# 1.[HibernateDaoSupport类的使用](http://www.cnblogs.com/azai/archive/2011/03/25/1995158.html)

1. 1、        继承了HibernateDaoSupport类的类获取session时，已不可用SessionFactory.OpenSessioon的形式来获 取Session了，由于HibernateDaoSupport本身已有获取session的方法getSession(),所以直接用Session se=this.getSession();来获取，
2. 2、        在依据hql获取用户信息时，继承了HibernateDaoSupport类的类中不能在使用Query类了，而是用List<Ssh> list = this.getHibernateTemplate().find(hql);形式来获取实体类集合

2.  getGenericSuperclass() 通过反射获取当前类表示的实体（类，接口，基本类型或void）的直接父类的Type，getActualTypeArguments()返回参数数组。

这段代码是这个意思，看是不是支持泛型的，Type type = this.getClass().getGenericSuperclass(); 是得到这个类的得到泛型父类 ，if(!(type instanceof ParameterizedType))，这行代码的意思是  
如果没有实现ParameterizedType接口,即不支持泛型；Type[] params = ((ParameterizedType) genType).getActualTypeArguments();这行代码的意思是，如果支持泛型，返回表示此类型实际类型参数的Type对象的数组,数组里放的都是对应类型的Class，因为可能有多个，所以是数组。  
后面的应该你很容易就看懂了。

超类就是父类了，  
getActualTypeArguments()就是获取泛型参数的类型  
  
比如Test<T>，最后就得到T类型

# 3.

在jsp页面中，即便用struts标签来代替字符串的值，字符串的双引号（单引号）也是不能够省略的。

# 4.

org.springframework.dao.InvalidDataAccessApiUsageException: The given object has a null identifier: com.paixie.domain.Users;

                出现这个异常，一般来说主要修改时候主键没有传递或者name属性指定错误。

            解决方案：你要把你的更新的实例的主键传过去，可以以hidden的方式传。

**[java]** [view plain](http://blog.csdn.net/chenssy/article/details/8362871) [copy](http://blog.csdn.net/chenssy/article/details/8362871)

[print?](http://blog.csdn.net/chenssy/article/details/8362871)

<s:hidden name="action中实例对象名.id" id="id"></s:hidden>

# 5

tomcat在关闭应用时，对资源做了一些清理，避免了泄露，这个工作主要是WebappClassLoader里做的，WebappClassLoader也实现自Lifecycle接口，在应用关闭时，会触发其stop方法，其中对JDBC Driver的清理，是clearReferencesJdbc方法，它检查当前WebappClassLoader加载过的，在关闭时未注销掉的JDBC Driver，给出警告信息，并强行将这些Driver反注册掉。如果servlet在初始化时注册了一个Driver，但销毁时未将这个Driver给反注册掉；这时不管是显式的通过命令来stop tomcat，还是因为设置了自动reload，而且恰好检查到应用有变，执行了reload的时候(reload也是对app context进行stop，然后再重新start)，就会被tomcat判断为泄露，给出警告并强制反注册Driver。

    要避免这个信息，应用或框架应该自己来保证在销毁时将JDBC Driver反注册掉。例如在destroy方法里：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @Override      public void destroy() {          super.destroy();          try{              DriverManager.deregisterDriver(DriverManager.getDrivers().nextElement());          }catch(Exception e){              e.printStackTrace();          }      } |

   因为tomcat自带了DBCP数据库连接池，很多用户在使用DBCP时遇到了这个问题，并建议在 DBCP 的 BasicDataSource的close方法里执行反注册驱动的行为来解决这个警告。但DBCP的开发者认为这个应该是使用者的责任，不愿意接受这种建议。

# 6

org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'sessionFactory' defined in class path resource [applicationContext.xml]: Invocation of init method failed; nested exception is org.hibernate.MappingException: Duplicate property mapping of dept found in cn.itcast.nsfw.user.entity.User

