Theo mặc định thì Maven compile source từ src/main/java

**import** org.springframework.boot.\*;

**import** org.springframework.boot.autoconfigure.\*;

**import** org.springframework.web.bind.annotation.\*;

*@RestController*

*@EnableAutoConfiguration*

**public** **class** Example {

*@RequestMapping("/")*

String home() {

**return** "Hello World!";

}

**public** **static** **void** main(String[] args) **throws** Exception {

SpringApplication.run(Example.**class**, args);

}

}

**The @RestController and @RequestMapping Annotations**

Đây là stereotype annotation. Nó cung cấp gợi ý cho người đọc code và cho Spring rằng class có một vai trò cụ thể

Trong trường hợp này sử dụng @Controller vì vậy Spring xem xét nó khi cần handle các yêu cầu web đến.

@RequestMapping annotation cung cấp thông tin routing. Nó cho Spring biết rằng bất kỳ yêu cầu HTTP nào có đường dẫn / path .

**The @EnableAutoConfiguration Annotation**

Annotation này nói Spring Boot biết cách mà muốn cấu hình Spring dựa trên các phụ thuộc jar mà đc thêm vào.

**Creating an Executable Jar**

Executable Jar là các tệp lưu trữ chứa các class đã biên dịch cùng với tất cả các phụ thuộc jar mà code cần để chạy.

Để tạo 1 jar thực thi, cần them plugin spring-boot-maven vào tệp POM

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

Sử dụng để biến project thành dạng jar mà ta có thể dùng nó để chạy trực tiếp

Build : mvn package

Chạy : java –jar target/myproject-0.0.1-SNAPSHOT

Vì mặc định nó sẽ ở target

**Using Spring Boot**

Maven có thể thùa kế từ spring-boot-starter-parent để có được đánh giá mặc định hợp lý.

**Using Spring Boot without the Parent POM**

Không phải ai cũng thích kế thừa từ Spring-boot-starter-parent. Bạn có thể có parent từ công ty khác sử dụng hoặc có thể muốn hai báo rõ ràng tất cả cấu hình Maven

Nếu ko muốn sử dụng spring-boot-started-parent mà vẫn giữu lại lợi ích của việc quản lý phụ thuộc bằng cách sử dụng scope=import

<dependencyManagement>

<dependencies>

<dependency>

*<!-- Import dependency management from Spring Boot -->*

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-dependencies</artifactId>

<version>2.0.3.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

**Using the Spring Boot Maven plugin**

Spring Boot bao gồm một plugin Maven có thể đóng gói dự án như là 1 jar thực thi . Thêm plugin vào phần <plugins> nếu muốn sử dụng plugin như đưuọc hiển thị như dưới :

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

**Starters**

Trình khởi tạo ( starter ) là một bộ mô tả phụ thuộc tiện lợi mà có thể đưa vào trong ứng dụng.

Starter chứa rất nhiều depedentcy mà cần để có được 1 dự án và chạy 1 cách nhanh chóng với quản lý dependency một cách nhất quán và được hỗ trợ.

Các chương trình khởi động ứng dụng sau đây cung cấp bởi Spring Boot trong nhóm org.springframework.boot

Sử dụng @SpringBootApplication để đánh dấu là class chính và chạy Spring Boot

**package** com.example.myapplication;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

**public** **class** Application {

**public** **static** **void** main(String[] args) {

SpringApplication.run(Application.**class**, args);

}

}

Mặc dù có thể sử dụng SpringApplication với XML nhưng nếu mà muốn config nào đó thì sử dụng @Configuration

Nhiều cấu hình để muốn sử dụng cần sử dụng @Enable để kích hoạt chúng .

@Import có thể được sử dụng để nhập cấu hình bổ sung của class. Ngoài ra có thể sử dụng @ComponentScan để tự động nhận tất cả các phần Spring bao gồm các class @Configuration.

Nếu đang sử dụng @Configuration mà muốn sử dụng them XML thì có thể sử @ImportResource để tải các tệp cấu hình XML

Cấu hình tự đổng khởi động Spring Boot sẽ tự động cấu hình ứng dụng Spring của bạn dựa trên các phụ thuộc mà jar mà đã theme vào .

