Spring\_day03总结

今日内容

* Spring的事务管理
* 三大框架整合

## 上次课的内容回顾:

Spring的AOP开发:

\* AOP:面向切面编程,是对OO思想延伸.

\* AOP底层实现原理:动态代理.

\* JDK动态代理:针对实现了接口的类生产代理.

\* CGLIB代理:针对没有实现接口的类，产生一个子类.

\* AOP术语:

\* JoinPoint:可以被拦截点.

\* Ponitcut:真正被拦截.

\* Advice:通知，增强的代码.

\* 引介:特殊通知，类级别上添加属性或方法.

\* Target:目标对象.

\* Proxy:代理对象.

\* Weaving:

\* Aspect:

\* Spring的AOP的开发:

\* 配置applicationContext.xml生成代理对象.

\* 使用ProxyFactoryBean类生产代理:

\* 根据目标对象是否实现了接口，选择使用JDK还是CGLIB.

\* 缺点:需要为每个类都去配置一个ProxyFactoryBean.

\* 采用Spring自动代理:

\* 基于类名称的自动代理:(采用后处理Bean)

\* 基于切面信息的自动代理:(采用后处理Bean)

\* Spring的AspectJ的切面开发.

\* AspectJ:本身第三方切面框架.

\* AspectJ基于注解开发:

\* 定义切面:

@Aspect

\* 定义增强:

@Before:前置通知.

@AfterReturing:后置通知.

@Around:环绕通知.

@AfterThrowing:异常抛出通知.

@After:最终通知.

\* 定义切点:

@Pointcut

\* AspectJ基于XML开发:

\* 引入aop名称空间.

<aop:config>

<aop:pointcut expression=”” id=””/>

<aop:aspect ref=””>

<aop:before...>

</aop:aspect>

</aop:config>

Spring的JDBCTemplate:

\* 配置连接池:

\* 默认

\* DBCP

\* C3P0(\*\*\*\*\*)

\* 引入外部属性文件.

\* 在DAO中注入JdbcTemplate.

\* 在DAO中不直接注入模板.Dao集成JdbcDaoSupport.

\* CRUD的操作.

## Spring的事务管理:

### 事务:

事务:是逻辑上一组操作，要么全都成功，要么全都失败.

事务特性:

ACID:

原子性:事务不可分割

一致性:事务执行的前后，数据完整性保持一致.

隔离性:一个事务执行的时候，不应该受到其他事务的打扰

持久性:一旦结束，数据就永久的保存到数据库.

如果不考虑隔离性:

脏读 :一个事务读到另一个事务未提交数据

不可重复读 :一个事务读到另一个事务已经提交数据（update）导致一个事务多次查询结果不一致

虚读 :一个事务读到另一个事务已经提交数据（insert）导致一个事务多次查询结果不一致

事务的隔离级别:

未提交读 :以上情况都有可能发生。

已提交读 :避免脏读，但不可重复读，虚读是有可能发生。

可重复读 :避免脏读，不可重复读，但是虚读有可能发生。

串行的 :避免以上所有情况.

### Spring中事务管理:

分层开发：事务处在Service层.

#### Spring提供事务管理API:

PlatformTransactionManager:平台事务管理器.

[commit](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/PlatformTransactionManager.html" \l "commit(org.springframework.transaction.TransactionStatus))([TransactionStatus](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/TransactionStatus.html" \o "interface in org.springframework.transaction) status)

[getTransaction](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/PlatformTransactionManager.html" \l "getTransaction(org.springframework.transaction.TransactionDefinition))([TransactionDefinition](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/TransactionDefinition.html" \o "interface in org.springframework.transaction) definition)

[rollback](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/PlatformTransactionManager.html" \l "rollback(org.springframework.transaction.TransactionStatus))([TransactionStatus](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/TransactionStatus.html" \o "interface in org.springframework.transaction) status)

TransactionDefinition:事务定义

ISOLation\_XXX :事务隔离级别.

PROPAGATION\_XXX :事务的传播行为.(不是JDBC中有的，为了解决实际开发问题.)

过期时间:

TransactionStatus:事务状态

是否有保存点

是否一个新的事务

事务是否已经提交

关系:PlatformTransactionManager通过TransactionDefinition设置事务相关信息管理事务，管理事务过程中，产生一些事务状态:状态由TransactionStatus记录.

