iOS development Learning Note

Rui Zhao

March 20, 2013

Contents

1 MVC

Controller has delegate of view. view dont own the data they display. controller is the data source. Model wanna tell controller something, controller can listen to model(observation pattern)

2 Start from an example

2.1 Vew heander

```
@interface CalculatorViewController : UIViewController
//delegate,more strightforword way, but hide the view code
@property (weak, nonatomic) IBOutlet UILabel *display;
@end
```

Objective C support week reference and reference counting. Drag the view to make a delegation is cool, but we can't see the code for view.

2.2 View implementation

```
//()make it private
@interface CalculatorViewController ()
@property (nonatomic) BOOL userIsInTheMiddleOfEnteringANumber;
@property (nonatomic,strong) CalculatorBrain *brain;
@end

@implementation CalculatorViewController
//_display is action location
@synthesize display = _display;
@synthesize brain = _brain;

- (CalculatorBrain *) brain {
    if (!_brain) {
    //syntax to new
        _brain = [[CalculatorBrain alloc] init];
    }
    return _brain;
}
```

make a member property and synthesize it can omit the getter and setter code, but we need to implement the getter if we wanna lazy initialize it.

```
//IBAcion is actually void,id is a pointer to unknow class of object
- (IBAction)digitPressed:(UIButton *)sender {
    NSString *digit = sender.currentTitle;
    //print the debug log
    NSLog(@"digit pressed = %@", digit);
    if (self.userIsInTheMiddleOfEnteringANumber) {
        self.display.text = [self.display.text stringByAppendingString:digit];
    } else {
        self.userIsInTheMiddleOfEnteringANumber = YES;
        self.display.text = digit;
    }

}
- (IBAction)operationPressed:(UIButton *)sender {
    if (self.userIsInTheMiddleOfEnteringANumber) [self enterPressed];
    double result = [self.brain performOperation:sender.currentTitle];
    NSString *resultString = [NSString stringWithFormat:@"%g", result];
    self.display.text = resultString;


}
 // don't need parameter here
- (IBAction)enterPressed {
    [self.brain pushOperand:[self.display.text doubleValue]];
    self.userIsInTheMiddleOfEnteringANumber = NO;
}

@end
```

2.3 Model implemtation

```
@interface CalculatorBrain()
@property (nonatomic, strong) NSMutableArray *operandStack;
@end


@implementation CalculatorBrain

//_opreandStack is the actually name of the location
@synthesize operandStack = _operandStack;

- (NSMutableArray *) operandStack {
    if (_operandStack == NULL) {
        _operandStack = [[NSMutableArray alloc] init];
    }
    return _operandStack;

}

- (void) pushOperand:(double) operand{
    //boxing like java
    [self.operandStack addObject:[NSNumber numberWithDouble:operand]];
}

- (double) popOperand {
    NSNumber *operandObject = [self.operandStack lastObject];
    if (operandObject) [self.operandStack removeLastObject];
    return [operandObject doubleValue];

}

- (double) performOperation:(NSString *) operation{
    double result = 0;
    if ([operation isEqualToString:@"+"]) {
        result = [self popOperand] + [self popOperand];
    } else if ([@"*" isEqualToString:operation]) {
        result = [self popOperand] * [self popOperand];
    }

    return result;
}


@end
```

Define a class's public @interface and private @implementation in a .h and .m file respectively and a private @interface to .m file

3 Objective C

3.1 Syntax, Language

3.1.1 Strong vs Weak

strong:keep it in the heap until i don't point to it anymore; weak:keep this as long as someone else points to it strongly, strong means ownership, weak means i just wanna get info of you.

3.1.2 Comma vs Colon

comma used when there is indeterminate parameters

```
NSLog(@"digit pressed = %@", digit);
```

3.1.3 Initializers

Classes must designate an initializer for subclassers,(the reason is trying to avoid loop initialize) for subclassing reasons, init methods should be typed to return id(not statically typed)

```
@implementation MyObject

//designated one. documentation
- (id) init
{
    self = [super init]; //call our super's designated initializer , convention
    if (self) {
        // initialize  our subclass  here
    }
    return  self ;
}

−(id) initWithString (NSString ∗) str
{
        //do we have to do the assignment here?
     self  = [self  init ];
    return  self ;
}
```

3.1.4 Dynamic Binding

All object are allocated in the heap, so always use a pointer, id itself is a pointer. we can cast a pointer to any type and i will crash if it doesn't have the method we expected.

```
 // call  NSString's class  method. not sure why Objective C don't just use the class  name
 if  ([obj isKindOfClass:[NSString class ]])
```

3.1.5 Introsepction

Method testing

```
if  ([] obj responseToSelector:@selector(shoot)]) {
     [obj shoot];
}
```

SEL is type for selector

```
SEL shootSelector = @selector(shoot);
SEL moveToSelector = @selector(moveTo:withPenColor);
[obj performSelector:shootSelector ];
[array makeObjectPerformSelector:shootAtSelector withObject:target];
//in UIButton.
[button addTarget:self action:@selector(digitPressed:)  ...];
```

4 Foundation Framework

4.1 NSObject

Base class for pretty much every object in the iOS SDK, implements introspection methods, etc.

