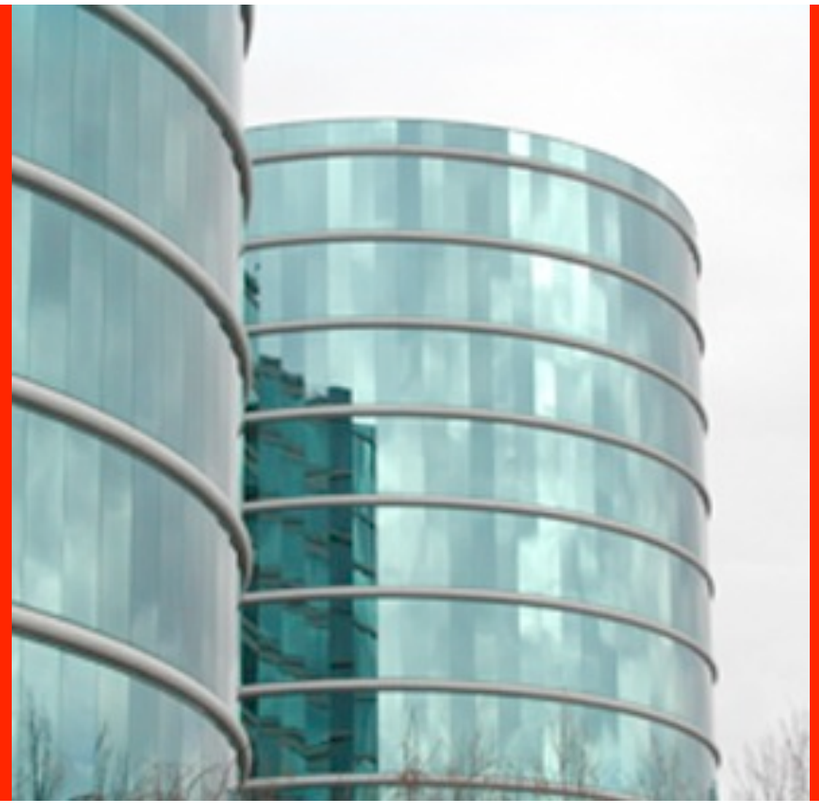

ORACLE



ORACLE®

The Java EE 6 Programming Model Explained

Alexis Moussine-Pouchkine
Oracle

Contents

- What's new in Java EE 6?
- The core programming model explained
- Features you should know about

What's new in the Java EE 6 Platform

New and updated components

- EJB 3.1(+Lite)
- JPA 2.0
- Servlet 3.0
- JSF 2.0
- JAX-RS 1.1
- Bean Validation 1.0
- DI 1.0
- CDI 1.0
- Connectors 1.6
- Managed Beans 1.0
- Interceptors 1.1
- JSP 2.2
- EL 2.2
- JSR 250 1.1
- JASPIC 1.1
- JACC 1.5
- JAX-WS 2.2
- JSR 109 1.3

