Функциональный стиль в Kotlin

Лекция 2: Коллекции, цепочки вызовов, безопасная обработка, scope-функции и Pair/Triple

Александр Глускер

Курс «Мобильная разработка»

3 апреля 2026 г.

Содержание

Цели лекции

- Научиться писать в функциональном стиле
- Освоить цепочки вызовов без var и mutable коллекций
- Уметь безопасно обрабатывать ошибки
- Подготовиться к семинару

Навигация по лекции

Функциональный стиль — суть

- **Функции как объекты первого класса**: можно передавать, возвращать, хранить
- **Иммутабельность**: данные не изменяются — создаются новые структуры
- **Чистые функции**: одинаковые входы → одинаковые выходы, без побочных эффектов
- **Выражения вместо операторов**: почти всё возвращает значение
- **Декларативность**: описываем *что*, а не *как*

В Kotlin

Поддержка через лямбды, расширения, стандартную библиотеку коллекций, scope-функции.

Почему функциональный стиль?

- Читаемость: цепочки вызовов выражают намерение
- Безопасность: меньше ошибок, особенно с null и состоянием
- Композируемость: легко комбинировать операции
- Тестируемость: чистые функции проще тестировать
- Поддержка многопоточности: иммутабельность → нет гонок

Важно

Не догма — Kotlin позволяет смешивать парадигмы. Используйте функциональный стиль там, где он уместен.

Навигация по лекции

List — упорядоченная коллекция

```kotlin
val numbers = listOf(1, 2, 3, 4, 5)
val empty = emptyList<Int>()

// Индексация
println(numbers[0]) // 1

// Доступкпервомупоследнему  /
println(numbers.first()) // 1
println(numbers.last())  // 5
```

Set — уникальные элементы без порядка

```
1  val unique = setOf(1, 2, 3, 2, 1) // {1, 2, 3}
2  val emptySet = emptySet<Int>()
3
4  // Проверканаличия
5  println(unique.contains(2)) // true
6  println(2 in unique)        // true
7
8  // Операциинадмножествами
9  val a = setOf(1, 2, 3)
10 val b = setOf(3, 4, 5)
11 println(a intersect b) // {3}
12 println(a union b)     // {1, 2, 3, 4, 5}
```

Map — пары ключ-значение

```kotlin
val map = mapOf("a" to 1, "b" to 2, "c" to 3)
val emptyMap = emptyMap<String, Int>()

// Полученииезначения
println(map["a"])           // 1 может( быть null)
println(map.getValue("a"))  // 1 бросит( исключение, еслинет )
println(map.getOrDefault("d", 0)) // 0

// Итерация
map.forEach { (key, value) ->
    println("$key: $value")
}
```

Навигация по лекции

Зачем Sequence?

Проблема: при работе с большими коллекциями, каждый вызов map/filter создает промежуточный список.

```
1  val result = (1..1000000)
2      .map { it * 2 }     // Создаетсписокиз   1M элементов
3      .filter { it > 10 } // Создаетещеодинсписок
4      .first()
```

Решение: Sequence — ленивая коллекция. Операции выполняются по одной на элемент.

```
1  val result = (1..1000000).asSequence()
2      .map { it * 2 }     // Ничегоневычисляется
3      .filter { it > 10 } // Ничегоневычисляется
4      .first()            // Вычисляетсятолькодопервогоподходящего
```

Создание и преобразование Sequence

```
1  // Создание
2  val seq1 = sequenceOf(1, 2, 3)
3  val seq2 = (1..10).asSequence()
4
5  // Генератор
6  val fib = generateSequence(Pair(0, 1)) { (a, b) -> Pair(b, a + b) }
7      .map { it.first }
8      .take(10) // первые 10 чиселФибоначчи
9
10 // Преобразованиевколлекцию
11 val list = seq1.toList()
12 val set = seq1.toSet()
```

