1. 开发环境

本项目客户端在`windows`操作系统下开发，Agent选择`ubuntu22.04`虚拟机

客户端使用python编写，主要使用了`snmp-cmds`库用于SNMP相关处理，`PyQt5`库用于GUI设计，此外还需要配置Net-SNMP环境，在防火墙处将UDP的161端口与162端口打开

Agent端需要安装`snmpd` `snmp` `snmp-mibs-downloade`三者，然后将UDP161端口打开

2. 整体思路

项目主体包括以下三个部分：

- `func.py`:调用`snmp-cmds`库来实现SNMP功能，如Get,Set,Trap等
- `Gui.py`:调用`PyQt`库设计GUI界面
- `main.py`:充当中间件，接收来自GUI界面的信号，将信号与相关的槽函数进行绑定，槽函数调用funcs.py中的功能函数，并将返回结果展示在GUI界面上

3. 功能模块设计

这一部分分析Get、Set、Trap功能以及监控系统状态功能的实现

3.1 Get功能

调用`snmpwalk`函数根据`目的IP`、`oid`以及`community`等参数获取oid对应的值，最后将结果返回

```
def GetByOid(des_ip, oid):
    try:
        res = snmp_cmds.snmpwalk(des_ip, oid, 'public', 161, TIMEOUT)
        return [varBind for varBind in res]
    except:
        return "connect error"
```

为了简化设计，便于使用，在设计时将Agent的接收端口固定为161，community固定为public，超时参数设置为100s

3.2 Set功能

原理同Get功能，调用`snmpset`函数设置指定oid对应的值，并将结果返回

```
def SetByOid(des_ip, oid, value, type):
    try:
        types = {"integer": 'i',
                 "unsigned_integer": 'u',
                 "time_ticks": 't',
                 "ip_address": 'a',
                 "object_identifier": 'o',
                 "string": 's',
                 "hex_string": 'x',
                 "decimal_string": 'd',
                 "bit_string": 'b'}
        type_index = types[type]
```

```
        res = snmp_cmds.snmpset(des_ip, oid, type_index, value, 'public', 161,
TIMEOUT)
        return res
    except:
        return "NoChangable!"
```

函数首先对`value`的类型进行检查，根据其类型选择传递给`snmpset`函数的参数。一些默认设置同Get功能

3.3 Trap功能

Trap功能主要分为发送trap包与接收trap包两种

3.3.1 发送trap包

该功能通过`sendNotification`方法实现

```
def sendTrap(des_ip, oid, oid_extra, value):
    try:
        # sendNotification函数用来发送SNMP消息，包括trap和inform
        errorIndication, errorStatus, errorIndex, varBinds = next(
            sendNotification(
                SnmpEngine(),
                CommunityData('public'),
                UdpTransportTarget((des_ip, 162)),
                ContextData(),
                'trap',
                NotificationType(
                    ObjectIdentity(oid)
                ).addVarBinds(
                    (oid_extra, OctetString(value))
                )
            )
        )

        if errorIndication:
            print('Notification not sent: %s' % errorIndication)
            return errorStatus, errorIndex
    except:
        return "send error"
```

`sendNotification`方法有如下参数：

- `snmpEngine`：SNMP引擎实例
- `CommunityData`：社区相关属性
- `UdpTransportTarget`：目标机器IP及端口配置
- `ContextData`：发送数据的类型（trap/inform）
- `NotificationType`：要发送的trap数据

如果发送成功，则将结果返回；否则抛出错误

3.3.2 监听trap包

本项目实现的监听功能可以实现实时展示监听到的trap包，且可以点击查看细节

定义`TrapListener`类用于实现监听功能

```
class TrapListener:
    def __init__(self, external_function):
        # 该结构用于存储trap包中除数据外的内容
        self.trap_info = {
            "1.3.6.1.2.1.1.3.0": "",        # 超时时间
            "1.3.6.1.6.3.1.1.4.1.0": "",    # trapOid
            "1.3.6.1.6.3.18.1.3.0": "",     # IP
            "1.3.6.1.6.3.18.1.4.0": "",     # community
            "1.3.6.1.6.3.1.1.4.3.0": ""     # trapType
        }
        self.snmpEngine = SnmpEngine()
        # 当捕获到一个trap包时调用external_function函数传递给中间件，中间件再将trap包传递给GUI界面进行展示
        self.external_function = external_function

    # 处理trap包的函数
    def cbFun(self, snmpEngine, stateReference, contextEngineId, contextName, varBinds, cbCtx):
        for name, val in varBinds:
            self.trap_info[name.prettyPrint()] = val.prettyPrint()
        # 将trap包传递出去
        self.external_function(self.trap_info)

    # 循环监听trap包
    def listenTrap(self):
        # 配置监听器
        config.addTransport(
            self.snmpEngine,
            udp.domainName,
            udp.UdpTransport().openServerMode(('0.0.0.0', 162))
        )
        # 指定community
        config.addV1System(self.snmpEngine, 'my-area', 'public')
        # 将监听器与trap包处理函数绑定
        ntfrcv.NotificationReceiver(self.snmpEngine, self.cbFun)
        # 启动监听器
        self.snmpEngine.transportDispatcher.jobStarted(1)
        try:
            self.snmpEngine.transportDispatcher.runDispatcher()
        except:
            self.snmpEngine.transportDispatcher.closeDispatcher()
            raise

    def getTrapInfo(self):
        return self.trap_info
```

