如何3D可视化ns3中的网络场景

1. 初始化

1.1 头文件

```
#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/mobility-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/ssid.h"
#include "ns3/netanim-module.h"
#include "ns3/netsimulyzer-module.h"
#include "string.h"
#include "ns3/bridge-helper.h"
#include "ns3/aodv-helper.h"
#include "ns3/wifi-module.h"

using namespace ns3;

/*
这两个日志机制可以同时存在而不会互相影响。
NS_LOG_COMPONENT_DEFINE 是用于 NS-3 控制台日志，便于实时调试；
LogStream 则将事件记录发送到 NetSimulyzer 中，用于仿真场景的可视化和分析。
*/
//日志输出，采用*eventLog<<Simulator...方式直接写入Netsimulyzer
Ptr<netsimulyzer::LogStream> eventLog;
//设置日志组件，可以在ns3控制台设置不同的级别来查看日志
NS_LOG_COMPONENT_DEFINE ("mycommunication");
//Ptr<const MobilityModel> model 是一个指向 MobilityModel 对象的智能指针，表示传入的模型是不可修改的（const）
void
CourseChanged(Ptr<const MobilityModel> model)
{
    //auto 关键字用于在声明变量时自动推导变量的类型。编译器会根据变量的初始化值来确定其类型
    //获取节点对象：model->GetObject<Node>() 返回与该移动模型关联的节点对象。
```

```
    //获取节点 ID: GetId() 方法返回该节点的唯一标识符（ID），并将其存储在 nodeId 变量中。
    const auto nodeId = model->GetObject<Node>()->GetId();
    //返回节点当前位置
    const auto position = model->GetPosition();
    //返回节点当前速度向量
    const auto velocity = model->GetVelocity();
    //Simulator::Now().GetMilliSeconds()获取当前的模拟时间，以毫秒为单位
    //<<操作符用于将信息格式化并写入日志流
    /*
    position.x, position.y, position.z 和 velocity.x, velocity.y, velocity.z 用于输出节点的新位置和速度。
    */
    *eventLog << Simulator::Now().GetMilliSeconds() << ": Node [" << nodeId
              << "] Course Change Position: [" << position.x << ", " << position.y << ", "
              << position.z << "] "
              << "Velocity [" << velocity.x << ", " << velocity.y << ", " << velocity.z << "]\n";
}
```

1.2 初始值设定

```
//初始值一般包括范围的横纵长度，如minposition，maxposition，速度的范围，minspeed，maxspeed，仿真时长等
double minposition=-100;
double maxposition=100;
double minspeed=0.1;
double maxspeed=100;
double duration=100;
```

1.3 输出JSON文件

```
//设置输出文件名称
std::string outputFileName = "myFile.json";
```

1.4 模型路径初始化

```
//最后的文件名router全部小写要大写
std::string routerPath=netsimulyzer::models::ROUTER;
```

```
3   //输出模型路径检查
4   std::cout<<routerPath<<std::endl;
```

1.5 命令行读取命令

```
1   //设置命令行
2   CommandLine cmd;
3   //选择可以从命令行读取哪些参数
4   //例如，移动范围，速度范围，仿真时长
5   cmd.AddValue("命令行输入参数名,如duration=100中对duration的识别","对输入参数的描
    述",要关联的变量);
6   cmd.AddValue("minposition","Minimum X/Y position a Node may move
    to",minposition);
7   //agrc表示参数个数，命令行什么都不输入默认argc为1，argv为指向参数的指针
8   cmd.Parse(argc,argv);
9   //该命令会在参数满足特定条件时终止程序，并打印错误信息
10  NS_ABORT_MSG_IF(duration < 1.0, "仿真时间过短！" );
```

1.6 初始化节点

```
1   NodeContainer nodes;
2   //创建节点
3   nodes.Create(2);
4   //添加节点
5   nodes.Add(node);
```

1.7 初始化网络设备

```
1   NetDeviceContainer device;
2   device=csma.Install(nodes);
```

