UIKit相关

最开始

main.m

整个程序的入口，UIApplication涉及到delegate设计模式，其中有很多功能需要实现，他通过在一个protocol上面声明了这些接口，然后把接口交给了AppDelegate去实现（没了解过，来自 iOS学习 ）

AppDelegate

根VC，声明了一个UIWindow *window，是程序的主界面
application didFinishLaunchingWithOptions：当应用程序启动时执行，应用程序启动入口，只在应用程序启动时执行一次。若用户直接启动，launchOptions内无数据,若通过其他方式启动应用，launchOptions包含对应方式的内容。
applicationWillResignActive：在应用程序将要由活动状态切换到非活动状态时候，要执行的委托调用，如按下 home 按钮，返回主屏幕，或全屏之间切换应用程序等。
applicationDidEnterBackground：在应用程序已进入后台程序时，要执行的委托调用。
applicationWillEnterForeground：在应用程序将要进入前台时(被激活)，要执行的委托调用，刚好与applicationWillResignActive 方法相对应。
applicationDidBecomeActive：在应用程序已被激活后，要执行的委托调用，刚好与applicationDidEnterBackground 方法相对应。
applicationWillTerminate：在应用程序要完全推出的时候，要执行的委托调用，这个需要要设置UIApplicationExitsOnSuspend的键值。

ViewController

是MVC模式里的Control，负责view的控制，以及其他VC的通信、控制等
视图控制器，一种是切换控制器UITabBarController(标签栏控制器，表现出一个array的结构，各个ViewController是并列的)，一种是导航控制器UINavigationController（表现出一种stack结构，push一个ViewController或pop一次）
内容显示，如ViewController，TableViewController(表格图控制器)等

生命周期

```
1 + (void)initialize {
2 NSLog(@"========   类初始化方法：  initialize   =======\n");
```

```
}
- (instancetype)init {
self = [super init];
NSLog(@"========   实例初始化方法：  init   =======\n");
return self;
}
- (instancetype)initWithCoder:(NSCoder *)aDecoder {
self = [super initWithCoder:aDecoder];
NSLog(@"========   从归档初始化：   initWithCoder:(NSCoder *)aDecoder   =======\n");
return self;
}
- (void)loadView {
[super loadView];
NSLog(@"========   加载视图：  loadView   =======\n");
}
#pragma mark- life cycle
- (void)viewDidLoad {
[super viewDidLoad];
NSLog(@"========   将要加载视图：  viewDidLoad   =======\n");
}
- (void)viewWillLayoutSubviews {
[super viewWillLayoutSubviews];
NSLog(@"========   将要布局子视图：  viewWillLayoutSubviews   =======\n");
}
- (void)viewDidLayoutSubviews {
[super viewDidLayoutSubviews];
NSLog(@"========   已经布局子视图：  viewDidLayoutSubviews   =======\n");
}
- (void)didReceiveMemoryWarning {
[super didReceiveMemoryWarning];
NSLog(@"========   收到内存警告：  didReceiveMemoryWarning   =======\n");
}
- (void)viewWillAppear:(BOOL)animated {
[super viewWillAppear:animated];
NSLog(@"========   视图将要出现：  viewWillAppear:(BOOL)animated   =======\n");
```

```
}
- (void)viewDidAppear:(BOOL)animated {
[super viewDidAppear:animated];
NSLog(@"========   视图已经出现：  viewDidAppear:(BOOL)animated   =======\n");
}
- (void)viewWillDisappear:(BOOL)animated {
[super viewWillDisappear:animated];
NSLog(@"========   视图将要消失：  viewWillDisappear:(BOOL)animated   =======\n");
}
- (void)viewDidDisappear:(BOOL)animated {
[super viewDidDisappear:animated];
NSLog(@"========   视图已经消失：  viewDidDisappear:(BOOL)animated   =======\n");
}
- (void)dealloc {
NSLog(@"========   释放：  dealloc   =======\n");
}

// 在AppDelegate的didFinishLaunchingWithOptions使用
RootViewController *rc = [[RootViewController alloc] init];
self.window.rootViewController = rc;

// 在loadView使用代码创建view
UIView *view = [[UIView alloc] initWithFrame: [UIScreen
mainScreen].applicationFrame];
self.view = view;
```

loadView 除非手动调用，在生命周期内只调用一次
viewDidLoad 是我们最常用的方法，类成员对象和变量的初始化我们都会放在这个方法中，
viewDidLoad之后view才算加载成功，viewDidLoad是可以多次调用的（多次加载）

viewWillAppear：控制器的view将要显示
viewWillLayoutSubviews：控制器的view将要布局子控件
viewDidLayoutSubviews：控制器的view布局子控件完成
viewDidAppear：控制器的view完全显示
viewWillDisappear：控制器的view即将消失的时候
viewDidDisappear：控制器的view完全消失的时候

dealloc：释放

UIWindow

UIWindow对象是所有UIView的根视图，管理和协调的应用程序的显示、分发事件给View。UIWindow类是UIView的子类，可以看作是特殊的UIView。一般应用程序只有一个UIWindow对象，即使有多个UIWindow对象，也只有一个UIWindow可以接受到用户的触屏事件。UIWindow初始化在AppDelegate里面的didFinishLaunchingWithOptions方法。

```
self.window = [[[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]] autorelease];
self.window.backgroundColor = [UIColor whiteColor];
[self.window makeKeyAndVisible]; // 显示window
```

