Глава 11. Опциональные типы данных

Как вы знаете, типы данных обеспечивают хранение информации различных видов: строковой, логической, числовой и т. д. Помимо фундаментальных типов, изученных ранее, в Swift есть также опциональные типы данных. Это одно из важных нововведений языка, которое, вероятно, не встречалось вам ранее.

Все переменные и константы, которые мы определяли до этого момента, всегда имели некоторое конкретное значение.

Когда вы объявляете какой-либо параметр, например `var name: String`, то с ним ассоциируется определенное значение, например `Владимир`, которое **всегда** возвращается по имени данного параметра. Значение всегда имеет определенный тип, даже если это пустая строка, пустой массив и т. д. Это одна из функций безопасного программирования в Swift: если объявлен параметр определенного типа, то при обращении к нему вы гарантированно получите значение этого типа. Без каких-либо исключений!

В этой главе вы познакомитесь с концепцией опционалов, позволяющих представлять не только значение определенного типа, но и полное отсутствие какого-либо значения.

11.1. Введение в опционалы

Опциональные типы данных, также называемые **опционалами**, — это особый тип, который говорит о том, что параметр либо имеет значение определенного типа, либо вообще не имеет никакого значения. Иногда очень полезно оперировать отсутствием значения. Рассмотрим два примера.

Пример 1

Представьте, что перед вами бесконечная двумерная плоскость (с двумя осями координат). В ходе эксперимента на ней устанавливают точку с координатами `x=0` и `y=0`, которые в коде могут быть представлены либо как два целочисленных параметра (`x:Int` и `y:Int`), либо как кортеж типа `(Int, Int)`. В зависимости от ваших потребностей вы можете передвигать точку, изменяя ее координаты. В любой момент времени вы можете говорить об этой точке и получать конкретные значения `x` и `y`.

Что произойдет, если убрать точку с плоскости? Она все еще будет существовать в вашей программе, но при этом не будет иметь координат. Совершенно никаких координат. В данном случае `x` и `y` не могут быть установлены в том числе и в `0`, так как `0` — это точно такая же координата, как и любая другая.

Данная проблема может быть решена с помощью введения дополнительного параметра (например, `isSet: Bool`), определяющего, установлена ли точка на плоскости. Если `isSet = true`, то можно производить операции с координатами точки, в ином случае считается, что точка не установлена на плоскости. При таком подходе велика вероятность ошибки, так как необходимо контролировать значение `isSet` и проверять его перед каждой операцией с точкой.

В такой ситуации наиболее верным решением станет использование опционального типа данных в качестве типа значения, определяющего координаты точки. В случае, когда точка находится на плоскости, будут возвращаться конкретные целочисленные значения, а когда она убрана — специальное ключевое слово, определяющее отсутствие координат (а значит, и точки на плоскости).

Пример 2

Ваша программа запрашивает у пользователя его имя, возраст и место работы. Если первые два параметра могут быть определены для любого пользователя, то конкретное рабочее место может отсутствовать. Конечно же, чтобы указать на то, что работы нет, можно использовать пустую строку, но опционалы, позволяющие определить отсутствие значения, являются наиболее правильным решением. Таким образом, обращаясь к переменной, содержащей место работы, вы будете получать либо конкретное строковое значение, либо специальное ключевое слово, сигнализирующее об отсутствии работы.

Самое важное, чего позволяют достичь опционалы, — это исключение неоднозначности. Если значение есть, то оно есть, если его нет, то оно не сравнивается с нулем или пустой строкой, его просто нет.

> **ПРИМЕЧАНИЕ** Важно не путать отсутствие какого-либо значения в опциональном типе данных с пустой строкой или нулем. Пустая строка — это обычный строковый литерал, то есть вполне конкретное значение переменной типа `String`, а ноль — вполне конкретное значение числового типа данных. То же относится и к пустым коллекциям.

У вас мог возникнуть вопрос: как Swift показывает, что в параметре опционального типа отсутствует значение? Для этого используется ключевое слово `nil`. С ним мы, кстати, уже встречались, когда изучали коллекции.

Рассмотрим практический пример использования опционалов.

