



FOR-COMPREHENSION

FOR-COMPREHENSION

- **for ... yield ...** — это синтаксический сахар, позволяющий записать цепочку вызовов методов `.map`, `.flatMap`, `withFilter` в выражение в императивном стиле
- **Метод withFilter** — аналог `filter`, используемый в `for-yield` выражениях для производительности

FOR-COMPREHENSION

```
val nums = List(2, 5, 1, 7, 4)
```

```
val nums2 =  
  nums.map(x => x * 2)
```

```
val nums = List(2, 5, 1, 7, 4)
```

```
val nums2 =  
  for(x <- nums)  
  yield x * 2
```

FOR-COMPREHENSION

```
val nums = List(2, 5, 1, 7, 4)

val nums2 =
  nums.flatMap(x => 1 to x)
    .map(y => y * 2)
```

```
val nums = List(2, 5, 1, 7, 4)

val nums2 =
  for(x <- nums; y <- 1 to x)
    yield y * 2
```

FOR-COMPREHENSION

```
val nums = List(2, 5, 1, 7, 4)

val nums2 =
  nums.flatMap(x => 1 to x)
    .map(y => y * 2)
```

```
val nums = List(2, 5, 1, 7, 4)

val nums2 = for {
  x <- nums
  y <- 1 to x
} yield y * 2
```

FOR-COMPREHENSION

```
val nums = List(2, 5, 1, 7, 4)
```

```
val nums2 =  
  nums.flatMap(x => 1 to x)  
    .filter(y => y > 3)  
    .map(y => y * 2)
```

```
val nums = List(2, 5, 1, 7, 4)
```

```
val nums2 = for {  
  x <- nums  
  y <- 1 to x if y > 3  
} yield y * 2
```

FOR-COMPREHENSION

```
val nums = List(2, 5, 1, 7, 4)

val nums2 =
  nums.flatMap(x =>
    (1 to x)
    .withFilter(y => y > 3)
    .map(y => y * 2))
```

```
val nums = List(2, 5, 1, 7, 4)

val nums2 = for {
  x <- nums
  y <- 1 to x if y > 3
} yield y * 2
```

FOR-COMPREHENSION

```
val nums = List(2, 5, 1, 7, 4)

val nums2 =
  nums.flatMap(x =>
    (1 to x)
      .withFilter(y => y > 3)
      .map(y => (y, y * 2))
      .flatMap { case (y, y2) =>
        nums
          .withFilter(z => z < y)
          .map(z => z + y2 - y)
      })
```

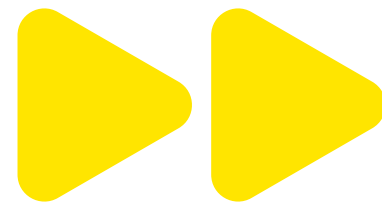
```
val nums = List(2, 5, 1, 7, 4)

val nums2 = for {
  x <- nums
  y <- 1 to x if y > 3
  y2 = y * 2
  z <- nums if z < y2
} yield z + y2 - y
```


ОБЛАСТЬ ПРИМЕНЕНИЯ

- Быстрый выход с Option или Either
- Обход сложных коллекций из `scala.collection.immutable`
- Неблокирующие вычисления с Future, Task, IO
- Комбинаторы асинхронных коллекций Source, Observable, Stream
- Запись вычислений при функциональном подходе

**В этом разделе мы изучили
for-comprehension**



**В следующем
продолжим**