iOS 10 UICollectionView新特性

前言
关于 iOS 10 UICollectionView的新特性，主要还是体现在如下3个方面

顺滑的滑动体验
现在基本上人人都离不开手机，手机的app也每天都有人在用。一个app的好坏由它的用户体验决定。在可以滑动的视图里面，必须要更加丝滑柔顺才能获得用户的青睐。这些UICollectionView的新特性可以让你们的app比原来更加顺滑，而且这些特性只需要你加入少量的代码即可达到目的。
针对self-sizing的改进
self-sizing的API在iOS8的时候被引进，iOS10中加入更多特性使cell更加容易去适配。
Interactive reordering重排
这个功能在iOS9的时候介绍过了，苹果在iOS 10的API里面大大增强了这一功能。

目录

一. UICollectionViewCell顺滑的滑动体验
众所周知，iOS设备已良好的用户体验赢得了广大的用户群。iOS系统在用户点击屏幕会立即做出响应。而且很大一部分的操作是来自于用户的滑动操作。所以滑动的顺滑是使用户沉浸在app中享受的必要条件。接下来我们就谈谈iOS 10 中增加了那些新特性。

我们先来看一下之前 UICollectionView 的体验，假设我们每个cell都是简单的蓝色，实际开发app中，cell会比这复杂很多。 我们先生成100个cell。当用户滑动不是很快的时候，还感觉不出来卡顿，当用户大幅度滑动，整个UICollectionView的卡顿就很明显了。如果整个cell的DataSource又是从网络

加载的，那就更加卡顿了。效果如下图。

如果这种app上架，用户使用过后，很可能就直接给1星评价了。但是为什么会造成这种问题呢？我们来分析一下，我们模拟一下系统如何处理重用机制的，效果如下图

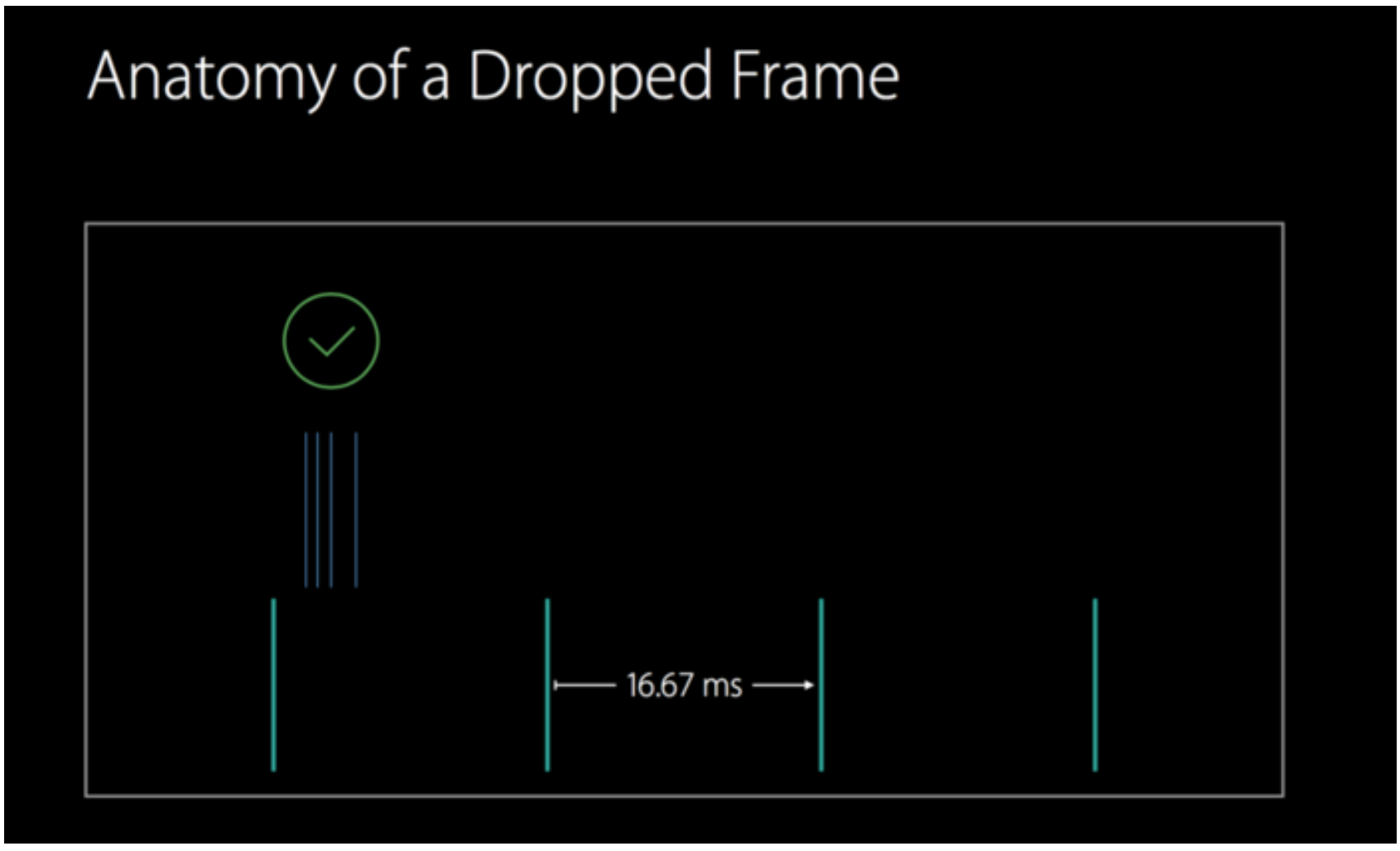
在上图中，我们可以看出，当cell准备加载进屏幕的时候，整个cell都已经加载完成，等待在屏幕外面了。而且更重要的是，在屏幕外面等待加载的cell是整整一行！这一行的cell都已经加载完数据。这是UICollectionView在用户大幅度滑动时卡顿的根本原因。用专业的术语来说，掉帧。

