一、前言

Spring中在生命周期中有一个非常重要的阶段就是推断构造方法，Bean的生命周期中，不管是单例的对象还是原型的对象都有实例化这个阶段，spring在扫描的阶段，也就是上篇笔记中所提到的spring使用插件的机制去扫描配置类，最后解析成BeanDefinition，spring的扫描是通过Asm字节码技术完成了bean的扫描工作，在扫描完成过后，生成的BeanDefinition中的BeanclassName是一个普通类的全限定名，这个类没有加载到jvm中的，只是把符合条件的类的全限定名作为className放入了BeanDefinition的属性中，所以在实例化阶段就需要加载到jvm中，然后进行实例化，在实例化的阶段就需要进行构造方法的推导，spring推导bean的构造方法过程非常复杂，考虑到了很多情况，这篇笔记就主要记录下spring是如何推导构造方法的。

二、推断构造方法原理

Spring中在推断构造方法的时候，流程分为两大步：
第一步：通过spring提供的插件机制，也就是spring的bean后置处理器去决定构造方法的个数，决定构造方法的个数的逻辑为：

- 找到bean中的所有构造方法
- 循环每个构造方法，如果是实现了@Autowired的就加入到候选candidates集合中；
- 如果说找到了两个@Autowired的方法，那么就报错；
- 如果说构造方法中有@Autowired的，但是也找到了一个默认的无参数构造，那么也加入到candidates，但是这个时候只有@Autowired中的required为false才会加入，也就是说如果找到了一个默认无参数构造，一个@Autowired=true的构造，那么这个默认的无参数构造就会被舍弃，只有找到的@Autowired=false，那么又找到了一个无参数构造，才会加入到candidates中,也就是说如果找到了@Autowired=false的构造和一般的无参数构造一样，都加入到了后续列表中，spring把它当成了一个不确定的因素，需要后续继续推导；
- 如果说找到的构造方法只有一个，并且这个构造方法的参数是大于0的，那么就返回这个构造方法；
- 如果前三个都不能满足，也就说是bean中存在2个以上的构造方法，但是都没有加@Autowired注解的构造，那么直接返回null；

第二步是根据推导出的构造方法再去推导最终的构造方法，然后去实例化，在做换个过程中很很多推导条件：
首先开发者是否传入了参数，如果传入了参数，就算前面一步推导出的构造方法为空，那么有了参数，那么可以直接实例化；
如果开发者没有传入参数，那么这个时候就要再次推导构造方法，如果这个时候的构造方法只有一个，那么你在BeanDefinition中也没有定义参数，那么就可以直接通过无参数构造方法进行实例化；如果有参数，并且存在多个构造方法，并且没有默认无参的构造方法，就会报错。

当构造方法大于1个的时候，如果不加@Autowired来决定使用的构造方法，那么你没有无参数的构造，那么spring最后无法决定你采用的构造方法，会采用默认的构造方法，但是你没有写默认的构造方法，那么这个时候就会报错，比如：

```
@Component
public class UserService {
   public UserService(User user){
      System.out.println("one");
   }
   public UserService(User user,User user1){
      System.out.println("two");
   }
}
```

如果这样修改代码，是可以的：

```
public UserService(){
   System.out.println("non");
}
public UserService(User user,User user1){
   System.out.println("two");
}
```

再者如果多个构造方法，开发者通过getBean传入参数并且是原型的也是可以的：

```
@Component
@Scope("prototype")
public class UserService {
   public UserService(User user){
      System.out.println("one");
   }
   public UserService(User user,User user1){
      System.out.println("two");
   }
}
public static void main(String[] args) {

   AnnotationConfigApplicationContext ac = new AnnotationConfigApplicationContext();
   ac.register(AppConfig.class);
   ac.refresh();
   System.out.println(ac.getBean(UserService.class, new User()));


}
```

那么这个时候会去执行一个参数的构造方法，但是需要你的bean的作用域是原型的，因为单例的在容器启动就会实例化推导构造方法，这个时候是无法确定的，只有原型的或者懒加载的时候每次getBean去获取的时候才行。

还有中就是bean中的构造方法只有一个，这个时候如果构造方法的参数也大于0也是可以的，spring也是可以推导出来，因为只有一个，肯定得用这个，如果有两个，无法确定，spring就会去尝试使用无参数的构造，最终导致的就是如果你没有无参数构造就会报错。

