02: Spring Bean

Trần Văn Thịnh

Spring bean

Spring bean là gì

IoC và DI

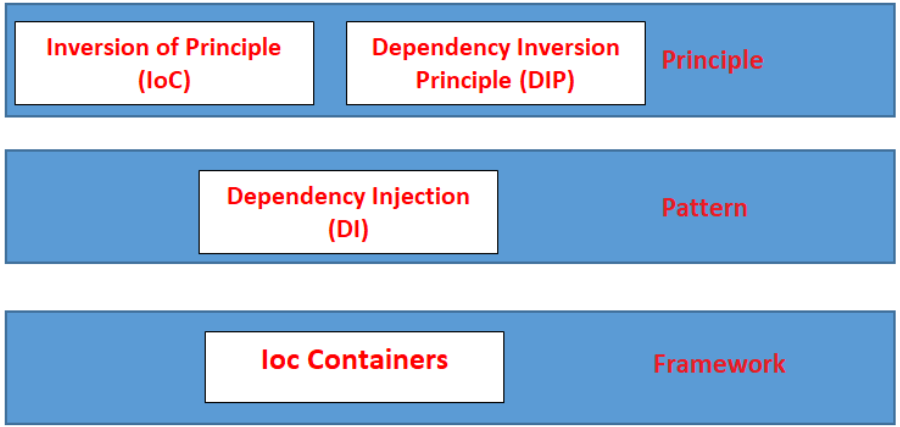
- Khái niệm IoC và DI
- Các cách thực hiện DI trong Spring

Spring IoC Container

Spring bean là gì

- Khái niệm quan trọng trong Spring
- Là một object được khởi tạo, lắp ráp, hủy bỏ và quản lý bởi Spring IoC Container
- Beans và các dependencies của nó được phản ánh trong dữ liệu cấu hình metadata, sử dụng bởi container

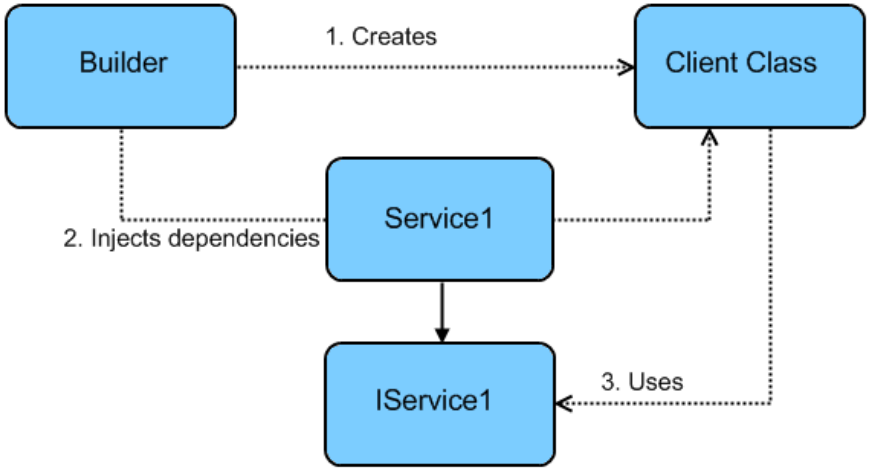
IoC và DI



Container về mặt kỹ thuật, mô tả các thành phần (component) cung cấp cơ sở hạ tầng cho các thành phần khác

