

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



Properties在classpath中 属于代码的一部分

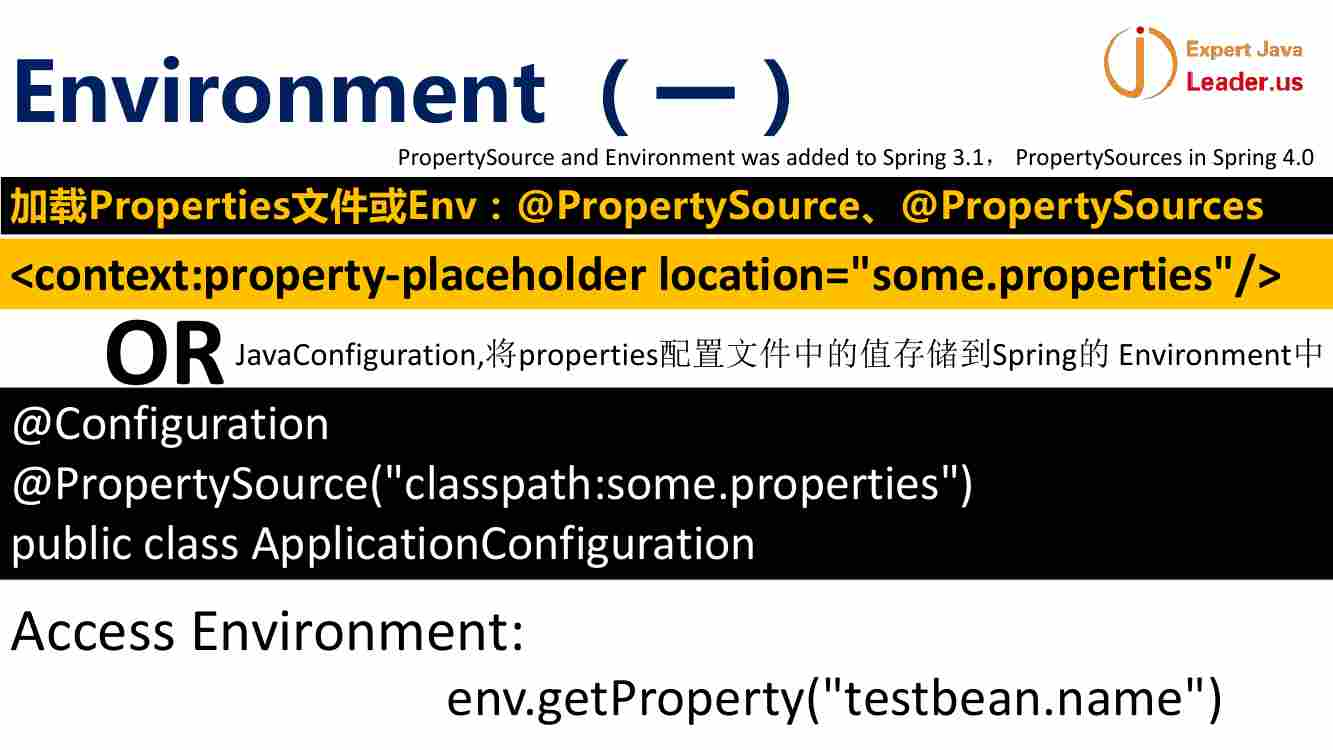
系统环境变量 里面 包括系统的Properties和jvm自身的Properties

用户的home目录 临时文件目录

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

J222 context变量

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

在4.0的时候 支持了PropertySourc**es【可以用多个】**

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

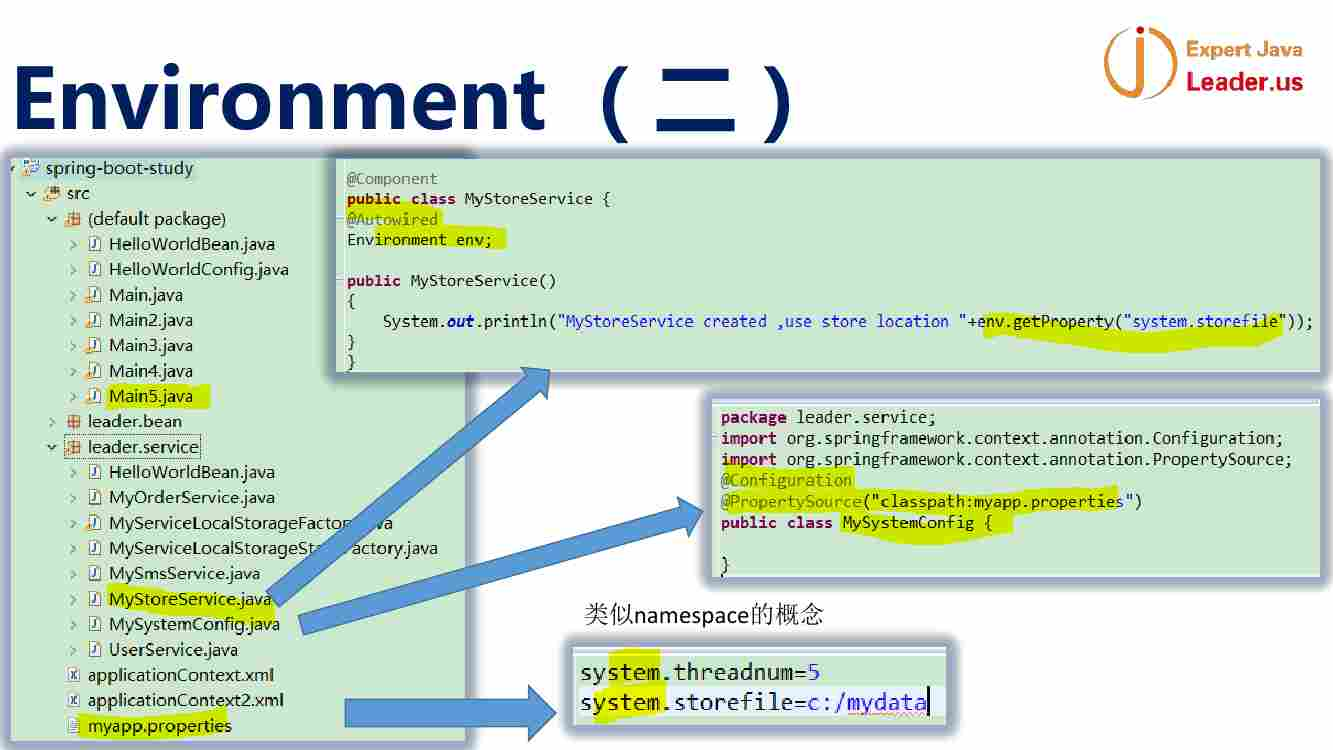
@Configuration + @PropertySource(“classpath:xxxxx”)