解决方案：实体类的\*.hbm.xml中某个属性字段配置有重复

# 7

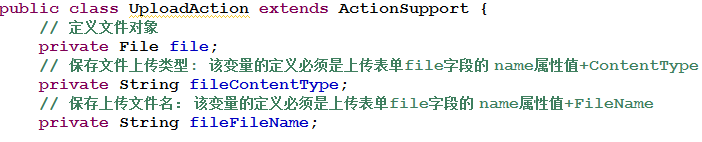
在struts中实现文件上传时的步骤：

表单属性 enctype = **multipart/form-data 表单类型**

表单属性 method = **post 提交方式**

输入属性 type = **file 文件域**

在java文件中必须设置的3个属性：：同时必须提供get/set方法



# 8

<s:param name="value"/> name用来设置属性，value指定当前标签的上一级标签的属性值（即应该在页面中显示的内容）

# 9

在一个文件中的扩展名的大小写，系统是不予区别的

正则表达式的相关用法：

　(?i)即匹配时不区分大小写。表示匹配时不区分大小写。  
  
　　(?s)即Singleline(单行模式)。表示更改.的含义，使它与每一个字符匹配（包括换行 符\n）。  
  
　　(?m)即Multiline(多行模式) 。 表示更改^和$的 含义，使它们分别在任意一行的行首和行尾匹配，而不仅仅在整个字符串的开头和结尾匹配。(在此模式下,$的 精确含意是:匹配\n之前的位置以及字符串结束前的位置.)   
　　(?x)：表示如果加上该修饰符，表达式中的空白字符将会被忽略，除非它已经被转义。   
　　(?e)：表示本修饰符仅仅对于replacement有用，代表在replacement中作为PHP代码。   
　　(?A)：表示如果使用这个修饰符，那么表达式必须是匹配的字符串中的开头部分。比如说"/a/A"匹配"abcd"。   
　　(?E)：与"m"相反，表示如果使用这个修饰符，那么"$"将匹配绝对字符串的结尾，而不是换行符前面，默认就打开了这个模式。   
　　(?U)：表示和问号的作用差不多，用于设置"贪婪模式"。

# 10

# [Ajax请求中的async:false/true的作用](http://blog.csdn.net/yanfangphp/article/details/8261333)

async: false,（默认是true）;  
如上：false为同步，这个 testAsync()方法中的Ajax请求将整个浏览器锁死，  
只有tet.php执行结束后，才可以执行其它操作。  
  
当async: true 时，ajax请求是异步的。但是其中有个问题：testAsync()中的ajax请求和其后面的操作是异步执行的，那么当tet.php还未执行完，就可能已经执行了 ajax请求后面的操作，

async. 默认是true，即为异步方式，$.ajax执行后，会继续执行ajax后面的脚本，直到服务器端返回数据后，触发$.ajax里的success方法，这时候执行的是两个线程。若要将其设置为false  
，则所有的请求均为同步请求，在没有返回值之前，同步请求将锁住浏览器，用户其它操作必须等待请求完成才可以执行。

# 11

# [js中return false,return,return true的用法及区别](http://blog.csdn.net/yulei_qq/article/details/45317579)

**1.返回控制与函数结果**

语法为：**return 表达式**; 语句结束函数执行，返回调用函数，而且把表达式的值作为函数的结果

#### 2.返回控制无函数结果

#### 语法为：return;在大多数情况下,为事件处理函数返回false,可以防止默认的事件行为.例如,默认情况下点击一个<A>元素,页面会跳转到该元素href属性指定的页. return false 就相当于终止符，return true 就相当于执行符。在js中return false的作用一般是用来取消默认动作的。比如你单击一个链接除了触发你的"onclick"事件以外还要触发一个默认事件就是执行页面的跳转。所以如果你想取消对象的默认动作就可以return false。也就是说如果你想用JS代码来局部改变一些数据而不引起页面其他部位的变化，那么你就应该在该onclick事件代码的后面加上return false；

在js中，我们常用return false来**阻止提交表单**或**者继续执行下面的代码，**通俗的来说就是阻止执行默认的行为。 例如下面的例子：

**总的来说在js中对于return用法的三种情况的总结如下：**

**retrun true； 返回正确的处理结果。**

**return false；返回错误的处理结果；终止处理；阻止提交表单；阻止执行默认的行为。**

**return；把控制权返回给页面。**

# 12用EL表达式取出valuestack中的值

struts2中可以[struts2](https://www.baidu.com/s?wd=struts2&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d-rAmvnhfdPjD1PHRsn16z0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPj64P16znWcd" \t "_blank)中action及其属性会被包装成actionContext，在actionContext中有request、session等各个域，action中的属性会同步到request中，也就是放到了request作用域中，所以[el表达式](https://www.baidu.com/s?wd=el%E8%A1%A8%E8%BE%BE%E5%BC%8F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d-rAmvnhfdPjD1PHRsn16z0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPj64P16znWcd" \t "_blank)也可以取出。

**自定义Result返回类型(StrutsResultSupport)**

总的来说，写一个自定义的Result Type有三个步骤：

（1）写一个实现了Result接口的类

（2）对该类进行注册

（3）使用该类

在有些特殊情况下，如果没有异常信息，但是有错误，并且有错误信息等内容，此时也需要进行友好的错误处理的话，那么可以借助StrutsResultSupport 返回结果类型来实现特定处理。此种方式先需要继承StrutsResultSupport ，然后可以在子类中获取本次请求的相关信息，再根据相关信息进行结果处理：

## **1.1、自定义实现了Result接口的类**

public class SysResult extends StrutsResultSupport {

    @Override

    protected void doExecute(String finalLocation, ActionInvocation invocation)

            throws Exception {

        HttpServletRequest request = ServletActionContext.getRequest();

        HttpServletResponse response = ServletActionContext.getResponse();

        BaseAction action = (BaseAction)invocation.getAction();

        // do something

        System.out.println("进入了SysResult...");

    }

}

1.2、注册和使用添加的Result Type类

<!-- 配置全局结果及异常映射 -->

<package name="base-default" extends="struts-default" abstract="true">

<!-- 返回结果类型 -->

<result-types>

<result-type name="sys-error" class="com.rk.core.action.SysResult"></result-type>

