**Threading and GCD**

Quản lý luồng (**thread**) trong **Swift** là một phần quan trọng của lập trình đồng thời (**concurrency**), giúp bạn xử lý nhiều tác vụ cùng một lúc mà không làm chậm ứng dụng của bạn. Dưới đây là một số khái niệm và kỹ thuật quan trọng để làm việc với thread trong Swift.

Nội dung bài viết.

* Khái niệm về thread
  + Tìm hiểu về Grand Central Dispathch(GCD) là gì?
    - Tìm hiểu về DispatchGroup
  + Tìm hiểu về Operation/OperationQueue.
  + So sánh tự khác và giống nhau giữa Operation/OperationQueue và DispatchQueue

1. **Khái niệm về thread**

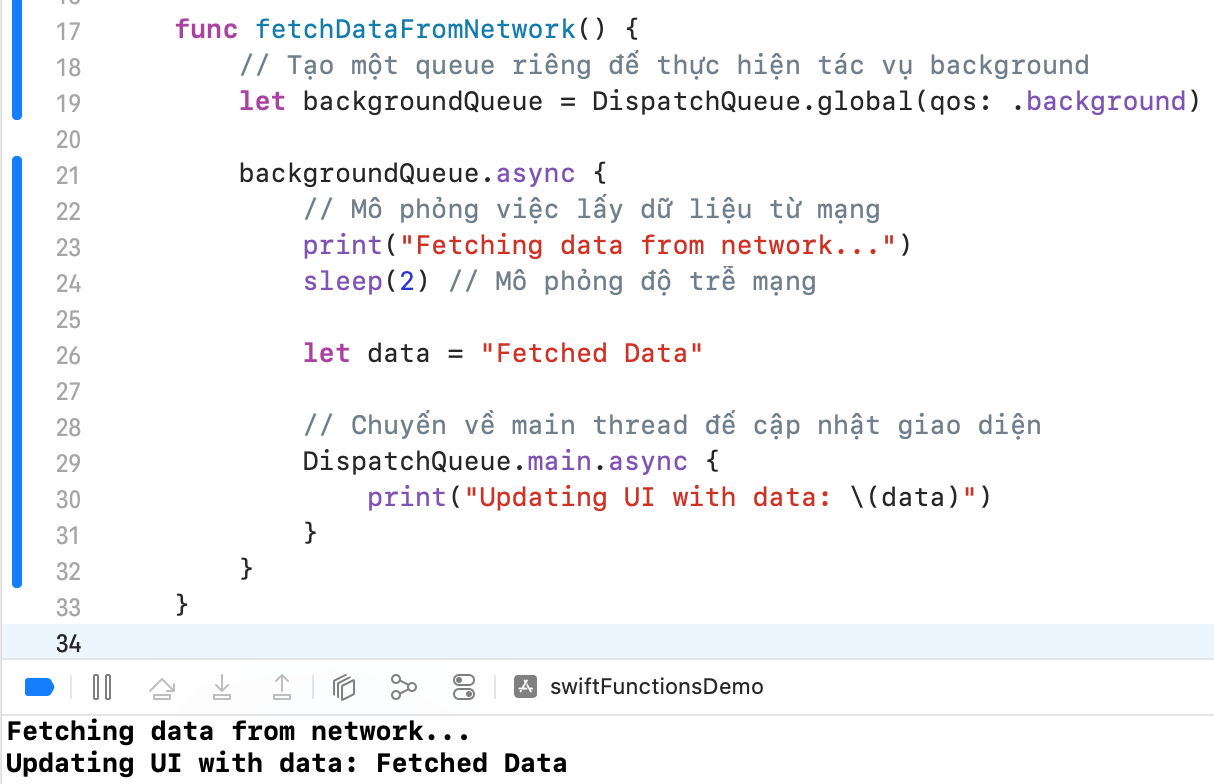
Trước hết chúng ta cùng tìm hiểu một số định nghĩa về **thread**Trong lập trình, một thread (luồng) là một đơn vị cơ bản của việc thực thi, cho phép chương trình thực hiện nhiều tác vụ đồng thời. Trong Swift, quản lý và sử dụng thread có thể giúp ứng dụng thực hiện nhiều công việc song song, tăng hiệu suất và cải thiện trải nghiệm người dùng.

