《Effective Objective－c 2.0》

官方文档

斯坦福 网易公开课

www.raywenderlich.com

cocoachina.com code4app.com

www.stactoverflow.com提问题社区

PaintCode绘图工具

www.sqlite.org

官方软件：WWDC——Videos what’s new in Xcode/Cocoa Touch/Swift

分类：对不知道源代码的类的扩展，只能扩展方法，不能扩展成员。作用：不借助继承实现对类的扩展；实现非正式协议。可以访问目标类中成员。使用：导入分类头文件。

分类中不可以创建实例变量和属性。

分类可以访问主类的属性，但不可以访问主类的实例变量。

扩展中可以声明实例变量和属性；扩展通常定义在文件的.m中，不能分开；扩展是用来声明私有的属性和方法。文件名通常为：”主类类名\_扩展标识.h”

strong当一个对象被引用指向时，此对象会隐式地retain一次，当强引用超出作用域时，指向的对象会隐式地发送release消息一次。

weak仅仅就是指向对象，但不会隐式地发送retain消息，出了作用域也不会发送release消息。当一个弱引用指向的对象被销毁时，弱引用本身会自动赋值为空。

unsafe\_unretained等同于assgin，功能和\_weak几乎一样。

\_autoreleaseing将返回值的对象放入自动释放池。

将MRC的代码转换成ARC的代码（手动）：retain=>strong；删掉release、autorelease、[super dealloc]。

菜单栏(Edit)->Refacotor(重构)->Convert to Objective-C ARC

在ARC项目中继续使用MRC编译的类，在编译选项中标识MRC文件即可”-fno-objc-arc”；

-MRC项目中继续使用ARC编译的类，在编译选项中标识ARC文件即可”-fobjc-arc”

声明式属性：编译器语法（自动生成set和get方法） .h中: @property int age（属性）; .m中（IOS6中这一步可省略）:@synthesize age（实例变量）=\_age（\_属性名）;

协议：@interface类名(分类类名):父类名<协议名>

使用协议类型的引用指向实现了协议或者遵守了协议的对象id<TRProtocol> p = [[MyClass]init];

协议的继承：@protocol TRTarena2<TRTarena>

一个类可以同时遵守多个协议：@interface TRStudent: NSObject<TRTarena,TRTarena3>

协议选择器：判断一个类是否遵守某个协议 Protocol \*pro = @protocol(NSCopying); BOOL isflag = [TRPhone conformsToProtocol:pro];

（不可变字符串）字面值内容相同的字符串只开辟一个内存空间，但按指定格式创建新的字符串对象，就算内容相同，地址也不一样。 可变字符串：NSMutableString 其他:NSmutableArray NSmutableSet NSmutableDictionary

增append…、删Delete… 创建字面值将退化成NSString

只能NSMutableString \*str = [[NSMutableString alloc]init];

[str2 insertString:<#(NSString \*)#> atIndex:<#(NSUInteger)#>]

[str2 appendString:<#(NSString \*)#>]

[str2 deleteCharactersInRange:<#(NSRange)#>]

[str2 replaceCharactersInRange:<#(NSRange)#> withString:<#(NSString \*)#>]

分割字符串：1.从开头到中间某个位置substringToIndex；2.从中间某个位置到结尾substringFromIndex；3.从中间某个位置开始，到长度是多少[str substringWithRange:(NSRange){2,3}]。初始化：initWithFormat；追加：stringByAppendingFormat: 判断字符串相等：isEqualToString。

修饰符：

@public可以在任意位置访问

@package可以在包内部访问，一个项目一定是在同一个包下。

@protected可以在本类内部和子类内部访问（默认实例变量修饰符）

@privated只可以在本类的内部分类访问

并不是所有对象都有copy方法，如果一个对象支持copy功能，首先要遵守协议NSCopying协议，必须实现copyWithZone方法。如果不但想复制对象，而且还要复制对象的值，一般还要重写有参的初始化方法。

比较“类”信息的时候，需要使用类对象，判断一个类是否是另一个类的子类：[TRPerson isSubclassOfClass:[TRStudent class]];

判断一个引用指向的对象是否是某种类型或子类型：

-(BOOL)isKindOfClass:(Class)c;

判断一个引用(实例)指向的对象是否是某种类型：

-(BOOL)isMemberOfClass:(Class)c;

方法选择器：和C语言中的函数指针相类似

SEL 就是方法的类型 sel是方法变量；

SEL sel = @selector(study);

某个类是否存在某个方法(消息)：

BOOL b = [TRStudent instancesRespondToSelector:@selector(study)];

可以向对象发送任何消息，而不需要在编译的时候声明这些消息：

[stu2 performSelector:sel2];

协议选择器：

协议的引用指向一个协议

Protocol\* p = @protocol(NSCopying);

可以判断一个类是否遵守了某个协议

BOOL b = [TRStudent conformsToProtocol:p];

封装，OC中基本数据类型可能无法用

int i = 10; NSNumber \*num = [NSNumber numberWithint:i];

拆封

int i2 = [num intValue];

封装，将栈放入堆

NSValue\* value = [NSValue valueWithBytes:首地址 objCType:@encode(类型)]

拆封

int p1;[value getValue:&p1];

NSDate：

NSDate \*date = [[NSDate alloc]init];得到世界标准时间

NSDate\* time=[NSDate dateWithTimeIntervalSinceNow:30.0f ] 延迟30 s

得到相差秒数NSTimeInterval\* interv=[time timeIntervalSinceNow];

NSTimeInterval\* interv=[time timeIntervalSinceDate:time1];

isEqualToDate LaterDate

时间格式

NSDateFormatter \*df = [[NSDateFormatter alloc]init];

df.dateFormat =@"yyyy-MM-dd hh:mm:ss";

NSLog(@"%@",[df stringFromDate:time]);

排序

SEL sel = @selector(compare:); array = [array sortedArrayUsingSelector:sel];

方法重写:-(NSComparisonResult)compare:(TRStudent\*)otherStudent

{if(self.age-otherStudent.age>0){ return NSOrderedDescending;}else if ((self.age-otherStudent.age)<0){ return NSOrderedAscending; }else{ return NSOrderedSame;} }

数组复制：

深拷贝（内容复制）：将对象生成副本initWithArray:对象名 copyItems:YES

浅拷贝（引用复制）：仅将对象的引用计数加1

发送消息mutableCopy可将不可变数组变成可变数组。

末尾添加：addObject

插入:insertObject:对象名 atIndex:(int)

修改：replaceObjectAtIndex:(int) withObject:对象

replaceObjectsInRange:r withObjectsFromArray:对象

删除：最后一个removeLastObject

指定对象removeObject:对象

指定位置removeObjectAtIndex:2

指定范围removeObjectsInRange:(NSRange)

指定对象s：removeObjectsInArray:(NSMutableArray\*)

清空数组removeAllObjects

遍历数组：

枚举的时候不可改变数组。枚举时纪录，枚举完后改变。

NSEumerator\* enumerator = [obj objectEnumerator];

While(obji = [obj nextobject]){};

集合类NSSet：无序、不允许重复。

重复：两个引用指向同一个对象。

若两个引用分别指向不同的对象，但对象的值相同，则不认为是重复。

NSSet \*set = [NSSet setWithObjects:对象1,对象2,nil];

复制：NSSet\* set1=[NSSet setWithSet:set];

由数组创建：setWithArray:(NSArray\*)

hash方法和equeal方法

hash码是用来唯一标识对象的。

当对象以方法方式（区别于赋值方式）放入NSSet集合中，会自动调用对象hash码进行判断，当hash码相同，会自动调用isEqual方法。需要时，可重写这两个方法。

<1>自反性（hash）

<2>类型

<3>值

集合运算：

交集:[obj1 intersectSet:obj2]; 并集:[obj1 unionSet:obj2];

从集合中删除另一指定集合的元素:[obj1 minusSet:obj2];

NSDictionary

key不能重复，value可以重复。

[NSDictionary dictionaryWithObectsAndKeys:obj1,@”1”,obj2,@”2”,nil]

[NSDictionary dictionaryWithObects:(NSArray\*) forKeys:(NSArray\*)];

复制:[NSDictionary dictionaryWithDictionary:(NSdictionary\*)];

获取建值对个数：[(NSdictionary\*) count];

所有的key: NSArray\* key = [dict allKeys];

通过obj获取key：NSArray\* key=[(NSdictionary\*) allKeysForObjects:objs];

通过key找到obj：[[dict2 objectForKey:@”1”] print]; dict2[@”1”];

获取所有的obj：NSArray\* value1=[dict allValues];

根据多个key找到多个value：

NSArray\* values=[dict objectsForKeys:[NSArray arrayWtihObjects:@”1”,@”2”,nil] notFoundMarker:@”no-found”];

创建优化：[NSDictionary\* dict]=@{@”1”:obj1,@”2”:obj2};

按key排序：NSArray\* sortedKey=[dict2 keysSortedByValueUsingSelector: @selecor(compare:)];

将字典内容写入文件：如不知读取应如何创建模型，可写通过写的方式创建模版。

BOOL a=[dict writeToFile:@”/Users/tarena/Desktop/dict.plist” atomically:YES];

由文件内容创建字典：

NSDictionary\* dict1=[NSDictionary ditionaryWithContentsOfFile:@”/Users /tarena/Desktop/dict.plist”];

NSMutableDictionary:

空值：NSMutableDictionary\* dict=[NSMutableDictionary dictionary];

预估值：NSMutableDictionary\* dict=[NSMutableDictionary dictionaryWtih Capacity:18];

标准创建方法：NSMutableDictionary\* dict=[NSMutableDictionary dictionaryWithObjectsAndKeys:stu1,@”1”,stu2,@”2”,nil];

添加一个元素：[dict setValue:stu2 forKey:@”3”];

添加一个字典:[mdict addEntriesFromDictionary:dict];

覆盖一个字典：[mdict setDictionary:dict];

删除键值对:一个[dict removeObjectForKey:@”1”];

多个[dict removeObjectsForKeys:(NSArray\*)];

清空字典[dict removeAllObjects];

block:

返回值类型(^block的名字)(类型1,类型2)=^返回值类型无返回值可省略(类型1 参数1,类型2 参数2){}；block的声明和定义可写在一起，=右边是定义。

调用int result = block的名字(参数1,参数2)；

声明一个Block类型的变量：

BlockType b = ^{};

Block可读值不可改值；要改则要先\_\_block修饰

Block作为参数（方法调用方法）、返回值类型、属性。

[stu sortedArrayUsingComparator:^NSComparisonResult(id obj1,id obj2) {TRStudent\* stu1=obj1; TRStudent\* stu2=obj2; return [stu1.name compare:stu2.name]}];

- (void)makeFool:(void (^)(NSString \*))callback;

SEL sel ＝ NSSelectorFromString(@”方法名”)；

[obj performSelector:sel];

%%%%%%%%%%%%%%%%%%%%%%

控制器：

step1: UIApplicationMain

• 根据第三个参数创建了应用程序对象，写nil就是创建了UIApplication类型的实例，这个对象负责监控应用程序的当前状态。

• 根据第四个参数创建了应用程序代理类对象。

• 启动了事件循环。

(void)applicationWillResignActive:(UIApplication \*)application释放激活。

Step2: 应用程序对象将当前状态交给代理类对象，由代理来处理，启动程序后，第一个状态就是didFinishLaunchingWithOption，于是，代理就来针对这个状态做响应，也就是程序启动后要做的事情写在这个方法中即可。

Step3: 执行代理的启动方法，创建改window，再创建文本，把文本加到window中，最后把window显示。

viewDidLoad创建视图（界面）时自动调用。其他如viewWillAppear类似。

Label.font = [UIFont systemFontOfSize:24];

Label.lineBreakMode=NSLineBreakByTruncatingMiddle;（…出现的位置）

Label.numberOfLines=0;最多显示的行数

Label.textColor=[UIColor redColor];

UIButton\* button = [UIButton buttonWithType:UIButtonTypeSystem];

[button setTitle:@"OK" forState:UIControlStateNormal];

[button setBackgroundImage:[UIImage imageNamed:@"icon180.png"] forState: UIControlStateNormal];设置button的背景图

[button addTarget:obj action:@selector(fun:) forControlEvents: UIControlEventTouchUpInside];

IBOutlet: 让该引用成为外部引用(添加一个输出口); 系统创建的一般用weak；

设置快捷键：系统偏好设置－键盘－快捷键

switch开关：on属性；setOn： animated：；

textFeild：Placehold:灰色提示；

触摸动作：hit-test view－坐标（UITouch对象）给window－给直接子视图发hit－test消息，把坐标传给直接子视图，坐标是否位于直接子视图－该子视图给自己的直接子视图发消息－…－触点不在范围，返回nil－在范围内若无响应，返回父视图响应－返回window响应－返回app响应。——－响应者链机制。textFeild是自动第一响应者（成为第一响应者becomeFirstResponder），所以要收起键盘，则让textFeild放弃第一响应者身份（resignFirstResponder）(方法一)。或结束文本框所在的父视图的编辑状态，即让该父视图放弃第一响应者身份，[self.view endEditing:YES];（方法二）。

关闭键盘时机：1.右下角事件：Did End On Exist；2. 点空白处-(void)touchesBegan:(NSSet \*)touches withEvent:(UIEvent \*)event

警告框：UIAlertView：1.创建 2.show方法

操作表：UIActionSheet

代理（委托）模式：委托方、代理方（被委托方）

• 遵守协议 2.实现方法 3.将自己设置为代理方

<委托方类名Delegate>

出现错误“.h has been modified”复制路径到“前往…”，再回到xcode，product, clear ,command+B。

#pragma mark – 代码书签

MVC模式

M－数据模型（存储数据）、业务模型（业务规则的运算）

M和V之间绝对不会有通信。V通过委托/代理发送请求给C，C通过创建对象访问M，M通过广播等建立与C的关系。

当多个控件，连线选择Outlet Collection，数组还会变化，自己去创建，所以strong。

但其中元素的引用是weak。

NSString不可变，当被赋以mutable时用strong会较危险，而若不成为mutable则没问题。copy生成副本，产生的是不可变。Strong效率更高。

@property(nonatomic,getter = isFaceUp)BOOL faceUp;将getter方法名字改为isFaceUp，调用getter方法：XXX＝ self.isFaceUp。

在.h和.m之间切换快捷键：control＋command＋上／下;

\_属性名 是实例变量，不走getter／setter方法。

生成无符号随机数：arc4random();

创建控制器对象不等于创建了里面的视图，当present才创建。

反向传值的关键：1.将A添加为B的属性（为防止循环引用造成内存泄漏，需weak）；2.A加入viewWillApear.

