# Kotlin基本语法之(四)成员变量与get、set方法

wanderingGuy 关注

0.277 2019.05.22 22:30:35 字数 527 阅读 11,799

在之前的文章中我们讲到，Kotlin类中的属性既可以用关键字 `var` 声明为可变的，也可以用关键字 `val` 声明为只读的。

## 成员变量(属性)

默认情况下，使用var/val声明的属性可通过对象直接访问，即是public修饰的，除非为属性声明private修饰符。

```
open class Person {
    var age: Int? = null
}

@JvmStatic
fun main(args: Array<String>) {
    val p = Person()
    //为属性赋值
    p.age = 10
    println(p.age.toString())
}
```

在Kotlin的世界里成员变量也可被子类复写。同方法的复写一样，需要在父类的成员属性前声明open表示可复写，子类声明override表示重写。

```
open class Person {
   //属性声明open表示可重写
   open var age: Int? = null
}
open class Student: Person() {
    //重写父类属性
    override var age: Int? = 10*10
}
```

默认情况下属性在声明时必须赋值，除非把属性也声明为abstract的，类中有抽象属性时必须声明为抽象类。

非基本类型的不可空类型(val)的属性可延迟初始化赋值，使用 `lateinit` 实现该功能。只要保证在使用此属性时已赋值即可，若仍未赋值则会抛出属性尚未初始化异常。

```
open class Person {
    //延迟初始化
    lateinit var str: String

    fun getUpper(): String {
        return str.toUpperCase()
    }
}

@JvmStatic
fun main(args: Array<String>) {
    val p = Person()
    //kotlin.UninitializedPropertyAccessException
    //lateinit property str has not been initialized
    println(p.getUpper())
}
```

若想避免上述异常可以在使用属性前使用isInitialized方法判断。

```
open class Person {
    lateinit var str: String

    fun work() {
        if(::str.isInitialized) {
            println("str is isInitialized")
        } else {
            println("str is not isInitialized")
        }
    }
}
```

## getter/setter

默认情况下每个属性都具有getter/setter方法
声明一个属性的完整语法如下：

```
var <propertyName>[: <PropertyType>] [= <property_initializer>]
    [<getter>]
    [<setter>]
```

属性初始值、getter/setter是可缺省，如果属性类型可以从初始值或getter中推断出来则也可缺省。val类型的属性不具备setter。

属性的getter/setter均可复写，即自定义访问器。如果我们定义了一个自定义的setter，那么每次给属性赋值时都会调用它。

来看一个例子：

```
open class Person {
    var age: Int = 10
        //getter缺省为默认
        //setter设置参数前打印参数
        set(value) {
            println("setter $value")
            //field关键字指向属性本身
            field = value
        }
}

@JvmStatic
fun main(args: Array<String>) {
    val p = Person()
    println(p.age)
    p.age = 30
    println(p.age)
}
```

打印结果：

```
10
setter 30
30
```

这里需要解释一下，set方法声明的value是参数名，表示属性实际赋值时的那个对象，约定俗成写做value，可以随意写成其他。

`field` 指向当前属性，field标识符只能用在属性的访问器内。

若想控制setter访问，可以私有化setter。

```
var setterVisibility: String = "abc"
        // 此 setter 是私有的并且有默认实现
        private set
```