Class

NSFetchedResultsController

A controller that you use to manage the results of a Core Data fetch request and to display data to the user.

iOS 3.0+ | iPadOS 3.0+ | Mac Catalyst 13.1+ | macOS 10.12+ | tvOS | visionOS 1.0+ | watchOS 2.0+

```
class NSFetchedResultsController<ResultType> where ResultType : NSFetchRequest
Result
```

Overview

While you can use table views can in several ways, fetched results controllers primarily assist you with a primary list view. `UITableView` expects its data source to provide cells as an array of sections made up of rows. You configure a fetched results controller using a *fetch request* — an object that specifies what type of entity to fetch and how to sort the results. You can also add criteria for when to include a specific instance of the entity.

The fetched results controller efficiently analyzes the result of the fetch request and computes all the information about sections in the result set. It also computes all the information for the index based on the result set.

In addition, fetched results controllers:

- Optionally monitor changes to objects in the associated managed object context, and report changes in the results set to its delegate (see The controller's delegate).
- Optionally cache the results of its computation to enable redisplaying the same data without repeating the work to fetch it. For more information, see The cache.

A controller thus effectively has three modes of operation, determined by whether it has a delegate and whether you set the cache file name.

- No tracking: The delegate is `nil`. The controller provides access to the data as it was when it fetched it.
- Memory-only tracking: the delegate is non-`nil` and the file cache name is `nil`. The controller monitors objects in its result set and updates section and ordering information in response to relevant changes.
- Full persistent tracking: the delegate and the file cache name are non-`nil`. The controller monitors objects in its result set and updates section and ordering information in response to relevant changes. The controller maintains a persistent cache of the results of its computation.

Important

A delegate must implement at least one of the change tracking delegate methods to enable change tracking. Providing an empty implementation of `controllerDidChangeContent(_:)` is sufficient.

Using NSFetchedResultsController

Creating the fetched results controller

You typically create an instance of `NSFetchedResultsController` as an instance variable of a table view controller. When you initialize the fetch results controller, you provide four parameters:

- A fetch request. This must contain at least one sort descriptor to order the results.
- A managed object context. The controller uses this context to execute the fetch request.
- Optionally, a key path on result objects that returns the section name. The controller uses the key path to split the results into sections (passing `nil` indicates that the controller should generate a single section).
- Optionally, the name of the cache file the controller should use (passing `nil` prevents caching). Using a cache can avoid the overhead of computing the section and index information.

After creating an instance, you invoke `performFetch()` to actually execute the fetch:

```
// Get the managed object context from the persistent container.
let context = coreDataStack.persistentContainer.viewContext

// Create a fetch request and sort descriptor for the entity to display
// in the table view.
let fetchRequest: NSFetchRequest<Item> = Item.fetchRequest()
let sortDescriptor = NSSortDescriptor(key: "name", ascending: true)
```

```swift
fetchRequest.sortDescriptors = [sortDescriptor]

// Initialize the fetched results controller with the fetch request and
// managed object context.
fetchedResultsController = NSFetchedResultsController(
    fetchRequest: fetchRequest,
    managedObjectContext: context,
    sectionNameKeyPath: nil,
    cacheName: nil)

// Set the controller's delegate.
fetchedResultsController?.delegate = self

// Perform a fetch.
do {
    try fetchedResultsController?.performFetch()
} catch {
    // Handle error appropriately. It's useful to use
    // `fatalError(_:file:line:)` during development.
    fatalError("Failed to perform fetch: \(error.localizedDescription)")
```

Important

If you use a cache, call `deleteCache(withName:)` before changing any of the fetch request, its predicate, or its sort descriptors. Don't reuse the same fetched results controller for multiple queries unless you set the `cacheName` to `nil`.

The controller's delegate

If you set a delegate for a fetched results controller, the controller registers to receive change notifications from its managed object context. The controller processes any change in the context that affects the result set or section information and updates the results as necessary. The controller notifies the delegate when result objects change location or when changes occur in sections. For more information, see `NSFetchedResultsControllerDelegate`. You typically use these methods to update the display of the table view.

