前言 Spring 3.0 准备

Spring 帮助文档

spring-framework-2.5.6-with-dependencies\spring-framework-2.5.6\docs\reference\html\_single\index.html

spring2.5.6 jar：

spring-framework-2.5.6-with-dependencies\spring-framework-2.5.6\dist\ spring.jar

commons-logging jar:

spring-framework-2.5.6-with-dependencies\spring-framework-2.5.6\lib\jakarta-commons\ commons-logging.jar

Annotation @Resource 需要的 commons-annotations.jar

spring-framework-2.5.6-with-dependencies\spring-framework-2.5.6\lib\j2ee\ commons-annotations.jar

Annotation AOP 自动动态代理 需要的aspectjrt.jar 和aspectjweaver.jar

spring-framework-2.5.6-with-dependencies\spring-framework-2.5.6\lib\\aspectj\aspectjrt.jar

spring-framework-2.5.6-with-dependencies\spring-framework-2.5.6\lib\\aspectj\aspectjweaver.jar

Annotation AOP 自动动态代理 用在Service 层（用在没有实现任何接口的类上）需要的cglib-nodep-2.1\_3.jar

spring-framework-2.5.6-with-dependencies\spring-framework-2.5.6\lib\cglib\cglib-nodep-2.1\_3.jar

Spring DataSource 数据库连接池需要的jar文件

spring-framework-2.5.6-with-dependencies\spring-framework-2.5.6\lib\jakarta-commons\commons-dbcp.jar

spring-framework-2.5.6-with-dependencies\spring-framework-2.5.6\lib\jakarta-commons\commons-pool.jar

数据库连接驱动： mysql-connector-java-5.1.6-bin.jar

工作空间编码设置：GBK

知识点1

1 知识点：IOC

模拟Spring

2 代码工程ES0100：

Spring\_0100\_AbstractOrientedProgranming

3 详解：

1\_) 模拟读取 xml 文件(jdom 读取 xml文件)

添加jdom jar -------- 路径：jdom-1.1.1\jdom\build\jdom.jar

2\_) 模拟spring 面向抽象编程

<beans>

<bean id="userdfo" class="com.baishui.dao.impl.UserDaoForOracleImpl"></bean>

</beans>

读取bean.xml配置文件内容编程

public ClassPathXmlApplicationContext() throws Exception{

SAXBuilder sb=new SAXBuilder();

Document doc=sb.build(this.getClass().getClassLoader().getResourceAsStream("bean.xml")); //构造文档对象

Element root=doc.getRootElement(); //获取根元素

List list=root.getChildren("bean");//取名字为bean的所有元素

for(int i=0;i<list.size();i++){

Element element=(Element)list.get(i);

String id=element.getAttributeValue("id"); //取bean子元素id

String clazz=element.getAttributeValue("class");//取bean子元素class

System.out.println("id:"+id);

System.out.println("class:"+clazz);

Object o = Class.forName(clazz).newInstance(); //反射

beans.put(id, o);

}

}

知识点1.1

1 知识点：IOC

模拟Spring ( IOC 控制反转 )

2 代码工程ES0110：

Spring\_0110\_AbstractOrientedProgranming

3 详解：

SAXBuilder sb=new SAXBuilder();

Documentdoc=sb.build(this.getClass().getClassLoader().getResourceAsStream("bean.xml")); //构造文档对象

Element root=doc.getRootElement(); //获取根元素

List list=root.getChildren("bean");//取名字为bean的所有元素

for(int i=0;i<list.size();i++){

Element element=(Element)list.get(i);

String id=element.getAttributeValue("id"); //取bean子元素class

String clazz=element.getAttributeValue("class");//取bean子元素class

System.out.println("id:"+id);

System.out.println("class:"+clazz);

Object o = Class.forName(clazz).newInstance(); //反射 创建对应类型的 对象

beans.put(id, o);

for(Element propertyElement : (List<Element>)element.getChildren("property")) {

String name = propertyElement.getAttributeValue("name"); //userDAO

String bean = propertyElement.getAttributeValue("bean"); //userdfo

Object beanObject = beans.get(bean);//UserDAOImpl instance

String methodName = "set" + name.substring(0, 1).toUpperCase() + name.substring(1);//拼出方法名

System.out.println("method name = " + methodName);

Method m = o.getClass().getMethod(methodName, beanObject.getClass().getInterfaces()[0]);//控制反转 将service的userDao 属性赋值

m.invoke(o, beanObject);//执行setUserDao方法

}

}

知识点2

1 知识点：IOC

Spring 基础配置 (IOC 依赖容器注入思想)

2 代码工程ES0200：

Spring\_0200\_IOC\_Introduction

3 详解：

1.添加需要的jar :spring2.5.6 jar commons-logging jar

2.配置spring xml配置文件

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean id="u" class="com.baishui.dao.impl.UserDaoForOracleImpl" />

<bean id="service" class="com.baishui.service.UserService">

<property name="userDao" ref="u" />

</bean>

<!-- more bean definitions go here -->

</beans>

(小技巧 添加API帮助文档： 在jar文件右键-JavaDoc Location ----添加Spring API

spring-framework-2.5.6-with-dependencies\spring-framework-2.5.6\docs\api)

3 调用Spring API

//BeanFactory applicationContext = new ClassPathXmlApplicationContext("beans.xml");

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("beans.xml");

//建议用 ApplicationContext 功能更强大 父接口是BeanFactory

UserService service = (UserService)applicationContext.getBean("service");

User u = new User();

u.setUsername("zhangsan");

u.setPassword("zhangsan");

service.addUser(u);

(小技巧 spring xml 自动提示:

a) window – preferences – myeclipse –xml catalog

b) User Specified Entries –add

i file system:

spring-framework-2.5.6-with-dependencies\spring-framework-2.5.6\dist\resources\spring-beans-2.5.xsd

ii key : <http://www.springframework.org/schema/beans/spring-beans-2.5.xsd>

iii key type : Namespace Name

)

