Tìm hiểu về Spring boot

1. Tight coupled

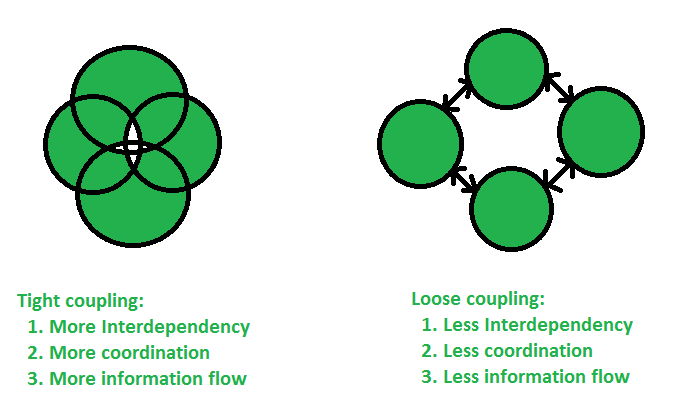
* Tight coupled (liên kết ràng buộc) là một khái niệm trong Java ám chỉ mối quan hệ giữa các class quá chặt chẽ. Khi yêu cầu thay đổi logic hay một class bị lỗi sẽ dẫn tới ảnh hưởng tới những class khác liên kết với nó.
* Do mối quan hệ chặt chẽ nên nó thường không tốt bởi nó giảm sự linh hoạt và tái sử dụng trong mã.
* Đối tượng được liên kết chặt chẽ là một đối tượng cần biết về đối tượng khác và thường phụ thuộc rất nhiều vào nhau
* Thay đổi một đối tượng trong một ứng dụng tight coupled thường yêu cầu thay đổi một số đối tượng khác
* Trong các ứng dụng nhỏ, ta có thể dễ dàng xác định những thay đổi và có ít khi bỏ lỡ điều gì. Nhưng trong các ứng dụng lớn, các phụ thuộc không phải lúc nào cũng được biết hết bởi tất cả lập trình viên và khó nhìn thấy sự thay đổi.

1. Loosely coupled

* Loosely coupled là cách ám chỉ việc giảm bớt sự phụ thuộc giữa các Class với nhau
* Loosely coupled khuyến khích việc thiết kế các lớp có vai trò riêng biệt và giảm sự liên quan lẫn nhau trong chương trình
* Mục tiêu của loosely coupled là giảm nguy cơ một thay đổi được thực hiện trong một thành phần sẽ tạo những thay đổi trong các thành phần khác. Hạn chế mối quan hệ giúp cô lập các vấn đề khi có lỗi và đơn giản hóa các thủ tục kiểm tra, bảo trì và khắc phục sự cố.
* Mặc dù loosely couped giảm thiểu mối quan hệ giữa các thành phần trong hệ thống nhưng nó vẫn có thể tạo ra các tương tác khi mong muốn.
* Một class loosely couped có được sử dụng và thử nghiệm độc lập với các lớp khác
* Interface là một công cụ mạnh mẽ để sử dụng cho loosely couped. Các lớp có thể giao tiếp thông qua các interface hơn là với các lớp khác, với bất kì lớp nào cần giao tiếp, giao tiếp sẽ đơn giản hơn nếu sử dụng interface

1. So sánh giữa tight coupled và loosely coupled

* Nói chung tight coupled thường không tốt, nó làm giảm tính linh hoạt và tái sử dụng code, làm cho việc thay đổi trở lên khó khăn hơn, cản trở khả năng thử nghiệm,… Loosely coupled là một lựa chọn tốt hơn bởi vì một kết nối lỏng lẻo sẽ giúp nhiều hơn khi cần thay đổi. Nếu thiết kế với loosely coupled thì chỉ vài phần của ứng dụng bị ảnh hưởng khi cần thay đổi.



* Tight coupled không tốt ở khả năng kiểm tra, loosely coupled cải thiện khả năng kiểm tra
* Tight coupled không sử dụng các Interface , loosely coupled sử dụng interdace
* Trong tight coupled, khó khăn để trao đổi giữa hai lớp, nhưng nó dễ dàng để trao đổi các phần khác của code, modules, objects components trong loosely coupt
* Tight coupled không có khả năng thay thế, loosely coupled rất dễ để thay thế.

