iOS (http://lib.csdn.net/base/ios)

iOS (http://lib.csdn.net/base/ios) - 设计模式 (http://lib.csdn.net/ios/node/673) - 设计模式 (http://lib.csdn.net/ios/knowledge/1462)



# 移动开发(IOS) – Objective-C-10-KVC、谓词、KVO与通知

作者：Demon_JiAo (http://my.csdn.net/Demon_JiAo)

## 移动开发(IOS) – Objective-C-10-KVC、谓词、KVO与通知

By docoder (http://docoder.com/author/alexlee/) in 博客 (http://docoder.com/category/blogs/), 学习 (http://docoder.com/category/blogs/learn/) on 2014/05/31

**1.KVC**

1.1.基本概念

1.1.1.KVC 是 KeyValue Coding 的简称,它是一种可以直接通过字符串的名字( key ) 来访问类属性的机制。

1.1.2.使用该机制不需要调用存取方法和变量实例就可访问对象属性。

1.1.3.本质上讲,键-值编码定义了你的程序存取方法需要实现的样式及方法签名。

1.1.4.在应用程序中实现键-值编码兼容性是一项重要的设计原则。存取方法可以加强合适的数据封装,而键-值编码方法在多数情况下可简化程序代码。

1.1.5.键-值 编码方法在 Objective-C 非标准协议(类目) NSKeyValueCoding 中 被声明,默认的实现方法由 NSObject 提供。

1.1.6.键-值编码支持带有对象值的属性,同时也支持纯数值类型和结构。非对象参数和返回类型会被识别并自动封装/解封。

1.1.7.使用 KVC 为对象赋值或者取值时，需要知道准确的键值， 相比较点语法，KVC 是一种间接的传递方式，这种方式有利于对象解耦，让对象彼此之间的耦合度不会太高。

1.2.设置和访问

1.2.1.键/值编码中的基本调用包括 -valueForKey: 和 -setValue:forkey: 这两个方法,它们以字符串的形式向对象发送消息,字符串为属性名，即键：

```
1                          Person *jack = [[Person alloc] init];
2       NSMutableString *name = [[NSMutableString alloc] initWithFormat:@"jack"];
3                  [jack setValue:name forKey:@"name"]; //通过KVC设值
4          NSLog(@"Jack's name : %@", [jack valueForKey:@"name"]);//通过KVC取值
```

1.2.2.是否存在 setter、getter 方法, 若存在优先调用相应方法；若不存在,它将在内部查找名为 _key 或 key 的实例变量。

1.2.3.通过 KVC 设置对象，此对象会 retain。

1.2.4.通过 setValue:forKey: 设置对象的值,或通过 valueForKey 来获取对象的值时,如若对象的实例变量为基本数据类型时（char、int、float、BOOL）,我们需要对数据进行封装。

1.2.5.赋值语句 setValue:forKey: 是给对象当前的属性赋值，而 setValue:forKeyPath: 是按照对象的层级关系为其中的属性赋值

。 forKeyPath可以替代forKey，但是forKey不能替代forKeyPath。

1.2.6.setValuesForKeysWithDictionary: 可以从 plist 文件中读取对应的数据字典，对对象属性赋值。

1.3.批处理

```
                        //同时获取 Student 的 age 和 name
        NSArray *keys = [NSArray arrayWithObjects:@"name", @"age", nil];
       NSDictionary *dict = [student dictionaryWithValuesForKeys:keys];
                        //同时设置 Student 的 age 和 name
        NSArray *keys = [NSArray arrayWithObjects:@"name", @"age", nil];
NSArray *values = [NSArray arrayWithObjects:@"MJ", [NSNumber numberWithInt:16], nil];
   NSDictionary *dict = [NSDictionary dictionaryWithObjects:values forKeys:keys];
                [student setValuesForKeysWithDictionary:dict];
```

1.4.路径

除了通过键设值或取值外, 键/值编码还支持指定路径设值或取值,像文件系统一样, 用 “ . ” 号隔开:

```
    [book setValue:@"比尔" forKeyPath:@"author.name (http://author.name)"];
```

```
        NSNumber *price=[book valueForKeyPath:@"relativeBooks.price"]
```

1.4.数组的整体操作

如果向 NSArray 请求一个键值,它实际上会查询数组中的每个对象来查找这个键值, 然后将查询结果打包到另一个数组中并返回给你:

```
                    // 获取 Student 中所有 Book 的 name
        NSArray *names = [student.books valueForKeyPath:@"name"]; 或者
NSArray *names = [student valueForKeyPath:@"books.name (http://books.name)"];
            //注意：不能在键路径中为数组添加索引，比如 @"books[0].name"
```

