Collège technique des Aumôniers du Travail de Charleroi

Projet internet / intranet

Les annotations

2011-2012

Renaud DIANA

Table des matières

1 Présentation

Java SE 5 a introduit les annotations qui sont des métas données incluses dans le code source. Les annotations ont été spécifiées dans la JSR 175 : leur but est d'intégrer au langage Java des métas données.

Les annotations de Java 5 apportent une standardisation des métas données dans un but généraliste. Ces métas données associés aux entités Java peuvent être exploitées à la compilation ou à l'exécution.

Java a été modifié pour permettre la mise en oeuvre des annotations :

- une syntaxe dédiée a été ajoutée dans Java pour permettre la définition et l'utilisation d'annotations.
- le byte code est enrichi pour permettre le stockage des annotations.

Les annotations peuvent être utilisées avec quasiment tous les types d'entités et de membres de Java : packages, classes, interfaces, constructeurs, méthodes, champs, paramètres, variables ou annotations elles même.

Java 5 propose plusieurs annotations standards et permet la création de ces propres annotations.

Une annotation s'utilise en la faisant précéder de l'entité qu'elle concerne. Elle est désignée par un nom précédé du caractère @.

Il existe plusieurs catégories d'annotations :

- les marqueurs (markers) : (exemple : @Deprecated, @Override, ...)
- les annotations paramétrées (exemple : @MonAnnotation("test"))
- les annotations multi paramétrées : exemple : @MonAnnotation(arg1="test 1", arg2="test 2", arg3="test3")

Les arguments fournis en paramètres d'une annotation peuvent être de plusieurs types : les chaînes de caractères, les types primitifs, les énumérations, les annotations, le type Class.

2 La mise en oeuvre des annotations

Les annotations fournissent des informations sur des entités : elles n'ont pas d'effets directs sur les entités qu'elles concernent.

Les annotations utilisent leur propre syntaxe. Une annotation s'utilise avec le caractère @ suivi du nom de l'annotation : elle doit obligatoirement précéder l'entité qu'elle annote. Par convention, les annotations s'utilisent sur une ligne dédiée.

Les annotations peuvent s'utiliser sur les packages, les classes, les interfaces, les méthodes, les constructeurs et les paramètres de méthodes.

Exemple :

```
@Override
public void maMethode() {
}
```

Une annotation peut avoir un ou plusieurs attributs : ceux ci sont précisés entre parenthèses, séparés par une virgule. Un attribut est de la forme clé=valeur.

Exemple :

```
@SuppressWarnings(value = "unchecked")
void maMethode() { }
```

Lorsque l'annotation ne possède qu'un seul attribut, il est possible d'omettre son nom.

Exemple :

```
@SuppressWarnings("unchecked")
void maMethode() { }
```

Un attribut peut être de type tableau : dans ce cas, les différentes valeurs sont fournies entre accolade, chacune séparée par une virgule.

Exemple :

```
@SuppressWarnings(value={"unchecked", "deprecation"})
```

Il existe trois types d'annotations :

- Les marqueurs (markers) : ces annotations ne possèdent pas d'attribut
- Les annotations simples (single value annotations) : ces annotations ne possèdent qu'un seul attribut
- Les annotations complètes (full annotations) : ces annotations possèdent plusieurs attributs

3 L'utilisation des annotations

Les annotations prennent une place de plus en plus importantes dans la plate-forme Java et de nombreuses API open source.

Les utilisations des annotations concernent plusieurs fonctionnalités :

- Utilisation par le compilateur pour détecter des erreurs ou ignorer des avertissements
- Documentation
- Génération de code
- Génération de fichiers

3.1 La documentation

Les annotations peuvent être mise en oeuvre pour permettre la génération de documentations indépendantes de JavaDoc : liste de choses à faire, liste de services ou de composants, ...

Il peut par exemple être pratique de rassembler certaines informations mises sous la forme de commentaires dans des annotations pour permettre leur traitement.

Par exemple, il est possible de définir une annotation qui va contenir les métas données relatives aux informations sur une classe. Traditionnellement, une classe débute par un commentaire d'en-tête qui contient des informations sur l'auteur, la date de création, les modifications, ... L'idée est de fournir ces informations dans une annotation dédiée. L'avantage est de permettre facilement d'extraire et de manipuler ces informations qui ne seraient qu'informatives sous leur forme commentaires.

3.2 L'utilisation par le compilateur

Les trois annotations fournies en standard avec la plate-forme entre dans cette catégorie qui consiste à faire réaliser par le compilateur quelques contrôles basiques.