Swift cung cấp nhiều cách để làm việc với thread, trong đó phổ biến nhất là sử dụng Grand Central Dispatch (GCD) và OperationQueue.

Ví dụ:

1. **Sử dụng Grand Central Dispatch (GCD)**

**Grand Central Dispatch** (**GCD**) là một thư viện mạnh mẽ để quản lý **thread** và thực thi các tác vụ không đồng bộ. Dưới đây là một ví dụ về cách sử dụng **GCD** để thực hiện một công việc trong background và cập nhật giao diện trên **main** **thread**.

****

Trên ảnh thể hiện đoạn mã và kết quả khi dử dụng GCD

Trong ví dụ này:

- **DispatchQueue.global(qos: .background)** tạo một queue background để thực hiện các tác vụ không đồng bộ.

- **async** cho phép chúng ta thực hiện công việc trong thread nền.

- **DispatchQueue.main.async** đảm bảo rằng việc cập nhật giao diện được thực hiện trên main thread.

1. **Sử dụng OperationQueue**

**OperationQueue** cung cấp một cách cao cấp hơn để quản lý và sắp xếp các công việc. Bạn có thể sử dụng **OperationQueue** để thực thi các operation một cách đồng bộ hoặc không đồng bộ.



Trong ví dụ này:

- **OperationQueue** được sử dụng để quản lý các operation.

- **BlockOperation** là một loại operation thực hiện một hoặc nhiều block.

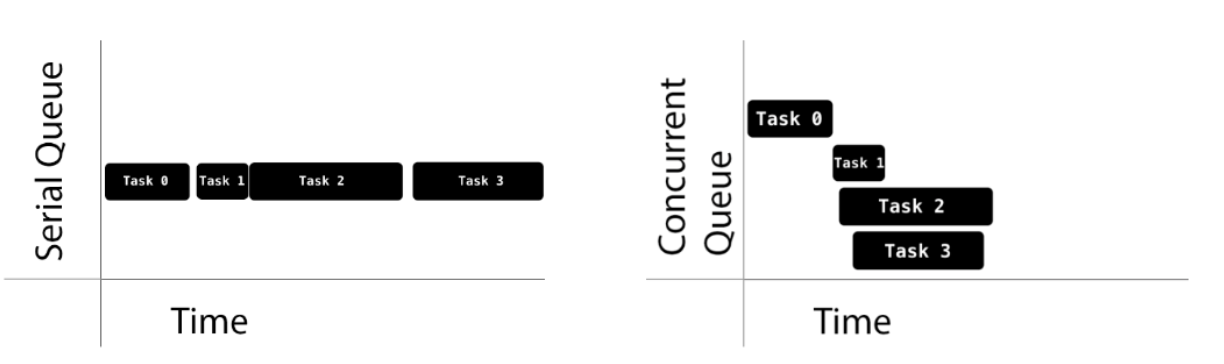
- **addDependency** thiết lập mối quan hệ phụ thuộc giữa các operation, đảm bảo **operation2** chỉ bắt đầu sau khi **operation1** hoàn thành.

**(\*)** **Grand Central Dispathch(GCD)**

- **Grand Central Dispatch** (**GCD**) là một thư viện mạnh mẽ trong hệ điều hành macOS và iOS, giúp quản lý đồng thời các tác vụ một cách hiệu quả và dễ dàng. **GCD** sử dụng các hàng đợi (**queues**) để quản lý các công việc (**tasks**) và phân phối chúng tới các thread có sẵn.