Java EE 6 Web profile

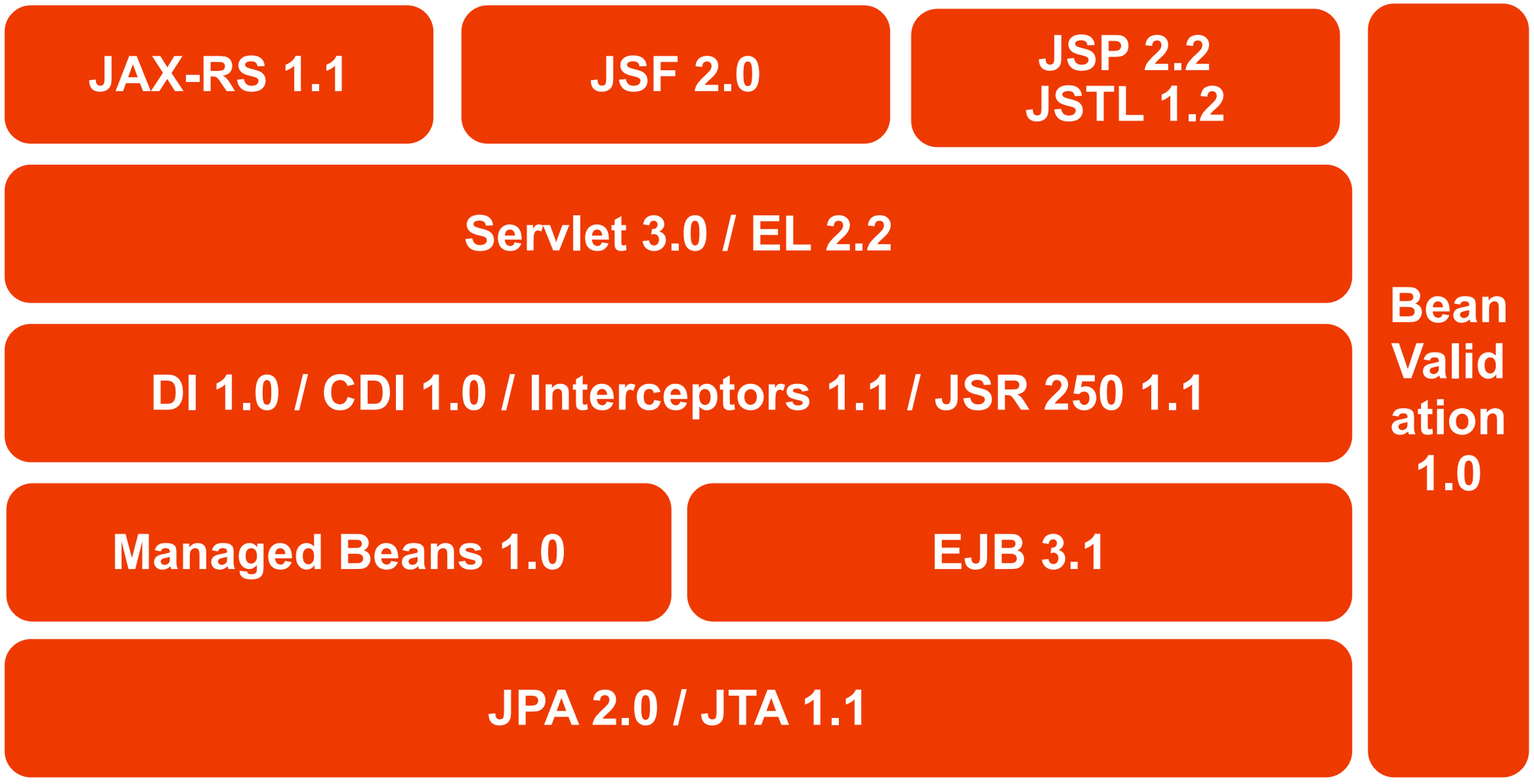
New and updated components and unchanged components

- EJB 3.1 Lite
- JPA 2.0
- Servlet 3.0
- JSF 2.0
- Bean Validation 1.0
- DI 1.0
- CDI 1.0
- Managed Beans 1.0
- Interceptors 1.1
- JSP 2.2
- EL 2.2
- JSR 250 1.1
- JSR-45 1.0
- JSTL 1.2
- JTA 1.1

What's New?

- Several new APIs
- Web Profile
- Pluggability/extensibility
- Dependency injection
- Lots of improvements to existing APIs

The Core Programming Model Explained



EJB 3.1 Lite

- Session bean programming model
  - Stateful, stateless, singleton beans
- Declarative security and transactions
- Simplified packaging in WAR files
- Annotation-based
- Descriptor OK but optional

Managed Beans

- Plain Java objects (POJOs)
- Foundation for other component types
- Fully support injection (`@Resource`, `@Inject`)
- `@PostConstruct`, `@PreDestroy` callbacks
- Optional name
- Interceptors

Sample Managed Beans

```
@ManagedBean
public class A {
    @Resource
    DataSource myDB;

    public doWork() {
        // ...
    }
}
```

```
@ManagedBean("foo")
public class B {
    @Resource A a;

    @PostConstruct
    init() { ... }
}
```

EJB Components are Managed Beans too!

