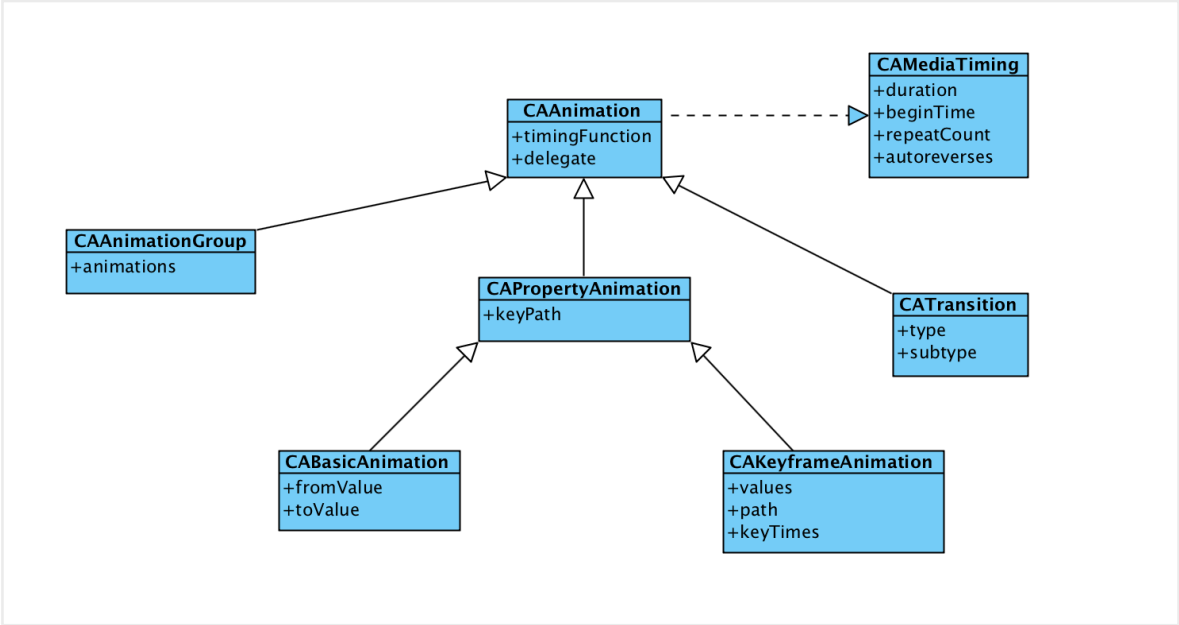
00 - index

组件化
防止 NSMutableDictionary 添加 nil 崩溃了
防止 @selector 找不到就崩溃
A 业务需要 B 业务的部分，怎么解耦
路由
NSHashTable 相关的知识
1000 条数据怎么急速写入数据库

01 - 动画

小结概述



1.UIView 的动画

在 IOS4.0 以前，用 begin 和 commit 模式，代码如下：

```
[UIView beginAnimations:@"View Flip" context:nil];
//动画持续时间
[UIView setAnimationDuration:1.5];
//设置动画的回调函数，设置后可以使用回调方法
[UIView setAnimationDelegate:self];
//设置动画曲线，控制动画速度
[UIView  setAnimationCurve: UIViewAnimationCurveEaseInOut];
//设置动画方式，并指出动画发生的位置
[UIView setAnimationTransition:UIViewAnimationTransitionCurlUp forView:self.view  cache:YES];
//提交UIView动画
[UIView commitAnimations];
```

```objc
[UIView beginAnimations:@"View Flip" context:nil];
//动画持续时间
[UIView setAnimationDuration:1.5];
//设置动画的回调函数，设置后可以使用回调方法
[UIView setAnimationDelegate:self];
//设置动画曲线，控制动画速度
[UIView  setAnimationCurve: UIViewAnimationCurveEaseInOut];
//设置动画方式，并指出动画发生的位置
[UIView setAnimationTransition:UIViewAnimationTransitionCurlUp forView:self.view  cache:YES];
//提交UIView动画
[UIView commitAnimations];
```


```objc
[UIView beginAnimations:@"View Flip" context:nil];
//动画持续时间
[UIView setAnimationDuration:1.5];
//设置动画的回调函数，设置后可以使用回调方法
[UIView setAnimationDelegate:self];
//设置动画曲线，控制动画速度
[UIView  setAnimationCurve: UIViewAnimationCurveEaseInOut];
//设置动画方式，并指出动画发生的位置
[UIView setAnimationTransition:UIViewAnimationTransitionCurlUp forView:self.view  cache:YES];
//提交UIView动画
[UIView commitAnimations];
```


可选的动画代理：
-(void)animationDidStart:(CAAnimation *)anim

-(void)animationDidStop:(CAAnimation *)anim finished:(BOOL)flag

在iOS4.0及以后支持block模式，代码如下：

在动画视图（UIView）上添加点击手势，在手势中添加动画代码如下：

```objc
__block CGRect rect = gesture.view.frame;
[UIView animateWithDuration:1.5 delay:0 options:UIViewAnimationOptionCurveEaseInOut animations:^{
    rect.origin.y += 200;
    [gesture.view setFrame:rect];
    [gesture.view setUserInteractionEnabled:NO];
} completion:^(BOOL finished) {
    [gesture.view removeFromSuperview];
}];
```

2. Core Animation 动画

1. CABasicAnimation 基本单一类型的动画：
rotation
scale
translation

2. CAKeyframeAnimation 帧动画
主要操作属性有 keyPath 和 values 值组合。

3. CAAnimationGroup 组合动画
操作属性：animations 将CAAnimation类型的动画加入数组，FIFO的方式执行。

02 - 常见排序算法
小结概述

2.1冒泡排序
```
for (int i = 0; i < length - 1; i++) {
    for (int j = 0; j < length - i - 1; j++) {
```

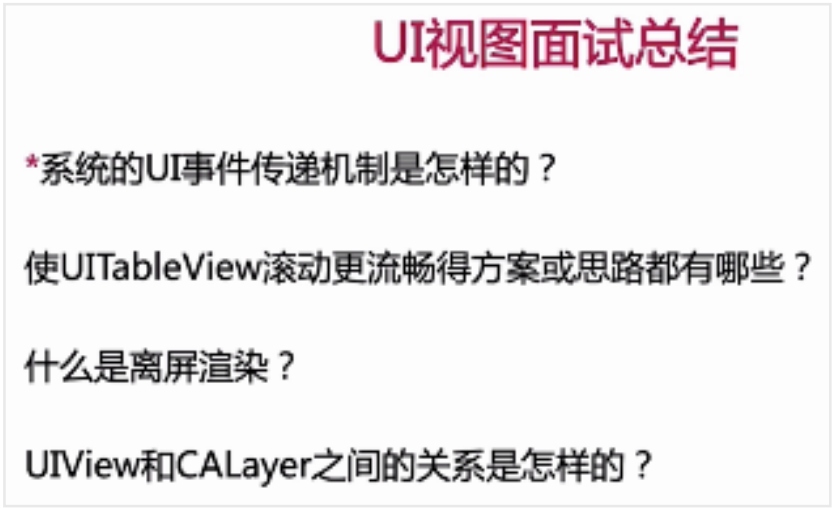
```
            if (array[j] > array[j+1]) {
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
            }
        }
    }
```

2.2 简单的选择排序

```
    for (int i = 0; i < length; i++) {
        int min = i;
        for (int j = i+1; j < length; j++) {
            if (array[j] < array[min]) {
                min = j;
            }
        }
        int temp = array[i];
        array[i] = array[min];
        array[min] = temp;
    }
```

03 - UI视图

小结概览



3.1 UITableView相关

3.1.1 重用机制 (重用池)

3.1.2 数据源同步

1.并发访问、数据拷贝。(多内存内存开销)

2.串行访问. (界面延迟)

3.2UI时间传递和相应

3.2.1UIView和CALayer(单一职责原则)

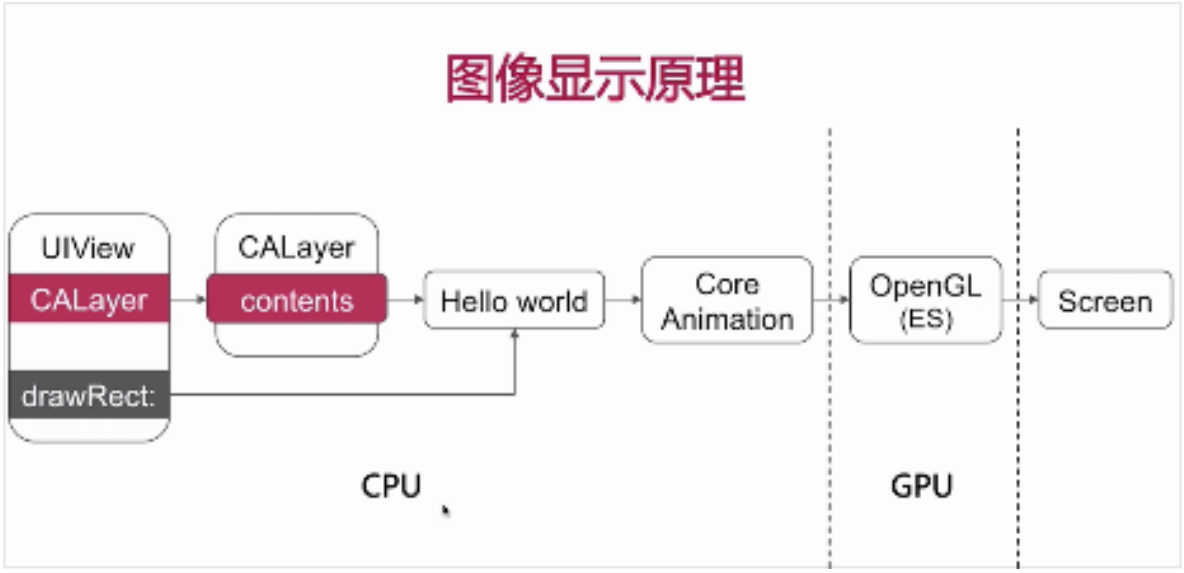
UIView为其提供内容，以及负责处理触摸等事件，参与响应链

CALayer负责显示内容contents

3.2.2 事件传递与视图响应链
事件传递流程 (hitTest withEvent、pointInside withEvent)

事件响应流程 (touchBegin、touchMoved、touchEnded)

