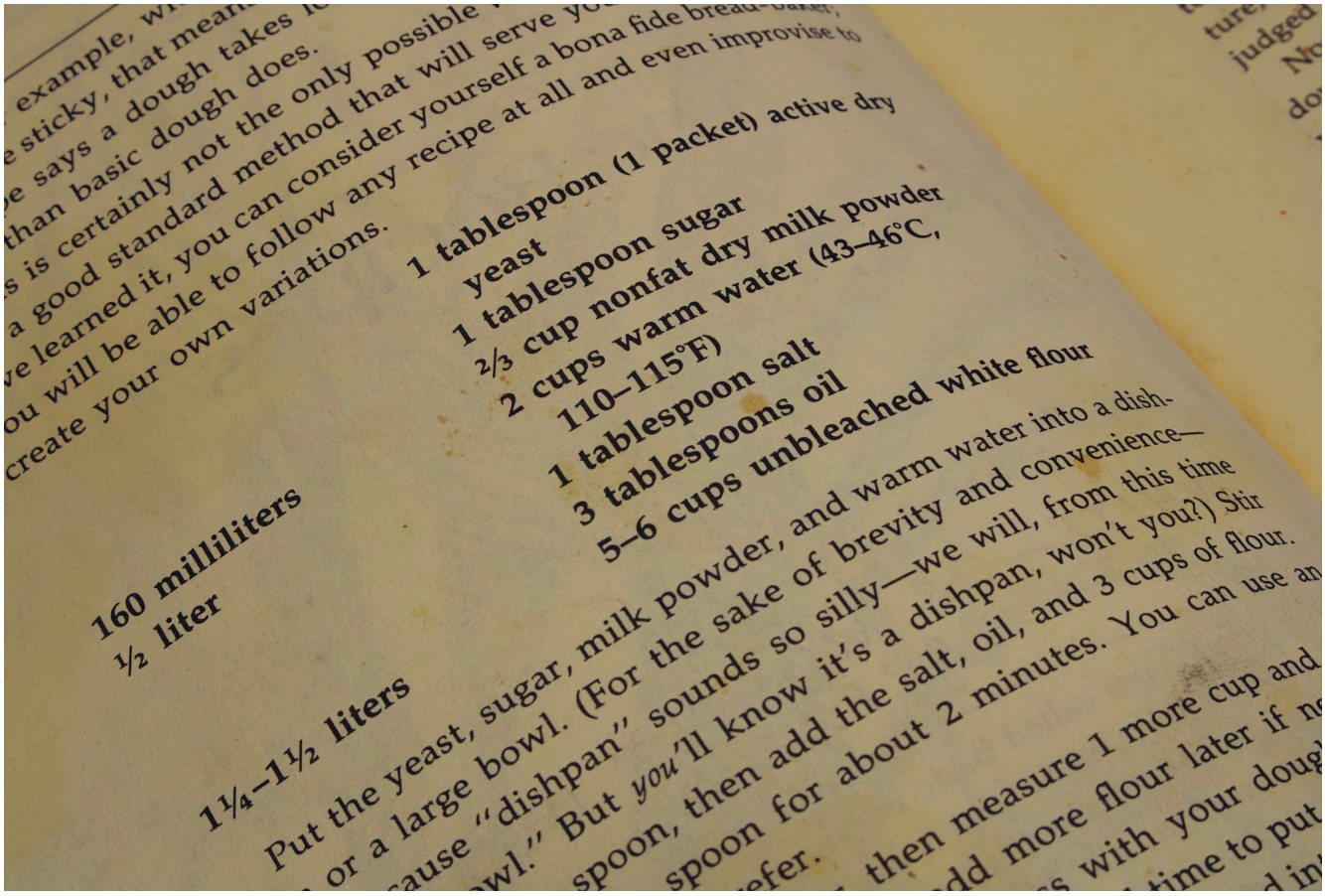

OpenRewrite: Refactoring as code

2024, NOV 27

Logo d'openrewrite

[fa lightbulb] *Mieux vaut prévenir que guérir*, je n'utiliserai pas le mot *reusiner*, et certainement pas non plus le mot *refactoriser*. Cette note sera donc pleine d'anglicismes. La vie.

