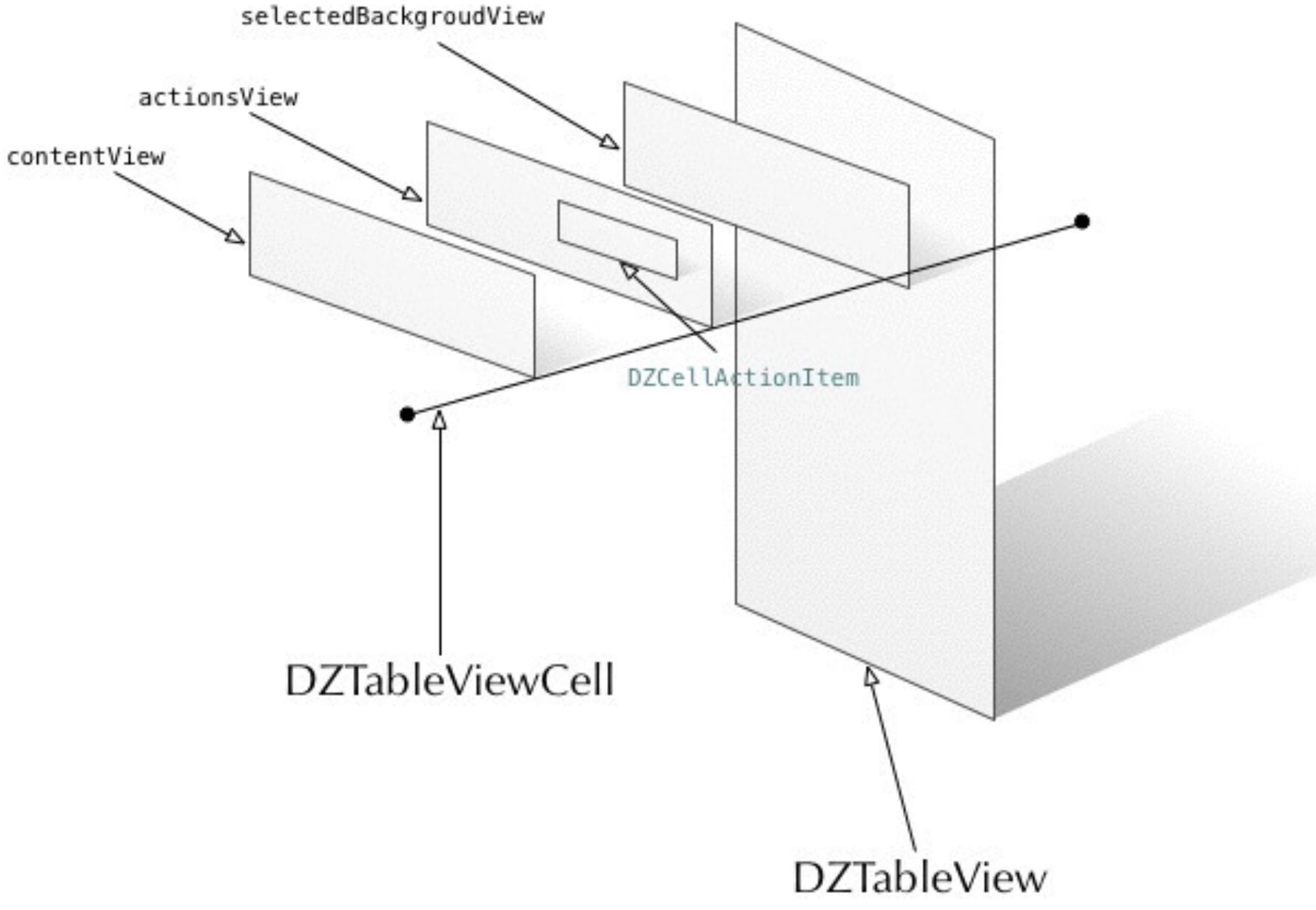
构建TableView

——理解IOS UI编程



StoneDong

主要内容

1. UIKit理解
2. 构建TableView
3. 定制ViewController
4. 如何在实践中应用设计模式以及一些设计思想
5. TableView可扩展性探讨

有的放矢，百发百中

我们要一个什么样的TableView

能够展示不定数量的视图，而且是表视图

删除一个Cell

增加一个Cell

要有动画，一定要有丝滑的动画

下拉刷新这么流行也该有吧

对于使用者应该有个“人性化”的接口

右滑删除应该有

最好右滑还能扩展功能

背景可以换

能够定制表头和表尾的视图

....

