## 1、JavaScript基础

### 1）JSON

JSON相当于JS世界里的POJO（相当于JavaScript中由List和Map组成的POJO）

var d = {prop1:”value”,prop2:true}

复杂的例子：

{

name: "Phone",

propertyDefs: [{

name: "product",

label: "型号",

required: true

},

{

name: "manufacturer",

label: "生产商"

},

{

name: "type",

label: "机型",

mapValues: [{

key: "A",

value: "直板"

},

{

key: "B",

value: "翻盖"

},

{

key: "C",

value: "滑盖"

},

{

key: "D",

value: "旋盖"

},

{

key: "Z",

value: "其它"

}

]

},

{

name: "price",

label: "价格",

dataType: "float"

}

]

}

多层的JSON对象首先在顶层的两个属性（name和propertyDefs），propertyDefs是一个数组，内部又放了很多歌子对象。

根据前面的JSON的学习我们知道申明一个对象即可以用new Object，也可以直接用{}.上面就通过这种方式申明了多个对象，且每个子对象有不同的属性。其中第三个子对象又包含了一个数组。

上面的JSON的分析，我们可以看出这是一个复杂的JSON对象，是一个树状结构的JSON，通过JSON可以很方便的创建一颗树形结构的对象。在AJAX越来越流行的今天，JSON更容易被开发人员所接受，它比XML更为简洁，易读。

### 2）闭包

function a() { //定义一个函数a

var text = "Hello!" //声明了一个text变量

function b() { //又定义了一个函数b闭包函数

alert(text); //使用了text变量

}

setTimeout(b, 500); //延时500ms再执行函数b

}

对于Java程序员可以得出结论：500毫秒的延时，function的调用应该已经结束了，如果结束的话私有变量text变量就应该已经被释放，从而导致function b无法访问到变量text。但是在JavaScript的世界就允许你这样使用。并且对这种用法中的function b在JavaScript中就被称为闭包函数。在JS中闭包函数可以访问所处环境中的局部变量。并且浏览器中的GC对于变量的回收处理也会有所差别，当它判断到一个局部变量被某一个闭包函数所引用时，JS引擎就会阻止该变量被GC回收。直到这个变量没有被引用。

闭包函数不一定需要命名，也可以是匿名函数，如：

setTimeout(function() {

alert(text);

},500);

### 3）回调函数

回调函数最常见于Ajax操作和延时操作。通常都是利用闭包完成的。也就是说回调函数就是闭包的一种使用场景。在Dorado开发中会有较多的方法使用到回调函数，如：

$ajax.request(“data/phones.js”,function(result) {

grid.set(“items”,result.getJsonData());

});

$ajax是Dorado中ajax调用的一个基本工具类，我们调用其提供的request方法，方法中的参数是一个URL。当这个URL请求结束后，$ajax工具类就会自动触发内部的回调函数，由于是异步的，因此内部代码的执行是要等到ajax请求结束后才能进行。其中的grid等都是闭包机制中的局部变量，不用担心局部变量被释放的问题。

同步操作如下：

var result = $ajax.requestSync(“data/phones.js”);

grid.set(“items”,result.getJsonData());

采用同步的运算方法和异步运算方法在浏览器中的执行效果会有很大差别，例如一个耗时5秒的http请求,如果采用同步处理机制，在这5秒内用户是不能做任何事情的，整个浏览器会完全卡住，使用户无法进行包括页面上的按钮单击，查询，表单输入等动作，必须等到ajax请求结束后才能进行，而采用异步处理方式，浏览器就允许在ajax执行的同时，还接受用户在界面上做的各种操作。而在5秒之后，回调函数依然会被触发进行正常的数据逻辑处理。AJAX技术本身强调就是要尽可能的采用异步技术。尽可能的确保界面的友好性，和更有效率的界面操作。

