



Paramétrage des EJB3 en XML

Fini les fichiers de paramétrage XML! L'introduction des annotations, utilisées conjointement avec la pratique dite de *l'agressive defaulting*, ont permis de les éliminer. Les informations sont directement dans le code sous la forme d'annotations.

Cela ne doit pas faire oublier que les applications JEE sont prévues au départ pour permettre la répartition des responsabilités entre plusieurs rôles. Le rôle du développeur, qui réalise des composants, ou bien une application complète, et celui du déployeur, qui installe et paramètre l'application dans un environnement donné, voire souvent dans plusieurs environnements (tests fonctionnels, tests d'intégration, pré-production, production par exemple). Or le paramétrage suppose de pouvoir modifier les valeurs de certains paramètres de l'application sans avoir à recompiler. Heureusement, il a été prévu de satisfaire aux deux catégories de besoins : les valeurs spécifiées dans les annotations que le déployeur a besoin de modifier peuvent l'être en ajoutant un fichier de paramétrage XML. Ce fichier de paramétrage pourra ne contenir que les valeurs à surcharger, il sera inutile de redéfinir la totalité des paramètres.

Paramètres de type simple sur EJB3 session

Prenons une application JEE déployée sous la forme d'un serveur par région. Supposons que l'application dialogue avec une autre application, via un protocole quelconque, mais que ce serveur, bien entendu, est aussi régional. Par conséquent, le nom du serveur avec lequel l'application communique devra figurer dans un fichier de paramétrage texte, facile à modifier. Cela signifie-t-il qu'il faille revenir aux fichiers `properties` lus à la main depuis le code d'un EJB ? En dehors du fait que cette façon de faire n'est pas recommandée, elle engendre un supplément de code dans les EJB. En fait, les EJB3 ont conservé les possibilités de paramétrage disponibles dans les versions antérieures, la principale différence étant qu'en l'absence de ce paramétrage, la valeur indiquée dans le code est utilisée par défaut. La quasi-totalité des informations présentes dans les annotations peuvent être modifiées par simple paramétrage, sans toucher au code de l'application. Dans notre exemple, nous avons un EJB 3 session répondant à l'interface suivante :

```
import javax.ejb.Remote;
@Remote
public interface Service {
    void sendMessageToOtherApplication(String message);
}
```

associé à la classe :

```
import javax.annotation.Resource;
import javax.ejb.Stateless;

@Stateless
public class ServiceBean implements Service {

    String serverName = "testserver";
    public void sendMessageToOtherApplication(String message) {
        // ... pour illustration uniquement ...
    }
}
```

```
System.out.println("serverName = "+serverName);
}
}
```

Dans cet exemple de code, le nom du serveur ne pourra pas être modifié au moment de l'installation sans recompilation. Ajoutons un simple tag devant l'attribut :

```
@Resource(name = "serverName")
String serverName = "testserver";
```

Cela ne changera rien à la valeur de l'attribut, qui par défaut vaut toujours `testserver`. Ajoutons maintenant dans le répertoire META-INF de l'application un fichier `ejb-jar.xml` :

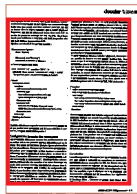
```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd"
  version="3.0">
  <enterprise-beans>
    <session>
      <ejb-name>ServiceBean</ejb-name>
      <env-entry>
        <env-entry-name>serverName</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <env-entry-value>productionserver</env-entry-value>
      </env-entry>
    </session>
  </enterprise-beans>
</ejb-jar>
```

La seule présence de ce fichier, lu et analysé par le serveur JEE, suffit pour demander à ce dernier d'injecter la valeur `productionserver` dans l'attribut `serverName` de l'EJB.

Autres paramètres

L'annotation `@Resource` est valable pour à peu près n'importe quelle ressource, à travers l'annuaire JNDI. Cependant, en dehors du cas des ressources de types simples, la correspondance n'est pas établie dans le fichier `ejb-jar.xml`, mais dans le descripteur de déploiement spécifique au serveur d'application.

En effet, le fichier `ejb-jar.xml` ne contient que des informations portables d'un serveur JEE à un autre, et si la ressource que l'on veut se faire injecter n'est pas définie dans le même fichier `ejb-jar.xml`, il est probable que le moyen de la référencer varie selon l'installation du container. Nous parlons ici de portabilité d'un serveur JEE à un autre, au sens implémentation JEE. Par exemple, Jonas et JBoss sont deux implémentations JEE, et une application JEE doit pouvoir



être déployée sur ces serveurs. Dans le cas de JBoss, les informations de mapping pour se faire injecter une ressource de type autre qu'un type simple sont dans un fichier *jboss.xml*, qui doit se trouver dans le même répertoire que le fichier *ejb-jar.xml*. Supposons que l'on ait maintenant besoin dans notre EJB de service de poster un message sur une file JMS. Nous avons besoin pour ce faire de la référence vers la file en question ainsi que d'une fabrique de connexions JMS.

