

Работа с сетью в Objective – C

Урок 6

Знакомство с работой с сетью в языке Objective – C. Изучим некоторые базовые понятия для работы с сетью. Поговорим про базовые классы NSURLConnection и другие, а также про форматы данных и передачу непосредственно на сервер, также научимся отлаживать нашу программу и отображать WebView.



План курса





Что будет на уроке сегодня

- 📌 Работа с файлами
- 📌 Класс NSData
- 📌 Обработка исключений
- 📌 Отладчик
- 📌 Что такое Objective – C Runtime
- 📌 Интроспекция
- 📌 Swizzling
- 📌 Ассоциативные ссылки
- 📌 Жизненный цикл приложения
- 📌 Жизненный цикл VC





Работа с файлами





Списки свойств

В среде Cocoa существует специальный вид файлов, известный как список свойств (property list или plist).





Списки свойств могут хранить разновидности объектов Objective-C:

NSString

NSData

NSNumber

NSArray

NSDate

NSDictionary



Класс **NSData**

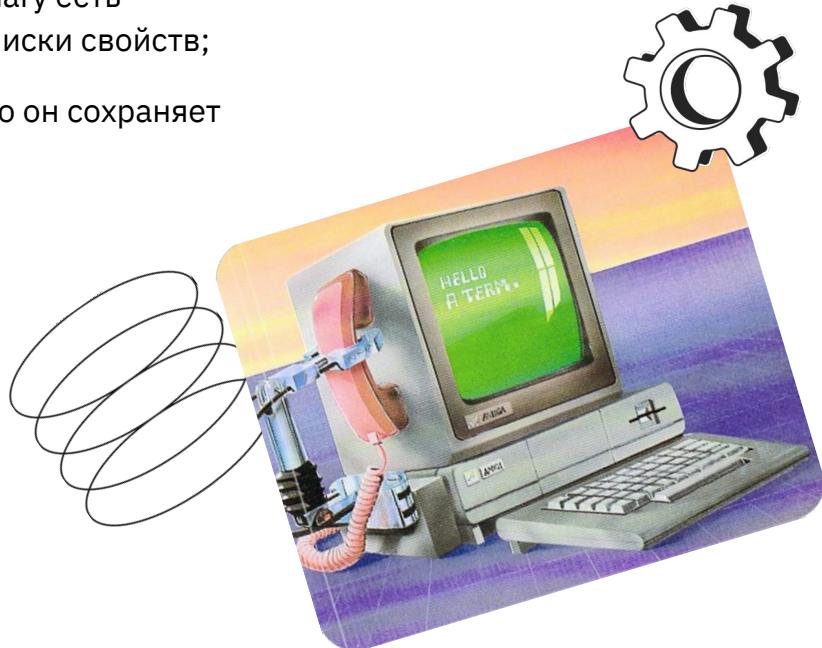
Это тип в Сocoa, который может хранить в себе данные.
Это своеобразная «обертка» для блока байтов. Можно
получить указатель на начало блока и его длину.





Запись и чтение списка свойств

- ✓ Для записи в файл у объектов NSArray и NSDictionary есть специальный метод, который преобразует их в списки свойств;
- ✓ У типов NSString и NSData тоже есть этот метод, но он сохраняет не списки свойств, а строки и блоки байтов.





Кодирование объектов

Напрямую сохранить собственный объект в память нельзя.

Для этого необходимо реализовать протокол NSCoding.

Объекты сохраняются путем кодирования в NSData.





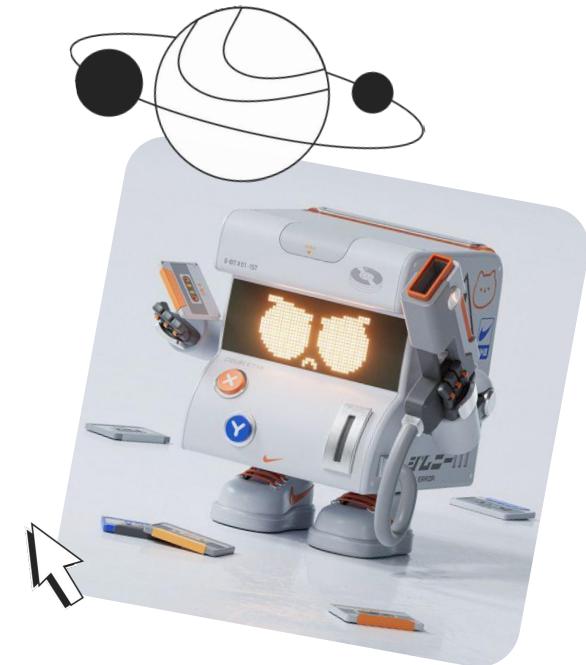
Обработка ошибок





Исключения

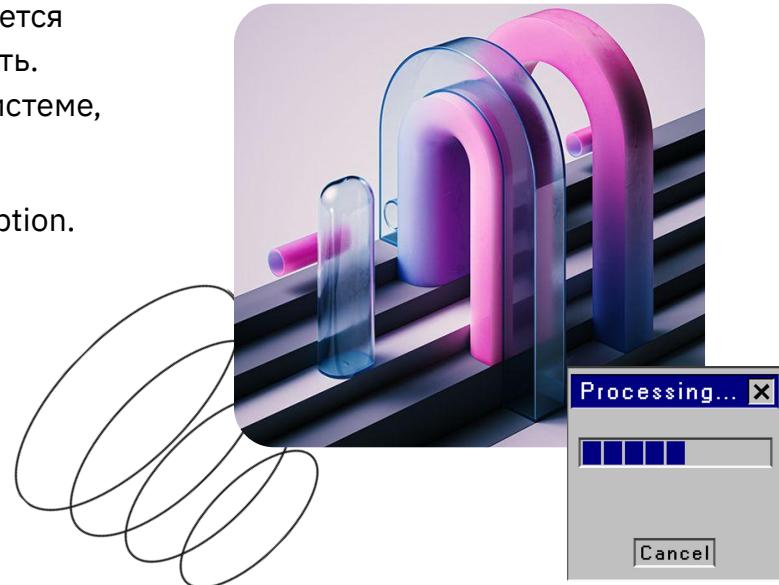
💡 Для отслеживания нежелательных событий в ходе выполнения программы существуют исключения.





