Процессоры аннотаций, Lombok

Юрий Литвинов
yurii.litvinov@gmail.com

18.05.2017г

Процессоры аннотаций

- Обрабатывают аннотации во время компиляции
  - Можно понимать как плагины к компилятору
- Генерируют код
  - Который потом подаётся на вход компилятору
- Появились в Java 6
- Используются многими библиотеками

API

```
public class MyProcessor extends AbstractProcessor {

    @Override
    public synchronized void init(ProcessingEnvironment env) { }

    @Override
    public boolean process(
        Set<? extends TypeElement> annotations,
        RoundEnvironment env) { }

    @Override
    public Set<String> getSupportedAnnotationTypes() { }

    @Override
    public SourceVersion getSupportedSourceVersion() { }
}
```

API, с Java 7

```
@SupportedSourceVersion(SourceVersion.latestSupported())
@SupportedAnnotationTypes({...})
 public class MyProcessor extends AbstractProcessor {

    @Override
    public synchronized void init(
        ProcessingEnvironment env) { }

    @Override
    public boolean process(
        Set<? extends TypeElement> annoations,
        RoundEnvironment env) { }
}
```

Регистрация процессора

Надо собрать специальный jar-файл

MyProcessor.jar
- com
  - example
    - MyProcessor.class

- META-INF
  - services
    - javax.annotation.processing.Processor

javax.annotation.processing.Processor содержит полностью квалифицированные имена классов-процессоров через перевод строки

Компиляция кода с процессором

Добавить процессор в classpath (на самом деле, в -processorpath у javac)

```
>javac -cp com.example.annotations.jar;
    com.example.annotations.processors.jar
    MyCode.java
```

Пример: фабрика

```
public interface Meal {
  public float getPrice();
}
public class MargheritaPizza implements Meal {
  @Override public float getPrice() {
    return 6.0f;
  }
}
public class CalzonePizza implements Meal {
  @Override public float getPrice() {
    return 8.5f;
  }
}
public class Tiramisu implements Meal {
  @Override public float getPrice() {
    return 4.5f;
  }
}
```

Основной класс

```
public class PizzaStore {

  private MealFactory factory = new MealFactory();

  public Meal order(String mealName) {
    return factory.create(mealName);
  }

  public static void main(String[] args) throws IOException {
    PizzaStore pizzaStore = new PizzaStore();
    Meal meal = pizzaStore.order(readConsole());
    System.out.println("Bill: " + meal.getPrice());
  }
}
```

Фабрика

```
public class MealFactory {

  public Meal create(String id) {
    if (id == null) {
      throw new IllegalArgumentException("id is null!");
    } else if ("Calzone".equals(id)) {
      return new CalzonePizza();
    } else if ("Tiramisu".equals(id)) {
      return new Tiramisu();
    } else if ("Margherita".equals(id)) {
      return new MargheritaPizza();
    }

    throw new IllegalArgumentException("Unknown id = " + id);
  }
}
```

Аннотация @Factory

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.CLASS)
public @interface Factory {

  /** The name of the factory */
  Class type();

  /** The identifier for determining which item should be instantiated */
  String id();
}
```

Пиццы

```
@Factory(id = "Margherita", type = Meal.class)
public class MargheritaPizza implements Meal {
  @Override public float getPrice() {
    return 6f;
  }
}

@Factory(id = "Calzone", type = Meal.class)
public class CalzonePizza implements Meal {
  @Override public float getPrice() {
    return 8.5f;
  }
}
```

Самое интересное: процессор

```
@AutoService(Processor.class)
public class FactoryProcessor extends AbstractProcessor {
  private Types typeUtils;
  private Elements elementUtils;
  private Filer filer;
  private Messager messager;
  private Map<String, FactoryGroupedClasses> factoryClasses
      = new LinkedHashMap<String, FactoryGroupedClasses>();
  @Override
  public synchronized void init(ProcessingEnvironment processingEnv) {
    super.init(processingEnv);
    typeUtils = processingEnv.getTypeUtils();
    elementUtils = processingEnv.getElementUtils();
    filer = processingEnv.getFiler();
    messager = processingEnv.getMessager();
  }
  @Override
  public Set<String> getSupportedAnnotationTypes() {
    Set<String> annotations = new LinkedHashSet<String>();
    annotations.add(Factory.class.getCanonicalName());
    return annotations;
  }
  ...
}
```

Элементы

