Протоколы, расширения, блоки, диспетчиризация в Objective-C

Devteam

Макаровская Вероника Михайловна @MVeronika

Devteam

Что мы узнаем сегодня?

- Протоколы
- Расширения
- Блоки
- Диспетчиризация

Devteam

Протоколы

Devteam

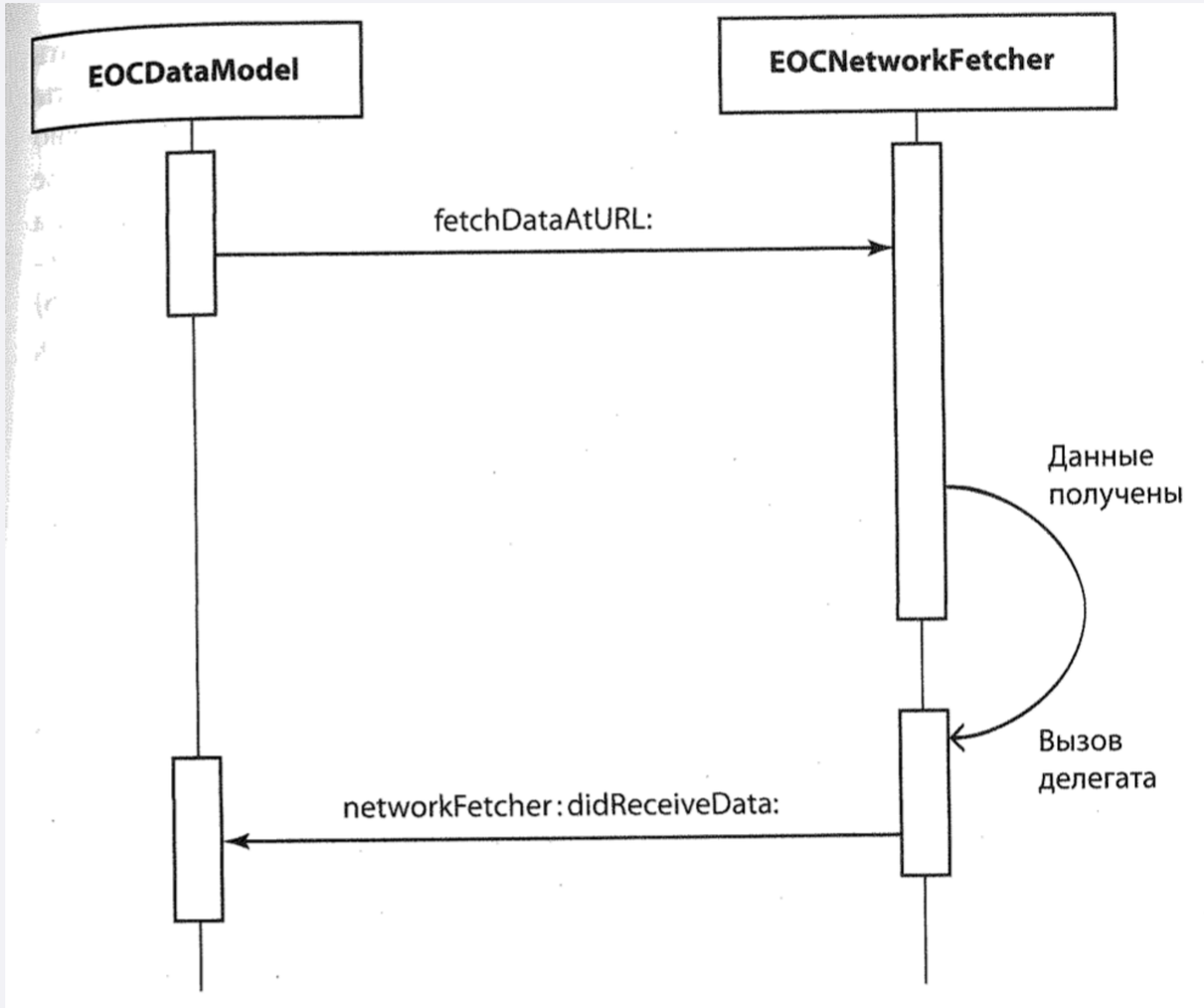
Протоколы

- Аналог интерфейса в Java.
- В Objective-C нет множественного наследования.
- Используются для реализации паттерна «Делегат»
- Определяют набор методов, которые могут или должны реализовываться объектом.

