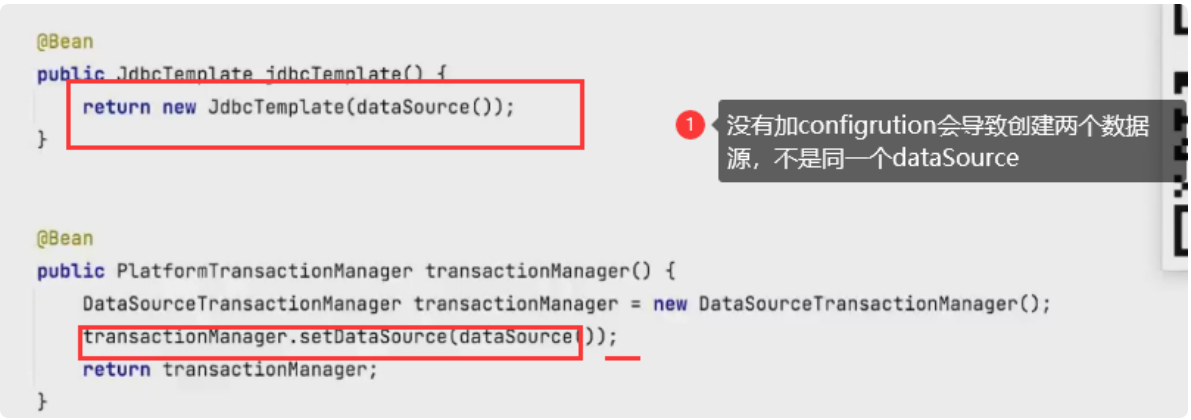
走近spring to 走进spring

1.spring中Transactional失效问题

- spring事务其实是aop的一个体现：怎么说呢，我们在控制事务的时候需要操作数据库，我们势必会autoCommit=false;提交事务；回滚事务；而我们将这些共同要做的事情交给transManager事务管理器来做通过切面的方式，在sql执行前开启事务，然后没有异常可以提交事务，有异常回滚事务。
- **事务失效因素一**：当我们在配置好所有时忘记加上configruation这个注解时会导致事务失效,没有加你在配置数据源的时候直接调用，也就是this.dataSource(),这个时候就是一个普通对象在调用数据源方法，导致失效.



加上就是同一个了,为什么呢? 加了之后appConfig就是一个代理对象，代理对象调用dataSource（）方法就会先在容器中找，找到就返回，所以是同一个。

- **事务失效因素二**：当一个有事务的方法调用了另一个有事务的方法，调用方式不对也会导致事务失效

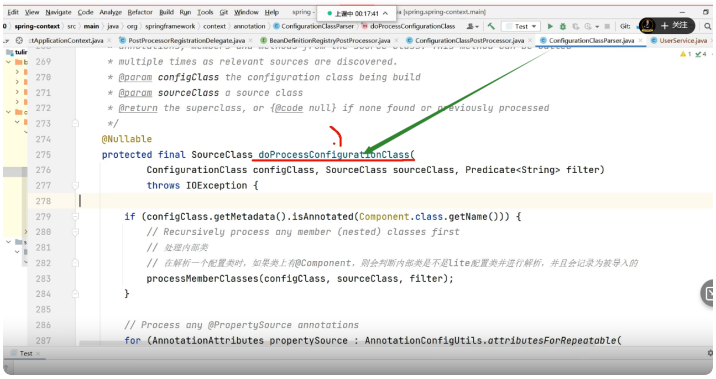
- **cglib方式导致事务失效**：cglib动态代理是我们的代理类继承一个被代理的类可以通过super.test()在方法前后做一些事情