1.8 初始化建筑物节点

```
1   //创建一个建筑管理容器
2   BuildingContainer buildings;
3   //创建一个简单建筑物
4   Ptr<Building> simpleBuilding = CreateObject<Building>();
```

```
//设置建筑物有几层楼,这里设置为2层楼
simpleBuilding->SetNFloors(2);
//设置简单建筑物（长方体）的六元组
//六元组分别表示长方体横向、纵向、垂向的最小值和最大值，通常每层楼有20个单位长度
simpleBuilding->SetBoundaries({10.0,20.0,10.0,20.0,0.0,20.0});
//将该简单建筑物加入到建筑管理容器中
buildings.Add(simpleBuilding);
```

2. 设置网络模型/p2p/wifi/csma

2.1 设置点对点链路

```
//设置点对点助手
PointToPointHelper pointtopoint;
//设置数据传输率
pointtopoint.SetDeviceAttribute("DataRate",StringValue("5Mbps"));
//设置时延
pointtopoint.SetChannelAttribute("Delay",StringValue("2ms"));
device=pointtopoint.Install(nodes);
```

2.2 设置无线链路（无线链路中注意各设备之间距离不要拉的太远，这可能导致各设备之间无法连接）

```
//设置wifi信道助手,信道包含的各个参数值均为初始值default
YansWifiChannelHelper channel = YansWifiChannelHelper::Default();
//YansWifiPhyHelper设置信道物理层属性
YansWifiPhyHelper phy;
//配置NS-3中的无线信道属性，特别是范围传播损耗模型（RangePropagationLossModel）.
//其中MaxRange设置了节点之间的最大通信距离
//wifiChannel.AddPropagationLoss("ns3::RangePropagationLossModel",
"MaxRange", DoubleValue(500.0));

//设置信号发射功率,当AP离设备较远时应适当增大发射功率
//发射功率下限
phy.Set("TxPowerStart", DoubleValue(50.0));
//发射功率上限
phy.Set("TxPowerEnd", DoubleValue(50.0));

//将配置好的信道与物理层关联起来
phy.SetChannel(channel.Create());

```

```
18
19  //配置mac层和SSID
20  WifiMacHelper mac;
21  //创建一个SSID，用于标识无线网络的名称，这里设置为Ulraman Tiga
22  Ssid ssid = Ssid("Ulraman Tiga");
23
24
25  //设置wifi助手
26  WifiHelper wifi;
27  //wifi.SetStandard(WIFI_PHY_STANDARD_80211b);
28  //配置并安装STA设备
29  //设置MAC层为StaWifiMac类型，即Wi-Fi终端MAC，并配置了SSID和ActiveProbing属性
30  //设置为false表示禁用主动探测
31  // 设置为 Ad-hoc 模式
32  //Ad-hoc（自组网）模式下，每个节点都可以充当中继节点，直接转发其他节点的数据，因此不需要
    单独的路由器设备
33  //mac.SetType("ns3::AdhocWifiMac");//适用于自组网
34  mac.SetType("ns3::StaWifiMac", "Ssid", SsidValue(ssid), "ActiveProbing",
    BooleanValue(false));
35  /*
36  如果安装MAC层为ApWifiMac类型，则为接入点（AP）
37  mac.SetType("ns3::ApWifiMac", "Ssid", SsidValue(ssid));
38  */
39  //将前面设置好的phy、mac属性安装到节点容器nodes中，并返回到设备容器device中
40  device = wifi.Install(phy, mac,nodes);
```

2.3 设置有线链路

```
1  //设置有线助手
2  CsmaHelper csma;
3  //设置数据传输率
4  csma.SetChannelAttribute("DataRate",StringValue("100Mbps"));
5  //设置时延为6560纳秒，即6.56微秒
6  csma.SetChannelAttribute("Delay",TimeValue(NanoSeconds(6560)));
7  device=csma.Install(nodes)
```

3. 设置移动模型

3.1 定点移动模型

```
1  MobilityHelper mobility;
```

```
//定点模型
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
//安装模型
mobility.Install(nodes);
//设置定点位置
nodes.Get(0)->GetObject<MobilityModel>()->SetPosition(Vector(0.0, 0.0, 0.0));
// 节点坐标
```

