Spring Boot ITS

# Xử lý Bất đồng bộ, Logging, và Hoàn thành Dự án

1. Xử lý các tác vụ bất đồng bộ với Spring Boot (Async, Scheduling).
2. Khái niệm, ưu nhược đa luồng - bất đồng bộ

* Đa luồng
* Đa luồng (multithreading) trong Java là khả năng của một chương trình để thực thi nhiều luồng cùng một lúc. Mỗi luồng sẽ thực hiện một nhiệm vụ riêng biệt, cho phép chương trình thực hiện nhiều công việc song song.
* Ưu điểm
* Tăng hiệu suất: khi sử dụng đa luồng, chương trình có thể thực hiện nhiều công việc cùng một lúc, giúp tối ưu hóa thời gian thực thi.
* Tận dụng tài nguyên: cho phép tận dụng tốt các tài nguyên hệ thống như CPU và bộ nhớ.
* Phản ứng nhanh: khi sử dụng đa luồng, chương trình có thể phản ứng nhanh hơn với các sự kiện xảy ra đồng thời.
* Nhược điểm
* Phức tạp khi quản lý: Khi sử dụng nhiều luồng cùng hoạt động, lập trình viên cần phải xử lý các vấn đề liên quan đến đồng bộ hóa (synchronization) giữa các luồng. Điều này dẫn đến các vấn đề như **deadlock** (kẹt tài nguyên), **race condition** (trạng thái cạnh tranh) hay **livelock** (vòng lặp vô tận), khó phát hiện và khắc phục.
* Tốn tài nguyên và chi phí quản lý: Việc khởi tạo và quản lý các luồng tiêu tốn nhiều tài nguyên hệ thống như CPU và bộ nhớ. Khi số lượng luồng lớn, việc quản lý luồng có thể gây ra sự quá tải (thread contention), dẫn đến hiệu suất thấp hơn nếu không được tối ưu hóa đúng cách.
* Khó gỡ lỗi và kiểm tra: Khi các luồng chạy song song, hành vi của chương trình có thể khác nhau mỗi khi chạy, đặc biệt khi xảy ra lỗi. Điều này làm cho việc gỡ lỗi (debug) trở nên rất khó khăn vì các lỗi liên quan đến đa luồng thường xuất hiện không thường xuyên và khó tái hiện.
* Khó khăn trong việc đảm bảo tính nhất quán của dữ liệu: Khi nhiều luồng truy cập vào cùng một tài nguyên hoặc dữ liệu mà không có cơ chế khóa hợp lý, dữ liệu có thể bị hỏng hoặc không nhất quán.
* Bất đồng bộ
* Công việc bất đồng bộ (asynchronous tasks) trong Java liên quan đến việc thực hiện các tác vụ mà không cần chờ đợi kết quả trả về ngay lập tức. Thay vì chờ đợi, chương trình có thể tiếp tục thực thi các công việc khác trong khi công việc bất đồng bộ đang được xử lý.
* Ưu điểm
* Tối ưu hóa hiệu suất: công việc bất đồng bộ giúp tối ưu hóa hiệu suất của chương trình bằng cách cho phép thực hiện các tác vụ mà không cần chờ đợi kết quả trả về ngay lập tức.
* Xử lý sự kiện: khi xử lý các sự kiện trong ứng dụng, công việc bất đồng bộ giúp chương trình không bị chậm trễ do việc chờ đợi kết quả trả về từ các tác vụ.
* Nhược điểm
* Việc xử lý bất đồng bộ đòi hỏi phải viết mã để xử lý kết quả khi quá trình hoàn tất (callback, Future, CompletableFuture). Điều này có thể làm mã trở nên khó đọc, khó hiểu và khó bảo trì, nhất là khi có nhiều tầng gọi lồng nhau (callback hell).
* Mặc dù bất đồng bộ có thể giúp giảm thiểu chi phí về số lượng luồng, nhưng nó lại cần nhiều hơn sự quản lý về trạng thái (state management). Quá trình chuyển đổi trạng thái giữa các tác vụ bất đồng bộ có thể gây thêm độ trễ và tốn tài nguyên cho hệ thống.
* Tương tự như đa luồng, các tác vụ bất đồng bộ có thể được hoàn thành tại những thời điểm không dự đoán trước, khiến việc kiểm tra (testing) và gỡ lỗi gặp nhiều thách thức.
* Nếu các tác vụ bất đồng bộ không được sắp xếp và kiểm soát chặt chẽ, có thể dẫn đến các vấn đề tương tự về dữ liệu, đặc biệt là khi dữ liệu phụ thuộc vào các tác vụ hoàn thành theo một thứ tự nhất định.
* Overhead về lập trình và kỹ năng: **Đa luồng** và **bất đồng bộ** đều yêu cầu lập trình viên có kiến thức sâu hơn và kỹ năng lập trình phức tạp hơn so với các mô hình lập trình tuần tự thông thường. Điều này có thể tăng thời gian phát triển và khó khăn cho việc đào tạo hoặc bảo trì mã.
* loading ♾️

