NetAnalyzer 使用说明书

二．扩展与开发

墨云软件 ● 2019-03-16

目录

第一部分 MangoScript

《道德经》中有“道生一，一生二，二生三，三生万物”的说法，描述了万物从少到多，从简单到复杂的一个过程。在计算机中我们所面对的各种各样的文件，如：图片，文本，音乐甚至最基本的程序文件其实都是通过二进制数据也就是大量的 0 或 1 的方式存储在硬盘或内存中的。但是如何从 0 和 1 转换为我们熟知的各种媒体数据呢，这就需要根据 0 和 1 不同的排列顺来完成，这就是编码方案，而这种编码方案更通俗的来说就是一种协议，这种协议来约束不同的设备，不同的系统当遇到对应的数据是应该将其解析为什么文件。

当今网络作为与我们生活朝夕相关的事物，给我们带来了便利的生活体验，有些应用甚至可以做到计算机与智能手机之间的无缝切换，这就得益于网络中各个层次的协议完美对接。目前的互联网模型大部分都是基于经典的 TCP/IP 协议，虽然其安全性、传输效率等问题在这些年逐步暴露出来，但是其拥有的完整协议体系却是其他协议体系不具备的。从物理层使用的 CSMA/CD(载波监听多路访问冲突检测)协议实现端到端的数据传输，再到网络层中 IP 通信协议，RIP、OSPF 网

络路径发现协议，实现从主机与主机实现跨网数据传输的功能，在而到保证让主机接口可以获取到无差别数据的 TCP 协议、实现最终数据呈现应用层协议，如 http 协议。这些协议都是公共开放的协议类型，而有部分软件就是基于这些公共协议进行工作的，如基于 http 的各种浏览器、基于 FTP 的各种文件传输 软件，虽然基于公共协议的软件很多，但是我们大部分情况下使用的更多的是专属软件，这部分软件具有自己独立的协议，而且很大的一部分是在 TCP 协议之上建立起来的私有独立应用协议。

创建 MangoScript 的初衷就是为了解析这部分协议，私有协议是一个公司或一个组织定义的一套专属于内部的数据交流方案、这些协议可能因为涉密或是团体影响力过小并不能被外部人员获取到。而想要分析这些数据，箱借助协议分析工具进行分析是不可能的，而手动从各种二进制数据中获取信息，效率又极其低下。MangoScript 的思想就是通过将数据方案转换为对应的脚本代码，将代码绑定到 NetAnalyzer，通过 NetAnalyzer 实现与解析公共协议无差别的数据分析。

MangoScript 作为 NetAnalyzer 扩展协议分析的专职语言，设计的更像一种配置文件，可以通过不同的配置方式，实现对数据流的解析。脚本使用协议分析树的逻辑方法，脚本编辑方式就是协议树的呈现方式，即是没有接触过编程的人也可以轻松进行代码编写。

当然，因为 MangoScript 正处于测试开发阶段，所提供的功能也不近完善，这需要读者的体谅，也很希望读者可以提供一些好的建议与意见。目前脚本采取宽泛执行的方式，即对于一些语法错误会自动忽略，以保证尽可能的完成数据分析。

1. MangoScript 语法规则

通过 MangoScript 可以快速的对数据的结构进行描述与呈现。并且语法非常简单，适合快速入手使用。 在 MangoScript 中大小写不敏感（部分函数提供的参数除外），支持定义中文字段。

代码整体可以认为有两部分：

- 对代码整体结构的约束定义（block 代码）
- 对具体数据呈现方式的定义（node 代码）

从某种意义上来说 MangoScript 更像是一种配置文件，因为该语言目前还不支持判断、循环等逻辑，只支持一种简单的分支，此外还不能自定义数据处理函数。这也是墨云一直以来称其为语言有迟疑的地方。 然而从解析数据来看却要比真正所谓的脚本语言要快捷的多。

这是一段最简单的 MangoScript 代码：

```
/*
  定义 block(结构块)
*/
block main
{
   //定义标题
   title "我是标题";
   //定义一个 node(节点)
   node node1= select(0,1/*注释 选则长度*/);
}
```

1.1. 数据类型

在函数处理脚本中，需要明确函数的输入输出类型。以便于对数据更好的处理。在 MangoScript 中有 字节数组、数字、字符串 三种数据类型

- **字节数组** 由一个或多个字节组成的一中数据结构，可以通过索引与长度值获取子字节数组，MangoScript 待测数据都是字节数组。
- **数字** 包含无符号数字(uint)和有符号数字(int)，目前 MangoScript 包含最大为 4 个字节的数字(C#中为一个 int 值所占的字节)，在 MangoScript 中数字可以使用三种方式输入：十进制，我们输入的普通数字；十六进制，以 0x 开始的数字，如 0x1A；二进制，以 0b 开始的数字，如 0b01010
- **字符串** 字符串或串(String)是由数字、字母、下划线组成的一串字符。它是编程语言中表示文本的数据类型。在 MangoScript 中每个节点完成数据转换最终呈现的都是字符串。

1.2. 注释

MangoScript 使用两种注释方式，通过双斜线 **//** 实现单行注释 使用 **/* 内容 */** 实现行内选择注释和实现多行注释。

单行注释示例：

```
    //定义标题
```

多行注释示例：

```
 /*
   定义 block(结构块)
 */
```

行内注释：

```
    node node1= select(0,1/*注释 选则长度*/);
```

注释内容只是用于编码辅助记忆作用，不参与代码编译与分析。

1.3. 关键字

代码中定义了一些关键字，以方便进行代码的语法分析，在 MangoSctipt 中定义的关键字有：

block	title	node	enum	case

大部分的定义的关键字需要连同":"一起使用。因此在代码中定义节点名称使用代码中已经定义的关键字也是允许的。 如：对 block 的定义 `block block{…}`或对 node 的定义 `node node= select(0,1);`都是可以正常使用的。

1.4. block

block 我称之为结构块。用于定义一组数据呈现的结构体。block 中包含一整块数据的规划与处理,代码定义方式为

```
 block <name>
{
   …
}
```

代码 1-2

其中代码中 name 为必填项。

如在的代码 1-1 中就定义了一个名字叫 main 的 block。

在结果呈现上，大部分情况下表现为父级节点。 在一段代码中可以定义多个 block，显式定义中，不允许存在嵌套(在 switch 函数下可以定义匿名 block,这种定义方式为隐式定义。具体请看该函数的功能说明)。并且在该段代码中必须存

在一个名称为 **main** 的 block 作为代码起点，结构块的先后顺序不受影响数据解析方式。

1.5. title

title 主要用来显示父级节点具体的文本内容，如

```
title "测试文本";
```

那么在定义的 block 呈现的父级节点上面就会显示“测试文本”，该定义为非必须字段，如果缺省则会显示 block 的名称，如"main"。

1.6. node

node 是 MangoScript 用于描述数据呈现方式的最基本部分。node 通常用来描述一个子节点由数据转为自己制定类型过程和数据的呈现方式，具体的呈现内容则是通过一系列函数链进行不断演进的结果。 函数部分通过内部定义(MangoScript 不支持自定义函数)MangoScipt 通过对各种函数进行对应的选取排列来获取需要的值，而具体的函数方式可以通过函数 API 文档得到相关的帮助信息,如下代码为定义一个节点：

```
node data = select(2, 8).text("ascii");
```

首先使用 node:作为前缀，然后定义节点名称对应的变量没成 data ，定义完名称之后通过=开始函数部分的编写。

首先我们需要选取数据区域，在这里我们通过选取函数 select 获取从数据块中第 2 个开始 8 个字节的数据块。

当完成数据选取以后，再执行 text 将选取到的 8 个字节转为文本编码为“ASCII”的字符串。

最后将结果转换后的结果赋予变量 data ,node 定义以分号“;”结束。部分 node 还有方法体，使用大括号包裹起来，还是以分号“;”结尾。

函数

在 node 中用于描述数据转换的方式，就是这里要说的函数。函数通常使用 “.” 符号开始，如上面的代码，其中的 select 函数和 text 的函数都是通过 “.” 符号开始的，接下来就是函数名称，并且在括号中输入相关的控制参数。因为不同的函数输入的参数类型和内容不一样，并且随后还会不断的扩展函数库， 通过多个函数一起链接，node 就形成列函数链，函数链从左往右，前一个的函数输出数据是后一个函数的输入数据，对于第一个函数，默认为全部的待处理数据，这就是为大节点都是以 select 函数作为开始的，对于最后一个函数则统一处理为文本进行输出。 在开发中需要明确每个函数的输入输出数据类型，对于函数类型或参数可以通过文档后面的函数 API 进行查阅。

一些特殊函数说明

在 MangoScript 中有一些特殊的函数需要单独的说明一下。这些函数为脚本提供了最基本的数据访问和结构控制的功能，整个 MangoScript 都是建立在这些功能上面的。

select(offset,length) select 函数，数据选择函数，是 MangoScript 中核心函数之一，主要功能是从待分析数据块中根据 offset 参数和 length 参数获取到需要处理的数据，如上面的代码中从第 3 个字节开始(索引都是从 0 开始的,所以 offset=2)找到 8 个字节(length=8)作为待处理的子字节数组。对于 select 函数，

除了可以进行正常的选择转换之外，还可以进行细节扩展，对该字段进行进一步的描述，通过以子节点的方式呈现出来。代码如下：

```
node 时间戳= select([帧长度]-12,14).text("ascii")
            {
                    node 年=select(0,4).text("ascii");
                    node 月=select(4,2).text("ascii");
                    node 日=select(6,2).text("ascii");
                    node 时=select(8,2).text("ascii");
                    node 分=select(10,2).text("ascii");
                    node 秒=select(12,2).text("ascii");
            };
```

如上面的时间戳节点，呈现的只是简单的将选中的数据转为以ASCII编码的文本，但是如果我们想要知道其内部的具体细节，则需要借助子节点功能。在主节点最后一个函数后面添加大括号，再在大括号中定义子节点，这里的子节点选择的数据为主节点选中的数据，所以需要将索引值置为 0 重新开始。如：

```
node 年= select(0,4).text("ascii");
```

最后需要注意的是，在大括号后面加需要有一个分号，结束对该节点的定义。

while(offset,length) while 函数，结构循环函数。在数据分析过程中，需要特定的结构，对数据块依次进行相同的分析，很多情况下，我们并不知道该循环需要执行的次数，这就需要通过脚本来自行判断，这时候就用到了 while 函数，该函数是有方法和 select 带子节点结构一致，但是解析方式是不一样的，该函数只需要定义开始分析位置，以及所要分析的长度，之后再在函数后面添加需要循环的分析块，该函数会自动根据填写的内容自动判断需要循环的次数。示例代码如下：

```
node 序号= while(4,16)
      {
          node sub=select(0,2);
          node sub2=select(2,6);
      };
```

该代码会根据偏移量最大的一个节点(该节点的偏移量加选择长度)作为一次循环的结束的标志，进行自我判断，对数据进行循环解析，直到所选数据结束为止。如上面代码，会被解析两次，因为 sub2 节点结束时候数据偏移到 8(2+6)，当前选择的数据为 16，所以可以再次进行一次循环。

switch(offset,length) switch 函数，转换函数。在一些业务分析过程中，通过会有根据不同字段，后续所要分析协议格式不同的问题。这种情况下就会用到 switch 函数，在 switch 中有两个参数，和 select 一样用来选择数据，但是与 select 不同的是，当 switch 选择完数据后，会直接转为数字类型(也就是说 length 最大为 4)，并在 switch 对应的函数体内进行判断，在改函数体内，通过 case 关键字列出不同的值，并且指向不同的 block，如果 switch 选择的数据与其中一个 case 的值相对应，则会指向对应的 block 代码，在 case 中指向的 block 有两种方式：1.通过>方式的指向，该种指向为外部定义的 block,后面只需要输入对应的 block 名称即可；2.通过:方式的指向，这种指向使用匿名方式建立一个 block，不需要是使用 block 关键字，不需要定义名称，只要在后面输入大括号，就可以进行代码输入了，和平时定义 block 一样。

如下代码：

```
node date = switch(0,2)  //switch 自动将其转为无符号整数
{
  case 0x0101>Test,//指向一个 block
  case 0x0102:  //该种结构又叫做匿名 block
  {
    node test=select(3,34);
  }
};

      ……
```

```
//定义的另外一个节点
block Test
{
  node name = select(0,1).num(4,"hex");
}
```

对于输入数据[0x01 0x01 0x03 .....]，我们通过 switch(0,2) 获取到数字 0x0101(十六进制方式)，则会跳转到 block 名称为 Test 的结构块进行分析，注意我们这里使用 `case 0x0101>Test` 这是使用外部 block 调用的方式。

如果输入数据为[0x01 0x02 0x03 .....] 得到的数字为 0x0102，在这里使用的方式是内建匿名 block：

```
case 0x0102:
{
  node test= select(3,34);
}
```

这两种方式都可以按照普通的 block 一样使用。 当分析完数据后，并不会显示 switch 所在的节点内容，而是使用对应 case 所指向的 block 中的所有节点来代替。

ifblock(flag) ifblock 函数，结构判定函数，在处理一些协议总会看到 Magic 字段，如某款 IM 软件协议中第一个字节就是 0x02，这些协议通常是和其他服务共用了某些特征，如某款 IM 软件使用 8000 端口号，但是有好多应用都会使用这个端口，为了正确的识别这些协议，于是有了 ifblock 方法。 该方法的 flag 为数字类型，所以前面 select 所输入的长度不能超过 4。比如我们在判定是否为某款 IM 软件协议的时候,输入代码：

```
node flag=select(0,4).ifblock(0x02);
```

如果待测数据第一个字节为 0x02 则继续进行下面的分析，如果不是则直接跳出，返回空的 block。

display(flag) display，显示函数，在一些协议中，有些字段本身其代表的是一种类型，如：icmp 中的类型字段，对应每个字段都有不同的意义。但是在做协议分析的时候，我们拿到的仅仅是代码，如果能将代码对应的字段使用文本方式呈现出来，则更加具有可读性。 display 函数正是基于这种思想实现的。在使用 display 之前，我们首先需要定义对应关系。display 的对应关系我们这里叫做 enum，该结构被定义在 block 外部，主要是为了共享 enum，以下以某款 IM 软件协议(精简过)为例，定义方式如下：

```
block main
{
   …… //nodes


}
enum imcomond
{
     case 0x0001 > "注销登录",
     case 0x0002 > "心跳信息",
     case 0x0004 > "更新用户信息",
     case 0x0005 > "搜索用户"
     ……
}
```

在 block 中定义 display Node

```
  node 命令= select(3,2).display(imcomond);
```

当待测数据为[0x01 0x01 0x01 **0x00 0x02** 0x00]，输出为：

命令:心跳信息

对于更多的函数可以查看附录部分的 MangoScript 函数列表。

1.7. enum

enum 作为一种在 MangoScript 中的块，主要作用是提供协议中的数据类型识别。Enum 可以被定义在 block 外部和 block 平级，也可以定义在 block 内部，但是对于分析器仍然会将其解析到公共区域，所以任何 block 中的 display 等相关函数可以使用任何位置定义的 enum，当然这也决定了，在一次执行的 MangoScript 中 enum 名称必须是唯一的。

