**《软件测试》**

**实验报告一 ——JUnit测试方法**

**姓 名： 胡国昌 学 号： 202210120518**

**院 系： 计算机与信息学院 专 业：计算机与科学**

**实 验 室： B1-119 实验日期： 2025.2.22**

**总评成绩： 审阅教师： 杨青**

# 一、实验目的：

1. 掌握 git 协作式管理工具；
2. 掌握 gradle 或 Maven 项目构建工具基本操作
3. 淬炼基本代码编程能力
4. 掌握单元测试框架Junit基本操作

# 二、实验环境：

IDEA Goland

# 三、实验要求：

1. 学习使用 JUnit5 编写测试用例，参考[Junit5 三分钟教程](https://star.jmhui.com.cn/p1/795.html" \o "Junit5 三分钟教程)
2. 使用 git 管理代码（gitee或GitHub均可）
3. 使用 gradle 项目构建工具，参考 [gradle 入门教程](https://www.cnblogs.com/hellxz/p/helloworld-gradle.html" \o "gradle 入门教程)，深入学习可以参考B站学习视频[来自Gradle开发团队的Gradle入门教程](https://www.bilibili.com/video/BV1DE411Z7nt?p=1" \o "来自Gradle开发团队的Gradle入门教程);或者使用maven构建工具创建项目，参考[Maven使用入门指南](https://star.jmhui.com.cn/p1/1054.html" \o "Maven使用入门指南)；
4. 按照以下实验要求，使用 Java 语言实现判断三角形类型程序Triangle，并采用Junit5 编写完成三角形类型程序测试用例，并提交到代码仓库
5. 按照[实验报告模板](https://star.jmhui.com.cn/u/cms/www/202203/06160105hi3m.docx) 编写实验报告，以“学号-姓名-软件测试实验一”命名，提交到雨课堂“软件测试实验一”

# 四、实验步骤与内容

## 题目：三角形题目

**输入**3个整数a、b和c分别作为三角形的三条边，要求a、b和c必须满足以下条件：

1、整数

2、3个数

3、边长大于等于1，小于等于100

4、任意两边之和大于第三边

**输出**为5种情况之一：

如果不满足条件1、2、3，则程序输出为“输入错误”。

如果不满足条件4，则程序输出为“非三角形”。

如果三条边相等，则程序输出为“等边三角形”。

如果恰好有两条边相等，则程序输出为“等腰三角形”。

如果三条边都不相等，则程序输出为“一般三角形”。

## 代码实现：

### 2.1Java部分

**TriangleChecker.java**

package com.ctgu.api;

public class TriangleChecker {

// 解析并验证输入参数

public static int[] parseAndValidate(String[] args) throws IllegalArgumentException {

if (args.length != 3) {

throw new IllegalArgumentException("参数不足");

}

try {

int a = Integer.parseInt(args[0]);

int b = Integer.parseInt(args[1]);

int c = Integer.parseInt(args[2]);

if (a < 1 || a > 100 || b < 1 || b > 100 || c < 1 || c > 100) {

throw new IllegalArgumentException("参数超出范围时抛出异常");

}

return new int[] {

a,

b,

c

};

} catch (NumberFormatException e) {

throw new IllegalArgumentException("非整数参数时抛出异常");

}

}

// 判断是否为三角形

public static boolean isTriangle(int a, int b, int c) {

return a + b > c && a + c > b && b + c > a;

}

// 判断三角形类型

public static String determineTriangleType(int a, int b, int c) {

if (a == b && b == c) {

return "等边三角形";

} else if (a == b || a == c || b == c) {

return "等腰三角形";

} else {

return "一般三角形";

}

}

}

**TriangleCheckerTest.java**

package com.ctgu.api;

import org.junit.jupiter.api.DisplayName;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.CsvSource;

import org.junit.jupiter.params.provider.ValueSource;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import static org.junit.jupiter.api.Assertions.\*;

public class TriangleCheckerTest {

private static final Logger logger = LoggerFactory.getLogger(TriangleCheckerTest.class);

// 参数不足时抛出异常

@

ParameterizedTest@ DisplayName("参数不足时抛出异常")@ ValueSource(strings = {

"1", "1,2", "1,2,3,4"

})

public void testParseAndValidate\_InsufficientParameters(String input) {

String[] args = input.split(",");

IllegalArgumentException exception = assertThrows(

IllegalArgumentException.class, () - > TriangleChecker.parseAndValidate(args)

);

logger.info("InsufficientParameters: {} Catch exception: {}", args, exception.getMessage());

assertEquals("参数不足", exception.getMessage());

}

// 非整数参数时抛出异常

@

ParameterizedTest@ DisplayName("非整数参数时抛出异常")@ CsvSource({

"a, 2, 3",

"1, b, 3",

"1, 2, c"

})

public void testParseAndValidate\_NonIntegerParameters(String a, String b, String c) {

IllegalArgumentException exception = assertThrows(

IllegalArgumentException.class, () - > TriangleChecker.parseAndValidate(new String[] {

a, b, c

})

);

logger.info("NonIntegerParameters :{} Catch exception: {}", new String[] {

a, b, c

}, exception.getMessage());

assertEquals("非整数参数时抛出异常", exception.getMessage());

}

// 参数超出范围时抛出异常

@

ParameterizedTest@ DisplayName("参数超出范围时抛出异常")@ CsvSource({

"0, 10, 20",

"10, 0, 30",

"10, 20, 0",

"101, 20, 30",

"10, 200, 101",

"10, 20, 101"

})

public void testParseAndValidate\_OutOfRangeParameters(String a, String b, String c) {

IllegalArgumentException exception = assertThrows(

IllegalArgumentException.class, () - > TriangleChecker.parseAndValidate(new String[] {

a, b, c

})

);

logger.info(" OutOfRangeParameters :{} Catch exception: {}", new String[] {

a, b, c

}, exception.getMessage());

assertEquals("参数超出范围时抛出异常", exception.getMessage());

}

@

Test@ DisplayName("合法参数解析正确")

public void testParseAndValidate\_ValidParameters() {

int[] expected = {

3, 4, 5

};

logger.info("testParseAndValidate\_ValidParameters : {}", TriangleChecker.parseAndValidate(new String[] {

"3", "4", "5"

}));

assertArrayEquals(expected, TriangleChecker.parseAndValidate(new String[] {

"3", "4", "5"

}));

