概述

这篇文章，我将讲述几种转场动画的自定义方式，并且每种方式附上一个示例，毕竟代码才是我们的语言，这样比较容易上手。其中主要有以下三种自定义方法，供大家参考：

- **Push & Pop**
- **Modal**
- **Segue**

前两种大家都很熟悉，第三种是 Stroyboard 中的拖线，属于 UIStoryboardSegue 类，通过继承这个类来自定义转场过程动画。

Push & Pop

首先说一下 Push & Pop 这种转场的自定义，操作步骤如下：

1. 创建一个文件继承自 NSObject, 并遵守 UIViewControllerAnimatedTransitioning协议。

2. 实现该协议的两个基本方法：

```
//指定转场动画持续的时长
func transitionDuration(transitionContext: UIViewControllerContextTransitioning) -> NSTi
//转场动画的具体内容
func animateTransition(transitionContext: UIViewControllerContextTransitioning)
```

3. 遵守 UINavigationControllerDelegate 协议，并实现此方法：

```
 func navigationController(navigationController: UINavigationController, animationContro
```

在此方法中指定所用的 UIViewControllerAnimatedTransitioning，即返回 第1步 中创建的类。

注意：由于需要 Push 和 Pop，所以需要两套动画方案。解决方法为：

在 第1步 中，创建两个文件，一个用于 Push 动画，一个用于 Pop动画，然后 第3步 中在返回动画类之前，先判断动画方式（Push 或 Pop），使用 operation == UINavigationControllerOperation.Push 即可判断，最后根据不同的方式返回不同的类。

到这里就可以看到转场动画的效果了，但是大家都知道，系统默认的 **Push** 和 **Pop** 动画都支持手势驱动，并且可以根据手势移动距离改变动画完成度。幸运的是，**Cocoa** 已经集成了相关方法，我们只用告诉它百分比就可以了。所以下一步就是 手势驱动。

4. 在第二个 UIViewController 中给 View 添加一个滑动（Pan）手势。

创建一个 UIPercentDrivenInteractiveTransition 属性。

在手势的监听方法中计算手势移动的百分比，并使用 UIPercentDrivenInteractiveTransition 属性的 updateInteractiveTransition() 方法实时更新百分比。

最后在手势的 state 为 ended 或 cancelled 时，根据手势完成度决定是还原动画还是结束动画，使用 UIPercentDrivenInteractiveTransition 属性的 cancelInteractiveTransition() 或 finishInteractiveTransition() 方法。

5. 实现 UINavigationControllerDelegate 中的另一个返回 UIViewControllerInteractiveTransitioning 的方法，并在其中返回 第4步 创建的 UIPercentDrivenInteractiveTransition 属性。

至此，Push 和 Pop 方式的自定义就完成了，具体细节看下面的示例。

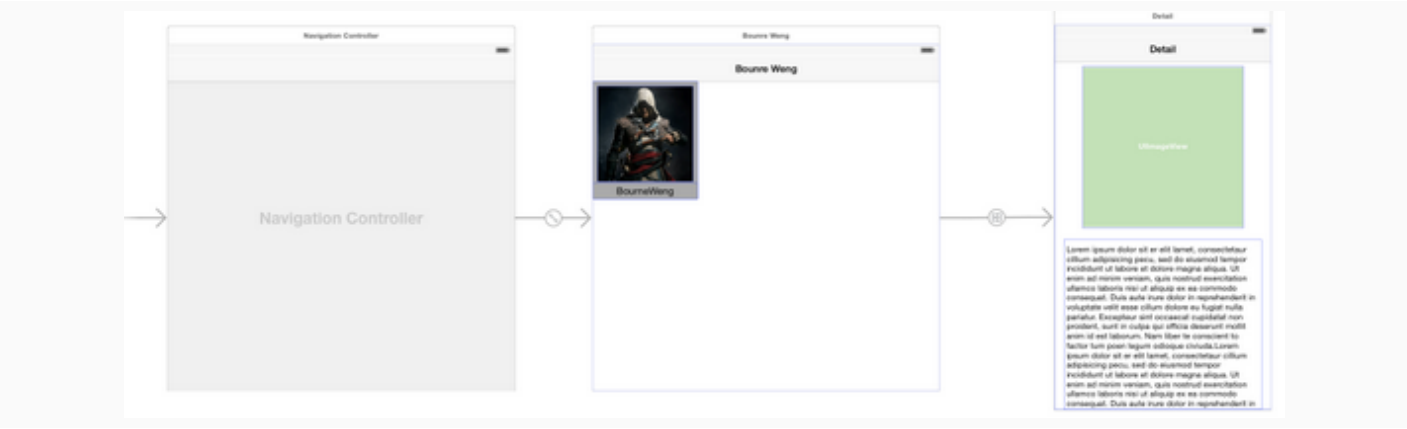
自定义 Push & Pop 示例

此示例来自 Kitten Yang 的blog 实现Keynote中的神奇移动效果，我将其用 **Swift** 实现了一遍，源代码地址:MagicMove，下面是运行效果。

