1.1 OMNeT++ 简介

OMNeT++，一个基于 eclipse 开发套件的开源网络仿真工具，目前主要在高校实验室进行一些网络仿真测试，对一些算法进行对比，它可以供使用者进行完成以下开发：- **C/C++** 开发 - 网络仿真程序设计

毫无疑问，基于 eclipse 的开发工具肯定能支持普通的 C/C++ 工程。另外，在 OMNeT++ 上网络仿真设计领域的优势在于，它是一个开源的项目，对大量的网络模型都提供代码支持。但是问题在于国内的确没有什么社区支持，出现问题只能自己解决，其实对于开源的项目大多存在这种问题，往往开源的项目，使用起来难度较大，开源项目往往比那些商业的软件开发难度较大，支持也较少，开源可不代表简单。 OMNeT++ 对初学者能力要求高，它假定使用者对编程有一定了解的，对 eclipse 开发环境也是特别熟悉的，另外这是一个网络仿真的软件，需要你对计算机网络有足够的认识，它提供了大量现有各种网络的仿真例子，如果你对网络认识足够强，那么这个软件你用起来会感到特别顺手。 目前有大量的开源仿真库用于 OMNeT++ 环境，拥有丰富的外文资料，官方将其分为两类，包括 Supported Models 和 Contributed Models：- Supported Models 模型库的开发处于激活状态，有开发者在维护，定期会推出新的版本。- Contributed Models 完成后只推出过一次或几次版本，目前没有人在维护。

1.2 OMNeT++ 开源库

下面简单介绍一下几种常见的开源库。

- INET

由 Simucraft 公司主持开发，用于仿真有线及无线网络。

应用层协议：

- HTTP、FTP、Telnet、不同优先级的 Video、Ping

传输层协议：

- TCP、UDP、RTP (RealtimeTransport Protocol)

网络层协议：

- IPv4、IPv6、ICMP、ARP、MPLS、LDP、RSVP、OSPF、Mobile IPV6、AODV、DSDV、DSR

数据链路层协议：

- Ethernet、PPP、IEEE 802.11、FDDI、Token Ring
- 官网：http://inet.omnetpp.org
- INETMANET

由 Simucraft 公司主持开发，用于仿真无线、有线网络，在 INET 的基础上增加了大量的 MANET 协议，INETMANET= INET+MANET，在 INET 的基础上增加：

- 802.11a,g:Ieee80211aMac, Ieee80211gMac, Ieee80211aRadioModel, Ieee80211gRadioModel
- Ieee80211Mesh,Ieee80211MeshMgmt
- radiomodels: TwoRayModel, ShadowingModel, qamMode
- Ns2MotionMobility
- ARP:global ARP cache
- AODV,DSDV, DSR, DYMO, OLSR
- 官网：http://inet.omnetpp.org
- Mobility Framework
- 由 Simucraft 公司主持开发
- 是一个无线传感器仿真模型库
- 绝大多数协议已经被 INET 吸收
- 官网：http://mobility-fw.sourceforge.net/hp/index.html
- SensorSimulator
- 美国路易斯安娜州立大学开发
- 用于仿真无线传感器网络
- 官网：http://csc.lsu.edu/sensor_web/
- Castalia
- 澳大利亚国家信息技术中心（NICTA）开发
- 是一个基于 OMNeT++ 的侧重于无线网络的仿真器

- 基于实测数据的高级 channel/radio 模型
- Radio 详细的状态转移，允许多传输功率电平
- 高度灵活的 physical process model
- 感应设备的噪声、偏差（bias）和功耗
- 节点时钟漂移，CPU 功耗
- 资源监控，如超出功率限制（如 CPU 或内存）
- 拥有大量可调参数的 mac 协议
- 用于设计优化和扩展
- 官网：https://github.com/boulis/Castalia
- OverSim
- 德国德国卡尔斯鲁厄大学开发
- 用于仿真点对点（p-to-p）协议，如 chard，GIA 等
- 官网：http://www.oversim.org

1.3 目录

本手册与现有的那两本书风格不同，我希望读者通过此手册可以快速的上手 OMNeT++，快速的掌握 OMNeT++ 提供的各种接口，目前包括以下内容：- [1] OMNeT++ 的安装 - [2] INET 库的安装 && INET 库的基本使用 - [3] OMNeT++ 个性化设置 - [4] OMNeT++ 工程设计技巧 - [5] cModule | cPar | cGate | cTopology 相关类使用 - [6] 仿真结果分析 - [7] 仿真错误记录

2.1 OMNeT++ 下载

OMNeT++ 可以直接从网上下载，网站地址是：https://www.omnetpp.org, 但是国内直接从该网站下载，下载较慢，同时时常在安装下载过程中出现下载中断的情况，导致前功尽弃，下载成功较难。

2.2 OMNeT++ 安装

2.2.1 安装准备

由于 OMNeT++ 支持多个操作系统环境的安装，包括 **MacOS**、**linux** 和 **Windows**，在这里只描述 **Windows** 环境下的安装。软件的安装说明肯定在软件的安装文件有说明，我们没有必要每次安装一个软件的时候都去百度一下软件安装的过程，作者的观点是对于一些破解较难，安装复杂的软件安装可以写写 blog，记录记录。我们可以在 OMNeT++ 的安装包下发现 readme 文件和 doc 目录下的 installguide，去看看吧，总会发现我们的安装执行步骤，掌握这种办法，断网了也能安装、无论过多久还能记得安装过程。好了，废话不多说了。下面是几个你在安装过程中可能会用到的命令：

- [1] **./configure**

在 PC 机上第一次安装的时候，需要根据配置文件配置一下具体我们需要的软件的功能：静态编译程序、依赖库路径、其他什么文件路径

- [2] **make**

在 PC 机上第一次安装的时候，需要根据配置文件配置一下具体我们需要的软件的功能：静态编译程序、依赖库路径、其他什么文件路径

- [3] **make clean**

清除前面安装过程中产生的中间二进制文件，这个命令主要用于重新安装软件的过程中，如果遇到 make 出错的问题，可以选择这个命令清除到二进制文件，然后在使用 make 命令编译安装（因为有些时候下载的安装包不是原始文件）。

2.2.2 图文并茂

其实这一部分没有说明必要，姑且就当作者无聊，还是想写写，我的原则就是坚持把故事讲得透彻明白，有些时候，在阅读别人博客的时候，老是会有很多疑问，其实博主以为读者懂，但读者的专业背景不一样，导致可能很简单的问题，还得下边留个言……好了，我们还是回到本节的话题上。　　以下三张图：

图 2-4-1 doc 目录

上图文件是 OMNeT++ 团队提供给开发者的基本帮助文档，我在写这个文档的时候，自我觉得还没有把这些文档都翻开看一遍，查阅这些文档久了，就会慢慢觉得这些资料本身已经够用了……我会在后续的文档中，描述一下 OMNeT++ 提供给我们的地图。

2.3 INET 库

2.3.1 INET 库的介绍

从一个初学者的角度，当安装 OMNeT++ 后，大多数的情况下是需要安装 INET 库的，这个集成库包含了丰富的仿真模型，多数时候，读者如果设计一个网络仿真程序，有不想重新编写代码，这时候，可以在 INET 下寻找是否有满足要求的 example，包括的网络有：

- adhoc
- aodv
- ethernet
- ipv6

等等，上面列举出的只是其中经常用到的一小部分，但是这也存在读者的不同研究背景，可能其中涉及的还不算很全。目前，我对于 INET 的使用较浅薄，水平还停留在调用 INET 库中的 ned 文件中的节点类型，或者其他诸如移动模型的水平上。在该小节，作者先为读者描述一下如何在 OMNeT++ 下快速的使用 INET 库和目前我经常使用的技巧。

2.3.2 INET 库的安装

通常有两种方法安装 INET，在安装之前，首先需要到: https://inet.omnetpp.org 下载合适的版本，由于前面的 OMNeT++ 使用的 5.2 的版本，这里我们可以选择 **inet-3.6.2**，下载结束以后，将 inet 解压到

omnetpp 的安装路径下的 samples 文件下，此时 inet 文件的路径可能是：**xxx/omnetpp-5.2/samples/inet** (解压 INET-3.6.2 文件后只有一个 inet 文件)。接下来，我们需要：- 方法一：命令窗口安装 INET

其实，如果需要为 omnetpp 安装新的插件或者库，都可以通过命令行的形式进行安装，甚至，你可以在命令行的环境下对编写好的网络进行编译和运行。　　我编写这个学习手册的原则，就是为读者提供一个学习 OMNeT++ 的地图，而不是特别详细的字典，可能我的水平还远远没有达到写一本学习 OMNeT++ 的大全。首先，安装这个 INET 库，我们到 inet 文件下看看有什么有用的文件没有，当然是先看看 **README.md** 了，这个文件提示我们安装请看：**INSTALL**，下面是这个 INSTALL 的英文：

```
If you are building from command line:
--------------------------------------
3. Change to the INET directory.

4. Type "make makefiles". This should generate the makefiles for you automatically.

5. Type "make" to build the inet executable (debug version). Use "make MODE=release"
   to build release version.

6. You can run specific examples by changing into the example's directory and executing "./
run"
```

当然，你可以选择 **mingwenv.cmd** 命令窗口，输入以上指令进行编译安装。上面的英文安装较为简洁，下面是我使用命令窗口安装的过程：

- [1] 在安装 INET 库之前，应先确保 OMNeT++ 已经安装成功。进入到 OMNeT++ 安装路径，找到 **mingwenv.cmd** 文件，双击执行，进入下图：

- [2] 接下来，使用命令：**cd samples/inet**，进入到 samples 下的 inet，另可使用 ls 命令查看当前 inet 文件下各子文件。
- [3] 然后，执行 **make makefiles** 命令生成编译整个 inet 库的 makefile 文件，结束以后输入命令 **make**。到这来，使用命令窗口编译 inet 库就结束了。
- [4] 最后需要在 OMNeT++ IDE 中 Project Explore 窗口空白处右击, 如图，使用 Import 功能导入已经编译好的 inet 库。过程如下图：