接下来我们就来详细的说说掉帧的问题。

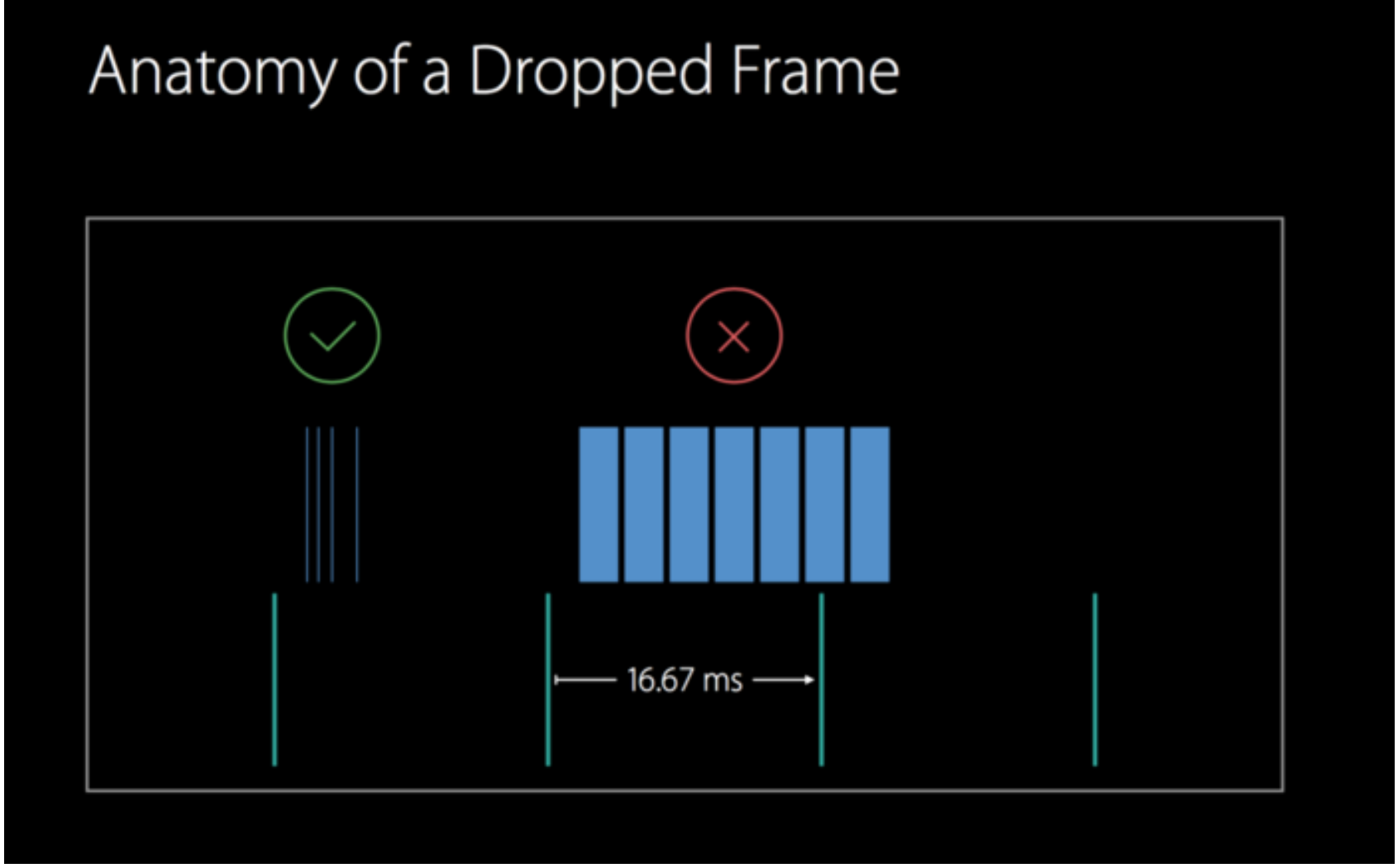
当今的用户是很挑剔的，用户需要一个很顺滑的体验，只要有一点卡顿，很可能一言不合就卸载app了。要想用户感觉不到卡顿，那么我们的app必须帧率达到60帧/秒。用数学换算一下就是每帧16毫秒就必须刷新一次。

我们用图标来分析一下掉帧的问题。下面会出现2种不同的帧。

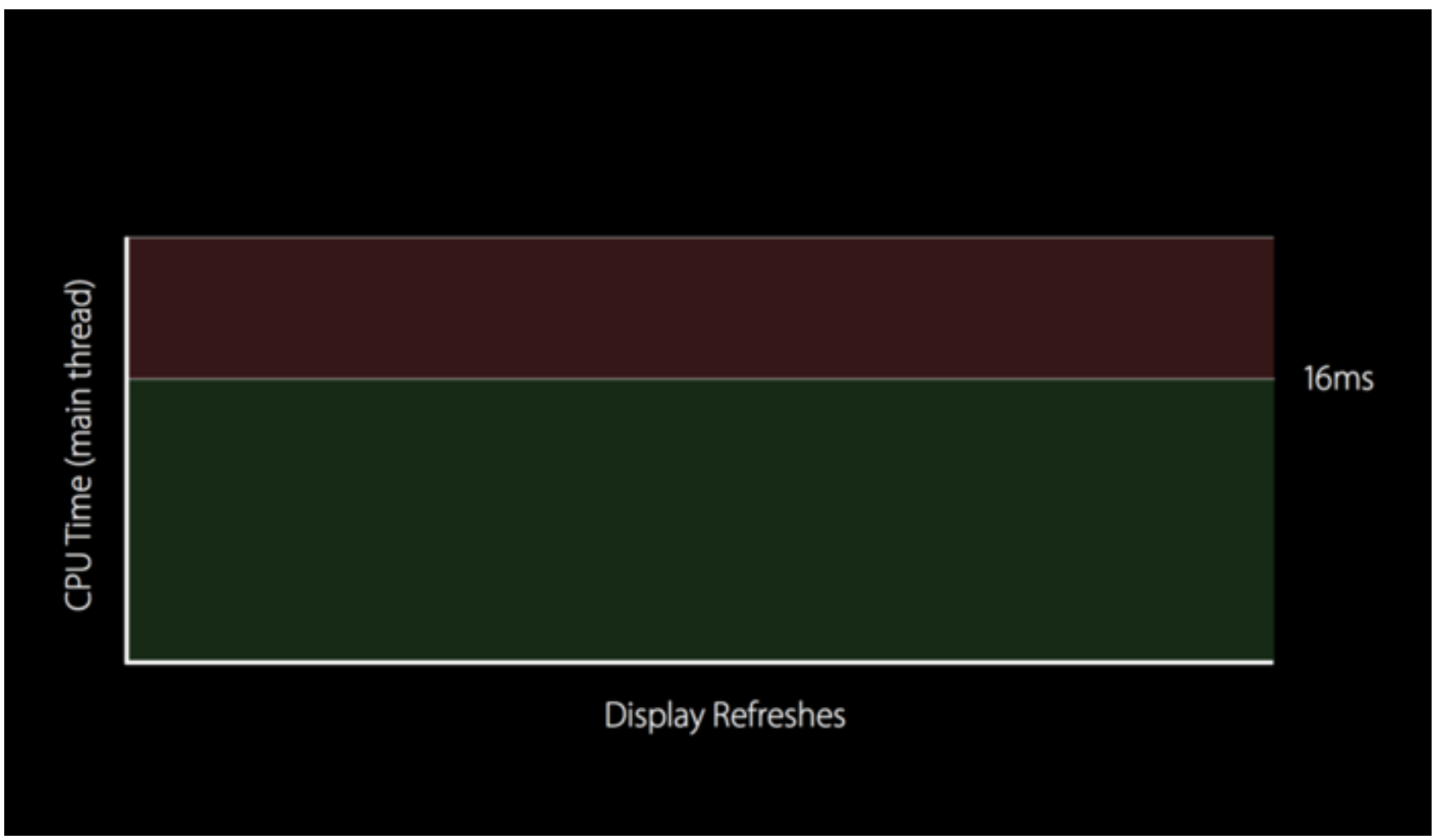
第一种情况，下图是当用户轻微的上下小幅度滑动。这个时候每个cell的加载压力都不大，iOS针对这种情况，已经做了很好的优化了，所以用户感觉不到任何卡顿。这种情况是不会掉帧，用户也希望能使用如此顺滑的app。



