

Framework

Dispatch

Execute code concurrently on multicore hardware by submitting work to dispatch queues managed by the system.

Overview

Dispatch comprises language features, runtime libraries, and system enhancements that provide systemic, comprehensive improvements to the support for concurrent code execution on multicore hardware in macOS, iOS, watchOS, and tvOS.

The BSD subsystem, Core Foundation, and Cocoa APIs have all been extended to use these enhancements to help both the system and your application to run faster, more efficiently, and with improved responsiveness. Consider how difficult it is for a single application to use multiple cores effectively, let alone doing it on different computers with different numbers of computing cores or in an environment with multiple applications competing for those cores. GCD, operating at the system level, can better accommodate the needs of all running applications, matching them to the available system resources in a balanced fashion.

Topics

- SDKs
- iOS 8.0+
- macOS 10.10+
- tvOS 9.0+
- watchOS 2.0+

- On This Page
- Overview ☺
- Topics ☺

Managing Dispatch Queues

GCD provides and manages FIFO queues to which your application can submit tasks in the form of block objects. Work submitted to dispatch queues are executed on a pool of threads fully managed by the system. No guarantee is made as to the thread on which a task executes.

Synchronous and Asynchronous Execution

Each work item can be executed either *synchronously* or *asynchronously*. When a work item is executed synchronously with the sync method, the program waits until execution finishes before the method call returns. When a work item is executed asynchronously with the async method, the method call returns immediately.

Serial and Concurrent Queues

A dispatch queue can be either *serial*, so that work items are executed one at a time, or it can be *concurrent*, so that work items are dequeued in order, but run all at once and can finish in any order. Both serial and concurrent queues process work items in *first in, first-out* (FIFO) order.

System-Provided Queues

When an app launches, the system automatically creates a special queue called the *main queue*. Work items enqueued to the main queue execute serially on your app’s main thread. You can access the main queue using the `main` type property.

Important

Attempting to synchronously execute a work item on the main queue results in dead-lock.

In addition to the serial main queue, the system also creates a number of global concurrent dispatch queues. You can access the global concurrent queue that best matches a specified quality of service (QoS) using the `global(attributes:)` type method.

```
class DispatchQueue
    DispatchQueue manages the execution of work items. Each work item submitted to a queue is processed on a pool of threads managed by the system.

struct DispatchQueue.Attributes

class DispatchSpecificKey
    DispatchSpecificKey is a class that encapsulates keys that can be associated with a contextual value on a DispatchQueue using the setSpecific<T>(key:value:) method, and accessed using the getSpecific<T>(key:) method.

enum DispatchQueue.AutoreleaseFrequency

func dispatch_get_current_queue()
    Returns the queue on which the currently executing block is running.
    

Deprecated



func setTarget(queue: DispatchQueue?)
    Sets the target queue for the given object.
```

Managing Units of Work

Work items allow you to configure properties of individual units of work directly. They also allow you to address individual work units for the purposes of waiting for their completion, getting notified about their completion, and/or canceling them.

```
class DispatchWorkItem
    DispatchWorkItem encapsulates work that can be performed. A work item can be
    dispatched onto a DispatchQueue and within a DispatchGroup. A DispatchWork
    Item can also be set as a DispatchSource event, registration, or cancel handler.

struct DispatchWorkItemFlags
    DispatchWorkItemFlags are an option set that configure the behavior of a
    DispatchWorkItem value, including its quality of service class and whether to create
    a barrier or spawn a new detached thread.
```

Prioritizing Work and Specifying Quality of Service

```
struct DispatchQoS
    DispatchQoS encapsulates quality of service classes.

enum DispatchQoS.QoSClass
    DispatchQoS.QoSClass encapsulates quality of service classes.

📖 Dispatch Queue Priorities
    Used to select the appropriate global concurrent queue.
```

Using Dispatch Groups

Grouping blocks allows for aggregate synchronization. Your application can submit multiple blocks and track when they all complete, even though they might run on different queues. This behavior can be helpful when progress can't be made until all of the specified tasks are complete.

```
class DispatchGroup
    DispatchGroup allows for aggregate synchronization of work. You can use them to
    submit multiple different work items and track when they all complete, even though they
    might run on different queues. This behavior can be helpful when progress can't be
    made until all of the specified tasks are complete.
```

Using Dispatch Semaphores

A dispatch semaphore is an efficient implementation of a traditional counting semaphore. Dispatch semaphores call down to the kernel only when the calling thread needs to be blocked. If the calling semaphore does not need to block, no kernel call is made.

```
class DispatchSemaphore
    DispatchSemaphore provides an efficient implementation of a traditional counting
    semaphore, which can be used to control access to a resource across multiple
    execution contexts.
```

Using Dispatch Data


`struct DispatchData`
DispatchData objects manage a memory-based data buffer. The data buffer is exposed as a contiguous block of memory, but internally, it may be comprised of multiple, discontinuous blocks of memory.

