

全部课程 (/courses/) / Scala开发教程 (/courses/490) / 基本数据类型及其操作

在线实验，请到PC端体验

# 基本数据类型及其操作

## 一、实验介绍

### 1.1 实验内容

在本节实验中，将会讲解 Scala 支持的基本数据类型。

### 1.2 实验知识点

- 基本数据类型简介
- 操作基本数据类型
- 常用操作符
- 基本数据类型的实现方法

### 1.3 实验环境

- Scala 2.11.7
- Xfce 终端

### 1.4 适合人群

本课程难度为一般，属于初级级别课程，适合零基础或具有 Java 编程基础的用户。

## 二、开发准备

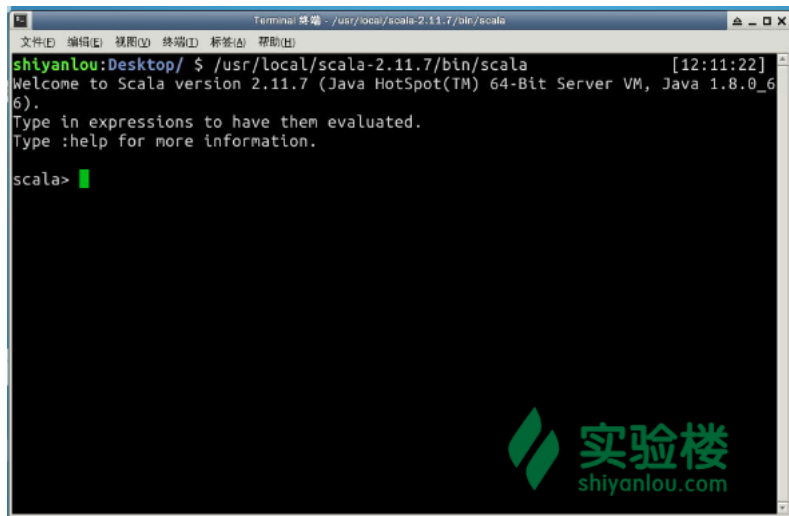
为了使用交互式 Scala 解释器，你可以在打开的终端中输入命令：

```
cd /usr/local/scala-2.11.7/bin/  
  
scala
```

当出现 scala> 开始的命令行提示符时，就说明你已经成功进入解释器了。如下图所示。

动手实践是学习 IT 技术最有效的方式！

开始实验



```
Terminal 终端 - /usr/local/scala-2.11.7/bin/scala
文件(F) 编辑(E) 视图(V) 终端(T) 标签(T) 帮助(H)
shityanlou:Desktop/ $ /usr/local/scala-2.11.7/bin/scala [12:11:22]
Welcome to Scala version 2.11.7 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_66).
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

## 三、实验步骤

### 3.1 基本数据类型简介

如果你是个 Java 程序员，你会发现 Java 支持的基本数据类型，Scala 都有对应的支持，不过 Scala 的数据类型都是对象(比如整数)，这些基本类型都可以通过隐式自动转换的形式支持比 Java 基本数据类型更多的方法。

隐式自动转换的概念将在后面介绍，简单的说就是可以为基本类型提供扩展，比如如果调用 `(-1).abs()`，Scala 发现基本类型 `Int` 没有提供 `abs()` 方法，但可以发现系统提供了从 `Int` 类型转换为 `RichInt` 的隐式自动转换，而 `RichInt` 具有 `abs` 方法，那么 Scala 就自动将 `1` 转换为 `RichInt` 类型，然后调用 `RichInt` 的 `abs` 方法。

Scala 的基本数据类型有：`Byte`、`Short`、`Int`、`Long` 和 `Char`（这些成为整数类型）。整数类型加上 `Float` 和 `Double` 成为数值类型。此外还有 `String` 类型，除 `String` 类型在 `java.lang` 包中定义，其它的类型都定义在包 `scala` 中。比如 `Int` 的全名为 `scala.Int`。实际上 Scala 运行环境会自动会载入包 `scala` 和 `java.lang` 中定义的数据类型，你可以使用直接使用 `Int`、`Short`、`String` 而无需再引入包或是使用全称。

下面的例子给出了这些基本数据类型的字面量用法，由于 Scala 支持数据类型推断，你在定义变量时多数可以不指明数据类型，而是由 Scala 运行环境自动给出变量的数据类型：