Alors comme ça vous voulez *refactor* code ? Voici les différentes options qui s'offrent habituellement à vous (je sais, il y en a d'autres) :

- Pour commencer le fameux (ou l'infâme) **Chercher/Remplacer**, plus connu sous le nom de `Ctrl+F/Ctrl+R`.
- La technique un peu plus avancée de l'expression régulière, aussi connue sour le nom de *Les regexps c'est illisible 15 jours après les avoir écrites*.
- Mais de façon plus probable, vous utiliserez le menu *click droit* de votre IDE, ou un bon paquet de raccourcis clavier, technique également connue sous le nom de *tu peux toujours courir, jamais tu reproduiras ce que j'ai fait*.

Si vous êtes dans le cas d'un refactoring qui nécessite plus d'une étape et/ou qui touche à de nombreux fichiers, vous êtes très probablement condamné à suivre un guide de migration, aussi triste qu'un long dimanche de pluie.

Disons que vous vouliez par exemple migrer de `JUnit 4` à `JUnit 5`, ou bien de `Spring Boot 2` à `Spring Boot 4`, ou d'`Hibernate 4` à `Hibernate 6`, ou encore de `Java 8` à `Java 21`. Il vous faudrait pour cela :

1. Faire les monter de versions dans vos fichiers de configuration `Maven` ou `Gradle`
2. Changer vos imports
3. Changer vos annotations
4. Changer vos invocations de méthodes
5. Peut-être changer vous signature de méthodes
6. Changer les noms de propriété
7. Je ne sais quoi encore

C'est chronophage, lourd, sujet aux erreurs et aux oublis.

Mais il y a un petit nouveau dans le monde du refactoring : **OpenRewrite**.

The big picture [fa camera]

OpenRewrite est un outil de refactoring qui a été créé par Moderne.

C'est un projet open source sous licence Apache 2.0, et il fait maintenant partie de la Fondation Commonhaus pour une gouvernance plus ouverte.

Il a commencé avec un fort accent sur Java (et ses fichiers de configuration : properties et yaml), mais il s'étend maintenant à d'autres langages et formats de fichiers :

- Langages de programmation : Java, Kotlin, Groovy
- Formats de données : XML, Properties, YAML, JSON, Protobuf
- Outils de build : Maven, Gradle

Je me concentrerai uniquement sur l'écosystème Java dans cet article.

Les concepts

Les principaux concepts d'OpenRewrite que vous devez garder à l'esprit sont les suivants.

Lossless Syntax Tree

Le code source que vous souhaitez consulter et transformer est rendu accessible via un ***LST***. C'est

un arbre de syntax abstrait (**AST**) contenant en plus des information de formatage. Une représentation asbtraite du code source pour le rendre plus facile à comprendre, à interroger ou à manipuler.

Recette [fa utensils]

Ou *recipe* dans la langue de Shakespear est l'élément atomique pour travailler sur le code. Elle peut contenir chacun des éléments suivants (tous étant optionnels):

- Un/des paramètre(s) pour personnaliser le comportement de la recette.
- Une liste de pré-conditions, lui permettant de savoir si elle doit traiter un fichier source.
- Un *visitor* qui permet d'effectuer ou non des modifications sur le code source.
- Une liste de recettes

Ce dernier point signifie que la façon d'implémenter un refactoring complexe est de composer des *recettes*, et à la fin, cela s'appelle aussi une recette.

¯_(ツ)_/¯ Ne blâmez pas le messager.

Le pattern visitor [fa spaghetti monster flying]

Je l'ai mentionné juste au dessus, mais letraitement d'un recette passe par le pattern visitor. Chaque rencontre d'un élément du *lst* va généré un *événement* qui sera transmis à tous les visiteurs déclarés. Pour vous donner une idée, il y a actuellement 73 type d'événements différent associé à la *visite* pour de code java, chacun associé à des méthodes avec des signatures différentes.