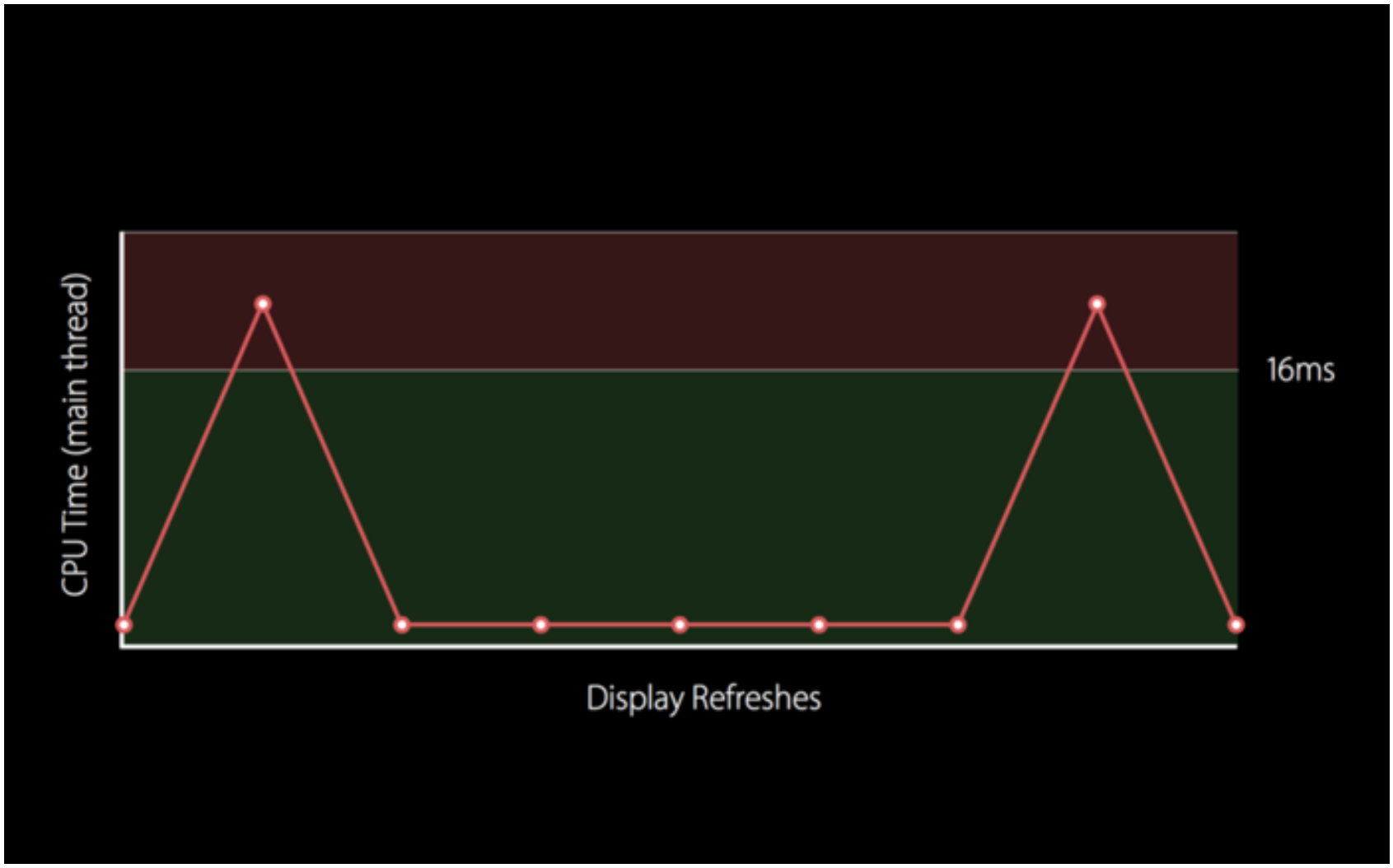
第二种情况，当用户大幅度滑动，每个cell加载的压力很大，也许需要网络请求，也许需要读取数据库，而且每次都加载一行cell出来，这样每个cell的加载时间都增加了，加载一行的总时间也就大大增加了，如下图所示。这样，不仅仅当前帧在加载cell，总的时间还会挤压到下一帧的时间里面去。这种情况下，用户就感觉到了卡顿了。



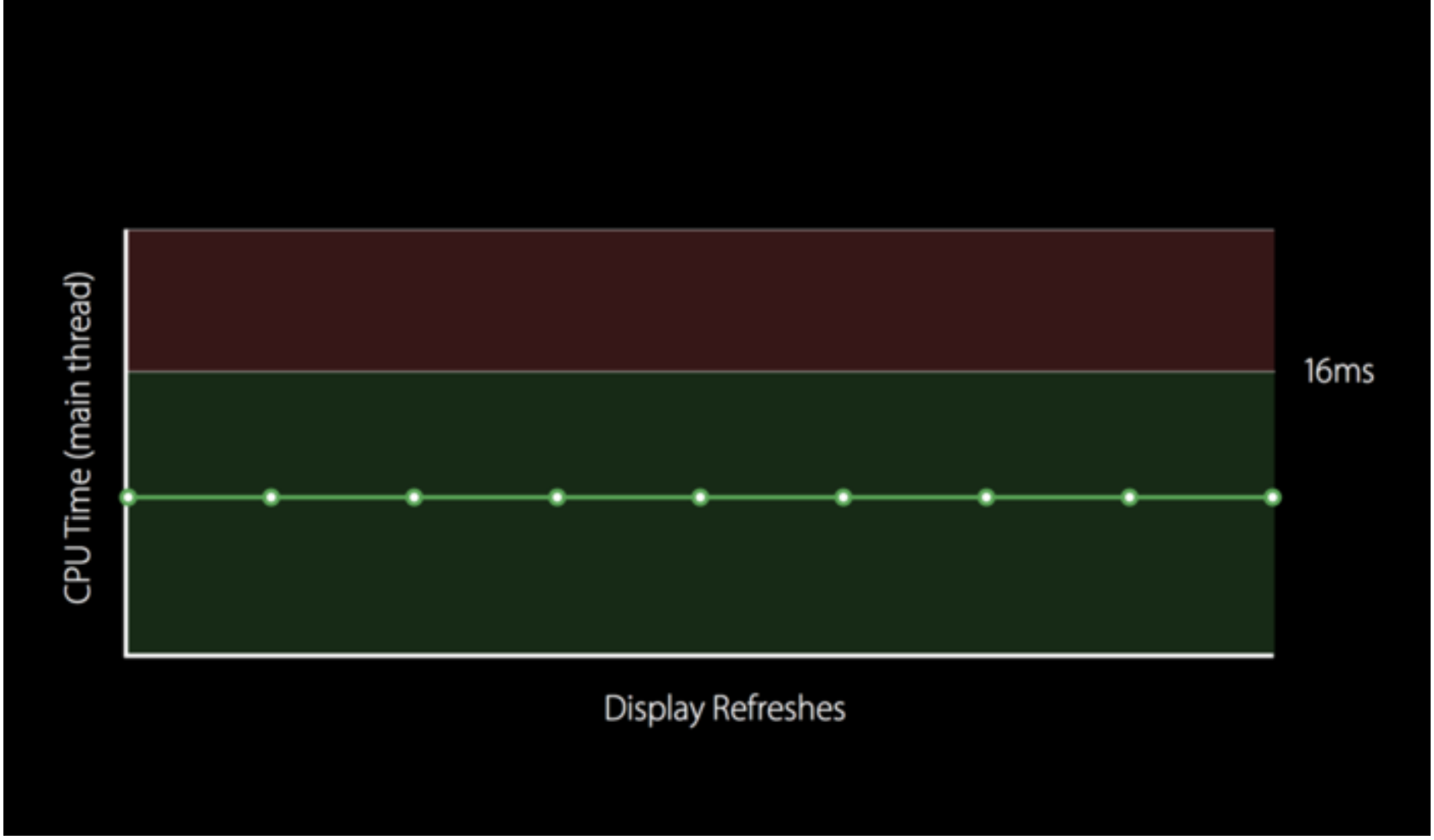
我们换种方式在说明一下2种情况下掉帧的情况。我们用下图的标准来衡量一下上面2种情况。下图分为2部分，上面红色的区域，就是表示掉帧的区域，因为高于16ms。红色和绿色区域的分界线就在16ms处。y轴我们表示的是CPU在主线程中花费的时间。x轴表示的是在用户滑动中发生的刷新事件。



针对上述掉帧的情况，绘制出实验数据，如下图。值得我们关注的是，曲线是很曲折的，非常的不平滑。当用户大幅度滑动的时候，峰值超过了16ms，当用户慢速滑动的时候，帧率又能保持在比较顺滑的区域。处于绿色区域内的cell加载压力都是很小的。这就是时而掉帧时而顺滑的场景。这种场景下，用户体验是很糟糕的。



