I'm not robot
reCAPTCHA
Continue

Java jacoco coverage report

JaCoCo is a free code coverage library for Java created by the EclEmma team based on the experience gained from using and integrating existing libraries for many years. Snapshot Builds The master branch of JaCoCo is automatically created and published. Because of the test-driven development approach, each build is considered fully functional. For the latest features and bug fixes, see the change history. SonarQube code quality metrics of the current JaCoCo implementation are available on SonarCloud.io. Documentation Download (Maven Repository) Coverage Report Release Builds The official releases builds are available for download below. JaCoCo is also available in the Maven repository. Contact/Feedback Please do not hesitate to contact the JaCoCo and EclEmma Users group and provide feedback. We are particularly curious about your experience with using or integrating JaCoCo, missing features that we don't have on the output list yet, and any suggestions for feature or implementation improvements. When I started using Java 7, I immediately noticed that the Cobertura Maven plugin doesn't support it. That was a huge problem for me because I used code coverage reports every day. I did some research and found the JaCoCo Code Cover Library. It looked interesting and I decided to give it. The problem was that the configuration was really difficult and a lot of time in the way of ice. I read numerous tutorials just to find out that the instructions in them don't work for me. Then I ran into this blog post and everything falls in place. Although this blog post was extremely valuable to me, it is a bit vague. I felt that a more detailed explanation about the use of the JaCoCo Maven plugin would be valuable. This blog post describes how we can create code coverage reports for unit and integration testing using the JaCoCo Maven plugin. The requirements of our build are as follows: Our build must generate code coverage reports for component and integration tests when the tests are run. The code coverage reports must be created in separate directories. In other words, the unit test code coverage report must be created in a directory other than the integration test code coverage report. Note: The sample application of this blog post is based on the sample application of my blog post called Integration Testing with Maven. If you haven't read it yet, I recommend that you read it before reading this blog post. Configure the JaCoCo Maven plugin We use the JaCoCo Maven plugin for two purposes: it gives us access to the JaCoCo runtime agent, which records execution coverage data. It Code coverage reports from the execution data recorded by the JaCoCo runtime agent. We can configure the JaCoCo Maven plugin by following these steps: Add the JaCoCo Maven plugin to the plugin area of our POM file. Configure the code coverage report for unit tests. Configure the code coverage report for integration testing. These are described below with further details. Add the JaCoCo Maven plugin to the POM file We can add the JaCoCo Maven plugin to our POM file by adding the following plugin declaration to its plugin area: <plugin> <groupId>org.jacoco</groupId> <artifactId>jacoco-maven-plugin>artifact <version>0.7.5.201505241946</version> </plugin> If you are using Java 7 (or earlier) and the 0.7.x version is not working properly, downgrade to version 0.6.5.201403032054. Let's move on and find out how we can configure the code coverage for our component tests. Configure the code coverage report for unit tests We can configure the code coverage report for unit tests by adding two runs to the plugin declaration. These statements are described below: The first run creates a property that references the JaCoCo runtime agent. Make sure that the execution data is written to target/coverage-reports/jacoco-ut.exec. Set the name of the property to surefireArgLine. The value of this property is passed as a VM argument when our unit tests are run. The second run creates the unit test code coverage report after unit tests have been run. Ensure that the execution data is read from the target/coverage-reports/jacoco-ut.exec file and that the code coverage report is written to the target/site/jacoco-ut directory. The relevant part of our plugin configuration looks like this: <plugin> <groupId>org.jacoco</groupId> <artifactId>jacoco-maven-plugin</artifactId> <version>0.7.5.201505241946</version> <executions> <!-- Prepares the property pointing to the JaCoCo runtime agent that is passed as VM argument when Maven the Surefire plugin is executed. --> <execution> <id>pre-unit-test</id> <goals> <goal>prepare-agent</goal> </goals> <configuration> <!-- sets the path to the file that contains the data execution. --> <destFile>.project.build.directory/coverage-reports/jacoco-ut.exec</destFile> <!