Исключения

- ✓ Они возникают в случаях, когда программа оказывается в затруднительной ситуации и не знает, как поступить. Тогда создается объект исключения и передается системе, которая определяет дальнейшие действия;
- ✓ Исключения в Соса представлены классом NSException.





Ключевые слова

- ✓ **@try** – определяет блок кода, который будет проверяться на необходимость генерирования исключения;
- ✓ **@catch()** – определяет блок кода для обработки сгенерированного исключения;
- ✓ **@finally** – определяет блок кода, который выполняется вне зависимости от того, было ли сгенерировано исключение или нет;
- ✓ **@throw** – генерирует исключение.





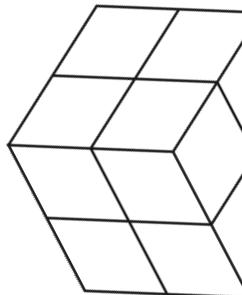
Отладка





Отладчик

- ✓ Отладчик необходим для поиска ошибок в приложении;
- ✓ Он позволяет выполнять код пошагово.





Objective-C Runtime





Objective-C Runtime

При подробном рассмотрении очевидно, что язык Objective-C – это надстройка над языком С, которая добавляет объектно-ориентированные парадигмы. Именно Runtime, библиотека времени выполнения, предоставляет тот набор функций, которые вдыхают в язык жизнь, реализуя его динамические возможности и обеспечивая функционирование ООП.





Objective-C Runtime

По сути, Objective-C - это библиотека, набор ключевых слов и различных конструкций над привычным C, которые работают в Runtime.





Функции Runtime-библиотеки

- ✓ Управление классами;
- ✓ Создание новых классов;
- ✓ Интроспекция;
- ✓ Управление объектами;
- ✓ Работа с ассоциативными ссылками.





Базовые структуры данных

Функции и структуры Runtime-библиотеки определены в нескольких заголовочных файлах: `objc.h`, `runtime.h` и `message.h`. Сначала обратимся к файлу `objc.h` и посмотрим, что представляет из себя объект с точки зрения Runtime:

```
1 typedef struct objc_class *Class;
2 typedef struct objc_object {
3     Class isa;
4 } *id;
5
```

```
1 struct objc_class {
2     Class isa;
3 };
4
```



Базовые структуры данных

Класс в Objective-C — это полноценный объект и у него тоже присутствует `isa`-указатель на «класс класса» 🤦, так называемый мета класс в терминах Objective-C. Аналогично, С-структуры определены и для других сущностей языка:

```
1 typedef struct objc_selector *SEL;
2 typedef struct objc_method *Method;
3 typedef struct objc_ivar *Ivar;
4 typedef struct objc_category *Category;
5 typedef struct objc_property *objc_property_t;
6
```



Интроспекция объекта

```
1 @implementation COBaseObject
2
3 - (NSString *)description {
4     NSMutableDictionary *propertyValues = [NSMutableDictionary dictionary];
5     unsigned int propertyCount;
6     objc_property_t *properties = class_copyPropertyList([self class], &propertyCount);
7     for (unsigned int i = 0; i < propertyCount; i++) {
8         char const *propertyName = property_getName(properties[i]);
9         const char *attr = property_getAttributes(properties[i]);
10        if (attr[1] == '@') {
11            NSString *selector = [NSString stringWithCString:propertyName encoding:NSUTF8StringEncoding];
12            SEL sel = sel_registerName([selector UTF8String]);
13            NSObject * propertyValue = objc_msgSend(self, sel);
14            propertyValues[selector] = propertyValue.description;
15        }
16    }
17    free(properties);
18    return [NSString stringWithFormat:@"%@: %@", self.class, propertyValues];
19 }
20
```



Messanges

```
1 // Вызов метода
2 [array insertObject:foo atIndex:1];
3 // Соответствующий ему вызов Runtime-функции
4 objc_msgSend(array, @selector(insertObjectAtIndex:), foo, 1);
5
```

```
1 + (BOOL)resolveInstanceMethod:(SEL)aSelector {
2     if (aSelector == @selector(myDynamicMethod)) {
3         class_addMethod(self, aSelector, (IMP)myDynamicIMP, "v@:");
4         return YES;
5     }
6     return [super resolveInstanceMethod:aSelector];
7 }
8
```



Method Swizzling

```
1 @implementation NSMutableArray (CO)
2
3 + (void)load {
4     Method addObject = class_getInstanceMethod(self, @selector(addObject:));
5     Method logAddObject = class_getInstanceMethod(self, @selector(logAddObject:));
6     method_exchangeImplementations(addObject, logAddObject);
7 }
8
9 - (void)logAddObject:(id)aObject {
10     [self logAddObject:aObject];
11     NSLog(@"Добавлен объект %@ в массив %@", aObject, self);
12 }
13
14 @end
15
```



Ассоциативные ссылки

```
1 @interface UITableView (Additions)
2
3 @property(nonatomic, strong) UIView *placeholderView;
4
5 @end
6
```

```
1 static char key;
2
3 @implementation UITableView (Additions)
4
5 -(void)setPlaceholderView:(UIView *)placeholderView {
6     objc_setAssociatedObject(self, &key, placeholderView, OBJC_ASSOCIATION_RETAIN_NONATOMIC);
7 }
8
9 -(UIView *) placeholderView {
10    return objc_getAssociatedObject(self, &key);
11 }
12
13 @end
14
```

