

Objective-C Runtime in Practice

CocoaheadsBE - Kontich, 2013-12-03

Introduction

Tom Adriaenssen



I love...

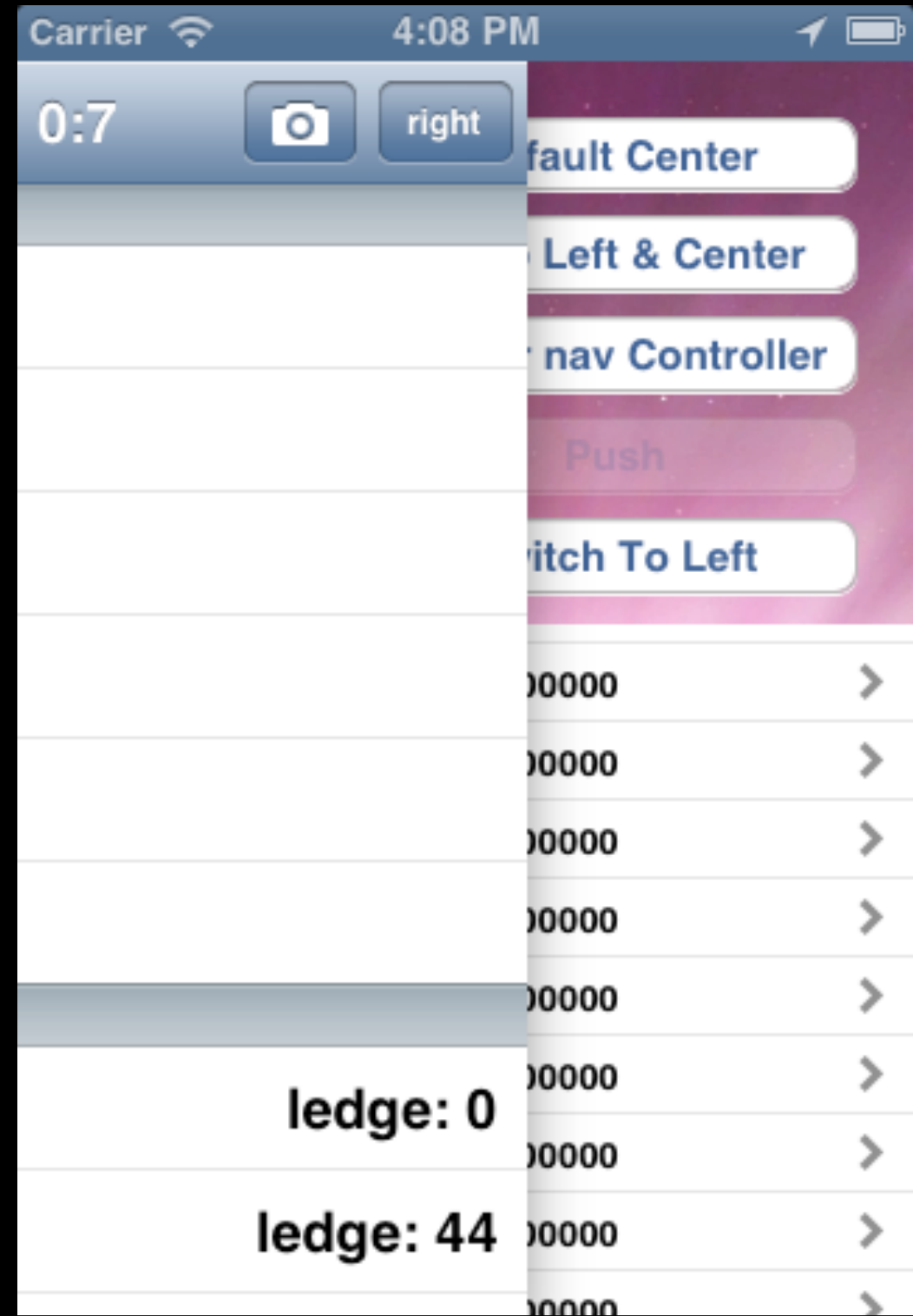
- ▶ ... my wife
- ▶ ... my 4 kids
- ▶ ... to code
- ▶ ... to play a game of squash
- ▶ ... good beer

I open sourced...

