集合视图

在视图中显示基于单元格的可滚动视图，并采用任何布局。

在iOS 6之前，要实现网格布局，要么用表示图，要么用自定义的逻辑创建视图加到滚动视图上。实现困难，耗时，耗内存，容易出错。集合视图解决了这些问题，它提供单元格管理架构，同时将单元格布局抽象出来。

集合视图提供了一种默认布局方式-流试布局，用于快速轻松实现众多常见的网格试布局，并支持水平和垂直布局。要实现特殊的网格布局或非网格布局，可创建自定义布局。

可将集合视图划分成多个部分，并且添加section header和section footer，另外还可以指定装饰视图让集合视图更美观。

集合视图支持各种动画，包括自定义单元格状态以及布局过渡动画。

简介

集合视图在多个类的支持下实现，最主要的是UICollectionView,它是UIScrollView的子类。

遵守UICollectionViewDataSource的类提供单元格（UICollectionViewCell），如果集合视图包含标题或注角，数据源还提供配置好的UICollectionReusableView实例。

遵守UICollectionViewDelegate的类,它负责管理单元格选择和突出工作。

另外UICollectionViewLayout提供UICollectionView的布局

UICollectionView绑定单元格、布局对象、附加视图

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

    self.collectionView.delegate = self;
    self.collectionView.dataSource = self;
    self.collectionView.backgroundColor = [UIColor whiteColor];

    UICollectionViewFlowLayout *flowLayout = [[UICollectionViewFlowLayout
 alloc] init];
    flowLayout.itemSize = CGSizeMake(50, 50);
    flowLayout.minimumInteritemSpacing = 10;
    flowLayout.minimumLineSpacing = 10;
    flowLayout.sectionInset = UIEdgeInsetsMake(10, 10, 10, 10);
    flowLayout.headerReferenceSize = CGSizeMake(320, 30);
    flowLayout.footerReferenceSize = CGSizeMake(320, 30);
    self.collectionView.collectionViewLayout = flowLayout;


    [self.collectionView registerNib:[UINib nibWithNibName:@"QYImageCell"
 bundle:[NSBundle mainBundle]] forCellWithReuseIdentifier:@"QYImageCell"]
;

    [self.collectionView registerNib:[UINib nibWithNibName:@"QYCollection
HeaderView" bundle:[NSBundle mainBundle]] forSupplementaryViewOfKind:UICo
llectionElementKindSectionHeader withReuseIdentifier:@"QYCollectionHeader
View"];
    [self.collectionView registerNib:[UINib nibWithNibName:@"QYCollection
FooterView" bundle:[NSBundle mainBundle]] forSupplementaryViewOfKind:UICo
llectionElementKindSectionFooter withReuseIdentifier:@"QYCollectionFooter
View"];
}
```

快速创建集合视图

UICollectionViewController遵守了UICollectionViewDataSource和UICollectionViewDelegate协议，但是如果控制器中只需要UICollectionView，使用该控制器就比较方便，可以继承该类，用于显示网格布局。在IB中注册各个类以及设置各种属性，相应的在代码中也可以。

集合视图数据源方法

实现数据源的方法：

返回集合视图有几个Section

```
-(NSInteger)numberOfSectionsInCollectionView:(UICollectionView *)collectionView{
    return 5;
}
```

返回每个组有几个item

```

-(NSInteger)collectionView:(UICollectionView *)collectionView numberOfItemsInSection:(NSInteger)section{
    return 8;
}
```

返回headerView或footerView

```

-(UICollectionReusableView *)collectionView:(UICollectionView *)collectionView viewForSupplementaryElementOfKind:(NSString *)kind atIndexPath:(NSIndexPath *)indexPath{

