Objective-C switch using objects?

Asked 16 years, 3 months ago Modified 11 years, 10 months ago Viewed 31k times

14

I'm doing some Objective-C programming that involves parsing an NSXmlDocument and populating an objects properties from the result.

First version looked like this:

```
if([elementName compare:@"companyName"] == 0)
  [character setCorporationName:currentElementText];
else if([elementName compare:@"corporationID"] == 0)
  [character setCorporationID:currentElementText];
else if([elementName compare:@"name"] == 0)
  ...
```

But I don't like the `if-else-if-else` pattern this produces. Looking at the `switch` statement I see that i can only handle `ints`, `chars` etc and not objects... so is there a better implementation pattern I'm not aware of?

BTW I did actually come up with a better solution for setting the object's properties, but I want to know specifically about the `if`-`else` vs `switch` pattern in Objective-C

objective-c design-patterns switch-statement

Share
Improve this question
Follow

edited Nov 10, 2008 at 14:01
The Archetypal Paul
41.7k ● 20 ● 106 ● 135

asked Sep 19, 2008 at 18:29
craigb
16.9k ● 7 ● 53 ● 63

This doesn't help at all, but in VB.Net, you can use switch (select in vb) statements on any type of value, and the compiler will do the conversion to if-elseif-else pattern when it compiles, if it is not possible to do via a direct jump. Personally, I think all languages should have this feature. – Kibbee Nov 10, 2008 at 14:04

Related answer: stackoverflow.com/questions/8161737/… – Graham Perks Jan 3, 2013 at 16:22

14 Answers

Sorted by: Highest score (default)

You should take advantage of Key-Value Coding:

13

```
[character setValue:currentElementText forKey:elementName];
```

If the data is untrusted, you might want to check that the key is valid:

```
if (![validKeysCollection containsObject:elementName])
    // Exception or error
```

Share Improve this answer Follow

answered Sep 21, 2008 at 4:37
jmah
2,216 ● 14 ● 16

As stated in question, I already found a better way of setting these properties and was just looking for advice on the if-else pattern that i didn't like. – craigb Sep 21, 2008 at 19:11

But that is the point, to avoid doing the multiple dispatch yourself, and letting aspects of the language (or framework) handle it. The if-else-else-else or switch pattern on object value should be discouraged, when you can do things like dictionary look-ups. – jmah Sep 22, 2008 at 3:23

12

✓

I hope you'll all forgive me for going out on a limb here, but I would like to address the more general question of parsing XML documents in Cocoa without the need of if-else statements. The question as originally stated assigns the current element text to an instance variable of the character object. As jmah pointed out, this can be solved using key-value coding. However, in a more complex XML document this might not be possible. Consider for example the following.

```
<xmlroot>
    <corporationID>
        <stockSymbol>EXAM</stockSymbol>
        <uuid>31337</uuid>
    </corporationID>
    <companyName>Example Inc.</companyName>
</xmlroot>
```

There are multiple approaches to dealing with this. Off of the top of my head, I can think of two using NSXMLDocument. The first uses NSXMLElement. It is fairly straightforward and does not involve the if-else issue at all. You simply get the root element and go through its named elements one by one.

```
NSXMLElement* root = [xmlDocument rootElement];

// Assuming that we only have one of each element.
[character setCorperationName:[[[root elementsForName:@"companyName"]
objectAtIndex:0] stringValue]];

NSXMLElement* corperationId = [root elementsForName:@"corporationID"];
```

```
[character setCorperationStockSymbol:[[[corperationId
elementsForName:@"stockSymbol"] objectAtIndex:0] stringValue]];
[character setCorperationUUID:[[[corperationId elementsForName:@"uuid"]
objectAtIndex:0] stringValue]];
```

The next one uses the more general NSXMLNode, walks through the tree, and directly uses the if-else structure.

```
// The first line is the same as the last example, because NSXMLElement
inherits from NSXMLNode
NSXMLNode* aNode = [xmlDocument rootElement];
while(aNode = [aNode nextNode]){
    if([[aNode name] isEqualToString:@"companyName"]){
        [character setCorperationName:[aNode stringValue]];
    }else if([[aNode name] isEqualToString:@"corporationID"]){
        NSXMLNode* correctParent = aNode;
        while((aNode = [aNode nextNode]) == nil && [aNode parent !=
correctParent){
            if([[aNode name] isEqualToString:@"stockSymbol"]){
                [character setCorperationStockSymbol:[aNode stringValue]];
            }else if([[aNode name] isEqualToString:@"uuid"]){
                [character setCorperationUUID:[aNode stringValue]];
            }
        }
    }
}
```

