Java **应用框架** Spring

实验指导

艾青

lyaiqing@126.com

实验一、Spring 概述

【实验目的】

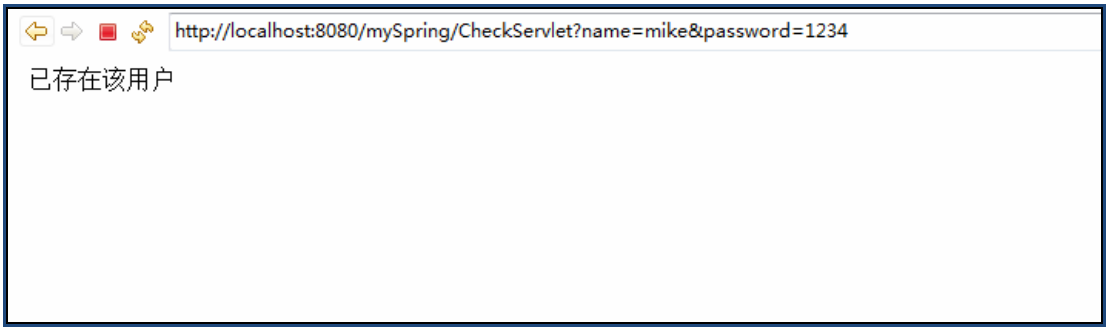
1. 通过 Spring 技术在 jsp 页面中进行用户注册

【实验内容】

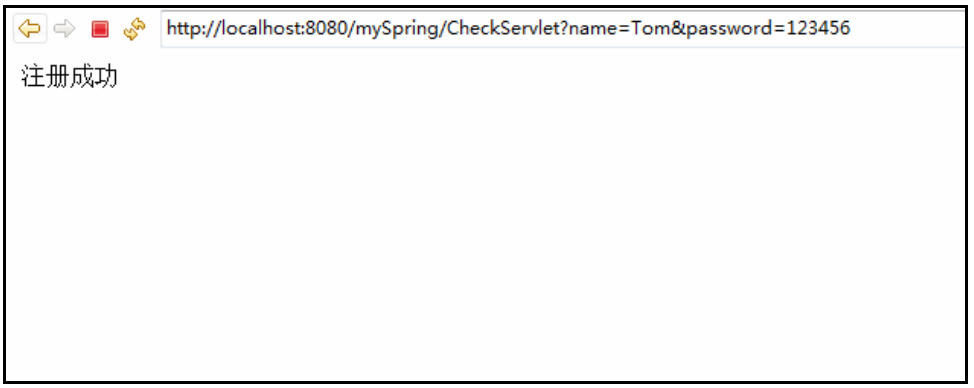
实验案例 001：Spring 快速入门

实现效果

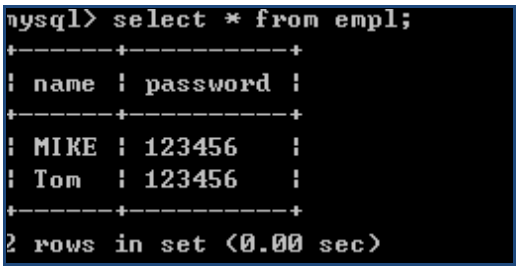
在 name 中输入 mike（password 为任意字符串）时：



在 name 中输入 Tom password 中输入 123456 时：



Mysql 数据库截图：

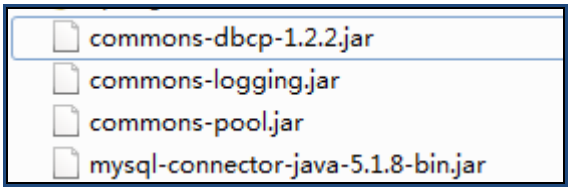


实现步骤

1. **搭建环境：**
   a) **新建相关数据表：**
      i. **在** mysql **数据库中新建** Empl **表,并且插入一条数据（相关** sql **语句在实验指导手册环境工程包目录下）**
         1. **相关表结构如下：**

```
+------+----------+
| name | password |
+------+----------+
| MIKE | 123456   |
+------+----------+
```

2. **进行实验：**
   a) **在** Eclipse **开发工具中新建** Web Project**，我们将其命名为**:mySpring
   b) **导入** jar **包：**
      i. **在实验指导手册/环境工程包目录下的** spring-framework-3.2.0.RELEASE **文件夹内** libs **下的** jar **包复制到** MySpring **项目的** lib **目录内**
      ii. **将实验指导手册/环境工程包目录下的如下** jar **包复制到** MySpring **项目的** lib **目录内**



   c) **进行数据库持久层开发**
      i. **在当前项目的** src **目录下创建一个包，我们将包名命名为：** com.oracle.csg.entity。
         1. **在** com.oracle.csg.entity **包下创建实体类** Empl**，实现** Serializable **接口**
         2. **为** Empl **添加两个属性，并为其添加** set、get **方法，如下**
            a) **属性：**name **数据类型：**String **访问权限：**private
            b) **属性：**password **数据类型：** String **访问权限：**private
      ii. **在当前项目的** src **目录下再创建一个包，命名为** com.oracle.csg.dao。
         1. **在** com.oracle.csg.dao **包下创建一个类，命名为** EmplDao。
         2. **在** ErmplDao **中创建** Spring **提供的数据库模板类** jdbcTemplate **对象，命名为** jdbcTemplate**，访问权限为** private。**并为其添加** set **方法**
         3. **创建方法** queryEmpl();
            a) **访问权限：**public **返回类型：**int **入参：**String name
            b) **具体实现：**
               i. **创建** hql **查询语句：**
                  1. "from Empl where name=?”

ii. 调用 jdbsTemplate 的 queryForList(String hql,Object… args)方法，将 hql 和 name 作为参数传入。返回 List 类型的集合对象 list

iii. 调用 list 的 size()方法获取集合个数，并将其作为方法的返回值

4. 创建 save（）方法，用于保存注册用户
   a) 访问权限：public
   b) 返回类型：void
   c) 入参：String name　String password
   d) 具体实现：
      i. 编写一条 String 类型的 hql 查询语句。如下所示：
         1. String hql="insert into Empl values(?,?)";
      ii. 创建一个 String 类型的数组，将 name 和 password 放入该数组中，调用 jdbcTemplate 的 update()方法，将 hql 和数组作为参数传入。如下所示
         1. jdbcTemplate.update(hql,newString[]{name,password});

d) 创建配置文件

i. 在实验指导手册/环境工程包目录下已为学员提供了配置文件的模板，学员将 applicationContext.xml 文件复制到当前项目的 src 目录下

ii. 在 applicationContext.xml 文件中，创建数据源：
   1. 将 bean 标签的 id 设置为：dataSource ，添加 class 属性，属性值为：org.apache.commons.dbcp.BasicDataSource
   2. 通过标签 property，为 dataSource 添加属性(定义数据库的驱动、url、用户名、密码)
      a) name：driverClassName
         value：com.mysql.jdbc.Driver
      b) name: url
         value: jdbc:mysql:localhost:3306/empl（根据自己数据库实际情况填写正确的 url）
      c) name：username
         value：root（根据自己数据库用户名进行正确的填写）
      d) name：password
         value：root（根据自己数据库密码进行正确的填写）

iii. 在 applicationContext.xml 文件中，使用 spring 容器创建一个 jdbcTemplate
   1. 创建一个 bean 节点。属性如下：
      a) id ：jdbcTemplate
      b) class：
         org.springframework.jdbc.core.JdbcTemplate
   2. 在当前 bean 节点下创建一个 property 标签。property 内的属性如下
      a) name：dataSource
      b) ref：dataSource

iv. 在 applicationContext.xml 文件中，实现对 EmplDao 的管理
    1. 创建一个 bean 节点，属性如下
        a) id:：emplDao
        b) class：com.oracle.csg.dao.EmplDao
    2. 在 bean 节点内创建 property 节点。property 内的属性如下
        a) name：jdbcTemplate
        b) ref：jdbcTemplate

e) 进行业务逻辑层开发
    i. 在当前项目的 src 目录下创建包：com.oracle.csg.service，并在 com.oracle.csg.service 包下创建服务类：EmplService
    ii. 创建 UserDao 对象 userDao，访问权限为 private，并为其添加 set 方法
    iii. 添加方法 check（String name，String password）方法，判断是否允许使用注册，并实现
        1. 调用 emplDao 的 queryEmpl(String name)方法,将 name 作为参数传入，返回 int 类型的 count 对象
        2. 如果 count 等于 0，则返回 true（表示允许使用该用户注册），调用 emplDao 的.save（String name,String password）方法,将 name 和 password 作为参数传入（新增该用户）。否则返回 false
    iv. 在 applicationContext.xml 文件中添加对 EmplService 的管理
        1. 创建 bean 节点。属性如下：
            a) id：EmplService
            b) class：com.oracle.csg.service
        2. 在 bean 节点内添加 property 节点。属性如下
            a) id：emplDao
            b) class：com.oracle.csg.service.EmplDao

f) 进行视图层开发
    i. 在 src 下创建包 com.oracle.csg.servlet
    ii. 在 com.oracle.csg.servlet 包下创建一个 servlet，命名为 CheckServlet
    iii. 在 CheckServlet 里创建 EmplService 对象 emplService，访问权限为 private
    iv. 本次实验我们只对 doPost 方法进行操作。故将如下代码放入 doGet 方法中即可
        1. this.doPost(request, response);
    v. 调用 request.getParameter（String arg）方法分别从页面中获取用户名和密码
    vi. 调用 WebApplicationContextUtils 的 getWebApplicationContext(this.getServletContext())方法，返回 WebApplicationContext 类型对象，命名为 ct
    vii. 通过 cx.getBean("emplService")方法，获得 id emplService 的 bean 的值，并将获得的结果进行类型转换，转化为 EmplService 类型。赋值给 emplService
    viii. 调用 emplService 的 check(String name，String password)方法，进行判断。如果结果为 true，通过如下代码，页面将跳转 allow.jsp 的页面

1. request.getRequestDispatcher("allow.jsp").forward(request, response

ix. **否则，页面将跳转到** forbid.jsp **页面**

（相关 jsp **页面在实验指导手册环境工程包下，请学员将其复制到** WebContent/**目录下）**

x. **在** web.xml **文件中，将** welcome-file **标签内的** index.jsp **修改为** regist.jsp

xi. **在**<display-name>**标签上方创建一个** listener **标签，在该标签内添加** spring **监听器，用于负责启动** spring **容器。如下所示**

1. <listener-class>
   org.springframework.web.context.ContextLoaderListener</listener-class>

xii. **在** listener **标签上方定义一个上下文参数，添加**<context-param>**节点**

1. param-name：contextConfigLocation
2. param-value：classpath:applicationContext.xml

实验二、Spring 的 IOC（一）

【实验目的】

1. Spring **完成依赖关系注入**

【实验内容】

实验案例 001：通过容器完成依赖关系的注入

实现效果

```
2015-2-3 10:29:35 org.springframework.context.support.AbstractApplic
信息: Refreshing org.springframework.context.support.ClassPathXmlAppl
2015-2-3 10:29:35 org.springframework.beans.factory.xml.XmlBeanDefin
信息: Loading XML bean definitions from class path resource [applicat
2015-2-3 10:29:35 org.springframework.beans.factory.support.DefaultL
信息: Pre-instantiating singletons in org.springframework.beans.facto
hello! 通过容器依赖注入
```

实现步骤

1. **构造函数注入**:
   a) **在** Eclipse **开发工具中新建** java Project**，我们将其命名为**: SpringDemo001
   b) **在** src **目录下导入相关** jar **包**
      i. **右击当前项目**——》Build Path——》Configure Build Path…——》**单机** Libraries——》Add Library…..——》User Library——》User Libraries….——New…..
      ii. **创建一个用于放** jar **包的文件夹，命名为** springlib。
      iii. **创建完** springlib **后，选择左侧的** Add External JARS….。**选择**“实验指导手册\环境工程包目录下的“commons-logger.jar”**文件 ，以及当前目录下的**“spring-framework-3.2.0.RELEASE\libs”**目录下的所有** jar **包。将相关** jar **包添加到当前项目中**
   c) **在** src **目录下创建一个包，命名为：**com.oracle.csg.Spring
      i. **在** com.oracle.csg.Spring **包下创建一个接口，命名为：**UserDao。
         1. **在接口中添加方法：**sayHello（String arg）。**返回类型：**void
      ii. **在** src **目录下创建一个类，命名为** User。**实现接口：**UserDao
         1. **在** User **类实现接口中的** sayHello（String arg）**方法：**
            a) **编写一条输出语句：**System.out.println(arg)
      iii. **在** src **目录下创建一个类，命名为** UserService。

1. 在该类中添加属性：UserDao 类型的对象：userDao。访问权限为 private
2. 添加方法：addUser()。返回类型：void。访问权限：public。具体实现：
   a) 调用 userDao 的 sayHello(String arg)方法，将字符串"hello！通过容器依赖注入"作为参数传入。
3. 为 UserService 类添加有参构造方法

iv. 将“实验指导手册\环境工程包目录下的模板文件：applicationContext.xml。复制到当前项目 src 目录下，并对其进行编辑：
1. 将第一个 bean 标签中的的 id 的值改为：User。class 的值改为：com.oracle.csg.Spring.UserService
2. 将第二个 bean 标签中的 id 的值改为 UserService。Class 的值改为：com.oracle.csg.Spring.UserService。并在当前 bean 标签内添加 constructor-arg 标签。属性如下（依赖于 User）：
   a) ref：User

v. 在 src 目录下创建测试类 Test
1. 在 Test 类中创建一个主方法。在主方法中通过如下代码加载 applicationContext.xml 文件
   a) ApplicationContext ct=new ClassPathXmlApplicationContext("applicationContext.xml")
   b) 通过 ct 的 getBean("UserService")方法获取 id 为 UserService 的 bean。并将其进行类型强转。转化为 UserService 类型。返回 UserService 类型的对象：user
   c) 调用 user 对象的 addUser()方法

d) 运行 Test 测试类，观察运行结果

2. 属性注入:
   a) 在 UserService 类下,将 UserService 的有参构造函数注释掉，为 userDao 属性添加 set 方法。
   b) 打开 applicationContext.xml，将第二个 bean 标签内的“constructor-arg”标签注释掉。添加一个“property”标签。属性如下：
      i. name：userDao
      ii. ref：User
   c) 运行 Test 测试类，观察运行结果

