

# TDD 测试驱动开发实验

刘铭宸

软件工程 2003 班

U202010783

2022 年 6 月 8 日

## 目录

<b>1</b>	<b>关于测试驱动开发</b>	<b>2</b>
1.1	何谓测试驱动开发? . . . . .	2
1.2	为何要使用 TDD? . . . . .	2
1.3	如何进行测试驱动开发? . . . . .	2
<b>2</b>	<b>实验内容——判断字符串是否是 IPV4 地址</b>	<b>3</b>
2.1	判断字符串是否为空串 . . . . .	3
2.2	判断字符串是否可被分成四段 . . . . .	4
2.3	判断每一段是否是在 0 到 255 之间的数字 . . . . .	6
2.4	判断是否存在以 0 开头的非零数字 . . . . .	8
2.5	判断字符串是否同时满足上述条件 . . . . .	10
<b>3</b>	<b>代码展示</b>	<b>12</b>
3.1	程序代码 . . . . .	12
3.2	测试代码 . . . . .	13
<b>4</b>	<b>参考资料</b>	<b>16</b>

# 1 关于测试驱动开发

## 1.1 何谓测试驱动开发？

正如《测试驱动开发》一书中所言，测试驱动开发（*TDD*）以测试作为开发过程的中心，它要求在编写任何产品代码之前，首先编写用于定义产品代码行为的测试，而编写的产品代码又要以使测试通过为目标。测试驱动开发要求测试可以完全自动化地运行，在对代码进行重构前后必须运行测试。这是一种革命性的开发方法，能够造就简单、清晰、高质量的代码。

## 1.2 为何要使用 TDD？

测试驱动开发的主要好处有以下这些：

1. 代码覆盖。原则上讲，所编写的每个代码段都应当有至少一个相关的测试。这样，就可以有把握地相信系统中的所有代码都被执行过了。代码在编写时就会被测试，因此可以在开发过程中的早期发现缺陷。
2. 回归测试。测试集随着程序的开发增量进行开发。可以总是运行回归测试来确认对程序的修改没有引入新的 bug。
3. 简化的调试。当一个测试失败时，问题出在哪里应该很明显。新写的代码需要进行检查和修改。不需要使用调试工具来定位问题。关于测试驱动开发的使用报告建议说，在测试驱动的开发中一般不需要使用自动化的调试器（Martin 2007）。
4. 系统文档化。测试自身可以作为某种形式的文档来描述代码应该做什么。阅读测试可以使理解代码变得更容易。

## 1.3 如何进行测试驱动开发？

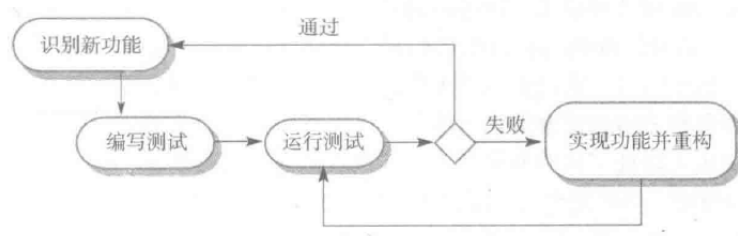


图 1: 测试驱动开发过程

## 2 实验内容——判断字符串是否是 IPV4 地址

采用测试驱动开发方法，将需求分为 5 个子功能实现，并使用 JUnit 框架分别进行测试。

### 2.1 判断字符串是否为空串

编写测试

```
@Test
void firstJudgeIpv4k(){
    assertTrue(strJudge.firstJudgeIpv4("1234567"));
    assertTrue(strJudge.firstJudgeIpv4("..."));
    assertTrue(strJudge.firstJudgeIpv4("00000"));
}
```

图 2: 正确用例测试

```
@Test
void firstJudgeIpv4Error(){
    assertFalse(strJudge.firstJudgeIpv4(""));
    assertFalse(strJudge.firstJudgeIpv4(null));
}
```