@PropertySource就会把配置文件中的内容加载到Enviroment 这个变量中去

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

一般为Key加上前缀



Myapp.properties属于源码的一部分 所以在classpath下

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”)

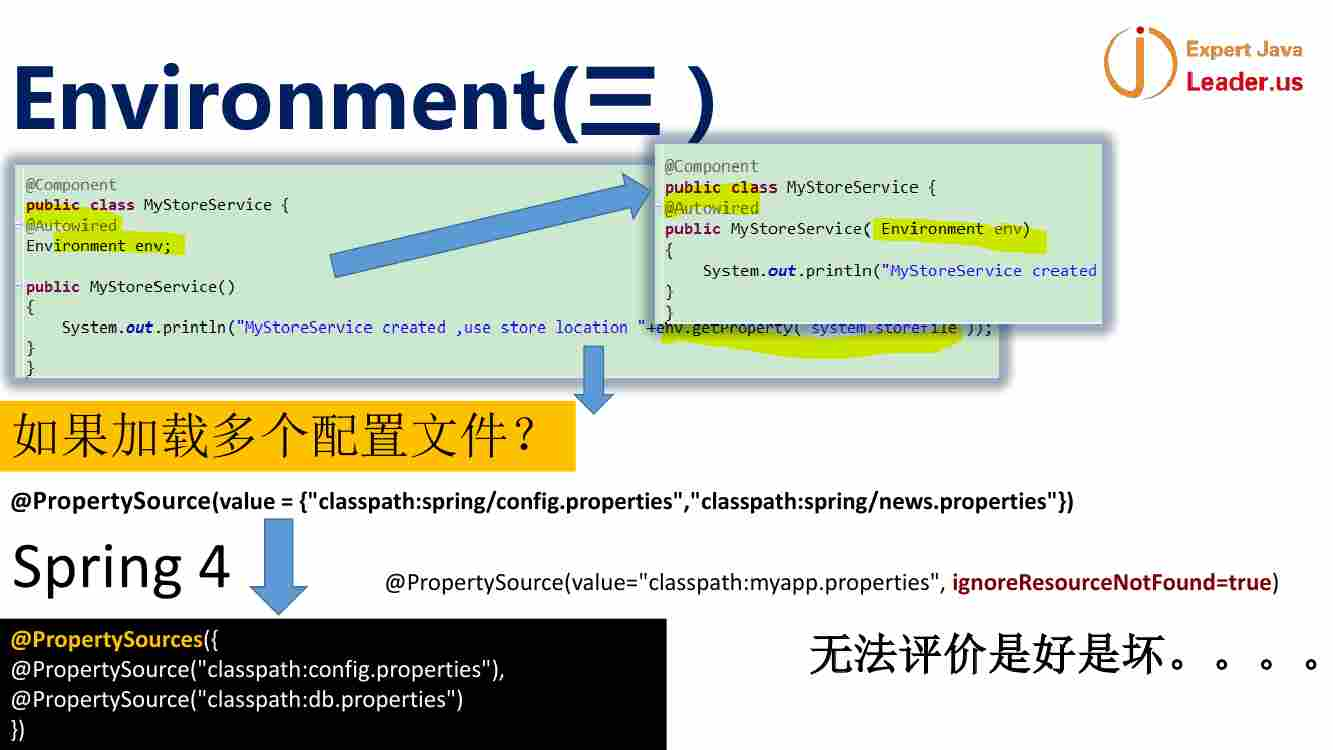
Public class MySystemConfig{ //这个类叫MySystem 所以 建议给myapp.properties的key加上一个前缀 system 之后

}

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

【Enviroment变量是Spring容器托管的 直接注入就可以使用】



加载多个 在spring3中可以使用数组模式

@PropertySource(value={xxx, xxxx})