API详解:

PlatformTransactionManager:接口.

Spring为不同的持久化框架提供了不同PlatformTransactionManager接口实现

**org.springframework.jdbc.datasource.DataSourceTransactionManager : 使用Spring JDBC或iBatis 进行持久化数据时使用**

**org.springframework.orm.hibernate3.HibernateTransactionManager : 使用Hibernate3.0版本进行持久化数据时使用**

org.springframework.orm.jpa.JpaTransactionManager 使用JPA进行持久化时使用

org.springframework.jdo.JdoTransactionManager 当持久化机制是Jdo时使用

org.springframework.transaction.jta.JtaTransactionManager 使用一个JTA实现来管理事务，在一个事务跨越多个资源时必须使用

TransactionDefinition:

\* [ISOLATION\_DEFAULT](mk:@MSITStore:D:\\itcast\\20141110\\spring\\spring_day01\\Spring3.0.2-RELEASE-API.chm::/org/springframework/transaction/../../../org/springframework/transaction/TransactionDefinition.html" \l "ISOLATION_DEFAULT):默认级别. Mysql repeatable\_read oracle read\_commited

ISOLATION\_READ\_UNCOMMITTED

ISOLATION\_READ\_COMMITTED

ISOLATION\_REPEATABLE\_READ

ISOLATION\_SERIALIZABLE

\* 事务的传播行为:(不是JDBC事务管理，用来解决实际开发的问题.) 传播行为：解决业务层之间的调用的事务的关系.

**PROPAGATION\_REQUIRED**  :支持当前事务，如果不存在 就新建一个

\* A,B 如果A有事务，B使用A的事务，如果A没有事务，B就开启一个新的事务.(A,B是在一个事务中。)

PROPAGATION\_SUPPORTS :支持当前事务，如果不存在，就不使用事务

\* A,B 如果A有事务，B使用A的事务，如果A没有事务，B就不使用事务.

PROPAGATION\_MANDATORY :支持当前事务，如果不存在，抛出异常

\* A,B 如果A有事务，B使用A的事务，如果A没有事务，抛出异常.

**PROPAGATION\_REQUIRES\_NEW** 如果有事务存在，挂起当前事务，创建一个新的事务

\* A,B 如果A有事务，B将A的事务挂起，重新创建一个新的事务.(A,B不在一个事务中.事务互不影响.)

PROPAGATION\_NOT\_SUPPORTED 以非事务方式运行，如果有事务存在，挂起当前事务

\* A,B 非事务的方式运行，A有事务，就会挂起当前的事务.

PROPAGATION\_NEVER 以非事务方式运行，如果有事务存在，抛出异常

**PROPAGATION\_NESTED** 如果当前事务存在，则嵌套事务执行

\* 基于SavePoint技术.

\* A,B A有事务，A执行之后，将A事务执行之后的内容保存到SavePoint.B事务有异常的话，用户需要自己设置事务提交还是回滚.

\* 常用:(重点)

PROPAGATION\_REQUIRED

PROPAGATION\_REQUIRES\_NEW

PROPAGATION\_NESTED

### Spring的事务管理:

Spring的事务管理分成两类:

\* 编程式事务管理:

\* 手动编写代码完成事务管理.

\* 声明式事务管理:

\* 不需要手动编写代码,配置.

### 事务操作的环境搭建:

CREATE TABLE `account` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(20) NOT NULL,

`money` double DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=4 DEFAULT CHARSET=utf8;

INSERT INTO `account` VALUES ('1', 'aaa', '1000');

INSERT INTO `account` VALUES ('2', 'bbb', '1000');

INSERT INTO `account` VALUES ('3', 'ccc', '1000');

创建一个web项目:

\* 导入相应jar包

\* 引入配置文件:

\* applicationContext.xml、log4j.properties、jdbc.properties

创建类:

\* AccountService

\* AccountDao

在Spring中注册:

<!-- 业务层类 -->

<bean id="accountService" class="cn.itcast.spring3.demo1.AccountServiceImpl">

<!-- 在业务层注入Dao -->

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

</bean>

<!-- 持久层类 -->

<bean id="accountDao" class="cn.itcast.spring3.demo1.AccountDaoImpl">

<!-- 注入连接池的对象,通过连接池对象创建模板. -->

**<property name="dataSource" ref="dataSource"/>**

</bean>

编写一个测试类:

