**Spring Bean**

1. **Bean Definition**

* Spring Bean không có gì đặc biệt, bất kỳ object nào trong Spring framework mà chúng ta khởi tạo thông qua Spring container được gọi là Spring Bean. Bất kỳ class Java POJO thông thường nào cũng có thể là Spring Bean nếu nó được cấu hình để khởi tạo thông qua container bằng cách cung cấp thông tin metadata cấu hình
* Trọng tâm của cách cấu hình **Bean** bằng Java code là dùng 2 annotation sau:
* **org.springframework.context.annotation.Bean**: annotation ở phương thức
* **org.springframework.context.annotation.Configuration**: annotation ở class
* Annotation **@Bean** được dùng để chỉ định một phương thức sẽ khởi tạo một bean (hay nói cách khác là trả về một instance của bean). Mặc định tên của bean trùng với tên của phương thức này. Chúng ta cũng có thể chỉ định tên của bean thông qua thuộc tính name của **@Bean**.
* Các phương thức dùng annotation **@Bean** thì đầu class chứa phương thức đó phải có annotation **@Component** hoặc **@Configuration**
* Dưới đây là ví dụ về định nghĩa một bean với **@Bean** và **@Configuration**.
* Step 1: import dependency spring-context

<dependency>

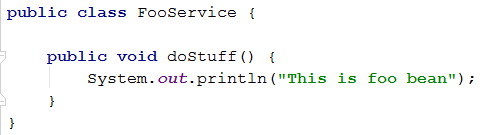
<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

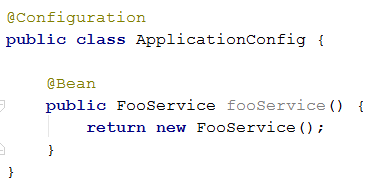
<version>5.1.2.RELEASE</version>

</dependency>

* Step 2: Tạo Bean



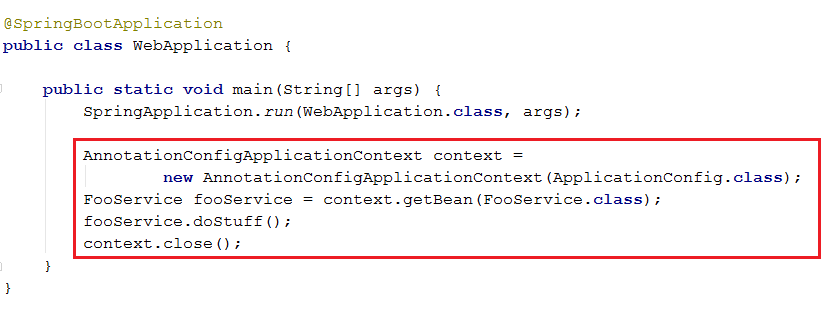
* Step 3: Tạo class ApplicationConfig với annotation @Configuration để chỉ định Bean



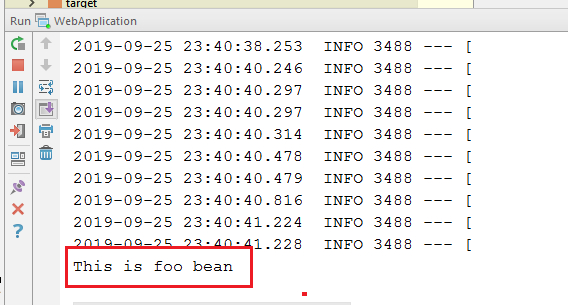
* Step 4:

Khởi tạo Spring IoC Container thông qua interface ApplicationContext với implementation là AnnotationConfigApplicationContext. Khi Spring IoC Container được khởi tạo, các bean cũng sẽ được khởi tạo theo.

Chúng ta sẽ lấy ra được các bean thông qua phương thức getBean(Bean.class) của ApplicationContext.



* Kết quả



1. **Bean Scope**

* Bean scope được hiểu là phạm vi hoạt động của các instance của bean**.**
* Khi định nghĩa một bean trong Spring, chúng ta còn phải định nghĩa scope của bean. Việc định nghĩa scope có thể thực hiện thông qua việc sử dụng thuộc tính tên là “**scope**”.
* Ví dụ, khi bean phải tạo mới mỗi lần cần sử dụng, thuộc tính scope sẽ là “**prototype**”. Mặt khác, khi bean luôn luôn trả về một instance giống nhau khi sử dụng, thuộc tính scope sẽ là “**singleton**”.
* Có tất cả 6 loại bean scope:
* singleton
* prototype
* request
* session
* application
* websocket
* Trong đó, 4 scope là request, session, application và websocket chỉ hoạt động bên trong ngữ cảnh của các ApplicationContext implementation cho ứng dụng web như XmlWebApplicationContext hay AnnotationConfigWebApplicationContext.
* Để khai báo bean scope, chúng ta thường sử dụng annotation org.springframework.context.annotation.Scope với 2 thuộc tính:
* value: tên của scope
* proxyMode=ScopedProxyMode.TARGET\_CLASS: được dùng để tạo AOP proxy của bean. Chúng ta khai báo giá trị này khi muốn inject một bean có scope request, session, application vào một bean có scope singleton, prototype. Hoặc inject một bean có scope prototype vào bean có scope singleton.