Qu'y a-t-il dans la boîte [fa gift] ?

OpenRewrite est plus qu'un framework monolithique. Ces différents composants sont :

- Un module *core*, qui contient toute la représentation générique d'un LST et la logique de refactoring commune.
- Un module pour chaque langage, avec des API et SDK dédiés pour une cible spécifique (Java, XML, Yaml, etc).
- De nombreux modules contenant des recettes pour un sous-ensemble d'intérêt spécifique, telles que les recettes de frameworks de test, les recettes de Spring, les recettes de Quarkus, l'identification et la correction des problèmes d'analyse statique, etc.

Le catalogue [fa book open]

La grande puissance d'Openrewrite est la mise à disposition recette déjà disponible pour traiter un très grand nombre de cas sans que l'on ait à produire la moindre ligne de code. Il y a actuellement plus de 500 recettes disponibles pour Java, et il y en a probablement plus pour les autres langages et formats de fichiers.

Le catalogue est disponible en ligne à l'adresse suivante : https://docs.openrewrite.org/recipes

Il contient toutes les recettes produites par l'équipe de Moderne, triées par catégories, documentées et avec des exemples d'utilisation. Mais il contient également d'autres sections intéressantes. On y trouve par exemple une liste des recettes les plus populaires, comme un accélérateur de recherche, mais aussi une page dédiée aux recettes écrites par d'autre, comme *Apache Camel*, *AWS*, *Quarkus*, *Morphia* et bien d'autres.

Execution d'une recette [fa cogs]

Attaquons-nous à la partie la plus simple de cette article : l'exécution d'une recette. Petit prérequis: vous devez avoir `Maven` ou `Gradle` installé sur votre machine. Que vous voulier exécuter une recette qui concerne *Java* ou *Kubernetes*, la procédure est la même. Désolé.

Pour exécuter une recette, vous avez deux options.

En modifiant vos descripteurs de build

Je vais prendre l'exemple d'un projet *Maven*, mais les étapes à suivre sont les mêmes pour un projet *Gradle*.

Pour commencer, vous devez ajouter le plugin `rewrite-maven-plugin` à votre fichier `pom.xml` :

```
<build>
  <plugins>
    <plugin>(1)
      <groupId>org.openrewrite.maven</groupId>
      <artifactId>rewrite-maven-plugin</artifactId>
      <version>5.46.1</version>(2)
    </plugin>
  </plugins>
</build>
```

1. Déclaration du plugin
2. Adapter le numéro pour utiliser la version la plus à jour

Ensuite, vous devez déclarer la recette que vous voulez exécuter. Ici par exemple la suppression de *Cobertura* qui n'est plus compatible avec un projet *Java* dont la version est supérieure à *Java 11* :

```
<build>
  <plugins>
    <plugin>
      <groupId>org.openrewrite.maven</groupId>
      <artifactId>rewrite-maven-plugin</artifactId>
      <version>5.46.1</version>
      <configuration> (1)
          <activeRecipes>
            <recipe>org.openrewrite.java.migrate.cobertura.RemoveCobert
          </activeRecipes>
      </configuration>
    </plugin>
  </plugins>
</build>
```

1. Configuration du plugin
2. Activation de la recette

Ajout de la dépendance dans laquelle se trouve la recette (si elle n'est pas dans le module core), ce qui donne la configuration complète suivante :

```
<build>
  <plugins>
    <plugin>
      <groupId>org.openrewrite.maven</groupId>
      <artifactId>rewrite-maven-plugin</artifactId>
      <version>5.46.1</version>
      <configuration>
        <activeRecipes>
          <recipe>org.openrewrite.java.migrate.cobertura.RemoveCobertur
        </activeRecipes>
      </configuration>
      <dependencies>
        <dependency>
          <groupId>org.openrewrite.recipe</groupId>
          <artifactId>rewrite-migrate-java</artifactId>
          <version>2.30.1</version>
        </dependency>
      </dependencies>
    </plugin>
```

```
    </plugins>
  </build>
```

Pour exécuter la recette, il suffit de lancer la commande suivante :

```
$ mvn rewrite:run
```

Mais on ne veut pas modifier nos fichiers de build, n'est-ce pas ? Et on ne se trouve peut-être même pas dans un projet *Maven* ou *Gradle*.

