Mục Lục

[**1.** **Giải thích :** 1](#_Toc510745881)

[**2.** **Công Dụng :** 1](#_Toc510745882)

[**3.** **Cách Dùng :** 1](#_Toc510745883)

[**4.** **Phân Biệt :** 3](#_Toc510745884)

[**5.** **Bean tự khai báo :** 3](#_Toc510745885)

[**6.** **Thực hành :** 4](#_Toc510745886)

[**7.** **Dùng các bean có sẵn : upload, email.** 5](#_Toc510745887)

1. **Giải thích :**

**\_** Ta viết ra các class, class A sử dụng class B => class B phụ thuộc vào class A. class B sẽ có nhiệm vụ là lưu trữ thông tin gmail, giả sử ta muốn class B nhận nhiệm vụ lưu trữ thông tin yahoo mail hoặc 1 mail khác không trùng khớp định dạng. Vậy ta phải chỉnh sửa class A. Ta muốn dễ dàng thay đổi B mà khu class A vẫn sử dụng bình thường => Đây là lập trình độc lập ( không phụ thuộc ).

**\_** Đầu tiên người ta dùng interface để làm đối số. Khi class A cần chỉ việc implement. Cách lập trình như thế nào gọi là DI. Và để làm được việc này ta cần có **IoC Container** ( môi trường Inversion Of Control ). Tùy framework sẽ hỗ trợ DI khác nhau. DI là truyền đối tượng vào thành phần khác để sử dụng ( class, hàm, … )

**\_** Khi ta tiêm vào và sử dụng lưu ý chỉ tạo ra 1 object duy nhất ( singleton )

\_ Dùng chung 1 vùng nhớ, có thể tiêm cho mọi chỗ.

1. **Công Dụng :**

\_ Bean : khai báo, tiêm với @Autowire, phân biệt với @Qualifier.

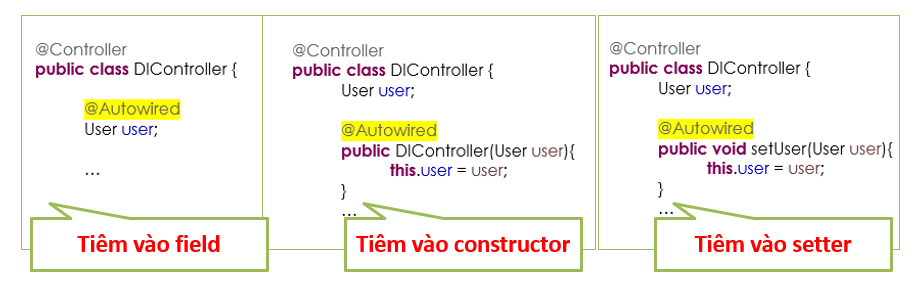
\_ Bean có sẵn :

+ CommonsMultipartResolver để upload file.

+ JavaMailSender để gửi mail

1. **Cách Dùng :**

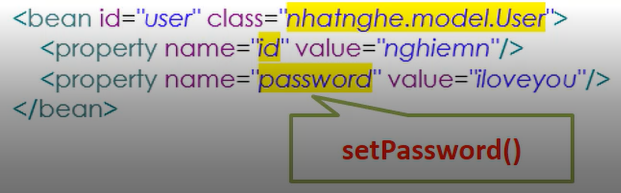
**Các cách tiêm bean vào controller :**

* Tiêm vào field.
* Tiêm thông qua constructor.
* Dùng setter.

**B1 :**

**\_** Định nghĩa 1 bean ( 1 class có **getter/setter** ).

\_ Khai báo vào file cấu hình thông qua thẻ bean.



\_ Hệ thống sẽ tạo cho ta đối tượng user có id và password tương ứng. Dù class=”” có thay đổi thì user của ta vẫn không ảnh hưởng => DI.

