「Spring Boot #3」 Spring Bean Life Cycle + @PostConstruct và @PreDestroy

---

Giới thiệu

Trong các bài trước, các bạn đã hiểu các khái niệm cơ bản về `Bean` và cách inject nó trong Spring Boot bằng `@Component` + `@Autowired`

1. 「Spring Boot #1」 Hướng dẫn @Component và @Autowired
2. 「Spring Boot #2」 @Autowired - @Primary - @Qualifier

Hôm nay chúng ta sẽ tìm hiểu kỹ hơn về vòng đời của `Bean`.

Cài đặt

```
<?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/4.0.0 http://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>
```

Cấu trúc thư mục:

@PostConstruct

`@PostConstruct` được đánh dấu trên một method duy nhất bên trong `Bean`. `IoC Container` hoặc `ApplicationContext` sẽ gọi hàm này **sau khi** một `Bean` được tạo ra và quản lý.

```
@Component
public class Girl {

    @PostConstruct
    public void postConstruct(){
        System.out.println("\t>> Đối tượng Girl sau khi khởi tạo xong sẽ chạy hàm này");
    }
}
```

@PreDestroy

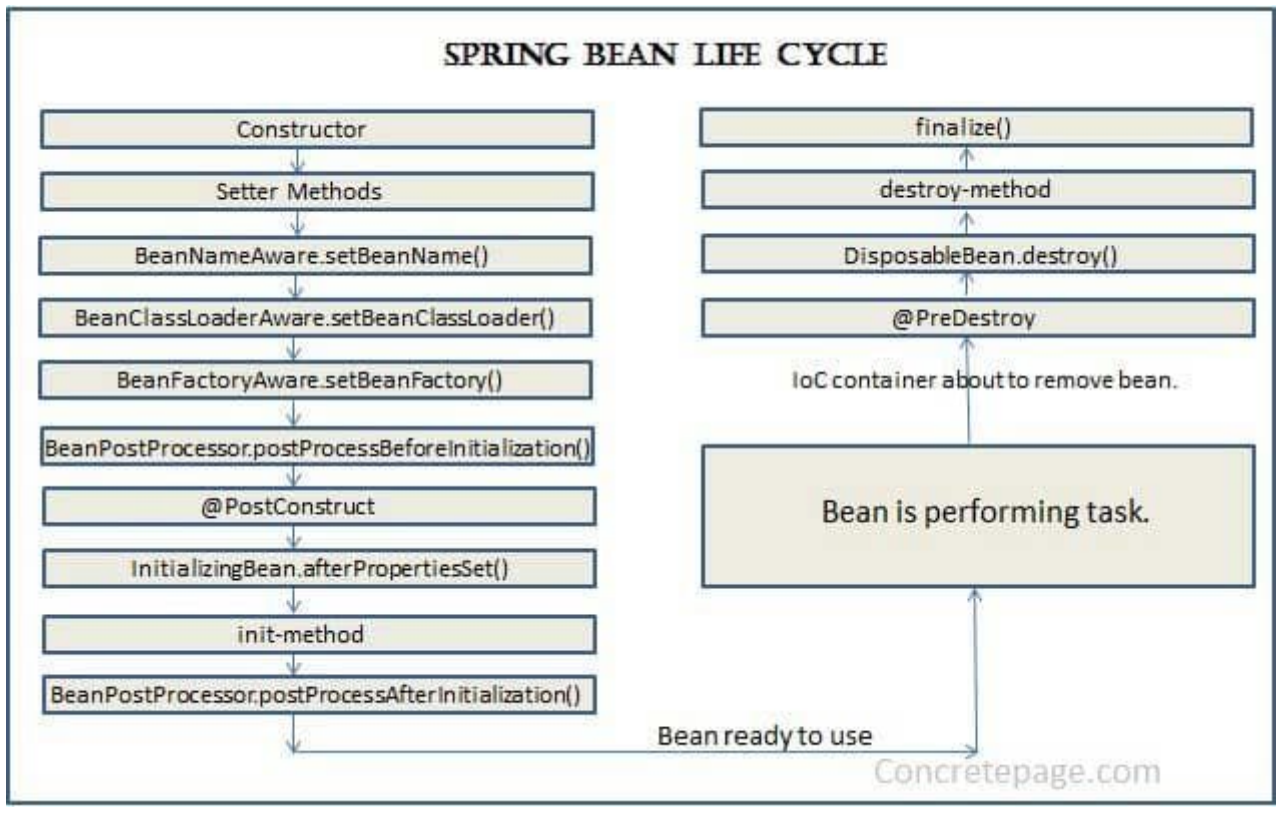
`@PreDestroy` được đánh dấu trên một method duy nhất bên trong `Bean`. `IoC Container` hoặc `ApplicationContext` sẽ gọi hàm này **trước khi** một `Bean` bị xóa hoặc không được quản lý nữa.

```
@Component
public class Girl {

    @PreDestroy
    public void preDestroy(){
        System.out.println("\t>> Đối tượng Girl trước khi bị destroy thì chạy hàm này");
    }
}
```

