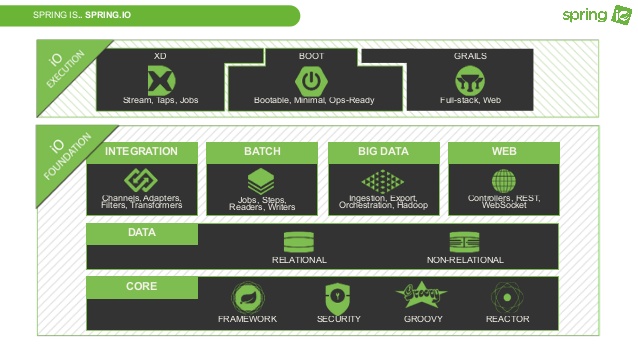
# Tổng quan

## Spring Boot là gì?

Nếu đã từng lập trình với các Module của Spring Framework như MVC, hibernate, security, … chúng ta thấy rõ ràng 1 điều rằng chúng ta phải cấu hình rất nhiều. Từ các file xml đến các class, interface, … Và việc phải nhớ các chi tiết để thiết lập cấu hình đôi khi thực sự là một việc nhàm chán vì nó thường lặp đi lặp lại, đó cũng có thể được coi là nhược điểm của các module đó.

Spring Boot ra đời để giải quyết nhược điểm vừa được nêu trên, đồng thời cũng cung cấp 1 hướng phát triển phần mềm trong tương lai.

Khi so sánh kiến trúc của các module khác với Spring Boot, chúng ta sẽ thấy phần cấu hình được đẩy lên layer cao hơn. Rồi cung cấp xuống phía dưới các interface đơn giản hơn cho việc cấu hình nhưng vẫn đầy đủ những gì mà các module khác có.



## Các tính năng nổi bật

* Đóng gói và chạy ứng dụng với file JAR nhưng vẫn có thể build project ra file WAR để deploy lên server.
* Free Marker.
* Cấu hình ít, tối ưu hóa tự động cấu hình, không cần thiết phải cấu hình với xml.
* Hỗ trợ nhiều plugin.
* Cung cấp class SpringApplication với method static “run” để chạy ứng dụng từ hàm main.
* Cấu hình bên ngoài với YAML, file Properties, các biến môi trường và parameter inject trực tiếp với annotation @Value hoặc với @ConfigurationProperties, …
* Profiles.
* Kết hợp mượt mà với các module khác.
* …

## Tham khảo

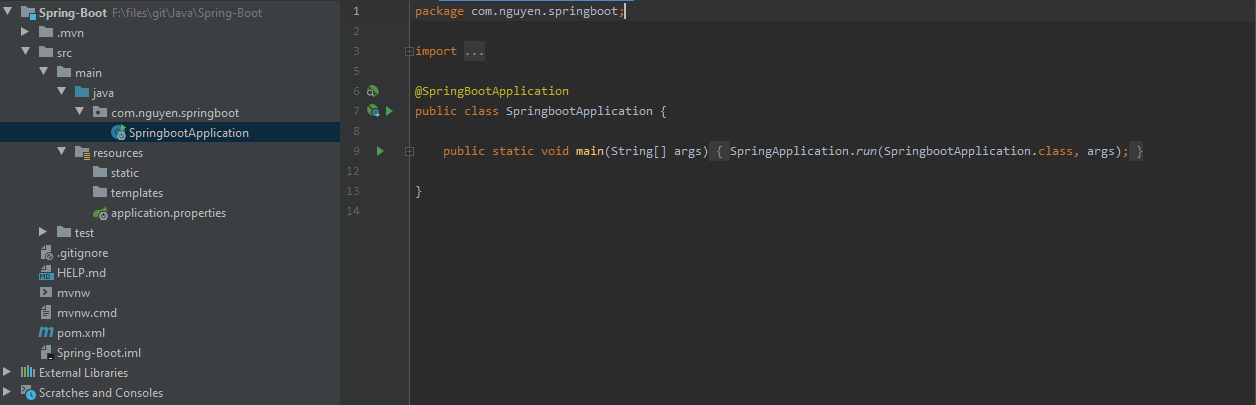
# Khởi động

## Tạo dự án đầu tiên

### Triển khai dependency Spring Web trong file pom.xml



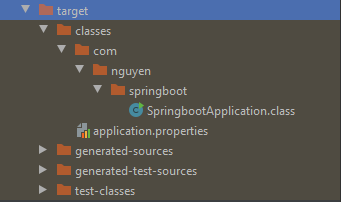
### Cấu trúc dự án và class SpringApplication



## Nói thêm về cấu trúc

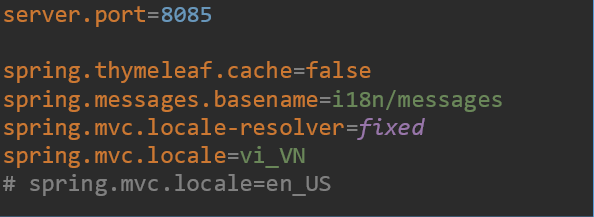
Nếu đã từng làm việc với Spring MVC thì có lẽ sẽ không còn xa lạ gì lắm với cấu trúc này.

* Chúng ta có src/main/java nơi chứa các packages và file .java.
* Cũng có thư mục test nơi thực hiện các test case (tôi cũng đã thực hiện 1 bài viết cơ bản về chủ để này với Junit4: Tham khảo tại đây nếu có nhu cầu).
* Chúng ta cũng có thể sẽ thấy thư mục target nơi chứa các file đã được biên dịch như sau:



* Tài nguyên được đặt tại thư mục resources bao gồm:

1. Static, nơi chứa nguồn tài nguyên tĩnh như các file javascript, css, …
2. Templates, nơi chứa file view như jsp, html hoặc layout.
3. Application.properties, nơi thiết lập cấu hình dự án với các cặp key và value và dưới đây là ví dụ minh họa nội dung:



1. Ngoài ra, đối với yêu cầu của 1 số dự án, chúng ta cũng cần tạo ra những thư mục riêng chỉ định cho nguồn tài nguyên mà dự án đó yêu cầu, ví dụ như i18n chứa các messages phục vụ cho yêu cầu đa ngôn ngữ.

Cũng còn 1 số file khác như gitignore giúp git bỏ qua và không theo dõi các file có mẫu xác định, HELP.md, mvnw, … cũng không cần thiết phải quan tâm ngay lúc này ngoại trừ pom.xml là bắt buộc hiểu để có thể làm việc với một dự án maven.

## Kịch bản

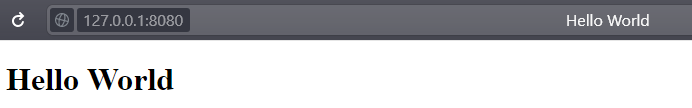
Tôi có 1 file index.html với nội dung “Hello World”. Một FontController sẽ handler tất cả các request từ trình duyệt đến server 127.0.0.1:8080 và triệu gọi Controller tương ứng để nhận request có dạng / sau đó trả về 1 logical view “index”, viewResolver sẽ nhận và triệu gọi physical view tương ứng là index.html thực hiện render và gửi response lại trình duyệt.

Nếu thấy khó hiểu, có thể bạn cần xem lại luồng thực thi và các component của SpringMVC tại đây. Thực chất là không cần phải hiểu như vậy, tôi nêu ra để bạn có ý niệm về nó hoặc nếu bạn đã biết, thật tốt nếu nó giúp bạn ôn lại phần nào đó kiến thức.

Nói chung, request có dạng 127.0.0.1:8080 sẽ trả về nội dung file index.html trong thư mục template có đường dẫn tương đối /resources/templates/index.html.

Tôi xem như bạn đã có kiến thức cơ bản nền tảng về Spring MVC, nếu bạn cần tham khảo vui lòng click vào đây. Đây là hướng dẫn step-by-step nhưng sẽ không giải thích về các thành phần vì tôi mặc định bạn đã biết.

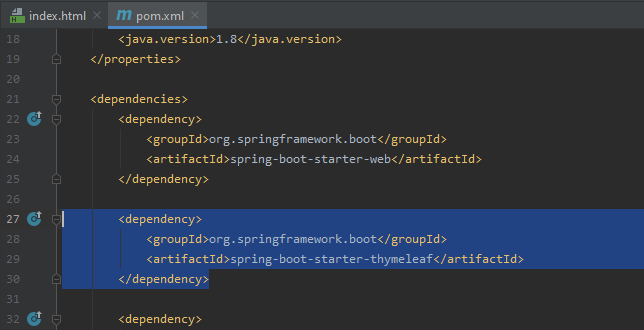
### Kết quả



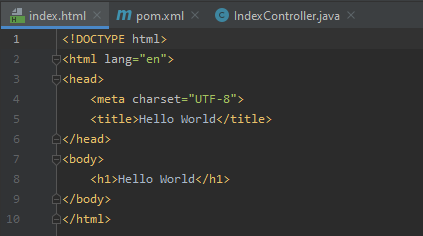
### Dependency thymeleaf

Tôi sẽ trình bày thymeleaf cơ bản trong 1 tài liệu khác, bạn có thể đọc nó tại đây. Về cơ bản nó là một java template để làm việc với html, xml, javascript, … Ở đây chúng ta dùng nó để resolver view.