本项目中的监听功能会接收所有发往服务端的trap包

中间件程序（`main.py`）会在启动监听时创建一个线程来调用上述的监听函数`listenTrap`，`InsertTrap`函数对应上述的`external_function`，用于处理接收到的trap包

```
traplistener = funcs.TrapListener(self.InsertTrap)
thread_listen = threading.Thread(target=traplistener.listenTrap)
# 将daemon属性设置为True后，当主窗口被关闭时该线程也会自动结束
thread_listen.daemon = True
thread_listen.start()
```

```
# 储存trap包
self.trap_packets = []

def InsertTrap(self, trap_info):
    # 根据trap包创建一个新的副本，否则当新的trap包到达时会覆盖trap_packets中已有trap包的内
容
    new_trap = copy.deepcopy(trap_info)
    self.trap_packets.append(new_trap)
    ip = self.trap_packets[-1]["1.3.6.1.6.3.18.1.3.0"]
    community = self.trap_packets[-1]["1.3.6.1.6.3.18.1.4.0"]
    # 在tableWidget控件中插入新trap包内容
    row_num = self.ui.tableWidget.rowCount()
    self.ui.tableWidget.insertRow(row_num)
    self.ui.tableWidget.setItem(row_num, 0, QTableWidgetItem(ip))
    self.ui.tableWidget.setItem(row_num, 1, QTableWidgetItem(community))
```

截止到这里，为了简洁美观，只将trap包的来源IP及来源community展示在了tableWidget中，用户可以点击单元格查看对应trap包的详细信息

```
# 当用户点击单元格时触发槽函数
self.ui.tableWidget.itemClicked.connect(lambda item: self.showTrapInfo(item))

def showTrapInfo(self,item):
    row_num = item.row()
    packet = self.trap_packets[row_num]
    packet_list = list(packet.items())
    packet_values = packet_list[5:]
    packet_values = dict(packet_values)

    self.ui.textEdit_7.setText("过期时间：{} \ntrap消息对应的oid：{} \ntrap包的源IP:
{} \ntrap包的源社区:{} \ntrap包类型:{} \n数据:{}".format(
        packet["1.3.6.1.2.1.1.3.0"],
        packet["1.3.6.1.6.3.1.1.4.1.0"],
        packet["1.3.6.1.6.3.18.1.3.0"],
        packet["1.3.6.1.6.3.18.1.4.0"],
        packet["1.3.6.1.6.3.1.1.4.3.0"],
        packet_values
    ))
```

至此，监听trap包并展示的功能全部结束

3.4 监控系统性能

3.4.1 监控CPU使用率

本项目中的CPU使用率对应**系统CPU使用率**，而非用户CPU使用率等其他指标

```
def monitor_cpu(des_ip):
    try:
        res = snmp_cmds.snmpwalk(des_ip, '.1.3.6.1.4.1.2021.11.11.0', 'public',
161, TIMEOUT)
        return 100 - int(res[0][1])
    except:
        return "connect error"
```

此处的逻辑也很简单，不多赘述

3.4.2 监控内存使用率

此功能的实现要分为三步：

- 查询可用内存总量

```
res1 = GetByOid(des_ip, '.1.3.6.1.4.1.2021.4.6.0')
```

- 查询当前使用内存量

```
res2 = GetByOid(des_ip, '.1.3.6.1.4.1.2021.4.5.0')
```

- 计算使用率并返回

```
res1_value = float(res1[0][1].replace(' kB', ''))
res2_value = float(res2[0][1].replace(' kB', ''))
return res1_value / res2_value
```