委托方：

1.定义协议：a.协议名：委托方类名Delegate；b.方法的第一个参数一定是委托方自己；c.方法名称要尽量体现发消息的时机；d.定义在.h文件。

2.增加用于纪录代理方的属性；

@class BViewController;

@protocol BViewControllerDelegate<NSObject>

-(void)bViewController:(BViewController \*)BVC didFinishInputWithMessage: (NSString \*)msg;

@end

@interface BViewController : UIViewController

@property(nonatomic,weak)id<BViewControllerDelegate> delegate;

@end

3.在合适的时机给代理方发消息。

[self.delegate bViewController:self didFinishInputWithMessage: self.BTextField.text ]

代理方：

1.遵守协议；.m中：@interface AViewController ()<BViewControllerDelegate>

@end

2.实现方法；

-(void)bViewController:(BViewController \*)BVC didFinishInputWithMessage: (NSString \*) msg{ self.resultLabel.text = msg; }

3.设置自己为代理方。

bvc.delegate = self;

window的根视图控制器是navigationController，还要设置navigationController的根视图控制器。

[self.navigationController pushViewController:BVC animated:YES];

返回： [self.navigationController popViewControllerAnimated:YES];

设置各自VC中的导航栏标题：.m中的viewDidLoad中：self.navigationItem.title = @"AVC";

self.navigationItem.rightBarButtonItem = (UIBarButtonItem \*)item;

显示工具栏：self.navigationController.toolbarHidden = NO;

UIBarButtonSystemItem; UIBarButtonItemStyle

设置toolBar：self.toolbarItems = (NSArray \*)array;

木棍特效：固定 UIBarButtonSystemItemFixedSpace

弹簧特效：可变 UIBarButtonSystemItemFlexibleSpace

隐藏状态栏（返回时该隐藏不生效, navigation和toolbar对所有VC有效，但其中的内容仅针对当前vc）：-(BOOL)prefersStatusBarHidden{ return YES; }

imageView： View.mode：1. scale Fill：填满，拉伸。2. Aspect Fit宽高比不变，看到全部，留白。3. Aspect Fill看到部分，不留白。（代码是.contentMode = UIViewContentModeScaleAspect…） 默认imageView不设置frame时，则位于左上角，size与原图相同。

滚动视图UIScrollView：

step1：创建滚动视图实例；

step2: 创建滚动视图内要容纳的其他控件；

step3: 将其他控件添加到滚动视图中；

step4: 添加滚动视图的其他属性；

step5: 将滚动视图添加到控制器的视图中。

设置大小位置、边界： scrollview.frame = self.view.frame; scrollview.contentSize = imageView.frame.size;

边缘弹跳效果：scrollview.bounces = NO;

水平滚动条不显示：scrollview.showsHorizontalScrollIndicator = NO;

属性contentOffSet：左顶点偏移量。

整页滚动：scrollView.pagingEnabled = YES;

缩放：1.遵守协议 2.设置maximumZoomScale和maximumZoomScale属性，设置自己为代理3.执行方法： - (UIView \*)viewForZoomingInScrollView: (UIScrollView \*)scrollView{ return self.imageView; }

分页控件UIPageControl:

1.创建；2.设置属性；

3.添加：添加到self.view中。

小圆点个数：pageController.numberOfPages = 4;

底色：pageController.pageIndicatorTintColor = [UIColor blackColor];

小圆点颜色：pageController.currentPageIndicatorTintColor = [UIColor redColor];

关闭用户交互功能：pageController.userInteractionEnabled = NO;

得到从CGPoint型的字符串型：NSStringFromCGPoint((CGPoint)point);

当前页：pageController.currentPage;

四舍五入函数：round()

表视图：

表头视图tableHeaderView、表尾视图tableFooterView、tableViewCell、分区头section header、分区尾section footer

设置表尾视图，使用频率不高，系统未创建实例，需自己创建实例：[footerView addSubview:label];self.tableView.tableFooterView = footerView;

2.表格属性：a. frame属性 b. 设置dataSource代理 d.设置delegate代理。

去队列中取单元格：UITableViewCell \*cell = [tableView dequeueReusableCellWithIdentifier: @"myCell"]; if (cell == nil) { cell = [[UITableViewCell alloc]initWithStyle: UITableViewCellStyleDefault reuseIdentifier: @"myCell"]; }

tableViewCell的三个属性：textLabel、detailTextLabel、imageView

辅助视图：通过.accessoryViewType属性或accessoryView设置单元格样式

cell.accessoryType = UITableViewCellAccessoryCheckmark; 后面对勾

cell.accessoryView =(UIView \*)

安装官方文档：

• 031-07735-A.dmg

• 安装

• 粘贴到 应用程序－xcode－显示表内容－contents—developer－documentation－docsets—docsets

xib创建tableView：1:cell对象，2:连线file’owner－delegate，dataSource，3:在三问和viewDidLoad调用

注册：[self.tableView registerClass:[UITableViewCell class] forCellReuseIdentifier: @"Cell"]；从队列取： UITableViewCell \*cell = [tableView dequeueReusableCellWithIdentifier: @"Cell" forIndexPath:indexPath];不用if。

开启表格的编辑模式：设置表格的editing属性为YES。

实现编辑：两问一答

1:能否编辑-(BOOL)tableView:(UITableView \*)tableView canEditRowAtIndexPath: (NSIndexPath \*)indexPath{ return YES; }

2:-(UITableViewCellEditingStyle)tableView:(UITableView \*)tableView editingStyleForRowAtIndexPath: (NSIndexPath \*)indexPath{ return UITableViewCellEditingStyleDelete; } |会出现勾选

3:该行做什么：-(void)tableView:(UITableView \*)tableView commitEditingStyle:(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath: (NSIndexPath \*) indexPath{[self.allContacts removeObjectAtIndex:indexPath.row];

[tableView deleteRowsAtIndexPaths: @[indexPath] withRowAnimation:UITableViewRowAnimationLeft];}

移动：一问一答

1:能否移动-(BOOL)tableView:(UITableView \*)tableView canMoveRowAtIndexPath:(NSIndexPath \*)indexPath{ return YES; }

2:移动后做什么,默认数据中的顺序没变。

-(void)tableView:(UITableView \*)tableView moveRowAtIndexPath: (NSIndexPath \*) sourceIndexPath toIndexPath:(NSIndexPath \*) destinationIndexPath{ }

阴影偏移：label.shadowOffset = CGSizeMake(1, 1);

字体居中：label.textAlignment = NSTextAlignmentCenter;

自定义单元格：

一：添加到contentView属性。

给其中的view加标签：UILabel \*label = (UILabel \*)[cell.contentView viewWithTag:1];……………label.tag=1;

二：自定义继承UITableViewCell的类：

注册：[self.tableView registerNib:[UINib nibWithNibName:@"TRTableViewCell" bundle:nil] forCellReuseIdentifier:@"myCell"];

行高：-(CGFloat)tableView:(UITableView \*)tableView heightForRowAtIndexPath: (NSIndexPath \*)indexPath{ return 100; }

圆圈i的设置：cell.accessoryType = UITableViewCellAccessoryDetailButton;

与点击：-(void)tableView:(UITableView \*)tableView accessoryButtonTapped…

标签控制器：UITabBarController （下方、并列）

UITabBarController \*tab = [[UITabBarController alloc]init];tab.viewControllers = @[avc,bvc,cvc]; self.window.rootViewController = tab;

设置图片：self.tabBarItem.image self.tabBarItem.selectedImage

红色角标：self.tabBarItem.badgeValue

跳转： self.tabBarController.selectedIndex = (int) ;

推出时隐藏： dvc.hidesBottomBarWhenPushed = YES;

16进制表示颜色：#a3 ad bc

UICollectionViewController

Cell属性.backgroundColor .backgroundView .contentView

UICollectionView继承自UIScrollView

AppDelegate中需要继承UICollectionViewController和UICollectionViewF lowLayout的类：

self.window.rootViewController = [[myCollectionViewController alloc] initWithCollectionViewLayout: [[myFlowLayout alloc]init]];

UISegmentedControl：调用：sender.selectedSegmentIndex

segment.addTarget(self, action: "controlPressed:", forControlEvents: UIControlEvents.ValueChanged)

UIActivityIndicatorView: 调用：[self.indicator startAnimating];

UIProgressView： 调用：self.myProgress.progress+=0.1;

补充：NSTimer: NSTimer scheduledTimerWithTimeInterval:(NSTimeInterval) target:<#(id)#> selector:<#(SEL)#> userInfo:<#(id)#> repeats:<#(BOOL)#>

当使用NSTimer \*timer= [NSTimer timerWithTimeInterval:1 target:self selector:@ selector (changeValue:) userInfo:nil repeats:YES];需要手动将定时器添加到事件循环中：[[NSRunLoop currentRunLoop]addTimer:timer forMode:NSDefaultRunLoopMode]; 停止定时器：[self.timer invalidate];

日期控件UIDatePicker:使用：NSDate \*date = self.datePicker.date; NSDateFormatter \*formatter = [[NSDateFormatter alloc]init]; formatter.dateFormat = @"yyyy-MM-dd HH:mm:ss"; NSString \*dateString = [formatter stringFromDate:date]; self.dateLabel.text = dateString;

UIPickerView选择器：遵守<UIPickerViewDataSource,UIPickerViewDelegate>

三问：-(NSInteger)numberOfComponentsInPickerView:(UIPickerView \*)pickerView{ }; -(NSInteger)pickerView:(UIPickerView \*)pickerView numberOfRowsInComponent: (NSInteger)component{ }; -(NSString \*)pickerView:(UIPickerView \*)pickerView titleForRow:(NSInteger)row forComponent:(NSInteger)component{ }

选择：-(void)pickerView:(UIPickerView \*)pickerView didSelectRow: (NSInteger)row inComponent:(NSInteger)component{可能用到：[pickerView reloadComponent:1]; [pickerView selectRow:0 inComponent:1 animated:YES]; }

添加工具栏：navigation Controller.（4）show Toolbar

隐藏某个vc的工具栏: vc.(4)hide Bottom Bar on Push

当创建控制器时会自动调用-(id)initWithCoder:(NSCoder \*)aDecoder{ }

tableView的detail加其他：先拖到最上边，在（6）连线accessory…

拖view到表格顶部，可设置表头视图。为出现圆圈i，用accessory push

静态表视图

动态表视图：系统原型、自定义原型、混合原型。

用storyBoard创建需推出的界面:blueViewController \*bvc =[self.storyboard instantiateViewControllerWithIdentifier:@"BlueScene"];若有xib,创建跟以前一样

segue传值: -(void)prepareForSegue:(UIStoryboardSegue \*)segue sender:(id)sender{ blueViewController \*bvc = segue.destinationViewController;

bvc.msg = self.myTextField.text; }

多个segue时，在prepareForSegue方法中用segue.identifier判断是哪个。

[self.mySlider setMaximumTrackImage:[UIImage imageNamed:@" playing\_volumn\_slide\_foreground"] forState:UIControlStateNormal];设置右边形状

绘图写在drawRect、drawLayer、drawImage…方法中

基于CGGraphyic：

开启上下文UIGraphicsBeginImageContextWithOptions (CGSizeMake(imageWidth, imageHeight), NO, 0);

（drawRect方法中）获取上下文对象（画笔）CGContextRef context = UIGraphicsGetCurrentContext();

CGContextMoveToPoint(context, 40, 40);CGContextAddLineToPoint(context, 40, 140);CGContextAddLineToPoint(context, 140, 40); CGContextAddLineToPoint (context, 40, 40);CGContextSetStrokeColorWithColor(context, [[UIColor redColor]CGColor]);CGContextStrokePath(context); CGColor方法得到CGColorRef类型

既描边又填充，加上：CGContextSetFillColorWithColor(context, [[UIColor redColor]CGColor]);.. CGContextDrawPath(context, kCGPathFillStroke);

drawRect时机：只在创建视图实例时，由系统自动调用，不能手动调用。

UIBezierPath \*path = [UIBezierPath bezierPath];

[[UIColor redColor] setStroke];把颜色给画笔

path.lineCapStyle = kCGLineCapRound;线头样式

path.lineJoinStyle = kCGLineJoinRound;线连接处样式

得到合适的大小(画字符串)CGRect rectOfRect=[str boundingRectWithSize:<#(CGSize)#> options:<#(NSStringDrawingOptions) #> attributes:<#(NSDictionary \*)#> context:<#(NSStringDrawingContext \*)#>];

UIGraphicsBeginImageContext(CGSize size);(中间区域可以编写任何与绘图有关的代码，该区域外不准写类似setStroke绘图代码…; UIImage \*newImage= UIGraphicsGetImageFromCurrentImageContext();)UIGraphicsEndImageContext();

画字符串和图片：drawAtPoint不能换行，drawInRect可以。

设置导航栏颜色：[[UINavigationBar appearance]setBarTintColor:[UIColor blackColor]]; 不能setBackgroundColor 有导航时修改状态栏中文字的颜色 [[UINavigationBar appearance]setBarStyle:UIBarStyleBlack]; 设置导航栏背景图：[[UINavigationBar appearance]setBackgroundImage:<#(UIImage \*)#> forBarMetrics:<#(UIBarMetrics)#>] UIBarMetrics是枚举。 左右按钮的颜色：[[UINavigationBar appearance]setTintColor:[UIColor whiteColor]]; 返回按钮：[[UINavigationBar appearance] setBackIndicatorImage:<#(UIImage \*)#>]; [[UINavigationBar appearance] setBackIndicatorTransitionMaskImage:<# (UIImage \*)#>] title字体样式：[[UINavigationBar appearance] setTitleTextAttributes:<#(NSDictionary \*)#> forState: UIControlStateNormal]; title字体偏移[[UITabBarItem appearance]setTitlePositionAdjustment:UIOffsetMake(0, -1)];

阴影NSShadow \*shadow=[[NSShadow alloc]init];shadow.shadowColor=[UIColor redColor]; shadow.shadowOffset=CGSizeMake(0, 1);作属性:NSShawdowAttribu..