**B2 :** Tiêm vào và sử dụng đối tượng :

Cú pháp :



Lưu ý : bên trong bean nên khai báo **class.** Còn tiêm vào sử dụng nên khai báo **interface.**

VD : Bean khai báo là ArrayList còn khi tiêm vào sử dụng => dùng List. Vì list là interface và không phụ thuộc vào class.

Hệ thống sẽ lấy bean trong file XML và tiêm vào đây.

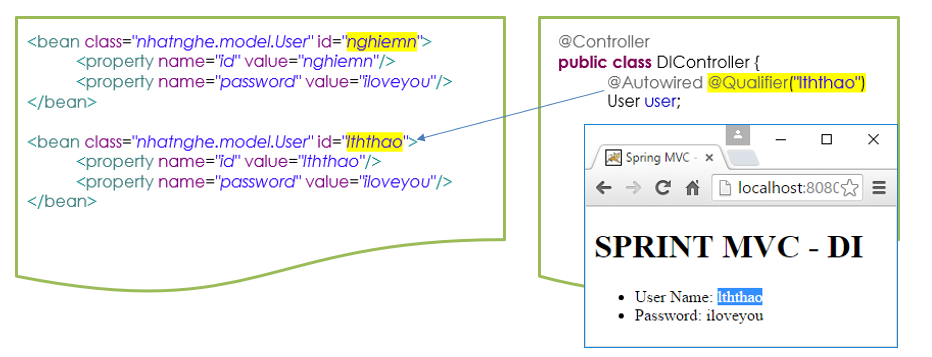
\*\* **Lưu ý :** khi ta tiêm vào nó chỉ tạo ra 1 object duy nhất. ( Singleton ) – 1 instance duy nhất. Thông tin dùng chung.

1. **Phân Biệt :**

**Để @Autowired làm sao biết lấy bean nào ?**

Nhận biết thông qua kiểu dữ liệu . Nếu trùng kiểu dữ liệu ta dùng đến **ID**.

Từ khóa : **@Qualifier**



1. **Bean tự khai báo :**

**\_** Loại : **Component, Controller, Repository, Service.**

**\_** Component : tên chung. Bean bình thường.

\_ Controller : là component, nhiệm vụ của nó là controller.

\_ Repository : lưu trữ dữ liệu, xử lý. => Sử dụng DAO

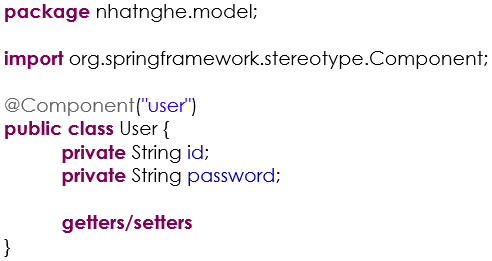
\_ Service : xử lý nghiệp vụ.

\_ Không phân biệt, giống nhau. Chỉ khác nhau về mặt ngữ nghĩa, cho dễ hiểu.

\_ Đây là bean tự khai báo trong spring config xml.

\_ Bean này không có property và các giá trị.

\_ Ta cần khai báo để sử dụng thông qua từ khóa **@Component**



\_ Ta cần chỉ ra **package** đang chứa bean tự khai báo để sau này gọi @**Autowired** để sử dụng. Chỗ khai báo controller, cách nhau dấu phẩy. Nơi quét các component.



Mẹo : nghiepuit.controller, nghiepuit.service ta có thể tóm gọn thành nghiepuit. Nhưng mất thời gian hơn ! Chỉ cụ thể quét sẽ nhanh. Quá trình quét xảy ra khi Dispatcher Servlet init. Chỉ quét lúc khởi động.

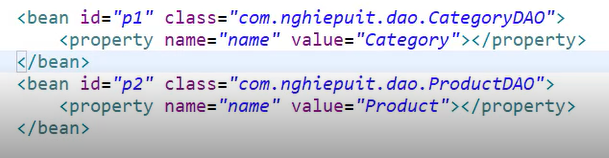
1. **Thực hành :**

\_ Tạo bean 2 cách XML : chứa các phương thức, sau đó tiêm vào controller và sử dụng. ( log xem ).

