# Hibernate框架

1. Hibernate核心配置文件  
   ORM思想，对象关系映射

--如果单独使用hibernate框架，核心配置文件的名称hibernate.cfg.xml和src下的位置是固定不变的。

--hibernate和spring整合的时候，hibernate的配置文件和位置没有固定要求

2.hibernate的操作步骤

Spring框架对hibernate进行封装，使用hibernateTemplate

# Struuts2

1. Action

1.action的创建的三种方式

继承ActionSupport

2.设置action访问路径

创建struts.xml配置文件，这个文件的位置是固定在src下边的

3.配置访问action的多个方法，使用通配符方式 配置

4.在action获取表单提交数据

得到request对象，使用ServletActionContext类获取

属性封装

模型驱动（重点）

表达式封装

5.action里面怎么操作域对象

使用servletActionContext获取域对象

6.配置struts2的过滤器（StrutsPrepareAndExecuteFilter）

1. 值栈
2. 向值栈中放数据

Set、push、定义变量，生成get方法

2.从值栈中获取数据

--在jsp中使用struts2标签+OGNL获取

--<s:property>

--<s:iterator>

1. 拦截器

1,AOP和责任链模式

2，自定义拦截器

--继承MethodFilterInterceptor

--重写类里面的方法

--配置拦截器和action的关联

# Spring

1. spring核心配置文件
2. 名称和位置没有固定要求
3. 在spring核心配置文件中引入schema约束

2.创建对象

1，xml配置方式，<bean id=”” class=”” scope=””/>

1. 注解方式，四个注解

3.注入属性

1xml配置方式

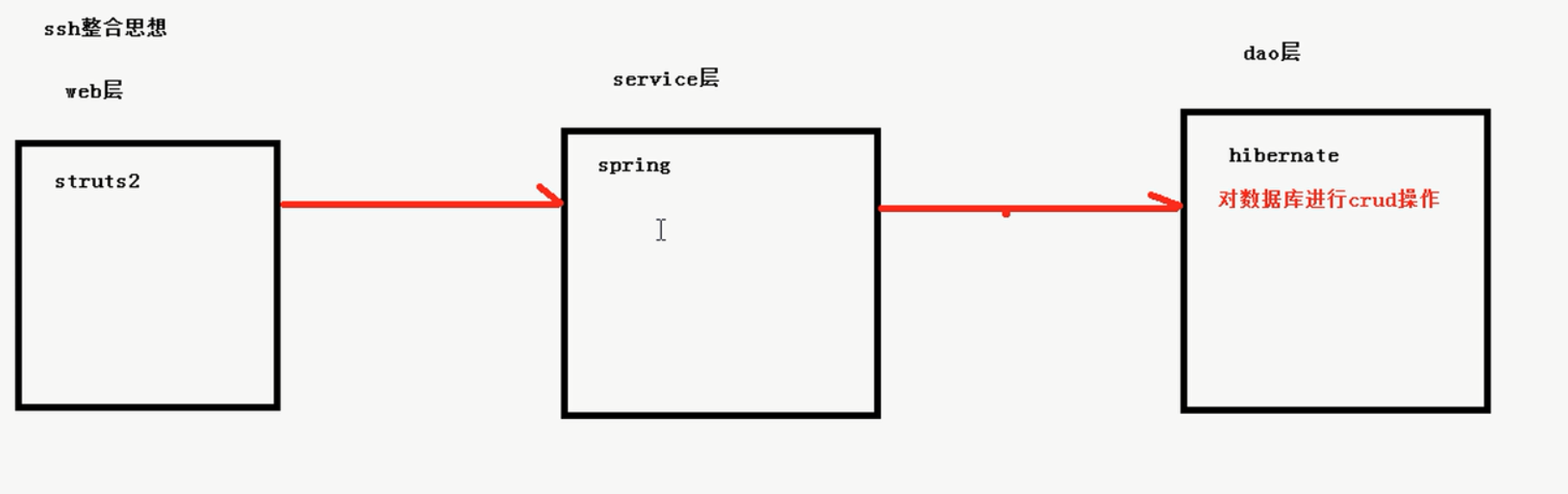
2，注入方式，两个注解

--autoWired

--resource

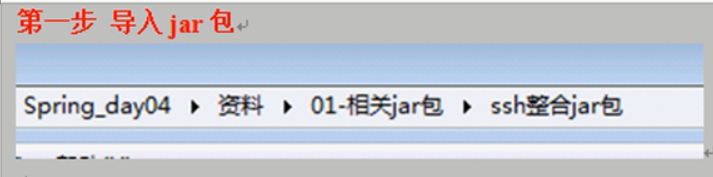
4,使用setvletContext对象和监听器实现

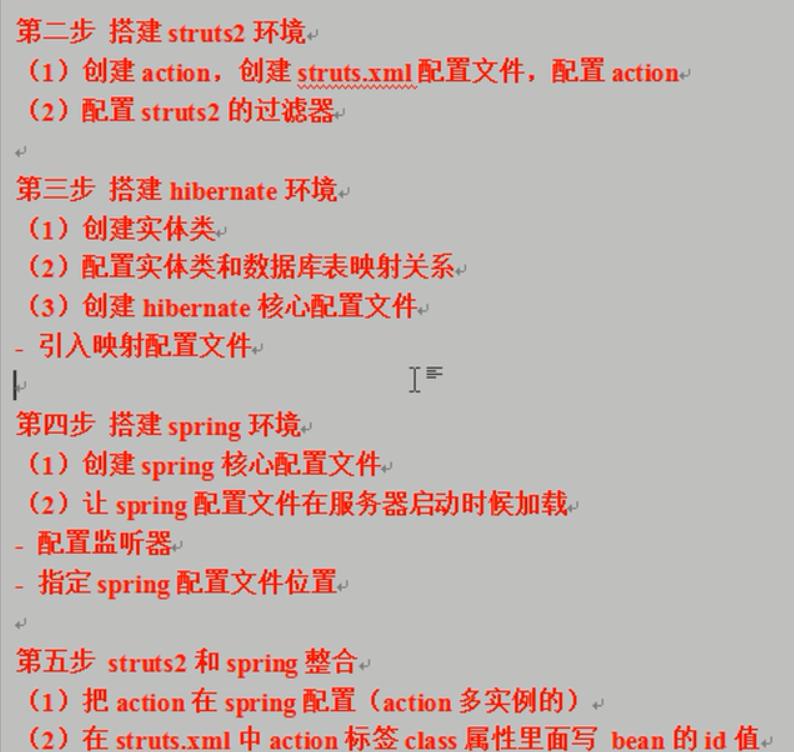
1. 在服务器启动的时候，加载spring配置文件，创建对象
2. 配置spring的监听器