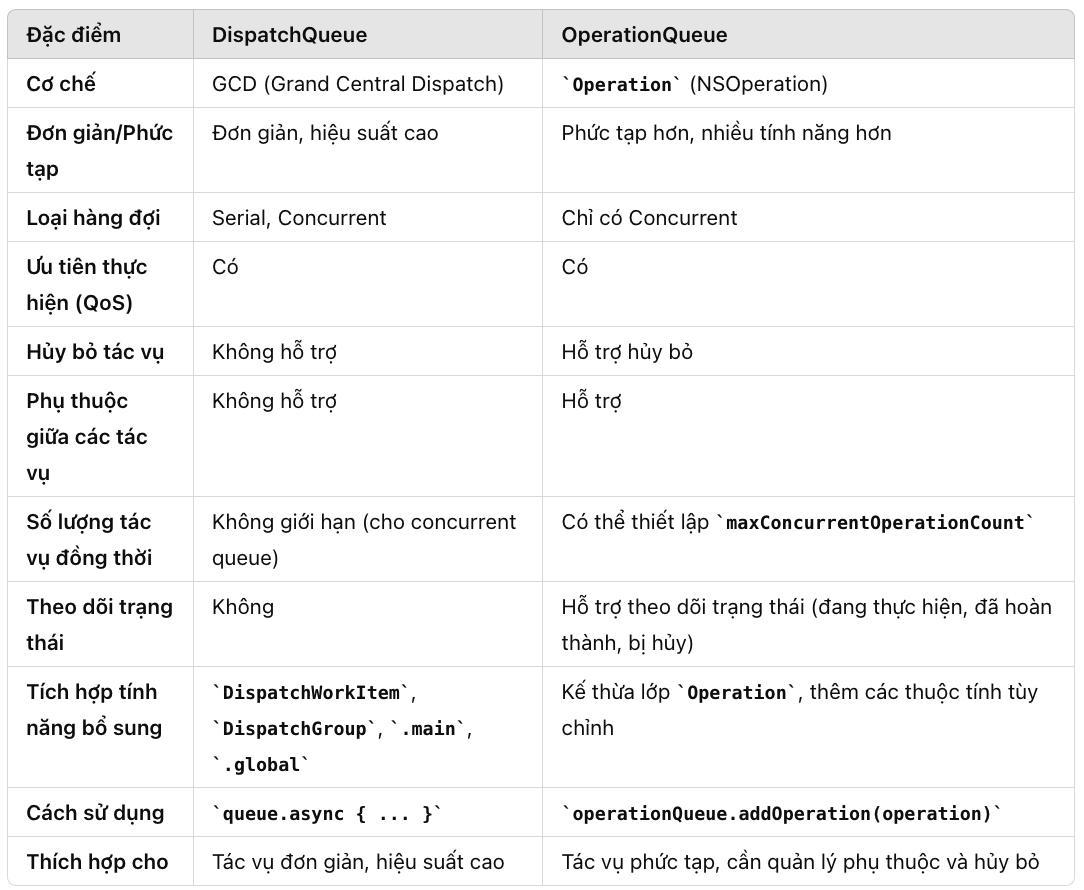
Các Khái Niệm Cơ Bản:

* **Dispatch Queue**: Là các hàng đợi quản lý thứ tự và thời điểm thực thi các block (mã) được đặt vào. Có hai loại chính:
  + **Serial Queue**: Chỉ thực hiện một task tại một thời điểm. Các task sẽ được thực hiện tuần tự.
  + **Concurrent Queue**: Có thể thực hiện nhiều task đồng thời, không đảm bảo thứ tự thực thi.



Hình ảnh minh họa thứ tự thực hiện task

Bảng tóm tắt sự khác nhau và giống nhau giữa **DispatchQueue** và **OperationQueue**:



**Khi nào sử dụng**

* **DispatchQueue**: Thích hợp cho các tác vụ đơn giản, không yêu cầu quản lý phức tạp.
* **OperationQueue**: Thích hợp cho các tác vụ phức tạp, yêu cầu quản lý phụ thuộc và hủy bỏ.

Bảng này tóm tắt các điểm chính giúp bạn dễ dàng so sánh và lựa chọn giữa **DispatchQueue** và **OperationQueue** trong phát triển ứng dụng Swift.

* **Main Queue**: Là một serial queue đặc biệt, thực hiện các task trên main thread. Đây là nơi bạn thực hiện các cập nhật giao diện.
* **Global Queue**: Là các concurrent queue toàn cục, có sẵn trong hệ thống, với các mức độ ưu tiên khác nhau.