3.2 （动点）初始化位置分配器

```
//随机分配节点位置助手
auto positionAllocator = CreateObject<RandomBoxPositionAllocator>();
//设置分配位置的范围
auto positionStream = CreateObject<UniformRandomVariable>();
//设置最小值和最大值
positionStream->SetAttribute("Min", DoubleValue(minNodePosition));
positionStream->SetAttribute("Max", DoubleValue(maxNodePosition));
//将这个范围通知到助手
positionAllocator->SetX(positionStream);//横向范围
positionAllocator->SetY(positionStream);//纵向范围
//确认助手分配位置的高度为Z=0，表示所有节点在Z=0的平面上移动
positionAllocator->SetAttribute("Z",
StringValue("ns3::ConstantRandomVariable[Constant=0.0])"));
```

3.3 （动点）初始化速度分配器

```
//设置分配速度的助手
auto velocityStream = CreateObject<UniformRandomVariable>();
//设置速度的范围
velocityStream->SetAttribute("Min", DoubleValue(minSpeed));
velocityStream->SetAttribute("Max", DoubleValue(maxSpeed));
```

3.4 （动点）根据建立好的位置和速度分配器建立动点模型

```
//创建模型助手
MobilityHelper mobility;
//指定移动模型参数
mobility.SetMobilityModel(
        "ns3::RandomDirection2dMobilityModel",//2d平面内随机方向移动
        "Bounds",//设置移动边界
```

```
        RectangleValue({min,max,min,max}),//分别设置横向和纵向的最大最小值
        "Speed",//设置速度属性
        PointerValue(velocityStream),//设置移动模型为之前设置好的速度分配器
        "Pause",//节点在改变方向时暂停时间
        StringValue("ns3::ConstantRandomVariable[Constant=1.0]")//每到达一个方向时间停止1秒钟
);
//为模型设置初始值/初始位置
mobility.SetPositionAllocator(positionAllocator);
//将该移动模型安装到节点/节点容器中
mobility.Install(nodes);
```

4. 安装协议和通信设备

4.1 设置网络栈

```
//初始化网络栈助手
InternetStackHelper stack;
//将助手安装到节点上
stack.Install(nodes);
```

4.2 设置路由协议

```
#include "ns3/aodv-helper.h"

InternetStackHelper stack;
AodvHelper aodv;
stack.SetRoutingHelper(aodv);  // 使用AODV作为路由协议
stack.Install(nodes);
/*
#include "ns3/olsr-helper.h"

OlsrHelper olsr;
stack.SetRoutingHelper(olsr);  // 使用OLSR作为路由协议
stack.Install(nodes);
*/
```

4.3 设置交换机

```
// 创建3个节点，两个终端节点和一个交换机节点
NodeContainer terminals;
terminals.Create(2); // 创建两个终端节点
Ptr<Node> switchNode = CreateObject<Node>(); // 创建一个交换机节点
// 安装互联网协议栈
InternetStackHelper internet;
internet.Install(terminals);
internet.Install(switchNode);
// 创建PointToPoint链路
PointToPointHelper p2p;
p2p.SetDeviceAttribute("DataRate", StringValue("100Mbps"));
p2p.SetChannelAttribute("Delay", StringValue("2ms"));
// 连接终端节点和交换机节点
NetDeviceContainer terminalDevices;
NetDeviceContainer switchDevices;
for (uint32_t i = 0; i < terminals.GetN(); ++i)
{
    NetDeviceContainer link = p2p.Install(terminals.Get(i), switchNode);
    terminalDevices.Add(link.Get(0));
    switchDevices.Add(link.Get(1));
}
// 在交换机节点上安装桥接设备
BridgeHelper bridge;
Ptr<NetDevice> bridgeDevice = bridge.Install(switchNode, switchDevices);
// 分配IP地址
Ipv4AddressHelper address;
address.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces = address.Assign(terminalDevices);
```

4.4 分配IP地址

```
//设置ipv4地址助手
Ipv4AddressHelper address;
//设置子网地址与子网掩码,助手会从192.168.1.1开始给节点分配地址
address.SetBase("192.168.1.0", "255.255.255.0");
//设置子网容器
Ipv4InterfaceContainer Interfaces;
//分配地址，并用子网容器记录节点和对应ipv4地址
Interfaces = address.Assign(Device);
Interfaces=address.Assign (apDevices);//给路由器网络设备分配地址
/*
//根据子网容器获得子网内某个节点的网络地址
Interfaces.GetAddress(0);
*/
//全局路由
```

```
15  Ipv4GlobalRoutingHelper::PopulateRoutingTables();
```

