

配置信息的加载

所有信息都有配置项和信息 如何加载到程序中？ 这是框架的基础工具 非常重要



**我们可以用到的配置**：

properties在classpath中 属于代码的一部分

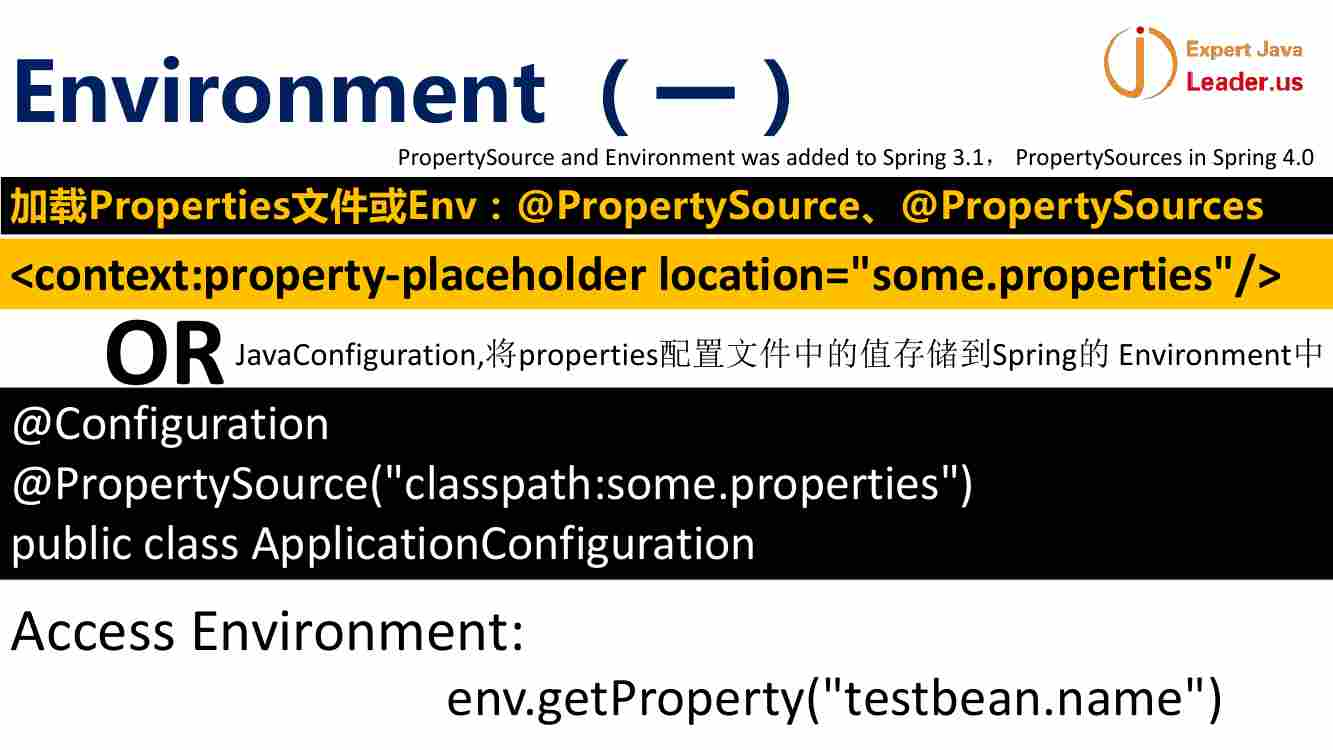
系统环境变量 里面 包括系统的Properties和jvm自身的Properties

用户的home目录 临时文件目录 JDK版本 等等

System ENV 在linux中用的很多 linux下面传递参数的方式 就是环境变量 ---- 系统的环境变量 Linux用的很多 利用环境变量传递参数

J2EE context变量 ---- J23EE容器运行的时候 有上下行下文

**Spring中都支持上面**



Spring对这个的支持也是渐变的

**在3.1中支持了PropertySource和Enviroment 支持属性的加载**

在4.0的时候 支持了PropertySourc**es【可以用多个】 ---- 可以用多个 --- XxxxSource 就可以理解为产生Xxxx的来源**

**在Spring中 XxxxSource 就会生成Xxxx ---- 把XXXXSource看成了Xxx一种来源 ---- 和DataSource效果一样**

**在xml中**

**<context:property-placeholder location=””>**

**对应的注解**

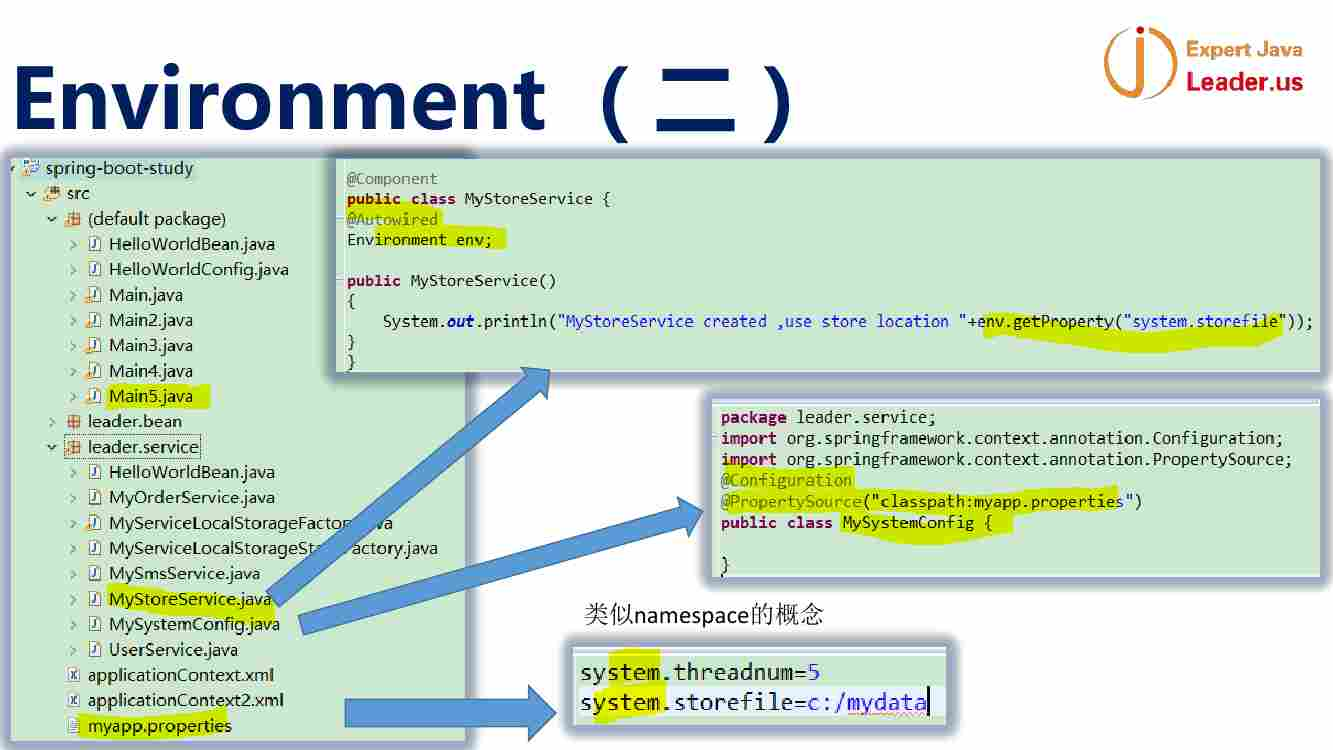
**@Configuration + @PropertySource(“classpath:xxxxx”)**

