Гуляев Г.М.

Современные технологии программирования (часть 2)

Лекция 5. Case - классы и поиск по шаблону (pattern matching)

Курс лекций для студентов АлтГТУ

Равенство объектов

- Проверка на равенство объектов при помощи ==:

```
val a =  "abc"
val b =  "abc"
a == b // true (вызывается метод equals затем eq)
```

- Проверка происходит не по ссылке (как в java), а по значению (внутреннему содержанию объекта):

```
List(1,2,3,4,5) == (1 to 5).toList // true
```

- К сожалению, есть исключение для массивов:

```
Array(1,2,3,4,5) == Array(1,2,3,4,5) // false
```

- Сравнение основано на методах **equals** и **hashCode**. Массивы Scala приводятся к массивам Java и **hashCode** у них вычисляется по ссылке как в Java.
- Для своих классов, без перекрытия этих методов сравнение не работает по значению:

```
class Person(name: String, age: Int)
val p1 = new Person("Иван",21) // Person@393d3e1d
val p2 = new Person("Иван",21) // Person@45cfaf37
p1 == p2 // false
```

Равенство объектов

- Переопределим методы equals и hashCode:

```
class Person(name: String, age: Int) {
  override def hashCode = 13*name.hashCode+age.hashCode*17
  override def equals(other: Any) = {
    val that = other.asInstanceOf[Person]
    if (that == null) false
    else hashCode == that.hashCode // name==that.name&&age==that.age
  }
}
```

- Теперь уже сравнение происходит так как нужно:

```
val p1 = new Person("Иван",21) // Person@1904abe0
val p2 = new Person("Иван",21) // Person@1904abe0
p1 == p2 // true
```

- Проверка на равенство уже будет работать если переопределить только equals. Однако hashCode используется во многих других случаях, например, при работе со списками из данных объектов. Поэтому, если уж переопределять, то оба этих метода.

Case - классы

- В Scala существует возможность сгенерировать методы equals и hashCode (а также toString) автоматически:

```
case class Person(name: String, age: Int)
val p1 = Person("Иван",21) // Person(Иван,21)
val p2 = Person("Иван",21) // Person(Иван,21)
p1 == p2 // true
p1.hashCode // 778101263
p2.hashCode // 778101263
p1.name // Иван
p1.copy(name="Федор") // Person(Федор,21)
```

- Это так называемые case-классы. Отметим еще необязательность new при создании объектов таких классов и автоматически создаваемые геттеры переменных конструктора без объявления val.
- Case - классы убирают однообразные рутинные операции и уменьшают размер кода, делая его более понятным.

Оператор match

- Для выбора варианта из списка по шаблону в Scala существует очень мощное средство - оператор **match**:

```
expr match { // expr - любое выражение
  case pattern1 => result1
  case pattern2 => result2
  ...
}
```

- Простой пример:

```
val list = List(true, 1,"a", false)

for (x <- list) {
  x match {
    case true => println("начало")
    case false => println("конец")
    case _ => println("что-то внутри")
  }
}
```

- Подчеркивание (_) означает значение по-умолчанию (все что осталось). Выбор первого подходящего значения в порядке записи. Подчеркивание должно быть в конце.

Оператор match (примеры)

- **Может быть несколько подчеркиваний в конце:**

```
for (ch <- "+-3!") {
  var sign, digit = 0
  ch match {
    case '+' => sign = 1
    case '-' => sign = -1
    case _ if Character.isDigit(ch) => digit = Character.digit(ch, 10)
    case _ => sign = 0
  }
  println(ch + ": " + sign + " " + digit)
}
```

- **Выбор значения переменной:**

```
def number = {
  import scala.util.Random
  val x = new Random().nextInt(10)
  x  match {
    case 7 => println("Ура, счастливая семерка!")
    case other => println("Увы, это скучное число " + other)
  }
}
```

Оператор match (примеры)

- **Выбор типа объекта:**

```
val list = List(23, "Hello", 8.5, 'q')

for (x <- list) {
  x match {
    case i: Int => println("Это целое: " + i)
    case s: String => println("Это строка: " + s)
    case f: Double => println("Это вещественное: " + f)
    case other => println("Что-то другое: " + other)
  }
}
```

