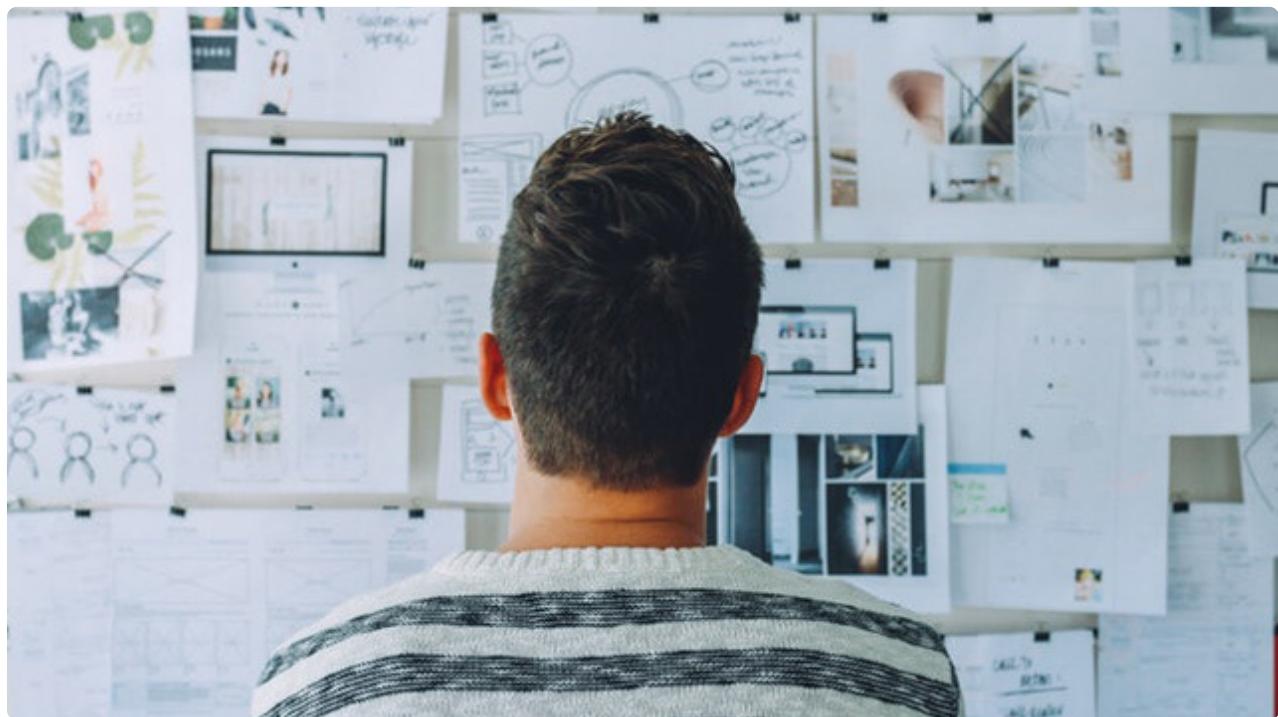


21 单元测试构造数据的正确姿势

更新时间：2019-12-04 10:01:57



不经一翻彻骨寒，怎得梅花扑鼻香。——宋帆

1.前言

前面讲到了单元测试的概念和好处，讲到了 Java 单元测试常用框架。

写过很多单元测试的朋友会发现，单元测试的重要环节就是构造测试数据，单元测试构造测试数据往往非常耗时，这也是很多人不喜欢写单元测试的重要原因之一。

因此本节将重点讲述有哪些单元测试中构造数据的方式，各种构造测试数据方式的优劣以及实际开发中该如何选择。

2构造单测数据的方式

2.1 手动

所谓手动构造单元测试工具，是指在测试类或者函数中直接声明测试数据，或在初始化函数中填充数据：

```
private List<String> mockStrList;

@Before
public void init() {
    mockStrList = new ArrayList<>();
    final int size = 10;
    for (int i = 0; i < size; i++) {
        mockStrList.add("something" + i);
    }
}
```

还可以在测试类中编写私有 `mock` 数据的函数来实现：

```
private UserDO mockUserDO() {
    UserDO userDO = new UserDO();
    userDO.setId(0L);
    userDO.setName("测试");
    userDO.setAge(0);
    userDO.setNickName("test");
    userDO.setBirthDay(Date.from(Instant.now()));
    return userDO;
}
```

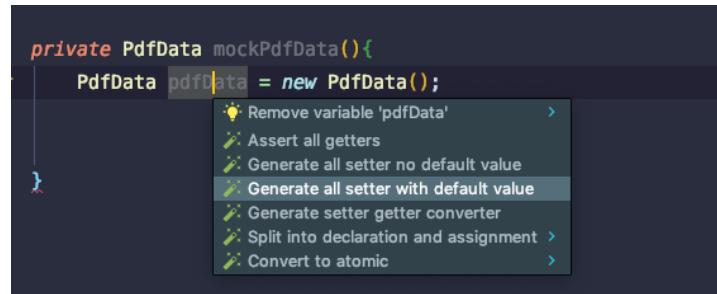
上述手动构造测试对象，当属性较多时，容易出错而且占据源码空间，而且不太优雅。

2.2 半自动

当所要构造的数据为复杂对象(属性较多的对象)时，手动构造对象非常耗时，而且属性设置容易遗漏。

所谓半自动是指使用插件自动填充所要构造对象的属性。

比如可以使用 **“Generate All setters”** IDEA 插件，选择 “generate all setter with default value“ 填充默认值，效率提高很多。



