3-UIKit view and controls

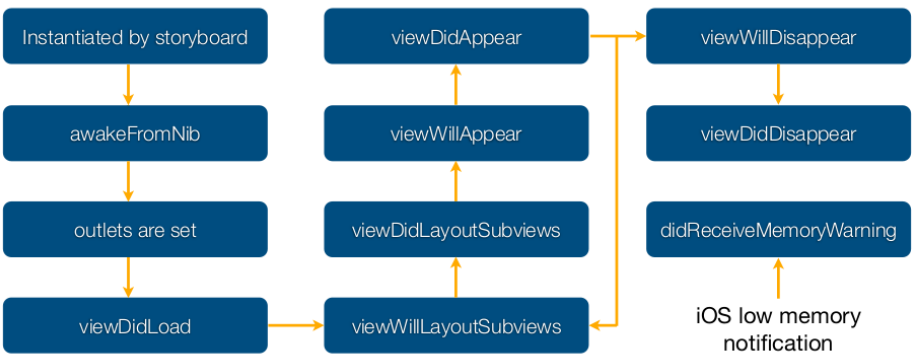
Il ciclo di vita del view controller

Il view controller riceve messagi quando avvengono determinati eventi relativi al suo ciclo di vita. Ogni view controller è una sottoclasse della classe `UIViewController` che definisce un insieme di metodi che devono essere eseguiti come controllori di ricezione dei messaggi del ciclo di vita, ogni sottoclasse deve sovrascrivere i metodi della super-classe per gestire al meglio questi messaggi.

Il ciclo di vita del view controller:

1. Il view controller deve essere creato, mediante lo storiboards, programmativamente
2. Gli outlets del view vengono impostati
3. Le viste dei controllori pososno apparire e scomparire dallo schermo
4. Notifica di poca memoria

> Nota
> Ogni volta che avviene uno di questi eventi il sistema lo notifica al view controller.



viewDidLoad

Dopo che il view controller è stato creato e gli outlet sono stati impostati viene invocato `viewDidLoad` (solo una volta per controller), in questo metodo la maggior parte dei view controller viene inizializzata. A questo punto tutti gli outlets sono stati impostati, ma gli estremi delle viste no, quindi non è ancora sicuro eseguire in impostazione delle geometrie (geometry-based settings) nel `viewDidLoad`.

```
- (void)viewDidLoad{
        [super viewDidLoad];
        // Do any additional setup after loading the view, typically from a nib.
        // ...
}
```

viewWillAppear

Quando una vista sta per apparire sullo schermo viene invocato (anche più volte) `viewWillAppear`, l'argomento segnala se la vista appare mediante un animazione o istantaneamente. Generalmente questo metodo viene utilizzato per:

- Operazioni che sono relative a cambiamenti che avvengono mentre la vista non è sullo schermo
- Le operazione di lunga durata che possono non essere necessarie se la vista non viene mai mostrata o potrebbero bloccare il rendering della vista se eseguite nel `viewDidLoad`

```
- (void)viewWillAppear:(BOOL)animated{
        [super viewWillAppear:animated];
        // ...
}
```

viewWillDIsappear

Quando una vista sta per scomparire dallo schermo viene invocato (anceh più volte) il metodo `viewWillDIsappear`, l'argomento segnala se la vista scompare mediante un animazione o istantaneamente. Generalmente questo metodo viene utilizzato per:

- Salvare lo stato della vista per successivi recuperi
- Liberare risorse che non sono necessarie e che possono essere riottenute successivamente la volta successiva che la vista appare.

```
- (void)viewWillDisappear:(BOOL)animated{
        [super viewWillDisappear:animated];
        // ...
}
```

viewWillLayoutSubviews e viewDidLayoutSubviews

`viewWillLayoutSubviews` e `viewDidLayoutSubviews` sono metodi che vengono invocati quando una sotto-vista di una vista stanno per essere/sono appena state disposte. Tra l'esecuzione di questi due metodi viene impostato un autolayout, il codice geometry.related può essere scritto all'interno di questi due metodi
.

```
- (void)viewWillLayoutSubviews{
        [super viewWillLayoutSubviews];
        // ...
}
```

```
- (void)viewDidLayoutSubviews{
        [super viewDidLayoutSubviews];
        // ...
}
```

autorotation

Il view controller è responsabile di gestire la rotazione del dispositivo (deve essere impostato nelle impostazionidel progetto `Info.plist`), se può ruotare, ovvero se il metodo `shouldAutorotate` ritorna `YES`, gli elementi della vista dovrebbero ruotare.

```
- (BOOL)shouldAutorotate{
        return YES;
}
```