1. async
2. Cấu hình @EnableAsync

* Thêm annotation @EnableAsync vào một trong các lớp cấu hình

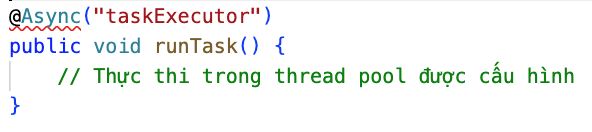
1. Kiểu trả về

* void
* future
* Future là một interface trong Java dùng để đại diện cho một kết quả chưa có sẵn ngay lập tức mà sẽ có trong tương lai. Tuy nhiên, Future có một số hạn chế như không hỗ trợ kết hợp nhiều tác vụ bất đồng bộ hoặc không cung cấp phương pháp để xử lý ngoại lệ một cách trực tiếp.
* completable future
* CompletableFuture mở rộng khả năng của Future bằng cách hỗ trợ chaining (chuỗi các tác vụ), xử lý ngoại lệ và lập trình bất đồng bộ dễ dàng hơn.
* **Chaining tasks:** Có thể liên kết nhiều tác vụ lại với nhau.
* **Exception handling:** Hỗ trợ dễ dàng quản lý ngoại lệ.
* **Non-blocking:** Cho phép xử lý kết quả ngay khi có mà không cần chặn luồng chính (main thread).

1. Tối ưu hoá với executer

* Mặc định, Spring sử dụng SimpleAsyncTaskExecutor, nhưng bạn có thể tùy chỉnh để sử dụng ThreadPoolTaskExecutornhằm kiểm soát số lượng thread, hàng đợi (queue), và các tùy chọn quản lý khác.



* Sử dụng Executor này trong các tác vụ async
* loading ♾️

1. Những hạn chế và lưu ý

* Phương thức chú thích @Async **phải trả về void**, Future, hoặc CompletableFuture.
* @Async **không hoạt động trong cùng một lớp**. Nếu bạn gọi phương thức @Async từ trong cùng một lớp, nó sẽ không kích hoạt bất đồng bộ vì Spring không có cơ hội để proxy phương thức đó.
* **Exception handling trong @Async** cần đặc biệt chú ý, vì exception không được xử lý bình thường khi chạy bất đồng bộ. Bạn cần xử lý lỗi cẩn thận trong thread bất đồng bộ hoặc sử dụng Future/CompletableFuture để bắt lỗi.

1. Các trường hợp sử dụng phổ biến

* **Gửi email**: Các tác vụ như gửi email không cần phản hồi ngay lập tức.
* **Đọc/ghi file**: Các tác vụ I/O có thể chạy bất đồng bộ để giảm tải cho thread chính.
* **Tính toán phức tạp**: Các tác vụ tốn tài nguyên như tính toán toán học phức tạp có thể tách riêng thành các thread riêng biệt.

1. Ví dụ
2. scheduling
3. KN

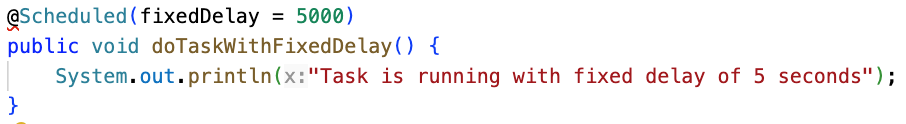
* @Scheduled lên lịch cho các tác vụ chạy tự động tại các thời điểm định sẵn.