1. Dependency ịnjection

* Là một khía cạnh cơ bản của Spring framework, thông qua đó các Spring container tiêm các đối tượng vào các đối tượng khác hoặc dependency
  1. . Inversion of control
* là một nguyên tắc trong kĩ thuật phần mềm mà theo đó kiểm soát của các đối tượng hoặc các phần của chương trình được chuyển giao cho một container hoặc framework. Nó thường được sử dụng trong lập trình hướng đối tượng.
* IoC cho phép một framework kiểm soát hoạt động của một chương trình và thực hiện gọi đến các code tùy chỉnh. Để thực hiện tính năng này, các framework sử dụng abstractions với chế độ bổ sung được tích hợp sẵn. Nếu muốn thêm hành vi, cần mở rộng các lớp của framework hoặc các plugin của các class.
* Lợi thế:

+ Tách việc thực hiện nhiệm vụ trong thực thi(implement)

+ Dễ dàng hơn để chuyển đổi giữa các thực thi khác nhau

+ Module lớn hơn trong 1 chương trình

+ Dễ dàng hơn trong việc thử nghiệm một chương trình bằng cách cô lập 1 thành phần, giảm phụ thuộc, cho phép các thành phần giao tiếp thông qua hợp đồng.

* 1. Dependency ịnjection
* Là một mẫu để thông qua đó thực hiện IoC, thiết lập các đối tượng phụ thuộc
* Các thành phần hoặc lớp java nên được độc lập với các lớp java khác. Để tách các thành phần từ các thành phần khác, sư phụ thuộc vào lớp khác nhất định nên được tiêm vào chúng thay vì chính nó tạo ra đối tượng này. Lớp A phụ thuộc vào lớp B nếu nó sử dụng lớp B như một biến
* Là một kĩ thuật lập trình giúp tách các class độc lập với các biến phụ thuộc.

4.3. Các loại dependency injection phổ biến

- constructor injection: các biến phụ thuộc được cung cấp thông qua một hàm constructor

- Setter injection: Các phụ thuộc sẽ được truyền vào class thông qua các hàm setter

- Interface injection: Class cần inject sẽ implement 1 interface.

4.4. Lợi ích

- Dễ test:

- Dễ thấy quan hệ giữa các object: Dependency injection sẽ tiêm các object phụ thuộc và các interface thành phần của object bị phụ thuộc nên dễ dàng thấy được các dependency của một object

- Dễ dàng hơn trong việc mở rộng ứng dụng

- Giảm sự kết dính giữa các thành phần, tránh ảnh hưởng quá nhiều khi có thay đổi

