Noveo
Software University

7. Асинхронность

Noveo University — iOS

Александр Горбунов

Сегодня

- Блоки в Objective-C
- Потоки и очереди
- `NSThread`
- `NSOperation`
- Grand Central Dispatch

Блок

- Функция
- Может иметь входные параметры
- Может возвращать значение
- Не имеет имени
- Является объектом
- Может ссылаться на свой контекст
- Может модифицировать свой контекст
- Может захватывать свой контекст

Блоки в Objective-C

- Функция
- Может иметь входные параметры
- Может возвращать значение
- Не имеет имени

```
^ (int a, int b) {
    return a + b;
}
```

Блоки в Objective-C

- Функция
- Является объектом

```
typedef int(^blockType)(int, int);
```

```
@property (copy) blockType myBlock;
```

```
blockType block = ^ (int a, int b) {
    return a + b;
}
self.myBlock = block;
int resultA = block(3, 5);
int resultB = self.myBlock(3, 5);
```

Блоки в Objective-C

- Функция
- Может ссылаться на свой контекст

```
int b = 5;

^ (int a) {
    return a + b;
}
```

Блоки в Objective-C

- Функция
- Может модифицировать свой контекст

```
__block int c;

^ (int a, int b) {
    c = a + b;
}
```

Блоки в Objective-C

- Функция
- Может захватывать свой контекст

```
NSArray *a = @[@0, @1, @2, @3, @4, @5, @6, @7];
int b = 2;

^ (int c) {
    int index = a.count + c - b;
    Object *obj = a[index];
    ...
}
```

Блоки в Objective-C

Передача объекта-блока в метод в качестве аргумента:

```
NSArray *array = @[@0, @1, @2, @3, @4, @5, @6, @7];

NSUInteger maxElements = 5;

[array enumerateObjectsUsingBlock:
    ^(id obj, NSUInteger idx, BOOL *stop) {
        NSLog(@"Element %d : %@", idx, obj);
        stop = idx >= maxElements;
    }];
```

Блоки в Objective-C

Получение объекта-блока в метод в качестве аргумента:

```
- (void)enumerateObjectsUsingBlock:(void(^)(id obj, NSUInteger idx, BOOL *stop))block
{
    if (!block) {
        return;
    }

    BOOL stop = NO;
    for (NSUInteger i = 0; i < self.count; i++) {
        block(self[i], i, &stop);
        if (stop) {
            return;
        }
    }
}
```

Асинхронность в iOS

- Все долгие операции нужно выполнять не в главном потоке
- Все операции с UI нужно выполнять в главном потоке
- Нельзя обращаться к изменяемому объекту из разных потоков
- Результат работы не должен зависеть от порядка выполнения асинхронных операций
- Распараллеливание независимых операций может ускорить работу программы

Асинхронность в iOS

Поток

- "Бесконечные" и пошаговые вычисления
- Организация runloop
- Явное управление порядком исполнения

Очередь

- Логически обособленные задачи
- Балансировка загрузки ядер ЦП
- Организация цепочки задач на высоком уровне

NSThread

- Используется для длительных или бесконечных задач
- Явное создание потока (нужно использовать с умом)
- Создание потока — дорогая операция

NSThread

- Исполняем уже имеющийся метод в отдельном потоке:

```
NSThread *thread = [[NSThread alloc] initWithTarget:self
    selector:@selector(calculate) object:nil];

[thread start];
```

- Создаём подкласс `NSThread`, переопределяем метод `main`, у экземпляра класса вызываем `start`.

NSOperation

- Используется для логически обособленных задач
- Потоки создаются автоматически
- Поддержка отмены операций
- Поддержка графа зависимостей между операциями
- Поддержка приоритезации операций

NSOperation

Объявление и запуск двух блоковых операций:

```
NSBlockOperation *operation1 = [NSBlockOperation blockOperationWithBlock:^{
        ...
    }];

NSBlockOperation *operation2 = [NSBlockOperation blockOperationWithBlock:^{
        ...
    }];

NSOperationQueue *queue = [[NSOperationQueue alloc] init];
[queue addOperation:operation1];
[queue addOperation:operation2];
```

NSOperation

Задание приорететов и зависимостей операций:

```
NSOperation *operation1, *operation2, *operation3;
NSOperationQueue *queue;

...

operation1.queuePriority = NSOperationQueuePriorityLow;
operation2.queuePriority = NSOperationQueuePriorityHigh;

[operation3 addDependency:operation1];
[operation3 addDependency:operation2];

[queue addOperation:operation1];
[queue addOperation:operation2];
[queue addOperation:operation3];
```

NSOperation

Объявление операции из метода и кастомного объекта-операции:

```
NSOperation *operation1, *operation2;
NSOperationQueue *queue;

...

operation1 = [[NSInvocationOperation alloc] initWithTarget:self
    selector:@selector(printCount) object:nil];

operation2 = [[MyCustomOperation alloc] init];

[queue addOperation:operation1];
[queue addOperation:operation2];
```

GCD

Grand Central Dispatch

- Лучшая производительность
- Относительно низкоуровневое API на языке C
- Используется для быстрого перехода между главным и фоновым потоками
- Для простых задач — короткая запись
- Поддержка очередей, групп, таймеров, ...