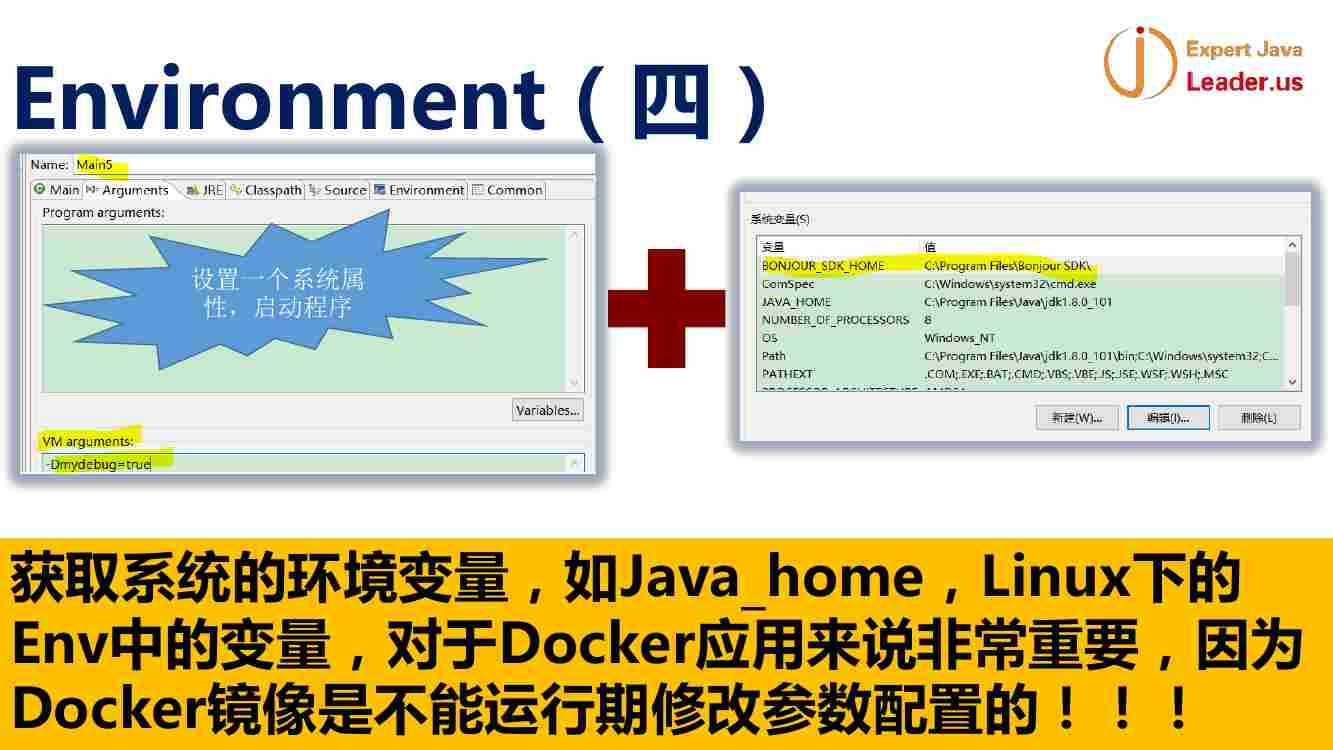
到了Spring4中 添加了@PropertySources({@PropertySource{xxxx}, @PropertySource{yyyyy}})

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

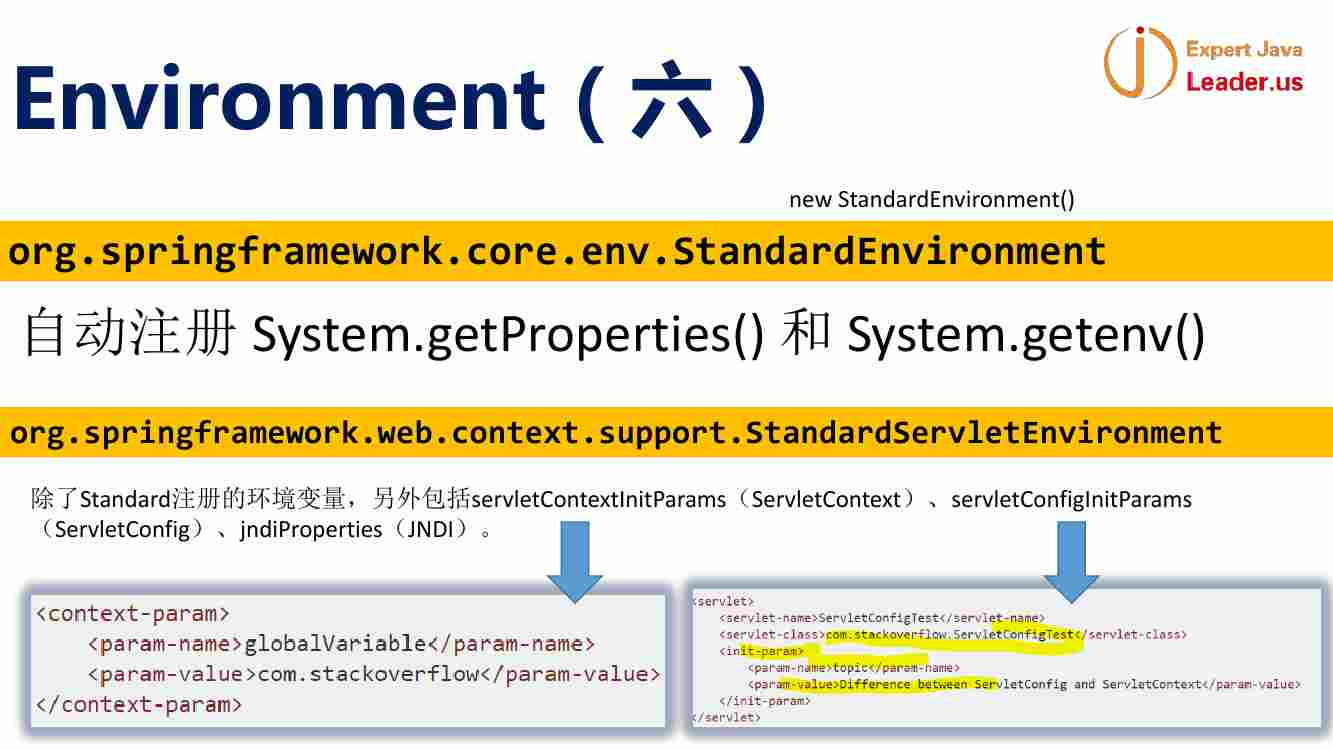


这个spring中的Enviroment对象 既可以获取一个在Eclipse调试阶段的配置的参数（左图的KV） 还可以获取一个类似于右边的windows或者linux下面的配置的kv ----- 所以PPT上面说 Java\_home Linux下面的环境变量 都可以获取到 ---- 好处就是 无论在哪里 访问方式都是一样的

