链滴

使用 Java8 Optional 的正确姿势

作者：honglan

原文链接：https://ld246.com/article/1510626954005

来源网站：链滴

原文出处： 隔叶黄莺 Unmi Blog

我们知道 Java 8 增加了一些很有用的 API, 其中一个就是 Optional. 如果对它不稍假探索, 只是轻描写的认为它可以优雅的解决 NullPointException 的问题, 于是代码就开始这么写了

```
Optional user = ……
if (user.isPresent()) {
return user.getOrders();
} else {
return Collections.emptyList();
}
```

那么不得不说我们的思维仍然是在原地踏步, 只是本能的认为它不过是 User 实例的包装, 这与我们之写成

```
User user = …..
if (user != null) {
return user.getOrders();
} else {
return Collections.emptyList();
}
```

实质上是没有任何分别. 这就是我们将要讲到的使用好 Java 8 Optional 类型的正确姿势.

在里约奥运之时, 新闻一再提起五星红旗有问题, 可是我怎么看都看不出来有什么问题, 后来才道是小星膜拜中央的姿势不对. 因此我们千万也别对自己习以为常的事情觉得理所当然, 丝毫不会觉得有何不妥 换句话说也就是当我们切换到 Java 8 的 Optional 时, 不能继承性的对待过往 null 时的那种思维, 应掌握好新的, 正确的使用 Java 8 Optional 的正确姿势.

直白的讲, 当我们还在以如下几种方式使用 Optional 时, 就得开始检视自己了

1. 调用 isPresent() 方法时
2. 调用 get() 方法时
3. Optional 类型作为类/实例属性时
4. Optional 类型作为方法参数时

isPresent() 与 obj != null 无任何分别, 我们的生活依然在步步惊心. 而没有 isPresent() 作铺垫的 get() 调用在 IntelliJ IDEA 中会收到告警

Reports calls to java.util.Optional.get() without first checking with a isPresent() call if a value is available. If the Optional does not contain a value, get() will throw an exception. (调用 Optiona .get() 前不事先用 isPresent() 检查值是否可用. 假如 Optional 不包含一个值, get() 将会抛出一个异常)

把 Optional 类型用作属性或是方法参数在 IntelliJ IDEA 中更是强力不推荐的

Reports any uses of java.util.Optional, java.util.OptionalDouble, java.util.OptionalInt, java.util. ptionalLong or com.google.common.base.Optional as the type for a field or a parameter. Opt onal was designed to provide a limited mechanism for library method return types where the

e needed to be a clear way to represent “no result”. Using a field with type java.util.Optional is also problematic if the class needs to be Serializable, which java.util.Optional is not. (使用何像 Optional 的类型作为字段或方法参数都是不可取的. Optional 只设计为类库方法的, 可明确表示能无值情况下的返回类型. Optional 类型不可被序列化, 用作字段类型会出问题的)

所以 Optional 中我们真正可依赖的应该是除了 isPresent() 和 get() 的其他方法:

1. public Optional map(Function mapper)
2. public T orElse(T other)
3. public T orElseGet(Supplier other)
4. public void ifPresent(Consumer consumer)
5. public Optional filter(Predicate predicate)
6. public Optional flatMap(Function> mapper)
7. public T orElseThrow(Supplier exceptionSupplier) throws X

我略有自信的按照它们大概使用频度对上面的方法排了一下序.

先又不得不提一下 Optional 的三种构造方式: Optional.of(obj), Optional.ofNullable(obj) 和明确的 ptional.empty()

Optional.of(obj): 它要求传入的 obj 不能是 null 值的, 否则还没开始进入角色就倒在了 NullPointerException 异常上了.

Optional.ofNullable(obj): 它以一种智能的, 宽容的方式来构造一个 Optional 实例. 来者不拒, 传 null 进到就得到 Optional.empty(), 非 null 就调用 Optional.of(obj).

那是不是我们只要用 Optional.ofNullable(obj) 一劳永逸, 以不变应二变的方式来构造 Optional 实就行了呢? 那也未必, 否则 Optional.of(obj) 何必如此暴露呢, 私有则可?

我本人的观点是: 1. 当我们非常非常的明确将要传给 Optional.f(obj) 的 obj 参数不可能为 null 时, 比如它是一个刚 new 出来的对象(Optional.of(new User(...))), 者是一个非 null 常量时; 2. 当想为 obj 断言不 null 时, 即我们想在万一 obj 为 null 立即报告 NullPointException 异常, 立即修改, 而不是隐藏空指异常时, 我们就应该果断的用 Optional.of(obj) 来构造 Optional 实例, 而不让任何不可预计的 null 值可乘之机隐身于 Optional 中.

现在才开始怎么去使用一个已有的 Optional 实例, 假定我们有一个实例 Optional user, 下面是几个遍的, 应避免 if(user.isPresent()) { ... } else { ... } 几中应用方式.

存在即返回, 无则提供默认值

return user.orElse(null); //而不是 return user.isPresent() ? user.get() : null;

return user.orElse(UNKNOWN_USER);

存在即返回, 无则由函数来产生

return user.orElseGet(() -> fetchAUserFromDatabase()); //而不要 return user.isPresent() ? user: etchAUserFromDatabase();

存在才对它做点什么

```
user.ifPresent(System.out::println);

//而不要下边那样

if (user.isPresent()) {

System.out.println(user.get());

}
```

map 函数隆重登场

当 user.isPresent() 为真, 获得它关联的 orders, 为假则返回一个空集合时, 我们用上面的 orElse, orElsGet 方法都乏力时, 那原本就是 map 函数的责任, 我们可以这样一行

```
return user.map(u -> u.getOrders()).orElse(Collections.emptyList())

//上面避免了我们类似 Java 8 之前的做法

if (user.isPresent()) {

return user.get().getOrders();

} else {

return Collections.emptyList();

}
```

map 是可能无限级联的, 比如再深一层, 获得用户名的大写形式

```
return user.map(u -> u.getUsername())

.map(name -> name.toUpperCase())

.orElse( null );
```

这要搁在以前, 每一级调用的展开都需要放一个 null 值的判断

```
User user = .....

if (user !=null) {

String name = user.getUsername();

if (name !=null) {

return name.toUpperCase();

} else {

return null ;
```

```
}

} else {

return null ;

}
```

针对这方面 Groovy 提供了一种安全的属性/方法访问操作符 `?.

```
user?.getUsername()?.toUpperCase();
```

Swift 也有类似的语法, 只作用在 Optional 的类型上.

用了 isPresent() 处理 NullPointerException 不叫优雅, 有了 orElse, orElseGet 等, 特别是 map 方才叫优雅.

其他几个, filter() 把不符合条件的值变为 empty(), flatMap() 总是与 map() 方法成对的, orElseThro() 在有值时直接返回, 无值时抛出想要的异常.

一句话小结: 使用 Optional 时尽量不直接调用 Optional.get() 方法, Optional.isPresent() 更应该被为一个私有方法, 应依赖于其他像 Optional.orElse(), Optional.orElseGet(), Optional.map() 等这样方法.

最后, 最好的理解 Java 8 Optional 的方法莫过于看它的源代码 java.util.Optional, 阅读了源代码才真真正正的让你解释起来最有底气, Optional 的方法中基本都是内部调用 isPresent() 判断, 真时处值, 假时什么也不做.

参考链接:

1. Java 8 Optional: How to Use it
2. Tired of Null Pointer Exceptions? Consider Using Java SE 8's Optional!