初始化

- 创建两个 ViewController，一个继承自 UICollectionViewController，取名 ViewController。另一个继承 UIViewController，取名 DetailViewController。在 Stroyboard 中创建并绑定。

- 在 Stroyboard 中拖一个 UINavigationController，删去默认的 rootViewController，使 ViewController 作为其 rootViewController，再拖一条从 ViewController 到 DetailViewController 的 segue。
- 在 ViewController 中自定义 UICollectionViewCell，添加 UIImageView 和 UILabel。
- 在 DetailViewController 中添加 UIImageView 和 UITextView



添加 UIViewControllerAnimatedTransitioning

- 添加一个 Cocoa Touch Class，继承自 NSObject，取名 MagicMoveTransion，遵守 UIViewControllerAnimatedTransitioning 协议。
- 实现协议的两个方法，并在其中编写 Push 的动画。 具体的动画实现过程都在代码的注释里：

```
func transitionDuration(transitionContext: UIViewControllerContextTransitioning) -> NST
    return 0.5
}

func animateTransition(transitionContext: UIViewControllerContextTransitioning) {
    //1.获取动画的源控制器和目标控制器
    let fromVC = transitionContext.viewControllerForKey(UITransitionContextFromViewCont
    let toVC = transitionContext.viewControllerForKey(UITransitionContextToViewControll
    let container = transitionContext.containerView()

    //2.创建一个 Cell 中 imageView 的截图，并把 imageView 隐藏，造成使用户以为移动的就是 ima
    let snapshotView = fromVC.selectedCell.imageView.snapshotViewAfterScreenUpdates(fal
    snapshotView.frame = container.convertRect(fromVC.selectedCell.imageView.frame, fro
    fromVC.selectedCell.imageView.hidden = true

    //3.设置目标控制器的位置，并把透明度设为0，在后面的动画中慢慢显示出来变为1
    toVC.view.frame = transitionContext.finalFrameForViewController(toVC)
    toVC.view.alpha = 0

    //4.都添加到 container 中。注意顺序不能错了
    container.addSubview(toVC.view)
    container.addSubview(snapshotView)

    //5.执行动画
    UIView.animateWithDuration(transitionDuration(transitionContext), delay: 0, options
            snapshotView.frame = toVC.avatarImageView.frame
            toVC.view.alpha = 1
        }) { (finish: Bool) -> Void in
            fromVC.selectedCell.imageView.hidden = false
            toVC.avatarImageView.image = toVC.image
            snapshotView.removeFromSuperview()

            //一定要记得动画完成后执行此方法，让系统管理 navigation
            transitionContext.completeTransition(true)
    }
}
```

使用动画

- 让 ViewController 遵守 UINavigationControllerDelegate 协议。
- 在 ViewController 中设置 NavigationController 的代理为自己：

```
override func viewDidAppear(animated: Bool) {
    super.viewDidAppear(animated)

    self.navigationController?.delegate = self
}
```

- 实现 UINavigationControllerDelegate 协议方法:

```
func navigationController(navigationController: UINavigationController, animationControl
    if operation == UINavigationControllerOperation.Push {
        return MagicMoveTransion()
    } else {
        return nil
    }
}
```

- 在 ViewController 的 controllerCell 的点击方法中，发送 segue

```
override func collectionView(collectionView: UICollectionView, didSelectItemAtIndexPath
    self.selectedCell = collectionView.cellForItemAtIndexPath(indexPath) as! MMCollectic

    self.performSegueWithIdentifier("detail", sender: nil)
}
```

- 在发送 segue 的时候，把点击的 image 发送给 DetailViewController

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "detail" {
        let detailVC = segue.destinationViewController as! DetailViewController
        detailVC.image = self.selectedCell.imageView.image
    }
}
```

至此，在点击 **Cell** 后，就会执行刚刚自定义的动画了。接下来就要加入手势驱动。

手势驱动

- 在 DetailViewController 的 ViewDidAppear() 方法中，加入滑动手势。

```
 let edgePan = UIScreenEdgePanGestureRecognizer(target: self, action: Selector("edgePanG
    edgePan.edges = UIRectEdge.Left
    self.view.addGestureRecognizer(edgePan)
```

- 在手势监听方法中，创建 UIPercentDrivenInteractiveTransition 属性，并实现手势百分比更新。

