What's new in Java EE 6?
« EJBs are dead... »

Rod Johnson

« ...Long live EJBs ! »

Antonio Goncalves
Antonio Goncalves

- Software Architect
- Former BEA Consultant
  - Experience with Application Servers
- Java EE 5 author (in French)
- Java EE 6 author (in English)
- JCP Expert Member
  - Java EE 6, EJB 3.1, JPA 2.0
- Co-creator, co-leader of the Paris JUG
- Les Cast Codeurs
Agenda

Presentation Tier
- Servlet 3.0
- JSP 2.2
- JSF 2.0
- EJB 3.1
- JTA 1.1

Business Logic Tier
- Bean Valid 1.0
- JPA 2.0

Persistence Tier

Interoperability Tier
- JAX-WS 2.2
- JAX-RS 1.1
- JMS 1.1
- Injection

Java EE 6
Java EE 6

Presentation Tier
- Servlet 3.0
- JSP 2.2
- JSF 2.0

Business Logic Tier
- EJB 3.1
- JTA 1.1

Persistence Tier
- Bean Valid 1.0
- JPA 2.0

Interoperability Tier
- JAX-WS 2.2
- JAX-RS 1.1
- JMS 1.1

Injection
A Bit of history

Project JPE
May 1998
10 specs

J2EE 1.2
Dec 1999
13 specs
Servlet
JSP
EJB
JMS
RMI/IIOP

J2EE 1.3
Sept 2001
20 specs
CMP
JCA

J2EE 1.4
Nov 2003
23 specs
WS
Management
Deployment

Java EE 5
May 2006
~28 specs
Annotations
EJB 3
JPA 1.0
WS-*
JSF

Java EE 6
Q3 2009
EJB 3.1
JPA 2.0
Servlet 3.0
JSF 2.0
JAX-RS 1.1
JCDI 1.0
Bean Validation

Ease of
development
(web)

Ease of
development

Web Profile

Enterprise
Application

Robust
Scalable
Java EE 6 is Richer, Easier, Lighter

- Richer
  - New specifications

- Easier
  - POJO model
  - Less XML...
  - ... even on the web tier

- Lighter
  - EJB Lite
  - Profiles and Pruning
### Richer: 29 specifications

<table>
<thead>
<tr>
<th>Web Services</th>
<th>Enterprise</th>
<th>Web</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAX-RPC</td>
<td>EJB</td>
<td>JSF</td>
</tr>
<tr>
<td>JAX-WS</td>
<td>JAF</td>
<td>JSP</td>
</tr>
<tr>
<td>JAXM</td>
<td>JavaMail</td>
<td>JSTL</td>
</tr>
<tr>
<td><strong>JAX-RS</strong></td>
<td>JCA</td>
<td>Servlet</td>
</tr>
<tr>
<td>JAXR</td>
<td>JMS</td>
<td>Expression Language 1.2</td>
</tr>
<tr>
<td>StAX</td>
<td>JPA</td>
<td></td>
</tr>
<tr>
<td>Web Services Metadata</td>
<td>JTA</td>
<td></td>
</tr>
<tr>
<td>Web Services 1.2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Management, Security and other

<table>
<thead>
<tr>
<th>JCDI</th>
<th>@Inject</th>
<th>JACC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common Annotations</td>
<td></td>
<td>RMI</td>
</tr>
<tr>
<td>Java EE Application Deployment</td>
<td></td>
<td>JMX</td>
</tr>
<tr>
<td>Java EE Management</td>
<td></td>
<td>JAAS</td>
</tr>
<tr>
<td>Java Authentication Service Provider Interface for Containers</td>
<td>1.0</td>
<td>JAXP...</td>
</tr>
<tr>
<td>Debugging Support for Other Languages</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>Bean Validation</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>

+ Java SE 6

<table>
<thead>
<tr>
<th>JAXB</th>
<th>JDBC</th>
<th>JNDI</th>
</tr>
</thead>
<tbody>
<tr>
<td>JRI</td>
<td>JMX</td>
<td>RMI</td>
</tr>
<tr>
<td>JAAS</td>
<td></td>
<td>JMX</td>
</tr>
<tr>
<td>JAXP...</td>
<td></td>
<td>JMX</td>
</tr>
</tbody>
</table>
Lighter : Profiles

Full Java EE 6

Web Profile

Profile X

Profile Y
Lighter : Web Profile

- Subset of full platform
- Focuses on web development
- Separate specification
- Others will come
  - Minimal (Servlet/JSP)
  - Portal....