改self.view.bounds的origin改变的是self.view的坐标原点，从而影响其中子视图的相对位置

button1.translatesAutoresizingMaskIntoConstraints = NO;关闭默认约束

NSDictionaryOfVariableBindings(obj1,obj2);返回{@”obj1”:obj1,@”obj2”,obj2};

当变屏改动较大时：画布

UISearchController

<UISearchResultsUpdating,UISearchBarDelegate>

关联：self.searchC = [[UISearchController alloc] initWithSearchResultsController:self.resultTVC];

设置搜索控制器的结果更新代理对象self.searchC.searchResultsUpdater = self;

设置显示的searchBar的大小和样式 [self.searchC.searchBar sizeToFit];

self.searchC.searchBar.scopeButtonTitles = @[@"设备",@"软件",@"其他"];

将seachBar添加到表头中显示

self.tableView.tableHeaderView = self.searchC.searchBar;

开启在当前控制器中，允许切换另一个视图做呈现

self.definesPresentationContext = YES;

为了感知用户在搜索条上选择了不同的button，设置代理

self.searchC.searchBar.delegate = self;…………

-(void)searchBar:(UISearchBar \*)searchBar selectedScopeButtonIndexDidChange:(NSInteger)selectedScope{ …………

NSString \*searchText = self.searchC.searchBar.text; NSInteger selectedButton = self.searchC.searchBar.selectedScopeButtonIndex;

for (Product \*p in self.allProducts) { if (p.type == selectedButton) { NSRange range = [p.name rangeOfString:searchText]; if (range.length>0) { [self.searchResultArray addObject:p]; } } }

吸附 self.attachment.damping = 0.8;阻尼 self.attachment.frequency = 0.8;

self.attachment.action = ^{这里面再用self会造成循环引用，需要先在block外\_\_weak另外一个引用}

碰撞collision.translatesReferenceBoundsIntoBoundary = YES;

[collision addBoundaryWithIdentifier:@"p1" forPath:path];

字符串本身就有.lastPathComponent属性

获取Documents的路径: NSString \*docPath = [NSHomeDirectory() stringByAppendingPathComponent:@"Documents"]; 或：

NSString \*docPath2 = NSSearchPathForDirectoriesInDomains( NSDocumentDirectory, NSUserDomainMask, YES)[0];

获取tmp文件夹路径的方法NSString \*tmpPath = NSTemporaryDirectory();

一个项目：

1:deadline 2:软硬件:(iphone、ipad、ios版本) 3:开发人员:美工(fulltime、30%?)、开发人数、测试人员 4:4.1主要功能(最重要) 4.2code review 5测试: 测试人员: new->passed; new->defect; invalid->reopened开发人员: defect->worked->fixed; defect->invalid 6.上线 7.备注:自然日还是工作日

大型公司：1:需求：客户 2:开发设计 3:开发任务分配 4:测试 5:上线

规则：类前缀；类命名：首字母大写、驼峰；变量命名：首字母小写、驼峰

project分组：group分类：MVC、功能；code的分类：#progma mark -

常用代码的封装：

comments(注释)

自上而下:UI->业务逻辑->数据处理 自下而上:数据处理->业务逻辑->UI

三方库：www.github.com SDWebImage-master.zip

XML官方网站：http:w3school.com.cn

网络封包工具Charles

查询www.google.com www.stackoverflow.com

游戏课程http://pan.baidu.com/s/leQEpXPS

UIPopoverController浮动窗口，苹果提供ipad，iphone用的是第三方的。

Step1: 创建浮动窗口视图控制器；

Step2:创建浮动窗口控制器；

Step3:将浮动窗口控制器与导航栏右按钮关联。

％％％％％％％％％

视图层：

绘制、贴图、手势、变形、布局、动画、动力、特效。

显示器分辨率

图片像素，1个像素由4个数表示：R、G、B、Alpha，占4Byte。图片大小：像素×4。原图一般是bmp，压缩是jpg／png。

保存方式：位图（由像素点组成的图）。矢量图（保存的不是点阵的数据，而是数学公式，函数等，点、线、面几何形状，阴影放大缩小，不会失真，电子贺卡）。

绘图的API

UIKit Framework OC语言 API

＋Core Animation C语言 ApI

＋Core Graphics C 语言 API

＋System（IOS）

UIView——>drawRect会在视图显示前自动调用。

Graphics Context：保存绘图信息、绘图的形状、决定绘制的输出目标。

绘制好一个图形（保存）－>图形上下文－>输出目标（可以输出到Bitmap、PDF、Window、Layer、Printer）

1.画板：上下文

CGContextRef context = UIGraphicsGetCurrentContext();得到当前环境下的上下文。

2.勾勒：路径

CGContextMoveToPoint(context, <#CGFloat x#>, <#CGFloat y#>);

CGContextAddLineToPoint(context, 20, 100);

3.调色：设置颜色

CGContextSetStrokeColorWithColor(context, [[UIColor redColor]CGColor]);

4.上色：画

CGContextStrokePath(context);

BezierPath是对底层Core Graphics的封装。

线宽：path.lineWidth=20;

线头样式：path.lineCapStyle = kCGLineCapSquare;

连结处样式：path.lineJoinStyle = kCGLineJoinRound;

画圆弧：UIBezierPath \*path = [UIBezierPath bezierPathWithArcCenter:

<#(CGPoint)#> radius:<#(CGFloat)#> startAngle:<#(CGFloat)#> endAngle:M\_PI\_2 clockwise:<#(BOOL)#>]

路径：[path addLineToPoint:CGPointMake(160, 200)];

调色： [[UIColor redColor]setStroke]; [path stroke];

画： [path stroke];

由控制点画曲线：[path addCurveToPoint:CGPointMake(50, 120) controlPoint1: CGPointMake(50, 20) controlPoint2:CGPointMake(100, 120)];

画圆角矩形：UIBezierPath \*path2 = [UIBezierPath bezierPathWithRoundedRect: CGRectMake(150, 50, 100, 80) cornerRadius:10];

画椭圆：UIBezierPath \*path3 = [UIBezierPath bezierPathWithOvalInRect: CGRectMake(50, 160, 180, 180)];

画字符串：[str drawAtPoint:CGPointMake(20, 50) withFont:[UIFont italicSystemFontOfSize:30]];已过期

[message drawInRect:CGRectMake(20, 80, 280, 300) withFont:[UIFont italicSystemFontOfSize:30]];已过期

自适应画字符串：UIFont \*font = [UIFont italicSystemFontOfSize:20];

CGSize size = [message sizeWithFont:font constrainedToSize:CGSizeMake(280, 999)];

[message drawInRect:CGRectMake(20, 200, size.width, size.height) withFont:font];过期

IOS带的画字符串：NSDictionary \*attributes = @{NSFontAttributeName:[UIFont italicSystemFontOfSize:90],NSForegroundColorAttributeName:[UIColor redColor],nil};

[str drawAtPoint:CGPointMake(20, 30) withAttributes:attributes];

画图片：

[image drawAtPoint:CGPointMake(20, 20)];

剪切图片的内容：UIBezierPath \*path=[UIBezierPath bezierPathWithOvalInRect: CGRectMake(50, 50, 180, 180)];[path addClip]; [path stroke]; addClip有效区

贴图：

基本概念：几寸屏是对角线长度

iphone4/4S 640\*960 320\*480

5/5s/5c 640\*1136 320\*568

6 750\*1334 375\*667(4v1)2x

6plus 1242\*2208 414\*736(9v1)3x

new ipad、ipad4、ipad Air、ipad

Air2、ipad mini2、ipad mini3 2048\*1536 1024\*768

非Retina高清：

ipad、ipad2

ipad mini 1024\*768 1024\*768

iphone3GS 320\*480 320\*480

IOS系统加载图片时，会对Retina相关图片进行处理。自动选择@2x的资源，若没有，则选择不含@2x的。

清空项目缓存（product－>Clear）

清空模拟器缓存（删除模拟器中的app或IOS模拟器->还原内容和设置）

9切片技术(9slick)：解决不同尺寸，相同设计风格的图片，需要多张图片的问题。

UIImage \*resizableImage = [image resizableImageWithCapInsets: UIEdgeInsetsMake(0,10,0,10) resizingMode:(处理方式：默认tile)];

Images.xcassets目录可不写代码支持9slicing

titColor属性：通过修改父视图颜色来批量修改子视图颜色。

UIApearance处理相同类型的多个控件: [[UIButton appearance]setImage: [UIImage imageNamed:@”TypeSelectorBtn\_Black” forState: UIControlStateNormal]];

应用图标：image.xcassets中，根据对应尺寸拖图片到Appicon

闪屏界面：点最上面，找到App Icons and Lauch Images, Launch Image Sourse选LaunchImage, 再image.xcassets中，根据警告拖相应图片。

手势（GestureRecognizer）

一次性手势Tap：点、按 Swipe：轻扫（查看图片）

连续性手势：LongPress长按(选择功能，复制、粘贴) Pan拖动(移动图标) Pinch捏／拨／扩(放大或缩小) Rotation旋转

1.创建手势对象：UITapGestureRecognizer \*tapGR = [[UITapGestureRecognizer alloc]initWithTarget:self action:@selector(tap)];

2.设置属性：

点击多少次tapGR.numberOfTapsRequired = 1（2双击）;

几个手指：tapGR.numberOfTouchesRequired;

3.添加到相应视图中：[self.view addGestureRecognizer:tapGR];

UISwipeGestureRecognizer轻扫、滑动：

.numberOfTouchesRequired .direction扫的方向， 可以用｜将多个方向的枚举值合并触发同一个事件方法，但不支持同时垂直的两个方向。GS.direction = UISwipeGestureRecognizerDirectionLeft|UISwipeGestureRecognizerDirectionRight;；

长按：UILongPressGestureRecognizer：

属性.numberOfTouchesRequired；.minimunRessDuration多长时间算长按；.allowableMovement允许在手势识别期间手指可移动的范围。

捏／扩：UIPinchGestureRecognizer

属性：.scale捏／扩的倍数 .velocity速度，负（捏），正（扩）

拖动：UIPanGestureRecognizer

属性：.minimunNumberOfTouches最少几个手指

方法：

获取用户当前手指所在位置坐标（相对于启始位置）：-(CGPoint)translationInView:self.view

获取用户当前拖动方向和速度：-(CGPoint)velocityInView:(UIView \*)view;

返回绝对位置的坐标（从父类继承）：-(CGPoint)locationInView:self.view

旋转：UIRotationGestureRecognizer

属性：.rotation旋转的弧度 .velocity旋转的速度 （顺时针为正，逆时针为负）

变形（Transform）：可以让一个视图进行旋转、缩放和位移的变化。改变一个视图的对象属性.transform就可以得到变形的效果。对.transform的属性做不同的改变，就可以使视图：旋转(Rotation)、缩放(scale)、位移(translation)。

.transform是CGAffineTransform的结构体，有6个成员值，再加上3个固定值，形成了3\*3的矩阵，6个值发生变化，就会导致视图发生不同的变形。变形与自动布局冲突。

CGAffineTransformRotation创建一个旋转的transform

CGAffineTransform transform = self.myimageView.transform得到当前状态

transform=CGAffineTransformRotate(transform, sender.rotation)在原来的基础上叠加旋转角度。CGAffineTransformMakeRotation(<#CGFloat angle#>)则是从一开始计算。

self.myImageView.transform = transform;赋值给原来的view；sender.rotation=0不清零则会有Rotation叠加的效果}

CGAffineTransformMakeScale创建一个缩放的transform，从1开始计算。

CGAffineTransformScale(transform,scale,scale)在原来的基础上进行叠加。与pinch手势配合使用。

位移CGAffineTransformTranslate(transform,translation.x,translation.y);

在原来的位移基础上进行叠加[sender setTranslation:CGPointZero inView:selfView]，与Pan手势配合使用;

手势状态属性：state

self.myImageView.transform=CGAffineTransformMakeScale()

清零:[sender setTranslation:CGPointZero inView:self.view];

拖动后还原初值:self.myImageView.transform=CGAffineTransformIdentity;

CGAffineTransformIdentity系统定义的所有视图对象.fransform属性的常量初值。

UIImageView的手势交互需要手动开启（默认关闭）：

Self.imageView.userInteractionEnabled = YES;

Storyboard创建手势：

1:拖拽到相应的view中；2:连线，修改交互属性；3:添加功能

若要视图同时满足多个手势，需要向被委托对象发消息。

1：遵守协议UIGestureRecognizerDelegate

2:实现方法：-(BOOL)gestureRecognizer:(UIGestureRecognizer \*) gestureRecognizer shouldRecognizeSimultaneouslyWithGestureRecognizer: (UIGestureRecognizer \*)otherGestureRecognizer{if ([gestureRecognizer isKindOfClass:[UIRotationGestureRecognizer class]]&&[otherGestureRecognizer isKindOfClass:[UIPanGestureRecognizer class]]) { return YES; }return NO;}

注：isKindOf:判断本类和子类，isMemberOf:仅本类

3:将委托对象与被委托对象关联在一起。

坐标系的四个属性：

.frame .bounds .transform .center

.frame是相对父视图而言，与transform和AutoLayout冲突。

改.frame会改变.center，不会改变.transform(增量)，会改变bounds(x和y永远是0，得到视图本身大小，排除用.frame时父视图部分包含子视图的情况，或者写View的子类（自定义Cell），改bounds不会改变center和transform)；

center改变会改变.frame，不会改变bounds和transform

.transform会改变.frame（间接被改变），不会改变.bounds和center?

