Kotlin 继承 Kotlin 扩展

# Kotlin 接口

Kotlin 接口与 Java 8 类似，使用 interface 关键字定义接口，允许方法有默认实现：

```
interface MyInterface {
fun bar() // 未实现
fun foo() { //已实现
// 可选的方法体
println("foo")
}
}
```

## 实现接口

一个类或者对象可以实现一个或多个接口。

```
class Child : MyInterface {
override fun bar() {
// 方法体
}
}
```

**实例**

```
interface MyInterface {
fun bar()
fun foo() {
// 可选的方法体
println("foo")
}
}
class Child : MyInterface {
override fun bar() {
// 方法体
println("bar")
}
}
fun main(args: Array<String>) {
val c = Child()
c.foo();
c.bar();
}
```

输出结果为：

```
foo
bar
```

## 接口中的属性

接口中的属性只能是抽象的，不允许初始化值，接口不会保存属性值，实现接口时，必须重写属性：

```
interface MyInterface{
var name:String //name 属性, 抽象的
}
class MyImpl:MyInterface{
override var name: String = "runoob" //重写属性
}
```

**实例**

```
interface MyInterface {
var name:String //name 属性, 抽象的
fun bar()
fun foo() {
// 可选的方法体
println("foo")
}
}
class Child : MyInterface {
override var name: String = "runoob" //重写属性
override fun bar() {
// 方法体
println("bar")
}
}
fun main(args: Array<String>) {
val c = Child()
c.foo();
c.bar();
println(c.name)
}
```

输出结果为：

```
foo
bar
runoob
```

# 函数重写

实现多个接口时，可能会遇到同一方法继承多个实现的问题。例如:

**实例**

```
interface A {
fun foo() { print("A") } // 已实现
fun bar() // 未实现，没有方法体，是抽象的
}
interface B {
fun foo() { print("B") } // 已实现
fun bar() { print("bar") } // 已实现
```

```
}
class C : A {
override fun bar() { print("bar") } // 重写
}
class D : A, B {
override fun foo() {
super<A>.foo()
super<B>.foo()
}
override fun bar() {
super<B>.bar()
}
}
fun main(args: Array<String>) {
val d = D()
d.foo();
d.bar();
}
```

输出结果为：

```
ABbar
```

实例中接口 A 和 B 都定义了方法 foo() 和 bar()， 两者都实现了 foo(), B 实现了 bar()。因为 C 是一个实现了 A 的具体类，所以必须要重写 bar() 并实现这个抽象方法。

然而，如果我们从 A 和 B 派生 D，我们需要实现多个接口继承的所有方法，并指明 D 应该如何实现它们。这一规则 既适用于继承单个实现（bar()）的方法也适用于继承多个实现（foo()）的方法。

← Kotlin 继承 Kotlin 扩展 →