Рендеринг элементов управления

Создание нового элемента

Последнее обновление: 02.10.2016

За визуализацию элементов управления в Xamarin Forms на конкретных платформах отвечают специальные классы - рендереры (renderer). Например, для рендеринга элемета Button предназначен класс ButtonRenderer, для элемента Label - класс LabelRenderer и так далее. По соглашениям о наименовании класс рендерера называется по имени класса элемента плюс суффикс "Renderer".

Для каждой платформы есть своя реализация классов рендереров, которая опирается на нативные элементы управления, которые имеются на платформе. Так, элемент Label, доступный в Xamarin.Forms, по факту отсутствует на iOS, Android, UWP. Зато на iOS есть элемент со схожей функциональностью - UILabel. Также на Android есть похожий элемент - TextView. На UWP есть аналогичный элемент - TextBlock. И задача рендерера для элемента Label использовать нужный нативный элемент для рендеринга Label.

При создании своего рендерера у нас есть три различные стратегии:

- Создание с нуля нового класса для элемента управления и рендерера для него
- Наследования от уже имеющегося класса элемента
- Создание своего рендерера имеющегося элемента

Так, создадим новый элемент и для него рендерер. Для этого в начале создадим новый проект Xamarin Forms по типу Portable Class Library. Пусть проект называется **CustomRendererApp**.

Далее добавим в Portable-проект новый класс, который назовем **HeaderView**:

```
using Xamarin.Forms;

namespace CustomRendererApp
{
    public class HeaderView : View
    { }
}
```

Пусть данный элемент будет выполнять функцию заголовка.

Для создания своего элемента нам надо унаследовать свой класс от класса **View**. Больше пока нам ничего не потребуется.

После этого мы уже можем использовать данный класс на странице. Например:

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        Content = new HeaderView();
    }
}
```

Или в xaml:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:CustomRendererApp;assembly=CustomRendererApp"
             x:Class="CustomRendererApp.MainPage">
  <local:HeaderView></local:HeaderView>
</ContentPage>
```

Но если мы запустим приложение, то мы увидим пустую страницу, так как для элемента HeaderView еще не определен рендерер.

При запуске приложения Xamarin.Forms использует механизм рефлексии для поиска в загруженных библиотеках атрибута **ExportRenderer**. Данный атрибут указывает на наличие рендерера для элемента Xamarin.Forms.

При этом важно понимать, что мы не можем создать какой-то общий рендерер сразу для всех платформ. На каждой платформе есть свой набор визуальных элементов, свои механизмы для их создания, и наша задача - задействовать эти элементы для каждой из платформ при определении рендерера. Поэтому при создании рендерера могут потребоваться знания о том, какие элементы доступны на той или иной платформе, как они работают и т.д.

Итак, добавим в проект для Android новый класс, который назовем **HeaderViewRenderer**:

```
using Xamarin.Forms;
using Xamarin.Forms.Platform.Android;
using Android.Util;
using Android.Widget;
using CustomRendererApp;
using CustomRendererApp.Droid;

[assembly: ExportRenderer(typeof(HeaderView), typeof(HeaderViewRenderer))]
namespace CustomRendererApp.Droid
{
    public class HeaderViewRenderer : ViewRenderer<HeaderView, TextView>
    {
        protected override void OnElementChanged(ElementChangedEventArgs<HeaderView> args)
        {
            base.OnElementChanged(args);
            if (Control == null)
            {
                // создаем и настраиваем элемент
                TextView textView = new TextView(Context);
                textView.SetTextColor(Android.Graphics.Color.DarkGreen);
                textView.Text = "Android";
                textView.SetTextSize(ComplexUnitType.Dip, 28);

                // устанавливаем элемент для класса из Portable-проекта
                SetNativeControl(textView);
```

```
            }
        }
    }
}
```

Атрибут ExportRenderer принимает два параметра: тип элемента, для которого создается рендерер, и сам тип рендерера.

Сам класс рендерера наследуется от базового класса **ViewRenderer**, который типизируется двумя параметрами (`ViewRenderer<HeaderView, TextView>`). Первый параметр указывает на класс из Xamarin Forms, для которого создается рендерер (здесь это класс HeaderView). А второй параметр определяет нативный класс платформы, который будет использоваться для создания визуального элемента. В нашем случае таким классом является **TextView**, который в Android представляет простую метку с текстом.

Основная работа во ViewRenderer производится в методе **OnElementChanged()**. Этот метод вызывается при создании объекта HeaderView и создает нативный элемент управления, используемый для рендеринга HeaderView.

У класса ViewRender можно выделить два важнейших свойства, которые мы можем использовать:

- **Control**: указывает на создаваемый нативный элемент для данной платформы. В нашем случае TextView.
- **Element**: указывает на элемент из Xamarin.Forms, для которого создается нативный объект. В нашем случае HeaderView.

