Scala School

Лекция 12: Тестирование

BINARY DISTRICT

План лекции

- ScalaTest
- ScalaCheck
- ScalaMeter
- Akka Test Kit

ScalaTest

ScalaTest™
simply productive™

ScalaTest

Основная библиотека для написания тестов на Scala

- http://www.scalatest.org/
- https://github.com/scalatest/scalatest

build.sbt:

```
libraryDependencies += "org.scalatest" %% "scalatest" % "3.0.1" % "test"
```

ScalaTest - возможности

- Юнит-тесты
- Property-based тесты
- Асинхронные тесты
- Моки
- …

```
class PrimeFactorsSpec extends FunSuite with Matchers {
  test("Should find prime factors of prime number" ) {
    PrimeFactors.primeFactors(7) should contain only(7)
  }
}
```

ScalaTest

- Гибкий
- Использует implicit преобразования
- Хороший пример DSL на Scala

```
val res: Seq[Seq[Int]] = RunLengthEncoding.pack(List(1, 1, 2, 3, 3, 4, 5,
5))
res should have size(5)
res(0) should contain theSameElementsInOrderAs( List(1, 1))
res(1) should contain only(2)
```

assert

В ScalaTest есть 3 вида assert (`org.scalatest.Assertions`):

- `assert`
- `assertResult`
- `assertThrows`

assert

Отличается от дефолтного `assert` тем, что вместо `AssertionError` выбрасывает `TestFailedException`, содержащий больше информации о несоответствии

assert

```
val right = 1
val left = 2
assert(left == right)
java.lang.AssertionError: assertion failed
```

assert

-
  ```
  val right = 1
  val left = 2
  assert(left == right)
  java.lang.AssertionError: assertion failed
  ```

-
  ```
  import org.scalatest.Assertions._
  val left = 2
  val right = 1
  assert(left == right)
  org.scalatest.exceptions.TestFailedException: 2 did not equal 1
  ```

assertResult, assertThrows

```
import org.scalatest.Assertions._
val a = 5
val b = 2
assertResult(2) {
  a - b
}

assertThrows[IllegalArgumentException] {
  IntegerRoot.calculateRoot(-1)
}
```

Стили тестов

В **ScalaTest** существуют различные стили тестов, они определяют, как будут описываться сценарии.

Функционально все стили одинаковы

Стили тестов: FunSuite

- FunSuite

```
class IntegerRootSpec extends FunSuite with Matchers {
  test("IntegerRoot throws IllegalArgumentException for negative integer" ) {
    assertThrows[IllegalArgumentException] {
      IntegerRoot.calculateRoot(-1)
    }
  }
}
```

Стили тестов: FlatSpec

- FunSuite
- FlatSpec

```
class SetSpec extends FlatSpec {
  "An empty Set" should "have size 0" in {
    assert(Set.empty.size == 0)
  }
}
```

Стили тестов: FunSpec

- FunSuite
- FlatSpec
- FunSpec

```scala
class SetSpec extends FunSpec {
  describe("A Set") {
    describe("when empty") {
      it("should have size 0") {
        assert(Set.empty.size == 0)
      }
    }
  }
}
```

Стили тестов: PropSpec

- FunSuite
- FlatSpec
- FunSpec
- PropSpec

```
class SetSpec extends PropSpec with TableDrivenPropertyChecks
with Matchers {
  val examples = Table("set", BitSet.empty,
HashSet.empty[Int])

  property("an empty Set should have size 0" ) {
    forAll(examples) { set =>
      set.size should be (0)
    }
  }
```

Стили тестов: FeatureSpec

- `FunSuite`
- `FlatSpec`
- `FunSpec`
- `PropSpec`
- `FeatureSpec`

Стили тестов: FeatureSpec

```
class TVSet {
  private var on: Boolean = false
  def isOn: Boolean = on
  def pressPowerButton() {
    on = !on
  }
}
```

Стили тестов: FeatureSpec

```
class TVSetSpec extends FeatureSpec with GivenWhenThen {
  info("As a TV set owner")
  info("I want to be able to turn the TV on and off" )
  info("So I can watch TV when I want" )
  info("And save energy when I'm not watching TV" )
```

