# 单元测试

162130117-傅锦龙

本次单元测试中，我选择了Apache commons math库中的Complex类进行测试。此处选取了Equals，Multiply，Divide方法来注入错误，并进行单元测试。

对于Equals方法，我选择了将if (c.isNaN)改为if (!c.isNaN)来注入错误。

public boolean equals(Object other) {  
 if (this == other) {  
 return true;  
 } else if (other instanceof Complex) {  
 Complex c = (Complex)other;  
 if (!c.isNaN) {//将if (c.isNaN)改为if (!c.isNaN)  
 return this.isNaN;  
 } else {  
 return MathUtils.*equals*(this.real, c.real) && MathUtils.*equals*(this.imaginary, c.imaginary);  
 }  
 } else {  
 return false;  
 }  
}

对于Multiply方法，我通过将实数部分和虚数部分的乘法进行交换来注入错误

public Complex multiply(Complex factor) throws NullArgumentException {  
 MathUtils.*checkNotNull*(factor);  
 if (!this.isNaN && !factor.isNaN) {  
 return !Double.*isInfinite*(this.real) && !Double.*isInfinite*(this.imaginary) && !Double.*isInfinite*(factor.real) && !Double.*isInfinite*(factor.imaginary) ? this.createComplex(this.real \* factor.imaginary + this.imaginary \* factor.real,this.real \* factor.real - this.imaginary \* factor.imaginary) : *INF*;// 将实数部分和虚数部分的乘法进行交换  
 } else {  
 return *NaN*;  
 }  
}

对于Divide方法，我通过将返回值NAN更换为INF来注入错误

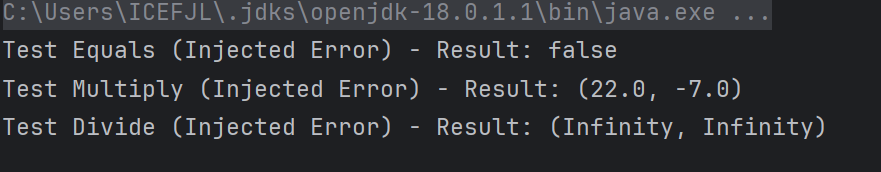
public Complex divide(Complex divisor) throws NullArgumentException {  
 MathUtils.*checkNotNull*(divisor);  
 if (!this.isNaN && !divisor.isNaN) {  
 double c = divisor.getReal();  
 double d = divisor.getImaginary();  
 if (c == 0.0 && d == 0.0) {  
 return *INF*;//原本为NAN  
 } else if (divisor.isInfinite() && !this.isInfinite()) {  
 return *ZERO*;  
 } else {  
 double q;  
 double denominator;  
 if (FastMath.*abs*(c) < FastMath.*abs*(d)) {  
 q = c / d;  
 denominator = c \* q + d;  
 return this.createComplex((this.real \* q + this.imaginary) / denominator, (this.imaginary \* q - this.real) / denominator);  
 } else {  
 q = d / c;  
 denominator = d \* q + c;  
 return this.createComplex((this.imaginary \* q + this.real) / denominator, (this.imaginary - this.real \* q) / denominator);  
 }  
 }  
 } else {  
 return *NaN*;  
 }  
}

## 不使用Junit工具的单元测试

具体代码如下

package org.apache.commons.math3.complex;  
  
public class TestComplex {  
 public static void main(String[] args) {  
 *testEquals*();  
 *testMultiply*();  
 *testDivide*();  
 }  
 // 测试比较相等方法，注入错误的版本  
 public static void testEquals(){  
 Complex complex1 = new Complex(1.0, 2.0);  
 Complex complex2 = new Complex(1.0, 2.0);  
  
 boolean result = complex1.equals(complex2);  
 // 期望结果是错误的  
 System.*out*.println("Test Equals (Injected Error) - Result: " + result);  
 }  
  
 // 测试乘法方法，注入错误的版本  
 public static void testMultiply() {  
 Complex complex1 = new Complex(2.0, 3.0);  
 Complex complex2 = new Complex(4.0, 5.0);  
  
 Complex result = complex1.multiply(complex2);  
  
 // 期望结果是错误的  
 System.*out*.println("Test Multiply (Injected Error) - Result: " + result);  
 }  
  
 // 测试除法方法，注入错误的版本  
 public static void testDivide() {  
 Complex complex1 = new Complex(1.0, 2.0);  
 Complex complex2 = new Complex(0.0, 0.0);  
  
 Complex result = complex1.divide(complex2);  
  
 // 期望结果是错误的  
 System.*out*.println("Test Divide (Injected Error) - Result: " + result);  
 }  
}

