API Collection

Concurrency

Perform asynchronous and parallel operations.

Topics

Essentials

{} Code-along: Elevating an app with Swift concurrency

Code along with the WWDC presenter to elevate a SwiftUI app with Swift concurrency.

{} Updating an app to use strict concurrency

Use this code to follow along with a guide to migrating your code to take advantage of the full concurrency protection that the Swift 6 language mode provides.

{} Updating an App to Use Swift Concurrency

Improve your app's performance by refactoring your code to take advantage of asynchronous functions in Swift.

Tasks

`struct Task`

A unit of asynchronous work.

`struct TaskGroup`

A group that contains dynamically created child tasks.

```
func withTaskGroup<ChildTaskResult, GroupResult>(of: ChildTaskResult
.Type, returning: GroupResult.Type, isolation: isolated (any Actor)?,
body: (inout TaskGroup<ChildTaskResult>) async -> GroupResult) async ->
GroupResult
```

Starts a new scope that can contain a dynamic number of child tasks.

```
struct ThrowingTaskGroup
```

A group that contains throwing, dynamically created child tasks.

```
func withThrowingTaskGroup<ChildTaskResult, GroupResult>(of: ChildTask
Result.Type, returning: GroupResult.Type, isolation: isolated (any
Actor)?, body: (inout ThrowingTaskGroup<ChildTaskResult, any Error>)
async throws -> GroupResult) async rethrows -> GroupResult
```

Starts a new scope that can contain a dynamic number of throwing child tasks.

```
struct TaskPriority
```

The priority of a task.

```
struct DiscardingTaskGroup
```

A discarding group that contains dynamically created child tasks.

```
func withDiscardingTaskGroup<GroupResult>(returning: GroupResult.Type,
isolation: isolated (any Actor)?, body: (inout DiscardingTaskGroup)
async -> GroupResult) async -> GroupResult
```

Starts a new scope that can contain a dynamic number of child tasks.

```
struct ThrowingDiscardingTaskGroup
```

A throwing discarding group that contains dynamically created child tasks.

```
func withThrowingDiscardingTaskGroup<GroupResult>(returning: Group
Result.Type, isolation: isolated (any Actor)?, body: (inout Throwing
DiscardingTaskGroup<any Error>) async throws -> GroupResult) async
throws -> GroupResult
```

Starts a new scope that can contain a dynamic number of child tasks.

```
struct UnsafeCurrentTask
```

An unsafe reference to the current task.

Asynchronous Sequences

```
protocol AsyncSequence
```

A type that provides asynchronous, sequential, iterated access to its elements.

`struct AsyncStream`

An asynchronous sequence generated from a closure that calls a continuation to produce new elements.

`struct AsyncThrowingStream`

An asynchronous sequence generated from an error-throwing closure that calls a continuation to produce new elements.

Continuations

`struct CheckedContinuation`

A mechanism to interface between synchronous and asynchronous code, logging correctness violations.

`func withCheckedContinuation<T>(isolation: isolated (any Actor)?, function: String, (CheckedContinuation<T, Never>) -> Void) async -> sending T`

Invokes the passed in closure with a checked continuation for the current task.

`func withCheckedThrowingContinuation<T>(isolation: isolated (any Actor)?, function: String, (CheckedContinuation<T, any Error>) -> Void) async throws -> sending T`

Invokes the passed in closure with a checked continuation for the current task.

`struct UnsafeContinuation`

A mechanism to interface between synchronous and asynchronous code, without correctness checking.

`func withUnsafeContinuation<T>(isolation: isolated (any Actor)?, (UnsafeContinuation<T, Never>) -> Void) async -> sending T`

Invokes the passed in closure with a unsafe continuation for the current task.

~~`typealias UnsafeThrowingContinuation`~~ Deprecated

`func withUnsafeThrowingContinuation<T>(isolation: isolated (any Actor)?, (UnsafeContinuation<T, any Error>) -> Void) async throws -> sending T`

Invokes the passed in closure with a unsafe continuation for the current task.

Actors

`protocol Sendable`

A thread-safe type whose values can be shared across arbitrary concurrent contexts without introducing a risk of data races.

`protocol Actor`

Common protocol to which all actors conform.

~~`typealias AnyActor`~~

Common marker protocol providing a shared “base” for both (local) `Actor` and (potentially remote) `DistributedActor` types.

Deprecated

`actor MainActor`

A singleton actor whose executor is equivalent to the main dispatch queue.

`protocol GlobalActor`

A type that represents a globally-unique actor that can be used to isolate various declarations anywhere in the program.

`protocol SendableMetatype`

A type whose metatype can be shared across arbitrary concurrent contexts without introducing a risk of data races. When a generic type T conforms to `SendableMetatype`, its metatype `T.Type` conforms to `Sendable`. All concrete types implicitly conform to the `SendableMetatype` protocol, so its primary purpose is in generic code to prohibit the use of isolated conformances along with the generic type.

~~`typealias ConcurrentValue`~~ Deprecated

~~`protocol UnsafeSendable`~~

A type whose values can safely be passed across concurrency domains by copying, but which disables some safety checking at the conformance site.

Deprecated

~~`typealias UnsafeConcurrentValue`~~ Deprecated

`macro isolation<T>() -> T`

Produce a reference to the actor to which the enclosing code is isolated, or `nil` if the code is nonisolated.

~~`func extractIsolation<each Arg, Result>((repeat each Arg) async throws -> Result) -> (any Actor)?`~~

Deprecated

Task-Local Storage

`class TaskLocal`

Wrapper type that defines a task-local value key.

`macro TaskLocal()`

Macro that introduces a `TaskLocal` binding.

Executors

`protocol Executor`

A service that can execute jobs.

`struct ExecutorJob`

A unit of schedulable work.

`protocol SerialExecutor`

A service that executes jobs.

`protocol TaskExecutor`

An executor that may be used as preferred executor by a task.

~~`typealias PartialAsyncTask`~~ Deprecated

`struct UnownedJob`

A unit of schedulable work.

`struct JobPriority`

The priority of this job.

`struct UnownedSerialExecutor`

An unowned reference to a serial executor (a `SerialExecutor` value).

`struct UnownedTaskExecutor`

`var globalConcurrentExecutor: any TaskExecutor`

The global concurrent executor that is used by default for Swift Concurrency tasks.

```
func withTaskExecutorPreference<T, Failure>((any TaskExecutor)?,
isolation: isolated (any Actor)?, operation: () async throws(Failure) -
> T) async throws(Failure) -> T
```

Configure the current task hierarchy's task executor preference to the passed `TaskExecutor`, and execute the passed in closure by immediately hopping to that executor.

Deprecated

~~`struct`~~ ~~`Job`~~

Deprecated equivalent of `ExecutorJob`.

Deprecated

See Also

Programming Tasks

Input and Output

Print values to the console, read from and write to text streams, and use command line arguments.

Debugging and Reflection

Fortify your code with runtime checks, and examine your values' runtime representation.

Macros

Generate boilerplate code and perform other compile-time operations.

Key-Path Expressions

Use key-path expressions to access properties dynamically.

Manual Memory Management

Allocate and manage memory manually.

Type Casting and Existential Types

Perform casts between types or represent values of any type.

C Interoperability

Use imported C types or call C variadic functions.

Operator Declarations

Work with prefix, postfix, and infix operators.