5. Annotation

* 1. @Autowired:
* Annotation được áp dụng trên các trường, phương thức setter và các constructor. Chú thích này tiêm phụ thuộc đối tượng ngầm.
* Khi sử dụng chú thích này trên các trường và chuyển giá trị trên các trường bằng thuộc tính, Spring sẽ tự động gán các trường với các giá trị có sẵn
* Khi sử dụng trên phương thức setter, Spring sẽ cố gắng thực hiện tự động theo loại của phương thức. Spring sẽ khởi tạo property sử dụng phương thức getter nơi ta có thể thêm code tùy chỉnh.
* Khi sử dụng trên một constructor, constructor sẽ tiêm những gì xảy ratrong thời gian tạo đối tượng. Các constructor để Autowired khi được sử dụng như 1 bean.
  1. @Qualifier
* Được sử dụng cùng @Autowired. Khi cần kiểm soát nhiều hơn quá trình tiêm phụ thuộc thì @Qualifier được sử dụng. @Qualifier có thể xác định trên các tham số constructor và tham số của phương thức. Chú thích này được sử dụng để tránh nhầm lẫn xảy ra khi tạo nhiều bean cùng loại và muốn xác định chính xác chỉ 1 trong số đó.
  1. @Configuration
* Chú thích này được sử dụng trên các class xác định bean. @Configuration là 1 analog cho tập tin cấu hình XML. Lớp với chú thích @Configuration là một cấu hình chính nó và có những phương thức để khởi tạo và cấu hình các dependency.
  1. @ComponentScan
* Chú thích này được sử dụng với chú thích @Configuration để cho phép Spring biết các gói để quét các thành phần chú thích.
* @ComponentScan cũng được sử dụng để xác định các gói cơ sở bằng thuộc tính basePackageClasses và basePackage để quét. Nếu thuộc tính đó không được xác định, quét sẽ được xảy ra từ package của lớp chứa chú thích này
  1. @Bean
* Chú thích này được sử dụng với phương thức.
* Chú thích này làm việc với @Configuration để tạo ra bean cho Spring
* Phương thức được chú thích và chú thích này hoạt động như bean ID và nó tạo ra trả về bean thực tế.
  1. @Lazy
* Chú thích này được sử dụng trên class component
* Theo mặc định, tất cả dependency được autowired được tạo ra vào cấu hình khi khởi động. Nếu muốn tạo 1 bean lazy, sử dụng @Lazy trên class. Điều này khiến cho bean được tạo ra và khởi động chỉ khi nó lần đầu tiên được yêu cầu.
* Cũng có thể sử dụng trên các lớp sử dụng @Configuration, tất cả các phương thức sử dụng @bean trong lớp đó sẽ trở thành lazy
  1. @Value
* Chú thích này được sử dụng trên các trường, tham số constructor, tham số của phương thức
* @Value cho biết giá trị mặc định của trường hoặc tham số để khởi tạo thuộc tính.
  1. @Component
* Chú thích được sử dụng trên các class để chỉ ra 1 thành phần của Spring
* @Component đánh dấu lớp là 1 bean hoặc thành phần để khi quét có thể thêm vào application context.
  1. @Controller
* Sử dụng cho biết lớp là 1 controller của spring
* Chú thích này sử dụng để xác định các controller cho Spring MVC hoặc Spring WebFlux
  1. @Service
* Chú thích được sử dụng trên 1 lớp.
* @Service đánh dấu 1 lớp thược hiện một số dịch vụ như thưc hiện business logic, các tính toán, gọi các API bên ngoài.
* Chú thích này là 1 hình thức của @Component sử dụng trong lớp dịch vụ
  1. @Repository
* Chú thích này sử dụng trên các lớp trực tiếp truy cập vào cơ sở dữ liệu.
* @Repository đánh dấu bất kì lớp nào đáp ứng vai trò lưu trữ hoặc đối tượng truy cập dữ liệu.
* Chú thích này có khả năng tự động dịch. Nếu có 1 ngoại lệ xảy ra trong class có chú thích @Repository, có 1 trình xử lí ngoại lệ đó và không cần thiết phải sử dụng try catch
  1. @EnableAutoConfiguration
* Chú thích này được sử dụng trên main application class.
* @EnableAutoConfiguration ngầm định nghĩa 1 gói tìm kiếm cơ sở.
* Chú thích này chỉ cho Spring boot bắt đầu thêm các bean dựa trên cài đặt classpath, các bean khác, các thiết lập property khác nhau.
  1. @SpringBootApplicaion
* Chú thích này được sử dụng trên các lớp ứng dụng trong khi thiết lập dự án Spring boot.
* Lớp được chú thích @SpringBootApplication phải được giữ trong các package cơ sở.
