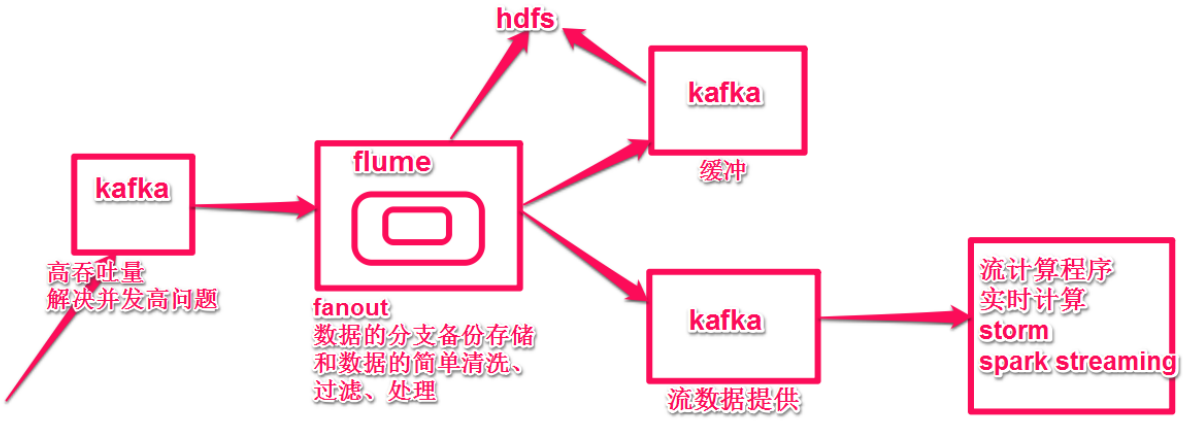
Day25_Scala的安装与基础操作

大数据-张军锋 Day25 Scala 安装 基础语法

总结Kafka和Flume



Scala的安装

Scala是一门多范式的编程语言，一种类似java的编程语言 ，设计初衷是实现可伸缩的语言、并集成面向对象编程和函数式编程的各种特性。

下载地址：http://www.scala-lang.org/download/2.11.1.html

Archive	System	Size
scala-2.11.1.tgz	Mac OS X, Unix, Cygwin	24.50M
scala-2.11.1.msi	Windows (msi installer)	93.05M
scala-2.11.1.zip	Windows	24.51M
scala-2.11.1.deb	Debian	92.01M
scala-2.11.1.rpm	RPM package	91.98M
scala-docs-2.11.1.txz	API docs	39.51M
scala-docs-2.11.1.zip	API docs	70.83M
scala-sources-2.11.1.tar.gz	sources	

linux

windows

License
The Scala distribution is released under the 3-clause BSD license.

1. **Java 设置**
   确保你本地以及安装了 JDK 1.5 以上版本，并且设置了 JAVA_HOME 环境变量及 JDK 的bin目录。
   我们可以使用以下命令查看是否安装了 Java和Java编译器：

```
java -version
java version "1.8.0_31"
Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)

javac -version
javac 1.8.0_31
```

