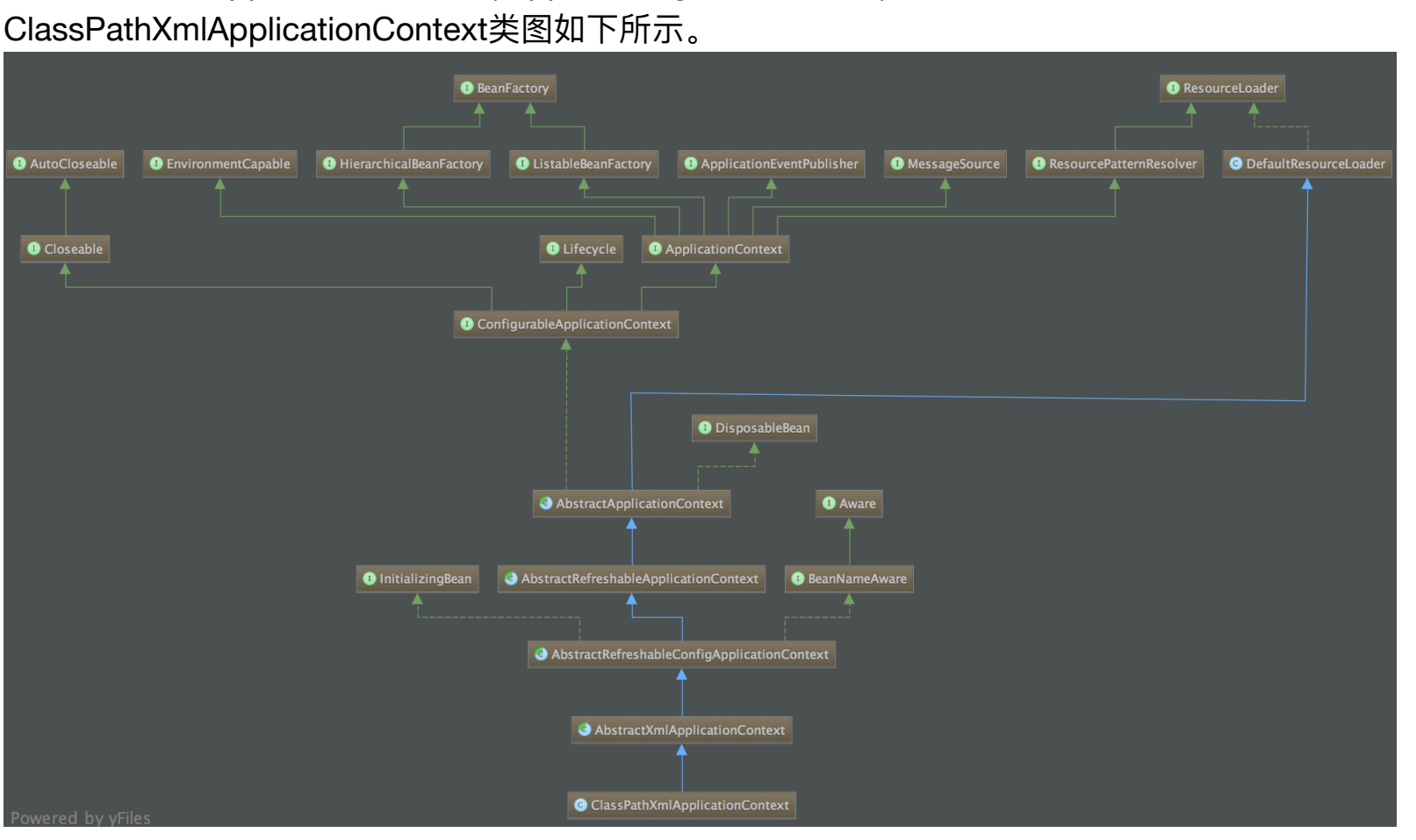
框架的源码分析，有些代码可以暂时忽略，如Spring如何进行XML模式校验的、XML解析的细节等，这些代码可以在了解了整体的原理之后，再做针对性的分析，关注重点内容即可，切记在一开始就去深挖每个细节，这样不仅会耗费很长时间，而且容易陷入某个坑里出不来。

以《Spring源码学习一：源码分析概述》示例中的ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationgContext.xml")为入口，进入源码内部，ClassPathXmlApplicationContext类图如下所示。