3.3 图像显示原理



CPU 工作
Layout：UI 布局、文本计算
Display: 绘制 (drawRect)
Prepare: 图片解编码
Commit: 提交位图
GPU 渲染管线
顶点着色 图元装配 光栅化 片段着色 片段处理
3.4UI 卡顿、掉帧的原因
下一个 VSync 信号到来之前，CPU 和 GPU 并没有对下一帧画面的合成
3.4.1 滑动优化方案

CPU

对象创建 调整 销毁放在子线程

预排版 (布局计算、文本计算) 放在子线程

预渲染 (文本等异步绘制、图片编解码等)

GPU

纹理渲染 (避免离屏渲染)

视图混合 (减轻视图层级的复杂度)

3.5UIView 的绘制原理

当前 Runloop 即将结束的时候、才会开始介入到当前 UI 视图绘制中

3.5.1 异步绘制

[layer.delegate displayLayer:]如果实现了

代理负责生成对应的位图

设置该位图作为 layer.contents 属性的值

3.5.2 离屏渲染 (指定了 UI 视图的某些属性、标记为在未预合之前不能用于当前屏幕显示的时候,就会触发离屏渲染)

在屏渲染：GPU 的渲染操作是当前用于显示的屏幕缓冲区中进行

离屏渲染：GPU 在当前屏幕缓冲区以外新开辟一个缓冲区进行渲染操作

3.5.3 何时触发离屏渲染

圆角 (maskToBounds) 阴影 图层蒙版 光栅化

3.5.4 为什么要避免

离屏渲染增加了 GPU 工作量

03.2 - MVVM

可以看到相对 MVP 的 view 触发 P 的业务逻辑，然后 P 再回调改变 View 的显示的操作，使用 MVVM 的数据绑定来实现让逻辑更加清晰，代码也更少。这就是 MVVM 相对于 MVP 的改进之处

Interactor（交互器）- 这是应用程序的主干，因为它包含应用程序中用例描述的业务逻辑。交互器负责从数据层获取数据，并执行特定场景下的业务逻辑，其实现完全独立于用户界面。

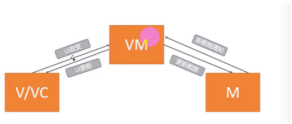
Presenter（展示器）- 它的职责是从用户操作的 Interactor 获取数据，创建一个 Entities 实例，并将其传送到 View 以显示它。

Entities（实体）- 纯粹的数据对象。不包括数据访问层，因为这是 Interactor 的职责。

Router（路由）- 负责 VIPER 模块之间的跳转

View（视图）- 视图的责任是将用户操作发送给演示者，并显示 presenter 告诉它的任何内容

MVVM



MVVM

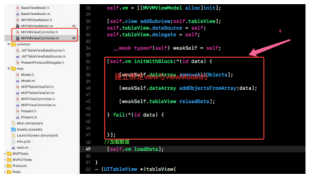
MVVM：中心思想是数据实现双向绑定，业务逻辑与试图的分离。

单独的看图可能太抽象了项目地址：demo

项目总体框架：

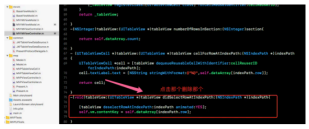
整体的框架

正向绑定：view--->viewModel



正向绑定

反向绑定：viewModel---->view(用了KVO)



数据变化

到了这里基本上实现了MVVM的双向绑定，没有很难的理解的地方：简单说一下mvvm优缺点：

优点：

1. 任务均摊 -- MVVM 的 View 要比 MVP 中的 View 承担的责任多。因为前者通过 ViewModel 的设置绑定来更新状态，而后者只监听 Presenter 的事件但并不会对自己有什么更新。

2.可测试性 -- ViewModel 不知道关于 View 的任何事情，这允许我们可以轻易的测试 ViewModel，同时 View 也可以被测试，但是由于属于 UIKit 的范畴，对他们的测试通常会被忽略。

3.易用性 -- 在实际开发中必须把 View 中的事件指向 Presenter 并且手动的来更新 View，如果使用绑定的话，MVVM 代码量将会小的多。

4.MVC视图控制器太大且难以管理，这个模式提供了一个很好的代替MVC的方案，它保证了让视图控制器的轻量性。

5.双向绑定数据。

缺点：

3.数据绑定使得 Bug 很难被调试，你看到界面异常了，有可能是你 View 的代码有 Bug，也可能是 Model 的代码有问题。数据绑定使得一个位置的 Bug 被快速传递到别的位置，要定位原始出问题的地方就变得不那么容易了。

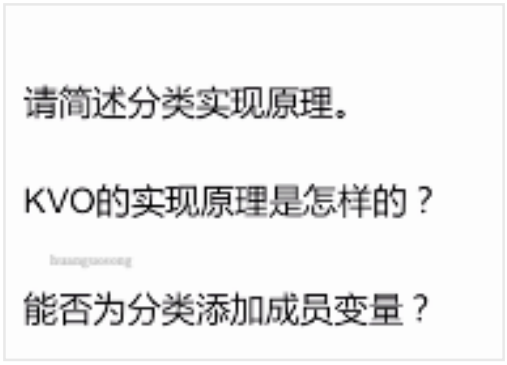
2.对于过大的项目，数据绑定需要花费更多的内存。

10人点赞 >

日记本

"小礼物走一走，来简书关注我"

04 - Object-C
小结概览



4.1 Category
4.1.1Category 作用
声明私有方法
分解体积庞大的类文件
把 Framework 的私有方法公开
4.1.2 特点
运行时决议
可以为系统类添加分类
4.1.3 可以添加什么?
实例方法
类方法
协议
属性
4.2.1Extension 作用
声明私有属性
声明私有方法
声明私有成员变量
4.2.2 特点
编译时决议
只能声明的形式存在
不能为系统类添加扩展
4.3 关联对象
是由 AssociationsManager 管理并在 AssociationsHashMap 存储。所以对象的关联内容都是放在同一个全局容器中
4.4 代理
是一种设计模式
4.5 通知
是使用观察者模式来实现的用于跨层传递消息的机制
4.6KVO
KVO 是 Key-value-observing 的缩写
KVO 是 OC 对观察者设计模式的又一实现
Apple 采用了 isa 混写 (isa-swizzing) 来实现 KVO
注意：
使用 setter 方法改变值 KVO 才能生效
使用 setValue:forKey:改变值 KVO 才能生效
成员变量直接修改需要手动添加方法，KVO 才能生效
4.7KVC
Key-value coding 键值编码技术
4.8 属性关键字

读写权限 readwrite readonly
原子性 nonatomic atomic(只保证赋值获取是线程安全的)
引用技术
retain/strong
assign/weak
copy

源对象类型	拷贝方式	目标对象类型	拷贝类型(深/浅)
mutable对象	copy	不可变	深拷贝
mutable对象	mutableCopy	可变	深拷贝
immutable对象	copy	不可变	浅拷贝
immutable对象	mutableCopy	可变	深拷贝

05 - Runtime
小结概览

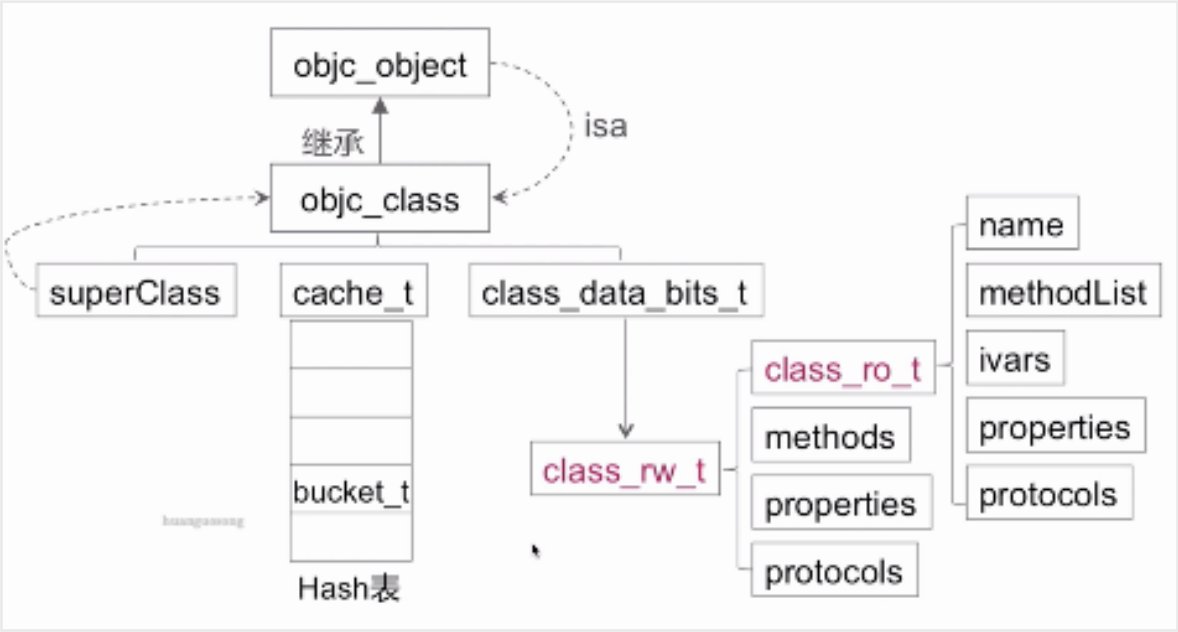
5.1数据结构
objc_object
    isa_t
    关于isa操作相关
    弱引用相关
    关联对象相关
    内存管理相关
objc_class
    继承自objc_object
    superClass
    cache_t cache方法缓存
    class_data_bits_t bits(属性、方法等)
isa指针
    指针isa
    非指针isa