1.8. case

case 作为 MangoScript 的分支关键字，主要的作用是提供一个备选的值，如果当输入值和被选值一致是则执行刚当前 case 定义的块或获取对应的值。

目前 case 只用于两种结构中。

- switch 结构

  该结构在讲述 switch 函数的时候已经做过说明，switch 传出参数作为输入值，其结构中各个 case 备选值进行比较，当与其中某一个 case 相等则执行该 case 对应的块。在 switch 结构中 case 有两种方案，指向类型和定义类型。指向类型 使用 `case xxx > block1` 该种指向为外部定义的 block1,后面只需要输入对应的 block1 名称即可；定义类型为 `case xxx:{ node:… }` 方式，这种指向使用匿名方式建立一个 block，不需要是使用 block 关键字，不需要定义名称，只要在后面输入大括号，就可以进行代码输入了，和平时定义 block 一样，注意各个 case 之间使用逗号分隔。

- enum 结构

  对于 enum，case 使用指向方式，指向一个对应的字符串即可，`case xxx > "注销登录"`；注意各个 case 之间使用逗号分隔。

1.9. 代码分析

下面我们一起来分析这个示例

```
block main
{
  title "协议测试";
  node 前缀= select(0,2);
  node 序号= select(2,2).num();
  node 版本= select(4,3).eachbyte(".","num");//num text
  node 帧类型= select(7,1).display(FrameType);
  node 帧长度= select(8,2).num();
  node 数据类型=select(10,1).display(mtype);
  node 代码= select(11,3).reverse().num();
  node 数据块= select(14,[帧长度]-26);
  node 时间戳= select([帧长度]-12,14).text("ascii")
              {
                      node 年= select(0,4).text("ascii");
                      node 月= select(6,2).text("ascii");
                      node 日= select(6,2).text("ascii");
                      node 时= select(8,2).text("ascii");
                      node 分= select(10,2).text("ascii");
                      node 秒= select(12,2).text("ascii");
              };
  node 校验和=select([帧长度]+2,2).num();
  node 后缀=select([帧长度]+4,2);

}
enum FrameType
{
    case 0x00 > "帧类型 1",
    case 0x01 > "帧类型 2"
```

```
}

enum mtype
{
    case 0x00 > "测试类型 1",
    case 0x01 > "测试类型 2",
    case 0x02 > "测试类型 3"
}
```

在该代码中，可以看到只定义了一个数据块 main 和 enum 块，main 数块的 title 为“协议测试”然后定义了 11 个 Node，都是常规定义定义方法，这里需要注意一点，从**数据块**开始在 select 函数的参数中有一个用中括号括起来的**帧长度**字段。这是一种引用字段数据的方式，但是使用引用字段的字段必须定义在被引用字段之后。除了引用字段还有找一种叫做公共参数的变量、虽然目前在 MangoScript 中只定义了一个也就是 **END** 表示一直到数据末尾，所以我能可以这样使用它：

```
   node 数据块= select(0,END);
```

通过改代码可以获取到整个数据块。

2. MangoScript 在 NetAnalyzer 中的使用

通过前面介绍 MangoScript 的语法规则。我们已经能够编写该类型脚本了，但是这种脚本存在的意义呢。前面在介绍 MangoScript 的时候已经提到过，该脚本主要是为了配合 NetAnalyzer 解析协议而从在，这一章就是用来说明如何在 NetAnalyzer 中使用 MangoScript 的。

2.1. 认识代码编辑器

MangoScript 编辑器(简称 MgsEditor，下同)是专门为 MangoScript 语言而开发的集成开发环境，该编辑器作为 NetAnalyzer 组件之一随 NetAnalyzer 安装包一起发布。集成了脚本的编辑、调试、分析、维护和 NetAnalyzer 关联等一系列的功能。

MgsEditor 窗口和 NetAnalyzer 主程序框架和主题是一致的，都是使用现在主流的 Ribbon 方式。上面是菜单栏，下面则是工作区域，包含了代码库、函数列表、代码编辑窗口、结果、测试数据、日志、分析等各个功能区域。

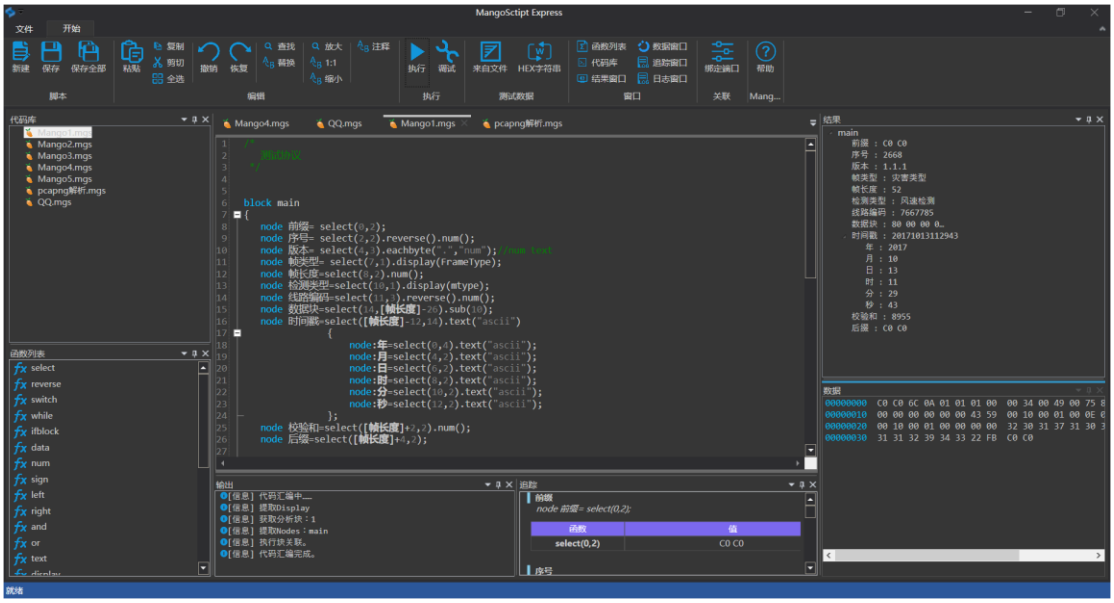


图 2.2-1 MangoScript 代码编辑与调试界面

如上图所示，这就是 MgsEditor 的主界面。菜单部分使用类型 Office 的 Ribbon 方案，而工作区域则使用 VS 类型的 Dock 方案，这样编辑器在使用的时候更加方便。MgsEditor 还具有布局自动保存功能，这样在下次启动的时候会自动加载已经上次关闭所保留的布局方案，对于代码部分 MgsEditor 更进一步会保留上次代码的编辑状态，包含是否保存状态，这样在遇到需要快速关闭的时候，就不会弹出是否保存的情况。

接下来多各个工作区域进行说明。

代码库

在 MgsEditor 中代码的维护全部在代码库中维护，目前 MangoScript 都是用单文档方案，所以每个文档都是一个可以独立运行的个体。

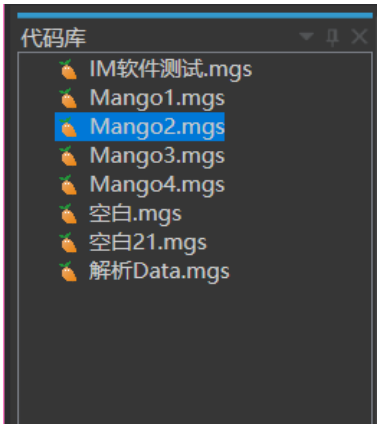


图 2.1-2 代码库

通过代码库可以将所有的 MangoScript 进行集中的维护与使用，之所以使用这种方法是为了配合 NetAnalyzer 中对脚本的使用。通过代码库，我们可以对脚本进行简单的管理如重命名、删除等操作。

函数列表

函数列表主要是用来查询一些函数的信息，辅助编程。通过这个列表可以查到目前代码版本所支持的函数信息，以及其相对应的控制参数信息。通过双击选中的函数，可以向已经打开的代码编辑器光标位置插入函数。

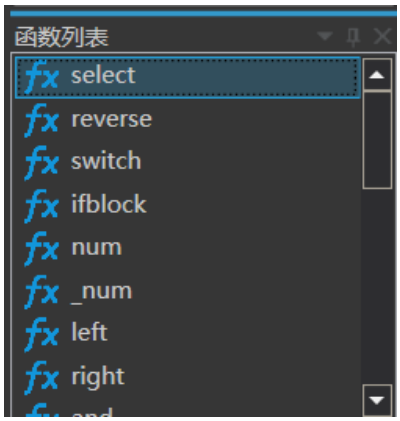


图 2.1-3 函数列表

代码编辑器

代码编辑器作为 MgsEditor 的核心部分，已经支持 MangoScript 的语法高亮显示，并且还支持函数部分的代码提示，每次输入 “.” 即函数前缀符合，则会自动列出备选函数信息。并且给出对应的函数提示信息。在编辑过程中还可以使用 ctrl+shift+>键强制进行自动提示，方便编辑使用。

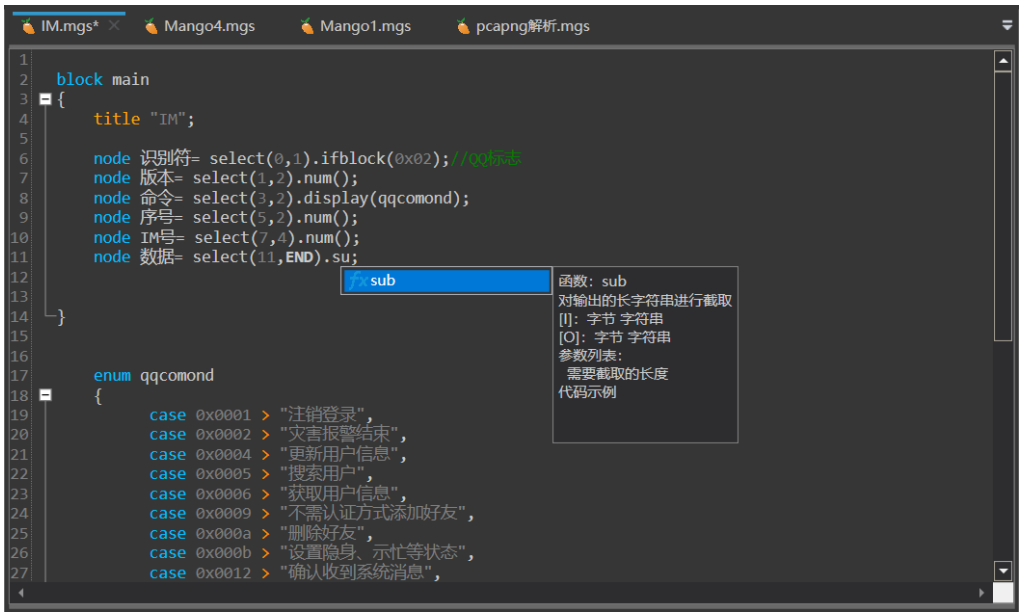


图 2.1-4 代码编辑器

MgsEditor 支持多文档编辑的功能，可以同时打开多个文档进行处理，通过点击上面的标签进行切换，当文档标签上面有* 的时候表示需要当前代码已经存在更改，并且没有保存。

在菜单栏中，提供了编辑器常用到的各种功能，这部分功能只对于当前正在进行编辑的代码有效，这部分功能中除了常规文本的编辑信息，还可以对编辑器的内容进行缩放处理，满足不同人的编码习惯。

图 2.1-5 编辑菜单

结果与测试数据

这两部分基本可以联合起来一起使用。主要用在对代码的测试与呈现部分。当完成代码的编写后，需要进行测试，上面部分为脚本解析并且执行后呈现的最终结果，而下面部分就是对应解析结果的原始数据，通过点击上面的字段，可以看到下面被选中的部分，就是改字段呈现内容的原始数据。

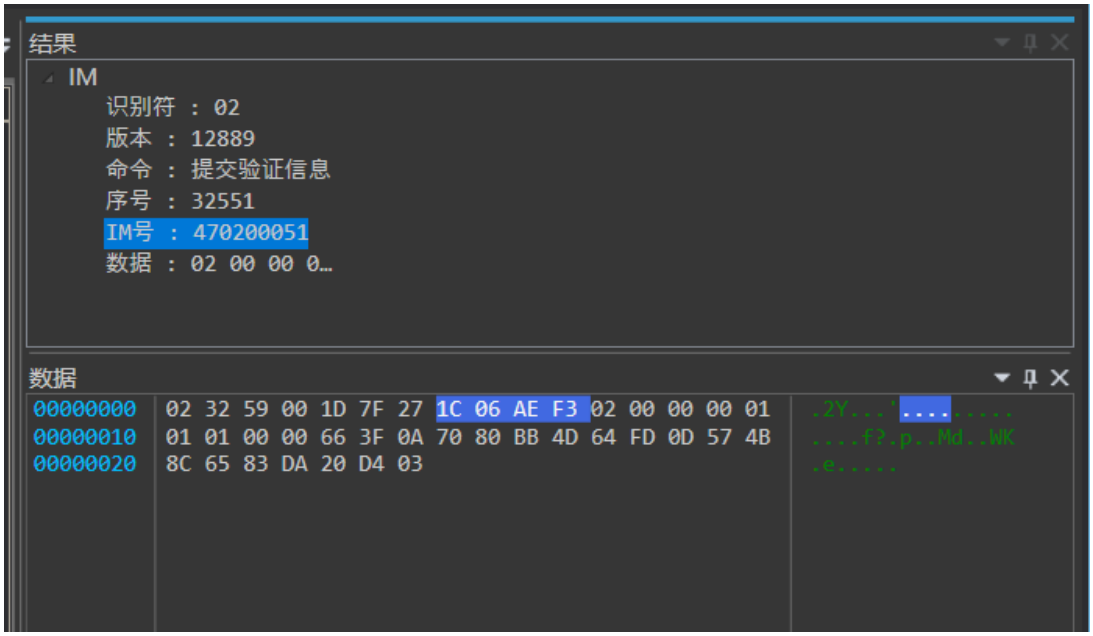


图 2.1-6 结果与测试数据

对于调试部分，因为涉及内容很多，所以具体是的使用方法可以通过下面一节进行说明，在这里只是简单的介绍一下窗口信息，在下一节中进行详细说明。

输出日志

日志窗口为调试代码过程中最为关键的信息输出窗口，无论在代码编辑或调试过程中都非常重要，该窗口不但提供了错误信息的显示，更进一步提供对错误部分的快速定位功能。

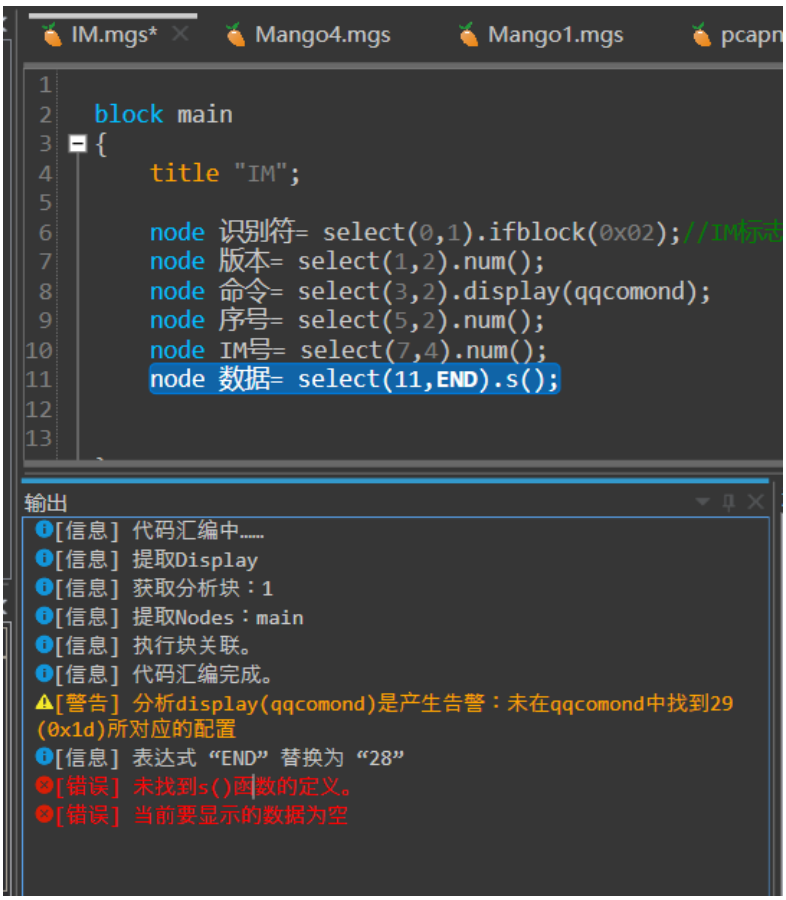


图 2.1-7 日志信息输出和错误定位

在日志窗口中的信息分为三个等级【信息】、【警告】、【错误】，信息通常为一些代码汇编数据解析信息，根据主题不同按照常规文本形式输出；警告信息则是提示一些不影响代码执行，但是可能输出错误的信息。作为着重提示用，通常为橘色文字输出；最后一种错误信息则是表示代码分析过程中已经产生了严重无法处理的错误，需要必须修正。这部分信息会以红色文字的方式输出到日志窗口。如果是字段级别产生错误。用鼠标点击红色部分，代码编辑器会自动跳转到执行的代码，并且高亮选中当前出错误的行。

