

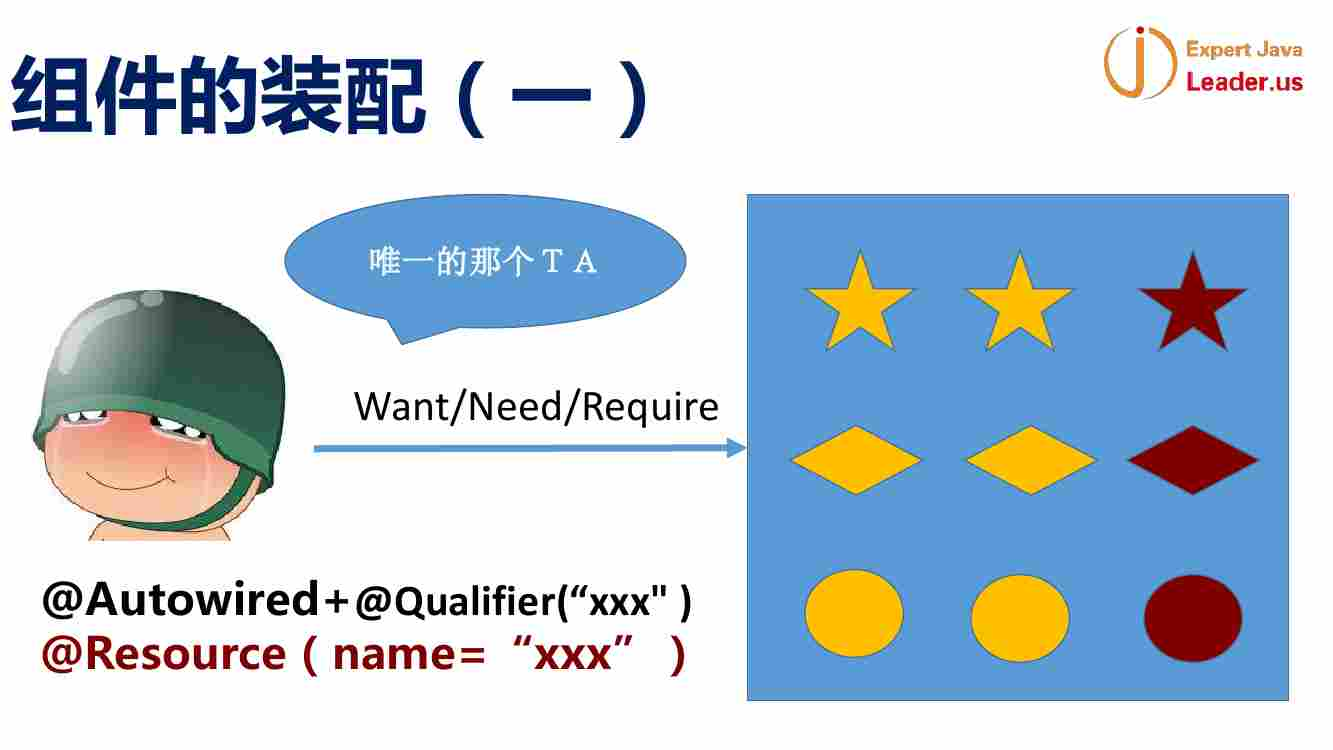
讲讲组件怎么装配 基本是最复杂的一部分



第一种 AutoWired　＋ Resource

另一个难一点的是AOP拦截注入

最后一个是Spring新增的泛型注入



里面有很多组件 有的有名称 这种情况产生两种装配的方式

(1). 按照类型 --- 由某一个接口或者类派生出来的

(2). 按照名字 --- 使用JavaBean的名字进行装配

按类型 ---- 多个类型的时候 spring中 autowired + qualified ---- 所以定义组件的时候 给组件一个名字

另外就是根据Java自身的 @Resource注解 ---- 默认是按照名称进行注解 Java中认为按照名称是最重要的

Java这个是规范的 简单



Spring对待@AutoWired和Resource是有差别的

我们看 在 Configuration中增加了一个 叫 @Bean HelloWorldBean anotherHelloBean(){return new HelloWorldBean();} 【可以学习@Bean 这个可以修饰方法】

