Kotlin 类和对象 Kotlin 接口

Kotlin 继承

Kotlin 中所有类都继承该 Any 类，它是所有类的超类，对于没有超类型声明的类是默认超类：

```
class Example // 从 Any 隐式继承
```

Any 默认提供了三个函数：

```
equals()

hashCode()

toString()
```

注意：Any 不是 java.lang.Object。

如果一个类要被继承，可以使用 open 关键字进行修饰。

```
open class Base(p: Int)           // 定义基类

class Derived(p: Int) : Base(p)
```

构造函数

子类有主构造函数

如果子类有主构造函数， 则基类必须在主构造函数中立即初始化。

```
open class Person(var name : String, var age : Int){// 基类

}

class Student(name : String, age : Int, var no : String, var score : Int) : Person(name, age) {

}

// 测试
fun main(args: Array<String>) {
    val s =  Student("Runoob", 18, "S12346", 89)
    println("学生名:  ${s.name}")
    println("年龄:  ${s.age}")
    println("学生号:  ${s.no}")
```

```
    println("成绩： ${s.score}")
}
```

输出结果：

```
学生名： Runoob
年龄： 18
学生号： S12346
成绩： 89
```

子类没有主构造函数

如果子类没有主构造函数，则必须在每一个二级构造函数中用 super 关键字初始化基类，或者在代理另一个构造函数。初始化基类时，可以调用基类的不同构造方法。

```
class Student : Person {

    constructor(ctx: Context) : super(ctx) {
    }

    constructor(ctx: Context, attrs: AttributeSet) : super(ctx,attrs) {
    }
}
```

实例

```
/**用户基类**/
open class Person(name:String){
    /**次级构造函数**/
    constructor(name:String,age:Int):this(name){
        //初始化
        println("-------基类次级构造函数---------")
    }
}

/**子类继承 Person 类**/
class Student:Person{

    /**次级构造函数**/
    constructor(name:String,age:Int,no:String,score:Int):super(name,age){
        println("-------继承类次级构造函数---------")
        println("学生名： ${name}")
        println("年龄： ${age}")
        println("学生号： ${no}")
        println("成绩： ${score}")
    }
```

```
}

fun main(args: Array<String>) {
    var s =  Student("Runoob", 18, "S12345", 89)
}
```

输出结果：

```
-------基类次级构造函数---------
-------继承类次级构造函数---------
学生名： Runoob
年龄： 18
学生号： S12345
成绩： 89
```

重写

在基类中，使用fun声明函数时，此函数默认为final修饰，不能被子类重写。如果允许子类重写该函数，那么就要手动添加 open 修饰它, 子类重写方法使用 override 关键词：

```
/**用户基类**/
open class Person{
    open fun study(){       // 允许子类重写
        println("我毕业了")
    }
}

/**子类继承 Person 类**/
class Student : Person() {

    override fun study(){    // 重写方法
        println("我在读大学")
    }
}

fun main(args: Array<String>) {
    val s =  Student()
    s.study();

}
```

输出结果为:

```
我在读大学
```

如果有多个相同的方法（继承或者实现自其他类，如A、B类），则必须要重写该方法，使用super范型去选择性地调用父类的实现。

```
open class A {
    open fun f () { print("A") }
    fun a() { print("a") }
}

interface B {
    fun f() { print("B") } //接口的成员变量默认是 open 的
    fun b() { print("b") }
}

class C() : A() , B{
    override fun f() {
        super<A>.f()//调用 A.f()
        super<B>.f()//调用 B.f()
    }
}

fun main(args: Array<String>) {
    val c =  C()
    c.f();

}
```

C 继承自 a() 或 b(), C 不仅可以从 A 或则 B 中继承函数，而且 C 可以继承 A()、B() 中共有的函数。此时该函数在中只有一个实现，为了消除歧义，该函数必须调用A()和B()中该函数的实现，并提供自己的实现。
输出结果为:

```
AB
```

属性重写

属性重写使用 override 关键字，属性必须具有兼容类型，每一个声明的属性都可以通过初始化程序或者getter方法被重写：

```
open class Foo {
    open val x: Int get { …… }
}

class Bar1 : Foo() {
    override val x: Int = ……
}
```

你可以用一个var属性重写一个val属性，但是反过来不行。因为val属性本身定义了getter方法，重写为var属性会在衍生类中额外声明一个setter方法

你可以在主构造函数中使用 override 关键字作为属性声明的一部分:

```
interface Foo {
    val count: Int
}

class Bar1(override val count: Int) : Foo

class Bar2 : Foo {
    override var count: Int = 0
}
```

← Kotlin 类和对象 Kotlin 接口 →

1 篇笔记

写笔记

几点补充：

1、子类继承父类时，不能有跟父类同名的变量，除非父类中该变量为 private，或者父类中该变量为 open 并且子类用 override 关键字重写:

```
open class Person(var name: String, var age: Int) {
    open var sex: String = "unknow"
    init {
        println("基类初始化")
    }
}
// 子类的主构造方法的 name 前边也加了 var，这是不允许的，会报'name' hides member of supertype
 and needs 'override' modifier
class Student(var name: String, age: Int, var no: String, var score: Int) : Person(name,
 age) {
    override var sex: String = "male"
}
```

如上代码片段中，子类 Student 主构造方法的第一个字段 name 前边加 var 关键字会报错。

2、子类不一定要调用父类和接口中共同拥有的同名的方法

引用文章中的话：“C 继承自 a() 或 b(), C 不仅可以从 A 或则 B 中继承函数，而且 C 可以继承 A()、B() 中共有的函数。此时该函数在中只有一个实现，为了消除歧义，该函数必须调用A()和B()中该函数的实现，并提供自己的实现。”

我试过了，不是必须调用 A() 和 B() 中该函数的实现，代码如下：

```
open class A {
    open fun f() {
        println("A")
    }
    fun a() {
        println("a")
    }
}
interface B {
    fun f() {
        println("B")
    }
    fun b() {
        println("b")
    }
}
class C : A(), B {
    override fun f() {
        // super<A>.f()
        // super<B>.f()
        println("C")
    }
}
```

如上代码片段，注释掉 `super<A>.f()` 和 `super<B>.f()` 也不报错。

3、关于子类不能用 val 重写父类中的 var，我的猜测是：子类重写父类属性，也就相当于必须重写该属性的 getter 和 setter 方法，而子类中的 val 不能有 setter 方法，所以无法“覆盖”父类中 var 的 setter 方法，相当于缩小了父类中相应属性的使用范围，是不允许的，就像我们不能把父类中一个 public 方法重写成 private 方法一样。

4、如果一个变量想要在定义的时候被初始化，则该变量必须拥有 backing field 字段，该变量的默认 getter 和 setter 方法中是有定义 field 字段的，但是如果我们重写了这个变量的 getter 方法和 setter 方法，并且在 getter 方法和 setter 方法中都没有出现过 filed 这个关键字，则编译器会报错，提示 Initializer is not allowed here because this property has no backing field，除非显式写出 filed 关键字（哪怕它什么都不干，只要放在那里就可以了，我理解是出现一次就相当于“声明”过了，就可以用了，而在定义变量的时候初始化是要求 field 被“声明”过才可以）：

```
var aaa: Int = 0
get() {
    field // 这里必须出现一下field关键字，否则 var aaa: Int = 0 会报错，除非你去掉 = 0这部分，不要给它赋初始化值
    return 0
}
set(value) {}
```

applixy 8个月前 (07-27)