

Java Next Generation Day

www.java.at

Agenda

09:00-09:15 Begrüßung

09:15-10:15 Neuerungen in Java Standard Edition 7.0

10:15-10:30 Java 5 EOL - was nun? Java SE for Business!

10:30-10:45 Pause

10:45-11:45 Internas zum Java Hotspot Compiler

11:45-12:45 JavaFX

12:45-13:45 Mittagsbuffet

13:45-14:00 Java EE 6 & JSR 299

14:00-15:30 Neuerungen in Java Enterprise Edition 6.0

15:30 Ende

JavaEE 6 Pruned APIs

- JAX-RPC
- EJB Entity Beans
- JAXR
- Java EE Application Deployment (JSR-88)
- Java EE Management (JSR-77)

Java EE6 Profiles

API	Web Profile	Full Profile
Servlet 3.0	✓	✓
JSP 2.2	✓	✓
JSTL 1.2	✓	✓
EL 1.2	✓	✓
JSF 2.0	✓	✓
WebBeans 1.0 (?)	✓	✓
EJB 3.1 (Lite)	✓	✓
EJB 3.1 (Full)	✗	✓
JPA 2.0	✓	✓
JTA 1.1	✓	✓
JMS 1.1	✗	✓
JavaMail 1.4	✗	✓
JAX-WS 2.2	✗	✓
JAX-RS 1.1	✗	✓
JAXB 2.2	✗	✓
JACC 1.0	✗	✓
JCA 1.6	✗	✓

Was gibt es Neues in JavaEE 6?

- Servlet 3.0
- JSF 2.0
- Bean Validation
- Java Persistence API 2.0
- JAX-RS 1.1
- Enterprise JavaBeans 3.1
- Web-Beans

Servlet 3.0 - web.xml

- New annotations make web.xml optional (see later in this presentation)
- Modularity of web.xml
 - frameworks can come with their own web-fragment.xml in their jar files META-INF to preconfigure controller-servlets, filters or listeners
 - the jar file has to be placed in the WEB-INF/lib of a web-application so that the container scans for the web-fragment.xml
 - web-fragments can be named (see spec 8.2.2)
 - the processing order of the web-fragments can be defined (see spec 8.2.2)

Servlet 3.0 - web.xml

- example web-fragment.xml

```
<web-fragment>
  <servlet>
    <servlet-name>welcome</servlet-name>
    <servlet-class>
      WelcomeServlet
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>welcome</servlet-name>
    <url-pattern>/Welcome</url-pattern>
  </servlet-mapping>
  ...
</web-fragment>
```

Servlet 3.0 - Annotations

- programmatic configuration of the web.xml
- must be done within ContextListeners

```
@ServletContextListener
```

```
public class MyListener {  
    public void contextInitialized (ServletContextEvent sce) {  
        ServletContext sc = sce.getServletContext();  
        sc.addServlet("myServlet", "Sample servlet",  
                    "foo.bar.MyServlet", null, -1);  
        sc.addServletMapping("myServlet",  
                            new String[] {"/urlpattern/*"});  
    }  
}
```

Servlet 3.0 - Annotations

```
@WebFilter(filterName = "NewSimpleFilter", urlPatterns = {"/"})
public class NewSimpleFilter implements Filter {

    public void doFilter(ServletRequest request, ServletResponse response,
        FilterChain chain)
        throws IOException, ServletException {

        chain.doFilter(request, response);
        response.getWriter().println("filter was here");
    }

    public void init(FilterConfig filterConfig) throws ServletException {

    }

    public void destroy() {

    }
}
```

Servlet 3.0 - Annotations

```
@WebServlet(name="MyServlet", urlPatterns={"/foo", "/bar"})
public class SampleUsingAnnotationAttributes {
    @Override
    public void doGet(HttpServletRequest req,
                      HttpServletResponse res)
    {
        ...
    }
}
```

Servlet 3.0 - Async Ajax Techniques

- **Service streaming (streaming)** allows a server to send a message to a client when an event occurs, without an explicit request from the client. In real-world implementations, the client initiates a connection to the server through a request, and the response returns bits and pieces each time a server-side event occurs; the response lasts (theoretically) forever. Those bits and pieces can be interpreted by client-side JavaScript and displayed through the browser's incremental rendering ability.
- **Long polling**, also known as asynchronous polling, is a hybrid of pure server push and client pull. It is based on the Bayeux protocol, which uses a topic-based publish-subscribe scheme. As in streaming, a client subscribes to a connection channel on the server by sending a request. The server holds the request and waits for an event to happen. Once the event occurs (or after a predefined timeout), a complete response message is sent to the client. Upon receiving the response, the client immediately sends a new request. The server, then, almost always has an outstanding request that it can use to deliver data in response to a server-side event. Long polling is relatively easier to implement on the browser side than streaming.
- **Passive piggyback**: When the server has an update to send, it waits for the next time the browser makes a request and then sends its update along with the response that the browser was expecting.

