反引号

键盘左上角与波浪线在一起的符号

- 可用于解决关键字冲突的问题
- 将一个不合法的字符变成合法的

因为反引号的存在，我们可以在代码中做对方法如下定义：

```
fun normal() {
    println("test")
}

fun `1234`() {
    println("test1")
}

fun ` `() {
    println("test2")
}

fun `   `() {
    println("test3")
}
```

在kotlin中调用：

```
fun main(args: Array<String>) {
    normal()
    `1234`()
    ` `()
    `   `()
}
```

这么做意义何在呢？这样可以保证某些方法我们只希望在kotlin中被调用，而不能在java中调用，如果在java中：除了上述的normal方法以外，其他声明都是不合法的。

比较对象:

kotlin	java
a==b	a.equals(b)
a===b	a==b

kotlin中：

```
fun test() {
    var a = "String"
    //kotlin没有直接的构造方法
    val b =
    String("String".toByteArray())
    println(a == b)
    print(a === b)
}
```

在java中：

```
 public static void main(String[] args) {
        String a = "String";
        String b = new String("String");
        System.out.println(a.equals(b));
        System.out.println(a==b);
    }
```

输出结果是一致的

类型别名: typealias

类型别名为现有类型提供替代名称。 如果类型名称太长，你可以另外引入较短的名称，并使用新的名称替代原类型名。

可以理解为类的昵称： 例如我给File类加一个昵称A

```
//给普通类
typealias A = File
val a:File =A("test.jpg")

//给内部类
class A {
     inner class Inner
}
typealias AInner = A.Inner
```

再比如，kotlin中的hashMap就是映射java中的hashMap

```
@SinceKotlin("1.1") public actual typealias HashMap<K, V> =
java.util.HashMap<K, V>
```

还可以为函数类型提供另外的别名：

```
fun test(p: (Int) -> Unit) {
    p(10)
}

typealias NiuBi = (Int) -> Unit

fun test2(p:NiuBi){
    p(20)
}

fun main(args: Array<String>) {
    test {
        println(it)
    }
    test2 {
        println(it)
    }
}
```