- 方法二：OMNeT++ 窗口安装

一样的，在 **INSTALL** 下命令行安装方式下面就是使用 IDE 的安装方式，这个 IDE 的使用方式就是将 INET 库使用 OMNeT++ 打开，当然此时库文件 inet 已经在 **samples** 文件下，我们需要做的就是打开 OMNeT++ IDE，然后导入整个 inet 工程。

```
If you are using the IDE:
-------------------------
3. Open the OMNeT++ IDE and choose the workspace where you have extracted the inet directo
   The extracted directory must be a subdirectory of the workspace dir.

4. Import the project using: File | Import | General | Existing projects into Workspace.
   Then select the workspace dir as the root directory, and be sure NOT to check the
   "Copy projects into workspace" box. Click Finish.

5. Open the project (if already not open) and wait until the indexer finishes.
   Now you can build the project by pressing CTRL-B (Project | Build all)

6. To run an example from the IDE open the example's directory in the Project Explorer vie
   find the corresponding omnetpp.ini file. Right click on it and select Run As / Simulati
   This should create a Launch Configuration for this example.

If the build was successful, you may try running the demo simulations.
Change into examples/ and type "./rundemo".
```

根据上面的步骤，需要点击：**File | Import | General | Existing projects into Workspace**，导入 inet 整个工程文件，对整个工程进行编译即可。

2.4 常规使用

2.4.1 导入工程

其实我觉得还是有必要把这一小节的内容加入其中，考虑了一下，这个软件的有些操作还是不太一样，可能初学者自己去找需要花大量的时间。 在学习如何导入工程前，先观察一张图：

图 2-4-1 IDE 视图

图 2-4-1 中，左边窗口为 **Project Explorer**，在软件安装首次打开 **IDE** 时，**Project Explorer** 中已经默认有相关的 **samples** 目录下的工程，如果开发者想要打开已有的工程，需要在 **Project Explorer** 窗口空白处右击鼠标，进行 **Import | General | Existing Projects into Workspace**，最后选择工程文件即可，不需要任何设置直接 **finish** 即可。其中相关的中间过程如下：

图 2-4-2 点击 Import 后视图

图 2-4-3 点击 Existing Projects into Workspace 后视图

前面已经描述了相关过程，需要注意的是保证工程文件不要放在有中文名的路径下，如果包括的中文路径，在后期编译工程时，可能在 **ned** 文件下出现大量错误，无法识别 **ned** 文件路径。

2.4.2 程序执行与调试

导入了工程，如何执行程序和调试还是很重要的，尤其是对于 OMNeT++ 工程，在 **omnetpp** 工程下有三种文件：**ned** 、**cpp** 和 **ini**，下面是这三种文件的介绍：**- ned** ：网络拓扑描述文件、简单节点模型和复合节点模型；**- cpp** ：cpp 文件为描述简单节点编程，定义简单节点各种行为；**- ini** ：ned 文件中相关参数的配置，在 ned 文件中一般会设置诸如节点数量的变量，一般有默认值，但是为了修改方便，可以在 ini 文件里边直接配置修改；**- msg** ：消息描述文件，会被 opp_msgc 转化成 *_m.cc/h 文件。

为了更好的说明以上三种文件在一个工作里边的关系，下面展示一张图：

图 2-4-4 编译与执行仿真流程

这张图还是比较重要的，尤其是当你发现你的 ned 或者 ini 文件不启作用的时候，可以根据上面的仿真程序流程思考一下找到 bug 所在。再这张图中，可以看出 **Simulation program** 就是我们工程生成的可执行文件 **exe**，也许你会发现在我们执行程序的时候有两种选择：

- **Local C/C++ Application**
- **OMNeT++ Simulation**

这两种方式执行仿真程序有何不同了，结合图 **2-4-4**，选择第一种执行方式其实就是执行的 **Simulation program**，但是这种执行方式运行的仿真程序没有加入 **ned** 文件和 **ini** 配置文件，因此就是模型节点参数没有配置好或者就没有配置。第二种执行方式就比较完整了，其模型加入了 **ned** 文件和 **ini** 配置文件。其他类似问题读者可自行揣测。　　前面介绍这么多，还是直接进入主题，我们执行 OMNeT++ 工程大致有三种方式：- [1] 直接右击工程文件**->RUN as->Local C/C++ Application** 或者 **OMNeT++ Simulation**；- [2] 选中描述网络的 **ned** 文件，右击执行上面一样的操作；- [3] 选择配置网络参数的 **omnetppp.ini** 文件，右击执行上面一样的操作。

对于上面执行工程的后两种方式一般都是可以的，但是对于第一种方式来说，需要执行 **exe** 文件直接在工程目录下，不能在工程目录的子文件中，否则就只能选择后面两种执行方式。　　关于 OMNeT++ 工程如何调试不再说明，其调试的方式与程序执行的方式相似，同时与其他程序的调试一样使用 **gdb** 调试，其中设置断点、单步调试或者进入函数内部等基本一样，以及添加观察变量。

3.1 初入 OMNeT++

欢迎来到第三章，本章主要介绍 OMNeT++ 官方已经提供的学习资料有哪些，并以 OMNeT++ 内一系类 tictoc 作为实例进行简单的设计说明，通过本章你可以快速的了解到如何学习 OMNeT++、掌握官方的学习资料和利用 OMNeT++ 可以做哪些事情。

3.2 学习 map

就目前学习 OMNeT++ 的资料来说，网上的资料有：- [1] 《omnet++ 中文使用手册》- [2] 《OMNeT++ 与网络仿真》- [3] 《OMNeT++ 网络仿真》（紫色）

目前较全的资料就上面三种，其中前两种参考价值比较好一些，其中第一本就是 OMNeT++ 官方提供的资料的翻译版，主要介绍范范的仿真程序设计，不能称其为学习教程，应该叫参考资料。第二本《OMNeT++ 与网络仿真》与第一本相比，在仿真程序的设计时更有价值一些，对部分函数接口有介绍，但是没有给出使用场景。其实到目前，作者认为还是官方提供的入门手册对初学者较友好一些，但是问题在于初学的时候我们不知道它的存在，包括我在初学的时候也是恍恍惚惚的，为了使读者在初学的时候就更好的利用这些资料，我在这里总结出官方到底提供了哪些资料。

3.2.1 OMNeT++ 文档与指导书

在 OMNeT++ 安装路径下，官方提供了较多的使用指南，大多数以网页的形式给出。第一个要介绍的就是包括安装手册在内的多个文档入口：

- 路径：**omnetpp-5.2/doc/index.html**

其内容包括从软件安装、初学 Tictoc 多个仿真例子、API 参考到提升篇：IDE 自定义指南和并行仿真指南等，详细如下：

介绍、指导手册 - 安装指导 - IDE 浏览 - TicToc 指导手册

文档 - 仿真手册 - IDE 用户指南 - API 参考书

其他 - IDE 开发者指导 - IDE 自定义指南 - 并行仿真指南 - NEDXML 接口函数

这里都是以中文的形式展现出官方提供的资料目录，而原目录都是以英文的形式给出。

3.2.2 tictoc 指导手册

tictoc 相当于程序中的 **hello world** 级别的例子，初学 OMNeT++ 一般通过仿真修改 **tictoc** 例子，其路径在软件的安装路径下，点击该路径下的 **index.html**：- 路径：**omnetpp-5.2/doc/tictoc-tutorial/index.html**

包括的内容如下：- 开始：一个简单的仿真模型（**tictoc1.ned txc1.cc omnetpp.ini**）- 仿真程序的执行和仿真 - 改进两个节点仿真模型（**tictoc9.ned txc9.cc omnetpp.ini**）- 一个复杂的网络（**tictoc13.ned, tictoc13.msg, txc13.cc, omnetpp.ini**）- 如何添加统计量（**tictoc17.ned, tictoc17.msg, txc17.cc, omnetpp.ini**）- 如何可视化观察仿真结果 - 如何添加参数（在 **omnetpp.ini** 中配置**.ned** 文件需要的参数）

对于以上资料是目前入学 OMNeT++ 较全系统的资料，从工程搭建、调试到添加统计量这些都是实际的网络仿真程序中一般会用到的，比如统计量，一般在网络中包括端到端延迟、入队排队时间、丢包数等。最后的仿真结果可视化观察，OMNeT++ 仿真程序结束后，在 out 文件下会生成仿真结果文件，OMNeT++ 提供可视化工具观察程序中统计的变量，可以转换成直方图和折线图，在后续会详细说明如何使用 OMNeT++ 提供的观察和分析仿真结果工具。

3.2.3 仿真手册

官方也提供了一个较详细的仿真手册，这里还是介绍一下这个手册。

- 路径：**omnetpp-5.2/doc/manual/index.html**

入口如下（绿色部分标出）

Simulation Manual 仿真手册提供了设计一个 OMNeT++ 工程各方面详细的介绍，从某种意义上来说，本手册也许就是 Simulation Manual 手册的一个子集，但是为了使本手册的意义更大，我将结合自己的几个月使用 OMNeT++ 的经验，提供一些在设计网络时可能会出现的问题，以及解决办法。

3.3 个性化 **IDE**

添加这一节，只是个人乐趣而已，我曾经修改代码高亮花了一个下午的时间，每次总是很难找到相关的设置地方，只能说 OMNeT++ 代码高亮设置放置的太隐秘了，到目前为止，我还是不能找到修改.ned 文件的高亮设置（似乎没有这个功能）。相关.cc 文件的设置地方下面会简单指明，其他更为详细的内容需要读者自行发挥。

3.3.1 CPP 高亮设置

首先，进入到 IDE 设置界面：Window 》》Preferences，如图：

cpp 高亮设置

从上图也可以看出，我们需要选择 c/c++》》Syntax Coloring，根据图中的窗口选择我们需要修改的高亮块设置。其中包括以下五种：

- Code: 代码块
- Assembly: 汇编块
- Comments: 注释块
- Preprocessor: 预处理器块
- Doxygen: 其他

你可以根据自己的爱好选择不同的块设置，可选择颜色、加粗、斜体或者下划线等。

3.3.2 其他设置

3.3.2.1 显示行号

在 OMNeT++ 下默认没有显示行号的，但是毕竟有这个毛病，不显示行号就写不下去程序。 如下图所示：

其他设置界面

在这一界面中，我们 Tab 键宽度、显示行号或者当前行背景, 其他设置读者可自行编辑看看。

4.1 循规蹈矩

在完成第五章后，考虑需要在之前加一章节关于 OMNeT++ 类说明，在这个仿真软件中，主要使用的语言是 C++，因此大多数数据类型是类或者结构，本章还是走其他技术书一样的老路线，注释这些数据类型，对类成员函数进行说明，可能与第五章有些重复的地方，但是其五章更多的偏向于实际应用，可能读者看过这里后，会发现 OMNeT++ 接口是真好用。

