Class

OperationQueue

A queue that regulates the execution of operations.

iOS 2.0+ | iPadOS 2.0+ | Mac Catalyst 13.1+ | macOS 10.5+ | tvOS 9.0+ | visionOS 1.0+ | watchOS 2.0+

```
class OperationQueue
```

Overview

An operation queue invokes its queued `Operation` objects based on their priority and readiness. After you add an operation to a queue, it remains in the queue until the operation finishes its task. You can't directly remove an operation from a queue after you add it.

> **Note**
>
> Operation queues retain operations until the operations finish, and queues themselves are retained until all operations are finished. Suspending an operation queue with operations that aren't finished can result in a memory leak.

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

Determine the Execution Order

An operation queue organizes and invokes its operations according to their readiness, priority level, and interoperation dependencies. If all of the queued operations have the same `queuePriority` and the `isReady` property returns `true`, the queue invokes them in the order you added them. Otherwise, the operation queue always invokes the operation with the highest priority relative to the other ready operations.

However, don't rely on queue semantics to ensure a specific execution order of operations because changes in the readiness of an operation can change the resulting execution order. Interoperation dependencies provide an absolute execution order for operations, even if those operations are located in different operation queues. An operation object isn't ready to run until all of its dependent operations have finished running.

For details on how to set priority levels and dependencies, see Managing Dependencies in `Operation`.

Respond to Operation Cancelation

Finishing its task doesn't 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 stop 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**
>
> Canceling an operation causes the operation to ignore any dependencies it may have. This behavior makes it possible for the queue to invoke 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.

For more information about operation cancellation, see Responding to the Cancel Command in `Operation`.

Observe Operations Using Key-Value Observing

The `OperationQueue` class is key-value coding (KVC) and key-value observing (KVO) compliant. You can observe these properties to control other parts of your application. To observe the properties, use the following key paths:

- `operations` — Read-only
- `operationCount` — Read-only
- `maxConcurrentOperationCount` — Readable and writable
- `isSuspended` — Readable and writable
- `name` — Readable and writable

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

For more information about KVO and how to attach observers to an object, see the Key-Value Observing Programming Guide.

Plan for Thread Safety

You can safely use a single `OperationQueue` object from multiple threads without creating additional locks to synchronize access to that object.

Operation queues use the Dispatch framework to initiate the execution of their operations. As a result, queues always invoke operations on a separate thread, regardless of whether the operation is synchronous or asynchronous.

Topics

Accessing Specific Operation Queues

`class var main: OperationQueue`

Returns the operation queue associated with the main thread.

`class var current: OperationQueue?`

Returns the operation queue that launched the current operation.

Managing Operations in the Queue

`func addOperation(Operation)`

Adds the specified operation to the receiver.

`func addOperations([Operation], waitUntilFinished: Bool)`

Adds the specified operations to the queue.

`func addOperation(() -> Void)`

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

`func addBarrierBlock(() -> Void)`

Invokes a block when the queue finishes all enqueued operations, and prevents subsequent operations from starting until the block has completed.

`func cancelAllOperations()`

Cancels all queued and executing operations.

`func waitUntilAllOperationsAreFinished()`

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

~~`var operations: [Operation]`~~

The operations currently in the queue.

Deprecated

~~`var operationCount: Int`~~

The number of operations currently in the queue.

Deprecated

Managing the Execution of Operations

`var qualityOfService: QualityOfService`

The default service level to apply to operations that the queue invokes.

`var maxConcurrentOperationCount: Int`

The maximum number of queued operations that can run at the same time.

`class let defaultMaxConcurrentOperationCount: Int`

The default maximum number of operations to invoke concurrently in a queue.

Monitoring Progress of Operations

`var progress: Progress`

An object that represents the total progress of the operations executing in the queue.

Suspending Execution

`var isSuspended: Bool`

A Boolean value indicating whether the queue is actively scheduling operations for execution.

Configuring the Queue

`var name: String?`

The name of the operation queue.

`var underlyingQueue: dispatch_queue_t?`

The dispatch queue that the operation queue uses to invoke operations.

Scheduling Operations

`struct SchedulerTimeType`

The scheduler time type the operation queue uses.

`struct SchedulerOptions`

A type that defines options the operation queue accepts.

Default Implementations

Scheduler Implementations

Relationships

Inherits From

`NSObject`

Conforms To

```
CVarArg
Copyable
CustomDebugStringConvertible
CustomStringConvertible
Equatable
Hashable
NSObjectProtocol
ProgressReporting
Scheduler
Sendable
SendableMetatype
```

See Also

Operations

`class Operation`

An abstract class that represents the code and data associated with a single task.

`class BlockOperation`

An operation that manages the concurrent execution of one or more blocks.