Scala 基础语法

目录

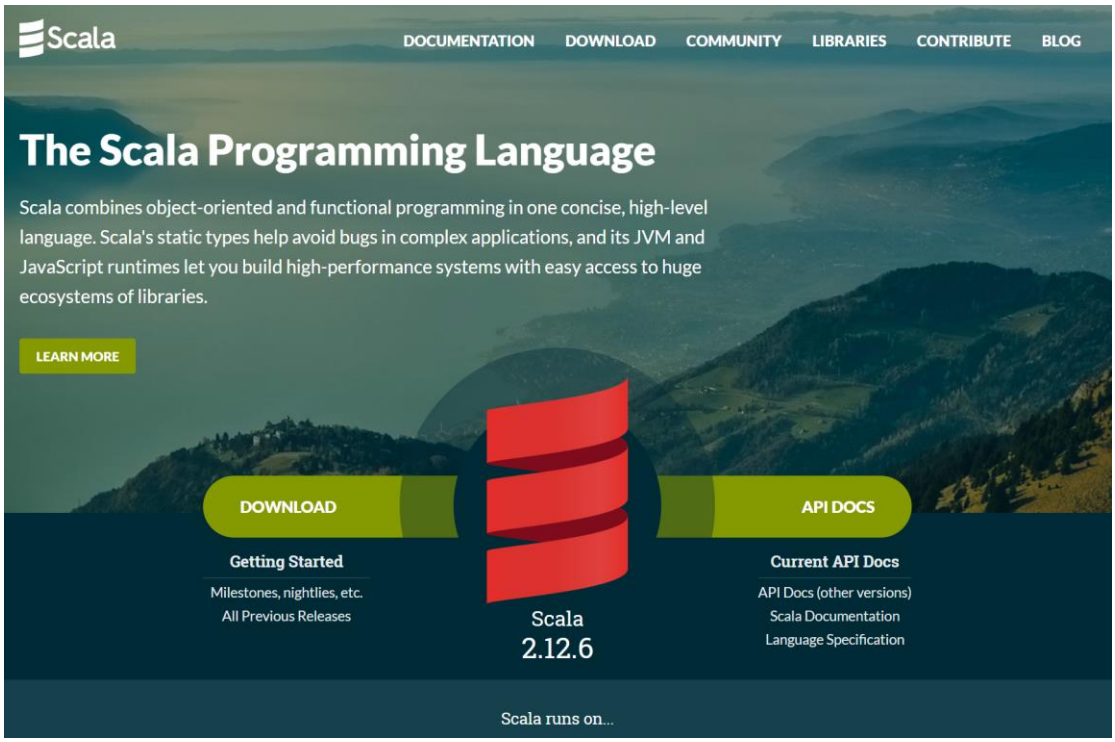
1、Scala 概述

1.1、什么是 Scala

Scala 官网：https://www.scala-lang.org/

Scala 是一种多范式的编程语言，其设计的初衷是要集成面向对象编程和函数式编程的各种特性。Scala 运行于 Java 平台（Java 虚拟机），并兼容现有的 Java 程序。

Scala(Scalable Language 的简称)语言是一种能够运行于 JVM 和.Net 平台之上的通用编程语言，既可用于大规模应用程序开发，也可用于脚本编程，它由由 Martin Odersky 于 2001 开发，2004 年开始程序运行在 JVM 与.Net 平台之上，由于其简洁、优雅、类型安全的编程模式而受到关注。



注意三者的区别：
面向对象编程
面向过程编程
函数式编程

编程语言之分：
1、 面向对象和函数式编程
2、 静态编程语言和动态编程语言
3、 编译型和解释型
4、 汇编语言，脚本语言，机器语言，高级语言
5、 强类型语言和弱类型语言

Python 是动态类型语言，是强类型语言。
JavaScript 是动态类型语言，是弱类型语言。
Java 是静态类型语言，是强类型语言。

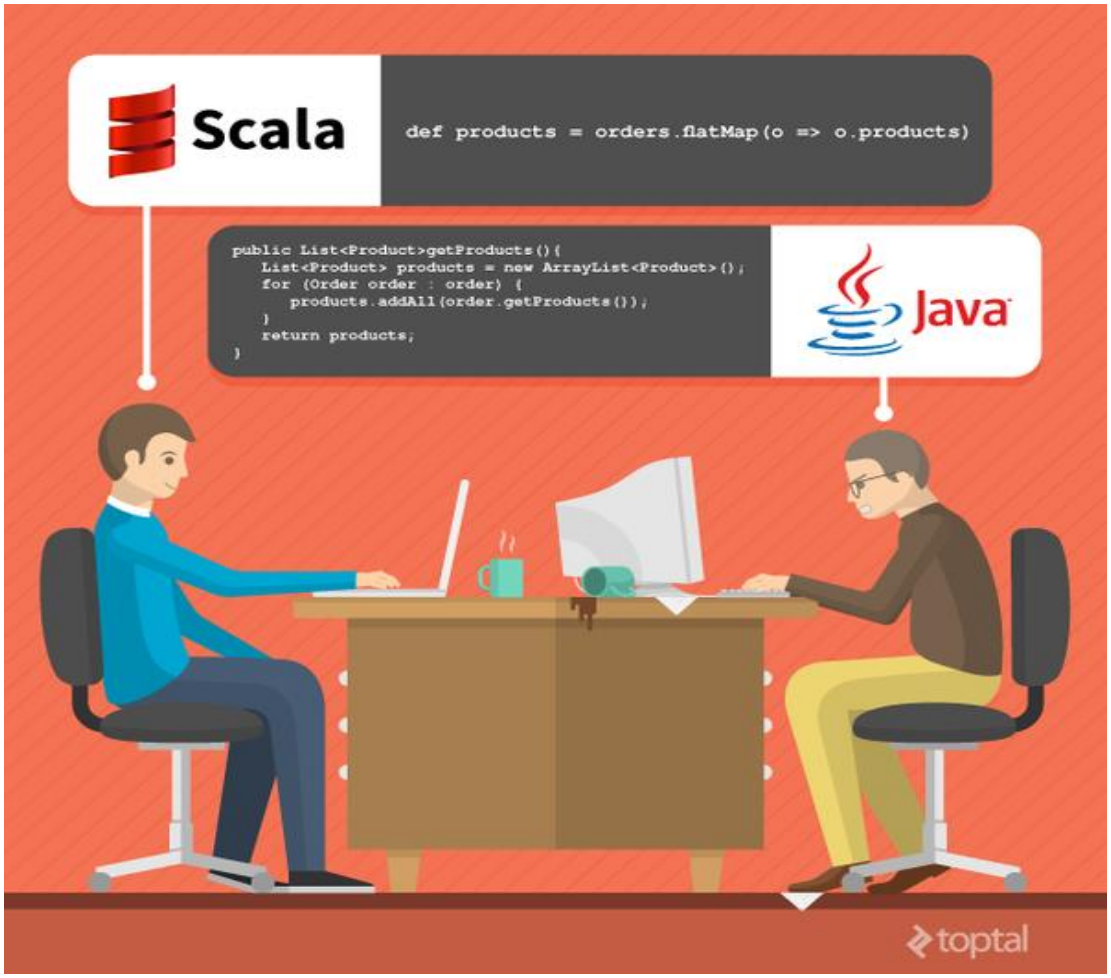
1.2、为什么要学 Scala

第一，基于编程语言自身：

1、 **优雅**：这是框架设计师第一个要考虑的问题，框架的用户是应用开发程序员，API 是否优雅直接影响用户体验。

2、 **速度快**：Scala 语言表达能力强，一行代码抵得上 Java 多行，开发速度快；Scala 是静态编译的，所以和 JRuby,Groovy 比起来速度会快很多。

3、 **能融合到 Hadoop 生态圈**：Hadoop 现在是大数据事实标准，Spark 的出现并不是要取代 Hadoop，而是要完善 Hadoop 生态。JVM 语言大部分可能会想到 Java，但 Java 做出来的 API 太丑，或者想实现一个优雅的 API 太费劲。



第二，基于活跃度：

1、作为流行的开源大数据内存计算引擎 Spark 的源码编程语言--Spark 有着良好的性能优势

2、Scala 将成为未来大数据处理的主流语言
“If I were to pick a language to use today other than Java, it would be Scala.” -- James Gosling

3、最新 TIOBE 编程语言排行榜--Scala 进入前 20

May 2018	May 2017	Change	Programming Language	Ratings	Change
1	1		Java	16.380%	+1.74%
2	2		C	14.000%	+7.00%
3	3		C++	7.668%	+2.92%
4	4		Python	5.192%	+1.64%
5	5		C#	4.402%	+0.95%
6	6		Visual Basic .NET	4.124%	+0.73%
7	9	^	PHP	3.321%	+0.63%
8	7	v	JavaScript	2.923%	-0.15%
9	-	^^	SQL	1.987%	+1.99%
10	11	^	Ruby	1.182%	-1.25%
11	14	^	R	1.180%	-1.01%
12	18	^^	Delphi/Object Pascal	1.012%	-1.03%
13	8	vv	Assembly language	0.998%	-1.86%
14	16	^	Go	0.970%	-1.11%
15	15		Objective-C	0.939%	-1.16%
16	17	^	MATLAB	0.929%	-1.13%
17	12	vv	Visual Basic	0.915%	-1.43%
18	10	vv	Perl	0.909%	-1.69%
19	13	vv	Swift	0.907%	-1.37%
20	31	^^	Scala	0.900%	+0.18%

2、Scala 编译器安装

2.1、安装 JDK

因为 Scala 是运行在 JVM 平台上的，所以安装 Scala 之前要安装 JDK
具体安装见文档：**资料-JDK 安装.pdf**

2.2、Windows 平台安装 Scala

访问 Scala 官网 http://www.scala-lang.org/下载 Scala 编译器安装包，目前最新版本是 2.12.x，但是目前大多数的框架都是用 2.10.x 或者 2.11.x 编写开发的，将来我们要基于 Spark-2.3.1 进行学习，所以这里推荐 2.11.x 版本，下载 scala-2.11.8.msi 后点击下一步就可以了

