.2 Services auf CDI umstellen 288
8.7.3 Ausführungszeit einer Methode messen 288
8.8 Weiterführende Literatur 289

9 Iteration Nr. 5 – Kommunikation und HTML5 291

9.1 Einleitung 291
9.2 Neue Technologien 291
9.2.1 Webservices 291
9.2.2 WebSockets 293
9.3 Kommunikationsarchitektur 293
9.4 REST-Webservice-Schnittstelle für My-Aktion 295
9.4.1 REST-Webservice für das Management von Aktionen . . . 298
9.4.1.1 Liste aller Aktionen eines Organisators abfragen 299
9.4.1.2 Aktion löschen 301
9.4.1.3 Neue Aktion hinzufügen 303
9.4.1.4 Aktion aktualisieren 305
9.4.2 REST-Webservice zur Abfrage und Ausführung von Spenden 306
9.4.2.1 Spendenliste einer Aktion abfragen 307
9.4.2.2 Geld spenden 308
9.5 Die Webapplikation My-Aktion-Monitor 314
9.5.1 Anwendungsserver Glassfish installieren 315
9.5.2 Bestandteile der Anwendung My-Aktion-Monitor 320
9.5.2.1 Spendenliste einer Aktion anzeigen 321
9.5.2.2 SOAP-Webservice für My-Aktion-Monitor erstellen 341
9.5.2.3 SOAP-Webservice testen 343
9.5.2.4 Spendenliste über SOAP aktualisieren 346

9.6 Nicht behandelt 348
9.6.1 JMS 348
9.6.2 Java Mail 349
9.6.3 JMX 350
9.7 Aufgaben 352
9.7.1 Erweiterung der REST-Services um XML 352
9.7.2 Eigenen REST-Client erstellen 352
9.8 Literaturverzeichnis 353

10 Java EE 7 und die Cloud 355

10.1 Einleitung 355
10.2 Was ist eine Cloud? 355
10.2.1 Standardisierungsbedarf für Java-EE-Cloud-Szenarien . . . 357
10.3 My-Aktion in OpenShift installieren 358
10.3.1 Zugang für OpenShift anlegen 358
10.3.2 Notwendige Werkzeuge installieren 358
10.3.3 Neue Anwendung in OpenShift anlegen 361
10.3.4 My-Aktion in OpenShift deployen 362
10.3.5 Anpassungen für OpenShift vornehmen 367
10.4 Literaturverzeichnis 370

A Anhang 371

A.1 Vom Servlet zu JSF mit CDI und EJB 371
A.1.1 Ein paar Hinweise vorweg 371
A.1.2 Java Servlets 372
A.1.3 JavaServer Pages und JavaBeans 373
A.1.4 JavaServer Faces und JSF Managed Beans 375
A.1.5 JavaServer Faces und CDI 376
A.1.6 JSF, CDI und Enterprise JavaBeans 377
A.2 My-Aktion und My-Aktion-Monitor auf GitHub 378
A.2.1 Git installieren 378
A.2.2 Git-Repository des Workshops verwenden 380
A.2.3 Literatur zu Git 381

Index 383

1 Ein Einstieg mit Profil

1.1 Java EE 7 – der Standard für Enterprise Java

Oracle wirbt für Java mit dem Hinweis: »3 Milliarden Geräte verwenden Java.« Offensichtlich haben sie recht damit: Auf nahezu jedem PC ist eine Java-Laufzeitumgebung (*Java Runtime Environment*) installiert. Java ist zwar überall präsent – es stellt sich allerdings die Frage: Ist es wirklich in jedem Fall die dominierende Plattform?

Dies kann von Fall zu Fall (Desktop, Server, mobile Geräte, Blu-Ray-Player etc.) sicher kontrovers diskutiert werden. Bei kritischen Geschäftsanwendungen hat sich Java jedoch zweifelsfrei durchgesetzt.

Sicherlich hat zu diesem Erfolg beigetragen, dass Java von Beginn an auf zukunftsfähige Konzepte wie Objektorientierung und Plattformunabhängigkeit setzte, die Wartung und Verteilung einer Anwendung vereinfachten. Außerdem ließen sich bereits in frühen Versionen komplexe Funktionalitäten umsetzen, da hierzu umfangreiche Klassenbibliotheken mitgeliefert wurden, u.a. für Threads, entfernte Methodenaufrufe, Datenstrukturen, grafische Benutzeroberflächen und I/O.

Diese Basis wurde schnell um weitere Aspekte von Geschäftsanwendungen wie z.B. Transaktionsmanagement, Sicherheitsmechanismen und Webtechnologien erweitert und zeichnete damit den Weg zur Standardisierung und Etablierung der Java-Plattform im Feld des Enterprise Computing vor – auch wenn der Weg dahin in den letzten Jahren nicht immer geradlinig war.

