**从源代码角度看Struts2返回JSON数据的原理**

日期:2012-01-03来源: 杨胜寒分享至:

前面一篇文章其实只是介绍了如何在S[tr](http://www.linuxso.com/command/tr.html)uts2中返回JSON数据到客户端的具体范例而无关其原理，内容与标题不符惹来标题党嫌疑确实是笔者发文不够严谨，目前已修改标题，与内容匹配。本文将从struts2-json插件的源码角度出发，结合之前的应用范例来说明struts2-json插件返回JSON数据的原理。

用winrar打开struts2-json-plugin-xx.jar(笔者使用版本为2.1.8.1)，根目录下有一个struts-plugin.xml，这个文件想必大家都很了解，不做过多介绍了。打开该文件，内容非常简答，如下：

<?xml version="1.0" e[nc](http://www.linuxso.com/command/nc.html)oding="UTF-8" ?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<package name="json-default" [ex](http://www.linuxso.com/command/ex.html)tends="struts-default">

<re[su](http://www.linuxso.com/command/su.html)lt-types>

<result-type name="json" class="org.apache.struts2.json.JSONResult"/>

</result-types>

<interceptors>

<interceptor name="json" class="org.apache.struts2.json.JSONInterceptor"/>

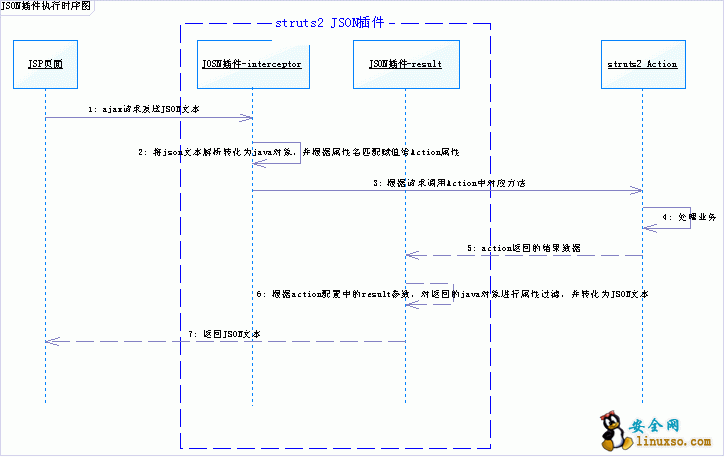
</interceptors>

</package>

</struts>

前文提到，如果要使用Struts2返回JSON数据到客户端，那么action所在的package必须继承自json-default包，原因就在上边的配置文件中:这里的配置文件指定了该插件的包名为json-default，所以要使用该插件的功能，就必须继承自该包——json-default。

上面的配置文件中，配置了两个类：org.apache.struts2.json.JSONResult和org.apache.struts2.json.JSONInterceptor，前者是结果类型，后者是一个拦截器。简单说一下，org.apache.struts2.json.JSONResult负责将action中的“某些”(通过相关参数可以指定，前文已有详述)或action中所有"可获取"(有getter方法的属性或一个有返回值的getter方法的返回值)数据序列化成JSON字符串，然后发送给客户端；org.apache.struts2.json.JSONInterceptor负责拦截客户端到json-default包下的所有请求，并检查客户端提交的数据是否是JSON类型，如果是则根据指定配置来反序列化JSON数据到action中的bean中(说的有点简单，其实该拦截器内部对数据做了很多判断)，拦截器不是本文的重点，介绍到此为止。看一张图，或许能够更加清晰明了的说明JSON插件执行的流程:



下面重点说说org.apache.struts2.json.JSONResult。

首先看一下org.apache.struts2.json.JSONResult源码的核心部分:

部分属性

private String defaultEncoding = "ISO-8859-1";//默认的编码

private List<Pattern> includeProperties;//被包含的属性的[正则](http://www.linuxso.com/book/11723.html)表达式，这些属性的值将被序列化为JSON字符串，传送到客户端

private List<Pattern> excludeProperties;//被排除的属性的[正则表达式](http://www.linuxso.com/book/11723.html)，这些属性的值在对象序列化时将被忽略

private String root;//根对象，即要被序列化的对象，如不指定，将序列化action中所有可被序列化的数据

private boolean wrapWithComments;//是否包装成注释

private boolean prefix;//前缀

private boolean [enable](http://www.linuxso.com/command/enable.html)GZIP = fa[ls](http://www.linuxso.com/command/ls.html)e;//是否[压缩](http://www.linuxso.com/linuxpeixun/12737.html)