数据追踪

处理日志窗口，还有一个追踪窗口在进行 node 数据分析时很有作用。该窗口在进行生成代码操作过程中，会自动列出各个 node 在函数链中每个时期的数据内容。

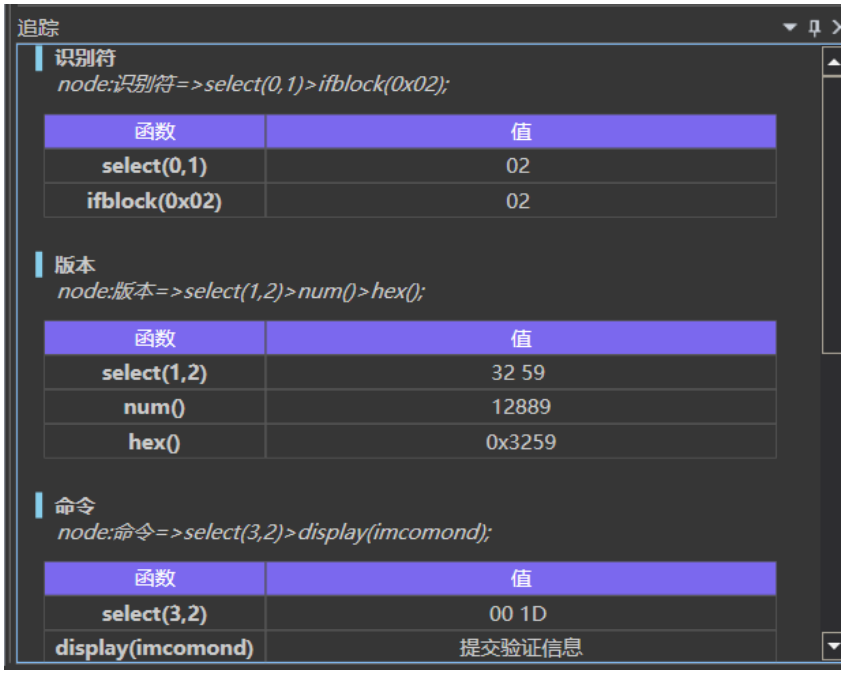


图 2.1.-8 菜单栏

这在进行调试 node 字段取值时候非常有用。

菜单栏

在介绍代码编辑器的时候我们提到了菜单栏，这里我们简单进行一些说明。在 MgsEditor 中菜单相对比较简单，只有一个菜单标签。分为五组。

图 2.1.-9 菜单栏

这五组作为分别对应着七类功能点。

- 脚本：代码编辑器中的常规功能点，在在建立代码和代码编辑时候使用。
- 编辑：为代码编辑区域通用文本代码编辑功能。
- 执行：主要用来执行与调试代码使用。
- 测试数据：可以通过文件方式或字符串方式提供待测数据
- 窗口：视图功能，当某个窗口被关闭了可以通过这个部分重新打开。
- 管理：与 NetAnalyzer 中建立脚本映射，绑定脚本到对应的特征字段上，之后再 NetAnalyzer 进行协议分析的时候就会自动加载绑定的脚本(目前只针

对基于 TCP/UDP 的应用层协议，特征字段为端口号)。

MangoScriptAPI：函数库 API 文档，与文后附录中的 MangoScript 函数列表一直。

2.2. 开发与调试

前面对 MangoScript 编辑器进行了说明。那么接下来就来开始编写第一个 MangoScript 脚本吧。

新建代码

在开始菜单里面找到NetAnalyzer文件夹，在文件夹下就有MangoScript编辑器，根据 Windows 系统版本不同呈现方式不一样，这是 Window10 下面的开始菜单。

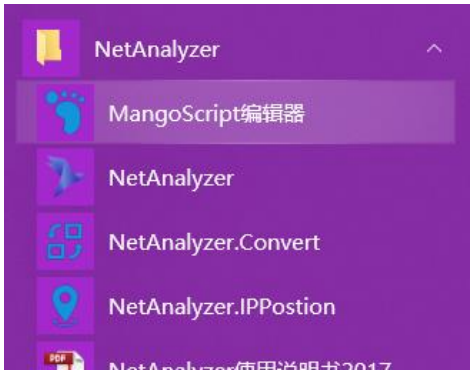


图 2.2-1 找到 MangScript

在菜单栏中的脚本组中点击新建按钮，打开新建脚本对话框，输入脚本名称，注意，此处要建立以名称为文件名的文件，所以此处需要遵循 Windows 中的文件命名规则。

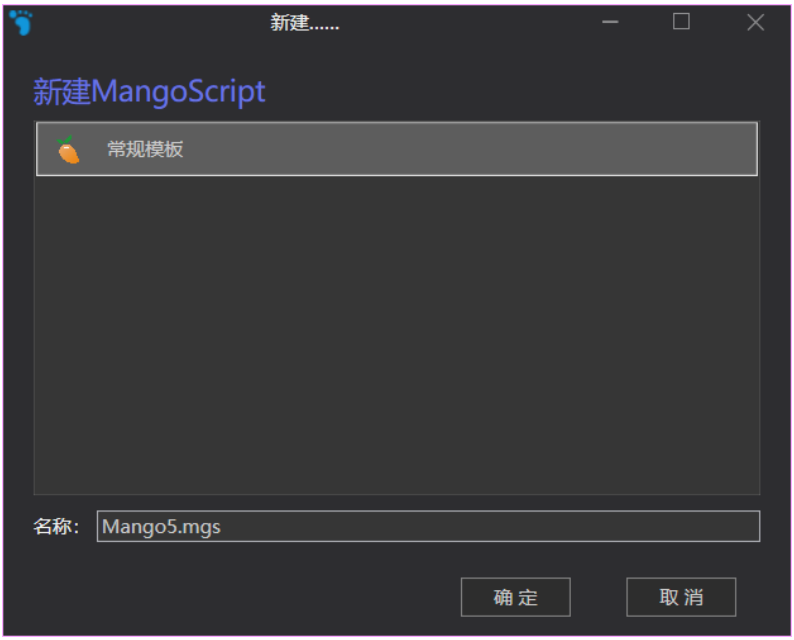


图 2.2-1 新建脚本

点击确定就可以完成新建本的建立。

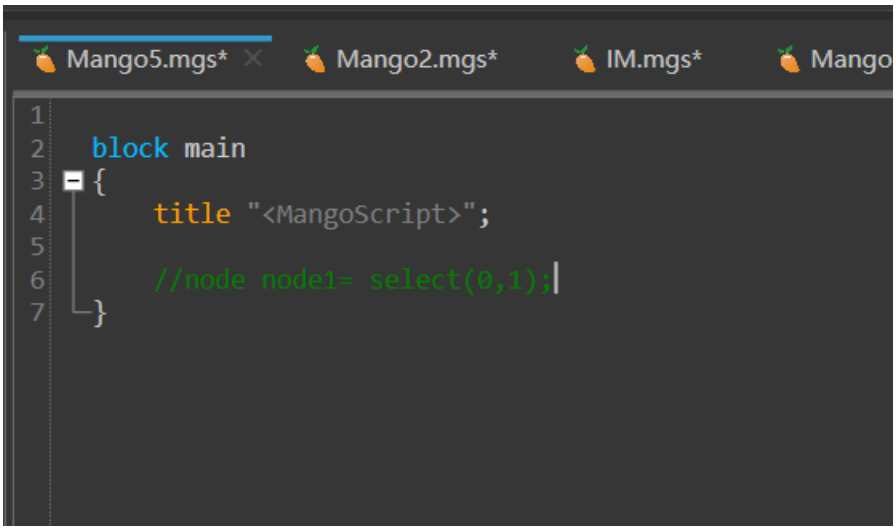


图 2.2-2 建立脚本

我们可以看到在编辑器中已经自动添加了 MangoScript 的基本框架代码代码。

虽然什么也没有，但是这也是一个完整的代码。点击菜单中执行组中的调试按钮，这时候观察日志输出窗口，代码是正常运行的。

接下来就可以按照自己的目标进行代码编辑了。

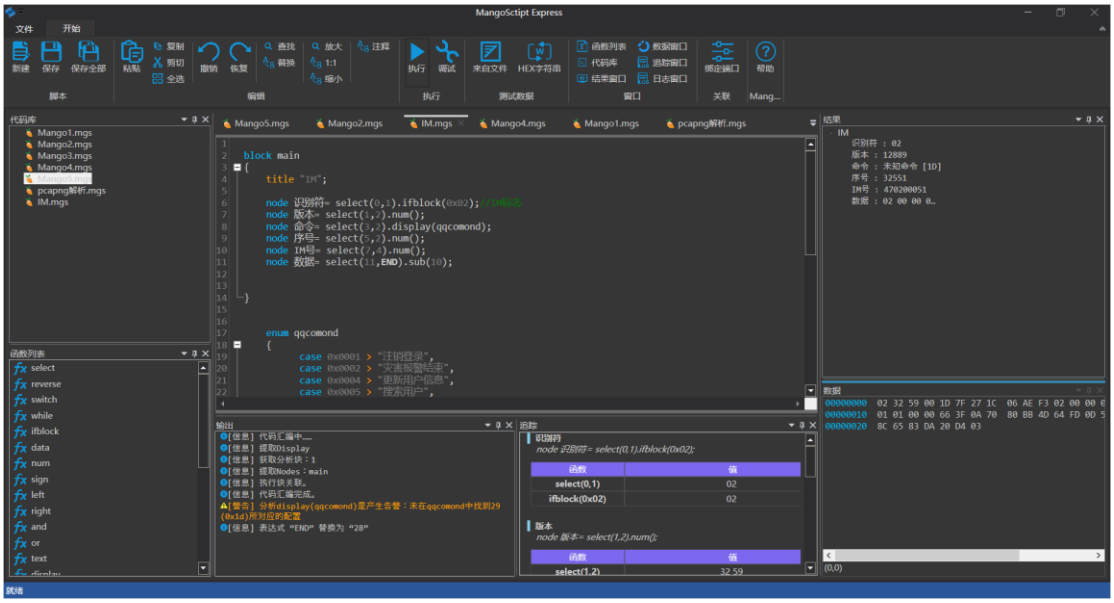


图 2.2-3 编辑完成的代码

测试与调试

通常来说在完成代码的开发以后需要对脚本进行测试与验证，才能真正交付使用者。作为脚本的 MangoScript 在完成开发以后同样需要进行验证正常以后才能被交付 NetAnalyzer 使用。

在 MgsEditor 中对 MangoScript 的执行分为两步：代码汇编，数据解析

代码汇编，主要的作用就是分析代码编辑器中的代码，并且转换为内部识别结构。在这个过程中会去检查代码语法语义是否有异常，当代码正确的被转为内部结构，那代码汇编部分完成。进入数据待测阶段。这个阶段产生的异常为语法错误。

数据解析，这一步是完成了代码汇编，开始根据已经转换完成的内部结构对测试数据进行解析，这个过程中会产生很多未知的因素，尤其是如果在下一个 node 解析中存在使用了前面的 node 作为参数，这部分在进行协议分析时候尤为常见，如：在很多情况下需要根据协议的长度去确定下一个字段出现的位置。这是后就需要认真排查，这个过程会产生很多错误，这种错误叫去运行时错误。

MgsEditor 在菜单栏中提供了**执行**组，用于对 MangScript 的测试与调试。

图 2.2-4 执行与测试组菜单

执行功能，是 MgsEditor 中最重要的功能，该功能一次完成了上面所提到的两种功能：代码汇编、数据解析。如果是第一次使用这个功能会提示不存在测试数据，这是因为在 MgsEditor 中还没有设定需要测试的数据，当点击确定以后会自动启动打开文件窗口选择需要测试的数据即可执行。

调试，这部分相当于只完成代码汇编部分的操作，主要用来检查语法规则，或是不方便测试数据的情况，但是这个功能只能检测出语法类错误，并不能完全排除脚本出错的可能性。

测试数据　提供两种方式来载入待测数据，通过文件直接读入或是通过十六进制字符串方式输入，随着这部分的数据设定，在数据窗口会以二进制方式呈现当前选定的数据。

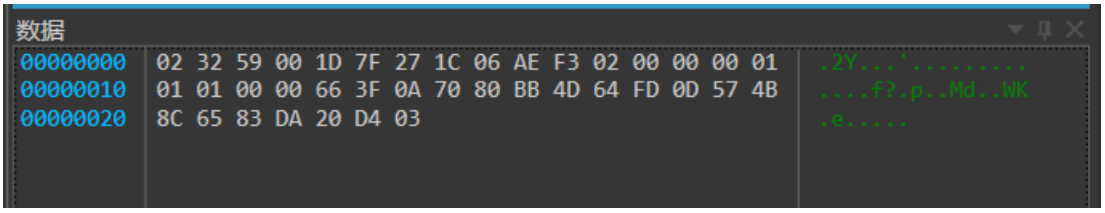


图 2.2-5 载入待测数据

在前面已经介绍过了，在执行代码时候，日志输出作为调试信息输出的重要功能，在编辑完成代码后，点击执行载入待测数据，就可以进行测试了。

如下图所示我们会发现，在输出窗口中有红色文字，并且调试没有找到 su()方法的定义，这其实就是说明我们当前函数库中并没有 su()这个方法，所以出现了错误，当我们点击红色部分，代码中的

```
node 数据= select(11,END) su();
```