1.1.1 Struktur einer Enterprise-Java-Anwendung

Im Unterschied zu gewöhnlichen Java-Programmen werden Enterprise-Java-Anwendungen für verteilte, mehrschichtige Architekturen entwickelt. Das heißt, Anwendungsteile werden in der Regel auf unterschiedlichen Rechnern ausgeführt. Java-EE-Anwendungen für diese Architekturen teilen sich logisch in bis zu vier Schichten auf: *Client-, Web-, Business- und Persistenzschicht*. Die Clientschicht enthält die Teile der Anwendung, die der Benutzer zur Interaktion mit der Anwendung benötigt. Bei Webanwendungen ist der Client ein Browser. Die Realisierung

der Ablaufsteuerung und die Erzeugung der Webseiten ist Aufgabe der Webschicht. Dadurch kann der Nutzer indirekt die Geschäftslogik der Businessschicht nutzen. Wird keine Webanwendung realisiert, so wird die Webschicht nicht benötigt. Stattdessen kann ein *Application-Client*, z.B. eine Desktop-Java-Anwendung mit grafischer Oberfläche, verwendet werden. Ein Application-Client realisiert die Ablaufsteuerung der Anwendung selbst und verwendet dabei die bereitgestellte Geschäftslogik durch direkte Benutzung der Businessschicht oder indirekt durch den Aufruf von Webservices, die die Geschäftslogik der Businessschicht kapseln. Die zur Realisierung der Geschäftslogik in der Businessschicht zu verarbeitenden Daten werden in der Persistenzschicht dauerhaft gespeichert. In der Regel kommen dabei Datenbanksysteme inklusive deren Abstraktionen zum Einsatz, sodass sie einfach in die Businessschicht eingebunden werden können. Die Daten können aber auch in anderen, externen Systemen abgelegt werden, die sich auf verschiedene Weise (z.B. durch Nachrichtenkommunikation) einbinden lassen.

Enterprise-Java-Anwendungen werden über einen Java-Applikationsserver bereitgestellt und verwaltet. Neben entsprechenden Laufzeitumgebungen für die verschiedenen Teile der Anwendung, auch *Container* genannt, stellt er auch Querschnittsdienste bereit, die von allen Anwendungen verwendet werden können. Dazu zählt z.B. das Transaktionsmanagement, Authentifizierung, Autorisierung, Namens- und Verzeichnisdienste oder auch der standardisierte Zugriff auf Datenbanken, verteilte Warteschlangen (engl. *message queues*) und andere externe Systeme.

Wichtige Eigenschaften von Geschäftsanwendungen wie Synchronisation, Lastverteilung und Verfügbarkeit werden ebenfalls vom Applikationsserver garantiert und damit von der Anwendungslogik entkoppelt. Dadurch erleichtert der Applikationsserver die Entwicklung von Geschäftsanwendungen, da der Programmierer sich voll auf die funktionale Gestaltung der eigentlichen Anwendung konzentrieren kann.

Eine Anwendung ist aus Komponenten aufgebaut, die einen klar definierten Zweck innerhalb der Anwendung erfüllen und einer Schicht der Anwendung zugeordnet sind. Ziel des Komponentenmodells sind klare Strukturen und damit einhergehend eine höhere Transparenz, bessere Verständlichkeit sowie eine einfachere Wartbarkeit und Erweiterbarkeit der Anwendung.

1.1.2 Die Java Enterprise Edition (Java EE)

Der Java-Standard für mehrschichtige Geschäftsanwendungen firmiert unter dem Namen *Java Platform Enterprise Edition* 7 (Java EE 7) und wird durch die Spezifikation JSR-342 (`http://jcp.org/en/jsr/detail?id=342`) des *Java Community Process* (JCP) definiert (DeMichiel & Shannon, 2013).

Der Standard basiert auf der *Java Platform Standard Edition* (Java SE) sowie zusätzlichen APIs, die entweder das Komponentenmodell realisieren (z.B. Enter-

prise JavaBeans) oder spezielle Aspekte einer Geschäftsanwendung adressieren (z.B. Transaktionen mit der Java Transaction API). Diese APIs werden (wie alle offiziellen Java-APIs) durch den JCP standardisiert und unter einer eigenen JSR-Nummer (*Java Specification Request*) veröffentlicht.

Es handelt sich um einen wichtigen Aspekt, da dadurch sowohl Herstellern von Applikationsservern oder Bibliotheken als auch Entwicklern eine Vorgabe für ihre Arbeit gegeben wird. Die Hersteller wissen dadurch genau, was von ihrem Produkt verlangt wird, und die Entwickler wissen, was sie bei der Programmierung ihrer Applikation voraussetzen können. Auf diese Weise können Applikationsserver auch für Java-EE-Versionen zertifiziert werden. Alle Spezifikationen sind öffentlich über das Internet (`http://jcp.org`) zugänglich.

Im Verlauf der Geschichte des Java-EE-Standards gab es diverse Hochs und Tiefs. Zu Beginn des neuen Jahrtausends wich die anfängliche Begeisterung schnell der Ernüchterung, da der Standard (damals kurz J2EE genannt) selbst zu umständlich und komplex für den Programmierer war.