@PropertySource就会把配置文件中的内容加载到Enviroment 这个变量中去 ***【上节课讲的是@ImportResource这个注解 把xml中的内容加载成容器管理的组件 但是properties中的内容都是KV 谈不上组件的级别 就是一些配置的时候 需要的参数 所以这个时候*** [***使用@PropertySource进行load load完之后 不能AC.getBean***](mailto:使用@PropertySource进行load%20load完之后%20不能AC.getBean)***来获取 而是 env.getProperty来获取】【通过后面的代码 可以知道 这个env可以通过@Autowired直接通过Spring Container进行诸如即可】***

我们代码中 可以**env.getProperty(“xxxxx”)**

**多个Properties会合成一个 ------ 重名的 后面会覆盖前面的 ---- 这个就是配置文件的加载问题**

所以 要设计 properties的 Key ----- 加上前缀 保证不冲突



myapp.properties属于源码的一部分 所以在classpath下 所以属于源码的一部分 【实际上 就是 <context:place-holder locations=””>的配置 可以修改成@Configuration+@PropertySource进行统一配置 ------ 注解取代了xml 直接操作properties 然后 配合Environment进行使用】

这个key的前缀是system表示 给什么组件使用的

这样有了myapp.properties -🡪

然后在

**@Configuration**

**@PropertySource(“classpath:myapp.properties”)**

**Public class MySystemConfig{ //这个类叫MySystemConfig 所以 建议给myapp.properties的key加上一个前缀 system 之后 ---- 就是表示这个是一个properties的配置 而不是xml的Configuration的配置 别和上一节课弄混淆了！！！ 上面是使用@Configuration直接标记即可**

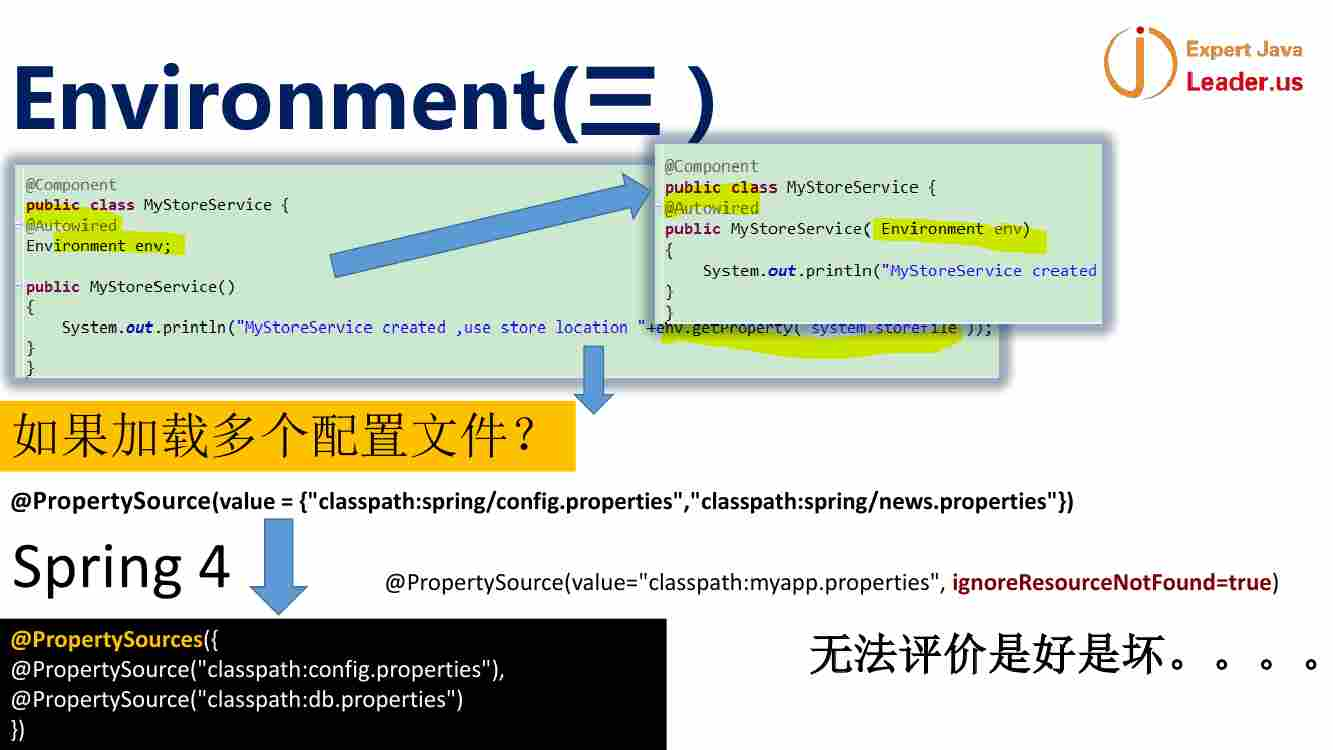
**//这样操作之后 我的myapp.properties这个里面的kv就会被spring初始化的时候 装载到环境变量中去**

**}**

之后 就可以在@Component标记为组件的 并且装配了@Autowired为 Environment env的环境变量中 获取到Properties给出的配置值

**Spring3以后的做法 ---- Spring3之后的做法**

***【Enviroment变量是Spring容器托管的 直接注入就可以使用】 简单明了***



**如果要加载个 在spring3中可以使用数组模式 --- @PropertySource后面可以是数组模式**

@PropertySource(value={xxx, xxxx})

