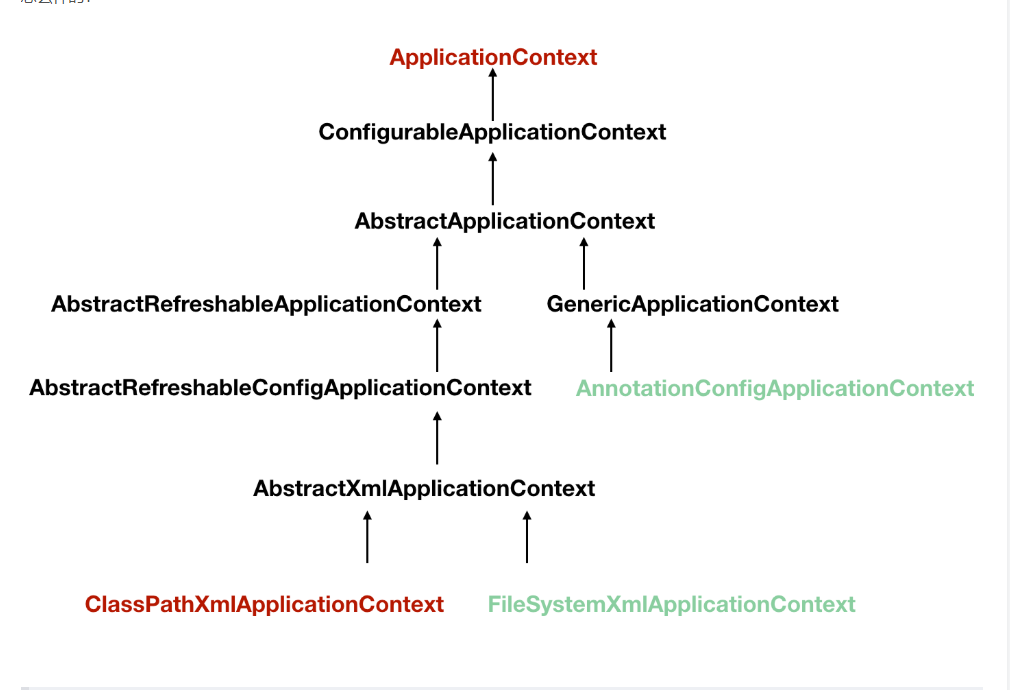
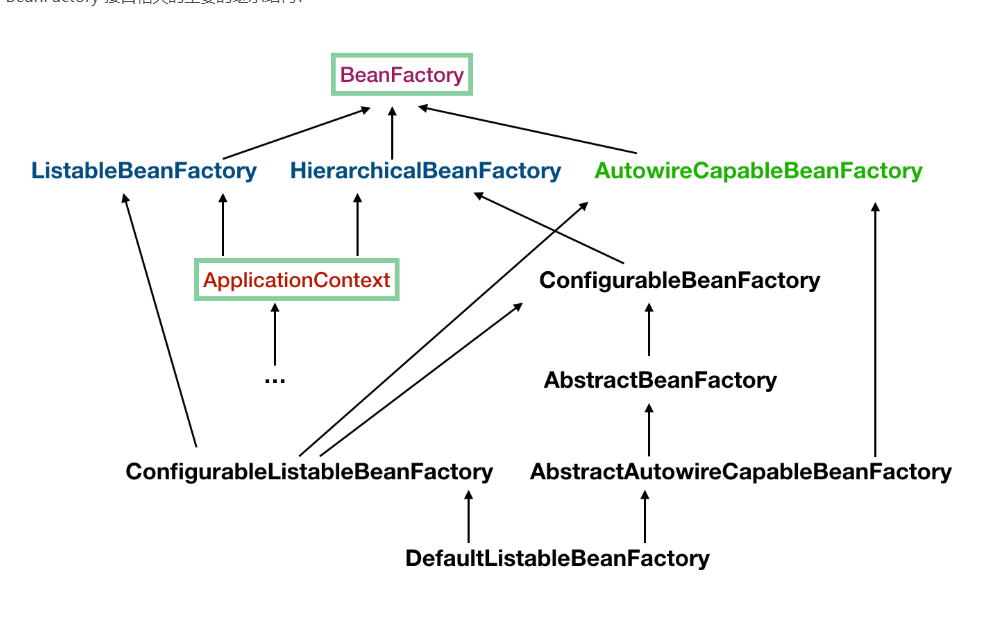
# ApplicationContext



# BeanFactory



1. ApplicationContext 继承了 ListableBeanFactory，这个 Listable 的意思就是，通过这个接口，我们可以获取多个 Bean，最顶层 BeanFactory 接口的方法都是获取单个 Bean 的。
2. ApplicationContext 继承了 HierarchicalBeanFactory，Hierarchical 单词本身已经能说明问题了，也就是说我们可以在应用中起多个 BeanFactory，然后可以将各个 BeanFactory 设置为父子关系。
3. AutowireCapableBeanFactory 这个名字中的 Autowire 大家都非常熟悉，它就是用来自动装配 Bean 用的，但是仔细看上图，ApplicationContext 并没有继承它，不过不用担心，不使用继承，不代表不可以使用组合，如果你看到 ApplicationContext 接口定义中的最后一个方法 getAutowireCapableBeanFactory() 就知道了。
4. ConfigurableListableBeanFactory 也是一个特殊的接口，看图，特殊之处在于它继承了第二层所有的三个接口，而 ApplicationContext 没有。这点之后会用到。

