Spring工作原理探秘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_ti2020/620/2021_2022_Spring_E5_B7_A5_E4_c104_620984.htm Spring的哲学是在不影响Java对象的设计的情况下将Java对象加入到框架中。 我们下面来看看Spring的工作原理，看看Spring是如何做到不影响Java对象的。 EJB的框架采用了一种侵略性(Invasive)的方法来设计对象，它要求你在设计中加入符合EJB规范的代码。一些轻量级的COP框架，例如Avalon，也要求对象设计时必须符合某种规范，例如Serviceable接口，这种做法是典型的Type 1做法。这种设计思路要求Spring采用一种动态的、灵活的方式java认证，加入收藏！来设计框架。在Spring的工作原理中大量采用了反射。首先Spring要解决的一个问题就是如何管理bean。因为IOC的思想要求bean之间不能够直接调用，而应该采用一种被动的方式进行协作。所以bean的管理是Spring工作原理中的核心部分。 反射和内省在代码的层次上思考问题，有时候能够带来出人意料的灵活性。但它的使用有时候也是一个哲学问题，不论是在ORM设计还是在AOP设计上都出现了类似的问题-究竟是使用反射，还是使用代码生成。 在Spring中，处理这个问题的核心是在org.springframework.beans包中。而其中最为核心的部分，则是 BeanWrapper。BeanWrapper，顾名思义，就是bean的包装器。所以，它的主要工作，就是对任何一个bean，进行属性(包括内嵌属性)的设置和方法的调用。在 BeanWrapper的默认实现类BeanWrapperImpl中，虽然代码较长，但完成的工作却是非常的集中的。 BeanWrapper的深入研究 我们看看这个BeanWrapper是如何发挥运作的，假设

我们有两个bean： 1. public class Company { 2. private String name. 3. private Employee managingDirector. 4. 5. public String getName() { 6. return this.name. 7. } 8. public void setName(String name) { 9. this.name = name. 10. } 11. public Employee getManagingDirector() { 12. return this.managingDirector. 13. } 14. public void setManagingDirector(Employee managingDirector) { 15. this.managingDirector = managingDirector. 16. } 17. } 18. 19. public class Employee { 20. private float salary. 21. 22. public float getSalary() { 23. return salary. 24. } 25. public void setSalary(float salary) { 26. this.salary = salary. 27. } 28. } 然后我们使用BeanWrapper来调用这两个bean： 1. Company c = new Company(). 2. BeanWrapper bwComp = BeanWrapperImpl(c). 3. // setting the company name... 4. bwComp.setPropertyValue("name", "Some Company Inc."). 5. // ... can also be done like this: 6. PropertyValue v = new PropertyValue("name", "Some Company Inc."). 7. bwComp.setPropertyValue(v). 8. 9. // ok, lets create the director and tie it to the company: 10. Employee jim = new Employee(). 11. BeanWrapper bwJim = BeanWrapperImpl(jim). 12. bwJim.setPropertyValue("name", "Jim Stravinsky"). 13. bwComp.setPropertyValue("managingDirector", jim). 14. 15. // retrieving the salary of the managingDirector through the company 16. Float salary = (Float)bwComp.getPropertyValue("managingDirector.salary"). 看起来麻烦了许多，但是这样Spring就可以使用统一的方式来管理bean的属性了。 Bean的制造工厂 有了对单个Bean的包装，