可以不用手动配置环境变量：因为会自动配置好

Java 全手动配置环境变量
Python 在安装引导界面可以选择 add path 配置环境变量
Scala 完全不用理会，会自动配置好环境变量

https://www.scala-lang.org/download/all.html

Archive	System	Size
scala-2.11.8.tgz Linux平台二进制安装包	Mac OS X, Unix, Cygwin	27.35M
scala-2.11.8.msi windows平台一键安装包	Windows (msi installer)	109.35M
scala-2.11.8.zip windows二进制安装包	Windows	27.40M
scala-2.11.8.deb	Debian	76.02M
scala-2.11.8.rpm	RPM package	108.16M
scala-docs-2.11.8.txz	API docs	46.00M
scala-docs-2.11.8.zip API文档	API docs	84.21M
scala-sources-2.11.8.tar.gz 源码	Sources	

2.3、Linux 平台安装 Scala

访问官网的下载 Scala 的地址
https://www.scala-lang.org/download/all.html
https://www.scala-lang.org/download/2.11.8.html
https://downloads.lightbend.com/scala/2.11.8/scala-2.11.8.tgz
下载想要的版本，我们这里下载的是：
scala-2.11.8.tgz（Linux 安装版本）和 scala-2.11.8.msi（windows 一键安装版本）

然后解压 Scala 到指定目录
tar -zxvf scala-2.11.8.tgz -C /usr/local/scala

配置环境变量，将 scala 加入到 PATH 中
vi /etc/profile
export JAVA_HOME=/usr/local/java/jdk1.8.0_73
export PATH=$PATH:$JAVA_HOME/bin
export SCALA_HOME=/usr/local/scala/scala-2.11.8
export PATH=$PATH:$SCALA_HOME/bin
source /etc/profile

然后监测是否安装成功：

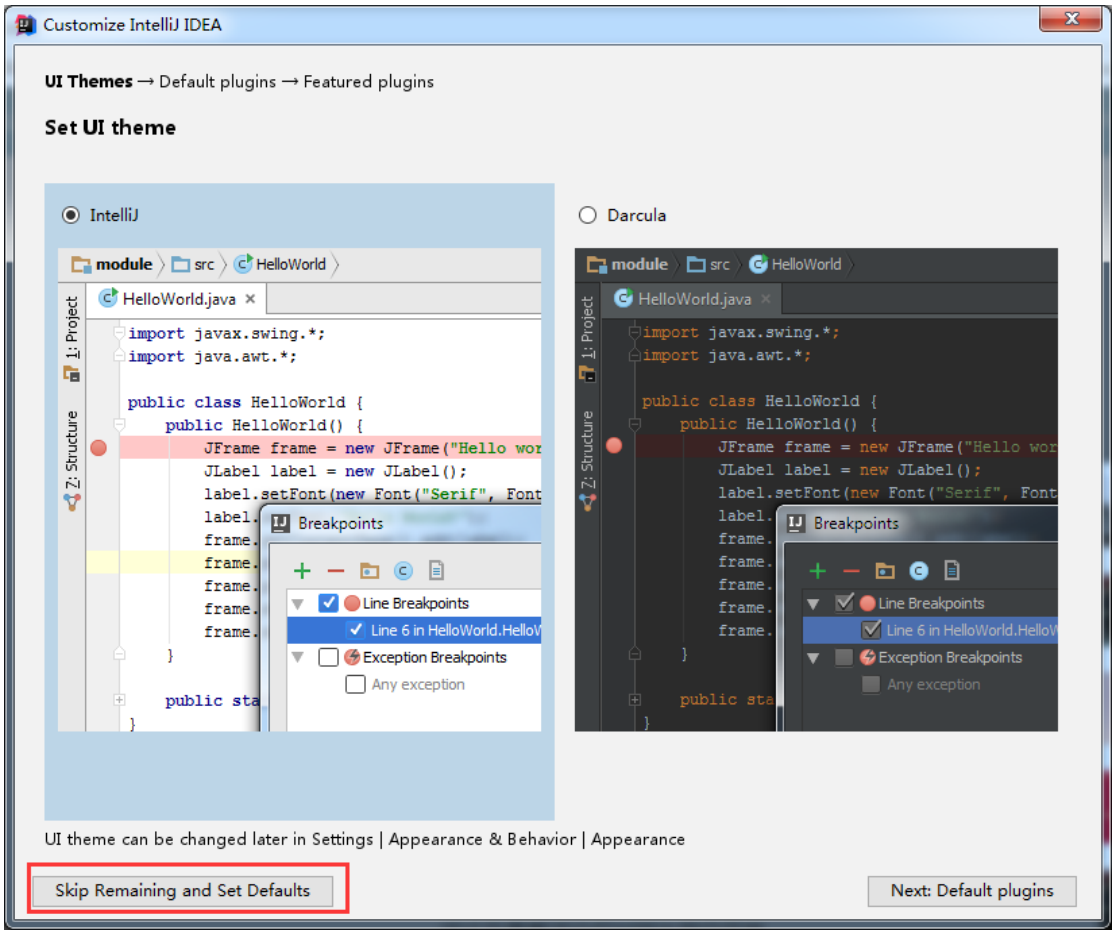
```
[root@hadoop05 ~]# java -version
java version "1.8.0_73"
Java(TM) SE Runtime Environment (build 1.8.0_73-b02)
Java HotSpot(TM) 64-Bit Server VM (build 25.73-b02, mixed mode)
[root@hadoop05 ~]# scala -version
Scala code runner version 2.11.8 -- Copyright 2002-2016, LAMP/EPFL
[root@hadoop05 ~]#
```

2.4、Scala 集成开发环境 Intellgence IDEA 安装

目前 Scala 的开发工具主要有两种：**Eclipse** 和 **IDEA**，这两个开发工具都有相应的 Scala 插件，如果使用 Eclipse，直接到 Scala 官网下载即可 http://scala-ide.org/download/sdk.html

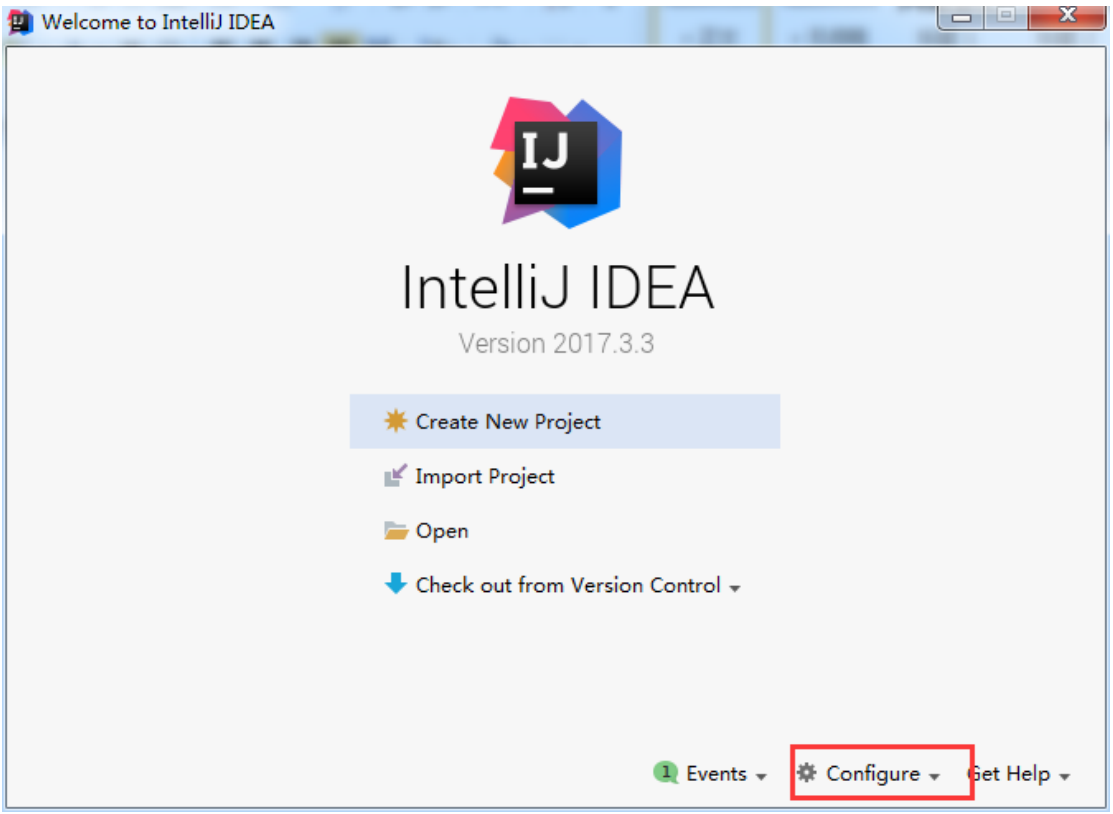
由于 IDEA 的 Scala 插件比 Eclipse 的更优秀，所以大多数 Scala 程序员都愿意选择 IDEA，可以到 http://www.jetbrains.com/idea/download/下载社区免费版，点击下一步安装即可，安装时如果有网络可以选择在线安装 Scala 插件。这里我们使用离线安装 Scala 插件：

