下别人的，免费共享了。

Spring bean生命周期

在传统的Java应用中，Bean的生命周期非常简单。

Java的关键词new用来实例化Bean（或许他是非序列化的）。这样就够用了。

相反，Bean的生命周期在Spring容器中更加细致。

理解Spring Bean的生命周期非常重要，因为你或许要利用Spring提供的机会来订制Bean的创建过程。

1.容器寻找Bean的定义信息并且将其实例化。

2.受用依赖注入，Spring按照Bean定义信息配置Bean的所有属性。

3.如果Bean实现了BeanNameAware接口，工厂调用Bean的setBeanName()方法传递Bean的ID。

4.如果Bean实现了BeanFactoryAware接口，工厂调用setBeanFactory()方法传入工厂自身。

5.如果BeanPostProcessor和Bean关联，那么它们的postProcessBeforeInitialzation()方法将被调用。

6.如果Bean指定了init-method方法，它将被调用。

7.最后，如果有BeanPsotProcessor和Bean关联，那么它们的postProcessAfterInitialization()方法将被调用。

到这个时候，Bean已经可以被应用系统使用了，并且将被保留在Bean Factory中知道它不再需要。

有两种方法可以把它从Bean Factory中删除掉。

1.如果Bean实现了DisposableBean接口，destory()方法被调用。

2.如果指定了订制的销毁方法，就调用这个方法。

Bean在Spring应用上下文的生命周期与在Bean工厂中的生命周期只有一点不同，

唯一不同的是，如果Bean实现了ApplicationContextAwre接口，setApplicationContext()方法被调用。

只有singleton行为的bean接受容器管理生命周期。

non-singleton行为的bean，Spring容器仅仅是new的替代，容器只负责创建。

对于singleton bean，Spring容器知道bean何时实例化结束，何时销毁，

Spring可以管理实例化结束之后，和销毁之前的行为，管理bean的生命周期行为主要未如下两个时机：

Bean全部依赖注入之后

Bean即将销毁之前

1）依赖关系注入后的行为实现：

有两种方法：A.编写init方法 B.实现InitializingBean接口

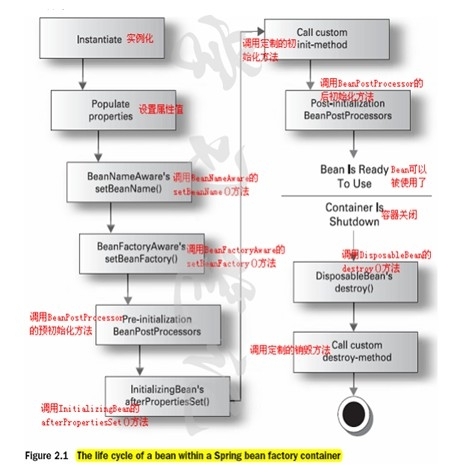
afterPropertiesSet和init同时出现，前者先于后者执行，使用init方法，需要对配置文件加入init-method属性

2）bean销毁之前的行为

有两种方法：A.编写close方法 B.实现DisposableBean接口

destroy和close同时出现，前者先于后者执行，使用close方法，需要对配置文件加入destroy-method属性

**Spring 中bean 的生命周期短暂吗?**    在spring中，从BeanFactory或ApplicationContext取得的实例为Singleton，也就是预设为每一个Bean的别名只能维持一个实例，而不是每次都产生一个新的对象使用Singleton模式产生单一实例，对单线程的程序说并不会有什么问题，但对于多线程的程序，就必须注意安全(Thread-safe)的议题，防止多个线程同时存取共享资源所引发的数据不同步问题。  
    然而在spring中 可以设定每次从BeanFactory或ApplicationContext指定别名并取得Bean时都产生一个新的实例：例如：  
    在spring中，singleton属性默认是true，只有设定为false，则每次指定别名取得的Bean时都会产生一个新的实例  
    一个Bean从创建到销毁，如果是用BeanFactory来生成,管理Bean的话，会经历几个执行阶段(如图1.1)：  
   
   **1：Bean的建立：**    容器寻找Bean的定义信息并将其实例化。  
    **2：属性注入：**    使用依赖注入，Spring按照Bean定义信息配置Bean所有属性  
   **3：BeanNameAware的setBeanName()：**    如果Bean类有实现org.springframework.beans.BeanNameAware接口，工厂调用Bean的setBeanName()方法传递Bean的ID。



**4：BeanFactoryAware的setBeanFactory()：**    如果Bean类有实现org.springframework.beans.factory.BeanFactoryAware接口，工厂调用setBeanFactory()方法传入工厂自身。  
   **5：BeanPostProcessors的ProcessBeforeInitialization()**    如果有org.springframework.beans.factory.config.BeanPostProcessors和Bean关联，那么其postProcessBeforeInitialization()方法将被将被调用。  
  **6：initializingBean的afterPropertiesSet()：**    如果Bean类已实现org.springframework.beans.factory.InitializingBean接口，则执行他的afterProPertiesSet()方法  
   **7：Bean定义文件中定义init-method：**    可以在Bean定义文件中使用"init-method"属性设定方法名称例如：  
    如果有以上设置的话，则执行到这个阶段，就会执行initBean()方法  
    **8：BeanPostProcessors的ProcessaAfterInitialization()**    如果有任何的BeanPostProcessors实例与Bean实例关联，则执行BeanPostProcessors实例的ProcessaAfterInitialization()方法  
    此时，Bean已经可以被应用系统使用，并且将保留在BeanFactory中知道它不在被使用。有两种方法可以将其从BeanFactory中删除掉(如图1.2):  
   
    1：DisposableBean的destroy()  
    在容器关闭时，如果Bean类有实现org.springframework.beans.factory.DisposableBean接口，则执行他的destroy()方法  
    2：Bean定义文件中定义destroy-method  
    在容器关闭时，可以在Bean定义文件中使用"destroy-method"属性设定方法名称，例如：  
    如果有以上设定的话，则进行至这个阶段时，就会执行destroy()方法，如果是使用ApplicationContext来生成并管理Bean的话则稍有不同，使用ApplicationContext来生成及管理Bean实例的话，在执行BeanFactoryAware的setBeanFactory()阶段后，若Bean类上有实现org.springframework.context.ApplicationContextAware接口，则执行其setApplicationContext()方法，接着才执行BeanPostProcessors的ProcessBeforeInitialization()及之后的流程。

