ЭКРАН

Обработка нажатий производится: Responder chain обработка touch событий (цепочка ответственности) внутри которого лежит механизм Hit Test

Этапы обработки нажатий на View

* Нажатие на экран, на кнопку;
* Железо оповещает систему о нажатии;
* Система упаковывает информацию о нажатие в объект класса UIEvent, который содержит координаты этого нажатия, время нажатия, количество пальцев которые участвовали в нажатии;
* View обрабатывает полученное событие.

Разница между UITouch и UIEvent:

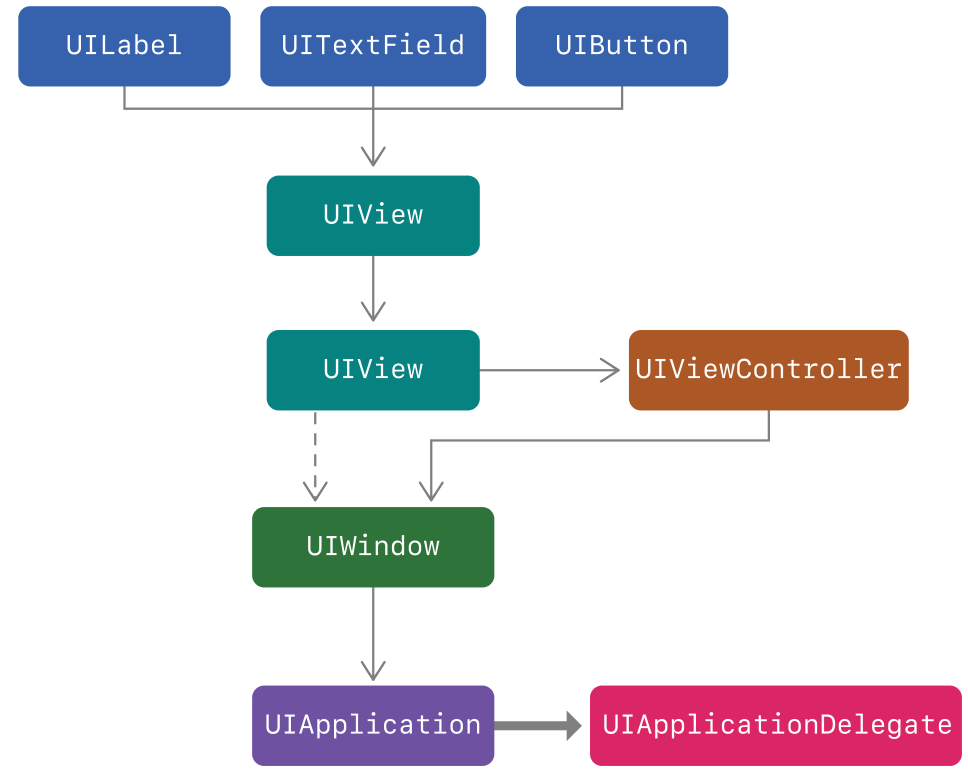
* UITouch – Инкапсулируют внутри себя все данные которые позволяют понять, каким образом случилось то что случилось, т.е если мы нажали на экран то где именно это было нажато, с какой силой это было или какое кол-во пальцев участвовало в процессе;
* UIEvent – Объект инкапсулирующий внутри себя UITouch т.е рассказывает что случилось и предоставляет данные которые позволяют отреагировать на это. На каждый Touch пальцев заводится отдельный UITouch объект.

Hit test – Проверяет относятся ли координаты нажатия к Bounds этой View. Используется для поиска views с которой взаимодействовал пользователь. Если у нас маленькая кнопка и нужно чтобы она нажималась даже если человек промахивается на несколько пиксилей, то нужно переопределить Hit test. Начинает свою работу в UIWindow.

First Responder – это просто корневой элемент списка, и если ответчик не может обработать определенное действие / событие, действие рекурсивно отправляется следующему ответчику списка, пока кто-нибудь не сможет обработать действие или список не закончится.

