Software Engineering and Applications 软件工程与应用, 2018, 7(2), 84-90
Published Online April 2018 in Hans. http://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2018.72010

Hans汉斯

Design and Implementation of Graphical Network Monitoring System

Yonghong Xiao, Lijun Yu, Changtai Zhou

PLA Navy 92012 Troop, Zhoushan Zhejiang
Email: heumfw@163.com

Received: Apr. 6th, 2018; accepted: Apr. 17th, 2018; published: Apr. 24th, 2018

Abstract

Although there are already ready-made softwares for network on-off monitoring, there are certain deficiencies, inconvenient operation, and the display is not intuitive. The summary does not meet the actual needs. Based on this situation, it is of great significance to develop a set of network continuity monitoring software that can take its long and shortcomings. Based on the existing practical work, this article introduces the design method and implementation idea of network on-off monitoring software using WPF technology and MVVM model. The software consists of four modules: table and graphic interface, asynchronous test, documents' management, log's management. Compared to the original system, the software has been greatly improved on interface, operability and practicability.

Keywords

Network Monitoring, WPF, MVVM

图形化网络监测系统的设计与实现

肖永红，俞立军，周昶太

海军92012部队，浙江 舟山
Email: heumfw@163.com

收稿日期：2018年4月6日；录用日期：2018年4月17日；发布日期：2018年4月24日

摘 要

网络通断监测方面虽然已经有现成的软件，但存在一定不足，如操作不便，显示不直观，汇总不符合实际需要。基于这样的情况，开发一套能够取其长而舍其短的网络通断监测软件还是很有意义的。本文在

文章引用: 肖永红, 俞立军, 周昶太. 图形化网络监测系统的设计与实现[J]. 软件工程与应用, 2018, 7(2): 84-90.
DOI: 10.12677/sea.2018.72010

现有实际工作基础上，利用WPF技术和MVVM模式来设计实现一个网络通断监测软件。软件主要包含四个模块：表格和图形界面、异步通断测试、文档资料查阅管理、日志查询管理。相比于原有监测系统，该软件在界面、操作性、实用性等方面均有大幅提升。

关键词

网络监测，WPF，MVVM

1. 引言

1.1. 网络通断监测系统的现状

在基层网络保障单位中，监控各个站点网络通断情况是很重要的工作内容。出现网络阻断要求能够尽快发现查找原因并排除。目前进行网络通断监控的主要方法，一个是每天不同时间段发测试报，还有一个是利用监控软件。现有的监控软件有两套，一套使用表格形式，在表格中列出需要监控的各个点，后台通过一定间隔对各个点使用 Ping 方法检测，把监测结果显示在表格中。另一套倒是采用图形界面，但操作复杂，并且仅限于特定网络。在实际使用中两套软件都能提供一定的帮助，但都不能很好切合基层工作需要。

1.2. 网络通断监测再次开发的意义

在实际工作中，我们需要一套能够以拓扑图直观显示各个站点通断情况的系统，操作简便，并且能够方便地查看统计历史通断情况；能够查找相关文档；能够形成所需报表等等。换句话说，就是根据单位实际保障需要量身定做，这样可以极大地减少工作量，提高工作效率，更好地完成网络维护保障任务。

2. WPF 和 MVVM 简介

WPF 全称是 Windows Presentation Foundation，是微软新一代图形系统，运行在.NET Framework3.0 及以上版本下，为用户界面、2D/3D 图形、文档和媒体提供了统一的描述和操作方法。WPF 是 Windows 操作系统中的一次重大变革，与早期的 GDI+/GDI 不同，WPF 是基于 DirectX 引擎的，支持 GPU 硬件加速，在不支持硬件加速时也可以使用软件绘制多线程绘制，自动识别显示器分辨率并进行缩放，这都提高了使用者的体验[1]。

MVVM (Model-View-View Model)是 MVP (Model-View-Presenter)模式与 WPF 结合的一种新型架构框架，如图 1 所示。WPF 的数据绑定使得开发人员可以将 View 和逻辑分离开来，通过 ViewModel 获得两者之间的松散耦合。MVVM 模式的主要目的是分离视图(View)和模型(Model)，它具有低耦合、可重用、独立开发、可测试等多个优点[2]。

