KVC & KVO

Simon Blommegård, Wrapp

Key-Value Coding

Key-Value Coding

- Set & get the value of a variable by its name

Key-Value Coding

- Set & get the value of a variable by its name
- The name is a string, called *key*

Key-Value Coding

- Set & get the value of a variable by its name
- The name is a string, called *key*
- **setValue:forKey:** & **valueForKey:** defined in **NSObject**

Key-Value Coding

- Set & get the value of a variable by its name
- The name is a string, called *key*
- **setValue:forKey:** & **valueForKey:** defined in **NSObject**

1. Setter: **set<Key>** Getter: **get<Key>**, **<key>**, **is<Key>**

Key-Value Coding

- Set & get the value of a variable by its name
- The name is a string, called *key*
- **setValue:forKey:** & **valueForKey:** defined in **NSObject**

I. Setter: **set<Key>** Getter: **get<Key>**, **<key>**, **is<Key>**

II. Ivar: **_<key>**, **_is<Key>**, **<key>**, **is<Key>**

Key-Value Coding

Key-Value Coding

```
@interface WRPPerson : NSObject
@property (nonatomic, strong) NSString *firstName;
@end
```

Key-Value Coding

```
@interface WRPPerson : NSObject
@property (nonatomic, strong) NSString *firstName;
@end
```

```
WRPPerson *person = [WRPPerson new];
NSString *key = @"firstName";

[person setValue:@"Simon" forKey:key];
NSString *value = [person valueForKey:key];

NSLog(@"%@ : %@", key, value);
```

Key-Value Coding

```objc
@interface WRPPerson : NSObject
@property (nonatomic, strong) NSString *firstName;
@end
```

```objc
WRPPerson *person = [WRPPerson new];
NSString *key = @"firstName";

[person setValue:@"Simon" forKey:key];
NSString *value = [person valueForKey:key];

NSLog(@"%@ : %@", key, value);
```

```
Cocoaheads-KVO[22258:c07] firstName : Simon
```

Key Path

Key Path

- Path of keys

Key Path

- Path of keys
- **@"gift.recipient.firstName"**

Key Path

- Path of keys
- **@"gift.recipient.firstName"**
- **setValue:forKeyPath:** & **valueForKeyPath:**

Key Path

- Path of keys
- **@"gift.recipient.firstName"**
- **setValue:forKeyPath:** & **valueForKeyPath:**
- Collection operators: **@avg**, **@count**, **@max**, **@min**, **@sum**

Key Path

- Path of keys
- **@"gift.recipient.firstName"**
- **setValue:forKeyPath:** & **valueForKeyPath:**
- Collection operators: **@avg**, **@count**, **@max**, **@min**, **@sum**
- **@"friends.@avg.age"**

Key Path

Key Path

```objc
@interface WRPPerson : NSObject
@property (nonatomic, strong) NSString *firstName;
@property (nonatomic, assign) NSInteger age;
@property (nonatomic, strong) WRPPerson *bestFriend;
@end
```

Key Path

```objc
@interface WRPPerson : NSObject
@property (nonatomic, strong) NSString *firstName;
@property (nonatomic, assign) NSInteger age;
@property (nonatomic, strong) WRPPerson *bestFriend;
@end
```

```objc
WRPPerson *simon = [WRPPerson new];
[simon setAge:20];
[simon setFirstName:@"Simon"];

WRPPerson *greg = [WRPPerson new];
[greg setAge:24];
[greg setFirstName:@"Greg"];
[greg setBestFriend:simon];

NSLog(@"Gregs best friend : %@", [greg valueForKeyPath:@"bestFriend.firstName"]);

NSArray *persons = @[simon, greg];
NSLog(@"Avg. age : %@", [persons valueForKeyPath:@"@avg.age"]);
```

Key Path

```objc
@interface WRPPerson : NSObject
@property (nonatomic, strong) NSString *firstName;
@property (nonatomic, assign) NSInteger age;
@property (nonatomic, strong) WRPPerson *bestFriend;
@end
```

```objc
WRPPerson *simon = [WRPPerson new];
[simon setAge:20];
[simon setFirstName:@"Simon"];

WRPPerson *greg = [WRPPerson new];
[greg setAge:24];
[greg setFirstName:@"Greg"];
[greg setBestFriend:simon];

NSLog(@"Gregs best friend : %@", [greg valueForKeyPath:@"bestFriend.firstName"]);

NSArray *persons = @[simon, greg];
NSLog(@"Avg. age : %@", [persons valueForKeyPath:@"@avg.age"]);
```

```
Cocoaheads-KVO[25082:c07] Gregs best friend : Simon
Cocoaheads-KVO[23433:c07] Avg. age : 22
```

Key-Value Observation

Key-value observing provides a mechanism that allows objects to be notified of changes to specific properties of other objects. It is particularly useful for communication between model and controller layers in an application. (In OS X, the controller layer binding technology relies heavily on key-value observing.)

