第2章 Java深入泛型与注解面试题汇总

摘要：本章内容，享学课堂在Java高级进阶核心中有系统化-全面完整的直播讲解，详情加微信：xxgfwx03

2.1 Java泛型的特点与优缺点，泛型擦除是怎么回事？~leo

这道题想考察什么？

主要考察对泛型的理解

考察的知识点

泛型的特点和优缺点以及泛型擦除

考生应该如何回答

泛型的特点

1. 泛型类型不能使用基本类型，只能使用对象类型
2. 获取泛型类的时候是获取的原生类型，与传入的泛型类型无关
3. 泛型不能用 instanceof

泛型的优点

其实就是我们为什么需要使用泛型。

1.适用于多种数据类型执行相同的代码，例如，两个数据相加

```
public int addInt(int x,int y){
    return x+y;
}

public float addFloat(float x,float y){
    return x+y;
}
```

不同的类型，我们就需要增加不同的方法实现，使用泛型，就会更简单，一个就行了

```
public <T> T addInt(T x,T y){
    return x+y;
}
```

2.编译时指定数据类型，例如下面代码

```
List list = new ArrayList();
list.add("Hello");
list.add(10); // 1

for (int i = 0; i < list.size(); i++) {
    // 2
    String name = (String) list.get(i);
    System.out.println("name:" + name);
}
```

代码1处，编译时不会报错，代码2处，需要做类型强转，使用泛型如下

```
List<String> list = new ArrayList();
list.add("Hello");
list.add(10); // 3

for (int i = 0; i < list.size(); i++) {
    // 4
    String name = list.get(i);
    System.out.println("name:" + name);
}
```

代码3处，编译时会报错，代码4处，不再需要做类型强转。

泛型的缺点

1. 静态域或者方法里不能引用泛型变量，因为泛型是在new对象的时候才知道，而类的构造方法是在静态变量之后执行。
2. 不能捕获泛型类对象

泛型擦除

jdk中实现的泛型实际上是伪泛型，例如泛型类 Fruit ，编译时 T 会被擦除，成为 Object。

2.2 List<String>能否转为List<Object> ~ leo

这道题想考察什么?

考察对泛型转换的理解

考察的知识点

泛型的转换

考生应该如何回答

List不能转为List，虽然 String 和 Object 有继承关系，但是 List 和 List 是没有继承关系的。

2.3 Java 的泛型，<? super T> 和 <? extends T> 的区别~leo

这道题想考察什么?

考察同学对泛型的通配符和边界是否熟悉

考察的知识点

通配符和边界

考生应该如何回答

为什么要有通配符

```
class Food {}

class Fruit extends Food{}

class Apple extends Fruit {}

class Plate<T>{
    private T item;
    public Plate(T t){item=t;}
    public void set(T t){item=t;}
    public T get(){return item;}
}
```

定义上述三个类，Apple 继承自 Fruit。案例如下代码应该是可以的，但是实际不行。所以就需要通配符和上下界来解决这个问题。

```
Plate<Fruit> p=new Plate<Apple>(new Apple());
```

extends

? extends Fruit 为通配符上界，也就是说传入的类型只能是Fruit 或者 Fruit的子孙类

```
Plate<? extends Fruit> p=new Plate<>(new Apple()); // ok

Plate<? extends Fruit> p=new Plate<>(new Food()); // 不行

Apple a = new Apple();
p.set(a); // 不行，因为这个只是规定了上界，如果Plate的泛型类是Apple的子类，这个时候你传入Apple对象，就是错的
p.get(a); // 也不行，如果Plate的泛型类是Fruit类，这个时候你传入Apple对象就错了

Fruit f = p.get(); // 可以，因为不管Plate的泛型类是哪个类，肯定是Fruit或者Fruit的子孙类，都属于Fruit
```

super

? super Fruit 为通配符下界，也就是说传入的类型只能是Fruit 或者 Fruit的父类

```
Plate<? super Fruit> p=new Plate<>(new Food()); // ok

Plate<? super Fruit> p=new Plate<>(new Apple()); // 不行

Fruit f = new Fruit();
p.set(f); // 可以，Food则不行，原理同上

Object o= p.get(); // 可以，其他的不行，原理同上
```

