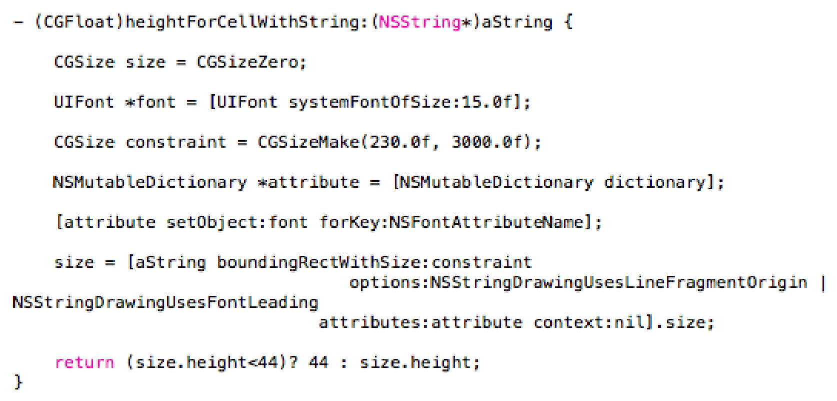
UITableViewCell动态调整高度的三种解决方案

解决方案一：对动态的字符串文本进行适配

调用NSString的 - boundingRectWithSize:options:attributes:context:计算字符串文本的尺寸，然后在UITableView的委托方法 - tableView:heightForRowAtIndexPath:设置UITableViewCell的高度。

代码片段：



```
- (CGFloat)heightForCellWithString:(NSString*)aString {
    CGSize size = CGSizeZero;
    UIFont *font = [UIFont systemFontOfSize:15.0f];
    CGSize constraint = CGSizeMake(230.0f, 3000.0f);
    NSMutableDictionary *attribute = [NSMutableDictionary dictionary];
    [attribute setObject:font forKey:NSFontAttributeName];
    size = [aString boundingRectWithSize:constraint
                                  options:NSStringDrawingUsesLineFragmentOrigin |
NSStringDrawingUsesFontLeading
                               attributes:attribute context:nil].size;
    return (size.height<44)? 44 : size.height;
}
```

