SpotBugs Maven Plugin

v. 3.1.4-SNAPSHOT

Project Documentation

hazendaz

2018-06-12

Table Of Content

Table Of Content i
1. Introduction .....
2. Goals .....
3. Usage .....
4. FAQ 8
5. Sample Report .....
6. Sample XML xdoc Report .....
7. Sample Legacy XML Report .....
8. Multi-Module Configuration .....
9. Violation Checking Configuration .....

1 Introduction

1.1 SpotBugs Maven Plugin

1.1.1 Please Note - This version is using Spotbugs 3.1.3.

SpotBugs looks for bugs in Java programs. It is based on the concept of bug patterns. A bug pattern is a code idiom that is often an error. Bug patterns arise for a variety of reasons:

- Difficult language features
- Misunderstood API methods
- Misunderstood invariants when code is modified during maintenance
- Garden variety mistakes: typos, use of the wrong boolean operator

SpotBugs uses static analysis to inspect Java bytecode for occurrences of bug patterns. We have found that SpotBugs finds real errors in most Java software. Because its analysis is sometimes imprecise, SpotBugs can report false warnings, which are warnings that do not indicate real errors. In practice, the rate of false warnings reported by SpotBugs is generally less than 50%.

SpotBugs is free software, available under the terms of the Lesser GNU Public License. It is written in Java, and can be run with any virtual machine compatible with Java 8. It can analyze programs written for any version of Java. SpotBugs was originally developed by Bill Pugh. It is maintained by Bill Pugh, David Hovemeyer, and a team of volunteers.

SpotBugs uses BCEL to analyze Java bytecode. It uses dom4j for XML manipulation.

This introduction is an excerpt from the Facts Sheet at SpotBugs home page.

To see more documentation about SpotBugs' options, please see the SpotBugs Manual.

1.1.2 Please Note

As of version 3.1.0, you will need to use JDK 8 to run this plugin. This is a requirement imposed by Spotbugs.

2 Usage

2.1 Usage *version*3.1.4-SNAPSHOT */version* The following examples describe the basic usage of the SpotBugs plugin.

2.1.1 Generate SpotBugs Report As Part of the Project Reports

To generate the SpotBugs report as part of the Project Reports, add the SpotBugs plugin in the `<reporting>` section of your `pom.xml`.

```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>com.github.spotbugs</groupId>
        <artifactId>spotbugs-maven-plugin</artifactId>
        <version>3.1.4-SNAPSHOT</version>
      </plugin>
    </plugins>
  </reporting>
  ...
</project>
```

Then, execute the site plugin to generate the report.

```
mvn site
```

2.1.2 Generate SpotBugs xdoc Report As Part of the Project Reports

To generate the SpotBugs xdoc report as part of the Project Reports, add the SpotBugs plugin in the `<reporting>` section of your `pom.xml`. This will be the same report as that of the Maven 1 SpotBugs report. It is also the format used by Hudson. The output file will be written as `spotbugs.xml` to either the default output directory of ${project.build.directory} or by that started in the `<xmlOutputDirectory>` option.

```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>com.github.spotbugs</groupId>
        <artifactId>spotbugs-maven-plugin</artifactId>
        <version>3.1.4-SNAPSHOT</version>
        <configuration>
          <xmlOutput>true</xmlOutput>
          <!-- Optional directory to put spotbugs xdoc xml report -->
          <xmlOutputDirectory>target/site</xmlOutputDirectory>
        </configuration>
      </plugin>
    </plugins>
  </reporting>
  ...
</project>
```

Then, execute the site plugin to generate the report.

```
mvn site
```

2.1.3 Filter bugs to report

To filter the classes and methods which are analyzed or omitted from analysis you can use filters. The filters allow specifying by class and method which bug categories to include/exclude in/from the reports. The filter format specification also contains useful examples.

```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>com.github.spotbugs</groupId>
        <artifactId>spotbugs-maven-plugin</artifactId>
        <version>3.1.4-SNAPSHOT</version>
        <configuration>
          <excludeFilterFile>spotbugs-exclude.xml</excludeFilterFile>
          <includeFilterFile>spotbugs-include.xml</includeFilterFile>
        </configuration>
      </plugin>
    </plugins>
  </reporting>
  ...
</project>
```

Then, execute the site plugin to generate the report.

```
mvn site
```

2.1.4 Specifying which bug filters to run

To filter the classes and methods which are analyzed or omitted from analysis you can use filters. The filters allow specifying by class and method which bug categories to include/exclude in/from the reports. The filter format specification also contains useful examples.

```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>com.github.spotbugs</groupId>
        <artifactId>spotbugs-maven-plugin</artifactId>
        <version>3.1.4-SNAPSHOT</version>
        <configuration>
          <excludeFilterFile>spotbugs-exclude.xml</excludeFilterFile>
          <includeFilterFile>spotbugs-include.xml</includeFilterFile>
        </configuration>
      </plugin>
    </plugins>
  </reporting>
  ...
</project>
```