2. **Scala安装**
   解压文件，配置SCALA_HOME和Path即可

执行 scala 命令，输出以下信息，表示安装成功：

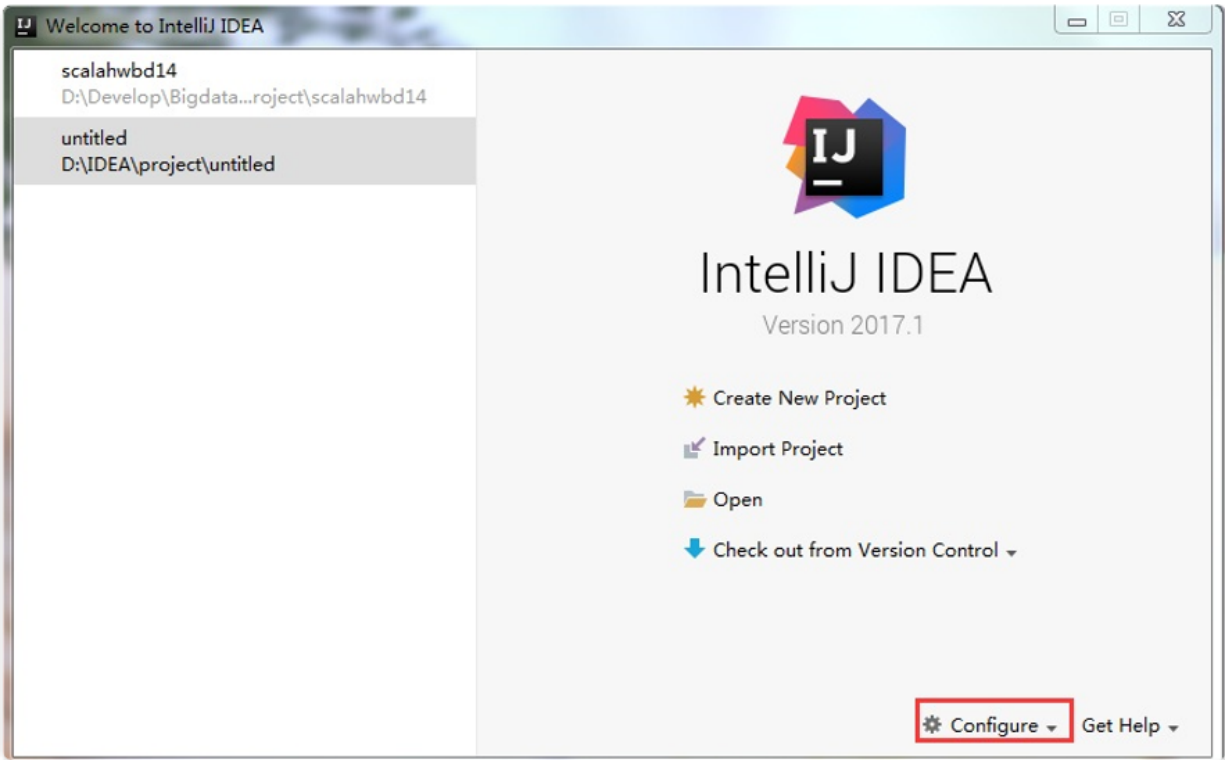
```
scala
Welcome to Scala version 2.11.11 (Java HotSpot(TM) 64-Bit Server V
M, Java 1.8.0_31).
Type in expressions to have them evaluated.
Type :help for more information.
```