```
Welcome to Scala version 2.11.0-M5 (OpenJDK 64-Bit Server VM, Java 1.7.0_25).
Type in expressions to have them evaluated.
Type :help for more information.

scala> var hex=0x5
hex: Int = 5

scala> var hex2=0x00ff
hex2: Int = 255

scala> val prog=0xcafebabe1
prog: Long = 3405691582

scala> val littler:Byte= 38
littler: Byte = 38

scala> val big=1.23232
big: Double = 1.23232

scala> val a='A'
a: Char = A

scala> val f='\u0041'
f: Char = A

scala> val hello="hello"
hello: String = hello

scala> val longString=""" Welcome to Ultamix 3000. Type "Help" for help.""
longString: String = " Welcome to Ultamix 3000. Type "Help" for help."

scala> val bool=true
bool: Boolean = true

scala>
```

Scala 的基本数据类型的字面量也支持方法（这点和 Java 不同，Scala 中所有的数值字面量也是对象），比如：

```
scala> (-2.7).abs
res3: Double = 2.7

scala> -2.7 abs
warning: there were 1 feature warning(s); re-run with -feature for details
res4: Double = 2.7

scala> 0 max 5
res5: Int = 5

scala> 4 to 6
res6: scala.collection.immutable.Range.Inclusive = Range(4, 5, 6)
```

这些方法其实是对于数据类型的 Rich 类型的方法，Rich 类型将在后面再做详细介绍。

## 3.2 操作基本数据类型

Scala 提供了丰富的运算符用来操作前面介绍的基本数据类型。前面说过，这些运算符（操作符）实际为普通类方法的简化（或者称为美化）表示。比如 `1+2`，实际为 `(1).+(2)`，也就是调用 `Int` 类型的 `+` 方法。

例如：

```
scala> val sumMore = (1).+(2)
sumMore: Int = 3
```

实际上类 `Int` 定义了多个 `+` 方法的重载方法（以支持不同的数据类型）比如和 `Long` 类型相加。

`+` 符号是一个运算符，并且是一个中缀运算符。在 Scala 中你可以定义任何方法为一操作符。比如 `String` 的 `indexOf` 方法也可以使用操作符的语法来书写。例如：

**动手实践是学习 IT 技术最有效的方式！**

**开始实验**

```
scala> val s = "Hello, World"
s: String = Hello, World

scala> s indexOf 'o'
res0: Int = 4
```

由此可以看出，运算符在 Scala 中并不是什么特殊的语法，任何 Scala 方法都可以作为操作符来使用。是否是操作符取决于你如何使用这个方法，当你使用 `s.indexOf('o')` 时，`indexOf` 不是一个运算符。而你写成 `s indexOf 'o'`，`indexOf` 就是一个操作符，因为你使用了操作符的语法。

除了类似 `+` 的中缀运算符（操作符在两个操作符之间），还可以有前缀运算符和后缀运算符。顾名思义前缀运算符的操作符在操作数前面，比如 `-7` 前面的 `-`。后缀运算符的运算符在操作数的后面，比如 `7 toLong` 中的 `toLong`。前缀和后缀操作符都使用一个操作数，而中缀运算符使用前后两个操作数。Scala 在实现前缀和后缀操作符的方法，其方法名都以 `unary_` 开头。

比如，表达式 `-2.0` 实际上调用 `(2.0).unary_-` 方法。

```
scala> -2.0
res1: Double = -2.0

scala> (2.0).unary_-
res2: Double = -2.0
```

如果你需要定义前缀方法，你只能使用 `+`、`-`、`!` 和 `~` 四个符号作为前缀操作符。

后缀操作符在不使用 `.` 和括号调用时不带任何参数。在 Scala 中，你可以省略掉没有参数的方法调用的空括号。按照惯例，如果你调用方法是为了利用方法的“副作用”，此时写上空括号，如果方法没有任何副作用（没有修改其它程序状态），你可以省略掉括号。

比如：

```
scala> val s = "Hello, World"
s: String = Hello, World

scala> s toLowerCase
res0: String = hello, world
```

具体 Scala 的基本数据类型支持的操作符，可以参考 Scala API 文档。下面以示例介绍一些常用的操作符：

### 3.2.1 算术运算符 `+`、`-`、`*` 和 `/`