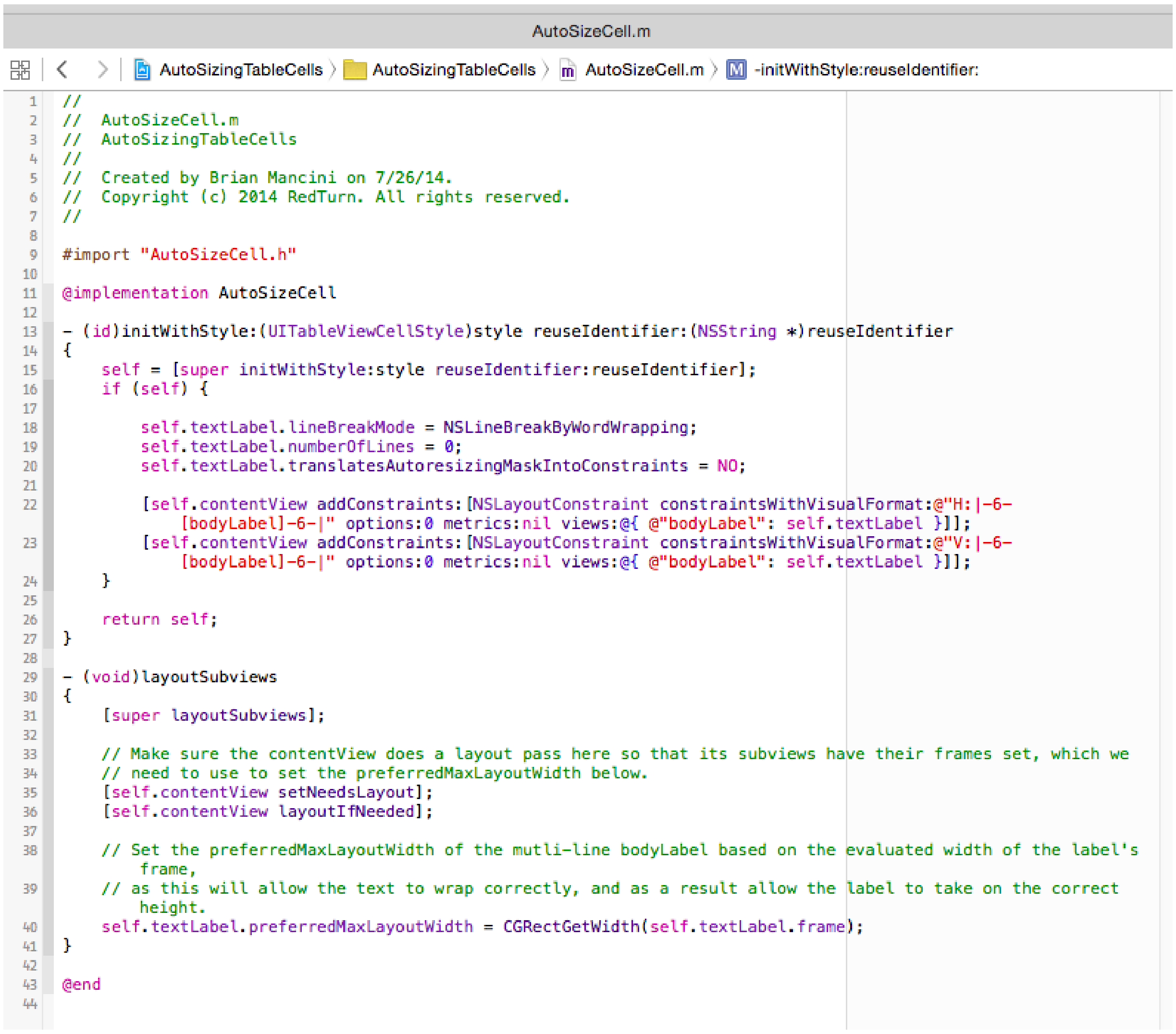
Screen Shot 2015-06-01 at 15.05.44.png

使用该方案的局限性较大，主要在于：
1、计算高度会出现误差，字符串会被截取；
2、只能用于字符串文本适配，对其它控件不具有扩展性。

解决方案二：iOS的Auto Layout，使用Visual Format Language

该方案使用了iOS的新技术Auto Layout实现UITableViewCell的自动适配，用代码实现时需要掌握基础的Visual Format Language语法。

