**iOS基础题**

**1. 分类和扩展有什么区别？可以分别用来做什么？分类有哪些局限性？分类的结构体里面有哪些成员？**

分类的作用：在不知道类的实现细节，不修改类实现的前提下，为类添加方法。

分类的使用场景：为系统自带的类添加方法。或者一些面向切面编程的使用。

扩展的作用：添加私有方法和属性。

扩展的使用场景：实现对外只读，对内可读写的属性，或者添加私有方法。

除了作用以及使用场景又区别以外，分类不能为类添加成员变量。但是可以通过runtime associate方法添加关联对象来实现类似属性的功能。

分类的结构体

{

const char \*name;

class\_t \*cls;

instance\_methodlist;

class\_methodlist;

protocollist;

propertylist;

}

分类的结构体并没有ivar\_list。每个分类编译后都是类似的结构，最后会生成一个数组，里面的元素就是各个分类的结构体。用来在运行时加载使用。

**2. 讲一下atomic的实现机制；为什么不能保证绝对的线程安全（最好可以结合场景来说）？**

automic的实现机制就是在属性的getter/setter方法中加锁。所以这只是保证属性的读写操作的原子性，是读写安全的，但并不能保证是线程安全的。

例如：两个线程分别对一个全局变量进行+1操作，正确的结果应该是+2。

正确的执行顺序是 A读0->A+1->A写1->B读1->B+1->B写2。但是可能存在A读0->B读0->A+1->A写1->B+1->B写1。最后结果只+1。所以为了保证线程安全，必须对读写操作打包另外加锁处理。

**3. 被weak修饰的对象在被释放的时候会发生什么？是如何实现的？知道sideTable么？里面的结构可以画出来么？**

**4. 关联对象有什么应用，系统如何管理关联对象？其被释放的时候需要手动将其指针置空么**

**5. KVO的底层实现？如何取消系统默认的KVO并手动触发（给KVO的触发设定条件：改变的值符合某个条件时再触发KVO）？**

**6. Autoreleasepool所使用的数据结构是什么？AutoreleasePoolPage结构体了解么？**

**7. 讲一下对象，类对象，元类，跟元类结构体的组成以及他们是如何相关联的？为什么对象方法没有保存的对象结构体里，而是保存在类对象的结构体里？**

对象结构体

typedef struct objc\_object \*id;

typedef struct objc\_class \*Class;

struct objc\_object

{

Class isa;

}

struct objc\_class

{

class isa;

class super\_class;

const char \*name;

long version;

long info;

long instance\_size;

struct objc\_ivar\_list \*ivars;

struct objc\_method\_list \*\*methodsList;

struct objc\_protocol\_list \*protocols;

struct objc\_chche \*cache;

struct objc\_property\_list \*properties;

}

对象实例在内存中结构：

{

isa

最上层超类成员变量

……

再上一层超类成员变量

上一层超类成员变量

成员变量

}

对象中除了首地址存放了一个isa指针外，其他空间存储的都是成员变量。查找成员变量，或者调用实例方法的时候，通过isa指针找到类对象，在类对象中找到对应成员变量偏移量或方法实现。类对象中首元素也是一个同样类型的isa指针指向元类，元类是与类对象同样的结构体，只不过元类中存放的是类方法，super\_class指向上一层父类对象。

方法调用时，会通过对象的isa指针找到类对象，然后在类对象cache和methodlist列表中找到对应的方法实现，找到则跳转执行。

如果在类对象中没有找对对应的方法，则通过super\_class到上一层超类对象查找，查找到则跳转执行，没查找到则继续查找上一层超类。

如果是类方法，则通过类对象的isa指针到元类中查找。

如果一直查找到根父类或者根元类都没能查找到对应方法，则会触发消息转发过程：

1. 动态方法解析。运行时向当前类对象发送消息+resloveInstanceMethod:，如果返回值为YES。则系统认为方法已经被添加，重新向当前对象发送消息。

2. 快速消息转发。运行时检查当前类是否实现-forwardingTargetForSelector:,如果实现则调用该方法，如果方法返回值为非nil且非self对象，则向该对象重新发送消息。

