# Kotlin by 关键字


竖起大拇指 关注

0.806 2021.03.01 15:03:46 字数 726 阅读 5,533

## 1.Kotlin委托

在委托模式中，两个对象参与处理同一请求，接受请求的对象讲请求委托给另外一个对象来处理。Kotlin直接支持委托模式，更加优雅，简洁。kotlin通过关键字 by 实现委托。

## 2.类委托

类的委托即一个类中定义的方法实际是调用另一个类的对象的方法来实现的。
以下实例中派生类Derived继承了接口Base所有方法，并且委托一个传入的Base类的对象来执行这些方法。

```
//创建接口
interface Base {

    fun print()
}

//实现此接口的被委托的类
class BaseImp(val x:Int) : Base {
    override fun print() {

        println(x)
    }
}

//通过关键字by建立委托类

class Derived (b:Base):Base by b


class Main {

    companion object{

        @JvmStatic
        fun main(args: Array<String>) {
            var baseImp=BaseImp(100)
            Derived(baseImp).print()  //输出100
        }
    }
}

```

在Derived声明中，by子句表示，将b保存在Derived的对象实例内部，而且编译器将会生成继承自Base接口的所有方法，并将调用转发给b。我们看看生成的java代码。

```
public final class Derived implements Base {
   // $FF: synthetic field
   private final Base $$delegate_0;

   public Derived(@NotNull Base b) {
      Intrinsics.checkNotNullParameter(b, "b");
      super();
      this.$$delegate_0 = b;
   }

   public void print() {
      this.$$delegate_0.print();
   }
}
```

# 3.属性委托

属性委托指的是一个类的某个属性值不是在类中直接进行定义，而是将其委托给一个代理类，从而实现对该类的属性统一管理。
属性委托语法格式：

> val/var <属性名>：<类型> by <表达式>

by关键字之后的表达式就是委托，属性的get()方法(以及set()方法)将被委托给这个对象的getValue()和setValue()方法。属性委托不必实现任何接口，但是必须提供getValue()函数(对于var属性，还需要setValue()函数)。

## 3.1定义一个被委托的类

该类包含getValue()方法和setValue()方法，且参数thisRef为进行委托的类的对象，prop为进行委托的属性的对象。

```
//定义包含属性委托的类
class Example {

    var p:String by Delegate()
}

//委托的类
open class Delegate {

    operator fun getValue(thisRef:Any?,property:KProperty<*>):String{
        return "$thisRef,这里委托了${property.name} 属性"
    }

    operator fun setValue(thisRef: Any?,property: KProperty<*>,value:String){
        println("$thisRef 的 ${property.name} 属性赋值为 $value")
    }

}


class Main {

    companion object{

        @JvmStatic
        fun main(args: Array<String>) {

            var e=Example()

            println(e.p) //访问该属性 调用getValue函数

            e.p="rururn" //调用setValue()函数

            println(e.p)

        }
    }
}

```

输出结构为:

```
com.geespace.lib.kotlin.by2.Example@3f99bd52,这里委托了p 属性
com.geespace.lib.kotlin.by2.Example@3f99bd52 的 p 属性赋值为 rururn
com.geespace.lib.kotlin.by2.Example@3f99bd52,这里委托了p 属性
```

## 3.2标准委托

Kotlin的标准库已经内置了很多工厂方法来实现属性的委托。

### 延迟属性Lazy

lazy()是一个函数，接受一个Lambda表达式作为参数，返回一个Lazy<T>实例的函数，返回的实例可以作为延迟属性的委托：第一次调用get()会执行已传递给lazy()的lamda表达式并记录结果，后续调用get()只是返回记录的结果。

```
class LazyTest {

    companion object{

        val lazyValue:String by lazy {
            println("computed!") //第一次调用输出，第二次调用不执行
            "Hello"
        }

        @JvmStatic
        fun main(args: Array<String>) {
            println(lazyValue)
            println(lazyValue)
        }
    }

}
```

执行输出结果：

```
computed!
Hello
Hello
```

### 3.3 把属性存储在映射中

一个常见的用例是在一个映射(map)里存储属性的值。这经常出现在像解析JSON或者其他"动态"事情的应用中。这种情况下，你可以使用映射实例自身作为委托来实现委托属性。

```

class Site(val map:Map<String,Any?>) {

    val name:String by map

    val url:String by map
}


class TestMain {

    companion object{

        @JvmStatic
        fun main(args: Array<String>) {

            val site=Site(mapOf(
                "name" to "maozh",
                "url"  to "www.baidu.com"
            ))

            //读取映射值
            println(site.name)
            println(site.url)
        }
    }
}


```

执行输出结果:

```
maozh
www.baidu.com

```

### 3.4 Not Null

notNull适用于那些无法在初始化阶段就确定属性值的场合。

```
class Foo{
    var notNullBar:String by Delegates.notNull<String>()
}

foo.notNullBar="bar"
println(foo.notNullBar)
```

需要注意，如果属性在赋值前就被访问的话则会抛出异常。