Стили тестов: FeatureSpec

```
feature("TV power button") {
  scenario("User presses power button when TV is off" ) {
    Given("a TV set that is switched off" )
    val tv = new TVSet
    assert(!tv.isOn)
    When("the power button is pressed" )
    tv.pressPowerButton()
    Then("the TV should switch on" )
    assert(tv.isOn)
  }
```

Стили тестов

- `FunSuite`
- `FlatSpec`
- `FunSpec`
- `PropSpec`
- `FeatureSpec`
- `....`

Fixtures

Fixture - набор вспомогательных объектов для теста (файлы, сокеты, …)

```
override def withFixture(test: NoArgTest) = {
  super.withFixture(test) match {
    case failed: Failed =>
      val currDir = new File(".")
      val fileNames = currDir.list()
      info("Dir snapshot: " + fileNames.mkString(", "))
      failed
    case other => other
  }
}
```

BeforeAndAfter / BeforeAndAfterAll

```
class FixturesExample extends FunSuite with Matchers with BeforeAndAfter {
  val builder = new StringBuilder
  before { builder.append("Scala") }
  after { builder.clear() }


  test("1") {
    builder.append("1").toString() should equal("Scala1")
  }
  test("2") {
    builder.append("2").toString() should equal("Scala2")
  }
}
```

Matchers DSL

В трейте `org.scalatest.Matchers` содержится DSL для написания более сложных вариаций `assert`

```
class ExampleSpec extends FlatSpec with Matchers {
 test("Should calculate statistics for correct file" ) {
    val resOpt = FileStatistics.calculateStatistics("good_numbers.txt")
    resOpt.isSuccess should be(true)
    val res = resOpt.get
    res.average should equal(7.7 +- 0.001)
  }
}
```

Проверка равенства значений

```
result should equal (3) // can customize equality
result should === (3)   // can customize equality and enforce type
constraints
result should be (3)    // cannot customize equality, so fastest to compile
result shouldEqual 3    // can customize equality, no parentheses required
result shouldBe 3       // cannot customize equality, so fastest to compile,
no parentheses required
```

Везде требуется наличие `implicit org.scalactic.Equality[L]`

Проверка строк

В `Matchers` есть методы для проверки строк как на вхождение подстроки, так и с помощью Regexp

```
test("string substring") {
  val str = "Scala is cool"
  str should have size 13
  str should startWith ("Scala")
  str should endWith ("cool")
  str should include ("is")
}
```

Проверка строк Regex

```
test("string regex") {
  val str = "Scala is cool"
  str should startWith regex  "Sc.la"
  str should endWith regex  "co*l"
  str should include regex  ".s"
  str should fullyMatch regex  "S[cC]ala i. c.*"
}
```

Проверка строк Regex с группами

```
test("string regex with groups") {
  "abbccx" should startWith regex ("a(b*)(c*)" withGroups ("bb", "cc"))
  "xabbcc" should endWith regex ("a(b*)(c*)" withGroups ("bb", "cc"))
  "xabbccx" should include regex ("a(b*)(c*)" withGroups ("bb", "cc"))
  "abbcc" should fullyMatch regex ("a(b*)(c*)" withGroups ("bb", "cc"))
}
```

Проверка boolean с помощью be

Значение be <smth> с помощью reflection конвертится в вызов метода .is<smth> и проверку

```
test("1") {
  Some(3) should be('defined)
  List(1) shouldBe('traversableAgain)
  Iterator(1, 2) should not be('empty)
  Iterator(1, 2) should not be("empty")
}
```

Кастомный BeMatcher

```
object OddMatchers {
  class OddMatcher extends BeMatcher[Int] {
    def apply(left: Int) =
      MatchResult(
        left % 2 == 1,
        left.toString + " was even",
        left.toString + " was odd"
      )
  }
  val odd = new OddMatcher
  val even = not (odd)
}
```

Кастомный BeMatcher

```
test("even odd") {
  import OddMatchers._

  1 shouldBe odd
  2 shouldBe even
}
```

