NSObject Protocol Reference

Data Management: Data Types & Collections

2009-11-23

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

NSObject Protocol Reference

Adopted by	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Declared in	NSObject.h
Companion guides	Cocoa Fundamentals Guide Memory Management Programming Guide

Overview

The `NSObject` protocol groups methods that are fundamental to all Objective-C objects.

If an object conforms to this protocol, it can be considered a first-class object. Such an object can be asked about its:

- Class, and the place of its class in the inheritance hierarchy
- Conformance to protocols
- Ability to respond to a particular message

In addition, objects that conform to this protocol—with its retain (page 16), release (page 14), and autorelease (page 7) methods—can also integrate with the object management and deallocation scheme defined in Foundation (for more information see, for example, *Memory Management Programming Guide*). Thus, an object that conforms to the `NSObject` protocol can be managed by container objects like those defined by `NSArray` and `NSDictionary`.

The Cocoa root class, `NSObject`, adopts this protocol, so all objects inheriting from `NSObject` have the features described by this protocol.

Tasks

Identifying Classes

`- class` (page 8) *required method*

Returns the class object for the receiver's class. (required)

- `superclass` (page 18) *required method*
  Returns the class object for the receiver's superclass. (required)

Identifying and Comparing Objects

- `isEqual:` (page 10) *required method*
  Returns a Boolean value that indicates whether the receiver and a given object are equal. (required)
- `hash` (page 9) *required method*
  Returns an integer that can be used as a table address in a hash table structure. (required)
- `self` (page 17) *required method*
  Returns the receiver. (required)

Managing Reference Counts

- `retain` (page 16) *required method*
  Increments the receiver's reference count. (required)
- `release` (page 14) *required method*
  Decrements the receiver's reference count. (required)
- `autorelease` (page 7) *required method*
  Adds the receiver to the current autorelease pool. (required)
- `retainCount` (page 16) *required method*
  Returns the receiver's reference count. (required)

Testing Object Inheritance, Behavior, and Conformance

- `isKindOfClass:` (page 10) *required method*
  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. (required)
- `isMemberOfClass:` (page 11) *required method*
  Returns a Boolean value that indicates whether the receiver is an instance of a given class. (required)
- `respondsToSelector:` (page 15) *required method*
  Returns a Boolean value that indicates whether the receiver implements or inherits a method that can respond to a specified message. (required)
- `conformsToProtocol:` (page 8) *required method*
  Returns a Boolean value that indicates whether the receiver conforms to a given protocol. (required)

Describing Objects

- `description` (page 9) *required method*
  Returns a string that describes the contents of the receiver. (required)

Sending Messages

- `performSelector:` (page 12) *required method*
  Sends a specified message to the receiver and returns the result of the message. (required)
- `performSelector:withObject:` (page 13) *required method*
  Sends a message to the receiver with an object as the argument. (required)
- `performSelector:withObject:withObject:` (page 13) *required method*
  Sends a message to the receiver with two objects as arguments. (required)

Determining Allocation Zones

- `zone` (page 18) *required method*
  Returns a pointer to the zone from which the receiver was allocated. (required)

Identifying Proxies

- `isProxy` (page 12) *required method*
  Returns a Boolean value that indicates whether the receiver does not descend from `NSObject`. (required)

Instance Methods

autorelease

Adds the receiver to the current autorelease pool. (required)

```
- (id)autorelease
```

Return Value
`self`.

Discussion
You add an object to an autorelease pool so it will receive a `release` message—and thus might be deallocated—when the pool is destroyed. For more information on the autorelease mechanism, see *Memory Management Programming Guide*.

Special Considerations

If garbage collection is enabled, this method is a no-op.

Availability
Available in iOS 2.0 and later.

See Also
- `retain` (page 16)

Related Sample Code
BonjourWeb

CryptoExercise
GKRocket
GKTank
WiTap

Declared In
`NSObject.h`

class

Returns the class object for the receiver's class. (required)

```
- (Class)class
```

Return Value
The class object for the receiver's class.

Availability
Available in iOS 2.0 and later.

See Also
`class` (`NSObject` class)

Declared In
`NSObject.h`

conformsToProtocol:

Returns a Boolean value that indicates whether the receiver conforms to a given protocol. (required)

```
- (BOOL)conformsToProtocol:(Protocol *)aProtocol
```

Parameters

aProtocol
A protocol object that represents a particular protocol.

Return Value
`YES` if the receiver conforms to *aProtocol*, otherwise `NO`.

Discussion
This method works identically to the `conformsToProtocol:` class method declared in `NSObject`. It's provided as a convenience so that you don't need to get the class object to find out whether an instance can respond to a given set of messages.