Next Responder – возвращает следующего ответчика в цепочке ответчиков или nil, если следующего ответчика нет. В том случае если нажатие не обработалось у кнопки, мы спускаемся вниз до UIApplication и скипаем нажатие, если Event отработал то вниз не идет.

**UIApplicationDelegate**– информирует о важных событиях (запуске приложения, не хватки памяти, завершении работы приложения.

1. Обращение к UIApplication, он не обрабатывает событие поэтому оно переходит к UIApplicationDelegate;
2. UIApplicationDelegate имеет ссылку на UIWindow, обработка пойдет сюда. UIWindow при помощи Hit Test определить по какой ветке UIViewControllers пойти до Button / TextField / Label.

Как система обрабатывает нажатия:

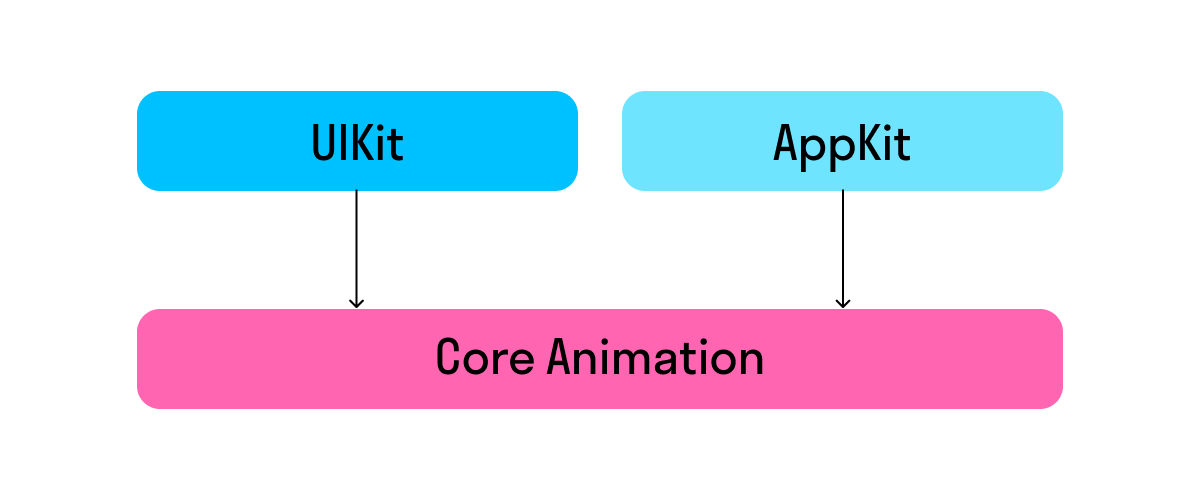
Что происходит, когда пользователь касается экрана? Для каждого нажатия система создаёт событие и передаёт его в приложение. Там оно поступает в обработку к UIApplication, который с помощью метода sendEvent: пробрасывает его дальше –в window. Задача объекта класса UIWindow – запустить процесс прохода по дереву вьюх и найти ту, которой адресован этот touch. Если положить две кнопки друг на друга, то почему нажиматься будет верхняя? – Проходим с помощью Hit Test в самую глубину и потом весь цикл наоборот и если никто не отвечает на это нажатие то оно не выполняется.

UIResponder – отвечает на пользовательские жесты. Но как наша вью узнает, что пользователь нажал именно на нее и что ей нужно обработать этот экшн? Здесь нам на помощь приходит механизм Responder Chain. UIViewController, UIView, UIApplication наследуют от UIResponder (Последовательная цепочка объектов).