Để cấu hình chú thích bằng cách sử dụng @EnableAutoConfiguration hoặc @SpringBootApplication vào một class @Configuration \=> Có nghĩa là 1 trong 3 anotation đều kích hoạt @Configuration

Tại bất kỳ thời điểm nào , để có thể cấu hình riêng để thay thay thế các phần cụ thể của cấu hình tự đọng. Sử dụng Data Source

**Disabling Specific Auto-Configuration Classes**

Nếu bạn thấy rằng các class cấu hình tự động cụ thể mà bạn k muốn đang được áp dụng, có thể sử dụng thuộc tính exclude của @EnableAutoConfiguration để vô hiệu hóa chúng.

**import** org.springframework.boot.autoconfigure.\*;

**import** org.springframework.boot.autoconfigure.jdbc.\*;

**import** org.springframework.context.annotation.\*;

*@Configuration*

*@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})*

**public** **class** MyConfiguration {

}

**Spring Bean and Depadency Injection**

Bạn được tự do sử dụng bất kỳ kỹ thuật Framework Spring chuẩn nào để định nghĩa Bean và Depedency Injection.

Chúng ta thường thấy việc sử dụng @ComponentScan ( để tìm Bean ) và sử dụng @Autowired ( để inject) .

Nếu cấu trúc @ComponentScan mà o thêm bất kỳ đối số nào thì tất cả các thành phần ứng dụng như @Component, @Service, @Repository, @Controller đều được đăng ký tự động như Spring Beans.

**package** com.example.service;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

*@Service*

**public** **class** DatabaseAccountService **implements** AccountService {

**private** **final** RiskAssessor riskAssessor;

*@Autowired*

**public** DatabaseAccountService(RiskAssessor riskAssessor) {

**this**.riskAssessor = riskAssessor;

}

*// ...*

}

Đây là ví dụ sử dụng @Service Bean và được @Autowired tiêm vào .

Nếu một Bean có hàm khởi tạo thì có thể bỏ qua @Autowired .

**Using the @SpringApplication Annotation**

Nhiều developer Spring Boot thích ứng dụng cấu hình tự động , component scan và có thể xác định cấu hình bổ sung trên “application class”.

@SpringBootApplication có thể sử dụng để kích hoạt 3 tính năng đó là

* @EnableAutoConfiguration : Bật tự động cấu hình của Spring Boot
* @ComponentScan: Cho phép @Component scan trên các gói ứng dụng đưuọc đặt
* @Configuration : Cho phép đăng ký Bean trong Context hoặc import các cấu hình bổ sung .

Nó tương đương với cấu hình sau :

**package** com.example.myapplication;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

*// same as @Configuration @EnableAutoConfiguration @ComponentScan*

**public** **class** Application {

**public** **static** **void** main(String[] args) {

SpringApplication.run(Application.**class**, args);

}

}

SpringBootApplication cũng cung cấp các alias để tùy chỉnh thuộc tính của @EnableAutoConfiguration và @ComponentScan.

Tất cả tính năng của SpringBoot là ko bắt buộc.

Nếu ko muốn sử dụng @SpringBootApplication nó tương đương với cấu hình sau :

**package** com.example.myapplication;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.context.annotation.ComponentScan

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.context.annotation.Import;

*@Configuration*

*@EnableAutoConfiguration*

*@Import({ MyConfig.class, MyAnotherConfig.class })*

**public** **class** Application {

**public** **static** **void** main(String[] args) {

SpringApplication.run(Application.**class**, args);

}

}

**Runnig Your Application**

Một trong những lợi thế lớn nhất của việc đóng gói ứng dụng như 1 jar và sử dụng để nhúng HTTP là có thể chạy ứng dụng như bất kỳ ứng dụng nào khác.