```
package com.example;    // PackageElement

public class Foo {      // TypeElement

    private int a;      // VariableElement
    private Foo other;  // VariableElement

    public Foo () {}    // ExecuteableElement

    public void setA (  // ExecuteableElement
        int newA        // TypeElement
        ) {}
}
```

Пример работы с элементами

```
TypeElement fooClass = ...;
for (Element e : fooClass.getEnclosedElements()) {
    Element parent = e.getEnclosingElement();
}
```

Вернёмся к процессору

```
@AutoService(Processor.class)
public class FactoryProcessor extends AbstractProcessor {
  ...
  @Override
  public boolean process(
      Set<? extends TypeElement> annotations,
      RoundEnvironment roundEnv) {
    // Iterate over all @Factory annotated elements
    for (Element annotatedElement
        : roundEnv.getElementsAnnotatedWith(Factory.class)) {
      ...
    }
  }
 ...
}
```

FactoryAnnotatedClass

```
public class FactoryAnnotatedClass {
  private TypeElement annotatedClassElement;
  private String qualifiedSuperClassName;
  private String simpleTypeName;
  private String id;
  public FactoryAnnotatedClass(TypeElement classElement)
      throws IllegalArgumentException {
    this.annotatedClassElement = classElement;
    Factory annotation = classElement.getAnnotation(Factory.class);
    id = annotation.id();
    if (StringUtils.isEmpty(id)) {
      throw new IllegalArgumentException(
          String.format("id() in @%s for class %s is null or empty! that's not allowed",
          Factory.class.getSimpleName(), classElement.getQualifiedName().toString()));
    }
    ...
```

FactoryAnnotatedClass (2)

```
public class FactoryAnnotatedClass {
    public FactoryAnnotatedClass(TypeElement classElement)
            throws IllegalArgumentException {
    ...
    try {
        Class<?> clazz = annotation.type();
        qualifiedSuperClassName = clazz.getCanonicalName();
        simpleTypeName = clazz.getSimpleName();
    } catch (MirroredTypeException mte) {
        DeclaredType classTypeMirror = (DeclaredType) mte.getTypeMirror();
        TypeElement classTypeElement = (TypeElement) classTypeMirror.asElement();
        qualifiedSuperClassName = classTypeElement.getQualifiedName().toString();
        simpleTypeName = classTypeElement.getSimpleName().toString();
    }
}
```

FactoryGroupedClasses

```
public class FactoryGroupedClasses {
   private String qualifiedClassName;
   private Map<String, FactoryAnnotatedClass> itemsMap =
         new LinkedHashMap<String, FactoryAnnotatedClass>();
   public FactoryGroupedClasses(String qualifiedClassName) {
      this.qualifiedClassName = qualifiedClassName;
   }
   public void add(FactoryAnnotatedClass toInsert) throws IdAlreadyUsedException {
      FactoryAnnotatedClass existing = itemsMap.get(toInsert.getId());
      if (existing != null) {
         throw new IdAlreadyUsedException(existing);
      }
      itemsMap.put(toInsert.getId(), toInsert);
   }
   public void generateCode(Elements elementUtils, Filer filer) throws IOException {
      ...
   }
}
```

Вернёмся к процессору ещё раз

```
public class FactoryProcessor extends AbstractProcessor {
  @Override
  public boolean process(Set<? extends TypeElement> annotations,
      RoundEnvironment roundEnv) {
    for (Element annotatedElement
        : roundEnv.getElementsAnnotatedWith(Factory.class)) {
      ...
      TypeElement typeElement = (TypeElement) annotatedElement;
      try {
        FactoryAnnotatedClass annotatedClass =
            new FactoryAnnotatedClass(typeElement);
        if (!isValidClass(annotatedClass)) {
          return true;
        }
      } catch (IllegalArgumentException e) {
        error(typeElement, e.getMessage());
        return true;
      }
      ...
  }
```

Продолжим

```
try {
    FactoryAnnotatedClass annotatedClass = new FactoryAnnotatedClass(typeElement);
    if (!isValidClass(annotatedClass)) {
        return true; // Error message printed, exit processing
    }
    // Everything is fine, so try to add
    FactoryGroupedClasses factoryClass =
            factoryClasses.get(annotatedClass.getQualifiedFactoryGroupName());
    if (factoryClass == null) {
        String qualifiedGroupName = annotatedClass.getQualifiedFactoryGroupName();
        factoryClass = new FactoryGroupedClasses(qualifiedGroupName);
        factoryClasses.put(qualifiedGroupName, factoryClass);
    }
    factoryClass.add(annotatedClass);
} catch (IllegalArgumentException e) {
    // @Factory.id() is empty --> printing error message
} catch (IdAlreadyUsedException e) {
    // Already existing
}
```

И генерация кода

