



Синтаксис Objective-C. Коллекции и типы данных.



Макаровская Вероника Михайловна @MVeronika

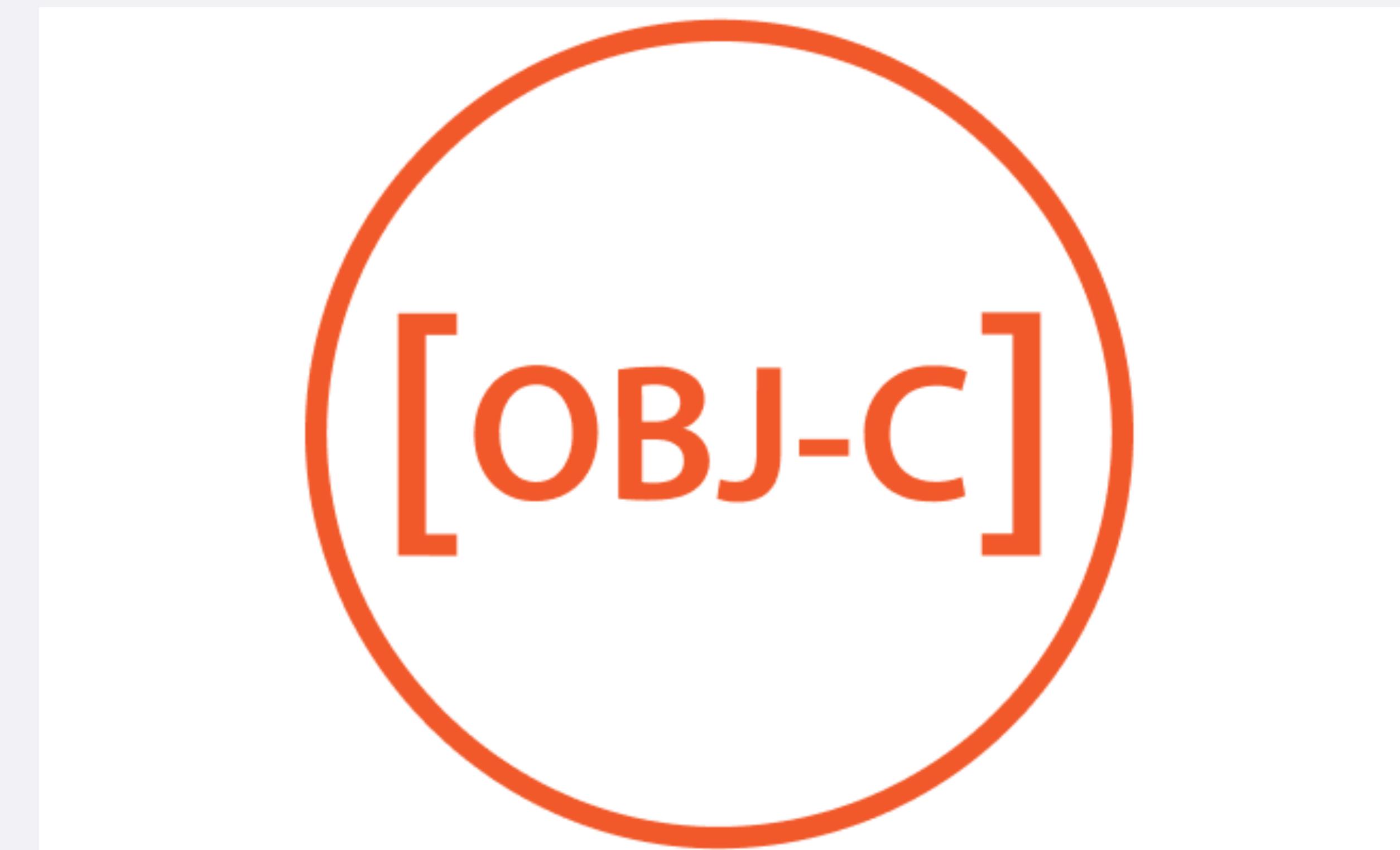


Devteam



Что мы узнаем сегодня?

- Синтаксис Objective-C
- C
- NSArray
- NSDictionary
- NSSet





Синтаксис Objective-C



Токены

- Программа Objective-C состоит из различных токенов.
- Токеном являются ключевые слова, идентификаторы, константы, строковые литералы или символы.
- `NSLog(@"Hello, World! \n");`



Точка с запятой;

- В программе на Objective-C точка с запятой является разделителем операторов.
- То есть каждое отдельное утверждение должно заканчиваться точкой с запятой. Это указывает на конец одного логического объекта.