private boolean ignoreHierarchy = true;//是否忽略层次关系，即是否序列化对象父类中的属性

private boolean ignoreInterfaces = true;//是否忽略接口

private boolean enumAsBean = false;//是否将枚举类型作为一个bean处理

private boolean excludeNullProperties = false;//是否排除空的属性，即是否不序列化空值属性

private int [stat](http://www.linuxso.com/command/stat.html)usCode;//HTTP状态码

private int errorCode;//HTTP错误码

private String contentType;//内容类型，通常为appli[cat](http://www.linuxso.com/command/cat.html)ion/json，在IE浏览器中会提示下载，可以通过参数配置<param name="contentType">text/html</param>，则不提示下载

private String wrapPrefix;//包装前缀

private String wrapSuffix;//包装后缀

看一下上一篇文章中的相关参数配置:

<package name="json" extends="json-default" namespace="/test">

<action name="testByAction"

class="cn.ysh.studio.struts2.json.demo.action.UserAction" method="testByAction">

<result type="json">

<!-- 这里指定将被Struts2序列化的属性，该属性在action中必须有对应的getter方法 -->

<!-- 默认将会序列所有有返回值的getter方法的值，而无论该方法是否有对应属性 -->

<param name="root">dataMap</param>

<!-- 指定是否序列化空的属性 -->

<param name="excludeNullProperties">true</param>

<!-- 这里指定将序列化dataMap中的那些属性 -->

<param name="includeProperties">

user.\*

</param>

<!-- 指定内容类型，默认为application/json，IE浏览器会提示下载 -->

<param name="contentType">text/html</param>

<!-- 这里指定将要从dataMap中排除那些属性，这些排除的属性将不被序列化，一半不与上边的参数配置同时出现 -->

<param name="excludeProperties">

SUCCESS

</param>

</result>

</action>

</package>

配置中出现了JSONResult的部分属性名，是的，JSONResult中的属性都可以根据需要在struts.xml中配置对应参数以改变默认值来满足我们的需要。

接下来看看它的两个核心方法:

public vo[id](http://www.linuxso.com/command/id.html) exe[cut](http://www.linuxso.com/command/cut.html)e(ActionInvocation invocation) throws Exception {

ActionContext actionContext = invocation.getInvocationContext();

HttpServletRequest request = (HttpServletRequest) actionContext.get(StrutsStatics.HTTP\_REQUEST);

HttpServletResponse response = (HttpServletResponse) actionContext.get(StrutsStatics.HTTP\_RESPONSE);

try {

String json;

Object rootObject;

//查找指定的需要序列化的对象，否则序列化整个action(上文包括前一篇文章中一提到过多次)

if (this.enableSMD) {

// generate SMD

rootObject = this.[write](http://www.linuxso.com/command/write.html)SMD(invocation);

} else {

// generate JSON

if (this.root != null) {

ValueStack stack = invocation.getStack();

rootObject = stack.[find](http://www.linuxso.com/command/find.html)Value(this.root);

} else {

rootObject = invocation.getAction();

}

}

//这是最核心的一行代码，包括了如何从rootObject抽取"可以"被序列化的属性的值，然后包装称JSON字符串并返回

json = JSONUtil.serialize(rootObject, excludeProperties, includeProperties, ignoreHierarchy,

enumAsBean, excludeNullProperties);

//针对JSONP的一个成员方法

json = a[dd](http://www.linuxso.com/command/dd.html)CallbackIfApplicable(request, json);

boolean writeG[zip](http://www.linuxso.com/command/zip.html) = enableGZIP && JSONUtil.isGzipInRequest(request);

//该方法是org.apache.struts2.json.JSONResult的一个成员方法，用于将JSON字符串根据指定参数包装后发送到客户端

writeToResponse(response, json, writeGzip);

} catch (IOException exception) {

LOG.error(exception.getMessage(), exception);

throw exception;

}

}

/\*\*

\* 负责根据相关参数配置，将制定JSON字符串发送到客户端

\* @param response

\* @param json

\* @param [gzip](http://www.linuxso.com/command/gzip.html)

\* @throws IOException

\*/

protected void writeToResponse(HttpServletResponse response, String json, boolean gzip)

throws IOException {

JSONUtil.writeJSONToResponse(new SerializationParams(response, getEncoding(), isWrapWithComments(),

json, false, gzip, noCache, statusCode, errorCode, prefix, contentType, wrapPrefix,

wrapSuffix));

}

恕笔者愚钝，找了好多资料，始终不明白这里的"SMD"是个什么意思，所在这里包括下文，都将忽略"SMD"。

可以看到，Struts2序列化对象为JSON字符串的整个过程都被JSONUtil的serialize方法包办了，所以有必要跟入这个方法一探究竟:

/\*\*

\* Serializes an object into JSON, excluding any properties matching any of

\* the regular [expr](http://www.linuxso.com/command/expr.html)essions in the given [col](http://www.linuxso.com/command/col.html)lection.