可以理解为：.bounds和.center通过.transform映射变化得.frame

响应者对象：只有继承了UIResponder的对象，才能接受并处理事件。

UITouch：比如实现绘图、涂鸦和手写等操作。

实现：touchesBegan、touchesMoved、touchesEnded方法

-(void)touchesBegan:(NSSet \*)touches withEvent:(UIEvent \*)event{UITouch \*touch = [touches anyObject]; CGPoint point=[touch locationInView:self]; CGPoint prePoint = [touch previousLocationInView:self];}

布局（Layout）：一个视图中如何摆放子视图（安排子视图的位置和大小）

当屏幕大小变化，父视图的bounds会改变。

1:纯代码布局（手动布局，一定不选AutoLayout）：屏幕大小发生变化时，自动执行一段代码计算各视图的frame

代码方式：-(void)viewDidLayoutSubviews{//写计算位置大小的代码}，当屏幕大小改变时，会自动调用。

上边栏所占空间的大小：是id类型id<UILayoutSupport> top=self.topLayoutGuide top.length

下边栏所占空间的大小： .bottomLayoutGuide

隐藏状态栏；-(BOOL)prefersStatusBarhidden

设置导航栏隐藏：self.navigationBarHidden=YES;

绘图布局：将属性contentMode设置成Redraw

layoutSubView是自动，setNeedsLayout是主动

调用顺序：VC: viewWillLayoutSubViews

View: layoutSubViews

VC : viewDidLayoutSubViews

2:Autoresizing：ios5及以前的自动布局，可以和代码布局一起使用。关Autolayout

原理：当界面大小发生变化时，根据变化的比例，对视图进行同比例变化。

第4个选项卡。

代码方式会在autoResizing之后执行。

button.autoresizingMask = UIViewAutoresizingFlexibleLeftMargin;

3:AutoLayout：ios6开始有，可以代码，也可以storyboard。

本质是创建很多对子视图的约束，让系统自动计算。

代码方式：NSLayoutConstraint \*constraint1 = [NSLayoutConstraint constraintWithItem: <#(id)#> attribute: <#(NSLayoutAttribute)#> relatedBy: <#(NSLayoutRelation)#> toItem:<#(id)#> attribute:<#(NSLayoutAttribute)#> multiplier:<#(CGFloat)#> constant:<#(CGFloat)#>]; [self.view addConstraints: @[constraint1,constraint2]];

VFL: NSString \*vflString = @"H:|-20-[button1]-10-[button2(button1)] -10-[button3(button2)]-20-|";

NSArray \*constrans=[NSLayoutConstraint constraintsWithVisualFormat: vflString options: NSLayoutFormatAlignAllCenterY metrics:nil views:@ {@"button1":button1,@"button2":button2,@"button3":button3,}];

NSDictionary \*metrics = @{@"left":@20,@"right":@20,@"top":@20,

@"spacing":@10 };

NSDictionary \*dictionay = NSDictionaryOfVariableBindings(b1,b2,b3);

NSString \*hVFL=@"|-left-[b1]-spacing-[b2(b1)]-spacing-[b3(b1)]-right-|";

NSArray \*cs1=[NSLayoutConstraint constraintsWithVisualFormat:hVFL options:NSLayoutFormatAlignAllCenterY metrics:metrics views:dictionay];……

NSString \*vVFL=@"V:|-top-[b1]";……..

FPS(帧率):Frame Per Second

UIKit Dynamic动力 Motion Effects特效 Sprite Kit(引擎) Metal

[UIImage animatedImageNamed:<#(NSString \*)#> duration:<#(NSTimeInterval) #>];

动画：NSTimer、UIView、CoreAnimation

NSTimer手动动画：

[NSTimer scheduledTimerWithTimeInterval:1 target:self selector:@selector (updateTitle:) userInfo:nil repeats:YES];….-(void)updateTitle:(NSTimer \*)timer{}

停止：[timer invalidate];

匀速动画: 当前值＝开始值＋当前的帧数\*(结束值-开始值)/(帧率\*时长)

变速动画：当前值＝上一次值＋(目标值-上一次值)\*渐变因子

UIVIew动画：[UIView animateWithDuration:<#(NSTimeInterval)#> animations:^{结束属性值（当对象取该值结束动画）}];

高级动画(与AutoLayout冲突)：[UIView animateWithDuration: <#(NSTimeInterval)#> delay:<#(NSTimeInterval)#> options:<# (UIViewAnimationOptions)#> animations:^{ self.myLabel.frame = endFrame; self.myLabel.transform = CGAffineTransformMakeScale(0.1, 0.1);越来越小} completion:<#^(BOOL finished) completion#>] option特效:repeat重复、EaseInOut慢快慢、CurveEaseIn越来越快、CurveEaseOut越来越慢、CurveLinear匀速，可用“|”支持多个特效

动画标识：[UIView beginAnimations:[NSString stringWithFormat: @"%d",self.count] context:nil];

设置：snow.tag = self.count; [UIView setAnimationDuration:arc4random ()%MAX\_DURATION+2];

[UIView setAnimationCurve:UIViewAnimationCurveEaseIn];

[UIView setAnimationDelegate:self]; [UIView setAnimationDidStopSelector:@selector(snowDisapear:)]; [UIView commitAnimations];

-(void)snowDisapear:(NSString \*)animatedID{

[UIView beginAnimations:animatedID context:nil];

[UIView setAnimationDuration:arc4random()%DISPEAR\_DURATION+2];

[UIView setAnimationCurve:UIViewAnimationCurveEaseIn];

UIImageView \*snow=(UIImageView \*)[self.view viewWithTag:[animatedID intValue]]; snow.alpha = 0.0; [UIView setAnimationDelegate:self];

[UIView setAnimationDidStopSelector:@selector(snowRemoved:)];

[UIView commitAnimations]; }

-(void)snowRemoved:(NSString \*)animatedID{

UIView \*snow=[self.view viewWithTag:[animatedID intValue]];

NSInteger count = [self.view.subviews count];NSLog(@"%lu",count);

[snow removeFromSuperview];count=[self.view.subviews count];

NSLog(@"%lu",count); }

CoreAnimation框架

CAlayer作用：1.修改view的角：self.myButton.layer.cornerRadius = 5;

对某些视图，打开蒙板效果，遮盖：self.myImageView.layer.cornerRadius =10;

self.myImageView.layer.masksToBounds = YES;

自定义layer： CALayer \*customlayer2 = [[CALayer alloc]init]; customlayer2.backgroundColor = [[UIColor grayColor]CGColor];(解封) …. [self.view.layer addSublayer:customlayer2];

layer.transform是CATransform3D类型

关键帧动画与UIBeziapath结合做动画

CAKeyframeAnimation \*moveAnimation = [CAKeyframeAnimation animationWithKeyPath:@"position"];………….moveAnimation.path = [path CGPath];

moveAnimation.duration = 2.0; moveAnimation.removedOnCompletion = YES;

[self.myImageView.layer addAnimation:moveAnimation forKey:@" moveAnimation"]; forKey对应其他调用，如remove

基础动画、缩放动画：

CABasicAnimation \*scaleAnimation = [CABasicAnimation animationWithKeyPath:@"transform"]; scaleAnimation.duration = 2.0; scaleAnimation.fromValue = [NSValue valueWithCATransform3D:CATransform3DIdentity];

scaleAnimation.toValue = [NSValue valueWithCATransform3D: CATransform3DMakeScale(0.1, 0.1, 0.1)]; scaleAnimation.removedOnCompletion = YES;

[self.myImageView.layer addAnimation:scaleAnimation forKey:nil];

基础动画没.path属性，做旋转、缩放、透明度“opacity”，关键帧动画一般做位置变化。

绕z旋转CABasicAnimation \*rotationAnimation = [CABasicAnimation animationWithKeyPath:@"transform.rotation.z"];

rotationAnimation.toValue = @(M\_PI\*2); rotationAnimation.duration = duration; rotationAnimation.repeatCount = 2147483647;重复次数 rotationAnimation.removedOnCompletion = NO; .fillMode = kCAFillModeBackwards;方向

if (sender.state == UIGestureRecognizerStateEnded) { [UIView animateWithDuration:2 delay:0 options:UIViewAnimationOptionCurveEaseInOut animations:^{

self.topLayoutContraint.constant = self.topConstant; } completion:nil]; }

CATransform3DIdentity:任何一个layer对象的初始值。

旋转：CABasicAnimation \*ro = [CABasicAnimation animationWithKeyPath:@"transform"];………….. ro.toValue = [NSValue valueWithCATransform3D:CATransform3DRotate(CATransform3DIdentity, M\_PI, 0.0, 0.0, 1.0)];………….. 0和1表示对应的旋转是否执行

停在最后状态：加上ro.delegate = self; -(void)animationDidStop:(CAAnimation \*)anim finished:(BOOL)flag{ self.myImageView.layer.transform = CATransform3DRotate(CATransform3DIdentity, M\_PI,0.0, 1.0, 0.0); }

动力UIKit Dynamic

UIDynamicAnimator(核心类)，需要父视图环境

UIDynamicBehavior(具体动力)：UIGravityBehavior; UICollisionBehavior; UIAttachmentBehavior; UISnapBehavior; UIPushBehavior

创建动力总管：UIDynamicAnimator \*animator = [[UIDynamicAnimator alloc]initWithReferenceView:self.view]; self.animator = animator;

创建重力对象：UIGravityBehavior \*gravity=[[UIGravityBehavior alloc]initWithItems:@[self.myImageView]]; self.gravity = gravity;

将重力对象加入总管：[self.animator addBehavior:self.gravity];

移除重力： [self.animator reMoveBehavior:self.gravity];

重力方向gravity.gravityDirection = CGVectorMake(0, 1);数值大小也能决定强度

重力强度：gravity.magnitude = 0.1;

能发生碰撞：collision.translatesReferenceBoundsIntoBoundary = YES;

添加碰撞边界[collision addBoundaryWithIdentifier:@"myPath" forPath:path];

碰撞后触发-(void)collisionBehavior:(UICollisionBehavior \*)behavior endedContactForItem:(id<UIDynamicItem>)item withBoundaryIdentifier:(id <NSCopying>)identifier{ }

？？碰撞后处理：-(void)collisionBehavior:(UICollisionBehavior \*)behavior beganContacForItem:(id<UIDynamicItem>)item withBoundaryIdentifier:(id <NSCopying>)identifier atPoint:(CGPoint)p{ UIImageView \*box = (UIImageView \*)item; box.tintColor = [UIColor redColor];box.image = [box.image imageWithRenderingMode:UIImageRenderingModeAlwaysTemplate]; }

瞬移：在tap事件方法中self.snapBehavoir = [[UISnapBehavior alloc] initWithItem:self.myImageView snapToPoint: [sender locationInView:self.view]];

NSNotificationCenter \*notificationCenter = [NSNotificationCenter defaultCenter];

通知中心注册(监听)：[notificationCenter addObserver:self selector:@selector (doit) name:@"identifier" object:nil]; 参数1:消息接收者，参数2:接收后的动作，3:通知的标识，区分不同的消息，4:接收哪些对象消息，nil为接收所有消息。

NSNotification \*notification=[[NSNotification alloc]initWithName:@"identifier" object:nil userInfo:nil]; 发送消息：[notificationCenter postNotification:notification];

[[NSNotificationCenter defaultCenter] postNotificationName:@" TRSortDidChange" object:nil userInfo:@{@"TRSelectedSort" : [TRMetaDataTool sorts] [button.tag]}]; 参1:通知标识 参3:发送哪些消息,其中key用于接收者拿到并获取value

模型层：

文件处理：NSFileManage(创建、删除、改名、探测) NSFileHandle(读写) NSdata

补充路径NSString ＊path2=[path stringByAppendingPathComponent:@"time副本.txt"];

按字符串拆NSArray \*components=[path componentsSeparatedByString:@"/"]

添加扩展名NSString \*path3=[path stringByAppendingPathExtension:@"txt"]

得到扩展名：NSString \*extension = [path3 extension];

获取最后一个component: NSString \*lastComponent =[path lastPathComponent]

删除最后一个/得到父路径:NSString \*path=[path stringByDeletingLastPathComponent]

判断文件是否存在NSFileManager \*fileManager = [[NSFileManager alloc]init]; BOOL isExist =[ fileManager fileExistsAtPath:path];

区分是文件或文件夹BOOL isFolder;if ([fileManager fileExistsAtPath:path isDirectory:&isFolder]) { NSLog(@"%@",isFolder?@"文件夹":@"文件"); }方法返回是否存在，isFolder得到是文件夹(YES)还是文件

返回文件夹下的所有项目NSString \*path2 = @"/Users/tarena/Desktop"; NSArray \*contents=[fileManager contentsOfDirectoryAtPath:path2 error:nil];

创建文件夹(参数2表示是否同意隔级)：[fileManager createDirectoryAtPath:@"/Users/tarena/Desktop/time" withIntermediateDirectories:YES attributes:nil error:nil];

沙盒机制: Documents保存应用过程中建立的文件，Library保存不希望暴露给用户的文件，如缓存Caches和偏好设置Preference。tmp保存程序运行时，需要的临时处理的数据，使用完毕后再将相应的文件从该目录删除，如果当前程序没有在运行状态，系统会自动删除该目录的文件。

得到沙盒所在根目录NSString \*homePath = NSHomeDirectory();

获取Documents目录:NSArray \*documentPaths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);

获取tmp目录: NSString \*tmpPath = NSTemporaryDirectory();

写入：BOOL isSuccess = [content writeToFile:filePath atomically:YES encoding:NSUTF8StringEncoding error:nil];

获取文件管理器：NSFileManager \*fileManager = [NSFileManager defaultManager];

创建文件：NSString \*firstTestFile = [testDirectory stringByAppendingPathComponent:@"first.txt"]; [fileManager createFileAtPath:firstTestFile contents: [@"second" dataUsingEncoding:NSUTF8StringEncoding] attributes:nil];

获取某一目录下的文件列表：NSArray \*fileArray = [fileManager subpathsOfDirectoryAtPath:testDirectory error:nil];包括子文件，comtents是当前文件夹

处理隐藏文件：command+F -> 种类 -> 文件不可见

迭代器: NSEnumerator \*enum=[fileManager enumeratorAtPath:testDirectory]; NSString \*obj = nil; while (obj=[Enum nextObject]) { NSLog(@"%@",obj); }

拷贝：[fileManager copyItemAtPath:testDirectory toPath:firstTestCopyPath error:nil];

删除：[fileManager removeItemAtPath:testDirectory error:nil];

使用url创建：NSURL \*url = [NSURL URLWithString:@"file:///Users/tarena /.../Documents/testurl"]; [fileManager createDirectoryAtURL:url withIntermediateDirectories:YES attributes:nil error:nil];

判断后缀: [path hasSuffix:@".png"]||[path hasSuffix:@".jpg"]

NSFileHandle:操作具体某个文件的内容

写入NSFileHandle \*fileHandle = [NSFileHandle fileHandleForWritingAtPath:path]; NSString \*content = @"用NSFileHandle写入测试内容";[fileHandle writeData:[content dataUsingEncoding:NSUTF8StringEncoding]]; [fileHandle closeFile];

拷贝：NSFileHandle \*sourceFileHandle=[NSFileHandle fileHandleForReadingAtPath:self.sourceFilePath]; NSData \*sourceData=[sourceFileHandle readDataToEndOfFile]; NSFileHandle \*targetFileHandle=[NSFileHandle fileHandleForWritingAtPath:path1]; [targetFileHandle writeData:sourceData];