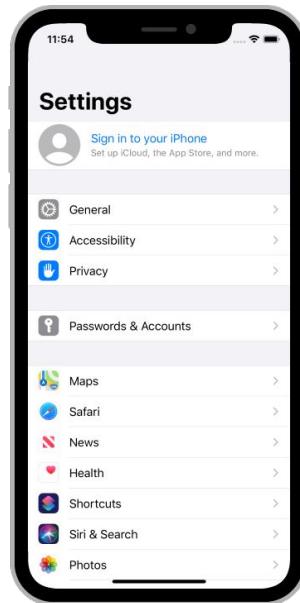
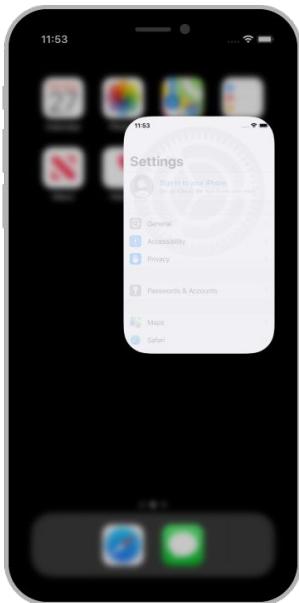
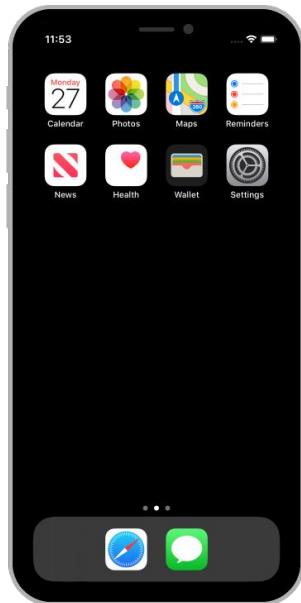