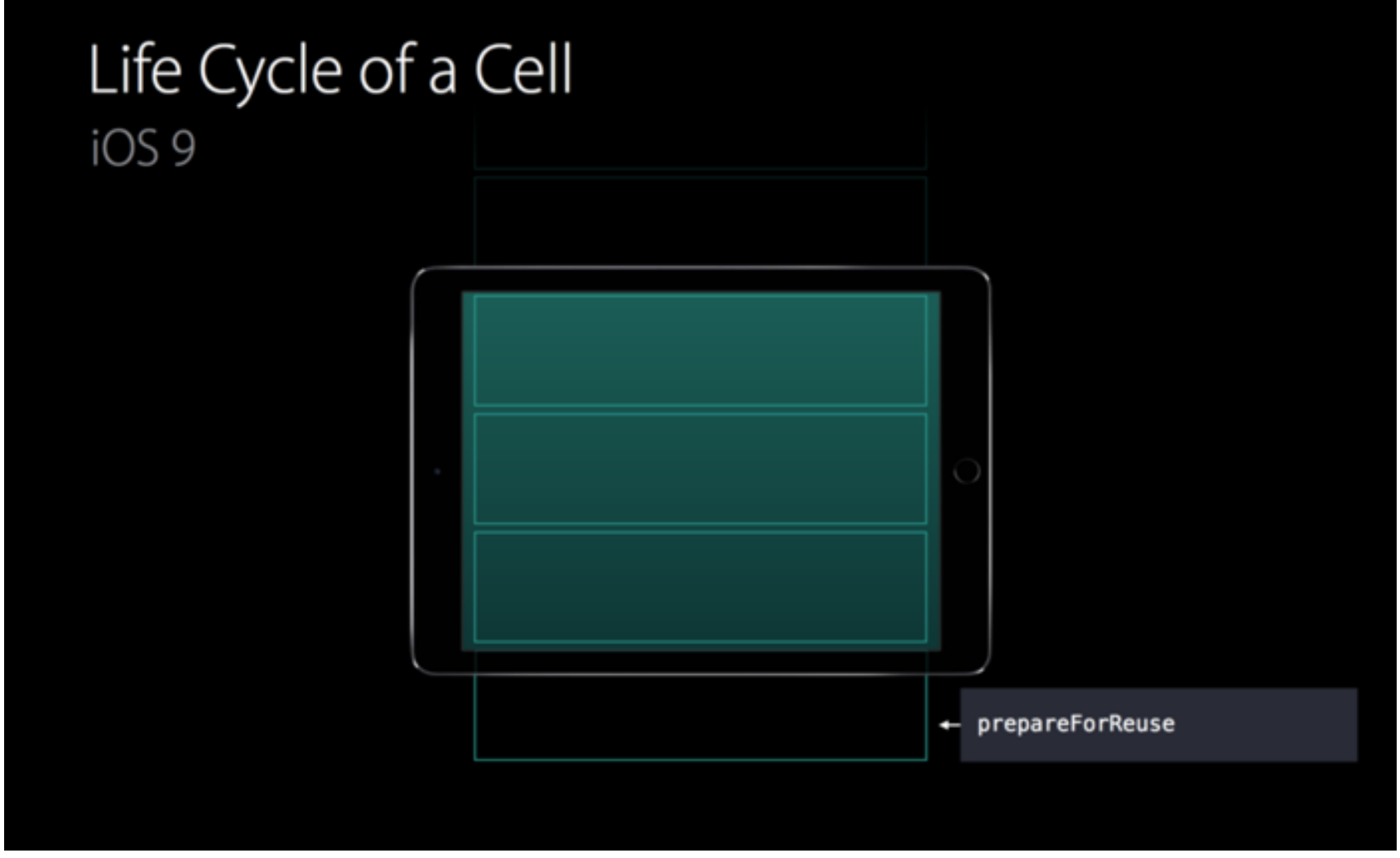
那怎么解决这么问题的呢？我们来看下图：



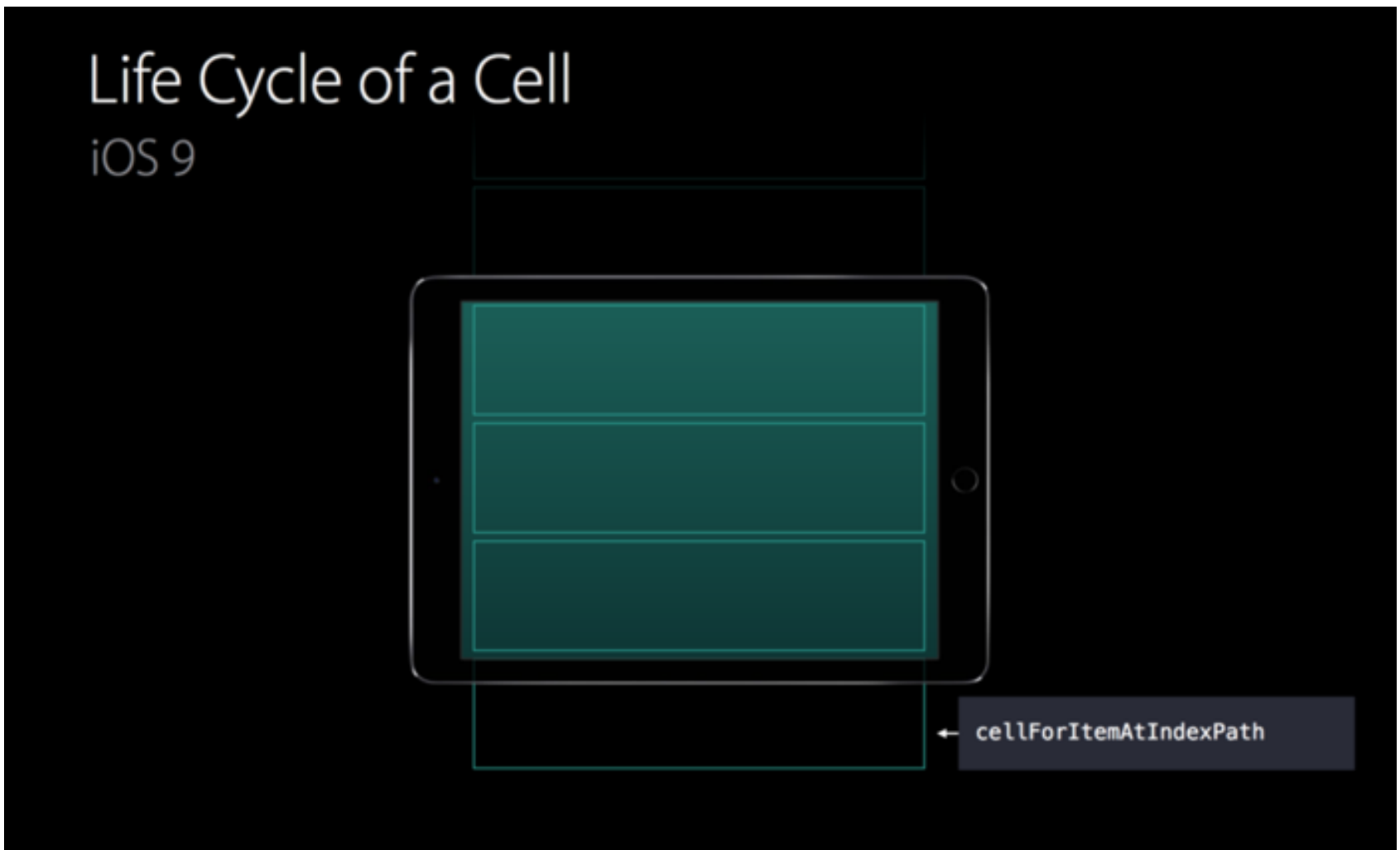
上图中的曲线我们看着就很平缓了，而且这种情况也不会出现掉帧的情况了，每个滑动中的时间都能达到60帧了。这是怎样做到的呢？因为把每个cell的加载事件都平分了，每个cell不会再出现很忙和很闲的两个极端。这样我们就取消了之前的波峰和波谷。从而让该曲线达到近乎水平的直线。

