【有道云笔记】01-Spring底层核心原理解析：https://note.youdao.com/s/9XCU79lG

本节课会把Spring中核心知识点都给大家进行串讲，让大家对Spring的底层有一个整体的大致了解，比如：

1. Bean的生命周期底层原理
2. 依赖注入底层原理
3. 初始化底层原理
4. 推断构造方法底层原理
5. AOP底层原理
6. Spring事务底层原理

但都只是大致流程，**后续会针对每个流程详细深入的讲解并分析源码实现。**

先来看看入门使用Spring的代码：

```
ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("spring.xml");
UserService userService = (UserService) context.getBean("userService");
userService.test();
```

对于这三行代码应该，大部分同学应该都是比较熟悉，这是学习Spring的hello world。可是，这三行代码底层都做了什么，比如：

1. 第一行代码，会构造一个ClassPathXmlApplicationContext对象，ClassPathXmlApplicationContext该如何理解，调用该构造方法除开会实例化得到一个对象，还会做哪些事情?
2. 第二行代码，会调用ClassPathXmlApplicationContext的getBean方法，会得到一个UserService对象，getBean()是如何实现的？返回的UserService对象和我们自己直接new的UserService对象有区别吗?
3. 第三行代码，就是简单的调用UserService的test()方法，不难理解。

光看这三行代码，其实**并不能体现出来Spring的强大之处**，也不能理解为什么需要ClassPathXmlApplicationContext和getBean()方法，随着课程的深入将会改变你此时的观念，而对于上面的这些疑问，也会随着课程深入逐步得到解决。对于这三行代码，你现在可以认为：如果你要用Spring，你就得这么写。就像你要用Mybatis，你就得写各种Mapper接口。

但是用ClassPathXmlApplicationContext其实已经过时了，在新版的Spring MVC和Spring Boot的底层主要用的都是**AnnotationConfigApplicationContext**，比如：

```
AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.clas
s);
//ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("spring.xml");
UserService userService = (UserService) context.getBean("userService");
userService.test();
```

可以看到AnnotationConfigApplicationContext的用法和ClassPathXmlApplicationContext是非常类似的，只不过需要传入的是一个class，而不是一个xml文件。

而AppConfig.class和spring.xml一样，表示Spring的配置，比如可以指定扫描路径，可以直接定义Bean，比如：

spring.xml中的内容为：

```
<context:component-scan base-package="com.zhouyu"/>
<bean id="userService" class="com.zhouyu.service.UserService"/>
```

AppConfig中的内容为：

```
@ComponentScan("com.zhouyu")
public class AppConfig {

 @Bean
 public UserService userService(){
  return new UserService();
 }

}
```

所以spring.xml和AppConfig.class本质上是一样的。

目前，我们基本很少直接使用上面这种方式来用Spring，而是使用Spring MVC，或者Spring Boot，但是它们都是基于上面这种方式的，都需要在内部去创建一个ApplicationContext的，只不过：

1. Spring MVC创建的是**XmlWebApplicationContext**，和**ClassPathXmlApplicationContext**类似，都是基于XML配置的
2. Spring Boot创建的是**AnnotationConfigApplicationContext**

因为AnnotationConfigApplicationContext是比较重要的，并且AnnotationConfigApplicationContext和ClassPathXmlApplicationContext大部分底层都是共同的，后续课程我们会着重将AnnotationConfigApplicationContext的底层实现，对于ClassPathXmlApplicationContext，同学们可以在课程结束后作为作业，业余时间看看相关源码即可。

Spring中是如何创建一个对象?

其实不管是AnnotationConfigApplicationContext还是ClassPathXmlApplicationContext，目前，我们都可以简单的将它们理解为就是用来创建Java对象的，比如调用getBean()就会去创建对象（此处不严谨，getBean可能也不会去创建对象，后续课程详解）。

在Java语言中，肯定是根据某个类来创建一个对象的。我们在看一下实例代码：

```
AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.clas
s);
UserService userService = (UserService) context.getBean("userService");
userService.test();
```

当我们调用context.getBean("userService")时，就会去创建一个对象，但是getBean方法内部怎么知道"userService"对应的是UserService类呢?

所以，我们就可以分析出来，在调用AnnotationConfigApplicationContext的构造方法时，也就是第一行代码，会去做一些事情：

1. 解析AppConfig.class，得到扫描路径
2. 遍历扫描路径下的所有Java类，如果发现某个类上存在@Component、@Service等注解，那么Spring就把这个类记录下来，存在一个Map中，比如Map<String, Class>。（**实际上，Spring源码中确实存在类似的这么一个Map，叫做BeanDefinitionMap，后续课程会讲到**）
3. Spring会根据某个规则生成当前类对应的beanName，作为key存入Map，当前类作为value

这样，但调用context.getBean("userService")时，就可以根据"userService"找到UserService类，从而就可以去创建对象了。

