Foundation Framework

448460
Fall 2016
09/27/2016
Kyoung Shin Park
Multimedia Engineering
Dankook University

Foundation Framework

- Value and collection classes
- Archiving
- Notifications
- Undo manager
- Tasks, timers, threads
- File system, piles, I/O, bundles

NSObject

- Cocoa root class for all UIKit and Foundation classes
- Base class for all Objective-C classes
- NSObject is imported into **Swift** with **NSObjectProtocol**
- Implements many basics
  - Memory management
  - Introspection (**isSubclassOfClass, isKindsOfClass**)
  - Object equality (**isEqual, hash**) - **Equatable & Hashable protocol**
  - Function availability checking (**respondsToSelector**)
  - Describing (**description**)

Class Introspection

- All objects that inherit from NSObject know these methods.
  - **isSubclassOfClass** : returns whether an object is a subclass of a given class
  - **isKindOfClass** : returns whether an object is that kind of class (inheritance included)
  - **respondsToSelector** : returns whether an object responds to a given method

NSObject

```
var str: NSString = "Swift"
var result = str.isKindOfClass(NSString) // true
result = str.isKindOfClass(NSObject) // true
result = str.isKindOfClass(NSArray) // false
result = str is String // Warning: 'is' test is always true
result = str is NSObject // Warning: 'is' test is always true
result = str is NSMutableString // true
result = str is NSArray // Warning: Cast from 'NSString' to unrelated type 'NSArray' always fails
result = NSString.isSubclassOfClass(NSObject) // true
result = NSString.instancesRespondToSelector(Selector("compare:")) // true
result = str.respondsToSelector(Selector("compare:")) // true
```

NSObject

```
class MyPoint : NSObject {
    var x = 5; var y = 5
    func set(x: Int, y: Int) {
        self.x = x; self.y = y
    }
    override func isEqual(object: AnyObject?) -> Bool {
        if let point = object as? MyPoint {
            return (self.x == point.x) && (self.y == point.y)
        } else {
            return false
        }
    }
    override var hash: Int {
        return self.x.hashValue ^ self.y.hashValue
    }
}
```

NSObject

```
var a = MyPoint()
var b = MyPoint()
var set = NSMutableSet()
a.set(10, y: 10)
b.set(10, y:10)
set.addObject(a)
a == b // true
a.isEqual(b) // true
set.containsObject(b) // true
let u = MyPoint.instancesRespondToSelector(Selector("set:y:")) // true (because MyPoint class contains set:y: method)
let v = a.respondsToSelector(Selector("set:")) // false
let w = a.respondsToSelector(Selector("set:y:")) // true
```

NSNumber/NSValue

- NSNumber
  - NSNumber is generic number holding class

```
let n = NSNumber(35.5)
let intVal = n.intValue // also doubleValue, boolValue, etc
```

  - Subclass of NSValue
- NSValue
  - Generic object wrapper for other non-object data types

```
let rect = CGRect(x: 0, y: 0, width: 1, height: 1)
let val = NSValue(CGRect: rect)

let point = CGPointMake(25.0, 15.0)
var pointVal = NSValue(CGPoint: point)
```

NSData/NSMutableData
NSDate/NSCalendarDate

- NSData/NSMutableData
  - Arbitrary sets of bytes
  - Used to save/restore/transmit data throughout the iOS SDK
- NSDate/NSCalendarDate
  - Times and dates
  - Used to find out the time/date right now or to store past or future times/dates
  - See also, NSCalendar, NSDateFormatter, NSDateComponents
  - If you are displaying a data in your UI, there are localization ramifications

NSDateComponents/NSDateFormatter

```
let now = NSDate() // the exact moment this initializer is called
let todayComponents = NSDateComponents() // if you need to specify a particular NSDate
todayComponents.year = 2015
todayComponents.month = 9
todayComponents.day = 21
todayComponents.hour = 7
todayComponents.minute = 0
todayComponents.second = 0
let today = NSCalendar.currentCalendar().dateFromComponents(todayComponents)!
/***** NSDateFormatter *****/
let formatter = NSDateFormatter()
formatter.dateStyle = NSDateFormatterStyle.LongStyle
formatter.timeStyle = .MediumStyle
let dateString = formatter.stringFromDate(today)
//dateString = " September 21, 2015 at 7:00:00 AM"
```