method_t

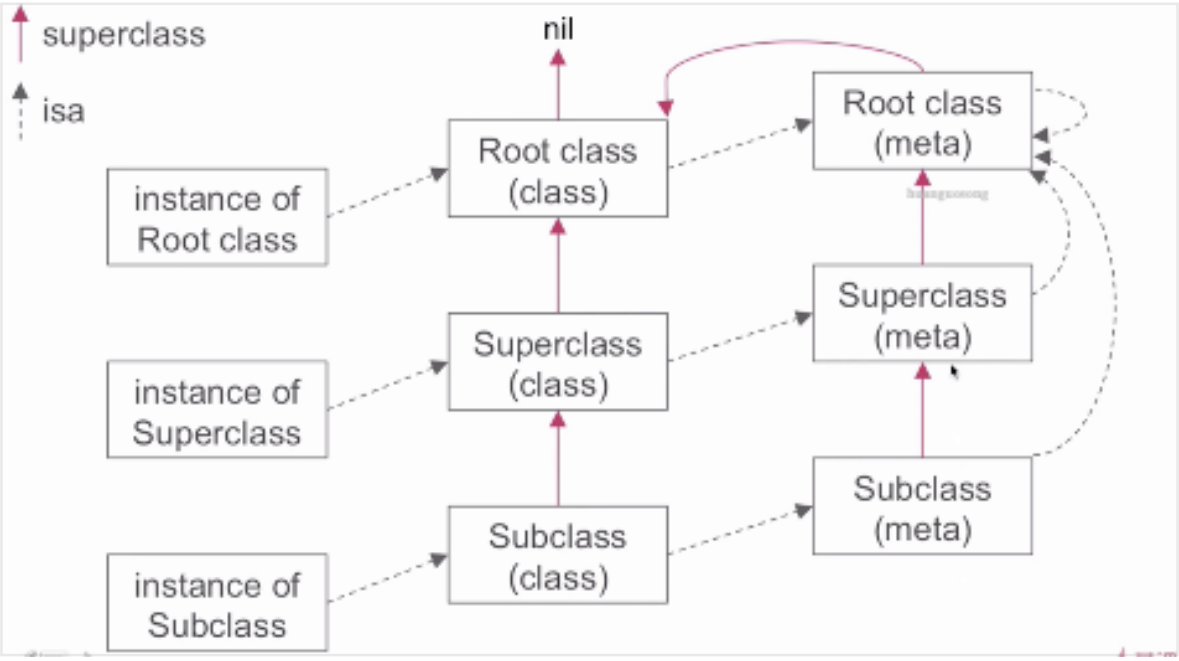
SEL name

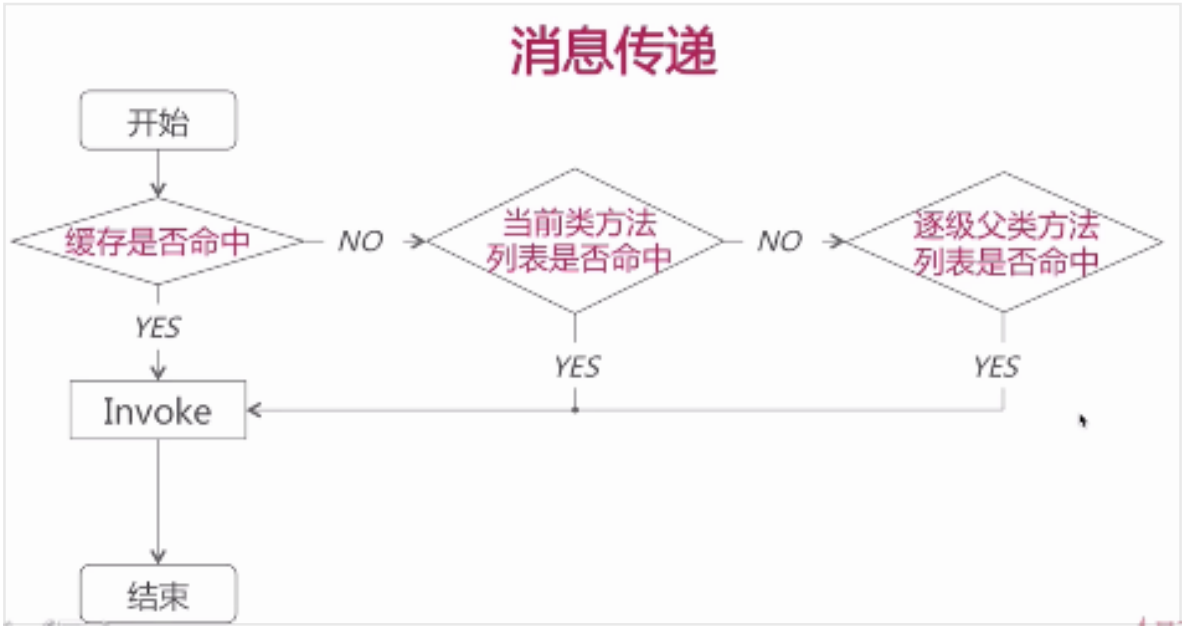
Const char* types(V@: = -(void)aMethod;)

IMP imp

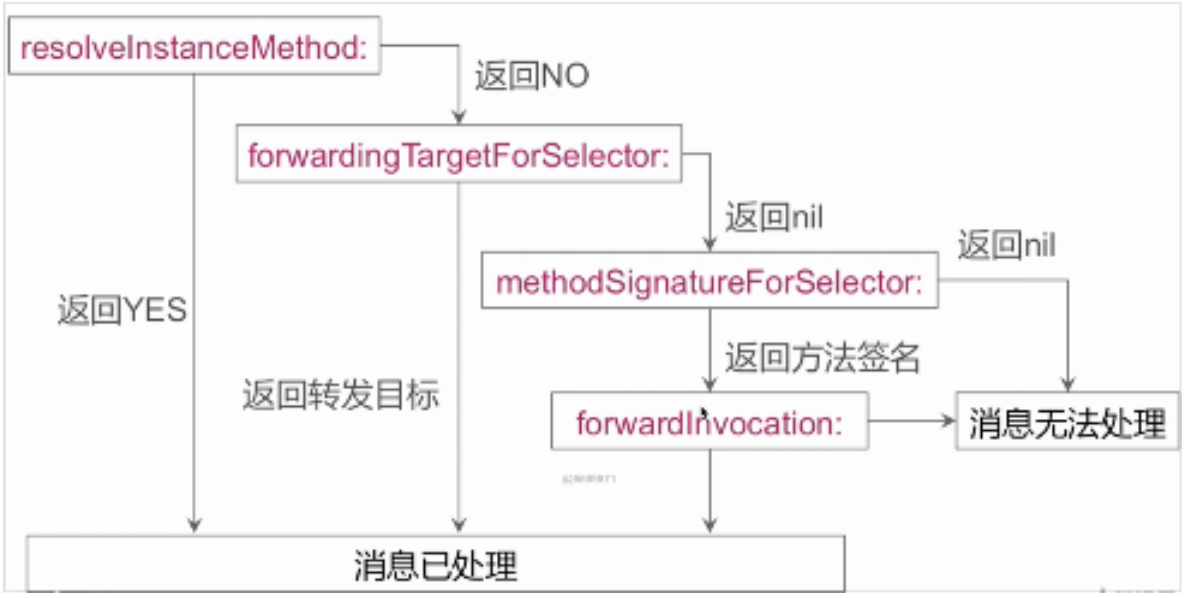


5.2 消息传递



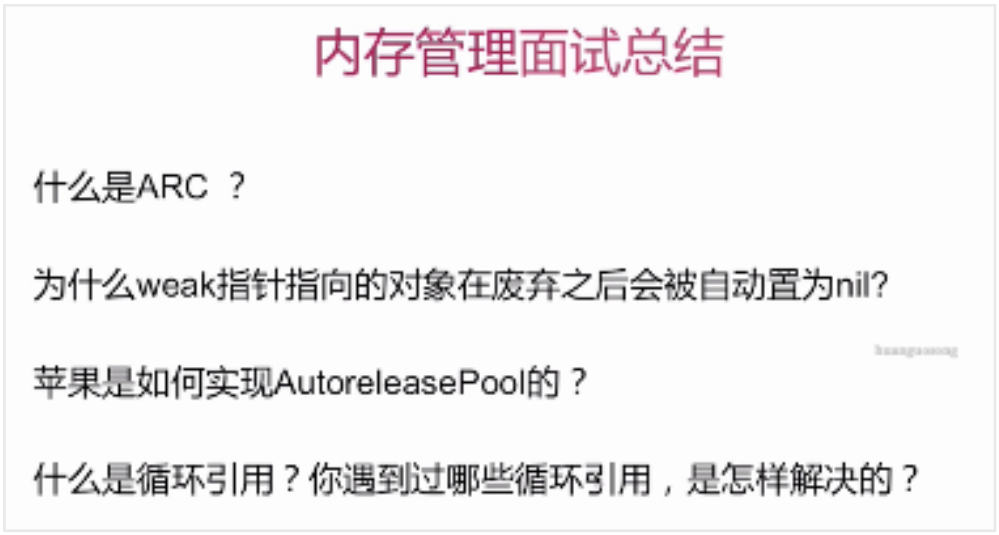


5.3 消息转发

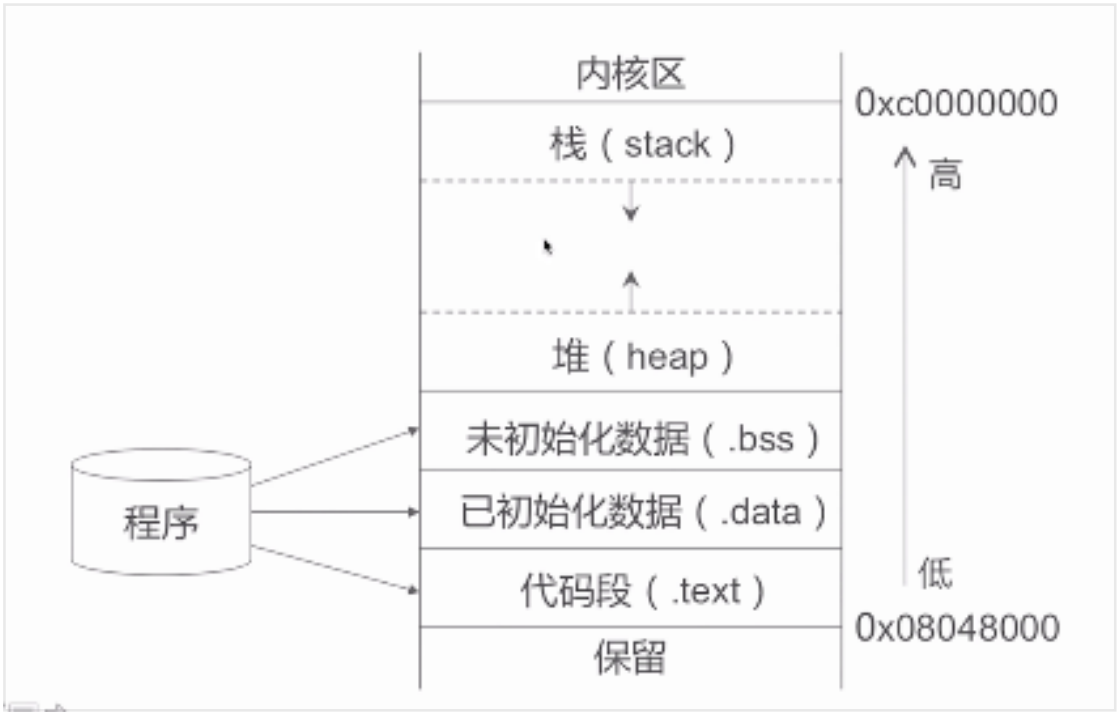


06 - 内存管理

小结概览



6.1 内存布局



stack:方法调用
heap:通过alloc等分配的对象
bss:未初始化的全局变量等
data:已初始化的全局变量等
text:程序代码
6.2 内存管理方案
TaggedPointer(NSNumber等小对象)
NONPointer_isa(非指针的isa)
散列表(引用计数表、弱引用表、关联对象)
6.2.1散列表方法
SideTables
SideTable
spinlock_t 自旋锁(忙等的表、适用于轻量访问)