```
scala> 1.2 + 2.3
res0: Double = 3.5

scala> 'b' - 'a'
res1: Int = 1

scala> 11 % 4
res2: Int = 3

scala> 11.0f / 4.0f
res3: Float = 2.75

scala> 2L * 3L
res4: Long = 6
```

### 3.2.2 关系和逻辑运算符（包括 `>`、`<`、`>=` 和 `!` 等）

```
scala> 1 > 2
res5: Boolean = false

scala> 1.0 <= 1.0
res6: Boolean = true

scala> val thisIsBoring = !true
thisIsBoring: Boolean = false

scala> !thisIsBoring
res7: Boolean = true

scala> val toBe=true
toBe: Boolean = true

scala> val question = toBe || ! toBe
question: Boolean = true
```

要注意的是，逻辑运算支持“短路运算”，比如 `op1 || op2`，当 `op1=true`，`op2` 无需再计算，就可以知道结果为 `true`。这时 `op2` 表示式不会执行。例如：

```
scala> def salt()= { println("salt");false}
salt: ()Boolean

scala> def pepper()={println("pepper");true}
pepper: ()Boolean

scala> pepper() && salt()
pepper
salt
res0: Boolean = false

scala> salt() && pepper()
salt
res1: Boolean = false
```

### 3.2.3 位操作符

```
scala> 1 & 2
res2: Int = 0

scala> 1 | 2
res3: Int = 3

scala> 1 ^ 2
res4: Int = 3

scala> ~1
res5: Int = -2
```

### 3.2.4 对象恒等比较

如果需要比较两个对象是否相等，可以使用 `==` 和 `!=` 操作符。

```
scala> 1 == 2
res6: Boolean = false

scala> 1 != 2
res7: Boolean = true

scala> List(1,2,3) == List (1,2,3)
res8: Boolean = true

scala> ("he"+"llo") == "hello"
res9: Boolean = true
```

Scala 的 `==` 和 Java 不同，scala 的 `==` 只用于比较两个对象的值是否相同，而对于引用类型的比较使用另外的操作符 `eq` 和 `ne`。

Scala 的 `==` 和 Java 不同，Scala 的 `==` 只用于比较两个对象的值是否相同，而對於引用類型的比較使用另外的操作符 `eq` 和 `ne`

### 3.2.5 操作符的优先级和左右结合性

Scala 的操作符的优先级和 Java 基本相同，如果有困惑时，可以使用 `()` 改变操作符的优先级。操作符一般为左结合，Scala 规定了操作符的结合性由操作符的最后一个字符定义。对于以 `:` 结尾的操作符都是右结合，其它的操作符多是左结合。

例如：`a*b` 为 `a.*(b)`，而 `a::b` 为 `b.:::(a)`，而 `a::b::c` 等价于 `a:: (b :: c)`，`a*b*c` 等价于 `(a*b)*c`。

## 3.3 【进阶】基本数据类型的实现方法

本小节内容可能需要部分后面章节的知识作为基础。若理解有困难，可以尝试学习后续课程后再返回查看。

Scala 的基本数据类型是如何实现的呢？实际上，Scala 以与 Java 同样的方式存储整数：把它当作 32 位的字类型。这对于有效使用 JVM 平台和与 Java 库的互操作性方面来说都很重要。

标准的操作，如加法或乘法，都被实现为数据类型基本运算操作。然而，当整数需要被当作（Java）对象看待的时候，Scala 使用了“备份”类 `java.lang.Integer`。如在整数上调用 `toString` 方法或者把整数赋值给 `Any` 类型的变量时，就会这么做。当需要的时候，`Int` 类型的整数能自动转换为 `java.lang.Integer` 类型的“装箱整数（boxed integer）”。

这些听上去和 Java 的 `box` 操作很像，实际上它们也很像，但这里有一个重要的差异，Scala 使用 `box` 操作比在 Java 中要少的多：

```
// Java代码
boolean isEqual(int x,int y) {
    return x == y;
}
System.out.println(isEqual(421,421));
```

你当然会得到 `true`。现在，把 `isEqual` 的参数类型变为 `java.lang.Integer`（或 `Object`，结果都一样）：