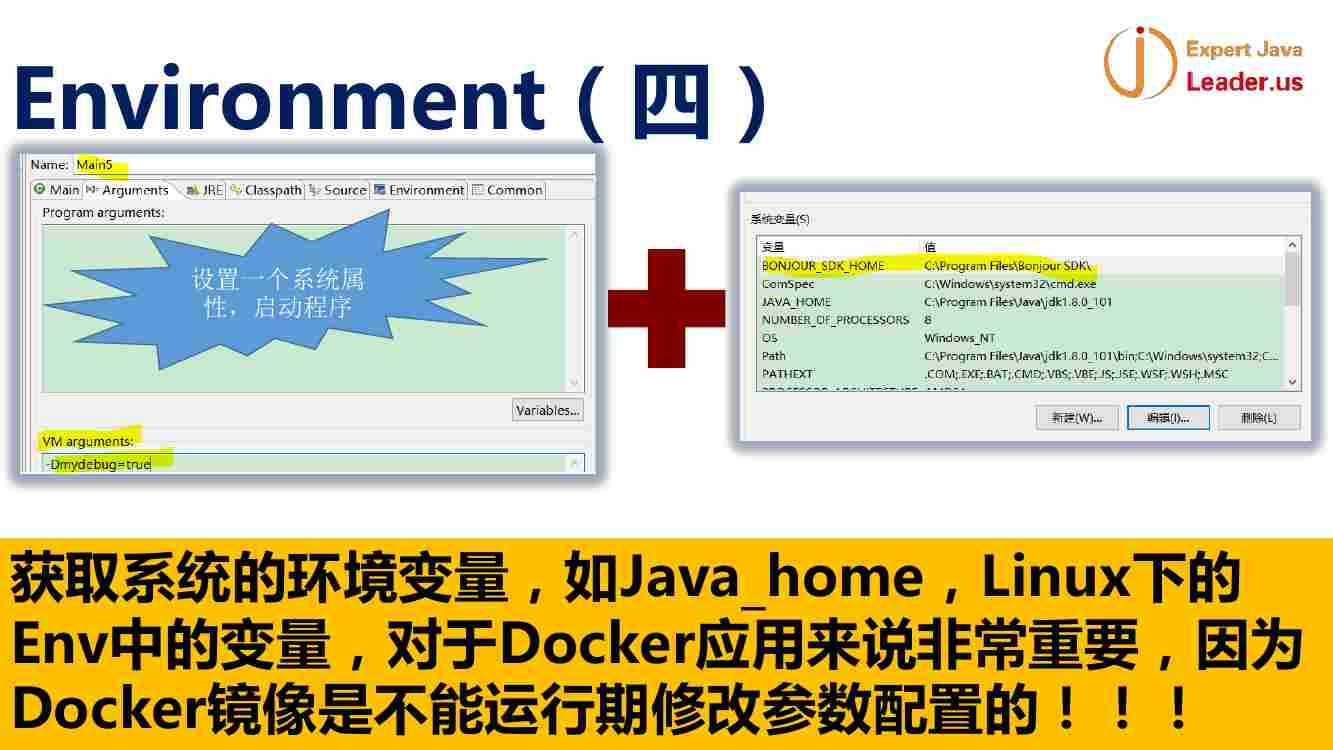
**到了Spring4中 添加了@PropertySources**({@PropertySource{xxxx}, @PropertySource{yyyyy}}) 这里面多个Property对象

这样 spring4中又分开了

**同时 @PropertySource 也升级了 加上了ignoreResourceNotFound** -----如果是以前没有这个属性的时候 如果找不到这个properties文件 就会抛出异常 not found ------ 但是 spring4给出了一个这种属性 就是没有这个文件 也可以正常运行***【老师说 不知道这是好是坏 没法评价】***

**多个properties文件 就是各自做各自的事情**

**多个Properties文件各有优缺点**



Env不仅仅是Property方式被解析 还可以 运行Java程序的时候 java xxx -Dmydebug=true 这样 也可以被获取到

系统环境变量

这个spring中的Enviroment对象 既可以获取一个在Eclipse调试阶段的配置的参数（左图的KV） 还可以获取一个类似于右边的windows或者linux下面的配置的kv ----- 所以PPT上面说 Java\_home Linux下面的环境变量 都可以获取到 ---- 好处就是 无论在哪里 访问方式都是一样的

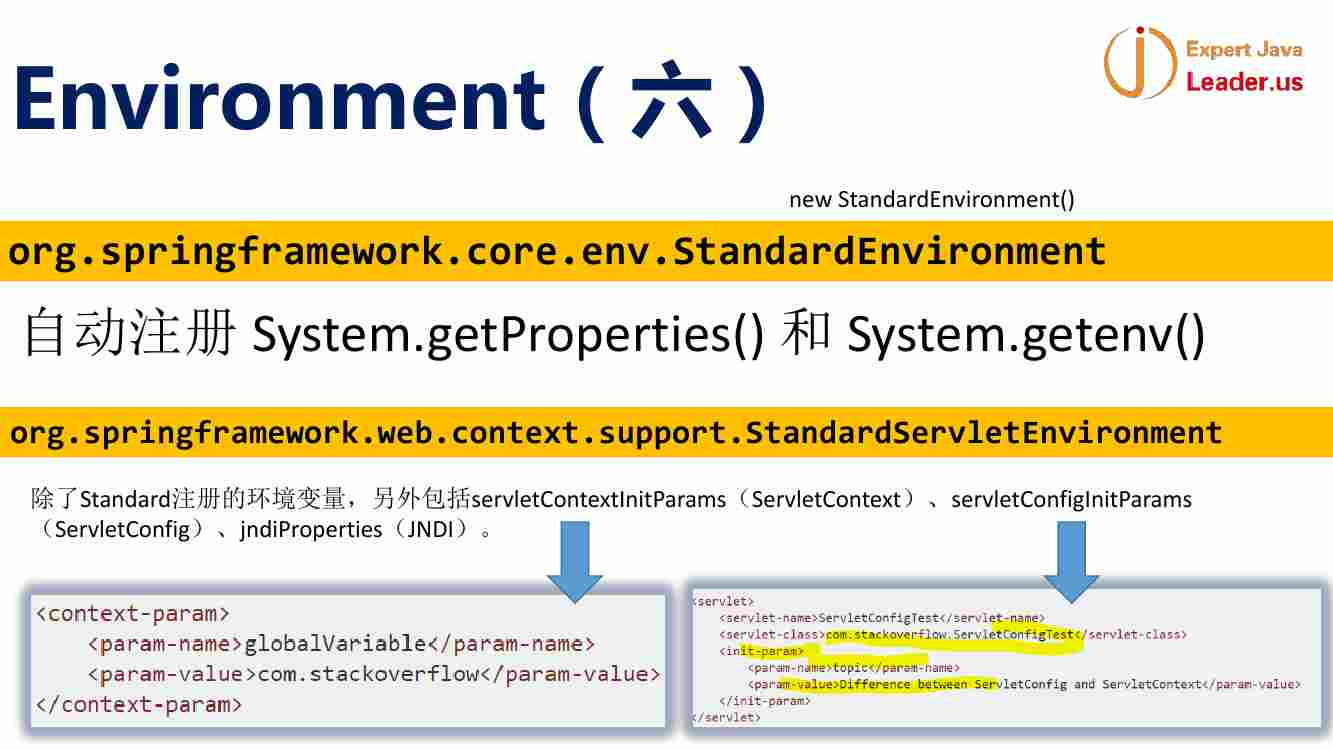
Docker有点特殊 Docker应用来说 运行期间是不能改变参数配置的 所以 Docker是系统 参数在应用中得到 也就是原来的classpath的properites中信息 等等 都需要转移到应用中去 ----- 但是 对于spring 无论是哪一种方式 都是env.getproperty来获取 这个是spring的好处

