Università degli Studi di Trento

Web Architectures

Writing an annotation preprocessor

Gianvito Taneburgo

October 26, 2016

1 Introduction

The following report describes the procedure that has been followed to develop an application which is able to transform a suitably annotated POJO into a JavaBean. In particular, the program relies on Java introspection capability to verify the POJO is properly annotated and to detect which source code modifications are required. At the end of the process a new Java file is created.
The Implementation paragraph provides details on how POJOs are inspected and transformed. Some details about Java Reflection have been omitted: the reader is supposed to know the benefits of the technology and the basic mechanisms that make it possible. The Deployment paragraph, on the contrary, describes how to compile and run the application on a sample POJO.
For the remainder of this report the following versions of specifications, frameworks and libraries will be considered:

- Java Platform, Standard Edition: *8*
- Apache Maven: *3.3.9*
- JavaParser: *1.8*

2 Implementation

The application is made up of two parts: one is responsible of checking the input POJO, the other of modifying its source code. The two tasks are respectively encoded into

the *BeanAnnotator.java* and *BeanAnnotatorToolkit.java* files. The *BeanAnnotator* class also provides the main method executed when running the application jar. A third file, *Bean.java*, completes the source code and defines the *@Bean* annotation required for POJOs to be transformed. The annotation is defined by declaring a special interface (*@interface*). No annotation type elements are present inside *Bean*.
Many powerful and flexible Java libraries permit easy code manipulation. Two very important ones are JavaAssist [1] and JavaParser [2] . The entire application could have been developed by only using one of them, nonetheless the POJO checking task has been accomplished with pure standard Java Reflection. The *BeanAnnotator* class, in fact, has no third-party dependencies and shows how powerful are the bultin Java introspection functionalities. On the contrary, the source code manipulation relies on JavaParser in order to avoid tedious and error-prone file parsing duties. The result is an expressive code.
The *BeanAnnotator* class first compiles the input file by using the builtin JavaCompiler class [3]. Since the input file should be a POJO, there is no need to worry about external dependencies. The only expected one is from the *@Bean* interface, but the Java ClassLoader fetches it when running the program itself. By compiling the POJO into a temporary folder, in addition, potential syntactic errors are also automatically detected and no useless transformations are launched. Once the class is compiled, it is dynamically loaded with the method *Class.forName()* [4] and inspected. The instance of the *Class* [5] class is used to verify the presence of the *@Bean* annotation. If the annotation is not found the program halts, otherwise the POJOs constructors get scanned in search for a 0-argument one.
Once the validation process is completed, the file can be transformed with the JavaParse library. By design choice, the manipulations must happen by defining a class that extends a library's abstract class and by overriding some methods. For such reason, the *BeanAnnotatorToolkit* class contains an inner static class, *BeanAnnotatorVisitor*, extending the JavaParse *VoidVisitorAdapter* class. This visitor contains three methods that are automatically called by the library once certain kind of items are detected while scanning the POJO source code. In particular, one visitor is executed when the entire file is loaded, one for each occurrence of new class definition, and the last for each occurrence of new field declaration. The following is the list of modifications performed on the original POJO code:

- At the beginning of the file the required Java EE dependencies are imported with *import javax.persistence.*;*
- The class is annotated with *@Entity* and *@Table(name="CLASSNAME")*
- A 0-argument constructor with empty body is added if absent

[1]http://jboss-javassist.github.io/javassist/
[2]http://javaparser.org/
[3]https://docs.oracle.com/javase/8/docs/api/javax/tools/JavaCompiler.html
[4]https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html#forName
[5]https://docs.oracle.com/javase/8/docs/api/java/lang/Class.html

- A new *int* field *tableIdForCLASSNAME* is added to the class. This field will be used as the object identifier, so it gets annotated with *@Id*, *@GeneratedValue(strategy = GenerationType.AUTO)* and *@Column(name="FIELDNAME")*
- Getter and setter for *tableIdForCLASSNAME* are generated as well
- Each preexisting field is annotated with *@Column(name="FIELDNAME")*

3 Deployment

Download the *src.zip* file into a folder. All the following commands have to be executed from the download directory.
To unzip the archive type:

```
$ unzip src.zip
```

To compile and package the application type:

```
$ mvn clean install package
```

To run the application and convert the *TestPojo* class provided type:

```
$ java -jar target/bean-generator-v1.0.jar TestPojo.java
```

The new class will be created inside the *generated* folder.
One should be able to check how the original POJO has been successfully transformed.
Original *TestPojo.java* file:

```
@Bean
public class TestPojo {

    private String value;

    private Integer param;

    public TestPojo(String value, Integer param) {
        this.value = value;
        this.param = param;
    }

    public TestPojo(String value) {
        this.value = value;
        this.param = 42;
    }

    public String getValue() {
        return this.value;
    }

    public Integer getParam() {
        return this.param;
    }

    public void setValue(String value) {
        this.value = value;
    }

    public void setParam(Integer param) {
        this.param = param;
    }
}
```

New *TestPojo.java* file:

```
import javax.persistence.*;

@Bean
@Entity
@Table(name = "TESTPOJO")
public class TestPojo {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "tableIdForTestPojo")
    private int tableIdForTestPojo;

    public TestPojo() {
    }

    @Column(name = "value")
    private String value;

    @Column(name = "param")
    private Integer param;

    public TestPojo(String value, Integer param) {
        this.value = value;
        this.param = param;
    }

    public TestPojo(String value) {
        this.value = value;
        this.param = 42;
    }

    public String getValue() {
        return this.value;
    }

    public Integer getParam() {
        return this.param;
    }

    public void setValue(String value) {
        this.value = value;
    }

    public void setParam(Integer param) {
        this.param = param;
    }

    public int getTableIdForTestPojo() {
        return this.tableIdForTestPojo;
    }

    public void setTableIdForTestPojo(int tableIdForTestPojo) {
        this.tableIdForTestPojo = tableIdForTestPojo;
    }
}
```