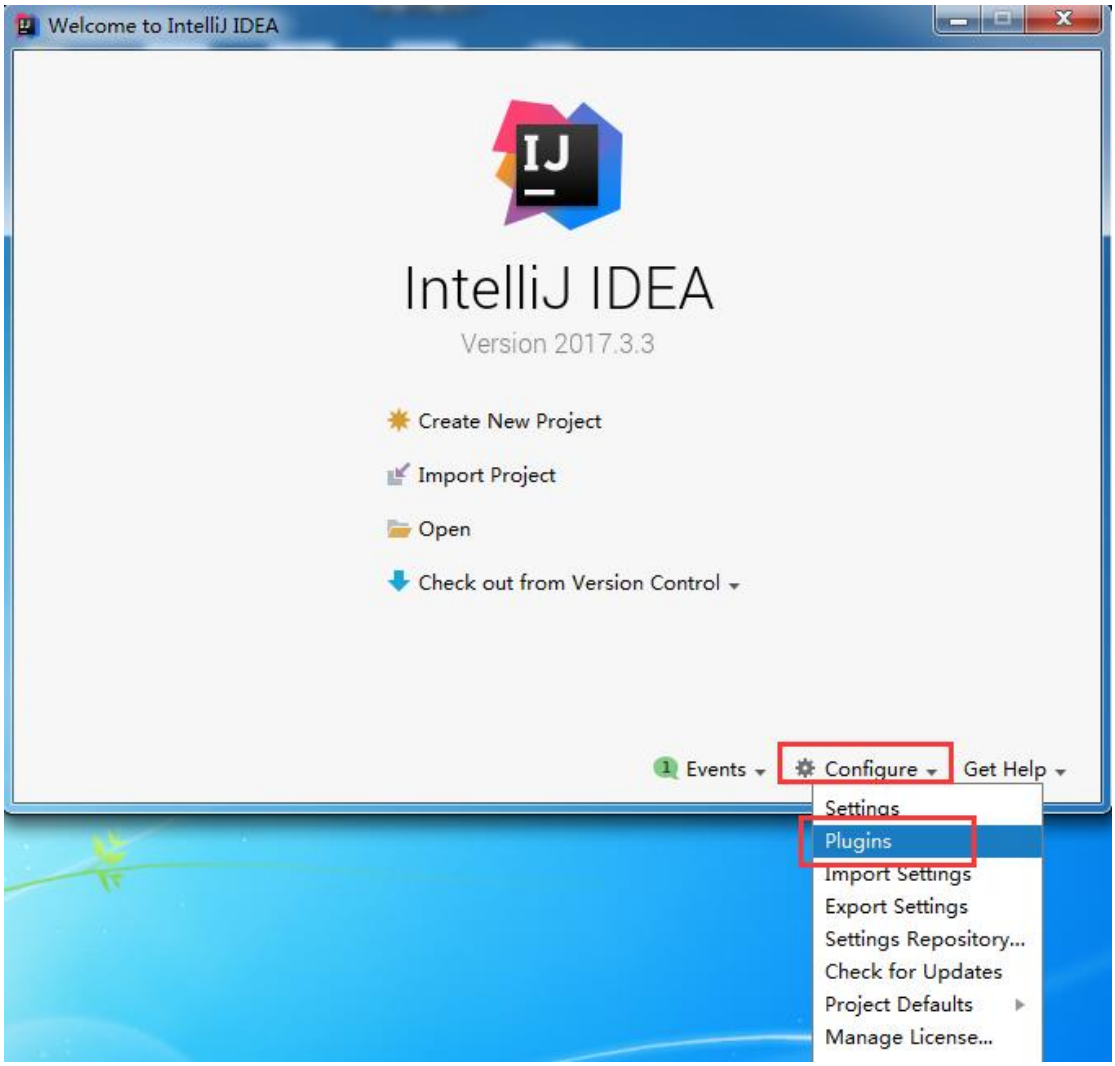
1、安装 IDEA，点击下一步即可。由于我们离线安装插件，所以点击 Skip All and Set Defaul
2、下载 IEDA 的 scala 插件，地址 http://plugins.jetbrains.com/?idea_ce 选择 scala 下载对应 IDEA 版本的 scala 插件：scala-intellij-bin-2017.3.3.zip
https://plugins.jetbrains.com/plugin/1347-scala

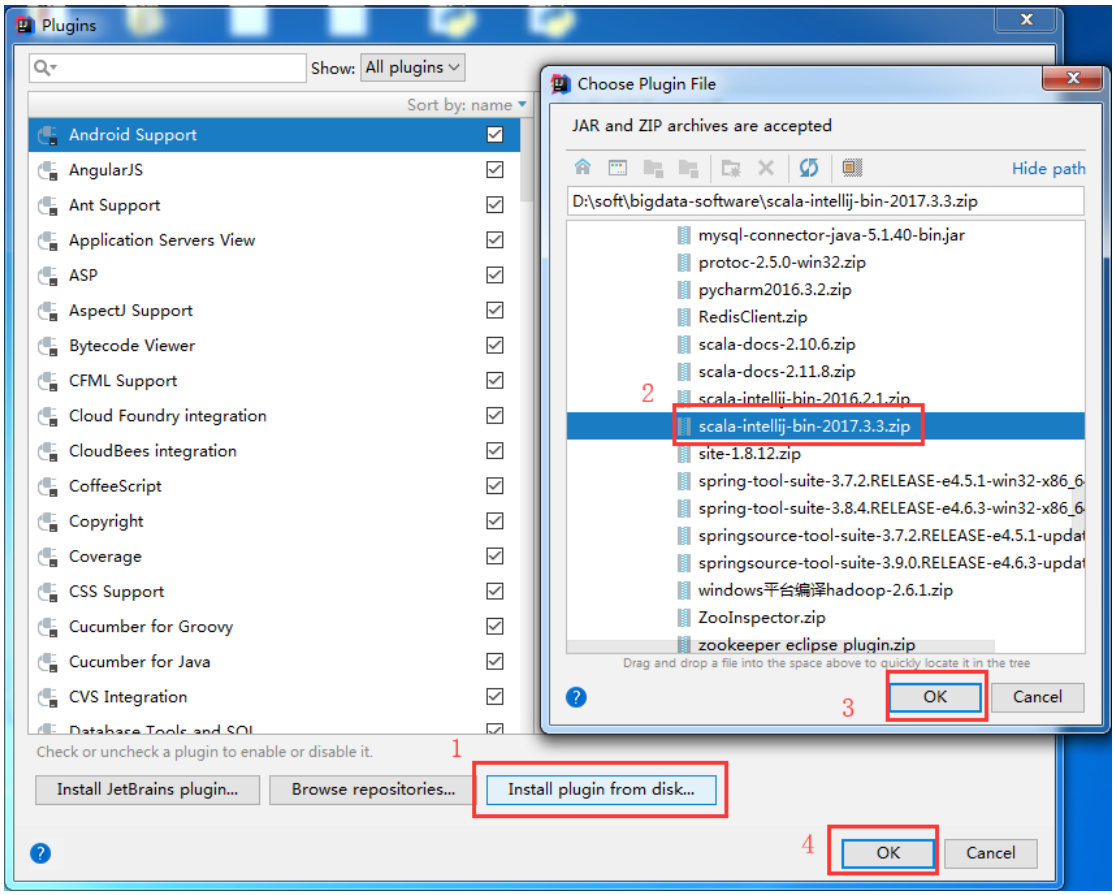


3、安装 Scala 插件：

Configure -> Plugins -> Install plugin from disk -> 选择 Scala 插件 -> OK -> 重启 IDEA


Welcome to IntelliJ IDEA
IntelliJ IDEA
Version 2017.3.3
Create New Project
Import Project
Open
Check out from Version Control
Events
Configure
Get Help


Welcome to IntelliJ IDEA
IntelliJ IDEA
Version 2017.3.3
Create New Project
Import Project
Open
Check out from Version Control
Events
Configure
Get Help
Settings
Plugins
Import Settings
Export Settings
Settings Repository...
Check for Updates
Project Defaults
Manage License...



点击 OK 之后，再重启就可以了

3、Scala 基础语法

3.1、Hello Scala

```
package com.mazh.scala.helloworld


/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */
object HelloWorld {
    // 作用类似于 Java 的 main 方法
    def main(args: Array[String]): Unit = {
    println("hello world")
  }
}
```

关于 HelloWorld 程序：Scala 和 Java 的对比不同处

1、文件后缀名

2、编译和运行方式
3、类的声明
4、main 方法的声明
5、分号

3.2、变量定义

```
package com.mazh.scala.core


/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */
object VariableDemo {
  def main(args: Array[String]) {
    //使用 val 定义的变量值是不可变的，相当于 java 里用 final 修饰的变量
    val i = 1
    //使用 var 定义的变量是可变得，在 Scala 中鼓励使用 val
    var s = "hello"
    //Scala 编译器会自动推断变量的类型，必要的时候可以指定类型
    //变量名在前，类型在后
    val str: String = "spark"
  }
}
```

总结：
1）数据类型可以指定，也可以不指定，如果不指定，那么就会进行数据类型的**自动推断**。
2）如果指定数据类型，数据类型的执行方式是在变量名后面写一个**冒号**，然后写上数据类型。
3）我们的 scala 里面变量的修饰符一共有两个，一个是 var，一个是 val
如果是 var 修饰的变量，那么这个变量的值是可以修改的
如果是 val 修饰的变量，那么这个变量的值是不可以修改的

