### 1、Addon

Dorado Addon是指Dorado7的各种扩展包，通常是jar的形式提供的。事实上在Dorado7标准版提供的jar包中，除dorado-core，其他都可以视为Dorado Addon。Dorado Addon即可以提供新的控件、也可以提供新的皮肤、甚至是一整套完整的应用。

1）实现：

dorado-package.properties文件声明，需要在jar包的META-INF目录中添加一个dorado-package.properties文件，其内容（以dorado-chart-7.0.\*.jar为例）。

#Mon, 30 Jan 2012 05:50:47 -0500

#Addon的名称，建议为package路径

name=com.bstek.dorado.chart

#依赖的其他Dorado7的jar包的Addon的name，如有多个请用逗号隔开

depends=com.bstek.dorado.core

#该Addon的默认spring的配置文件

contextConfigLocations=classpath\:com/bstek/dorado/ofc/context.xml

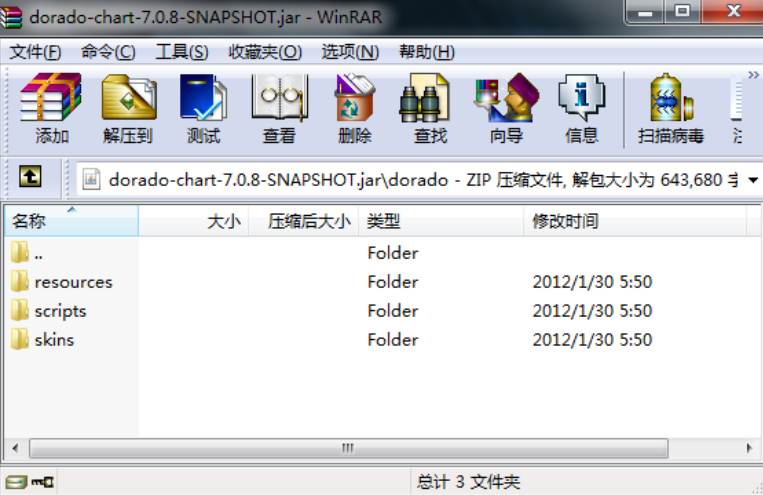
#Addon本身的版本说明

version=7.0.8-SNAPSHOT.120130.0550

2）client资源

如果该Addon附带还有Browser中使用的资源，如chart中含有swf，js，gif，jpg等等，则我们可以在jar的目录中添加一个dorado目录，并按照如下目录结构放置资源：

|  |  |
| --- | --- |
| 目录 | 说明 |
| resources | 其他资源 |
| scripts | 脚本资源 |
| skins | 皮肤资源 |



3）packages-config.xml

Dorado7中的js资源实现了按需加载机制，为了配合JS的按需加载，我们需要给当前Addon中相关的js资源命名，如：

<?xml version="1.0" encoding="UTF-8"?>

<config>

<Packages>

<Package name="chart" pattern="dorado-js" depends="widget">

scripts/dorado/chart

</Package>

</Packages>

</config>

chart中我们将资源命名为cahrt，这样当某一个视图希望使用这个Addon时，就可以在View的packages属性中添加chart，这样Dorado7 JS解析引擎就知道需要加载该Addon中scripts/doraod/chart对应的js文件。

4）view资源的读取

Add-on如果定义了View文件，在debug环境下，由于默认context.xml的文件加载方式为：

<bean parent=*"dorado.viewConfigFactoryRegister"*>

<property name=*"viewNamePattern"* value=*"\*\*"* />

<property name=*"viewConfigFactory"*>

<bean parent=*"dorado.xmlViewConfigDefinitionFactory"*>

<property name=*"pathPrefix"* value=*"${view.root}"* />

<property name=*"pathSubfix"* value=*".view.xml"* />

</bean>

</property>

</bean>

dorado默认从${view.root}中查找View资源文件，在debug运行环境下，通常都会对应一个物理路径，如：

view.root=classpath:

那么按照这种规则是无法找到add-on中的view资源的，辞职add-on中的资源实际是在Resource中，我们可以再添加一个dorado.viewConfigFactoryRegister的配置到context.xml。