总结

其实不管是上界还是下界，set、get方法能传入什么类型，都是通过数据类型传入是否安全来判断的。

2.4 注解是什么？有哪些使用场景？（滴滴）~lance

这道题想考察什么？

Java基础，高级语言特性

考察的知识点

注解

考生如何回答

Java 注解（Annotation）又称 Java 标注，是 JDK5.0 引入的一种注释机制。 注解是元数据的一种形式，提供有关于程序但不属于程序本身的数据。注解本身没有特殊意义，对它们注解的代码的操作没有直接影响。

按照**@Retention** 元注解定义的注解保留级，注解可以一般常见于以下场景使用：

SOURCE

`RetentionPolicy.SOURCE` ，作用于源码级别的注解，在类中使用 `SOURCE` 级别的注解，其编译之后的class中会被丢弃。可提供给Lint 检查、APT等场景使用。

Lint

在Android开发中， `support-annotations` 与 `androidx.annotation` 中均有提供 `@IntDef` 注解，此注解的定义如下：

```
@Retention(SOURCE)  //源码级别注解
@Target({ANNOTATION_TYPE})
public @interface IntDef {
    int[] value() default {};

    boolean flag() default false;

    boolean open() default false;
}
```

> Java中Enum(枚举)的实质是特殊单例的静态成员变量，在运行期所有枚举类作为单例，全部加载到内存中。比常量多5到10倍的内存占用。

此注解的意义在于能够取代枚举，实现如方法入参限制。

如：我们定义方法 `test` ，此方法接收参数 `teacher` 需要在：**Lance**、**Alvin**中选择一个。如果使用枚举能够实现为:

```
public enum Teacher{
    LANCE,ALVIN
}

public void test(Teacher teacher) {

}
```