UIView

视图基础元素
什么是控件？屏幕上所有UI元素都是控件
基本属性：尺寸、位置、背景颜色等
所有控件都最终继承自UIView
每一个控制器（UIViewController）内部都有默认UIView *view，控制器中管理的其他所有控件都是这个view的子控件，控制器本身是不可见的
每个控件都是一个容器，可以将其他控件放进该控件的内部，主要还是将UIView作为容器
在UIKit中，坐标(0,0)是左上角
UIViewController 控制器：管理UI界面以及事件处理，要想关联UI界面的元素要把方法返回值修改IBAction。

```
+ @property(nonatomic,readonly) UIView *superview;
//获得自己的父控件对象

+ @property(nonatomic,readonly,copy) NSArray *subviews;
//获得自己的所有子控件对象

+ @property(nonatomic) NSInteger tag;
//控件的ID(标识)，父控件可以通过tag来找到对应的子控件

+ @property(nonatomic) CGAffineTransform transform;
// 控件的形变属性(可以设置旋转角度、比例缩放、平移等属性)

```

```
+ @property(nonatomic) CGRect frame;
// 控件矩形框在父控件中的位置和尺寸(以父控件的左上角为坐标原点)

+ @property(nonatomic) CGRect bounds;
//控件矩形框的位置和尺寸(以自己左上角为坐标原点，所以bounds的x、y一般为0)

+ @property(nonatomic) CGPoint center;
// 控件中点的位置(以父控件的左上角为坐标原点)

- (void)addSubview:(UIView *)view;
//添加一个子控件view

- (void)removeFromSuperview;
//从父控件中移除

- (UIView *)viewWithTag:(NSInteger)tag;
//根据一个tag标识找出对应的控件（一般都是子控件）

UIView *view1=[[UIView alloc]initWithFrame:CGRectMake(60, 50, 200, 100)];
```

UIView的自适应高度

UIButton

默认情况（Default） 对应的枚举常量：UIControlStateNormal
highlighted（高亮状态） 按钮被按下去的时候（手指还未松开） 对应的枚举常量：UIControlStateHighlighted
disabled（失效状态，不可用状态） 如果enabled属性为NO，就是处于disable状态，代表按钮不可以被点击 对应的枚举常量：UIControlStateDisabled

```
typedef enum {
UIButtonTypeCustom = 0,           // no button type   自定义，无风格
UIButtonTypeRoundedRect,          // rounded rect, flat white button, like in address card 白色圆角矩形，类似偏好设置表格单元或者地址簿卡片
UIButtonTypeDetailDisclosure,//蓝色的披露按钮，可放在任何文字旁
UIButtonTypeInfoLight,//微件(widget)使用的小圆圈信息按钮，可以放在任何文字旁
UIButtonTypeInfoDark,//白色背景下使用的深色圆圈信息按钮
```

```
UIButtonTypeContactAdd,//蓝色加号(+)按钮，可以放在任何文字旁
} UIButtonType;

- (void)setTitle:(NSString *)title forState:(UIControlState)state;
//设置按钮的文字

- (void)setTitleColor:(UIColor *)color forState:(UIControlState)state;
//设置按钮的文字颜色

- (void)setImage:(UIImage *)image forState:(UIControlState)state;
//设置按钮内部的小图片

- (void)setBackgroundImage:(UIImage *)image forState (UIControlState)state;
//设置按钮的背景图片

- (NSString *)titleForState:(UIControlState)state;
//获得按钮的文字

- (UIColor *)titleColorForState:(UIControlState)state;
//获得按钮的文字颜色

- (UIImage *)imageForState:(UIControlState)state;
//获得按钮内部的小图片

- (UIImage *)backgroundImageForState:(UIControlState)state;
//获得按钮的背景图片

UIButton *btn = [UIButton buttonWithType:UIButtonTypeRoundedRect];

// 监听按钮的点击
[button addTarget: self
action: @selector(clickButton:)
forControlEvents: UIControlEventTouchUpInside];
- (IBAction) clickButton: (UIButton *)button{
button.enabled = NO;
```

```
}
```

NSMutableAttributedString

https://blog.csdn.net/qq_34195670/article/details/52586460

强制刷新视图

https://www.jianshu.com/p/a84f85729952

UILabel

```objc
// 1 创建UILabel对象
UILabel *label = [[UILabel alloc] init];
// 2 设置frame
label.frame = CGRectMake(0,0,1,1);
// 3 设置背景颜色
label.backgroundColor = [UIColor redColor];
// 4 设置文字
label.text = @"hehe";
// 5 居中
label.textAlignment = NSTextAlignmentCenter;
// 6 设置字体大小
label.font = [UIFont systemFontOfSize:20.f];
label.font = [UIFont boldSystemFontOfSize:25.f];
label.font = [UIFont italicSystemFontOfSize:20.f];
// 7 设置文字颜色
label.textColor = [UIColor whiteColor];
// 8 设置阴影
label.shadowColor = [UIColor blackColor];
label.shadowOffset= CGSizeMake(-2,1);
// 9 设置行数（0 自动换行）
label.numberOfLines = 1;
// 10 显示模式
label.lineBreakMode =  NSLineBreakByTruncatingHead;
/*
```

```
NSLineBreakByWordWrapping = 0, 单词包裹,换行的时候会以一个单词换行
NSLineBreakByCharWrapping,     字符包裹换行,换行的时候会以一个字符换行
NSLineBreakByClipping,        裁剪超出的内容
NSLineBreakByTruncatingHead,     一行中头部省略(注意:numberOfLines要为1): "...wxyz"
NSLineBreakByTruncatingTail,     一行中尾部省略: "abcd..."
NSLineBreakByTruncatingMiddle  一行中中间部省略:  "ab...yz"
*/

// 11 添加到控制器的view中
[self.view addSubview: label];
```

sizeToFit 的简单使用

https://www.jianshu.com/p/5503d4e6a7bd
在分类页headerCell使用了，如果多行，第一次sizeToFit会计算width，除非提前设置width，这也是为什么会在那里出错的原因。