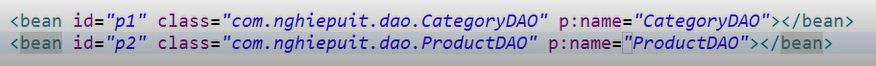
+ Tạo 2 DAO implement interface DBAction ( insert, update, delete ) sau đó ta tiêm interface thay vì class. Sau đó sẽ thấy lỗi và sử dụng id để phân biệt.

+ Tạo bean có property :

**C1 :**

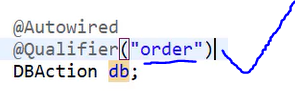


**C2 :**



Sau đó bên trong bean – cần có thuộc tính name và mỗi phương thức ta show log ra xem.

Tại controller :

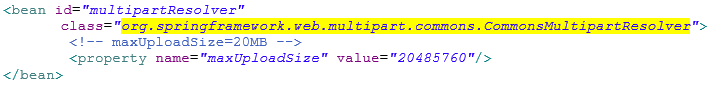
Có thể order, product, category, tùy…

Ứng dụng : Tạo các thư viện để xài chung.

1. **Dùng các bean có sẵn : upload, email.**

**Upload :**

B1 : Cấu hình XML ( spring-config-upload ) – kích thước tối đa : đơn vị **byte**

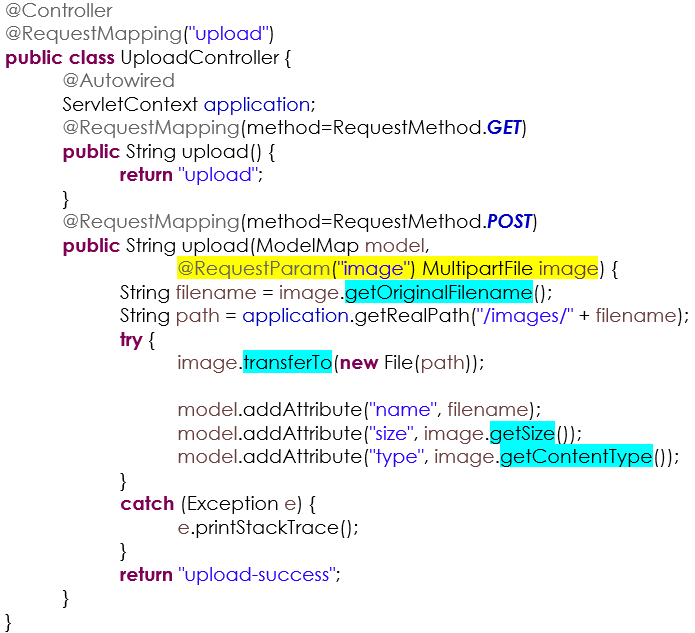
****

* Bean này có setter là setMaxUploadSize và nhiều phương thức khác.

B2 : Form : **POST**, enctype : **multipart/form-data,** input type là file có name.

B3 : Controller ( dùng RequestMethod là POST, dùng @RequestParam lấy tham số image )

Sử dụng MultipartFile ( phải có tiêm CommonsMultipartResolver )

