3、训练营期间：从BeanDefinition到一个Bean

2020/11/23 22:51

1、BeanDefinition的原始表示（所谓元信息）

除了常用的XML方式，还有Properties 资源配置，比如下面的user.properties文件

```
user.(class) = org.geekbang.thinking.in.spring.ioc.overview.domain.User
user.id = 001
user.name = 小马哥
user.city = HANGZHOU,BEIJING #枚举值
```

可以用PropertiesBeanDefinitionReader 读取properties文件配合DefaultListableBeanFactory 来生成bean

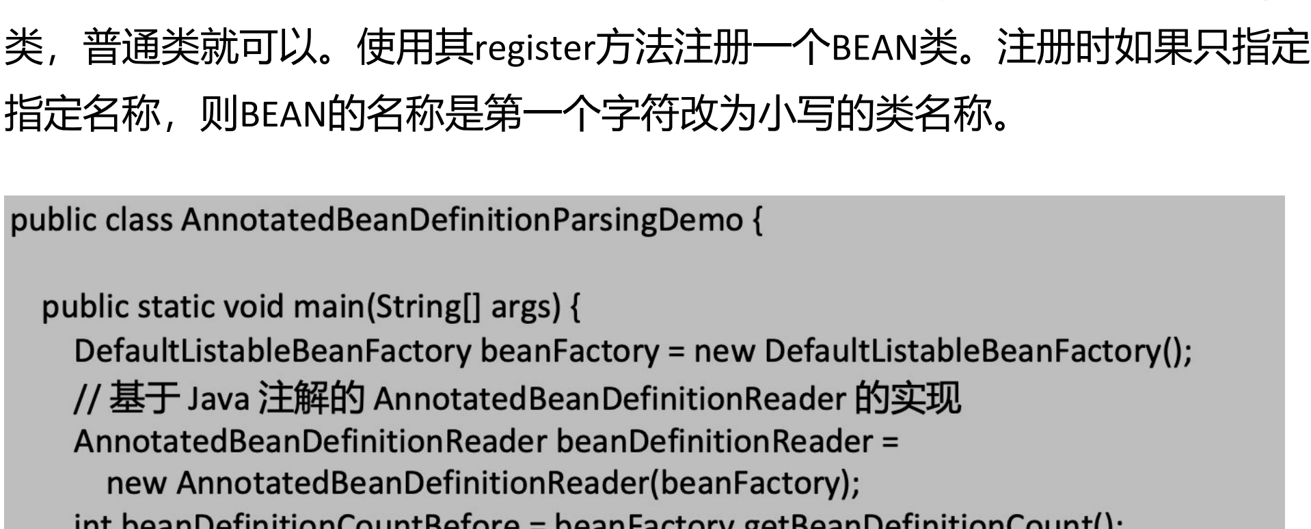
```
public class BeanMetadataConfigurationDemo {

  public static void main(String[] args) {
    DefaultListableBeanFactory beanFactory = new DefaultListableBeanFactory();
    // 实例化基于 Properties 资源 BeanDefinitionReader
    PropertiesBeanDefinitionReader beanDefinitionReader = new
PropertiesBeanDefinitionReader(beanFactory);
    String location = "META-INF/user.properties";
    // 基于 ClassPath 加载 properties 资源
    Resource resource = new ClassPathResource(location);
    // 指定字符编码 UTF-8
    EncodedResource encodedResource = new EncodedResource(resource, "UTF-8");
    int beanNumbers = beanDefinitionReader.loadBeanDefinitions(encodedResource);
    System.out.println("已加载 BeanDefinition 数量： " + beanNumbers);
    // 通过 Bean Id 和类型进行依赖查找
    User user = beanFactory.getBean("user", User.class);
    System.out.println(user);
  }

}
```

可以看出properties文件在JAVA中就是一种Resource。关于资源管理，详见

2、BeanDefinition的读取

- 面向资源BeanDefinition解析
    - BeanDefinitionReader
    - XML 解析器 - BeanDefinitionParser
- 面向注解BeanDefinition解析
    - AnnotatedBeanDefinitionReader