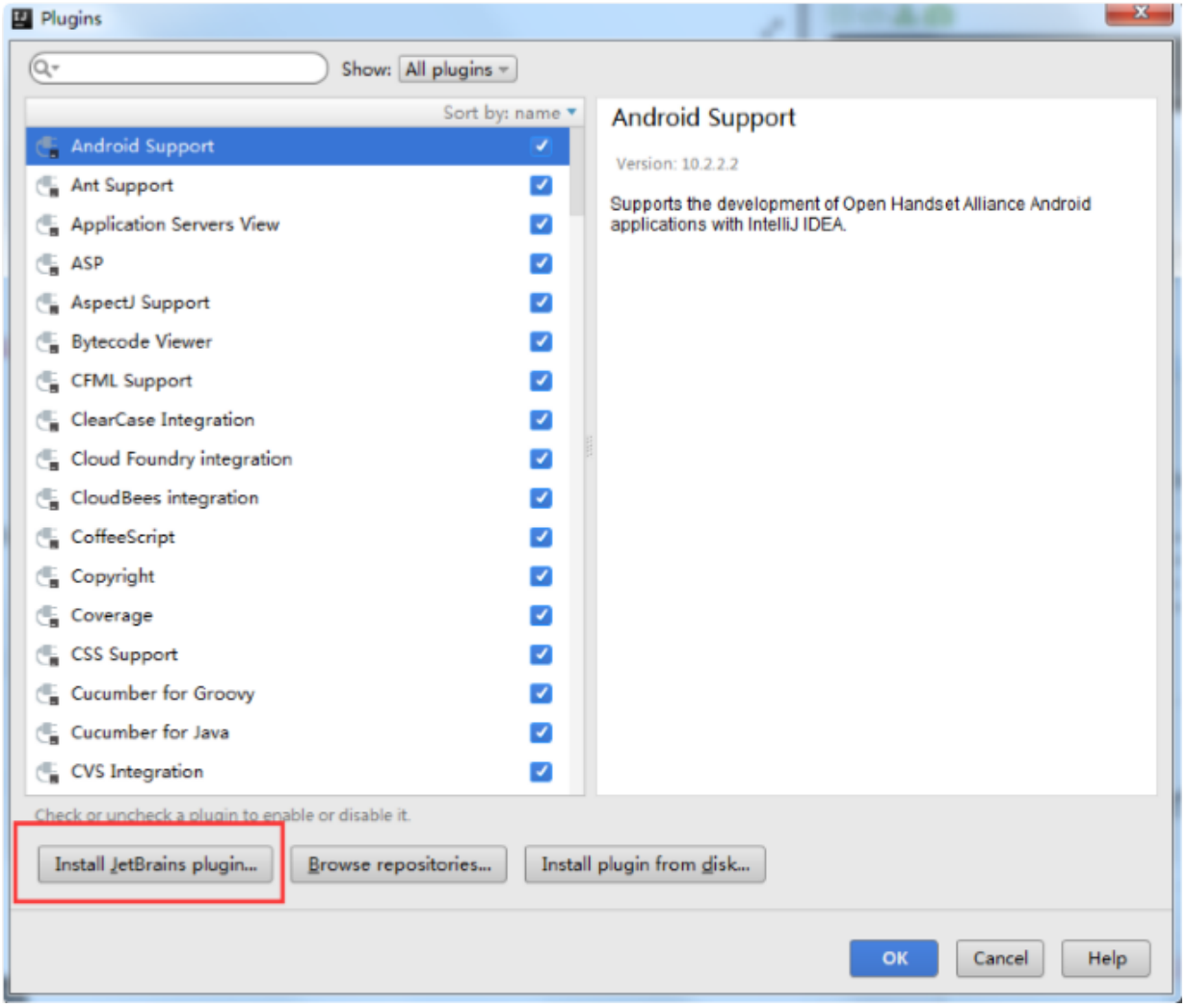
> **注意：**在编译的时候，如果有中文会出现乱码现象，解决方法查看：http://www.scala-lang.org/download/2.11.1.html