    UICollectionReusableView *supplementary = nil;

    if ([kind isEqualToString:UICollectionElementKindSectionHeader]) {
        QYHeaderView *header = [collectionView dequeueReusableSupplementaryViewOfKind:UICollectionElementKindSectionHeader withReuseIdentifier:@"QYHeaderView" forIndexPath:indexPath];
        header.nameLabel.text = [NSString stringWithFormat:@"Group %d Header", indexPath.section];
        supplementary = header;
    }else{
        QYFooterView *foot = [collectionView dequeueReusableSupplementaryViewOfKind:UICollectionElementKindSectionFooter withReuseIdentifier:@"QYFooterView" forIndexPath:indexPath];
        foot.nameLabel.text = [NSString stringWithFormat:@"Group %d Footer", indexPath.section];
        supplementary = foot;
    }
    return supplementary;

}
```

返回集合视图元素的方法

```
-(UICollectionViewCell *)collectionView:(UICollectionView *)collectionVie
w cellForItemAtIndexPath:(NSIndexPath *)indexPath{
    static NSString *identifier = @"QYImageCell";
    QYImageCell *item = [collectionView dequeueReusableCellWithReuseIdent
ifier:identifier forIndexPath:indexPath];
    item.imageView.image = [UIImage imageNamed:[NSString stringWithFormat
:@"%d.jpg", indexPath.row + 1]];
    return item;
}
```

集合视图的委托方法

可处理单元格选择和突出工作、跟踪单元格或部分的删除以及为单元格显示Edit菜单并执行菜单中的操作。 集合单元格有三个子视图层：contentView，selectdBackgroundView, backgroundView,后两者可以自定义。 单元格选择默认只支持单个，可通过 `[self.collectionView setAllowsMultipleSelection:YES];` 方法实现多选。

将被选择或将被取消或已经被选择或已经被取消

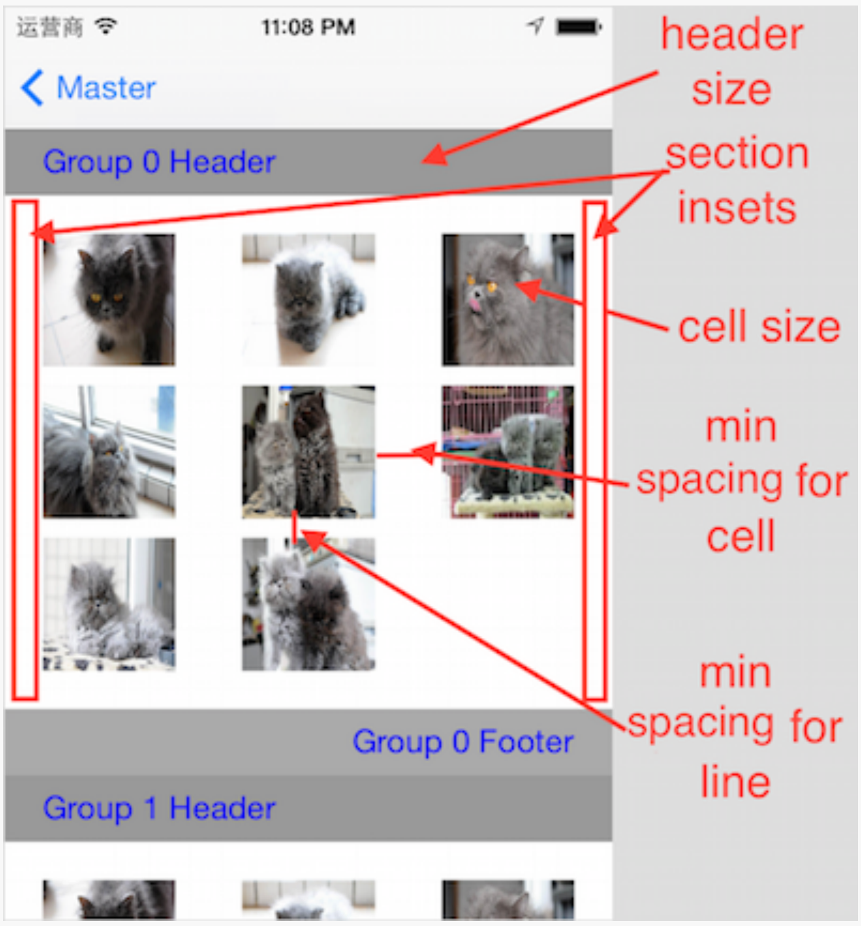
```