Key-Value Observation

- Registering as an Observer

Key-value observing provides a mechanism that allows objects to be notified of changes to specific properties of other objects. It is particularly useful for communication between model and controller layers in an application. (In OS X, the controller layer binding technology relies heavily on key-value observing.)

Key-Value Observation

- Registering as an Observer
- Receiving Notification of a Change

Key-value observing provides a mechanism that allows objects to be notified of changes to specific properties of other objects. It is particularly useful for communication between model and controller layers in an application. (In OS X, the controller layer binding technology relies heavily on key-value observing.)

Key-Value Observation

- Registering as an Observer
- Receiving Notification of a Change
- Removing an Object as an Observer

Key-value observing provides a mechanism that allows objects to be notified of changes to specific properties of other objects. It is particularly useful for communication between model and controller layers in an application. (In OS X, the controller layer binding technology relies heavily on key-value observing.)

Registering as an Observer

Registering as an Observer

```
[self.person addObserver:self
              forKeyPath:@"age"
                 options:(NSKeyValueObservingOptionNew | NSKeyValueObservingOptionInitial)
                 context:NULL];
```

Registering as an Observer

```
[self.person addObserver:self
              forKeyPath:@"age"
                 options:(NSKeyValueObservingOptionNew | NSKeyValueObservingOptionInitial)
                 context:NULL];
```

- NSKeyValueObservingOptionNew

Registering as an Observer

```
[self.person addObserver:self
              forKeyPath:@"age"
                 options:(NSKeyValueObservingOptionNew | NSKeyValueObservingOptionInitial)
                 context:NULL];
```

- NSKeyValueObservingOptionNew
- NSKeyValueObservingOptionOld

Registering as an Observer

```
[self.person addObserver:self
              forKeyPath:@"age"
                 options:(NSKeyValueObservingOptionNew | NSKeyValueObservingOptionInitial)
                 context:NULL];
```

- NSKeyValueObservingOptionNew
- NSKeyValueObservingOptionOld
- NSKeyValueObservingOptionInitial

Registering as an Observer

```
[self.person addObserver:self
              forKeyPath:@"age"
                 options:(NSKeyValueObservingOptionNew | NSKeyValueObservingOptionInitial)
                 context:NULL];
```

- NSKeyValueObservingOptionNew
- NSKeyValueObservingOptionOld
- NSKeyValueObservingOptionInitial
- NSKeyValueObservingOptionPrior

Receiving Notification of a Change

Collections supports indexes
Kind: setting,insert,removal & replacement

Receiving Notification of a Change

```
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object
                        change:(NSDictionary *)change
                       context:(void *)context {

  if ([keyPath isEqualToString:@"age"]) {
    [self.ageLabel setText:[NSString stringWithFormat:@"%@", change[NSKeyValueChangeNewKey]]];
  } else
    [super observeValueForKeyPath:keyPath ofObject:object change:change context:context];
}
```

Collections supports indexes
Kind: setting,insert,removal & replacement

Receiving Notification of a Change

```
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object
                        change:(NSDictionary *)change
                       context:(void *)context {

  if ([keyPath isEqualToString:@"age"]) {
    [self.ageLabel setText:[NSString stringWithFormat:@"%@", change[NSKeyValueChangeNewKey]]];
  } else
    [super observeValueForKeyPath:keyPath ofObject:object change:change context:context];
}
```

Collections supports indexes
Kind: setting,insert,removal & replacement

- NSKeyValueChangeKindKey

Receiving Notification of a Change

```
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object
                        change:(NSDictionary *)change
                       context:(void *)context {

  if ([keyPath isEqualToString:@"age"]) {
    [self.ageLabel setText:[NSString stringWithFormat:@"%@", change[NSKeyValueChangeNewKey]]];
  } else
    [super observeValueForKeyPath:keyPath ofObject:object change:change context:context];
}
```

Collections supports indexes
Kind: setting,insert,removal & replacement

- NSKeyValueChangeKindKey
- NSKeyValueChangeNewKey

Receiving Notification of a Change

```
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object
                        change:(NSDictionary *)change
                       context:(void *)context {

  if ([keyPath isEqualToString:@"age"]) {
    [self.ageLabel setText:[NSString stringWithFormat:@"%@", change[NSKeyValueChangeNewKey]]];
  } else
    [super observeValueForKeyPath:keyPath ofObject:object change:change context:context];
}
```

Collections supports indexes
Kind: setting,insert,removal & replacement

- NSKeyValueChangeKindKey
- NSKeyValueChangeNewKey
- NSKeyValueChangeOldKey

Receiving Notification of a Change

```
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object
                        change:(NSDictionary *)change
                       context:(void *)context {

  if ([keyPath isEqualToString:@"age"]) {
    [self.ageLabel setText:[NSString stringWithFormat:@"%@", change[NSKeyValueChangeNewKey]]];
  } else
    [super observeValueForKeyPath:keyPath ofObject:object change:change context:context];
}
```

