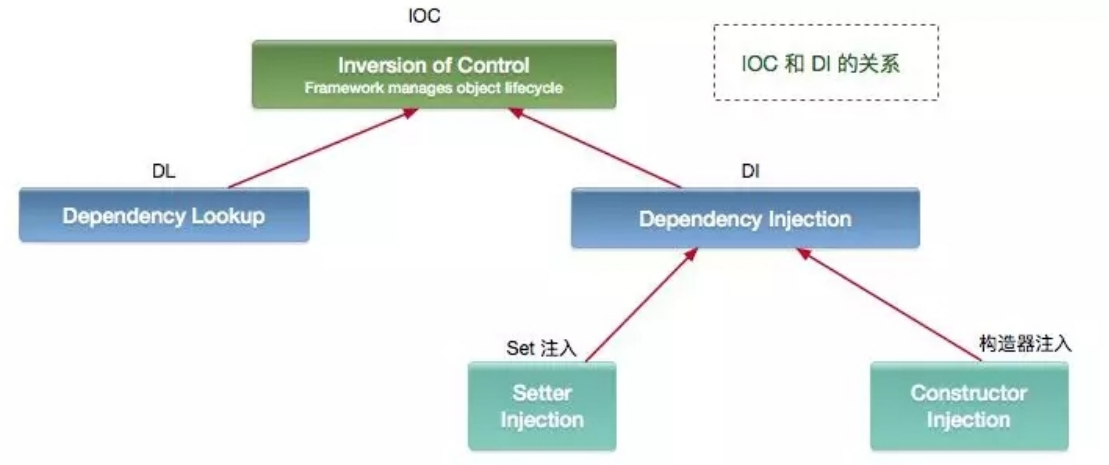
广义的 IOC

1. IoC(Inversion of Control) 控制反转，即“不用打电话过来，我们会打给你”。

   两种实现： 依赖查找（DL）和依赖注入（DI）。

   IOC 和 DI 、DL 的关系（这个 DL，Avalon 和 EJB 就是使用的这种方式实现的 IoC）：



2. DL 已经被抛弃，因为他需要用户自己去是使用 API 进行查找资源和组装对象。即有侵入性。
3. DI 是 Spring 使用的方式，容器负责组件的装配。

   注意：Java 使用 DI 方式实现 IoC 的不止 Spring，包括 Google 的 Guice，还有一个冷门的 PicoContainer（极度轻量，但只提供 IoC）。

Spring 的 IoC

Spring 的 IoC 设计支持以下功能：

1. 依赖注入
2. 依赖检查
3. 自动装配
4. 支持集合
5. 指定初始化方法和销毁方法
6. 支持回调某些方法（但是需要实现 Spring 接口，略有侵入）

其中，最重要的就是依赖注入，从 XML 的配置上说， 即 ref 标签。对应 Spring RuntimeBeanReference 对象。

对于 IoC 来说，最重要的就是容器。容器管理着 Bean 的生命周期，控制着 Bean 的依赖注入。

那么， Spring 如何设计容器的呢？

Spring 作者 Rod Johnson 设计了两个接口用以表示容器。

1. BeanFactory
2. ApplicationContext

BeanFactory 粗暴简单，可以理解为就是个 HashMap，Key 是 BeanName，Value 是 Bean 实例。通常只提供注册（put），获取（get）这两个功能。我们可以称之为 “低级容器”。

