第 3 章　图　　表

在人口普查等统计信息时，只有图或者表，往往很难直观地展示统计信息的属性，如时间性、数量性等。这时，引进了新的概念——图表。最常见的图表有饼状图、柱状图、折线图等。本章将主要讲解关于图表的一些实例。

实例 40　饼　状　图

【实例描述】

饼状图是一个划分为几个扇形的圆形统计图表，用于描述量、频率或百分比之间的相对关系。在饼图中，每个扇区的弧长（以及圆心角和面积）大小为其所表示的数量的比例。本实例主要的功能就是绘制这么一个饼状图，用户通过第一个滑块控件，可以将饼状图变为一个圆环效果的图。用户通过第二个滑块控件，可以将饼状图中的扇形改变。运行效果如图 3.1 所示。

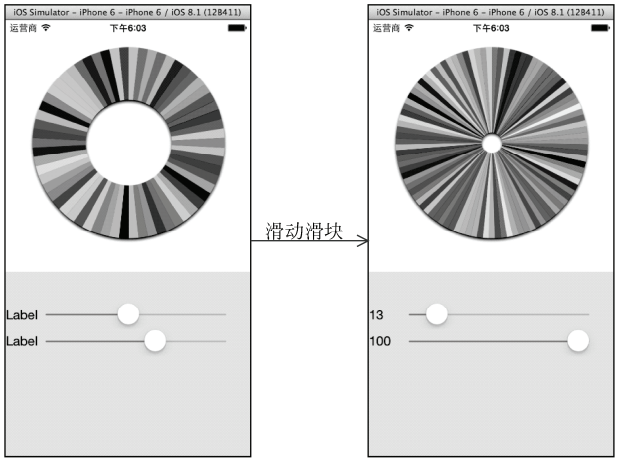

图 3.1　运行效果

【实现过程】

（1）创建一个项目，命名为“饼状图”。

（2）创建一个基于 UIView 类的 PieView 类。

（3）打开 PieView.h 文件，编写代码，实现头文件、协议、属性以及方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@class PieView;
```

```
// PieViewDelegate 协议
@protocol PieViewDelegate <NSObject>
- (CGFloat)centerCircleRadius;
@end
// PieViewDataSource 协议
@protocol PieViewDataSource <NSObject>
@required
- (int)numberOfSlicesInPieChartView:(PieView *)pieChartView;  //获取切片个数
- (double)pieChartView:(PieView *)pieChartView valueForSliceAtIndex: (NSU
Integer)index;                                                //获取切片的值
- (UIColor*)pieChartView:(PieView*)pieChartView colorForSliceAtIndex:(NSU
Integer)index;                                                //获取切片的颜色
@optional
-(NSString*)pieChartView:(PieView*)pieChartViewtitleForSliceAtIndex:(NSU
Integer)index;
@end
@interface PieView : UIView
//属性
@property (nonatomic, assign) id <PieViewDataSource> datasource;
@property (nonatomic, assign) id <PieViewDelegate> delegate;
-(void)reloadData;                                            //加载数据
@end
```

（4）打开 PieView.m 文件，编写代码，实现饼状图的绘制。使用的方法如表 3-1 所示。

表 3-1　PieView.m文件中方法总结

方　法	功　能
initWithFrame:	初始化
reloadData	加载数据
drawRect:	绘制

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，initWithFrame:方法实现对饼状图的初始化。程序代码如下：

```
- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        self.backgroundColor = [UIColor clearColor];
        self.layer.shadowColor = [[UIColor blackColor] CGColor]; //设置阴影颜色
        self.layer.shadowOffset = CGSizeMake(0.0f, 2.5f); //获取阴影的偏移量
        self.layer.shadowRadius = 1.9f;
        self.layer.shadowOpacity = 0.9f;
    }
    return self;
}
```

drawRect:方法实现了对饼状图的绘制。程序代码如下：

```
- (void)drawRect:(CGRect)rect
{
    CGContextRef context = UIGraphicsGetCurrentContext();  //创建图形上下文
    CGFloat theHalf = rect.size.width/2;
    CGFloat lineWidth = theHalf;
    //判断是否执行了 centerCircleRadius 方法
    if ([self.delegate respondsToSelector:@selector(centerCircleRadius)])
```

```
    {
        lineWidth -= [self.delegate centerCircleRadius];
        NSAssert(lineWidth <= theHalf, @"wrong circle radius");
    }
    //为变量赋值
    CGFloat radius = theHalf-lineWidth/2;
    CGFloat centerX = theHalf;
    CGFloat centerY = rect.size.height/2;
    //绘制
    double sum = 0.0f;
    int slicesCount = [self.datasource numberOfSlicesInPieChartView:self];
                                    //获取切片个数
    //循环，求和
    for (int i = 0; i < slicesCount; i++)
    {
        sum += [self.datasource pieChartView:self valueForSliceAtIndex:i];
    }
    float startAngle = - M_PI_2;
    float endAngle = 0.0f;
    //循环，绘制切片
    for (int i = 0; i < slicesCount; i++)
    {
        double value = [self.datasource pieChartView:self valueForSlice
        AtIndex:i];
        endAngle = startAngle + M_PI*2*value/sum;
        CGContextAddArc(context, centerX, centerY, radius, startAngle, end
        Angle, false);       //添加弧
        UIColor  *drawColor = [self.datasource pieChartView:self colorFor
        Slice AtIndex:i];
        CGContextSetStrokeColorWithColor(context, drawColor.CGColor); //设置线的颜色
        CGContextSetLineWidth(context, lineWidth);          //设置线的宽度
        CGContextStrokePath(context);                       //绘制
        startAngle += M_PI*2*value/sum;
    }
}
```

（5）打开 ViewController.h 文件，编写代码，实现头文件、插座变量、对象以及动作的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import "PieView.h"
@interface ViewController : UIViewController<PieViewData Source, Pie View
Delegate>{
    //声明关于视图的插座变量
    IBOutlet UIView *vv;
    //声明关于滑块控件的插座变量
    IBOutlet UISlider *holeSlider;
    IBOutlet UISlider *slicesSlider;
    //声明关于标签的插座变量
    IBOutlet UILabel *holeLabel;
    IBOutlet UILabel *valueLabel;
    PieView *pieView;
}
- (IBAction)Change:(UISlider*)slider;
@end
```

（6）打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，

效果如图 3.2 所示。

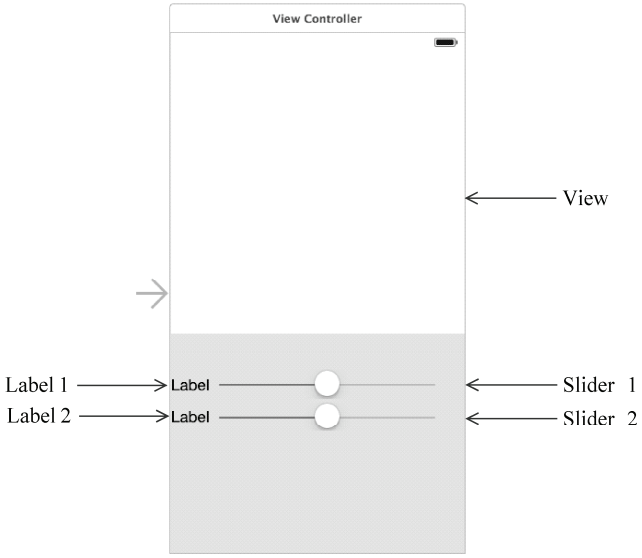

图 3.2　View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 3-2 所示。

表 3-2　视图、控件设置

视图、控件	属 性 设 置	其　　他
设计界面	Background：浅灰色	
View		与插座变量 vv 关联
Label1		与插座变量 holeLabel 关联
Label2		与插座变量 valueLabel 关联
Slider1		与插座变量 holeSlider 关联 与动作 Change:关联
Slider2		与插座变量 slicesSlider 关联 与动作 Change:关联

（7）打开 ViewController.m 文件，编写代码，实现将绘制的饼状图添加到界面中，并实现对饼状图的控制。使用的方法如表 3-3 所示。

表 3-3　ViewController.m文件中方法总结

方　　法	功　　能
UIColor ∗GetRandomUIColor	获取颜色
viewDidLoad	视图加载后调用，实现初始化
centerCircleRadius	获取中心圆的半径
numberOfSlicesInPieChartView:	获取切片个数
pieChartView:colorForSliceAtIndex:	获取切片的颜色
pieChartView:valueForSliceAtIndex:	获取切片的值
Change:	滑动滑块

其中，UIColor ∗GetRandomUIColor 方法实现了对随机产生颜色的获取。程序代码如下：

```
static inline UIColor *GetRandomUIColor()
{
    //生成随机值
    CGFloat r = arc4random() % 255;
    CGFloat g = arc4random() % 255;
    CGFloat b = arc4random() % 255;
    UIColor * color = [UIColor colorWithRed:r/255 green:g/255 blue:b/255
     alpha:1.0f];
    return color;                    //返回颜色
}
```

viewDidLoad 方法实现的功能是将绘制的具有饼状图的视图添加到界面中，以及滑块控件的初始化。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
     // Do any additional setup after loading the view, typically from a nib.
    pieView = [[PieView alloc] initWithFrame:CGRectMake(35.0f, 35.0f,250.0f,
250.0f)];
    pieView.delegate = self;                          //设置协议
    pieView.datasource = self;                        //设置委托
    [vv addSubview:pieView];
    holeSlider.tag = 7;                               //设置 tag 值
    holeSlider.minimumValue = 0.0f;
    holeSlider.maximumValue = 250/2 - 1;              //设置最大值
    int max = holeSlider.maximumValue;
    holeSlider.value = arc4random() % max;
    slicesSlider.tag = 77;                            //设置 tag 值
    slicesSlider.minimumValue = 0.0f;                 //设置最小值
    slicesSlider.maximumValue = 100.0f;
    slicesSlider.value = arc4random() % 100;          //设置当前的值
}
```

Change:方法实现了当滑动滑块控件中的滑块时，将饼状图的形状进行重新设置以及在标签中显示滑块当前的值。程序代码如下：

```
-(IBAction)Change:(UISlider *)slider{
if (slider.tag == 77)
     valueLabel.text = [NSString stringWithFormat:@"%.0f",slider.value];
            //设置标签内容
if (slider.tag == 7)
     holeLabel.text = [NSString stringWithFormat:@"%.0f",slider.value];
            //设置标签内容
    [pieView reloadData];
}
```

【代码解析】

本实例关键功能是饼状图的绘制。下面就是这个知识点的详细讲解。

饼状图的绘制就是在圆形图中创建一些曲线段，需要使用 CGContext 的 CGContextAddArc 函数实现。其语法形式如下：

