Before you dive into UICollectionView, you should get familiar with some of the

conventions and terms used in this book. The book starts with the basics of the iOS application lifecycle and then discusses the Model-View-Controller (MVC) paradigm. Even if you’re an experienced iOS developer already familiar with these topics, I encourage you to read this chapter to make sure that you’re on the same page (or screen, so to speak) that I am while you’re reading the rest of this book.

在你深入了解UICollectionView之前，你应该熟悉这本书中的一些惯例和术语，这本书从iOS应用程序生命周期的基本知识开始讲解,然后讨论了模型-视图-控制器(MVC)设计模式。即使你是一个有经验的iOS开发人员并且已经熟悉了这些话题,我仍然鼓励您阅读这一章,以确保你在相同的页面上(或屏幕,可以这么说),阅读这本书的其余部分。

Basics of the Application Lifecycle

（应用程序的基本生命周期）

The iOS application lifecycle differs a little from typical native applications on other platforms (although recent changes to OS X show Apple is interested in making the iOS lifecycle the norm). Developers no longer have hard-and-fast rules for when their applications are terminated, suspended, and so on. Let’s start with a simple scenario to describe a typical application lifecycle.

iOS的应用程序和本机其他平台的应用程序略有不同（尽管最近OSX的变化显示苹果对使iOS的生命周期变得规范变得感兴趣），当应用程序终止、挂起、或则其他情况的时候，开发人员不再有固定的规则。让我们从一个简单的场景来描述一个典型的应用程序的生命周期。

The user has just turned on his phone, and no applications are running except for those that belong to the operating system. Your application is not running. After the user taps your app’s icon, Springboard—the part of the OS that operates the Home screen of iOS— launches your app. Your app, and the shared libraries it needs to execute, is loaded into memory while Springboard animates your Default.png on the screen. Eventually, your app begins execution, and your application delegate receives the appropriate notification. When your application is running and in the foreground, it is in the active state.

用户在刚刚打开手机，而且手机除了操作系统的一些程序没有其他程序在运行的时候。你自己的应用程序同样也是没有运行。在用户轻点你的app图标，使用OS操作系统的一部分通过操作iOS的主屏幕来启动你的app。你的app，需要执行共享库，默认图片被展示到屏幕上同时被加载到内存中。最终应用程序开始执行，同事应用程序的委托或则代理受到相应的通知。当你的应用程序活动在前台，那你的应用程序就处于活动的状态

On iOS, users tend to only use any given application for a few seconds before returning their phones to their pockets. After the user has put away your app by pressing the Home button on her iPhone or iPad, your application enters the background state. Typically, apps have 10 seconds to complete any database saves or other long-running tasks (though applications can request additional time from the OS). When all the background processing is complete, the application finally becomes suspended. While suspended, applications remain in memory but may not execute code. The state of your application is persisted. If the user opens your application while it is suspended, it begins execution exactly where it left off. If memory becomes low, the OS can kill your app while it is in the suspended state. The user can also manually terminate your app from the multitasking tray. Once terminated, applications return to their initial state of not running.

对于iOS上，用户倾向于在他们的手机返回他们的口袋之前的那几秒种使用应用程序。在用户通过按压iPhone或则ipad上面的home键将你的应用程序从主屏幕上隐藏，你的app会进入后台运行状态。通常,应用程序有10秒来完成任何数据库保存或其他长时间运行的任务(虽然从OS应用程序可以请求更多的时间)。当所有的后台处理完成,应用程序最后变成了暂停。暂停时,应用程序保持在内存中但可能不再不执行代码。您的应用程序的状态被保存。如果用户打开暂停中的应用程序,它开始执行确切位置。如果内存变得低,操作系统可以杀死你的应用即便它处于暂停状态。用户也可以手动终止应用程序的多任务托盘。一旦终止,返回应用程序未运行时候的初始状态。

But wait, it gets more complicated! If the user receives a calendar alert, opens the multitasking tray, or gets a phone call, your application can be put into the inactive state. Your application is still running, but it is no longer the foremost thing the user interacts with. For example, games pause themselves. As an application developer, you need to be aware of this and use it as an indication that the user might leave your application soon.

但是等等,应用程序变得更加复杂!如果用户收到一个日历提醒,打开多任务托盘,或者接到一个电话,您的应用程序可以放在不活跃的状态。应用程序仍在运行,但它不再是最重要的用户与之交互的对象。例如,游戏暂停。作为应用程序开发人员,您需要了解和使用它作为一个指示,用户可能很快离开你的应用程序。

The user can open your application without tapping its icon on the Home screen. If your application receives local or push notifications, or if it is registered for custom URL scheme handling, the user can open it in any number of ways.