被高亮选中了，在这段代码中果然存在 su() 的函数调用，所以出现了错误，此时我们看到结果窗口中，数据字段的结果是空的，这符合代码的执行结果，对于

其他字段为什么回执行成功，这是因为 MangoScript 遵循尽最大可能产生结果的设计初衷而来的。

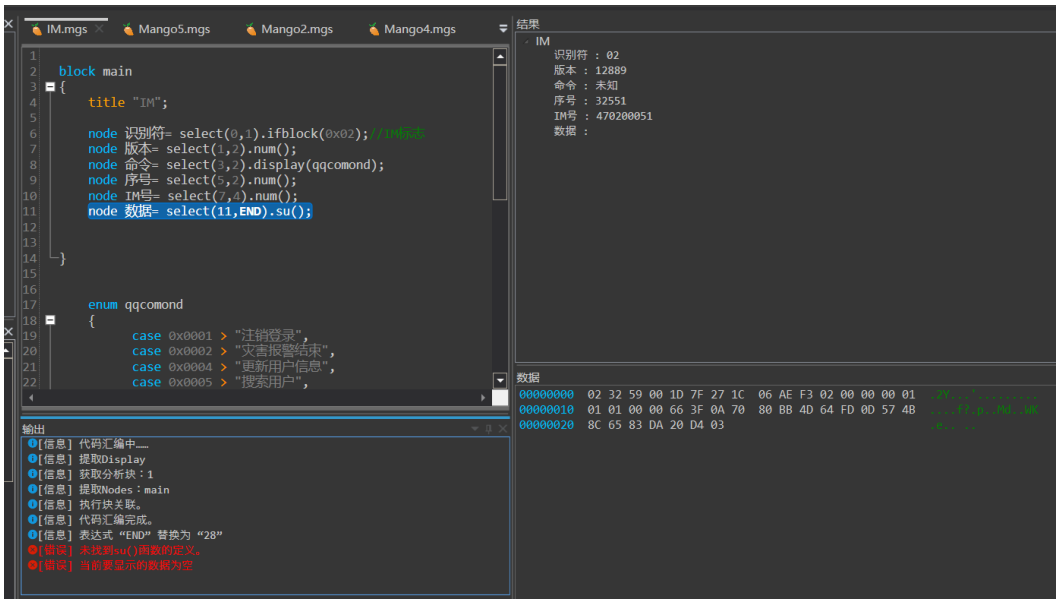


图 2.2-6 调试错误代码

这些可以被编辑器识别的错误都可以快速被快速的处理完成。通常我们会接触到各种输出结果与预期不一致的情况，这时候我们还需要对这个字段进行单独的分析，用于确定在数据解析的各个阶段输入输出的检测，这就用到了我们的 node 追踪功能。

Node 追踪，就是对每个 node 各个时期的值进行呈现。通过列表的方式列出字段在每个函数被执行完成以后得到值的样子。这样我们就可以精确的获取到每个时期数据是否正确，而不必为每个函数编写单独的测试代码。

为了代码调试方便，MgsEditor 增加了选定执行功能，即在选定某一段代码，单独执行。需要注意的是。这种方法只能选择 block 内部的代码，并且如果选定了多部分，或者参数中有前一个 node，则必须连同那个 node 一起被选中，然后点击执行就可以进行单独的测试了，在执行这段代码时候，MgsEditor 会自动在选择代码的外侧添加一个 block，所以在调试的时候可以把这种执行方案当成一个独立有效的 MangoScript 代码。

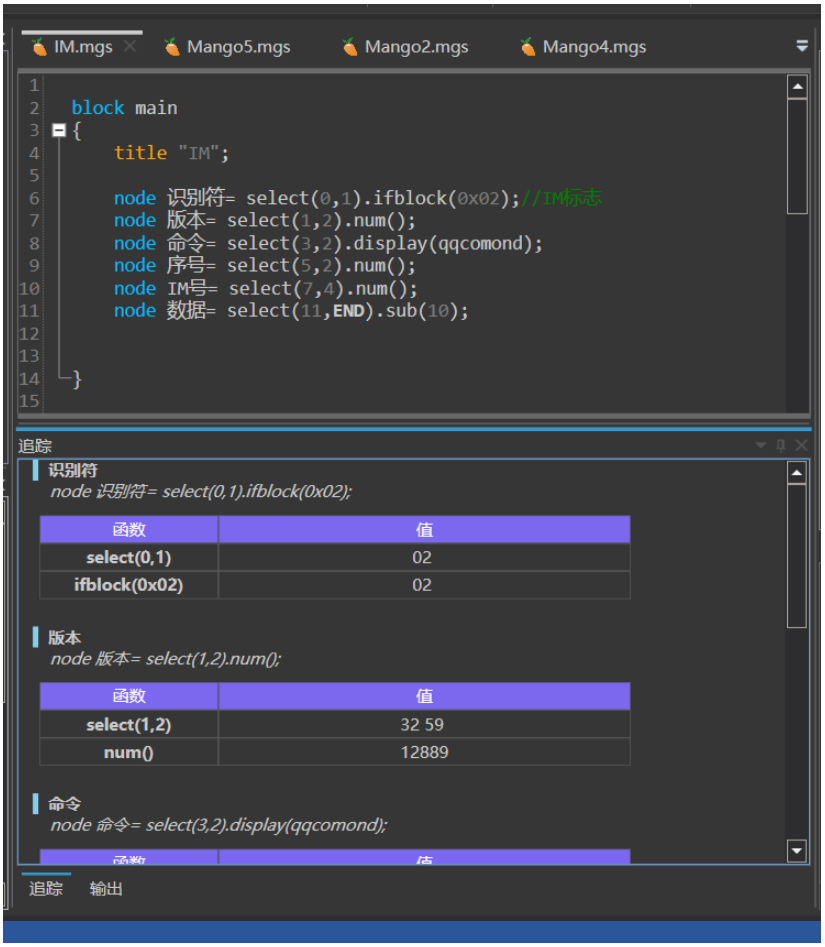


图 2.2-7 代码追踪功能

2.3. 脚本关联

当完成代码编写，并且全部测试准确以后就可以开始应用到 NetAnalyzer。因为 MangoScript 脚本体系正在处于测试与开发阶段。所以此处只开放基于传输协议的应用层协议的绑定关联

点击菜单栏中的**绑定端口**，

图 2.3-1 端口绑定按钮

就可以打开 NetAnalyzer 中的端口编辑器，

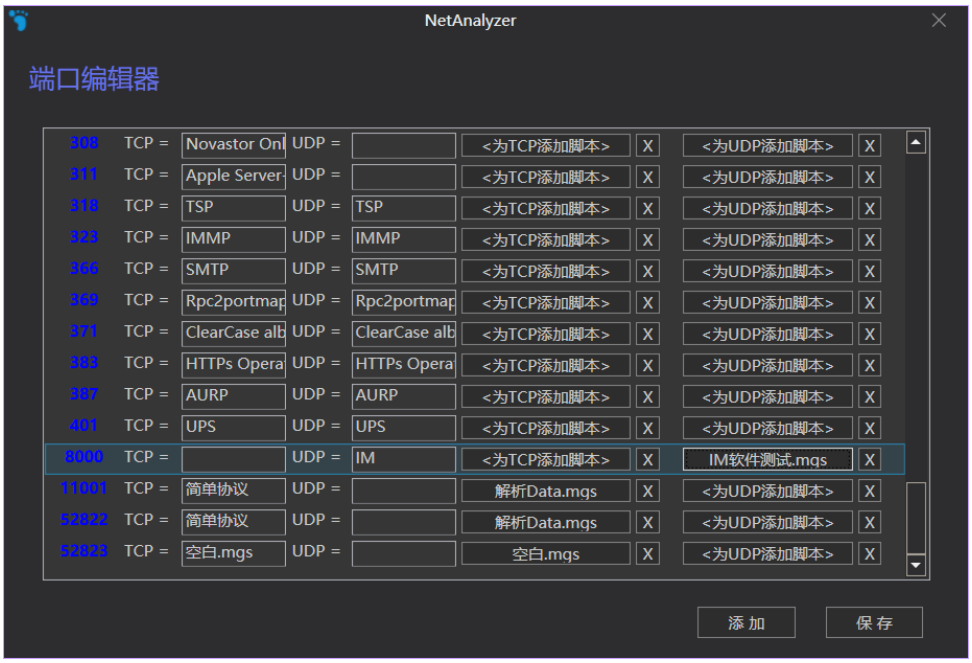


图 2.3-2 端口绑定设置

这个编辑器在介绍端口编辑的时候已经说过，此处就不在进行说明。在这里找到 8000 端口，如果不存在，则通过**添加**按钮添加对应的端口号，然后点击**<为 UDP 添加脚本>** 按钮，打开脚本选择窗口，

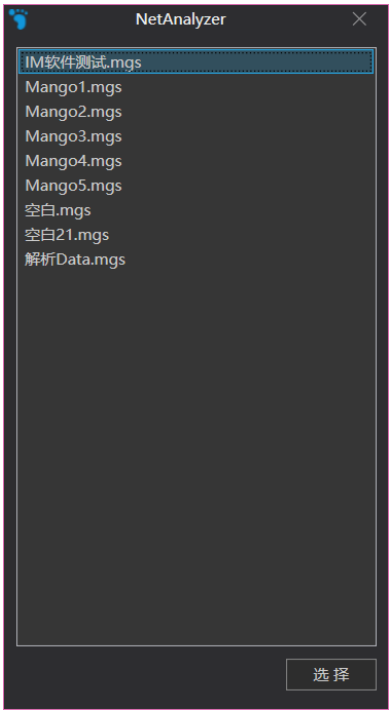


图 2.3-3 端口绑定脚本选择

这里的脚本与 MgsEditor 中代码库中的脚本完全一样，然后选择自己编辑完成的

脚本，点击选择，此时我们看到原来**<为 UDP 添加脚本>**的按钮文字已经变为对应脚本的名称，点击下面的保存就可以完成配置了。如果不想关联这个脚本，点击后面的取消关联按钮就可以取消关联。这里我们完成了针对于 IM 脚本的关联，我们可以开大 NetAnalyzer 进行测试了。

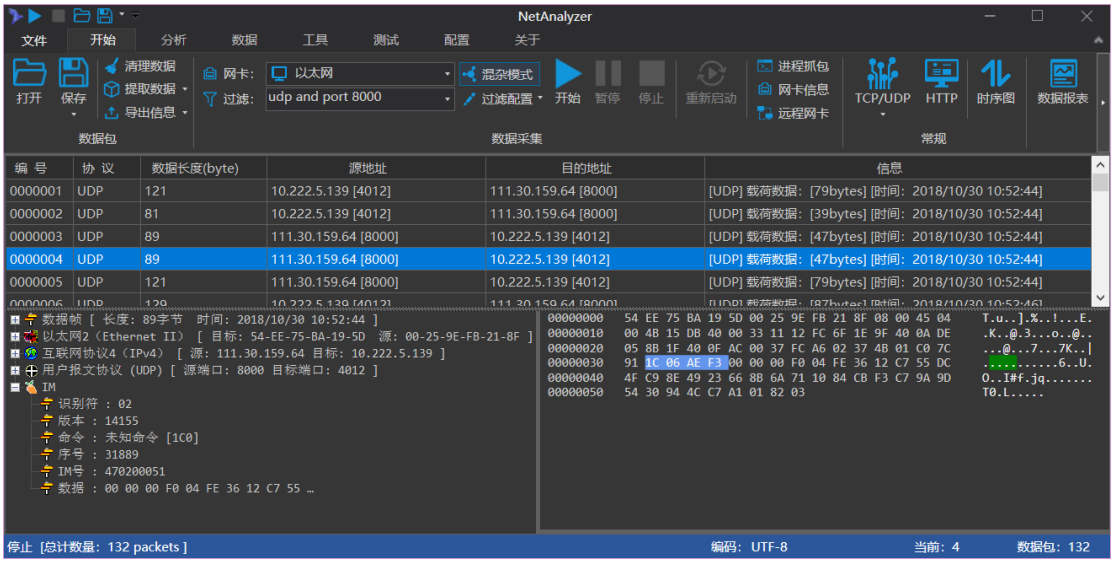


图 2.3-4 MangoScript 在 NetAnalyzer 中的使用

NetAnalyzer 中已经可以完成正常解析 IM 协议了。

至此对于 MangoScript 部分的内容已经完全介绍完毕了，MangoScript 还在刚起步阶段，目前所支持的解析还很微弱，而且语法规则也会随着版本的演进发生一些变化，欢迎阅读到这里的人提出意见和建议。

第二部分 NetAnalyzer 插件开发

上一部分提到的 MangoScript 是针对数据分析部分的扩展。而这部分是主要针对 NetAnalyzer 软件本身的扩展，

随着 NetAnalyzer 软件功能越来越多，代码结构也越来越复杂，然而面对层出不穷的个性化需求，却依旧显得无能无力。在以前的版本中只需要一到两个月就可以完成一个版本的更新，到现在一年都未必有一次更新，这除了墨云本身只是利用闲暇时间进行开发以外，更多的是面对更大的 NetAnalyzer 代码需要更多的时间去维护其正常运行不出现问题，这样在面对一些比较重大的功能时候，总是望而却步。

Lines by type

Type	Files	Size	Lines	Code	Code//Comm	Comm	Blank
.cs	2111	12233533	330060	215454	2210	62190	50206
.xaml	243	2877929	45133	41458	24	1316	2335

Lines

Total: 375193 lines

Type	Lines	Percentage(%)
Code	256912	68.47%
Code//Comment	2234	0.60%
Comment	63506	16.93%
Blank	52541	14.00%

2.1-1NetAnalyzer 代码量统计(2018-10-31)

所以在经过无数次被代码折磨以后，决定为 NetAnalyzer 加入插件功能，和一直以来支持墨云的网友一起使 NetAnalyzer 壮大起来。

1. NetAnalyzer 中的插件方案

NetAnalyzer 使用 C#开发，使用.NET4.0 平台版本，UI 部分基于 WPF 开发，整体 UI 使用 Fluent.Ribbon 框架，所以对插件开发的基本要求为开发环境要求安装.NET Framework4.0。

该部分对于读者只需要具备一定的.NET 开发基础，因此在后续的讲解过程中，都将默认读者为已经具备一定的.net 基础了。

NetAnalyzer 使用两种插件方案，基于.net 自身的方案和基于 CSharpScript 的方案

- 基于.net 的方案，生成的插件文件形式为*.dll 或*.exe，该方案主要优点就是可以基于 Visual Studio 进行快速开发，但缺点是需要针对新一版的 NetAnalyzer 需要重新引用编译以后才能使用(正在查找解决方案)。
- 基于 CSharpScript 的方案，该方案与 NetAnalyzer 接口交互使用*.cs 脚本方式，优点正是针对与上一中方案缺点而设计，只要在 NetAnalyzer 基本框架不变的情况下，完成后的 CSharpScript 脚本可以适用于各个 NetAnalyzer 版本，而且对于没有开发环境的情况下，也可以进行对开发工作。缺点在没有集成开发环境的支持下，对编程的硬性要求较高。