Docker有点特殊（没听清）



从上面开始 分别是配置文件中 系统环境变量 java环境变量 JAVA\_HOME里面 还有就是刚刚在环境变量配置的新内容



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()

【回去看一下Java的环境变量】

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 ---- -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

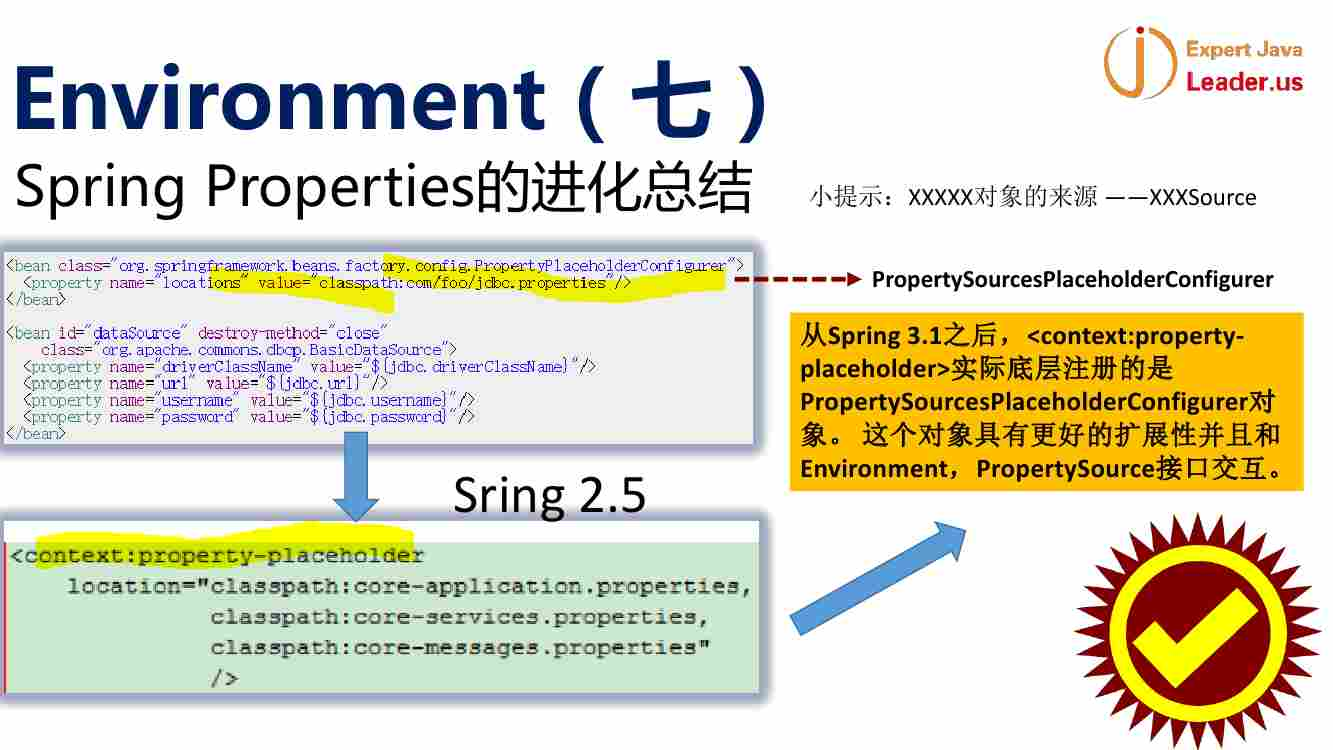
【理解Servlet容器 这个在马哥linux中讲解的时候 也是一种容器 所以 这个上面文 context 同样也是容器 所以 <context-param> 定义的就是容器的参数 容器的参数不就是全局使用的么？ 所以 这个参数可以被Servlet容器共享

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

】



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

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]