1.5.KVC的简单运算

```
                                //count
      NSString *count = [book valueForKeyPath:@"relativeBooks.@count"];
                        NSLog(@"count : %@", count);
                                //sum
    NSString *sum = [book valueForKeyPath:@"relativeBooks.@sum._price"];
                            NSLog(@"sum : %@", sum);
                                //avg
    NSString *avg = [book valueForKeyPath:@"relativeBooks.@avg._price"];
                            NSLog(@"avg : %@", avg);
                                //min
    NSString *min = [book valueForKeyPath:@"relativeBooks.@min._price"];
```

```
                                NSLog(@"min : %@", min);
                                        //max
        NSString *max = [book valueForKeyPath:@"relativeBooks.@max._price"];
                                NSLog(@"max : %@", max);
```

**2.谓词**

2.1.基本概念

cocoa 中提供了 NSPredicate 类,指定过滤器的条件。将符合条件的对象保留 下来。

2.2.创建谓词:

```
                                    // 设置谓词条件
      NSPredicate *predicate = [NSPredicate predicateWithFormat:@"age <= 28"];
                            for (Person *person in array) {
                                // 表示指定的对象是否满足谓词条件
                        if ([predicate evaluateWithObject:person]) {
                NSLog(@"person's name : %@", person.name (http://person.name));
                                        }
                                    }
                                // 返回一个符合谓词条件的数组
          NSArray *newArray = [array filteredArrayUsingPredicate: predicate];
                            for (Person *person in newArray) {
                NSLog(@"person's name : %@", [person valueForKey:@"_name"]);
                                        }
```

2.3.格式占位符

```
                                    // 格式占位符号
      NSPredicate *pre = [NSPredicate predicateWithFormat:@" age <= %d", 30];
            NSArray *array2 = [array filteredArrayUsingPredicate:pre];
                            for (Person *person in array2) {
                NSLog(@"person's name : %@", [person valueForKey:@"_name"]);
                                        }
```

2.4.运算符

2.4.1.逻辑运算符

```
                                // 运算符号 && AND || OR
                            NSPredicate *pre = [NSPredicate
                predicateWithFormat:@"name > 'bruse' && age < %d", 30];
                NSArray *array = [array filteredArrayUsingPredicate:pre];
```

2.4.2.IN (http://2.4.2.IN)

```
                                //注意字符串一定要添加''
                            NSPredicate *pre = [NSPredicate
predicateWithFormat:@"self.name (http://self.name) IN {'rose', 'bruse'}"];//self.可以省
                    NSArray *array = [array filteredArrayUsingPredicate:pre];
                NSLog(@"person's name : %@", [array valueForKey:@"_name"]);
```

2.4.3.以...开始: BEGINSWITH

```
// BEGINSWITH 检查某个字是否以...开头
NSPredicate *pre = [NSPredicate
predicateWithFormat:@"self.name (http://self.name) BEGINSWITH 'J'"];
NSArray *array = [array filteredArrayUsingPredicate:pre];
NSLog(@"person's name : %@", [array valueForKey:@"name"]);
```

2.4.4.以...结束: ENDSWITH

```
// ENDSWITH 检查某个字符是以...结尾
NSPredicate *pre = [NSPredicate
predicateWithFormat:@"self.name (http://self.name) endswith 'e'"];
NSArray *array = [array filteredArrayUsingPredicate:pre];
NSLog(@"person's name : %@", [array valueForKey:@"name"]);
```

2.4.5.包含: CONTAINS

```
// CONTAINS 检查包含某个字符
NSPredicate *pre = [NSPredicate
predicateWithFormat:@"self.name (http://self.name) CONTAINS '小'"];
NSArray *array = [array filteredArrayUsingPredicate:pre];
NSLog(@"person's name : %@", [array valueForKey:@"name"]);
```

2.4.6.like

```
// like *: 匹配任意多个字符 ?:表示一个字符 (正则)
NSPredicate *pre = [NSPredicate predicateWithFormat:@"name like '?a*'"];
NSArray *array = [array filteredArrayUsingPredicate:pre];
NSLog(@"person's name : %@", [array valueForKey:@"name"]);
```

**3.KVO**

3.1.基本概念

3.1.1.Key Value Observing,直译为:基于键值的观察者。它提供一种机制,当 指定的对象的属性被修改后,则对象就会接受到通知。简单的说就是每次指定的被 观察的对象的属性被修改后,KVO就会自动通知相应的观察者了。

3.1.2.与 NSNotification不同,键-值观察中并没有所谓的中心对象来为所有观察者 提供变化通知。取而代之地,当有变化发生时,通知被直接发送至处于观察状态的 对象。NSObject提供这种基础的键-值观察实现方法。

