NSOperationQueue Class Reference

Developer

Contents

NSOperationQueue Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in iOS 2.0 and later.
Companion guide	Concurrency Programming Guide
Declared in	NSOperation.h
Related sample code	CryptoExercise ListAdder MotionGraphs MVCNetworking TopSongs

Overview

The `NSOperationQueue` class regulates the execution of a set of `NSOperation` objects. After being added to a queue, an operation remains in that queue until it is explicitly canceled or finishes executing its task. Operations within the queue (but not yet executing) are themselves organized according to priority levels and inter-operation object dependencies and are executed accordingly. An application may create multiple operation queues and submit operations to any of them.

Inter-operation dependencies provide an absolute execution order for operations, even if those operations are located in different operation queues. An operation object is not considered ready to execute until all of its dependent operations have finished executing. For operations that are ready to execute, the operation queue always executes the one with the highest priority relative to the other ready operations. For details on how to set priority levels and dependencies, see *NSOperation Class Reference* .

You cannot directly remove an operation from a queue after it has been added. An operation remains in its queue until it reports that it is finished with its task. Finishing its task does not necessarily mean that the operation performed that task to completion. An operation can also be canceled. Canceling an operation object leaves the object in the queue but notifies the object that it should abort its task as quickly as possible. For currently executing operations, this means that the operation object's work code must check the cancellation state, stop what it is doing, and mark itself as finished. For operations that are queued but not yet executing, the queue must still call the operation object's `start` method so that it can processes the cancellation event and mark itself as finished.

> **Note:** In OS X v10.6 and later, canceling an operation causes the operation to ignore any dependencies it may have. This behavior makes it possible for the queue to execute the operation's `start` method as soon as possible. The `start` method, in turn, moves the operation to the finished state so that it can be removed from the queue. In OS X v10.5, a canceled operation does not ignore its dependencies, meaning that those dependencies must complete normally before the canceled operation can run and be removed from the queue.

Operation queues usually provide the threads used to run their operations. In OS X v10.6 and later, operation queues use the `libdispatch` library (also known as Grand Central Dispatch) to initiate the execution of their operations. As a result, operations are always executed on a separate thread, regardless of whether they are designated as concurrent or non-concurrent operations. In OS X v10.5, however, operations are executed on separate threads only if their `isConcurrent` method returns `NO`. If that method returns `YES`, the operation object is expected to create its own thread (or start some asynchronous operation); the queue does not provide a thread for it.

> **Note:** In iOS 4 and later, operation queues use Grand Central Dispatch to execute operations. Prior to iOS 4, they create separate threads for non-concurrent operations and launch concurrent operations from the current thread. For a discussion of the difference between concurrent and non-concurrent operations and how they are executed, see *NSOperation Class Reference*.

For more information about using operation queues, see *Concurrency Programming Guide*.

KVO-Compliant Properties

The `NSOperationQueue` class is key-value coding (KVC) and key-value observing (KVO) compliant. You can observe these properties as desired to control other parts of your application. The properties you can observe include the following:

- `operations` - read-only property
- `operationCount` - read-only property
- `maxConcurrentOperationCount` - readable and writable property

- `suspended` - readable and writable property
- `name` - readable and writable property

Although you can attach observers to these properties, you should not use Cocoa bindings to bind them to elements of your application's user interface. Code associated with your user interface typically must execute only in your application's main thread. However, KVO notifications associated with an operation queue may occur in any thread.

For more information about key-value observing and how to attach observers to an object, see *Key-Value Observing Programming Guide* .

Multicore Considerations

It is safe to use a single `NSOperationQueue` object from multiple threads without creating additional locks to synchronize access to that object.

Additional Operation Queue Behaviors

An operation queue executes its queued operation objects based on their priority and readiness. If all of the queued operation objects have the same priority and are ready to execute when they are put in the queue—that is, their `isReady` method returns `YES`—they are executed in the order in which they were submitted to the queue. For a queue whose maximum number of concurrent operations is set to 1, this equates to a serial queue. However, you should never rely on the serial execution of operation objects. Changes in the readiness of an operation can change the resulting execution order.

Tasks

Managing Operations in the Queue

– `addOperation:` (page 8)
  Adds the specified operation object to the receiver.

– `addOperations:waitUntilFinished:` (page 9)
  Adds the specified array of operations to the queue.

– `addOperationWithBlock:` (page 9)
  Wraps the specified block in an operation object and adds it to the receiver.

– `operations` (page 12)
  Returns a new array containing the operations currently in the queue.

– `operationCount` (page 12)
  Returns the number of operations currently in the queue.

– `cancelAllOperations` (page 10)
  Cancels all queued and executing operations.

– `waitUntilAllOperationsAreFinished` (page 15)
  Blocks the current thread until all of the receiver's queued and executing operations finish executing.

Managing the Number of Running Operations

– `maxConcurrentOperationCount` (page 11)
  Returns the maximum number of concurrent operations that the receiver can execute.

– `setMaxConcurrentOperationCount:` (page 13)
  Sets the maximum number of concurrent operations that the receiver can execute.

