# Toolkit5自定义控件

Toolkit5在Razor改版中，吸取了部分Java的经验，因此具备了自定义控件的功能。下面简述一下自定义控件的开发方法。

## IControlHtml

Toolkit5在新版的Razor中定义了IControlHtml接口，用来生成各个HTML控件。例如Text，Combo原先都是用枚举进行穷举，现在则是通过注册名找到对应的插件，然后调用该插件来生成HTML。

IControlHtml的定义如下：

public interface IControlHtml

{

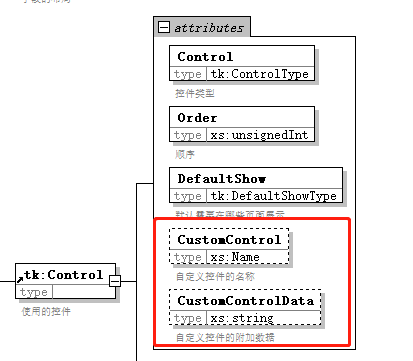
string GetHtml(Tk5FieldInfoEx field, IFieldValueProvider provider, bool needId);

}

有了这个接口，我们就可以进一步来构造自己的控件。

## DataXml的配置调整

为了配合自定义控件的开发，Toolkit5对DataXml的定义进行了微调，调整结构如下：



如图，tk:Control节点增加了CustomControl和CustomControlData两个自定义属性供自定义控件使用。

如果是自定义节点，那么Control属性需要配置**Custom**，在CustomControl写上对应IControlHtml的插件注册名。CustomControlData为自定义控件需要的扩展属性配置，可以不填，也可以填，对于多个属性的值，建议使用Json格式存储。

## 编写自定义控件示例

Toolkit5中没有一个现成的控件可以同时录入两个字段的数据，下面我们做一个简单的例子，来说明如何开发一个可以录入两个字段的控件。

由于DataXml中的定义中，Control是附着在一个字段下的，那么如果还需要另外一个字段的昵称。这里CustomControlData就发挥作用了，这里，我们采用Json的格式对另外一个字段的昵称进行设置。格式为{ “NickName”: “…”}。

示例代码如下：

[ControlHtml]

[InstancePlugIn]

internal class TestHelloControlHtml : IControlHtml

{

public static readonly IControlHtml Instance = new TestHelloControlHtml();

private const string HTML = "<span {0}><input {1}/><input {2} /></span> <label class=\"help-block\"></label>";

private static readonly WriteSettings Settings = new WriteSettings { Encoding = Encoding.Default };

internal class CustomData

{

[SimpleAttribute]

public string NickName { get; set; }

}

private TestHelloControlHtml()

{

}

#region IControlHtml 成员

public string GetHtml(Tk5FieldInfoEx field, IFieldValueProvider provider, bool needId)

{

if (string.IsNullOrEmpty(field.InternalControl.CustomControlData))

return string.Format(ObjectUtil.SysCulture,

"<!-- {0}没有配置CustomControlData，无法正确生成控件 -->", field.NickName);

CustomData custom = new CustomData();

custom.ReadJson(field.InternalControl.CustomControlData);

CustomData[] fields = new CustomData[2] { new CustomData { NickName = field.NickName }, custom };

HtmlAttributeBuilder div = new HtmlAttributeBuilder();

HtmlAttributeBuilder input1 = new HtmlAttributeBuilder();

HtmlAttributeBuilder input2 = new HtmlAttributeBuilder();

if (needId)

div.Add("id", field.NickName);

div.Add("name", field.NickName);

div.Add("data-getValue", "testHelloGetValue");

div.Add("data-setValue", "testHelloSetValue");

div.Add("data-fields", ToJsonString(fields));

input1.Add("data-name", field.NickName);

input1.Add("class", "form-control");

input1.Add("value", provider[field.NickName].ConvertToString());

input2.Add("data-name", custom.NickName);

input2.Add("class", "form-control");

input2.Add("value", provider[custom.NickName].ConvertToString());

return string.Format(ObjectUtil.SysCulture, HTML, div.CreateAttribute(),

input1.CreateAttribute(), input2.CreateAttribute());

