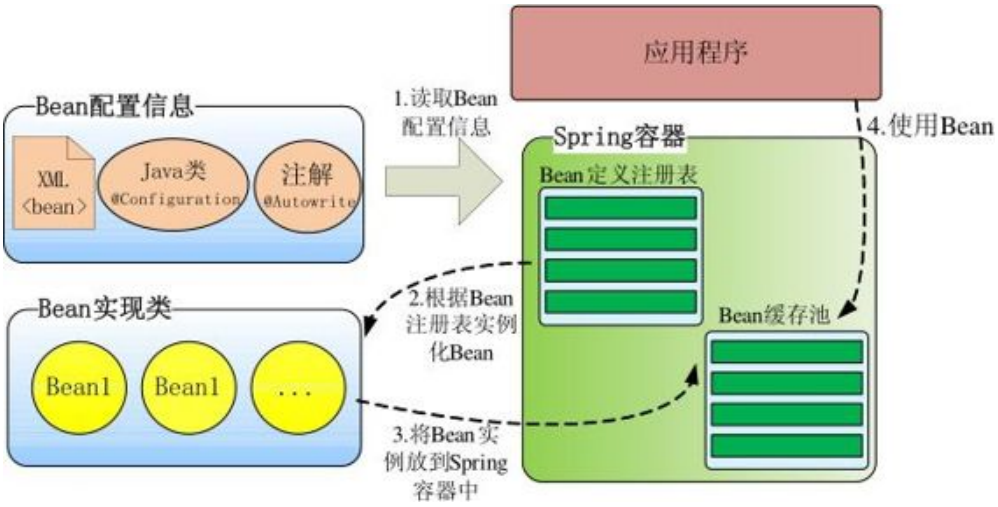
Spring生命周期简介

1. 概要

- Spring容器启动的高层视图；
- 了解Spring容器启动的内部工作机制；
- 通过代码层面了解Bean的生命周期；
- 利用Spring的生命周期加载过程，我们能做些什么？

2.Spring容器技术内幕

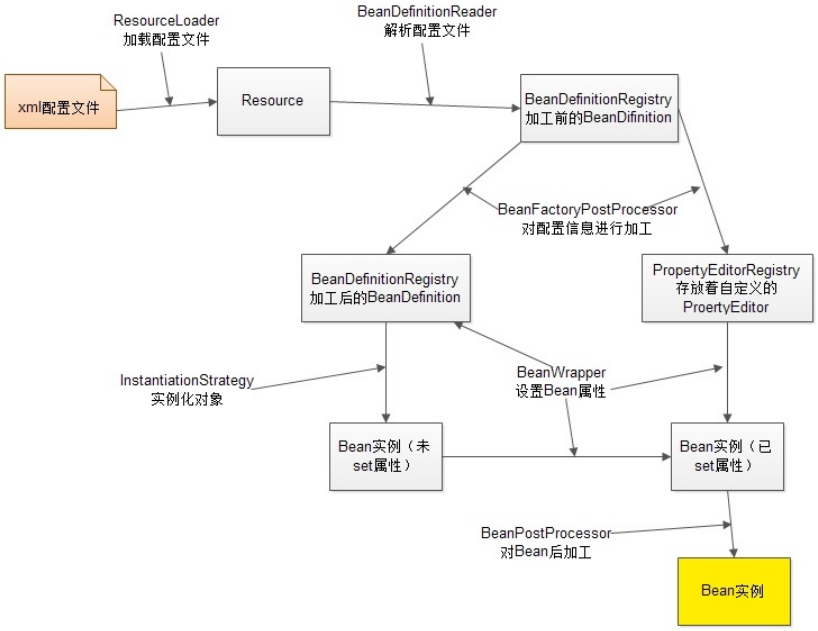
- Spring容器高层视图



Spring 启动时读取应用程序提供的Bean配置信息，Bean配置信息定义了Bean的实现及依赖关系；

> 1）Spring容器根据【各种形式的Bean配置信息】在容器内部建立Bean定义注册表，
>
> 2）然后根据注册表加载、实例化Bean；并建立Bean和Bean之间的依赖关系；
>
> 3）最后将这些准备就绪的Bean放入Bean缓存池中，以供外层的应用程序调用；

- 内部工作机制



1. ResourceLoader从存储介质中(xml、@Configuration等)加载Spring配置信息，并使用Resource表示这个配置文件的资源；

2. BeanDefinitionReader读取Resource所指向的配置文件资源，然后解析配置文件。配置文件中每一个解析成一个BeanDefinition对象，并保存到BeanDefinitionRegistry中；

3. 容器扫描BeanDefinitionRegistry中的BeanDefinition，使用Java的反射机制自动识别出Bean工厂后处理后器（实现BeanFactoryPostProcessor接口）的Bean，然后调用这些Bean工厂后处理器对BeanDefinitionRegistry中的BeanDefinition进行加工处理。