实验三、Spring 的 IOC（二）

【实验目的】

1. 通过 BeanFactory 和 ApplicationContext 读取并访问 bean 配置、注入装配 Bean

【实验内容】

实验案例 001：Spring 容器

实现效果

```
2015-2-3 11:06:14 org.springframework.context.support.AbstractAppl
信息: Refreshing org.springframework.context.support.ClassPathXmlAp
2015-2-3 11:06:14 org.springframework.beans.factory.xml.XmlBeanDef
信息: Loading XML bean definitions from class path resource [applic
2015-2-3 11:06:14 org.springframework.beans.factory.support.Defaul
信息: Pre-instantiating singletons in org.springframework.beans.fac
Book [name=上下五千年, price=290.0, page=2648]
```

实现步骤

1. 搭建环境:
   a) 将”实验指导手册\环境工程包”中的”SpringDemo003”导入到 eclipse 开发工具中。

   说明：

   i. 在 SpringDemo003 内已添加相关 jar 包的引用，如 jar 路径与当前不一致，请学员重新引入 jar 包
   ii. 在 SpringDemo003 内已创建了一个 Book 的实体类，并且为 Book 的属性添加 set 方法和有参以及无参的构造方法。）

   b) 打开 src 目录下的“applicationContext.xml”文件，将文件内 bean 标签中的 id 的值改为：Book。class 的值改为：com.oracle.csg.Spring.Book
      i. 在 bean 标签内添加三个 property 标签，为书籍的属性进行赋值。property 标签内的 name 和 value 的值如下：
         1. name：name　　value：上下五千年
         2. name：price　　value：290.00
         3. name：page　　value：2648
2. 进行实验:
   a) BeanFactory 的应用：
      i. 在 com.oracle.csg.Spring 包下创建一个测试类：beanFactoryTest。并为其添加 main 主方法

1. **通过如下代码加载配置文件，返回** Resource **类型对象** res
   a) new FileSystemResource(
      "src/applicationContext.xml")
2. **通过如下代码，实例化** BeanFactory**。返回** BeanFactory **类型对象** bf
   a) new XmlBeanFactory(res)
3. **调用** bf **的** getBean（“Book“）**方法，并将其进行类型转换，转化为** Book **类型，获取** spring **容器中管理的** bean**。返回** Book **类型对象** book
4. **通过** book **的** toString()**方法在控制台打印输出** book **对象的属性。**

ii. **运行测试方法，观察运行结果。**

b) ApplicationContext **的应用：**

i. **在当前包下创建第二个测试类：**applicationTest**。并为其添加** main **主方法**
   1. **通过如下代码实例化** spring **容器。返回** ApplicationContext **类型对象** app:
      a) new ClassPathXmlApplicationContext("applicationContext.xml")
   2. **通过** app **容器的** getBean(“Book”)**方法，并进行类型转换，转化为** Book **类型。返回** Book **类型对象** book
   3. **通过** book **对象的** toString（）**方法，在控制台打印输出** book **对象的属性。**

ii. **运行测试类，观察运行结果。**

实验案例 002：注入装配 Bean

实现效果

```
2015-2-3 16:23:18 org.springframework.context.support.AbstractApplicationCo
信息: Refreshing org.springframework.context.support.ClassPathXmlApplicatior
2015-2-3 16:23:19 org.springframework.beans.factory.xml.XmlBeanDefinitionRe
信息: Loading XML bean definitions from class path resource [applicationCont
2015-2-3 16:23:19 org.springframework.beans.factory.support.DefaultListable
信息: Pre-instantiating singletons in org.springframework.beans.factory.supp
Car [type=BMW, price=670000.0]
Car [type=BMW, price=580000.0]
```

实现步骤

1. **搭建环境**:

a) 将”实验指导手册\环境工程包”中的”SpringDemo004”导入到 eclipse 开发工具中。

说明：在 SpringDemo04 内已为学员添加相关 spring 的 jar 包(如 jar 包路径不正确,请学员重新引入 spring 的相关 jar 包)

i. 在 SpringDemo04 内已存在一个 Car 的实体类和 Type 的实体类，
   1. Type 类中有一个 String 类型的属性 type。并为其添加了 set、get 方法。
   2. 在 Car 类中有两个属性，分别是：Type 类型的对象 type 和 Double 类型的对象 price。并且为属性添加 set 方法和有参以及无参的构造方法。并且重写了 toString 方法

2. 进行实验:
   a) 属性注入装配 Bean：
      i. 打开 src 目录下的 applicationContext.xml 文件。
      ii. 将 Bean 标签中的 id 的值改为：BMW 。class 的值改为：com.oracle.csg.Spring.Type。
         1. 在 Bean 标签中添加 property 标签。property 标签内的属性如下
            a) name：type
            b) value：BMW
      iii. 另外创建一个 Bean 标签。id 的值为：type。class 的值为：com.oracle.csg.Spring.Car。
         1. 在当前 Bean 标签内添加一个 property 标签。property 标签内的属性如下：
            a) name：type
            b) ref：BMW （依赖于 BMW 的 Bean）
         2. 在当前 Bean 标签内添加第二个 property 标签。属性如下：
            a) name：price
            b) value：670000
      iv. 打开 com.oracle.csg.Spring 包下的 Test 类，在主方法中创建测试方法
         1. 通过如下代码加载配置文件，返回 ApplicationContext 类型对象 app
            a) new ClassPathXmlApplicationContext("applicationContext.xml")
         2. 调用 app 的 getBean(String arg)，将“Car”作为参数传入。并将其进行类型转换，转换为 Car 类型。返回 Car 类型对象 car
         3. 调用 car 对象的 toString（）方法，并将其在控制台打印输出
      v. 运行 Test 测试类，观察运行结果。
   b) 构造函数注入装配 Bean：
      i. 打开打开 src 目录下的 applicationContext.xml 文件。
      ii. 创建第三个 Bean 标签，用于进行构造函数注入装配 Bean 的实验。Bean 标签中属性如下：
         1. id：Scar
         2. class：com.oracle.csg.Spring.Car
      iii. 在当前 Bean 标签中添加 constructor-arg 标签。constructor-arg 标签的属性如下：

1. name：type
2. ref：BMW （**依赖于** BMW **的** Bean）

iv. **在当前** Bean **标签中添加第二个** constructor-arg **标签。**constructor-arg **标签的属性如下：**
1. name：price
2. value：580000

v. **打开** com.oracle.csg.Spring **包下的** Test **类，在之前的代码基础上进行如下操作**
1. **调用** app **的** getBean(String arg)**，将**“SCar”**作为参数传入。并将其进行类型转换，转换为** Car **类型。返回** Car **类型对象** Scar
2. **调用** Scar **对象的** toString（）**方法，并将其在控制台打印输出**

vi. **运行** Test **测试类，观察运行结果。**

实验四、Spring 的 IOC（三）

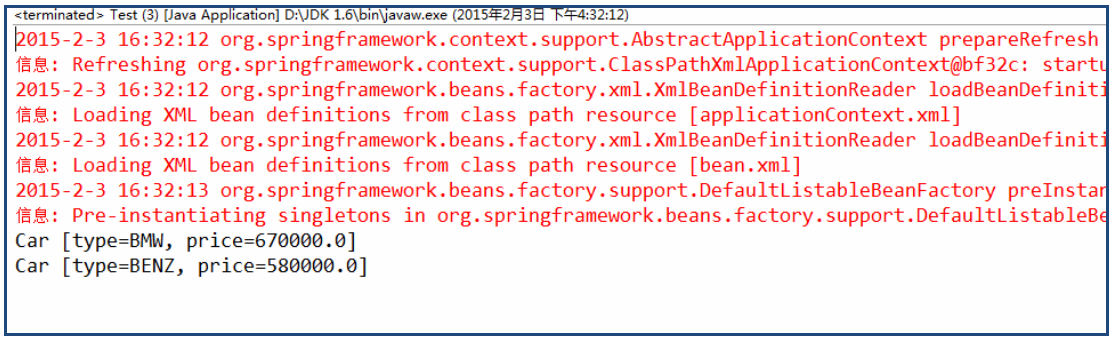
【实验目的】

1. **整合多个配置**
2. **参数注入**
3. **方法注入**

【实验内容】

实验案例 001：整合多个配置

实现效果



实现步骤

1. **搭建环境**:
   a) **将**"实验指导手册\环境工程包"**中的**"SpringDemo005"**导入到** eclipse **开发工具中。**

   说明:

   在 SpringDemo005 **中已添加** spring **的相关** jar **包的引用(** **如** jar **包路径不正确,请学员重新引入** spring **的相关** jar **包**)
2. **进行实验**:
   a) **复制** src **目录下的** applicationContext.xml **配置文件到** src **目录下，并对将其名字改为为：**bean.xml。
   b) **打开** bean.xml **配置文件，并对相关内容进行修改**
      i. **将第一个** Bean **标签内的** id **的值改为：**BENZ。
         1. **将当前** Bean **标签内的** property **标签的** value **值改为：**BENZ
      ii. **将第二个** Bean **标签内的** id **的值改为：**SCar、
         1. **将当前** Bean **标签内的第一个** property **标签的** ref **值改为：**BENZ
         2. **将第二个** property **标签的** value **值改为：**580000
   c) **打开** com.oracle.csg.Spring **包下的** Test **测试类，并对其进行相应的修改**

i. **在主方法中，将**"new ClassPathXmlApplicationContext("applicationContext.xml")"**中括号内的参数：** "applicationContext.xml" **改为如下代码。**

1. new String[]{"applicationContext.xml","bean.xml"}

ii. **在主方法中进行如下操作**

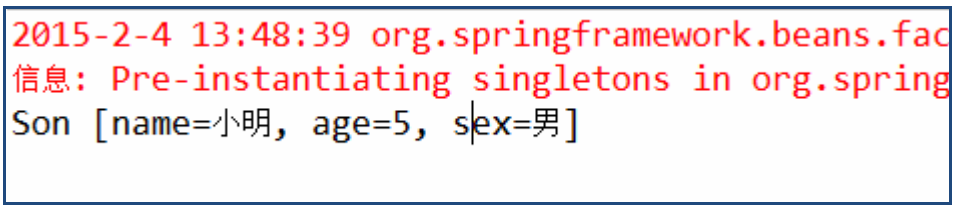
1. **调用** app **的** getBean(String arg)**方法，将** Scar **作为参数传入。并对其进行类型转换，转化为** Car **类型。返回** Car **类型对象** Scar**。**
2. **通过** Scar **的** toString **() 方法，在控制台中输出** Scar **的相关属性。**
   说明：将两个配置文件放入一个字符串数组中，并将其作为新的参数传入。

d) **运行** Test **测试类。观察运行结果**

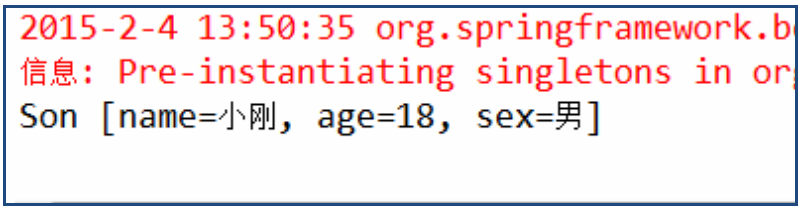
实验案例 002：引用 Bean

实现效果

1、 **当前容器的** Bean **的引用**



2、 **父容器的** Bean **的引用**



实现步骤

1. **搭建环境**:
   a) **将"实验指导手册\环境工程包"中的**"SpringDemo006"**导入到** eclipse **开发工具中。**
2. **进行实验**:
   说明：
   1. **在** SpringDemo006 **中已添加** spring **的相关** jar **包的引用(** **如** jar **包路径不正确,请学员重新引入** spring **的相关** jar **包**)
   2. **在提供的环境工程包**"SpringDemo006"**中存在** Parent **类。**Parent **类中有一个** Son **类型的属性** son**。已为其添加** set、get **方法**

3. 在提供的环境工程包“SpringDemo006”中存在 Son 类。Son 类中有 name、age、sex 属性。已分别为其添加 set 方法。并且为 Son 类添加有参和无参构造方法。最后重写了 toString（）方法，便于实验结果的观察。

a) 引用当前容器的 Bean：
   i. 打开 src 目录下的配置文件：applicationContext.xml。
   ii. 在当前文件内添加一个 bean 元素。bean 元素内的属性如下：
      1. id：Son
      2. class：com.oracle.csg.Spring.Son
   iii. 在 bean 元素内在添加一个 property 元素。通过属性注入的方式对 Son 进行赋值。property 元素的属性值如下：
      1. name：name
      2. value：小明
   iv. 同上，在添加两个 property 元素。对 sex 和 age 进行赋值。将 sex 设置为”男”，age 设置为“5”。
   v. 接下来需要创建 Parent 的 bean，并且 Parent 的 bean 需要引用 Son 的 bean
      1. 再创建一个 bean 元素。id 为：Parent。calss 为：com.oracle.csg.Spring.Parent。
      2. 在当前 bean 元素内添加 property 元素。name 的值为：Son
      3. 在 property 元素内添加 ref 元素。ref 元素内的 bean 的值为：“Son”
   vi. 在 com.oracle.csg.Spring 目录下已提供测试类 Test，直接运行该测试类。观察运行结果。
b) 引用父容器的 Bean
   i. 在 src 目录下创建一个新的配置文件，命名为 bean.xml。并将 applicationContext.xml 文件中的内容（除去 Parent 的 bean 配置）复制到 bean.xml 文件中
   ii. 将 bean 的 id：“Son”改为“Son1”。
   iii. 将 applicationContext.xml 文件中 Parent 的 bean 配置中，ref 元素内的 bean 的值改为“Son1”。
   iv. 在 com.oracle.csg.Spring 包下创建一个新的测试类 ParentTest。将 Test 中的代码全部复制到 ParentTest 中，然后进行相应的修改
      1. 在主方法里的第一行，声明一个父容器。如下所示：
         a) ApplicationContext parent=new ClassPathXmlApplicationContext("bean.xml");
      2. 将 ApplicationContext app=new ClassPathXmlApplicationContext("applicationContext.xml")中括号内的“applicationContext.xml “改为如下代码：
         a) new String[]{"applicationContext.xml"},parent
      3. 其余代码无需改动。直接运行 ParentTest 测试类。观察运行结果。

