Kotlin 枚举类 kotlin 委托

Kotlin 对象表达式和对象声明

Kotlin 用对象表达式和对象声明来实现创建一个对某个类做了轻微改动的类的对象，且不需要去声明一个新的子类。

对象表达式

通过对象表达式实现一个匿名内部类的对象用于方法的参数中：

```
window.addMouseListener(object : MouseAdapter() {
    override fun mouseClicked(e: MouseEvent) {
        // ...
    }
    override fun mouseEntered(e: MouseEvent) {
        // ...
    }
})
```

对象可以继承于某个基类，或者实现其他接口:

```
open class A(x: Int) {
    public open val y: Int = x
}

interface B {……}

val ab: A = object : A(1), B {
    override val y = 15
}
```

如果超类型有一个构造函数，则必须传递参数给它。多个超类型和接口可以用逗号分隔。

通过对象表达式可以越过类的定义直接得到一个对象：

```
fun main(args: Array<String>) {
    val site = object {
        var name: String = "菜鸟教程"
        var url: String = "www.runoob.com"
    }
    println(site.name)
    println(site.url)
}
```

请注意，匿名对象可以用作只在本地和私有作用域中声明的类型。如果你使用匿名对象作为公有函数的 返回类型或者用作公有属性的类型，那么该函数或属性的实际类型 会是匿名对象声明的超类型，如果你没有声明任何超类型，就会是 Any。在匿名对象 中添加的成员将无法访问。

```
class C {
    // 私有函数，所以其返回类型是匿名对象类型
    private fun foo() = object {
        val x: String = "x"
    }

    // 公有函数，所以其返回类型是 Any
    fun publicFoo() = object {
        val x: String = "x"
    }

    fun bar() {
        val x1 = foo().x        // 没问题
        val x2 = publicFoo().x  // 错误：未能解析的引用“x”
    }
}
```

在对象表达中可以方便的访问到作用域中的其他变量:

```
fun countClicks(window: JComponent) {
    var clickCount = 0
    var enterCount = 0

    window.addMouseListener(object : MouseAdapter() {
        override fun mouseClicked(e: MouseEvent) {
            clickCount++
        }

        override fun mouseEntered(e: MouseEvent) {
            enterCount++
        }
    })
    // ……
}
```

对象声明

Kotlin 使用 object 关键字来声明一个对象。

Kotlin 中我们可以方便的通过对象声明来获得一个单例。

```
object DataProviderManager {
    fun registerDataProvider(provider: DataProvider) {
        // ……
    }

    val allDataProviders: Collection<DataProvider>
        get() = // ……
}
```

引用该对象，我们直接使用其名称即可：

```
DataProviderManager.registerDataProvider(……)
```

当然你也可以定义一个变量来获取获取这个对象，当时当你定义两个不同的变量来获取这个对象时，你会发现你并不能得到两个不同的变量。也就是说通过这种方式，我们获得一个单例。

```
var data1 = DataProviderManager
var data2 = DataProviderManager
data1.name = "test"
print("data1 name = ${data2.name}")
```

实例

以下实例中，两个对象都输出了同一个 url 地址：

```
object Site {
    var url:String = ""
    val name: String = "菜鸟教程"
}
fun main(args: Array<String>) {
    var s1 =  Site
    var s2 = Site
    s1.url = "www.runoob.com"
    println(s1.url)
    println(s2.url)
}
```

输出结果为:

```
www.runoob.com
www.runoob.com
```

对象可以有超类型：

```
object DefaultListener : MouseAdapter() {
    override fun mouseClicked(e: MouseEvent) {
        // ……
    }

    override fun mouseEntered(e: MouseEvent) {
        // ……
    }
}
```

与对象表达式不同，当对象声明在另一个类的内部时，这个对象并不能通过外部类的实例访问到该对象，而只能通过类名来访问，同样该对象也不能直接访问到外部类的方法和变量。

```
class Site {
    var name = "菜鸟教程"
    object DeskTop{
        var url = "www.runoob.com"
        fun showName(){
            print{"desk legs $name"} // 错误，不能访问到外部类的方法和变量
        }
    }
}
fun main(args: Array<String>) {
    var site = Site()
    site.DeskTop.url // 错误，不能通过外部类的实例访问到该对象
    Site.DeskTop.url // 正确
}
```

伴生对象

类内部的对象声明可以用 companion 关键字标记，这样它就与外部类关联在一起，我们就可以直接通过外部类访问到对象的内部元素。

```
class MyClass {
    companion object Factory {
        fun create(): MyClass = MyClass()
    }
}

val instance = MyClass.create()   // 访问到对象的内部元素
```

我们可以省略掉该对象的对象名，然后使用 Companion 替代需要声明的对象名：

```
class MyClass {
    companion object {
    }
}

val x = MyClass.Companion
```

注意：一个类里面只能声明一个内部关联对象，即关键字 companion 只能使用一次。

请伴生对象的成员看起来像其他语言的静态成员，但在运行时他们仍然是真实对象的实例成员。例如还可以实现接口：

```
interface Factory<T> {
    fun create(): T
}


class MyClass {
    companion object : Factory<MyClass> {
        override fun create(): MyClass = MyClass()
    }
}
```

对象表达式和对象声明之间的语义差异

对象表达式和对象声明之间有一个重要的语义差别：

- 对象表达式是在使用他们的地方立即执行的
- 对象声明是在第一次被访问到时延迟初始化的
- 伴生对象的初始化是在相应的类被加载（解析）时，与 Java 静态初始化器的语义相匹配

← Kotlin 枚举类 kotlin 委托 →

点我分享笔记