Комментарии

- Игнорируются компилятором.
- Они начинаются с `/*` и заканчиваются символами `*/`
- Для однострочных комментариев можно использовать `//`
- Нельзя иметь комментарии в комментариях.
- Они не встречаются внутри строковых или символьных литералов.



Идентификаторы

- Идентификатор — это имя, используемое для идентификации переменной, функции или любого другого пользовательского элемента. Идентификатор начинается с буквы от A до Z или от a до z или подчеркивания _, за которым следуют ноль или более букв, подчеркиваний и цифр (от 0 до 9).
- Objective-C является регистрозависимым языком программирования.



Ключевые слова

auto	else	long	switch
break	enum	register	typedef
case	extern	return	union
char	float	short	unsigned
const	for	signed	void
continue	goto	sizeof	volatile
default	if	static	while
do	int	struct	_Packed
double	protocol	interface	implementation
NSObject	NSInteger	NSNumber	CGFloat
property	nonatomic;	retain	strong
weak	unsafe_unretained;	readonly	readwrite



Пробелы

- Стока, содержащая только пробел, возможно, с комментарием, называется пустой строкой, и компилятор Objective-C полностью игнорирует ее.
- Пробелы отделяют одну часть оператора от другой и позволяют компилятору определить, где заканчивается один элемент в выражении.



Типы данных



Типы данных

- Objective-C является надмножеством C (это значит, что любая программа написанная на C будет являться программой на Objective-C), из этого следует, что все типы данных из C будут присутствовать в Objective-C.
- То есть нам доступны как примитивные типы int, float, double char, указатели, так и C-массивы, структуры, перечисления, объединения.



Типы данных

- Кроме этого в Objective-C добавляются свои псевдонимы над примитивами и собственные объектные типы для работы со строками `NSString`, с числами `NSNumber`, с датами `NSDate`, с кодом как с объектами (это блоки, но сейчас мы их рассматривать не будем).



Примитивы С



Тип Void

- Void в переводе пустота.
- Функция, возвращающая тип void, не возвращает ничего.
- Не стоит путать тип void с указателем void*.



Целочисленные типы

- Целочисленные типы данных характеризуются их длиной и наличием знака (`signed` и `unsigned`).
- Тип `char` всегда занимает 1 байт, но нужно очень хорошо запомнить, что конкретные размеры целочисленных типов зависят от реализации.
- Гарантируется только что `short <= int <= long <= long long`.
- `BOOL` является целым и является частью Objective-C, а не C.



Целые фиксированной длины

- Так как фактический размер целочисленных зависит от реализации, то для некоторых случаев более удобно выбрать тип с определенной длиной.



Числа с плавающей точкой

- В С присутствуют три типа для чисел с плавающей точкой.
- Картина с ними такая же как и с целочисленными в С, их размер тоже зависит от реализации, дается гарантия что float \leq double \leq long double.
- В коде литералы представляющие float должны заканчиваться на f, а long double на L.



Определение размера для типа

- Для определения размера используется функция `sizeof()`, которая возвращает количество байт, которое тип занимает в памяти. Использование данной функции является самым простым способом посмотреть какой размер имеет тот или иной зависимый от реализации тип на конкретной архитектуре.
- При использовании `sizeof()` на массиве вернется количество байт занимаем массивом.



Какое количество
элементов можно хранить
в массиве?



```
size_t numberOfElements =  
    sizeof(anArray)/  
    sizeof(anArray[0]);
```



Выбор целых

- Так как в С довольно много вариантов целочисленных типов, не всегда понятно какой из них выбрать.
- Совет простой - используйте `int` всегда, когда у вас нет причины использовать какой другой тип.
- Для индекса массивов стоит использовать целые без знака (индексы никогда не бывают отрицательные).
- `int` традиционно близок к нативному машинному слову текущей архитектуры.



Деление целых

- Если в операции деления оба аргументы целые, то будет использоваться целочисленная арифметика, однако если хотя бы один аргумент будет с плавающей точкой, то будет использоваться арифметика с плавающей точкой, с неявным преобразованием типа аргумента. Это стоит помнить.



Сравнение чисел с плавающей точкой на равенство

- Необходимо запомнить, что числа с плавающей точкой нельзя сравнивать на равенство.



Приimitives Objective-C



Примитивы Objective-C

- К имеющимся примитивам С добавляются примитивы Objective-C: `id`, `SEL`, `Class`, которые являются базисом динамической природы Objecive-C, также кратко рассмотрим псевдонимы `NSInteger` и `NSUInteger`.