</result-types>

<!-- 全局返回结果 -->

<global-results>

<result name="error" type="sys-error">/WEB-INF/jsp/error.jsp</result>

<result name="input">/WEB-INF/jsp/error.jsp</result>

</global-results>

<!-- 全局异常映射 -->

<global-exception-mappings>

<exception-mapping result="input" exception="java.lang.Exception"></exception-mapping>

</global-exception-mappings>

</package>

其中注册的是

|  |  |
| --- | --- |
| 1 | <result-type name="sys-error" class="com.rk.core.action.SysResult"></result-type> |

其中使用的声明是

|  |  |
| --- | --- |
| 1 | <result name="error" type="sys-error">/WEB-INF/jsp/error.jsp</result> |

# 13

pplicationContext是Spring的核心，Context我们通常解释为上下文环境，我想用“容器”来表述它更容易理解一些，ApplicationContext则是“应用的容器”了；在Web应用中，我们会用到WebApplicationContext，WebApplicationContext继承自ApplicationContext；WebApplicationContext的初始化方式和BeanFactory.ApplicationContext有所区别,因为WebApplicationContext需要ServletContext实例,也就是说它必须拥有Web容器的前提下才能完成启动的工作.有过Web开发经验的读者都知道可以在web.xml中配置自启动的Servlet或定义Web容器监听器(ServletContextListener),借助着两者中的任何一个,我们就可以启动Spring Web应用上下文的工作.

[**spring**](http://lib.csdn.net/base/javaee)分别提供了用于启动WebApplicationContext的Servlet和Web容器监听器:

org.springframework.web.context.ContextLoaderServlet;

org.springframework.web.context.ContextLoaderListener.

这两个方法都是在web应用启动的时候来初始化WebApplicationContext，我个人认为Listerner要比Servlet更好一些，因为Listerner监听应用的启动和结束，而Servlet得启动要稍微延迟一些，如果在这时要做一些业务的操作，启动的前后顺序是有影响的。

配置例子如下：  
context-param>   
<param-name>contextConfigLocation</param-name>   
<param-value>/WEB-INF/applicationContext.xml</param-value>   
</context-param>   
  
<listener>   
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>   
</listener>

那么在ContextLoaderListener和ContextLoaderServlet中到底做了什么呢？   
以ContextLoaderListener为例，我们可以看到   
public void contextInitialized(ServletContextEvent event) {   
this.contextLoader = createContextLoader();   
this.contextLoader.initWebApplicationContext(event.getServletContext());   
}   
protected ContextLoader createContextLoader() {   
return new ContextLoader();   
}   
ContextLoader是一个工具类，用来初始化WebApplicationContext，其主要方法就是initWebApplicationContext，我们继续追踪initWebApplicationContext这个方法（具体代码我不贴出，大家可以看Spring中的源码），我们发现，原来ContextLoader是把WebApplicationContext（XmlWebApplicationContext是默认实现类）放在了ServletContext中，ServletContext也是一个“容器”，也是一个类似Map的结构，而WebApplicationContext在ServletContext中的KEY就是WebApplicationContext.ROOT\_WEB\_APPLICATION\_CONTEXT\_ATTRIBUTE，我们如果要使用WebApplicationContext则需要从ServletContext取出，Spring提供了一个WebApplicationContextUtils类，可以方便的取出WebApplicationContext，只要把ServletContext传入就可以了。

# 14

<a href="javascript:doGoPage(<s:property value='pageResult.pageNo+1'/>)">下一页</a>

Javascript和函数之间**不能有空格**

# 15

## 错误原因是在Action页面！！！错误在于你添加的sql查询语句（或sql条件是错误的）

java.lang.ClassCastException: java.lang.String cannot be cast to java.util.Date

at org.hibernate.type.descriptor.java.JdbcTimestampTypeDescriptor.unwrap(JdbcTimestampTypeDescriptor.java:42)

at org.hibernate.type.descriptor.sql.TimestampTypeDescriptor$1.doBind(TimestampTypeDescriptor.java:53)

at org.hibernate.type.descriptor.sql.BasicBinder.bind(BasicBinder.java:89)

at org.hibernate.type.AbstractStandardBasicType.nullSafeSet(AbstractStandardBasicType.java:282)

at org.hibernate.type.AbstractStandardBasicType.nullSafeSet(AbstractStandardBasicType.java:277)

at org.hibernate.param.PositionalParameterSpecification.bind(PositionalParameterSpecification.java:68)

at org.hibernate.loader.hql.QueryLoader.bindParameterValues(QueryLoader.java:571)

at org.hibernate.loader.Loader.prepareQueryStatement(Loader.java:1716)

at org.hibernate.loader.Loader.doQuery(Loader.java:801)

at org.hibernate.loader.Loader.doQueryAndInitializeNonLazyCollections(Loader.java:274)

at org.hibernate.loader.Loader.doList(Loader.java:2533)

at org.hibernate.loader.Loader.listIgnoreQueryCache(Loader.java:2276)

at org.hibernate.loader.Loader.list(Loader.java:2271)

