Kotlin抽象类

JunChow520 关注
2019.08.10 03:36:25 字数 597 阅读 2,644

Kotlin中抽象可分为抽象类、抽象函数、抽象属性，抽象类的成员包括抽象函数和抽象属性。抽象类的成员只有定义而没有实现。

声明抽象使用 `abstract` 关键字进行修饰，抽象类的子类必须全部重写带有 `abstract` 修饰的抽象属性和函数。

```
abstract class 抽象类名[(主构造函数)][:继承父类和实现接口]{
  ...
}
```

```
abstract class User{
    var no:String = "001"
    abstract var id:Int
    abstract fun create()
}
```

继承

抽象类可以理解为类定义一个模板，子类会根据模板填充自己的代码。抽象类是为子类定义了一个模板，不同的子类根据抽象父类实现不同的功能。

```
abstract class User{
    abstract fun create()
}
class Member:User(){
    override fun create(){
        println("member create")
    }
}
class Admin:User(){
    override fun create(){
        println("admin create")
    }
}

fun main(args:Array<String>) {
    var user:User = Member()
    user.create()//member create
    //多态
    user = Admin()
    user.create()//admin create
}
```

抽象类与普通类除了都可以具有自己的属性、构造函数、方法等组成部分外，抽象类还可以包含抽象函数和抽象属性。抽象类本身具有普通类的特性以及组成部分，不过抽象类并不能直接被实例化。

```
abstract class User{
    var no:String = "001"
    abstract var id:Int
    abstract fun create()
}
class Member:User(){
    override var id:Int = 1
    override fun create(){
        println("member create: id=$id")
    }
}
class Admin:User(){
```

```
    override var id:Int = 2
    override fun create(){
        println("admin create: id=$id")
    }
}

fun main(args:Array<String>) {
    var user:User = Member()
    user.create()
    //多态
    user = Admin()
    user.create()
}
```

可以使用抽象成员去覆盖非抽象的开放成员

规则

Kotlin中抽象类在顶层定义时只能使用 `public` 可见性修饰符

抽象类中可以定义内部抽象类

抽象类只能继承自一个抽象类

若要实现抽象类的实例化则必须依靠子类向上转型

抽象类可以继承自一个继承类，也就是说抽象类可以作为子类。但是，抽象类建议不要使用 `open` 修饰符，因为会重写抽象类的父类的函数。

应用

设计模式中有一种叫做模板设计模式，即定义一个操作中算法的骨架，将一些步骤延迟到子类中，模板方法使得子类可以不改变算法的结构即可重新定义算法的某些特定的步骤。简单来说，当完成某件事有固定数量的操作步骤，每个步骤根据对象不同，实现细节各不相同。可以在父类中定义完成该事情的总方法，按照完成事件所需的步骤调用每个步骤的实现方法，每个步骤的具体实现可由子类完成。

多态

多态是指同种功能不同的表现形式