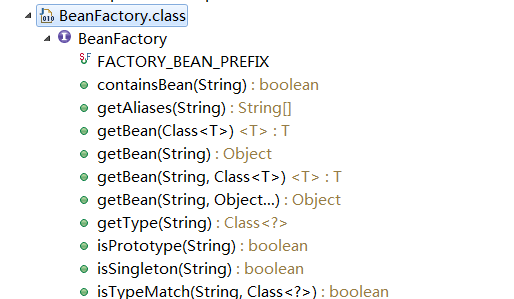
1. bean 的几个类型
2. 
3. Ioc容器的初始化过程

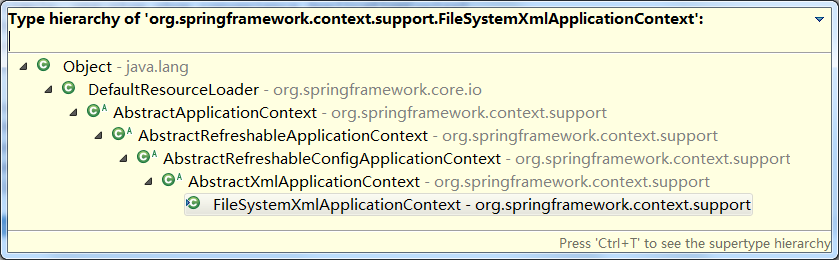
Resource定位的功能：

public class FileSystemXmlApplicationContext extends AbstractXmlApplicationContext

FileSystemXmlApplicationContext中都是构造方法，

public abstract class AbstractApplicationContext extends DefaultResourceLoader

关于IOC容器相关功能的实现，都是在FileSystemXmlApplicationContext中完成的，是在构造函数中的refresh来启动的，这个refresh方法是非常重要的方法。



代码可以说明一切：

几个片段：

FileSystemXmlApplicationContext中

**public** FileSystemXmlApplicationContext(String[] configLocations, **boolean** refresh, ApplicationContext parent)

**throws** BeansException {

**super**(parent);

setConfigLocations(configLocations);

**if** (refresh) {

refresh();

}

}

//1.用于文件系统中Resource的实现，通过构造一个FileSystemResource来得到一个在文件系统中定位的BeanDefinition

//这边的getResourceByPath在BeanDefinitionReader的LoadBeanDefiniton中被调用到的。

@Override

**protected** Resource getResourceByPath(String path) {

**if** (path != **null** && path.startsWith("/")) {

path = path.substring(1);

}

**return** **new** FileSystemResource(path);

}

AbstractApplicationContext中：

public void refresh() throws BeansException, IllegalStateException {

synchronized (this.startupShutdownMonitor) {

// Prepare this context for refreshing.

prepareRefresh();

// Tell the subclass to refresh the internal bean factory.

ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();//这边就有refreshBeanFactory()的调用。

// Prepare the bean factory for use in this context.

prepareBeanFactory(beanFactory);

try {

// Allows post-processing of the bean factory in context subclasses.

postProcessBeanFactory(beanFactory);

// Invoke factory processors registered as beans in the context.

invokeBeanFactoryPostProcessors(beanFactory);

// Register bean processors that intercept bean creation.

registerBeanPostProcessors(beanFactory);

// Initialize message source for this context.

initMessageSource();

// Initialize event multicaster for this context.

initApplicationEventMulticaster();

// Initialize other special beans in specific context subclasses.

onRefresh();

// Check for listener beans and register them.

registerListeners();

// Instantiate all remaining (non-lazy-init) singletons.

finishBeanFactoryInitialization(beanFactory);

// Last step: publish corresponding event.

finishRefresh();

}

catch (BeansException ex) {

// Destroy already created singletons to avoid dangling resources.

destroyBeans();

// Reset 'active' flag.

cancelRefresh(ex);

// Propagate exception to caller.

throw ex;