### Spring的事务管理:

#### 手动编码的方式完成事务管理:

需要事务管理器:**真正管理事务对象.**

**\* Spring提供了事务管理的模板(工具类.)**

**第一步:注册事务管理器:**

<!-- 配置事务管理器 -->

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<!-- 需要注入连接池,通过连接池获得连接 -->

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

</bean>

第二步:注册事务模板类:

<!-- 事务管理的模板 -->

<bean id="transactionTemplate" class="org.springframework.transaction.support.TransactionTemplate">

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

</bean>

第三步:在业务层注入模板类:(模板类管理事务)

<!-- 业务层类 -->

<bean id="accountService" class="cn.itcast.spring3.demo1.AccountServiceImpl">

<!-- 在业务层注入Dao -->

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

<!-- 在业务层注入事务的管理模板 -->

**<property name="transactionTemplate" ref="transactionTemplate"/>**

</bean>

第四步:在业务层代码上使用模板:

public void transfer(final String from, final String to, final Double money) {

**transactionTemplate.execute(new TransactionCallbackWithoutResult() {**

**@Override**

**protected void doInTransactionWithoutResult(TransactionStatus status) {**

**accountDao.out(from, money);**

**int d = 1 / 0;**

**accountDao.in(to, money);**

**}**

**});**

}

手动编码方式缺点:

\* 代码量增加,代码有侵入性.

#### 声明式事务管理:(原始方式)

基于TransactionProxyFactoryBean.

导入:aop相应jar包.

第一步:注册平台事务管理器:

<!-- 事务管理器 -->

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<!-- 注入连接池 -->

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

</bean>

第二步:创建业务层代理对象:

<!-- 配置生成代理对象 -->

<bean id="accountServiceProxy" class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">

<!-- 目标对象 -->

<property name="target" ref="accountService"/>

**<!-- 注入事务管理器 -->**

**<property name="transactionManager" ref="transactionManager"/>**

**<!-- 事务的属性设置 -->**

**<property name="transactionAttributes">**

**<props>**

**<prop key="transfer">PROPAGATION\_REQUIRED</prop>**

**</props>**

**</property>**

</bean>

第三步:编写测试类:

**\*\*\*\*\* 千万注意:注入代理对象**

@Autowired

@Qualifier("**accountServiceProxy**")

private AccountService accountService;

**prop格式：PROPAGATION,ISOLATION,readOnly,-Exception,+Exception**

**\* 顺序:传播行为、隔离级别、事务是否只读、发生哪些异常可以回滚事务（所有的异常都回滚）、发生了哪些异常不回滚.**

**\*\*\*\*\* 缺点:就是需要为每一个管理事务的类生成代理.需要为每个类都需要进行配置.**

#### 声明式事务管理:(自动代理.基于切面 \*\*\*\*\*\*)

**第一步:导入相应jar包.**

**\* aspectj**

**第二步:引入相应约束:**

**\* aop、tx约束.**

**<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"**

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

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

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

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

**http://www.springframework.org/schema/context/spring-context.xsd**

**http://www.springframework.org/schema/aop**

**http://www.springframework.org/schema/aop/spring-aop.xsd**

**http://www.springframework.org/schema/tx**

**[http://www.springframework.org/schema/tx/spring-tx.xsd">](http://www.springframework.org/schema/tx/spring-tx.xsd%22%3e)**

**第三步:注册事务管理器;**

<!-- 事务管理器 -->

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

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

</bean>

第四步:定义增强(事务管理)

<!-- 定义一个增强 -->

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

<!-- 增强(事务)的属性的配置 -->

<tx:attributes>

<!--

isolation:DEFAULT :事务的隔离级别.

propagation :事务的传播行为.

read-only :false.不是只读

timeout :-1

no-rollback-for :发生哪些异常不回滚

rollback-for :发生哪些异常回滚事务

-->

<tx:method name="transfer"/>

</tx:attributes>

</tx:advice>

**第五步:定义aop的配置(切点和通知的组合)**

<!-- aop配置定义切面和切点的信息 -->

<aop:config>

<!-- 定义切点:哪些类的哪些方法应用增强 -->

<aop:pointcut expression="execution(\* cn.itcast.spring3.demo3.AccountService+.\*(..))" id="mypointcut"/>

<!-- 定义切面: -->

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

</aop:config>

第六步:编写测试类:

\* 注入Service对象,不需要注入代理对象(生成这个类的时候,已经是代理对象.)