```
func edgePanGesture(edgePan: UIScreenEdgePanGestureRecognizer) {
    let progress = edgePan.translationInView(self.view).x / self.view.bounds.width

    if edgePan.state == UIGestureRecognizerState.Began {
```

```
            self.percentDrivenTransition = UIPercentDrivenInteractiveTransition()
            self.navigationController?.popViewControllerAnimated(true)
        } else if edgePan.state == UIGestureRecognizerState.Changed {
            self.percentDrivenTransition?.updateInteractiveTransition(progress)
        } else if edgePan.state == UIGestureRecognizerState.Cancelled || edgePan.state == U
            if progress > 0.5 {
                self.percentDrivenTransition?.finishInteractiveTransition()
            } else {
                self.percentDrivenTransition?.cancelInteractiveTransition()
            }
            self.percentDrivenTransition = nil
        }
    }
```

- 实现返回 UIViewControllerInteractiveTransitioning 的方法并返回刚刚创建的 UIPercentDrivenInteractiveTransition属性。

```
    func navigationController(navigationController: UINavigationController, interactionCo
     if animationController is MagicMovePopTransion {
         return self.percentDrivenTransition
     } else {
         return nil
     }
}
```

OK，到现在，手势驱动就写好了，但是还不能使用，因为还没有实现 **Pop** 方法！现在自己去实现 **Pop** 动画吧！请参考源代码：**MagicMove**

Modal

modal转场方式即使用 presentViewController() 方法推出的方式，默认情况下，第二个视图从屏幕下方弹出。下面就来介绍下 modal 方式转场动画的自定义。

1. 创建一个文件继承自 NSObject, 并遵守 UIViewControllerAnimatedTransitioning协议。

2. 实现该协议的两个基本方法：

```
//指定转场动画持续的时长
func transitionDuration(transitionContext: UIViewControllerContextTransitioning) -> NSTi
//转场动画的具体内容
func animateTransition(transitionContext: UIViewControllerContextTransitioning)
```

以上两个步骤和 **Push & Pop** 的自定义一样，接下来就是不同的。

3. 如果使用 Modal 方式从一个 VC 到另一个 VC，那么需要第一个 VC 遵循 UIViewControllerTransitioningDelegate 协议，并实现以下两个协议方法：

```
//present动画
optional func animationControllerForPresentedController(presented: UIViewController, pr

//dismiss动画
optional func animationControllerForDismissedController(dismissed: UIViewController) ->
```

4. 在第一个 VC 的 prepareForSegue() 方法中，指定第二个 VC 的 transitioningDelegate 为 self。

由 第3步 中两个方法就可以知道，在创建转场动画时，最好也创建两个动画类，一个用于 **Present**， 一个用于 **Dismiss**，如果只创建一个动画类，就需要在实现动画的时候判断是 **Present** 还是 **Dismiss**。

这时，转场动画就可以实现了，接下来就手势驱动了

5. 在第一个 VC 中创建一个 UIPercentDrivenInteractiveTransition 属性，并且在 prepareForSegue() 方法中为第二个 VC.view 添加一个手势，用以 dismiss. 在手势的监听方法中处理方式和 Push & Pop 相同。

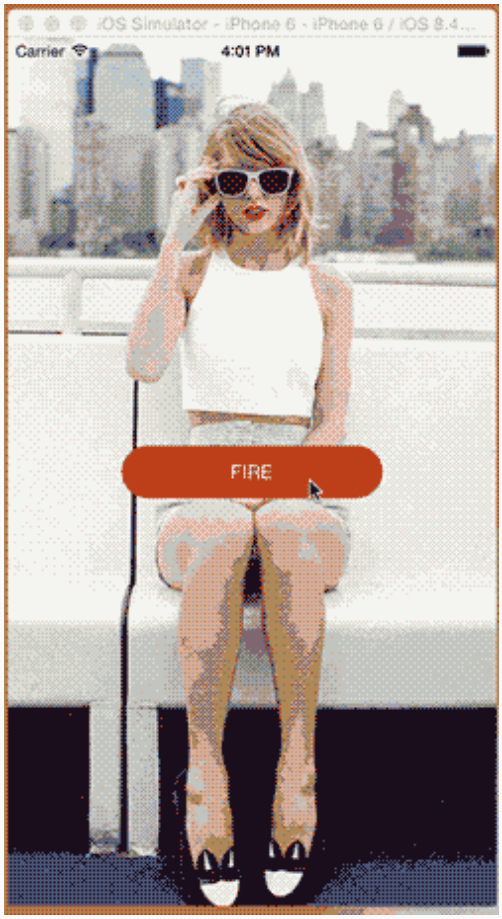
6. 实现 UIViewControllerTransitioningDelegate 协议的另外两个方法，分别返回 Present 和 Dismiss 动画的百分比。

