链滴

还在重复写空指针检查代码？考虑使用 Optional 吧！

作者：9526xu
原文链接：https://ld246.com/article/1571995971705
来源网站：链滴

一、前言

如果要给 Java 所有异常弄个榜单，我会选择将 NullPointerException 放在榜首。这个异常潜伏在代中，就像个遥控炸弹，不知道什么时候这个按钮会被突然按下（传入 null 对象）。

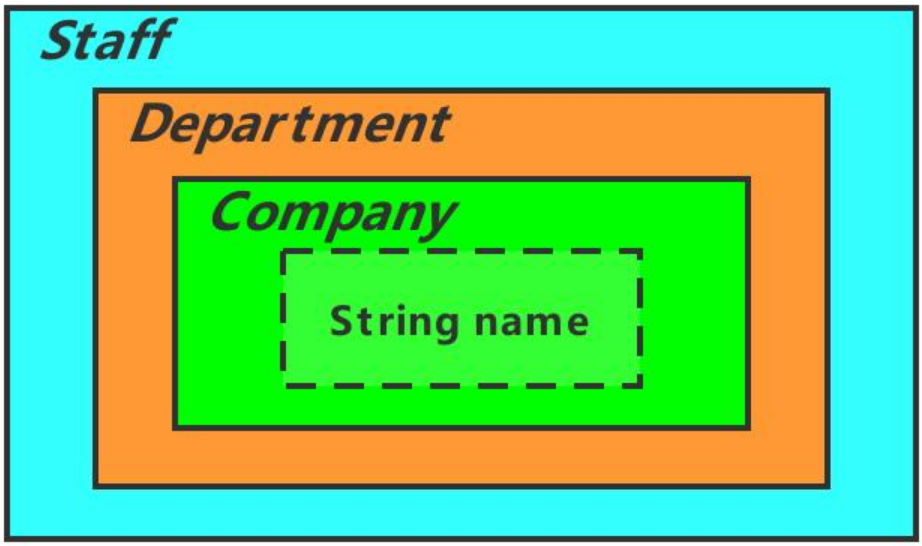
<!--more-->

还记得刚入行程序员的时候，三天两头碰到空指针异常引发的 Bug，解决完一个，又在另一处碰到。时候师兄就教我，不要相信任何『对象』，特别是别人给你的，这些地方都加上判断。于是代码通常会变成下面这样：

```
if(obj!=null){
    // do something
}
```

有了这个防御之后，虽然不用再担心空指针异常，但是过多的判断语句使得代码变得臃肿。

假设我们存在如下对象关系



原本为了获取图中的 name 属性，原本一句代码就可以轻松完成。

```
Staff staff=..;
staff.getDepartment().getCompany().getName();
```

但是很不幸，为了代码的安全性，我们不得不加入空指针判断代码。

```
Staff staff=..;
if (staff != null) {
    Department department = staff.getDepartment();
    if (department != null) {
        Company company = department.getCompany();
        if (company != null) {
            return company.getName();
        }
    }
}
return "Unknown";
```

当其中对象为 null 时，可以返回默认值，如上所示。也可以直接抛出其他异常快速失败。

虽然上面代码变得更加安全，但是过多嵌套 if 语句降低代码整体可读性，提高复杂度。

所幸 Java 8 引入引入一个新类 Java.util.Optional<T>,依靠 Optional 类提供 API，我们可以写出既全又具有阅读性的代码。

还在使用 JDK 6？那你也别急着关闭这篇文章。可以考虑使用 Guava Optional。不过需要注意的是 Guava Optional API 与 JDK 存在差异，以下以 JDK8 Optional 为例。

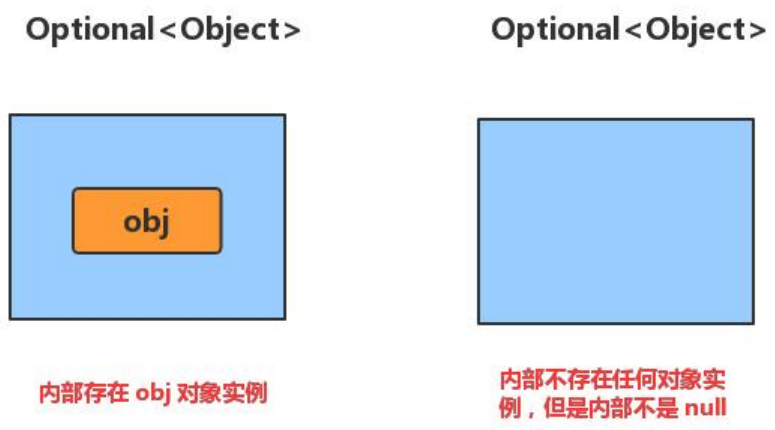
二、Optional API

2.1、Optional#of 与 Optional#ofNullable

Optional<T> 本质是一个容器，需要我们将对象实例传入该容器中。Optional 的构造方法为 private 无法直接使用 new 构建对象，只能使用 Optional 提供的静态方法创建。

