UIScrollView: A Brief Overview

UIScrollView的简要概述

UICollectionView is a direct subclass of UIScrollView, much like UITableView. Similarly to the UICollectionView inheritance, the UICollectionViewDelegate protocol conforms to the UIScrollViewDelegate protocol. In practical terms, this means that if an object is the delegate of a collection view, it receives callbacks notifying it of UICollectionViewDelegate events as well as UIScrollViewDelegate events.

和UITableView一样，UICollectionView 是UIScrollView的直接子类,。类似于UICollectionView继承,UICollectionViewDelegate协议符合UIScrollViewDelegate协议。实际上,这意味着如果一个集合视图的委托对象可接收到UICollectionViewDelegate事件的回调通知以及UIScrollViewDelegate代理对象的回调通知。

UIScrollView is a versatile class in UIKit and has been around since iOS was iPhone OS 2.0. It provides a friendly way for developers to scroll content, whether it be a list of emails, a grid of apps, or a single photo. If you can scroll something in any given app, chances are that the app uses a scroll view.

UIScrollView是UIKit通用的类，因为iOS2.0的时候已经出现。它提供了一个友好的方式为开发人员滚动内容，无论是电子邮件列表，网格的应用程序，或一张照片。如果你可以在任何给定的应用程序滚动东西，它很可能是应用程序使用了（scroll View）滚动视图。

Scroll views give a familiar feel to the user and make any application that uses them seem more like it belongs in iOS and less like its developer wrote his own scroll view. Scroll views offer a lot of power to developers for very little work; all that developers need to do is set up the scroll view and add subviews to it. In addition, you get to rely on the work that Apple has already done for you, like emulating physics and deceleration. Take a look at an example in which the user can scroll to see more content than can fit on the screen simultaneously.

滚动视图给用户一个熟悉的感觉，使任何应用程序使用它们似乎就已经明确知道它是属于iOS和并不象那些自己写了自己的滚动视图开发的app。滚动视图提供了很有力的帮助给开发者然开发者可以做根少的工作，所有的开发人员需要做的是建立滚动查看和添加子视图。此外，你还得依靠苹果已经为你做的工作，比如模拟物理和减速。看看一个例子，用户可以滚动看到更多的内容比可以在屏幕上全部显示更加适合移动程序。

Create a new Xcode project with the Single View template. Copy a large image into the project and open the main view controller’s implementation file. Replace the viewDidLoad implementation with the one in Listing 2.3.

与单一视图模板创建一个新的Xcode项目。将大图像复制到项目中，并打开主视图控制器的实现文件。与清单2.3中的一个取代viewDidLoad实施。

Listing 2.3 A Simple Scroll View Example

-(void)viewDidLoad

{

[super viewDidLoad];

//First we create an image to display to the user. //Replace "cat.jpg" with whatever your image is named

UIImage \*image = [UIImage imageNamed:@"cat.jpg"];

//Next we create an image view to display the image.

//It should be the same size as the image with its origin

//in the top-left corner

UIImageView \*imageView = [[UIImageView alloc] initWithImage:image]; imageView.frame = CGRectMake(0, 0,

image.size.width, image.size.height);

//Finally we create our scroll view. We give it a frame //corresponding to our view’s bounds so it fills the entire view UIScrollView \*scrollView = [[UIScrollView alloc]

initWithFrame:self.view.bounds];

//This line is very important - it makes the scroll view scroll

scrollView.contentSize = image.size;

//This is just to get rotation to work correctly

scrollView.autoresizingMask = UIViewAutoresizingFlexibleWidth | UIViewAutoresizingFlexibleHeight;

//Finally, set up the view hierarchy

[scrollView addSubview:imageView];

[self.view addSubview:scrollView];

}

Run the application, and you see output similar to Figure 2.4; the image is too large to fit on the screen at one time, but the user can scroll around the image to see it all. (Notice the scroll indicators.) The magic that makes this all work is the contentSize property. This is a CGSize value that represents the size (in points) of the scrollable area. Its default value is zero, and it must be set to use any scroll view, even if the content size is smaller than the scroll view’s own size.