较大文件多次拷贝：NSDictionary \*fileDic = [self.fileManager attributesOfItemAtPath:sourceFilePath error:nil];获取原文件的属性。NSNumber \*fileSizeNum = [fileDic objectForKey:NSFileSize];得到属性中文件大小的值。从文件中读取一定长度NSData \*data = [sourceFileHandle readDataOfLength:sizePreTimes];原文件的读取指针移动一定距离[sourceFileHandle seekToFileOffset:readSize];

数据持久化技术: NSUserDefault、plist属性列表、Archieving归档技术、SQLite3、Core Data、NSFileManager

NSUserDefault：支持的对象类型：NSString、NSNumber、NSData、NSArray、NSDictionary、BOOL、NSInteger、NSFloat，不支持自定义对象类型。

特性：往NSUserDefault添加数据后，它就变成了一个全局变量，即整个app中都可读、写NSUserDefault中的数据。可重复读写针对同一个关键字对应的对象。NSUserDefault定时往磁盘/硬盘写数据，而不是及时写入，如果需手动写入磁盘，需调用synchronize函数。持久化后的数据存到Documents/Library

command+shift+h ＝ home键 command+shift+h+h ＝ 进入后台

从文件读取数组：NSArray \*dataArray=[NSArray arrayWithContentsOfFile:self.plistFilePath];

归档/解档(Archiving)

Archiving：既可基本数据类型，还可保存自定义类型。

\_archivingFilePath = [[NSHomeDirectory() stringByAppendingPathComponent:@"Documents"] stringByAppendingPathComponent:@"archiving.bin"];

针对自定义类型：遵循NSCoding协议；实现NSCoding中的方法；- (void) encodeWithCoder:(NSCoder \*)aCoder{ [aCoder encodeObject:self.name forKey: @"name"]; [aCoder encodeObject:self.age forKey:@"age"]; }

- (id)initWithCoder:(NSCoder \*)aDecoder{ if (self=[super init]) {self.name= [aDecoder decodeObjectForKey:@"name"]; self.age=[aDecoder decodeObjectForKey:@"age"]; } return self; }

针对调用归档的对象：准备一个缓冲区NSMutableData \*data = [[NSMutableData alloc]init];；创建NSKeyArchiving对象，并和缓冲区绑定NSKeyedArchiver \*archiver=[[NSKeyedArchiver alloc]initForWritingWithMutableData: data];；将需要保存的数据进行编码 [archiver encodeObject:firstPerson forKey:@" firstPerson"]; [archiver encodeObject:secondPerson forKey:@"secondPerson"];

[archiver finishEncoding];；将编码完的数据进行归档[data writeToFile: self.archivingFilePath atomically:YES];。

针对调用解档的对象：准备一个缓冲区；创建NSArchving对象，绑定缓冲区；对归档的数据进行解码；开始解档操作。NSData \*unArchivingData=[NSData dataWithContentsOfFile:self.archivingFilePath];NSKeyedUnarchiver \*unArchier = [[NSKeyedUnarchiver alloc] initForReadingWithData:unArchivingData];Person \*firstPerson = [unArchier decodeObjectForKey:@"firstPerson"]; Person \*secondPerson = [unArchier decodeObjectForKey:@"secondPerson"];[unArchier finishDecoding];

NavicatPreMium11-1-12数据库软件

SQLite：是开源的嵌入式数据库；

SQL：增insert into<表名>(<列名1>,<列名2>,…) values(1,tonny,19)

删delete from <表名> where <列名>=””

改update <表名>set<列名1>=”” where <列名>=”” 查select \*from<表名>

创建表：creat table<表名> (<列名1>,<列名2>,…)

creat table person(id integer primary key, name text, age integer);

mac终端创建数据库：1:sqlite3 <数据库名字.db> 2:查找当前的所有数据库.databases; 3:create table person(id integer primary key, name text, age integer); 4:.tables; 5:insert into person (name, age) values(“tonny”,19) 6:select \*from person; 7:update person set age=20 where id=3; 8:delete \*from person where id=3;

IOS创建数据库: 左顶targets->Building Phases->Link Binary->libsQlite

sqlite3 \*db = NULL; int ret=sqlite3\_open([dbPath cStringUsingEncoding:NSUTF8StringEncoding], &db); if (ret != SQLITE\_OK) { NSLog(@"数据库创建失败,原因：%s",sqlite3\_errmsg(db)); } const char \*createSql = "create table people(id integer primary key,name text, age integer)"; const char \*insertSql = "insert into people(name,age)values('jonny',19)"; char \*errmsg = NULL; int execRet=sqlite3\_exec(db, createSql, NULL, NULL, &errmsg); if (execRet != SQLITE\_OK) { NSLog(@"失败原因：%s",errmsg); } execRet = sqlite3\_exec(db, insertSql,nil,nil,&errmsg); sqlite3\_free(errmsg); sqlite3\_close(db);

用xcode创建plist： 先从xcode创建property.list

找到plist所在文件的路径NSString \*plistPath = [[NSBundle mainBundle] pathForResource:@"student" ofType:@"plist"];

core data：从xcode创建coreDataModel、….NSManageMent Object SubClass

@dynamic不会生成也不能重写set和get方法

#import <CoreData/CoreData.h>

创建被管理对象上下文：NSManagedObjectContext \*context = [[NSManagedObjectContext alloc]init];

数据模型文件(.xcdatamodeld文件)->相当于数据库文件

NSManagedObjectModel(被管理对象模型)->相当于数据库中的所有表(数据模型中的所有Entity)

NSManagedObject(被管理对象)->相当于数据库中的一个表(一个Entity)

创建持久化存储协调器并绑定被管理对象模型：

NSURL \*modelURL = [[NSBundle mainBundle]URLForResource:@"Model" withExtension:@"momd"]; Model.xcdatamodeld编译后会变成momd

NSManagedObjectModel \*objectModel=[[NSManagedObjectModel alloc] initWithContentsOfURL:modelURL]; 创建被管理对象模型

NSPersistentStoreCoordinator \*storeCoordinator = [[NSPersistentStoreCoordinator alloc]initWithManagedObjectModel:objectModel];

使用持久化存储协调器来配置持久化相关的操作和sqlite所在的沙盒url：

NSString \*documentPath = [NSSearchPathForDirectoriesInDomains( NSDocumentDirectory, NSUserDomainMask, YES) lastObject]; NSString \*sqlitePath = [documentPath stringByAppendingPathComponent:@"Model.sqlite"]; NSURL \*sqliteURL = [NSURL fileURLWithPath:sqlitePath]; NSPersistentStore \*store = [storeCoordinator addPersistentStoreWithType:NSSQLiteStoreType configuration:nil URL:sqliteURL options:nil error:nil];

将持久化存储协调器和被管理上下文进行绑定：

[context setPersistentStoreCoordinator:storeCoordinator];

插入：首先返回一个给定Entity和被管理对象上下文的“空的”实体对象：

PersonModel \*person = [NSEntityDescription insertNewObjectForEntityForName:@"PersonModel" inManagedObjectContext:self.context]; ………

[self.context save:nil];

查询：创建NSFetchRequest对象，需要绑定Entity：

NSFetchRequest \*request = [NSFetchRequest fetchRequestWithEntityName:@" PersonModel"]; (谓词)NSPredicate \*predicate = [NSPredicate predicateWithFormat: @"name=%@",@"Bob"]; request.predicate = predicate;

NSArray \*persons=[self.context executeFetchRequest:request error:nil];

排序: NSSortDescriptor \*sortDes = [NSSortDescriptor sortDescriptorWithKey:@"height" ascending:YES]; request.sortDescriptors = @[sortDes];

删:for(PersonModel \*person in persons) {[self.context deleteObject:person];}

谓词：NSPredicate \*redicate = [NSPredicate alloc] initWithFormat

截图：command+shift+4

tableView与模型层结合：

协议#import <CoreData/CoreData.h> <NSFetchedResultsControllerDelegate>

上下文的懒加载（通过AppDelegate单例）：

-(NSManagedObjectContext\*)managedObjectContext{if (\_managedObjectContext) {AppDelegate \*delegate=[[UIApplication sharedApplication] delegate]; \_managedObjectContext = delegate.managedObjectContext;} return \_managedObjectContext; }

NSManagedObjectModel管理所有entity；NSManagedObjectContext保存数据到SQLite；NSFetchRequest相当于SQL语句；NSPredicate相当于SQL中where部分

coreData+TableView FMDB ORM(Object Relationship Mapping)

多线程技术：

1：Pthread（POSIX）Portable Operating System Interface，基于c的API，跨平台，需开发者手动创建、销毁线程。

建：pthread\_t pthread; 建子线程pthread\_create(&pthread, NULL, task, nil); 打印当前线程信息NSLog(@"%@",[NSThread currentThread]);

NSThread

需开发者自行管理线程生命周期、线程同步、加锁、解锁2:管理多个线程困难

创建：NSThread \*thread = [[NSThread alloc]initWithTarget:self selector:@ selector(download) object:nil]; object的主线程传给子线程的对象 thread.name = @"Download";线程名

启动：[thread start]; 判断是否主线程BOOL isMain = [NSThread isMainThread];

[NSTread exit]退出当前子线程正在执行的任务

[NSThread detachNewThreadSelector:@selector(download) toTarget:self withObject:nil]; 创建并启动子线程。

[self performSelectorInBackground:@selector(download) withObject:nil];系统在后台自动创建子线程，执行download中的方法。

[NSThread sleepForTimeInterval:5];当前线程暂停5秒再继续执行。

@syncronized(self){ 加解锁 } 耗费大量资源，不建议用

NSLock \*ticketLock;……..[self.ticketLock lock];……..[self.ticketLock unlock];

atomic: 在set方法中加锁，防止多个线程同时执行set方法

nonatomic: 不在set方法加锁，占用资源少。

下载图片：NSString \*str = @"http://img4.imgtn.bdimg.com/it/u= 2239797258,3441719612&fm=11&gp=0.jpg"; NSURL \*url = [NSURL URLWithString:str]; NSData \*imageData = [NSData dataWithContentsOfURL:url];

UIImage \*image = [UIImage imageWithData:imageData];

回到主线程[self performSelectorOnMainThread:@selector(downloadFinish:) withObject:image waitUntilDone:YES];

终端：thread list查看所有线程；breakpoint set -saleTicket Thread 8对线程8在-saleTicket方法中设置断点

breakpoint set -l 46 thread 7:对线程7的46行设置断点

临界区：线程中访问共享资源的代码段 临界资源：共享的资源 任务：线程需要执行的那段代码

gcd(Grand Central Dispatch)：为减少开发者的工作量(加解锁；手动创建线程)C

串行Serial Dispatch Queue子线程按顺序

并行Concurrent Dispatch Queue子线程并行

同步执行(Synchronous Dispatch):队列中的任务在主线程中执行,按顺序

异步执行(Asynchronous Dispatch):任务在子线程中执行

创建队列dispatch\_queue\_t queue=dispatch\_queue\_create("FirstSerialQueue", DISPATCH\_QUEUE\_SERIAL); //day8\_4GCD

执行子线程dispatch\_sync(queue, ^{ });

建立主队列dispatch\_queue\_t mainQueue = dispatch\_get\_main\_queue();

全局队列dispatch\_queue\_t globalQueue=dispatch\_get\_global\_queue (DISPATCH\_QUEUE\_PRIORITY\_DEFAULT, 0);参数1:优先级

[self performSelector:@selector(task) withObject:nil afterDelay:3];任务在子线程

延迟3秒执行, 其中dispatch\_time第二个参数单位是纳秒:

dispatch\_after(dispatch\_time(DISPATCH\_TIME\_NOW, 3E9 , dispatch\_get\_global \_queue(DISPATCH\_QUEUE\_PRIORITY\_DEFAULT, 0), ^{ NSLog(@"..."); });

确保一段代码只运行一次(即使在循环中)：static dispatch\_once\_t onceToken; dispatch\_once(&onceToken, ^{NSLog(@"..."); });可用于加锁和单例(双核…)等

单例类：始终返回同一个对象，对应的方法称为单例方法(once、加锁)

闭包：Smalltalk, Ruby, OC: Block; Lisp, Python, C++11:Lambda

JavaScript: Anonymous Function Swift: Closures

block做参数、返回值、局部变量、属性(copy)

在Block修饰外部局部变量时，需局部变量前有\_\_block修饰。在block使用self经常出现内存引用循环，解决办法：\_\_weak typeof(self) weakSelf = self;

NSOperationQueue：对GCD的封装

NSOperationQueue \*queue = [[NSOperationQueue alloc]init]; NSInvocationOperation \*operation1 = [[NSInvocationOperation alloc]initWithTarget:self selector:@selector(printPlusSignal) object:nil]; [queue addOperation:operation1];默认并行, 一旦add就执行。

[operation2 addDependency:operation1];线2等线1完了再执行。可用于串行

NSBlockOperation \*operation1 = [NSBlockOperation blockOperationWithBlock:^{ … }];

创建主线程：NSOperationQueue \*mainQueue = [NSOperationQueue mainQueue];

准备服务器：eclipse开发工具：依赖java运行环境（安装JDK）

apache tomcat(Web服务器，容器) firefox，firebug查看请求、应答

部署服务：TServer.zip解压到eclipse的workspace中

启动eclipse->左边空白右键->import->import->general->Existing…

->select root directory->browse->workspace->TServer->open->Finish

将tomcat服务器配置到eclipse中

跟着x一直->server->apache->Tomcat->Tomcat directory(browse)-> apache…(open)->finish

启动服务器：左边TServer->Run As->1 Run on Server->Tom…6.0->finish console显示Server startup in … ms说明服务器已启动

localhost:8080/TServer/

转换：编码NSData \*strData=[str dataUsingEncoding:NSUTF8StringEncoding];

解码NSString \*converted = [[NSString alloc]initWithData:strData encoding: NSUTF8StringEncoding]; 编解码的coding要一致

data和image： NSString \*imagePath = [[NSBundle mainBundle] pathForResource:@"channel06" ofType:@"png"]; NSData \*imageData=[NSData dataWithContentsOfFile:imagePath]; UIImage \*image = [UIImage imageWithData:imageData];