3.标准消息转发。runtime向当前对象发送消息-methodSignatureForSelector获取方法签名。如果有方法签名返回。则根据方法签名创建描述消息的NSInvocation，向当前对象发送-forwardInvocation进行消息转发。若方法签名返回nil，则向当前对象发送-doesNotRecognizeSelector：消息，应用崩溃退出。

**8. class\_ro\_t  和  class\_rw\_t 的区别？**

class\_ro\_t：存储了objc类在编译期就确定了的方法，属性，成员变量以及遵循的协议。

class\_rw\_t：存储了运行时runtime加载完成后最终的类的方法，属性，成员变量以及遵循的协议。

class\_ro\_t 结构体

{

uint32\_t flags;

     uint32\_t instanceStart;

     uint32\_t instanceSize;

     uint32\_t reserved;

     const uint8\_t \* ivarLayout;

     const char \* name;

     method\_list\_t \* baseMethodList;

     protocol\_list\_t \* baseProtocols;

     const ivar\_list\_t \* ivars;

     const uint8\_t \* weakIvarLayout;

     property\_list\_t \*baseProperties;

}

class\_rw\_t结构体

{

uint32\_t flags;

     uint32\_t version;

     const class\_ro\_t \*ro;

     method\_array\_t methods;

     property\_array\_t properties;

     protocol\_array\_t protocols;

     Class firstSubclass;

     Class nextSiblingClass;

}

objc类的结构体如下：

{

isa\_t \*isa;

Class \*super\_class;

cache\_t \*cache;

class\_data\_bits\_t \*data;

}

其中isa指向元类，super\_class指向上一级超类，cache为了加快方法的调用速度。而class\_data\_bits\_t是一个64位的指针，指向类的方法，属性，遵循的协议等信息。

在编译期，编译期将类文件，分类文件（.h.m文件）编译成单独的结构。当程序加载时，第一步加载镜像文件之后类结构中class\_data\_bits\_t指向class\_ro\_t，这个结构存储了在编译期就确定了的类的base特性。镜像加载完成之后交由runtime运行时进一步将类，分类文件组合。这时运行时用class\_ro\_t 初始化class\_rw\_t，将class\_ro\_t赋值与class\_rw\_t中对应的元素，然后加载类以及类的方法，属性，以及遵循的协议存储在class\_rw\_t的结构中，组合成最终运行时的类结构。最后类结构的class\_data\_bits\_t指针指向class\_rw\_t。

**9. iOS 中内省的几个方法？class方法和objc\_getClass方法有什么区别?**

内省是面向对象语言和环境的一个强大特性。是指对象揭示自己作为一个运行时对象的详细信息的一种能力。这些详细信息包括了对象在继承树上所处的位置。对象是否遵循特定的协议，以及对象是否能够响应特定的消息。

NSObject协议和类定义了很多内省的方法，便于查询对象的运行时信息，根据对象的特征进行识别。

这些方法有：-class，-isKindOfClass，-respondsToSelector，-conformsToProtocol，等等。

**10. 在运行时创建类的方法objc\_allocateClassPair的方法名尾部为什么是pair（成对的意思）**

因为会创建一个类对象，一个元类对象，一共两个，所以是一对。

**11. 一个int变量被\_\_block修饰与否的区别？**

讨论的是局部变量的情况下，int变量在被\_\_block修饰时，变量将被拷贝一份，从栈中地址拷贝到堆中地址，如果是对象变量，则会捕获对象引用，这样才可以在block内部修改int变量的值。因为对于block来说和block外部，处于不同的作用域。而对于集合类对象，因为对象地址是在栈中，但是对象指向堆中的内容，给集合对象添加元素，只是修改堆中的内容，所以是可以的。

