struts2 模型驱动要注意的一个问题

作者：shollin
原文链接：https://ld246.com/article/1425524580484
来源网站：链滴

先来说一下模型驱动，在struts2的默认拦截器栈struts-default包中有一个 ModelDriverInterceptor拦截器，如果有我们的action实现ModerDriver接口，即可实现模型驱动。action如下代码
<pre class="brush: java">public abstract class BaseAction<T> extends ActionSupport implements ModelDriven<T>,SessionAware,ServletRequestAware{

/** model的支持 */
protected T model;

public void setModel(T model) {
    this.model = model;
}

public BaseAction() {
    ParameterizedType genericSuperclass = (ParameterizedType) this.getClass().getGenericSuperclass();
    Class<T> clazz=(Class<T>) (genericSuperclass.getActualTypeArguments()[0]);
    try {
        model=clazz.newInstance();
        log.info("BaseAction model:"+model);
    } catch (Exception e) {
        //throw new RuntimeException(e);
    }

}

@Override
public T getModel() {
    return model;
}
//部分代码省略
}</pre>

从代码上看，每次都new了一个对象， 如果我们用 XXXService.update(model)的话，会将空值默认值也会更新进去，显示这不是我们想要的。
，可以采用这一种方式：先通过id查找到这个象，然后将model的非空属性赋值给这个对象，最后更新对象。

public String edit() throws Exception {
        log.info("修改用户："+model);
        User u=userService.getById(model.getId());
        if (StUtils.notNull(model.getPassword())) {
        &nbp;   model.setPassword(DigestUtils.md5Hex(model.getPassword()));
         }
        BeaUtils.copyNotNullProperties(model, u);
        useService.update(u);
//省略部分代码...
}
最后上一个BeanUtils工具类：
<pre class="brush: java">package core.utils;