1. Kích hoạt

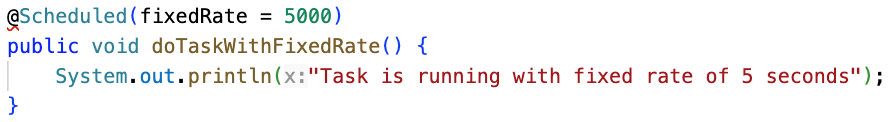
* Thêm annotation @EnableScheduling vào một trong các class cấu hình chính của ứng dụng để bật chức năng lên lịch.

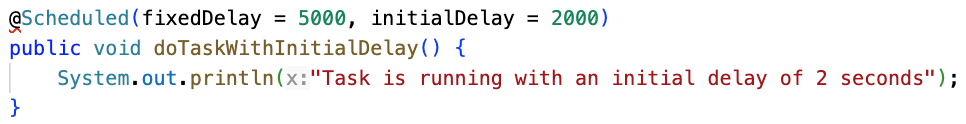
1. Phương thức tuỳ chỉnh thời gian

* Fixed Delay

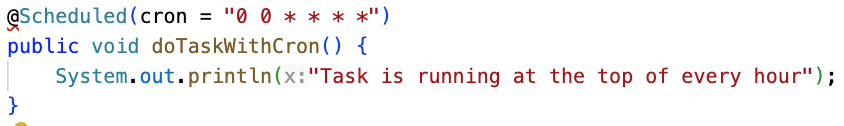


* Fixed Rate



* Initial Delay
* 
* Cron Expression

second - minute - hour - dayOfMonth - month - dayOfWeek



1. Tối ưu hóa và quản lý logging với Spring Boot (Logback, SLF4J).
2. tìm hiểu logging
3. Logging

Logging là quá trình ghi lại thông tin về quá trình thực thi của ứng dụng khi nó đang chạy. Những thông tin này có thể bao gồm:

* Thông tin gỡ lỗi (debugging information)
* Lỗi (errors)
* Cảnh báo (warnings)
* Các thông điệp tổng quát khác giúp lập trình viên và quản trị viên hệ thống hiểu rõ hơn về cách mà ứng dụng đang hoạt động.

1. Các Framework Logging phổ biến trong Java

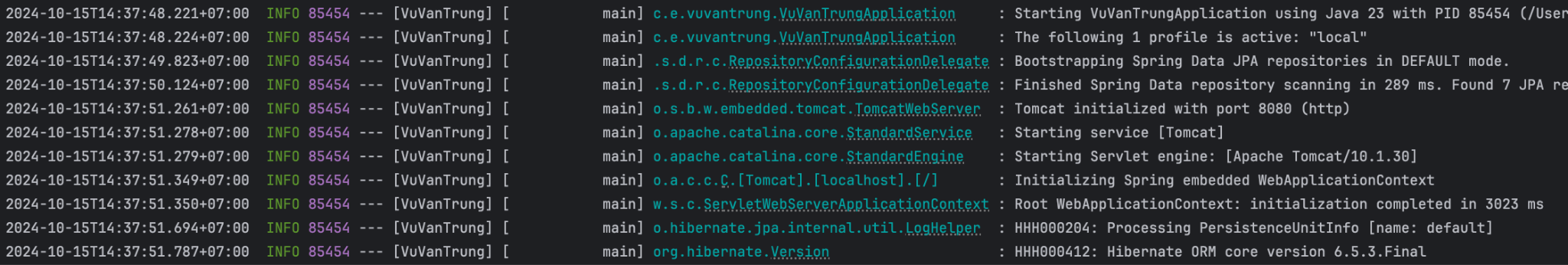
* **java.util.logging (JUL)**: Là một phần của thư viện chuẩn của Java từ JDK 1.4, tích hợp sẵn trong Java.
* **Log4j**: Là một framework ghi log rất phổ biến, nổi bật nhờ tính linh hoạt và nhiều tính năng.
* **SLF4J** (Simple Logging Facade for Java): Là một facade (lớp giao diện) cung cấp một chuẩn chung cho các framework logging khác (ví dụ: Log4j, java.util.logging), cho phép người dùng tùy ý chọn framework muốn sử dụng.
* **Logback**: Một framework logging hiện đại, được phát triển để kế thừa Log4j, thường được sử dụng cùng với SLF4J.