用户可以打开你的app不通过主屏幕上的图标。如果你的app收到一个本地通知或则推送，或者如果它是注册自定义URL方案处理,用户可以以任意数量的方式打开它。

The application lifecycle is important to understand for all iOS developers who want to make enriched, immersive experiences. These types of applications are exactly what UICollectionView is great for, so no comprehensive discussion of UICollectionView would be complete without a summary of the application lifecycle.

对于想要丰富,身临其境的体验的iOS开发者，理解程序生命周期都是重要的。这些类型的应用程序正是确切展示UICollectionView的伟大,所以没有全面的讨论UICollectionView总结就是不完整的应用程序的生命周期。

If your app enters the inactive state, stop updating your interface. It would be disconcerting for a user to see your collection-view contents move about while he’s deciding whether to view the details of an appointment that has popped up over your application. Likewise, don’t update your app’s interface while the application is in the background. The state of the user interface should remain fixed between the switch from active to background and back to active.

如果你的应用程序进入不活动状态,停止更新你的界面。对用户来说在他决定是否查看预约却看到你的集合视图的内容移动而突然出现在你的应用程序的细节的时候将是令用户不安的。同样，在应用程序是在后台的时候，不要更新您的应用程序的接口。用户界面的状态应该保持在从主动到后台切换到主动。

How to Use MVC

MVC is not a difficult concept, but there are two main reasons for emphasizing its importance in iOS:

* ▪  MVC is used by CocoaTouch (and Cocoa on OS X). If you adhere to the same paradigm as the frameworks used for writing all iOS applications, your code will flow well and not clash with the built-in classes, including UICollectionView.
* ▪  MVC is generally a good framework, and using it will help you make well-written, maintainable apps.   
  Now that you know why MVC is important, it’s time to look at what MVC is. Figure 1.1 shows the basics of MVC; strong relationships are represented with solid lines, and weak relationships are represented by dashed ones. Strong and weak relationships indicate to the compiler how to manage memory and are important to avoid memory leaks, which would eventually lead to the app being terminated.

VO 
Model 
Controller 
Owns 
User Interaction 
View 
Owns 

Figure 1.1 Basics of MVC

MVC并不是一个困难的概念,但有两个主要原因在iOS强调它的重要性:

▪使用MVC CocoaTouch(OS X和可可)。如果你坚持相同的范式框架用于编写所有iOS应用程序,您的代码将会非常顺畅,而不是冲突与内置的类,包括UICollectionView。

▪MVC通常是一个很好的框架,和使用它将帮助您使编写良好的、可维护的应用程序。现在你知道为什么MVC是很重要的,现在是时候看看什么是MVC。图1.1显示了基本的MVC;牢固的关系用实线表示,和弱关系用虚线表示的。强和弱关系指示编译器如何管理内存和是重要的,以避免内存泄漏,这最终将导致应用程序被终止。

At the heart of MVC is the controller object. The controller is a view controller—as in UIViewController—and it controls the view. It maintains a strong relationship to this view, which is what is presented to the user on the screen. The controller also maintains a strong relationship to the model. The model represents data that is represented in the view.

If your view ever has a reference to your model, or vice versa, you’re doing it wrong. This book uses MVC and you should, too.

Most of the code in any given application resides in the controller; controllers mediate the interactions between views and models, which is why the code in controllers is often referred to as glue code.

MVC是控制器的核心对象。控制器是一个视图控制器上UIViewController-and控件视图。保持一个强大的关系这一观点,也就是在屏幕上呈现给用户。控制器还维护一个强大的关系模型。中模型表示的数据视图。

如果你的视图关联到一个参考模型,反之亦然,你做错了。这本书使用MVC,你也应该这么做。

大部分的代码在任何给定应用程序驻留在控制器;控制器协调视图和模型之间的交互,这就是为什么控制器中的代码通常被称为胶水代码。

What sort of interactions does a controller mediate? Well, if the view contains a button, the view controller is notified when the user taps that button. User interactions usually trigger actions to modify, create, or delete models belonging to the controller. The controller receives the user interaction from the view, updates the model, and then updates the view to reflect the changes made to the model.