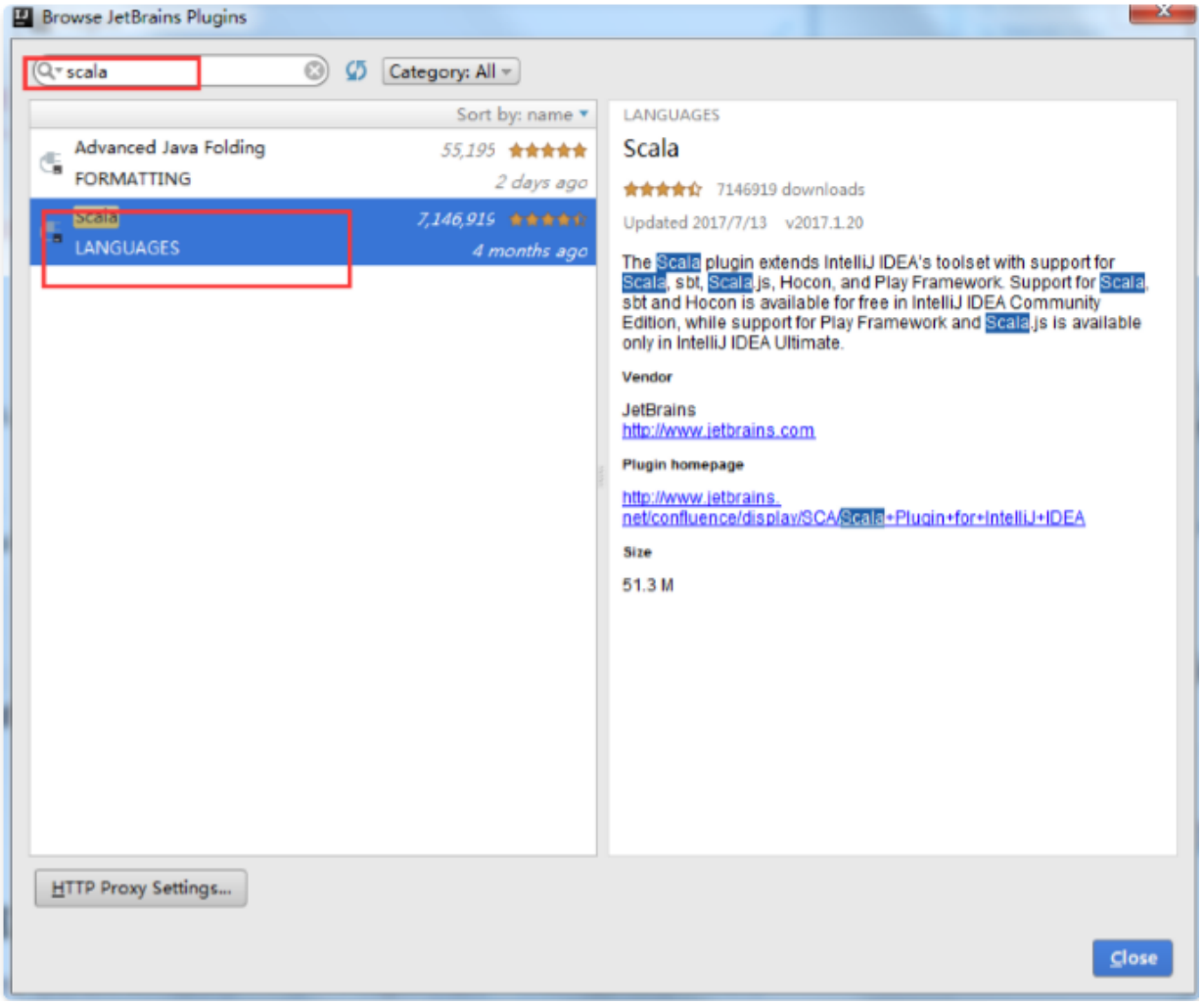
3. **Windows上安装**
   Windows上安装亦是如此，一路next，然后配置SCALA_HOME和Path即可
4. **安装与破解IntelliJ IDEA2017**
   参考链接：http://blog.csdn.net/yangying496875002/article/details/73603303

idea安装scala插件

依次点击[configure/plugins/Install JetBrains plugin]


Welcome to IntelliJ IDEA
scalahwbd14
D:\Develop\Bigdata...roject\scalahwbd14
untitled
D:\IDEA\project\untitled
IntelliJ IDEA
Version 2017.1
Create New Project
Import Project
Open
Check out from Version Control
Configure
Get Help

