kotlin海量的操作符

kotlin提供的操作符可以极大的方便我们的开发。

作用域函数

内置的可以作用于所有对象的操作符，常用的有如下几个：

- run{...}
- with()T{...}
- let{...}
- apply{...}
- also{...}

简单例子说明

```
val user = User("xiezhen")
```

let与run:

都会返回闭包的执行结果，区别在于let有闭包参数，而run没有，run闭包中可以用this拿到user对象

```
    val resultLet = user.let { user: User -> "let: ${user.javaClass}" }
     println(resultLet)
     val resultRun = user.run { "run: ${this.javaClass}" }
     println(resultRun)
```

当然输出结果都是两个类名。

also和apply:

都不返回闭包的执行结果而是返回该对象（可方便链式调用），其余区别类似于 let 和run：

```
  user.also { println("also: ${it.javaClass}") }
            .apply {
                println("apply: ${this.javaClass}")
            }
            .name = "qw";
    println(user.name)
```

输出上述相同结果的时候，最后输出了我在链式最后赋值的qw

takeIf和takeUnless:

这是kotlin1.2以后才加入的特性、 takeIf 的闭包返回一个判断结果 ， 当为true时候，语句返回对象本身，返回false时，takeIf 会返回空

```
    user.takeIf { user -> user.name.length > 0 }?.also { println("姓名为:
${it.name}") } ?: print("姓名为空")
```

由于user对象长度大于0，所以执行输出姓名的的闭包块。

而takeUnless 与take if 结果相反，闭包的返回结果为true的时候会返回空

所以如果要用takeUnless输出刚刚的结果，代码应该这么写：

```
user.takeUnless { user -> user.name.length > 0 }?.also { println("姓名为:
${it.name}") }
            ?: println("姓名为: ${user.name}")
```

with（特殊）：

不是以拓展形式存在，是一个顶级函数

```
@kotlin.internal.InlineOnly
public inline fun <T, R> with(receiver: T, block: T.() -> R): R {
    contract {
```

```
    contract {
        callsInPlace(block, InvocationKind.EXACTLY_ONCE)
    }
    return receiver.block()
}
```

用法示例：

```
with(user) {
      this.name="1"
   }
```

等价于：

```
user.apply {
       this.name="1"
   }
```

集合操作符

kotlin原生的对集合类操作符提供了原生的支持，如图所示：

举几个简单的例子

1.集合基本操作：

```
fun test() {
    val list = listOf("kotlin", "Android", "Java", "PHP", "Python", "IOS")
    println(list.contains("kotlin"))
}

fun test1() {
    val list = listOf("kotlin", "Android", "Android", "PHP", "Python",
"IOS")
    println(list.indexOfLast { it == "Android" })
}
```

2.排序：

```
fun test2() {
```

```
fun test2() {
    val list1 = listOf(-1, -2, -3, 1, 11, 2, 33, 10, 6)
    // 升序
    println(list1.sorted())
    //降序
    println(list1.sortedDescending())
    // 反序
    println(list1.reversed())
}
```

3.过滤：

```
fun test3() {
    val list1 = listOf(-1, -2, -3, 1, 11, 2, 33, 10, 6)
    //选择大于0
    println(list1.filter { it > 0 })
    //选择下标大于1且小于4
    println(list1.filterIndexed { index, result -> index > 1 && index < 4
})
}
```

当然也可链式调用： 找出一个数组里面大于0，并且最后一个小于3的数可以这么写：

```
   val list = arrayListOf<Int>(-1, 1, 2, 3, 4, 5)
    var a = list.filter { it > 0 }
            .findLast { it < 3 }
    println(a)
```

共同点：

除with以外，它们都是通过拓展函数实现的