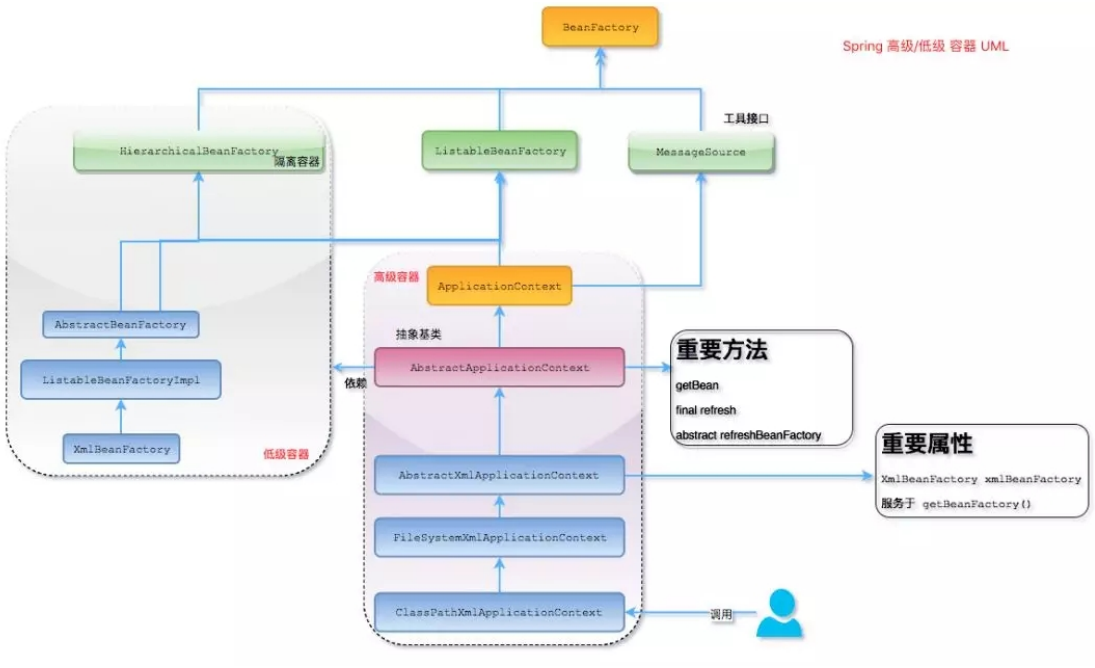
ApplicationContext 可以称之为 “高级容器”。因为他比 BeanFactory 多了更多的功能。他继承了多个接口。因此具备了更多的功能。

> 例如资源的获取，支持多种消息（例如 JSP tag 的支持），对 BeanFactory 多了工具级别的支持等待。所以你看他的名字，已经不是 BeanFactory 之类的工厂了，而是 “应用上下文”， 代表着整个大容器的所有功能。

该接口定义了一个 refresh 方法，此方法是所有阅读 Spring 源码的人的最熟悉的方法，用于刷新整个容器，即重新加载/刷新所有的 bean。

当然，除了这两个大接口，还有其他的辅助接口，但我今天不会花太多篇幅介绍他们。

为了更直观的展示 “低级容器” 和 “高级容器” 的关系，我这里通过常用的 ClassPathXmlApplicationContext 类，来展示整个容器的层级 UML 关系。



有点复杂？ 先不要慌，我来解释一下。

最上面的 BeanFactory 知道吧？我就不讲了。

下面的 3 个绿色的，都是功能扩展接口，这里就不展开讲。

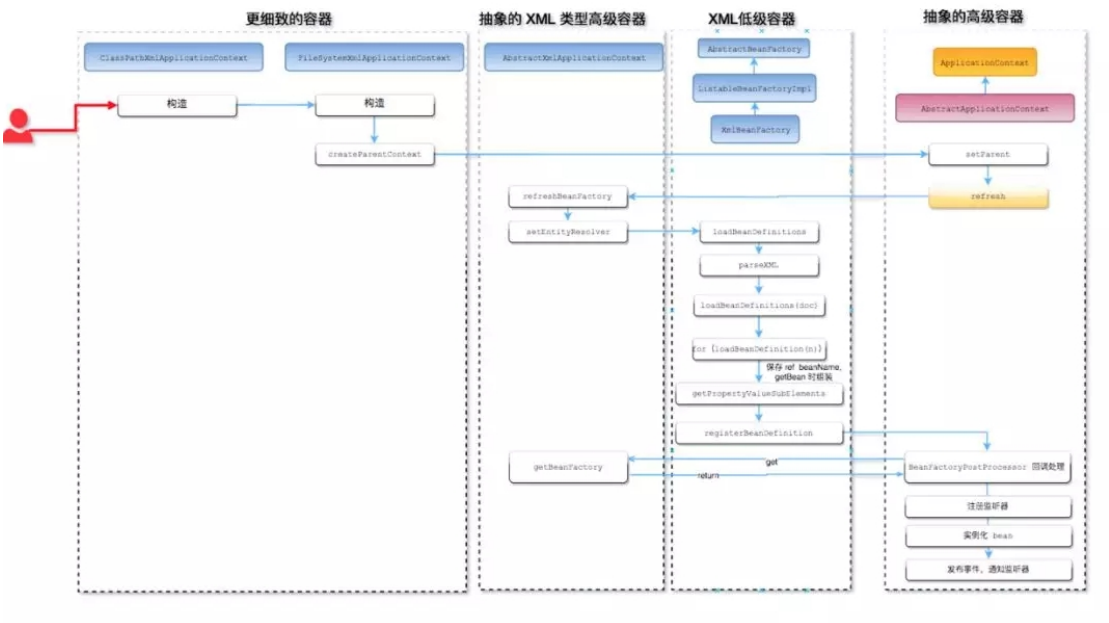
看下面的隶属 ApplicationContext 粉红色的 “高级容器”，依赖着 “低级容器”，这里说的是依赖，不是继承哦。他依赖着 “低级容器” 的 getBean 功能。而高级容器有更多的功能：支持不同的信息源头，可以访问文件资源，支持应用事件（Observer 模式）。