知识点3

1 知识点：IOC

Spring IOC 注入类型配置

I set方法注入 （常用）

Ii 构造方法注入 （不重要）

Iii 接口注入 （不重要）

2 代码工程ES0300：

Spring\_0300\_IOC\_Injection\_Type

3 详解：

I 依赖set方法注入

<bean id="u" class="com.baishui.dao.impl.UserDaoForOracleImpl" />

<bean id="service" class="com.baishui.service.UserService">

<property name="userDao" ref="u"/> //依靠方法注入

</bean>

II 依赖 构造方法注入

<bean id="u" class="com.baishui.dao.impl.UserDaoForOracleImpl" />

<bean id="service" class="com.baishui.service.UserService">

<constructor-arg>

<ref bean="u"/> //构造方法注入

</constructor-arg>

</bean>

（注： 当多个属性的构造方法，同时注入时，根据配置序列依次注入）

<constructor-arg index="0">

<ref bean="u"/>

</constructor-arg>

<constructor-arg index="1" type="java.lang.String" value="ssss"> //第二个参数 直接用类型 赋值注入

</constructor-arg>

知识点4

1 知识点：IOC

Spring xml id 和 name 的区别

2 代码工程ES0400：

Spring\_0400\_IOC\_Id\_Name

3 详解：

1 用name 时的区别就是可以使用 其他的特殊字符

<bean name="u" class="com.baishui.dao.impl.UserDaoForOracleImpl" />

知识点5

1 知识点：IOC

简单类型的注入

2 代码工程ES0500：

Spring\_0500\_IOC\_SimpleProperty

3 详解：

为userDao 对象中的属性 id 和 userDaoStatus 赋值

Xml配置

<bean name="u" class="com.baishui.dao.impl.UserDaoForOracleImpl">

<property name="id" value="8"></property>

<property name="userDaoStatus" value="good"></property>

</bean>

UserDaoForOracleImpl

private int id;

private String userDaoStatus;

（注：简单类型属性的注入，同样是根据setXXX 方法 注入的，所以必须添加属性的 get set方法）

知识点6

1 知识点：IOC

Bean的生命范围 scope

2 代码工程ES0600：

Spring\_0600\_IOC\_Bean\_Scope

3 详解：

1.scope 的取值 ：

i Singleton (默认 单例，无论那多少次这个类型的对象 都是同一个)

ii Prototype （每次拿的是不同类型的对象）

iii( Request session global session 与web 结合时能用到)

验证

1 配置beans.xml 的 scope属性

<bean id="service" class="com.baishui.service.UserService" scope="prototype">

<property name="userDao" ref="u"/>

</bean>

2 判断两次拿到的对象是不是同一个

UserService service = (UserService)applicationContext.getBean("service");

UserService service2= (UserService)applicationContext.getBean("service");

System.out.println(service==service2);

结论：

当scope 默认时 打印 true,是同一对象， 当scope 设置为prototype时 打印的是false 不是同一对象

知识点7

1 知识点：IOC

Set List Map 等 Collection 的依赖注入

2 代码工程ES0700：

Spring\_0700\_IOC\_Collection

3 详解：

1.在UserDaoForOracleImpl类中分别创建 set list map 类型的 属性

private Set<String> sets;

private List<String> lists;

private Map<String, String> maps;

2 在beans.xml 配置 bean userDao 的属性，依赖注入

<bean name="u" class="com.baishui.dao.impl.UserDaoForOracleImpl">

<property name="sets">

<set>

<value>1</value>

<value>2</value>

</set>

</property>

<property name="lists">

<list>

<value>1</value>

<value>2</value>

<value>3</value>

</list>

</property>

<property name="maps">

<map>

<entry key="1" value="1"></entry>

<entry key="2" value="2"></entry>

<entry key="3" value="4"></entry>

</map>

</property>

</bean>

知识点8

1 知识点：IOC

属性自动注入 autowire (不常用，一般都是手动装配)

2 代码工程ES0800：

Spring\_0700\_IOC\_Collection

3 详解：

autowire 常用取值：

i byname : 根据属性名匹配注入，注此时 配bean的name一定要和对象的属性名称一致

ii bytype ：根据bean的类型注入，此时一定只有一个对应类型的bean配置，否则出错

iii default ：使用在全局 default-autowire指定好的装配类型

实践：

<bean name="userDao" class="com.baishui.dao.impl.UserDaoForOracleImpl" >

<property name="daoId" value="1"></property>

</bean>

<bean name="userDao2" class="com.baishui.dao.impl.UserDaoForOracleImpl">

<property name="daoId" value="2"></property>

</bean>

<bean id="service" class="com.baishui.service.UserService" scope="prototype" autowire="byname">

</bean>

知识点9

1 知识点：IOC

Bean 的 生命周期 lifecycle

2 代码工程ES0900：

Spring\_0900\_IOC\_lifecycle

3 详解：

1 lazy-init (初始化时机属性配置) : 当应用程序启动时，默认spring 会初始化所有配置好的bean,优化速度时可以用到 lazy-init=”true”,含义：当用到时在初始化， 全局的属性 default-lazy-init=”true”

2 init-method destroy-method （初始 和 销毁方法配置）

i在UserService 中 填入 init 和 destroy 方法

ii 在beans.xml中配置

<bean id="service" class="com.baishui.service.UserService" init-method="init" destroy-method="destroy">

<property name="userDao" ref="userDao"/>

</bean>

验证：

//验证init destroy 方法

ClassPathXmlApplicationContext applicationContext = new ClassPathXmlApplicationContext("beans.xml");

UserService service = (UserService)applicationContext.getBean("service");

applicationContext.destroy();

（注：不要与 scope=”prototype”一起使用， ClassPathXmlApplicationContext 不会监管prototype类型的bean,