* @SpringBootApplication không là một thành phần quét nhưng nó quét các package phụ của nó.
  1. @RequestMapping
* Chú thích này được sử dụng trên cả phương thức và class.
* @RequestMapping sử dụng để khi có request sẽ ứng với các lớp xử lí cụ thể và phương thức xử lí
* @RequestMapping được sử dụng trên lớp, nó sẽ tạo ra 1 URI cơ sở mà controller sử dụng.
* @RequestMapping sử dụng trên phương thức, nó sẽ cung cấp cho 1URI mà phương thức đó sẽ xử lí thực hiện.
* @GetMapping: Sử dụng để ánh xạ request HTTP GET vào phương thức xử lí cụ thể. Viết tắt của @RequestMapping(method = requestmethod. Get)
* @PostMapping: Sử dụng để ánh xạ request HTTP POST vào phương thức xử lí cụ thể. Viết tắt của @RequestMapping(method = requestmethod. Post)
* @PutMapping: Sử dụng để ánh xạ request HTTP PUTvào phương thức xử lí cụ thể. Viết tắt của @RequestMapping(method = requestmethod.Put)
* @PatchMapping: Sử dụng để ánh xạ request HTTP PATCH vào phương thức xử lí cụ thể. Viết tắt của @RequestMapping(method = requestmethod. Patch)
* @DeleteMapping: Sử dụng để ánh xạ request HTTP DELETE vào phương thức xử lí cụ thể. Viết tắt của @RequestMapping(method = requestmethod.Delete)
  1. @CookieValue
* Chú thích này được sử dụng ở tham số của phương thức.
* Sử dụng như tham số của các phương thức xử lí request
* HTTP Cookie bị ràng buộc tham số cho tên cookie nhất định.
* Chú thích này thường được sử dụng với các phương thức có chú thích @RequestMappring
  1. @CrossOrigin
* Chú thích này được sử dụng ở lớp và phương thức.
* Trong trường hợp các máy chủ sử dụng JS khác nhau truy cập đến máy chủ phục vụ lưu trữ dữ liệu, CORS cho phép hoặc ngăn chặn giao tiếp qua các miền. Để bật giao tiếp này, thêm chú thích @CrossOrigin
* Mặc định, cho phép tất cả nguồn gốc, tiêu đề, phương thức Http được chỉ định và maxage là 30 phút. Có thể điều chỉnh bằng cách xác định các giá trị thuộc tính tương ứng
  1. @ExceptionHandler
* Được sử dụng ở phương thức để xử lí ngoại lệ của controller
* Được sử dụng để xác định class ngoại lệ nó sẽ bắt
* Có thể sử dụng trên các phương thức gọi khi xử lí 1 ngoại lệ.
* Các giá trị được đặt thành 1 mảng các loại ngoại lệ
* Nếu một ngoại lệ được catch phù hợp với 1 trong các loại trong mảng, phương thức đó sẽ được gọi.
  1. @PathVariable
* Sử dụng để chú thích yêu cầu tham số của phương thức.
* @RequestMapping có thể sử dụng để xử lí các thay đổi động trong URI, trong đó 1 số giá trị trong URI là tham số. Có thể chỉ định tham số này bằng 1 kí hiệu. Chú thích @PathVariable sử dụng để định nghĩa tham số này.
  1. @RequestBody
* Sử dụng để chú thích yêu cầu tham số phương thức
* Chỉ ra rằng 1 tham số phương thức phải được ràng buộc với giá trị và chuyển đổi từ request sang đối tượng
  1. @RequestHeader
* Sử dụng để chú thích yêu cầu tham số của phương thức
* Chỉ ra rằng yêu cầu giá trị header.
* Khi 1 request đến, nó kiểm tra header với tên được chỉ định và gắn giá trị và tham số của phương thức. Nó giúp có được chi tiết tiêu đề trong controller
  1. @ResponseBody
* Sử dụng để chú thích các phương thức xử lí yêu cầu
* Chỉ ra rằng loại kết quả được viết thẳng trong cơ chế phản hồi ở bất kì định dạng nào chỉ định (JSON, XML, …)

1. File cấu hình

* Được sử dụng để giữ nhiều thuộc tính trong 1 file duy nhất để chạy trên các môi trường khác nhau
* Nó lưu trữ các cặp key – value đơn giản cho các thuộc tính cấu hình.
* Có thể gói các tập tin cấu hình trong file jar hoặc đặt các tập tin trong file hệ thống của môi trường chạy và tải nó vào khi khởi động
* Vị trí mặc định: src/main/resources

1. Profiles

* Profiles cung cấp một cách để phân biệt phần của cấu hình ứng dụng và làm cho nó chỉ có sẵn trong môi trường nhất định.
* Nó có thể được đánh dấu để hạn chế khi nó được tải
* Có thể sử dụng thuộc tính spring.profiles.active để chỉ định hồ sơ nào hoạt động
* Có thể chỉ đinh cấu hình active profiles trong application.properties và sau đó thay thế nó sử dụng command line