```
//百分比Push
func interactionControllerForPresentation(animator: UIViewControllerAnimatedTransitioni
    return self.percentDrivenTransition
}
//百分比Pop
func interactionControllerForDismissal(animator: UIViewControllerAnimatedTransitioning)
    return self.percentDrivenTransition
}
```

至此，**Modal** 方式的自定义转场动画就写完了。自己在编码的时候有一些小细节需要注意，下面将展示使用 **Modal** 方式的自定义动画的示例。

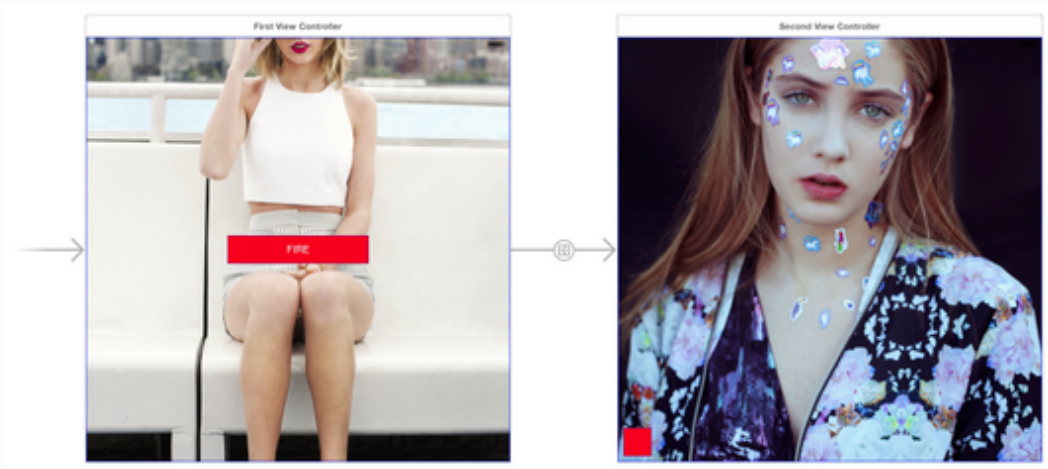
自定义 Modal 示例

此示例和上面一个示例一样，来自 Kitten Yang 的blog 实现3D翻转效果，我也将其用 **Swift** 实现了一遍，同样我的源代码地址：FlipTransion，运行效果如下：



初始化

- 创建两个 UIViewController, 分别命名为：FirstViewController 和 SecondViewController。并在 Storyboard 中添加两个 UIViewController 并绑定。
- 分别给两个视图添加两个 UIImageView，这样做的目的是为了区分两个控制器。当然你也可以给两个控制器设置不同的背景，总之你开心就好。但是，既然做，就做认真点呗。注意：如果使用图片并设置为 Aspect Fill 或者其他的 Fill，一定记得调用 imageView 的 clipsToBounds() 方法裁剪去多余的部分。
- 分别给两个控制器添加两个按钮，第一个按钮拖线到第二个控制器，第二个控制器绑定一个方法用来dismiss。



添加 UIViewControllerAnimatedTransitioning

- 添加一个 Cocoa Touch Class，继承自 NSObject，取名 BWFlipTransionPush（名字嘛，你开心就好。），遵守 UIViewControllerAnimatedTransitioning 协议。
- 实现协议的两个方法，并在其中编写 Push 的动画。 具体的动画实现过程都在代码的注释里：

```
func animateTransition(transitionContext: UIViewControllerContextTransitioning) {
    let fromVC = transitionContext.viewControllerForKey(UITransitionContextFromViewCont
    let toVC = transitionContext.viewControllerForKey(UITransitionContextToViewControll
    let container = transitionContext.containerView()
    container.addSubview(toVC.view)
    container.bringSubviewToFront(fromVC.view)

    //改变m34
    var transfrom = CATransform3DIdentity
    transfrom.m34 = -0.002
    container.layer.sublayerTransform = transfrom

    //设置anrchPoint 和 position
    let initalFrame = transitionContext.initialFrameForViewController(fromVC)
    toVC.view.frame = initalFrame
    fromVC.view.frame = initalFrame
    fromVC.view.layer.anchorPoint = CGPointMake(0, 0.5)
    fromVC.view.layer.position = CGPointMake(0, initalFrame.height / 2.0)

    //添加阴影效果
    let shadowLayer = CAGradientLayer()
    shadowLayer.colors = [UIColor(white: 0, alpha: 1).CGColor, UIColor(white: 0, alpha:
    shadowLayer.startPoint = CGPointMake(0, 0.5)
    shadowLayer.endPoint = CGPointMake(1, 0.5)
    shadowLayer.frame = initalFrame
    let shadow = UIView(frame: initalFrame)
    shadow.backgroundColor = UIColor.clearColor()
    shadow.layer.addSublayer(shadowLayer)
    fromVC.view.addSubview(shadow)
    shadow.alpha = 0

    //动画
    UIView.animateWithDuration(transitionDuration(transitionContext), delay: 0, options
            fromVC.view.layer.transform = CATransform3DMakeRotation(CGFloat(-M_PI_2), 0
            shadow.alpha = 1.0
        }) { (finished: Bool) -> Void in
            fromVC.view.layer.anchorPoint = CGPointMake(0.5, 0.5)
            fromVC.view.layer.position = CGPointMake(CGRectGetMidX(initalFrame), CGRect
            fromVC.view.layer.transform = CATransform3DIdentity
            shadow.removeFromSuperview()

            transitionContext.completeTransition(!transitionContext.transitionWasCancel
    }
}
```

动画的过程我就不多说了，仔细看就会明白。

使用动画

- 让 FirstViewController 遵守 UIViewControllerTransitioningDelegate 协议，并将 self.transitioningDelegate 设置为 self。
- 实现 UIViewControllerTransitioningDelegate 协议的两个方法，用来指定动画类。