懒加载：两个例子

```
scala> import scala.io.Source
import scala.io.Source

scala> lazy val file = Source.fromFile("/home/hadoop/student.txt")
file: scala.io.BufferedSource = <lazy>

scala> file
res2: scala.io.BufferedSource = non-empty iterator

scala> for (f <- file.getLines) println(f)
```

```
95002,刘晨,女,19,IS
95017,王风娟,女,18,IS
95018,王一,女,19,IS
95013,冯伟,男,21,CS
95014,王小丽,女,19,CS
95019,邢小丽,女,19,IS
95020,赵钱,男,21,IS
95003,王敏,女,22,MA
95004,张立,男,19,IS
95012,孙花,女,20,CS
95010,孔小涛,男,19,CS
95005,刘刚,男,18,MA
95006,孙庆,男,23,CS
95007,易思玲,女,19,MA
95008,李娜,女,18,CS
95021,周二,男,17,MA
95022,郑明,男,20,MA
95001,李勇,男,20,CS
95011,包小柏,男,18,MA
95009,梦圆圆,女,18,MA
95015,王君,男,18,MA
```

```
scala> import scala.io.Source
import scala.io.Source

scala> lazy val file = Source.fromFile("/home/hadoop/words.txt")
file: scala.io.BufferedSource = <lazy>

scala> for (f <- file.getLines) println(f)
hello huangbo
hello xuzheng
hello wangbaoqiang

scala> val file = Source.fromFile("/home/hadoop/words.txt")
file: scala.io.BufferedSource = non-empty iterator

scala> for (f <- file.getLines) println(f)
hello huangbo
hello xuzheng
hello wangbaoqiang

scala> lazy val aaa = 3 + 4
aaa: Int = <lazy>

scala> println(aaa)
7

scala> aaa
res3: Int = 7

scala>
```

3.3、数据类型

3.3.1、数据类型概述

Scala 和 Java 一样，有 7 种数值类型 Byte、Char、Short、Int、Long、Float 和 Double（无包装类型）和一个 Boolean 类型，再加上常用的 String 类型

数据类型：http://www.runoob.com/scala/scala-data-types.html

注意：scala 里面没有基本数据类型和包装类型之说。
如果大家非要说的话，那么大家都可以认为都有的类型的类型都是包装类型。

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 32767
Int	32位有符号补码整数。数值区间为 -2147483648 到 2147483647
Long	64位有符号补码整数。数值区间为 -9223372036854775808 到 9223372036854775807
Float	32位IEEE754单精度浮点数
Double	64位IEEE754单精度浮点数
Char	16位无符号Unicode字符, 区间值为 U+0000 到 U+FFFF
String	字符序列
Boolean	true或false
Unit	表示无值，和其他语言中void等同。用作不返回任何结果的方法的结果类型。Unit只有一个实例值，写成()。
Null	null 或空引用
Nothing	Nothing类型在Scala的类层级的最低端；它是任何其他类型的子类型。
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

```
package com.mazh.scala.core


/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */
object TypeDemo {

  def main(args: Array[String]): Unit = {

    val var_int = 1
    val var_double = 3.33
    val var_float = 2.4F
    val var_char = 'A'
    val var_bool = true

    val var_16 = 0x29
    val var_string = "aa"
    val var_string1 = "\"huangbo\""
    val var_string2 = """hello\thello\na\t\\"""
    println(var_string2)
  }
}
```

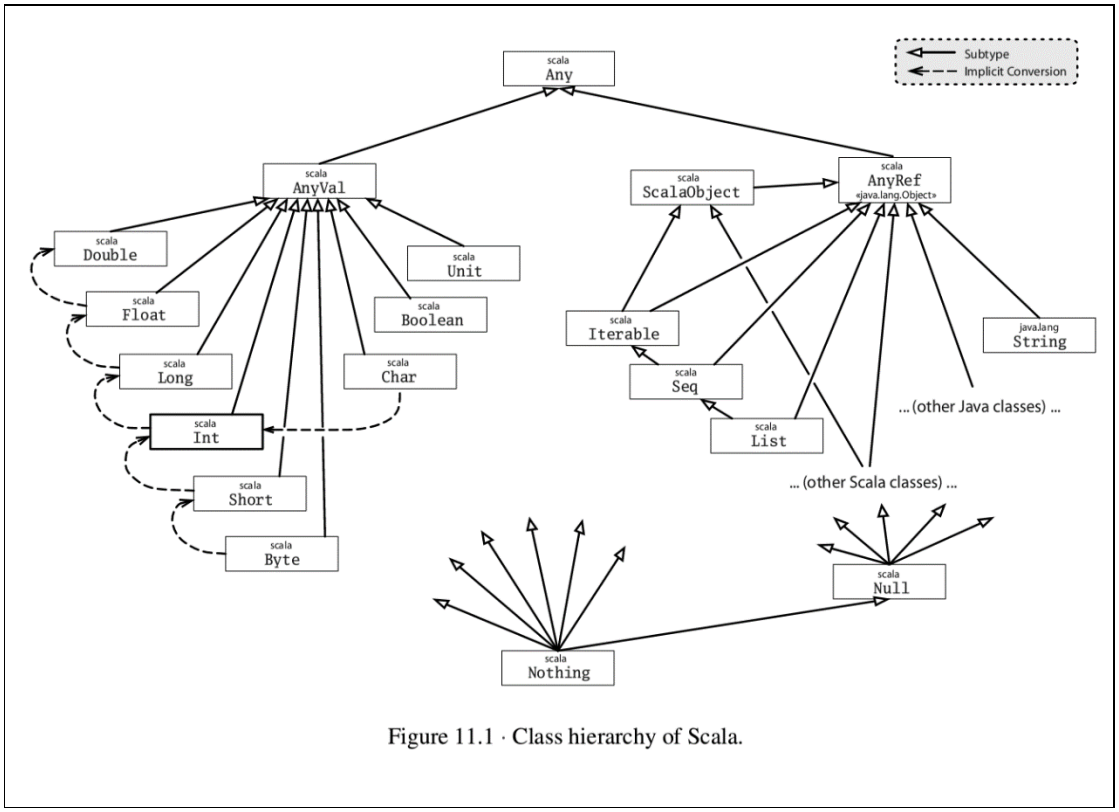


Figure 11.1 · Class hierarchy of Scala.

要点：

1、 Any 是所有类的父类，包括值类型 AnyVal，和引用类型 AnyRef

2、 AnyVal 是所有值类型的父类，包括 Int，Double，Boolean，Unit 等等

3、 AnyRef 是所有引用类型的父类，包括 Null

4、 Null 是所有引用类型的子类

5、 Nothing 是所有类的子类

6、 Unit 类型只有一个实例，是()，相当于 java 中的 void，没有任何的实质意义

7、 Null 也只有一个实例，是 null，相当于 java 中的 null，能赋值给任何引用类型变量，不能赋值给值类型变量

3.3.2、Scala 基本类型操作

算术操作：+ - * / % //
关系运算：> >= < <= == !=
逻辑运算：&& || !
位运算： & | ^ ~ >> << >>>
对象比较：1==1 1==1.0 "huangbo" == "huangbo" ne eq

特别注意：ne eq equals ==

3.4、编码规范

1、分号：在 scala 编码中，不强制在代码末尾加分号，但是如果有多句代码写在同一行，那

么必须使用分号进行隔开

2、注释：在 scala 编程中，注释的方式和 java 中注释方式一样，原则：少而精

3、关键字：关注新关键字：yield, match, object, def, implicit, trait, sealed, var/val

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
requires	return	sealed	super	this
throw	trait	try	true	type
val	var	while	with	yield
_	:	=	=>	<-	<:	<%	>:	#	@

3.5、流程控制--条件表达式 if

Scala 的的条件表达式比较简洁，例如：

```
package com.mazh.scala.core


/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */
object ConditionDemo {
  def main(args: Array[String]) {
    val x = 1
    //判断x 的值，将结果赋给y
    val y = if (x > 0) 1 else -1
    //打印y 的值
    println(y)

    //支持混合类型表达式
    val z = if (x > 1) 1 else "error"
    //打印z 的值
    println(z)

    //如果缺失else，相当于if (x > 2) 1 else ()
    val m = if (x > 2) 1
```

```
    println(m)

    //在scala中每个表达式都有值，scala中有个Unit类，写做(),相当于Java中的void
    val n = if (x > 2) 1 else ()
    println(n)

    //if 和else if
    val k = if (x < 0) 0
    else if (x >= 1) 1 else -1
    println(k)
  }
}
```

总结：

1）if 条件表达式它是有返回值的，返回值是多个分支的返回结果的共同父类

2）返回值会根据条件表达式的情况会进行自动的数据类型的推断（返回的是多个分支的共同父类）

3.6、块表达式

```
package com.mazh.scala.core

/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */
object BlockExpresstionDemo {
  def main(args: Array[String]) {
    val x = 0
    //在scala中{}中课包含一系列表达式，块中最后一个表达式的值就是块的值
    //下面就是一个块表达式
    val result = {
      if (x < 0){
        -1
      } else if(x >= 1) {
        1
      } else {
        "error"
      }
    }
    //result的值就是块表达式的结果
    println(result)
  }
}
```

总结：就算是赋值表达式，也是有返回值的。是空，是 Unit

```
scala> val a = {val bb = 2}
a: Unit = ()

scala> val a = {val bb = 2; 2}
a: Int = 2
```

3.7、流程控制--循环 for 和 while

在 scala 中有 for 循环和 while 循环，用 for 循环比较多

for 循环语法结构：for (i <- 表达式/数组/集合)

```
package com.mazh.scala.core


/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */
object ForDemo {
  def main(args: Array[String]) {
    //for(i <- 表达式),表达式 1 to 10 返回一个 Range（区间）
    //每次循环将区间中的一个值赋给 i
    for (i <- 1 to 10)
      println(i)

    //for(i <- 数组)
    val arr = Array("a", "b", "c")
    for (i <- arr)
      println(i)

    // 倒序打印
    for (str <- arr.reverse){
      println(str)
    }

    // 使用数组下标的方式进行打印
    for (i <- 0 to arr.length - 1){
      println(arr(i))
    }

    for (i <- 0 until arr.length)
      println(arr(i))

    println("-----------------")
    for (i <- 0 until (arr.length, 2))
      println(arr(i))
```

```
    //高级for 循环
    //每个生成器都可以带一个条件，注意：if 前面没有分号
    for(i <- 1 to 3; j <- 1 to 3 if i == j)
      println((10 * i + j) + " ")
    println()

    //for 推导式：如果for 循环的循环体以yield 开始，则该循环会构建出一个集合
    //每次迭代生成集合中的一个值
    val v = for (i <- 1 to 10) yield i * 10
    println(v)

  }
}
```