This is a good candidate for eliminating the if-else structure, but like the original problem, we can't simply use switch-case here. However, we can still eliminate if-else by using performSelector. The first step is to define the a method for each element.

```
- (NSNode*)parse_companyName:(NSNode*)aNode
{
    [character setCorperationName:[aNode stringValue]];
    return aNode;
}

- (NSNode*)parse_corporationID:(NSNode*)aNode
{
    NSXMLNode* correctParent = aNode;
    while((aNode = [aNode nextNode]) == nil && [aNode parent != correctParent){
        [self invokeMethodForNode:aNode prefix:@"parse_corporationID_"];
    }
    return [aNode previousNode];
}

- (NSNode*)parse_corporationID_stockSymbol:(NSNode*)aNode
{
    [character setCorperationStockSymbol:[aNode stringValue]];
    return aNode;
}

- (NSNode*)parse_corporationID_uuid:(NSNode*)aNode
{
    [character setCorperationUUID:[aNode stringValue]];
    return aNode;
}
```

The magic happens in the invokeMethodForNode:prefix: method. We generate the selector based on the name of the element, and perform that selector with aNode as the only parameter. Presto bango, we've eliminated the need for an if-else statement. Here's the code for that method.

```
- (NSNode*)invokeMethodForNode:(NSNode*)aNode prefix:(NSString*)aPrefix
{
    NSNode* ret = nil;
    NSString* methodName = [NSString stringWithFormat:@"%@%@:", prefix, [aNode
name]];
    SEL selector = NSSelectorFromString(methodName);
    if([self respondsToSelector:selector])
        ret = [self performSelector:selector withObject:aNode];
    return ret;
}
```

Now, instead of our larger if-else statement (the one that differentiated between companyName and corporationID), we can simply write one line of code

```
NSXMLNode* aNode = [xmlDocument rootElement];
while(aNode = [aNode nextNode]){
    aNode = [self invokeMethodForNode:aNode prefix:@"parse_"];
}
```

Now I apologize if I got any of this wrong, it's been a while since I've written anything with NSXMLDocument, it's late at night and I didn't actually test this code. So if you see anything wrong, please leave a comment or edit this answer.

However, I believe I have just shown how properly-named selectors can be used in Cocoa to completely eliminate if-else statements in cases like this. There are a few gotchas and corner cases. The performSelector: family of methods only takes 0, 1, or 2 argument methods whose arguments and return types are objects, so if the types of the arguments and return type are not objects, or if there are more than two arguments, then you would have to use an NSInvocation to invoke it. You have to make sure that the method names you generate aren't going to call other methods, especially if the target of the call is another object, and this particular method naming scheme won't work on elements with non-alphanumeric characters. You could get around that by escaping the XML element names in your method names somehow, or by building an NSDictionary using the method names as the keys and the selectors as the values. This can get pretty memory intensive and end up taking a longer time. performSelector dispatch like I described is pretty fast. For very large if-else statements, this method may even be faster than an if-else statement.

Share

Improve this answer

Follow

edited Jul 30, 2011 at 1:06

community wiki
4 revs, 3 users 87%
Michael Buckley

I do like the compactness of this solution. Just a couple of remarks: it's NSXMLNode not NSNode. NSXMLNode *ret -- I can't think of any use for that since aNode is already set by nextNode. Using performSelector like this will give the warning: "performSelector may cause a leak because its selector is unknown". Shortest way around this is to use objc_msgSend but then you have to turn off 'Enable strict checking of objc_msgSend Calls'. Do wrap this in [self respondsToSelector:selector] to alleviate some of objc_msgSend shortcomings.
– Elise van Looij Jun 6, 2016 at 17:18

Hey, thanks for the comment. This answer is a community wiki, so please feel free to edit any mistakes you may find. I would, however, like to point out that this answer was more an exercise in exploring an implementation with an arbitrary limitation, and not intended to be advice on how to actually write production code. – Michael Buckley Jun 27, 2016 at 15:59

7

If you want to use as little code as possible, and your element names and setters are all named so that if elementName is @"foo" then setter is setFoo:, you could do something like:

```
SEL selector = NSSelectorFromString([NSString stringWithFormat:@"set%@:",
[elementName capitalizedString]]);

[character performSelector:selector withObject:currentElementText];
```

or possibly even:

```
[character setValue:currentElementText forKey:elementName]; // KVC-style
```

Though these will of course be a bit slower than using a bunch of if statements.