## 2、虚拟面向对象

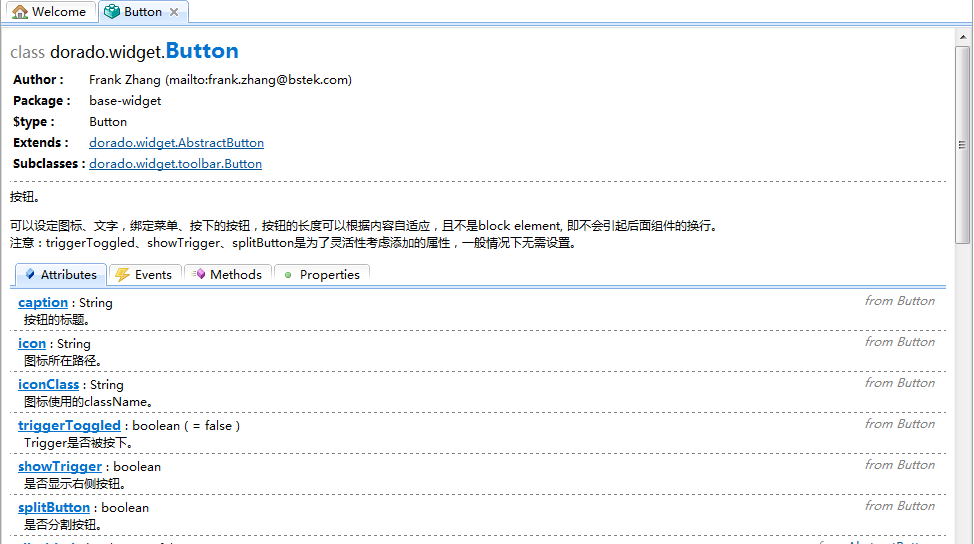
JavaScript本身并不支持面向对象，但是我们可以利用其Prototype去模拟部分面向对象的语言特征。虚拟面向对象就是指Dorado7中提供的JavaScript模拟面向对象的编程技术。

### 1）虚拟属性

Dorado的JS对象通常都包含很多的attributes，如我们访问URL:

<http://www.bsdn.org/projects/dorado7/deploy/jsdoc/>

访问dorado.widget下的Button控件:



这些attributes就是Dorado7对象的虚拟属性，因为它们与Java中的属性不同的地方在于，Java中的属性我们都可以通过专用的getXXX/setXXX方法进行读取操作，Dorado7对象的虚拟属性我们需要通过如下的方式：

var v = editor.get("value");

button.set("caption", "OK");

也就是说当我们看到JSDOC中某一个Dorado控件具有的某些attributes，我们就可以通过get/set方法，例如刚才我们看到的Button中有icon属性，则我们就可以通过如下的方式设置其icon属性：

button.set("icon", "images/button\_ok.gif");

下面介绍虚拟属性的一些其他特色:

### 2）set方法支持批量属性设置

我们对一个对象的多个属性进行赋值操作，虚拟属性的set方法支持属性批量设置，如下：

button.set({

icon: "images/ok.gif",

caption: "OK",

onClick: function() {

alert("You clicked OK.");

}

});

### 3）迭代式操作

JSON可以构造一个很复杂的树结构的对象，Dorado对象在很多情况下也是树形结构，例如有一个oop为一个Dorado虚拟对象，其中含address虚拟属性，而address本身又是一个Dorado虚拟属性，我们希望取出其中的postCode，如果我们要存取postCode的值，利用虚拟属性的功能，我们可以这么些代码:

oop.get("address.postCode");

oop.set("address.postCode", "7232-00124");

对于迭代式get和set方法，还提供了一个更有意义的功能，即加入在迭代过程中遇到的不是一个Dorado的虚拟对象，而是一个标准的JSON对象，则这种迭代操作的处理机制依然有效。

### 4）事件

Dorado为虚拟对象增加了很多事件，事件的添加方法：

方法一

button1.addListener("onClick", function(self) {

//此方法可以为一个事件添加多个监听函数

});

方法二

button1.set("onClick", function(self) {

// 此方法只能定义一个监听函数

});

上述代码的意图就是设置button1的onClick属性，由于不存在实际的onClick属性，且属性的值我们给了一个function,则Dorado事件引擎会自动调用button1.addListener处理机制，将这个function注册到事件管理器中。推荐使用第二种方法，代码更为直接。