1. Logging Levels

Hầu hết các framework logging cho phép bạn chỉ định mức độ cho từng thông điệp log. Dưới đây là các mức độ phổ biến trong các framework như Log4j và SLF4J:

* **TRACE**: Mức chi tiết nhất, thường dùng để ghi lại tất cả các hoạt động của hệ thống.
* **DEBUG**: Mức dành cho các thông tin gỡ lỗi, giúp lập trình viên theo dõi dòng chảy của chương trình trong quá trình phát triển.
* **INFO**: Ghi lại các thông tin quan trọng về hoạt động của ứng dụng, nhưng không mang tính gỡ lỗi.
* **WARN**: Cảnh báo về những tình huống có thể không bình thường, nhưng ứng dụng vẫn có thể tiếp tục hoạt động.
* **ERROR**: Thông báo khi xảy ra lỗi, điều này có thể làm gián đoạn hoạt động của hệ thống.
* **FATAL**: Lỗi nghiêm trọng, có thể khiến ứng dụng phải dừng ngay lập tức.

1. Định dạng log mặc định của spring boot



* Ngày và thời gian: độ chính xác mili giây và có sắp xếp.
* Mức độ log: ERROR, WARN, INFO, DEBUG và TRACE.
* ID của tiến trình.
* Ký hiệu --- ngăn cách để phân biệt bắt đầu của các nội dung log cụ thể.
* Tên luông: Bao bởi ngoặc vuông(có thể bị cắt bớt ở đầu ra của console).
* Tên logger: Thường là tên của class nguồn(thường viết tắt khi tên package dài và nằm lòng nhiều tầng).
* Nội dung log.

1. loading ♾️
2. Logback
3. Logback

* Logback là một framework logging hoàn chỉnh, và nó cũng là backend logging mặc định khi sử dụng SLF4J.

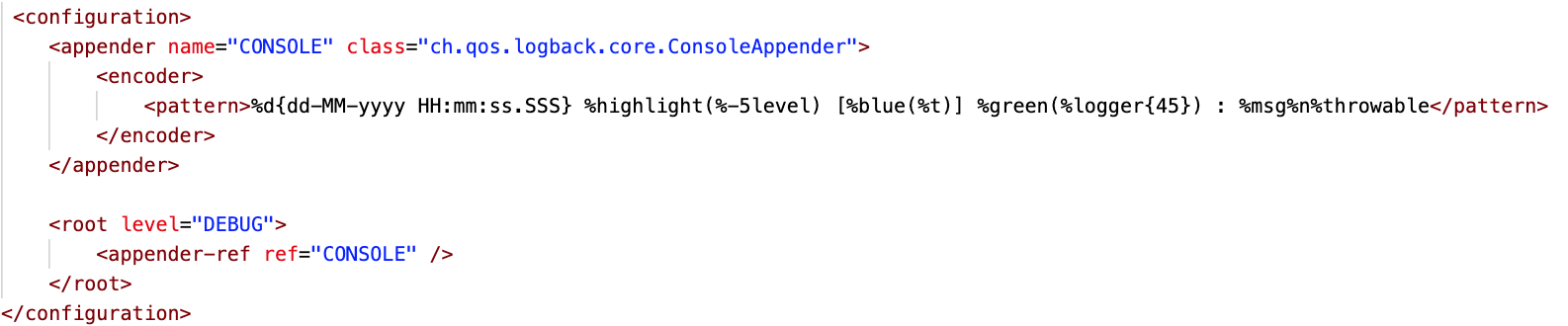
1. Tính năng chính

* **Hiệu suất tốt:** Logback được thiết kế để có hiệu suất cao hơn so với các framework logging khác như Log4j.
* **Cấu hình mạnh mẽ:** Logback hỗ trợ cấu hình thông qua file logback.xml, cho phép cấu hình các appender (nơi log được ghi, như file, console, v.v.) và layout (định dạng của log).
* **Đa dạng appender:** Logback cung cấp nhiều appender hữu ích (ConsoleAppender, FileAppender, RollingFileAppender, v.v.).
* **Hỗ trợ Rolling Logs:** Logback có thể quản lý việc xoay vòng file log (rolling log) để tránh việc file log quá lớn, bằng cách tách chúng ra thành nhiều file.