Se la rotazione autmatica è abilitata, le orientazioni supportate sono definite come valore (`UIInterfaceOrientationMask`) di ritorno del metodo `supportedInterfaceOrientations`

```
- (NSUInteger)supportedInterfaceOrientations{
        return UIInterfaceOrientationMaskPortrait;
}
```

Per notificare al view controller che c'è stata un evento di rotazione vengono invocati alcuni eventi:

```
- (void)willRotateToInterfaceOrientation:(UIInterfaceOrientation)toInterfaceOrientation
duration:(NSTimeInterval)duration;
- (void)willAnimateRotationToInterfaceOrientation:
(UIInterfaceOrientation)toInterfaceOrientation
duration:(NSTimeInterval)duration;
- (void)didRotateFromInterfaceOrientation:(UIInterfaceOrientation)fromInterfaceOrientation;
```

didReceiveMemoryWarning

Quando la memoria è bassio il view controller notiva che il metodo `didReceiveMemoryWarning` è stato invocato, tutte le risorse (allocate nell'heap) non necessarie e di grandi dimensioni devono essere rilasciate, per fare ciò tutti gli strong pointers devono essere impostati a `nil`.

```
- (void)didReceiveMemoryWarning{
        [super didReceiveMemoryWarning];
        // ...
}
```

awakeFromNib e l'inizializzazione dei view controllers

`init` non viene invocato sugli oggetti inizializzati dalla storyboard, ma viene invocato `awakeFromNib` per ogni oggetto che viene instanziato prima dell'impostazione degli outlets.

> Nota importante
> `awakeFromNib` deve avere del codice che non può essere eseguito altrove.

> Nota importante
> L'inizializzatore designato di `UIViewController` e `awakeFromNib` devono avere lo stesso codice di inizializzazione.

```
- (void)setup{...}
- (void)awakeFromNib{
        [self setup];
}
- (instancetype)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil{
        self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
        [self setup];
        return self;
}
```

> Nota
> È preferibile `viewDidLoad`.

Grafica

UIColor

La classe `UIColor` rappresenta un colore che può essere inizializzato da:

- RGB
- HSB
- Immagini (patterns)

I colori presentanto anche una trasparenza attraveso la proprietà `alpha` che si trova nel range [0 - 1]

> Nota
> Il sistema mette a disposizione dei colori, es. `blackColor, redColor, ecc....`

Fonts

I fonts possono rendere le applicazioni belle da vedere, migliorando così l'esperienza utente.

> Nota
> La scelta corretta del font e della usa dimensione estremamente importante per rendere.

La classe `UIFont` rappresenta i fonts, il miglior modo per prendere un font p chiede al sistema operativo per i font preferiti per ogni tipo stile di test, es `UIFontTextStyleHeadline, UIFontTextStyleBody, ecc...`.

```
UIFont *font = [UIFont preferredFontForTextStyle:UIFontTextStyleHeadline];
```

> Nota
> È possibile creare un font specificandone il nome mediante il metodo `(UIFont *)fontWithName:(NSString *)fontName size:(CGFloat)fontSize`.
> es. `UIFont *font = [UIFont fontWithName:@"HelveticaNeue-Bold" size:15.0];`
>
> Per ottenre la lista dei font disponibili si utilizza `(NSArray *)fontNamesForFamilyName:(NSString *)familyName`

NSAttributedString

L'oggetto `NSAttributedString` gestisce le stringhe di caratte e associati insiemi di attributi applicati ad un singolo carattere o un insieme di caratteri. Il `UIKit framework` aggiunge metodi a `NSAttributedString` per supportare la rappresentazione di stringhe di stile e per calcolare le grandezze e le metriche prima rappresentazione.