```
void CGContextAddArc (
   CGContextRef c,
   CGFloat x,
   CGFloat y,
   CGFloat radius,
   CGFloat startAngle,
   CGFloat endAngle,
   int clockwise
);
```

其中，参数说明如下：

- CGContextRef c 表示上下文；
- CGFloat x 表示圆点 x 坐标值；
- CGFloat y 表示圆点 y 坐标值；
- CGFloat radius 表示半径；
- CGFloat startAngle 表示开始的弧度；
- CGFloat endAngle 表示结束的弧度；
- int clockwise 表示绘制的方向（0 为顺时针，1 为逆时针）。

在此代码中就使用了 CGContextAddArc 函数绘制了一个饼状图，代码如下：

```
CGContextAddArc(
   context,
   centerX,
   centerY,
   radius,
   startAngle,
   endAngle,
   false
);
```

其中，参数说明如下：

- context 表示上下文；
- centerX 表示圆点 x 坐标值；
- centerY 表示圆点 y 坐标值；
- radius 表示半径；
- startAngle 表示开始的弧度；
- endAngle 表示结束的弧度；
- false 表示绘制的方向，它是顺时针绘制的。

实例 41　柱　状　图

【实例描述】

柱状图是一种以长方形的长度为变量的统计图表，它主要用来比较两个或者两个以上的数值。本实例就为读者实现一个柱状图的效果。当用户单击界面的 Change 按钮后，柱

状图将重新绘制。运行效果如图 3.3 所示。

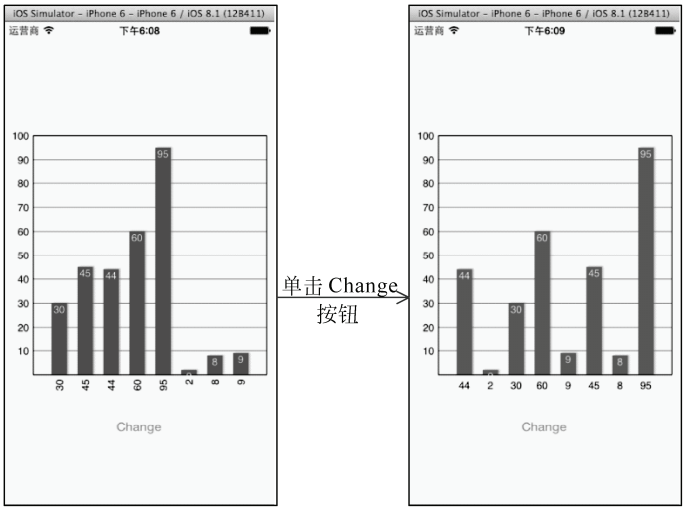

图 3.3　运行效果

【实现过程】

（1）创建项目，命名为“柱状图”。

（2）创建一个基于 NSMutableArray 类的分类 MutableArray。

（3）打开 NSMutableArray+MutableArray.h 文件，编写代码，实现方法的声明。程序代码如下：

```
#import <Foundation/Foundation.h>
@interface NSMutableArray (MutableArray)
- (void)shuffle;
@end
```

（4）打开 NSMutableArray+MutableArray.m 文件，编写代码，对 shuffle 方法进行定义，实现随机元素的交换。程序代码如下：

```
#import "NSMutableArray+MutableArray.h"
@implementation NSMutableArray (MutableArray)
- (void)shuffle
{
    NSUInteger count = [self count];
    //循环，交换对象
     for (NSUInteger i = 0; i < count; ++i)
     {
        NSInteger nElements  = count - i;
        NSInteger n= arc4random_uniform(nElements) + i;
        [self exchangeObjectAtIndex:i withObjectAtIndex:n];      //交换对象
    }
}
@end
```

（5）创建一个基于 UIView 类的 BarView 类。

（6）打开 BarView.h 文件，编写代码，实现宏定义、数据结构的定义、协议、实例变量、对象、属性、方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#define DEGREES_TO_RADIANS(x) (M_PI * x / 180.0)                    //宏定义
//数据结构的定义
typedef enum
{
    SimpleBarChartXLabelTypeVerticle,
    SimpleBarChartXLabelTypeHorizontal,
    SimpleBarChartXLabelTypeAngled
} SimpleBarChartXLabelType;
typedef enum
{
    SimpleBarChartBarTextTypeRoof,
    SimpleBarChartBarTextTypeTop,
    SimpleBarChartBarTextTypeMiddle
} SimpleBarChartBarTextType;
@class BarView;
//BarViewDataSource 协议
@protocol BarViewDataSource <NSObject>
@required
- (NSUInteger)numberOfBarsInBarChart:(BarView *)barChart;
    //获取柱形的个数
- (CGFloat)barChart:(BarView *)barChart valueForBarAtIndex: (NSUInteger)
index;
@optional
- (UIColor *)barChart:(BarView*)barChart colorForBarAtIndex: (NSUInteger)
index;                              //获取柱形显示的颜色
- (NSString *)barChart:(BarView *)barChart textForBarAtIndex:(NSUInteger)
index;
- (NSString *)barChart:(BarView *)barChart xLabelForBarAtIndex:(NSUInteger)
index;
@end
//BarViewDelegate 协议
@protocol BarViewDelegate <NSObject>
@optional
- (void)animationDidEndForBarChart:(BarView *)barChart;
@end
@interface BarView : UIView{
    __weak id <BarViewDataSource> _dataSource;
    __weak id <BarViewDelegate> _delegate;
    CGFloat _xLabelMaxHeight;
    NSInteger _topValue;
    ……
    UIView *_xLabelView;
    UIView *_barTextView;
}
//属性
@property (weak, nonatomic) id <BarViewDataSource> dataSource;
@property (weak, nonatomic) id <BarViewDelegate> delegate;
……
```

```
@property (nonatomic, assign) NSInteger incrementValue;
- (void)reloadData;
@end
```

（7）打开 BarView.m 文件，编写代码，实现柱形图的绘制。使用的方法如表 3-4 所示。

表 3-4　BarView.m文件中方法总结

方　　法	功　　能
initWithFrame:	初始化
reloadData	加载数据
setupBorders	创建边框
drawBorders	绘制边框
setupBars	创建柱形
animateBarAtIndex:	柱形的动画
drawBar:	绘制柱形
setupGrid	创建网格
drawGrid	绘制网格
setupYAxisLabels	创建 y 轴的标签
setupXAxisLabels	创建 x 轴的标签
setupBarTexts	创建柱形中的文本
displayAxisLabels	显示标签

可以将这些方法划分为两个步骤：创建柱状图以及显示（动画效果）。这里需要讲解几个重要的方法（其他方法请读者参考源代码）。创建柱状图需要使用 initWithFrame:、reloadData 、 setupBorders 、 setupBorders 、 setupBars 、 setupGrid 、 setupYAxisLabels 、setupXAxisLabels、setupBarTexts 方法。其中，initWithFrame:方法实现对柱状图的初始化。程序代码如下：

```
- (id)initWithFrame:(CGRect)frame
{
    if (!(self = [super initWithFrame:frame]))
        return self;
    self.animationDuration= 1.0;                    //设置动画持续时间
    self.hasGrids= YES;
    self.incrementValue= 10;
    self.barWidth= 20.0;                            //设置柱形的宽度
    self.barAlpha= 1.0;
   //颜色设置
    self.chartBorderColor= [UIColor blackColor];
    self.gridColor= [UIColor grayColor];
   //标签设置
    self.hasYLabels= YES;
    self.yLabelFont= [UIFont fontWithName:@"Helvetica" size:12.0];
    self.yLabelColor= [UIColor blackColor];         //设置 y 轴上标签的颜色
    self.xLabelFont= [UIFont fontWithName:@"Helvetica" size:12.0];
    self.xLabelColor= [UIColor blackColor];         //设置 x 轴上标签的颜色
    self.xLabelType= SimpleBarChartXLabelTypeVerticle;
    self.barTextFont= [UIFont fontWithName:@"Helvetica" size:12.0];
    self.barTextColor= [UIColor whiteColor];        //设置柱形的文本颜色
    self.barTextType= SimpleBarChartBarTextTypeTop;
    //创建可变数组对象
    _barPathLayers= [[NSMutableArray alloc] init];
    _barHeights= [[NSMutableArray alloc] init];
```

```
    _barLabels= [[NSMutableArray alloc] init];
    _barTexts= [[NSMutableArray alloc] init];
    //创建并设置图层对象
    _gridLayer= [CALayer layer];
    [self.layer addSublayer:_gridLayer];          //添加图层对象
    _barLayer= [CALayer layer];
    [self.layer addSublayer:_barLayer];           //添加图层对象
    _borderLayer= [CALayer layer];
    [self.layer addSublayer:_borderLayer];        //添加图层对象
    //创建并设置空白视图对象
    _yLabelView= [[UIView alloc] init];
    _yLabelView.alpha= 0.0;                       //设置透明度
    [self addSubview:_yLabelView];
    _xLabelView= [[UIView alloc] init];
    _xLabelView.alpha= 0.0;                       //设置透明度
    [self addSubview:_xLabelView];
    _barTextView= [[UIView alloc] init];
    _barTextView.alpha= 0.0;                      //设置透明度
    [self addSubview:_barTextView];
    return self;
}
```