和服务器交互：

URL(uniform Resourse Locator)是一个字符串，可以在全世界范围内唯一地定位一个资源(如一张图片、一首歌、一部影片、一个网页…..)。

协议://主机地址/资源路径 协议代表不同资源地查找方式

主机地址：存放资源的那个计算机的IP或域名

常见协议：

http://超文本传输协议，用来访问远程网络上的资源 file:// 访问本地计算机上的资源 ftp:// mailto: 发邮件

NSString \*urlStr = self.textField.text; NSURL \*url = [NSURL URLWithString: urlStr]; 根据URL创建并发送请求对象: NSURLRequest \*request = [NSURLRequest requestWithURL:url]; NSData \*receivedData = [NSURLConnection sendSynchronousRequest:request returningResponse:nil error:nil];

使用webView显示从网络上获得的数据

[self.webView loadData:receivedData MIMEType:@"text/html" textEncodingName:@"utf-8" baseURL:nil]; 参数2:格式

或：[self.webView loadRequest:request];直接发请求

登陆到远程服务器

给服务器发请求[NSString stringWithFormat: @"http://localhost:8080/login? username=%@&pwd=%@",userName,passWord];其中username和pwd的写法取决于服务器端的标记

[NSURLConnection sendAsynchronousRequest:request queue:nil completionHandler:^(NSURLResponse \*response, NSData \*data, NSError \*connectionError) { }

转码(有中文)urlStr = [urlStr stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding];

< NSURLConnectionDataDelegate >

发请求的链接:NSURLConnection \*conn=[[NSURLConnection alloc] initWithRequest:request delegate:self startImmediately:NO]; 参数3决定立即发请求

[conn start];发请求

接受到服务器来的数据调用此方法: -(void)connection:(NSURLConnection \*)connection didReceiveData:(NSData \*)data{ }

服务器开始响应时调用：-(void)connection:(NSURLConnection \*)connection didReceiveResponse:(NSURLResponse \*)response{ }

接受完数据后调用：-(void)connectionDidFinishLoading:(NSURLConnection \*)connection{ NSLog(@"接受完数据"); }

NSURLSession在NSURLConnection(ios7后被淘汰)基础上加了新功能,操作文件

NSURLSessionConfiguration \*config = [NSURLSessionConfiguration defaultSessionConfiguration];

NSURLSession \*session = [NSURLSession sessionWithConfiguration:config];

使用session发请求NSURLSessionDataTask \*task= [session dataTaskWithRequest:request completionHandler:^(NSData \*data(返回的数据), NSURLResponse \*response(应答头), NSError \*error) { NSInteger responseCode = [(NSHTTPURLResponse \*)response statusCode];if (responseCode == 200) {

NSLog(@"返回成功: %@", [NSThread currentThread]); NSDictionary \*jsonDic = [NSJSONSerialization JSONObjectWithData:data options:0 error:nil];}] ;//jsonDic即是由data得到的信息,字典形式 [task resume];

或：[NSURLSession sharedSession] dataTaskWithRequest:request completionHandler:^(NSData \*data, NSURLResponse \*response, NSError \*error) {…}

各种task:获取数据NSURLSessionDataTask;下载文件NSURLSessionDownloadTask; 上传文件NSURLSessionUploadTask

获取数据是将数据下载到内存，下载文件是下载到sandbox中

NSURLSessionDownloadTask \*task=[session downloadTaskWithRequest: request completionHandler:^(NSURL \*location, NSURLResponse \*response, NSError \*error) {…. }] location指下载到哪儿

NSHTTPURLResponse \*httpResponse = (NSHTTPURLResponse \*)response;

httpResponse.statusCode; block中的对象针对子类，这里是为得到.statusCode

保存到一个位置[[NSFileManager defaultManager]moveItemAtPath:[location path] toPath:destPath error:nil];

显示下载进度,用代理

<NSURLSessionDataDelegate,NSURLSessionDownloadDelegate>

NSURLSession \*session = [NSURLSession sessionWithConfiguration: [NSURLSessionConfiguration defaultSessionConfiguration] delegate:self delegateQueue:nil];

实现方法:

-(void)URLSession:(NSURLSession \*)session downloadTask:( NSURLSessionDownloadTask \*)downloadTask didFinishDownloadingToURL:(NSURL \*)location{ …下载完成后拷入 }

控制进度条-(void)URLSession:(NSURLSession \*)session downloadTask: (NSURLSessionDownloadTask \*)downloadTask didWriteData:(int64\_t) bytesWritten totalBytesWritten:(int64\_t)totalBytesWritten totalBytesExpectedToWrite:(int64\_t)totalBytesExpectedToWrite{double progress=1.0\* totalBytesWritten/totalBytesExpectedToWrite; dispatch\_async(dispatch\_get\_main \_queue(), ^{ [self.downloadProgress setProgress:progress animated:YES];});}

文件偏移量发生变化后的回调方法:-(void)URLSession:(NSURLSession \*)session downloadTask:(NSURLSessionDownloadTask \*)downloadTask didResumeAtOffset:(int64\_t)fileOffset expectedTotalBytes:(int64\_t) expectedTotalBytes{ 一般可不填 }

断点续传：

暂停并纪录任务中已下载的数据[self.task cancelByProducingResumeData:^(NSData \*resumeData) { self.resumeData = resumeData; }];………

从断点继续下载：self.task = [self.session downloadTaskWithResumeData:self.resumeData]; [self.task resume];

HTTP(Hypertext Transfer Protocol)协议

OSI七层网络模型: 物理层-链路层-网络层-传输层-会话层-表示层-应用层

TCP/IP协议簇 事实上的工业标准 网络接口层(网卡等数据规范)-网际网层(IP)-传输层((Socket通信)TCP(面向链接、安全的传输协议)、UDP(无链接、不安全的传输协议、可能丢失数据、传输效率高))-应用层(HTTP、FTP、SMTP)

IOS开发中:Socket链接一般叫长链接(对服务器的负担重)、HTTP链接叫短链接

HTTP协议优点:1:简单 2:快速 3:服务器比较好实现，负担轻 4:灵活，传输任意类型的数据

HTTP通信方式：1:请求 2:应答

www.tarena.com.cn:80(80是端口号) ->互联网域名解析->IP地址(118.145.21.176:80)

请求包含的内容：1:请求行:Get login?username=123&pwd=123; 2:请求头:host: localhost:8080//服务器主机 user-Agent:iOS客户端类型 Accept:text/html//客户端所能接受的数据类型 Accept-Language:zh-CN客户端的语言环境

3:请求体:客户端发给服务器的具体数据

响应包含的内容：1:状态行:HTTP协议版本,状态码,状态信息 2:响应头server、Content-Type:json,html,xml,image服务器返回的数据的类型，Content-Length: 123//返回的数据长度 3:响应体服务器实际给客户端的数据

一些状态码的含义：200 OK 请求成功；400 Bad Request请求本身错误，服务器无法识别；404NotFound请求的资源不存在；500 Server Error服务器错误。

GET和POST请求的区别：GET请求会将发给服务器的所有参数全部用xxx=xxx& xxx=xxx的方法缀到URL后面，URL的长度有限，发的数据太多不适合用GET；POST将发送给服务器的所有参数全部放到请求体中，而不是URL后面。

如何发POST请求：

NSMutableURLRequest \*request = [NSMutableURLRequest requestWithURL:url]; request.HTTPMethod = @"POST"; request.HTTPBody = [@"username=123& pwd=123" dataUsingEncoding:NSUTF8StringEncoding];

其他request属性：

request.timeoutInterval = 5;请求超时就不需要服务器的返回数据

告诉服务器客户端的类型：[request setValue:@"Android5.0" forHTTPHeaderField:@"User-Agent"];

httpResponse.statusCode状态码 .MIMEType类型 .expectedContentLength响应体中数据的长度

三次握手协议，服务端没接收到就再发一次，位于TCP中

json解析：NSJSONSerialization

NSDictionary \*dict=[NSJSONSerialization JSONObjectWithData:data options: NSJSONReadingMutableContainers error:nil]; options参数决定返回类型

导入播放器头文件@import MediaPlayer; iOS7开始新的导入方式

MPMoviePlayerViewController \*player = [[MPMoviePlayerViewController alloc]initWithContentURL:[NSURL URLWithString:urlStr]];

[self presentMoviePlayerViewControllerAnimated:player];

选择VC支持的方向：左上角->Deployment Info->Device Orientation

或仅针对某个VC中-(NSUInteger)supportedInterfaceOrientations{ return UIInterfaceOrientationMaskLandscape; }

swift中：UIDevice.current.setValue(UIInterfaceOrientation.landscapeLeft.rawValue, forKey: "orientation")

?移除通知,以防退到后台时自动销毁播放器[[NSNotificationCenter defaultCenter]removeObserver:self name: UIApplicationDidEnterBackgroundNotification object:nil];

播放音乐的框架

AVAudioPlayer

self.player = [[AVAudioPlayer alloc]initWithContentsOfURL:url error:nil];

[self.player play]; 点击按钮后会释放，所以应以属性存在

AFNetWorking第三方框架 一般用于：下载/上传大文件

AFHTTPRequestOperationManager \*manager = [AFHTTPRequestOperationManager manager]; [manager GET:@"http://localhost:8080/TServer/video" parameters:nil success:^(AFHTTPRequestOperation \*operation, id responseObject) { NSLog(@"sucess%@",responseObject[@"videos"]); } failure:^(AFHTTPRequestOperation \*operation, NSError \*error) { NSLog(@"failure%@",error.userInfo); }]; 发POST只需改GET

上传文件只能用POST:

[manager POST:@"http://localhost:8080/TServer/upload" parameters:params constructingBodyWithBlock:^(id<AFMultipartFormData> formData) {NSString \*file = [[NSBundle mainBundle]pathForResource:@"battle.zip" ofType:nil]; NSData \*data = [NSData dataWithContentsOfFile:file]; [formData appendPartWithFileData:data name:@"file" fileName:@"upload.zip" mimeType:@"zip"];} success:^(AFHTTPRequestOperation \*operation, id responseObject) {…} failure:^(AFHTTPRequestOperation \*operation, NSError \*error) {…}]; ?Server

分段下载问题：

NSMutableURLRequest \*request = [NSMutableURLRequest requestWithURL:nil];

[request setValue:@"bytes = 101-200" forHTTPHeaderField:@"Range"];

网络状态：

AFNetworkReachabilityManager \*manager = [AFNetworkReachabilityManager sharedManager]; [manager setReachabilityStatusChangeBlock:^( AFNetworkReachabilityStatus status) { switch (status) {case AFNetworkReachabilityStatusNotReachable: NSLog(@"unkown"); break; case AFNetworkReachabilityStatusReachableViaWiFi: NSLog(@"Wifi"); case AFNetworkReachabilityStatusReachableViaWWAN: NSLog(@"3G"); default:break; }

}]; [manager startMonitoring];

[manager stopMonitoring];停止监控 [manager isReachableViaWiFi];是否wifi

XML(Extenstion Mark Language) 是格式化的文本文件，主要用于保存数据

1)第一行内容：<?xml version=”1.0” encoding=”utf-8” ?>

2)XML文件中都是标签，所有的开始标签都要有结束标签对应<name>Daniel</name>

3)标签组成的区域叫元素(Element)，元素可以嵌套，但不能重叠

4)一个XML文件中有且只有一个根元素

5)属性值用””括起来

iOS解析方式

1) NSXMLParser类 苹果官方类 SAX解析:读一句，解析一句

2) libxml2 c语言的开源解析器 可以进行SAX解析，也可进行DOM解析（一次性将XML文件读入内存,从内存读取,可随机读取XML中内容,耗内存，小文件）

3) 第三方框架:TBXML:(适合小文件) GDataXML KissXML

<NSXMLParserDelegate>

NSXMLParser \*parser = [[NSXMLParser alloc]initWithData:data];parser.delegate = self; [parser parse];

-(void)parser:(NSXMLParser \*)parser didStartElement:(NSString \*) elementName namespaceURI:(NSString \*)namespaceURI qualifiedName:(NSString \*)qName attributes:(NSDictionary \*)attributeDict{…}参数1:元素名；2:命名空间；3:qName；4:元素的属性

解析时找到内容调用此方法-(void)parser:(NSXMLParser \*)parser foundCharacters:(NSString \*)string{…}

字符串转数字[string integerValue]

加入毛玻璃效果：self.blurEffectiveView = [[UIVisualEffectView alloc]initWithEffect:[UIBlurEffect effectWithStyle:UIBlurEffectStyleExtraLight]]; [self.view insertSubview: self.blurEffectiveView atIndex:1];

去横线self.tableView.separatorStyle = UITableViewCellSeparatorStyleNone;

cell文本居中cell.textLabel.textAlignment = NSTextAlignmentCenter;

CADisplayLink计时器，比NSTimer更精确，更新频率更高

self.lrcTimer = [CADisplayLink displayLinkWithTarget:self selector:@selector (updateLrc)]; [self.lrcTimer addToRunLoop:[NSRunLoop mainRunLoop] forMode:NSDefaultRunLoopMode];

滚动tableView

NSIndexPath \*p = [NSIndexPath indexPathForRow:idx inSection:0];

UITableViewCell \*cell = [self.tableView cellForRowAtIndexPath:p];

[self.tableView reloadRowsAtIndexPaths:@[p] withRowAnimation: UITableViewRowAnimationAutomatic];

[self.tableView scrollToRowAtIndexPath:p atScrollPosition: UITableViewScrollPositionTop//? animated:YES];

定位Core Location

@import CoreLocation;

类CLLocationManager <CLLocationManagerDelegate>

[CLLocationManager locationServicesEnabled]是否启动定位服务

未授权则开启授权：if ([CLLocationManager authorizationStatus]== kCLAuthorizationStatusNotDetermined) { [\_locationManager requestWhenInUseAuthorization]; }

\_locationManager.delegate = self; \_locationManager.desiredAccuracy = kCLLocationAccuracyBest; \_locationManager.distanceFilter = 1.0;//频率

[\_locationManager startUpdatingLocation];

-(void)locationManager:(CLLocationManager \*)manager didUpdateLocations:(NSArray \*)locations{ NSLog(@"%@",locations); CLLocation \*location = [locations firstObject]; CLLocationCoordinate2D coord = location.coordinate;

NSLog(@"经度%f,纬度%f,海拔%f,航向%f,速度%f",coord.longitude, coord.latitude,location.altitude,location.course,location.speed); }

CLGeocoder：地理编码

[\_geocoder geocodeAddressString:@"北京潘家园" completionHandler: ^(NSArray \*placemarks, NSError \*error) { CLPlacemark \*placemark = [placemarks firstObject]; CLLocation \*location = placemark.location; NSLog (@"%@",placemark.addressDictionary);}]；

反编码，由坐标得到地址

CLLocation \*location = [[CLLocation alloc]initWithLatitude:39.54 longitude: 116.28]; [\_geocoder reverseGeocodeLocation:location completionHandler:^(NSArray \*placemarks, NSError \*error) { CLPlacemark \*placemark = [placemarks firstObject]; self.displayLabel.text = [placemark.addressDictionary description]; }];

地图@import MapKit; @import CoreLocation;

MKMapView \*\_mapView;………….