\*

\* @param object

\* to be serialized

\* @param excludeProperties

\* Patterns matching properties to exclude

\* @param ignoreHierarchy

\* whether to ignore properties defined on base classes of the

\* root object

\* @param enumAsBean

\* whether to serialized enums a Bean or name=value pair

\* @return JSON string

\* @throws JSONException

\*/

public static String serialize(Object object, Collection<Pattern> excludeProperties,

Collection<Pattern> includeProperties, boolean ignoreHierarchy, boolean enumAsBean,

boolean excludeNullProperties) throws JSONException {

JSONWriter writer = new JSONWriter();

writer.[set](http://www.linuxso.com/command/set.html)IgnoreHierarchy(ignoreHierarchy);

writer.setEnumAsBean(enumAsBean);

return writer.write(object, excludeProperties, includeProperties, excludeNullProperties);

}

该方法还有一个重载的兄弟方法，只是少了boolean enumAsBean这个参数，我们并不关心它，这里不讨论它。可以看到，这个方法更简单:构建一个JSONWriter实例，注入两个参数，然后调用该实例的write方法。我们进入JSONWriter，查看write方法的源码:

/\*\*

\* @param object

\* Object to be serialized into JSON

\* @return JSON string for object

\* @throws JSONException

\*/

public String write(Object object, Collection<Pattern> excludeProperties,

Collection<Pattern> includeProperties, boolean excludeNullProperties) throws JSONException {

this.excludeNullProperties = excludeNullProperties;

this.buf.setLength(0);

this.root = object;

this.exprStack = "";

this.buildExpr = ((excludeProperties != null) && !excludeProperties.isEmpty())

|| ((includeProperties != null) && !includeProperties.isEmpty());

this.excludeProperties = excludeProperties;

this.includeProperties = includeProperties;

this.value(object, null);

return this.buf.toString();

}

它同样有一个重载的方法，我们同样不关心，浏览整个方法，不难发现，它只是所做了一些赋值操作，然后将对象的序列化工作交给了value成员方法，那么我们进入value方法看一看:

/\*\*

\* Detect cyclic references

\*/

private void value(Object object, Method method) throws JSONException {

if (object == null) {

this.add("null");

return;

}

if (this.stack.contains(object)) {

Class clazz = object.getClass();

// cyclic reference

if (clazz.isPrimitive() || clazz.equals(String.class)) {

this.process(object, method);

} else {

if (LOG.isDebugEnabled()) {

LOG.debug("Cyclic reference detected on " + object);

}

this.add("null");

}

return;

}

this.process(object, method);

}

很简洁，进入process方法

/\*\*

\* Serialize object into json

\*/

private void process(Object object, Method method) throws JSONException {

this.stack.push(object);

if (object instanceof Class) {

this.string(object);

} else if (object instanceof Boolean) {

this.bool(((Boolean) object).booleanValue());

} else if (object instanceof Number) {

this.add(object);

} else if (object instanceof String) {

this.string(object);

} else if (object instanceof Character) {

this.string(object);

} else if (object instanceof Map) {

this.map((Map) object, method);

} else if (object.getClass().isArray()) {

this.array(object, method);

} else if (object instanceof Iterable) {

this.array(((Iterable) object).iterator(), method);

} else if (object instanceof Date) {

this.[date](http://www.linuxso.com/command/date.html)((Date) object, method);

} else if (object instanceof Calendar) {

this.date(((Calendar) object).getTime(), method);

} else if (object instanceof Locale) {

this.string(object);

} else if (object instanceof Enum) {

this.enumeration((Enum) object);

} else {

this.bean(object);

}

this.stack.pop();

}

发现它做了很多判断，并结合不同的方法来支持不同的数据类型，那么从这里我们可以知道Struts-json-plugin支持哪些数据类型了。对于每一种支持的数据类型，Struts-json-plugin都有相应的方法来从从对象中抽取数据并封装成JSON字符串，以Map为例，我们看一下map方法的源码:

/\*\*

\* Add map to buffer

\*/

private void map(Map map, Method method) throws JSONException {

//这是一个对象，按照JSON语法，应该以"{}"括起来

this.add("{");