reloadData 方法实现对数据的加载，即实现对柱形图的创建。程序代码如下：

```
- (void)reloadData
{
    if (_dataSource)
    {
        _numberOfBars = [_dataSource numberOfBarsInBarChart:self];
        //移除对象
        [_barHeights removeAllObjects];
        [_barLabels removeAllObjects];
        [_barTexts removeAllObjects];
        //遍历
        for (NSInteger i = 0; i < _numberOfBars; i++)
        {
            [_barHeights addObject:[NSNumber numberWithFloat:[_dataSource
            barChart:
            Self valueForBarAtIndex:i]]];                  //添加对象
            if (_dataSource && [_dataSource respondsToSelector:
            @selector(barChart:xLabelForBarAtIndex:)])
                [_barLabels addObject:[_dataSource barChart:self xLabelFor
                BarAtIndex:i]];                    //添加对象
            if (_dataSource && [_dataSource respondsToSelector:
            @selector(barChart:textForBarAtIndex:)])
                [_barTexts addObject:[_dataSource barChart:self textFor
                BarAtIndex:i]];                    //添加对象
        }
        _maxHeight= [_barHeights valueForKeyPath:@"@max.self"];
        _minHeight= [_barHeights valueForKeyPath:@"@min.self"];
        _topValue= (self.incrementValue - (_maxHeight.integerValue % self.
        incrementValue)) + _maxHeight.integerValue;
       //判断 x 轴标签的类型
        switch (self.xLabelType)
        {
            case SimpleBarChartXLabelTypeVerticle:
            default:
                _xLabelRotation = 90.0;                //设置旋转度数
                break;
```

```
        case SimpleBarChartXLabelTypeHorizontal:
            _xLabelRotation = 0.0;                    //设置旋转度数
            break;
        case SimpleBarChartXLabelTypeAngled:
            _xLabelRotation = 45.0;                   //设置旋转度数
            break;
    }
    //遍历
    for (NSString *label in _barLabels)
    {
    NSDictionary *dic=[[NSDictionary alloc]initWithObjectsAndKeys:
    self.xLabelFont ,NSFontAttributeName, nil];        //创建字典对象
      CGSize labelSize=[label sizeWithAttributes:dic];
                                                       //获取标签的尺寸大小
       CGFloat labelHeightWithAngle = sin(DEGREES_TO_RADIANS (_xLabel
       Rotation)) *labelSize.width;
       //判断 labelSize 的 height 属性值是否大于 labelHeightWithAngle
       if (labelSize.height > labelHeightWithAngle)
       {
            _xLabelMaxHeight = (_xLabelMaxHeight > labelSize.height) ?
            _xLabelMaxHeight : labelSize.height;   //获取标签的最大高度
       }
       else{
            _xLabelMaxHeight = (_xLabelMaxHeight > labelHeightWith
            Angle)?_ xLabelM axHeight : labelHeightWithAngle;
                                                       //获取标签的最大高度
       }
    }
    NSDictionary*dic=[[NSDictionaryalloc]initWithObjectsAndKeys:self.
    yLabelFont ,NSFontAttributeName, nil];
    CGSize yLabelSize=self.hasYLabels ?[[NSString stringWithFormat:
    @"%i",_topValue]sizeWithAttributes:
    dic]: CGSizeZero;
     //设置 x、y 轴视图的框架
    _yLabelView.frame= CGRectMake(0.0,0.0,self.hasYLabels ? yLabel
    Size.width + 5.0 : 0.0,self.bounds.size.height);
    _xLabelView.frame= CGRectMake(_yLabelView.frame.origin.x + _y
    LabelView.frame.size.width,self.bounds.size.height - _x
    LabelMaxHeight,self.bounds.size.width - (_yLabelView. frame.
    origin.x + _yLabelView.frame.size.width),_xLabelMaxHeight);
     //设置图层的框架
    _gridLayer.frame= CGRectMake(_yLabelView.frame.origin.x +_yLabel
    View. frame. size.width,0.0,self.bounds.size.width - (_yLabelView.
    frame. origin.x + _yLabelView. frame.size. width), self. bounds.
    size.height - (_xLabelMaxHeight > 0.0 ? (_xLabelMaxHeight + 5.0) :
    0.0));
    _barLayer.frame= _gridLayer.frame;
    _borderLayer.frame= _gridLayer.frame;
    _barTextView.frame= _gridLayer.frame;
    [self setupBorders];                               //创建边框
    [self drawBorders];                                //绘制边框
    @autoreleasepool {
        [self setupBars];
        [self animateBarAtIndex:0];
    }
    if (self.hasGrids)
    {
        [self setupGrid];                              //创建网格
```

```
            [self drawGrid];                                    //绘制网格
        }
        [self setupYAxisLabels];                                //创建 y 轴的标签
        [self setupXAxisLabels];
        [self setupBarTexts];                                   //创建柱形中的文本
    }
}
```

setupBorders 方法实现对柱形图边框的设置，程序代码如下：

```
- (void)setupBorders
{
    if (_borderPathLayer != nil)
    {
        [_borderPathLayer removeFromSuperlayer];        //移除指定的图层对象
        _borderPathLayer = nil;
    }
    //设置上下左右 4 个点
    CGPoint bottomLeft= CGPointMake(CGRectGetMinX(_borderLayer.bounds),
    CGRectGetMinY(_borderLayer.bounds));
    CGPoint bottomRight= CGPointMake(CGRectGetMaxX(_borderLayer.bounds),
    CGRectGetMinY(_borderLayer.bounds));
    CGPoint   topLeft=   CGPointMake(CGRectGetMinX(_borderLayer.bounds),
    CGRectGetMaxY(_borderLayer.bounds));
    CGPoint   topRight=   CGPointMake(CGRectGetMaxX(_borderLayer.bounds),
    CGRectGetMaxY(_borderLayer.bounds));
    //绘制路径
    UIBezierPath *path  = [UIBezierPath bezierPath];
    [path moveToPoint:bottomRight];                             //设置开始点
    [path addLineToPoint:topRight];                             //设置结束点
    [path addLineToPoint:topLeft];
    [path addLineToPoint:bottomLeft];
    [path addLineToPoint:bottomRight];
    //根据绘制的路径创建形状
    _borderPathLayer= [CAShapeLayer layer];
    _borderPathLayer.frame= _borderLayer.bounds;                //设置框架
    _borderPathLayer.bounds= _borderLayer.bounds;               //设置边界
    _borderPathLayer.geometryFlipped= YES;
    _borderPathLayer.path= path.CGPath;                         //设置路径
    _borderPathLayer.strokeColor= self.chartBorderColor.CGColor;
    _borderPathLayer.fillColor= nil;
    _borderPathLayer.lineWidth= 1.0f;                           //设置线的宽度
    _borderPathLayer.lineJoin= kCALineJoinBevel;
    [_borderLayer addSublayer:_borderPathLayer];
}
```

setupBars 方法实现对柱状图中柱形的创建，程序代码如下：

```
- (void)setupBars
{
    for (CAShapeLayer *layer in _barPathLayers)
    {
        if (layer != nil)
        {
            [layer removeFromSuperlayer];                   //移除指定的图层对象
        }
    }
    [_barPathLayers removeAllObjects];                      //移除所有的对象
    CGFloat barHeightRatio= _barLayer.bounds.size.height / (CGFloat) _
```

```
    topValue;
    CGFloat xPos= _barLayer.bounds.size.width / (_numberOfBars + 1);
    //循环，添加图层对象和对象
    for (NSInteger i = 0; i < _numberOfBars; i++)
    {
        CGPoint bottom= CGPointMake(xPos, _barLayer.bounds.origin.y);
        CGPoint top = CGPointMake(xPos, ((NSNumber *)[_barHeights object
        AtIndex:i]).floatValue * barHeightRatio);
        xPos+= _barLayer.bounds.size.width / (_numberOfBars + 1);
        UIBezierPath *path= [UIBezierPath bezierPath];     //创建贝塞尔路径
        [path moveToPoint:bottom];                         //设置开始点
        [path addLineToPoint:top];                         //设置结束点
        UIColor *barColor= [UIColor darkGrayColor];
        if (_dataSource && [_dataSource respondsToSelector: @selector
        (barChart:colorForBarAtIndex:)])
            barColor = [_dataSource barChart:self colorForBarAtIndex:i];
        //根据路径创建形状，即柱形
        CAShapeLayer *barPathLayer= [CAShapeLayer layer];
        barPathLayer.frame= _barLayer.bounds;              //设置框架
        barPathLayer.bounds= _barLayer.bounds;
        barPathLayer.geometryFlipped   = YES;
        barPathLayer.path= path.CGPath;                    //设置路径
        barPathLayer.strokeColor= [barColor colorWithAlphaComponent:
        self.self.barAlpha].CGColor;
        barPathLayer.fillColor= nil;
        barPathLayer.lineWidth= self.barWidth;             //设置线宽
        barPathLayer.lineJoin= kCALineJoinBevel;
        barPathLayer.hidden= YES;
        barPathLayer.shadowOffset= self.barShadowOffset;   //设置阴影偏移量
        barPathLayer.shadowColor= self.barShadowColor.CGColor;
        barPathLayer.shadowOpacity= self.barShadowAlpha;   //设置阴影透明度
        barPathLayer.shadowRadius= self.barShadowRadius;
        [_barLayer addSublayer:barPathLayer];
        [_barPathLayers addObject:barPathLayer];
    }
}
```

setupGrid 方法实现对网格的创建，即柱状图中横线的创建。程序代码如下：

```
- (void)setupGrid
{
    //判断_gridPathLayer 是否不为空
    if (_gridPathLayer != nil)
    {
        [_gridPathLayer removeFromSuperlayer];             //移除指定的图层
        _gridPathLayer = nil;
    }
    CGFloat gridUnit= _gridLayer.bounds.size.height / _topValue;
    CGFloat gridSeperation= gridUnit * (CGFloat)self.incrementValue;
    CGFloat yPos= gridSeperation;
    UIBezierPath *path= [UIBezierPath bezierPath];         //创建贝塞尔路径
    //循环设置点
    while (yPos <= _gridLayer.bounds.size.height)
    {
        CGPoint left= CGPointMake(0.0, yPos);
        CGPoint right= CGPointMake(_gridLayer.bounds.size.width, yPos);
        yPos+= gridSeperation;
```

```
        [path moveToPoint:left];                                    //设置开始点
        [path addLineToPoint:right];                                //设置结束点
    }
    //创建横线
    _gridPathLayer= [CAShapeLayer layer];
    _gridPathLayer.frame= _gridLayer.bounds;                        //设置框架
    _gridPathLayer.bounds= _gridLayer.bounds;
    _gridPathLayer.geometryFlipped= YES;
    _gridPathLayer.path= path.CGPath;                               //设置路径
    _gridPathLayer.strokeColor= self.gridColor.CGColor;
    _gridPathLayer.fillColor= nil;
    _gridPathLayer.lineWidth= 1.0f;                                 //设置线宽
    _gridPathLayer.lineJoin= kCALineJoinBevel;
    [_gridLayer addSublayer:_gridPathLayer];
}
```

setupYAxisLabels 方法实现对 y 轴标签的创建。程序代码如下：

```
- (void)setupYAxisLabels
{
    if (!self.hasYLabels)
        return;
   //判断_yLabelView 的 alpha 属性值是否大于 0.0
    if (_yLabelView.alpha > 0.0)
    {
        _yLabelView.alpha = 0.0;                                    //设置透明度
        [[_yLabelView subviews] makeObjectsPerformSelector: @selector
        (removeFromSuperview)];
    }
    CGFloat gridUnit= _gridLayer.bounds.size.height / _topValue;
    CGFloat gridSeperation= gridUnit * (CGFloat)self.incrementValue;
    NSDictionary *dic=[[NSDictionary alloc]initWithObjectsAndKeys: self.y
    Label Font ,NSFontAttributeName, nil];
    CGSize   yLabelSize=[[NSString   stringWithFormat:@"%i",_top   Value]
    sizeWithAttributes:dic];
    CGFloat yPos= 0.0;
    NSInteger maxVal= _topValue;
    //判断 yPos 是否大于_gridLayer.bounds.size.height 的属性值
    while (yPos < _gridLayer.bounds.size.height)
    {
        CGRect yLabelFrame= CGRectMake(0.0,0.0,yLabelSize. width,yLabel
        Size.height);
        //创建标签对象
        UILabel *yLabel = [[UILabel alloc] initWithFrame:yLabelFrame];
        yLabel.font = self.yLabelFont;                              //设置字体
        yLabel.backgroundColor= [UIColor clearColor];
        yLabel.textColor= self.yLabelColor;
        yLabel.textAlignment= NSTextAlignmentRight;                 //设置对齐方式
        yLabel.center= CGPointMake(yLabel.center.x, yPos);
        yLabel.text= [NSString stringWithFormat:@"%i", maxVal];     //设置文本内容
        [_yLabelView addSubview:yLabel];
        maxVal-= self.incrementValue;
        yPos+= gridSeperation;
    }
```

```
}
```