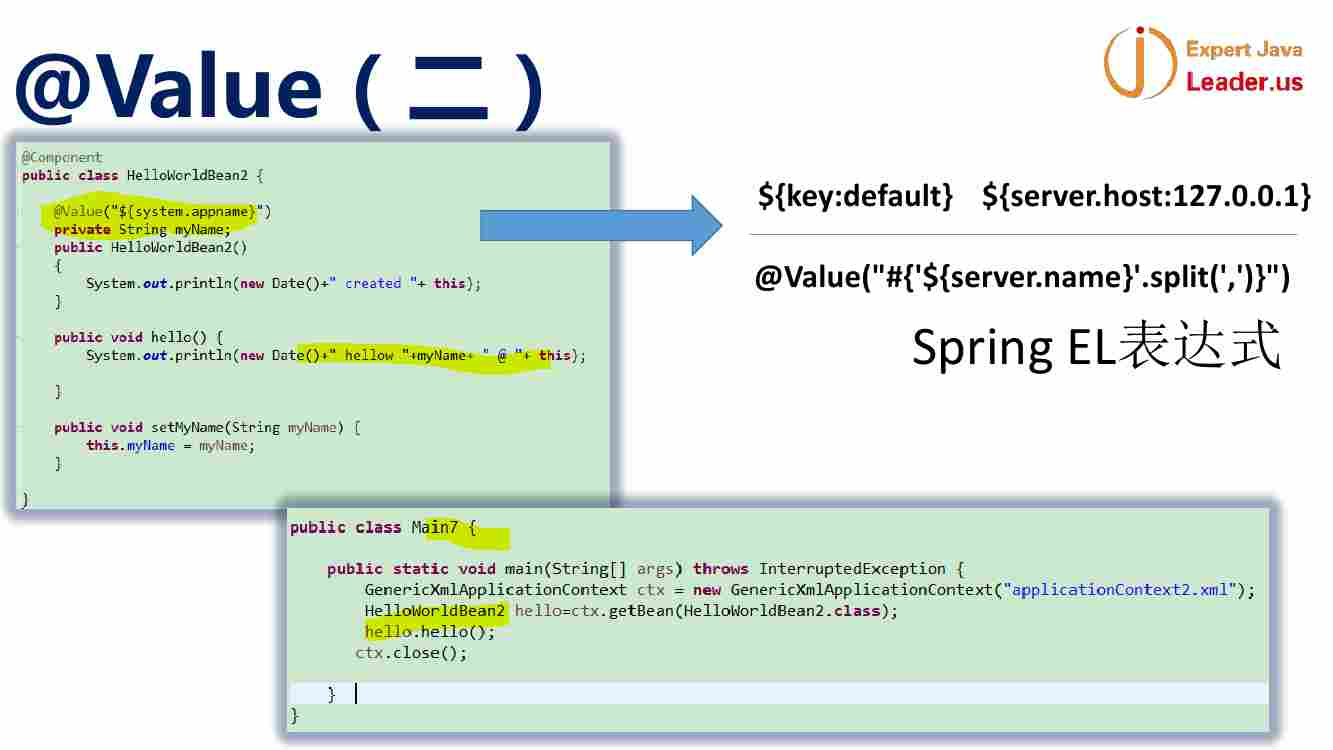
这种配置完成 我们以后都是通过Enviroment来进行属性的访问



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}”)