Servlet 3.0 - Asynchronous Servlets

```
@WebServlet(name="myServlet", urlPatterns={"/slowprocess"},
asyncSupported=true)
public class MyServlet extends HttpServlet {

public void doGet(HttpServletRequest request,
                HttpServletResponse response) {
    AsyncContext aCtx = request.startAsync(request, response);
    ...
    anotherThread.doSomethingWith(aCtx);
    //response is not completed here
    //another Thread is completing the response explicitly
    //with response.complete() after writing to the outputStream

}
}
```

Servlet 3.0 - Asynchronous Servlets

```
@WebServlet(name="myServlet", urlPatterns={"/slowprocess"},
asyncSupported=true)
public class MyServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response) {
        AsyncContext aCtx = request.startAsync(request, response);
        ...
        aCtx.dispatch("/asynchResource"); //returns immediately
    }
}
```

Servlet 3.0 - Login & Logout

Login & Logout in JavaEE 5:

- done by container (http basic, form, certificate)
- no way to login programmatically
- `session.invalidate()` to „logout“ user

In JavaEE 6:

- alternative login method for programmatic login
- logout method for logout

Servlet 3.0 - Login & Logout

```
@WebServlet(name="myServlet", urlPatterns={"/login"})
public class MyServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response) {
        String username = ...;
        String password = ...;
        ...
        try{
            request.login(username, password);
        } catch (ServletException e) {
            //Login was not successful
        }
    }
}
```

JSP 2.1 - Maintenance

- Only minor changes in the JSP Spec

Most important:

- Expression Language Method Expressions:

```
...  
The stock price bought: ${trader.buy('JAVA')}  
...
```

- defining parameter types:

```
...  
${trader['buy(java.lang.String)']('JAVA')}  
...
```

Was gibt es Neues in JavaEE 6?

- Servlet 3.0
- JSF 2.0
- Bean Validation
- Java Persistence API 2.0
- JAX-RS 1.1
- Enterprise JavaBeans 3.1
- Web-Beans

JSF 2.0 -Facelets

Facelets is a powerful but lightweight page declaration language to design JavaServer Faces

views using HTML style templates and to build component trees.

Facelets features include the following:

- Use of xhtml for building web pages
- Support for templating for components and pages
- Support for Facelets Tag libraries in addition to JavaServer Faces and JSTL tag libraries
- Support for unified EL
- Compile-time EL Validation

JSF 2.0 -Facelets

Facelets

- are xhtml documents
- use XML namespace declarations to support the **JSF Tag Libraries**
- support a subset of the JSTL Core tag library and the entirety of the JSTL Function tag library
- support the Unified Expression Language
- are validated at compile time
- are not based on JSP technology
- JSF web-apps may use JSPs **or** Facelets for presentation

JSF 2.0 -Facelets

supported tag libraries

Tag Library	URI	prefix	example
JSF UI Tag Library	http://java.sun.com/jsf/facelets	ui:	ui:component ui:insert
JSF HTML Tag Library	http://java.sun.com/jsf/html	h:	h:head h:body h:outputText h:inputText
JSF Core Tag Library	http://java.sun.com/jsf/core	f:	f:actionListener f:attribute
JSTL Core Tag Library	http://java.sun.com/jsp/jstl/core	fn:	fn:toUpperCase fn:toLowerCase
JSTL Functions Library	http://java.sun.com/jsp/jstl/functions	c:	c:forEach c:catch c:if

JSF 2.0 Facelets Templating

- new feature in JSF
- allows creating a page that will act as the base or template for the other pages in the application.
- avoids creation of a number of similar pages
- helps maintaining a standard look and feel

- templates -> common xhtml content of views
- template client -> view specific content

JSF 2.0 Facelets Templating

a template

```
<?xml version=?1.0? encoding=?UTF-8? ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html">
<head>
  <title>
    <ui:insert name="title">Page Title</ui:insert>
  </title>
</body>
</head>
  <body>
    <div>
      <ui:insert name="Links"/>
    </div>>
    <div>
      <ui:insert name="Data"/>
    </div>>
  </body>
</html>
```