Das Resultat war die Entwicklung zahlreicher (erfolgreicher) Frameworks, die quasi in Konkurrenz zu Teilen bzw. zum Standard im Ganzen entstanden, etwa Spring, Seam, Hibernate oder Struts. Dies führte natürlich nicht zur Vereinfachung der Java-Landschaft in den Unternehmen, war aber eine logische Konsequenz, die aus der Praxis heraus resultierte.

Auch heute gilt deshalb noch, dass nicht jede Enterprise-Java-Anwendung auf Java EE basiert. Letztendlich stand in dieser Zeit nicht mehr und nicht weniger als die Existenz des Java-EE-Standards auf dem Spiel. Am Scheideweg stehend wurde 2006 mit Java EE 5 (Motto: »Ease of Development«) die Wende eingeleitet, indem ein Weg eingeschlagen wurde, der eine Vereinfachung des Standards zum Ziel hatte und damit die Arbeit des Entwicklers wieder in den Mittelpunkt stellte.

In diesem Sinne wurden mit Java EE 6 (2009) und aktuell mit Java EE 7 (2013) viele Ideen und Best Practices aus den diversen Java-Frameworks einvernehmlich übernommen. Der Standard wurde dadurch wieder konkurrenzfähig und zukunftsweisend.

1.1.3 Anatomie einer Java-EE-Anwendung

Java-EE-Anwendungen unterscheiden sich von gewöhnlichen Java-Programmen. Zwar werden sie größtenteils ebenfalls mit der Programmiersprache Java erstellt, der Programmierer orientiert sich jedoch an einem durch die Spezifikation vorgegebenen Komponentenmodell.

Java EE definiert Webkomponenten, die innerhalb von Java-EE-Webanwendungen verantwortlich sind für die Generierung von Webseiten für den Browser. Für die Realisierung der Komponenten der Webschicht stellt der Standard die Technologien *Java Servlets*, *JavaServer Pages* (JSP) und *JavaServer Faces* (JSF) zur Verfügung.

Des Weiteren definiert Java EE die Komponententechnologie *Enterprise JavaBeans* (EJBs) für die Businessschicht. Mit den EJBs wird die Geschäftslogik realisiert. Sie bringen von Haus aus transaktionales Verhalten und Sicherheitsmechanismen mit. Eine neuere Technologie unter dem Namen *Contexts and Dependency Injection* (CDI) ermöglicht mit seinen Eigenschaften die Konstruktion einer sehr flexiblen Softwarearchitektur für die Anwendung.

Eine Komponente kann aus unterschiedlichen technischen Artefakten (z.B. XML-Dateien, Schnittstellen oder Klassen) bestehen. Für jeden Komponententyp gibt es einen eigenen Container innerhalb des Applikationsservers, der verantwortlich für die Überwachung des Lebenszyklus der Komponenten ist. Abbildung 1–1 veranschaulicht die Situation. Im Folgenden gehen wir genauer auf die skizzierte Komponentenarchitektur der Java EE ein.

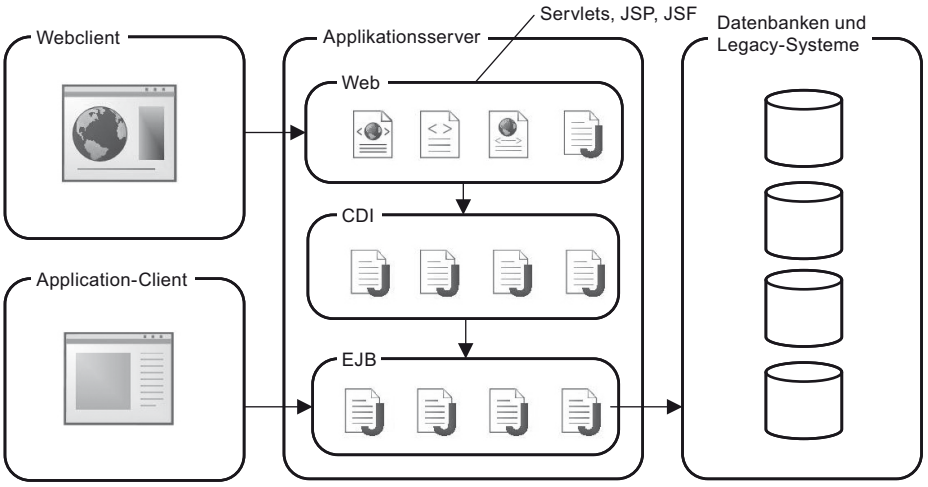

Abb. 1–1 *Skizze der Ausführungsumgebung von Java-EE-Anwendungen*

1.2 Die Komponentenarchitektur von Java EE 7

Hinweis

Leser, die bereits an dieser Stelle etwas mehr zur Programmierung der Komponenten erfahren wollen, können parallel oder im Anschluss den Abschnitt A.1 im Anhang lesen. Dort werden die Erläuterungen der Komponententechnologien durch kleine Programmcodes unterstützt, die zwar nicht die komplette Mächtigkeit der einzelnen Technologien erklären, aber dennoch zu einem besseren Verständnis beitragen können. Nach der Bearbeitung von Kapitel 2 und der Einrichtung der Entwicklungsumgebung können Sie die Beispiele natürlich auch ausprobieren.