### 5）self和arg

下面说明事件的几个公用特性，这些特性并不是每一个事件都有，但是这些特性是虚拟对象的事件的基本特性：

Dorado7中所有的事件都支持self和arg参数，其中self表示激发事件的自身，如Button激活onClick事件，则self表示Button自身，如Editor激活onClick事件，则self表示Editor对象自身，arg的类型与具体事件类型有关，很多情况下都是一个JSON对象，其中可能包含有非常丰富的传入信息。不同的事件中arg的JSON对象的属性是不一样的，事件A可能arg中含有prop1,prop2，我们可以通过如下代码获取其属性：

var v1 = arg.prop1;

var v2 = arg.prop2;

而事件B中可能传入的信息就是prop3,prop4,则代码就可能为：

var v3 = arg.prop3;

var v4 = arg.prop4;

不同事件的arg的差别，要通过JSDOC查看，例如我们看Button控件:

<http://bsdn.org/projects/dorado7/deploy/jsdoc/>

其中的onClick事件:



注意其中arg的button,event和returnValue的说明。

### 6）this

AJAX范例中，按钮Ajax Multiply的onClick事件代码中我们接触到this的写法，但是我们未细讲这个this的含义:

var action = this.get("#multiplyAction");

dorado.MessageBox.prompt("Please input two numbers here", {

defaultText: "3,5",

callback: function(text) {

var nums = text.split(",");

var parameter = {

num1: nums[0],

num2: nums[1]

};

action.set("parameter", parameter).execute(function(result) {

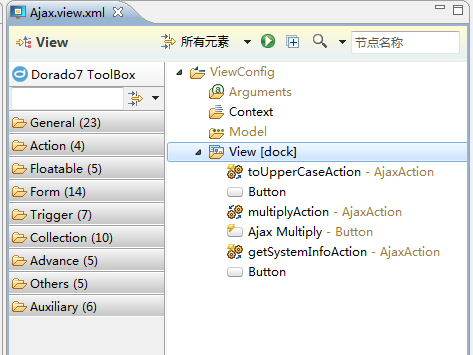
dorado.MessageBox.alert(nums[0] + " \* " + nums[1] + " = " + result);

});

}

});

其实这个this指的就是按钮所在的视图View：



几乎所有事件中的this均指向的是该控件隶属的View对象，View对象的onCreate事件中的this是指向其自身的。

onCreate事件中的this是个例外，因为控件总是先被创建然后才被添加到控件树上，因此在onCreate事件被触发时控件并不知道其隶属的View对象。

### 7）逻辑返回值

所有事件的返回值类型都是逻辑型，用于通知系统是否继续触发同一事件的后续监听器。不返回任何值则系统按true来处理。

有一种特殊情况需要关注这个逻辑返回值：

Dorado中对象某一个事件可以定义多个Listener的时候，这些Listener会按顺序依次触发，如果我们希望在第一个事件触发的过程中，屏蔽后面Listener的触发，则我们需要在第一个Listener中明确的返回一个false。这样后面的事件就会被跳过，不再执行。

### 8）processDefault

很多beforeXXX事件的arg参数中都支持一个名为processDefault的可写属性，用于通知系统是否要执行该事件所代表的后续操作。

如下，我们对一个DataType添加一个beforeRemove的事件，并在其中判断，如果是已婚的则不允许删除，通过MessageBox给出提示信息，并设置arg的processDefault为false，用以告诉beforeRemove动作如果是已婚则不执行默认的删除动作。

employeeDataType.addListener("beforeRemove",

function(self, arg) {

if (arg.entity.get("married")) {

dorado.MessageBox.alert("已婚的员工不能被删除！");

arg.processDefault = false;

}

});

抛出一个异常，同样可以达到类似的目的。

employeeDataType.addListener("beforeRemove",

function(self, arg) {

if (arg.entity.get("married")) {

throw new dorado.Exception("已婚的员工不能被删除！");

}

});

