Коллекции. Основы

В Swift standard library есть три всем известные структуры данных: массивы, словари и множества. Они содержат соответственно упорядоченный набор элементов, неупорядоченный набор пар ключ-значение и неупорядоченный набор уникальных элементов.

В отличии от Objective-C в Swift нет разделения на изменяемые и неизменяемые коллекции. Это будет зависеть от того объявили ли вы коллекцию как константу или как переменную.

// --- Коллекции CODE SNIPPET #1 --- //

```
var mutableArray: [Int] // Изменяемый массив целых чисел
let immutableDictionary: [String: String] // Неизменяемый словарь строк
var mutableSet: Set<Int> // Изменяемое множество целых чисел
```

Не забывайте, что неизменяемость относится только к содержимому самой коллекции. Т.е. если коллекция содержит объекты, то гарантируется только то, что указатель на них не изменится. При этом данные внутри самих объектов могут изменяться.

Тоже самое происходит при копировании коллекций. В Swift все коллекции являются value type. Поэтому при передаче коллекции в функцию можно не беспокоиться, что функция изменит вашу коллекцию даже если она объявлена как var.

// --- Коллекции CODE SNIPPET #2 --- //

```
var arrayOfInt = [1, 2, 3]
var copyOfArrayOfInt = arrayOfInt
copyOfArrayOfInt[1] = 5
copyOfArrayOfInt // [1, 5, 3]
arrayOfInt           // [1, 2, 3]
```

Для инициализации коллекций можно использовать литералы. При этом, как и для простых переменных, тип коллекции указывать не обязательно.

// --- Коллекции CODE SNIPPET #3 --- //

```
let fibs = [0, 1, 1, 2, 3, 5] // [Int]
let JSON = ["language":"Swift", "version":"4.0"]
```

// --- Коллекции CODE SNIPPET #3 --- //

Множеству можно присваивать литерал массива. При этом все повторяющиеся элементы будут отброшены.

// --- Коллекции CODE SNIPPET #4 --- //

```
let uniqueNumbers: Set = [1, 2, 3, 3, 4] // Полностью тип указывать не обязательно. Swift поймет, что uniqueNumbers должен быть Set<Int>
uniqueNumbers // [2, 3, 1, 4] Порядок элементов во множестве не гарантируется
```

Элементы множества и ключи в коллекции должны поддерживать протокол Hashable. Это необходимо т.к. имплементация множеств и коллекций построена на хэш таблицах.

Для получения элементов можно использовать subscipt.

// --- Коллекции CODE SNIPPET #5 --- //

```
fibs[2] // 1
JSON["version"] // "4.0"
```

Помимо обычного subscript в коллекциях есть возможность получить сразу диапазон элементов по индексам.

// --- Коллекции CODE SNIPPET #6 --- //

```
let slice = fibs[2..<fibs.endIndex]
slice // [1, 2, 3, 5]
type(of: slice) // ArraySlice<Int>
```

В общем случае такой вызов возвращает Slice. Для массива определена коллекция ArraySlice. С ним можно осуществлять те же операции, что и с обычным массивом. При создании Slice не происходит копирования элементов. Вместо этого он хранит указатель на оригинальный массив, начало и конец выбранного диапазона.

Разумеется мы можем использовать в Swift все коллекции из Foundation. Однако, следует всегда помнить, что их поведение отличается. Например, при использовании NSArray нужно помнить о том, что он не value type и может измениться несмотря на то, что мы думаем, что это NSArray и он неизменяемый. Для решения этой проблемы нужно делать явную копию NSArray.

// --- Коллекции CODE SNIPPET #7 --- //

```
let nsmutableArray = NSMutableArray(array: [1, 2, 3])
let immutableReference: NSArray = nsmutableArray
let immutableCopy = nsmutableArray.copy() as! NSArray

nsmutableArray[0] = 6
immutableReference // [6, 2, 3]
```

```
immutableCopy // [1, 2, 3]
```

Некоторые коллекции перенесены в Swift как value type и даже поддерживают протоколы коллекций. IndexSet используется для хранения индексов. Его преимущества перед простым Set в том, что он хранит индексы как набор диапазонов. Но вы при этом можете работать с ним как с обычным множеством.

```
// --- Коллекции CODE SNIPPET #8 --- //

var indices = IndexSet()
indices.insert(integersIn: 1..<5)
indices.insert(integersIn: 11..<15)

let evenIndices = indices.filter { $0 % 2 == 0 } // [2, 4, 12, 14]
evenIndices.contains(4) // true
```

Еще один пример это CharacterSet. Он используется для хранения набора Unicode совместимых символов. Он поддерживает операции над множествами однако коллекцией не является.

В обратную сторону перенос тоже работает. До Swift 3.0 для использования Swift коллекций в Objective-C все его объекты должны были быть наследниками AnyObject. Сейчас любая коллекция может быть использована в Objective-C. При этом примитивные типы будут автоматически обернуты в box class т.к. Obj-c коллекции могут содержать только объекты.