Discover the Power and Elegance of "Java Contexts and Dependency Injection" (Web Beans)

Magnus Kastberg
Java Application Architect

NASDAQ OMX℠

Biography – Magnus Kastberg

- Java Architect at NASDAQ OMX in Stockholm
- Over 12 years experience building Java / Java EE based systems
- Prior NASDAQ OMX I worked 7 years for Sun Microsystems as a Java developer and architect, and 2 years for CIBER
- Currently spend my days building financial, business-critical Java EE based systems

Background

- Java EE 5 and EJB 3.0 made it a lot simpler to develop Java EE apps, but still some problems…
- There is a split between web tier and business tier technologies…
  - Unnecessary complicated to access EJB components (JNDI lookup…)
  - Complicated to share state between components (EJB components are not aware of web-tier contexts)
- A general dependency injection mechanism needed

What is ”Java Contexts and Dependency Injection”?

- JSR 299 (spec lead Gavin King)
- Advanced typesafe dependency injection (DI) service
- Injection of different Java EE components and resources
- Allow different Java EE components to be bound to a context
- Container handles injection and lifecycle management of components
- Integration with Unified Expression Language making is possible to use a component within a JSF or JSP page
- Events
- SPI that allows non-platform technologies to integrate with the container, for example alternative web presentation technologies
- Influenced mostly by JBoss Seam and Google Guice

Supported environments

- Java EE 6 containers
- Embeddable EJB Lite containers $\Rightarrow$ Use in Java SE
- Java EE 5 containers optional

Bean implementations

<u>Bean implementations</u>

- Simple beans (plain Java classes)
- EJB session/singleton beans
- Resources
  - Java EE resources (JDBC datasource)
  - Entity managers
  - Remote EJBs
  - Webservice references
- JMS resources (queues and topics)

You can implement support for other kinds of Beans!

Bean definition

Different attributes can be declared on a bean which serves as input to the dependency injection mechanism and context management.

<u>Bean attributes:</u>

- Bean types
- Binding types
- Deployment type
- Scope
- Bean Name
- Bean Implementation

The attributes may either be:
-declared by using Java annotations
-declared in beans.xml
-defaulted by the container

NASDAQ OMX℠

Binding types

- A binding type lets a client choose between multiple implementations of an API
- The client don’t want to specify the implementation class!
- The default binding type is @Current
- You specify own binding types by using @BindingType annotation

```
@BasicLogin
public class BasicLoginManager implements LoginManager {
    public void login(String username, String password) { ... }
}
@SecureLogin
public class SecureLoginManager implements LoginManager {
    public void login(String username, String password) { ... }
}
```

```
@BasicLogin LoginHandler login;
```

Injection of BasicLoginManager Bean

```
@SecureLogin LoginHandler login;
```

Injection of SecureLoginManager Bean

Deployment type

- Represents different deployment scenarios (test, production, etc)
- Makes it really easy to switch implementations of different bean types at deployment-time
- The built-in deployment types are @Production and @Standard
- You specify own deployment types by using @DeploymentType

```
@Mock
public class BasicLoginManager implements LoginManager { … }

@Production
public class SecureLoginManager implements LoginManager { … }


@Current LoginManager login;
login.login(…);
```

Injection of BasicLoginManager Bean

```
<Deploy>
   <Standard/>
   <Production/>
   <Mock/>
</Deploy>
```

Highest precedence

beans.xml

Bean scope

- All beans have a scope which is associated with a context
- A context handles the lifecycle of all bean instances with a specific scope
- The built-in scopes are:
  - @RequestScoped, @SessionScoped, @ApplicationScoped
  - @ConversationScoped
  - @Dependent (default)
- All scopes except @Dependent are ”normal” scopes
- An injected Bean instance with @Dependent scope is bound to the client, it is never shared between multiple injection points

Bean Name

- A bean can be given a name with the @Named annotation
- A bean may be referred to by its name only in Unified EL expressions
- Allows a bean, including EJB session beans, to be used directly in a JSP or JSF page!

Example bean:

```
@Named("password")
@SessionScoped
@Stateful
public class PasswordManager {
  public void setOld(String old) { … }
  public void setNew(String new) { … }
  public void update() { em.merge(…); }
}
```

Example JSF page:

```
<h:inputText value="#{password.old}"/>
<h:inputText value="#{password.new}"/>
<h:commandButton value="Change pwd" action="#{password.update}"/>
```

Typesafe Dependency injection

- When matching a bean to an injection point, the container considers:
  - Bean type
  - Binding types
  - Deployment type precedence
- When matching a bean in Unified EL expressions, the container considers:
  - Bean Name
  - Deployment type precedence

Bean integration

- A bean can interoperate with another bean (using DI)
- Any type of EJB can interoperate with a bean (using DI)
- A Servlet can interoperate with a bean (using DI)
- JSP and JSF pages can interoperate with beans (using Unified EL expressions)

Events

- Beans may interact via events in a completely decoupled way (no compile-time dependency between the producer and consumer beans)
- An event consumer observes events of a specific event type and a specific set of event binding types
- An observer method is defined via the @Observes annotation
- Event observers may receive events asynchronously using @Asynchronously
- Event types may be mapped to JMS topics for distributed events sent between different processes

Events (continued...)

Example (event producer firing an event)

Event object

Implementation of CreditCard event binding type

```
public void pay() {
  Payment payment = …;
  manager.fireEvent(new PaymentDoneEvent(payment), new CreditCardBinding() {});
}
```

Example (event consumer observing the event)

```
public void afterCreditCardPayment(@Observes @CreditCard PaymentDoneEvent event) {
  Payment payment = event.getPayment();
  …
}
```

Observed event binding type

Observed event type

More Info

- JSR 299
  - http://jcp.org/en/jsr/detail?id=299
- Gavin King’s Blog:
  - http://in.relation.to/Bloggers/GavinsBlog/Tag/Web+Beans

Questions?

I will be here today…or

Email: magnus.kastberg@gmail.com