Навигация по лекции

map и mapNotNull

```
1 val numbers = listOf(1, 2, 3, 4)
2
3 // map: преобразуеткаждыйэлемент
4 val doubled = numbers.map { it * 2 } // [2, 4, 6, 8]
5
6 // mapNotNull: отфильтровывает null послепреобразования
7 val strings = listOf("1", "a", "3", "b")
8 val ints = strings.mapNotNull { it.toIntOrNull() } // [1, 3]
```

filter, filterNot, filterTo

```
1  val numbers = listOf(1, 2, 3, 4, 5)
2
3  // filter: оставляет, еслиусловие  true
4  val evens = numbers.filter { it % 2 == 0 } // [2, 4]
5
6  // filterNot: оставляет, еслиусловие  false
7  val odds = numbers.filterNot { it % 2 == 0 } // [1, 3, 5]
8
9  // filterTo: записываетрезультатвсуществующуюколлекцию
10 val mutableList = mutableListOf<Int>()
11 numbers.filterTo(mutableList) { it > 3 } // mutableList = [4, 5]
```

take, drop, takeWhile, dropWhile

```kotlin
val numbers = listOf(1, 2, 3, 4, 5)

// take: первые N элементов
println(numbers.take(3)) // [1, 2, 3]

// drop: пропускаетпервые  N
println(numbers.drop(3)) // [4, 5]

// takeWhile: берет, покаусловие  true
println(numbers.takeWhile { it < 4 }) // [1, 2, 3]

// dropWhile: пропускает, покаусловие  true
println(numbers.dropWhile { it < 4 }) // [4, 5]
```

reduce, fold, sum, maxOrNull

```
1  val numbers = listOf(1, 2, 3, 4)
2
3  // reduce: сворачиваетсписокводнозначениепервый     ( элемент—начальный  )
4  val sum1 = numbers.reduce { acc, n -> acc + n } // 10
5
6  // fold: сворачиваетсявнымначальнымзначением
7  val sum2 = numbers.fold(0) { acc, n -> acc + n } // 10
8
9  // sum: суммачисел
10 val sum3 = numbers.sum() // 10
11
12 // maxOrNull: максимальныйэлементилиа  ( null, еслипусто )
13 val max = numbers.maxOrNull() // 4
```

groupBy, associate, partition

```kotlin
val words = listOf("a", "bb", "ccc", "dd")

// groupBy: группируетпоключу
val byLength = words.groupBy { it.length }
// {1=["a"], 2=["bb", "dd"], 3=["ccc"]}

// associate: создает Map поключуизначению
val wordToLength = words.associate { it to it.length }
// {a=1, bb=2, ccc=3, dd=2}

// partition: разделяетнадвечастипоусловию
val (evens, odds) = numbers.partition { it % 2 == 0 }
// evens = [2, 4], odds = [1, 3, 5]
```

flatMap, flatten

```
1 val lists = listOf(listOf(1, 2), listOf(3, 4), listOf(5))
2
3 // flatten: объединяетвложенныесписки
4 val flat = lists.flatten() // [1, 2, 3, 4, 5]
5
6 // flatMap: преобразуеткаждыйэлементвколлекциюиобъединяет
7 val words = listOf("Hello", "World")
8 val chars = words.flatMap { it.toList() } // [H, e, l, l, o, W, o, r, l, d]
```

distinct, sorted, sortedBy

```
1  val numbers = listOf(3, 1, 4, 1, 5)
2
3  // distinct: уникальныеэлементы
4  val unique = numbers.distinct() // [3, 1, 4, 5]
5
6  // sorted: сортировкаповозрастанию
7  val sorted = numbers.sorted() // [1, 1, 3, 4, 5]
8
9  // sortedBy: сортировкапоключу
10 val words = listOf("bb", "a", "ccc")
11 val byLength = words.sortedBy { it.length } // [a, bb, ccc]
```

Навигация по лекции

let: безопасная работа с nullable

```kotlin
val str: String? = "Hello"

// let выполняется, толькоеслиобъектне    null
str?.let {
    println(it.length) // 5
    it.uppercase()     // возвращаетрезультатблока
}

// Частоиспользуетсядляцепочек
val result = str?.let { it.uppercase() } ?: "DEFAULT"
```

run: выполнить блок и вернуть результат

```
1  val result = run {
2      val x = 10
3      val y = 20
4      x + y // возвращаемоезначение
5  }
6  println(result) // 30
7
8  // Можновызыватьнаобъекте
9  val str = "Hello"
10 val length = str.run {
11     println(this) // "Hello"
12     length        // 5
13 }
```