Be

```
ref1 should be theSameInstanceAs ref2

result1 shouldBe a [Tiger]
result1 should not be an [Orangutan]

result shouldBe a [List[_]] // recommended
result shouldBe a [List[Fruit]] // discouraged

sevenDotOh should be (6.9 +- 0.2)
```

Проверка empty

```
test("Test empty") {
  List.empty shouldBe empty
  None shouldBe empty
  Some(1) should not be empty
  new java.util.HashMap[Int, Int] shouldBe empty
  new { def isEmpty = true} shouldBe empty
}
```

Проверка contains

```
test("containing") {
  List(1, 2, 3) should contain (2)
  Map('a' -> 1, 'b' -> 2, 'c' -> 3) should contain ('b' -> 2)
  Set(1, 2, 3) should contain (2)
  "123" should contain ('2')
  Some(2) should contain (2)
  util.Arrays.asList(1, 2) should contain(1)
}
```

contain

```
List(1, 2, 3, 4, 5) should contain oneOf (5, 7, 9)
List(1, 2, 3, 4, 5) should contain noneOf (7, 8, 9)

List(1, 2, 3) should contain atLeastOneOf (2, 3, 4)
List(1, 2, 3, 4, 5) should contain atMostOneOf (5, 6, 7)

List(1, 2, 3, 4, 5) should contain allOf (2, 3, 5)
List(1, 2, 3, 2, 1) should contain only (1, 2, 3)

List(1, 2, 2, 3, 3) should contain theSameElementsAs Vector(3, 2, 3, 1, 2
```

contain

```
    List(1, 2, 2, 3, 3, 3) should contain inOrderOnly ( 1, 2, 3)
    List(0, 1, 2, 2, 99, 3, 3, 3, 5) should contain inOrder ( 1, 2, 3)

    List(1, 2, 3) should contain theSameElementsInOrderAs
collection.mutable.TreeSet(3, 2, 1)
    List(1, 2, 3) shouldBe sorted
```

Проверка Java-коллекций

```
test("Java collections") {
  new util.HashSet(util.Arrays.asList(1, 2)) should not be empty
  util.Arrays.asList(1, 2, 3, 4) should contain only(1, 2, 3, 4)
  Map(1 -> 2, 3 -> 5).asJava should contain key( 1)
}
```

Проверка исключений

```
the [ArithmeticException] thrownBy 1 / 0 should have message "/ by zero"


the [IndexOutOfBoundsException] thrownBy {
  s.charAt(-1)
} should have message "String index out of range: -1"
```

Mock

ScalaTest интегрирован с mock-библиотеками:

- ScalaMock
- EasyMock (Java)
- JMock (Java)
- Mockito (Java)

ScalaMock

Mock-фреймворк, написанный на Scala

- http://scalamock.org/
- https://github.com/paulbutcher/ScalaMock

build.sbt:

```
libraryDependencies += "org.scalamock" %% "scalamock-scalatest-support" %
"3.5.0" % "test"
```

Mock

```
class MockExample extends FunSuite with MockFactory with Matchers {
  test("mock turtle") {
    val mockedTurtle = mock[Turtle]

    (mockedTurtle.setPosition _).expects( 10.0, 10.0)
    (mockedTurtle.forward _).expects( 5.0)
    (mockedTurtle.getPosition _).expects().returning( 15.0, 10.0)

    drawLine(mockedTurtle, ( 10.0, 10.0), (15.0, 10.0))
  }
}
```

Асинхронное тестирование

Трейты для написания асинхронных тестов

- `AsyncFeatureSpec`
- `AsyncFlatSpec`
- `AsyncFreeSpec`
- `AsyncFunSpec`
- `AsyncFunSuite`
- `AsyncWordSpec`