Id

- id это общий тип для всех объектов в Objective-C, для простоты понимания его можно сравнивать с указателем void * из C. И также как и void *, id может хранить ссылку на любой объект.

```
id mysteryObject = @"An NSString object";
 NSLog(@"%@", [mysteryObject description]);
mysteryObject = @{@"model": @"Ford", @"year": @1967};
 NSLog(@"%@", [mysteryObject description]);
```



Class

- Класс представляет собой сам класс объекта.

Воспользовавшись им можно, например проверить
динамически проверить тип объекта:

```
Class targetClass = [NSString class];
id mysteryObject = @"An NSString object";
if ([mysteryObject isKindOfClass:targetClass]) {
    NSLog(@"Yup! That's an instance of the target class");
}
```



SEL

- Тип SEL - внутреннее представление имени метода. Манипуляции с SEL используются для различных операций связанных с динанизмом языка. В примере ниже имя метода сохраняется в переменную:

```
SEL someMethod = @selector(sayHello);
```



NSInteger, NSUInteger

- Код написанный на Objective-C работает на разных типах архитектур: 32-битных, 64-битных. Из-за разного размера С-типов писать код под разные архитектуры не всегда удобно и не всегда приятно. Поэтому для упрощения перехода между архитектурами были сделаны NSInteger и NSUInteger, которые дают единый размер целочисленной переменной на разных архитектурах.



Коллекции в Objective-C



Изменяемость и неизменяемость

- В Objective-C для изменяемых типов в имени будет присутствовать часть «Mutable».
- Примеры, `NSString`, `NSMutableString`, `NSDictionary`, `NSMutableDictionary`, `NSArray`, `NSMutableArray` и т.п.
- Это означает, что однажды созданный объект нельзя модифицировать.
- Удобно при работе с несколькими потоками и передаче между ними объектов.



Какие типы можно хранить?

- Во всех коллекциях Objective-C можно хранить только объектные типы.
- Для скалярных типов, например BOOL, int, double придется воспользоваться контейнером NSNumber.
- Не получится в коллекции сохранить nil. придется воспользоваться специальным объектом NSNull.



Коллекции и управление памятью

- Удерживают объект при добавлении в коллекцию (увеличивают счетчик ссылок на него).



Lightweight generics

```
NSArray<NSString *> *animals = @[@"Cat", @"Dog", @"Bird"]; // (1)

NSDictionary<NSString *, NSNumber *> *legsAndPaws = @{
    @"Cat": @(4),
    @"Dog": @(4),
    @"Bird": @(2)
}; // (2)
```



Lightweight generics

- Суть дженериков - проверка типа объектов на этапе компиляции. Но ведь Objective-C язык динамический и во время выполнения что угодно можно положить куда угодно.
- Как с этим справляются дженерики? Плохо. Даже предупреждения компилятора можно получить не всегда.
- Такую конструкцию можно использовать для повышения читаемости кода, главное помнить, что использование дженериков не может избавить от проверок времени выполнения.



Lightweight generics

```
NSArray<NSString *> *fakeAnimals = @[@"Cat", @(20), @"Dog"];
```



NSArray



Массивы (NSArray)

- NSArray - это основной класс для представления массива в Objective-C, упорядоченный. Существует также его изменяемый вариант NSMutableArray.



Оценки производительности операций

- Вставка в конец массива - $O(1)$
- Вставка в произвольное место - $O(N)$
- Удаление элемента из произвольного места - $O(N)$
- Доступ по индексу - $O(1)$
- Бинарный поиск (на отсортированном массиве) - $O(\log N)$
- Сортировка - $O(n^* \log N)$



Создание

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                           @"Opel", @"Volkswagen", @"Audi"];
NSArray *ukMakes = [NSArray arrayWithObjects:@"Aston Martin",
                    @"Lotus", @"Jaguar", @"Bentley", nil];
```

```
NSLog(@"First german make: %@", germanMakes[0]);
NSLog(@"First U.K. make: %@", [ukMakes objectAtIndex:0]);
```



Обход

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche", @"Opel", @"Volkswagen", @"Audi"];
// With fast-enumeration
for (NSString *item in germanMakes) {
    NSLog(@"%@", item);
}
// With a traditional for loop
for (int i=0; i<[germanMakes count]; i++) {
    NSLog(@"%@", i, germanMakes[i]);
}
// With block
[germanMakes enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
    NSLog(@"%@", idx, obj);
}];
```