4.2 NSString

Immutable! Unicode!

4.3 NSMutableString

mutable one still need to go to heap and allocating memories. and it's not optimized like immutable one.

4.4 NSNumber

Object wrapper around primitive types like int, float, double etc. so we can throw them into collection(NSArray or NSDictionary)

```
NSNumber *num = [NSNumber numberWithInt:36];
```

4.5 NSValue

Generic object wrapper for other non-object data types. (c Struct)

4.6 NSData

"Bag of bits"

4.7 NSArray

Ordered collection of objects. Immutable. make sense because we don't wanna screw things up after give the pointer to others. most language's array is mutable but we can use lock to make sure we don't change it when somebody else is changing it.

4.8 NSMutableArray

```
+(id) arrayWithCapacity:(int) initialSpace;
-(id) copy;//returns an NSAarry.
```

4.9 NSDictionary

Immutable hash table.

4.10 NSSet

set, unordered. i guess it's more efficient than c++'s set because that one use red-black tree.

```
-(int) count;
-(BOOL) containsObject:(id) anObject;
-(id) anyObjects;
-(void) makeObjectsPerformSelector:(SEL) aSelector;
```

4.11 NSMutableSet

Mutable version of NSSet.

```
- (void) addObject:(id) anObject;// does nothing if object that isEqual:anObject is already in
- (void) removeObject:(id) anObject.
```

4.12 NSOrderedset

Immutable ordered collection of distinct object.

4.13 NSMutableOrderedSet

Mutable version of NSOrderedSet

4.14 Enumeration

Looping through members of a collection in an efficient manner.

4.14.1 array

```
NSArray *myArray = [[NSArray alloc] init];
for (NSString *string in myArray) { //no way for compiler to know what myArray contains
    double value = [string doubleValue]; // crash here if string is not an NSString
}
```

4.14.2 dictionary

```
for (id key in myDictionary) {
    id value = [myDictionary objectForKey:key];
}
```

4.15 Property List

A collection of collections. i think it simply has to have an order so we can read and write it to somewhere it is any graph of objects containing only the following classes:

```
NSAarray, NSDictionary, NSNumber, NSString, NSDate, NSData.

[ plist writeToFile:(NSString *) path atomically:(BOOL)]; // plist is NSArray or NSDictionary
```

4.16 NSUserDefaults

Lightweight storage of Property List.

```
[[ NSUserDefaults standardUserDefaults] setArray:rvArray forKey:@"RecentlyViewed"]
```

4.17 Continue our example

We use a mutable array to save the operand and operators.

```
@interface CalculatorBrain()
@property (nonatomic, strong) NSMutableArray *programStack;
@end
```

we push everything to the array, runProgram takes program/array as a parameter and creates an mutable copy from it.

```
- (void) pushOperand:(double) operand{
    //boxing like java
    [ self .programStack addObject:[NSNumber numberWithDouble:operand]];
}
- (double) performOperation:(NSString *) operation{
    [ self .programStack addObject:operation];
    return [CalculatorBrain runProgram:self.program];
}
+(double)runProgram:(id)program{
    NSMutableArray *stack;
    if ([program isKindOfClass:[NSArray class]]) {
        stack = [program mutableCopy];
    }
    return [ self popOperandOffStack:stack];

}
```

calculate the result, each time we got the mutable array, we recursively call the popOperandOffStack to get the result. drawback: we need to do the recursive call each time. so we recalculated lots of stuff.

```
+(double) popOperandOffStack:(NSMutableArray *)stack {
    double result = 0;
    id topOfStack = [stack lastObject];
    if (topOfStack) [stack removeLastObject];
    if ([topOfStack isKindOfClass:[NSNumber class]]){
        result = [topOfStack doubleValue];
    } else if ([topOfStack isKindOfClass:[NSString class]]) {
        NSString *operation = topOfStack;
        if ([operation isEqualToString:@"+"]) {
            result = [self popOperandOffStack:stack] + [self popOperandOffStack:stack];
        } else if ([@"*" isEqualToString:operation]) {
            result = [self popOperandOffStack:stack] * [self popOperandOffStack:stack];
        }
    }

    return result;
}
```

5 Views

A view i.e. UIView subclass, is an rectangle on the screen. a view has only one superview. but can have different number of subviews. UIWindow is the UIView at the top of view hierarchy, only have one UIWindow in an iOS application.