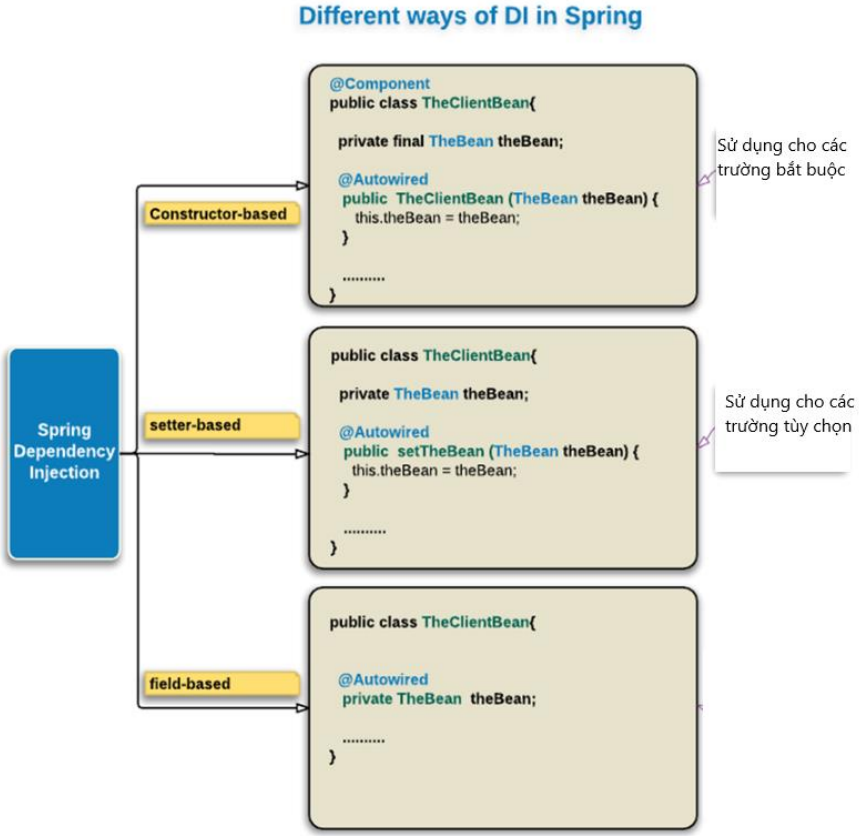
- IoC: một nguyên tắc (principle) trong phần mềm, chuyển quyền kiểm soát các objects hoặc các thành phần của một chương trình sang một container hay 1 framework
- Lợi ích:
  - Tách việc thực hiện một nhiệm vụ ra khỏi việc triển khai (implementation)
  - Chuyển đổi giữa các implementation khác nhau dễ dàng
  - Giúp chương trình đóng gói module dễ dàng
  - Giúp việc kiểm thử từng thành phần dễ dàng

IoC và DI

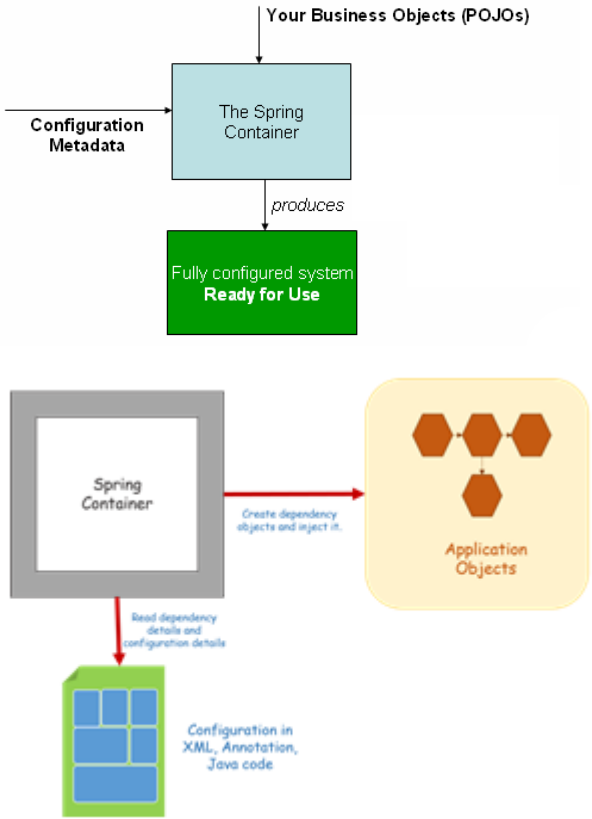


- DI (chèn, tiêm phụ thuộc) là một design pattern (mẫu thiết kế) thực thi IoC, trong đó điều khiển được đảo ngược là việc thiết lập các dependencies của một object
- Việc kết nối các đối tượng với các đối tượng khác, hoặc "tiêm‟ (inject) đối tượng vào các đối tượng khác, được thực hiện bởi một trình lắp ráp chứ không phải bởi chính các đối tượng.