生成如下代码：

```
private PdfData mockPdfData() {
    // 使用插件填充PdfData
    PdfData pdfData = new PdfData();
    pdfData.setId(0);
    pdfData.setName("some");
    pdfData.setWaterMark("test");
    pdfData.setPages(4);
    // 再次使用一次插件，填充PdfAttribute
    PdfAttribute pdfAttribute = new PdfAttribute();
    pdfAttribute.setWeight(1024L);
    pdfAttribute.setHeight(512L);
    pdfData.setPdfAttribute(pdfAttribute);
    return pdfData;
}
```

还有一种非常常见的“半自动”构造测试数据的方式，采用 **JSON** 序列化和反序列化方式。

将构造的对象通过 **JSON** 序列化到 **JSON** 文件里，使用时反序列化为Java对象即可：

```

@Test
public void testPdfData() {
    // 构造测试数据
    PdfData pdfData = ResourceUtil.parseJson(PdfData.class, "/data/pdfData.json");
    System.out.println(JSON.toJSONString(pdfData));

    log.info("构造的数据:{}", JSON.toJSONString(pdfData));

    // 测试export 函数
    Boolean export = PdfUtil.export(pdfData);
    Assert.assertTrue(export);
}

```

2.3 自动

半自动的方式构造单元测试数据效率仍然不够高，而且缺乏灵活性，比如需要构造随机属性的对象，需要构造不同属性的 N 个对象，就会造成编码复杂度陡增。

因此，`java-faker` 和 `easy-random` 应运而生。

2.3.1 java-faker

Java 构造测试数据中最常见的一种场景是：构造字符串。

如果想随机构造人名、地名、天气、学校、颜色、职业，甚至符合某正则表达式的字符串等，肿么办？

`java-faker` 是不二之选。

源码地址：<https://github.com/DiUS/java-faker>

基本用法如下：

```

@Slf4j
public class FakerTest {

    @Test
    public void test() {
        // 指定语言
        Faker faker = new Faker(new Locale("zh-CN"));

        // 姓名
        String name = faker.name().fullName();
        log.info(name);
        String firstName = faker.name().firstName();
        String lastName = faker.name().lastName();
        log.info(lastName + firstName);
        // 街道
        String streetAddress = faker.address().streetAddress();
        log.info(streetAddress);

        // 颜色
        Color color = faker.color();
        log.info(color.name() + "-->" + color.hex());

        // 大学
        University university = faker.university();
        log.info(university.name() + "-->" + university.prefix() + ":" + university.suffix());
    }
}

```

另外特别建议大家通过 Codota 的方式来查看其他开源项目中该类或者函数的用法：

The screenshot shows a Java code editor on the left with a snippet of code using the Faker library. On the right, a sidebar titled 'Best code snippets for: Faker.name()' lists snippets from various projects. The first snippet is from 'benas/random-beans' and overrides the 'name()' method to return a random first name. The second snippet is also from 'benas/random-beans' and overrides the 'name()' method to return a random last name. There is also a snippet from 'neo4j-contrib/neo4j-apoc-procedures'.

```
import ...  
  
@Slf4j  
public class FakeTest {  
  
    @Test  
    public void test() {  
        Faker faker = new Faker(new Locale("language"));  
  
        // 姓名  
        String name = faker.name().fullName();  
        log.info(name);  
        String firstName = faker.name().firstName();  
        String lastName = faker.name().lastName();  
        log.info(lastName + firstName);  
        // 街道  
        String streetAddress = faker.address().street();  
        log.info(streetAddress);  
  
        // 颜色  
    }  
}
```

还可以下载源码后查看核心类的核心函数来了解主要功能。

也可以通过源码提供的单元测试代码来学习更多用法，还可以通过调试来验证一些效果：

The screenshot shows an IDE with a debugger session. A tooltip for the 'faker.regexify()' method is displayed, showing the expression 'faker.regexify("[aeiou]{2,3}")' and the result 'result = "eua"'. The tooltip also includes a 'Evaluate' button and a 'Use ⌘C to add to Watches' link.

```
39  
40    @Test  
41    public void regexifyShouldGenerateBracketsQuantifiersOnBasicCharacterClasses() {  
42        assertThat(faker.regexify("[aeiou]{2,3}"), matchesRegularExpression("[aeiou]{2,3}"));  
43    }  
  
45    @Test  
46    public void regexifyShouldGenerateBracketsQuantifiersOnBasicCharacterClasses() {  
47        assertThat(faker.regexify("[aeiou]{2,3}"), matchesRegularExpression("[aeiou]{2,3}"));  
48    }  
  
50    @Test  
51    public void regexifyShouldGenerateBracketsQuantifiersOnBasicCharacterClasses() {  
52        assertThat(faker.regexify("[aeiou]{2,3}"), matchesRegularExpression("[aeiou]{2,3}"));  
53    }  
  
FakerTest > regexifyShould...
```