VD về việc sử dụng Main và Global Queue

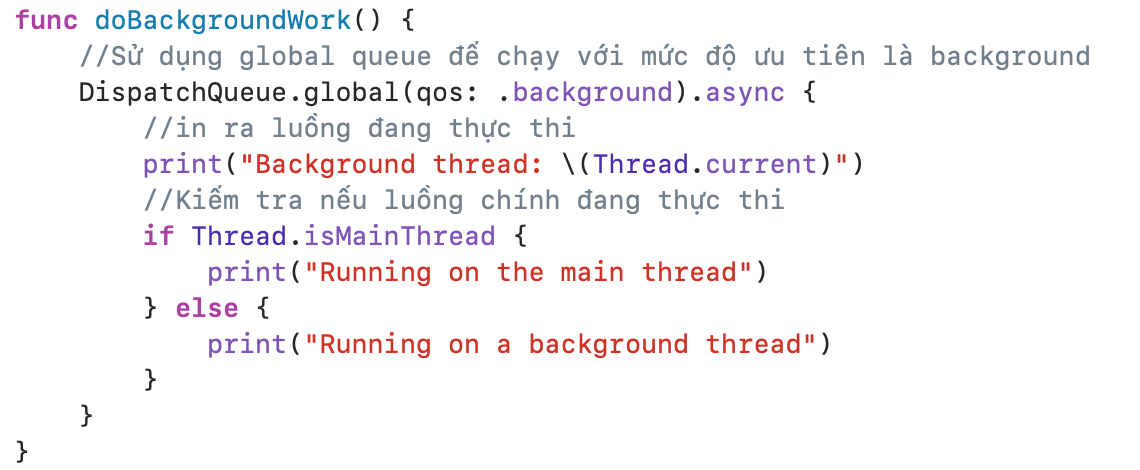
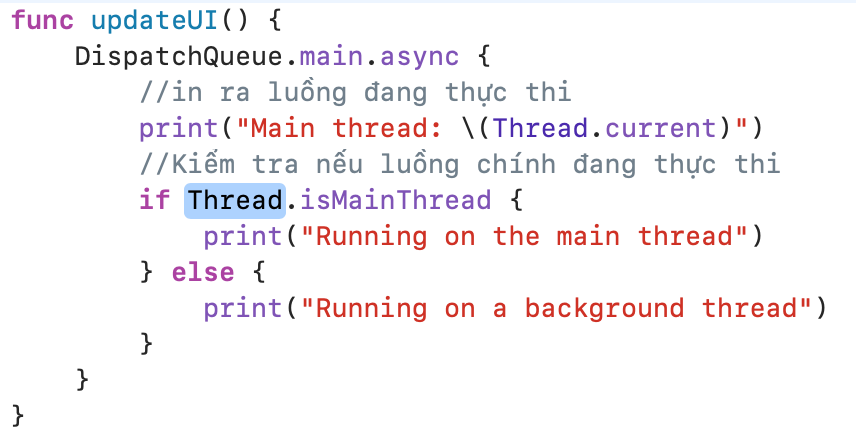
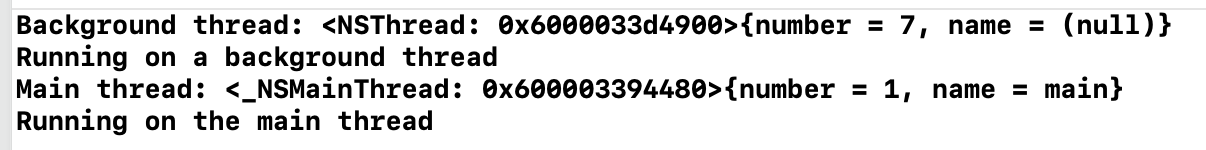
**** ****

Image 1 Global và Main queue

**Sau khi thực hiện và chạy đoạn mã trên sẽ được kết quả sau.**

****

* **Synchronous (sync) và Asynchronous (async)**:
  + **sync**: Chờ cho đến khi task hoàn thành trước khi tiếp tục thực thi mã.
  + **async**: Không chờ task hoàn thành, tiếp tục thực thi mã ngay lập tức.