IoC và DI

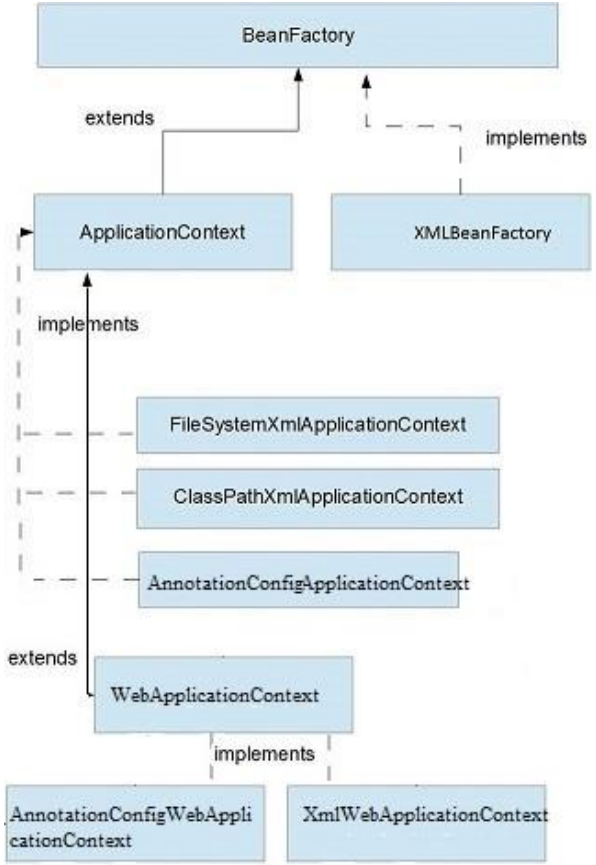

Different ways of DI in Spring
Spring Dependency Injection
Constructor-based
setter-based
field-based
@Component
public class TheClientBean{
private final TheBean theBean;
@Autowired
public TheClientBean (TheBean theBean) {
this.theBean = theBean;
}
.....
}
Sử dụng cho các trường bắt buộc
public class TheClientBean{
private TheBean theBean;
@Autowired
public setTheBean (TheBean theBean) {
this.theBean = theBean;
}
.....
}
Sử dụng cho các trường tùy chọn
public class TheClientBean{
@Autowired
private TheBean theBean;
.....
}

Spring (IoC) Container



- Spring (IoC) Container là trung tâm của hệ thống Spring, quản lý vòng đời hoàn chỉnh của mỗi bean từ khi tạo ra đến khi bị phá hủy
- Spring container sử dụng DI để quản lý bean

Spring (IoC) Container



- Đại diện cho Spring IoC Container:
  - BeanFactory: root interface, thao tác cơ bản để quản lý bean trong ứng dụng
  - ApplicationContext: sub-interface với thêm một số tính năng quan trọng:
    - Bean instantiation/wiring
    - Automatic BeanPostProcessor registration
    - Automatic BeanFactoryPostProcessor registration
    - Convenient MessageSource access (for i18n)
    - ApplicationEvent publication

CẤU HÌNH SPRING BEAN

- Làm thế nào để định nghĩa một bean trong Spring
  - Các cách khác nhau để định nghĩa Spring bean
  - Bean annotation trong Spring
- Spring bean properties
  - Làm thế nào để định nghĩa nhiều bean của cùng một class
- IoC tạo một instance của bean như thế nào

Làm thế nào để định nghĩa một bean trong Spring

- Sử dụng Annotations trong Spring – stereotype annotation with @ComponentScan (auto scan)

@Component	Generic annotation chỉ ra java class là một bean. Spring sử dụng cơ chế auto scan để tìm và đăng kí bean này vào context
@Repository	Kế thừa từ Component, chỉ ra class này là repository(tương tác với DB)
@Service	Kế thừa từ Component, chỉ ra class này là service, chứa tất cả business logic
@Controller	Sử dụng ở class level trong spring MVC, đánh dấu class này là một spring web controller, xử lý HTTP request

Làm thế nào để định nghĩa một bean trong Spring

- Sử dụng Annotations trong Spring – stereotype annotation with @ComponentScan (auto scan)

```
@Component
public class Student {
    private int Id;
    private String firstname;
    private String lastname;
    private String email;
    public Student() {
    }
    // Getters + Setters
}
```