}

@

ParameterizedTest@ DisplayName("非三角形判断")@ CsvSource({

"1, 2, 3",

"1, 1, 2",

"10, 1, 1"

})

public void testIsTriangle\_NonTriangle(int a, int b, int c) {

logger.info("NonTriangle :{} testIsTriangle\_NonTriangle: {}", new int[] {

a, b, c

}, TriangleChecker.isTriangle(a, b, c));

assertFalse(TriangleChecker.isTriangle(a, b, c));

}

@

ParameterizedTest@ DisplayName("合法三角形判断")@ CsvSource({

"3, 4, 5",

"5, 12, 13",

"7, 24, 25"

})

public void testIsTriangle\_ValidTriangle(int a, int b, int c) {

logger.info("ValidTriangle: {}testIsTriangle\_ValidTriangle: {}", new int[] {

a, b, c

}, TriangleChecker.isTriangle(a, b, c));

assertTrue(TriangleChecker.isTriangle(a, b, c));

}

// 等边三角形判断

@

ParameterizedTest@ DisplayName("等边三角形判断")@ ValueSource(ints = {

3, 5, 7

})

public void testDetermineTriangleType\_Equilateral(int side) {

logger.info("Equilateral: {}testDetermineTriangleType\_Equilateral: {}", new int[] {

side, side, side

}, TriangleChecker.determineTriangleType(side, side, side));

assertEquals("等边三角形", TriangleChecker.determineTriangleType(side, side, side));

}

@

ParameterizedTest@ DisplayName("等腰三角形判断")@ CsvSource({

"3, 3, 4",

"5, 5, 8",

"7, 7, 10"

})

public void testDetermineTriangleType\_Isosceles(int a, int b, int c) {

logger.info("Isosceles: {} testDetermineTriangleType\_Isosceles: {}", new int[] {

a, b, c

}, TriangleChecker.determineTriangleType(a, b, c));

assertEquals("等腰三角形", TriangleChecker.determineTriangleType(a, b, c));

}

// Test for scalene triangle

@

ParameterizedTest@ DisplayName("一般三角形判断")@ CsvSource({

"3, 4, 5",

"5, 12, 13",

"7, 24, 25"

})

public void testDetermineTriangleType\_Scalene(int a, int b, int c) {

logger.info("Scalene: {} testDetermineTriangleType\_Scalene: {}", new int[] {

a, b, c

}, TriangleChecker.determineTriangleType(a, b, c));

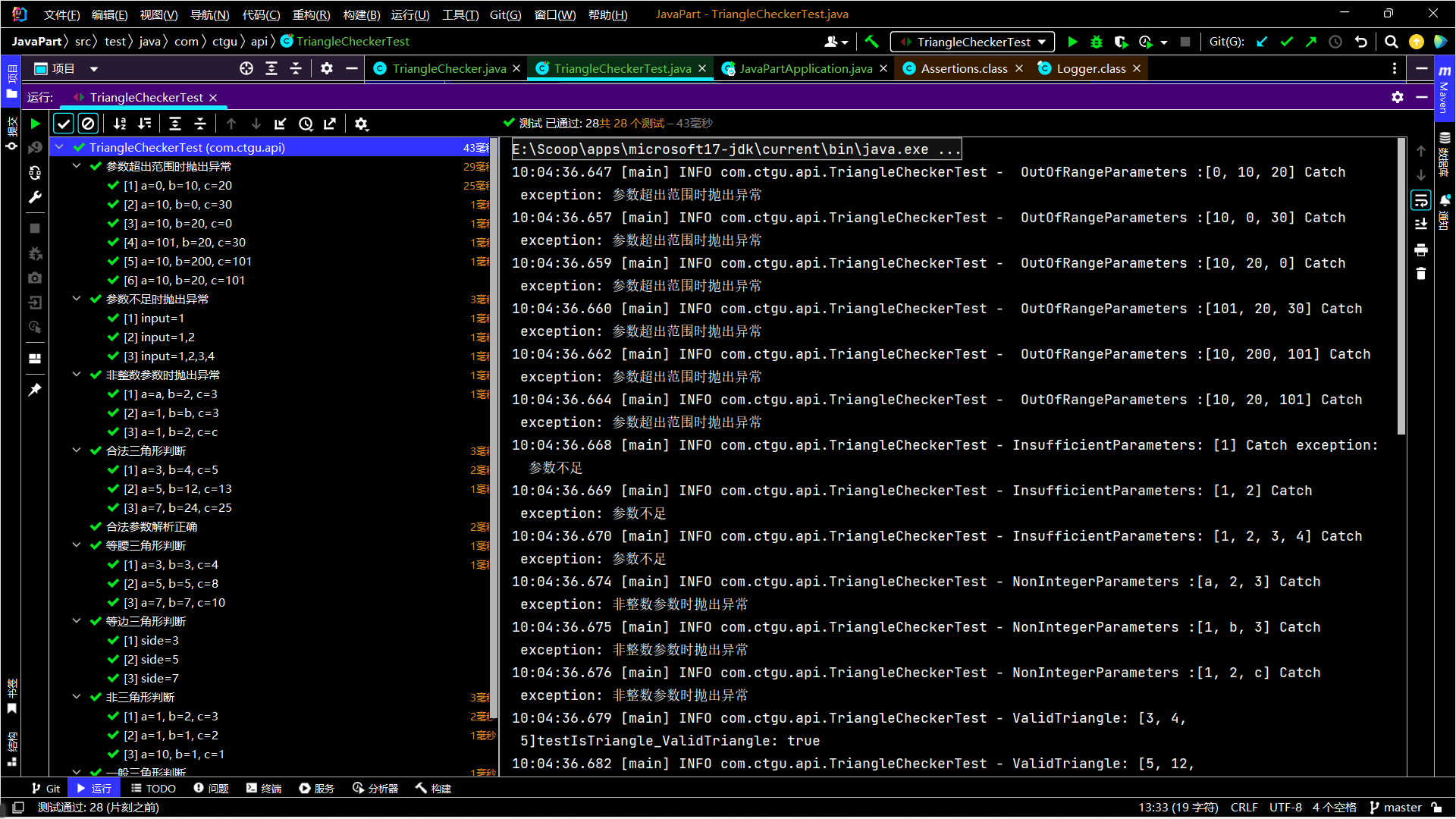
assertEquals("一般三角形", TriangleChecker.determineTriangleType(a, b, c));

