Scala: Обобщенные типы, Стандартная библиотека ввода-вывода, Конкурентность / параллелизм

Егоркин Станислав и Зайдиев Артур

27 ноября 2025 г.

План

Обобщенные типы
- Generic классы и методы
- Ограничения типов (bounds)
- Вариантность (variance)

Стандартная библиотека ввода-вывода
- Ввод с консоли
- Чтение файлов
- Запись в файлы

Конкурентность / параллелизм
- Потоки (threads)
- Futures и Promises
- Параллельные коллекции

Обобщенные типы: Generic классы

Generic классы позволяют работать с разными типами

```
class Box[T](val content: T)
val intBox = new Box[Int](42)
println(intBox.content) // Вывод: 42

// Дополнительный пример
val strBox = new Box[String]("Hello")
println(strBox.content) // Вывод: Hello
```

Обобщенные типы: Generic методы

```
 Generic методы для универсальных функций

def printBox[T](box: Box[T]): Unit = {
 println(box.content)
}
printBox(intBox) // Вывод: 42

// Метод с несколькими параметрами
def merge[A, B](a: A, b: B): (A, B) = (a, b)
println(merge(1, "one")) // (1,one)
```

Ограничения типов: Upper bounds

```
 Upper bound: T <: A (T подтип A)

def max[T <: Comparable[T]](a: T, b: T): T = {
 if (a.compareTo(b) > 0) a else b
}
println(max("apple", "banana")) // banana

// Дополнительно: с числами
println(max(5, 10)) // 10
```

Ограничения типов: Lower bounds

```
 Lower bound: T >: A (T супер-тип A)

class Animal
class Dog extends Animal
def addPet[T >: Dog](pet: T): Unit = println(pet)
addPet(new Animal) // OK

// Дополнительно
addPet(new Dog) // OK
```

Вариантность: Covariance

Covariance: +T (подтипы совместимы)

```
class Covariant[+T](val elem: T)
val dogCov: Covariant[Animal] = new Covariant[Dog](new Dog)
println(dogCov.elem) // Dog instance

// Работает, т.к. Dog <: Animal
```

Вариантность: Contravariance

Contravariance: -T (супер-типы совместимы)

```
trait Writer[-T] { def write(value: T): Unit }
val animalWriter: Writer[Dog] = new Writer[Animal] { def write(a:
Animal) = println(a) }
// Работает, т.к. Animal >: Dog
```

Ввод-вывода: Ввод с консоли

Используем scala.io.StdIn

```
import scala.io.StdIn
val name = StdIn.readLine("Enter your name: ")
println(s"Hello, $name!")

// Дополнительно: чтение числа
val age = StdIn.readInt()
println(s"Age: $age")
```

Ввод-вывода: Чтение файлов

Используем scala.io.Source

```
import scala.io.Source
val lines = Source.fromFile("file.txt").getLines().toList
lines.foreach(println)

// Дополнительно: чтение строки
val text = Source.fromFile("file.txt").mkString
println(text)
```

Ввод-вывода: Запись в файлы

Используем java.io.PrintWriter

```
import java.io._
val writer = new PrintWriter(new File("output.txt"))
writer.write("Hello, Scala!")
writer.close()

// Дополнительно: append
val fw = new FileWriter("output.txt", true)
fw.write("\nMore text")
fw.close()
```

Конкурентность: Потоки (threads)

Создание потоков с Thread

```
val t = new Thread(() => println("Hello from thread"))
t.start()
t.join()

// Дополнительно: Runnable
class MyRunnable extends Runnable {
 def run(): Unit = println("Running")
}
new Thread(new MyRunnable).start()
```

Конкурентность: Futures

Асинхронные вычисления с Future

```
import scala.concurrent.Future
import scala.concurrent.ExecutionContext.Implicits.global
val f = Future { Thread.sleep(1000); 42 }
f.foreach(println) // Вывод: 42 (асинхронно)

// Обработка
f.onComplete { case Success(v) => println(v); case Failure(e) =>
println(e) }
```

Конкурентность: Promises

Promise для управления Future

```
import scala.concurrent.Promise
val p = Promise[Int]()
val f = p.future
Future { p.success(42) }
f.foreach(println) // 42

// Failure
p.failure(new Exception("Error"))
```

Параллелизм: Параллельные коллекции

Par для параллельного выполнения

```
val list = (1 to 1000000).toList
val sum = list.par.sum
println(sum)

// Map параллельно
val doubled = list.par.map(_ * 2)
println(doubled.take(5)) // List(2,4,6,8,10)
```

Конкурентность: Синхронизация

Synchronized для thread-safety

```
var counter = 0
def inc(): Unit = this.synchronized { counter += 1 }
(1 to 100).foreach(_ => new Thread(() => inc()).start())

// Без synchronized может быть race condition
```

Параллелизм: Executors

Пул потоков с Executors

```
import java.util.concurrent.Executors
val pool = Executors.newFixedThreadPool(4)
pool.submit(() => println("Task 1"))
pool.submit(() => println("Task 2"))
pool.shutdown()

// Управление параллелизмом
```

Конкурентность: Await

```
 Блокирующее ожидание Future

import scala.concurrent.duration._
import scala.concurrent.Await
val result = Await.result(f, 2.seconds)
println(result) // 42

// С таймаутом
```

Параллелизм: Parallel for

Параллельный for с par

```
val range = (1 to 10).par
range.foreach(i => println(i)) // Параллельный вывод

// Filter параллельно
val evens = range.filter(_ % 2 == 0)
println(evens) // ParVector(2,4,6,8,10)
```

Вывод и литература

- Обобщенные типы обеспечивают типобезопасность
- Библиотека I/O проста и эффективна
- Конкурентность упрощает параллельный код

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

Спасибо за внимание!