两种方法其实有区别，可能导致不同的结果！例如我们对10条员工记录，循环做删除动作，例如第三条为已婚员工，则如果是用processDefault = false的处理机制，最终系统的效果是删除第三条的时候给出用户警告，但是系统会接着删除第四，五条直到第十条记录，也就是说会继续循环，最终结果是除了第三条记录之外，其它记录全部被删除。但是如果采用throw new dorado.Exception()处理机制，则循环到第三条弹出异常，并终止循环，后面的记录不会不删除。它们之间的差别，用Java关键字表示，就是processDefault相当与循环体中的continue，而throw new Dorado.Exception()相当与break。

## 3、创建Dorado7对象

首先可以根据JSDOC提供的文档参考创建Dorado7对象，例如：

new dorado.widget.Button({

caption: “Say Hello”,

onClick: function(self,arg) {

dorado.MessageBox.alert(“Hello!”);

}

});

上述代码创建一个Button对象，并设置其caption属性和onClick事件。大部分Dorado JS对象的构造参数都是一个用于初始化新对象的JSON对象。如上例次构造器参数等于传递给set方法的批量写入模式的参数。

### 1）$type

在创建复杂的复合对象的时候，例如一个Panel，其中含有10个Button对象，创建Panel的同时把其中的Button一起创建好。Dorado对此提供的思路是，我们能不能通过之前了解过的定义一个复杂的属性结构的JSON对象达到创建一个复杂Dorado对象的目的。这个时候我们就需要用到Dorado对象创建中的$type特性。

{

$type: “Button”,

caption: “Say Hello”,

onClick: function(self,arg) {

dorado.MessageBox.alert(“Hello!”);

}

}

这段代码声明了一个JSON对象，它在声明体内部的第一行定义了一个$type属性，告诉Dorado对象创建引擎这是一个dorado.widget.Button对象。

$type与widget的对应关系我们可以通过JSDOC对象的声明处查找，如Button对象中dorado.widget.Button的$type为Button。也就是说如果我们想要JSON的方式创建Dorado对象，其$type的值我们可以在JSDOC中通过这种方式查找。上述JSON对象$type之后的代码就相当于创建了Button对象之后，再通过set的批量赋值操作。

另外当一个带创建的对象不需要传入构造参数时，甚至可以根据一个代表$type的字符串来创建相应的Dorado对象。也就是说“$type”关键字可以省略。如上的JSON我们可以定义为：

{

“Button”,

caption: “Say Hello”,

onClick: function(self,arg) {

dorado.MessageBox.alert(“Hello!”);

}

}

### 2）示例分析

定义一个Grid的一个JSON声明：

new dorado.widget.Grid({columns: [

“#”, //创建一个RowNumColumn对象，省略了关键字$type

{property: “product”}, //创建一个DataColumn对象

{ //创建一个ColumnGroup对象

$type: “Group”,

caption: ”Volumn”,

column: [ //定义column属性

{property: “length”,width: 50},

{property: “width”,width: 50},

{property: “height”,width: 50},

]

},

{ //创建一个DataColumn对象

property: “price”,

onRenderCell: function(self,arg) {

arg.cell.style.color = (arg.data < 100) ? “red” : ””;

}

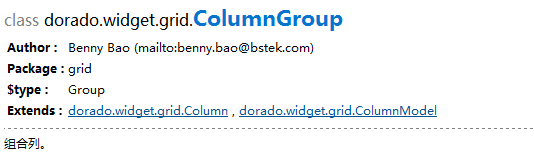
}

]

});

Grid中有很多列，在创建这个Grid的时候把很多列也都一起初始化好了。

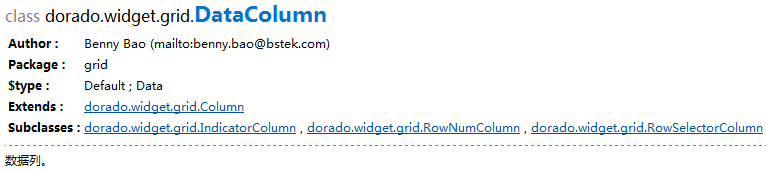
首先关注其中的$type:”Group”，在Grid中的列有好几种，其中一种为普通列，一种为组合列，组合列的JSDOC为：