```
//动画Push
func animationControllerForPresentedController(presented: UIViewController, presentingCo
    return BWFlipTransionPush()
}
```

```
//动画Pop
func animationControllerForDismissedController(dismissed: UIViewController) -> UIViewCor
    return BWFlipTransionPop()
}
```

OK，如果你完成了**Pop**动画，那么现在就可以实现自定义 **Modal** 转场了。现在只差手势驱动了。

手势驱动

- 想要同时实现 Push 和 Pop 手势，就需要给两个 viewController.view 添加手势。首先在 FirstViewController 中给自己添加一个屏幕右边的手势，在 prepareForSegue() 方法中给 SecondViewController.view 添加一个屏幕左边的手势，让它们使用同一个手势监听方法。
- 实现监听方法，不多说，和之前一样，但还是有仔细看，因为本示例中转场动画比较特殊，而且有两个手势，所以这里计算百分比使用的是 KeyWindow。同时不要忘了：UIPercentDrivenInteractiveTransition属性。

```
func edgePanGesture(edgePan: UIScreenEdgePanGestureRecognizer) {
    let progress = abs(edgePan.translationInView(UIApplication.sharedApplication().keyW

    if edgePan.state == UIGestureRecognizerState.Began {
        self.percentDrivenTransition = UIPercentDrivenInteractiveTransition()
        if edgePan.edges == UIRectEdge.Right {
            self.performSegueWithIdentifier("present", sender: nil)
        } else if edgePan.edges == UIRectEdge.Left {
            self.dismissViewControllerAnimated(true, completion: nil)
        }
    } else if edgePan.state == UIGestureRecognizerState.Changed {
        self.percentDrivenTransition?.updateInteractiveTransition(progress)
    } else if edgePan.state == UIGestureRecognizerState.Cancelled || edgePan.state == U
        if progress > 0.5 {
            self.percentDrivenTransition?.finishInteractiveTransition()
        } else {
            self.percentDrivenTransition?.cancelInteractiveTransition()
        }
        self.percentDrivenTransition = nil
    }
}
```

- 实现 UIViewControllerTransitioningDelegate 协议的另外两个方法，分别返回 Present 和 Dismiss 动画的百分比。

```
//百分比Push
func interactionControllerForPresentation(animator: UIViewControllerAnimatedTransitionir
    return self.percentDrivenTransition
}
//百分比Pop
func interactionControllerForDismissal(animator: UIViewControllerAnimatedTransitioning)
    return self.percentDrivenTransition
}
```

现在，基于 **Modal** 的自定义转场动画示例就完成了。获取完整源代码：**FlipTransion**

Segue

这种方法比较特殊，是将 Stroyboard 中的拖线与自定义的 UIStoryboardSegue 类绑定自实现定义转场过程动画。

首先我们来看看 UIStoryboardSegue 是什么样的。

```
@availability(iOS, introduced=5.0)
class UIStoryboardSegue : NSObject {

    // Convenience constructor for returning a segue that performs a handler block in i
    @availability(iOS, introduced=6.0)
    convenience init(identifier: String?, source: UIViewController, destination: UIVie

    init!(identifier: String?, source: UIViewController, destination: UIViewController)

    var identifier: String? { get }
    var sourceViewController: AnyObject { get }
    var destinationViewController: AnyObject { get }