# SSH整合

第一步：导入jar包，注意版本的一致性



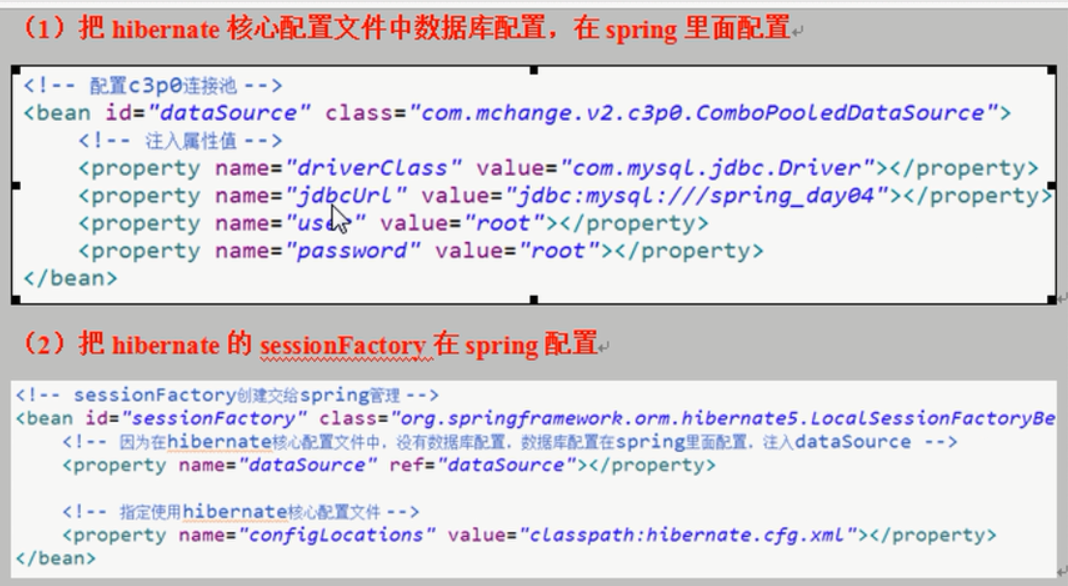


若需要在服务器启动的时候加载spring的配置文件，则需要配置监听器。





第六步：spring和hibernate的整合：







其中web.xml中必须配置struts2的过滤器，总的内容如下

<filter>

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

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

</filter>

<filter-mapping>

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

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

</filter-mapping>

<!--在ContextLoaderListener的源码中找到它的父类，在里面找到param-name的值 -->

<context-param>

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

<param-value>classpath:bean1.xml</param-value>

</context-param>

<!-- 配置监听器 -->

<listener>

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

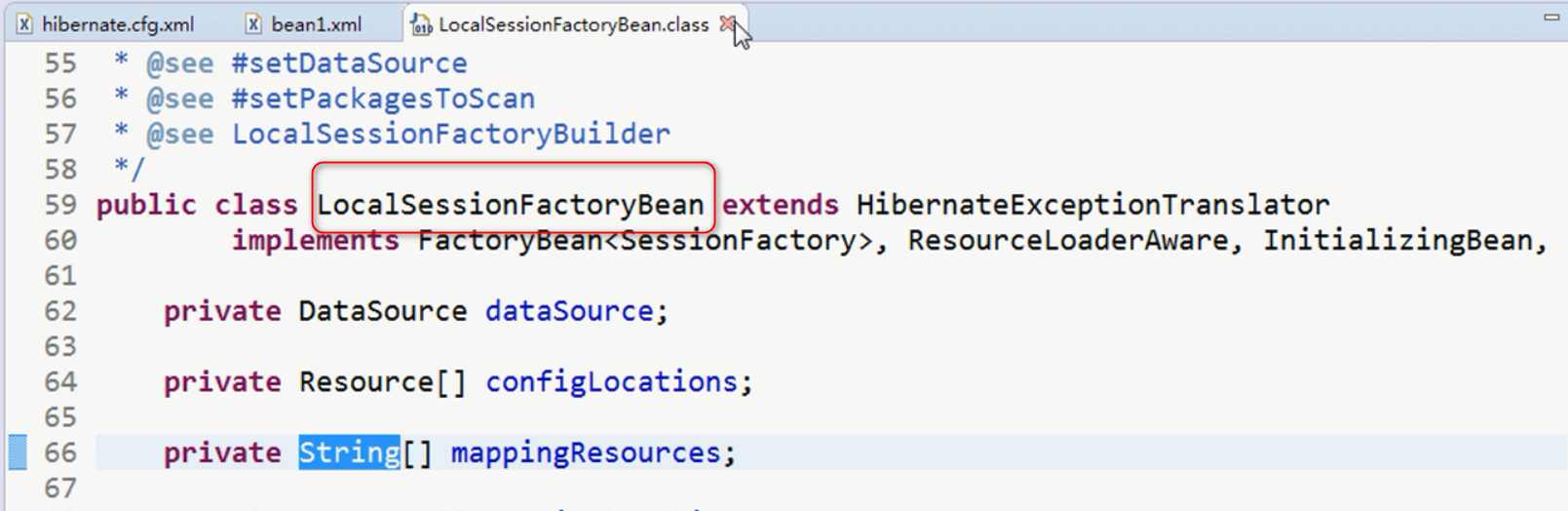
</listener>

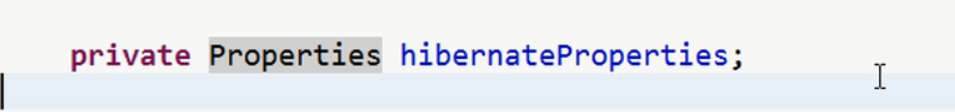
## 整合的其他方式

将hibernate.cfg.xml可以删除，并在spring配置文件总配置出hibernate.cfg.xml里面的信息：



但是这个需要用到LocalSessionFactoryBean这个类中的属性来配置：





但是mappingResource是一个数组，需要用到List标签，而hibernateProperties是一个properties类型的，需要用键值对来配置里面的属性。 其整理后的spring配置文件如下：

<?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: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"*>

<!-- 配置c3p0连接池 -->

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

<!-- 注入里面的属性值 -->

<property name=*"driverClass"* value=*"com.microsoft.sqlserver.jdbc.SQLServerDriver"*></property>

<property name=*"jdbcUrl"* value=*"jdbc:sqlserver://localhost:1433;DatabaseName=struts\_spring"*></property>

<property name=*"user"* value=*"sa"*></property>

<property name=*"password"* value=*"wbs19950305"*></property>

</bean>

<!-- sessionfactory的创建交给spring -->

<bean id=*"sessionFactory"* class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*>

<!-- 因为在hibernate的核心配置文件中没有数据库的配置，所以这儿要注入DataSource -->

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

<!-- 指定使用hibernate的核心配置文件 ,configLocations在localsessionFactoryBean中找-->

<!-- 当没有了hinernate.cfg.xml配置文件的时候，就不需要配置下边这句话了 -->

<!-- <property name="configLocations" value="classpath:hibernate.cfg.xml"></property> -->