5.1 CGFloat, CGPoint,, CGSize, CGRect

Just a floating point number, always use it for graphics.

```
CGRect aRect = CGRectMake (4.5, 3.5, 3,3);
aRect.size.height+=45;
aRect.origin.x+=30;
```

5.2 Coordinates

(0.0) is the upper left, Units are "points" not pixels. for portable reason. UIView has a property to tell.

```
@property CGFloat contentScaleFactor;//returns pixels per point on the screen this view is on
```

View has 3 properties about size and location

```
@property CGRect bounds;// adjust itself on screen
@property CGPoint center;// the center of your view in your superview's coord.
@property CGRect frame;// a rectangle in your superview's coordinate space which entirely contains your
    view's bounds.size;
```

5.3 Drawing is fun

Graphic is expensive, don't do premature optimization on other stuff as long as the code is clean;

```
-(void) drawRect:(CGRect) aRect; // newer call drawRect;
```

We can use Core Graphics framework to implement drawRect. The API is C. The context determines where your drawing goes.

```
CGContextRef context = UIGraphicsGetCurrentContext();
[[UIColor greenColor] setFill ];
[[UIColor redColor] setStroke];
CGContextDrawPath(context, kCGPathFillStroke);//kCGPathFillStroke is a constant
CGContextBeginPath(context);//Begin the path
CGContextMoveToPoint(context, 7,5);
CGContextAddLineToPoint(context, 1,2);
CGContextClosePath(context);
```

we can build a path and reuse it later. CGPath. Transparency is not Cheap. we can use UIGraphicsPushContext(ctxt) and popContext to keep graphics state liner.

5.4 Drawing Text

UILabel. UIFont can get a font. NSString are defined in UIKit via a mechanism called ”categories”.

5.5 Drawing Image

Create a UIImage object from a file in your Resources folder

```
UIImage *image = [UIImage imageNamed:@foo.jpg];
```

Or create one from a named file or from raw data

```
UIImage *image = [[UIImage alloc] initWithContentsOfFile:(NSString *)fullPath];
UIImage *image = [[UIImage alloc] initWithData:(NSData *)imageData];
```

6 Debugging

6.1 gdb

```
gdb print [self operand]
gdb print self
```

we can overwrite the description, so the print self will print that string.

7 Protocols

Protocols is actually interfaces in other languages, and "interface" in Objective C is "public.

7.1 Basic

```
@protocol Foo <Other, NSObject> // implementors must implement Other and NSObject too
- (void)doSomething; // implementors must implement this (methods are @required by default) @optional
- (int)getSomething; // implementors do not have to implement this
- (void)doSomethingOptionalWithArgument:(NSString *)argument; // also optional
@required
- (NSArray *)getManySomethings:(int)howMany; // back to being must implement
@property (nonatomic, strong) NSString *fooProp; // note that you must specify strength
@end                    // (unless its readonly, of course)
```

7.2 Why

Number 1 use of protocols in iOS: delegates and sources, This assumes that the object serving as delegate will outlive the object doing the delegating. Especially true in the case where the delegator is a View object

8 UIGestureRecognizer

There are two sides to using a gesture recognizer 1. Adding a gesture recognizer to a UIView to ask it to recognize that gesture. 2. Providing the implementation of a method to handle that gesture when it happens.

8.1 adding

Adding a gesture recognizer to a UIView from a Controller

```
 - (void)setPannableView:(UIView *)pannableView
     {
         _pannableView = pannableView;
         UIPanGestureRecognizer *pangr =
             [[UIPanGestureRecognizer alloc] initWithTarget:pannableView action:@selector(pan:)];
         [pannableView addGestureRecognizer:pangr];
}
```

```
- (void)pan:(UIPanGestureRecognizer *)recognizer
{
if ((recognizer.state == UIGestureRecognizerStateChanged) ||
    (recognizer.state == UIGestureRecognizerStateEnded)) {
    CGPoint translation = [recognizer translationInView:self];
    @property called origin self.origin = CGPointMake(self.origin.x+translation.x, self.origin.y
      +translation.y);
    [recognizer setTranslation:CGPointZero inView:self];
    }
}
```

8.2 Enumeration

8.3 Enumeration

8.4 Enumeration

8.5 Enumeration

8.6