极客学院
jikexueyuan.com

Spring IoC容器深入理解

Spring IoC容器深入理解 — **课程概要**

- IoC概述
- Java反射机制
- 资源访问工具类
- BeanFactory和ApplicationContext的介绍
- Bean的生命周期

IoC概述

IoC概述

IoC是Spring容器的内核，AOP、声明式事务等功能都依赖于此功能，它涉及代码解耦，设计模式、代码优化等问题的考量，我们将通过以下三方面来深入了解IoC：

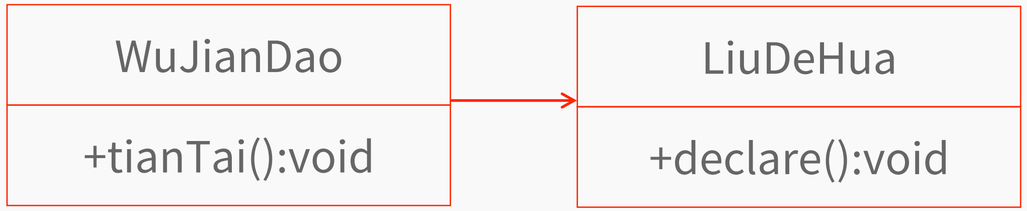
- IoC的初步理解
- IoC的注入类型
- IoC的注入方式

IoC概述 – IoC的初步理解

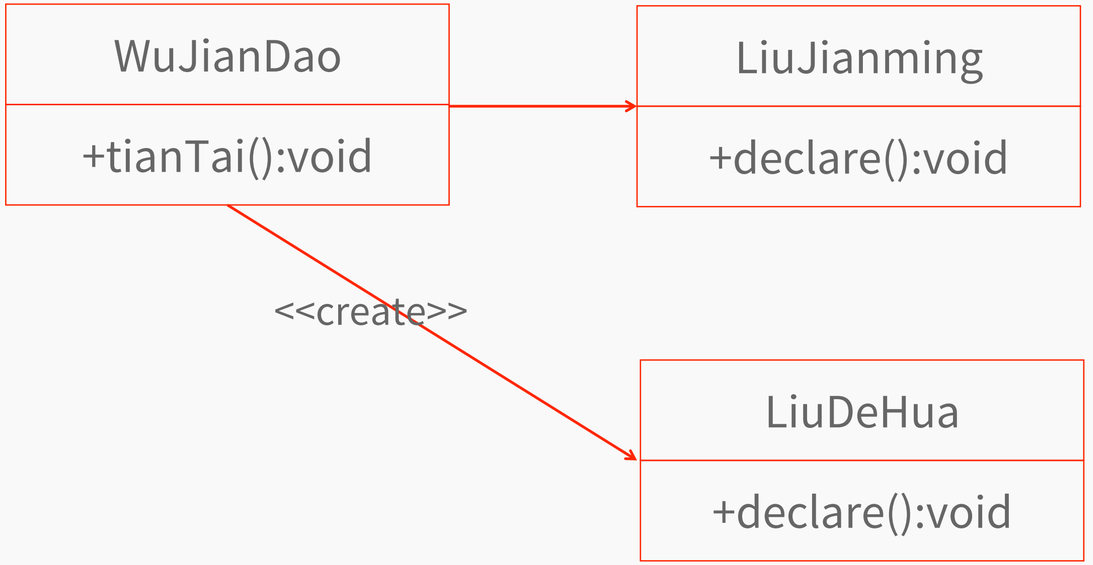
IoC的概念重要但比较晦涩难懂，如下将通过一个小例子来说明这个概念：

- 示例场景：**电影：无间道 -》角色：刘建明 -》演员：刘德华**

剧本和演员直接耦合

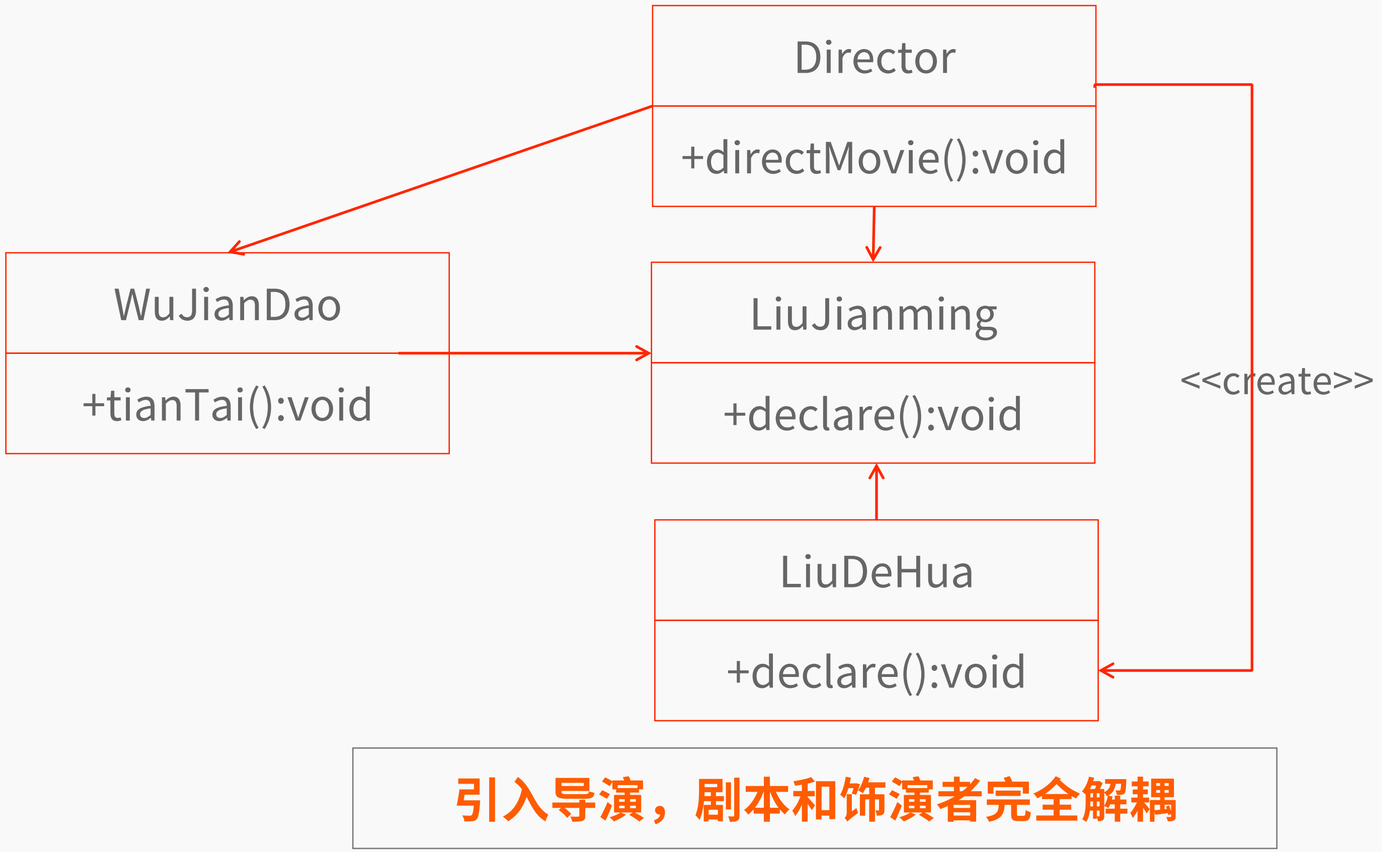


引入角色接口后的关系



IoC概述 – IoC的初步理解

- 示例场景：**电影：无间道 -》角色：刘建明 -》演员：刘德华**



IoC的字面理解：

- 其一：控制
- 其二：反转

DI概念的引入：

- 让调用类对某一接口实现类的依赖关系由第三方注入，以移除调用类对某一接口实现类的依赖

IoC概述 – IoC的注入类型

从注入方法上看，主要划分为三种类型：

- 构造函数注入
- 属性注入
- 接口注入

IoC概述 – IoC的注入类型

构造函数注入：通过调用类的构造函数，将接口实现类通过构造函数变量传入：

```
Public class WuJianDao {
  private LiuJianming ljm;
  //1：注入刘建明的具体扮演者
  public WuJianDao(LiuJianming ljm) {
     this.ljm = ljm;
  }
    public void tianTai() {
          ljm.declare(“我想做一个好
人！”)
    }
}
```

```
Public class Director {
  public void direct() {
    //2.指定角色的扮演者
    LiuJianming ljm = new LiuDeHua();
    //3.注入具体扮演者到剧本中
    WuJianDao wjd = new WuJianDao(ljm);
    wjd.tianTai();
  }
}
```