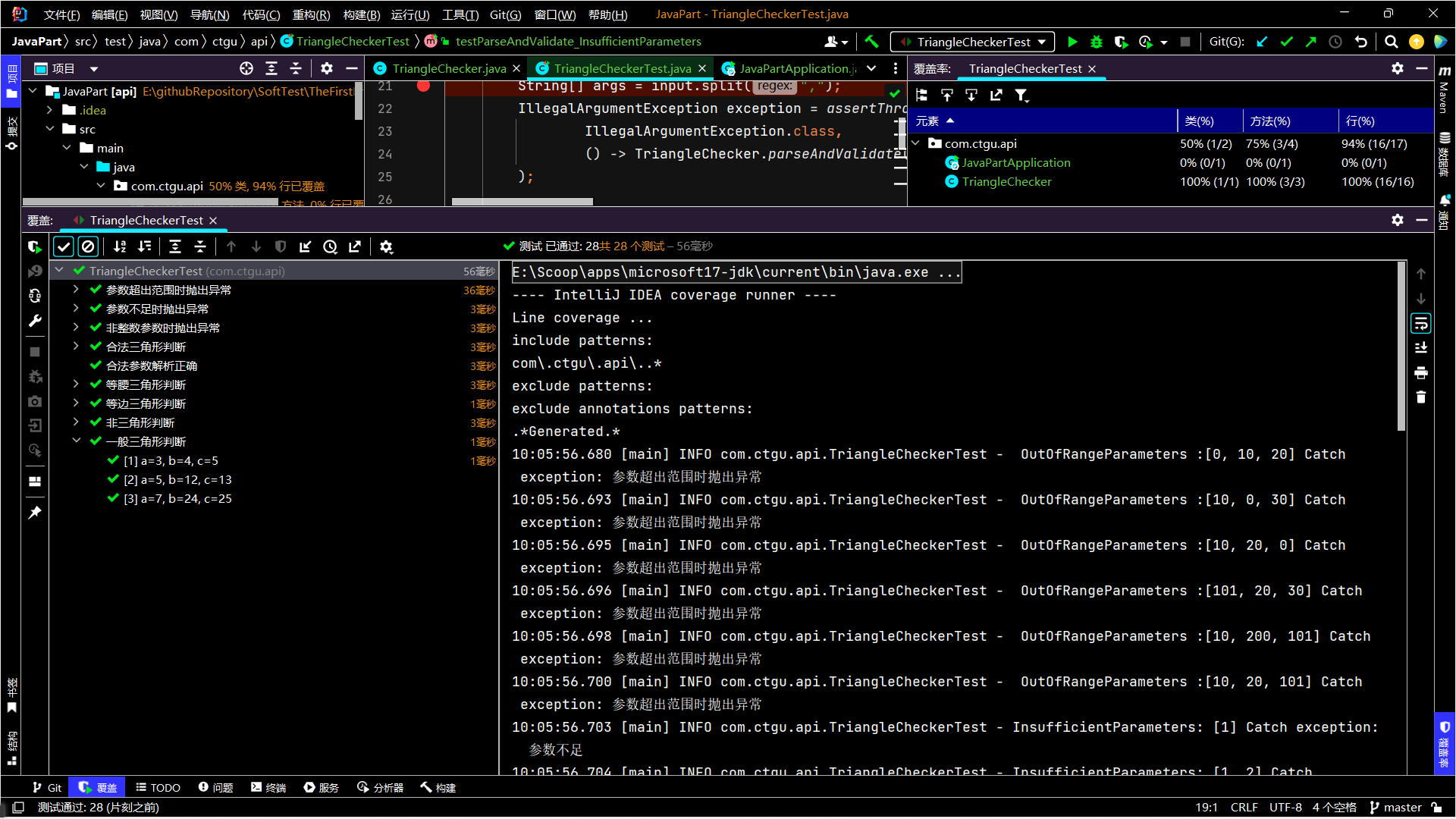
}

}

#### Idea工具测试用例和覆盖率结果

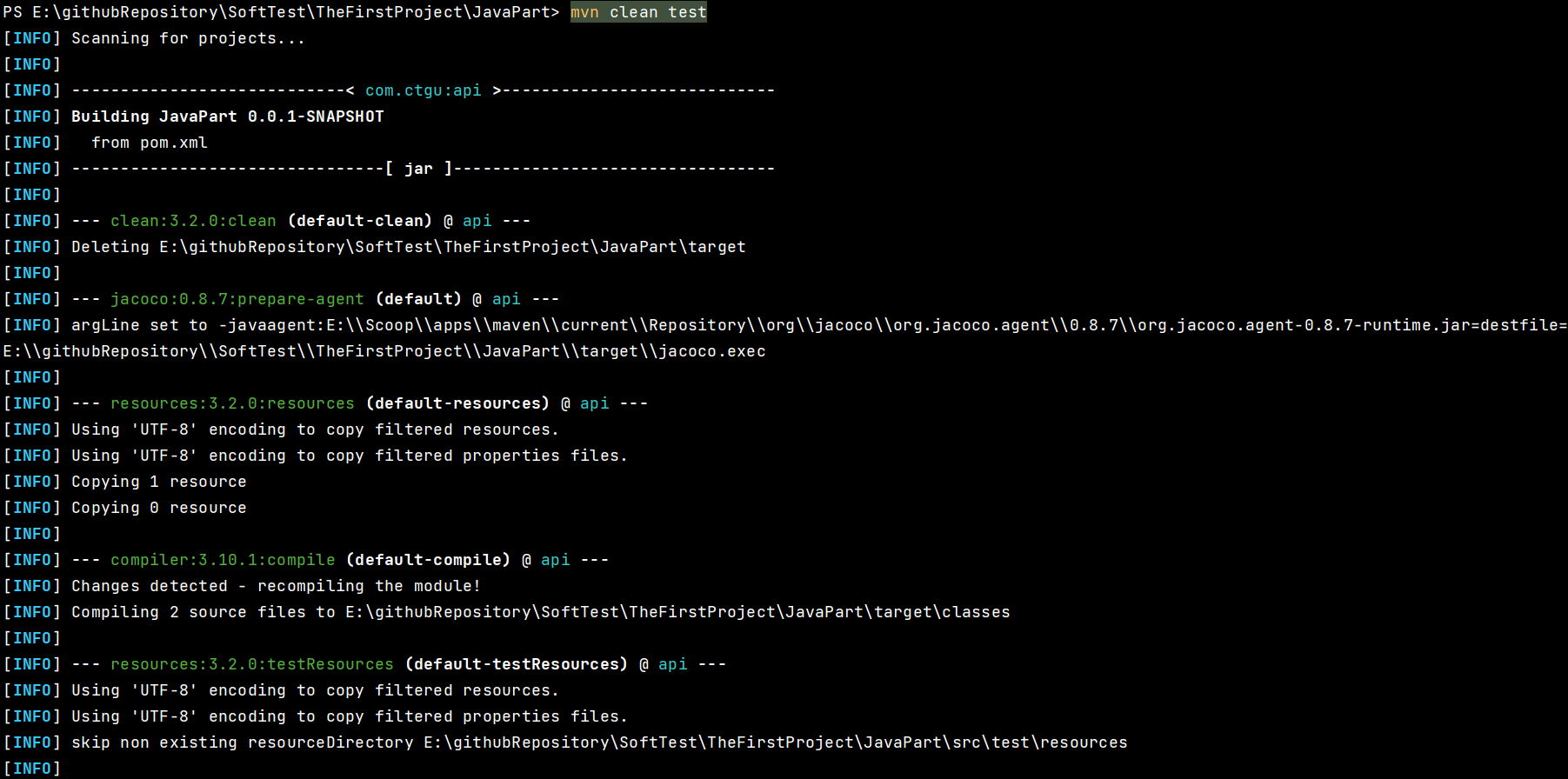
**我使用idea工具测试用例 每个用例都通过了 测试覆盖率 在目标代码达到100%的覆盖**