图 3: 错误用例测试

运行测试

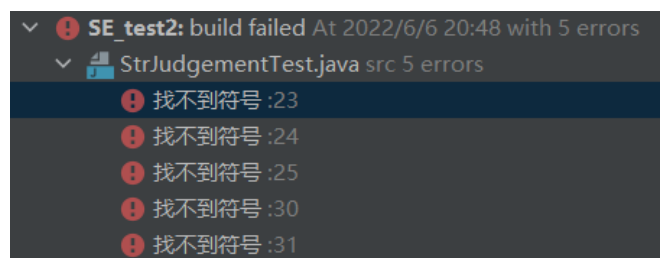


图 4: 第一次测试

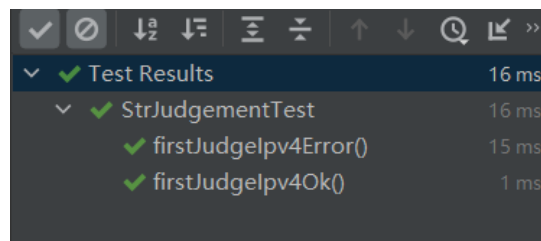
由于该功能尚未实现，无法通过测试。

### 实现功能并重构

```
public class StrJudgement {  
    private String[] parts;  
    public boolean firstJudgeIpv4(String address){  
        return address != null && address.length() != 0;  
    }  
}
```

图 5: 实现功能并重构

### 再次运行测试



✓	Test Results	16 ms
✓	StrJudgementTest	16 ms
✓	firstJudgeIpv4Error()	15 ms
✓	firstJudgeIpv4Ok()	1 ms

图 6: 第二次测试

测试通过

## 2.2 判断字符串是否可被. 分成四段

### 编写测试

```
@Test  
void secondJudgeIpv4Ok(){  
    assertTrue(strJudge.secondJudgeIpv4("255.3.3.255"));  
    assertTrue(strJudge.secondJudgeIpv4("1.1.1.1"));  
    assertTrue(strJudge.secondJudgeIpv4(".0.0.12"));  
}
```

图 7: 正确用例测试

```
@Test
void secondJudgeIpv4Error(){
    assertFalse(strJudge.secondJudgeIpv4("1234567"));
    assertFalse(strJudge.secondJudgeIpv4("0.0.0.0.0.0"));
    assertFalse(strJudge.secondJudgeIpv4(".1.1"));
}
```

图 8: 错误用例测试

## 运行测试

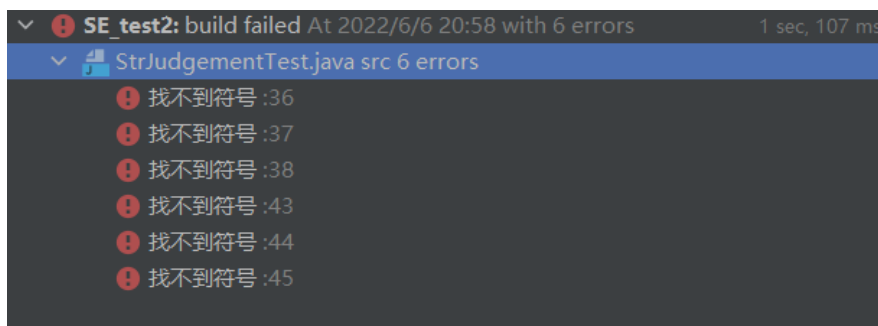


图 9: 第一次测试

由于该功能尚未实现，无法通过测试。

## 实现功能并重构

```
public boolean secondJudgeIpv4(String address){
    parts = address.split( regex: "\\.");
    return parts.length == 4;
}
```

图 10: 实现功能并重构

### 再次运行测试

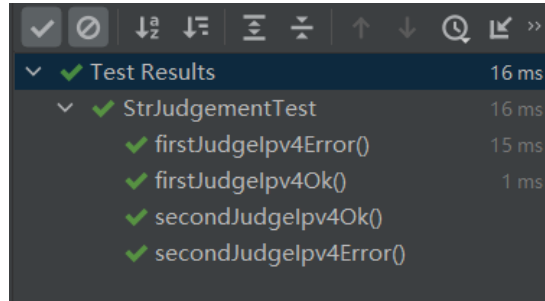


图 11: 第二次测试

测试通过

## 2.3 判断每一段是否是在 0 到 255 之间的数字

### 编写测试

```
@Test
void thirdJudgeIpv4Ok(){
    assertTrue(strJudge.thirdJudgeIpv4("255.255.255.255"));
    assertTrue(strJudge.thirdJudgeIpv4("01.01.1.1"));
    assertTrue(strJudge.thirdJudgeIpv4("0.0.0.0"));
}
```

图 12: 正确用例测试

```
@Test
void thirdJudgeIpv4Error(){
    assertFalse(strJudge.thirdJudgeIpv4("256.256.256.256"));
    assertFalse(strJudge.thirdJudgeIpv4("abc.acd.l.1"));
    assertFalse(strJudge.thirdJudgeIpv4("-1.0.-2.1"));
}
```