总结：

1、在 scala 里面没有运算符，都有的符号其实都是方法。

2、在 **scala 里面没有 ++ -- 的用法**

3、**for(i <- 表达式/数组/集合)**

4、在 for 循环里面我们是可以添加 if 表达式

5、有两个特殊表达式需要了解：

To	1 to 3	1 2 3
To	1 to (3,2)	1 3
Until	1 until 3	12

6、如果在使用 for 循环的时候，for 循环的时候我们需要获取，我们可以是使用 yield 关键字

While 循环测试例子：

```
package com.mazh.scala.core


/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */
object WhileDemo {
  def main(args: Array[String]) {
    var n = 10;
    while (n > 0) {
      println(n)
      n -= 1
    }
  }
}
```

总结：

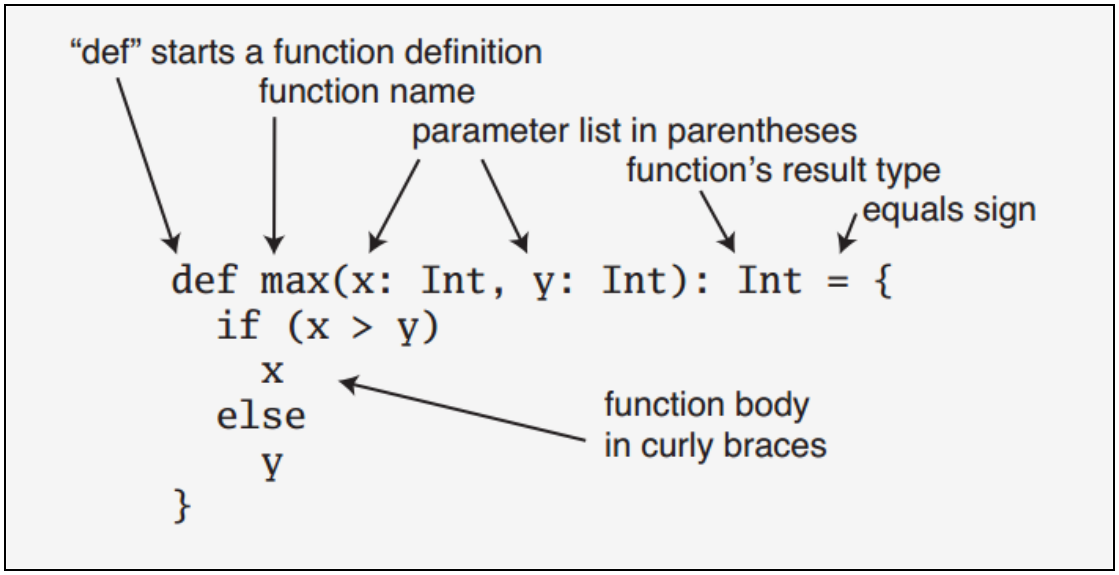
1）while 使用跟 java 一模一样

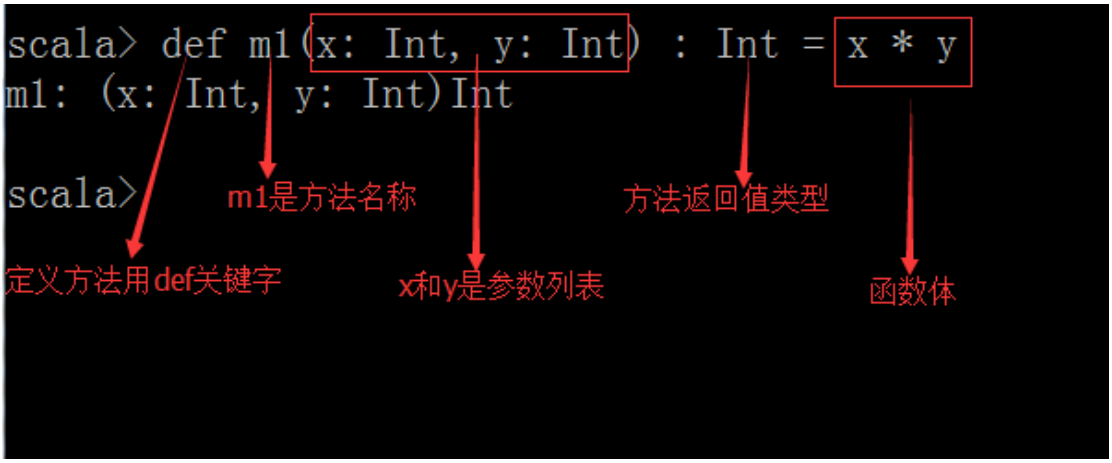
2）**注意点：在 scala 里面不支持 i++ i-- 等操作统一写成 i+=1 i-=1**

3.8、方法和函数

Scala 中的+ - * / %等操作符的作用与 Java 一样，位操作符 & | ^ >> <<也一样。
只是有一点特别的：这些操作符实际上是方法。例如：
a + b
是如下方法调用的简写：
a.+(b)
a 方法 b 可以写成 a.方法(b)

3.8.1、定义方法





方法的返回值类型可以不写，编译器可以自动推断出来，但是对于递归函数，必须指定返回类型
注意：函数体应该改成叫方法体!!! 如果不写等号，代表没有返回值。

```
scala> def m2(x:Int, y:Int){println("aa"); var aa = 2; aa}
m2: (x: Int, y: Int)Unit

scala>
```

3.8.2、定义函数

```
scala> def f1(x:Int, y:Int) = x + y
f1: (x: Int, y: Int)Int

scala> def f2(x:Int, y:Int):Int = x + y
f2: (x: Int, y: Int)Int

scala> f1(2,3)
res3: Int = 5

scala> f2(2,3)
res4: Int = 5

scala> val add = (x:Int, y:Int) => x + y
add: (Int, Int) => Int = <function2>

scala> add(2,3)
res5: Int = 5
```

这是方法

这是函数，表示两个Int类型的值经过计算变成一个Int的结果返回

函数的意义：表示接受两个 Int 类型的变量，然后做累加。
经过 scala 的自动类型推断得知，最后返回的结果数据的类型也是 Int。
Function2 中 2 表示这个函数接收的参数个数是 2 个。

3.8.3、方法和函数的区别

1、函数可以作为参数传递给方法，也就是说函数可以作为方法的参数
在函数式编程语言中，函数是“头等公民”，它可以像任何其他数据类型一样被传递和操作
案例：首先定义一个方法，再定义一个函数，然后将函数传递到方法里面

```
scala> def m2(f: (Int, Int) => Int) = f(2,6)
m2: (f: (Int, Int) => Int)Int

scala> val f2 = (x: Int, y: Int) => x - y
f2: (Int, Int) => Int = <function2>

scala> m2(f2)
res0: Int = -4
```

1.定义一个方法

2，定义一个函数

3.将函数作为参数传入到方法中

2、函数可以作为方法的参数，但是也可以作为函数的参数，例如：

```
scala> val aa = (x:Int) => x + 1
aa: Int => Int = <function1>

scala> val bb = (f:Int => Int, x:Int) => f(x) + 1
bb: (Int => Int, Int) => Int = <function2>

scala> bb(aa, 5)
res2: Int = 7
```

红框是函数，黄框也是函数，但是黄框函数是红框函数的参数

3、方法也可以作为方法的参数。在需要传入函数作为参数的位置上传入一个方法的话，那么这个方法会被自动的转换为函数作为参数，也可以通过“_”把方法转换为参数

```
scala> def mm(x:Int, y:Int):Int = x + y
mm: (x: Int, y: Int)Int

scala> def mm1(f:(Int, Int) => Int, x:Int, y:Int) = f(x, y)
mm1: (f: (Int, Int) => Int, x: Int, y: Int)Int

scala> mm1(mm, 2, 4)
res6: Int = 6

scala> mm1(mm _ , 2, 4)
res7: Int = 6
```