setupXAxisLabels 方法实现对 x 轴标签的创建。程序代码如下：

```
- (void)setupXAxisLabels
{
    //判断_barLabels 的 count 属性值是否为 0
    if (_barLabels.count == 0)
        return;
    //判断_xLabelViews 的 alpha 属性值是否为 0.0
    if (_xLabelView.alpha > 0.0)
    {
        _xLabelView.alpha = 0.0;                              //设置透明度
        [[_xLabelViewsubviews]makeObjectsPerformSelector:@selector (remo
        veFromSuperview)];
    }
    CGFloat xPos= _barLayer.bounds.size.width / (_numberOfBars + 1);
    //循环，添加视图对象
    for (NSInteger i = 0; i < _numberOfBars; i++)
    {
        NSString *xLabelText= [_barLabels objectAtIndex:i];
        NSDictionary *dic=[[NSDictionary alloc]initWithObjectsAndKeys:
        self.xLabelFont ,NSFontAttributeName, nil];        //创建字典对象
       CGSize xLabelSize=[xLabelText sizeWithAttributes:dic];
        CGRect xLabelFrame= CGRectMake(0.0,0.0,xLabelSize. width,xLabel
        Size.height);
        //创建并设置标签对象
        UILabel *xLabel= [[UILabel alloc] initWithFrame:xLabelFrame];
        xLabel.font= self.xLabelFont;                    //设置字体
        xLabel.backgroundColor= [UIColor clearColor];
        xLabel.textColor= self.xLabelColor;
        xLabel.textAlignment= NSTextAlignmentRight;     //设置对齐方式
        xLabel.text= xLabelText;                         //设置文本内容
        xLabel.transform= CGAffineTransformMakeRotation
        (DEGREES_TO_RADIANS(-_xLabelRotation));
        //判断标签的类型，从而进行中心点的设置
        switch (self.xLabelType)
        {
            case SimpleBarChartXLabelTypeVerticle:
            default:
               xLabel.center = CGPointMake(xPos, (xLabelSize.width / 2.0));
                //设置中心位置
                break;
            case SimpleBarChartXLabelTypeHorizontal:
                xLabel.center = CGPointMake(xPos, _xLabelMaxHeight / 2.0);
                //设置中心位置
                break;
            case SimpleBarChartXLabelTypeAngled:
            {
                CGFloat labelHeightWithAngle= sin(DEGREES_TO_RADIANS
                (_xLabel Rotation)) * xLabelSize.width;
                xLabel.center= CGPointMake(xPos - (labelHeightWithAngle /
```

```
                2.0), labelHeightWithAngle / 2.0);          //设置中心位置
                break;
            }
        }
        [_xLabelView addSubview:xLabel];                    //添加视图对象
        xPos+= _barLayer.bounds.size.width / (_numberOfBars + 1);
    }
}
```

setupBarTexts 方法实现对柱形中显示文本的创建。程序代码如下：

```
-(void)setupBarTexts
{
    //判断_barLabels 的 count 属性值是否为 0
    if (_barTexts.count == 0)
        return;
    //判断_xLabelViews 的 alpha 属性值是否为 0.0
    if (_barTextView.alpha > 0.0)
    {
        _barTextView.alpha = 0.0;                           //设置透明度
        [[_barTextView subviews] makeObjectsPerformSelector: @selector
        (removeFromSuperview)];
    }
    CGFloat xPos= _barLayer.bounds.size.width /(_numberOfBars + 1);
                                        //遍历，实现视图对象的添加
    for (NSInteger i = 0; i < _numberOfBars; i++)
    {
        NSString *barLabelText = [_barTexts objectAtIndex:i];
        NSDictionary *dic=[[NSDictionary alloc]initWithObjectsAndKeys:
        self.barTextFont ,NSFontAttributeName, nil];   //创建字典对象
        CGSize barTextSize=[barLabelText sizeWithAttributes:dic];
        CGRect barTextFrame= CGRectMake(0.0,0.0,barTextSize. width,bar
        TextSize.height);
        UILabel *barText= [[UILabel alloc] initWithFrame:barTextFrame];
        //实例化标签对象
        barText.font= self.barTextFont;
        barText.backgroundColor= [UIColor clearColor];     //设置背景颜色
        barText.textColor= self.barTextColor;
        barText.textAlignment  = NSTextAlignmentCenter;
        barText.text= barLabelText;                        //设置文本内容
        CGFloat barHeight= (_barLayer.bounds.size.height / (CGFloat)_top
        Value) * ((NSNumber *)[_barHeights objectAtIndex:i]).floatValue;
        //判断标签的类型，从而进行中心点的设置
        switch (self.barTextType)
        {
            case SimpleBarChartBarTextTypeTop:
            default:
                barText.center=CGPointMake(xPos,_barLayer.bounds.size.
                height-(barHeight-(barTextSize.height/2.0)));    //设置中心位置
                break;
            case SimpleBarChartBarTextTypeRoof:
                barText.center = CGPointMake(xPos, _barLayer.bounds.size.
```

```
                height-(barHeight+(barTextSize.height/2.0)));    //设置中心位置
                break;
            case SimpleBarChartBarTextTypeMiddle:
            {
                CGFloat minBarHeight= (_barLayer.bounds.size.height /
                (CGFloat)_topValue) * _minHeight.floatValue;
                barText.center= CGPointMake(xPos, _barLayer.bounds.size.
                height - (minBarHeight / 2.0));            //设置中心位置
                break;
            }
        }
        [_barTextView addSubview:barText];
        xPos += _barLayer.bounds.size.width / (_numberOfBars + 1);
    }
}
```

在本实例中要显示柱状图，都是以动画的形式显示的。需要使用 drawBorders、animateBarAtIndex:、drawBar:、drawGrid、displayAxisLabels 方法实现。其中，drawBorders 方法实现以绘制的动画形式显示柱状图上的边框。程序代码如下：

```
- (void)drawBorders
{
    //判断动画持续时间是否为 0.0
    if (self.animationDuration == 0.0)
        return;
    [_borderPathLayer removeAllAnimations];                //移除所有的动画
    //动画
    CABasicAnimation *pathAnimation= [CABasicAnimation animationWith
    KeyPath:@"strokeEnd"];
    pathAnimation.duration= self.animationDuration;    //创建动画持续时间
    pathAnimation.fromValue= [NSNumber numberWithFloat:0.0f]; //设置开始值
    pathAnimation.toValue= [NSNumber numberWithFloat:1.0f];   //设置结束值
    [_borderPathLayer addAnimation:pathAnimation forKey:@"strokeEnd"];
}
```

animateBarAtIndex:方法实现以动画的形式显示柱形，程序代码如下：

```
- (void)animateBarAtIndex:(NSInteger)index
{
    //判断 index 是否大于等于_barPathLayers.count
    if (index >= _barPathLayers.count)
    {
        [self displayAxisLabels];                                  //显示标签
        return;
    }
    __block NSInteger i= index + 1;
    __weak BarView *weakSelf = self;
    //动画
    [CATransaction begin];
    //设置动画持续时间
    [CATransaction setAnimationDuration:(self.animationDuration / (CG Fl
```

```
    oat)_barPathLayers.count)];
    [CATransaction     setAnimationTimingFunction:[CAMediaTimingFunction
    functionWithName:kCAMediaTimingFunctionEaseOut]];
    //设置完成动画时执行的块
    [CATransaction setCompletionBlock:^{
        [weakSelf animateBarAtIndex:i];
    }];
    CAShapeLayer *barPathLayer= [_barPathLayers objectAtIndex:index];
    barPathLayer.hidden= NO;                    //不隐藏 barPathLayer 对象
    [self drawBar:barPathLayer];
    [CATransaction commit];
}
```

displayAxisLabels 方法实现对 x、y 轴以及柱形上标签的显示。

```
- (void)displayAxisLabels
{
    if (self.hasYLabels || _barTexts.count > 0 || _barLabels.count > 0)
    {
        //判断动画持续时间是否为 0.0
        if (self.animationDuration > 0.0)
        {
            __weak BarView *weakSelf = self;
            //实现动画
            [UIView animateWithDuration:self.animationDuration / 2.0
            delay:0.0 options:UIViewAnimationOptionCurveEaseOut animations:^{
            //设置透明度
                _yLabelView.alpha= 1.0;
                _xLabelView.alpha= 1.0;
                _barTextView.alpha= 1.0;
            } completion:^(BOOL finished) {
                if (weakSelf.delegate && [weakSelf.delegate respondsTo
                Selector:@selector(animationDidEndForBarChart:)])
                //调用 animationDidEndForBarChart:方法
                    [weakSelf.delegate animationDidEndForBarChart:weak Self];
            }];
        }
        else
        {
            //设置透明度
            _yLabelView.alpha= 1.0;
            _xLabelView.alpha= 1.0;
            _barTextView.alpha= 1.0;
            if (_delegate && [_delegate respondsToSelector:@selector
            (animationDidEndForBarChart:)])
                [_delegate animationDidEndForBarChart:self];
                //调用 animationDidEndForBarChart:方法
        }
    }
    else
    {
```

```
        if (_delegate && [_delegate respondsToSelector: @selector
        (animationDidEndForBarChart:)])
            [_delegate animationDidEndForBarChart:self];
            //调用 animationDidEndForBarChart:方法
    }
}
```

（8）打开 ViewController.h 文件，编写代码，实现头文件、遵守协议、对象、实例变量的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import "BarView.h"
#import "NSMutableArray+MutableArray.h"
@interface ViewController : UIViewController<BarViewDataSource, Bar View
Delegate>{
   NSArray *_values;
    BarView *_chart;
    NSArray *_barColors;
    NSInteger _currentBarColor;
}
@end
```

（9）打开 ViewController.m 文件，编写代码，实例柱形图在界面的显示，以及单击按钮后柱状图的重绘功能。使用的方法如表 3-5 所示。

表 3-5　ViewController.m文件中方法总结

方　　法	功　　能
loadView	加载视图
changeClicked	单击按钮
viewDidAppear:	视图即将加载时调用，实现加载柱形图
numberOfBarsInBarChart:	获取柱形的个数
barChart:valueForBarAtIndex:	获取柱形的值，用来控制器柱形的高度
barChart:textForBarAtIndex:	获取显示在柱形上的文本内容
barChart:xLabelForBarAtIndex:	获取显示在 x 轴上标签的内容
barChart:colorForBarAtIndex:	获取柱形显示的颜色

其中，loadView 方法实现视图的加载，即实现图表的显示。程序代码如下：