\_mapView = [[MKMapView alloc]initWithFrame:self.view.frame]; [self.view addSubview: \_mapView];………….

用户位置追踪\_mapView.userTrackingMode = MKUserTrackingModeFollow;

地图类型\_mapView.mapType = MKMapTypeStandard;

\_mapView.delegate = self; <MKMapViewDelegate>

-(void)mapView:(MKMapView \*)mapView didUpdateUserLocation:( MKUserLocation \*)userLocation{//获取当前位置的地理信息CLLocation \*location = userLocation.location; [\_geocoder reverseGeocodeLocation:location completionHandler:^(NSArray \*placemarks, NSError \*error) { CLPlacemark \*placemark = [placemarks firstObject]; NSLog(@"%@", placemark.addressDictionary); }];

//设置地图显示的范围MKCoordinateSpan span = MKCoordinateSpanMake(0.05, 0.05); MKCoordinateRegion region = MKCoordinateRegionMake( location.coordinate, span); [\_mapView setRegion:region animated:NO]; }

创建大头针类：@import MapKit; 遵守<MKAnnotation> 三个属性

@property (nonatomic, readonly) CLLocationCoordinate2D coordinate;必须

@optional

@property (nonatomic, readonly, copy) NSString \*title;

@property (nonatomic, readonly, copy) NSString \*subtitle;

@end

…

添加大头针annotation.coordinate = CLLocationCoordinate2DMake(39.875, 116.456); annotation.title = @"达内";annotation.subtitle = @"移动互联网学院";

[\_mapView addAnnotation:annotation];

设置大头针样式:该方法会处理所有大头针，需要区分，当用上自定义的属性时要强转-(MKAnnotationView \*)mapView:(MKMapView \*)mapView viewForAnnotation:(id <MKAnnotation>)annotation{…deque方法创建…设置是否可点和Callout图片：annotationView.canShowCallout=YES; annotationView.leftCalloutAccessoryView = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@" icon\_classify\_cafe"]];绑定annotationView.annotation = annotation;}

两地点间画线:@import CoreLocation; @import MapKit; <MKMapViewDelegate>

创建请求MKDirectionsRequest \*request = [MKDirectionsRequest new];

起点MKPlacemark \*mkStartPlace = [[MKPlacemark alloc]initWithPlacemark: (CLPlaceMark \*)startPlace]; request.source = [[MKMapItem alloc] initWithPlacemark:mkStartPlace]; 终点……

MKDirections \*directions = [[MKDirections alloc]initWithRequest:request];

计算路线[directions calculateDirectionsWithCompletionHandler:^( MKDirectionsResponse \*response, NSError \*error) {for (MKRoute \*route in response. routes) { [\_mapView addOverlay:route.polyline]; } }];

画线设置-(MKOverlayRenderer \*)mapView:(MKMapView \*)mapView rendererForOverlay:(id<MKOverlay>)overlay { MKPolylineRenderer \*render = [[MKPolylineRenderer alloc]initWithPolyline:overlay]; render.lineWidth = 4; render.strokeColor = [UIColor redColor]; return render; }

两个位置s：MKMapItem \*mapItem1 = [[MKMapItem alloc]initWithPlacemark:mkPlacemark1];… NSDictionary \*options = @{ MKLaunchOptionsMapTypeKey:@(MKMapTypeStandard), MKLaunchOptionsDirectionsModeKey : MKLaunchOptionsDirectionsModeDriving}; [MKMapItem openMapsWithItems:@[mapItem1, mapItem2] launchOptions:options];

isd1503 day62-63天气项目

SDK(software development kit)软件开发包: 包括代码样例等

[category setValuesForKeysWithDictionary:categoryDic];通过key对应属性

得到自定义View： [[[NSBundle mainBundle] loadNibNamed:@"TRNavLeftView" owner:nil options:nil] lastObject]; xib上可以有很多controller，所以last…

UIBarButtonItem \*sortItem = [[UIBarButtonItem alloc] initWithCustomView: sortView]; 由自定义view得到UIBarButtonItem

[[UIPopoverController alloc] initWithContentViewController: categoryViewController]; 创建UIPopoverController，与categoryViewController绑定

[self.categoryPopController presentPopoverFromRect:self.categoryView.bounds inView:self.categoryView permittedArrowDirections: UIPopoverArrowDirectionAny animated:YES]; (在事件方法中)弹出pop

获取用户点击左边表视图的行NSInteger selectedRow = [self.mainTableView indexPathForSelectedRow].row;

设置VC大小viewController.preferredContentSize = CGSizeMake(100, 200);

返回section的title: -(NSString \*)tableView:(UITableView \*)tableView titleForHeaderInSection:(NSInteger)section { … }

以sheet形式弹出VC：citiesViewController.modalPresentationStyle = UIModalPresentationFormSheet;

返回tableView右侧索引：-(NSArray \*) sectionIndexTitlesForTableView:(UITableView \*)tableView

三方库：SDWebImage #import "UIImageView+WebCache.h"

[self.imageView sd\_setImageWithURL:[NSURL URLWithString:deal.image\_url] placeholderImage:[UIImage imageNamed:@"placeholder\_deal"]];

git? KVC？KVO？APP上架流程？ Socket!!!

百度搜“常用三方库”

三方库：BottomFresh 关于refresh day67 代码 day0630

#import "UIScrollView+BottomRefreshControl.h"

UIRefreshControl \*refreshControl = [UIRefreshControl new]; refreshControl.triggerVerticalOffset = 100; refreshControl.attributedTitle = [[NSAttributedString alloc] initWithString:@"正在等待加载更多团购订单"];[refreshControl addTarget:self action:@selector(loadMoreDeals) forControlEvents: UIControl EventValueChanged];self.collectionView.bottomRefreshControl = refreshControl;

……结束：[self.collectionView.bottomRefreshControl endRefreshing];

版本控制：作用：1.保存 2.查看不同版本之间的不同

git：开源的分布式VCS(Version Control System)

xcode在sourceControl中：提交Commit… 纪录history…

回退：View->Version Editor->……

github: www.github.com 账号:lyluoyuan hnb314159

newrepository->(与xcode关联)复制HTTP ->(xcode)preference->+-> add repository->粘贴-> git的用户密码

Source Control->第一项->Configure->Remotes：Name:工程名字、Address:链接地址，然后就可以push了

以后下代码：Check out an existing project->….

Push代码：先Source Control->commit->Source Control->push

创建和切换分枝：Source Control第二项 合并：merge

真机测试 day68 pm isd1503

1.开发者账号：member center

2.淘宝账号

3.xcode7(APP ID) icloud邮箱 www.icloud.com

1.mac不是黑苹果 2.app 3.device 4. 1,2,3->profile file 5.确保设备ios版本更低

launchPad->其他->钥匙->钥匙串访问->证书助理->从证书颁发机构请求证书->填写邮箱，存储到磁盘->存储

developer.apple.com->Certificates,identifier&Profiles-> Certificates-> development->＋->IOS App Development->…continue…->Upload CSR file：刚才的证书请求->Download:ios\_development.cer双击添加,钥匙串中的iPhone Developer即是

回到Certificates,identifier&Profiles->App IDs->添加->(Explict App ID:一个explicit app id必须对应一个xcode工程；开发多个xcode项目，就要申请相应个数的id；Wildcard App ID:多个应用可用一个ID: 例com.tarena.test.\*)->Devices->All(已注册的合法的设备)->＋->其中的UDID见(xcode->window->identifier) Provisioning profiles->All->＋->Development->填刚才申请的App ID->勾选…continue->勾选设备->填写..genarate->Download->

找到下载的后缀为.mobileprovision->复制到刚才xcode创建的工程中双击(会自动添加到build setting下的code signing identity下debug下…)->xcode下的Info下Custom…->Bundle identifier: 填写com.tarena.test.\*

真机调试:xcode->preference->＋,App ID

cocoa pods：管理第三方库的工具：自动支持下载哪些cocoa pods的第三方库的最新版本，判定github上第三方库是否支持cocopods (1)查看README.md (2)命令行方式:pod search AFNetworking

gem sources --add https://gems.ruby-china.org/ --remove https://rubygems.org/

gem sources -l

(1)安装cocoa pods: 终端输入：sudo gem install cocoapods ??????

(2)删除被屏蔽的源: gem sources—remove https://rubygems.org/

gem sources –a http://ruby.taobao.org/

gem sources -l

例子：1）查看github上的第三方库是否支持cocoa pods：输入命令：pod search AFNetworking

2）使用xcode创建一个工程；或使用已有的工程；

3）终端输入命令：cd xxxx/xxx/xxx（cd后面是xcode工程的绝对路径）

4）终端输入命令，创建Podfile文件

5）开始编辑Podfile文件，将第三方库README.md文件中关于podfile的配置信息，粘贴过来

6）输入命令：pod install（开始下载第三方库）

7）备注：如果出现第三方库倒入不成功的情况，首先输入命令：pod update；然后再输入pod install

8）此时进入刚才xcode工程的目录，注意此时应该点击xxxx.xcworkspace结尾的文件；而不是.xcodepro结尾的文件

补充：

FMDB框架：xcode工程中导入libsqlite3.0.dylib和libsqlite3.dylib：左顶targets->Building Phases->Link Binary->libsQlite

拷贝fmdb文件夹

NSString \*doc = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) lastObject]; NSString \*fileName = [doc stringByAppendingPathComponent:@"student.sqlite"]; FMDatabase \*db = [FMDatabase databaseWithPath:fileName];

[db open]; 创表BOOL result=[db executeUpdate:@"CREATE TABLE IF NOT EXISTS t\_student (id integer PRIMARY KEY AUTOINCREMENT, name text, age integer);"];

插[self.db executeUpdate:@"INSERT INTO t\_student (name, age) VALUES (?, ?);", name, @(arc4random\_uniform(40))];除了查，都是调用该方法

其他sqlite语句：删除整个表@"DROP TABLE IF EXISTS t\_student;"

按照某一列进行升序排序：select \* from people where name='shirley' order by height asc; 更新表中的某条记录：update people set name='changedName' where id=1; 删除表中的某条记录：delete from people where id=1;

查FMResultSet \*resultSet = [self.db executeQuery:@"SELECT \* FROM t\_student"]; int ID = [resultSet intForColumn:@"id"]; NSString \*name = [resultSet stringForColumn:@"name"]; int age = [resultSet intForColumn:@"age"];

键值编码(无需set和get方法，即使私有也可访问)：KVC ：协议：<NSKeyValueCoding> NSObject实现了这个协议

写：[person1 setValue:@28 forKey:@"age"]; 读：[person1 valueForKey:@"age"]

写：[person1 setValue:@100000000.0 forKeyPath:@"account.balance"];

读：[person1 valueForKeyPath:@"account.balance"] floatValue]

KVC机理：优先调用setA/getA方法，然后\_a，a，setValue:forUndefinedKey：/ valueforUndefinedKey:

KVO(键值监听)：NSKeyValueObServing协议，NSObject已实现

添加对account的监听-(void)setAccount:(Account \*)account{\_account=account; [self.account addObserver:self forKeyPath:@"balance" options: NSKeyValueObservingOptionNew context:nil]; }

重写监听方法-(void)observeValueForKeyPath:(NSString \*)keyPath ofObject: (id)object change:(NSDictionary \*)change context:(void \*)context{if( [keyPath isEqualToString:@"balance"]){NSLog ( @"keyPath=%@,object =%@ ,newValue=%.2f,context=%@",keyPath,object,[[change objectForKey:@"new"] floatValue],context); } }

重写销毁方法：-(void)dealloc{ [self.account removeObserver:self forKeyPath:@"balance"];//移除监听//[super dealloc];//注意启用了ARC，此处不需要调用}

操作系统版本：NSString \* systeam = [UIDevice currentDevice].systemVersion; float number = [systeam floatValue];

二次授权

社交：SystemConfiguration.framework UMSocial\_Sdk\_4.2.3

[UMSocialSnsService presentSnsIconSheetView:self

appKey:@"55d585cf67e58e5cca004bd1" shareText:@"gfgfdgfdgdsgfds" shareImage:[UIImage imageNamed:@"icon40.png"] shareToSnsNames:[NSArray arrayWithObjects:UMShareToSina, UMShareToTencent, UMShareToRenren,UMShareToWechatSession,UMShareToWechatTimeline,nil] delegate:nil];

SocialFrameWork？

静态库：不共享，.a和.framework形式

OC: new project->Cocoa Touch Static Library->build->products中的.a文件

Swift: new project->Cocoa Touch Framework->(需要暴露的类设置public)->edit scheme->build Configuration=Release->run ，使用:将framework拖入目标工程->General->Embedded Binaries(添加)->Build Phases->Copy Bundle Resources(添加), 如果还报运行错误，Build Phases->Link Binary With Libraries改成optional

动态库：共享，.dylib和.framework形式，app不能上架

蓝牙：

ExternalAccessor.framework

CoreBluetooth.framework可用于第三方连接，必须支持蓝牙4.0，也叫BLE(Bluetooth Low Energy低功耗) 测试麻烦，需要至少2台是蓝牙4.0设备。核心结构：中心设备＋外围设备

步骤：1建立中心设备 2扫描外设 3

GameKit.framework:前提:两台苹果设备

@Import GameKit

<GKPeerPickerControllerDelegate,UINavigationControllerDelegate, UIImagePickerControllerDelegate>

创建GKPeerPickerController \*peerPC = [[GKPeerPickerController alloc]init]; peerPC.delegate = self; [peerPC show];

连接成功执行-(void)peerPickerController:(GKPeerPickerController \*)picker didConnectPeer:(NSString \*)peerID toSession:(GKSession \*)session { NSLog(@"连接成功:%@",peerID); self.session = session; [self.session setDataReceiveHandler:self withContext:nil]; [picker dismiss]; }

session接收到传来的数据执行- (void) receiveData:(NSData \*)data fromPeer:(NSString \*)peer inSession: (GKSession \*)session context:(void \*)context { self.imageView.image = [UIImage imageWithData:data]; }

