Schedule with quartz lib :/

1. **Quartz, tại sao nên dùng quartz**
2. **Quartz**

Quartz là một thư viện lập lịch (scheduling) mạnh mẽ, linh hoạt và phổ biến trong Java

1. **Ưu điểm**

1. Linh Hoạt và Đa Dạng Tính Năng

* Hỗ trợ nhiều loại Trigger
* Biểu thức Cron linh hoạt
* Quản lý Misfire: các tùy chọn quản lý lỗi thời gian (misfire) nếu trigger không kích hoạt đúng thời điểm

2. Tính Bền Vững và Hỗ Trợ Clustering

3. Khả Năng Mở Rộng và Tùy Chỉnh Cao

* Job và Trigger tùy chỉnh
* Listeners
* Hỗ trợ Dependency Injection với Spring

4. Quản Lý Lịch Trình Phức Tạp

5. Độ Tin Cậy Cao

6. Khả Năng Tích Hợp Mạnh Mẽ Với Các Ứng Dụng Java

* Spring Boot và Spring Framework
* Java EE
* Hỗ trợ đồng bộ và bất đồng bộ

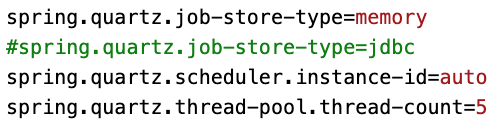
7. Quartz Scheduler Tương Thích Với Nhiều Loại Ứng Dụng

* Ứng dụng nhỏ dùng ram job store
* Hệ thống lớn, yêu cầu clustering có thể dùng JDBCJobStore hoặc TerracottaJobStore

1. **Thêm dependency**



1. **Cấu hình**



* spring.quartz.job-store-type=...
* memory : Lưu trong ram, phù hợp cho môi trường phát triển và thử nghiệm vì công việc sẽ bị mất khi ứng dụng tắt
* jdbc : Lưu trong cơ sở dữ liệu, phù hợp production
* custom : tự cấu hình các thành phần Quartz và cung cấp một JobStore tùy chỉnh
* spring.quartz.scheduler.instance-id=auto để Quartz tự động sinh ID.
* spring.quartz.thread-pool.thread-count=5 để đặt số lượng luồng cho Quartz Scheduler.
* spring.quartz.jdbc.initialize-schema=always # Tạo các bảng Quartz cần thiết tự động

1. **Các khái niệm**

**A. Job**

1. Là các công việc cụ thể cần thực hiện
2. Các lớp triển khai

* Job là interface cốt lõi của Quartz, yêu cầu bạn phải triển khai phương thức execute(JobExecutionContext context).
* QuartzJobBean là một abstract class của Spring dành cho việc sử dụng Quartz trong Spring. Nó giúp tạo ra sự tích hợp giữa Spring và Quartz dễ dàng hơn. Giúp dễ inject vào service, repo …

**B. JobDetail**

1. là đối tượng chứa thông tin mô tả job (ví dụ tên job, nhóm của job) nhưng không bao gồm logic chạy công việc. Nó được dùng để đăng ký job với scheduler.
2. Các lớp triển khai

* JobDetailFactoryBean: factory bean trong Spring giúp tạo ra JobDetail. Nó tích hợp chặt chẽ với Spring, hỗ trợ quản lý và cấu hình JobDetailnhư một Spring Bean
* JobDetail: lớp của quartz, giúp mô tả job

**C. Trigger**

1. Để xác định thời điểm và tần suất thực thi Job. Quartz có 2 loại phổ biến là SimpleTrigger (chạy một lần hoặc nhiều lần cách nhau khoảng thời gian nhất định) và CronTrigger (định lịch theo cron expression).
2. Triển khai

* TriggerBuilder + CronScheduleBuilder hoặc SimpleScheduleBuilder

**D. Scheduler**

1. Là thành phần quản lý và khởi chạy các job và trigger. Đây là trung tâm điều khiển cho tất cả các hoạt động của Quartz.
2. Note: nhưng chỉ cần tập trung cho job, job detail, trigger

**E. JobStore**

1. Để lưu trữ thông tin về các job và trigger. Có thể sử dụng memory hoặc jdbc tùy vào nhu cầu lưu trữ và tính bền vững của dữ liệu.

