


**Bean**

## Bean이란?

A bean is an object that is instantiated, assembled, and managed by a **Spring IoC container**.



**Spring IoC Container**가 관리하는 순수 자바 객체(POJO)

역할 - 1. 객체 생명 주기 관리  
2. 의존성 관리

### + POJO (Plain Old Java Object)

: Java로 생성하는 순수한 객체

: 특정 '기술'에 종속되어 동작하는 것이 아닌 Getter와 Setter로 구성된 가장 순수한 형태의 기본 클래스

### + SingletonBeanRegistry

- Registry: key-value 형태로 데이터를 저장하는 방법 (Spring 말고도 CS에서 사용됨)
- 인스턴스화된 스프링 빈이 저장되는 곳
- 인터페이스로, 구현은 DefaultSingletonBeanRegistry에 되어 있음

## IoC 컨테이너란?

- 스프링 빈의 생명 주기를 관리, 생성된 빈들에게 추가적인 기능을 제공하는 역할을 함

### # BeanFactory

- 스프링 컨테이너의 최상위 인터페이스로, 스프링 빈을 관리하고 조회하는 역할을 담당
- Bean을 생성하고 관계를 설정하는 IoC 기본 기능에 초점을 맞춘 것 (getBean() 메소드 등)
- 기본적으로 Lazy Loading(지연 로딩) 방식으로 작동  
: 빈 객체의 로딩 요청이 올 때 빈이 생성/로딩됨

### # ApplicationContext

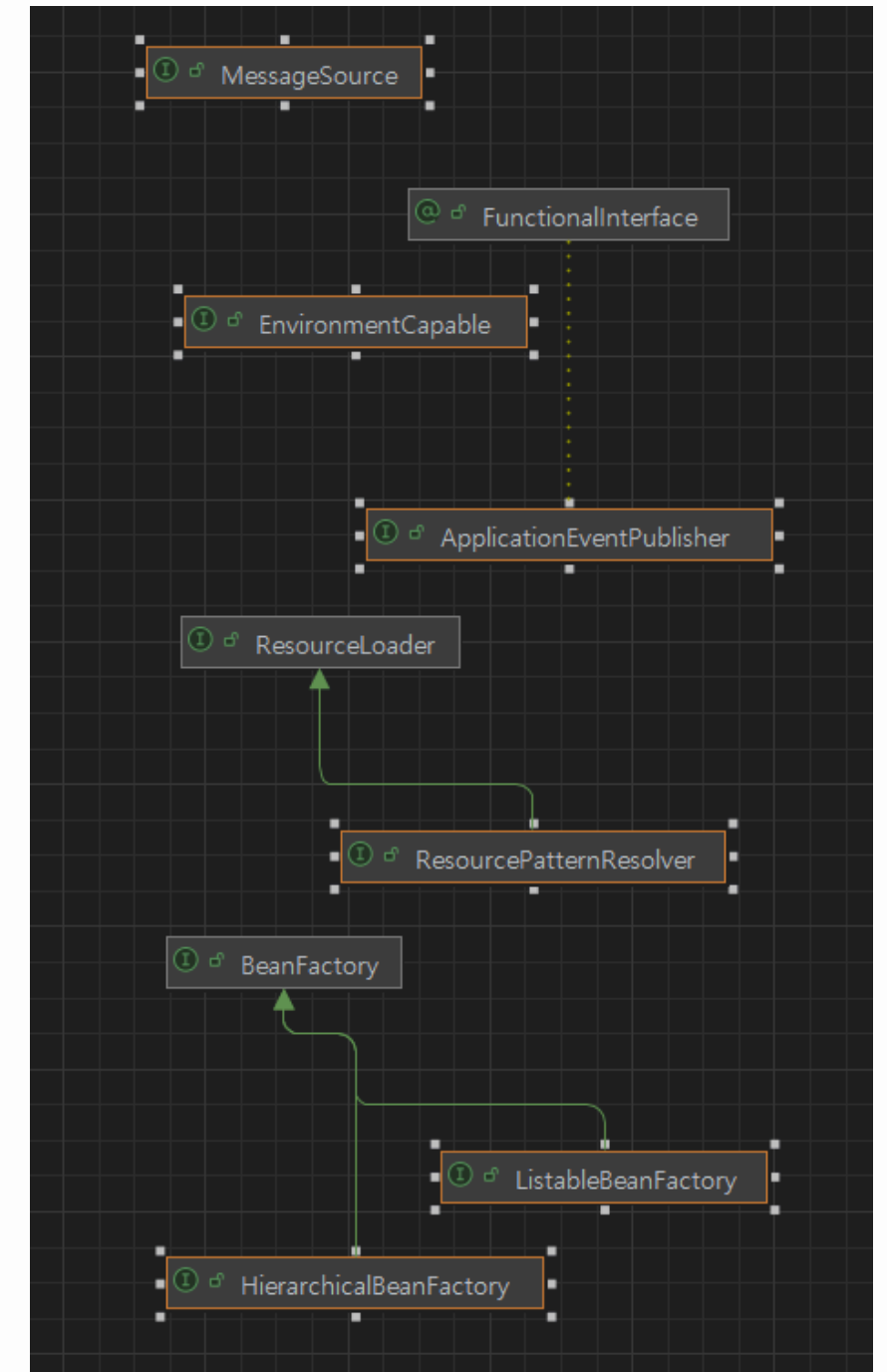
- Bean Factory를 확장한 하위 인터페이스 (스프링이 제공하는 부가 서비스를 추가로 제공)
- 기본적으로 Pre-loading(즉시 로딩) 방식으로 작동  
: 애플리케이션 실행될 때 빈을 미리 생성/로딩 → 시작시간 단축 가능

