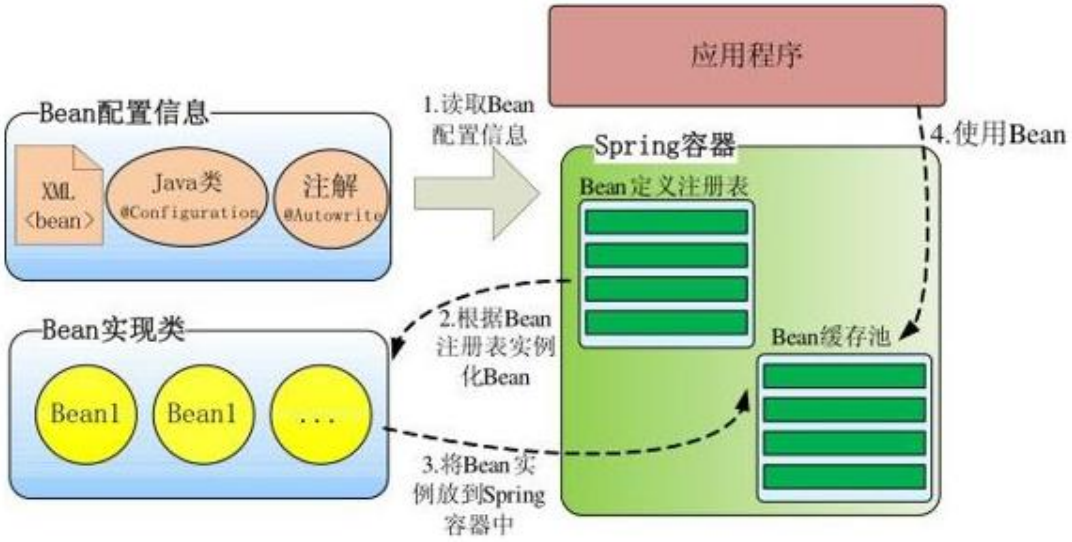
Spring 底层源码分析

一、 Spring 回顾

Spring 案例

二、 Spring 运行原理

Spring 启动时读取应用程序提供的 Bean 配置信息，并在 Spring 容器中生成一份相应的 Bean 配置注册表，然后根据这张注册表实例化 Bean，装配好 Bean 之间的依赖关系，为上层应用提供准备就绪的运行环境。



三、 Spring 源码分析

1 ApplicationContext

1.1 Spring 中 IOC 容器分类

Spring 中有两个主要的容器系列：

1）实现 BeanFactory 接口的简单容器；

2）实现 ApplicationContext 接口的高级容器。

1.2 ApplicationContext 容器介绍

ApplicationContext 内部封装了一个 BeanFactory 对象，来实现对容器的操作，BeanFactory 封装了 bean 的信息，而 ApplicationContext 通过访问 BeanFactory 对象获取 bean 的对象信息 ApplicationContext 也实现了一系列的 BeanFactory 接口(可以说 ApplicationContext 对 BeanFactory 对象实现一种代理)。

ApplicationContext 在应用这个 DefaultListableBeanFactory 对象的基础上，不仅实现了 BeanFactory 接口提供的功能方法，并且黏合了一些面向应用的功能，如资源/国际化支持/框架事件支持等

```
public interface ApplicationContext extends EnvironmentCapable,
                                ListableBeanFactory,    //继承于 BeanFactory
                                HierarchicalBeanFactory,//继承于 BeanFactory
                                MessageSource,          //
                                ApplicationEventPublisher,//
                                ResourcePatternResolver   //继承 ResourceLoader,
用于获取 resource 对象
```

2 ClassPathXmlApplicationContext

2.1 refresh()方法

2.2 实例化 Bean 工厂

2.2.1 DefaultListableBeanFactory

在 BeanFactory 子类中有一个 DefaultListableBeanFactory 类，它包含了基本 Spirng IoC 容器所具有的重要功能，开发时不论是使用 BeanFactory 系列还是 ApplicationContext 系列来创建容器基本都会使用到 DefaultListableBeanFactory 类，可以这么说，在 spring 中实际上把它当成默认的 IoC 容器来使用

2.3解析配置文件

2.3.1XmlBeanDefinitionReader

```
 */
protected void parseBeanDefinitions(Element root, BeanDefinitionParserDelega
    if (delegate.isDefaultNamespace(root)) {
        NodeList nl = root.getChildNodes();
        for (int i = 0; i < nl.getLength(); i++) {
            Node node = nl.item(i);
            if (node instanceof Element) {
                Element ele = (Element) node;
                if (delegate.isDefaultNamespace(ele)) {
                    parseDefaultElement(ele, delegate);
                }
                else {
                    delegate.parseCustomElement(ele);
                }
            }
        }
    }
```

3 将 BeanDefiniton 注册到容器中

```
    else {
        this.beanDefinitionNames.add(beanName);
        this.manualSingletonNames.remove(beanName);
        this.frozenBeanDefinitionNames = null;
    }
    this.beanDefinitionMap.put(beanName, beanDefinition);

    if (oldBeanDefinition != null || containsSingleton(beanName)) {
        resetBeanDefinition(beanName);
    }
}
```

4 Bean 对象的实例化

```
@Override
public void preInstantiateSingletons() throws BeansException {
    if (this.logger.isDebugEnabled()) {
        this.logger.debug("Pre-instantiating singletons in " + this);
    }

    // Iterate over a copy to allow for init methods which in turn register
    // While this may not be part of the regular factory bootstrap, it does
    List<String> beanNames = new ArrayList<String>(this.beanDefinitionNames

    // Trigger initialization of all non-lazy singleton beans...
    for (String beanName : beanNames) {
        RootBeanDefinition bd = getMergedLocalBeanDefinition(beanName);
        if (!bd.isAbstract() && bd.isSingleton() && !bd.isLazyInit()) {
            if (isFactoryBean(beanName)) {
                final FactoryBean<?> factory = (FactoryBean<?>) getBean(FAC
                boolean isEagerInit;
                if (System.getSecurityManager() != null && factory instance
                    isEagerInit = AccessController.doPrivileged(new Privile
                        @Override
```

5 GetBean

```
        throws BeansException {

    final String beanName = transformedBeanName(name);
    Object bean;

    // Eagerly check singleton cache for manually registered singletons.
    Object sharedInstance = getSingleton(beanName);
    if (sharedInstance != null && args == null) {
        if (logger.isDebugEnabled()) {
            if (isSingletonCurrentlyInCreation(beanName)) {
                logger.debug("Returning eagerly cached instance of singleton bean '" + beanName
                        "' that is not fully initialized yet - a consequence of a circular refer
            }
            else {
                logger.debug("Returning cached instance of singleton bean '" + beanName + "'");
            }
        }
        bean = getObjectForBeanInstance(sharedInstance, name, beanName, null);
    }

    else {
```