有了Enviroment对象 就有上面的好处



从上面开始 分别是配置文件中（system.storefile） ----- 系统环境变量(java xxx –Dmydebug=true) ------- java环境变量(java.io.tmpdir) ------ JAVA\_HOME(自定义的环境变量)里面 还有就是刚刚在环境变量配置的新内容

看看Enviroment实现机制 ---- Environment的标准实现是core.env.StandardEnvironment 这个类



背后实现的机制

就是 其实是 **创建了一个StandardEnviroment ---- 一旦创建了 spring容器就会自动注入System.getProperties()和System.env() 这两个都是System提供的方法 Java自带 --- 系统的properties和应用都注入了**

【回去看一下Java的环境变量】【参看帖子 <http://blog.csdn.net/dslztx/article/details/50188251> 写的非常好】

在Web的情况下 增加了一个StandardServletEnvironment这个类 ---web情况下 有除了上面的两种额外的参数

\*\*\*\*\*web的时候 要有context-param 这个是SevletContext的全局配置 对于不同的Servlet的配置 就是 init-param

***如果是Web环境 加了一个StandardServletEnviroment 这个里面 需要机上图中的ServletContext和ServletConfig这两个配置的kv(web.xml)***

**但是这种用的不多**

**注意 在Java –d运行的时候 -d的参数=值 就是一种kv ---- -d就是为每个应用单独提供的参数 就可以使用context-param；来替代 【web和非web得时候 kv的不同的表现形式】**

【理解Servlet容器 这个在马哥linux中讲解的时候 也是一种容器 所以 这个上面文 context 同样也是容器  **所以 <context-param> 定义的就是容器的参数 容器的参数不就是全局使用的么？ 所以 这个参数可以被Servlet容器共享**

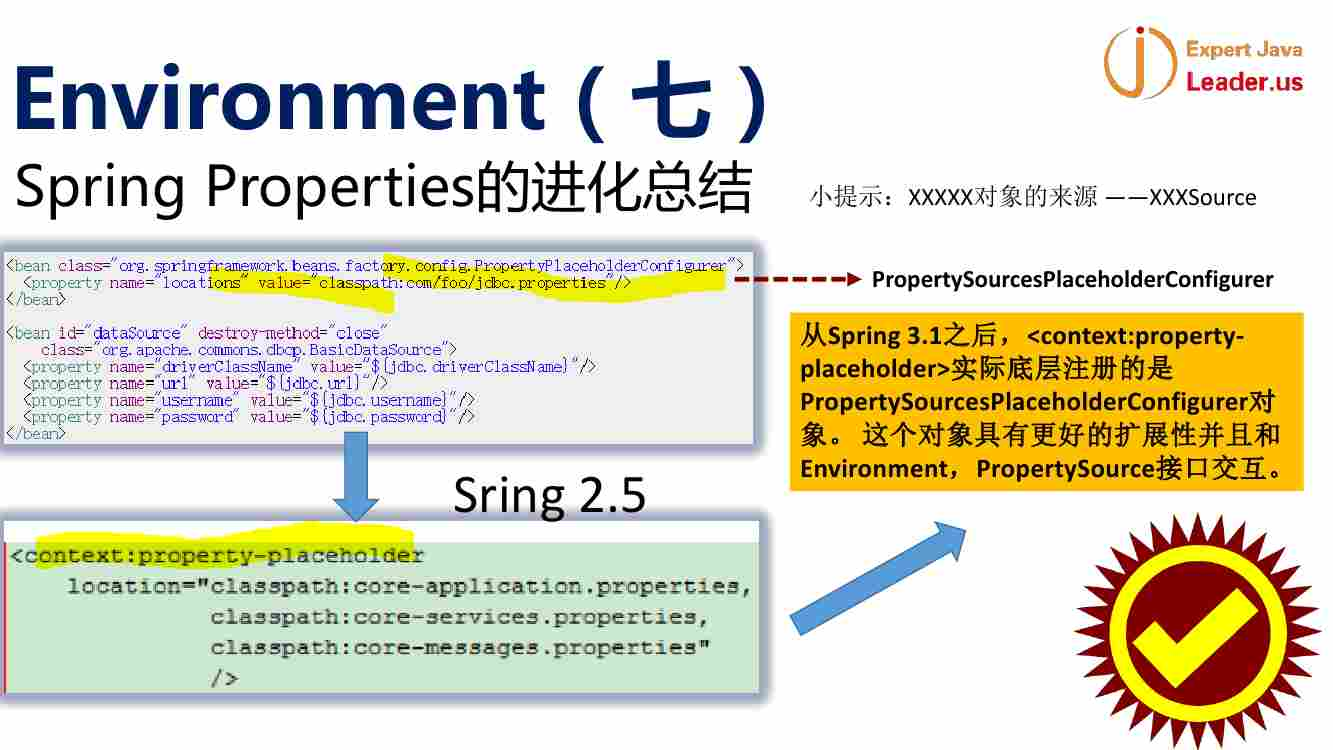
还有就是某一个Servlet的中<init-param>中定义的参数

注意  **ServletContext ---- 看成Servlet容器** 那么 容器中就有组件 管理组件 创建组件 初始化组件 ----- ServletContext这个容器中 组件就有 **Servlet Filter等等 Spring的ApplicationContext中 组件就是Bean**

====== 所以 Servlet Struts中的Context都可以看成容器 ------- 是容器 里面就有很多组件

】

这些参数还是有用的 大多数参数都是在配置文件 或者数据库中 所以 这种用的少



Spring Properties这个进化

最早的时候 加载properties这个文件 是配置在配置文件 以<bean>这种标签 进行配置的类是 --- 以Bean的方式去定义

Class.springframework.beans.factory.config.PropertyPlaceholderConfigurer 这个配置 -----

之后 **到了2.5就是使用前缀标签**

**<context:property-placeholder>**

再往后 **就到了3.1** 变成了<context:property-placeholder> 底层是PropertySource**s**PlaceholderConfigure [多了一个复数] 从此之后 Properties的读就是使用Environment进行读取配置中的信息

**这种配置完成 我们以后都是通过Enviroment来进行属性的访问 【目前进化的版本】**

这种方式还是被动的读取内容 ---- [Spring3.1提供了@Value](mailto:Spring3.1提供了@Value)注解更加方便的读取Environment变量中的内容

 ========这个是Properties读取最终的进化的版本

如果使用Enviroment 的方式获取属性值 还是使用起来很麻烦 所以 [在Spring3.0就引入了@value](mailto:在Spring3.0就引入了@value)