#### 基于注解的事务管理:

第一步:事务管理器:

<!-- 事务管理器 -->

<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

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

</bean>

第二步:注解事务:

<!-- 开启注解的事务管理 -->

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

第三步:在Service上使用注解

@Transactional

\* 注解中有属性值:

\* isolation

\* propagation

\* readOnly

...

## SSH框架整合:

### Struts2+Spring+Hibernate导包

Struts2导入jar包:

\* struts2/apps/struts2-blank.war/WEB-INF/lib/\*.jar

\* 导入与spring整合的jar

\* struts2/lib/struts2-spring-plugin-2.3.15.3.jar --- 整合Spring框架

\* struts2/lib/struts2-json-plugin-2.3.15.3.jar --- 整合AJAX

\* struts2/lib/struts2-convention-plugin-2.3.15.3.jar --- 使用Struts2注解开发.

\* 配置

web.xml

<filter>

  <filter-name>struts2</filter-name>

  <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

  <filter-name>struts2</filter-name>

  <url-pattern>/\*</url-pattern>

</filter-mapping>

struts.xml

<struts>

<constant name="struts.devMode" value="true" />

<package name="default" namespace="/" extends="struts-default">

</package>

</struts>

Spring导入jar包:

Spring3.2 开发最基本jar包

spring-beans-3.2.0.RELEASE.jar

spring-context-3.2.0.RELEASE.jar

spring-core-3.2.0.RELEASE.jar

spring-expression-3.2.0.RELEASE.jar

com.springsource.org.apache.commons.logging-1.1.1.jar

com.springsource.org.apache.log4j-1.2.15.jar

AOP开发

spring-aop-3.2.0.RELEASE.jar

spring-aspects-3.2.0.RELEASE.jar

com.springsource.org.aopalliance-1.0.0.jar

com.springsource.org.aspectj.weaver-.6.8.RELEASE.jar

Spring Jdbc开发

spring-jdbc-3.2.0.RELEASE.jar

spring-tx-3.2.0.RELEASE.jar

Spring事务管理

spring-tx-3.2.0.RELEASE.jar

Spring整合其他ORM框架

spring-orm-3.2.0.RELEASE.jar

Spring在web中使用

spring-web-3.2.0.RELEASE.jar

Spring整合Junit测试

spring-test-3.2.0.RELEASE.jar

(Spring没有引入c3p0和数据库驱动)

\* 配置:

applicationContext.xml

Log4j.properties

在web.xml中配置监听器;

<!-- 配置Spring的监听器 -->

<listener>

<!-- 监听器默认加载的是WEB-INF/applicationContext.xml -->

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

</listener>

<!-- 指定Spring框架的配置文件所在的位置 -->

<context-param>

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

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

</context-param>

Hibernate的jar包导入:

\* 核心包:hibernate3.jar

\* lib/required/\*.jar

\* lib/jpa/\*.jar

\* 引入hibernate整合日志系统的jar包:

\* 数据连接池:

\* 数据库驱动:

\* 二级缓存:(可选的.)

\* backport-util-concurrent.jar

\* commons-logging.jar

\* ehcache-1.5.0.jar

\* Hibernate的配置:

\* hibernate.cfg.xml

\* 映射:

\* 格式:类名.hbm.xml

### Struts2和Spring的整合:

1.新建包结构:

\* cn.itcast.action

\* cn.itcast.service

\* cn.itcast.dao

\* cn.itcast.vo

2.创建实体类:

\* Book

3.新建一个jsp页面:

\* addBook.jsp

<s:form action="book\_add" namespace="/" method="post" theme="simple">

图书名称:<s:textfield name="name"/><br/>

图书价格:<s:textfield name="price"/><br/>

<s:submit value="添加图书"/>

</s:form>

1. 编写Action:

public class BookAction extends ActionSupport implements ModelDriven<Book>{

// 模型驱动类

private Book book = new Book();

public Book getModel() {

return book;

}

// 处理请求的方法:

public String add(){

System.out.println("web层的添加执行了...");

return NONE;

}

}

5.配置struts.xml

<action name="book\_\*" class="cn.itcast.action.BookAction" method="{1}">

</action>

### Struts2和Spring的整合两种方式:

#### Struts2自己管理Action:(方式一)

<action name="book\_\*" class="cn.itcast.action.BookAction" method="{1}">

\* Struts2框架自动创建Action的类.