<bean parent="dorado.viewConfigFactoryRegister">

<property name="viewNamePattern" value="d7.\*\*" />

<property name="viewConfigFactory">

<bean parent="dorado.xmlViewConfigDefinitionFactory">

<property name="pathPrefix" value="view.root " />

<property name="pathSubfix" value=".view.xml" />

</bean>

</property>

</bean>

### 2、Ajax

* 了解“可暴露服务”的基本概念及其背后的默认约定。
* 了解“智能方法适配”。
* 掌握在JavaScript中调用异步操作的基本方法。

### 3、Dorado7中的SpringMVC

Dorado7的Web服务是依托SpringMVC构建起来的。在web.xml的配置中可以找到一个com.bstek.dorado.web.servlet.DoradoServlet的配置，其继承自org.springframework.web.servlet.DispatcherServlet。因此可以直接借用因此而引入的SpringMVC特性。当然，使用方法与标准的SpringMVC略有区别。

1）配置url-mapping

在web.xml中默认配置只拦截了\*.d、\*.dpkg、/dorado/\*3中URL，为了在其中加入自己的web服务，增加一个自己的url-mapping。例如追加一个\*.do。

<servlet-mapping>

<servlet-name>doradoServlet</servlet-name>

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

</servlet-mapping>

2）配置servlet-context.xml

servlet-context.xmlshi Dorado7提供的一种Spring配置文件，专门用于配置与DoradoServlet相关的内容。这里给\*.do请求指派一个处理器。处理器通常是org.springframework.web.servlet.mvc.AbstractController的实现类。

<bean parent=*"dorado.resolverRegister"*>

<property name=*"url"* value=*"/login.do"* />

<property name=*"resolver"*>

<bean

class=*"xxx.xxx.LoginActionResolver"* />

</property>

</bean>

可以根据实际需要定义多个resolverRegister来处理各种请求。

### 4、客户端开发的改进建议

1）了解JavaScript中的逻辑判断

JavaScript是弱类型语言。所有类型的数值都可以用于逻辑判断和逻辑运算。

所以要清楚String、Number、Boolean、Object这些类型在用于逻辑判断时都会出现什么样的结果。

if(str) { //判断字符串是否有效

…

}

if(!str) { //进行逻辑运算

…

}

2）熟练运行Dorado对象的get和set

使用批量set，当需要对一个对象的多个属性进行连续的赋值时，应尽可能使用下面的语法：

foo.set{

readOnly: true,

value: “dorado”

}

使用迭代式的get和set：

get，set方法都支持迭代式的属性访问，通过“.”来分割一组属性名，交由此方法一层层向下挖掘并返回最终结果或设置目标属性。

当进行迭代式的读取时，系统会自动判断前一个属性返回的对象是dorado.AttributeSupport的实例还是普通的JSON对象，并借此决定如何进一步执行读取操作：

* 如果碰到的中间对象dorado.AtrributeSupport的实例，系统会自动读取它的Attribute。
* 如果碰到中间对象是普通的JSON对象，系统会直接读取它的属性。

oop.get("address.postCode");

// addressdorado.AttributeSupportoop.get("address").get("postCode")

// addressJSONoop.get("address").postCode

oop.set("address.postCode", 54733226);

//oop.get("address").set("postCode",54733226) //oop.get("address").postCode = 54733226

运用某些对象为get方法提供的特殊扩展：

通过扩展可以访问一些对象的Attribute之外的功能。

* View
  + 属性名以“#”开头时，会将其后的内容识别为控件的id。根据此id获取相应的控件。
  + 属性名以“^”开头时，会将其后的内容识别为对象的tag。根据此tag获取相应的对象。
  + 属性名以“@”开头时，会将其后的内容识别为DataType的name。根据此name获取相应的DataType。
* DataType
  + 属性名以“@”开头时，会将其后的内容识别为其中某个PropertyDef的name。根据此name获取相应的PropertyDef。
* DataSet
  + 属性以“data:”开头时，会将其后的内容识别为dataPath。利用该dataPath调用此DataSet的queryData()方法。

利用set为对象绑定事件：

// labelonClick

oop.set({

label : "Sample Text",

onClick : function(self, arg) {

... ...

