Scala: Конструкции потока управления

Егоркин Станислав и Зайдиев Артур

13 ноября 2025 г.

План

Конструкции потока управления

- Условные конструкции
- Циклы
- Блоки

Обработка ошибок

- Исключения
- Assert и отладка

Управление сложностью кода

- Пространства имен и модули
- Пакеты и импорт
- Экспорт (в Scala — public API)

Условные конструкции

- if / else — стандартные условия
- match — сопоставление с образцом (аналог switch, но мощнее)

```
val x = 10
val res = if (x > 0) "positive" else "non-
    positive"
println(res)

x match {
  case 0 => println("zero")
  case n if n > 0 => println("positive")
  case _ => println("negative")
}
```

Сопоставление с образцом как выражение

```
val x = 10
val desc = x match {
  case 0 => "zero"
  case n if n > 0 => "positive"
  case _ => "negative"
}
println(desc)
```

Циклы: while, do-while, foreach

```
var i = 0
while (i < 3) {
  println(s"i = $i")
  i += 1
}

do {
  println(s"i = $i")
  i -= 1
} while (i > 0)

(1 to 3).foreach(i => println(s"i = $i"))
```

Циклы: for и итераторы

- for — универсальный цикл с генераторами
- Можно использовать с коллекциями и yield

```
for (i <- 1 to 5) println(i)

val doubled = for (x <- List(1, 2, 3)) yield x *
    2
println(doubled) // List(2,4,6)

// С фильтром
for (x <- 1 to 10 if x % 2 == 0) println(x)
```

For comprehension с несколькими генераторами

```
val pairs = for {
  x <- List(1, 2)
  y <- List("a", "b")
} yield (x, y)

println(pairs)
// List((1,a), (1,b), (2,a), (2,b))
```

Блоки и выражения

- Блоки возвращают последнее выражение
- Локальные объявления видны только внутри блока

```
val result = {
  val a = 2
  val b = 3
  a + b // возвращаемое значение
}
println(result) // 5
```

Исключения

- try / catch / finally — обработка ошибок
- Можно использовать выражением, возвращающим значение

```
def safeDivide(a: Int, b: Int): Int = {
  try {
    a / b
  } catch {
    case _: ArithmeticException => 0
  } finally {
    println("Attempted division")
  }
}

println(safeDivide(10, 2)) // 5
println(safeDivide(10, 0)) // 0
```

Безопасное деление через Option

```
def safeDivide(a: Int, b: Int): Option[Int] =
  if (b == 0) None else Some(a / b)

println(safeDivide(10, 2)) // Some(5)
println(safeDivide(10, 0)) // None
```

Безопасное деление через Either

```
def safeDivideEither(a: Int, b: Int): Either[
    String, Int] =
  if (b == 0) Left("division by zero") else
     Right(a / b)

println(safeDivideEither(10, 2)) // Right(5)
println(safeDivideEither(10, 0)) // Left("
    division by zero")
```

Встроенные средства отладки

- assert(cond, message) — проверка во время выполнения
- require(cond) — для проверки предусловий

```
val x = 5
assert(x > 0, "x должен быть положительным")

// require используется в конструкторах
case class Person(name: String, age: Int) {
  require(age > 0, "Возраст должен быть > 0")
}
```

Различие require и assert

```
case class Person(name: String, age: Int) {
  require(age > 0, "age must be positive") //
      IllegalArgumentException
  assert(name.nonEmpty) // AssertionError
}
```

Пространства имен и модули

- Пространства имен реализованы через `package` и `object`
- `object` может содержать методы и константы (Singleton)

```
package myapp

object MathUtils {
  def square(x: Int): Int = x * x
}

println(MathUtils.square(4))
```

Пакеты и импорт

- Импорт выполняется с помощью ключевого слова import
- Можно импортировать целый пакет или отдельные символы

```
import scala.math._
println(sqrt(16))

// Импорт части пакета
import scala.collection.mutable.{Map,
    ArrayBuffer}
val buf = ArrayBuffer(1,2,3)
```

Экспорт и доступность

- Модификаторы доступа: public (по умолчанию), private, protected
- Экспорт (API) — через явное определение методов/классов в пакете

```
package geometry

class Point(private val x: Int, private val y:
    Int) {
  def show() = println(s"($x, $y)")
}

object Main extends App {
  val p = new Point(3,4)
  p.show()
}
```

Переэкспорт членов с помощью export

```
package myapp

object MathUtils:
  def square(x: Int): Int = x * x
  def cube(x: Int): Int = x * x * x

export MathUtils.{square, cube}

@main def run(): Unit =
  println(square(4))
  println(cube(3))
```

Сопоставление с образцом для sealed trait

```
sealed trait Shape
case class Circle(r: Double) extends Shape
case class Rectangle(w: Double, h: Double)
    extends Shape

def area(s: Shape): Double = s match {
  case Circle(r) => math.Pi * r * r
  case Rectangle(w, h) => w * h
}

println(area(Circle(2)))
println(area(Rectangle(3,4)))
```

Вывод

- Scala поддерживает мощные конструкции управления потоком
- Исключения и assert — инструменты надёжного кода
- Модули, пакеты и объекты помогают структурировать проект

Список литературы

- https://scalabook.ru/
- https://docs.scala-lang.org/

Спасибо за внимание!