一种基于Spring的java程序常量管理思路Java认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/644/2021_2022__E4_B8_80_E7_A7_8D_E5_9F_BA_E4_c104_644774.htm 在编写程序的时候，总是不可避免的需要使用一些常量，甚至很多的常量。我们可以对常量进行一个很简单的分类： 记忆性常量：主要出于程序结构上的考虑而设定的常量。譬如为了避免一个没有字面意思的魔法数，或者避免拼写容易出错，或者不容记住的内容。 业务性常量：表示一个业务上的一个特定业务实体的属性或属性值。 很多的时候，一个业务性常量很多时候也是一个记忆性常量。 在一个大型项目中，参与的人员和代码数量通常都会比较多，没有好的管理策略，常量的使用往往想入混乱中。譬如重复定义，其维护的值甚至还不一致，以外覆盖；譬如仅仅为了使用某个常量，而引入某个包或者类，由此可能引出模块间的循环依赖等。 良好的设计结构，以及严格的开发纪律基本上可以解决上述问题。除了有时的确是不可避免的出现以上问题外，有时一些所谓的业务常量只有在部署期间或同别的系统集成是才能获得。所以有必要进一步的探讨常量的管理手段。 记得在JavaEye上看到一篇关于Spring属性注入的文章，灵机一动，不是恰好可以用来处理这个问题吗？所谓属性注入，意思是指将配置信息写在Properties文件中，通过IOC容器透明的注入。 这么设想下，如果常量最终都可以用配置文件配置，那么就可以解决“业务常量只有在部署期间或同别的系统集成是才能获得”的问题，如果同时还可以透明的宣称使用常量，那么几乎所有的问题就完美了： 不害怕重复定义错误 消除有常量引用引起

的循环引用 提供从部署期覆盖编译期的灵活性 使用Spring的扩展名称空间和Java5的Annotation语法，我们可以整理出以以下思路 定义一个Annotation类 实现一个Annotation的Processor 配置Processor Bean 实现过程大致如下： 一、Annotation @Target({ElementType.FIELD,ElementType.TYPE}) @Retention(RetentionPolicy.RUNTIME) public @interface Properties { String name() default "". String value() default "". String namePrefix() default "". String namePostfix() default "". } 二、Processor public class AnnotationPropertiesBeanPostProcessor extends PropertyPlaceholderConfigurer implements BeanPostProcessor, InitializingBean { private java.util.Properties pros. private String namePrefix = "". private String namePostfix = "". public void setEnabledClassList(Class[] enabledClassList) { this.enabledClassList = enabledClassList. } public Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException { return bean. } @Override public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException { HandlePropertiesAnnotatedBean(bean, bean.getClass()). return bean. } 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com