Tại file pom.xml thêm dependency sau:



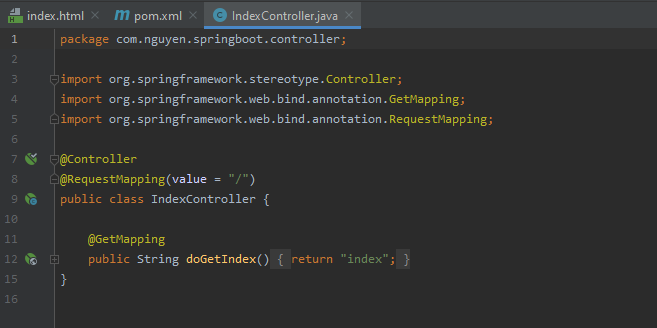
### Tạo view index



Bạn biết nó phải được đặt ở đâu rồi đúng không? Tạo file index.html trong thư mục /resources/templates với nội dung là tùy ở bạn, nội dung file này sẽ được trả về khi có request mapping với URI / được gọi.

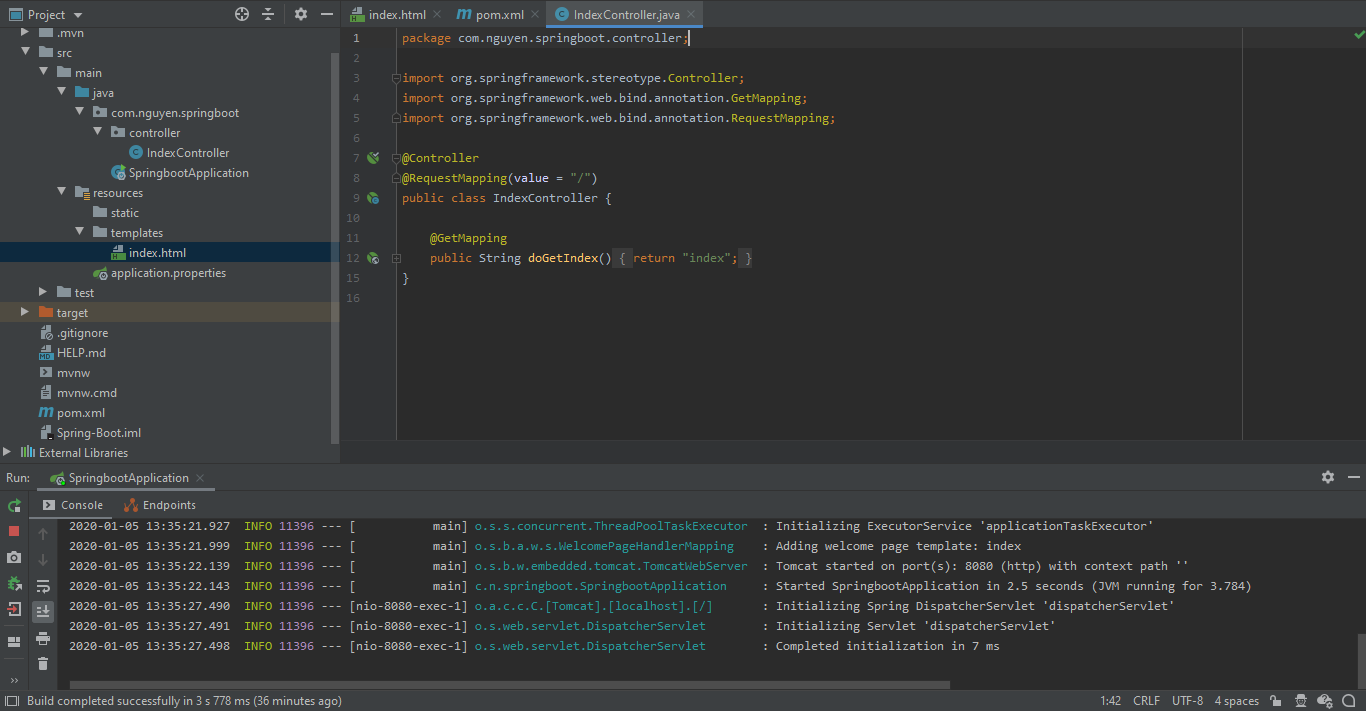
### Tạo controller index

Cũng giống như SpringMVC chúng ta có IndexController như sau:



### Thực thi

Build và Run Project sau đó mở 1 trình duyệt bất kỳ và truy cập với URL 127.0.0.1:8080 để xem kết quả.



Port 8080 là port mặc định mà chúng ta không cần phải cấu hình, để tùy chỉnh port, thêm giá trị sau vào file application.properties (server.port=number port) với number port là số của cổng mà ta muốn mở, ví dụ server.port=9000.

## Lời kết

Ta đã có chương trình “hello world” đầu tiên với Spring Boot. Không quá phức tạp cũng không phải cấu hình quá nhiều như Spring MVC. Nếu bạn có nhớ, trong Spring MVC chúng ta phải cấu hình 1 số file như web.xml hay servlet-context.xml thì ở đây không có những file đó.

## Tham khảo

# Thao tác với các thành phần

## Spring Boot Starters

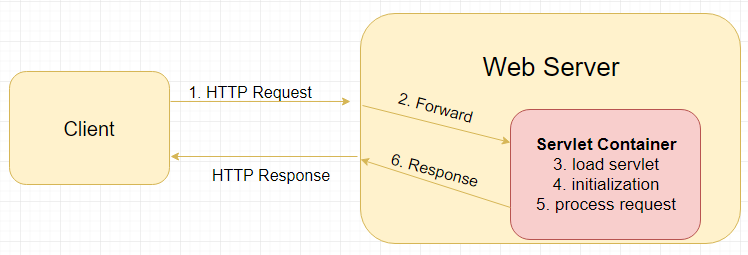
Tôi định bắt tay ngay vào phần thao tác với file application.properties, nhưng tôi nghĩ bạn cần biết sâu thêm 1 chút nữa về Spring Boot và cách mà nó hoạt động. Vì thế nên tôi dừng lại và viết ra phần này. Bạn có thể bỏ qua và đi đến phần tiếp theo nếu không có nhu cầu muốn hiểu sâu hơn.

Spring Boot Starters là một tập các mô tả phụ thuộc mà ta có thể include vào trong application của mình. Ta có thể lấy “một cho tất cả” các phụ thuộc liên quan mà không cần phải tìm kiếm và copy như thông thường vẫn làm. Nói chung, nó giúp chúng ta tự động quản lý các dependencies và tự động cấu hình.

Ở ví dụ chương 2, nếu bạn có để ý sẽ thấy chúng ta đã thêm 1 denpendency là springboot-starter-web nó bao gồm các thư viện để phát triển ứng dụng Spring MVC như spring-webmvc, jackson-json, validation-api, tomcat. Chúng ta không cần phải thêm chúng vào file pom.xml. Ngoài ra chúng ta cũng không cần cấu hình các bean như DispatcherServlet, ResourceHandler, MessageSource, …

Trong quá suốt tài liệu này, bạn có thể thấy các Spring Boot Starter thể hiện qua các dependencies dạng spring-boot-starter-\*.

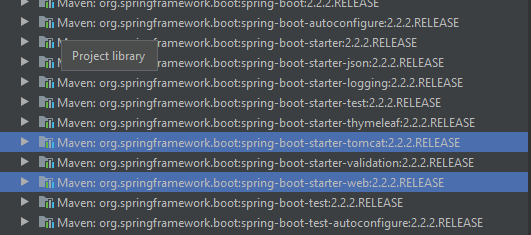
### Servlet Container



Hình trên giúp bạn gợi nhớ về Servlet Container, một khái niệm cơ bản trong java-ee.

Sau khi chạy ứng dụng tại class có đánh dấu với annotation @SpringApplication thì chúng ta có thể truy cập ứng dụng với URL 127.0.0.1:8080. Và để làm được điều đó, chúng ta phải có Servlet Container. Spring Boot đã nhúng sẵn cho chúng ta. Và nhúng bằng cách nào?

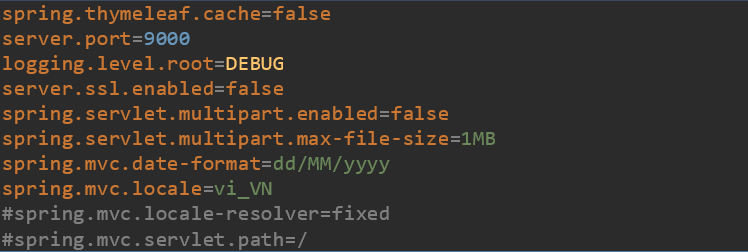
Khi thêm dependency spring-boot-starter-web thì spring-boot-starter-tomcat cũng được tự động thêm vào, và khi khởi động ứng dụng, tomcat được nhúng bên trong cũng sẽ được khởi động vì vậy ta không cần phải deploy ứng dụng lên bất cứ server nào. Điều này đôi khi là lý do thực sự làm những lập trình viên làm việc với Spring Boot yêu thích nó.



