

# 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:
  - o **org.springframework.context.annotation.Bean**: annotation ở phương thức
  - o **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

```
public class FooService {

    public void doStuff() {
        System.out.println("This is foo bean");
    }
}
```

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

```
@Configuration
public class ApplicationConfig {

    @Bean
    public FooService fooService() {
        return new FooService();
    }
}
```

- 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.

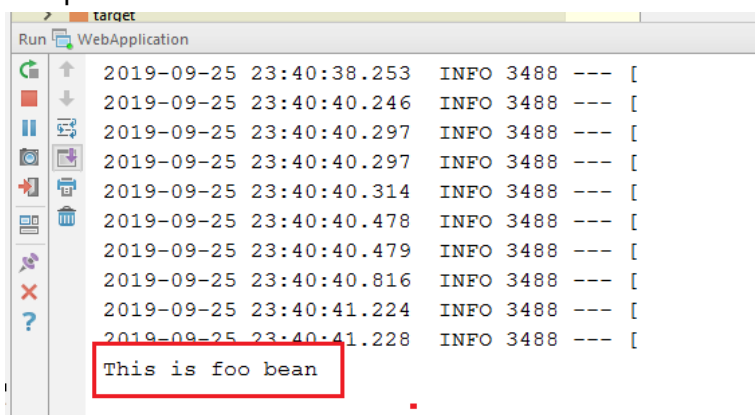
Chúng ta sẽ lấy ra được các bean thông qua phương thức `getBean(Beans.class)` của `ApplicationContext`.

```
@SpringBootApplication
public class WebApplication {

    public static void main(String[] args) {
        SpringApplication.run(WebApplication.class, args);

        AnnotationConfigApplicationContext context =
            new AnnotationConfigApplicationContext(ApplicationConfig.class);
        FooService fooService = context.getBean(FooService.class);
        fooService.doStuff();
        context.close();
    }
}
```

- Kết quả



```
target
Run WebApplication
2019-09-25 23:40:38.253 INFO 3488 --- [
2019-09-25 23:40:40.246 INFO 3488 --- [
2019-09-25 23:40:40.297 INFO 3488 --- [
2019-09-25 23:40:40.297 INFO 3488 --- [
2019-09-25 23:40:40.314 INFO 3488 --- [
2019-09-25 23:40:40.478 INFO 3488 --- [
2019-09-25 23:40:40.479 INFO 3488 --- [
2019-09-25 23:40:40.816 INFO 3488 --- [
2019-09-25 23:40:41.224 INFO 3488 --- [
2019-09-25 23:40:41.228 INFO 3488 --- [
This is foo bean
```

## 2. 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:
  - o singleton
  - o prototype
  - o 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 request, 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.

## 3. 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:
  - o **@Component**: đây là annotation tổng quát để chỉ định 1 lớp là component.
  - o **@Controller**, **@RestController**: dùng để chỉ định 1 lớp thuộc *presentation layer* là component.
  - o **@Service**: dùng để chỉ định 1 lớp thuộc *business layer* là component.
  - o **@Repository**: dùng để chỉ định 1 lớp thuộc *persistence layer* là component.
- Cả 5 stereotype annotation này đều là annotation mức **class**.

## 4. getBean()

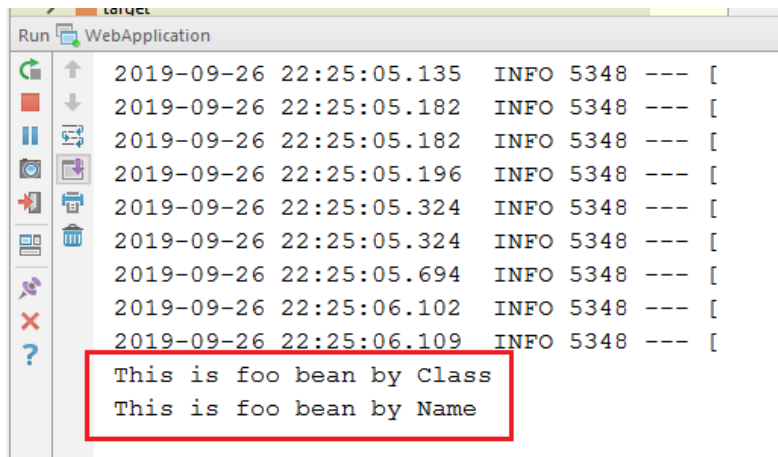
- Có nhiều cách để get Bean, thông dụng nhất là get bean thông qua name và get bean thông qua class
- Dưới đây là ví dụ get bean by name và get bean by class

