EXML基本语法(二)

数据绑定

EXMl支持数据绑定功能，数据绑定相当于是给静态的EXML文件添加动态脚本的功能，能够极大简化视图刷新的代码量。得益于JavaScript的动态语言特性，所有的Object对象都可以实现动态数据绑定，并不限定于Egret框架内的对象。在EXML中实现数据绑定只需要一对{}括号：

```
<e:ItemRenderer class="skins.ItemRendererSkin" xmlns:e="http://ns.egret.com/eui">
    <e:Label id="labelDisplay" text="{data.label}"/>
</e:ItemRenderer>
```

例如上面的例子，在Label节点的text属性上声明了一个数据绑定： `{data.label}` ，它表示text属性的值始终与根节点ItemRender的data属性上的label值相同。当ItemRender的data属性发生改变，或data上的label属性发生改变时，数据绑定都会自动通知Label重新对text属性赋值，从而刷新视图。

注意目前数据绑定只支持属性链访问，即 `a.b.c.d.x` 这种形式,还不支持直接声明复杂表达式的绑定。若遇到复杂表达式的情况，需要自行在外部完成转换，将最终计算结果赋值到数据对象上。

使用数据绑定特性请参考：数据容器 (../../../../extension/EUI/dataCollection/dataGroup/index.html)

内部类

在很多情况下，都需要引用一些只使用一次的EXML文件，对于这种情况，现在更好的做法是直接内嵌它，而不再需要额外创建一个文件再引用它。例如一个Button，需要引用一个皮肤，通常的做法是分成两个文件，创建一个ButtonSkin.exml文件用于描述按钮皮肤，在另一个EXML文件中使用skinName属性引用它的类名：

ButtonSkin.exml

```
<e:Skin class="skins.ButtonSkin" states="up,over,down,disabled" xmlns:e="http://ns.egret.com/eui">
    <e:Label id="labelDisplay" left="20" right="20" top="10" bottom="10"/>
</e:Skin>
```

MyGroup.exml

```
<e:Group class="skins.MyGroup" xmlns:e="http://ns.egret.com/eui">
    <e:Button label="按钮" skinName="skins.ButtonSkin">
    </e:Button>
</e:Group>
```

上面的例子，可以直接写在同一个文件内,等价于以下内容：

MyGroup.exml

```
<e:Group class="skins.MyGroup"  xmlns:e="http://ns.egret.com/eui">
    <e:Button label="按钮">
        <e:Skin states="up,over,down,disabled">
            <e:Label id="labelDisplay" left="20" right="20" top="10" bottom="10"/>
        </e:Skin>
    </e:Button>
</e:Group>
```

这种写法能够有效减少EXML的文件数量，仅当皮肤需要复用时，再创建独立的EXML文件。

除了皮肤，ItemRenderer也是一个典型的使用率很高且不复用的组件。可以直接内嵌ItemRender的皮肤到List节点中：

```
<e:Group class="skins.MyGroup" xmlns:e="http://ns.egret.com/eui">
    <e:List id="list">
        <e:itemRendererSkinName>
            <e:Skin>
                <e:Label id="labelDisplay" text="{data.label}"/>
            </e:Skin>
        </e:itemRendererSkinName>
    </e:List>
</e:Group>
```

视图状态

视图状态也算语法糖的一种。它跟数据绑定功能类似，也是给静态的EXML加入了动态刷新的功能。视图状态简单说就是预设一组状态刷新的操作，当切换到某个状态时，自动批量执行一系列的添加删除显示对象或改变对象属性的操作。下面是一个按钮皮肤的例子：

```
<e:Skin class="skins.ButtonSkin" states="up,over,down,disabled" xmlns:e="http://ns.egret.com/eui">
    <e:Image source="image/button_up.png" includeIn="up" width="100%" height="100%"/>
    <e:Image source="image/button_over.png" includeIn="over" width="100%" height="100%"/>
    <e:Image source="image/button_down.png" includeIn="down" width="100%" height="100%"/>
    <e:Image source="image/button_disabled.png" includeIn="disabled" width="100%" height="100%"/>
    <e:Label id="labelDisplay" textColor.down="0x009aff" left="20" right="20" top="10" bottom="10"/>
</e:Skin>
```

在根节点上，声明了视图状态名称列表 `states="up,over,down,disabled"` ,它表示这个皮肤具有 `up,over,down,disabled` 这四种状态，当皮肤的currentState属性被逻辑组件设置为这四个状态之一时，就会自动执行一些列的状态操作。这些状态操作通常有两类：1.添加移除对象，2.设置属性。

第一种状态操作，可以看到例子中有四个Image节点，每个节点上都有一个 `includeIn` 属性， `includeIn="over"` 表示这个Image节点只存在于over状态下，一旦，切换到其他状态时，自动移除这个节点，切换回over状态时又会重新添加这个节点并显示。这样我们只要每次设置视图状态的名称，几个图片就会自动显示或隐藏。操作对象的添加移除，关键字除了 `includeIn` 之外，还有一个 `excludeFrom` ,它表示不存在于某个状态。刚才的 `includeIn="over"` 等价于 `excludeFrom="up,down,disabled"` ,表示不存在与up，down，disabled几个状态中，即只存在于over状态中。

第二种状态操作是设置属性，可以看到Label节点上的这句： `textColor.down="0x009aff"` 它表示当状态切换到down时，设置Label的textColor属性为0x009aff。写法很简单，都是直接在属性名后加个点再加上状态名即可，表示在指定状态对此属性的赋值。

引用自定义组件

请参考：在EXML中使用自定义组件 (../../../../extension/EUI/advancedSkills/useComponents/index.html)