## File application.properties

Như chúng ta cũng biết, Spring Boot đã cấu hình sẵn hầu hết mọi thứ, tuy nhiên chúng ta vẫn có thể tùy chỉnh các thông số cấu hình thông qua file application.properties.

Ở đây tôi sẽ đề cập đến một số thuộc tính cấu hình mà có lẽ chúng ta sẽ gặp nhiều khi thao tác và làm việc với Spring Boot, ngoài ra sẽ còn rất nhiều thuộc tính nữa và nếu bạn có nhu cầu thì có thể tìm hiểu tại đây.



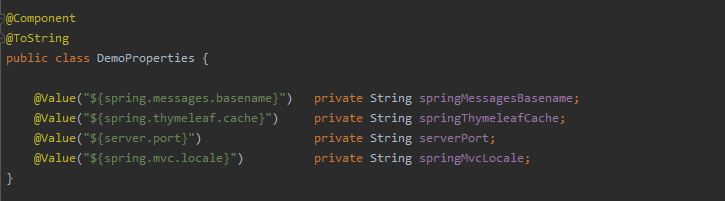
Mở chú thích bằng cách bỏ dấu # ở đầu mỗi dòng. Tôi để những thuộc tính cấu hình thường dùng là “bật”, và tất nhiên bạn có thể chú thích nó nếu muốn. Toàn bộ thuộc tính mà tài liệu này sử dụng được tổng hợp tại đây.

## Làm việc với file properties

### File application.properties

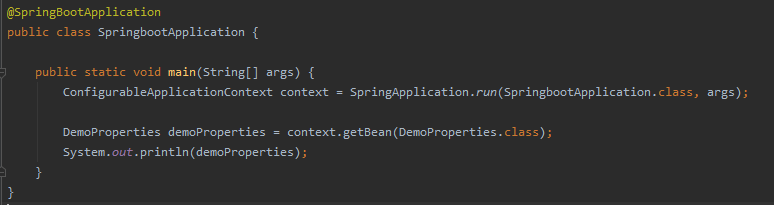
Để đọc các giá trị từ file application.properties, ta có thể sử dụng annotation @Value với cú pháp như sau @Value(“${key}”), trong đó key là khóa mà chúng ta muốn đọc vào giá trị.

Chúng ta xây dựng một class như sau:



Các giá trị tương ứng với các key bên trên trong file application.properties sẽ được inject tự động vào các biến tương đương đã được khai báo.

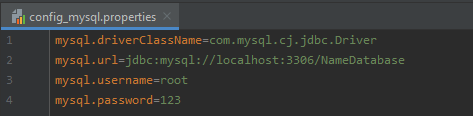
Để demo kết quả, tôi chỉnh sửa lại phương thức main trong class được đánh dấu @SpringBootApplication như sau:



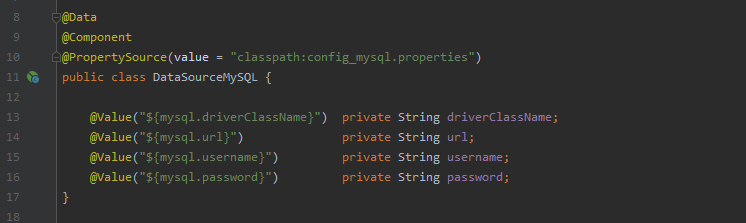
Kết quả là một chuỗi các giá trị được in ra dưới màn hình console.

### File properties bất kỳ

Tôi đã hướng dẫn bạn đọc file properties với annotation @Value như ở trên, ở phần này, tôi sẽ giới thiệu đến bạn những thao tác khác khi làm việc với 1 file properties bất kỳ. Ta có file properties với nội dung như sau:

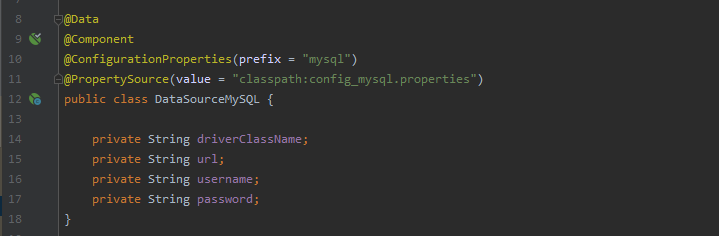


Để đọc những giá trị từ file này, cũng giống như cách ở trên nhưng thêm annotation @PropertySource để chỉ định tài nguyên cung cấp. Nếu thiếu nó, một exception IllegaArgumentException sẽ bị ném ra. Qua đó bạn thấy, nếu không chỉ định value cho annotation @PropertySource thì mặc định nó là application.properties hoặc application.yml.



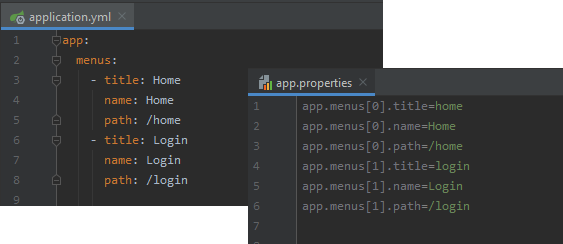
Đôi khi việc đánh dấu bằng các annotation như vậy sẽ làm mã trông rối và khó nhìn, bởi đơn giản về sau này có thể chúng ta cũng cần nhiều đánh dấu khác nữa, một ví dụ cho việc này là cấu hình mapping khi làm việc với jpa, hibernate, …

ConfigurationProperties sẽ đơn giản hóa phần nào cho chúng ta:



Nó sẽ tự động mapping các thuộc tính có key trùng với tên thuộc tính trong class, kết quả ta có 1 class với các thuộc tính có giá trị như được khai báo trong file properties.

Với những dạng properties phức tạp như:



Annotation ConfigurationProperties cũng giúp chúng ta tự động mapping các thuộc tính như trên vào các collection. Riêng đối với file yml nếu bạn chưa biết nó là gì, vui lòng tìm hiểu tại đây.

Cũng nhắc lại, nếu không chỉ định value cho annotation PropertySource mặc định các thuộc tính sẽ được dò tìm trong các file application.properties và application.yml, hình dưới minh họa cho việc đọc menus và lưu vào collection List.



## Đôi lời

Dừng lại 1 chút ở đây, như yêu cầu để đọc tài liệu này, bạn phải có kiến thức nền tảng với Spring MVC, điều đó giúp tôi và bạn tiết kiệm rất nhiều thời gian cho những thao tác giống nhau giữa 2 module.

Tôi không đề cập đến các annotation như RequestMapping, GetMapping, ModelAttribute, RequestParam, PathVariable, Valid, PostConstruct, Component, Service, Configuration, Bean, … và còn rất nhiều thứ khác nữa. Nếu bạn chưa biết về chúng vui lòng tìm hiểu tại đây.

Tôi cũng sẽ trình bày những chương mà tại đó, cũng yêu cầu bạn phải có kiến thức cơ bản với Hibernate, Spring Data JPA và Spring sercurity. Vui lòng tìm hiểu tại một số nguồn khác nếu gặp khó khăn khi bắt đầu.

## Tham khảo

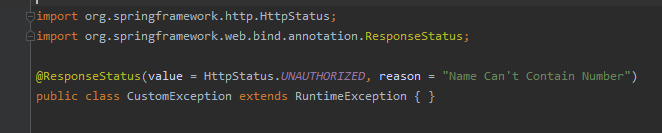
# Xử lý Exception

## Tổng quan

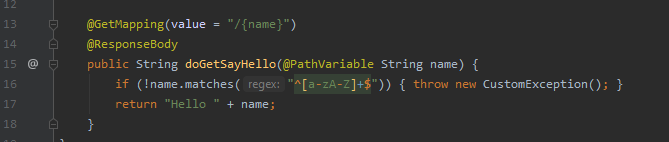
Đây là thao tác để bắt các exception được ném ra trong Spring Framework, nó không chỉ chỉ riêng Spring Boot. Tôi giành 1 chương để giới thiệu những thao tác cơ bản nhất với nội dung này, giúp cho bạn có thể tra cứu nhanh khi cần dùng tới.

## Annotation ResponseStatus

Chỉ định HttpStatus kèm theo exception.

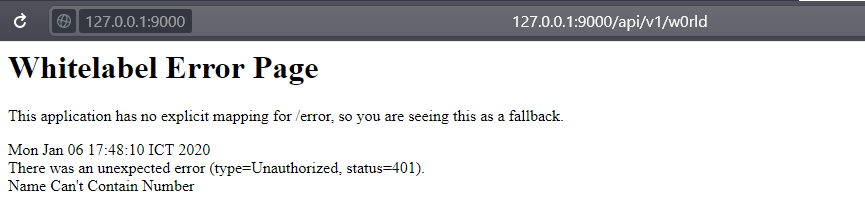
Tôi định nghĩa 1 exception như sau:

Chú ý tới annotation ResponseStatus, nó định nghĩa HttpStatus cùng message đi kèm mỗi khi 1 exception bị server ném ra.

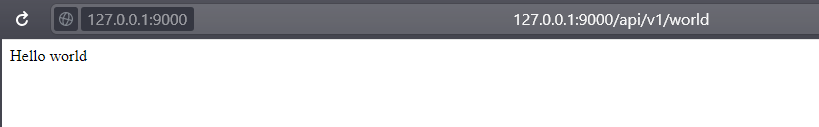


Hình minh họa phía trên là 1 method kiểm tra PathVariable, nếu nó chứa số, exception CustomException sẽ được ném ra. Ngược lại hiển thị dòng text “Hello <name>”.