Жизненный цикл



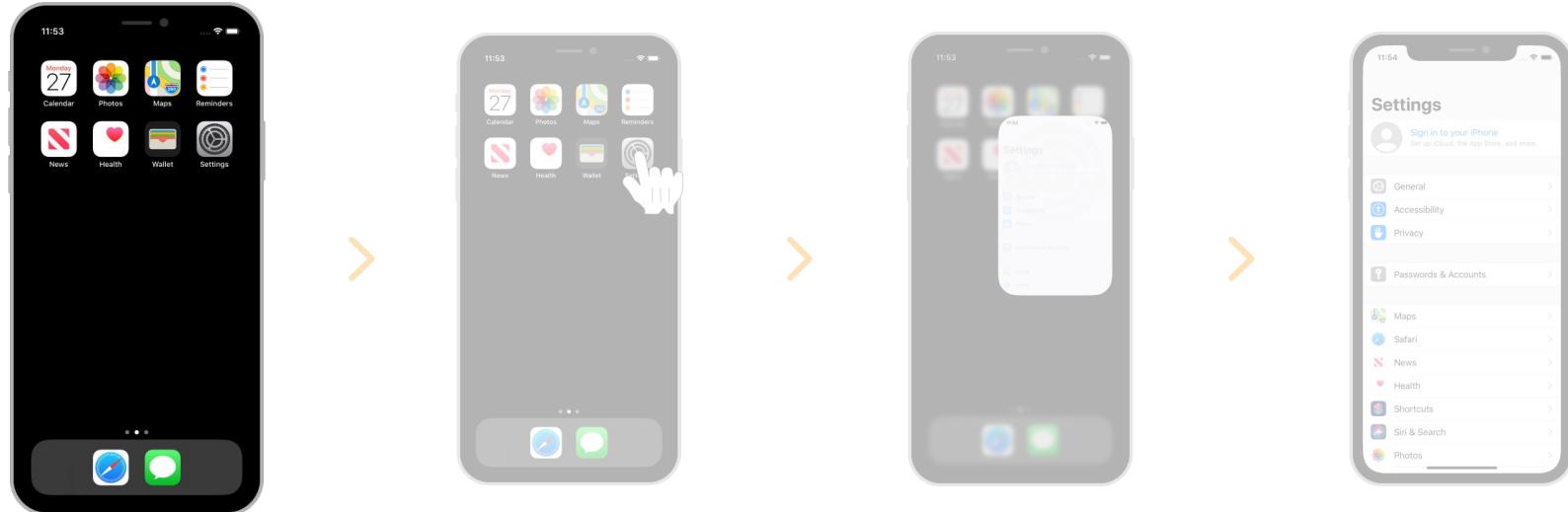


Как запускается приложение





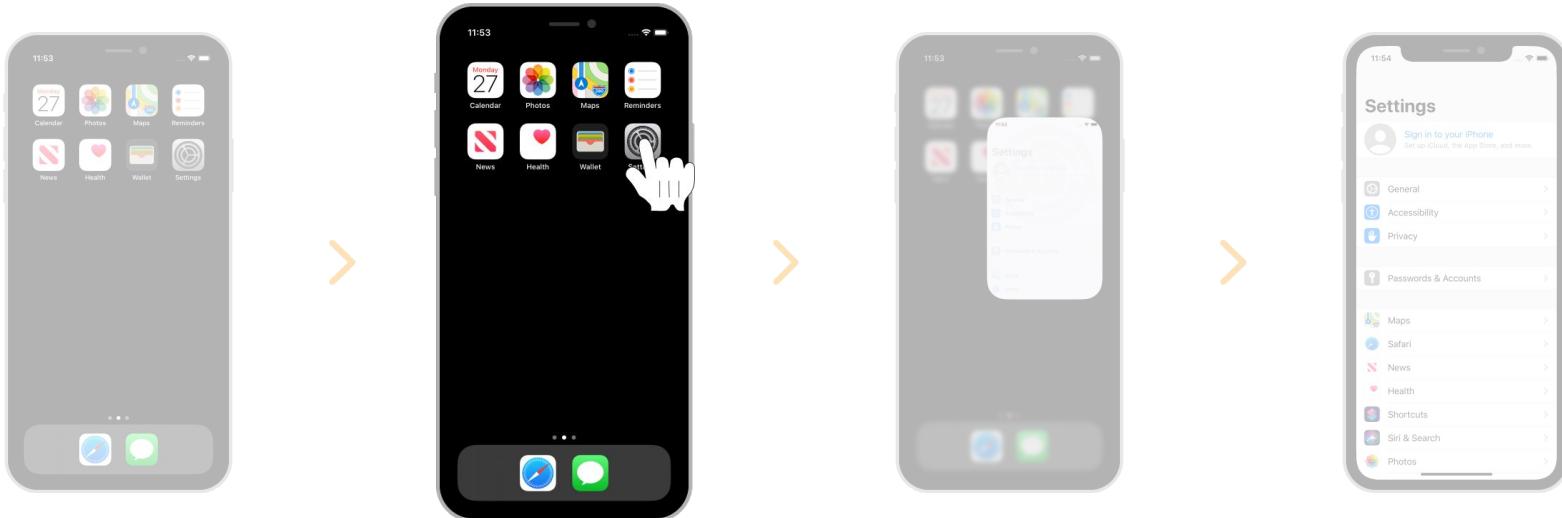
Как запускается приложение



Приложение не запущено



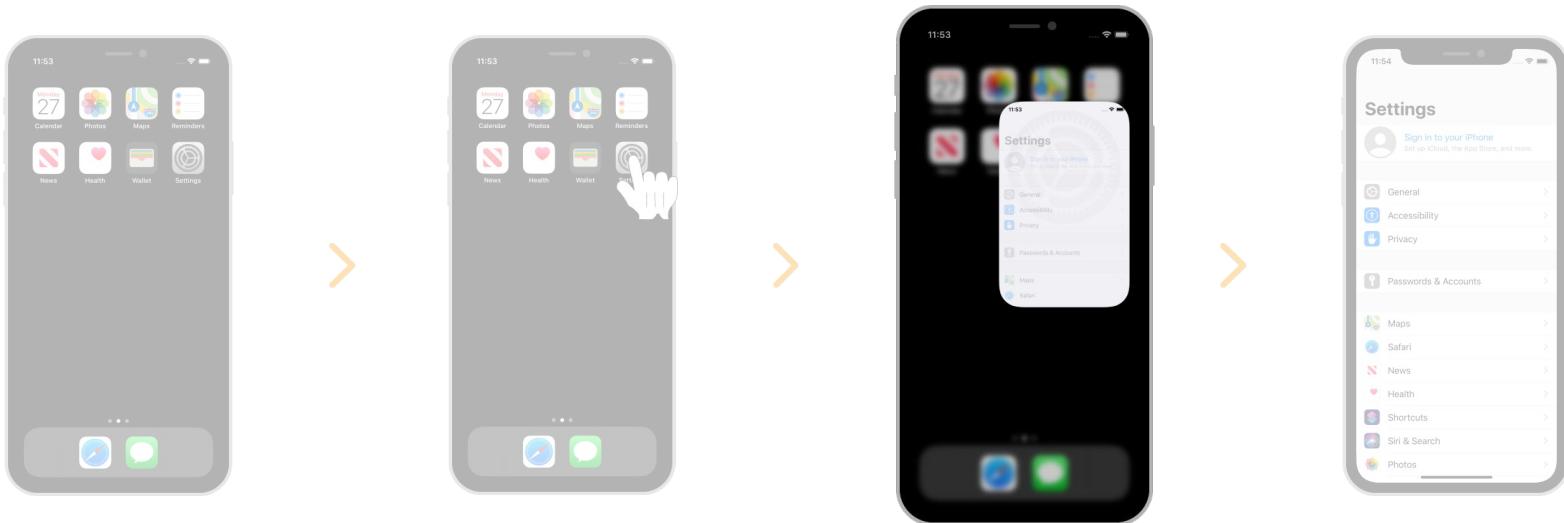
Как запускается приложение



Springboard запускает
приложение



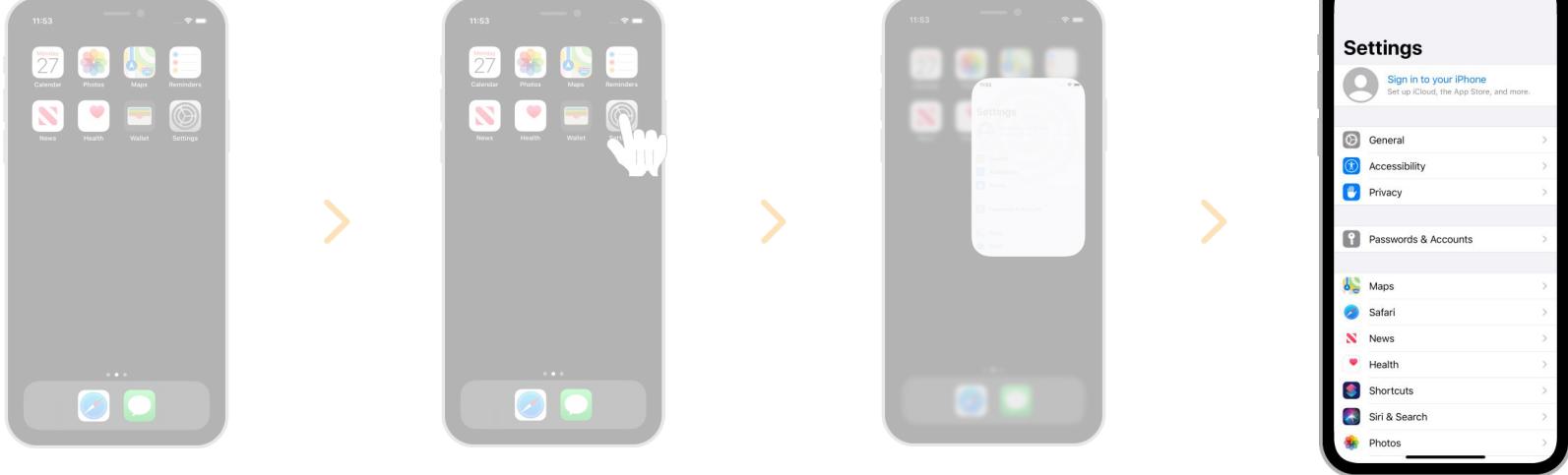
Как запускается приложение



Приложение загружается в память



Как запускается приложение



Приложение
запущено. Делегат
приложения получает
уведомления



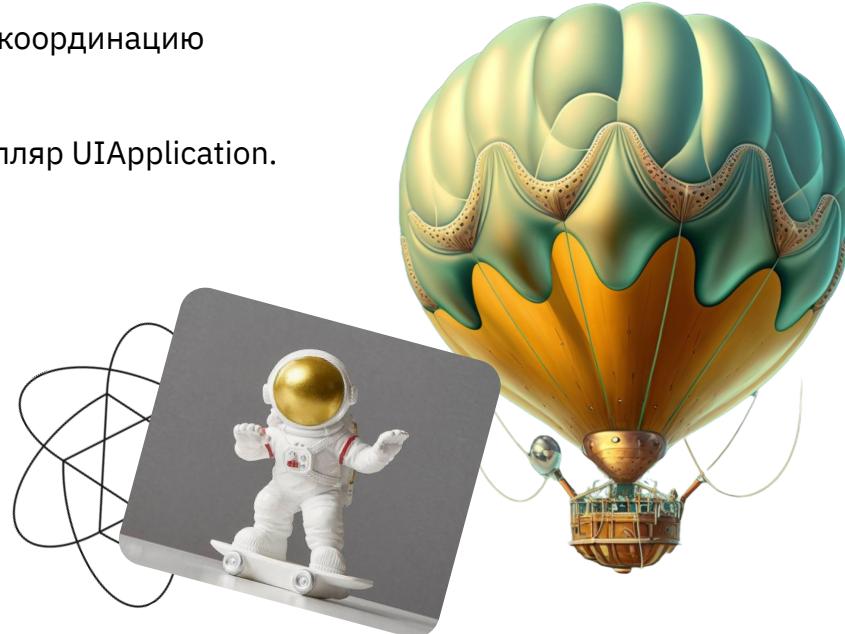
Как запускается приложение

```
int main(int argc, char * argv[]) {
    NSString * appDelegateClassName;
    @autoreleasepool {
        // Setup code that might create autoreleased objects goes here.
        appDelegateClassName = NSStringFromClass([AppDelegate class]);
    }
    return UIApplicationMain (argc, argv, nil, appDelegateClassName);
}
```



UIApplication

- ✓ Обеспечивает централизованный контроль и координацию приложений, работающих в iOS.
- ✓ Каждое приложение имеет ровно один экземпляр UIApplication.





Состояние приложения

Not Running

Active

Suspended

Inactive

Background



Состояние приложения

Not Running

Приложение не запущено или выполнялось,
но было остановлено системой.



Состояние приложения

Inactive

Приложение работает на переднем плане, но в настоящее время не получает событий. Приложение обычно остается в состоянии в течение короткого периода времени на пути к другому состоянию. В этом состоянии мы не можем взаимодействовать с пользовательским интерфейсом приложения.



Состояние приложения

Active

Приложение работает на переднем плане и получает события. Это нормальный режим для приложений переднего плана. Единственный способ перейти в активное состояние или выйти из него — через неактивное состояние. Пользователь обычно взаимодействует с пользовательским интерфейсом и может видеть ответ/результат действий пользователя.



Состояние приложения

Background