}

});

3）学会使用控件的tag属性

通过设置tag属性以便批量的查找和操作一组控件或对象。在利用tag查找对象时Dorado会将所有匹配的对象封装成一个ObjectGroup并返回。通过ObjectGroup，可以对其中的所有对象进行批量的属性设置或方法调用。

view.set("^advanceButton.disabled", true);

4）尽可能使用异步的数据通讯

* 使用dataSet.flushAsync()而不是dataSet.flush()
* 不要将action的async属性设置为false
* 使用ajax.request()而不是ajax.requestSync()

尽管异步的数据通讯会带来一定的复杂度，但是出于改变用户界面整体的操作效率和提高界面友好度的角度出发。强烈建议使用异步的通讯机制。要使用异步的通讯机制必须掌握闭包（回调函数的基本用法）。

5）减少无谓的dataSet.flush()

在很多系统都见到过在页面初始化阶段执行一到多次的dataSet.flush()，其实这些操作理论上都是可以避免的。出现此类操作的主要原因是开发人员在页面初始化阶段通过JavaScript确定DataSet的parameter属性的值，然后再利用dataSet.flush()从服务端提取相应的数据。而这里所说的parameter在很多情况下是可以在服务端的页面准备阶段直接获得的。例如通过在DataSet的parameter属性中填写EL表达式的方式。

6）善用Argument和EL表达式简化代码

7）掌握each迭代

Dorado7中的很多对象都支持each迭代，each迭代可以令代码看起来更加简洁明了。目前支持each迭代的对象包括：

* Array（即标准的JavaScript数组，Dordado修改了JavaScript中Array的prototype）
* dorado.EntityList
* dorado.ObjectGroup
* dorado.util.KeyedArray
* dorado.util.KeyedList

var s = '';

['A', 'B', 'C'].each(function(item) {

s += item;

});

// s == "ABC"

8）不要总是有JavaScript注释

JavaScript是解释执行的语言，在JavaScript中，被注释掉的代码仍然会被输出到客户端。这意味这着这些注释仍然会产生网络开销。

在实际应用的场景中，我甚至见到过很多Client端事件中除了大段的代码注释外没有任何一行有效代码的情况。这意味着Dorado仍会为这些注释生成一个无用的事件监听器。所以，对于那些已经不再需要的代码，请果断删除而不是使用注释。

### 5、dorado-home配置文件参数说明

### 6、dorado对象监听器

1）普通的对象监听器

Dorado7中的很多对象都支持listener的属性，可以在此属性中定义一到多个的Listener用于监听对象的初始化过程。目前支持此类监听的对象包括：

* DataType
* DataProvider
* DataResolver
* ViewConfig
* Component（即所有的控件）

可监听的动作包括：

* beforeInit在对象将要被初始化之前触发，此时系统为尚未将XML中配置的信息初始化到对象中。
* onInit在对象被初始化之后触发，此时系统已将XML中配置的信息初始化到对象中。

定义一个监听器的方法与大多数自动方法适配相仿，同时Dorado还在此处提供了自动根据方法名确定监听动作的功能。具体的规则是如果您的方法名是以before开头的，那么Dorado会认为你希望监听beforeInit动作，否则您的方法监听到的将是onInit动作。

2）全局对象监听器

全局对象监听器可以批量的监听上面列出的所有对象或其中的一部分对象。

全局对象监听器不支持方法自动适配，要定义一个全局对象监听器必须继承Dorado7提供的com.bstek.dorado.data.listener.GenericObjectListener<T>类。该类包含3个属性：

* order：监听优先级，数字越小越先被触发
* pattern：要监听的范围，可使用通配符，不定义则表示全部拦截
* excludePattern：不监听的范围，可使用通配符。

还可以使用GenericObjectListener的泛型来设定监听范围。

public class ViewListener extends GenericObjectListener<View> {

@Override

public boolean beforeInit(View view) throws Exception {

// your code

}

@Override

public void onInit(View view) throws Exception {

// your code

}

}

向系统中注册一个全局对象监听器的方法：

<bean parent="dorado.genericObjectListenerRegister">

<property name="listener">

<bean class="test.ViewListener" >

<property name="order" value="1" /> <!-- -->

<property name="pattern" value="\*" /> <!-- View -->

<property name="excludePattern"

value="com.bstek.public.\*" /> <!-- View -->

</bean>

</property>

</bean>

### 7、Http请求的异常处理

Dorado中的请求大致分为：普通请求和Ajax类请求。

1）普通请求的异常处理

普通请求常指那些直接访问某个视图的请求，例如通过.d结尾的URI直接映射某个.view.xml的请求。除此之外也包括那些装载Dorado7的脚本库、CSS、以及通过/resources/\*装载各种资源文件的请求。