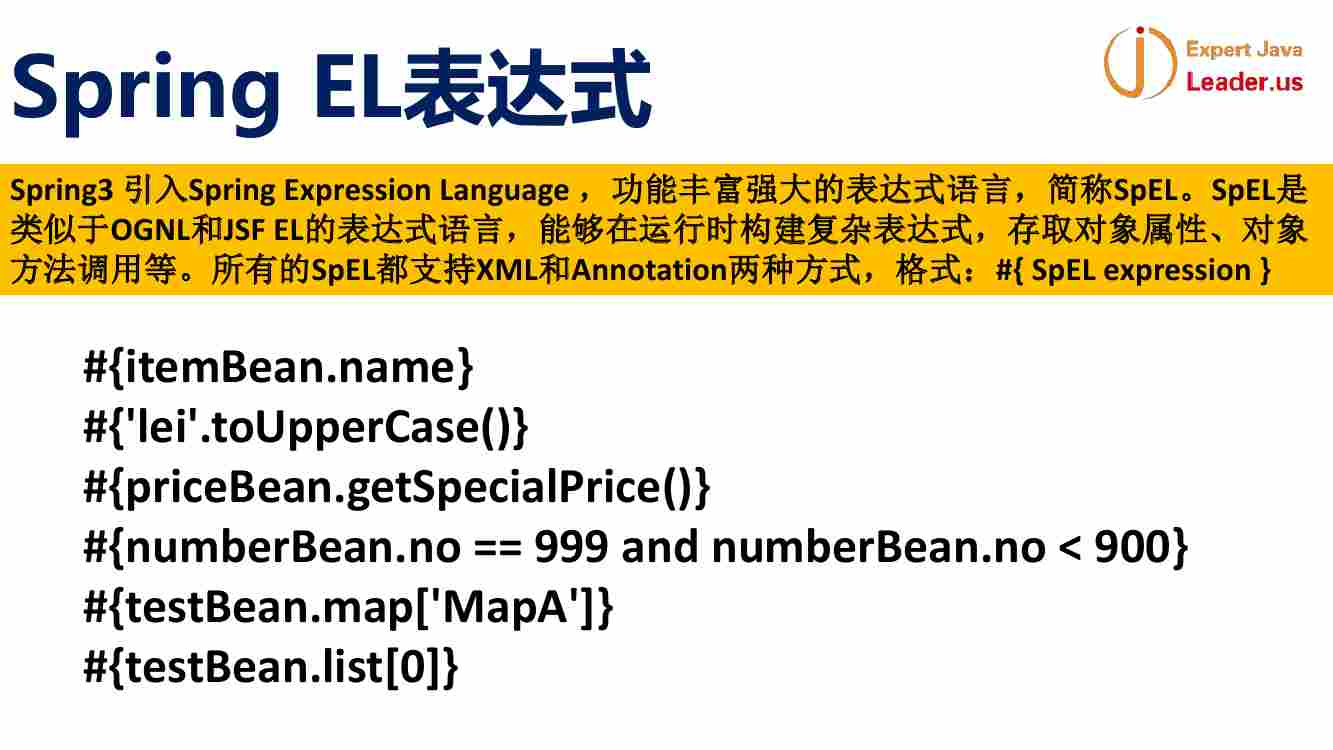


@Value中的表达式 还可以这样写

${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

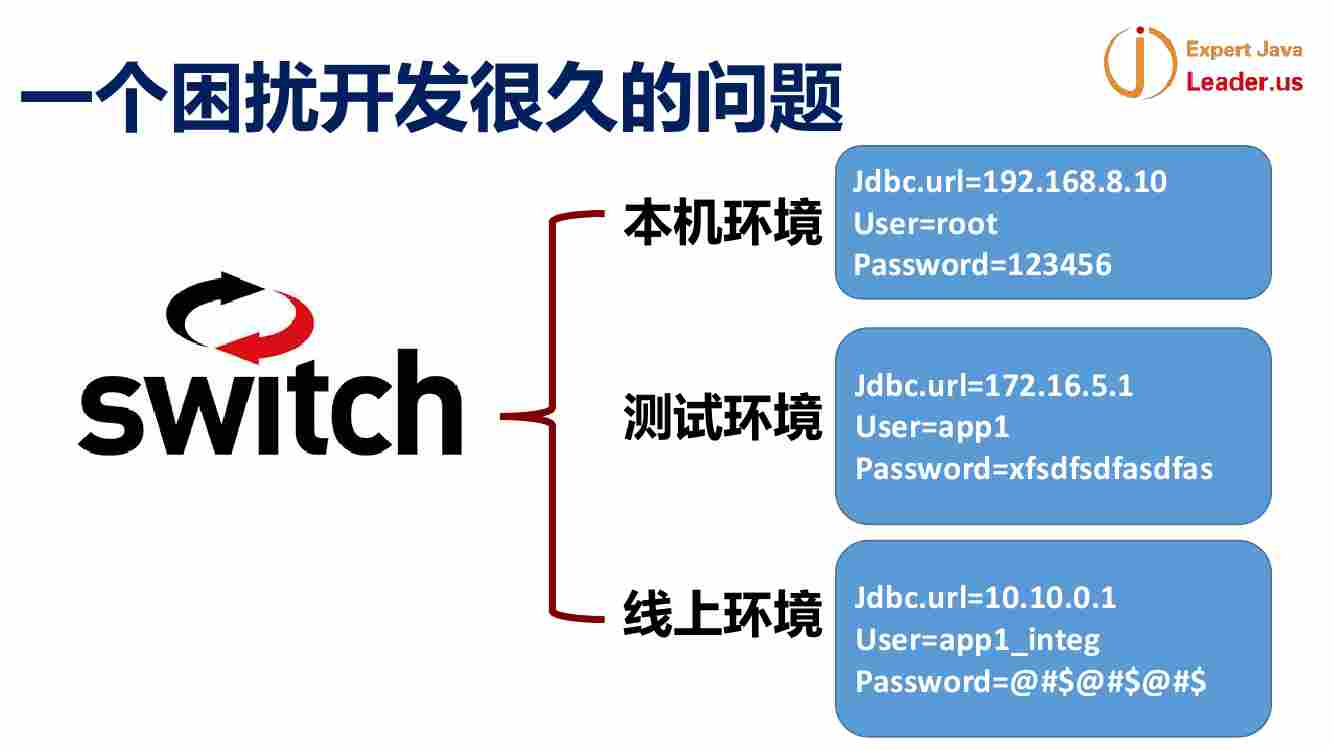
上面的如果servere.host没有 就获取后面的值 这样程序中就不用判断

还有更加复杂的 Spring EL



Spring EL简单灵活

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

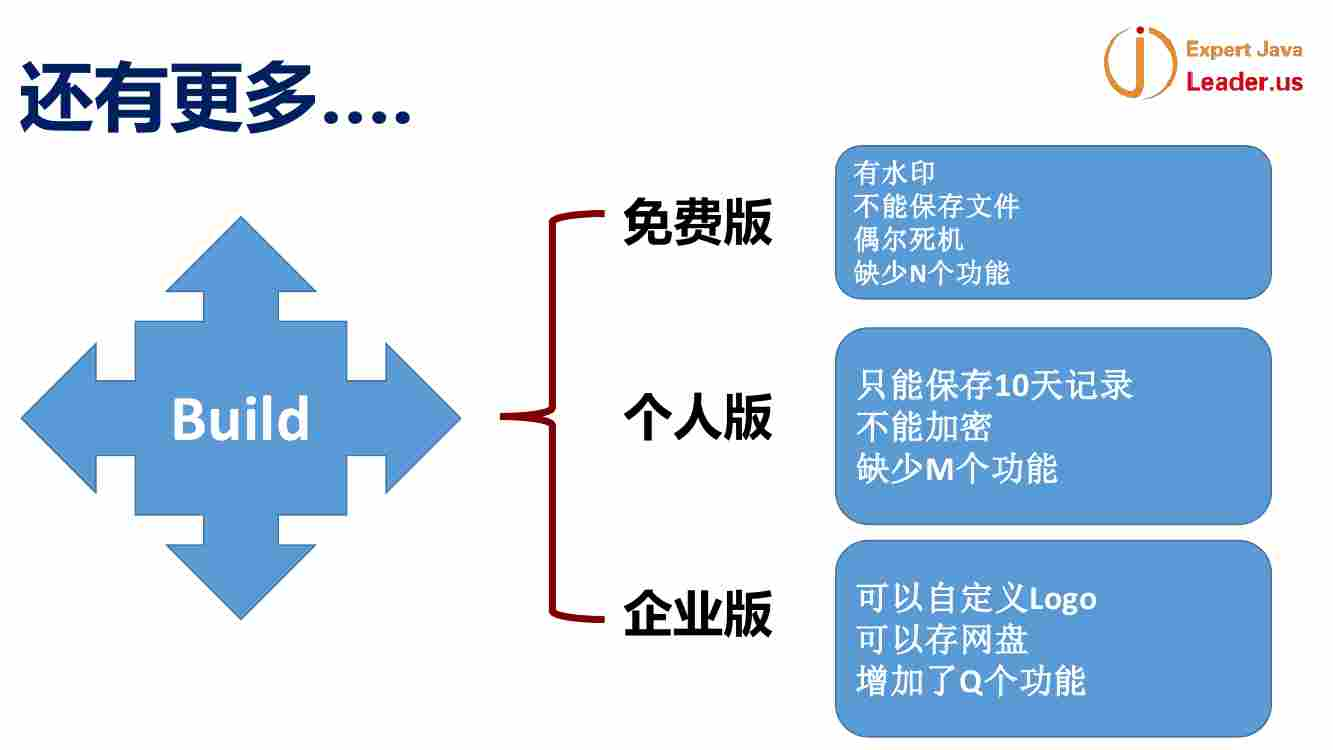


Spring解决了这样的场景

也就是 本机环境 测试环境 线上环境