- **Выбор составного типа:**

```
for (o <- Array(Map("Иван" -> 42), Map(42 -> "Иван"), Array(42), Array("Иван"))) {
  println(o match {
    case m: Map[String, Int] => "Это Map[String, Int]"
    case a: Array[Int] => "Это Array[Int]"
    case a: Array[_] => "Это тоже Array, но не Array[Int]"
  })
}
```

Оператор match (примеры)

- **Выбор списка по шаблону:**

```
val list1 = List(1, 3, 23, 90)
val list2 = List(4, 18, 52)
val empty = List()
for (l <- List(list1, list2, empty)) {
  l match {
    case List(_, 3, _, _) => println("4 элемента, второй - 3")
    case List(_*) => println("Какой-то другой список")
  }
}
```

- **Выбор массива по его структуре:**

```
for (arr <- Array(Array(0), Array(1, 0), Array(0, 1, 0), Array(1, 1, 0))) {
  println(arr match {
    case Array(0) => "0"
    case Array(x, y) => x + " " + y
    case Array(0, _*) => "0 ..."
    case _ => "другое"
  })
}
```

Оператор match (примеры)

- **Выбор кортежей:**

```
for (pair <- Array((0, 1), (1, 0), (1, 1))) {
  println(pair match {
    case (0, _) => "0 ..."
    case (y, 0) => y + " 0"
    case _ => "на 1-2 месте не 0"
  })
}
```

- **Выбор кортежей по условию:**

```
for (t <- List(("Доброе", "утро!"), ("Good", "Morning!"))) {
  t match {
    case (one, two) if one == "Good" =>  println("Первый элемент: Good")
    case (one, two) =>  println("Эта пара: " + one + " " + two)
  }
}
```

- **Регулярные выражения:**

```
val pattern = "([0-9]+) ([а-я]+)".r
"99 бутылок" match {
  case pattern(num, item) => (num.toInt, item)
} // (99,бутылок)
```

Оператор match (примеры)

- **Выбор case-классов:**

```
case class Person(name: String, age: Int)
val alice = Person("Алиса", 25)
val bob = Person("Боб", 32)
val charlie = Person("Чарли", 32)
for (person <- List(alice, bob, charlie)) {
  person match {
    case Person("Алиса", 25) => println("Привет Алиса!")
    case Person("Боб", 32) => println("Привет Боб!")
    case Person(name, age) =>
            println(age+" года, зовут "+name+" - кто это?")
  }
}
```

- **Рекурсивный обход List:**

```
def process(l: List[Person]): Unit = l match {
  case head::tail => println(head); process(tail)
  case Nil => println("")
}
process(List(alice,bob,charlie))
```

Оператор match (примеры)

- **Вложенные match:**

```
class Role
case object Manager extends Role
case object Developer extends Role
case class Person(name: String, age: Int, role: Role)

val alice = Person("Алиса", 25, Developer)
val bob = Person("Боб", 32, Manager)
val charlie = Person("Чарли", 32, Developer)

for (item <- Map(1 -> alice, 2 -> bob, 3 -> charlie)) {
  item match {
    case (id, p: Person) => p.role match {
      case Manager => println (p.name+" - высокая зарплата")
      case _ => println (p.name+" - средняя зарплата")
    }
  }
}
```

Выводы

- Поиск по шаблону (pattern matching) является мощным и эффективным способом извлечения информации из объектов
- Благодаря ему в Scala удается избегать необходимости хранить отдельно состояние или иерархию объектов. Он помогает находить решения многих задач в чисто функциональном стиле
- Добавление ключевого слова **case** при объявлении класса заставляет компилятор автоматически добавить в класс ряд полезных функций (геттеры, сеттеры, **toString**, **equals**, **hashCode**)
- Case-классы в сочетании с поиском по шаблону позволяют в нескольких строках кода решать достаточно сложные задачи, требующие в некоторых других языках написания нескольких страниц кода

Спасибо за внимание !

www.altailand.ru