虽然有XML 解析器 - BeanDefinitionParser，但是基于 XML 资源装载 Spring Bean 配置元信息的底层实现是XmlBeanDefinitionReader。



实现场景	实现类
XML 资源	XmlBeanDefinitionReader
Properties 资源	PropertiesBeanDefinitionReader
Java 注解	AnnotatedBeanDefinitionReader

1.1节的PropertiesBeanDefinitionReader 就是一种BeanDefinitionReader 。上面的而集中解析手段，AnnotatedBeanDefinitionReader 比较特殊，因为它不是面向资源的，它处理的是配置类。

AnnotatedBeanDefinitionReader处理的不一定是@Commponent或者 @Configuration标注的类，普通类就可以。使用其register方法注册一个BEAN类。注册时如果只指定类型，没 有指定名称，则BEAN的名称是第一个字符改为小写的类名称。

```
public class AnnotatedBeanDefinitionParsingDemo {

  public static void main(String[] args) {
    DefaultListableBeanFactory beanFactory = new DefaultListableBeanFactory();
    // 基于 Java 注解的 AnnotatedBeanDefinitionReader 的实现
    AnnotatedBeanDefinitionReader beanDefinitionReader =
      new AnnotatedBeanDefinitionReader(beanFactory);
    int beanDefinitionCountBefore = beanFactory.getBeanDefinitionCount();
    // 注册当前类（非 @Component class）
    beanDefinitionReader.register(AnnotatedBeanDefinitionParsingDemo.class);
    int beanDefinitionCountAfter = beanFactory.getBeanDefinitionCount();
    int beanDefinitionCount = beanDefinitionCountAfter - beanDefinitionCountBefore;
    System.out.println("已加载 BeanDefinition 数量： " + beanDefinitionCount);
    // 普通的 Class 作为 Component 注册到 Spring IoC 容器后，
    // 通常 Bean 名称为 annotatedBeanDefinitionParsingDemo
    // Bean 名称生成来自于 BeanNameGenerator，注解实现
AnnotationBeanNameGenerator
    AnnotatedBeanDefinitionParsingDemo demo =
      beanFactory.getBean("annotatedBeanDefinitionParsingDemo",
        AnnotatedBeanDefinitionParsingDemo.class);
    System.out.println(demo);
  }
}
```

读取后，配置信息（XML或者注解或者properties文件）就以PropertyValue的形式保存到了BeanDefinition中，用于生成Bean，可参见《5、训练营期间：把BeanDefinition保存的配置信息映射到Bean的属性中》。

3、BeanDefinition的注册

BeanDefinition注册接口是 BeanDefinitionRegistry，有一个实现是DefautListableBeanFactory。实现过程就是把BeanDefinition放入一个Map中。但是 MAP中的元素是无序的，所以还有一个ArrayList来按照注册的顺序保存BEAN的名字。

4、BeanDefinition的合并

父子 BeanDefinition 合并:参见《依赖处理过程》第2节。最顶层的父类对应RootBeanDefinition，此时不存在父类，不需要合并。有父类的情况，使用GenericBeanDefinition。默认的都是GenericBeanDefinition，即使没有父类，只 有在合并完成后，才会创建出**RootBeanDefinition**。即使原始的BEAN有父类。

5、利用 RootBeanDefinition生成Bean

BeanDefinition里保存了从资源中读取的配置信息，这些信息可以理解为键值对，这些键值对最终要对应到Bean的属性中去，具体的映射过程详见《5、训练营期间：把BeanDefinition保存的配置信息映射到Bean的属性中》。