其中$type为Group，则上面的JSON申明就表示创建一个dorado.widget.grid.ColumnGroup对象。对于像：

{property: “product”}

创建了一个dorado.widget.grid.DataColumn对象。



注意其中的$type的值为Default;Data，这表示有两个$type。如果我们在columns类型的对象内部定义如下的JSON：

{$type:”Data”,property;”product}或{property:”product”}

都表示创建了一个DataColumn对象。

ColumnGroup就需要我们显示的设置$type为”Group”，Group内部又包含一些DataColumn对象，这样我们又可以在内部定义columns属性：

columns: [

{ property: "length", width: 50 },

{ property: "width", width: 50 },

{ property: "height", width: 50 }

]

## 4、常用技巧和API

### 1）通过控件ID获得控件的方法

Dorado7中不支持以全局变量的形式访问控件，例如id为“buttonOK”的控件，JS端不允许直接通过buttonOK.set(“caption”,”OK”);这种形式的操作，由于Dorado7中采用了命名空间的做法，一个空间必须属于一个视图，而一个页面可能有多个视图，这样可以很好的解决JS全局变量命名冲突的的问题，在Dorado7中我们推荐使用如下方法获得控件：

view.get(“#buttonOK”);

其中上面的代码都是针对一个控件的某一时间执行的。如果get方法内部不加“#”，按照之前虚拟属性的了解，就会知道view.get(“buttonOK”)这是访问View对象的buttonOK属性，而添加了“#”则表示要获取其中的buttonOK控件。另外结合介绍的虚拟属性迭代的特性，可以这样使用：

view.get(“#buttonOK.caption”);

//通过虚拟属性的迭代，最终可以直接获得buttonOK的caption属性

Dorado7还支持一种获取控件的办法：

this.id(“buttonOK”); //这种获取控件的方法不支持迭代属性操作，不推荐

### 2）掌握对象的tags属性

Dorado允许开发人员为对象定义一到多个标签，然后利用标签批量的获得一组对象以便于对它们统一进行操作。

根据标签获得一批对象的2种方法：

view.get(“^foo”) //推荐

view.tag(“foo”)



如图编号为19的按钮，一共有三个标签：button表示按钮，odd表示奇数，green表示绿色（每个空间都可以为它定义不同的标签，其实就相当于关键字）接下来就可以利用Dorado7中的tags机制批量的查找按钮。

选择不同的按钮可以看到具有相同标签且符合条件的所有按钮都会处于按下的状态，在Dorado7中通过如下的代码实现tags的一批对象的选取：

view.get(“^foo”); //获取所有tags为foo的一批对象

这批对象在Dorado中以ObjectGroup形式封装，在Dorado7中ObjectGroup允许你对一批对象做一个统一的设定，如：

view.set(“^foo.diabled”,true);

//将所有标签为foo的控件对象的disabled属性设置为false。

找到所有的奇数按钮：

view.set(“^button”).set(“toggled”,false); //让所有按钮恢复常态

view.set(“^odd”).set(“toggled”,true); //找到所有的奇数按钮并使其按下

找到所有的偶数按钮：

view.set(“^button”).set(“toggled”,false); //让所有按钮恢复常态

view.set(“^odd”).set(“even”,true); //找到所有的偶数按钮并使其按下

找到所有红色按钮：

view.set(“^button”).set(“toggled”,false); //让所有按钮恢复常态

view.set(“^red”).set(“toggled”,true); //找到所有的红色按钮并使其按下

绿色按钮：

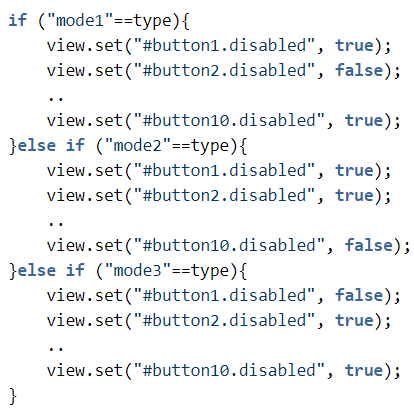
view.set(“^button”).set(“toggled”,false); //让所有按钮恢复常态

view.set(“^green”).set(“toggled”,true); //找到所有的绿色按钮并使其按下

还原所有按钮：

view.set(“^button”).set(“toggled”,false);

tags的操作在某些场景下能给我们带来很多好处，如果一个页面有十个按钮，三、四种不同的编辑模式下，要求这些按钮的disabled状态不同，通常情况下我们会采用如下非常繁琐的代码：