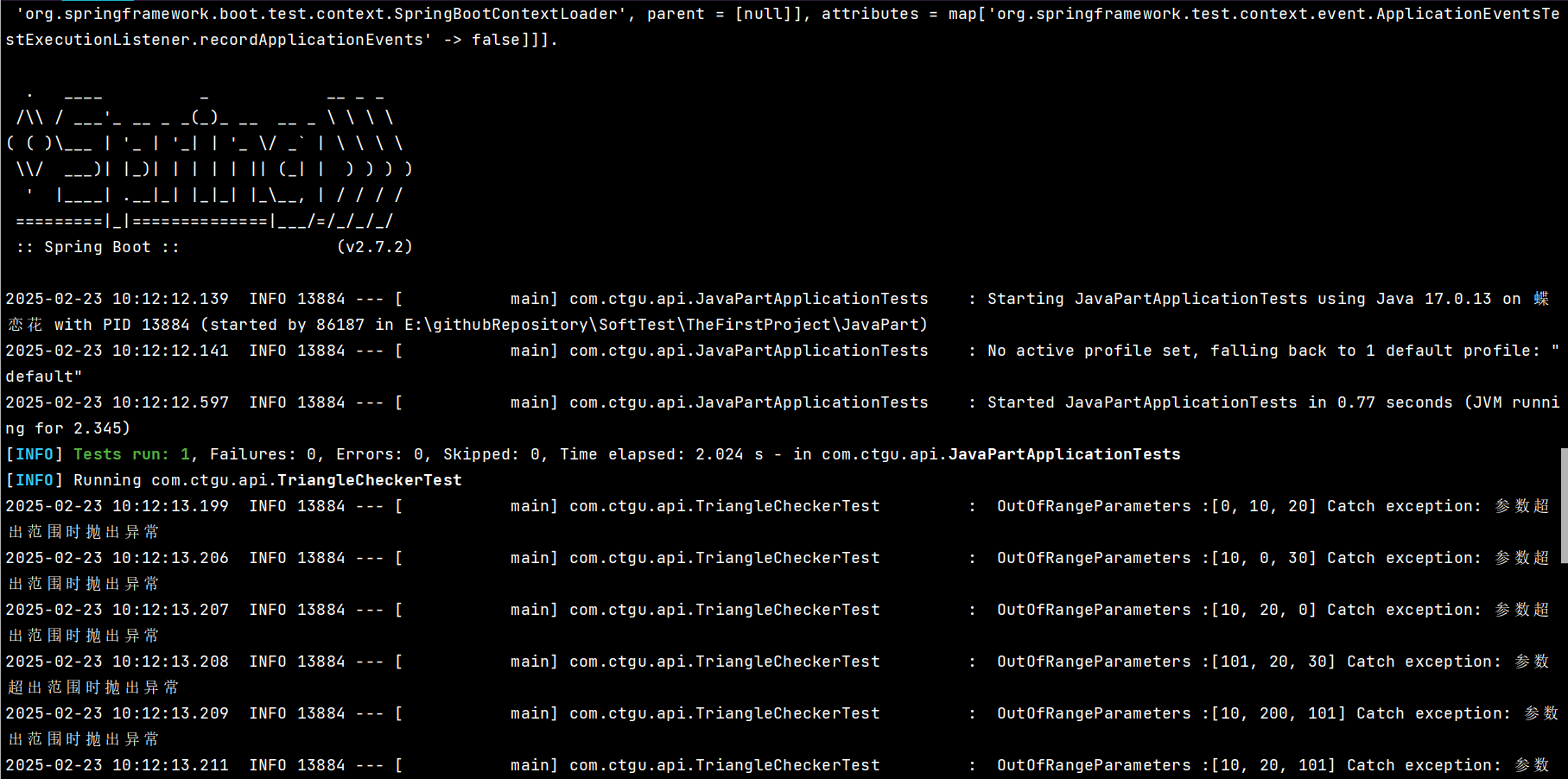


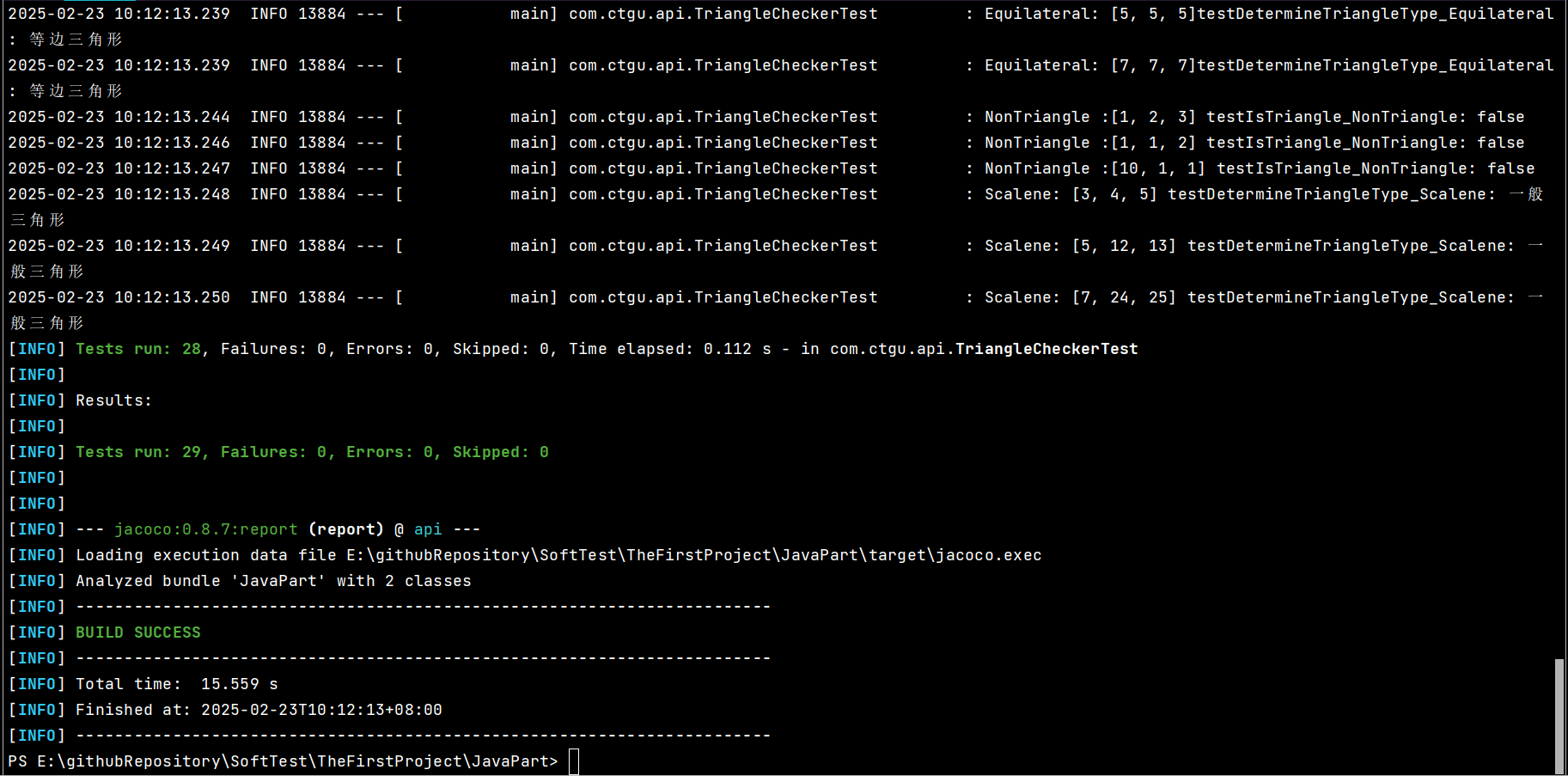


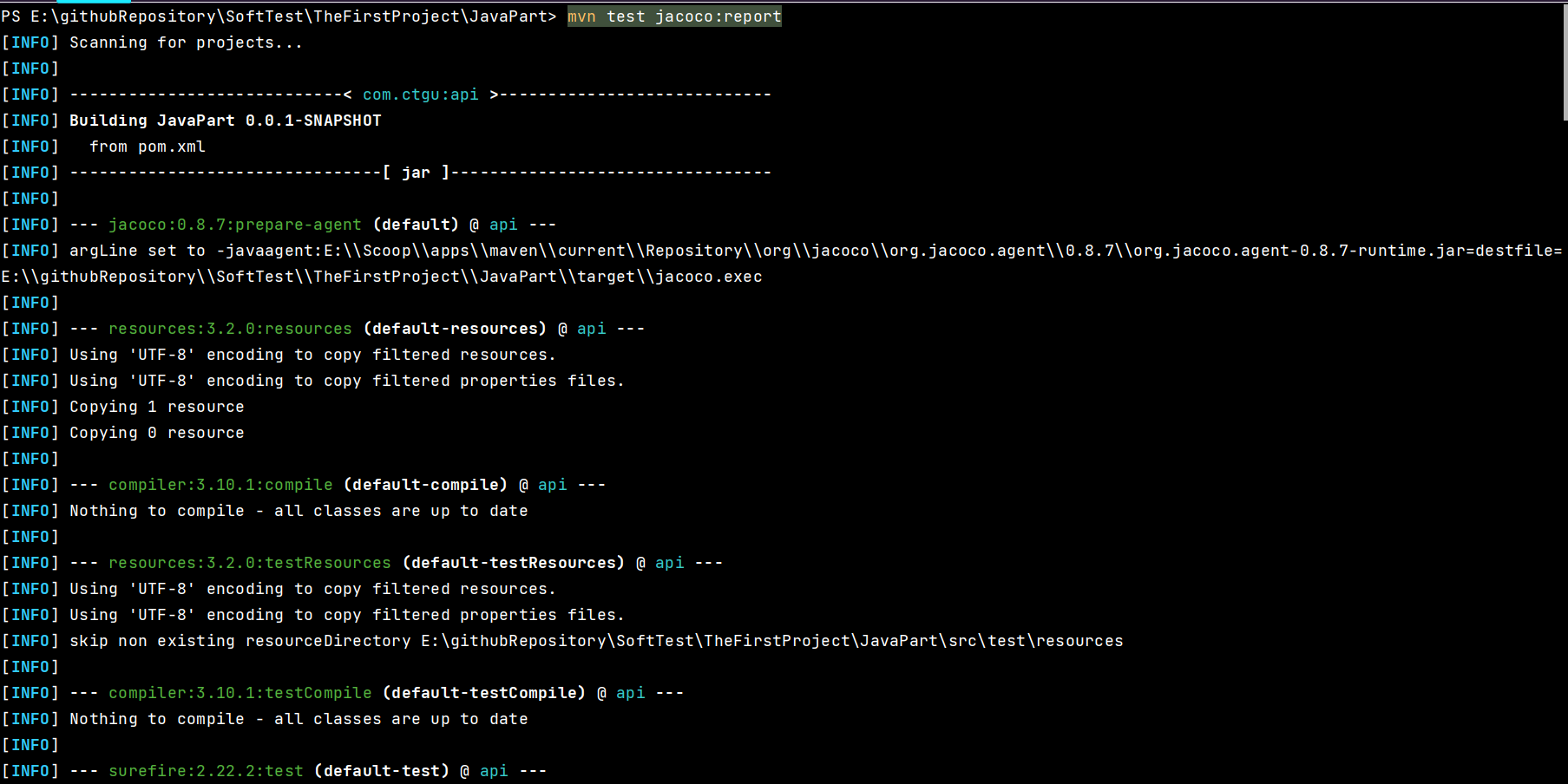
#### 使用命令行测试用例和覆盖率结果

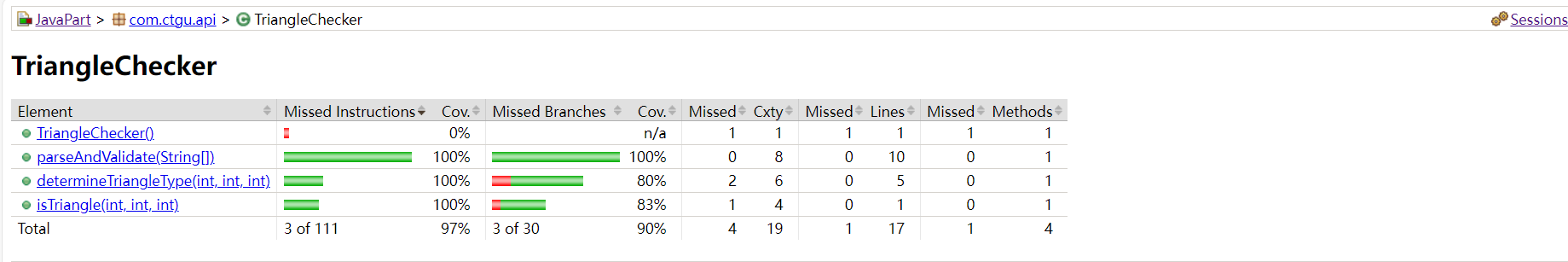
**使用mvn clean test命令显示测试用例都通过了 之后使用mvn test jacoco:report生成测试报告 显示测试覆盖100%**











### golang部分

package main

import (

"errors"

"strconv"

)

// parseAndValidate 解析并验证输入参数

func parseAndValidate(args[] string)(int, int, int, error) {

if len(args) != 3 {

return 0, 0, 0, errors.New("参数不足")

}

a, err1: = strconv.Atoi(args[0])

b, err2: = strconv.Atoi(args[1])

c, err3: = strconv.Atoi(args[2])

if err1 != nil || err2 != nil || err3 != nil {

return 0, 0, 0, errors.New("参数超出范围时抛出异常")

}

if a < 1 || a > 100 || b < 1 || b > 100 || c < 1 || c > 100 {

return 0, 0, 0, errors.New("非整数参数时抛出异常")

}

return a, b, c, nil

}

// isTriangle 判断是否为三角形

func isTriangle(a, b, c int) bool {

return a + b > c && a + c > b && b + c > a

}

// determineTriangleType 判断三角形类型

func determineTriangleType(a, b, c int) string {

if a == b && b == c {

return "等边三角形"

} else if a == b || a == c || b == c {

return "等腰三角形"

} else {

return "一般三角形"

}