ClassPathXmlApplicationContext有多个构造方法，跟踪代码可以发现，最终使用的是下面这个方法，

```
public ClassPathXmlApplicationContext(String[] configLocations, boolean refresh,
ApplicationContext parent)
    throws BeansException {

  super(parent);
  setConfigLocations(configLocations);
  if (refresh) {
    refresh();
  }
 }
```

方法的参数很容易理解，configLocations指Spring的xml配置文件；refresh指是否需要刷新，这个refresh决定了是否进行bean解析、注册及实例化；parent指父ApplicationContext。setConfigLocations方法就是设置框架要加载的资源文件的位置。进入refresh方法，这个方法继承自AbstractApplicationContext，所以具体实现在AbstractApplicationContext中，代码如下。

```
public void refresh() throws BeansException, IllegalStateException {
  synchronized (this.startupShutdownMonitor) {
    //容器预先准备，记录容器启动时间和标记
    prepareRefresh();

    //创建bean工厂，里面实现了BeanDefinition的装载
    ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();

    //配置bean工厂的上下文信息，如类装载器等
    prepareBeanFactory(beanFactory);

    try {
      //在BeanDefinition被装载后，提供一个修改BeanFactory的入口
      postProcessBeanFactory(beanFactory);

      //在bean初始化之前，提供对BeanDefinition修改入口，PropertyPlaceholderConfigurer在这里被调用
      invokeBeanFactoryPostProcessors(beanFactory);

      //注册各种BeanPostProcessors，用于在bean被初始化时进行拦截，进行额外初始化操作
      registerBeanPostProcessors(beanFactory);

      //初始化MessageSource
      initMessageSource();

      //初始化上下文事件广播
      initApplicationEventMulticaster();

      //模板方法
      onRefresh();

      //注册监听器
      registerListeners();

      //初始化所有未初始化的非懒加载的单例Bean
      finishBeanFactoryInitialization(beanFactory);

      //发布事件通知
      finishRefresh();
    }catch (BeansException ex) {
      if (logger.isWarnEnabled()) {
        logger.warn("Exception encountered during context initialization - " +
            "cancelling refresh attempt: " + ex);
      }
      // Destroy already created singletons to avoid dangling resources.
      destroyBeans();
      // Reset 'active' flag.
      cancelRefresh(ex);
      // Propagate exception to caller.
      throw ex;
    }finally {
      // Reset common introspection caches in Spring's core, since we
      // might not ever need metadata for singleton beans anymore...
      resetCommonCaches();
    }
  }
 }
```

这个方法里面就是IOC容器初始化的大致步骤了。上面步骤的第二步完成了BeanDefinition的装载，进入obtainFreshBeanFactory方法，这个方法的具体实现也在AbstractApplicationContext类中，代码如下所示。

```
protected ConfigurableListableBeanFactory obtainFreshBeanFactory() {
  refreshBeanFactory();
  ConfigurableListableBeanFactory beanFactory = getBeanFactory();
  if (logger.isDebugEnabled()) {
    logger.debug("Bean factory for " + getDisplayName() + ": " + beanFactory);
  }
  return beanFactory;
}
```

这里面主要关注refreshBeanFactory方法，这个方法在AbstractApplicationContext类中并未实现，具体实现在子类AbstractRefreshableApplicationContext中，代码如下所示。

```
protected final void refreshBeanFactory() throws BeansException {
  if (hasBeanFactory()) {
    destroyBeans();
    closeBeanFactory();
  }
  try {
    DefaultListableBeanFactory beanFactory = createBeanFactory();
    beanFactory.setSerializationId(getId());
    customizeBeanFactory(beanFactory);
    loadBeanDefinitions(beanFactory);
    synchronized (this.beanFactoryMonitor) {
      this.beanFactory = beanFactory;
    }
  }
  catch (IOException ex) {
    throw new ApplicationContextException("I/O error parsing bean definition source for " +
getDisplayName(), ex);
  }
}
```

这个方法使用了final修饰，也就是不能被重写了。首先检查BeanFactory是否已经存在，如果存在则销毁并关闭，然后新建一个BeanFactory，其实就是一个DefaultListableBeanFactory，这个DefaultListableBeanFactory就是《Spring源码学习一：源码分析概述》写到的那个。然后进行BeanFactory的属性设置，设置是否允许重写BeanDefinition、是否允许循环引用，接着loadBeanDefinitions方法就是BeanDefinition载入的入口了，这个方法在AbstractRefreshableApplicationContext本类中并未实现，具体在其子类中实现，根据用途不同有多个实现子类，后续将分析基于最基本的解析xml方式的AbstractXmlApplicationContext类中的实现。