上图假如直接super.test()那必然会导致事务失效，调用的对象是一个普通对象，肯定没有任何用处，当然spring里面肯定不会这么搞。

2.compentScan是如何做扫描

2.1 那些注解代表一个配置类

在源码中显示: configruation,compent,compentScan,Import,ImportResource 只要有上面的注解都代表这个类是一个bean配置类



```
 * multiple times as relevant sources are discovered.
 * @param configClass the configuration class being build
 * @param sourceClass a source class
 * @return the superclass, or {@code null} if none found or previously processed
 */
@Nullable
protected final SourceClass doProcessConfigurationClass(
        ConfigurationClass configClass, SourceClass sourceClass, Predicate<String> filter)
        throws IOException {

    if (configClass.getMetadata().isAnnotated(Component.class.getName())) {
        // Recursively process any member (nested) classes first
        // 处理内部类
        // 在解析一个配置类时，如果类上有@Component，则会判断内部类是不是lite配置类并进行解析，并且会记录为被导入的
        processMemberClasses(configClass, sourceClass, filter);
    }

    // Process any @PropertySource annotations
    for (AnnotationAttributes propertySource : AnnotationConfigUtils.attributesForRepeatable(
```

```
Map<String, Object> config = metadata.getAnnotationAttributes(Configuration.class.getName());

// 存在@Configuration，并且proxyBeanMethods不为false(为true或为null)时，就是Full配置类
if (config != null && !Boolean.FALSE.equals(config.get("proxyBeanMethods"))) {
    beanDef.setAttribute(CONFIGURATION_CLASS_ATTRIBUTE, CONFIGURATION_CLASS_FULL);
}
// 存在@Configuration，并且proxyBeanMethods为false时，是lite配置类
// 或者不存在@Configuration，但是只要存在@Component、@ComponentScan、@Import、@ImportResource四个中的一个
// 或者不存在@Configuration，只要存在@Bean注解了的方法，就是lite配置类
else if (config != null || isConfigurationCandidate(metadata)) {
    beanDef.setAttribute(CONFIGURATION_CLASS_ATTRIBUTE, CONFIGURATION_CLASS_LITE);
}
else {
    return false;
}
```

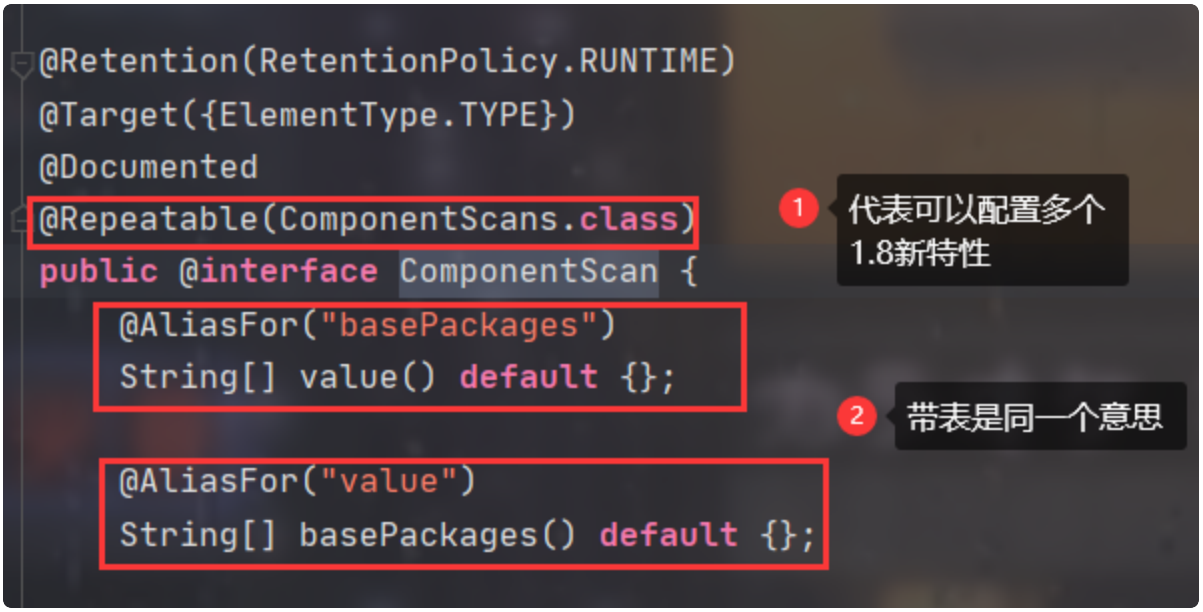
```
private static final Log logger = LogFactory.getLog(ConfigurationClassUtils.class);

private static final Set<String> candidateIndicators = new HashSet<>( initialCapacity: 8);

static {
    candidateIndicators.add(Component.class.getName());
    candidateIndicators.add(ComponentScan.class.getName());
    candidateIndicators.add(Import.class.getName());
    candidateIndicators.add(ImportResource.class.getName());
}
```

