# ssh(struts+spring+hibernate) 三大框架配合使用来开发项目,是目前java ee 最流行的开发方式，必须掌握.

# 每加入一个框架，测试通过

# 1.先搞定spring

## 1.1先引入包

## 1.2编写一个applicationcontext.xml文件，将他放到一个固定的地方

## 1.3测试通过

# 2.引入hibernate，

# 注解：现在是联合作战，和之前的不一样来了，hibernate中的核心将会被spring接管了.hibernate.cfg.xml 文件对象映射文件，SessionFactory在spring的文件中配置即可.

## 2.1配置数据源，spring 配置数据源

### 2.1.1数据源要打开一个连接就要打开一个数据库，用完之后就要关闭，经常这样不太好，Spirng 中上来先创建3个连接，当程序需要一个数据库的时候，就会先打开一个连接，在第一个，第二个还没有释放链接的时候，就会打开第三个连接

<!-- 配置数据源 -->

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

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

<property name=*"url"* value=*"jdbc:mysql://localhost:3306/ssh"*/>

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

<property name=*"password"* value=*"147094"*/>

!-- 下面的东西一般情况下不需要改动 -->

<!-- 连接池启动时的初始值 -->

<property name=*"initialSize"* value=*"30"*/>

<!-- 连接池的最大值 -->

<property name=*"maxActive"* value=*"500"*/>

<!-- 最大空闲值.当经过一个高峰时间后，连接池可以慢慢将已经用不到的连接慢慢释放一部分，一直减少到maxIdle为止 -->

<property name=*"maxIdle"* value=*"2"*/>

<!-- 最小空闲值.当空闲的连接数少于阀值时，连接池就会预申请去一些连接，以免洪峰来时来不及申请 -->

<property name=*"minIdle"* value=*"1"*/>

</bean>

## 2.2配置SessionFactory对象

<!-- 配置会话工厂() -->

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

<!-- 设置数据源 -->

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

<!-- 接管了hibernate对象映射文件 -->

<property name=*"mappingResources"*>

<list>

<value>com/hsp/domain/Employee.hbm.xml</value>

<value>com/hsp/domain/Department.hbm.xml</value>

</list>

</property>

<property name=*"hibernateProperties"*>

<value>

hibernate.dialect=org.hibernate.dialect.MySQLDialect

hibernate.hbm2ddl.auto=update

hibernate.show\_sql=true

<!-- hibernate.cache.use\_second\_level\_cache=true

hibernate.cache.provider\_class=org.hibernate.cache.EhCacheProvider

hibernate.generate\_statistics=true -->

</value>

</property>

### 2.3进行测试导入到employee中数据

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

ApplicationContext ac=**new** ClassPathXmlApplicationContext("applicationContext.xml");

SessionFactory sf = (SessionFactory) ac.getBean("sessionFactory");

Session session = sf.openSession();

Employee employee=**new** Employee("hlj2","aa@sohu.com","145",1,**new** java.util.Date(),234.56f);

/\*Transaction ts = session.beginTransaction();\*/

session.save(employee);

/\*ts.commit();\*/

session.close();

}

# 3.先写接口

**public** **class** EmployeeService **implements** EmployeeServiceInter {

**public** SessionFactory sessionFactory;

**public** **void** setSessionFactory(SessionFactory sessionFactory) {

**this**.sessionFactory = sessionFactory;

}

@Override

**public** **void** addEmployList(Employee e) {

// **TODO** Auto-generated method stub

Session session = sessionFactory.openSession();

Transaction tsTransaction= session.beginTransaction();

session.save(e);

tsTransaction.commit();

//session 不关闭，因为还要使用

/\*session.close();\*/

}

## 3.2.然后配置spring

<!-- 配置EmployeeService对象 -->

<bean id=*"employeeService"* class=*"com.hlj.service.imp.EmployeeService"*>

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

</bean>

## 3.3.进行测试

Employee employee=**new** Employee("hlj3","aa@sohu.com","145",1,**new** java.util.Date(),234.56f);

EmployeeServiceInter employeeServiceInter=(EmployeeServiceInter) ac.getBean("employeeService");

employeeServiceInter.addEmployList(employee);

# 4使用事物来进行管理，配置事物管理器，统一管理sessionfactory的事物

<!-- 配置事务管理器，统一管理sessionFactory的事务 -->

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

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

</bean>

## 4.1启动事物的注解

<!-- 配置事务管理器，统一管理sessionFactory的事务 -->

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

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

</bean>

<!-- 启用事务注解 -->

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

//这个配置@Transactional 是让事物管理器来管理service 事物

@Transactional

**public** **class** EmployeeService **implements** EmployeeServiceInter {

**public** SessionFactory sessionFactory;

## 4.2测试

**public** **void** addEmployList(Employee e) {

// **TODO** Auto-generated method stub

sessionFactory.getCurrentSession().save(e);

**private** **static** **void** TestTransation(ApplicationContext ac) {

Employee employee=**new** Employee("hlj3","aa@sohu.com","145",1,**new** java.util.Date(),234.56f);

EmployeeServiceInter employeeServiceInter=(EmployeeServiceInter) ac.getBean("employeeService");

employeeServiceInter.addEmployList(employee);

