# Optional 是个好东西，你真的会用么？

原创 了不起 Java极客技术 2023-05-19 07:30 发表于湖北



## 一、摘要

NullPointerException，中文名：**空指针异常**，也简称 NPE，是软件系统中最常见的错误异常之一。

很久以前 Google Guava 项目引入了 `Optional` 作为解决空指针异常的一种方式，不赞成写过多的代码来显式检查 `null`，以期望程序员写出整洁同时可读性更高的代码。

受 Google Guava 的影响，Optional 现在也成为了Java 8 及以上库代码的一部分。

在介绍 `Optional` 技术之前，我们不禁会发出一个疑问：为什么谷歌不赞成写过多的代码来显式检查 `null`？

下面是某个常见的参数判空代码，样例如下。

```
// 判断行政区是否为空
if(country != null){
    // 判断行政区的上一级，行政城市是否为空
    if(country.getCity() != null){
        // 判断行政城市的上一级，行政省是否为空
        if(country.getCity().getProvince() != null){
            // 获取对应的行政省相关的数据
            return country.getCity().getProvince().getName();
        }
    }
}
```

这还是最普通的三层判断，假如有很大一段业务逻辑处理的时候，你会发现代码不光看起来很臃肿，并且难以阅读，可读性很差！

如果调整为使用 `Optional` 来编写的话，可以转换成如下写法：

```
// 获取当前行政区最顶级的省信息名称
String result  = Optional.ofNullable(country)
                .map(Country::getCity)
                .map(City::getProvince)
                .map(Province::getName)
                .orElse("error");
```

采用 `Optional` 来编程之后，整个代码的可读性和整洁度，是不是要干净很多！

这也是为什么推荐大家使用 `Optional` 的原因啦！

当然废话也不多说，代码直接撸起来！

## 二、案例实践

在 JDK8 中，Optional 共有 12 个核心方法，下面我们一起来看看他们的用法！

### 2.1、empty()

empty 方法返回一个不包含值的 Optional 实例，单独使用没什么意义，主要和其他方法搭配使用。

```
Optional optional = Optional.empty();
System.out.println(optional);
```

```
-- 输出结果
Optional.empty
```

### 2.2、of()

of 方法会返回一个 Optional 实例，如果传入的值非空，会返回包含指定值的对象；如果传入空，会立刻抛出空指针异常。

```
// 非空情况下，会正常返回
Optional optional = Optional.of("hello world");
System.out.println(optional);
```

```
-- 输出结果
Optional[hello world]
```

```
// 为空情况下，会抛空指针异常
Optional optional = Optional.of(null);
System.out.println(optional);
```

```
-- 输出结果
Exception in thread "main" java.lang.NullPointerException
 at java.util.Objects.requireNonNull(Objects.java:203)
 at java.util.Optional.<init>(Optional.java:96)
 at java.util.Optional.of(Optional.java:108)
```

### 2.3、ofNullable()

ofNullable 方法会返回一个 Optional 实例，如果传入的值非空，会返回包含指定值的对象；如果传入空，会返回不包含任何值的 empty 对象，也就是最开始介绍的 `Optional.empty()` 对象。

```
// 非空情况下，会正常返回
Optional optional = Optional.ofNullable("hello world");
System.out.println(optional);
```

```
-- 输出结果
Optional[hello world]
```

```
// 为空情况下，会返回 empty 对象
Optional optional = Optional.ofNullable(null);
System.out.println(optional);
```

```
-- 输出结果
Optional.empty
```

### 2.4、isPresent()

isPresent 方法用来判断实例是否包含值，如果包含非空值，返回 true，否则返回 false。

```
// 非空值，返回true
boolean rs1 =  Optional.ofNullable("hello").isPresent();
System.out.println(rs1);

// 空值，返回false
boolean rs2 =  Optional.ofNullable(null).isPresent();
System.out.println(rs2);
```

```
-- 输出结果
true
false
```

