

一、Bean生命周期的五个阶段

Java 中的公共类称之为 Bean 或 Java Bean，而 Spring 中的 Bean 指的是将对象的生命周期，交给 Spring IoC 容器来管理的对象。所以 Spring 中的 Bean 对象在使用时，无需通过 new 来创建对象，只需要通过 DI（依赖注入），从 Spring 中取出要使用的对象即可。

1.Bean的各个生命周期阶段

Bean作为一个Java对象，具有一定的生命周期。它的生命周期包括以下几个阶段：

- 实例化：在Java应用程序中，Bean对象是通过new关键字或者反射机制来实例化的。在这个阶段，Bean对象被创建，并分配了内存空间。
- 设置属性(Bean注入和装配)
- 初始化：当Bean对象被创建后，需要进行初始化，包括设置属性值、执行一些初始化操作等。在Spring框架中，Bean的初始化可以通过配置文件中的init-method属性进行指定。
- 使用：在Bean初始化之后，它就可以被应用程序使用了。在使用过程中，Bean可能会调用其他对象的方法，从而导致其他Bean对象被实例化和初始化。
- 销毁：当Bean对象不再被使用时，应该将其销毁并释放占用的内存空间。在Spring框架中，Bean的销毁可以通过配置文件中的destroy-method属性进行指定。

总的来说，Bean对象的生命周期可以通过实例化、初始化、使用和销毁这几个阶段来描述。在Spring框架中，Bean的生命周期还可以通过BeanPostProcessor接口来进行扩展和定制。