1.2.1 Die Komponenten der Webschicht

Die Technologie der *Java Servlets* war die erste Java-API zur Realisierung von Webclients. Ein Servlet ist eine Java-Klasse, die innerhalb des Applikationsservers Anfrage-Antwort-Protokolle, insbesondere HTTP, unterstützt.

Die Erzeugung von Webseiten durch Java Servlets führt jedoch leicht zu einer engen Verzahnung von Darstellung und Logik. Neuere Webtechnologien basieren konzeptionell auf dem *Model-View-Controller-Muster* (MVC-Muster). Zwar ist es prinzipiell auch mit Java Servlets möglich, das MVC-Muster zu realisieren, jedoch ist eine saubere Trennung von Darstellung (View) und Logik (Controller) nicht direkt gegeben. Java Servlets werden daher heute nicht mehr direkt zur Erzeugung von Webseiten eingesetzt. Sie dienen in Webframeworks jedoch häufig als Mittler zwischen den anfragenden Webclients und den für die Erzeugung der Antwort zuständigen Komponenten.

Model-View-Controller (MVC)

Das MVC ist ein Muster, das vorgibt, wie Darstellung, Logik und Daten in einer Applikation getrennt werden sollen. Ziel dieser Trennung ist die Verbesserung der Programmstruktur und damit die Wartbarkeit, Erweiterbarkeit und Wiederverwendbarkeit des Codes.

Das Modell kapselt die Daten und enthält je nach MVC-Ausprägung ggf. auch die fachliche Logik. Die View visualisiert das Modell und der Controller realisiert die Anwendungssteuerung. Der Controller reagiert auf Benutzerinteraktionen innerhalb der View und aktualisiert ggf. die Daten am Modell. Die View wiederum passt sich je nach Ausprägung des MVC entweder automatisch an das veränderte Modell an oder wird durch den Controller über die Änderungen informiert.

Im Laufe der Zeit wurden unter Einbezug der Java-Servlet-Technologie Webkomponenten auf einem höheren Abstraktionsniveau geschaffen. Der Java-EE-Standard kennt hier die JavaServer Pages (JSP) und die JavaServer Faces (JSF).

Während ein Java Servlet eine Java-Klasse ist, bestehen JSP- und JSF-Komponenten zum einen aus einer textbasierten Seitenbeschreibung der View und zum anderen aus Klassen, die im Hintergrund die Controller-Logik realisieren. Die Beschreibung der View geschieht mit einer Seitenbeschreibungssprache (engl. *View Declaration Language, kurz VDL*), die je nach Technologie auf HTML bzw. XHTML basiert.

Modelle werden als Klassen realisiert. Da sie aber in allen Schichten der Anwendung benötigt werden, zählen sie nicht direkt zu den Webkomponenten. Die Objekte der Modellklassen werden deshalb auch als *Transferobjekte* bezeichnet, da sie häufig von der Clientschicht bis zur Persistenzschicht und umgekehrt durchgereicht werden. In diesem Zusammenhang fallen auch häufig die Begriffe POJO und JavaBean.

POJOs (Plain Old Java Objects)

werden durch gewöhnliche Java-Klassen unabhängig von jeglichem zusätzlichen Framework nur auf Basis der Java-Programmiersprache realisiert. In der Praxis wird der Begriff aber häufig nicht so eng ausgelegt. Wenn eine Klasse z.B. lediglich zusätzlich Annotationen enthält, wird sie häufig dennoch als POJO bezeichnet, da es sich nur um Metadaten handelt.

JavaBeans

sind Java-Klassen, die häufig zur Kapselung von wiederkehrenden Aufgaben oder als Datencontainer eingesetzt werden. JavaBeans haben als gemeinsame Merkmale einen öffentlichen Konstruktor ohne Parameter, sind serialisierbar und für jedes Attribut einer JavaBean gibt es öffentliche Zugriffsmethoden (Getter/Setter) (vgl. `http://docs.oracle.com/javase/tutorial/javabeans/index.html`).

JavaServer Pages

JavaServer Pages erlauben die Entwicklung einer View mit einer Seitenbeschreibungssprache auf Basis von HTML. Zur Realisierung der Controller-Logik ist es möglich, mit spezieller Syntax Java-Deklarationen, Java-Code und Java-Statements in die Seite einzubetten. Des Weiteren können über Direktiven JavaBeans bekannt gemacht werden und es kann auf deren Eigenschaften zugegriffen werden. Die *JavaServer Pages Standard Tag Library* (JSTL) ist eine Erweiterung der JSP-Technologie um spezielle Tags für häufig benötigte Aufgaben und UI-Komponenten. Die Verwendung der JSTL vereinfacht und standardisiert die Erstellung von JSP-Views. JSP ist im Java-EE-Standard nicht mehr die bevorzugte View-Technologie, daher verzichten wir auf weitere Erläuterungen.