-(BOOL)collectionView:(UICollectionView *)collectionView shouldSelectItem
AtIndexPath:(NSIndexPath *)indexPath{
    return YES;
}

-(BOOL)collectionView:(UICollectionView *)collectionView shouldDeselectIt
emAtIndexPath:(NSIndexPath *)indexPath{
    return YES;
}

- (void)collectionView:(UICollectionView *)collectionView
didSelectItemAtIndexPath:(NSIndexPath *)indexPath
{
    NSLog(@"Item selected at indexPath: %@",indexPath);
}

- (void)collectionView:(UICollectionView *)collectionView
didDeselectItemAtIndexPath:(NSIndexPath *)indexPath
{
    NSLog(@"Item deselected at indexPath: %@",indexPath);
}
```

定制集合视图和流式布局

可以定制这些尺寸：单元格的尺寸，header和footer的尺寸，单元格之间、单元格与header和footer之间以及单元格与边界的距离不小于指定值；还可以再集合视图的任意位置放置装饰视图

基本定制

UICollectionViewFlowLayout参数说明



- 通过IB中设置UICollectionViewFlowLayout的属性
- 通过代码直接修改
- 在集合视图的委托中实现UICollectionViewDelegateFlowLayout

自定义布局

不太适合使用网格布局的时候，可为它创建自定义布局。通过创建UICollectionViewLayout子类实现自定义布局。

需要实现多个方法：

`prepareLayout 可选，非常适合用于计算布局属性，因为每当需要更新布局时都将调用它。`

```
- (void)prepareLayout
{
    NSInteger numSections = [self.collectionView numberOfSections];

    CGFloat currentYPosition = 0.0;
    self.centerPointsForCells = [[NSMutableDictionary alloc] init];
    self.rectsForSectionHeaders = [[NSMutableArray alloc] init];

    for (NSInteger sectionIndex = 0; sectionIndex < numSections;
         sectionIndex++)
    {
        CGRect rectForNextSection = CGRectMake(0, currentYPosition,
        self.collectionView.bounds.size.width, kSectionHeight);

        self.rectsForSectionHeaders[sectionIndex] =
        [NSValue valueWithCGRect:rectForNextSection];

        currentYPosition +=
        kSectionHeight + kVerticalSpace + kCellSize / 2;

        NSInteger numCellsForSection =
        [self.collectionView numberOfItemsInSection:sectionIndex];

        for (NSInteger cellIndex = 0; cellIndex < numCellsForSection;
             cellIndex++)
        {
            CGFloat xPosition = (cellIndex%2 + 1) * 100;

            CGPoint cellCenterPoint =
            CGPointMake(xPosition, currentYPosition);

            NSIndexPath *cellIndexPath = [NSIndexPath
            indexPathForItem:cellIndex inSection:sectionIndex];

            self.centerPointsForCells[cellIndexPath] =
            [NSValue valueWithCGPoint:cellCenterPoint];

            currentYPosition += kCellSize + kVerticalSpace;
        }
    }

    self.contentSize =
    CGSizeMake(self.collectionView.bounds.size.width,
               currentYPosition + kVerticalSpace);
}
```

`collectionViewContentSize` 告诉集合视图整个可滚动视图有多大。

```
- (CGSize)collectionViewContentSize
{
    return self.contentSize;
}
```

`layoutAttributesForElementsInRect:` 将指定举行内单元格、部分标题和部分注角以及装饰视图的所有布局属性告诉集合视图。

```
-(NSArray*)layoutAttributesForElementsInRect:(CGRect)rect
{
    NSMutableArray *attributes = [NSMutableArray array];
    for (NSValue *sectionRect in self.rectsForSectionHeaders)
    {
        if (CGRectIntersectsRect(rect, sectionRect.CGRectValue))
        {
            NSInteger sectionIndex =
            [self.rectsForSectionHeaders indexOfObject:sectionRect];

            NSIndexPath *secIndexPath =
            [NSIndexPath indexPathForItem:0 inSection:sectionIndex];