}

package main

import (

"log"

"testing"

)

// TestParseAndValidate 测试 parseAndValidate 函数

func TestParseAndValidate(t \* testing.T) {

tests: = [] struct {

name string

args[] string

wantErr bool

} {

{

"Insufficient Parameters", [] string {

"1", "2"

}, true

}, {

"Non-Integer Parameters", [] string {

"a", "2", "3"

}, true

}, {

"Parameters Out of Range", [] string {

"0", "101", "50"

}, true

}, {

"Valid Parameters", [] string {

"3", "4", "5"

}, false

},

}

for \_,

tt: = range tests {

t.Run(tt.name, func(t \* testing.T) {

\_, \_, \_, err: = parseAndValidate(tt.args)

if (err != nil) != tt.wantErr {

log.Printf("Test failed: %s, args: %v, error: %v, wantErr: %v\n", tt.name, tt.args, err, tt.wantErr)

t.Errorf("parseAndValidate(%v) error = %v, wantErr %v", tt.args, err, tt.wantErr)

} else {

log.Printf("Test passed: %s, args: %v\n", tt.name, tt.args)

}

})

}

}

// TestIsTriangle 测试 isTriangle 函数

func TestIsTriangle(t \* testing.T) {

tests: = [] struct {

name string

a, b, c int

want bool

} {

{

"Non-Triangle", 1, 2, 3, false

}, {

"Valid Triangle", 3, 4, 5, true

},

}

for \_,