GCD

Пример выполнения асинхронной операции:

```
dispatch_queue_t queue =
    dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

dispatch_async(queue, ^{
    // Блок выполняется асинхронно.
    ...
});

// Программа продолжает выполняться, не дожидаясь выполнения блока.
...
```

GCD

Пример перехода в главную очередь после выполнения асинхронной операции:

```
dispatch_queue_t queue =
    dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);

dispatch_async(queue, ^{
        // Блок выполняется асинхронно.
        ...

        dispatch_async(dispatch_get_main_queue(), ^{
                // Блок выполняется на главной очереди.
                // Можно обновить UI, используя асинхронно полученные данные.
                ...
            });
    });

// Программа продолжает выполняться, не дожидаясь выполнения блоков.
...
```

GCD очереди

```
dispatch_queue_t queue1 = dispatch_queue_create("MyQueue1", NULL);
dispatch_queue_t queue2 = dispatch_queue_create("MyQueue2", NULL);

void (^myBlock1)(void) = ^ {/*...*/};
void (^myBlock2)(void) = ^ {/*...*/};
void (^myBlock3)(void) = ^ {/*...*/};

dispatch_async(queue1, myBlock1);
dispatch_async(queue2, myBlock2);
dispatch_async(queue2, myBlock3);
```

```
QUEUE 1: |——————block 1——————|

QUEUE 2: |———block 2———| |——block 3——|

Время ————————————————————————————————————>
```

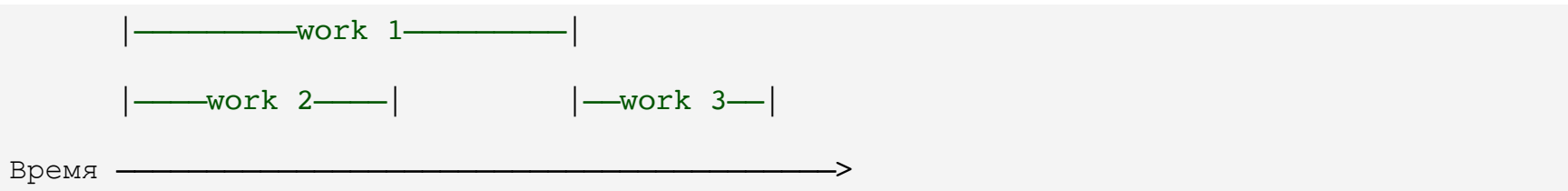
GCD группы

```
dispatch_queue_t queue =
dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
dispatch_group_t group = dispatch_group_create();

dispatch_async(queue, ^{
dispatch_group_async(group, queue, ^{
        // Do some work 1.
        ...
    });

// Do some work 2.
...

dispatch_group_wait(group, DISPATCH_TIME_FOREVER);
// Do some work 3.
...
```



Потокобезопасность

```
+ (Singleton *)sharedInstance
{
    static Singleton *_sharedInstance = nil;
    if (!_sharedInstance) {
        _sharedInstance = [[Singleton alloc] init];
    }
    return _sharedInstance;
}
```

Что произойдёт, если вызвать `sharedInstance` из двух разных потоков "одновременно", когда объект ещё не создан?

Потокобезопасность

Используем dispatch-once:

```
+ (Singleton *)sharedInstance
{
    static Singleton *_sharedInstance = nil;

    static dispatch_once_t predicate;
    dispatch_once(&predicate, ^{
            _sharedInstance = [[Singleton alloc] init];
        });

    return _sharedInstance;
}
```

Потокобезопасность

```
- (void)didReceiveSomeStringFromNetwork:(NSString *)someString
{
    self.someString = someString;
    [self showCurrentData];
}

- (void)showCurrentData
{
    self.someLabel.text = self.someString;
}
```

Что произойдёт когда мы получим новые данные по сети (в фоновом потоке)?

Потокобезопасность

При работе с UI нужно перейти в главный поток.

```
- (void)didReceiveSomeStringFromNetwork:(NSString *)someString
{
    self.someString = someString;

    dispatch_async(dispatch_get_main_queue(), ^{
            [self showCurrentData];
        });
}

- (void)showCurrentData
{
    self.someLabel.text = self.someString;
}
```

Потокобезопасность

При работе с UI нужно перейти в главный поток.

```
- (void)didReceiveSomeStringFromNetwork:(NSString *)someString
{
    self.someString = someString;

    [self performSelectorOnMainThread:@selector(showCurrentData)
        withObject:nil waitUntilDone:NO];
}

- (void)showCurrentData
{
    self.someLabel.text = self.someString;
}
```

Потокобезопасность

При работе со свойствами из разных потоков имеет смысл объявить их как `atomic`.

```
@property (atomic) NSInteger count;
```

Потокобезопасность

При работе с изменяемыми объектами из разных потоков, `atomic` не спасёт, т.к. относится только к *указателю*, но не к объекту.

```
@property (atomic) NSMutableArray *tasks;
```

Потокобезопасность

Однако мы можем вынести обращения к такому свойству в отдельную очередь.

```
@property (atomic) NSMutableArray *tasks;
@property (atomic) dispatch_queue_t tasksQueue;
```

```
- (instancetype)init
{
    self = [super init];
    if (self) {
        _tasksQueue = dispatch_queue_create("com.mydomain.myapp.tasksQueue", NULL);
    }
    return self;
}

- (void)addTask:(Task *)task
{
    dispatch_async(self.tasksQueue, ^{
            [self.tasks addObject:task];
        });
}
```