还需要对多个的bean进行管理。在spring中，把bean纳入到一个核心库中进行管理。bean的生产有两种方法：一种是一个bean产生多个实例，一种是一个bean只产生一个实例。如果对设计模式熟悉的话，我们就会想到，前者可以采用Prototype，后者可以采用Singleton。 注意到，反射技术的使用使得我们不再像原始的工厂方法模式那样创建对象。反射可以非常灵活的根据类的名称创建一个对象。所以spring只使用了Prototype和Singleton这两个基本的模式。 Spring正是这样处理的，但是我们希望用户能够维护统一的接口，而不需要关心当前的bean到底是Prototype产生的独立的 bean，还是Singleton产生的共享的bean。所以，在org.springframework.beans.factory包中的 BeanFactory定义了统一的getBean方法。 JDBC再封装JDBC优雅的封装了底层的数据库，但是JDBC仍然存在诸多的不变。你需要编写大量的代码来完成CRUD操作，而且，JDBC 无论是遇到什么样的问题，都抛出一个SQLException，这种做法在异常使用上被称为不完备的信息。因为问题可能是很复杂的，也许是数据库连接的问题，也许是并发控制的问题，也许只是SQL语句出错。没有理由用一个简单的SQLException就搞定全部的问题了，这种做法有些不负责任。针对这两个问题，Spring Framework提出了两种解决方法：首先，提供一个框架，把JDBC应用中的获取连接、异常处理、释放等比较通用的操作全部都集中起来，用户只需要提供特定的实现就OK了。实现的具体细节采用的是模板方法。举个例子，在org.springframework.jdbc.object包中，MappingSqlQuery类实现了将SQL查询映射为具体的业务对象。JavaDoc中这样写到

：Reusable query in which concrete subclasses must implement the abstract mapRow(ResultSet, int) method to convert each row of the JDBC ResultSet into an object. 用户必须实现mapRow方法，这是典型模板方法的应用。我们拿一个具体的例子来看看： 1. class UserQuery extends MappingSqlQuery { 2. 3. public UserQuery(DataSource datasource) { 4. super(datasource, "SELECT * FROM PUB_USER_ADDRESS WHERE USER_ID = ?"). 5. declareParameter(new SqlParameter(Types.NUMERIC)). 6. compile(). 7. } 8. 9. // Map a result set row to a Java object 10. protected Object mapRow(ResultSet rs, int rownum) throws SQLException { 11. User user = new User(). 12. user.setId(rs.getLong("USER_ID")). 13. user.setForename(rs.getString("FORENAME")). 14. return user. 15. } 16. 17. public User findUser(long id) { 18. // Use superclass convenience method to provide strong typing 19. return (User) findObject(id). 20. } 21. } 其次是第二个问题，最麻烦的地方应该说是需要截住JDBC的异常，然后判断异常的类型，并重新抛出异常。错误的问题可以通过连接来获取，所以麻烦的是如何截获异常。Spring 框架采用的方法是回调，处理回调的类在Spring Framework中被称为template 。 1. JdbcTemplate template = new JdbcTemplate(dataSource). 2. final List names = new LinkedList(). 3. template.query("SELECT USER.NAME FROM USER", 4. new RowCallbackHandler() { 5. public void processRow(ResultSet rs) throws SQLException { 6. names.add(rs.getString(1)). 7. } 8. }). 回调函数是一个匿名类，其中也使用了模板方法，异常的处理都在父类中完成了 层间松

耦合 在开放源码界已经出现了大量的基于MVC的Web容器，但是这些容器都仅限于Web的范围，不涉及Web层次后端的连接，Spring作为一个整体性的框架，定义了一种Web层和后端业务层的连接方式，这个思路仍然疏运图MVC的范畴，但耦合更松散，不依赖于具体的集成层次。 1. public class GoogleSearchController 2. implements Controller { 3. 4. private IGoogleSearchPort google. 5. 6. private String googleKey. 7. 8. public void setGoogle(IGoogleSearchPort google) { 9. this.google = google. 10. } 11. 12. public void setGoogleKey(String googleKey) { 13. this.googleKey = googleKey. 14. } 15. 16. public ModelAndView handleRequest(17. HttpServletRequest request, HttpServletResponse response) 18. throws ServletException, IOException { 19. String query = request.getParameter("query"). 20. GoogleSearchResult result = 21. // Google property definitions omitted... 22. 23. // Use google business object 24. google.doGoogleSearch(this.googleKey, query,start, maxResults, filter, r 25. estrict, safeSearch, lr, ie, oe). 26. 27. return new ModelAndView("googleResults", "result", result). 28. } 29. } 回调函数是一个匿名类，其中也使用了模板方法，异常的处理都在父类中完成了。 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