[Edit: The second option was already mentioned by someone; oops!]

Share

Improve this answer

Follow

edited Sep 27, 2008 at 0:56

answered Sep 26, 2008 at 21:35

Wevah
28.3k ● 7 ● 85 ● 74

I added a correction to this below -- stackoverflow.com/questions/104339/… – Dennis Munsie Aug 1, 2009 at 3:48

Dare I suggest using a macro?

7

```
#define TEST( _name, _method ) \
  if ([elementName isEqualToString:@ _name] ) \
    [character _method:currentElementText]; else
#define ENDTEST { /* empty */ }

TEST( "companyName",      setCorporationName )
TEST( "setCorporationID", setCorporationID   )
TEST( "name",             setName            )
:
:
ENDTEST
```

Share
Improve this answer
Follow

edited Jan 10, 2009 at 22:06

answered Sep 26, 2008 at 19:49
epatel
46k ●17 ●111 ●144

Nice, hadn't thought of that one. Not used to using macros much. – craigb Sep 27, 2008 at 0:46

4

One way I've done this with NSStrings is by using an NSDictionary and enums. It may not be the most elegant, but I think it makes the code a little more readable. The following pseudocode is extracted from one of my projects:

```
typedef enum { UNKNOWNRESIDUE, DEOXYADENINE, DEOXYCYTOSINE, DEOXYGUANINE,
DEOXYTHYMINE } SLSResidueType;

static NSDictionary *pdbResidueLookupTable;
...

if (pdbResidueLookupTable == nil)
{
    pdbResidueLookupTable = [[NSDictionary alloc] initWithObjectsAndKeys:
                           [NSNumber numberWithInteger:DEOXYADENINE], @"DA",
                           [NSNumber numberWithInteger:DEOXYCYTOSINE], @"DC",
                           [NSNumber numberWithInteger:DEOXYGUANINE], @"DG",
                           [NSNumber numberWithInteger:DEOXYTHYMINE], @"DT",
                           nil];
}

SLSResidueType residueIdentifier = [[pdbResidueLookupTable
objectForKey:residueType] intValue];
switch (residueIdentifier)
{
    case DEOXYADENINE: do something; break;
    case DEOXYCYTOSINE: do something; break;
    case DEOXYGUANINE: do something; break;
    case DEOXYTHYMINE: do something; break;
}
```

Share Improve this answer Follow

answered Sep 23, 2008 at 0:14

Brad Larson
170k ● 45 ● 400 ● 572

3

The `if-else` implementation you have is the right way to do this, since `switch` won't work with objects. Apart from maybe being a bit harder to read (which is subjective), there is no real downside in using `if-else` statements this way.

Share
Improve this answer
Follow

edited Sep 19, 2008 at 19:19

answered Sep 19, 2008 at 19:07
Palmin
2,811 ● 2 ● 20 ● 14

3

Although there's not necessarily a better way to do something like that for one time use, why use "compare" when you can use "isEqualToString"? That would seem to be more performant since the comparison would halt at the first non-matching character, rather than going through the whole thing to calculate a valid comparison result (though come to think of it the comparison might be clear at the same point) - also though it would look a little cleaner because that call returns a BOOL.

```
if([elementName isEqualToString:@"companyName"] )
  [character setCorporationName:currentElementText];
else if([elementName isEqualToString:@"corporationID"] )
  [character setCorporationID:currentElementText];
else if([elementName isEqualToString:@"name"] )
```

Share Improve this answer Follow

answered Sep 21, 2008 at 4:30
Kendall Helmstetter Gelner
75k ● 26 ● 131 ● 151

As indicted in the original question i already found a better way to do the setting of properties. I was just looking for advice on any better pattern for dealing with the if-else pattern. – craigb Sep 21, 2008 at 19:10

Nice downvote. It's still useful information for people who would prefer cleaner ways to compare string. You are abusing downvotes, the only consolation is that it's costing you reputation to discard perfectly good information. I'll not make the mistake of helping you again. – Kendall Helmstetter Gelner Sep 21, 2008 at 20:44

3

There is actually a fairly simple way to deal with cascading if-else statements in a language like Objective-C. Yes, you can use subclassing and overriding, creating a group of subclasses that implement the same method differently, invoking the correct implementation at runtime using a common message. This works well if you wish to choose one of a few implementations, but it can result in a needless proliferation of subclasses if you have many small, slightly different implementations like you tend to have in long if-else or switch statements.