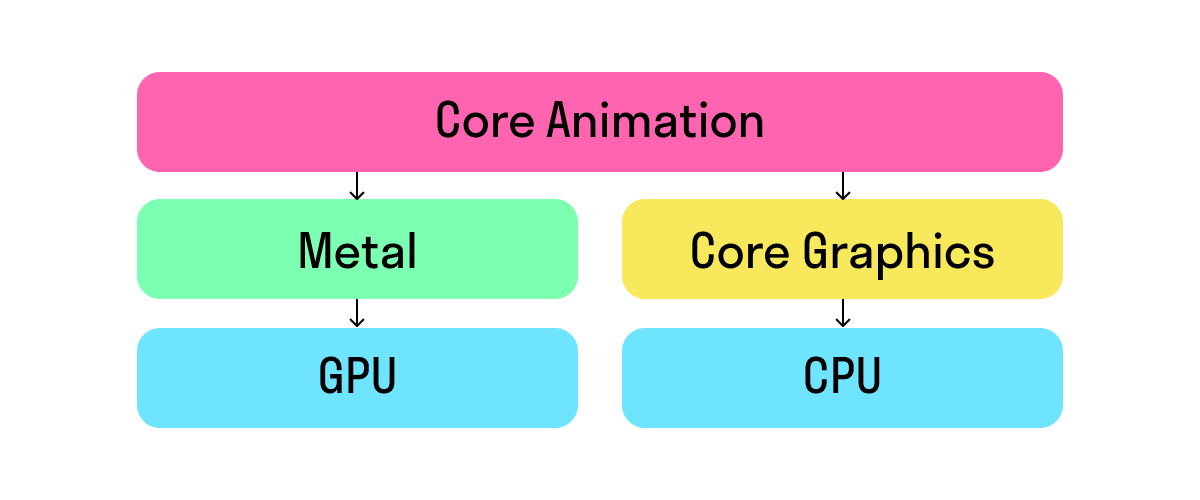
Когда пользователь нажимает на экран это событие попадает в наше приложение (объект UIApplication). Дальше оно отправляется в UIWindow, где и запускается цепочка поиска firstResponder’а, в границах которого и было произведено нажатие. Цепочка запускается рекурсивным вызовом метода по всей иерархии дочерних вью:

Если точка находится внутри вью, поиск продолжается уже среди своих дочерних вью, вызывая метод hitTest у них. Так продолжается пока не будет найдена самая нижняя в иерархии (самая верхняя на экране) вью, в которую попадает нажатие.

Слои



Верхний уровень в случае iOS является UIKit, в случае macOS - AppKit. Этот уровень отвечает за распознавание пользовательских активностей (тач пальцем в область экрана или наведение курсора мышки на какой-то элемент). Ниже лежит Core Animation, вот он-то и отвечает за то, что мы видим на экране и каким будет каждый элемент.



Несмотря на то, что Core Animation отвечает за графический интерфейс, отрисовкой контента занимается не он. Эту функцию выполняют более низкоуровневые фреймворки Core Graphics и Metal, с той лишь разницей, что Core Graphics для вычислений использует CPU, а Metal - GPU.

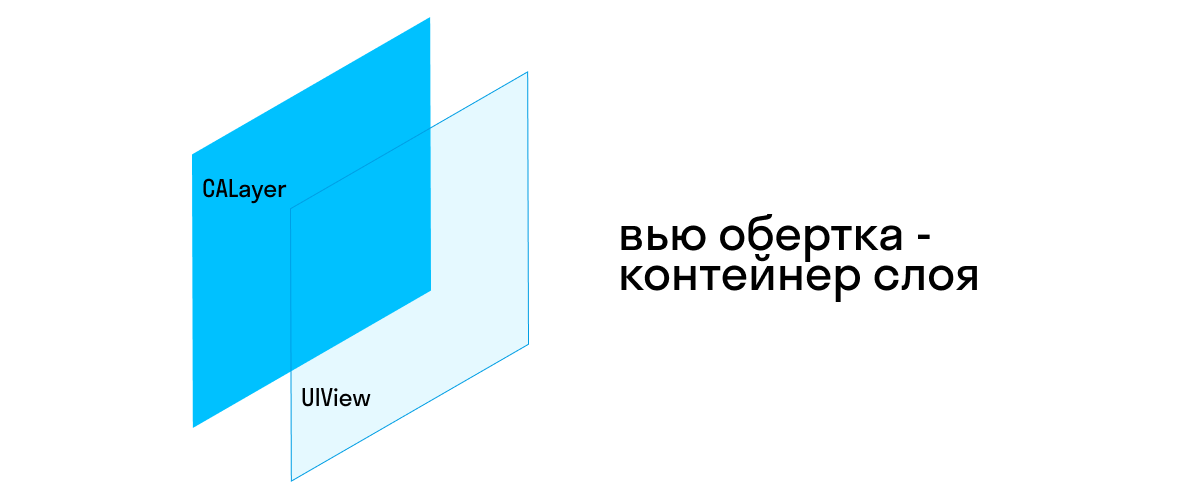
Грубо говоря, Core Animation отвечает на вопрос «Что рисовать?», а Core Graphics и Metal - «Как рисовать?». Можем ли мы работать напрямую с этими фреймворками? Можем. Это даже положительно скажется на производительности нашего приложения, но очень сильно увеличит стоимость разработки. То, что мы можем сделать в UIKit или Core Animation за пару строк кода, в Core Graphics или Metal может занять десятки. Важно понимать, как это устроено, однако использование не всегда оправданно.