4.5 配置静态路由

```
1  // 创建静态路由
2  Ipv4StaticRoutingHelper staticRoutingHelper;
3  Ptr<Ipv4StaticRouting> staticRouting =
   staticRoutingHelper.GetStaticRouting(nodes.Get(0)->GetObject<Ipv4>());

4  // 添加静态路由
5  //AddNetworkRoute(目标ip地址，目标子网掩码，下一跳地址，出接口索引)
6  staticRouting->AddNetworkRoute(Ipv4Address("10.1.1.1"),
   Ipv4Mask("255.255.255.255"), Ipv4Address("10.1.1.2"), 1);
```

4.6 安装UDP客户机/服务器

```
1  //设置udp服务器助手，收发端口为9
2  UdpEchoServerHelper echoServer(9);
3  //将服务器安装到通信双方其中之一上
4  ApplicationContainer serverApps = echoServer.Install(nodes.Get(0));
5  //设置服务器的运行开始与结束的时间
6  serverApps.Start(Seconds(1.0));
7  serverApps.Stop(Seconds(10.0));
8  //设置udp客户机助手，告诉客户机助手服务器的ip地址与端口
9  UdpEchoClientHelper echoClient(csmaInterfaces1.GetAddress(0), 9);
10 //设置回显请求参数
11 //设置发送回显请求个数
12 echoClient.SetAttribute("MaxPackets", UintegerValue(1));
13 //设置每两个回显请求时间发送间隔
14 echoClient.SetAttribute("Interval", TimeValue(Seconds(1.0)));
15 //回显请求大小
16 echoClient.SetAttribute("PacketSize", UintegerValue(1024));
17
18 //将客户机安装到两个通信设备中的另一台设备上
19 ApplicationContainer clientApps = echoClient.Install(nodes.Get(1));
20
21 //设置客户机的运行时间
22 clientApps.Start(Seconds(2.0));
23 clientApps.Stop(Seconds(10.0));
24
25 //以LOG_INFO级别输出udp日志
26 LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
```

```
27  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
```

5. 仿真

5.1 监听位置和移动路径的变化/回调函数

```
1   //iter=NodeList::Begin()获取节点列表中的第一个节点
2   //NodeList::End()用于获取最后一个节点的下一个位置
3   //iter++获取下一个节点位置
4   for (auto iter = NodeList::Begin(); iter != NodeList::End(); iter++)
5       {
6           //获取节点的移动模型
7           auto m = (*iter)->GetObject<MobilityModel>();
8           //判断节点是否安装了移动模型，如果m为nullptr，则m没有附加移动模型
9           if (!m)
10              continue;
11          //连接轨迹回调
12          //这一行代码将移动性模型的"CourseChange"事件连接到回调函数CourseChanged
13          //回调函数在移动性模型的路径或未知发生变化时被触发
14          //MakeCallback(&CourseChanged)用于将CourseChanged函数作为回调传递
15          m->TraceConnectWithoutContext("CourseChange",
    MakeCallback(&CourseChanged));
16      }
```

5.2 监听数据包的发送与接收

```
1   Ptr<netsimulyzer::LogStream> eventLog;
2
3   void PacketSendLogger (Ptr<const Packet> packet) {
4       *eventLog << Simulator::Now().GetSeconds() << "s: Packet sent, size: " <<
    packet->GetSize() << " bytes\n";
5   }
6
7   void PacketReceiveLogger (Ptr<const Packet> packet) {
8       *eventLog << Simulator::Now().GetSeconds() << "s: Packet received, size: "
     << packet->GetSize() << " bytes\n";
9   }
10
11
12  // 设置回调函数，用于捕捉发送和接收事件
13  //记录客户机发送数据包
```

```
clientApps.Get(0)->TraceConnectWithoutContext("Tx", MakeCallback(&PacketSendLogger));
//记录服务器接收数据包
serverApps.Get(0)->TraceConnectWithoutContext("Rx", MakeCallback(&PacketReceiveLogger));
//记录服务器发送数据包
serverApps.Get(0)->TraceConnectWithoutContext("Tx", MakeCallback(&PacketSendLogger));
//记录客户机接收数据包
clientApps.Get(0)->TraceConnectWithoutContext("Rx", MakeCallback(&PacketReceiveLogger));
```