**F. Listener**

1. Vấn đề : Mỗi khi một công việc nào đó bắt đầu, kết thúc, hoặc gặp lỗi, có thể theo dõi và xử lý chúng ===> listener
2. Là gì: là các thành phần giúp bạn "lắng nghe" các sự kiện quan trọng trong quá trình Quartz xử lý công việc (Jobs) và lịch trình (Triggers)

* Khi một công việc bắt đầu thực thi
* Khi một công việc hoàn tất
* Khi một công việc bị hủy
* Khi một lịch trình bị lỗi thời gian (không chạy đúng lúc)
* Khi Quartz Scheduler khởi động hoặc dừng lại

1. Các loại listener

* Job Listener - Lắng nghe các sự kiện liên quan đến công việc (Job)
* Trigger Listener - Lắng nghe các sự kiện liên quan đến lịch trình (Trigger)
* Scheduler Listener - Lắng nghe các sự kiện liên quan đến toàn bộ Quartz Scheduler

1. **Tìm hiểu chi tiết hơn về job detail, job store, trigger, listener**

**A. JobDetail**

1. Tạo 1 jobdetail

* Có thể sử dụng JobBuilder
* Các thuộc tính
* .newJob(MyJob.class) : Đặt lớp Job cần thực thi
* .withIdentity("myJob", "group1") : Đặt tên và nhóm của Job
* withDescription("This is a test job") : Đặt mô tả cho Job
* .usingJobData("key", "value") : Truyền dữ liệu cho Job
* .storeDurably() : // Cấu hình để giữ JobDetail kể cả khi chưa có Trigger

1. 1 số phương thức

* getDescription()
* isDurable() : Kiểm tra xem JobDetail có được cấu hình để giữ lại trong bộ nhớ khi chưa có trigger hay không.
* getJobDataMap() : Trả về JobDataMap chứa dữ liệu của job.

**B. Trigger**

1. Các khái niệm cơ bản

* startTime: Thời gian bắt đầu kích hoạt Trigger.
* endTime: Thời gian kết thúc, sau thời điểm này Trigger sẽ không kích hoạt nữa.
* repeatInterval: Khoảng thời gian lặp lại giữa các lần kích hoạt (dùng cho SimpleTrigger).
* misfireInstruction: Xử lý lỗi "misfire" khi Trigger không thể kích hoạt đúng thời điểm dự kiến (chẳng hạn do hệ thống quá tải hoặc ứng dụng bị tạm dừng).

1. simple với cron

2.1 simple trigger

* thực thi một công việc tại một thời điểm cụ thể và lặp lại sau một khoảng thời gian nhất định.
* Các Thành Phần Chính của SimpleTrigger
* repeatCount: Số lần lặp lại của trigger (sử dụng SimpleTrigger.REPEAT\_INDEFINITELY nếu muốn lặp lại vô hạn).
* repeatInterval: Khoảng thời gian lặp lại giữa các lần kích hoạt.

2.2 cron trigger

* biểu thức cron : <giây> <phút> <giờ> <ngày> <tháng> <ngày trong tuần>
* Special Characters

| Ký tự | Ý nghĩa | Ví dụ |
| --- | --- | --- |
| \* | Tất cả các giá trị | \* \* \* \* \* ? - Chạy mỗi giây |
| ? | Không xác định giá trị (dùng cho ngày trong tháng hoặc ngày trong tuần) | 0 0 12 \* \* ? - Mỗi ngày vào 12 giờ trưa |
| - | Khoảng giá trị | 0 0 12 1-5 \* ? - Từ ngày 1 đến 5 hàng tháng |
| , | Nhiều giá trị | 0 0 12 1,15 \* ? - Ngày 1 và ngày 15 hàng tháng |
| / | Bước nhảy công sai | 0 0/15 \* \* \* ? - Mỗi 15 phút |
| L | Ngày cuối cùng trong tháng hoặc trong tuần | 0 0 12 L \* ? - Ngày cuối của tháng |
| W | Ngày làm việc gần nhất | 0 0 12 15W \* ? - Ngày làm việc gần nhất ngày 15 |
| # | Ngày trong tuần theo thứ tự | 0 0 12 ? \* 2#1 - Thứ Hai đầu tiên của tháng |