实验案例 003：null 值属性和级联属性

实现效果

1、 注入 null 值属性

```
信息: Refreshing org.springframework.context.support.ClassPa
2015-2-4 13:59:37 org.springframework.beans.factory.xml.Xml
信息: Loading XML bean definitions from class path resource
2015-2-4 13:59:37 org.springframework.beans.factory.support
信息: Pre-instantiating singletons in org.springframework.be
Goods [price=300.0, weight=null, size=]
```

2、 级联属性

```
2015-2-4 14:23:07 org.springframework.context.support.AbstractApplica
信息: Refreshing org.springframework.context.support.ClassPathXmlAppli
2015-2-4 14:23:07 org.springframework.beans.factory.xml.XmlBeanDefini
信息: Loading XML bean definitions from class path resource [applicati
2015-2-4 14:23:07 org.springframework.beans.factory.support.DefaultLi
信息: Pre-instantiating singletons in org.springframework.beans.factor
Goods [price=150.0, weight=50.23, size=5]
```

实现步骤

1. **搭建环境**:
   a) **将”实验指导手册\环境工程包”中的**”SpringDemo007”**导入到** eclipse **开发工具中。**
2. **进行实验**:
   说明：
   1. **在** SpringDemo007 **中已添加** spring **的相关** jar **包的引用(** **如** jar **包路径不正确,请学员重新引入** spring **的相关** jar **包**)
   2. **在提供的环境工程包**“SpringDemo007”**中存在货物** Goods **类。**Goods **类中有** price(**价格**)、weight(**重量**)、size(**体积**)**属性。已分别为其添加** set **方法。并且为** Goods **类添加有参和无参构造方法。最后重写了** toString **（）方法，便于实验结果的观察。**

说明：

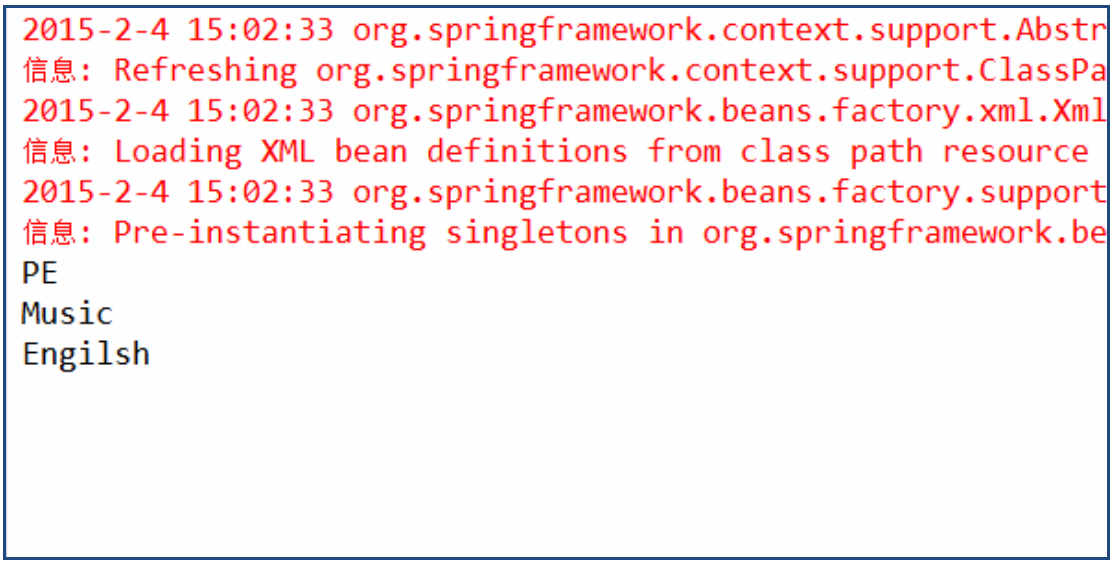
3. 在提供的环境工程包“SpringDemo007”中存在仓库 Warehouse 类。Warehouse 类中有 Goods 类型的属性 goods。已其添加 set、get 方法

a) 注入 null 值的属性：
   i. 打开 src 目录下的配置文件：applicationContext.xml。
   ii. 在当前文件内添加一个 bean 元素。bean 元素内的属性如下：
      1. id：Goods
      2. class：com.oracle.csg.Spring.Goods
   iii. 在 bean 元素内在添加一个 property 元素。通过属性注入的方式对 Goods 进行赋值。property 元素的属性值如下：
      1. name：price
      2. value：300.00
   iv. 在当前 bean 元素内再添加第二个 property 元素。name 的值为：“size“
      1. 在当前 property 元素内添加 value 元素。（value 元素内不添加任何代码）
   v. 在当前 bean 元素内添加第三个 property 元素。name 的为：“weight“。
      1. 在当前 property 元素内添加 null 标签
   vi. 相关测试类已为学员提供，学员直接运行测试类 Test 类即可观察结果。
b) 级联属性
   i. 打开 applicationContext.xml 文件内再创建一个 bean 元素。
      1. id：Warehouse
      2. class：com.oracle.csg.Spring.Warehouse
   ii. 在当前 bean 元素内添加 property 元素。property 元素内的属性如下
      1. name：goods.price
      2. value：150.00
   iii. 同上，创建第二个 property 元素。属性如下
      1. name：goods.weigth
      2. value：50.23
   iv. 同上，创建第三个 property 元素。属性如下
      1. name：goods.weight
      2. value：5
   v. 打开实体类 Warehouse,需要对 goods 对象实例化.如下所示
      1. private Goods goods=new Goods();
   vi. 相关测试类 CascadeTest 已为学员提供，直接运行 CascadeTest 类即可观察实验结果

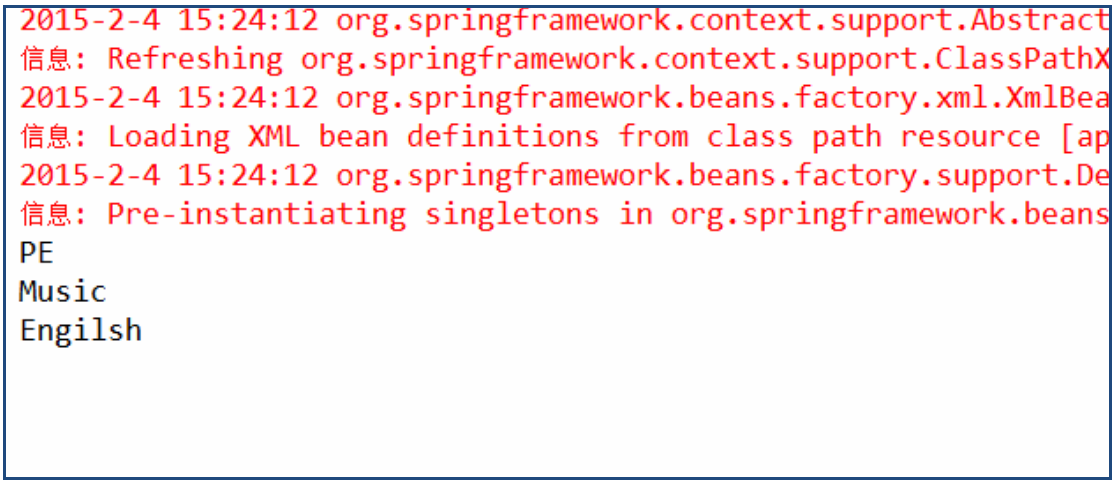
实验案例 004：集合类型属性

实现效果

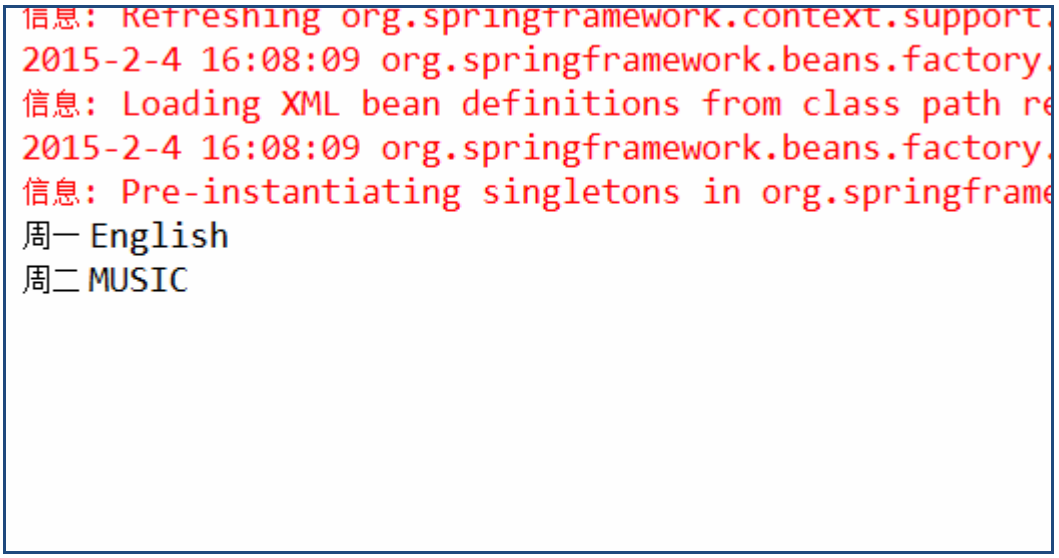
1、 注入 ArrayList 类型的集合类型属性



2、 **注入** HashSet **类型的集合类型属性**



3、 **注入** HashMap **的集合类型属性注入**



实现步骤

1. 搭建环境:
   a) 将”实验指导手册\环境工程包”中的”SpringDemo008”导入到 eclipse 开发工具中。
2. 进行实验:

   说明：

   1. 在 SpringDemo008 中已添加 spring 的相关 jar 包的引用(如 jar 包路径不正确,请学员重新引入 spring 的相关 jar 包)
   2. 在提供的环境工程包“SpringDemo008”中存在学生 ListStudents、SetStudent、MapStudent 类。三个实体类中有 name(姓名)、grade(年级)、subject(选择的课程)属性。已分别为其添加 set、get 方法。并且为实体类添加有参和无参构造方法。

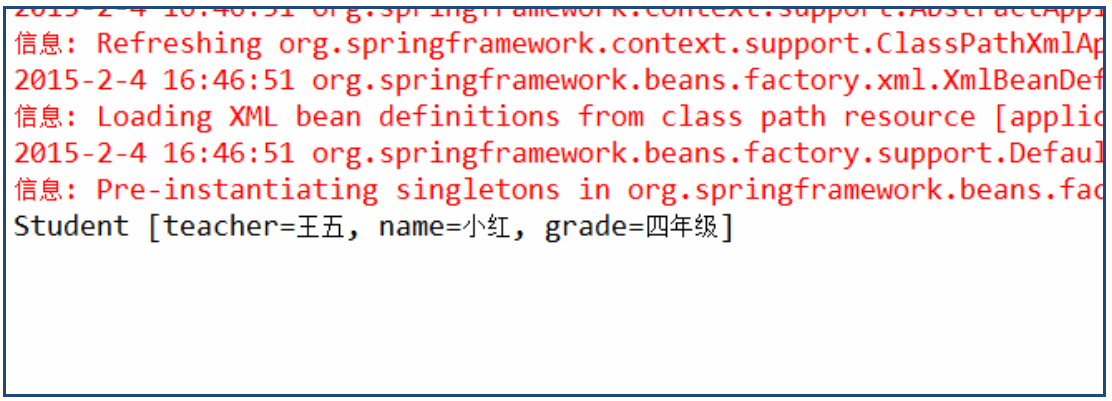
   a) 注入 ArrayList 类型的集合类型属性：
      i. 打开 src 目录下的配置文件：applicationContext.xml。
      ii. 在当前文件内添加一个 bean 元素。bean 元素内的属性如下：
         1. id：Student
         2. class：com.oracle.csg.Spring.entity.ListStudent
      iii. 在 bean 元素内在添加一个 property 元素。通过属性注入的方式对 Goods 进行赋值。property 元素的属性值如下：
         1. name：subjects
      iv. 由于 subject 是 list 类型对象。故在 property 元素下创建 list 元素。
         1. 通过 value 元素分别添加课程：PE、music、English
      v. 相关测试类 ListTest 已为学员提供，直接运行 com.oracle.csg.Spring.test 包下的 ListTest 类即可观察实验结果
   b) 注 HashSet 类型的集合类型属性
      i. 在 applicationContext.xml 文件中，复制上一步中的 bean 配置。并对其进行修改
         1. 将 bean 的 id 改为 Student1，class 改为：com.oracle.csg.Spring.entity.SetStudent
         2. 将 property 元素中的 list 元素改为 set 元素。
      ii. 相关测试类 SetTest 已为学员提供,直接运行 com.oracle.csg.Spring.test 包下的 SetTest 类即可观察实验结果
   c) 注 HashMap 类型的集合类型属性
      i. 在 applicationContext.xml 文件中创建第三个 bean。
         1. id：Students2
         2. class：com.oracle.csg.Spring.entity.MapStudent
      ii. 在当前 bean 元素内添加 property 元素
         1. name：subjects
      iii. 在 property 元素内添加 map 元素。在 map 元素内添加 entry 元素。
         1. 在 entry 元素内添加 key 和 value 元素。
            a) 在 key 元素内添加 value 元素，value 元素的文本内容为：周一
            b) value 元素的文本内容为：English

iv. 同上，添加第二个 entry 元素。类似的，将 key 元素内的 value 元素的文本内容设置为：周二。Value 元素的文本内容设置为 MUSIC

相应的测试类 MapTest 已为学员提供，请学员直接运行 MapTest 类即可观察实验结果。

实验案例 005：简化配置方法

实现效果



实现步骤

1. 搭建环境:
   a) 将”实验指导手册\环境工程包”中的”SpringDemo009”导入到 eclipse 开发工具中。
2. 进行实验:

   说明：

   1. 在 SpringDemo009 中已添加 spring 的相关 jar 包的引用(如 jar 包路径不正确,请学员重新引入 spring 的相关 jar 包)
   2. 在提供的环境工程包“SpringDemo009”中已存在老师的实体类 Teacher.。在 Teacher 中有 String 类型的属性 name。已添加 set、get 方法
   3. “SpringDemo009”中还有学生的实体类 Student。在 Student 中有 Teacher 类型的属性 teacher 以及 String 类型的 name 和 grade 属性。已为属性添加 set、get 方法并未类添加有参和无参构造方法。

   a) 简化装配：
      i. 打开 src 目录下的配置文件：applicationContext.xml。在配置文件中引入“p”命名空间。
         1. 将如下代码放入文件开头的 bean 元素内
            a) xmlns:p="http://www.springframework.org/schema/p"
      ii. 将 Teacher 的 bean 里的 property 标签及内容删除，在 class 属性的后面添加：
         1. p:name=“王五”（对 teacher 的 name 进行赋值）

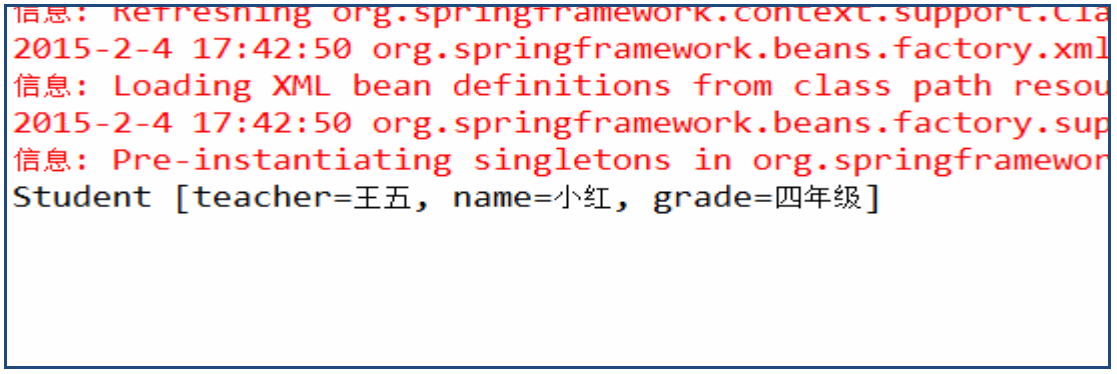
iii. 将 Student 的 bean 里的三个 property 标签及内容删除。在 class 属性的后面添加（三个 p 之间用空格隔开）
   1. p:teacher-ref="Teacher"（依赖 Teacher 的 bean）
   2. p:name="小红" （对学生的 name 进行赋值）
   3. p:grade="四年级"（对学生的年纪进行赋值）

iv. 相关测试类 Test 已为学员完成，直接运行 Test，观察实验结果

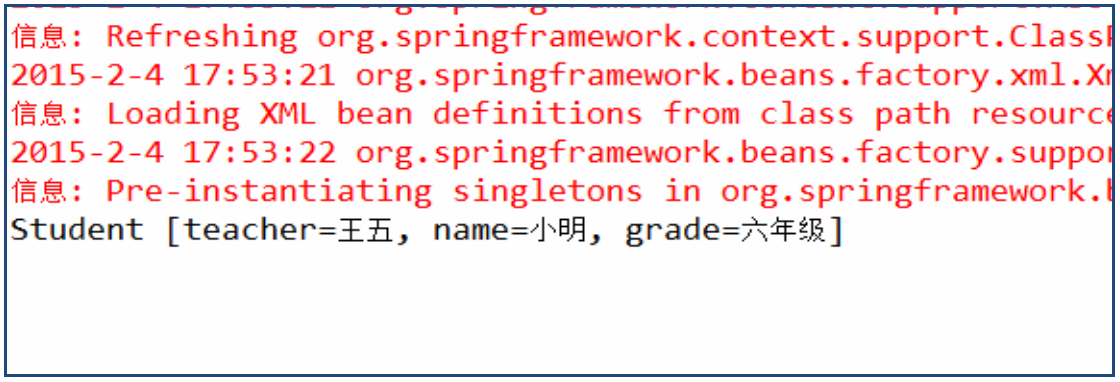
实验案例 006：自动化装配

实现效果

属性注入（byName、byType）自动化装配



构造函数注入的自动化装配



实现步骤

3. 搭建环境:
   a) 将"实验指导手册\环境工程包"中的"SpringDemo010"导入到 eclipse 开发工具中。
4. 进行实验:
   说明：

1. 在 SpringDemo010 中已添加 spring 的相关 jar 包的引用(如 jar 包路径不正确,请学员重新引入 spring 的相关 jar 包)
2. 在提供的环境工程包“SpringDemo010”中已存在老师的实体类 Teacher.。在 Teacher 中有 String 类型的属性 name。已添加 set、get 方法
3. “SpringDemo010”中存在学生的实体类 Student。在 Student 中有 Teacher 类型的属性 teacher 以及 String 类型的 name 和 grade 属性。已为属性添加 set、get 方法并未类添加有参和无参构造方法。

a) 属性注入的”byName”的自动化装配：
   i. 打开 applicationContext.xml 文件。将 Student 的 bean 中的如下代码删除
      1. <property name="teacher" ref="Teacher"></property>
   ii. 在 Student 的 bean 标签的 class 属性后面添加“autowire”属性。值为：“byName”。
   iii. “byName”的自动装配要求容器中存在于 Student 的属性名一样的 bean，由于 Student 类中是“teacher”属性，故将 Teacher 的 bean 的 id 改为“teacher”
   iv. 相关测试类 Test 已为学员完成，直接运行 Test，观察实验结果
b) 属性注入的“byType”的自动装配
   i. 将上面步骤中“autowire”属性：byName 改为 byType。（由于 byType 是根据类型，故对学生的 bean 的 id 没有要求）
   ii. 运行 Test 测试类。观察实验结果。
c) 构造函数注入的自动化装配
   i. 在 id 为 Student1 的 bean 标签内的 class 属性后面添加“autowire”属性，值为：constructor
   ii. 将以下依赖注入的相关配置代码删除
      1. <constructor-arg type="com.oracle.csg.Spring.Teacher"
                                ref="Teacher"></constructor-arg>
   iii. 打开 com.oracle.csg.Spring 包下的 Test 测试类,将如下代码中 getBean（）方法内的参数 Student 改为 Student1
      1. Student stu=(Student) app.getBean("Student");
   iv. 运行 Test 测试类，观察实验运行结果

实验五、Spring 的 IOC（四）

【实验目的】

1. Bean **作用域**
2. FactoryBean

【实验内容】

实验案例 001：singleton 和 prototype 作用域

实现效果

singleton **的作用域**:

```
2015-2-6 17:51:36 org.springframework.context.support
信息: Refreshing org.springframework.context.support.C
2015-2-6 17:51:37 org.springframework.beans.factory.x
信息: Loading XML bean definitions from class path res
2015-2-6 17:51:37 org.springframework.beans.factory.s
信息: Pre-instantiating singletons in org.springframew
true
false
true
```

prototype **的作用域**

```
2015-2-6 18:21:11 org.springframework.context.support.Abs
信息: Refreshing org.springframework.context.support.Class
2015-2-6 18:21:11 org.springframework.beans.factory.xml.X
信息: Loading XML bean definitions from class path resourc
2015-2-6 18:21:11 org.springframework.beans.factory.suppo
信息: Pre-instantiating singletons in org.springframework.
false
false
false
```

实现步骤

1. **搭建环境**

- **将实验指导手册目录下的**”SpringDemo012”**导入到**eclipse**中。**

  说明：

  1. **在** SpringDemo012 **中已添加** spring **的相关** jar **包的引用(** **如** jar **包路径不正确,请学员重新引入** spring **的相关** jar **包**)

2. 在提供的环境工程包“SpringDemo012”中已存在老师的实体类 Teacher.。在 Teacher 中有 String 类型的属性 name。已添加 set、get 方法

3. “SpringDemo012”中存在学生的实体类 Student。在 Student 中有 Teacher 类型的属性 teacher 以及 String 类型的 name 和 grade 属性。已为属性添加 set、get 方法并未类添加有参和无参构造方法。

2. 配置单例模式 singleton 的作用域：

- 在applicationContext.xml文件中，在teacher的bean配置中，在class属性的后面添加scope属性，属性值为：singleton
- 相关测试类已为学员提供，以下操作已为学员完成，学员直接运行Test测试类，观察运行结果。
  - 在com.oracle.csg.Spring包下的Test中有三条输出语句。
    - 两次获取teacher的bean对象，并通过==进行比较，将比较结果在控制台输出
    - 分别获取Student1和Student2的bean对象，通过并通过“ == ”进行比较，将比较结果在控制台输出
    - 分别获取Student1和Student2的bean对象的teacher属性，通过并通过“ == ”进行比较，将比较结果在控制台输出

3. 配置原型模式 prototype 的作用域

- 将teacher的bean配置中scope属性改为：prototype。
- 运行Test测试类。观察运行结果。

实验案例 002：工厂 Bean(一)

实现效果

getObject（）属性赋值

方法一：

```
信息: Refreshing org.springframework.context.support.ClassPathXmlApplic
2015-2-9 12:23:58 org.springframework.beans.factory.xml.XmlBeanDefinit
信息: Loading XML bean definitions from class path resource [applicatio
2015-2-9 12:23:58 org.springframework.beans.factory.support.DefaultLis
信息: Pre-instantiating singletons in org.springframework.beans.factory
Pet [主人=赵六, 名字=旺旺, 品种=金毛犬]
```

方法二：

```
2015-2-9 12:20:34 org.springframework.context.suppor
信息: Refreshing org.springframework.context.support.
2015-2-9 12:20:35 org.springframework.beans.factory.
信息: Loading XML bean definitions from class path re
2015-2-9 12:20:35 org.springframework.beans.factory.
信息: Pre-instantiating singletons in org.springframe
Pet [主人=张三, 名字=大旺, 品种=藏獒]
```

实现步骤

1. 搭建环境

- 将实验指导手册目录下的”SpringDemo013”导入到eclipse中。

说明：

1. 在 SpringDemo013 中已添加 spring 的相关 jar 包的引用(如 jar 包路径不正确,请学员重新引入 spring 的相关 jar 包)
1. “SpringDemo013”中存在宠物的实体类 Pet。在 Pet 中有 String 类型的属性 owner 以及 String 类型的 name 和 variety 属性。已为属性添加 set、get 方法并未类添加有参和无参构造方法。
2. 在配置文件 applicationContext.xml 文件中已添加了 Pet 的 Bean 配置

- 通过FactoryBean创建Bean的环境搭建：
  - 在com.oracle.csg.Spring包下创建一个类：PetBean。实现FactoryBean接口。（将FactoryBean接口的泛型修改为PetBean）
  - 实现接口中的方法。
  - 打开配置文件applicationContext.xml。在bean配置里，将class的值修改为：com.oracle.csg.Spring. PetBean

2. getObject（）方法对属性赋值

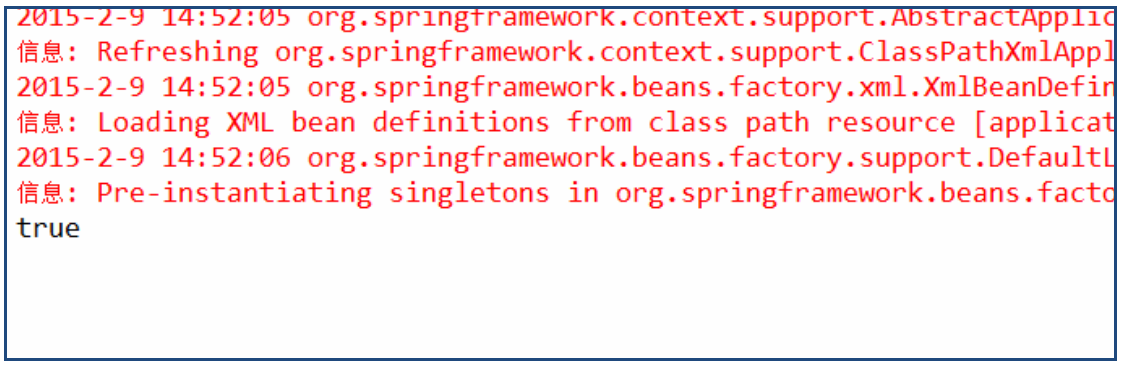
- 方法一：
  - 在PetBean类中的getObject（）方法内，实例化一个Pet对象，命名为pet。
  - 通过宠物（Pet）类里的属性的set方法对pet的属性进行赋值。具体属性值如下
    - name：旺旺
    - variety：金毛犬
    - owner：赵六
  - 将pet对象作为方法的返回值
  - 相关测试类Test已为学员完成，请学员直接运行Test测试类。观察实验结果。
- 方法二：
  - 将getObject（）方法中的代码注释掉，在PetBean类内创建一个String类型的对象，命名为petInfo，访问权限为private。并为其添加set方法
  - 将通过split（String arg）方法，将petInfo里的字符串用“ ，”分隔开。并存放在String类型的数组对象info中。
  - 在getObject（）方法中实例化Pet对象，命名为pet。
  - 通过下表分别获取Info中的数据。然后对pet对象的属性进行赋值。如下所示
    - pet.setName(info[0]);
    - pet.setOwner(info[1]);
    - pet.setVariety(info[2]);
  - 将pet对象作为方法的返回值
  - 在applicationContext.xml文件中，在Pet的bean内中添加property元素。
    - name：petInfo
    - value：大旺,张三,藏獒
  - 运行Test测试类，观察运行结果。