Sans modifier vos descripteurs de build

Dans ce cas il est possible de préciser directement tout dans la ligne de commande, mais celle-ci deviendra forcément plus complexe :

```
$ mvn -U org.openrewrite.maven:rewrite-maven-plugin:run <1>
   -Drewrite.recipeArtifactCoordinates=org.openrewrite.recipe:rewrite-j
   -Drewrite.activeRecipes=org.openrewrite.java.migrate.cobertura.Remov
```

1. Déclaration du plugin
2. Ajout de la dépendance de la recette
3. Activation de la recette

Concevoir ses propres recettes

Les façons de faire décrites ci-dessus ne sont valables que si les recettes ne prennent pas de paramètres. Si telle n'est pas le cas il va falloir passer à l'étape suivante : la conception de recettes.

Pour concevoir ses propres recettes, le guide de bonne pratique d'Openrewrite nous dit que tout ce qui peut être faît de manière déclarative doit l'être. Oui, je sais, c'est dur. Vous êtes des développeurs, vous voulez écrire du code. Mais c'est comme ça.

Openrewrite nous offre pour cela un format de déclaration de recette en *YAML*. Oh oui youpiiiii [fa dancer].

Recette déclarative (*Declarative recipe*)

Le format proposé par Openrewrite pour recette déclarative permet d'assigner une sous partie de ce qui est possible en Java. Il n'est notamment pas possible d'ajouter des paramètres, ni de renvoyer un visiteur dans une recette déclarative.

Voici un exemple de recette déclarative qui supprime la dépendance `com.github.jtama:toxic` d'un projet *Maven*. La recette doit-être écrite dans un fichier s'appelant `rewrite.yml` et se trouvant soit à la racine du projet, soit dans le répertoire `META-INF/rewrite` :

```yaml
---
type: specs.openrewrite.org/v1beta/recipe (1)
name: com.github.jtama.openrewrite.RemovesThatToxicDependency (2)
displayName: Removes that toxic dependency (3)
description: |
  Migrate from AcmeToxic ☠️ to AcmeHealthy 😇,
  removes dependencies and migrates code.  (4)
tags: (5)
  - acme
  - toxic
recipeList: (6)
  - org.openrewrite.java.ChangeMethodTargetToStatic: (7)
      methodPattern: com.github.jtama.toxic.toxic.BigDecimalUtils value
      fullyQualifiedTargetTypeName: java.math.BigDecimal
  - org.openrewrite.maven.RemoveUnusedProperties:
      properties: .*toxic\.version
  - org.openrewrite.maven.RemoveDependency:
      groupId: com.github.jtama
      artifactId: toxic-library
  - com.github.jtama.openrewrite.VousAllezVoirCeQueVousAllezVoir
---
type: specs.openrewrite.org/v1beta/recipe
name: com.github.jtama.openrewrite.VousAllezVoirCeQueVousAllezVoir
displayName: Ça va vous épater
description: |
  Rech. proj. pr proj. priv. Self Dem. Brt. Poss. S'adr. à l'hô. Mart
tags:
  - acme
preconditions:
  - org.openrewrite.text.Find: (8)
      find: com.github.jtama
recipeList:
  - com.github.jtama.openrewrite.RemoveFooBarUtilsIsEmpty
  - com.github.jtama.openrewrite.RemoveFooBarUtilsStringFormatted
  - com.github.jtama.openrewrite.UseObjectsCompare
```

1. Déclaration du type de recette
2. Nom de la recette
3. Nom affiché lors de l'exécution de la recette
4. Description de la recette
5. Tags pour faciliter la recherche
6. Liste des recettes à exécuter
7. Passage de paramètre à une recette
8. Un exemple de précondition. [fa warning] Attention cette précondition va s'exécuter pour toutes les recettes de la liste.