Đây là kết quả cho lần thử có name = w0rld:



Và hình phía dưới là kết quả với name = world

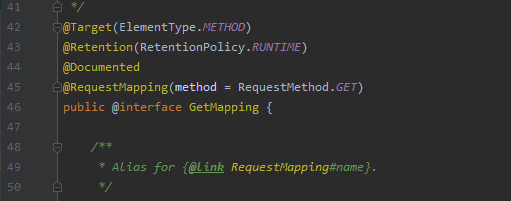


## Annotation ExceptionHandler

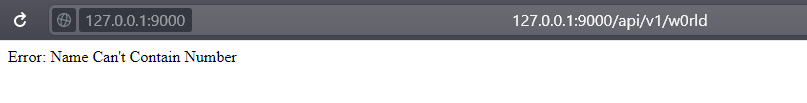
Tôi xin trích lại nguyên văn document về annotation ExceptionHandler, nội dung của nó như sau: “Annotation for handling exceptions in specific handler classes and/or handler methods. Handler methods which are annotated with this annotation are allowed to have very flexible signatures.”



Qua đoạn mã phía trên, bạn sẽ thấy annotation ExceptionHandler đánh dấu method HandlerRuntimeException sẽ xử lý exception RuntimeException khi nó được ném ra từ các class được đánh dấu RequestMapping (@GetMapping được xây dựng dựa trên RequestMapping để nó trông ngắn gọn hơn).



Sau đây là kết quả với name = w0rld:



## Annotation ControllerAdvice và RestControllerAdvice

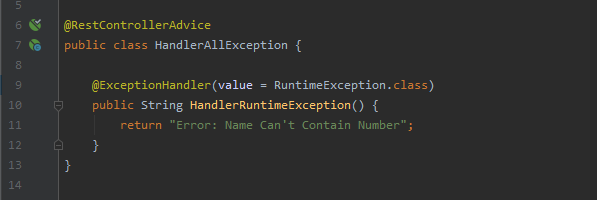
Tôi đã trình bày 2 thao tác:

1. Thứ nhất, đặt annotation ResponseStatus tại class định nghĩa Exception để chỉ định HttpStatus và message được đưa ra khi xảy là exception tại mục [4.2].
2. Thứ hai, đặt annotation ExceptionHandler tại method để chỉ định method sẽ xử lý exception khi nó được ném ra bởi các method khác trong cùng 1 class tại mục [4.3].

Tại mục [4.4] này, tôi sẽ trình bày tiếp 1 thao tác để bắt tất cả các exception được ném ra từ tất cả các class hoặc 1 package được chỉ định cụ thể.

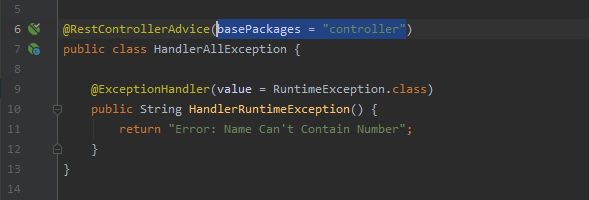
Annotation ControllerAdvice sẽ bắt exception từ class được đánh dấu với annotation Controller cũng thế RestControllerAdvice sẽ bắt exception từ class được đánh dấu với annotation RestController (@RestController = @Controller + @ResponseBody).

Hãy xóa bỏ phương thức HandlerRuntimeException ở phương thức main hiện tại và tạo ra 1 class HandlerAllException với nội dung như sau:



Hãy chạy thử để xem kết quả, và tất nhiên bạn cũng có thể định nghĩa thêm nhiều phương thức xử lý exception khác trong class này.

Để chỉ định 1 package cụ thể, hãy sử dụng thuộc tính basePackages như sau:



Nếu có nhiều hơn 1 packages hãy đặt chúng trong cặp dấu nhọn.

## Ví dụ tổng hợp

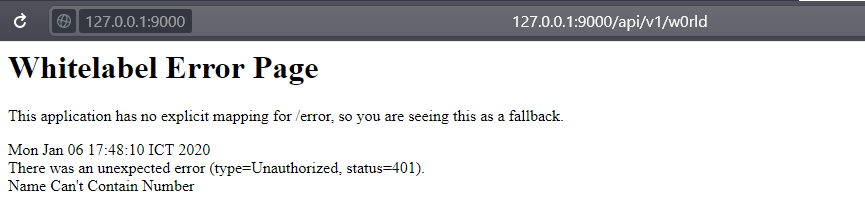
### Xác thực dữ liệu

Đầu tiên, quy tắc quan trọng của 1 ứng dụng web ở phía back-end là luôn phải xác thực dữ liệu được gửi lên từ người dùng. Dù cho có lẽ, bạn đã thực hiện việc xác thực đó bằng javascript ở phía front-end, nhưng người dùng luôn có quyền tắt javascript. Vì vậy, phải luôn xác thực.

Nếu bạn chưa biết các xác thực dữ liệu từ form trong Spring, vui lòng tham khảo thêm các nguồn thông tin về nó.

### Tùy chỉnh thông báo lỗi

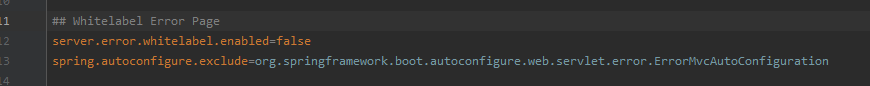
Tiếp theo ở ví dụ này, tôi sẽ chỉ cho bạn cách để tùy chỉnh các thông báo lỗi để nó trở nên thân thiện hơn với người dùng cuối. Thay vì như thế này:



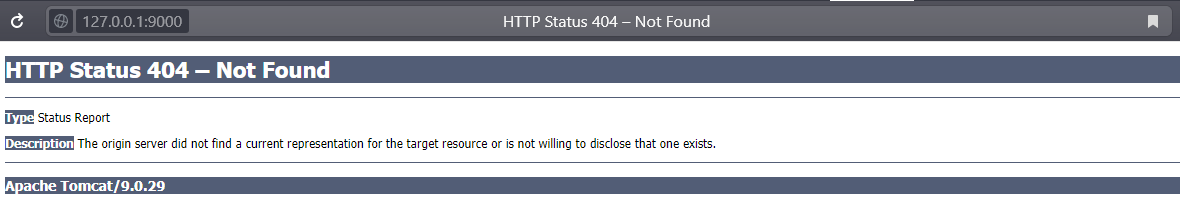
Cần lưu ý:

Khi không có trang error.html tất cả các exception bị ném ra từ hệ thống sẽ được trang hiển thị lỗi mặc định của Spring Boot xử lý nếu bạn không chủ động bắt nó bằng annotation ExceptionHandler, giao diện của nó như hình bên trên bạn đang thấy.

Chúng ta có thể tắt trang này đi và sử dụng thông báo lỗi mặc định của server (ở đây là tomcat) bằng cách thêm 2 dòng sau vào file cấu hình (application.properties):



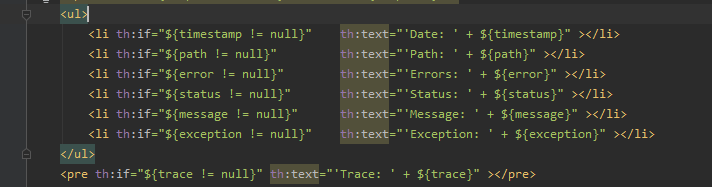
Khi đó, giao diện hiển thị exception sẽ như sau:



Để tùy chỉnh giao diện này, quyết định thông tin nào về exception sẽ được hiển thị với người dùng nếu bạn **KHÔNG BẮT EXCEPTION với annotation @ExceptionHandler**, hãy làm theo các bước sau:

Thứ 1: Tạo trang error (ở đây tôi dùng thymeleaf nên phần mở rộng là html), sẽ là error.jsp nếu bạn dùng jsp.

Thứ 2: Dùng các variable có sẵn sau để quyết định thông tin hiển thị:

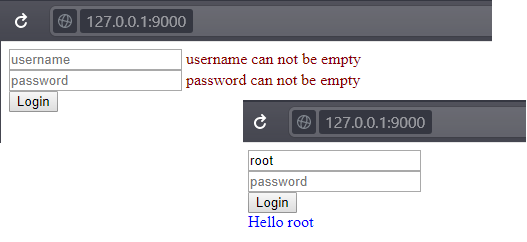


Tất nhiên, bạn có thể tùy chỉnh giao diện nếu muốn chứ không nhất thiết phải nằm trong thẻ ul và li như tôi.