JavaServer Faces

In früheren JSF-Versionen diente JSP als Seitenbeschreibungssprache. Mit der Version 2.0 führte JSF eine eigene View-Technologie auf Basis von XHTML ein, die *Facelets*. Facelets bieten einige Vorteile gegenüber JSP, wie z.B. das Vorlagensystem, das wir in Abschnitt 4.4 vorstellen werden. JSF liefert zusätzlich zum MVC-Muster ein UI-Komponentenmodell inkl. Ereignisverarbeitung (engl. *event handling*) analog zu UI-Bibliotheken für Desktop-Anwendungen (z.B. JavaFX). Die UI-Komponenten stehen in den Facelets in Form von Tags zur Verfügung. JSF bringt eigene umfangreiche Tag-Bibliotheken mit.

Die im Facelet durch Tags eingebundenen UI-Komponenten einer View werden zur Laufzeit durch Instanzen von Klassen der JSF-API auf dem Applikationsserver repräsentiert. Die View im Ganzen entspricht einem Komponentenbaum, d.h., das Instanzengeflecht stellt eine Baumstruktur dar. Der Komponentenbaum wird beim ersten Zugriff eines Webclients auf eine View aus dem Facelet aufge-

baut und bildet die Basis für die Bearbeitung der folgenden Anfragen des Webclients an die View.

Im Browser arbeitet der Benutzer nur mit einer Repräsentation des Komponentenbaums in HTML/JavaScript, die mit dem Server über HTTP kommuniziert. Die Aufgabe der JSF-Implementierung ist es, Anfragen eines Webclients an eine View zu bearbeiten und als Ergebnis die Repräsentation einer neuen View (bei der Navigation auf eine andere Seite) oder die geänderte Repräsentation der ursprünglichen View zu generieren und als Antwort zurückzusenden. Dazu implementiert JSF ein komplexes Bearbeitungsmodell, bestehend aus sechs Phasen, an deren Anfang die Wiederherstellung des Komponentenbaums für die View und am Ende das Rendering der Antwort steht.

Dazwischen können Anfrageparameter übernommen, konvertiert und validiert werden, Attribute der an UI-Komponenten gebundenen Modelle aktualisiert sowie bei UI-Komponenten registrierte Action- oder Action-Listener-Methoden aufgerufen werden. In jeder Phase können Events (z.B. Fehler in der Validierung) auftreten, die dazu führen, dass direkt in die letzte Phase gesprungen wird, um eine Antwort zu generieren (z.B. eine Seite mit Fehlermeldungen). Abbildung 1–2 stellt die Bearbeitung einer JSF-Anfrage (Faces-Request) im Kontext der Ausführungsumgebung schematisch dar.

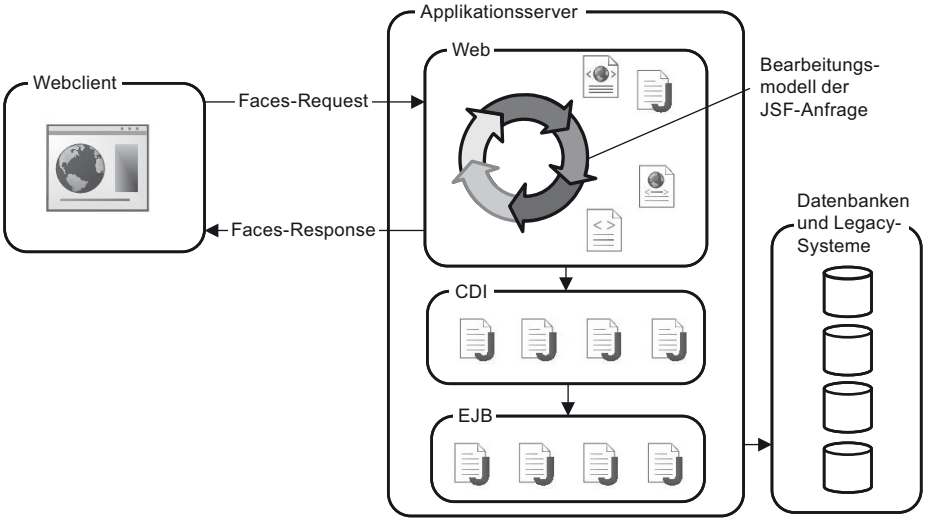

Abb. 1–2 *Schematische Darstellung der Bearbeitung einer JSF-Anfrage (Faces-Request) durch den Server*

Bei JSF kann man JSF Managed Beans oder CDI-Beans als Controller einsetzen. Beide Typen von Beans können mithilfe der *Expression Language* (EL), durch Verwendung in der Seitenbeschreibung, in die View eingebunden werden. Im vorliegenden Buch werden wir CDI-Beans zu diesem Zweck einsetzen, da diese bessere Möglichkeiten bieten, eine flexible Softwarearchitektur zu realisieren.

In Tabelle 1–1 sind die unterschiedlichen Webtechnologien des Java-EE-Standards kurz zusammengefasst.

