**Java8（5）：使用 Optional 处理 null**

[](https://www.jianshu.com/u/687448bb2477)

[MiZhou](https://www.jianshu.com/u/687448bb2477) 关注

2017.03.16 17:17\* 字数 1548 阅读 2944评论 3喜欢 8

写过 Java 程序的同学，一般都遇到过 NullPointerException :) —— 为了不抛出这个异常，我们便会写如下的代码：

User user = getUserById(id);

if (user != null) {

String username = user.getUsername();

System.out.println("Username is: " + username); *// 使用 username*

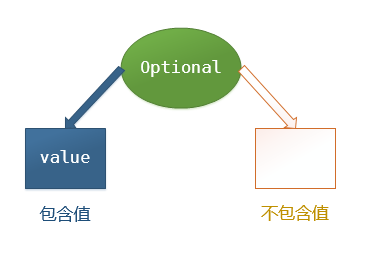
}

但是很多时候，我们可能会忘记写 if (user != null) —— 如果在开发阶段就发现那还好，但是如果在开发阶段没有测试到问题，等到上线却出了 NullPointerException ... 画面太美，我不敢继续想下去。

为了解决这种尴尬的处境，JDK 终于在 Java8 的时候加入了 Optional 类。Optional 的 javadoc 介绍：

A container object which may or may not contain a non-null value. If a value is present, isPresent() will return true and get() will return the value.

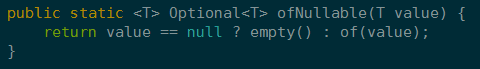
这是一个可以包含或者不包含非 null 值的容器。如果值存在则 isPresent()方法会返回 true，调用 get() 方法会返回该对象。



java.util.Optional<T>

JDK 提供三个静态方法来构造一个 Optional：  
1.Optional.of(T value)，该方法通过一个非 null 的 *value* 来构造一个 Optional，返回的 Optional 包含了 *value* 这个值。对于该方法，传入的参数一定不能为 null，否则便会抛出 NullPointerException。

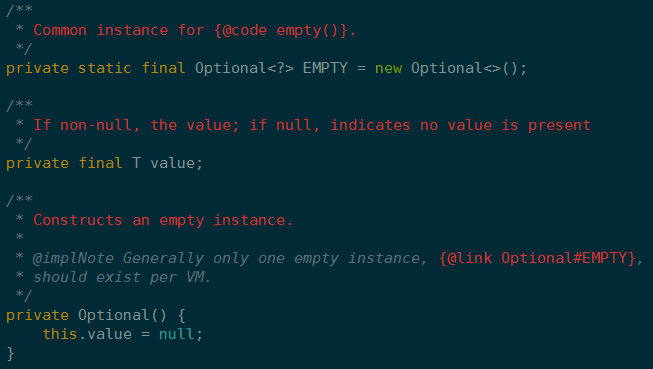
2.Optional.ofNullable(T value)，该方法和 of 方法的区别在于，传入的参数可以为 null—— 但是前面 javadoc 不是说 Optional 只能包含非 null 值吗？我们可以看看 ofNullable 方法的源码：



Optional.ofNullable 方法

原来该方法会判断传入的参数是否为 null，如果为 null 的话，返回的就是 Optional.empty()。

3.Optional.empty()，该方法用来构造一个空的 Optional，即该 Optional 中不包含值 —— 其实底层实现还是 **如果 Optional 中的 *value* 为 null 则该 Optional 为不包含值的状态**，然后在 API 层面将 Optional 表现的不能包含 null 值，使得 Optional 只存在 **包含值** 和 **不包含值** 两种状态。



Optional.empty() 的实现

前面 javadoc 也有提到，Optional 的 isPresent() 方法用来判断是否包含值，get() 用来获取 Optional 包含的值 —— 值得注意的是，如果值不存在，即在一个Optional.empty上调用 get() 方法的话，将会抛出 NoSuchElementException 异常。  
我们假设 getUserById 已经是个客观存在的不能改变的方法，那么利用 isPresent 和 get两个方法，我们现在能写出下面的代码：

