Scala

一种在JVM下运行的静态语言
翻译：Java套皮

Scala的特点

1. 多范式：面向对象+面向过程
   （一共有四种变成方法：面向过程、面向对象、）
2. 带函数的Java,与Java无缝衔接
3. 简洁高效

环境搭建

（不想写了）

第一个Scala代码

```
object ScalaTest1 {
  def main(args: Array[String]): Unit = {
    println("张泽铠虽然没去上课，但是有在正常完成实验，录像也在跟着看。")
  }
}
```

Scala基础

注释

1. 单行注释： //
2. 多行注释：/* */

变量与常量

基本格式：var/val 变量名：数据类型 = 初始值

其中：*var* 代表定义变量，*val*代表定义常量（就是Java的final） 如：

```
var a:Int = 10  // 定义一个可变的整型a,初始值为10.
val b:Int = 20  // 定义一个不可变的整型b，处置值为20
```

甚至其实也可以不加数据类型，Scala会自动指定

```
var a = 10
val b = 20
```

在定义变量上，好像完全可以拿他当加了var和val的Python用

Scala中的数据类型

1. 整数类型：
   Byte：2^(8-1)-1
   Short：2^(16-1)-1
   Int:2^(32-1)-1
   Long：2^(64-1)-1
   一般其实Int和Long也就够了

2. 浮点数
   Float:32位单精度浮点数
   Double:64位双精度浮点数

3. 字符型
   Char

4. 布尔型
   Boolean:只允许取true或false

5. **其他类型（重点）**
   Unit:无值（相当于void），仅有一个实例值——().
   Null:空值，仅有一个实例值——null.
   Nothing:最底层类（类似于Java的Object）可用于提高函数返回值等情况在特殊情况下的兼容性。

Scala的标识符命名规范

标识符：Scala中凡是可以自己起名字的都是"标识符"
命名规则:

1. 字母或下划线开头，后接字母、数字、下划线
2. 以操作符开头，并且**只包含**操作符
3. ``（反引号）包括的**任意字符串**

举几个例子：

```
 var 1hello: String = "" // error 数字不能开头
 var h-b: String = "" // error 不能用-
 var x h: String = "" // error 不能有空格
 var Int: String = "" // ok 因为在 Scala 中 Int 是预定义的字符，
vf不是关键字，但不推荐
 var _: String = "hello" // ok 单独一个下划线不可以作为标识符，
因为_被认为是一个方法
var +*-/#! : String = "" // ok
 var +*-/#!1 : String = "" // error 以操作符开头，必须都是操作
 var if : String = "" // error 不能用关键字
 var `if` : String = "" // ok 用反引号`....`包括的任意字符串，
包括关键字
```

Scala的基本语法

输入输出

1. 输出：
   printf/println，两个用法稍微有点不同

```
object ScalaTest1 {
  def main(args: Array[String]): Unit = {
    var age = 10
    var name = "zhangzekai"
    printf("age = %d, name = %s\n",age,name)
    println("age = %d, name = %s",age,name)
    println("age =" + age, "name = " + name)
  }
}
```

```
上面三行打印出来的结果分别是：
    age = 10, name = zhangzekai
    (age = %d, name = %s,10,zhangzekai)
    (age =10,name = zhangzekai)
```

总之：printf用"%"传参，println用"+"传参

2. 键盘输入
   StdIn.readLine()
   StdIn.readShort()
   StdIn.readDouble()
   ..... *没啥好说的，赋值就完了*

分支

if-esle

```
if(条件){
    // 代码1
}else if(条件){
    // 代码2
}else{
    // Scala不支持Switch
}
```

例：

```
import scala.io.StdIn

object ScalaTest2 {
  def main(array: Array[String]): Unit = {
    println("age:")
    var age = StdIn.readInt()
    if (age<18){
      println("小于18")
    }else if (age >=18 && age <=60){
      println("18-60")
    }else{
      println("》60")
    }
  }
}
```

循环

for循环

```
    for(循环条件){
        // 代码块
    }
```

其中循环条件有两种，一种是**To**，一种是**Until**

```
    println("------To------")
    for(i <- 1 to 3){
      println(i)
    }
    println("-----Until----")
    for(j <- 1 until 3){
      println(j)
    }
```

对于同样的区间:
To输出1、2、3
Until输出1、2
To为闭区间，Until为前闭后开区间 当然也可以嵌套循环

```
    for(i <- 1 to 3; j <- 1  to 3 ){
      println("i:" + i + ", j:" + j)
    }
```

输出结果为：

```
    i:1, j:1
    i:1, j:2
    i:1, j:3
    i:2, j:1
    i:2, j:2
    i:2, j:3
    i:3, j:1
    i:3, j:2
    i:3, j:3
```

等价于

```
for (i <- 1 to 3) {
 for (j <- 1 to 3) {
 println("i =" + i + " j=" + j)
 }
}
```

不过一般会把不同条件放到不同行去写

```
for {
 i <- 1 to 3
 j <- 1 to 3
} {
 println("i=" + i + " j=" + j)
}
```

其中当然也可以引入新变量，比如：

```
for {
 i <- 1 to 3
 j = 3 - i
} {
 println("i=" + i + " j=" + j)
}
```

输出结果为

```
    i=1 j=2
    i=2 j=1
    i=3 j=0
```

当然也可以设置步长,比如我们将步长设置为2

```
for(i <- 1 to 10 by 2) {
  print(i +"\t")
}
```

输出结果为

```
1   3   5   7   9
```

在Scala可以加入 ***循环守卫*** 功能，也称条件判断式、守卫。

```
for(i <- 1 to 5 if i != 2) {
  println(i)
}
```

输出结果为

```
1   3   4   5
```

然后还有一些其他的玩法，比如 ***循环返回值***和 ***倒叙打印***

1. 循环返回值

```
var res = for (i <- 1 to 10) yield i*2
println(res)
```

上述代码的含义是:
定义res存储结果 → i由1到10循环 → 将每个i乘2添加到res中 输出结果为

```
Vector(2, 4, 6, 8, 10, 12, 14, 16, 18, 20)
```

当然，yield后面也可以加 {} ，然后里面写各种东西 由于 ***循环返回值很少使用***，所以不再赘述

2. 倒叙打印 添加一个 ***reverse*** 就可以
   比如:

```
for (i <- 1 to 10 reverse)
  print(i + "\t")
```

输出为

```
10  9   8   7   6   5   4   3   2   1
```

While循环

While和do...While循环的用法和Java中一样，不再赘述。

循环控制

Scala中不支持break和continue作为单独的函数,一般使用 ***breakable***结构实现break和continue的功能。
一般有两种控制方法

1. breakable结构 例：循环遍历1-10的数字，在到5的时候结束循环，并输出"正常结束循环"

```
    breakable {
      for (elem <- 1 to 10) {
        printf(elem+ "\t")
        if (elem == 5) break
      }
    }
    println("正常结束循环")
```

输出结果如下：

```
1   2   3   4   5   正常结束循环
```

2. 使用异常丢出报错，退出循环

```
    try{
      for (i <- 1 to 10) {
        printf(i + "\t")
        if (i == 5)
          throw new RuntimeException
      }
    }catch {
      case e:RuntimeException =>
    }
  println("结束循环")
```

运行结果如下：

```
1    2    3    4    5    结束循环
```