逻辑如上，不多赘述

3.4.3 监控磁盘使用率

要实现此功能，与监控内存使用率同理：计算可用磁盘总量与当前使用量，然后计算磁盘使用率

3.4.4 监控网络流量

网络流量分为两部分：上行流量与下行流量

可以根据oid查询的数据是某一时刻上行或下行字节数的总量，因此要在第一次查询后等待2s，然后再次查询，最后得到网络流量

3.4.5 实时展示

本项目实现的监控功能可以每3s自动刷新各个系统指标，并自动更新变化曲线

定义了`MonitorThread`类来实现实时刷新系统指标的功能

```
class MonitorThread(QThread):
    data_updated = pyqtSignal()
```

```
    def __init__(self, window, des_ip):
        super().__init__()
        self.window = window
        self.des_ip = des_ip
        self.should_stop = False  # 添加一个标志来表示线程是否应该停止

    def run(self):
        cpu = []
        RAM = []
        disk = []
        upload = []
        download = []

        while not self.should_stop:
            # 调用上述监控函数来获取当前指标，并将其存储起来
            ..............................
            # 更新图像
            self.window.update_matplotlib_figure(self.window.ui.groupBox_4, cpu)
            self.window.update_matplotlib_figure(self.window.ui.groupBox_5, RAM)
            self.window.update_matplotlib_figure(self.window.ui.groupBox_6,
disk)
            self.window.update_matplotlib_figure(self.window.ui.groupBox_7,
upload)
            self.window.update_matplotlib_figure(self.window.ui.groupBox_8,
download)

            # 触发更新绘图界面的函数
            self.data_updated.emit()
            # 休眠3s后更新界面
            time.sleep(3)
    # 用于终止线程
    def stop(self):
        self.should_stop = True
```

在中间件（`main.py`）中创建新线程负责上述监控功能

```
def monitor(self,LineEdit):
    des_ip = LineEdit.text()
    if des_ip == "" or self.checkip(des_ip) == False:
        QMessageBox.information(self, "提示", "请输入正确的IP地址")
        return

    for groupBox in self.groupBoxes:
        # 为每项指标创建图像
        self.create_matplotlib_figure(groupBox)
        # 创建工作线程
        self.monitor_thread = MonitorThread(self, des_ip)
        # 将信号与槽函数绑定
        self.monitor_thread.data_updated.connect(self.update_ui)
        self.monitor_thread.start()
```

更新图像函数负责根据最新的数据对折线进行更新

```
def update_matplotlib_figure(self, groupBox, data):
```

```
    # 获取matplotlib图像的第一个子图
    ax = groupBox.layout().itemAt(0).widget().figure.axes[0]

    # 更新线条的数据
    x_data = range(len(data))
    ax.lines[0].set_data(x_data, data)

    # 更新坐标轴的范围
    ax.relim()
    ax.autoscale_view()

    # 重绘图像
    ax.figure.canvas.draw()
```

3.5 阈值报警

此项功能的实现与系统指标监控相同，在中间件（`main.py`）中创建线程来循环调用`funcs.py`中的功能函数监控CPU与内存的使用率

如果超过阈值则弹窗告警

- 功能函数

```
def warn_cpu(des_ip, warn_level):
    # 获取阈值
    warn_level = int(warn_level)
    # 获取当前使用率
    res = monitor_cpu(des_ip)
    # 超出阈值则返回异常信息
    if int(res) > warn_level:
        return False
    time.sleep(1)
    return True
```

- 创建线程监听

```
def warnCpu(self, ComboBox, LineEdit):
    des_ip = LineEdit.text()
    level = ComboBox.currentText()
    # 创建线程进行监控
    self.thread_cpu = WarnCpu(self, des_ip, level)
    # 超过阈值则调用弹窗函数进行告警
    self.thread_cpu.cpu_overload.connect(self.show_CPUwarning)
    self.thread_cpu.start()
```

- 线程类定义：

```
class WarnCpu(QThread):
    cpu_overload = pyqtSignal()  # 定义一个信号

    def __init__(self, window, des_ip, level):
        super().__init__()
        self.window = window
        self.des_ip = des_ip
```