运行应用程序，你会看到类似于图2.4的输出；图像太大，无法在屏幕上同时安装一次，但是用户可以在图像周围滚动看到所有的图像。（注意滚动指标。）魔法让这一切工作的contentsize财产。这是一个cgsize表示大小（点）的滚动区域。它的默认值是0，并且必须设置为使用任何滚动视图，即使内容大小小于滚动视图自身的大小。



Figure 2.4 A simple scroll view example

When the scroll view knows the size of the content it’s displaying, it scrolls. The contentSize property can change at any time.

当滚动视图知道它显示的内容的大小时，它会滚动。的contentsize属性可以随时更改。

Figure 2.5 demonstrates the idea of content size. The light region of the photo, in the upper left, defines the visible part of the image when the application first launches. This is the size of the scroll view and is represented by dashed lines. The solid lines represent the content size of the scroll view.

图2.5演示了内容大小的例子。照片的光区，在左上角，在应用程序启动时定义图像的可见部分。这是滚动视图的大小，由虚线表示。实线表示滚动视图的内容大小。

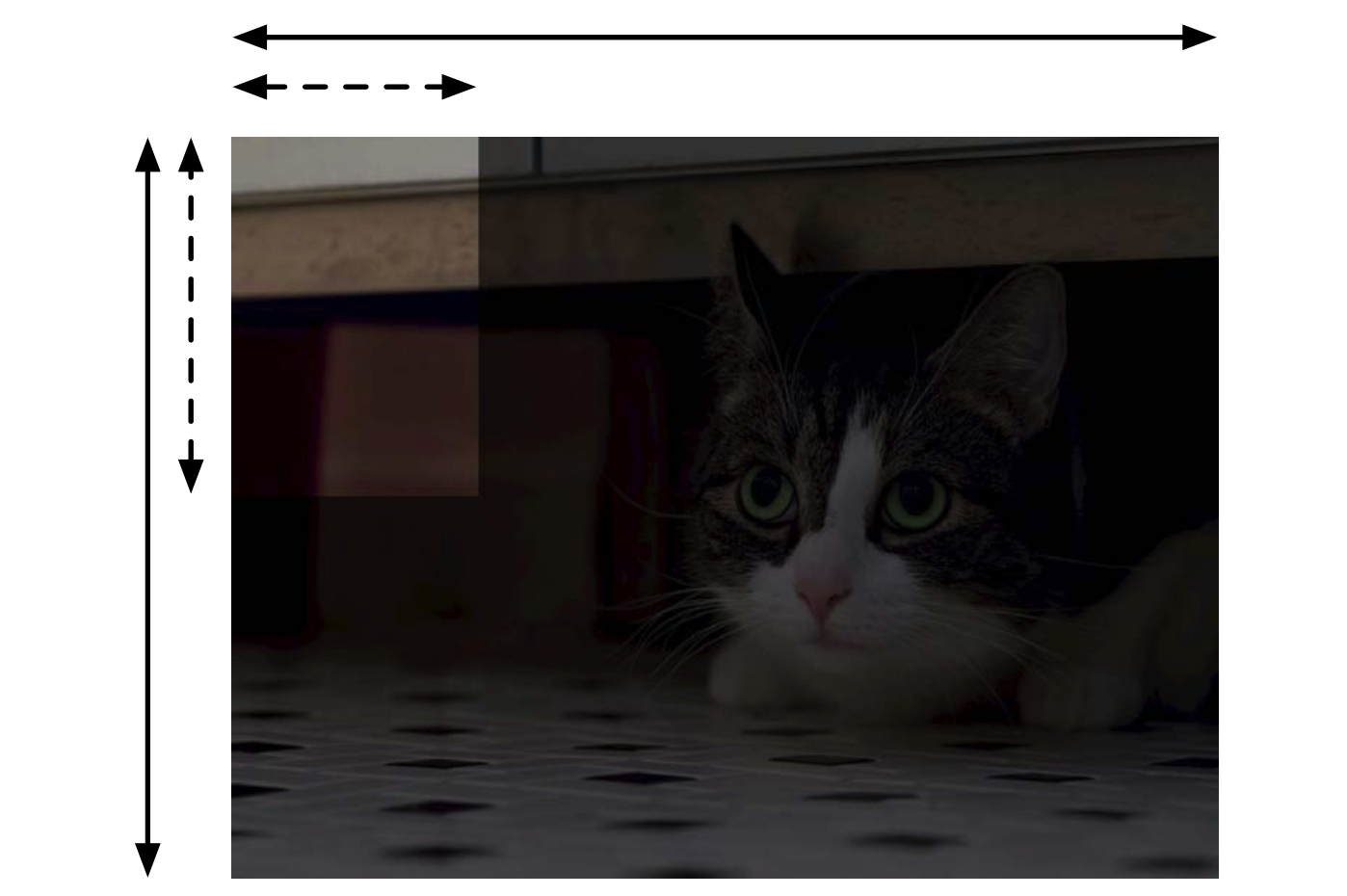


Figure 2.5 Content size example

When the user scrolls the scroll view, the content area visible to the user changes. The position of the content view within the scroll view is called the content offset and is represented by the contentOffset property, a CGPoint value. This property is defined by the distance from the visible region’s origin (top-left corner) to the origin of the content. Figure 2.6 demonstrates content offset with white dashed lines. The content size remains the same, but the content offset changes to respond to user interaction.

图2.5内容大小示例

当用户滚动滚动视图时，用户可见的内容区域发生变化。内容视图在滚动视图中的位置被称为是内容偏移是由contentoffset属性表示一个值，是一个CGPoint类型。此属性是由可见区域的原点（左上角）与内容的原点所定义的距离来定义的。图2.6演示了带有白色虚线的内容偏移量。内容大小保持不变，但内容偏移量更改响应用户交互。

igure 2.6 
Content offset example 

