

Compiling annotations

Introduction

When you are writing your annotation in JavaDoc, you need to post-compile your classes in order to parse the sources, retrieve the JavaDoc annotations, process and validate the annotations and finally put them into the bytecode as regular Java 5 RuntimeVisible annotations.

This is done by using the backport175 compiler, `AnnotationC`.

AnnotationC compiler

You can run `AnnotationC` either from the command line or using the [Ant task](#).

You invoke the compiler like this:

```
java [options...]  
org.codehaus.backport175.compiler.Annotation  
C  
    [-verbose]  
    -src <path to src dir>  
    -classes <path to classes dir>  
    [-dest <path to destination dir>]  
    [-config <optional property file(s) for  
annotations>]
```

The last option `-config <property file(s)>` points to the (or several files separated by classpath separator `;` or `:` depending on you OS) property file which defines annotation aliases (or imports) by mapping the names to the fully qualified class names of the annotation interface.

Note that if you are using the `-dest` option, the anonymous inner classes will not be copied to the destination directory, since the anonymous classes are not taken into account by the annotation compiler. In such a case it is recommended to add the following (if using *Ant*) just after the call to `AnnotationC` when the `-dest` option is used: (adapt according to the directories you are using)

```
<copy todir="classes/annotated"
overwrite="false">
  <fileset dir="classes/regular"/>
</copy>
```

Annotation definition file

Since the annotations are written in JavaDoc, there is no way of doing imports of the annotations interfaces.

This leaves you two options:

- either you use the fully qualified name of the annotation interface when writing the the annotations, e.g.:

```
/**
 * @org.codehaus.backport175.annotation.OneWay
 */
public void method() {
    ...
}
```

which can be a bit verbose and cumbersome in the long run.

- or you define *aliases* (shortcuts/abbreviations), for the annotations and map these to the fully qualified class names for the annotation interfaces. This is done using the `annotations.properties` file.

Example:

```
OneWay = org.codehaus.backport175.annotation.OneWay
TwoWay = org.codehaus.backport175.annotation.TwoWay
```

If we now feed this property file to the compiler we are able to write the annotations like this instead:

```
/**
 * @OneWay
 */
public void method() {
    ...
}
```

You can have more than one `annotation.properties` file (see the compiler section) and those are not at all required at runtime.

Rules for annotation naming consistency

It is important to remember that if you are using an alias through a property file, the annotation name is still the fully qualified name of the interface class for the annotation when using the `Annotations.getAnnotation` API.

If you decide to use an alias, then you have to use this alias in the whole sources. It is not possible to use from time to time in the source an alias or the fully qualified name, since backport175 has to ensure an element cannot be annotated twice with the same annotation, as per JSR-175.

If you decide to use the fully qualified name, the same rule applies, it needs to stay consistent.

If the annotation is a nested class like `Target.OneWay` below, you can use a `$` in the name (as when using `java.lang.reflect.*` API) or a `.` which can be more readable.

Once again, this choice has to be consistent thus one or the other has to be used across the code base but not both:

```
package demo;
/**
 * @demo.Target$OneWay
 */
public class Target {
    public static interface OneWay {}
}
```

or (with dot)

```
package demo;
/**
 * @demo.Target.OneWay
 */
public class Target {
    public static interface OneWay {}
}
```

Annotation default values

JSR-175 annotation support default values. This is supported in backport175 with the `org.codehaus.backport175.DefaultValue` special annotation, that only accepts one single anonymous value whose type must match the one of the annotation element which has this default value :

```
package demo;
/**
 * @demo.Target.OneWayDefaulted
 */
public class TargetUsingDefaultValue {

    public static interface OneWayDefaulted {
        /**
         *
         * @org.codehaus.backport175.DefaultValue
         * ("default message")
         */
        String message();
    }
}

/**
 * @demo.Target.OneWayDefaulted
 * (message="this is not the default message
 * !")
 */
public class TargetNOTUsingDefaultValue {

    ...
}
```