Наследуют `AsyncFeatureSpec,` предоставляющий `ExecutionContext`

Асинхронное тестирование

```
class AsyncSpecExample extends AsyncFunSuite {
  def sum(num: Seq[Int]): Future[Int] = Future(num.sum)

  test("Sum should work") {
    val res = sum(Seq(1, 2, 3, 4))
    res.map { s => assert(s == 10) // Future[Assertion]
    }
  }
}
```

ignore

С помощью `ignore` можно выключить отдельный сценарий

```
class FileStatisticsSpec extends FunSuite with Matchers {
  ignore("Should return None for string which can not be parsed" ) {
    val res = FileStatistics.parseIntOpt("xxx")
    res should be('empty)
  }
}
```

Теги

Тестам можно присваивать теги. Теги наследуют класс

`org.scalatest.Tag`

```scala
object Slow extends Tag("wtf.scala.Slow")
object NonStable extends Tag("wtf.scala.NonStable")

class TagExample extends FunSuite with Matchers {
  test("Run slow test", Slow) {
    assert(2 + 8 == 10)
  }
}
```

Теги

При запуске можно указать, тесты с каким тегом запускать / игнорировать

-n <*tag name*> - теги для запуска

```
-n UnitTests -n FastTests
```

-l <*tag name*> - игнорируемые теги

```
-l SlowTests -l PerfTests
```

Запуск тестов

Из SBT:

```
> test
```

```
> test-only -- [arguments]
```

Из консоли:

```
> scala [-cp scalatest-<version>.jar:...]
org.scalatest.tools.Runner [arguments]
```

Запуск тестов

-D*key*=*value* - ключ / значение конфигураций

```
-DmaxConnections=100
```

-P*[S][integer thread count]* - параллельный запуск + число потоков

```
-P, -PS, -PS 8, or -P8
```

-s <*suite class name*> - тестовый класс

```
-s com.company.project.StackSpec
```

```
...
```

И многое другое...

- Selenium
- Scala.js
- ...

ScalaCheck

ScalaCheck

Property-based тесты, имеют интеграцию со ScalaTest

- https://www.scalacheck.org/
- https://github.com/rickynils/scalacheck

build.sbt:

```
libraryDependencies += "org.scalacheck" %% "scalacheck" % "1.13.4" % "test"
```

ScalaCheck

```
object ScalaCheckExample extends Properties("String") {
  property("startsWith") = forAll { (a: String, b: String) =>
    (a+b).startsWith(a)
  }
  property("concatenate") = forAll { (a: String, b: String) =>
    (a+b).length > a.length && (a+b).length > b.length
  }
  property("substring") = forAll { (a: String, b: String, c: String) =>
    (a+b+c).substring(a.length, a.length+b.length) == b
  }
}
```

ScalaCheck

```
+ String.startsWith: OK, passed 100 tests.
! String.concatenate: Falsified after 0 passed tests.
> ARG_0: ""
> ARG_1: ""
+ String.substring: OK, passed 100 tests.
Found 1 failing properties.
```

ScalaCheck: Conditions

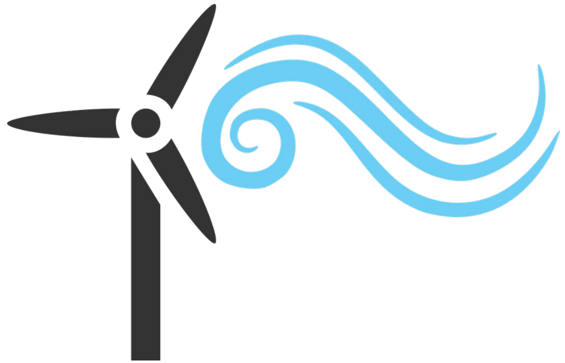
```
import org.scalacheck.Prop.BooleanOperators


forAll { n: Int =>
  (n >= 0 && n < 10000) ==> (List.fill(n)("").length == n)
}
```

ScalaCheck: Generators

```scala
val tupleGen = for {
  n <- Gen.choose(10, 20)
  m <- Gen.choose(2 * n, 500)
} yield (n, m)


val letterGen = Gen.oneOf('A', 'E', 'I', 'O', 'U', 'Y')
```



