# 比较时忽略大小写：

**val** str1 = **"Hello"  
val** str2 = **"hello"***println*(str1.*equals*(str2, **true**))

# 区间

**val** numbers: IntRange = 1..100 *// 1到100*  
**for** (number **in** numbers) *println*(number)

**val** numbers: IntRange = 1 *until* 100 *// 1到99*

**val** numbers2 = numbers.*reversed*() *// 反转区间，必须用一个新变量保存，原变量不变。新变量的区间为99 ~ 1*

*println*(numbers2.*count*()) *// 区间有多少个数字*

# 遍历集合时带下标

**val** list = *listOf*(**"A"**, **"B"**, **"C"**)  
**for** ((index, element) **in** list.*withIndex*()) {  
 *println*(**"第${**index**}个元素为：$**element**"**)  
}

# 函数表达式

**val** addFun: (x: Int, y: Int) -> Int = **{**x, y **->** x + y**}***// val addFun为声明了一个变量，这个变量的类型为函数类型，具体函数类型为：(x: Int, y: Int) -> Int，  
// 使用函数表达式（{x, y -> x + y}）创建了一个函数并赋值，因为有类型推断，所以函数表达式中的x和y省略了类型***val** sum = addFun(5, 6)  
*println*(sum) *// 输出：11*

我们都知道Kotlin声明变量时可以不写明类型，通过值推断，则上面的可以简写为：

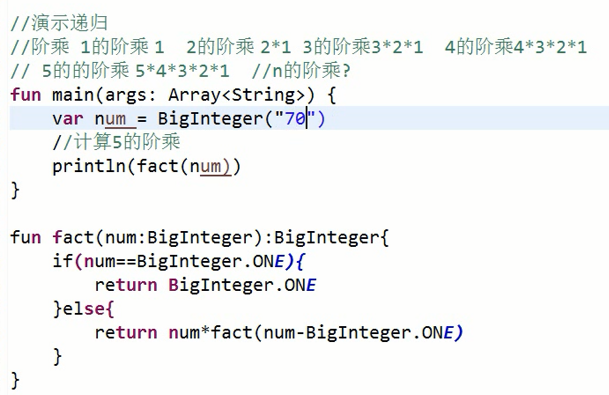
**val** addFun = **{**x: Int, y: Int **->** x + y**}** *// 此时就必须指明x、y的具体类型*

# 读控制台的内容：

**fun** main(args: Array<String>) {  
 **val** string = *readLine*()  
}

# 大整数

如果一个整数太大了，加Long类型都装不下，这时可用BigInteger，要多大的整数都可以装的下。

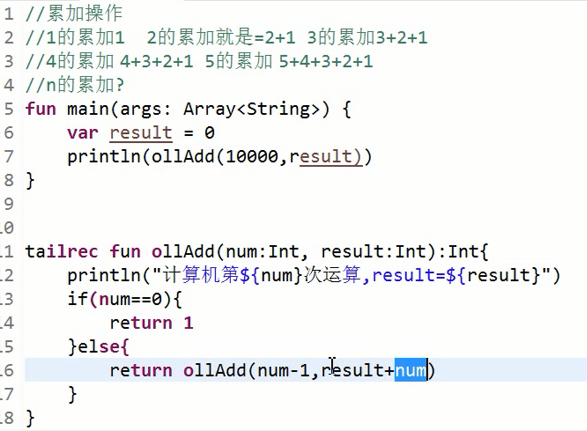


# 尾递归

尾递归的意思是在函数的最后一行代码语句必须是调用函数自己，之后没有其它代码了

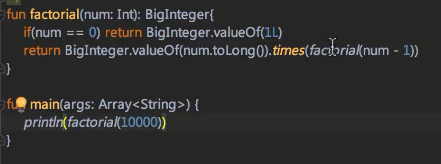
**fun** main(args: Array<String>) {  
 **val** result = *sum*(1000000)  
}  
  
**fun** sum(n: Int): Int {  
 *println*(**"n = $**n**"**)  
 **if** (n == 1) {  
 **return** n  
 } **else** {  
 **return** n + *sum*(n - 1)  
 }  
}