import java.beans.PropertyDescriptor;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;

```
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.BeansException;
import org.springframework.beans.FatalBeanException;
import org.springframework.util.Assert;

/**
● 扩展spring的BeanUtils，增加拷贝属性排除null值的功能(注：String为null不考虑)
● 扩展Apache Commons BeanUtils, 提供一些反射方面缺失功能的封装.
●
● @author shollin
●

/
public class BeanUtils extends org.springframework.beans.BeanUtils {
/* logger. */
private static Logger logger = LoggerFactory.getLogger(BeanUtils.class);

 /** getter prefix length. */

public static final int LENGTH_GETTER_PREFIX = &amp;quot;
et&amp;quot;.length();

/** 保护的构造方法. */
protected BeanUtils() {
}

public static void copyNotNullProperties(Object source, Object target, String[] ignoreProperti
s) throws BeansException {
    copyNotNullProperties(source, target, null, ignoreProperties);
}

public static void copyNotNullProperties(Object source, Object target, Class&amp;lt;?&amp;gt; editable)
```

```
throws BeansException {
    copyNotNullProperties(source, target, editable, null);
}

public static void copyNotNullProperties(Object source, Object target) throws BeansException
{
    copyNotNullProperties(source, target, null, null);
}

private static void copyNotNullProperties(Object source, Object target, Class&lt;?&gt; editable
 String[] ignoreProperties) throws BeansException {

    Assert.notNull(source, &quot;Source must not be null&quot;);
    Assert.notNull(target, &quot;Target must not be null&quot;);

    Class&lt;?&gt; actualEditable = target.getClass();
    if (editable != null) {
        if (!editable.isInstance(target)) {
            throw new IllegalArgumentException(&quot;Target class [&quot; + target.getClass().g
tName() + &quot;] not assignable to Editable class [&quot; + editable.getName() + &quot;]&
uot;);
        }
        actualEditable = editable;
    }
    PropertyDescriptor[] targetPds = getPropertyDescriptors(actualEditable);
    List&lt;String&gt; ignoreList = (ignoreProperties != null) ? Arrays.asList(ignoreProperties) :
ull;

    for (PropertyDescriptor targetPd : targetPds) {
        if (targetPd.getWriteMethod() != null &amp;&amp; (ignoreProperties == null || (!ignoreL
st.contains(targetPd.getName())))) {
            PropertyDescriptor sourcePd = getPropertyDescriptor(source.getClass(), targetPd.get
ame());
            if (sourcePd != null &amp;&amp; sourcePd.getReadMethod() != null) {
                try {
                    Method readMethod = sourcePd.getReadMethod();
                    if (!Modifier.isPublic(readMethod.getDeclaringClass().getModifiers())) {
                        readMethod.setAccessible(true);
                    }
                    Object value = readMethod.invoke(source);
                    if (value != null || readMethod.getReturnType().getName().equals(&quot;java.lang
String&quot;)) {// 这里判断以下value是否为空，当然这里也能进行一些特殊要求的处理 例如绑定
格式转换等等，如果是String类型，则不需要验证是否为空
                        boolean isEmpty = false;
                        if (value instanceof Set) {
                            Set s = (Set) value;
                            if (s == null || s.isEmpty()) {
                                isEmpty = true;
                            }
                        } else if (value instanceof Map) {
                            Map m = (Map) value;
                            if (m == null || m.isEmpty()) {
                                isEmpty = true;
                            }
```

```
                    } else if (value instanceof List) {
                        List l = (List) value;
                        if (l == null || l.size() &lt; 1) {
                            isEmpty = true;
                        }
                    } else if (value instanceof Collection) {
                        Collection c = (Collection) value;
                        if (c == null || c.size() &lt; 1) {
                            isEmpty = true;
                        }
                    }
                    if (!isEmpty) {
                        Method writeMethod = targetPd.getWriteMethod();
                        if (!Modifier.isPublic(writeMethod.getDeclaringClass().getModifiers())) {
                            writeMethod.setAccessible(true);
                        }
                        writeMethod.invoke(target, value);
                    }
                }
            } catch (Throwable ex) {
                throw new FatalBeanException(&quot;Could not copy properties from source to t
rget&quot;, ex);
            }
        }
    }
}
/**
 * 循环向上转型,获取对象的DeclaredField.
 *
 * @param object
 *            对象实例
 * @param propertyName
 *            属性名
 * @return 返回对应的Field
 * @throws NoSuchFieldException
 *             如果没有该Field时抛出
 */
public static Field getDeclaredField(Object object, String propertyName)