UIImageView

```
// 1 创建对象
UIImageView *imageView = [[UIImageView alloc] init];
// 4 设置图片
imageView.image = [UIImage imageNamed: @"1"];
// 5 设置图片的内容模式
imageView.contentMode = UIViewContentModeScaleAspectFill;
/*
重新绘制 (核心绘图) drawRact
UIViewContentModeRedraw,

带有Scale,标明图片有可能被拉伸或压缩
UIViewContentModeScaleToFill,     //完全的压缩或拉伸

Aspect 比例,缩放是带有比例的
UIViewContentModeScaleAspectFit,  //宽高比不变 Fit 适应
UIViewContentModeScaleAspectFill, //宽高比不变 Fill 填充

不带有Scale,标明图片不可能被拉伸或压缩
```

```
UIViewContentModeCenter,
UIViewContentModeTop,
UIViewContentModeBottom,
UIViewContentModeLeft,
UIViewContentModeRight,
UIViewContentModeTopLeft,
UIViewContentModeTopRight,
UIViewContentModeBottomLeft,
UIViewContentModeBottomRight,
*/

// 7 裁剪多余部分
imageView.clipsToBounds = YES;

// 用颜色填充图片
+(UIImage*)imageFromColor:(UIColor*)color size:(CGSize)size{
    CGRect rect=CGRectMake(0.0f, 0.0f, size.width,size.height);
    UIGraphicsBeginImageContext(size);//创建图片
    CGContextRef context = UIGraphicsGetCurrentContext();//创建图片上下文
    CGContextSetFillColorWithColor(context, [color CGColor]);//设置当前填充颜色的图形上下文
    CGContextFillRect(context, rect);//填充颜色

    UIImage *theImage = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    return theImage;
}

```

懒加载

用到的时候再加载，全局只会被加载一次，全局都可以使用
重写成员变量的getter方法，判断为空就加载，否则直接返回数据
不需要写到viewDidLoad中，增强代码可读性，降低耦合性

```
@interface UserViewController()
```

```
@property (nonatomic,strong) NSMutableArray *users;
@end

@implementation UserViewController
- (void) viewDidLoad{
[super viewDidLoad];
_users = [NSMutableArray new];
}
//以下是懒加载
- (void) viewDidLoad{
[super viewDidLoad];
}
- (NSArray *)usersdata{
if(!_users){
_users = [NSMutableArray new];
}
return _users;
}

// 编译器自动生成getter和setter方法，以及以下划线开头的实力变量，不需要自己手动再去写实例变量

@property (nonatomic,copy) (NSString *) str;
// 1. 生成_str成员变量的getter和setter方法声明;
// 2. 实现
// 3. 生成一个private的_str的变量
```

plist

property list，属性列表文件，用来存储串行化之后的对象的文件，文件是xml格式的，可用于存储用户设置，也可以存储捆绑的消息。

读取plist的信息

```
NSString *plistPath = [[NSBundle mainBundle] pathForResource:@"test" ofType:@"plist"]; // 通过文件名 和 扩展名获取文件路径
```

```
NSMutableDictionary *data = [[NSMutableDictionary alloc] initWithContentsOfFile:plistPath];// 读取路径文件中的信息
```

往plist写入信息

```
//数组
NSArray *names = @[@“aaa”, @“bbb”, @“ccc”, @“ddd”];
BOOL flag = [names writeToFile:@"/Users/xy/Desktop/names.plist" atomically:YES];
//字典
NSDictionary *persons = @{
@"name" : @“aaa”,
@"age" : @18,
@"height" : @1.88
};
BOOL flag = [persons writeToFile:@"/Users/xy/Desktop/person.plist" atomically:YES];
//数组和字典混合
NSArray *persons = @[
@{@"name" : @“aaa”, @"age":@38},
@{@"name" : @“bbb”, @"age":@25, @“cf”:@[@“xxx”, @“xy”]}
];
BOOL flag = [persons writeToFile:@"/Users/xy/Desktop/persons.plist" atomically:YES];

NSMutableDictionary *data = [[NSMutableDictionary alloc] init];// 读取路径文件中的信息
[data setObject:@"value_a" forKey:@"key_a"];
[data setObject:@"value_b" forKey:@"key_b"];
[data setObject:@"value_c" forKey:@"key_c"];
// 获取应用程序沙盒的Documents目录
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);
NSString *plistPath1 = [paths objectAtIndex:0];
// 得到完整的文件名
NSString *filename = [plistPath1 stringByAppendingPathComponent:@"test123456.plist"];
[data writeToFile:filename atomically:YES];
```

```
// 读取文件中的信息就可以判断是否文件操作成功
NSMutableDictionary *data1 = [[NSMutableDictionary alloc]
initWithContentsOfFile:filename];
NSLog(@"%@", data1);
```

