

1.使用ClassPathXmlApplicationContext通过xml文件来创建实例
ApplicationContext xc = new ClassPathXmlApplicationContext("aa.xml");

2.通过文件路径加载配置文件
setConfigLocations(configLocations);

3.方法入口 refresh
refresh();

4.refresh()方法内部包含13个方法

5.执行类的实例化和初始化操作
finishBeanFactoryInitialization(beanFactory)

6.初始化不是懒加载的单例
beanFactory.preInstantiateSingletons();

7.1 getBean(beanName)
7.2 getBean(beanName)

8.一级缓存查找失败
Object sharedInstance = getSingleton(beanName);

```
@Nullable  
protected Object getSingleton(String beanName, boolean allowEarlyReference) {  
    Object singletonObject = this.singletonObjects.get(beanName);  
    if (singletonObject == null && isSingletonCurrentlyInCreation(beanName)) {  
        synchronized (this.singletonObjects) {  
            singletonObject = this.singletonObjects.get(beanName);  
            if (singletonObject == null && allowEarlyReference) {  
                ObjectFactory<?> singletonFactory = this.singletonFactories.get(beanName);  
                if (singletonFactory != null) {  
                    singletonObject = singletonFactory.getObject();  
                    this.earlySingletonObjects.put(beanName, singletonObject);  
                    this.singletonFactories.remove(beanName);  
                }  
            }  
        }  
    }  
    return singletonObject; // singletonObject: null  
}
```

markBeanAsCreated(beanName);

public Object getSingleton(String beanName, ObjectFactory<?> singletonFactory)

singletonObject = singletonFactory.getObject();

createBean(String beanName, RootBeanDefinition mbd, @Nullable Object[] args)

Class<?> resolvedClass = resolveBeanClass(mbd, beanName); // 通过反射创建了对象实例

Object beanInstance = doCreateBean(beanName, mbdToUse, args);

addSingletonFactory(beanName, () -> getEarlyBeanReference(beanName, mbd, bean));

populateBean(beanName, mbd, instanceWrapper); // 填充bean

if (pvs != null) {
 applyPropertyValues(beanName, mbd, bw, pvs);
}

//这里去实例化InstantB
Object resolvedValue = valueResolver.resolveValueIfNecessary(pv, originalValue);

```
protected void addSingletonFactory(String beanName, ObjectFactory<?> singletonFactory) {  
    Assert.notNull(singletonFactory, "Singleton factory must not be null");  
    synchronized (this.singletonObjects) {  
        if (!this.singletonFactories.containsKey(beanName)) {  
            this.singletonFactories.put(beanName, singletonFactory);  
            this.earlySingletonObjects.remove(beanName);  
            this.registeredSingletons.add(beanName);  
        }  
    }  
}
```

```
else {  
    String propertyName = pv.getPropertyName();  
    Object originalValue = pv.getValue();  
    //这里去实例化InstantB  
    Object resolvedValue = valueResolver.resolveValueIfNecessary(pv, originalValue);  
    Object convertedValue = resolvedValue;  
    boolean convertible = bw.isWritableProperty(propertyName) && !PropertyAccessorUtils.isNestedOrIndexedProperty(propertyName);  
    if (convertible) {  
        convertedValue = convertForProperty(resolvedValue, propertyName, bw, converter);  
    }  
}
```

同理InstanceB去缓存中去找A是否存在

把A从三级缓存移除，放入二级缓存，并返回A对象的实例

中间经过一系列操作，开始初始化B

exposedObject = initializeBean(beanName, exposedObject, mbd);

直接把B从三级缓存扔进了一级缓存？

addSingleton(beanName, singletonObject);

```
beanInstance  
├── beanInstance = (InstanceA@1626)  
│   ├── b = (InstanceB@1737)  
│   │   ├── a = (InstanceA@1626)  
│   │   ├── b = (InstanceB@1737)  
│   │   ├── a = (InstanceA@1626)  
│   │   └── b = (InstanceB@1737)  
│   └── a = (InstanceA@1626)  
└── a = (InstanceA@1626)
```