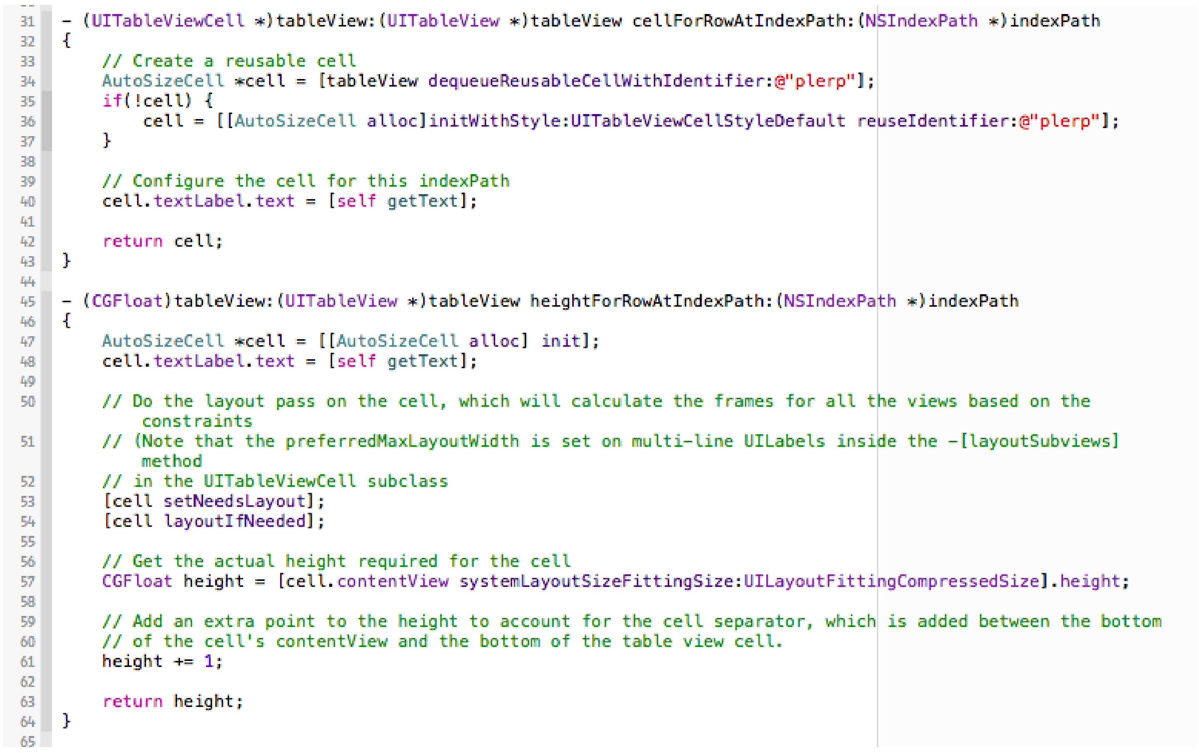
代码片段：UITableViewCell子类


AutoSizeCell.m
AutoSizingTableCells 〉 AutoSizingTableCells 〉 AutoSizeCell.m 〉 -initWithStyle:reuseIdentifier:

```objc
//
//  AutoSizeCell.m
//  AutoSizingTableCells
//
//  Created by Brian Mancini on 7/26/14.
//  Copyright (c) 2014 RedTurn. All rights reserved.
//

#import "AutoSizeCell.h"

@implementation AutoSizeCell

- (id)initWithStyle:(UITableViewCellStyle)style reuseIdentifier:(NSString *)reuseIdentifier
{
    self = [super initWithStyle:style reuseIdentifier:reuseIdentifier];
    if (self) {

        self.textLabel.lineBreakMode = NSLineBreakByWordWrapping;
        self.textLabel.numberOfLines = 0;
        self.textLabel.translatesAutoresizingMaskIntoConstraints = NO;

        [self.contentView addConstraints:[NSLayoutConstraint constraintsWithVisualFormat:@"H:|-6-
            [bodyLabel]-6-|" options:0 metrics:nil views:@{ @"bodyLabel": self.textLabel }]];
        [self.contentView addConstraints:[NSLayoutConstraint constraintsWithVisualFormat:@"V:|-6-
            [bodyLabel]-6-|" options:0 metrics:nil views:@{ @"bodyLabel": self.textLabel }]];
    }

    return self;
}

- (void)layoutSubviews
{
    [super layoutSubviews];

    // Make sure the contentView does a layout pass here so that its subviews have their frames set, which we
    // need to use to set the preferredMaxLayoutWidth below.
    [self.contentView setNeedsLayout];
    [self.contentView layoutIfNeeded];

    // Set the preferredMaxLayoutWidth of the mutli-line bodyLabel based on the evaluated width of the label's
        frame,
    // as this will allow the text to wrap correctly, and as a result allow the label to take on the correct
        height.
    self.textLabel.preferredMaxLayoutWidth = CGRectGetWidth(self.textLabel.frame);
}

@end
```