Sometimes, the model changes without user interaction. For example, consider a view that displays a large JPEG, which is being downloaded. When the download completes, the controller should be notified so that it can update the view. On iOS, you have a few different choices for how to notify the controller. My favorite is Key-Value Observation (KVO). Controllers can register themselves as observers on model objects so that they are notified whenever the model’s properties are changed. Other ways for models to interact with controllers on iOS include NSNotificationCenter, delegation, and NSFetchedResultsController. I would avoid NSNotificationCenter for model- controller interaction in favor of NSFetchedResultsController or KVO. Although this book doesn’t discuss Core Data, UICollectionView works very well with NSFetchedResultsController in a similar way to UITableViewController.

This last example demonstrates a gaping hole in MVC: Where does the network code go? As a responsible iOS developer, you should keep the view controller to only mediating the interactions between the view and the model. If that’s the case, it shouldn’t be used to View the network access code. As discussed in Chapter 6, “Adding Interactivity to UICollectionView,” the network code should be placed outside of the typical MVC pyramid. Network access should not involve the view whatsoever, but it can sometimes involve the model.

什么样的交互是控制器调节?如果视图包含一个按钮,当用户点击按钮的时候，试图控制器会被通知。通常用户交互触发动作修改、创建或删除属于控制器的模型。从视图控制器接收到用户交互,更新模型,然后更新视图,以反映对模型所做的变更。

有时候,模型的变化并没有用户交互。例如,考虑一个视图显示JPEG,被下载。当下载完成时,应当通知控制器,以便它可以更新视图。在iOS,你有几个不同的选择,如何通知控制器。我最喜欢的是键值观察(KVO)。控制器可以注册自己是观察者模型对象,模型的属性被更改时得到通知。其他方式iOS模型与控制器的交互包括NSNotificationCenter（通知中心）,代理,NSFetchedResultsController。我将避免使用NSNotificationCenter进行模型—控制器交互，支持NSFetchedResultsController或KVO进行交互。

虽然这本书不讨论CoreData,UICollectionView通过NSFetchedResultsController的方式运行效果效果很好，这效果和UITableViewController 类似。

最后一个示例演示了一个大洞在MVC:网络请求的代码应该放在哪个位置?作为一个负责任的iOS开发人员,您应该保持视图控制器只调停视图和模型之间的交互。如果是这种情况,它不应该被用来查看网络访问代码。正如第六章中所讨论的,“添加交互性UICollectionView,”网络代码应放置在典型的MVC金字塔。网络访问不应涉及任何视图,但它有时会涉及到模型。

Well, that’s mostly true. In fact, a common paradigm for fetching details about a model from an application programming interface (API) involves Grand Central Dispatch blocks. A block lets developers treat anonymous functions as first-class Objective-C objects. These blocks can be invoked later. Controllers can start a network request and pass the network- fetching object a callback block that updates the view. Technically, the network code has an indirect reference to the view, but you ignore it lest you find yourself falling down a rabbit hole of pedantry.

If you are experienced in iOS development, all of this should sound familiar. UICollec- tionView and UICollectionViewController don’t exist in silos; they are used within applications with models and with the rest of CocoaTouch. It would be irresponsible to present them in any other context than that of MVC.

这是真实的。事实上,一个常见的模式获取详细信息模型从一个应用程序编程接口(API)涉及中央调度块。一block可以让开发人员将匿名函数作为一流的objective - c的对象。这些模块之后可以调用。控制器可以启动一个网络请求,并通过网络,获取对象回调块更新视图。从技术上讲,网络代码和视图是有着间接的关联,但是你忽略它以免你发现自己掉进迂腐的兔子洞。如果你是经验丰富的在iOS开发者,所有的这一切应该听起来很熟悉。UICollectionView和UICollectionViewController在筒仓是不存在的;它们被使用在应用程序中通过模型和CocoaTouch的其余部分。和MVC相比任何其他上下文展示他们将是不负责任的。

MVC and UICollectionView

Now that you've read about the MVC paradigm, look at its application in the context of

writing UICollectionView code.

The view component of MVC with UICollectionView is unsurprisingly the UICollectionView itself; the controller is either a subclass of UICollectionView- Controller or a subclass of UIViewController that conforms to the UICollection- ViewDataSource and UICollectionViewDelegate protocols; the model can be anything.

