Noveo
Software University

9. KVC, KVO

Noveo University — iOS

Александр Горбунов

Сегодня

- Key-Value Coding
- Key-Value Observing
- `NSPredicate`

Key-Value Coding

KVC — механизм, позволяющий обращаться к значению (Value) свойства объекта, используя строковые идентификаторы — ключи (Key). Ключи можно выстраивать в цепочку (KeyPath), обращаясь к вложенным свойствам. KVC основывается на методах протокола `NSKeyValueCoding`.

Key-Value Coding

... используя строковые идентификаторы...

Это позволяет решить к какому свойству обратиться *во время выполнения*.

```
NSString *valueB = [myObject valueForKey:@"someString"];
NSString *valueC = [myObject valueForKeyPath:@"parentObject.someString"];
```

Key-Value Coding

Нужно создать массив объектов `Person`:

```
@interface Person : NSObject
@property (nonatomic, copy) NSString *name;
@property (nonatomic, strong) NSNumber *age;
...
@end
```

из полученного JSON:

```
[
    {
        "name": "Alice",
        "age": 22,
        ...
    },
    {
        "name": "Bob",
        "age": 24,
        ...
    }
]
```

Key-Value Coding

Вручную работаем с каждым свойством:

```
NSArray <NSDictionary *> *personsDescriptions = ...;
NSMutableArray <Person *> *persons = ...;

for (NSDictionary *personDescription in personsDescriptions) {
    Person *person = [[Person alloc] init];

    person.name = personDescription[@"name"];
    person.age = personDescription[@"age"];
    ...
}
```

Key-Value Coding

Автоматический проход по всем свойствам:

```
NSArray <NSDictionary *> *personsDescriptions = ...;
NSMutableArray <Person *> *persons = ...;

for (NSDictionary *personDescription in personsDescriptions) {
    Person *person = [[Person alloc] init];

    for (NSString *key in personDescription.allKeys) {
        [person setValue:personDescription[key] forKey:key];
    }
}
```

Key-Value Coding

```
@property NSString *name;
@property NSNumber *age;
...
```

```
{
    "name": ...,
    "age": ...,
    ...
}
```

Нужна осторожность: **мы отвечаем за совпадение ключей**!

Key-Value Coding

KVC позволяет включать в KeyPath не только простые свойства, но и коллекции объектов. На выходе мы получим коллекцию, аггрегирующую значения свойства у всех элементов коллекции.

```
@interface Person : NSObject
@property (nonatomic, copy) NSString *name;
@property (nonatomic, strong) NSNumber *age;
@end
```

```
@interface Department : NSObject
@property (nonatomic, copy) NSArray <Person *> *staff;
@end
```

```
NSArray *allNames = [department valueForKeyPath:@"staff.name"];

// allNames: @[@"Alice", @"Bob"]
```

Key-Value Coding

KVC предоставляет несколько операторов для обработки значений элементов коллекции: `@avg`, `@max`, `@min`, `@distinctUnionOfObjects` и др.

```
@interface Person : NSObject
@property (nonatomic, copy) NSString *name;
@property (nonatomic, strong) NSNumber *age;
@end
```

```
@interface Department : NSObject
@property (nonatomic, copy) NSArray <Person *> *staff;
@end
```

```
NSNumber *avgAge = [department valueForKeyPath:@"staff.@avg.age"];

// avgAge: @23
```

NSPredicate

`NSPredicate` позволяет делать достаточно хитрые запросы (с фильтрами и выражениями), в том числе при доступе к значениям через KVC. Синтаксис выражений похож на SQL запросы и может использовать регулярные выражения.

```
@interface Person : NSObject
@property (nonatomic, copy) NSString *name;
@property (nonatomic, strong) NSNumber *age;
@end
```

```
@interface Department : NSObject
@property (nonatomic, copy) NSArray <Person *> *staff;
@end
```

```
NSPredicate *predicate = [NSPredicate predicateWithFormat:@"name CONTAINS 'Bob'"];
NSArray <Person *> *bobs = [self.staff filteredArrayUsingPredicate:predicate];
```

Key-Value Coding

По умолчанию обращение по несуществующему ключу вызывает исключение, поэтому нужно реализовать одну из политик:

- Гарантированно не обращаться к несуществующим ключам
- Переопределить методы, обрабатывающие обращение к несуществующим ключам (`valueForUndefinedKey:` и `setValue:forUndefinedKey:`)

Key-Value Coding

Значения при использовании KVC имеют тип `id`, поэтому:

- При обращении к скаляру (`BOOL`, `int`, `float`, ...) через KVC, он автоматически оборачивается в `NSNumber`.
- При обращении к структуре (`struct`) через KVC, она автоматически оборачивается в `NSValue`.
- При присвоении значения `nil` скаляру через KVC, вызывается метод `setNilValueForKey:`, в котором нужно определить желаемое поведение.