通常用户看到的就是 “高级容器”。 但 BeanFactory 也非常够用啦！

左边灰色区域的是 “低级容器”，只负载加载 Bean，获取 Bean。容器其他的高级功能是没有的。例如上图画的 refresh 刷新 Bean 工厂所有配置。生命周期事件回调等。

> 好，解释了低级容器和高级容器，我们可以看看一个 IoC 启动过程是什么样子的。说白了，就是 ClassPathXmlApplicationContext 这个类，在启动时，都做了啥。（由于我这是 interface21 的代码，肯定和你的 Spring 4.x 系列不同）。

下图是 ClassPathXmlApplicationContext 的构造过程，实际就是 Spring IoC 的初始化过程。



注意，这里为了理解方便，有所简化。

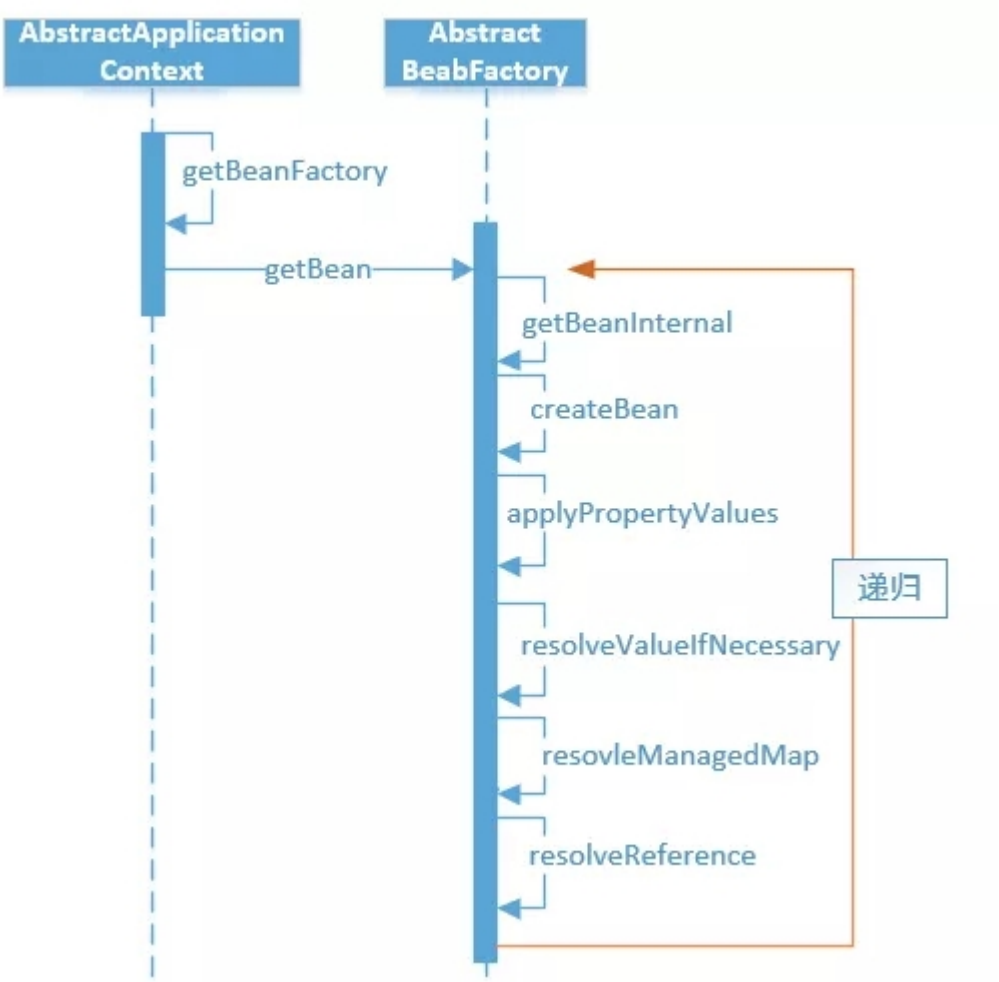
这里再用文字来描述这个过程：

1. 用户构造 ClassPathXmlApplicationContext（简称 CPAC）
2. CPAC 首先访问了 “抽象高级容器” 的 final 的 refresh 方法，这个方法是模板方法。所以要回调子类（低级容器）的 refreshBeanFactory 方法，这个方法的作用是使用低级容器加载所有 BeanDefinition 和 Properties 到容器中。
3. 低级容器加载成功后，高级容器开始处理一些回调，例如 Bean 后置处理器。回调 setBeanFactory 方法。或者注册监听器等，发布事件，实例化单例 Bean 等等功能，这些功能，随着 Spring 的不断升级，功能越来越多，很多人在这里迷失了方向 ：）。

简单说就是：

1. 低级容器 加载配置文件（从 XML，数据库，Applet），并解析成 BeanDefinition 到低级容器中。
2. 加载成功后，高级容器启动高级功能，例如接口回调，监听器，自动实例化单例，发布事件等等功能。
   所以，一定要把 “低级容器” 和 “高级容器” 的区别弄清楚。不能一叶障目不见泰山。

好，当我们创建好容器，就会使用 getBean 方法，获取 Bean，而 getBean 的流程如下：



从图中可以看出，getBean 的操作都是在低级容器里操作的。其中有个递归操作，这个是什么意思呢？

> 假设：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，你不能要求用户把 Bean 的加载顺序排列好，这是不人道的。