    func perform()
}
```

以上是 UIStoryboardSegue 类的定义。从中可以看出，只有一个方法 perform()，所以很明显，就是重写这个方法来自定义转场动画。

再注意它的其他属性：sourceViewController 和 destinationViewController，通过这两个属性，我们就可以访问一个转场动画中的两个主角了，于是自定义动画就可以随心所欲了。

只有一点需要注意：在拖线的时候，注意在弹出的选项中选择 custom。然后就可以和自定义的 UIStoryboardSegue 绑定了。

那么，问题来了，这里只有 perform，那 返回时的动画怎么办呢？请往下看：

Dismiss

由于 perfrom 的方法叫做：segue，那么返回转场的上一个控制器叫做： **unwind segue**

- 解除转场（unwind segue）通常和正常自定义转场（segue）一起出现。
- 要解除转场起作用，我们必须重写perform方法，并应用自定义动画。另外，导航返回源视图控制器的过渡效果不需要和对应的正常转场相同。

其 **实现步骤** 为：

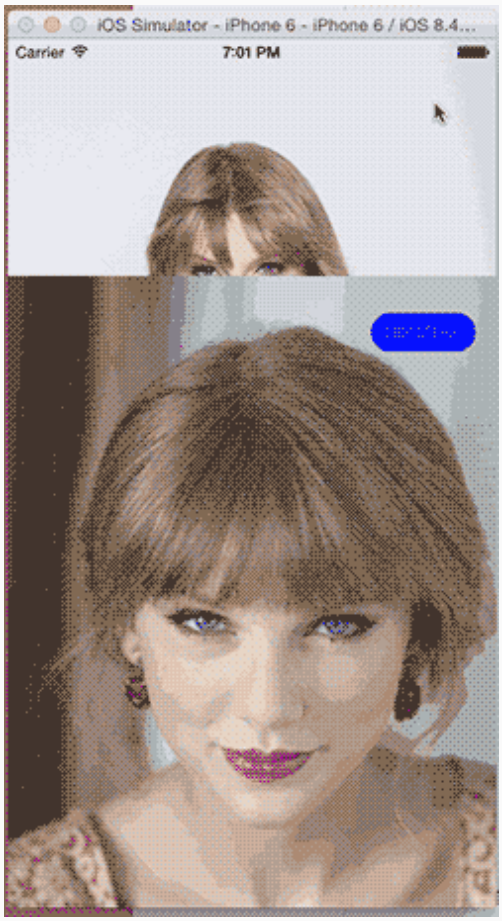
- 创建一个 IBAction 方法，该方法在解除转场被执行的时候会选择地执行一些代码。这个方法可以有你想要的任何名字，而且不强制包含其它东西。它需要定义，但可以留空，解除转场的定义需要依赖这个方法。
- 解除转场的创建，设置的配置。这和之前的转场创建不太一样，等下我们将看看这个是怎么实现的。
- 通过重写 UIStoryboardSegue 子类里的 perform() 方法，来实现自定义动画。
- UIViewController类 提供了特定方法的定义，所以系统知道解除转场即将执行。

当然，这么说有一些让人琢磨不透，不知道什么意思。那么，下面再通过一个示例来深入了解一下。

Segue 示例

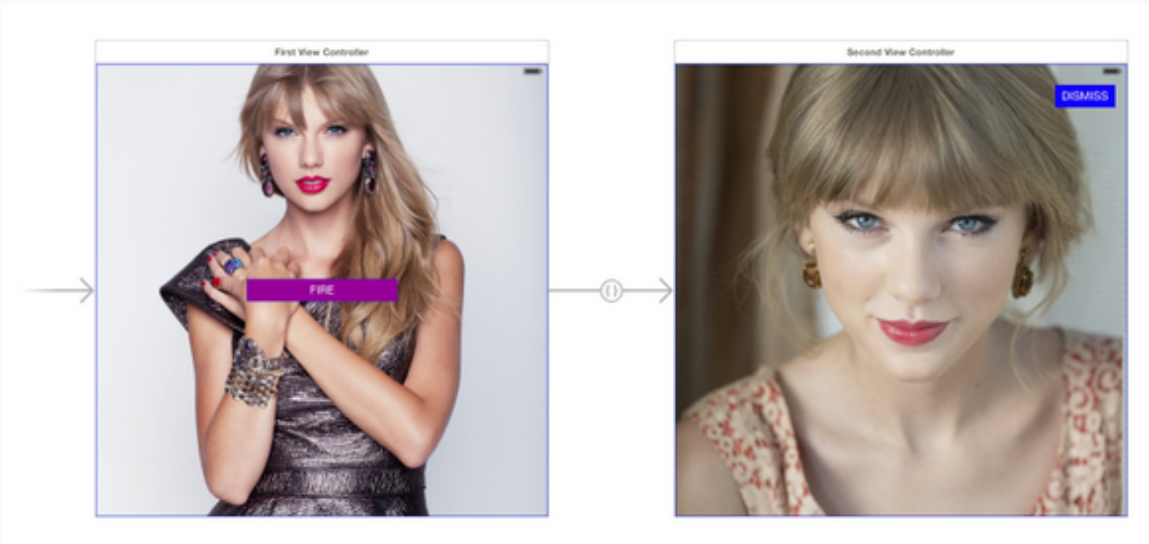
这个示例是我自己写的，源代码地址：SegueTransion，开门见山，直接上图。

GIF演示



初始化

- 创建两个 UIViewController, 分别命名为：FirstViewController 和 SecondViewController。并在 Storyboard 中添加两个 UIViewController 并绑定。
- 分别给两个控制器添加背景图片或使用不同的背景色，用以区分。在 FirstViewController 中添加一个触发按钮，并拖线到 SecondViewController 中，在弹出的选项中选择 custion。



Present

- 添加一个 Cocoa Touch Class，继承自 UIStoryboardSegue，取名 FirstSegue（名字请随意）。并将其绑定到上一步中拖拽的 segue 上。
- 重写 FirstSegue 中的 perform() 方法，在其中编写动画逻辑。