RefcountMap 引用计数表
weak_table_t 弱引用表

6.3.1MRC 手动引用计数

Alloc retain release retainCount autorelease dealloc

6.3.2ARC 自动引用计数

ARC 是编译器和 runtime 协作的结果

禁止调用 MRC 中的 retain/release/retainCount/dealloc

ARC 中新增 weak strong 属性等关键字

6.4 自动释放池

是以栈对结点通过双向链表的形式组合而成的。

是和线程是一一对应的

在当次 runloop 将要结束的时候调用 AutoreleasePoolPage::pop()

多层嵌套就是多次插入哨兵对象

在 for 循环中 alloc 图片数据等内存消耗大的场景手动插入 autoreleasePool

6.5 循环引用

自循环引用 (self.block)

相互循环引用 (delegate)

多循环引用

6.5.1 破除循环引用

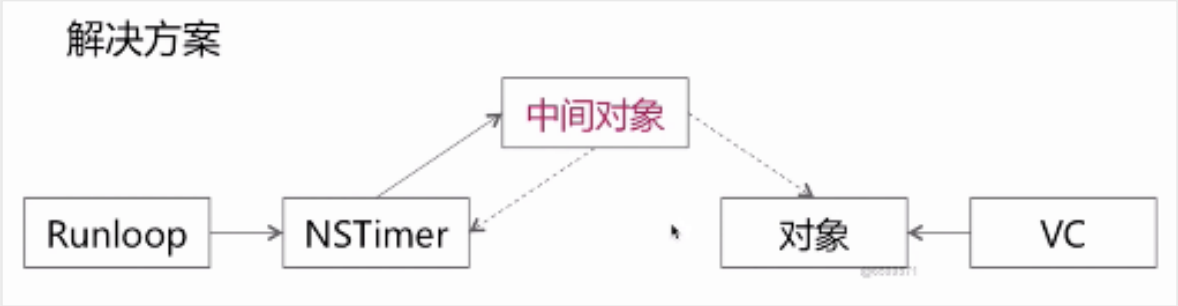
__weak(delegate)

__block

MRC 下，__block 修饰的对象不会增加引用计数，避免了循环引用

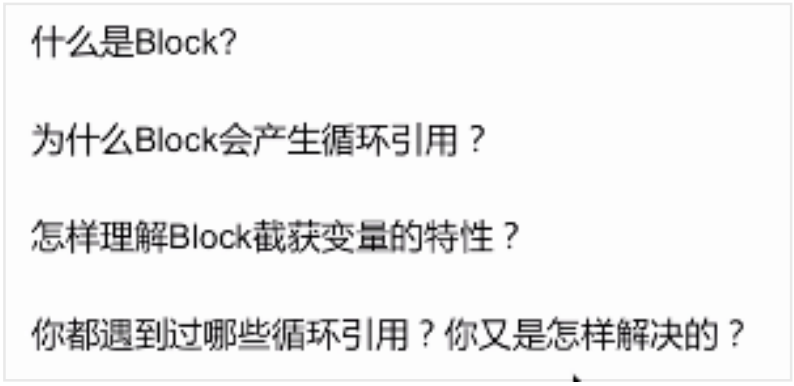
ARC 下，__block 修饰对象会被强引用，无法避免循环引用，需要手动解环

6.5.2NSTimer 的循环引用问题



07 - 关于 Block

小结概览



7.1什么是 block
Block 是将函数及其执行上下文封装的对象
Block 的调用即使函数的调用

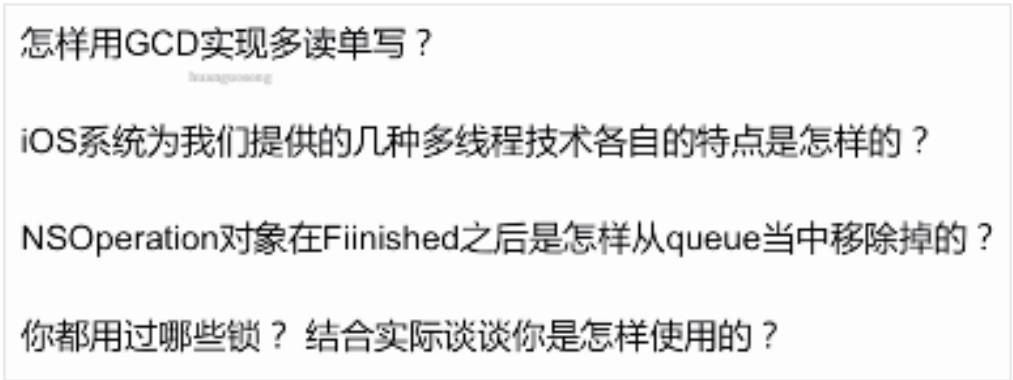
7.2 截获变量
局部变量
　　基本数据类型：截获其值
　　对象类型：连同所有权修饰符一起截获 (强引用)
局部静态变量：指针形式
不能截获全局变量、静态变量

7.3__block 修饰符 (__block 修饰的变量变成了对象)
一般情况下，对截获变量进行[赋值操作]需要添加__block 修饰符
不需要__block 修饰符 静态局部变量 全局变量 静态全局变量

7.4Block 内存的管理
NSGlobalBlock(全局区)　copy 结果：什么也不做
NSStackBlock(栈区)　　copy 结果：堆
NSMallocBlock(堆区)　　copy 结果：增加引用计数

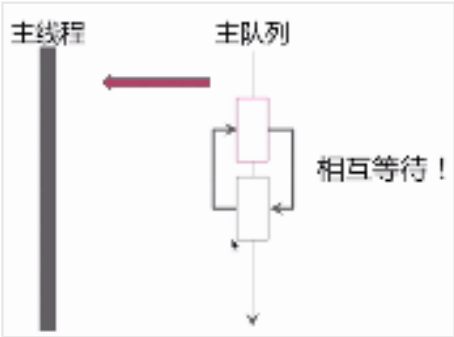
7.5Block 的循环引用 (__weak)

08 - 多线程和锁
小结概览

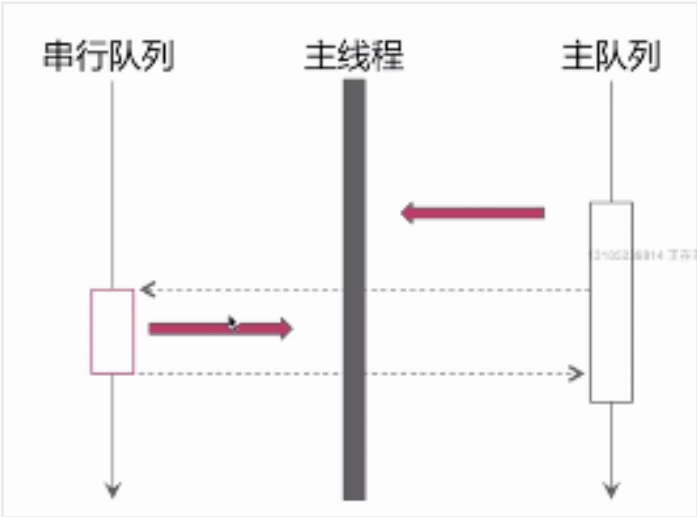


8.1 GCD

8.1.1 同步/异步 和 串行/并发
同步串行主队列：死锁、队列引起的循环等待



同步串行其他串行队列，没问题



同步并发：只要是同步方式去提交任务、无论是串行还是并行，都是在当前线程执行
异步串行：回到主线程
异步并发：下截图是：13，GCD 创建的线程没有 runloop，即使 delay0s。也无法运行

```
- (void)viewDidLoad {

        dispatch_async(global_queue, ^{

                NSLog(@" 1" ) ;

                [self performSelector : @selector(printLog)
                                withObject : nil
                                 afterDelay : 0 ] ;

                NSLog(@" 3" );

        });

}

- (void)printLog { NSLog(@" 2" ) ; }
```

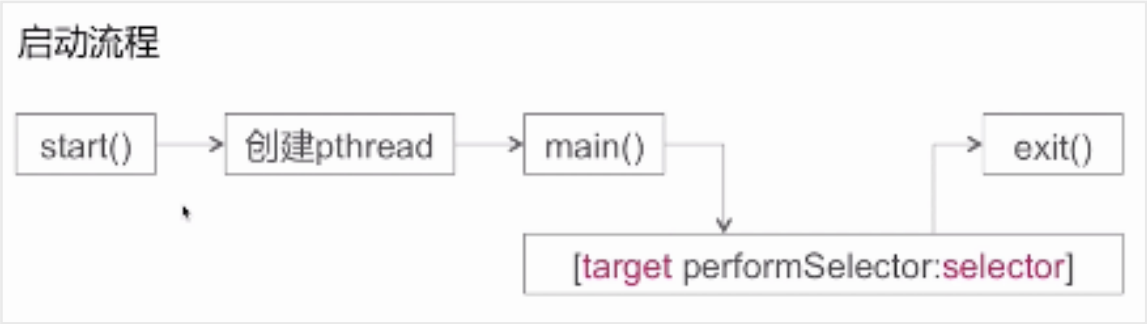
8.2dispatch_barrier_async() 多读单写
同步读取指定数据
异步栅栏调用设置数据

8.3dispatch_group_async()

8.4 NSOperation(需要和NSOperationQueue配合使用来实现多线程方案)
可以添加任务依赖
可以设置操作执行的优先级
任务执行状态的控制(isReady isExecuting isFinished isCancelled) 通过KVO实现
控制最大并发量