Bean的创建过程

那么Spring到底是如何来创建一个Bean的呢，这个就是Bean创建的生命周期，大致过程如下

1. 利用该类的构造方法来实例化得到一个对象（但是如何一个类中有多个构造方法，Spring则会进行选择，这个叫做**推断构造方法**）
2. 得到一个对象后，Spring会判断该对象中是否存在被@Autowired注解了的属性，把这些属性找出来并由Spring进行赋值（**依赖注入**）
3. 依赖注入后，Spring会判断该对象是否实现了BeanNameAware接口、BeanClassLoaderAware接口、BeanFactoryAware接口，如果实现了，就表示当前对象必须实现该接口中所定义的setBeanName()、setBeanClassLoader()、setBeanFactory()方法，那Spring就会调用这些方法并传入相应的参数（**Aware回调**）
4. Aware回调后，Spring会判断该对象中是否存在某个方法被@PostConstruct注解了，如果存在，Spring会调用当前对象的此方法（**初始化前**）
5. 紧接着，Spring会判断该对象是否实现了InitializingBean接口，如果实现了，就表示当前对象必须实现该接口中的afterPropertiesSet()方法，那Spring就会调用当前对象中的afterPropertiesSet()方法（**初始化**）
6. 最后，Spring会判断当前对象需不需要进行AOP，如果不需要那么Bean就创建完了，如果需要进行AOP，则会进行动态代理并生成一个代理对象做为Bean（**初始化后**）

通过最后一步，我们可以发现，当Spring根据UserService类来创建一个Bean时：

1. 如果不用进行AOP，那么Bean就是UserService类的构造方法所得到的对象。
2. 如果需要进行AOP，那么Bean就是UserService的代理类所实例化得到的对象，而不是UserService本身所得到的对象。

Bean对象创建出来后：

1. 如果当前Bean是单例Bean，那么会把该Bean对象存入一个Map<String, Object>，Map的key为beanName，value为Bean对象。这样下次getBean时就可以直接从Map中拿到对应的Bean对象了。（实际上，在Spring源码中，这个Map就是**单例池**）
2. 如果当前Bean是原型Bean，那么后续没有其他动作，不会存入一个Map，下次getBean时会再次执行上述创建过程，得到一个新的Bean对象。

推断构造方法

Spring在基于某个类生成Bean的过程中，需要利用该类的构造方法来实例化得到一个对象，但是**如果一个类存在多个构造方法，Spring会使用哪个呢?**

Spring的判断逻辑如下：

1. 如果一个类只存在一个构造方法，不管该构造方法是无参构造方法，还是有参构造方法，Spring都会用这个构造方法
2. 如果一个类存在多个构造方法
    i. 这些构造方法中，存在一个无参的构造方法，那么Spring就会用这个无参的构造方法
    ii. 这些构造方法中，不存在一个无参的构造方法，那么Spring就会**报错**

Spring的设计思想是这样的：

1. 如果一个类只有一个构造方法，那么没得选择，只能用这个构造方法
2. 如果一个类存在多个构造方法，Spring不知道如何选择，就会看是否有无参的构造方法，因为无参构造方法本身表示了一种默认的意义
3. 不过如果某个构造方法上加了@Autowired注解，那就表示程序员告诉Spring就用这个加了注解的方法，那Spring就会用这个加了@Autowired注解构造方法了

需要重视的是，如果Spring选择了一个有参的构造方法，Spring在调用这个有参构造方法时，需要传入参数，那这个参数是怎么来的呢？

Spring会根据入参的类型和入参的名字去Spring中找Bean对象（以单例Bean为例，Spring会从单例池那个Map中去找）：

1. 先根据入参类型找，如果只找到一个，那就直接用来作为入参
2. 如果根据类型找到多个，则再根据入参名字来确定唯一一个
3. 最终如果没有找到，则会报错，无法创建当前Bean对象

确定用哪个构造方法，确定入参的Bean对象，这个过程就叫做**推断构造方法**。

AOP大致流程

AOP就是进行动态代理，在创建一个Bean的过程中，Spring在最后一步会去判断当前正在创建的这个Bean是不是需要进行AOP，如果需要则会进行动态代理。

如何判断当前Bean对象需不需要进行AOP:

1. 找出所有的切面Bean
2. 遍历切面中的每个方法，看是否写了@Before、@After等注解
3. 如果写了，则判断所对应的Pointcut是否和当前Bean对象的类是否匹配
4. 如果匹配则表示当前Bean对象有匹配的的Pointcut，表示需要进行AOP

利用cglib进行AOP的大致流程：

1. 生成代理类UserServiceProxy，代理类继承UserService
2. 代理类中重写了父类的方法，比如UserService中的test()方法
3. 代理类中还会有一个target属性，该属性的值为被代理对象（也就是通过UserService类推断构造方法实例化出来的对象，进行了依赖注入、初始化等步骤的对象）
4. 代理类中的test()方法被执行时的逻辑如下：
    i. 执行切面逻辑（@Before）
    ii. 调用target.test()

当我们从Spring容器得到UserService的Bean对象时，拿到的就是UserServiceProxy所生成的对象，也就是代理对象。

UserService代理对象.test()--->执行切面逻辑--->target.test()，注意target对象不是代理对象，而是被代理对象。

Spring事务

当我们在某个方法上加了@Transactional注解后，就表示该方法在调用时会开启Spring事务，而这个方法所在的类所对应的Bean对象会是该类的代理对象。

Spring事务的代理对象执行某个方法时的步骤：

1. 判断当前执行的方法是否存在@Transactional注解
2. 如果存在，则利用事务管理器（TransactionMananger）新建一个数据库连接
3. 修改数据库连接的autocommit为false
4. 执行target.test()，执行程序员所写的业务逻辑代码，也就是执行sql
5. 执行完了之后如果没有出现异常，则提交，否则回滚

Spring事务是否会失效的判断标准：**某个加了@Transactional注解的方法被调用时，要判断到底是不是直接被代理对象调用的，如果是则事务会生效，如果不是则失效。**