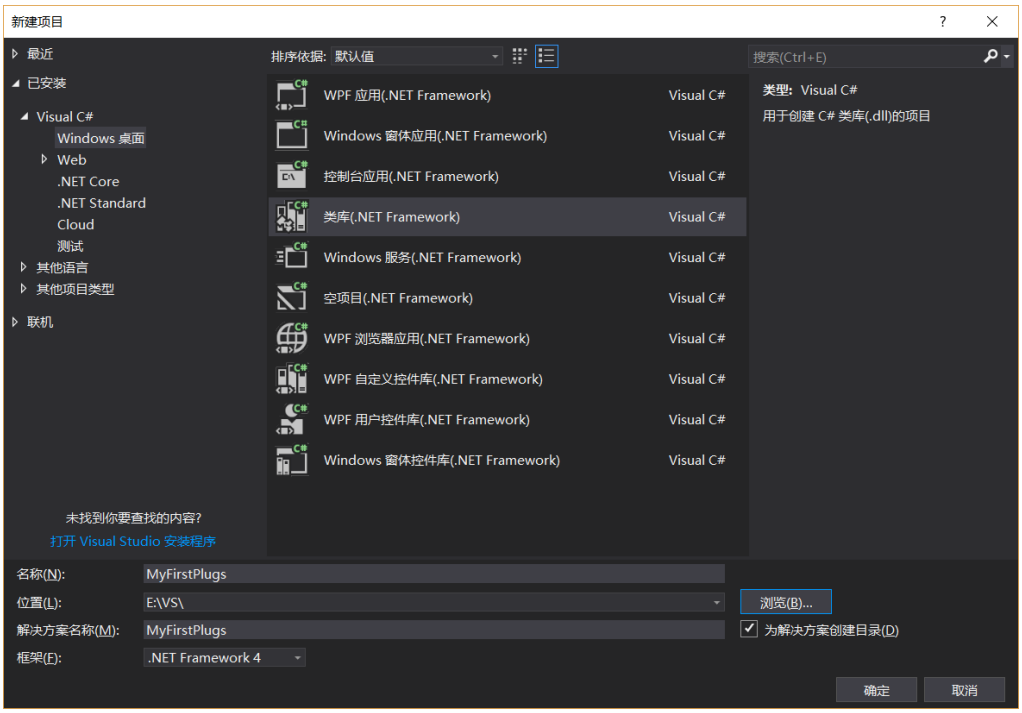
在进行插件开的过程中，并不需要严格遵循上面的方案的一种处理方式，在必要时可以合理搭配使用。

2. 开发第一个插件

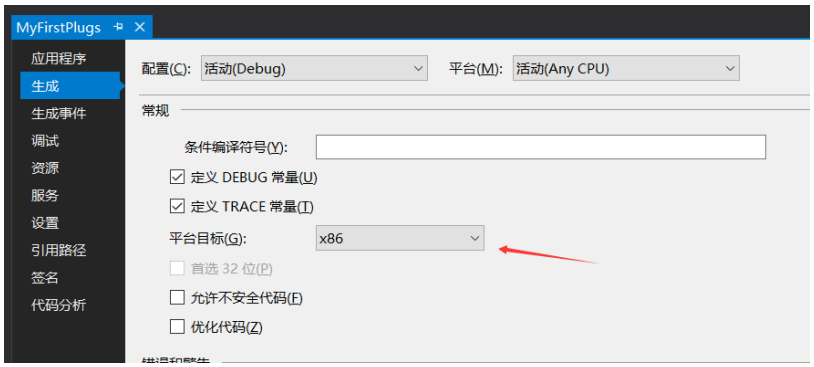
通过上面的空洞的说明，并不会让我们有多少想法。所以我们一起来写一个插件示例吧。在这个插件中，我们将会通过 NetAnalyzerAPI 获取到当前已经采集到的数据包和选中的数据包，并把内容数据呈现在我们自己的插件上面：

2.1. 工程建立与配置

首先启动 Visual Studio (简称 VS 下同)建立一个类库的程序，在这里我使用的 Visual Studio2017 如果你使用的是其他版本的 Visual Studio 请按对应的版本进行配置与开发，修改名称为 MyFirstPlugs,选择插件工程位置，最后我们需要注意一点就是框架部分需要修改为：.NET Framework4.0 ，点击确定后我们的插件工程就建立起来了。



2.1-2 建立 MyFirstPlugs 工程



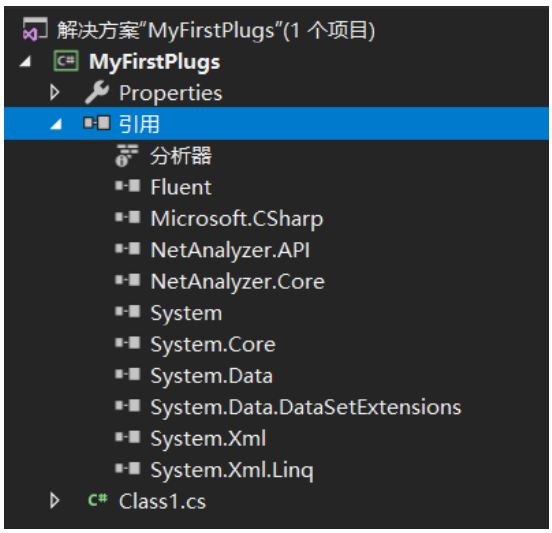
2.1-3 插件配置目标平台为 x86

然后右击工程属性->生成->平台目标 设置为 x86

2.2. 引用 NetAnalyzer 接口

接下来我们需要引用一些 NetAnalyzer 中的框架类库来完成插件的接入。

右击**引用->添加引用** ，打开添加引用对话框，选择**浏览**按钮，然后在 NetAnalyzer 安装目录下，64 位系统为(D:\Program Files (x86)\Twzy\NetAnalyzer 32 位的系统为 (D:\Program Files\Twzy\NetAnalyzer，盘符请根据自己实际情况进行设置)选择 NetAnalyzer.Core.dll、NetAnalyzer.API.dll、 Fluent.dll 三个 DLL 文件，然后点击添加，完成对于 NetAnalyzer 中的引用。



2.2-1 引用 API 类库

里先说明一下三个 DLL 类库的作用：

- NetAnalyzer.Core.dll　NetAnalyzer 中的核心库，软件中的网卡获取、监听、过滤表达式配置、数据包存储、分析、统计、tcp 重组、http 数据流分析、MangoScript 解析等各种功能都在这个库里面完成。
- NetAnalyzer.API.dll　为 NetAnalyzer 插件提供各种 API 的类库。
- Fluent.dll NetAnalyzer 中的 UI 框架，如果你的插件需要在 NetAnalyzer 有菜单内容呈现，那就必须引用这个类库。

通过引用这些类库，我们就可以完成现在的开发工作了。接下来我们就开始编码吧。

2.3. 编码与测试

首先我们建立一个类，并让其继承 NetAnalyzer.API.ExBase 并且实现 NetAnalyzer.API.IPugins 接口。

APIHost.cs

MyFirstPlugs　MyFirstPlugs.APIHost

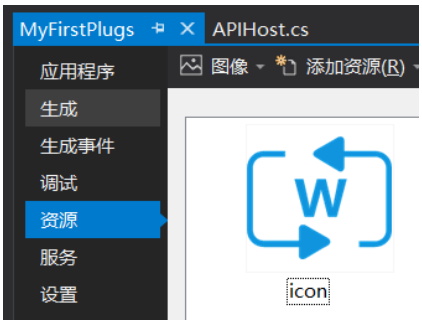
```
using System.Text;

namespace MyFirstPlugs
{
    public class APIHost :NetAnalyzer.API.ExBase, NetAnalyzer.API.IPlugins
    {
        #region 插件信息

        //插件名称
        public string PluginName { get { return "MyFirstPlugs"; } }
        //版本
        public string Version { get { return "1.0"; } }
        //图标
        public System.Windows.Media.Imaging.BitmapSource Icon
        {
            get
            {
                return NetAnalyzer.API.ExTools.BitmapToBitmapSource(Properties.Resources.icon);
            }
        }
        //作者
        public string PluginAuthor {  get  { return "墨云";  } }
        //网站
        public string PluginUrl { get { return "http://twzy.sinaapp.com/";  } }
        //插件描述
        public string PluginDescription { get { return "这是一个示例插件";  } }

        #endregion
    }
}
```

2.3-1 插件信息配置

首先我们需要实现 IPugins 的接口，这部分内容主要是插件注册需要使用的，也就是通过 NetAnalyzer 的插件插件管理是看到信息的那些信息。

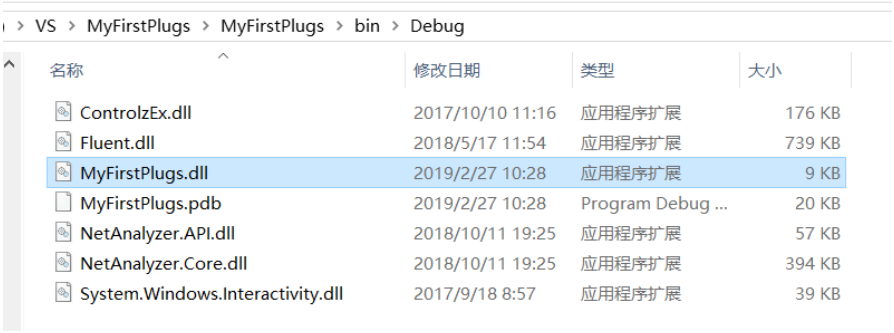
这里需要注意一下对于插件图标的使用引用。

首先在工程资源里面添加需要的图片文件*.jpg 、*.png 都可以，然后通过 API 所带的扩展工具类 ExTools 将资源转换为需要的数据类型返回即可。



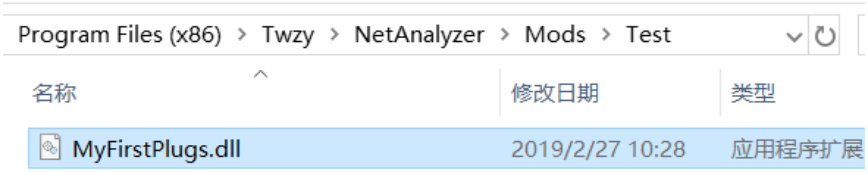
2.3-2 配置插件图标

通过上面的代码与配置我们就完成了一个最基本的插件，虽然没有提供如何功能。这时候是不是想要看看插件的样子呢？通过编译可以在 debug 目录下可以看生成的 MyFirstPlugs.dll。还有一些到其他 dll，这些可以忽略。



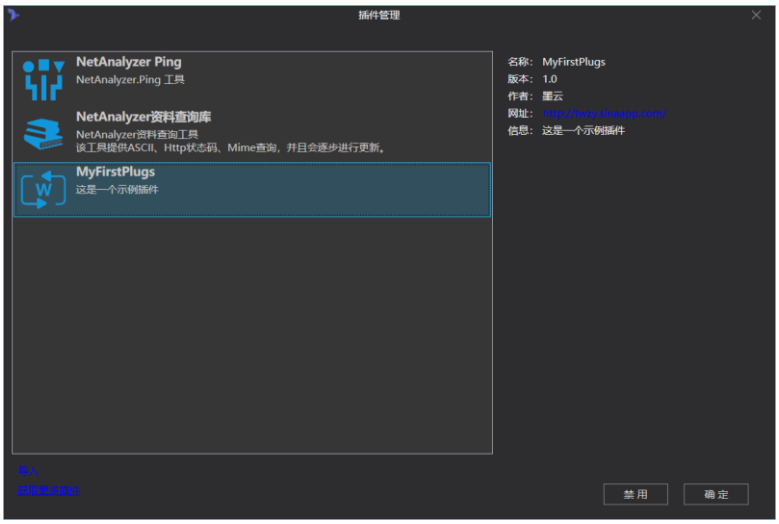
2.3-3 生成插件文件

在 NetAnalyzer 安装目录下，有个 Mods 的文件夹，在文件夹中我们新建一个文件夹命名为 Test，然后把刚生成的 MyFirstPlugs.dll 复制到 Test 目录下。



2.3-4 部署插件

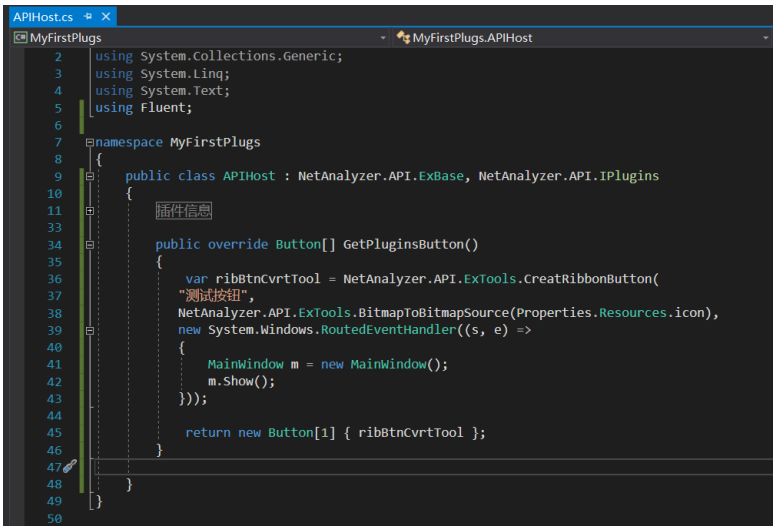
启动 NetAnalyzer，在工具->管理->插件管理就可以看到我们的插件了



2.3-5 运行插件

正如前面说到的一样，这个插件并没有什么功能，那接下来我们就可以进行功能开发了。

在我们建立的工程中先添加一个窗体 MainWindow，用于呈现界面内容；接下来利用工具类 ExTools 中提供的 CreatRibbonButton 方法创建一个按钮，通过这个按钮将界面打开，这个按钮命名为“测试按钮”，这个按钮将会出现在 NetAnalyzer 工具菜单下。



2.3-6 添加功能代码

创建“测试按钮”代码

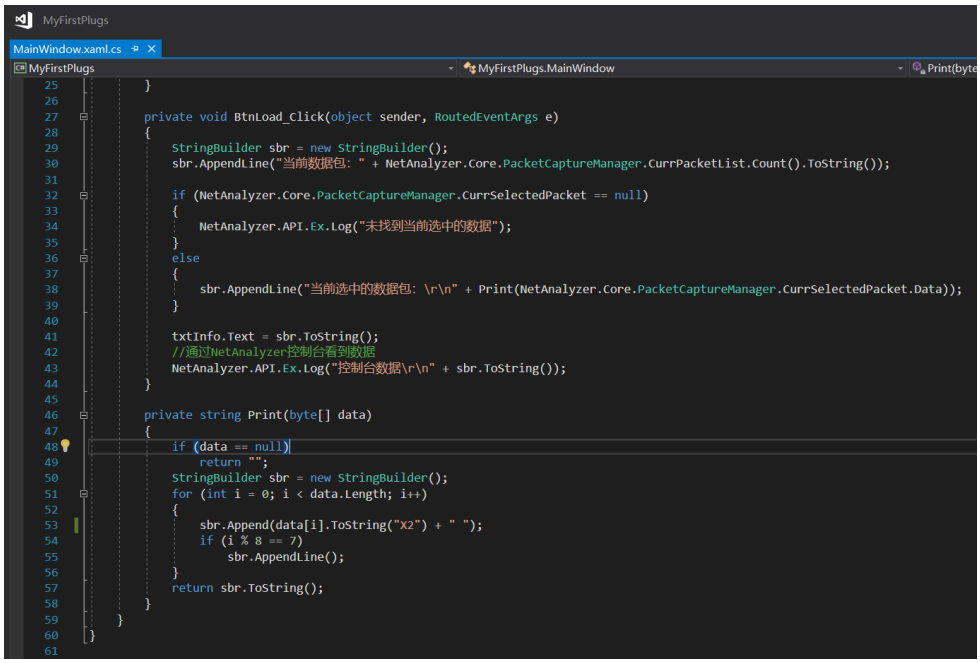
接下来我们就可以完成对 MainWindow 的代码设计了，前面提到过，我们将要在这个示例中呈现当前状态下 NetAnalyzer 获取到的数据包条数以及当前选中的数据包的内容。

下面是 MainWindow 的窗口代码，这里使用了 WPF 方式，当然也可以使用 WinForm 形式，具体内容不再这里讨论。在该段代码中我们定义了一个按钮一个文本框，通过点击按钮在文本框中显示我们需要的信息