The cache

Where possible, a controller uses a cache to avoid the need to repeat work performed in setting up any sections and ordering the contents. The system maintains the cache across launches of your application.

When you initialize an instance of `NSFetchedResultsController`, you typically specify a cache name. If you don't specify a cache name, the controller doesn't cache data. When you create a controller, it looks for an existing cache with the given name:

- If the controller can't find an appropriate cache, it calculates the required sections and the order of objects within sections. It then writes this information to disk.
- If it finds a cache with the same name, the controller tests the cache to determine whether its contents are still valid. The controller compares the current entity name, entity version hash, sort descriptors, and section key-path with those stored in the cache, as well as the modification date of the cached information file and the persistent store file.

If the cache is consistent with the current information, the controller reuses the previously-computed information.

If the cache isn't consistent with the current information, then the controller recomputes the required information and updates the cache.

Any time the section and ordering information change, the controller updates cache.

If you create multiple fetched results controllers with different configurations, such as different sort descriptors, give each configuration a different cache name.

You can purge a cache using `deleteCache(withName:)`.

Implementing the table view datasource methods

You ask the object to provide relevant information in your implementation of the table view data source methods:

```
// Get the number of sections in the table view from the fetched results
// controller.
override func numberOfSections(in tableView: UITableView) -> Int {
    fetchedResultsController?.sections?.count ?? 0
}

// Get the number of rows in each section of the table view from the fetched results
override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int
    guard let sectionInfo = fetchedResultsController?.sections?[section] else {
        return 0
    }

    return sectionInfo.numberOfObjects
}
```

```
// Get table view cells for index paths from the fetched results controller.
override func tableView(_ tableView: UITableView,
                        cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "ItemCell", for: indexP
    let item = fetchedResultsController?.object(at: indexPath)
    cell.textLabel?.text = item?.name
    return cell
}

// Get the title of the header for the specified table view section from the
// fetched results controller.
override func tableView(_ tableView: UITableView, titleForHeaderInSection section: I
    guard let sectionInfo = fetchedResultsController?.sections?[section] else {
        return nil
    }

    return sectionInfo.name
}

// Get the section index titles from the fetched results controller.
override func sectionIndexTitles(for tableView: UITableView) -> [String]? {
    fetchedResultsController?.sectionIndexTitles
}

// Get the section for the specified index title from the fetched
// results controller.
override func tableView(_ tableView: UITableView,
                        sectionForSectionIndexTitle title: String,
                        at index: Int) -> Int {
    guard let result = fetchedResultsController?.section(forSectionIndexTitle: title
                                                                 at: index) else {
        fatalError("Failed to locate section for \(title) at index \(index)")
    }

    return result
```

Responding to changes

`NSFetchedResultsController` responds to changes at the model layer, and informs its delegate when result objects change location or when sections change.

If you allow a user to reorder table rows, then your implementation of the delegate methods must take this into account; see `NSFetchedResultsControllerDelegate`.

The controller doesn't show changes until after its managed object context receives a processPendingChanges() message. Therefore, if you change the value of a managed object's attribute so that its location in a fetched results controller's results set changes, its index as reported by the controller won't typically change until the end of the current event cycle, when the system calls processPendingChanges(). For example, the following code would log "same":

```
// Obtain a managed object from the fetched objects.
guard let managedObject = fetchedResultsController?.object(at: IndexPath(index: 0)),
      // Get the object's index path before changing it.
      let beforeIndexPath = fetchedResultsController?.indexPath(forObject: managedOb
else {
    return
}

// Change the object.
managedObject.setValue("My Item", forKey: "name")

// Get the object's index path after changing it.
if let afterIndexPath = fetchedResultsController?.indexPath(forObject: managedObject
   // Compare the index paths before and after the change.
   beforeIndexPath.compare(afterIndexPath) == .orderedSame {
    print("Same")
}
```

Modifying the fetch request

You can't change the fetch request to modify the results. Do the following if you want to change the fetch request:

1. Delete the cache if you're using one, by calling deleteCache(withName:).

2. Change the fetch request.

3. Call performFetch().

> **Note**
>
> Don't use a cache if you're changing the fetch request.