4.2 类说明

4.2.1 cModule

为了能更好的解释这个的库的使用，程序清单 4.1 为类 cModule 原型，cModule 类在 OMNeT++ 中表示一个节点的对象，这个节点可以是复合节点或者简单节点，通过这个类，程序员可以访问描述这个节点的.ned 文件中设置的参数，或者是由 omnetpp.ini 传入的参数。简而言之，我们最后就是面向这些类进行网络设计。

程序清单 4.1

```
class SIM_API cModule : public cComponent //implies noncopyable
{
    friend class cGate;
    friend class cSimulation;
    friend class cModuleType;
    friend class cChannelType;

  public:
    /*
     * 模块门的迭代器
     * Usage:
     * for (cModule::GateIterator it(module); !it.end(); ++it) {
     *      cGate *gate = *it;
     *      ...
     * }
     */
    class SIM_API GateIterator
    {
      ...
    };

    /*
```

```
     * 复合模块的子模块迭代器
     * Usage:
     * for (cModule::SubmoduleIterator it(module); !it.end(); ++it) {
     *     cModule *submodule = *it;
     *     ...
     * }
     */
    class SIM_API SubmoduleIterator
    {
      ...
    };

    /*
     * 模块信道迭代器
     * Usage:
     * for (cModule::ChannelIterator it(module); !it.end(); ++it) {
     *     cChannel *channel = *it;
     *     ...
     * }
     */
    class SIM_API ChannelIterator
    {
      ...
    };

  public:
    // internal: currently used by init
    void setRecordEvents(bool e)  {setFlag(FL_RECORD_EVENTS,e);}
    bool isRecordEvents() const  {return flags&FL_RECORD_EVENTS;}

  public:
#ifdef USE_OMNETPP4x_FINGERPRINTS
    // internal: returns OMNeT++ V4.x compatible module ID
    int getVersion4ModuleId() const { return version4ModuleId; }
#endif

    // internal: may only be called between simulations, when no modules exist
    static void clearNamePools();

    // internal utility function. Takes O(n) time as it iterates on the gates
    int gateCount() const;

    // internal utility function. Takes O(n) time as it iterates on the gates
    cGate *gateByOrdinal(int k) const;
    // internal: calls refreshDisplay() recursively
    virtual void callRefreshDisplay() override;

    // internal: return the canvas if exists, or nullptr if not (i.e. no create-on-demand)
    cCanvas *getCanvasIfExists() {return canvas;}

    // internal: return the 3D canvas if exists, or nullptr if not (i.e. no create-on-dema
    cOsgCanvas *getOsgCanvasIfExists() {return osgCanvas;}

  public:

    /** @name Redefined cObject member functions. */
    //@{

    /**
     * Calls v->visit(this) for each contained object.
     * See cObject for more details.
     */
    virtual void forEachChild(cVisitor *v) override;

    /**
     * Sets object's name. Redefined to update the stored fullName string.
     */
    virtual void setName(const char *s) override;

    /**
     * Returns the full name of the module, which is getName() plus the
     * index in square brackets (e.g. "module[4]"). Redefined to add the
     * index.
     */
    virtual const char *getFullName() const override;

    /**
     * Returns the full path name of the module. Example: <tt>"net.node[12].gen"</tt>.
     * The original getFullPath() was redefined in order to hide the global cSimulation
     * instance from the path name.
     */
    virtual std::string getFullPath() const override;

    /**
     * Overridden to add the module ID.
     */
    virtual std::string str() const override;
    //@}
```

```
/** @name Setting up the module. */
//@{

/**
 * Adds a gate or gate vector to the module. Gate vectors are created with
 * zero size. When the creation of a (non-vector) gate of type cGate::INOUT
 * is requested, actually two gate objects will be created, "gatename$i"
 * and "gatename$o". The specified gatename must not contain a "$i" or "$o"
 * suffix itself.
 *
 * CAUTION: The return value is only valid when a non-vector INPUT or OUTPUT
 * gate was requested. nullptr gets returned for INOUT gates and gate vectors.
 */
virtual cGate *addGate(const char *gatename, cGate::Type type, bool isvector=false);

/**
 * Sets gate vector size. The specified gatename must not contain
 * a "$i" or "$o" suffix: it is not possible to set different vector size
 * for the "$i" or "$o" parts of an inout gate. Changing gate vector size
 * is guaranteed NOT to change any gate IDs.
 */
virtual void setGateSize(const char *gatename, int size);

/*
 * 下面的接口是关于模块自己的信息
 */
// 复合模块还是简单模块
virtual bool isSimple() const;

/**
 * Redefined from cComponent to return KIND_MODULE.
 */
virtual ComponentKind getComponentKind() const override  {return KIND_MODULE;}

/**
 * Returns true if this module is a placeholder module, i.e.
 * represents a remote module in a parallel simulation run.
 */
virtual bool isPlaceholder() const  {return false;}

// 返回模块的父模块，对于系统模块，返回 nullptr
virtual cModule *getParentModule() const override;

/**
 * Convenience method: casts the return value of getComponentType() to cModuleType.
 */
cModuleType *getModuleType() const  {return (cModuleType *)getComponentType();}

// 返回模块属性，属性在运行时不能修改
virtual cProperties *getProperties() const override;

// 如何模块是使用向量的形式定义的，返回 true
bool isVector() const  {return vectorSize>=0;}

// 返回模块在向量中的索引
int getIndex() const  {return vectorIndex;}

// 返回这个模块向量的大小，如何该模块不是使用向量的方式定义的，返回 1
int getVectorSize() const  {return vectorSize<0 ? 1 : vectorSize;}

// 与 getVectorSize() 功能相似
_OPPDEPRECATED int size() const  {return getVectorSize();}


/*
 * 子模块相关功能
 */

// 检测该模块是否有子模块
virtual bool hasSubmodules() const {return firstSubmodule!=nullptr;}

// 寻找子模块 name，找到返回模块 ID，否则返回-1
// 如何模块采用向量形式定义，那么需要指明 index
virtual int findSubmodule(const char *name, int index=-1) const;

// 直接得到子模块 name 的指针，没有这个子模块返回 nullptr
// 如何模块采用向量形式定义，那么需要指明 index
virtual cModule *getSubmodule(const char *name, int index=-1) const;

/*
 * 一个更强大的获取模块指针的接口，通过路径获取
 *
 * Examples:
 *   "" means nullptr.
 *   "." means this module;
 *   "<root>" means the toplevel module;
 *   ".sink" means the sink submodule of this module;
 *   ".queue[2].srv" means the srv submodule of the queue[2] submodule;
 *   "^.host2" or ".^.host2" means the host2 sibling module;
 *   "src" or "<root>.src" means the src submodule of the toplevel module;
```

```
     *   "Net.src" also means the src submodule of the toplevel module, provided
     *   it is called Net.
     *
     *  @see cSimulation::getModuleByPath()
     */
    virtual cModule *getModuleByPath(const char *path) const;

    /*
     * 门的相关操作
     */

    /**
     * Looks up a gate by its name and index. Gate names with the "$i" or "$o"
     * suffix are also accepted. Throws an error if the gate does not exist.
     * The presence of the index parameter decides whether a vector or a scalar
     * gate will be looked for.
     */
    virtual cGate *gate(const char *gatename, int index=-1);

    /**
     * Looks up a gate by its name and index. Gate names with the "$i" or "$o"
     * suffix are also accepted. Throws an error if the gate does not exist.
     * The presence of the index parameter decides whether a vector or a scalar
     * gate will be looked for.
     */
    const cGate *gate(const char *gatename, int index=-1) const {
        return const_cast<cModule *>(this)->gate(gatename, index);
    }


    /**
     * Returns the "$i" or "$o" part of an inout gate, depending on the type
     * parameter. That is, gateHalf("port", cGate::OUTPUT, 3) would return
     * gate "port$o[3]". Throws an error if the gate does not exist.
     * The presence of the index parameter decides whether a vector or a scalar
     * gate will be looked for.
     */
    const cGate *gateHalf(const char *gatename, cGate::Type type, int index=-1) const {
        return const_cast<cModule *>(this)->gateHalf(gatename, type, index);
    }

    // 检测是否有门
    virtual bool hasGate(const char *gatename, int index=-1) const;

    // 寻找门，如果没有返回-1，找到返回门 ID
```

```
virtual int findGate(const char *gatename, int index=-1) const;

// 通过 ID 得到门地址，目前我还没有用到过
const cGate *gate(int id) const {return const_cast<cModule *>(this)->gate(id);}

// 删除一个门（很少用）
virtual void deleteGate(const char *gatename);


//返回模块门的名字，只是基本名字 (不包括向量门的索引, "[]" or the "$i"/"$o")
virtual std::vector<const char *> getGateNames() const;

// 检测门（向量门）类型，可以标明"$i","$o"
virtual cGate::Type gateType(const char *gatename) const;

// 检测是否是向量门，可以标明"$i","$o"
virtual bool isGateVector(const char *gatename) const;

// 得到门的大小，可以指明"$i","$o"
virtual int gateSize(const char *gatename) const;

// 对于向量门，返回 gate0 的 ID 号
// 对于标量 ID，返回 ID
// 一个公式：ID = gateBaseId + index
// 如果没有该门，抛出一个错误
virtual int gateBaseId(const char *gatename) const;

/**
 * For compound modules, it checks if all gates are connected inside
 * the module (it returns <tt>true</tt> if they are OK); for simple
 * modules, it returns <tt>true</tt>. This function is called during
 * network setup.
 */
virtual bool checkInternalConnections() const;

/**
 * This method is invoked as part of a send() call in another module.
 * It is called when the message arrives at a gates in this module which
 * is not further connected, that is, the gate's getNextGate() method
 * returns nullptr. The default, cModule implementation reports an error
 * ("message arrived at a compound module"), and the cSimpleModule
 * implementation inserts the message into the FES after some processing.
 */
virtual void arrived(cMessage *msg, cGate *ongate, simtime_t t);
//@}
```

```
/*
 * 公用的
 */
// 在父模块中寻找某个参数，没找到抛出 cRuntimeError
virtual cPar& getAncestorPar(const char *parname);

/**
 * Returns the default canvas for this module, creating it if it hasn't
 * existed before.
 */
virtual cCanvas *getCanvas() const;

/**
 * Returns the default 3D (OpenSceneGraph) canvas for this module, creating
 * it if it hasn't existed before.
 */
virtual cOsgCanvas *getOsgCanvas() const;

// 设置是否在此模块的图形检查器上请求内置动画。
virtual void setBuiltinAnimationsAllowed(bool enabled) {setFlag(FL_BUILTIN_ANIMATIONS,

/**
 * Returns true if built-in animations are requested on this module's
 * graphical inspector, and false otherwise.
 */
virtual bool getBuiltinAnimationsAllowed() const {return flags & FL_BUILTIN_ANIMATIONS
//@}

/** @name Public methods for invoking initialize()/finish(), redefined from cComponent
 * initialize(), numInitStages(), and finish() are themselves also declared in
 * cComponent, and can be redefined in simple modules by the user to perform
 * initialization and finalization (result recording, etc) tasks.
 */
//@{
/**
 * Interface for calling initialize() from outside.
 */
virtual void callInitialize() override;

/**
 * Interface for calling initialize() from outside. It does a single stage
 * of initialization, and returns <tt>true</tt> if more stages are required.
 */
virtual bool callInitialize(int stage) override;
```

```
    /**
     * Interface for calling finish() from outside.
     */
    virtual void callFinish() override;


    /*
     * 动态模块创建
     */

    /**
     * Creates a starting message for modules that need it (and recursively
     * for its submodules).
     */
    virtual void scheduleStart(simtime_t t);

    // 删除自己
    virtual void deleteModule();

    // 移动该模块到另一个父模块下，一般用于移动场景。规则较复杂，可到原头文件查看使用说明
    virtual void changeParentTo(cModule *mod);
};
```