**Running as a Packaged Application**

Nếu sử dụng các plugin bổ sung của Spring Boot Maven hoặc Gradle để tạo jar thực thi , và có thể chạy ứng dụng bằng cách sử dụng java -jav

$ java -jar target/myapplication-0.0.1-SNAPSHOT.jar

Và nó cũng có thể chạy ứng dụng packaged với hỗ trợ gỡ lỗi từ xa được kích hoạt. Làm như vậy cho phép đính kèm một gỡ nào vào ứng dụng :

$ java -Xdebug -Xrunjdwp:server=y,transport=dt\_socket,address=8000,suspend=n \

-jar target/myapplication-0.0.1-SNAPSHOT.jar

**Using the Maven Plugin**

Plugin Spring Boot Maven bao gồm một mục để có thể nhanh chóng compile và run ứng dụng . Ứng dụng chạy dưới dạng exploded form giống như run bằng IDE

$ mvn spring-boot:run

**Developer Tools**

Spring Boot bao gồm 1 bộ công cụ bổ sung có thể làm cho trải nghiệm phát triển ứng dụng dễ chịu hơn một chút.

Module spring-boot-devtools có thể được bao gồm trong bất kỳ dự án nào để cung cấp tính năng phát triển bổ sung

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

</dependencies>

**Property Default**

Một số thư viện được Spring Boot hỗ trợ sử dụng cache để cải thiện hiệu suất. Ví dujg template engine cache được biên dịch các template để tránh lặp lại việc phân tích cú pháp các template.

Spring MVC có thể them các header cache HTTP để vào reponse.

Mặc dù cache có lợi trong production nhưng nó có phản ứng phụ trong quá trình develop , ngăn bạn nhìn thấy những thay đổi vừa tạo troing ứng dụng của mình. Chính vì lý do này Spring-Boot-Devtools dùng để vô hiệu hóa các tùy chọn bộ nhớ đệm theo mặc định .

Tùy chọn bộ nhớ cache thường được định cấu hình theo cài đặt trong tệp application.properties .

Thymeleaf cung cấp thuộc tính spring.thymeleaf.cache . Thay vì cần thiết lập các thuộc tính này theo cách thủ công, module spring-boot-devtools sẽ tự động áp dụng cấu hình development-time configuration

**Automatic Restart**

Các ứng dụng sử dụng spring-boot-devtools sẽ tự động khởi động lại bất cứ khi nào các tệp trên đường dẫn với class thay đổi.

Đây là tính năng hữu ích khi làm việc trong một IDE vì nó cung cấp một vòng phản hồi rất nhanh để thay đổi code.

Theo mặc định , mọi mục nhập trên đường dẫn class trỏ tới một thư mục sẽ được theo dõi để thay đổi. Lưu ý rằng mọt số tài nguyên nhất định cần phải khởi động lại ứng dụng.

**Logging changes in condition evaluation**

Để tắt logging báo cáo có thể dùng thuộc tính sau :

spring.devtools.restart.log-condition-evaluation-delta=false

**Excluding Resource**

Một số tài nguyên nhất định không nhất thiết phải kích hoạt khởi động lại khi chúng được thay đổi.

Ví dụ, các Thymeleaf có thể được chỉnh sửa tại chỗ.

Theo mặc định, việc thay đổi tài nguyên trong /META-INF/maven, /META-INF/resource , /resource,/static,/public, hoặc /templates ko kích hoạt không khởi động lại nhưng lại kích hoạt tải lại trực tiếp.

Nếu muốn tùy chỉnh các loại trừ này, có thể sử dụng thuộc tính spring.devtools.restart.exclude

spring.devtools.restart.exclude=static/\*\*,public/\*\*

**Disabling Restart**

Nếu ko muốn sử dụng tính năng restart có tắt nó bằng cách sử dụng thuộc tính spring.devtools.restart.enable

Có thể đặt nó ở application.property hoặc trong file main

**public** **static** **void** main(String[] args) {

System.setProperty("spring.devtools.restart.enabled", "false");

SpringApplication.run(MyApp.**class**, args);

}

**LiveReload**

Các module spring-boot-devtools bao gồm một máy chủ LiveReload nhúng có thể được sử dụng để kích hoạt làm mới trình duyệt mỗi khi tài nguyên được thay đổi.