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="40"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <Button Name="btnLoad" Content="载入" Grid.Row="0" Click="BtnLoad_Click"/>
    <TextBox Name="txtInfo" Grid.Row="1" AcceptsReturn="True" IsReadOnly="True" />

</Grid>
```

2.3-7 界面代码

接下来需要引用 NetAnalyzer.pcap.dll 文件，该文件为抓包的核心部件，在内部定义了采集数据包的基础模型，因为这里需要使用该模型，所以需要添加对其的引用。



2.3-8 功能代码

在 MainWindow 的后台代码中定义了连个方法,BtnLoad_Click 和 Print 方法，第二个则是一个把字节转为字符串的工具方法，不做重点讨论。

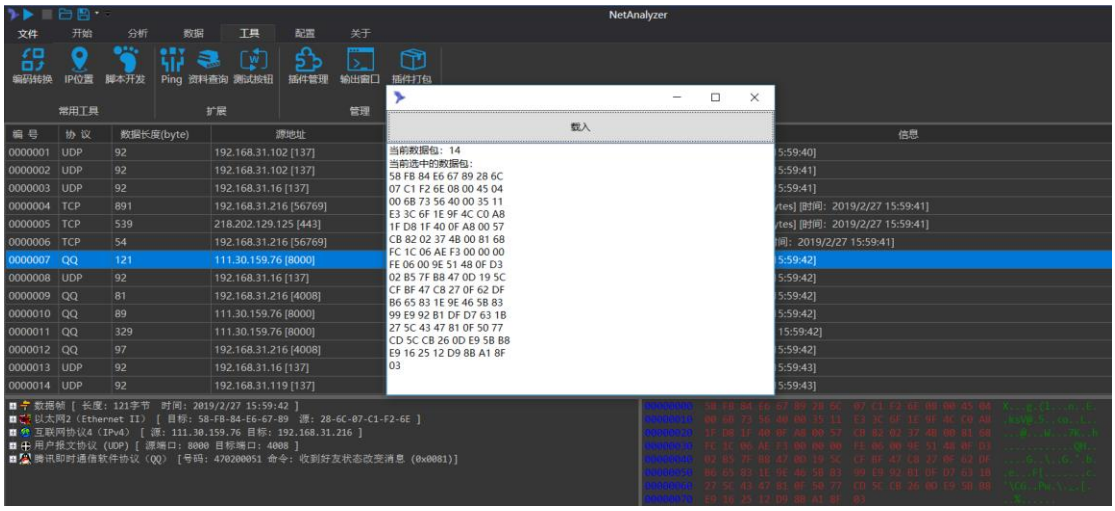
这里主要是说一下第一个方法 BtnLoad_Click，在这个方法中我们用到了三个 NetAnalyzer 提供的接口。

- NetAnalyzer.Core.PacketCaptureManager.CurrPacketList NetAnalyzer 获取到的数据列表
- NetAnalyzer.Core.PacketCaptureManager.CurrSelectedPacket NetAnalyzer 当前选中的数据包
- NetAnalyzer.API.Ex.Log("xxx") NetAnalyzer 插件日志信息输出,(通过工具->管理->输出窗口就可以看得到信息)

完成上面的代码编写，然后编译完成，就完成了我们的第一个插件的开发了，然后开始测试这个插件吧。通过前面提到方法把 MyFirstPlugs.dll 复制到 NetAnalyzer 安装目录中 Mods 下面的自己建立的文件夹下

2.3-9 插件运行按钮

重新启动 NetAnalyzer 我们就可以在工具菜单下看到**测试按钮**这个功能了，接下来我们启动抓包获取到一些网络数据，并且选中一个数据包后，点击**测试按钮**，然后点击输入，此时就可以在下面的窗口中看我们想要的数据了。



2.3-10 插件执行结果

2.4. 插件控制台应用

在代码中我们使用了 log 来输出一些调试数据，我们可以通过工具->管理->输出窗口就可以看得到信息。

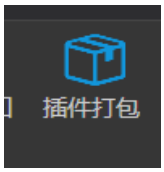
2.4-1 启动插件控制台

2.4-2 控制台信息

注意 在这里我们在编译完成后有很多 dll 文件，但是我们只是用了插件代码编译完成的那个文件放置到插件文件下，并没有放其他的 dll，是因为 NetAnalyzer 早已经再内存中载入了这些 dll，但是如果插件使用了三方的 dll 文件则需要一并复制到插件文件夹下。

2.5. 插件打包与安装

至此，我们就完成了一个简单插件的开发，接下来就是分享了，这里可以使用我们在上文提到方案。还可以通过生成一个安装包，由 NetAnalyzer 提供的安装程序自动安装。

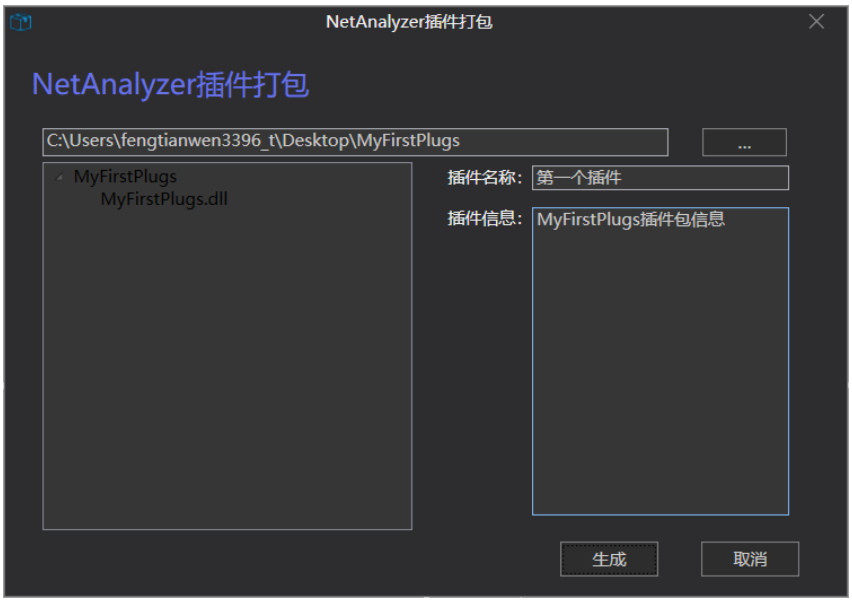


2.5-1 插件打包

打开工具->管理->插件打包就可以帮助我们实现这个功能

先准备一个文件夹，这里继续使用 MyFirstPlugs 作为文件夹的名称，然后将我们编译后的 MyFirstPlugs.dll 复制到这个目录下。

打开插件打包对话框。点击 … 按钮选中文件夹，输入对应的插件名称（插件名称为最终的文件名称，请使用符合 windows 路径规则名称）和插件信息，点击生成就可以生成对应插件安装包了。



2.5-2 插件打包界面

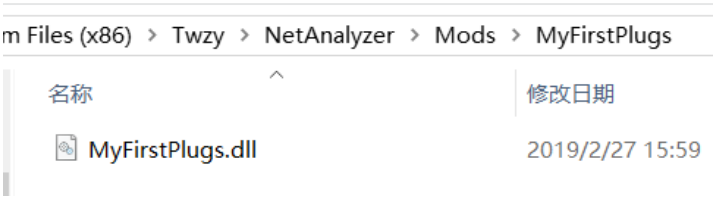
2.5-3 生成插件包

这样就完成了安装包的生成，扩展名为 *.ntpk。

双击该安装包

2.5-4 插件安装

打开安装界面，然后点击安装就完成了对插件的安装了。



2.5-5 安装结果

本章通过一个简单的例子讲述了 NetAnalyzer 插件的开发生成测试等各个环节的内容。在这个过程中我们也注意到一些关于 API 的操作之类的内容，这部分将会在下一节进行着重说明。

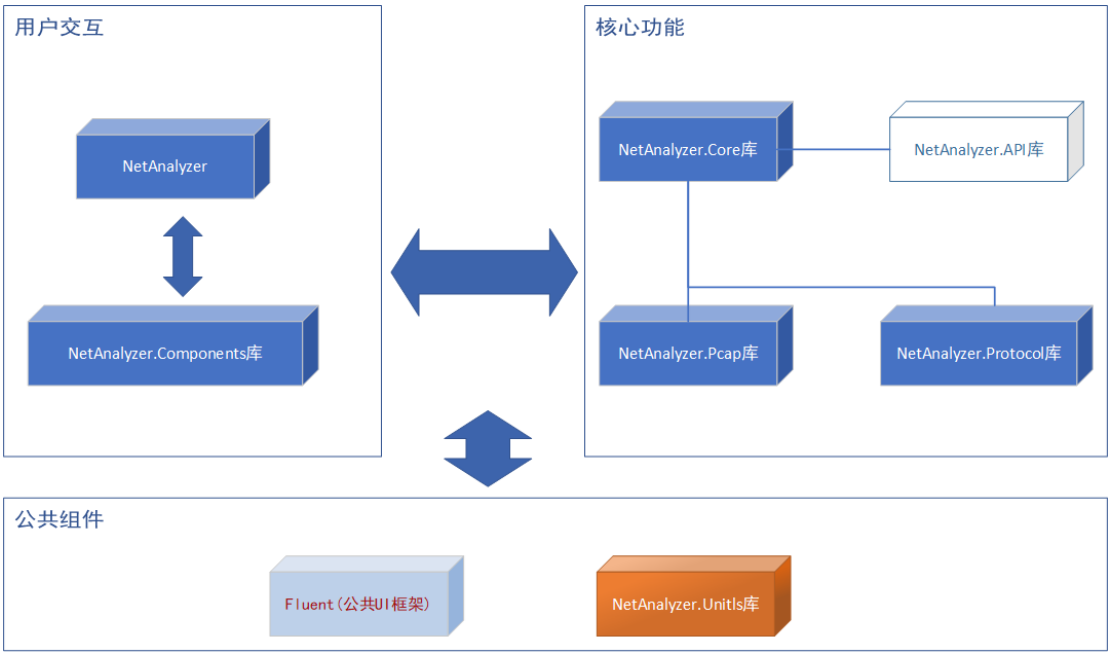
3. NetAnalyzer 框架与 API

通过前面关于 NetAnalyzer 插件的开发我们已经可以做一个简单的程序了，但是如果只是能简单使用数据包列表或选中的数据包，并不能满足对 NetAnalyzer 个性化扩展，所以本节将会大量的提供 NetAnalyzer 框架和 API 相关的内容。

3.1. NetAnalyzer 框架

通过前面的讲解，我们可以熟练的使用 NetAnalyzer 了，并且因为插件开发部分的讲解，已经对 NetAnalyzer 框架有一些特定的了解，而本节将会通过代码角度，以开发者的眼光对 NetAnalyzer 进行说明。

NetAnalyzer 开发之初就考虑到将来会集成插件扩展的功能，所以再后来的开发过程中有意识的将一些核心功能集中的一个库里面，并且使用开放性的开发模式将这些接口暴露出来。



3.1-1 NetAnalyzer 框架

在 NetAnalyzer 框架主要分为三个大部分：

- 用户交互；

  用户交互部分泛指与 UI 操作相关的各种呈现与触发方式，包含主窗口与各种组件窗口都在这里定义，NetAnalyzer 程序集包含了主窗口的 UI 设计，而 Components 库则提供了各个弹出窗口的功能实现。

- 核心功能；

  核心部分是整个 NetAnalyzer 最重要的部分，作为协议分析工具，最主要的

功能就是抓包和数据分析，在核心功能中 Pcap 库完成了对 Winpcap 在.NET 层面的托管代码封装，通过该库可以获取到主机网卡信息，控制网卡进行数据采集等；Protocol 是 NetAnalyzer 的数据分析模型库，该库定义了大量的协议模型，用于对采集到的数据进行匹配与分析；API 库，正如前面示例所提到的，该库是专门用于处理插件相关的类库；Core 库负责整个 NetAnalyzer 中的数据调度和程序运转。

- 公共组建

  公共组件提供了 NetAnalyzer 中的常用的公共工具类。

3.2. NetAnalyzer.API 库

在上一节的插件开发中简单介绍了 NetAnalyzer.API 类库（简称 API 库，下同）的信息，在这一节中将会进行详细的说明。

IPlugins 接口

IPlugins 作为 NetAnalyzer 插件注册类，主要用于显示与插件相关的基本信息，其定义结构如下。

```
namespace NetAnalyzer.API
{
    public interface IPlugins
    {
        #region 程序集信息

        /// <summary>
        /// 插件名称
        /// </summary>
        string PluginName { get; }

        /// <summary>
        /// 版本信息
```

```
        /// </summary>
        string Version { get; }

        /// <summary>
        /// 插件图标（*.jpg,*.png,*.bmp 格式）
        /// </summary>
        BitmapSource Icon { get; }

        /// <summary>
        /// 插件作者姓名
        /// </summary>
        string PluginAuthor { get; }

        /// <summary>
        /// 插件地址
        /// </summary>
        string PluginUrl { get; }
        /// <summary>
        /// 插件信息说明
        /// </summary>
        string PluginDescription { get; }
        #endregion

    }
}
```

ExBase 类

插件交互基础类，提供用于插件接入类的基本的初始接口方法，对于插件接口类，无论是基于.NET 程序集还是基于 CSharpScript 都需要继承此类。

string ModsDir 当前插件文件所在的文件夹。

virtual void OnInite() 当插件被启动时执行的方法，通过该方法，可以对插件进行运行开始前的初始化操作。比如对现有 NetAnalyzerUI 进行调整等。

virtual Fluent.Button[] GetPluginsButton() 快捷方法，可以通过该方法在 NetAnalyzer 工具菜单的扩展组面板中添加一组按钮。

virtual Fluent.RibbonGroupBox[] GetPluginsGroups() 快捷方法，可以通过该方法在 NetAnalyzer 工具菜单添加一组扩展面板。

virtual ToolStripMenuItem[] GetGridRightMenu() 快捷方法，可以通过该方法在 NetAnalyzer 数据包列表的右击菜单中添加一组菜单选项。

virtual ToolStripMenuItem[] GetTreeRightMenu() 快捷方法，可以通过该方法在 NetAnalyzer 数据包分析树面板的右击菜单中添加一组菜单选项。

virtual Fluent.Button[] GetHexCvrtMenu() 快捷方法，可以通过该方法在 NetAnalyzer 分析菜单的扩展菜单下添加一组针对于十六进制面板中数据的转换方法。

virtual System.Windows.Controls.MenuItem[] GetHexRightMenu() 快捷方法，可以通过该方法在 NetAnalyzer 十六进制面板的右击菜单中添加一组菜单选项。

virtual Fluent.RibbonTabItem[] GetRibbonTab() 快捷方法，可以通过该方法在 NetAnalyzer 菜单栏添加一组菜单。

Ex 类

插件扩展类，提供需要快速访问的 UI 成员，以及控制台日志输出方法。所有成员通过静态成员方式访问，UI 部分提供了对 NetAnalyzer 所有菜单的访问权限，你可以自由添加或修改这些 UI 成员，其中大部分 UI 菜单使用 Fluent 方案，可以参考 Fluent.Ribbon 项目提供的方案，如果不想使用其他功能，可以对 UI 进行
隐藏，因为删除这些菜单可能引起不可以预知的错误。

Fluent.Ribbon RibbonHost NetAnalyzer 提供的整个 UI 菜单，包含对“文件”，“开始”，“数据”，“分析”，“工具”，“配置”等菜单面板的的集中访问容器框架。

Fluent.RibbonTabItem RibTabStart Fluent 开始菜单面板

Fluent.RibbonTabItem RibTabAnalyise Fluent 分析菜单面板

Fluent.RibbonTabItem RibTabData Fluent 数据菜单面板

Fluent.RibbonTabItem RibTabTools Fluent 工具菜单面板

Fluent.RibbonTabItem RibTabConfig Fluent 配置菜单面板

ExTools 类

插件交互工具类，为插件接口类提供了一些可以快速创建 UI 交互组件的方法，包含对菜单按钮，右击菜单选项，图标等一系列的方法，后期还会根据用户的实际使用体验增加更多的功能。

static ToolStripMenuItem CreatRightItem(string itemName, EventHandler funClick = null)创建 WinForm 右击菜单，通过该方法可以向数据包列表或数据包分析树的右击菜单添加选项。