1. Misfire

* MISFIRE\_INSTRUCTION\_IGNORE\_MISFIRE\_POLICY: Bỏ qua misfire và tiếp tục chạy theo lịch trình ban đầu.
* MISFIRE\_INSTRUCTION\_FIRE\_NOW: Kích hoạt job ngay khi có thể, rồi tiếp tục lịch trình lặp lại.
* MISFIRE\_INSTRUCTION\_RESCHEDULE\_NOW\_WITH\_EXISTING\_REPEAT\_COUNT: Đặt lại lịch trình bắt đầu từ hiện tại và tiếp tục với số lần lặp còn lại.

**C. JobListener**

1. Mục đích

* theo dõi các sự kiện như: khi một job sắp được thực thi, khi job hoàn thành, hoặc khi job bị hủy.

1. Các phương thức

* getName() : Bắt buộc phải triển khai phương thức này và trả về một tên duy nhất cho listener.
* jobToBeExecuted(JobExecutionContext context) : được gọi ngay trước khi một job bắt đầu thực thi. Bạn có thể dùng phương thức này để ghi log hoặc kiểm tra trước khi job chạy.
* jobExecutionVetoed(JobExecutionContext context) : được gọi nếu việc thực thi job bị hủy trước khi bắt đầu. Đây là cơ hội để bạn xử lý hoặc ghi log khi một job bị hủy. Lưu ý rằng phương thức này chỉ được gọi khi có TriggerListener hoặc một phần mềm khác hủy bỏ job.
* jobWasExecuted(JobExecutionContext context, JobExecutionException jobException) : được gọi ngay sau khi job hoàn thành, bất kể job thực thi thành công hay bị lỗi. Nếu job bị lỗi, thông tin về lỗi sẽ có trong tham số JobExecutionException.

**D. TriggerListener**

1. Mục đích

* Trigger kích hoạt một job, gặp lỗi thời gian (misfire) hoặc hoàn thành, bạn có thể sử dụng TriggerListener để theo dõi và thực hiện các hành động cụ thể.

1. Các phương thức

* getName() : Tên này cần được định danh duy nhất, hữu ích khi có nhiều TriggerListener trong hệ thống.
* triggerFired(Trigger trigger, JobExecutionContext context): Trigger được kích hoạt và job sắp thực sự chạy.
* vetoJobExecution(Trigger trigger, JobExecutionContext context): cho phép bạn chặn việc thực thi của job nếu trả về true, triggerMisfired sẽ không được gọi.
* triggerMisfired(Trigger trigger): Được gọi khi Trigger không thể kích hoạt đúng thời gian dự kiến (misfire).
* triggerComplete(Trigger trigger, JobExecutionContext context, Trigger.CompletedExecutionInstruction triggerInstructionCode): Được gọi khi job hoàn thành việc thực thi và trigger đã kích hoạt xong.

**E. Job store**

1. Tổng quan

* Lưu trữ thông tin về các job và trigger. Quản lý dữ liệu về các lịch trình, bao gồm thông tin của các công việc, lịch kích hoạt, trạng thái của các trigger, và các chi tiết khác liên quan đến lịch trình.
* Các loại job store chính
* RAMJobStore: Lưu trữ dữ liệu trong bộ nhớ RAM.
* JDBCJobStore: Lưu trữ dữ liệu trong cơ sở dữ liệu.
* TeracottaJobStore: Lưu trữ dữ liệu phân tán (được sử dụng trong môi trường phân tán, sử dụng Teracotta để lưu trữ trạng thái job).

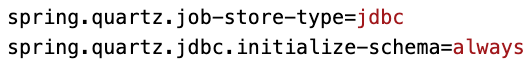
1. RAMJobStore - mặc định của quartz

* Đặc điểm
* Hiệu suất cao: Vì lưu trữ trong RAM, RAMJobStore có tốc độ truy xuất rất nhanh.
* Không bền vững: Tất cả dữ liệu sẽ bị mất khi ứng dụng hoặc hệ thống khởi động lại.
* Không hỗ trợ cluster: RAMJobStore chỉ hoạt động trong một node duy nhất và không hỗ trợ clustering (chạy trên nhiều máy chủ).
* Cấu Hình