            [attributes addObject:
             [self layoutAttributesForSupplementaryViewOfKind:
              UICollectionElementKindSectionHeader
              atIndexPath:secIndexPath]];
        }
    }

    [self.centerPointsForCells enumerateKeysAndObjectsUsingBlock:
     ^(NSIndexPath *indexPath, NSValue *centerPoint, BOOL *stop) {

        CGPoint center = [centerPoint CGPointValue];

        CGRect cellRect = CGRectMake(center.x - kCellSize/2,
        center.y - kCellSize/2, kCellSize, kCellSize);

        if (CGRectIntersectsRect(rect, cellRect)) {
            [attributes addObject:
            [self layoutAttributesForItemAtIndexPath:indexPath]];
        }
    }];

    return [NSArray arrayWithArray:attributes];
}
```

方法`layoutAttributesForItemAtIndexPath:` 将返回指定索引路径处单元格的布局属性。

```
- (UICollectionViewLayoutAttributes *)layoutAttributesForItemAtIndexPath:
(NSIndexPath *)path
{
    UICollectionViewLayoutAttributes *attributes =
    [UICollectionViewLayoutAttributes
     layoutAttributesForCellWithIndexPath:path];

    attributes.size = CGSizeMake(kCellSize, kCellSize);
    NSValue *centerPointValue = self.centerPointsForCells[path];
    attributes.center = [centerPointValue CGPointValue];
    return attributes;
}
```

`layoutAttributesForSupplementaryViewOfKind:atIndexPath:` 返回指定索引路径处标题或注释的布局属性。如果没有实现标题或注释，就不用实现该方法。

```
- (UICollectionViewLayoutAttributes *)layoutAttributesForSupplementaryVie
wOfKind:(NSString *)kind atIndexPath:(NSIndexPath *)indexPath
{
    UICollectionViewLayoutAttributes *attributes =
    [UICollectionViewLayoutAttributes
     layoutAttributesForSupplementaryViewOfKind:
     UICollectionElementKindSectionHeader withIndexPath:indexPath];

    CGRect sectionRect =
    [self.rectsForSectionHeaders[indexPath.section] CGRectValue];

    attributes.size =
    CGSizeMake(sectionRect.size.width, sectionRect.size.height);

    attributes.center =
    CGPointMake(CGRectGetMidX(sectionRect),
                CGRectGetMidY(sectionRect));

    return attributes;
}
```

`layoutAttributesForDecorationViewOfKind:atIndexPath:` 返回指定索引路径处装饰视图的布局属性。如果集合视图没有实现装饰视图，就不用实现这个方法。

`shouldInvalidateLayoutForBoundsChange:` 用于实现滚动动画。如果这个方法返回YES，集合视图将重新计算可见区域内所有所有内容的布局属性，这使得可以根据内容在屏幕上位置修改其布局属性。

装饰视图

可以放在集合视图的任何位置，需要创建UICollectionViewLayout子类，以计算装饰视图的布局属性，并在他们可见的时将它们加入集合视图。因为装饰视图和数据源没有任何关系，所以，其初始化不再有UICollectionView管理。

首先要向UICollectionViewFlowLayout注册一个类或nib，这个类是UICollectionReusableView的子类，

```
 [self registerClass:[QYRowDecorationView class] forDecorationViewOfKind:
[QYRowDecorationView kind]];
    }
```

然后在prepareLayout方法中计算修饰视图的frame

```
- (void)prepareLayout
{
    [super prepareLayout];

    NSInteger sections = [self.collectionView numberOfSections];

    CGFloat availableWidth = self.collectionViewContentSize.width -
    (self.sectionInset.left + self.sectionInset.right);

    NSInteger cellsPerRow =
    floorf((availableWidth + self.minimumInteritemSpacing) /
           (self.itemSize.width + self.minimumInteritemSpacing));

    NSMutableDictionary *rowDecorationWork =
    [[NSMutableDictionary alloc] init];

