# 脚本加载

## 创建脚本

创建一个文本文件，命名为 你想要的名字.lua。

按第2章的内容写好脚本文件

## 注册脚本

注册脚本有两种方法，各有利弊，可以选择自己喜欢的方法注册。

方法一：将lua脚本文件放到wireshark安装目录的plugins/<版本号>目录下。该方法在wireshark更新后，需要重新将脚本文件放到新的版本好下。升级wireshark的时候会删除原来的脚本文件，记得备份

方法二：将lua脚本文件放到wireshark安装目录下，修改wireshark安装目录下的init.lua文件，在最后加上dofile(".\\路径名\\你的脚本名字.lua ")

## 加载脚本

方法一：重新打开wireshark文件

方法二：菜单→分析→重新载入Lua插件

方法三：使用快捷键Ctrl + Shift + L

# Lua脚本写法

## 框架

框架包括解析结构、字段、解析器的定义和注册两部分

|  |
| --- |
| do  --[[  创建一个新的协议结构 PtpSendPtpPktAsk  第一个参数是协议名称会体现在过滤器中  第二个参数是协议的描述信息，无关紧要  --]]  local DebugString = Proto("DebugString", "DebugString")    --[[  下面定义字段  --]]  local DebugString\_string = ProtoField.string("DebugString.string", "String", base.STRING)    -- 将字段添加到协议中  DebugString.fields = {  DebugString\_string  }    --[[  下面定义 foo 解析器的主函数，这个函数由 wireshark调用  第一个参数是 Tvb 类型，表示的是需要此解析器解析的数据  第二个参数是 Pinfo 类型，是协议解析树上的信息，包括 UI 上的显示  第三个参数是 TreeItem 类型，表示上一级解析树  --]]  function DebugString.dissector(tvb, pinfo, treeitem)  -- 设置一些 UI 上面的信息  pinfo.cols.protocol:set("DebugString")  pinfo.cols.info:set("Debug String")    local offset = 0  local tvb\_len = tvb:len()    -- 在上一级解析树上创建 DebugString 的根节点  local DebugString\_tree = treeitem:add(DebugString, tvb:range(offset))    -- 下面是想该根节点上添加子节点，也就是自定义协议的各个字段  -- 注意 range 这个方法的两个参数的意义，第一个表示此时的偏移量  -- 第二个参数代表的是字段占用数据的长度  DebugString\_tree:add(DebugString\_string, tvb:range(offset))  pinfo.cols.info:set(tvb:range(offset):string())  end    -- 向 wireshark 注册协议插件被调用的条件  local ethertype\_table = DissectorTable.get("ethertype")  ethertype\_table:add(65002, DebugString)  end |

## 语法

### 创建解析器

解析器的写法如下

|  |
| --- |
| local DebugString = Proto("DebugString", "DebugString") |

### 字段定义

字段定义主要是为了定义协议中的各个字段的类型和名称

|  |
| --- |
| local DebugString\_string = ProtoField.string("DebugString.string", "String", base.STRING) |

#### 解析类型

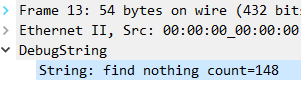
其中，ProtoField.string可以使用的类型有boolean/uint8/uint16/uint24/uint32/uint64/ int8/int16/int24/int32/int64/float/double/absolute\_time/relative\_time/string/stringz/uint\_string/ether/bytes/uint\_bytes/ipv4/ipv6/ipxnet/framenum/pcre/guid/oid/eui64。一些IP类型，MAC地址类型，boolean类型，string类型，时间类型等都可以直接使用，不需要自己去解析