at org.hibernate.loader.hql.QueryLoader.list(QueryLoader.java:452)

at org.hibernate.hql.ast.QueryTranslatorImpl.list(QueryTranslatorImpl.java:363)

at org.hibernate.engine.query.HQLQueryPlan.performList(HQLQueryPlan.java:196)

at org.hibernate.impl.SessionImpl.list(SessionImpl.java:1268)

at org.hibernate.impl.QueryImpl.list(QueryImpl.java:102)

at cn.itcast.core.dao.impl.BaseDaoImpl.getPageResult(BaseDaoImpl.java:92)

at cn.itcast.core.service.impl.BaseServiceImpl.getPageResult(BaseServiceImpl.java:60)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)

at java.lang.reflect.Method.invoke(Method.java:597)

at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:309)

at org.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint(ReflectiveMethodInvocation.java:183)

at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:150)

at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:110)

at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:172)

at org.springframework.aop.interceptor.ExposeInvocationInterceptor.invoke(ExposeInvocationInterceptor.java:89)

at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:172)

at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:202)

at $Proxy8.getPageResult(Unknown Source)

at cn.itcast.nsfw.complain.action.ComplainAction.listUI(ComplainAction.java:77)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)

at java.lang.reflect.Method.invoke(Method.java:597)

at ognl.OgnlRuntime.invokeMethod(OgnlRuntime.java:891)

at ognl.OgnlRuntime.callAppropriateMethod(OgnlRuntime.java:1293)

at ognl.ObjectMethodAccessor.callMethod(ObjectMethodAccessor.java:68)

at com.opensymphony.xwork2.ognl.accessor.XWorkMethodAccessor.callMethodWithDebugInfo(XWorkMethodAccessor.java:117)

at com.opensymphony.xwork2.ognl.accessor.XWorkMethodAccessor.callMethod(XWorkMethodAccessor.java:108)

at ognl.OgnlRuntime.callMethod(OgnlRuntime.java:1369)

at ognl.ASTMethod.getValueBody(ASTMethod.java:90)

at ognl.SimpleNode.evaluateGetValueBody(SimpleNode.java:212)

at ognl.SimpleNode.getValue(SimpleNode.java:258)

at ognl.Ognl.getValue(Ognl.java:494)

at ognl.Ognl.getValue(Ognl.java:458)

at com.opensymphony.xwork2.ognl.OgnlUtil$2.execute(OgnlUtil.java:309)

at com.opensymphony.xwork2.ognl.OgnlUtil.compileAndExecute(OgnlUtil.java:340)

at com.opensymphony.xwork2.ognl.OgnlUtil.getValue(OgnlUtil.java:307)

at com.opensymphony.xwork2.DefaultActionInvocation.invokeAction(DefaultActionInvocation.java:423)

at com.opensymphony.xwork2.DefaultActionInvocation.invokeActionOnly(DefaultActionInvocation.java:287)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:250)

at org.apache.struts2.interceptor.DeprecationInterceptor.intercept(DeprecationInterceptor.java:41)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at org.apache.struts2.interceptor.debugging.DebuggingInterceptor.intercept(DebuggingInterceptor.java:256)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at com.opensymphony.xwork2.interceptor.DefaultWorkflowInterceptor.doIntercept(DefaultWorkflowInterceptor.java:167)

at com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at com.opensymphony.xwork2.validator.ValidationInterceptor.doIntercept(ValidationInterceptor.java:265)

at org.apache.struts2.interceptor.validation.AnnotationValidationInterceptor.doIntercept(AnnotationValidationInterceptor.java:76)

at com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at com.opensymphony.xwork2.interceptor.ConversionErrorInterceptor.intercept(ConversionErrorInterceptor.java:138)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at com.opensymphony.xwork2.interceptor.ParametersInterceptor.doIntercept(ParametersInterceptor.java:229)

at com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at com.opensymphony.xwork2.interceptor.ParametersInterceptor.doIntercept(ParametersInterceptor.java:229)

at com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at com.opensymphony.xwork2.interceptor.StaticParametersInterceptor.intercept(StaticParametersInterceptor.java:191)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at org.apache.struts2.interceptor.MultiselectInterceptor.intercept(MultiselectInterceptor.java:73)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at org.apache.struts2.interceptor.DateTextFieldInterceptor.intercept(DateTextFieldInterceptor.java:125)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at org.apache.struts2.interceptor.CheckboxInterceptor.intercept(CheckboxInterceptor.java:91)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at org.apache.struts2.interceptor.FileUploadInterceptor.intercept(FileUploadInterceptor.java:253)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at com.opensymphony.xwork2.interceptor.ModelDrivenInterceptor.intercept(ModelDrivenInterceptor.java:100)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at com.opensymphony.xwork2.interceptor.ScopedModelDrivenInterceptor.intercept(ScopedModelDrivenInterceptor.java:141)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at com.opensymphony.xwork2.interceptor.ChainingInterceptor.intercept(ChainingInterceptor.java:145)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at com.opensymphony.xwork2.interceptor.PrepareInterceptor.doIntercept(PrepareInterceptor.java:171)

