Spring第二天：

1. 代理

2. aop 编程

\* XML

\*注解

3 Spring对jdbc的支持

目标：

1. Spring声明式事务管理

2. Spring 与 Hibernate 整合

\* Spring创建SessionFactory几种方式

\* Spring对dao操作的支持

3. SSH 初步整合

# 1. Spring声明式事务管理

## 概念

* 事务是一组操作的执行单元，相对于数据库操作来讲，事务管理的是一组SQL指令，比如增加，修改，删除等，事务的一致性，要求，这个事务内的操作必须全部执行成功，如果在此过程种出现了差错，比如有一条SQL语句没有执行成功，那么这一组操作都将全部回滚
* 事务特性（ACID）
  + Atomic(原子性):要么都成功，要么都失败
  + Consistent(一致性):数据应该不被破坏
  + Isolate(隔离性):用户间操作不相混淆
  + Durable(持久性):永久保存

程序中两种事务管理方式

* **编程式事务管理**
* 编写程序式的事务管理可以清楚的定义事务的边界，可以实现**细粒度的事务控制**，比如你可以通过程序代码来控制你的事务何时开始，何时结束等，与后面介绍的声明式事务管理相比，它可以实现细粒度的事务控制，例如jdbc,hibernate，spring中不提倡使用。

JDBC事务控制：

**con.setAutoCommite (false); 设置事务手动提交**

Hibernate中事务控制：

**session.beginTransaction(); 开启事务**

优缺点：

1. 事务控制精确

2. 事务代码，与业务逻辑处理代码，耦合在一起!

事务代码，不能共用！ 重新写事务控制操作！

开发效率低，不便于维护！ (不想用事务，要改代码！)

* **声明式事务管理 (在Spring中使用)**
* 如果你并不需要细粒度的事务控制，你可以使用声明式事务，在Spring中，你只需要在Spring配置文件中做一些配置，即可将操作纳入到事务管理中，解除了和代码的耦合， 这是对应用代码影响最小的选择，从这一点再次验证了Spring关于**AOP的概念**。当你不需要事务管理的时候，可以直接从Spring配置文件中移除该设置

特点：

1. Spring提供的声明式事务管理，用到Aop概念！

2. 对指定的方法添加事务控制，这里只需要配置即可！

3. 修改事务控制实现或删除事务控制操作，只需要移除xml事务相关配置！

注意：

只能对某个方法应用事务！ （因为“切入点表达式”拦截的是方法，控制不了方法内部代码！）

所以，Spring声明式事务管理，即为**粗粒度的事务控制！**

声明式事务管理器类：

Jdbc:

DataSourceTransactionManager 管理jdbc中事务控制

Hibernate:

HibenateTransactionManager 管理hibernate中事务控制

## 声明式事务管理 – JDBC

|  |
| --- |
|  |

### XML 配置方式实现

事务控制在Service层：

步骤：

1. 引入jar文件

Spring 核心

Spring Aop 切面编程

Spring-jdbc / Spring-tx / 驱动包、连接池

2. dao/service

3. 配置

\* 数据源

\* JdbcTemplate

\* Dao/Service

\* spring声明式事务管理配置

(拦截service方法的执行，动态植入事务控制代码！)

4. 测试

Save();

Int i = 1/0;

Save();

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:p=*"http://www.springframework.org/schema/p"*  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"*>  <!-- 1. 数据源配置 -->  <bean id=*"dataSource"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>  <property name=*"driverClass"* value=*"com.mysql.jdbc.Driver"*></property>  <property name=*"jdbcUrl"* value=*"jdbc:mysql:///hib\_demo"*></property>  <property name=*"user"* value=*"root"*></property>  <property name=*"password"* value=*"root"*></property>  <property name=*"initialPoolSize"* value=*"3"*></property>  <property name=*"maxPoolSize"* value=*"6"*></property>  </bean>  <!-- 2. JdbcTemplate配置 , 注入数据源-->  <bean id=*"jdbcTemplate"* class=*"org.springframework.jdbc.core.JdbcTemplate"*>  <property name=*"dataSource"* ref=*"dataSource"*></property>  </bean>  <!-- 3. dao实例，注入jdbcTemplate -->  <bean id=*"deptDao"* class=*"cn.itcast.a\_tx\_jdbc.DeptDao"*>  <property name=*"jdbcTemplate"* ref=*"jdbcTemplate"*></property>  </bean>    <!-- 4. Service实例，注入dao实例 -->  <bean id=*"deptService"* class=*"cn.itcast.a\_tx\_jdbc.DeptService"*>  <property name=*"deptDao"* ref=*"deptDao"*></property>  </bean>  <!-- 5. Spring声明式事务管理配置 -->    <!-- 5.1 配置事务管理器类 -->  <bean id=*"txManager"* class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <property name=*"dataSource"* ref=*"dataSource"*></property>  </bean>    <!-- 5.2 事务通知配置, 拦截到指定的方法后如何管理事务 -->  <!-- find\* find开头的方法，是只读的事务 -->  <!-- \* 上面所有的方法都不满足时候，采用的事务控制规则 -->  <tx:advice id=*"txAdvice"* transaction-manager=*"txManager"*>  <tx:attributes>  <tx:method name=*"find\*"* read-only=*"true"*/>  <tx:method name=*"get\*"* read-only=*"true"*/>  <tx:method name=*"\*"* read-only=*"false"*/>  </tx:attributes>  </tx:advice>    <!-- 5.3 事务Aop配置 = 切入点表达式 + 应用上面的事务通知 -->  <aop:config>  <aop:pointcut expression=*"execution(\* cn.itcast.a\_tx\_jdbc.\*Service.\*(..))"* id=*"pt"*/>  <aop:advisor advice-ref=*"txAdvice"* pointcut-ref=*"pt"*/>  </aop:config>  </beans> |

