

1.3. Null-безопасность

Сайт: [Samsung Innovation Campus](https://innovationcampus.ru)
Курс: Мобильная разработка на Kotlin
Книга: 1.3. Null-безопасность

Напечатано:: Murad Rezvan
Дата: понедельник, 3 июня 2024, 17:41

Оглавление

- 1.3.1. Тип `null` в Kotlin
- 1.3.2. Оператор безопасного вызова (`?.`)
- 1.3.3. Оператор объединения по `null` (`?:`)
- 1.3.4. Оператор контроля `non-null` (`!!`)

1.3.1. Тип null в Kotlin

В данном разделе речь пойдет о важной особенности языка Kotlin, связанной с контролем обращений к объектам, которые потенциально способны получить null значение в процессе выполнения программы. Обращение к null-объектам - это очень частая ошибка, возникающая при программировании на Java, даже в уже отлаженных программах. В языке Kotlin введен механизм, позволяющий очень точно контролировать значения ссылочных типов. Этому механизму, обеспечивающему null-безопасность, и посвящен данный раздел. Первый этап обеспечения null-безопасности заключен в самой системе типов Kotlin. Система типов в языке Kotlin нацелена на то, чтобы искоренить опасность обращения к null значениям, более известную как "Ошибка на миллиард".

Самым распространённым подводным камнем многих языков программирования, в том числе Java, является попытка произвести доступ к null значению. Это приводит к ошибке. В Java такая ошибка называется NullPointerException (сокр. "NPE").

Kotlin призван исключить ошибки подобного рода из нашего кода. NPE могут возникать только в случае:

- Явного указания throw NullPointerException();
- Использования оператора !! (описано ниже);
- Эту ошибку вызвал внешний Java-код;
- Есть какое-то несоответствие при инициализации данных (в конструкторе использована ссылка this на данные, которые не были ещё проинициализированы).

Система типов Kotlin различает ссылки на те, которые могут иметь значение null (nullable ссылки) и те, которые таковыми быть не могут (non-null ссылки). Типы, которые допускают значение null, должны быть помечены вопросительным знаком, например

```
var k:Int = 0 //null - недопустимое значение k
var k1:Int? = null //null может быть значением k1
```

[Open in Playground →](#)

Target: JVM Running on v.2.0.0

Это верно для всех типов. Таким образом, если вы собираетесь использовать значение null, вы должны выбрать соответствующий тип.

Рассмотрим еще один пример: переменная часто используемого типа String не может быть null:

```
var a: String = "abc"
a = null // ошибка компиляции
```

[Open in Playground →](#)

Target: JVM Running on v.2.0.0

Для того, чтобы разрешить null значение, мы можем объявить эту строковую переменную как String?:

```
var b: String? = "abc"
b = null // ok
```

[Open in Playground →](#)

Target: JVM Running on v.2.0.0

Теперь, при вызове метода с использованием переменной a, исключены какие-либо NPE. Вы спокойно можете писать:

```
val l = a.length
```

[Open in Playground →](#)

Target: JVM Running on v.2.0.0

Но в случае, если вы захотите получить доступ к значению b, это будет небезопасно. Компилятор предупредит об ошибке:

```
val l = b.length // ошибка: переменная b может быть null
```

[Open in Playground →](#)

Target: JVM Running on v.2.0.0

Но нам по-прежнему надо получить доступ к этому свойству/значению, так? Есть несколько способов этого достичь.

Первый способ. **Проверка на null.** Вы можете явно проверить b на null значение и обработать два варианта по отдельности:

```
val l = if (b != null) b.length else -1
```

[Open in Playground →](#)

Target: JVM Running on v.2.0.0

Компилятор отслеживает информацию о проведенной вами проверке и позволяет вызывать length внутри блока if. Также поддерживаются более сложные конструкции:

```
if (b != null && b.length > 0) {
    print("String of length ${b.length}")
} else {
    print("Empty string")
}
```

[Open in Playground →](#)

Target: JVM Running on v.2.0.0

Обратите внимание: это работает только в том случае, если `b` является неизменной переменной (ориг.: `immutable`). Например, если это локальная переменная, значение которой не изменяется в период между его проверкой и использованием. Также такой переменной может служить `val`. В противном случае может так оказаться, что переменная `b` изменила своё значение на `null` после проверки.

1.3.2. Оператор безопасного вызова (?.)