at com.opensymphony.xwork2.interceptor.MethodFilterInterceptor.intercept(MethodFilterInterceptor.java:98)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at com.opensymphony.xwork2.interceptor.I18nInterceptor.intercept(I18nInterceptor.java:139)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at org.apache.struts2.interceptor.ServletConfigInterceptor.intercept(ServletConfigInterceptor.java:164)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at com.opensymphony.xwork2.interceptor.AliasInterceptor.intercept(AliasInterceptor.java:193)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at com.opensymphony.xwork2.interceptor.ExceptionMappingInterceptor.intercept(ExceptionMappingInterceptor.java:189)

at com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:244)

at org.apache.struts2.impl.StrutsActionProxy.execute(StrutsActionProxy.java:54)

at org.apache.struts2.dispatcher.Dispatcher.serviceAction(Dispatcher.java:564)

at org.apache.struts2.dispatcher.ng.ExecuteOperations.executeAction(ExecuteOperations.java:81)

at org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter.doFilter(StrutsPrepareAndExecuteFilter.java:99)

at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:235)

at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:206)

at cn.itcast.core.filter.LoginFilter.doFilter(LoginFilter.java:47)

at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:235)

at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:206)

at org.apache.catalina.core.StandardWrapperValve.invoke(StandardWrapperValve.java:233)

at org.apache.catalina.core.StandardContextValve.invoke(StandardContextValve.java:191)

at org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:127)

at org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:103)

at org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:109)

at org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:293)

at org.apache.coyote.http11.Http11Processor.process(Http11Processor.java:861)

at org.apache.coyote.http11.Http11Protocol$Http11ConnectionHandler.process(Http11Protocol.java:606)

at org.apache.tomcat.util.net.JIoEndpoint$Worker.run(JIoEndpoint.java:489)

at java.lang.Thread.run(Thread.java:619)

# 16

# $("#form").serialize()的作用

有时候我们需要使用ajax提交去提交form的值，这样就需要使用serialize()去获取form的值

# 17

# [JSONObject put,accumulate,element的区别](http://blog.csdn.net/z69183787/article/details/18655143)

public Object put (Object key, Object value) 将value映射到key下。如果此JSONObject对象之前存在一个value在这个key下，当前的value会替换掉之前的value

public JSONObject accumulate (String key, Object value) 累积value到这个key下。这个方法同element()方法类似，特殊的是，如果当前已经存在一个value在这个key下那么一个JSONArray将会存储在这个key下来保存所有累积的value。如果已经存在一个JSONArray，那么当前的value就会添加到这个JSONArray中

。相比之下replace方法会替代先前的value

# 18

**当前action类必须继承json-default**

There is no result type defined for type 'json' mapped with name 'annualStatisticData'. Did you mean 'json'? - result - file:/F:/apache-tomcat-7.0.69/webapps/itcasttax/WEB-INF/classes/cn/itcast/nsfw/complain/conf/complain-struts.xml:22:54

# 19

no result defined for action …. and result ….

错误原因： 1.在跳转的jsp页面中，action的地址有写错；

2.在struts对应的action类中，跳转页面返回的地址有写错；

1. 在struts对应的返回页面，或者说是显示页面的jsp的名称看是否对应。

# 20

<!-- 1.注解驱动，2.提供对json的支持 -->

<mvc:annotation-driven></mvc:annotation-driven>

# 21

# [java.lang.NoSuchMethodError: org.springframework.util.ClassUtils.forName(Ljava](http://stackoverflow.com/questions/22465906/java-lang-nosuchmethoderror-org-springframework-util-classutils-fornameljava)

|  |  |
| --- | --- |
| 10 down vote | A NoSuchMethodError at runtime indicates that you are using a different version of a library than the version the code was built against.  In your case, Spring is the culprit. Check what is on the classpath at runtime and ensure the following:   * the version is the same as the compile time jar * if there is more than one version, remove the one not required   Looking at <http://docs.spring.io/spring/docs/3.2.0.M1/api/org/springframework/util/ClassUtils.html>, it would appear that ClassUtils.forName(String) is deprecated as of Spring 3. My guess would be you have built your code against a version which had this method but are running it with a version where the method has been removed.  The ClassUtils class is contained within the spring-core jar so I would ensure the same version of this jar is used at both compile and run time. |

**Spring的版本不匹配**

# 22

# [context:exclude-filter 与 context:include-filter 转](http://www.cnblogs.com/Struts-pring/p/4963621.html)

在xml配置了这个标签后，spring可以自动去扫描base-pack下面或者子包下面的java文件，如果扫描到有@Component @Controller@Service等这些注解的类，则把这些类注册为bean

注意：如果配置了<context:component-scan>那么<context:annotation-config/>标签就可以不用再xml中配置了，因为前者包含了后者。另外<context:annotation-config/>还提供了两个子标签