Screen Shot 2015-06-01 at 15.11.51.png

代码片段：UITableViewDelegate



```objc
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // Create a reusable cell
    AutoSizeCell *cell = [tableView dequeueReusableCellWithIdentifier:@"plerp"];
    if(!cell) {
        cell = [[AutoSizeCell alloc]initWithStyle:UITableViewCellStyleDefault reuseIdentifier:@"plerp"];
    }

    // Configure the cell for this indexPath
    cell.textLabel.text = [self getText];

    return cell;
}

- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    AutoSizeCell *cell = [[AutoSizeCell alloc] init];
    cell.textLabel.text = [self getText];

    // Do the layout pass on the cell, which will calculate the frames for all the views based on the
        constraints
    // (Note that the preferredMaxLayoutWidth is set on multi-line UILabels inside the -[layoutSubviews]
        method
    // in the UITableViewCell subclass
    [cell setNeedsLayout];
    [cell layoutIfNeeded];

    // Get the actual height required for the cell
    CGFloat height = [cell.contentView systemLayoutSizeFittingSize:UILayoutFittingCompressedSize].height;

    // Add an extra point to the height to account for the cell separator, which is added between the bottom
    // of the cell's contentView and the bottom of the table view cell.
    height += 1;

    return height;
}
```

Screen Shot 2015-06-01 at 15.13.16.png

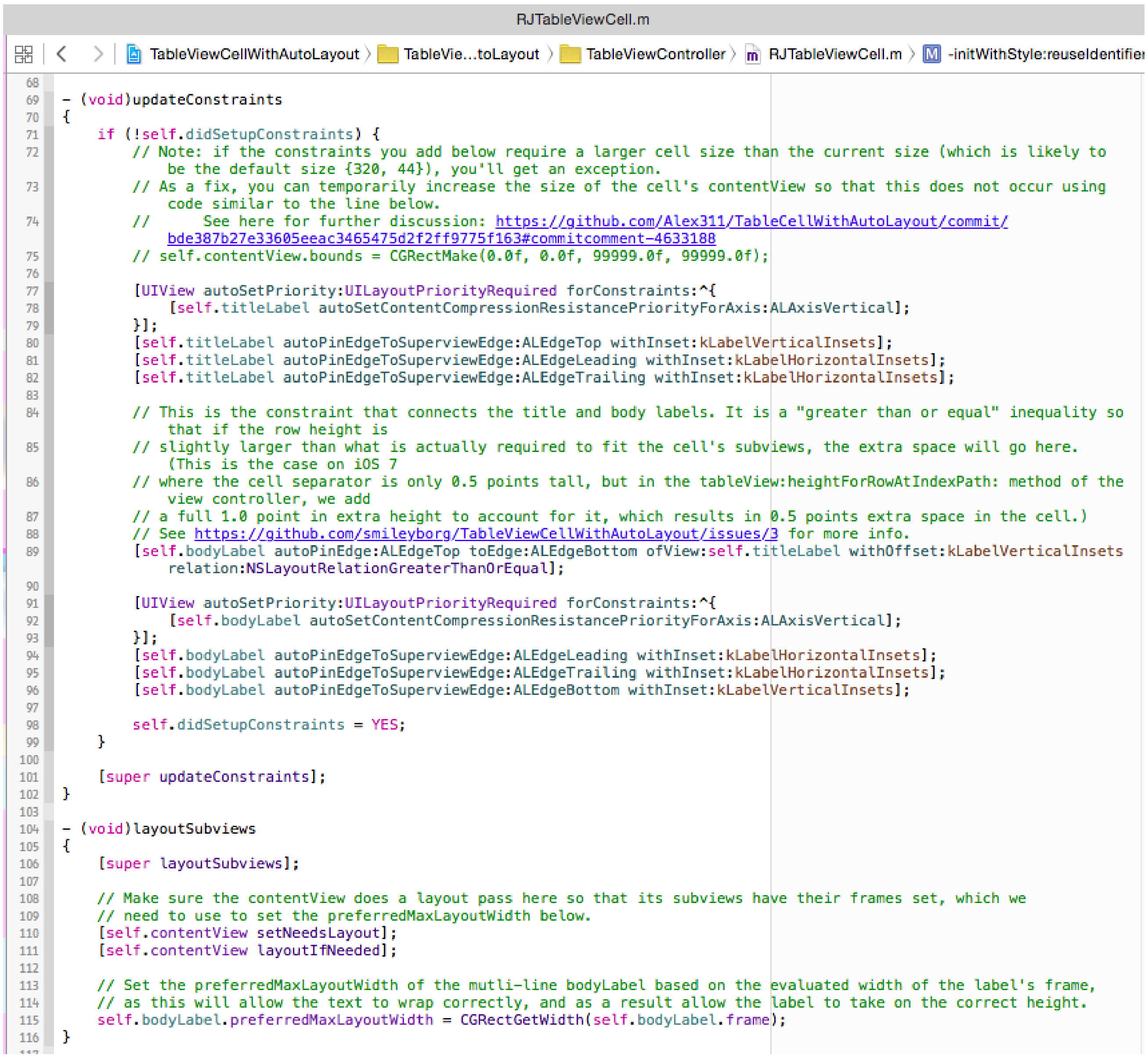
源代码： iOSExamples-AutoSizingTableCells

使用该方案有如下特点：
优点：适用性强，除了能适配字符串文本，还可以适配其它控件；
缺点：iOS的Visual Format Language语法不容易记，不太好用。