<!-- 1配置hibernate的基本信息 -->

<property name=*"hibernateProperties"*>

<props>

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

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

<prop key=*"hibernate.hbm2ddl.auto"*>update</prop><!-- 自动创建表 -->

<prop key=*"hibernate.dialect"*>org.hibernate.dialect.SQLServerDialect</prop>

</props>

</property>

<!-- 2配置映射文件的引入 -->

<property name=*"mappingResources"*>

<list>

<value>com/wbs/entity/user.hbm.xml</value>

<!-- <value>...</value> -->

</list>

</property>

</bean>

<!-- 配置action的创建,scope="prototype"将action设置为多实例的-->

<bean id=*"userAction"* class=*"com.wbs.action.UserAction"* scope=*"prototype"*>

<!-- 注入service -->

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

</bean>

<!-- 创建service对象 -->

<bean id=*"userService"* class=*"com.wbs.service.UserService"*>

<!-- 注入dao对象,name的属性为接口，即：接口=new 实现类 -->

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

</bean>

<!-- 创建实现类的对象 -->

<bean id=*"userDaoimpl"* class=*"com.wbs.dao.UserDaoImpl"*>

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

</bean>

<!-- 创建hibernateTemplate -->

<bean id=*"hibernateTemplate"* class=*"org.springframework.orm.hibernate5.HibernateTemplate"*>

<!-- 注入sessionFactory -->

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

</bean>

<!-- 配置事务 -->

<!-- 第一步：配置事务管理器 -->

<bean id=*"transactionManager"* class=*"org.springframework.orm.hibernate5.HibernateTransactionManager"*>

<!-- 注入sessionFactory,也可以注入DataSource，但是sessionFactory中已经配置并且注入了DataSource

所以在下边的注入中注入有两种方式：

-->

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

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

</bean>

<!-- 第二步：开启事务注解 -->

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

<!-- 第三步：在service上边加上@Transational,也可以加载dao上边 -->

</beans>

## 整合Struts2和Spirng

1. 把struts2的action交给spring管理、
2. Struts.xml如下

<struts>

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

<package name=*"demo1"* extends=*"struts-default"* namespace=*"/demo"*>

<!-- struts的属性里面不写action的路径了，防止action创建两次

但是要写sprig配置的action的id值，前提是必须引入struts-spring-plugs的JAR包

-->

<action name=*"useraction"* class=*"userAction"*></action>

</package>

</struts>

1. 导入struts2的jar包

若在struts.xml中配置了action，又在bean.xml中配置了action，则会多次创建同一个action，当整合的时候，只需要在bean.xml中创建一次就可以的。但是struts.xml中的action的class路径应该是bena.xml中的action的id值，但是必须将其设置为多实例，因为action是多实例的，scope=“prototype”可以设置为多实例。

具体步骤如下：

Bean.xml：

<!-- 配置c3p0连接池 -->

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

<!-- 注入里面的属性值 -->

<property name=*"driverClass"* value=*"com.microsoft.sqlserver.jdbc.SQLServerDriver"*></property>

<property name=*"jdbcUrl"* value=*"jdbc:sqlserver://localhost:1433;DatabaseName=jdbcTemplate"*></property>

<property name=*"user"* value=*"sa"*></property>

<property name=*"password"* value=*"wbs19950305"*></property>

</bean>

<!-- 配置action的创建,scope="prototype"将action设置为多实例的-->

<bean id=*"userAction"* class=*"com.wbs.action.UserAction"* scope=*"prototype"*></bean>

Struts.xml

‘<struts>

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

<package name=*"demo1"* extends=*"struts-default"* namespace=*"/demo"*>

<!-- struts的属性里面不写action的路径了，防止action创建两次

但是要写sprig配置的action的id值，前提是必须引入struts-spring-plugs的JAR包

-->

<action name=*"useraction"* class=*"userAction"*></action>

</package>

</struts>

Web.xml:

<filter>

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

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

</filter>

<filter-mapping>

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

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

</filter-mapping>

<!--在ContextLoaderListener的源码中找到它的父类，在里面找到param-name的值 -->

<context-param>

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

<param-value>classpath:bean1.xml</param-value>

</context-param>

<!-- 配置监听器 -->

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

</listener>

配置web.xml的时候需要注意的几点：

1.必须配置过滤器

2.配置监听器（在服务器启动的时候，就加载配置文件，classpath是指的是配置文件的类路径，但是默认的src路径是默认的类路径，如果在src目录下边的话，就不需要写出类，但是在其他的包、类中的时候，就需写上全路径）

## 整合spring和hibernate

1. 导入JAR包，记得到如jdbc的JAR包
2. Hibernate的核心配置文件如下：

<!-- 核心配置文件 -->

<hibernate-configuration>

<session-factory>

<!-- <property name="hibernate.connection.driver\_class">com.microsoft.sqlserver.jdbc.SQLServerDriver</property>

<property name="hibernate.connection.url">jdbc:sqlserver://localhost:1433;DatabaseName=struts\_spring</property>

<property name="hibernate.connection.username">sa</property>

<property name="hibernate.connection.password">wbs19950305</property> -->

<!-- hibernate可选信息 -->

<property name=*"hibernate.show\_sql"*>true</property>

<property name=*"hibernate.format\_sql"*>true</property>

<property name=*"hibernate.hbm2ddl.auto"*>update</property>

<property name=*"hibernate.dialect"*>org.hibernate.dialect.SQLServerDialect</property>