- All managed beans core services available in EJBs
- EJB component model extends managed beans
- New capabilities engaged on demand via annotations
  - @Stateful, @Stateless, @Singleton
  - @TransactionAttribute
  - @RolesAllowed, @DenyAll, @PermitAll
  - @Remote
  - @WebService

Incremental Programming Model

- Start with POJOs (@ManagedBean)
- @Use all the basic services
- Turn them into session beans as needed
- Take advantage of new capabilities
- No changes to (local) clients

Incremental Programming Model

- Start with POJOs (@ManagedBean)
- @Use all the basic services
- Turn them into session beans as needed
- Take advantage of new capabilities
- No changes to (local) clients

```
public class A {
    @Resource DataSource myDB;
    // ...
}
```

Incremental Programming Model

- Start with POJOs (@ManagedBean)
- @Use all the basic services
- Turn them into session beans as needed
- Take advantage of new capabilities
- No changes to (local) clients

```
@ManagedBean
public class A {
    @Resource DataSource myDB;
    // ...
}
```

Incremental Programming Model

- Start with POJOs (@ManagedBean)
- @Use all the basic services
- Turn them into session beans as needed
- Take advantage of new capabilities
- No changes to (local) clients

```
public class A {
    @Resource DataSource myDB;
    // ...
}
```

Incremental Programming Model

- Start with POJOs (@ManagedBean)
- @Use all the basic services
- Turn them into session beans as needed
- Take advantage of new capabilities
- No changes to (local) clients

@Stateful

```
public class A {
    @Resource DataSource myDB;
    // ...
}
```

Context and Dependency Injection (CDI)

- Managed beans on steroids
- Opt-in technology on a per-module basis
  - `META-INF/beans.xml`
  - `WEB-INF/beans.xml`
- `@Resource` only for container-provided objects
- Use `@Inject` for application classes

```
@Inject @LoggedIn User user;
```

Context and Dependency Injection (CDI)

- Managed beans on steroids
- Opt-in technology on a per-module basis
  - `META-INF/beans.xml`
  - `WEB-INF/beans.xml`
- `@Resource` only for container-provided objects
- Use `@Inject` for application classes

```
@Inject @LoggedIn User user;
```

The **type** describes the capabilities (methods)

Context and Dependency Injection (CDI)

- Managed beans on steroids
- Opt-in technology on a per-module basis
  - `META-INF/beans.xml`
  - `WEB-INF/beans.xml`
- `@Resource` only for container-provided objects
- Use `@Inject` for application classes

```
@Inject @LoggedIn User user;
```

Qualifiers describes qualities/characteristics/identity

APIs That Work Better Together
Example: Bean Validation

- Bean Validation integrated in JSF, JPA
- All fields on a JSF page validated during the "process validations" stage
- JPA entities validated at certain lifecycle points (pre-persist, post-update, pre-remove)
- Lower-level validation API equivalent integration with other frameworks/APIs

Uniformity

Combination of annotations "just work"

- Example: JAX-RS resource classes are POJOs
- Use JAX-RS-specific injection capabilities
- But you can turn resource classes into managed beans or EJB components ...
- ... and use all new services !