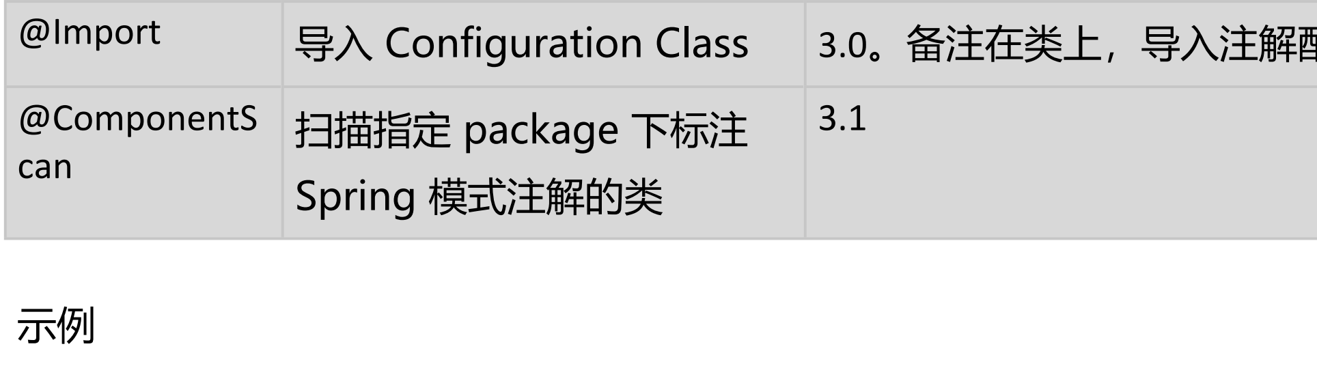
这里只描述从 RootBeanDefinition生成Bean的步骤，详见《Spring Bean生命周期》

阶段	形态	使用的API
	RootBeanDefinition	
Spring Bean Class加载阶段	↓	AbstractBeanFactory# resolveBeanClass
	得到class对象	
实例化前阶段	↓	InstantiationAwareBeanPostProcessor# postProcessBeforeInstantiation
	得到对象Object	
上一阶段没有进行， 则进入实例化阶段		1、使用策略InstantiationStrategy，用反射调用午餐构造器； 2、构造器依赖注入
	得到对象Object	
实例化后阶段		InstantiationAwareBeanPostProcessor# postProcessAfterInstantiation
	（判断是否该设置BEAN的属性值）	
属性赋值前阶段		1、InstantiationAwareBeanPostProcessor# postProcessPropertyValues（5.1之前） 2、InstantiationAwareBeanPostProcessor# postProcessProperties (Spring5.1)
	(对配置元信息做一些增删 改查的操作)	
Spring Aware 接口		框架调用如下接口进行设置 BeanNameAware BeanFactoryAware EnvironmentAware ApplicationEventPublisherAware ApplicationContextAware ……
Spring Bean 初始化前阶段		BeanPostProcessor# postProcessBeforeInitialization
Spring Bean 初始化阶段		• 调用@PostConstruct 标注方法 • 调用实现 InitializingBean 接口的 afterPropertiesSet() • 调用自定义初始化方法
Spring Bean 初始化后阶段		BeanPostProcessor# postProcessAfterInitialization
Spring Bean 初始化完成阶段		SmartInitializingSingleton# afterSingletonsInstantiated

6、Spring IoC 容器配置元信息

详见《Spring配置元信息》。

6.1、Spring IoC 容器相关 XML 配置



命名空间	所属模块	Schema 资源 URL
beans	spring-beans	https://www.springframework.org/schema/beans/spring-beans.xsd
context	spring-context	https://www.springframework.org/schema/context/spring-context.xsd
aop	spring-aop	https://www.springframework.org/schema/aop/spring-aop.xsd
tx	spring-tx	https://www.springframework.org/schema/tx/spring-tx.xsd
util	spring-beans	https://www.springframework.org/schema/util/spring-util.xsd
tool	spring-beans	https://www.springframework.org/schema/tool/spring-tool.xsd

