You:

Assume you are a tester. Write a unit test for Hadoop-common-project to test configuration parameter hadoop.security.instrumentation.requires.admin - HADOOP_SECURITY_INSTRUMENTATION_REQUIRES_ADMIN It is being used in below method public static boolean isInstrumentationAccessAllowed(ServletContext servletContext, HttpServletRequest request, HttpServletResponse response) throws IOException This method is in hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/http/HttpServer2.java class And the test should be in package org.apache.hadoop.http

---

ChatGPT:

Assuming you're using a testing framework like JUnit, here's a basic outline for a unit test in Java to test the `isInstrumentationAccessAllowed` method from the `HttpServer2.java` class, taking into consideration the configuration parameter `hadoop.security.instrumentation.requires.admin` (HADOOP_SECURITY_INSTRUMENTATION_REQUIRES_ADMIN).

Ensure that the necessary dependencies for JUnit are added to your Maven `pom.xml` or Gradle build file.

Unit Test Code

Below is an example of a unit test class. Please note that the actual implementation might depend on other code or configurations specific to your project.

```
java
package org.apache.hadoop.http;

import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;
import static org.mockito.Mockito.*;

import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.hadoop.conf.Configuration;
import org.junit.Before;
import org.junit.Test;

public class TestHttpServer2 {

    private ServletContext servletContext;
    private HttpServletRequest request;
    private HttpServletResponse response;
    private Configuration conf;

    @Before
    public void setup() {
        servletContext = mock(ServletContext.class);
        request = mock(HttpServletRequest.class);
        response = mock(HttpServletResponse.class);
        conf = new Configuration();
    }

    @Test
    public void testInstrumentationAccessAllowed() throws Exception {
        // Assuming HttpServer2 is modified to accept Configuration in its method or is otherwise accessible.
        HttpServer2 httpServer2 = new HttpServer2.Builder().setName("test")
                                    .addEndpoint(new URI("http://localhost")).build();

        // When HADOOP_SECURITY_INSTRUMENTATION_REQUIRES_ADMIN is true, access should be restricted.
        conf.setBoolean("hadoop.security.instrumentation.requires.admin", true);
        httpServer2.setAttribute("conf", conf);
        boolean accessAllowed = HttpServer2.isInstrumentationAccessAllowed(servletContext, request, response);
        assertFalse(accessAllowed);

        // When HADOOP_SECURITY_INSTRUMENTATION_REQUIRES_ADMIN is false, access should be allowed.
        conf.setBoolean("hadoop.security.instrumentation.requires.admin", false);
        httpServer2.setAttribute("conf", conf);
        accessAllowed = HttpServer2.isInstrumentationAccessAllowed(servletContext, request, response);
        assertTrue(accessAllowed);
    }
}
```

Note:

1. **Mocking:** In the example, the Mockito framework is used to create mock objects for `ServletContext`, `HttpServletRequest`, and `HttpServletResponse`. Ensure Mockito is included in your dependencies if not already.
2. **Configuration:** Ensure that the `HttpServer2` method `isInstrumentationAccessAllowed` has some way to access the Hadoop `Configuration` instance or refactor it in a way that you can inject/mock the configuration for testing purposes.
3. **HttpServer2 Instance:** You might have to adjust how the `HttpServer2` instance is created or configured based on how it's implemented in your project, as it's instantiated here just for the sake of the example.
4. **Assertion:** Ensure to add appropriate assertion checks as per your use case.
5. **Exception Handling:** Handle exceptions as per your project guidelines. This example throws an exception, but you might want to add more specific error handling in your test or tested method.

Adjustments might be needed based on the actual implementation and usage in your project. Ensure to adapt it according to your use case and existing codebase.

You:

The URI class is not defined

ChatGPT:

My apologies for the oversight. To use the `URI` class, you'll need to import it. The `URI` class belongs to the `java.net` package.

Add the following import statement to the top of your test class:

```
java
import java.net.URI;
```

With this import, the `URI` class should be recognized correctly in the code.