<!-- <property name="hibernate.current\_session\_context\_class">thread</property> -->

<mapping resource=*"com/wbs/entity/user.hbm.xml"*/>

</session-factory>

</hibernate-configuration>

1. 只在hibernate中的时候，创建sessionfactory的时候，需要写出一个方法或者一个工具类来实现：

**public** **class** HibernateUtils {

**private** **static** SessionFactory factory=**null**;

**private** **static** Configuration cfg=**null**;

**private** **static** Session s=**null**;

**private** **static** ThreadLocal<Session> tl=**new** ThreadLocal<Session>();

//hibernate将程序运行是能够预见的异常都转成了运行时异常

**static**{

**try** {

cfg=**new** Configuration();

cfg.configure();

factory=cfg.buildSessionFactory();

} **catch** (ExceptionInInitializerError e) {

**throw** **new** ExceptionInInitializerError("初始化session失败，请检查配置文件");

}

}

/\*\*

\* 获取一个session对象,保证只有一个session对象

\* 只要使用opensession 每次得到的都是一个新的session

\*/

**public** **static** Session openSession(){

s=tl.get();

**if**(s==**null**){

tl.set(factory.openSession());

}

s=tl.get();

**return** s;

//return factory.openSession();

}

/\*\*

\* 从当前线程上边获取Session对象

\* 当我们把线程和session绑定之后，hibernate在提交或者回滚事务之后，会自动关闭Session

\*/

**public** **static** Session getCurrentSession(){

**return** factory.getCurrentSession();//只有设置了把session和当前线程绑定了之后，才能使用此方法，不然的到的是null

}

**public** **static** **void** main(String[] args) {

}

}

1. 但是在spring整合的过程中，秩序交给服务器加载的时候交给LocalSessionFactoryBean这个类去创建sessionfactory，用到spring的配置文件来得到：

<!-- 配置c3p0连接池 -->

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

<!-- 注入里面的属性值 -->

<property name=*"driverClass"* value=*"com.microsoft.sqlserver.jdbc.SQLServerDriver"*></property>

<property name=*"jdbcUrl"* value=*"jdbc:sqlserver://localhost:1433;DatabaseName=struts\_spring"*></property>

<property name=*"user"* value=*"sa"*></property>

<property name=*"password"* value=*"wbs19950305"*></property>

</bean>

<!-- sessionfactory的创建交给spring -->

<bean id=*"sessionfactory"* class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*>

<!-- 因为在hibernate的核心配置文件中没有数据库的配置，所以这儿要注入DataSource -->

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

<!-- 指定使用hibernate的核心配置文件configLocations在localsessionFactoryBean中找- -->

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

</bean>

注意：LocalSessionFactoryBean的作用就是加载配置文件，创建sessionfactory。帮我们进行封装。但是在其中需要注入DataSource，和指定hibernate.cfg.xml的位置。

1. 整合的过程中会出下边的异常



是因为没有配置事务管理。

事务管理配置如下：

<!-- 配置事务 -->

<!-- 第一步：配置事务管理器 -->

<bean id=*"transactionManager"* class=*"org.springframework.orm.hibernate5.HibernateTransactionManager"*>

<!-- 注入sessionFactory,也可以注入DataSource，但是sessionFactory包含DataSource -->

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

</bean>

<!-- 第二步：开启事务注解 -->

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

<!-- 第三步：在service上边加上@Transational,也可以加载dao上边 -->

1. 当action调用service，service层调用dao层的时候，方法如下：

现在配置文件中创建service和dao对象，并在service中注入dao对象，在dao中注入hibernateTemlate模板，如下：

<!-- 创建service对象 -->

<bean id=*"userService"* class=*"com.wbs.service.UserService"*>

<!-- 注入dao对象,name的属性为接口，即：接口=new 实现类 -->

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

</bean>

<!-- 创建实现类的对象 -->

<bean id=*"userDaoimpl"* class=*"com.wbs.dao.UserDaoImpl"*>

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

</bean>

先写一个dao层的implement接口，然后在写一个实现该接口的dao对象类，

Dao接口：

**public** **interface** UserDao {

**public** **void** add();

}

UserDao:需要在其中注入hibernateTemplate模板

其中bean.xml中的配置如下：

<!-- 创建hibernateTemplate -->

<bean id=*"hibernateTemplate"* class=*"org.springframework.orm.hibernate5.HibernateTemplate"*>