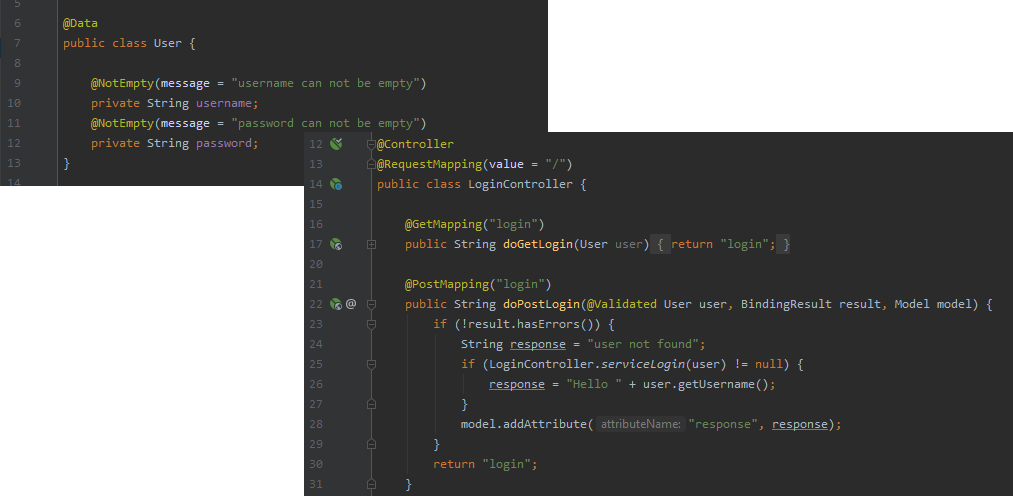
Lưu ý: Để hiển thị stacktrace vui lòng thêm giá trị sau vào file cấu hình (application.properties) “server.error.include-stacktrace=always”.

### Step – By – Step

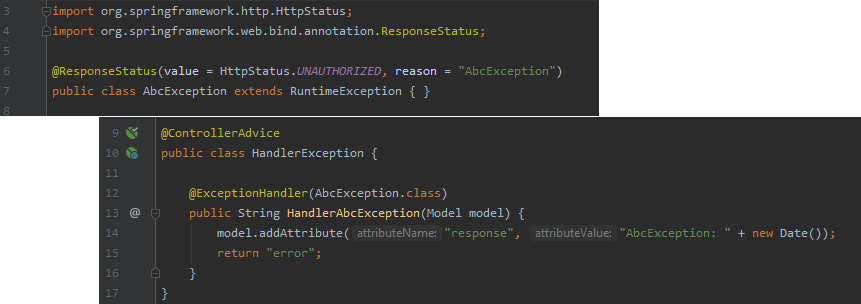
Kịch bản của chúng ta là tạo ra 2 URI /login và /exception, với /login cho phép nhập username và password để đăng nhập, thực hiện xác thực nếu thông tin nhập vào là rỗng. Thực hiện câu chào khi username được nhập vào là root và password là 123, kết quả như sau:



Mã nguồn tương đương:

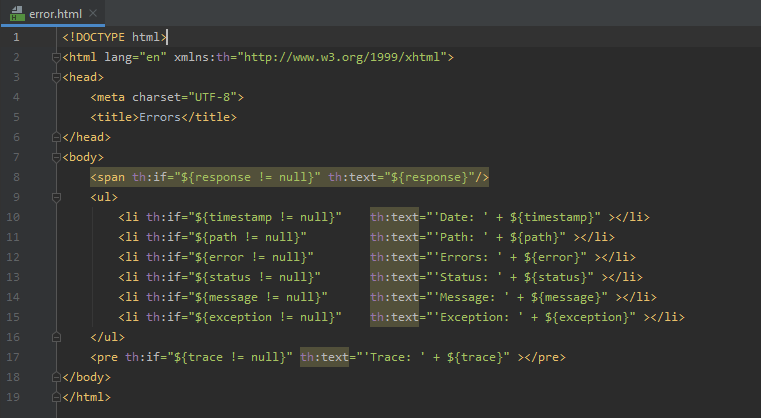


Còn lại /exception sẽ chủ động quăng ra một ngoại lệ AbcException do tôi tự định nghĩa, và sẽ được bắt lấy và xử lý bởi 1 method có đánh dấu @ExceptionHandler. Nó có nội dung như sau:

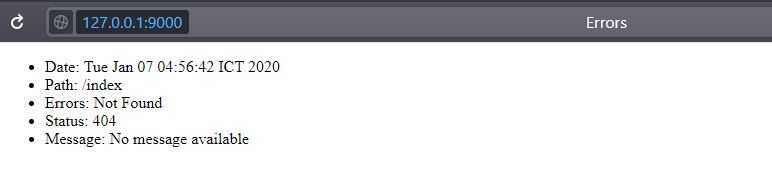


Tất nhiên, nếu không bắt lấy và xử lý, Spring hoặc Server sẽ luôn làm việc đó thay ta, việc bắt exception giúp chúng ta chủ động hơn trong các thao tác, không nhất thiết là phải thông báo lỗi mà cũng có thể xử lý dữ liệu sao cho không xảy ra ngoại lệ nữa.

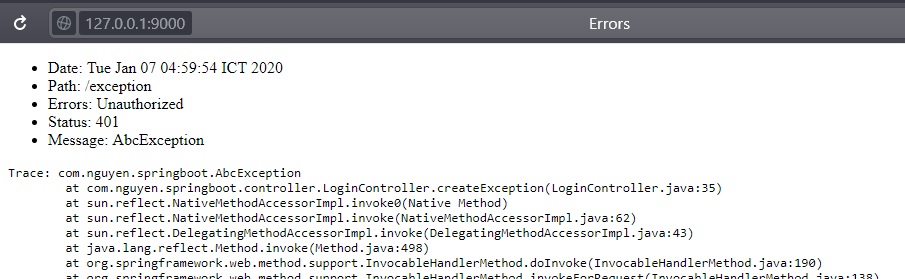
Tiếp theo, tôi sẽ không dùng trang thông báo lỗi mặc định của Spring Boot hay của Server (tomcat) để hiển thị lỗi mà sẽ tùy chỉnh theo ý mình, nội dung như sau:



Do đó, các exception mà tôi không xử lý hay các thông báo HttpStatus từ server sẽ hiển thị theo trang error.html, dưới đây là ảnh chụp màn hình khi truy cập với URL chưa được định nghĩa tại server:



Hoặc khi tôi không chủ động bắt exception AbcException thông điệp trả về sẽ như sau:



## Tham khảo

Mã nguồn mục [4.5.3] có sẵn tại đây.

# Đa ngôn ngữ

## Tổng quan

Đôi khi chúng ta muốn trang web của chúng ta tiếp cận được với nhiều đối tượng hơn (đối tượng ở đây được hiểu là người dùng với ngôn ngữ khác nhau). Khi đó chúng ta cần viết ra 1 trang web đa ngôn ngữ.

Website đa ngôn ngữ còn được biết đến với tên gọi Internationalization (quốc tế hóa), bắt đầu với ‘i’, kết thúc với ‘n’ và có 18 ký tự ở giữa vì vậy nó được viết tắt là (i18n).

Tôi và bạn sẽ cùng tìm hiểu về đa ngôn ngữ trong Spring theo mức độ từ đơn giản đến phức tạp dần.

## Khái niệm cơ bản

Thứ nhất: interceptor sẽ xử lý request trước, sau, … khi nó được gửi tới (hoặc rời khỏi) 1 controller xác định.

Thứ hai: Các xxxResolver sẽ có nhiệm vụ xử lý các giá trị để trả về nguồn tài nguyên tương ứng. Ví dụ thường thấy là ViewResolver sẽ xử lý ánh xạ giữa logical view và physical view, và tương tự đối với đa ngôn ngữ chúng ta sẽ có LocaleResolver.

Thứ ba: Các xxxSource được Spring dùng để quản lý các nguồn tài nguyên tĩnh trong hệ thống. Ví dụ thường thấy là MessageSource và trong chương này chúng ta cũng có DataSource.

Luồng cơ bản: Interceptor sẽ lấy các LocaleResolver tương ứng với Request tới, LocaleResolver xử lý và trả về Locale. Interceptor quyết định cách lưu trữ Locale.

## Quy định đặt tên file

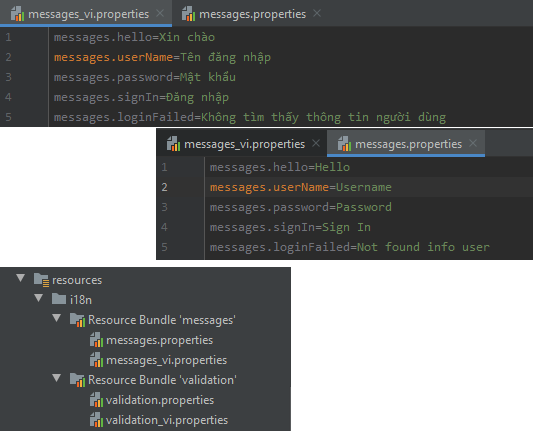
Không có quy tắc xác định trong việc đặt tên file properties nhưng tôi và bạn sẽ cùng quy ước 2 nguyên tắc cơ bản để làm việc dễ hơn với đa ngôn ngữ trong Spring.

Thứ nhất: Các file messages\_xx.properties chứa các messages đa ngôn ngữ. Ví dụ: messages\_vi.properties chứa các messages cho tiếng việt và messages\_en.properties chứa các messages cho tiếng anh.

Thứ hai: Các key trong các file properties là trùng khớp với nhau và bắt đầu bởi tiền tố là tên file cho đến dấu ‘\_’.