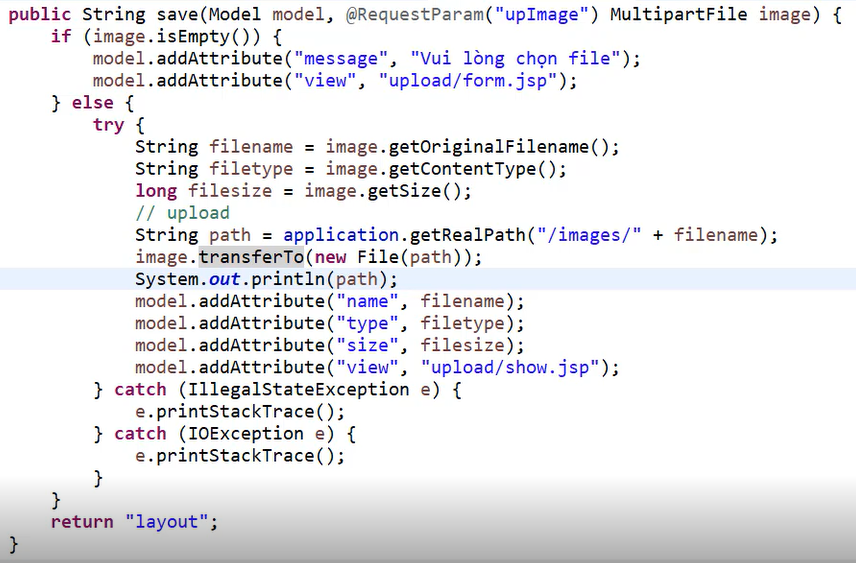


\_ Khi upload file từ khách hàng lên server ta phải đổi tên ( bảo mật ) : getOriginalFilename(); để lấy tên file gốc.

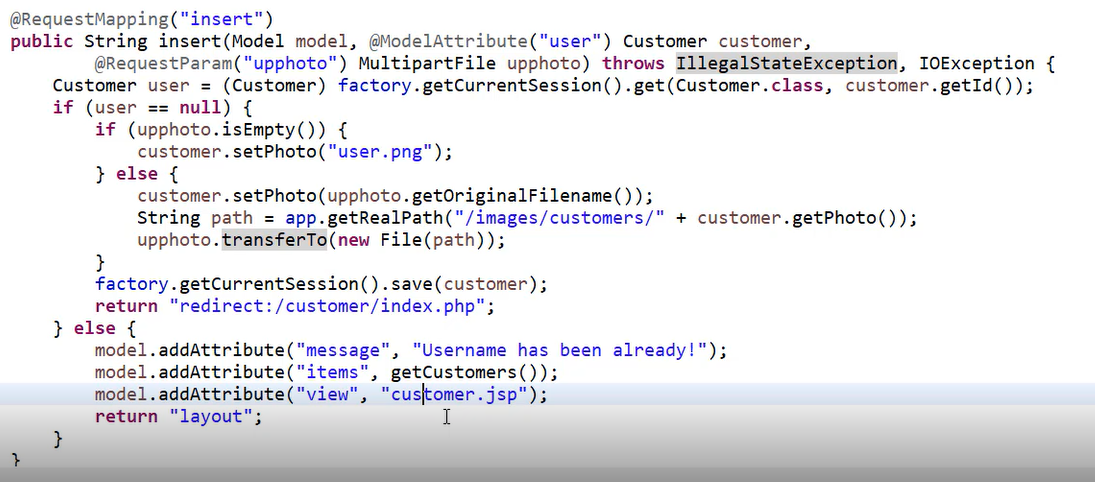
\_ Sử dụng ServletContext ( application – tiêm bằng @Autowired ) : để lấy đường dẫn tính từ website : app.getRealPath(“/Thư-Mục/” + filename)

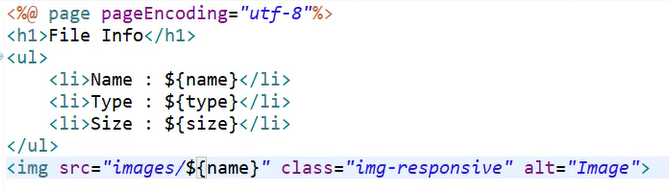
\_ Upload : image.transferTo(new File(path))

**Mẹo :** gõ CommonMultipartResolver bên java để lấy tên bean đầy đủ. Truy xuất đến phương thức – tuân thủ theo java bean để lấy property. Bỏ getter/setter – chuyển ký tự đầu thành thường.