-- Sets the name of the property containing the settings for JaCoCo runtime agent. --> <propertyName>surefireArgLine</propertyName> </configuration> </execution> <!-- Ensures that the code coverage report for unit tests is created after unit tests have been run. --> <execution> <id>Post-Unit Test Report</id> <phase></phase> <goals> <goal></goal> </goals> <configuration> <!-- Sets the path --> <!-- sets the output directory for the code coverage report. --> <outputDirectory>.project.report.outputDirectory/jacoco-ut</outputDirectory> </configuration> </execution> </executions> </plugin> Let's figure out how to configure the code coverage report for our integration tests. Configure the code coverage report for integration testing We can configure the code coverage report for integration tests by adding two executions to the plugin declaration. These explanations are described below: creates a property that references the JaCoCo runtime agent. Make sure that the execution data is written to target/coverage-reports/jacoco-it.exec. Set the name of the property to failsafeArgLine. The value of this property is passed as a VM argument when our integration tests are run. Create an execution that creates the code coverage report for integration tests after integration tests have been run. Make sure that the execution data is read from the target/coverage-reports/jacoco-it.exec file and that the code coverage report is written to the target/site/jacoco-it directory. The relevant part of our plugin configuration looks like this: <plugin> <groupId>org.jacoco</groupId> <artifactId>jacoco-maven-plugin</artifactId> <version>0.7.5.201505241946</version> <executions> <!-- The Executions required by unit tests are omitted. --> <!-- Prepares the property pointing to the JaCoCo runtime agent which is passed as VM argument when Maven the Failsafe plugin is executed. --> <execution> <id>pre-integration-test pre-integration-test</id> <phase></phase> <goals> <goal>prepare-agent</goal> </> &lt !--;configuration> --> <destFile>reports/jacoco-it.exec</destFile> <!-- Sets the name of the property containing the settings for JaCoCo runtime agent. --> <propertyName>failsafeArgLine</propertyName> </configuration> </execution> <!-- Ensures that the code coverage report for integration tests after integration tests have been run. --> <execution> <id>post-integration-test</id> <phase>post-integration-test</phase> <goals> <goal>report</goal> </goals> &lt !--;configuration> --> <dataFile>.project.build.directory/coverage-reports/jacoco-it.exec</dataFile> <!-- Sets the output directory for the code coverage report. --> <outputDirectory>.project.report.outputDirectory/jacoco-it</outputDirectory> </configuration> </execution> </executions> </plugin> That's. We have now configured the JaCoCo Maven plugin. Our next step is to configure the Maven Surefire plugin. Let's figure out how we can do this. Configure the Maven Surefire plugin We use the Maven Surefire plugin to run the unit tests of our sample application. Because we want to create a code coverage report for our unit tests, we need to make sure that the JaCoCo agent runs when our unit tests are run. We can ensure this by increasing the value of the als Wert des konfigurationsparameters argLine hinzufügen. Die Konfiguration des Maven Surefire Plugins sieht wie folgt aus (die erforderliche Änderung wird hervorgehoben): <plugin> <groupId>org.apache.maven.plugins</groupId> <artifactId>maven-surefire-plugin</artifactId> <version>2.15</version> <configuration> <!-- Sets the VM argument line used when unit tests are run. --> <argLine></argLine> <!-- Skips unit tests if the value of skip.unit.tests property is true </configuration> </plugin> </plugin> <skipTests>skip.unit.tests</skipTests> <!-- Excludes integration tests when unit tests are run. --> <excludes> <exclude>**/IT*.java</exclude> </excludes> We're almost done. The only thing we need to do is configure the Maven Failsafe plugin. Let's find out how we can do it. Configure the Maven Failsafe plugin The integration tests of our sample application are run by the Maven Failsafe plugin. Because we want to create a code coverage report for our integration tests, we need to make sure that the JaCoCo agent runs when our integration tests are run. We can do this by adding the value of the failsafeArgLine property as the value of the argLine configuration parameter. The configuration of the Maven Failsafe plugin looks like this (the required change is highlighted): <plugin> <groupId>org.apache.maven.plugins</groupId> <artifactId>maven-failsafe-plugin</artifactId> <version>2.15</version> <executions> <!-- Ensures that both integration-test and verify goals of the Fail Masafeven plugin are executed. --> <execution> <id>integration-tests integration-test</id> <goals> <goal></goal> <goal>verify</goal> </goals> <configuration> <!-- Sets the VM argument --> <argLine>'failsafeArgLine'''''''