修改plist的信息

```
// 获取应用程序沙盒的Documents目录
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
NSString *plistPath1 = [paths objectAtIndex:0];
// 得到完整的文件名
NSString *filename = [plistPath1
stringByAppendingPathComponent:@"test123456.plist"];
NSMutableDictionary *data = [[NSMutableDictionaryalloc]
initWithContentsOfFile:filename];// 读取路径文件中的信息
//设置属性值,没有的数据就新建，已有的数据就修改
[[data objectForKey:@"users"] setObject:@"逗比"forKey:@"name"];
[data removeObjectForKey:@"key_c"]; // 删除数据
[data setObject:@"ccccccccccc"forKey:@"key_c"];// 测试时无法直接修改已存在 key 的值，
可以先删除再添加方式来修改
[data writeToFile:filename atomically:YES];
// 读取文件中的信息就可以判断是否文件操作成功
NSMutableDictionary *data1 = [[NSMutableDictionaryalloc]
initWithContentsOfFile:filename];
NSLog(@"%@", data1);

```

MVC

Model View Controller，模型-视图-控制器，是一种软件设计典范。
将业务逻辑、数据和界面显示 分离出来的方法 来组织代码，将业务逻辑聚集到一个部件里，在需要改进和个性化定制界面以及用户交互的时候，不需要重写业务逻辑，提高了效率。

View的封装

如果一个view内部子控件较多，考虑自定义一个view，屏蔽内部子控件，外界可以传模型数据给view
基本步骤：

1.在initWithFrame方法中添加子控件，提供便利构造方法；
2.在layoutSubviews方法中设置子控件的frame（一定要调用super的layoutSubviews）
3.增加模型属性，在模型属性set方法中设置数据到子控件上

自定义控件

UIView表示屏幕上的一个矩形区域，负责渲染区域内的内容，响应触摸事件。
UIView内部有一个CALayer，提供内容绘制和显示，UIView的frame实际返回CALayer的frame。
创建一个继承自UIView的类
Class: CircleImageView
Subclass: UIView
实现initWithFrame，设置自定义控件的属性，并创建、添加子视图

```
-(instancetype)initWithFrame:(CGRect)frame {
self = [super initWithFrame:frame];
if (self) {
_imageView = [[UIImageView alloc] initWithFrame:CGRectMake(0, 0, frame.size.width, frame.size.height)];
_imageView.contentMode = UIViewContentModeScaleAspectFill;
_imageView.layer.masksToBounds = YES;
_imageView.layer.cornerRadius = frame.size.width/2;
[self addSubview:_imageView];

}
return self;
}
```

如果需要对子视图重新布局，需要调用layoutSubViews方法

```
-(void)layoutSubviews {
[super layoutSubviews];
_imageView.frame = self.frame;
_imageView.layer.cornerRadius = self.frame.size.width/2;
}
```

layoutSubviews在以下情况被调用
1、init初始化不会触发layoutSubviews
2、addSubview会触发layoutSubviews
3、设置view的Frame会触发layoutSubviews，当然前提是frame的值设置前后发生了变化
4、滚动一个UIScrollView会触发layoutSubviews

5、旋转Screen会触发父UIView上的layoutSubviews事件
6、改变一个UIView大小的时候也会触发父UIView上的layoutSubviews事件

这个自定义控件提供对外接口方法，为自定义的控件赋值

```
(void)configeWithImage:(UIImage *)image {
_imageView.image = image;
}

添加自定义控件到界面上
_circleImageView = [[CircleImageView alloc] initWithFrame:CGRectMake(0, 80, 150, 150)];
[_circleImageView configeWithImage:[UIImage imageNamed:@"tree"]];
[self.view addSubview:_circleImageView];
```

自动布局autoresizing

https://www.cnblogs.com/GarveyCalvin/p/4165151.html

UIScrollView

https://blog.csdn.net/jymn_chen/article/details/14456197 基本样式
https://my.oschina.net/wangdk/blog/162945 代理

UITableView

https://juejin.im/post/58a6a41e2f301e006d8eea73 这篇文章把基础操作讲的很清晰
有两种样式：普通和分组
只有一列。每一行都是一个UITableViewCell
UITableViewCell中含有一个容器contentView，一个内容textLabel和一个详情detailTextLabel，以及一个图片imageView
四种风格
table head view 顶部视图
table foot view 底部视图
Section 一般为1
Section head
Section foot

UITableView继承自UIScrollView,且有两个协议，UITableViewDelegate委托协议和UITableViewDataSource数据源协议