3. 网络通断监测系统设计与实现

原有的监测系统是用表格视图，如图 2 所示。而我们的主要目标是能够以拓扑图实时显示各个站点的通断情况，但依然保留表格视图，只是升级成带目录树的表格。图 3 所示就是我们目前初步完成的表格视图和拓扑图视图。

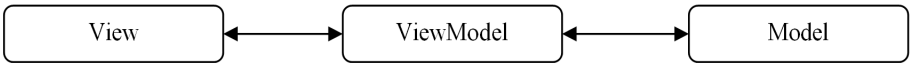

Figure 1. MVVM functional diagram
图 1. MVVM 功能图

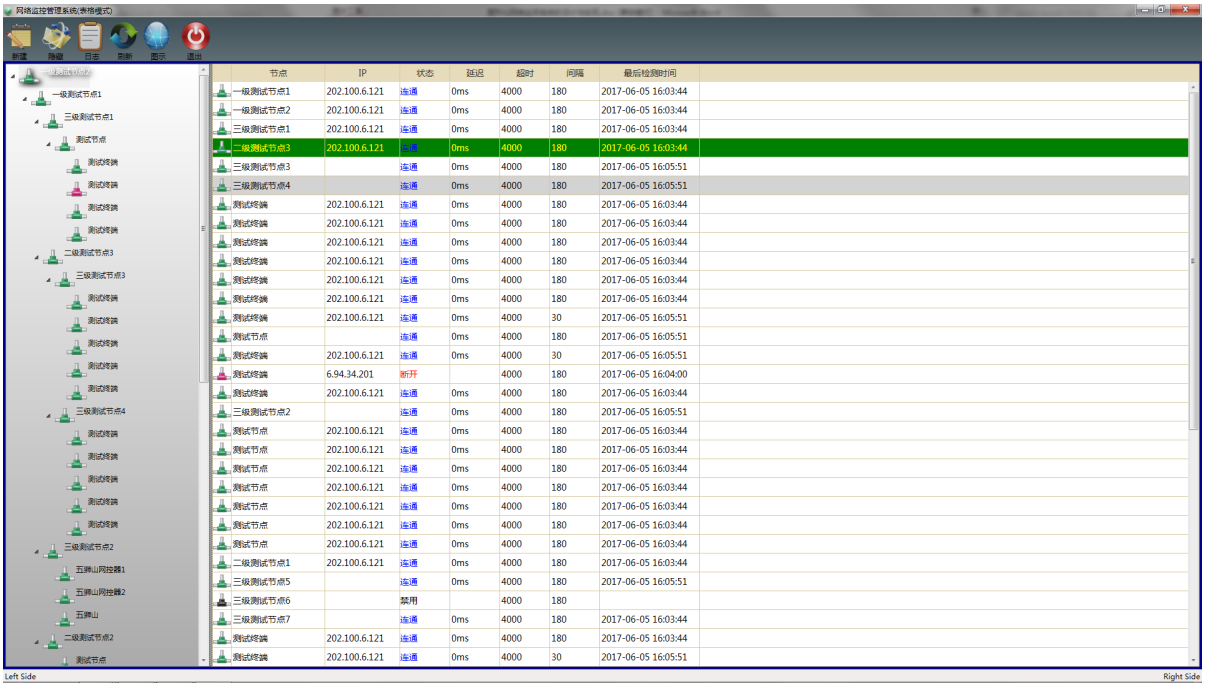

Figure 2. Table view
图 2. 表格视图

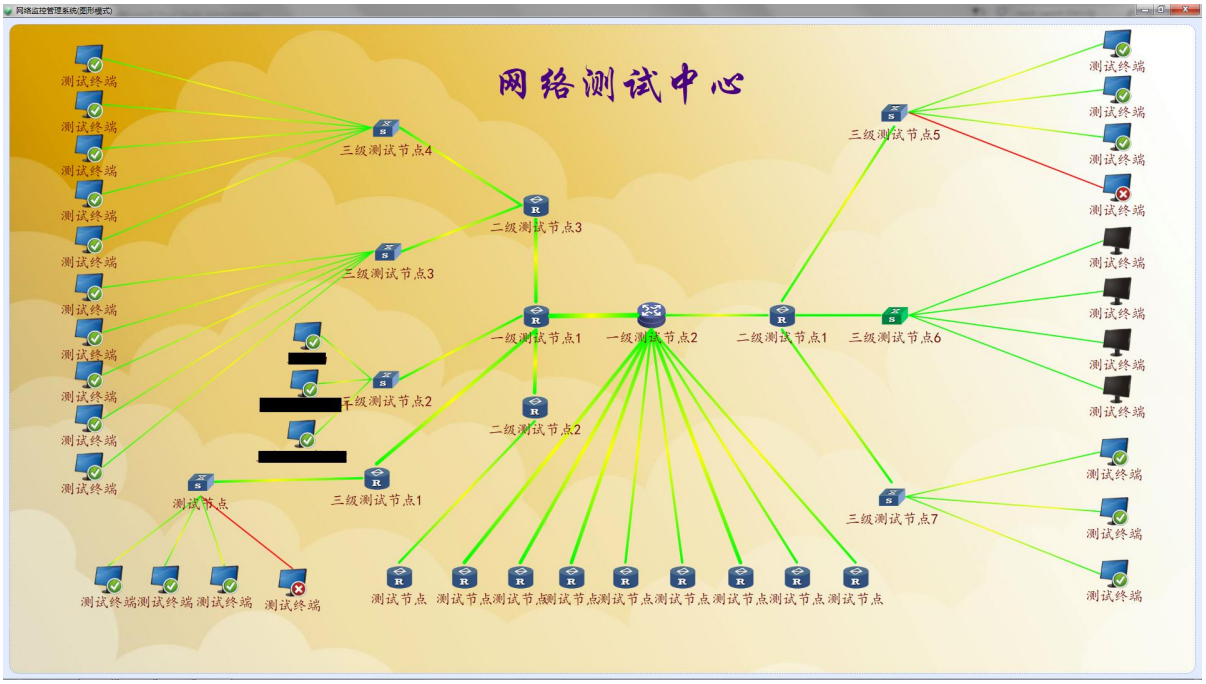

Figure 3. Graphical view
图 3. 图形视图

系统主要包含四个模块：表格和图形界面的实现以及状态同步；后台进行通断测试的异步操作并过滤偶尔丢包现象；文档资料的查阅管理；日志查询管理。日志部分是基本数据库操作，这里不作详细说明。

3.1. 界面实现及状态同步

表格界面类似于资源管理器，采用 TreeView 控件和 DataGrid 控件的组合即可实现。

拓扑图界面采用 Canvas 控件，布局代码如下：

```
<Grid    Background="sc#1.000000, 0.769689, 0.831936, 1.000000" Name="DocumentRoot" >
          <Rectangle Name="rectBack" Fill="white" RadiusX="14" RadiusY="14" Margin="10" Stroke="sc#1.000000, 0.250141,
0.333404, 0.884413" StrokeDashArray="2"/>
          <Canvas Name="gCanvas"    Margin="20" Background="Transparent">
               <Canvas.RenderTransform>
                    <TransformGroup>
                         <ScaleTransform x:Name="scaler"/>
                         <TranslateTransform x:Name="translater"/>
                    </TransformGroup>
               </Canvas.RenderTransform>
          </Canvas>
</Grid>
```