FooService.java

```
public class FooService {  
  
    public void doBeanByClass() {  
        System.out.println("This is foo bean by Class");  
    }  
  
    public void doBeanByName() {  
        System.out.println("This is foo bean by Name");  
    }  
}
```

WebApplication.java

```
@SpringBootApplication  
public class WebApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(WebApplication.class, args);  
  
        AnnotationConfigApplicationContext context =  
            new AnnotationConfigApplicationContext(ApplicationConfig.class);  
  
        // get bean by class  
        FooService fooServiceByClass = context.getBean(FooService.class);  
        fooServiceByClass.doBeanByClass();  
  
        // get bean by name  
        FooService fooServiceByName = (FooService) context.getBean(name: "fooService");  
        fooServiceByName.doBeanByName();  
  
        context.close();  
    }  
}
```



```
Run WebApplication
2019-09-26 22:25:05.135 INFO 5348 --- [
2019-09-26 22:25:05.182 INFO 5348 --- [
2019-09-26 22:25:05.182 INFO 5348 --- [
2019-09-26 22:25:05.196 INFO 5348 --- [
2019-09-26 22:25:05.324 INFO 5348 --- [
2019-09-26 22:25:05.324 INFO 5348 --- [
2019-09-26 22:25:05.694 INFO 5348 --- [
2019-09-26 22:25:06.102 INFO 5348 --- [
2019-09-26 22:25:06.109 INFO 5348 --- [
This is foo bean by Class
This is foo bean by Name
```

#### ❖ **Cần 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()`

## 5. 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

```
public class BeanLifecycleService1 {

    public void initBeanLifecycleService1() {
        System.out.println("BeanLifecycleService1 was initialized");
    }

    public void doStuff() {
        System.out.println("This is BeanLifecycleService1");
    }

    public void destroyBeanLifecycleService1() {
        System.out.println("BeanLifecycleService1 was destroyed");
    }
}

@Configuration
public class ApplicationConfig {

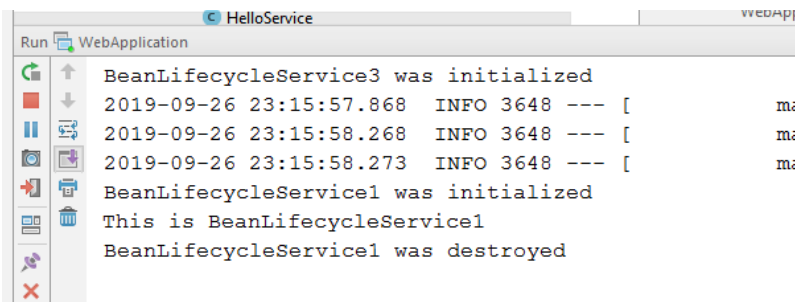
    @Bean(initMethod = "initBeanLifecycleService1", destroyMethod = "destroyBeanLifecycleService1")
    public BeanLifecycleService1 beanLifecycleService1() {
        return new BeanLifecycleService1();
    }
}

@SpringBootApplication
public class WebApplication {

    public static void main(String[] args) {
        SpringApplication.run(WebApplication.class, args);

        AnnotationConfigApplicationContext context =
            new AnnotationConfigApplicationContext(ApplicationConfig.class);

        // Bean Lifecycle by initMethod & destroyMethod
        BeanLifecycleService1 beanLifecycleService1 =
            context.getBean(BeanLifecycleService1.class);
        beanLifecycleService1.doStuff();
    }
}
```



## 5.2 @PostConstruct và @PreDestroy

```
public class BeanLifecycleService2 {

    @PostConstruct
    public void initBeanLifecycleService2() {
        System.out.println("BeanLifecycleService2 was initialized");
    }

    public void doStuff() {
        System.out.println("This is BeanLifecycleService2");
    }

    @PreDestroy
    public void destroyBeanLifecycleService2() {
        System.out.println("BeanLifecycleService2 was destroyed");
    }
}
```

