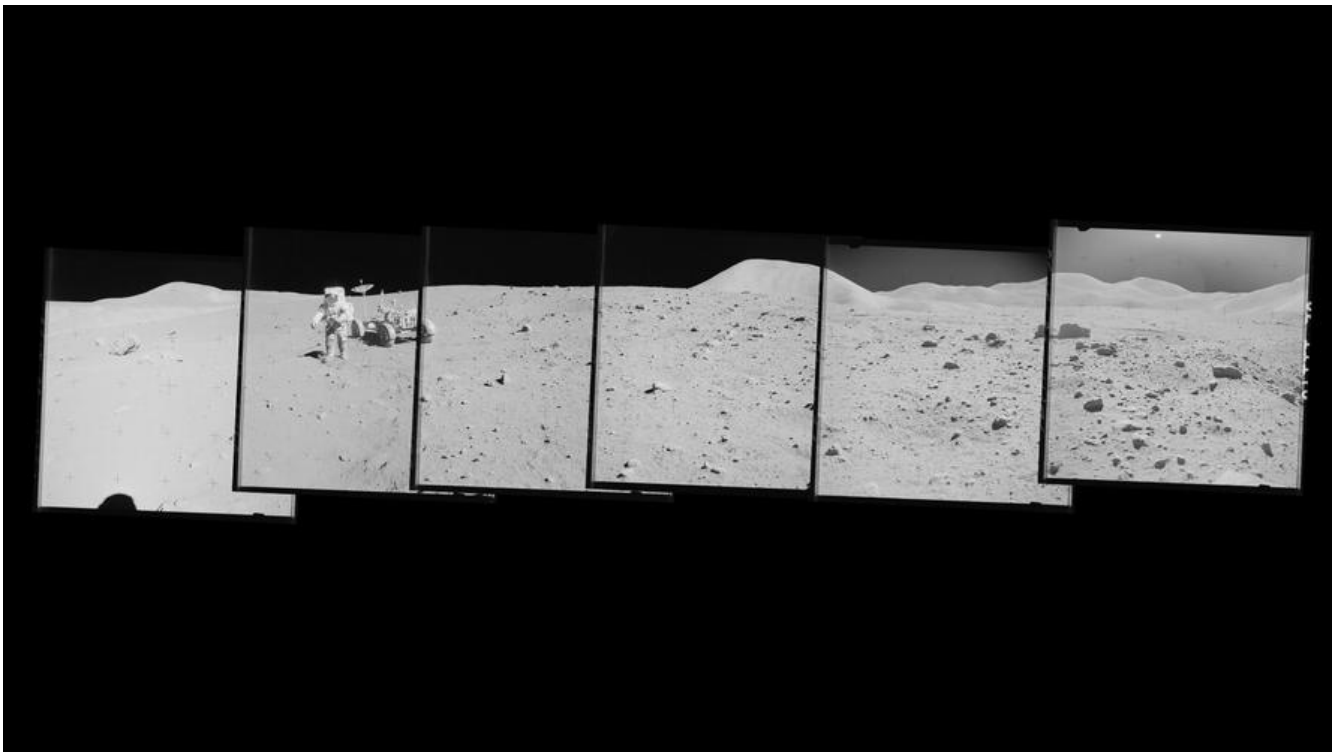
链滴

java8 系列 -05 Optional

作者：pleaseok
原文链接：https://ld246.com/article/1552123280991
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)



空指针异常我想是程序员见到比较多的异常啦，今天来探索的这个Optional工具类就是来处理这类问的。用过Google Guava（后期应该会出一个系列，请持续关注）的都应该知道，Guava这套工具类里面也有一个Optional，而且作用和今天来探索的这个java新特性是一样的。既然又是探索，那我们打开idea，然后ctrl+鼠标左键进到optional的世界看一下吧。