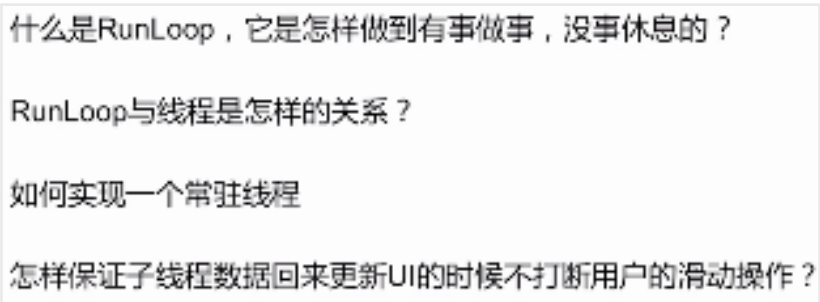
8.4.1状态控制(系统是通过KVO的方式移除一个isFinished=YES的Operation)
如果只重写了main方法，底层控制变更任务执行完成状态，以及任务退出
如果重写了start方法，自行控制任务的状态

8.5NSTread



8.6 锁
@synchronized 创建单例的使用, 现在使用dispatch_once
atomic 对被修饰对象进行原子性操作(不负责使用),
OSSpinLock 自旋锁、循环等待询问，不释放当前资源。用于轻量级数据访问(int++)
NSRecursiveLock
NSLock
dispatch_semaphore_t

09 - RunLoop
小结概览



9.1 什么是RunLoop
RunLoop是通过内部维护的事件循环来对事件/消息进行管理的一个对象。
　　没有消息需要处理时，休眠以避免资源占用(用户态 -> 内核态)
　　有消息需要处理时，立刻被唤醒(用户态 <- 内核态)

9.2RunLoop的数据结构

CFRunLoop
    pthread：一一对应
    currentMode: CFRunLoopModel的实例
    modes: NSMutableSet<CFRunLoopMode*>
    commonModes: NSMutableSet<NSString*>
    commonModeItems: 多个Timer、Observe、Source

CFRunLoopMode
    name: NSDefaultRunLoopMode
    source0 MutableSet需要手动唤醒线程
    source1 MutableSet具备唤醒线程的能力
    observers MutableArray
    timers MutableArray

CFRunLoopSource
    source0 MutableSet需要手动唤醒线程
    source1 MutableSet具备唤醒线程的能力

CFRunLoopObserver观测时间点
    kCFRunLoopEntry
    kCFRunLoopBeforeTimers
    kCFRunLoopBeforeSources
    kCFRunLoopBeforeWaiting
    kCFRunLoopAfterWaiting
    kCFRunLoopExit

CommonMode的特殊性
它不是实际存在的的一种Mode
是同步Source/Timer/Observer到多个Mode中的一种技术方案

9.3 RunLoop与NSTimer
1.CommonMode
2.添加到子线程，并开启子线程的RunLoop

9.4 RunLoop与多线程
线程和RunLoop一一对应的
自己创建的线程默认是没有RunLoop的

怎么实现一个常驻线程
为当前线程开启一个RunLoop
向该RunLoop中添加一个Port/Source等维持RunLoop的事件循环
启动该RunLoop

10 - 网络
小结概览

HTTP中的GET和POST方式有什么区别？

HTTPS连接建立流程是怎样的？

TCP和UDP有什么区别

请简述TCP的慢开始过程

客户端怎样避免DNS劫持？

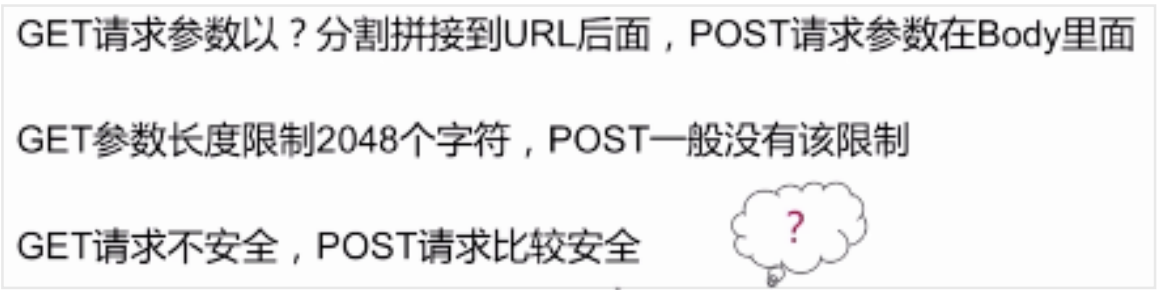
10.1HTTP(超文本传输协议)
请求/相应报文
连接建立流程
HTTP的特点