方法

方法，第一个参数是函数

需要函数，但是传入方法，会自动转换成函数

显示的通过下划线转成函数

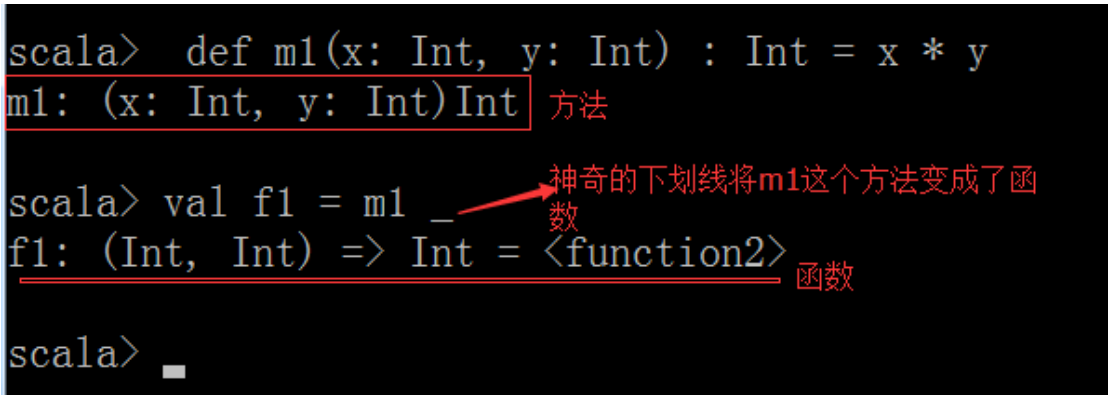
4、方法也可以作为函数的参数。其实，原理就是方法会被自动转换为函数，所以也就是传入一个函数到一个函数作为参数。

```
scala> def m1(x:Int, y:Int) = x + y
m1: (x: Int, y: Int)Int

scala> val f2 = (f:(Int, Int)=>Int, z:Int, b:Int) => f(z, b)
f2: ((Int, Int) => Int, Int, Int) => Int = <function3>

scala> f2(m1, 2, 3)
res6: Int = 5

scala> f2(m1 _, 2, 3)
res7: Int = 5
```

```scala
package com.aura.mazh.day1.core


/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */
object MethodAndFunctionDemo {
  //定义一个方法
  //方法m2 参数要求是一个函数，函数的参数必须是两个Int 类型
  //返回值类型也是Int 类型
  def m1(f: (Int, Int) => Int) : Int = f(2, 6)
```

```
  // 定义一个计算数据不被写死的方法
  def m2(f: (Int, Int) => Int, x:Int, y:Int) : Int = f(x, y)

  // 定义一个需要两个 Int 类型参数的方法
  def m3(x:Int, y:Int):Int = x + y

  //定义一个函数 f1，参数是两个 Int 类型，返回值是一个 Int 类型
  val f1 = (x: Int, y: Int) => x + y
  //再定义一个函数 f2
  val f2 = (m: Int, n: Int) => m * n

  //main 方法
  def main(args: Array[String]) {

    //调用 m1 方法，并传入 f1 函数
    val r1 = m1(f1)
    println(r1)

    //调用 m1 方法，并传入 f2 函数
    val r2 = m1(f2)
    println(r2)

    // 调用 m2 方法， 传入 f1 函数
    val result1 = m2(f1, 2, 4)
    println(result1)

    // 调用 m2 方法，传入 f2 函数
    val result2 = m2(f2, 2, 4)
    println(result2)

    // 调用 m2 方法， 传入 mm 方法作为参数
    println(m2(m3, 2, 4))
  }
}
```

3.8.4、将方法转换成函数使用



3.9、Scala 函数式编程特点

（1）高阶函数（Higher-order functions）
（2）闭包（closures）
（3）模式匹配（Pattern matching）
（4）单一赋值（Single assignment）
（5）延迟计算（Lazy evaluation）
（6）类型推导（Type inference）
（7）尾部调用优化（Tail call optimization）
（8）类型推导（Type inference）

4、Scala 数组 Array

4.1、定长数组和变长数组

1、由于 Array 是不可变（长度不可变）的，初始化之初就有了固定的长度，所以不能直接地对其元素进行删除操作，也不能多增加元素，只能修改某个位置的元素的值，要实现删除可以通过过滤生成新的 Array 的方式来删除不要的元素。所以也就没有 add,insert,remove 等操作。

2、而 ArrayBuffer 是可变的，本身提供了很多元素的操作，当然包括增加，删除操作。

3、如果你需要在 Array 和 ArrayBuffer 之间转换，那么分别调用 toBuffer()和 toArray()方法即可

```
package com.mazh.scala.core


import scala.collection.mutable.ArrayBuffer
```

```scala
/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */
object ArrayDemo {

  def main(args: Array[String]) {

    //初始化一个长度为8 的定长数组，其所有元素均为0
    val arr1 = new Array[Int](8)
    //直接打印定长数组，内容为数组的hashcode 值
    println(arr1)
    //将数组转换成数组缓冲，就可以看到原数组中的内容了
    //toBuffer 会将数组转换长数组缓冲
    println(arr1.toBuffer)

    //注意：如果new，相当于调用了数组的apply 方法，直接为数组赋值
    //初始化一个长度为1 的定长数组
    val arr2 = Array[Int](10)
    println(arr2.toBuffer)

    //定义一个长度为3 的定长数组
    val arr3 = Array("hadoop", "storm", "spark")
    //使用()来访问元素
    println(arr3(2))

    //////////////////////////////////////////////////
    //变长数组（数组缓冲）
    //如果想使用数组缓冲，需要导入import scala.collection.mutable.ArrayBuffer 包
    val ab = ArrayBuffer[Int]()
    //向数组缓冲的尾部追加一个元素
    //+=尾部追加元素
    ab += 1
    //追加多个元素
    ab += (2, 3, 4, 5)
    //追加一个数组++=
    ab ++= Array(6, 7)
    //追加一个数组缓冲
    ab ++= ArrayBuffer(8,9)
    //打印数组缓冲ab

    //在数组某个位置插入元素用insert
    ab.insert(0, -1, 0)
    //删除数组某个位置的元素用remove
```

```
    ab.remove(8, 2)


    println(ab)
  }
}
```

```
scala> import scala.collection.mutable.ArrayBuffer
import scala.collection.mutable.ArrayBuffer

scala> var intArrayVar=ArrayBuffer(1,1,2)
intArrayVar: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(1, 1, 2)

scala> intArrayVar.insert(0,6)

scala> intArrayVar
res24: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(6, 1, 1, 2)

scala> intArrayVar.insert(0,7,8,9)

scala> intArrayVar
res26: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(7, 8, 9, 6, 1, 1, 2)

scala> intArrayVar.remove(0,4)

scala> intArrayVar += 3
res28: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(1, 1, 2, 3)

scala> intArrayVar += (4,5,6,7)
res29: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(1, 1, 2, 3, 4, 5, 6, 7)

scala> intArrayVar ++= Array(8,9,10)
res30: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

scala> intArrayVar ++= List(8,9,10)
res31: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 8, 9, 10)

scala> intArrayVar.tr
transform   transpose   trimEnd   trimStart

scala> intArrayVar.trimEnd(3)

scala> intArrayVar
res33: scala.collection.mutable.ArrayBuffer[Int] = ArrayBuffer(1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

scala> intArrayVar.toArray
res34: Array[Int] = Array(1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

scala> res34.toBuffer
res35: scala.collection.mutable.Buffer[Int] = ArrayBuffer(1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

4.2、遍历数组

1、增强 for 循环

2、使用 to 可以生成序列，**0 to 10 包含 0 包含 10**

3、好用的 until 会生成脚标，**0 until 10 包含 0 不包含 10**

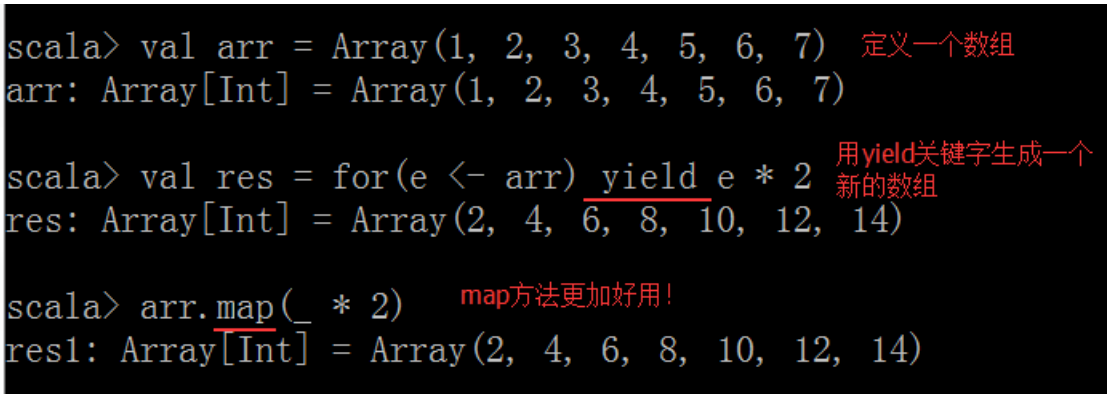
```
scala> 0 until 10
res0: scala.collection.immutable.Range = Range(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

```
package com.mazh.scala.core


/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */
object ArrayForDemo {


  def main(args: Array[String]) {
    //初始化一个数组
    val arr = Array(1,2,3,4,5,6,7,8)
```

```
    //增强for循环
    for(i <- arr)
      println(i)

    //使用to可以生成一个序列作为脚标
    for(i <- (0 to arr.length - 1).reverse)
      println(arr(i))

    //好用的until会生成一个Range,reverse是将前面生成的Range反转
    for(i <- (0 until arr.length).reverse)
      println(arr(i))

    //步长为2
    for(i <- (0 until (arr.length, 2)).reverse)
      println(arr(i))
  }
}
```

4.3、数组转换

yield 关键字将原始的数组进行转换会产生一个新的数组，原始的数组不变




```
package com.mazh.scala.core


/**
  * 作者:  马中华:http://blog.csdn.net/zhongqi2513
  */
object ArrayYieldDemo {
  def main(args: Array[String]) {
    //定义一个数组
    val arr = Array(1, 2, 3, 4, 5, 6, 7, 8, 9)
    //将偶数取出乘以10后再生成一个新的数组
    val res = for (e <- arr if e % 2 == 0) yield e * 10
    println(res.toBuffer)
```

```
    //更高级的写法,用着更爽
    //filter 是过滤，接收一个返回值为boolean 的函数
    //map 相当于将数组中的每一个元素取出来，应用传进去的函数
    val r = arr.filter(_ % 2 == 0).map(_ * 10)
    println(r.toBuffer)
  }
}
```

4.4、数组常用算法

在 Scala 中，数组上的某些方法对数组进行相应的操作非常方便！

```
scala> val intArr=Array(1,2,3,4,5,6,7,8,9,10)
intArr: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

scala> intArr.sum
res37: Int = 55

scala> intArr.max
res38: Int = 10

scala> intArr.min
res39: Int = 1

scala> intArr.toString()
res40: String = [I@79d7ae7d

scala> intArr.mkString(",")
res41: String = 1,2,3,4,5,6,7,8,9,10

scala> intArr.mkString("<",",",">")
res42: String = <1,2,3,4,5,6,7,8,9,10>
```

4.5、多维数组

```
scala> var multiDimArr=Array(Array(1,2,3),Array(2,3,4))
multiDimArr: Array[Array[Int]] = Array(Array(1, 2, 3), Array(2, 3, 4))

scala> multiDimArr(0)(2)
res43: Int = 3

scala> for(i <- multiDimArr) println( i.mkString(","))
1,2,3
2,3,4

scala>
```

5、Scala 集合相关

5.1、Scala 集合

Scala 的集合有三大类：序列 Seq、集合 Set、映射 Map，所有的集合都扩展自 Iterable 特质
在 Scala 中集合有可变（mutable）和不可变（immutable）两种类型，immutable 类型的集合

初始化后就不能改变了（注意与 val 修饰的变量进行区别）

官网解释：

Scala collections systematically distinguish between mutable and immutable collections. A mutable collection can be updated or extended in place. This means you can change, add, or remove elements of a collection as a side effect. Immutable collections, by contrast, never change. You have still operations that simulate additions, removals, or updates, but those operations will in each case return a new collection and leave the old collection unchanged.