Ajoutons sur notre EJB les attributs suivants :

```
@Resource(name="queue")
Queue laQueue;

@Resource(name="factory")
QueueConnectionFactory factory;
```

Le fichier *jboss.xml* sera alors :

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 4.2//EN"
"http://www.jboss.org/j2ee/dtd/jboss_4_2.dtd">

<jboss>
  <enterprise-beans>
    <session>
      <ejb-name>ServiceBean</ejb-name>
      <resource-ref>
        <res-ref-name>queue</res-ref-name>
        <jndi-name>queue/A</jndi-name>
      </resource-ref>
      <resource-ref>
        <res-ref-name>factory</res-ref-name>
        <jndi-name>QueueConnectionFactory</jndi-name>
      </resource-ref>
    </session>
  </enterprise-beans>
</jboss>
```

Ce fichier définit la correspondance entre le nom de la ressource tel que précisé dans l'annotation java et le nom JNDI de la ressource, les types doivent correspondre !

Configuration avancée des ressources

Les annotations *@Resource* peuvent être positionnées soit sur un attribut, soit sur un *getter* de l'attribut, soit directement sur la classe. Une même annotation ne peut pas figurer plusieurs fois sur un même élément. C'est pourquoi l'annotation *@Resources* a été prévue pour rassembler en fait plusieurs annotations *@Resource* que l'on souhaiterait toutes placer au niveau de la classe. L'annotation *@Resource* a un attribut *mappedName*, qu'il est recommandé de ne pas utiliser : son interprétation est susceptible d'être variable d'un container à un autre. Son usage fait perdre de la portabilité au code. Cet attribut se retrouve dans la définition de la ressource dans le fichier de configuration *ejb-jar.xml*, et pour la même raison de portabilité, il est préférable de déclarer la correspondance des ressources en utilisant le fichier de configuration spécifique au container JEE. Remarque : la majorité des serveurs JEE interprète le *mapped-name* comme étant le nom JNDI de la ressource.

Déployer plusieurs fois un même EJB Session

Pourquoi aurait-on besoin de déployer plusieurs fois un même EJB ? Dès lors que les paramètres peuvent être modifiés depuis le fichier de configuration, une application pourrait avoir besoin d'utiliser une version de l'EJB (session) avec certaines valeurs pour des paramètres et une autre version avec d'autres valeurs. Reprenons l'exemple d'un EJB session qui gère une communication avec une autre application (via JMS ou autre). Si en fait, en fonction de certaines règles de gestion, l'application destinataire est susceptible de varier, alors nous aurons besoin de disposer de plusieurs variantes de l'EJB session, une avec un paramétrage adapté pour chaque destination.

Le paramétrage se fait dans le fichier *ejb-jar.xml*, dans lequel il est possible de déclarer un EJB session plusieurs fois. Nous avons vu que la correspondance entre la classe de l'EJB et le paramétrage dans le fichier *ejb-jar.xml* est établie par le container à partir du nom de l'EJB (le nom de la classe par défaut). Pour déployer une seconde fois le même EJB dans la même application, le second déploiement doit avoir un nom différent. Ce nom ne peut donc plus être utilisé par le container pour faire la correspondance avec la classe, il faut donc ajouter dans le paramétrage le nom de la classe de l'EJB. Voici un exemple :

```
<session>
  <ejb-name>ServiceBean</ejb-name>
  <ejb-class>test.ServiceBean</ejb-class>
  <env-entry>
    <env-entry-name>testParam</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>ServiceBean2 param</env-entry-value>
  </env-entry>
</session>
```

Avec ces lignes ajoutées dans le fichier *ejb-jar.xml*, la même classe d'EJB sera déployée une seconde fois, sous le nom *ServiceBean2*. Et l'EJB *ServiceBean* sera dans tous les cas déployé, même s'il n'est pas explicitement mentionné dans le fichier *ejb-jar.xml*. Il est donc possible de déployer la même classe d'EJB session plusieurs fois, ce qui permet d'avoir différentes configurations de paramétrage pour cet EJB. **Attention** : lorsqu'un EJB session est déployé plusieurs fois, il faut préciser lors de l'injection quel déploiement doit être injecté ! Dans le cas contraire, une erreur analogue à la suivante sera indiquée au moment du déploiement. Ainsi, pour se faire injecter une référence à un session bean dont la classe est déployée plusieurs fois sous des noms différents, il convient d'ajouter l'attribut *beanName* au moment de l'injection.

Conclusion

Le confort apporté au développeur par les annotations n'annihile pas pour autant la possibilité de paramétrer au moment du déploiement, et sans recompiler, les EJB. Les fichiers XML sont maintenant réduits au minimum en ne comportant que les écarts entre la configuration par défaut et la configuration installée. Une condition : que le déployeur ait à sa disposition une documentation suffisante sur les paramètres, leur signification et la manière de les configurer.

■ Dominique Méra - Directeur de projets Objet Direct, filiale d'Homsys Group.