Neues in der Java Enterprise Edition 7

Mehr als nur einen Blick wert – auch ohne Cloud!

W3L AG
info@W3L.de

2012

Inhaltsverzeichnis

zurück

Allgemeines

Java Enterprise Edition

- **Spezifikation von Softwarekomponenten und Diensten für Enterprise Anwendungen**
  - 40 Teilspezifikationen für
    - User Interfaces
    - Geschäftslogik
    - Persistenz
    - Etc.
- **Entwicklung**
  - Erste Version *J2EE 1.0* (1999)
  - Letzte Version *Java EE 6* (2009)
  - Aktuelle Version *Java EE 7* (2013)
  - Nächste Version *Java EE 8* (2015)

Allgemeines

Java EE 6

- **3,5 Jahre nach Java EE 5**

- **Hauptthema Usability**
  - Verzicht auf XML-Deskriptoren
  - Verzicht auf technische Interfaces / Exceptions
  - Verstärkte Verwendung von Annotationen
  - *Convention Over Configuration*
  - *Web-Profile* verschlankt Umfang zielgerichtet

Java EE 7 - JSR 345

- **Verschoben von Q3 2012 auf Q2 2013**
  - Freigegeben seit 29.04.2013
- **Hauptthema Usability**
- ***Cloud*-Unterstützung und *Mandantenfähigkeit* auf *Java EE 8* verschoben**
  - Spec. lead Linda DeMichiel: „... and partially it is due to our conservative approach in trying to get things "right" in view of limited industry experience in the cloud area when we started this work.“

Java EE 7

Neue APIs und Updates

	Java EE 6	Java EE 7	JSR
Web Services Technologies			
RESTful Web Services (**JAX-RS**)	1.1	2.0	JSR 339
WebSockets	-	1.0	JSR 356
JSON Processing (**JSON-P**)	-	1.0	JSR 353
Web Application Technologies			
Java Servlet (**Servlet**)	3.0	3.1	JSR 340
JavaServer Faces (**JSF**)	2.0	2.2	JSR 344
Expression Language (**EL**)	2.2	3.0	JSR 341
Enterprise Application Technologies			
Contexts and Dependency Injection (**CDI**)	1.0	1.1	JSR 346
Bean Validation	1.0	1.1	JSR 349
Enterprise JavaBeans (**EJB**)	3.1	3.2	JSR 345
Java Persistence API (**JPA**)	2.0	2.1	JSR 338
Java Message Service (**JMS**)	1.1	2.0	JSR 343
Batch Applications for Java (**JBatch**)	-	1.0	JSR 352

RESTful Web Services 2.0

JSR: 339

Java EE 6 Version: 1.1

JAX-RS 2.0

Client API

- **Erweiterung durch *Provider***

```
Client client = ClientFactory.newClient();
client.register(SeminarReader.class);
client.register(SeminarWriter.class);
client.register(ShopItemFeature.class);
client.setProperty("maxCartSize", 100);
```

- **Feature**
  - Erbt von Provider
  - Bündelt Gruppen von *Providern* zu *Features*

JAX-RS 2.0

Client API

- ***Filter***
  - Erbt von `Provider`
  - Greift vor oder nach Absenden des *Requests* ein
  - Vereinfachen Realisierung querschneidender Belange
- ***Interceptor***
  - Greift beim Senden oder Empfangen von Daten ein
  - Können auf *EntityStreams* zugreifen
  - Vereinfachen Realisierung von Kompression oder Verschlüsselung
- **Filter und Interceptors werden auch in der Server-API unterstützt**

JAX-RS 2.0

Client API

- ***WebTarget***
  - WebTarget-Objekte repräsentieren Ressourcen mit ihren URIs
    - Dokumente
    - Websites
    - Java Objekte
    - Etc.
  - Anforderung in verschiedenen Repräsentationen (XML, HTML, Java-Objekt, etc.)

```
// Client - WebTarget - Invocation
Seminar mySeminar =
       client.target("http://.../katalog/seminarData") // Client
       .queryParam("pin", "1234")          //WebTarget
       .request(MediaType.TEXT_XML)        //WebTarget
       .get(Seminar.class);                //Invocation
```

