Trong bài hôm nay chúng ta sẽ tìm hiểu nốt 2 khái niệm @Configuration và @Bean để hoàn thiện phần căn bản của Spring Boot.

**Cài đặt**

*pom.xml*

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<packaging>pom</packaging>

<parent>

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

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.5.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<groupId>me.loda.spring</groupId>

<artifactId>spring-boot-learning</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>spring-boot-learning</name>

<description>Everything about Spring Boot</description>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<!--spring mvc, rest-->

<dependency>

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

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

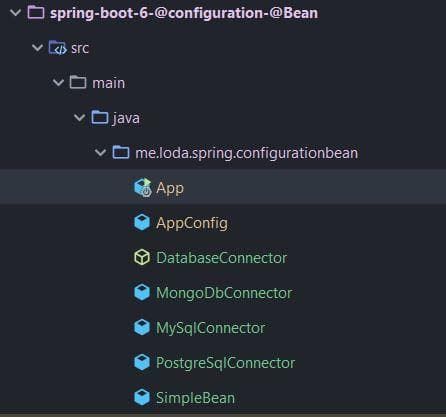
</plugin>

</plugins>

</build>

</project>

Cấu trúc thư mục:



**@Configuration và @Bean**

@Configuration là một Annotation đánh dấu trên một Class cho phép **Spring Boot** biết được đây là nơi định nghĩa ra các *Bean*.

@Bean là một Annotation được đánh dấu trên các method cho phép **Spring Boot** biết được đây là *Bean* và sẽ thực hiện đưa *Bean* này vào Context.

@Bean sẽ nằm trong các class có đánh dấu @Configuration.

Ví dụ:

*SimpleBean.java*

/\*\*

\* Một class cơ bản, không sử dụng `@Component`

\*/

public class SimpleBean {

private String username;

public SimpleBean(String username) {

setUsername(username);

}

@Override

public String toString() {

return "This is a simple bean, name: " + username;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

}

*AppConfig.java*

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class AppConfig {

@Bean

SimpleBean simpleBeanConfigure(){

// Khởi tạo một instance của SimpleBean và trả ra ngoài

return new SimpleBean("loda");

}

}

*App.java*

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

@SpringBootApplication

public class App {

public static void main(String[] args) {

ApplicationContext context = SpringApplication.run(App.class, args);

// Lấy ra bean SimpleBean trong Context

SimpleBean simpleBean = context.getBean(SimpleBean.class);

// In ra màn hình

System.out.println("Simple Bean Example: " + simpleBean.toString());

}

}

Output:

Simple Bean Example: This is a simple bean, name: loda

Bạn sẽ thấy là SimpleBean là một object được quản lý trong Context của **Spring Boot**, mặc dù trong bài này, chúng ta không hề sử dụng tới các khái niệm @Component.

**In Background**

Đằng sau chương trình, **Spring Boot** lần đầu khởi chạy, ngoài việc đi tìm các @Component thì nó còn làm một nhiệm vụ nữa là tìm các class @Configuration.

1. Đi tìm class có đánh dấu @Configuration
2. Tạo ra đối tượng từ class có đánh dấu @Configuration
3. tìm các method có đánh dấu @Bean trong đối tượng vừa tạo
4. Thực hiện gọi các method có đánh dấu @Bean để lấy ra các *Bean* và đưa vào `Context.

Ngoài ra, về bản chất, @Configuration cũng là @Component. Nó chỉ khác ở ý nghĩa sử dụng. (Giống với việc class được đánh dấu @Service chỉ nên phục vụ logic vậy).

@Target({ElementType.TYPE})

@Retention(RetentionPolicy.RUNTIME)

@Documented

@Component // Nó được đánh dấu là Component

public @interface Configuration {

@AliasFor(

annotation = Component.class

)

String value() default "";

}

**Có ý nghĩa gì?**

Nhiều bạn sẽ tự hỏi rằng @Configuration và @Bean sẽ có ý nghĩa gì khi chúng ta đã có @Component? Sao không đánh dấu SimpleBean là @Component cho nhanh?

Các bạn thắc mắc rất đúng, và việc sử dụng @Component cũng hoàn toàn ổn.

Thông thường thì các class được đánh dấu @Component đều có thể tạo tự động và inject tự động được.

Tuy nhiên trong thực tế, nếu một Bean có quá nhiều logic để khởi tạo và cấu hình, thì chúng ta sẽ sử dụng @Configuration và @Bean để tự tay tạo ra Bean. Việc tự tay tạo ra Bean như này có thể hiểu phần nào là chúng ta đang *config* cho chương trình.

**Ví dụ**

Chúng ta sẽ ví dụ với việc cấu hình kết nối tới Database. Đây vẫn là một ví dụ hết sức đơn giản.

