Java 泛型是 JDK5 引进的新特性，对于泛型的引入，社区褒贬不一，好的地方是泛型可以在编译期帮我们发现一些明显的问题，不好的地方是泛型在设计上因为考虑兼容性等原因，留下了比较大的坑。

网上有很多喷 Java 的泛型设计，甚至《Thinking in Java》的作者都发表了一篇文章来批评 JDK5 中的泛型实现，知乎也有很多类似的帖子。

Java 泛型更像是一个 Java 语言的语法糖，我们将从字节码的角度分析一下泛型。

0x01 当泛型遇到重载

```
public void print(List<String> list)  { }
public void print(List<Integer> list) { }
```

上面的代码编译的时候会报错，提示name clash: print(List<Integer>) and print(List<String>) have the same erasure

这两个函数对应的字节码都是

```
descriptor: (Ljava/util/List;)V
Code:
  stack=0, locals=2, args_size=2
     0: return
  LocalVariableTable:
    Start  Length  Slot  Name   Signature
        0       1     0  this   LMyClass;
        0       1     1  list   Ljava/util/List;
}
```

为了弄懂这个问题，需要先了解泛型的类型擦除

0x02 泛型的核心概念：类型擦除（type erasure）

理解泛型概念的最重要的是理解类型擦除。Java 的泛型是在 javac 编译期这个级别实现的。在生成的字节码中，已经不包含类型信息了。这种在泛型使用时加上类型参数，在编译时被抹掉的过程被称为泛型擦除。

比如在代码中定义：List<String> 与 List<Integer> 在编译以后都变成了 List。JVM 看到的只是 List，而 JVM 不允许相同签名的函数在一个类中同时存在，所以上面代码中编译无法通过。

由泛型附加的类型信息对 JVM 来说是不可见的。Java 编译器会在编译时尽可能的发现可能出错的地方，但是也不是万能的。

很多泛型的奇怪语法规定都与类型擦除的存在有关

- 泛型类并没有自己独有的 Class 类对象，比如并不存在 List<String>.class 或是 List<Integer>.class，而只有 List.class。
- 泛型的类型参数不能用在 Java 异常处理的 catch 语句中。因

> 为异常处理是由 JVM 在运行时刻来进行的。由于类型信息被擦除，JVM 是无法区分两个异常类型 MyException<String> 和 MyException<Integer>的。对于 JVM 来说，它们都是 MyException 类型的

0x03 泛型真的被完全擦除了吗

学习泛型的时候，我们被大量的文章警示「泛型信息在编译之后是拿不到的，因为已经被擦除掉」，真的是这样吗?

我们在 javac 编译的时候加上 -g 参数生成更多的调试信息，使用 javap -c -v -l 来查看字节码时可以看到更多有用的信息

```
  public void
print(java.util.List<java.lang.String>);
    descriptor: (Ljava/util/List;)V
      stack=0, locals=2, args_size=2
         0: return
      LocalVariableTypeTable:
        Start  Length  Slot  Name   Signature
            0       1     1  list
Ljava/util/List<Ljava/lang/String;>;
    Signature: #18                          //
(Ljava/util/List<Ljava/lang/String;>;)V
```

LocalVariableTypeTable 和 Signature 是针对泛型引入的新的属性，用来解决泛型的参数类型识别问题，Signature 最为重要，它的作用是存储一个方法在字节码层面的特征签名，这个属性保存的不是原生类型，而是包括了参数化类型的信息。我们依然可以通过反射的方式拿到参数的类型。所谓的擦除，只是把方法 code 属性的字节码进行擦除。

0x04 小结

这篇文章我们讲解了字节码在 Java 泛型上的应用，一起来回顾一下要点：第一，由于类型擦除的存在，List<String>.class、List<Integer>.class在 JVM 层面只有 List.class，因此泛型在重载上有一些问题。第二，通过 javap 可以看到泛型的类型擦除并不是完全擦除了，字节码中 Signature 域存储了方法带有泛型的签名。

0x05 思考

留一道作业题：下面的代码，你可以看出为什么 Java 编译器会提示编译错误吗?

```
public void inspect(List<Object> list) {
}
public void test() {
    List<String> strs = new ArrayList<String>();
    inspect(strs); // 编译错误
}
```

欢迎你在留言区留言，和我一起讨论。