

Project Module: hadoop-common-project/hadoop-common Configuration Parameter: io.file.buffer.size Method: LineReader(InputStream in, Configuration conf)

Write a test that tests the configuration parameter used in the method.

GitHub Copilot

First, let's plan the test in pseudocode:

Import necessary libraries and modules.

Create a test class.

Define a setup method to initialize the Configuration object and the InputStream.

Define a test method:

Set the io.file.buffer.size configuration parameter to a specific value.

Create a LineReader object using the InputStream and the Configuration object.

Assert that the buffer size in the LineReader object equals the value set in the configuration.

Now, let's write the test in Java:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.IOUtils;
import org.junit.Before;
import org.junit.Test;
import java.io.ByteArrayInputStream;
import java.io.InputStream;
import java.lang.reflect.Field;
```

```
import static org.junit.Assert.assertEquals;
```

```
public class LineReaderTest {
    private Configuration conf;
    private InputStream in;
```

```
    @Before
```

```
    public void setup() {
        conf = new Configuration();
        in = new ByteArrayInputStream("test data".getBytes());
    }
```

```
    @Test
```

```
    public void testBufferSizeConfig() throws Exception {
        int expectedBufferSize = 128 * 1024; // Set your expected buffer size
        conf.setInt("io.file.buffer.size", expectedBufferSize);
```

```
        LineReader reader = new LineReader(in, conf);
```

```

    Field bufferSizeField = LineReader.class.getDeclaredField("bufferSize");
    bufferSizeField.setAccessible(true);
    int actualBufferSize = (int) bufferSizeField.get(reader);

    assertEquals(expectedBufferSize, actualBufferSize);

    IOUtils.closeStream(reader);
}
}

```

This test uses reflection to access the private `bufferSize` field in the `LineReader` class. Please note that this approach breaks encapsulation and should be used sparingly and with caution. It's often better to test the public API and the behavior of the class, rather than its internal state. However, in this case, it seems to be the only way to verify that the configuration parameter is being used correctly.

How to test this test case and how to verify the quality of the test generated?

GitHub Copilot

To test this case, you would typically run it as part of your project's test suite. In a Maven project, you can run the test suite with the `mvn test` command in the terminal. If you're using an IDE like Visual Studio Code, you can also run the test directly from the IDE.

To verify the quality of the test, you can consider the following factors:

Coverage: Does the test cover all the important parts of the method under test? In this case, the test checks whether the `io.file.buffer.size` configuration parameter is correctly used to set the `bufferSize` field in the `LineReader` class.

Correctness: Does the test correctly check the behavior of the method under test? The test should fail if the method doesn't work as expected, and pass otherwise.

Robustness: Does the test handle edge cases? For example, what happens if the configuration parameter is set to an invalid value?

Maintainability: Is the test easy to understand and maintain? A good test should be clear and concise, and avoid unnecessary dependencies.

Performance: Does the test run quickly? Slow tests can slow down the development process.

You can also use tools like code coverage analyzers to measure how much of your code is covered by tests. However, keep in mind that high code coverage doesn't necessarily mean that

your tests are good. It's also important to write meaningful tests that check the actual behavior of your code.

Could you provide some invalid values and how to check the test if it fails for these?

GitHub Copilot

Sure, in the context of the `io.file.buffer.size` configuration parameter, invalid values could be negative numbers or zero, as buffer size cannot be less than or equal to zero.

Here's how you could modify the test to check for these invalid values:

```
@Test
public void testInvalidBufferSizeConfig() throws Exception {
    int invalidBufferSize = -1; // or 0
    conf.setInt("io.file.buffer.size", invalidBufferSize);

    try {
        LineReader reader = new LineReader(in, conf);
        fail("Expected an IllegalArgumentException to be thrown");
    } catch (IllegalArgumentException e) {
        // Test passed
    }
}
```

In this test, we're expecting the `LineReader` constructor to throw an `IllegalArgumentException` when given an invalid buffer size. If it doesn't, the test fails.

Please note that this test assumes that the `LineReader` constructor checks the buffer size and throws an `IllegalArgumentException` if it's invalid. If it doesn't, you might need to modify the `LineReader` class to add this check, or handle the invalid buffer size in some other way.