VD về **Sync:**

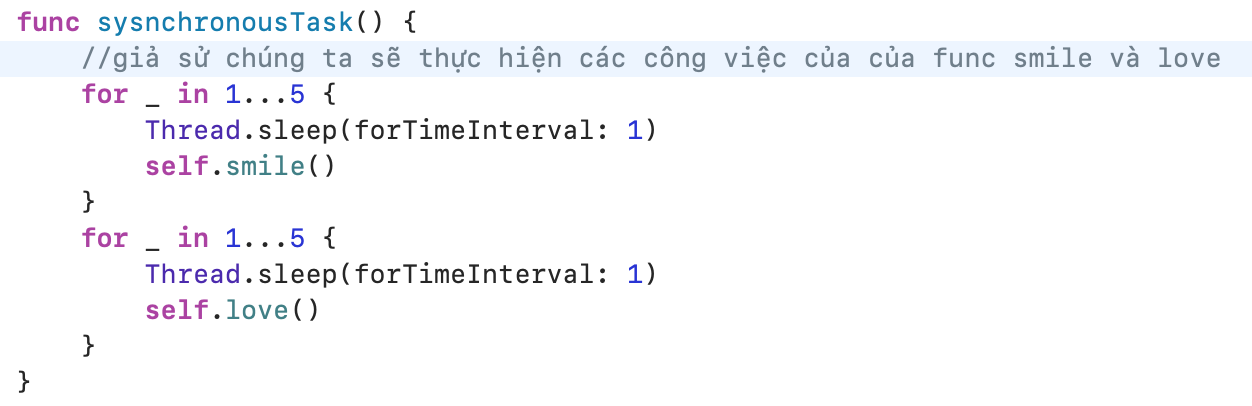
 

Image 2 VD các tác vụ thực hiện theo **Sync** và kết quả

VD về **Async:**

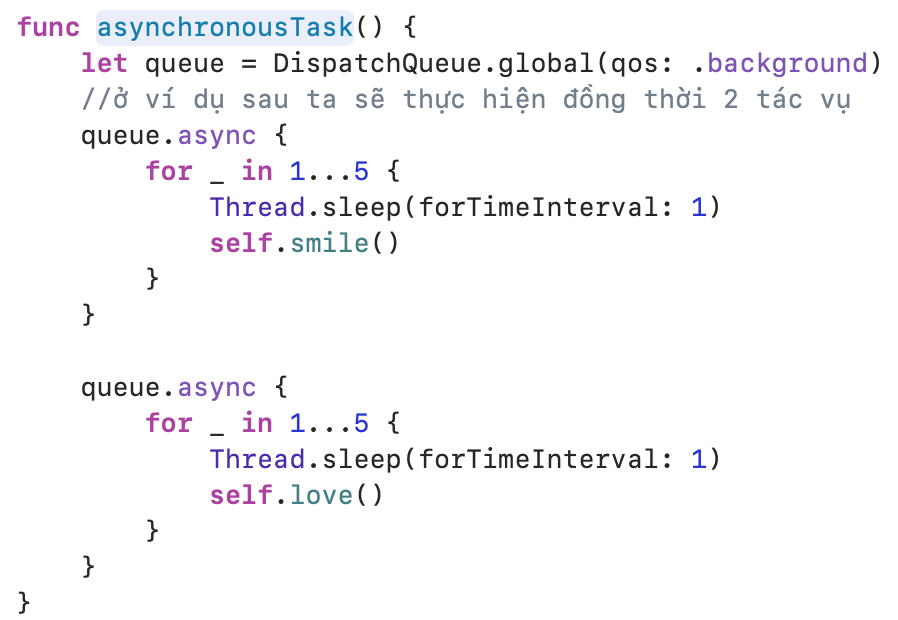
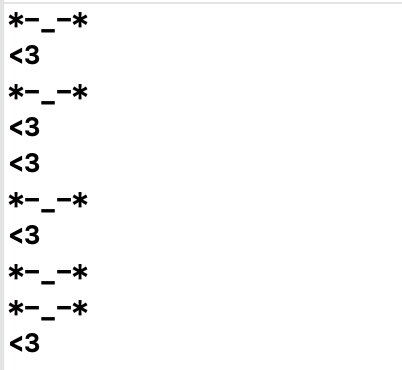
 

Image 3 VD các tác vụ thực hiện theo **Async** và kết quả

Ở ví dụ sau đã thấy rõ sự khác biệt khi kết quả thực hiện sẽ chạy đồng thời 2 tác vụ cùng một lúc và đồng thời in ra kết quả của smile() và love(), có sự khác biệt so với Syns chỉ thực hiện các xong tác vụ smile() rồi mới tiếp tục thực hiện tác vụ love().

**Qos** là một cách thức thể hiện độ ưu tiên trong **global** **queue**, giúp chúng ta setup một global concurrent queue.