3.3 La génération de code

Les annotations sont particulièrement adaptées à la génération de code source afin de faciliter le travail des développeurs notamment sur des tâches répétitives.

Attention, le traitement des annotations ne peut pas modifier le code existant mais simplement créer de nouveaux fichiers sources.

3.4 La génération de fichiers

Les API standards ou les frameworks open source nécessitent fréquemment l'utilisation de fichiers de configuration ou de déploiement généralement au format XML.

Les annotations peuvent proposer une solution pour maintenir le contenu de ces fichiers par rapport aux entités incluses dans le code de l'application.

La version 5 de Java EE fait un important usage des annotations dans le but de simplifier les développements de certains composants notamment les EJB, les entités et les services web. Pour cela, l'utilisation de descripteurs est remplacée par l'utilisation d'annotations ce qui rend le code plus facile à développer et plus clair.

3.5 Les API qui utilisent les annotations

De nombreuses API standards utilisent les annotations depuis leur intégration dans Java notamment :

- JAXB 2.0 : JSR 222 (Java Architecture for XML Binding 2.0)
- Les services web de Java 6 (JAX-WS) : JSR 181 (Web Services Metadata for the Java Platform) et JSR 224 (Java APIs for XML Web Services 2.0 API)
- Les EJB 3.0 et JPA : JSR 220 (Enterprise JavaBeans 3.0)

De nombreuses API open source utilisent aussi les annotations notamment JUnit, TestNG, Hibernate, ...

4 Les annotations standards

Java 5 propose plusieurs annotations standards.

4.1 L'annotation @Deprecated

Cette annotation a un rôle similaire au tag de même nom de Javadoc. C'est un marqueur qui précise que l'entité concernée est obsolète et qu'il ne faudrait plus l'utiliser. Elle peut être utilisée avec une classe, une interface ou un membre (méthode ou champ)

Exemple :

```
public class TestDeprecated {

  public static void main(
      String[] args) {
    MaSousClasse td = new MaSousClasse();
    td.maMethode();
  }
}

@Deprecated
class MaSousClasse {

  /**
   * Afficher un message de test
   * @deprecated methode non compatible
   */
  @Deprecated
  public void maMethode() {
    System.out.println("test");
  }
}
```

Les entités marquées avec l'annotation @Deprecated devrait être documentée avec le tag @deprecated de Javadoc en lui fournissant la raison de l'obsolescence et éventuellement l'entité de substitution.

Il est important de tenir compte de la casse : @Deprecated pour l'annotation et @deprecated pour Javadoc.

4.2 L'annotation @Override

Cette annotation est un marqueur utilisé par le compilateur pour vérifier la réécriture de méthode héritée.

@Override s'utilise pour annoter une méthode qui est une réécriture d'une méthode héritée. Le compilateur lève une erreur si aucune méthode héritée ne correspond.

Exemple :

```
@Override
public void maMethode() {
}
```

Son utilisation n'est pas obligatoire mais recommandée car elle permet de détecter certains problèmes.

Exemple :

```
public class MaClasseMere {

}

class MaClasse extends MaClasseMere {

  @Override
  public void maMethode() {

  }
}
```

Ceci est particulièrement utile pour éviter des erreurs de saisie dans le nom des méthodes à redéfinir.

4.3 L'annotation @SuppressWarning

L'annotation @SuppressWarning permet de demander au compilateur d'inhiber certains avertissements qui sont pris en compte par défaut.

La liste des avertissements utilisables dépend du compilateur. Un avertissement utilisé dans l'annotation non reconnu par le compilateur ne provoque par d'erreur mais éventuellement un avertissement.

Le compilateur fourni avec le JDK supporte les avertissements suivants :

Nom	Rôle
deprecation	Vérification de l'utilisation d'entités déclarées deprecated
unchecked	Vérification de l'utilisation des generics
fallthrough	Vérification de l'utilisation de l'instruction break dans les cases des instructions switch
path	Vérification des chemins fournis en paramètre du compilateur
serial	Vérification de la définition de la variable serialVersionUID dans les beans
finally	Vérification de la non présence d'une instruction return dans une clause finally

Il est possible de passer en paramètres plusieurs types d'avertissements sous la forme d'un tableau

Exemple :

```
@SuppressWarnings(value={"unchecked", "fallthrough"})
```

L'exemple ci-dessous génère un avertissement à la compilation

Exemple :

```
import java.util.ArrayList;
import java.util.List;

public class TestSuppresWarning {
  public static void main(
      String[] args) {
    List donnees = new ArrayList();
    donnees.add("valeur1");
  }
}
```

Pour supprimer cet avertissement, il faut utiliser les génerics dans la déclaration de la collection ou utiliser l'annotation SuppressWarning.