```
// Java代码
boolean isEqual(Integer x, Integer y) {
    return x == y;
}
System.out.println(isEqual(421,421));
```

你会发现你得到了 `false` 的原因是，数 421 使用“`box`”操作了两次，因此参数 `x` 和 `y` 是两个不同的对象。因为在引用类型上，`==` 表示引用相等，而 `Integer` 是引用类型，所以结果是 `false`。这是展示了 Java 不是纯面向对象语言的一个方面。我们能清楚观察到基本数据值类型和引用类型之间的差别。

现在在 Scala 里尝试同样的实验：

```
scala> def isEqual(x:Int,y:Int) = x == y
isEqual: (x: Int, y: Int)Boolean

scala> isEqual(421,421)
res0: Boolean = true

scala> def isEqual(x:Any,y:Any) = x == y
isEqual: (x: Any, y: Any)Boolean

scala> isEqual(421,421)
res1: Boolean = true
```

Scala 的 `==` 设计出自动适应变量类型的操作，对值类型来说，就是自然的（数学或布尔）相等。对于引用类型，`==` 被视为继承自 `Object` 的 `equals` 方法的别名。比如对于字符串比较：

```
scala> val x = "abcd".substring(2)
x: String = cd

scala> val y = "abcd".substring(2)
y: String = cd

scala> x==y
res0: Boolean = true
```

Java

而在 Java 里，`x` 与 `y` 的比较结果将是 `false`，程序员在这种情况下应该用 `equals`，不过它容易被忘记。

动手实践是学习 IT 技术最有效的方式！

开始实验

然而，有些情况下，你可能需要使用引用相等代替用户定义的相等。例如，某些时候效率是首要因素，你想要把某些类进行哈希合并（`hash cons`），然后通过引用相等比较它们的实例。为了满足这种情况，类 `AnyRef` 定义了附加的 `eq` 方法，它不能被重载并且实现为引用相等（也就是说，它表现得就像Java里对于引用类型的 `==` 那样）。同样也有一个 `eq` 的反义词，被称为 `ne`。例如：

```
scala> val x = new String("abc")
x: String = abc

scala> val y = new String("abc")
y: String = abc

scala> x == y
res0: Boolean = true

scala> x eq y
res1: Boolean = false

scala> x ne y
res2: Boolean = true
```

四、实验总结

在本节实验中，我们了解了什么是基本数据类型，并学习了如何操作基本数据类型。同时，我们还学习如何使用一些常用的操作符。最后一小节我们还提到了基本数据类型的实现方法，这部分内容可能对于初学者稍显难度，建议此部分同学在学习完整个 Scala 课程后再来回顾它们。

◀ 上一节 (/courses/490/labs/1685/document)

下一节 ▶ (/courses/490/labs/1687/document)

课程教师



**引路蜂**  
共发布过6门课程

CSDN 专家博主，擅长Java ME, Blackberry ,LWUIT , iPhone, Android, Windows Mobile, Mono , Windows Phone 7等平台开发，主页 <http://www.imobilebbs.com/>

[查看老师的所有课程 > \(/teacher/164063\)](/teacher/164063)

进阶课程

- Scala 专题教程 - Case Class和模式匹配 (/courses/514)
- Scala 专题教程 - 隐式变换和隐式参数 (/courses/515)
- Scala 专题教程 - 抽象成员 (/courses/516)
- Scala 专题教程 - Extractor (/courses/526)



动手做实验，轻松学IT



公司 (http://weibo.com/shiyanlou2013)

- 关于我们 (/aboutus)
- 联系我们 (/contact)
- 加入我们 (http://www.simplecloud.cn/jobs.html)
- 技术博客 (https://blog.shiyanlou.com)

服务

- 企业版 (/saas)
- 实战训练营 (/bootcamp/)
- 会员服务 (/vip)

合作

- 我要投稿 (/contribute)
- 教师合作 (/labs)
- 高校合作 (/edu/)
- 友情链接 (/friends)
- 开发者 (/developer)
- 学习路径
  - Python学习路径 (/paths/python)
  - Linux学习路径 (/paths/linux)
  - 大数据学习路径 (/paths/bigdata)

动手实践是学习 IT 技术最有效的方式！开始实验