1.        <context:include-filter>

2.       <context:exclude-filter>

在说明这两个子标签前，先说一下<context:component-scan>有一个use-default-filters属性，改属性默认为true,这就意味着会扫描指定包下的全部的标有@Component的类，并注册成bean.也就是@Component的子注解@Service,@Reposity等。所以如果仅仅是在配置文件中这么写

<context:component-scan base-package=*"tv.huan.weisp.web"*/>

 Use-default-filter此时为true那么会对base-package包或者子包下的所有的进行java类进行扫描,并把匹配的java类注册成bean。

 可以发现这种扫描的粒度有点太大，如果你只想扫描指定包下面的Controller，该怎么办？此时子标签<context:incluce-filter>就起到了勇武之地。如下所示

<context:component-scan base-package="tv.huan.weisp.web .controller">

<context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"/>

</context:component-scan>

这样就会只扫描base-package指定下的有@Controller下的java类，并注册成bean

但是因为use-dafault-filter在上面并没有指定，默认就为true，所以当把上面的配置改成如下所示的时候，就会产生与你期望相悖的结果（注意base-package包值得变化）

<context:component-scan base-package="tv.huan.weisp.web ">

<context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"/>

</context:component-scan>

此时，spring不仅扫描了@Controller，还扫描了指定包所在的子包service包下注解@Service的java类

此时指定的include-filter没有起到作用，只要把use-default-filter设置成false就可以了。这样就可以避免在base-packeage配置多个包名这种不是很优雅的方法来解决这个问题了。

另外在我参与的项目中可以发现在base-package指定的包中有的子包是不含有注解了，所以不用扫描，此时可以指定<context:exclude-filter>来进行过滤，说明此包不需要被扫描。综合以上说明

Use-dafault-filters=”false”的情况下：<context:exclude-filter>指定的不扫描，<context:include-filter>指定的扫描

# 23

# [前缀、中缀、后缀表达式](http://blog.csdn.net/antineutrino/article/details/6763722) <http://blog.csdn.net/antineutrino/article/details/6763722>

# 24

# 在C语言中while（1）{} 编译能通过，在java中不行，C语言中的1可以当做boolean类型，而java中的1只能是数值型；

# 25

不出现断点的原因，可能是因为断点的上面存在一个死循环。

# 26

// System.out.print(4/10+4%10); 输出为4

// System.out.print(4/10+""+4%10); 输出为04

在java中，当判断你的第一个数字为单个时，会忽略前面的那个0，所以需要在数字中间加上一个空字符串。

# 27

Java针对大数有一个专门处理的类：BigInteger和 BigDecimal 这个理论上是无限位的，但会受到你的电脑位数内存的限制

# 28.

使用java的Scanner接收char类型的变量方法：

Scanner s=new Scanner(System.in);  
char b =s.next().charAt(0);  
System.out.println(b);

# 29

DecimalFormat 包含一个模式 和一组符号

　　符号含义：  
　　0 一个数字  
　　# 一个数字，不包括 0  
　　. 小数的分隔符的占位符  
　　, 分组分隔符的占位符  
　　; 分隔格式。  
　　- 缺省负数前缀。  
　　% 乘以 100 和作为百分比显示  
　　? 乘以 1000 和作为千进制货币符显示；用货币符号代替；如果双写，用  
　　国际货币符号代替。如果出现在一个模式中，用货币[十进制](http://baike.baidu.com/subview/359301/359301.htm)[分隔符](http://baike.baidu.com/subview/1268377/1268377.htm)代  
　　替十进制分隔符。  
　　X 前缀或后缀中使用的任何其它字符，用来引用前缀或后缀中的[特殊字符](http://baike.baidu.com/subview/1621/1621.htm" \t "_blank)。  
　　例子：  
　　DecimalFormat df1 = new DecimalFormat("0.0");  
　　DecimalFormat df2 = new DecimalFormat("#.#");  
　　DecimalFormat df3 = new DecimalFormat("000.000");  
　　DecimalFormat df4 = new DecimalFormat("###.###");  
　　System.out.println(df1.format(12.34));  
　　System.out.println(df2.format(12.34));  
　　System.out.println(df3.format(12.34));  
　　System.out.println(df4.format(12.34));  
　　结果：  
　　12.3  
　　12.3  
　　012.340  
　　12.34

# 30

scanner.next()   读取下一个字符串，遇到空格就不在读入

scanner.nextLine()读取下一行

# 31

Spring中的注解解释：

## @RequestMapping

RequestMapping是一个用来处理请求地址映射的注解，可用于类或方法上。用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。

RequestMapping注解有六个属性，下面我们把她分成三类进行说明。

**1、 value， method；**

value：     指定请求的实际地址，指定的地址可以是URI Template 模式（后面将会说明）；

method：  指定请求的method类型， GET、POST、PUT、DELETE等；

value的uri值为以下三类：

A） 可以指定为普通的具体值；

B)  可以指定为含有某变量的一类值(URI Template Patterns with Path Variables)；

C) 可以指定为含正则表达式的一类值( URI Template Patterns with Regular Expressions);

