# Razor升级需要注意的问题

Toolkit5.5依旧使用razor作为界面生成的引擎。但是和Toolkit5有重大不同，Toolkit5的razor移植于RazorEngine这个共享软件，而Toolkit5.5的razor移植于RazorLight这个共享软件（因为RazorEngine一直都测试不成功）。由于来源不同，所以，两个版本运行下，razor文件会有稍许不同。本文档尝试阐述这些不同点。

## 继承基类

Razor文件的第一行一般来说都是描述继承于哪个基类。新版的写法如下：

@inherits YJC.Toolkit.Razor.ToolkitTemplatePage<TModel>

大多数情况下，都是把YJC.Toolkit.Razor.ToolkitTemplatePage<TModel>作为基类。系统还提供了两个基类，如下：

@inherits YJC.Toolkit.Razor.ListTemplatePage<TModel>

@inherits YJC.Toolkit.Razor.MultiEditTemplatePage<TModel>

这两个基类，一个为列表页面服务，一个是为多表编辑服务。可以根据具体需要选择。但是大多数情况下，都是用第一个。

可以编写自己的基类，不过这样有点复杂，这里不描述。

## 单语句必须用{}

在Toolkit5中，我们可以写这样的razor代码

if (partTable != null)

{

foreach (DataRow row in partTable.Rows)

partSet.Add(row["FuncId"].Value<int>());

}

其中foreach语句中，因为子句只有一行代码，所以按照C#的语法，我们可以省去{}。但是在新版razor中，这样会导致编译错误。因此我们必须改成如下代码：

if (partTable != null)

{

foreach (DataRow row in partTable.Rows)

{

partSet.Add(row["FuncId"].Value<int>());

}

}

## 去除不必要的@{}

在旧版中，常常存在这样的代码：

@foreach (Tk5FieldInfoEx field in fields)

{

@{

string fieldString = RenderFieldItem(dataRow, field);

string value = provider[field.NickName].ToString();

}

if (fieldString != null)

{

@fieldString

}

这里@{}可以不必要存在。在新版中，如果使用这样的代码，会编译报错，必须改成如下代码：

@foreach (Tk5FieldInfoEx field in fields)

{

HtmlString fieldString = RenderFieldItem(provider, field);

string value = provider[field.NickName].ToString();

if (fieldString != null)

{

@Raw(fieldString.Value)

}

## 使用Raw函数

在新版Razor中，所有用@输出的文本会被自动转义。这样，带有HTML标记的内容会被自动转义，这样HTML就会作为文本显示，而不是HTML原生标记的方式显示。

为了解决这个问题，提供了Raw函数。使用Raw函数后，HTML标记不再转义。因此，原有的代码中，需要转义的代码可以删除，而没有转义的代码需要根据具体情况，增加Raw函数的调用

## 工具类变换

因为一些需要，原有的HtmlUtil和BootcssUtil分别变为HtmlCommonUtil和BootcssCommonUtil。如果引用这两个类，需要变换一下。原有函数应该不会改变。

## HtmlString引入

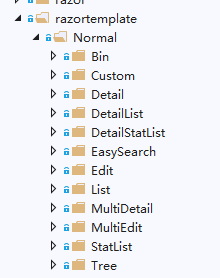
新版的RazorLight由于使用了Microsoft.AspNetCore.Html.HtmlString类，所以，很多函数以前返回string，现在都返回HtmlString了，如：RenderSection，RenderFieldItem。HtmlString和string一样，可以用@直接输出。只是，有些函数，比如Raw函数，需要的是string，而不是HtmlString，这时，我们可以用HtmlString的Value属性。比如代码：

@Raw(fieldString.Value)

## Razor模板库引入

在Toolkit5中，如果需要对界面有大的调整，往往都是直接修改razortemplate中对应的文件，这样，就会造成实际使用的和Toolkit标准存在差异。

现在，Toolkit5.5已经从架构到实现上完成了模板库的处理。以后，可以自行设计适合自己的模板库，而无须覆盖原有的Toolkit提供的标准模板库。



Normal目录是Toolkit5.5提供的标准的razor模板库。如果，对此模板不满意，可以在razortemplate下，建立一个新的目录，目录名随意。子目录结构务必和Normal下的子目录结构一致。

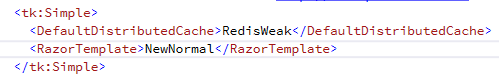
每个子目录下，凡是符合\*main.cshtml和\*template.cshtml命名的必须严格保留。其他的文件随意，根据自己的实际需要，可以多，也可以少。

关于Bin目录，css.cshtml和js.cshtml是被共享调用的，所以根据自己的需要保留或者放弃。TraceManager.cshtml、FactoryDetail.cshtml和ConfigDetail.cshtml是TracePage使用的，不需要重新构造。ErrorPage.cshtml是出现ErrorPageException后显示的界面，如果觉得显示风格不一致可以重写。Exception.cshtml是有例外后显示的错误提示，如果觉得显示风格不一致可以重写，通常情况下，这个是开发时看的，一般不需要重写。