<!-- 注入sessionFactory -->

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

</bean>

具体的实现类如下：

**public** **class** UserDaoImpl **implements** UserDao {

//得到hibernateTemplate对象

**private** HibernateTemplate hibernateTemplate;

**public** **void** setHibernateTemplate(HibernateTemplate hibernateTemplate) {

**this**.hibernateTemplate = hibernateTemplate;

}

**public** **void** add(){

//hibernateTemplate对hibernate进行封装，在没有配置文件的情况下用代码的实现是这样的，

//但是有了配置文件的情况下，直接配置配置文件就可以了，不需要写下边这两行代码了

// HibernateTemplate hibernateTemplate=new HibernateTemplate(sessionFactory);

// hibernateTemplate.save(entity)

//添加操作，临时 创建一个对象，作为测试用

User user=**new** User();

user.setUsername("博盛");

user.setAddress("大连");

hibernateTemplate.save(user);

}

}

注意的问题：

1必须配置事务，因为在 数据提交的时候，若没有配置事务，则hibernateTemplate对象是Null，无法完成对数据库的CRUD操作。

2，在配置事务的时候，可以注入sessionFactory，因为我们已经在服务器启动的时候，创建了sessionfactory对象，并在其中注入了DataSource对象：  
<!-- sessionfactory的创建交给spring -->

<bean id=*"sessionFactory"* class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*>

<!-- 因为在hibernate的核心配置文件中没有数据库的配置，所以这儿要注入DataSource -->

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

<!-- 指定使用hibernate的核心配置文件 ,configLocations在localsessionFactoryBean中找-->

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

</bean>

只需在配置事务的时候，

<!-- 配置事务 -->

<!-- 第一步：配置事务管理器 -->

<bean id=*"transactionManager"* class=*"org.springframework.orm.hibernate5.HibernateTransactionManager"*>

<!-- 注入sessionFactory,也可以注入DataSource，但是sessionFactory中已经配置并且注入了DataSource ，所以在下边的注入中有了两种方式，都可以用的-->

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

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

</bean>

<!-- 第二步：开启事务注解 -->

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

<!-- 第三步：在service上边加上@Transational,也可以加载dao上边 -->

# 利用hibernateTemplate进行操作数据库

前提是要在Dao的对象中注入hibernateTamplate模板：只有注入了模板，才能在dao层进行相应的各种crud操作。

//得到hibernateTemplate对象

**private** HibernateTemplate hibernateTemplate;

**public** **void** setHibernateTemplate(HibernateTemplate hibernateTemplate) {

**this**.hibernateTemplate = hibernateTemplate;

}

## 查询

基本的查询：

save(Object entity)

update(Object entity)

delete (Object entity)

Get方法查询：

1. //get方法，根据ID查询

User user=hibernateTemplate.get(User.**class**, 2);

System.***out***.println("username="+user.getUsername()+"userAddress"+user.getAddress());

Find查询所有（第一个参数是HQL语句，第二个是语句的参数值）

//find方法查询表中的所有

List<User> list=(List<User>)hibernateTemplate.~~find~~("from User");

**for**(User u:list){

System.***out***.println("username="+u.getUsername()+" userAddress"+u.getAddress());

}

Find条件查询：

//find条件查询(查询username=LUCY,若有多个？，在后边写多个值)

List<User> list1=(List<User>)hibernateTemplate.~~find~~("from User where username=?0","LUCY");

**for**(User u:list1){

System.***out***.println("UName::"+u.getUsername());

}

# Spring中分模块开发

1. Spring里面可以配置多个内容，造成配置混乱，不利于维护。
2. 可以吧spring的核心配置中的action和dao放到别的配置文件中，然后在spring的配置文件总引入就可以了。

<!--引入spring的其他配置文件,若在包中要加包名。在src下边不需要些，classpath可以不写 -->

<import resource=*"classpath:user.xml"*/>

# 附加：

1．。。。。

如果在实体类中用注解的方式，则在spring的配置文件applicationContext.xml中的SessionFactory中需要配置packagesToScan这个property,它会自动扫描自己配置的实体类的注解，从而实现配置文件的方式一样的效果，

<?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: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"*>

<!-- 配置spring创建容器时要扫描的包 -->

<context:component-scan base-package=*"com.wbs"*></context:component-scan>

<!--配置hibernateTemplate -->

<bean id=*"hibernateTemplate"* class=*"org.springframework.orm.hibernate5.HibernateTemplate"*>

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

</bean>

<!-- 配置sessionFactory -->

<bean id=*"sessionFactory"* class=*"org.springframework.orm.hibernate5.LocalSessionFactoryBean"*>

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

<property name=*"hibernateProperties"*>

<props>

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

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

<prop key=*"hibernate.hbm2ddl.auto"*>update</prop><!-- 自动创建表 -->

<prop key=*"hibernate.dialect"*>org.hibernate.dialect.SQLServerDialect</prop>

<!-- 把session和线程绑定，从而实现一个线程只有一个session -->

<prop key=*"hibernate.current\_session\_context\_class"*>org.springframework.orm.hibernate5.SpringSessionContext</prop>