实验案例 003：工厂 Bean(二)

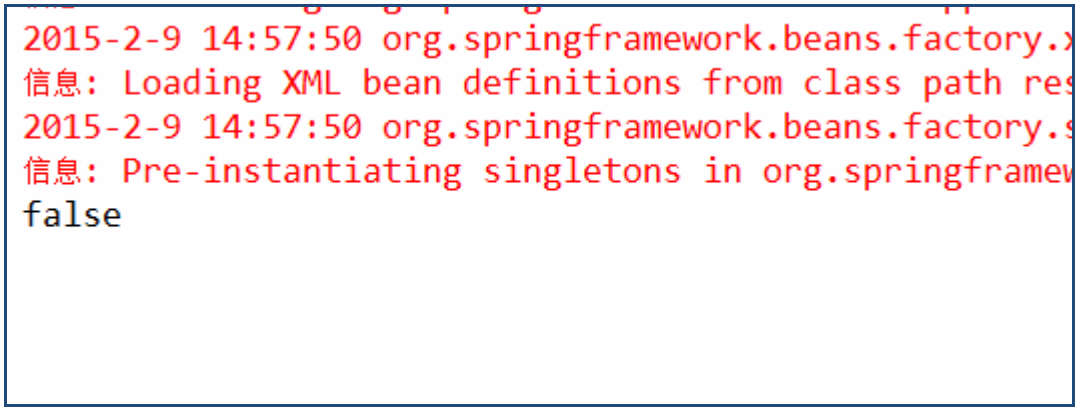
实现效果

isSingleton()方法

方法返回值为 true 时运行结果：



方法返回值为 false 时运行结果：



实现步骤

1. 搭建环境

- 将实验指导手册目录下的”SpringDemo014”导入到eclipse中。

说明：

2. 在 SpringDemo014 中已添加 spring 的相关 jar 包的引用(如 jar 包路径不正确,请学员重新引入 spring 的相关 jar 包)
3. 在提供的环境工程包“SpringDemo014”中已存在主人（Owner）的实体类。在 Owner 中有 String 类型的属性 name。已添加 set、get 方法
4. “SpringDemo014”中存在宠物的实体类 Pet。在 Pet 中有 Owner 类型的属性 owner 以及 String 类型的 name 和 variety 属性。已为属性添加 set、get 方法并未类添加有参和无参构造方法。
5. 在配置文件 applicationContext.xml 文件中已添加了 Pet 的 Bean 配置

- 通过FactoryBean创建Bean的环境搭建：
  - 在com.oracle.csg.Spring包下创建一个类：PetBean。实现FactoryBean接口。（将

FactoryBean接口的泛型修改为PetBean）

- 实现接口中的方法。
- 打开配置文件applicationContext.xml。在bean配置里，将class的值修改为：com.oracle.csg.Spring. PetBean

2. isSingleton()方法的运用:

- 在PetBean 类中的getObject()方法中实例化Pet对象,命名为pet.对pet的属性进行赋值。设置name为:旺旺。Variety：金毛犬。Owner：赵六。（这里的属性的值对实验结果没有影响，学员也可以为这三个属性设置其它的值）
- 将PetBean类中isSingleton()方法的返回值false改为true
- 运行Test测试类，观察运行结果。（此时控制台为true，由于在isSingleton()方法中返回true，表示使用单例模式）
- 将isSingleton()方法的返回值改回false。运行Test，观察运行结果。（此时表示使用原型模式）

3. getObjectType()方法

- 由于在当前FactoryBean中创建的是Pet类型的bean，所以在getObjectType（）方法内，将返回值改为Pet.class。（表示FactoryBean创建Bean的类型为Pet类型）

实验六、Spring 的 IOC（五）

【实验目的】

1. 注解配置

【实验内容】

实验案例 001：注解配置

实现效果

控制台运行结果：

```
2015-2-9 18:27:47 org.springframework.context.support.Abstra
信息: Refreshing org.springframework.context.support.ClassPat
2015-2-9 18:27:47 org.springframework.beans.factory.xml.XmlB
信息: Loading XML bean definitions from class path resource [
2015-2-9 18:27:48 org.springframework.beans.factory.support.
信息: Pre-instantiating singletons in org.springframework.bea
Dao
Service
Dao
Controller
Service
Dao
```

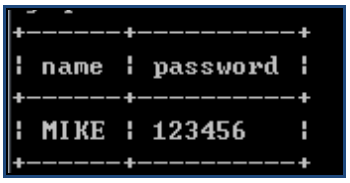
数据库数据截图：（由于执行了三次 save 方法，故数据库中出现三条 jack 的数据）

```
mysql> select * from empl;
+------+----------+
| name | password |
+------+----------+
| mike | 123456   |
| Jack | 1111     |
| Jack | 1111     |
| Jack | 1111     |
+------+----------+
4 rows in set (0.00 sec)
```

实现步骤

1. 搭建环境

- 新建相关数据表（如本地已存在该数据库和表无需新建）：
  - 在mysql数据库中新建Empl表,并且插入一条数据(相关sql语句在实验指导手册环境工程包目录下)
  - 相关表结构如下：



- 将实验指导手册目录下的”SpringDemo015”导入到eclipse中。

  说明：

  1. 在 SpringDemo015 中已添加 spring 的相关 jar 包的引用(如 jar 包路径不正确,请学员重新引入 spring 的相关 jar 包)
  2. 在“ SpringDemo015 ”中，在com.oracle.csg.entity包下有Empl实体类，并且已添加set、get方法
  3. 在com.oracle.csg.dao包下的Dao类里，已完成持久层的开发。在Dao类里有一个save（）方法，该方法是将用户名为“ jack ”，密码为“ 1111 ”的对象保存到数据库中
  4. 在applicationContext,xml内创建相关数据库连接的配置。

- 将applicationContext.xml文件中dataSource的bean配置中的url、username、password根据自己的数据库进行相应的修改

2. 通过注解定义 bean：

- 在Dao类中save（）方法内的最后一行添加一条输出语句，便于实验观察结果：
  - System.out.println("Dao");
- 在Dao类的上面添加如下注解：
  - @Repository
- 在如下代码上面添加@Autowired注解，表示自动装配
  - private JdbcTemplate jdbcTemplate;

说明：关于“ 自动装配的注解 ”将在接下来的学习中讲解，本次实验里不做过多解释，我们直接引用

- 在com.oracle.csg.service包下创建Service类。并为Service类添加“ @Service ”注解
  - 在service类中创建一个私有的Dao类型对象dao。在dao的上一行添加@Autowired注解。
  - 在service类中添加一个save（）方法。访问权限：public 。返回类型：void。具体实现如下
    - 编写一条输出语句，便于实验结果观察
      - System.out.println("Service");
    - 调用dao的save()方法。
- 在com.oracle.csg.controller包下创建Controller类。为Controller类添加“ @Controller ”注解
  - 在Controller类中创建一个私有的Service类型对象service。并未其添加自动装配的注解
  - 在Controller类中添加一个save（）方法。访问权限：public 。返回类型：void。具体实现如下
    - 调价一条输出语句，便于实验结果观察
      - System.out.println("Controller");
    - 调用service的save()方法。

3. 通过注解配置信息启动 Spring 容器

- 在applicationContext.xml文件中增加context的命名空间的声明
  - 在beans标签内添加如下代码：
    - xmlns:context=“ http://www.springframework.org/schema/context ”
  - 在beans标签内的
    “ http://www.springframework.org/schema/beans/spring-beans-3.0.xsd ”后面添加如下代码（注意加在引号里面，并用空格或换行分隔）：
    - http://www.springframework.org/schema/context
    - http://www.springframework.org/schema/context/spring-context-3.0.xsd
- 在当前文件内添加“ context:component-scan ”元素，base-package的值为：“ com.oracle.csg ”（从com.oracle.csg包下进行组件扫描）
- 相关测试类Test已提供，依次揭开如下代码的注释，并依次运行程序。观察运行结果
  - emplDao.save();
  - emplService.save();
  - emplController.save();

实验七、Spring 的 AOP（一）

【实验目的】

1. Jdk **动态代理**
2. cglib **动态代理**

【实验内容】

实验案例 001：JDK 动态代理

实现效果

```
吃饭
**************************************
洗手
吃饭
散步
```

实现步骤

1. **搭建环境**

- **将实验指导手册目录下的**”SpringDemo016”**导入到**eclipse**中。**

说明：

1. **在** SpringDemo016 **中已添加** spring **的相关** jar **包的引用(** **如** jar **包路径不正确,请学员重新引入** spring **的相关** jar **包**)
2. **在“** SpringDemo016 **”中，在**com.oracle.csg.Spring**包下有**Eat**类，**Eat**类中有**Eating**方法，在该方法内有一条输出语句。输出：“吃饭”**

2. JDK **动态代理：**

- **先运行**Test**测试类，观察实验前的运行结果。运行后控制台的输出为如下截图**

■

```
<terminated> Test (13) [Java Application] D:\JDK 1.6\bin\javaw.exe (2015年2月10日 下午4:47:32)
吃饭
**************************************
```

- **在**com.oracle.csg.Spring**包下创建接口：**IEat**。并添加**Eating（）**方法。返回类型为** void
- **使**Eat**类实现**Ieat**接口。**
- **创建代理类：**HandlerEat**。实现**InvocationHandler**接口。**
- **在**invoke(Object arg0, Method arg1, Object[] arg2)**方法内添加两条输出语句（我们要**

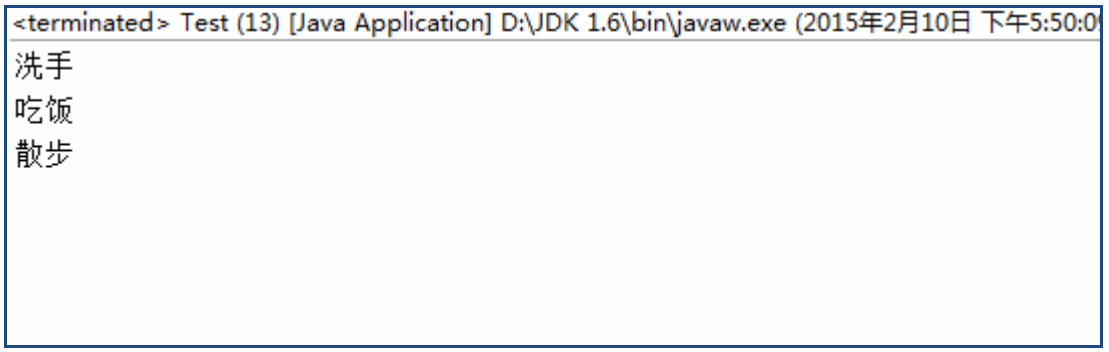
求在吃饭前要洗手，然后吃完饭去散步）

- System.out.println("洗手");
- System.out.println("散步");

- 在HandlerEat里定义一个私有的Eat对象eat，并且为HandlerEat类添加有参和无参构造方法
- 在添加的两个输出语句中间。调用方法中的Method类型参数arg1的invoke(obj, args)方法。将eat和arg2作为参数传入
- 编写相关测试代码，打开Test测试类。实例化Eat对象eat。
- 通过HandlerEat的有参构造方法，将eat作为参数传入，实例化HandlerEat对象handler
- 调用Proxy的newProxyInstance(arg0, arg1, arg2)。三个参数含义分别是：目标对象运行环境的类加载器、目标参数的类的接口、需要植入的增强业务类。三个参数分别为：
  - eat.getClass().getClassLoader()
  - eat.getClass().getInterfaces()
  - handler
- 返回Ieat类型对象eatHandler （注意：需要进行类型转换）
- 调用eatHandler的Eating（）方法。
- 运行Test测试类，观察运行结果。

实验案例 002：CGLIB 动态代理

实现效果



实现步骤

1. 搭建环境

- 将实验指导手册目录下的”SpringDemo017”导入到eclipse中。

说明：

1. 在 SpringDemo017 中已添加 spring 的相关 jar 包以及 CGLIB 的 jar 包的引用(如 jar 包路径不正确,请学员重新引入 spring 的相关 jar 包 ，环境工程包中已提供所需的所有 jar 包)
2. 在“ SpringDemo017 ”中，在com.oracle.csg.Spring包下有Eat类，Eat类中有Eating方法，在该方法内有一条输出语句。输出：“ 吃饭 ”

2. CGLIB动态代理：

- 在com.oracle.csg.Spring包下创建代理类：CGProxy。并使其实现MethodInterceptor接口。
- 通过实例化Enhancer类创建子类实例。实例化对象命名为：en。访问权限为私有的
- 定义方法getProxy(Class clazz)。返回类型：void。访问权限：public
  - 调用en的setSuperclass(Class clazz)方法，将clazz作为参数传入。用于指定哪个superclass创建子类
  - 调用en的回调方法setCallback(Callback Callback)方法，将this作为参数传入。
  - 方法返回en对象的create（）方法
- 接下来，在intercept（Object arg0, Method arg1, Object[] arg2,MethodProxy arg3）方法中编写横切关注点代码
  - 添加如下输出语句
    - System.out.println("洗手");
    - System.out.println("散步");
  - 在以上两行输出代码的中间调用intercept（）方法中的参数arg3的invokeSuper(arg0, arg1)方法。其中arg0参数表示为：目标对象。arg1参数表示：目标对象方法的参数。故将arg0和arg2。返回Object类型对象result
- 在Test测试类中的主方法中，实例化CGProxy对象cg。
- 调用cg对象的getProxy（clazz）方法，将Eat.class作为参数传入。返回Eat类型对象eat。（需要进行类型转换）
- 调用eat的Eating（）方法。
- 运行Test测试类。观察运行结果。