    - 主要完成以下两项工作：

        > - 对使用到占位符的元素标签进行解析，得到最终的配置值，这意味对一些半成品式的BeanDefinition对象进行加工处理并得到成品的BeanDefinition对象；
        > - 对BeanDefinitionRegistry中的BeanDefinition进行扫描，通过Java反射机制找出所有属性编辑器的Bean（实现java.beans.PropertyEditor接口的Bean），并自动将它们注册到Spring容器的属性编辑器注册表中（PropertyEditorRegistry）；

4. Spring容器从BeanDefinitionRegistry中取出加工后的BeanDefinition，并调用InstantiationStrategy着手进行Bean实例化的工作；

5. 在实例化Bean时，Spring容器使用BeanWrapper对Bean进行封装，BeanWrapper提供了很多以Java反射机制操作Bean的方法，它将结合该Bean的BeanDefinition以及容器中属性编辑器，完成Bean属性的设置工作；

6. 利用容器中注册的Bean后处理器（实现BeanPostProcessor接口的Bean）对已经完成属性设置工作的Bean进行后续加工，直接装配出一个准备就绪的Bean；

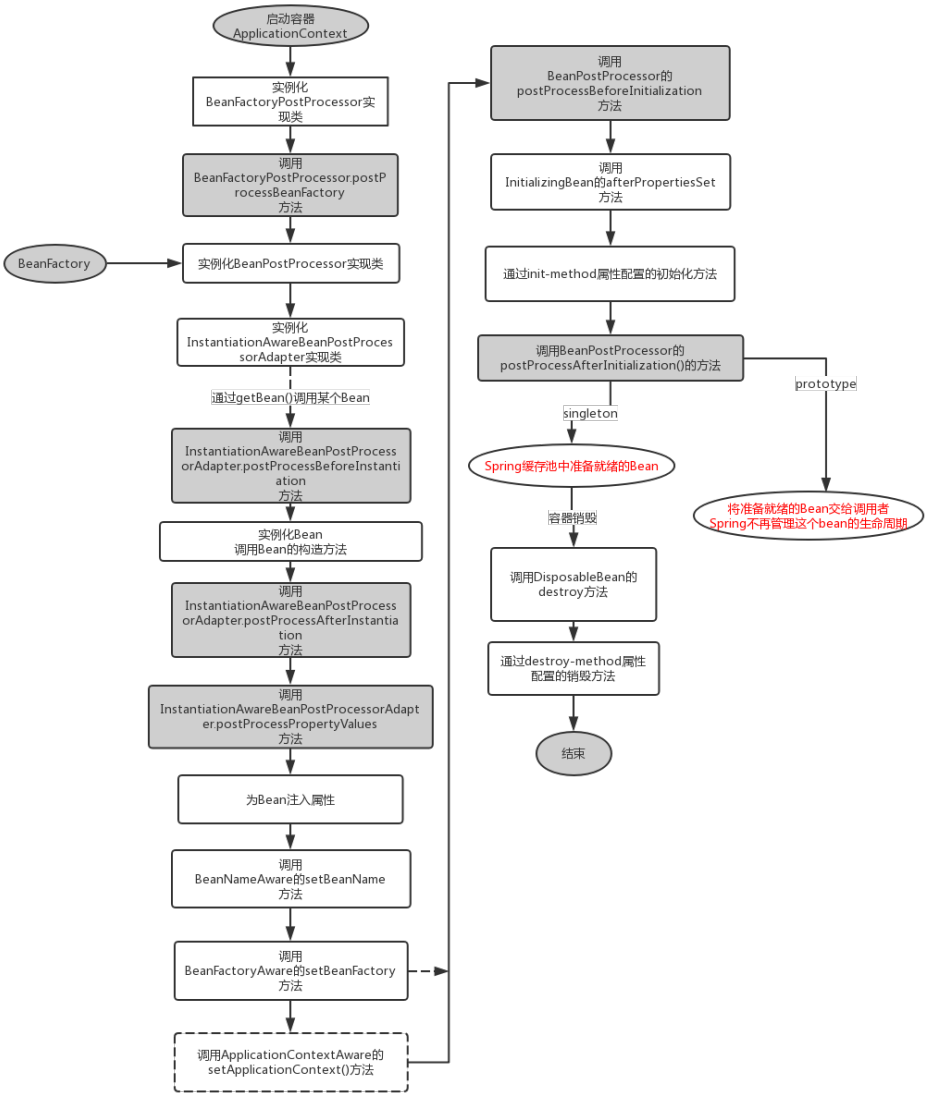
- BeanFactory、ApplicationContext

> 1. 两者都是通过xml配置文件加载bean,ApplicationContext和BeanFacotry相比,提供了更多的扩展功能，但其主要区别在于后者是延迟加载,如果Bean的某一个属性没有注入，BeanFacotry加载后，直至第一次使用调用getBean方法才会抛出异常；而ApplicationContext则在初始化自身是检验，这样有利于检查所依赖属性是否注入；所以通常情况下我们选择使用ApplicationContext.。
> 2. ApplicationContext接口,它由BeanFactory接口派生而来，因而提供BeanFactory所有的功能。ApplicationContext以一种更向面向框架的方式工作以及对上下文进行分层和实现继承，ApplicationContext包还提供了以下的功能： ·MessageSource, 提供国际化的消息访问 ·资源访问，如URL和文件 ·事件传播 ·载入多个（有继承关系）上下文 ，使得每一个上下文都专注于一个特定的层次； 【springMvc的父子容器】
> 3. 其他区别：
>   - BeanFactory 采用的是延迟加载形式来注入Bean的，即只有在使用到某个Bean时(调用getBean())，才对该Bean进行加载实例化，这样，我们就不能发现一些存在的Spring的配置问题。而ApplicationContext则相反，它是在容器启动时，一次性创建了所有的Bean。这样，在容器启动时，我们就可以发现Spring中存在的配置错误。
>   - BeanFactory和ApplicationContext都支持BeanPostProcessor、BeanFactoryPostProcessor的使用，但两者之间的区别是：BeanFactory需要手动注册，而ApplicationContext则是自动注册
> 4. 参考：http://blog.csdn.net/hi_kevin/article/details/7325554

3.Bean的生命周期

- Spring Bean的完整生命周期；

> SpringBean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；



- Bean的完整生命周期经历了各种方法调用，这些方法可以划分为以下几类：
  - Bean自身的方法：
    > - 包括了Bean本身调用的方法
    > - 通过配置文件中的init-method和destroy-method指定的方法
  - Bean级生命周期接口方法
    > - BeanNameAware、BeanFactoryAware、InitializingBean和DiposableBean