Bổ sung nếu không tồn tại lấy hình ? ngược lại upload.





**Mail Bean: JavaMailSender**

B1 : cấu hình XML ( spring-config-gmail )

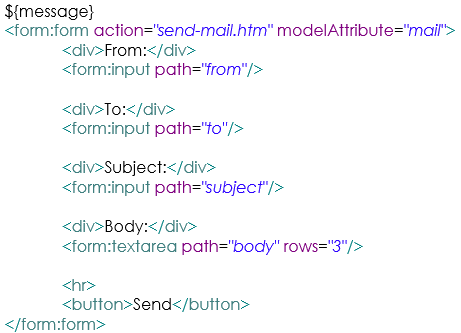


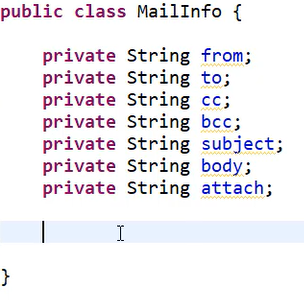
JavaMailSenderImpl : là class. Ta sẽ tiêm vào controller 1 interface để dùng.

Khi deploy bỏ debug.

Lưu ý :

\_ Tài khoản đăng nhập : dùng tk gg, để đăng nhập, bỏ thư vào, không phải tài khoản người gửi. Đây là **bưu điện**. Lưu ý phải kích hoạt cho phép gửi mail.





Gửi mail : nhờ vào helper. Message là 1 tờ giấy.



\_ From : người gửi

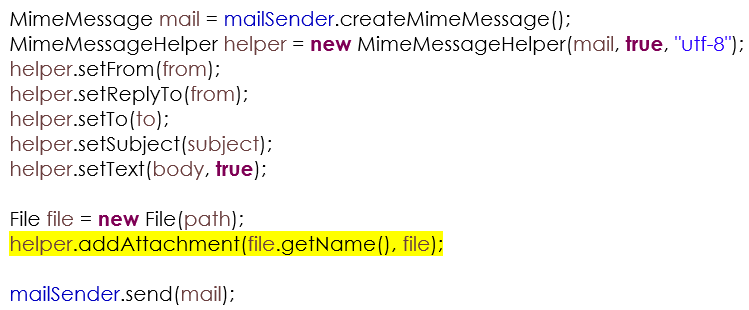
\_ To : người nhận

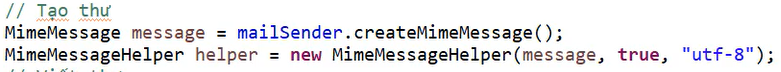
\_ ReplyTo : khi bấm reply trên mail thì gửi ngược lại cho ai.

\_ Subject : tiêu đề

\_ Text : nội dung

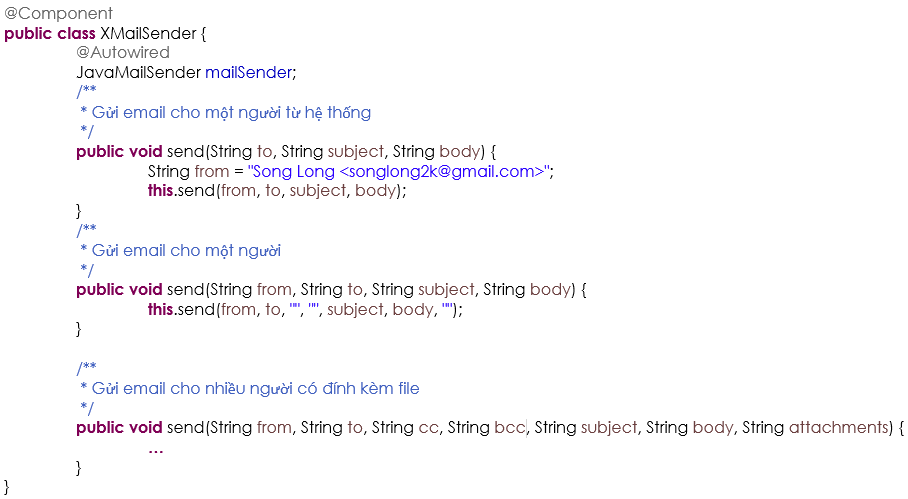
Attach file :



