14_源码分析：如何根据 bean 名称或 bean 类型进行查找？

1、开篇

上节课介绍了 Spring IOC 依赖查找的定义，以及其包含的两种最为常用的查找 Bean 实例的方式：根据 Bean 名称查找和根据 Bean 类型查找。本节课，会进入源码分析的环节，聊聊如何根据 bean 名称或 bean 类型进行查找。有如下内容：

- 获取单个 Bean 类型实例
- 获取集合 Bean 类型实例
- 获取集合 Bean 类型名称

2、Bean 类型实例

由于 getBean(Class) 最终会调用 resolveNamedBean 方法完成 Bean 的查找，因此源码分析围绕 resolveNamedBean 展开。如图 1 ，2 所示，将 resolveNamedBean 方法放在两张图中，后面我们会针对图中的步骤逐一展开给大家解释。

```
private <T> NamedBeanHolder<T> resolveNamedBean(ResolvableType requiredType,
        Object[] args, boolean nonUniqueAsNull) throws BeansException {
    // 1. 根据类型查找，不会实例化 bean
    String[] candidateNames = getBeanNamesForType(requiredType);

    // 2. 多个类型，如何过滤?
    if (candidateNames.length > 1) {
        List<String> autowireCandidates = new ArrayList<>(candidateNames.length);
        for (String beanName : candidateNames) {
            // 如果容器中定义的beanDefinition.autowireCandidate=false（默认为true）则剔除
            // ①没有定义该beanDefinition或②beanDefinition.autowireCandidate=true时合法
            // 什么场景下会出现：没有定义该BeanDefinition，但根据类型可以查找到该beanName?
            if (!containsBeanDefinition(beanName) || getBeanDefinition(beanName).isAutowireCandidate()) {
                autowireCandidates.add(beanName);
            }
        }
        if (!autowireCandidates.isEmpty()) {
            candidateNames = StringUtils.toStringArray(autowireCandidates);
        }
    }
    // 3. 单个candidateNames，则调用getBean(beanName)实例化该bean
    if (candidateNames.length == 1) {
        String beanName = candidateNames[0];
        return new NamedBeanHolder<>(beanName, (T) getBean(beanName, requiredType.toClass(), args));
```

图 1 resolveNamedBean

```
// 4. 多个candidateNames，先尝试是否标注Primary属性，再尝试类上@Priority注解
} else if (candidateNames.length > 1) {
    Map<String, Object> candidates = new LinkedHashMap<>(candidateNames.length);
    for (String beanName : candidateNames) {
        if (containsSingleton(beanName) && args == null) {
            Object beanInstance = getBean(beanName);
            candidates.put(beanName, (beanInstance instanceof NullBean ? null : beanInstance));
        } else {
            candidates.put(beanName, getType(beanName));
        }
    }
    //4.a 查找 primary Bean，即 beanDefinition.primary=true
    String candidateName = determinePrimaryCandidate(candidates, requiredType.toClass());
    // 比较 Bean 的优先级。@javax.annotation.Priority
    if (candidateName == null) {
        candidateName = determineHighestPriorityCandidate(candidates, requiredType.toClass());
    }
    //4.b 过滤后只有一个符合条件，getBean(candidateName)实例化
    if (candidateName != null) {
        Object beanInstance = candidates.get(candidateName);
        if (beanInstance == null || beanInstance instanceof Class) {
            beanInstance = getBean(candidateName, requiredType.toClass(), args);
        }
        return new NamedBeanHolder<>(candidateName, (T) beanInstance);
    }
    //4.c 多个bean，抛出NoUniqueBeanDefinitionException异常
    if (!nonUniqueAsNull) {
        throw new NoUniqueBeanDefinitionException(requiredType, candidates.keySet());
    }
}

return null;
```

图 2 resolveNamedBean

我们根据 resolveNamedBean 代码注释中的步骤解释如下：

1. 根据 Bean 类型查找 Spring IoC 容器中所有符合条件的 Bean 名称，可能是一个或者多个。getBeanNamesForType 只会读取 BeanDefinition 信息，这里并没有对 Bean 进行实例化。
2. 查找完类型会将其放到 candidateNames 中，这里的过滤条件时没有定义 Bean 对应的 BeanDefinition 或者 BeanDefinition 的 autowireCandidate 为 true 的情况。如果满足上述情况就将其加入到 autowireCandidates 中。
3. 如果容器中只注册了一个这种类型的 Bean，也就是 candidateNames 长度为 1 的时候，直接实例化该 Bean 后返回即可。
4. 当 Spring IoC 容器中注册有多个 Bean 时需要完成如下几个步骤：
5. 查找 primary Bean，即 beanDefinition.primary=true。如果有多个，则抛出 NoUniqueBeanDefinitionException 异常。