Сравнение

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                           @"Opel", @"Volkswagen", @"Audi"];
NSArray *sameGermanMakes = [NSArray arrayWithObjects:@"Mercedes-Benz",
                           @"BMW", @"Porsche", @"Opel",
                           @"Volkswagen", @"Audi", nil];

if ([germanMakes isEqualToString:sameGermanMakes]) {
    NSLog(@"Oh good, literal arrays are the same as NSArrays");
}
```



Проверка элемента на вхождение

- Вернет лишь первое вхождение.
- NSSet намного эффективней для решения задачи «проверка на вхождение».

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                           @"Opel", @"Volkswagen", @"Audi"];
// BOOL checking
if ([germanMakes containsObject:@"BMW"]) {
    NSLog(@"BMW is a German auto maker");
}
// Index checking
NSUInteger index = [germanMakes indexOfObject:@"BMW"];
if (index == NSNotFound) {
    NSLog(@"Well that's not quite right...");
} else {
    NSLog(@"BMW is a German auto maker and is at index %ld", index);
}
```



Сортировка

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                           @"Opel", @"Volkswagen", @"Audi"];
NSArray *sortedMakes = [germanMakes sortedArrayUsingComparator:
    ^NSComparisonResult(id obj1, id obj2) {
        if ([obj1 length] < [obj2 length]) {
            return NSOrderedAscending;
        } else if ([obj1 length] > [obj2 length]) {
            return NSOrderedDescending;
        } else {
            return NSOrderedSame;
        }
    }];
 NSLog(@"%@", sortedMakes);
```



Фильтрация

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
 @"Opel", @"Volkswagen", @"Audi"];
NSPredicate *beforeL = [NSPredicate predicateWithBlock: ^BOOL(id evaluatedObject,
 NSDictionary *bindings) {
 NSComparisonResult result = [@"L" compare:evaluatedObject];
 if (result == NSOrderedDescending) {
     return YES;
 } else {
     return NO;
 }
}];
NSArray *makesBeforeL = [germanMakes filteredArrayUsingPredicate:beforeL];
NSLog(@"%@", makesBeforeL); // BMW, Audi
```



Разделение на части

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                           @"Opel", @"Volkswagen", @"Audi"];  
  
NSArray *lastTwo = [germanMakes subarrayWithRange:NSMakeRange(4, 2)];  
 NSLog(@"%@", lastTwo); // Volkswagen, Audi
```



Объединение

```
NSArray *germanMakes = @[@"Mercedes-Benz", @"BMW", @"Porsche",
                           @"Opel", @"Volkswagen", @"Audi"];
NSArray *ukMakes = @[@"Aston Martin", @"Lotus", @"Jaguar", @"Bentley"];

NSArray *allMakes = [germanMakes arrayByAddingObjectsFromArray:ukMakes];
NSLog(@"%@", allMakes);
```



Reverse enumerator

```
for (id object in [array reverseObjectEnumerator]) {  
    //do something with object  
}
```



Бинарный поиск

- Возможно найти элемент быстрее, чем за $O(n)$, в отсортированном массиве, если использовать бинарный поиск
- С параметром `NSBinarySearchingFirstEqual`.
- Вызывать данный метод можно на частично отсортированном массиве, для этого в `range` нужно указать отсортированную часть.

```
- (NSUInteger)indexForObject:(ObjectType)obj  
    inSortedRange:(NSRange)r  
    options:(NSBinarySearchingOptions)opts  
    usingComparator:(NSComparator)cmp;
```



Изменяющиеся массивы (NSMutableArray)

- В изменяющиеся массивы можно динамически добавлять и удалять объекты.



Создание

```
NSMutableArray *brokenCars = [NSMutableArray arrayWithObjects:  
    @"Audi A6", @"BMW Z3",  
    @"Audi Quattro", @"Audi TT", nil];
```

- Литерального синтаксиса нет.
- Существует вариант вызвать mutableCopy для созданного через
литеральный конструктор неизменяемого массива.



Добавление и удаление объектов

- Элементы массива можно добавить в конец.
- Удалить последний.
- Вставить в произвольное место.
- Удалить из произвольного места.
- Заменить объект по индексу.



Добавление и удаление объектов

```
NSMutableArray *brokenCars = [NSMutableArray arrayWithObjects:  
    @"Audi A6", @"BMW Z3",  
    @"Audi Quattro", @"Audi TT", nil];  
[brokenCars addObject:@"BMW F25"];  
NSLog(@"%@", brokenCars); // BMW F25 added to end  
[brokenCars removeLastObject];  
NSLog(@"%@", brokenCars); // BMW F25 removed from end  
  