源码片段

接下来我将逐行解释一下源码(像我这样给jdk源码这样加注释的怕是很少了^_^)

```
// final修饰说明此类是无法被引用的哈
public final class Optional<T> {
    // 这个是EMPTY的实例
    private static final Optional<?> EMPTY = new Optional<>();

    // 如果非空，则为该值；如果为空，则表示不存在任何值。
    private final T value;

    // 构造私有看来此类也不能被new
    private Optional() {
        this.value = null;
    }

    // 返回一个空的实例，@SuppressWarnings不提醒警告
    public static<T> Optional<T> empty() {
        @SuppressWarnings("unchecked")
        Optional<T> t = (Optional<T>) EMPTY;
        return t;
    }

    // 根据value来构造出一个实例
    private Optional(T value) {
```

```
    this.value = Objects.requireNonNull(value);
}

// 返回一个带有value的optional，注意value如果为空则会报空指针异常
public static <T> Optional<T> of(T value) {
    return new Optional<>(value);
}

// 和上面一样，但是这里value为空不会有空指针异常
public static <T> Optional<T> ofNullable(T value) {
    return value == null ? empty() : of(value);
}

// 如果当前有值则返回当前值，否则抛出NoSuchElementException
public T get() {
    if (value == null) {
        throw new NoSuchElementException("No value present");
    }
    return value;
}

// 如果这个Optional是存在的则返回true，否则为false
public boolean isPresent() {
    return value != null;
}

// 如果option对象保存的值不是null，则调用consumer对象，否则不调用
public void ifPresent(Consumer<? super T> consumer) {
    if (value != null)
        consumer.accept(value);
}

// 判断Optional对象中保存的值是否满足Predicate，并返回新的Optional。
public Optional<T> filter(Predicate<? super T> predicate) {
    Objects.requireNonNull(predicate);
    if (!isPresent())
        return this;
    else
        return predicate.test(value) ? this : empty();
}

// 如果value不为null，则进行函数运算，返回一个新的Optional，否则返回empty
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));
    }
}

// 上一章讲到过Stream里面的flatMap,就是对一个大的集合进行一个扁平化的操作。
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));
        }
    }

    // 不解释
    public T orElse(T other) {
        return value != null ? value : other;
    }

    // 怎么办？也不想解释，关于Supplier供给者函数式的作用麻烦翻到此系列的第二章
    public T orElseGet(Supplier<? extends T> other) {
        return value != null ? value : other.get();
    }

    public <X extends Throwable> T orElseThrow(Supplier<? extends X> exceptionSupplier) th
ows X {
        if (value != null) {
            return value;
        } else {
            throw exceptionSupplier.get();
        }
    }

    // 重写了Object的方法
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }

        if (!(obj instanceof Optional)) {
            return false;
        }

        Optional<?> other = (Optional<?>) obj;
        return Objects.equals(value, other.value);
    }

    // 重写了Object的方法
    @Override
    public int hashCode() {
        return Objects.hashCode(value);
    }

    // 如果值不为空则返回Optional[value],否则Optional.empty。认真看过我上一章的也许还能记忆
来
    @Override
    public String toString() {
        return value != null
            ? String.format("Optional[%s]", value)
            : "Optional.empty";
```

```
    }
}
```

优雅食用

这里捡几个比较常用的在这里列出来吧

of && ofNullable

上面注释有提到，这两个都是被用来创建Optional对象。区别在于ofNullable还判空了。这就造成了oNullable的实用性比of要高那么一丢丢。下面来看一个例子吧~

```
public class OptionalDemo {
    public static void main(String[] args) {
        Optional<String> ofNullable = Optional.ofNullable(null);
        System.out.println(ofNullable);// Optional.empty

        Optional<String> of = Optional.of(null); // 代码执行到这里会抛出NullPointerException
    }
}
```

isPresent

isPresent有两种作用：一个是判断是否有值，还有一个返回的是void，如果不为空的话执行Consume消费型函数式代码。

```
public class OptionalDemo {
    public static void main(String[] args) {
        // 先创建一个Optional
        Optional<String> val = Optional.ofNullable("yes");
        System.out.println(val.isPresent()); // true

        val.ifPresent(s -> System.out.println(s.concat(" or no?"))); // yes or no?
    }
}
```

filter && map

直接看例子最好啦

```
public class OptionalDemo {
    public static void main(String[] args) {
        Optional<String> optional = Optional.ofNullable("code666");
        Optional<String> newOptional = optional.map(param->param.concat(".top"));
        System.out.println(newOptional); // Optional[code666.top]

        Optional filterOptional = optional.filter(param->param.contentEquals("Sean")); // 如果
件成立的话会返回Optional[code666]
        System.out.println(filterOptional); // Optional.empty
    }
}
```

orElse && orElseGet

长得虽然差不多，orElseGet还带了一个Supplier供给者函数看着高大上一点，其实也差不多。记得第二章有说过，供给者函数就有点像工厂的味道而已。

```
public class OptionalDemo {
    public static void main(String[] args) {
        String res = (String) Optional.ofNullable(null).orElse("这个结果为空");
        System.out.println(res); // 这个结果为空

        String res2 = (String) Optional.ofNullable(null).orElseGet(()-> "The result is empty.");
        System.out.println(res2); // The result is empty.
    }
}
```