IoC概述 – IoC的注入类型

属性注入：通过Setter方法完成调用类所需依赖的注入，更加灵活方便：

```
Public class WuJianDao {
  private LiuJianming ljm;
  //1.属性注入方法
  public void setLjm(LiuJianmin ljm) {
    this.ljm = ljm;
  }
  public void tianTai() {
    ljm.declare(“我想做一个好人！”)
   }
}
```

```
Public class Director {
  public void direct() {
    LiuJianming ljm = new LiuDeHua();
    WuJianDao wjd = new WuJianDao();
    //2.调用属性Setter方法注入
    wjd.setLjm(ljm);
    wjd.tianTai();
  }
```

IoC概述 – IoC的注入类型

接口注入：将调用类所有依赖注入的方法抽取到一个接口中，调用类通过实现该接口提供相应的注入方法。

```
Public interface ActorArrangable {
  void injectLjm(LiuJianming ljm);
}
```

```
Public class WuJianDao
implements ActorArrangable {
  private LiuJianming ljm;
  //1.实现接口方法
  public void injectLjm(LiuJianming ljm)
    {this.ljm = ljm; }
  public void tianTai() {
     ljm.declare(“我想做一个好人！”)
    }
}
```

```
Public class Director {
  public void direct() {
     LiuJianming ljm = new LiuDeHua();
     WuJianDao wjd= new WuJianDao();
     wjd.injectLjm(ljm);
     wjd.tianTai();
   }
}
```

IoC概述 – IoC的注入方式

Spring作为一个容器，通过配置文件或者注解描述类和类之间的依赖关系，自动完成类的初始化和依赖注入的工作，下面是对以上实例进行配置的配置文件片段：

```
    //1.实现类实例化
    <bean id=“ljm" class=“LiuDeHua" />
    //2.通过ljm-ref建立依赖关系
    <bean id=“wjd"  class=“WuJianDao"  p:ljm-ref=“ljm”  />
</beans>
```

Java反射机制

Java反射机制

Java语言允许通过程序化的方式间接对Class的对象实例操作，Class文件由类装载器装载后，在JVM中将形成一份描述Class结构的元信息对象，通过该元信息对象可以获知Class的结构信息，如构造函数、属性和方法等：

- 示例讲解：通过实例探访Java反射机制
- ClassLoader：介绍ClassLoader的工作机制以及重要方法
- Java反射机制：深入讲解Java的反射机制
- 与IoC的关系：通过实例介绍Java反射机制与Spring IoC之间的关系

Java反射机制 – 示例讲解

编写一个简单示例开始探访Java反射机制的征程，通过比较传统方法以及反射机制创建类实例的不同，来介绍Java反射机制的原理：

- Car类：拥有两个构造函数，一个方法以及三个属性
- 传统调用方法，使用构造函数设置属性或者set方法设置属性
  1. **构造函数方法：Car car = new Car(“红旗轿车”，“黑色”，“180”);**
  2. **Set方法：Car car = new Car(); car.setBrand(“红旗轿车”);**
- Java反射机制，以一种更加通用的方式间接地操作目标类

Java反射机制 – ClassLoader

类装载器就是寻找类的字节码文件并构造出类在JVM内部表示的对象组件，主要工作由ClassLoader及其子类负责，ClassLoader是一个重要的Java运行时系统组件，它负责在运行时查找和装入Class字节码文件：

- 工作机制：
  1. 装载：查找和导入Class文件
  2. 链接：执行校验，准备和解析步骤
  3. 初始化：对类的静态变量、静态代码块执行初始化工作

- 重要方法：
  1. Class loadClass（String name）
  2. Class defineClass(String name, byte[]b, int off,int len)
  3. Class findSystemClass(String name)
  4. Class findLoadedClass(String name)
  5. ClassLoader getParent()