Các file validations\_xx.properties cũng vậy nhưng nó chứa các messages cho việc xác thực dữ liệu form.

Hình sau minh họa cho quy tắc của chúng ta:



File messages.properties hay file validation.properties chứa các message mặc định (ở đây là message tiếng anh) sẽ được dùng nếu các message chưa được định nghĩa cho ngôn ngữ yêu cầu.

Về cơ bản là vậy, sau đây chúng ta sẽ cùng đi đến các trường hợp sử dụng từ cơ bản đến nâng cao hơn với đa ngôn ngữ trong Spring.

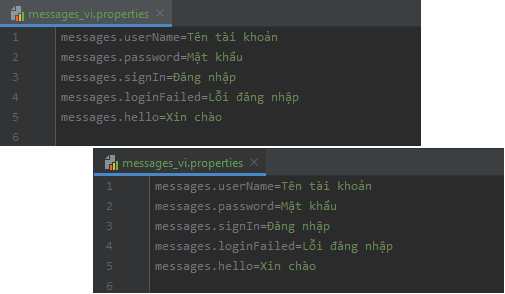
## i18n cố định

Đây là mức cơ bản nhất trong đa ngôn ngữ với Spring khi mà ngôn ngữ hiển thị được quyết định bởi việc thiết lập giá trị spring.mvc.locale trong file application.properties (hoặc application.yml).

Để ví dụ nhanh phần này, tôi sẽ bỏ qua phần xác thực dữ liệu form. Vì vậy, hãy tạo các file messages\_xx.properties và thiết lập nội dung tương ứng với từng ngôn ngữ của bạn.

### Tạo file messages

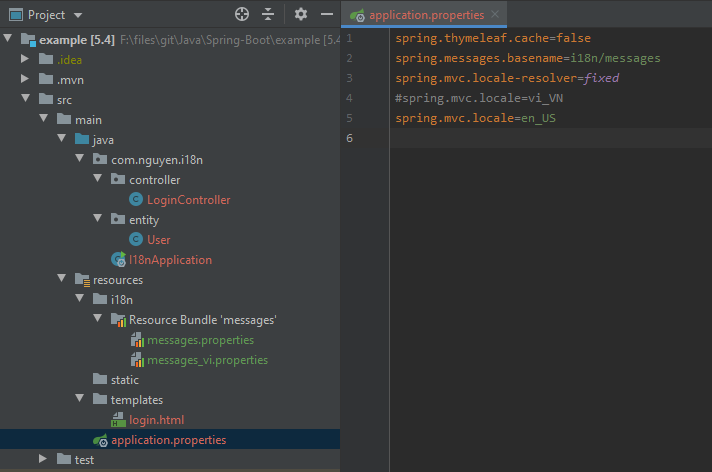
Nội dung 2 file messages\_vi.properties giành cho ngôn ngữ tiếng Việt và messages.properties giành cho ngôn ngữ tiếng Anh.



Đây gần như là một bước quan trọng và không thể thiếu trong quá trình thao tác và làm việc với đa ngôn ngữ.

### Thiết lập cấu hình trong file application.properties

Hình minh họa sau thể hiện nội dung file application.properties và cấu trúc project:



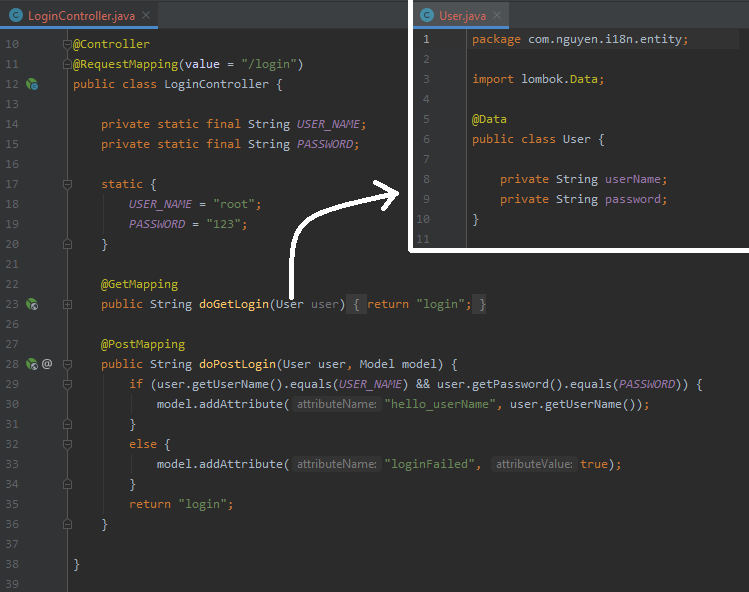
1. Giá trị đầu tiên giúp tôi tắt chức năng cache, không quan trọng, bạn có thể bỏ đi nếu thấy không cần thiết.
2. Giá trị thứ 2 giúp chúng ta thiết lập pattern cho tài nguyên messages, ở đây là thư mục i18n và file có tiền tố là messages.
3. Giá trị thứ 3 sẽ cố định LocaleResolver là hiển thị tiếng anh, thay vì xác định locale từ trình duyệt, hãy thử chú thích nó lại và chạy ứng dụng bạn sẽ thấy sự khác biệt.
4. Giá trị thứ 4 và 5 quy định Locale.

### Tạo và xử lý object ModelAttribute

Chúng ta sẽ tạo ra phương thức doGetLogin cho phép tạo ra 1 đối tượng user và truyền nó view ‘login’. Đối tượng này sẽ được mapping với các field thích hợp trong form login sẽ được trình bày như trong mục [5.4.4].

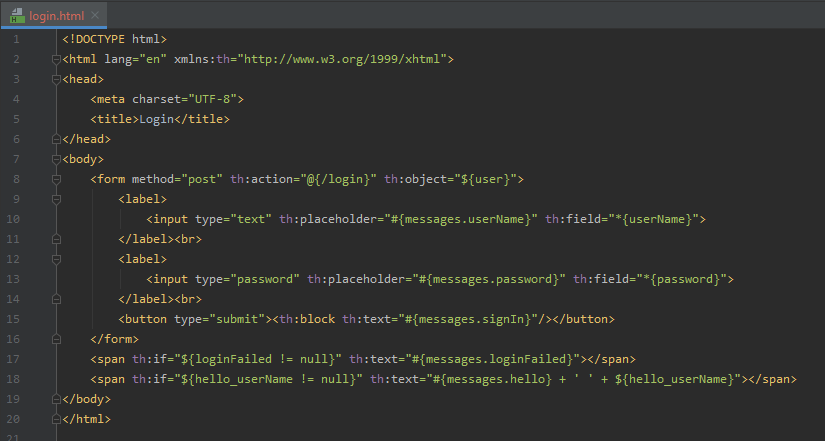
Phương thức doPostLogin sẽ nhận lại đối tượng user này, thực hiện các xử lý thích hợp như kiểm tra tên đăng nhập và mật khẩu sau đó trả về thông điệp ‘hello\_userName’ nếu thông tin đăng nhập là đúng, hoặc ‘loginFailed’ nếu đăng nhập thất bại.

File LoginController của chúng ta có nội dung như sau:



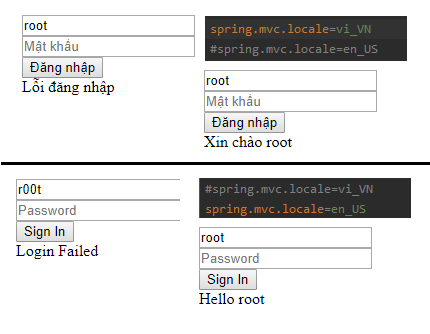
### Ánh xạ đối tượng ModelAttribute trong view

Đây là các thao tác khi làm việc với Thymeleaf, và không nằm trong phạm vi tài liệu này, vì vậy tôi sẽ không giải thích các cú pháp của nó. Bạn có thể tham khảo thêm về thymeleaf tại đây.



Đặc biệt chú ý tới các cú pháp dạng #{content} ngôn ngữ ở vị trí đó sẽ thay đổi khi bạn thiết lập giá trị cho thuộc tính spring.mvc.locale trong file cấu hình.

### Kết quả



### Tham khảo

Bạn có thể tham khảo mã nguồn tại đây.

### Có thể bạn sẽ cần

Spring sử dụng chuẩn UTF-8 để lưu trữ và đọc các file message. Encode nên luôn là UTF-8. Nếu messages của bạn hiển thị không đúng như trong nội dung file. Vui lòng thiết lập encode là UTF-8 để messages được hiển thị bình thường.

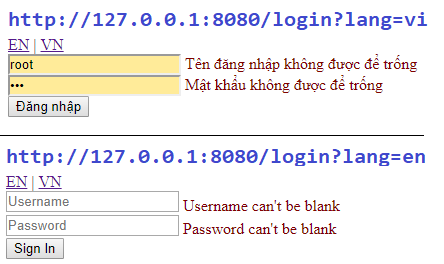
## i18n với request param

### Tổng quan

Nội dung mục [5.4] đã hướng dẫn bạn cách làm việc với đa ngôn ngữ trong Spring với Locale cố định được thiết lập trong file cấu hình tại server.