Handling object invalidation

When a managed object context notifies the fetched results controller of invalidated *individual* objects, the controller treats these as deleted objects and sends the proper delegate calls.

Simultaneous invalidation of *all* the objects in a managed object context is possible, for example, as a result of calling `reset()`, or if you remove a store from the persistent store coordinator. When this happens, `NSFetchedResultsController` doesn't invalidate all objects, nor does it send individual notifications for object deletions. Instead, you need to call `performFetch()` to reset the state of the controller then reload the data in the table view (`reloadData()`).

Subclassing notes

You create a subclass of this class if you want to customize the creation of sections and index titles. You override `sectionIndexTitle(forSectionName:)` if you want the section index title to be something other than the capitalized first letter of the section name. You override `sectionIndexTitles` if you want the index titles to be something other than the array created by calling `sectionIndexTitle(forSectionName:)` on all the known sections.

Topics

Initializing a Fetched Results Controller

```
init(fetchRequest: NSFetchRequest<ResultType>, managedObjectContext: NSManagedObjectContext, sectionNameKeyPath: String?, cacheName: String?)
```

Returns a fetch request controller initialized using the given arguments.

```
func performFetch() throws
```

Executes the controller's fetch request.

Getting Configuration Information

```
var fetchRequest: NSFetchRequest<ResultType>
```

The fetch request used to do the fetching.

```
var managedObjectContext: NSManagedObjectContext
```

The managed object context used to fetch objects.

```
var sectionNameKeyPath: String?
```

The key path of the attribute that determines which section the fetched entity belongs to.

```
var cacheName: String?
```

The name of the file used to cache section information.

```
var delegate: (any NSFetchedResultsControllerDelegate)?
```

The object that is notified when the fetched results changed.

```
class func deleteCache(withName: String?)
```

Deletes the cached section information with the given name.

Accessing Results

```
var fetchedObjects: [ResultType]?
```

The results of the fetch.

```
func object(at: IndexPath) -> ResultType
```

Returns the object at the given index path in the fetch results.

```
func indexPath(forObject: ResultType) -> IndexPath?
```

Returns the index path of a given object.

Querying Section Information

```
var sections: [any NSFetchedResultsSectionInfo]?
```

The sections for the fetch results.

```
func section(forSectionIndexTitle: String, at: Int) -> Int
```

Returns the section number for a given section title and index in the section index.

Configuring Section Information

```
func sectionIndexTitle(forSectionName: String) -> String?
```

Returns the corresponding section index entry for a given section name.

```
var sectionIndexTitles: [String]
```

The array of section index titles.

Responding to Changes

```
protocol NSFetchedResultsControllerDelegate
```

A delegate protocol that describes the methods that the associated fetched results controller calls when the fetch results change.

`protocol NSFetchedResultsSectionInfo`

A protocol that defines the interface for section objects vended by a fetched results controller.

`struct NSFetchRequestResultType`

Constants that specify the possible result types a fetch request can return.

`enum NSFetchedResultsChangeType`

Constants that specify the possible types of changes that are reported.

Relationships

Inherits From

`NSObject`

Conforms To

`CVarArg`
`CustomDebugStringConvertible`
`CustomStringConvertible`
`Equatable`
`Hashable`
`NSObjectProtocol`

See Also

Fetch requests

`class NSFetchRequest`

A description of search criteria used to retrieve data from a persistent store.

`class NSAsynchronousFetchRequest`

A fetch request that retrieves results asynchronously and supports progress notification.

`class NSAsynchronousFetchResult`

A fetch result object that encompasses the response from an executed asynchronous fetch request.