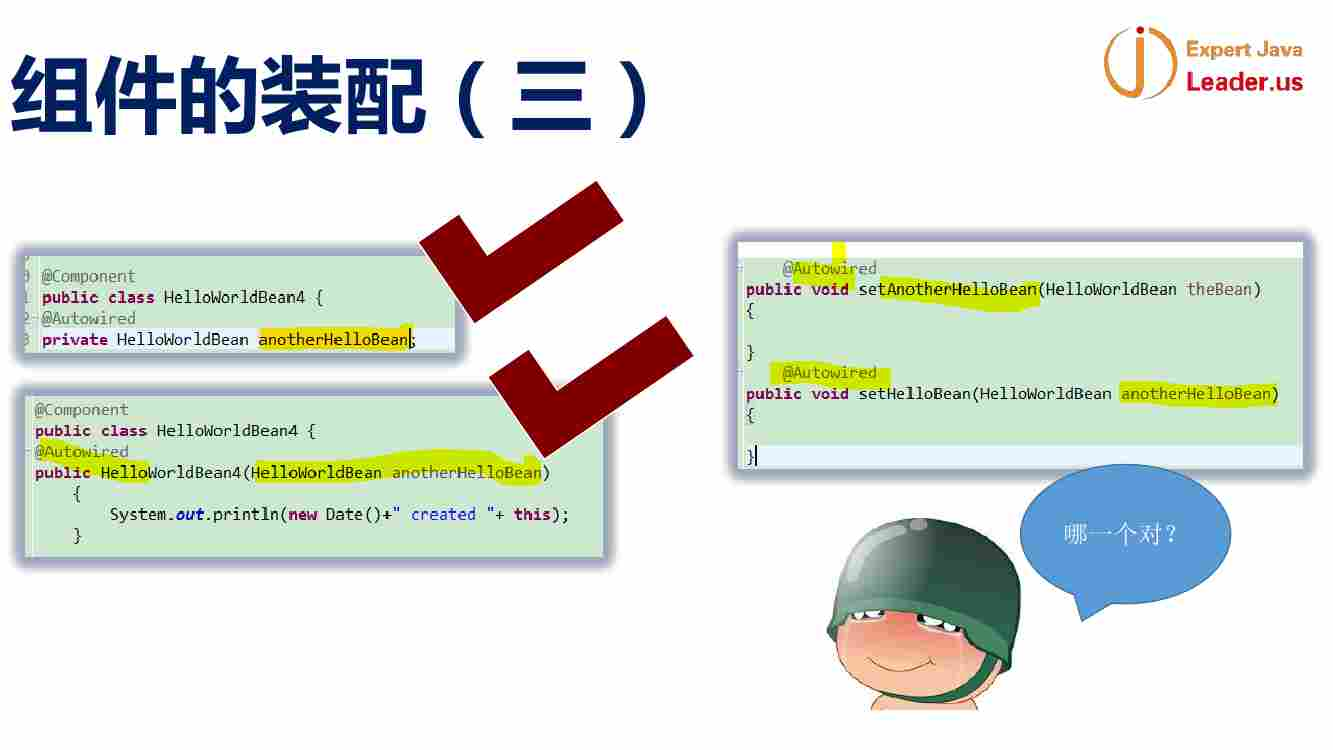
还有就是 有一个@Component(“helloBean”) public class HelloWorldBean{} ---- -这样整个Context中 有两个HelloWorldBean这种类型的Bean

这样 我们的的public class HelloWorldBean4中 仅仅使用了@Autowired这个进行注解

----- 一个叫helloBean 另一个叫anotherHelloBean ----- 这样出现的结果 是报错 存在两个

名字不一样 这样 他就不知道该用什么了

这时候 我们的解决办法就是使用@Qualified指定名称【这样 就说明了@Autowired默认是 依据类型进行装配】



有两个相同名字的bean的时候 就是刚才配置的情况 有一个有趣的现象

就是 我的要装配的bean的属性的属性名称或者构造函数的参数名 是其中的某一个Bean的名称 也有可能通过

还有就是setter注入的时候 setXxxx –这个Xxxx ------ 这个要求练习的时候 名称变化的时候 该怎么？【需要自己去对Autowired深刻理解】【我认为应该看源码】