如何让每个cell都分摊加载任务的压力？这就要谈到新的cell的生命周期了。

先来看看老的 UICollectionViewCell的声明周期。当用户滑动屏幕，屏幕外有一个cell准备加载显示进来。

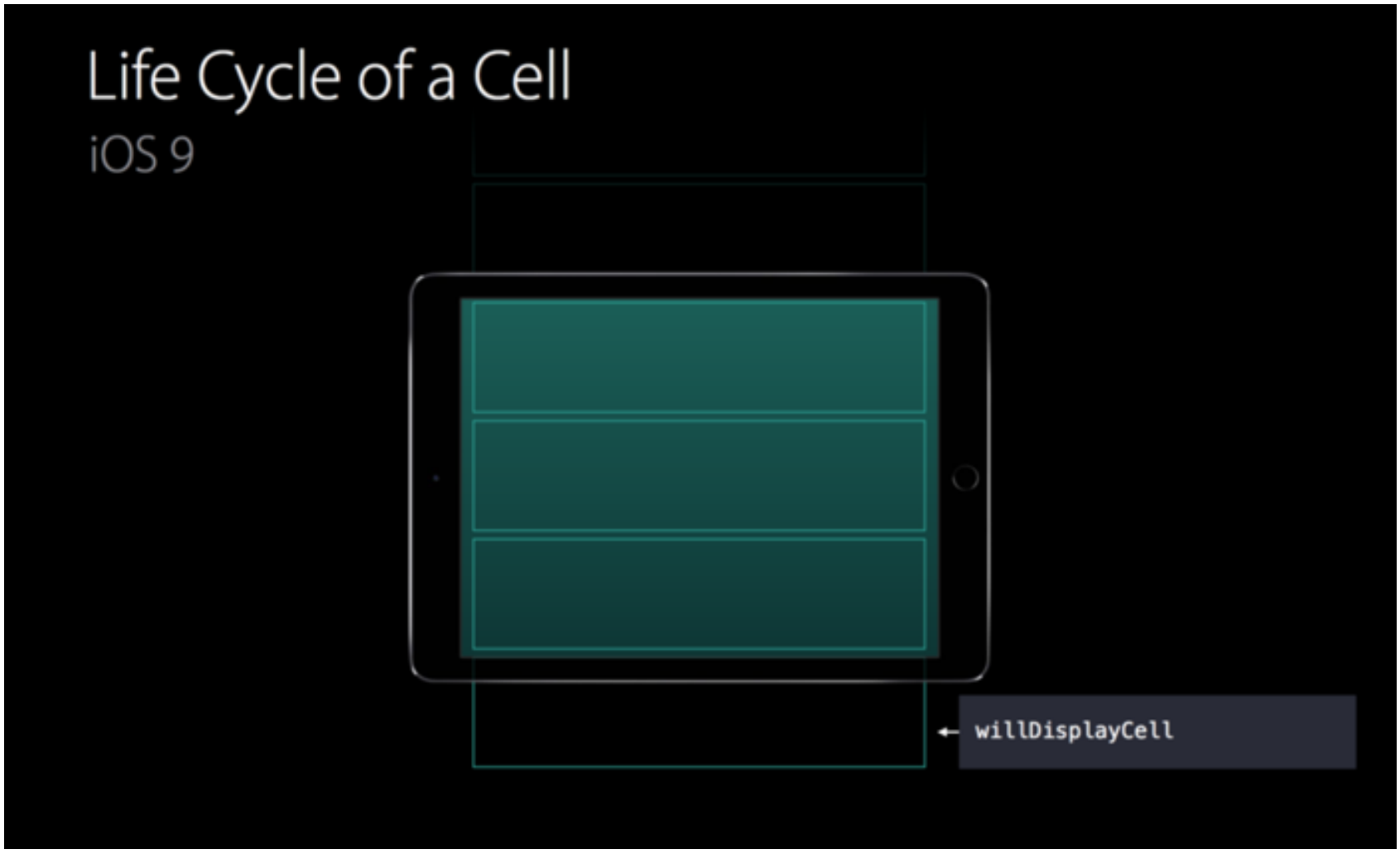


这个时候我们把这个cell从reuse队列里面拿出来，然后调用prepareForReuse方法。这个方法就给了cell时间，用来重置cell，重置状态，刷新cell，加载新的数据。

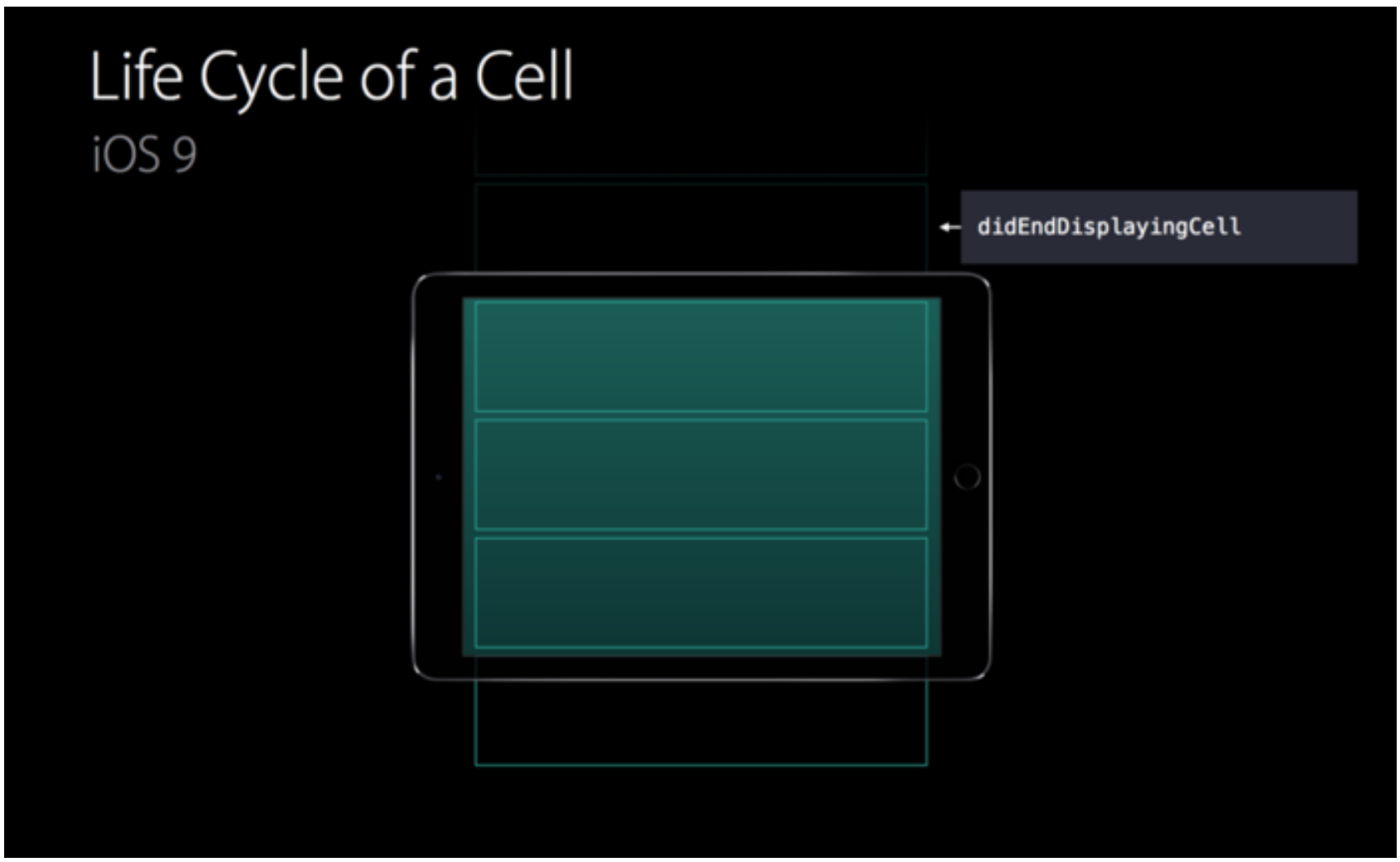


再滑动，我们就会调用cellForItemAtIndexPath方法了。这个方法里面就是我们开发者自定义的填充cell的方式了。这里会填充data model，然后赋值给