JAX-RS 2.0

Asynchrone Verbindungen

- **Serverseitig**
  - *HTTP Worker Thread* kann Anfragen beantworten, während er auf Beendigung der *Ressourcenmethode* eines asynchronen Aufrufs wartet
- **Clientseitig**
  - Kein Warten auf Antwort
  - Antwort / Fehlschlag wird von `InvocationCallback`-Objekt verarbeitet

WebSocket 1.0

JSR: 356

Java EE 6 Version: -

WebSocket

- **HTML5-Technik für Verbindung über TCP**
  - Bidirektional
  - Full-Duplex
- **Serverseitig**
  - *Endpoint*-Definition per Annotation @WebSocket

```
@WebSocket(path="/helloWorld")
public class EchoBean
{
    @WebSocketMessage
    public String echo(String message)
    {
        return message + " - Oh hello!";
    }
}
```

WebSocket

- **Clientseitig**
  - Verbindung per JavaScript
  - send() sendet Anfrage an Server
  - onmessage() registriert Methode für Empfang von Nachrichten

```
var wsUri = "ws://localhost:8080/websockets/helloWorld";
var websocket = new WebSocket(wsUri);
websocket.onopen = function(evt) { onOpen(evt) };
websocket.onmessage = function(evt) { onMessage(evt) };
websocket.onerror = function(evt) { onError(evt) };
websocket.send(„Hello world");
```

JSON Processing 1.0

JSR: 353

Java EE 6 Version: -

JSON-P

- **Umfang**
  - Erzeugen
  - Lesen
  - Schreiben
  - Stream- oder DOM-API-basiert
  - **Kein** *Binding* zu Java-Objekten

JSON-P

DOM-API

- ***High-Level-API***
  - Äquivalent zu XML-DOM
  - Flexible Bearbeitung von JSON-Objekten
- ***JsonObjectBuilder***

```
JsonObject myJsonSeminar = Json.createObjectBuilder()
        .add("titel", "Führerscheinvorbereitung")
        .add("unterrichtseinheiten", 25)
        .add("referenten", Json.createArrayBuilder()
                .add(Json.createObjectBuilder()
                        .add("vorname", "Peter")
                        .add("nachname", "Mayer"))
                .add(Json.createObjectBuilder()
                        .add("vorname", "Max")
                        .add("nachname", "Mustermann")))
.build();
```

JSON-P

DOM-API

JsonReader

```
String myJsonString = "{"
                    + "    'referent': {"
                    + "        'vorname': 'Max',"
                    + "        'nachname': 'Mustermann'"
                    + "    }"
                    + "}";
JsonReader jr = Json.createReader(new StringReader(json));
JsonValue myJsonObject = jr.readObject();
jr.close();
```

JSON-P

Streaming-API

- ***Low-Level API***
  - Effiziente Verarbeitung großer Datenmengen

- ***JsonParser***
  - *Pull parsing*
    - Nur relevante Informationen werden verarbeitet

```
Event event = parser.next(); // START_OBJECT : {
event = parser.next();       // KEY_NAME      : "referent"
event = parser.next();       // START_OBJECT : {
event = parser.next();       // KEY_NAME      : "vorname"
event = parser.next();       // VALUE_STRING : "Max"
parser.getString();          // "Max"
```

JSON-P

Streaming-API

- ***JsonGenerator***
  - Schreiben in *Stream*

```
JsonGenerator generator = Json.createGenerator(
                new FileWriter(..));

generator
      .writeStartObject()
                .writeStartObject("referent")
                        .write("vorname", "Max")
                        .write("nachname", "Mustermann")
                .writeEnd()
      .writeEnd();
generator.close();
```

Java Servlet 3.1

JSR: 340

Java EE 6 Version: 3.0

Servlet 3.1

- ***Non-blocking I/O***
  - Asynchrone Verarbeitung von Requests
  - Threads werden sofort frei
  - `ReadListener` oder `WriteListener` werden benachrichtigt, wenn Daten bereitstehen
- ***HTTP protocol upgrade***
  - Wechseln von HTTP1.1 zu anderen Protokollen (z.B. ***ws***)
- ***ServletResponse***
  - `reset()`
    - Setzt bisherige Rückgaben und Metadaten zurück
  - `setCharacterEncoding()`
    - Legt Encoding des Rückgabestroms fest