cModule 是 OMNeT++ 中用于代表一个模块的对象实体，如果你在编写网络仿真代码时，这个模块可以是简单模块或者复合模块，当需要得到这个模块相关属性时可以考虑到这个 **cModule** 类里边找找，说不定有意外的惊喜，也许有现成的函数实现你需要的功能。下面将这个类原型解剖看看：

- 迭代器：**GateIterator**

```
usage:
for (cModule::GateIterator it(module); !it.end(); ++it) {
         cGate *gate = *it;
         ...
}
```

该迭代器可用于遍历模块 **module** 的门向量，得到该门可用于其他作用。

- 迭代器：**SubmoduleIterator**

```
usage:
for (cModule::SubmoduleIterator it(module); !it.end(); ++it) {
        cModule *submodule = *it;
        ...
}
```

对于一个复合模块，包括多个简单模块或者复合模块，可使用该迭代器进行遍历操作，在第五章涉及到这个迭代器的使用。

- 迭代器：**ChannelIterator**

```
usage:
for (cModule::ChannelIterator it(module); !it.end(); ++it) {
        cChannel *channel = *it;
        ...
}
```

可用于遍历该模块的所有的信道。

4.2.2 cPar

cPar 同样是我们设置网络时不可避免的类，通过 cPar 得到节点在网络拓扑文件和配置文件中设置的参数，浏览完 cPar 所有成员函数，可以看出 cPar 基本提供了网络设计者想要的所有数据转换接口。

```
class SIM_API cPar : public cObject
{
    friend class cComponent;
  public:
    enum Type {
        BOOL = 'B',
        DOUBLE = 'D',
        LONG = 'L',
        STRING = 'S',
        XML = 'X'
    };

  private:
    cComponent *ownerComponent;
    cParImpl *p;
    cComponent *evalContext;

  private:
    // private constructor and destructor -- only cComponent is allowed to create paramete
    cPar() {ownerComponent = evalContext = nullptr; p = nullptr;}
    virtual ~cPar();
    // internal, called from cComponent
    void init(cComponent *ownercomponent, cParImpl *p);
    // internal
    void moveto(cPar& other);
    // internal: called each time before the value of this object changes.
    void beforeChange();
    // internal: called each time after the value of this object changes.
    void afterChange();

  public:
    // internal, used by cComponent::finalizeParameters()
    void read();
    // internal, used by cComponent::finalizeParameters()
    void finalize();
    // internal: applies the default value if there is one
    void acceptDefault();
    // internal
    void setImpl(cParImpl *p);
    // internal
    cParImpl *impl() const {return p;}
    // internal
    cParImpl *copyIfShared();

#ifdef SIMFRONTEND_SUPPORT
    // internal
    virtual bool hasChangedSince(int64_t lastRefreshSerial);
#endif

  public:
    /** @name Redefined cObject methods */
    //@{
    /**
     * Assignment operator.
     */
    void operator=(const cPar& other);

    // 返回参数的名字
    virtual const char *getName() const override;
```

```
// 以字符串的形式返回参数
virtual std::string str() const override;

/**
 * Returns the component (module/channel) this parameter belongs to.
 * Note: return type is cObject only for technical reasons, it can be
 * safely cast to cComponent.
 */
virtual cObject *getOwner() const override; // note: cannot return cComponent* (covari

/**
 * Calls v->visit(this) for contained objects.
 * See cObject for more details.
 */
virtual void forEachChild(cVisitor *v) override;
//@}

/** @name Type, flags. */
//@{
/**
 * Returns the parameter type
 */
Type getType() const;

/**
 * Returns the given type as a string.
 */
static const char *getTypeName(Type t);

/**
 * Returns true if the stored value is of a numeric type.
 */
bool isNumeric() const;

/**
 * Returns true if this parameter is marked in the NED file as "volatile".
 * This flag affects the operation of setExpression().
 */
bool isVolatile() const;

/**
 * Returns false if the stored value is a constant, and true if it is
 * an expression. (It is not examined whether the expression yields
 * a constant value.)
```

```
     */
    bool isExpression() const;

    /**
     * Returns true if the parameter value expression is shared among several
     * modules to save memory. This flag is purely informational, and whether
     * a parameter is shared or not does not affect operation at all.
     */
    bool isShared() const;

    /**
     * Returns true if the parameter is assigned a value, and false otherwise.
     * Parameters of an already initialized module or channel are guaranteed to
     * assigned, so this method will return true for them.
     */
    bool isSet() const;

    /**
     * Returns true if the parameter is set (see isSet()) or contains a default
     * value, and false otherwise. Parameters of an already initialized module or
     * channel are guaranteed to be assigned, so this method will return true for them.
     */
    bool containsValue() const;

    /**
     * Return the properties for this parameter. Properties cannot be changed
     * at runtime.
     */
    cProperties *getProperties() const;
    //@}

    /** @name Setter functions. Note that overloaded assignment operators also exist. */
    //@{

    /**
     * Sets the value to the given bool value.
     */
    cPar& setBoolValue(bool b);

    /**
     * Sets the value to the given long value.
     */
    cPar& setLongValue(long l);

    /**
```

```
     * Sets the value to the given double value.
     */
    cPar& setDoubleValue(double d);

    /**
     * Sets the value to the given string value.
     * The cPar will make its own copy of the string. nullptr is also accepted
     * and treated as an empty string.
     */
    cPar& setStringValue(const char *s);

    /**
     * Sets the value to the given string value.
     */
    cPar& setStringValue(const std::string& s)  {setStringValue(s.c_str()); return *this;}

    /**
     * Sets the value to the given cXMLElement.
     */
    cPar& setXMLValue(cXMLElement *node);

    /**
     * Sets the value to the given expression. This object will assume
     * the responsibility to delete the expression object.
     *
     * The evalcontext parameter determines the module or channel in the
     * context of which the expression will be evaluated. If evalcontext
     * is nullptr, the owner of this parameter will be used.
     *
     * Note: if the parameter is marked as non-volatile (isVolatile()==false),
     * one should not set an expression as value. This is not enforced
     * by cPar though.
     *
     * @see getOwner(), getEvaluationContext(), setEvaluationContext()
     */
    cPar& setExpression(cExpression *e, cComponent *evalcontext=nullptr);

    /**
     * If the parameter contains an expression (see isExpression()), this method
     * sets the evaluation context for the expression.
     *
     * @see getEvaluationContext(), isExpression(), setExpression()
     */
    void setEvaluationContext(cComponent *ctx)  {evalContext = ctx;}
    //@}
```

```
/** @name Getter functions. Note that overloaded conversion operators also exist. */
//@{

/**
 * Returns value as a boolean. The cPar type must be BOOL.
 */
bool boolValue() const;

/**
 * Returns value as long. The cPar type must be LONG or DOUBLE.
 */
long longValue() const;

/**
 * Returns value as double. The cPar type must be LONG or DOUBLE.
 */
double doubleValue() const;

/**
 * Returns the parameter's unit ("s", "mW", "Hz", "bps", etc),
 * as declared with the @unit property of the parameter in NED,
 * or nullptr if no unit was specified. Unit is only valid for LONG and DOUBLE
 * types.
 */
const char *getUnit() const;

/**
 * Returns value as const char *. The cPar type must be STRING.
 * This method may only be invoked when the parameter's value is a
 * string constant and not the result of expression evaluation, otherwise
 * an error is thrown. This practically means this method cannot be used
 * on parameters declared as "volatile string" in NED; they can only be
 * accessed using stdstringValue().
 */
const char *stringValue() const;

/**
 * Returns value as string. The cPar type must be STRING.
 */
std::string stdstringValue() const;

/**
 * Returns value as pointer to cXMLElement. The cPar type must be XML.
 *
```

```
     * The lifetime of the returned object tree is undefined, but it is
     * valid at least until the end of the current simulation event or
     * initialize() call. Modules are expected to process their XML
     * configurations at once (within one event or within initialize()),
     * and not hang on to pointers returned from this method. The reason
     * for the limited lifetime is that this method may return pointers to
     * objects stored in an internal XML document cache, and the simulation
     * kernel reserves the right to discard cached XML documents at any time
     * to free up memory, and re-load them on demand (i.e. when xmlValue() is
     * called again).
     */
    cXMLElement *xmlValue() const;

    /**
     * Returns pointer to the expression stored by the object, or nullptr.
     */
    cExpression *getExpression() const;

    /**
     * If the parameter contains an expression, this method returns the
     * module or channel in the context of which the expression will be
     * evaluated. (The context affects the resolution of parameter
     * references, and NED operators like <tt>index</tt> or <tt>sizeof()</tt>.)
     * If the parameter does not contain an expression, the return value is
     * undefined.
     *
     * @see isExpression(), setEvaluationContext()
     */
    cComponent *getEvaluationContext() const  {return evalContext;}
    //@}

    /** @name Miscellaneous utility functions. */
    //@{
    /**
     * For non-const values, replaces the stored expression with its
     * evaluation.
     */
    void convertToConst();

    /**
     * Converts the value from string, and stores the result.
     * If the text cannot be parsed, an exception is thrown, which
     * can be caught as std::runtime_error& if necessary.
     *
     * Note: this method understands expressions too, but does NOT handle
```

```
 * the special values "default" and "ask".
 */
void parse(const char *text);
//@}

/** @name Overloaded assignment and conversion operators. */
//@{

/**
 * Equivalent to setBoolValue().
 */
cPar& operator=(bool b)  {return setBoolValue(b);}

/**
 * Converts the argument to long, and calls setLongValue().
 */
cPar& operator=(char c)  {return setLongValue((long)c);}

/**
 * Converts the argument to long, and calls setLongValue().
 */
cPar& operator=(unsigned char c)  {return setLongValue((long)c);}

/**
 * Converts the argument to long, and calls setLongValue().
 */
cPar& operator=(int i)  {return setLongValue((long)i);}

/**
 * Converts the argument to long, and calls setLongValue().
 */
cPar& operator=(unsigned int i)  {return setLongValue((long)i);}

/**
 * Converts the argument to long, and calls setLongValue().
 */
cPar& operator=(short i)  {return setLongValue((long)i);}

/**
 * Converts the argument to long, and calls setLongValue().
 */
cPar& operator=(unsigned short i)  {return setLongValue((long)i);}

/**
 * Equivalent to setLongValue().
 */
cPar& operator=(long l)  {return setLongValue(l);}

/**
 * Converts the argument to long, and calls setLongValue().
 */
cPar& operator=(unsigned long l) {return setLongValue((long)l);}

/**
 * Equivalent to setDoubleValue().
 */
cPar& operator=(double d)  {return setDoubleValue(d);}

/**
 * Converts the argument to double, and calls setDoubleValue().
 */
cPar& operator=(long double d)  {return setDoubleValue((double)d);}

// 等同于 setStringValue() 函数
cPar& operator=(const char *s)  {return setStringValue(s);}

// 等同于 setStringValue() 函数
cPar& operator=(const std::string& s)  {return setStringValue(s);}

 // 等同于 setXMLValue() 函数
cPar& operator=(cXMLElement *node)  {return setXMLValue(node);}

operator bool() const  {return boolValue();}

operator char() const  {return (char)longValue();}

operator unsigned char() const  {return (unsigned char)longValue();}

operator int() const  {return (int)longValue();}

operator unsigned int() const  {return (unsigned int)longValue();}

operator short() const  {return (short)longValue();}

operator unsigned short() const  {return (unsigned short)longValue();}

// 返回 long 值，与 longValue() 相同
operator long() const  {return longValue();}

/**
```

```
    // 调用 longValue()，转换结果为 unsigned long 类型
     */
    operator unsigned long() const  {return longValue();}

    // 返回 double 值，与 doubleValue() 相同
    operator double() const  {return doubleValue();}

    /**
    // 调用 doubleValue()，将结果转换成 long double 类型返回
     */
    operator long double() const  {return doubleValue();}

    // 与 stringValue()
    operator const char *() const  {return stringValue();}

    // 与 stdstringValue() 功能一样
    operator std::string() const  {return stdstringValue();}

    // 与 xmlVlaue() 等同。注意：返回对象树的生命周期被限制了，具体看 xmlValue 说明。
    operator cXMLElement *() const  {return xmlValue();}
};
```