  - 容器级生命周期接口方法

> - 包括InstantiationAwareBeanPostProcessor 和 BeanPostProcessor 这两个接口实现，一般称它们的实现类为“后处理器”。
> - 后处理器接口一般不由Bean本身实现，它独立于Bean，实现类似于以容器附加形式注册到Spring容器中，并且通过反射的方式在Spring容器中预先识别；
> - 当Spring容器创建任何Bean的时候这些后处理器都会发生作用，所以后处理器的的影响是全局性的；

  - 工厂后处理器接口方法

> 包括BeanFactoryPostProcessor等等非常有用的工厂后处理器接口的方法。工厂后处理器也是容器级的

- BeanFactoryPostProcessor 、 BeanPostProcessor、 InstantiationAwareBeanPostProcessor作用的区别在哪?
  - BeanFactoryPostProcessor:
    - spring中有内置的一些BeanFactoryPostProcessor实现类常用的有：

> org.springframework.beans.factory.config.PropertyPlaceholderConfigurer
>
> org.springframework.beans.factory.config.PropertyOverrideConfigurer
>
> org.springframework.beans.factory.config.CustomEditorConfigurer

    - 参考：http://blog.csdn.net/caihaijiang/article/details/35331979
  - BeanPostProcessor
    - BeanNameAutoProxyCreator
  - InstantiationAwareBeanPostProcessor 调用

> 当实例化bean之前 AbstractAutowireCapableBeanFactory#createBean()；
>
> Object bean = resolveBeforeInstantiation(beanName, mbd);

  - Spring的BeanFactoryPostProcessor和BeanPostProcessor区别；

> 参考： http://blog.csdn.net/caihaijiang/article/details/35552859

- Spring的事件处理

> - Spring提供了部分内置事件，主要有以下几种：
>   - ContextRefreshedEvent ： ApplicationContext发送该事件时，表示该容器中所有的Bean都已经被装载完成，此ApplicationContext已就绪可用
>   - ContextStartedEvent：生命周期 beans的启动信号
>   - ContextStoppedEvent: 生命周期 beans的停止信号
>   - ContextClosedEvent：ApplicationContext关闭事件，则context不能刷新和重启，从而所有的singleton bean全部销毁(因为singleton bean是存在容器缓存中的)

- Spring MVC父子容器；

> 1. 通过HierarchicalBeanFactory接口，Spring的IoC容器可以建立父子层级关联的容器体系，子容器可以访问父容器中的Bean，但父容器不能访问子容器的Bean。在容器内，Bean的id必须是唯一的，但子容器可以拥有一个和父容器id相同的Bean。父子容器层级体系增强了Spring容器架构的扩展性和灵活性，因为第三方可以通过编程的方式，为一个已经存在的容器添加一个或多个特殊用途的子容器，以提供一些额外的功能。
> 2. Spring使用父子容器实现了很多功能，比如在Spring MVC中，展现层Bean位于一个子容器中，而业务层和持久层的Bean位于父容器中。这样，展现层Bean就可以引用业务层和持久层的Bean，而业务层和持久层的Bean则看不到展现层的Bean。