Tạo ra một Abstract Class DatabaseConnector chịu trách nhiệm kết nối tới Database.

*DatabaseConnector.java*

public abstract class DatabaseConnector {

private String url;

/\*\*

\* Hàm này có nhiệm vụ Connect tới một Database bất kỳ

\*/

public abstract void connect();

public String getUrl() {

return url;

}

public void setUrl(String url) {

this.url = url;

}

}

Kế thừa class này có 3 class đại diện: MySqlConnector, PostgreSqlConnector, MongoDbConnector

*MongoDbConnector.java*

public class MongoDbConnector extends DatabaseConnector {

@Override

public void connect() {

System.out.println("Đã kết nối tới Mongodb: " + getUrl());

}

}

*MySqlConnector.java*

public class MySqlConnector extends DatabaseConnector {

@Override

public void connect() {

System.out.println("Đã kết nối tới Mysql: " + getUrl());

}

}

*PostgreSqlConnector.java*

public class PostgreSqlConnector extends DatabaseConnector{

@Override

public void connect() {

System.out.println("Đã kết nối tới Postgresql: " + getUrl());

}

}

Tạo ra Bean trong AppConfig

*AppConfig.java*

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class AppConfig {

@Bean("mysqlConnector")

DatabaseConnector mysqlConfigure() {

DatabaseConnector mySqlConnector = new MySqlConnector();

mySqlConnector.setUrl("jdbc:mysql://host1:33060/loda");

// Set username, password, format, v.v...

return mySqlConnector;

}

@Bean("mongodbConnector")

DatabaseConnector mongodbConfigure() {

DatabaseConnector mongoDbConnector = new MongoDbConnector();

mongoDbConnector.setUrl("mongodb://mongodb0.example.com:27017/loda");

// Set username, password, format, v.v...

return mongoDbConnector;

}

@Bean("postgresqlConnector")

DatabaseConnector postgresqlConfigure(){

DatabaseConnector postgreSqlConnector = new PostgreSqlConnector();

postgreSqlConnector.setUrl("postgresql://localhost/loda");

// Set username, password, format, v.v...

return postgreSqlConnector;

}

}

Chạy thử:

*App.java*

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

@SpringBootApplication

public class App {

public static void main(String[] args) {

ApplicationContext context = SpringApplication.run(App.class, args);

DatabaseConnector mysql = (DatabaseConnector) context.getBean("mysqlConnector");

mysql.connect();

DatabaseConnector mongodb = (DatabaseConnector) context.getBean("mongodbConnector");

mongodb.connect();

DatabaseConnector postgresql = (DatabaseConnector) context.getBean("postgresqlConnector");

postgresql.connect();

}

}

Output:

Đã kết nối tới Mysql: jdbc:mysql://host1:33060/loda

Đã kết nối tới Mongodb: mongodb://mongodb0.example.com:27017/loda

Đã kết nối tới Postgresql: postgresql://localhost/loda

Chúng ta tạo ra DatabaseConnector phục vụ cho nhiều ngữ cảnh.

**@Bean có tham số**

Nếu method được đánh dấu bởi @Bean có tham số truyền vào, thì **Spring Boot** sẽ tự inject các Bean đã có trong Context vào làm tham số.

Ví dụ:

*AppConfig.java*

@Configuration

public class AppConfig {

@Bean

SimpleBean simpleBeanConfigure(){

// Khởi tạo một instance của SimpleBean và trả ra ngoài

return new SimpleBean("loda");

}

@Bean("mysqlConnector")

DatabaseConnector mysqlConfigure(SimpleBean simpleBean) { // SimpleBean được tự động inject vào.

DatabaseConnector mySqlConnector = new MySqlConnector();

mySqlConnector.setUrl("jdbc:mysql://host1:33060/" + simpleBean.getUsername());

// Set username, password, format, v.v...

return mySqlConnector;

}

}

**Thực tế**

Trong thực tế, việc sử dụng @Configuration là hết sức cần thiết, và nó đóng vai trò là nơi cấu hình cho toàn bộ ứng dụng của bạn. Một Ứng dụng sẽ có nhiều class chứa @Configuration và mỗi class sẽ đảm nhận cấu hình một bộ phận gì đó trong ứng dụng.

Ví dụ đây là một đoạn code cấu hình cho **Spring Security**

@Configuration

@EnableWebSecurity

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.antMatchers("/", "/home").permitAll()

.anyRequest().authenticated()

.and()

.formLogin()

.loginPage("/login")

.permitAll()

.and()

.logout()

.permitAll();

}

@Bean

@Override

public UserDetailsService userDetailsService() {

UserDetails user =

User.withDefaultPasswordEncoder()

.username("user")

.password("password")

.roles("USER")

.build();

return new InMemoryUserDetailsManager(user);

}

}