Availability
Available in iOS 2.0 and later.

Declared In
`NSObject.h`

description

Returns a string that describes the contents of the receiver. (required)

```
- (NSString *)description
```

Return Value
A string that describes the contents of the receiver.

Discussion
The debugger's print-object command indirectly invokes this method to produce a textual description of an object.

Availability
Available in iOS 2.0 and later.

Declared In
`NSObject.h`

hash

Returns an integer that can be used as a table address in a hash table structure. (required)

```
- (NSUInteger)hash
```

Return Value
An integer that can be used as a table address in a hash table structure.

Discussion
If two objects are equal (as determined by the `isEqual:` (page 10) method), they must have the same hash value. This last point is particularly important if you define `hash` in a subclass and intend to put instances of that subclass into a collection.

If a mutable object is added to a collection that uses hash values to determine the object's position in the collection, the value returned by the `hash` method of the object must not change while the object is in the collection. Therefore, either the `hash` method must not rely on any of the object's internal state information or you must make sure the object's internal state information does not change while the object is in the collection. Thus, for example, a mutable dictionary can be put in a hash table but you must not change it while it is in there. (Note that it can be difficult to know whether or not a given object is in a collection.)

Availability
Available in iOS 2.0 and later.

See Also
`- isEqual:` (page 10)

Related Sample Code
CryptoExercise

Declared In
`NSObject.h`

isEqual:

Returns a Boolean value that indicates whether the receiver and a given object are equal. (required)

```
- (BOOL)isEqual:(id)anObject
```

Parameters

anObject

The object to be compared to the receiver.

Return Value

`YES` if the receiver and *anObject* are equal, otherwise `NO`.

Discussion

This method defines what it means for instances to be equal. For example, a container object might define two containers as equal if their corresponding objects all respond `YES` to an `isEqual:` request. See the `NSData`, `NSDictionary`, `NSArray`, and `NSString` class specifications for examples of the use of this method.

If two objects are equal, they must have the same hash value. This last point is particularly important if you define `isEqual:` in a subclass and intend to put instances of that subclass into a collection. Make sure you also define `hash` (page 9) in your subclass.

Availability

Available in iOS 2.0 and later.

See Also

`- hash` (page 9)

Declared In

`NSObject.h`

isKindOfClass:

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. (required)

```
- (BOOL)isKindOfClass:(Class)aClass
```

Parameters

aClass

A class object representing the Objective-C class to be tested.

Return Value

`YES` if the receiver is an instance of *aClass* or an instance of any class that inherits from *aClass*, otherwise `NO`.

Discussion

For example, in this code, `isKindOfClass:` would return `YES` because, in Foundation, the `NSArchiver` class inherits from `NSCoder`:

```
NSMutableData *myData = [NSMutableData dataWithCapacity:30];
id anArchiver = [[NSArchiver alloc] initForWritingWithMutableData:myData];
if ( [anArchiver isKindOfClass:[NSCoder class]] )
    ...
```

Be careful when using this method on objects represented by a class cluster. Because of the nature of class clusters, the object you get back may not always be the type you expected. If you call a method that returns a class cluster, the exact type returned by the method is the best indicator of what you can do with that object. For example, if a method returns a pointer to an `NSArray` object, you should not use this method to see if the array is mutable, as shown in the following code:

```
// DO NOT DO THIS!
if ([myArray isKindOfClass:[NSMutableArray class]])
{
    // Modify the object
}
```

If you use such constructs in your code, you might think it is alright to modify an object that in reality should not be modified. Doing so might then create problems for other code that expected the object to remain unchanged.

If the receiver is a class object, this method returns `YES` if *`aClass`* is a Class object of the same type, `NO` otherwise.

Availability
Available in iOS 2.0 and later.

See Also
`- isMemberOfClass:` (page 11)

Declared In
`NSObject.h`

isMemberOfClass:

Returns a Boolean value that indicates whether the receiver is an instance of a given class. (required)

```
- (BOOL)isMemberOfClass:(Class)aClass
```

Parameters

`aClass`

A class object representing the Objective-C class to be tested.

Return Value
`YES` if the receiver is an instance of *`aClass`*, otherwise `NO`.

Discussion
For example, in this code, `isMemberOfClass:` would return `NO`:

```
NSMutableData *myData = [NSMutableData dataWithCapacity:30];
id anArchiver = [[NSArchiver alloc] initForWritingWithMutableData:myData];
if ([anArchiver isMemberOfClass:[NSCoder class]])
    ...
```

Class objects may be compiler-created objects but they still support the concept of membership. Thus, you can use this method to verify that the receiver is a specific Class object.

Availability
Available in iOS 2.0 and later.