4.2.3 cGate

如果你需要在网络仿真运行时，动态实现两个节点之间的连接或者断开，那么你就需要在程序中用到这个类。

```
class SIM_API cGate : public cObject, noncopyable
{
    friend class cModule;
    friend class cModuleGates;
    friend class cPlaceholderModule;

  public:
    /**
     * Gate type
     */
    enum Type {
        NONE = 0,
        INPUT = 'I',
        OUTPUT = 'O',
        INOUT = 'B'
    };

  protected:
    // internal
    struct SIM_API Name
    {
        opp_string name;  // "foo"
        opp_string namei; // "foo$i"
        opp_string nameo; // "foo$o"
        Type type;
        Name(const char *name, Type type);
        bool operator<(const Name& other) const;
    };

  public:
    // Internal data structure, only public for technical reasons (GateIterator).
    // One instance per module and per gate vector/gate pair/gate.
    // Note: gate name and type are factored out to a global pool.
    // Note2: to reduce sizeof(Desc), "size" might be stored in input.gatev[0],
    // although it might not be worthwhile the extra complication and CPU cycles.
    //
    struct Desc
    {
        cModule *owner;
        Name *name;  // pooled (points into cModule::namePool)
        int vectorSize; // gate vector size, or -1 if scalar gate; actually allocated size
        union Gates { cGate *gate; cGate **gatev; };
        Gates input;
        Gates output;

        Desc() {owner=nullptr; vectorSize=-1; name=nullptr; input.gate=output.gate=nullptr
        bool inUse() const {return name!=nullptr;}
        Type getType() const {return name->type;}
        bool isVector() const {return vectorSize>=0;}
        const char *nameFor(Type t) const {return (t==INOUT||name->type!=INOUT) ? name->nam
        int indexOf(const cGate *g) const {return (g->pos>>2)==-1 ? 0 : g->pos>>2;}
        bool deliverOnReceptionStart(const cGate *g) const {return g->pos&2;}
        Type getTypeOf(const cGate *g) const {return (g->pos&1)==0 ? INPUT : OUTPUT;}
        bool isInput(const cGate *g) const {return (g->pos&1)==0;}
        bool isOutput(const cGate *g) const {return (g->pos&1)==1;}
        int gateSize() const {return vectorSize>=0 ? vectorSize : 1;}
        void setInputGate(cGate *g) {ASSERT(getType()!=OUTPUT && !isVector()); input.gate=g
        void setOutputGate(cGate *g) {ASSERT(getType()!=INPUT && !isVector()); output.gate=
        void setInputGate(cGate *g, int index) {ASSERT(getType()!=OUTPUT && isVector()); in
        void setOutputGate(cGate *g, int index) {ASSERT(getType()!=INPUT && isVector()); ou
```

```
      static int capacityFor(int size) {return size<8 ? (size+1)&~1 : size<32 ? (size+3)
    };

  protected:
    Desc *desc; // descriptor of the gate or gate vector, stored in cModule
    int pos;    // b0: input(0) or output(1); b1: deliverOnReceptionStart bit;
                // rest (pos>>2): array index, or -1 if scalar gate

    int connectionId;   // uniquely identifies the connection between *this and *nextgatep
    cChannel *channel;  // channel object (if exists)
    cGate *prevGate;    // previous and next gate in the path
    cGate *nextGate;

    static int lastConnectionId;

  protected:
    // internal: constructor is protected because only cModule is allowed to create instan
    explicit cGate();

    // also protected: only cModule is allowed to delete gates
    virtual ~cGate();

    // internal
    static void clearFullnamePool();

    // internal
    void installChannel(cChannel *chan);

    // internal
    void checkChannels() const;

#ifdef SIMFRONTEND_SUPPORT
    // internal
    virtual bool hasChangedSince(int64_t lastRefreshSerial);
#endif

  public:
    /** @name Redefined cObject member functions */
    //@{
    /*
     * 例如返回门 out
     */
    virtual const char *getName() const override;

    /*
```

```
     * 与 getName() 不同，需要返回门索引，例如 out[4]
     */
    virtual const char *getFullName() const override;

    /**
     * Calls v->visit(this) for each contained object.
     * See cObject for more details.
     */
    virtual void forEachChild(cVisitor *v) override;

    /**
     * Produces a one-line description of the object's contents.
     * See cObject for more details.
     */
    virtual std::string str() const override;

    /**
     * Returns the owner module of this gate.
     */
    virtual cObject *getOwner() const override; // note: cannot return cModule* (covariant
    //@}

    /**
     * This function is called internally by the send() functions and
     * channel classes' deliver() to deliver the message to its destination.
     * A false return value means that the message object should be deleted
     * by the caller. (This is used e.g. with parallel simulation, for
     * messages leaving the partition.)
     */
    virtual bool deliver(cMessage *msg, simtime_t at);

    /** @name Connecting the gate. */
    //@{
    /**
     * Connects the gate to another gate, using the given channel object
     * (if one is specified). This method can be used to manually create
     * connections for dynamically created modules.
     *
     * This method invokes callInitialize() on the channel object, unless the
     * compound module containing this connection is not yet initialized
     * (then it assumes that this channel will be initialized as part of the
     * compound module initialization process.) To leave the channel
     * uninitialized, specify true for the leaveUninitialized parameter.
     *
     * If the gate is already connected, an error will occur. The gate
```

```
 * argument cannot be nullptr, that is, you cannot use this function
 * to disconnect a gate; use disconnect() for that.
 *
 * Note: When you set channel parameters after channel initialization,
 * make sure the channel class is implemented so that the changes take
 * effect; i.e. the channel should either override and properly handle
 * handleParameterChange(), or should not cache any values from parameters.
 */
cChannel *connectTo(cGate *gate, cChannel *channel=nullptr, bool leaveUninitialized=fa

/**
 * Disconnects the gate, and also deletes the associated channel object
 * if one has been set. disconnect() must be invoked on the source gate
 * ("from" side) of the connection.
 *
 * The method has no effect if the gate is not connected.
 */
void disconnect();

/**
 * Disconnects the gate, then connects it again to the same gate, with the
 * given channel object (if not nullptr). The gate must be connected.
 *
 * @see connectTo()
 */
cChannel *reconnectWith(cChannel *channel, bool leaveUninitialized=false);
//@}

/** @name Information about the gate. */
//@{
/**
 * Returns the gate name without index and potential "$i"/"$o" suffix.
 */
const char *getBaseName() const;

/**
 * Returns the suffix part of the gate name ("$i", "$o" or "").
 */
const char *getNameSuffix() const;

/**
 * Returns the properties for this gate. Properties cannot be changed
 * at runtime.
 */
cProperties *getProperties() const;
```

```
/**
 * Returns the gate's type, cGate::INPUT or cGate::OUTPUT. (It never returns
 * cGate::INOUT, because a cGate object is always either the input or
 * the output half of an inout gate ("name$i" or "name$o").
 */
Type getType() const  {return desc->getTypeOf(this);}

/**
 * Returns the given type as a string.
 */
static const char *getTypeName(Type t);

/**
 * Returns a pointer to the owner module of the gate.
 */
cModule *getOwnerModule() const;

/**
 * Returns the gate ID, which uniquely identifies the gate within the
 * module. IDs are guaranteed to be contiguous within a gate vector:
 * <tt>module->gate(id+index) == module->gate(id)+index</tt>.
 *
 * Gate IDs are stable: they are guaranteed not to change during
 * simulation. (This is a new feature of \opp 4.0. In earlier releases,
 * gate IDs could change when the containing gate vector was resized.)
 *
 * Note: As of \opp 4.0, gate IDs are no longer small integers, and
 * cannot be used for iterating over the gates of a module.
 * Use cModule::GateIterator for iteration.
 */
int getId() const;

/**
 * Returns true if the gate is part of a gate vector.
 */
bool isVector() const  {return desc->isVector();}

/**
 * If the gate is part of a gate vector, returns the ID of the first
 * element in the gate vector. Otherwise, it returns the gate's ID.
 */
int getBaseId() const;

/**
```

```
     * If the gate is part of a gate vector, returns the gate's index in the vector.
     * Otherwise, it returns 0.
     */
    int getIndex() const  {return desc->indexOf(this);}

    /**
     * If the gate is part of a gate vector, returns the size of the vector.
     * For non-vector gates it returns 1.
     *
     * The gate vector size can also be obtained by calling the cModule::gateSize().
     */
    int getVectorSize() const  {return desc->gateSize();}

    /**
     * Alias for getVectorSize().
     */
    int size() const  {return getVectorSize();}

    /**
     * Returns the channel object attached to this gate, or nullptr if there is
     * no channel. This is the channel between this gate and this->getNextGate(),
     * that is, channels are stored on the "from" side of the connections.
     */
    cChannel *getChannel() const  {return channel;}

    /**
     * This method may only be invoked on input gates of simple modules.
     * Messages with nonzero length then have a nonzero
     * transmission duration (and thus, reception duration on the other
     * side of the connection). By default, the delivery of the message
     * to the module marks the end of the reception. Setting this bit will cause
     * the channel to deliver the message to the module at the start of the
     * reception. The duration that the reception will take can be extracted
     * from the message object, by its getDuration() method.
     */
    void setDeliverOnReceptionStart(bool d);

    /**
     * Returns whether messages delivered through this gate will mark the
     * start or the end of the reception process (assuming nonzero message length
     * and data rate on the channel.)
     *
     * @see setDeliverOnReceptionStart()
     */
    bool getDeliverOnReceptionStart() const  {return pos&2;}
```

```
//@}

/** @name Transmission state. */
//@{
/**
 * Typically invoked on an output gate, this method returns <i>the</i>
 * channel in the connection path that supports datarate (as determined
 * by cChannel::isTransmissionChannel(); it is guaranteed that there can be
 * at most one such channel per path). If there is no such channel,
 * an error is thrown.
 *
 * This method only checks the segment of the connection path that
 * <i>starts</i> at this gate, so, for example, it is an error to invoke
 * it on a simple module input gate.
 *
 * Note: this method searches the connection path linearly, so at
 * performance-critical places it may be better to cache its return
 * value (provided that connections are not removed or created dynamically
 * during simulation.)
 *
 * @see cChannel::isTransmissionChannel()
 */
cChannel *getTransmissionChannel() const;

/**
 * Like getTransmissionChannel(), but returns nullptr instead of throwing
 * an error if there is no transmission channel in the path.
 */
cChannel *findTransmissionChannel() const;

/**
 * Typically invoked on an input gate, this method searches the reverse
 * path (i.e. calls getPreviousGate() repeatedly) for the transmission
 * channel. It is guaranteed that there can be at most one such channel
 * per path. If no transmission channel is found, the method throws an error.
 *
 * @see getTransmissionChannel(), cChannel::isTransmissionChannel()
 */
cChannel *getIncomingTransmissionChannel() const;

/**
 * Like getIncomingTransmissionChannel(), but returns nullptr instead of
 * throwing an error if there is no transmission channel in the reverse
 * path.
 */
```

```
cChannel *findIncomingTransmissionChannel() const;
//@}

/** @name Gate connectivity. */
//@{

/**
 * Returns the previous gate in the series of connections (the path) that
 * contains this gate, or nullptr if this gate is the first one in the path.
 * (E.g. for a simple module output gate, this function will return nullptr.)
 */
cGate *getPreviousGate() const {return prevGate;}

/**
 * Returns the next gate in the series of connections (the path) that
 * contains this gate, or nullptr if this gate is the last one in the path.
 * (E.g. for a simple module input gate, this function will return nullptr.)
 */
cGate *getNextGate() const   {return nextGate;}

/**
 * Returns an ID that uniquely identifies the connection between this gate
 * and the next gate in the path (see getNextGate()) during the lifetime of
 * the simulation. (Disconnecting and then reconnecting the gate results
 * in a new connection ID being assigned.) The method returns -1 if the gate
 * is unconnected.
 */
int getConnectionId() const  {return connectionId;}

/**
 * Return the ultimate source of the series of connections
 * (the path) that contains this gate.
 */
cGate *getPathStartGate() const;

/**
 * Return the ultimate destination of the series of connections
 * (the path) that contains this gate.
 */
cGate *getPathEndGate() const;

/**
 * Determines if a given module is in the path containing this gate.
 */
bool pathContains(cModule *module, int gateId=-1);
```

```
/**
 * Returns true if the gate is connected outside (i.e. to one of its
 * sibling modules or to the parent module).
 *
 * This means that for an input gate, getPreviousGate() must be non-nullptr; for an ou
 * gate, getNextGate() must be non-nullptr.
 */
bool isConnectedOutside() const;

/**
 * Returns true if the gate (of a compound module) is connected inside
 * (i.e. to one of its submodules).
 *
 * This means that for an input gate, getNextGate() must be non-nullptr; for an output
 * gate, getPreviousGate() must be non-nullptr.
 */
bool isConnectedInside() const;

/**
 * Returns true if the gate fully connected. For a compound module gate
 * this means both isConnectedInside() and isConnectedOutside() are true;
 * for a simple module, only isConnectedOutside() is checked.
 */
bool isConnected() const;

/**
 * Returns true if the path (chain of connections) containing this gate
 * starts and ends at a simple module.
 */
bool isPathOK() const;
//@}

/** @name Display string. */
//@{
/**
 * Returns the display string for the gate, which controls the appearance
 * of the connection arrow starting from gate. The display string is stored
 * in the channel associated with the connection. If there is no channel,
 * this call creates an installs a cIdealChannel to hold the display string.
 */
cDisplayString& getDisplayString();

/**
 * Shortcut to <tt>getDisplayString().set(dispstr)</tt>.
```

```
     */
    void setDisplayString(const char *dispstr);
    //@}
};
```