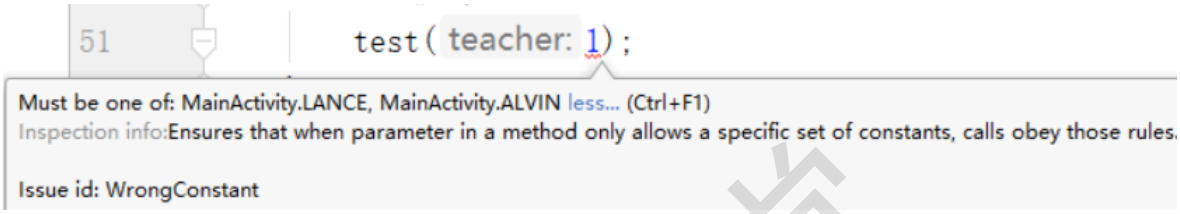
而现在为了进行内存优化，我们现在不再使用枚举，则方法定义为：

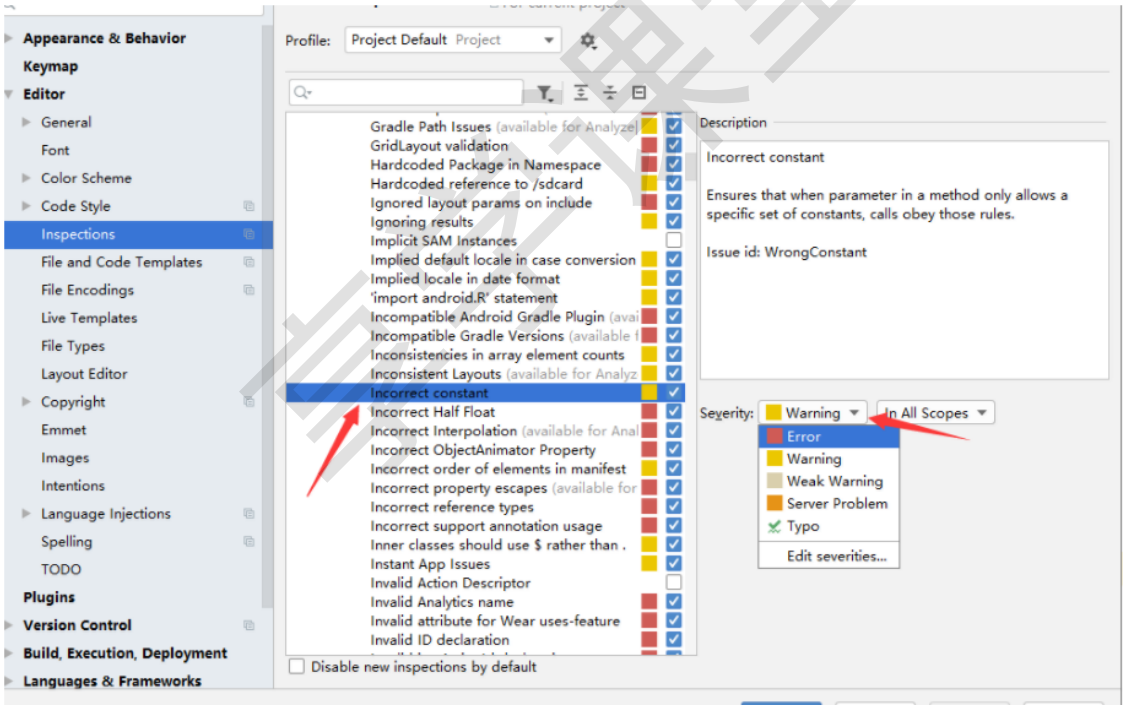
```
public static final int LANCE = 1;
public static final int ALVIN = 2;

public void test(int teacher) {

}
```

然而此时，调用 `test` 方法由于采用基本数据类型int，将无法进行类型限定。此时使用@IntDef增加自定义注解：

```
public static final int LANCE = 1;
public static final int ALVIN = 2;

@IntDef(value = {LANCE, ALVIN}) //限定为LANCE，ALVIN
@Target(ElementType.PARAMETER)  //作用于参数的注解
@Retention(RetentionPolicy.SOURCE) //源码级别注解
public @interface Teacher {
}

public void test(@Teacher int teacher) {

}
```

此时，我们再去调用`test`方法，如果传递的参数不是`LANCE`或者`ALVIN`则会显示 **Inspection** 警告(编译不会报错)。



> 可以修改此类语法检查级别：
>
> 
>

APT注解处理器

SOURCE另一种更常见的应用场景是结合APT使用。APT全称为："Anotation Processor Tools"，意为注解处理器。顾名思义，其用于处理注解。编写好的Java源文件，需要经过`javac`的编译，翻译为虚拟机能够加载解析的字节码Class文件。注解处理器是 javac 自带的一个工具，用来在编译时期扫描处理注解信息。你可以为某些注解注册自己的注解处理器。 注册的注解处理器由`javac`调起，并将注解信息传递给注解处理器进行处理。

> 注解处理器是对注解应用最为广泛的场景。在Glide、EventBus3、Butterknifer、Tinker、ARouter等等常用框架中都有注解处理器的身影。但是你可能会发现，这些框架中对注解的定义并不是`SOURCE`级别，更多的是`CLASS`级别，其实：**CLASS包含了SOURCE，RUNTIME包含SOURCE、CLASS。**所以CLASS是包含了SOURCE的场景，RUNTIME则包含了所有保留级的注解

> 使用场景

CLASS

定义为`CLASS`的注解，会保留在class文件中，但是会被虚拟机忽略(即无法在运行期反射获取注解)。此时完全符合此种注解的应用场景为字节码操作。如：AspectJ、热修复Roubust中应用此场景。

> 在Android开发中，保留在class，但是会在dex被抛弃

RUNTIME

注解保留至运行期，意味着我们能够在运行期间结合反射技术获取注解中的所有信息。