解决方案三：iOS的Auto Layout，使用开源类库 PureLayout

该方案的解决思路与方案二完全一致，只是使用了第三方开源类库 PureLayout去实现Auto Layout，实现起来更容易。

代码片段：UITableViewCell子类



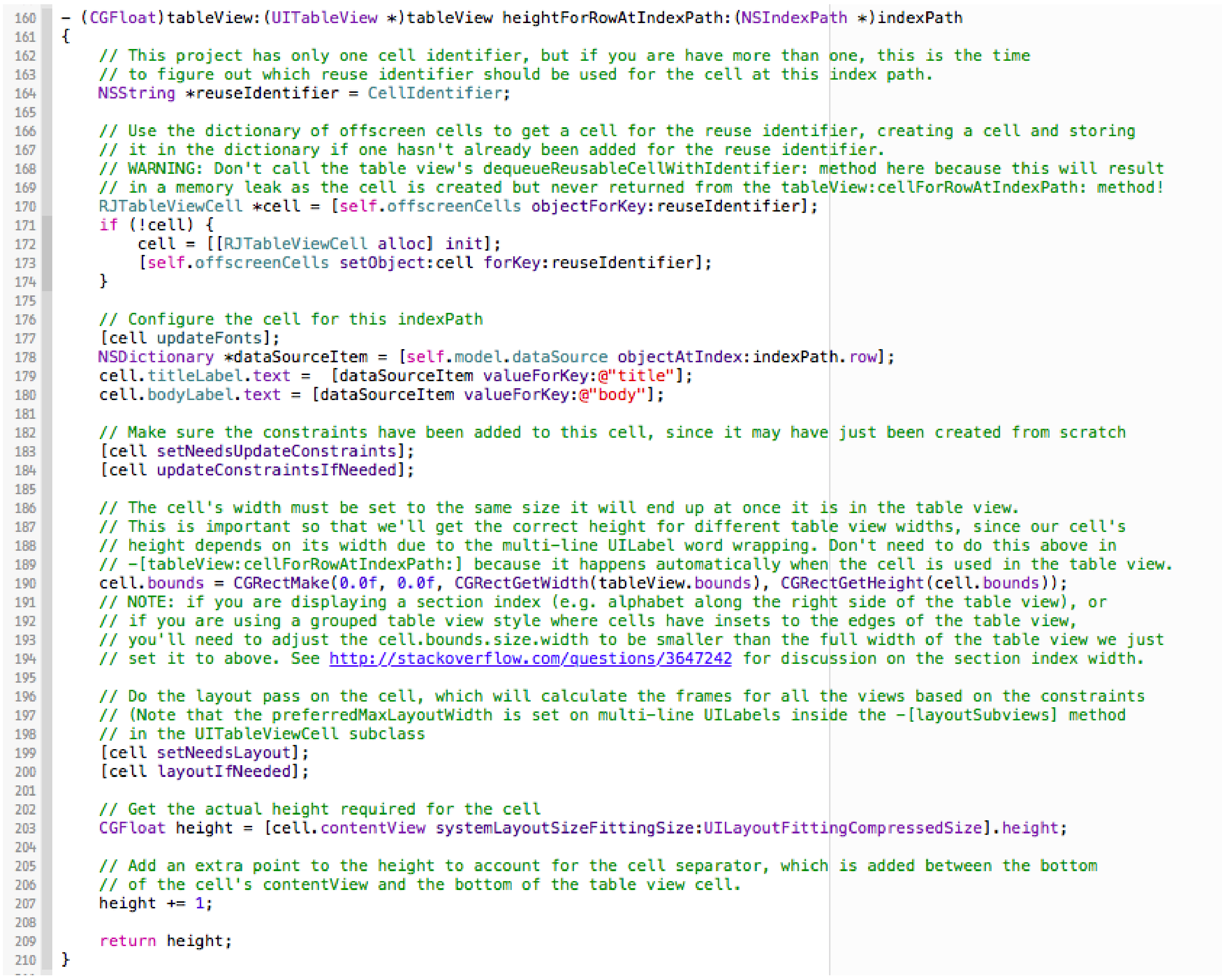
RJTableViewCell.m

TableViewCellWithAutoLayout 〉 TableVie...toLayout 〉 TableViewController 〉 RJTableViewCell.m 〉 -initWithStyle:reuseIdentifier

```objc
- (void)updateConstraints
{
    if (!self.didSetupConstraints) {
        // Note: if the constraints you add below require a larger cell size than the current size (which is likely to
            be the default size {320, 44}), you'll get an exception.
        // As a fix, you can temporarily increase the size of the cell's contentView so that this does not occur using
            code similar to the line below.
        //      See here for further discussion: https://github.com/Alex311/TableCellWithAutoLayout/commit/
            bde387b27e33605eeac3465475d2f2ff9775f163#commitcomment-4633188
        // self.contentView.bounds = CGRectMake(0.0f, 0.0f, 99999.0f, 99999.0f);

        [UIView autoSetPriority:UILayoutPriorityRequired forConstraints:^{
            [self.titleLabel autoSetContentCompressionResistancePriorityForAxis:ALAxisVertical];
        }];
        [self.titleLabel autoPinEdgeToSuperviewEdge:ALEdgeTop withInset:kLabelVerticalInsets];
        [self.titleLabel autoPinEdgeToSuperviewEdge:ALEdgeLeading withInset:kLabelHorizontalInsets];
        [self.titleLabel autoPinEdgeToSuperviewEdge:ALEdgeTrailing withInset:kLabelHorizontalInsets];

        // This is the constraint that connects the title and body labels. It is a "greater than or equal" inequality so
            that if the row height is
        // slightly larger than what is actually required to fit the cell's subviews, the extra space will go here.