Content offset can be changed programmatically; the contentOffset property is readwrite. More interestingly, you can use the setContentOffset:animated: method to animate the change in content offset. This “moves” the scroll view, just as it would if the user moved it herself. The content offset can also be changed with scrollRectToVisible:animated:, but this is more often used with zooming than simple scrolling.

内容偏移可以以编程方式更改的contentoffset属性读写。更有趣的是，你可以使用setContentOffset:animated:进行动画：动画的改变内容的偏移偏移变化的方法。这“移动”滚动视图的方式，就像是用户自己移动滚动视图。内容偏移也可以改变通过scrollRectToVisible: animated:，但这个更常用与缩放而不是简单的滚动。

The last thing I want to mention about scroll views is the contentInset property. This is a UIEdgeInset value that represents the area around the scroll view’s content that it should“pad.”SettingthecontentInsetpropertytoUIEdgeInsetsMake(10, 10, 10, 10) would create a 10-point margin surrounding the scroll view’s content. The edge inset values can also be negative; this would represent area around the scroll view content that can’t be seen by the user (unless she scrolls past the edge of the scroll view). Try playing around with contentInset to see how it works.

我想提到滚动视图的最后一件事是contentinset属性。这是一个UIEdgeInset类型，表示滚动视图的内容的周围地区应该呈现的“间距”。Setting the contentInset property to UIEdgeInsetsmake（10，10，10，10）围绕着滚动视图周围会显示10像素的间距，边插入值也可以为负值；这将代表滚动视图内容的区域，用户不能看到该内容（除非滚动滚动视图的边缘）。试着用用contentinset看看它是如何工作的。

This contentInset is a widely used property and is often employed using UITableView and custom pull-to-refresh controls. It’s also useful if you have a navigation bar over top of a view controller with wantsFullScreenLayout set to YES. The inset’s top value would be equal to negative the height of the status bar and the navigation bar.