... some code:

- ▶ UINavigationController: “An implementation of the sliding functionality found in the Path 2.0 or Facebook iOS apps.”
- ▶ NSDateExtensions
- ▶ UPopoverStatusItem

See: <http://github.com/inferis>



I made...

... some apps:



Butane

<http://getbutane.com>

Hi, @10to1!



Drash

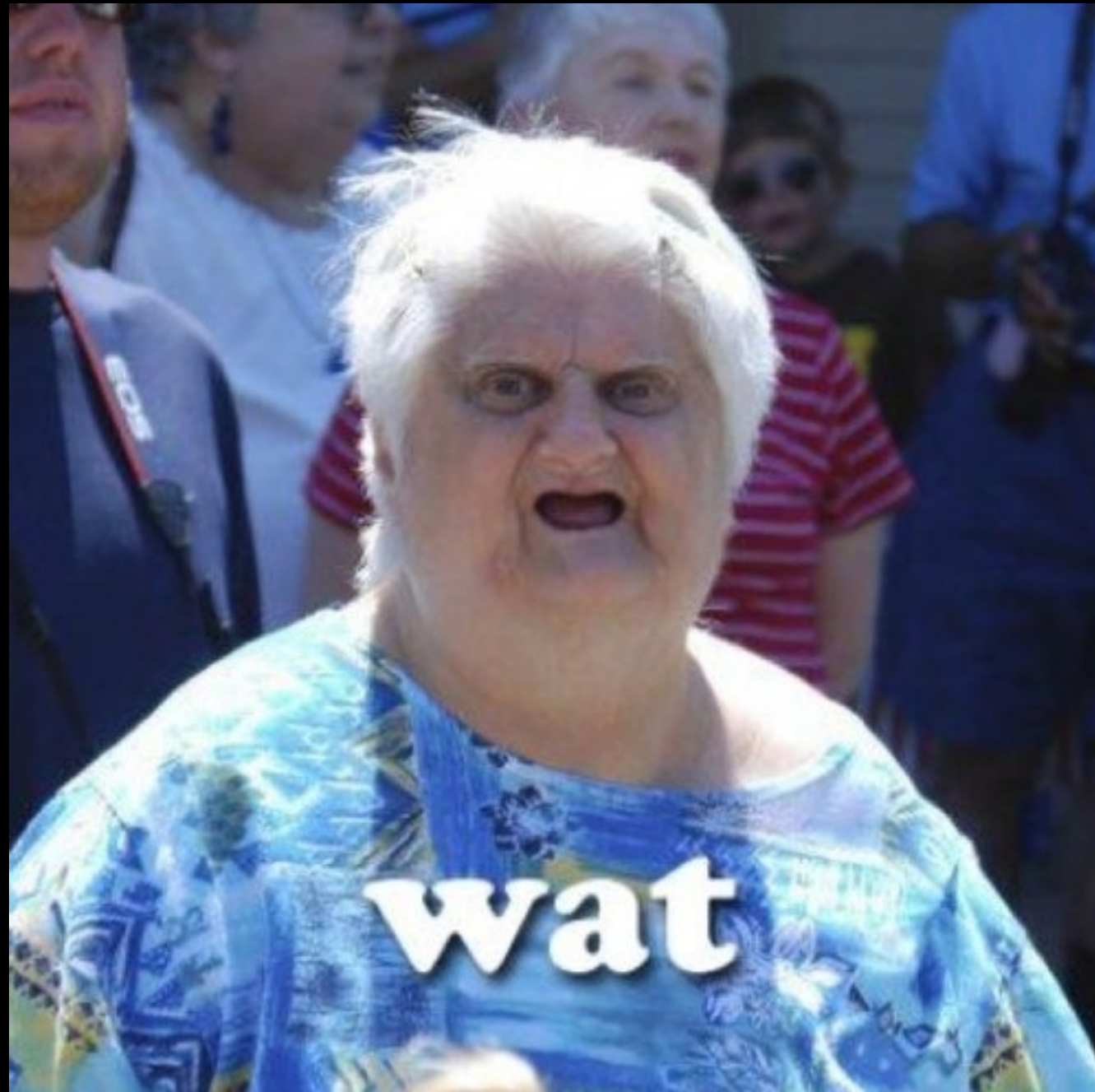
<http://dra.sh>

Agenda

Agenda

- ▶ RUNTIME WHUT?
- ▶ What is an object?
- ▶ In practice

RUNTIME



Runtime WHUT?