> Nota
> `NSAttributedString` è utilizzata per rappresentrare fonts sullo schermo

> Nota importante
> `NSAttributedString` non è una sottoclasse di `NSString`, però è sempre possibile ottenere una stinga da una `NSAttributedString` mediante il metodo `string`

Le `NSAttributedString` sono generalmente create da:

- Una semplice `NSString`
- Un esistente `NSAttributedString`
- Una `NSString` insieme ad una `NSAttributeString`

```
- (id)initWithString:(NSString *)aString
- (id)initWithAttributedString:(NSAttributedString *)attributedString
- (id)initWithString:(NSString *)aString attributes:(NSDictionary *)attributes
```

Esiste anche una versione mutabile delle `NSAttributedString` che è `NSMutableAttributedString`, che permette di cambiare dinamicamente i caratteri e gli attibuti della stringa mediante i metodi:

```
- (void)addAttribute:(NSString *)name value:(id)value range:(NSRange)aRange
- (void)removeAttribute:(NSString *)name range:(NSRange)aRange
- (void)setAttributes:(NSDictionary *)attributes range:(NSRange)aRange
```

Attributi	Tipo
NSFontAttributeName	UIFont*
NSForegroundColorAttributeName	UIColor*
NSBackgroundColorAttributeName	UIColor*
NSStrokeWidthAttributeName	NSNumber*
NSStrokeColorAttributeName	UIColor*

Esempio

```
NSString *string = @"testo piccolo, testo grande!";
NSMutableAttributedString *attrString = [[NSMutableAttributedString alloc]
initWithString:string];

[attrString addAttribute:NSFontAttributeName
        value:[UIFont systemFontOfSize:24.0]
        range:NSMakeRange(15,13)];

[attrString addAttribute:NSForegroundColorAttributeName
        value:[UIColor redColor]
        range:NSMakeRange(6,7)];
```

UIKit Views

La classe `UIView` definisce un area rettangolare sullo schermo e le interfacce per la gestione dei contenuti di quest'area. Le vieste possono contenere altre viste e creare sofisticate gerarchie visive, creando una relazione padre-figlio tra una vista e una sua sottovista.
La geometria di una vista è definita da alcune proprietà:

- `frame`, origine e dimenzioni della vista nelle coordinate del sistema della sua super-vista
- `bounds`, le dimensioni interne dalla vista come quella le vede
- `center`, le coordinate del punto centrale dell'area rettangolare coperta dalla vista.

> Nota
> Tutti gli elementi della UI sono ereditati da `UIView`

Le proprietà più usate:

Property	Value type	Description
`frame`	`CGRect`	Frame rectangle describing the view's location and size in the superview's coordinate system
`bounds`	`CGRect`	Bounds rectangle describing the view's location and size in its own coordinate system
`center`	`CGPoint`	Center of the frame
`backgroundColor`	`UIColor*`	Background color; defaults to `nil` (transparent)
`alpha`	`CGFloat`	0.0 means transparent and 1.0 means opaque
`hidden`	`BOOL`	YES means the view is invisible, NO means visible
`userInteractionEnabled`	`BOOL`	NO means user events are ignored

UILabel

Gli oggetti UILabel sono usati per rappresentare testo statico su un numero di linee fisse.
Le proprietà per `UILabel`:

Property	Value type	Description
`text`	`NSString*`	Text being displayed
`font`	`UIFont*`	Font of the text
`textColor`	`UIColor*`	Color of the text
`textAlignment`	`NSTextAlignment`	Alignment of the text (NSTextAlignmentLeft, NSTextAlignmentRight, NSTextAlignmentCenter...)
`attributedText`	`NSAttributedString*`	Styled text being displayed
`numberOfLines`	`NSInteger`	Maximum number of lines to use

UIKit Controls

Un controllo (control) è un tool di comunicazione tra utente ed applicazione, trasmette una particolare azione/intenzione all'applicazione mediante l'interazioni dell'utente. I controlli possono essere usati per manipolare il contenuto, fornire un input utente, navigare nell'applicazione, o eseguire altre azioni predefinite.

> Nota
> La classe `UIControl` è una sotto-classe di `UIView`, ed è la base per tutti i controlli.