// Add BMW F25 to front  
[brokenCars insertObject:@"BMW F25" atIndex:0];  
// Remove BMW F25 from front  
[brokenCars removeObjectAtIndex:0];  
// Remove Audi Quattro  
[brokenCars removeObject:@"Audi Quattro"];  
  
// Change second item to Audi Q5  
[brokenCars replaceObjectAtIndex:1 withObject:@"Audi Q5"];
```



Сортировка

- Сортировать можно и с использованием `sortUsingComparator`, который работает также как и для неизменяемых массивов.



Соглашение по обходу массивов

- Во время обхода изменяемых массивов (любым способом) нельзя добавлять или удалять из него объекты.



NSDictionary



Словари (NSDictionary)

- NSDictionary - класс для представления словаря в Objective-C, неупорядоченный. Существует также его изменяемый вариант NSMutableDictionary.
- Ключи словаря должны удовлетворять протоколу NSCopying.



Оценки производительности операций

- Вставка, в лучшем случае - $O(1)$, в худшем $O(N)$
- Поиск по ключу - $O(1)$
- По значению - $O(N)$
- Удаление элемента - $O(1)$



Создание

```
// Literal syntax
NSDictionary *inventory = @{
    @"Mercedes-Benz SLK250" : [NSNumber numberWithInt:13],
    @"Mercedes-Benz E350" : [NSNumber numberWithInt:22],
    @"BMW M3 Coupe" : [NSNumber numberWithInt:19],
    @"BMW X6" : [NSNumber numberWithInt:16],
};

// Values and keys as arguments
inventory = [NSDictionary dictionaryWithObjectsAndKeys:
    [NSNumber numberWithInt:13], @"Mercedes-Benz SLK250",
    [NSNumber numberWithInt:22], @"Mercedes-Benz E350",
    [NSNumber numberWithInt:19], @"BMW M3 Coupe",
    [NSNumber numberWithInt:16], @"BMW X6", nil];

// Values and keys as arrays
NSArray *models = @[@"Mercedes-Benz SLK250", @"Mercedes-Benz E350",
    @"BMW M3 Coupe", @"BMW X6"];
NSArray *stock = @[[NSNumber numberWithInt:13],
    [NSNumber numberWithInt:22],
    [NSNumber numberWithInt:19],
    [NSNumber numberWithInt:16]];
inventory = [NSDictionary dictionaryWithObjects:stock forKeys:models];
```



Доступ по ключу

```
NSDictionary *inventory = @{
    @"Mercedes-Benz SLK250" : [NSNumber numberWithInt:13],
    @"Mercedes-Benz E350" : [NSNumber numberWithInt:22],
    @"BMW M3 Coupe" : [NSNumber numberWithInt:19],
    @"BMW X6" : [NSNumber numberWithInt:16],
};

NSLog(@"There are %@ X6's in stock", inventory[@"BMW X6"]);
NSLog(@"There are %@ E350's in stock",
      [inventory objectForKey:@"Mercedes-Benz E350"]);
```



Обход словаря

```
// Fast enumeration.  
NSDictionary *inventory = @{  
    @"Mercedes-Benz SLK250" : [NSNumber numberWithInt:13],  
    @"Mercedes-Benz E350" : [NSNumber numberWithInt:22],  
    @"BMW M3 Coupe" : [NSNumber numberWithInt:19],  
    @"BMW X6" : [NSNumber numberWithInt:16],  
};  
NSLog(@"We currently have %ld models available", [inventory count]);  
for (id key in inventory) {  
    NSLog(@"There are %@ %@", inventory[key], key);  
}  
  
// Keys and values  
NSLog(@"Models: %@", [inventory allKeys]);  
NSLog(@"Stock: %@", [inventory allValues]);  
  
// Blocks  
[inventory enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {  
    NSLog(@"There are %@ %@", obj, key);  
}];
```



Сравнение словарей

```
NSDictionary *warehouseInventory = @{
    @"Mercedes-Benz SLK250" : [NSNumber numberWithInt:13],
    @"Mercedes-Benz E350" : [NSNumber numberWithInt:22],
    @"BMW M3 Coupe" : [NSNumber numberWithInt:19],
    @"BMW X6" : [NSNumber numberWithInt:16],
};

NSDictionary *showroomInventory = @{
    @"Mercedes-Benz SLK250" : [NSNumber numberWithInt:13],
    @"Mercedes-Benz E350" : [NSNumber numberWithInt:22],
    @"BMW M3 Coupe" : [NSNumber numberWithInt:19],
    @"BMW X6" : [NSNumber numberWithInt:16],
};

if ([warehouseInventory isEqualToString:showroomInventory]) {
    NSLog(@"Why are we storing so many cars in our showroom?");
}
```



Сортировка ключей

- Словарь нельзя отсортировать в новый словарь.
- Можно отсортировать ключи словаря в массив используя `keysSortedByValueUsingComparator`.

