在传统的Java应用中，bean的生命周期很简单，使用Java关键字 new 进行Bean 的实例化，然后该Bean 就能够使用了。一旦bean不再被使用，则由Java自动进行垃圾回收。

相比之下，Spring管理Bean的生命周期就复杂多了，Spring对Bean的管理可扩展性非常强，下面展示了一个Bean的构造过程：







可以看出设计策略是“先顾大局”——类似的操作BeanFactory一般出现在Bean之前，操作完Bean之后，BeanFactory会进行“管理”；Bean操作的前提是应用了BeanPostProcessor。

#### Bean生命周期的接口和抽象类

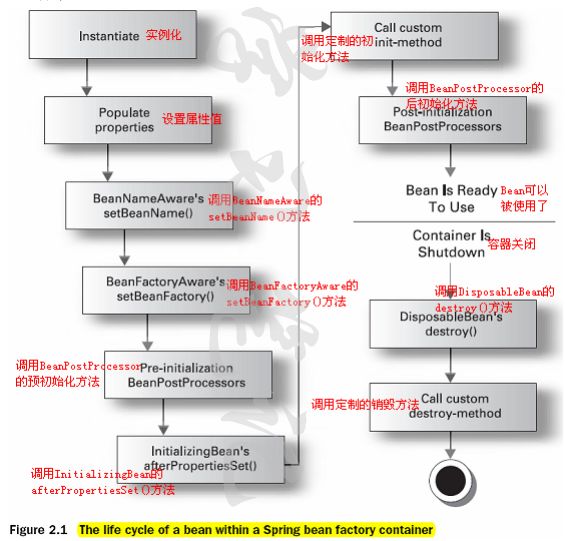
以下为一个回答的参考模板：

1. 实例化一个Bean——也就是我们常说的new；
2. 按照Spring上下文对实例化的Bean进行配置——也就是IoC注入；
3. 如果这个Bean已经实现了BeanNameAware接口，会调用它实现的setBeanName(String)方法，此处传递的就是Spring配置文件中Bean的id值。
4. 如果这个Bean已经实现了BeanFactoryAware接口，会调用它实现的setBeanFactory(setBeanFactory(BeanFactory))传递的是Spring工厂自身（可以用这个方式来获取其他Bean，只需在Spring配置文件中配置一个普通的Bean就可以）；
5. 如果这个Bean已经实现了ApplicationContextAware接口，会调用setApplicationContext(ApplicationContext)方法，传入Spring上下文（同样这个方式也可以实现步骤4的内容，但比4更好，因为ApplicationContext是BeanFactory的子接口，有更多的实现方法）；
6. 如果这个Bean关联了BeanPostProcessor接口，将会调用postProcessBeforeInitialization(Object obj, String s)方法，BeanPostProcessor经常被用作是Bean内容的更改，并且由于这个是在Bean初始化结束时调用那个的方法，也可以被应用于内存或缓存技术；
7. 如果Bean实现了InitializingBean接口，Spring将调用他们的afterPropertiesSet()方法。类似的，如果bean使用init-method声明了初始化方法，该方法也会被调用。
8. 如果这个Bean关联了BeanPostProcessor接口，将会调用postProcessAfterInitialization(Object obj, String s)方法；

注：以上工作完成以后就可以应用这个Bean了，那这个Bean是一个

Singleton的，所以一般情况下我们调用同一个id的Bean会是在内容地址相同的实例，当然在Spring配置文件中也可以配置非Singleton。

1. 此时，Bean已经准备就绪，可以被应用程序使用了。它们将一直驻留在应用上下文中，直到应用上下文被销毁。
2. 当Bean不再需要时，会经过清理阶段，如果Bean实现了DisposableBean接口，会调用那个其实现的destroy()方法；同样如果这个Bean的Spring配置中配置了destroy-method属性，会自动调用其配置的销毁方法。



#### 声明周期的接口和抽象类