```
@Override
public boolean process(Set<? extends TypeElement> annotations,
        RoundEnvironment roundEnv) {
    ...
    try {
        for (FactoryGroupedClasses factoryClass : factoryClasses.values()) {
            factoryClass.generateCode(elementUtils, filer);
        }
    } catch (IOException e) {
        error(null, e.getMessage());
    }
    return true;
}
```

Собственно, генерация

```
public class FactoryGroupedClasses {
  private static final String SUFFIX = "Factory";
  private String qualifiedClassName;
  private Map<String, FactoryAnnotatedClass> itemsMap =
      new LinkedHashMap<String, FactoryAnnotatedClass>();
  ...
  public void generateCode(Elements elementUtils, Filer filer) throws IOException {
    TypeElement superClassName = elementUtils.getTypeElement(qualifiedClassName);
    String factoryClassName = superClassName.getSimpleName() + SUFFIX;
    JavaFileObject jfo = filer.createSourceFile(qualifiedClassName + SUFFIX);
    Writer writer = jfo.openWriter();
    JavaWriter jw = new JavaWriter(writer);
    PackageElement pkg = elementUtils.getPackageOf(superClassName);
    if (!pkg.isUnnamed()) {
      jw.emitPackage(pkg.getQualifiedName().toString());
      jw.emitEmptyLine();
    } else {
      jw.emitPackage("");
    }
```

Собственно, генерация (2)

```
        jw.beginType(factoryClassName, "class", EnumSet.of(Modifier.PUBLIC));
        jw.emitEmptyLine();
        jw.beginMethod(qualifiedClassName, "create", EnumSet.of(Modifier.PUBLIC), "String"
        jw.beginControlFlow("if (id == null)");
        jw.emitStatement("throw new IllegalArgumentException(\"id is null!\")");
        jw.endControlFlow();
        for (FactoryAnnotatedClass item : itemsMap.values()) {
            jw.beginControlFlow("if (\"%s\".equals(id))", item.getId());
            jw.emitStatement("return new %s()", item.getTypeElement().getQualifiedName().toS
            jw.endControlFlow();
            jw.emitEmptyLine();
        }
        jw.emitStatement("throw new IllegalArgumentException(\"Unknown id = \" + id)");
        jw.endMethod();
        jw.endType();
        jw.close();
    }
}
```

Полезные ссылки

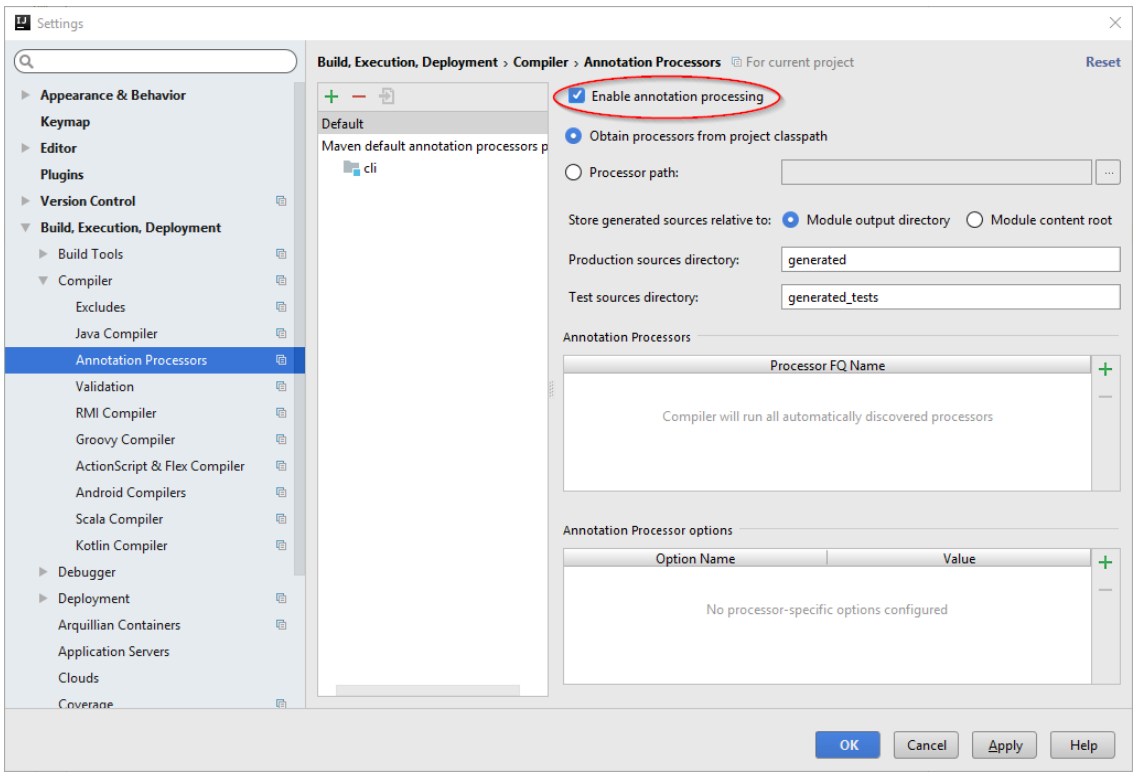
- http://hannesdorfmann.com/annotation-processing/annotationprocessing101
- https://deors.wordpress.com/2011/10/08/annotation-processors/

Lombok

- Очень известная библиотека аннотаций и процессоров, делающая программирование на Java несколько менее неудобным
- https://projectlombok.org/
- Генерирует boilerplate-код
  - Геттеры и сеттеры
  - Конструкторы
  - Метод toString
  - Методы equals и hashCode
  - Всё это сразу
  - Безопасный synchronized
  - Строитель (Builder)

Что надо сделать

1. Подключить артефакт org.projectlombok:lombok как compileOnly-зависимость
2. Включить процессоры аннотаций в IDEA
   - File -> Settings... -> Build, Execution, Deployment -> Compiler -> Annotation Processors



Геттеры и сеттеры