}

# 5.Struts

## 5.1先引入包

## 5.2创建struts-cofig.xml文件 然后手动建立相应的form 和action 使我们可以登录跳转

# 6.开始使用Struts 和spring hibernate 联合进行用户登录验证

## 6.1在web容器中实例化spring容器（因为我们如果使用ApplactionContext,每次调用不合算，在我们直接将他进行初始化）让我们的服务器进行初始化的时候，进行实例化，这样我们的spring容器就会和 tomcat的 时间是一样长的

<!-- 指定spring的配置文件，默认从web根目录寻找配置文件，我们可以通过spring提供的classpath:前缀指定从类路径下寻找 -->

<context-param>

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

<!-- 下面这个 是唯一进行改变的地方-->

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

</context-param>

<!-- 对Spring容器进行实例化 -->

<listener>

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

</listener>

## 6.2获取web 中的spring容器

//下面这个是没有在 web 中进行配置的时候使用的

// ApplicationContext ac= new ClassPathXmlApplicationContext("applicationContext.xml");

//如果在web中进行配置了之后 通过下面的就可以获取 spring 容器了

WebApplicationContext wc =

WebApplicationContextUtils

.*getWebApplicationContext*(**this**.getServlet().getServletContext());

## 6.3数据库验证登录

**public** ActionForward login(ActionMapping mapping, ActionForm form,

HttpServletRequest request, HttpServletResponse response)

**throws** Exception {

EmployeeForm employeeForm=(EmployeeForm)form;

//下面这个是没有在 web 中进行配置的时候使用的

// ApplicationContext ac= new ClassPathXmlApplicationContext("applicationContext.xml");

//如果在web中进行配置了之后 通过下面的就可以获取 spring 容器了

WebApplicationContext cxt =

WebApplicationContextUtils .*getWebApplicationContext*(**this**.getServlet().getServletContext());

//从容器中获取bean

Employee employee = **new** Employee();

employee.setId(Integer.*parseInt*(employeeForm.getId()));

employee.setPwd(employeeForm.getPwd());

System.*out*.println(employee);

EmployeeServiceInter employeeServiceInter=(EmployeeServiceInter) cxt.getBean("employeeService");

employee = employeeServiceInter.check(employee);

**if**(employee!=**null**){

request.getSession().setAttribute("loginuser", employee);

**return** mapping.findForward("ok");

}**else**{

**return** mapping.findForward("err");

}

# 7.让spring接管Struts

## 7.1在struts-config.xml文件中添加如下代码配置:

### 7.1.1配置代理请求处理，Struts会去查询 config文件，查询总控中的action文件，下面这个就是请求代理，如果发现有login.do ，就会去找spring容器不会直接去找Struts中的action了，到了spring容器中applicationContext.xml 找

<!-- 配置代理请求处理 DelegatingRequestProcessor ,它的用户是 -->

<controller>

<set-property property=*"processorClass"* value=*"org.springframework.web.struts.DelegatingRequestProcessor"*/>

</controller>

### 7.1.2 applicationContext.xml 中配置action的路径

<!-- 配置action 下面的name就是path 路径 -->

<bean name=*"/login"* scope=*"prototype"* class=*"com.hlj.web.action.LoginAction"*/>

### 7.1.3 然后Struts-config.xml中的type 路径就没有用了

<action-mappings>

<action

path=*"/login"*

parameter=*"flag"*

name=*"employeeForm"* >

# 8这样action初始化就不会交给 Struts了，而是交给了spring了这样就可以 通过Spring 来获取bean，也就是下面这个 接口来了，将loginAction 当做bean来处理

**public** **class** LoginAction **extends** DispatchAction {

EmployeeServiceInter employeeServiceInter;

**public** EmployeeServiceInter getEmployeeServiceInter() {

**return** employeeServiceInter;

}

**public** **void** setEmployeeServiceInter(EmployeeServiceInter employeeServiceInter) {

**this**.employeeServiceInter = employeeServiceInter;

}

## 8.1到了spring 容器中进行接口的配置，然后我们到了logingAction的界面就可以直接使用了

<!-- 配置action 下面的name就是path 路径 -->

<bean name=*"/login"* class=*"com.hlj.web.action.LoginAction"*>

<property name=*"employeeServiceInter"* ref=*"employeeService"*/>

</bean>

## 8.2可以解决action是单态的问题（不管有多少个请求只有一个acting为你服务） 解决方法：prototype

通过使用sping来接管我们的action,还有一个好处,可以解决action 是单例的问题.通过在applicationContext.xml文件中配置属性

<bean scope=”singlton/prototype/request/sesssion/global session”/>

<!-- 配置action 下面的name就是path 路径 -->

<bean name=*"/login"* scope=*"prototype"* class=*"com.hlj.web.action.LoginAction"*>

<!-- 将action作为 bean 进行 注 入接口 -->

<property name=*"employeeServiceInter"* ref=*"employeeService"*/>

</bean

>

## 8.2.3如果配置一个prototype 那么 a 就不会因为重复的提交二增加了 ，这样表示不止一个action为我们服务

System.*out*.println("证明 scope 是一个单例 如果 a 会增加那么这个a 不是 一个单态");

System.*out*.println("a = "+(++a));

# 9.中文乱码的问题过滤器是属于web层的问题

## 9.1过滤器解决（我们会）

## 9.2使用spring框架提供的处理中文乱码的过滤器，相当于直接打了一个包让我们使用

### 在web文件中

<!-- 通过spring解决中文乱码的问题 -->

<filter>

<filter-name>encoding</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encoding</filter-name>

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

</filter-mapping>