</props>

</property>

<property name=*"mappingResources"*>

<list>

<value>com/wbs/domain/customer.hbm.xml</value>

</list>

</property>

<property name=*"packagesToScan"*>

<list>

<value>com.wbs.domain</value>

</list>

</property>

</bean>

<!-- 配置数据源 -->

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

<property name=*"driverClass"* value=*"com.microsoft.sqlserver.jdbc.SQLServerDriver"*></property>

<property name=*"jdbcUrl"* value=*"jdbc:sqlserver://localhost:1433;DatabaseName=struts\_spring"*></property>

<property name=*"user"* value=*"sa"*></property>

<property name=*"password"* value=*"wbs19950305"*></property>

</bean>

<!-- 配置事务 -->

<bean id=*"transactionManager"* class=*"org.springframework.orm.hibernate5.HibernateTransactionManager"*>

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

</bean>

<!-- 开启事务 -->

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

<!-- 在需要使用事务的地方使用@Transaction注解 -->

</beans>

2.。。。。

一般在service层和dao层中先写一个接口，在接口中定义出方法，然后用一个继承它的类去事项它，在具体的实现类中，进行各种操作，

Service接口：

public interface ICustomerService {

/\*\*

\* 查询客户列表

\* @param detachedCriteria

\* @return

\*/

List<Customer> findAllCustomer (DetachedCriteria detachedCriteria);

/\*\*

\* 保存客户

\* @param customer

\*/

void saveCustomer(Customer customer);

}

Dao接口：

/\*\*

\* 客户的持久层接口

\* **@author** bo

\*/

**public** **interface** IcustomerDao {

/\*(\*

\* 查询所有

\*/

List<Customer> findAll(DetachedCriteria detachedCriteria);

/\*\*

\* 保存客户

\* **@param** customer

\*/

**void** save(Customer customer);

}

实现接口的service：

/\*\*

\* 客户的业务层实现类

\* **@author** bo

\*/

@Service("customerService")

@Transactional(readOnly=**true**,propagation=Propagation.***SUPPORTS***)

**public** **class** CustomerServiceImpl **implements** ICustomerService {

@Resource(name="customerdao")

**private** IcustomerDao customerdao;

**public** List<Customer> findAllCustomer(DetachedCriteria detachedCriteria) {

**return** customerdao.findAll(detachedCriteria);

}

@Transactional(readOnly=**false**,propagation=Propagation.***REQUIRED***)

**public** **void** saveCustomer(Customer customer) {

customerdao.save(customer);

}

}

[收藏代码](javascript:void())

[其中的@Transactional(readOnly=**true**,propagation=Propagation.***SUPPORTS***)](mailto:其中的@Transactional(readOnly=true,propagation=Propagation.SUPPORTS))

这句话的意思是只是只读数据库的，即不能进行对数据的添加等操作，只能进行查询的操作。而后边的配置：

在声明式的事务处理中，要配置一个切面，即一组方法，如

<!-- 配置事务的增强,transaction-manager的值必须和事务管理器的id值一样 -->

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

<!-- 做什么事务操作 -->

<tx:attributes>

<!-- 设置进行事务操作的方法匹配规则 ，name是对哪些方法进行增强，比如这个配置是对service中的account\*方法进行事务增强propagation是事务隔离级别，不写也可以-->

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

<!-- <tx:method name="insert\*"/> -->

</tx:attributes>

</tx:advice>   
其中就用到了propagation，表示打算对这些方法怎么使用事务，是用还是不用，其中propagation有七种配置，REQUIRED、SUPPORTS、MANDATORY、REQUIRES\_NEW、NOT\_SUPPORTED、NEVER、NESTED。默认是REQUIRED。   
2.七种配置的意思   
下面是Spring中Propagation类的事务属性详解：   
  
REQUIRED：支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。   
  
SUPPORTS：支持当前事务，如果当前没有事务，就以非事务方式执行。   
  
MANDATORY：支持当前事务，如果当前没有事务，就抛出异常。   
  
REQUIRES\_NEW：新建事务，如果当前存在事务，把当前事务挂起。   
  
NOT\_SUPPORTED：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。   
  
NEVER：以非事务方式执行，如果当前存在事务，则抛出异常。   
  
NESTED：支持当前事务，如果当前事务存在，则执行一个嵌套事务，如果当前没有事务，就新建一个事务。   
3.注意.   
这个配置将影响数据存储，必须根据情况选择。

