在EXML中使用自定义组件

引用自定义组件

之前例子中用到的节点都是EUI标准库中的组件，此外 EUI 可以自定义组件。例如有一个自定义的按钮类： `control.MyButton` ,在EXML中描述自定义组件的方式如下：

```
<e:Group class="skins.MyGroup" xmlns:e="http://ns.egret.com/eui" con="control.*"> <con:MyButton/> </e:Group>
```

首先我们要在根节点添加一个自定义的命名空间： `con="control.*"` ，等号之前的 `con` 表示命名空间前缀，这个可以随意写，只要不跟现有的前缀重名即可。等号后面的部分 `control.*` 表示在 `control` 这个模块名下的类。
声明了命名空间后，就可以合法地引用自定义组件： `<con:MyButton/>` 表示的类就是 `control.MyButton` 。
要是类不在任何模块下，可以直接声明命名空间为： `local="*"` ，同理前缀是可以随意起的，等号后面只需要一个 `*` 即表示不含模块名。

自定义组件规范

这里需要说明的是，虽然EXML中可以使用直接引用自定义组件，但作为最佳实践，我们推荐尽可能避免在EXML里直接使用它。因为 `在EXML中使用的自定义组件，对组件代码的健壮性有一定要求` ，才能被解析器正常实例化。并且绝大多数使用自定义组件的场景，应该都会有更好的组织方式而不是直接放自定义组件进去。有两种常见的情况是我们应该避免使用自定义组件：

1.在皮肤中放置一个不复用的自定义组件，并且为这个组件再设置一个皮肤，这种情况完全可以直接把它对应皮肤里的内容直接放到父级皮肤内，而不需要多这一层嵌套。

2.含有与业务逻辑耦合的组件。这类组件应该在逻辑代码中实例化动态添加，而不是放在皮肤中实例化。因为皮肤被实例化的时候，相关的业务逻辑依赖并没有初始化完全，容易产生报错。

当组件需要复用并且有很强通用性的时候。简单说就是这个自定义组件，起到的功能作用是跟框架里的UI组件库类似的，应该与具体的业务逻辑无关，能够独立被实例化使用。例如继承一个Button实现一个能播放影片剪辑功能的按钮。这种情况下我们才在EXML直接引用自定义组件。

当必须在EXML中使用自定义组件时，编码上具体要注意什么呢？为了更好的理解这个编码要求，这里简单讲解一下EXML运行时解析自定义组件的原理。当你在EXML中放置一个自定义组件时，EXML解析器在运行时需要去分析这个组件的属性列表以及对应的属性类型，用于类型检查以及格式化正确的数据类型。但由于JS语言的限制，原生并没有类的概念，要读取一个组件的具体含有哪些属性就必须先实例化它。所以开发者会遇到自定义组件构

造函数被多调用一次的情况，这是是正常的现象，一个组件只会实例化一次，读取属性列表后就会缓存下来。因此规则实际上只有这一条： `自定义组件要能单独实例化而不报错，且能正常访问到属性默认值不报错` 。具体可以拆解为以下应注意的点：

- 1.属性必须要有默认值（赋值为null也可以），因为TS编译器会把没有默认值的属性直接优化掉，在运行时并不存在。
- 2.属性的getter方法内要判断访问的对象是否为空，确保外部任何情况下访问属性都不会报错。
- 3.组件构造函数参数必须为空，或者参数有默认值，否则无法用空构造函数实例化。
- 4.组件的构造函数内不应该有对外部业务逻辑依赖的代码，这部分代码可以转移到组件被添加到舞台的时候启动而不是实例化时。

其中注意一下：虽然解析器可能无法实例化组件，但是你在EXML中其实并没有使用到它自身定义的属性，而是只用到了继承自框架组件的属性，这种情况其实应该可以忽略。因此为了最大限度确保显示正常，防止程序中断，解析器对这个报错进行了屏蔽，只会输出一些警告。例如当看到#2104号警告时，就是提示你自定义组件无法单独实例化。