Optional<User> user = Optional.ofNullable(getUserById(id));

if (user.isPresent()) {

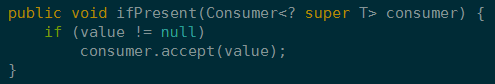
String username = user.get().getUsername();

System.out.println("Username is: " + username); *// 使用 username*

}

好像看着代码是优美了点 —— 但是事实上这与之前判断 null 值的代码没有本质的区别，反而用 Optional 去封装 *value*，增加了代码量。所以我们来看看 Optional 还提供了哪些方法，让我们更好的（以正确的姿势）使用 Optional。

1.ifPresent



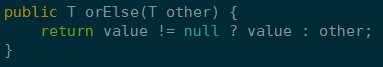
Optional.ifPresent

如果 Optional 中有值，则对该值调用 consumer.accept，否则什么也不做。  
所以对于上面的例子，我们可以修改为：

Optional<User> user = Optional.ofNullable(getUserById(id));

user.ifPresent(u -> System.out.println("Username is: " + u.getUsername()));

2.orElse



Optional.orElse

如果 Optional 中有值则将其返回，否则返回 orElse 方法传入的参数。

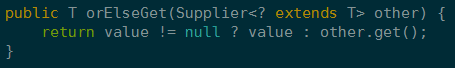
User user = Optional

.ofNullable(getUserById(id))

.orElse(new User(0, "Unknown"));

System.out.println("Username is: " + user.getUsername());

3.orElseGet



Optional.orElseGet

orElseGet 与 orElse 方法的区别在于，orElseGet 方法传入的参数为一个 Supplier 接口的实现 —— 当 Optional 中有值的时候，返回值；当 Optional 中没有值的时候，返回从该 Supplier 获得的值。

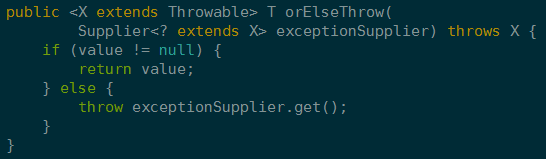
User user = Optional

.ofNullable(getUserById(id))

.orElseGet(() -> new User(0, "Unknown"));

System.out.println("Username is: " + user.getUsername());

4.orElseThrow



Optional.orElseThrow

orElseThrow 与 orElse 方法的区别在于，orElseThrow 方法当 Optional 中有值的时候，返回值；没有值的时候会抛出异常，抛出的异常由传入的 *exceptionSupplier* 提供。

User user = Optional

.ofNullable(getUserById(id))

.orElseThrow(() -> new EntityNotFoundException("id 为 " + id + " 的用户没有找到"));

举一个 orElseThrow 的用途：在 SpringMVC 的控制器中，我们可以配置统一处理各种异常。查询某个实体时，如果数据库中有对应的记录便返回该记录，否则就可以抛出 EntityNotFoundException ，处理 EntityNotFoundException 的方法中我们就给客户端返回Http 状态码 404 和异常对应的信息 —— orElseThrow 完美的适用于这种场景。

@RequestMapping("/{id}")

public User getUser(@PathVariable Integer id) {

Optional<User> user = userService.getUserById(id);

return user.orElseThrow(() -> new EntityNotFoundException("id 为 " + id + " 的用户不存在"));

}

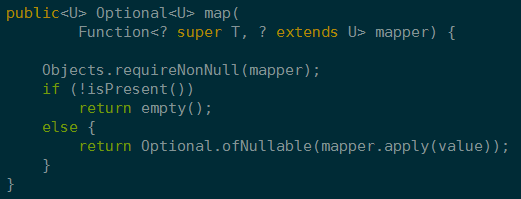
@ExceptionHandler(EntityNotFoundException.class)

public ResponseEntity<String> handleException(EntityNotFoundException ex) {

return new ResponseEntity<>(ex.getMessage(), HttpStatus.NOT\_FOUND);

}

5.map



Optional.map

如果当前 Optional 为 Optional.empty，则依旧返回 Optional.empty；否则返回一个新的 Optional，该 Optional 包含的是：函数 *mapper* 在以 *value* 作为输入时的输出值。

Optional<String> username = Optional

.ofNullable(getUserById(id))

.map(user -> user.getUsername());

System.out.println("Username is: " + username.orElse("Unknown"));