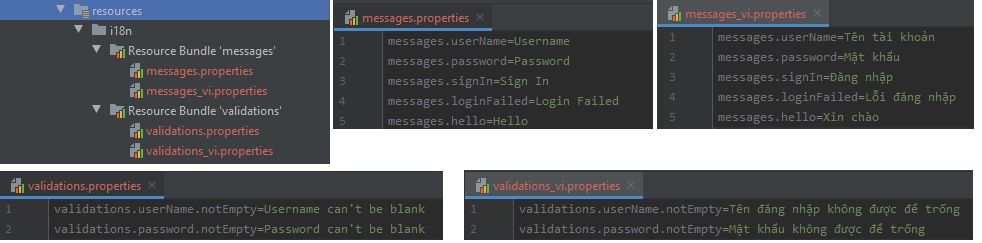
Phần [5.5] này sẽ cung cấp một cách tiếp cận khác, linh động và thân thiện hơn, nó giúp người dùng chủ động hơn trong việc lựa chọn ngôn ngữ sẽ được hiển thị qua tham số khi request.

Kết quả của chúng ta sẽ có dạng như sau:



Giá trị sẽ được lưu lại trong cookie hoặc session, vì lý do đó đối với những request tiếp theo sẽ không cần kèm theo tham số này.

### Tạo file messages



Ngoài thiết lập đa ngôn ngữ cho các message được hiển thị ra màn hình khi lần đầu tải trang. Chúng ta cũng cần thiết lập đa ngôn ngữ cho các thông điệp được server gửi trả khi xác thực dữ liệu từ form, theo quy định của chúng ta ở mục [5.3], thông điệp đa ngôn ngữ dùng để gửi trả được quy định trong các file validations\_xx.properties, nội dung minh họa như hình bên trên.

### Thiết lập cấu hình

Chúng ta đã thiết lập cấu hình nhanh trong file application.properties ở ví dụ trước, đôi khi trong 1 số trường hợp nó là chưa đủ, vì vậy ở ví dụ này tôi và bạn sẽ cùng thao tác cấu hình trong lớp cấu hình được đánh dấu bởi annotation Configuration.

Hãy đọc lại các khái niệm cơ bản trong mục [5.2] nếu bạn thấy khó khăn trong việc tiếp cận phần này.

#### Lớp cấu hình

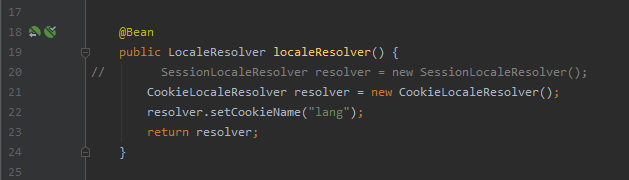
Lớp cấu hình của chúng ta sẽ implement interface WebMvcConfigurer và có nội dung như sau:



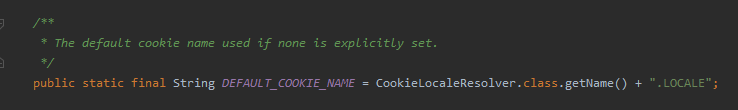
Các đối tượng được xem như 1 Bean trong servlet context vì thế chúng được tự động sử dụng khi cần thiết.

#### LocaleResolver

Nếu đã từng làm việc với Spring MVC thì có lẽ chúng ta đã quá quen thuộc với ViewResolver, nó thực hiện ánh xa logical view đến physical view. Chúng ta thường khởi tạo nó trong file cấu hình xml. Thì ở đây, LocaleResolver cũng có chức năng tương tự như vậy, nó thực hiện ánh xạ các request đến các Locale thích hợp. Và lưu trữ vào cookie hoặc session tùy theo nhu cầu của người lập trình. Đối với cookie chúng ta có CookieLocaleResolver và với session chúng ta có SessionLocaleResolver.



Không bắt buộc thiết lập setCookieName, giá trị mặc định của nó là:

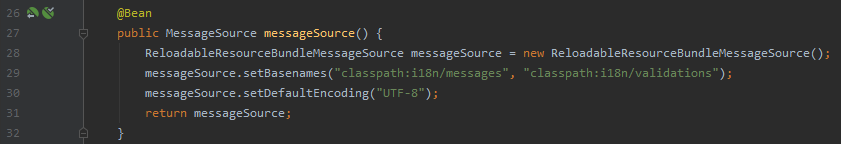


Nếu request không thể ánh xa đến Locale phù hợp, nó sẽ được lấy từ cookie, và nếu cookie không chứa giá trị phù hợp, nó được lấy từ defaultLocale.

#### MessageSource

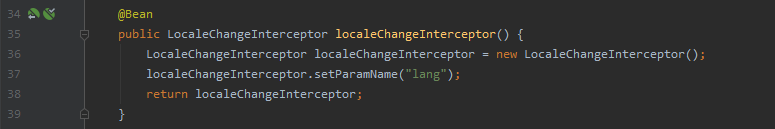
Quản lý các tài nguyên message, Từ Locale được gởi trả bởi LocaleResolver nó quyết định message theo ngôn ngữ nào sẽ được sử dụng, chúng ta cũng sẽ gặp RoutingDataSource ở ví dụ của phần [5.7], nó là 1 loại DataSource đặc biệt, từ request của người dùng, nó quyết định data source nào sẽ được sử dụng nếu ứng dụng của chúng ta có nhiều database.

Và như tôi đã từng đề cập, Spring sử dụng encode UTF-8 để lưu trữ và đọc các file message, vì thế chúng ta cũng cần thiết lập DefaultEncoding cho MessageSource là UTF-8, và chúng ta cũng cần thiết lập Pattern cho tài nguyên message, chỉ định thư mục và tiền tố cho chúng, hình minh họa phía dưới thể hiện nội dung của Bean MessageSource như sau:



#### LocaleChangeInterceptor

Bất cứ khi nào người dùng có nhu cầu thay đổi ngôn ngữ, LocaleChangeInterceptor sẽ lấy giá trị đó từ request với ParamName xác định, tạo Locale và tiến hành sử dụng phương thức setLocale của LocaleResolver để lưu trữ lại Locale.

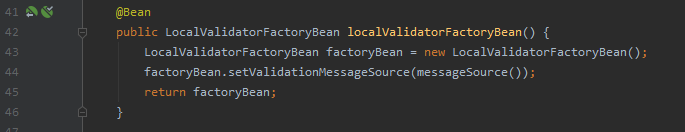


Bạn có thể đọc kỹ hơn về Interceptor này tại đây.

#### LocalValidatorFactoryBean

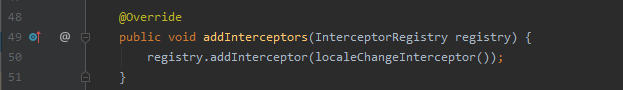
Lớp này thực thi interface ValidatorFactory, ValidatorFactory là mặc định được sử dụng để xác thực dữ liệu được gửi lên từ form. Khi chúng ta cần phụ thuộc nhiều hơn vào việc messages nào sẽ được trả về với Locale tương ứng thì chúng ta nghĩ đến LocalValidatorFactoryBean, bạn có thể đọc chi tiết về nó trong đặc tả tại đây.

Hình dưới đây minh họa nội dung của phương thức localValidatorFactoryBean, bạn sẽ nhìn thấy phương thức setValidationMessageSource được gọi từ LocalValidationFactoryBean. Hãy xem lại phương thức messageSource phía trên.



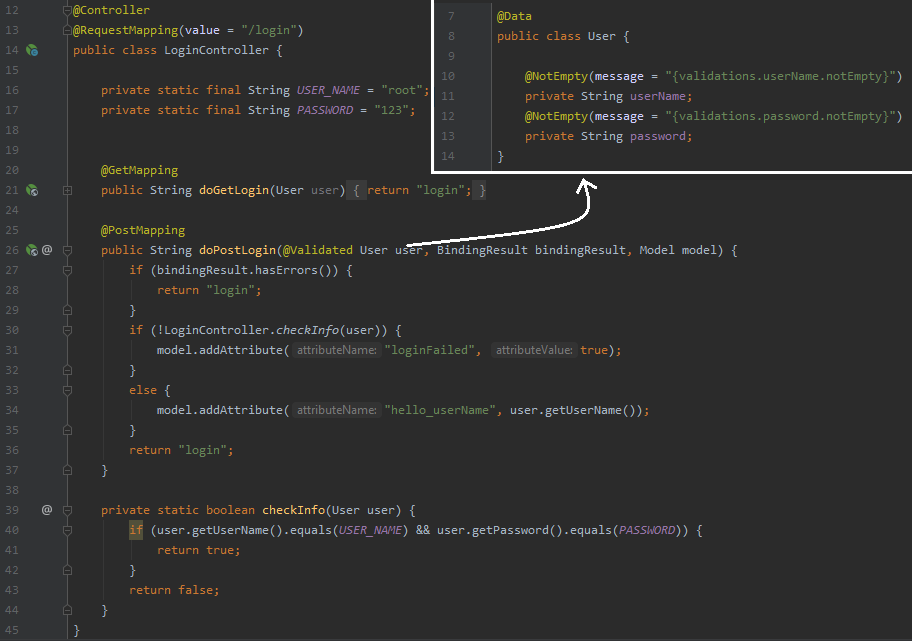
#### Thêm interceptor

Sau khi có interceptor tại mục [5.5.3.4] chúng ta phải đăng ký để nó để xử lý các request trước khi các request được gửi tới 1 controller xác định.