```
- (void)loadView
{
    [super loadView];
    _values= @[@30, @45, @44, @60, @95, @2, @8, @9];          //创建数组
    _barColors=  @[[UIColor  blueColor],  [UIColor  redColor],  [UIColor
    blackColor], [UIColor orangeColor], [UIColor purpleColor], [UIColor
    greenColor]];                      //创建数组
    _currentBarColor= 0;
    CGRect chartFrame= CGRectMake(0.0,0.0,300.0,300.0);
    //绘制有柱形图视图对象的创建并设置
    _chart= [[BarView alloc] initWithFrame:chartFrame];
    _chart.center=  CGPointMake(self.view.frame.size.width  /  2.0,  self.
    view.frame.size.height / 2.0);
```

```
    _chart.delegate= self;                                  //设置委托
    _chart.dataSource= self;
    _chart.barShadowOffset= CGSizeMake(2.0, 1.0);
    _chart.animationDuration= 1.0;                          //设置动画持续时间
    _chart.barShadowColor= [UIColor grayColor];
    _chart.barShadowAlpha= 0.5;
    _chart.barShadowRadius= 1.0;
    _chart.barWidth= 18.0;                                  //设置柱形的宽度
    _chart.xLabelType= SimpleBarChartXLabelTypeVerticle;
    _chart.incrementValue= 10;
    _chart.barTextType= SimpleBarChartBarTextTypeTop;
    _chart.barTextColor= [UIColor whiteColor];              //设置柱形的文本颜色
    _chart.gridColor =[UIColor grayColor];
    [self.view addSubview:_chart];
    //创建并设置按钮对象
    UIButton *changeButton= [UIButton buttonWithType:UIButton TypeRounded Rect];
    changeButton.frame=  CGRectMake(0.0,_chart.frame.origin.y  +  _chart.
    frame. size. height + 20.0,100.0,44.0);
    changeButton.center=  CGPointMake(self.view.frame.size.width  /  2.0,
    changeButton.center.y);
    [changeButton setTitle:@"Change" forState:UIControlStateNormal]; //设置标题
    [changeButton addTarget:self action:@selector(changeClicked) for
    ControlEvents:UIControlEventTouchDown];                 //添加动作
    [self.view addSubview:changeButton];
}
```

changeClicked 方法实现单击按钮后，重回柱形图的功能。程序代码如下：

```
- (void)changeClicked
{
    NSMutableArray *valuesCopy = _values.mutableCopy;
    [valuesCopy shuffle];
    _values = valuesCopy;
   //判断_chart.xLabelTyp 是否为 SimpleBarChartXLabelTypeVerticle
    if (_chart.xLabelType == SimpleBarChartXLabelTypeVerticle)
        _chart.xLabelType = SimpleBarChartXLabelTypeHorizontal;
    else
        _chart.xLabelType = SimpleBarChartXLabelTypeVerticle;
    //设置柱状图的类型
    _currentBarColor = ++_currentBarColor % _barColors.count;
    [_chart reloadData];
}
```

【代码解析】

由于本实例中的代码和方法非常多，为了方便读者的阅读，笔者绘制了一个执行流程图，如图 3.4 所示。其中，单击运行按钮到柱状图以动画的形式显示在界面上，需要使用 initWithFrame:、reloadData、setupBorders、drawBorders、setupBars、animateBarAtIndex:、drawBar:、setupGrid、drawGrid、setupYAxisLabels、setupXAxisLabels、setupBarTexts、displayAxisLabels、loadView、viewDidAppear:、numberOfBarsInBarChart:、barChart:valueForBarAtIndex:、barChart:textForBarAtIndex:、barChart:xLabelForBarAtIndex:、barChart:colorForBarAtIndex:方法共同实现。

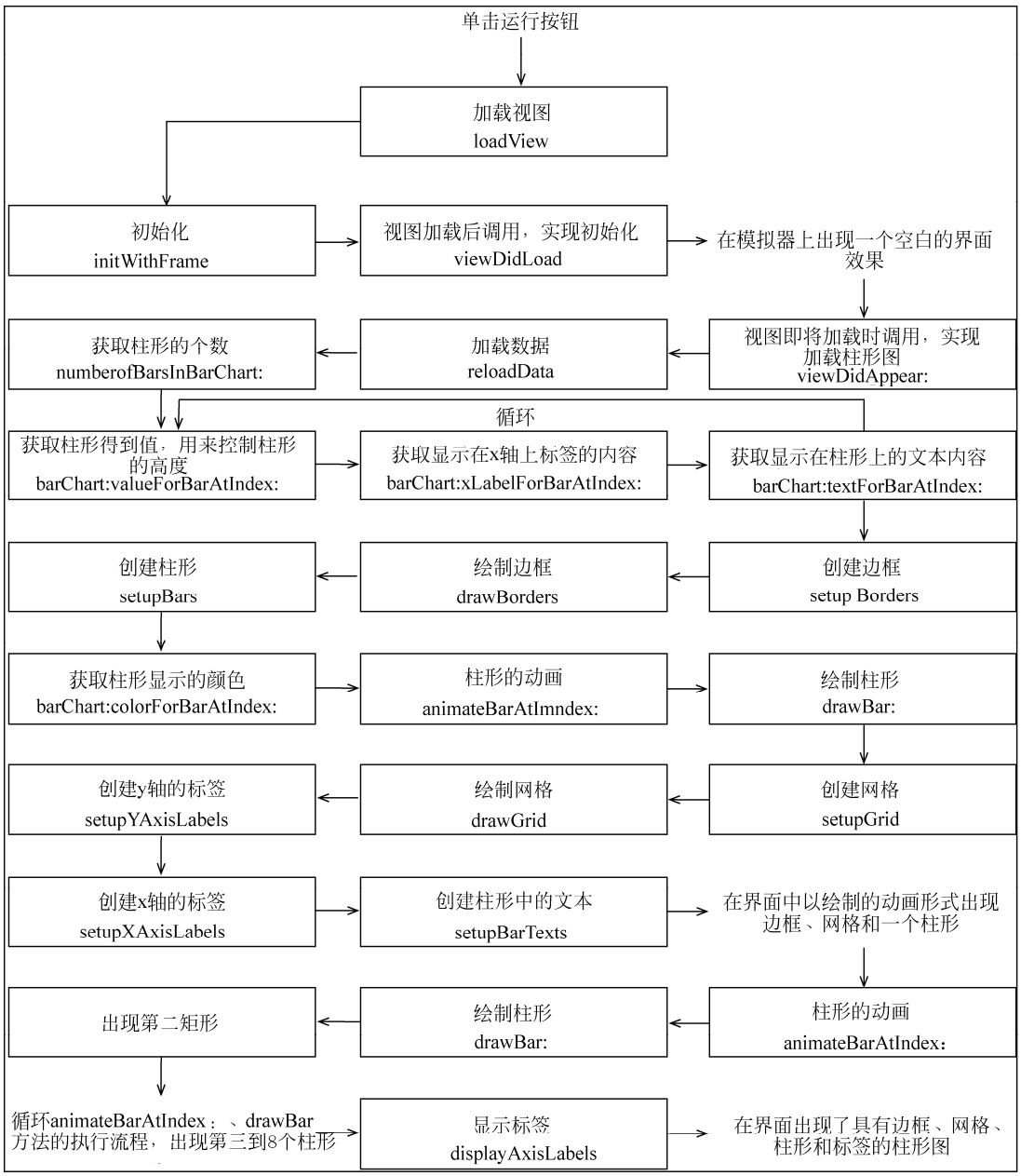

图 3.4　程序执行流程

这时会在界面显示一个柱状图和一个 Change 按钮。当用户单击此按钮后，会重新对柱状图进行绘制，并且颜色、柱形的高度以及内容都会发生改变，实现这些需要使用 shuffle、reloadData、setupBorders、drawBorders、setupBars、animateBarAtIndex:、drawBar:、setupGrid、drawGrid、setupYAxisLabels、setupXAxisLabels、setupBarTexts、displayAxisLabels、changeClicked 、 numberOfBarsInBarChart: 、 barChart:valueForBarAtIndex: 、 barChart:textForBarAtIndex:、barChart:xLabelForBarAtIndex:、barChart:colorForBarAtIndex: 方法共同实现。它们的程序执行流程如图 3.5 所示。

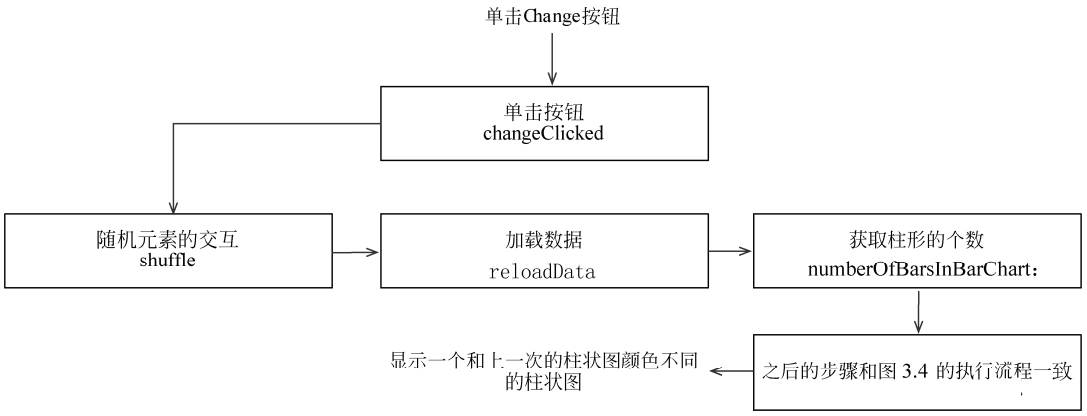

图 3.5　程序执行流程

实例 42　折　线　图

【实例描述】

折线图用来表示数据变化的统计图，它常用于股票趋势走向等地方。本实例就为读者实现一个具有动态效果的折线图。运行效果如图 3.6 所示。

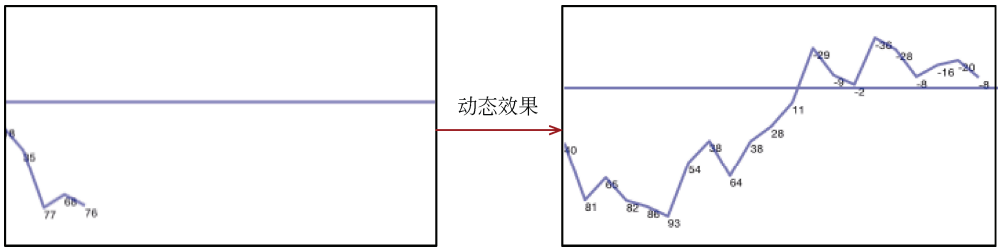

图 3.6　运行效果

【实现过程】

（1）创建一个项目，命名为“折线图”。

（2）创建一个基于 UIView 类的 View 类。

（3）打开 View.h 文件，编写代码，实现宏定义、属性的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
//宏定义
#define GraphColor [[UIColor blueColor] colorWithAlphaComponent:0.5]
#define str(index) [NSString stringWithFormat : @"%.f", [[self.values
objectAtIndex:(index)] floatValue] * kYScale]
#define point(x, y) CGPointMake((x) * kXScale, yOffset + (y) * kYScale)
@interface View : UIView
//属性
@property (nonatomic, readonly, strong) NSMutableArray *values;
@property (nonatomic, strong)   dispatch_source_t timer;
```

```
@end
```