5.3 设置可视化助手orchestrator

```
//netsimulyzer中的Orchestrator对象用于管理可视化场景，并输出记录到指定的JSON文件中
//Orchestrator对象用于管理和协调仿真中的所有可视化组件
auto orchestrator = CreateObject<netsimulyzer::Orchestrator>(outputFileName);
//设置场景为矩形
 auto possibleNodeLocations = CreateObject<netsimulyzer::RectangularArea>(
        orchestrator,
        Rectangle{minNodePosition, maxNodePosition, minNodePosition, maxNodePosition});
//为区域设置名称
possibleNodeLocations->SetAttribute("Name", StringValue("Possible Node Locations"));
//为区域设置颜色
//SetAttribute("FillColor", ...)：设置矩形区域的填充颜色为浅绿色。
//Color3Value 是一个表示颜色的结构体，使用 RGB 颜色模型。
possibleNodeLocations->SetAttribute("FillColor", netsimulyzer::Color3Value{204u, 255u, 204u});

```

5.4 安装一些装饰物

```
auto decoration = CreateObject<netsimulyzer::Decoration>(orchestrator);
decoration->SetAttribute("Model", StringValue(netsimulyzer::models::CELL_TOWER_POLE_VALUE));
decoration->SetPosition(Vector3D(5.0, 5.0, 0.0));
```

5.5 输出场景配置/可视化场景配置到控制台

```
//全局声明eventLog，这里应该放在整个文件的开始
//netsimulyzer是一个命名空间，包含了与Netsimulyzer相关的所有类和函数
Ptr<netsimulyzer::LogStream> eventLog;

void CourseChanged(Ptr<const MobilityModel> model)
{
    //auto 关键字用于在声明变量时自动推导变量的类型。编译器会根据变量的初始化值来确定其类型
    //获取节点对象：model->GetObject<Node>() 返回与该移动模型关联的节点对象。
    //获取节点 ID：GetId() 方法返回该节点的唯一标识符（ID），并将其存储在 nodeId 变量中。
    const auto nodeId = model->GetObject<Node>()->GetId();
    //返回节点当前位置
    const auto position = model->GetPosition();
    //返回节点当前速度向量
    const auto velocity = model->GetVelocity();
    //Simulator::Now().GetMilliSeconds()获取当前的模拟时间，以毫秒为单位
    //<<操作符用于将信息格式化并写入日志流
    //position.x, position.y, position.z 和 velocity.x, velocity.y, velocity.z
    //用于输出节点的新位置和速度到日志流中
    *eventLog << Simulator::Now().GetMilliSeconds() << ": Node [" << nodeId
              << "] Course Change Position: [" << position.x << ", " << position.y << ", "
              << position.z << "] "
              << "Velocity [" << velocity.x << ", " << velocity.y << ", " << velocity.z << "]\n";
}

/*上面的函数在回调函数中调用
for (auto iter = NodeList::Begin(); iter != NodeList::End(); iter++)
    {
        auto m = (*iter)->GetObject<MobilityModel>();
        if (!m)
            continue;
        m->TraceConnectWithoutContext("CourseChange", MakeCallback(&CourseChanged));
    }

*/

//创建两个日志流对象 infoLog 和 eventLog，用于记录场景配置和事件信息
//声明该日志流属于orchestrator可视化助手
auto infoLog = CreateObject<netsimulyzer::LogStream>(orchestrator);
eventLog = CreateObject<netsimulyzer::LogStream>(orchestrator);

//记录场景配置，以下内容会被输出到netsimulyzer的控制台中
```

```
*infoLog << "----- Scenario Settings -----\n";


//记录移动范围
*infoLog << "Node Position Range: [" << minNodePosition << ',' << maxNodePosition << "]\n";
/*
也可以这样输出
*infoLog << "Node Position Range (X): [" << minXPosition << ", " << maxXPosition << "]\n";
*infoLog << "Node Position Range (Y): [" << minYPosition << ", " << maxYPosition << "]\n";
*/

//记录速度范围
*infoLog << "Node Speed Range: [" << minSpeed << ',' << maxSpeed << "]\n";
//记录模型的存储位置
*infoLog << "Models: Phone [" << phoneModelPath << "], Drone [" << spaceship << "]\n";
//记录仿真持续时间
*infoLog << "Scenario Duration (Seconds): " << duration << '\n';
```