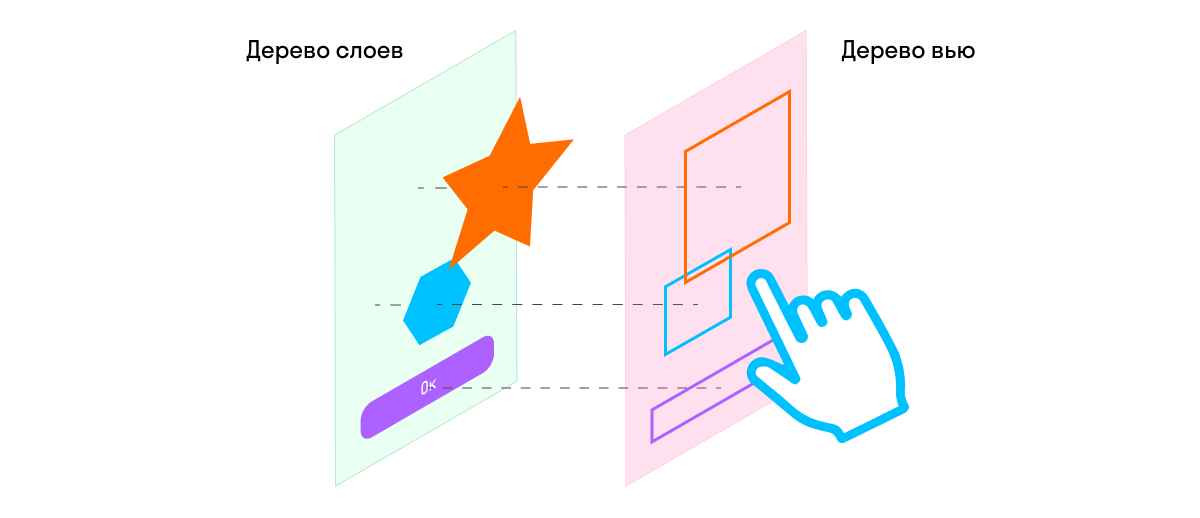
UIView и CALayer

CALayer (Core Animation Layer) – слой, это сущность которая во многом похожа на. UIView. Отвечает за отображение, на ней можно делать анимацию, но нажатие, свайп и т.д она обрабатывать не может – это отличие от UIView. CALayer работает только с графикой. Находится в фреймворке (CoreAnimation) он ниже чем UIKit т.е на CALayer можно делать более сложные анимации и рисования.

Отрисовки на CALayer выполняются на GPU (Графический процессор) а UIView на CPU (центральный процессор), соответственно отрисовка на CALayer происходит быстрее т.к разгружается центральный процессор и главный поток приложения.

View является контейнером слоя и предоставляет более высокоуровневое API для работы с внешним видом.





Точно так же, как и вью, мы можем добавлять слои друг на друга и выстраивать целые иерархии. Все они складываются в свое дерево, не ограничиваясь той вью, которой они принадлежат. Все View обязательно содержат Layer, а вот Layer могут существовать без привязки к какому-то View.