Java反射机制 – Java反射机制

Class反射对象描述类语义结构，可以从Class对象中获取构造函数，成员变量，方法等类元素的反射对象，并以编程的方式通过这些反射对象对目标类对象进行操作。这些反射对象类在java.reflect包中定义，下面是最主要的三个反射类：

- Constructor
- Method
  1. Class getReturnType()
  2. Class[] getParameterTypes()
  3. Class[] getExceptionTypes()
  4. Annotation[][] getParameterAnnotations()
- Field

Java反射机制 – 与IoC的关系

在Spring中，通过IOC可以将实现类、参数信息等配置在其对应的配置文件中，那么当需要更改实现类或参数信息时，只需要修改配置文件即可，我们还可以对某对象所需要的其它对象进行注入，这种注入都是在配置文件中做的。

Spring的IOC的实现原理利用的就是Java的反射机制，Spring的工厂类会帮我们完成配置文件的读取、利用反射机制注入对象等工作，我们可以通过bean的名称获取对应的对象。

资源访问工具类

资源访问工具类

JDK所提供的访问资源的类并不能很好的满足各种底层资源的访问需求，因此，Spring设计了一个Resource接口，它为应用提供了更强大的访问底层资源的能力：

- 主要方法：
  1. boolean exists()
  2. boolean isOpen()
  3. URL getURL()
  4. File getFile()
  5. InputStream getInputStream()

- 具体实现类：
  1. ByteArrayResource
  2. ClassPathResource
  3. FileSystemResource
  4. InputStreamResource
  5. ServletContextResource
  6. UrlResource

资源访问工具类

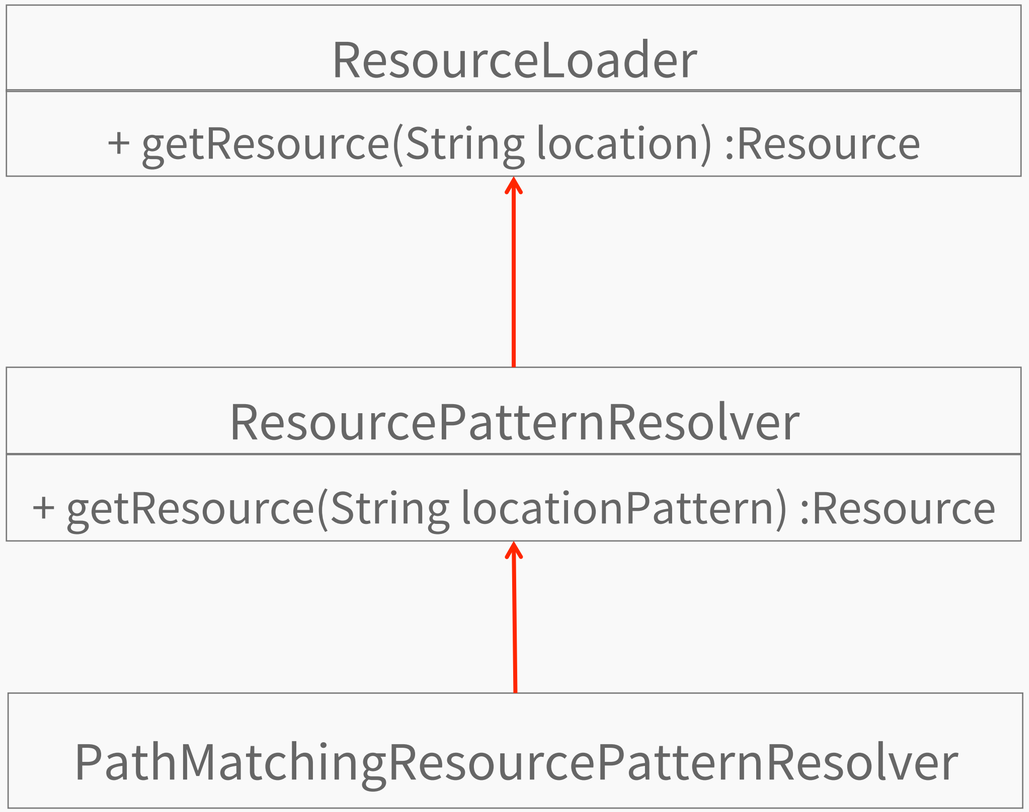
为了访问不同类型的资源，必须使用相应的Resource实现类，这是比较麻烦的，Spring提供了一个强大的加载资源的机制，能够自动识别不同的资源类型：