Web应用时，web服务器会自动执行 applictioncontext 的 destroy 方法）

知识点10

1 知识点：IOC ----Annotation

IOC –Annotation 版本的 autowire 和 qualifier 不常用

2 代码工程ES1000：

Spring\_1000\_IOC\_Annotation\_autowire-qulifier

3 详解：

1 beans.xml 中引入Annotation 的配置（红色标记） (xsd文件引入)

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

http://www.springframework.org/schema/context

<http://www.springframework.org/schema/context/spring-context-2.5.xsd>">

<context:annotation-config/>

</beans>

2 配置 xml自动提示 xml catalog

3 不在xml中配置属性的注入，利用Annotation标注 @Autowire 实现注入（注意无参构造方法）

4 当xml配置了多个同种类型的bean 时，在属性的set方法上添加注解

@Qualifier(value=”userDao”) 含义：符合条件者，相当于byname

@Autowired

public void setUserDao(@Qualifier(value=”userDao”) UserDao userDao) {

this.userDao = userDao;

}

备注 ： @Autowire 属性required = false (默认true) 含义：必须的

知识点11

1 知识点：IOC ----Annotation

IOC –Annotation @Resource 引用资源注入（默认：bytype） 常用的

2 代码工程ES1100：

Spring\_1100\_IOC\_Annotation\_Resource

3 详解：

1.添加 Annotation @Resource 需要的 commons-annotations.jar

spring-framework-2.5.6-with-dependencies\spring-framework-2.5.6\lib\j2ee\ common-annotations.jar

2．在UserService类中 setUserDao方法上标注 @Resource

含义：应用资源注入，根据xml配置bean中名称相同的资源注入

（注：默认 按名称找对应的注入，名称找不到时，按类型注入）

实践：

@Resource(name="u") // 如果使用名称这样写

public void setUserDao(UserDao userDao) {

this.userDao = userDao;

}

不足之处是没有源代码时，不能用Annotation

知识点12

1 知识点：IOC ----Annotation

IOC –Annotation @Component 和 @Resource

组件 与 资源标注 实现注入

2 代码工程ES1200：

Spring\_1200\_IOC\_Annotation\_Component

3 详解：

1 配置 <context:component-scan base-package=”com.baishui”/> 向容器说明bean从那个路径自动扫描标注了@Component 的实体类，这样省略bean的配置

2 在标注后指定名字相当于在 beans.xml 中配置 <bean> 为对象指定名称

在UserService类上标注，并给对象起个名字

@Component(value="service")

在UserDaoForOracleImpl类上标注，并给对象起个名字

@Component(value="userDao")

在UserService的setUserDao方法上标注 根据对象名称注入

@Resource(name="userDao")

//调用时 与在xml配置类似

UserService service = (UserService)applicationContext.getBean("service"); //根据标注命名取出对象

UserDaoForOracleImpl u1 =(UserDaoForOracleImpl)applicationContext.getBean("userDao");

//根据标注命名取出对象

System.out.println(u1.toString());

service.addUser(new User());

System.out.println(service.getUserDao());

（注： 1.根据对象名称取出来的对象 和 注入的 是同一个对象

2 如果没有为对象指定名称，默认就是类名小写作为对象的名称）

知识点13

1 知识点：IOC ----Annotation

A\_) IOC –Annotation 标注bean 的声明范围 @Scope

B\_) bean 等生命周期方法 @PostConstruct @PreDestroy

2 代码工程ES1300：

Spring\_1300\_IOC\_Annotation\_PrePost\_Scope

3 详解：

1 标注bean的范围：同 xml中一样，@Scope(value=”prototype”) 含义：原型 不同的

在UserService 上标注

@Scope(value="prototype")

2 标注 初始和销毁方法：同 xml :init-method destroy-method

@PostConstruct

public void init(){

System.out.println("UserService @PostConstruct init method...");

}

@PreDestroy

public void destroy(){

System.out.println("UserService @PreDestroy destroy method...");

}

验证：

ClassPathXmlApplicationContext applicationContext = new ClassPathXmlApplicationContext("beans.xml");

UserService service = (UserService)applicationContext.getBean("service"); //根据标注命名取出对象

UserService service1 = (UserService)applicationContext.getBean("service"); //根据标注命名取出对象

System.out.println(service==service1); //Scope = prototype 输入 false

applicationContext.destroy(); //执行 init 和 destroy 方法

知识点14

1 知识点：AOP

AOP初识 （AOP的实现原理，面向切面编程思想）

以实现添加日志的功能为例

2 代码工程ES1400：

Spring\_1400\_AOP\_Intoduction

3 详解：

当程序需要扩展业务逻辑时，为了扩展性，如果没有源码，有以下几种设计模式

以UserDao 的业务逻辑为例

A\_) 继承UserDaoForOracleImpl 类

//重写添加saveUser业务逻辑

public class UserDaoForOracleImpl2 extends UserDaoForOracleImpl{

@Override

public void saveUser(User u) {

System.out.println("save start...");

super.saveUser(u);

System.out.println("save over...");

}

}

B\_) 重新实现UserDao接口，添加之前的UserDaoForOracleImpl 成员属性，目的执行之前的saveUser方法，随意组合的设计思想

public class UserDaoForOracleImpl3 implements UserDao {

private UserDaoForOracleImpl userDaoForOracleImpl=new UserDaoForOracleImpl();

public void saveUser(User u) {

System.out.println("save user start...");

userDaoForOracleImpl.saveUser(u);//执行之前的业务逻辑不变

System.out.println("save user over...");

}

}

C\_) 通过动态代理，实现添加功能模块，优点 ： 多个 模块中都需要添加 同一个 共同的功能时非常方便

i 模拟Spring AOP 创建动态代理控制类

public class LogInterceptor implements InvocationHandler {

private Object target; //声明 动态代理对象

public Object getTarget() {

return target;

}

public void setTarget(Object target) {

this.target = target;