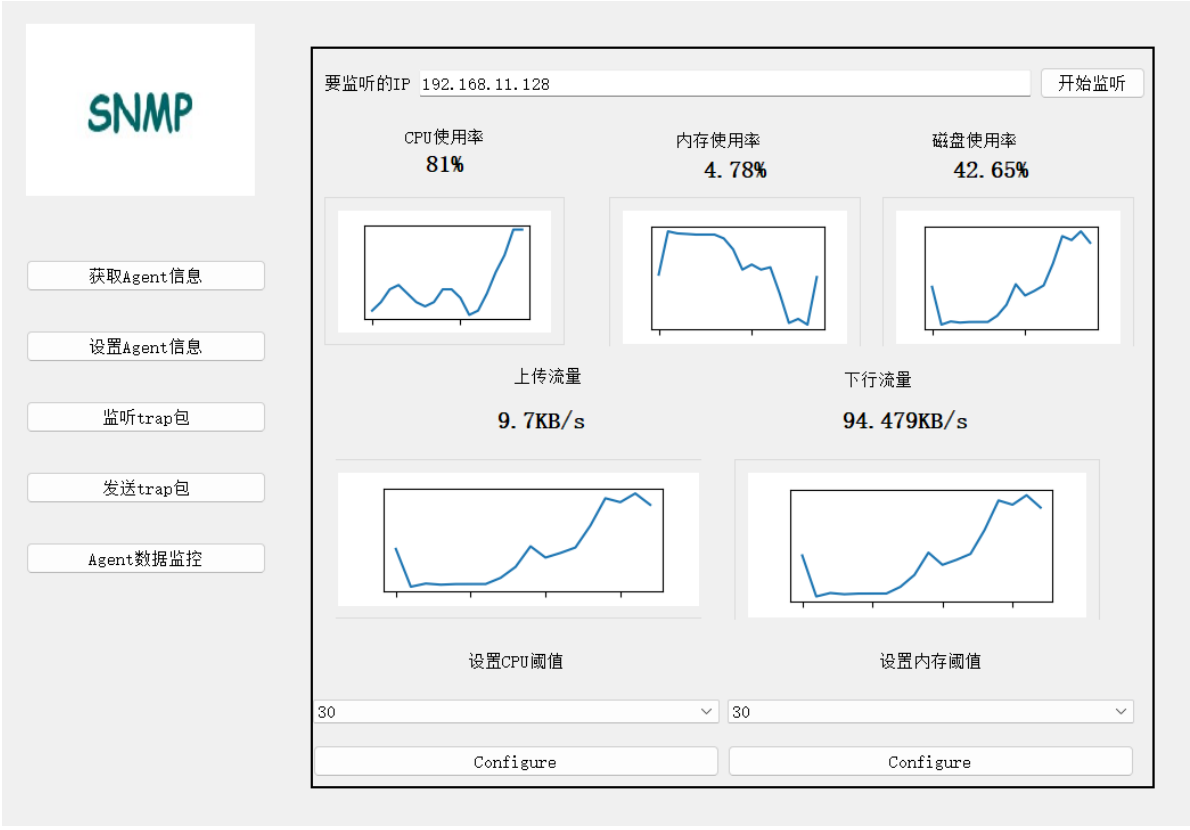
```
        self.level = level
        self.stop = False

    def run(self):
        while not self.stop:
            if funcs.warn_cpu(self.des_ip, self.level) == False:
                # 超过阈值
                self.cpu_overload.emit()
                # 休眠3s继续监控
                time.sleep(3)
            else:
                continue

    def stop(self):
        self.stop = True
```

- 弹窗函数

```
def show_CPUwarning(self):
    QMessageBox.information(self, "警告", "CPU使用率超过设定的阈值！")
```