Протоколы

- Используем паттерн Делегат для формирования интрефейса к объектам, которые должны сообщать другим объектам об актуальных событиях
- Используем паттерн Делегат, когда объект должен получать данные от другого объекта

Протоколы



Протоколы

```
@protocol EOCNetworkFetcherDelegate
@optional
- (void)networkFetcher:(EOCNetworkFetcher*)fetcher
        didReceiveData:(NSData*)data;
- (void)networkFetcher:(EOCNetworkFetcher*)fetcher
        didFailWithError:(NSError*)error;
- (void)networkFetcher:(EOCNetworkFetcher*)fetcher
        didUpdateProgressTo:(float)progress;
@end
```

Протоколы

```
if ([_delegate respondsToSelector:
            @selector(someClassDidSomething:)])
{
    [_delegate someClassDidSomething];
}
```

Devteam

Анонимные объекты (anonymous objects)

- Скрываем подробности реализации API
- id <EOCDelegate> delegate - класс делегата может быть любым
- Он даже не обязан быть унаследован от NSObject
- Для класса, у которого объявлено свойство delegate, класс этого делегата неизвестен.
- Мы можем вызывать метод class и узнать класс delegate. (Костыль!)

Анонимные объекты (anonymous objects)

```
- (void)setObject:(ObjectType)anObject
           forKey:(id<NSCopying>)aKey;
```

Devteam

Анонимные объекты (anonymous objects)

- Протоколы используются для реализации анонимности Используем анонимные объекты, если тип(имя класса) требуется скрыть.
- Используем анонимные объекты, если важен факт реакции объекта на определённые методы

Let’s Code

Devteam

Расширения

Devteam

```
#import <Foundation/Foundation.h>

@interface EOCPerson : NSObject
@property (nonatomic, copy, readonly) NSString *firstName;
@property (nonatomic, copy, readonly) NSString *lastName;
@property (nonatomic, strong, readonly) NSArray *friends;

- (id)initWithFirstName:(NSString*)firstName
            andLastName:(NSString*)lastName;
@end

@interface EOCPerson (Friendship)
- (void)addFriend:(EOCPerson*)person;
- (void)removeFriend:(EOCPerson*)person;
- (BOOL)isFriendsWith:(EOCPerson*)person;
@end

@interface EOCPerson (Work)
- (void)performDaysWork;
- (void)takeVacationFromWork;
@end
```

Devteam

Расширения(Категории)

- Механизм добавления методов в класс без применения субклассирования
- Для разделения класса на разделы
- Категории можно создавать как в одном файле, так и в нескольких
- Использование категорий - полезный способ разделения кода на функциональные области
- Имя категории полезно для уточнения функциональности класса

Расширения(Категории)

- Создавайте категорию Private для сокрытия реализации методов
- Категории часто используются для расширения функциональности классов, чей код вам недоступен
- Также стоит помнить о том, что методы, добавленные в класс через категорию, становятся доступны для всего приложения.

Расширения(Категории)

- Всегда добавляйте префикс в имена категорий и их методов, если класс вам не принадлежит. Так как одна из особенностей категорий в Objective-C — метод, определенный в категории, полностью перекрывает метод базового класса.

Расширения(Категории)

- Свойства лучше не объявлять в категориях.

Devteam

Категория продолжения класса

```
@interface EOCPerson ()
// Методы
@end
```

Категория продолжения класса

- Используются для добавления переменных экземпляров в класс.
- Можно объявлять прототипы методов в категориях продолжения классов.
- Используются для объявления протоколов, факт реализации которых вашим классом вы не хотите разглашать.
- В таких категориях можно изменять уровень чтения на уровень чтения/записи

Devteam

Let’s Code

Devteam

Диспетчиризация

Devteam

Диспетчиризация

- Передача сообщений
- Селектор - имя сообщения
- Могут принимать параметры и возвращать значения

Динамическая привязка

- Механизм автивизации методов в Objective-C при передаче сообщений объекту.
- Решение, какой метод будет вызван для заданного сообщения, принимается во время выполнения.
- Выбор также может изменяться в процессе выполнения приложения.