}

public void beforeMethod(){

System.out.println("需要公共添加的业务逻辑功能块...");

}

public Object invoke(Object proxy, Method method, Object[] args)

throws Throwable {

beforeMethod();

method.invoke(target, args);

return null;

}

}

ii 实现动态代理(JDK 动态代理 模拟 Spring AOP)

//创建一个需要代理的对象

UserDao userDAO = new UserDaoForOracleImpl();

//把需要代理的对象交给动态代理控制器

LogInterceptor li = new LogInterceptor();

li.setTarget(userDAO);

//得到一个动态代理对象

UserDao userDAOProxy = (UserDao)Proxy.newProxyInstance(userDAO.getClass().getClassLoader(), userDAO.getClass().getInterfaces(), li);

System.out.println(userDAOProxy.getClass());

//通过动态代理对象 执行相应的方法时，就会添加上 共同要执行的方法

userDAOProxy.saveUser(new User());

userDAOProxy.deleteUser(new User());

用途： i 做日志 ii 业务逻辑功能添加 iii 检查效率 iiii 事务处理

知识点15

1 知识点：AOP

AOP的annotation实现

2 代码工程ES1500：

Spring\_1500\_AOP\_Annotation

3 详解：

1.添加Annotation AOP 自动动态代理 需要的aspectjrt.jar 和aspectjweaver.jar

2在beans.xml添加 AOP xsd 配置规范 文件的配置(红色标注)

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

<http://www.springframework.org/schema/context> http://www.springframework.org/schema/context/spring-context-2.5.xsd

<http://www.springframework.org/schema/aop> http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

3 自动提示，和以前的一样，配置CataLog

4 添加<aop:aspectj-autoproxy></aop:aspectj-autoproxy> 标签 （执行自动代理）

5 创建动态代理类，标注@Aspect 含义：切面

@Aspect // 标注切面，

@Component //Spring 管理初始化

public class LogInterceptor {

//用在对象的方法前执行，注意对象必须是被spring管理起来的类

//before 的植入点语法,指定加入执行哪个类的哪个方法上

@Before("execution(public void com.baishui.dao.impl.UserDaoForOracleImpl.saveUser(com.baishui.model.User))")

public void before(){

System.out.println("method start...");

}

}

拓展： spring aop 其他的 植入点语法 （参考Spring 的帮助文档：[6. Aspect Oriented Programming with Spring](file:///D:\baishui_C\我的文件夹\开发视频\Spring3.0\尚学堂马士兵_Spring_00_项目源码\soft\spring-framework-2.5.6-with-dependencies\spring-framework-2.5.6\docs\reference\html_single\index.html#aop#aop)）

//如果AOP 动态代理用在没实现接口的类上时，需要cglib 直接操作二进制码，来实现aop

\spring-framework-2.5.6-with-dependencies\spring-framework-2.5.6\lib\cglib\cglib-nodep-2.1\_3.jar

@Pointcut("execution(public \* com.baishui.service..\*.addUser(..))")

public void myMethod(){

}

//使用定义切入的方法

@Before("myMethod()")

public void before(){

System.out.println("method start...");

}

@AfterReturning("myMethod()")

public void afterReturning(){

System.out.println("method after...");

}

//声明抛出异常时执行的方法

//注意在UserDaoForOracleImpl saveUser 中抛出异常

@AfterThrowing("myMethod()")

public void afterThrowing(){

System.out.println("method afterThrowing...");

}

//执行方法的前后，都执行的方法

@Around("myMethod()")

public void around(ProceedingJoinPoint pjp) throws Throwable{

System.out.println("method around...");

System.out.println("method start...");

pjp.proceed();

System.out.println("method end...");

}

（注：如果AOP用在实现接口的类中，通过Proxy 和InvocationHandler，就可以实现；如果AOP 动态代理用在没实现接口的类上时，需要cglib 直接操作二进制码，来实现aop）

Spring AOP 应用

概念：

JoinPoint

PointCut

Aspect（切面）

Advice

Target

Weave

Spring AOP配置与应用

两种方式：

使用Annotation

使用xml

Annotation

加上对应的xsd文件spring-aop.xsd

beans.xml <aop:aspectj-autoproxy />

此时就可以解析对应的Annotation了

建立我们的拦截类

用@Aspect注解这个类

建立处理方法

用@Before来注解方法

写明白切入点（execution …….）

让spring对我们的拦截器类进行管理@Component

常见的Annotation:

@Pointcut

@Before

@AfterReturning

@AfterThrowing

@After

@Around

织入点语法

void !void

参考文档（\* ..）

xml配置AOP

把interceptor对象初始化

<aop:config

<aop:aspect …..

<aop:pointcut

<aop:before

知识点16

1 知识点：AOP

AOP的xml 方式实现 （常用的）

2 代码工程ES1600：

Spring\_1600\_AOP\_XML

3 详解：

Xml 实现AOP 的配置写法

<bean id="logInterceptor" class="com.baishui.aop.LogInterceptor"></bean>

<aop:config>

//配置一个切面

<aop:pointcut id="servicePointcut" expression="execution(public \* com.baishui.service..\*.addUser(..))"/>

//配置切面的引用

<aop:aspect id="logAspect" ref="logInterceptor" >

<!--局部的pointcut

<aop:pointcut id="servicePointcut" expression="execution(public \* com.baishui.service..\*.addUser(..))"/>

-->

//配置切入点方法

<aop:before method="before" pointcut-ref="servicePointcut"/>

<!-- 不引用pointcut ref

<aop:before method="before" pointcut="execution(public \* com.baishui.service..\*.addUser(..))"/>

-->

<aop:after method="afterReturning" pointcut-ref="servicePointcut"/>

</aop:aspect>

</aop:config>

知识点17

1 知识点：DataSource

Spring DataSource 数据源

2 代码工程ES1700：

Spring\_1700\_Spring\_DataSource

3 详解：

1 添加需要的 jar文件

spring-framework-2.5.6-with-dependencies\spring-framework-2.5.6\lib\jakarta-commons\commons-dbcp.jar

spring-framework-2.5.6-with-dependencies\spring-framework-2.5.6\lib\jakarta-commons\commons-pool.jar