}

}

}

// AbstractRefreshableApplicationContext中：

**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;

}

}

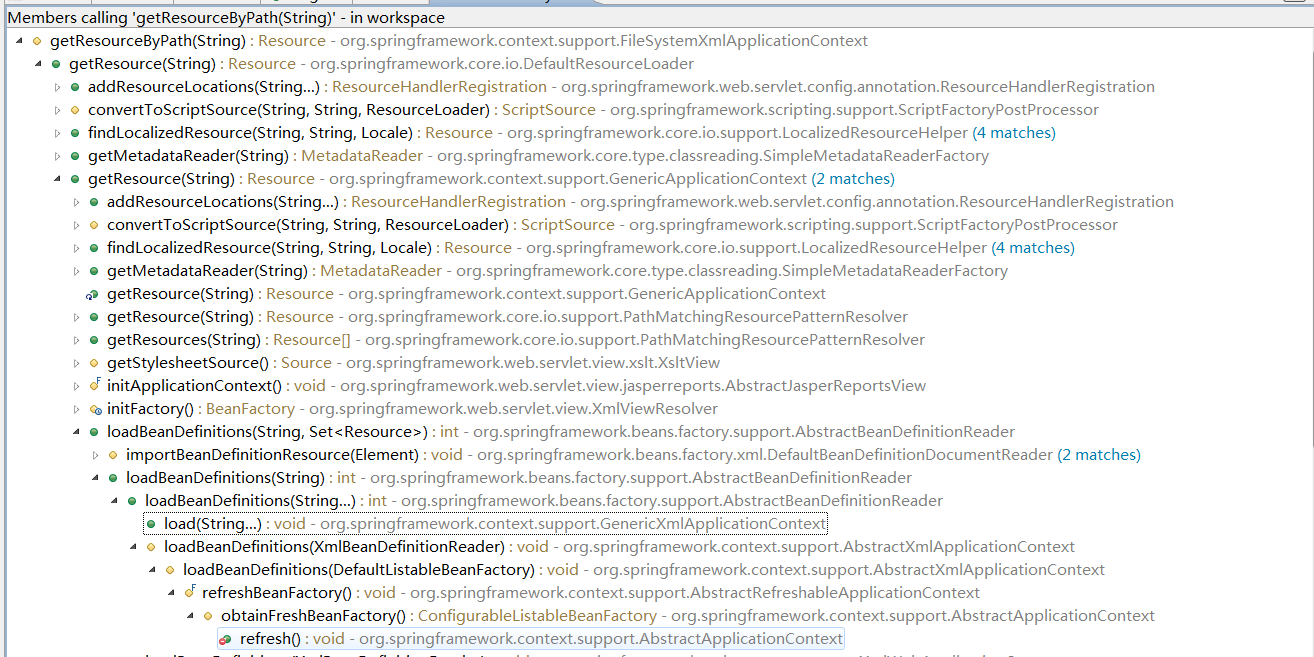
**catch** (IOException ex) {

**throw** **new** ApplicationContextException("I/O error parsing bean definition source for " + getDisplayName(), ex);

}

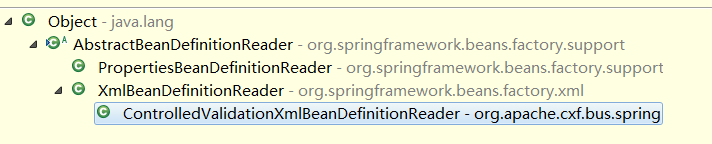
}

createBeanFactory创建了一个IOC容器供ApplicationContext使用，这个容器是DefaultListableBeanFactory,同时，它启动了loadBeanDefinitions来载入BeanDefinition。下面这张图是一张很关键的图，表示了refresh启动的过程，以及资源的加载。



在初始化FileSystmXmlApplicationContext的过程中，通过的方法就是refresh方法，使用的ioc容器是DefultListableBeaFactory.具体的资源的载入：

AbstractBeanDefinitionReader中的loadBeanDefinitions实现了资源的加载：可以看一下，继承结构



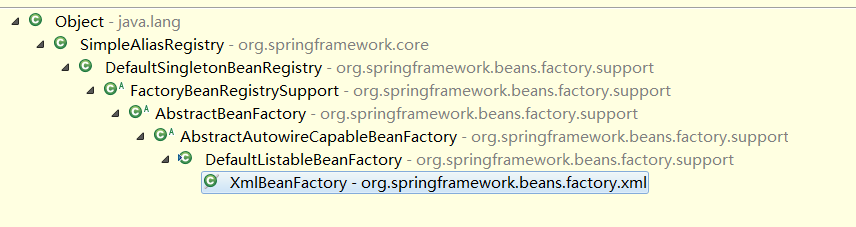
关于FileSystemXmlApplicationContext中的getResourceByPath方法，最后的时候生成的是一个FileSystemResource对象。如果是其他的ApplicationContext会生成其他类型的Resource.