UITableViewDataSource

```
@protocol UITableViewDataSource<NSObject>// 这个是必须实现的两个方法
@required
// 这个tableVIew中有每一组有多少行的数据      * 如果没有分组则默认为1 组
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section;
//  返回值为UITableViewCell 在这个方法中确定每一行tableView中所显示的数据     *每一行叫做一个 Cell
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath;
// 可以选择实现的方法
@optional
// 每一个tableView中有多少组数据   Default is 1 if not implemented 默认为1
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView;               // Default is 1 if not implemented
// 设置tableView  分组的头部的文字  Header的文字
- (nullable NSString *)tableView:(UITableView *)tableView titleForHeaderInSection:(NSInteger)section;
// 设置tableView  分组的尾部的文字  Fotter的文字
- (nullable NSString *)tableView:(UITableView *)tableView titleForFooterInSection:(NSInteger)section;
// Editing//  是否可以编辑     默认为全部可编辑
- (BOOL)tableView:(UITableView *)tableView canEditRowAtIndexPath:(NSIndexPath *)indexPath;
// Moving/reordering
// 是否可以移动    进行重新排序
- (BOOL)tableView:(UITableView *)tableView canMoveRowAtIndexPath:(NSIndexPath *)indexPath;
//  返回每组标题索引
- (nullable NSArray<NSString *> *)sectionIndexTitlesForTableView:(UITableView *)tableView __TVOS_PROHIBITED;
@end

```

对于tableView: cellForRowAtIndexPath:
一般在这里创建cell对象，这里可以使用dequeueReusableCellWithIdentifier来从对象池中获取UITableViewCell对象，缓冲池里存放不显示的cell对象

需要对cell定义一个可重用的标志符。dequeueReusableCellWithIdentifier会根据重用标志符去缓存池里取出对象，如果返回nil说明没找到，这时候再创建cell对象。
在这里存在cell重用的视图问题（比如小说历史往下拉，会发现cell里面的信息是最开始几行的信息，也就是使用了可重用的cell，但是信息没有更新）
使用cell重用的时候，存在详情无法显示的问题 cell style 问题

```
static NSString *ID = @"cellID";
UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:ID];
if(cell == nil){
cell = [[UITableViewCell alloc] initWithStyle: UITableViewCellStyleSubtitle
reuseIdentifier: ID];
}
```

dequeueReusableCellWithIdentifier: 和 dequeueReusableCellWithIdentifier: forIndexPath:的区别，在于前者不必向tableView注册cell的Identifier，但需要判断获取的cell是否为nil；而后者则必须向table注册cell，可省略判断获取的cell是否为空，因为无可复用cell时runtime将使用注册时提供的资源去新建一个cell并返回，这个新建的cell的style就导致了detail不能显示。

UITableViewDelegate

用哪个ViewController给代理赋值，就在哪个VC里给对应的delegate实现接口。

```
tableView.delegate = self;
```

主要用来设置section的头尾，响应事件

一些操作

点击cell作出反应

设置tableView的属性为可选的

```
tableView.allowsSelection = YES;
tableView.selectionStyle = UITableViewCellSelectionStyleBlue; // 设置颜色
```

添加响应事件，实现didSelectRowAtIndexPath函数

```
-(void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath
*)indexPath{
NSLog(@"%d",(int)indexPath.row);
}
```

索引条

```
//设置索引条的文字颜色
```

```
self.tableView.sectionIndexColor = [UIColor redColor];
// 设置索引条的背景颜色
self.tableView.sectionIndexBackgroundColor = [UIColor yellowColor];
```

确定tableView的展示行数

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section{
if(hasClickCleanButton)return 0;
return [bookMessages getMessageSize];// 根据容器读了多少内容来确定加载多少行，section数量也是同理
}
```

UICollectionView

UICollectionView 常用的方法
另一篇博客
间距相关
使用上可能遇到的问题

与UITableView不同的是，还需要创建布局参数，UICollectionViewFlowLayout。

- 需要实现三种类型的委托（UICollectionViewDelegateFlowLayout实际上是UICollectionViewDelegate的一个子协议）

```
@interface ViewController : UIViewController <UICollectionViewDelegateFlowLayout, UICollectionViewDataSource>
```

- 初始化

```
    /**
     创建layout
     */
UICollectionViewFlowLayout *layout = [[UICollectionViewFlowLayout alloc] init];
    /**
     创建collectionView
     */
    UICollectionView* collectionView = [[UICollectionView alloc]initWithFrame:CGRectMake(0, 64, kScreenWidth, kScreenHeight-64) collectionViewLayout:layout];
    collectionView.delegate = self;
    collectionView.dataSource = self;
```

```
    collectionView.backgroundColor = [UIColor cyanColor];
    /*
     注册item和区头视图、区尾视图
     */
    [collectionView registerClass:[MyCollectionViewCell class] forCellWithReuseIdentifier:@"MyCollectionViewCell"];
    [collectionView registerClass:[UICollectionReusableView class] forSupplementaryViewOfKind:UICollectionElementKindSectionHeader withReuseIdentifier:@"MyCollectionViewHeaderView"];
    [collectionView registerClass:[UICollectionReusableView class] forSupplementaryViewOfKind:UICollectionElementKindSectionFooter withReuseIdentifier:@"MyCollectionViewFooterView"];
    [self.view addSubview:collectionView];

```

- 首先是分区数量（如果没实现，默认为1）

```
- (NSInteger)numberOfSectionsInCollectionView:(UICollectionView *)collectionView{
    return 2;
}
```

- 指定分区包含的数据源条目数（number of items）

```
- (NSInteger)collectionView:(UICollectionView *)collectionView numberOfItemsInSection:(NSInteger)section{
    return 7;
}
```

- 某个indexPath对应的cell，必须实现