Làm thế nào để định nghĩa một bean trong Spring

- Sử dụng Annotations trong Spring – @Bean với @Configuration

```
@Configuration
public class SpringConfig {
    @Bean
    public Student studentBean() {
        return new Student();
    }
    @Bean
    public Employee employeeBean() {
        return new Employee();
    }
    // Other spring beans ...
}
```

Spring bean properties

- class : java class của bean, bắt buộc
- id: id duy nhất của bean (unique bean identifier)
- name: cách khác để xác định bean identifier. Spring sử dụng id/name để xác định bean. Nếu các bean có cùng class thì sẽ cần sử dụng thêm
- Scope: scope của bean object
- Property: sử dụng chủ yếu để quản lý và xác định các bean dependencies, và giá trị của chúng thông qua setter method
- Initialization: initialization mode, lazily or eagerly

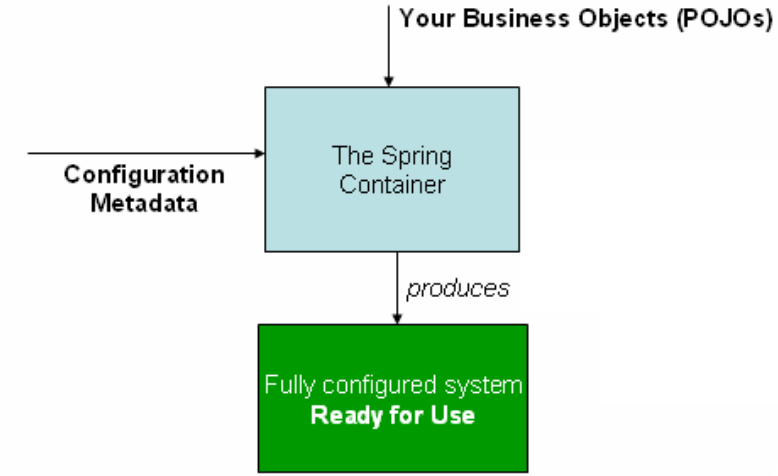
Định nghĩa nhiều bean của cùng 1 class

- Thêm thuộc tính name của @Bean để phân biệt

```
@Configuration
public class MyBeansConfig {
    @Bean(name="simpleBeanA")
    public SimpleBean simpleBeanA() {
        return new SimpleBean();
    }
    @Bean(name="simpleBeanB")
    public SimpleBean simpleBeanB(){
        return new SimpleBean();
    }
}
```

Spring IoC khởi tạo bean thế nào

- Khi chạy ứng dụng, spring trước tiên tạo IoC container
- Sau đó, container sẽ đọc cấu hình metadata được cung cấp bằng java hoặc xml, sau đó khởi tạo các beans được yêu cầu phụ thuộc vào các properties và các dependencies của chúng



Bean Dependencies

Bean Dependencies

- Dependencies là các object của các class mà là data member trong các class khác

```
public class Address {
    private String street;
    private String city;
    public Address() {
    }
    // Getters + Setters
}
public class Employee {
    private String firstname;
    private String lastname;
    // Dependency
    private Address address
    public Employee() {
    }
    // Getters + Setters
}
```

Bean Dependencies

- Định nghĩa một bean dependency
  - Không cần suy nghĩ đến việc làm thế nào để khởi tạo các objects hoặc quản lý tất cả các dependencies giữa chúng- Spring làm hết
  - Ở ví dụ dưới đây, Spring sử dụng constructor arguments như là bean dependencies

```
@Bean
public Employee empBean(Address address) {
    return new Employee(address);
}
```

```
@Component
public class Employee {
    private Address address;
    public Employee(Address address) {
        this .address = address;
    }
}
```

Bean Dependencies

- Wiring trong Spring
  - @Autowired
  - @Resource (thuộc JSR-250 annotation collection)
  - @Inject (thuộc JSR-330 annotations collection)
- @Autowired thường dùng
- Độ ưu tiên
  - Match by Type
  - Match by Qualifier
  - Match by Name