### 注解方式实现

步骤：

1. 引入aop相关包

2. 开启

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

3. 使用@Transactional 注解

在需要添加事务控制的方法上写这个注解

@Transactional

写到方法上， 表示当前方法应用事务控制

写到类上， 表示当前类的所有方法都会应用事务

写到父类上， 当执行父类的这个方法时候才应用事务！

### 事务属性

// 当前方法应用事务

@Transactional(

readOnly=**false**, // 读写的事务，当修改数据时候用；如果查询就设置为true

isolation=Isolation.*DEFAULT*, // 事务隔离级别

timeout=-1, // 事务执行的超时时间, -1 表示不超时

noRollbackFor=ArithmeticException.**class**, // 遇到指定的异常不回滚

propagation=Propagation.*REQUIRES\_NEW* // 事务传播行为

)

事务传播行为:

Propagation.

REQUIRES\_NEW 当前执行方法必须在事务环境下运行！

且 当前执行方法始终开启一个新的事务！

REQUIRED 当前执行方法必须在事务环境下运行！

如果调用当前方法时时候已经有一个事务环境，当前执行方法会加入当前事务环境，就不开启新的事务；

如果调用当前方法时候没有事务环境，就开启一个新的事务!

SUPPORTS 支持事务环境！ 如果当前方法没有事务，也可以运行！

Never 当前方法不能在事务环境下运行！

案例：

插入部门信息，同时插入日志(日志)！

1. t\_log 日志表

2. LogService.java 插入记录

insertLog() REQUIRES\_NEW

不管当前执行方法有没有事务环境，都开启新事务！

3. DeptService.java

REQUIRED

Void Save() {

// 日志提示： 再插入部门….

insertLog(); // 始终插入

int I = 1/0;