```
- (UICollectionViewCell *)collectionView:(UICollectionView *)collectionView cellForItemAtIndexPath:(NSIndexPath *)indexPath
```

cell的结构由下至上：首先cell本身是个容器view，然后是一个大小自动适应整个cell的backgroundView，作为cell平时的背景，再然后是一个selectedBackgroundView，是cell被选中时的背景，最后是一个contentView，自定义内容应被加在这个view上。

```
- (UICollectionReusableView *)collectionView:(UICollectionView *)collectionView viewForSupplementaryElementOfKind:(NSString *)kind atIndexPath:(NSIndexPath *)indexPath
//为collection view添加一个补充视图(页眉或页脚)
```

```
3
4  /**
5   创建区头视图和区尾视图
6   */- (UICollectionReusableView *)collectionView:(UICollectionView *)collectionView viewForSupplementaryElementOfKind:(NSString *)kind atIndexPath:(NSIndexPath *)indexPath{
7      if (kind == UICollectionElementKindSectionHeader){
8          UICollectionReusableView *headerView = [collectionView dequeueReusableSupplementaryViewOfKind:kind withReuseIdentifier:@"MyCollectionViewHeaderView" forIndexPath:indexPath];
9          headerView.backgroundColor = [UIColor yellowColor];
10         UILabel *titleLabel = [[UILabel alloc]initWithFrame:headerView.bounds];
11         titleLabel.text = [NSString stringWithFormat:@"第%ld个分区的区头",indexPath.section];
12         [headerView addSubview:titleLabel];
13         return headerView;
14     }else if(kind == UICollectionElementKindSectionFooter){
15         UICollectionReusableView *footerView = [collectionView dequeueReusableSupplementaryViewOfKind:kind withReuseIdentifier:@"MyCollectionViewFooterView" forIndexPath:indexPath];
16         footerView.backgroundColor = [UIColor blueColor];
17         UILabel *titleLabel = [[UILabel alloc]initWithFrame:footerView.bounds];
18         titleLabel.text = [NSString stringWithFormat:@"第%ld个分区的区尾",indexPath.section];
19         [footerView addSubview:titleLabel];
20         return footerView;
21     }
22     return nil;
23 }
24
25
26 - (CGSize)collectionView:(UICollectionView *)collectionView layout:(UICollectionViewLayout*)collectionViewLayout referenceSizeForHeaderInSection:(NSInteger)section
27 //设定页眉的尺寸
28
```

```
29 - (CGSize)collectionView:(UICollectionView *)collectionView layout:(UICollectionViewLayout*)collectionViewLayout referenceSizeForFooterInSection:(NSInteger)section
30 //设定页脚的尺寸
31
32 - (void)registerClass:(Class)viewClass forSupplementaryViewOfKind:(NSString *)elementKind withReuseIdentifier:(NSString *)identifier
33 //添加页眉和页脚以前需要注册类和标识
```

UICollectionViewFlowLayout

https://blog.csdn.net/Mr_XiaoJie/article/details/73195241

```
1 //同一组当中，垂直方向：行与行之间的间距；水平方向：列与列之间的间距
2 @property (nonatomic) CGFloat minimumLineSpacing;
3 //垂直方向：同一行中的cell之间的间距；水平方向：同一列中，cell与cell之间的间距
4 @property (nonatomic) CGFloat minimumInteritemSpacing;
5 //每个cell统一尺寸
6 @property (nonatomic) CGSize itemSize;
7 //滑动反向，默认滑动方向是垂直方向滑动
8 @property (nonatomic) UICollectionViewScrollDirection scrollDirection;
9 //每一组头视图的尺寸。如果是垂直方向滑动，则只有高起作用；如果是水平方向滑动，则只有宽起作用。
10 @property (nonatomic) CGSize headerReferenceSize;
11 //每一组尾部视图的尺寸。如果是垂直方向滑动，则只有高起作用；如果是水平方向滑动，则只有宽起作用。
12 @property (nonatomic) CGSize footerReferenceSize;
13 //每一组的内容缩进
14 @property (nonatomic) UIEdgeInsets sectionInset;
```

如何返回一个不确定的cell子类

```
1
2 - (nonnull __kindof UICollectionViewCell *)collectionView:(nonnull UICollectionView *)collectionView cellForItemAtIndexPath:(nonnull NSIndexPath *)indexPath {
3     SSBook *book = [self.books ss_safeObjectAtIndex:indexPath.row];
4     UICollectionViewCell *cell = nil;
5     if (indexPath.row == 0) {
```

```
        SSBookNewStyle1Cell *style1Cell = [self.collectionView dequeueReusableCellWithReuseIdentifier:NSStringFromClass([SSBookNewStyle1Cell class]) forIndexPath:indexPath];
        style1Cell.book = book;
        style1Cell.isEnableTapAlphaEffect = true;
        cell = style1Cell;
    } else {
        SSBookNewStyle2Cell *style2Cell = [self.collectionView dequeueReusableCellWithReuseIdentifier:NSStringFromClass([SSBookNewStyle2Cell class]) forIndexPath:indexPath];
        style2Cell.book = book;
        style2Cell.isEnableTapAlphaEffect = true;
        cell = style2Cell;
    }

    if (![self.showModels containsObject:book]) {
        [GET_SERVICE(SSReporter) report_event_store_operation_detail:self.model book:book rank:(int)(indexPath.row + 1) event:event_show];
        if ([self.delegate respondsToSelector:@selector(bookDidShow:bookData:bookRank:)]) {
            [self.delegate bookDidShow:self bookData:book bookRank:indexPath.row];
        }