You:

Rewrite the full test suite using the above correction

ChatGPT:

Certainly! Below is the revised full test suite, including the necessary import statement for the `URI` class.

```
java
package org.apache.hadoop.http;

import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;
import static org.mockito.Mockito.*;

import java.net.URI;

import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.hadoop.conf.Configuration;
import org.junit.Before;
import org.junit.Test;

public class TestHttpServer2 {
```

```
    private ServletContext servletContext;
    private HttpServletRequest request;
    private HttpServletResponse response;
    private Configuration conf;

    @Before
    public void setup() {
        servletContext = mock(ServletContext.class);
        request = mock(HttpServletRequest.class);
        response = mock(HttpServletResponse.class);
        conf = new Configuration();
    }

    @Test
    public void testInstrumentationAccessAllowed() throws Exception {
        // Assuming HttpServer2 is modified to accept Configuration in its method or is otherwise accessible.
        HttpServer2 httpServer2 = new HttpServer2.Builder().setName("test")
                                    .addEndpoint(new URI("http://localhost")).build();

        // When HADOOP_SECURITY_INSTRUMENTATION_REQUIRES_ADMIN is true, access should be restricted.
        conf.setBoolean("hadoop.security.instrumentation.requires.admin", true);
        httpServer2.setAttribute("conf", conf);
        boolean accessAllowed = HttpServer2.isInstrumentationAccessAllowed(servletContext, request, response);
        assertFalse(accessAllowed);

        // When HADOOP_SECURITY_INSTRUMENTATION_REQUIRES_ADMIN is false, access should be allowed.
        conf.setBoolean("hadoop.security.instrumentation.requires.admin", false);
        httpServer2.setAttribute("conf", conf);
        accessAllowed = HttpServer2.isInstrumentationAccessAllowed(servletContext, request, response);
        assertTrue(accessAllowed);
    }
}
```

In this test suite:

- `URI` is imported to handle the URI used in the `HttpServer2.Builder().addEndpoint()` method.
- Two basic scenarios are tested: when `hadoop.security.instrumentation.requires.admin` is set to `true`, and when it is set to `false`.
- The test involves checking whether `isInstrumentationAccessAllowed` returns the expected boolean value under each scenario.

Make sure that your project has dependencies for JUnit and Mockito if you are going to run this test, and adjust the test according to any specifics or updates from your codebase. This example serves as a general guide and might require tweaks for your particular use case or setup.

You:

The Configuration is read from servlet COntext using hadoop.conf property

---

ChatGPT:

If the `Configuration` object is retrieved from the `ServletContext` using a `"hadoop.conf"` attribute, you should mock this behavior in your test setup to simulate the real-world scenario. The adjusted test suite below demonstrates this:

```
java
package org.apache.hadoop.http;

import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;
import static org.mockito.Mockito.*;

import java.net.URI;

import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.hadoop.conf.Configuration;
import org.junit.Before;
import org.junit.Test;
```

```
public class TestHttpServer2 {

    private ServletContext servletContext;
    private HttpServletRequest request;
    private HttpServletResponse response;
    private Configuration conf;

    @Before
    public void setup() {
        // Mock the servletContext, request, and response objects
        servletContext = mock(ServletContext.class);
        request = mock(HttpServletRequest.class);
        response = mock(HttpServletResponse.class);

        // Create a Configuration object and set an initial parameter
        conf = new Configuration();

        // When servletContext.getAttribute("hadoop.conf") is called, return the conf object
        when(servletContext.getAttribute("hadoop.conf")).thenReturn(conf);
    }

    @Test
    public void testInstrumentationAccessAllowed() throws Exception {
        // Assuming HttpServer2 is modified to accept Configuration in its method or is otherwise accessible.
        HttpServer2 httpServer2 = new HttpServer2.Builder().setName("test")
                                    .addEndpoint(new URI("http://localhost")).build();

        // When HADOOP_SECURITY_INSTRUMENTATION_REQUIRES_ADMIN is true, access should be restricted.
        conf.setBoolean("hadoop.security.instrumentation.requires.admin", true);
        boolean accessAllowed = HttpServer2.isInstrumentationAccessAllowed(servletContext, request, response);
        assertFalse(accessAllowed);

        // When HADOOP_SECURITY_INSTRUMENTATION_REQUIRES_ADMIN is false, access should be allowed.
        conf.setBoolean("hadoop.security.instrumentation.requires.admin", false);
        accessAllowed = HttpServer2.isInstrumentationAccessAllowed(servletContext, request, response);
        assertTrue(accessAllowed);
    }
}
```

In this adjusted test suite:

- `ServletContext` is mocked to return the `Configuration` object (`conf`) when the method `getAttribute("hadoop.conf")` is called.
- This `Configuration` object is then configured as needed for the different test scenarios.

Please note:

This example demonstrates the general idea and might need further adjustments based on your actual implementation, usage, and any possible updates in the codebase. Make sure that your project has dependencies for JUnit and Mockito if you're going to run this test.