2.3.2 easy-random

Java-faker 虽然可以构造不同种类的字符串测试数据，但是如果需要构造复杂对象就有些“力不从心”。

此时 easy-random 就要上场了。

源码地址：<https://github.com/j-easy/easy-random>

官方文档：<https://github.com/j-easy/easy-random/wiki>

easy-random 可以轻松构造复杂对象，支持定义对象中集合长度，字符串长度范围，生成集合等。

正如前面手动构造和半自动构造测试数据所给出的示例代码所示，构造复杂对象非常耗时且编码量较大，而使用 **easy-random**，直接调用 `easyRandom#nextObject` 一行代码即可自动构建测试对象：

```
private EasyRandom easyRandom = new EasyRandom();  
  
{@Test  
public void testPdfData() {  
    // 构造测试数据  
    PdfData pdfData = easyRandom.nextObject(PdfData.class);  
    System.out.println(JSON.toJSONString(pdfData));  
  
    log.info("构造的数据:{}", JSON.toJSONString(pdfData));  
  
    // 测试export 函数  
    Boolean export = PdfUtil.export(pdfData);  
    Assert.assertTrue(export);  
}}  
}
```

Easy-random 还支持通过 **EasyRandomParameters** 来定制构造对象的细节，如对象池大小、字符集、时间范围、日期范围、字符串长度范围、集合大小的范围等。

```
EasyRandomParameters parameters = new EasyRandomParameters()  
.seed(123L)  
// 对象池大小  
.objectPoolSize(100)  
// 对象图的最大深度  
.randomizationDepth(3)  
// 字符集  
.charset.forName("UTF-8"))  
// 时间范围  
.timeRange(nine, five)  
// 日期范围  
.dateRange(today, tomorrow)  
// 字符串长度范围  
.stringLengthRange(5, 50)  
// 集合元素个数的范围  
.collectionSizeRange(1, 10)  
// 接口或抽象类时是否扫描具体的实现类  
.scanClasspathForConcreteTypes(true)  
// 是否重写默认的初始化方法  
.overrideDefaultInitialization(false)  
// 是否忽略错误  
.ignoreRandomizationErrors(true);  
  
EasyRandom easyRandom = new EasyRandom(parameters);
```

建议大家一定要拉取 **easy-random** 源码，查看更多属性，包括 **EasyRandomParameters** 的默认值，以及运行其官方的单元测试来了解更多高级用法。

如可以查看其日期时间范围参数测试类：**DateTimeRangeParameterTests**，来学习如何设置日期范围构造数据的日期范围：

```

@Test
void testDateRange() {
    // Given
    LocalDate minDate = LocalDate.of(2016, 1, 1);
    LocalDate maxDate = LocalDate.of(2016, 1, 31);
    EasyRandomParameters parameters = new EasyRandomParameters().dateRange(minDate, maxDate);

    // When
    TimeBean timeBean = new EasyRandom(parameters).nextObject(TimeBean.class);

    // Then
    assertThat(timeBean.getLocalDate()).isAfterOrEqualTo(minDate).isBeforeOrEqual(maxDate);
}

```

我们不仅可以通过官方的单元测试来学习该框架的用法，还通过源码单元测试的范例来学习如何更好地编写单元测试。

可以在单元测试中打断点来观察构造对象的属性值，甚至可以通过单步来研究构造对象的过程。

更多高级用法，请自行拉取源码继续学习。

3如何选择？

前面讲到了构造单元测试数据的常用手段主要分为三种：手动构造、半自动、自动构造。

那么该如何做出恰当的选择呢？

下面给出一些建议：

- 当构造的测试数据非常简单时，如构造一个整型测试数据或者待构造的对象属性极少时，可以使用手动构造的方式，简单快速；
- 当待构造的对象属性均需要手动修改时，建议采用半自动的方式，使用插件构造测试对象并手动赋值或者使用 JSON 反序列化的方式；
- 当待构造的测试数据为特定字符串时，如人名、地名、大学名称时，建议使用 **java-faker**；
- 当待构造的测试对象较为复杂时，如属性极多或者属性中又嵌套复杂对象时，建议使用 **easy-random**。

4总结

本小节主要介绍了构造单元测试数据的几种常见手段，如手动构造、半自动、自动三种方式。并介绍了每种方式的常见构造方法以及各自的优劣，并给出了如何根据具体场景做出恰当的选择。

希望大家在学习其他知识时，也要对知识进行归类和对比，这样才能深刻理解知识，才能举一反三。

下一节将给出单元测试的一些具体案例。

5课后作业

拉取 **java-faker** 和 **easy-random** 的源码，运行关键类的单测来快速学习它们的用法。

}