采用tags特性后，只需给每个Button设置不同的tags，设置其中为”model1”,”model2”,”model3”中的一到多种，多个tag逗号隔开，之后JS代码就可以非常简洁了：

if ("mode1"==type){

view.set("^model1.disabled", true);

}

else if ("mode2"==type){

view.set("^model2.disabled", true);

}

else if ("mode3"==type){

view.set("^model1.disabled", true);

}

### 3）View的get和set中更多使用技巧

综合对View的this方法的了解，一种有四种用法：

view.get(“title”) //读取View自身的虚拟属性

view.get(“#buttonOK”) //获取View中的某一个对象

view.get(“^foo”) //根据tags机制获取View中的一批对象

view.get(“@DataTypeProduct”) //根据名称获得某个DataType

### 4）$url()

定义一个URL的时候，如果定义一个字符串，其开头以“/”开始（包括Dorado开发当中也会接触到的，如button的icon也是一个url字符串），则表示从端口号开始计算，而不是以webapp的名字开头，而在很多场景下我们还需要webapp的名字，为了解决这个问题，很多系统都将这个webapp的名字作为JS变量输出到客户端，然后在客户端拼写最终的URL。在Dorado7中如果我们的URI以应用开头，则只要以“>”开头就可以了，对于这种URL，Dorado会自动帮助你转换为webapp开头的URI，如：“>images/icon.gif”将转换为“/appContextPath/images/icon.gif”

另外Dorado中还支持带有skin>前缀的写法，如：“skin>widget.css”将转换为Dorado当前皮肤路径下的widget.css的URL。因为skin是预设的URL前缀代表当前皮肤的根路径。

### 5）$DomUtils.xCreate()

当我们想动态创建一堆HTML元素，就需要不停地调用createElementById方法，并要设置好不同element之间的父子关系，如：

var button = document.createElement("INPUT");

button.context="Click Me";

button.style.border="1px black solid";

button.style.backgroundColor="white";

button.onclick=function(){

alert("Button clicked.");

}

或通过一段HTML脚本，赋值到一个具体的DOM对象的INNERHTML中：

var HTML= "<input type='button' style='border: 1px black solid;

backgroundColor:white' >Click Me</input>"

oo.innerHTML = HTML;

上面两种方式一是繁琐，二是采用HTNML形式拼写字符串不够形式化，不易维护。

Dorado推荐以JSON的方式利用xCreate方法创建DOM对象，根据以JSON形式定义的组件的模板信息快速的创建DOM元素：

$DomUtils.xCreate({

tagName: “button”, //定义类型

content: “Click Me”, //定义按钮标题

style: { //定义按钮的style

border: ”1px black solid”,

backgroundColor: “White”

},

onClick: function() { //定义onClick事件

alert(“Button clicked.”);

}

});

//例子1：

container.addChild(new dorado.widget.RadioGroup().set('radioButtons',

[

new dorado.widget.RadioButton({

text: '测试一',

value: '1'

}),

new dorado.widget.RadioButton({

text: '测试二',

value: '2'

})

]

);

//例子2：

container.addChild(new dorado.widget.RadioGroup({

'radioButtons': [

{

text: '测试一',

value: '1'

},

{

text: '测试二',

value: '2'

}

]

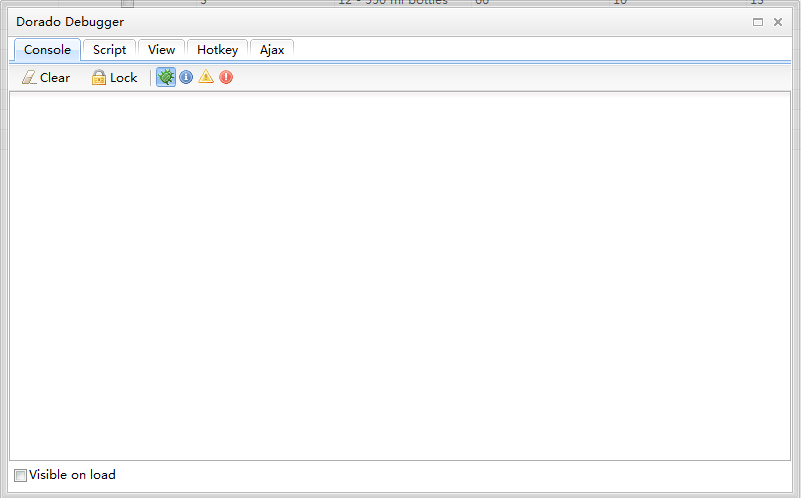
});