cell，再把cell返回给iOS系统。

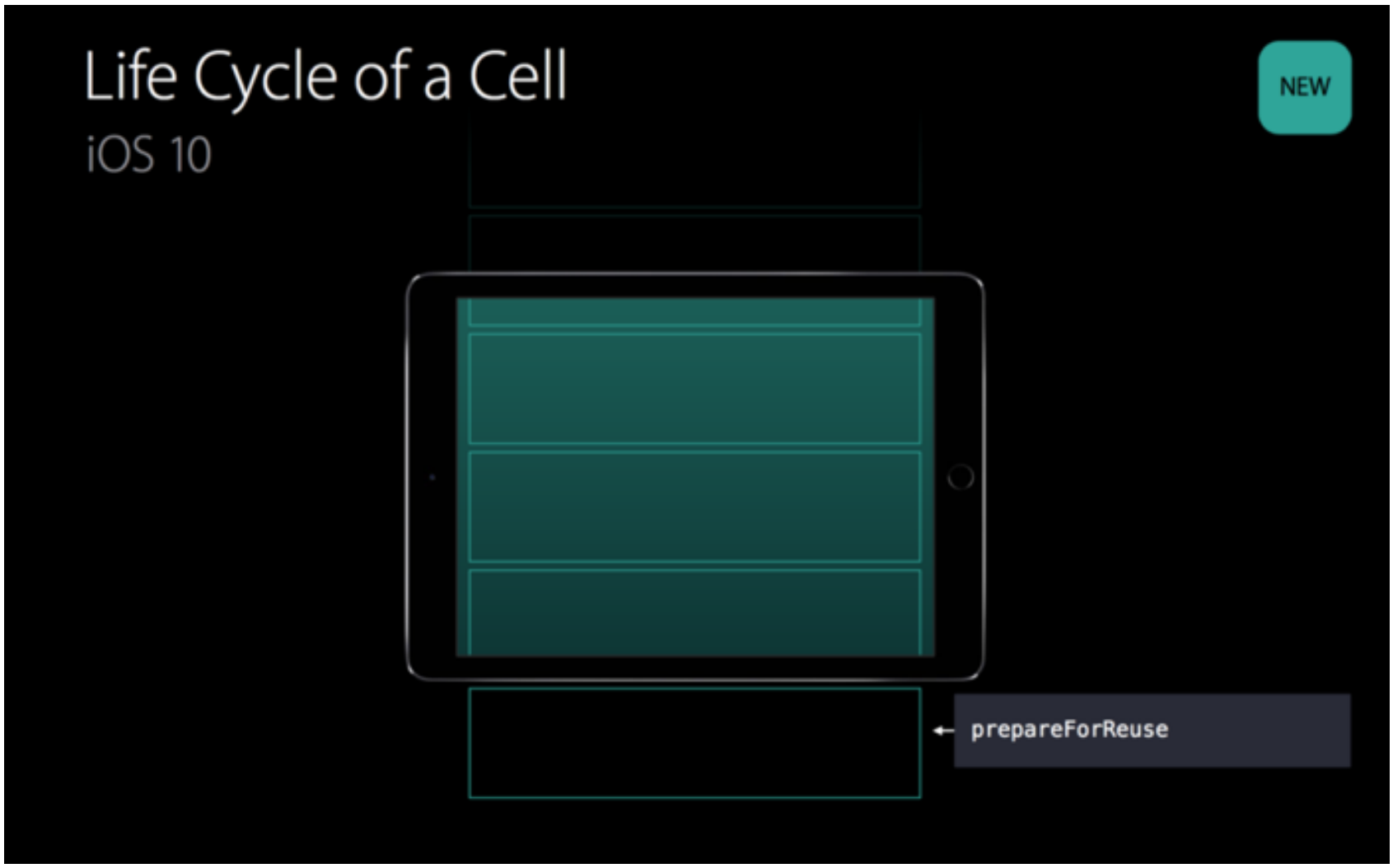


当cell马上要进入屏幕的时候，就会调用willDisplayCell的方法。这个方法给了我们app最后一次机会，为cell进入屏幕做最后的准备工作。执行完willDisplayCell之后，cell就进入屏幕了。