数据库连接驱动： mysql-connector-java-5.1.6-bin.jar

2 在xml中配置 DataSource 类的引用

<bean id="datasource" class="org.apache.commons.dbcp.BasicDataSource"

destroy-method="close">

<!-- results in a setDriverClassName(String) call -->

<property name="driverClassName"> <value>com.mysql.jdbc.Driver</value> </property>

<property name="url">

<value>jdbc:mysql://localhost:3306/bbs</value>

</property>

<property name="username"> <value>root</value> </property>

<property name="password"> <value>root</value> </property>

</bean>

create table \_category(id int primary key auto\_increment, name varchar(50), descrtiption varchar(200));

3 通过IOC 将数据库连接源注入到 UserDaoForMySql 类中

@Resource(name="datasource")

（注：datasource 的其他的参数设置，用到时可去网上查）

maxActive ： 连接池最大连接数

maxIdle : 最大等待缓冲数

maxWait : 当前连接无操作，超时时间

性能的调优，必须根据具体测试来确定性能调优设置

知识点18

1 知识点：

Spring 初步整合 hibernate

2 代码工程ES1800：

Spring\_1800\_Spring\_Hibernate

3 详解：

1.添加 hibernate用到的 jar lib （参考hibernate自学笔记中总结）

2.在Spring beans.xml 中 加入 hibernate IOC (详解看一下代码注释)

<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

<property name="locations">

<value>classpath:jdbc.properties</value>

</property>

</bean>

<bean id="datasource"destroy-method="close"class="org.apache.commons.dbcp.BasicDataSource">

<property name="driverClassName" value="${jdbc.driverClassName}"/>

<property name="url" value="${jdbc.url}"/>

<property name="username" value="${jdbc.username}"/>

<property name="password" value="${jdbc.password}"/>

</bean>

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">

<!-- hibernate annotation 配置方法 -->

<property name="annotatedClasses"> <!—配置实体类 可以设置成 自动扫面，后面注释->

<list>

<value>com.baishui.model.User</value>

</list>

</property>

<property name="hibernateProperties">

<props>

<prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>

<prop key="hibernate.show\_sql">true</prop>

<prop key="hibernate.format\_sql">true</prop>

<prop key="hibernate.hbm2ddl.auto">update</prop>

</props>

</property>

3 在 Dao类里加入 hibernate sessionFactory 的注入

@Component(value="userDaoForMySql")

public class UserDaoForMySqlImpl implements UserDao{

private SessionFactory sessionFactory;

public SessionFactory getSessionFactory() {

return sessionFactory;

}

@Resource(name="sessionFactory")

public void setSessionFactory(SessionFactory sessionFactory) {

this.sessionFactory = sessionFactory;

}

public void saveUser(User u) {

System.out.println("UserDaoForMySqlImpl saveUser ...");

Session session = sessionFactory.openSession(); //没有完全整合，事务管理，先用 openSession

session.beginTransaction();

session.save(u);

session.getTransaction().commit();

}

}

（注：设置自动扫描实体类

<!-- hibernate annotation 实体类 自动扫描配置方法 -->

<property name="packagesToScan">

<list><value>com.baishui.model</value></list>

</property>

</bean>

4 代码工程ES1810：

Spring\_1810\_Spring\_Hibernate\_xml

Spring 与 hibernate 的 xml 整合 配置练习

（注： hibernate 实体类 的 hbm.xml 在 Spring beans.xml 的引入的路径 详解；

A\_ 放在根目录，此时在beans.xml中直接写文件的名称， 例如：User.hbm.xml

B\_ 如实体类放在同一目录，此时beans.xml 中必须填入完整路径

例如：com/baishui/model/User.hbm.xml )

知识点19

1 知识点：

Spring 中 hibernate transaction (事物的管理) Annotation 实现

2 代码工程ES1900：

Spring\_1900\_Spring\_Hibernate\_Annotation\_transaction

3 详解：Spring 自己已经通过AOP 的形式 完成此功能

1．Beans.xml加入Spring hibernate TransactionManager 的 xsd

xmlns:tx="http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx-2.5.xsd

2.加入HibernateTransactionManager 管理配置

<!--添加HibernateTransactionManager spring 进行管理-->

<bean id="txManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory" />

</bean>

<!--添加Annotation事务管理器声明-->

<tx:annotation-driven transaction-manager="txManager"/>

3.在service层业务逻辑方法上添加事务管理Annotation 标注 @Transacational

（注：当添加事务管理标注之后，当抛出RunTimeException时spring 会自动回滚事务

@Transactional(propagation=Propagation.REQUIRED) //REQUIRED 含义：如果当前有Transaction对象就用当前的对象，如果没有创建一个新的Transaction

(注： @Transaction propagation属性值查文档，不重要，只要记住REQUIRED

readOnly 属性 设置Connection 为只读)

4.在dao类里 用sessionFactory的 getCurrentSession() 方法，事物交给spring自动处理

Session session = sessionFactory.getCurrentSession();

session.save(msg);

throw new RuntimeException("error");//模拟一个运行时异常

（注：如果不用Spring 的 hibernateTransactionManager 时，写法必须是 在dao类里抛出一个 HibernateException 之后在service中捕获这个异常,之后添加处理异常的事物处理逻辑代码，比较费事）