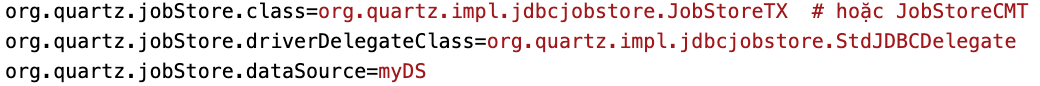


1. JDBCJobStore - lưu trữ tất cả các thông tin về job, trigger, và lịch trình trong cơ sở dữ liệu

* Đặc điểm
* Bền vững: Dữ liệu về các job và trigger được lưu trữ trong cơ sở dữ liệu, nên vẫn giữ lại sau khi khởi động lại hệ thống.
* Hỗ trợ cluster: Nhiều node có thể chia sẻ cùng một cơ sở dữ liệu để thực hiện clustering.
* Hiệu suất phụ thuộc vào cơ sở dữ liệu: Do tất cả thao tác lưu trữ đều diễn ra trên cơ sở dữ liệu, hiệu suất phụ thuộc vào tốc độ và khả năng xử lý của cơ sở dữ liệu.
* Cấu hình

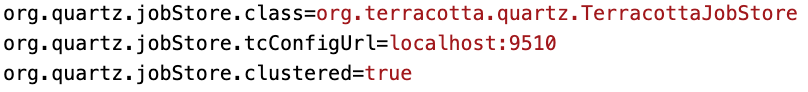


* Các Chế Độ của JDBCJobStore
* JobStoreTX: Sử dụng giao dịch và yêu cầu một DataSource quản lý giao dịch. Phù hợp cho các ứng dụng yêu cầu tính nhất quán cao.
* JobStoreCMT: Phối hợp với các giao dịch JTA (Java Transaction API). Sử dụng trong các ứng dụng Java EE hoặc các ứng dụng đòi hỏi giao dịch phân tán.



1. **TerracottaJobStore - Job Store phân tán**

* Đặc điểm
* Hỗ trợ clustering phân tán: Thích hợp cho các hệ thống lớn, yêu cầu độ bền vững và chạy trên nhiều node trong môi trường phân tán.
* Bền vững: Lưu trữ trạng thái job trên Terracotta Server, cho phép giữ lại trạng thái job giữa các lần khởi động lại.
* Phụ thuộc vào Terracotta: Cần cài đặt và cấu hình Terracotta Server Array.
* Cấu hình



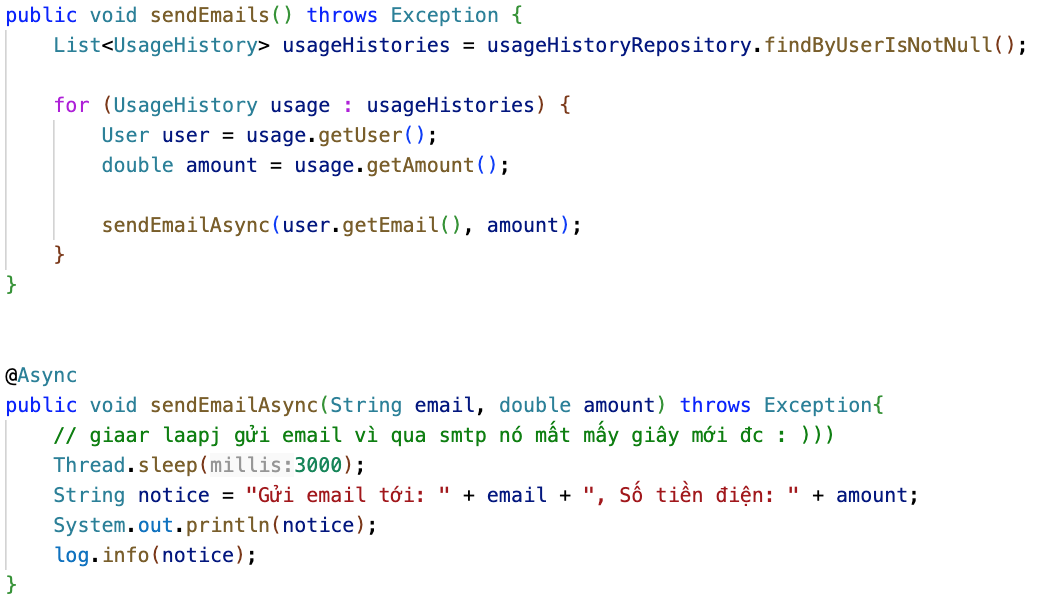
1. **Triển khai thay cho @scheduled lập lịch trước đó**