Optional 三个创建方法如下：

- Optional.of(obj),如果对象为 null，将会抛出 NPE。
- Optional.ofNullable(obj),如果对象为 null,将会创建不包含值的 **empty** Optional 对象实例。
- Optional.empty() 等同于 Optional.ofNullable(null)
-



只有在确定对象不会为 null 的情况使用 Optional#of，否则建议使用 Optional#ofNullable方法。

2.2、Optional#get 与 Optional#isPresent

对象实例存入 Optional 容器中之后，最后我们需要从中取出。Optional#get 方法用于取出内部对实例，不过需要注意的是，如果是 **empty Optional** 实例，由于容器内没有任何对象实例，使用 get 方法将会抛出 NoSuchElementException 异常。

为了防止异常抛出，可以使用 Optional#isPresent 。这个方法将会判断内部是否存在对象实例，若在则返回 true。

示例代码如下:

```
Optional<Company> optCompany = Optional.ofNullable(company);
// 与直接使用空指针判断没有任何区别
if (optCompany.isPresent()) {
    System.out.println(optCompany.get().getName());
}
```

仔细对比，可以发现上面用法与空指针检查并无差别。刚接触到 Optional ，看到很多文章介绍这个法，那时候一直很疑惑，这个解决方案不是更加麻烦?

后来接触到 Optional 其他 API，我才发现这个类真正有意义是下面这些 API。如果使用过 Java8 Stre m 的 API，下面 Optional API 你将会很熟悉。

2.3、Optional#ifPresent

通常情况下，空指针检查之后，如果对象不为空，将会进行下一步处理，比如打印该对象。

```
Company company = ...;
if(company!=null){
    System.out.println(company);
}
```

上面代码我们可以使用 Optional#ifPresent 代替，如下所示：

```
Optional<Company> optCompany = ...;
optCompany.ifPresent(System.out::println);
```

使用 ifPresent 方法，我们不用再显示的进行检查，如果 Optional 为空，上面例子将不再输出。

2.4、Optional#filter

有时候我们需要某些属性满足一定条件，才进行下一步动作。这里假设我们当 Company name 属性 Apple，打印输出 ok。

```
if (company != null && "Apple".equals(company.getName())) {
    System.out.println("ok");
}
```

下面使用 Optional#filter 结合 Optional#ifPresent 重写上面的代码，如下所示：

```
Optional<Company> companyOpt=...;
companyOpt
        .filter(company -> "Apple".equals(company.getName()))
        .ifPresent(company -> System.out.println("ok"));
```

filter 方法将会判断对象是否符合条件。如果不符合条件，将会返回一个空的 Optional 。

2.5、Optional#orElse 与 Optional#orElseThrow

当一个对象为 null 时，业务上通常可以设置一个默认值，从而使流程继续下去。

```
String name = company != null ? company.getName() : "Unknown";
```

或者抛出一个内部异常，记录失败原因，快速失败。

```
if (company.getName() == null) {
    throw new RuntimeException();
}
```

Optional 类提供两个方法 orElse 与 orElseThrow ，可以方便完成上面转化。

```
// 设置默认值
String name=companyOpt.orElse(new Company("Unknown")).getName();

// 抛出异常
String name=companyOpt.orElseThrow(RuntimeException::new).getName();
```