2.2 注解解析的一个流程

当spring知道那些是配置类的时候，解析配置---->返回一个beanDefinition的集合-------->遍历---------->是不是还有配置类，如果有配置bean就在去解析.

2.3 compentScan注解的属性

1. value compentScan



2. compent中的value

- spring 会先判断你有没有给value值 给了用你的
- 没给value值spring 会根据你的类名,将你的类名传给了jdk中的一个方法规则是 第一个和第二个都大写，bean的名字直接返回
- 如果不是，就首字母替换成小写

3. compentScan中的 scopedProxy属性

- bean的作用域：解释

解释 XML 复制代码

```
在springmvc中 @requestScope 或者 @sessionScope 或者  原型bean
@Scope(WebApplicationContext.Scope_request)   接收一个请求才生成===>相当于@requestScope
场景:我们可能会依赖注入这个有作用域的bean
@RequestScope
public class OrderService{
}
@Component
public class UserService{
  @Autowired
  private OrderService orderService;      ===>赋值没有这个对象，不能为null会报错，搞代理
}

//指定scope时
@Scope(WebApplicationContext.Scope_request,proxyMode=ScopedProxyMode.Interfaces)
public class OrderService{
}
```

- 而scopedProxy就是一个全局的代理对象的说明

4 resourcePattern 属性

- 默认值是 String resourcePattern() default "**/*.class"；代表的就是扫描那个目录下的那类型文件

5 excludeFilters属性

```
ComponentScan.Filter[] includeFilters() default {};

ComponentScan.Filter[] excludeFilters() default {};
```

```
@Retention(RetentionPolicy.RUNTIME)
@Target({})
public @interface Filter {
    FilterType type() default FilterType.ANNOTATION;

    @AliasFor("classes")
    Class<?>[] value() default {};

    @AliasFor("value")
    Class<?>[] classes() default {};

    String[] pattern() default {};
}
```

filter怎么配置 XML 复制代码

```
 @ComponentScan.Filter(type = FilterType.ASSIGNABLE_TYPE,value =
UserService.class)
 @ComponentScan.Filter(type =
FilterType.ANNOTATION,value=Component.class)