（4）打开 View.m 文件，编写代码，实现具有动态效果的折线图。使用的方法如表 3-6 所示。

表 3-6　View.m文件中方法总结

方　　法	功　　能
CGAffineTransformMakeScaleTranslate	获取偏移量
awakeFromNib	在初始化后调用
updateValues	值的更新
drawRect:	绘制
drawAtPoint:withStr:	绘制文本

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，awakeFromNib 方法在初始化后调用，实现对定时器的创建以及设置。程序代码如下：

```
- (void)awakeFromNib
{
    [self setContentMode:UIViewContentModeRight];
    _values = [NSMutableArray array];
    __weak id weakSelf = self;
    double  delayInSeconds = 0.25;
    //创建定时器
    self.timer =dispatch_source_create(DISPATCH_SOURCE_TYPE_TIMER, 0, 0,
    dispatch_get_main_queue());
   dispatch_source_set_timer(_timer,dispatch_walltime(NULL,0),(unsigned)
    (delayInSeconds * NSEC_PER_SEC), 0);
                    //使用 dispatch_source_set_timer 函数设置定时器的参数
    //设置回调
    dispatch_source_set_event_handler(_timer, ^{[weakSelf updateValues];
    });
    dispatch_resume(_timer);
}
```

updateValues 方法实现在每隔 0.25 秒后对数值的更新。程序代码如下：

```
- (void)updateValues
{
    double nextValue = sin(CFAbsoluteTimeGetCurrent()) + ((double)rand() /
    (double)RAND_MAX);
    [self.values addObject: [NSNumber numberWithDouble:nextValue]];    //添加对象
    CGSize size = self.bounds.size;                                   //获取边界的尺寸
    CGFloat maxDimension = size.width;
    NSUInteger maxValues =(NSUInteger)floorl(maxDimension / kXScale);
    if ([self.values count] > maxValues) {
        [self.values removeObjectsInRange:
        NSMakeRange(0, [self.values count] - maxValues)];      //移除对象
    }
    [self setNeedsDisplay];
}
```

drawRect:方法实现折线图的绘制。程序代码如下：

```
- (void)drawRect:(CGRect)rect
{
    if ([self.values count] == 0) {
        return;
```

```
    }
    CGContextRef ctx = UIGraphicsGetCurrentContext();
    CGContextSetStrokeColorWithColor(ctx,[GraphColor CGColor]);
    //设置线的颜色
    CGContextSetLineJoin(ctx, kCGLineJoinRound);
    CGContextSetLineWidth(ctx, 2);                          //设置线宽
    CGMutablePathRef path = CGPathCreateMutable();
    CGFloat yOffset = self.bounds.size.height / 2;
    CGAffineTransform   transform   =CGAffineTransformMakeScaleTranslate
    (kXScale, kYScale,0, yOffset);
    //中间轴的绘制
    CGPathMoveToPoint(path, &transform, 0, 0);              //设置开始点
    CGPathAddLineToPoint(path,  &transform,  self.bounds.size.width,  0);
    //设置结束点
    CGFloat y = [[self.values objectAtIndex:0] floatValue];
    CGPathMoveToPoint(path, &transform, 0, y);
    [self drawAtPoint:point(0, y) withStr:str(0)];          //绘制
    for (NSUInteger x = 1; x < [self.values count]; ++x) {
        y = [[self.values objectAtIndex:x] floatValue];
        CGPathAddLineToPoint(path, &transform, x, y);       //设置结束点
        [self drawAtPoint:point(x, y) withStr:str(x)];
    }
    CGContextAddPath(ctx, path);                            //添加路径
    CGPathRelease(path);
    CGContextStrokePath(ctx);
}
```

drawAtPoint:str 方法实现文本的绘制。程序代码如下：

```
- (void)drawAtPoint:(CGPoint)point withStr:(NSString *)str
{
    [str  drawAtPoint:point  withAttributes:@{NSFontAttributeName:[UIFont
    sys temFontOfSize:8], NSStrokeColorAttributeName:GraphColor}];
                                                            //绘制
}
```

【代码解析】

本实例关键功能是动态折线图的绘制以及指定点文本的绘制。下面依次讲解这两个知识点。

1．动态折线图的绘制

在本实例中折线的绘制是使用多个线段组成的。在每隔 0.25 秒后就会执行一次以下的程序。从而实现动态折线的绘制。

```
CGPathMoveToPoint(path, &transform, 0, y);
[self drawAtPoint:point(0, y) withStr:str(0)];
for (NSUInteger x = 1; x < [self.values count]; ++x) {
    y = [[self.values objectAtIndex:x] floatValue];
    CGPathAddLineToPoint(path, &transform, x, y);
    [self drawAtPoint:point(x, y) withStr:str(x)];
}
```

2．指定点文本的绘制

通过指定的某一点绘制文本，需要使用 drawAtPoint:withAttributes:方法。其语法形式如下：

```
- (void)drawAtPoint:(CGPoint)point withAttributes:(NSDictionary *)attrs;
```

其中，(CGPoint)point 表示指定的点；(NSDictionary ∗)attrs 表示文本属性。在本实例中就是使用了 drawAtPoint:withAttributes:方法实现了在固定的点对文本进行显示。代码如下：

```
[str drawAtPoint:pointwithAttributes:@{NSFontAttributeName:[UIFont system
FontOfSize:8],
NSStrokeColorAttributeName:GraphColor}];
```

其中，point 表示指定的点。@{NSFontAttributeName:[UIFont systemFontOfSize:8], NSStrokeColorAttributeName:GraphColor}表示文本的属性。

实例 43　波　形　图

【实例描述】

本实例主要实现一个具有动画的波形图。当用户点击“暂停”按钮，波形图的动画就会停止。当用户点击“开始”按钮，停止的波形图就会运动。运行效果如图 3.7 所示。

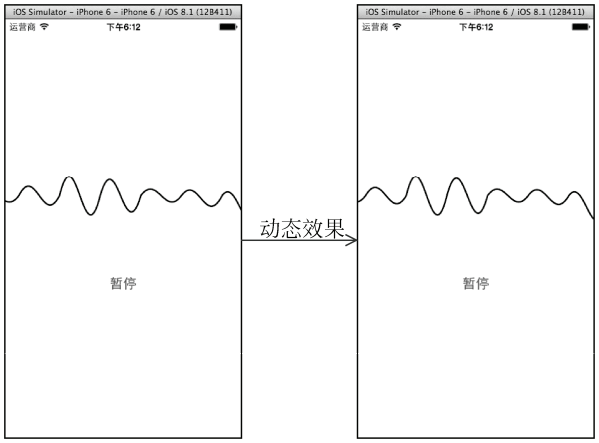

图 3.7　运行效果

【实现过程】

（1）创建一个项目，命名为“波形图”。

（2）创建一个基于 UIView 类的 Wave 类。

（3）打开 Wave.h 文件，编写代码，实现实例变量以及方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
@interface WaveView : UIView{
    float swing;
    int   i;
    float j;
}
- (void)callDraw:(float)f;
@end
```

（4）打开 Wave.m 文件，编写代码，实现动态效果的波形图。使用的方法如表 3-7 所示。

表 3-7　Wave.m文件中方法总结

方　　法	功　　能
initWithFrame:	初始化
callDraw:	判断波的幅度，并调用绘制方法
drawRect:	绘制波

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。其中，callDraw:方法实现对波的幅度的判断。程序代码如下：

```
- (void)callDraw:(float)f
{
   //判断 f 是否小于 1
    if (f < 1) {
       swing = f;
    }else {
       swing = 1;
    }
    [self setNeedsDisplay];                                    //重新绘制
}
```

drawRect:方法实现对波的绘制。程序代码如下：

```
- (void)drawRect:(CGRect)rect
{
    //判断 i 是否大于 55
    if (i > 55) {
        i = 0;
        j = 0;
    }
    CGContextRef context = UIGraphicsGetCurrentContext();
    //设置
    CGContextSetStrokeColorWithColor(context, [UIColor black Color]. CGColor);
    CGContextSetLineWidth(context, 2.0);
    //绘制波
    CGContextMoveToPoint(context, -55+i, 40);
    //添加弧线
    CGContextAddCurveToPoint(context, -32.5+i, 30-(swing*20), -22.5+i, 50+
    (swing*20), 0+i, 40);
    CGContextMoveToPoint(context, 0+i, 40);
    //添加弧线
    CGContextAddCurveToPoint(context, 22.5+i, 30-(swing*(20+j)), 32.5+i,
    50+(swing*(20+j)), 55+i, 40);
    CGContextMoveToPoint(context, 55+i, 40);
    //添加弧线
    CGContextAddCurveToPoint(context, 77.5+i, 30-(swing*80), 87.5+i, 50+
    (swing*80), 110+i, 40);
    CGContextMoveToPoint(context, 110+i, 40);                  //设置开始点
    CGContextAddCurveToPoint(context, 132.5+i, 30-(swing*(80-j)), 142.5+i,
    50+(swing*(80-j)), 165+i, 40);
    CGContextMoveToPoint(context, 165+i, 40);
    CGContextAddCurveToPoint(context, 187.5+i, 30-(swing*20), 197.5+i, 50+
    (swing*20), 220+i, 40);
    CGContextMoveToPoint(context, 220+i, 40);                  //设置开始点
    CGContextAddCurveToPoint(context, 242.5+i, 30-(swing*20), 252.5+i, 50+
```

```
    (swing* (20+j)), 275+i, 40);
    CGContextMoveToPoint(context, 275+i, 40);
    CGContextAddCurveToPoint(context, 297.5+i, 30-(swing*20), 307.5+i, 50+
    (swing* 80), 330+i, 40);
    CGContextStrokePath(context);                          //绘制
    i++;
    j = j+0.727;
}
```

（5）打开 ViewController.h 文件，编写代码，实现头文件、对象以及实例变量的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import "WaveView.h"
@interface ViewController : UIViewController{
   WaveView *view;
   NSTimer *timer;                      //声明时间定时器
   UIButton *open;
   BOOL  on_off;
}
@end
```