@Configuration

```
public class ApplicationConfig {
```

@Bean

```
    public BeanLifecycleService2 beanLifecycleService2() {
        return new BeanLifecycleService2();
    }
}
```

@SpringBootApplication

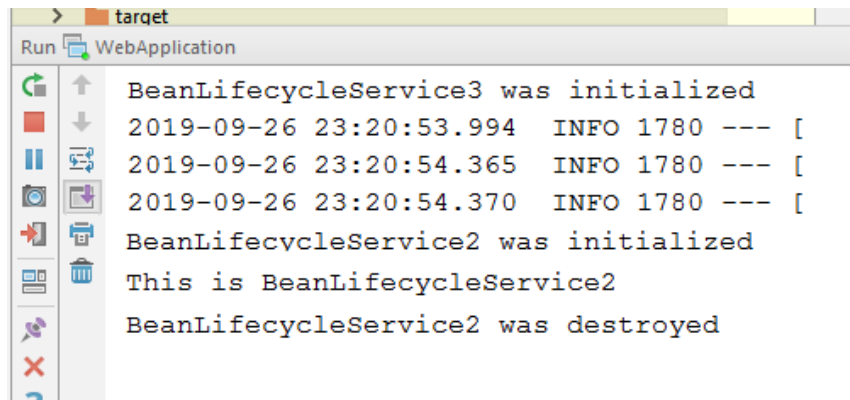
```
public class WebApplication {
```

```
    public static void main(String[] args) {
        SpringApplication.run(WebApplication.class, args);

        AnnotationConfigApplicationContext context =
            new AnnotationConfigApplicationContext(ApplicationConfig.class);

        // Bean Lifecycle by @PostConstruct & @PreDestroy
        BeanLifecycleService2 beanLifecycleService2 =
            context.getBean(BeanLifecycleService2.class);
        beanLifecycleService2.doStuff();
    }
}
```





### 5.3 InitializingBean và DisposableBean

```

public class BeanLifecycleService3 implements InitializingBean, DisposableBean {

    @Override
    public void afterPropertiesSet() throws Exception {
        System.out.println("BeanLifecycleService3 was initialized");
    }

    public void doStuff() {
        System.out.println("This is BeanLifecycleService3");
    }

    @Override
    public void destroy() throws Exception {
        System.out.println("BeanLifecycleService3 was destroyed");
    }
}

@Configuration
public class ApplicationConfig {

    @Bean
    public BeanLifecycleService3 beanLifecycleService3() {
        return new BeanLifecycleService3();
    }
}

```

```

@SpringBootApplication
public class WebApplication {

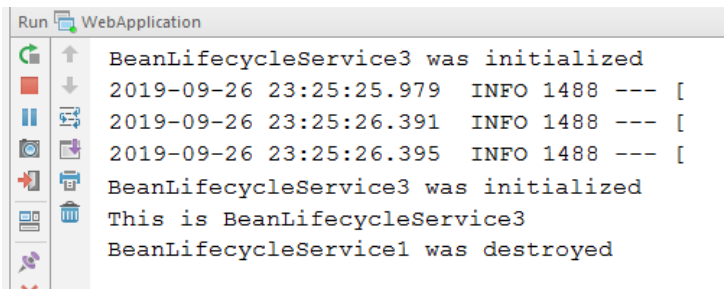
    public static void main(String[] args) {
        SpringApplication.run(WebApplication.class, args);

        AnnotationConfigApplicationContext context =
            new AnnotationConfigApplicationContext(ApplicationConfig.class);

        // Bean Lifecycle by implement InitializingBean & DisposableBean
        BeanLifecycleService3 beanLifecycleService3 =
            context.getBean(BeanLifecycleService3.class);
        beanLifecycleService3.doStuff();

        context.close();
    }
}

```



```

Run WebApplication
BeanLifecycleService3 was initialized
2019-09-26 23:25:25.979 INFO 1488 --- [
2019-09-26 23:25:26.391 INFO 1488 --- [
2019-09-26 23:25:26.395 INFO 1488 --- [
BeanLifecycleService3 was initialized
This is BeanLifecycleService3
BeanLifecycleService1 was destroyed

```

#### ❖ 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>