```

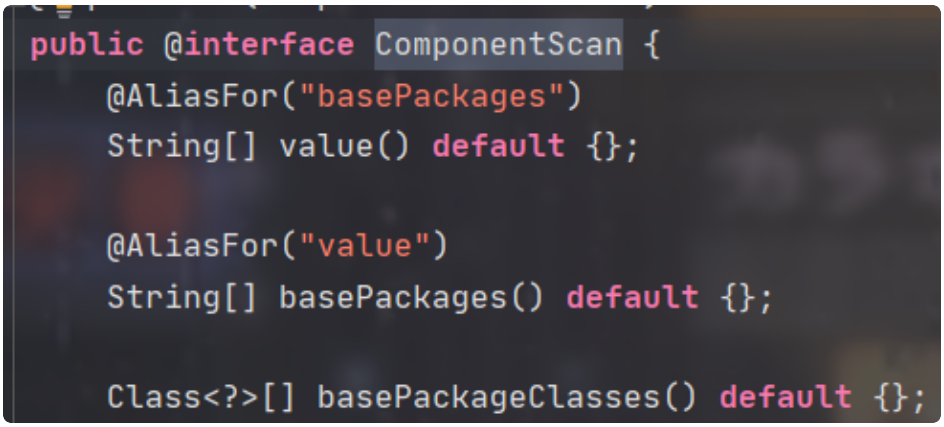
6 includeFilters属性

配置一个过滤器，bean过滤的时候用是不是符合includeFilters相匹配才有可能成为定义bean

7 lazyInit 属性

全局懒加载配置

8.basePackages和basePackageClass属性

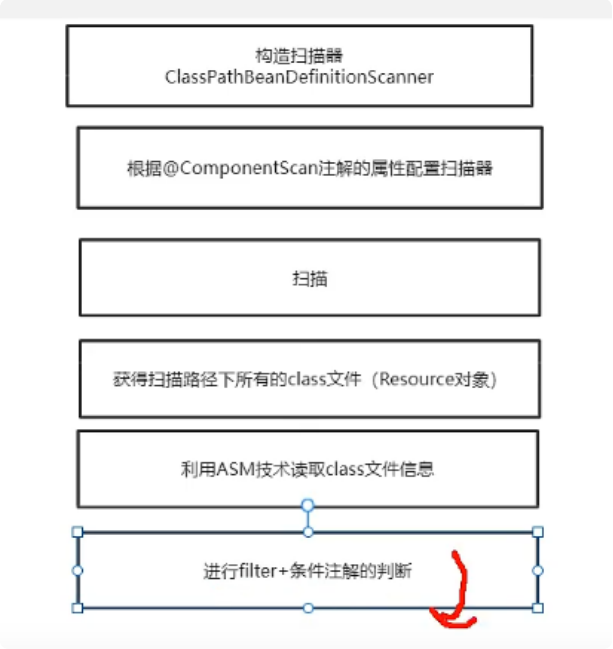


- basePackagesClass 代表某个类所在的包是扫描包
- basePackages和value代表配置的扫描包
- 什么都没配代表，加compentScan所在的类作为扫描包

3.怎么利用CompentScan进行扫描

扫描过程最好对着源码理解 XML | 复制代码

```
//声明: userDefaultFilters 默认为true  sping默认为includeFilters添加过滤器
//3个compent named mangaBean
@CompentScan
=> 对basePackages的集合进行排除
=> execulteFilter排除配置类
=> findCandipageComent(basePages)[可能有多个compentScan]
=>  遍历
=>scanCandiateCompoents(basePages)
=>(得到所有的包名)   构造一个扫描路径
=>pacakageSearchPath= classpath*/com/zhouyu/**/*.class
===>得到一个包下的所有class文件 resource[]
=>遍历文件.class
=>ASM技术 (ASM(操作字节码文件):字节码文件有自己的规范, ASM它会遵守字节码规范,
    ASM它的API可以获取到注解, 接口, 名字, 方法等 (关键是它不需要将.java文件加入到jvm
中
    , 打破jvm的懒加载机制) --->扫描对象字节码文件)
=>判断那些类需要排除 excluderFilter 过滤掉
=> includeFilters  判断那些过滤掉的bean匹配includeFilters的匹配条件
=> 符合就进一步过滤 某些类上有没有加@Conditional这种注解条件
  @Component
  @Conditional(ZhouyuCondition.class)
  public class ZhouYuService{
  }
  ZhouyuCondition.class implments Condition{
      方法条件  ==>boot里面大量这么干
  }
=>这样才有成功定义bean的资格 定义beanDeifed放在map中
=> 对这些定义了beandefined的bean在过滤
   先看你是不是一个独立类  不是 false 内部类,静态内部类(独立)
=>是独立类 判断你是不是接口
  是接口 false
=>不是接口 判断你是不是抽象类
=>是抽象类 判断抽象类中有没有被lookup修饰的方法
  @lookup (value="orderService")
  public OrderService test(){
    return null;   //找一个orderService的bean返回
  }
```



4.单例bean和单例模式的区别

单例bean是单例模式的一种实现

对于单例模式而言：

- 当只有一个进程的时候，可以保证jvm中只有一个类型的对象
- 对于多个进程而言，多个jvm你的单例模式就不能保证只有一个类型的对象；
- 所以单例模式它是有范围限制的

同样单例bean也是有范围的

- 单例bean只的是在spring容器，**只有一个bean的名字对应一个对象**
- 并不是所谓的在spring容器中，我只有一个类型的对象才是单例bean 一个类型就是可以定义多个对象
- **多例bean是,一个名字对应多个对象**