接口

告诉开发者我能干什么

布局

我的确能干什么

交互

我怎么和用户打情骂俏的

为了实现TableView我们大概需要那些对象

进行表示图布局
接收用户操作 ..... DZTableView

展示被布局的内容 ..... DZTableViewCell

提供展示数据 ..... DZTableViewDataSource

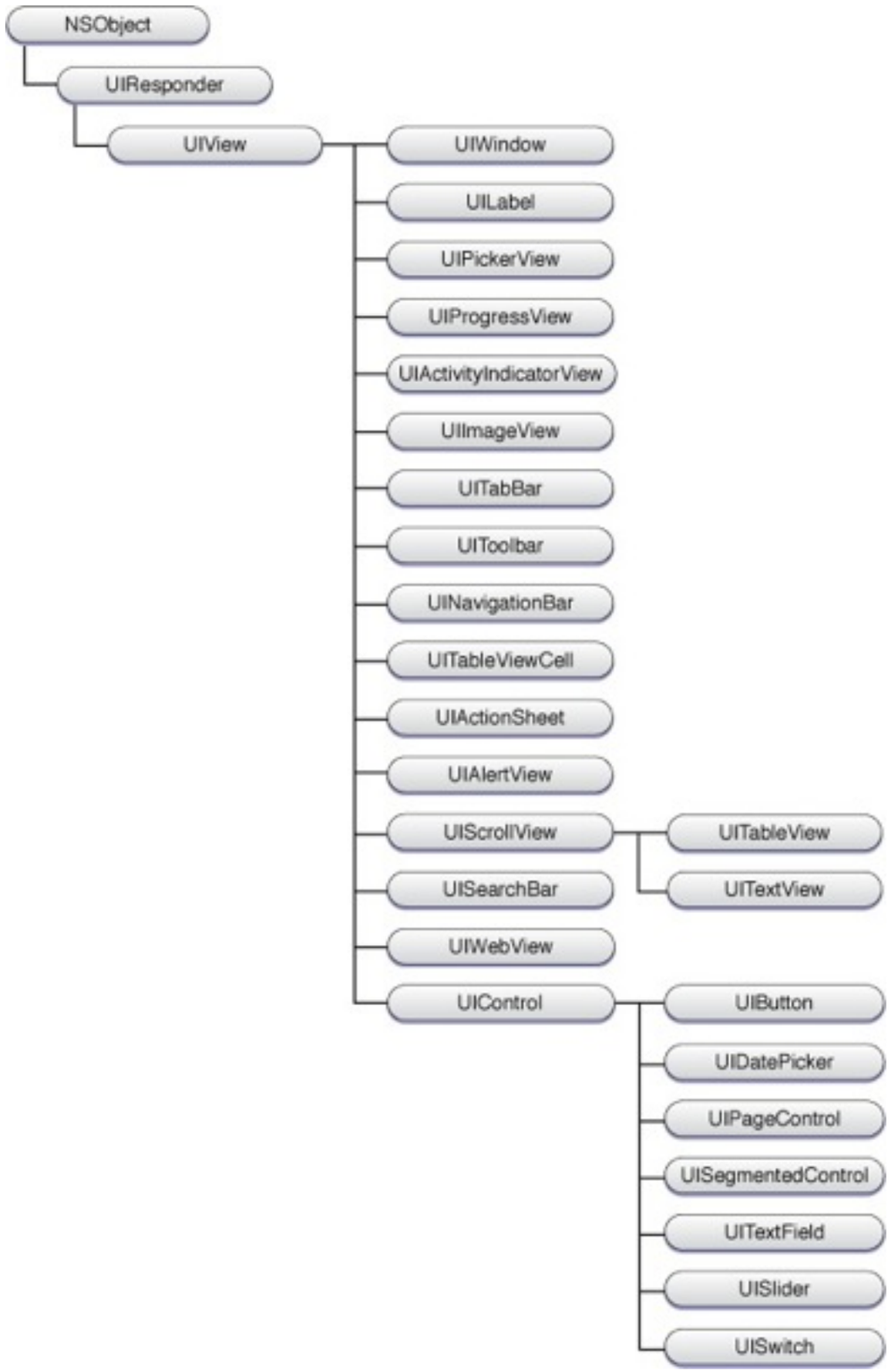
对用户操作作出响应 ..... DZTableViewActionDelegate

合抱之木，生于毫末
九层之台，起于累土
千里之行，始于足下

UIKit印象

我们在讲一个故事：

1. 时间
2. 地点
3. 人物
4. 故事情节

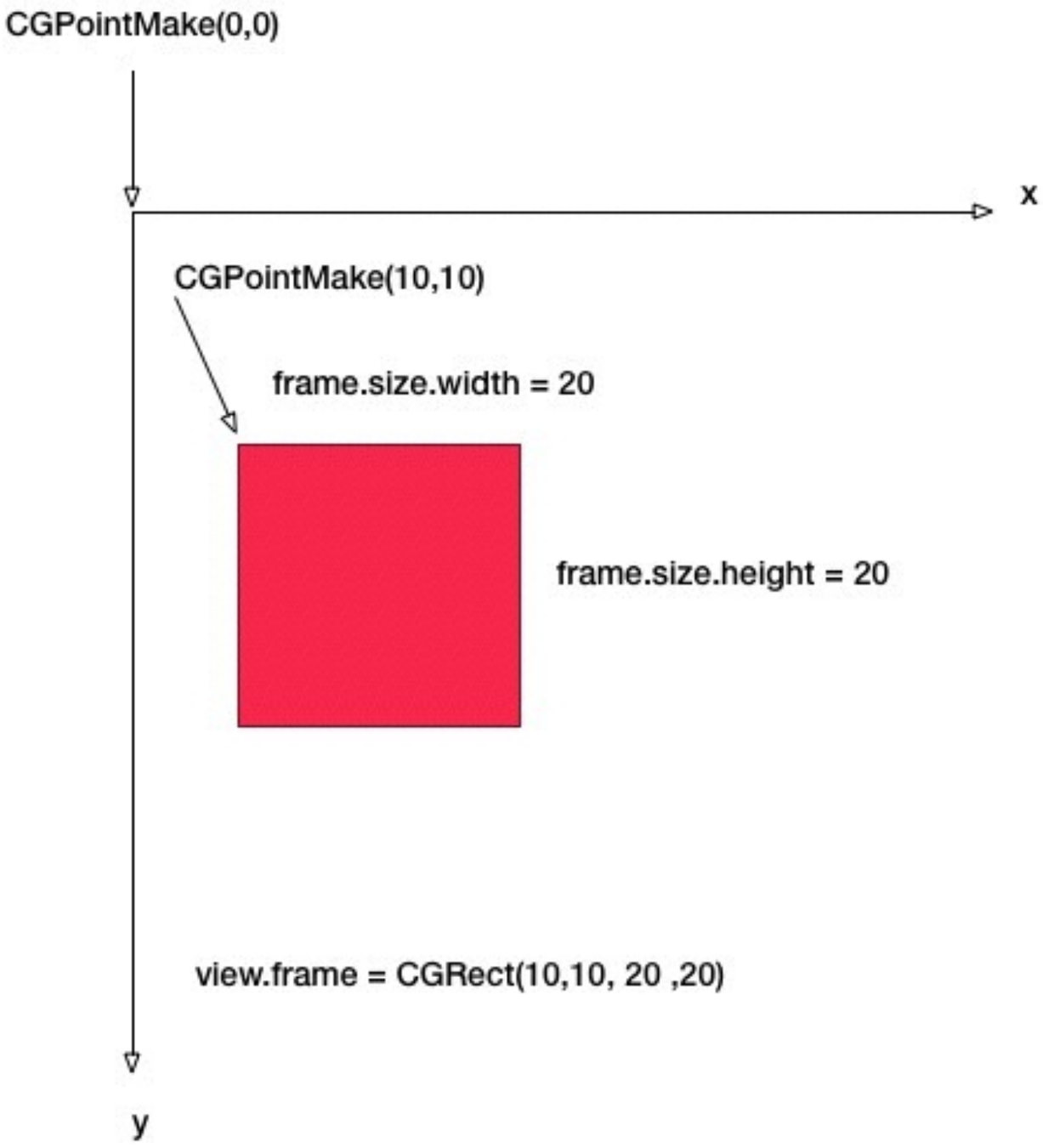


几何布局

```
struct CGPoint {
   CGFloat x;
   CGFloat y;
};
typedef struct CGPoint CGPoint;
```

```
struct CGSize {
   CGFloat width;
   CGFloat height;
};
typedef struct CGSize CGSize;
```