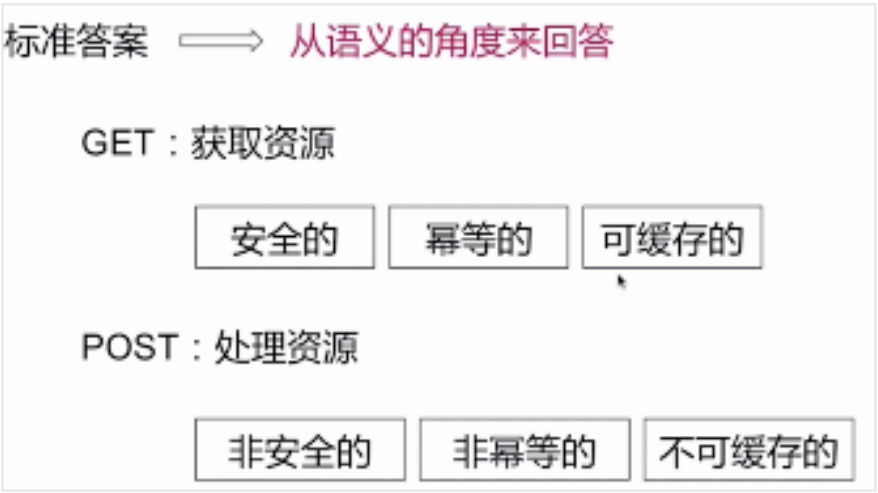
10.1.1请求报文

10.1.2相应报文

10.1.3 HTTP 的请求方式?

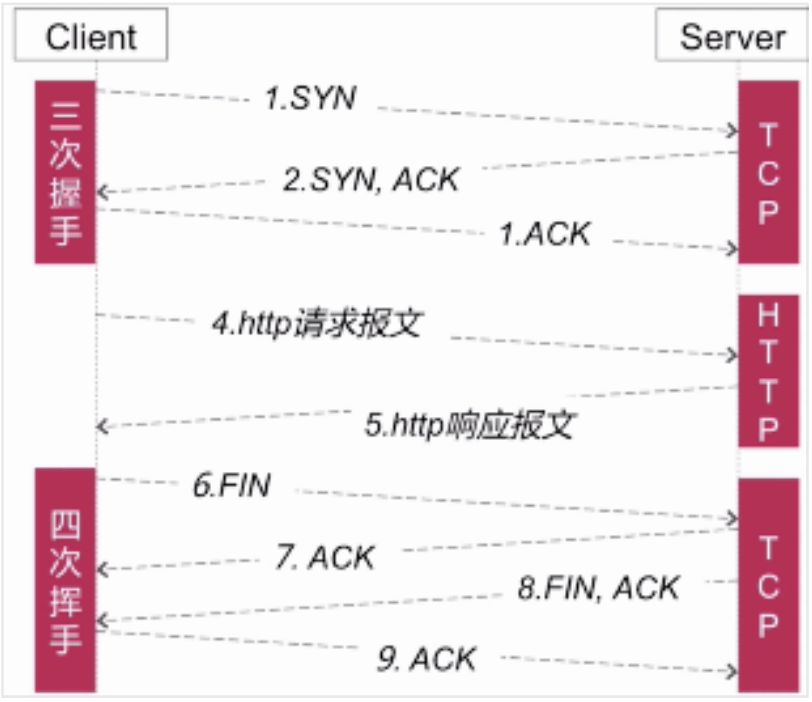
10.1.4 GET 和 POST 的区别





安全：不会引起 Server 的任何状态的变化
幂等： 多次执行的效果是相同的

10.1.5 连接建立的流程



10.1.6 HTTP的特点

无连接

HTTP的持久连接

无状态

Cookie/Session

10.1.7怎么持久连接

10.1.8 怎么判断请求结束

持久连接

怎样判断一个请求是否结束的？

- Content-length：1024
- chunked，最后会有一个空的chunked

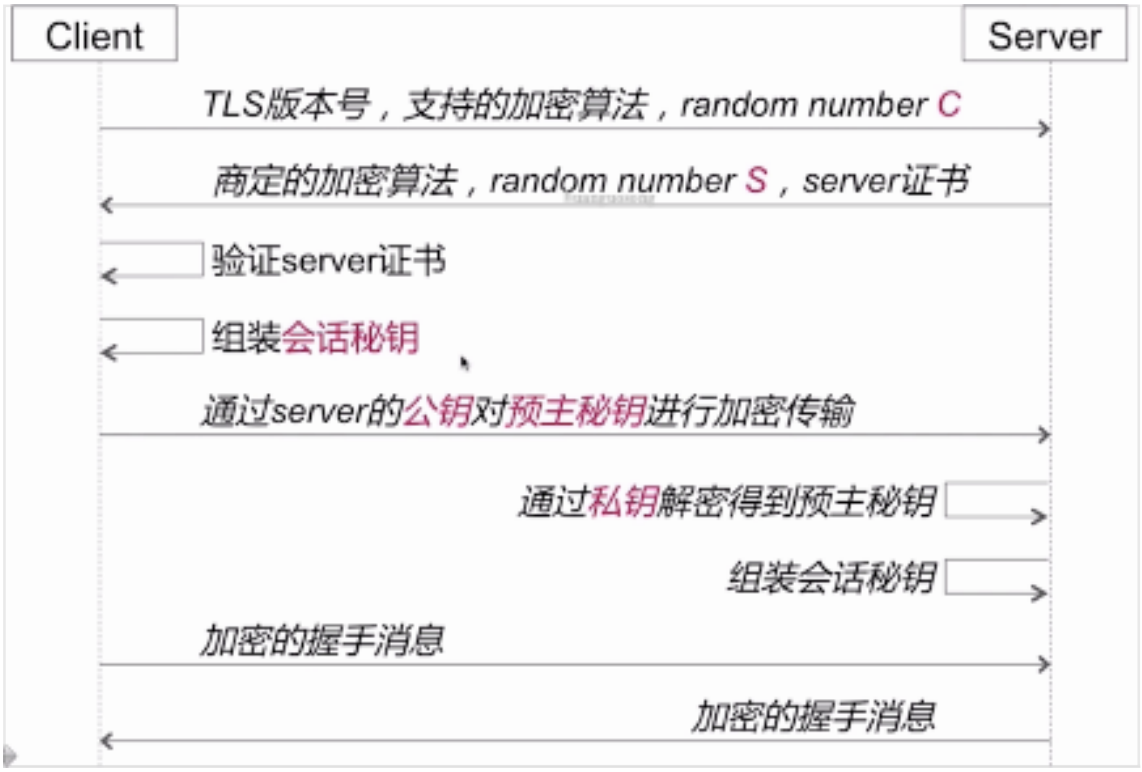
Charles抓包原理

HTTP协议

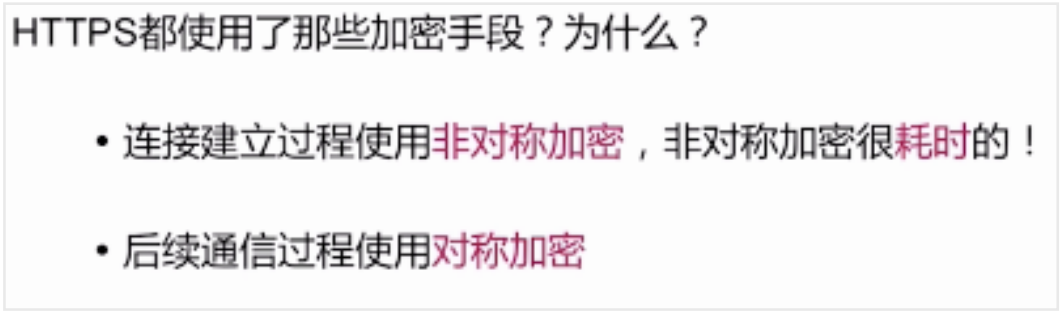
Charles抓包原理是怎样的？

中间人攻击

10.2HTTPS与网路安全
HTTPS = HTTP + SSL/TLS
10.2.1HTTPS连接建立的流程

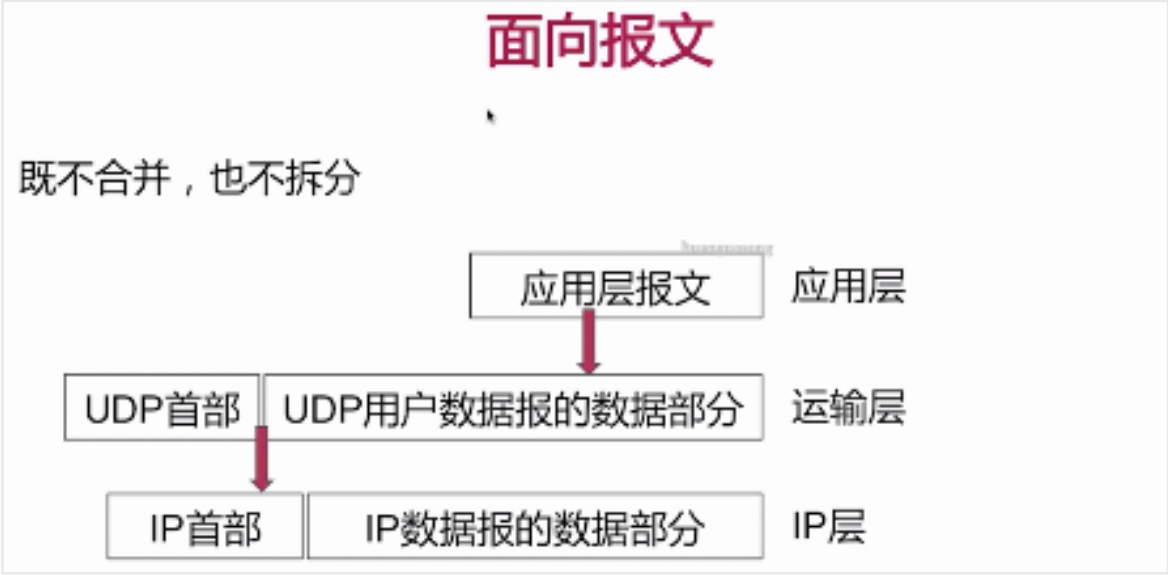


会话密钥 = random S+ random C + 预主秘钥

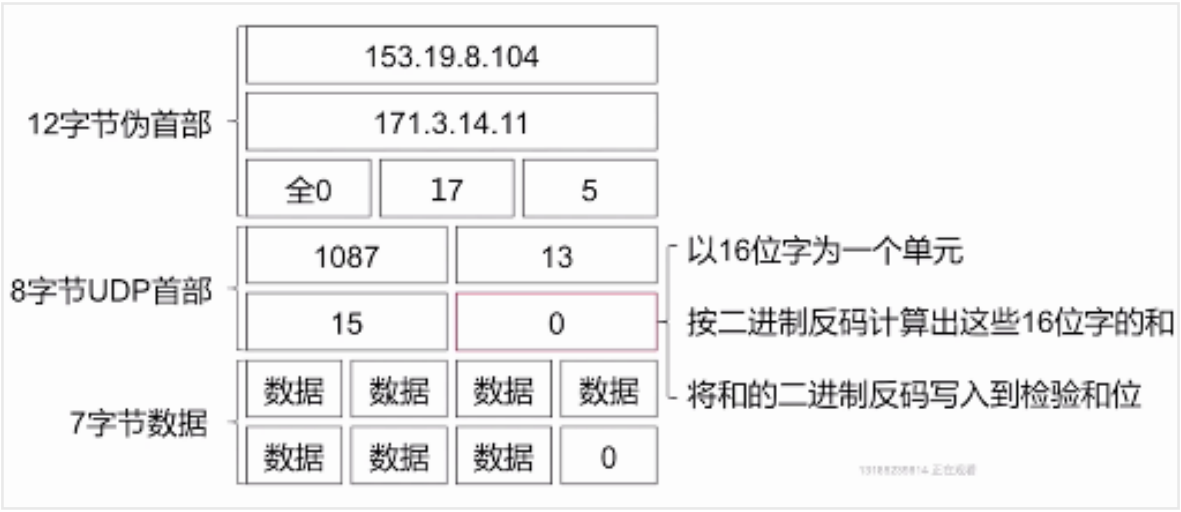


10.3 TCP(传输控制协议) 和 UDP(用户数据报协议)

UDP: 无连接 尽最大努力交付 面向报文



UDP 查错检错



TCP:

面向连接

数据传输之前需要建立连接，传输之后，释放连接

可靠传输

无差错 不丢失 不重复 按序到达

(通过停止等待协议实现)