# IoC 컨테이너란? - 참고

ApplicationContext.class

```
public interface ApplicationContext extends EnvironmentCapable, ListableBeanFactory, HierarchicalBeanFactory, MessageSource, ApplicationEventPublisher, ResourcePatternResolver {  
    @Nullable @Override  
    <T> T getBean(String name, Class<T> requiredType, Object... args) throws BeansException;  
}
```

- **ListableBeanFactory**  
: (BeanFactory) 빈 목록 조회하는 기능
- **HierarchicalBeanFactory**  
: (BeanFactory) 부모 BeanFactory에 등록된 빈 객체들을 상속받아 사용 가능
- **EnvironmentCapable** :  
애플리케이션 환경을 알아내는 기능 제공, 실행 환경에 따라 다양한 설정/환경 변수 사용 가능
- **MessageSource**  
: 국제화 기능 지원하는 메시지 관리 기능 (한국-한국어, 미국-영어 메시지 제공 등)
- **ApplicationEventPublisher**  
: 이벤트 발생시키거나 해당 이벤트 처리하는 리스너 등록 가능
- **ResourcePatternResolver**  
: 다양한 리소스 검색/이용해 필요한 작업 수행 가능



# Bean 생명 주기

Spring Container 생성

Bean 생성

의존성 주입

## Configuration Metadata

XML 설정 파일  
or  
Java Configuratoin  
or  
Annotation



POJO

Bean  
Definition



## Spring IoC Container

빈 생성 (+저장)

초기화 콜백

Bean 사용

소멸 전 콜백

스프링 종료

## Callback (콜백)

- 빈의 특정 상태 변화(예: 초기화 완료 후, 소멸 전)에 호출되는 메서드
- 즉, 스프링 빈이 생성/소멸될 때 초기화 및 종료 작업을 수행할 수 있도록 하는 것
- Spring에서 생명주기 콜백 지원하는 방법
  1. 인터페이스 방식: 사실상 거의 사용X
  2. @Bean 파라미터 방식
    - : 외부 라이브러리를 빈으로 등록하면서 초기화 및 종료 메서드가 필요할 경우 사용
  3. 어노테이션 방식
    - : @PostConstruct, @PreDestroy를 사용 (가장 권장)

콜백 참고: <https://twoline.tistory.com/162>

# Bean Scope

→ 빈이 존재할 수 있는 범위

## 1. Singleton

- 객체의 인스턴스가 오직 1개만 생성되는 것을 보장
- 스프링 컨테이너의 시작-종료까지 유지됨 (가장 넓은 스코프)
- 주의점: 객체가 stateless하도록 설계해야 함

```
@Scope("prototype") | 1 usage new *  
@Component  
public class TestBean {
```