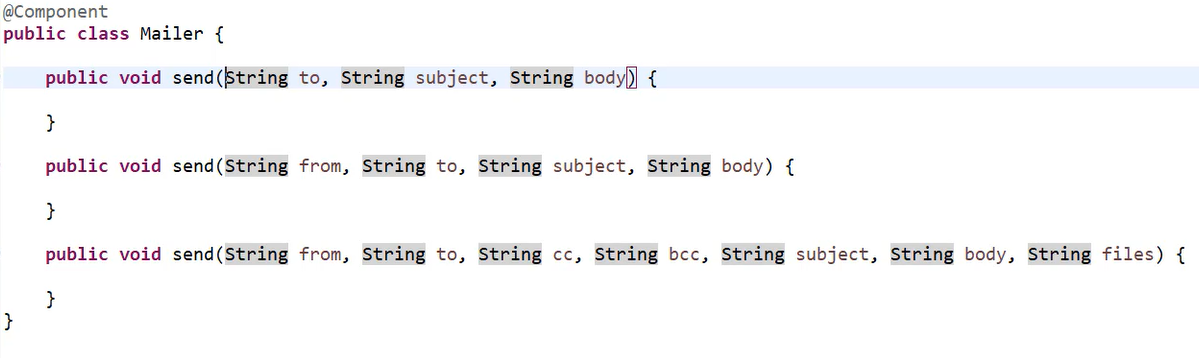




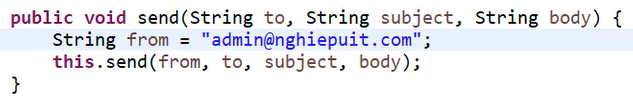
Tách class riêng để thành thư viện : ( Component )



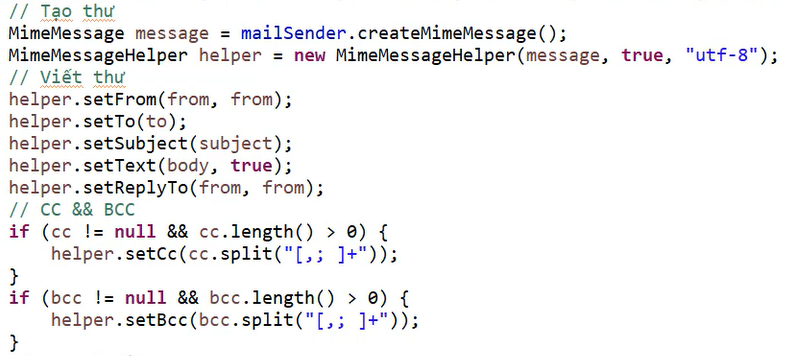


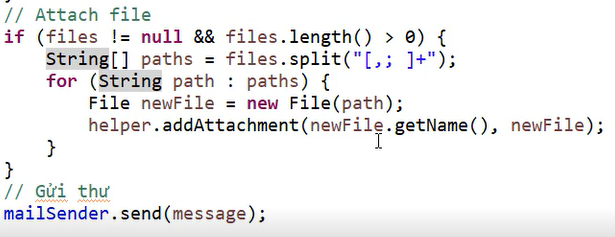


Hàm 1 : Gọi hàm 2



Hàm 3 :





Chú ý : các file cách nhau , hoặc ; hoặc dấu cách.

Hàm 2 :

Gọi hàm 3 . Nếu thiếu để null.

**Sử dụng :**

B1 : Khai báo Mailer là 1 component tại spring-config-mvc vào component-scan.

B2 : Tiêm bean do mình tạo **Mailer** .