Scala

基本语法

Scala 基本语法需要注意以下几点：

- 区分大小写 - Scala是大小写敏感的，这意味着标识Hello 和 hello在Scala中会有不同的含义
- 类名 - 对于所有的类名的第一个字母要大写。 如果需要使用几个单词来构成一个类的名称，每个单词的第一个字母要大写

```
    class MyFirstScalaClass
```

- 方法名称 - 所有的方法名称的第一个字母用小写。 如果若干单词被用于构成方法的名称，则每个单词的第一个字母应大写

```
    def myMethodName()
```

- def main(args: Array[String]) - Scala程序从main()方法开始处理，这是每一个Scala程序的强制程序入口部分。

Scala 关键字

abstract	case	catch	class
def	do	else	extends
false	final	finally	for
forSome	if	implicit	import
lazy	match	new	null
object	override	package	private
protected	return	sealed	super
this	throw	trait	try
true	type	val	var
while	with	yield	
-	:	=	=>
<-	<:	<%	>:
#	@		

Scala注释

```
object HelloWorld {
   /* 这是一个 Scala 程序
    * 这是一行注释
    * 这里演示了多行注释
    */
   def main(args: Array[String]) {
      // 输出 Hello World
      // 这是一个单行注释
      println("Hello, world!")
   }
}
```

换行符

Scala是面向行的语言，语句可以用分号（;）结束或换行符。Scala 程序里,语句末尾的分号通常是可选的。如果你愿意可以输入一个,但若一行里仅 有一个语句也可不写。另一方面,如果一行里写多个语句那么分号是需要的。例如

```
val s = "hello world"; println(s)
```

定义包

Scala 使用 package 关键字定义包。在文件的头定义包名，这种方法就后续所有代码都放在该包中。 比如：

```
package com.runoob
class HelloWorld
```

引用

Scala 使用 import 关键字引用包。

```
import java.awt.Color  // 引入Color

import java.awt._  // 引入包内所有成员

def handler(evt: event.ActionEvent) { // java.awt.event.ActionEvent
  ...  // 因为引入了java.awt，所以可以省去前面的部分
}
```

如果想要引入包中的几个成员，可以使用selector（选取器）：

```
import java.awt.{Color, Font}

// 重命名成员
import java.util.{HashMap => JavaHashMap}

// 隐藏成员
import java.util.{HashMap => _, _} // 引入了util包的所有成员，但是HashMap被隐藏了
```

注意：默认情况下，Scala 总会引入 java.lang._ 、 scala._ 和 Predef._，这里也能解释，为什么以scala开头的包，在使用时都是省去scala.的。

scala 数据类型

数据类型	描述
Byte	8位有符号补码整数。数值区间为 -128 到 127
Short	16位有符号补码整数。数值区间为 -32768 到 3
Int	32位有符号补码整数。数值区间为 -2147483648 到 2
Long	64位有符号补码整数。数值区间为 -9223372036854775808 到 92
Float	32 位, IEEE 754 标准的单精度浮点数
Double	64 位 IEEE 754 标准的双精度浮点数
Char	16位无符号Unicode字符, 区间值为 U+
String	字符序列
Boolean	true或
Unit	表示无值，和其
Null	null 或空引用
Nothing	Nothing类型
Any	Any是所有其他类的超类
AnyRef	AnyRef类是Scala里所有引用类(reference class)的基类

上表中列出的数据类型都是对象，也就是说scala没有java中的原生类型。在scala是可以对数字等基础类型调用方法的。

Scala 基础字面量

整型字面量

整型字面量用于 Int 类型，如果表示 Long，可以在数字后面添加 L 或者小写 l 作为后缀。：

```
scala> 21
res4: Int = 21

scala> 21L
res5: Long = 21
```

浮点型字面量

如果浮点数后面有f或者F后缀时，表示这是一个Float类型，否则就是一个Double类型的。实例如下：

```
scala> 11.1
res6: Double = 11.1

scala> 11.1f
res7: Float = 11.1
```

布尔型字面量

布尔型字面量有 true 和 false。

符号字面量

符号字面量被写成： '<标识符> ，这里 <标识符> 可以是任何字母或数字的标识（注意：不能以数字开头）。这种字面量被映射成预定义类scala.Symbol的实例。

如： 符号字面量 'x 是表达式 scala.Symbol("x") 的简写

字符字面量

在 Scala 字符变量使用单引号 ' 来定义，如下：

```
'a'
'\n'
```

字符串字面量

在 Scala 字符串变量使用双引号 " 来定义，如下：

```
"hello world"
```

多行字符串

多行字符串用三个双引号来表示分隔符，格式为：""" ... """。

实例如下：

```
val foo = """菜鸟教程
www.runoob.com
www.w3cschool.cc
www.runnoob.com
以上三个地址都能访问"""
```

Null 值

空值是 scala.Null 类型。

Scala.Null和scala.Nothing是用统一的方式处理Scala面向对象类型系统的某些"边界情况"的特殊类型。

Null类是null引用对象的类型，它是每个引用类（继承自AnyRef的类）的子类。Null不兼容值类型。

Scala 转义字符

下表列出了常见的转义字符：

转义字符	**Unicode**	描述
\b	\u0008	退格(BS)，将当前位置移到前一列
\t	\u0009	水平制表(HT)（跳到下一个TAB位置）
\n	\u000a	换行(LF)，将当前位置移到下一行开头
\f	\u000c	换页(FF)，将当前位置移到下页开头
\r	\u000d	回车(CR)，将当前位置移到本行开头
\"	\u0022	代表一个双引号(")字符
\'	\u0027	代表一个单引号（'）字符
\\	\u005c	代表一个反斜线字符 ''