后台代码中则是读取数据库中各节点信息并显示在 Canvas 中，并实现鼠标操作。拓扑图上每个节点有三个部分：主体图标、与上级连接线、IP 地址标签，在节点类 ModelHost 也包含了创建显示这三部分的方法。如下示例创建图表：

```
public void CreateImage()
  {
                if(HostImage==null)
                {
                     HostImage = new Image();
                     HostImage.Tag = this;
                     HostImage.Width = 48;
                     HostImage.Height = 48;
                     Binding binding = new Binding();
                     binding.Source = this;
                     binding.Path = new PropertyPath("BigImage");
                     binding.Mode = BindingMode.OneWay;
                     HostImage.SetBinding(Image.SourceProperty, binding);

                     if(!VisibleInGrid)
                     {
                          HostImage.SetBinding(Image.VisibilityProperty, new Binding("IsShowAll") { Source = VMParame-
                                ter.ZdhParameter });
                     }
                }

                if (this.Position.HasValue)
                {
                     HostImage.SetValue(Canvas.TopProperty, this.Position.Value.Y);
                     HostImage.SetValue(Canvas.LeftProperty, this.Position.Value.X);
                }
                else
                {
                     int result;
                     Math.DivRem(HostsCountWithNoPosition, 10, out result);
                     HostImage.SetValue(Canvas.TopProperty, (double)(result*80+20));
                     HostImage.SetValue(Canvas.LeftProperty, (double)(HostsCountWithNoPosition/10*80+20));
                     HostsCountWithNoPosition++;
                }
                HostImage.ToolTip = "IP: " + this.IP;
                if (this.ParentHost != null) CreateLine();

  }
```

如果节点很多，界面上放置不下，则可以考虑把多个节点合成一个，然后为该节点创建子拓扑图，展开子拓扑图就可以同时进行监控。

对于在两个视图之间以及包含子拓扑图的多个视图之间进行状态快速同步，采用 WPF 的数据绑定技术和属性更改通知来实现是非常合适的。

```
public EnumNetStatus Status
        {
            get { return _Status; }
            set
            {
                if (_Control == 0||_Status == value) return;
                _Status = value;
                if (value != EnumNetStatus.Testing) PreStatus = value;
                SetImage();
                NotifyPropertyChanged("Status");
            }
        }
```

如上代码，我们定义了一个 Status 属性，NotifyPropertyChanged("Status")语句表示当该属性值变化后将自动通知与它绑定的所有目标对象，不需要手工逐个检索。我们的软件有表格视图和拓扑视图，拓扑视图还会包含子拓扑，其个数是不定的。如果逐个对象检索，不仅麻烦，而且效率低，代码不灵活，易疏漏。当我们在创建每个视图的各个对象时，如果需要和 Status 属性相关联，使用 SetBind 语句即可。

3.2. 后台异步操作以及个别丢包过滤

一个监测系统至少要包含几十几百个节点，毫无疑问应该采用异步调用的方式。C#提供了 Ping 类，该类包含 SendAsync 方法，就是发送异步检测，绑定 PingCompleted 事件回调以获取检测结果。检测中偶尔会出现个别丢包现象，可以认为网络不是很好，但不能认为不通，所以需要一定的过滤措施。我们采用的方法也很简单，对于一个 IP，如果一次检测结果为通，则认为是通的，结束这次检测；如果不通，则不再等待而是继续检测该 IP，直到次数为 4 还是不通，则认为此次检测的结果是不通[3]。

以下是调用异步检测的关键代码：

```
host.TestCount = 0;
Ping pingSender = new Ping();
pingSender.PingCompleted += new PingCompletedEventHandler(PingCompletedCallback);
int timeout = host.Timeout >= 4000 ? host.Timeout : 4000;
PingOptions options = new PingOptions(64, true);
host.Status = EnumNetStatus.Testing;
try
{
    host.SImageRotate.Dispatcher.BeginInvoke(new DelegateChangeLineState(host.StartRotate));
    pingSender.SendAsync(host.IP, timeout, host);
}
catch (PingException err)
{
      NetworkErr = true;
      System.Diagnostics.Debug.Print("PingException(HostName:" + host.HostName + " err:"+err.Message);
       host.SImageRotate.Dispatcher.BeginInvoke(new DelegateChangeLineState(host.StopRotate));
        pingSender.Dispose();
       SetHostStatus(host, EnumNetStatus.Off, 9999999);
}
```