See Also
- `isKindOfClass:` (page 10)

Declared In
`NSObject.h`

isProxy

Returns a Boolean value that indicates whether the receiver does not descend from `NSObject`. (required)

```
- (BOOL)isProxy
```

Return Value
`NO` if the receiver really descends from `NSObject`, otherwise `YES`.

Discussion
This method is necessary because sending `isKindOfClass:` (page 10) or `isMemberOfClass:` (page 11) to an `NSProxy` object will test the object the proxy stands in for, not the proxy itself. Use this method to test if the receiver is a proxy (or a member of some other root class).

Availability
Available in iOS 2.0 and later.

Declared In
`NSObject.h`

performSelector:

Sends a specified message to the receiver and returns the result of the message. (required)

```
- (id)performSelector:(SEL)aSelector
```

Parameters

aSelector

A selector identifying the message to send. If *aSelector* is `NULL`, an `NSInvalidArgumentException` is raised.

Return Value
An object that is the result of the message.

Discussion
The `performSelector:` method is equivalent to sending an *aSelector* message directly to the receiver. For example, all three of the following messages do the same thing:

```
id myClone = [anObject copy];
id myClone = [anObject performSelector:@selector(copy)];
id myClone = [anObject performSelector:sel_getUid("copy")];
```

However, the `performSelector:` method allows you to send messages that aren't determined until runtime. A variable selector can be passed as the argument:

```
SEL myMethod = findTheAppropriateSelectorForTheCurrentSituation();
[anObject performSelector:myMethod];
```

The *aSelector* argument should identify a method that takes no arguments. For methods that return anything other than an object, use NSInvocation.

Availability
Available in iOS 2.0 and later.

See Also
- performSelector:withObject: (page 13)
- performSelector:withObject:withObject: (page 13)

Declared In
NSObject.h

performSelector:withObject:

Sends a message to the receiver with an object as the argument. (required)

```
- (id)performSelector:(SEL)aSelector withObject:(id)anObject
```

Parameters

aSelector
A selector identifying the message to send. If *aSelector* is NULL, an NSInvalidArgumentException is raised.

anObject
An object that is the sole argument of the message.

Return Value
An object that is the result of the message.

Discussion
This method is the same as performSelector: (page 12) except that you can supply an argument for *aSelector*. *aSelector* should identify a method that takes a single argument of type id. For methods with other argument types and return values, use NSInvocation.

Availability
Available in iOS 2.0 and later.

See Also
- performSelector:withObject:withObject: (page 13)

methodForSelector: (NSObject class)

Declared In
NSObject.h

performSelector:withObject:withObject:

Sends a message to the receiver with two objects as arguments. (required)

```
- (id)performSelector:(SEL)aSelector withObject:(id)anObject
    withObject:(id)anotherObject
```

Parameters

aSelector

A selector identifying the message to send. If *aSelector* is NULL, an NSInvalidArgumentException is raised.

anObject

An object that is the first argument of the message.

anotherObject

An object that is the second argument of the message

Return Value

An object that is the result of the message.

Discussion

This method is the same as performSelector: (page 12) except that you can supply two arguments for *aSelector*. *aSelector* should identify a method that can take two arguments of type id. For methods with other argument types and return values, use NSInvocation.

Availability

Available in iOS 2.0 and later.

See Also

- performSelector:withObject: (page 13)

methodForSelector: (NSObject class)

Declared In

NSObject.h

release

Decrements the receiver's reference count. (required)

```
- (oneway void)release
```

Discussion

The receiver is sent a dealloc message when its reference count reaches 0.

You would only implement this method to define your own reference-counting scheme. Such implementations should not invoke the inherited method; that is, they should not include a release message to super.

For more information on the reference counting mechanism, see *Memory Management Programming Guide*.

Special Considerations

If garbage collection is enabled, this method is a no-op.

You must complete the object initialization (using an init method) before invoking release. For example, the following code shows an error:

```
id anObject = [MyObject alloc];
[anObject release];
```

You may call release from within an init method if initialization fails for some reason provided that you have at least called superclass's designated initializer.

Availability
Available in iOS 2.0 and later.

Related Sample Code
BonjourWeb
CryptoExercise
GKRocket
ScrollViewSuite
SpeakHere

Declared In
`NSObject.h`

respondsToSelector:

Returns a Boolean value that indicates whether the receiver implements or inherits a method that can respond to a specified message. (required)

```
- (BOOL)respondsToSelector:(SEL)aSelector
```

Parameters

aSelector
A selector that identifies a message.

Return Value
`YES` if the receiver implements or inherits a method that can respond to *aSelector*, otherwise `NO`.

Discussion
The application is responsible for determining whether a `NO` response should be considered an error.