（6）打开 ViewController.m 文件，编写代码，实现动态波的效果。使用的方法如表 3-8 所示。

表 3-8　ViewController.m文件中方法总结

方　法	功　能
viewDidLoad	视图加载后调用，实现初始化
backButton	单击按钮，控制动态波
detectionVoice	为波的幅度赋值

其中，viewDidLoad 方法实现对绘制有波形图的视图的对象、按钮等进行创建和设置。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    view = [[WaveView alloc] initWithFrame:CGRectMake(0, 200, 320, 80)];
    view.backgroundColor = [UIColor clearColor];          //设置背景颜色
    view.clipsToBounds = YES;
    [self.view addSubview:view];
    //设置定时检测
    timer  =  [NSTimer  scheduledTimerWithTimeInterval:0.01  target:self
    selector:@selector(detectionVoice) userInfo:nil repeats:YES];
                                                          //创建定时器
    //创建并设置按钮对象
    open = [UIButton buttonWithType:UIButtonTypeCustom];
    open.frame = CGRectMake(140, 350, 40, 20);                //设置框架
    [open setTitleColor:[UIColor blueColor] forState:UIControlStateNormal];
    [open setTitle:@"暂停" forState:UIControlStateNormal];     //设置标题
    [open addTarget:self action:@selector(backButton) forControlEvents:
    UIControlEventTouchUpInside];
    [self.view addSubview:open];
}
```

backButton 方法实现对波形的动画效果的控制。程序代码如下：

```
- (void)backButton
{
    if (on_off) {
        [open setTitle:@"暂停" forState:UIControlStateNormal];
        //开启定时器
        [timer setFireDate:[NSDate distantPast]];
    }else {
        [open setTitle:@"开始" forState:UIControlStateNormal];
        //关闭定时器
        [timer setFireDate:[NSDate distantFuture]];
    }
    on_off = !on_off;
}
```

【代码解析】

本实例关键功能是波的绘制以及波的动态效果。下面着重讲解波的绘制。

波形其实就是一条曲线。曲线的绘制需要使用 CGContext 的 CGContextAddCurveToPoint 函数，其语法形式如下：

```
void CGContextAddCurveToPoint (
   CGContextRef c,
   CGFloat cp1x,
   CGFloat cp1y,
   CGFloat cp2x,
   CGFloat cp2y,
   CGFloat x,
   CGFloat y
);
```

其中，参数说明如下：

- CGContextRef c 表示上下文；
- CGFloat cp1x 表示第一个指定点的 x 坐标；
- CGFloat cp1y 表示第一个指定点的 y 坐标；
- CGFloat cp2x 表示第二个指定点的 x 坐标；
- CGFloat cp2y 表示第二个指定点的 y 坐标；
- CGFloat x 表示终点 x 的坐标；
- CGFloat y 表示终点 y 的坐标。

注意在使用 CGContextAddCurveToPoint 函数之前，必须要先使用 CGContext MoveToPoint 函数指定起始点。在此代码中就是使用 CGContextAddCurveToPoint 函数实现了波形的绘制。代码如下：

```
CGContextAddCurveToPoint(
    context,
    -32.5+i,
    30-(swing*20),
    -22.5+i,
    50+(swing*20),
    0+i,
```

```
    40
);
```

其中，参数说明如下：

- context 表示上下文；
- -32.5+i 表示第一个指定点的 x 坐标；
- 30-(swing∗20)表示第一个指定点的 y 坐标；
- -22.5+i 表示第二个指定点的 x 坐标；
- 50+(swing∗20)表示第二个指定点的 y 坐标；
- 0+i 表示终点 x 的坐标；
- 40 表示终点 y 的坐标。

实例 44　油　量　表

【实例描述】

在汽车中，油量表是很常见的，它用来表示油量的多少。在本实例中就为读者实现一个类似于汽车中的油量表。当用户单击 Change 按钮后，油量表中的数字以及指针（线）就会发生改变。运行效果如图 3.8 所示。

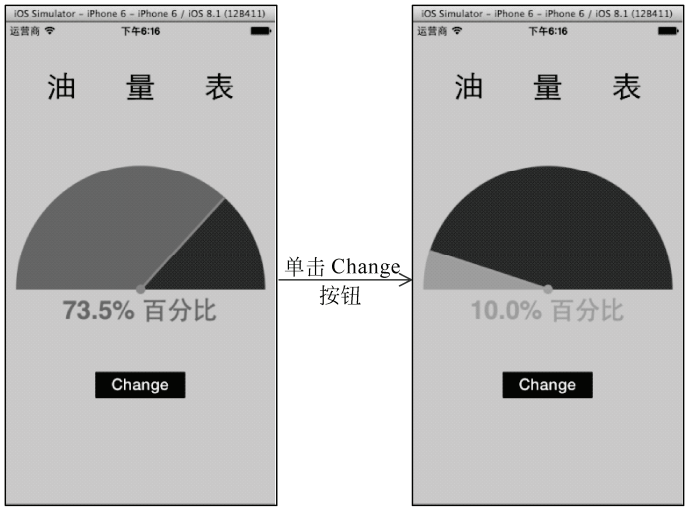

图 3.8　运行效果

【实现过程】

（1）创建一个项目，命名为“油量表”。

（2）添加 CoreText.framework 框架到创建的项目中。

（3）创建一个基于 CALayer 类的 SemicircularLayer 类。

（4）打开 SemicircularLayer.h 文件，编写代码，实现头文件、宏定义、属性以及方法的声明。程序代码如下：

```
#import <QuartzCore/QuartzCore.h>
```

```
#import <CoreText/CoreText.h>                    //头文件
//宏定义
#define DEG2RAD(angle) angle*M_PI/180.0
#define INITIAL_ANGLE 180
#define ENDING_ANGLE 0
#define CENTER_WIDTH 8
@interface SemicircularLayer : CALayer
//属性
@property(nonatomic) CGFloat percentage;
@property(nonatomic, strong) NSString *text;
……
@property(nonatomic) CGFloat fontSize;
- (id)initWithLayer:(id)aLayer;                  //初始化方法
@end
```

（5）打开 SemicircularLayer.m 文件，编写代码，实现对油量表的绘制以及动画效果的实现。使用的方法如表 3-9 所示。

表 3-9　SemicircularLayer.m文件中方法总结

方　　法	功　　能
makeAnimationForKey:	获取动画
actionForKey:	获取行为对象
initWithLayer:	初始化图层
needsDisplayForKey:	修改了指定的键后判断是否需要重新显示该层
drawInContext:	绘制油量表

这里需要讲解几个重要的方法（其他方法请读者参考源代码）。油量表的绘制需要使用 initWithLayer:、needsDisplayForKey:和 drawInContext:方法实现。其中，initWithLayer:表示对图层的初始化。程序代码如下：

```
- (id)initWithLayer:(id)aLayer
{
    if (self = [super initWithLayer:aLayer])
{
     //判断 aLayer 是否在 SemicircularLayer 中
        if ([aLayer isKindOfClass:[SemicircularLayer class]])
        {
            SemicircularLayer *layer = (SemicircularLayer *)aLayer;
            //判断是否实现了 setContentsScale:方法
            if ([layer respondsToSelector:@selector(setContentsScale:)])
            {
                layer.contentsScale = [[UIScreen mainScreen] scale];
            }
            self.percentage = layer.percentage;
            self.text = layer.text;                          //设置文本内容
            self.mainColor = layer.mainColor;
            self.secondaryColor = layer.secondaryColor;
            self.lineColor = layer.lineColor;                //设置线的颜色
            self.fontName = layer.fontName;
            self.fontSize = layer.fontSize;                  //设置字体的大小
        }
    }
    return self;
}
```

drawInContext:方法实现对油量表的绘制。程序代码如下

```
-(void)drawInContext:(CGContextRef)ctx
{
    NSDictionary *dic=[NSDictionary dictionaryWithObjectsAndKeys:[UIFont
    fontWithName:self.fontName size:self.fontSize],NSFontAttributeName ,
    nil];                                                    //创建字典对象
    CGSize  sizeControl=[@"sample"  boundingRectWithSize:self.frame.size
    options:NSStringDrawingUsesLineFragmentOrigin | NSString DrawingUses
    FontLeading attributes:dic context:nil].size;          //获取尺寸
    CGPoint center = CGPointMake( self.bounds.size.width/2, self.bounds.
    size.height - sizeControl.height - 10.0 );
    CGFloat radius = MIN( center.x, center.y ) - 1;
    CGFloat startingAngleRad = DEG2RAD( INITIAL_ANGLE );
    CGFloat endingAngleRad = DEG2RAD( ENDING_ANGLE );
    CGFloat  currentAngle  =  INITIAL_ANGLE  +  (  INITIAL_ANGLE  *  self.
    percentage/100.0 );
    CGFloat currentAngleRad = DEG2RAD( currentAngle );
    CGPoint  startingPoint  =  CGPointMake(  center.x  +  radius  *  cosf
    (startingAngleRad), center.y + radius * sinf(startingAngleRad) );
                                                             //开始点
    CGPoint  endPoint  =  CGPointMake(  center.x  +  radius  *  cosf
    (currentAngleRad) , center.y + radius * sinf(currentAngleRad) );
                                                             //结束点
    //绘制半圆
    CGContextBeginPath( ctx );
    CGContextMoveToPoint( ctx, center.x, center.y );       //设置开始点
    CGContextAddLineToPoint( ctx, startingPoint.x, startingPoint.y );
    CGContextAddArc( ctx, center.x, center.y, radius, startingAngleRad,
    currentAngleRad, NO );                                   //添加弧
    CGContextClosePath(ctx);                                 //关闭路径
    CGContextSetFillColorWithColor( ctx, self.mainColor );
    CGContextSetStrokeColorWithColor( ctx, self.mainColor );
    CGContextSetLineWidth( ctx, 1 );                         //设置线宽
    CGContextDrawPath( ctx, kCGPathFillStroke );
    //绘制背景
    if( self.percentage < 100.0 )
    {
        CGContextBeginPath( ctx );
        CGContextMoveToPoint( ctx, center.x, center.y );
        CGContextAddLineToPoint( ctx, endPoint.x, endPoint.y );  //设置结束点
        CGContextAddArc( ctx, center.x, center.y, radius, currentAngleRad,
        endingAngleRad, NO );
        CGContextClosePath(ctx);                             //关闭路径
        CGContextSetFillColorWithColor( ctx, self.secondaryColor. CGColor );
                                                             //设置填充颜色
        CGContextSetStrokeColorWithColor( ctx, self.secondaryColor. CGColor );
                                                             //设置线的颜色
        CGContextSetLineWidth( ctx, 1 );                     //设置线宽
        CGContextDrawPath( ctx, kCGPathFillStroke );
    }
    // 绘制中心点和进程线
    CGContextBeginPath( ctx );
    CGContextMoveToPoint( ctx, center.x, center.y );
        //设置开始点
    CGRect  rect  =  CGRectMake(  center.x  -  CENTER_WIDTH/2,  center.y  -
    CENTER_WIDTH/2, CENTER_WIDTH, CENTER_WIDTH );
```