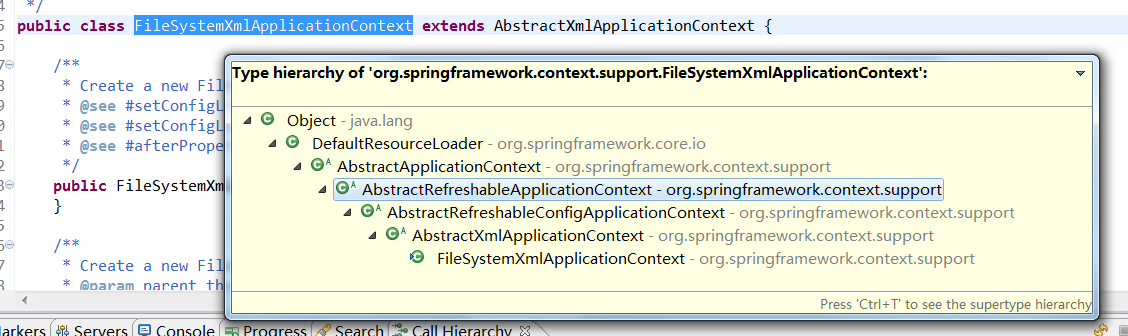
//到此为止都是资源的定位过程，接下来是数据的读入的过程，这就是：BeanDefinition的载入和解析的过程。

//BeanDefinition的载入和解析

载入的过程相当于把定义的BeanDefinition在IOC容器中转换成一个Spring内部表示的一个数据结构的过程。

看一下这个类: DefaultListableBeanFactory





Refresh方法的实现是在AbstractApplicationContext中。

在AbstractRefreshableApplicationContext中有一个方法: refreshBeanFactory();是来实例化一个BeanFactory的，在创建Ioc容器之前，如果已经有容器存在了，那就将已有的容器销毁和关闭掉。

//这里的loadBeanDefinitions中，初始化了读取器XmlBeanDefinitionReader，然后这个读取器在Ioc容器中设置好，最后是启动读取器来完成BeanDefinition在IOC容器中的载入。

AbstractXmlApplicationContext中的loadBeanDefinitions()作为实现。

**protected** **void** loadBeanDefinitions(DefaultListableBeanFactory beanFactory) **throws** BeansException, IOException {

// Create a new XmlBeanDefinitionReader for the given BeanFactory.

XmlBeanDefinitionReader beanDefinitionReader = **new** XmlBeanDefinitionReader(beanFactory);

// Configure the bean definition reader with this context's

// resource loading environment.

beanDefinitionReader.setEnvironment(**this**.getEnvironment());

beanDefinitionReader.setResourceLoader(**this**);

beanDefinitionReader.setEntityResolver(**new** ResourceEntityResolver(**this**));

// Allow a subclass to provide custom initialization of the reader,

// then proceed with actually loading the bean definitions.

initBeanDefinitionReader(beanDefinitionReader);

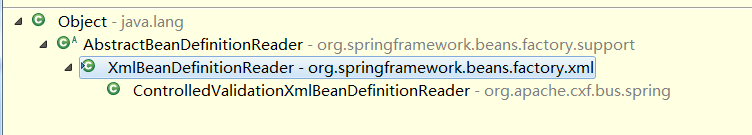
loadBeanDefinitions(beanDefinitionReader);

}

**protected** **void** initBeanDefinitionReader(XmlBeanDefinitionReader reader) {

reader.setValidating(**this**.validating);