## 2. Prototype

- IoC 컨테이너와 함께 생성/소멸되는 것이 아닌, 요청 올때마다 객체가 생성됨
- 즉, 모든 스레드에서 공유되는 것이 아니라 상태를 가질 수 있음
- 스프링 컨테이너가 프로토타입 빈의 생성/의존 관계 주입까지만 관여, 이후는 관리 X
- 주의점: 자동주입 시 구현체가 여러개면, 어떤 것을 주입해야할 지 몰라서 충돌 발생  
→ Annotation으로 우선순위 지정 가능
  - 여러개 중 하나만 쓰는 경우: @Primary 붙이기
  - 상황에 따라 다른 구현체 씬 : @Qualifier("~") 안에 빈 이름 넣어서 특정 구현체 주입

+) 웹 스코프 (웹 환경에서만 동작하는 Scope, 비교적 덜 중요)

- 3. **request**: HTTP 요청이 들어오고 나갈 때까지 유지
- 4. **session**: HTTP 세션이 생성되고 종료될 때까지 유지
- 5. **application**: 서블릿 컨텍스트와 동일한 생명 주기를 가짐
- 6. **websocket**: 웹 소켓과 동일한 생명 주기를 가짐

## Bean 등록 방법 : 1. XML 설정 파일

### TestBean

```
package com.spring;

public class TestBean { 3 usages new *
    private String name = "TEST_NAME"; 2 usages
    private int price; 2 usages

    public String getName() { new *
        return name;
    }

    public int getPrice() { no usages new *
        return price;
    }

    public void setName(String name) { new *
        this.name = name;
    }

    public void setPrice(int price) { new *
        this.price = price;
    }
}
```

application.xml (resources 폴더 아래에 생성)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="testBean" class="com.spring.TestBean"/>
    <!-- <bean id="testBean2" class="com.spring.TestBean2"/>-->
</beans>
```

### TestCode

```
class ApplicationTests {
    @Test new *
    @DisplayName("XML로 빈 등록")
    void xmlBeanTest() {
        ApplicationContext context = new ClassPathXmlApplicationContext(configLocation: "application.xml");
        TestBean tb = (TestBean) context.getBean(name: "testBean");
        System.out.println(tb.getName());
    }
}
```

ApplicationTests (c 603 ms	✓ Tests passed: 1 of 1 test – 603 ms
XML로 빈 등록 603 ms	"C:\Program Files\Java\jdk-17\bin\java.exe"
	TEST_NAME
	Process finished with exit code 0



## Bean 등록 방법 : 2. Java Configuration

### BeanTestConfig

```
6  @Configuration new *
7  public class BeanTestConfig {
8
9      @Bean new *
10     public TestBean testBean() {
11         return new TestBean();
12     }
13 }
```

### TestCode

```
@ExtendWith(SpringExtension.class)
@ContextConfiguration(classes = BeanTestConfig.class)
class ApplicationTests {

    @Autowired
    private TestBean tb;

    @Test new *
    @DisplayName("Config로 빈 등록")
    void configBeanTest() {
        System.out.println(tb.getName());
    }
}
```

```
✓ Applicat 1 sec 698 ms
✓ Conf 1 sec 698 ms
✓ Tests passed: 1 of 1 test – 1 sec 698 ms
"C:\Program Files\Java\jdk-17\bin\java.exe"
Java HotSpot(TM) 64-Bit Server VM warning:
TEST_NAME

Process finished with exit code 0
```

## Bean 등록 방법 : 3. Annotation

### TestBean

```
@Component 3 usages new *
public class TestBean {
    private String name = "TEST_NAME"; 2 usages
    private int price; 2 usages
    public String getName() { new *
```

### 빈 이름

- Annotation에서는 클래스명 맨앞을 소문자로 만든 것
- @Component("tb") 처럼 직접 빈이름 지정 가능

### @Component

- 해당 어노테이션이 붙은 클래스들을 자동으로 빈 등록 (컴포넌트 스캔)
- @Controller, @Service, @Repository, @Configuration 등에 기본으로 들어있음!!

### TestCode

```
@SpringBootTest * dudxo *
class ApplicationTests {
    @Autowired
    private TestBean tb;

    @Test new *
    @DisplayName("Annotation으로 빈 등록")
    void annoBeanTest() {
        System.out.println(tb.getName());
    }
}
```