tt: = range tests {

t.Run(tt.name, func(t \* testing.T) {

if got: = isTriangle(tt.a, tt.b, tt.c);

got != tt.want {

log.Printf("Test failed: %s, inputs: %d, %d, %d, want: %v\n", tt.name, tt.a, tt.b, tt.c, tt.want)

t.Errorf("isTriangle(%d, %d, %d) = %v, want %v", tt.a, tt.b, tt.c, got, tt.want)

} else {

log.Printf("Test passed: %s, inputs: %d, %d, %d\n", tt.name, tt.a, tt.b, tt.c)

}

})

}

}

// TestDetermineTriangleType 测试 determineTriangleType 函数

func TestDetermineTriangleType(t \* testing.T) {

tests: = [] struct {

name string

a, b, c int

want string

} {

{

"Equilateral Triangle", 3, 3, 3, "等边三角形"

}, {

"Isosceles Triangle", 3, 3, 4, "等腰三角形"

}, {

"Scalene Triangle", 3, 4, 5, "一般三角形"

},

}

for \_,

tt: = range tests {

t.Run(tt.name, func(t \* testing.T) {

if got: = determineTriangleType(tt.a, tt.b, tt.c);

got != tt.want {

log.Printf("Test failed: %s, inputs: %d, %d, %d, want: %s\n", tt.name, tt.a, tt.b, tt.c, tt.want)

t.Errorf("determineTriangleType(%d, %d, %d) = %v, want %v", tt.a, tt.b, tt.c, got, tt.want)

} else {

log.Printf("Test passed: %s, inputs: %d, %d, %d\n", tt.name, tt.a, tt.b, tt.c)