将图片压缩成二进制数据NSData \*data = UIImagePNGRepresentation(image);

基于会话,发送数据 [self.session sendDataToAllPeers:data withDataMode:GKSendDataReliable error:nil];

二维码、摇一摇

MKNetworkKit框架 https://github.com/MugunthKumar/MKNetworkKit

添加：SystemConfiguration.framework，CFNetwork.framework，Security.framework和ImageIO.framework

两个主类：MKNetworkOperation和MKNetworkEngine

console调试命令

po 左侧 description

cell自适应高度用到的：fittingSize = [cell.contentView systemLayoutSizeFittingSize:UILayoutFittingCompressedSize];

三方和工具：

https://github.com/ibireme/YYKit YYKit，关键是富文本：YYText 图片异步加载：YYWebImage

注: 参照demo，不能let yyLabel = YYLabel(frame: …) 要yyLabel.frame = CGRectMake(100, 50, 200, 100) （不对齐问题：记得text.yy\_font=font） 暂不支持storyBoard(2015/11/16)?(可通过布局好的替代frame)

注: 自适应大小：let layout = YYTextLayout(containerSize: CGSizeMake(200, 999), text: text) yyLabel.frame.size = layout.textBoundingSize

其实http://www.itnose.net/detail/6177538.html中有更原生的富文本,但不支持控件文混排

Snapkit 方便的纯代码布局

PureLayout https://github.com/PureLayout/PureLayout功能更强大的纯代码布局： view=UIView() let view = UIView.newAutoLayoutView(); view.setNeedsUpdateConstraints()； 布局代码写在updateViewConstraints, setNeedsUpdateConstraints和layoutIfNeeded都会调用updateViewConstraints； [redView, yellowView].autoSetViewsDimension(.Height, toSize: 50.0) ；(views as NSArray).autoMatchViewsDimension(.Width)自适应宽度； view.autoPinEdge(.Left, toEdge: .Right, ofView: previousView)； super.updateViewConstraints()

水平中心：views.autoSetViewsDimension(.Height, toSize: 40.0) ；views.autoDistributeViewsAlongAxis(.Horizontal, alignedTo: .Horizontal, withFixedSpacing: 10.0, insetSpacing: true, matchedSizes: true) ，self.redView.autoAlignAxisToSuperviewAxis(.Horizontal)

约束不够时：NSLayoutConstraint.autoSetPriority(UILayoutPriorityDefaultHigh) { self.blueView.autoSetDimensionsToSize(CGSize(width: 10000.0, height: 10000.0)) } //???

关联约束：self.blueViewHeightConstraint = [self.blueView autoSetDimension:ALDimensionHeight toSize:kBlueViewInitialHeight];

设置并返回约束：NSArray \*constraints = [[views autoSetViewsDimension:ALDimensionWidth toSize:60.0] autoIdentifyConstraints:@"Set Width of All Views"];

快速改变布局：[self.verticalLayoutConstraints autoRemoveConstraints];

[self.horizontalLayoutConstraints autoInstallConstraints];

但暂只支持UIVIewController中的布局

https://github.com/jessesquires/JSQMessagesViewController.git JSQMessagesViewController气泡聊天界面

https://github.com/onevcat/VVDocumenter-Xcode 快速注释，含详细参数说明

Carthage https://github.com/Carthage/Carthage 类似cocapods

Carlos 缓存 https://github.com/WeltN24/Carlos //page

瀑布流布局,用法似UITableView :https://github.com/1000Memories/TMQuiltView

FlowSlideMenu https://github.com/MatrixHero/FlowSlideMenu 滑页回弹效果，类似MMDrawerControler

https://github.com/jamy0801/LGWeChatKit swift2.0 高仿微信界面 其中左右滑cell可用 MGSwipeTableCell

三个以上drawer https://github.com/dekatotoro/SlideMenuControllerSwift 左边可选

ESJsonFormat-Xcode https://github.com/EnjoySR/ESJsonFormat-Xcode 将Json格式化输出为模型的属性，目前不支持驼峰命名

MJExtension https://github.com/CoderMJLee/MJExtension 可与ESJsonFormat-Xcode结合，用于json转model

eg treeModel.append(TreeModel.mj\_objectWithKeyValues(d.description))，如果转下划线，在Model加 override class func mj\_replacedKeyFromPropertyName121(propertyName: String!) -> String! {return propertyName.mj\_underlineFromCamel() }最好写个BaseModel

GPUImage : 多种图片模糊效果

Carousel： 图片旋转、排列效果 https://github.com/nicklockwood/iCarousel

用法类似tableview。 属性perspective：(旋转)展示范围-0.01~0; contentOffset item中心偏移位置; viewpointOffset当前中心偏移位置; decelerationRate灵活度; vertical垂直轮播; clipsToBounds; autoscroll自动匀速轮播; CoverFlow(type)折纸片效果

Xmpp: pod 'Magnet-XMPPFramework', '~> 3.6.10'不是3.6.13 桥文件中: #import "XMPP.h" http://blog.csdn.net/kangx6/article/details/7744304

zm004demac-mini-2.local

ejabberd: 安装好后：应用程序-bin-start-admin interface-要选记住钥匙串 用户名@主机名 密码-

adium注册：添加XMPP(Jabber) 刚才ejabberd 建立的账号,记得@主机名-选项：连接服务器localhost

客户端demo: 服务器也是localhost

adium中倒着看发消息，除非消息第一行显示为下部的联系人。 在didReceiveMessage中if let msgo = message.elementForName("body"){let msg = msgo.stringValue() …} 因为消息在编辑还未发出时就会调用该方法

发送消息：let body = DDXMLElement.elementWithName("body") as! DDXMLElement， let mes = DDXMLElement.elementWithName("message") as! DDXMLElement 需要加as…

带动画的折线图和柱状图表UUChartView https://github.com/ZhipingYang/UUChartView

LTMorphingLabel多种文字炫酷效果的label，简单易用，swift写的。https://github.com/lexrus/LTMorphingLabel 设置label为LTMorphingLabel，morphingEffect，label.text = “test”

原理:计算原text和现text，定时器cadisplaylink(每个时刻setneeddisplay)动画控制每个字符的偏移(偏移体现在drawTextInRect(当setNeedsDisplay会调用),每个字符用s.drawInRect)

用到了粒子系统(火/砸动画也是用计时器改变每个字符emitterLayer位置)

粒子系统:CAEmitterLayer http://blog.csdn.net/yongyinmg/article/details/38946311

emitterLayer.emitterCells = [emitterCells] addSublayer开始动画

发射方向：setValue: forKeyPath: emitterCells.triangle.emissionLongitude

MGSwipeTableCell: tableViewCell加左右滑出现各种编辑类目。设置leftButtons(MGSwipeButton)和rightButtons就OK

FoldingTabBar可折叠tab bar和Tab Bar Controller： https://github.com/Yalantis/FoldingTabBar.iOS

Item与vc一一对应，YALTabBarItem中的left和right表示点击该item后的tabBar显示的左右item

DLSlideView顶部滑动分页 https://github.com/agdsdl/DLSlideView 与JYSlideSegmentController相比多了对每个vc的titile的image的支持且是view不是Controller

MMProgressHUD：集大成的HUD，可自定义。https://github.com/mutualmobile/MMProgressHUD MMProgressHUD.sharedHUD().overlayMode = .Gradient MMProgressHUD.setPresentationStyle(.Fade) MMProgressHUD.showWithTitle("title", status: "dd") MMProgressHUD.showWithTitle("long", status: "hold", confirmationMessage: "cancel?") { () -> Void in //超时取消} MMProgressHUD.showWithTitle("title", status: "dd", images:images)//自定义刷新动画 MMProgressHUD.dismiss() 若要改变hud背景用extension覆盖configureInitialDisplayAttributes方法或改原代码 progressViewContainer是紧接indicator的小方框 改变背景：在MMProgressHUD.m中，最后一条代码改成self.alpha = 0.1; self.backgroundColor = [UIColor greenColor]; [\_window addSubview:self.hud];

EasyTipView:提示框 https://github.com/teodorpatras/EasyTipView 如果点击同一个button想在出现和收起之间切换，最好对该button封装Bool属性

DCPathButton围绕button的扇形排列小图标 https://github.com/Tangdixi/DCPathButton 关键帧与UIBeziapath结合 使用：参照swift Demo，代码不多

SphereMenu与DCPathButton类似的圆形排列小图标，带动力回弹效果https://github.com/itouch2/SphereMenu 代码不多 原理:UISnapBehavior

KYGooeyMenu带粘性排列小图标https://github.com/KittenYang/KYGooeyMenu

同上

Twinkle为字体加钻石闪耀效果，swift。https://github.com/piemonte/Twinkle

粒子系统，通过改变emitterLayer的位置变化，透明度变化。view的扩展方法twinkle()

FSCalendar 日历https://github.com/WenchaoIOS/FSCalendar

JTCalendar日历,带中文日历https://github.com/jonathantribouharet/JTCalendar 注：需真机才能显示中文 BuildDayView在prepareDayView之前调用 显示的日期范围应在设置menuView或contentView之前 JTVerticalCalendar没有对应的垂直menuView

minDate = calendarManager.dateHelper.addToDate(todayDate, months: -5)

maxDate = calendarManager.dateHelper.addToDate(todayDate, months: 5)

PDTSimpleCalendar据说是最棒的日历组件https://github.com/jivesoftware/PDTSimpleCalendar/issues

AKPickerView-Swift https://github.com/Akkyie/AKPickerView-Swift 横向pickerView

iOS-RatingBar 星星评分控件 https://github.com/saiwu-bigkoo/iOS-RatingBar

原理:通过点击的坐标覆盖

RKNotificationHub 给UIView加badge，比BBBadgeBarButtonItem多个加无数字红点功能(hidCount),且可用在UIView和UIBarButton 位置: moveCircleByX:Y: 大小:scaleCircleSizeBy

WZLBadge横竖屏、动态badge，https://github.com/weng1250/WZLBadge

UIView的扩展方法: showBadgeWithStyle: value:animationType 可自定义badge: btn.badge 用法最广，可UIView和UIBarButton以及UITabBarItem

HACursor类似网易新闻，标题栏目和下面的子页面https://github.com/HAHAKea/HACursor

ZTPageController同样类似网易新闻https://github.com/IOStao/ZTPageController

awesome-ios-animation多个动画收集https://github.com/sxyx2008/awesome-ios-animation

SCNavigation 导航栏支持右滑返回，可隐藏navigationbar https://github.com/singro/SCNavigation

animated-tab-bar 让tabbar Items显示动画 swift https://github.com/Ramotion/animated-tab-bar 原理:计算、覆盖、替换原来的tabbarItem 自定义动画:继承RAMItemAnimation

JHChainableAnimations链式动画 https://github.com/jhurray/JHChainableAnimations 用的CAKeyframeAnimation 用法: myView.makeScale()(2). moveXY()(100,50).animate()(2) thenAfter分段动画的持续时间 效果spring跳跃, bounce回弹 等待wait()(2) 路径let path = myVIew.bezierPathForAnimation(), path.addLineToPoint(CGPoint(x: 25, y: 400)) myVIew.moveOnPath()(path).animate()(0.3)

UIImageView-PlayGIF gif动画https://github.com/liyong03/YLGIFImage-Swift

YLGIFImage 更省内存的gif动画https://github.com/liyong03/YLGIFImage

ADo\_GuideView旋转欢迎界面demo https://github.com/Nododo/ADo\_GuideView

Spring封装好的动画类https://github.com/MengTo/Spring swift 可以没代码

KYCuteView QQ消息类似的浮动小球https://github.com/KittenYang/KYCuteView 如果要拖拽断后消失\_frontView.hidden = YES; 判断断开的状态r1 <= 6

WZFlashButton点击button后出现水波纹效果https://github.com/SatanWoo/WZFlashButton

动画加获取轮廓：UIBezierPath(roundedRect: CGReckMake(-25,-25,50,50), cornerRadius: radius)（操作CAShapeLayer） 使用：设置buttonType

Mta: 腾讯云统计，查看app活跃（用户数，启动次数）

CocoaAsyncSocket：https://github.com/robbiehanson/CocoaAsyncSocket socket

XMLDictionary xml与NSDictionary相互转化开源类库

SwiftDate 时区、时间管理，计算

Slidden自定义键盘 https://github.com/Brimizer/Slidden

AFImageHelper https://github.com/melvitax/AFImageHelper swift,一套针对 UIImage 和 UIImageView 的实用扩展库，功能包含填色和渐变、裁剪、缩放以及具有缓存机制的在线图片获取。

SDCycleScrollView https://github.com/gsdios/SDCycleScrollView 图片轮播器

只需要cycleScrollView.imageURLStringsGroup = []

PictureWatermark https://github.com/cgwangding/PictureWatermark 水印

image.imageWaterMarkWith…

QQBugly http://bugly.qq.com 捕获发布应用崩溃的bug exp\_call\_back\_func = { CrashReporter.sharedInstance().setAttachLog("自定义回调的信息") return 1 } “查看更多->更多信息->额外数据” 可以看到自定义的回调信息

即时通讯三方：leanCloud：https://leancloud.cn/

发送图片：放到AVFile中，接收：通过file的url，记得改http

发送语音：用recordingMangaer转为mp3格式，其中不要roadInfo 接收并播放语音：下载到沙盒以后，播放用audioPlayer = AVAudioPlayer(data: NSData(contentsOfFile: filePath)!) 最好在真机上试

UICollectionViewDelegateFlowLayout根据不同的indexpath确定cell的大小

仿射变换公式: 即x’=ax+cy+tx y’=bx+dy+ty

平移: a = 1,b = 0, c = 0, d = 1 缩放:b = 0, c = 0,tx = 0,ty = 0

旋转: x' = x\*cos α - y \*sin α, y'= x \*sin α + y \*cos α

TangramKit或MyLayout：基于frame纯代码布局框架:

1.获取自适应布局的值：S.tg\_sizeThatFits()

2.动画: self.C.tg\_height.equal(100) S.tg\_layoutAnimationWithDuration(0.5)

3.其中TGLinearLayout用CAKeyframeAnimation不太好，因为加入不了布局因素

4. TGPathLayout可以很好地作路径布局和动画，这个时候可以用CAKeyframeAnimation

5.最常用的是TGRelativeLayout相对布局

6. TGFrameLayout就相当于基于父亲视图的绝对布局

7. TGTableLayout相当于html中的table标签

8. TGFlowLayout流式布局

9. TGFloatLayout浮动布局