Then, execute the site plugin to generate the report.

```
mvn site
```

2.1.5 Specifying which bug detectors to run

The `visitors` option specifies a comma-separated list of bug detectors which should be run. The bug detectors are specified by their class names, without any package qualification. By default, all detectors which are not disabled are run.

```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>com.github.spotbugs</groupId>
        <artifactId>spotbugs-maven-plugin</artifactId>
        <version>3.1.4-SNAPSHOT</version>
        <configuration>
          <visitors>FindDeadLocalStores,UnreadFields</visitors>
        </configuration>
      </plugin>
    </plugins>
  </reporting>
  ...
</project>
```

Then, execute the site plugin to generate the report.

```
mvn site
```

2.1.6 Specifying which bug detectors to skip

The `omitVisitors` option is like the visitors attribute, except it specifies detectors which will not be run.

```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>com.github.spotbugs</groupId>
        <artifactId>spotbugs-maven-plugin</artifactId>
        <version>3.1.4-SNAPSHOT</version>
        <configuration>
          <omitVisitors>FindDeadLocalStores,UnreadFields</omitVisitors>
        </configuration>
      </plugin>
    </plugins>
  </reporting>
  ...
</project>
```

Then, execute the site plugin to generate the report.

```
mvn site
```

2.1.7 Specifying which classes to analyze

The `onlyAnalyze` option restricts analysis to the given comma-separated list of classes and packages.

```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>com.github.spotbugs</groupId>
        <artifactId>spotbugs-maven-plugin</artifactId>
        <version>3.1.4-SNAPSHOT</version>
        <configuration>
         <onlyAnalyze>com.github.spotbugs.spotbugs.*</onlyAnalyze>
        </configuration>
      </plugin>
    </plugins>
  </reporting>
  ...
</project>
```

Then, execute the site plugin to generate the report.

```
mvn site
```

2.1.8 Using Third party or your own detectors

The `pluginList` option specifies a comma-separated list of optional BugDetector Jar files to add.

```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>com.github.spotbugs</groupId>
        <artifactId>spotbugs-maven-plugin</artifactId>
        <version>3.1.4-SNAPSHOT</version>
        <configuration>
          <pluginList>myDetectors.jar, yourDetectors.jar</pluginList>
        </configuration>
      </plugin>
    </plugins>
  </reporting>
  ...
</project>
```

Then, execute the site plugin to generate the report.

```
mvn site
```

2.1.9 Using Detectors from a Repository

The `plugins` option defines a collection of PluginArtifact to work on. (PluginArtifact contains groupId, artifactId, version, type.)

```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>com.github.spotbugs</groupId>
        <artifactId>spotbugs-maven-plugin</artifactId>
        <version>3.1.4-SNAPSHOT</version>
        <configuration>
          <plugins>
            <plugin>
              <groupId>com.timgroup</groupId>
              <artifactId>spotbugs4jmock</artifactId>
              <version>0.2</version>
            </plugin>
          </plugins>
        </configuration>
      </plugin>
    </plugins>
  </reporting>
  ...
</project>
```

Then, execute the site plugin to generate the report.

```
mvn site
```

2.1.10 Launch the Spotbugs GUI

This will launch the SpotBugs GUI configured for this project and will open the spotbugsXml.xml file if present. It therefore assumes a pom.xml with the minimum as follows.

```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>com.github.spotbugs</groupId>
        <artifactId>spotbugs-maven-plugin</artifactId>
        <version>3.1.4-SNAPSHOT</version>
        <configuration>
          <!-- Optional directory to put spotbugs xml report -->
        </configuration>
      </plugin>
    </plugins>
  </reporting>
  ...
</project>
```

Then, execute the spotbugs plugin with the gui option.

```
mvn spotbugs:gui
```

3 FAQ

3.1 Frequently Asked Questions

1. Is there an easy way to generate the aggregate report ?
2. How do I avoid OutOfMemory errors?

Is there an easy way to generate the aggregate report ?

Unfortunately Maven's internal support for report aggregation is rather poor and does have a number of limitations.

This may be addressed in the future by a module like dashboard to get an aggregate report.

[top]

How do I avoid OutOfMemory errors?

When running spotbugs on a project, the default heap size might not be enough to complete the build. For now there is no way to fork spotbugs and run with its own memory requirements, but the following system variable will allow you to do so for Maven:

```
export MAVEN_OPTS=-Xmx384M
```

You can also use the fork option which will for a new JVM. You then use the maxHeap option to control the heap size.