```
    CGContextAddEllipseInRect( ctx, rect );                    //添加圆
    //进程线
    CGContextMoveToPoint( ctx, center.x, center.y );
    CGContextAddLineToPoint( ctx, endPoint.x, endPoint.y );
    CGContextClosePath(ctx);                                   //关闭路径
    CGContextSetFillColorWithColor( ctx, self.lineColor );
    CGContextSetStrokeColorWithColor( ctx, self.lineColor );//设置线的颜色
    CGContextSetLineCap( ctx, kCGLineCapRound );
    CGContextSetLineWidth( ctx, 3 );                           //设置线宽
    CGContextDrawPath( ctx, kCGPathFillStroke );
    //对文本的绘制
    CGContextSetTextMatrix( ctx, CGAffineTransformMakeScale( 1.0, -1.0 ));
    NSString *str = [NSString stringWithFormat:@"%.1f%@ %@", self.
    percentage, @"%", self.text];
    NSDictionary *arr=[NSDictionary dictionaryWithObjectsAndKeys:[UIFont
    fontWithName:self.fontName size:self.fontSize],NSFontAttributeName ,
    nil];                                                      //创建字典
    //绘制尺寸
    CGSize strSize=[str boundingRectWithSize:self.frame.size options:
    NSStringDrawingUsesLineFragmentOrigin|NSStringDrawingUsesFontLeading
    attributes:arr context:nil].size;                          //获取尺寸
    CTFontRef sysUIFont = CTFontCreateWithName( (__bridge CFStringRef)
    self.fontName, self.fontSize, NULL );
    NSDictionary *attributesDict = [NSDictionary dictionaryWithObjects
    AndKeys:(__bridge id)sysUIFont, (id)kCTFontAttributeName, self.main
    Color, (id)kCTForegroundColorAttributeName, nil];//创建字典
    NSAttributedString *attributedStr = [[NSAttributedString alloc]init
    WithString:str attributes:attributesDict];
    //创建一个文本行对象
    CTLineRef lineref = CTLineCreateWithAttributedString( (__bridge
    CFAttributedStringRef)attributedStr );
    CGContextSetTextPosition( ctx, center.x - ( strSize.width/2.0 ),
    center.y + strSize.height );                               //设置位置
    CTLineDraw( lineref, ctx );                                //绘制
    CFRelease( lineref );
    CFRelease( sysUIFont );
}
```

动画效果的实现需要使用 makeAnimationForKey:和 actionForKey:方法。其中，makeAnimationForKey:方法实现获取动画对象。程序代码如下：

```
-(CABasicAnimation *)makeAnimationForKey:(NSString *)key
{
    CABasicAnimation *anim = [CABasicAnimation animationWithKeyPath:key];
    anim.fromValue = [[self presentationLayer] valueForKey:key]; //设置开始值
    anim.timingFunction = [CAMediaTimingFunction functionWithName:
    kCAMediaTimingFunctionEaseOut];
    anim.duration = 0.5;
    return anim;
}
```

（6）创建一个基于 UIView 类的 Semicircular 类。

（7）打开 Semicircular.h 文件，编写代码，实现头文件、对象、属性以及方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
```

```
#import "SemicircularLayer.h"                          //头文件
@interface Semicircular : UIView{
    CALayer *_mainLayer;
}
//属性
@property(nonatomic) CGFloat percentage;
@property(nonatomic, strong) NSString *text;
……
@property(nonatomic) CGFloat fontSize;
-(void) refresh;                                       //刷新方法
@end
```

（8）打开 Semicircular.m 文件，编写代码，实现对一些属性的设置以及刷新功能。使用的方法如表 3-10 所示。

表 3-10　Semicircular.m文件中方法总结

方　　法	功　　能
setPercentage:	设置百分比
setText:	设置文本
setMainColor:	设置线滑动过的颜色
setSecondaryColor:	设置线没有滑动过的颜色
setLineColor:	设置线的颜色
setFontName:	设置字体的名称
setFontSize:	设置字体的尺寸
initialize	初始化
initWithFrame:	使用框架初始化
initWithCoder:	初始化、实例化对象
refresh	刷新

其中，setPercentage:方法执行在单击按钮进行刷新后对百分比的设置。程序代码如下：

```
-(void) setPercentage:(CGFloat)newValue
{
   if( newValue > 100.0 ) newValue = 100.0;
   else if( newValue < 0.0 ) newValue = 0.0;
   percentage = newValue;                                  //设置百分比
   [self refresh];
}
```

initialize 方法实现一些初始化的设置。程序代码如下：

```
-(void) initialize
{
   _mainLayer = [SemicircularLayer layer];
   [self.layer addSublayer:_mainLayer];
   self.text = [NSString string];
   self.fontName = @"Helvetica";                           //设置字体
   self.fontSize = 30.0;                                   //设置字体大小
}
```

refresh 方法实现单击按钮后的刷新效果。程序代码如下：

```
-(void) refresh
{
   _mainLayer.frame = self.bounds;
```

```
    SemicircularLayer *layer = (SemicircularLayer *) _mainLayer;
    layer.percentage = self.percentage;
    layer.text = self.text;                                  //设置文本内容
    layer.mainColor = self.mainColor.CGColor;
    layer.secondaryColor = self.secondaryColor;
    layer.lineColor = self.lineColor.CGColor;                //设置线的颜色
    layer.fontName = self.fontName;
    layer.fontSize = self.fontSize;                          //设置字体大小
}
```

（9）打开 ViewController.h 文件，编写代码，实现头文件、宏定义、插座变量、实例变量以及方法的声明。程序代码如下：

```
#import <UIKit/UIKit.h>
#import "Semicircular.h"                              //头文件
//宏定义
#define  MAIN_ORANGE  [UIColor  colorWithRed:0.83  green:0.38  blue:0.0
alpha:1.0]
#define LINE_ORANGE [UIColor orangeColor]
#define MAIN_RED [UIColor colorWithRed:0.70 green:0.0 blue:0.0 alpha:1.0]
#define LINE_RED [UIColor redColor]
#define MAIN_GREEN [UIColor colorWithRed:0.47 green:0.7 blue:0.0 alpha:1.0]
#define LINE_GREEN [UIColor colorWithRed:0.0 green:0.7 blue:0.0 alpha:1.0]
@interface ViewController : UIViewController{
    //变量
    IBOutlet Semicircular *chart;
    CGFloat _percentage;
}
- (IBAction)Change:(id)sender;
@end
```

（10）打开 Main.storyboard 文件，对 View Controller 视图控制器的设计界面进行设计，效果如图 3.9 所示。

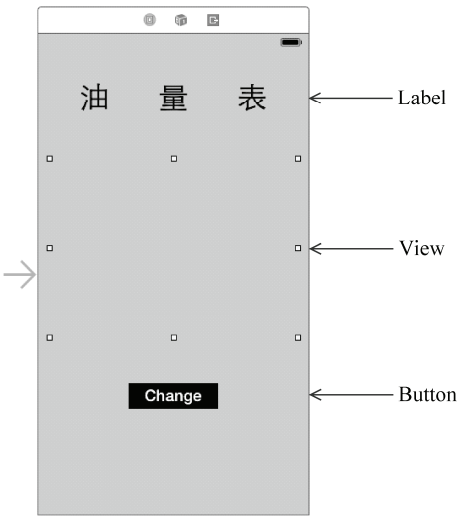

图 3.9　View Controller 视图控制器的设计界面

需要添加的视图、控件以及对它们的设置如表 3-11 所示。

表 3-11　视图、控件设置

视图、控件	属 性 设 置	其　　他
设计界面	Background：浅灰色	
Label	Text：油　　量　　表 Font：System 36.0 Alignment：居中	与插座变量 valueLabel 关联
View	Background：透明色	Class：Semicircular 与插座变量 chart 关联
Button	Title：Change Font：System Bold 19.0 Text Color：白色 Background：黑色	与动作 Change:关联

（11）打开 ViewController.m 文件，编写代码，实现对界面初始化的设置以及实现单击按钮后的响应。程序代码如下：

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    _percentage = 10.0;
    [chart setMainColor:MAIN_ORANGE];
    [chart setLineColor:LINE_ORANGE];                          //设置线的颜色
    [chart setSecondaryColor:[UIColor darkGrayColor]];
    [chart setFontName:@"Helvetica-Bold"];                     //设置字体
    [chart setFontSize:30.0];
    [chart setText:@"百分比"];                                  //设置文本内容
    [chart setPercentage:73.5];
}
//单击按钮
- (IBAction)Change:(id)sender {
[chart setPercentage:_percentage];
//判断 chart 的 percentage 是否大于 85
    if( [chart percentage] > 85 )
    {
        [chart setMainColor:MAIN_RED];
        [chart setLineColor:LINE_RED];
         } else if( [chart percentage] > 65 ) {  //判断chart的percentage是否大于65
        [chart setMainColor:MAIN_ORANGE];
        [chart setLineColor:LINE_ORANGE];                  //设置线的颜色
     } else {
        [chart setMainColor:MAIN_GREEN];
        [chart setLineColor:LINE_GREEN];                   //设置线的颜色
    }
    _percentage +=10;
    if( _percentage > 100.0 )
        _percentage -= 101.0;
}
```

【代码解析】

本实例关键功能是油量表颜色的改变以及油量表的动画效果。下面依次讲解这两个知识点。

1．油量表颜色的改变

在本实例中油量表颜色的改变是通过对百分比值的判断实现的。代码如下：

```
if( [chart percentage] > 85 )
{
    [chart setMainColor:MAIN_RED];
    [chart setLineColor:LINE_RED];
} else if( [chart percentage] > 65 ) {
    [chart setMainColor:MAIN_ORANGE];
    [chart setLineColor:LINE_ORANGE];
} else {
    [chart setMainColor:MAIN_GREEN];
[chart setLineColor:LINE_GREEN];
}
```

2．油量表的动画效果

在本实例中看到的油量表的动画效果是通过使用 CABasicAnimation 动画效果实现的。代码如下：

```
CABasicAnimation *anim = [CABasicAnimation animationWithKeyPath:key];
anim.fromValue = [[self presentationLayer] valueForKey:key]; //设置开始值
anim.timingFunction = [CAMediaTimingFunction functionWithName: kCAMedia
TimingFunctionEaseOut];
anim.duration = 0.5;
```