View и CALayer хоть и похожи, но некоторые плюсы все равно на стороне CALayer:

* Позволяют создавать более сложную анимацию;
* Позволяют работать с тенями.

Геометрия UIView - Frame (рамка) и Bounds (граница)

Frame – это прямоугольник, у которого есть x и y и размером width,height относительно своего контейнера – родителя (SuperView).

Bounds – это внутренний прямоугольник, заданный координатами (x,y) и размером (width,height) относительно собственной координатной системы (0,0). Ничего не знает о SuperView.

Когда использовать frame и когда использовать bounds?

Поскольку frame связывает местоположение view в superview, используйте его при внесении внешних изменений: ширины или вычисления расстояния между view и вершиной его родительского view.

Используйте bounds для внутренних изменений: рисования или организации subview в пределах view. Также используйте bounds для получения размера view, если вы сделали какие-то преобразование для него.

Center – точка которая описывает центр View в системе координат родитель. Center зависит от frame как и frame от center (если поменяем центр то поменяется frame и наоборот)

Origin – значения в углу frame / bounds с которого идет отсчет. Например: 0,0

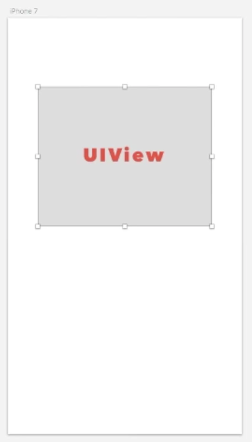
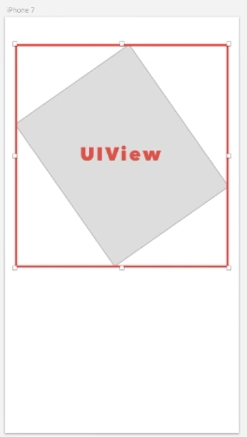
Х – координат

У – координат

Height – размер

Width – размер

Начало координат в iOS верхний левый угол

Первоначально bounds имел один размер но после поворота стал больше, об это необходимо помнить работая с элементами. В этом ситуации frame раздувается а bounds остается таким-же.

Жизненный цикл ViewController

На маленький проектах это не так важно нежели на крупных, ибо появляются уже проблемы с памятью. Начинают теряться данные, некорректно работать View и тд.