Сообщения

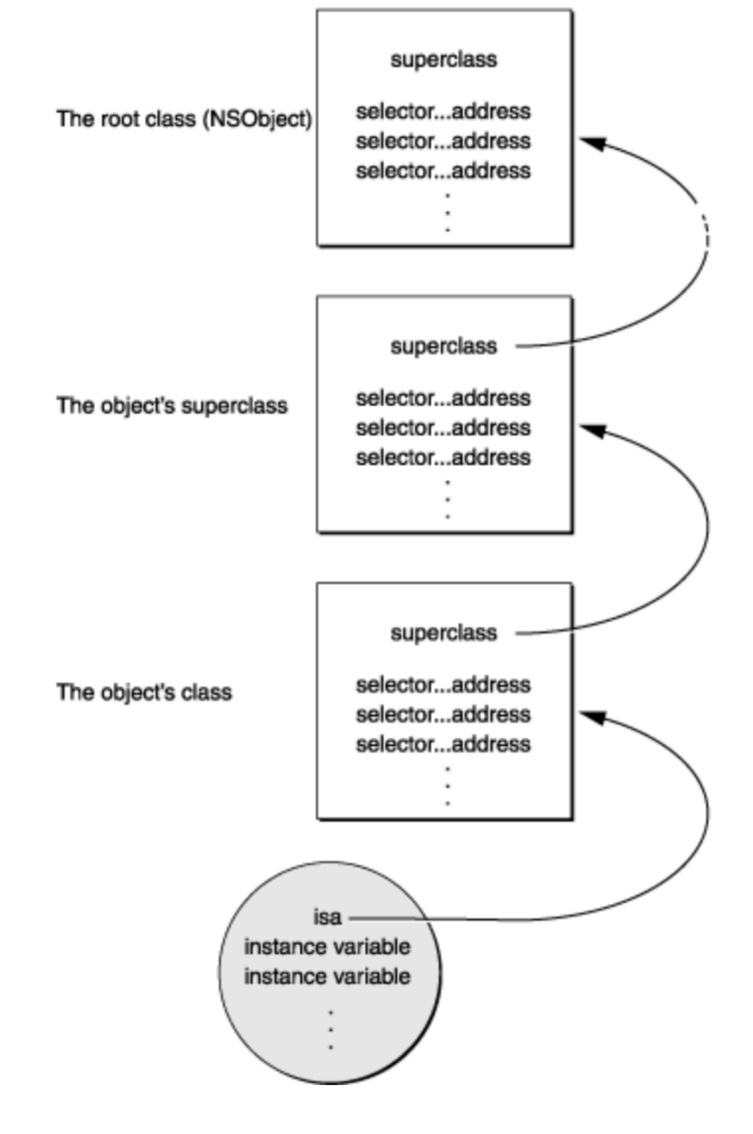
- Получатель
- Селектор
- Параметры
- Селектор + параметры = сообщение

Devteam

Функция objc_msgSend

- objc_msgSend(receiver, selector, arg1, arg2, …)
- Компилятор: [receiver message] -> objc_msgSend(receiver, message)
- Просматриваем список методов у receiver, потом у его супер-класса(Таблица диспетчеризации класса). Если метод не найден, то выполняется механизм перенаправления сообщений.
- objc_msgSend кэширует результаты