Like with UITableView, your controller can either subclass UIViewController and conform to the two protocols for the collection view data source and delegate or it can subclass UICollectionViewController itself. If you look in the header file of UICollectionViewController, you see that it’s very sparse. The controller inherits from UIViewController—conforming to UICollectionViewDataSource and UICollectionViewDelegate—and has a convenience initializer to programmatically create an instance of it using a collection view with a specific layout. It contains a property to access the collection view and another property to specify whether the selection in a collection view becomes cleared when it (re)appears.

When using a UICollectionViewController subclass, the view property of UIViewController points to the same object as the collectionView property of UICollectionViewController. The view is the collection view. If you plan to use only UICollectionView to display data to your user, I strongly recommend subclassing this prebuilt controller. In my experience, you run into fewer “gotchas” using these special controllers from Apple.

现在你已经读到MVC模式中,看它的应用程序在上下文中编写UICollectionView代码。

使用UICollectionView的MVC的视图组件，不出所料是UICollectionView本身;控制器要么是UICollectionView--Controller的一个子类或者是UIViewController的一个子类,UIViewController遵守UICollectionView的数据源协议和 UICollectionViewDelegate协议;模型可以是任何东西。像UITableView,控制器可以是UIViewController子类并且符合数据源集合视图的两个协议和委托或则是UICollectionViewController本身或他的子类。如果你看UICollectionViewController的头文件,你会看的很稀疏。控制器继承自UIViewController-conforming UICollectionViewDataSource UICollectionViewDelegate-and有方便初始化以编程方式创建一个实例,它使用一个集合视图与特定的布

局。它包含一个属性来访问集合视图和另一个属性来指定是否选择在一个集合视图时便成了清除(重新)出现。在使用一个UICollectionViewController子类的时候，UIViewController中的视图即（view）属性，和collectionView 与 UICollectionViewController的关系是类似的。那个视图就是集合视图的视图。如果你计划只使用UICollectionView向您的用户显示数据,我强烈建议子类化这个预先构建的控制器。根据我的经验,你遇到更少的使用这些特殊控制器时的“陷阱”。

In some circumstances, subclassing UIViewController is preferable. For example, if your view contains a collection view, but also contains other views, it’s easier to have the collection view as a subview of the controller’s view. The distinction is minor, but important.

Figures 1.2 and 1.3 demonstrate the differences in the two approaches to using collection views. UICollectionViewController is much simpler; it should be the approach you take first. If you find you can’t solve your problem with it, switch to using the second approach. It’s usually easy to switch from using the first method to the second.

在某些情况下,让控制器继承自UIVewController是可取的。例如,如果您的视图包含一个collectionView（集合视图,但也包含其他视图,它更容易有集合视图的子视图控制器的视图。区别是很小,但很重要。图1.2和1.3演示了这两种方法的差异使用集合视图。UICollectionViewController是简单得多,它应该是你首先采用的方式。如果你发现你不能解决你的问题,切换到使用第二种方法。通常容易从使用第一种方法转换到第二。lCollectionViewController 
Model 
Figure 1.2 
Example Of MVC using UIC011ectionViewContr011er 
UlCollectionView 
Model 
UlViewController 
<UlCollectionViewDataSource, 
UlCollectionViewDelegate 
UlView 
Contains 
UlCollectionView 
Figure 1.3 
Example of MVC using Ulc011ectionView's protocols 

This book uses the first approach unless there is a good reason not to. Even though the view property of UICollectionViewController is the same as its collectionView property, the code used in this book carefully distinguishes between the two.

Now that you’ve seen how collection views fit within the MVC paradigm of iOS apps, look at the following simple example. Don’t worry; you experiment a lot with collection views in Chapter 2, “Displaying Content Using UICollectionView.”

In the following example, you create a simple iPhone app that displays a bunch of cells with random colors. To get started, create a new application with the Single View template. Make sure that Use Storyboards is unchecked; this book focuses on collection views, and I don’t want to have to diverge to discuss the peculiarities of storyboards. Delete everything in the view controller header file and replace it with the code in Listing 1.1.

通常这本书使用是第一种方法除非有一个很好的理由不去使用。尽管UICollectionViewController视图的属性 和collectionView属性一样,这本书中的代码在使用中任然会仔细区分这两个概念。现在您已经了解了如何收集意见符合MVC范式的iOS应用程序,看看下面的简单的例子。别担心,你与集合视图在第二章实验很多,用UICollectionView”来显示内容。”在以下示例中,您创建一个简单的iPhone应用程序显示一群细胞,随机颜色。首先,创建一个新的应用程序的单一视图模板。确保使用故事板是无节制的;这本书侧重于收集意见,和我不想要发散讨论故事板的特性。删除视图控制器中所有的头文件和替换清单1.1中的代码。

Listing 1.1 Basic UICollectionViewController Header File

@interface AFViewController : UICollectionViewController

@end

Replace AFViewController with the name of your view controller. My initials are AF, so

I prefix my class names with them to avoid namespace collisions.