Non viene mai usato direttamente `UIControl`, ma vengono utilizzate le sue sotto-classi come `UIButton` o `UISlide`.

Controlli usabili su iOS
Buttons
Date Pickers
Page Controls
Segmented Controls
Text Fields

Controlli usabili su iOS
Sliders
Steppers
Switches

Gli stati dei controlli

Uno stato di un controllo descrive lo stato interattivo corrente di un controllo, che varia quando l'utente interagisce con il controllo stesso.

Stati
Normale
Selezionato
Disabilitato
Evidenziato

> Nota
> È possibile specificatre un comportamento specifico per ogni stato del controllo

Gli eventi deli controlli

Gli eventi rappresentatno il modo, i gesti fisici, che gli utenti possono fare sui controlli.

Eventi	Descrizione
UIControlEventTouchDown	Tocco all'interno di un controllo
UIControlEventTouchDownRepeat	Tocchi all'interno di un controllo ripetuti
UIControlEventTouchDragInside	Un dito è trascinato all'interno dei confini di un controllo
UIControlEventTouchDragOutside	Un dito è trascinato all'esterno dei confini di un controllo
UIControlEventTouchDragEnter	Un dito è trascinato dall'esterno all'interno dei confini di un controllo
UIControlEventTouchDragExit	Un dito è trascinato dall'interno all'esterno dei confini di un controllo
UIControlEventTouchUpInside	Un dito è sollevato mentre si trova all'interno dei confini di un controllo
UIControlEventTouchUpOutside	Un dito è sollevato mentre si trova all'esterno dei confini di un controllo
UIControlEventTouchCancel	Eventop di sistema che cancella il tocco corrente per il controllo
UIControlEventValueChanged	Tocco, trascinamento o altro gestiscono il controllo, causando una serie di valori differenti

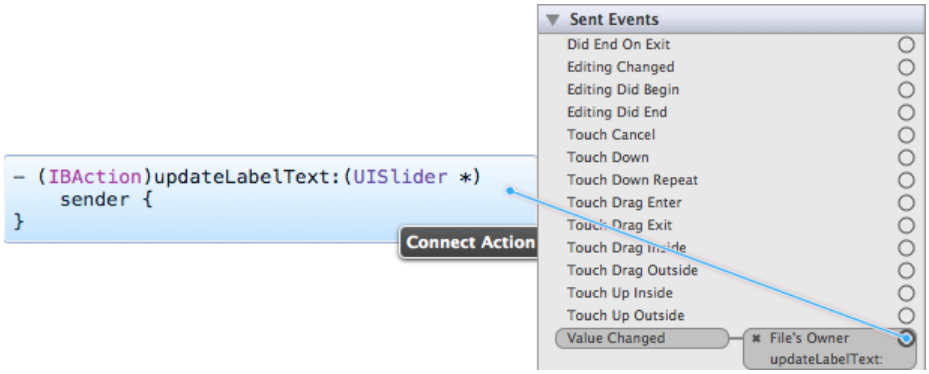
Target-Action

Il meccanismo target-action è un modello per configurare un controllo per mandare un messaggio di azione ad un obbiettivo specifico dopo uno specifico evento di un controllo.

Target-Action binding

Modi per legare un target-action ad un evento di un controllo:

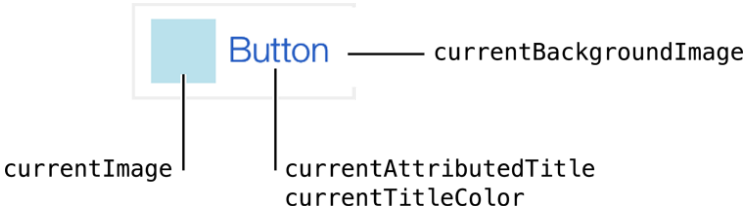
- Programmaticamente `[self.mySlider addTarget:self action:@selector(myAction:) forControlEvents:UIControlEventValueChanged];`.
- Con il `Connection Inspector` nel `Interface Builder`.



- Mediante il `Interface builder`, selezionando lo slider all'terno dell'interfaccia e trascinando il suo `Value Changed event` all'oggetto target nello storyboard e selezionare un azione appropriata dalla lista delle azioni disponibili per il target.