Ранее мы неоднократно использовали функцию `Int(_:)` для создания и приведения целочисленных значений. Но не каждый переданный в нее литерал может быть преобразован к целочисленному типу данных: к примеру, строку `1945` можно конвертировать в число, а `Одна тысяча сто десять` вернуть в виде числа не получится (листинг 11.1).

Листинг 11.1

```
let possibleString = "1945"
let convertPossibleString = Int(possibleString) // 1945

let unpossibleString = "Одна тысяча сто десять"
let convertUnpossibleString = Int(unpossibleString) // nil
```

При конвертации строкового значения `1945`, состоящего только из цифр, возвращается число. А во втором случае возвращается ключевое слово `nil`, сообщающее о том, что в результате конвертации не получено никакого целочисленного значения. То есть это не ноль, это не пустая строка, а именно отсутствие значения как такового.

Самое интересное, что в обоих случаях (и при числовом, и при строковом значении переданного аргумента) возвращается значение опционального типа данных. То есть `1945` — это значение не целочисленного, а опционального целочисленного типа данных. Также и `nil` — **в данном примере** это указатель на отсутствие значения в хранилище опционального целочисленного типа.

В этом примере функция `Int(_:)` возвращает опционал, то есть значение такого типа данных, который может либо содержать конкретное значение (целое число), либо не содержать совершенно ничего (`nil`).

Опционалы — это отдельная самостоятельная группа типов данных. Целочисленный тип и опциональный целочисленный тип — это два совершенно разных типа данных. По этой причине опционалы должны иметь собственное обозначение типа. И они его имеют. Убедимся в этом, определив тип данных констант из предыдущего листинга (листинг 11.2).

Листинг 11.2

```
type(of: convertPossibleString)   // Optional<Int>.Type
type(of: convertUnpossibleString) // Optional<Int>.Type
```

`Optional<Int>` — это идентификатор **опционального целочисленного типа данных**, то есть значение такого типа может либо быть целым числом, либо отсутствовать полностью. Тип `Int` является базовым для этого опционала, то есть основан на типе `Int`.

Более того, опциональные типы данных всегда строятся на основе базовых неопциональных. Они могут брать за основу совершенно любой тип данных, включая `Bool`, `String`, `Float` и `Double`, а также типы данных кортежей, ваши собственные типы, типы коллекций и т. д.

Напомню, что опционалы являются самостоятельными типами, отличными от базовых, то есть тип `Int` и тип `Optional<Int>` — это два разных типа данных.

> **ПРИМЕЧАНИЕ** Функция `Int(_:)` не всегда возвращает опционал, а лишь в том случае, если в нее передано нечисловое значение. Так, если в `Int(_:)` передается значение типа `Double`, то нет никакой необходимости возвращать опционал, так как при любом значении `Double` оно может быть преобразовано в `Int` (чего нельзя сказать про преобразование `String` в `Int`).

Далее показано, что приведение `String` и `Double` к `Int` дает значения различных типов данных (`Optional<Int>` и `Int`).

```
let x1 = Int("12")
type(of: x1) // Optional<Int>.Type
let x2 = Int(43.2)
type(of: x2) // Int.Type
```

В общем случае тип данных опционала имеет две формы записи.

СИНТАКСИС

Полная форма записи:

```
Optional<T>
```

Краткая форма записи:

```
T?
```

`T: Any` — наименование типа данных, на котором основан опционал.

При объявлении параметра, имеющего опциональный тип, **необходимо явно указать его тип данных**. Для этого можно использовать полную форму записи. В листинге 11.3 приведен пример объявления переменной опционального типа, основанного на `Character`.

Листинг 11.3

```
let optionalChar: Optional<Character> = "a"
```

При объявлении опционала Swift также позволяет использовать сокращенный синтаксис. Для этого в конце базового типа необходимо добавить знак вопроса, никаких других элементов не требуется. Таким образом, тип `Optional<Int>` может быть переписан в `Int?`, `Optional<String>` в `String?` и в любой другой тип. В листинге 11.4 показан пример объявления опционала с использованием сокращенного синтаксиса.