无差错情况

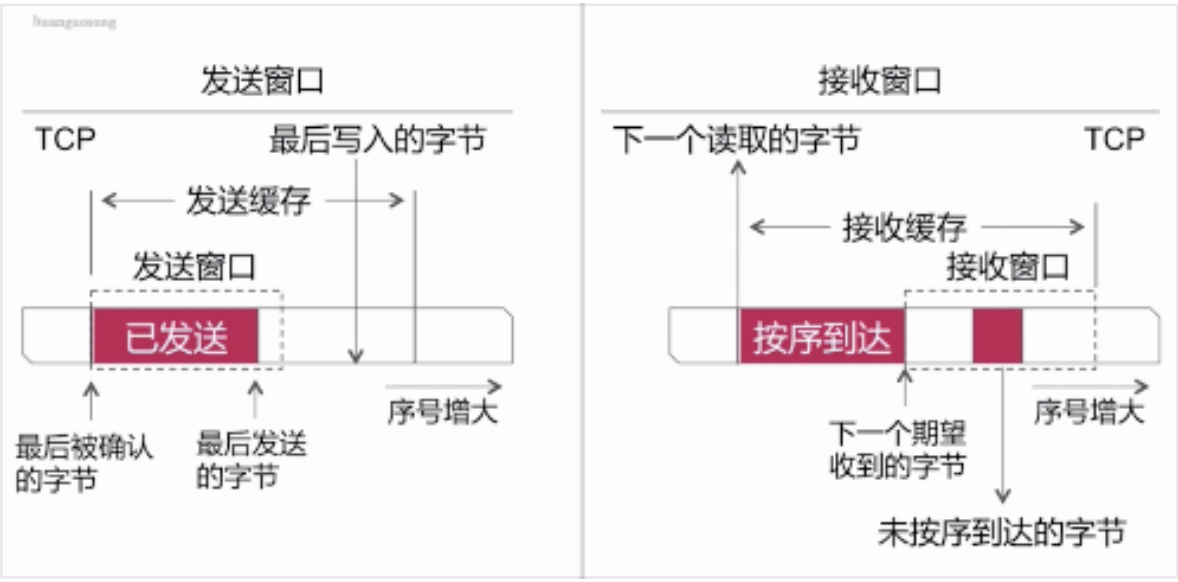
超时重传

确认丢失

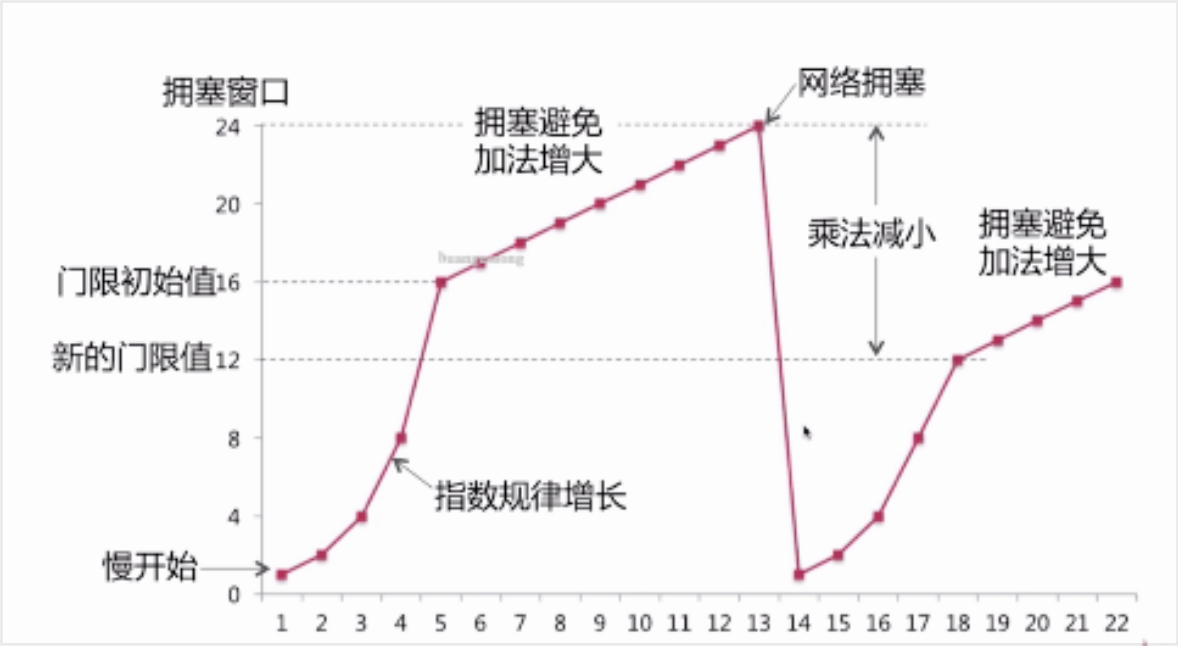
确认迟到

面向字节流

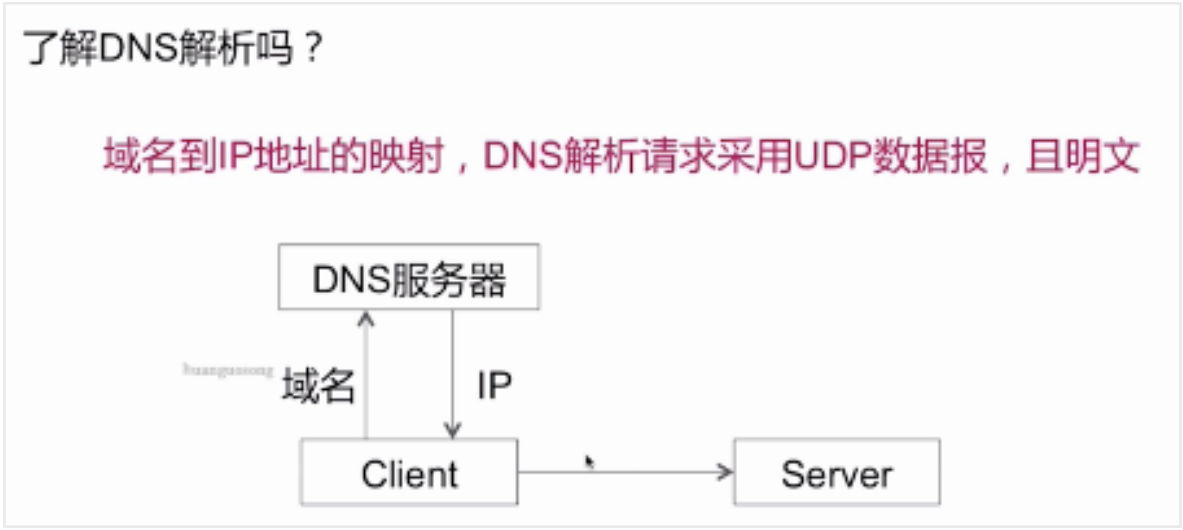
流量控制 (滑动窗口协议)

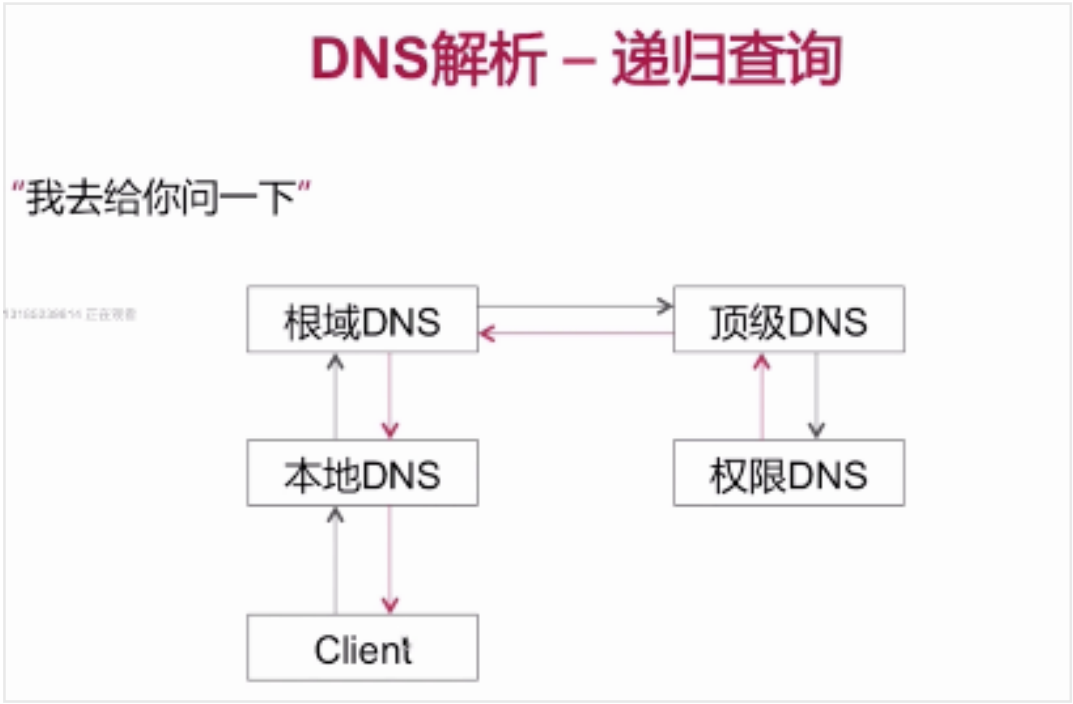


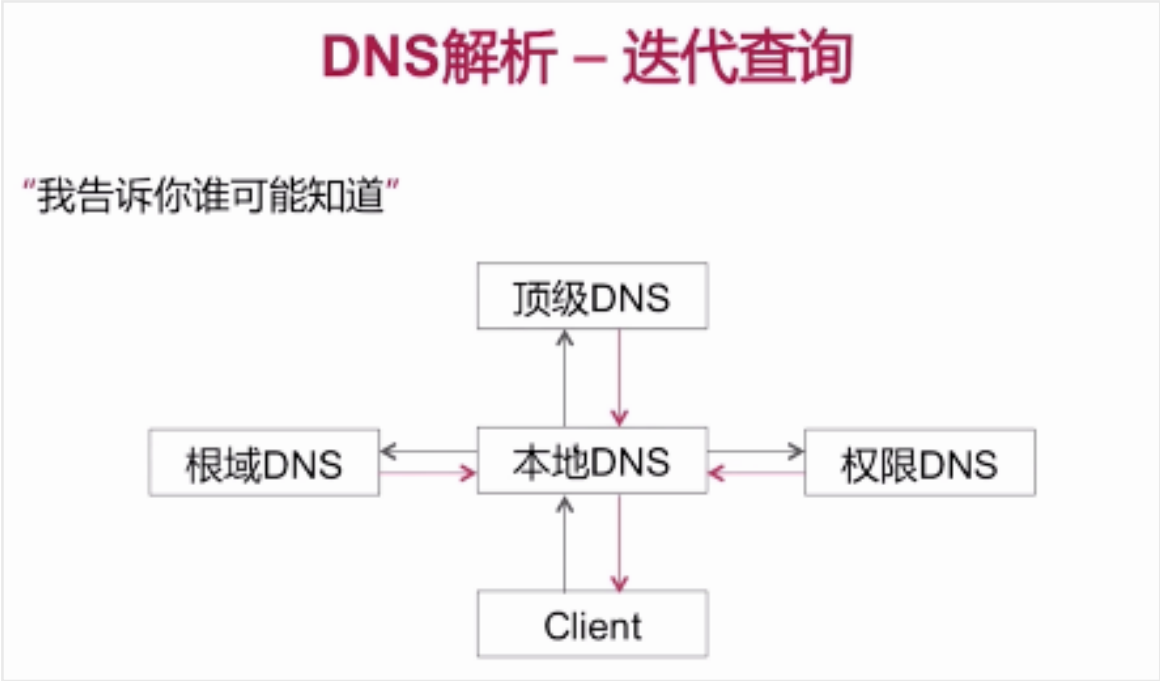
拥塞控制 (慢开始、拥塞避免；快恢复、快重传)



