# KVC-KVO

## KVC的底层实现?

```
当一个对象调用setValue方法时，方法内部会做以下操作：
①检查是否存在相应key的set方法，如果存在，就调用set方法
②如果set方法不存在，就会查找与key相同名称并且带下划线的成员属性，如果有，则直接给成员属性赋值
③如果没有找到_key,就会查找相同名称的属性key，如果有就直接赋值
④如果还没找到，则调用valueForUndefinedKey:和setValue:forUndefinedKey:方法。
这些方法的默认实现都是抛出异常，我们可以根据需要重写它们。
```

## KVO的底层实现?

```
- kvo基于runtime机制实现。
- 使用了isa 混写（isa-swizzling），当一个对象(假设是person对象，person的类是MYPerson)的属性值(假设person的age
,而这两个方法内部会主动调用监听者内部的 - (void)observeValueForKeyPath 这个方法。
- 想要看到NSKVONotifying_MYPerson很简单，在self.person.age = 20; 这里打断点，在调试区域就能看到 _person->NSO
```

## 什么是KVO和KVC?

```
答：KVC:键 – 值编码 使用字符串直接访问对象的属性。
KVO:键值观察机制，它提供了观察某一属性变化的方法
```

## KVO的缺陷?

KVO是一个对象能够观察另外一个对象的属性的值，并且能够发现值的变化。前面两种模式更加适合一个controller与任何其他的对象进行通信，而KVO更加适合任何类型的对象侦听另外一个任意对象的改变（这里也可以是controller，但一般不是controller）。这是一个对象与另外一个对象保持同步的一种方法，即当另外一种对象的状态发生改变时，观察对象马上作出反应。它只能用来对属性作出反应，而不会用来对方法或者动作作出反应。

优点:

1.能够提供一种简单的方法实现两个对象间的同步。例如：model和view之间同步;

2.能够对非我们创建的对象，即内部对象的状态改变作出响应，而且不需要改变内部对象（SKD对象）的实现;

3.能够提供观察的属性的最新值以及先前值;

4.用key paths来观察属性，因此也可以观察嵌套对象;

5.完成了对观察对象的抽象，因为不需要额外的代码来允许观察值能够被观察

缺点：

1.我们观察的属性必须使用strings来定义。因此在编译器不会出现警告以及检查；

2.对属性重构将导致我们的观察代码不再可用；

3.复杂的“IF”语句要求对象正在观察多个值,这是因为所有的观察代码通过一个方法来指向；

4.当释放观察者时需要移除观察者。