AFNetworkRachabilityManager

该类主要是对进行SCNetworkReachability的封装

初始化方法：

**+sharemanger 单例调用manager方法**

**+managerfordomain: 对指定域名监听**

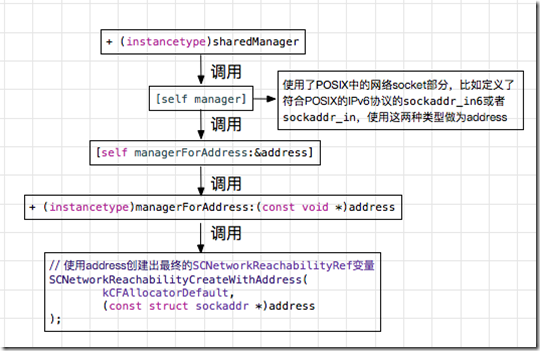
**调用**

**SCNetworkReachabilityCreateWithName（kCFALLocator,UTF8String）**

**+managerforadress 对指定端口监听**

**SCNetworkReachabilityCreateWithAddress(kCFALLocator,Struct)**

**+manager 底层调用adrss**



回调方法：

-setReachabilityStatusChangeBlock 完成回调

状态值：

**AFNetworkReachabilityStatus**

**//未知**

**AFNetworkReachabilityStatusUnknow -,**

**//没有网络**

**A AFNetworkReachabilityStatusUnReach**

**//手机网络**

**AFNetworkReachabilityStatusRachableViaWan ,**

**//wifi网络**

**AFNetworkReachabilityStatusRachableViaViaWifi**

**};**

核心方法:

-startMonitoring

SCNetworkReachabilityContext：

**context是一个结构体**

**typedef struct {**

**// 创建一个SCNetworkReachabilityContext结构体时，需要调用SCDynamicStore的创建函数，而此创建函数会根据version来创建出不同的结构体，SCNetworkReachabilityContext对应的version是0**

**CFIndex version;**

**// 下面两个block（release和retain）的参数就是info，此处表示的是网络状态处理的回调函数**

**void \* \_\_nullable info;**

**// 该retain block用于对info进行retain，下面那个AFNetworkReachabilityRetainCallback核心就是调用了Block\_copy（用于retain一个block函数，即在堆空间新建或直接引用一个block拷贝）**

**const void \* \_\_nonnull (\* \_\_nullable retain)(const void \*info);**

**// 该release block用于对info进行release，下面那个AFNetworkReachabilityReleaseCallback核心就是调用了Block\_release（用于release一个block函数，即将block从堆空间移除或移除相应引用）**

**void (\* \_\_nullable release)(const void \*info);**

**// 提供info的description，此处调用为NULL**

**CFStringRef \_\_nonnull (\* \_\_nullable copyDescription)(const void \*info);**

**} SCNetworkReachabilityContext;**

**\*/**

**SCNetworkReachabilitySetCallback 为初始化的CFReachability 设置状态变更回调 回调具体方法参数为context里定义的info 此处即为**AFNetworkReachabilityStatusBlock callback

**SCNetworkReachabilityScheduleWithRunLoop 加入线程loop**

**主动获取一次**

注册键值依赖 ：

有时候一个属性的值依赖于另一对象中的一个或多个属性，如果这些属性中任一属性的值发生变更，被依赖的属性值也应当为其变更进行标记。因此，object 引入了依赖键。

+ (NSSet<NSString \*> \*)keyPathsForValuesAffectingValueForKey:(NSString \*)key ;

**AFSecurityPolicy - 网络安全策略**

**https原理**

****

**初始化：**

//1. 不允许无效或过期的证书

//2. 验证domain名称

//3. 不对证书和公钥进行验

**+defaultPolicy**

**+policyWithPinningMode:pinngingMode 调用下面的函数**

**+ policyWithPinningMode: withPinnedCertificates 设置参与验证的本地证书**

**证书校验模式**

1. **AFSSLPiningModenon 不是不校验 校验证书的权威性**
2. **AFSSLPiningModepublickey 通过比较证书当中公钥(PublicKey)部分来进行验证，通过SecTrustCopyPublicKey方法获取本地证书和服务器证书，然后进行比较，如果有一个相同，则通过验证，**

