1. KVO内部实现原理
2. KVO是基于runtime机制实现的
3. 当某个类的对象第一次被观察时， 系统就会在运行期动态地创建该类的一个派生类，在这个派生类中重写基类中任何被观察属性的 setter 方法。

派生类在被重写的 setter 方法实现真正的通知机制（Person🡪 NSKVONotifying\_Person）

1. 是否可以把比较耗时的操作放在NSNotificationCenter中

如果在异步线程发的通知，那么可以执行比较耗时的操作；

如果在主线程发的通知，那么就不可以执行比较耗时的操作

3．Foundation对象与Core Foundation对象有什么区别

1> Foundation对象是OC的，Core Foundation对象是C对象

2> Foundation对象和Core Foundation对象是可以互相转换的，数据类型之间的转换

* ARC:\_\_bridge\_retained、\_\_bridge\_transfer
* 非ARC: \_\_bridge

4．不用中间变量,用两种方法交换A和B的值

A = A + B

B = A - B

A = A - B

或者

A = A^B;

B = A^B;

A = A^B;

5．简单描述下对单利模式设计的理解？

节省内存资源，一个应用就一个对象。

6．什么是动态，举例说明

1> 在程序运行过程才执行的操作

7．runtime实现的机制是什么,怎么用，一般用于干嘛. 你还能记得你所使用的相关的头文件或者某些方法的名称吗？

运行时机制，runtime库里面包含了跟类、成员变量、方法相关的API，比如获取类里面的所有成员变量，为类动态添加成员变量，动态改变类的方法实现，为类动态添加新的方法等 需要导入<objc/message.h><objc/runtime.h>

1. runtime，运行时机制，它是一套C语言库
2. 实际上我们编写的所有OC代码，最终都是转成了runtime库的东西，比如类转成了runtime库里面的结构体等数据类型，方法转成了runtime库里面的C语言函数，平时调方法都是转成了objc\_msgSend函数（所以说OC有个消息发送机制）
3. 因此，可以说runtime是OC的底层实现，是OC的幕后执行者
4. 有了runtime库，能做什么事情呢？runtime库里面包含了跟类、成员变量、方法相关的API，比如获取类里面的所有成员变量，为类动态添加成员变量，动态改变类的方法实现，为类动态添加新的方法等
5. 因此，有了runtime，想怎么改就怎么改

8．是否使用Core Text或者Core Image等？如果使用过，请谈谈你使用Core Text或者Core Image的体验。

CoreText

* 随意修改文本的样式
* 图文混排(纯C语言)
* 国外:Niumb

Core Image(滤镜处理)

\* 能调节图片的各种属性(对比度, 色温, 色差等)

9．NSNotification和KVO的区别和用法是什么？什么时候应该使用通知，什么时候应该使用KVO，它们的实现上有什么区别吗？如果用protocol和delegate（或者delegate的Array）来实现类似的功能可能吗？如果可能，会有什么潜在的问题？如果不能，为什么？（虽然protocol和delegate这种东西面试已经面烂了…）

通知比较灵活(1个通知能被多个对象接收, 1个对象能接收多个通知),

代理比较规范，但是代码多(默认是1对1)

KVO性能不好(底层会动态产生新的类)，只能监听某个对象属性的改变, 不推荐使用(1个对象的属性能被多个对象监听, 1个对象能监听多个对象的其他属性)

更详细参考：

http://blog.csdn.net/dqjyong/article/details/7685933

10．Block内部的实现原理

Objective-C是对C语言的扩展，block的实现是基于指针和函数指针

11. 有两个数组a,b，大小都为n,数组元素的值任意，无序；

要求：通过交换a,b中的元素，使数组a元素的和与数组b元素的和之间的差最小

算法题，要么靠功底，要么百度