所以，Spring 将其分为了 2 个步骤：

1. 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。
2. 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。
   可以看到，依赖注入实际上，只需要 “低级容器” 就可以实现。

   这就是 IoC。

   所以 ApplicationContext refresh 方法里面的操作不只是 IoC，是高级容器的所有功能（包括 IoC），IoC 的功能在低级容器里就可以实现。

总结

说了这么多，不知道你有没有理解 Spring IoC？ 这里小结一下：IoC 在 Spring 里，只需要低级容器就可以实现，2 个步骤：

1. 加载配置文件，解析成 BeanDefinition 放在 Map 里。
2. 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。

上面就是 Spring 低级容器（BeanFactory）的 IoC。

至于高级容器 ApplicationContext，他包含了低级容器的功能，当他执行 refresh 模板方法的时候，将刷新整个容器的 Bean。同时其作为高级容器，包含了太多的功能。一句话，他不仅仅是 IoC。他支持不同信息源头，支持 BeanFactory 工具类，支持层级容器，支持访问文件资源，支持事件发布通知，支持接口回调等等。

可以预见，随着 Spring 的不断发展，高级容器的功能会越来越多。

> 诚然，了解 IoC 的过程，实际上为了了解 Spring 初始化时，各个接口的回调时机。例如 InitializingBean，BeanFactoryAware，ApplicationListener 等等接口，这些接口的作用，笔者之前写过一篇文章进行介绍，有兴趣可以看一下，关键字：Spring 必知必会 扩展接口。

但是请注意，实现 Spring 接口代表着你这个应用就绑定死 Spring 了！代表 Spring 具有侵入性！要知道，Spring 发布时，无侵入性就是他最大的宣传点之一 —— 即 IoC 容器可以随便更换，代码无需变动。而现如今，Spring 已然成为 J2EE 社区准官方解决方案，也没有了所谓的侵入性这个说法。因为他就是标准，和 Servlet 一样，你能不实现 Servlet 的接口吗？: -)

好了，下次如果再有面试官问 Spring IoC 初始化过程，就再也不会含糊其词、支支吾吾了！！！