Méta-programmation en Java : *Introspection et annotations*

Rezak AZIZ
rezak.aziz@lecnam.net

CNAM Paris

Objectifs du cours

- Comprendre le concept de **métaprogrammation** : écrire du code qui manipule du code ;
- Découvrir deux grands mécanismes en Java :
  - **Introspection** ;
  - **Annotations** et leur traitement ;
- Identifier les usages dans les frameworks modernes : *Spring*, *Hibernate*, *JUnit*, *Lombok*.

Qu'est-ce que la métaprogrammation ?

Définition

Capacité d'un programme à **inspecter**, **modifier** ou **générer du code**.

- En Java, la métaprogrammation se fait par :
  1. **Introspection** (inspection du code à l'exécution) ;
  2. **Annotations** (ajout de métadonnées) ;
  3. **Proxys dynamiques** (génération de classes au runtime).
- Utilisée massivement dans :
  - *Spring* (injection de dépendances) ;
  - *Hibernate* (mapping objet-relationnel) ;
  - *JUnit* (repérage automatique de tests).

Introspection

Principe de l'introspection

Définition simple

L'**introspection** permet à un programme Java d'**explorer sa propre structure** :

- connaître le nom d'une classe ;
- voir les méthodes et attributs qu'elle contient ;
- et même les exécuter ou les modifier.

Cela se fait grâce à la classe `java.lang.Class`.

Exemple : observer une classe

```
// On récupère la description de la classe String
Class<String> clazz = String.class;

// On affiche le nom de la classe
System.out.println("Nom de la classe : " + clazz.getName());

// On liste toutes ses méthodes publiques
for (Method m : clazz.getMethods()) {
    System.out.println("Méthode : " + m.getName());
}
```

Explication

Ce programme « observe » la classe `String` :

- il découvre son nom (`String`);
- il affiche la liste de toutes ses méthodes publiques.

Les principales méthodes de la classe `Class`

`getMethod(String name, Class<?>... parameterTypes)`

Récupère une méthode publique d'après son nom et les types de ses paramètres. Exemple :
`clazz.getMethod("substring", int.class, int.class);`

`getMethods()`

Récupère **toutes les méthodes publiques** de la classe (y compris celles héritées).

`getDeclaredAnnotations()`

Récupère toutes les **annotations** déclarées sur la classe.

Remarque : les objets de type `Method` possèdent aussi des méthodes utiles comme `getName()`, `getReturnType()` et `invoke(...)`.

Appel dynamique d'une méthode

```
// Une classe simple
class Person {
  private String name = "Alice";
  public void sayHello() {
    System.out.println("Bonjour " + name);
  }
}

// On charge la classe dynamiquement
Class<?> clazz = Class.forName("Person");

// On crée un objet Person
Object p = clazz.getDeclaredConstructor().newInstance();

// On récupère la méthode publique sayHello()
Method m = clazz.getMethod("sayHello");

// Et on 'lexécute !
m.invoke(p); // Bonjour Alice
```

Modifier un champ privé

```
Field f = clazz.getDeclaredField("name");

// Autorise l'accès au champ privé
f.setAccessible(true);

// Modifie la valeur du champ
f.set(p, "Bob");

// Réexécution
m.invoke(p); // Bonjour Bob
```

À retenir

- `getMethod(...)` : récupère une méthode publique par son nom ;
- `invoke(obj)` : exécute la méthode sur un objet donné ;
- `getDeclaredField(...)` : accède à un champ même s'il est privé.

Annotations

Les annotations en Java

Rôle

Les annotations ajoutent des **métadonnées** au code : elles informent le compilateur, l'IDE, ou le programme lui-même.

- Exemples : `@Override`, `@Deprecated`, `@Test`, `@WebServlet`.
- Elles peuvent être utilisées :
  - à la compilation (vérifications, génération de code). Eg. `@Override` ;
  - ou à l'exécution (via introspection). Eg. `@WebServlet`

Arguments des annotations

Principe général

Les annotations peuvent recevoir des **arguments**, qui sont des objets Java

Cas particuliers

- Si l'annotation définit un seul argument nommé `value`, on peut **omettre le nom du paramètre**.
- On peut passer un **tableau de valeurs** en l'écrivant entre accolades : `{...}`.