知识点20

1 知识点：

Spring 中 hibernate transaction (事物的管理) Xml 实现 常用

2 代码工程ES2000：

Spring\_2000\_Spring\_Hibernate\_Xml\_transaction

3 详解：

<!--声明 HibernateTransactionManager-->

<bean id="txManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory" />

</bean>

<!-- 声明Transaction 建议 -->

<tx:advice id="txAdvice" transaction-manager="txManager">

<tx:attributes>

<!-- all methods starting with 'get' are read-only -->

<tx:method name="getUser" read-only="true"/>

<!-- other methods use the default transaction settings (see below) -->

<tx:method name="add\*" propagation="REQUIRED"/>

</tx:attributes>

</tx:advice>

<!-- 声明事务处理AOP 切面 -->

<aop:config>

<aop:pointcut id="fooServiceOperation" expression="execution(\* com.baishui.service.UserService.\*(..))"/>

<aop:advisor advice-ref="txAdvice" pointcut-ref="fooServiceOperation"/>

</aop:config>

（小技巧： 关于AOP面试 总结

问题一：说说AOP 答：通过举例说明和画图 说明AOP，在讲解具体实现

（注：如果AOP用在实现接口的类中，通过Proxy 和InvocationHandler，就可以实现；如果AOP 动态代理用在没实现接口的类上时，需要cglib 直接操作二进制码，来实现aop）

问题二：AOP都有那些实际应用到？ 答：例如：日志系统，声明式事物处理

问题三：你在实际开发中用到AOP？ 答：主要是声明式事物处理

知识点21

1 知识点：

Spring 使用 hibernate 的封装，HibernateTemplate

设计模式：template method (模版方法)

2 代码工程ES2100：

Spring\_2100\_Spring\_HibernateTemplate

3 详解：

1.在beans.xml 中添加对 HibernateTemplate 的管理

<bean id="hibernateTemplate" class="org.springframework.orm.hibernate3.HibernateTemplate">

<property name="sessionFactory" ref="sessionFactory"/>

</bean>

2.在Dao类里面添加 对HibernateTemplate 的注入（此时就不用SessionFactory,因为HibernateTemplate已经对SessionFactory 和 Session 进行了封装）

private HibernateTemplate hibernateTemplate;

public HibernateTemplate getHibernateTemplate() {

return hibernateTemplate;

}

@Resource

/\*

\*注入HibernateTemplate，对Session的方法进行封装

\*/

public void setHibernateTemplate(HibernateTemplate hibernateTemplate) {

this.hibernateTemplate = hibernateTemplate;

}

3.直接使用HibernateTemplate的数据库操作方法

public void saveUser(User u) {

hibernateTemplate.save(u);

}

（注：实现原理，参见代码工程中，callback 钩子函数回调设计 模拟）

知识点22

1 知识点：

另一种实现，数据库操作的方法，继承HibernateDaoSupport

比起向Dao类中注入HibernateTemplat的优点是：不用每个dao类都的添加注入配置

2 代码工程ES2200：

Spring\_2200\_Spring\_HibernateDaoSupport（简单实现）

Spring\_2210\_Spring\_HibernateDaoSuppory(标准写法1)(推荐)

Spring\_2220\_Spring\_HibernateDaoSuppory(标准写法2)

3 详解：

A:简单实现：（缺点：与上文大致思路一样通过注入HibernateTemplate，配置复杂）

1 在dao类中 继承HibernateDaoSupport (此类中Spring已经封装了HibernateTemplate，所以直接向这个类中注入 HibernateTemplate 或 session 就可以实现数据库操作)

2 由于必须注入HibernateTempLate 或 session，但HibernateDaoSupport中set属性的方法是finall, 不能用Annotation,所以只能用XML的形式，实现注入, 还有种方法是直接写一个顶级的Dao类（具体实现参见后文标准写法的配置）

3 在beans.xml中配置DaoImpl (练习中以UserDaoForMySQLImpl为例)

<bean id="userDaoForMySql" class="com.baishui.dao.impl.UserDaoForMySqlImpl">

<property name="hibernateTemplate" ref="hibernateTemplate"/>

</bean>

B:标准写法：

1.自己定义一个 顶级的dao类 （SupperDao）,然后自定义一个 HibernateTemplate属性，进行注入，其他的DaoImpl 继承这个类，就可以实现，只注入一次HibernateTemplate配置

（ 注：因为如果这个类继承HibernateDaoSupport继承时，注入HibernateTemplate时必须用xml的形式，此时就会与DaoImpl 中用到的Annotation冲突，所以只用Annotation的形式实现注入，但是HibernateDaoSupport中的方法都是finall,所以只能自定一个方式来实现数据库操作，并且避免xml配置复杂的情况）

源代码：

@Component

public class SupportDao{

private HibernateTemplate hibernateTemplate;

public HibernateTemplate getHibernateTemplate() {

return hibernateTemplate;

}

@Resource

public void setHibernateTemplate(HibernateTemplate hibernateTemplate) {

this.hibernateTemplate = hibernateTemplate;

}

}

第二种标准写法： 目的为了比标准，继承HibernateDaoSupport，定一个SetSuperHibernateTemplate()方法，参数 为hibernateTemplate,将 Spring 初始的 hibernateTemplate对象注入给这个方法参数；方法中 调用HibernateDaoSupport的setHibernateTemplate()方法

@Component

public class SupportDao extends HibernateDaoSupport{

//通过继承，并且实现注入一次其他子类 Dao不用重复配置注入

/\*第一种写法，将beans.xml中的 hibernateTemplate 注入给参数

\* @Resource(name="hibernateTemplate")

public void setSuperHibernateTemplate(HibernateTemplate hibernateTemplate) {

super.setHibernateTemplate(hibernateTemplate);

}\*/

//第二中写法（推荐），直接将sessionfactory注入给参数，此时就不用在beans.xml 配置事物管理器 HibernateTemplate

@Resource(name="sessionFactory")

public void setSupperSessionFactory(SessionFactory sessionFactory) {

super.setSessionFactory(sessionFactory);