这contentinset是一种广泛使用的性能，经常使用表格和自定义下拉刷新控件。如果你想要导航控制器的导航栏在一个想要进行全屏设置的子控制器的顶部他也同样是有用的需要将其设置为Yes(wantsFullScreenLayout 这个属性在iOS 7.0之后已经废弃)。插图的顶部值将等于状态栏的高度和导航栏的高度。

Those are the three main components to UIScrollView: contentSize, contentOffset, and contentInset. Now it’s time for a quick discussion about the scroll view delegate before the chapter moves on to some more collection view material.

There are three groups of methods in UIScrollViewDelegate: those responding to dragging and scrolling, those responding to zooming, and those responding to scrolling animations initiated explicitly by code (see Table 2.1). You’re going to be dealing only with the first and last groups because collection views don’t use the zoom functionality of UIScrollView.

这些是UIScrollView的三个主要组成部分：contentsize，contentoffset，和contentinset。现在是在章节快速讨论滚动视图委托的时候了，在移动到更多关于集合视图的内容之前。

在滚动视图的代理中有三种方法：拖动和滚动，那些对缩放，和那些对滚动的代码明确启动动画（见表2.1）。你将只处理第一个和最后一个组，因为集合视图不使用UIScrollView的缩放功能

Table 2.1 Useful UIScrollViewDelegate Methods

Method Name Description

|  |  |
| --- | --- |
| scrollViewDidScroll: | Called whenever the content offset of the scroll view changes, either programmatically or in response to user interaction. Possible use could be in a custom pull-to-refresh control. |
| scrollViewWillBeginDrag-ging: | dragged by the user. Possible use could be Called whenever the scroll view is about to be  disabling updates to the scroll view that might interrupt smooth-scrolling performance. |
| scrollViewWillEndDragging: withVelocity: targetContentOffset: | Called whenever the user has lifted his finger from the scroll view after dragging. The secondparameter specifies a speed, in points/second, that the scroll view has at the moment the user lifts his finger. The third parameter is a pointer to a CGPoint, representing where the scroll view will scroll. Modifying the CGPoint at that pointer changes where the scroll view scrolls to. Possible use could be calculating what content is *going to be* visible when the scrolling animation ends and prefetching it from an application programming interface (API). |
| scrollViewDidEndDragging: willDecelerate: | Called whenever the user has lifted his finger from the scroll view after dragging. The second parameter specifies whether the scroll view animates its deceleration to come to a stop or if it was already stopped when the user lifted his finger. Possible use includes restarting any paused computations halted in |
| scrollViewWillBeginDragging: | as long as the second parameter is NO. |
| scrollViewShouldScrollToTop: | determine whether tapping on the status bar should animate the scroll view to the top. Only one visiblescroll view should return YES from this method at a time. Called whenever the operating system needs to |
| scrollViewDidScrollToTop: | Called after the scroll view scrolled to the top in response to the user tapping the status bar. |
| scrollViewWillBeginDecelerating: | Called whenever the scroll view is about to begin a decelerating animation. |
| scrollViewDidEndDecelerating: | Called after the scroll view’s deceleration animation completes. Possible use includes restarting any paused computations halted in scrollViewWillBeginDragging:. |
| scrollViewDidEndScrollingAnimation: | Called after the scroll view’s content offset change animation has completed. This method is only invoked on the delegate if the content offset was changed programmatically and with explicit animation enabled. |

You use some scroll view delegate methods later on in more advanced chapters in this book and in some case studies. They are useful tools to solving many problems, and you should be aware of them.

在本书中的一些更高级章节和一些案例研究中使用一些滚动视图委托方法。它们是解决许多问题的有用工具，你应该意识到它们