- 资源类型地址前缀：
  1. classpath classpath:com/jike/bean.xml
  2. File file:/com/jike/bean.xml
  3. http:// http://www.jike.com/bean.xml
  4. ftp ftp://www.jike.com/bean.xml
  5. 无前缀 com/jike/bean.xml

- Ant风格的匹配符：
  1. ？：匹配文件名中的一个字符
  2. *：匹配文件名中的任意字符
  3. **：匹配多层路径
- Ant风格的资源路径示例：
  1. Classpath:com/t*st.xml
  2. File:D:/conf/*.xml
  3. Classpath:com/**/test.xml
  4. Classpath:org/springframework/**/*.xml

资源访问工具类

Spring定义了一套资源加载的接口，并提供了实现类，如下：




```
ResourcePatternResolver resolver =
        new PathMatchingResourcePatternResolver();
//加载所有类包com（及子孙包）下的以xml为后缀的资源
Resource resources[] =
        resolver.getResources("classpath*:com/**/*.xml");
for(Resource resource:resources){
        System.out.println(resource.getDescription());
}
```

BeanFactory和ApplicationContext的介绍

BeanFactory和ApplicationContext的介绍

BeanFactory是Spring框架最核心的接口，它提供了高级IoC的配置机制。ApplicationContext建立在BeanFactory基础之上，提供了更多面向应用的功能，它提供了国际化支持和框架事件体系，更易于创建实际应用一般成BeanFactory为IoC容器，而称ApplicationContext为应用上下文：

BeanFactory和ApplicationContext的介绍 – BeanFactory的介绍

BeanFactory是一个类工厂，可以创建并管理各种类的对象，Spring称这些创建和管理的Java对象为Bean。在Spring中，Java对象的范围更加宽泛。接下来我们对BeanFactory的类体系结构以及装载初始化顺序进行说明：

- 类体系结构：
  1. XmlBeanFactory
  2. ListableBeanFactory
  3. HierarhicalBeanFactory
  4. ConfigurableBeanFactory
  5. AutowireCapableBeanFactory
  6. SingletonBeanFactory
  7. BeanDefinitionRegistry

- 初始化顺序：
  1. 创建配置文件
  2. 装载配置文件
  3. 启动IoC容器
  4. 获取Bean实例

BeanFactory和ApplicationContext的介绍 – ApplicationContext的介绍

ApplicationContext由BeanFactory派生而来，提供了更多面向实际应用的功能。在BeanFactory中，很多功能需要以编程的方式方式实现，而在ApplicationContext中则可以通过配置的方式实现。接下来介绍一下ApplicationContext的实现类以及类体系结构：

- 具体实现类：
  1. ClassPathXmlApplicationContext
  2. FileSystemXmlApplicationContext
  3. ConfigurableApplicationContext
- 扩展接口：
  1. ApplicationEventPublisher
  2. MessageSource
  3. ReaourcePatternResolver
  4. LifeCycle