```
ExTools.CreatRightItem(""菜单 1"",new EventHandler((s,e)=>{
    //ToDo 单击事件
}));
```

static ToolStripMenuItem CreatRightItem(string itemName, Image img, EventHandler funClick = null) 创建 WinForm 右击菜单，通过该方法可以向数据包列表或数据包分析树的右击菜单添加选项。

```
ExTools.CreatRightItem(""菜单 1"",null,new EventHandler((s,e)=>{
    //ToDo 单击事件
}));
```

static Fluent.Button CreatRibbonButton(string buttonName, BitmapSource img, RoutedEventHandler funClick = null) 创建一个基于 Fluent.Ribbon 中的一个常规按钮。

```
ExTools.CreatRibbonButton(""按钮"",null,new System.Windows.RoutedEventHandler(((s,e)=>{
    //ToDo 单击事件
}));
```

static Fluent.RibbonTabItem CreatRibbonTab(string tabName) 创建一个基于 Fluent 中的一个 TabItem。

```
ExTools.CreatRibbonTab(""tab1"");
```

static Fluent.RibbonGroupBox CreatRibbonGroup(string groupName) 创建一个基于 Fluent.Ribbon 菜单面板中的组

```
ExTools.CreatRibbonGroup(""Group1"");
```

static BitmapSource BitmapToBitmapSource(Bitmap bmap) 将 GDI+图片转为 WPF 中的图片，主要用于菜单按钮图标的生成

static BitmapSource GetBitmapSource(string file) 将图片文件转为 WPF 中的图片，主要用于菜单按钮图标的生成。

3.3. NetAnalyzer.Core 库

NetAnalyzer.Core 库（简称 Core 库，下同）是整个 NetAnalyzer 的中核心库，该库提供了数据采集、存储、分析、统计、还原等多种功能，通过该库可以完成对 NetAnalyzer 内核绝大部分的控制。首先看看 Core 库中最核心的数据采集类。

PacketCaptureManager 类

数据包采集以及管理核心类，该类包含了对网卡，数据包、文件读写等相关的全部控制方法。通过该方法，可以获取到当前采集的数据包列表，选中的数据包等

信息。

static ICaptureDevice CurrDevice 网卡属性，在使用前请加载 NetAnalyzer.pcap.dll，通过该属性可以获取当当前选择的网卡名称，驱动名称、MAC 地址、IP 地址等信息。

static DeviceMode CurrMode 抓包采集模式属性，在使用前请加载 NetAnalyzer.pcap.dll，通过该属性可以获取或设置当前数据采集方式，通常我们使用混杂模式，关于混杂模式的说明，请见《使用手册 一》中的内容。

static List<RawCapture> CurrPacketList 数据包列表，在使用前请加载 NetAnalyzer.pcap.dll，当前 NetAnalyzer 内存中缓存的基础数据包，这些数据包可能来自网卡采集或来自文件加载。使用最基本的二进制方式存储。

static bool IsSaved 指示当前内存中的数据包是否已经保存为文件，该属性主要用于保存文件判断之用。

static Window Host NetAnalyzer 主界面，WPF 中的 Window 类型，调用的时候，请注意线程是否安全。

static DataGridView grdProtocol NetAnalyzer 界面中的数据包列表控件，在使用时，注意线程安全。

static Unitls.WPFHexEditor hexEditor NetAnalyzer 界面中的数据包十六进制显示控件，在使用时，注意线程安全。

static System.Windows.Forms.TreeView treeProtocol NetAnalyzer 界面中的数据包分析树控件，在使用时，注意线程安全。

static string Filter 过滤表达式属性，获取或设置在记性数据采集时候的过滤表达式，使用时候请保证过滤表达式安全可用，否则或产生网卡启动采集异常。

static string File 文件路径，存储数据包的文件路径，如果未保存文件则为空字符串。

static OptionHandlerStr ShowStatus 状态栏信息提示委托属性，向状态栏输出文本

信息。

static delProgressBar ShowProgerssBar 文件加载进度委托属性，用于输出文件加载进度信息。

static OptionHandlerBool UICaptureConfig UI 可用性控制属性委托，通过该属性可以控制在某些状态下 UI 可用状态，如进行抓包或文件加载时候的 UI 可用性。

static bool OpenFileCancel 获取在打开文件时候，是否进行了取消操作。

static Pcap.RawCapture CurrSelectedPacket 当前 NetAnalyzer 选择要分析的数据包，使用时注意判断是否为空。

static event PacketArrivalHandler PacketArrival 在通过网卡获取数据的时候，当获取到数据触发该事件。

static bool IsPacketUsedDefault 当数据包到来的时候是否在系统中使用，在某些情况下，插件开发者只是自己需要数据包而不需要进入 NetAnalyer,默认为 true。

static string CurrDevName （只读）当前选择网卡的名称。

static OptionHandler ShowTotalCountHandler 显示数据包数量委托属性，用于向主界面输出当前的数据包数量。

static void ResetFiled() 重置所有字段，用于进行数据采集前核心部分字段的初始化操作。

static bool ClearData() 清空数据表，重置所有核心部分的采集字段，该方法需要确认是否将未保存的数据包进行文件保存。

static void SetDevice(int devIndex) 设定需要进行抓包的网卡，通过索引进行传递设定。

static List<DeviceItem> GetDeviceNameList() 获取网卡信息列表，包含本地网卡和通过 winpcap 获取到的远程网卡信息。

static CaptureDeviceList GetDeviceList() 获取本地网卡信息。

static string GetDeviceInfo() 获取当前选定网卡的全部描述性信息。

static string GetDeviceInfo(int devIndex) 通过索引获取指定网卡的全部描述性信息。

static string GetDeviceInfo(ICaptureDevice dev) 通过指定实体网卡获取该网卡的全部描述性信息。

static bool Start(int devIndex) 通过指定网卡索引的方式启动抓包功能。

static bool Start() 在已经制定好网卡的情况下，启动抓包功能。

static void Stop() 停止抓包功能。

static void ReStar() 重新启动抓包功能。

static void Pause() 暂停抓包。

static void ReCapture() 暂停后恢复重新抓包的功能。

static void PacketFileRead(string filePath) 读取指定数据包文件到内存中数据包列表。文件必须为可支持的 pcap 文件。否则可能引起不可预知的错误。

static void PackeFileSave(string filePath) 将当前内存中的数据包保存为指定路径的文件。

PacketSendManager 类

该类提供了基于 Winpcap 数据包发送相关的功能，可以通过该类模拟进行伪造数据包发送，在 NetAnalyzer 并没有提供这部分功能，所以具体效果如何还需要进一步验证，如果在使用过程中产生程序错误请联系墨云，另外请注意在法律规定的范围内使用该功能，墨云不会承担因使用该功能而产生的任何责任。

static void SendOne(Pcap.ICaptureDevice dev, byte[] data) 打开指定网卡，并且发送已经准备好的数据。

static void SendAsyn(Pcap.ICaptureDevice dev, List<byte[]> data, int timeSpan) 通过

异步方式发送多条数据。

static event SendStartHandler SendStart 异步开始发送的委托通知属性。

static event SendCompleteHandler SendCompleted 异步结束发送的委托通知属性。

PacketStatisticsManager 类

统计类，该类完成对捕获到的数据报文的统计，有 TCP,UDP、ARP、ICMP、Other 五种类型的总数据大小与数量

static string OSInfo 当前系统相关的信息。

static string Device 网卡信息。

static string Filter 过滤表达式。

static string BitInfo 系统位数,x86 或 x64。

static int TcpPacket TCP 数据包总字节大小。

static int TcpCount TCP 数据包个数。

static int UdpPacket UDP 数据包总字节大小。

static int UdpCount UDP 数据包个数。

static int TcpErrorCount TCP 校验和错误的数据包。

static int UdpErrorCount UDP 校验和错误的数据。

static int IcmpPacket ICMP 数据报文长度。

static int IcmpCount ICMP 数据包数量。

static int igmpPacket IGMP 报文长度。

static int igmpCount IGMP 数据包数量。

static int OtheronIPPacket 基于 IP 的其他数据包。

static int OtheronIPCount 基于 IP 的其他数据包数量。

static int ArpPacket ARP 数据报文长度。

static int ArpCount ARP 数据包个数。

static int OtherPakcet 其他数据报文长度。

static int OtherCount 其他数据包数量。

static void Clear() 清除当前统计的数量。

static float[] ToPacketArry() 将数据量转换为数组。

static int TotalLength 计算当前总报文长度。

static int ToatalCount 计算当前总数据包数量。

static int ToIPLength 获取 IP 数据包的大小。

static int ToIPCount IP 报文数量。

static List<StatisticsItem> GetPacketCountInfo() 各个协议数据包数量个数。

static List<PacketTimeItem> GetPacketLine() 获取各个时间段获取到的数据包数量。

FilterManager 类

过滤表达式管理类，通过该类可以简单完成对过滤表达式记录的管理。

static List<string> FilterList 过滤表达式记录表，需要判空

static void AddOrMoveFrist(string filter) 向过滤表达式列表添加新的过滤表达式，如果该表达式存在，则自动移动到第一位。

static void Del(string filter) 删除过滤表达式。

static void SaveFilterList() 保存过滤表达式。

TraceCore.TraceManager 类

MangoScript 运行与管理类，通过该类可以运行与调试 MangoScript 脚本。

static event TraceLogEventhandler EventTraceLog 脚本运行日志输出事件。

static ScriptAssem AssemCode(string code) 代码调试，作为前期代码分析用。

static TraceNode Analyse(byte[] data, ScriptAssem ass) 通过汇编后的模型对数据进行分析。

对于 Core 库除了上面的内容外还存在大量的可操作类和功能。这里限于篇幅将不做过多的说明，如有实际需要请直接询问墨云。

3.4. NetAnalyzer.Unitls 库

NetAnalyzer.Unitls 库（简称 Unitls 库，下同）该库为 NetAnalyzer 提供大量的工具类，包含数据、格式转换工具，该部分将着重说明一些工具方法的使用说明，以方便插件开发可以快速进行插件开发。

Tools 类

该类提供了 HTML 转为 PDF、文本文件编码类型、序列化、加解密、二维码生成等多种功能。

static void HtmlToPdf(string filePath, string html, bool isOrientation = false, int horMarg = 50, int verMarg = 60) 将 HTML 页面输出为 PDF 格式，在该方法中，传入输出文件路径，html 页面字符串就可以生成对应的文件，同时还可以指定 pdf 页面方向、同时还可以指定水平或垂直方向的边距。本方法基于 Pechkin.dll 组件完成。

static System.Text.Encoding GetFileType(string FILE_NAME) 给定文件的路径，读取文件的二进制数据，判断文件的编码类型。

static System.Text.Encoding GetFileType(FileStream fs) 通过给定的文件流，判断文件

的编码类型 。

static void WriteByXML<T>(T p, string path) 以 XML 文件方式保存序列化信息。

static T ReadByXML<T>(string path) 通过读取 XML 文件获取序列化信息。

static void WriteByBinary<T>(T p, string path) 通过读取二进制文件获取序列化信息。该方法区别于 XML 方式，可以保存一些私有字段的数据。

static T ReadByBinary<T>(string path) 通过读取二进制文件获取序列化信息。可以获取私有字段的数据。

static string Encrypt(string src) 将字符串通过 DES 算法加密后以 Base64 编码的方式返回。

static string Decrypt(string src) 解密字符串，密文为通过上面加密后的 Base64 字符串。

static void SnapShotPNG(UIElement source, string filePath) WPF 中的 UI 元素截图保存为文件。

static byte[] SnapShotPNGToStream(UIElement source) WPF 中的 UI 元素截图并且返回为图片字节数组。

static BitmapSource BitmapToBitmapSource(Bitmap bmap) GDI+图片转换为 WPF 框架下使用的 BitmapSource 类型。

static BitmapSource GetBitmapSource(string file) 图片文件转换为 WPF 框架下使用的 BitmapSource 类型。

static bool GetEncodeType(byte[] data, ref Encoding encode) 通过给定的文件流，判断文件的编码类型。

static Bitmap GenByZXingNet(string msg) 将字符串转为二维码，使用时注意字符限制。

ConvertTools 类

static string EnBase64(string src) Base64 编码，将字符串转换为字节，然后字节在转换为 Base64 字符串输出。

static string DeBase64(string src) Base64 解码，将 Base64 字符串解码，输入数据为上面方法的结果。

static string DeBase64ByteArry(string src,out bool IsSuccess) Base64 解码，解码后按字节字符串输出。

static string UrlEncode(string src) URL 编码。

static string UrlDecode(string src) URL 解码。

static string HtmlEncode(string src) HTML 编码。

static string HtmlDecode(string src) HTML 解码。

static string MD5(string src) 获取字符串的 MD5 值。

static string MD5(byte[] src) 获取字节数组的 MD5 值。

static string SHA1(string src) 获取字符串的 SHA1 值（系统默认编码）。

static string SHA1_Grph(string src) 获取字符串的 SHA1 值。

static string SHA1_UTF8(string src) 获取字符串的 SHA1 值（UTF-8 编码）。

static string SHA1(byte[] src) 获取字节数组 SHA1 值。

static string FormatJson(string str) 格式化 JSON 字符串。

static string FormatXml(string sUnformattedXml) 格式化 XML 字符串。

static string FormatHTML(string src) 格式化 HTML 字符串。

static string NoHTML(string Htmlstring) 去除 HTML 标签，html 内容提取。

static string Crc32(string src) 计算字符串 Crc32 校验码。

static string Crc32(byte[] data) 计算字节数组 Crc32 校验码。

static string Crc32_uint(byte[] data) 计算字符串 Crc32 校验码输出为无符号整数字符串。

static string CookieFormat(string src) CookieFormat 字符串格式化。

static System.Drawing.Color ColorConvert(System.Windows.Media.Color color) 将 WPF 框架中的颜色结构转换为 WinForm 框架下的颜色结构。

static System.Windows.Media.Color ColorConvert(System.Drawing.Color color) 将 WinForm 框架下的颜色结构转换为 WPF 框架下的颜色结构。

static byte[] HexStrToBytes(string HexStr) 将十六进制字符串转换为字节数组。

FormatTools 类

static string MacFormat(string MacAddress)，MAC 地址字符串格式化。

static string MacFormat(byte[] MacAddress)，MAC 地址字节格式化。

static string getArryString(byte[] data) 字节数组转换为字符串。

static byte[] StrToArry(string str) 十六进制字符串转换为字节数组

static string getArryStringWithSpace(byte[] data) 字节数组转换为使用空格分开的字节字符串。

static string GetCodeArryString(byte[] data) 将字节转换为 C#数据代码字符串。

static string RawDataFormat(byte[] data) 将字节数组转换为字符串，每隔 16 个字节换行。

static string RawDataShotFormat(byte[] data) 将字节数组转换为字符串，可以控制需要最多转换多少个字节。

static string getIPv4Address(byte[] rawData, int offset) 将字节数组转换为 IP 地址。

static string getPointAddress(byte[] rawData) 将字节数组换为按照点分割开的字符串。

static string GetAppleTalkID(byte[] data) 获取苹果协议 Apple Talk ID。

static short getStaus(bool isSelect) 获取选中状态。

static string GetIPLoaction(string ipAddress) 通过 IP 地址字符串获取当前 IP 地址所在的地理位置。

static string GetPacketSize(double size) 获取字节数量，size 参数为 byte ，该数字将会自动被修正为 KB 或 MB。

static string ConvertToHexText(byte[] data, bool shortLine = false) 将字节数组转换为字节字符串，该字符串没有换行。

static int StringToInt(string src, int defaultValue) 字符串转换为整型数字

static string ReplaceControlChar(string str) 替换 ASCII 码中的控制字符。

static string ToASCIIString(byte[] values) 字节转为 ASCII 字符串。

static string ToGB2312String(byte[] values) 字节转为 GB2312 字符串。

static string ToUnicodeString(byte[] values) 字节数组转换为 Unicode 字符串。

static string ToUtf8String(byte[] values) 字节转换为 UTF-8 字符串。

3.5. NetAnalyzer.pcap 库

NetAnalyzer.pcap 库（简称 pcap 库，下同）主要是负责与 Winpcap 驱动进行交互与数据操作。该库定义了最基本的数据包模型，以及与 Winpcap 相关联的.NET 托管借口。Core 库通过操作 pcap 库相关接口获取到网卡和数据包内容，相关大部分代码来自于 Sharppcap 工程，为了更加容易与现有 NetAnalyzer 集成，所以

这里做了集中整合。

对于 pcap 中的代码，可以参考 Sharppcap 相关资料来进行操作。

3.6. NetAnalyzer.Protocol 库

NetAnalyzer.Protocol 类（Protocol 库，下同）负责对数据内容进行集中分析。该库目前支持将近 90 个常规协议的分析。包含物理成层的各种变种协议。还支持 IPv6 体系下的协议。该库的小部分协议来自 Packet.Net 工程，后经过 NetAnalyzer 不断的修改和拓展已经形成具有一定商业价值的数据分析库。

Protocol 库中的各个协议与字段都已独立类方式提供，可以通过相关协议的说明即可完成操作。

3.7. 补充信息

虽然 NetAnlyzer 接口开发尽可能的满足最大可访问资源权限，但是因为插件开发的不确定性，墨云也不能保证在可访问的资源中能够找到合理的接口，所以在插件开发的过程中，可以随时垂询，在时间允许的时候墨云尽最大努力做到有问必答。
另外对于提供的接口，插件开发者需要作必要的异常处理，以防止出现不可预知的错误。
另外插件作者可以不必局限于文档所提供的接口信息，如有必要自行寻找可用的处理方案。

4. CSharpScript 简单用法

NetAnalyzer 除了使用常规的方法进行插件开发，还可以使用基于 CSharpScript 方式的脚本开发。

对于 CSharpScript 脚本开发，除了不用继承 IPlugs 其他方式与常规开发一致，完成后只需要把脚本放入 Mods 中建立的插件文件夹下即可。

在这里我们继续使用一个小例子说明通过 CSharpScript 方式进行插件开发的方法。

首先需要说明一下这个示例的功能，我们需要在 NetAnalyzer 添加一个菜单，并在菜单中添加一个按钮，然后通过按钮调用一个第三方的类库，最后在界面上显示第三方类库提供的一句话。

4.1. 建立第三方类库

首先我们建立一个第三方的库，代码很简单，在 Class1 类中添加一个 Test 方法，调用后返回“i am three dll” 字符串。然后编译为 ThreeDLL.dll 文件