上面对于CPU的阈值监控和对于内存的阈值监控相同，不再赘述

4. GUI设计



左侧用于切换各项功能，右侧是功能区

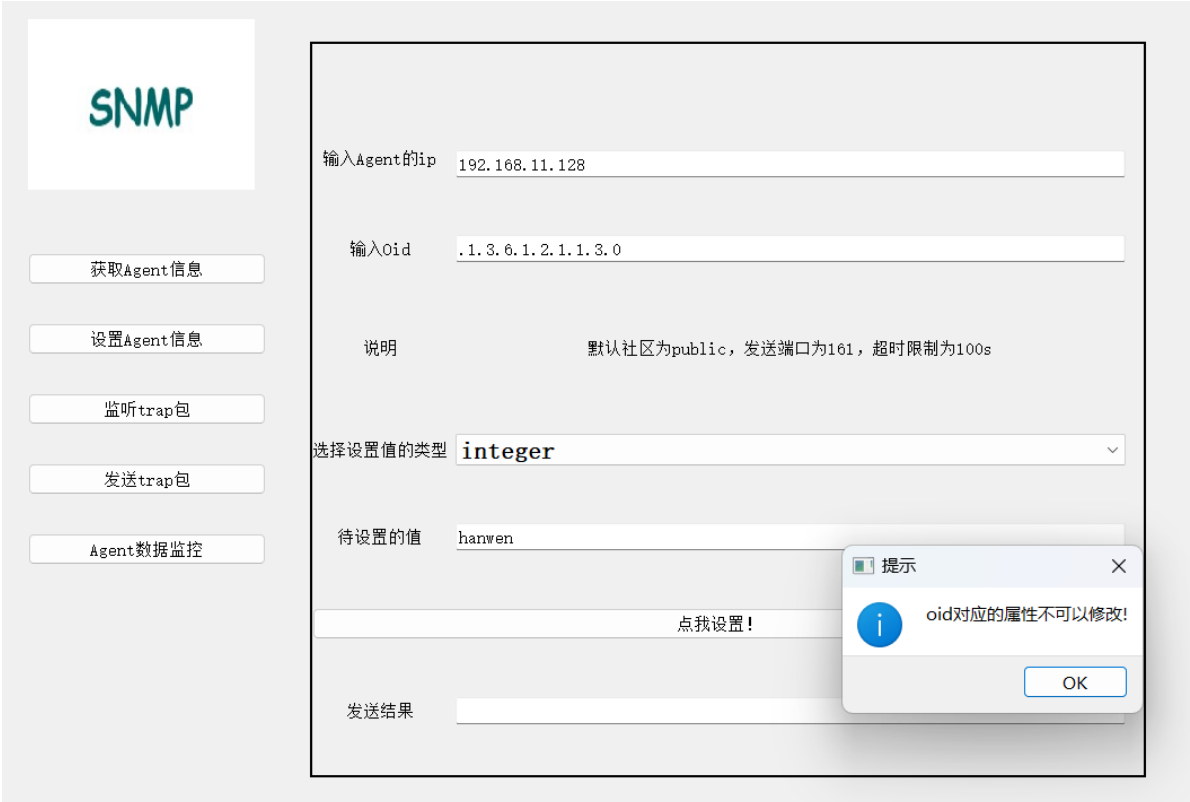
功能区使用了`stackWidget`来进行功能区页面的切换

5. 成果展示

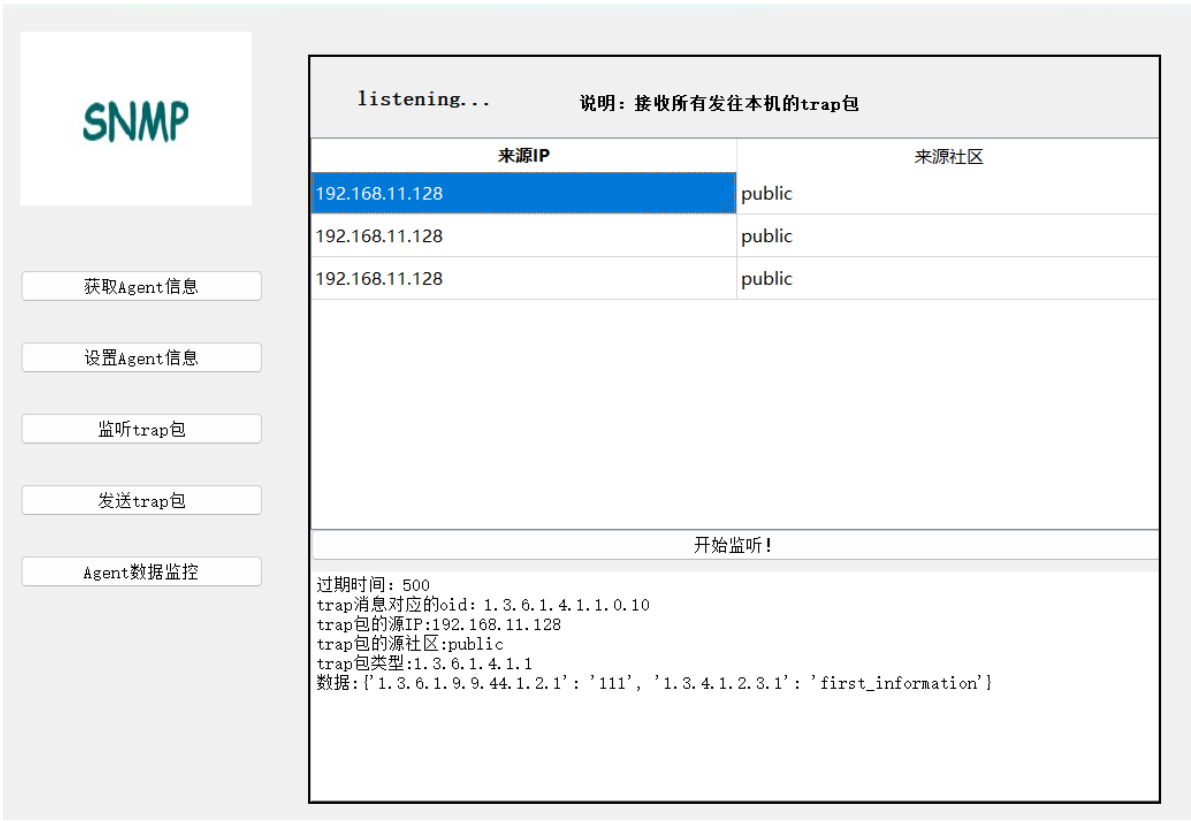