ScalaCheck: Generators

```
val letterWithProbGen = Gen.frequency(
    (3, 'A'),
    (4, 'E'),
    (2, 'I'),
    (3, 'O'),
    (1, 'U'),
    (1, 'Y')
  )


val smallEvenInteger = Gen.choose(0, 200) suchThat (_ % 2 == 0)
```

ScalaCheck: Generators

```
val genIntList = Gen.containerOf[List, Int](Gen.oneOf(1, 3, 5))
val genStringStream = Gen.containerOf[Stream, String](Gen.alphaStr)
val genBoolArray = Gen.containerOf[Array, Boolean](true)

val genNonEmptySet = Gen.nonEmptyContainerOf[Set, Int](Gen.oneOf(1, 2, 3))
val genSizeNList = Gen.containerOfN[List, Int](5, Gen.oneOf(1, 2, 3, 4))
```

ScalaCheck: Generators

```
property("letters") = forAll(letterGen) { l: Char =>
  l < 'Z'
}

property("integers") = forAll(smallEvenInteger) { i: Int =>
  i > -1
}

property("nonEmptySets") = forAll(genNonEmptySet) { s: Set[Int] =>
  s.nonEmpty
}
```

ScalaCheck

Серия статей на habrahabr (в процессе написания)

https://habrahabr.ru/post/319456/

https://habrahabr.ru/post/320104/

https://habrahabr.ru/post/323038/

ScalaMeter

ScalaMeter

Бенчмарк и performance-тестирование

- https://scalameter.github.io/
- https://github.com/scalameter/scalameter
- https://github.com/scalameter/scalameter-examples

build.sbt:

```
libraryDependencies += "com.storm-enroute" % "scalameter_2.12" % "0.8.2" %
"test"
```

ScalaMeter

```
object ScalaMeterExample extends Bench.LocalTime {
  val sizes = Gen.range("size")(300000, 1500000, 300000)

  val ranges = for { size <- sizes } yield 0 until size
  performance of "Range" in {
    measure method "map" in {
      using(ranges) in {
        r => r.map(_ + 1)
      }
    }
  }
}
```

ScalaMeter

```
::Benchmark Range.map::
cores: 4
hostname: artem-2.local
name: Java HotSpot(TM) 64-Bit Server VM
osArch: x86_64
osName: Mac OS X
vendor: Oracle Corporation
version: 25.71-b15
Parameters(size -> 300000): 2.7651
Parameters(size -> 600000): 5.456531
Parameters(size -> 900000): 8.093276
Parameters(size -> 1200000): 10.786011
Parameters(size -> 1500000): 13.530386
```

ScalaMeter: .par

```
import org.scalameter._


val numbers = Random.shuffle(Vector.tabulate(5000000)(i => i))
val time = config(Key.exec.minWarmupRuns -> 20,
  Key.exec.maxWarmupRuns -> 60,
  Key.exec.benchRuns -> 30,
  Key.verbose -> true) withWarmer(new Warmer.Default) measure {
    numbers.par.max
}
println(s"Parallel time $time")
```

ScalaMeter: .par

```
Starting warmup.
0. warmup run running time: 249.738219 (covNoGC: NaN, covGC: NaN)
1. warmup run running time: 266.063918 (covNoGC: 0,0448, covGC: 0,0448)
...
Steady-state detected.
Ending warmup.
measurements: 222.205909 ms, 221.627235 ms, ...
Parallel time 74.35343833333333 ms
...
Non parallel time 219.57679319999997 ms
```

Akka Test Kit

Akka Test Kit

Акторов нельзя протестировать “обычными” тестами, поэтому у Akka есть своя библиотека для тестирования

http://doc.akka.io/docs/akka/current/scala/testing.html

build.sbt:

```
libraryDependencies += "com.typesafe.akka" %% "akka-testkit" % "2.4.17"
```

Тестирование FSM

```
class FSMActor extends FSM[State, Data] {
  startWith(First, Uninitialized)

  when(First) {
    case Event(Input(d), _) =>
      if (d.length % 2 == 0) {
        stay using Previous(d)
      } else {
        goto(Second) using Previous(d)
      }
  }
```