BeanFactory和ApplicationContext的介绍 – ApplicationContext的介绍

和BeanFactory初始化相似，ApplicationContext的初始化也很简单，根据配置文件路径不同可以选择不同的实现类加载：

- ClassPathXmlApplicationContext
- FileSystemXmlApplicationContext
- Bean的实例化问题

Bean的生命周期

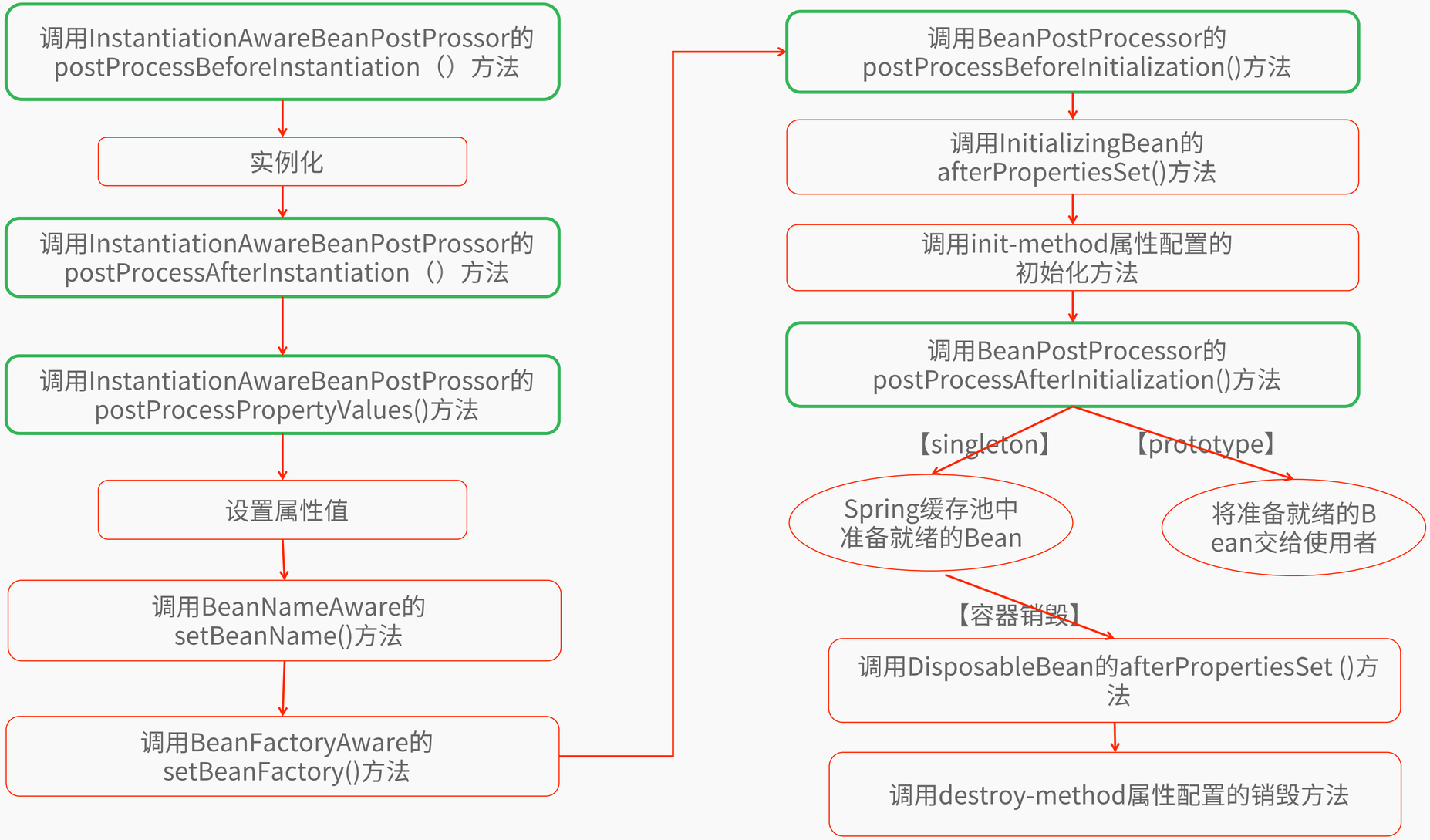
Bean的生命周期

Spring容器中的Bean拥有明确的生命周期，由多个特定的生命阶段组成，每个生命阶段都允许外界对Bean施加控制。在Spring中，我们从Bean的作用范围和实例化Bean时所经历的一系列阶段来描述Bean的生命周期：