JavaServer Faces 2.2

JSR: 344

Java EE 6 Version: 2.0

JSF 2.2

- ***ViewAction***
  - Werden an *View* gebunden
  - Ausführung **vor** dem Laden der Komponenten
  - onPostback spezifiziert, ob die Aktion nur bei Nicht-JSF-Requests ausgeführt werden soll

```
<f:view>
        <f:metadata>
                <f:viewAction execute="#{myBean.myAction}"
                        onPostback="true" />
        </f:metadata>
</f:view>
```

JSF 2.2

- ***FacesFlow***
  - Bündelt *Views*, Methodenaufrufe, *NavigationRuleSets* und *FacesFlows*
  - *ControlFlowCases* verbinden *FacesFlows*
  - *ControlFlowRules* definieren Regeln für Transitionen
  - Neuer Scope *FlowScope*
    - Browsertabs im gleichen Browser und Flow beeinträchtigen sich nicht
- ***FileUpload* Komponente**
  - Standardvariante
  - *AJAX*-Variante

JSF 2.2

- ***Convention Over Configuration***
  - name-Attribut wird optional in *ManagedBeans*, *Validatoren*, *Komponenten* und *Konvertern*
    - Klassenname ist Standardwert

- **Definition von *Tags* ohne `faces-config.xml`**
  - Attribut *createTag* für *@FacesComponent*
  - *Tag* ist sofort in *Facelets* verwendbar

```
@FacesComponent(value="components.CustomComponent",
                createTag=true)
public class CustomComponent extends UIComponentBase {
…
```

JSF 2.2

- ***Resource Library Contracts***
  - „Abgespecktes“ *MultiTemplating*
  - *Contract* bündelt verschiedene Ressourcen (CSS, Abbildungen, etc.) und eine `template.xhtml`-Datei
  - Wird für *View* festgelegt
  - Pfade in *View* werden **relativ zum *Contract*** aufgelöst

```
<f:view contracts="#{contractsBean.contract}">
 <ui:composition template="/template.xhtml">
 ...
</f:view>
```

Expression Language 3.0

JSR: 341

Java EE 6 Version: 2.2

EL 3.0

- ***Stand-Alone-API***
  - Nutzung außerhalb von *Java EE Container*
- ***String-Concatenation* Operator**
- ***Lambda Expressions***

```
v = (x,y)->x+y;

v(3,4)
// Evaluiert zu 7
```

EL 3.0

- **Erstellen von und arbeiten mit *Collections***
  - Syntax basiert z.T. auf *LINQ* in C#
  - Filter
  - Sortierfunktionen
  - Operationen (Average, etc.)
- ***EL-Variablen* und *EL-Funktionen***
- **Zugriff auf statische Felder**

Contexts and Dependency Injection 1.1

JSR: 346

Java EE 6 Version: 1.0

CDI 1.1

- ***Priority Ordering***
  - Definierbare Aufrufreihenfolge von *Interceptors* und *Decorators*

- ***Embedded Mode***
  - Lauffähig in *Java SE* Umgebung
  - Beschränkt auf einen Teil des vollen Umfangs

CDI 1.1

- **Scope *ConversationScope***
  - Gültig innerhalb einer Session
  - Starten und beenden mit `Conversation.begin()` und `Conversation.end()`
  - Jedes *Request* wird einer *Conversation* zugeordnet
    - Requestparameter: `cdi`

Bean Validation 1.1

JSR: 349

Java EE 6 Version: 1.0

Bean Validation 1.1

- **OpenSource**
- **Neue Validierungen**
  - Methodenparameter
  - Rückgabewerte
- ***Bean Validation Provider* können injiziert werden**

Enterprise JavaBeans 3.2

JSR: 345

Java EE 6 Version: 3.1

EJB 3.2

- **Nur noch optional**
  - EJB 2.1 kompatible *Entity Beans*
  - JAX-RPC-basierte *Web Service Endpoints*
  - JAX-RPC-basierte *Client Views*
- **Neue *Lifecycle callback interceptor* Methoden**
  - `@AroundConstruct`
  - `@PostConstruct`
  - `@PreDestroy`
  - `@PostActivate`
  - `@PrePassivate`

