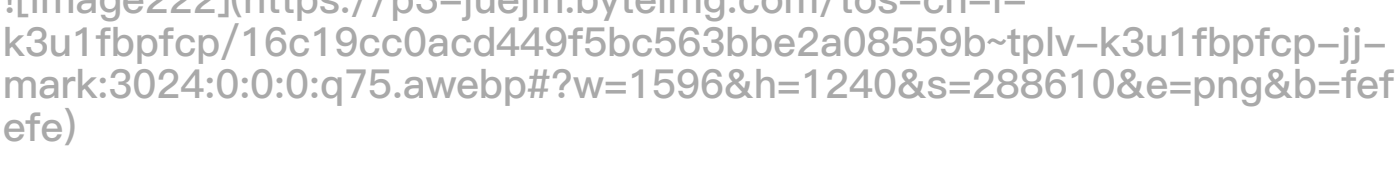
Java学习十二—Java8特性之Optional类

一、简介

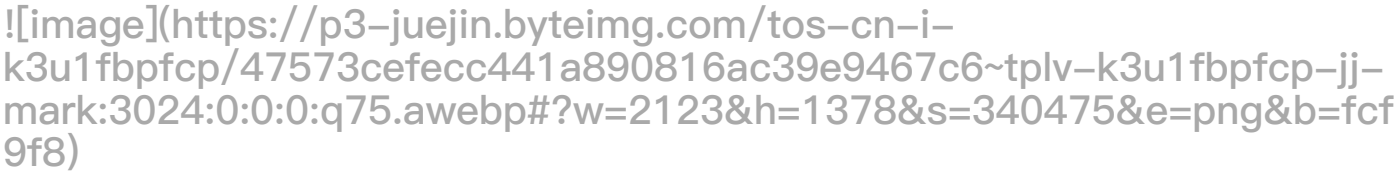
Java 8 引入了 `Optional` 类作为一种容器，可以用来显式地表示一个值存在或不存在。它解决了传统上可能会遇到的空指针异常问题，同时提供了一种更优雅的方式来处理可能为null的情况。



> Java 8 中引入 `Optional` 类的背景可以从以下几个方面来理解：
>
>
> 0. **空指针异常问题：** 在 Java 中，空指针异常（NullPointerException）是一个常见的问题，特别是当处理可能为null的对象时。这种异常可能会在运行时导致程序崩溃，难以调试和处理。
> 1. **编程语言发展趋势：** 在 Java 8 发布之前，其他一些现代编程语言（如Scala、Swift等）已经引入了类似的 Option 类型或者可空类型（Nullable Type），这些类型可以更安全和清晰地处理可能为空的值。
> 2. **函数式编程的影响：** Java 8 引入了函数式编程的元素，例如 lambda 表达式和流式 API。在函数式编程中，避免副作用和空值处理是重要的理念之一，因此需要一种适合函数式编程风格的空值处理机制。
> 3. **更优雅的代码风格：** 使用 `Optional` 类可以使代码更加清晰和优雅。通过强制显式处理可能为null的情况，可以减少条件判断和嵌套，提高代码的可读性和可维护性。
> 4. **API 设计的进步：** 引入 `Optional` 类使得 Java 标准库的 API 设计更加完善和一致。在一些情况下，例如集合操作中的元素查找或者返回值可能为空的方法，使用 `Optional` 可以更准确地表达返回值的可能性。
>
>
> 综上所述，Java 8 引入 `Optional` 类旨在提供一种更加安全、清晰和优雅的方式来处理可能为null的值，以及促进函数式编程风格在 Java 中的应用。

二、类申明
=====

```
public final class Optional<T>
```



三、方法
====

以下是 `Optional` 类的一些常用方法：

0. **`Optional.of(T value)`** – 创建一个 `Optional` 实例，其非空值由给定的参数指定。如果给定的参数为 `null`，则抛出 `NullPointerException`。
1. **`Optional.ofNullable(T value)`** – 创建一个 `Optional` 实例，其值可以是 `null`。如果给定的参数为 `null`，则返回一个空的 `Optional` 对象。
2. **`Optional.empty()`** – 返回一个空的 `Optional` 实例。
3. **`Optional.isPresent()`** – 检查 `Optional` 是否包含非空值。如果包含非空值，则返回 `true`，否则返回 `false`。
4. **`Optional.ifPresent(Consumer<? super T> consumer)`** – 如果 `Optional` 包含非空值，则将该值传递给提供的 `Consumer` 对象。如果 `Optional` 是空的，则不执行任何操作。
5. **`Optional.orElse(T other)`** – 如果 `Optional` 包含非空值，则返回该值；否则返回提供的其他值。

6. **`Optional.orElseGet(Supplier<? extends T> other)`** – 如果 `Optional` 包含非空值，则返回该值；否则返回由 `Supplier` 调用返回的值。
7. **`Optional.orElseThrow()`** – 如果 `Optional` 包含非空值，则返回该值；否则抛出 `NoSuchElementException`。
8. **`Optional.map(Function<? super T,? extends U> mapper)`** – 如果 `Optional` 包含非空值，则将该值传递给提供的 `Function` 对象，并返回结果包装在 `Optional` 中。如果 `Optional` 是空的，则返回空的 `Optional`。
9. **`Optional.flatMap(Function<? super T, Optional<? extends U>> mapper)`** – 如果 `Optional` 包含非空值，则将该值传递给提供的 `Function` 对象，并返回 `Function` 返回的 `Optional`。如果 `Optional` 是空的，或者 `Function` 返回空的 `Optional`，则返回空的 `Optional`。
10. **`Optional.filter(Predicate<? super T> predicate)`** – 如果 `Optional` 包含非空值，并且该值满足提供的 `Predicate`，则返回包含该值的 `Optional`。如果 `Optional` 是空的，或者非空值不满足 `Predicate`，则返回空的 `Optional`。