Вторым способом является оператор безопасного вызова ?.:

```
println(b?.length)
```

[Open in Playground →](#)

Target: JVM Running on v.2.0.0

Этот код возвращает `b.length` в том, случае, если `b` не имеет значение `null`. Иначе он возвращает `null`. Типом этого выражения будет `Int?`.

Такие безопасные вызовы полезны в цепочках. К примеру, если `Bob`, `Employee` (работник), может быть прикреплен (или нет) к отделу `Department`, и у отдела может быть управляющий, другой `Employee`. Для того, чтобы обратиться к имени этого управляющего (если такой есть), напишем:

```
bob?.department?.head?.name
```

[Open in Playground →](#)

Target: JVM Running on v.2.0.0

Такая цепочка вернёт `null` в случае, если одно из свойств имеет значение `null`.

Для проведения каких-либо операций исключительно над non-null значениями вы можете использовать `let` оператор вместе с оператором безопасного вызова:

```
val listWithNulls: List<String?> = listOf("A", null)
for (item in listWithNulls) {
    item?.let { println(it) } // выводит A и игнорирует null
}
```

[Open in Playground →](#)

Target: JVM Running on v.2.0.0

1.3.3. Оператор объединения по null (?:)

Оператор `?:` позволяет объединить проверку значения объекта на `null` и выполнение функции этого объекта. Этот оператор ещё называют Эллис-оператором. Не путайте его с похожим тернарным оператором в языках C/C++ и Java. В Kotlin оператор `?:` работает совершенно иначе. Разберем его работу на примерах.

Начнём "издалека". У строк есть свойство `length`, которое возвращает длину строки в символах. Если мы выбрали тип объекта `String?` (разрешающий `null` значение), то мы просто так не можем обратиться к свойству `length`, так как если объект `String?` равен `null`, то и строки как таковой нет, и, соответственно, длину строки нельзя определить. В этом случае мы можем применить оператор `?:`. (см. "Тип `null` в Kotlin"):

```
val name : String? = "Tom"
val length: Int? = name?.length
```

[Open in Playground →](#)

Target: JVM Running on v.2.0.0

Если переменная `name` вдруг равна `null`, то переменная `length` получит значение `null`. Если переменная `name` содержит строку, то возвращается длина этой строки. Это так называемый безопасный вызов метода или свойства. По сути выражение `val length: Int? = name?.length` эквивалентно следующему коду:

```
val length: Int?
if(name != null)
    length = name.length
else
    length = null
```

[Open in Playground →](#)

Target: JVM Running on v.2.0.0

С помощью оператора `?:` подобным образом можно обращаться к любым свойствам и функциям объекта. Однако, у оператора безопасного вызова есть недостаток, суть которого в том, что в рассмотренном примере нам нужно получить именно число, как длину строки. `Null` нас не интересует. Хотелось бы, чтобы в случае, если строка `name = null`, длина строки получила бы значение `0`, а не `null`. Для такой цели оператор безопасного вызова `?:` не годится, а Эллис оператор `?:` очень полезен. Посмотрим, как выглядит применение Эллис-оператора.

```
val name : String? = "Tom"
val length: Int = name?.length ?: 0
```

[Open in Playground →](#)

Target: JVM Running on v.2.0.0

Теперь переменная `length` не допускает значения `null`. И если переменная `name` не определена, то `length` получает число `0`. Оператор `?:` обеспечивает саму возможность вызова свойства `length`, а оператор `?:` определяет какое значение должно присвоиться в `length`, если `name = null`

1.3.4. Оператор контроля non-null (!!)

Оператор !! (not-null assertion operator) принимает один операнд. Если операнд равен null, то генерируется исключение kotlin.KotlinNullPointerException. Если операнд не равен null, то возвращается его значение.

```
val company : String? = "Samsung"
val id: String = company!!
println(id)
```

[Open in Playground →](#)

Target: JVM Running on v.2.0.0

В примере выше есть уверенность, что переменная company гарантированно не null. Тогда оператор !! позволяет присвоить обычной строке типа String значение из строки с разрешенным null значением типа String?.

!! - опасная операция и применять ее следует только тогда, когда есть уверенность, что значение операнда точно не null.

В примере ниже также есть уверенность, что переменная company не null, значит после применения оператора !! мы можем обратиться к методам и свойствам этого объекта:

```
val company : String? = null
val length :Int = company!!.length
```

[Open in Playground →](#)

Target: JVM Running on v.2.0.0

Особенность: если все же значение операнда равно null, то будет сгенерировано исключение kotlin.KotlinNullPointerException.

[Начать тур для пользователя на этой странице](#)