Bean Scope trong Spring

- Bean Scope
- Làm thế nào để định nghĩa Bean Scope
- Spring bean life cycle

Bean Scope

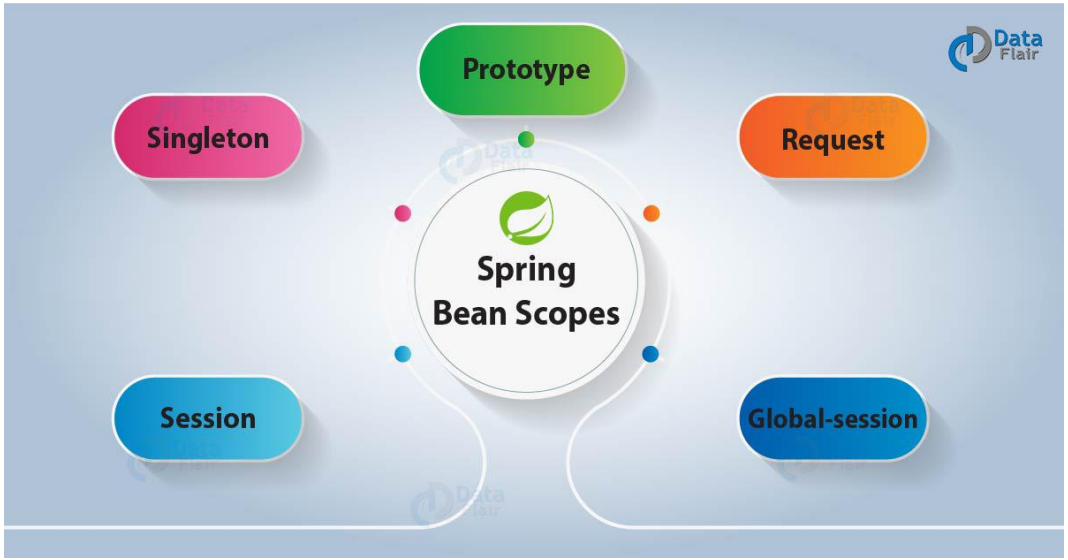
Scope	Description
singleton	Scope mặc định. Chỉ có 1 thể hiện (Instance) của bean được khởi tạo trong container
prototype	Mỗi lần bean được yêu cầu, khởi tạo một instance mới.
request	Một instance mới của bean được cấp cho mỗi Http request
session	Giống với request scope, một instance duy nhất của bean được tạo cho mỗi Http session.
application	Tạo một instance duy nhất của bean được chia sẻ thông qua cùng một web-aware Spring ApplicationContext
websocket	Chỉ một instance được tạo trong suốt một vòng đời (lifecycle) hoàn chỉnh của WebSocket.