测试结果如下

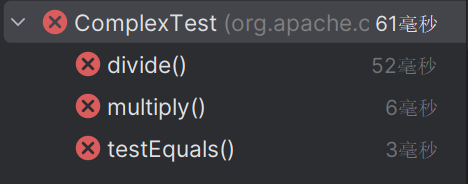


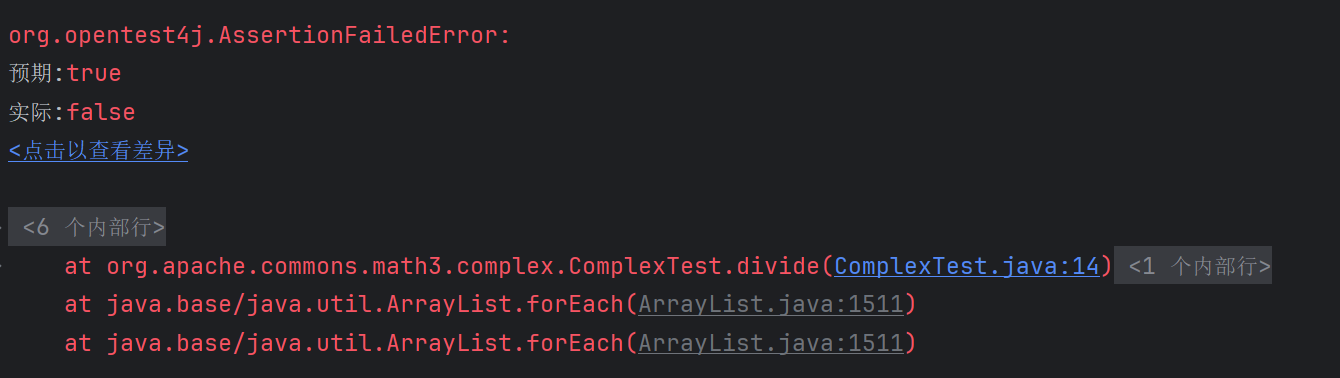
## 使用Junit实施单元测试

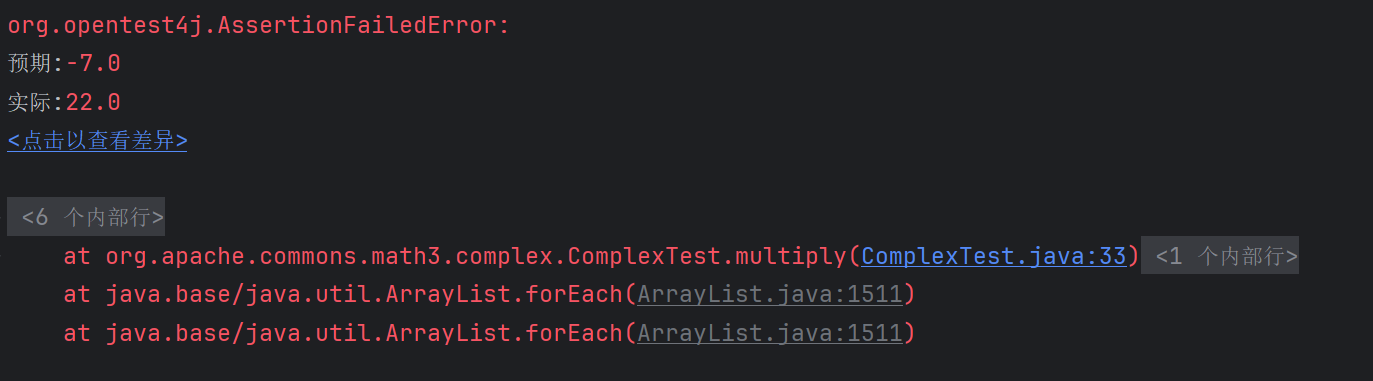
具体代码如下

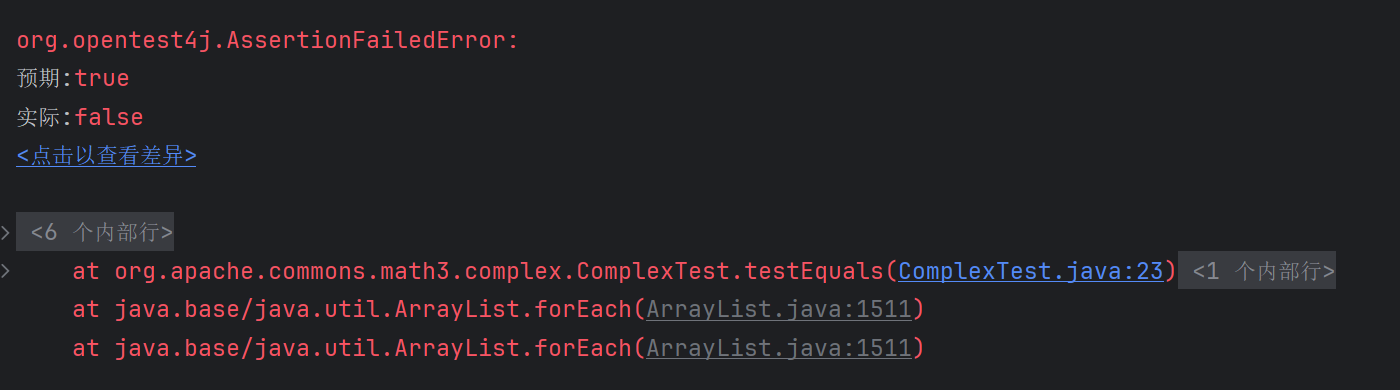
package org.apache.commons.math3.complex;  
  
import static org.junit.jupiter.api.Assertions.\*;  
import org.junit.Test;  
  
public class ComplexTest {  
  
 @Test  
 public void divide() {  
 Complex complex1 = new Complex(1.0, 2.0);  
 Complex complex2 = new Complex(0.0, 0.0);  
  