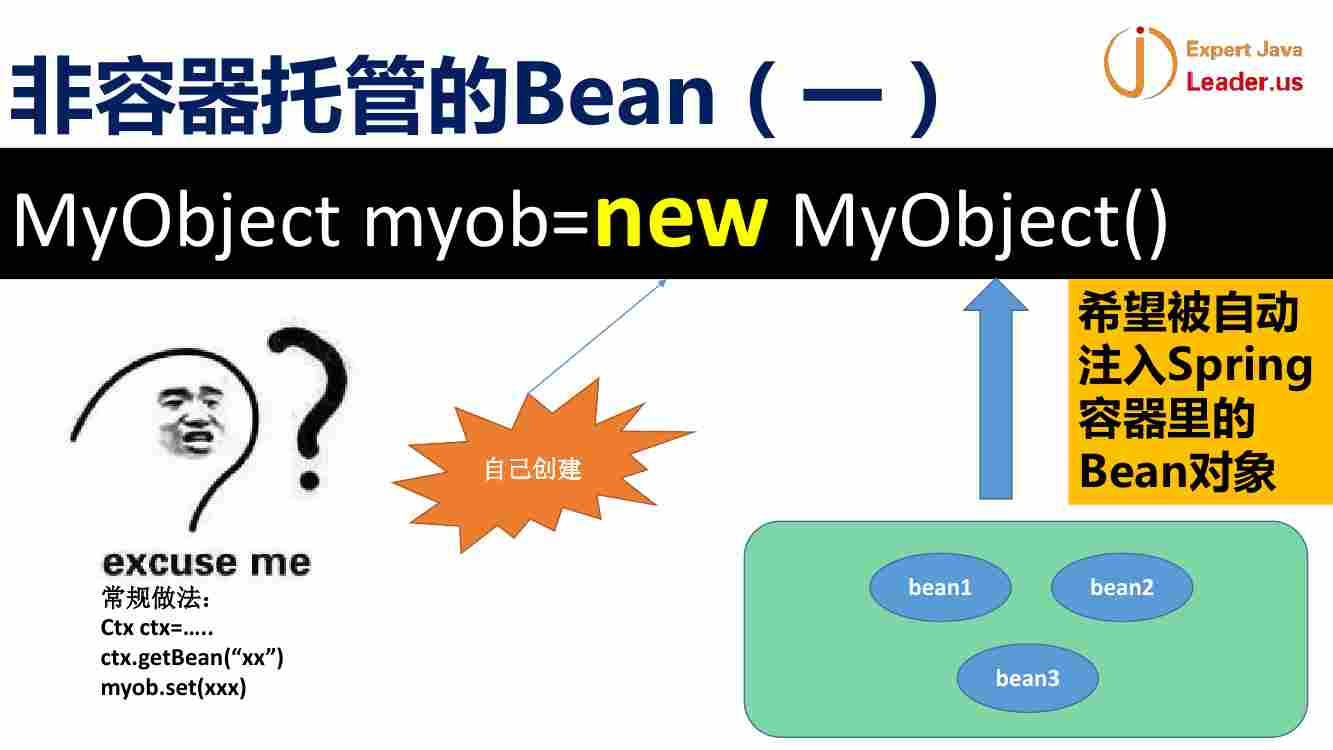


Resource是默认byName注入 ---name属性 如果使用了type就是byType指定

字段上 方法上都可以

大多数情况下 Resource替代Autowried更安全

系统比较复杂的时候 按照名称注入加方便



这个对象是自己new出来的 但是 这个对象的属性可以是Spring自己装配的 ------ 这个是spring的一个新特性

常规做法： 上面的截图部分

用法就是 我们的有些bean 要包装一些数据连接 或者事务相关的 可以加进来 ---- 场景用的不多



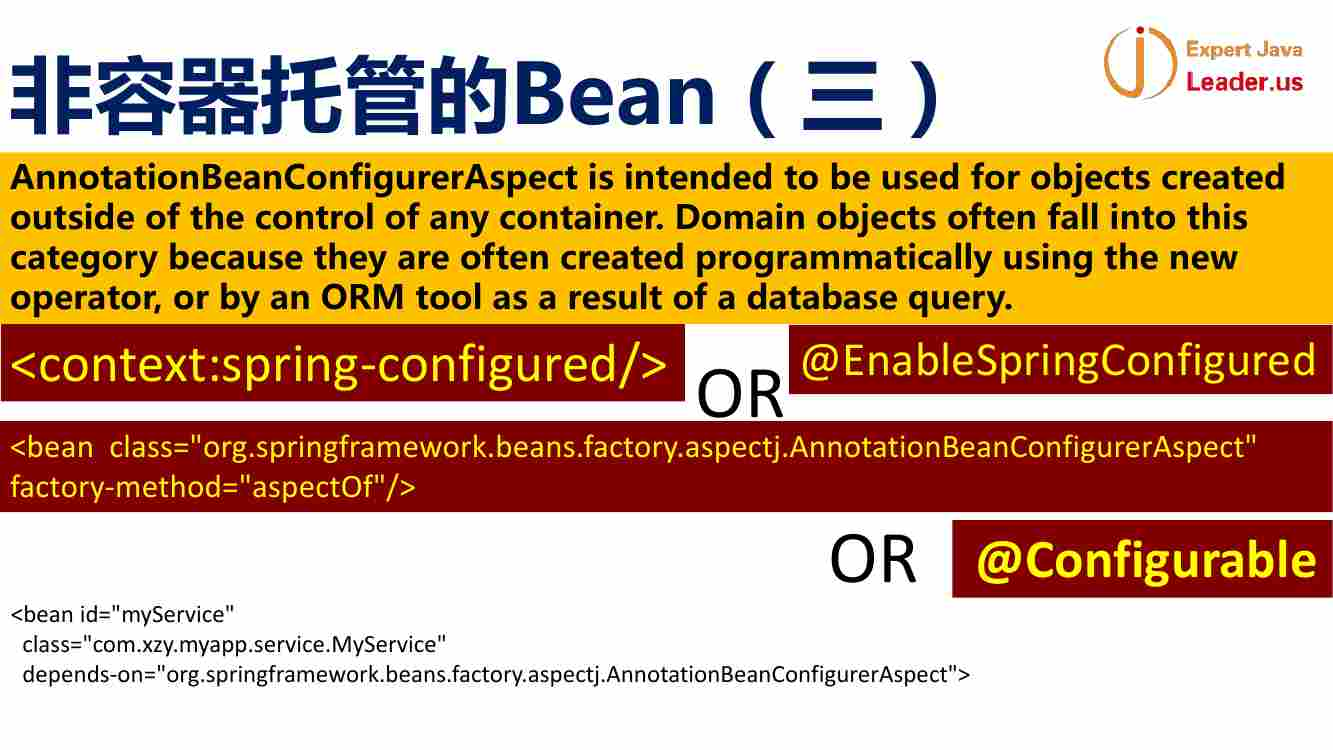
这个是AOP做的 但是不是Spring本身的AOP做的！！！是AspectJ的拦截做的