**12. 为什么在block外部使用\_\_weak修饰的同时需要在内部使用\_\_strong修饰？**

这是为了防止在block代码执行过程中，\_\_weak修饰的对象因为外部已经没有其他引用，引用计数为0而被释放掉，而可能导致不可预知的错误，或者闪退。

**13. RunLoop的作用是什么？它的内部工作机制了解么？结合线程和内存管理来说明。**

**14. 哪些场景可以触发离屏渲染？**

圆角、阴影、遮罩、光栅化、视图透明图层混合复杂、抗锯齿可以触发离屏渲染。

GPU的屏幕渲染有两种方式：

On-Screen Rendering，当前屏幕渲染。是指GPU的渲染操作是在当前用于显示的屏幕缓冲区中进行的。

Off-Screen Rendering，离屏渲染。指的是GPU在当前屏幕缓冲区之外新开辟一个缓冲区进行渲染操作。

一般情况下，GPU都是将应用提交的图像动画直接渲染显示，但是对于一些复杂的图像动画并不能直接渲染叠加显示。而是需要根据command buffer分通道进行渲染之后再组合。在这组合的过程中，有些渲染通道是不会直接显示的。Masking渲染需要更多的渲染通道和合并的步骤。而这些没有直接显示到屏幕的通道缓冲区就是Off-Screen Rendering Pass。

离屏渲染有一定的性能损耗，这是因为开辟更多的渲染通道，在不同渲染通道切换需要一定的时间。而后也要组合多个通道渲染结果。

**iOS实战题**

**1. AppDelegate如何瘦身？**

**2. 反射是什么？可以举出几个应用场景么？**

**3. 有哪些场景是NSOperation比GCD更容易实现的？NSOperation与GCD的区别。**

**4. App 启动优化策略？最好结合启动流程来说，main函数前后**

**5. 你知道有哪些情况会导致app崩溃，分别可以用什么方法拦截并化解？**

**6. 你知道有哪些情况会导致app卡顿，分别可以用什么方法来避免？**

**网络题**

**1. App 网络层有哪些优化策略？**

**2. TCP为什么要三次握手，四次挥手？**

**3. 对称加密和非对称加密的区别？分别有哪些算法的实现？**

**4. HTTPS的握手流程？为什么密钥的传递需要使用非对称加密？双向认证了解么？**

**5. HTTPS是如何实现验证身份和验证完整性的？**

**6. 如何用Charles抓HTTPS的包？其中原理和流程是什么？**

**7. 什么是中间人攻击？如何避免？**

**计算机系统题**

**1. 了解编译的过程么？分为哪几个步骤？**

**2. 静态链接了解么？静态库和动态库的区别？**

**3. 内存的几大区域，各自的职能分别是什么？**

**4. static和const有什么区别？**

**5. 了解内联函数么？**

**6. 什么时候会出现死锁？如何避免？**

**7. 说一说你对线程安全的理解？**

**8. 列举你知道的线程同步策略？**

**9. 有哪几种锁？各自的原理？它们之间的区别是什么？最好可以结合使用场景来说。**

**设计模式**

**1. 除了单例，观察者设计模式以外，还知道哪些设计模式？分别介绍一下**

**2. 最喜欢哪个设计模式？为什么？**

**3. iOS SDK 里面有哪些设计模式的实践？**

**4. \*\*设计模式是为了解决什么问题的？**

**架构 & 设计题**

**1. MVC和MVVM的区别？MVVM和MVP的区别？**

**2. 面向对象的几个设计原则了解么？最好可以结合场景来说。**

**3. 可以说几个重构的技巧么？你觉得重构适合什么时候来做？**

**4. 你觉得框架和设计模式的区别是什么？**

**5.看过哪些第三方框架的源码，它们是怎么设计的？设计好的地方在哪里，不好的地方在哪里，如何改进？**

**数据结构 & 算法题**

**1. 链表和数组的区别是什么？插入和查询的时间复杂度分别是多少？**

**2. 哈希表是如何实现的？如何解决地址冲突？**

**3. 排序题：冒泡排序，选择排序，插入排序，快速排序（二路，三路）能写出那些？**

**4. 链表题：如何检测链表中是否有环？如何删除链表中等于某个值的所有节点？**

**5. 数组题：如何在有序数组中找出和等于给定值的两个元素？如何合并两个有序的数组之后保持有序？**

**6. 二叉树题：如何反转二叉树？如何验证两个二叉树是完全相等的？**