        [self.showModels addObject:book];

    }
    return cell;
}
```

CollectionViewCell的对齐

https://www.jianshu.com/p/ac3edf92c5fd

提示框UIAlertController UIAlertAction

https://www.cnblogs.com/XYQ-208910/p/4889611.html 详细

弹出提示信息，并给出一些按钮来处理相应的要求。

```
typedef NS_ENUM(NSInteger, UIAlertControllerStyle) {
```

```
UIAlertControllerStyleActionSheet = 0,  //在视图底部弹出的提示框,它不能添加文本框,而且在ipad中必须使用popover形式展示
UIAlertControllerStyleAlert                    //在视图中间弹出的提示框
} NS_ENUM_AVAILABLE_IOS(8_0);

typedef NS_ENUM(NSInteger, UIAlertActionStyle) {
UIAlertActionStyleDefault = 0,    //默认的确认按钮
UIAlertActionStyleCancel,          //默认的取消按钮
UIAlertActionStyleDestructive    //默认的红色按钮
}NS_ENUM_AVAILABLE_IOS(8_0);

// UIAlertController创建方法
+ (instancetype)alertControllerWithTitle:(NSString *)title message:(NSString *)message preferredStyle:(UIAlertControllerStyle)preferredStyle;

// 在提示框添加按钮
- (void) addAction: (UIAlertAction *)action;

// UIAlertAction创建方法
+ (instancetype)actionWithTitle:(NSString *)title style:(UIAlertActionStyle)style handler:(void (^)(UIAlertAction *action))handler;

// 添加实例
[self presentViewController:alertview animated:YES completion:nil];
```

给UIView添加边框

https://www.jianshu.com/p/2b202f15ad02
https://www.jianshu.com/p/317ec018bb32
任意大小任意圆角： https://github.com/MrGCY/AnyCornerRadius

NSAttributedString的使用

https://www.jianshu.com/p/3f85f91d1208

行高，行间距

https://www.jianshu.com/p/a9b6c96882e7

关于视图层级

https://blog.csdn.net/xiaojinIT/article/details/56842894

UINavigationController 导航控制器

管理视图控制器，以栈的形式管理视图控制器，push和pop方法弹入弹出控制器，只能显示处于栈顶的视图控制器

```
// 用push的方法将某个控制器入栈
- (void)pushViewController:(UIViewController *)viewController animated:(BOOL)animated;

// 用setViewControllers一次压入多个控制器，会显示最后的控制器
UINavigationController *nav = [[UINavigationController alloc] init];
window.rootViewController = nav;
// 创建3个测试控制器
UIViewController *vc1 = [[UIViewController alloc] init]; vc1.view.backgroundColor = [UIColor blueColor]; UIViewController *vc2 = [[UIViewController alloc] init]; vc2.view.backgroundColor = [UIColor redColor]; UIViewController *vc3 = [[UIViewController alloc] init]; vc3.view.backgroundColor = [UIColor greenColor];
// 最终会显示vc3
[nav setViewControllers:@[vc1,vc2,vc3] animated:YES];

// 使用pop方法可以移除栈顶控制器
- (UIViewController *)popViewControllerAnimated:(BOOL)animated;

// 返回根控制器
-(NSArray *)popToRootViewControllerAnimated:(BOOL)animated;

// 获取被管理的控制器
/// 当前管理的所有的控制器
@property(nonatomic,copy) NSArray<__kindof UIViewController *> *viewControllers;
 /// 栈顶控制器
@property(nullable, nonatomic,readonly,strong) UIViewController *topViewController;
/// 当前可见的VC，可能是topViewController，也可能是当前topViewController present(modal)出来的VC，总而言之就是可见的VC
```

```
@property(nullable, nonatomic,readonly,strong) UIViewController *visibleViewController;
```

导航条

导航条的内容由控制器的navigationItem属性决定
使用self.navigationItem.对应属性
使用self.navigationController，再得到想要设置的viewController

```
// 中间的标题文字
@property(nullable, nonatomic,copy) NSString *title;
// 中间标题视图
@property(nullable, nonatomic,strong) UIView *titleView;
// 导航栏附加说明
@property(nullable,nonatomic,copy)   NSString *prompt;
// 左上角返回按钮
@property(nullable, nonatomic,strong) UIBarButtonItem *leftBarButtonItem;
// 子视图后退按钮
@property(nullable,nonatomic,strong) UIBarButtonItem *backBarButtonItem;
// 右上角的按钮/多个按钮
@property(nullable, nonatomic,strong) UIBarButtonItem *rightBarButtonItem;
/// 一次设置多个按钮
@property(nullable,nonatomic,copy) NSArray<UIBarButtonItem *> *rightBarButtonItems;
```

UIToolBar

UINavigationController自带工具栏，通过self.navigationController setToolbarHidden:NO 来显示工具栏，工具栏内容可以通过VC的toolbarItems设置。
这种使用较少，大部分情况下用UITabBarController

一些使用

```
// 对当前视图绑定navigationController，在Appdelegate里实现
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
// Override point for customization after application launch.
self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen].bounds];
ViewController *viewController = [[ViewController alloc] init];
```