Định nghĩa bean scope

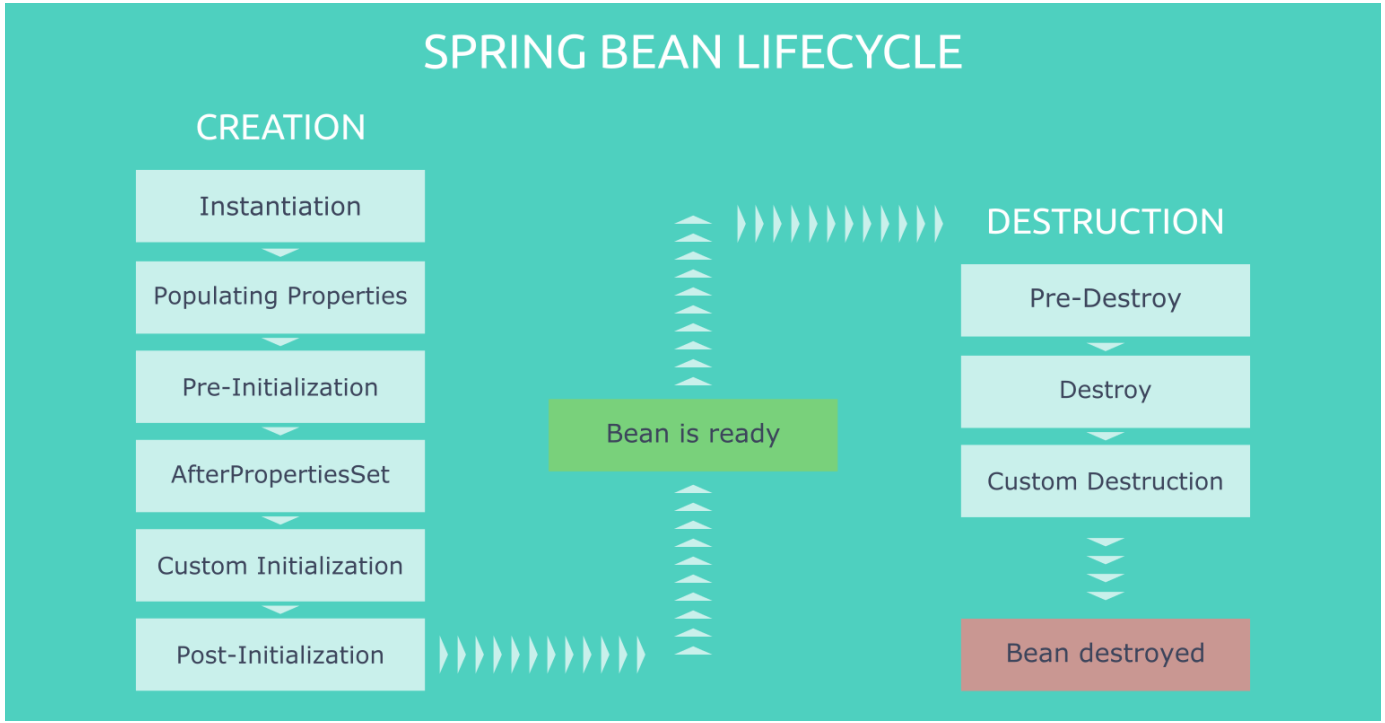
- @Scope

```
@Scope(value = ConfigurableBeanFactory.SCOPE_SINGLETON)
```

```
@Scope(value = ConfigurableBeanFactory.SCOPE_PROTOTYPE)
```



Vòng đời (lifecycle) của Spring bean



Vòng đời (lifecycle) của Spring bean

- Giai đoạn tạo bean (Bean Creation Phases)
  - Thực thể hóa (Instantiation) : Spring khởi tạo các bean object giống như cách chúng ta tạo thủ công một object instance java.
  - Điền các thuộc tính (Populating Properties): Sau khi khởi tại các đối tượng, Spring sẽ quét (scan) các bean mà implement Aware interfaces và bắt đầu thiết lập các thuộc tính có liên quan.
  - Tiền khởi tạo (Pre-initialization): BeanPostProcessors của Spring bắt đầu hoạt động trong giai đoạn này. Các method postProcessBeforeInitialization() thực hiện công việc của chúng. Ngoài ra các method được đánh dấu @PostConstruct chạy ngay sau đó
  - Hậu thiết lập thuộc tính (AfterPropertiesSet): Spring thực thi các method afterPropertiesSet () của các bean mà implement interface InitializingBean.
  - Khởi tạo tùy chỉnh (Custom Initialization): Spring kích hoạt các method khởi tạo mà chúng ta đã xác định trong thuộc tính initMethod của @Bean annotations.
  - Hậu khởi tạo (Post-Initialization): BeanPostProcessors của Spring hoạt động lần thứ hai. Giai đoạn này kích hoạt các method postProcessAfterInitialization ().

Vòng đời (lifecycle) của Spring bean

- Giai đoạn hủy bỏ bean (Bean Destruction Phases)
  - Tiền hủy bỏ (Pre-Destroy): Spring kích hoạt các method được đánh dấu @PreDestroy trong giai đoạn này.
  - Hủy bỏ (Destroy): Spring thực thi các method destroy() của các class mà implement interface DisposableBean.
  - Hủy tùy chỉnh (Custom Destruction): Chúng ta có thể xác định các móc hủy bỏ tùy chỉnh bằng thuộc tính destryMethod trong annotation @Bean và Spring chạy chúng trong giai đoạn cuối.