- ▶ obj-c runtime is "always present"
 - ▶ You can't use Objective-c without the runtime.
- ▶ Works behind the scenes:
 - ▶ most developers aren't even aware it is there and that they'r using it
 - ▶ puts the objective in Objective-C
- ▶ implemented as dynamic shared library
 - ▶ loaded by default by the OS

Runtime WHUT?

- ▶ Supports the most important features of the language
 - ▶ object oriented
 - ▶ messaging
 - ▶ protocols
 - ▶ dynamic typing
 - ▶ forwarding

Foundation

- ▶ is a support framework
 - ▶ included by default
- ▶ The Foundation framework defines a base layer of Objective-C classes.
 - ▶ a set of useful primitive object classes (NSObject, NSProxy, ...)
 - ▶ introduces several paradigms that define functionality not covered by the Objective-C language.
 - ▶ reflection
 - ▶ memory management
 - ▶ archiving

C + Runtime = Obj-C

- ▶ The runtime is what makes objective-c.
- ▶ The runtime is the implementation of the syntactic "objective" sugar on top of c
- ▶ You can write any cocoa program using pure C, but it's hard and verbose.

Demo

A pure C Objective-C app

In practice

- ▶ runtime.h overview
- ▶ Foundation.h, the “simple” stuff
- ▶ runtime.h, the “spicy” stuff

runtime.h

- ▶ `#import <objc/runtime.h>`
 - ▶ a number of C functions to interact with the runtime
- ▶ Several “categories” of interactions
 - ▶ `objc_...` interact with toplevel runtime (eg register a class)
 - ▶ `class_...` interact with classes (eg make subclass)
 - ▶ `object_...` interact with objects (eg get classname)
 - ▶ `method_...` interact with methods (eg get the number of arguments)
 - ▶ `ivar_...` interact with ivars (eg get the type of an ivar)
 - ▶ `property_...` interact with properties (eg get the name of a property)
 - ▶ `protocol_...` interact with protocols (eg get properties of a protocol)
 - ▶ `sel_...` interact with selectors (eg register selector names)
 - ▶ `imp_...` interact with method implementations (provide implementations using blocks)

C? Yuk.



Foundation.h to the rescue

- ▶ The Foundation library provides an obj-c interface to some of the runtime calls.

```
#include <Foundation/Foundation.h>
```

- ▶ Check your .pch file: it should be there
 - ▶ iOS: directly
 - ▶ OSX: via `#include <Cocoa.h>`.

Foundation.h to the rescue

- ▶ NSObject:

- ▶ `-(BOOL)isKindOfClass:(Class)class;`

- ▶ `-(Class)class;`

- ▶ Functions:

- ▶ `NSString* NSStringFromClass(Class aClass);`

- ▶ `Class NSSelectorFromString(NSString* aSelectorName);`

Dealing with classes

- ▶ `+ (Class)class;`

- ▶ Returns self (the class object). Since this is a class object, it returns the class itself.

- ▶ `+ (Class)superclass;`

- ▶ Returns the class object for the receiver's superclass. Gets the parent class of a given class.

- ▶ `+ (BOOL)isSubclassOfClass:(Class)aClass;`

- ▶ Returns a Boolean value that indicates whether the receiving class is a subclass of, or identical to, a given class.

Demo

Classes

Dealing with classes

- ▶ - `(BOOL)isKindOfClass:(Class)aClass;`
 - ▶ Returns a Boolean value that indicates whether the receiver is an instance of given class or an instance of any class that inherits from that class.
- ▶ - `(BOOL)isMemberOfClass:(Class)aClass;`
 - ▶ Returns a Boolean value that indicates whether the receiver is an instance of a given class.
- ▶ These are not the same!
 - ▶ **isKindOfClass** also works on subclass instances
 - ▶ **isMemberOfClass** only works on exact class instances

Demo

More classes

Protocols

- ▶ - `(BOOL)conformsToProtocol:(Protocol *)aProtocol;`
 - ▶ Returns a Boolean value that indicates whether the receiver conforms to a given protocol.
 - ▶ A class is said to “conform to” a protocol if it adopts the protocol or inherits from another class that adopts it. Protocols are adopted by listing them within angle brackets after the interface declaration.
 - ▶ This does not mean that the class listens to the protocols messages explicitly!