}

private static string ToJsonString(CustomData[] fields)

{

return fields.WriteJson(Settings);

}

#endregion IControlHtml 成员

}

这里[ControlHtml]简写了插件注册的信息，这样插件的注册名为TestHello。如果实际编写，还是希望添加Author，CreateDate，Description等信息。

因为这个类无私有变量，因此它是无状态的。因此，我们可以采用实例插件。[InstancePlugIn]表示这是一个实例插件，也就是说，该类必须要有一个静态变量，名称为Instance或者INSTANCE。在类中，我们把构造函数声明为private，这就封闭了该类，也就是说插件工厂无法实例化该类，使用静态变量可以节约内存和创建时间。

在类中，我们定义了常量HTML，这是HTML的架子，两个input控件，由一个span包围。此外，如果需要显式后台校验的错误信息，需要在最后添加<label class=\"help-block\"></label>，如果有数据校验错误，会在这个label中显示。

在具体实现上，在span上我们需要追加一些附加信息。Toolkit的js为了支持自定义控件也有部分设置。在获取控件数据进行提交时，它需要经历三个步骤，首先通过getElement获得相应的DOM控件，然后通过getValue函数，从DOM控件中获取提交的数据，最后通过setValue函数把获取的数据写入到需要提交的Java Object中。上面三个步骤，都有默认处理过程，例如getElement通过$(“#NickName”)来获取DOM控件，getValue通过control.val()来取值，而setValue会根据传入的参数，默认为obj[NickName] = value。

如果自定义控件是单字段，上述三个函数一般都不需要重载。但是在我们现在的例子中，我们需要重载getValue和setValue函数。Toolkit的js为支持三个函数的重载，规定，在控件的html标签中分别写入data-getValue来重载getValue函数，data-setValue来重载setValue函数，data-getElement来重载getElement函数。在相应的attribute中写上重载的函数名即可。

所以，我这里额外写了一个js文件，来编写重载函数。代码如下：

function testHelloGetValue(elm) {

var fieldStr = elm.Control.attr("data-fields");

var fields = fieldStr.parseJSON();

var result = [];

$.each(fields, function (index, element) {

var filter = "input[data-name=" + element.NickName + "]";

var ctrl = elm.Control.find(filter);

var item = { "Name": element.NickName, "Value": ctrl.val() };

result.push(item);

});

return result;

}

function testHelloSetValue(data, field, value) {

$.each(value, function (index, element) {

data[element.Name] = element.Value;

});

}

结合这段js和上面的代码，我们就可以理解

div.Add("data-getValue", "testHelloGetValue");

div.Add("data-setValue", "testHelloSetValue");

这两行的含义了。

div.Add("data-fields", ToJsonString(fields));

上面的代码中，我们将两个需要提交的字段以Json的格式写入data-fields中。而对应的js中，会读出data-fields的信息，根据这个信息，找到对应DOM控件，取出相应的value，然后组合成JS Array返回。而setValue函数根据这个JS Array对要提交的JS Object进行赋值。

其他的代码应该都比较直观，这里就不解释了。

编写完后，记得在DataXml中写如下配置：

<tk:Control Control=”Custom” CustomControl=”TestHello” CustomControlData=”{&quot;NickName&quot;:&quot;AnotherNickName&quot;}” …/>

此外，在ModuleXml中还需要在PageMaker中挂接额外编写的js文件。

## 关于Toolkit5.5

Toolkit5.5现阶段在Razor上和Toolkit5只有语法上的区别，因此，上述功能在Toolkit5.5上可以完全适用。由于Toolkit5.5可以在linux上运行，编写的时候要注意大小写的问题。

不过Toolkit5.5今后可能采用Vue的方式来展示前端，如果那样，IControlHtml接口可能不再适用。此外，现有的自定义的方式只是一种补丁，不是最优的。如果DataXml进行重新定义，将可能会有更好的方法解决自定义控件。不过，暂时这些都是展望，还无实现。