**3 AFSSLPiningModeCertificate 直接将本地的证书设置为信任的根证书，然后来进行判断，并且比较本地证书的内容和服务器证书内容是否相同，来进行二次判断，此方式适用于较高安全要求的验证。**

**核心方法:**

**-evaluateSeverTrust:for:domain**

SecCertificateCreateWithData（CFAllocatorRef，CFdata）内存分配类型,数据 创建证书

SecPolicyCreateBasicX509(void)

SecPolicyCreateSSL(boll,hostName) **创建验证策略验证domain 使用SecPolicyCreateSSL函数创建验证策略，其中第一个参数为true表示验证整个SSL证书链，第二个参数传入domain，用于判断整个证书链上需要验证的节点表示的那个domain是否和此处传入domain一致**

SecTrustCreateWithCertificates(certificates,policuew,SecTrustRef) 基于给定的证书和政策创建一个信任对象

SecTrustSetPolicies(SecTrust,Policies) 设置信任对象策略

SecTrustSetAnchorCertificates（SecTrustRef,CFArray） 设置根证书

SecTrustEvaluate 同步验证证书

SecTrustGetCertificateCount

获取证书数量

SecTrustGetCertificateAtIndex

获取信任对象指定位置证书

SecTrustEvaluate(SecTrustRef trust, SecTrustResultType \* \_\_nullable result)  
Trust           信任管理对象，里面包含了待验证的证书，和自定义的策略。  
Result          验证结果，有以下情况：

* kSecTrustResultUnspecified 证书验证成功，但是用户没有明确指出信任此证书。这是最常见的返回值。
* kSecTrustResultProceed 用户选择信任此证书。
* kSecTrustResultDeny 用户选择不信任次证书。
* kSecTrustResultRecoverableTrustFailure 证书不可信，但是经过较小的改动可以修复问题，例如忽略过期证书、增加信任链节点等。在iOS中通常会拒绝该证书，但是收到信息时证书未过期可以通过验证，因此可以验证证书在收到信息时是否过期。
* kSecTrustResultConfirm 用户预先选择了证书链中得某一个证书在每次使用前询
* 问允许。这个返回值已经不再使用，只在老版本的OS X中使用。

**最后通过AFSevertrustIsValid代码块返回结果**

**只有当result为kSecTrustResultUnspecified（此标志表示serverTrust评估成功，此证书也被暗中信任了，但是用户并没有显示地决定信任该证书），或者当result为kSecTrustResultProceed（此标志表示评估成功，和上面不同的是该评估得到了用户认可），这两者取其一就可以认为对serverTrust评估成功**

* **\_\_builtin\_expect 说明**
* 这个指令是gcc引入的，作用是**允许程序员将最有可能执行的分支告诉编译器**。这个指令的写法为：\_\_builtin\_expect(EXP, N)。  
  意思是：EXP==N的概率很大。
* #define likely(x) \_\_builtin\_expect(!!(x), 1) //x很可能为真 #define unlikely(x) \_\_builtin\_expect(!!(x), 0) //x很可能为假
* \_\_builtin\_expect((x),1)表示 x 的值为真的可能性更大；  
  \_\_builtin\_expect((x),0)表示 x 的值为假的可能性更大。

编译器在编译过程中，会将可能性更大的代码紧跟着前面的代码，从而减少指令跳转

CF和OC相互转化  
我们可以通过**bridge,**bridge\_transfer,\_\_bridge\_retained 来进行内存管理

1.\_\_bridge  
CF和OC对象转化时只涉及对象类型不涉及对象所有权的转化

2.\_\_bridge\_transfer  
常用在CF对象转化成OC对象时，将CF对象的所有权交给OC对象，此时ARC就能自动管理该内存,作用同CFBridgingRelease()

3.**bridge\_retained  
与**bridge\_transfer 相反，常用在将OC对象转化成CF对象，且OC对象的所有权也交给CF对象来管理，即OC对象转化成CF对象时，涉及到对象类型和对象所有权的转化，作用同CFBridgingRetain()

<http://www.jianshu.com/p/1e3cebda8ec4>

<http://www.jianshu.com/p/11c3bc21f56e>

<http://www.jianshu.com/p/2684613a300f>