**Async and Awati**

|  |
| --- |
|  **async** chỉ đơn giản là một từ khóa giúp chỉ ra rằng phương thức sẽ thực hiện công việc bất đồng bộ.  (khi 1 phương thức được khái báo với async thì không cần start vì khi gọi phương thức thì sẽ tự động start )  (chỉ start khi khai báo Task thủ công, và nó sẽ chạy song song với Thread chính)   **await** được sử dụng để chờ đợi kết quả từ các tác vụ bất đồng bộ mà không làm chặn dòng xử lý chính. |

|  |
| --- |
| **Chú ý :**  Khi bạn sử dụng **Wait()** hoặc **Result** trong **bất kỳ phương thức nào**, kể cả trong phương thức bất đồng bộ hay phương thức Main, thì **luồng gọi tới** (thường là luồng chính) **sẽ bị chặn** cho đến khi tác vụ hoàn thành. |

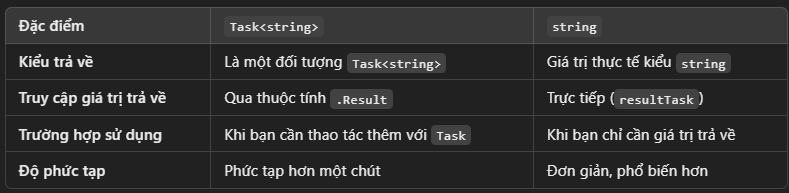
|  |  |
| --- | --- |
| Loại 1 : Task (). | + Đại diện cho một tác vụ bất đồng bộ **không trả về kết quả**.  + Thường được sử dụng với các phương thức chỉ  thực hiện hành động như ghi file, gửi dữ liệu mạng, v.v. |

|  |  |
| --- | --- |
| Loại 2 : Task<TResult> |  Đại diện cho một tác vụ bất đồng bộ **có trả về kết quả**.   Kết quả được xác định bằng kiểu TResult. |

|  |  |
| --- | --- |
| Loại 3 : **ValueTask** (Non-Generic và Generic)    Ví dụ (Generic):    ***Hàm Main gọi tải thử ;*** |  Là một biến thể nhẹ hơn của Task và thường được sử  dụng để tối ưu hiệu suất khi một tác vụ bất đồng bộ  **có thể hoàn thành ngay lập tức**.   Dùng trong trường hợp hiệu năng quan trọng và  tránh tạo đối tượng không cần thiết.  --------------------------------------------------------------------------  **Tạo Task một cách thủ công**    **Task.Run** sẽ tạo một tác vụ mới và chạy nó trên **Thread Pool**,  **Điểm mạnh:**   * **Kiểm soát cao**: Bạn hoàn toàn kiểm soát được thời   điểm Task bắt đầu (Start()).   * **Tùy biến linh hoạt**: Có thể sử dụng các tham số   hoặc logic đặc biệt trong việc khởi tạo.  **Điểm yếu:**   * **Không tiện lợi**: Phải tự quản lý việc khởi động Task. * **Dễ nhầm lẫn**: Nếu quên gọi Start()   Task sẽ không chạy, và điều này có thể gây lỗi logic. |

**Có 2 kiểu giá trị khi gọi phương thức bất đồng bộ TASK <T>**

|  |  |
| --- | --- |
| Task<string> resultTask = FetchDataAsync();  Console.WriteLine(resultTask.Result); // Truy cập giá trị  *+ kiểu trả về đối tượng kiểu Task*   **Giải thích**:   * Khi sử dụng Task<string>, bạn vẫn nhận được đối tượng Task<string>, nhưng vì đã dùng await, Task đã hoàn thành. * Bạn có thể thao tác với Task để lấy thêm thông tin như trạng thái (IsCompleted, IsFaulted, ...) hoặc gắn tiếp các tác vụ khác (dùng .ContinueWith()).    **Kết quả**:   * **resultTask** là một đối tượng **Task<string>**, không phải giá trị thực tế (string).   🡺 khi trả về kiểu Task thì không thể sử dụng **awati** vì awati chỉ trả về giá trị thức tế mà k phải đối tượng | String resultTask = await FetchDataAsync();  Console.WriteLine(resultTask); // In trực tiếp giá trị  + *kiểu trả về kiểu nguyên thủy (String , int ,double…)*   **Giải thích**:   * Khi bạn sử dụng string, await sẽ **giải nén kết quả từ Task<string>**. Nghĩa là bạn trực tiếp nhận được giá trị trả về (kiểu string). * Cách này phổ biến hơn vì bạn thường chỉ cần giá trị trả về, không cần xử lý Task.    **Kết quả**:   * **resultTask** là giá trị thực tế (string), không phải một Task. |



|  |
| --- |
| **Cách thức hoạt động của Thread trong lập trình bất đồng bộ :**  + Nếu bạn sử dụng **Task.Run()**tác vụ sẽ được chạy trong **một luồng riêng biệt** từ ***Thread pool.*** Mỗi lần bạn gọi **Task.Run()**, một luồng mới trong thread pool sẽ được lấy ra để thực thi tác vụ đó. ( ***khi sử dụng Task.Run() có nghĩa là bạn đang yêu cầu 1 Thread riêng để chạy Task đó*** )  + Mỗi lần một Task được gửi tới thread pool (thông qua Task.Run() hoặc các phương thức khác), thread pool sẽ sử dụng các thread có sẵn trong pool để thực thi các tác vụ.  + Thread pool có thể tái sử dụng các thread để xử lý các Task khác nhau, do đó mỗi thread có thể chạy nhiều Task trong suốt vòng đời của nó (nhưng 1 Thread không thể chạy đồng thời nhiều Task).  **---------------------------------------------------------------------------------------------------------------------------------------**  **Thread pool** là một tập hợp các thread đã được tạo sẵn và sẵn sàng để thực thi công việc khi có yêu cầu. Khi bạn cần thực hiện một tác vụ (ví dụ như một tính toán hoặc tác vụ I/O), thay vì tạo mới một thread, bạn có thể **yêu cầu một thread từ thread pool**.  Các lợi ích của việc sử dụng thread pool bao gồm:   * **Giảm chi phí tạo thread mới**: Tạo và hủy một thread có thể tiêu tốn tài nguyên, đặc biệt là khi bạn phải tạo nhiều thread cho các tác vụ ngắn. Thread pool tái sử dụng các thread đã được tạo sẵn. * **Tối ưu hóa hiệu suất**: Thread pool giúp giảm độ trễ và cải thiện khả năng phản hồi của ứng dụng khi phải thực thi nhiều tác vụ song song. * **Quản lý tài nguyên tốt hơn**: Thread pool kiểm soát số lượng thread tối đa mà ứng dụng có thể sử dụng, tránh việc sử dụng quá nhiều tài nguyên hệ thống. |