Messages

- ▶ - `(BOOL)respondToSelector:(SEL)selector`
 - ▶ Returns a Boolean value that indicates whether the receiving class responds to a given selector.
 - ▶ If this returns YES, you can safely send the message to the object.
- ▶ + `(BOOL)instancesRespondToSelector:(SEL)aSelector;`
 - ▶ Returns a Boolean value that indicates whether instances of the receiver are capable of responding to a given selector.
 - ▶ When you have a Class handy and not an instance of that class. Saves you creating an instance.
 - ▶ Is smart enough to discover if the class actually implements the message!
- ▶ - `(id)performSelector:(SEL)selector`
 - ▶ Sends a specified message to the receiver and returns the result of the message.

Demo

protocols & messages

Dynamic messaging

- ▶ So what actually happens when you call `[foo bar]`?
- ▶ when foo implements bar, that bar get executed. **Instant happy.**
- ▶ but what when there's no bar implementation?
 1. try Lazy Method Resolution
 2. try Fast forwarding
 3. try Normal forwarding
 4. **kaboom**

Dynamic messaging

1. **Lazy method resolution**

the runtime sends **+resolveInstanceMethod:**
(or **+resolveClassMethod:** for class methods)
to the class in question.

- ▶ If that method returns **YES**, the message send is restarted under the assumption that the appropriate method has now been added.

2. **Fast forwarding**

3. **Normal forwarding**

4. ***kaboom***

Dynamic messaging

1. Lazy method resolution

2. Fast forwarding

The instance receiving the message is sent - `forwardingTargetForSelector:`, but only if it implements it.

- ▶ If it implements this method and it returns something other than nil or self, the whole message sending process is restarted with that return value as the new target.
 - ▶ forwards the message to another object
 - ▶ no method implementation is added
 - ▶ target object can use whatever method implementation as it sees fit.

3. Normal forwarding

4. *kaboom*

Dynamic messaging

1. Lazy method resolution

2. Fast forwarding

3. Normal forwarding

Two step process:

1. First the runtime will send `-instanceMethodSignatureForSelector:` to see what kind of argument and return types are present.
2. If a valid method signature is returned, the runtime creates an `NSInvocation` describing the message being sent
3. finally `-forwardInvocation:` is sent to the instance. The instance should then use the `NSInvocation` on a target object to execute the method.

4. *kaboom*

Dynamic messaging

1. **Lazy method resolution**

2. **Fast forwarding**

3. **Normal forwarding**

4. ***kaboom***

The runtime calls `-doesNotRecognizeSelector:` on the instance.

- ▶ Default behavior is to throw an `NSInvalidArgumentException`, but you could override this if you'd want to
- ▶ but! be careful -> errors will go undetected!

Lazy method resolution

- ▶ Resolves/creates a method at runtime. Allows a class to create a method when it doesn't exist.
- ▶ Override one (or both) of these:
- ▶ **+ (BOOL)resolveClassMethod:(SEL)sel;**
 - ▶ Dynamically provides an implementation for a given selector for a class method.
- ▶ **+ (BOOL)resolveInstanceMethod:(SEL)sel;**
 - ▶ Dynamically provides an implementation for a given selector for an instance method.

Lazy method resolution

- ▶ So how does this work?
 - ▶ implement `+resolveInstanceMethod:`
 - ▶ check the selector
 - ▶ provide an implementation
 - ▶ `class_addMethod()`
 - ▶ need a method `IMP`:
 - ▶ copy an existing method
 - ▶ use a function
 - ▶ make new method using a block
- ▶ Same applies to `+resolveClassMethod:`
- ▶ resolve happens the first time a method is not found (and only then if you return `YES` from the resolver method)
- ▶ if you don't add an implementation but return `YES` anyway the you'll get an `NSInvalidArgumentException`.