```
UINavigationController *navigationController = [[UINavigationController alloc] initWithRootViewController: viewController];
[[UINavigationBar appearance] setBarTintColor:[UIColor whiteColor]];
self.window.rootViewController = navigationController;
[self.window makeKeyAndVisible];
return YES;
}
/// 需要注意UINavigationBar是只读的，要设置颜色可以用
[[UINavigationBar appearance] setBarTintColor:[UIColor whiteColor]];

// https://code.byted.org/qujing.kukyo/qujingdemo01
```

关于DEMO练习

需要实现的功能

小说阅读历史详情页，数据来自books.json

viewDidLoad

设置左右UIBarButtonItem，并且可以绑定方法

```
UIBarButtonItem *clearButton = [[UIBarButtonItem alloc] initWithTitle:@"清空" style:UIBarButtonItemStylePlain target:self action:@selector(clearAllItems:)];
self.navigationItem.rightBarButtonItem = clearButton;
```

loadView

第一次loadView的时候加载JSON，然后就能在后面显示出历史信息，点击“清空”后会重新loadView，根据标记参数跳过加载JSON的操作

点击跳转

demo原本的实现应该是用了segue，奈何不会改啊.....

```
-(void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
if(hasInit)return;
_detailViewController = [[DetailViewController alloc] init];
_detailViewController.detailDescriptionLabel = [[UILabel alloc] init];
```

```
[_detailViewController.detailDescriptionLabel setText:detailAbstract[indexPath.row]];
NSLog(@"%@ %@",detailAbstract[indexPath.row], _detailViewController.detailDescriptionLabel.text);
[self.navigationController pushViewController:_detailViewController animated:NO];
}
```

JSON解析

```
// 存储json解析出来的书名，详情，以及图片地址
NSMutableArray *strBookName, *strOriginTitle, *ImageUrl, *detailAbstract;
BOOL hasInit = NO;
// 解析json
- (void) initWithJsonData{
NSString *path = [[NSBundle mainBundle] pathForResource: @"books.json" ofType: nil];
if(path == nil){
NSLog(@"json path not found");return;
}
else{
NSLog(@"path correct");
}

NSData *data = [NSData dataWithContentsOfFile:path];
NSMutableDictionary *dicArray = [NSJSONSerialization JSONObjectWithData:data options:NSJSONReadingAllowFragments error:nil];
strBookName = [[NSMutableArray alloc] init];
strOriginTitle = [[NSMutableArray alloc] init];
ImageUrl = [[NSMutableArray alloc] init];
detailAbstract = [[NSMutableArray alloc]init];
for(id itmes in [dicArray objectForKey:@"data"]){
NSString *tmp = [itmes objectForKey:@"book_name"];
[strBookName addObject:tmp];
tmp = [itmes objectForKey:@"origin_chapter_title"];
[strOriginTitle addObject:tmp];
tmp = [itmes objectForKey:@"thumb_url"];
[ImageUrl addObject:tmp];
```

```
tmp = [itmes objectForKey:@"abstract"];
[detailAbstract addObject:tmp];
}
}
- (UIImage *)getImageFromUrl: (long) index{
UIImage *res;
NSData *data = [NSData dataWithContentsOfURL:[NSURL
URLWithString:ImageUrl[index]]];
res = [UIImage imageWithData:data];
return res;
}
```

加载Cell

懒加载的两种方式，为了避免被坑还是用了需要检查nil的版本
每次加载的时候都设置Cell信息为对应位置的信息

动画基础

https://www.jianshu.com/p/8fa9f89f8854
这篇排版稍好

transform: CGAffineTransformIdentity

https://www.jianshu.com/p/8a079f61e807

添加圆角

```
- (UILabel *)hotLabel {
    if (nil == _hotLabel) {
        _hotLabel = [[UILabel alloc] init];
        _hotLabel.text = @"热门";
        _hotLabel.font = [UIFont PingFangSCMediumFontWithSize:9];
        _hotLabel.textColor = [UIColor whiteColor];
        _hotLabel.backgroundColor = [UIColor colorWithHexString:@"#FA6725"];
        _hotLabel.textAlignment = NSTextAlignmentCenter;
        //_hotLabel.layer.cornerRadius = 4;
        self.hotLabel.size = CGSizeMake(26, 16); // 固定size 放在这里便于添加部分圆角
```

```
        UIRectCorner corner = UIRectCornerTopRight|UIRectCornerBottomLeft;
        UIBezierPath * path = [UIBezierPath bezierPathWithRoundedRect:self.hotLabel.bounds byRoundingCorners:corner cornerRadii:CGSizeMake(4, 4)];
        CAShapeLayer * maskLayer = [[CAShapeLayer alloc] init];
        maskLayer.frame = self.hotLabel.bounds;
        maskLayer.path  = path.CGPath;
        _hotLabel.layer.mask = maskLayer;
    }
    return _hotLabel;
}
```

automaticallyAdjustsScrollViewInsets

https://www.jianshu.com/p/9884f13074b8

点击事件

手势识别器-UIGestureRecognizer

```
UITapGestureRecognizer *tap = [[UITapGestureRecognizer alloc] initWithTarget:self action:@selector(onTapView)];
    tap.cancelsTouchesInView = NO;
    [self.view addGestureRecognizer:tap];
```

Frame 和 bounds的区别

frame and bounds

UITextField

http://www.beyondabel.com/blog/2014/01/10/uitextfield/
another

去除collectionView reload的隐式动画

https://blog.csdn.net/chenyong05314/article/details/86590808