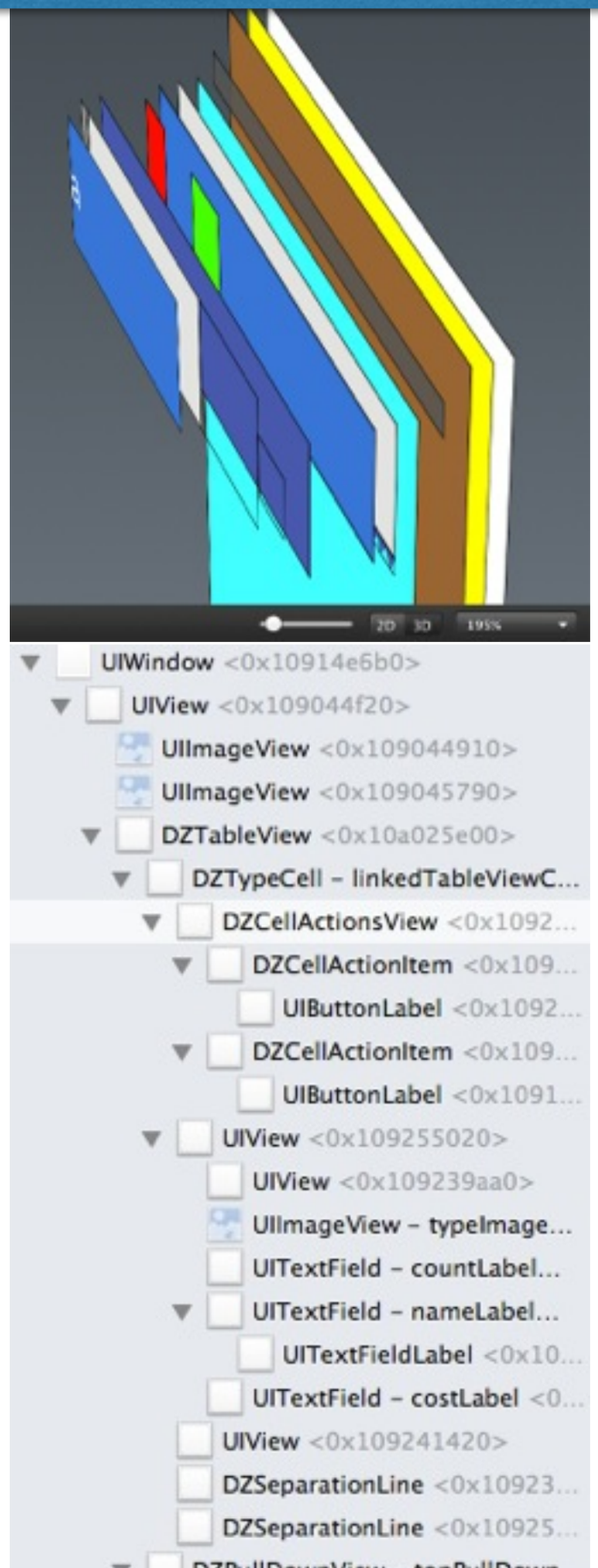
```
struct CGRect {
   CGPoint origin;
   CGSize size;
};
typedef struct CGRect CGRect;
```



Z-order

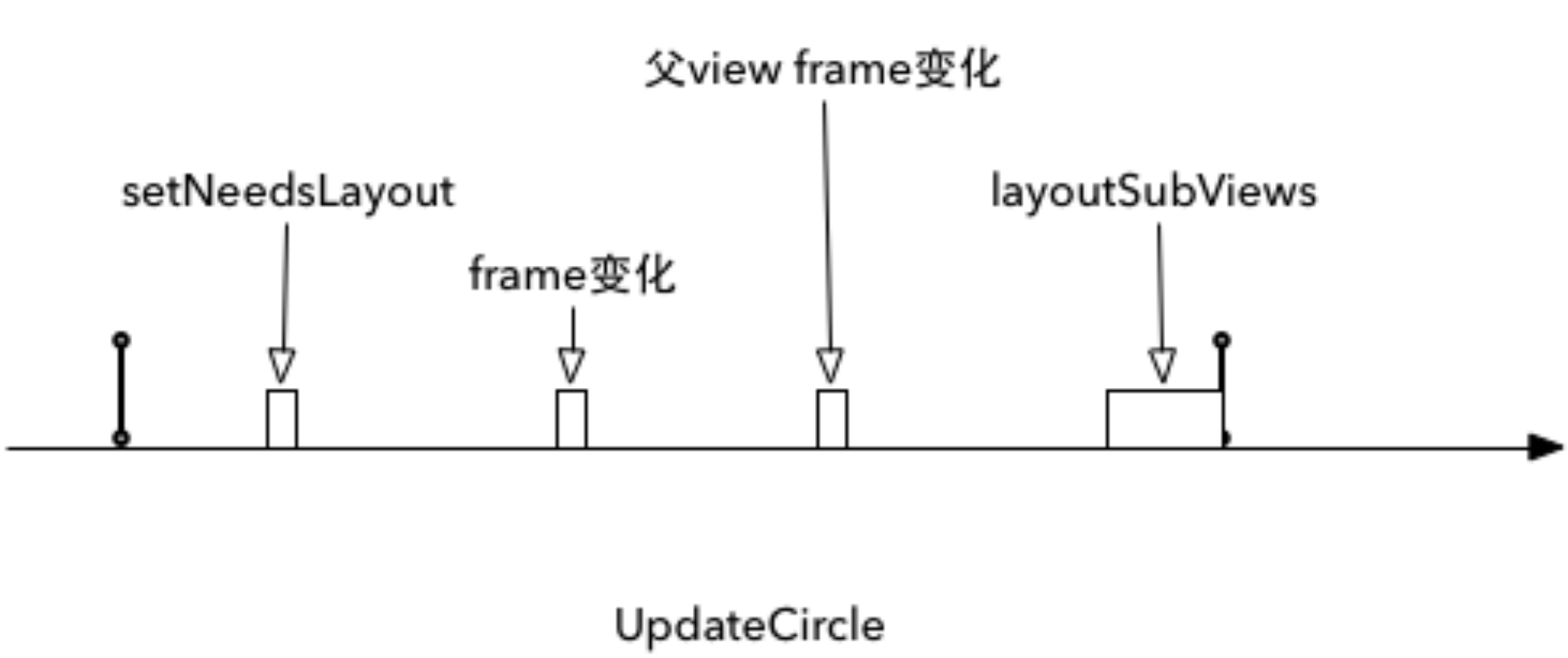
view们像千层饼一样一层叠一层

- superview property
- subviews property
- window property
- – addSubview:
- – bringSubviewToFront:
- – sendSubviewToBack:
- – removeFromSuperview
- – insertSubview:atIndex:
- – insertSubview:aboveSubview:
- – insertSubview:belowSubview:
- – exchangeSubviewAtIndex:withSubviewAtIndex:
- – isDescendantOfView:



什么时候布局（在那个函数中布局）

因为UIKit使用了延迟布局的策略，所以布局的时候我们往往是setNeedsLayout置一下需要布局的标志位，在合适的时机，系统会调用UIView的layoutSubviews等函数进行布局。



PS：切记不要在initWithFrame中进行几何布局

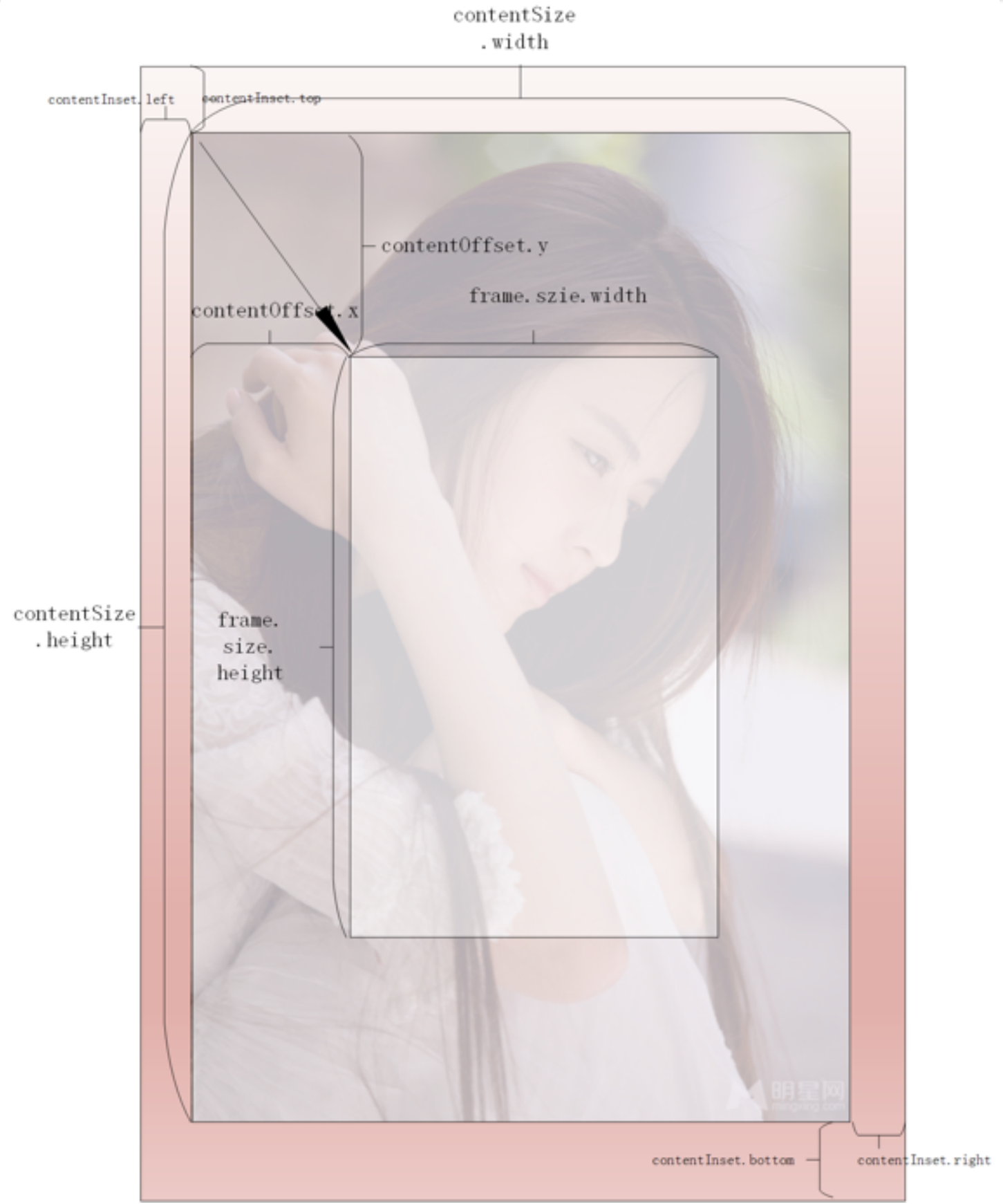
UIScrollView

frame：视窗大小

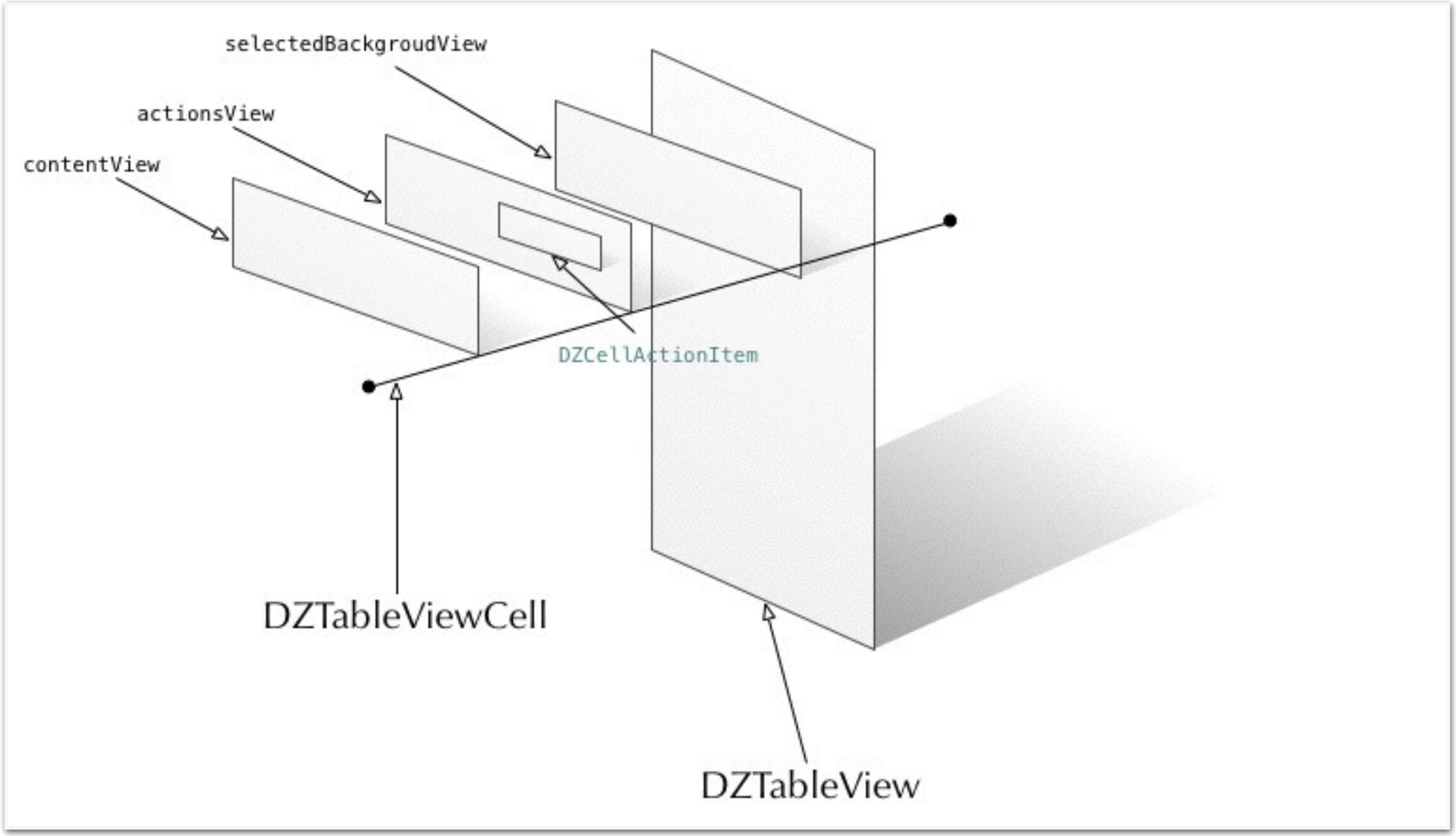
contentSize：画布大小

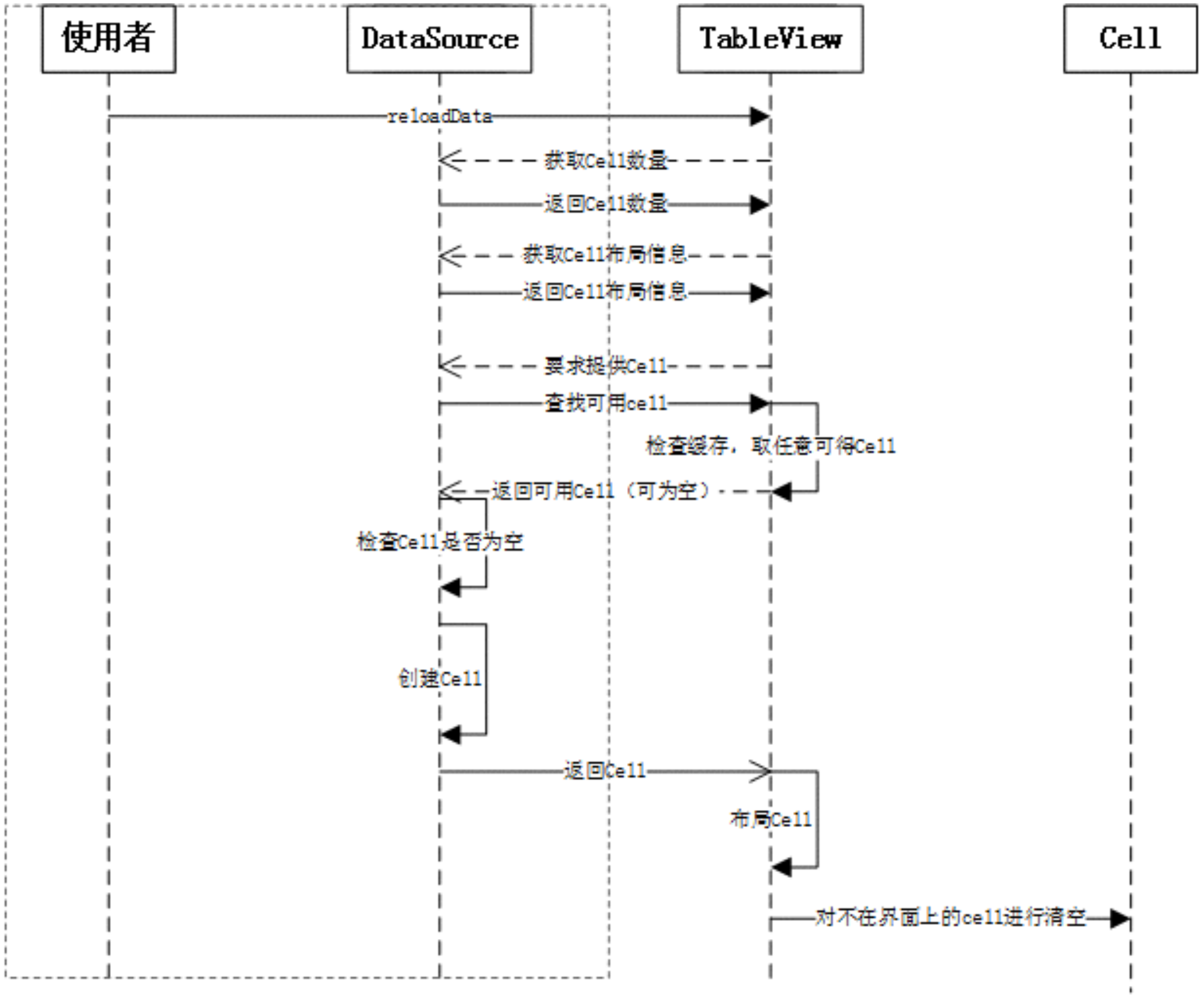
contentOffset：视窗偏移

contentInset：画布扩展

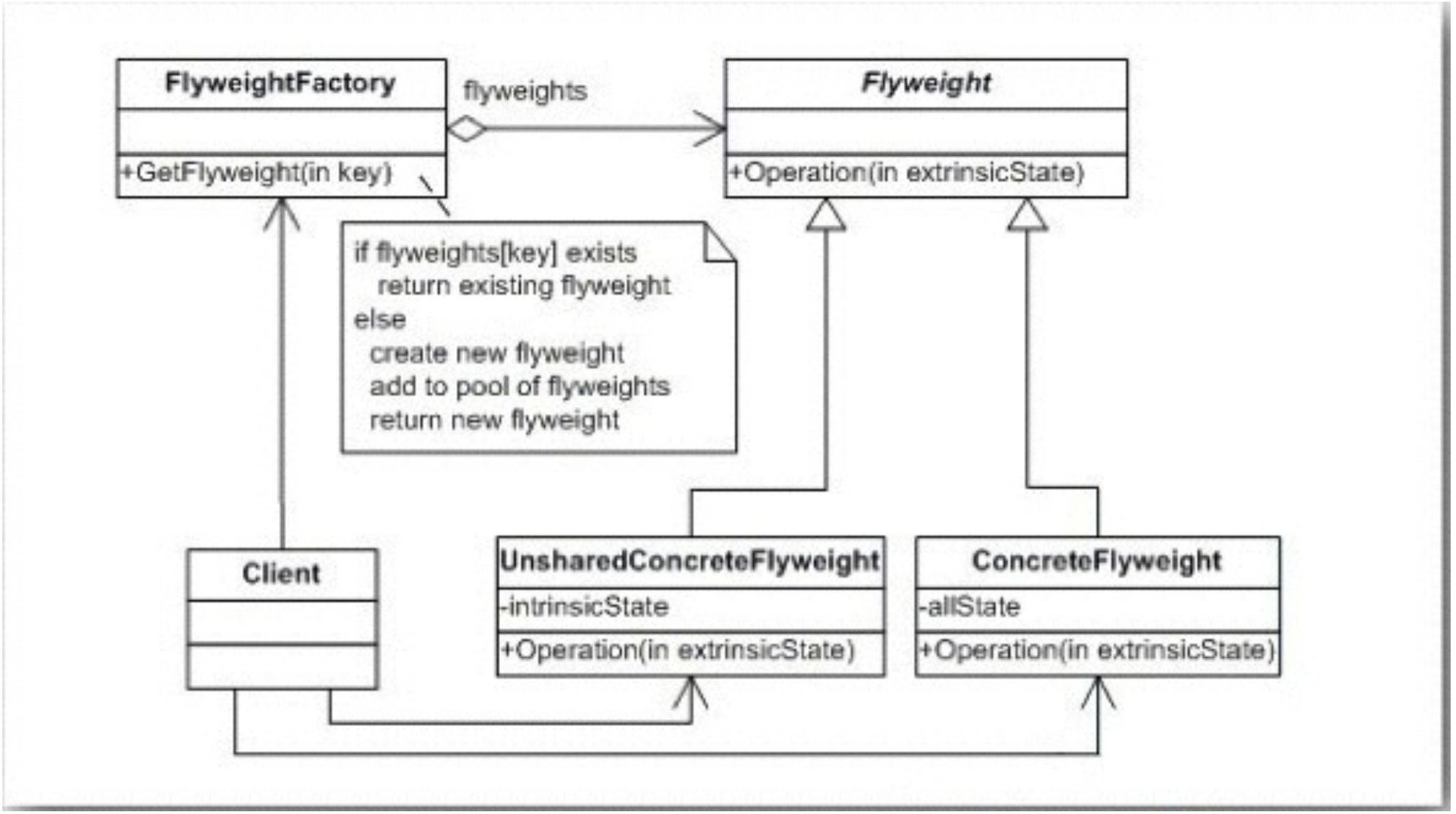


子类化实现布局


selectedBackgroudView
actionsView
contentView
DZCellActionItem
DZTableViewCell
DZTableView


使用者
DataSource
TableView
Cell
reloadData
获取Cell数量
返回Cell数量
获取Cell布局信息
返回Cell布局信息
要求提供Cell
查找可用cell
检查缓存，取任意可得Cell
返回可用Cell（可为空）
检查Cell是否为空
创建Cell
返回Cell
布局Cell
对不在界面上的cell进行清空

Cell的重用-享元模式



```
////////////////////////////////////////////////////////////////////////////
@protocol DZTableViewActionDelegate <NSObject>

- (void) dzTableView:(DZTableView*)tableView didTapAtRow:(NSInteger)row;
- (void) dzTableView:(DZTableView *)tableView deleteCellAtRow:(NSInteger)row;
- (void) dzTableView:(DZTableView *)tableView editCellDataAtRow:(NSInteger)row;