You cannot test whether an object inherits a method from its superclass by sending `respondsToSelector:` to the object using the `super` keyword. This method will still be testing the object as a whole, not just the superclass's implementation. Therefore, sending `respondsToSelector:` to `super` is equivalent to sending it to `self`. Instead, you must invoke the `NSObject` class method `instancesRespondToSelector:` directly on the object's superclass, as illustrated in the following code fragment.

```
if( [MySuperclass instancesRespondToSelector:@selector(aMethod)] ) {
    // invoke the inherited method
    [super aMethod];
}
```

You cannot simply use `[[self superclass] instancesRespondToSelector:@selector(aMethod)]` since this may cause the method to fail if it is invoked by a subclass.

Note that if the receiver is able to forward *aSelector* messages to another object, it will be able to respond to the message, albeit indirectly, even though this method returns `NO`.

Availability
Available in iOS 2.0 and later.

See Also
`forwardInvocation:` (`NSObject` class)
`instancesRespondToSelector:` (`NSObject` class)

Declared In
`NSObject.h`

retain

Increments the receiver's reference count. (required)

```
- (id)retain
```

Return Value
`self`.

Discussion
You send an object a `retain` message when you want to prevent it from being deallocated until you have finished using it.

An object is deallocated automatically when its reference count reaches 0. `retain` messages increment the reference count, and `release` (page 14) messages decrement it. For more information on this mechanism, see *Memory Management Programming Guide*.

As a convenience, `retain` returns `self` because it may be used in nested expressions.

You would implement this method only if you were defining your own reference-counting scheme. Such implementations must return `self` and should not invoke the inherited method by sending a `retain` message to `super`.

Special Considerations

If garbage collection is enabled, this method is a no-op.

Availability
Available in iOS 2.0 and later.

See Also
- `autorelease` (page 7)
- `release` (page 14)

Related Sample Code
BonjourWeb
CryptoExercise
GKRocket
SpeakHere
WiTap

Declared In
`NSObject.h`

retainCount

Returns the receiver's reference count. (required)

```
- (NSUInteger)retainCount
```

Return Value
The receiver's reference count.

Discussion
You might override this method in a class to implement your own reference-counting scheme. For objects that never get released (that is, their `release` (page 14) method does nothing), this method should return `UINT_MAX`, as defined in `<limits.h>`.

The `retainCount` method does not account for any pending `autorelease` (page 7) messages sent to the receiver.

Important: This method is typically of no value in debugging memory management issues. Because any number of framework objects may have retained an object in order to hold references to it, while at the same time autorelease pools may be holding any number of deferred releases on an object, it is very unlikely that you can get useful information from this method.

To understand the fundamental rules of memory management that you must abide by, read "Memory Management Rules". To diagnose memory management problems, use a suitable tool:

- The LLVM/Clang Static analyzer can typically find memory management problems even before you run your program.
- The Object Alloc instrument in the Instruments application (see *Instruments User Guide*) can track object allocation and destruction.
- Shark (see *Shark User Guide*) also profiles memory allocations (amongst numerous other aspects of your program).

Special Considerations

If garbage collection is enabled, the return value is undefined.

Availability
Available in iOS 2.0 and later.

See Also
- `autorelease` (page 7)
- `retain` (page 16)

Related Sample Code
CryptoExercise

Declared In
`NSObject.h`

self

Returns the receiver. (required)

```
- (id)self
```

Return Value
The receiver.

Availability
Available in iOS 2.0 and later.

See Also
`- class` (page 8)

Related Sample Code
BonjourWeb
KeyboardAccessory

Declared In
`NSObject.h`

superclass

Returns the class object for the receiver's superclass. (required)

```
- (Class)superclass
```

Return Value
The class object for the receiver's superclass.

Availability
Available in iOS 2.0 and later.

See Also
`superclass` (`NSObject` class)

Declared In
`NSObject.h`

zone

Returns a pointer to the zone from which the receiver was allocated. (required)

```
- (NSZone *)zone
```

Return Value
A pointer to the zone from which the receiver was allocated.

Discussion
Objects created without specifying a zone are allocated from the default zone.

Availability
Available in iOS 2.0 and later.

See Also
`allocWithZone:` (`NSObject` class)

Declared In
`NSObject.h`

Document Revision History

This table describes the changes to *NSObject Protocol Reference*.

Date	Notes
2009-11-23	Removed use of a deprecated NSString method in an example.
2008-12-22	Updated discussion of retainCount method.
2007-07-19	Updated definition of release, added a clarification to -hash, and added links to companion documents.
2006-06-28	Updated to conform to reference consistency guidelines.
2006-05-23	First publication of this content as a separate document.

Document Revision History