Next, head over to your .xib file and delete the view. Drag a collection view onto the blank canvas and connect the collection view’s delegate and dataSource outlets to the File’s Owner, the view controller. It should look like Figure 1.4 when you’re done.

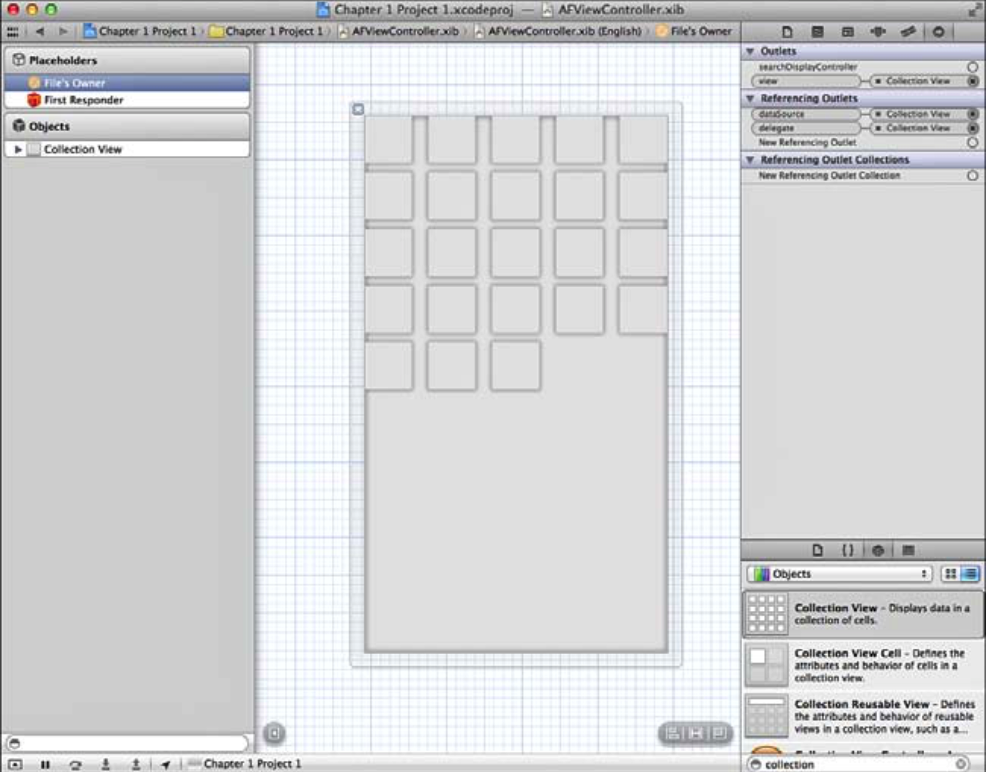


Figure 1.4 Basic UICollectionView setup using a .xib

将AFViewController换成你的视图控制器的名称。我的首字母是AF,所以我类名添加AF前缀来避免命名空间的冲突。接下来,去你的xib文件和删除视图。将一个集合视图拖到空白画布并连接集合视图的委托和数据源文件的所有者,视图控制器。它的外观应该类似于图1.4当你完成。

Now comes the fun part: the code! UICollectionViewDataSource has two required methods. One returns the number of items in a section, and another configures a cell for a given index path.

If you’re not familiar with these terms, don’t worry. Chapter 2 explains everything in great detail. This quick example just gets your feet wet.

Following MVC, you need a model. Use a basic array that you'll populate with a bunch of randomly generated colors. The top of your implementation file should look something like Listing 1.2.