图 13: 错误用例测试

### 运行测试

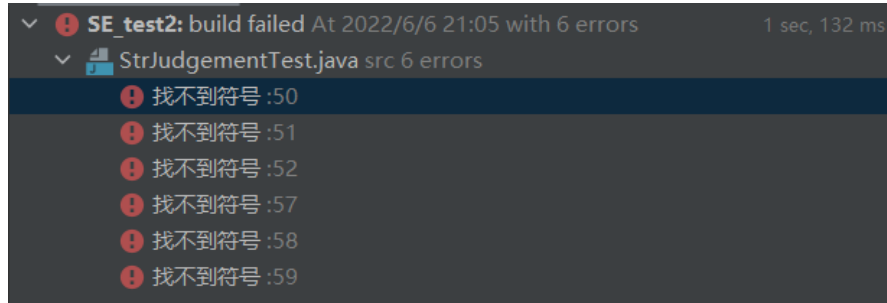


图 14: 第一次测试

由于该功能尚未实现，无法通过测试。

### 实现功能并重构

```
public boolean thirdJudgeIpv4(String address){
    secondJudgeIpv4(address);
    for (String part : parts) {
        try {
            int n = Integer.parseInt(part);
            if (n < 0 || n > 255) {
                return false;
            }
        } catch (NumberFormatException e) {
            return false;
        }
    }
    return true;
}
```

图 15: 实现功能并重构



### 再次运行测试

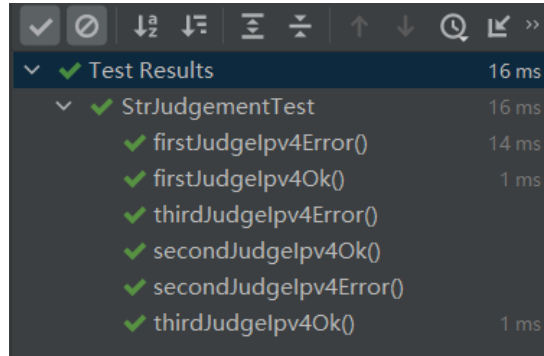


图 16: 第二次测试

测试通过

## 2.4 判断是否存在以 0 开头的非零数字

### 编写测试

```
@Test
void forthJudgeIpv4Ok(){
    assertTrue(strJudge.forthJudgeIpv4("255.255.255.255"));
    assertTrue(strJudge.forthJudgeIpv4("100.100.255.5"));
    assertTrue(strJudge.forthJudgeIpv4("0.0.0.0"));
}
```

图 17: 正确用例测试

```
@Test
void forthJudgeIpv4Error(){
    assertFalse(strJudge.forthJudgeIpv4("05.010.002.1"));
    assertFalse(strJudge.forthJudgeIpv4("056.110.khk"));
    assertFalse(strJudge.forthJudgeIpv4("0.0.0.000"));
}
```

图 18: 错误用例测试

### 运行测试

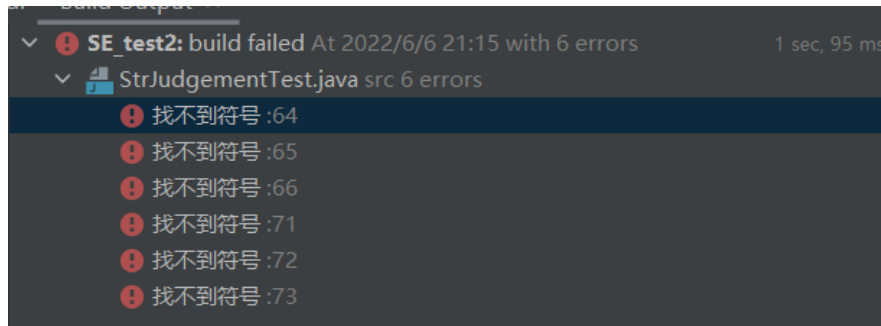


图 19: 第一次测试

由于该功能尚未实现，无法通过测试。

### 实现功能并重构

```
public boolean forthJudgeIpv4(String address){
    secondJudgeIpv4(address);
    for (String part : parts) {
        if(part.startsWith("0") && !part.equals("0")){
            return false;
        }
    }
    return true;
}
```