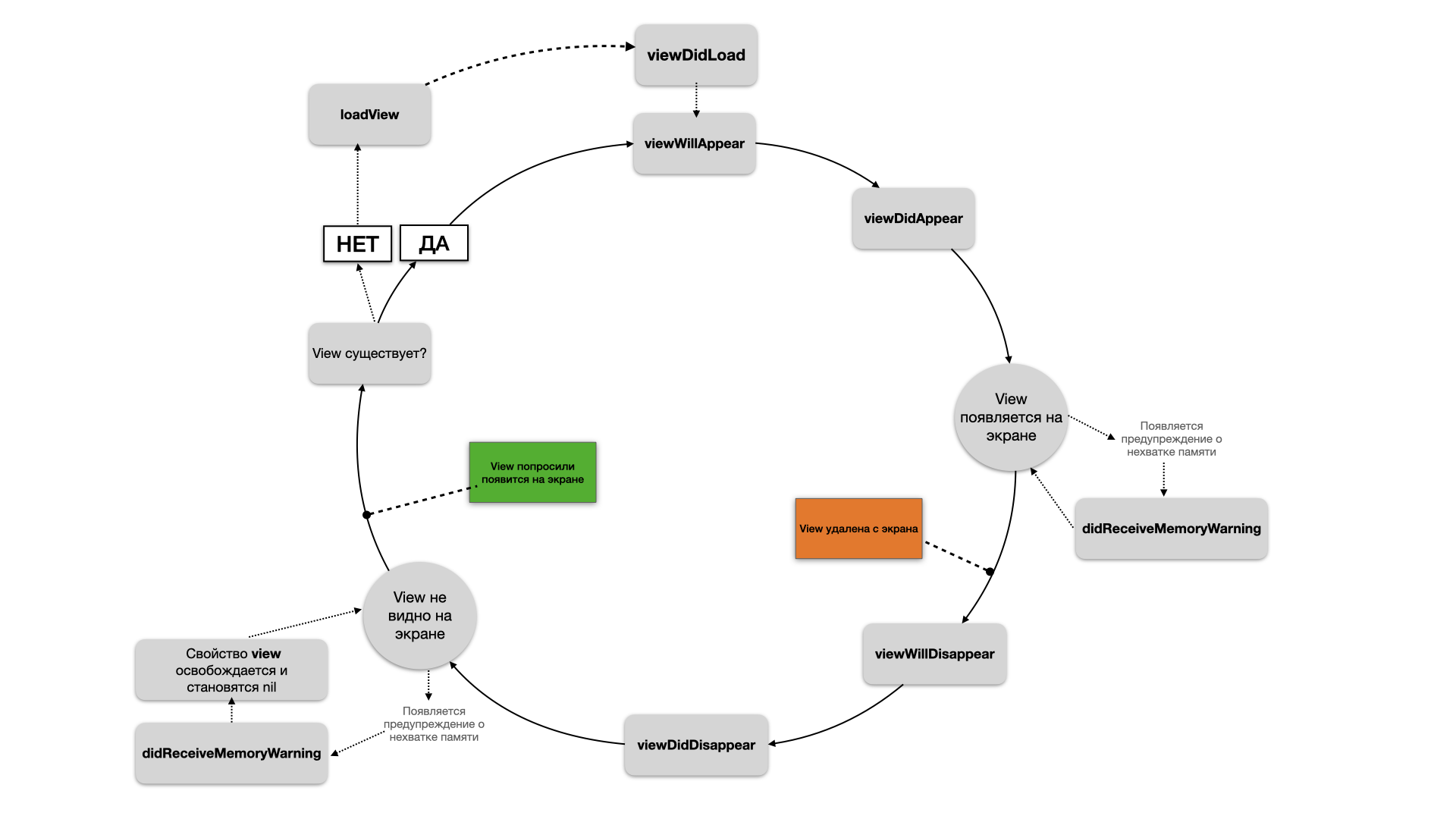


* LoadView – Переопределяется этот метод только в том случае, если нужно построить весь интерфейс для ViewController’a из кода а не Storyboards / Nib. Вызывать super не нужно;
* viewDidLoad – Срабатывает после загрузки view. Вызывается один раз в жизненном цикле когда View загружается в память. Границы вью ещё не образованы, поэтому к ширине, высоте и прочим параметрам обращаться не можем. На экране не видим но в памяти оно уже загрузилось. Оутлеты уже установлены и можно производить настройку интерфейса из модели но могут быть не вычислены размер и положение элементов на экране поэтому старайтесь не использовать код связанный с .bounds в этой функции. Тут неизвестны размеры frame UI элементов;
* viewWillAppear – Перед появлением View на экране. Тут геометрические настройки уже доступны. Работаем с данными которые могли измениться пока контроллер не отображается на экране (все размеры уже установлены);
* **viewWillLayoutSubViews –** Вызывается, чтобы уведомить ViewController о том, что его view собирается разместить свои subviews. Этот метод вызывается каждый раз, когда frame изменяется, например, при повороте экрана;
* **viewDidLayoutSubviews –** Вызывается, чтобы уведомить ViewController о том, что его view только что разместило свои subview;
* viewDidAppear – Экран только что был показан. Работаем с геометрией, начинаем работу с анимацией (не раньше и не позже);
* viewWillDisappear – Перед тем как View будет закрыто. Если нужно останавливаем анимацию и делаем чистку экрана если необходимо;
* viewDidDisappear – После того, как View закрылось.
* Did receive memory warning – метод срабатывает когда на устройстве заканчивается память. Если не среагировать то приложение крашнится. Функция вызывается по мере жизни нашего view.
* Deinit – выгружает объект из памяти, срабатывает в последнюю очередь.

setNeedsLayout или layoutIfNeeded, чтобы запросить layout pass для view. Ключевое различие между этими методами заключается в том, что layout pass обновляет view вот так:

* setNeedsLayout: немедленно возвращается без обновления layout. Вместо этого он отмечает layout во view как измененный и планирует отложенный layout pass для запуска в application run loop. Обновление Layout при удобной возможности для системы;
* layoutIfNeeded: вызывает layoutSubviews на приемнике, если есть ожидающие изменения, чтобы заставить layout engine немедленно обновить размер и положение subviews из своей внутренней модели. Принудительный (ручной) пересчет Layout.

