CDI: Context and Dependencies Injections (>= JEE6)

CDI *(alias JSR 299 alias **WebBean**)* constitue l'**un des principaux apports de JEE6** vis à vis de JEE5.

Les spécifications "**CDI**" visent à mettre en œuvre des injections de dépendances automatiques entre des composants (de conteneurs éventuellement différents).
Un "EJB 3.1 session" pourra par exemple être injecté dans un "managedBean" de JSF .

Attention: CDI (alias JSR-299) utilise en interne DI / JSR-330 (@Inject et @Named) mais va plus loin en apportant bien plus de fonctionnalités (dans un environnement JEE6) .

Les fonctionnalités additionnelles de CDI par rapport à DI sont essentiellement :

- la prise en charge des contextes (associés aux scopes "request", "session" , "application" et "conversation")
- l'association automatique "EJB session stateful" avec session Http .
- des injections transparentes entre EJB et JavaBean web (jsf,...) .
- prise en charge de certains événements
- prise en charge des intercepteurs (décorateurs/aop) .
- une approche fortement typée et avec un très faible couplage

En un mot les spécifications CDI / WebBeans tentent d'unifier harmonieusement le monde des EJB 3.x (gérant les transactions et la persistance) avec le monde du Web (Servlet/JSP/JSF) .

On reconnaît l'utilisation de CDI à la présence d'un fichier "**beans.xml**" (éventuellement vide mais obligatoire) dans les répertoires **WEB-INF** (web) ou **META-INF** (ejb) .

Pour l'instant, les deux principales implémentations des spécifications CDI sont :

- **Weld** (de Jboss)
- **OpenWebBeans** (de Apache)

Beaucoup de principes de CDI/JSR-299 proviennent du framework Seam (V1,V2) de Jboss .

Maintenant que CDI/JSR-299 est normalisé au niveau de JEE6 , **Seam** V3 s'intègre parfaitement dans JEE6 en utilisant à la lettre CDI/JSR-299 (et n'est plus à considérer comme un framework propriétaire mais comme une **extension portable pour JEE6**) .

Le sous ensemble JSR-330 (DI)

JSR-330 (**D**ependencies **I**njections) [API packagée dant ***java-inject.jar***] est essentiellement constituée de deux annotations fondamentales normalisées :

- **@Named** (pour identifier/nommer ce qui pourra être injecté)
- **@Inject** (pour effectuer une injection de dépendance)

et de quelques ***méta-annotations*** qualificatives (**@Qualifier , @Scope , …**)

Ces annotations ont le mérite de constituer les bases d'un **standard** assez universel dans le monde java récent (***@Named et @Inject sont utilisables au niveau de Spring 3 , de Seam 3 et de JEE6***).

….

Anatomies des liaisons/injections (JSR 299)

Les injections automatiques seront paramétrées par une série de critères qui devront être mis en concordance entre :

- un élément/composant potentiellement injectable
- une référence sur une dépendance à injecter.

Critères influençant les liaisons par injections :

Api type : type java (souvent une Interface , éventuellement une classe)

Qualifier (qualificatif): Annotation spécifique (définie par l'utilisateur/développeur) (ex : @Variante1 , @Variante2) et elle même annotée par la **méta-annotations @Qualifie**r .

Les **"qualificatifs" par défaut** sont **@Any** (du coté composant injectable) et **@Default @Any**(du coté référence à initialiser) .

Alternative de déploiement (une ou plusieurs ***@Alternative*** dans le code) et alternative à utiliser (à la place de @Default) déclarée au sein de ***<alternatives>*** dans *beans.xml*.

Portée/Scope (@RequestScope, @SessionScope , ….) sachant qu'un composant d'une portée globale/longue_durée_de_vie peut être injecté dans un composant d'un scope plus étroit/plus éphémère et non l'inverse .

Correspondance de nom (entre @Named() de web bean et EL in JSF2 , JSP2)

@Named(***"webBeanXY"***) <---> ***#{webBeanXY***.ppp**}**

Rappel : le nom donné par défaut par @Named à un composant WebBean est le nom de la classe java (en remplacement le premier caractère par une minuscule).

Qualifier

On peut définir de nouveaux qualificatifs (personnalisés) en créant de nouvelles annotations elles mêmes basées sur la méta-annotation @Qualifier.

Exemple :

```
package tp.myapp.web.cdi.qualifier;
```

```
import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.PARAMETER;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import javax.inject.Qualifier;

@Retention(RetentionPolicy.RUNTIME)
@Target({TYPE,METHOD,FIELD,PARAMETER})
@Qualifier
public @interface English { }
```

et

```
@Retention(RetentionPolicy.RUNTIME)
@Target({TYPE,METHOD,FIELD,PARAMETER})
@Qualifier
public @interface French {}
```

utilisation possible :

```
public interface MessageGenerator {
	public String messageFromInput(String input);
}
```

```
@Named //javax.inject
@ApplicationScoped //javax.enterprise.context
@English
//@Default //javax.enterprise.inject
public class MsgGenV1 implements MessageGenerator {

	public String messageFromInput(String input) {
		return "hello " + input;
	}
}
```

```
@Named //javax.inject
@ApplicationScoped //javax.enterprise.context
@French
//@Any //(by default)
public class MsgGenV2 implements MessageGenerator {

	public String messageFromInput(String input) {
		return "bonjour " + input;
	}
}
```

```
@Named
@SessionScoped
public class MySessionBean implements Serializable{

	private static final long serialVersionUID = 1L;
```

```
	private String name; //+get/set
	private String message; //+get/set

	@Inject
	//@English
	@French
	//@Default by default
	private MessageGenerator msgGen;

	public String doAction(){
		message = msgGen.messageFromInput(name);
		System.out.println("message: "+message);
		return null;
	}
}
```