Листинг 11.4

```
var xCoordinate: Int? = 12
```

В любой момент значение опционала может быть изменено на `nil`. Это можно сделать как и при объявлении параметра, так и потом (листинг 11.5).

Листинг 11.5

```
xCoordinate // 12
xCoordinate = nil
xCoordinate // nil
```

Переменная `xCoordinate` является переменной опционального целочисленного типа данных `Int?`. Изначально ей было присвоено значение, соответствующее базовому для опционала типу данных, которое позже было заменено на `nil` (то есть значение переменной было уничтожено).

Если объявить переменную опционального типа, но не проинициализировать ее значение, Swift по умолчанию сочтет ее равной `nil` (листинг 11.6).

Листинг 11.6

```
var someOptional: Bool? // nil
```

Для создания опционала помимо явного указания типа также можно использовать функцию `Optional(_:)`, в которую необходимо передать инициализируемое значение требуемого базового типа (листинг 11.7).

Листинг 11.7

```
// опциональная переменная с установленным значением
var optionalVar = Optional("stringValue") // "stringValue"
// уничтожаем значение опциональной переменной
optionalVar = nil // nil
type(of: optionalVar) // Optional<String>.Type
```

Так как в функцию `Optional(_:)` в качестве аргумента передано значение типа `String`, то возвращаемое ею значение имеет опциональный строковый тип данных `String?` (или `Optional<String>`, что является синонимами).

Опционалы в кортежах

Так как в качестве базового для опционалов может выступать любой тип данных, вы можете использовать в том числе и кортежи. В листинге 11.8 приведен пример объявления опционального кортежа.

Листинг 11.8

```
var tuple: (code: Int, message: String)? = nil
tuple = (404, "Page not found") // (code 404, message "Page not found")
```

В этом примере опциональный тип основан на типе кортежа `(Int, String)`.

При необходимости вы можете использовать опционал для отдельных элементов кортежей (листинг 11.9).

Листинг 11.9

```
let tupleWithoptelements: (Int?, Int) = (nil, 100)
tupleWithoptelements.0 // nil
tupleWithoptelements.1 // 100
```

11.2. Извлечение опционального значения

Важно отметить, что нельзя производить прямые операции между значениями опционального и базового типов данных, так как это приведет к ошибке (листинг 11.10).

Листинг 11.10

```
let a: Int = 4
let b: Int? = 5
a+b // ОШИБКА. Несоответствие типов
```

В переменной `a` хранится значение неопционального типа `Int`, в то время как значение `b` является опциональным (`Int?`).

Типы `Int?` и `Int`, `String?` и `String`, `Bool?` и `Bool` — все это разные типы данных. Для решения проблемы их взаимодействия можно применить прием, называемый **извлечением опционального значения**, или, другими словами, преобразовать опционал в соответствующий ему базовый тип.

Выделяют три способа извлечения опционального значения:

- принудительное извлечение;
- косвенное извлечение;
- операция объединения с `nil` (рассматривается в конце главы).

После извлечения значение опционального типа приводится к базовому, а значит, может взаимодействовать с другими значениями базового типа. Рассмотрим каждый из указанных способов подробнее.

Принудительное извлечение значения

Принудительное извлечение (force unwrapping) преобразует значение опционального типа в значение базового (например, `Int?` в `Int`) с помощью знака восклицания (`!`), указываемого после имени параметра с опциональным значением. Пример принудительного извлечения приведен в листинге 11.11.

Листинг 11.11

```
var optVar: Int? = 12
var intVar = 34
let result = optVar! + 34 // 46

// проверяем тип данных извлеченного значения
type(of: optVar!) // Int.Type
```

Константа `optVar` — это опционал. Для проведения арифметической операции с целочисленным значением используется принудительное извлечение (после имени переменной указан восклицательный знак). Таким образом, операция сложения производится между двумя неопциональными целочисленными значениями.

Точно такой же подход используется и при работе с типами, отличными от `Int` (листинг 11.12).

Листинг 11.12

```
let optString: String? = "Vasiliy Usov"
let unwrapperString = optString!
print( "My name is \(unwrapperString)" )
```