Servlet 3.0
JSP 2.2
EL 1.2
JSTL 1.2
EJB Lite 3.1
JTA 1.1
JPA 2.0
JSF 2.0
Bean Validation 1.0

« ...you'll see gradual move toward the Web profile » - Rod Johnson
Lighter : EJB Light

- Subset of the EJB 3.1 API
- To be used in Web profile

Local Session Bean
Injection
CMT / BMT
Interceptors
Security

Message Driven Beans
EJB Web Service Endpoint
RMI/IIOP Interoperability
Remote interface
EJB 2.x
Timer service
CMP / BMP
Lighter : Pruning (Soon less specs)

• Makes some specifications optional in next version
• Pruned in Java EE 6
  – Entity CMP 2.x
  – JAX-RPC
  – JAX-R
  – JSR 88 (Java EE Application Deployment)
• Stronger than @Deprecated
• Might disappear from Java EE 7
  – Evolve (or not) separately from Java EE
• Easier for future containers
Easier?

Of course!
Ease of development

- Annotations based programming model
  - `@WebServlet`
  - `@ServletFilter`
  - `@WebServletContextListener`
  - `@InitParam`
- Deployment descriptors optional (web.xml)
  - Modular
Ease of development

public class MyServlet extends HttpServlet {
    public void doGet (HttpServletRequest req, HttpServletResponse res) {
        ....
    }
}

Deployment descriptor (web.xml)

<web-app>
    <servlet>
        <servlet-name>MyServlet</servlet-name>
        <servlet-class>samples.MyServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/MyApp</url-pattern>
    </servlet-mapping>

    ...
</web-app>
Ease of development

@WebServlet(urlMappings={"/MyApp"})
public class MyServlet extends HttpServlet {

    public void doGet (HttpServletRequest req, HttpServletResponse res){
      ....
    }
}

web.xml is optional

- Same for filters and listeners
Extensibility

- Fragments or modular web.xml
  - Logical partitioning of a web application
- Annotations and web fragments are merged
- Overridden by main web.xml

```xml
<web-fragment>
  <servlet>
    <servlet-name>myservlet</servlet-name>
    <servlet-class>samples.MyServlet</servlet-class>
  </servlet>
  <listener>
    <listener-class>samples.MyListener</listener-class>
  </listener>
</web-fragment>
```
Asynchronous support

- Servlets have to wait for a response from:
  - Web service
  - JDBC connection
  - JMS message....

- Comet style of programming
- @WebServlet (asyncSupported = true)
- New APIs to ServletRequest / ServletResponse
  - Suspending, resuming, querying the status of the request
JSF 2.0

Presentation Tier
- Servlet 3.0
- JSP 2.2
- JSF 2.0

Business Logic Tier
- EJB 3.1
- JTA 1.1

Persistence Tier
- Bean Valid 1.0
- JPA 2.0

Interoperability Tier
- JAX-WS 2.2
- JAX-RS 1.1
- JMS 1.1

Injection

Java EE 6
Ease of development

- Annotations
  - @ManagedBean
  - @ApplicationScoped, @SessionScoped...
  - @FacesConverter
  - @FacesValidator
- faces-config.xml optional
- Page declaration language (PDL)
  - Facelets (preferred)
  - JSP (still supported)
- Easier resources management
- Easier way of Component Development
- Ajax support
Managed bean

```java
public class DatabaseUtil {

    private Cities cities;
    ...
}
```

```xml
<managed-bean>
    <managed-bean-name>dbUtil</managed-bean-name>
    <managed-bean-class>server.DatabaseUtil</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
    <managed-property>
        <property-name>cities</property-name>
        <value>#{cities}</value>
    </managed-property>
</managed-bean>
```
Managed bean

```java
@ManagedBean(name="dbUtil")
@ApplicationScoped
public class DatabaseUtil {

    @ManagedProperty(value="#{cities}")
    private Cities cities;
}
```

faces-config.xml is optional

- Same for converters and validators
Composite component

- No Java code needed
- Use XHTML and JSF tags to create components
- Like Java programming you need:
  - «an interface»
  - «an implementation»
Composite component

<html>
<composite:interface>
  <composite:attribute name="item" required="true"/>
</composite:interface>