```
ApplicationTests: 631 ms
Annotation: 631 ms
Tests passed: 1 of 1 test - 631 ms
Java HotSpot(TM) 64-Bit Server VM
TEST_NAME
```

## ComponentScan

- 기본적으로 @Component 어노테이션을 스캔하여 실제 빈으로 등록해줌
- @ComponentScan의 설정들을 기준으로 스캔
- 기본적으로 @SpringBootApplication 안에 들어있음

### # basePackages

- @ComponentScan(basePackages = "com.spring")
- 해당 패키지 내부에 대해 스캔

### # basePackageClasses

- @ComponentScan(basePackageClasses = Application.class)
- 괄호 안의 Class가 위치한 곳에서부터 하위 패키지까지 모두 스캔

## @SpringBootApplication

```
public @interface SpringBootApplication {  
    @AliasFor(  
        annotation = EnableAutoConfiguration.class  
    )  
    Class<?>[] exclude() default {};  
  
    @AliasFor(  
        annotation = EnableAutoConfiguration.class  
    )  
    String[] excludeName() default {};  
  
    @AliasFor(  
        annotation = ComponentScan.class,  
        attribute = "basePackages"  
    )  
    String[] scanBasePackages() default {};  
  
    @AliasFor(  
        annotation = ComponentScan.class,  
        attribute = "basePackageClasses"  
    )  
    Class<?>[] scanBasePackageClasses() default {};  
}
```

## 비교

### @Component

- Component Scan 기능으로 자동 등록
- 개발자가 직접 제어 가능한 클래스를 빈으로 등록할 때
- 클래스 레벨에서 선언 가능

### @Configuration + @Bean

- 수동 등록 (하나하나 적어줘야함)
- 외부라이브러리/내장 클래스 등 개발자가 직접 제어할 수 없는 클래스를 빈으로 등록하고자 할 때
- 특정 Bean 들을 한곳에서 관리하고 싶을 때
- 클래스 레벨에서의 선언 불가능
- @Bean이 선언된 메서드 내부에서 생성한 객체를 빈으로 등록
- 1개 이상의 @Bean을 제공하는 클래스의 경우 반드시 @Configuration을 명시해 주어야 싱글톤이 보장됨

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Component
public @interface Configuration {
    @AliasFor(
        annotation = Component.class
    )
    String value() default "";

    boolean proxyBeanMethods() default true;

    boolean enforceUniqueMethods() default true;
}
```

## @Component+@Bean를 쓴다면?

싱글톤 보장이 되지 않을 수도 있음!

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Component
public @interface Configuration {
    @AliasFor(
        annotation = Component.class
    )
    String value() default "";

    boolean proxyBeanMethods() default true;

    boolean enforceUniqueMethods() default true;
}
```

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Indexed
public @interface Component {
    String value() default "";
}
```

```
@Component new *
public class BeanTestConfig {
    @Bean new *
    public A a() {
        return new A(b());
    }

    @Bean new *
    public B b() {
        return new B();
    }
}
```

BeanTestConfig.java C:\Users\Wamsung\uplus-ureka\spring-study\src\main\java\com\...  
! Method annotated with @Bean is called directly. Use dependency injection instead. :10

- 빈 여러개 생성될 수도 있음 (메모리 효율↓)
- 의존성 주입으로 쓰라고 오류뜸

```
@Configuration new *
public class BeanTestConfig {
    @Bean new *
    public A a() {
        return new A(b());
    }

    @Bean new *
    public B b() {
        return new B();
    }
}
```

- CGLIB로 프록시 패턴 적용됨
- proxyBeanMethods가 true일 때
  - @Bean 메소드 내부에서 @bean메소드 호출  
→ 객체를 새롭게 생성하지 X  
→ 이미 등록된 빈을 찾아서 반환해줌
  - 싱글톤 보장!!!

+ ) @Configuration(proxyBeanMethods = false)

- 빈을 매번 생성하도록 만들 수 있음
- 단, 공식 문서에서는 @Configuration의 장점을 살리지 못하는 행위라며 권장하지 않음

## <참고>

- <https://docs.spring.io/spring-framework/reference/core/beans/definition.html>
- [https://youtu.be/3gURJvJw\\_T4?si=zwQhyDoIRJQPDhIP](https://youtu.be/3gURJvJw_T4?si=zwQhyDoIRJQPDhIP)
- <https://youtu.be/UcpLNgko8lg?si=YaMmfdB52oRxDguu>
- <https://mangkyu.tistory.com/75>
- <https://developer111.tistory.com/entry/스프링-빈-생성-원리-2-beanFactoryPostProcessor-beanPostProcessor>
- <https://ngp9440.tistory.com/48>
- <https://kth990303.tistory.com/395>
- <https://twoline.tistory.com/162>