而且我们可以多次使用 map 操作：

Optional<String> username = Optional

.ofNullable(getUserById(id))

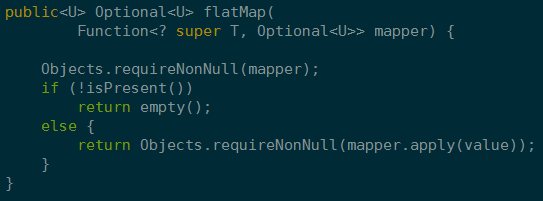
.map(user -> user.getUsername())

.map(name -> name.toLowerCase())

.map(name -> name.replace('\_', ' '));

System.out.println("Username is: " + username.orElse("Unknown"));

6.flatMap



Optional.flatMap

flatMap 方法与 map 方法的区别在于，map 方法参数中的函数 mapper 输出的是值，然后 map 方法会使用 Optional.ofNullable 将其包装为 Optional；而 flatMap 要求参数中的函数 mapper 输出的就是 Optional。

Optional<String> username = Optional

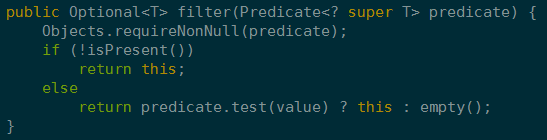
.ofNullable(getUserById(id))

.flatMap(user -> Optional.of(user.getUsername()))

.flatMap(name -> Optional.of(name.toLowerCase()));

System.out.println("Username is: " + username.orElse("Unknown"));

7.filter



Optional.filter

filter 方法接受一个 Predicate 来对 Optional 中包含的值进行过滤，如果包含的值满足条件，那么还是返回这个 Optional；否则返回 Optional.empty。

Optional<String> username = Optional

.ofNullable(getUserById(id))

.filter(user -> user.getId() < 10)

.map(user -> user.getUsername());

System.out.println("Username is: " + username.orElse("Unknown"));

有了 Optional，我们便可以方便且优雅的在自己的代码中处理 null 值，而不再需要一昧通过容易忘记和麻烦的 if (object != null) 来判断值不为 null。如果你的程序还在使用 Java8 之前的 JDK，可以考虑引入 Google 的 Guava 库 —— 事实上，早在 Java6 的年代，Guava 就提供了 Optional 的实现。

**号外：Java9 对 Optional 的增强**  
即将在今年 7 月到来的 JDK9 中，在 Optional 类中添加了三个新的方法：

1. public Optional<T> or(Supplier<? extends Optional<? extends T>> supplier)  
   or 方法的作用是，如果一个 Optional 包含值，则返回自己；否则返回由参数 *supplier* 获得的 Optional
2. public void ifPresentOrElse(Consumer<? super T> action, Runnable emptyAction)  
   ifPresentOrElse 方法的用途是，如果一个 Optional 包含值，则对其包含的值调用函数 *action*，即 action.accept(value)，这与 ifPresent 一致；与 ifPresent 方法的区别在于，ifPresentOrElse 还有第二个参数 emptyAction —— 如果 Optional 不包含值，那么 ifPresentOrElse 便会调用 emptyAction，即 emptyAction.run()
3. public Stream<T> stream()  
   stream 方法的作用就是将 Optional 转为一个 Stream，如果该 Optional 中包含值，那么就返回包含这个值的 Stream；否则返回一个空的 Stream（Stream.empty()）。  
   举个例子，在 Java8，我们会写下面的代码：

*// 此处 getUserById 返回的是 Optional<User>*

public List<User> getUsers(Collection<Integer> userIds) {

return userIds.stream()

.map(this::getUserById) *// 获得 Stream<Optional<User>>*

.filter(Optional::isPresent)*// 去掉不包含值的 Optional*

.map(Optional::get)

.collect(Collectors.toList());

}

而有了 Optional.stream()，我们就可以将其简化为：

public List<User> getUsers(Collection<Integer> userIds) {

return userIds.stream()

.map(this::getUserById) *// 获得 Stream<Optional<User>>*

.flatMap(Optional::stream) *// Stream 的 flatMap 方法将多个流合成一个流*

.collect(Collectors.toList());

}