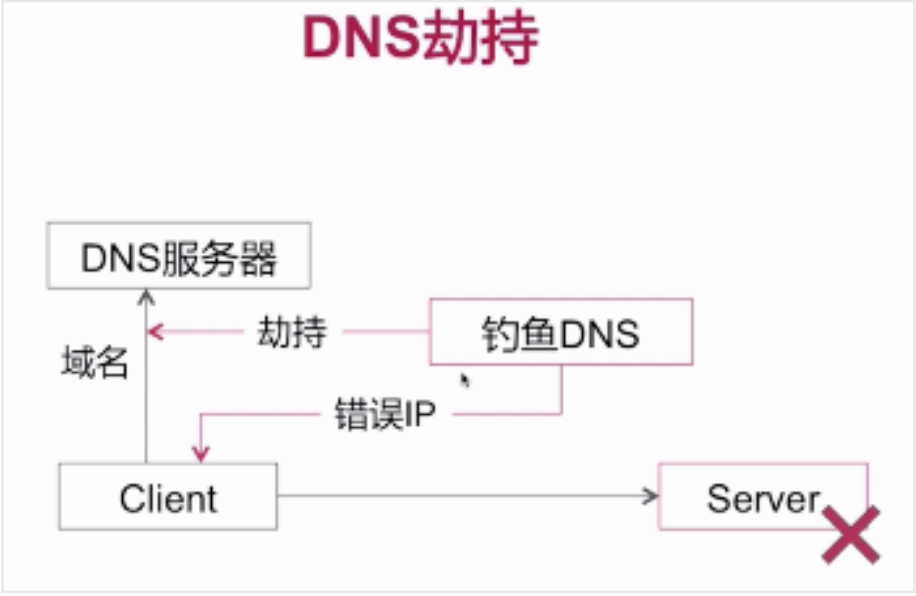
DNS解析

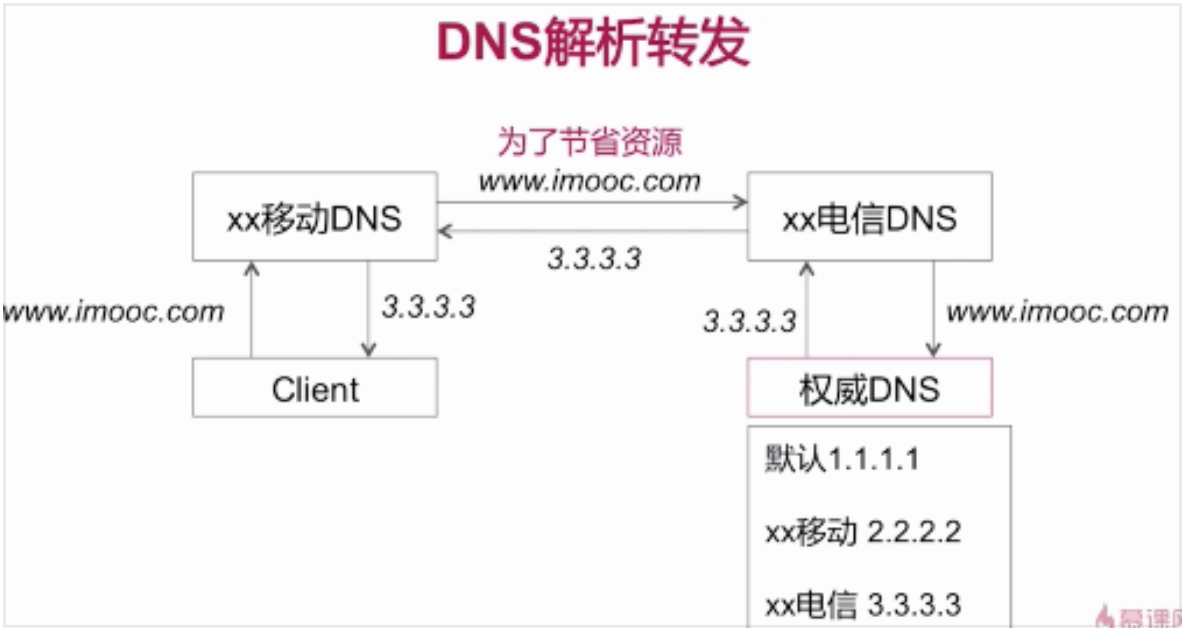

了解DNS解析吗？
域名到IP地址的映射，DNS解析请求采用UDP数据报，且明文
DNS服务器
域名
IP
Client
Server


DNS解析 – 递归查询
“我去给你问一下”
根域DNS
顶级DNS
本地DNS
权限DNS
Client



DNS解析存在哪些常见的问题?





怎么解决DNS劫持？(httpDNS、长连接)

Cookie和Session

怎样删除Cookie？

- 新cookie覆盖旧cookie
- 覆盖规则：name、path、domain等需要与原cookie一致
- 设置cookie的expires=过去的一个时间点，或者maxAge=0

怎样保证Cookie的安全？

- 对Cookie进行加密处理
- 只在https上携带Cookie
- 设置Cookie为httpOnly，防止跨站脚本攻击

Session

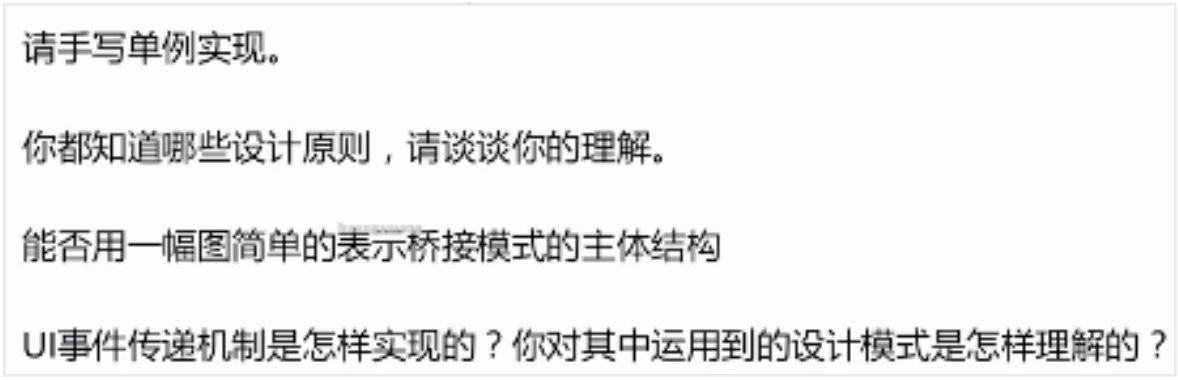
Session也是用来记录用户状态，区分用户的；状态存放在服务器端。

Session和Cookie的关系是怎样的？

- Session需要依赖于Cookie机制

11 - 设计模式

小结概述



11.1 六大设计模式

1.单一职责原则

2.开闭原则 (对修改关闭、对扩展开放)

3.接口隔离原则

使用多个专门的协议，而不是一个庞大臃肿的协议 (UITableViewDelegate DataSource)

协议中的方法应当尽量少

4.依赖倒置原则 抽象不应该依赖具体实现，具体实现可以依赖于抽象 (数据库CURD)

5.里氏替换原则 父类可以被子类无缝替换，且原有功能不受任何影响 (KVO)

6.迪米特法则

一个对象应当对其他对象有尽可能少的了解

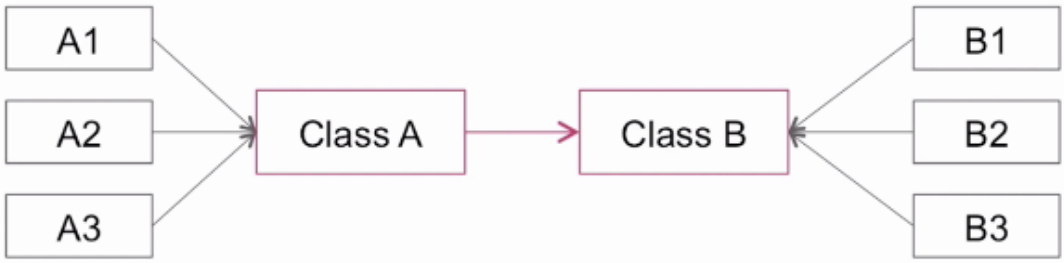
高内聚、低耦合

责任链

一个关于需求变更的问题

桥接

类构成



对象适配器

类构成

命令模式

行为参数化

降低代码重合度

12 - 架构和框架

小结概述

图片缓存

huangxxxxg

阅读时长统计

复杂页面架构

客户端整体架构

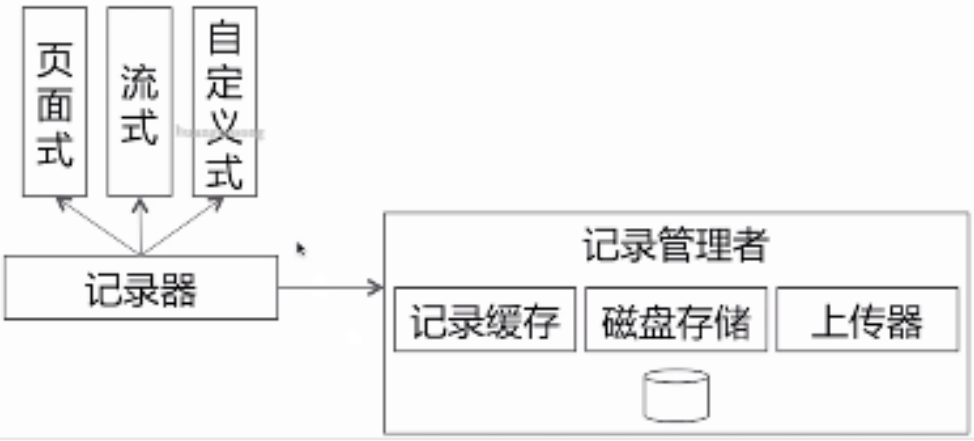
图片缓存

huangxxxxg

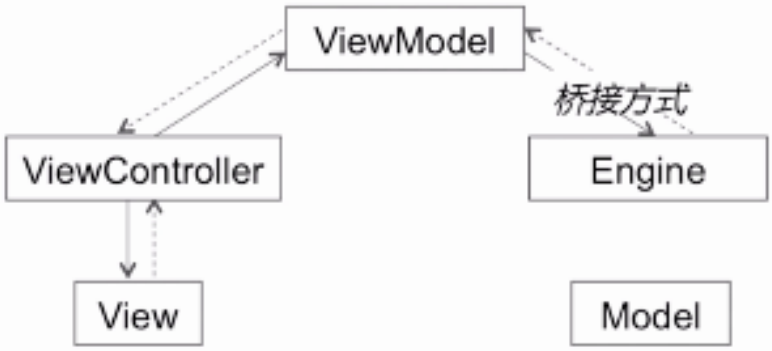
怎样设计一个图片缓存框架？

阅读时长统计

怎样设计一个时长统计框架？



整体架构



13 - 算法

小结概述

链表反转

有序数组合并

Hash算法

查找两个子视图的共同父视图

14 - 第三方库

小结概述

AFNetworking的整体结构是怎样的？

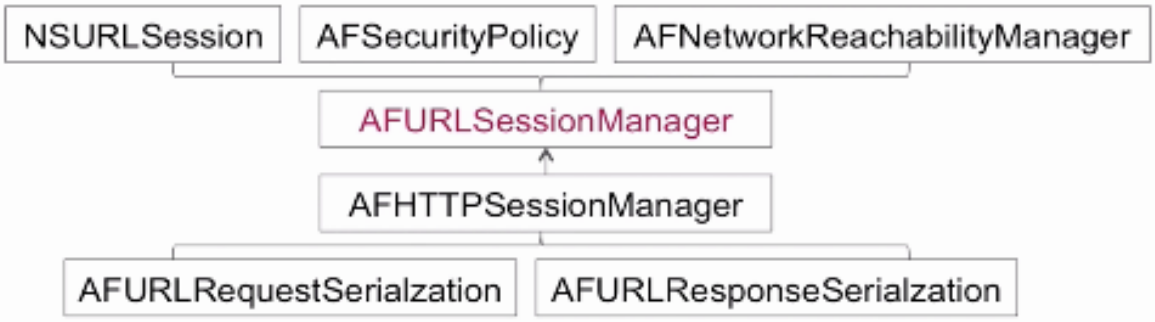
SDWebImage框架是怎样加载图片的？

AFNetworking

框架图

AFNetworking

主要类关系图



SDWebImage

架构简图