发现有newXxx()的时候 进行拦截 织入相应的代码 ---Spring 3.0产生的



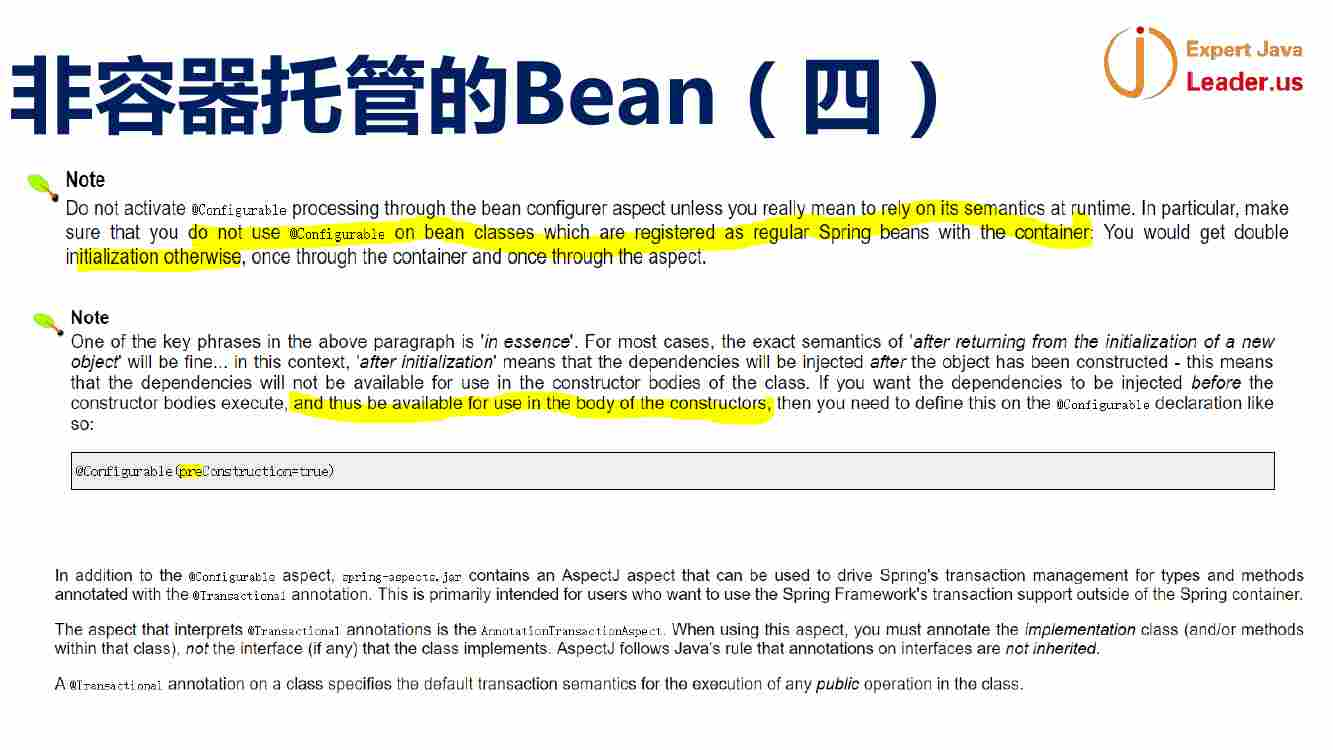
AspectJ拦截了这个对象的创建 并加了Spring框架的代码 ------ 这个注解是@Configurable 而不是@Configuration 这两个注解非常容易弄混淆

@Configurable是负责让Spring**之外的对象**进行装配的



看一下 这个xml定义的话 就是 <context:spring-configured> 对应的注解@EnableSpringConfigured 来开启

这个是xml方式定义的【感觉这个地方老师没有说清楚 貌似PPT上面介绍的是AnnotationBeanConfigurerAspect这个类的作用 和 哪些情况出现了Container管理之外的Bean 一个是new出来的 另一个ORM框架通过一次查询给出来的】



【分析一下PPT中每一段的意思

\*\*注意

除非你真的想在运行期要依赖@Configurable这个注解语义 否则 你不要通过bean configure apsect来激活@Configurable的处理。 尤其是确保不要在已经注册成常规的Spring [bean的bean的类上面使用@configurable这个注解。如果非要这么做](mailto:bean的bean的类上面使用@configurable这个注解。如果非要这么做) ----- 你就会出现两次初始化---- 一次是通过Spring container进行的初始化 另一个是通过aspectj的拦截给出的第二次初始化

\*\*注意

对于大多数的情况，”在新建对象的初始化运行完成之后” 是合适的。 这个上下文中 “初始化以后”意味着在这个对象已经被创建之后，对应的依赖就会被注入进来。----- -这就意味着 在构造方法被执行之前 需要的依赖是不可用的（指的是在构造方法体内）。如果你想在构造方法体被执行之前 ---- 也就是让依赖在构造方法体重可用 ---- 那么你需要定义这个注解@Configurable ---- 定义方式如下：@Configurable(preConstructor=true)

\*\*除了@Configurable这个方面，[spring-aspect.jar还包含了用于驱动标记了@Transaction注解的Spring的事务管理的类型或者方法的AspectJ的一个切面 aspect。这主要是为了那些想在Spring](mailto:spring-aspect.jar还包含了用于驱动标记了@Transaction注解的Spring的事务管理的类型或者方法的AspectJ的一个切面%20aspect。这主要是为了那些想在Spring)容器之外使用使用框架的事务支持的用户而生成的。

