设计模式:可更新的注册式的单实例模式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_ti2020/571/2021_2022__E8_AE_BE_E8_AE_A1_E6_A8_A1_E5_c104_571347.htm 遇到这样一个应用。在系统中需要大量的配置信息，为了不每次都找数据库或者配置文件。需要一个生命周期和App一样的容器（=静态变量），但是在配置信息被修改时还需要去更新这个容器。 首先选用的是单实例模式。单实例模式中又可分为恶汉，懒汉，以及一种基于饿汉型的注册型。 个人感觉懒汉型单例模式没什么，而饿汉型的更能体现java特点。然注册行的可扩展性较强，个人感觉有点像 一个实例工厂。下面来一一列举。 恶汉： Java代码 public class EagerSingleton { private static final EagerSingleton m_instance = new EagerSingleton(). private EagerSingleton() { } public static EagerSingleton getInstance() { return m_instance. } } 懒汉： Java代码 public class LazySingleton { private static LazySingleton m_instance = null. private LazySingleton() { } synchronized public static LazySingleton getInstance() { if (m_instance == null) { m_instance = new LazySingleton(). } return m_instance. } } 注册型： Java代码 public class RegSingleton { static private HashMap m_registry = new HashMap(). static { RegSingleton x = new RegSingleton(). m_registry.put(x.getClass().getName(), x). } protected RegSingleton() { } static public RegSingleton getInstance(String name) { if (name == null) { name = "name". } if (m_registry.get(name) == null) { try { m_registry.put(name, Class.forName(name).newInstance()). } catch (Exception e) {

System.out.println("Error happened."). } } return (RegSingleton) (m_registry.get(name)). } } Java代码 public class RegSingletonChild extends RegSingleton { private RegSingletonChild() { } /** * 静态工厂方法 */ 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