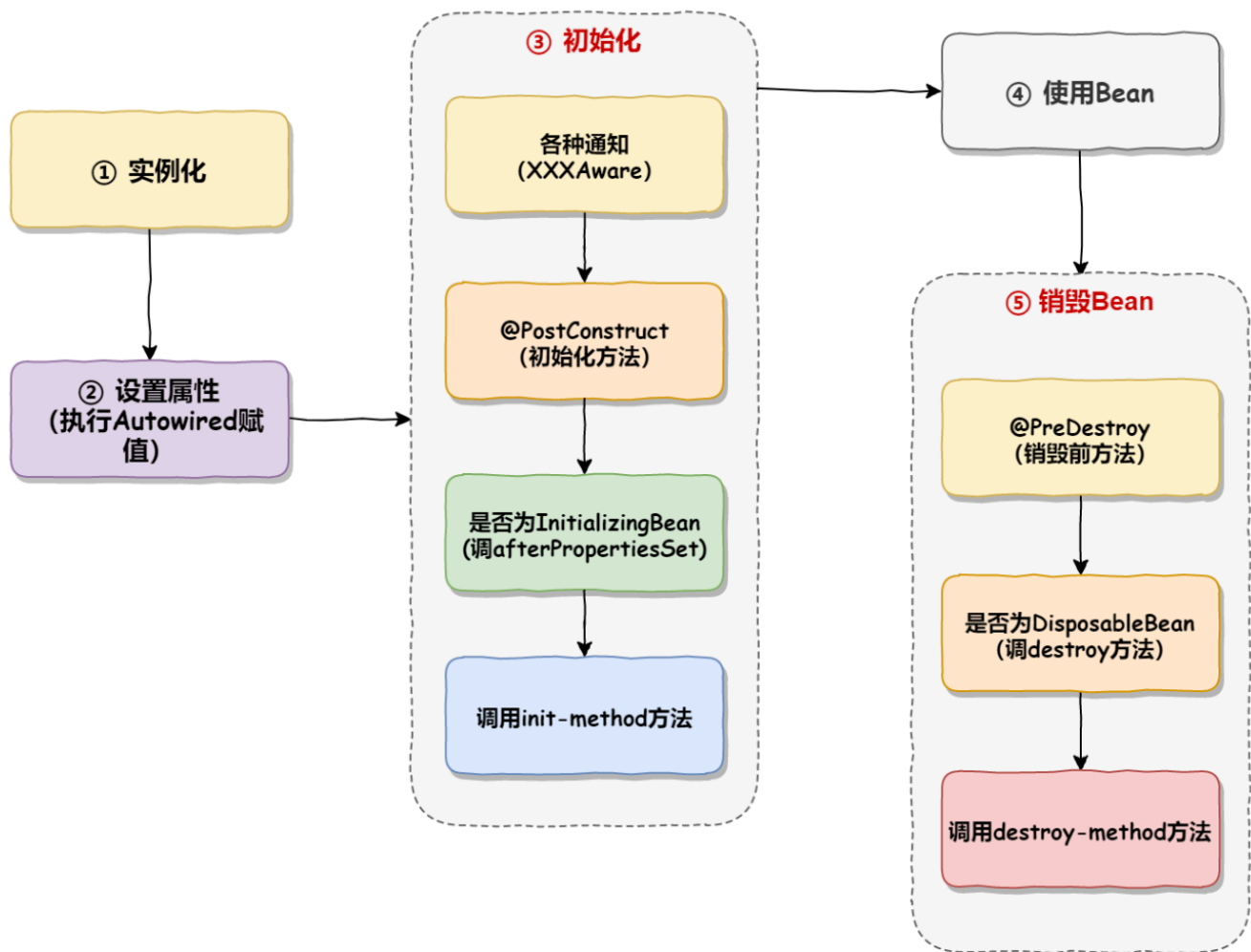
为了便于理解，我们引用一个现实中的事件来形容：

Bean 的生命流程看似繁琐，但咱们可以以生活中的场景来理解它，比如我们现在需要买一栋房子，那么我们的流程是这样的：

- 先买房（实例化，从无到有）；
- 装修（设置属性）；
- 买家电，如洗衣机、冰箱、电视、空调等（[各种]初始化）；
- 入住（使用 Bean）；
- 卖出去（Bean 销毁）。

2.Bean的初始化

- 执行各种通知(BeanNameAware、BeanFactoryAware)等接口方法
- 初始化的前置方法(PostConstruct)
- 初始化方法
- 初始化的后置方法(PreDestroy)



下面用代码演示整个Bean的生命周期:

我们用构造方式来展示实例化的效果,实例化和属性设置是Java级别的系统“事件”,其操作过程不可人工干预和修改,所以下面没有演示设置属性:

```
package com.example.bean.test;

import org.springframework.beans.factory.BeanNameAware;
import org.springframework.stereotype.Component;

import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;

@Component
public class BeanLifeComponent implements BeanNameAware {

    private String name;

    public BeanLifeComponent() {
        System.out.println("实例化方法执行拉");
    }
}
```

```

    public void setBeanName(String s) {
        System.out.println("执行 BeanName 的通知方法");
    }

    @PostConstruct
    public void postConstruct() {
        System.out.println("初始化方法执行");
    }

    public void use() {
        System.out.println("使用 Bean");
    }

    @PreDestroy
    public void preDestroy() {
        System.out.println("销毁方法执行");
    }
}

```

因为初始化的前置方法和后置方法是为所有 Bean 服务的，而非为某一个 Bean 服务的，所以这两个方法不能写在某个具体的 Bean 中，否则（这两个方法）不会执行。所以我们另外创建一个类来实现这两个方法：

```

package com.example.bean.test;

import org.springframework.beans.BeansException;
import org.springframework.beans.factory.config.BeanPostProcessor;
import org.springframework.stereotype.Component;

@Component
public class MyBeanPostProcessor implements BeanPostProcessor {
    @Override
    public Object postProcessBeforeInitialization(Object bean, String beanName) throws
BeansException {
        if (beanName.equals("beanLifeComponent")) {
            System.out.println("初始化前置方法执行");
        }
        return bean;
    }

    @Override
    public Object postProcessAfterInitialization(Object bean, String beanName) throws
BeansException {
        if (beanName.equals("beanLifeComponent")) {
            System.out.println("初始化后置方法执行");
        }
        return bean;
    }
}