XML配置的原理，可以参见《1、Java训练营学习补充》的第4节。

6.2、基于 Java 注解装载 Spring IoC 容器配置元信息

Spring 注解	场景说明	备注
@ImportResource	替换 XML 元素 <import>	3.0。备注在类上，导入XML配置
@Import	导入 Configuration Class	3.0。备注在类上，导入注解配置类
@ComponentScan	扫描指定 package 下标注 Spring 模式注解的类	3.1

示例

```
// 将当前类作为 Configuration Class
@ImportResource("classpath:/META-INF/dependency-lookup-context.xml")
@Import(User.class)
public class AnnotatedSpringIoCContainerMetadataConfigurationDemo {
   public static void main(String[] args) {
        AnnotationConfigApplicationContext context =
                new AnnotationConfigApplicationContext();
        // 注册当前类作为 Configuration Class
        context.register(AnnotatedSpringIoCContainerMetadataConfigurationD
emo.class);
        // 启动 Spring 应用上下文
        context.refresh();
        // beanName 和 bean 映射
        Map<String, User> usersMap = context.getBeansOfType(User.class);
        for (Map.Entry<String, User> entry : usersMap.entrySet()) {
            System.out.printf("User Bean name : %s , content : %s \n",
                    entry.getKey(), entry.getValue());
        }
        // 关闭 Spring 应用上下文
        context.close();
    }
}
```

6.3、属性配置注解

6.3.1、注解配置BEAN，读取资源文件配置的属性值

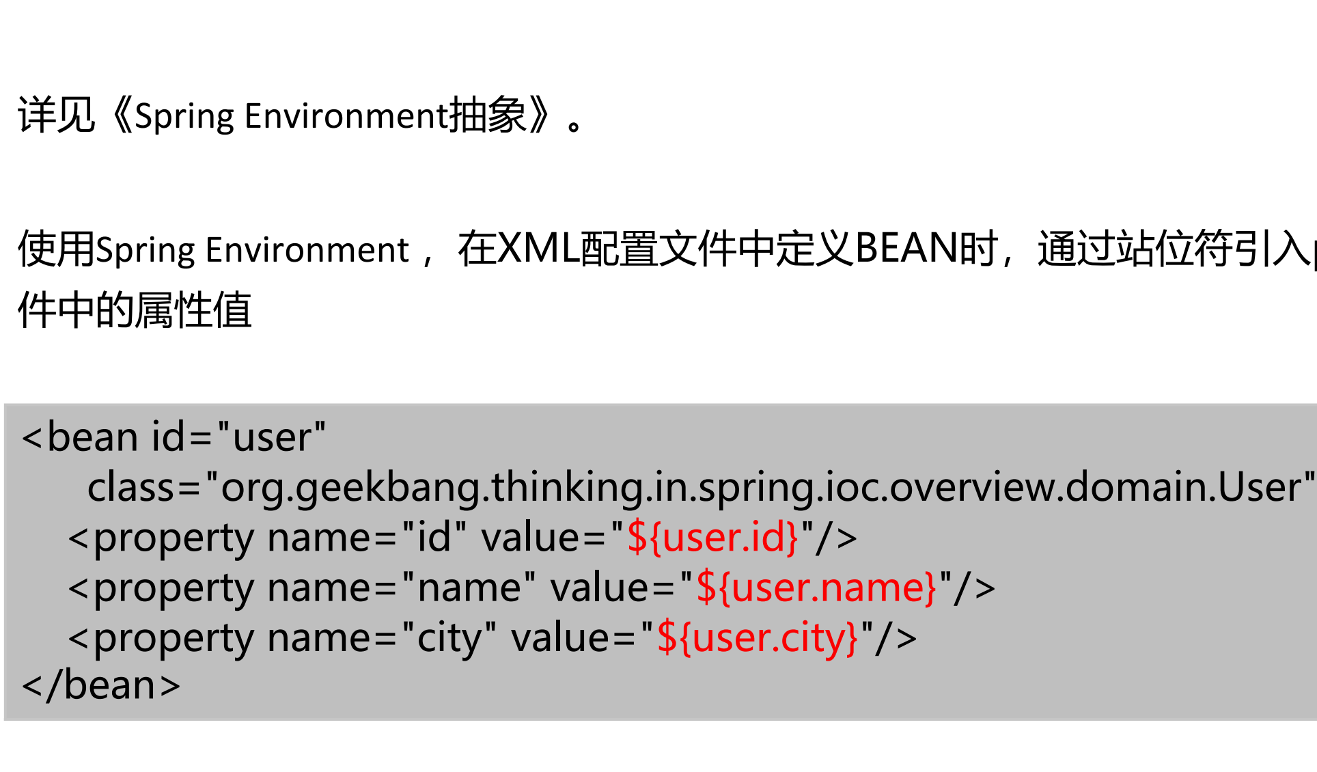
Spring 注解	场景说明	备注
@PropertySource	配置属性抽象 PropertySource 注解	3.1。备注在类上
@PropertySources	@PropertySource 集合注解	4.0。备注在类上