@end
////////////////////////////////////////////////////////////////////////////
@protocol DZTableViewSourceDelegate <NSObject>
- (NSInteger) numberOfRowsInDZTableView:(DZTableView*)tableView;
- (DZTableViewCell*) dzTableView:(DZTableView*)tableView cellAtRow:(NSInteger)row;
- (CGFloat) dzTableView:(DZTableView*)tableView cellHeightAtRow:(NSInteger)row;
@end
////////////////////////////////////////////////////////////////////////////
@interface DZTableView : UIScrollView
DEFINE_PROPERTY_STRONG(UIColor*, gradientColor);
@property (nonatomic, strong) UIImageView* backgroudView;
@property (nonatomic, strong, readonly) NSArray* visibleCells;
@property (nonatomic, weak) id<DZTableViewActionDelegate> actionDelegate;
@property (nonatomic, weak) id<DZTableViewSourceDelegate> dataSource;
@property (nonatomic, assign) NSInteger selectedIndex;
@property (nonatomic, strong) DZPullDownView* topPullDownView;
DEFINE_PROPERTY_STRONG(UIView*, bottomView);
- (DZTableViewCell*) dequeueDZTalbeViewCellForIdentifiy:(NSString*)identifiy;
- (void) reloadData;
- (void) insertRowAt:(NSSet *)rowsSet withAnimation:(BOOL)animation;
- (void) removeRowAt:(NSInteger)row withAnimation:(BOOL)animation;
- (void) manuSelectedRowAt:(NSInteger)row;
@end
////////////////////////////////////////////////////////////////////////////
@interface DZTableViewCell : UIView
{
    UIView* _contentView;
}
DEFINE_PROPERTY_STRONG(DZSeparationLine*, topSeperationLine);
DEFINE_PROPERTY_STRONG(DZSeparationLine*, bottomSeperationLine);
DEFINE_PROPERTY_STRONG(CAGradientLayer*, gradientLayer);
@property (nonatomic, strong) UIView* contentView;
@property (nonatomic, strong) DZCellActionsView* actionsView;
@property (nonatomic, strong) UIView* selectedBackgroudView;
@property (nonatomic, assign) BOOL isSelected;
- (instancetype) initWithIdentifiy:(NSString*)identifiy;
- (void) showGradientStart:(UIColor*)startColor endColor:(UIColor*)end;
@end
```

交互

核心数据结构

UIEvent

```
– locationInView:
type property
subtype property
timestamp property
```