Suspending Operations

– `setSuspended:` (page 14)
  Modifies the execution of pending operations

– `isSuspended` (page 10)
  Returns a Boolean value indicating whether the receiver is scheduling queued operations for execution.

Managing the Queue's Name

– `setName:` (page 14)
  Assigns the specified name to the receiver.

– `name` (page 11)
  Returns the name of the receiver.

Getting Specific Operation Queues

+ `currentQueue` (page 7)
  Returns the operation queue that launched the current operation.

`+ mainQueue` (page 7)
Returns the operation queue associated with the main thread.

Class Methods

currentQueue

Returns the operation queue that launched the current operation.

```
+ (id)currentQueue
```

Return Value
The operation queue that started the operation or `nil` if the queue could not be determined.

Discussion
You can use this method from within a running operation object to get a reference to the operation queue that started it. Calling this method from outside the context of a running operation typically results in `nil` being returned.

Availability
Available in iOS 4.0 and later.

Declared in
`NSOperation.h`

mainQueue

Returns the operation queue associated with the main thread.

```
+ (id)mainQueue
```

Return Value
The default operation queue bound to the main thread.

Discussion
The returned queue executes operations on the main thread. The main thread's run loop controls the execution times of these operations.

Availability
Available in iOS 4.0 and later.

Related Sample Code
AVMovieExporter

Managed App Configuration

MotionGraphs

MTAudioProcessingTap Audio Processor

Simple Core Data Relationships

Declared in
`NSOperation.h`

Instance Methods

addOperation:

Adds the specified operation object to the receiver.

```
- (void)addOperation:(NSOperation *)operation
```

Parameters

`operation`

The operation object to be added to the queue. In memory-managed applications, this object is retained by the operation queue. In garbage-collected applications, the queue strongly references the operation object.

Discussion

Once added, the specified `operation` remains in the queue until it finishes executing.

An operation object can be in at most one operation queue at a time and this method throws an `NSInvalidArgumentException` exception if the operation is already in another queue. Similarly, this method throws an `NSInvalidArgumentException` exception if the operation is currently executing or has already finished executing.

Availability

Available in iOS 2.0 and later.

See Also

`cancel` (NSOperation)
`isExecuting` (NSOperation)

Related Sample Code
MVCNetworking

Declared in
`NSOperation.h`

addOperations:waitUntilFinished:

Adds the specified array of operations to the queue.

```
- (void)addOperations:(NSArray *)ops waitUntilFinished:(BOOL)wait
```

Parameters

`ops`
The array of `NSOperation` objects that you want to add to the receiver.

`wait`
If `YES`, the current thread is blocked until all of the specified operations finish executing. If `NO`, the operations are added to the queue and control returns immediately to the caller.

Discussion
An operation object can be in at most one operation queue at a time and cannot be added if it is currently executing or finished. This method throws an `NSInvalidArgumentException` exception if any of those error conditions are true for any of the operations in the `ops` parameter.

Once added, the specified `operation` remains in the queue until its `isFinished` method returns `YES`.

Availability
Available in iOS 4.0 and later.

Declared in
`NSOperation.h`

addOperationWithBlock:

Wraps the specified block in an operation object and adds it to the receiver.

```
- (void)addOperationWithBlock:(void (^)(void))block
```

Parameters

`block`
The block to execute from the operation object. The block should take no parameters and have no return value.

Discussion

This method adds a single block to the receiver by first wrapping it in an operation object. You should not attempt to get a reference to the newly created operation object or divine its type information.

Availability

Available in iOS 4.0 and later.

See Also

`cancel` (NSOperation)
`isExecuting` (NSOperation)

Declared in

`NSOperation.h`

cancelAllOperations

Cancels all queued and executing operations.

```
- (void)cancelAllOperations
```

Discussion

This method sends a `cancel` message to all operations currently in the queue. Queued operations are cancelled before they begin executing. If an operation is already executing, it is up to that operation to recognize the cancellation and stop what it is doing.

Availability

Available in iOS 2.0 and later.

See Also

`cancel` (`NSOperation`)

Declared in

`NSOperation.h`

isSuspended

Returns a Boolean value indicating whether the receiver is scheduling queued operations for execution.

```
- (BOOL)isSuspended
```

Return Value

`NO` if operations are being scheduled for execution; otherwise, `YES`.

Discussion

If you want to know when the queue's suspended state changes, configure a KVO observer to observe the `suspended` key path of the operation queue.

Availability

Available in iOS 2.0 and later.

See Also

– `setSuspended:` (page 14)

Declared in

`NSOperation.h`

maxConcurrentOperationCount

Returns the maximum number of concurrent operations that the receiver can execute.

```
- (NSInteger)maxConcurrentOperationCount
```

Return Value

The maximum number of concurrent operations set explicitly on the receiver using the `setMaxConcurrentOperationCount:` method. If no value has been explicitly set, this method returns `NSOperationQueueDefaultMaxConcurrentOperationCount` by default.

Availability

Available in iOS 2.0 and later.

See Also

– `setMaxConcurrentOperationCount:` (page 13)

Declared in

`NSOperation.h`

name

Returns the name of the receiver.

```
- (NSString *)name
```

Return Value

The name of the receiver.