接下来有趣的部分:代码!UICollectionViewDataSource有两个必需的方法。一个返回在每个分区的itemS数量,和另一个用来配置items通过给定的索引路径。如果你不熟悉这些术语,别担心。第二章详细地解释了一切。这个简单的例子只是让你润润脚。MVC之后,你需要一个模型。使用一个基本的数组,您会使用一群随机生成的颜色进行填充。实现文件的顶部应该如清单1.2所示。

Listing 1.2 Setting Up the Model

static NSString \*kCellIdentifier = @"Cell Identifier";

@implementation AFViewController

{

NSArray \*colorArray;

}

- (void)viewDidLoad {

[super viewDidLoad];

[self.collectionView registerClass:[UICollectionViewCell class] forCellWithReuseIdentifier:kCellIdentifier];

const NSInteger numberOfColors = 100;

NSMutableArray \*tempArray = [NSMutableArray arrayWithCapacity:numberOfColors];

for (NSInteger i = 0; i < numberOfColors; i++)

{

CGFloat redValue = (arc4random() % 255) / 255.0f; CGFloat blueValue = (arc4random() % 255) / 255.0f; CGFloat greenValue = (arc4random() % 255) / 255.0f;

[tempArray addObject:[UIColor colorWithRed:redValue green:greenValue blue:blueValue alpha:1.0f]];

}

colorArray = [NSArray arrayWithArray:tempArray];

}

Notice the copy of the array; we’re doing so to avoid a mutable instance as our color array, which would be unnecessarily slower.

The kCellIdentifier string is used to register a plain UICollectionViewCell as the cell for the collection view to use, so don’t pay much attention to it. The part that involves the model is the instance variable called colorArray. In viewDidLoad, you use a for loop to populate this array with random colors.

Now that you have the model set up, you need to configure your view to represent it. For this, use the two UICollectionViewDataSource methods mentioned earlier (see Listing 1.3).

注意数组的副本,我们这样做,以避免一个可变的实例作为我们的颜色数组,将不必要的慢。kCellIdentifier字符串用于注册一个普通UICollectionViewCell作为集合视图使用的细胞,所以不要太关注它。这部分涉及到模型实例变量被称为colorArray。在viewDidLoad,您使用一个for循环和随机颜色填充该数组。现在您已经建立的模型,您需要配置您的视图来表示它。为此,使用前面提到的两个UICollectionViewDataSource方法(参见清单1.3)。

Listing 1.3 Configuring the View

-(NSInteger)collectionView:(UICollectionView \*)collectionView numberOfItemsInSection:(NSInteger)section

{

return colorArray.count;

}

- (UICollectionViewCell \*)collectionView:(UICollectionView \*)collectionView cellForItemAtIndexPath:(NSIndexPath \*)indexPath

{

UICollectionViewCell \*cell = [collectionView dequeueReusableCellWithReuseIdentifier:kCellIdentifier forIndexPath:indexPath]; //Discussed in Chapter 2 - pay no attention

cell.backgroundColor = colorArray[indexPath.item];

return cell;

}

The first method—collectionView:numberOfItemsInSection:—lets the collection view know how many cells it’s going to display. You rely on the model to let the controller know what number to return. Next is collectionView:cellForItemAtIndexPath:, which returns a cell that you are responsible for configuring in a way that represents your model. To do this, you grab the model at the given index and use that color as the background color for the cell. If you run the app, you get something like what you see in Figure 1.5. Because the colors are randomly generated, of course, your app will look different.

第一个method-collectionView:numberOfItemsInSection:让集合视图知道有多少items将要被展示。你依赖于模型让控制器知道返回数量。接下来是collectionView:cellForItemAtIndexPath:返回一个细胞,你负责配置的方式表示你的模型。要做到这一点,你抓住指数给出的模型和使用颜色的单元格的背景颜色。如果你运行这个应用程序,你得到类似于图1.5你所看到的。因为颜色是随机生成的,当然,你的应用将会不同。



Figure 1.5 First run of the basic app

Note that we’re not using this collection view within a UINavigationController, so the status bar is transparent. In production code on iOS 7+, you’ll usually encapsulate your collection view within a navigation controller, whose navigation bar is extended behind the status bar.

So, this simple example demonstrates how a model can represent a view and how you can configure a view to represent that model without either being aware of the other. This example demonstrates the platonic ideal of what you should strive for: clear separation between model, view, and controller.

请注意,我们并没有将集合视图放在UINavigationController中进行使用,因此状态栏是透明的。在iOS7.0以上的版本中,你通常将代码封装在导航控制器的视图中,导航栏的状态栏后面的扩展。

因此,这个简单的例子演示了如何一个模型可以代表一个视图,以及如何配置一个视图来表示该模型没有意识到另一个。这个例子演示了你应该争取的柏拉图式的理想:清晰的分离模型,视图和控制器。