代码示例

```
@PropertySource("classpath:/META-INF/user-bean-
definitions.properties") // Java 8+ @Repeatable 支持
@PropertySource("classpath:/META-INF/user-bean-definitions.properties")
// @PropertySources(@PropertySource(...))
public class AnnotatedSpringIoCContainerMetadataConfigurationDemo {
    /**
     * user.name 是 Java Properties 默认存在，当前操作系统用户：mercyblitz,
     *            而非配置文件中定义"小马哥"
     * @param id
     * @param name
     * @return
     */
    @Bean
    public User configuredUser(@Value("${user.id}") Long id, @Value("${use
r.name}") String name) {
        User user = new User();
        user.setId(id);
        user.setName(name);
        return user;
    }
}
```

@PropertySource的数据来源除了properties外，还可以来源于Java System Properties，也就是JAVA进程运行时的-D参数，比如上述代码注释的内容。

user-bean-definitions.properties的内容如下:



```
# User BeanDefinition 定义
user.(class) = org.geekbang.thinking.in.spring.ioc.overview.domain.User
# <property name="id" value="1"/>
user.id = 1
# <property name="name" value="小马哥"/>
user.name = 小马哥
# <property name="city" value="HANGZHOU"/>
user.city = HANGZHOU
# <property name="workCities" value="BEIJING,HANGZHOU"/>
user.workCities = BEIJING,HANGZHOU
# <property name="lifeCities">
## <list>
## <value>BEIJING</value>
## <value>SHANGHAI</value>
##</list>
# </property>
user.lifeCities = BEIJING,SHANGHAI
# <property name="configFileLocation" value="classpath:/META-INF/user-
config.properties"/>
user.configFileLocation = classpath:/META-INF/user-config.properties
```

6.3.2、XML方式读取，读取资源文件配置的属性值

详见《Spring Environment抽象》。

使用Spring Environment，在XML配置文件中定义BEAN时，通过站位符引入properties文件中的属性值

```
<bean id="user"
     class="org.geekbang.thinking.in.spring.ioc.overview.domain.User">
   <property name="id" value="${user.id}"/>
   <property name="name" value="${user.name}"/>
   <property name="city" value="${user.city}"/>
</bean>
```

资源文件default.properties内容如下

```
user.id  =1
user.name = 小马哥
user.city = HANGZHOU
```

为了处理上面红色的占位符，需要引入几个特殊的工具BEAN

- Spring3.1之前

```
<bean
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"
>
   <property name="location" value="classpath:/META-
INF/default.properties"/>
   <property name="fileEncoding" value="UTF-8" />
</bean>
```

- Spring3.1+

```
<bean class="org.springframework.context.support.PropertySourcesPlaceholderConfigurer" >
  <!-- user.name  = "mercyblitz" 而非 "小马哥" -->
  <property name="location" value="classpath:/META-INF/default.properties"/>
  <property name="fileEncoding" value="UTF-8" />
</bean>
```

注意，上面的用户名不会使用资源文件中的内容，而是系统环境变量里的用户名。