JSF 2.0 Facelets Templating

a template client

```
<?xml version=?1.0? encoding=?UTF-8? ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html">
<ui:composition template="/template.xhtml">
  This text will not be displayed.
  <ui:define name="title">
    Welcome page
  </ui:define>
  <ui:define name="Links">
    Links should be here.
  </ui:define>
  <ui:define name="Data">
    Data should be here.
  </ui:define>
</ui:composition>
  This text also will not be displayed.
</html>
```

JSF 2.0 Facelets Composite Components

- define reusable xhtml code for a certain functionality
- can be stored in a library

JSF 2.0 Facelets

Composite Components

an email input field

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:composite="http://java.sun.com/jsf/composite"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core">
<h:head>
  <title>This content will not be displayed
</title>
<h:body>
  <composite:interface>
    <composite:attribute name="Email"/>
  </composite:interface>
  <composite:implementation>
    <h:inputText value="#{Bean.Email}">
    </h:inputText>
  </composite:implementation>
</h:body>
</html>
```

JSF 2.0 Facelets Composite Components

an email input field

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:em="http://java.sun.com/jsf/composite/components">
<head>
  <title>Using a sample composite component</title>
</head>
<body>
  <h:form>
    <em:inputText id="Email"
      value="#{sampleBean.email}" />
  </h:form>
</body>
</html>
```

JSF 2.0 Annotations

managed (backing) bean configuration before JSF 2.0

```
<managed-bean>
  <managed-bean-name>UserNumberBean</managed-bean-name>
  <managed-bean-class>
    guessNumber.UserNumberBean
  </managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
  <managed-property>
    <property-name>minimum</property-name>
    <property-class>long</property-class>
    <value>0</value>
  </managed-property>
  <managed-property>
    <property-name>maximum</property-name>
    <property-class>long</property-class>
    <value>10</value>
  </managed-property>
</managed-bean>
```

JSF 2.0 Annotations

- managed beans may contain JavaEE Annotations like `@Resource` or `@EJB`
- `/WEB-INF/classes` is scanned for JSF Annotations
- JSF Annotations for managed beans are:
 - `ManagedBean`
 - `ManagedProperty`
 - `ReferencedBean`
 - `NoneScoped`, `ApplicationScoped`, `RequestScoped`, `SessionScoped`, `ViewScoped`, `CustomScoped`

JSF 2.0 Annotations

annotated managed bean

```
@ManagedBean(name="order")
@SessionScoped
public class Order {
    @ManagedProperty(value = "#{orderService.maxCount}")
    private int maxCount;
    private String id;
    private int count;
    String orderItem;
    @ManagedProperty(value="#{customer}")
    private Customer customer;
}
```

Was gibt es Neues in JavaEE 6?

- Servlet 3.0
- JSF 2.0
- **Bean Validation**
- Java Persistence API 2.0
- JAX-RS 1.1
- Enterprise JavaBeans 3.1
- Web-Beans

JSR 303 - Bean Validation

Requirements on classes to be validated

Objects that are to be validated must fulfill the following requirements.

- Properties to be validated must follow the method signature conventions for JavaBeans read properties, as defined by the JavaBeans specification.
- Static fields and static methods are excluded from validation.
- Constraints can be applied to interfaces and superclasses.

JSR 303 - Bean Validation

some annotations

Annotation	Description
@Valid	Mark an association as cascaded. The associated object will be validated by cascade.
@NotNull	The annotated element must not be null.
@Size	The annotated element size must be between the specified boundaries (included). Supported types are:
@Min, @Max	The annotated element must be a number whose value must be higher (lower) or equal to the specified minimum (maximum).
@Past, @Future	The annotated element must be a date in the past (future).
@Pattern	The annotated String must match the following regular expression. The regular expression follows the Java regular expression conventions see Pattern. Accepts String. null elements are considered valid.

JSR 303 - Bean Validation

example

```
@ManagedBean(name = "order")
@SessionScoped
public class Order {
    @NotNull()
    @Size(max = 5, min = 5, message = "article number lenght
                                     must be {max}")

    private String itemNumber = "#####";
    @Min(value = 1, message = "minimum quantity is 1")
    @Max(value = 99, message = "do not spend all money")
    @NotNull
    private Integer quantity = 1;

    @Pattern(regex="^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.
[azA-Z]{2,4}$")
    String email;

    ...
}
```

JSR 303 -Bean Validation

- extensible - create your own application constraints
- JSF 2.0 implementations must support Bean Validation when running in an JavaEE 6 container
- which means JSF 2.0 validates input fields based on Bean Validation

JSR 303 -Bean Validation

constraint groups

```
public class User {  
    @NotNull  
    private String firstname;  
    @NotNull(groups = Default.class)  
    private String lastname;  
    @NotNull(groups = {Billable.class, BuyInOneClick.class})  
    private CreditCard defaultCreditCard;  
}
```

Was gibt es Neues in JavaEE 6?