所以：

### 声明式事务管理（Xml配置）

1. 配置文件使用AOP思想

第一步：配置事务管理器

<!-- 配置事务管理器 -->

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

<!-- 注入dataSource ,ref是连接池的数据源 -->

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

</bean>

第二步：配置事务增强

<!-- 配置事务的增强,transaction-manager的值必须和事务管理器的id值一样 -->

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

<!-- 做什么事务操作 -->

<tx:attributes>

<!-- 设置进行事务操作的方法匹配规则 ，name是对哪些方法进行增强，比如这个配置是对service中的account\*方法进行事务增强propagation是事务隔离级别，不写也可以-->

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

<!-- <tx:method name="insert\*"/> -->

</tx:attributes>

</tx:advice>

### 

第三步：配置切面

<!-- 3.配置切面 -->

<aop:config>

<!-- 切入点 -->

<aop:pointcut expression=*"execution(\* com.wbs.service.OrderService.\*(..))"* id=*"pointcur1"*/>

<!-- 切面,advice-ref是增强来自，必须和增强配置的id一样;pointcut-ref是切面来自，必须和切面的id一致 -->

<aop:advisor advice-ref=*"txadvice"* pointcut-ref=*"pointcur1"*/>

</aop:config>

### 声明式事务管理《注解》

第一步：配置事务管理器

<!-- 第一步：配置事务管理器 -->

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

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

</bean>

第二步：配置事务注解

<!-- 第二步：开启事务的注解 -->

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

第三步：要在使用事务的方法所在的类的上边添加注解

@Transactional

**public** **class** OrderService {

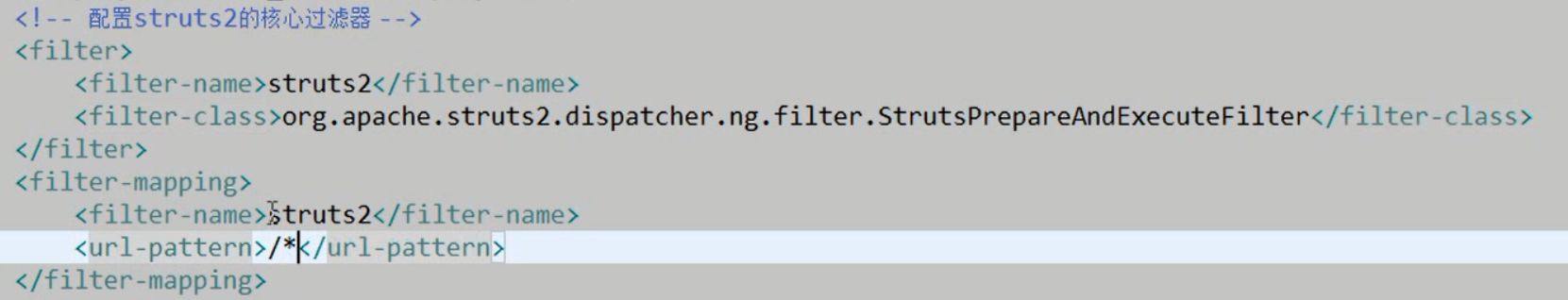
原理：开启事务之后，会在类中找@transactional注解，然后在这个类的所有方法都加上事务。

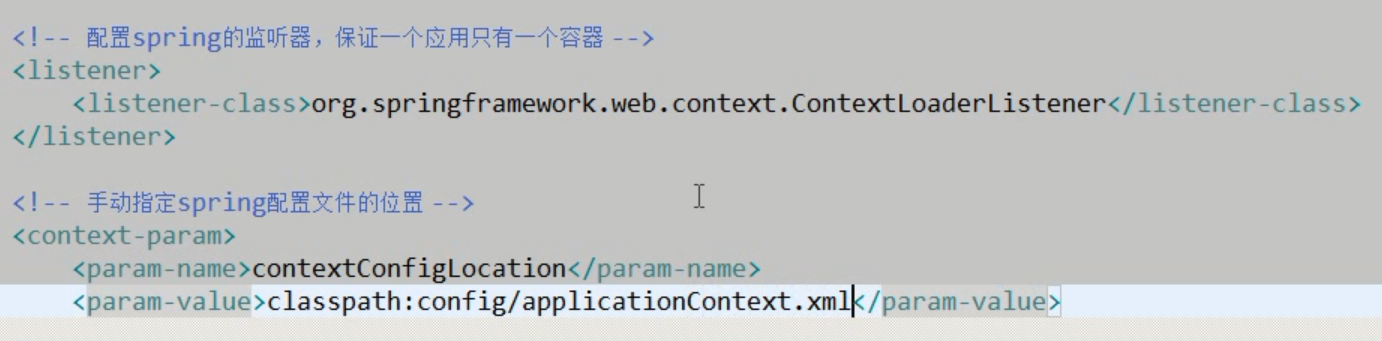
，也可以写成：

@Transactional(readOnly=**true**,propagation=Propagation.***SUPPORTS***)

**public** **class** OrderService {

一定不要忘了在web.xml中配置过滤器和spring的监听器：





Struts2的注解：