Функция objc_msgSend



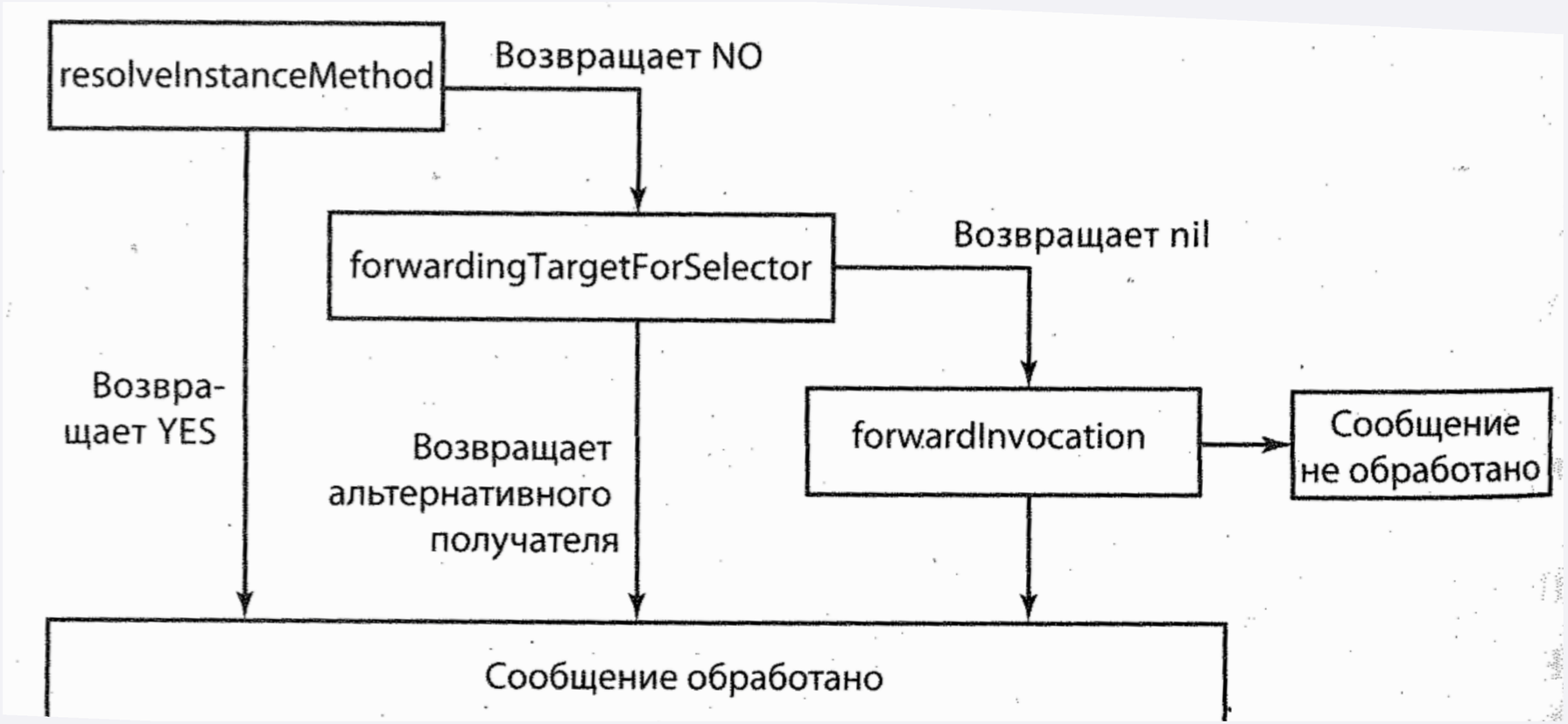
Devteam

Перенаправление сообщений

- Процесс, который позволяет разработчику указать, как должно обрабатываться незаконное сообщение.
- Вызывается один из специальных методов: `resolveInstanceMethod` или `resolveClassMethod`

Devteam

Перенаправление сообщений



Let’s Code

Devteam

Блоки

Devteam

Блоки

- Представляют механизм передачи фрагмента кода, словно те являются объектами для выполнения в другом контексте
- Могут использовать все, что угодно, из области, в которой они определены
- Блок можно присвоить переменной

Блоки

```
^{
    NSLog(@"This is a block");
}
```

```
void (^simpleBlock)(void);
simpleBlock = ^{
    NSLog(@"This is a block");
};
```

Devteam

Блоки

```
double (^multiplyTwoValues)(double, double) =
                          ^(double firstValue, double secondValue) {
                              return firstValue * secondValue;
                          };

double result = multiplyTwoValues(2,4);

NSLog(@"The result is %f", result);
```

Capture

- Блоки захватывают область видимости. То есть переменные, определенные в той же области, что и блок, доступны внутри блока
- По умолчанию, переменные захваченные блоком не могут модифицироваться.
- __block
- При захвате переменной объектного типа блок неявно удерживает её.
- Захватывается только значение.

Capture

```
- (void)testMethod {
    int anInteger = 42;

    void (^testBlock)(void) = ^{
        NSLog(@"Integer is: %i", anInteger);
    };

    testBlock();
}
```

```
int anInteger = 42;

void (^testBlock)(void) = ^{
    NSLog(@"Integer is: %i", anInteger);
};

anInteger = 84;

testBlock();
```

Capture

```
__block int anInteger = 42;

void (^testBlock)(void) = ^{
    NSLog(@"Integer is: %i", anInteger);
};

anInteger = 84;

testBlock();
```

```
__block int anInteger = 42;

void (^testBlock)(void) = ^{
    NSLog(@"Integer is: %i", anInteger);
    anInteger = 100;
};

testBlock();
NSLog(@"Value of original variable is now: %i", anInteger);
```