Comme nous l'avons vu dans l'exemple précédent, permet de construire des recettes complexes en les composant les unes avec les autres.

Deux points d'attention sont à noter :

1. Le fichier doit s'appeler `rewrite.yml`, pas `rewrite.yaml`. 🙄
2. Pour que cette recette puisse s'exécuter, les 3 recettes filles doivent être accessibles dans le *classpath*

```
$ mvn -U org.openrewrite.maven:rewrite-maven-plugin:run \
   -Drewrite.recipeArtifactCoordinates=com.github.jtama:toxic-library-r
   -Drewrite.activeRecipes=com.github.jtama.openrewrite.RemovesThatToxi
```

Distribution

Vous êtes heureux de ce que vous avez fait, vous voulez partager votre recette avec le monde entier. Pour cela, il vous suffit de créer un module *Maven* ou *Gradle* et de le publier. Chacun pourra dès lors utilisez à loisir votre recette.

Le projet devra comprendre le fichier `rewrite.yml` et les dépendances nécessaires pour que la recette puisse s'exécuter.

On code nos recettes [fa pencil]

Pour les chapitres suivants, nous partons du principe que vous voulez vous débarasser d'une dépendance toxique (com.github.jtama:toxic-library:19.666.45-RC18-FINAL) qui comprend les classes suivantes :

```java
package com.github.jtama.toxic;

import java.util.Comparator;
import java.util.List;

public class FooBarUtils {

    public String stringFormatted(String template, Object... args) {
        return String.format(template, args);
    }

    public static boolean isEmpty(String value) {
        if (value == null) return true;
        return value.isEmpty();
    }

    public static <T> boolean isEmpty(List<T> value) {
        if (value == null) return true;
        return value.isEmpty();
    }

    public <T> int compare(T o1, T o2, Comparator<T> comparator) {
        return comparator.compare(o1, o2);
    }
}
```

```java
package com.github.jtama.toxic;

import java.math.BigDecimal;

public class BigDecimalUtils {

    public static BigDecimal valueOf(Long value) {
        return new BigDecimal(value);
    }
}
```

On ne se pose pas de question le code en lui même, dîtes-vous c'est axiome.

Nous allons mettre en oeuvre 2 types de recettes :

- `Refaster template recipes`, ou recettes *refaster*. Simples, mais limitées.
- Full custom java recipes (Bam ! Pas un seul mot français).

Refaster template recipes [fa bolt]

Ces patrons de recettes utilisent *refaster*.

Elles permettent de décrire simplement des templates recettes via du code. L'outillage *OpenRewrite* génère ensuite les recettes complètes à partir de ces templates.

Pour les utiliser il vous faut ajouter les dépendances suivantes à votre projet. Le code suivant est un copier/coller de la documentation officielle :

```
<dependencies>
    <!-- Refaster style recipes need the rewrite-templating annotation
    <dependency>
        <groupId>org.openrewrite</groupId>
        <artifactId>rewrite-templating</artifactId>
    </dependency>

    <!-- If you are developing recipes in Java, you'll need to bring in
    <dependency>
        <groupId>org.openrewrite</groupId>
        <artifactId>rewrite-java</artifactId>
    </dependency>

    <!-- The `@BeforeTemplate` and `@AfterTemplate` annotations are nee
    <dependency>
        <groupId>com.google.errorprone</groupId>
        <artifactId>error_prone_core</artifactId>
        <version>2.19.1</version>
        <scope>provided</scope>
        <exclusions>
            <exclusion>
                <groupId>com.google.auto.service</groupId>
                <artifactId>auto-service-annotations</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.12.1</version>
            <configuration>
                <source>17</source>
                <target>17</target>
```

```
                    <compilerArgs>
                        <arg>-parameters</arg>
                    </compilerArgs>
                    <annotationProcessorPaths>
                        <path>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                            <version>1.18.32</version>
                        </path>
                        <path>
                            <groupId>org.openrewrite</groupId>
                            <artifactId>rewrite-templating</artifactId>
                            <version>1.19.1</version>
                        </path>
                    </annotationProcessorPaths>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

Nous pouvons maintenant créer une classe qui va supprimer les invocations des méthodes `FooBarUtils.isEmpty` :