<composite:implementation>
  <tr>
    <td>Title : </td>
    <td>
      <h:inputText value="#{compositeComponent.attrs.item.title}"/>
    </td>
  </tr>
  <tr>
    <td>Description : </td>
    <td>
      <h:inputText value="#{compositeComponent.attrs.item.desc}" />
    </td>
  </tr>
</composite:implementation>
</html>
Ajax support

- Previous versions had no native Ajax solution
- Ajax support has been specified
- JavaScript library (jsf.js)
  - Several specified JavaScript functions
    - request, response, execute, render...
- Easier integration in your pages
Easier & Richer

- Optional Local Interfaces
- Singleton
- Asynchronous calls
- Cron-based Timer Service
- Packaging in a war
- Portable JNDI name
- Embeddable Container
- EJB Lite
Optional Local Interface

- `@Local`, `@Remote`
- Interfaces are not always needed
  - Only for local interfaces
  - Remote interfaces are not optional!

```java
@Stateless
public class HelloBean {

    public String sayHello() {
        return "Hello GeeCon";
    }
}
```
Singleton

- New component
  - Looks like a stateless / stateful
  - No/local/remote interface

- Follows the Singleton pattern
  - One single EJB per application per JVM

- Use to share state in the entire application
  - State not preserved after container shutdown

- Added concurrency management
  - @ConcurrencyManagement
public class CachingBean {

    private Map cache;

    @PostConstruct void init() {
        cache = ...;
    }

    public Map getCache() {
        return cache;
    }

    public void addToCache(Object key, Object val) {
        cache.put(key, val);
    }
}
Asynchronous calls

- How to have asynchronous call in EJBs?
- JMS is to send messages not to do asynchronous calls
- Threads are not allowed (don't integrate well)
- @Asynchronous
- Method returns `void` or `Future<T>`
  - `java.util.concurrent` package
Asynchronous calls

@Stateless
public class OrderBean {

    public void createOrder() {
        Order order = persistOrder();
        sendEmail(order);
    }

    public Order persistOrder() {...}

    @Asynchronous
    public void sendEmail(Order order) {...}
}

Packaging in a war

**foo.ear**
- `lib/foo_common.jar`
- `com/acme/Foo.class`
- `foo_web.war`
  - WEB-INF/web.xml
  - WEB-INF/classes
    - `com/acme/FooServlet.class`
- `foo_ejb.jar`
  - `com/acme/FooEJB.class`

**foo.war**
- WEB-INF/classes
  - `com/acme/Foo.class`
  - `com/acme/FooServlet.class`
  - `com/acme/FooEJB.class`
Portable JNDI Name

• Client inside a container (use DI)
  @EJB Hello h;

• Client outside a container
  Context ctx = new InitialContext();
  Hello h = (Hello) ctx.lookup(??);

• Portable JNDI name is specified
  java:global/env/foo/HelloEJB

  java:global/(app)/(module)/(bean)#(intf)
Embeddable Container

- API allowing to:
  - Initialize a container
  - Get container context
  - ...

- Can run in any Java SE environment
  - Batch processing
  - Simplifies testing
  - Just a jar file in your classpath
public class PlaceBidClient {
    public static void main(String[] args) throws Exception {

        EJBContainer container =
            EJBContainerFactory.createEJBContainer();

        Context context = container.getContext();

        Hello h = (Hello)
            context.lookup("java:global/app/foo/HelloEJB");

        h.sayHello;

        container.close();
    }
}
Timer Service

• Programmatic and Calendar based scheduling
  – « Last day of the month »
  – « Every five minutes on Monday and Friday »

• Cron-like syntax
  – second [0..59], minute[0..59], hour[0..23], year
  – DayOfMonth[1..31]
  – dayOfWeek[0..7] or [sun, mon, tue..]
  – Month[0..12] or [jan,feb..]
Timer Service

@Stateless
public class WakeUpBean {

    @Schedule(dayOfWeek="Mon-Fri", hour="9")
    void wakeUp() {
        ...
    }
}

- EJB Lite + Timer + Asynch calls + Embeddable Container = Batch processing
JPA 2.0

Presentation Tier

Servlet 3.0  JSP 2.2  JSF 2.0

Business Logic Tier

EJB 3.1  JTA 1.1

Persistence Tier

Bean Valid 1.0  JPA 2.0

Interoperability Tier

JAX-WS 2.2  JAX-RS 1.1  JMS 1.1

Injection

Java EE 6
JPA 2.0

- Java Persistent API
- Evolves separately from EJB now
  - JSR 317
- Can also be used in Java SE
- More mappings
  - JoinTable for OneToOne relationship
- Criteria API
- Standard properties in persistence.xml
- Simple Cache API
Collection of basic types