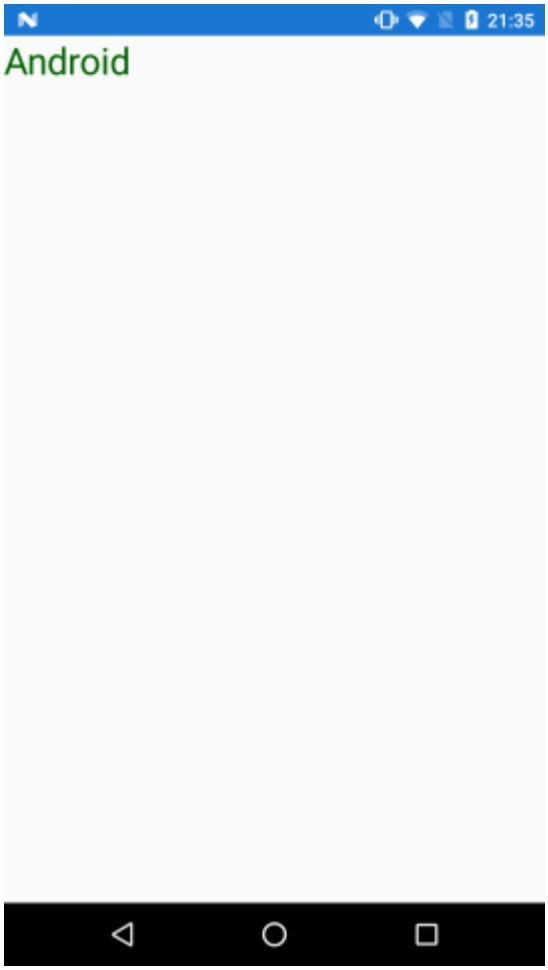
И в примере выше в методе `OnElementChanged()` вначале проверяем свойство `Control`, которое определено в базовом классе ViewRenderer. Это свойство должно представлять создаваемый элемент, то есть в данном случае TextView.

При первом вызове метода `OnElementChanged()` свойство Control имеет значение `null`. Поэтому нам надо создать объект TextView.

Для настройки TextView мы можем применять различные его свойства и методы. В данном случае устанавливается свойство `Text`, а также зеленый цвет текста и высота шрифта в 25 единиц.

После его создания объект передается в метод **SetNativeControl()**. После этого устанавливается свойство Control, которое в качестве значения приобретает созданный TextView.

Теперь если мы запустим приложение на Android, то увидим наш элемент HeaderView, за которым фактически будет скрываться нативный элемент TextView.



Далее добавим новый класс HeaderViewRenderer в проект для iOS:

```
using Xamarin.Forms;
using Xamarin.Forms.Platform.iOS;
using UIKit;
using CustomRendererApp;
using CustomRendererApp.iOS;

[assembly: ExportRenderer(typeof(HeaderView), typeof(HeaderViewRenderer))]
namespace CustomRendererApp.iOS
{
    public class HeaderViewRenderer : ViewRenderer<HeaderView, UILabel>
    {
        protected override void OnElementChanged(ElementChangedEventArgs<HeaderView> args)
        {
            base.OnElementChanged(args);
            if (Control == null)
            {
                UILabel uilabel = new UILabel
                {
                    Text = "iOS",
                    TextColor = UIColor.Red,
                    Font = UIFont.SystemFontOfSize(25)
                };
                SetNativeControl(uilabel);
            }
        }
    }
}
```

Здесь применяется похожая схема: также используется атрибут ExportRenderer, а класс рендерера наследуется от ViewRenderer. Только теперь для создания визуального элемента применяется

нативный класс **UILabel**, который имеется на iOS.

И также добавим новый класс HeaderViewRenderer в проект для UWP:

```
using Xamarin.Forms.Platform.UWP;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media;
using CustomRendererApp;
using CustomRendererApp.UWP;

[assembly: ExportRenderer(typeof(HeaderView), typeof(HeaderViewRenderer))]
namespace CustomRendererApp.UWP
{
    public class HeaderViewRenderer : ViewRenderer<HeaderView, TextBlock>
    {
        protected override void OnElementChanged(ElementChangedEventArgs<HeaderView> args)
        {
            base.OnElementChanged(args);
            if (Control == null)
            {
                TextBlock textBlock = new TextBlock
                {
                    Text = "Windows 10",
                    Foreground = new SolidColorBrush(Windows.UI.Colors.Blue),
                    FontSize = 28
                };
                SetNativeControl(textBlock);
            }
        }
    }
}
```

На UWP для создания элемента применяется класс TextBlock из пространства имен `Windows.UI.Xaml.Controls`.

И теперь наш элемент HeaderView будет работать на всех трех платформах.

Назад **Содержание** **Вперед**