            (This is the case on iOS 7
        // where the cell separator is only 0.5 points tall, but in the tableView:heightForRowAtIndexPath: method of the
            view controller, we add
        // a full 1.0 point in extra height to account for it, which results in 0.5 points extra space in the cell.)
        // See https://github.com/smileyborg/TableViewCellWithAutoLayout/issues/3 for more info.
        [self.bodyLabel autoPinEdge:ALEdgeTop toEdge:ALEdgeBottom ofView:self.titleLabel withOffset:kLabelVerticalInsets
            relation:NSLayoutRelationGreaterThanOrEqual];

        [UIView autoSetPriority:UILayoutPriorityRequired forConstraints:^{
            [self.bodyLabel autoSetContentCompressionResistancePriorityForAxis:ALAxisVertical];
        }];
        [self.bodyLabel autoPinEdgeToSuperviewEdge:ALEdgeLeading withInset:kLabelHorizontalInsets];
        [self.bodyLabel autoPinEdgeToSuperviewEdge:ALEdgeTrailing withInset:kLabelHorizontalInsets];
        [self.bodyLabel autoPinEdgeToSuperviewEdge:ALEdgeBottom withInset:kLabelVerticalInsets];

        self.didSetupConstraints = YES;
    }

    [super updateConstraints];
}

- (void)layoutSubviews
{
    [super layoutSubviews];

    // Make sure the contentView does a layout pass here so that its subviews have their frames set, which we
    // need to use to set the preferredMaxLayoutWidth below.
    [self.contentView setNeedsLayout];
    [self.contentView layoutIfNeeded];

    // Set the preferredMaxLayoutWidth of the mutli-line bodyLabel based on the evaluated width of the label's frame,
    // as this will allow the text to wrap correctly, and as a result allow the label to take on the correct height.
    self.bodyLabel.preferredMaxLayoutWidth = CGRectGetWidth(self.bodyLabel.frame);
}
```

