还在用"a!=null"判空吗？老掉牙了，尝试下新写法...

程序员阿晨 程序员阿晨 2023-04-12 09:03 Posted on 江苏

NullPointerException出现原因：

如果一个对象为空，但是此时我们调用它的方法，就会遇到NullPointerException问题

我们定义Passenger、Address类

```
class Passenger{
    private String name;
    private String phone;
    private Address address;
}
```

```
class Address{
    private String province;
    private String city;
}
```

然后执行如下方法：

```
public static void main(String[] args) {
        Passenger passenger = new Passenger();
        passenger.getAddress().getCity();
}
```

很显然，此时，我们没有给address赋值，却调用它的方法，会抛出NullPointerException

```
Exception in thread "main" java.lang.NullPointerException
at com.study.nulljudge.Test.main(Test.java:7)
```

平时，我们会这样进行 NullPointerException异常的避免：

```
if (passenger.getAddress() != null){
    passenger.getAddress().getCity();
}
```

或者：

```
if (null != passenger.getAddress()) {
    passenger.getAddress().getCity();
}
```

至于上面的null是写在==前面，还是后面，对于Java语言来说是没有什么区别的，但是Java规范**建议**null写在==前面。

而对于其它语言，例如C语言，由于C允许if(v1=null)这种判断，其实这里漏写个=号，但是程序不会报错，会导致一些问题。因此C语言为了避免漏写=，因此要求null写在==号前面，即if(null==v1)，这样即使漏写=号，即写成if(null=v1)，编译器就会编译不通过，通知程序员代码有误，可以及时更改。

上述已经可以处理对象为空的情况，通过if(null != passenger.getAddress()){...}，但是这种写法不够优雅，Java8提供Optional类要优化这种写法，具体如下：

一、获取Optional对象的方法

1.Optional构造方法

```
public final class Optional<T> {
private Optional() {
    this.value = null;
    }
private Optional(T value) {
    this.value = Objects.requireNonNull(value);
    }
}

```

可以看到，Optional被final所修饰，该类不允许被继承，并且构造方法修饰符private，不允许外部通过构造方法获取Optional实例。

Objects.requireNonNull(value)方法中value值为空，也会报NullPointerException

2.of(T vlaue)方法

那么如何获取Optional实例呢，通过调用of方法

```
public static <T> Optional<T> of(T value) {
    return new Optional<>(value);
}
```

- 可以看到of方法内部调用Optional有参构造方法，而Optional有参构造方法上面也讲了，如果传入的value为null，依旧会报NullPointerException
- 如果传入的value值不为空，会正常构造Optional对象

二、empty()方法

```
private static final Optional<?> EMPTY = new Optional<>();
private Optional() {this.value = null;}
public static<T> Optional<T> empty() {
    @SuppressWarnings("unchecked")
    Optional<T> t = (Optional<T>) EMPTY;
    return t;
}
```

empty方法也会返回一个Optional类型的对象EMPTY，但是对象的值为null

三、ofNullable(T value)

```
public static <T> Optional<T> ofNullable(T value) {
    return value == null ? empty() : of(value);
}
```

- 当传递进来的value为null时，返回一个EMPTY对象
- 当传递进来的value不为null，调用of方法构造对象

和普通of方法的区别在于：当value为空，ofNullable方法不报错，返回一个EMPTY对象，但是of方法报错

四、orElse，orElseGet，orElseThrow

```
public T orElse(T other) {
    return value != null ? value : other;
}
public T orElseGet(Supplier<? extends T> other) {
    return value != null ? value : other.get();
}
public <X extends Throwable> T orElseThrow(Supplier<? extends X> exceptionSup
    if (value != null) {
        return value;
    } else {
        throw exceptionSupplier.get();
    }
}
```

可以看到，orElse，orElseGet，orElseThrow当value为空时有效果

```
public<U> Optional<U> map(Function<? super T, ? extends U> mapper) {
    Objects.requireNonNull(mapper);
    if (!isPresent())
        return empty();
    else {
        return Optional.ofNullable(mapper.apply(value));
    }
}
```

这个函数的作用就是转换值的操作，例如：

```
String phone = Optional.ofNullable(passenger).map(p -> p.getPhone()).get();
```

五、flatMap(Function<? super T, Optional<U>> mapper)

```
public<U> Optional<U> flatMap(Function<? super T, Optional<U>> mapper) {
    Objects.requireNonNull(mapper);
    if (!isPresent())
        return empty();
    else {
        return Objects.requireNonNull(mapper.apply(value));
    }
}
```

如果Passenger的getPhone方法变为这样：

```
class Passenger{
    private String name;
    private String phone;
    private Address address;

    public Optional<String> getPhone(){
        return Optional.ofNullable(phone);
    }
}
```

此时可以使用flatMap进行值的转换

```
String phone = Optional.ofNullable(passenger).flatMap(p -> p.getPhone()).get()
```

六、isPresent()

isPresent()判断当前值是否为空

```
public boolean isPresent() {
    return value != null;
}
```

七、ifPresent(Consumer<? super T> consumer)

ifPresent就是在当前值不为空的情况下做一些操作

```
public void ifPresent(Consumer<? super T> consumer) {
    if (value != null)
        consumer.accept(value);
}
```

例如：

```
Optional.ofNullable(passenger).ifPresent(p -> {
    //做一些其它操作
});
```

八、filter(Predicate<? super T> predicate)

```
public Optional<T> filter(Predicate<? super T> predicate) {
    Objects.requireNonNull(predicate);
    if (!isPresent())
        return this;
    else
        return predicate.test(value) ? this : empty();
}
```

接受一个过滤条件，如果条件满足，返回原Optional对象，否则返回EMPTY对象

例如：

```
Optional.ofNullable(passenger).filter(p -> p.getPhone().length() == 12);
```

好了，上述就是Optional的具体用法了，那么如何替换掉 `if(p != null)`呢?

用法如下：

旧写法

```java
public String getProvince(Passenger p){
    if (p != null) {
        Address address = p.getAddress();
        if (address != null) {
            String province = address.getProvince();
            if (province != null) {
                return province;
            }
        }
    }
    throw new NullPointerException("无法找到！！！");
}
```

新写法

```java
public String getProvince(Passenger p){
    return Optional.ofNullable(p)
            .map(p -> p.getAddress())
            .map(address -> address.getProvince())
            .orElseThrow(() -> new NullPointerException("无法找到！！！"));
}
```

虽然使用链式表达式，代码看起来更优雅了，但是可读性降低了，大家可以根据需要自行选择哪种判空方式。