Tiện ích mở rộng trình duyệt LiveReload có sẵn cho Chrome, FileFox, Safari từ livereload.com

Nếu ko muốn khởi động máy chủ LiveReload khi ứng dụng chạy , có thể đặt thuộc tính spring.devtools.livereload.enabled = false

**Global Settings**

Có thể cấu hình các thiết lập devtools Global bằng cách them một tệp có tên .spring-boot-devtools.property vào thư mục $HOME .

Bất kỳ thuộc tính them vào đều áp dụng cho tất cả các ứng dụng Spring Boot trên máy của bạn sử dụng devtools

**~/.spring-boot-devtools.properties.**

spring.devtools.reload.trigger-file=.reloadtrigger

**Remote Application**

Các công cụ phát triển Spring Boot ko chỉ giới hạn trong phát triển cục bộ. Bạn có thể sử dụng một số tính năng khi chạy ứng dụng remote .

Hỗ trợ từ xa là 1 lựa chọn. Để kích hoạt nó cần đảm bảo rằng các devtools được bao gồm trong kho kuwu trữ được đóng goi lại

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<excludeDevtools>false</excludeDevtools>

</configuration>

</plugin>

</plugins>

</build>

Sau đó cần thiết lập thuộc tính spring.devtools.remote.secret và kích hoạt spring-boot-devtools

spring.devtools.remote.secret=mysecret

Hỗ trợ devtools từ xa được cung cấp trong 2 phần : điểm cuối phía máy chủ chấp nhận các kết nối và ứng dụng khách đang chạy trong IDE .

Thành phần máy chủ được tự động kích hoạt khi thuộc tính spring.devtools.remote.secret được thiết lập.

Thành phần client phải được khởi chạy thành công.

**Running the Remote Client Application**

**Sử dụng** org.springframework.boot.devtools.RemoteSpringApplication với class pass như dự án từ xa mà kết nối tới. Đối số yêu cầu duy nhất của ứng dụng URL từ xa mà nó kết nối.

Bởi vì client từ xa đang sử dụng cùng classpath ứng dụng thực, nó có thể đọc trực tiếp các thuộc tính của ứng dụng. Đây là thuộc tính spring.devtools.remote.secret đưuọc truyền vào máy chủ để xác thực.

Nên sử dụng giao thực https:// làm giao thực kết nối để lưu lượng truy cập được mã hóa và mạt khẩu ko thể bị chặn.

Nếu cần sử dụng proxy để truy cập ứng dụng từ xa , cấu hình thuộc tính spring.devtools.remote.proxy.host và spring.devtools.remote.proxy.port

**Spring Boot features**

**SpringApplication**

SpringApplication cung cấp một cách thuận tiện để khởi động một ứng dụng Spring được bắt đầu từ phương thức main

Trong nhiều tình huống có thể quyền cho method SpringApplication.run static như ví dụ sau :

**public** **static** **void** main(String[] args) {

SpringApplication.run(MySpringConfiguration.**class**, args);

}

**Customizing the Banner**

Banner được in khi khởi động có thể được thay đổi bằng cách them tệp banner.txt vào classpath bằng cách đặt thuộc tính spring.banner.location vào vị trí của tệp đó. Nếu tệp có mã hóa khác với UTF-8 có thể đặt spring.banner.charset .

Ngoài tệp văn bản cũng có thể thêm tệp banner.gif hoặc png vào đường dẫn classpath hoặc đặt thuộc tính spring.banner.image.location

**Customizing SpringApplication**

Nếu mặc định SpringApplication ko theo sở thích của bạn , bạn có thể tạo 1 cá thể cục bộ và tùy chỉnh nó.

Để tắt Banner có thể viết.

**public** **static** **void** main(String[] args) {

SpringApplication app = **new** SpringApplication(MySpringConfiguration.**class**);

app.setBannerMode(Banner.Mode.OFF);

app.run(args);

}

Cũng có thể cấu hình trong application.property

**Fluent Builder API**

Nếu bạn cần xây dụng 1 hệ thống phân cấp ApplicationContext hoặc nếu thích sử dụng một API xây dụng “fluent” có thể sử dụng SpringApplicationBuilder

SpringApplicationBuilder cho phép kết nối nhiều cuộc gọi method với nhau và bao gồm các method cha con cho phép tạo ra hệ thống phân cấp .