 Complex result = complex1.divide(complex2);  
  
 *assertTrue*(result.isNaN());  
 }  
  
 @Test  
 public void testEquals() {  
 Complex complex1 = new Complex(1.0, 2.0);  
 Complex complex2 = new Complex(1.0, 2.0);  
  
 boolean result = complex1.equals(complex2);  
 *assertTrue*(complex1.equals(complex2));  
 }  
  
 @Test  
 public void multiply() {  
 Complex complex1 = new Complex(2.0, 3.0);  
 Complex complex2 = new Complex(4.0, 5.0);  
  
 Complex result = complex1.multiply(complex2);  
  
 *assertEquals*(-7.0, result.getReal());  
 *assertEquals*(22.0, result.getImaginary());  
 }  
}

测试结果如下







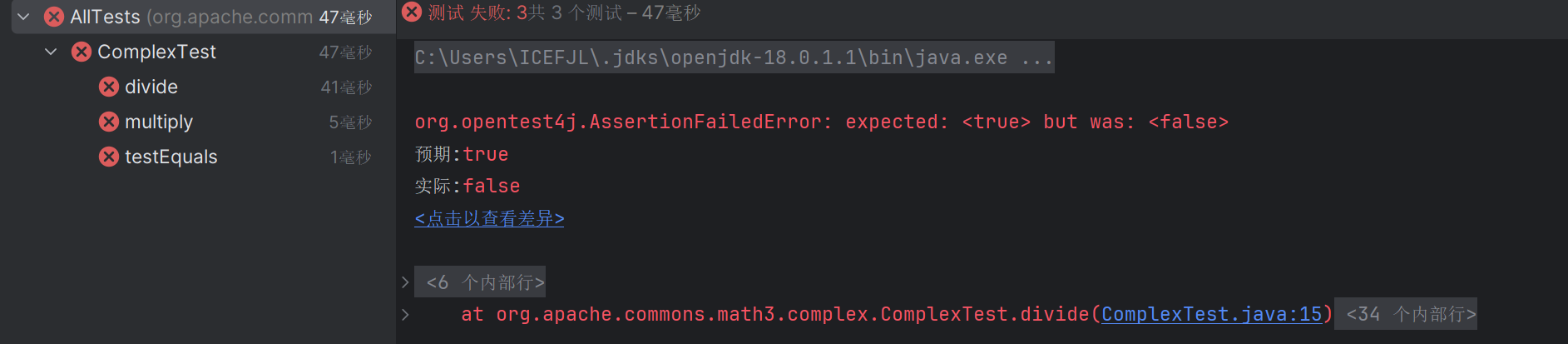


## 测试集测试

具体代码如下

package org.apache.commons.math3.complex;  
  
import org.junit.runner.RunWith;  
import org.junit.runners.Suite;  
  
@RunWith(Suite.class)  
@Suite.SuiteClasses({ ComplexTest.class })  
public class AllTests {  
}

测试结果如下



## 使用和不使用Junit框架的比较

**使用 JUnit 框架：**

1. **易于组织和执行测试：** JUnit 提供了注解来标记测试方法，可以轻松组织测试套件，并使用 JUnit Runner 来执行测试。
2. **断言和验证：** JUnit 提供了一组内置的断言方法，使得测试结果的验证变得容易。这些方法包括 **assertEquals**, **assertTrue**, **assertFalse** 等。
3. **测试报告：** JUnit 生成详细的测试报告，告诉你哪些测试通过，哪些失败，以及失败的原因。这对于快速识别和修复问题非常有帮助。
4. **运行多个测试用例：** JUnit 支持一次运行多个测试用例，可以轻松地执行整个测试套件。
5. **生命周期管理：** JUnit 支持 **@Before** 和 **@After** 注解，用于在测试方法执行前后进行设置和清理操作。

**不使用 JUnit 框架：**

1. **手动编写测试代码：** 如果不使用 JUnit，你需要自己编写测试代码，包括测试方法的组织、断言和结果验证。这会导致更多的手动劳动和潜在的错误。
2. **难以组织和执行测试：** 你需要自己编写测试套件和测试执行代码，这会增加复杂性和维护成本。
3. **没有内置的断言：** 没有 JUnit 提供的方便的断言方法，你需要自己编写断言逻辑，这可能会导致重复的测试代码。
4. **缺乏自动报告：** 你需要手动收集和报告测试结果，这会导致测试结果的可读性下降。
5. **生命周期管理：** 没有 JUnit 的生命周期管理功能，需要手动编写设置和清理代码。

总的来说，使用 JUnit 框架能够显著简化单元测试的编写和执行，提供了丰富的功能和工具，有助于提高测试的可维护性和可读性。所以一般建议基于JUnit等工具开展单元测试，而不是自己编写测试代码。