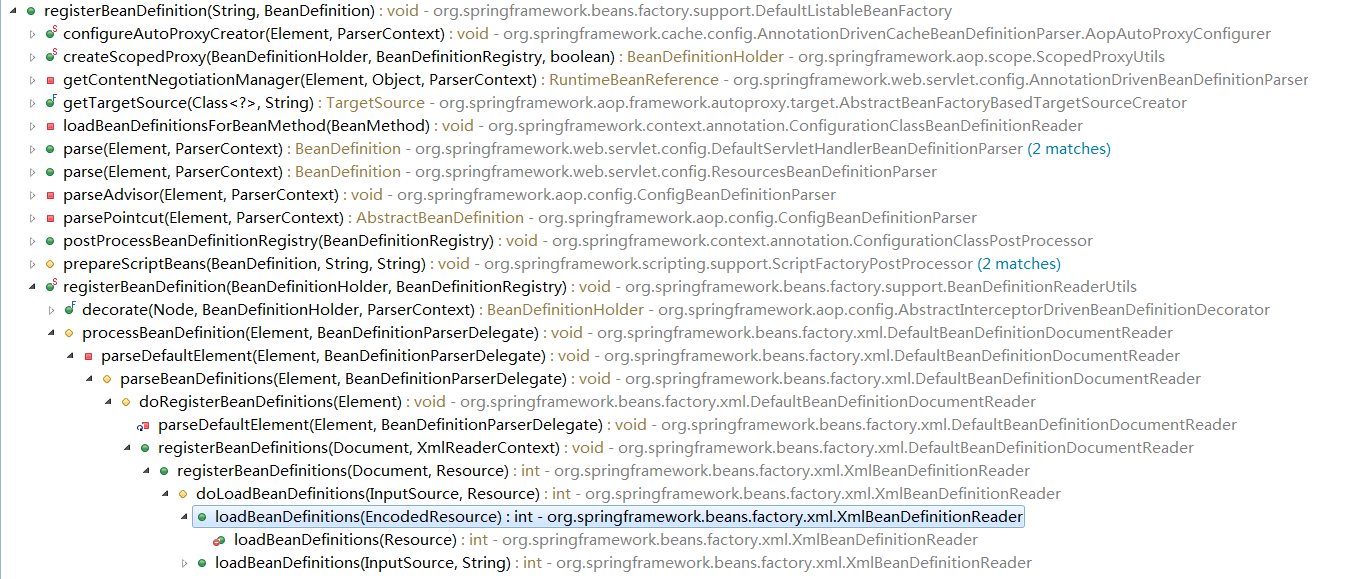
}



经过逐层的解析，我们在XML中定义的BeanDefinition就被整个的载入到了Ioc容器中去了，并在容器中建立了数据的映射。至此，Ioc容器大致完成了管理Bean对象的数据准备，接着需要完成的就是注册的过程

BeanDefinition在IOC容器中的注册

HashMap结构。



//图表示: registerBeanDefinition的调用关系。

**public** **void** registerBeanDefinition(String beanName, BeanDefinition beanDefinition)

**throws** BeanDefinitionStoreException {

Assert.*hasText*(beanName, "Bean name must not be empty");

Assert.*notNull*(beanDefinition, "BeanDefinition must not be null");

**if** (beanDefinition **instanceof** AbstractBeanDefinition) {

**try** {

((AbstractBeanDefinition) beanDefinition).validate();

}

**catch** (BeanDefinitionValidationException ex) {

**throw** **new** BeanDefinitionStoreException(beanDefinition.getResourceDescription(), beanName,

"Validation of bean definition failed", ex);

}

}

**synchronized** (**this**.beanDefinitionMap) {

Object oldBeanDefinition = **this**.beanDefinitionMap.get(beanName);

**if** (oldBeanDefinition != **null**) {

**if** (!**this**.allowBeanDefinitionOverriding) {

**throw** **new** BeanDefinitionStoreException(beanDefinition.getResourceDescription(), beanName,

"Cannot register bean definition [" + beanDefinition + "] for bean '" + beanName +

"': There is already [" + oldBeanDefinition + "] bound.");

}

**else** {

**if** (**this**.logger.isInfoEnabled()) {

**this**.logger.info("Overriding bean definition for bean '" + beanName +

"': replacing [" + oldBeanDefinition + "] with [" + beanDefinition + "]");

}

}

}

/\*\*\*

\*这是正常的注册的过程，beanDefinitionNames，beanDefinitionMap

\*

\*\*/

**else** {

**this**.beanDefinitionNames.add(beanName);

**this**.frozenBeanDefinitionNames = **null**;

}

**this**.beanDefinitionMap.put(beanName, beanDefinition);

}

resetBeanDefinition(beanName);

}

//至此注册完成

注意的是几个方法的调用