```
NSDictionary *prices = @{
    @"Mercedes-Benz SLK250" : [NSDecimalNumber decimalNumberWithString:@"42900.00"],
    @"Mercedes-Benz E350" : [NSDecimalNumber decimalNumberWithString:@"51000.00"],
    @"BMW M3 Coupe" : [NSDecimalNumber decimalNumberWithString:@"62000.00"],
    @"BMW X6" : [NSDecimalNumber decimalNumberWithString:@"55015.00"]
};
NSArray *sortedKeys = [prices keysSortedByValueUsingComparator:
    ^NSComparisonResult(id obj1, id obj2) {
        return [obj2 compare:obj1];
}];
```



Фильтрация ключей

- Ключи, прошедшие тест в методе `keysOfEntriesPassingTest`, будут записаны в множество.

```
NSDictionary *prices = @{
    @"Mercedes-Benz SLK250" : [NSDecimalNumber decimalNumberWithString:@"42900.00"],
    @"Mercedes-Benz E350" : [NSDecimalNumber decimalNumberWithString:@"51000.00"],
    @"BMW M3 Coupe" : [NSDecimalNumber decimalNumberWithString:@"62000.00"],
    @"BMW X6" : [NSDecimalNumber decimalNumberWithString:@"55015.00"]
};

NSDecimalNumber *maximumPrice = [NSDecimalNumber decimalNumberWithString:@"50000.00"];
NSSet *under50k = [prices keysOfEntriesPassingTest: ^BOOL(id key, id obj, BOOL *stop) {
    if ([obj compare:maximumPrice] == NSOrderedAscending) {
        return YES;
    } else {
        return NO;
    }
}];
```



Изменяемые словари (NSMutableDictionary)

- В изменяемые словари можно добавлять пары ключ-значения и можно их удалять, производительность аналогична неизменяемому словарю.



Создание

- Литерального синтаксиса нет.
- Существует вариант вызвать mutableCopy для созданного через литеральный конструктор неизменяемого словаря.

```
NSMutableDictionary *jobs = [NSMutableDictionary  
    dictionaryWithDictionary:@{  
        @"Audi TT" : @"John",  
        @"Audi Quattro (Black)" : @"Mary",  
        @"Audi Quattro (Silver)" : @"Bill",  
        @"Audi A7" : @"Bill"  
    }];
```



Добавление и удаление объектов

```
NSMutableDictionary *jobs = [NSMutableDictionary
    dictionaryWithDictionary:@{
        @"Audi TT" : @"John",
        @"Audi Quattro (Black)" : @"Mary",
        @"Audi Quattro (Silver)" : @"Bill",
        @"Audi A7" : @"Bill"
    }];
// Transfer an existing job to Mary
[jobs setObject:@"Mary" forKey:@"Audi TT"];
// Finish a job
[jobs removeObjectForKey:@"Audi A7"];
// Add a new job
jobs[@"Audi R8 GT"] = @"Jack";
```



Объединение словарей

```
NSMutableDictionary *jobs = [NSMutableDictionary dictionaryWithDictionary:@{
    @"Audi TT" : @"John",
    @"Audi Quattro (Black)" : @"Mary",
    @"Audi Quattro (Silver)" : @"Bill",
    @"Audi A7" : @"Bill"
}];
NSDictionary *bakerStreetJobs = @{
    @"BMW 640i" : @"Dick",
    @"BMW X5" : @"Brad"
};
[jobs addEntriesFromDictionary:bakerStreetJobs];

// Create an empty mutable dictionary
NSMutableDictionary *jobs = [NSMutableDictionary dictionaryWithDictionary];

// Populate it with initial key-value pairs
[jobs addEntriesFromDictionary:@{
    @"Audi TT" : @"John",
    @"Audi Quattro (Black)" : @"Mary",
    @"Audi Quattro (Silver)" : @"Bill",
    @"Audi A7" : @"Bill"
}];
```



Соглашение по обходу словарей

- Справедливо то же правило, что и для массивов. При обходе изменяемого словаря нельзя добавлять и удалять элементы из него.



Сортировка с дескрипторами