Iterator it = map.entrySet().iterator();

boolean warnedNonString = false; // one report per map

boolean hasData = false;

while (it.hasNext()) {

Map.Entry entry = (Map.Entry) it.next();

//如果key不是String类型，将发出警告

Object key = entry.getKey();

//当前属性的OGNL表达式

String expr = null;

if (this.buildExpr) {

if (key == null) {

LOG.error("Cannot build expression for null key in " + this.exprStack);

continue;

} else {

//获取完整的OGNL表达式

expr = this.expandExpr(key.toString());

//是否是被排除的属性

//如果你对上边生成的OGNL表达式的格式有所了解，那么includeProperties和excludeProperties的正则配置绝对不是问题

if (this.shouldExcludeProperty(expr)) {

continue;

}

//如果不被排除，则将当前属性名压入表达式栈(其实就是一个String而非传统意义上的栈，此处是模拟，非常精巧的算法)

//该方法返回原来的表达式，稍后还将恢复该表达式到"栈"中

expr = this.setExprStack(expr);

}

}

//如果还有数据，则以","风格，这是JSON的语法格式

if (hasData) {

this.add(',');

}

hasData = true;

//如果key不是String类型，将发出警告，且只警告一次

if (!warnedNonString && !(key instanceof String)) {

LOG.warn("JavaScript doesn't support non-String keys, using toString() on "

+ key.getClass().getName());

warnedNonString = true;

}

this.value(key.toString(), method);

this.add(":");

//递归抽取数据

this.value(entry.getValue(), method);

//下一层的数据递归完成后，恢复表达式栈值为当前层的属性名

if (this.buildExpr) {

this.setExprStack(expr);

}

}

this.add("}");

}

这个方法中比较重要的几行代码都做了注释，不再赘述。过滤某些属性，以使其不被序列化时struts2-JSON应用中非常常见的，比如在序列化一个用户对象的时候，密码信息时不应该被传送到客户端的，所以要排除掉。了解shouldExcludeProperty方法的过滤规则，可以帮助我们更好的使用此功能。源码如下:

private boolean shouldExcludeProperty(String expr) {

if (this.excludeProperties != null) {

for (Pattern pattern : this.excludeProperties) {

if (pattern.matcher(expr).matches()) {

if (LOG.isDebugEnabled())

LOG.debug("Ignoring property because of exclude rule: " + expr);

return true;

}

}

}

if (this.includeProperties != null) {

for (Pattern pattern : this.includeProperties) {

if (pattern.matcher(expr).matches()) {

return false;

}

}

if (LOG.isDebugEnabled())

LOG.debug("Ignoring property because of include rule: " + expr);

return true;

}

return false;

}

非常简单，就是简单的正则匹配，如果有排除配置，则先判断当前属性是否被排除，如果没有被排除，且有包含配置则检查是否被包含，如果没有被包含，则不序列化该属性，如果没有被排除且没有包含配置，则将序列化该属性。

源码跟踪到这里，已经没有继续下去的必要了，因为我们已经很清楚Struts2是如何将一个对象转换成JSON字符串并返回客户端的:

**1、收集用户配置；  
2、JSONWriter通过判断对象的类型来有针对性的抽取其中的属性值，对于嵌套的对象则采用递归的方式来抽取，抽取的同时，包装成符合JSON语法规范的字符串；  
3、JSONUtil.writeJSONToResponse将序列化的JSON字符串按照相关配置发送到客户端；**  
  
不难看出，代码逻辑清晰，简单，朴素，没有半点花巧和卖弄，但确实是非常的精巧，表现出作者扎实的编程功底和过人的逻辑思维能力。尤其是递归抽取嵌套对象的属性值和获取当前属性的OGNL表达式的算法，堪称经典！

通过以上的源码跟踪，我们很清楚的了解Struts2序列化对象的原理和过程，并对相关参数的配置有了深刻的体会。有很多人说不知道includeProperties和excludeProperties的正则表达式该怎么配置，我想说其实很简单，除了正则知识外，就是"对象名.属性名"，数组稍微不同，以为它有下标，所以是"数组对象名\[\d+\]\.属性名"。如果这里觉得说的不清楚，可以阅读以下JSONWriter中关于OGNL表达式是如何获取的部分代码，就会明白正则该如何写了。

纯属个人理解，如有错误，烦请指正，不胜荣幸！

原创文章，转载请注明出处:http://yshjava.iteye.com/blog/1333602