\*\*这个能够解释@Transactional注解的切面aspect就是AnnotationTransactionAspect类。当使用这个切面的时候，你必须标注这个实现类（和/或者 这个类中的方法）---- 而不是这个类实现的接口。**AspectJ遵循Java的规则 ---- 就是 接口上面的注解不能被继承**。

\*\*一个类上面的@Transactional注解明确了类中任何public的方法的执行的默认事务的语义】

我们看一下具体用法

这个用法 就是spring容器之外创建一个对象 并对这个对象进行注入

有一个特殊的用法就是 @Configurable(preConstruction=true) ----- 这个就是 你的bean还没有构建出来 就想应用其他的Spring容器中管理的Bean -----

----- 一般用法是构造之后 再装配

---- 这个是AOP中最复杂的 使用额外的包【下页PPT】



Spring aop的包 aspectjrt的包 和 aspectjweaver.jar这个包

------ spring-instrument.jar做了一个增强 ----- 就是AOP那届讲清楚

用的是javaagenet 有一个jvm的启动参数 【ppt上说的比较清楚 老师本身没说清楚】



这里面的MyOrder这个类 就是使用@Configurable 表示这个是container管理之外的类对象 ---- 里面的@Resource表明了 Spring容器管理之外的对象 也是可以对其进行Spring装配的

---- 在xml中 配置是

前面讲了 <context:load-time weaver> 是开启运行时的编织 还要开启@Configurable的功能 那就是 <context:spring-configured>

在main中看 这个MyOrder是自己的再main中new出来的 new MyOrder(); ----- 下面的代码就是需要看下 我这个MyOrder是一个什么样子的代理 ------

打印一下这三句话 这个背后很复杂

=-=== 看一下这个用法：

<context:load-time weaver>是开启AspectJ的功能

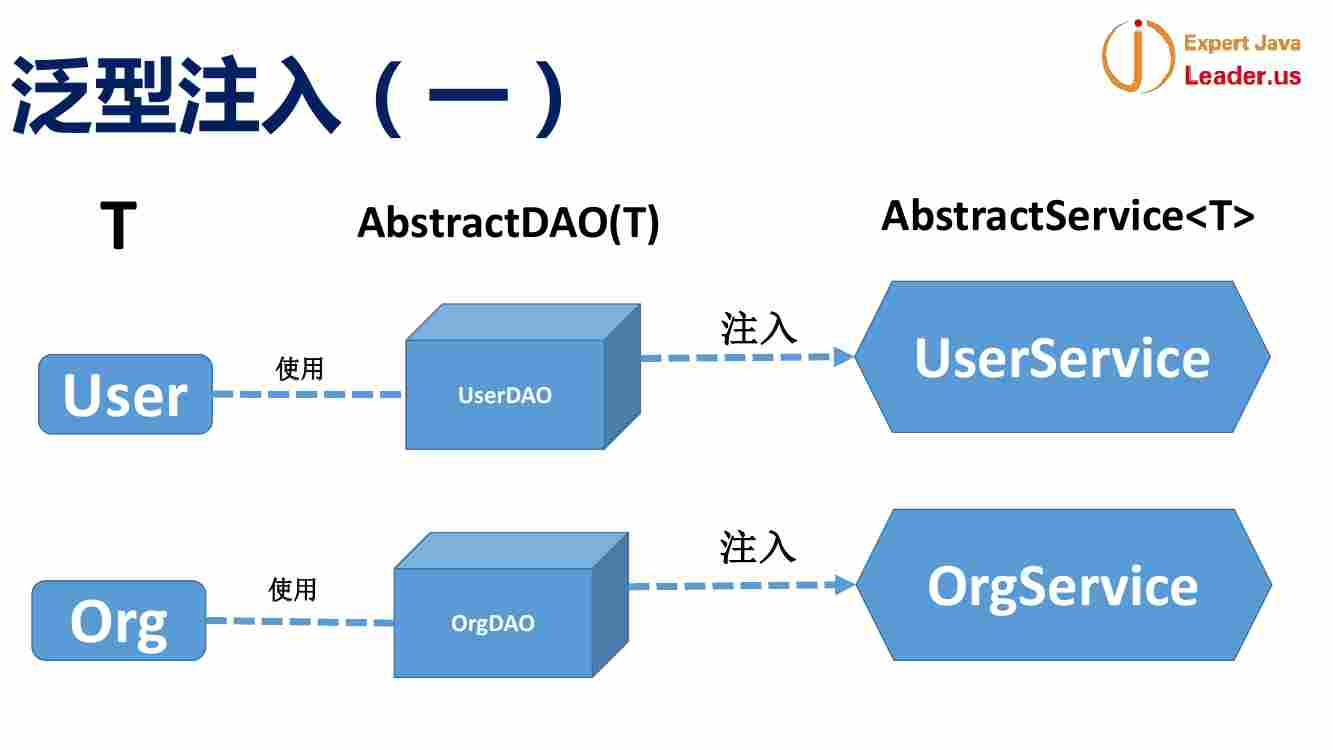
<context:spring-configured>是开启@Configurable这个注解的能力