```
namespace ThreeDLL
{
    public class Class1
    {
        public string Test()
        {
            return "i am Three dll";
        }
    }
}
```

4.1-1 三方类库代码

4.2. 插件脚本

接下来编写 TestScript.cs 脚本，需要注意该脚本 3 个关注点，

- 在开始我们使用了 //css_reference ThreeDLL.dll 的注释方法，这是 CSharpScript 中的引用三方类库的方法。
- 命名空间为 NetAnalyzer.API，如果命名空间不为改名称可能会引起不可预知的错误。
- 脚本中的类只需要继承 ExBase 即可，不需要实现 IPlugs 接口

对于其他部分完全和正常插件开发一致。

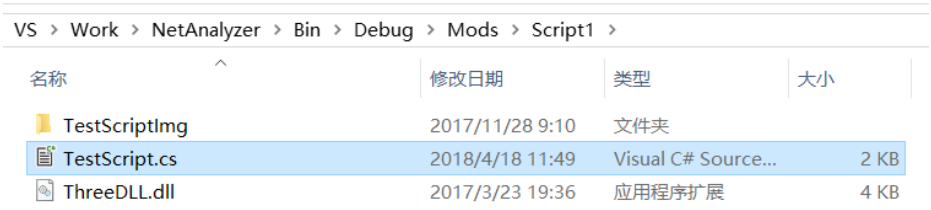
TestScript.cs

```
//css_reference ThreeDLL.dll

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using Fluent;
namespace NetAnalyzer.API
{
    public class TestScript : ExBase
    {
        public override RibbonTabItem[] GetRibbonTab()
        {
            RibbonTabItem[] tmps = new RibbonTabItem[1];

            var tab = NetAnalyzer.API.ExTools.CreatRibbonTab("测试");
            var group = NetAnalyzer.API.ExTools.CreatRibbonGroup("group");
            var button = NetAnalyzer.API.ExTools.CreatRibbonButton("测试",
                NetAnalyzer.API.ExTools.GetBitmapSource(ModsDir + "\\Script1\\TestScriptImg\\数字.png"),
                new System.Windows.RoutedEventHandler((s, e) =>
                    {
                        System.Windows.Forms.MessageBox.Show("测试" + new ThreeDLL.Class1().Test());

                    }));
            group.Items.Add(button);
            tab.Groups.Add(group);

            tmps[0] = tab;

            return tmps;
        }

    }
}
```

4.2-1 脚本代码

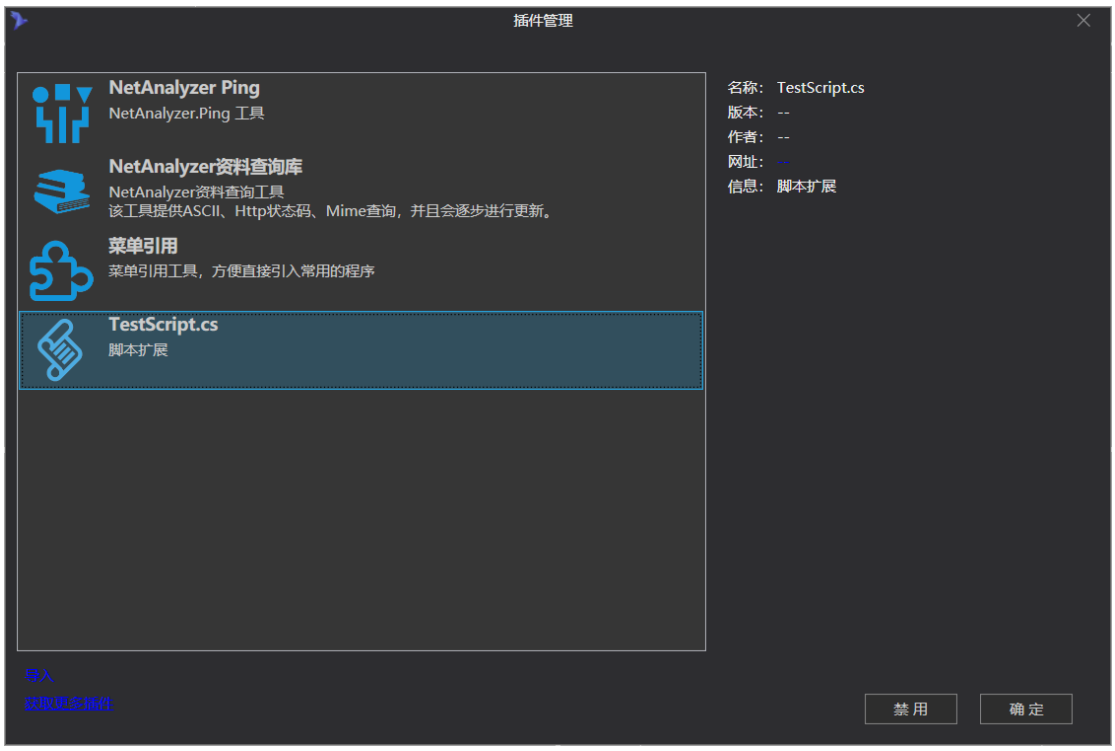
4.3. 代码部署与测试

完成了代码编写，就可以使用了，将所有的文件复制到插件目录下，如果使用了三方的 dll 需要放在插件同一目录下。



4.3-1 脚本代码

NetAnalyzer 中打开插件管理，就可以看到我们的插件了。



4.3-2 脚本代码

并且在 NetAnalyzer 我们看到了添加的测试菜单栏，点击测试按钮，就可以看到三方 dll 提供的信息了。

4.3-3 脚本代码

对于 CSharpScript 更多的特性内容限于篇幅，这里将不再过多的描述，如果对此感兴趣可以自行通过网络查找相关资历。

总结

至此，关于 NetAnalyzer 扩展与开发的相关的内容就基本完结了，虽然想尽可能的将所有的相关的内容都讲清楚，但是限于写作水平和对 NetAnalyzer 编程是产生的主观理解的偏差原因，该手册可能并没有很好的将所要描述的信息表达完全，如果你能读到这里还请谅解，并且墨云非常希望你能将自己的感受与意见通过最后一页的任意一种联系方式告诉我。

NetAnalyzer 开发已经有 8 年了，从最开始的只有自己玩的抓包小工具，发展到已经可以帮助一部分读者辅助工作，再到现在想要为 NetAnalyzer 开发更加有用的功能，一起推动 NetAnalyzer 进化，这一路走来离不开大家的支持。墨云在这里向各位使用过 NetAnalyzer 和支持过墨云的各位致以由衷的谢意。

参考资料

[1] Bastian Schmidt　Fuent.Ribbo [EB/OL] https://github.com/fluentribbon/Fluent.Ribbon

[2] Chris Morgan　Sharppcap [EB/OL] https://github.com/chmorgan/sharppcap

[3] Chris Morgan　Packet.Net [EB/OL] https://github.com/chmorgan/packetnet

附录

1.MangoScript 函数列表

select

获取将要选择的数据。通过传入选择的位置和长度参数在待测数据块中获取数据。

参数说明：

[1]偏移量 可以使用具体的值或是数学表达式

[2]数据长度 要选择的数据长度 数字或数学表达式

示例代码：

```
输入：[0x01 0x02 0x03]
代码：node test=select(0,1);
结果：test=01
```

reverse

将输入的字节数组逆序输出。

示例代码：

```
输入：[0x01 0x02 0x03]
代码：node test= select(0,3).reverse();
结果：test=03 02 01
```

switch

根据选择产生的值不同，将后续的输出转为特定的结构。

参数说明：

[1]偏移量 可以使用具体的值或是数学表达式

[2]数据长度 要选择的数据长度 数字或数学表达式

示例代码：

```
输入：[0x01 0x02 0x03]
代码：
node test= switch(1,1)
{
    case 0x01>Test, //执行外部定义的 block
    case 0x02:      //匿名 block
```

```
    {
        note sub=select(2,1)
    }
}
结果：sub=03
```

while

节点循环解析结构，通过该结构可以将当前选择的数据块中的数据使用特定方式进行重新解析。

参数说明：

[1]偏移量 可以使用具体的值或是数学表达式

[2]数据长度 要选择的数据长度 数字或数学表达式

示例代码：

```
输入：[0x01 0x02 0x03]
代码：
node test= while(0,3)
{

    note sub=select(0,1)

}
结果：sub=01
      sub=02
      sub=03
```

ifblock

用于判定数据块是否为特定的数据块。

参数说明：

[1]数值标记，推荐是 16 进制的表述方式，如 0x01

示例代码：

```
输入：[0x01 0x02 0x03]
代码：node test= select(0,1).ifblock(0x01);
结果：test=0x01
```

num

将字节数组数据转为无符号数字类型。(uint 类型，不存在负数部分，最小值为0)

示例代码：

```
输入：[0x01]
代码：node test=select(0,1).num();
结果：test=01
```

sign

将无符号数字转为有符号的数字，该函数输入数据必须是无符号数字。

示例代码：

```
输入：[0xFF 0x02 0x03 0x04]
代码：node test=select(0,4).num().sign();
结果：test=-16645372
```

left

将数字向左做移位运算。

参数说明：

[1]移动位数

示例代码：

```
输入：[0x01 0x02 0x03]
代码：node test= select(0,3).num().left(10);
结果：test=67636224
```

right

将数字向右做移位运算。

参数说明：

[1]移动位数

示例代码：

```
输入：[0x01 0x02 0x03]
代码：node test= select(0,3).num().right(10);
结果：test=64
```

and

数字做且运算。

参数说明：

[1]且运算数字

示例代码：

```
输入：[0x01 0x02 0x03]
代码：node test=select(0,3).num().and(0);
结果：test=0
```

or

数字做或运算。

参数说明：

[1]或运算数字

示例代码：

```
输入：[0x01 0x02 0x03]
代码：node test= select(0,3).num().or(0x000011);
结果：test=66067
```

text

字节转为字符串。

参数说明：

[1]字符串编码，如：ascii utf-8 gb2312 等

示例代码：

```
输入：[0x41 0x42 0x43]
代码：node test= select(0,3).text("ascii");
结果：test=ABC
```

display

将字节转为在编码中 enum 块中指定的显示内容。

参数说明：

[1]enum 所定义的名称

示例代码：

```
输入：[0x01 0x02 0x03]
代码：node test= select(0,1).diaplay(FrameType);
```

```
    enum FrameType
    {
            case 0x00 > "测试 1",
            case 0x01 > "测试 2",
            case 0x02 > "测试 3"
    }
结果：test=测试 2
```

padleft

向左补齐指定位数的自定义字符。

参数说明：

[1]向左数指定的宽度

[2]字符。如果输入字符串则取第一个

padright

向右补齐指定位数的自定义字符。

参数说明：

[1]向右数指定的宽度

[2]字符。如果输入字符串则取第一个

format

将数字按照指定格式输出字符串。

参数说明：

[1]二进制宽度

[2]十六进制字符宽度

[3]补齐的字符，如果是字符串则取第一个

[4]规定的格式字符串。其中占位符{@value}表示输入的数字，{@hex}表示十六进制输出，{@bin}表示二进制输出

hex

将数字出处为十六进制字符串。

bin

将数字出处为二进制进制字符串。

eachbyte

对每个字节按照指定分隔符进行输出。

参数说明：

[1]分隔符

[2]txt 对单个字节转为对应的 ascii 字符，num 转为对应的数字，hex 按照十六进制表示输出。

sub

对输出的长字符串进行截取

参数说明：

[1]需要截取的长度

insert

在指定位置插入对应的字符

参数说明：

[1]位置

[2]插入的字符串

xmlencode

xml 编码

xmldecode

xml 解码

urlencode

url 编码

urldecode

url 解码

随着 MangoScript 的升级，该列表也会做相应的更改。

NetAnalyzer 使用说明书

二．扩展与开发

作者：冯天文
时间：2019 年 3 月 16 日星期六
邮件：fitanwen@126.com
网站：http://twzy.sinaapp.com/
QQ 群：39753670