Instead, factor out the body of each if/else-if clause into its own method, all in the same class. Name the messages that invoke them in a similar fashion. Now create an NSArray containing the selectors of those messages (obtained using @selector()). Coerce the string you were testing in the conditionals into a selector using NSSelectorFromString() (you may need to concatenate additional words or colons to it first depending on how you named those messages, and whether or not they take arguments). Now have self perform the selector using performSelector:.

This approach has the downside that it can clutter-up the class with many new messages, but it's probably better to clutter-up a single class than the entire class hierarchy with new subclasses.

Share Improve this answer Follow

answered Sep 27, 2008 at 8:00
jake

3

Posting this as a response to Wevah's answer above -- I would've edited, but I don't have high enough reputation yet:

unfortunately the first method breaks for fields with more than one word in them -- like xPosition. capitalizedString will convert that to Xposition, which when combined with the format give you setXposition: . Definitely not what was wanted here. Here is what I'm using in my code:

```
NSString *capName = [elementName
stringByReplacingCharactersInRange:NSMakeRange(0, 1) withString:[[elementName
substringToIndex:1] uppercaseString]];
SEL selector = NSSelectorFromString([NSString stringWithFormat:@"set%@:",
capName]);
```

Not as pretty as the first method, but it works.

Share Improve this answer Follow

answered Aug 1, 2009 at 3:47
Dennis Munsie
1,211 ● 13 ● 24

3

I have come up with a solution that uses blocks to create a switch-like structure for objects. There it goes:

```
BOOL switch_object(id aObject, ...)
{
    va_list args;
    va_start(args, aObject);

    id value = nil;
    BOOL matchFound = NO;

    while ( (value = va_arg(args,id)) )
    {
        void (^block)(void) = va_arg(args,id);
        if ( [aObject isEqual:value] )
        {
            block();
            matchFound = YES;
            break;
        }
    }

    va_end(args);
    return matchFound;
}
```

As you can see, this is an oldschool C function with variable argument list. I pass the object to be tested in the first argument, followed by the case_value-case_block pairs. (Recall that Objective-C blocks are just objects.) The `while` loop keeps extracting these pairs until the object value is matched or there are no cases left (see notes below).

Usage:

```
NSString* str = @"stuff";
switch_object(str,
              @"blah", ^{
                  NSLog(@"blah");
              },
              @"foobar", ^{
                  NSLog(@"foobar");
              },
              @"stuff", ^{
                  NSLog(@"stuff");
              },
              @"poing", ^{
                  NSLog(@"poing");
              },
              nil);   // <-- sentinel

// will print "stuff"
```

Notes:

- this is a first approximation without any error checking

- the fact that the case handlers are blocks, requires additional care when it comes to visibility, scope and memory management of variables referenced from within
- if you forget the sentinel, you are doomed :P
- you can use the boolean return value to trigger a "default" case when none of the cases have been matched

Share Improve this answer Follow

edited Apr 13, 2012 at 8:47

answered Apr 12, 2012 at 13:16
Lvsti
1,525 ● 15 ● 15

I've put the case block extraction prior to the `isEqual:` check, otherwise the object is unnecessarily checked for equality with the block object. – Lvsti Apr 13, 2012 at 8:50

@Lvsti, +1 I like this block approach. I am curious, what you think about my block based filter chain approach I just posted as answer. – vikingosegundo Feb 9, 2013 at 15:21

2

The most common refactoring suggested for eliminating if-else or switch statements is introducing polymorphism (see http://www.refactoring.com/catalog/replaceConditionalWithPolymorphism.html). Eliminating such conditionals is most important when they are duplicated. In the case of XML parsing like your sample you are essentially moving the data to a more natural structure so that you won't have to duplicate the conditional elsewhere. In this case the if-else or switch statement is probably good enough.