然后 我们这个需要自己的new的对象但是 又要用Spring进行装配的class ---- 需要使用@Configurable进行标记



最后运行的时候 javaagent:runlib/spring-instrument-4.2.3.RELEASE.jar这个JVM的启动参数 用到了这个里面的类 这个负责去把前面MyOrder这个类对象进行编织 --- 生成的类对象 myorder对象已经不是这个类本身了 是被做过处理了 ---- 被AspectJ【后面的打印信息没讲明白】

下面看泛型注入



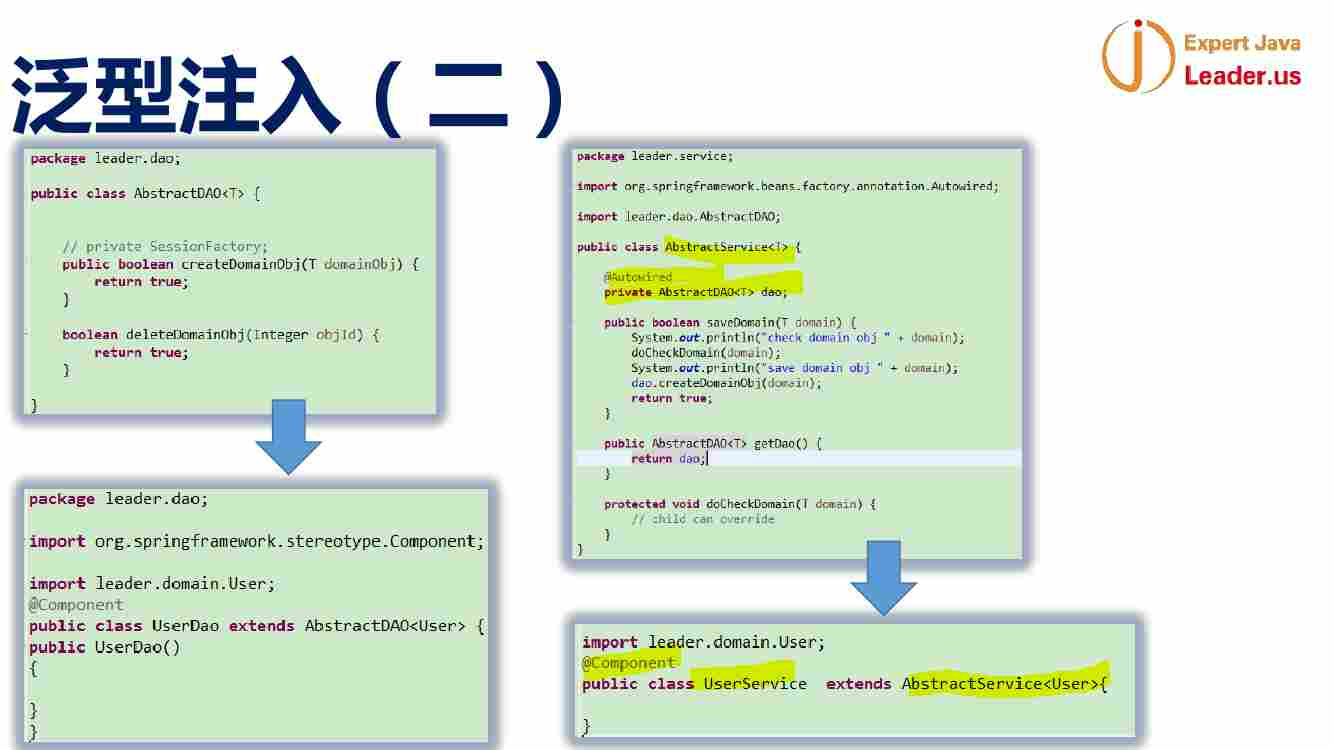
泛型注入 有点复杂

这个泛型 T 是一个User ---- 泛型的DAO 和泛型的Service --- 做法就是 AbstractDao操作的是T User

都是从AbstractDao<T> 实现的UserDAO和OrgDao --- AbstractService 实现类是 UserService和OrgService 但是 注入的分别是UserDAO和OrgDAO