当一个bean中的构造方法有1个以上，但是必须要包含一个无参数的构造，因为spring是这样规定的，如果一个bean中的构造方法有1个以上，但是这些构造都没有使用@Autowired来决定使用哪个哪个构造，那么第一步的推断构造方法返回的就是null，这个时候就要重新去推导，根据是否输入参数，参数个数，构造的个数和一些权重来决定了，但是必须要有一个无参数的构造，否则就要报错。

三、@Autowired推导构造方法

@Autowired在spring的属性依赖注入的时候使用的比较多，但是在一个bean中的多个构造方法也可以通过@Autowred去决定使用哪个构造方法，但是要注意的时候，如果一个bean中的构造方法使用了@Autowired修饰了，那么这个bean的其他构造方法不能再加@Autowired注解了，因为既然使用@Autowired来决定使用的构造方法，那么你多个构造方法都用@Autowired修饰，那么你是几个意思？就算@Autowired的required为false也不行。如果@Autowired都设置为false，也是可以的，因为对于spring来说，在构造方法中@Autowired如果设置

required为false，那么它只能是候选者中的一个，不是决定性因素，所以在一个bean中的构造方法，只能存在一个@Autowired，属性required为true的一个构造方法，存在了一个@Autowired，属性为required=true的构造方法，就不能再有其他的@Autowired，即使其他的required为false也不行。这是spring的设计者规定的。

四、Xml中构造方法推导

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans-2.0.xsd
            http://www.springframework.org/schema/aop
https://www.springframework.org/schema/aop/spring-aop-2.5.xsd
            http://www.springframework.org/schema/context
            https://www.springframework.org/schema/context/spring-context-2.5.xsd">


   <bean name="a2" class="com.xxx.service.UserService" autowire="constructor" >

   </bean>
</beans>
```

spring中xml配置文件中的注入方式为构造方法注入，这个时候spring的选择就不一样了， spring会选择一个最多参数的构造方法作为注入点，这个在源码里面已经体现了。

```
public class UserService {


   public UserService(){
      System.out.println("no");
   }


   public UserService(User user){
      System.out.println("one");
   }
   public UserService(User user,User user1){
      System.out.println("two");
   }
}
```

还有下面这种方式：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:aop="http://www.springframework.org/schema/aop"
      xmlns:context="http://www.springframework.org/schema/context"
```

```
      xsi:schemaLocation="http://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans-2.0.xsd
            http://www.springframework.org/schema/aop
https://www.springframework.org/schema/aop/spring-aop-2.5.xsd
            http://www.springframework.org/schema/context
            https://www.springframework.org/schema/context/spring-context-2.5.xsd">


   <bean name="a2" class="com.xxx.service.UserService" >
       <constructor-arg index="0" ref="user" />
   </bean>
</beans>
```

那么它就回去找到指定的构造方法进行注入，注入的模式多种多样，推导的范式也多种多样，这个不要去记，只要理解它的思想和理念就可以了。比如不加@Autowired的时候怎么卖推导的，加了@Autowired时候怎么推导的就可以了，不用去死记硬背。

五、@ConstructorProperties

在spring中的构造方法注入的情况中，spring会获取构造方法的参数去容器中找bean，和之前创建bean的过程一样，先byType，在byname,这样的话效率上有一定的影响，spring提供了一种新的方式可以让你手动指定参数中对应的beanname,然后根据beanName去获取bean，这样的方式就相当于去容器中getBean一样，性能非常高，但是这种方式spring在没有找到的情况下也是去byType和byName去获取，所以如果你要用这个注解，那么就要把bean的名字对应好，否则用了也白用，spring找不到对应的bean还是会走之前的逻辑，具体用法如下：

```
@Component
public class UserService {

   public UserService(){
      System.out.println("0");
   }

   @ConstructorProperties({"user","person"})
   @Autowired
   public UserService(User user,Person person){
      System.out.println("2");
   }
}
```

上面的用法如果说spirng在推断构造方法推到出来的构造方法是加了 @ConstructorProperties注解的，那么就很简单了，就直接获取 @ConstructorProperties的名字数组，然后依次去容器中getBean得到参数注入的对象，但是一定要写对，否则还是走以前的逻辑，这个参数就毫无意义了。