# 容器介绍

每个bean在容器中都有一个唯一的名字和0或多个别名。



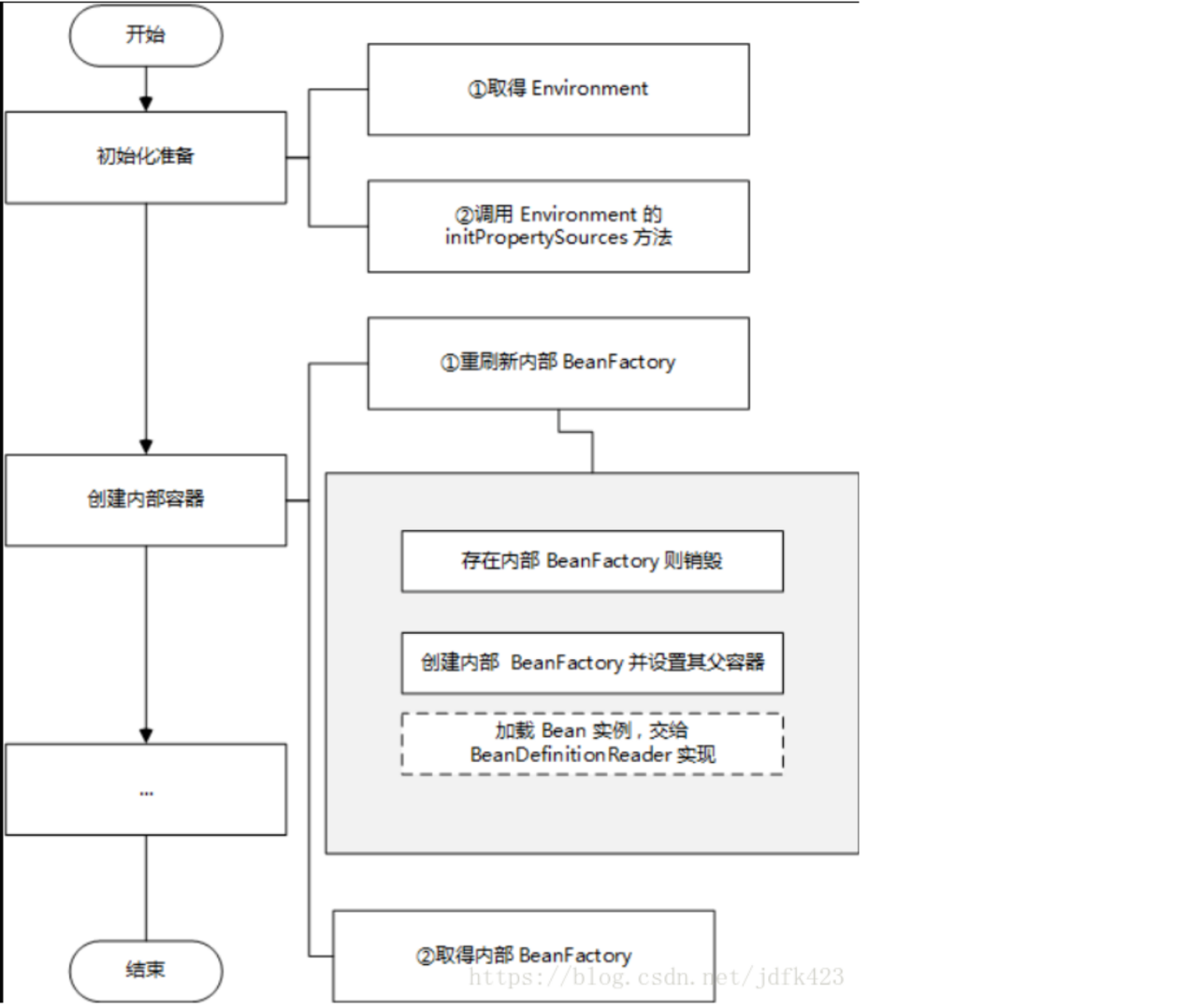
ApplicationContext实例化的时候同时将所有的对象(单例)实例化，如果不是单例也不会预加载。这种方式就消耗内存但是获取对象的速度快。