Discussion

The default value of this string is "`NSOperationQueue` *<id>*", where *<id>* is the memory address of the operation queue. If you want to know when a queue's name changes, configure a KVO observer to observe the `name` key path of the operation queue.

Availability

Available in iOS 4.0 and later.

Declared in

`NSOperation.h`

operationCount

Returns the number of operations currently in the queue.

```
- (NSUInteger)operationCount
```

Return Value

The number of operations in the queue.

Discussion

The value returned by this method reflects the instantaneous number of objects in the queue and changes as operations are completed. As a result, by the time you use the returned value, the actual number of operations may be different. You should therefore use this value only for approximate guidance and should not rely on it for object enumerations or other precise calculations.

Availability

Available in iOS 4.0 and later.

Declared in

`NSOperation.h`

operations

Returns a new array containing the operations currently in the queue.

```
- (NSArray *)operations
```

Return Value

A new array object containing the `NSOperation` objects in the order in which they were added to the queue.

Discussion

You can use this method to access the operations queued at any given moment. Operations remain queued until they finish their task. Therefore, the returned array may contain operations that are either executing or waiting to be executed. The list may also contain operations that were executing when the array was initially created but have subsequently finished.

Availability

Available in iOS 2.0 and later.

Declared in

`NSOperation.h`

setMaxConcurrentOperationCount:

Sets the maximum number of concurrent operations that the receiver can execute.

```
- (void)setMaxConcurrentOperationCount:(NSInteger)count
```

Parameters

`count`

The maximum number of concurrent operations. Specify the value `NSOperationQueueDefaultMaxConcurrentOperationCount` if you want the receiver to choose an appropriate value based on the number of available processors and other relevant factors.

Discussion

The specified value affects only the receiver and the operations in its queue. Other operation queue objects can also execute their maximum number of operations in parallel.

Reducing the number of concurrent operations does not affect any operations that are currently executing. If you specify the value `NSOperationQueueDefaultMaxConcurrentOperationCount` (which is recommended), the maximum number of operations can change dynamically based on system conditions.

Availability

Available in iOS 2.0 and later.

See Also

– `maxConcurrentOperationCount` (page 11)

Declared in

`NSOperation.h`

setName:

Assigns the specified name to the receiver.

```
- (void)setName:(NSString *)newName
```

Parameters

`newName`

The new name to associate with the receiver.

Discussion

Names provide a way for you to identify your operation queues at run time. Tools may also use this name to provide additional context during debugging or analysis of your code.

Availability

Available in iOS 4.0 and later.

Declared in

`NSOperation.h`

setSuspended:

Modifies the execution of pending operations

```
- (void)setSuspended:(BOOL)suspend
```

Parameters

`suspend`

If `YES`, the queue stops scheduling queued operations for execution. If `NO`, the queue begins scheduling operations again.

Discussion

This method suspends or resumes the execution of operations. Suspending a queue prevents that queue from starting additional operations. In other words, operations that are in the queue (or added to the queue later) and are not yet executing are prevented from starting until the queue is resumed. Suspending a queue does not stop operations that are already running.

Operations are removed from the queue only when they finish executing. However, in order to finish executing, an operation must first be started. Because a suspended queue does not start any new operations, it does not remove any operations (including cancelled operations) that are currently queued and not executing.

Availability

Available in iOS 2.0 and later.

See Also

– `isSuspended` (page 10)

Declared in

`NSOperation.h`

waitUntilAllOperationsAreFinished

Blocks the current thread until all of the receiver's queued and executing operations finish executing.

```
- (void)waitUntilAllOperationsAreFinished
```

Discussion

When called, this method blocks the current thread and waits for the receiver's current and queued operations to finish executing. While the current thread is blocked, the receiver continues to launch already queued operations and monitor those that are executing. During this time, the current thread cannot add operations to the queue, but other threads may. Once all of the pending operations are finished, this method returns.

If there are no operations in the queue, this method returns immediately.

Availability

Available in iOS 2.0 and later.

Declared in

`NSOperation.h`

Constants

Concurrent Operation Constants

Indicates the number of supported concurrent operations.

```
enum {
   NSOperationQueueDefaultMaxConcurrentOperationCount = -1
};
```

Constants

`NSOperationQueueDefaultMaxConcurrentOperationCount`

The default maximum number of operations is determined dynamically by the `NSOperationQueue` object based on current system conditions.

Available in iOS 2.0 and later.

Declared in `NSOperation.h`.

Declared in

`NSOperation.h`

Document Revision History

This table describes the changes to *NSOperationQueue Class Reference* .

Date	Notes
2013-04-23	Removed erroneous information from description of addOperationWithBlock: method.
2012-01-09	Corrected the explanation of how objects can be executed serially in an operation queue.
2010-04-23	Updated for iOS 4.0.
2009-08-19	Updated the description of the setSuspended: method.
2009-04-20	Updated for OS X v10.6.
2008-11-19	Updated the guidance related to KVO-compliant properties.
2008-10-15	Clarified ownership of operation objects when added to a queue.
2007-04-30	New document describing the methods for managing operation objects.