#### 实际名称

字段的第一个参数是字段的实际名称，必须唯一；

#### 解析名称

第二个参数是字段的解释信息，最终会显示在解析的位置；



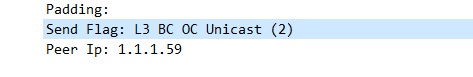
#### 类型

第三个参数是字段类型，可以取值如下：base.NONE/base.BOOLEAN/base.UINT8/ base.UINT16/base.UINT24/base.UINT32/base.UINT64/base.INT8/base.INT16/base.INT24/base.INT32/base.INT64/base.FLOAT/base.DOUBLE/base.ABSOLUTE\_TIME/base.RELATIVE\_TIME/base.STRING/base.STRINGZ/base.UINT\_STRING/base.ETHER/base.BYTES/base.UINT\_BYTES/base.IPV4/base.IPV6/base.IPXNET/base.FRAMENUM/base.PCRE/base.GUID/base.OID/base.EUI64

#### 值的枚举解释

第四个可选参数，是对字段值的枚举解析，用法如下：

|  |
| --- |
| --解析PtpSendPtpPktAsk\_send\_flag值代表的含义  local PtpSendPtpPktAsk\_send\_flag\_Em = {  [0] = "L2 packet",  [1] = "L3 TC packet",  [2] = "L3 BC OC Unicast",  [3] = "L3 BC OC TC Multicast",  }  local PtpSendPtpPktAsk\_send\_flag = ProtoField.uint8("PtpSendPtpPktAsk.sendFlag", "Send Flag", base.DEC, PtpSendPtpPktAsk\_send\_flag\_Em) |

解析后的效果如下：  


#### 位域的解析

第五个可选参数，是位掩码。位的长度取决于2.2.2.1解析类型。

|  |
| --- |
| IfIndexBitFiled\_Equipment\_Em = {  [0] = "first NE",  [1] = "second NE"  }  local IfIndexBitFiled\_Equipment = ProtoField.uint32("IfIndexBitFiled.Equipment", "Equipment", base.DEC, IfIndexBitFiled\_Equipment\_Em, 0x80000000) |

### 解析函数

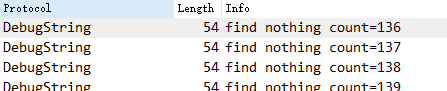
解析函数的写法如下

|  |
| --- |
| function DebugString.dissector(tvb, pinfo, treeitem)  -- 设置一些 UI 上面的信息  pinfo.cols.protocol:set("DebugString")  pinfo.cols.info:set("Debug String")    local offset = 0  local tvb\_len = tvb:len()    -- 在上一级解析树上创建 DebugString 的根节点  local DebugString\_tree = treeitem:add(DebugString, tvb:range(offset))    -- 下面是想该根节点上添加子节点，也就是自定义协议的各个字段  -- 注意 range 这个方法的两个参数的意义，第一个表示此时的偏移量  -- 第二个参数代表的是字段占用数据的长度  DebugString\_tree:add(DebugString\_string, tvb:range(offset))  pinfo.cols.info:set(tvb:range(offset):string())  end |

#### UI信息显示

|  |
| --- |
| pinfo.cols.protocol:set("DebugString")  pinfo.cols.info:set("Debug String") |

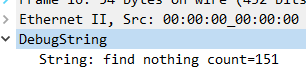
分别设置wireshark的protocol和info两列的内容，注意，该值会被后续的修改覆盖。如下图，info信息就被后来的pinfo.cols.info:set(tvb:range(offset):string())修改掉



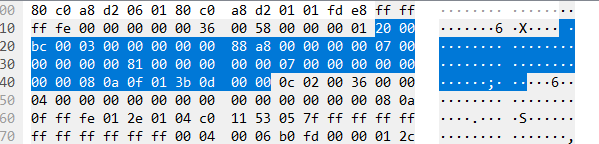
#### 创建节点

|  |
| --- |
| local DebugString\_tree = treeitem:add(DebugString, tvb:range(offset)) |

节点可以决定在哪个节点下创建，如本例子，就是在根节点treeitem下创建了节点。节点的名称为第一个参数的解析结构定义的名称决定。



节点的范围由第二个参数指定；range函数可带两个参数，第一个是起点，第二个是终点，默认为-1，表示到包的末尾。如果有指定长度，选中节点后，会选中对应的报文的字节。节点的范围可以后续改变，推荐使用这种方法，改变的的方法见2.2.3.3



