Spring 启动

以 `ClassPathXmlApplicationContext` 为例

- 创建 `StandardEnvironment` --- *New ClassPathXmlApplicationContext("classpath:application.xml")*
- 起点 --- *refresh()*
  - 设置启动时间 `startupDate` --- *prepareRefresh()*
    - 初始化上下文placeholder property sources，默认无实现 --- *initPropertySources()*
  - 验证所有必填的Property(使用到前面创建的StandardEnvironment)
  - 创建 `earlyApplicationEvents` (类型Set),一旦广播器创建成功，就立即执行广播事件；
  - 刷新BeanFactory --- *obtainFreshBeanFactory()*
    - destroy&close `beanFactory` --- *refreshBeanFactory()*
    - 创建新的 `beanFactory` (DefaultListableBeanFactory)
    - 设置 `beanFactory` 的id(`obj.getClass().getName() + "@" + getIdentityHexString(obj)`)
      - 设置 `allowBeanDefinitionOverriding` 属性 --- *customizeBeanFactory()*
      - 设置 `allowCircularReferences` 属性
      - 创建beanDefinitionReader，准备读取xml文件 --- *loadBeanDefinitions()*
      - 设置beanDefinitionReader属性（environment、resourceLoader、entityResolver）
      - initBeanDefinitionReader(设置了reader.setValidating())
        - reader加载loadBeanDefinitions（通过：configResources/configLocations） --- *loadBeanDefinitions*
          1. 步骤：resources(类型string[]) → Resources[] → foreach处理每个Resources → 从Resource中获取Stream → 解析文件流 → 解析xml中dom元素 → 创建beanDefinition → 将BeanDefinition放入BeanFactory的BeanDefinitionMap(以及添加到beanDefinitionNames，从manualSingletonNames移除)
          2. 解析过程中，解析过的Resource会添加到一个Set集合中，保存在ThreadLocal中；
          3. 解析dom的默认实现是 `DefaultBeanDefinitionDocumentReader` ，子类可以通过覆盖 `preProcessXml()` 和 `postProcessXml()` 实现一些自定义操作；
  - 将创建的新的BeanFactory赋值给内部beanFactory，并返回到上一级，供后续方法使用；
  - 设置ClassLoader --- *prepareBeanFactory()*
  - 设置 `BeanExpressionResolver` （类型StandardBeanExpressionResolver）：此类包含SpelExpressionParser，解析 `#{...}` 形式的表达式；
  - 设置 `PropertyEditorRegistrar` (类型ResourceEditorRegistrar)
  - 添加 `ApplicationContextAwareProcessor` (此类会处理几种Aware的实现类，自行查看源码)
  - `ignoreDependencyInterface`
  - `registerResolvableDependency`
  - 设置 `ApplicationListenerDetector`
  - 设置 `LoadTimeWeaverAwareProcessor` 和 `tempClassLoader`
  - 设置 `environment` 、 `systemProperties` 、 `systemEnvironment` 实例；
  - 子类实现 `postProcessBeanFactory()` 方法（可以再BeanFactory实例化后，进行定制化操作） --- *postProcessBeanFactory()*
  - 执行 `BeanFactoryPostProcessors` --- *invokeBeanFactoryPostProcessors()*
    - 执行包括 `BeanDefinitionRegistryPostProcessor` 和其他 `BeanFactoryPostProcessor` (涉及到Ordered/PriorityOrdered)
  - 设置 `LoadTimeWeaverAwareProcessor` 和 `tempClassLoader` （如果之前没设置成功）
  - 注册 `BeanPostProcessor` (通过 `getBeanNamesForType` 获取bpp，然后将获取到的bpp添加到beanFactory中) --- *registerBeanPostProcessors()*
  - 初始化 `messageSource` ：如果有自定义的 `messageSource` ，进行其他处理（设置父 `messageSource` ），否则，创建 `messageSource` （类型DelegatingMessageSource），再设置父 `messageSource` --- *initMessageSource()*
  - 初始化事件广播器（如果用户没有自定义，默认创建SimpleApplicationEventMulticaster），赋值给

`this.applicationEventMulticaster` --- *initApplicationEventMulticaster()*

- `onFresh()`空模板方法，用户可自定义 --- *onRefresh()*
- 将`ApplicationListener`监听器注册给前面初始化的广播器 --- *registerListeners()*
- 广播`earlyApplicationEvents`中的事件
- 如果有`conversionService` bean，将conversionService赋值给BeanFactory.conversionService --- *finishBeanFactoryInitialization()*
- 如果BeanFactory没有`embeddedValueResolver`，创建一个默认的，类型为：StringValueResolver的实现
- 初始化`LoadTimeWeaverAware`
- 清掉`TempClassLoader`
  - 冻结BeanDefinition（设置`configurationFrozen`和`frozenBeanDefinitionNames`） --- *freezeConfiguration()*
  - 实例化单例Bean --- *preInstantiateSingletons()*
  - 调用所有`SmartInitializingSingleton`的回调方法