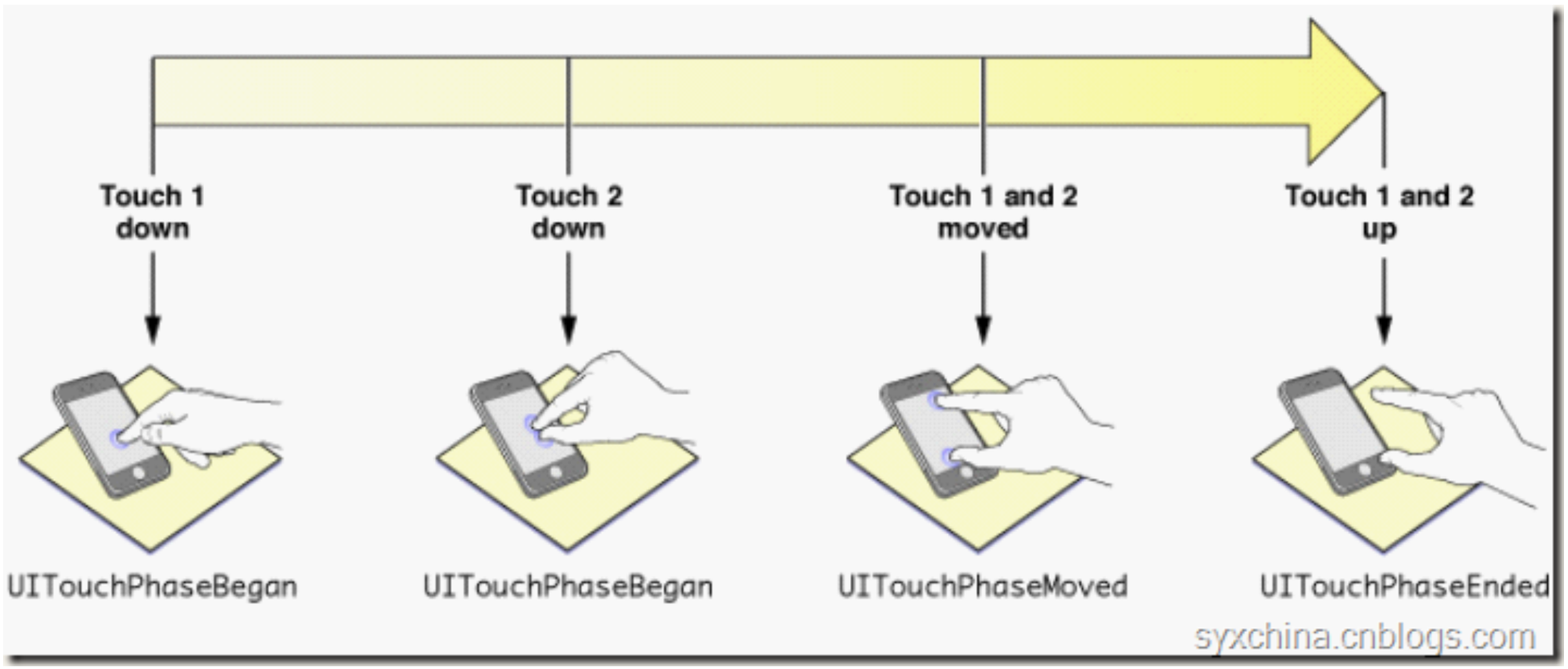
UITouch

```
tapCount property
timestamp property
phase  property
….
– locationInView:
```

UIResponse

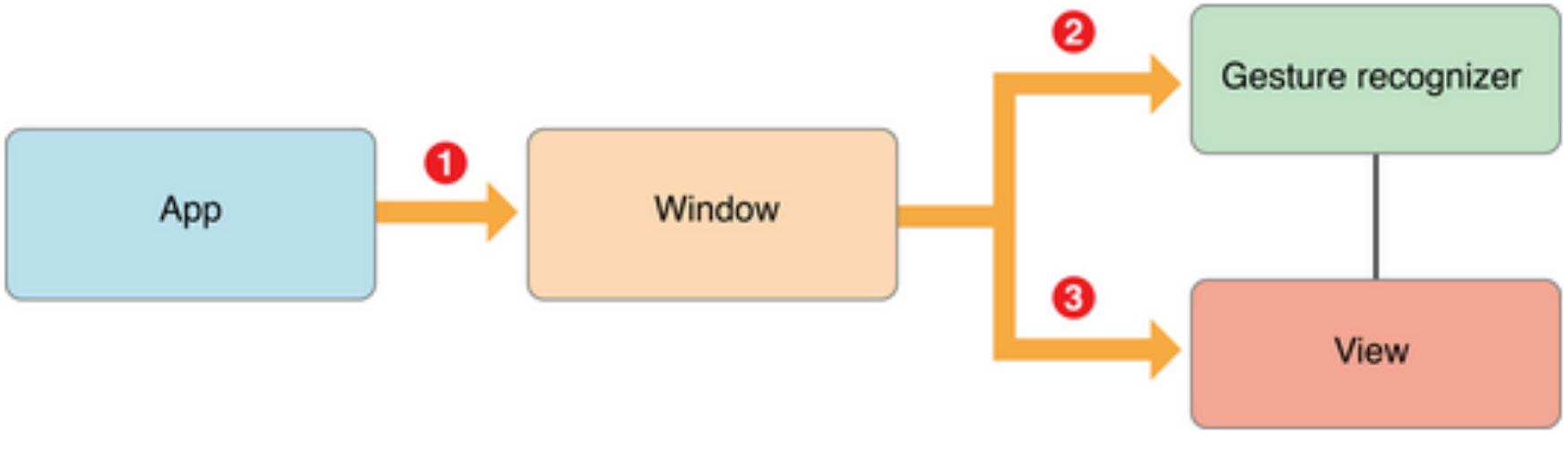
触摸事件

```
-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event;

-(void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event;

-(void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event;

-(void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event;
```



UIKit的事件分发机制是典型的责任链模式

假设一个单击事件发生在了View D里面，系统首先会从最顶层的View A开始寻找，发现事件是在View A或者其子类里面，那么接着从B和C找，发现事件是在C或者其子类里面，那么接着到C里面找，这时发现事件是在D里面，并且D已经没有子类了，那么hit-test view就是View D啦。



UIGestureRecognizer

当我们使用手势的时候，很明显不能每一个View都实现一遍响应触摸时间的所有函数。我们需要一个非常好用的对于事件处理的封装，apple为我们提供了UIGestureRecognizer

