Java EE 6
— Ein Überblick —

Bernd Müller

Fakultät Informatik
Ostfalia — Hochschule Braunschweig/Wolfenbüttel

GI-Regionalgruppe Braunschweig, 16.2.2012

Vorstellung Referent

- Studium Informatik, Uni Stuttgart
- Promotion in Informatik, Uni Oldenburg
- IBM, HIS
- Professor für Wirtschaftsinformatik, HS Harz
- Professor für Software-Technik, Fakultät Informatik, Ostfalia
- Buchautor (JSF, JPA, Seam, ...)
- Mitglied EG JSF 2.2, JPA 2.1
- GF PMST GmbH
- ...

Überblick

Java EE 6: Definition und Ziele

- Definition
  - Eine Umbrella-Spezifikation (28 einz.) im JCP: JSR 316
  - Released Dezember 2009
  - Breite Unterstützung durch Hersteller
  - Mittlerweile in der Praxis angekommen
- Ziele
  - Erweiterbarkeit
  - Profile
  - Pruning
  - Technologische Verbesserungen
  - ...

Neuerungen

Neu hinzugekommen

- Context and Dependency Injection (JSR 299)
- Bean Validation (JSR 303)
- RESTful Web Services (JSR 311)

Runderneuert

- Servlet 3.0
- JSF 2.0
- JPA 2.0
- EJB 3.1

JSF 2.0

JSF 2.0

- Ajax
- Annotationen
- Facelets
- faces-config.xml optional
- Navigationsregeln ohne XML
- Templating
- System-Events
- Bookmarks
- Automatische Bibliothekserkennung
- ...

JPA 2.0

JPA 2.0

- Verbessertes OR-Mapping
- Typesafe Criteria API
- Erweiterte Abfragesprache
- Second Level Cache
- Neue Lock-Modi
- ...

EJB 3.1

EJB 3.1

- No-interface views
- War Deployment
- Einfache, deklarative Timer (Cron-Syntax)
- Singletons
- Asynchrone Session-Bean-Methoden
- Embeddable Container (main(), Unit-Tests)
- Portables JNDI
- ...

CDI 1.0

CDI 1.0

- Übergeordnetes Thema: Strong Typing, loose Coupling
- Type-safe Dependency Injection
- Qualifier
- Scopes
- Alternatives
- EL-Namen
- Interceptoren und Dekoratoren
- Portable Erweiterungen (SPI)
- ...

Bean Validation 1.0

Bean Validation 1.0

- Tier-unabhängige Validierung (DRY)
- Integration mit JSF und JPA
- Einfacher Satz von Constraints
- Erweiterbar
- ...

JAX-RS 1.1

JAX-RS 1.1

- Der neue Stern am SOA-Himmel
- RESTful Web-Services
- POJO-basiert
- Annotationen
- HTTP zentriert (GET, POST, PUT, DELETE)
- ...

Fragen und Anmerkungen


NICHT QUATSCHEN...
MACHEN!

Der Code

Kundeneingabe mit JSF

```
<h:form>
  <h:panelGrid columns="2">
    Vorname:
    <h:inputText value="#{kundeView.kunde.vorname}" />
    Nachname:
    <h:inputText value="#{kundeView.kunde.nachname}" />
    Geburtsdatum:
    <h:inputText value="#{kundeView.kunde.geburtsdatum}">
      <f:convertDateTime pattern="dd.MM.yyyy"/>
    </h:inputText>
  </h:panelGrid>
  <h:commandButton action="#{kundeView.speichern}"
                   value="Speichern" />
</h:form>
```

JSF Managed Bean für Kundeneingabe

```
@Named
@RequestScoped
public class KundeView {

  private Kunde kunde;

  @Inject KundeService kundeService;

  public KundeView() {
    kunde = new Kunde();
  }

  public String speichern() {
    kundeService.speichern(kunde);
    return null;
  }
  ...
}
```

EJB KundenService

```
@Stateless
public class KundeService {

  @PersistenceContext
  EntityManager em;

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

}
```

JPA Entity

```
@Entity
public class Kunde {

  @Id @GeneratedValue
  private Integer id;

  @Column(length = 30)
  @NotNull
  @Size(min = 3, max = 30)
  private String vorname;

  ...

  @Temporal(TemporalType.DATE)
  @Past
  private Date geburtsdatum;

  public Kunde() {
  }
  ...
```

REST

```
@Path("kunden")
@Stateless
public class KundeService {

  // Erweiterung: alle Kunden als XML
  @GET
  @Produces(MediaType.TEXT_XML)
  public List<Kunde> kunden() {
    return em.createQuery("select k from Kunde k",
                          Kunde.class)
             .getResultList();
  }

}
```

Dazu noch Entity erweitern

```
@XmlRootElement  // JAXB ist Standard SE seit Java 5
@Entity
public class Kunde {

   ...

}
```

Asynchrone Verarbeitung

```
@Stateless
public class SecondService {

   @Asynchronous
   public Future<String> async() {
      ...
      return new AsyncResult<String>(...);
   }

}
```

Asynchrone Verarbeitung (Client)

```
@Stateless
public class FirstService {

  @Inject
  SecondService secondService;

  public String test() {
    ...
    Future<String> result = secondService.async();
    ...
    // blockiert, Alternative ueber CDI-Events
    String result = result.get();
    ...
  }
}
```

Timer

```
@Stateless
public class Timed {

   // jede Minute:
   @Schedule(minute = "*/1", hour = "*")

   // jede Sekunde:
   @Schedule(second = "*/1", minute = "*", hour = "*")

   // Mo-Fr, alle 20 Min:
   @Schedule(dayOfWeek = "1-5", hour = "0/1",
             minute = "0/20")
   public void doSomething() {
      ...
   }
}
```