#### Action交给Spring管理:(方式二)

可以在<action>标签上通过一个伪类名方式进行配置:

<action name="book\_\*" class="**bookAction**" method="{1}"></action>

在spring的配置文件中:

<!-- 配置Action -->

<bean id="**bookAction**" class="cn.itcast.action.BookAction" scope="prototype"></bean>

(\*\*\*\*\*)注意:Action交给Spring管理一定要配置scope=”**prototype**”

推荐使用二:

\* 在Spring中管理的类,可以对其进行AOP开发.统一的管理.

#### Web层获得Service:

传统方式:

\* 获得WebApplicationContext对象.

\* 通过WebAppolicationContext中getBean(“”);

实际开发中:

\* 引入了struts2-spring-plugin-2.3.15.3.jar

\* 有一个配置文件 : struts-plugin.xml

开启常量 :

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

引发另一个常量的执行:(Spring的工厂类按照名称自动注入)

struts.objectFactory.spring.autoWire = name

### Spring整合Hibernate:

Spring整合Hibernate框架的时候有两种方式:

#### 零障碍整合:(一)

可以在Spring中引入Hibernate的配置文件.

1.通过LocalSessionFactoryBean在spring中直接引用hibernate配置文件

<!-- 零障碍整合 在spring配置文件中引入hibernate的配置文件 -->

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

<property name="configLocation" value="classpath:hibernate.cfg.xml"/>

</bean>

2.Spring提供了Hibernate的模板.只需要将HibernateTemplate模板注入给DAO.

\* DAO继承HibernateDaoSupport.

<!-- DAO的配置 -->

<bean id="bookDao" class="cn.itcast.dao.BookDao">

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

</bean>

改写DAO:继承HibernateDaoSupport类.

public class BookDao extends HibernateDaoSupport{

public void save(Book book) {

System.out.println("DAO层的保存图书...");

**this.getHibernateTemplate().save(book);**

}

}

1. 创建一个映射文件 :

<hibernate-mapping>

<class name="cn.itcast.vo.Book" table="book">

<id name="id">

<generator class="native"/>

</id>

<property name="name"/>

<property name="price"/>

</class>

</hibernate-mapping>

1. 别忘记事务管理:

事务管理器:

<!-- 管理事务 -->

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

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

</bean>

1. 注解管理事务:

<!-- 注解开启事务 -->

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

1. 在业务层类上添加一个注解:

@Transactional

#### 没有Hibernate配置文件的形式(二)

不需要Hibernate配置文件的方式,将Hibernate配置文件的信息直接配置到Spring中.

Hibernate配置文件中的信息 :

\* 连接数据库基本参数:

\* Hibernate常用属性:

\* 连接池:

\* 映射:

把Hibernate配置文件整合Spring中:

连接池:

<!-- 引入外部属性文件. -->

<context:property-placeholder location="classpath:jdbc.properties"/>

<!-- 配置c3p0连接池 -->

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">

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

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

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

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

</bean>

**Hibernate常用属性:**

<!-- 配置Hibernate的属性 -->

<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>

<prop key="hibernate.connection.autocommit">false</prop>

</props>

</property>

**映射**

<!-- <property name="mappingResources">

<list>

<value>cn/itcast/vo/Book.hbm.xml</value>

</list>

</property> -->

<property name="mappingDirectoryLocations">

<list>

<value>classpath:cn/itcast/vo</value>

</list>

</property>

### HibernateTemplate的API:

Serializable save(Object entity) :保存数据

void update(Object entity) :修改数据

void delete(Object entity) :删除数据

<T> T get(Class<T> entityClass, Serializable id) :根据ID进行检索.立即检索

<T> T load(Class<T> entityClass, Serializable id) :根据ID进行检索.延迟检索.

