在WPF中可以使用数据绑定来实现数据驱动UI的能力，因此我们使用HostModel类来实现数据到UI的映射，同时继承INotifyPropertyChanged来实现当数据变化UI自动更新的功能，最后将数据JSON化以后进行本地存储

1. Model层
2. 新建HostModel类，修改为public类，同时继承INotifyPropertyChanged，并且引用System.ComponentModel，然后定义PropertyChanged，并实现NotifyPropertyChanged函数

public event PropertyChangedEventHandler PropertyChanged;

#region 通知UI更新数据

protected void NotifyPropertyChanged(string propertyName)

{

PropertyChangedEventHandler handler = PropertyChanged;

if (handler != null)

{

handler(this, new PropertyChangedEventArgs(propertyName));

}

}

#endregion

1. 添加Index、Enable、CheckShow、CheckHide、IP、Port、IpAndPort、Url属性，同时实现get/set方法，以Index为例

private int \_index;

public int Index

{

get { return \_index; }

set

{

if (\_index != value)

{

\_index = value;

NotifyPropertyChanged("Index");

}

}

}

1. 实现构造函数

public HostModel(int index, bool enable, string ip, string port, string url)

{

\_index = index;

\_enable = enable;

\_ip = ip;

\_port = port;

\_url = url;

}

1. 数据绑定

找到GlobalStyle.xaml中的content\_host模板，给其中的IP和URL的Tag、Content属性设置数据绑定，同样也给选中框的Tag、Visibility属性设置数据绑定

<Label Grid.Column="0" Tag="{Binding Path=Index}" Content="{Binding Path=IpAndPort}" Template="{StaticResource content\_text}"></Label>

<Label Grid.Column="1" Tag="{Binding Path=Index}" Content="{Binding Path=TipsAndUrl}" Template="{StaticResource content\_text}"></Label>

<Rectangle Grid.Column="2" Tag="{Binding Path=Index}" Visibility="{Binding Path=CheckHide}" Style="{StaticResource content\_style\_rect}"></Rectangle>

<Canvas Grid.Column="2" Width="12" Height="12" Visibility="{Binding Path=CheckShow}">

<Path Fill="#FFEC8E72" Data="M10.125 1.5l-5.625 5.625-2.625-2.625-1.875 1.875 4.5 4.5 7.5-7.5z" />

</Canvas>

<Rectangle Grid.Column="2" Tag="{Binding Path=Index}" Visibility="{Binding Path=CheckShow}" Style="{StaticResource content\_style\_rect\_check}"></Rectangle>

1. 初始化
2. 在Main类中新增静态属性mainData，并在OnLoad方法的第一行添加初始化代码

//配置数据

public static ArrayList mainData;

//初始化配置数据

mainData = new ArrayList();

1. 新建Tools文件夹，在Tools文件夹中新建DataTool类，并实现静态方法addHostRule

public static void addHostRule(string ip, string port, string url)

{

//新建数据

HostModel rule = new HostModel(mainData.Count, true, ip, port, url);

//添加数据

mainData.Add(rule);

}

1. 在Container类中实现addHostRule方法

public void addHostRule(HostModel rule)

{

//创建UI对象

Label label = new Label();

//设置UI对象属性

label.Template = Resources["content\_host"] as ControlTemplate;

label.DataContext = rule;

//添加Rule

this.host.Children.Add(label);