图 20: 实现功能并重构

### 再次运行测试

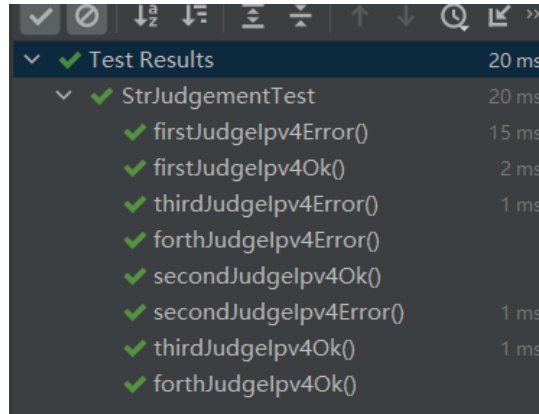


图 21: 第二次测试

测试通过

## 2.5 判断字符串是否同时满足上述条件

### 编写测试

```
@Test
void finalJudgeIpv4Ok(){
    assertTrue(strJudge.finalJudgeIpv4("255.255.255.255"));
    assertTrue(strJudge.finalJudgeIpv4("126.163.255.5"));
    assertTrue(strJudge.finalJudgeIpv4("0.0.0.1"));
}
```

图 22: 正确用例测试

```
@Test
void finalJudgeIpv4Error(){
    assertFalse(strJudge.finalJudgeIpv4(""));
    assertFalse(strJudge.finalJudgeIpv4("..1.1.1.2"));
    assertFalse(strJudge.finalJudgeIpv4("abd.126.16.1260"));
    assertFalse(strJudge.finalJudgeIpv4("056.12.5.126"));
}
```

图 23: 错误用例测试

## 运行测试

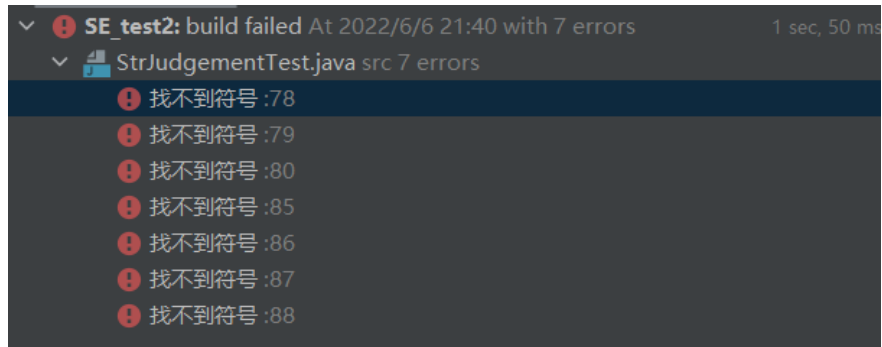


图 24: 第一次测试

由于该功能尚未实现，无法通过测试。

## 实现功能并重构

```
public boolean finalJudgeIpv4(String address){  
    return firstJudgeIpv4(address) && secondJudgeIpv4(address) && thirdJudgeIpv4(address) && forthJudgeIpv4(address);  
}
```

图 25: 实现功能并重构

## 再次运行测试

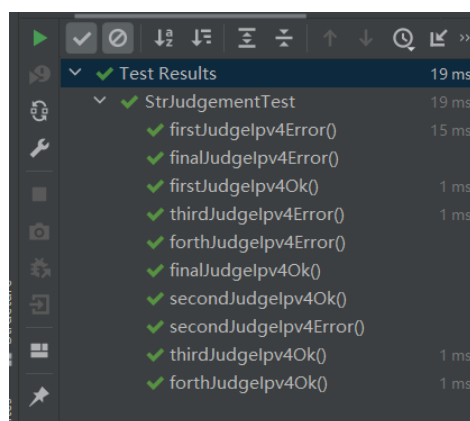


图 26: 第二次测试

测试通过

## 3 代码展示

### 3.1 程序代码

```
1      public class StrJudgement {
2          private String [] parts;
3          public boolean firstJudgeIpv4(String address){
4              return address != null && address.length()
5                  != 0;
6          }
7          public boolean secondJudgeIpv4(String address)
8          {
9              parts = address.split("\\.");
10             return parts.length == 4;
11         }
12         public boolean thirdJudgeIpv4(String address){
13             secondJudgeIpv4(address);
14             for (String part : parts) {
15                 try {
16                     int n = Integer.parseInt(part);
17                     if (n < 0 || n > 255) {
18                         return false;
19                     }
20                 } catch (NumberFormatException e) {
21                     return false;
22                 }
23             }
24             return true;
25         }
26
27         public boolean forthJudgeIpv4(String address){
28             secondJudgeIpv4(address);
29             for (String part : parts) {
30                 if(part.startsWith("0") && !part.
                    equals("0")){
```

```
31         return false;
32     }
33 }
34     return true;
35 }
36
37     public boolean finalJudgeIpv4(String address){
38         return firstJudgeIpv4(address) &&
39             secondJudgeIpv4(address) &&
40             thirdJudgeIpv4(address) &&
41             forthJudgeIpv4(address);
42     }
43 }
```