把A从二级缓存移到一级缓存

addSingleton(beanName, singletonObject);

到这里我们A的实例化和初始化都完成了

B要开始实例化和初始化了，但是会发现，B已经在一级缓存中已经存在了，这时候，直接拿来用就好了

1.prepareRefresh()
2.ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory()
3.prepareBeanFactory(beanFactory)
4.postProcessBeanFactory(beanFactory)
5.invokeBeanFactoryPostProcessors(beanFactory)
6.registerBeanPostProcessors(beanFactory)
7.initMessageSource()
8.initApplicationEventMulticaster()
9.onRefresh()
10.registerListeners()
11.执行类的实例化和初始化操作
finishBeanFactoryInitialization(beanFactory)
12.finishRefresh()
13.resetCommonCaches()

for(1.循环beanNames逐个实例化和初始化)
if(){
 2.判断是否是抽象类/单利/懒加载
 3.判断是否继承FactoryBean
 } else {
 4.//这里开始实例化
 getBean(beanName);
 }
}

这里通过调用
singletonFactory.getObject()来
调用匿名内部类

```
logger.debug("Creating instance of bean '" + beanName + "'");  
RootBeanDefinition mbdToUse = mbd; mbdToUse.setRootBeanClass(resolveClass(mbd, beanName));  
// Make sure bean class is actually resolved at this point, and  
// allow the bean definition in case of a dynamically resolved class  
// which cannot be stored in the shared cached bean definition.  
Class<?> resolvedClass = resolveClass(mbd, beanName);  
if (resolvedClass != null) {  
    mbdToUse.setResolvedClass(resolvedClass);  
}
```

```
protected void addSingletonFactory(String beanName, ObjectFactory<?> singletonFactory) {  
    Assert.notNull(singletonFactory, "Singleton factory must not be null");  
    synchronized (this.singletonObjects) {  
        if (!this.singletonFactories.containsKey(beanName)) {  
            this.singletonFactories.put(beanName, singletonFactory);  
            this.earlySingletonObjects.remove(beanName);  
            this.registeredSingletons.add(beanName);  
        }  
    }  
}
```

此时发现在三级缓存中已经有a,b

和之前的A去找B相比差别就是，A对象现在处于创建中，
所以可以进入if条件语句中

```
@Nullable  
protected Object getSingleton(String beanName, boolean allowEarlyReference) {  
    Object singletonObject = this.singletonObjects.get(beanName);  
    if (singletonObject == null && isSingletonCurrentlyInCreation(beanName)) {  
        synchronized (this.singletonObjects) {  
            singletonObject = this.singletonObjects.get(beanName);  
            if (singletonObject == null && allowEarlyReference) {  
                ObjectFactory<?> singletonFactory = this.singletonFactories.get(beanName);  
                if (singletonFactory != null) {  
                    singletonObject = singletonFactory.getObject();  
                    this.earlySingletonObjects.put(beanName, singletonObject);  
                    this.singletonFactories.remove(beanName);  
                }  
            }  
        }  
    }  
    return singletonObject;  
}
```

```
@Nullable  
protected Object getSingleton(String beanName, boolean allowEarlyReference) {  
    Object singletonObject = this.singletonObjects.get(beanName);  
    if (singletonObject == null && isSingletonCurrentlyInCreation(beanName)) {  
        synchronized (this.singletonObjects) {  
            singletonObject = this.singletonObjects.get(beanName);  
            if (singletonObject == null && allowEarlyReference) {  
                ObjectFactory<?> singletonFactory = this.singletonFactories.get(beanName);  
                if (singletonFactory != null) {  
                    singletonObject = singletonFactory.getObject();  
                    this.earlySingletonObjects.put(beanName, singletonObject);  
                    this.singletonFactories.remove(beanName);  
                }  
            }  
        }  
    }  
    return singletonObject;  
}
```

```
current  
├── current = (InstanceB@1737)  
└── a = (InstanceA@1626)  
    └── b = null
```

创建InstantB

填充b的时候
发现需要实例化
InstantB

B已经在一级缓存中了，再来处理A