//大致意思是：Scala 中的集合分为两种，一种是可变的集合，另一种是不可变的集合

//可变的集合可以更新或修改，添加、删除、修改元素将作用于原集合

//不可变集合一量被创建，便不能被改变，添加、删除、更新操作返回的是新的集合，老集合保持不变

https://blog.csdn.net/zhongqi2513/article/details/82956813

val 和 var：表明定义的变量（引用）是否能被修改而指向其他内容

immutable 和 mutable：表明的是内存中开辟出来的这块空间里的内容能否被修改，如果针对 immutable 变量进行修改，其实是开辟了一块新的内存空间，产生了一个新的变量，而原来的变量依然没有改变

5.2、Scala 序列--List

不可变的序列

import scala.collection.immutable._

在 Scala 中列表要么为空（Nil 表示空列表）要么是一个 head 元素加上一个 tail 列表。

9 :: List(5, 2)　:: 操作符是将给定的头和尾创建一个新的列表

注意：:: 操作符是右结合的，如 9 :: 5 :: 2 :: Nil 相当于 9 :: (5 :: (2 :: Nil))

```
package com.mazh.scala.core

/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */
object ImmutableListDemo {

  def main(args: Array[String]) {
    //创建一个不可变的集合
    val lst1 = List(1,2,3)

    //将 0 插入到 lst1 的前面生成一个新的 List
    val lst2 = 0 :: lst1
    val lst3 = lst1.::(0)
    val lst4 = 0 +: lst1
    val lst5 = lst1.+:(0)
```

```
    //将一个元素添加到lst1的后面产生一个新的集合
    val lst6 = lst1 :+ 3

    val lst0 = List(4,5,6)
    //将2个list合并成一个新的List
    val lst7 = lst1 ++ lst0
    //将lst0插入到lst1前面生成一个新的集合
    val lst8 = lst1 ++: lst0

    //将lst0插入到lst1前面生成一个新的集合
    val lst9 = lst1.:::(lst0)

    println(lst9)
  }
}
```

序列常用操作：

```
//采用::及 Nil 进行列表构建
scala> val nums = 1 :: (2 :: (3 :: (4 :: Nil)))
nums: List[Int] = List(1, 2, 3, 4)

//由于::操作符的优先级是从右往左的，因此上一条语句等同于下面这条语句
scala> val nums=1::2::3::4::Nil
nums: List[Int] = List(1, 2, 3, 4)

//判断是否为空
scala> nums.isEmpty
res108: Boolean = false

//取第一个无素
scala> nums.head
res109: Int = 1

//取除第一个元素外剩余的元素，返回的是列表
scala> nums.tail
res114: List[Int] = List(2, 3, 4)

//取列表第二个元素
scala> nums.tail.head
res115: Int = 2

//插入排序算法实现
def isort(xs: List[Int]): List[Int] =
if (xs.isEmpty) Nil
```

```
else insert(xs.head, isort(xs.tail))

def insert(x: Int, xs: List[Int]): List[Int] =
if (xs.isEmpty || x <= xs.head) x :: xs
else xs.head :: insert(x, xs.tail)

//List 连接操作
scala> List(1,2,3):::List(4,5,6)
res116: List[Int] = List(1, 2, 3, 4, 5, 6)

//取除最后一个元素外的元素，返回的是列表
scala> nums.init
res117: List[Int] = List(1, 2, 3)

//取列表最后一个元素
scala> nums.last
res118: Int = 4

//列表元素倒置
scala> nums.reverse
res119: List[Int] = List(4, 3, 2, 1)

//一些好玩的方法调用
scala> nums.reverse.reverse==nums
res120: Boolean = true

scala> nums.reverse.init
res121: List[Int] = List(4, 3, 2)

scala> nums.tail.reverse
res122: List[Int] = List(4, 3, 2)

//丢弃前 n 个元素
scala> nums drop 3
res123: List[Int] = List(4)

scala> nums drop 1
res124: List[Int] = List(2, 3, 4)

//获取前 n 个元素
scala> nums take 1
res125: List[Int] = List(1)

scala> nums.take(3)
```

```
res126: List[Int] = List(1, 2, 3)

//将列表进行分割
scala> nums.splitAt(2)
res127: (List[Int], List[Int]) = (List(1, 2),List(3, 4))

//前一个操作与下列语句等同
scala> (nums.take(2),nums.drop(2))
res128: (List[Int], List[Int]) = (List(1, 2),List(3, 4))

//Zip 操作
scala> val nums=List(1,2,3,4)
nums: List[Int] = List(1, 2, 3, 4)

scala> val chars=List('1','2','3','4')
chars: List[Char] = List(1, 2, 3, 4)

//返回的是 List 类型的元组(Tuple）
scala> nums zip chars
res130: List[(Int, Char)] = List((1,1), (2,2), (3,3), (4,4))

//List toString 方法
scala> nums.toString
res131: String = List(1, 2, 3, 4)

//List mkString 方法
scala> nums.mkString
res132: String = 1234

//转换成数组
scala> nums.toArray
res134: Array[Int] = Array(1, 2, 3, 4)
```

可变的序列

import scala.collection.mutable._

```
package com.mazh.scala.core


import scala.collection.mutable.ListBuffer


/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */
object MutableListDemo extends App{
  //构建一个可变列表，初始有 3 个元素 1,2,3
```

```
  val lst0 = ListBuffer[Int](1,2,3)
  //创建一个空的可变列表
  val lst1 = new ListBuffer[Int]
  //向 lst1 中追加元素，注意：没有生成新的集合
  lst1 += 4
  lst1.append(5)

  //将 lst1 中的元素最近到 lst0 中， 注意：没有生成新的集合
  lst0 ++= lst1

  //将 lst0 和 lst1 合并成一个新的 ListBuffer 注意：生成了一个集合
  val lst2= lst0 ++ lst1

  //将元素追加到 lst0 的后面生成一个新的集合
  val lst3 = lst0 :+ 5
}
```

5.3、Scala 集合--Set

不可变的 Set

```
package com.mazh.scala.core

import scala.collection.immutable.HashSet

/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  */
object ImmutableSetDemo extends App{
  val set1 = new HashSet[Int]()
  //将元素和 set1 合并生成一个新的 set，原有 set 不变
  val set2 = set1 + 4
  //set 中元素不能重复
  val set3 = set1 ++ Set(5, 6, 7)
  val set0 = Set(1,3,4) ++ set1
  println(set0.getClass)
}
```

可变的 Set

```
package com.mazh.scala.core

import scala.collection.mutable

/**
```

```
 * 作者:  马中华：http://blog.csdn.net/zhongqi2513
 */
object MutableSetDemo extends App{
  //创建一个可变的HashSet
  val set1 = new mutable.HashSet[Int]()
  //向HashSet 中添加元素
  set1 += 2
  //add 等价于+=
  set1.add(4)
  set1 ++= Set(1,3,5)
  println(set1)
  //删除一个元素
  set1 -= 5
  set1.remove(2)
  println(set1)
}
```

怎么求集合的交集并集差集呢？

```
scala> val set1 = Set(1,2,3,4,5)
set1: scala.collection.immutable.Set[Int] = Set(5, 1, 2, 3, 4)

scala> val set2 = Set(3,4,5,6,7)
set2: scala.collection.immutable.Set[Int] = Set(5, 6, 7, 3, 4)

scala> set1 union set2
res16: scala.collection.immutable.Set[Int] = Set(5, 1, 6, 2, 7, 3, 4)

scala> set1 diff set2
res17: scala.collection.immutable.Set[Int] = Set(1, 2)

scala> set1 intersect set2
res18: scala.collection.immutable.Set[Int] = Set(5, 3, 4)
```

5.4、Scala 集合--Map

```
package com.mazh.scala.core


import scala.collection.mutable


/**
 * 作者:  马中华：http://blog.csdn.net/zhongqi2513
 */
object MutableMapDemo extends App{
  val map1 = new mutable.HashMap[String, Int]()
  //向map 中添加数据
  map1("spark") = 1
  map1 += (("hadoop", 2))
```

```
  map1.put("storm", 3)
  println(map1)

  //从map中移除元素
  map1 -= "spark"
  map1.remove("hadoop")
  println(map1)
}
```

5.5、Scala 映射—Map

在 Scala 中，把哈希表这种数据结构叫做映射，在 Java 中也叫做映射
在 Python 中，把哈希表这种数据结构叫做字典
不管叫什么，存储的都是键值对形式的值

5.5.1、构建 Map

```
scala> val scores = Map("tom" -> 85, "jerry" -> 99, "kitty" -> 90)
scores: scala.collection.immutable.Map[String,Int] = Map(tom -> 85, jerry -> 99,
 kitty -> 90)    第一种创建Map的方式，用箭头

scala> val scores = Map(("tom", 85), ("jerry", 99), ("kitty", 90))
scores: scala.collection.immutable.Map[String,Int] = Map(tom -> 85, jerry -> 99,
 kitty -> 90)    第二种创建Map的方式，用元组
```