}

1. 在Main类中的Onload添加以下测试代码

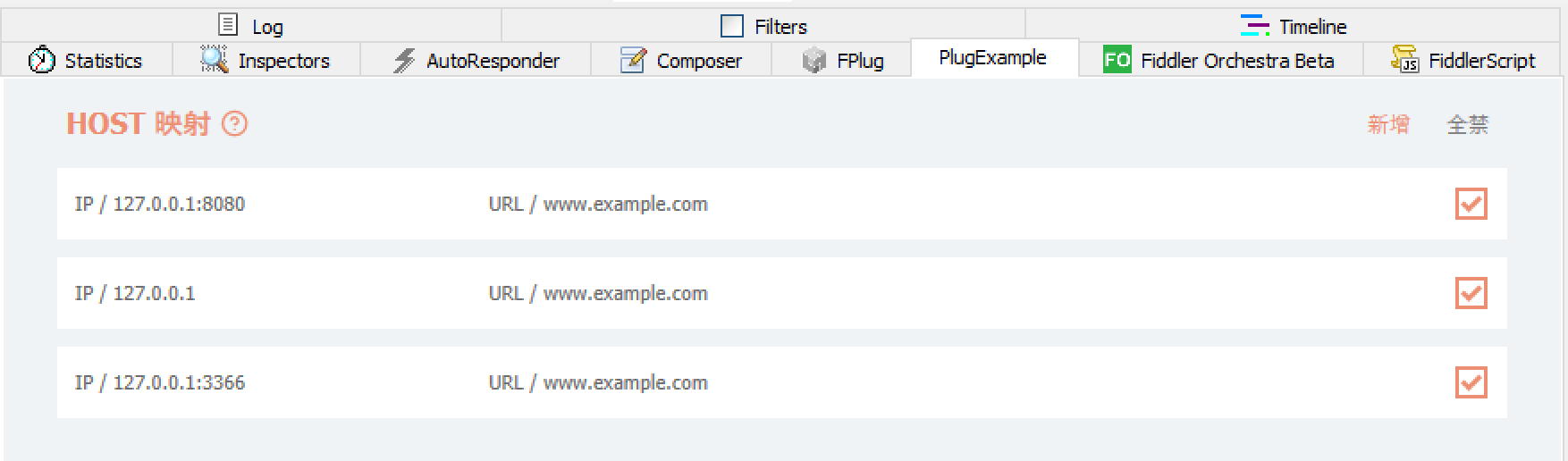
//测试代码

addHostRule("127.0.0.1", "8080", "www.example.com");

addHostRule("127.0.0.1", "", "www.example.com");

addHostRule("127.0.0.1", "3366", "www.example.com");

1. 结果预览



1. 本地存储
2. 新建Tools文件夹，然后在Tools文件夹中新建DataTool类
3. 本地存储需要写权限，因此我们选择文档文件夹，在DataTool类中定义静态属性path，并使用Environment获取文档文件夹的路径

//对应文件夹路径

private static string path = Environment.GetFolderPath(Environment.SpecialFolder.Personal) + "\\Fiddler2\\FiddlerExample";

1. 定义配置文件路径

//配置文件路径

private static string configPath = path + "\\config.json";

1. 使用NuGet给项目添加Newtonsoft.Json库，选择最新稳定版即可
2. 引入Newtonsoft.Json.Linq，然后定义私有静态方法formatConfigData

using Newtonsoft.Json.Linq;

#region 内部工具--配置转JSON

//格式化配置数据成JSON格式

private static JObject formatConfigData()

{

//最终结果

JObject result = new JObject();

//规则数据

JArray rules = new JArray();

//获取所有配置数据

ArrayList items = Main.mainData;

//遍历添加配置数据到rules

for (int i = 0, len = items.Count; i < len; i++)

{

//获取对应的Item

HostModel item = items[i] as HostModel;

//生成Json数据

JObject rule = new JObject();

rule.Add("enable", item.Enable);

rule.Add("ip", item.IP);

rule.Add("port", item.Port);

rule.Add("url", item.Url);

//填充进数据中

rules.Add(rule);

}

//添加骨架

result.Add("host", rules);

return result;

}

#endregion

1. 定义公有静态方法initFolder和writeConfigToFile

#region 暴露出去的方法

//初始化文件夹

public static void initFolder()

{

//备份文件夹不存在直接创建

if (!Directory.Exists(path))

{

Directory.CreateDirectory(path);

}

}

//将配置数据写到本地

public static void writeConfigToFile()

{

try

{

FileStream fs = new FileStream(configPath, FileMode.Create);

StreamWriter sw = new StreamWriter(fs);

//开始写入

sw.Write(formatConfigData().ToString());

//清空缓冲区

sw.Flush();

//关闭流

sw.Close();

fs.Close();

}

catch (Exception e)

{

FiddlerApplication.Log.LogString("FiddlerExample出现错误(writeConfigToFile函数)：" + e.ToString());

}

}

#endregion

1. 在Main类的OnLoad方法中添加测试代码

//本地存储测试代码

DataTool.initFolder();

DataTool.writeConfigToFile();

1. 进行编译打包后，将新添加的依赖库Newtonsoft.Json.dll也复制到Fiddler的Scripts文件夹中