**List find(String queryString, Object... values) :支持HQL查询.直接返回List集合.**

**List findByCriteria(DetachedCriteria criteria) :离线条件查询.**

**List findByNamedQuery(String queryName, Object... values) :命名查询的方式.**

### OpenSessionInView:

## 基于注解的方式整合SSH:

导入以上工程jar包:

\* 导入struts2的注解开发:

\* struts2-convention-plugin-2.3.15.3.jar

\* web.xml:

<!-- 配置Spring的监听器 -->

<listener>

<!-- 监听器默认加载的是WEB-INF/applicationContext.xml -->

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

</listener>

<!-- 指定Spring框架的配置文件所在的位置 -->

<context-param>

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

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

</context-param>

<!-- 配置Struts2的核心过滤器 -->

<filter>

<filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

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

</filter-mapping>

\* 创建包结构:

\* 引入spring的配置文件、log4j、jdbc属性文件.

\* 创建页面:

\* 创建Action:

@Namespace("/")

@ParentPackage("struts-default")

public class BookAction extends ActionSupport implements ModelDriven<Book>{

private Book book = new Book();

public Book getModel() {

return book;

}

@Action(value="book\_add")

public String add(){

System.out.println("web层添加图书...");

return NONE;

}

}

\* Action---Service----Dao

将各层类使用注解装配Spring中:

@Controller

@Service

@@Repository

完成属性注入:

@Autowired

@Qualifier("bookService")

\* 实体类:

@Entity

@Table(name="book")

public class Book {

@Id

@GeneratedValue(strategy=GenerationType.IDENTITY)

private Integer id;

@Column(name="name")

private String name;

private Double price;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Double getPrice() {

return price;

}

public void setPrice(Double price) {

this.price = price;

}

@Override

public String toString() {

return "Book [id=" + id + ", name=" + name + ", price=" + price + "]";

}

}

\* 事务管理:

\* 模板注入:

今天内容总结:

Spring的事务管理:

\* 编程式事务:(了解)

\* 声明式事务:

\* TransactionProxyFactoryBean.

**\* AOP和事务配置:(\*\*\*\*\*)**

**\* 基于注解事务管理:(\*\*\*\*\*)**

SSH整合:

\* SSH整合(带有hibernate配置文件)

\* 导包:

\* 配置文件:

\* Struts2+Spring

\* 两种方式:

\* Action的类由Struts框架创建.

\* Action的类由Spring框架创建.(scope=”prototype”)

\* Spring+Hibernate:

\* 在Spring框架中引入Hibernate的配置文件.

\* 管理事务:

\* DAO中注入sessionFactory.

\* SSH整合(不带Hibernate配置文件)

\* 导包:

\* 配置文件:

\* Struts2+Spring

\* 两种方式:

\* Action的类由Struts框架创建.

\* Action的类由Spring框架创建.(scope=”prototype”)

\* Spring+Hibernate

\* 把Hibernate配置信息配置到Spring中

\* 管理事务:

\* DAO中注入sessionFactory.

\* SSH注解.(\*\*)

\* Struts2:

\* 在Action的类上

\* @Namespace(“/”)

\* @ParentPackage("struts-default")

\* 在要执行的方法上:

\* @Action

\* 把Action/Service/Dao交给Spring.

\* Action:

@Controller("bookAction")

@Scope("prototype")

\* Service

@Service

\* Dao

@Repository

\* 配置Spring中自动扫描;

<context:component-scan base-package="cn.itcast.action,cn.itcast.service,cn.itcast.dao"/>

\* 映射:

@Entity

@Table(name="book")

public class Book {

@Id

@GeneratedValue(strategy=GenerationType.IDENTITY)

private Integer id;

@Column(name="name")

private String name;

...

}

\* 配置SessionFactory:

<!-- 配置Hibernate的其他属性: -->

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

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

<!-- 配置Hibernate的属性 -->

<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>

<prop key="hibernate.connection.autocommit">false</prop>

</props>

</property>

<!-- 映射扫描 -->

<property name="packagesToScan">

<list>

<value>cn.itcast.vo</value>

</list>

</property>

</bean>

\* 事务管理:

<!-- 事务管理器 -->

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

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

</bean>

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

\* DAO中使用Hibernate模板:

\* 手动注入HibernateTemplate :

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

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

</bean>

\* 在Dao中

@Autowired

@Qualifier("hibernateTemplate")

private HibernateTemplate hibernateTemplate;