```
NSDictionary *car1 = @{@"make": @"Volkswagen", @"model": @"Golf", @"price": [NSDecimalNumber decimalNumberWithString:@"18750.00"]};  
NSDictionary *car2 = @{@"make": @"Volkswagen", @"model": @"Eos", @"price": [NSDecimalNumber decimalNumberWithString:@"35820.00"]};  
NSDictionary *car3 = @{@"make": @"Volkswagen", @"model": @"Jetta A5", @"price": [NSDecimalNumber decimalNumberWithString:@"16675.00"]};  
NSDictionary *car4 = @{@"make": @"Volkswagen", @"model": @"Jetta A4", @"price": [NSDecimalNumber decimalNumberWithString:@"16675.00"]};  
NSMutableArray *cars = [NSMutableArray arrayWithObjects: car1, car2, car3, car4, nil];  
NSSortDescriptor *priceDescriptor = [NSSortDescriptor sortDescriptorWithKey:@"price" ascending:YES selector:@selector(compare)];  
NSSortDescriptor *modelDescriptor = [NSSortDescriptor sortDescriptorWithKey:@"model" ascending:YES selector:@selector(caseInsensitiveCompare)];  
  
NSArray *descriptors = @[priceDescriptor, modelDescriptor];  
[cars sortUsingDescriptors:descriptors];  
NSLog(@"%@", cars); // car4, car3, car1, car2
```



NSSet

Devteam



Множества (NSSet)

- NSSet - основной класс для представления множества, коллекция неупорядоченная. Существует изменяемый вариант NSMutableSet.



Оценки производительности операций

- Вставка, в лучшем случае - $O(1)$, в худшем $O(N)$
- Поиск по ключу - $O(1)$
- По значению - $O(N)$
- Удаление элемента - $O(1)$



Создание

- Множество можно создать с помощью метода `setWithObjects:` , в конце списка объектов должен обязательно идти `nil`.
- Удобно создавать множество из массива с помощью метода `setWithArray`.

```
NSSet *americanMakes = [NSSet setWithObjects:@"Chrysler", @"Ford", @"General Motors", nil];  
  
NSArray *japaneseMakes = @[@"Honda", @"Mazda", @"Mitsubishi", @"Honda"];  
NSSet *uniqueMakes = [NSSet setWithArray:japaneseMakes];
```



Unused



Обход

```
// Fast enumeration
NSSet *models = [NSSet setWithObjects:@"Civic", @"Accord", @"Odyssey", @"Pilot", @"Fit", nil];
NSLog(@"The set has %li elements", [models count]);
for (id item in models) {
    NSLog(@"%@", item);
}

// Blocks
[models enumerateObjectsUsingBlock:^(id obj, BOOL *stop) {
    NSLog(@"Current item: %@", obj);
    if ([obj isEqualToString:@"Fit"]) {
        NSLog(@"I was looking for a Honda Fit, and I found it!");
        *stop = YES; // Stop enumerating items
    }
}];
```



Сравнение

```
NSSet *japaneseMakes = [NSSet setWithObjects:@"Honda", @"Nissan", @"Mitsubishi", @"Toyota", nil];
NSSet *johnsFavoriteMakes = [NSSet setWithObjects:@"Honda", nil];
NSSet *marysFavoriteMakes = [NSSet setWithObjects:@"Toyota", @"Alfa Romeo", nil];
if ([johnsFavoriteMakes isEqualToSet:japaneseMakes]) {
    NSLog(@"John likes all the Japanese auto makers and no others");
}
if ([johnsFavoriteMakes intersectsSet:japaneseMakes]) {
    // You'll see this message
    NSLog(@"John likes at least one Japanese auto maker");
}
if ([johnsFavoriteMakes isSubsetOfSet:japaneseMakes]) {
    // And this one, too
    NSLog(@"All of the auto makers that John likes are Japanese");
}
if ([marysFavoriteMakes isSubsetOfSet:japaneseMakes]) {
    NSLog(@"All of the auto makers that Mary likes are Japanese");
}
```



Проверка элемента на вхождение

```
NSSet *selectedMakes = [NSSet setWithObjects:@"Maserati",
                                         @"Porsche", nil];
// BOOL checking
if ([selectedMakes containsObject:@"Maserati"]) {
    NSLog(@"The user seems to like expensive cars");
}
// nil checking
NSString *result = [selectedMakes member:@"Maserati"];
if (result != nil) {
    NSLog(@"%@", result);
}
```



Фильтрация