Приложение находится в фоновом режиме и выполняет код. Большинство приложений ненадолго переходят в это состояние на пути к приостановке. Запрос дополнительного времени выполнения может продлить состояние на некоторый период времени. Если приложение запущено в фоновом режиме, оно переходит в режим без активации неактивного режима.



Состояние приложения

Suspended

Приложение находится в фоновом режиме, но не выполняет код. Система переводит приложения в это состояние автоматически и не уведомляет их перед этим.

Во время приостановки приложение остается в памяти, но не выполняет никакого кода. При нехватке памяти система может удалить приостановленные приложения без уведомления, чтобы освободить место для приоритетных приложений.



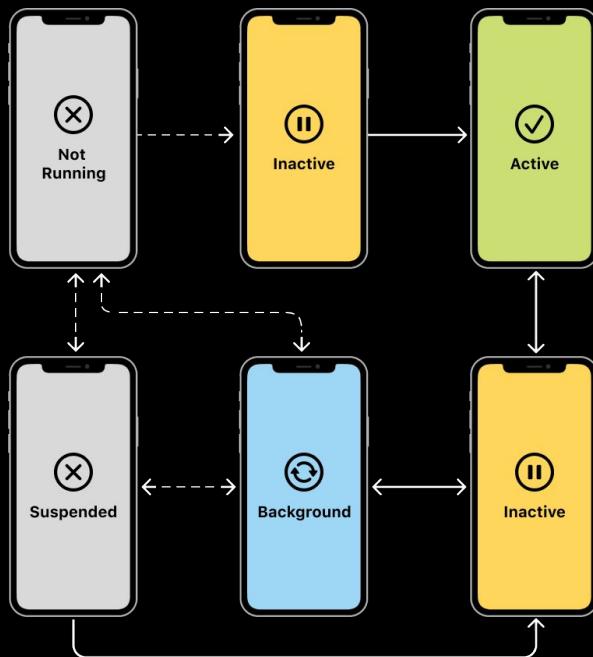
Состояние приложения

В iOS 12 и более ранних версиях, а также в приложениях, не поддерживающих сцены, UIKit доставляет все события жизненного цикла в объект UIApplicationDelegate. Делегат приложения управляет всеми окнами вашего приложения, включая те, которые отображаются на отдельных экранах. В результате изменения состояния приложения влияют на весь пользовательский интерфейс вашего приложения, включая содержимое на внешних дисплеях.





Жизненный цикл приложений





Жизненный цикл приложений

Переходы состояний сопровождаются соответствующими вызовами методов внутри вашего AppDelegate. объект. Эти методы дают вам возможность адекватно реагировать на изменения состояния.





Жизненный цикл приложений

```
- (BOOL) application : (UIApplication *) application  
willFinishLaunchingWithOptions : (NSDictionary *) launchOptions;  
  
- (BOOL) application : (UIApplication *) application  
didFinishLaunchingWithOptions : (NSDictionary *) launchOptions;  
  
- (void) applicationDidBecomeActive : (UIApplication *) application;  
  
- (void) applicationWillResignActive : (UIApplication *) application;  
  
- (void) applicationDidEnterBackground : (UIApplication *) application;  
  
- (void) applicationWillEnterForeground : (UIApplication *) application;  
  
- (void) applicationWillTerminate : (UIApplication *) application;
```

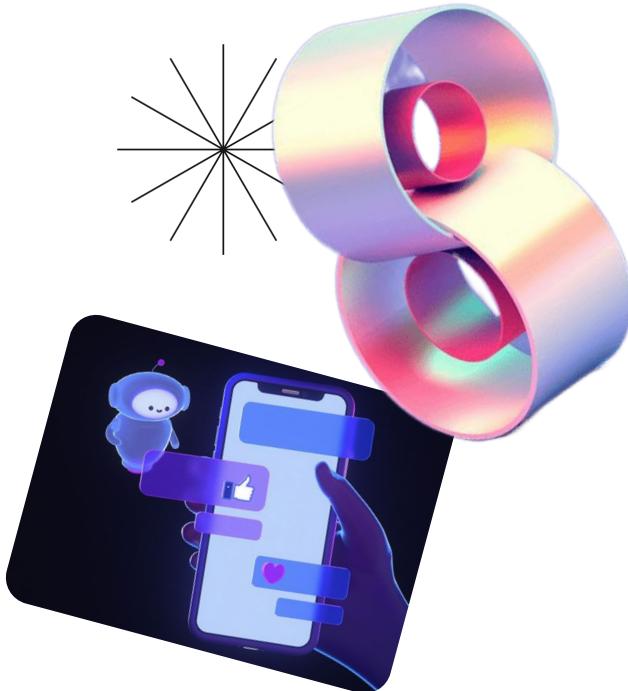
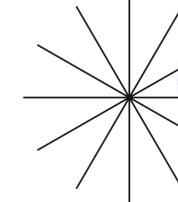