**这个 env.getProperty(“xxxx”)就等价于 @Value**

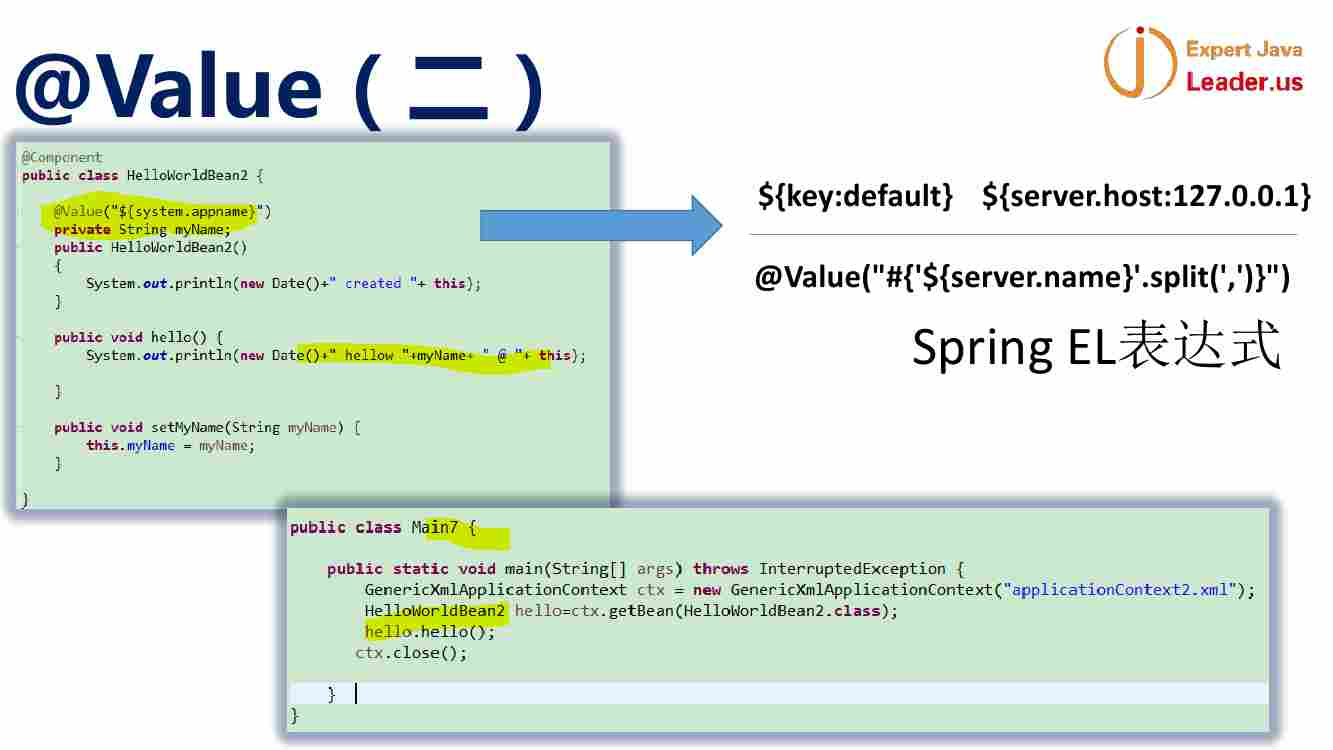
这样 **直接 @Value(“”)就替代了前面的代码部分 使用起来很简洁 ------ @Value(“${键名}”)**

**这样不再其引入Environment 而直接使用@Value这个注解 ===========**

【properties进化的过程：2.5 xml+PropertyPlaceHolderConfigurer <property name=xxxx value=”${xxxxx}”> 读取

🡪 <context:property-placeholder locations=””> 读取 底层是Property***Sources***PlaceHolderConfigurer 对应的注解方式 就是@Configuration + @PropertySource读取properties中的内容 并配合使用Environment这个类对象进行读取

