@FunctionInterface

一个概念

用于函数式接口类型声明的信息注解类型，这些接口的实例被 Lambda 表示式、方法引用或构造器引用创建。函数式接口只能有一个抽象方法，并排除接口默认方法以及声明中覆盖 Object 的公开方法的统计。同时，@FunctionalInterface 不能标注在注解、类以及枚举上。如果违背以上规则，那么接口不能视为函数式接口，当标注@FunctionalInterface 后，会引起编译错误。不过，如果任一接口满足以上函数式接口的要求，无论接口声明中是否标注@FunctionalInterface，均能被编译器视作函数式接口。

函数式接口的体现：

提供类型：Supplier<T>
消费类型：Consumer<T>
转换类型：Function<T, R>
判定类型：Predicate<T>
隐藏类型：Action

理解一个匿名内部类和lambda的一个小原理

```
public static void main(String[] args) {
    /**
     * 比较匿名内部类和lambda的一个方式
     */

    // 通过编译之后的字节码文件可以看到是创建了一个内部类 匿名内部类 ActionDemo$1.class
    Runnable r1 = new Runnable() {
        @Override
        public void run() {
            System.out.println("111111");
        }
    };
    // 通过编译之后的class文件 可以看到lambda并没有像匿名内部类那样创建一个新的类
    // 而是使用了 InvokeDynamic 指令 通过字节码提升实现了lambda

    // 具体的可以见  java.lang.invoke.MethodHandle  java.lang.invoke.InvokeDynamic
    Runnable r2 = () -> System.out.println("22222");
}
```

几个经典的函数式接口设计

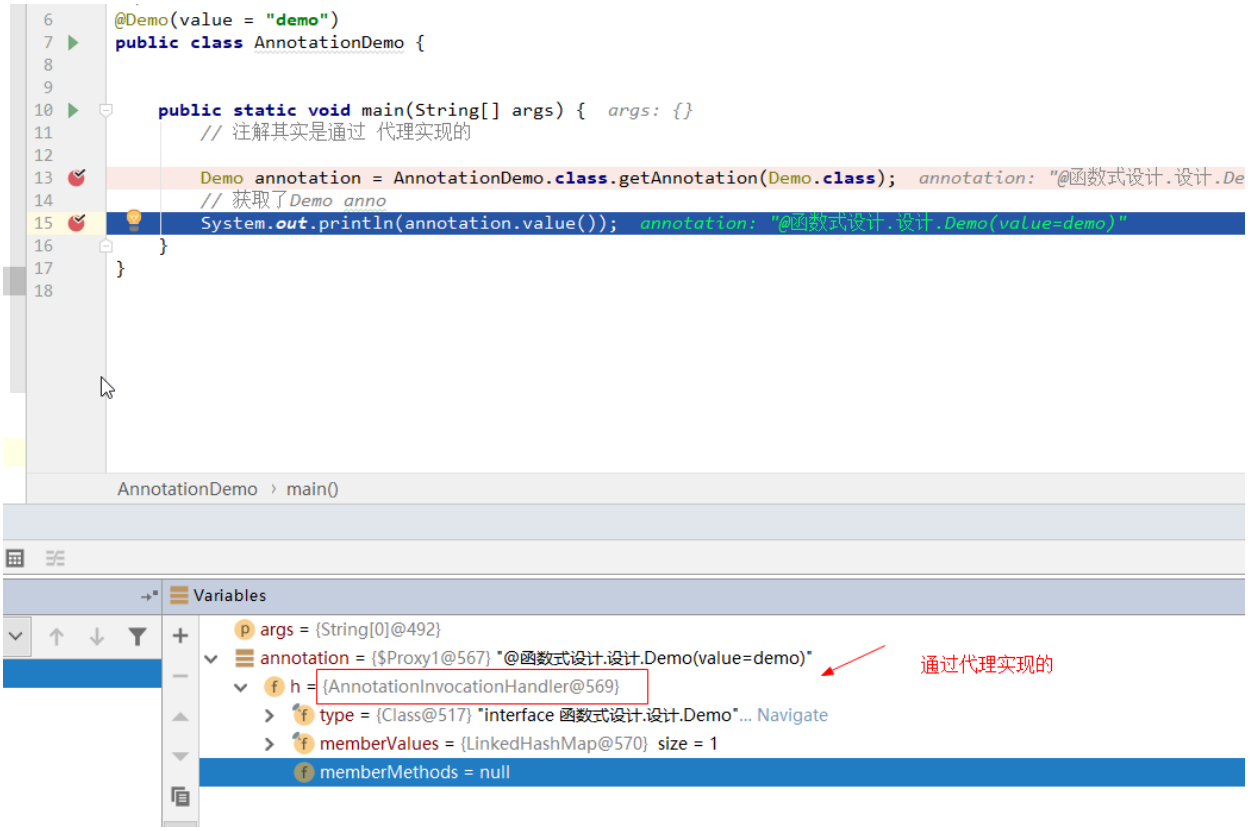
- Supplier<T> 接口定义
  - 基本特点：只出不进
  - 编程范式：作为方法/构造参数、方法返回值
  - 使用场景：数据来源，代码替代接口

- Consumer<T> 接口设计
  - 基本特点：只进不出
  - 编程范式：作为方法/构造参数
  - 使用场景：执行 Callback

- Function<T,R> 接口设计
  - 基本特点：有进有出
  - 编程范式：作为方法/构造参数
  - 使用场景：类型转换、业务处理等

关于注解的一个实现：

其实是一个代理，这里是用 sun的一个实现 AnnotationInvocationHandler类去实现的。



有个问题：为什么@FunctionInterface注解中的@Target(ElementType.*TYPE*)，还是不能在类上去使用?

答案： 其实是可以的， Type是代表类、接口、枚举，没有严格区分。其实关于字节码。其实在反编译之后，注解就是一个接口，继承了Annotation接口