```

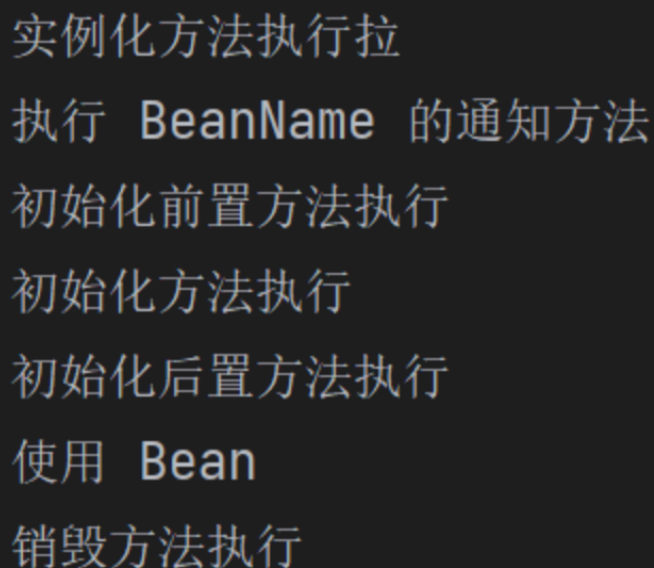
在得到上下文对象这里，因为我们是使用springboot演示的 所以我们使用ConfigurableApplicationContext 来得到上下文对象。

```
package com.example.bean.test;

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

@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        // 得到上下文对象, 并启动 Spring Boot 项目
        ConfigurableApplicationContext context =
            SpringApplication.run(DemoApplication.class, args);
        // 获取 Bean
        BeanLifeComponent component = context.getBean(BeanLifeComponent.class);
        // 使用 Bean
        component.use();
        // 停止 Spring Boot 项目
        context.close();
    }
}
```

结果如下:



实例化方法执行
执行 BeanName 的通知方法
初始化前置方法执行
初始化方法执行
初始化后置方法执行
使用 Bean
销毁方法执行

二、@PostConstruct 和 @PreDestroy 各自的效果

- 当Bean被容器初始化之后，会调用@PostConstruct的注解方法。
- 当Bean在容器销毁之前，调用被@PreDestroy注解的方法

所以，PostConstruct 注释用于在依赖关系注入完成之后需要执行的方法上，以执行任何初始化。此方法必须在将类放入服务之前调用。支持依赖关系注入的所有类都必须支持此注释。即使类没有请求注入任何资源，用PostConstruct 注释的方法也必须被调用。只有一个方法可以用此注释进行注释。

```

@PostConstruct
public void doPostConstruct(){
    System.out.println("执行了 新版本@PostConstruct 前置初始化方法");
}
@PreDestroy
public void doPreDestroy(){
    System.out.println("执行新版本 @PreDestroy 销毁方法");
}

```

PreDestroy 用与在依赖注入完成之前的方法前面执行，遵守准则：

- 该方法不得有任何参数
- 该方法的返回类型必须为 void；
- 该方法不得抛出已检查异常；
- 应用 PostConstruct 的方法可以是 public、protected、package private 或 private；
- 该方法不能是 static；该方法可以是 final；
- 该方法只会被执行一次

三、实例化和初始化的区别

实例化和属性设置是 Java 级别的系统“事件”，其操作过程不可人工干预和修改；而初始化是给开发者提供的，可以在实例化之后，类加载完成之前进行自定义“事件”处理。

四、为什么要先设置属性在进行初始化呢？

仔细观察下面代码：

```

@Service
public class UserService {
    public UserService(){
        System.out.println("调用 User Service 构造方法");
    }
    public void sayHi(){
        System.out.println("User Service SayHi.");
    }
}
@Controller
public class UserController {
    @Resource
    private UserService userService;
    @PostConstruct
    public void postConstruct() {
        userService.sayHi();
        System.out.println("执行 User Controller 构造方法");
    }
}

```

我们可以发现很是为了避免空指针异常。