Java函数式接口

定义

- A functional interface is an interface that has just one abstract method (aside from the methods of Object), and thus represents a single function contract.
- 只定义了一个抽象方法的接口（Object类的方法除外）

定义—方法签名

- **签名相同**：两个方法或构造器，当两者名称一样，类型参数一样(如果有的话)，形参一样，则称两者签名一样
- **子签名**：当满足下列任一条件时，称方法m1的签名是方法m2的签名的子签名：

  1.m1和m2签名一样

  2.m1的签名和m2类型擦除后的签名一样

定义—返回值类型的可替换性

- return-type-substitutability： 设方法声明d1的返回值类型是R1，方法声明d2的返回值类型是R2，当满足下列任一条件时，称d1对d2具有返回类型可替换性

  1.R1是void，R2也是void

  2.R1和R2是一样的原始类型

  3.若R1是引用类型，满足以下任一条件

  - R1适用d2的类型参数后，是R2的子类型
  - R1可以通过不受检查的转换被转换成R2的一个子类型
  - d1与d2的签名不一样，然而R1 = |R2|

如何界定只有一个抽象方法

- 给定一个接口I，设M是接口I的一组抽象方法，这些方法与Object类中任一public方法不具有相同的签名。若M中存在一个方法m，当同时满足下列两个条件时，则I是一个函数式接口：

  1.m的签名是M中所有方法的子签名

  2.m的返回值类型是M中所有方法的可替换的返回值类型

例子

```
public interface Runnable {

    public abstract void run();

}
```

例子

```
interface NonFunc {
        boolean equals(Object obj);
}
```

- 非函数式接口，因为boolean equals(Object obj)方法是Object类中的成员方法
- 对比：java.util.Comparator

例子

```
interface Func extends NonFunc {

        int compare(String o1, String o2);

}
```

- 非函数式接口的子接口可以通过声明一个非Object类的成员方法使子接口成为一个函数式接口

例子

```
interface Foo {

    int m();

    Object clone();

}
```

- 非函数式接口，虽然int m()非Object类的成员方法，Object clone()是Object类的成员方法，但Object clone()方法在Object类中非public

函数式接口与类型擦除

```
interface X { int m(Iterable<String> arg); }

interface Y { int m(Iterable<String> arg); }

interface Z extends X, Y {}
```

- 接口Z是函数式接口，虽然从两个父接口继承了两个方法，但两个方法签名相同，逻辑上代表一个方法

函数式接口与类型擦除

```
interface X { Iterable m(Iterable<String> arg); }

interface Y { Iterable<String> m(Iterable arg); }

interface Z extends X, Y {}
```

- Z仍然是一个函数式接口，因为Y.m是X.m子签名，且Y.m的返回值类型是X.m的返回值的可替换类型

函数式接口与类型擦除

interface X { int m(Iterable<String> arg); } interface Y { int m(Iterable<Integer> arg); } interface Z extends X, Y {}	interface X { int m(Iterable<String> arg, Class c); } interface Y { int m(Iterable arg, Class<?> c); } interface Z extends X, Y {}
interface X<T> { void m(T arg); } interface Y<T> { void m(T arg); } interface Z<A, B> extends X<A>, Y<B> {}	• 这三中情况Z都不是函数式接口，不满足子签名条件

函数式接口与类型擦除

```
interface Foo<T, N extends Number> {
    void m(T arg);
    void m(N arg);
}
interface Bar extends Foo<String, Integer> {}
interface Baz extends Foo<Integer, Integer> {}
```

- Foo<T,N> and Bar的声明是合法的，两个方法m非互为子签名，因为类型擦除不一样，也非函数式接口
- Baz是函数式接口

函数式接口与类型擦除

```
interface Exec { <T> T execute(Action<T> a); }

// Functional
```

```
interface X { <T> T execute(Action<T> a); }

interface Y { <S> S execute(Action<S> a); }

interface Exec extends X, Y {}

// Functional: signatures are logically "the same"
```

```
interface X { <T>   T execute(Action<T> a); }

interface Y { <S,T> S execute(Action<S> a); }

interface Exec extends X, Y {}

// Error: different signatures, same erasure
```

标准的函数式接口—6个基本接口

- UnaryOperator
- BinaryOperator
- Predicate
- Function
- Supplier
- Consumer

	接口	函数签名
`interface MyInterface<T> { T method(T t);}`	UnaryOperator<T>	T apply(T t)
`interface MyInterface<T> { T method(T t1, T t2);}`	BinaryOperator<T>	T apply(T t1, T t2)
`interface MyInterface<T> { boolean method(T t);}`	Predicate<T>	boolean test(T t)
`interface MyInterface<T,R> { R method(T t);}`	Function<T,R>	R apply(T t)
`interface MyInterface<T> { T method();}`	Supplier<T>	T get()
`interface MyInterface<T> { void method(T t);}`	Consumer<T>	void accept(T t)

优先使用标准的函数式接口

- 使API易于理解
- 减少概念
- 提供重要的互操作性好处

何时定义自己的函数式接口

- 常常被用到，能从描述性名字得到好处
- 有与之相关的强约定
- 能从自定义的默认方法中得到好处

技巧+最佳实践

- 优先使用标准的函数接口
- 标记@FunctionalInterface注解
- 不要在函数式接口中过度使用默认方法
- 用lambda表达式实例化函数式接口
- 避免在重载方法中将函数式接口作为参数
- 区别对待lambda表达式和内部类

技巧+最佳实践

- lambda表达式要短小精悍且能够自解释
    1. 避免在表达式掺入较大的代码块
    2. 避免指定函数类型
    3. 单一参数不需要使用圆括号
    4. 单行的表达式避免使用return和括号
    5. 优先使用方法引用
- 使用effectively final的变量
- 使用可变对象变量时，保持变量不变

Reference

- Gosling J, Joy B, Steele G, et al. The Java Language Specification (Java SE 8 edition)[J]. California: Oracle, 2015.
- Bloch J. Effective java[M]. Addison-Wesley Professional, 2017.
- Eugen Paraschiv. Lambda Expressions and Functional Interfaces: Tips and Best Practices[DB/OL]. https://www.baeldung.com/java-8-lambda-expressions-tips, 2019-09-04/2019-11-21.

自建vps服务器

- 自建服务器教程：https://github.com/Alvin9999/new-pac/wiki/%E8%87%AA%E5%BB%BAss%E6%9C%8D%E5%8A%A1%E5%99%A8%E6%95%99%E7%A8%8B
- shadowsocks下载：https://www.mediafire.com/ folder/btkdbx7j9lr98/Shadowsocks_%E7%9B%B8%E5%85%B3%E5%AE%A2%E6%88%B7%E7%AB%AF
- 直接购买账号：https://jailbreaker.fun/auth/register?code=EUuf