# iOS知识点

1. KVC、KVO

KVC：（键值编码，一种通过字符串获取值的方式）

setValue:forKey:/setValue:forKeyPath:

1. 首先查找setKey:和\_setKey方法，如果有就执行相应的方法
2. 如果没有找到，会调用accessInstanceVariablesDirectly方法的返回值，（是否允许直接进入对象的成员变量）如果返回YES,就按照\_key,\_isKey,key,isKey的顺序查找成员变量，找到直接赋值。没找到调用setValueForUndefinedKey:并抛出异常。
3. 如果accessInstanceVariablesDirectly返回NO,调用setValueForUndefinedKey:，并抛出异常。

valueForKey:

1. 首先按照getKey,key,isKey，\_key来进行方法查找，存在就调用方法赋值;不存在就调用accessIntanceVariablesDirectly判断是否允许获取成员变量的值，返回YES就按照\_key,\_isKey,key,isKey顺序取值。否则调用valueForUndefinedKey:方法，并抛出异常。

KVO:（观察者模式）

KVO是基于Runtime机制实现的，当某个类的对象属性第一次被观察时，系统就会在运行时动态创建该类的一个派生类，在这个派生类中重写被观察属性的setter方法。派生类在被重写的setter方法内实现真正的通知机制。如果原类为A,那么派生类为NSKVONotifying\_A。每个类中都有一个isa指针指向当前类，所以系统就在类对象第一次被观察的时候，系统将isa指针指向动态生成的派生类，从而在被监听属性赋值时被执行的是派生类的setter方法。键值观察依赖于NSObject两个方法：willChangeValueForKey:和didChangeValueForKey:;在一个被观察属性发生改变之前，willChangeValueForKey:会被调用记录旧值，发生改变后，didChangeValueForKey:会被调用记录新值，此时observeValueForKey:ofObject:change:context:会被调用。

手动触发KVO

1. 调用+（BOOL）automaticallyNotifiesObserversForName返回NO,手动关闭willChangeValueForKey:和didChangeValueForKey:的自动调用
2. 手动调用willChangeValueForKey:和didChangeValueForKey:
3. 代理、通知、KVO

代理

代理是以协议的形式体现的一种设计模式，代理方实现相应的协议，委托方调用代理遵从的协议方法后，代理方返回相应的结果。

通知

使用观察者模式实现的用于跨层传递消息的机制。

三、Block

Block

Block是封装了函数以及执行上下文的OC对象。它本质是一个OC对象，它内部也有个isa指针。

Block截获变量（值传递、指针传递）

1. 对于基本数据类型的局部变量截获其值。
2. 对于对象类型的局部变量连同所有权修饰符一起截获
3. 以指针形式截获局部静态变量
4. 不能截获全局变量、静态全局变量，直接访问

（理解：1、不能修改自动变量的值是因为block捕获的是自动变量的const值，名字一样，不能修改。2、可以修改静态变量的值是因为静态变量是属于类的，不是某一个变量，由于block内部不用调用self指针，所以block可以调用。）

\_\_block作用：

\_\_block用于解决block内部不能修改变量值的问题，编译器会将\_\_block修饰的变量包装成一个对象。对象类型的变量经过\_\_block修饰后，block会在结构体内自动添加copy和dispose函数对捕获的变量进行内存管理。

Block循环引用

1. MRC情况下用\_\_block可以消除循环引用。
2. ARC情况下用\_\_weak可以解决循环引用。

为什么block对auto和static变量捕获有差异？

auto自动变量可能会销毁，内存可能会消失，不采用指针访问；static变量一直保存在内存中，指针访问即可。

load和initialize区别

1. 调用方式
2. load是根据函数地址直接调用的
3. initialize是通过objc\_msgSend调用

b、调用时刻

1. load是runtime加载类、分类的时候调用（只调用一次）
2. Initialize是类第一次接收消息的时候调用，每个类只会initialize一次（父类的initialize方法可能会被调用多次）

c、调用顺序

1. 先调用类的load
2. 先编译的类优先调用load
3. 调用子类的load之前会先调用父类的load
4. 在调用分类的load
5. 先编译的分类，优先调用load