- Servlet 3.0
- JSF 2.0
- Bean Validation
- **Java Persistence API 2.0**
- JAX-RS 1.1
- Enterprise JavaBeans 3.1
- Web-Beans

JPA 2.0

- What's new?
 - enhanced pessimistic locking (... see specs)
 - Criteria API
 - Caching
 - Support of Bean Validation
- Not included
 - query-by-example :-(

JPA 2.0 - Criteria API

- ... alternative to JPA Query Language
- ... type safe
- ... programmatic queries
- ... requires metamodel

```
package domain; class Person
@Entity
public class Person {
    @Id
    private long ssn;
    private string name;
    private int age;
    ...
}
```

class Person_ (metamodel)

```
package domain;
import javax.persistence.metamodel.SingularAttribute;

@javax.persistence.metamodel.StaticMetamodel (domain.Person.class)

public class Person_ {
    public static volatile SingularAttribute<Person,Long> ssn;
    public static volatile SingularAttribute<Person,String> name;
    public static volatile SingularAttribute<Person,Integer> age;
}
```

JPA 2.0 - Criteria API

example

```
EntityManager em = ...
QueryBuilder qb = em.getQueryBuilder();
CriteriaQuery<Person> c = qb.createQuery(Person.class);
Root<Person> p = c.from(Person.class);
Predicate condition = qb.gt(p.get(Person_.age), 20);
c.where(condition);
TypedQuery<Person> q = em.createQuery(c);
List<Person> result = q.getResultList();
```

Person_ ist the metamodel class for Person

JPA query language equivalent

```
String jpql = "select p from Person p where p.age > 20";
```

JPA 2.0 - Criteria API

another example

```
CriteriaQuery<Double> c = cb.createQuery(Double.class);  
Root<Account> a = c.from(Account.class);  
  
c.select(cb.avg(a.get(Account_.balance)));
```

JPA query language equivalent

```
String jpql = "select avg(a.balance) from Account a";
```

JPA 2.0 - Criteria API

and another one

```
QueryBuilder qb = ...
CriteriaQuery<String> q = qb.createQuery(String.class);
Root<Customer> customer = q.from(Customer.class);
Join<Customer, Order> order = customer.join(Customer_.orders);
Join<Order, Item> item = order.join(Order_.lineitems);
q.select(customer.get(Customer_.name))
  .where(qb.equal(item.get(Item_.product)
    .get(Product_.productType), "printer"));
```

JPA query language equivalent

```
SELECT c.name
FROM Customer c JOIN c.orders o JOIN o.lineitems i
WHERE i.product.productType = 'printer'
```

JPA 2.0 - Caching

Cacheable Annotation

```
@Entity
@Cacheable(true)
@Table(name="ORDERS")
@ManagedBean(name = "order")
@SessionScoped
public class Order implements Serializable {
    @Id()
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;

    ...
}
```

JPA 2.0 - Caching

second level cach is configured in persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence ...>
  <persistence-unit name="JPADemoPU" transaction-type="JTA">
    <jta-data-source>jdbc/jpademo</jta-data-source>
    <class>at.java.Order</class>
    <exclude-unlisted-classes>true</exclude-unlisted-classes>
    <shared-cache-mode>ALL</shared-cache-mode>
    <properties>
    </properties>
  </persistence-unit>
</persistence>
```

! Caching may result in stale reads (old objects are read) !

JPA 2.0 - Caching

possible values

- **ALL** all entities are cached
- **NONE** implementation must not cache
- **ENABLE_SELECTIVE** only with `@Cacheable(true)` annotated entities are cached (if `Cacheable()` is not specified cache is disabled)
- **DISABLE_SELECTIVE** only with `@Cacheable(false)` annotated entities are **not** cached (if `Cacheable()` is not specified cache is enabled)

JPA 2.0 - Bean Validation

combined JSF Managed Bean - JPA Entity

```
@Entity
@ManagedBean(name = "order")
@RequestScoped
public class Order implements Serializable {
    @Id()
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;