**new** SpringApplicationBuilder()

.sources(Parent.**class**)

.child(Application.**class**)

.bannerMode(Banner.Mode.OFF)

.run(args);

**Application Events and Listeners**

Ngoài việc các sự kiện Spring Framwork thông thường chẳng hạn như ContextRefreshedEvent, SpringApplication gửi một số sự kiện ứng dụng bổ sung.

Các ứng dụng được gửi theo thứ tự sau, khi ứng dụng chạy:

* ApplicationStartingEvent : Đưuọc gửi vào lúc bắt đầu chạy nhưng trước khi xử lý bất kỳ, ngoài trừ việc đưang ký listener và người khởi tạo.
* ApplicationEnviromenPrepareEvent được gửi khi Enviroment được sử dụng trong Context được biết nhưng trước khi Context được tạo ra.
* ApplicationPrepareEvent : Được gửi ngay trước khi làm mới đưuọc bắt đầu nhưng sau khi định nghĩa bean đã được tải.
* ApplicationStartedEvent : được gửi khi Context dwuocj làm mới nhưng trước khi bất kỳ ứng dụng và trình chạy dòng lệnh được gọi.
* ApplicationReadyEvent : được gửi sau khi bất kỳ ứng dụng và người chạy dòng lệnh nào được gọi. Nó chỉ ra rằng ứng dụng đã sẵn sang cho các yêu cầu dịch vụ
* ApplicationFailedEvent : được gửi nếu có ngoại lệ khi khởi động.

Application events được gửi bằng cách sử dụng cơ chế publishing Spring Framework event. Một phần của cơ chế này đảm bảo rằng một sự kiện được xuất bản cho người nghe trong ngữ cảnh con cũng được publishing cho listener trong bất kỳ bối cảnh nào.

Do đó nếu sử dụng một hệ thống phân cấp SpringApplication , 1 listener có thể nhận được nhiều phiên bản của cùng một sự kiện ứng dụng.

**Web Enviroment**

SpringApplication cố gắng tạo đúng loại ApplicationContext thay cho bạn. Thuật toán được sử dụng để xác định một WebApplicationType khá đơn giản.

* Nếu Spring MVC có mặt một AnnotationConfigServletWebServerApplicationContext được sử dụng.
* Nếu Spring MVC không có mặt và Spring WebFlux hiện diện, một AnnotationConfigReactiveWebServerApplicationContext được sử dụng.
* Nếu ko , AnnotationConfigApplicationContext được sử dụng.

Nếu đang sử dụng Spring MVC và WebClient mới từ Spring WebFlux trong cùng một ứng dụng , Spring MVC sẽ được sử dụng theo mặc định. Bạn có thể ghi đè dễ dàng bằng cách gọi setWebApplicationType ( WebApplicationType)

Cũng có thể kiểm soát hoàn toàn kiểu ApplicationContext được sử dụng bằng cách gọi setApplicationContextClass

**Accessing Application Arguments**

Nếu cần truy cập vào các đối số ứng dụng đã được chuyển đến SpringApplication.run bạn có thể inject org.springframework.boot.ApplicationArguments

Nó cung cấp quyền truy cập vào cả 2 đối số String[] thô cũng như các tùy chọn đc phân tích cú phapf và các đối số không tùy chọn, như trong ví dụ sau :

**import** org.springframework.boot.\*;

**import** org.springframework.beans.factory.annotation.\*;

**import** org.springframework.stereotype.\*;

*@Component*

**public** **class** MyBean {

*@Autowired*

**public** MyBean(ApplicationArguments args) {

**boolean** debug = args.containsOption("debug");

List<String> files = args.getNonOptionArgs();

*// if run with "--debug logfile.txt" debug=true, files=["logfile.txt"]*

}

}

**Using the ApplicationRunner or CommandLineRunner**

Nếu cần chạy một số specific code khi SpringApplication đã bắt đầu, có thể thực hiện các Interface ApplicationRunner hoặc CommandLineRunner

Cả 2 giao diện hoạt động theo cùng một cách và cung cấp phương thức chạy duy nhất, được gọi ngay trước khi SpringApplication.run() hoàn thành