### Tạo và xử lý object ModelAttribute

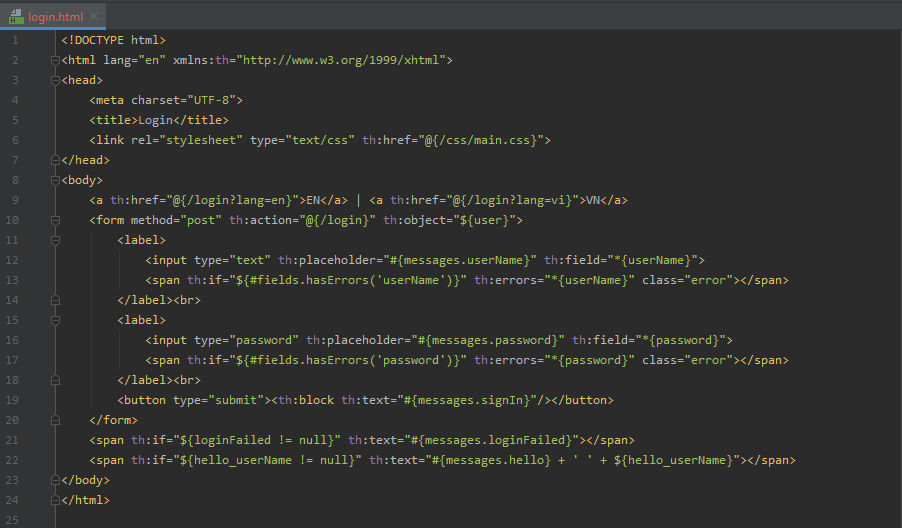
Không có quá nhiều sự khác biệt đối với ví dụ trước ngoại trừ tại phương thức doPostLogin, chúng ta sẽ kiểm tra lỗi xác thực, nếu nó xảy ra, trả về trang login mà không thao tác gì thêm.



Luôn nhớ đánh dấu thuộc tính cần xác thực với giá trị message được lấy từ file validations\_xx.properties.

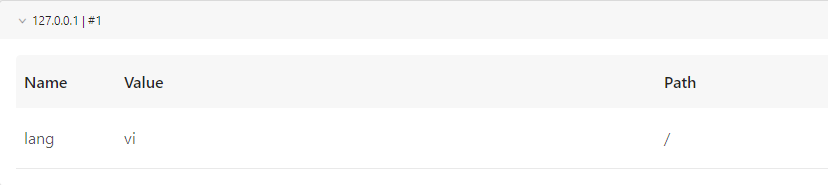
### Ánh xạ đối tượng ModelAttribute trong view

Sau đây là nội dung file login của chúng ta, hãy chú ý tới các thẻ <span>. Gần như đây là một thao tác bắt buộc để thực hiện in ra các thông báo lỗi khi xác thực dữ liệu phía server thất bại. Hãy nhớ cú pháp của nó, có thể sau này bạn sẽ cần.



### Kết quả

Mục [5.5.1] đã trình bày kết quả với bạn. Ngoài ra chúng ta cũng có thể theo dõi giá trị ‘lang’ trong cookie bằng nhiều cách (extension của browser hoặc trình quản lý cookie mặc định của browser). Mỗi browser sẽ có trình quản lý cookie khác nhau, vui lòng tìm hiểu tại các nguồn khác trên mạng.



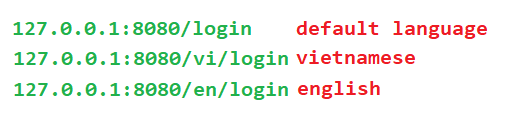
### Tham khảo

Mã nguồn ví dụ này có sẵn tại đây.

## i18n với path variable

### Tổng quan

Request Param là một cách để xác định Locale phù hợp và Path Variable cũng thế, nó có dạng như sau:



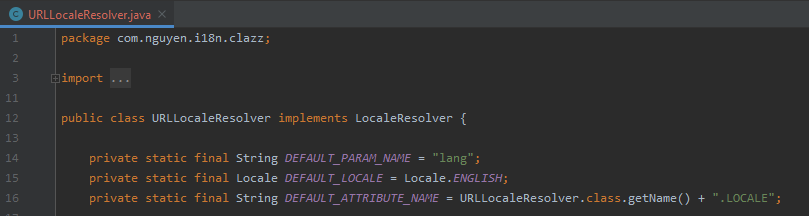
Với cách này chúng ta không cần lưu lại language trong cookie hay session, giá trị được chỉ định sẵn trên URL.

Để làm được điều này, chúng ta cần tạo ra một lớp thực thi giao diện LocaleResolver. Sau đó, thay thế nó khi tạo bean LocaleResolver trong file cấu hình.

### URLLocaleResolver

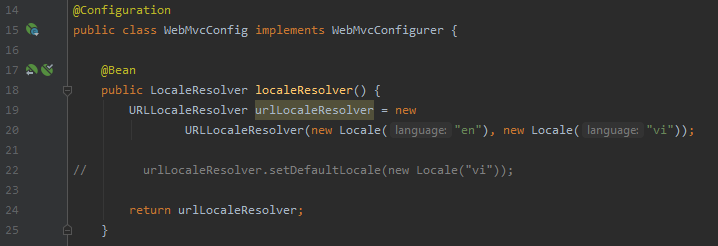
Nếu bạn hiểu về luồng thực thi mà tôi đã trình bày trong phần [5.2] cũng như nắm được cách thức hoạt động của LocaleChangeInterceptor và CookieLocaleResolver, thì tôi nghĩ bạn cũng có thể tùy chỉnh 2 lại lớp này mà không gặp quá nhiều khó khăn. Nếu khó để tiếp cận, bạn có thể sử dụng lớp URLLocaleResolver mà tôi sẽ cung cấp cho bạn trong source code tương ứng.

Đây là lớp mà tôi đã định nghĩa những thao tác cơ bản, bạn sẽ dùng lớp này để thay thế cho CookieLocaleResolver hoặc SessionLocaleResolver.



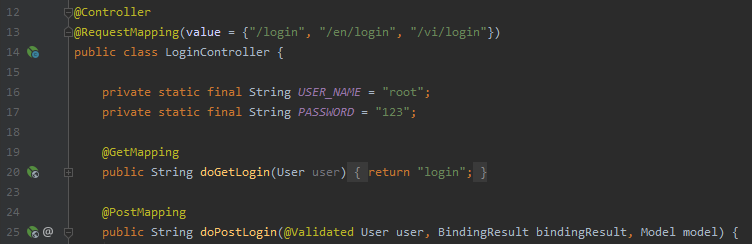
### Tạo bean localeResolver bằng URLLocaleResolver

Bạn chỉ cần cung cấp danh sách các Locale phù hợp với messages mà bạn đã định nghĩa. Thiết lập defaultLocale (nếu cần) mặc định là Locale.ENGLISH.



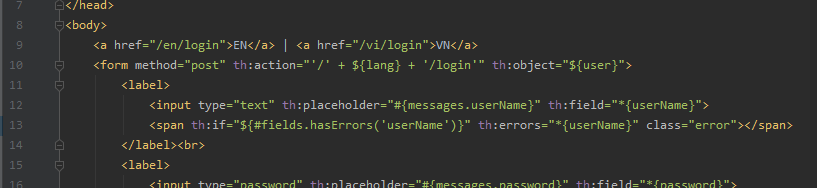
### Chấp nhận URL chỉ định ngôn ngữ hiển thị

URLLocaleResolver sẽ trả về Locale tương ứng với URL khi request mà bạn không phải quan tâm cách nó vận hành như thế nào, chỉ cần chấp nhận các request có dạng như sau:

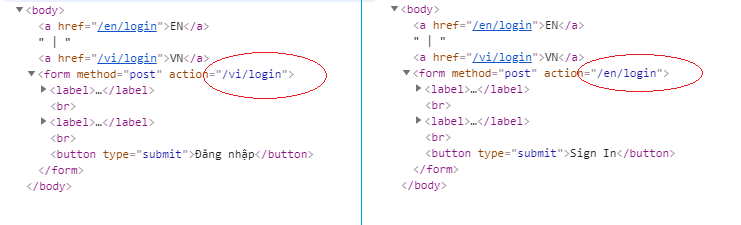


### Sử dụng thuộc tính ‘lang’ khi thao tác với view

Ngoài ra, URLLocaleResolver còn cung cấp 1 attribute với name là ‘lang’ để bạn có thể sử dụng tại file view, nó dùng để xác định ngôn ngữ hiện tại của trang web, bạn sẽ cần để thiết lập các đường dẫn phù hợp với ngôn ngữ hiện tại, sau đây là ví dụ để sử dụng nó:



Tùy vào ngôn ngữ hiện tại mà action của form sẽ thay đổi, đối với ví dụ của chúng ta, nó sẽ thay đổi như sau:



### Tham khảo

Mã nguồn ví dụ này có sẵn tại đây.