目录构造完毕后，我们需要修改一个配置，让系统不使用Normal，而是使用新的目录

在新的Default.xml中，我们加入<RazorTemplate>新的目录名</RazorTemplate>就可以了。

如下图：



不写这个配置，或配置中写Normal都是调用Tk5.5提供的标准Razor模板库

## 调用其他文件

原先有RenderLayoutPartial（调用模板侧razortemplate目录下的文件）和RenderLocalPartial（调用项目侧razor目录下的文件）两个函数。

新版的设计了一个新的获取外部文件的架构，基类是TkRazorProject，不同的实现类，能获取不同位置的外部资源。这里，Toolkit实现了一个特别的类，来获取原先razortemplate和razor目录下的问题。

如果razor目录下的文件，可以直接写UserManager/LogOn.cshtml（目录分隔符尽量使用/而不是\，这样linux环境下可以跑，无需任何修改）。

如果是razortemplate目录下的文件，需要在开始前加^，表示这是个模板文件，例如^Normal/List/template.cshtml，如果想根据模板库自适应，那么用如下代码WebRazorUtil.GetTemplateFile("List/template.cshtml")来实现。

在某个razor文件中，还可以用@开始，表示相对路径。以当前文件的路径为基准路径，进行相对处理。

调用外部资源的函数叫做RenderPart，如：

@RenderPart("^Normal/Bin/js.cshtml")

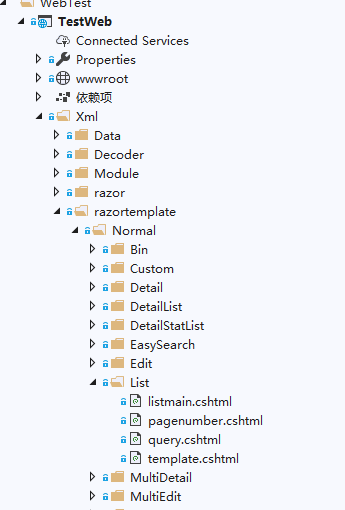
@RenderPart("@../Bin/js.cshtml")

对应的，另外一个函数RenderSectionIfDefined同样支持这样的文件格式，例如：

@RenderSectionIfDefined("Main", "@main.cshtml", Model)

## 编辑、调试Razor文件

.net core的web项目有如下变化，目录结构如下：



无论编辑还是调试，首先都需要把razor文件包含到项目中来。需要注意的是，在Windows下，只要包含进来就OK。但是在linux环境下，如果.cshtml的生成操作是内容，就会产生编译错误，必须设置成无，才能确保没有问题。如下图：



借助于VS2019的强大功能，编辑razor文件很容易了。不会再出现上一版本，不出现提示，正确的代码下面有波浪线提示这种情况了。当然，在编辑环境中，需要确认以下两个项目都在，这样应该确保没问题。

C:\Users\YJC\AppData\Local\Temp\1571382588(1).png

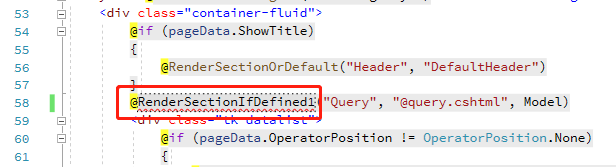
新版razor一个最大的优点就是可以对cshtml进行调试，可以设置断点，可以查看变量，如同调试C#代码一样。

调试方式很简单，和C#一样，在最左边点击，就能设置断点，运行到该处时，自己就会断，监测变量，单步跟踪等操作，如同C#代码一样，没有任何差别。只有一点遗憾，如果修改了cshtml的内容，必须重新编译运行，而原先可以直接刷新的。当然在非调试运行状态下，这个功能依旧存在。

## 关于编译报错

当razor模板编写时，如果有语法错误，那么这个由于编译不通过，导致无法运行。消除语法错误，首先必须准确的定位出哪里出错。为了方便找出语法错误，系统对相关的Exception进行了优化。

我们在List/template.cshtml中故意把一个函数的名字写错，如下图：



那么在展示List页面时，我们得到如下错误页面：



TemplateKey告诉你是哪个文件报错了，这里，按照Toolkit5.5的规定，显示^Normal\List/template.cshtml，那么就是razortemplate/Normal/List/template.cshtml错了，和我们改的文件一致。Error1（如果有多个错误，会继续显示Error2，Error3…），显示哪行那列什么错误，这里定位58行和我们的代码也一致。报错信息是英文的，这个看各人英文水平了。

## 一个比较邪门的问题

在移植的过程中，有一件比较邪门的事情，而且至今无法解释原因。在移植detail模板template.cshtml时（包括Detail和MultiDetail），都有这么一句代码：

IDetailModel model = CreateModel<IDetailModel>(Model);

提示似乎是说model是关键字，无法编译。把代码改成以下：

IDetailModel detailModel = CreateModel<IDetailModel>(Model);

修改后，编译就OK了。但是，在detail模板下的main.cshtml，还有其他模板下template.cshtml下的类似代码都无须修改。这个到底是什么原因，不清楚。