Жизненный цикл приложений

Application termination 🔥

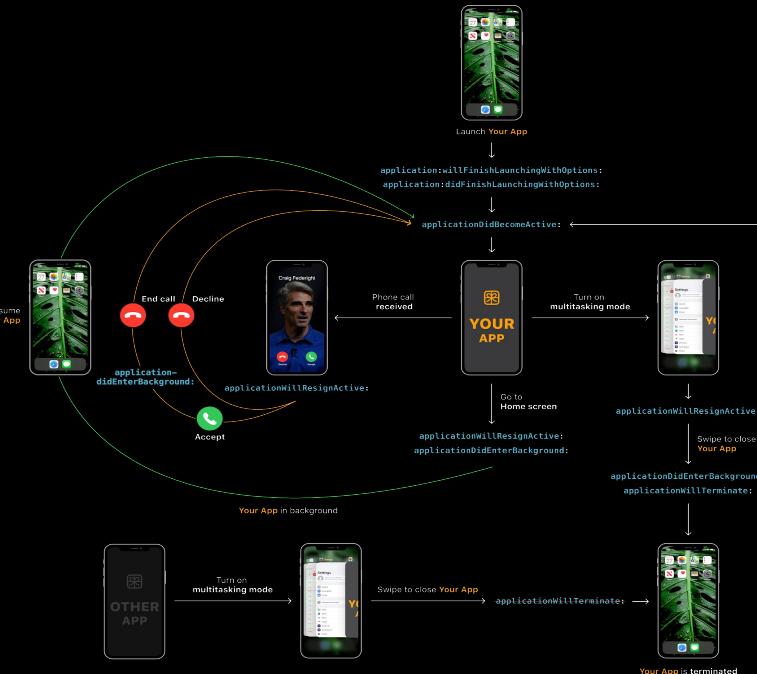
Приложение должно быть готово к прекращению работы в любое время и не должно ждать сохранения пользовательских данных или выполнения других важных задач. `applicationWillTerminate:` вызывается для приложений, которые не приостановлены.

Метод не называется предварительной перезагрузкой устройства.