Demo

lazy method resolution

Fast forwarding

- ▶ You can provide an interface but have the actual implementation be in another object.
- ▶ forward messages from one object to another
 - ▶ for the user, it is as if the first object handles the call
 - ▶ the actual handling object is "hidden" from the user
- ▶ So how does this work?
 - ▶ implement `-forwardingTargetForSelector:`
 - ▶ check the selector
 - ▶ provide an object that can handle the selector

Demo

fast forwarding

Normal forwarding

- ▶ have the object provide a method signature for the selector, so the runtime knows what arguments and return type there should be.
 - ▶ then forward an **NSInvocation** on an object you choose.
 - ▶ basically the same as fast forwarding but more low level and a bit more verbose
- ▶ So how does this work?
 - ▶ implement **+instanceMethodSignatureForSelector:**
 - ▶ check the selector
 - ▶ provide an **NSMethodSignature*** that describes the selector
 - ▶ implement **forwardInvocation:**

Demo

normal forwarding

Swizzling

- ▶ Swizzling is exchanging the implementation of one *factor* of the runtime with another *factor*. In Objective-C, you can apply this on two levels: method swizzling and class swizzling.
- ▶ Method swizzling
- ▶ Class swizzling
- ▶ Dynamic class generation

Method swizzling

- ▶ You need to have two methods with an implementation
- ▶ Can exchange the implementation of the methods with each other
- ▶ Not only in your own code, but you can modify framework code too! (eg UIView, ...)

Demo

method swizzling

Class swizzling

- ▶ No real swizzling...
- ▶ Just change the class on an existing object
 - ▶ best used with subclasses or classes with the same layout/interface
 - ▶ memory allocation is not changed when changing classes
 - ▶ otherwise: **NUKULAR EXCEPTION**

Demo

class swizzling

Dynamic class generation

- ▶ Generate a class at runtime
 - ▶ Provide methods and implementations as you see fit
 - ▶ add new functionality
 - ▶ change existing functionality

Demo

dynamic class generation

For closing...

- ▶ Generated properties
 - ▶ provide property storage in your own backing (eg plist)
- ▶ No implementations in code
 - ▶ generate them at runtime
 - ▶ only provide methods in interface
 - ▶ no compiler warnings

Demo

property generation

Warning-fixing

- ▶ When providing dynamic implementations of selectors, the compiler will emit warnings for the “unimplemented” messages.
 - ▶ Fix these by placing them in a category instead of in the `@interface` declaration

```
@interface AwesomeClass
@end

@interface AwesomeClass (Dynamic)

// look ma, no warning
- (void)withoutCodeButDynamicallyGenerated;

@end
```

- ▶ For properties
 - ▶ declare a property as you normally would using `@property` syntax in your `@interface`
 - ▶ specify `@dynamic <propertyName>` in your `@implementation` to make sure the compiler doesn't autosynthesize the property
 - ▶ or use the same technique as above

Opensourced examples

- ▶ You can find the example projects use to demo each aspect in my Github account:
- ▶ <https://developer.apple.com/library/mac/documentation/cocoa/reference/objcruntimeref/Reference/reference.html>

Useful References

- ▶ Apple's runtime documentation:
 - ▶ runtime reference: <https://developer.apple.com/library/mac/documentation/cocoa/reference/objcruntime/Reference/reference.html>
 - ▶ programming guide: <https://developer.apple.com/library/mac/documentation/cocoa/conceptual/objcruntimeguide/objcruntimeguide.pdf>
- ▶ Mike Ash's blog: <http://www.mikeash.com/pyblog>
 - ▶ objective-c: <http://www.mikeash.com/pyblog/?tag=objectiveC>
 - ▶ friday Q&A: <http://www.mikeash.com/pyblog/?tag=fridayqna>
- ▶ Jon Rentzsch swizzling helpers:
 - ▶ <https://github.com/rentzsch/jrswizzle>

Thanks for listening.

Questions? Contact me:

Twitter: **@inferis**

App.Net: **@inferis**

E-mail: **tom@interfaceimplementation.be**

vCard: **<http://inferis.org>**