* 最后演化到 使用 @Value替代Enviroment进行Properties的读取 在Value中 又恢复了使用最原始${}表达式的使用 】

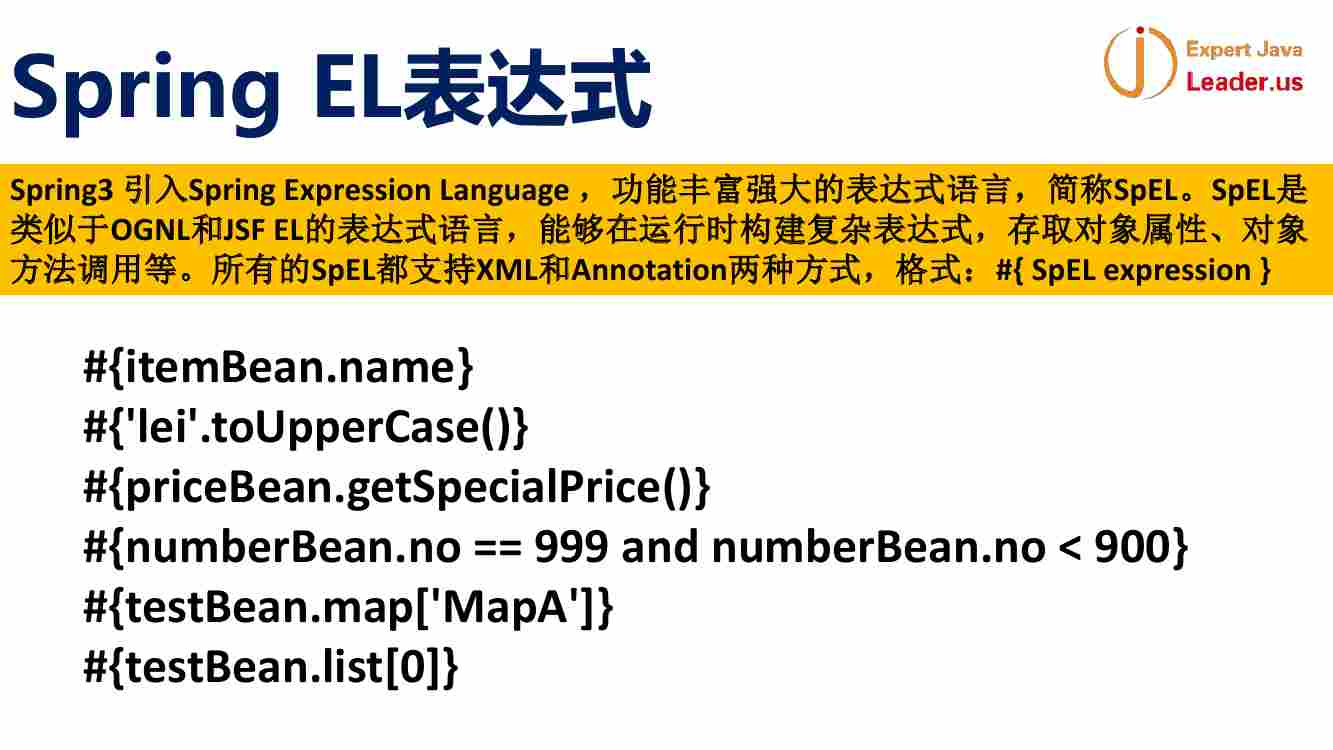


@Value中的表达式 还可以这样写

默认值的方式：${key:default}这种表达式 ------ 就是 如果key的值找不到 就可以使用default的值

上面的如果servere.host没有 就获取后面的值 这样程序中就不用判断

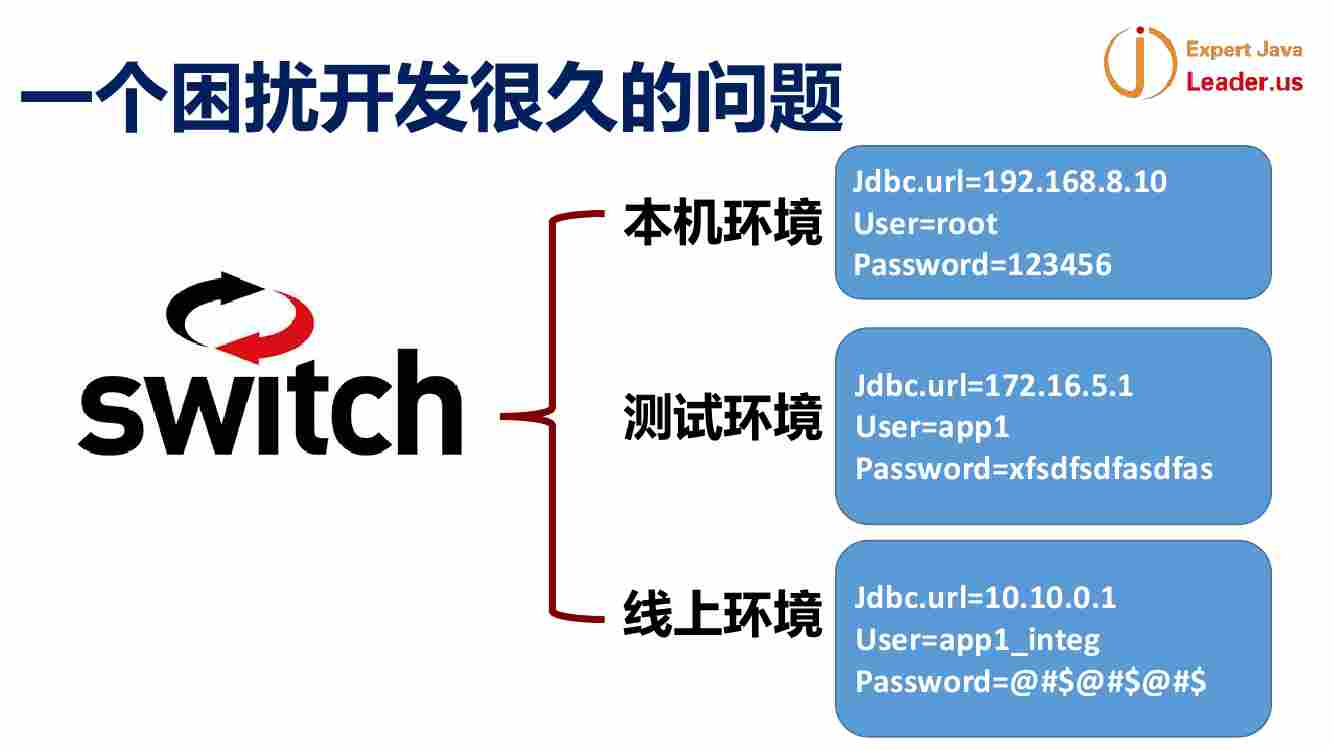
**还有更加复杂的 @Value注解支持 Spring EL表达式**



Spring EL简单灵活 --- OGNL简单灵活 可以做很多的计算 还可以做简单地判断

表达式灵活 ***如果我们的配置这么复杂 很难看 所以 选择性的时候 可以使用 需要 是可以这么做的***

------------------------------------- 配置的升级 场景 --------------------------------------- 前面是Spring中如何读取配置的方式