3.3. 文档资料的查阅管理

系统内部集成文档资料的管理，有助于更加快速便捷地查找解决故障。当然，这一模块也可以设计成能够单独运行的程序。

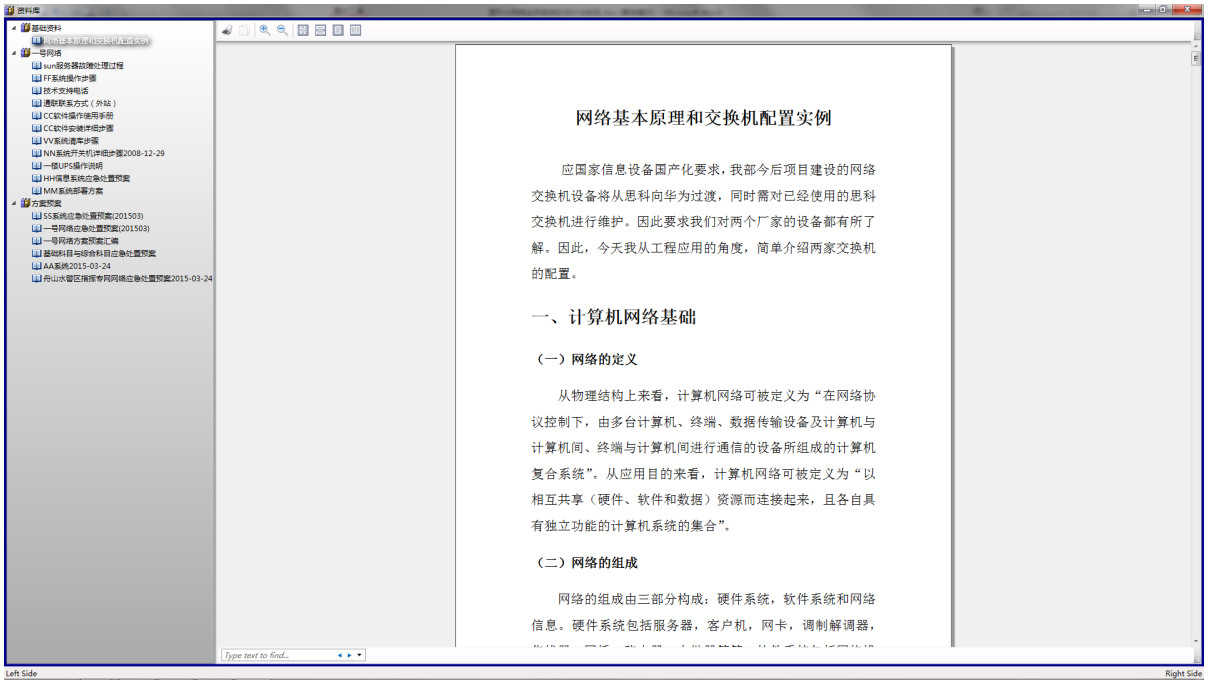

Figure 4. Document access interface
图 4. 文档查阅界面

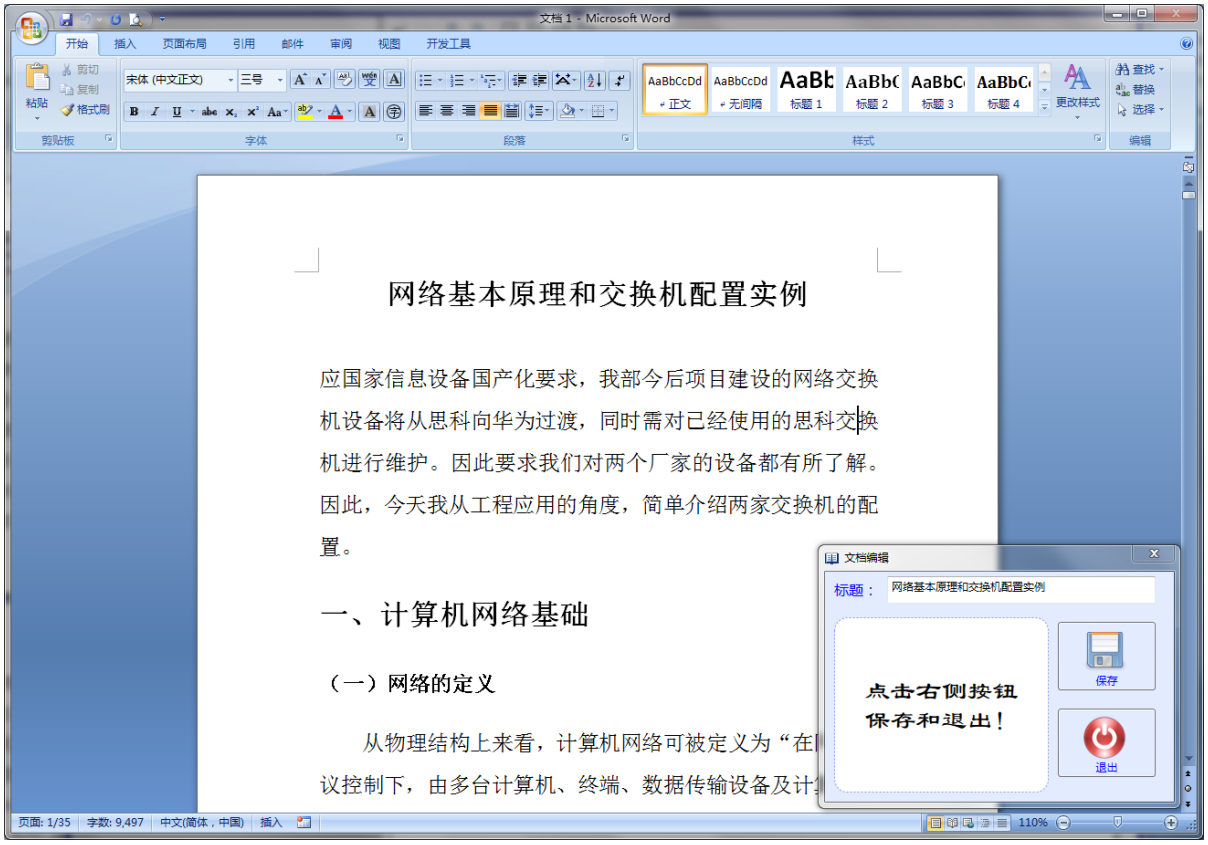

Figure 5. Document editing interface
图 5. 文档编辑界面

对于文档的查看，我们采用 XML 纸张规格(XPS 格式)，这是一种固定版式的电子文件格式，可以保留文档格式并支持文档共享。XPS 格式可确保在联机查看或打印文件时，文件可以完全保持预期格式，且文件中的数据不会轻易地被更改。查阅文档如图 4 所示。

对文档的更新维护，系统采用内部调用 Office 并实行一定的控制来实现。内部调用并用 Office 打开相应文档之后，禁用 Office 的关闭退出按钮，而是系统另外提供一个小面板用于保存和退出。这样可以在保存和退出时内部获取 XPS 格式的文档。如图 5 所示。

4. 性能体验和实际应用