Тестирование FSM

```
when(Second) {
    case Event(Input(d), _) =>
      if (d.length % 2 == 0) {
        goto(First) using Previous(d)
      } else {
        stay using Previous(d)
      }
  }
  initialize()
}
```

Тестирование FSM

```scala
// received events
case class Input(data: String)

// states
sealed trait State
case object First extends State
case object Second extends State

sealed trait Data
case object Uninitialized extends Data
case class Previous(data: String) extends Data
```

Тестирование FSM

```
class AkkaTestKitFsmExample extends TestKit(ActorSystem("MySpec")) with
FunSuiteLike with Matchers {


  val fsm = TestFSMRef(new FSMActor)
```

Тестирование FSM

```
test("fsm") {
    fsm.stateName should equal( First)
    fsm.stateData should equal ( Uninitialized)
    fsm ! Input("a")
    fsm.stateName should equal( Second)
    fsm.stateData should equal( Previous("a"))
```

Тестирование FSM

```
fsm.setState(First)
fsm.stateName should equal(First)

fsm.isTimerActive("test")should be(false)
fsm.setTimer("test", 12, 10 millis, true)

fsm.isTimerActive("test") should be(true)
fsm.cancelTimer("test")
fsm.isTimerActive("test")should be(false)
```

Тестирование

```
class TestActor extends Actor with ActorLogging {
  override def receive: Receive = {
    case InputMessage(d) =>
      log.info(s"Got message with data $d")
      val response = if (d.length % 2 == 0) {
        d.toUpperCase
      } else {
        d.toLowerCase
      }
      sender() ! OutputMessage(response)
  }
}
```

Тестирование

```
object TestActor {
  def props: Props = Props(classOf[TestActor])
}

case class InputMessage(data: String)
case class OutputMessage(data: String)
```

Тестирование

```
class AkkaTestKitExample extends TestKit(ActorSystem("MySpec")) with
ImplicitSender with FunSuiteLike with Matchers with BeforeAndAfterAll {

  val actor = system.actorOf(TestActor.props)

  override def afterAll {
    TestKit.shutdownActorSystem(system)
  }
```

Тестирование

```
test("Uppercase even length msg") {
  actor ! InputMessage("aa")
  expectMsg {
    OutputMessage("AA")
  }
}
```

TestProbe

```
class MyDoubleEcho extends Actor {
   var dest1: ActorRef = _
   var dest2: ActorRef = _
   def receive = {
     case (d1: ActorRef, d2: ActorRef) =>
       dest1 = d1
       dest2 = d2
     case x =>
       dest1 ! x
       dest2 ! x
   }
 }
```

TestProbe

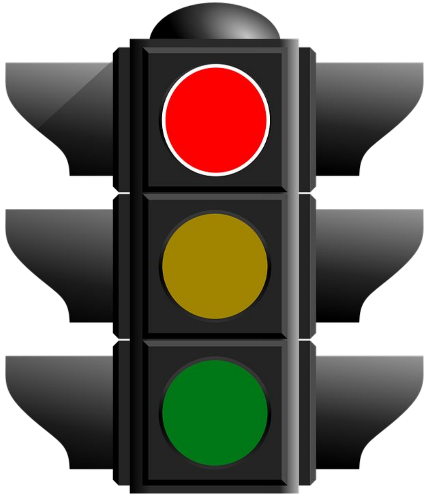
```
test("probes") {
    val doubleEcho = system.actorOf(Props(classOf[MyDoubleEcho]))
    val probe1 = TestProbe()
    val probe2 = TestProbe()
    doubleEcho ! ((probe1.ref, probe2.ref))
    doubleEcho ! "hello"
    probe1.expectMsg(500 millis, "hello")
    probe2.expectMsg(500 millis, "hello")
  }
```

AkkaTestKit: что еще?

- “Прослушка” логов и внутренних ошибок
-

Тестирование Scala: что еще?

- Specs2 https://github.com/etorreborre/specs2
- ScalaProps https://github.com/scalaprops/scalaprops
- µTest https://github.com/lihaoyi/utest
- ...



Спасибо!