### 2.5、get()

get 方法，如果实例包含非空值，则返回当前值；否则抛出 NoSushElementException 异常。

```
// 非空值，返回当前值
Object rs =  Optional.ofNullable("hello world").get();
System.out.println(rs);
```

```
-- 输出结果
hello world
```

```
// 空值，会抛出 NoSushElementException 异常
Object rs =  Optional.ofNullable(null).get();
System.out.println(rs);
```

```
-- 输出结果
Exception in thread "main" java.util.NoSuchElementException: No value present
 at java.util.Optional.get(Optional.java:135)
```

### 2.6、ifPresent()

ifPresent 方法作用是当实例包含非空值时，执行传入的 Consumer，比如调用一些其他方法；如果包含的值为空，不执行任何操作。

```
Optional.ofNullable("hello world")
                .ifPresent( x -> {
                    System.out.println(x);
                });
```

```
-- 输出结果
hello world
```

### 2.7、filter()

filter 方法用于过滤不符合条件的值，接收一个 `Predicate` 参数，如果符合条件，会返回当前的 `Optional` 实例，否则返回 empty 实例。

```
Optional.ofNullable("hello world")
                .filter(x -> x.contains("hello"))
                .ifPresent(x -> {
                    System.out.println(x);
                });
```

```
-- 输出结果
hello world
```

### 2.8、map()

map 方法是链式调用避免空指针的核心方法，当实例包含值时，对值执行传入的 `Function` 函数接口方法，并返回一个代表结果值新的 `Optional` 实例，也就是将返回的结果再次包装成 `Optional` 对象。

```
Optional.ofNullable("hello+world")
                .map(t -> {
                    if(t.contains("+")){
                        return t.replace("+", " ");
                    }
                    return t;
                }).ifPresent(t -> {
                    System.out.println(t);
                });
```

```
-- 输出结果
hello world
```

### 2.9、flatMap()

flatMap 方法与 map 方法类似，唯一不同的地方在于：**需要手动将返回的值，包装成`Optional`实例，并且参数值不允许为空。**

```
Optional.ofNullable("hello+world")
                .flatMap(t -> {
                    if(t.contains("+")){
                        t =  t.replace("+", " ");
                    }
                    // 不同之处
                    return Optional.of(t);
                }).ifPresent(t -> {
                    System.out.println(t);
                });
```

```
-- 输出结果
hello world
```

### 2.10、orElse()

orElse 方法作用是如果实例包含非空值，那么返回当前值；否则返回指定的默认值。

```
Object rs =  Optional.ofNullable(null).orElse("null");
System.out.println(rs);
```

```
-- 输出结果
null
```

### 2.11、orElseGet()

orElseGet 方法作用是如果实例包含非空值，返回这个值；否则，它会执行作为参数传入的`Supplier`函数式接口方法，并返回其执行结果。

```
Object result = Optional.ofNullable(null)
                .orElseGet(() -> {
                    return "error";
                });
System.out.println(result);
```

```
-- 输出结果
error
```

### 2.12、orElseThrow()

orElseThrow 方法作用是如果实例包含非空值，返回这个值；否则，它会执行作为参数传入的异常类。

```
Optional.ofNullable(null)
                .orElseThrow(() -> new RuntimeException("参数为空"));
```

```
-- 输出结果
Exception in thread "main" java.lang.RuntimeException: 参数为空
 at com.x.x.x.x.OptionalTest.lambda$main$10(OptionalTest3.java:144)
 at java.util.Optional.orElseThrow(Optional.java:290)
```

## 三、小结

以上就是 JDK8 新增的`Optional`类的常用方法总结，其中`ofNullable`、`map`和`orElse`方法搭配使用的最多。

另外`orElse`、`orElseGet`、`orElseThrow`区别如下：

- `orElse`：如果实例包含空值，返回传入指定的值
- `orElseGet`：如果实例包含空值，返回传入的方法中返回值
- `orElseThrow`：如果实例包含空值，返回指定的异常类型

在实际使用的时候，还得结合具体的场景进行合理选择，有时候并不是全部采用`Optional`来解决`NPE`异常代码才更加优雅，比如当前对象比较简单，就是一个简单判断，通过`obj != null`足以解决问题。

因此在保证业务功能的正确和稳定性的基础之上，适当的选择相关的工具来优化代码的整洁度和可读性，更能发挥出锦上添花的效果！

## 四、参考

1、https://blog.csdn.net/yy339452689/article/details/110670282