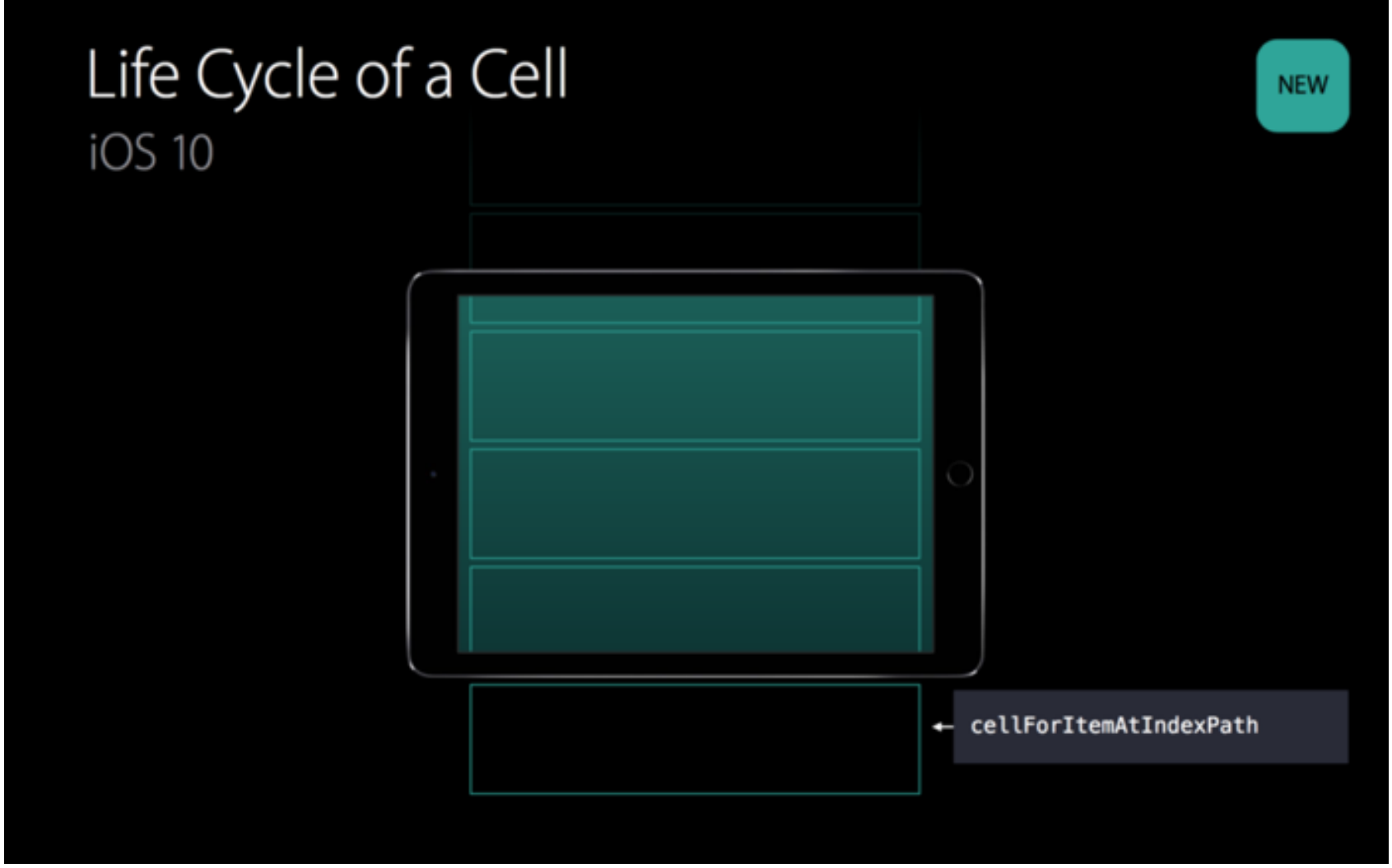


当cell完全离开屏幕之后，就会调用didEndDisplayingCell方法。以上就是在iOS10之前的整个UICollectionViewCell的生命周期。

接下来我们就来看看iOS 10的UICollectionViewCell生命周期是怎么样的。



这里还是和iOS9一样的，当用户滑动UICollectionView的时候，需要一个cell，我们就从reuse队列里面拿出一个cell，并调用prepareForReuse方法。注意调用这个方法的时间，当cell还没有进入屏幕的时候，就已经提前调用这个方法了。注意对比和iOS 9的区别，iOS 9 是在cell上边缘马上进入屏幕的时候才调用方法，而这里，cell整个生命周期都被提前了，提前到cell还在设备外面的时候。



这里还是和之前一样，在cellForItemAtIndexPath中创建cell，填充数据，刷新状态等等操作。注意，这里生命周期也比iOS 9提前了。

用户继续滑动，这个时候就有不同了！

这个时候我们并不去调用willDisplayCell方法了！这里遵循的原则是，何时去显示，何时再去调用willDisplayCell。

当cell要马上就需要显示的时候，我们再调用willDisplayCell方法。

当整个cell要从UICollectionView的可见区域消失的时候，这个时候会调用didEndDisplayingCell方法。接下来发生的事情和iOS9一样，cell会进入重用队列中。

如果用户想要显示某个cell，在iOS 9 当中，cell只能从重用队列里面取出，再次走一遍生命周期。并调用cellForItemAtIndexPath去创建或者生成一个cell。

在iOS 10 当中，系统会把cell保持一段时间。在iOS中，如果用户把cell滑出屏幕后，如果突然又想回来，这个时候cell并不需要再走一段的生命周期了。只需要直接调用willDisplayCell就可以了。cell就又会重新出现在屏幕中。这就是iOS 10 的整个UICollectionView的生命周期。

上面说的iOS 10里面的场景同样适用于多列的情况。 这时我们每次只加载一个cell，而不是每次加载一行的cell。当第一个cell准备好之后再叫第二个cell准备。当2个cell都准备好了之后，接着我们再调用willDisplayCell给每个cell，发送完这个消息之后，cell就会出现在屏幕上了。

这虽然看起来是一个很小的改动，但是这小小的改动就提升了很多的用户体验！

让我们来看看上述的改动对滑动的影响

滑动比iOS 9流程很多，这里可以看到整个过程都很平缓，不卡顿。

还是和iOS 9一样，我们来模拟一下系统是如何加载cell的情况。

我们可以很明显的看到，iOS 系统是一个个的加载cell的，一个cell加载完之后再去加载下一个cell。这里和iOS 9 的有很大的不同，iOS 9是加载整整一行的cell。

这是因为我们用了新的 UICollectionViewCell的生命周期。整个app完全没有加一行代码。现在iOS 10是丝滑的滑动体验实在是太棒了！！

二. UICollectionViewCell的Pre-Fetching预加载

当我们编译iOS 10的app的时候，这个Pre-Fetching默认是enable的。当然，如果有一些原因导致你必须用到iOS 10之前老的生命周期，你只需要给collectionView加入新的isPrefetchingEnabled属性即可。如果你不想用到Pre-Fetching，那么把这个属性变成false即可。

```
collectionView.isPrefetchingEnabled = false
```

为了最佳实践一下这个新特性。我们先改变一下我们加载cell的方式。我们把很重的读取数据的操作，所有内容的创建都放到cellForItemAtIndexPath方法里面去完成。保证我们在willDisplayCell 和 didEndDisplayCell这两个方法里面基本不做其他事情。最后，需要注意的是cellForItemAtIndexPath生成的某些cell，可能永远都不会被展示在屏幕上，有这样一种情况，当cell将要展示在屏幕上的时候，用户突然滑动离开了这个界面。

如果这个时候当你用iOS 10编译出你的app，那么非常顺滑的用户体验就会自动的优化出来。

UICollectionView的流畅的滑动解决了，那么在UICollectionViewCell在加载的时候所花费的时间，怎么解决呢？？

UICollectionViewCell加载的时间取决于DataModel。DataModel很可能回去加载图片，来自于网络或者来自于本地的数据库。这些操作大多数都是异步的操作。为了使data加载更快，iOS 10引入了新的API来解决这个问题。

UICollectionView有2个“小伙伴”，那就是data source和delegate。在iOS 10中，将会迎来第3个“小伙伴”。这个“小伙伴”叫prefetchDataSource。

```
protocol UICollectionViewDataSourcePrefetching {
    func collectionView(_ collectionView: UICollectionView,
                        prefetchItemsAt indexPaths: [NSIndexPath])
    optional func collectionView(_ collectionView: UICollectionView,
                                 cancelPrefetchingForItemsAt indexPaths: [N
}
class UICollectionView : UIScrollView {
    weak var prefetchDataSource: UICollectionViewDataSourcePrefetching?
    var isPrefetchingEnabled: Bool
}
```

这个协议里面只有一个必须要实现的方法——ColletionView prefetchItemsAt indexPaths。这个方法会在prefetchDataSource里面被调用，用来给你异步的预加载数据的。indexPaths数组是有序的，就是接下来item接收数据的顺序，让我们model异步处理数据更加方便。

在这个协议里面还有第二个方法CollectionView cancelPrefetcingForItemsAt indexPaths，不过这个方法是optional的。我们可以利用这个方法来处理在滑动中取消或者降低提前加载数据的优先级。

值得说明的是，新增加的这个“小伙伴”prefetchDataSource并不能代替原来的读取数据的方法，这个预加载仅仅只是辅助加载数据，并不能 删除原来我们读取数据的方法。

至此，我们来看看从文章开始到现在，UICollectionView的性能提升了多少。我们还是用掉帧的方法来看看UICollectionView的性能。

上图是iOS 9 UICollectionView的性能，很明显的看见，波峰波谷很明显，并且还掉了8帧，有明显的卡顿现象。

上图是iOS 10 UICollectionView的性能，我们可以很明显的看到，经过iOS 10的优化，整个曲线很明显平缓了一些，没有极端的波峰掉帧现象。但是依旧存在少量的波峰快到16ms分界线了。

上图是iOS 10 + Pre-Fetching API 之后的性能，已经优化的效果很明显了！整条曲线基本都水平了。近乎完美。但是还是能发现有个别波峰特别高。波峰特别高的地方就是那个cell加载压力大，时间花的比较长导致的。接下来我们继续优化！

先来总结一下使用Pre-Fetching API需要注意的地方。

1. 在我们使用Pre-Fetching API的时候，我们一定要保证整个预加载的过程都放在后台线程中进行。合理使用GCD 和 NSOperationQueue处理好多线程。