Screen Shot 2015-06-01 at 15.20.19.png

代码片段：UITableViewDelegate



```objc
- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // This project has only one cell identifier, but if you are have more than one, this is the time
    // to figure out which reuse identifier should be used for the cell at this index path.
    NSString *reuseIdentifier = CellIdentifier;

    // Use the dictionary of offscreen cells to get a cell for the reuse identifier, creating a cell and storing
    // it in the dictionary if one hasn't already been added for the reuse identifier.
    // WARNING: Don't call the table view's dequeueReusableCellWithIdentifier: method here because this will result
    // in a memory leak as the cell is created but never returned from the tableView:cellForRowAtIndexPath: method!
    RJTableViewCell *cell = [self.offscreenCells objectForKey:reuseIdentifier];
    if (!cell) {
        cell = [[RJTableViewCell alloc] init];
        [self.offscreenCells setObject:cell forKey:reuseIdentifier];
    }

    // Configure the cell for this indexPath
    [cell updateFonts];
    NSDictionary *dataSourceItem = [self.model.dataSource objectAtIndex:indexPath.row];
    cell.titleLabel.text =  [dataSourceItem valueForKey:@"title"];
    cell.bodyLabel.text = [dataSourceItem valueForKey:@"body"];

    // Make sure the constraints have been added to this cell, since it may have just been created from scratch
    [cell setNeedsUpdateConstraints];
    [cell updateConstraintsIfNeeded];

    // The cell's width must be set to the same size it will end up at once it is in the table view.
    // This is important so that we'll get the correct height for different table view widths, since our cell's
    // height depends on its width due to the multi-line UILabel word wrapping. Don't need to do this above in
    // -[tableView:cellForRowAtIndexPath:] because it happens automatically when the cell is used in the table view.
    cell.bounds = CGRectMake(0.0f, 0.0f, CGRectGetWidth(tableView.bounds), CGRectGetHeight(cell.bounds));
    // NOTE: if you are displaying a section index (e.g. alphabet along the right side of the table view), or
    // if you are using a grouped table view style where cells have insets to the edges of the table view,
    // you'll need to adjust the cell.bounds.size.width to be smaller than the full width of the table view we just
    // set it to above. See http://stackoverflow.com/questions/3647242 for discussion on the section index width.

    // Do the layout pass on the cell, which will calculate the frames for all the views based on the constraints
    // (Note that the preferredMaxLayoutWidth is set on multi-line UILabels inside the -[layoutSubviews] method
    // in the UITableViewCell subclass
    [cell setNeedsLayout];
    [cell layoutIfNeeded];

    // Get the actual height required for the cell
    CGFloat height = [cell.contentView systemLayoutSizeFittingSize:UILayoutFittingCompressedSize].height;

    // Add an extra point to the height to account for the cell separator, which is added between the bottom
    // of the cell's contentView and the bottom of the table view cell.
    height += 1;

    return height;
}
```

Screen Shot 2015-06-01 at 15.21.21.png

源代码： TableViewCellWithAutoLayout