4.2.4 cTopology

## 4.2.5 cExpression

4.2.6 EV 类

一个对调试程序有帮助的类。

```
//==========================================================================
//   CLOG.H  -  header for
//                     OMNeT++/OMNEST
//            Discrete System Simulation in C++
//
//==========================================================================

/*--------------------------------------------------------------*
  Copyright (C) 1992-2017 Andras Varga
  Copyright (C) 2006-2017 OpenSim Ltd.

  This file is distributed WITHOUT ANY WARRANTY. See the file
  `license' for details on this and other legal matters.
*--------------------------------------------------------------*/

#ifndef __OMNETPP_CLOG_H
#define __OMNETPP_CLOG_H

#include <ctime>
#include <sstream>
#include "simkerneldefs.h"

namespace omnetpp {

class cObject;
class cComponent;

/**
 * @brief Classifies log messages based on detail and importance.
 *
 * @ingroup Logging
 */
enum LogLevel
{
    /**
     * The lowest log level; it should be used for low-level implementation-specific
     * technical details that are mostly useful for the developers/maintainers of the
     * component. For example, a MAC layer protocol component could log control flow
     * in loops and if statements, entering/leaving methods and code blocks using this
     * log level.
     */
    LOGLEVEL_TRACE,

    /**
     * This log level should be used for high-level implementation-specific technical
     * details that are most likely important for the developers/maintainers of the
     * component. These messages may help to debug various issues when one is looking
     * at the code. For example, a MAC layer protocol component could log updates to
     * internal state variables, updates to complex data structures using this log level.
     */
    LOGLEVEL_DEBUG,

    /**
     * This log level should be used for low-level protocol-specific details that
     * may be useful and understandable by the users of the component. These messages
     * may help to track down various protocol-specific issues without actually looking
     * too deep into the code. For example, a MAC layer protocol component could log
     * state machine updates, acknowledge timeouts and selected back-off periods using
     * this level.
     */
    LOGLEVEL_DETAIL,

    /**
     * This log level should be used for high-level protocol specific details that
     * are most likely important for the users of the component. For example, a MAC
     * layer protocol component could log successful packet receptions and successful
     * packet transmissions using this level.
     */
    LOGLEVEL_INFO,
    /**
     * This log level should be used for exceptional (non-error) situations that
     * may be important for users and rarely occur in the component. For example,
     * a MAC layer protocol component could log detected bit errors using this level.
     */
    LOGLEVEL_WARN,

    /**
     * This log level should be used for recoverable (non-fatal) errors that allow
     * the component to continue normal operation. For example, a MAC layer protocol
     * component could log unsuccessful packet receptions and unsuccessful packet
     * transmissions using this level.
     */
    LOGLEVEL_ERROR,

    /**
     * The highest log level; it should be used for fatal (unrecoverable) errors
     * that prevent the component from further operation. It doesn't mean that
     * the simulation must stop immediately (because in such cases the code should
     * throw a cRuntimeError), but rather that the a component is unable to continue
     * normal operation. For example, a special purpose recording component may be
     * unable to continue recording due to the disk being full.
     */
    LOGLEVEL_FATAL,

    /**
     * Not a real log level, it completely disables logging when set.
     */
    LOGLEVEL_OFF,
};

/**
 * @brief For compile-time filtering of logs.
 *
 * One is free to define this macro before including <omnetpp.h>, or redefine
 * it any time. The change will affect subsequent log statements.
 * Log statements that use lower log levels than the one specified
 * by this macro will not be compiled into the executable.
 *
 * @ingroup Logging
 */
#ifndef COMPILETIME_LOGLEVEL
#ifdef NDEBUG
#define COMPILETIME_LOGLEVEL omnetpp::LOGLEVEL_DETAIL
#else
```

```
#define COMPILETIME_LOGLEVEL omnetpp::LOGLEVEL_TRACE
#endif
#endif

/**
 * @brief This predicate determines if a log statement gets compiled into the
 * executable.
 *
 * One is free to define this macro before including <omnetpp.h>, or redefine
 * it any time. The change will affect subsequent log statements.
 *
 * @ingroup Logging
 */
#ifndef COMPILETIME_LOG_PREDICATE
#define COMPILETIME_LOG_PREDICATE(object, logLevel, category) (logLevel >= COMPILETIME_LOG
#endif

/**
 * @brief This class groups logging related functionality.
 *
 * @see LogLevel
 * @ingroup Logging
 */
class SIM_API cLog
{
  public:
    typedef bool (*NoncomponentLogPredicate)(const void *object, LogLevel logLevel, const c
    typedef bool (*ComponentLogPredicate)(const cComponent *object, LogLevel logLevel, cons

  public:
    /**
     * This log level specifies a globally applied runtime modifiable filter. This is
     * the fastest runtime filter, it works with a simple integer comparison at the call
     * site.
     */
    static LogLevel logLevel;

    /**
     * This predicate determines if a log statement is executed for log statements
     * that occur outside module or channel member functions. This is a customization
     * point for logging.
     */
    static NoncomponentLogPredicate noncomponentLogPredicate;

    /**
```

```
     * This predicate determines if a log statement is executed for log statements
     * that occur in module or channel member functions. This is a customization
     * point for logging.
     */
    static ComponentLogPredicate componentLogPredicate;

  public:
    /**
     * Returns a human-readable string representing the provided log level.
     */
    static const char *getLogLevelName(LogLevel logLevel);

    /**
     * Returns the associated log level for the provided human-readable string.
     */
    static LogLevel resolveLogLevel(const char *name);

    static inline bool runtimeLogPredicate(const void *object, LogLevel logLevel, const cha
    { return noncomponentLogPredicate(object, logLevel, category); }

    static inline bool runtimeLogPredicate(const cComponent *object, LogLevel logLevel, co
    { return componentLogPredicate(object, logLevel, category); }

    static bool defaultNoncomponentLogPredicate(const void *object, LogLevel logLevel, cons
    static bool defaultComponentLogPredicate(const cComponent *object, LogLevel logLevel, c
};

// Creates a log proxy object that captures the provided context.
// This macro is internal to the logging infrastructure.
//
// NOTE: the (void)0 trick prevents GCC producing statement has no effect warnings
// for compile time disabled log statements.
//
#define OPP_LOGPROXY(object, logLevel, category) \
    ((void)0, !(COMPILETIME_LOG_PREDICATE(object, logLevel, category) && \
    omnetpp::cLog::runtimeLogPredicate(object, logLevel, category))) ? \
    omnetpp::cLogProxy::dummyStream : omnetpp::cLogProxy(object, logLevel, category, __FIL


// Returns nullptr. Helper function for the logging macros.
inline void *getThisPtr() {return nullptr;}

/**
 * @brief Use this macro when logging from static member functions.
 *
 * Background: EV_LOG and derived macros (EV_INFO, EV_DETAIL, etc) will fail
 * to compile when placed into static member functions of cObject-derived classes