### 6）$log()

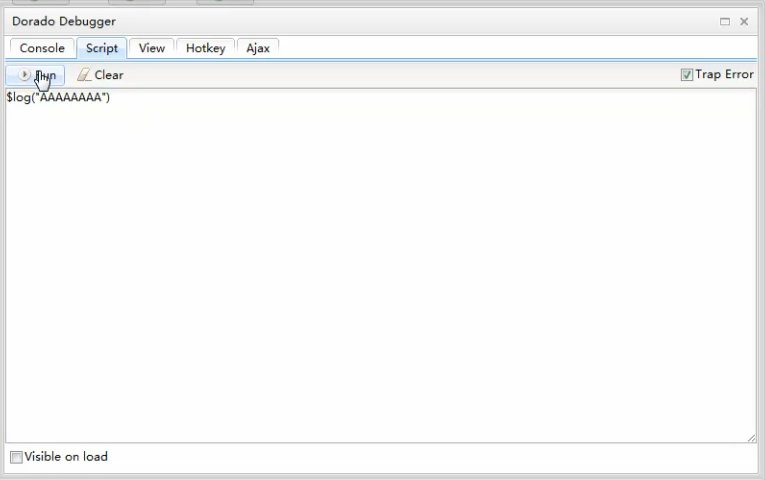
推荐使用$log()来代替alert()，alert()的缺点：

* 在循环体中，它会不停地弹出，让你确认，然后才能继续执行；
* 可能改变原来的逻辑，例如如果你的代码和焦点有关，alert会导致焦点的切换，让你无法完全模拟原有情况；

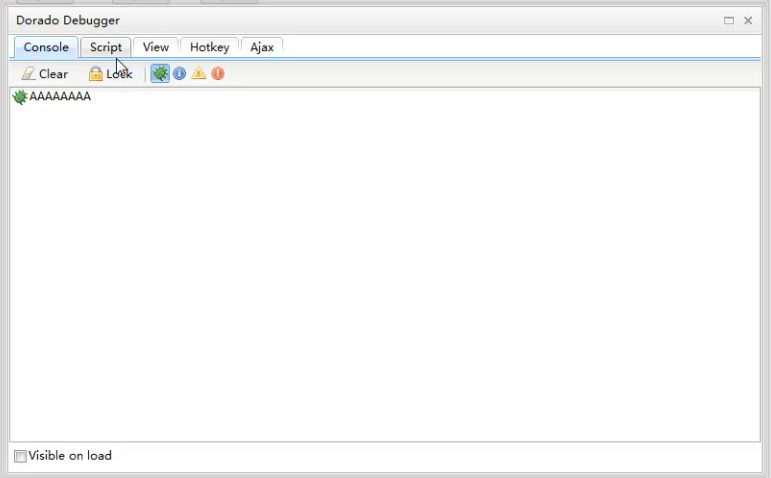
现在除了使用浏览器提供的debug技术之外，还有一种新选择$log()。这是记录调试日志的一个工具。日志结构可在页面上按F2后看到：