### 3.2 测试代码

```
1  import org.junit.jupiter.api.AfterEach;
2  import org.junit.jupiter.api.BeforeEach;
3  import org.junit.jupiter.api.Test;
4  import java.util.Arrays;
5  import static org.junit.jupiter.api.Assertions.*;
6
7  class StrJudgementTest {
8      StrJudgement strJudge;
9      @BeforeEach
10     void setUp() {
11         strJudge = new StrJudgement();
12     }
13
14     @AfterEach
15     void tearDown() {
16         strJudge = null;
17     }
18
19     @Test
20     void firstJudgeIpv4Ok() {
```

```
21         assertTrue(strJudge.firstJudgeIpv4("
22             1234567"));
23         assertTrue(strJudge.firstJudgeIpv4("..."));
24         ;
25         assertTrue(strJudge.firstJudgeIpv4("00000"
26             ));
27     }
28
29     @Test
30     void firstJudgeIpv4Error() {
31         assertFalse(strJudge.firstJudgeIpv4(""));
32         assertFalse(strJudge.firstJudgeIpv4(null));
33         ;
34     }
35
36     @Test
37     void secondJudgeIpv4Ok() {
38         assertTrue(strJudge.secondJudgeIpv4("
39             255.3.3.255"));
40         assertTrue(strJudge.secondJudgeIpv4("
41             1.1.1.1"));
42         assertTrue(strJudge.secondJudgeIpv4("
43             .0.0.12"));
44     }
45
46     @Test
47     void secondJudgeIpv4Error() {
48         assertFalse(strJudge.secondJudgeIpv4("
49             1234567"));
50         assertFalse(strJudge.secondJudgeIpv4("
51             0.0.0.0.0.0.0"));
52         assertFalse(strJudge.secondJudgeIpv4(".1.1.1
53             "));
54     }
55
56     @Test
```

```
47     void thirdJudgeIpv4Ok() {
48         assertTrue(strJudge.thirdJudgeIpv4("
49             255.255.255.255"));
50         assertTrue(strJudge.thirdJudgeIpv4("
51             01.01.1.1"));
52         assertTrue(strJudge.thirdJudgeIpv4("
53             0.0.0.0"));
54     }
55
56     @Test
57     void thirdJudgeIpv4Error() {
58         assertFalse(strJudge.thirdJudgeIpv4("
59             256.256.256.256"));
60         assertFalse(strJudge.thirdJudgeIpv4("abc.
61             acd.1.1"));
62         assertFalse(strJudge.thirdJudgeIpv4("
63             -1.0.-2.1"));
64     }
65
66     @Test
67     void forthJudgeIpv4Ok() {
68         assertTrue(strJudge.forthJudgeIpv4("
69             255.255.255.255"));
70         assertTrue(strJudge.forthJudgeIpv4("
71             100.100.255.5"));
72         assertTrue(strJudge.forthJudgeIpv4("
73             0.0.0.0"));
74     }
75
76     @Test
77     void forthJudgeIpv4Error() {
78         assertFalse(strJudge.forthJudgeIpv4("
79             05.010.002.1"));
80         assertFalse(strJudge.forthJudgeIpv4("
81             056.110.khk"));
82         assertFalse(strJudge.forthJudgeIpv4("
83             0.0.0.0"));
```



```

                                0.0.0.000")));
72     }
73
74     @Test
75     void finalJudgeIpv4Ok() {
76         assertTrue(strJudge.finalJudgeIpv4("
77             255.255.255.255"));
78         assertTrue(strJudge.finalJudgeIpv4("
79             126.163.255.5"));
80         assertTrue(strJudge.finalJudgeIpv4("
81             0.0.0.1"));
82     }
83
84     @Test
85     void finalJudgeIpv4Error() {
86         assertFalse(strJudge.finalJudgeIpv4(""));
87         assertFalse(strJudge.finalJudgeIpv4("
88             ..1.1.1.2"));
89         assertFalse(strJudge.finalJudgeIpv4("abd
90             .126.16.1260"));
91         assertFalse(strJudge.finalJudgeIpv4("
92             056.12.5.126"));
93     }
94 }
```

## 4 参考资料

- [1] 教学课件：感谢华中科技大学软件学院刘小峰老师！
- [2] (美) 贝克 (Kent Beck) 著. 孙平等译. 测试驱动开发. 北京：中国电力出版社. 2004
- [3] (英) 伊恩·萨默维尔 (Ian Sommerville) 著. 彭鑫等译. 软件工程 (原书第 10 版). 北京：机械工业出版社. 2018