**Có 6 loại** **Qos**

|  |  |
| --- | --- |
| Thứ tự từ cao đến thấp | **Chức năng** |
| **userInteractive** | Mức cao nhất, dùng cho các công việc tương tác trực tiếp với người dùng và cần kết quả ngay lập tức, ví dụ như cập nhật giao diện người dùng. Các công việc ở mức này được thực thi với ưu tiên cao nhất. |
| **userInitiated** | Dùng cho các công việc mà người dùng khởi tạo và mong đợi kết quả trong thời gian ngắn. Ví dụ như mở một tài liệu hay thực hiện một thao tác yêu cầu dữ liệu ngay lập tức. |
| **default** | Mức QoS mặc định cho các công việc không có yêu cầu đặc biệt về hiệu suất. Nếu không xác định QoS, hệ thống sẽ sử dụng mức này. |
| **utility** | Dùng cho các công việc có thời gian thực thi dài, không cần kết quả ngay lập tức, nhưng vẫn quan trọng. Ví dụ như tải về dữ liệu nền hay xử lý các tập tin lớn. |
| **background** | Mức thấp nhất, dùng cho các công việc mà người dùng không cần biết đến và có thể thực hiện trong nền, ví dụ như đồng bộ dữ liệu hay lưu trữ bộ nhớ cache. |
| **unspecified** | Không xác định mức QoS. Đây là mức thấp nhất và không được khuyến nghị sử dụng vì hệ thống không thể tối ưu hóa tài nguyên. |

VD: thể hiện độ ưu tiên khi thực hiện

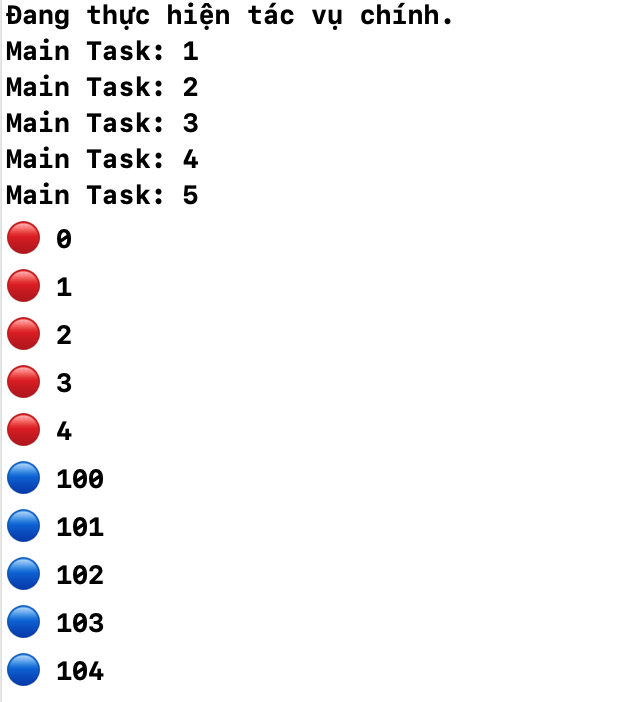
 

Image 4 Độ ưu tiên giữa **userInitiated** và **background**

**(\*) DispatchGroup**

- Một giải pháp giúp chúng ta có thể thực hiện các tác vụ song song Sau khi tất cả các tác vụ có trong dispatchGroup hoàn thành sẽ tiếp tục tại luồng chính.

Để thêm hàng đợi vào dispatchGroup:

cách 1: Bạn có thể thêm công việc vào DispatchGroup bằng cách sử dụng các phương thức bằng cách sử dụng các phương thức async(group:)

cách 2: cách sử dụng các phương thức **async**(group:)



Chờ đợi các công việc hoàn thành Có hai cách chính để chờ đợi tất cả các công việc trong DispatchGroup hoàn thành: **wait**() và **notify**(queue:).



Lợi ích của DispatchGroup

- **Đồng bộ hóa công việc bất đồng bộ**: **DispatchGroup** giúp bạn dễ dàng quản lý nhiều công việc bất đồng bộ và đảm bảo rằng bạn có thể thực hiện một hành động nào đó sau khi tất cả các công việc đó hoàn thành.

- **Tối ưu hóa hiệu suất**: Bằng cách thực hiện các công việc song song, bạn có thể tận dụng tối đa tài nguyên hệ thống và cải thiện hiệu suất ứng dụng của mình.

- **Dễ sử dụng**: **DispatchGroup** đơn giản và dễ sử dụng với các phương thức trực quan.