**2.1 Singleton scope**

* Một bean chỉ có một duy nhất một instance trong một Spring IoC Container. Spring IoC container khởi tạo instance của singleton bean tại thời điểm startup của ứng dụng. Sau đó, lưu trữ instance của bean vào một bộ nhớ cache. Với mỗi request, Spring IoC container sẽ trả về instance được lấy từ bộ nhớ cache này.

**2.2 Prototype scope**

- Với mỗi request, Spring IoC Container khởi tạo một instance mới của bean.

@Bean

@Scope(value = "prototype")

public FooService **fooService**() {

return new **FooService**();

}

**2.3 Request scope**

- Với mỗi HTTP requet, Spring IoC container khởi tạo một instance mới của bean. Instance này sẽ hoạt động trong suốt vòng đời của HTTP request.

@Component

@Scope(value = "request", proxyMode = ScopedProxyMode.TARGET\_CLASS)

public class **LoginAction** {

}

* Kể từ phiên bản Spring 4.3, chúng ta có thể đơn giản hóa cách khai báo trên như sau:

@Component

@RequestScope

public class **LoginAction** {

}

**2.4 Session scope**

- Với mỗi HTTP session, Spring IoC container khởi tạo một instance mới của bean. Instance này sẽ hoạt động trong suốt vòng đời của HTTP session.

@Component

@SessionScope

public class **Cart** {

}

**2.5 Application scope**

- Instance của bean sẽ hoạt động trong suốt vòng đời của một ứng dụng web (thực chất là vòng đời của một ServletContext).

@Component

@ApplicationScope

public class **AppSettings** {

}

**2.6 Websocket scope**

- Instance của bean sẽ hoạt động trong suốt vòng đời của một WebSocket.

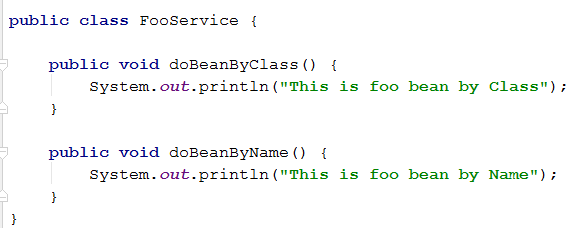
1. **Component**

* Đây chính là các bean được sử dụng thường xuyên nhất trong các ứng dụng Spring
* Như chúng ta đã biết, để định nghĩa một bean trong Spring, chúng ta sẽ cần sử dụng đến 2 annotation là @Configuration và @Bean. Bên cạnh đó, Spring còn cho phép chúng ta tự động định nghĩa một bean thông qua cơ chế **classpath scanning**. Với cách định nghĩa ngầm này, các bean sẽ được gọi là các **component**.
* Spring đã cung cấp 4 **stereotype annotation** thuộc package **org.springframework.stereotype** là:
* **@Component**: đây là annotation tổng quát để chỉ định 1 lớp là component.
* **@Controller, @RestController**: dùng để chỉ định 1 lớp thuộc *presentation layer* là component.
* **@Service**: dùng để chỉ định 1 lớp thuộc *business layer* là component.
* **@Repository**: dùng để chỉ định 1 lớp thuộc *persistence layer* là component.
* Cả 5 stereotype annotation này đều là annotation mức **class**.