apply: конфигурация объекта

```kotlin
val list = mutableListOf<String>().apply {
    add("a")
    add("b")
    add("c")
}
// list = ["a", "b", "c"]

// Частодляинициализации
val person = Person().apply {
    name = "Alice"
    age = 30
}
```

takeIf и takeUnless: условное продолжение

```kotlin
val number = 15

// takeIf: возвращаетобъект , еслиусловие  true, иначе null
val evenOrNull = number.takeIf { it % 2 == 0 } // null

// takeUnless: возвращаетобъект , еслиусловие  false
val oddOrNull = number.takeUnless { it % 2 == 0 } // 15

// Комбинируемс  let
number.takeIf { it > 10 }
    ?.let { println("Больше 10: $it") } // Больше" 10: 15"
```

also: побочные эффекты без изменения

```
val list = listOf(1, 2, 3).also {
    println("Список создан: $it") // [1, 2, 3]
}.map { it * 2 }.also {
    println("После map: $it") // [2, 4, 6]
}

// Длялогированиявцепочках
val result = computeSomething()
    .also { log("Результат: $it") }
    .process()
```

Навигация по лекции

Цепочка обработки с readlnOrNull()

```
1  fun main() {
2      println("Введите число:")
3      val number = readlnOrNull()
4          ?.toIntOrNull()              // безопасноепреобразование
5          ?.takeIf { it > 0 }          // толькоположительные
6          ?.let { "Результат: ${it * 2}" } // преобразованиевстроку
7          ?: "Ошибка: введитеположительноечисло  "
8
9      println(number)
10 }
```

Обработка списка чисел с ошибками

```
fun main() {
    println("Введите числачерезпробел  :")
    val sum = readlnOrNull()
        ?.split(" ")                    // разбиваемначасти
        ?.mapNotNull { it.toIntOrNull() } // преобразуем, отфильтровываем null
        ?.filter { it > 0 }             // толькоположительные
        ?.sum()                         // сумма
        ?.takeIf { it < 1000 }          // проверкапереполнения
        ?: -1                           // ошибка

    if (sum == -1) {
        println("Ошибка: неверныйвводилисумма    >= 1000")
    } else {
        println("Сумма: $sum")
    }
}
```

Навигация по лекции

Pair: пара значений

```
1  val pair = Pair("Hello", 42)
2  val (str, num) = pair // деструктуризация
3  println(str) // "Hello"
4  println(num) // 42
5
6  // Синтаксическийсахар
7  val pair2 = "Hello" to 42
8
9  // Частоиспользуетсяв   map
10 val map = mapOf("a" to 1, "b" to 2)
```

Triple: тройка значений

```kotlin
val triple = Triple("Alice", 30, "Engineer")
val (name, age, job) = triple
println("$name, $age, $job") // Alice, 30, Engineer

// Пример: возвращаемнесколькозначенийизфункции
fun getUserInfo(): Triple<String, Int, String> {
    return Triple("Bob", 25, "Designer")
}
```

Навигация по лекции

Навигация по лекции

Пример 1: Безопасный парсер координат

```kotlin
fun main() {
    println("Введите: x,y")
    val point = readlnOrNull()
        ?.split(",")
        ?.mapNotNull { it.trim().toDoubleOrNull() }
        ?.takeIf { it.size == 2 }
        ?.let { (x, y) -> "Точка: ($x, $y)" }
        ?: "Ошибка формата"
    println(point)
}
```

Пример 2: Анализ частотности букв

```
fun main() {
    println("Введите текст:")
    val top3 = readlnOrNull()
        ?.filter { it.isLetter() }
        ?.groupingBy { it.lowercaseChar() }
        ?.eachCount()
        ?.toList()
        ?.sortedByDescending { it.second }
        ?.take(3)
        ?.joinToString { "${it.first}=${it.second}" }
        ?: "Нет данных"
    println("Топ-3 буквы: $top3")
}
```

Пример 3: Валидация email-ов

```
1  fun isValidEmail(s: String) = s.count { it == '@' } == 1 &&
2      s.split('@').let { it.size == 2 && it[0].isNotEmpty() && '.' in it[1] }
3
4  fun main() {
5      println("Emailы' через ;")
6      val valid = readlnOrNull()
7          ?.split(";")
8          ?.map { it.trim() }
9          ?.filter { isValidEmail(it) }
10         ?.ifEmpty { null }
11         ?.joinToString("\n")
12         ?: "Нет валидных email"
13     println(valid)
14 }
```