[top]

4 Multi-Module Configuration

4.1 Multimodule Configuration

Note: This implemented in version 2.0 of the Spotbugs plugin.

Credit: This is a shameless plagiarization of the Checkstyle plugin for consistency and due to my laziness.

Configuring the Spotbugs plugin for use within large multimodule projects can be done, but it requires a little setup.

This example will use a mysterious project called *whizbang*. This is what the structure of that project looks like:

```
whizbang
|-- pom.xml
|-- core
|   `-- pom.xml
|-- gui
|   `-- pom.xml
|-- jmx
|   `-- pom.xml
`-- src
```

4.1.1 Create a subproject to house the resources

We'll start by adding another sub project that will house our common configuration. Let's call it *build-tools*. In it we put the resources that we want to include. In this example, we will add configuration files for the Spotbugs plugin. Configuration files for other plugins, like the PMD and Checkstyle plugin, can be included in the same subproject if you like. We will create another directory and call it *whiz-progs* and create a pom.xml file. We will move our core, gui, and jmx modules to *whiz-progs*.

```
whizbang
|-- pom.xml
|-- build-tools
|   |-- src
|   |   `-- main
|   |       `-- resources
|   |           `-- whizbang
|   |               |-- checkstyle.xml
|   |               |-- lib-filter.xml
|   |               `-- LICENSE.TXT
|   `-- pom.xml
|`-- src
|
`-- whiz-progs
    |-- pom.xml
    |-- core
    |-- gui
    `-- jmx
```

Tip: put the resources into a subdirectory that you can ensure will be unique and not conflict with anyone else.

4.1.2 Configure the top level pom

The top level pom jusr references the two modules

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example.whizbang</groupId>
  <artifactId>whizbang-parent</artifactId>
  <version>1.0</version>
  <packaging>pom</packaging>
  <name>WhizBang Parent</name>
  <modules>
    <module>build-tools</module>
    <module>modules</module>
  </modules>
</project>
```

4.1.3 Configure the other projects to build-tools

Now we can include the Spotbugs configuration in the whiz-progs `pom.xml`.

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.example.whizbang</groupId>
    <artifactId>whizbang-parent</artifactId>
    <version>1.0</version>
    <relativePath>../pom.xml</relativePath>
  </parent>
  <packaging>pom</packaging>
    <artifactId>whiz-progs</artifactId>
  <name>WhizBang Programs</name>
  <build>
    <extensions>
      <extension>
        <groupId>com.example.whizbang</groupId>
        <artifactId>build-tools</artifactId>
        <version>1.0</version>
      </extension>
    </extensions>
  </build>
  <reporting>
    <plugins>
      <plugin>
        <groupId>com.github.spotbugs</groupId>
        <artifactId>spotbugs-maven-plugin</artifactId>
        <version>3.1.4-SNAPSHOT</version>
        <configuration>
          <effort>Max</effort>
          <threshold>Low</threshold>
          <includeFilterFile>whizbang/lib-filter.xml</includeFilterFile>
        </configuration>
      </plugin>
    </plugins>
  </reporting>
  <modules>
    <module>core</module>
    <module>jmx</module>
    <module>gui</module>
  </modules>
</project>
```

Once you are done with that, ensure that you do not include spotbugs-maven-plugin in your sub modules, as their definition and configuration, will override the top level parent pom's definition.

Based on the Spotbugs plugin configuration above, the values of `includeFilterFile` will be resolved from the classpath. The *build-tools* jar was included in the classpath when it was declared as an dependency to the plugin.

Note: For the classpath reference, the build-tools was referred to as an extension and not as a plugin dependency. This is due to the fact that if it is declared as a plugin dependency, Maven will not download it from the internal repository and would just look for it in ibiblio.

Lastly, kick off a build of the site.

```
mvn site
```

Every sub project will now use the same Spotbugs setup and configuration.

5 Violation Checking Configuration

5.1 Violation Checking

The `spotbugs:check` goal allows you to configure your build to fail if any errors are found in the SpotBugs report.

The following code fragment enables the check in a build during the `verify` phase. The check will fail if any of the filter triggers in the include file are met.

```
<project>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>com.github.spotbugs</groupId>
        <artifactId>spotbugs-maven-plugin</artifactId>
        <version>3.1.4-SNAPSHOT</version>
        <configuration>
          <effort>Max</effort>
          <threshold>Low</threshold>
          <xmlOutput>true</xmlOutput>
        </configuration>
        <executions>
          <execution>
            <goals>
              <goal>check</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```