5.1 Get功能

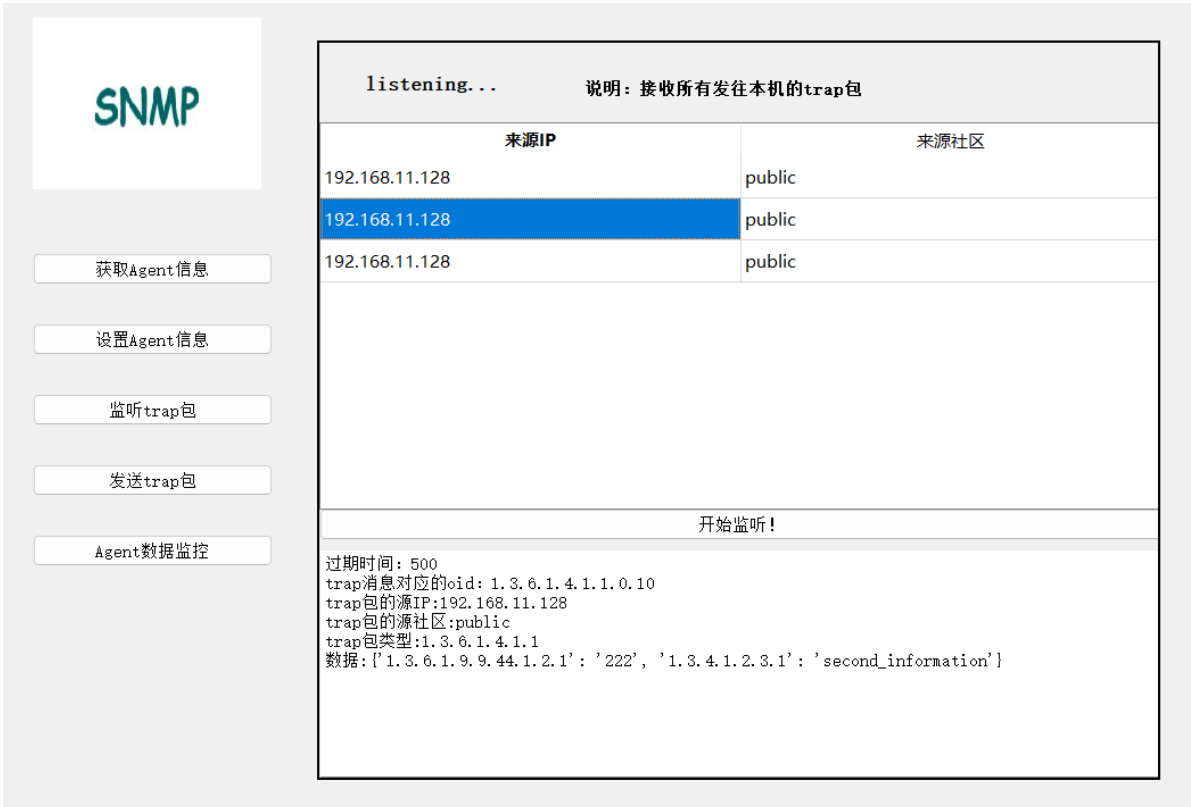
5.2 Set功能

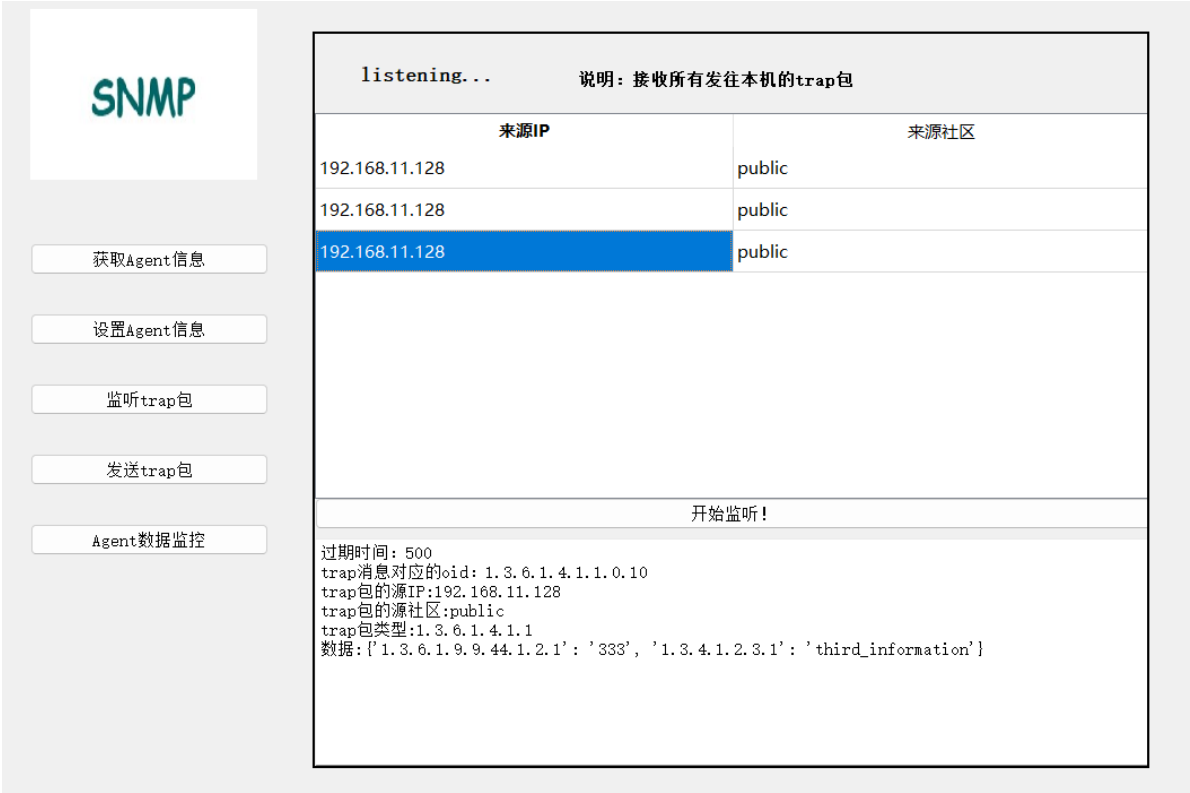
因为很难确定哪些oid对应的内容是可以通过set来修改的，所以选择错误反馈进行展示