实验八、Spring 的 AOP（二）

【实验目的】

1. 增强组件

【实验内容】

实验案例 001：前置增强

实现效果

实现步骤

1. 搭建环境

- 将实验指导手册目录下的”SpringDemo018”导入到eclipse中。

说明：

1. 在 SpringDemo018 中已添加 spring 的相关 jar 包的引用(如 jar 包路径不正确,请学员重新引入 spring 的相关 jar 包)
2. 在“ SpringDemo018 ”中，在com.oracle.csg.Spring包下有Penguin接口，Penguin接口中有SleepAndBeat方法.
3. 在com.oracle.csg.Spring包下有FirstPenguin类。FirstPenguin类实现Penguin接口。在FirstPenguin中实现了Penguin接口中的SleepAndBeat方法。

2. 前置增强：

- 在com.oracle.csg.Spring包下创建GreetBeforeAdvice类，并且实现MethodBeforeAdvice接口。在GreetBeforeAdvice类中重写接口中的before(Method arg0, Object[] arg1, Object arg2)：
  - 编写一条输出语句，将字符串“ 吃饭 ”打印输出在控制台。
- 编写测试类Test
  - 通过FirstPenguin实例化一个Penguin对象，命名为pe（定义目标对象）
  - 通过如下代码定义增强：
    - BeforeAdvice advice=new GreetBeforeAdvice();
  - 通过ProxyFactory生产代理对象
    - 实例化ProxyFactory对象，命名为pf
  - 调用pf的setTarget（Object target）方法，将pe作为参数传入

- 调用pf的setAddvice（Advice advice）方法，将advice对象作为参数传入
- 调用pf的getProxy()方法，返回Penguin类型对象，命名为proxy。（注意：这里需要进行类型强转）
- 调用proxy的SleepAndBeat（）方法。

- 运行Test测试类，观察运行结果。

实验案例 002：后置增强

实现效果

```
吃饭，睡觉！
打豆豆！
```

实现步骤

1. 搭建环境

- 将实验指导手册目录下的”SpringDemo019”导入到eclipse中。

说明：

1. 在 SpringDemo019 中已添加 spring 的相关 jar 包的引用(如 jar 包路径不正确,请学员重新引入 spring 的相关 jar 包)
2. 在“ SpringDemo019 ”中，在com.oracle.csg.Spring包下有Penguin接口，Penguin接口中有EatingAndSleep方法.
3. 在com.oracle.csg.Spring包下有FirstPenguin类。FirstPenguin类实现Penguin接口。在FirstPenguin中实现了Penguin接口中的EatingAndSleep方法。

2. 前置增强：

- 在com.oracle.csg.Spring包下创建GreetAfterAdvice类，并且实现AfterReturningAdvice接口。在GreetAfterAdvice类中重写接口中的afterReturning(Object arg0, Method arg1, Object[] arg2, Object arg3)方法：
  - 编写一条输出语句，将字符串“ 打豆豆！ ”打印输出在控制台。
- 编写配置文件：
  - 打开src目录下的applicationContext.xml文件
  - 通过如下代码引入p命名空间
    - xmlns:p=”http://www.springframework.org/schema/p”
  - 定义目标对象
    - 添加一个bean，id为：target。class为：com.oracle.csg.Spring.FirstPenguin
  - 定义后置增强
    - 添加一个bean，id为：afterAdvice。class为：com.oracle.csg.Spring.GreetAfterAdvice
  - 定义代理对象
    - 添加一个bean，id为：Penguin。class为：

org.springframework.aop.framework.ProxyFactoryBean

- **在当前的**bean**元素中添加**p**命名空间设置代理对象的属性。如下所示：**
  - p:proxyInterfaces="com.oracle.csg.Spring.Penguin"（**表示对**Penguin**接口进行代理**）**
  - p:interceptorNames="afterAdvice"（**表示使用**afterAdvice**这个**bean**进行拦截**）**
  - p:target-ref="target"

- **编写测试类**Test
  - **打开**com.oracle.csg.Spring**包下的**Test**测试类，在**main**方法中通过如下代码加载配置文件**
    - ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.xml")
  - **调用**ctx**的**getBean（String arg）**方法，将“** Penguin **”作为参数传入，返回**Penguin**类型对象**proxy。（**注意：此处需要进行类型转换**）**
  - **调用**proxy**的**EatingAndSleep（）**方法**

运行 Test **测试类，观察运行结果。**

实验九、Spring 的 AOP（三）

【实验目的】

1. 切面组件

【实验内容】

实验案例 001：静态普通方法名匹配切面

实现效果

```
吃饭！
睡觉！
打豆豆！
**********
睡觉
```

实现步骤

1. 搭建环境

- 将实验指导手册目录下的”SpringDemo022”导入到eclipse中。

说明：

1. 在 SpringDemo022 中已添加 spring 的相关 jar 包的引用(如 jar 包路径不正确,请学员重新引入 spring 的相关 jar 包)
2. 在“ SpringDemo022 ”中，在com.oracle.csg.spring包下有FirstPenguin类，FirstPenguin类中有Sleep和Beat方法.
3. 在com.oracle.csg.spring包下有SecondPenguin类。SecondPenguin类中有Sleep方法。

2. 静态普通方法名匹配切面：

- 在com.oracle.csg.spring包下创建GreetBeforeAdvice类，并且实现MethodBeforeAdvice接口。在GreetBeforeAdvice类中重写接口中的 before(Method method, Object[] args, Object arg2)方法：
  - 编写一条输出语句，将字符串“ 吃饭！ ”打印输出在控制台。
- 在com.oracle.csg.spring包下创建GreetAdvisor,继承StaticMethodMatcherPointcutAdvisor类。并重写StaticMethodMatcherPointcutAdvisor类中的”matches(Method arg0, Class<?> arg1)”方法。
  - 调用方法名中的参数arg0的getName（）方法，通过equals与“ Sleep ”进行比较。并将其比较结果作文方法的返回值
- 在GreetAdvisor类下添加getClassFilter()方法，方法返回类型：ClassFilter。访问全选：public。
  - 方法返回new ClassFilter()（由于ClassFilter无法实例化，故再匿名内部类中实例化）

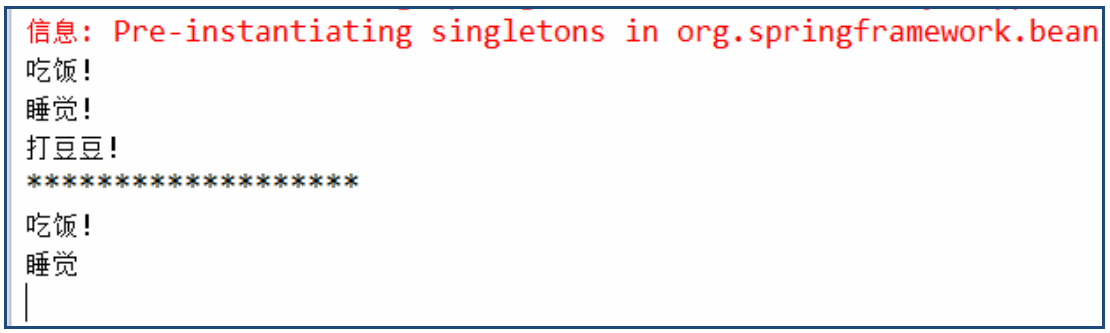
- **重写**matches(Class<?> arg0)**方法：（通过如下代码对**FirstPenguin**类进行匹配）**
  - return FirstPenguin.class.isAssignableFrom(arg0);

- **编写配置文件：**
  - **打开**src**目录下的**applicationContext.xml**文件**
  - **通过如下代码引入**p**命名空间**
    - xmlns:p=”http://www.springframework.org/schema/p”
  - **定义目标对象**
    - **添加一个**bean，id**为：**target1。class**为：**com.oracle.csg.spring.FirstPenguin
    - **再添加一个**bean，id**为：**target2。class**为：**com.oracle.csg.spring.SecondPenguin
  - **定义前置增强**
    - **添加一个**bean，id**为：**greetAdvice。class**为：**com.oracle.csg.spring.GreetBeforeAdvice
  - **定义切面**bean
    - **添加一个**bean，id**为：**greetAdvisor。Class**为：**com.oracle.csg.spring.GreetAdvisor。
    - **在当前**bean**元素内通过**p**命名空间指定使用的增强。如下所示**
      - p:advice-ref="greetAdvice"
  - **定义公共配置信息**
    - **添加一个**bean，id**为：**parent。abstract**为：**true 。class**为：**org.springframework.aop.framework.ProxyFactoryBean
    - **在当前的**bean**元素中添加**p**命名空间设置代理对象的属性。如下所示：**
      - p:interceptorNames="greetAdvisor "**（表示使用**greetAdvisor**这个**bean**进行拦截）**
      - p:proxyTargetClass="true"
    - **定义两个代理对象**
      - **创建一个**bean，id**为：**first。parent**为：**parent。p:target-ref**为：**target1
      - **再创建**bean。id**为：** second。parent**为：**parent。p:target-ref**为：**target2

- **相关测试类已提供，请学员直接运行**Test**测试类，并观察运行结果。**

实验案例 002：静态正则表达式方法匹配切面

实现效果



实现步骤

1. **搭建环境**

- **将实验指导手册目录下的**”SpringDemo023”**导入到**eclipse**中。**

说明：

1. **在** SpringDemo023 **中已添加** spring **的相关** jar **包的引用(** **如** jar **包路径不正确,请学员重新引入** spring **的相关** jar **包**)
2. **在“** SpringDemo023 **”中，在**com.oracle.csg.spring**包下有**FirstPenguin**类，**FirstPenguin**类中有**Sleep**和**Beat**方法.**
3. **在**com.oracle.csg.spring**包下有**SecondPenguin**类。**SecondPenguin**类中有**Sleep**方法。**

2. **静态普通方法名匹配切面：**

- **在**com.oracle.csg.spring**包下创建**GreetBeforeAdvice**类，并且实现** MethodBeforeAdvice**接口。在**GreetBeforeAdvice**类中重写接口中的** before(Method method, Object[] args, Object arg2)**方法：**
  - **编写一条输出语句，将字符串“吃饭！”打印输出在控制台。**
- **编写配置文件：**
  - **打开**src**目录下的**applicationContext.xml**文件**
  - **通过如下代码引入**p**命名空间**
    - xmlns:p=”http://www.springframework.org/schema/p”
  - **定义目标对象**
    - **添加一个**bean，id**为：**target1。class**为：**com.oracle.csg.spring.FirstPenguin
    - **再添加一个**bean，id**为：**target2。class**为：** com.oracle.csg.spring.SecondPenguin
  - **定义前置增强**
    - **添加一个**bean，id**为：**greetAdvice。class**为：** com.oracle.csg.spring.GreetBeforeAdvice
  - **定义静态正则表达式方法匹配切面**bean
    - **添加一个**bean，id**为：**regexAdvisor。Class**为：** org.springframework.aop.support.RegexpMethodPointcutAdvisor。
    - **在当前**bean**元素内通过**p**命名空间指定使用的增强。如下所示**

- p:advice-ref="greetAdvice"
- **在当前**bean**元素内通过**p**命名空间指定匹配规则.如下所示**
  - p:pattern=".*Sleep.*"(**表示所有包下的包含**Sleep**的类**)
- **定义公共配置信息**
  - **添加一个**bean**，**id**为：**parent**。**abstract**为：**true **。**class**为：** org.springframework.aop.framework.ProxyFactoryBean
  - **在当前的**bean**元素中添加**p**命名空间设置代理对象的属性。如下所示：**
    - p:interceptorNames="regexAdvisor"**（表示使用**regexAdvisor**这个**bean**进行拦截）**
    - p:proxyTargetClass="true"
  - **定义两个代理对象**
    - **创建一个**bean**，**id**为：**first**。**parent**为：**parent**。**p:target-ref**为：**target1
    - **再创建**bean**。**id**为： **second**。**parent**为：**parent**。**p:target-ref**为：**target2
- **相关测试类已提供，请学员直接运行**Test**测试类，并观察运行结果。**

实验十、业务层整合（一）

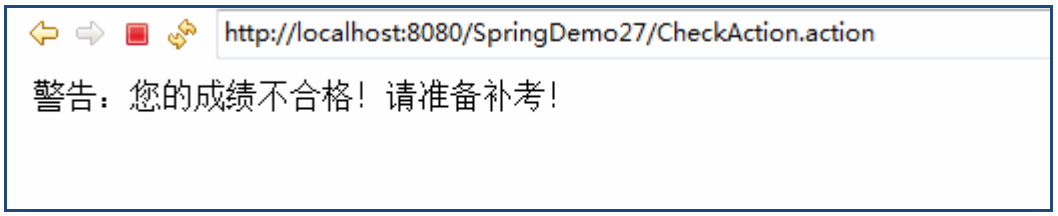
【实验目的】

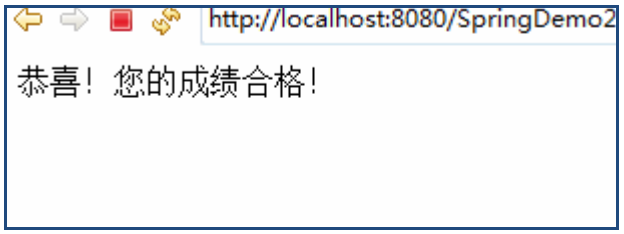
1. **整合** struts2 **框架**

【实验内容】

实验案例 001：整合 struts2 框架

实现效果





实现步骤

1. **搭建环境**

- **将实验指导手册目录下的**”SpringDemo027”**导入到**eclipse**中。**

说明：

1. **在** SpringDemo027 **中已添加** spring **的相关** jar **包的引用（ 如** jar **包路径不正确,请学员重新引入** spring **的相关** jar **包** ）
2. **在“** SpringDemo027 **”中，在**com.oracle.csg.spring**包下有**Student**类，**Student**类中科目**(subject)**和成绩**(subject)**属性,并已为其添加**set、get**方法。**
3. **在**com.oracle.csg.spring**包下有**StudentService**类。**StudentService**类中有**check（）方法，该方法中对学生的**score**进行判断。**60**分一下的方法返回**false，60**分以上的返回** true
4. **相关的**jsp**页面已为学员完成**