```
override func perform() {
    var firstVCView = self.sourceViewController.view as UIView!
    var secondVCView = self.destinationViewController.view as UIView!

    let screenWidth = UIScreen.mainScreen().bounds.size.width
    let screenHeight = UIScreen.mainScreen().bounds.size.height

    secondVCView.frame = CGRectMake(0.0, screenHeight, screenWidth, screenHeight)
    let window = UIApplication.sharedApplication().keyWindow
    window?.insertSubview(secondVCView, aboveSubview: firstVCView)

    UIView.animateWithDuration(0.5, delay: 0, usingSpringWithDamping: 0.5, initialSpr
            secondVCView.frame = CGRectOffset(secondVCView.frame, 0.0, -screenHeight)
        }) { (finished: Bool) -> Void in
            self.sourceViewController.presentViewController(self.destinationViewContr
                animated: false,
                completion: nil)
    }
}
```

还是一样，动画的过程自己看，都是很简单的。

Present手势

这里需要注意，使用这种方式自定义的转场动画不能动态手势驱动，也就是说不能根据手势百分比动态改变动画完成度。

所以，这里只是简单的添加一个滑动手势（swip）。

- 在 FisrtViewController 中添加手势：

```
var swipeGestureRecognizer: UISwipeGestureRecognizer = UISwipeGestureRecognizer(target:
```

```
    swipeGestureRecognizer.direction = UISwipeGestureRecognizerDirection.Up
    self.view.addGestureRecognizer(swipeGestureRecognizer)
```

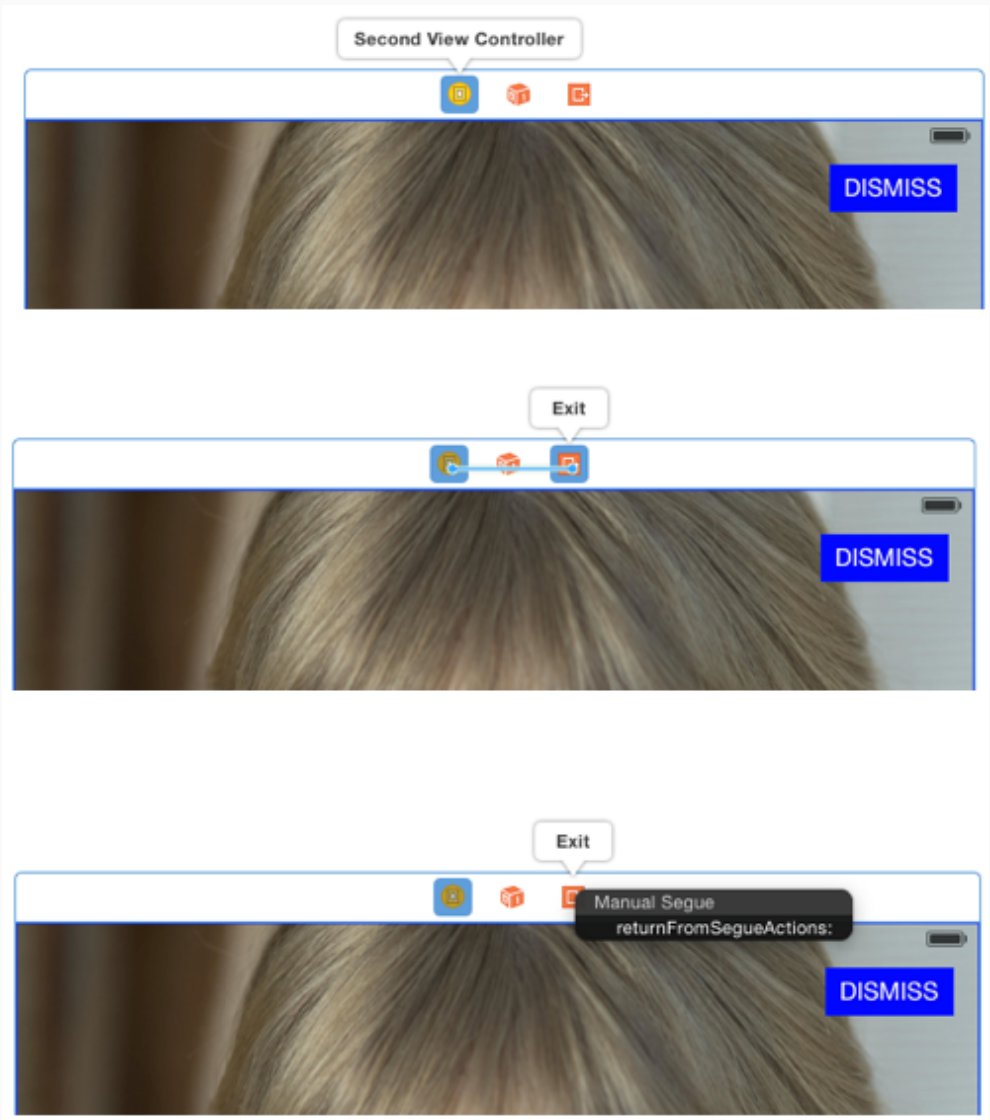
- 实现手势监听方法：