5.6 设置节点的可视化/模型可视化

```
//设置节点配置助手
netsimulyzer::NodeConfigurationHelper nodeConfigHelper(orchestrator);

//EnableMotionTrail：设置为 true，表示在可视化中启用节点的运动轨迹。
nodeConfigHelper.Set("EnableMotionTrail", BooleanValue(true));

//为节点安装模型—>声明模型地址，根据前面的指定模型路径来设置

//有多个模型就多次执行下面两行代码
//ModelPath直接用前面设置好的droneModelPath或spaceship即可
nodeConfigHelper.Set("Model",StringValue(ModelPath));
//模型大小设置为原来的十倍
nodeConfigHelper.Set("Scale",DoubleValue(10.0));
//更多见官方文档Attributes
//利用节点配置助手将模型安装到节点（容器）上
nodeConfigHelper.Install(nodes);
```

5.7 设置建筑物的可视化/模型可视化

```
//设置建筑物配置助手
netsimulyzer::BuildingConfigurationHelper buildingConfigHelper(orchestrator);
//建筑物没有模型，因此直接安装到建筑物（容器）上即可
buildingConfigHepler.Install(buildings);
```

5.8 设置在某个时间点执行某个函数

```
void MyFunction(Ptr<Node> node)
{
        Ptr<netsimulyzer::NodeConfiguration> nodeConfig = node->GetObject<netsimulyzer::NodeConfiguration>();
        if (nodeConfig != nullptr) {
            // Display a transmission animation for each node
            nodeConfig->Transmit(Seconds(3), 50.0, netsimulyzer::GRAY_30);
        }
}

//其中2.0为时间，MyFunction为自己写的回调函数，node表示传入节点
Simulator::Schedule(Seconds(2.0), &MyFunction,node);
```

5.9 设置传播动画

```
// Transmission animation
//为某个节点容器安装传播动画
for (auto iter = nodes.Begin(); iter != nodes.End(); ++iter) {
        Ptr<Node> node = *iter;
        Ptr<netsimulyzer::NodeConfiguration> nodeConfig = node->GetObject<netsimulyzer::NodeConfiguration>();

        if (nodeConfig != nullptr) {
            // Display a transmission animation for each node
            nodeConfig->Transmit(Seconds(3), 50.0, netsimulyzer::GRAY_30);
        }
}

//为所有节点安装传播动画
for (auto iter = NodeList::Begin(); iter != NodeList::End(); ++iter) {
        Ptr<Node> node = *iter;
        Ptr<netsimulyzer::NodeConfiguration> nodeConfig = node->GetObject<netsimulyzer::NodeConfiguration>();

        if (nodeConfig != nullptr) {
```

```
            // Display a transmission animation for each node
            nodeConfig->Transmit(Seconds(3), 50.0, netsimulyzer::GRAY_30);
        }
}

//设置节点在发送数据包时展示传播动画


void PacketSendanimation(Ptr<const Packet> packet) {

}
clientApps.Get(0)->TraceConnectWithoutContext("Tx",
MakeCallback(&PacketSendanimation));
serverApps.Get(0)->TraceConnectWithoutContext("Tx",
MakeCallback(&PacketSendanimation));
```

6. 启动/结束 仿真

```
//设置仿真在指定时间结束
Simulator::Stop(Seconds(duration));

Simulator::Run();

//在日志中记录仿真结束的消息
*infoLog << "Scenario Finished\n";

Simulator::Destroy();
```

7. 如何动态调整模型的位置

```
//动态更新位置
void UpdatePosition(Ptr<Node> node, Vector position) {
    Ptr<MobilityModel> mobility = node->GetObject<MobilityModel>();
    mobility->SetPosition(position);
    std::cout << "Updated position to: " << position << std::endl;
}

// 设置移动模型为 ConstantPositionMobilityModel
MobilityHelper mobility;
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility.Install(node);
```

```
12
13  //5秒后移动到 (10, 0, 0)
14  Simulator::Schedule(Seconds(5.0), &UpdatePosition, node, Vector3D(10.0, 0.0,
    0.0));
15  //10秒后移动到 (20, 20, 0)
16  Simulator::Schedule(Seconds(10.0), &UpdatePosition, node, Vector3D(20.0,
    20.0, 0.0));
```