1. **整合** struts2 **框架：**

- **对**web.xml**进行配置**
  - **打开**WebContent/WEB-INF/web.xml**配置文件，在**web.xml**文件中加入**struts2**的过滤器说明**

- 添加<filter>元素。并在<filter>元素下添加<filter-name>和<filter-class>元素。值分别为如下：
  - struts2
  - org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
- 添加<filter-mapping>元素，在<filter-mapping>元素下添加<filter-name>和<url-pattern>元素。值分别为：
  - struts2 (需与filter元素内的filter-name一致)
  - /*

- 添加<context-param>元素用于指定配置文件
  - 在<context-param>元素下添加<param-name>和<param-value>元素。值分别如下所示
    - contextConfigLocation
    - /WEB-INF/classes/applicationContext.xml
- 添加<listener>元素，使用ContextLoaderListener初始化容器
  - 在<listener>元素下添加<listener-class>元素，值如下所示
    - org.springframework.web.context.ContextLoaderListener

- 实现控制器逻辑
  - 在com.oracle.csg.spring包下创建CheckAction类，继承ActionSupport
    - 在CheckAction类下添加两个私有的属性，并为student添加set、get方法，为service添加set方法
      - student 类型：Student
      - service 类型：StudentService
    - 在当前类下添加execute()方法。访问权限：public。 返回类型：String。并通过throws Exception抛出异常信息。
    - 实现execute()方法
      - 调用service的check（）方法，将“ student.getSubject() ”和“ student.getScore() ”作为参数传入。返回布尔类型的对象isPass。
      - 如果isPass为false，则方法返回“ error ”
      - 否则方法返回“ success “
  - 配置struts.xml配置文件
    - 打开src目录下的struts.xml配置文件，在<struts>元素下添加<package>元素,定义package。<package>元素内的属性如下：
      - name：default
      - extends：struts-default
    - 在<package>元素下添加<action>元素。<action>元素内的属性如下
      - name：CheckAction （这里的name需要和form.jsp中表达的action属性一致）
      - class：checkAction （自定义为checkAction）
    - 在<action>元素下添加两个<result>元素。属性分别如下所示
      - name为：success 值为：congratulation.jsp
      - name为：error 值为：warn.jsp
- 配置applicationContext.xml文件
  - 打开src目录下的applicationContext.xml文件。创建一个bean，定义业务逻辑组件

        - id：studentService
        - class：com.oracle.csg.spring.StudentService
    - 创建第二个bean，定义控制逻辑组件，属性值注入业务逻辑组件
        - id：checkAction （此处id的值需与struts.xml配置文件中<action>元素中的class一致）
        - class：com.oracle.csg.spring.CheckAction
    - 在当前bean下添加<property>元素（在CheckAction类中声明了service变量，该变量依赖于studentService这个bean）
        - name：service
        - ref：studentService
- 运行项目，在地址栏输入：http://localhost:8080/SpringDemo27/form.jsp。在文本框内输入科目和成绩（数学 59）。运行程序，观察运行结果。科目不变，将成绩改为80.运行程序，观察运行结果。
- 指定自动装配
    - 打开struts.xml配置文件，在<action>元素内的 class的值改为：com.oracle.csg.spring.CheckAction
    - 打开applicationContext.xml配置文件，删除或注释掉“定义控制逻辑组件，属性值注入业务逻辑组件”的相关代码。
- 运行项目，在地址栏输入：http://localhost:8080/SpringDemo27/form.jsp。在文本框内输入科目和成绩（数学 59）。运行程序，观察运行结果。科目不变，将成绩改为80.运行程序，观察运行结果。
- 自动装配其他方式（一）
    - 在src目录下创建一个struts.properties文件，在该文件内添加如下代码
        - struts.objectFactory.spring.autoWire=type
    - 将applicationContext.xml配置文件中bean的id：studentService改为studentService1。
    - 运行程序，输入测试数据，观察运行结果。
- 自动装配其他方式（二）
    - 将struts.properties文件中的代码通过“##”注释掉。
    - 打开struts.xml文件，在<package>元素上面添加一个<constant>元素。用于定于常量
        - name：struts.objectFactory.spring.autoWire
        - value：type
    - 运行程序，输入测试数据，观察运行结果。

实验十一、业务层整合（二）

【实验目的】

1. spring **中使用** Quartz

【实验内容】

实验案例 001：在 spring 中使用 Quartz

实现效果

simpleTrigger

```
log4j:WARN No appenders could be found for logger (org.springframework.core.env.Stan
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Hello!Use Quartz in Spring!Tue Mar 03 16:55:08 CST 2015
Hello!Use Quartz in Spring!Tue Mar 03 16:55:11 CST 2015
Hello!Use Quartz in Spring!Tue Mar 03 16:55:14 CST 2015
Hello!Use Quartz in Spring!Tue Mar 03 16:55:17 CST 2015
```

cronTrigger

```
log4j:WARN No appenders could be found for logger (org.springframewo
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig
Hello!Use Quartz in Spring!Tue Mar 03 16:56:40 CST 2015
Hello!Use Quartz in Spring!Tue Mar 03 16:56:44 CST 2015
Hello!Use Quartz in Spring!Tue Mar 03 16:56:48 CST 2015
Hello!Use Quartz in Spring!Tue Mar 03 16:56:52 CST 2015
Hello!Use Quartz in Spring!Tue Mar 03 16:56:56 CST 2015
Hello!Use Quartz in Spring!Tue Mar 03 16:57:00 CST 2015
Hello!Use Quartz in Spring!Tue Mar 03 16:57:04 CST 2015
Hello!Use Quartz in Spring!Tue Mar 03 16:57:08 CST 2015
```

实现步骤

4. **搭建环境**

- **将实验指导手册目录下的**”SpringDemo031”**导入到**eclipse**中。**

说明：

5. **在** SpringDemo031 **中已添加** spring **的相关** jar **包的引用(** **如** jar **包路径不正确,请学员重新引入** spring **的相关** jar **包**)
6. **在** com.oracle.csg.spring **包下已创建类：**Service**。在该类中有一个** Hello() **方法,该方法中已添加一条输出语句**:

a) Hello!Use Quartz in Spring!"+new Date()

5. 创建 JobDetail ：

- 打开在applicationContext.xml配置文件。在配置文件中为Service类创建bean。
  - 创建一个bean元素。元素属性如下：
    - id：service
    - class：com.oracle.csg.spring.Service
- 创建一个bean元素。元素属性如下：
  - id：jobDetail
  - class：org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean
- 在当前bean元素内通过p命名空间配置属性。（提供的applicationContext.xml配置文件内已为学员引入了p命名空间的引用）
  - 通过“p:targetObject-ref”指定对“service”这个bean进行管理
  - 通过“p:targetMethod"”定义方法名：service。
  - 通过设置“p:concurrent”的值为false，指定多个job不会并发运行

6. 创建 Trigger

- 创建simpleTrigger
  - 创建一个bean元素。bean的属性如下
    - id：simpleTrigger
    - class：org.springframework.scheduling.quartz.SimpleTriggerFactoryBean
    - p:jobDetail-ref：jobDetail
    - p:startDelay：2000 （启动的延迟时间）
    - p:repeatInterval：3000 （间隔时间）
    - p:repeatCount：3 （重复次数）
- 创建CronTrigger
  - 创建一个bean元素，bean元素内的属性如下：
    - id：cronTrigger
    - class：org.springframework.scheduling.quartz.CronTriggerFactoryBean
    - p:jobDetail-ref：jobDetail
    - p:cronExpression：0/3 * * * * ? （设置每隔3秒触发一次）
- 创建schedualer
  - 创建bean元素，bean元素内的属性如下：
    - id：scheduler
    - class：org.springframework.scheduling.quartz.SchedulerFactoryBean
  - 在当前bean元素下添加property元素。元素内的属性如下
    - name: triggers
  - 在property元素下添加list元素。在list元素内添加ref元素。ref元素内的属性如下：
    - bean：simpleTrigger
- 打开Test测试类，通过如下代码加载配置文件
  - ApplicationContext ctx=new ClassPathXmlApplicationContext("applicationContext.xml");
- 调用ctx的getBean（String arg）方法，将“schedual”作为参数传入，返回Schedual类型对象sch（需要进行类型转换）

- 调用sch的start（）方法，调度启动。
- 运行Test测试类，观察运行结果。
- 打开applicationContext.xml配置文件，将id为scheduler的bean中property元素下的list元素下的ref元素中bean的值改为cronTrigger。
- 运行Test测试类。观察运行结果。

实验十二、Spring 整合 Hibernate 应用开发（一）

【实验目的】

1. 在 spring 中使用 hibernate

【实验内容】

实验案例 001：hibernate 监听器

实现效果

控制台运行结果截图

```
2015-3-6 14:56:15 org.hibernate.impl.SessionFactoryObjectFactory addInstanc
信息: Not binding factory to JNDI, no JNDI name configured
SaveOrUpdate Entity className:class com.oracle.csg.spring.entity.Empl
JACK
Sale
```

Mysql 数据库数据截图

```
mysql> select * from empl;
+----+------+------+------+
| id | name | sex  | dept |
+----+------+------+------+
|  1 | MIKE | male | Sale |
|  2 | JACK | fale | Sale |
+----+------+------+------+
```

实现步骤

1. 搭建环境

- 将实验指导手册目录下的”SpringDemo037”导入到eclipse中。

说明：

1. 环境工程包下有“ sql 语句 ”的记事本文档，里面内容为 sql 操作语句。将文档中的内容在 mysql 中运行。（如果本地数据库中已存在该表及该表中的数据可直接使用）
2. 在 SpringDemo038 中已添加 aopalliance、spring、mysql、dbcp、hibernate 等相关 jar 包的引用(如 jar 包路径不正确,请学员重新引入 jar 包):
3. 在 com.oracle.csg.spring.dao 包下有 EmplDao 接口，在该接口中存在 SaveOrUpdate（Empl empl）方法。
4. com.oracle.csg.spring.dao.impl 包下的 EmplDaoImpl 实现 EmplDao 接口，并实现接口中的方法
5. 在 src 目录下的 applicationContext.xml 配置文件中已定义了数据源、sessionFactory、hibernateTemplate 以及 EmplDaoImpl。

2. 在 Spring 中添加自定义 hibernate 监听器：

- 在com.oracle.csg.spring.listener包下 添加自定义监听器
  - 创建MyListener类，继承DefaultSaveOrUpdateEventListener
  - 在MyListener类中添加onSaveOrUpdate（）方法。
    入参：SaveOrUpdateEvent event
    返回类型：void 访问权限：public。通过“ throws HibernateException ”抛出异常信息。
  - 调用event的getObject()方法，再调用getClass()方法，获得监听对象的类。在其前面添加输出提示语句："SaveOrUpdate Entity className:"。如下所示，并将其在控制台输出。
    - "SaveOrUpdate Entity className:"+event.getObject().getClass())
  - 调用event的getObject（）方法，返回Object类型对象object
  - 通过“ instanceof ”将object与Empl进行类型比较。
    - 如果object为Empl类型的对象，将object对象进行类型强转，转化为Empl类型。返回Empl类型对象empl
    - 调用empl的getName()和getDept()方法，获得监听的员工对象的name和dept属性值，并将其在控制台输出。
  - 最后调用父类对应的方法，如下
    - super.onSaveOrUpdate(event);
- 打开配置文件applicationContext.xml文件，在配置文件中添加监听器的配置
  - 在sessionFactory的<bean>元素下添加<property>元素，元素内的属性name的值为：eventListeners
    - 在<property>元素下添加<map>元素
      - 在<map>元素下添加< entry >元素，<entry>的属性key的值为：、save-update
        - 在<entry>元素下添加<bean>元素。<bean>元素的属性class的值为：com.oracle.csg.spring.listener.MyListener（即我们之前创建的监听器）
- 打开Test测试类，在已有的代码基础上完成测试类
  - 实例化一个Empl对象empl。
  - 通过empl的set方法分别注入如下属性：
    - name：Jack
    - sex：fale
    - dept：Sale
  - 调用dao的SaveOrUpdate(Empl empl)方法，将empl作为参数传入。
- 运行Test测试类，观察控制台以及数据库的运行结果。