Обязательно вызывать super.view для всех методов потому что все эти методы существуют у класса UIViewController, если забыть написать super может тупить приложение.



Layout – тема относиться к жизненному циклу ViewControllers

Func viewWillLayoutSubviews()

Между will и did происходит Autolayout.

Func viewDidLayoutSubviews() – обычно используют для ручного подсчета frame (когда есть сложный UI)

Эти методы вызываются каждый раз когда frame был изменен (довольно часто). Необходимы для отслеживания геометрических параметров элементов до и после Autolayout. Могут вызываться много раз и мы не можем их предугадать.

Autolayout помогает создавать динамические пользовательские интерфейсы, масштабируемые и адаптированные к каждому размеру и ориентации устройств. Autolayout вычисляет размер и положение view в иерархии view на основе ограничений (constraints).

awekeFromNib() – метод вызывается для всех элементов которые появляются из Storyboard и Xib, происходит до того как будут установлены все оутлеты, постарайтесь не использовать этот метод и поместить код во viewDidLoad или viewWillApрear. Этот метод работает чуть быстрее чем viewDidLoad.

Навигация между контроллерами:

Segua и кодом

Навигация между контроллерами, быстрее работает через код чем через segua. У каждой segua должен быть уникальный ID.

Unwind segua (переходы размотки) – это всегда переходы от текущего контроллера представление к контроллеру которые уже присутствуют в иерархии навигации. Они могут осуществлять переход как на один так и на несколько шагов. Нужны для того чтобы: быстрее делать переходы и не создавать дополнительные связи которые будут возвращать назад.

Как это работает?

Создаем функцию unwind в контроллере в который хотим совершать переход. После этого создаем кнопку на viewController из которого хотим возвращаться назад и от этой кнопки протягиваем segua в Exit.

Storyboard / Xib (Nib) / Верстка в коде

Существует три типа подходов к дизайну пользовательского интерфейса:

* Раскадровки iOS: визуальный инструмент для создания нескольких представлений приложений и переходов между ними.
* NIBs (или XIBs): каждый файл NIB соответствует одному элементу представления и может быть размещен в построителе интерфейса, что также делает его визуальным инструментом. Обратите внимание, что название “NIB” происходит от расширения файла (ранее .nib, а теперь .xib, хотя старое произношение сохранилось).
* Пользовательский код: т. е. Никаких инструментов графического интерфейса, а скорее обработка всего пользовательского позиционирования, анимации и т.д. программно.

Чем Point (pt) отличается от Pixel (px)?

Pixel — точка на экране, а Point — плотность точки на экране.

Дополнительно:

IBOutlet – это связь элемента UI (кнопки или лейбла) с его ссылкой в коде.

IBAction – это действие (или метод, как удобнее) в коде, которое можно подключить к некоторому событию (нажатие кнопки, например) в разработанном интерфейсе.

При копировании элементов на экране (кнопок например), IBOutlet не копируются а IBAction копируются.

Почему IBOutlet weak?

Лагает UITableVIew

Решение:

* Проверить dequeueReusableCell – если ячейки не переисользуется это может не стабильно работать;
* Проверить как берутся данные для ячейки;
* Проверить Constraints, чем больше элементов лежит в вашей ячейке, тем больше уравнений приходится решать для расчёта высоты будущей ячейки;
* Pagination в TableView;
* Проверить на каком потоке все это работает, как получается JSON;
* Проблема с автоматическим рассчетом размера ячеек, лучше использовать в ручную;
* Долго можно грузиться из-за большого размера картинки;
* Проверить захишированы ли картинки.