}

}

知识点23

1 知识点：

架构的历史

2 详解：

1 架构历史一： client ---🡪jsp---🡪 DB

2 架构历史二： client ---🡪jsp---🡪Model---🡪DB

3 架构历史三： client ---🡪jsp---🡪 Service---🡪Model---🡪 DB

4 架构历史四： client ---🡪jsp---🡪Contral---🡪 Service---🡪Model---🡪 DB

代码工程ES2500： Spring\_2500\_Registartion\_hibernate

5 构架历史五： client ---🡪jsp---🡪Contral---🡪 Service---🡪Dao（hibernate）---🡪Model---🡪 DB

代码工程ES2600：Spring\_2600\_Registartion\_hibernate\_Dao

6 构架历史六： client ---🡪jsp---🡪Contral(struts2)---🡪 Service---🡪Dao（hibernate）---🡪Model---🡪 DB

代码工程ES2700：Spring\_2700\_Registartion\_Struts2\_hibernate

7 构架历史七： spring 初步整合 struts 2 和 hibernate

代码工程ES2800：Spring\_2800\_Registartion\_SSH

8 构架历史8： spring 与 struts2 的关联管理

代码工程ES2900：Spring\_2900\_Registartion\_Spring&Struts2

(注：参考Struts2 或 Spring 帮助文档)

I 配置web.xml 监听器，web应用程序启动时初始化所有的Spring管理的bean

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:beans.xml</param-value>

</context-param>

<listener>

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

</listener>

II 引入struts2-spring-plugin-2.1.6.jar

III 配置struts2 action； 在action中加入spring管理

在Action 类中 配置annotation 将struts action 加入到spring 管理起来

@Component(value="userAction")

@Scope(value="prototype") //注 action 不应是单例

将UserManager注入到 UserAction 中

@Resource(name="userManager")

9 构架历史9： SSH 整合 嵌入 Dto

代码工程ES3000：Spring\_3000\_Registartion\_SSH\_dto

I 在Action 中运用 dto 接受参数，进行数据的传递

注： action 必须实现 ModelDriven 接口 ，重写getModel() 方法，返回值类型 为 dto类型

II 以 User 和 UserDto 为例，页面表单中，为了不使用 对象.属性 的形式来传递表单参数，所以实现ModelDriven接口，但在Action中必须 添加属性 UserDto , 并为属性添加get set 方法

这样就实现了 dto 对象来 进行 数据的传输的功能

代码工程ES3010：Spring\_3010\_Registartion\_SSH\_xml

（注：ES3010 为 xml 以SSH 整合 的完整配置 ， 包括 中文问题， validate 细节问题）

知识点24

1 知识点：

Spring-Struts2-plubin Struts2 对Action的生成的管理 及注入

2代码工程ES3020：

Spring\_3200\_Registartion\_Spring-Struts2-pulgin

3 详解：

I：Spring-struts-plugin 中 可以自动对Action进行管理，并可以不用在Action 填入任何的Spring容器管理的标注，就可以自动实现

II： spring-struts-plugin 的autowire 默认是根据名称自动注入，此时Action标准的写法就是什么都不用写，但注意属性名称的命名规范；

III： 如果就是想用@Resource注入 时 这时做法是先把属性的 set方法去掉，在属性声明上加入@Resource标注注入）

IV：将plubin的自动注入属性 autowire 设置成 根据 type 注入

在 struts.xml 中配置：

将spring-struts-plubin autowire设置成根据type对Action 进行注入

<constant name="struts.objectFactory.spring.autoWire" value="type"/>

知识点25

1 知识点：

Spring-Struts2-hibernate 中的数据库操作Session的问题

如果用load 或 多个实体对象之间的关联为lazy时 session 已经关闭引发的异常

2 详解：

代码工程ES3030：Spring\_3300\_Spring-Struts2-Hibernate\_OpenSessionInView

3 解决方法：

I：可以直接用 get 方法

II：整个应用扩大hibernate session 的范围，添加 OpenSessionInViewFilter,

在web.xml中配置OpenSessionFilter

<!-- 配置在页面中管理hibernate sesion 的过滤器 -->

<filter>

<filter-name>OpenSessionInView</filter-name> <filter-class>org.springframework.orm.hibernate3.support.OpenSessionInViewFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>OpenSessionInView</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

（注：因过滤器又先后顺序，添加在struts过滤器的上面）

过滤器图解：

OpenSessionInView Struts2Filter(处理jsp)

Client ---request----🡪|--------------------🡪|---------------------🡪service

response<------|----------------------|<-----------------------|

知识点26

1 知识点：

SSH整合之后常见的问题

2 详解：

I : SSH整合之——OpenSessionInView 常见的三个问题

1.注意 OpenSessionInViewFilter 过滤器配置的位置，在Struts2的filter的前面

2.Spring bean 的配置时 SessionFactory 的名字不为 sessionFactory时，OpenSessionInViewFilter默认会找不到 sessionFactory，此时需要配置过滤器初始化参数

<!--指定sessionfactoryname param

<init-param>

<param-name>sessionFactoryBeanName</param-name>

<param-value>sessionFactory</param-value>

</init-param>

-->

3在使用OpenSessionInViewFilter 时 如果没有配置事物边界时，此时由于OpenSessionInViewFilter默认对session的事物操作指定为只读的（readonly）,这是就会有错误出现，所以必须配置事物处理切面，将不同方法制定好transaction事务处理的规则

II : SSH整合之——中文问题

1.解决方法一：直接用struts2的struts2\_i18n\_encoding 设置编码

（注：Struts2.1.6中 这个是个Bug，在struts2.1.8之后修正）

2.解决方法二：在web.xml中自定以一个encoding 过滤器

3.解决方法三：在web.xml添加Spring 中已经封装好的过滤器

III : mysql 数据库的默认编码格式设置

（注：mysql 数据库的默认编码设置，Mysql要解决中文乱码，最好是你把数据库的默认编码都改掉，一般这样处理：  
在mysql安装目录下的my.ini文件中：  
下面修改  
第57行改为：default-character-set=gbk  
下面修改  
第81行改为：default-character-set=gbk