Share Improve this answer Follow

answered Sep 19, 2008 at 19:03
Bradley Harris
932 ● 1 ● 6 ● 12

1

In this case, I'm not sure if you can easily refactor the class to introduce polymorphism as Bradley suggests, since it's a Cocoa-native class. Instead, the Objective-C way to do it is to use a class category to add an `elementNameCode` method to NSSting:

```
    typedef enum {
        companyName = 0,
        companyID,
        ...,
        Unknown
     } ElementCode;

     @interface NSString (ElementNameCodeAdditions)
     - (ElementCode)elementNameCode;
     @end

     @implementation NSString (ElementNameCodeAdditions)
     - (ElementCode)elementNameCode {
         if([self compare:@"companyName"]==0) {
             return companyName;
         } else if([self compare:@"companyID"]==0) {
             return companyID;
         } ... {

         }

         return Unknown;
     }
     @end
```

In your code, you could now use a switch on `[elementName elementNameCode]` (and gain the associated compiler warnings if you forget to test for one of the enum members etc.).

As Bradley points out, this may not be worth it if the logic is only used in one place.

Share

Improve this answer

Follow

edited Sep 22, 2008 at 20:11

answered Sep 20, 2008 at 23:00

Barry Wark

108k ● 24 ● 182 ● 206

1

What we've done in our projects where we need to so this sort of thing over and over, is to set up a static CFDictionary mapping the strings/objects to check against to a simple integer value. It leads to code that looks like this:

```
static CFDictionaryRef  map = NULL;
int count = 3;
const void *keys[count] = { @"key1", @"key2", @"key3" };
const void *values[count] = { (uintptr_t)1, (uintptr_t)2, (uintptr_t)3 };

if (map == NULL)
    map =
CFDictionaryCreate(NULL,keys,values,count,&kCFTypeDictionaryKeyCallBacks,NULL);


switch((uintptr_t)CFDictionaryGetValue(map,[node name]))
{
    case 1:
        // do something
        break;
    case 2:
        // do something else
        break;
    case 3:
        // this other thing too
        break;
}
```

If you're targeting Leopard only, you could use an NSMapTable instead of a CFDictionary.

Share Improve this answer Follow

answered Nov 4, 2008 at 17:48
Mike Shields
196 ● 2

0

Similar to Lvsti I am using blocks to perform a switching pattern on objects.

I wrote a very simple filter block based chain, that takes n filter blocks and performs each filter on the object.
Each filter can alter the object, but must return it. No matter what.

NSObject+Functional.h

```
#import <Foundation/Foundation.h>
typedef id(^FilterBlock)(id element, NSUInteger idx, BOOL *stop);

@interface NSObject (Functional)
-(id)processByPerformingFilterBlocks:(NSArray *)filterBlocks;
@end
```

NSObject+Functional.m

```
@implementation NSObject (Functional)
-(id)processByPerformingFilterBlocks:(NSArray *)filterBlocks
{
    __block id blockSelf = self;
    [filterBlocks enumerateObjectsUsingBlock:^( id (^block)(id,NSUInteger idx,
BOOL*) , NSUInteger idx, BOOL *stop) {
        blockSelf = block(blockSelf, idx, stop);
    }];

    return blockSelf;
}
@end
```

Now we can set up `n` FilterBlocks to test for the different cases.

```
FilterBlock caseYES = ^id(id element, NSUInteger idx, BOOL *breakAfter){
    if ([element isEqualToString:@"YES"]) {
        NSLog(@"You did it");
        *breakAfter = YES;
    }
    return element;
};

FilterBlock caseNO  = ^id(id element, NSUInteger idx, BOOL *breakAfter){
    if ([element isEqualToString:@"NO"] ) {
        NSLog(@"Nope");
        *breakAfter = YES;
    }
    return element;
};
```

Now we stick those block we want to test as a filter chain in an array:

```
NSArray *filters = @[caseYES, caseNO];
```

and can perform it on an object

```
id obj1 = @"YES";
id obj2 = @"NO";
[obj1 processByPerformingFilterBlocks:filters];
[obj2 processByPerformingFilterBlocks:filters];
```

This approach can be used for switching but also for any (conditional) filter chain application, as the blocks can edit the element and pass it on.

Share

Improve this answer

Follow

edited Feb 9, 2013 at 15:22

answered Feb 9, 2013 at 15:16

vikingosegundo

52.2k ● 14 ● 139 ● 183