        throws NoSuchFieldException {
    Assert.notNull(object);
    Assert.hasText(propertyName);

    return getDeclaredField(object.getClass(), propertyName);
}

/**
 * 循环向上转型,获取对象的DeclaredField.
 *
 * @param clazz
 *            类型
 * @param propertyName
 *            属性名
 * @return 返回对应的Field
```

```
     * @throws NoSuchFieldException
     *             如果没有该Field时抛出.
     */
    public static Field getDeclaredField(Class clazz, String propertyName)
            throws NoSuchFieldException {
        Assert.notNull(clazz);
        Assert.hasText(propertyName);

        for (Class superClass = clazz; superClass != Object.class; superClass = superClass
                .getSuperclass()) {
            try {
                return superClass.getDeclaredField(propertyName);
            } catch (NoSuchFieldException ex) {
                // Field不在当前类定义,继续向上转型
                logger.debug(ex.getMessage(), ex);
            }
        }

        throw new NoSuchFieldException(&amp;quot;No such field: &amp;quot; + clazz.getName()
                + '.' + propertyName);
    }

    /**
     * 暴力获取对象变量值,忽略private,protected修饰符的限制.
     *
     * @param object
     *             对象实例
     * @param propertyName
     *             属性名
     * @return 强制获得属性值
     * @throws NoSuchFieldException
     *             如果没有该Field时抛出.
     */
    public static Object forceGetProperty(Object object, String propertyName)
            throws NoSuchFieldException, IllegalAccessException {
        return getFieldValue(object, propertyName, true);
    }

    public static Object safeGetFieldValue(Object object, String fieldName) {
        return safeGetFieldValue(object, fieldName, true);
    }

    public static Object safeGetFieldValue(Object object, String fieldName,
            boolean targetAccessible) {
        try {
            return getFieldValue(object, fieldName, targetAccessible);
        } catch (NoSuchFieldException ex) {
            logger.warn(&amp;quot;&amp;quot;, ex);
        } catch (IllegalAccessException ex) {
            logger.warn(&amp;quot;&amp;quot;, ex);
        }

        return null;
    }
```

```
public static Object getFieldValue(Object object, String fieldName)
        throws NoSuchFieldException, IllegalAccessException {
    return getFieldValue(object, fieldName, false);
}

public static Object getFieldValue(Object object, String fieldName,
        boolean targetAccessible) throws NoSuchFieldException,
        IllegalAccessException {
    Assert.notNull(object);
    Assert.hasText(fieldName);

    Field field = getDeclaredField(object, fieldName);

    boolean accessible = field.isAccessible();
    field.setAccessible(targetAccessible);

    Object result = field.get(object);

    field.setAccessible(accessible);

    return result;
}

/**
 * 暴力设置对象变量值,忽略private,protected修饰符的限制.
 *
 * @param object
 *            对象实例
 * @param propertyName
 *            属性名
 * @param newValue
 *            赋予的属性值
 * @throws NoSuchFieldException
 *             如果没有该Field时抛出.
 */
public static void forceSetProperty(Object object, String propertyName,
        Object newValue) throws NoSuchFieldException,
        IllegalAccessException {
    setFieldValue(object, propertyName, newValue, true);
}

public static void safeSetFieldValue(Object object, String fieldName,
        Object newValue) {
    safeSetFieldValue(object, fieldName, newValue, true);
}

public static void safeSetFieldValue(Object object, String fieldName,
        Object newValue, boolean targetAccessible) {
    try {
        setFieldValue(object, fieldName, newValue, targetAccessible);
    } catch (NoSuchFieldException ex) {
        logger.warn(&amp;quot;&amp;quot;, ex);
    } catch (IllegalAccessException ex) {
```

```
            logger.warn(&amp;quot;&amp;quot;, ex);
        }
    }

    public static void setFieldValue(Object object, String propertyName,
            Object newValue, boolean targetAccessible)
            throws NoSuchFieldException, IllegalAccessException {
        Assert.notNull(object);
        Assert.hasText(propertyName);

        Field field = getDeclaredField(object, propertyName);

        boolean accessible = field.isAccessible();
        field.setAccessible(targetAccessible);

        field.set(object, newValue);

        field.setAccessible(accessible);
    }

    /**
     * 暴力调用对象函数,忽略private,protected修饰符的限制.
     *
     * @param object
     *            对象实例
     * @param methodName
     *            方法名
     * @param params
     *            方法参数
     * @return Object 方法调用返回的结果对象
     * @throws NoSuchMethodException
     *             如果没有该Method时抛出.
     */
    public static Object invokePrivateMethod(Object object, String methodName,
            Object... params) throws NoSuchMethodException,
            IllegalAccessException, InvocationTargetException {