`struct DispatchDataIterator`

`enum DispatchData.Deallocator`

Using Dispatch Time

`struct DispatchTime`
DispatchTime represents a point in time relative to the default clock with nanosecond precision. On Apple platforms, the default clock is based on the Mach absolute time unit.

 Dispatch Time Constants
Base time constants.

`struct DispatchWallTime`
DispatchTime represents an absolute point in time according to the wall clock with microsecond precision. On Apple platforms, the default clock is based on the result of `gettimeofday(2)`.

`enum DispatchTimeInterval`
DispatchTimeInterval represents a number of seconds, milliseconds, microseconds, or nanoseconds. You use DispatchTimeInterval values to specify the interval at which a DispatchSourceTimer fires or I/O handlers are invoked for a DispatchIO channel, as well as to increment and decrement DispatchTime values.

`enum DispatchTimeoutResult`
DispatchTimeoutResult indicates whether a dispatch operation finished before a specified time.

 Dispatch Time Operators

`typealias dispatch_time_t`
A somewhat abstract representation of time.

Managing Dispatch Sources

`class DispatchSource`

`DispatchSource` provides an interface for monitoring low-level system objects such as Mach ports, Unix descriptors, Unix signals, and VFS nodes for activity and submitting event handlers to dispatch queues for asynchronous processing when such activity occurs.

`protocol DispatchSourceProtocol`

Defines a common set of properties and methods that are shared with all dispatch source types.

`protocol DispatchSourceFileSystemObject`

A dispatch source that monitors a file descriptor for events defined by `DispatchSource.FileSystemEvent`. The handle is a file descriptor (`Int32`).

`struct DispatchSource.FileSystemEvent`

A file system event.

`protocol DispatchSourceMachSend`

A dispatch source that monitors a Mach port for dead name notifications, indicating that a send right no longer has any corresponding receive right. The handle is a Mach port with a send or send-once right (*mach_port_t*).

`struct DispatchSource.MachSendEvent`

A Mach port send event.

`protocol DispatchSourceMachReceive`

A dispatch source that monitors a Mach port for pending messages. The handle is a Mach port with a receive right (*mach_port_t*).

`protocol DispatchSourceMemoryPressure`

A dispatch source that monitors the system for changes in memory pressure condition.

`struct DispatchSource.MemoryPressureEvent`

A memory pressure event.

`protocol DispatchSourceProcess`

A dispatch source that monitors an external process for events defined by `DispatchSource.ProcessEvent`. The handle is a process identifier (`pid_t`).

`struct DispatchSource.ProcessEvent`

A process event.

`protocol DispatchSourceTimer`

A dispatch source that submits the event handler block based on a timer.

`struct DispatchSource.TimerFlags`

`protocol DispatchSourceUserDataAdd`

A dispatch source that coalesces data obtained via calls to `add(data:)` method.

`protocol DispatchSourceUserDataOr`

A dispatch source that coalesces data obtained via calls to the `or(data:)` method.

Managing Dispatch I/O

The dispatch I/O channel API lets you manage file descriptor–based operations. This API supports both stream-based and random-access semantics for accessing the contents of the file descriptor.

class DispatchIO

DispatchIO provides a channel to perform operations on file descriptor using either stream-based and random-access semantics for accessing the contents of a file descriptor.

enum DispatchIO.StreamType

The semantics for accessing the contents of a file descriptor.

struct DispatchIO.CloseFlags

The semantics of closing an I/O channel.

struct DispatchIO.IntervalFlags

Dispatch I/O Channel Types

The types of dispatch I/O channels that may be created.

Dispatch I/O Channel Closing Options

The options to use when closing a dispatch I/O channel.

Dispatch I/O Channel Configuration Options

The options to use when configuring a channel.

Working with Dispatch Objects

GCD provides dispatch object interfaces to allow your application to manage aspects of processing such as memory management, pausing and resuming execution, defining object context, and logging task data. Dispatch objects must be manually retained and released and are not garbage collected.

class DispatchObject

DispatchObject is the base class for many dispatch types, including DispatchQueue, DispatchGroup, and DispatchSource.

enum DispatchPredicate

DispatchPredicate represents logical conditions that can be evaluated within a given execution context using the dispatchPrecondition(condition:) method.

Dispatch Constants

func dispatchPrecondition(condition: () -> DispatchPredicate)

Checks a dispatch condition necessary for further execution.

func activate()

Activates the dispatch object.

~~func dispatch_debugv(DispatchObject, UnsafePointer<Int8>, CVarListPointer)~~

Deprecated

Enumerations

enum `DispatchQueue.GlobalQueuePriority`

Protocols

protocol `DispatchSourceUserDataReplace`