UIButton

I bottoni mostrano immagini o testo, e permettono all'utente di avviare un comportamento con un tocco. Quando un utente tocca un bottone, questo cambia il suo stato ad evidenziato e modifica il suo aspetto (per comunicare il cambiamento).



È possibile impsotare titolo, immagine, imamgine di sfondo del bottone per ogni stato in 2 modi:

- Mediante l'`attribute inspector`
- Programmaticamente:

```
- (void)setTitle:(NSString *)title forState:(UIControlState)state
- (void)setAttributedTitle:(NSAttributedString *)title forState:(UIControlState)state
- (void)setImage:(UIImage *)image forState:(UIControlState)state
- (void)setBackgroundImage:(UIImage *)image forState:(UIControlState)state
```

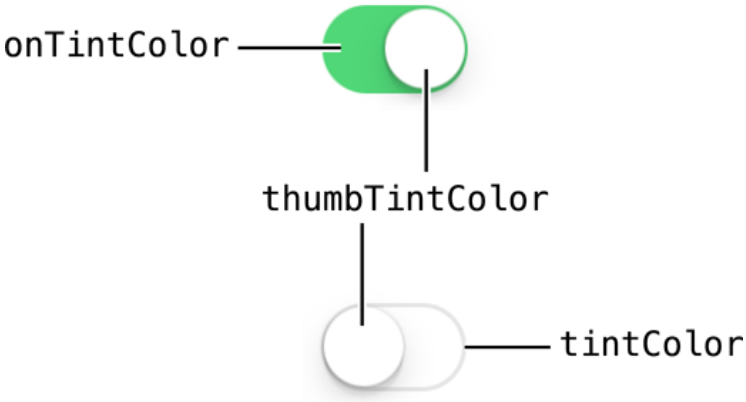
UISlider

Gli slider permettono all'utente di modificare interattivamente alcuni valori di un applicazione. I valori possono presentare un massimo, un minimo ed un valore di default, che sono rispettivamente `minimumValue`(default 0), `maximumValue` (default 1) e `value`(default 0.5).

> Nota
> È possibile personalizzare lo slider con le proprietà:
>
> - maximumTrackTintColor
> - thumbTintColor
> - minimumTrackTintColor

UISwitch

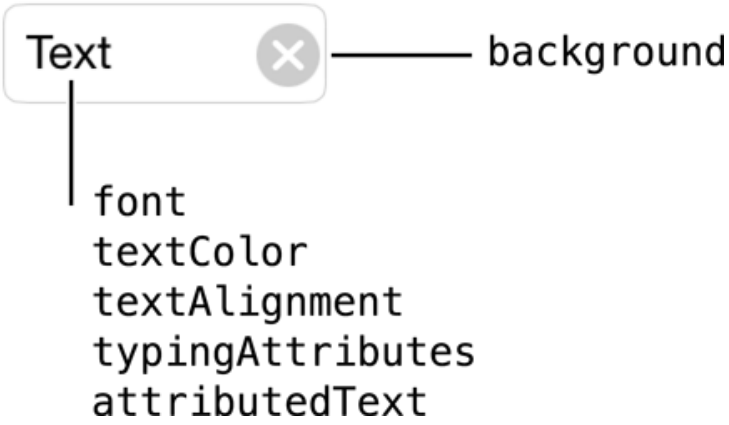
Uno switch permette all'utenre un opzione on/off. È possibile personalizzare l'aspetto dello switch mediante le sue proprietà



UITextField

Il campo testuale permette all'utenre di inserire una singola riga di testo nell'applicazione.

Le proprietà:
Proprietà	Utilizzo
`text`, `attributedText`	Impostare/leggere il contenuto dell'input di testo
`placeholder`, `attributePlaceholder`	Impostare un placeholder per il testo
`font`, `texsAlignment`, `textColor`	Impostare lo stile del testo
`clearButtonMode` con `UITextFieldViewMode`	Impostare se il bottone sulla destra deve essere mostrato oppure no
`UITextInputTraits`	Impostare lo stile e il layout della tastiera



Schiacciando un `UITextField` comporta la comparsa della tastiera, scivolando dalla parte sotto dello schermo e coprendo una certa area. Così facendo la tastirea diventa la prima a rispondere ("first responder").