 * ("cannot call member function 'cObject::getThisPtr()' without object" in GNU C++,
 * and "C2352: illegal call of non-static member function" in Visual C++).
 * To fix it, add this macro at the top of the function; it contains local declarations
 * to make the code compile.
 *
 * @ingroup Logging
 * @hideinitializer
 */
#define EV_STATICCONTEXT  void *(*getThisPtr)() = omnetpp::getThisPtr;

/**
 * @brief This is the macro underlying EV_INFO, EV_DETAIL, EV_INFO_C, and
 * similar log macros.
 *
 * This macro should not be used directly, but via the logging macros
 * EV, EV_FATAL, EV_ERROR, EV_WARN, EV_INFO, EV_DETAIL, EV_DEBUG, EV_TRACE,
 * and their "category" versions EV_C, EV_FATAL_C, EV_ERROR_C, EV_WARN_C,
 * EV_INFO_C, EV_DETAIL_C, EV_DEBUG_C, EV_TRACE_C.
 *
 * Those macros act as C++ streams: one can write on them using the
 * left-shift (<<) operator. Their names refer to the log level they
 * represent (see LogLevel). The "category" (_C) versions accept a category
 * string. Each category acts like a separate log channel; for example,
 * one can use the "test" category to log text intended for consumption
 * by an automated test suite.
 *
 * Log statements are wrapped with compile-time and runtime guards at the
 * call site to efficiently prevent unnecessary computation of parameters
 * and log content. Compile-time guards are COMPILETIME_LOGLEVEL and
 * COMPILETIME_LOG_PREDICATE. Runtime guards (runtime log level) can be
 * set up via omnetpp.ini.
 *
 * Under certain circumstances, compiling log statements may result in errors.
 * When that happens, it is possible that the EV_STATICCONTEXT macro needs to
 * be added to the code; please review its documentation for more info.
 *
 * Examples:
 *
 * \code
 * EV_INFO << "Connection setup complete" << endl;
 * EV_INFO_C("test") << "ESTAB" << endl;
 * \endcode
 *
 * @see LogLevel, EV_STATICCONTEXT, EV_INFO, EV_INFO_C, COMPILETIME_LOGLEVEL, COMPILETIME_
 * @ingroup Logging
 * @hideinitializer
 */
#define EV_LOG(logLevel, category) OPP_LOGPROXY(getThisPtr(), logLevel, category).getStrea

/**
 * @brief Short for EV_INFO.
 * @see EV_INFO @ingroup Logging
 */
#define EV        EV_INFO

/**
 * @brief Pseudo-stream for logging local fatal errors. See EV_LOG for details.
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_FATAL  EV_LOG(omnetpp::LOGLEVEL_FATAL, nullptr)

/**
 * @brief Pseudo-stream for logging local recoverable errors. See EV_LOG for details.
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_ERROR  EV_LOG(omnetpp::LOGLEVEL_ERROR, nullptr)

/**
 * @brief Pseudo-stream for logging warnings. See EV_LOG for details.
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_WARN   EV_LOG(omnetpp::LOGLEVEL_WARN, nullptr)

/**
 * @brief Pseudo-stream for logging information with the default log level. See EV_LOG for
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_INFO   EV_LOG(omnetpp::LOGLEVEL_INFO, nullptr)

/**
 * @brief Pseudo-stream for logging low-level protocol-specific details. See EV_LOG for de
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_DETAIL EV_LOG(omnetpp::LOGLEVEL_DETAIL, nullptr)

/**
 * @brief Pseudo-stream for logging state variables and other low-level information. See E
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_DEBUG  EV_LOG(omnetpp::LOGLEVEL_DEBUG, nullptr)

/**
 * @brief Pseudo-stream for logging control flow information (entering/exiting functions,
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_TRACE  EV_LOG(omnetpp::LOGLEVEL_TRACE, nullptr)

/**
 * @brief Short for EV_INFO_C.
 * @see EV_INFO_C @ingroup Logging
 */
#define EV_C(category)        EV_INFO_C(category)

/**
 * @brief Pseudo-stream for logging local fatal errors of a specific category. See EV_LOG
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_FATAL_C(category)  EV_LOG(omnetpp::LOGLEVEL_FATAL, category)

/**
 * @brief Pseudo-stream for logging local recoverable errors of a specific category. See E
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_ERROR_C(category)  EV_LOG(omnetpp::LOGLEVEL_ERROR, category)

/**
 * @brief Pseudo-stream for logging warnings of a specific category. See EV_LOG for detail
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_WARN_C(category)   EV_LOG(omnetpp::LOGLEVEL_WARN, category)

/**
 * @brief Pseudo-stream for logging information with the default log level of a specific c
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_INFO_C(category)   EV_LOG(omnetpp::LOGLEVEL_INFO, category)

/**
 * @brief Pseudo-stream for logging low-level protocol-specific details of a specific cate
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_DETAIL_C(category) EV_LOG(omnetpp::LOGLEVEL_DETAIL, category)
```

```
/**
 * @brief Pseudo-stream for logging state variables and other low-level information of a s
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_DEBUG_C(category)  EV_LOG(omnetpp::LOGLEVEL_DEBUG, category)

/**
 * @brief Pseudo-stream for logging control flow information (entering/exiting functions,
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_TRACE_C(category)  EV_LOG(omnetpp::LOGLEVEL_TRACE, category)

/**
 * @brief This class holds various data that is captured when a particular
 * log statement executes. It also contains the text written to the log stream.
 *
 * @see cEnvir::log(cLogEntry*)
 * @ingroup Internals
 */
class SIM_API cLogEntry
{
  public:
    // log statement related
    LogLevel logLevel;
    const char *category;

    // C++ source related (where the log statement appears)
    const void *sourcePointer;
    const cObject *sourceObject;
    const cComponent *sourceComponent;
    const char *sourceFile;
    int sourceLine;
    const char *sourceFunction;

    // operating system related
    clock_t userTime;

    // the actual text of the log statement
    const char *text;
    int textLength;
};


//
// This class captures the context where the log statement appears.
// NOTE: This class is internal to the logging infrastructure.
//
class SIM_API cLogProxy
{
  private:
    // This class is used for buffering the text content to be able to send whole
    // lines one by one to the active environment.
    class LogBuffer : public std::basic_stringbuf<char> {
      public:
        LogBuffer() { }
        bool isEmpty() { return pptr() == pbase(); }
      protected:
        virtual int sync() override;  // invokes getEnvir()->log() for each log line
    };

    // act likes /dev/null
    class nullstream : public std::ostream {
      public:
        nullstream() : std::ostream(nullptr) {}  // results in rdbuf==0 and badbit==true
    };

  public:
    static nullstream dummyStream; // EV evaluates to this when in express mode (getEnvir(

  private:
    static LogBuffer buffer;  // underlying buffer that contains the text that has been wr
    static std::ostream stream;  // this singleton is used to avoid allocating a new strea
    static cLogEntry currentEntry; // context of the current (last) log statement that has
    static LogLevel previousLogLevel; // log level of the previous log statement
    static const char *previousCategory; // category of the previous log statement

  private:
    void fillEntry(LogLevel logLevel, const char *category, const char *sourceFile, int sou

  public:
    cLogProxy(const void *sourcePointer, LogLevel logLevel, const char *category, const cha
    cLogProxy(const cObject *sourceObject, LogLevel logLevel, const char *category, const c
    cLogProxy(const cComponent *sourceComponent, LogLevel logLevel, const char *category, c
    ~cLogProxy();

    std::ostream& getStream() { return stream; }
    static void flushLastLine();
};

}  // namespace omnetpp
```

```
#endif
```

