Scala: Переменные, типы данных, функции

Егоркин Станислав, Зайдиев Артур

31.10.2025

Типы данных: Примитивные

Примитивные типы данных в Scala включают:

```scala
object PrimitiveTypesExample extends App {
  val i: Int     = 42          // 32-bit
  val l: Long    = 123456789L  // 64-bit L
  val d: Double  = 3.14        // 64-float point
  val f: Float   = 2.5f        // 32-bit
  val b: Byte    = 127         // 8-bit
  val s: Short   = 32000       // 16-bit
  val c: Char    = 'A'         // 16-bit Unicode
  val bool: Boolean = true     // true / false
  val u: Unit    = ()          // void
```

Типы данных: Пользовательские

Пользовательские типы создаются с помощью классов, case-классов, trait'ов и т.д.

```
1  object CustomTypesExample extends App {
2    case class Point(x: Double, y: Double)
3
4    sealed trait TrafficLight
5    case object Red extends TrafficLight
6    case object Yellow extends TrafficLight
7    case object Green extends TrafficLight
8
9    enum Color { case Red, Green, Blue }
10
11   val (p, light, color) = (Point(1, 2), Green, Color.
       Blue)
12    println(s"Point: $p, Light: $light, Color: $color")
       //Point: Point(1.0,2.0), Light: Green, Color: Blue
13 }
```

Операции над типами

Операции зависят от типа:

- Арифметические: +, -, *, / для чисел
- Логические: &&, ||, ! для Boolean
- Строковые: + для конкатенации

```scala
object TypeOperationsExample extends App {
  val (a, b, name, x, y) = (10, 3, "Scala", true, false)

  println(s"$a + $b = ${a + b}") //10 + 3 = 13
  println(s"$a * $b = ${a * b}") // 10 * 3 = 30
  println(s"$a / $b = ${a / b}") // 10 / 3 =3
  println(s"$a.toDouble / $b = ${a.toDouble / b}") // 10.toDouble / 3 =3.33335

  println(s"Hello, $name!") // Hello, Scala!
  println(name + " " + 2025) // Scala 2025

  println(s"$x && $y = ${x && y}") // false
  println(s"$x || $y = ${x || y}") //true
  println(s"!$x = ${!x}") // !true = false
}
```

Преобразование типов

Scala поддерживает явное преобразование с помощью методов like toInt, toString. Также implicit conversions.

Пример явного преобразования:

```
1  object TypeConversionExample extends App {
2    val s: String = "123"
3    val d: Double = 45.67
4
5    // String to num
6    val i: Int = s.toInt
7    val dFromStr: Double = s.toDouble
8
9    // Num to String
10   val strFromInt: String = i.toString
11   val strFromDouble: String = d.toString
12
13   // explitic
14   val intToLong: Long = i.toLong
15   val doubleToInt: Int = d.toInt   // 45
16
17 }
```

Сравнения типов

Сравнения: == (структурное равенство), eq (ссылочное равенство), !=, >, < и т.д.

Пример:

```
1  object TypeComparisonExample extends App {
2    val (x, y, z): (Any, Any, Any) = (42, "42", 42)
3
4    println(s"x == z: ${x == z}")   // true
5    println(s"x == y: ${x == y}")   // false
6
7    def describe(v: Any) = v match {
8      case i: Int => s"Int: $i"
9      case s: String => s"String: $s"
10     case _ => "Unknown"
11   }
12
13   println(describe(x)) //Int: 42
14   println(describe(y)) //String: 42
15
16   if (x.isInstanceOf[Int]) println(s"Casted: ${x.
       asInstanceOf[Int]}")
17 }
```