## @PathVariable的意思

## 当使用@RequestMapping URI template 样式映射时， 即 someUrl/{paramId}, 这时的paramId可通过 @Pathvariable注解绑定它传过来的值到方法的参数上。

1. @Controller
2. @RequestMapping("/owners/{ownerId}")
3. public class RelativePathUriTemplateController {
5. @RequestMapping("/pets/{petId}")
6. public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {
7. // implementation omitted
8. }
9. }

## 上面代码把URI template 中变量 ownerId的值和petId的值，绑定到方法的参数上。若方法参数名称和需要绑定的uri template中变量名称不一致，需要在@PathVariable("name")指定uri template中的名称。

## @ResponseBody

作用：

      该注解用于将Controller的方法返回的对象，通过适当的HttpMessageConverter转换为指定格式后，写入到Response对象的body数据区。

使用时机：

      返回的数据不是html标签的页面，而是其他某种格式的数据时（如json、xml等）使用；

接收页面请求商品id，调用service查询商品信息。直接返回一个json数据。需要使用@ResponseBody注解。

|  |
| --- |
| @Controller  **public** **class** ItemController {  @Autowired  **private** ItemService itemService;    @RequestMapping("/item/{itemId}")  @ResponseBody  **public** TbItem getItemById(@PathVariable Long itemId) {  TbItem tbItem = itemService.getItemById(itemId);  **return** tbItem;  }  } |
|  |

## @RequestParam

A） 常用来处理简单类型的绑定，**通过Request.getParameter() 获取的String可直接转换为简单类型的情况**（ String--> 简单类型的转换操作由ConversionService配置的转换器来完成）；因为使用request.getParameter()方式获取参数，所以可以处理**get 方式中queryString的值**，也可以处理**post方式中 body data的值**；

B）用来处理Content-Type: 为 application/x-www-form-urlencoded编码的内容，提交方式GET、POST；

C) 该注解有两个属性： value、required； value用来指定要传入值的id名称，required用来指示参数是否必须绑定；

# 32.maven项目中mybatis的映射文件找不到

org.apache.ibatis.binding.BindingException: Invalid bound statement (not found):

原因：在maven生成的目标文件中，没有把mapper目录下相应的.xml文件编译生成到目标文件下，所以项目报错找不到mybatis的配置。

解决方法：在mapper项目的pom.xml中添加如下内容

：

|  |
| --- |
| <!-- 如果不添加此节点mybatis的mapper.xml文件都会被漏掉。 -->  <build>  <resources>  <resource>  <directory>src/main/java</directory>  <includes>  <include>\*\*/\*.properties</include>  <include>\*\*/\*.xml</include>  </includes>  <filtering>false</filtering>  </resource>  </resources>  </build> |

## 扫描目录下的所有以.properties和.xml结尾的文件。

# 33

[maven 编译出错 Failed to execute goal org.apache.maven.plugins:maven-clean-plugin:2.5:clean](http://blog.csdn.net/mys328/article/details/54881927)

这类错误

出现这种错误，通常是由于您已启动了另一个tomcat 进程或者运行的javaw.exe进程，导致报错。

解决方法：

1. 鼠标点击 X 进行关闭运行失败的 Console页，（如果运行多次，程序的console都只会放在这里）

关闭后，应该可以看到，红色小正方形。点击，使tomcat进程或运行的javaw.exe进程停止。

再重新编译即可。

# 34

关于使用EasyUI的树控件，以列表的方式展示所有的菜单选项时，无法显示下拉的问题。

树控件读取URL。子节点的加载依赖于父节点的状态。当展开一个封闭的节点，如果节点没有加载子节点，它将会把**节点id**的值作为http请求参数并命名为'id'，通过URL发送到服务器上面检索子节点。

每一个树的菜单项都有一个节点对应的id编号，浏览器根据该菜单标签的id值通过http协议去查询该标签下的所有子标签。

* **注意：state：节点状态，'open' 或 'closed'，默认：'open'。如果为'closed'的时候，将不自动展开该节点。**
* **节点的状态只有open和closed两种状态**

# 35

## JSON.stringify(params)：将一个json对象转换为字符串的形式。

# 36

org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping#0': Invocation of init method failed; nested exception is java.lang.IllegalStateException: **Ambiguous mapping found**. Cannot map 'itemParamItemController' bean method

**此错误的原因是spring的controller层的映射有重复，即@RequestMapping（"”）中的请求地址重复了**

# 37

**JSP页面的编码格式错误的可能原因：**

**所跳转到jsp页面的java文件的默认编码要设置为utf-8,**

**(2)检查自己的jsp页面的编码方式是否是utf-8.**

# 38

Maven中运行tomcat插件常见的命令：

Clean tomcat7:run :首先清空编译文件，再在tomcat7上运行

Install –DskipTests: 在maven仓库中安装，并跳过测试

# 39

**\*.js文件即javascript是不能够跨域请求数据的，出于安全考虑。**

**跨域：域名不同时，即为跨域，端口号不同也属于跨域**

域名与端口号相同是为同一域对象。

Jsonp: 一个跨域的解决方案，js跨域请求数据不可以，但js跨域请求js脚本是可以的。

可以把数据封装为一个js语句，跨域请求js脚本，可以得到此脚本，得到脚本后会立即执行。可以把数据作为参数传递到方法中，就可以获得数据，从而解决跨域问题。

# 40

@JsonProperty("n")