JAX-RS Sample
Using JAX-RS injection annotations

```
@Path("root")
public class RootResource {
    @Context UriInfo ui;

    public RootResource() { ... }

    @Path("{id}")
    public SubResource find(@PathParam("id") String id) {
        return new SubResource(id);
    }
}
```

JAX-RS Sample

As a managed bean, using @Resource

```
@Path("root")
@ManagedBean
public class RootResource {
    @Context UriInfo ui;
    @Resource DataSource myDB;

    public RootResource() { ... }

    @Path("{id}")
    public SubResource find(@PathParam("id") String id) {
        return new SubResource(id);
    }
}
```

JAX-RS Sample

As an EJB, using @Resource and security annotations

```
@Path("root")
@Stateless
public class RootResource {
    @Context UriInfo ui;
    @Resource DataSource myDB;

    public RootResource() { ... }

    @Path("{id}")
    @RolesAllowed("manager")
    public SubResource find(@PathParam("id") String id) {
        return new SubResource(id);
    }
}
```

JAX-RS Sample

As a CDI bean, using @Inject and scope annotations

```
@Path("root")
@ApplicationScoped
public class RootResource {
    @Context UriInfo ui;
    @Inject @CustomerDB dataSource myDB;
    @Inject @LoggedIn User user;

    public RootResource() { ... }

    @Path("{id}")
    public SubResource find(@PathParam("id") String id) {
        // ...
    }
}
```

How to Obtain Uniformity

Combination of annotations "just work"

- Annotations are additive
- New behaviors refine pre-existing ones
- Strong guidance for new APIs

Classes can evolve smoothly

Extensibility

- Level playing field for third-party frameworks
- Pluggability contracts make extensions look like built-in technologies
- Two main sets of extension points :
  - Servlet 3.0 pluggability
  - CDI extensions

Servlet 3.0 Pluggability

- Drag-and-drop model
- Web frameworks as fully configured libraries
  - Contains "fragments" of `web.xml`
  - `META-INF/web-fragment.xml`
- Extensions can register listeners, servlets, filters dynamically
- Extensions can discover and process annotated classes

```
@HandlesTypes(WebServlce.class)
public class MyWebServiceExtension
    implements ServletContainerInitializer { ... }
```

Extensibility in CDI

- Drag-and-drop model here too
- Bean archives can contain reusable sets of beans
- Extensions can process any annotated types and define beans on-the-fly
- Clients only need to use `@Inject`
  - No `XYZFactoryManagerFactory` horrors !

```
@Inject @Reliable @PayBy(CREDIT_CARD) PaymentProcessor;
```

Using Extensibility

- Look for ready-to-use frameworks, extensions
- Refactor your libraries into reusable, auto-configured frameworks
- Take advantage of resource jars
- Package reusable beans into libraries

Features you should know about

Resource JARs
(aka `WEB-INF/lib/{*.jar}/META-INF/resources`)

- Modular web applications with static content
  - Images, CSS, JavaScript (think dojo, jquery, ...)
- a JAR placed in **`WEB-INF/lib`** has static content from its **`META-INF/resource`** directory accessible from the application web-context root :

```
WEB-INF/lib/pictures.jar/META-INF/resources/foo.png
WEB-INF/lib/styles.jar/META-INF/resources/foo.css
WEB-INF/lib/scripts.jar/META-INF/resources/foo.js

http://host:port/webcontext/foo.png
http://host:port/webcontext/foo.css
http://host:port/webcontext/foo.js
```

Interceptors and Managed Beans

- Interceptors can be engaged directly

```
@Interceptors(Logger.class)
@ManagedBean
public class A { ... }
```

- This kind of strong coupling is a code smell

Interceptor Bindings

- With CDI, you can use interceptor bindings

```
@Logged
@ManagedBean
public class A { ... }
```

where

```
@InterceptorBinding
public @interface Logged { ... }
```

and

```
@Interceptor
@Logged
public class Logger {
  @AroundInvoke Object foo(InvocationContext c) {...}
}
```

Interceptors vs. Decorators

- Decorators are type-safe
- Decorators are bound unobtrusively

```
@Decorator
class ActionLogger implements Loggerable {

    @Inject @Delegate Action action;

    @Override
    Result execute (Data someData) {
        // ... log the action here ...
        return action.execute(someData);
    }
}
```