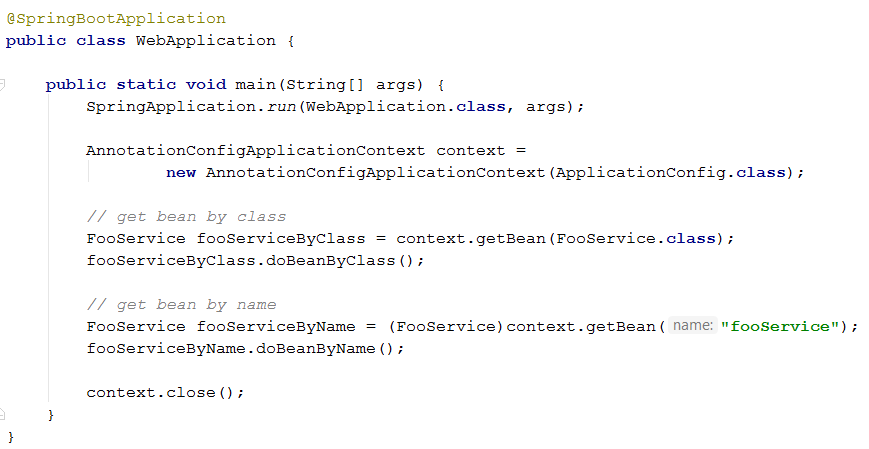
1. **getBean()**

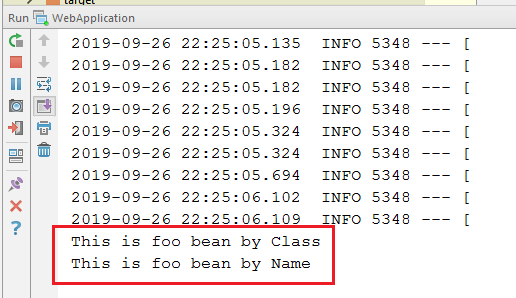
* Có nhiều cách để get Bean, thông dụng nhất là get bean thông qua name và get bean thông qua class
* Dưới đây là ví dụ get bean by name và get bean by class

FooService.java



WebApplication.java



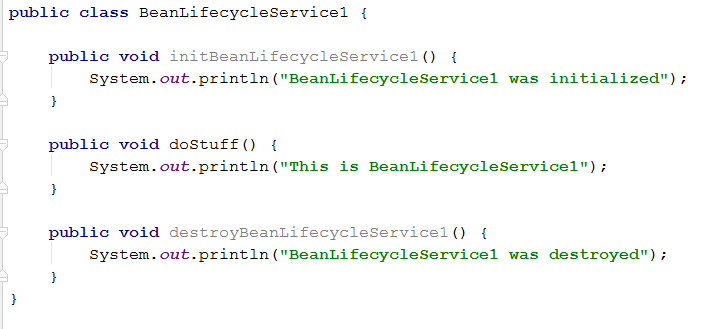


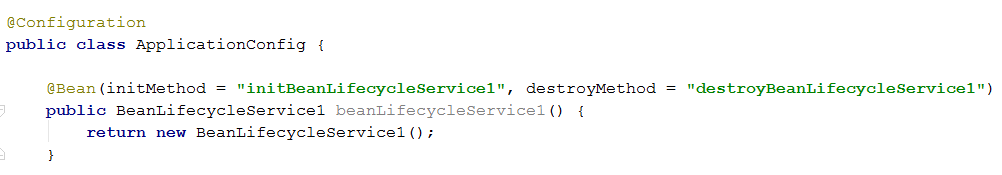
* **Cân nhắc khi sử dụng getBean()**
* Phương thức getBean() được sử dụng thường xuyên thông qua ApplicationContext. Thông thường ta nên hạn chế sử dụng getBean() một cách trực tiếp
* Bean nên được quản lý bởi Ioc container, dùng cơ chế dependency injection để get bean thì tốt hơn cách gọi trực tiếp getBean()

1. **Bean Lifecycle**

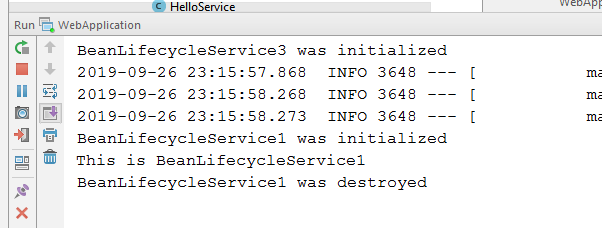
* Spring Bean hỗ trợ 2 loại lifecycle callback sau:
* Initialization method: phương thức này sẽ được gọi ngay sau khi Spring IoC container khởi tạo bean và inject các dependency của bean thành công, và ngay trước khi instance của bean được request.
* Destruction method: phương thức này sẽ được gọi ngay trước khi Spring IoC Container chứa bean đó bị hủy bỏ/đóng.
* Có 3 cách để chúng ta định nghĩa lifecycle callback:
* Chỉ định initialization method và destruction method thông qua 2 thuộc tính **initMethod** và **destroyMethod** của @Bean => không thể áp dụng cho component (@Component, @Controller, @Service, …)
* Chỉ định initialization method và destruction method thông qua 2 annotation **@PostConstruct** và **@PreDestroy**
* Bean implement 2 interface **InitializingBean** và **DisposableBean**, rồi override 2 phương thức **afterPropertiesSet()** và **destroy().**