EJB 3.2

- **Deaktivierung der Passivierung von *Stateful SessionBeans***
  - *@Stateful(passivationCapable=false)*
- **Verwendung des Pakets *java.io* erlaubt**
  - Vorher im *ClassLoader* unterbunden

Java Persistence API 2.1

JSR: 338

Java EE 6 Version: 2.0

JPA 2.1

- ***Stored Procedures***
  - Abfrage des *OUT*-Parameters
- ***UNSYNCHRONIZED* injizierte *EntityManager***
  - Keine automatische Verbindung zu laufender Transaktion
  - Sinnvoll für Speicherung erst am Ende von Abläufen

```
@PersistenceContext(synchronization=SynchronizationType.UNSYNCHRONIZED)
EntityManager em;
```

JPA 2.1

- *JOINs*
  - Zusätzliche Bedingungen in *JOIN*s mit ON
    - Bsp.: Nur Produkte mit Status „inStock“ suchen

```
// Criteria
CriteriaQuery<Tuple> q = cb.createTupleQuery();
Root<Supplier> s = q.from(Supplier.class);
Join<Supplier, Product> p = s.join(Supplier_.products, JoinType.LEFT);
p.on(cb.equal(p.get(Product_.status), "inStock"));
q.multiselect(s.get(Supplier_.name), cb.count(p));
```

```
/* JPQL */
SELECT s.name, COUNT(p)
FROM Suppliers s LEFT JOIN
s.products p
ON p.status = 'inStock‘
```

```
/* SQL */
SELECT s.name, COUNT(p.id)
FROM Suppliers s LEFT JOIN Products p
ON s.id = p.supplierId
AND p.status = 'inStock’
```

JPA 2.1

- **Downcasting mit *TREAT***
  - Bsp.: Nur Produkte der Unterklasse „Book“ suchen

```
// Criteria
CriteriaQuery<String> q = cb.createQuery(String.class);
Root<Order> order = q.from(Order.class);
Join<Order,Book> book =
        cb.treat(order.join(Order_.product), Book.class);
q.multiselect(book.get(Book_.name), book.get(Book_.isbn));
```

```
/* JPQL */
SELECT b.name, b.ISBN FROM Order o JOIN
TREAT(o.product AS Book) b
```

JPA 2.1

Bulk Operations für Criteria-API

- ***CriteriaUpdate***
  - *UPDATE*-Statement für Objekte einer Klasse

```
// Criteria
CriteriaUpdate<Customer> q = cb.createCriteriaUpdate(Customer.class);
Root<Customer> c = q.from(Customer.class);
q.set(c.get(Customer_.status), "outstanding")
        .where(cb.lt(c.get(Customer_.balance), 10000));
```

```
/* JPQL */
UPDATE Customer c
SET c.status = 'outstanding'
WHERE c.balance < 10000
```

JPA 2.1

Bulk Operations für Criteria-API

- ***CriteriaDelete***
  - *DELETE*-Statement für Objekte einer Klasse

```
// Criteria
CriteriaDelete<Customer> q = cb.createCriteriaDelete(Customer.class);
Root<Customer> c = q.from(Customer.class);
q.where(cb.equal(c.get(Customer_.status), "inactive"),
        cb.isEmpty(c.get(Customer_.orders)));
```

```
/* JPQL */
DELETE
FROM Customer c
WHERE c.status = 'inactive‘ AND c.orders IS EMPTY
```

Java Message Service 2.0

JSR: 343

Java EE 6 Version: 1.1

JMS 2.0

- **Modernisiert und vereinfacht**
- **Abwärtskompatibel**
- **Zeitversetzter Versand**
  - *MessageProducer.setDeliveryDelay()*
- **Nicht blockierendes Versenden**
  - `send()` gibt Kontrollfluss sofort zurück
  - `CompletionListener` behandelt Bestätigung oder Fehlerfall

JMS 2.0

Vereinfachungen

- ***try-with-resources***
  - Session, Connection, MessageProducer implementieren Autoclosable

```
try
(
      Connection connection = connectionFactory.createConnection();
      Session session = connection.createSession(false,
          Session.AUTO_ACKNOWLEDGE);
      MessageProducer producer = session.createProducer(queue);
)
{
      TextMessage textMessage = session.
          createTextMessage("Hello recipient");
      producer.send(textMessage);
}
catch (JMSException e)
{/* Exceptionhandling */}
```