就是UserService注入的是userDao 而不是OrgDAO ----- 虽然OrgDao也是AbstractDao实现类

这个是比较复杂的 Spring还是支持的

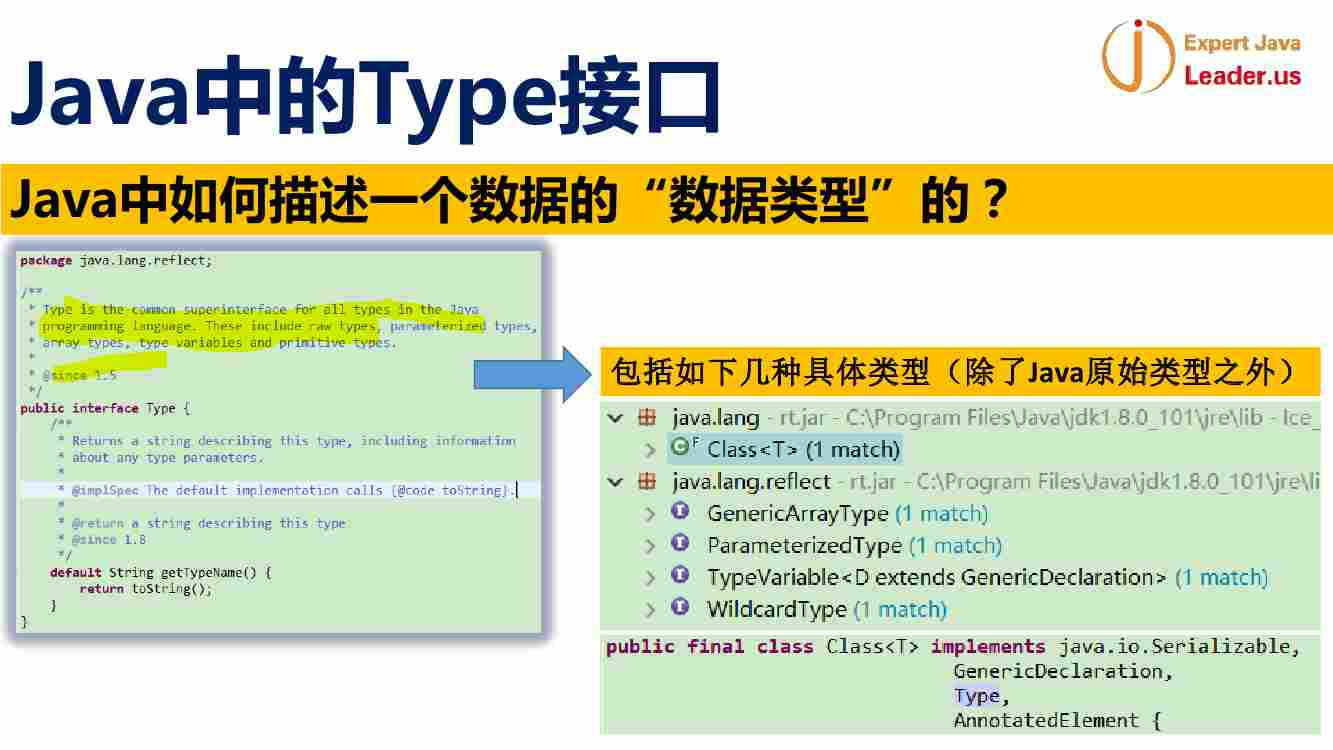


这个UserDao负责操作的是User这个域对象 UserService操作的是UserDAO



这个代码中 有一点 泛型不支持Resouce 但是 Autowired是支持的 不清楚Spring是不是故意的 还是疏忽

按照道理 都可以 但是 现在不一样 Resource不支持泛型注入 -----



JDK5中的Type表示数据的类型 java.lang.reflect.Type ----- Java中的所有都属于某种类型

Class只是一种Type

还有 就是数组结构 泛型参数 ParameterizedType 还有就是

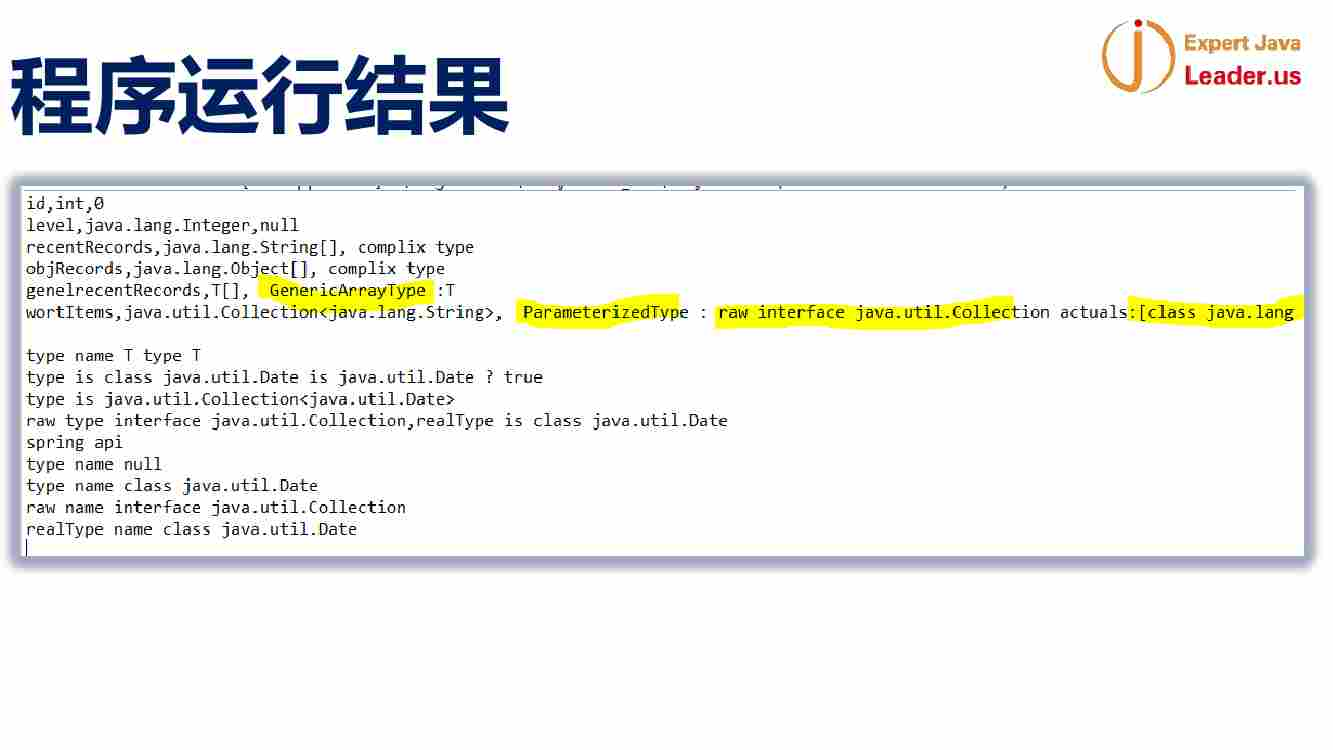
【没说清楚】



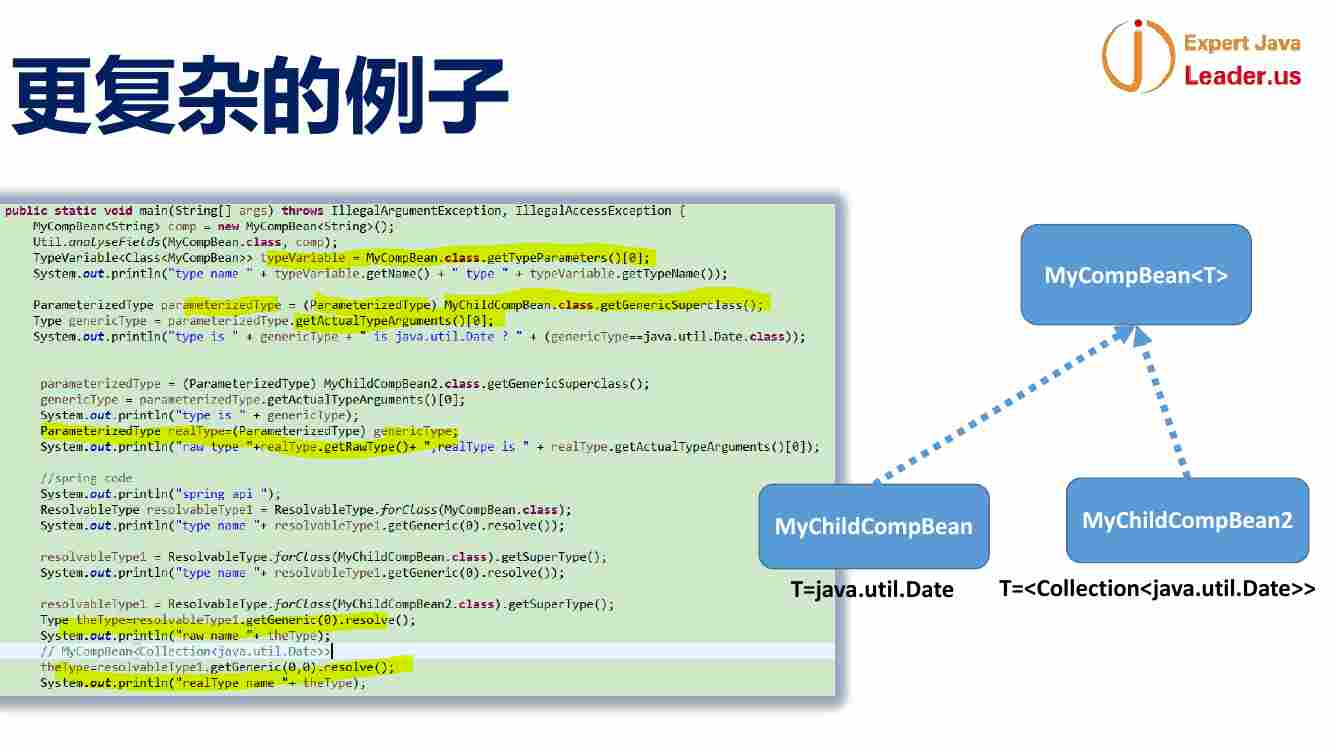
这个是关于反射的例子

这段代码 就是通过Class得到Object的信息 这个属性到底是什么类型的

参数化的- ---- 泛型的真正的类型是什么类型 ------ MyCompBean<T>中的T的真正的类型 才有具体的



可以自己分析一下运行的结果



这个里面有一个更加复杂的地方 就是 我的MyChildCompBean2的泛型里面还是泛型

发现这里面有一个嵌套的结果 这样有一个不断向下嵌套的过程

建议这里面代码 把这里面作对

Spring这里面提供了简单的类 ---------

【我可以把自己总结的Type整理出来】



这两个容易弄晕

isAssignableFrom ----- Y这个Class就是继承自X ===== 【可以方法调用的参数上面直接可以看出 使用的是Y.class 而不是 Y这个类的对象】

【貌似看Struts2那本书的时候 总结了isAssignableFrom等等 几个reflection的方法的使用】

下面X.class.isinstance(y)

这两者都是一个意思

把Y赋值给X事合法的

这是容易被问到 或者蒙掉的



------ 这个是google上面的讨论 ----- 就是没理解哪个方法使用的是对象本身 哪个使用的是Class参数



Spring中是可以注入集合属性 Map List都可以注入

比如 allDaos这个就是Map

这个注入的时候 value就是DAO本身 【老师没有点出来 要自己看看如何使用】

【main方法中 用了JDK8的函数式编程】