Объявление переменных

- val: неизменяемая (immutable) - var: изменяемая (mutable)

```
1  // val (иммутабельные) и var (мутабельные)
2  object VariableDeclarationExample extends App {
3    val immutable: String = "Не изменится"
4    // immutable = "Другое" // ОШИБКА!
5
6    var mutable: Int = 10
7    mutable += 5
8    println(s"mutable = $mutable")   // 15
9
10   // Ленивые val (вычисляются при первом обращении)
11   lazy val lazyValue: Int = {
12     println("Вычисляю ленивое значение...")
13     100 * 100
14   }
15
16   println("До обращения к lazyValue")
17   println(s"lazyValue = $lazyValue")   // здесь вычисли
       тся
18 }
```

Области видимости

- Локальные: внутри блока - Класса: члены класса - Пакета: package object

```
1  object ScopeExample extends App {
2    val outer = "Я в объекте"
3
4    def localScope(): Unit = {
5      val inner = " Я внутри функции"
6      println(outer)  // видим outer
7      println(inner)
8    }
9
10   {
11     val blockVar = "Я в блоке"
12     println(blockVar)
13   }
14
15   // println(blockVar) // ОШИБКА! вне блока
16
17   localScope()
18   println(outer)
19 }
```

Входные данные: Передача по ссылке и по значению

В Scala параметры передаются по значению (call-by-value) по умолчанию. Для call-by-name использовать =>.

```
1  object PassByExample extends App {
2    case class Counter(var value: Int)
3
4    def byValue(x: Int) = println(s"byValue: x = $x")
5    def byRef(c: Counter) = { c.value += 1; println(s"
       byRef: ${c.value}") }
6
7    val (num, counter) = (10, Counter(5))
8
9    byValue(num)
10   byRef(counter)
11   println(s"num: $num, counter: ${counter.value}") //
       num: 10, counter: 6
12 }
```

Выходные данные

Функции могут возвращать значение. Тип возврата указывается после :.

```scala
object ReturnValuesExample extends App {
  def divide(a: Int, b: Int): Either[String, Double] =
    if (b == 0) Left("Деление на ноль!")
    else Right(a.toDouble / b)
  def getUserName(id: Int): Option[String] =
    id match {
      case 1 => Some("Alice")
      case 2 => Some("Bob")
      case _ => None
    }
  def returnTuple(): (String, Int, Boolean) =
    ("Scala", 3, true)

  val result = divide(10, 2)
  println(result)   // Right(5.0)
  val name = getUserName(1)
  name.foreach(println)   // Alice
  val (lang, version, isCool) = returnTuple()
  println(s"$lang $version is cool: $isCool")

}
```

Рекурсия

Scala поддерживает рекурсию, включая хвостовую рекурсию для оптимизации.

```scala
import scala.annotation.tailrec

object RecursionExample extends App {

  // Факториал без tailrec - переполнит стек
  def factorial(n: Int): Int =
    if (n <= 1) 1 else n * factorial(n - 1)

  // Хвостовая рекурсия
  def factorialTail(n: Int): BigInt = {
    @tailrec
    def loop(acc: BigInt, x: Int): BigInt =
      if (x <= 1) acc
      else loop(acc * x, x - 1)
    loop(1, n)
  }
  println(s"5! = ${factorial(5)}") //120
  println(s"20! = ${factorialTail(20)}")   //
    2432902008176640000
}
```

Замыкания

```
1  object ClosureExample extends App {
2
3    var multiplier = 2
4    val multiplyBy: Int => Int = (x: Int) => x *
       multiplier
5
6    println(s"3 * 2 = ${multiplyBy(3)}")   // 6
7
8    multiplier = 5
9    println(s"3 * 5 = ${multiplyBy(3)}")   // 15 замыкани
       е "видит" изменение!
10
11   // Фабрика функций
12   def makeScaler(factor: Int): Int => Int =
13     (x: Int) => x * factor
14
15   val doubleIt = makeScaler(2)
16   val tripleIt = makeScaler(3)
17
18   println(s"doubleIt(4) = ${doubleIt(4)}")   // 8
19   println(s"tripleIt(4) = ${tripleIt(4)}")   // 12
20 }
```

Литература

https://docs.scala-lang.org/ru/scala3/book/