Scala变量

变量声明

在 Scala 中，使用关键词 "var" 声明变量(可以var重新定义更改数据类型)，使用关键词 "val" 声明常量。

声明变量实例如下：

```
var myVar : String = "Foo"
var myVar : String = "Too"
```

以上定义了变量 myVar，我们可以修改它。

声明常量实例如下：

```
val myVal : String = "Foo"
```

以上定义了常量 myVal，它是不能修改的。如果程序尝试修改常量 myVal 的值，程序将会在编译时报错。

Scala 访问修饰符

私有(Private)成员

用 private 关键字修饰，带有此标记的成员仅在包含了成员定义的类或对象内部可见，同样的规则还适用内部类

```
class Outer{
    class Inner{
    private def f(){println("f")}
    class InnerMost{
        f() // 正确
        }
    }
    (new Inner).f() //错误
}
```

保护(Protected)成员

在 scala 中，对保护（Protected）成员的访问比 java 更严格一些。因为它只允许保护成员在定义了该成员的的类的子类中被访问。而在java中，用protected关键字修饰的成员，除了定义了该成员的类的子类可以访问，同一个包里的其他类也可以进行访问。

```
package p{
class Super{
    protected def f() {println("f")}
    }
    class Sub extends Super{
        f()
    }
    class Other{
        (new Super).f() //错误
    }
}
```

公共(Public)成员

Scala中，如果没有指定任何的修饰符，则默认为 public。这样的成员在任何地方都可以被访问。

作用域保护

Scala中，访问修饰符可以通过使用限定词强调。格式为:

```
private[x]

或

protected[x]
```

这里的x指代某个所属的包、类或单例对象。如果写成private[x],读作"这个成员除了对[...]中的类或[...]中的包中的类及它们的伴生对像可见外，对其它所有类都是private。

这种技巧在横跨了若干包的大型项目中非常有用，它允许你定义一些在你项目的若干子包中可见但对于项目外部的客户却始终不可见的东西。

```
package bobsrocckets{
    package navigation{
        private[bobsrockets] class Navigator{
         protected[navigation] def useStarChart(){}
         class LegOfJourney{
             private[Navigator] val distance = 100
             }
            private[this] var speed = 200
            }
        }
        package launch{
        import navigation._
        object Vehicle{
        private[launch] val guide = new Navigator
        }
    }
}
```

if 语句

if 语句有布尔表达式及之后的语句块组成。

语法

if 语句的语法格式如下：

```
if(布尔表达式)
{
   // 如果布尔表达式为 true 则执行该语句块
}

if(布尔表达式){
   // 如果布尔表达式为 true 则执行该语句块
```

```
}else{
   // 如果布尔表达式为 false 则执行该语句块
}

if(布尔表达式 1){
   // 如果布尔表达式 1 为 true 则执行该语句块
}else if(布尔表达式 2){
   // 如果布尔表达式 2 为 true 则执行该语句块
}else if(布尔表达式 3){
   // 如果布尔表达式 3 为 true 则执行该语句块
}else {
   // 如果以上条件都为 false 执行该语句块
}
```

scala 循环

for

Scala 语言中 for 循环的语法：

```
for( var x <- Range ){
   statement(s);
}
```

以上语法中，Range 可以是一个数字区间表示 i to j ，或者 i until j。左箭头 <- 用于为变量 x 赋值。

实例 以下是一个使用了 i to j 语法(包含 j)的实例:

```
object Test {
   def main(args: Array[String]) {
      var a = 0;
      // for 循环
      for( a <- 1 to 10){
         println( "Value of a: " + a );
      }
   }
}
```

以下是一个使用了 i until j 语法(不包含 j)的实例:

```
object Test {
   def main(args: Array[String]) {
      var a = 0;
      // for 循环
      for( a <- 1 until 10){
         println( "Value of a: " + a );
```

```
      }
   }
}
```

在 for 循环 中你可以使用分号 (😉 来设置多个区间，它将迭代给定区间所有的可能值。以下实例演示了两个区间的循环实例：

```
object Test {
   def main(args: Array[String]) {
      var a = 0;
      var b = 0;
      // for 循环
      for( a <- 1 to 3; b <- 1 to 3){
         println( "Value of a: " + a );
         println( "Value of b: " + b );
      }
   }
}
```

for 循环集合

for 循环集合的语法如下：

```
for( var x <- List ){
   statement(s);
}
```

实例

以下实例将循环数字集合。

```
object Test {
   def main(args: Array[String]) {
      var a = 0;
      val numList = List(1,2,3,4,5,6);

      // for 循环
      for( a <- numList ){
         println( "Value of a: " + a );
      }
   }
}
```

for 循环过滤

Scala 可以使用一个或多个 if 语句来过滤一些元素。

以下是在 for 循环中使用过滤器的语法。

```
for( var x <- List
      if condition1; if condition2...
   ){
   statement(s);
```

你可以使用分号(;)来为表达式添加一个或多个的过滤条件。

for 使用 yield

你可以将 for 循环的返回值作为一个变量存储。语法格式如下：

```
var retVal = for{ var x <- List
     if condition1; if condition2...
}yield x
```

注意大括号中用于保存变量和条件，retVal 是变量， 循环中的 yield 会把当前的元素记下来，保存在集合中，循环结束后将返回该集合。

实例

以下实例演示了 for 循环中使用 yield：

```
object Test {
   def main(args: Array[String]) {
      var a = 0;
      val numList = List(1,2,3,4,5,6,7,8,9,10);

      // for 循环
      var retVal = for{ a <- numList
                        if a != 3; if a < 8
                       }yield a

      // 输出返回值
      for( a <- retVal){
         println( "Value of a: " + a );
      }
   }
}
```

无限循环

如果条件永远为 true，则循环将变成无限循环。我们可以使用 while 语句来实现无限循环：

SC