Bean Life Cycle

Spring Boot từ thời điểm chạy lần đầu tới khi *shutdown* thì các `Bean` nó quản lý sẽ có một vòng đời được biểu diễn như ảnh dưới đây:



Nhìn có vẻ loằng ngoằng, trong series căn bản này, bạn có lẽ sẽ chỉ cần hiểu như sau:

1. Khi `IoC Container` (`ApplicationContext`) tìm thấy một Bean cần quản lý, nó sẽ khởi tạo bằng `Constructor`
2. inject dependencies vào `Bean` bằng Setter, và thực hiện các quá trình cài đặt khác vào `Bean` như `setBeanName` , `setBeanClassLoader` , v.v..

3. Hàm đánh dấu `@PostConstruct` được gọi
4. Tiền xử lý sau khi `@PostConstruct` được gọi.
5. `Bean` sẵn sàng để hoạt động
6. Nếu `IoC Container` không quản lý bean nữa hoặc bị shutdown nó sẽ gọi hàm `@PreDestroy` trong `Bean`
7. Xóa `Bean`.

Ví dụ

Chúng ta tạo ra class `Girl` bao gồm:

```
import org.springframework.stereotype.Component;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;

@Component
public class Girl {

    @PostConstruct
    public void postConstruct(){
        System.out.println("\t>> Đối tượng Girl sau khi khởi tạo xong sẽ chạy hàm này");
    }

    @PreDestroy
    public void preDestroy(){
        System.out.println("\t>> Đối tượng Girl trước khi bị destroy thì chạy hàm này");
    }
}
```

In ra màn hình quá Spring Boot chạy lần đầu cho tới khi shutdown:

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication
public class App {
    public static void main(String[] args) {
        // ApplicationContext chính là container, chứa toàn bộ các Bean
        System.out.println("> Trước khi IoC Container được khởi tạo");
        ApplicationContext context = SpringApplication.run(App.class, args);
        System.out.println("> Sau khi IoC Container được khởi tạo");

        // Khi chạy xong, lúc này context sẽ chứa các Bean có đánh
        // dấu @Component.

        Girl girl = context.getBean(Girl.class);

        System.out.println("> Trước khi IoC Container destroy Girl");
        ((ConfigurableApplicationContext) context).getBeanFactory().destroyBean(girl);
        System.out.println("> Sau khi IoC Container destroy Girl");

    }
}
```

Output:

```
> Trước khi IoC Container được khởi tạo
> Trước khi IoC Container được khởi tạo
    >> Đối tượng Girl sau khi khởi tạo xong sẽ chạy hàm này
> Sau khi IoC Container được khởi tạo
> Trước khi IoC Container destroy Girl
    >> Đối tượng Girl trước khi bị destroy thì chạy hàm này
> Sau khi IoC Container destroy Girl
```

Bạn sẽ thấy dòng *"Trước khi IoC Container được khởi tạo"* được chạy 2 lần.

> *Điều này xảy ra bởi vì hàm* `App.main(args)` *được chạy 2 lần!*

Lần đầu là do chúng ta chạy.

Lần thứ hai là do **Spring Boot** chạy sau khi nó được gọi `SpringApplication.run(App.class, args)` . Đây là lúc mà **IoC Container** (`ApplicationContext`) được tạo ra và đi tìm `Bean` .

Ý nghĩa.

`@PostConstruct` và `@PreDestroy` là 2 Annotation cực kỳ ý nghĩa, nếu bạn nắm được vòng đời của một `Bean` , bạn có thể tận dụng nó để làm các nhiệm vụ riêng như setting, thêm giá trị mặc định trong thuộc tính sau khi tạo, xóa dữ liệu trước khi xóa, v.v.. Rất nhiều chức năng khác tùy theo nhu cầu.

Kết

Đây là một bài viết trong **Series làm chủ Spring Boot, từ zero to hero**

Như mọi khi, code được up tại Github