- Si le qualificatif @French est mentionné , la version française (MsgGenV2) est injectée/utilisée.
- Si le qualificatif @English est mentionné , la version anglaise (MsgGenV1) est injectée/utilisée.
- Si aucun qualificatif n'est mentionné , une exception est levée dès le démarrage de (en cas d'ambiguïté) .
- Si la version anglaise est doublement qualifiée (@English et **@Default**) et que la version française reste simplement qualifiée , il n'y plus d'ambiguïté , c'est la version par @Default qui sera injectée/utilisée .
- @Default et @Any sont prédéfinies dans le package `javax.enterprise.inject`

@Produces (pour méthode de production d'instances à injecter)

```
//@ Named  n'est pas indispensable
//@ ApplicationScoped  n'est pas indispensable
public class MsgGenAutomaticFactory {
	private Random r = new Random();

	@Inject @French
	private MessageGenerator msgGenFr;

	@Inject @English
	private MessageGenerator msgGenEn;

	@Produces @Default
	public MessageGenerator getMsgGen(){
		MessageGenerator msgGen=null;
		int n = r.nextInt() % 2;
		msgGen = (n==0) ? msgGenEn : msgGenFr;
		System.out.println("msgGen build by producer: "
				+ msgGen.getClass().getSimpleName());
		return msgGen;
	}
}
```

Utilisation possible:

```
@SessionScoped // or @ RequestScoped
public class MyManagedBean implements Serializable{
	...
	@Inject
	//@Default by default
	private MessageGenerator msgGen;

	public String doAction(){
		message = msgGen.messageFromInput(name);
		return null;
	}
}
```

NB: selon que l'injection soit effectué dans un bean de scope @RequestScoped ou @SessionScoped , la méthode de production sera invoquée une ou plusieurs fois.

La fabrication/production de l'instance peut prendre tout un tas de formes :

- statique (en masquant les indirections ou l'utilisation d'une fabrique ordinaire)
- pré-fabriquée (dans pool ou ...)
- selon alternative (au runtime)
- selon requête dynamique (entityManager.createQuery("....").getSingleResult())
- ….

Scopes (prédéfinis et extensions)

Les annotations de types "Scope" prédéfinies (dans *javax.enterprise.context*) sont :

- **@RequestScoped**
- **@SessionScoped**
- **@ApplicationScoped**
- **@ConversationScoped** ***(spécifique JSF request <= conversation <= session)***
- **@Dependent** ***(par défaut , selon contexte du bean contenant la référence)***

Méta annotation "Scope" pour éventuellement définir de nouveaux scopes :

```
@Retention(RUNTIME)
@Target({TYPE, METHOD})
@Scope //de javax.inject
public @interface MyNewScope {}
```

Encore faut-il associer une certaine sémantique à ce nouveau scope .

==> Pour de futures interrogations/réflexions .

Alternatives (de remplacement , explicitées dans beans.xml)

```
public interface NewsGenerator {
	public String getLastNews();
}
```

```
@Named
@ApplicationScoped
@Default
public class NewsGen implements NewsGenerator {
	public String getLastNews() {
		return "fresh news";
	}
}
```

```
@Named
@ApplicationScoped
@Alternative
public class AlternativeNewsGen implements NewsGenerator {
	public String getLastNews() {
		return "alternative news";
	}
}
```

et selon WEB-INF/beans.xml

```
<!--  config file for CDI (JEE6 / JSR299) in WEB-INF or META-INF (ejb) -->
<beans xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">

	<alternatives>
		<class>....</class> <class>....</class>
	</alternatives>
</beans>
```

avec

```
 <alternatives>
		<!-- <class>tp.myapp.web.cdi.news.AlternativeNewsGen</class> -->
	</alternatives>
```

ou bien

```
 <alternatives>
		<class>tp.myapp.web.cdi.news.AlternativeNewsGen</class>
	</alternatives>
```

la version qui sera injectée/utilisée sera :

- NewsGen *(par défaut)*
- ou bien AlternativeNewsGen (en remplacement).

Noms JNDI normalisés pour EJB 3.1 (depuis JEE6)

Les noms JNDI des EJB 3.0 dépendaient du serveur d'application hôte (Jboss, WebLogic, WebSphere, …) et n'étaient donc pas portables.

Les spécifications EJB 3.1 (JEE6) précisent (enfin) des noms JNDI portables (et de niveau global) :

Le nom JNDI d'un EJB3.1 doit être au format suivant:

"**java:global**/<app-name>/<module-name>/<bean-name>[!<fully-qualified-interface-name>]"

Ce nom complet global sera accompagné des deux alias suivants

"**java:app**/<module-name>/<bean-name>[!<fully-qualified-interface-name>]"

"**java:module**:<bean-name>[!<fully-qualified-interface-name>]"

utilisables depuis la même application (ou depuis le même module).

La partie !<fully-qualified-interface-name> est quelquefois facultative (lorsqu'il n'y a qu'une seule interface au niveau d'un EJB?).

NB: Tous ces noms "jndi globaux" (au niveau d'un serveur d'application) ne sont pas vus tels quels depuis l'extérieur (depuis une autre JVM/autre serveur) .

Par exemple au niveau du serveur jboss 7 , un accès externe à un ejb 3.1 doit se faire via un nom jndi du type :

"**ejb:**<app-name>/<module-name>/<distinct-name>/<bean-name>!<fully-qualified-classname-of-the-remote-interface>"

et lorsque distinct-name est vide comme souvent cela donne

"**ejb:**<app-name>/<module-name>//<bean-name>!<fully-qualified-classname-of-the-remote-interface>"

==> à priori , il vaudrait mieux utiliser des accès distants SOAP plutôt que RMI lorsque c'est possible .