Initialize

1. 先初始化父类
2. 在初始化子类（可能最终调用的是父类的initialize方法）

是否覆盖方法

load：分类中的load方法不会覆盖本类的load方法

Initialize:分类中的initialize方法会覆盖本类的initialize方法

消息转发

1. 调用resolveInstanceMethod:或resolveClassMethod:方法，允许用户此时为该class动态添加实现。如果有实现，调用并返回，没有实现继续如下：
2. 调用forwardingTargetForSelector:方法，尝试找到能响应该消息的对象，如果获取到就转发给它，如果返回nil,继续如下：
3. 调用methodSignatureForSelector：方法，尝试获得一个方法签名，如果获取不到就调用doseNotRecognizeSelector:抛出异常
4. 调用forwardInvocation:方法，将第三步获取到的方法签名包装成invocation传入。

RunLoop

RunLoop可以保持程序的持续运行，在没有事件处理的时候使程序进入休眠模式，从而节省CPU资源，提高程序性能。

RunLoop理解

RunLoopMode代表RunLoop的运行模式,一个RunLoop包含若干个mode,每个mode又包含若干个Source0、Source1、Timer、Observer,但至少必须有一个Source或者Timer，而RunLoop启动时只能选择一个mode作为currentMode。

RunLoop有五种模式，

1、KCFRunLoopDefaultMode:app默认mode,通常主线程在这个mode运行

2、UITrackingRunLoopMode:界面跟踪mode,用于ScrollView追踪触摸滑动，保证界面滑动时不受其他mode影响

3、UIInitializationRunLoopMode：在刚启动App时进入的第一个mode,启动完成后就不再使用，会切换到KCFRunLoopDefaultMode

4、GSEventReceiveRunLoopMode：接受系统事件内部的mode，通常用不到

5、KCFRunLoopCommonModes:这是一个占位mode,作为标记KCFRunLoopDefaultMode和UITrackingRunLoopMode用，并不是一种真正的mode。

CFRunLoopSourceRef（事件源/输入源）

Source分为Source0、Source1两种

Source0：非基于Port的，用于用户主动触发的事件（点击button或屏幕等）

Source1：基于Port的，通过内核和其他线程相互发送消息

CFRunLoopObserverRef

CFRunLoopObserverRef是观察者，能够监听RunLoop的状态改变，包括唤醒、休息、以及处理各种事件

RunLoop退出

1. 主线程销毁，RunLoop退出
2. Mode中一些Timer、Source、Observer,这些保证mode不为空时，RunLoop没有空转并且是在运行的。当mode中为空时，RunLoop会立刻退出
3. 启动RunLoop时可以设置手动停止,方法如下：

（[NSRunLoop currentRunLoop] runUntilDate:）

（[NSRunLoop currentRunLoop] runMode:beforeDate:）

RunLoop应用场景

1. 解决NSTimer失效问题
2. 控制线程生命周期（线程保活）
3. 自动释放池
4. 监控应用卡顿
5. 性能优化

线程保活

1. 定时器保活子线程

[NSRunLoop currentRunLoop]addTimer:forMode:;

[NSRunLoop currentRunLoop] run;

2、使用端口

[NSRunLoop currentRunLoop]addPort:[NSMachPort port]forMode:;

[NSRunLoop currentRunLoop] run;

3、条件锁

自动释放池

Timer和Source也是一些变量，需要占用一部分存储空间，所以要释放掉，如果不释放掉，占用内存会越来越大。RunLoop内部有一个自动释放池，当RunLoop开启时，就会自动创建一个自动释放池，当RunLoop在休息前会释放掉自动释放池的东西，然后重新创建一个新的空的自动释放池，当RunLoop被唤醒重新开始跑圈时，Timer、Source等新的事件就会放到新的自动释放池中，当RunLoop退出时也会被释放。