3.1.3.你可以观察任意对象属性,包括简单属性,对一或是对多关系。对多关系的观 察者将会被告知发生变化的类型-也就是任意发生变化的对象。

3.1.4.键-值观察为所有对象提供自动观察兼容性。你可以通过禁用自动观察通知并实现手动通知来筛选通知。

3.1.5.不能观察已经被释放的对象，如果要观察，需要是强应用对象，或者被其他对象强应用的对象。

3.2.注册观察者

为了正确接收属性的变更通知,被观察者必须首先调用 addObserver:forKeyPath:options:context: 方法进行注册:

```
/* anObserver ： 监听器对象
* keyPath ： 监听的属性
* options ： 决定了当属性改变时，要传递什么数据给监听器
*/
```

```
-(void)addObserver:(NSObject *)anObserver forKeyPath:(NSString
*)keyPath options:(NSKeyValueObservingOptions)options context:(void *)context

/* 使用NSKeyValueObservingOptionOld选项,可以将改变之前的值传递给观察者。(以变更字典中的一个项的
 * 指定 NSKeyValueObservingOptionNew选项,可以将改变的新值传递给观察者。
 * 可以使用逐位“|”这两个常量，来指定同时传递上述两种类型的值。
 */
[_child addObserver:self forKeyPath:@"tired"
options:NSKeyValueObservingOptionOld|NSKeyValueObservingOptionNew context:nil];
```

3.3.接受变更通知

当对象的一个被观察属性发生变动时,观察者收到一个 observeValueForKeyPath:ofObject:change:context:消息。所有观察者都必须 实现这一方法:

```
/*
 * keyPath : 键路径
 * object : 被观察者
 * change : 包含变更细节的字典
 * context : 注册观察者时提交的上下文指针,可以为任意类型的参数
 */
- (void)observeValueForKeyPath:(NSString *)keyPath
ofObject:(id)object
change:(NSDictionary *)change
context:(void *)c{
if ([keyPath isEqual:@"key"]) {
NSLog(@"");
}
}
// observeValueForKeyPath 方法是 NSObject 的分类，意味着可以观察任何对象。
```

3.4.移除观察者

当观察者销毁时，或达到目的无需再使用 KVO 时，应该将观察者移除:

```
[_child removeObserver:self forKeyPath:@"key"];
```

**4.通知**

4.1.与 KVO 不同:

4.1.1.自定触发通知，不像 KVO，属性值一经改变便触发通知。

4.1.2.回调方法自定，不像 KVO，需要重写一个方法。

4.1.3.观察者和被观察者都可以没有对方的引用，两者可以毫无关系。

4.2.监听通知

```
/*
 * self : 观察者对象为自身
 * @selector(notificationAction:) : 当收到通知时，调用notificationAction: 方法
 * @"hapyValueNotification" : 监听通知名为@"hapyValueNotification"
 * nil : 传递参数为nil
```

```
                                        */
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(notificationAction:)
                                                 name:@"hapyValueNotification"
                                               object:nil];
```

```
                                    //收到通知时的回调方法
- (void)notificationAction:(NSNotification *)notification {
                    Children *child = notification.object;
                            [self playWith:child];
                                    }
                - (void)playWith:(Children *)child {
                            child.hapyValue = 100;
                                    };
```

4.3.发送通知

```
                                        /*
            * @"hapyValueNotification" : 发送通知名为@"hapyValueNotification"
                                * self : 传递参数为self,自身对象
                                        */
[[NSNotificationCenter defaultCenter] postNotificationName:@"hapyValueNotification"
                                        object:self];
```

4.4.移除通知

当观察者销毁时，或达到目的无需再使用 通知 时，应该将通知移除:

```
    //移除当前对象上指定的通知，通知名：@"hapyValueNotification"
[[NSNotificationCenter defaultCenter] removeObserver:self
        name:@"hapyValueNotification" object:nil];
```

```
                                    //移除当前对象上所有的通知
                [[NSNotificationCenter defaultCenter] removeObserver:self];
```

查看原文>> (http://blog.csdn.net/Demon_JiAo/article/details/51055403)

**0**

**看过本文的人也看了：**

- iOS知识结构图 (http://lib.csdn.net/base/ios/structure)
- 【iOS与EV3混合机器人编程系列之六】i... (http://lib.csdn.net/article/ios/42131)
- Cocoa 框架 For iOS(二)对象的分配初...
- IOS设计模式之一（MVC模式，单例模式... (http://lib.csdn.net/article/ios/42103)
- iOS开发中尝试下超级表单页面和浮动Te... (http://lib.csdn.net/article/ios/43214)
- iOS 第三章设计模式

(http://lib.csdn.net/article/ios/44699) (http://lib.csdn.net/article/ios/42133)

**发表评论**

输入评论内容

发表

**0个评论**