- BeanFactory中的Bean的生命周期
- ApplicationContext中的Bean的生命周期

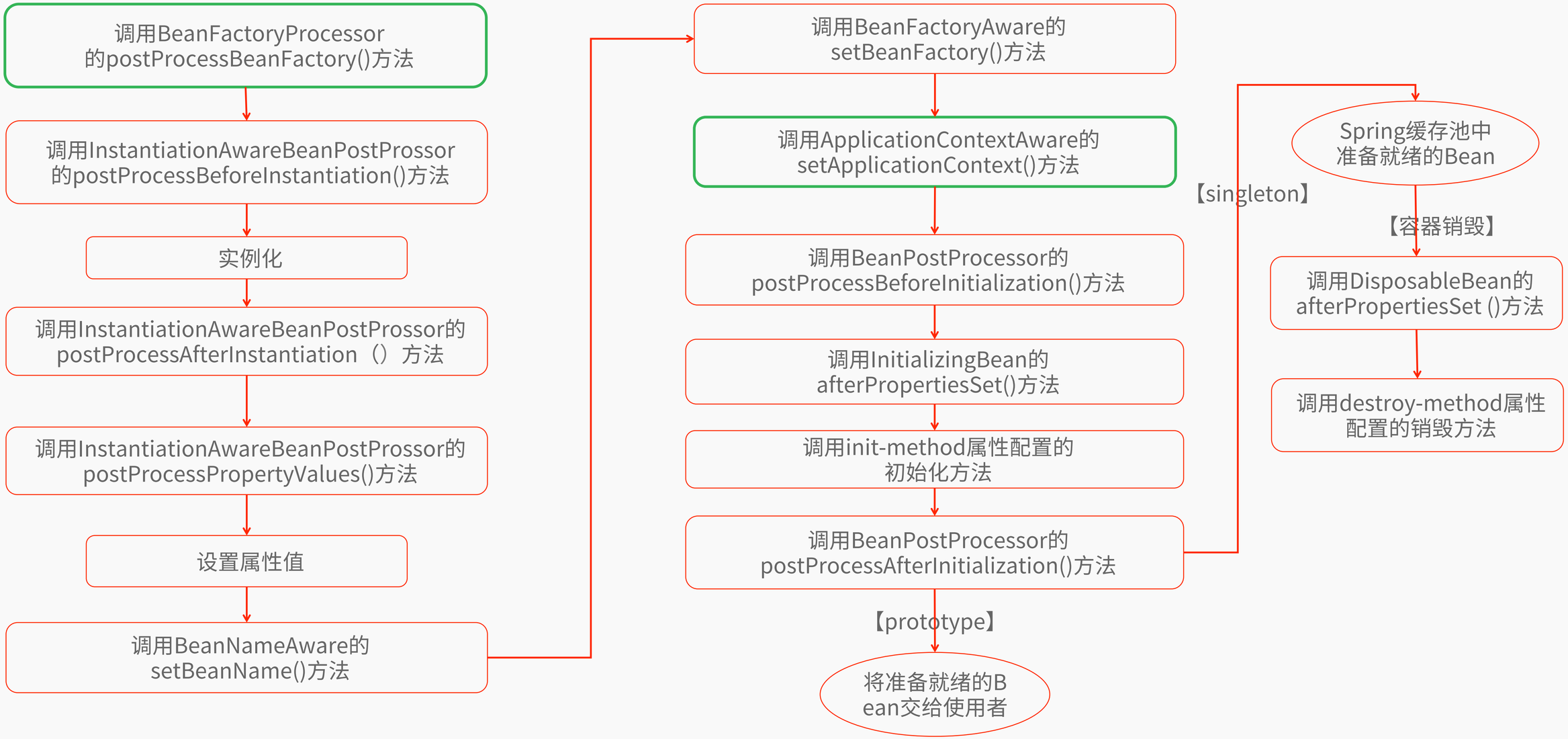
Bean的生命周期 – BeanFactory中的Bean的生命周期

如下通过一个图形化的方式进行描述BeanFactory中的Bean的生命周期：



Bean的生命周期 – ApplicationContext中的Bean的生命周期

如下通过一个图形化的方式进行描述BeanFactory中的Bean的生命周期：



Spring IoC容器的深入理解

在本套课程中我们学习了Spring IoC容器的知识，我们通过简单明了的实例逐步讲解IoC概念，详细分析Bean的生命周期，并探讨了生命周期接口的实际意义。你应当掌握了以下知识：

- 1. IoC概念所包含的设计思想
- 2. Java语言反射技术
- 3. BeanFactory、ApplicationContextt基础接口
- 4. Bean的生命周期

通过本课程的学习，你可以掌握依赖注入的设计思想，实现原理以及几个Spring IoC容器级接口的知识，如果想继续提高，你可以继续在极客学院学习Spring的相关课程。

极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台