在本单位的试用中，虽然监测节点只有 70 左右，但其中有很多节点代表一段连续的 IP 地址，实际真正监控的IP地址在200个左右。系统运行时在表格视图和拓扑图视图均打开的情况下占用内存约70 M，全体刷新检测时 CPU 占用率为 4%，网络使用率为 0.43%，应该说占用资源很少。

虽然程序功能不多，但对于实际工作却是帮助甚大。直观简洁的界面，详细的通断日志数据等等都给网络故障的发现查找排除带来极大的便利，更有为值班量身定做的一些功能，这都是原先监测软件所不具备的。比如每天上报的各节点断网时间，可能某个节点通断好几次，在原先软件中会记录在几个文件中或一个文件的好几个位置，需要仔细查找。利用现在的软件可以一键查看。

5. 结束语

实现这样一个系统，没有很难的技术点，但却非常适合实际使用需求，能有效提供网络保障效率。软件开发，技术实力固然重要，但只有符合实际使用需求，才能做出具有生命力的软件。

参考文献

[1] 李应宝. WPF 专业编程指南[M]. 北京: 电子工业出版社, 2010.

[2] (美)麦克唐纳. WPF 编程宝典[M]. 王德才, 译. 北京: 清华大学出版社, 2013.

[3] (美)库罗斯, 罗斯. 计算机网络: 自顶向下方法[M]. 原书第 6 版. 陈鸣, 译. 北京: 机械工业出版社, 2014.