}

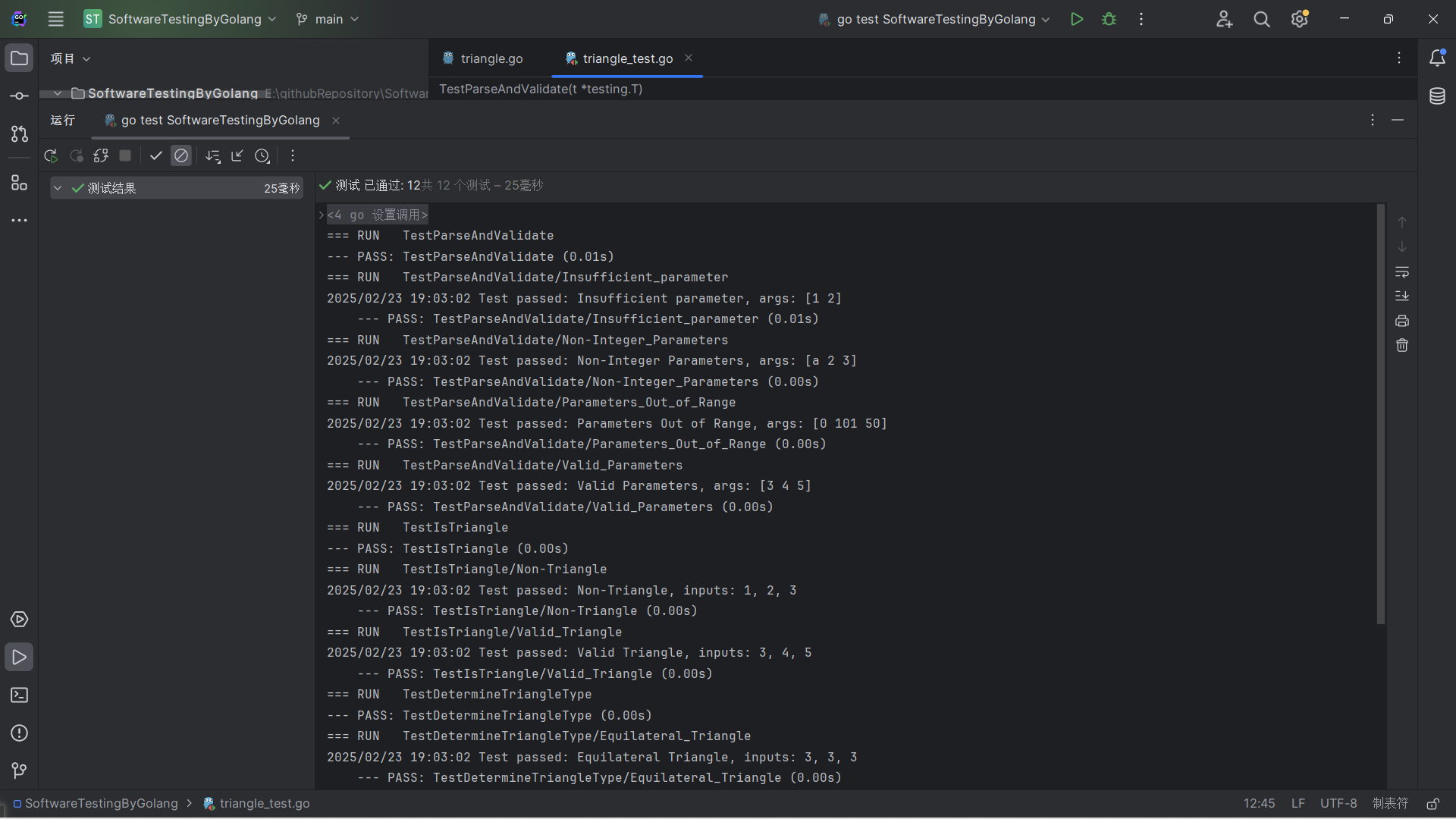
})