2. 请切记，Pre-Fetching API是一种自适应的技术。何为自适应技术呢？当我们滑动速度很慢的时候，在这种“安静”的时期，Pre-Fetching API会默默的在后台帮我们预加载数据，但是一旦当我们快速滑动，我们需要频繁的刷新，我们不会去执行Pre-Fetching API。

3. 最后，用cancelPrefetchingAPI去迎合用户的滑动动作的变换，比如说用户在快速滑动突然发现了有趣的感兴趣的事情，这个时候停下来滑动了，甚至快速反向滑动了，或者点击了事件，进去看详情了，这些时刻我们都应该开启cancelPrefetchingAPI。

综上所述，Pre-Fetching API对于提高UICollectionView的性能提升是很有帮助的，而且并不需要加入太多的代码。加入少量的代码就可以获得巨大的性能提升！

三. UITableViewCell的Pre-Fetching预加载

在iOS 10中，UITableViewCell也跟着UICollectionView一起得到了性能的提升，一样拥有了Pre-Fetching API。

```
protocol UITableViewDataSourcePrefetching {
    func tableView(_ tableView: UITableView, prefetchRowsAt indexPaths: [NS
    optional func tableView(_ tableView: UITableView, cancelPrefetchingForR
                                [NSIndexPath])
}
class UITableView : UIScrollView {
    weak var prefetchDataSource: UITableViewDataSourcePrefetching?
}
```

这里和上面 UICollectionView一样，会调用TableView prefetchRowsAt indexPaths方法。indexPaths还是一个有序数字，顺序就是列表上可见的顺序。第二个可选的API还是TableView cancelPrefetchingForRowsAt indexPaths，和之前提到的一样，也是用来取消预加载的。性能的提升和UICollectionView一样的，对UITableView的性能提升很大！

四. 针对self-sizing的改进

self-sizing API 第一次被引入是在iOS 8，然而现在在iOS 10中得到了一些改进。

在UICollectionView 中有一个固定的类，叫UICollectionViewFlowLayout，iOS已经在这个类中完全支持了self-sizing。为了能开启这一特性，需要我们开发者为一些不能为0的CGSize的cell设置一下estimated item size。

```
layout.estimatedItemSize = CGSize(width:50,height:50)
```

这会告诉UICollectionView我们想要开启动态计算内容的布局。

至今，我们能有3种方法来动态的布局。

1. 第一种方法是使用autolayout 当我们合理的加上了constrain，当cell加载的时候，就会根据内容动态的加载布局。

2. 第二种方法，如果你不想使用autolayout的方法，想更加手动的控制它，那么我们就需要重写sizeThatFits()方法。

3. 第三种方法，终极的方法是重写

preferredLayoutAttributesFittingAttributes()方法。在这个方法里面不仅仅可以提供size的信息，更可以得到alpha和transform的信息。

所以想指定cell的大小，就可以用上面3个方法之一。

但是实际操作中，我们可以发现，有时候设置一个合适的estimated item size，对于我们来说是很困难的事情。如果flow layout可以用数学的方法动态的计算布局，而不是根据我们给的size去布局，那会是件很酷的事情。

iOS 10中就引入了新的API来解决上述的问题。

```
layout.estimatedItemSize = UICollectionViewFlowLayoutAutomaticSize
```

对于开发者，我们需要做的事情，仅仅就是设置好flow layout ，然后给estimatedItemSize设定一个新的常数, 最后UICollectionViewFlowLayout 就会自动计算高度了。

系统会自动计算好所有的布局，包括已经定下来的size的cell，并且还会动态的给出接下来cell的大小的预测。

接下来看2个例子就可以很明显看出iOS 10针对self-sizing的改进了。

上图可以看到，iOS 9 的布局是针对单个cell计算的，当改变了单个的cell，其他的cell依旧没有变化，还是需要重新计算。

这里例子就可以很明显的看出差别了。当我们改变了第一个cell的size以后，系统会自动计算出所有的cell的size，并且每一行，每一个section的size都会被动态的计算出来，并且刷新界面！

以上就是iOS 10针对self-sizing的改进。

五. Interactive Reordering

谈到重新排列，这是我们就需要类比一下UITableView了，UICollectionView的重新排列就如同UITableView 把cell上下移动，只不过UITableView的重排是针对垂直方向的。

在iOS 9中，引入了UICollectionView的Interactive Reordering，在今年的iOS

10中，又加入了一些新的API。

在上图中，我们可以看到，我们即使任意拖动cell，整个界面也会重新排列，并且我们改变了cell的大小，整个 UICollectionView 也会重新动态的布局。

我们先来看看iOS 9里面的API

```
class UICollectionView : UIScrollView {
    func beginInteractiveMovementForItem(at indexPath: NSIndexPath) -> Bool
    func updateInteractiveMovementTargetPosition(_ targetPosition: CGPoint)
    func endInteractiveMovement()
    func cancelInteractiveMovement()
}
```

要想开启interactive movement，我们就需要调用beginInteractiveMovementForItem()方法，其中indexPath代表了我们将要移动走的cell。接着每次手势的刷新，我们都需要刷新cell的位置，去响应我们手指的移动操作。这时我们就需要调用updateInteractiveMovementTargetPosition()方法。我们通过手势来传递坐标的变化。当我们移动结束之后，就会调用endInteractiveMovement()方法。UICollectionView 就会放下cell，处理完整个layout，此时你也可以重新刷新model或者处理数据model。如果中间突然手势取消了，那么这个时候就应该调用cancelInteractiveMovement()方法。如果我们重新把cell移动一圈之后又放回原位，其实就是取消了移动，那这个时候就应该在cancelInteractiveMovement()方法里面不用去刷新data source。

在iOS 10中，如果你使用UICollectionViewController，那么这个重排对于你来说会更加的简单。

```
class UICollectionViewController : UIViewController {
    var installsStandardGestureForInteractiveMovement: Bool
}
```

你只需要把installsStandardGestureForInteractiveMovement这个属性设置为True即可。CollectionViewController会自动为你加入手势，并且自动为你调

用上面的方法。

以上就是去年iOS 9为我们增加的API。

今年的iOS 10新加入的API是在iOS 9的基础上增加了翻页的功能。UICollectionView继承自UIScrollView，所以只需要你做的是把isPagingEnabled属性设置为True，即可开启分页的功能。

```
collectionView.isPagingEnabled = true
```

开启分页之前：

开启分页之后就长这样子：

每次移动一次就会以页为单位的翻页。

六.UIRefreshControl

UIRefreshControl现在可以直接在CollectionView里面使用，同样的，也可以直接在UITableView里面使用，并且可以脱离UITableViewController。因为现在RefreshControl成为了ScrollView的一个属性了。

UIRefreshControl的使用方法很简单，就三步：

```
let refreshControl = UIRefreshControl()
refreshControl.addTarget(self, action: #selector(refreshControlDidFire(_:))
                         for: .valueChanged)
collectionView.refreshControl = refreshControl
```

先创建一个refreshControl，再关联一个action事件，最后把这个新的refreshControl赋给想要的控件的对应的属性即可。

总结

通过以上，我们谈到了以下的知识：

1. UICollectionView cell pre-fetching预加载机制

2. UICollectionView and UITableView prefetchDataSource 新增的API
3. 针对self-sizing cells 的改进
4. Interactive reordering

最后，谈谈我看了iOS 10 UICollectionView的优化的看法吧，原来有些地方用到AsyncDisplayKit优化UICollectionView速度的，现在可以考虑不用第三方库优化了，系统自带的方法可以解决一般性的卡顿的问题了。我感觉iOS 10的UICollectionView才像是一个完整版的，之前的系统优化的都不够。我还是很看好iOS 10的UICollectionView。

转自：https://juejin.im/post/57b012d16be3ff006ba7e184