Key-Value Coding

Для поддержки работы KVC нужно правильно именовать методы-акцессоры:

- Имя геттера должно совпадать с именем свойства
- Имя геттера `BOOL`-свойства должно иметь префикс `is`
- Имя сеттера должно иметь префикс `set`

```
@property (nonatomic, copy) NSString *name;
```

```
- (NSString *)name {...}
- (void)setName:(NSString *)name {...}
```

Key-Value Observing

KVO — механизм, автоматизирующий нотификацию об изменениях свойств объекта. KVO является стандартной (встроенной) реализацией паттерна Observer.

Key-Value Observing

Объект, который хочет получать KVO-нотификации должен:

- Подписаться на нотификации, указав получателя, объект и KeyPath для отслеживания, контекст, дополнительные опции (например нотификация до или после изменения значения).
- Обрабатывать полученные нотификации, проверив что они действительно должны быть обработаны. (Родительский класс мог так же подписаться на нотификации и их нужно передать в `super`).
- По необходимости или в конце жизненного цикла отписаться от всех ранее созданных подписок.

Key-Value Observing

KVO — мощный механизм, который не терпит ошибок...

Key-Value Observing

- Создаём контекст. Он поможет определить, принадлежат ли нам полученные нотификации.

```
static void *const myContext = (void *)&myContext;
```

- Подписываемся на изменения значений. (например в `init`)

```
[self.myPerson addObserver:self forKeyPath:@"name"
    options:NSKeyValueObservingOptionNew context:myContext];
```

- Отписываемся от нотификаций. (например в dealloc)

```
[self.myPerson removeObserver:self forKeyPath:@"name" context:myContext];
```

Key-Value Observing

- Получаем и обрабатываем нотификации.

```
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
    change:(NSDictionary *)change context:(void *)context
{
    if (context != myContext) {
        return [super observeValueForKeyPath:keyPath ofObject:object
            change:change context:context];
    }

    Person *person = (Person *)object;
    NSLog(@"New value is %@.", person.name);
}
```

Key-Value Observing

KVO поддерживает свойства, не имеющие под собой ivar, например для которых значение вычисляется в геттере.

```
@property (nonatomic, copy) NSString *firstName;
@property (nonatomic, copy) NSString *lastName;
@property (nonatomic, readonly) NSString *fullName;
```

```
- (NSString *)fullName
{
    return [NSString stringWithFormat:@"%@ %@", self.firstName, self.lastName];
}
```

Key-Value Observing

KVO поддерживает свойства, не имеющие под собой ivar, например для которых значение вычисляется в геттере.

```
+ (NSSet *)keyPathsForValuesAffectingValueForKey:(NSString *)key
{
    NSSet *keyPaths = [super keyPathsForValuesAffectingValueForKey:key];

    if ([key isEqualToString:@"fullName"]) {
        NSArray *affectingKeys = @[@"lastName", @"firstName"];
        keyPaths = [keyPaths setByAddingObjectsFromArray:affectingKeys];
    }

    return keyPaths;
}
```

Key-Value Observing

KVO поддерживает отслеживание изменений содержимого коллекций. Однако для обеспечения работоспособности этого механизма, изменения должны происходить не напрямую в коллекции, а через прокси-объект:

```
@property (nonatomic) NSArray <Person *> *staff;
```

```
NSMutableArray <Person *> *mutableStaff = [self mutableArrayValueForKey:@"staff"];
[mutableStaff addObject:newEmployee];
```

Key-Value Observing

- Все нотификации приходят в один метод-обработчик.
- При переименовании свойства мы ответственны за обновление всех KeyPath-строк.
- Каждый объект должен обрабатывать только "свои" нотификации. Нельзя отдавать в `super` свою нотификацию, не стоит обрабатывать нотификации, предназначенные для `super`.
- Если кто-то использует свойство для KVO, к нему нельзя обращаться через ivar, или нужно вручную нотифицировать об изменении значения (`willChangeValueForKey:` / `didChangeValueForKey:`).

Key-Value Observing

- Нельзя не отписываться от нотификаций.
- Нельзя отписываться дважды от одной и той же нотификации.
- Нельзя проверить, подписан ли объект на нотификацию.

Key-Value Observing

Для избавления от страданий создано много обёрток над API KVO.

ReactiveCocoa

```
[RACObserve(model, keyname) subscribeNext:^(NSString *newValue) {
        ...
    }];
```

KVOBlocks

```
[model addObserver:self forKeyPath:@"keyname" options:NSKeyValueObservingOptionNew
    context:nil withBlock:^(NSDictionary *change, void *context) {
        ...
    }];
```

Key-Value Observing

Чтобы обеспечить поддержку KVC/KVO для свойства, нужно соблюдать правила руководства по KVC/KVO compliance.

По умолчанию свойства стандартных классов нельзя считать KVO-совместимыми, а документация явно указывает какие свойства обладают этим свойством.