四、示例

4.1示例1

```
Optional<String> optional = Optional.of(”bam”);

optional.isPresent();           // true
optional.get();                 // ”bam”
optional.orElse(”fallback”);    // ”bam”

optional.ifPresent((s) -> System.out.println(s.charAt(0)));     // ”b”
```

4.2示例2

创建 Optional 对象：

* **`of(T value)`** **：** 创建一个包含指定非null值的 Optional。

```
Optional<String> optional = Optional.of(”Hello”);
```

* **`ofNullable(T value)`** **：** 创建一个可能为null的 Optional 对象。

```
String str = null;
Optional<String> optional = Optional.ofNullable(str);
```

* **`empty()`** **：** 创建一个空的 Optional 对象。

```
Optional<String> optional = Optional.empty();
```

判断值是否存在：

* **`isPresent()`** **：** 如果存在值，则返回true，否则返回false。

```
Optional<String> optional = Optional.of(”Hello”);
if (optional.isPresent()) {
    // 值存在时的操作
    System.out.println(”Value is present: ” + optional.get());
} else {
```

```
    // 值不存在时的操作
    System.out.println("Value is absent.");
}
```

获取值或默认值：

* **`get()`** **：** 如果存在值，则返回该值，否则抛出`NoSuchElementException`。

```
Optional<String> optional = Optional.of("Hello");
String value = optional.get();
```

* **`orElse(T other)`** **：** 如果存在值，则返回该值，否则返回指定的默认值。

```
Optional<String> optional = Optional.ofNullable(null);
String result = optional.orElse("Default Value");
```

* **`orElseGet(Supplier<? extends T> other)`** **：** 如果存在值，则返回该值，否则使用提供的 Supplier 生成一个默认值。

```
Optional<String> optional = Optional.ofNullable(null);
String result = optional.orElseGet(() -> "Default Value");
```

条件式操作：

* **`filter(Predicate<? super T> predicate)`** **：** 如果存在值并且满足给定条件，则返回包含该值的 Optional，否则返回空的 Optional。

```
Optional<String> optional = Optional.of(”Hello”);
Optional<String> filtered = optional.filter(s -> s.startsWith(”H”));
```

* **`map(Function<? super T, ? extends U> mapper)`** **：** 如果存在值，则对其进行转换，并返回包含转换后值的 Optional，否则返回空的 Optional。

```
Optional<String> optional = Optional.of(”Hello”);
Optional<String> upperCase = optional.map(String::toUpperCase);
```

* **`flatMap(Function<? super T, Optional<U>> mapper)`** **：** 如果存在值，则对其进行转换并返回结果，否则返回空的 Optional。

```
Optional<String> optional = Optional.of(”Hello”);
Optional<String> flatMapped = optional.flatMap(s -> Optional.of(s + ” World”));
```

条件式执行：

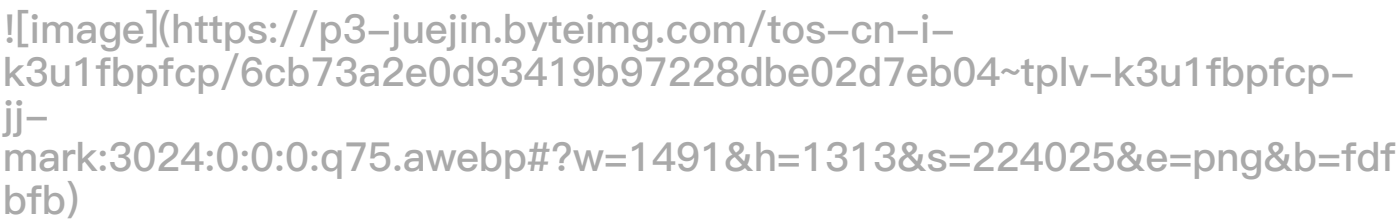
* **`ifPresent(Consumer<? super T> consumer)`** **：** 如果存在值，则执行指定的操作。

```
Optional<String> optional = Optional.of(”Hello”);
optional.ifPresent(s -> System.out.println(”Value is present: ” + s));
```

示例：

```
import java.util.Optional;

public class OptionalExample {
    public static void main(String[] args) {
        // 创建一个非空的 Optional
        Optional<String> optional1 = Optional.of(”Hello”);

        // 创建一个可能为 null 的 Optional
        String str = null;
        Optional<String> optional2 = Optional.ofNullable(str);

        // 判断值是否存在
        if (optional1.isPresent()) {
            System.out.println(”Value 1 is present: ” + optional1.get());
        } else {
            System.out.println(”Value 1 is absent.”);
        }

        // 使用 orElse 方法提供默认值
        String result = optional2.orElse(”Default Value”);
        System.out.println(”Value 2: ” + result);

        // 使用 map 对值进行转换
        Optional<String> upperCaseOptional =
optional1.map(String::toUpperCase);
        upperCaseOptional.ifPresent(s -> System.out.println(”Uppercase
value: ” + s));

        // 使用 flatMap 进行链式操作
        Optional<String> flatMapped = optional1.flatMap(s -> Optional.of(s
+ ” World”));
        flatMapped.ifPresent(s -> System.out.println(”FlatMapped value: ”
+ s));
```

```
    }
}
```



原文链接: https://juejin.cn/post/7387216861857792039