iOS Dev Camp #3 Week 4
Foundation Framework with value

Edward Chiang

2014.10.20 - 2014.10.24

Today

- NSObject
  - Initialize a class
- Creating, Copying and Deallocating
- Sending Messages

NSObject

NSObject

- Base class for pretty much every object in the iOS SDK.
  - (NSString *)description is a useful method to override (it's %@ in NSLog()).

- Copying objects.
  - (id)copy;
  - (id)mutableCopy;

- It's not uncommon to have an array or dictionary and make a mutableCopy and modify.

Initializing a Class

- Initialized the class before it receives its first message

  + (void)initialize

- The runtime sends initialize to each clans in program just before the class, or any class that inherits from its, is sent its first message from within the program.

```
+ (void)initialize {
    if (self == [ClassName self]) {
        // … do the initialization
    }
}
```

- Initialize is called in a thread-safe manner and the order of initialize being called on different classes is not guaranteed, it's important to do the minimum amount of work necessary in the methods.

Creating, Copying and Deallocating

Alloc

+ alloc

- Return a new instance of the receiving class.
- Must use an init… method to complete the initialization process.

```
TheClass *newObject = [[TheClass alloc] init];
```

- Do not override alloc to include initialization code. Instead, implement class-specific versions of init… methods.

Init

- init

- Implemented by subclasses to initialize to a new object (the receiver) immediately after memory for it has been allocated.
- Must use an init… method to complete the initialization process.

```
- (id)init {
    self = [super init];
    if (self) {
        // Initialize self.
    }
}
```

- An object isn't ready to be used until it has been initialized. The init method defined in the NSObject class does no initialization; it simply returns self.

Copy

- copy

- The object returned by the NSCopying protocol method copyWithZone: .
- This is a convenience method for classes that adopt the NSCopying protocol.
- An exception is raised if there is no implementation for copyWithZone:.
- NSObject does not itself support the NSCopying protocol. Subclasses must support the protocol and implement the copyWithZone: method.
- A subclass version of the copyWithZone: method should send the message to super first, to incorporate its implementation, unless the subclass descends directly from NSObject.

Copy

+ copyWithZone:

- This method exists so class objects can be used in situations where you need an object that conforms to the NSCopying protocol. For example, this method lets you use a class object as a key to an NSDictionary object. You should not override this method.

Dealloc

- dealloc:

- Deallocates the memory occupied by the receiver.
- Subsequent messages to the receiver may generate an error indicating that a message was sent to a deallocated object (provided the deallocated memory hasn't been reused yet).
- You never send a dealloc message directly. Instead, an object's dealloc method is invoked by the runtime.

Sending Messages

Sending Messages

- performSelector:withObject:afterDelay:

- Invokes a method of the receiver on the current thread using the default mode after a delay.
- aSelector A selector that identifies the method to invoke. The method should not have a significant return value and should take a single argument of type id, or no arguments.
- anArgument The argument to pass to the method when it is invoked. Pass nil if the method does not take an argument.
- delay The minimum time before which the message is sent. Specifying a delay of 0 does not necessarily cause the selector to be performed immediately. The selector is still queued on the thread's run loop and performed as soon as possible.

Sending Messages

- This method sets up a timer to perform the aSelector message on the current thread's run loop. The timer is configured to run in the default mode (NSDefaultRunLoopMode). When the timer fires, the thread attempts to dequeue the message from the run loop and perform the selector. It succeeds if the run loop is running and in the default mode; otherwise, the timer waits until the run loop is in the default mode.

Sending Messages

- If you want the message to be dequeued when the run loop is in a mode other than the default mode, use the performSelector:withObject:afterDelay:inModes: method instead. If you are not sure whether the current thread is the main thread, you can use the performSelectorOnMainThread:withObject:waitUntilDone: or performSelectorOnMainThread:withObject:waitUntilDone:modes: method to guarantee that your selector executes on the main thread. To cancel a queued message, use the cancelPreviousPerformRequestsWithTarget: or cancelPreviousPerformRequestsWithTarget:selector:object: method.

Todays Homework

- Creating A New Class
  - Subclass of: NSObject
  - Name it with your class name
  - Save it in a suitable directory within your project

~ END ~

http://www.alphacamp.tw