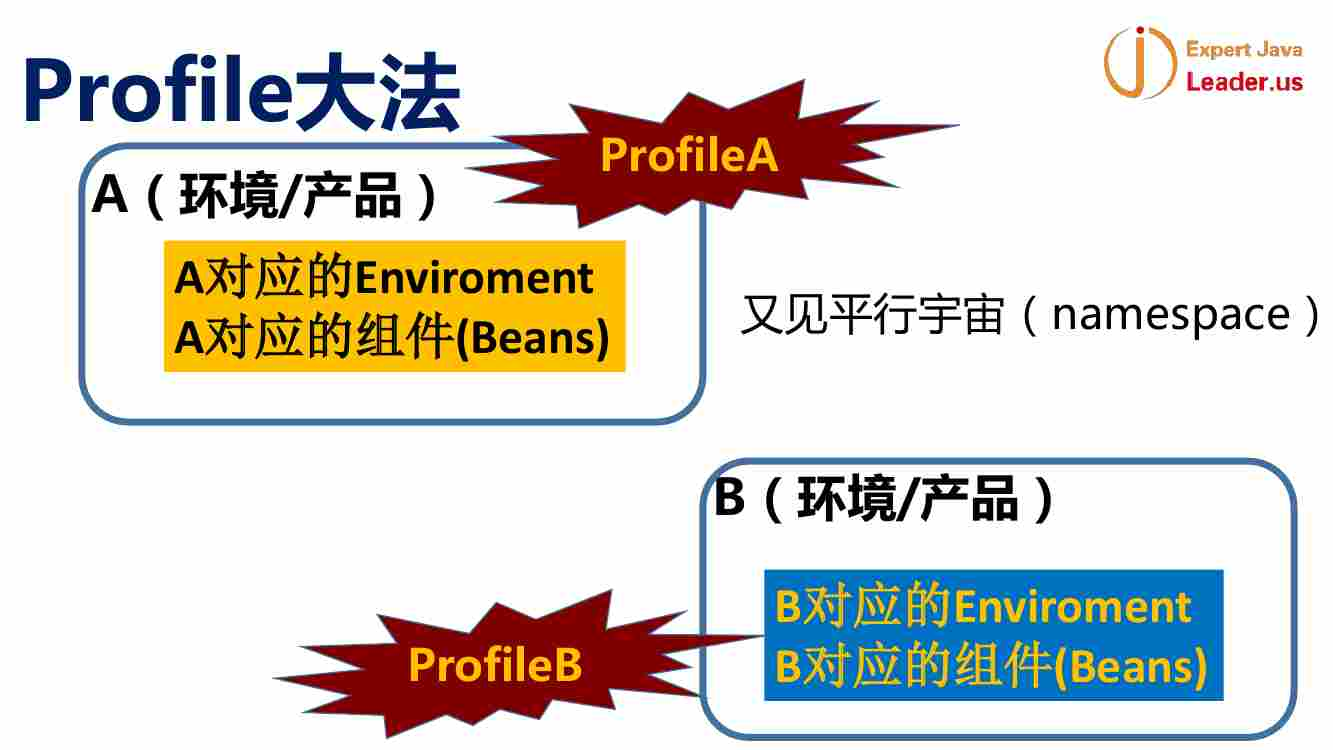
怎么切换？

可能切换错误 把本机的传递上去



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？

这两个结合在一起 Spring出来了一个Profile的功能



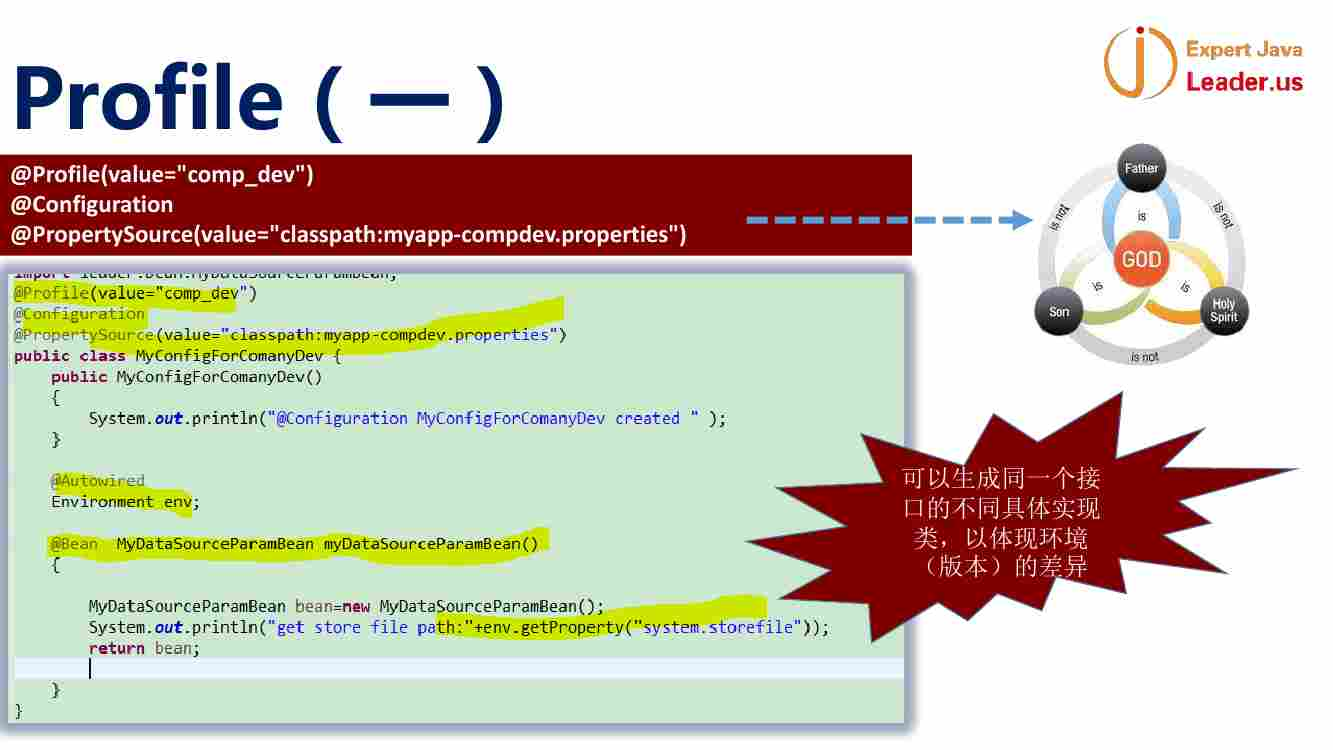
一组的Enviroment和一组Bean 构成一个Profile

这两个Profile是可以同时存在的

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



Profile用法不简单 和配置有单

Profile和@Configuration @PropertySource关联在一起