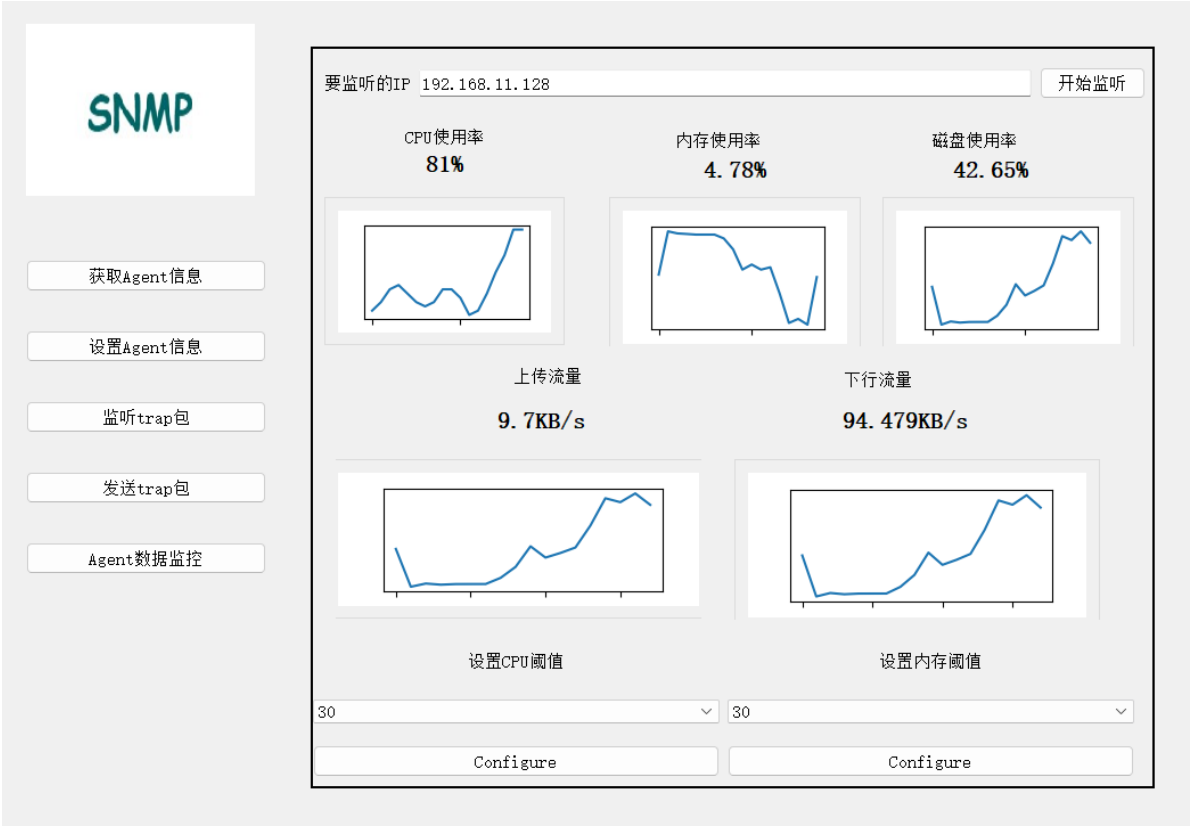
5.3 监听trap


SNMP
获取Agent信息
设置Agent信息
监听trap包
发送trap包
Agent数据监控
listening...
说明：接收所有发往本机的trap包
来源IP
来源社区
192.168.11.128
public
192.168.11.128
public
192.168.11.128
public
开始监听！
过期时间：500
trap消息对应的oid：1.3.6.1.4.1.1.0.10
trap包的源IP:192.168.11.128
trap包的源社区:public
trap包类型:1.3.6.1.4.1.1
数据:{'1.3.6.1.9.9.44.1.2.1': '111', '1.3.4.1.2.3.1': 'first_information'}