Webtechnologie	Kurzerklärung
Java Servlets	Werden heute nicht mehr verwendet, um eine View direkt zu erzeugen. Keine ausreichende intrinsische Trennung von Darstellung und Logik. Dient jedoch zur Realisierung von Webframeworks auf höherer Abstraktionsebene (z.B. JSP, JSF).
JavaServer Pages	Veraltete Technologie zur Gestaltung der View. Logik wird über JavaBeans oder Java-Code über spezielle Tags direkt in die View integriert. Wird aus Kompatibilitätsgründen weiter unterstützt.
JavaServer Faces	Aktuelle Technologie zur Beschreibung einer View. Als View-Technologie kommen Facelets auf Basis von XHTML zum Einsatz. JSF bringt ein UI-Komponentenmodell mit Ereignisverarbeitung mit. Logik wird durch JSF Managed Beans oder CDI mithilfe der angesprochenen Ereignisverarbeitung bzw. über spezielle Tags in die View integriert.

Tab. 1–1 *Webtechnologien des Java-EE-Standards*

1.2.2 CDI – ein Komponententyp mit viel Potenzial

CDI-Beans sind ein relativ neuer Bean-Typ, der mit der Version 6 Einzug in Java EE gehalten hat. Betrachtet man die Situation vor der Einführung von CDI, so fragt man sich, wozu CDI eigentlich gut sein soll. Eigentlich hatten wir doch bereits alles, was wir brauchen. Oder? Mit JSF-Facelets und JSF Managed Beans (View, Controller), POJO (Model) und EJB (fachliche Logik) sollten wir ohne Probleme eine mehrschichtige Geschäftsanwendung programmieren können! Antwort: Richtig!

Sicher? Die Macher des Java-EE-Standards stellen dem Programmierer trotzdem eine weitere Komponententechnologie namens *Contexts and Dependency Injection* (CDI) zur Verfügung. Aber warum?

Auf der einen Seite ist CDI eine Technik, mit der man die Webschicht und die transaktionale Businessschicht analog zu den JSF Managed Beans verknüpfen kann. In den Facelets kann ebenfalls über die Expression Language auf die CDI-Beans zugegriffen werden. Auf der anderen Seite geht das Konzept von CDI aber weit über das der JSF Managed Beans hinaus. Es handelt sich um einen generellen Ansatz, wie Beans zur Laufzeit durch *Dependency Injection* (DI) miteinander verknüpft werden. Beans werden nicht selbst erzeugt, sondern durch Deklaration und Annotation (`@Inject`) an bestimmten Stellen angefordert. Der Container sorgt zur Laufzeit für die Verfügbarkeit der Beans. CDI-Beans zielen darauf ab, durch die konsequente Anwendung von DI und ergänzenden Ansätzen (z.B. anwendungsweite Nachrichten) zwischen Komponenten und die klare Vorgabe von Sichtbarkeitsbereichen (Contexts) die Wartbarkeit, Erweiterungsfähigkeit und Testbarkeit von Anwendungen zu verbessern.

Dependency Injection (DI)

ist ein Muster, das die Aufgabe der Objekterzeugung, -bereitstellung und -verwaltung vom Nutzer des Objektes an eine Laufzeitumgebung (Container) übergibt. Der Nutzer definiert lediglich, was er an welcher Stelle haben möchte (Typ, Scope, Anzahl), und die Laufzeitumgebung sorgt für alles Weitere. Das vereinfacht den Zugriff, ermöglicht lose Kopplungen zwischen Objekten und behält dennoch die Typsicherheit bei.

CDI ist eine allgemeine, schichtenunabhängige Containertechnologie. CDI ist erweiterbar und kann dadurch schichtenspezifische Aufgaben übernehmen. Zum Beispiel bietet das Framework Seam 3 viele Erweiterungen an, u.a. eine für Transaktionen, was ja eigentlich eine Domäne der EJBs ist. CDI ist daher vom Ansatz allgemeiner als andere Bean-Technologien, die nur für einen bestimmten Zweck gebaut wurden. Der CDI-Container lässt sich nicht nur in Java-EE-Umgebungen einbetten, sondern auch in andere, z.B. in Java-SE-Anwendungen. Wir werden in Kapitel 6 genauer auf CDI und die Unterschiede zu den anderen Komponententechnologien eingehen.

1.2.3 Enterprise JavaBeans

Enterprise JavaBeans (EJBs) sind Komponenten, die die Geschäftslogik realisieren und damit zur Businessschicht einer mehrschichtigen Applikation gehören. Sie werden vom *EJB-Container* des Applikationsservers verwaltet. Über den Container haben die EJBs Zugriff auf die für Geschäftsanwendungen wichtigen Dienste wie z.B. Transaktionssteuerung oder Sicherheitsmechanismen. Die EJBs können auch als interne »Service-Beans« der Java-EE-Anwendung angesehen werden. Wann immer Geschäftslogik mit transaktionalem Verhalten benötigt wird, sind EJBs die erste Wahl, da sie dieses Verhalten von Haus aus bereits mitbringen.