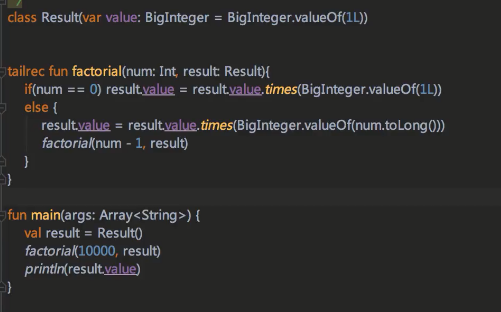
运行将会出现：StackOverflowError，递归次数太多，栈内存不够用了。使用尾递归解决此问题：



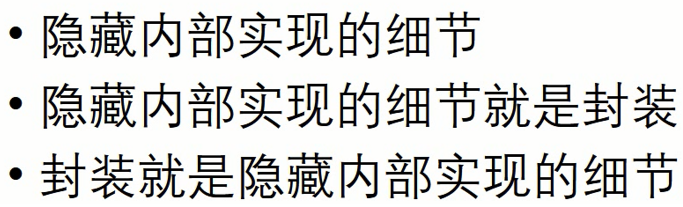
尾递归对函数的要求：函数最后一行代码必须是调用自己，后面没有其它的代码

求阶乘





# 封装：



比如我们买电视机、手机都会有壳把里面的电路什么的隐藏起来，只暴露出用户可使用的功能即可。外壳有的会贴有拆开不保修，标签撕毁不保修，因为这些是封装起来的不允许用户去修改。在代码中则使用private代表不封装。接口就相当于外壳，因为外壳都是可以让用户使用的，接口定义的所有方法也都是public的。

# 继承

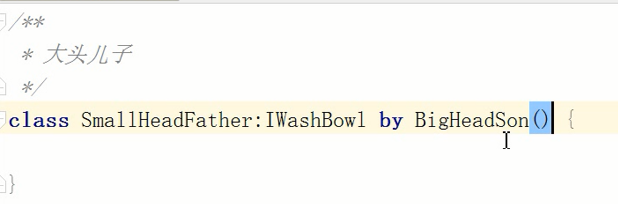
一个类要被继承必须声明为open，一个方法要被覆盖也要声明为open

**open class** Father {  
 **open fun** run() { }  
}



继承类或抽象类，后面要加括号表示调用父类的哪个构造方法，而实现接口则不用加括号

# 委托代理

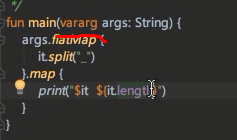


把对Father的调用转给Son

# Kotlin大事件



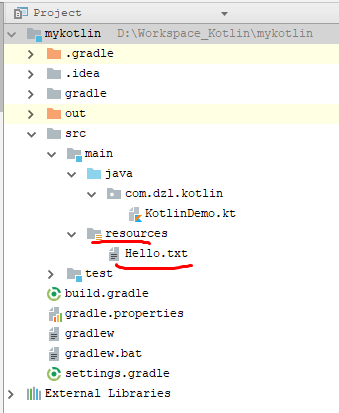
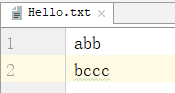
# 可用变量



# 不带val、var修饰的构造参数

**class** 人(**var 姓名**: String, 性别: String){  
 **fun** sayMyName() {  
 *println*(**"大家好，我叫$姓名"**)  
 }  
  
 **init** {  
 *println*(**"我的性别是：$**性别**"**) *// 不带val和var修饰的变量只能在构造函数中访问,因为它只是一个普通的函数参数，并不会被赋值给成员变量* }  
}

# 读文件

**fun** main(args: Array<String>) {  
 **val** text = File(ClassLoader.getSystemResource(**"Hello.txt"**).*path*).*readText*()  
 *println*(text)  
}

这样就能把文件内容读出来了，天上需要统计每一个英文字符出现的次数：

接下来就是统计字符个数了：

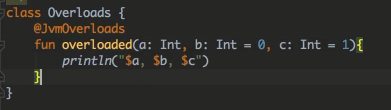
text.*toList*()  
 .*filter* **{** !**it**.*isWhitespace*() **}** .*sorted*()  
 .*groupBy* **{ it }** .*forEach* **{** *println*(**"${it**.**key} = ${it**.**value**.**size}"**) **}**

# 像Java中的属性



这里age属性加上了@JvmField，则Kotlin不会产生set\get方法，且这成员变量也不能私有化。

# 在Java中调用Kotlin中的重载



需要加上注解才行，否则只能在Kotlin中使用重载