RunLoop与线程的关系

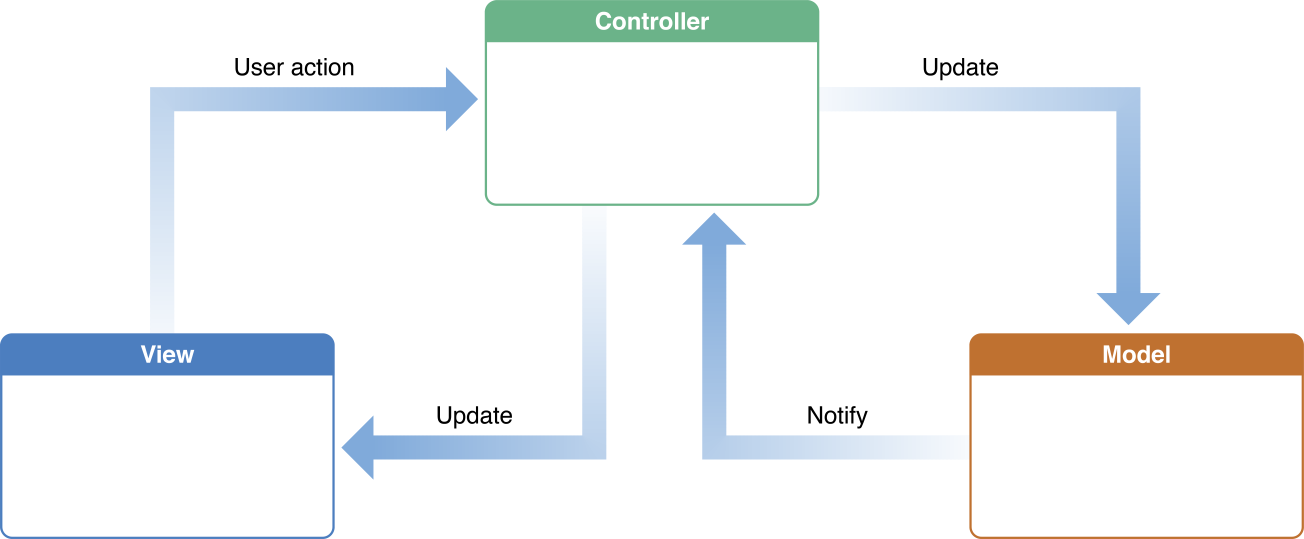
1. 每条线程都有一个唯一的一个与之对应的RunLoop对象
2. RunLoop保存在一个全局的Dictionary里，线程作为key,RunLoop作为value
3. 主线程的RunLoop已经自动创建好了，子线程的RunLoop需要主动创建
4. RunLoop在第一次获取时创建，在线程结束时销毁

MVC、MVVM、MVP

MVC:

MVC即Model View Controller（模型 视图 控制器）。模型对象封装了应用的数据，并定义操控和处理该数据的逻辑和运算。视图对象是应用中用户可以看见的对象，View的显示取决于Model,只要model数据改变，View的显示状态也跟着更改。控制器对象充当着媒介，Controller负责初始化Model，并将model数据传递给View去解析展示。

（用户操作View,在Controller层完成业务逻辑处理后更新Model层，将数据显示在View层。MVC中每层都有关联，耦合紧密。View把控制权交给Controller层，自己不执行业务逻辑。Controller层执行业务逻辑并操作Model层，但不会直接操作View层。View和Model层的消息同步是通过观察者模式进行的，可以做到多视图同时更新）



缺点：Controller变成了所有东西的代理和数据源，甚至还负责网络的请求发起和取消等，造成控制器臃肿，没有区分业务逻辑和业务展示，对单元测试很不友好。Controller还负责页面的跳转，容易把源界面和目的界面耦合起来。

MVP:

MVP即Model View Presenter（模型 视图 协调器）。View负责界面展示和布局管理，向Presenter暴露视图更新和数据获取的接口，Presenter负责接收来自View的事件，通过View提供的接口更新视图，并管理Model，只是Model和View层的一个桥梁。