```
NSSet *toyotaModels = [NSSet setWithObjects:@"Corolla", @"Sienna",
                        @"Camry", @"Prius",
                        @"Highlander", @"Sequoia", nil];
NSSet *cModels = [toyotaModels objectsPassingTest:^BOOL(id obj, BOOL
*stop) {
    if ([obj hasPrefix:@"C"])
        return YES;
    } else {
        return NO;
    }
];
NSLog(@"%@", cModels); // Corolla, Camry
```



Объединение

```
NSSet *affordableMakes = [NSSet setWithObjects:@"Ford", @"Honda",
                           @"Nissan", @"Toyota", nil];
NSSet *fancyMakes = [NSSet setWithObjects:@"Ferrari", @"Maserati",
                     @"Porsche", nil];
NSSet *allMakes = [affordableMakes setByAddingObjectsFromSet:fancyMakes];
NSLog(@"%@", allMakes);
```



Изменяемые множества (NSMutableSet)

- В изменяющиеся множества можно динамически добавлять и удалять объекты.



Создание

- Изменяемую коллекцию можно создавать, явно указав объекты в конструкторе.
- Можно создать пустое множество, выбрав его исходный размер в методе `setWithCapacity`. Задание `capacity = 5` не означает, что в множество нельзя добавить более 5 элементов, указание емкости необходимо лишь для внутренней оптимизации структуры.



Создание

```
NSMutableSet *brokenCars = [NSMutableSet setWithObjects:  
                           @{@"Honda Civic", @"Nissan Versa", nil}];  
NSMutableSet *repairedCars = [NSMutableSet setWithCapacity:5];
```



Добавление и удаление объектов

```
NSMutableSet *brokenCars = [NSMutableSet setWithObjects:@"Honda Civic", @"Nissan  
Versa", nil];  
[brokenCars removeObject:@"Honda Civic"];  
[brokenCars addObject:@"Honda Civic"];  
[brokenCars removeAllObjects];
```



Фильтрация с предикатом

- Не существует мутабельной версии метода `objectsPassingTest:`, однако того же самого результата можно добиться с использованием метода `filteringUsingPredicate:`.

```
NSMutableSet *toyotaModels = [NSMutableSet setWithObjects:@"Corolla", @"Sienna",
    @"Camry", @"Prius", @"Highlander", @"Sequoia", nil];
NSPredicate *startsWithC = [NSPredicate predicateWithBlock:^BOOL(id
    evaluatedObject, NSDictionary *bindings) {
    if ([evaluatedObject hasPrefix:@"C"]) {
        return YES;
    } else {
        return NO;
    }
}];
[toyotaModels filterUsingPredicate:startsWithC];
NSLog(@"%@", toyotaModels); // Corolla, Camry
```



Операции из теории множеств

```
NSSet *japaneseMakes = [NSSet setWithObjects:@"Honda", @"Nissan",
                           @"Mitsubishi", @"Toyota", nil];
NSSet *johnsFavoriteMakes = [NSSet setWithObjects:@"Honda", nil];
NSSet *marysFavoriteMakes = [NSSet setWithObjects:@"Toyota",
                             @"Alfa Romeo", nil];

NSMutableSet *result = [NSMutableSet setWithCapacity:5];
// Объединение
[result setSet:johnsFavoriteMakes];
[result unionSet:marysFavoriteMakes];
NSLog(@"Either John's or Mary's favorites: %@", result);
// Пересечение
[result setSet:johnsFavoriteMakes];
[result intersectSet:japaneseMakes];
NSLog(@"John's favorite Japanese makes: %@", result);
//Разность
[result setSet:japaneseMakes];
[result minusSet:johnsFavoriteMakes];
NSLog(@"Japanese makes that are not John's favorites: %@",
      result);
```



Домашнее задание

- Освежить/прочитать про указатели в С.
- Прочитать про перечисления (enum).
- Посмотреть на сайте Apple документацию на NSString, NSNumber, NSDecimalNumber.



Домашнее задание

- Рассмотреть коллекции `NSPointerArray`, `NSHashTable`.
- Описать несколько примеров их использования, написать код.
- Разобраться чем данные коллекции отличаются от `NSArray`, `NSSet`, `NSDictionary`.
- Рассмотреть дополнительные виды множеств `NSCountedSet`, `NSOrderedSet`.
- Изучить `NSPredicate`.



Let's Code

Devteam



Вопросы?



Полезные ссылки

- <https://developer.apple.com/documentation/foundation/nsarray/1412722-indexofobject>
- <https://developer.apple.com/documentation/foundation/collections?language=objc>
- https://developer.apple.com/documentation/foundation/numbers_data_and_basic_values?language=objc



Спасибо ❤