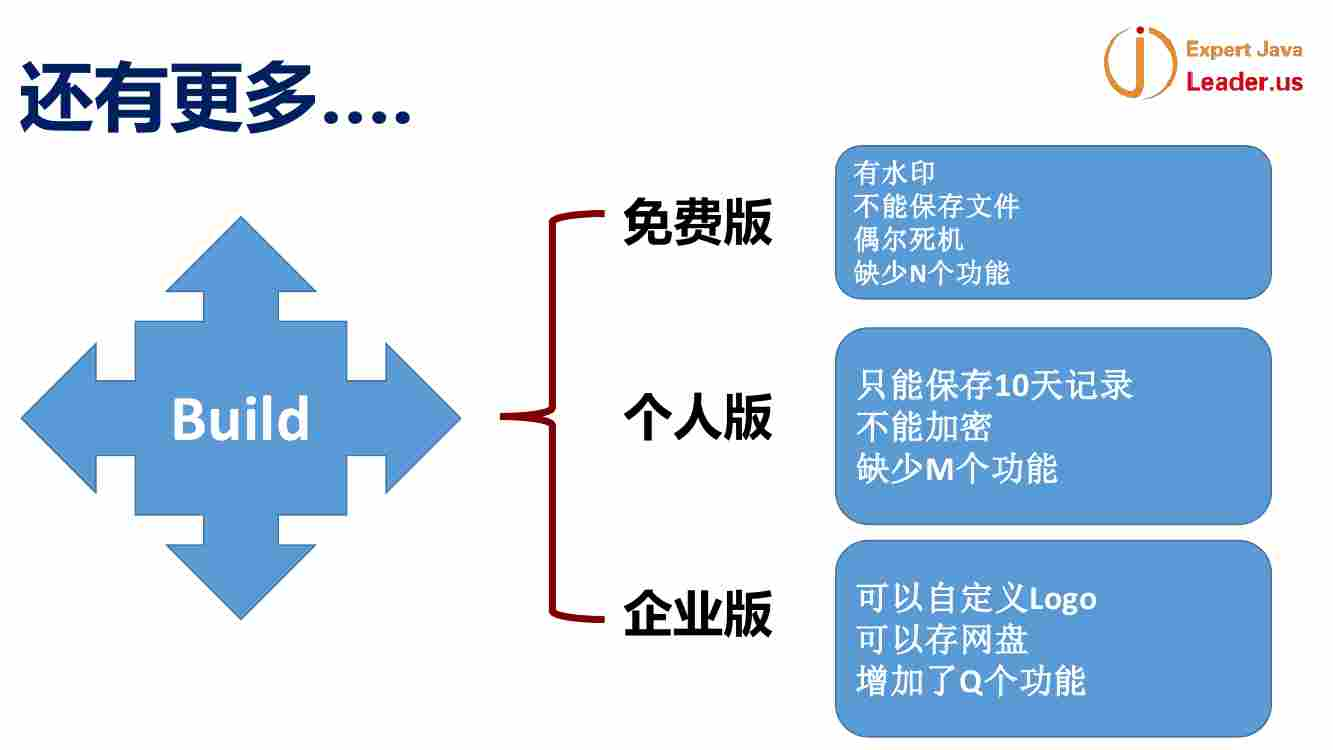
 **------ 表示环境变量不一样**

Spring解决了这样的场景

也就是 本机环境 测试环境 线上环境

**怎么切换？**

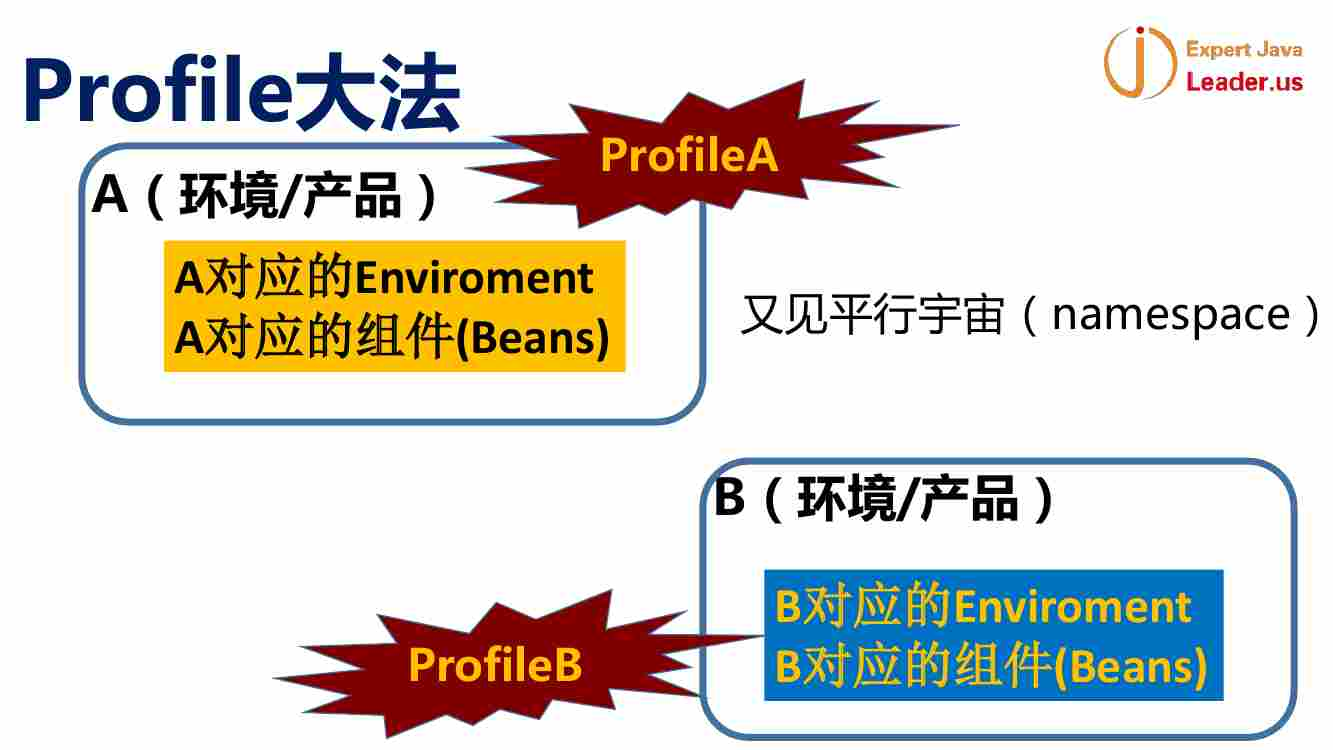
可能切换错误 把本机的传递上去 ---- 一般都是配置文件 改来改去 --- **非常头疼**

 **------ 表示组件的版本有差别**

同一个软件build出来 功能是有差异的 不同的版本 这样怎么办？ --- 不同的版本 功能有差距 ---个别的组件有区别 ---这种怎么办

上面两个结合起来 Spring就出现了Profile

**这两个结合在一起 Spring出来了一个Profile的功能**



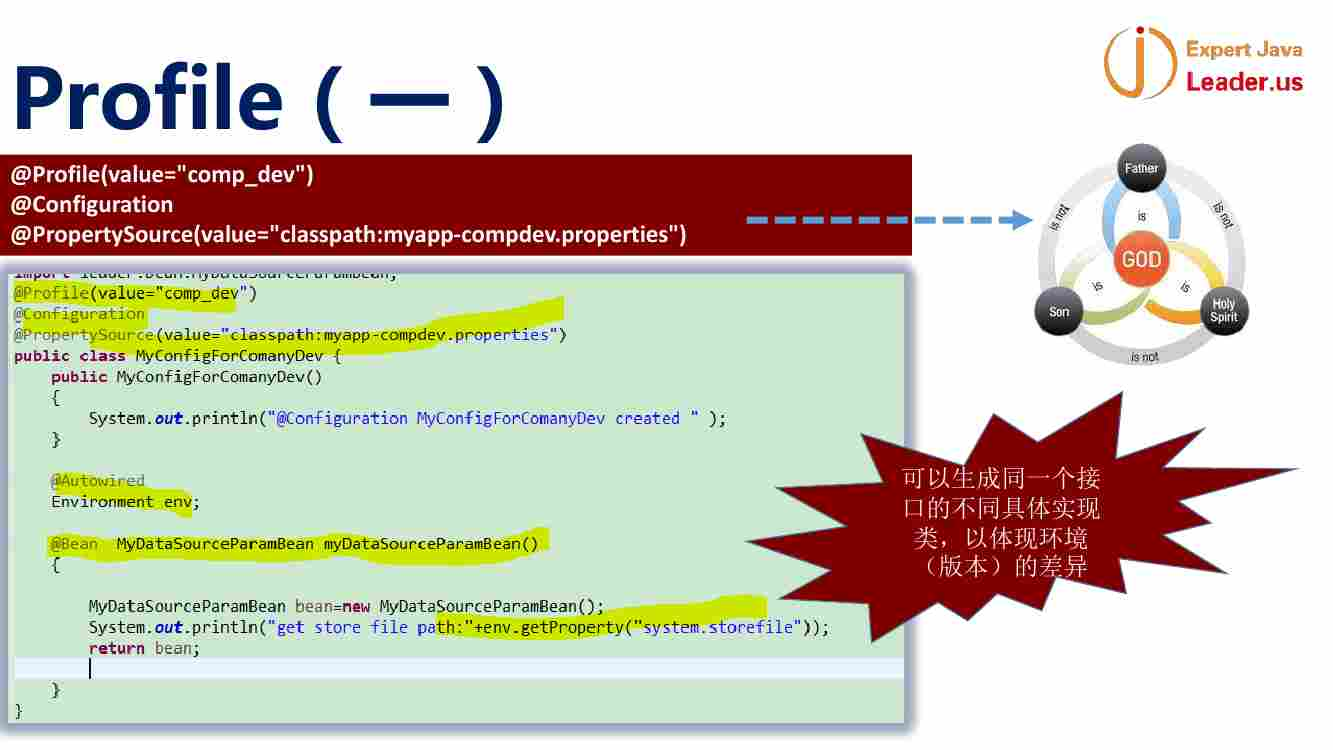
**一组的Enviroment**和**一组Bean** ***构成一个Profile***  【包在一起】