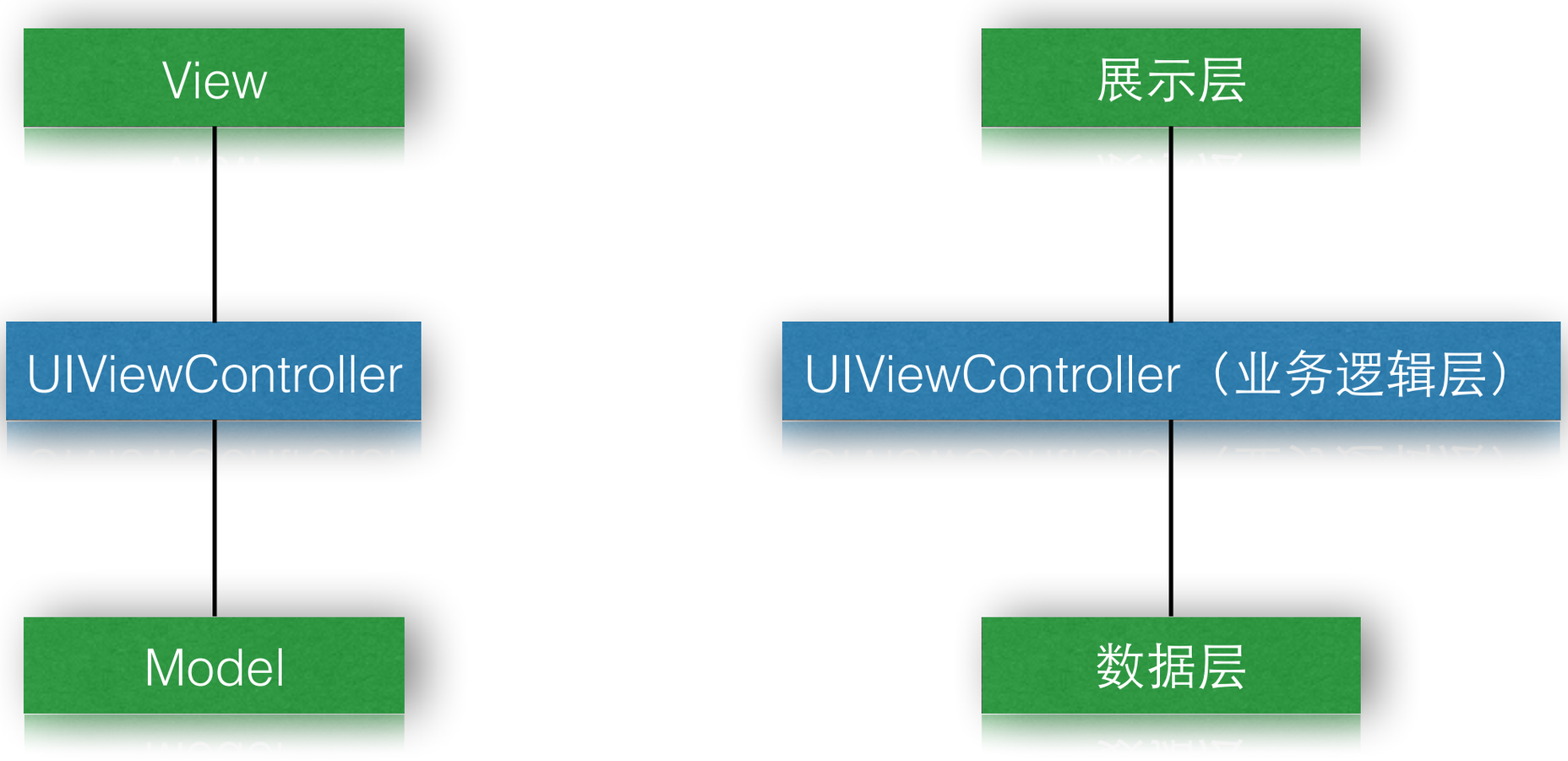
- UITapGestureRecognizer
- UIPinchGestureRecognizer
- UIRotationGestureRecognizer
- UISwipeGestureRecognizer
- UIPanGestureRecognizer
- UIScreenEdgePanGestureRecognizer
- UILongPressGestureRecognizer

响应和处理事件

```
- (void) addTapTarget:(id)target selector:(SEL)selecotr
{
    self.userInteractionEnabled = YES;
    UITapGestureRecognizer* tapGesture = [[UITapGestureRecognizer alloc] initWithTarget:target action:selecotr];
    tapGesture.numberOfTapsRequired = 1;
    tapGesture.numberOfTouchesRequired = 1;
    [self addGestureRecognizer:tapGesture];
}
...
[self addTapTarget:self selector:@selector(handleTapGestrue:)];
...
- (void) handleTapGestrue:(UITapGestureRecognizer*)tapGestrue
{
    CGPoint point = [tapGestrue locationInView:self];
    NSArray* cells = _visibleCellsMap.allValues;
    for (DZTableViewCell* each in cells) {
        CGRect rect = each.frame;
        if (CGRectContainsPoint(rect, point)) {
            if ([_actionDelegate respondsToSelector:@selector(dzTableView:didTapAtRow:)]) {
                [_actionDelegate dzTableView:self didTapAtRow:each.index];
            }
            each.isSelected = YES;
            _selectedIndex = each.index;
        }
        else
        {
            each.isSelected = NO;
        }
    }
}
```

UIViewController

在UIKit框架使用的MVC模式中UIViewController属于C层，介于View层和Model层之间。同是也是三层架构中的业务逻辑层。所以很多时候，他是数据层向界面层传递信息的通路，也是控制界面逻辑的管理者。可谓是责任重大。而往往我们很大一部分的界面相关的工作都是在某个ViewController中完成的。



DZTableViewController

```
@interface DZTableViewController : UIViewController <DZTableViewSourceDelegate, DZTableViewActionDelegate>
@property (nonatomic, strong) UIImageView* headerView;
@property (nonatomic, strong) UIImageView* backgroudView;
@property (nonatomic, strong ,readonly) DZTableView* tableView;
@end
……….
- (DZTableView*) tableView
{
    if (!_tableView) {
        _tableView = [[DZTableView alloc] initWithFrame:CGRectLoadViewFrame];
        _tableView.dataSource = self;
        _tableView.delegate = self;
        _tableView.actionDelegate = self;
    }
    return _tableView;
}
- (void) loadView
{
    DZTableView* tableView = self.tableView;
    DZPullDownView* pullView = [[DZPullDownView alloc] init];
    pullView.height = 44;
    pullView.delegate = self;
    tableView.topPullDownView = pullView;
}
- (void)viewDidLoad
{
    [super viewDidLoad];
}
/*
 *  – viewWillAppear:
 *  – viewDidAppear:
 *  – viewWillDisappear:
 *  – viewDidDisappear:
 *  – viewWillLayoutSubviews
 *  – viewDidLayoutSubviews
```

位置相戾，有画处多属赘疣

虚实相生，无画处皆成妙境

可扩展性探讨

程序员不是神仙不可能预知到将来产品会有什么样的需求，设计会要求什么样的UI。但是我们能够遇见的是，虽然现在程序界面是这个样子，将来肯定不是这个这个样子。面对越来越多的变化，如何应对?

- 原型模式，定制Cell扩展功能
- 依赖导致，细节要依赖抽象，抽象不能够依赖细节。footerView和HeaderView只记录了一个变量和确定了布局的方式，具体是什么类型和内容不去管它
- 职责分离，数据变了动dataSource，布局变了动TableView，避免大手术
- 。。。。

高楼虽然复杂，但是砖与瓦的结构却极其简单，真正复杂的是如何组合砖与瓦，钢筋与混凝土。
那就是设计。

Q&A

THAKNS