    @NotNull()
    @Size(max = 5, min = 5,
        message = "article number length must be {max}")
    private String itemNumber = "#####";
    @Min(value = 1, message = "minimum quantity is 1")
    @Max(value = 99, message = "do not spend all your money")
    @NotNull(message="please provide a quantity")
    private Integer quantity = 1;
```

JPA 2.0 - Bean Validation

constraints are validated by JPA implementation

```
public void persistOrder(Order order) {  
    try {  
        em.persist(order);  
    } catch (ConstraintViolationException e) {  
        logger.warning("Constraint violated");  
        for (ConstraintViolation v : e.getConstraintViolations()) {  
            logger.warning("Message: " + v.getMessage());  
            logger.warning("Value: " + v.getInvalidValue());  
        }  
    }  
}
```

could produce the output

```
WARNING: Constraint violated  
WARNING: Message: article number lentgh must be 5  
WARNING: Value: 1234
```

JPA 2.0 - Bean Validation

validation configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence ...>
  <persistence-unit name="JPADemoPU" transaction-type="JTA">
    <jta-data-source>jdbc/jpademo</jta-data-source>
    <class>at.java.Order</class>
    <exclude-unlisted-classes>true</exclude-unlisted-classes>
    <shared-cache-mode>NONE</shared-cache-mode>
    <validation-mode>CALLBACK</validation-mode>
    <properties>
    </properties>
  </persistence-unit>
</persistence>
```

- **AUTO** validate if ValidationProvider is present
- **CALLBACK** validate after PrePersist, ... callbacks
- **NONE** do not validate

Was gibt es Neues in JavaEE 6?

- Servlet 3.0
- JSF 2.0
- Bean Validation
- Java Persistence API 2.0
- **JAX-RS 1.1**
- Enterprise JavaBeans 3.1
- Web-Beans

JAX-RS - RESTful Web Services

- Representational State Transfer (REST)
 - resources that are accessed over http
 - in most cases XML documents
 - data is seen as documents
 - http method semantics are used to retrieve and manipulate data: **get, post, put, delete**
- JAX-RS now part of JavaEE 6
- Sun's implementation is called Jersey
- Simplest way to implement REST: Servlets, some XML API (e.g. DOM, JAXB)

JAX-RS - RESTful Web Services

JAX-RS uses annotated POJOs

```
@Path("/helloworld")
public class HelloWorldResource {
    @GET
    @Produces("text/plain")
    public String getClichedMessage() {
        return "Hello World";
    }
}
```

```
@Path("/hello/{username}")
@GET
public class HelloUser {
    @Produces("text/xml")
    public String getUser(
        @PathParam("username") String userName) {
        ...
    }
}
```

JAX-RS - RESTful Web Services

some annotations

Annotation	Description
@Path	defines URI to access the service. Parameters can be used e.g. „/helloworld/{username}“
@GET	method processes http get
@POST	method processes http post
@PUT	method processes http put
@DELETE	method processes http delete
@HEAD	method processes http head (same header result as get, but empty body)
@PathParam	define parameter which is part of the url e.g. „http://host/JAX-RS-Demo/resources/order/ 12345 “
@QueryParam	define parameter which is handed over as a query parameter http://host/JAX-RS-Demo/resources/order/345KJkj? item=2
@Consumes	MIME type that the resource is taking as input
@Produces	MIME type of the resource

JAX-RS - RESTful Web Services

more examples

```
@Path("/order/{orderid}")
public class RESTService {
    @GET
    @Produces(value="text/plain")
    public String getOrder(
        @PathParam(value="orderid") String orderid,
        @QueryParam(value="mode") String mode) {
        return "Orderid = " + orderid + "\nMode = " + mode;
    }
}
```

```
@Path("/myResource")
@Consumes("multipart/related")
public class SomeResource {
    @POST
    public String doPost(MimeMultipart mimeMultipartData) {
        ...
    }
}
```

JAX-RS - RESTful Web Services

supported types

- `byte[]`—All media types (`*/*`)
- `java.lang.String`—All text media types (`text/*`)
- `java.io.InputStream`—All media types (`*/*`)
- ...
- `javax.activation.DataSource`—All media types (`*/*`)
- `javax.xml.transform.Source`—XML types (`text/xml`, `application/xml` and `application/*+xml`)
- `javax.xml.bind.JAXBElement` and application-supplied JAXB classes XML media types (`text/xml`, `application/xml` and `application/*+xml`)
- ...
- custom data types by defining your own Entity Providers ...

Was gibt es Neues in JavaEE 6?