6. 比较 Bean 的优先级。Spring 默认的比较器是 AnnotationAwareOrderComparator，比较 Bean 上 @javax.annotation.Priority 的优先级，值越小优先级越高。同样的，如果最高级别的多个，则抛出 NoUniqueBeanDefinitionException 异常。
7. 多个 Bean，则抛出 NoUniqueBeanDefinitionException 异常。

3、获取集合 Bean 类型实例

与获取单个 Bean 类型实例的过程相似的是，获取集合 Bean 类型实例在有多个 candidateNames 时不用过滤，全部返回即可。如图 3 所示， getBeansOfType 同样调用 getBeanNamesForType 获取所有类型匹配的 beanNames，然后调用 getBean(beanName) 实例化所有的 Bean。

```
@Override
public <T> Map<String, T> getBeansOfType(@Nullable Class<T> type) throws BeansException {
    return getBeansOfType(type, true, true);
}
@Override
public <T> Map<String, T> getBeansOfType(@Nullable Class<T> type,
      boolean includeNonSingletons, boolean allowEagerInit) throws BeansException {
    String[] beanNames = getBeanNamesForType(type, includeNonSingletons, allowEagerInit);
    Map<String, T> result = new LinkedHashMap<>(beanNames.length);
    for (String beanName : beanNames) {
        Object beanInstance = getBean(beanName);
        if (!(beanInstance instanceof NullBean)) {
            result.put(beanName, (T) beanInstance);
        }
    }
    return result;
}
```

图 3 getBeansOfType

4、获取集合 Bean 类型

如果需要获取集合 Bean 类型需要使用 getBeanNamesForType 方法，Spring 内部根据类型匹配所有的 beanNames。getBeanNamesForType 不会初始化 Bean，根据其 BeanDefinition 或 FactoryBean#getObjectType 获取其类型。

```java
@Override
public String[] getBeanNamesForType(@Nullable Class<?> type, boolean includeNonSingletons, boolean allowEagerInit) {
    // 1. 查询的结果不使用缓存
    // configurationFrozen表示是否冻结BeanDefinition，不允许修改，因此查询的结果可能有误
    // 一旦调用refresh方法，则configurationFrozen=true，也就是容器启动过程中会走if语句
    // type=null，查找所有Bean?
    // !allowEagerInit 表示不允许提前初始化FactoryBean，因此可能获取不到Bean的类型
    if (!isConfigurationFrozen() || type == null || !allowEagerInit) {
        return doGetBeanNamesForType(ResolvableType.forRawClass(type), includeNonSingletons, allowEagerInit);
    }
    // 2. 允许使用缓存，此时容器已经启动完成，bean已经加载，BeanDefinition不允许修改
    Map<Class<?>, String[]> cache =
        (includeNonSingletons ? this.allBeanNamesByType : this.singletonBeanNamesByType);
    String[] resolvedBeanNames = cache.get(type);
    if (resolvedBeanNames != null) {
        return resolvedBeanNames;
    }
    resolvedBeanNames = doGetBeanNamesForType(ResolvableType.forRawClass(type), includeNonSingletons, true);
    if (ClassUtils.isCacheSafe(type, getBeanClassLoader())) {
        cache.put(type, resolvedBeanNames);
    }
    return resolvedBeanNames;
}
```

图 4 getBeanNamesForType

如图 4 所示，方法 getBeanNamesForType 中调用 了 doGetBeanNamesForType 方法。

参数 type 表示要查找的类型，includeNonSingletons 表示是否包含非单例 Bean，allowEagerInit 表示是否提前部分初始化 FactoryBean。

1、查询结果不使用缓存，需要满足如下场景：

- configurationFrozen=false，表示 Spring 容器在初始化阶段，可以对 BeanDefinition 进行调整，当然缓存结果毫无意义。在 refresh 后会设置为 true，此时可以对结果进行缓存。
- type=null ？
- allowEagerInit=false，即不允许通过实例化 FactoryBean 来获取 Bean 类型，那就只能通过 FactoryBean 上的泛型来获取其类型了，也可能导致查询的结果不准确。

2、允许使用缓存，当 getBeanNamesForType 方法根据类型查找匹配的 beanNames 结果时，这个匹配过程并不十分准确。因为方法不会通过提前实例化 Bean 的方式获取其类型，只会根据 BeanDefinition 或 FactoryBean#getObjectType 获取其类型。如果不实例化对象，有些场景可能并

不能获取对象的类型。因此这里使用缓存，在容器已经启动完成以后 Bean 已经加载完毕，BeanDefinition 就不允许修改。

5、总结

本节课通过源码分析的方式介绍了：获取单个 Bean 类型实例；获取集合 Bean 类型实例；获取集合 Bean 类型名称。下节课，会带给大家依赖查找中的经典异常问题的讨论：Bean 找不到的情况以及 Bean 不是唯一的情况，看看在这种情况下应该如何处理。下期见，拜拜。