

泛型相關

Java 泛型是一种支持参数化类型的机制，它可以使代码更加通用和安全。通过使用泛型，我可以在代码中使用占位符类型（如 T、E 等），然后在实际使用时再指定具体类型。这样就可以使代码更加灵活，可以避免类型错误和运行异常。

Java 泛型的基本用法是使用尖括号 "<>" 括起的一个或多个类型参数，这些类型参数可以用于类、接口、方法的声明中。例如：

```
public class MyClass<T> {
    private T data;

    public MyClass(T data) {
        this.data = data;
    }

    public T getData() {
        return data;
    }
}
```

在上面的代码中，我使用了一个类型参数 T，这个参数可以用于 MyClass 类中的任何地方，包括类的字段、方法参数、方法返回值等等。

而<?>和<T>的区别在于，<?>是一种无限制通配符类型，表示可以匹配任何类型，而<T>是一个类型参数，表示在使用时需要指定具体的类型。例如：

```
List<?> list = new ArrayList<>();
List<String> strList = new ArrayList<>();
list = strList; // 合法，因为 list 可以匹配任何类型
```

```
public <T> T getValue(T[] array, int index) {
    return array[index];
}

String[] strArray = {"hello", "world"};
String str = getValue(strArray, 0); // 合法，因为在使用 getValue 指定了类型参数 String
```

下面是一个使用无限制通配符类型（<?>）的例子，我定义了一个方法，可以接受任何类型的 List，并打印出其中的元素：

```
public static void printList(List<?> list) {
    for (Object element : list) {
        System.out.println(element);
    }
}
```

这个方法参数列表中使用了无限制通配符类型（<?>），表示可以匹配任何类型的 List。在方法内部，我使用了一个 for-each 循环遍历 list，并打印出其中的元素。

使用这个方法，我可以传入任何类型的 List，例如：

```
List<Integer> intList = Arrays.asList(1, 2, 3);
List<String> strList = Arrays.asList("hello", "world");
List<Object> objList = Arrays.asList(1, "hello", true);

printList(intList); // 输出 1, 2, 3
printList(strList); // 输出 hello, world
printList(objList); // 输出 1, hello, true
```

如果我上面的例子中的方法参数从无限制通配符类型（<?>）改为类型参数（<T>），那么代码会变成下面这样：

```
public static <T> void printList(List<T> list) {
    for (T element : list) {
        System.out.println(element);
    }
}
```

这个方法参数列表中使用了类型参数（<T>），表示在使用时需要指定 List 中元素的具体类型。在方法内部，我使用了一个 for-each 循环

遍历 list，并打印出其中的元素。

使用这个方法，我需要指定 List 中元素的具体类型，例如：

```
List<Integer> intList = Arrays.asList(1, 2, 3);
List<String> strList = Arrays.asList("hello", "world");
List<Object> objList = Arrays.asList(1, "hello", true);

printList(intList); // 输出 1, 2, 3
printList(strList); // 输出 hello, world
printList(objList); // 报错，因为 objList 中的元素类型不是 Object
```

在使用时，我需要指定 List 中元素的具体类型，例如使用 `printList(Integer)` 或 `printList(String)`，这样才能保证参类型的匹配，否则会报错。

在 Java 泛型中，类型参的名是可以任意指定的，通常使用以下的约定：

- T：表示任何类型。
- E：表示集合中的元素类型。
- K：表示映射中的键类型。
- V：表示映射中的值类型。
- N：表示数字类型。
- S、U、V 等：表示其他任意类型。

因此，使用 `<T>` 或 `<E>` 在定义上是等价的，只是命名不同而已。

除了以上提到的常用类型参名外，开发者也可以根据实际情况自行定义类型参的名，只需要遵循命名规范即可。不过，为了代码可读性和可维护性，建议使用常见的类型参名。

需要注意的是，不同的类型参名并非有实质上的差异，只是在定义上有所不同，因此类型参名应根据实际情况和需求进行选择，选择一个能更好地表达自己的意图的命名。

一个回传任意型别的范例

```
public static <T> T anyTypeMethod(T arg) {
    if (arg == null) {
        // 返回空对象
        return (T) new Object();
    }
    // 方法体
    return arg;
}
```

在这个方法中，我首先检查参是否 `null`。如果参 `null`，返回一个空对象，否则执行方法体，参原封不动地返回。

需要注意的是，在这个方法中，我新建了一个新的 `Object` 对象并将其泛型类型，这是为了避免空对象返回类型不匹配的。因为 Java 中的泛型是在编译时擦除的，如果我直接返回一个 `null`，编译器无法确定返回值的类型，可能会导致编译错误或运行异常。

使用这个方法，我可以传入任何类型的参，并取返回值，如果传入参 `null`，返回一个空对象，例如：

```
Integer intValue = anyTypeMethod(123);
String strValue = anyTypeMethod("hello");
Object objValue = anyTypeMethod(new Object());
Object nullValue = anyTypeMethod(null);

System.out.println(intValue); // 输出 123
System.out.println(strValue); // 输出 "hello"
System.out.println(objValue); // 输出 Object 对象的 toString() 值
System.out.println(nullValue); // 输出 "java.lang.Object@hashCode"
```

在这个示例中，我分别传入了一个整数、一个字符串、一个对象和一个 `null`，然后取了相应的返回值。传入参 `null` 时，返回一个空对象，其类型为 `Object`。

🔄 修订版本 #3

★ 由 treeman 建立於 14 🕒 2023 09:54:49

✍ 由 treeman 更新於 5 🕒 2023 10:18:39