@Entity
Public class Item {

    @ElementCollection
    private Set<String> tags;
}

• Mapped in a separate table
Better Support of Map

@Entity
public class Department {
    ...
    @ElementCollection
    public Map<Integer, Employee> employees
    ...
}

- Basic types, Objects, Embeddables
- Mapped in a separate table
Locking Enhancement

```java
public enum LockModeType {
    OPTIMISTIC,
    OPTIMISTIC_FORCE_INCREMENT,
    PESSIMISTIC,
    PESSIMISTIC_FORCE_INCREMENT,
    NONE
}
```

- JPA 1.0 only support optimist locking
- Now Pessimist locking
- Multiple places to specify lock
  - Lock, read and lock, read then lock
Criteria API

• Used to define dynamic queries
• Like JPQL, Criteria API is based on Entities
• Allow the construction of an object-based graph
• Strongly typed
• Uses a metamodel
  • Each entity X has a metamodel class X_
@Entity
public class Customer {
    @Id Integer custId;
    String name;
    Address shippingAddress;
    ...
}

CriteriaQuery q = qb.create();
Root<Customer> customer = q.from(Customer.class);
q.select(customer.get(Customer_.shippingAddress)).
    where(q.equal(customer.get(Customer_.name),"Peter"));
JAX-RS 1.1

Presentation Tier
- Servlet 3.0
- JSP 2.2
- JSF 2.0
- EJB 3.1
- JTA 1.1

Business Logic Tier
- Bean Valid 1.0
- JPA 2.0

Persistence Tier
- Bean Valid 1.0
- JPA 2.0

Interoperability Tier
- JAX-WS 2.2
- JAX-RS 1.1
- JMS 1.1

Java EE 6
JAX-RS 1.1

- RESTful Services
- POJO and Annotations Based
- Data and functionality are considered resources
- Map HTTP

<table>
<thead>
<tr>
<th>HTTP</th>
<th>Action</th>
<th>HTTP</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Get a resource</td>
<td>PUT</td>
<td>Create or update</td>
</tr>
<tr>
<td>POST</td>
<td>Create a resource</td>
<td>Delete</td>
<td>Deletes a resource</td>
</tr>
</tbody>
</table>

- JAX-RS 1.0 has been released
Hello World

@Path("/helloworld")
public class HelloWorldResource {

    @GET
    @Produces("text/plain")
    public String sayHello() {
        return "Hello World";
    }

    }

• http://example.com/helloworld
Hello World

Request

```
GET /helloworld HTTP/1.1
Host: example.com
Accept: text/plain
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 12 Nov 2008 16:41:58 GMT
Server: Apache/1.3.6
Content-Type: text/plain; charset=UTF-8
Hello World
```
MIME Types

@Path("/helloworld")
public class HelloWorldResource {

    @GET @Produces("image/jpeg")
    public byte[] paintHello() {
        ...
    }

    @POST @Consumes("text/xml")
    public void updateHello(String xml) {
        ...
    }
}
Parameters

@Path("/users/{userId}"")
public class UserResource {

    @GET
    @Produces("text/xml")
    public String getUser(@PathParam("userId") String userName) {
        ...
    }
}

- http://example.com/users/Smith123
...to richer ones

Servlet 3.0  JSF 2.0  JAX-WS 2.2
EJB 3.1  JTA 1.1  JAX-RS 1.1
JPA 2.0  JMS 1.1

Java EE 6
Reference Implementations

• All these specs have reference implementations
  – GlassFish V3 : EJB 3.1 and Servlet 3.0
  – EclipseLink : Java Persistence API (JPA 2.0)
  – Jersey : RESTful Web Services (JAX-RS 1.0)
  – Metro : Web Services (JAX-WS 2.2)
  – JBoss Seam : JCDI 1.0
  – Mojarra : JSF 2.0

• And they are production ready
Summary

• Java EE 6 is
  – Simpler (POJO, annotation, less XML, Pruning)
  – Richer (more specifications)
  – Lighter (profiles, pruning, EJB lite)
  – Standard (no vendor locking)
  – Robust (10th anniversary)
  – Book out in June 2009

• Java EE 6 out in November 2009

« Forget the past, look to the future, Java EE 6 is the place to go... » - Antonio Goncalves
Q&A