5.5.2、获取和修改 Map 中的值

```
scala> val scores = Map(("tom", 85), ("jerry", 99), ("kitty", 90))
scores: scala.collection.immutable.Map[String,Int] = Map(tom -> 85, jerry -> 99,
 kitty -> 90)

scala> scores("jerry")    获取映射中的值
res0: Int = 99
```

好用的 getOrElse

```
scala> scores.getOrElse("suke", 0)
res2: Int = 0 如果映射中有值，返回映射中的值，没有就返回默认值
```

注意：在 Scala 中，有两种 Map，一个是 immutable 包下的 Map，该 Map 中的内容不可变；另一个是 mutable 包下的 Map，该 Map 中的内容可变
例子：

```
scala> import scala.collection.mutable.Map 首先导入mutable包
import scala.collection.mutable.Map

scala> val scores = Map("tom" -> 80, "jerry" -> 99) val定义的scores变量意味着变量的引用不变，但是Map中的内容可变
scores: scala.collection.mutable.Map[String,Int] = Map(tom -> 80, jerry -> 99)

scala> scores("tom") = 88   修改Map中的内容

scala> scores += ("kitty" -> 99, "suke" -> 60)  用+=向原来的Map中追加元素
res9: scores.type = Map(tom -> 88, kitty -> 99, jerry -> 99, suke -> 60)
```

注意：通常我们在创建一个集合是会用 val 这个关键字修饰一个变量（相当于 java 中的 final），那么就意味着该变量的引用不可变，该引用中的内容是不是可变，取决于这个引用指向的集合的类型

Map 常用操作：

```
scala> val studentInfo=Map("john" -> 21, "stephen" -> 22,"lucy" -> 20)
studentInfo: scala.collection.immutable.Map[String,Int] = Map(john -> 21, stephen -> 22, lucy -> 20)

scala> studentInfo.clear()
<console>:13: error: value clear is not a member of scala.collection.immutable.Map[String,Int]
              studentInfo.clear()
                          ^

scala> val studentInfoMutable=scala.collection.mutable.Map("john" -> 21, "stephen" -> 22,"lucy" -> 20)
studentInfoMutable: scala.collection.mutable.Map[String,Int] = Map(john -> 21, lucy -> 20, stephen -> 22)

scala> studentInfoMutable.clear()

scala> studentInfoMutable
res2: scala.collection.mutable.Map[String,Int] = Map()

scala> for( i <- studentInfoMutable ) println(i)

scala> val studentInfoMutable=scala.collection.mutable.Map("john" -> 21, "stephen" -> 22,"lucy" -> 20)
studentInfoMutable: scala.collection.mutable.Map[String,Int] = Map(john -> 21, lucy -> 20, stephen -> 22)

scala> for( i <- studentInfoMutable ) println(i)
(john,21)
(lucy,20)
(stephen,22)

scala> studentInfoMutable.foreach(e=> println(e._1+":"+e._2))
john:21
lucy:20
stephen:22
```

```
scala> val xMap=new scala.collection.mutable.HashMap[String,Int]()
xMap: scala.collection.mutable.HashMap[String,Int] = Map()

scala> xMap.put("spark",1)
res6: Option[Int] = None

scala> xMap.put("spark",1)
res7: Option[Int] = Some(1)

scala> xMap.contains("spark")
res8: Boolean = true

scala> val xMap=scala.collection.mutable.Map(("spark",1),("hive",1))
xMap: scala.collection.mutable.Map[String,Int] = Map(spark -> 1, hive -> 1)

scala> "spark" -> 1
res9: (String, Int) = (spark,1)

scala> xMap.get("spark")
res10: Option[Int] = Some(1)

scala> xMap.get("SparkSQL")
res11: Option[Int] = None
```

Option, None, Some 类型

Option、None、Some 是 scala 中定义的类型，它们在 scala 语言中十分常用，因此这三个类型非学重要。 None、Some 是 Option 的子类，它主要解决值为 null 的问题，在 java 语言中，对于定义好的 HashMap，如果 get 方法中传入的键不存在，方法会返回 null，在编写代码的时候对于 null 的这种情况通常需要特殊处理，然而在实际中经常会忘记，因此它很容易引起 NullPointerException 异常。在 Scala 语言中通过 Option、None、Some 这三个类来避免这样的问题，这样做有几个好处，首先是代码可读性更强，当看到 Option 时，我们自然而然就知道它的值是可选的，然后变量是 Option，比如 Option[String]的时候，直接使用 String 的话，编译直接通不过。

5.6、Scala 元组--Tuple

映射是 K/V 对偶的集合，对偶是元组的最简单形式，元组可以装着多个不同类型的值。

5.6.1、创建元组

```
scala> val t = ("hadoop", 3.14, 65535)
t: (String, Double, Int) = (hadoop,3.14,65535)

scala>
```

定义元组时用小括号将多个元素包起来，元素之间用逗号分隔
元素的类型可以不一样，元素个数可以任意多个

5.6.2、获取元组中的值

```
scala> val t,(a,b,c) = ("hadoop", 3.14, 65535)
t: (String, Double, Int) = (hadoop,3.14,65535)
a: String = hadoop
b: Double = 3.14
c: Int = 65535

scala> val r1 = t._1
r1: String = hadoop

scala> val r2 = t._2
r2: Double = 3.14

scala>
```

获取元组中的元素可以使用下划线加脚标
但是需要注意的是元组中的元素脚标是从1开始的

5.6.3、将对偶的元组转成集合

```
scala> val arr = Array(("tom", 88), ("jerry", 95))
arr: Array[(String, Int)] = Array((tom,88), (jerry,95))

scala> arr.toMap
res2: scala.collection.immutable.Map[String,Int] = Map(tom -> 88, jerry -> 95)
```

toMap可以将对偶的集合转换成映射

5.6.4、元组拉链操作

zip 命令可以将多个值绑定在一起

```
scala> val scores = Array(88, 95, 80)
scores: Array[Int] = Array(88, 95, 80)

scala> val ns = names.zip(scores)    使用zip将对应的值绑定到一起
ns: Array[(String, Int)] = Array((tom,88), (jerry,95), (kitty,80))

scala> ns.toMap
res4: scala.collection.immutable.Map[String,Int] = Map(tom -> 88, jerry -> 95, k
itty -> 80)
```

注意：如果两个数组的元素个数不一致，拉链操作后生成的数组的长度为较小的那个数组的元素个数

6、Scala 编程练习

6.1、99 乘法表

```
package com.mazh.scala.funny


object Table99 {


  def main(args: Array[String]): Unit = {
    for (i <- 1 to 9){
      for (j <- 1 to i){
        printf("%d*%d=%2d\t", i, j, (i*j))
      }
      println()
    }
  }
}
```

其实也可以利用 scala 的语法特性，使用一句代码完成：

for(a <- 1 to 9; b <- 1 to a) printf("%d*%d=%d%s", a, b, a*b, if(a==b) "\n" else "\t")

或者：

for(x<- 1 to 9;y<- 1 to x)print(s"$y*$x=${x*y}${if(x==y) "\n" else "\t"}")

```
scala> for(a <- 1 to 9; b <- 1 to a) printf("%d*%d=%d%s", a, b, a*b, if(a==b) "\n" else "\t")
1*1=1
2*1=2   2*2=4
3*1=3   3*2=6   3*3=9
4*1=4   4*2=8   4*3=12  4*4=16
5*1=5   5*2=10  5*3=15  5*4=20  5*5=25
6*1=6   6*2=12  6*3=18  6*4=24  6*5=30  6*6=36
7*1=7   7*2=14  7*3=21  7*4=28  7*5=35  7*6=42  7*7=49
8*1=8   8*2=16  8*3=24  8*4=32  8*5=40  8*6=48  8*7=56  8*8=64
9*1=9   9*2=18  9*3=27  9*4=36  9*5=45  9*6=54  9*7=63  9*8=72  9*9=81

scala> for(x<- 1 to 9;y<- 1 to x )print(s"$y*$x=${x*y}${if(x==y) "\n" else "\t"}")
1*1=1
1*2=2   2*2=4
1*3=3   2*3=6   3*3=9
1*4=4   2*4=8   3*4=12  4*4=16
1*5=5   2*5=10  3*5=15  4*5=20  5*5=25
1*6=6   2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
1*7=7   2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49
1*8=8   2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64
1*9=9   2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81

scala>
```

6.2、Scala 版本的 WordCount

现在有一个数组变量读入了一段话形成了一个数组，求出这段话中的每个单词的出现次数，简单来说，就是单词统计：

val array = Array("hello huangbo", "hello xuzheng", "hello wangbaoqiang")

结果：

array.flatMap(_.split(" ")).map((_,1)).groupBy(t => t._1).map(t =>(t._1,t._2.length))
.toList.sortBy(t => t._2).reverse

6.3、Scala 版本的插入排序 InsertSort

```scala
package com.mazh.scala.sort

import scala.util.control.Breaks

/**
  * 作者:  马中华：http://blog.csdn.net/zhongqi2513
  * Scala 版本的插入排序
  */
object InsertSort {

  def main(args: Array[String]): Unit = {

    val array = Array(4,12,6,3,8,9,5)
    val ab = array.toBuffer

    // 创建 Breaks 对象
```

```
    val forLoop = new Breaks

    for (i <- 1 until ab.length){
      val value_i = ab(i)

      // 把需要可能break 的代码放在 breakable 中执行
      forLoop.breakable{
        for (j <- 0 to i - 1){

          val value_j = ab(j)
          if (value_j > value_i){
            ab.remove(i, 1)
            ab.insert(j, value_i)

            // 使用break 进行跳出
            forLoop.break()
          }
        }
      }
    }
    println(ab)
  }
}
```