NSString/NSMutableString

- NSString manages immutable strings.
- NSString is implemented to represent an array of **Unicode** characters, i.e., a text string.
- NSMutableString subclasses NSString.
- NSMutableString allows a string to be modified.
- Append strings

  func appendString(_ aString: String)

  func appendFormat(_ format: String, _ arguments: CVarArgType …)
- Combine strings

  func stringByAppendingFormat(_ format: String, _ arguments: CVarArgType…) -> String

  func stringByAppendingString(_ aString: String) -> String

NSString

- NSString has **length** and **characterAtIndex:** methods

```
let str1 =  “Hello” // String
let str2: NSString = “Hello, Swift”
var length = str2.length // 12
var str3 = String() // String
var str4 = String(“Hello”) // String
var unichar = (str4 as NSString).characterAtIndex(1) // “e”
var str5 = String(count: 3, repeatedValue: Character(“A”)) // AAA
var str6 = NSString()
var str7 = NSString(string: “Hello”)
print(str7.description) // “Hello”
var str8 = NSMutableString(capacity: 10)
```

NSString

- Divide strings

```
func substringFromIndex(_ anIndex: Int) -> String
func substringToIndex(_ anIndex: Int) -> String
func substringWithRange(_ aRange: NSRange) -> String
func componentsSeparatedByString(_ separator: String) -> [String]
let str = “Hello, Swift Programming Language!”
let startIndex = advance(str.startIndex, 7)
let str2 = str.substringFromIndex(startIndex)
=> str2 = Swift Programming Language!
let str3: NSString = str
let str4 = str3.substringFromIndex(7)
=> Swift Programming Language!
let list = str.componentsSeparatedByString(“ ”)
=> [“Hello,”,  “Swift”, “Programming”, “Language!”]
```

NSString

- Find characters and substrings

```
func rangeOfCharacterFromSet(_ aSet: NSCharacterSet) ->
NSRange
func rangeOfString(_ aString: String) -> NSRange
let charSet = NSCharacterSet(charactersInString: “,#$”)
let str = “Hello, Swift#Programming$Language”
if let range = str.rangeOfCharacterFromSet(charSet) {
    print(range.startIndex)
}
if let range = str.rangeOfString(“Swift”) {
    print(range.startIndex)
}
```

Collections

- NSArray – ordered collection of objects
- NSDictionary – collection of key-value pairs
- NSSet – unordered collection of unique objects
- Common enumeration mechanism
- Immutable and mutable versions
  - Immutable collections can be shared without side effect
  - Prevent unexpected changes
  - Mutable objects typically carry a performance overhead

NSArray

- array

```
var array = NSArray() // empty array
```

- arrayWithObjects method is not available on swift
- arrayWithArray

```
var tempArray: NSArray = NSArray(array: [“Dog”, “Cat”])
var arrArray = NSArray(array: tempArray)
```

- arrayWithContentsOfFile

```
var file = NSBundle.mainBundle().pathForResource(“Data”,
ofType: “plist”)
var fileArray = NSArray(contentsofFile: “file”)
```

- arrayWithContentsOfURL

```
var url = NSURL(string: “http://..../sample-array-plist.plist”)
var urlArray = NSArray(contentsofURL: url!)
```

NSMutableArray

- NSMutableArray subclasses NSArray
- create/init

```
var array: NSMutableArray = NSMutableArray() // empty array
var array: NSMutableArray = NSMutableArray(capacity: 1)
var array: NSMutableArray = NSMutableArray(array: [“Dog”, “Cat”])
```

- add/remove/insert/replace

```
array.addObject(“Horse”)
array.addObjectsFromArray([“Cow”, “Hen”])
array.insertObject(“Frog”, atIndex: 1)
array.removeObject(“Cat”)
array.removeObjectAtIndex(0)
array.removeLastObject()
array.removeObjectInRange(NSMakeRange(1, 1))
array.removeAllObjects()
```

NSDictionary

- Hash table. Look up objects using a key to get a value.
- Common NSDictionary methods

```
var count: Int { get }
func objectForKey(_ aKey: AnyObject) -> AnyObject?
func keyEnumerator() -> NSEnumerator
func objectEnumerator() -> NSEnumerator
```

- nil returned if no object found for given key

```
var colors: NSDictionary = [1: “Red”, 2: “Green”, 3: “Blue”]
let first: NSString = (colors.objectsForKey(1) as? NSString)! // Red
if  let second = colors.objectForKey(2) {
      print(second) // Green
}
for c in colors.objectEnumerator() {
   print(c) // Red, Green, Blue
}
```