如果 Optional 为空，提供默认值或抛出异常。

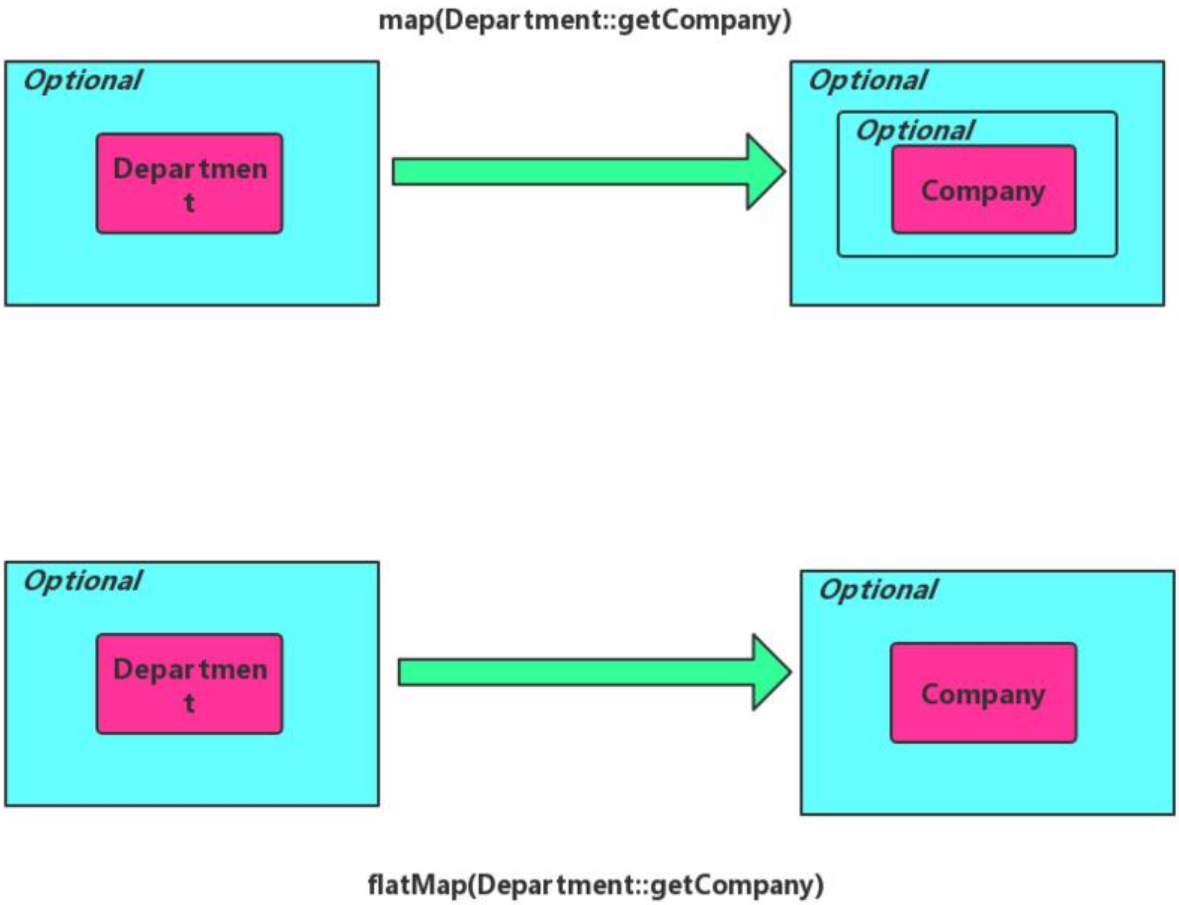
2.6、Optional#map 与 Optional#flatMap

熟悉 Java8 Stream 同学的应该了解，Stream#map 方法可以将当前对象转化为另外一个对象， Optional#map 方法也与之类似。

```
Optional<Company> optCompany = ...;
Optional<String> nameopt = optCompany.map(Company::getName);
```

map 方法可以将原先 Optional<Company> 转变成 Optional<String> ，此时 Optional 内部对象成 String 类型。如果转化之前 Optional 对象为空，则什么也不会发生。

另外 Optional 还有一个 flatMap 方法，两者区别见下图。



Department#getCompany 返回对象为 Optional<Company>

三、代码重构

上面我们学习了 Optional 类主要 API ，下面我们使用 Optional 重构文章刚开头的代码。为了方便者对比，将上面的代码复制了下来。

代码重构前：

```
if (staff != null) {
    Department department = staff.getDepartment();
    if (department != null) {
        Company company = department.getCompany();
        if (company != null) {
            return company.getName();
        }
    }
}
return "Unknown";
```

首先我们需要将 Staff ， Department 修改 getter 方法返回结果类型改成 Optional。

```
public class Staff {
    private Department department;
    public Optional<Department> getDepartment() {
        return Optional.ofNullable(department);
    }
    ...
}
public class Department {

    private Company company;
    public Optional<Company> getCompany() {
        return Optional.ofNullable(company);
    }
    ...
}

public class Company {
    private String name;
    public String getName() {
        return name;
    }
    ...
}
```

然后综合使用 Optional API 重构代码如下：

```
Optional<Staff> staffOpt =...;
staffOpt
        .flatMap(Staff::getDepartment)
        .flatMap(Department::getCompany)
        .map(Company::getName)
        .orElse("Unknown");
```

可以看到重构之后代码利用 Optional 的 Fluent Interface，以及 lambda 表达式，使代码更加流畅贯，并且提高代码整体可读性。

四、帮助文章

1、Tired of Null Pointer Exceptions? Consider Using Java SE 8's Optional!

3、Optionals: Patterns and Good Practices

3、Java8 in Action

原文链接：还在重复写空指针检查代码？考虑使用 Optional 吧！