Devteam

Inline blocks

```
NSArray* arr2 = [arr sortedArrayUsingComparator: ^(id obj1, id obj2) {
    NSString* s1 = obj1;
    NSString* s2 = obj2;
    NSString* string1end = [s1 substringFromIndex:[s1 length] - 1];
    NSString* string2end = [s2 substringFromIndex:[s2 length] - 1];
    return [string1end compare:string2end];
}];
```

Devteam

Блоки

- Занимаемая ими область памяти выделяется на стеке.
- Но если вызвать метод copy, то копируется из стека в кучу. Можно будет использовать вне области определения.
- После копирования в кучу блок становится объектом.

Глобальные блоки

- Вся информация для их выполнения известна во время компиляции
- Создаются в глобальной памяти, а не в стеке.

```
simpleBlock = ^{
    NSLog(@"This is a block");
};
```

Тип блока

- Определяется параметрами и возвращаемым типом

```
simpleBlock = ^{
    NSLog(@"This is a block");
};
```

Devteam

Typedef

```
typedef void (^XYZSimpleBlock)(void);


@interface XYZObject : NSObject

@property (copy) XYZSimpleBlock blockProperty;

@end
```

Devteam

Циклы удержания

```
@implementation XYZBlockKeeper
- (void)configureBlock {
    self.block = ^{
        [self doSomething];    // capturing a strong reference to self
                               // creates a strong reference cycle
    };
}
...
@end
```

Циклы удержания

```
- (void)p_requestCompleted {
    if (_completionHandler) {
        _completionHandler(_downloadedData);
    }
    self.completionHandler = nil;
}
```

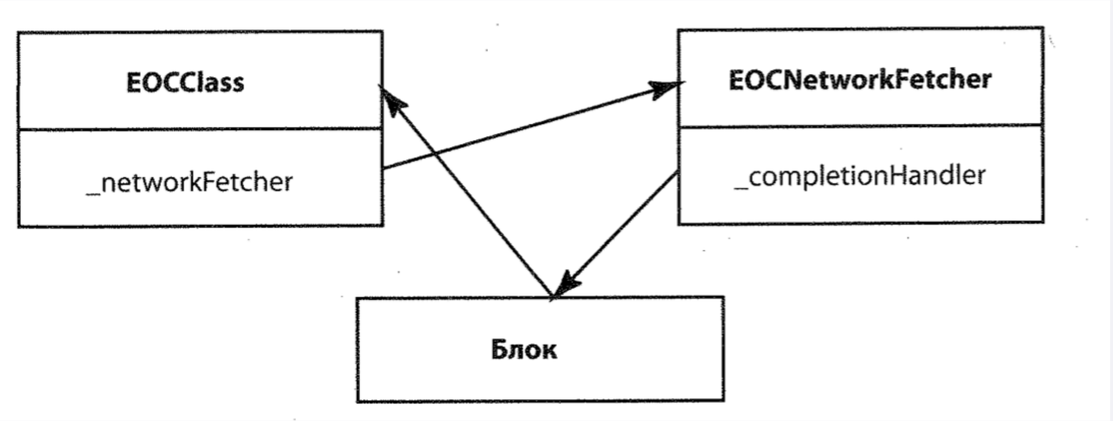
Devteam

Циклы удержания

```
@implementation EOCClass {
    EOCNetworkFetcher *_networkFetcher;
    NSData *_fetchedData;
}

- (void)downloadData {
    NSURL *url = [[NSURL alloc] initWithString:
                  @"http://www.example.com/something.dat"];
    _networkFetcher =
        [[EOCNetworkFetcher alloc] initWithURL:url];
    [_networkFetcher startWithCompletionHandler:^(NSData *data){
        NSLog(@"Request URL %@ finished", _networkFetcher.url);
        _fetchedData = data;
    }];
}

@end
```



Devteam

Let’s Code

Devteam

Домашнее задание

У вас должен быть делегат, который вернет вам массив строк
Вам нужно отсортировать массив по числу вхождения ваших первых букв имени в него

Devteam

Вопросы?

Devteam

Полезные ссылки

- https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ObjCRuntimeGuide/Introduction/Introduction.html
- https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/WorkingwithBlocks/WorkingwithBlocks.html

Devteam

Обратная связь

Devteam

Спасибо 💚

Devteam