1. Các Appender phổ biến

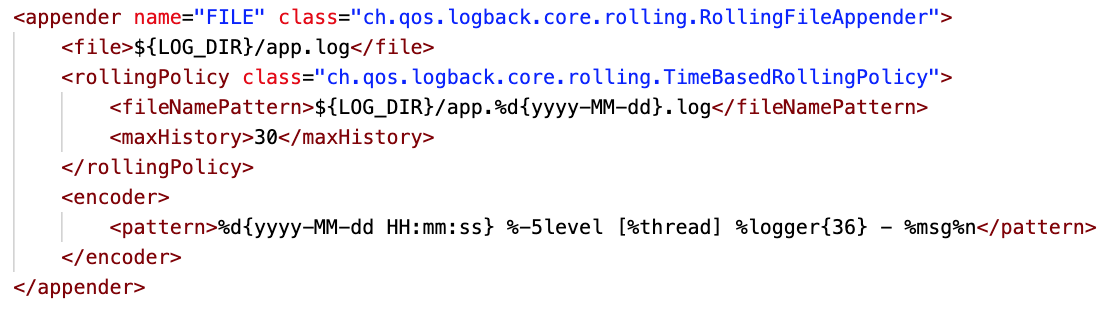
* **ConsoleAppender**: Ghi log ra console (terminal).
* **FileAppender**: Ghi log vào file.
* **RollingFileAppender**: Ghi log vào file và tự động xoay (tạo file log mới khi file hiện tại vượt quá giới hạn dung lượng hoặc thời gian).

1. Để cấu hình cho việc ghi log

* Ví dụ cơ bản để cấu hình các log với màu và level log
* 

1. Log ra file

* Ví dụ cấu hình log ra file



1. loading nếu cần ♾️
2. SLF4J
3. SLF4J

* SLF4J không phải là một framework logging mà là một **facade** (giao diện) chung cho các framework logging khác. Nó cung cấp một cách trừu tượng để bạn có thể thay đổi framework logging mà không cần sửa đổi mã nguồn.

1. Chức năng chính

* SLF4J hỗ trợ nhiều framework logging khác nhau (Log4j, Logback, JUL, v.v.) thông qua các bindings.
* Nó cung cấp một API logging thống nhất, nhưng bản thân SLF4J không thực hiện việc ghi log. Thay vào đó, nó chuyển tiếp các lệnh gọi log tới một backend cụ thể (như Logback, Log4j).
* SLF4J thường được sử dụng để tách biệt mã nguồn khỏi việc phụ thuộc trực tiếp vào một framework logging cụ thể.

1. Logging với các mức

* TRACE: Cấp thấp nhất, chủ yếu dành cho thông tin chi tiết.
* DEBUG: Thường được dùng để log thông tin khi phát triển và gỡ lỗi.
* INFO: Log các thông tin chung về tiến trình hoặc hoạt động của ứng dụng.
* WARN: Cảnh báo về những vấn đề có thể không gây lỗi ngay lập tức nhưng cần được chú ý.
* ERROR: Ghi lại các lỗi nghiêm trọng cần được xử lý.

1. Giao diện của logback để người dùng tương tác

* khởi tạo logger
* Khởi tạo Logger bằng **LoggerFactory.getLogger()**



===> MyApp.class được truyền vào LoggerFactory.getLogger(), điều này giúp logger có tên giống với class, giúp dễ dàng theo dõi log từ các class khác nhau.

* Logger với Lombok (**@Slf4j** Annotation)



* loading nếu cần ♾️
* cách ghi log
* Ví dụ: logger.Mức(“log ra”)



* haizzz
* loading nếu cần ♾️

1. loading nếu cần ♾️
2. Tạo các service bất đồng bộ và thực thi các tác vụ định kỳ.
3. Cấu hình logging hiệu quả và giám sát ứng dụng Spring Boot.
4. Hoàn thiện một ứng dụng Spring Boot đầy đủ tính năng.

III + IV + V ===> done trong quá trình tìm hiểu, vừa tìm hiểu vừa xây dựng ===> xem trong code :v (cái bài điện)