        return invokeMethod(object, methodName, true, params);
    }

    public static Object safeInvokeMethod(Object object, Method method,
            Object... params) {
        try {
            return method.invoke(object, params);
        } catch (IllegalAccessException ex) {
            logger.warn(&amp;quot;&amp;quot;, ex);
        } catch (InvocationTargetException ex) {
            logger.warn(&amp;quot;&amp;quot;, ex);
        }

        return null;
    }

    public static Object safeInvokeMethod(Object object, String methodName,
            Object... params) {
```

```
    try {
        return invokeMethod(object, methodName, params);
    } catch (NoSuchMethodException ex) {
        logger.warn(&amp;quot;&amp;quot;, ex);
    } catch (IllegalAccessException ex) {
        logger.warn(&amp;quot;&amp;quot;, ex);
    } catch (InvocationTargetException ex) {
        logger.warn(&amp;quot;&amp;quot;, ex);
    }

    return null;
}

public static Object invokeMethod(Object object, String methodName,
        Object... params) throws NoSuchMethodException,
        IllegalAccessException, InvocationTargetException {
    return invokeMethod(object, methodName, false, params);
}

public static Object invokeMethod(Object object, String methodName,
        boolean targetAccessible, Object... params)
        throws NoSuchMethodException, IllegalAccessException,
        InvocationTargetException {
    Assert.notNull(object);
    Assert.hasText(methodName);

    Class[] types = new Class[params.length];

    for (int i = 0; i &lt; params.length; i++) {
        types[i] = params[i].getClass();
    }

    Class clazz = object.getClass();
    Method method = null;

    for (Class superClass = clazz; superClass != Object.class; superClass = superClass
            .getSuperclass()) {
        try {
            method = superClass.getDeclaredMethod(methodName, types);

            break;
        } catch (NoSuchMethodException ex) {
            // 方法不在当前类定义,继续向上转型
            logger.debug(ex.getMessage(), ex);
        }
    }

    if (method == null) {
        throw new NoSuchMethodException(&amp;quot;No Such Method : &amp;quot;
                + clazz.getSimpleName() + &amp;quot;.&amp;quot; + methodName
                + Arrays.asList(types));
    }

    boolean accessible = method.isAccessible();
```

```
    method.setAccessible(targetAccessible);

    Object result = method.invoke(object, params);

    method.setAccessible(accessible);

    return result;
}

/**
 * 按Field的类型取得Field列表.
 *
 * @param object
 *            对象实例
 * @param type
 *            类型
 * @return 属性对象列表
 */
public static List&lt;Field&gt; getFieldsByType(Object object, Class type) {
    List&lt;Field&gt; list = new ArrayList&lt;Field&gt;();
    Field[] fields = object.getClass().getDeclaredFields();

    for (Field field : fields) {
        if (field.getType().isAssignableFrom(type)) {
            list.add(field);
        }
    }

    return list;
}

/**
 * 按FieldName获得Field的类型.
 *
 * @param type
 *            类型
 * @param name
 *            属性名
 * @return 属性的类型
 * @throws NoSuchFieldException
 *             指定属性不存在时，抛出异常
 */
public static Class getPropertyType(Class type, String name)
        throws NoSuchFieldException {
    return getDeclaredField(type, name).getType();
}

/**
 * 获得field的getter函数名称.
 *
 * @param type
 *            类型
 * @param fieldName
 *            属性名
```

```
 * @return getter方法名
 * @throws NoSuchFieldException
 *             field不存在时抛出异常
 *
 * @todo: 使用reflectUtils里的方法更合适，这里的实现方式，必须先有field才能有method，逻辑有问题 实际上，即使没有field也可以单独有method。
 */
public static String getGetterName(Class type, String fieldName)
        throws NoSuchFieldException {
    Assert.notNull(type, &amp;quot;Type required&amp;quot;);
    Assert.hasText(fieldName, &amp;quot;FieldName required&amp;quot;);

    Class fieldType = getDeclaredField(type, fieldName).getType();

    if ((fieldType == boolean.class) || (fieldType == Boolean.class)) {
        return &amp;quot;is&amp;quot; + StrUtils.capitalize(fieldName);
    } else {
        return &amp;quot;get&amp;quot; + StrUtils.capitalize(fieldName);
    }
}

/**
 * 获得field的getter函数,如果找不到该方法,返回null.
 *
 * @param type
 *             类型
 * @param fieldName
 *             属性名
 * @return getter方法对象
 */
public static Method getGetterMethod(Class type, String fieldName) {
    try {
        return type.getMethod(getGetterName(type, fieldName));
    } catch (NoSuchMethodException ex) {
        logger.error(ex.getMessage(), ex);
    } catch (NoSuchFieldException ex) {
        logger.error(ex.getMessage(), ex);
    }

    return null;
}

public static String getFieldName(String methodName) {
    String fieldName = methodName.substring(LENGTH_GETTER_PREFIX);

    return fieldName.substring(0, 1).toLowerCase() + fieldName.substring(1);
}

}
</pre>

<p>&amp;nbsp;</p>
<p>&amp;nbsp;</p>
```