```
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running org.apache.hadoop.util.CLineReaderTest
[ERROR] Tests run: 1, Failures: 0, Errors: 1, Skipped: 0, Time elapsed: 0.425 s <<< FAILURE! -
in org.apache.hadoop.util.CLineReaderTest
[ERROR] testBufferSizeConfig(org.apache.hadoop.util.CLineReaderTest) Time elapsed: 0.341
s <<< ERROR!
java.lang.NegativeArraySizeException: -1
    at org.apache.hadoop.util.LineReader.<init>(LineReader.java:82)
    at org.apache.hadoop.util.LineReader.<init>(LineReader.java:95)
```

```
    at
org.apache.hadoop.util.CLineReaderTest.testBufferSizeConfig(CLineReaderTest.java:47)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at
java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at
java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.base/java.lang.reflect.Method.invoke(Method.java:566)
    at
org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:59)
    at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
    at
org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:56)
    at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:17)
    at org.junit.internal.runners.statements.RunBefores.evaluate(RunBefores.java:26)
    at org.junit.runners.ParentRunner$3.evaluate(ParentRunner.java:306)
    at
org.junit.runners.BlockJUnit4ClassRunner$1.evaluate(BlockJUnit4ClassRunner.java:100)
    at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:366)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:103)
    at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:63)
    at org.junit.runners.ParentRunner$4.run(ParentRunner.java:331)
    at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:79)
    at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:329)
    at org.junit.runners.ParentRunner.access$100(ParentRunner.java:66)
    at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:293)
    at org.junit.runners.ParentRunner$3.evaluate(ParentRunner.java:306)
    at org.junit.runners.ParentRunner.run(ParentRunner.java:413)
    at org.apache.maven.surefire.junit4.JUnit4Provider.execute(JUnit4Provider.java:365)
    at
org.apache.maven.surefire.junit4.JUnit4Provider.executeWithRerun(JUnit4Provider.java:273)
    at
org.apache.maven.surefire.junit4.JUnit4Provider.executeTestSet(JUnit4Provider.java:238)
    at org.apache.maven.surefire.junit4.JUnit4Provider.invoke(JUnit4Provider.java:159)
    at
org.apache.maven.surefire.booter.ForkedBooter.invokeProviderInSameClassLoader(ForkedBooter.java:384)
    at
org.apache.maven.surefire.booter.ForkedBooter.runSuitesInProcess(ForkedBooter.java:345)
    at org.apache.maven.surefire.booter.ForkedBooter.execute(ForkedBooter.java:126)
    at org.apache.maven.surefire.booter.ForkedBooter.main(ForkedBooter.java:418)
```

[INFO]
[INFO] Results:
[INFO]
[ERROR] Errors:
[ERROR] CLineNumberTest.testBufferSizeConfig:47 » NegativeArraySize -1
[INFO]
[ERROR] Tests run: 1, Failures: 0, Errors: 1, Skipped: 0
[INFO]
[ERROR] There are test failures.

Please refer to
/home/nvadde2/hadoop/hadoop-common-project/hadoop-common/target/surefire-reports for the
individual test results.

Please refer to dump files (if any exist) [date].dump, [date]-jvmRun[N].dump and
[date].dumpstream.

[INFO]
[INFO] --- maven-antrun-plugin:1.7:run (common-test-bats-driver) @ hadoop-common ---
[INFO] Executing tasks

main:

[exec]
[exec]
[exec] ERROR: bats not installed. Skipping bash tests.
[exec] ERROR: Please install bats as soon as possible.
[exec]
[exec]

[INFO] Executed tasks
[INFO]
[INFO] --- jacoco-maven-plugin:0.8.6:report (report) @ hadoop-common ---
[INFO] Loading execution data file
/home/nvadde2/hadoop/hadoop-common-project/hadoop-common/target/coverage-reports/jaco
co-ut.exec
[INFO] Analyzed bundle 'Apache Hadoop Common' with 2160 classes
[INFO]
[INFO] --- jacoco-maven-plugin:0.8.6:report (default-cli) @ hadoop-common ---
[INFO] Skipping JaCoCo execution due to missing execution data file.
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 33.183 s
[INFO] Finished at: 2023-11-16T18:44:51-06:00
[INFO] -----