

Scala: Специфические конструкции языка, Виртуальная машина, Управление памятью

Егоркин Станислав и Зайдиев Артур

27 ноября 2025 г.

План

Специфические конструкции языка

- Traits
- Mixins
- Case classes
- Pattern matching
- Implicits

Виртуальная машина

- JVM overview
- Compilation to bytecode
- Java interoperability

Управление памятью

- Memory areas (heap, stack)
- Garbage collection
- GC algorithms

Специфические конструкции: Traits

Traits - интерфейсы с реализацией

```
trait Animal {  
    def speak(): String  
}  
class Dog extends Animal {  
    def speak(): String = "Woof"  
}  
val dog = new Dog  
println(dog.speak()) // Woof  
  
// Дополнительно: с дефолтной реализацией  
trait Greeter {  
    def greet() = println("Hello")  
}
```

Специфические конструкции: Mixins

Mixins - множественное наследование через traits

```
trait Flyer { def fly() = println("Flying") }
trait Swimmer { def swim() = println("Swimming") }
class Duck extends Flyer with Swimmer
val duck = new Duck
duck.fly() // Flying
duck.swim() // Swimming

// Дополнительно: порядок mixins
trait A { def msg = "A" }
trait B extends A { override def msg = super.msg + "B" }
class C extends A with B { override def msg = super.msg + "C" }
println(new C().msg) // ABC
```

Специфические конструкции: Case classes

Case classes - для immutable данных, auto equals/hashCode

```
case class Person(name: String, age: Int)
val p1 = Person("Alice", 30)
val p2 = Person("Alice", 30)
println(p1 == p2) // true

// Дополнительно: copy метод
val p3 = p1.copy(age = 31)
println(p3) // Person(Alice,31)
```

Специфические конструкции: Pattern matching

Pattern matching – мощный switch с деструктуризацией

```
val x: Any = "Scala"
x match {
  case i: Int => println("Int")
  case s: String => println("String: " + s)
  case _ => println("Other")
}
// String: Scala

// C case class
p1 match {
  case Person(name, age) => println(s"$name is $age")
}
```

Специфические конструкции: Implicits

Implicits - неявные преобразования и параметры

```
implicit def intToString(i: Int): String = i.toString  
val s: String = 42 // Implicit conversion  
println(s) // "42"
```

```
// Implicit parameters  
def greet(name: String)(implicit greeting: String) =  
  println(s"$greeting, $name")  
implicit val hello = "Hello"  
greet("World") // Hello, World
```

Виртуальная машина: JVM overview

Scala работает на Java Virtual Machine (JVM)

- JVM - виртуальная машина для выполнения bytecode
- Обеспечивает платформонезависимость
- Компоненты: Class Loader, Execution Engine, Garbage Collector

```
// Пример запуска Scala на JVM
scalac MyApp.scala // Компиляция в bytecode
scala MyApp // Запуск на JVM
```

Виртуальная машина: Compilation to bytecode

Scala компилируется в JVM bytecode

```
object Hello {  
    def main(args: Array[String]): Unit = {  
        println("Hello, JVM!")  
    }  
}  
  
// После компиляции: .class файлы  
// JVM интерпретирует/компилирует JIT bytecode
```

Виртуальная машина: Java interoperability

Полная совместимость с Java

```
import java.util.ArrayList
val list = new ArrayList[String]()
list.add("Scala")
println(list.get(0)) // Scala

// Вызов Scala из Java и наоборот
```

Управление памятью: Memory areas

JVM память: Heap и Stack

- Heap: для объектов, общий для потоков
- Stack: для локальных переменных, вызовов методов, per-thread

```
val obj = new Object() // Аллоцируется в heap
def method() { val local = 42 } // local в stack
```

Управление памятью: Garbage collection

Автоматический сбор мусора в JVM

- Освобождает память от недостижимых объектов
- Нет manual delete как в C++

```
// Пример: объект становится garbage
var ref = new Object()
ref = null // Теперь объект для GC
```

Управление памятью: GC algorithms

Алгоритмы GC в JVM

- Serial GC: простой, для малого heap
- Parallel GC: использует несколько потоков
- G1 GC: для больших heap, низкие паузы
- ZGC/Shenandoah: concurrent, очень низкие паузы

// Настройка: -XX:+UseG1GC

Вывод и литература

- Специфические конструкции делают Scala выразительным
- JVM обеспечивает portability и производительность
- Автоматическое управление памятью упрощает разработку

Список литературы:

- <https://scalabook.ru/>
- <https://docs.scala-lang.org/>
- <https://www.oracle.com/java/technologies/jvm.html>

Спасибо за внимание!