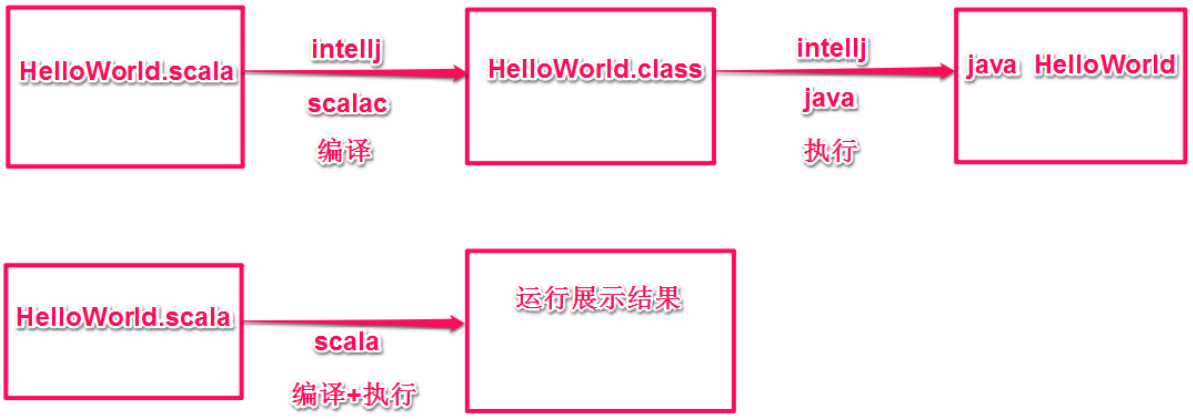

Plugins
Show: All plugins
Sort by: name
Android Support
Ant Support
Application Servers View
ASP
AspectJ Support
Bytecode Viewer
CFML Support
ClearCase Integration
Cloud Foundry integration
CloudBees integration
CoffeeScript
Copyright
Coverage
CSS Support
Cucumber for Groovy
Cucumber for Java
CVS Integration
Android Support
Version: 10.2.2.2
Supports the development of Open Handset Alliance Android applications with IntelliJ IDEA.
Check or uncheck a plugin to enable or disable it.
Install JetBrains plugin...
Browse repositories...
Install plugin from disk...
OK
Cancel
Help

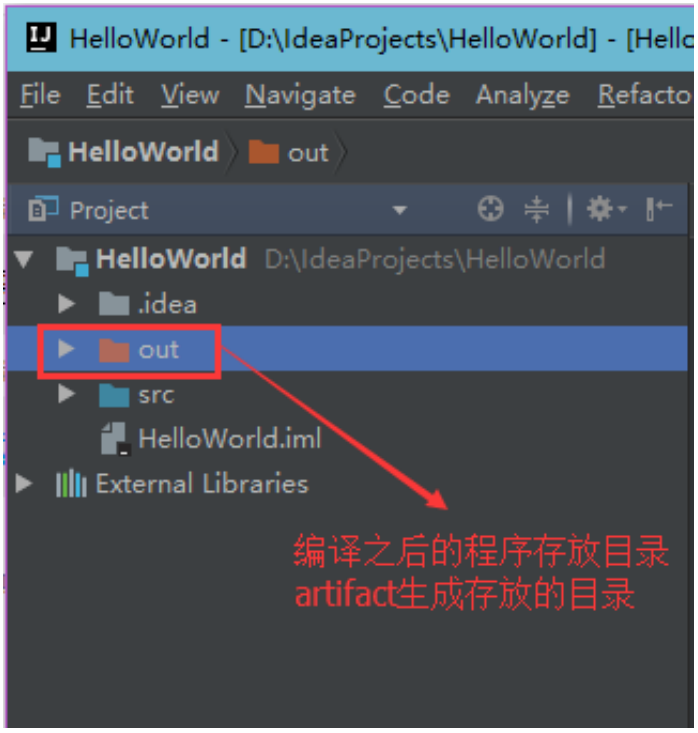


idea常用快捷键

- Ctrl + E，可以显示最近编辑的文件列表
- Shift + Click可以关闭文件
- Ctrl + [或]可以跳到大括号的开头结尾
- Ctrl + Shift + Backspace可以跳转到上次编辑的地方
- Ctrl + F12，可以显示当前文件的结构
- Ctrl + F7可以查询当前元素在当前文件中的引用，然后按F3可以选择
- Ctrl + N，可以快速打开类
- Ctrl + Shift + N，可以快速打开文件
- Alt + Q可以看到当前方法的声明
- Ctrl + W可以选择单词继而语句继而行继而函数
- Alt + F1可以将正在编辑的元素在各个面板中定位
- Ctrl + P，可以显示参数信息
- Ctrl + Shift + Insert可以选择剪贴板内容并插入
- Alt + Insert可以生成构造器/Getter/Setter等
- Ctrl + Alt + V 可以引入变量。例如把括号内的SQL赋成一个变量
- Ctrl + Alt + T可以把代码包在一块内，例如try/catch
- Alt + Up and Alt + Down可在方法间快速移动

Scala代码运行





强类型语言 & 弱类型语言

强类型语言

- 定义对象或变量时，需要指定其归属类型
- 一旦一个变量类型确定，它所归属类型不可改变

弱类型语言

- 定义变量时，不需要指定其归属类型
- 在程序运行中，可以改变变量的归属类型

Scala基础操作

声明变量的两种修饰符

var：变量可被重新赋值
val：（常量）不可被重新赋值
在编程过程中，能使用val的地方不要使用var

Scala基本类型

Scala中的基本数据类型如下图：

Value type	Range
Byte	8-bit signed two's complement integer (-2^7 to $2^7 - 1$, inclusive)
Short	16-bit signed two's complement integer (-2^{15} to $2^{15} - 1$, inclusive)