**private** String name;

@JsonProperty("u")

**private** String url;

@JsonProperty("i")

**private** List<?> item;

当把该数据转换为json数据时，key默认情况下会使用成员的名字，加上@JsonProperty后，会使用里面的值。

# 41

org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'itemCatController': Injection of autowired dependencies failed; nested exception is org.springframework.beans.factory.BeanCreationException: Could not autowire field: private com.taotao.rest.service.ItemCatService com.taotao.rest.controller.ItemCatController.itemCatService; nested exception is org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type [com.taotao.rest.service.ItemCatService] found for dependency: expected at least 1 bean which qualifies as autowire candidate for this dependency. Dependency annotations:

此错误的原因是：自动注入失败，失败的原因可能有：

[1.@Autowired](mailto:1.@Autowired)写错，导致对象不能够注入成功；

2.在spring的配置文件中，配有自动扫描的包，自动扫描的包名不正确。

# 42

在数据库中建表时，若指定主键的生活才能方式时自增长的时候，为了能够获取到自增长的主键的值，方法由：

1. select last\_insert\_id(); //通过sql语句获取最后一个插入的记录的Id

# 43 next()与nextLine()

next（）一定要读取到有效字符后才可以结束输入，对输入有效字符之前遇到的空格键、Tab键或Enter键等结束符，next（）方法会自动将其去掉，只有在输入有效字符之后，next（）方法才将其后输入的空格键、Tab键或Enter键等视为分隔符或结束符。简单地说，next（）查找并返回来自此扫描器的下一个完整标记。完整标记的前后是与分隔模式匹配的输入信息，所以next方法不能得到带空格的字符串而nextLine（）方法的结束符只是Enter键，即nextLine（）方法返回的是Enter键之前的所有字符，它是可以得到带空格的字符串的。

# 44 maven中找不到类

在maven项目的测试包中的测试类，当使用@Test编译发生错误时，此时可能是该类并没有加入到maven的编译的环境中，可以尝试清空当前maven项目重新编译并启动，若还是不行，则注释掉@Test中的部分代码，当重新运行通过编译到maven中时，再将注释取消。

需要重新编译一下，**先修改一下，再保存。**

# 45 访问服务出现406错误

解决方法：

1. 查看是否存在jackson包；
2. 如果访问地址的后缀是html,是不能够响应Json格式的数据，应将后缀名改为其他，比如\*.action,\*.do

|  |
| --- |
| <servlet-mapping>  <servlet-name>taotao-portal</servlet-name>  <url-pattern>\*.action</url-pattern>  </servlet-mapping> |

# 46 访问web层时出现post乱码的问题

解决方案：在@REquestMapping中增加produces=MediaType.\*\*\*+”;charset=utf-8”

|  |
| --- |
| @RequestMapping(value="/httpclient/post",method=RequestMethod.***POST***,produces=MediaType.***TEXT\_PLAIN\_VALUE***+";charset=utf-8")  @ResponseBody  **public** String testPost(String username,String password){  // return TaotaoResult.ok();  **return** "username : "+username+"\tpassword = "+password;  } |

由于在上面的代码中，返回的数据格式是json的格式，所以需要用户请求地址的后缀不为html.

# 47

[Solr错误：sorry, no dataimport-handler defined!](http://blog.csdn.net/lich0000/article/details/8628904)

原因是：apache-tomcat-6.0.35\webapps\solr\WEB-INF\lib没有导入

solr-dataimporthandler-4.1.0.jar

solr-dataimporthandler-extras-4.1.0.jar

# 48

priorityQueue类的offer和add方法的不同

add(E e)和offer(E e)的语义相同，都是向优先队列中插入元素，只是Queue接口规定二者对插入失败时的处理不同，前者在插入失败时抛出异常，后则则会返回false。对于PriorityQueue这两个方法其实没什么差别。

# 49 list集合的问题

List集合中，当使用remove移除掉一个元素后，后面的元素会整体向前移动，并且list的大小会减1.

# 50. pluginmanagement 和plugins 区别

maven会在当前项目中加载plugins声明的插件；  
  
pluginManagement是表示插件声明，即你在项目中的pluginManagement下声明了插件，maven不会加载该插件，pluginManagement声明可以被继承。  
  
pluginManagement的一个使用案例是当有父子项目的时候，父项目中可以利用pluginManagement声明子项目中需要用到的插件，之后，当某个或者某几个子项目需要加载该插件的时候，就可以在子项目中plugins节点只配置 groupId 和 artifactId就可以完成插件的引用。  
pluginManagement主要是为了统一管理插件，确保所有子项目使用的插件版本保持一致，类似的还是dependencies和dependencyManagement。

# 51. eclipse中maven的指令

Build install 对应的goals为：install

clean tomcat7:run 先清空编译的文件，再在服务器上运行

clean tomcat7:run

install –DskipTests 安装，并跳过测试类