Dorado7通过SpringMVC中的SimpleMappingExceptionResolver来处理在这类请求中发生的异常，该类在Dorado缺省的上下文中是这样配置的：

<bean id="dorado.mappingExceptionResolver"

class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">

<property name="defaultErrorView" value="/dorado/ErrorPage" />

</bean>

如果要自定义此处的异常处理，可以在servelt-context.xml中添加一个同名的bean，覆盖缺省配置。

<bean id="dorado.mappingExceptionResolver"

class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">

<property name="exceptionMappings">

<map>

<entry key="xxx.xxx.SecurityException"

value="/access-denied.jsp" />

</map>

</property>

<property name="defaultErrorView" value="/dorado/ErrorPage" />

</bean>

2）Ajax类请求的异常处理

动态装载数据、提交数据、执行AjaxAction、执行远程数据校验等动作都会发起Ajax类请求。由于此类请求对于返回数据的格式有着严格要求，无论Server的逻辑成功与否都必须确保向Client返回特定格式的数据。因此通常不会使用ErrorPage机制。

要统一的处理Ajax类请求中的异常应利用Dorado7提供的全局拦截器机制。

当需要实现类似于普通请求中的出错重定向功能时应该怎么做呢？必须一旦业务系统抛出用户为登录或Session超时的异常时，即使在Ajax类请求中发生了这样的错误，开发者也仍然希望界面能够自动跳转到登录页面。为了解决这一问题，Dorado7提供了一种特别的异常类：

com.bstek.dorado.view.resolver.ClientRunnableException

该类的作用是向Client端返回一段可执行的JS脚本，当Dorado7的客户端接收到这段脚本后会自动的执行它。

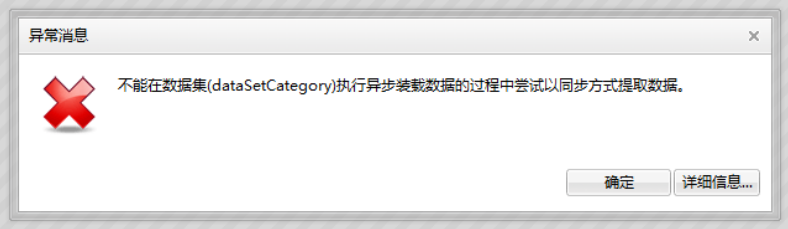
由于Dorado7中的Ajax请求支持请求自动合并的功能（一种自动将连续发出的Ajax请求合并为一次HTTP请求的优化功能），在这种合并请求中问题会变得比较复杂。因此，不建议你直接设置Response的content-type或者直接向Response的outputStream中进行输出。而是利用ClientRunnableException这类经过包装方法。

### 8、单表数据维护

这是一个对产品列表进行增删改页面，具备数据翻页的功能。

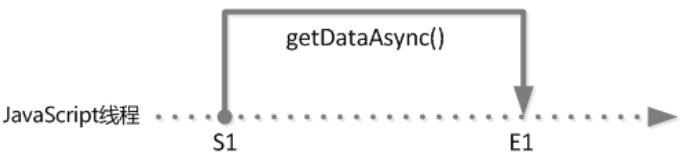
（如何配置基本的DataType，使用Annotation快速定义DataProvider和DataResolver，DataSet的数据绑定控件的基本使用。）

### 9、同步与异步装载冲突的分析和解决方法



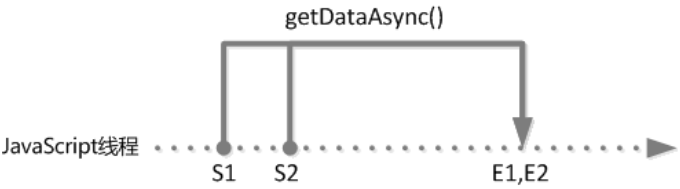
1）原因分析

Dorado7绝大部分与Server端的交互操作都是Ajax操作，即采用异步通信的操作。例如一旦我们调用DataSet的getDataAsync()时，如果此操作出发了数据装载那么Dorado将以异步的方式来执行这个装载操作。