Пример 4: Генератор безопасных паролей

```kotlin
fun main() {
    val chars = ('A'..'Z') + ('a'..'z') + ('0'..'9')
    val pwd = (1..12).map { chars.random() }
        .joinToString("")
        .let { p -> if (p.any { it.isDigit() }) p else "Ошибка генерации" }
    println("Пароль: $pwd")
}
```

Пример 5: Калькулятор выражений

```
fun eval(expr: String): Double? = try {
    val tokens = expr.split(" ")
    tokens.chunked(3).fold(tokens[0].toDouble()) { acc, part ->
        when (part[1]) {
            "+" -> acc + part[2].toDouble()
            "−" -> acc − part[2].toDouble()
            "∗" -> acc ∗ part[2].toDouble()
            "/" -> acc / part[2].toDouble()
            else -> return null
        }
    }
} catch (_: Exception) { null }

fun main() {
    println("Введите: a op b op c ...")
    val result = readlnOrNull()?.let(::eval)?.takeIf { it.isFinite() }
    println("Результат: ${result ?: "Ошибка"}")
}
```

Пример 6: Анализ зависимостей

```
fun main() {
    println("Зависимости: A->B; B->C; ...")
    val deps = readlnOrNull()
        ?.split(";")
        ?.mapNotNull { it.split("->").takeIf { p -> p.size == 2 } }
        ?.associate { it[0].trim() to it[1].trim() }
        ?.let { map ->
            map.keys.filter { it !in map.values }
                .ifEmpty { listOf("Цикл!") }
                .joinToString()
        } ?: "Нет входныхданных "
    println("Корневые: $deps")
}
```

Пример 7: Преобразование единиц измерения

```
1  val units = mapOf("km" to 1000.0, "m" to 1.0, "cm" to 0.01, "mm" to 0.001)
2
3  fun main() {
4      println("Введите: значениеединицаиз _ -> единицав_")
5      val result = readlnOrNull()
6          ?.split(" ")
7          ?.let { (v, from, _, to) ->
8              (v.toDoubleOrNull() ?: return@let null) *
9              (units[from] ?: return@let null) /
10             (units[to] ?: return@let null)
11         }
12         ?.takeIf { it.isFinite() && it > 0 }
13         ?.let { "%.2f".format(it) }
14         ?: "Ошибка конвертации"
15     println("Результат: $result")
16 }
```

Пример 8: Построение маршрута

```kotlin
fun main() {
    println("Маршрут: A–B, B–C, C–D")
    val path = readlnOrNull()
        ?.split(", ")
        ?.map { it.split("–").takeIf { p -> p.size == 2 } }
        ?.filterNotNull()
        ?.fold("") { acc, (from, to) ->
            if (acc.isEmpty() || acc.last() == from[0]) acc + to[0]
            else return@fold "" // разрывмаршрута
        }
        ?.takeIf { it.length > 1 }
        ?: "Невалидный маршрут"
    println("Путь: $path")
}
```

Навигация по лекции

Итоги лекции

- Функциональный стиль — это иммутабельность, чистые функции, выражения
- List, Set, Map — основные коллекции; Sequence — для ленивых вычислений
- Основные методы: map, filter, reduce, groupBy, sorted, take, drop и др.
- Scope-функции: let, run, apply, takeIf — для безопасной и читаемой работы с объектами
- Безопасная обработка: цепочки с ?. и ?: вместо try-catch
- Pair и Triple — для возврата нескольких значений
- Все примеры решены без var и mutable коллекций — как требуется на семинаре

Советы для семинара

- Начинайте с readlnOrNull() — это ваша точка входа
- Используйте ?. для безопасного доступа, ?: для значения по умолчанию
- Разбивайте сложные цепочки на логические блоки (даже если пишете в одну строку)
- Тестируйте на граничных случаях: пустой ввод, null, ноль, отрицательные числа
- Помните: в Kotlin почти всё — выражение. Используйте это!

Удачи на семинаре!