Get Bean với Applicationcontext : getBean() APIs

getBean() APIs

- Lấy Bean bằng Name

```
Object lion = context.getBean("lion");

assertEquals(Lion.class, lion.getClass());
```

- Lấy Bean bằng Type

```
Lion lion = context.getBean(Lion.class);
```

- Lấy Bean bằng Name và Type

```
Lion lion = context.getBean("lion", Lion.class);
```

- Và một số method khác
- **Không nên sử dụng applicationcontext getBean() một cách trực tiếp**

Sử dụng @SpringBootApplication

- Dùng để kích hoạt các tính năng sau
  - @EnableAutoConfiguration: Spring Boot’s auto-configuration
  - @ComponentScan: Sử dụng để scan các component (@Component và các annotation thừa kế nó) trong các package
  - @Configuration: Cho phép đăng kí các bean vào trong context hoặc import them các configuration class

Sử dụng Lombok cơ bản

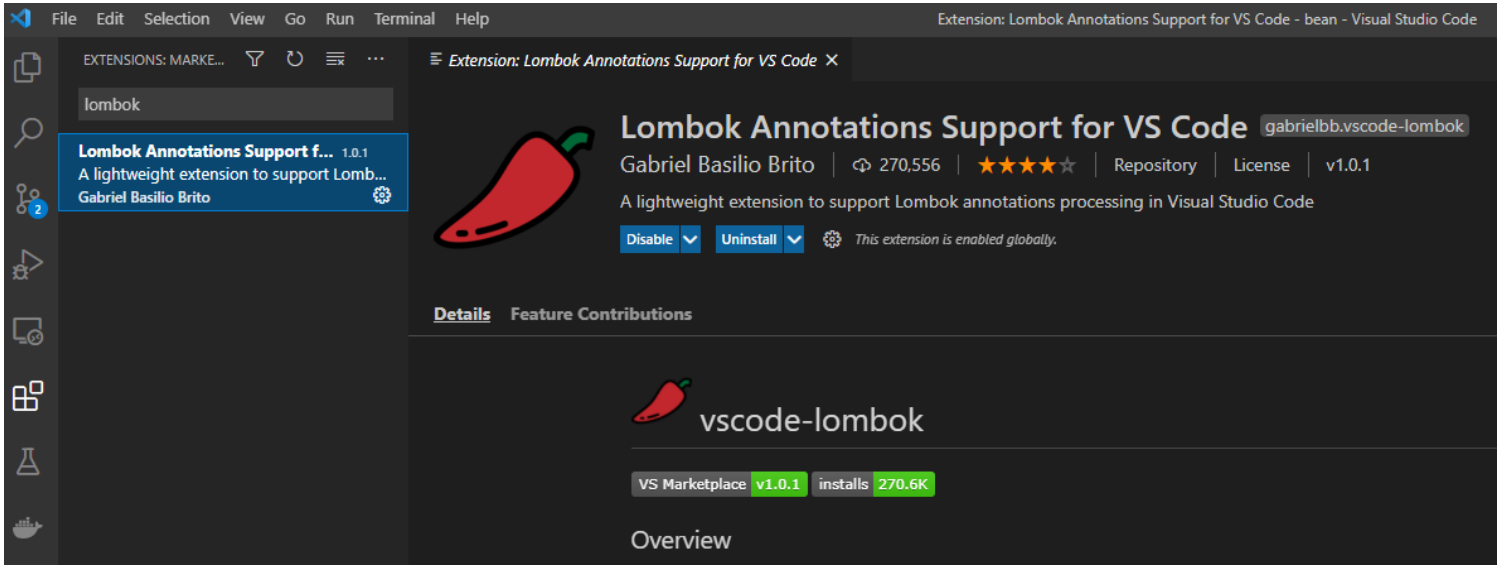
- Lombok là gì
- Cài đặt Lombok
- Sử dụng Lombok

Sử dụng Lombok cơ bản

- Lombok là gì
    - Lombok là một thư viện, một plugin, giúp chúng ta giảm thiểu các đoạn code thừa (boilerplate) bằng cách tự động sinh ra các hàm Get, Set, Constructor, các pattern Singleton, Builder .v.v..
- Cài đặt Lombok
        - Thêm dependency của Lombok vào project sử dụng Maven, Gradle hoặc ngay khi khai báo project trên https://start.spring.io/ (hay khởi tạo bằng IDE)
        - Cài đặt Lombok plugin cho IDE