**A. NOTE**

Quartz là một thư viện mạnh mẽ với nhiều tính năng, có thể có nhiều cách tiếp cận khác nhau. Ở doc này e triển khai theo các lớp hỗ trợ cho spring

**B. Code**

* Vẫn là chương trình lập lịch gửi mail bất đồng bộ đó



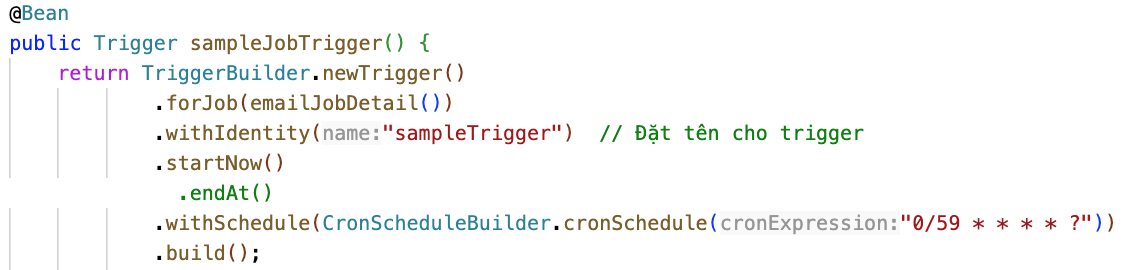
* Tạo job



* Tạo jobdetail



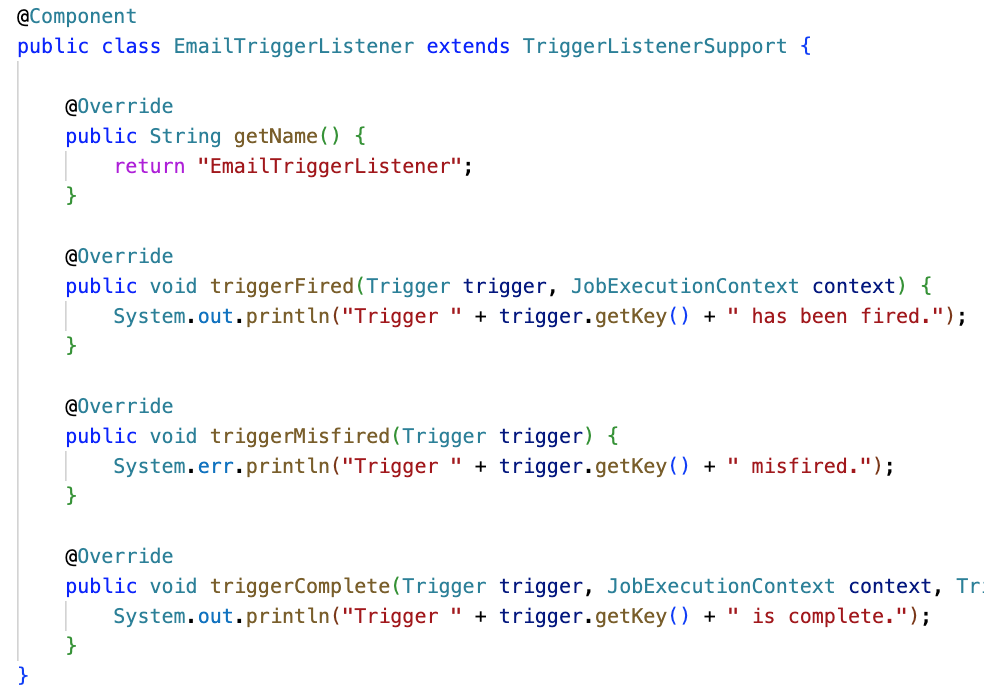
* Tạo trigger



* Triển khai job listener



* triển khai trigger listener



* Đăng ký listener

