Swift / DiscardingTaskGroup

Structure

DiscardingTaskGroup

A discarding group that contains dynamically created child tasks.

iOS 17.0+ | iPadOS 17.0+ | Mac Catalyst 17.0+ | macOS 14.0+ | tvOS 17.0+ | visionOS 1.0+ | watchOS 10.0+

```
@frozen
struct DiscardingTaskGroup
```

Overview

To create a discarding task group, call the `withDiscardingTaskGroup(returning:body:)` method.

Don't use a task group from outside the task where you created it. In most cases, the Swift type system prevents a task group from escaping like that because adding a child task to a task group is a mutating operation, and mutation operations can't be performed from a concurrent execution context like a child task.

Refer to `TaskGroup` documentation for detailed discussion of semantics shared between all task groups.

Discarding behavior

A discarding task group eagerly discards and releases its child tasks as soon as they complete. This allows for the efficient releasing of memory used by those tasks, which are not retained for future `next()` calls, as would be the case with a `TaskGroup`.

Cancellation behavior

A discarding task group becomes canceled in one of the following ways:

- when `cancelAll()` is invoked on it,
- when the `Task` running this task group is canceled.

Since a `DiscardingTaskGroup` is a structured concurrency primitive, cancellation is automatically propagated through all of its child-tasks (and their child tasks).

A canceled task group can still keep adding tasks, however they will start being immediately canceled, and may act accordingly to this. To avoid adding new tasks to an already canceled task group, use `addTaskUnlessCancelled(priority:body:)` rather than the plain `addTask(priority:body:)` which adds tasks unconditionally.

For information about the language-level concurrency model that `DiscardingTaskGroup` is part of, see Concurrency in The Swift Programming Language.

See Also

`TaskGroup`

See Also

`ThrowingTaskGroup`

See Also

`ThrowingDiscardingTaskGroup`

Topics

Instance Properties

`var isCancelled: Bool`

A Boolean value that indicates whether the group was canceled.

`var isEmpty: Bool`

A Boolean value that indicates whether the group has any remaining tasks.

Instance Methods

```
func addImmediateTask(name: String?, priority: TaskPriority?, executorPreference: consuming (any TaskExecutor)?, operation: sending () async -> Void)
```

Add a child task to the group and immediately start running it in the context of the calling thread/task.

```
func addImmediateTaskUnlessCancelled(name: String?, priority: TaskPriority?, executorPreference: consuming (any TaskExecutor)?, operation: sending () async -> Void) -> Bool
```

Add a child task to the group and immediately start running it in the context of the calling thread/task.

```
func addTask(executorPreference: (any TaskExecutor)?, priority: TaskPriority?, operation: sending () async -> Void)
```

Adds a child task to the group.

```
func addTask(name: String?, executorPreference: (any TaskExecutor)?, priority: TaskPriority?, operation: sending () async -> Void)
```

Adds a child task to the group.

```
func addTask(name: String?, priority: TaskPriority?, operation: sending () async -> Void)
```

Adds a child task to the group.

```
func addTask(priority: TaskPriority?, operation: sending () async -> Void)
```

Adds a child task to the group.

```
func addTaskUnlessCancelled(executorPreference: (any TaskExecutor)?, priority: TaskPriority?, operation: sending () async -> Void) -> Bool
```

Adds a child task to the group, unless the group has been canceled. Returns a boolean value indicating if the task was successfully added to the group or not.

```
func addTaskUnlessCancelled(name: String?, executorPreference: (any TaskExecutor)?, priority: TaskPriority?, operation: sending () async -> Void) -> Bool
```

Adds a child task to the group, unless the group has been canceled. Returns a boolean value indicating if the task was successfully added to the group or not.

```
func addTaskUnlessCancelled(name: String?, priority: TaskPriority?, operation: sending () async -> Void) -> Bool
```

Adds a child task to the group, unless the group has been canceled. Returns a boolean value indicating if the task was successfully added to the group or not.

```
func addTaskUnlessCancelled(priority: TaskPriority?, operation: sending
() async -> Void) -> Bool
```

Adds a child task to the group, unless the group has been canceled. Returns a boolean value indicating if the task was successfully added to the group or not.

```
func cancelAll()
```

Cancel all of the remaining tasks in the group.

Relationships

Conforms To

`BitwiseCopyable`, `Copyable`

See Also

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.

```
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.