Sử dụng Lombok cơ bản

```
<dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <scope>provided</scope>
</dependency>
```



Sử dụng Lombok cơ bản

- Sử dụng Lombok
  - Lombok sử dụng các annotation
  - Tham khảo web chính thức của project Lombok

    https://projectlombok.org/features/all
  - Một số Annotation hay dung
    - @Getter/@Setter, @ToString, @EqualsAndHashCode
    - @NoArgsConstructor, @RequiredArgsConstructor and @AllArgsConstructor
    - @Builder: Sử dụng builder pattern để tạo object instance
    - @Data : viết tắt cho @ToString, @EqualsAndHashCode, @Getter cho mọi fields, và @Setter cho các trường không phải final, và @RequiredArgsConstructor

Sử dụng Lombok cơ bản

```
@Getter
@Setter
public class GetterAndSetterDemo {
    String firstName;
}
```

```
public class GetterAndSetterDemo {
    String firstName;

    public GetterAndSetterDemo() {
    }

public String getFirstName() {
        return this.firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
}
```

Sử dụng Lombok cơ bản

```
@NoArgsConstructor(staticName = "getInstance")
public class NoArgsConstructorDemo {
    private long id;
    private String name;
    private int age;
}
```

```
public class NoArgsConstructorDemo {
    private long id;
    private String name;
    private int age;

    private NoArgsConstructorDemo() {
    }

    public static NoArgsConstructorDemo getInstance() {
        return new NoArgsConstructorDemo();
    }
}
```

Sử dụng Lombok cơ bản

```
@AllArgsConstructor
public class AllArgsConstructorDemo {
    private long id;
    private String name;
    private int age;
}
```

```
public class AllArgsConstructorDemo {
    private long id;
    private String name;
    private int age;

    public AllArgsConstructorDemo(long id, String name,
int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }
}
```

Sử dụng Lombok cơ bản

```
@RequiredArgsConstructor
public class RequiredArgsConstructorDemo {
    private final long id;
    private String name;
    private int age;
}
```

```
public class RequiredArgsConstructorDemo {
    private final long id;
    private String name;
    private int age;

    public RequiredArgsConstructorDemo(long id) {
        this.id = id;
    }
}
```

Sử dụng Lombok cơ bản

```
@EqualsAndHashCode
public class EqualsAndHashCodeDemo {
    String firstName;
    String lastName;
    LocalDate dateOfBirth;
}
```

```
public class EqualsAndHashCodeDemo {
    String firstName;
    String lastName;
    LocalDate dateOfBirth;

    public EqualsAndHashCodeDemo() {
    }

    public boolean equals(Object o) {
        if (o == this) {
            return true;
        } else if (!(o instanceof EqualsAndHashCodeDemo)) {
            return false;
        } else {
            EqualsAndHashCodeDemo other = (EqualsAndHashCodeDemo)o;
            if (!other.canEqual(this)) {
                return false;
            } else {
              // A lot of code has been omitted
        }
    }

    public int hashCode() {
        int PRIME = true;
        int result = 1;
        Object $firstName = this.firstName;
        int result = result * 59 + ($firstName == null ? 43 : $firstName.hashCode());
        Object $lastName = this.lastName;
        result = result * 59 + ($lastName == null ? 43 : $lastName.hashCode());
        Object $dateOfBirth = this.dateOfBirth;
        result = result * 59 + ($dateOfBirth == null ? 43 : $dateOfBirth.hashCode());
        return result;
    }
}
```

Sử dụng Lombok cơ bản

```
@ToString(exclude = {"dateOfBirth"})
public class ToStringDemo {
    String firstName;
    String lastName;
    LocalDate dateOfBirth;
}
```

```
public class ToStringDemo {
    String firstName;
    String lastName;
    LocalDate dateOfBirth;

    public ToStringDemo() {
    }

    public String toString() {
        return "ToStringDemo(firstName=" +
this.firstName + ", lastName=" +
          this.lastName + ")";
    }
}
```

DEMO