从上表中可以看出，Scala的基本数据类型与Java中的基本数据类型是一一对应的，不同的是Scala的基本数据类型头字母必须大写

Scala与Java基本类型的区别

java基础数据类型，它对应的变量，不是对象，不能用过”.”运算符来访问对象的方法
scala对应java的基础数据类型的类型，对应的变量，是对象，可以通过“.”运算符调用对象的属性或方法

整数类型变量定义

```
//16进制定义法
val x = 0x29
x: Int = 41

//十进制定义法
val x = 41
x: Int = 41

//八进制定义法
val x = 051
res0: Int = 41
```

浮点类型变量定义

```
//Double类型定义,直接输入浮点数，编译器会将其自动推断为Double类型
val doubleNumber = 3.141529
doubleNumber: Double = 3.141529

//要定义Float类型浮点数，需要在浮点数后面加F或f
val floatNumber = 3.141529F
floatNumber: Float = 3.141529

val floatNumber = 3.141529f
floatNumber: Float = 3.141529

//浮点数指数表示法，e也可以是大写E,0.314529e1与0.314529*10等同
val floatNumber = 0.314529e1
floatNumber: Double = 3.14529
```

字符变量定义

```
//字符定义，用''将字符包裹
var charLiteral = 'A'
charLiteral: Char = A
```

常用特殊字符包括

```
\n 换行符，其Unicode编码为 (\u000A)
\b 回退符，其Unicode编码为 (\u0008)
\t tab制表符 ，其Unicode编码 (\u0009)
\” 双引号，其Unicode编码为 (\u0022)
\’ 单引号，其Unicode编码为 (\u0027)
\ 反斜杆，其Unicode编码为(\u005C)
```

字符串变量定义

```
//字符串变量用""包裹
val helloWorld = "Hello World"
helloWorld: String = Hello World

//要定义"Hello World"，可以加入转义符\
val helloWorldDoubleQuote = "\"Hello World\""
helloWorldDoubleQuote: String = "Hello World"

//如果希望能够原样输出字符串中的内容，则用三个引号"""将字符串包裹起来，如
println(""" hello cruel world, \n \\\\ \b \\, I am " experienced" p
rogrammer""")
输出结果：hello cruel world, \n \\\\ \b \\, I am " experienced" progr
ammer

//字符串模板嵌套
println(s"name:$name,age:$age")
println(s"name:$name,age:${age}aa")
println(s"""ame:$name
            age:${age}
            over""")
```

布尔类型定义

```
var x = true
x: Boolean = true
```

基础数据类型之间的转换方法

对象.to类型

```
123.toDouble
"123".toInt
123.456.toString
```

> scala变量定义：`var str = "abbcd"`
> 这样的写法不是没有指定str的类型，而是没有显式的指定str的类型
> **显式写法：** `var str:String = "abbcd"`

Scala运算符操作

在Scala中一切操作皆方法，这意味着Scala中的一切皆为对象

算术操作

```
//整数求和，编译器会将其转换为(1).+(2)执行
var sumVlaue = 1 + 2
sumVlaue: Int = 3

//前一语句等同于下列语句
//前者是后者的简写
//当一个对象通过点调用其方法的时候，如果该方法只有一个参数，那么点号可以省略，小括号可以省略，对象、方法、参数之间用空格隔开即可
var sumVlaue = (1).+(2)
sumVlaue: Int = 3

//操作符重载,编译器会将其转换为(1).+(2L)执行
val longSum = 1 + 2L
longSum: Long = 3

//减法、除法、取模、乘法和加法一样

//scala中可以用+ -符号来表示正负数，例如-3 +3，并且可以加入到运算符当中
var y = 1 + -3
y: Int = -2
```

> **scala里面没有++和- -**

关系运算

```
//>运算符
3 > -3
Boolean = true

//<、>=、<=也是一样
```

逻辑运算

```
//逻辑与：&&
val bool = truebool: Boolean = true
bool && bool
Boolean = true//

//逻辑或：||
bool || bool
Boolean = true

//非：！
!(3<= -3)
Boolean = true
```