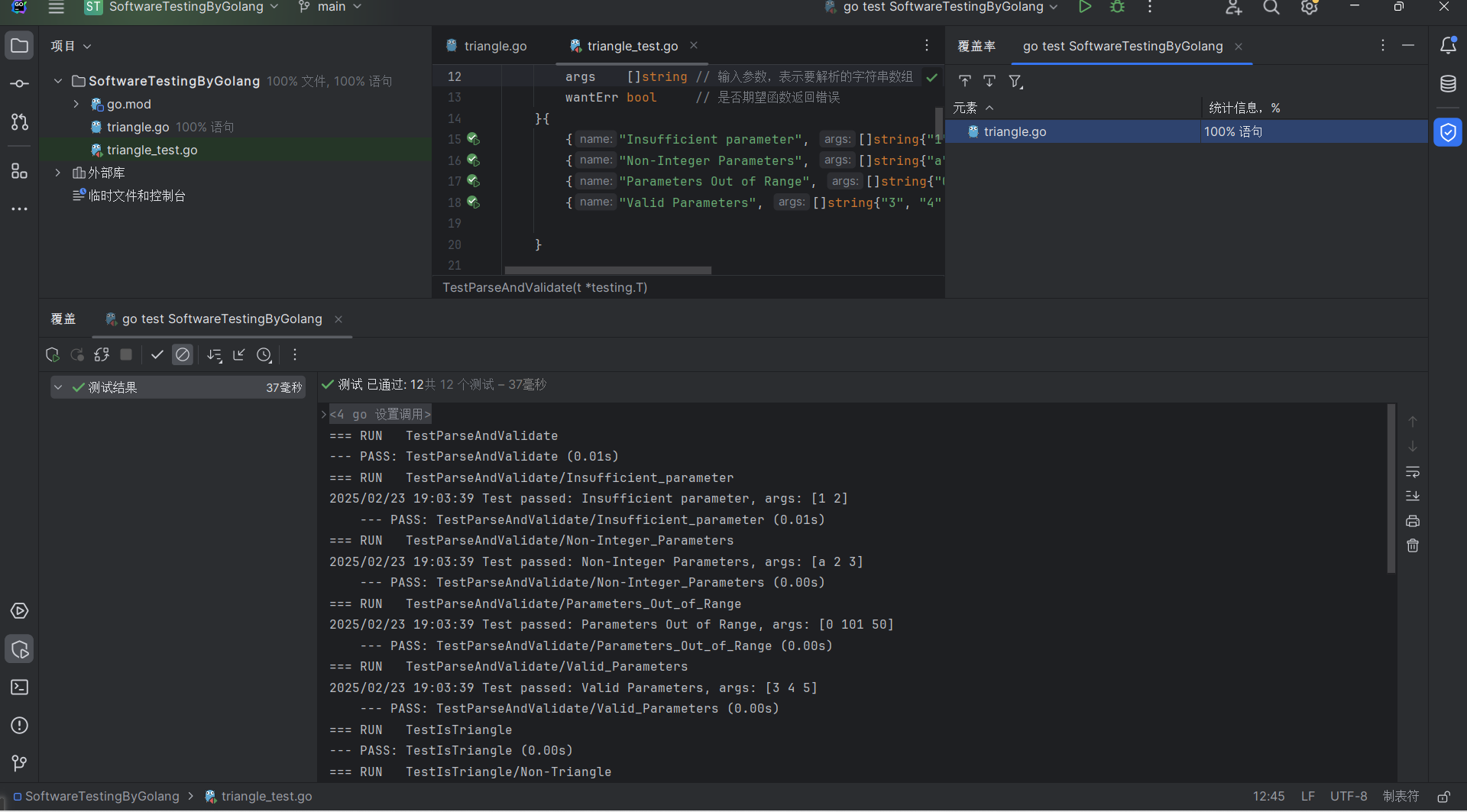
}

}

#### Goland工具测试用例和覆盖率结果

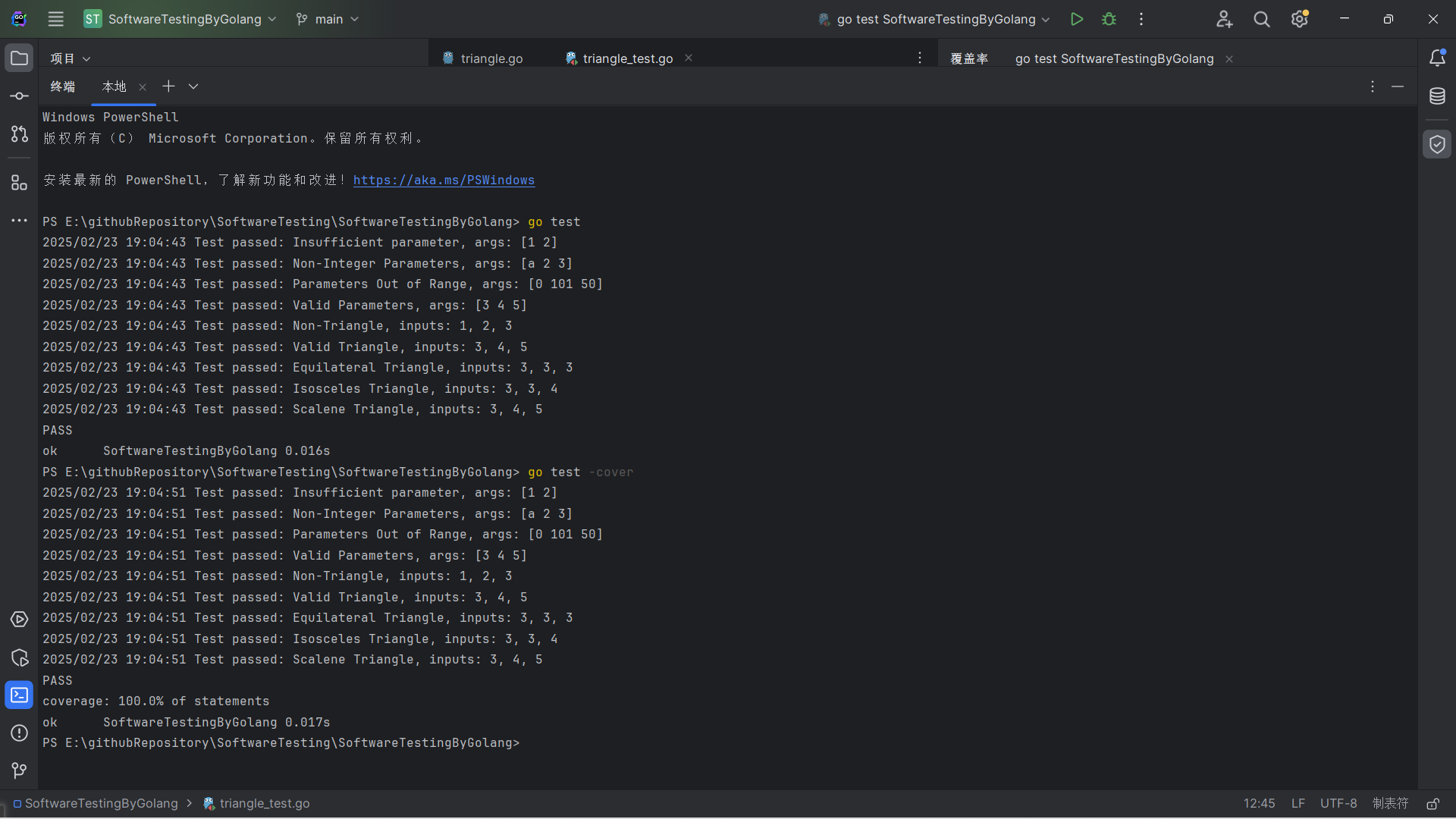
**我使用idea工具测试用例 每个用例都通过了 测试覆盖率 在目标代码达到100%的覆盖**





#### 使用命令行测试用例和覆盖率结果

**使用go test命令显示测试用例都通过了 之后使用go test -cover生成测试报告 显示测试覆盖100%**



# 五、结论分析与体会

在测试代码中集成日志记录功能，能够帮助开发者更好地跟踪测试执行过程，尤其是在测试失败时，日志可以提供详细的上下文信息，便于快速定位问题。我们使用了 Go 标准库 log 来实现日志记录，并在测试通过或失败时分别记录了相关信息。

测试与报告的依赖关系：无论是 Go 还是 Java，生成报告的前提都是先运行测试。例如，单独执行 mvn jacoco:report 会失败，必须通过 mvn test jacoco:report 确保测试数据已生成。

Go 语言通过内置的 go test 工具实现了开箱即用的测试能力，配合 -cover 和 -json 参数可以快速生成覆盖率报告和结构化测试结果。

Java 的生态复杂性：Java 依赖 Maven/Gradle 等构建工具，通过插件（如 Surefire、Jacoco）实现测试和覆盖率统计。

# 仓库地址

https://github.com/xieyangyuyue/SoftwareTesting

[xieyangyuyue/SoftwareTesting: 软件测试--java (github.com)](https://github.com/xieyangyuyue/SoftwareTesting)