Two ways to Connect Beans in CDI

- Dependencies (`@Inject`)
- (Somewhat) strongly coupled
- Clients hold references to beans

```
@Inject @Reliable OrderProcessor processor;
```

Two ways to Connect Beans in CDI

- Dependencies (`@Inject`)
- (Somewhat) strongly coupled
- Clients hold references to beans

```
@Inject @Reliable OrderProcessor processor;
```

- Events (`@Observes`)
- Loosely coupled
- Observers don't hold references to event sources

Two ways to Connect Beans in CDI

- Dependencies (`@Inject`)
- (Somewhat) strongly coupled
- Clients hold references to beans

```
@Inject @Reliable OrderProcessor processor;
```

- Events (`@Observes`)
- Loosely coupled
- Observers don't hold references to event sources

```
void onProcesses(@Observes OrderProcessesEvent e)
{ ... }
```

Events are Transaction-Aware

- Events queued and delivered near at boundaries
- Specify a `TransactionPhase`
  - `BEFORE_COMPLETION`
  - `IN_PROGRESS`
  - `AFTER_COMPLETION`
  - `AFTER_FAILURE, AFTER_SUCCESS`
- Supersedes TransactionSynchronizedRegistry API
- Mix transactional and non-transactional observers

```
public void
onProcesses(@Observes(during=AFTER_SUCCESS)
                OrderProcessedEvent e) { ... }
```

Global JNDI Names

- Standard names to refer to (remote) beans in (other) applications
- `java:global/`*`application/module/bean!interface`*
- Also used to lookup EJB component within the standalone container API (`EJBContainer`)

What is java:global ?

- New JNDI namespaces to match the packaging hierarchy

```
java:global
java:app
java:module
java:comp (legacy)
```

- Share resources as widely as possible

```
java:app/env/customerDB
```

- Refer to resources from any module in the application

```
@Resource(lookup="java:app/env/customerDB")
DataSource db;
```

DataSource Definitions

- New annotation `@DataSourceDefinition`
- Define a data source once for the entire application

```
@DataSourceDefinition(
        name="java:app/env/customerDB",
        className="com.acme.ACMEDataSource",
        portNumber=666,
        serverName="acmeStore.corp")
@Singleton
public class MyBeans {}
```

Using Producer Fields for Resources

- Expose container resources via CDI

```
public class MyResources {
   @Produces
   @Resource(lookup="java:app/env/customerDB")
   @CustomerDB DataSource ds;

   @Produces
   @PersistenceContext(unitName="payrollDB")
   @Payroll EntityManager em;
}
```

Using Producer Fields for Resources

- Expose container resources via CDI

```
public class MyResources {
   @Produces
   @Resources(lookup="java:app/env/customerDB")
   @CustomerDB DataSource ds;

   @Produces
   @PersistenceContext(unitName="payrollDB")
   @Payroll EntityManager em;
}
```

- Clients use @Inject

```
@Inject @CustomerDB DataSource customers;
@Inject @Payroll EntityManager payroll;
```

The Verdict
A Better Platform

- Less boilerplate code than ever
- Fewer packaging headaches
- Uniform use of annotations
- Sharing of resources to avoid duplication
- Transaction-aware observer pattern for beans
- Strongly-typed decorators
- ... and more!

Resources

- http://www.oracle.com/javaee

- http://youtube.com/user/GlassFishVideos
- http://blogs.sun.com/theaquarium
- http://glassfish.org

Tonight

- JavaEE BOF @ C3
  - Java EE 6 Adoption
  - Features & Plans for Java EE 7
  - <Your favorite topic/>

ORACLE®

alexis.moussine-pouchkine@oracle.com
@alexismp

http://blogs.sun.com/alexismp
http://blogs.sun.com/theaquarium