> Nota
> È necessario gestire in modo appropriato la comparsa della tastiera.

> Nota
> È necessario fare in modo che la tastira scompaia, dato che cliccando su "fatto" non scompare, per fare ciò si può impostare la vista di sfondo come `UIController` invece di `UIView`, così può ricevere eventi. Successivamente si può definire un `IBAction` che viene invocata quando l'utente clicca all'interno, e sfruttando il target-action si invoca il metodo `[textField resignFirstResponder];` che rimuove la tastiera.

UITextView

`UITextView` è modo più potente per rappresentare il testo, permette:

- Più righe
- Modificare e selezionare
- Scorrere

È possibile personalizzare un `UITextView` mediante le sue proprietà

Property	Value type	Description
`text`	`NSString*`	Text being displayed
`font`	`UIFont*`	Font of the text
`textColor`	`UIColor*`	Color of the text
`textAlignment`	`NSTextAlignment`	Alignment of the text (NSTextAlignmentLeft, NSTextAlignmentRight, NSTextAlignmentCenter...)
`attributedText`	`NSAttributedString*`	Styled text being displayed
`textStorage`	`NSTextStorage`	Efficient text manipulation
`editable`	`BOOL`	Whether the receiver is editable
`selectable`	`BOOL`	Whether the receiver is selectable
`selectedRange`	`NSRange`	Current selection range in the text view

UITextViewDelegate

È presente un protocollo che definisce un insieme di metodi per ricevere messaggi relativi allo modifica di un certo `UITextView`.

```
- (BOOL)textViewShouldBeginEditing:(UITextView *)textView
- (BOOL)textViewShouldEndEditing:(UITextView *)textView
- (void)textViewDidBeginEditing:(UITextView *)textView
- (void)textViewDidChange:(UITextView *)textView
- (void)textViewDidChangeSelection:(UITextView *)textView
- (void)textViewDidEndEditing:(UITextView *)textView
```

Notifiche

iOs fornisce un altro modo per interagire con gli oggetti, oltre allo scambio di messaggi. Le notifiche sono un modo standard con il quale è possibile notificare il controller di un determinato evento. La classe se fornisce questo meccanismo di trasmissione di informazioni è `NSNotificationCenter`

> Nota
> Un riferimento a `NSNotificationCenter` avviene come segue `NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];`

Ogni programma ha il suo `NSNotificationCenter`, quindi non è necessario crearne uno.

Gli oggetti possono memorizzare, mediante un centro notifiche, le notifiche ricevute e eseguire particolari metodi. È importante però cancellare le notifiche non più necessarie, perché possono causare un crash.

esempio registrazione

`metodo`

```
- (void)addObserver:(id)notificationObserver
selector:(SEL)notificationSelector
name:(NSString *)notificationName
object:(id)notificationSender
```

`instanzia`

```
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(aMethod)
name:UIKeyboardWillShowNotification
object:nil];
```

Esempio cancellazione notifica

`metodo`

```
- (void)removeObserver:(id)notificationObserver
name:(NSString *)notificationName
object:(id)notificationSender
```

`instanza`

```
[[NSNotificationCenter defaultCenter] removeObserver:self
name:UIKeyboardWillShowNotification
object:nil];
```

Esempio generazione notifica

`metodo`

```
- (void)postNotificationName:(NSString *)notificationName
object:(id)notificationSender
userInfo:(NSDictionary *)userInfo
```

`instanza`

```
[[NSNotificationCenter defaultCenter] postNotificationName:@"MyNotification"
object:self
userInfo:nil];
```

Quando il sistema mostra/nasconde la tastiera genera diverse notifiche, queste contengono informazioni sulla tastiera (tipo la dimensione), quindi registrare informazioni per questo tipo di notifiche è l'unico modo per reperire questo tipo di informazioni.

Notifiche di sistema
`UIKeyboardWillShowNotification`
`UIKeyboardDidShowNotification`
`UIKeyboardWillHideNotification`
`UIKeyboardDidHideNotification`