```
@Getter @Setter private boolean employed = true;
@Setter(AccessLevel.PROTECTED) private String name;
```

Эквивалентный код:

```
private boolean employed = true;
private String name;

public boolean isEmployed() {
    return employed;
}

public void setEmployed(final boolean employed) {
    this.employed = employed;
}

protected void setName(final String name) {
    this.name = name;
}
```

@ToString

```
@ToString(callSuper=true,exclude="someExcludedField")
public class Foo extends Bar {
   private boolean someBoolean = true;
   private String someStringField;
   private float someExcludedField;
}
```

Эквивалентный код:

```
public class Foo extends Bar {
   private boolean someBoolean = true;
   private String someStringField;
   private float someExcludedField;

   @java.lang.Override
   public java.lang.String toString() {
      return "Foo(super=" + super.toString() +
         ", someBoolean=" + someBoolean +
         ", someStringField=" + someStringField + ")";
   }
}
```

@EqualsAndHashCode

```
@EqualsAndHashCode(callSuper=true, exclude={ "address", "city", "state", "zip" })
public class Person extends SentientBeing {
   enum Gender { Male, Female }

   @NonNull private String name;
   @NonNull private Gender gender;

   private String ssn;
   private String address;
   private String city;
   private String state;
   private String zip;
}
```

@Data

```
@Data(staticConstructor="of")
public class Company {
    private final Person founder;
    private String name;
    private List<Person> employees;
}
```

@Synchronized

```
private DateFormat format = new SimpleDateFormat("MM-dd-YYYY");

@Synchronized
public String synchronizedFormat(Date date) {
    return format.format(date);
}
```

Эквивалентный код:

```
private final java.lang.Object $lock = new java.lang.Object[0];
private DateFormat format = new SimpleDateFormat("MM-dd-YYYY");

public String synchronizedFormat(Date date) {
    synchronized ($lock) {
        return format.format(date);
    }
}
```

@Builder

```
@Builder
class Example {
    private int foo;
    private final String bar;
}
```

@Builder, эквивалентный код

```
class Example<T> {
    private T foo;
    private final String bar;

    private Example(T foo, String bar) {
        this.foo = foo;
        this.bar = bar;
    }
    public static <T> ExampleBuilder<T> builder() {
        return new ExampleBuilder<T>();
    }
    public static class ExampleBuilder<T> {
        private T foo;
        private String bar;
        private ExampleBuilder() {}
        public ExampleBuilder foo(T foo) {
            this.foo = foo;
            return this;
        }
        public ExampleBuilder bar(String bar) {
            this.bar = bar;
            return this;
        }
        @java.lang.Override public String toString() {
            return "ExampleBuilder(foo = " + foo + ", bar = " + bar + ")";
        }
        public Example build() {
            return new Example(foo, bar);
        }
    }
}
```

@val

```
val x = 10.0;
val y = new ArrayList<String>();
```

- Требует **import lombok.val**;
- Выводит тип из выражения инициализации
- Делает переменную **final**