## 实验八 Spring框架应用基础

### 一、实验目的

1、掌握Spring框架的基本结构和基本用法；

2、掌握Spring配置文件的编写方法；

3、理解并掌握Spring框架中IOC、DI、容器等核心概念，Spring事务管理与任务调度；

重点：Spring控制反转IOC

### 二、实验内容与要求

1、掌握Spring框架的部署和使用方法，验证Spring配置信息的作用；

2、利用Spring实现对Bean的管理；

2、基于Spring框架，实现基于Spring框架的图书管理应用：

（1）能够将用户和图书数据从数据库中取出

（2）实现用户登录成功后进入主页面

（3）实现对图书信息的查询、增加、删除和修改功能

### 三、实验步骤（含核心参考代码与配置文件）

**1、基于Spring对Bean的管理方法介绍**

Bean 的实例化方法。Spring 容器负责对象的实例化，对象生命周期的管理，被 Spring管理的对象称之为 Bean。

（ 1）通过构造方法实例化。设计 Course 类：

public class Course {

private int id;

private String courseName;

//setter 方法， 有参与无参构造方法， toString()方法

}

Spring 配置文件 applicationContext.xml：

<!-- 定义第 1 个 Bean,通过无参构造方法实例化 -->

<bean id="course1" class="cs2014.pojo.Course" >

</bean>

<!-- 定义第 2 个 Bean,通过 setter 方法实例化 -->

<bean id="course2" class="cs2014.pojo.Course" >

<!-- 为 setter 方法提供参数 -->

<property name="id">

<value>201</value>

</property>

<property name="courseName">

<value>C++ Prime</value>

</property>

</bean>

测试的 Test.java

public static void main(String []args) {

Course course = null;

ApplicationContext ctx=

new FileSystemXmlApplicationContext("src/applicationContext.xml");System.out.println("====第 1 个 Bean,通过无参构造方法实例化====");

course = (Course) ctx.getBean("course1");

System.out.println(course);

System.out.println("====第 2 个 Bean,通过构造方法实例化====");

course = (Course) ctx.getBean("course2");

System.out.println(course);

}

[注]测试中输出关于 log4j 的警告，可以在 src/目录下复制一份课程示例中的 log4j,properties

文件，作为 log4j 的配置文件，并请查问相关资料，正确配置文件。

(2) 通过静态工厂方法实例化。修改 Course 类，添加以下方法：

public static Course createCourse(){

System.out.println("invoke createCourse()");

return new Course();

}

修改 Spring 配置文件 applicationContext.xml：

<!-- 定义第 3 个 Bean,通过静态工厂方法实例化 -->

<bean id="course3" class="cs2014.pojo.Course"

factory-method="createCourse">

</bean>

测试的 Test.java 的 main（）方法中：

System.out.println("====第 3 个 Bean,通过静态工厂方法实例化 ====");

course = (Course) ctx.getBean("course3");

System.out.println(course);

（ 3） 通过非工厂方法实例化

创建 CourseFactory 类:

package cs2014.factory;

public class CourseFactory {

public Course createCourse(){

System.out.println("invoke CourseFactory createCourse()");

return new Course(203,"Java Programming");//对象已经初始化

}

}

修改 Spring 配置文件 applicationContext.xml：

<!-- 定义第 4 个 Bean,通过工厂方法实例化 -->

<bean id="course4" class="cs2014.pojo.Course"

factory-method="createCourse"

factory-bean="courseFactory">

</bean>

<bean id="courseFactory"

class="cs2014.factory.CourseFactory"><!-- 属性 id=203 -->

</bean>

测试的 Test.java 的 main（）方法中：

System.out.println("====第 4 个 Bean,通过非静态工厂方法实例化====");

course = (Course) ctx.getBean("course4");

System.out.println(course);

2、 Bean 的注入方法

（ 1）通过 setter 注入。设计以下接口及实现类：

public interface Human {

void speak();

}

public interface Language {

public String kind();

}

public class Chinese implements Human{

private Language lan;

public void speak() {

System.out.println(lan.kind());

}

public void setLan(Language lan) {

this.lan = lan;

}

}

public class English implements Language{

public String kind() {

return "中国人也会说英语！ ";

}

}

修改 Spring 配置文件 applicationContext.xml：

<!-- 定义第 5 个 Bean，注入 Chinese 类对象 -->

<bean id="chinese" class="cs2014.pojo.Chinese">

<!-- property 元素用来指定需要容器注入的属性， lan 属性需要容器注入， ref 就指

向 lan 注入的 id -->

<property name="lan" ref="english"></property>

</bean>

<!-- 注入 English -->

<bean id="english" class="cs2014.pojo.English"></bean>

测试的 Test.java 的 main（）方法中：System.out.println("====第 5 个 Bean,通过非静态工厂方法实例化 ====");

Human human = null;

human = (Human) ctx.getBean("chinese");

human.speak();

（ 2） 构造注入。修改 Chinese.java

public class Chinese implements Human{

private int age;

private Language lan;

public Chinese(){};

// 构造注入所需要的带参数的构造函数

public Chinese(Language lan , int age){

this.lan=lan; this.age = age;

}

public void speak() {

System.out.println(lan.kind()+”\nage=” +age);

}

}

配置文件修改：

<bean id="chinese" class=" cs2014.pojo.Chinese">

<!-- 使用构造注入，为 Chinese 实例注入 Language 实例-->

<constructor-arg ref="english"></constructor-arg>

<!-- 使用构造注入，为 Chinese 实例注入属性 age-->

<constructor-arg index=“1">

<value>21</value>

</constructor-arg>

</bean>

<!-- 注入 English -->

<bean id="english" class=" cs2014.pojo.English"></bean>

测试略。

（ 3） Map、 List 的注入。创建 Student 类：

public class Student {

private int id;

private String name;

private Map<Integer,Course> courses1;

private List<Course> courses2;

//setter 方法、构造方法、 toString 方法

public String toString() {

return "[" + id + ", " + name + "," + courses1 + "]";

}public String toString2() {

return "[" + id + ", " + name + "," + courses2 + "]";

}

}

修改配置文件， bean 为 course2,course4 是前面已经配置生成的 bean：

<!-- 定义 Student 类 Bean，注入 Map 类对象属性 -->

<bean id="student" class="cs2014.pojo.Student">

<!-- property 元素用来指定需要容器注入的属性 -->

<property name="id">

<value>20140301</value>

</property>

<property name="name">

<value>华为</value>

</property>

<!-- map 属性注入 -->

<property name="courses1">

<map>

<entry>

<key>

<value>201</value>

</key>

<ref bean="course2"></ref>

</entry>

<entry>

<key>

<value>203</value>

</key>

<ref bean="course4"></ref>

</entry>

</map>

</property>

<!-- list 属性注入 -->

<property name="courses2">

<list>

<ref bean="course2"></ref>

<ref bean="course4"></ref>

</list>

</property>

</bean>

测试：

System.out.println("=====第 6 个 Bean,为 Map 类和 List 属性注入 =====");

System.out.println("=================Map 属性=============");

Student student1 = (Student)ctx.getBean("student");

System.out.println(student1);System.out.println("=================List 属性=============");

System.out.println(student1.toString2());

### 四、知识扩展

1、 IOC与DI

（1）什么是IOC与DI？

（2）请指出实验中的代码有哪些依赖？

（3） 请指出实验中的spring配置文件中使用哪两种注入方式？它们分别会调用什么方法？

（4） 如何利用Spring中提供依赖注入方式将第2步中的各个类进行解耦？请参照第3步实现，并注意，如果属性是对象的引用请将Spring配置文件中的value改为ref。