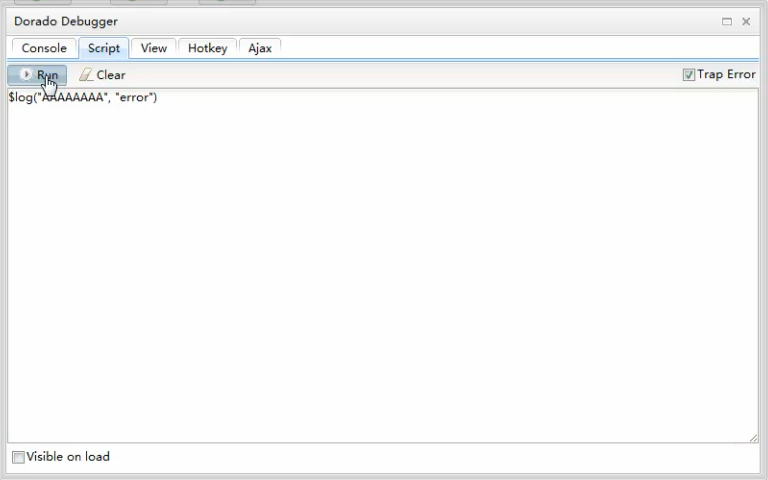
切换到script标签也行，并在编辑框中输入$log(“AAAAAAAA”);并单击Run按钮执行：



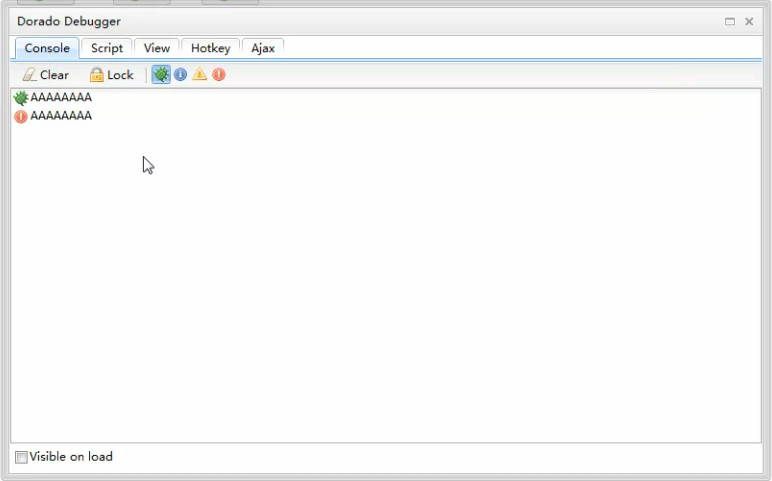
之后切换到Console标签页就可以看到输出的日志：



另外$log()功能还支持多种级别的日志，例如error，info，warn等，如我们输出一个error的信息：



运行后效果：



### 7）.each()

在JS循环中，除了按照Java类似的写法之外，each()是一种更值得推荐的方法。Dorado7扩展了标准的Array使之可支持each迭代，除Array之外Dorado7提供的EntityList、KeyedList等集合特性的对象也支持each迭代。

var s = ‘’;

[‘A’,’B’,’C’].each(function(item) {

s += item;

});

//最终结果为s == “ABC”

如上左边是一个数组，调用数组的each()方法进行循环，右侧是一个回调函数，所做的工作是在迭代循环中将字符拼装起来。另外each也支持循环的终止，只要在方法的内部写：

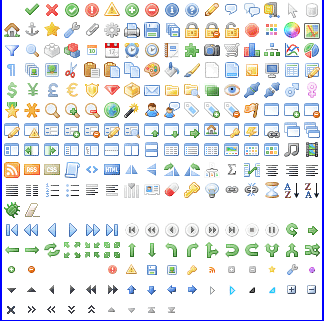
return false;

### 8）使用偏移量来定义图标

在定义按钮图标或其他控件图标的时候，采用图片偏移量的技术定义。将一些常用图标合并到一种图片中，然后采用图片偏移量的方式来定义图标。

url(skin>common/icons.gif) -140px -20px

在网页实际显示的时候就能出现一张图标。这是因为在Dorado中已经将一些常用的图标都合并到一个名称为icons.gif的图片中了，如下图：



当我们页面上的某一个控件对象希望显示其中的某一个按钮的时候，可以根据图标偏移量动态调整最后两个数字，参考如下的icontool，在这个工具中单击图标上图的位置，就可以看到图标的偏移量，以及剪切之后显示的内容。

在合并小图标的过程中要注意合并之后图片的大小，例如本例中图标合并后的gif的大小为27302字节，最好不要超过这个大小。