Exemples d'annotations avec arguments

```
// Exemple 1 : plusieurs arguments nommés
@Test(expected = VerificationException.class, timeout = 1000)

// Exemple 2 : un seul argument "value"
@SuppressWarnings("unchecked")

// Exemple 3 : un tableau de valeurs
@SuppressWarnings(value = {"unchecked", "deprecation"})
```

Définir une annotation personnalisée

Déclaration de l'annotation

```
// L'annotation sera visible à l'exécution
@Retention(RetentionPolicy.RUNTIME)
// Elle ne peut être utilisée que sur les méthodes
@Target(ElementType.METHOD)
public @interface Loggable {
  // Un argument optionnel "value"
  String value() default "Appel de méthode";
}
```

Exemple d'utilisation

```
public class Service {
  @Loggable("Traitement principal")
  public void process() {
    System.out.println("Processing...");
  }
}
```

- L'annotation `@Loggable` marque les méthodes à suivre.
- On pourra ensuite les détecter par introspection pour, par exemple, enregistrer un message dans un journal d'exécution.

Rétention des annotations

Trois possibilités :

SOURCE : plus de trace de l'annotation dans le byte code Java. Ex. possible : `@Suppresswarnings` ;

CLASS : l'annotation est conservée dans le byte code mais n'est pas consultable à l'exécution ;

RUNTIME : l'annotation est conservée à l'exécution, elle est visible par le code (ex. `@Test`, `@WebServlet`…

Lire une annotation à l'exécution

```
// Création d'un objet Service
Service s = new Service();

// Parcours des méthodes de la classe
for (Method m : s.getClass().getDeclaredMethods()) {

  // Vérifie si la méthode porte @Loggable
  if (m.isAnnotationPresent(Loggable.class)) {

    // Récupère l'annotation
    Loggable ann = m.getAnnotation(Loggable.class);

    // Affiche le message défini dans l'annotation
    System.out.println("[LOG] " + ann.value());

    // Exécute la méthode
    m.invoke(s);
  }
}
```

Manipulation des annotations

Observation

Dans l'exemple précédent :

- On a manipuler les annotations par introspection ;
- les `Class`, les `Methods` ont les méthodes :
  - `getAnnotation(A)` qui prend comme argument la classe d'annotation, renvoie un objet d'annotation ou null si l'annotation est absente ;
  - `getAnnotations()` : renvoie un tableau avec toutes les annotations de la classe ou de la méthode ;
- les champs de l'annotation sont accessibles ;

Annotations et héritage

- les annotations de méthodes ne sont jamais héritées ;
- les annotations de classe peuvent l'être (si l'annotation est annotée comme `@Inherited`)

Exemple Concret: Hibernate

Hibernate : qu'est-ce que c'est ?

Définition

Hibernate est un framework Java qui facilite l'accès aux bases de données. Il fait partie de la spécification **JPA (Java Persistence API)**.

Hibernate : qu'est-ce que c'est ?

Définition

Hibernate est un framework Java qui facilite l'accès aux bases de données. Il fait partie de la spécification **JPA (Java Persistence API)**.

- Il permet de sauvegarder des objets Java dans une base relationnelle sans écrire de SQL manuellement.
- Il fait le **mapping automatique** entre :
  - les classes Java et les tables SQL ;
  - les attributs et les colonnes.

Hibernate et les annotations

- Hibernate utilise la **méta-programmation** pour lire les annotations et générer le SQL correspondant.
- Les annotations définissent comment la classe Java est stockée dans la base.

```
@Entity
@Table(name = "clients")
public class Client {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  @Column(name = "nom")
  private String nom;
}
```

Hibernate et les annotations

- Hibernate utilise la **méta-programmation** pour lire les annotations et générer le SQL correspondant.
- Les annotations définissent comment la classe Java est stockée dans la base.

```
@Entity
@Table(name = "clients")
public class Client {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  @Column(name = "nom")
  private String nom;
}
```

→ Hibernate lit ces annotations par **introspection** et construit les requêtes SQL pour insérer, mettre à jour ou lire les objets.

Comment Hibernate utilise la méta-programmation

- Analyse les classes annotées (‘@Entity‘, ‘@Column‘, ‘@Id‘).
- Crée dynamiquement le SQL adapté.
- Appelle les constructeurs et setters pour reconstruire les objets à partir du résultat SQL.

Exemple de mapping automatique

Classe Java	Table SQL
Client.nom	nom VARCHAR(255)
Client.id	id INT AUTO_INCREMENT