实验十三、Spring 整合 Hibernate 应用开发（二）

【实验目的】

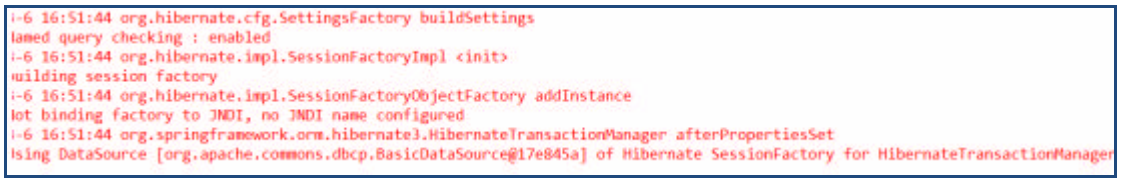
1. spring 对于事务支持

【实验内容】

实验案例 002：事务管理

实现效果

三次实验控制台运行结果都为如下截图



实现步骤

1. 搭建环境

- 将实验指导手册目录下的”SpringDemo038”导入到eclipse中。

说明：

1. 环境工程包下有“ sql 语句 ”的记事本文档，里面内容为 sql 操作语句。将文档中的内容在 mysql 中运行。（如果本地数据库中已存在该表及该表中的数据可直接使用）
2. 在 SpringDemo038 中已添加 aopalliance、spring、mysql、dbcp、hibernate 等相关 jar 包的引用(如 jar 包路径不正确,请学员重新引入 jar 包):
3. 在 com.oracle.csg.spring.dao 包下有 EmplDao 接口，在该接口中存在 SaveOrUpdate（Empl empl）方法。
4. com.oracle.csg.spring.dao.impl 包下的 EmplDaoImpl 实现 EmplDao 接口，并实现接口中的方法
5. 在 src 目录下的 applicationContext.xml 配置文件中已定义了数据源、sessionFactory、hibernateTemplate 以及 EmplDaoImpl。

- 打开applicationContext.xml配置文件，在配置文件中创建一个<bean>元素，用于定义事务管理器，<bean>元素的属性如下
  - id：transactionManager
  - name：org.springframework.orm.hibernate3.HibernateTransactionManager
  - 在当前<bean>元素下添加<property>元素，为事务管理注入sessionFactory。<property>元素的属性如下
    - name：sessionFactory
    - ref：sessionFactory

2. 通过 TransactionProxyFactoryBean 的声明式事务处理，添加 hibernate 事务管理：

- 在配置文件中创建一个<bean>元素，用于定义事务代理对象。元素内属性如下
  - id：emplDaoProxy
  - class：org.springframework.transaction.interceptor.TransactionProxyFactoryBean
  - 在当前<bean>元素下创建三个<property>元素，为其配置属性。各元素属性如下

- 第一个<property>:
  - name：proxyTargetClass
  - value:true
- 第二个<property>
  - name:：transactionManager
  - ref：transactionManager
- 第三个<property>
  - name：target
  - ref：emplDao

- 在当前<bean>元素下创建第四个<properyt>元素，定义事务属性
  - 在当前<properyt>元素下添加<props>元素。
    - 在<props>元素下添加<prop >元素。元素内属性如下
      - key：*　　（*表示emplDao下的所有方法）
    - <prop >元素的值：PROPAGATION_REQUIRED

- 运行Test1测试类，观察控制台运行结果。（控制台中将会出现如下提示信息，表示已经使用了事务管理，同时打开数据库查询empl表，数据库中已完成新增Jack的员工信息）

3. 通过 aop/tx 命名空间的声明式事务处理，添加 hibernate 事务管理

说明：在applicationContext.xml中，已引入了context、aop和tx命名空间。学员可直接使用以上命名空间

- 将之前"通过TransactionProxyFactoryBean 的声明式事务处理,添加hibernate事务管理"实验中的代码注释掉。
- 在applicationContext.xml中，创建<tx:advice>元素，定义事务增强，元素内的属性如下
  - id：txAdvice
  - transaction-manager：transactionManager
- 为上一步的增强定义增强属性。
  - 在<tx:advice>元素下添加<tx:attributes>元素。然后在<tx:attributes>元素下添加<tx:method >声明方法。属性如下
    - name：*
    - propagation：REQUIRED
  - 为emplDao增加一个切面，在<tx:advice>元素上一行添加<aop:config>元素。
  - 在<aop:config>元素下添加<aop:advisor>元素，属性如下
    - advice-ref：txAdvice
    - pointcut：execution(* com.oracle.csg.spring.dao.impl.EmplDaoImpl.*(..))
      （表示EmplDaoImpl类下的所有方法）
- 运行Test2测试类，观察控制台运行结果。（控制台中将会出现如下提示信息，表示已经使用了事务管理，同时打开数据库查询empl表，数据库中已完成新增Tom的员工信息）

4. 通过@Transactional 注解的声明式事务处理,，添加 hibernate 事务管理。

- 将之前“通过aop/tx命名空间的声明式事务处理，添加hibernate事务管理”实验中的代码注释掉

- **在**applicationContext**配置文件中通过如下元素使用注解声明事务**
  - <tx:annotation-driven/>
- **打开**com.oracle.csg.spring.dao.impl**包下的**EmplDaoImpl**类，在**EmplDaoImpl**类外面添加如下注解**
  - @Transactional(propagation=Propagation.REQUIRED)
- **运行**Test3**测试类，观察控制台运行结果。（控制台中将会出现如下提示信息，表示已经使用了事务管理，同时打开数据库查询**empl**表，数据库中已完成新增**Marry**的员工信息）**

实验十四、Spring 整合 Hibernate 应用开发（三）

【实验目的】

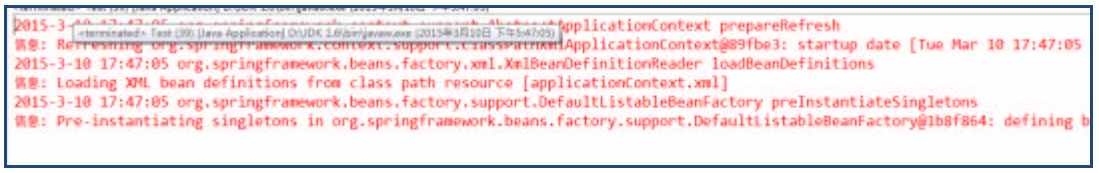
1. 配置 XML 声明式事务、配置注解声明事务

【实验内容】

实验案例 001：基于 tx/aop 命名空间的配置

实现效果

保存新员工是控制台运行结果都为如下截图



查询员工数量时,控制台运行结果截图

实现步骤

1. 搭建环境

- 将实验指导手册目录下的”SpringDemo041”导入到eclipse中。

说明：

1. 环境工程包下有“ sql 语句 ”的记事本文档，里面内容为 sql 操作语句。将文档中的内容在 mysql 中运行。（如果本地数据库中已存在该表及该表中的数据可直接使用）
2. 在 SpringDemo041 中已添加所需的 jar 包的引用(如 jar 包路径不正确,请学员重新引入 jar 包):
3. 在 com.oracle.csg.spring.dao 包下有 EmplDao 接口，在该接口中存在 save(Empl empl)方法和 count（）方法。
4. com.oracle.csg.spring.dao.impl 包下的 EmplDaoImpl 实现 EmplDao 接口，并实现接口中的方法

5. 在 src 目录下的 applicationContext.xml 配置文件中已定义了数据源(dataSource)、EmplDaoImpl 以及 jdbc 的事务管理器（transactionManager）。

2. 实施事务增强服务接口

- 在com.oracle.csg.spring.service包下创建服务接口EmplService,。
  - 在该接口中添加countEmpl()方法，返回类型：Integer　访问权限：public
  - 在该接口中添加save(Empl empl)方法，返回类型void　访问权限：public
- 在com.oracle.csg.spring.service.impl包下创建服务接口EmplService的实现类EmplServiceImpl。具体实现如下
  - 定义一个EmplDao类型属性emplDao，访问权限为：private。为其添加set方法。
  - 实现countEmpl()方法
    - 调用emplDao的countEmpl()方法，并将其作为方法的返回值
  - 实现save(Empl empl)方法
    - 调用emplDao的save(Empl empl)方法，将empl作为参数传入。

1. 基于 tx/aop 命名空间的配置：

- 在配置文件中创建一个<bean>元素，用于指定要织入事务增强的业务Bean。<bean>元素内属性如下
  - id：emplService
  - class：com.oracle.csg.spring.service.impl.EmplServiceImpl
- 在<bean>元素下添加< property>元素，为EmplServiceImpl注入emplDao属性。< property>元素属性如下
  - name：emplDao
  - ref：emplDao
- 使用tx命名空间定义事务增强

  说明：在配置文件中已引入”aop” ”tx”的命名空间，在以下的实验中可直接使用aop和tx明命名空间
  - 在当前配置文件中创建一个< tx:advice>元素，定义事务增强。元素内的属性如下
    - id：txAdvice
    - transaction-manager：transactionManager
  - 在< tx:advice>元素下添加<tx:attributes>元素
  - 在<tx:attributes>元素下添加<tx:method>元素，添加事务属性。事务属性如下
    - name：count*　（以count开头的所有方法）
    - read-only：true　（只读）
    - propagation：SUPPORTS　（设置传播行为类型。Support表示支持当前事务，如果当前没有事务，就以非事务方式执行）
  - 在当前<tx:attributes>元素下添加第二个<tx:method>元素，添加事务属性。事务属性如下
    - name：save*　（以save开头的所有方法）
    - rollback-for：Exception　（回滚规则设置为Exception类型的异常进行回滚）
    - propagation：REQUIRED（设置传播行为类型。REQUIRED表示如果当前没有事务，就新建一个事务，如果已经存在一个事务中，加入到这个事务中。）
- 通过aop应用增强到切点指定的范围

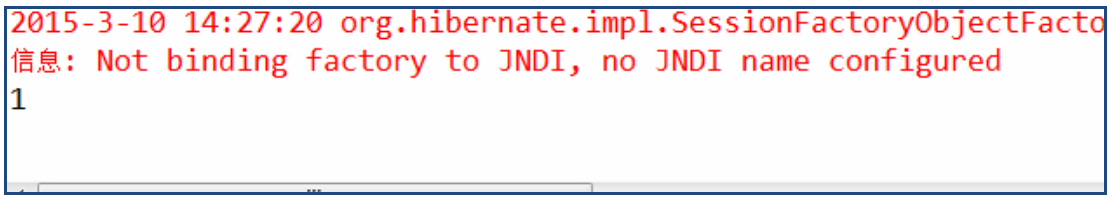
- 创建<aop:config>元素，在<aop:config>元素下创建<aop:pointcut>元素。<aop:pointcut>元素内的属性如下
  - expression：execution(* com.oracle.csg.spring.service.impl.*Impl.*(..))（表示在com.oracle.csg.spring.service. impl包下的所有以Impl结尾的方法）
  - id：txPointcut
- 在<aop:config>元素下创建<aop:advisor>元素，<aop:advisor>元素内的属性如下
  - advice-ref：txAdvice
  - pointcut-ref：txPointcut

- 在Test测试类中，在已给代码的基础上编写测试代码
  - 调用ctx的getBean(String arg)，将emplService作为参数传入，返回EmplService类型对象emplService。（此处需要进行类型强转）
  - 实例化Empl对象empl
  - 通过empl的set方法对empl的name、dept、set的属性进行赋值。
    - name：Salar
    - sex：female
    - dept：Sale
  - 调用service的save(empl)方法
- 运行Test测试类，观察运行结果。并打开mysql数据库，查询empl表内的数据
- 调用service的count()方法，运行Test测试类，观察运行结果。

实验案例 002：TransactionProxyFactoryBean

实现效果

控制台运行结果都为如下截图



实现步骤

1. 搭建环境

- 将实验指导手册目录下的”SpringDemo040”导入到eclipse中。

说明：

1. 环境工程包下有“ sql 语句 ”的记事本文档，里面内容为 sql 操作语句。将文档中的内容在 mysql 中运行。（如果本地数据库中已存在该表及该表中的数据可直接使用）
2. 在 SpringDemo040 中已添加所需的 jar 包的引用(如 jar 包路径不正确,请学员重新引入 jar 包):

3. 在 com.oracle.csg.spring.dao 包下有 EmplDao 接口，在该接口中存在 countEmpl（）方法。
4. com.oracle.csg.spring.dao.impl 包下的 EmplDaoImpl 实现 EmplDao 接口，并实现接口中的方法
5. 在 src 目录下的 applicationContext.xml 配置文件中已定义了数据源(dataSource)、EmplDaoImpl 以及 jdbc 的事务管理器（transactionManager）。

2. 实施事务增强服务接口

- 在com.oracle.csg.spring.service包下创建服务接口EmplService，在该接口中添加countEmpl()方法，返回类型：Integer　　访问权限：public
- 在com.oracle.csg.spring.service.impl包下创建服务接口EmplService的实现类EmplServiceImpl。具体实现如下
  - 定义一个EmplDao类型属性emplDao，访问权限为：private。为其添加set方法。
  - 实现countEmpl()方法
    - 调用emplDao的countEmpl()方法，并将其作为方法的返回值

3. 通过 TransactionProxyFactoryBean 对业务类进行代理,植入事务增强功能：

- 在配置文件中创建一个<bean>元素，定义代理的目标对象。<bean>元素内属性如下
  - id：emplService
  - class：com.oracle.csg.spring.service.impl.EmplServiceImpl
- 在<bean>元素下添加< property>元素，为EmplServiceImpl注入emplDao属性。< property>元素属性如下
  - name：emplDao
  - ref：emplDao
- 在配置文件中创建一个<bean>元素，用于定义事务代理对象。元素内属性如下
  - id：proxyServcebean
  - class：org.springframework.transaction.interceptor.TransactionProxyFactoryBean
  - 在当前<bean>元素下创建三个<property>元素，为其配置属性。各元素属性如下
    - 第一个<property>，定义需要引用的事务管理器:
      - name：transactionManager
      - ref：transactionManager
    - 第二个<property>，定义目标对象
      - name:：target
      - ref：emplService
    - 第三个<property>，属性如下。
      - name：transactionAttribute
    - 在第三个<property>元素下添加<props>元素，在<props>元素下添加一个<prop>元素，对目标对象的业务方法进行事务属性的设置
      - <prop>元素属性和值如下
        - key：count*　　（表示count开头的所有方法）
        - <prop>元素的值为：PROPAGATION_REQUIRED,readOnly
- 打开Test测试类，在已给代码基础上编写测试类。
  - 调用ctx的getBean(String arg)方法，将"proxyServce"作为参数传入，获取代理对象。返回EmplService类型对象service（需要进行类型强转）

- 调用service的countEmpl()方法，并将其结果在控制台输出

- 观察运行结果。