- Servlet 3.0
- JSF 2.0
- Bean Validation
- Java Persistence API 2.0
- JAX-RS 1.1
- Enterprise JavaBeans 3.1
- Web-Beans

EJB 3.1

Overview/Recap Enterprise JavaBeans Components

Component / Annotation	Description
Stateless Session Bean <code>@Stateless</code>	Instances are held in a pool, container guarantees that a bean instance is only called single threaded. Each time a client calls, it might get a different instance of the bean. Bean can act as a Web Service endpoint.
Stateful Session Bean <code>@Stateful</code>	For each lookup a new instance is created. Bean is destroyed after timeout or explicit call to the <code>@Destroy</code> method. Again bean only can be accessed single threaded. The bean cannot act as a WebService endpoint.
Message Driven Bean <code>@MessageDriven</code>	Receiver of incoming asynchronous messages for JMS Topics and Queues
new: Singleton Session Beans <code>@Singleton</code>	Single instance instead of pooled instances. Enforced load on application startup possible. Bean can act as a Web Service endpoint.

EJB 3.1 - Web Profile

EJBs now run in the Web Container aka EJB lite

Component/Service	EJB Lite	Full EJB
Session Beans (stateful, stateless, singleton)	YES	YES
Message-Driven Beans	NO	YES
2.x/1.x CMP/BMP Entity Beans	NO	YES
Java Persistence API 2.0	YES	YES
Local / No-Interface Access	YES	YES
3.0 Remote Access	NO	YES
2.x Remote Home/Component Access	NO	YES
JAX-WS Web Service Endpoint	NO	YES
EJB Timer Service	NO	YES
Asynchronous Session Bean	NO	YES
Interceptors	YES	YES
RMI-IIOP Interoperability	NO	YES
Container & Bean Managed Transactions	YES	YES
Declarative and Programmatic Security	YES	YES

EJB 3.1 - Singleton Session Beans

example

```
@Startup
@Singleton
public class SharedBean implements Shared {
    private SharedData state;
    @PostConstruct
    void init() {
        // initialize shared data
        ...
    }
    ...
}
```

Bean is loaded on startup as a Singleton.
PostConstruct method can be used for initialization.

EJB 3.1 - Singleton Session Beans

Dependencies can be expressed using the `@DependsOn` Annotation

```
@Singleton  
public class B { ... }  
  
@DependsOn("B")  
@Singleton  
public class A { ... }
```

EJB 3.1 - Singleton Session Beans

container managed concurrency

```
@Singleton
@LocalBean
@ConcurrencyManagement(ConcurrencyManagementType.CONTAINER)
public class NewSessionBean {

    @Lock(LockType.WRITE) //Default
    public void methodA() {
        //...
    }

    @Lock(LockType.READ)
    public void methodB() {
        //...
    }
}
```

any number of concurrent threads can access a Read-Locked methods
if a write locked method is accessed only one thread can enter the bean
(others wait)

EJB 3.1 - Singleton Session Beans

bean managed concurrency

```
@Singleton
@LocalBean //No interface view
@ConcurrencyManagement(ConcurrencyManagementType.BEAN)
//Default
public class NewSessionBean {
    public void methodA() {
        synchronized(this) {
            //...
        }
    }
    public void methodB() {
        synchronized(this) {
            //...
        }
    }
}
```

bean is using Java synchronization

EJB 3.1 - @Asynchronous

```
@Stateless
@LocalBean
public class AsyncBean {
    @Asynchronous
    public Future<String> asyncMethod(String hello) {
        String result = hello.toUpperCase();
        return new AsyncResult<String>(result);
    }
}
```

```
@EJB AsyncBean asyncBean;

public void methodA() throws Exception{
    Future<String> asyncResult =
        asyncBean.asyncMethod("Hello");
    while (!asyncResult.isDone()){
        System.out.println("Waiting for async task");
    }
    String result = asyncResult.get();
}
```

EJB 3.1 - cron-style scheduling

```
@Stateless
@LocalBean
public class MyBean {
    @Schedule(second="0", minute="0", hour="0",
              dayOfMonth="*", month="*",
              dayOfWeek="Mon", year="*")
    public void myMethod() {
        //doSomething
    }
}
```

- Scheduled Timers are started automatically on application deployment
- also can be scheduled programmatically

Was gibt es Neues in JavaEE 6?

- Servlet 3.0
- JSF 2.0
- Bean Validation
- Java Persistence API 2.0
- JAX-RS 1.1
- Enterprise JavaBeans 3.1 Lite
- **Web-Beans**