#### 改变节点的范围

节点的范围可以通过 节点名称:set\_len()设置，可以动态调整节点长度

|  |
| --- |
| PtpSendPtpPktAsk\_PortCfgTree:set\_len(offset - PtpSendPtpPktAsk\_PortCfgTree\_StartOffset) |

#### 自定义解析内容

|  |
| --- |
| PtpSendPtpPktAsk\_PortCfgTree:add(PtpSendPtpPktAsk\_PortCfg\_EncapMode, tvb:range(offset, 1))  offset = offset+1  offset = offset+3  PtpSendPtpPktAsk\_PortCfgTree:add(PtpSendPtpPktAsk\_PortCfg\_SpVlanTPID, tvb:range(offset, 4))  offset = offset+4  PtpSendPtpPktAsk\_PortCfgTree:add(PtpSendPtpPktAsk\_PortCfg\_SpVlanId, tvb:range(offset, 4))  offset = offset+4PtpSendPtpPktAsk\_PortCfgTree:add(PtpSendPtpPktAsk\_PortCfg\_SpVlanPri, tvb:range(offset, 1))  offset = offset+1  offset = offset+3 |

##### 值获取

值获取可以使用uint()等函数，获取到值后，可以用来做运算、判断等操作

|  |
| --- |
| if tvb:range(offset - 4,4):uint()==0 then  --do something  end |

##### 位域值的获取

位域值的获取，可以使用bitfield([offset], [length])函数。其中，可选参数offset是位域的起始位置，单位为bit，默认为0；可选参数length是位域的长度，单位是bit，默认为1

|  |
| --- |
| if tvb:range(0,1):bitfield(0,4)~=0 and (tvb:range(1,1):bitfield(0,4)~=2 or tvb:range(1,1):bitfield(0,4)~=1) then  --do something  end |

#### 使用系统自带的解析器解析内容

|  |
| --- |
| local ptpDissector = DissectorTable.get("udp.port"):get\_dissector(320)  ptpDissector:call(tvb:range(offset):tvb(), pinfo, PtpSendPtpPktAsk\_tree) |

## 自定义类型与系统类型冲突处理

如果自定义的类型不合理，刚好和标准类型冲突，则需要在标准类型上加上装饰器。对该类型进行自定义解析，判断是否是自定义的内容，如果是，则用自定义的解析器解析，否则用标准解析器解析。例如：

|  |
| --- |
| --代码里原始的快速以太的PTP报文的类型为0x88f7，和真实的PTP报文重叠了，此处区分开  MayBeFastEthProto = Proto("MayBeFastEth", "MayBeFastEth")  function MayBeFastEthProto.dissector(tvb, pinfo, treeitem)    --PTP协议的transportSpecific不为0(这个字段的位置为自定义的快速以太报文的ifindex的Equipment+Control+card index的第一个bit，一般为0xf或者2)  --且version字段不为1或2(这两个字段的位置为自定义的快速以太报文的ifindex的port index的前4个bit，一般为8(L3端口)、0(L2端口)、0xf(广播、发到指定槽位))。  if tvb:range(0,1):bitfield(0,4)~=0 and (tvb:range(1,1):bitfield(0,4)~=2 or tvb:range(1,1):bitfield(0,4)~=1) then  ptp\_fast\_eth\_proto.dissector(tvb, pinfo, treeitem)  else  local ptpDissector = DissectorTable.get("udp.port"):get\_dissector(320)  ptpDissector:call(tvb:range(offset):tvb(), pinfo, treeitem)  end  end  -- 向 wireshark 注册协议插件被调用的条件  local ethertype\_table = DissectorTable.get("ethertype")  ethertype\_table:add(0x88f7, MayBeFastEthProto) |