Nicht nur die Komponenten der Webschicht entwickelten sich über die Jahre weiter, auch bei den EJBs gab es Veränderungen. Zu Beginn wurden drei verschiedene Typen eingeführt, zwei davon existieren heute noch, können aber seit der Version Java EE 5 mit wesentlich weniger Programmieraufwand erzeugt werden: die *Session Beans* und die *Message-Driven Beans*. Die *Entity Beans*, die als persistente Fachklassen konzipiert waren, wurden wieder abgeschafft und stattdessen wurde die Java-weit einheitliche *Java Persistence API* (JPA) eingeführt.

Während JSF Managed Beans und CDI vor allem als Controller in der Webanwendung zum Einsatz kommen, wird mit den EJBs die »schwergewichtige«, transaktionale Geschäftslogik realisiert. EJBs werden niemals direkt in eine View eingebunden, sondern immer nur über den jeweiligen Controller. Java EE 7 ermöglicht die Verfügbarkeit von EJBs innerhalb der Controller-Klassen, analog zu CDI, in einfacher Art und Weise mittels Dependency Injection über Annotati-

onen. Der gleiche Mechanismus kann auch bei der Verwendung von EJBs untereinander verwendet werden.

1.2.4 Java Persistence API (JPA)

JPA ist keine Java-EE-Komponententechnologie, sondern eine Java-weite Technologie für die Abbildung von Objektgeflechten auf relationale Strukturen einer Datenbank. Das heißt, unabhängig davon, ob Desktop-Anwendung oder verteilte, mehrschichtige Geschäftsanwendung, mit JPA kann die objektrelationale Abbildung für die Anwendung definiert werden. Durch JPA wird den Java-Klassen suggeriert, mit einer objektorientierten Datenbank zu arbeiten, obwohl tatsächlich ein relationales Datenbanksystem zum Einsatz kommt.

Vor der Festlegung der Abbildung muss man sich überlegen, welche Objekte überhaupt persistiert werden sollen. In der Regel sind das eindeutig bestimmbare Informationsobjekte. Beispielsweise enthält eine Bankanwendung entsprechende Fachklassen zur Speicherung von Kunde und Konto.

Die eindeutige Zuordnung eines Objekts wird durch die Existenz eines eindeutigen Schlüssels der Klasse unterstrichen. Fachklassen mit einem eindeutigen Schlüssel werden auch *Entitäten* genannt. Nicht immer ist es sinnvoll, eine Entität auf genau eine Tabelle abzubilden, JPA ermöglicht hier aber viele Spielarten.

Die Abbildung einer Klasse und ihrer Attribute auf eine relationale Datenbank wird über Annotationen definiert. Die Annotation `@Entity` kennzeichnet z.B. eine Klasse als Entität, mit `@Id` kann ein Attribut der Klasse als Schlüssel definiert werden und mit `@Table` kann explizit der Entität ein Tabellenname in der Datenbank zugewiesen werden.

Dadurch lässt sich zum einen anhand der Metadaten der Entitäten beim Deployment ein Schema in der Zieldatenbank anlegen (oder ein existierendes für die Verwendung verifizieren). Zum anderen können die Daten von Objekten zur Laufzeit anhand der Metadaten korrekt in den Tabellen der Zieldatenbank gespeichert werden. Diese Aufgabe übernehmen ebenfalls Klassen (z.B. `javax.persistence.EntityManager`) der JPA.

Im Rahmen von Java-EE-Anwendungen werden diese Klassen der JPA innerhalb der EJBs verwendet, die den Controllern der Webschicht transaktionale Methoden zum Persistieren von Entitäten zur Verfügung stellen. Abbildung 1–3 veranschaulicht die Idee der objektrelationalen Abbildung. Im Bild ist eine Entität »Donation«(dt. Spende) dargestellt, deren Attribute auf eine Tabelle »Donation« in einer relationalen Datenbank abgebildet werden.