Exemple :

```
import java.util.ArrayList;
import java.util.List;

@SuppressWarnings("unchecked")
public class TestSuppresWarning {
  public static void main(
      String[] args) {
    List donnees = new ArrayList();
    donnees.add("valeur1");
  }
}
```

Il n'est pas recommandé d'utiliser cette annotation mais plutôt d'apporter une solution à l'avertissement.

5 Les annotations pour les servlets

Les annotations suivantes sont apllicables depuis la spécification servlet 3.0 :

- @WebServlet
- @WebServletContextListener
- @ServletFilter
- @InitParam

5.1 Web Servlet and InitParam Annotation

SimpleServlet.java

```
package net.javabeat.servlet30.newfeatures;

import javax.servlet.annotation.InitParam;
import javax.servlet.annotation.WebServlet;

@WebServlet(
    name = "SimpleServlet",
    urlPatterns = {"/simple"},
    initParams = {
        @InitParam(name = "param1", value = "value1"),
        @InitParam(name = "param2", value = "value2")}
)
public class SimpleServlet {

}
```

Nous avons déclaré une classe par le nom SimpleServlet et cette classe n'est pas faite pour étendre ou mettre en œuvre l'un des types de Servlet / HttpServlet. Au lieu de cela, pour qualifier cette classe comme une classe Servlet, nous l'avons annotée en utilisant l'annotation @ WebServlet. Notez que le nom de la servlet est SimpleServlet comme spécifié par l'attribut *name*. L'attribut *urlpatterns* définit un ensemble d'URL-modèles qui peuvent être utilisées pour invoquer la servlet. Le conteneur de servlet après la numérisation de cette classe va générer le descripteur de déploiement qui peut ressembler à ceci :

web.xml

```
...

<servlet>
    <servlet-name>SimpleServlet</servlet-name>
    <servlet-class>net.javabeat.servlet30.newfeatures.SimpleServlet</servlet-
class>

        <init-param>
            <param-name>param1</param-name>
            <param-value>value1</param-value>
        </init-param>

        <init-param>
```

```
            <param-name>param2</param-name>
            <param-value>value2</param-value>
    </init-param>

</servlet>

<servlet-mapping>
    <url-pattern>/simple</url-pattern>
    <servlet-name>SimpleServlet</servlet-name>
</servlet-mapping>

...
```

5.2 Filter annotation

L'annotation @ Filter définit un composant filtre de servlet d'une application web. Un filtre est généralement utilisé pour intercepter une requête Web pour effectuer une des opérations de pré-traitement bien avant d'atteindre la composante servlet.

SimpleFilter.java

```
package net.javabeat.servlet30.newfeatures;

import javax.servlet.annotation.InitParam;
import javax.servlet.annotation.ServletFilter;

@ServletFilter(
    filterName = "SimpleFilter",
    urlPatterns = "/simple",
    initParams =
        {@InitParam(name = "param1", value = "value1"),
         @InitParam(name = "param2", value = "value2")}
)
public class SimpleFilter {

}
```

Encore une fois, afin de soutenir la compatibilité descendante, les informations d'annotation dans la catégorie ci-dessus seront transformées en information dans le descripteur de déploiement par le conteneur de servlets et le descripteur de déploiement pourrait ressembler à ceci :

web.xml

```
...

<filter>

    <filter-name>SimpleFilter</filter-name>
    <filter-class>net.javabeat.servlet30.newfeatures.SimpleFilter</filter-
```

```
class>

    <init-param>
        <param-name>param1</param-name>
        <param-value>value1</param-value>
    </init-param>

    <init-param>
        <param-name>param2</param-name>
        <param-value>value2</param-value>
    </init-param>

</filter>

<filter-mapping>
    <filter-name>SimpleFilter</filter-name>
    <url-pattern>/simple</url-pattern>
<filter-mapping>

...
```

5.3 Servlet Context Listener annotation

Le servlet listener contexte est utilisée pour recevoir des événements lorsque le contexte de servlet est créée et détruite par le conteneur Web.

SimpleServletContextListener.java

```
package net.javabeat.servlet30.newfeatures;

import javax.servlet.ServletContextEvent;
import javax.servlet.annotation.WebServletContextListener;

@WebServletContextListener()
public class SimpleServletContextListener {

    public void contextInitialized(ServletContextEvent event){
    }

    public void contextDestroyed(ServletContextEvent event){
    }
}
```

web.xml

```
<web-app>

    <listener>
        <listener-
```

```
class>net.javabeat.servlet30.newfeatures.SimpleServletContextListener
    </listener-class>
    </listener>

</web-app>
```