**这两个Profiles是可以同时存在的 每一个是Profile A Profile B---- 虽然都存在 但是选择加载一个而已 根据参数来进行控制**

不是互斥的 不是修改来修改去

这样 我们可以根据我们的参数 来控制 到底加载哪一个 哪一个生效了

环境变量不一样的 + 对应的bean有差别 定义成Profile



Profile用法不简单 和配置有单

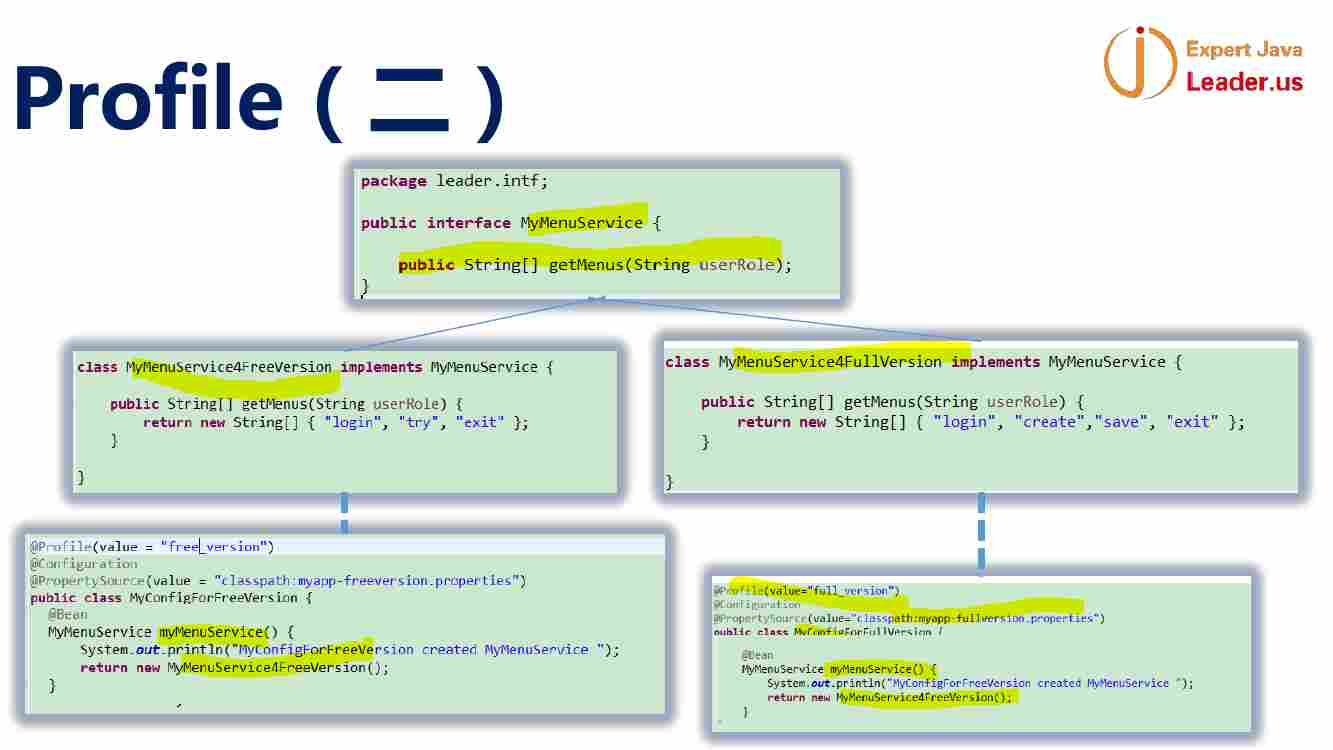
**Profile和@Configuration @PropertySource关联在一起 关联某一组的Enviroment 这个是** 三位一体的概念 --- Profile不能简单认为是标签 是Configuration 对应的某些Property 以及特定的bean对象 整体称为一个Profile

@Profile不能简单理解成注解 要看成 某个@Configuration下面 使用@PropertySource关联的配置文件 下 定义的@Bean的组件

这个整体成为一个Profile 给个名称

**意味着 可以做一些设计和改变的**

**对同一个接口 设计不同的类 ---- 在不同的版本中 功能是不一样的**



返回子菜单 有一个free版本 login try exit功能

还有一个full版本的功能 login try save exit四个功能 同一个接口实现两种不同的bean

这就是同一个接口实现的不同的bean

定义一个Profile

**有一个Bean ----- 这个里面返回的就是Free版本的bean 返回类型还是接口**

那边的full版本也是这样的 **右边的写错了 应该是 return new MyMenuService4FullVersion();[PPT写错了] --- 这两个不同的profile 对应的properties是不一样的 开启不同的profile 就是两个不同的产品**

**【这样 在一个Configuration中 有Profile 和 PropertySource这两个注解之后 相当于独立的xml两部分 一部分是教free\_version.xml 另一部分教full\_version.xml 这两个Configuration有不同的@Bean对应的实现类 就是完全不同的两个配置 打上Profile烙印了 然后 对不同的配置进行激活】**

 main方法中

**现在激活这两个profile** ----- System.setProperty(“**spring.profile.active**”, “xxxx”) 依靠这个属性来激活 --------- **可以激活多个Profiles --- 激活的属性 就是 spring.profiles.active和spring.profiles.defualt 这样设置完profile之后 就可以知道那个是被激活的 哪个是抑制的**

**Main中使用了@Configuration +xml的方式进行测试的 ---- 这样获取MyMenuService的时候 找到的是不同的接口的实现类 因为Configuration是不一样的**

---- 这两个是设置了两个不同的属性放到了System环境变量中 这样 Spring启动后 PropertySourcesPlaceHolder就会读取这些内容 然后 Environment变量就可以读取这些环境变量中的内容 ---- 下面的代码就是env.getDefaultProfiles(); env.getActiveProfiles() 这两个方法 直接返回

设置好之后 Spring会获取到这个属性

这种做法 在测试的时候 可以用这个

**没有active的 这里面就设置称为default 也就是spring.profiles.defualt**

**Profile是Spring中的一个特性 设置完这两个prifile的激活版本之后 ----- 这个是Spring中的一个非常重要的特性**



后面讲的是如何装配Component依赖的问题