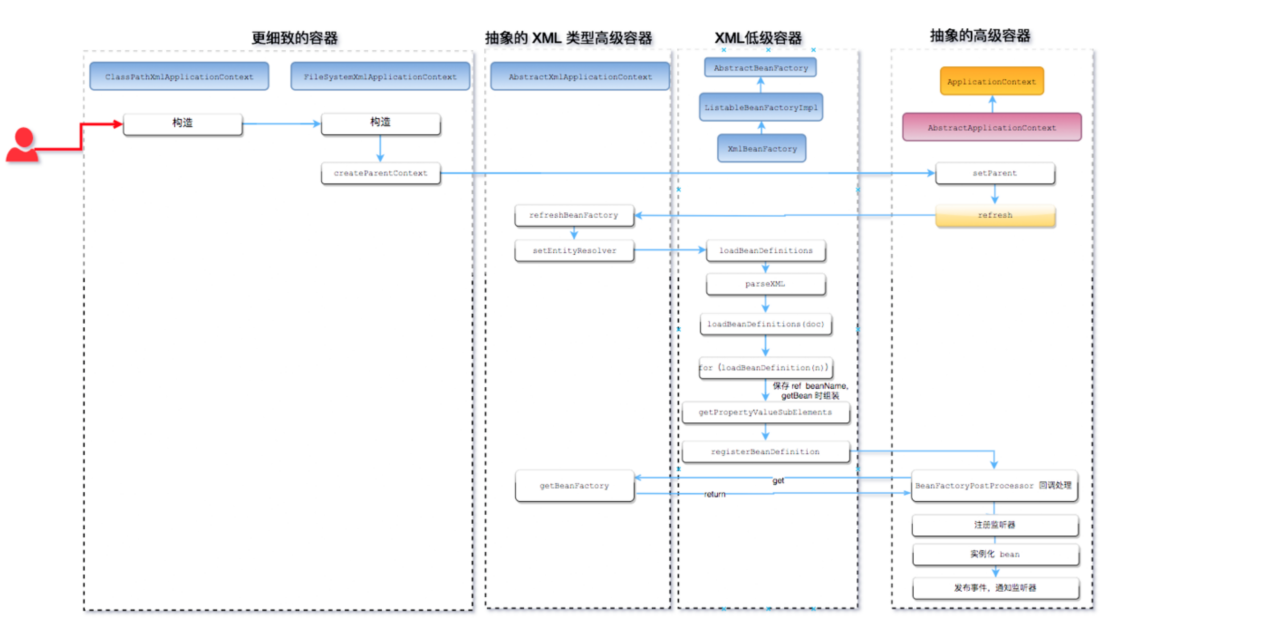
Beanfactory实例化容器的时候根据配置判断哪些对象立即实例化，如果没有配置立即实例化，只有在getBean的时候容器才会将对象实例化。缺点是获取对象慢点，并且报空指针异常的错误。

# AbstractApplicationContext

最终都会调用这个类中的refresh方法。

# 创建过程





1. 用户构造ClassPathXmlApplicationContext（简称CPXAC）
2. CPXAC首先访问高级容器的refresh方法，这个方法调用refreshBeanFactory方法，这个方法的作用就是使用低级容器加载所有的BeanDefinition和Properties到容器中
3. 低级容器加载成功后，高级容器就会处理一些回调。例如Bean的前置或者后置处理器，或者注册监听器，发布事件，实例化单例Bean等等功能。

# getBean

获取bean的时候有个递归调用，

当 Bean\_A 依赖着 Bean\_B，而这个 Bean\_A 在加载的时候，其配置的 ref = “Bean\_B” 在解析的时候只是一个占位符，被放入了 Bean\_A 的属性集合中，当调用 getBean 时，需要真正 Bean\_B 注入到 Bean\_A 内部时，就需要从容器中获取这个 Bean\_B，因此产生了递归。

为什么不是在加载的时候，就直接注入呢？因为加载的顺序不同，很可能 Bean\_A 依赖的 Bean\_B 还没有加载好，也就无法从容器中获取

调用 getBean 的时候，从 BeanDefinition 所属的 Map 里，拿出 Class 对象进行实例化，同时，如果有依赖关系，将递归调用 getBean 方法 —— 完成依赖注入。

加载所有的 Bean 配置成 BeanDefinition 到容器中，如果 Bean 有依赖关系，则使用占位符暂时代替。

然后，在调用 getBean 的时候，进行真正的依赖注入，即如果碰到了属性是 ref 的（占位符），那么就从容器里获取这个 Bean，然后注入到实例中 —— 称之为依赖注入。

依赖注入是在getBean方法调用时执行的，如果bean有依赖关系，就会递归调用getBean方法，将这个依赖注入到bean中。

# 注入类型

## byType类型

如果一个接口只有一个实现类，可以使用btType类型注入。

因为@Autowired 的注入方式是 byType 注入

## byName名称

如果一个接口有多个实现类，那么根据类型注入就会报错。此时要根据名称注入，@Resource或者@Qualifier

@Resource根据的是根据类在容器中的名字获取的。

@Qualifier根据类名获取的。

如上面所说，@autowired自动寻找 applicationContext.xml 里 该接口的实现类，而@Resource根据bean id 注入在赋值给注释下的接口

@autowired和@Resourc相当于  UserService userService = (UserService) applicationContext.getBean("userService");

# Spring是如何解决循环依赖的

A依赖B，B又依赖C，但是C又依赖A时。当创建C时A还没创建完成，那么C怎么创建呢？此时容器就会将A对象的早期引用返回给C，让C完成创建。

A对象的早期引用还没有注入参数。

Spring用的是setter方法解决的循环依赖问题，而没有解决构造器的循环依赖问题。