    CGFloat yPosition = 0;

    for (NSInteger sectionIndex = 0; sectionIndex < sections; sectionInde
x++)
    {
        yPosition += self.headerReferenceSize.height;
        yPosition += self.sectionInset.top;

        NSInteger cellCount =
        [self.collectionView numberOfItemsInSection:sectionIndex];

        NSInteger rows = ceilf(cellCount/(CGFloat)cellsPerRow);
        for (int row = 0; row < rows; row++)
        {
            yPosition += self.itemSize.height;

            CGRect decorationFrame = CGRectMake(0,
                                                yPosition-kDecorationYAdj
ustment,
                                                self.collectionViewConten
tSize.width,
                                                kDecorationHeight);

            NSIndexPath *decIndexPath = [NSIndexPath
                                         indexPathForRow:row inSection:se
ctionIndex];

            rowDecorationWork[decIndexPath] =
            [NSValue valueWithCGRect:decorationFrame];

            if (row < rows - 1)
                yPosition += self.minimumLineSpacing;
        }

        yPosition += self.sectionInset.bottom;
        yPosition += self.footerReferenceSize.height;
    }

    self.rowDecorationRects =
    [NSDictionary dictionaryWithDictionary:rowDecorationWork];
}
```

在layoutAttributesForElementsInRect方法中返回装饰视图的基本属性

```
- (NSArray *)layoutAttributesForElementsInRect:(CGRect)rect
{
    //调用父方法，布局其它元素
    NSArray *layoutAttributes =
    [super layoutAttributesForElementsInRect:rect];

    for (UICollectionViewLayoutAttributes *attributes
         in layoutAttributes)
    {
        attributes.zIndex = 1;
    }

    NSMutableArray *newLayoutAttributes =
    [layoutAttributes mutableCopy];

    [self.rowDecorationRects enumerateKeysAndObjectsUsingBlock:
     ^(NSIndexPath *indexPath, NSValue *rowRectValue, BOOL *stop) {

         if (CGRectIntersectsRect([rowRectValue CGRectValue], rect))
         {
             UICollectionViewLayoutAttributes *attributes =
             [UICollectionViewLayoutAttributes
              layoutAttributesForDecorationViewOfKind:
              [QYRowDecorationView kind] withIndexPath:indexPath];

             attributes.frame = [rowRectValue CGRectValue];
             attributes.zIndex = 0;
             [newLayoutAttributes addObject:attributes];
         }
     }];

    layoutAttributes = [NSArray arrayWithArray:newLayoutAttributes];

    return layoutAttributes;
}
```

集合视图动画

集合内容布局变更动画

```
void (^finishedBlock)(BOOL) = ^(BOOL finished) {

       };

[self.collectionView setCollectionViewLayout:customLayout
                                            animated:YES
                                          completion:finishedBlock];
```

集合视图滚动动画

```
- (BOOL)shouldInvalidateLayoutForBoundsChange:(CGRect)oldBounds
{
    return YES;
}

- (NSArray *)layoutAttributesForElementsInRect:(CGRect)rect
{
    NSArray *layoutAttributes =
    [super layoutAttributesForElementsInRect:rect];

    CGRect visibleRect;
    visibleRect.origin = self.collectionView.contentOffset;
    visibleRect.size = self.collectionView.bounds.size;

    for (UICollectionViewLayoutAttributes *attributes
         in layoutAttributes)
    {
        if (attributes.representedElementCategory ==
            UICollectionElementCategoryCell &&
            CGRectIntersectsRect(attributes.frame, rect))
        {
            CGFloat distanceFromCenter =
            CGRectGetMidY(visibleRect) - attributes.center.y;

            CGFloat distancePercentFromCenter =
            distanceFromCenter / kZoomDistance;

            if (ABS(distanceFromCenter) < kZoomDistance) {
                CGFloat zoom =
                1 + kZoomAmount * (1 - ABS(distancePercentFromCenter));

                attributes.transform3D =
                CATransform3DMakeScale(zoom, zoom, 1.0);
            }
            else
            {
                attributes.transform3D = CATransform3DIdentity;
            }
        }
    }

    return layoutAttributes;
}
```