位运算

```
// 00000001// 00000010// 00000000
1 & 2
Int = 0

// 00000001// 00000010// 00000011
1 | 2
Int = 3

// 00000001// 00000011// 00000010
1 ^ 3
Int =2

//左移位（shift left）//00000110//00001100
6 << 1
Int = 12

//右移位（shift left）//00000110//00000011
6 >> 1
Int = 3

//无符号右移（shift  left）
//11111111111111111111111111111111
//00000000000000000000000000000001
-1 >>> 31
Int = 1
```

对象比较

```
1 == 1
Boolean = true

1 == 1.0
Boolean = true

val x = "Hello"
x: String = Hello

val y = "Hello"
y: String = Hello

//Scala中的对象比较不同于Java中的对象比较
//Scala基于内容比较，而java中比较的是引用，进行内容比较时须定义比较方法
x == y
Boolean = true
```

==方法在scala中等同于equal方法

运算符的优先级

运算符优先级如下图所示，* / %优先级最高，依次类推

标识符 & 注释

标识符：符合java的规范

- 类标识符：驼峰式命名，首字母大写
- 变量：方法标识符，驼峰命名法，首字母小写
- 包标识符：全小写，层级使用点分割

> val在scala中虽然定义的是常量，但是一般都使用变量的规则来命名标识符

scala注释规则和java一致

语句块

- java中的语句块全部都是过程，没有返回值，只有方法语句块中用return才能有返回值
- scala中大部分的语句块都是有返回值的，而且不需要return
- java中语句块的作用主要用来划分作用域
- scala中的语句块除了划分作用域之外还可以带返回值

```
val str1 = "abc"
val str2 = {
    val str3 = s"${str1}def"
    str3
}
println(str3) //访问不到
println(str2) //abcdef
```

> scala中语句块的最后一句，就是该语句块的返回值，如果最后一句是println，则会返回一个空值

Scala程序控制结构

if...else...

- scala中的if...else...语法是有返回值的
- 另外scala中没有三目运算符表达式
- 因此可以再变量赋值上就使用if...else...语法

```
val result = if (score > 60) {
  "及格"
} else {
  "不及格"
}
println(result)
```

while循环

while语句块中是没有返回值的

for循环

for也是scala中少数没有返回值的语句块之一
但是scala中也是提供了一种方式（**yield**）**让其具有返回值能力**
scala的for更像foreach

scala中没有break，也没有continue

1. 使用scala中提供的特殊类型Breaks来实现break

```
val loop = new Breaks;
loop.breakable(
  for (i <- 1 to 5){
    if(i == 3){
      loop.break()
    }else{
      println(i)
    }
  }
)
```

2. 通过return终止整个函数的方式也可以终止循环

基本语法结构

```
for(i <- list){}
```

通过守卫来限定判断条件

```
for (i <- 1 to times if i % 2 == 0 && i > 5) {
  println(i)
}
```

处理复杂的for循环

对于条件复杂的for循环，可以把小括号换成大括号

```
for {
  i <- 1 to times
  j <- 1 to times
  x = i * j
  if x > 25
} println(s"长:$i,宽:$j,满级:${x}")
```

Scala写九九乘法表

```
for (i <- 1 to 9) {
  for (j <- 1 to i) {
    print(s"$j*$i=${i * j}\t")
  }
  println()
}

for (i <- 1 to 9; j <- 1 to i) {
  print(s"$j*$i=${i * j}\t")
  if (i == j) println()
}

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

Unit类型

java里无返回值的方法类型是void

scala中没有void，它是使用Unit类型来代替的

Unit的实例就是”（ ）”

```
//yield后面的语句块一定有返回值，即使表达式上没有返回值，它会以Unit的对象"{}"
var result1 = for (i <- 1 to 10) yield {
  if (i % 2 == 0) i
}
println(result1)
//输出结果：Vector((), 2, (), 4, (), 6, (), 8, (), 10)

var result2 = for (i <- 1 to 10 if i % 2 == 0) yield {
  i
}
println(result2)
//输出结果：Vector(2, 4, 6, 8, 10)
```