```
func showSecondViewController() {
    self.performSegueWithIdentifier("idFirstSegue", sender: self)
}
```

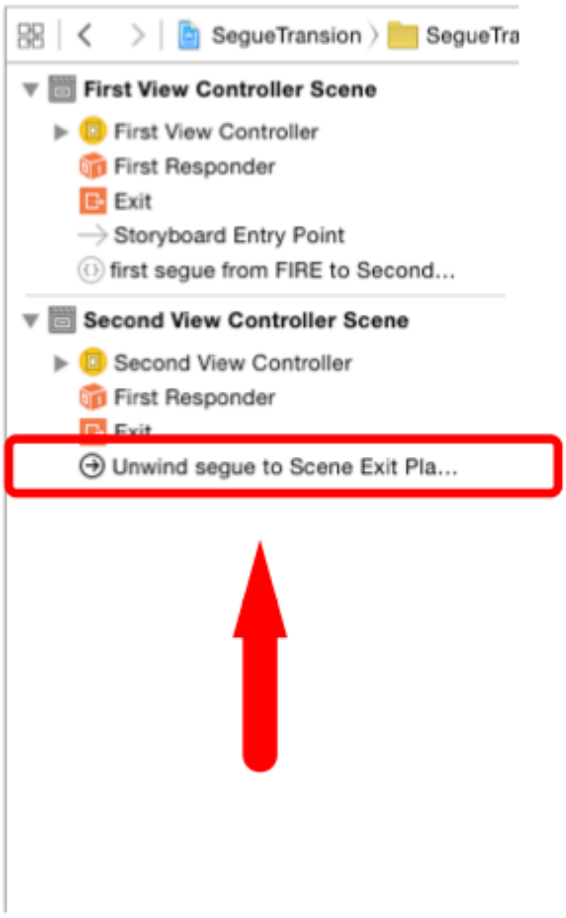
现在已经可以 **present** 了，接下来实现 **dismiss**。

Dismiss

- 在 FirstViewController 中添加一个 IBAction 方法，方法名可以随便，有没有返回值都随便。
- 在 Storyboard 中选择 SecondViewController 按住 control键 拖线到 SecondViewController 的 Exit 图标。并在弹出选项中选择上一步添加 IBAction 的方法。



- 在 Storyboard 左侧的文档视图中找到上一步拖的 segue，并设置 identifier



- 再添加一个 Cocoa Touch Class，继承自 UIStoryboardSegue，取名 FirstSegueUnWind（名字请随意）。并重写其 perform() 方法，用来实现 dismiss 动画。
- 在 FirstViewController 中重写下面方法。并根据 identifier 判断是不是需要 dismiss，如果是就返回刚刚创建的 FirstUnWindSegue。

```
override func segueForUnwindingToViewController(toViewController: UIViewController, from

    if identifier == "firstSegueUnwind" {
        return FirstUnwindSegue(identifier: identifier, source: fromViewController, dest
        })
    }

    return super.segueForUnwindingToViewController(toViewController, fromViewController:
}
```

- 最后一步，在 SecondViewController 的按钮的监听方法中实现 dismiss， 注意不是调用 self.dismiss...!

```
  @IBAction func shouldDismiss(sender: AnyObject) {
      self.performSegueWithIdentifier("firstSegueUnwind", sender: self)
  }
```

给 **SecondViewController** 添加手势，将手势监听方法也设置为以上这个方法, 参考代码：**SegueTransion**。

总结

一张图总结一下3种方法的异同点。

	Push & Pop	Modal	Segue
动画写在哪里	创建文件，遵守 UIViewControllerAnimatedTransitioning 协议，实现其 transitionDuration 和 animateTransition 方法。	创建文件，遵守 UIViewControllerAnimatedTransitioning 协议，实现其 transitionDuration 和 animateTransition 方法。	自定义 UIStoryboardSegue 类，与 storyboard 中转场segue绑定，重写其 perform() 方法
如果调用	遵守 UINavigationControllerDelegate 协议，实现 animationControllerForOperation 方法，根据 push 或 pop 选择返回不同的动画类	遵守 UIViewControllerTransitioningDelegate 协议，实现 animationControllerForPresentedController 和 animationControllerForDismissedController 方法，分别返回不同的动画类	present 和 dismiss 分别有一个 segue 表示，重写各自的 perform() 方法即可
手势动态驱动	创建 UIPercentDrivenInteractiveTransition 属性，用来更新动画百分比，在 UINavigationControllerDelegate 的某个协议方法中返回此属性	创建 UIPercentDrivenInteractiveTransition 属性，用来更新动画百分比，分别在 UIViewControllerTransitioningDelegate 的两个协议方法中返回不同动画的百分比	不支持手势动态驱动