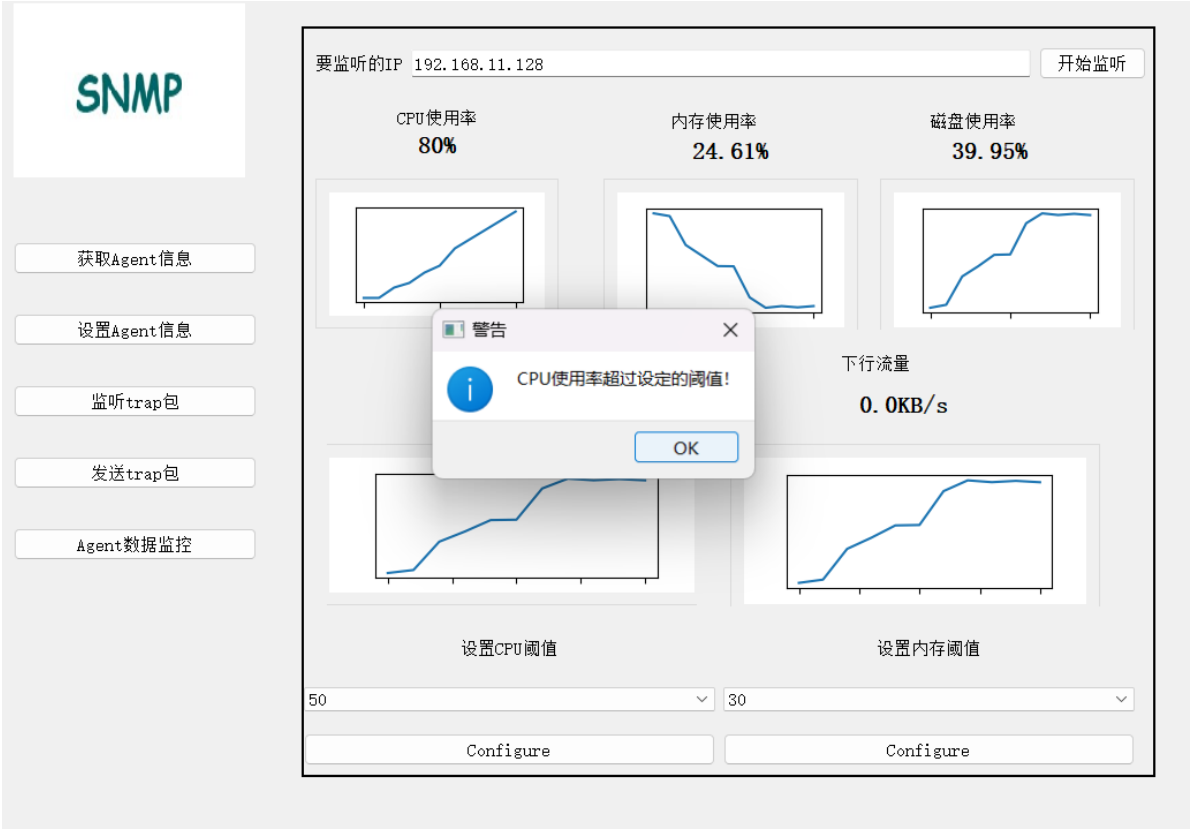

SNMP
获取Agent信息
设置Agent信息
监听trap包
发送trap包
Agent数据监控
listening...
说明：接收所有发往本机的trap包
来源IP
来源社区
192.168.11.128
public
192.168.11.128
public
192.168.11.128
public
开始监听！
过期时间：500
trap消息对应的oid：1.3.6.1.4.1.1.0.10
trap包的源IP:192.168.11.128
trap包的源社区:public
trap包类型:1.3.6.1.4.1.1
数据:{'1.3.6.1.9.9.44.1.2.1': '222', '1.3.4.1.2.3.1': 'second_information'}



5.4 性能监控



5.5 阈值告警



6. 现存不足

- 为了简化使用，默认将Get等操作时的社区设置为`public`，Set操作的社区设置为`private`，这是很不合理，也是很不安全的。之后可以改良项目，给用户手动修改community的机会
- 捕获trap包的界面过于简陋，不够美观，而且对trap数据包的数据解析不够细致
- 性能监控界面过于拥挤，不够美观