MVC和MVP的区别：

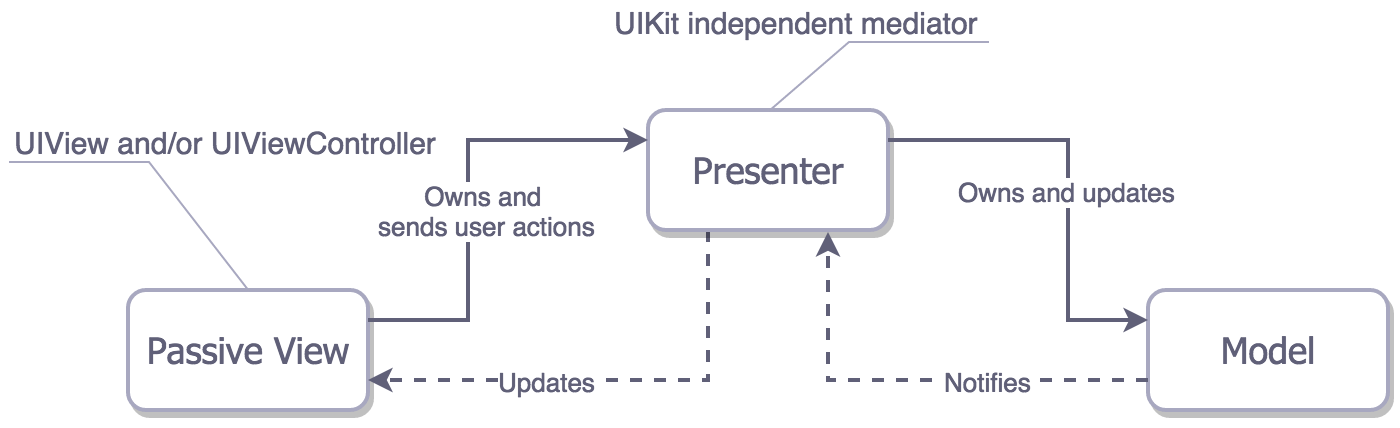
1、在MVP中M和V没有直接通信，模型和视图完全分离。

2、MVC中UIView相当于View,UIController是Controller，而在MVP中，UIView和UIController都相当于View,所以Presenter里面基本没什么布局相关的代码，它的职责只是通过数据和状态更新View。

3、MVC中C持有M和V，MVP中V持有P，P持有M

特点：任务均摊、可测试性、易用性

（用户操作View，在Presenter层完成业务逻辑处理更新Model层，通过Presenter回调将数据显示在View层，完全隔断Model和View间的通信，通过接口的方式连接View和Presenter层）

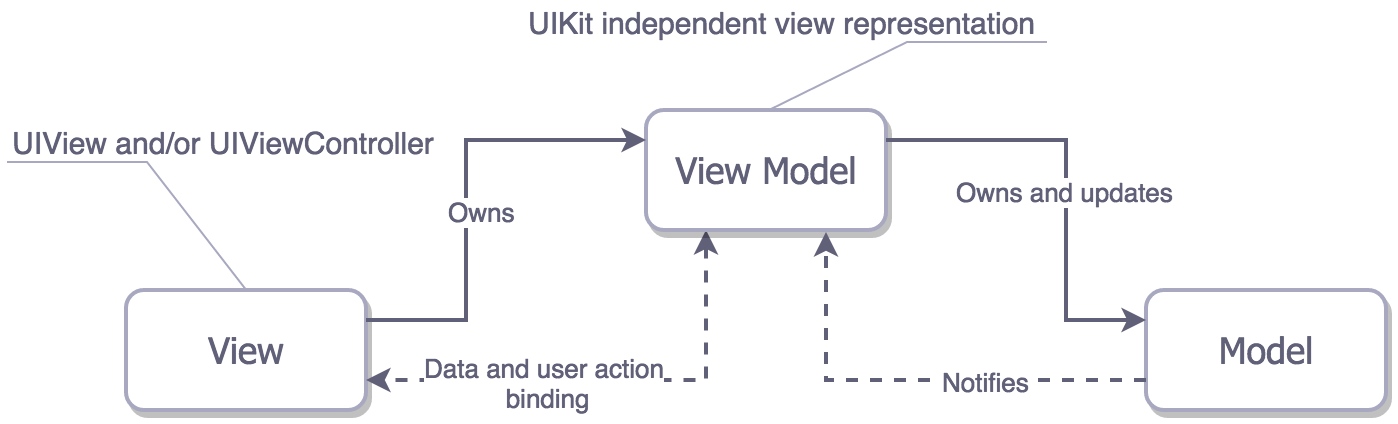


MVVM:

MVVM即Model View ViewModel(模型 视图 视图模型)。

ViewModel本质上讲它是独立于UIKit的，View和View状态的一个呈现。ViewModel能主动调用对Model做更改，也能在Model更新的时候对自身进行调整，然后通过View和ViewModel之间的绑定，对View进行相应的更新。

MVP和MVVM最大的区别是采用了双向绑定机制，View的变动，自动反应在ViewModel上。



1. masony原理
2. weak实现
3. 多线程任务执行的顺序（1先执行，2、3同时执行且在1后执行，4执行，上面执行完在执行5）
4. MJExtension数据解析
5. 数据传输加密，登录时密码的处理
6. HTTP HTTPS 加密实现
7. SetValueForkey和setobjectForkey区别
8. Reactn-navigation navige /push/pop/页面跳转，跨页面跳转