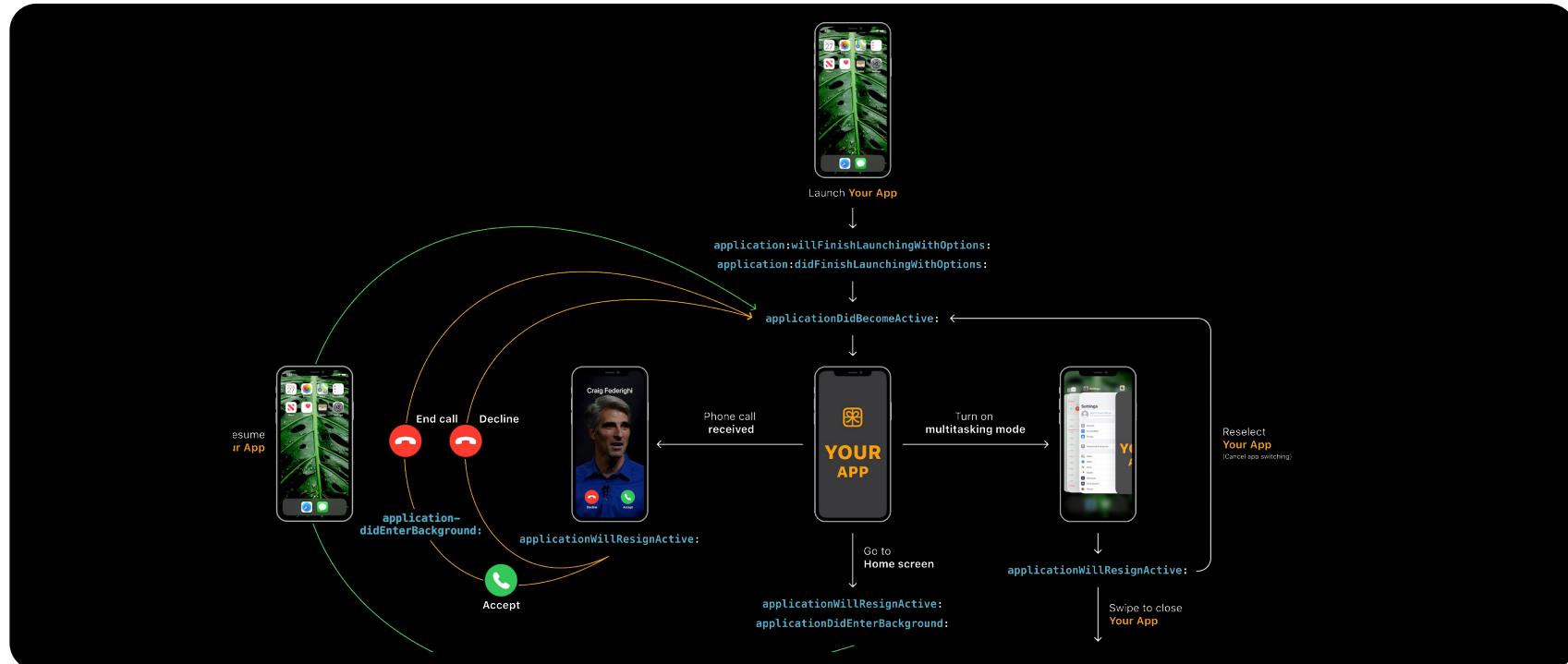


Жизненный цикл приложений



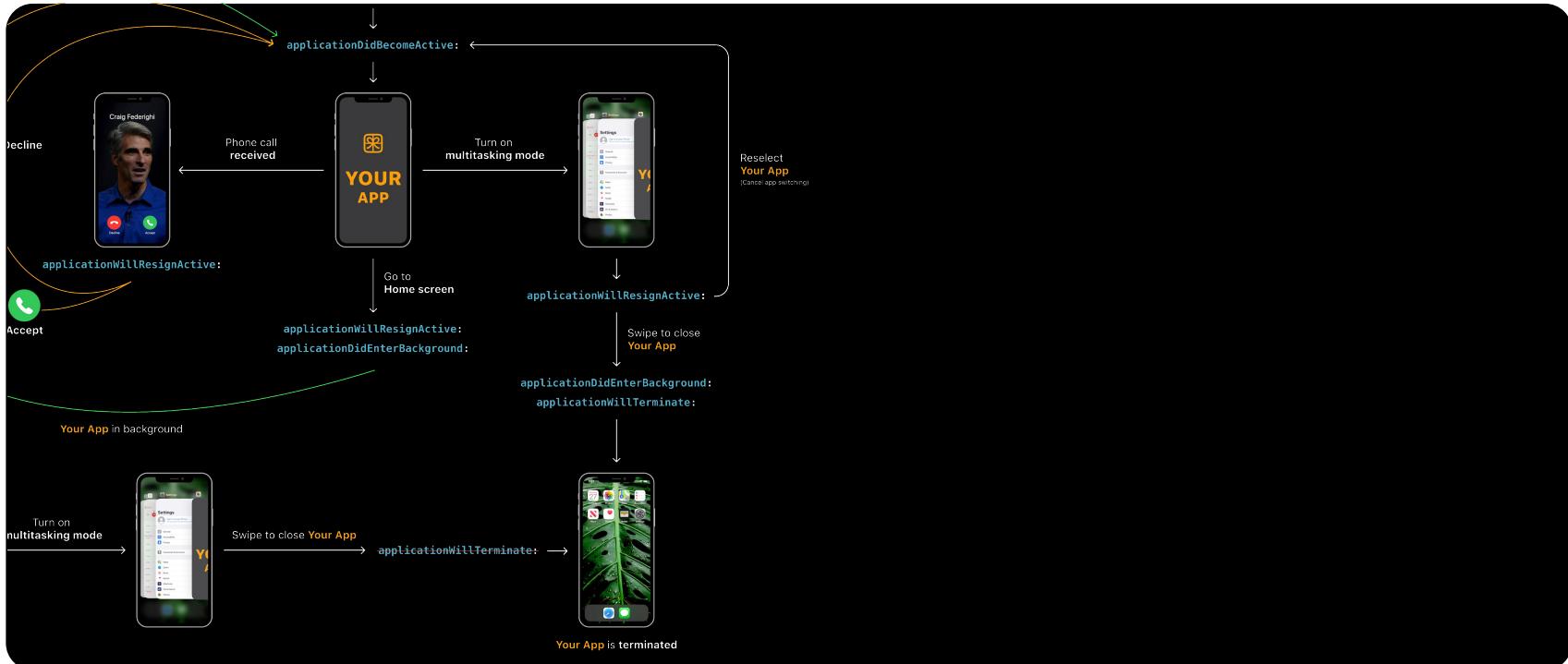


Жизненный цикл приложений



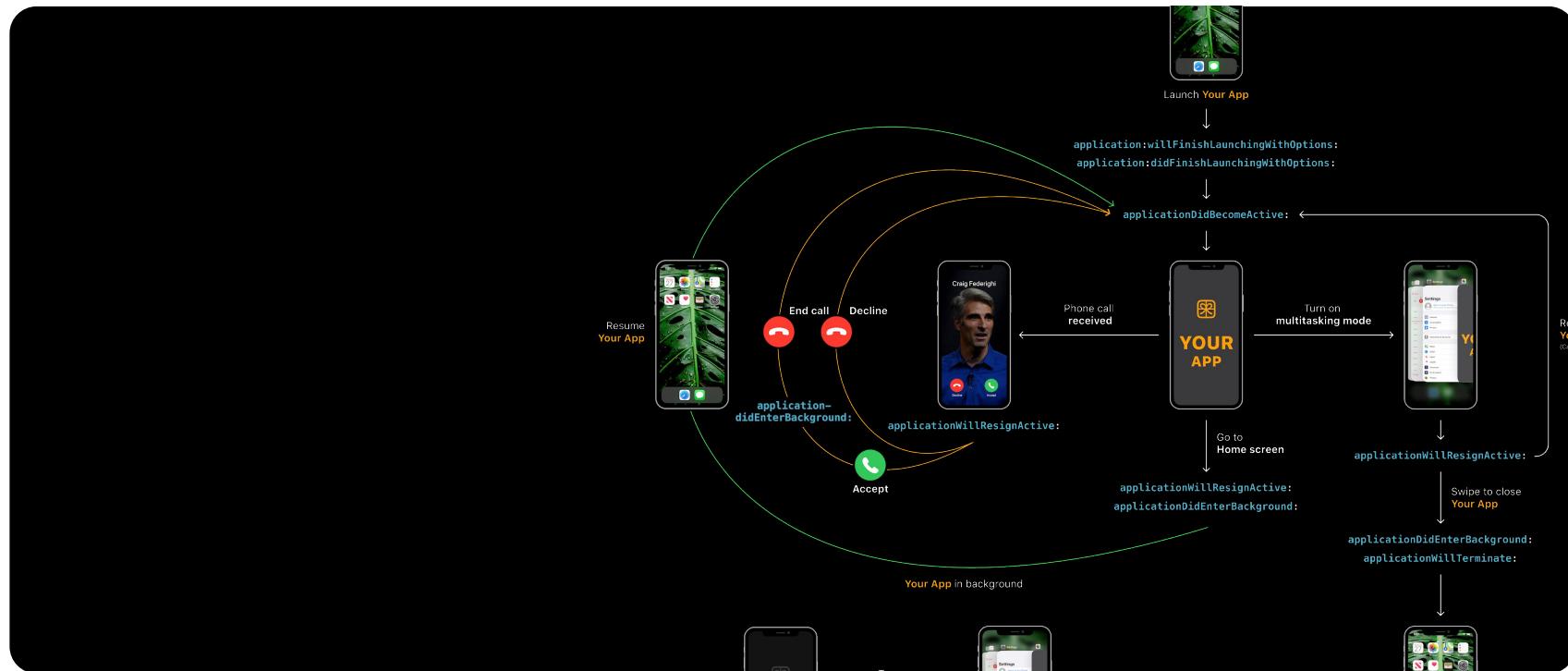


Жизненный цикл приложений



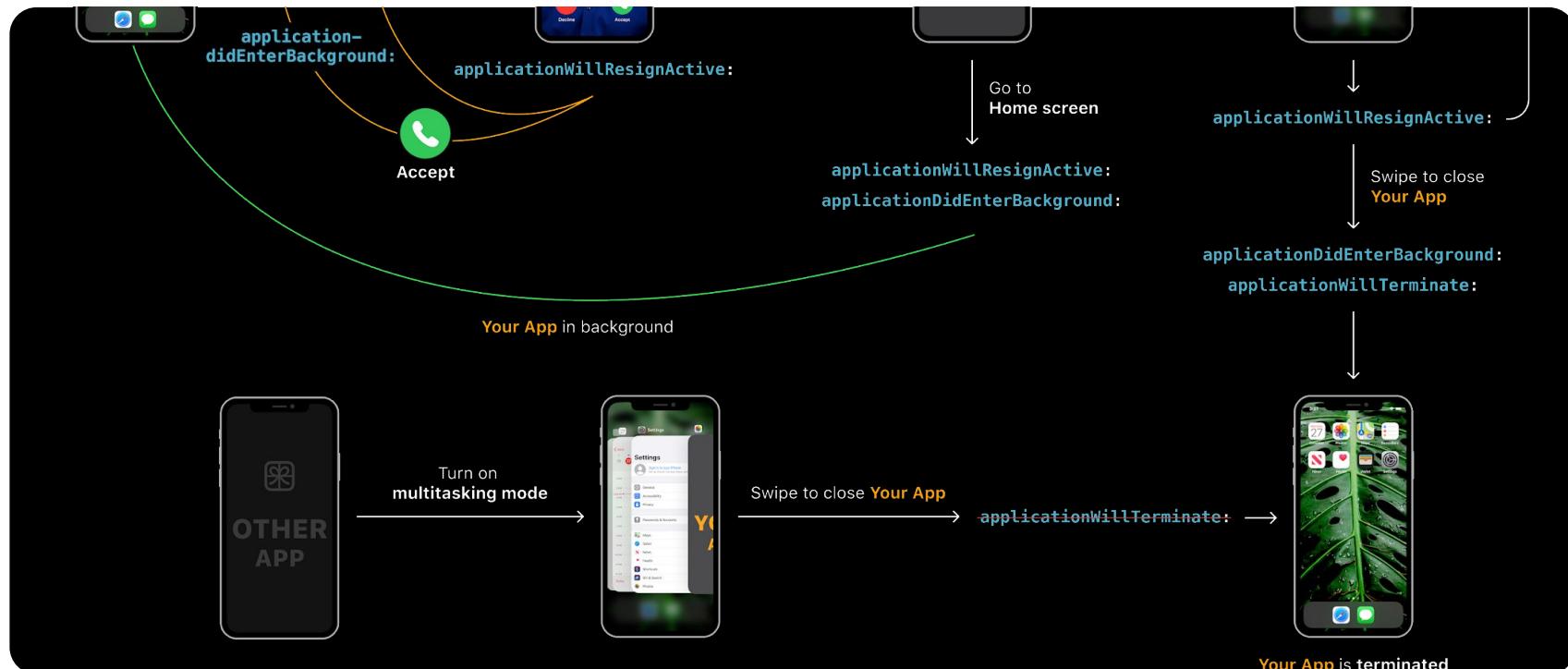


Жизненный цикл приложений



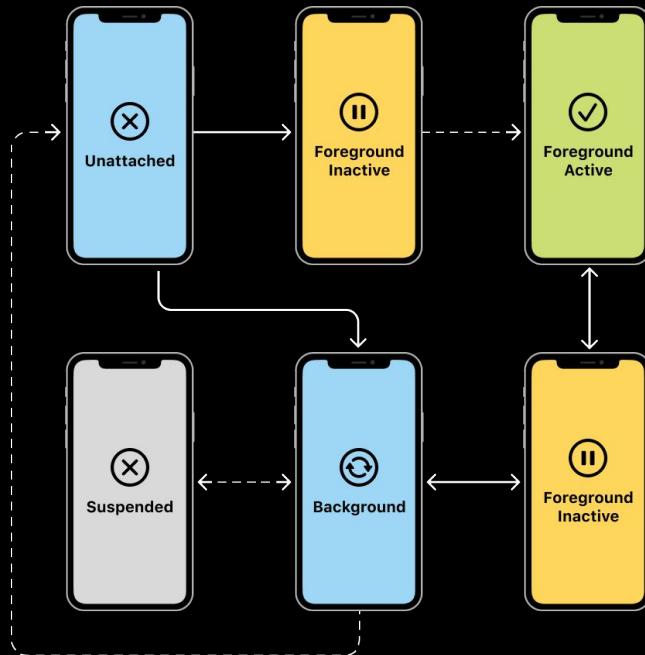


Жизненный цикл приложений





Жизненный цикл на основе сцены





Жизненный цикл на основе сцены

```
- (void) scene:(UIScene *)scene
    willConnectToSession:(UISceneSession *)session
        options:(UISceneConnectionOptions *)connectionOptions;

- (void) sceneDidDisconnect:(UIScene *)scene;

- (void) sceneDidBecomeActive:(UIScene *)scene;

- (void) sceneWillResignActive:(UIScene *)scene;

- (void) sceneWillEnterForeground:(UIScene *)scene;

- (void) sceneDidEnterBackground:(UIScene *)scene;
```



Немного истории





Что почитать?

- 📌 Стивен Кочан. «Программирование на Objective-C».
- 📌 Скотт Кнастер, Вакар Малик, Марк Далримпл.«Objective-C. Программирование для Mac OS X и iOS».
- 📌 <https://www.objc.io/>
- 📌 <https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/>





Вопросы?

Вопросы?



Вопросы?