**5.1 initMethod và destroyMethod**

****

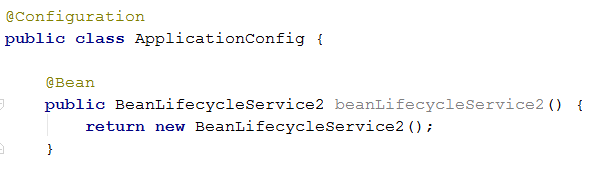


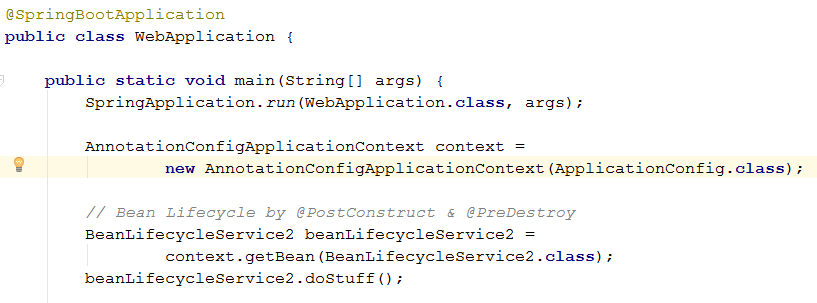


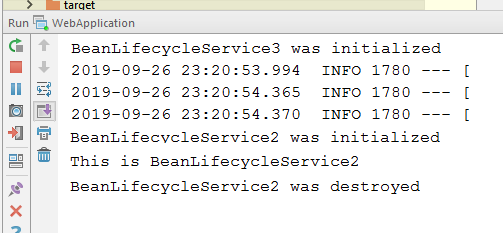


**5.2 @PostConstruct và @PreDestroy**

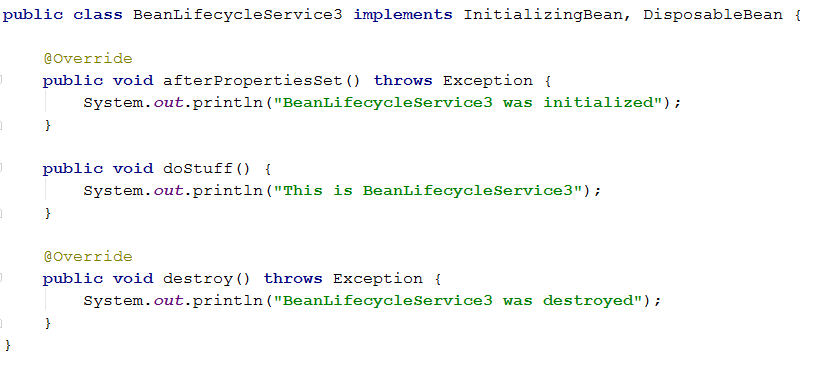
****

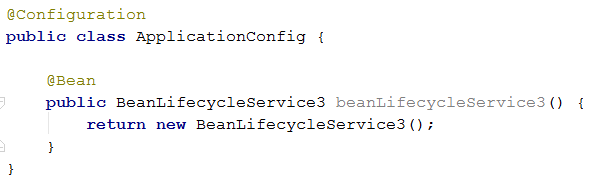
****

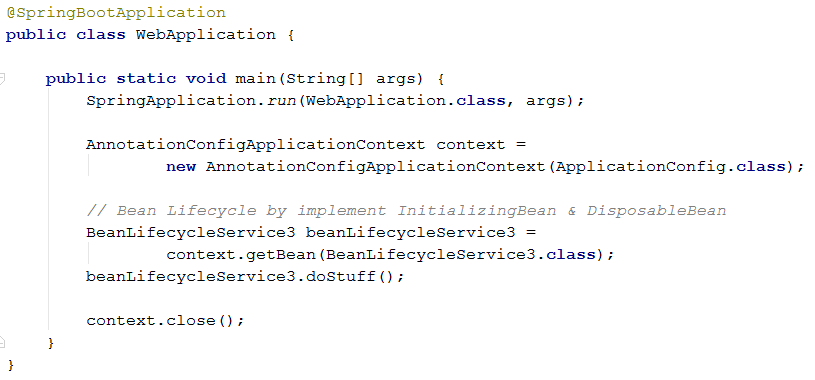


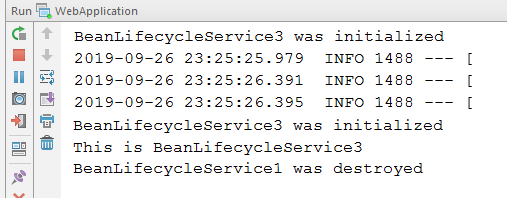


**5.3 InitializingBean và DisposableBean**









* **Tài liệu tham khảo**
* <https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-java-bean-annotation>
* <https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-factory-scopes>
* <https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-classpath-scanning>
* <https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-factory-lifecycle>