Collections supports indexes
Kind: setting,insert,removal & replacement

- NSKeyValueChangeKindKey
- NSKeyValueChangeNewKey
- NSKeyValueChangeOldKey
- NSKeyValueChangeIndexesKey

Receiving Notification of a Change

```
- (void)observeValueForKeyPath:(NSString *)keyPath
                      ofObject:(id)object
                        change:(NSDictionary *)change
                       context:(void *)context {

  if ([keyPath isEqualToString:@"age"]) {
    [self.ageLabel setText:[NSString stringWithFormat:@"%@", change[NSKeyValueChangeNewKey]]];
  } else
    [super observeValueForKeyPath:keyPath ofObject:object change:change context:context];
}
```

Collections supports indexes
Kind: setting,insert,removal & replacement

- NSKeyValueChangeKindKey
- NSKeyValueChangeNewKey
- NSKeyValueChangeOldKey
- NSKeyValueChangeIndexesKey
- NSKeyValueChangeNotificationIsPriorKey

Removing an Object as an Observer

Removing an Object as an Observer

```
[self.person removeObserver:self forKeyPath:@"age" context:NULL];
```

Dependent Keys

Not always backed by an ivar

Dependent Keys

```
#pragma mark - Dependent Keys

+ (NSSet *)keyPathsForValuesAffectingFullName
```

- keyPathsForValuesAffectingAge
- keyPathsForValuesAffectingBestFriend
- keyPathsForValuesAffectingFirstName
- keyPathsForValuesAffectingFullName
- keyPathsForValuesAffectingLastName
- keyPathsForValuesAffectingValueForKey:(NSString *)key

Not always backed by an ivar

Dependent Keys

```
#pragma mark - Dependent Keys

+ (NSSet *)keyPathsForValuesAffectingFullName
```

```
keyPathsForValuesAffectingAge
keyPathsForValuesAffectingBestFriend
keyPathsForValuesAffectingFirstName
keyPathsForValuesAffectingFullName
keyPathsForValuesAffectingLastName
keyPathsForValuesAffectingValueForKey:(NSString *)key
```

Not always backed by an ivar

```
+ (NSSet *)keyPathsForValuesAffectingFullName {
  return [NSSet setWithObjects:@"firstName", @"lastName", nil];
}
```

Manual Change Notifica

When you fiddling with the ivar for example

Lets say you dont want to automatically do it for a key

Manual Change Notifica

When you fiddling with the ivar for example

Lets say you dont want to automatically do it for a key

```
#pragma mark - Properties

- (void)setAge:(NSInteger)age {
  if (_age != age) {
    [self willChangeValueForKey:@"age"];
    _age = age;
    [self didChangeValueForKey:@"age"];
  }
}

#pragma mark - KVO

+ (BOOL)automaticallyNotifiesObserversOfAge {
  return NO;
}
```

Other KVO stuff

Other KVO stuff

- Synchronous

Other KVO stuff

- Synchronous
- **- (void *)observationInfo**

Other KVO stuff

- Synchronous
- **- (void *)observationInfo**

```
<NSKeyValueObservance 0x7177600:
  Observer: 0x8958a60,
  Key path: age,
  Options: <New: YES, Old: NO, Prior: NO>
  Context: 0x0,
  Property: 0x895fe70
>
```

Other KVO stuff

- Synchronous
- **- (void *)observationInfo**

```
<NSKeyValueObservance 0x7177600:
  Observer: 0x8958a60,
  Key path: age,
  Options: <New: YES, Old: NO, Prior: NO>
  Context: 0x0,
  Property: 0x895fe70
>
```

- Automatic done by isa-swizzling

Broken things

Broken things

- Neither block or custom selector-support

Broken things

- Neither block or custom selector-support
- The inability to deal with objects whose observations are never unregistered

Broken things

- Neither block or custom selector-support
- The inability to deal with objects whose observations are never unregistered
- No NSNotificationCenter styled targeted observation removal

Broken things

- Neither block or custom selector-support
- The inability to deal with objects whose observations are never unregistered
- No NSNotificationCenter styled targeted observation removal
- No “remove all observers for this object”

Broken things

- Neither block or custom selector-support
- The inability to deal with objects whose observations are never unregistered
- No NSNotificationCenter styled targeted observation removal
- No “remove all observers for this object”
- Only register/deregister for one key path at a time

Super epic mega awesome shit

Super epic mega awesome shit

- MAKVONotificationCenter

  https://github.com/mikeash/MAKVONotificationCenter

Super epic mega awesome shit

- MAKVONotificationCenter

  https://github.com/mikeash/MAKVONotificationCenter

- ReactiveCocoa

  https://github.com/github/ReactiveCocoa

Simon Blommegård

simon@blommegard.se

@blommegard

github.com/blommegard