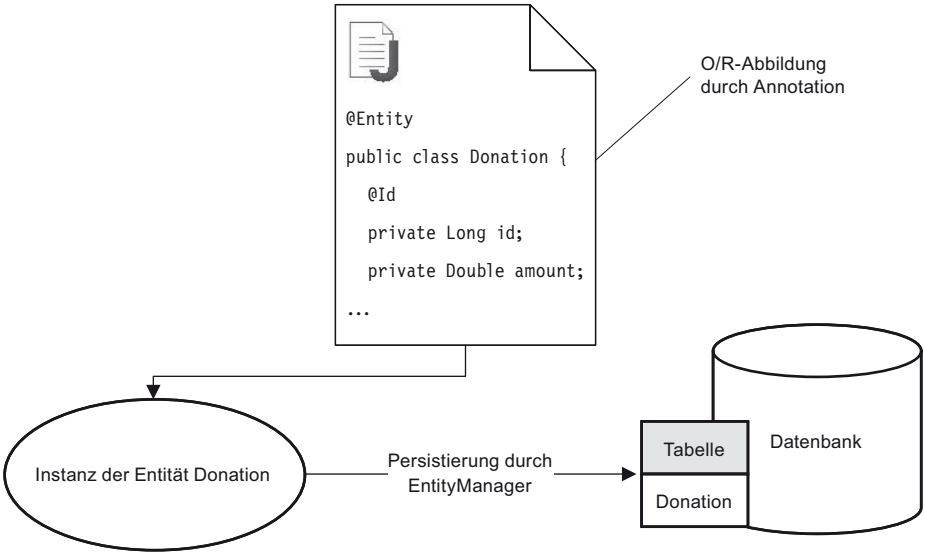

Abb. 1–3 *Objektrelationale Abbildung einer Instanz der Entität Donation auf eine Tabelle Donation in einer relationalen Datenbank*

1.2.5 Die Java-EE-Profile

Mit den Java-EE-Profilen wurde bereits in der Version 6 ein gänzlich neuer Weg in Richtung maßgeschneiderter Java-Applikationsserver für Anwendungsdomänen eingeschlagen. Der komplette Java-EE-Standard ist trotz aller Vereinfachungen nach wie vor sehr komplex und umfangreich. Die Java-EE-Profile berücksichtigen den Umstand, dass nicht für alle Projekte der komplette Umfang der Java EE notwendig ist.

Ein Profil entspricht einem standardisierten Teil-Stack des Java-EE-Standards, d.h., ein Profil definiert genau, welche APIs des Java-EE-Standards enthalten sind. Dadurch sollen nicht nur für einen speziellen Zweck zugeschnittene API-Pakete für Entwickler definiert, sondern auch sehr schmale und spezialisierte Server ermöglicht werden. Die Version 6 führte als erstes und bisher einziges Java-EE-Profil – neben dem vollen Umfang (*Full Profile*) – das *Web Profile* ein. Wie der Name bereits verrät, ist es speziell auf Webprojekte zugeschnitten und findet in diesem Buch besondere Beachtung. Java EE 7 führte keine weiteren Profile ein, erweiterte jedoch das Web Profile. In Tabelle 1–2 werden die wichtigsten Technologien aufgelistet. Eine komplette Übersicht finden Sie in der Spezifikation (DeMichiel & Shannon, 2013).

Technologie	Bemerkung
Java Servlets 3.1	Webtechnologien unterschiedlicher Abstraktionsstufen
JavaServer Pages (JSP) 3.2	
JavaServer Faces (JSF) 2.2	
Expression Language (EL) 3.0	Eine in die VDL einbettbare Sprache, die hauptsächlich zur Adressierung von Bean-Attributen und Action- bzw. Action-Listener-Methoden verwendet wird
Standard Tag Library for JavaServer Pages (JSTL) 1.2	Komponentenbibliothek für JavaServer Pages
Enterprise JavaBeans (EJB) 3.2 lite	Komponenten zur Realisierung der Geschäftslogik einer Anwendung. Der Zusatz »lite« deutet an, dass nicht die volle Mächtigkeit der EJBs in Anwendungen des Web Profile zur Verfügung steht (z. B. keine Web Service Endpoints oder Message-Driven Beans).
Contexts and Dependency Injection (CDI) 1.1	Komponententyp, der als Controller-Bean für JSF verwendet werden kann und den Aufbau flexibler Softwarearchitekturen ermöglicht
Managed Beans 1.0	Managed Beans werden zur Realisierung von Controller-Beans in JSF verwendet.
Interceptors 1.2	Ermöglicht aspektorientierte Programmierung in Java-EE-Anwendungen
Java Persistence API (JPA) 2.1	Spezielle API zur Realisierung der objektrelationalen Abbildung von Java-Objekten in Datenbanken, Datenspeicherung, Transaktionen und Überprüfung der Gültigkeit von Daten
Java Transaction API (JTA) 1.2	API zum Transaktionsmanagement
Bean Validation 1.1	API zur Überprüfung der Gültigkeit von Werten der Attribute bzw. der Parameter von Methoden einer Bean
Java API for RESTful Web Services (JAX-RS) 2.0	API für die Realisierung von HTTP-basierten, lose gekoppelten Webservices
Java API for WebSocket 1.0	API für die Realisierung eines bidirektionalen Kommunikationskanals zwischen dem Browser und dem Applikationsserver
Java API for JSON Processing (JSON-P) 1.0	API für das Parsing von JSON-Dateien

Tab. 1–2 *Die wichtigsten Technologien des Java EE 7 Web Profile*

Im vorliegenden Buch werden wir diese Technologien Schritt für Schritt kennenlernen. Damit schaffen wir einen Einstieg in die Java-EE-Welt, kommen aber auch zu Punkten, an denen wir Technologien benötigen, die nicht im Web Profile enthalten sind. In Kapitel 8 und 9 wird jeweils explizit darauf verwiesen, wenn etwas nicht mehr im Web Profile enthalten ist. Bis dahin gehören alle eingesetzten APIs zum Web Profile.