NSMutableDictionary

- NSMutableDictionary subclasses NSDictionary
- Common NSMutableDictionary methods

```
func setDictionary(otherDictionary: [NSObject: AnyObject])
func setObject(anObject: AnyObject, forKey: NSCopying)
func removeObjectForKey(aKey: AnyObject)
func removeAllObjects()

var colors: NSMutableDictionary = [1: “Red”, 2: “Green”, 3: “Blue”]
colors.setObject(“Orange”, forKey: 4) // add “4: Orange”
colors.setObject(“Cyan”, forKey: 5) // add “5: Cyan”
colors.removeObjectForKey(3) // remove “3: Blue”
colors.setDictionary([7: “Margenta”, 8: “Purple”])
for c in colors.objectEnumerator() {
   print(c) // Margenta, Purple
}
```

Objective-C Compatibility

Objective-C Compatibility

- Bridging
  - iOS was developed in a language called Objective-C.
  - Virtually all of the iOS API is accessible seamlessly from Swift.
  - A few special data types are handled specially (and powerfully) via bridging.
  - Bridging means that you can use the interchangeably.

NSString <=> **String**
NSArray <=> **Array<AnyObject>**
NSDictionary <=> **Dictionary<NSObject, AnyObject>**

Int, Float, Double, Bool => **NSNumber** (but not vice-versa)
To get from NSNumber to these types use doubleValue, intValue, etc.

Objective-C Compatibility

- Casting to/from bridged types
  - You can also "cast" (reliably, i.e., no need for as?) to/from a bridged type.

```
let length = (aString as NSString).length // length is an NSString method
(anArray as NSArray).componentsJoinedByString(NSString) // componentsJoinedByString is an NSArray method
```

  - **String, Array and Dictionary are structs**, not objects (classes). But they can still be an AnyObject. That's because they are bridged to these NS versions which are objects.

Property List

- The term "**Property List**" just means a collection of objects, containing only the following classes:
  - **NSArray, NSDictionary, NSNumber, NSString, NSDate, NSData**
- An NSArray is a Property List if all its members are too
  - So an NSArray of NSString is a Property List
  - So is an NSArray of NSArray as long as those NSArray's members are Property Lists
- An NSDictionary is one only if all keys and values are too
  - An NSArray of NSDictionarys whose keys are NSStrings and values are NSNumbers is one.

Property List

- In Swift, the definition of Property List is exactly the same and the bridging all works.
  - Handling Property Lists usually requires a fair amount of casting (i.e., is and as). That's because it's an AnyObject, so you have to figure out if it's what you expect.
  - Property Lists are used to pass around data "blindly". The semantics of the contents of a Property List are known only to its creator.
  - Property Lists are also used as a "generic data structure". And so can be passed to API that reads/writes generic data.

NSUserDefaults

- A storage mechanism for Property List data
  - It's essentially a very tiny database that stores Property List data.
  - It persists between launchings of your application.
  - Great for things like "settings" and such.
  - Do not use it for anything big!
  - It can store/retrieve entire Property Lists by name (keys)

```
setObject(AnyObject, forKey: String) // AnyObject must be a Property List
objectForKey(String) -> AnyObject?
arrayForKey(String) -> Array<AnyObject>? // returns nil if value is not set or not an array
```

  - It can also store/retrieve little pieces of data

```
setDouble(Double, forKey: String)
doubleForKey(String) -> Double // not an optional, returns 0 if no such key
```

NSUserDefaults

- Using NSUserDefaults
  - Get the defaults reader/writer

```
let defaults = NSUserDefaults.standardUserDefaults()
```

  - Then read and write

```
let plist: AnyObject = defaults.objectForKey(String)
defaults.setObject(AnyObject, forKey: String) // AnyObject must be a PropertyList
```

  - Your changes will be automatically saved. But you can be sure they are saved at any time by synchronizing.

```
if !defaults.synchronize() { /* when failed! Not much you can do about it */ }
```

References

- https://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/ObjC_classic/index.html#//apple_ref/doc/uid/20001091
- https://developer.apple.com/library/ios/documentation/CoreFoundation/Conceptual/CFStrings/introCFStrings.html#//apple_ref/doc/uid/10000131i
- Lecture 5 Slide from Developing iOS8 Apps with Swift (Winter 2015) @Stanford University