dao.save(); 调用dao的保存方法

}

# 2. Spring 与 Hibernate 整合

Spring与Hibernate整合，

**\* 单例的SessionFactory对象，交给spring的IOC容器创建！**

**\* 事务管理，交给spring声明式事务管理器**

演示步骤：

1. 没有整合案例

2. 整合

整合步骤：

1. 引入Hibernate/spring框架相关包

\* hibernate jar

\*spring – core

\* spring – aop

\* spring – orm 对orm支持

spring-jdbc-3.2.5.RELEASE.jar

**spring-orm-3.2.5.RELEASE.jar**

spring-tx-3.2.5.RELEASE.jar

2. hibernate.cfg.xml

3. dao/service

## \* Spring创建SessionFactory几种方式

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:p=*"http://www.springframework.org/schema/p"*  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"*>    <!-- 连接池, 通过spring管理 -->  <bean id=*"dataSource"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>  <property name=*"jdbcUrl"* value=*"jdbc:mysql:///hib\_demo"*></property>  <property name=*"driverClass"* value=*"com.mysql.jdbc.Driver"*></property>  <property name=*"user"* value=*"root"*></property>  <property name=*"password"* value=*"root"*></property>  <property name=*"initialPoolSize"* value=*"3"*></property>  <property name=*"maxPoolSize"* value=*"6"*></property>  </bean>    <!-- Spring 与 Hibenate整合 (Spring创建SessionFactory) -->    <!-- 方式1： 直接加载hibernate.cfg.xml的方式，创建sessionFactory对象  <bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">  <property name="configLocation" value="classpath:hibernate.cfg.xml"></property>  </bean>  -->    <!-- 方式2： 连接池交给spring管理，其他配置还是写到hibernate.cfg.xml中  <bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">  <property name="dataSource" ref="dataSource"></property>  <property name="configLocation" value="classpath:hibernate.cfg.xml"></property>  </bean>  -->    <!-- 方式3：(推荐) 所有的配置都在spring中完成-->  <bean id=*"sessionFactory"* class=*"org.springframework.orm.hibernate3.LocalSessionFactoryBean"*>  <!-- a. 注入连接池 -->  <property name=*"dataSource"* ref=*"dataSource"*></property>    <!-- b. hibernate常用配置： 方言、自动建表、显示sql -->  <property name=*"hibernateProperties"*>  <props>  <prop key=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</prop>  <prop key=*"hibernate.show\_sql"*>true</prop>  <prop key=*"hibernate.hbm2ddl.auto"*>update</prop>  </props>  </property>    <!-- c. 加载所有的映射(根据路径加载)  <property name="mappingLocations">  <list>  <value>classpath:cn/itcast/entity/\*.hbm.xml</value>  </list>  </property>  -->  <!-- c. 根据目录加载所有的映射 -->  <property name=*"mappingDirectoryLocations"*>  <list>  <value>classpath:cn/itcast/entity</value>  </list>  </property>  </bean>    <!-- 创建dao实例 -->  <bean id=*"deptDao"* class=*"cn.itcast.dao.DeptDao"*>  <property name=*"sessionFactory"* ref=*"sessionFactory"*></property>  </bean>    <!-- 创建service实例 -->  <bean id=*"deptService"* class=*"cn.itcast.service.DeptService"*>  <property name=*"deptDao"* ref=*"deptDao"*></property>  </bean>      <!--  Spring声明式事务管理配置  -->  <!-- a. 事务管理器 -->  <bean id=*"txManager"* class=*"org.springframework.orm.hibernate3.HibernateTransactionManager"*>  <property name=*"sessionFactory"* ref=*"sessionFactory"*></property>  </bean>    <!-- b. 事务通知 -->  <tx:advice id=*"txAdvice"* transaction-manager=*"txManager"*>  <tx:attributes>  <tx:method name=*"\*"* read-only=*"false"*/>  </tx:attributes>  </tx:advice>    <!-- c. Aop配置 = 切入点表达式 + 应用通知规则 -->  <aop:config>  <aop:advisor advice-ref=*"txAdvice"* pointcut=*"execution(\* cn..\*Service.\*(..))"*/>  </aop:config>      </beans> |

## \* Spring对dao操作的支持

如下：

1. JDBC

Spring 提供了JdbcTemplate模板工具类，对原始的jdbc操作进行简化！

2. Hibernate

Spring 提供了对hibernate的sessionFactory创建的支持 (整合)

🡪 直接在dao中使用sessionFactory对象操作数据库

🡪 使用Spring提供的 HibernateTemplate 工具类操作数据库

优点： 对session的常用操作进行封装！ 比较方便!

🡪 (推荐)HibernateDaoSupport工具类

Dao类直接继承HibernateDaoSupport工具类即可

HibernateDaoSupport对hibernateTemlate类进行了封装

# 3. SSH整合

SSH 整合：

Spring 与 Struts 整合

🡪 Action创建交给Spring完成

Spring 与 Hibernate整合

🡪 SessionFactory创建，交给spring完成 （管理事务）

步骤：

1. 引入jar文件

Struts核心jar

Hibernate核心jar

Spring

SpringCore 核心jar文件 （5个）

SpringWeb 对struts支持（2个）

SpringAop 声明式事务管理（4个）

SpringORM 对hibernate支持 （3个）

Orm + jdbc + tx jar文件

其他

驱动 + 连接池

2. 配置

Web.xml 配置struts核心过滤器 + Spring容器初始化

Struts.xml 配置访问路径与action类的映射关系

applicationContext-public.xml Spring容器配置 【公用配置】

applicationContext-dao.xml Spring容器配置 【dao配置】

applicationContext-service.xml Spring容器配置 【service配置】

applicationContext-action.xml Spring容器配置 【action配置】

3. 代码

cn.itcast.entity 实体类： 封装数据/业务

cn.itcast.dao 数据访问层接口： 定义功能

cn.itcast.dao.impl 接口实现： 功能实现

cn.itcast.service 业务逻辑层 （控制事务）

cn.itcast.service.impl 实现

cn.itcast.action 控制层： 接收请求数据、处理请求、返回结果视图标记跳转

任务：

Action中写add/update/delete/findById/showAll 方法

1. 数据要在页面输入

2. 查询到的结果要在jsp页面显示