```
    @RecipeDescriptor(
                name = "Replace `FooBarUtils.isEmptyString(String)` with st
                description = "Replace `FooBarUtils.isEmptyString(String)`
        ) (1)
        public static class RemoveStringIsEmpty {

            @BeforeTemplate
            boolean before(String value) {
                return FooBarUtils.isEmpty(value);
            }

            @AfterTemplate
            boolean after(String value) {
                return value == null || value.isEmpty();
            }

        }
```

1. Le nom et la description de la recette

Les annotations @BeforeTemplate et @AfterTemplate permettent de marquer les méthodes qui seront utilisées pour générer respectivement le template permettant de trouver les invocations à

modifier et le template permettant de générer le code de remplacement.

Le deux méthodes doivent avoir le même nombre de paramètres avec les mêmes types et noms.

Il est possible de grouper les templates de recettes refaster comme suit.

```
package com.github.jtama.openrewrite;

import com.github.jtama.toxic.FooBarUtils;
import com.google.errorprone.refaster.annotation.AfterTemplate;
import com.google.errorprone.refaster.annotation.BeforeTemplate;
import org.openrewrite.java.template.RecipeDescriptor;

import java.util.List;

@RecipeDescriptor(
        name = "Remove `FooBarUtils.isEmpty` methodes usages",
        description = "Replace any usage of `FooBarUtils.isEMpty` metho
public class RemoveFooBarUtilsIsEmpty {

    @RecipeDescriptor(
            name = "Replace `FooBarUtils.isEmptyString(String)` with st
            description = "Replace `FooBarUtils.isEmptyString(String)`
    )
    public static class RemoveStringIsEmpty {

        @BeforeTemplate
        boolean before(String value) {
            return FooBarUtils.isEmpty(value);
        }

        @AfterTemplate
        boolean after(String value) {
            return value == null || value.isEmpty();
        }
    }


    @RecipeDescriptor(
            name = "Replace `FooBarUtils.isEmptyList(List)` with standa
            description = "Replace `FooBarUtils.isEmptyList(List)` with
    )
    public static class RemoveListIsEmpty {

        @BeforeTemplate
        public boolean before(List value) {
            return FooBarUtils.isEmpty(value);
```

```
            }

            @AfterTemplate
            public boolean after(List value) {
                return value == null || value.isEmpty();
            }
        }
    }
```

Dans ce cas, la recette `RemoveFooBarUtilsIsEmptyRecipes` générée contiendra une liste de recette comprenant les recettes `RemoveStringIsEmptyRecipe` et `RemoveListIsEmptyRecipe`.

Dans les faits ce type de recette est relativement restreint. Le code ciblé doit pouvoir s'exprimer dans le bloc d'une méthode, et il sera toujours relativement simple et non paramètrable. Il ne pourra pas non plus retenir le style de formatage du code source d'origine.

Full custom java recipes [fa code]

Toujours pas de français

La recette suivante va remplacer les invocations de `FooBarUtils.stringFormatted` par des invocations de `String.format`. Celle-ci ne peut pas être réalisée avec un template, parce que le nombre de paramètres de ces méthodes ne peut être connu à l'avance.

Nous allons donc devoir passer à l'étape supérieure.

JAVA # TOOLS

Jérôme Tama
Techlead/Architecte/Compagnon du devoir