在MySQL命令行输入:set names gbk;

再在MySQL命令行输入：status;命令后，如果出现：Server Db Client Conn四个全部为GBK，则支持中文没问题

知识点27

1 知识点：

SSH整合拓展训练

I ：Spring testcontext : 基于transaction的保护数据现场的测试 框架

II ：OpenSessionInViewInterceptor: 在页面控制Hibernate Session的打开和关闭的另一种方法

知识点28

1 知识点：

Spring 与 Struts 谁来创建Action的问题补充: 用Spring 来管理

优点：Spring testcontext 测试时可以直接生成Action测试

2 详解

I如知识点24中所说的 用struts来管理Action的 创建 此时的Action 中的注入都是plugin自动完成

II 如过想要交给Spring来管理,此时的配置如下：

1.在Action加入标注 @Component（“userAction”）,@Scope(value=”prototype”)

2.在setUserManager方法上添加 @Resource,此时必须用Spring来管理注入userManager

3.在struts.xml中 配置action时 ，class=”填@Component标注的名称”

3代码工程ES3210： Spring\_3210\_Registartion\_Spring-Struts2-action2spring

知识点29

1 知识点：

Spring3.0新特性------------基础（环境搭建）

2 代码工程ES001：

Spring3.0\_001

3 解决方法：

1.所有spring3.0 的jar包解压spring-framework-3.0.0.RELEASE-with-docs.zip，dist文件夹下

（注：还需要commons-loggin.jar）

2．使用org.springframework.test.context.junit4.AbstractJUnit4SpringContextTests 进行测试

可以通过直接注入来测试方便快捷

package com.baishui.dao.impl;

import javax.annotation.Resource;

import org.junit.Test;

import org.springframework.test.context.ContextConfiguration;

import com.baishui.dao.UserDao;

@ContextConfiguration("classpath:beans.xml")

public class testDao extends org.springframework.test.context.junit4.AbstractJUnit4SpringContextTests{

@Resource(name="userDao")

private UserDao userDao;

@Test

public void testAddUser(){

this.userDao.addUser();

}

}

知识点30

1 知识点：

Spring3.0新特性------1------getBean() 添加了泛型 省略了强转

2 代码工程ES002：

Spring3.0\_002

3 解决方法：

@Test

public void testAddUser(){

ApplicationContext app = new ClassPathXmlApplicationContext("beans.xml");

UserDao userDao=app.getBean("userDao",UserDao.class); //新特性1:添加了泛型

userDao.addUser();

}

知识点31

1 知识点：

Spring3.0新特性------2------对bean的管理添加了第三种方法JavaConfig (xml annotation)

简单说就是用Java类替代xml配置文件

2 代码工程ES003：

Spring3.0\_003

3 解决方法：

1 需要的jar包 ：

Cglib.jar 和 asm.jar

1 beans.xml 中配置

<context:component-scan base-package="com.baishui.config"></context:component-scan>

2 创建AppConfig 管理类

package com.baishui.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import com.baishui.dao.UserDao;

import com.baishui.dao.impl.UserDaoForMySqlImpl;

@Configuration //此注解是 CGlib 中的

public class AppConfig {

@Bean

public UserDao userDao(){

return new UserDaoForMySqlImpl();

}

/\*xml

\* <bean id="userDao" class="com.baishui.dao.impl.UserDaoForMySqlImpl"></bean>

\*/

}

知识点32

1 知识点：

Spring3.0新特性------3------对bean的管理添加了第三种方法JavaConfig 的属性注入问题

2 代码工程ES004：

Spring3.0\_004

3 解决方法：

package com.baishui.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import com.baishui.dao.UserDao;

import com.baishui.dao.impl.UserDaoForMySqlImpl;

import com.baishui.service.UserService;

import com.baishui.service.impl.UserServiceImpl;

@Configuration //此注解是 CGlib 中的

public class AppConfig {

@Bean

public UserDao userDao(){

return new UserDaoForMySqlImpl();

}

@Bean

public UserService userService(){

UserServiceImpl userService = new UserServiceImpl();

userService.setUserDao(userDao());//设置属性的注入

return userService;

}

}

知识点33

1 知识点：

Spring3.0新特性------4-----Spring支持一种新的EL表达式语言：SpEl （将字符串动态执行）

2 代码工程ES005：

Spring3.0\_005

3 解决方法：

@Test

public void testSpEL(){

ExpressionParser parser = new SpelExpressionParser();

Expression exp = parser.parseExpression("new String('hello').toUpperCase()");

System.out.println((String)exp.getValue());

}

@Test

public void testSpEL1(){

User user = new User();

user.setUsername("zhangsan");

EvaluationContext context = new StandardEvaluationContext(user);//根对象

ExpressionParser parser = new SpelExpressionParser();

Expression exp = parser.parseExpression("username"); //定义一个新的表达式语言,相当于调用了user.getUsername()

System.out.println((String)exp.getValue(context));

System.out.println((String)exp.getValue(user));

}

知识点34

1 知识点：

Spring3.0新特性------4-----复杂的SpEl表达式 与配置文件关联

2 代码工程ES006：

Spring3.0\_006

3 解决方法：

1 在beans.xml中配置

<bean id="my" class="com.baishui.spel.MyBean">

<property name="randomNumber" value="#{T(java.lang.Math).random()\*100}"></property>

<property name="userRegion" value="#{systemProperties['user.country']}"></property>

</bean>

2 在测试类中测试 生成的动态代码

@ContextConfiguration("classpath:beans.xml")

public class testMyBean extends AbstractJUnit4SpringContextTests {

@Resource

private MyBean my;

@Test

public void testSpEL(){

System.out.println(my.getRandomNumber());

System.out.println(my.getUserRegion());

}

}

2010-11-1 下午 17：11 完成了！！！

Come on man!!