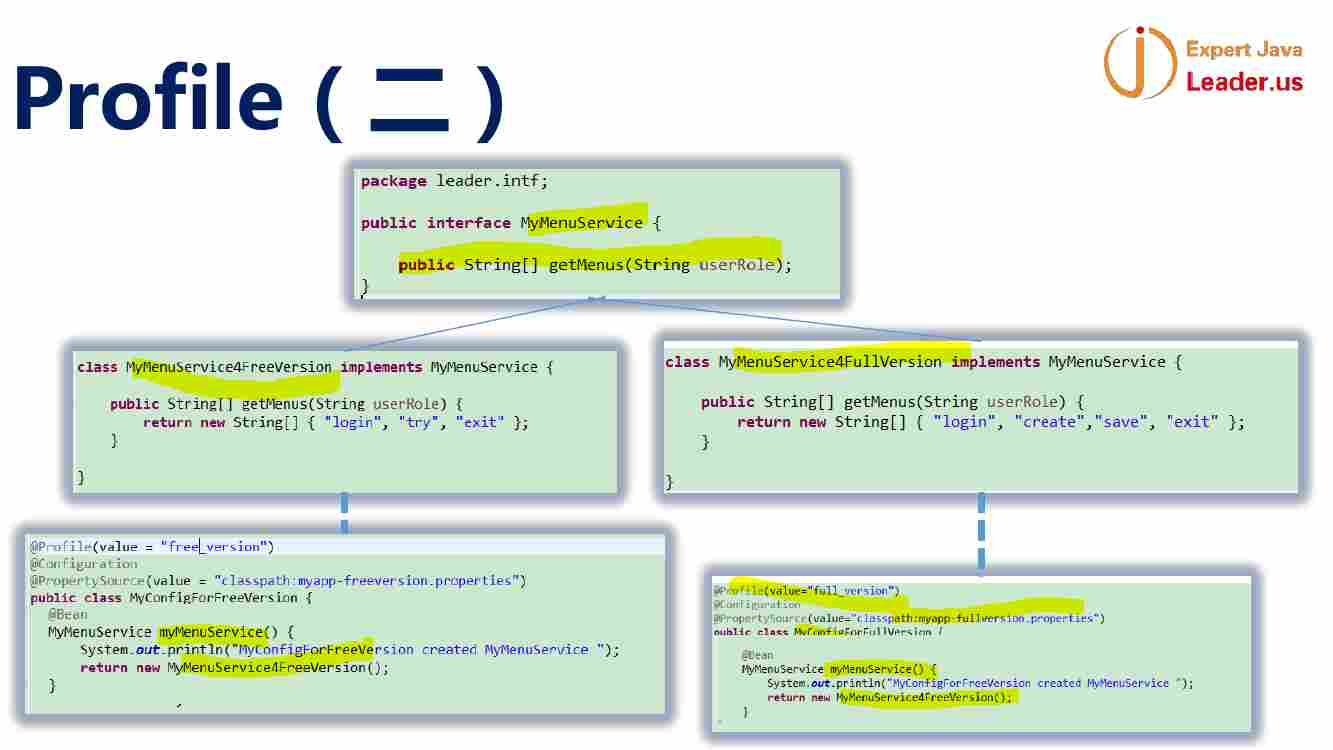
三位一体的概念

@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 功能室不一样的



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

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();



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Profile

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

Profile是Spring中的一个特性



后面讲的是如何装配Component依赖的问题