JMS 2.0

Neue Klassen

- ***JMSProducer***
  - Alternative zu `MessageProducer`
  - `Message`-Objekte direkt aus *Strings* oder *Maps* erzeugen und versenden
- ***JMSReceiver***
  - Alternative zu `MessageReceiver`
  - Liefert Daten statt `Message`-Objekten
- ***JMSContext***
  - Verbindet `Session` und `Connection`

JMS 2.0

Neue Klassen

- **Neue Klassen werfen keine *Checked Exceptions***
- ***JMSContext* kann injiziert werden**

```
@Inject
@JMSConnectionFactory("jms/myGlobalFactory")
private JMSContext context;

@Resouce(mappedName="jms/myGlobalQueue")
private Queue myQueue;

public void sendMessage(String msg)
{
     context.createProducer().send(myQueue, msg);
}
```

Batch Applications for Java 1.0

JSR: 352

Java EE 6 Version: -

JBatch 1.0

- **Ausführung und Scheduling von *Jobs***
- **Konfiguration von *Jobs* und *Steps* in *Job XML***
- **Zwei Modi**
  - *Chunk*orientierte Verarbeitung
  - *Batchlets*

JBatch 1.0

*Chunk*orientierte Verarbeitung

- ***Reader-Processor-Writer*-Pattern**
- ***Items* werden einzeln gelesen und verarbeitet**
- **Writer schreibt *Chunk* bei erreichen der *Chunk*-Länge**

JBatch 1.0

Batchlets

- ***Roll-Your-Own-Batch*-Pattern**
- **Durchgehende Ausführung**
- **Expliziter Aufruf**
- **Exit-Value**

JBatch 1.0

Listener

- **Job-Level**
  - Abbruch
  - Neustart
- **Step-Level**
  - *Chunk* und *Batchlet*
    - Vor und nach *Steps*
  - Nur *Chunk*
    - Vor und nach Ereignissen (*Read*, *Write*, *Process*, *NewChunk*, etc.)
    - Fehlerbehandlung
    - Abbruch
    - Neustart

JBatch 1.0

Parallelisierung

- ***Partitioned Step Mode***
  - Parallele Verarbeitung mehrerer *Chunks*
  - *Steps* unabhängig voneinander in nebenläufigen Threads

- ***Concurrent Mode***
  - *Step*-Sequenzen werden zu *Flows* gebündelt
  - Unabhängige nebenläufige Verarbeitung der *Flows* eines *Jobs*

Java Enterprise Edition 8

Java EE 8

- **Release: „Anfang 2015“**
- **Themen**
  - Die Cloud
    - Skalierbarkeit / Elastizität von Enterprise Anwendungen
    - Ressourcen-Teilung
  - *Multi-Tenancy* / Mandantenfähigkeit
    - Mandanten-Oberflächen
    - Mandanten-Prozesse / -Workflows
    - Mandanten-Geschäftslogik
    - Disjunkte und gemeinsame Daten

Quellen und weitere Informationen

- **Java Specification Requests**
  - http://jcp.org/en/jsr/overview
  - http://jcp.org/en/jsr/detail?id=342
- **Arun Gupta's Weblog**
  - https://blogs.oracle.com/arungupta/
- **The Aquarium**
  - https://blogs.oracle.com/theaquarium/
- **Java EE 7 progress page**
  - http://jdevelopment.nl/open-source/java-ee-7-progress-page/

Inhouse-Schulungen

Wir bieten Inhouse-Schulungen und Beratung durch unsere IT-Experten und –Berater.

Schulungsthemen

- Softwarearchitektur (OOD)
- Requirements Engineering (OOA)
- Nebenläufige & verteilte Programmierung

Gerne konzipieren wir auch eine individuelle Schulung zu Ihren Fragestellungen.

Sprechen Sie uns an!
Tel. 0231/61 804-0, info@W3L.de

W3L-Akademie

In Zusammenarbeit mit der Fachhochschule Dortmund bieten wir

zwei Online-Studiengänge

- B.Sc. Web- und Medieninformatik
- B.Sc. Wirtschaftsinformatik

und 7 Weiterbildungen im IT-Bereich an.

Besuchen Sie unsere Akademie!
http://Akademie.W3L.de