由于采用了异步的请求，因此从Dorado发出请求S1到Server端返回结果E1的这段时间内，浏览器中JavaScript线程并不会被阻塞。

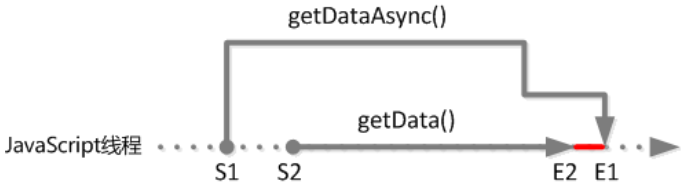
既然S1和E1之间的这段时间不阻塞，那就可能发生任何事情，比如用户又对同一个DataSet调用了一次getDataAsync()。当Dorado发现这种操作出现时并不会直接向Server端重复的发出另一次的数据装载请求，而会让第二次请求的回调方法直接等待前一次请求的返回结果。也就是说，对Dorado而言可以放心调的连续调用同一个DataSet的getDataAsync()方法，Dorado会确保这种操作正常执行且不会导致系统出现性能问题。



但是，如果用户在S1和E1之间调用了DataSet的getData()方法，即触发了同步的数据装载动作。由于浏览器中的JS是严格单线程的，并且不支持类似于sleep()、await()这样的阻塞方法。因此，没有办法让一个同步方法去等待另一个异步装载过程的返回结果！现在摆在Dorado面前的有两条路：

1. 不发送数据装载的请求，直接让getData()返回null；
2. 继续执行同步数据装载。

如果选择1，用户一定会认为Dorado出了BUG，因为DataSet可能真的有数据，不应该返回null。如果选择2，getData()将会得到正确的结果，但后面的事情就有可能变得更加复杂。



如果在S2处发出同步装载请求，那么在getData()执行的这段时间内（S2-E2），JS线程将处于完全阻塞的状态，即使在这段时间内，先前发出的getDataAsync()请求已经完成，由于JS线程被阻塞，它的回调处理也不得不等待，直到getData()的执行完全结束。假设用户的getData()如下：

var entity = dataSet.getData("#");

entity.set("price", 200);

dataSet.getData("#")将在E2时间点结束，不过由于JavaScript是单线程的，因此当entity.set("price",200)开始执行时JavaScript线程仍会处于阻塞状态。因此之前getDataAsync()的回调处理不可能在entity.set("price",200)之前完成。E1最终会在紧跟在E2和其后的阻塞代码之后被处理。这样问题就来了，用户已经在上面的代码端中对E2返回的数据做了修改，可是紧跟着，E1又带着新的返回结果完全覆盖掉了E2得到的数据，刚才的修改自然也被一并覆盖掉了。没什么比这更容易让用户误认为这是Dorado的BUG了，因为大多数用户很难理解这中间究竟发生了什么，这太隐晦了。根据上面的分析，我们既不能选择1也不能选择2，所以一旦碰到这种情况Dorado只能选择报错（至少在7.1.16之前一直都是这样的），也就是就是本文开头提到的那个错误信息。

解决方法：

* 改用异步的数据装载

dataSet.getDataAsync("#", function(entity) {

entity.set("price", 200);

});

* 利用onLoadData事件

既然DataSet已经开始执行异步的装载过程了，为什么不直接利用该装载过程的返回结果。所以，有时候考虑把逻辑代码移到DataSet或Reference的onLoadData事件上。

/\*\* @Bind #dsProduct.onLoadData \*/

!function(dsProduct) {

var entity = dsProduct.getData("#");

entity.set("price", 200);

}

* 禁用DataSet的自动装载

将DataSet的loadMode属性设置为manual。

* 提前使用同步装载

设法让getData()操作先于DataSet的自动装载就行了，比如可以将getData()操作移到该DataSet的onReady事件中。

* 将计就计（仅适用于7.1.16）