4.3 虚函数

4.3.1 initialize 函数

4.3.1 handleMessage 函数

4.3.1 refreshDisplay 函数

4.3.1 finish 函数

5.1 经验之谈

欢迎读者来到第五章的学习，本章我打算从工程应用的角度，结合现有的仿真经验分享一些技巧，用套路二字来形容也不为过。　　本章涉及的内容包括信道模型应用、节点分布相关、节点之间如何建立通信以及门向量的相关设置，同时也会涉及以上代码相关的说明，简而言之，本章采用情景分析的方法进行说明。也许你会发现本章好多内容可以在 OMNeT++ 社区提供的 Simulation Manual 手册中发现，所以推荐读者后续再阅读 Simulation Manual 手册进行深度研究。

5.2 设计技巧

5.2.1 技巧一：信道模型很重要

据说理想的运放可以摧毁整个地球，那么是不是理想的充电宝是不是充不满电，偏题了，那么理想的信道呢？当初我初次使用 OMNeT++ 时，遇到一个问题：

- 在节点之间传输消息的时候，如何加快消息的传输速度？当节点数量较大的时候，需要较快的实现消息传送的效果。

有此疑问是在运行社区提供的相关工程时发现在他们的的仿真场景中，两个节点似乎可以同时发送消息出去，给人一种并行运行的感觉，让我不得不怀疑是不是需要调用并行接口才能达到这种效果，并且也发现他们的仿真程序运行时间特别小，换句话说就是接近现实的时间级，而我的仿真程序中两个节点传输一个消息都到秒级了，问题很大。　　最后发现这个问题与信道模型有关，也与下一小节的 send 函数相关。在 OMNeT++ 中仿真的时候，如果没有添加信道模型，消息在两个节点之间传输线就是理想的信道模型，这个仿真信道会影响什么呢？

- 仿真结果
- 仿真现象

影响仿真结果好理解，仿真现象呢，那我们来看看仿真模型：

```
channel Channel extends DatarateChannel
{
    delay = default(uniform(20ns, 100ns));
    datarate = default(1000Mbps);
}
```

以上代码是一个简单的信道模型，将这个信道加入到传输线上将会有意想不到的效果。

5.2.2 技巧二：send 函数有套路

不知道读者有时候有没有感觉到 send 函数很麻烦，send 函数用于两个模块之间的消息传输，但是当我们需要发送多条消息的时候，我们不能使用 for 循环直接就上，其主要原因就上我们使用 send 函数发送的消息还没有到达目的节点，此时我们不能使用 send 函数发送下一条消息，那么怎么办呢？这里有两种方案：

- 利用 scheduleAt 函数

```
void Node::handleMessage(cMessage* msg)
{
    if(msg->isSelfMessage()){
        if(msg->getKind()==SMSG_INIT){
                ...
                ...
        cMessage* cloudMsg = new cMessage("hello");
        cloudMsg->setKind(SMSG_INIT);//设置节点类型
        scheduleAt(simTime()+0.01,cloudMsg); //调度一个事件，发送消息给自己
        }
    }
}
```

通过使用 scheduleAt 函数使仿真时间走动，完成上一个消息的完成，这里补充一点，如果读者想使用延时来等待消息传输完成是不可行的，因为使用这种方法仿真时间是不会走动的。例如下面一段代码：

```
time1 = simTime();
func();
time2 = simTime();
```

在上面这段代码中我们的使用 func 函数想使时间走动，但是实验结果告诉我们：$time1 == time2$，经过多次多个地方验证，发现在 OMNeT++ 中如果不调用与仿真时间相关的函数，仿真时间是不会走动的，与上面的实验现象是一致的。因此为了实现仿真时间的走动我们可以采用上面 scheduleAt 函数自我调度一个时间然后再发送下一个消息。

- 一定要采用 send 函数呢？

上述采用 scheduleAt 的方法太麻烦，需要 **new** 一个消息，然后还需要定义一个 SMSG_INIT，另外无端增多 handleMessage 函数内容，这种方法的确不是特别简洁。这里再分享另一种方法：

```
cPacket *pkt = ...; // packet to be transmitted
cChannel *txChannel = gate("out")->getTransmissionChannel();
simtime_t txFinishTime = txChannel->getTransmissionFinishTime();
if (txFinishTime <= simTime())
{
    // channel free; send out packet immediately
    send(pkt, "out");
}
else
{
    // store packet and schedule timer; when the timer expires,
    // the packet should be removed from the queue and sent out
    txQueue.insert(pkt);
    scheduleAt(txFinishTime, endTxMsg);
}
```

上面的代码用于通过 out 门发送一个 pkt 包，但是在传输前需要得到该门上传输的消息的完成时间，需要注意的是当 txFinishTime 为-1 时，说明该门没有消息传输，可以直接发送，如果 txFinishTime 为一个大于 0 的值，说明有消息正在传输，需要等待。所以在判断时我们采用 $txFinishTime <= simTime()$。 通过这种方式，我们可以在 for 循环中发送多个消息。但是对于有些需求不得不使用 scheduleAt 函数完成。

- 提一提 sendDirect 函数 ! "'c sendDirect(cMessage *msg, cModule* mod, int gateId) sendDirect(cMessage *msg, cModule* mod, const char *gateName, int index=-1) sendDirect(cMessage* msg, cGate *gate)

sendDirect(cMessage *msg, simtime_t propagationDelay, simtime_t duration, cModule* mod, int gateId) sendDirect(cMessage *msg, simtime_t propagationDelay, simtime_t duration, cModule* mod, const char *gateName, int index=-1) sendDirect(cMessage* msg, simtime_t propagationDelay, simtime_t duration, cGate *gate)

"' 对于其他 send 类似的函数都是有线的传输方式，需要我们将节点连接才能发送消息，那么如何实现无线的发送方式呢？这个也正是 OMNeT++ 中 wireless 仿真程序中使用的函数，该函数的参数与其他 send 函数不同，它需要指定目的节点，以及目的节点的门。相关详细可以阅读 INET 库代码。 这里有一个问题，当采用前三个函数进行消息传输时，传输的效果为一个圆点，如图 **5-1** 所示：

图 5-1 普通传输效果图

如果在设计网络时，需要将包传输效果设置成图 5-2 所示，对于有线连接和无线连接的两个节点方法不

同，对于有线连接的 send 函数无法在函数的参数上设置。需要在网络拓扑连接时设置好信道，如代码段 5-1 所示：

```
channel Channel extends DatarateChannel
{
    delay = default(uniform(20ns, 100ns));
    datarate = default(2000Mbps);
}
```

对于无线连接的 sendDirect 函数，要想达到相同的效果，就没有设置 channel 一说了，在使用 sendDirect 函数时，有三个重载函数包括有两个参数 simtime_t propagationDelay/simtime_t duration，一个是传播延时时间和持续时间，通过设置这两个参数可以达到图 5-2 的效果。

图 5-2 设置传输延迟和持续时间

5.2.3 技巧三：如何访问同一级的其他模块

在设计网络拓扑时，我们有时需要在一个模块中直接访问同一级其他模块的相关参数，不再经过消息之间传输进行传输。这种接口在 OMNeT++ 下也被提供了，如下一个代码示例：

```
cModule *parent = getParentModule();

// 取出父模块下的 beBuffer 模块
cModule *psubmodBE = parent->getSubmodule("beBuffer");
BEBuffer *pBEBuffer = check_and_cast<BEBuffer *>(psubmodBE);

cModule *psubmodRC = NULL;
RCBuffer *pRCBuffer = NULL;
// 取出父模块下的 rcBuffer 模块
psubmodRC = parent->getSubmodule("rcBuffer");
pRCBuffer = check_and_cast<RCBuffer *>(psubmodRC);
```

上面的代码片段主要通过 getParentModule 和 getSubmodule 两个接口得到指向目的模块的指针，得到指针相当于我们拿到了这个目的模块的所有，需要注意的是这种方式的前提是目的模块是一个简单模块，需要与复合模块区分开，在 OMNeT++ 中复合模块只有对应的**.ned** 文件，其描述方式如下：

```
module Node{
        parameters:
```

```
        ...
    gates:
    ...
}
```

而简单模块有三个文件：**.nde**、**.cc**、**.h**，其.ned 文件中描述方式如下：

```
simple Node{
    parameters:
        ...
    gates:
    ...
}
```

因此对于没有.cc/.h 文件的复合模块，在编写代码时就没有对应的 C++ 类，因此使用上述方法就出现问题，无法事先知道指针类型，那么对于复合模块的访问，我们可以通过下面的代码实现：

```
// 得到当前父模块下的所以模块
for(cModule::SubmoduleIterator iter(getParentModule()); !iter.end(); iter++){
        string ES = string("ES");
        cModule *submodule = *iter;
        string ESnode = string(submodule->getFullName(),0,2);
        // 判断是否是 ES 节点
        if(ESnode == ES){
                // 访问父模块参数
                string realname = string(submodule->getFullName());
                EScpu += submodule->par("cpu").longValue();
                ESmem += submodule->par("mem").longValue();

                cout<<EScpu<<" "<<ESmem<<endl;
        }
        else{
                continue;
        }
}
```

上面的代码段涉及到了 OMNeT++ 下的 SubmoduleIterator 迭代器，该迭代器在较多的库中都有使用，比如：INET，当然这种方式可以经过简单的修改也可以对简单的模块进行访问。在上面的代码段中，getParentModule() 指明是当前模块的父模块，该代码目的就是在当前简单模块中，得到同一父模块下的 ES 复合模块的 cpu/mem 两个参数。

5.2.4 技巧四：遍历所有模块

在有些场景下，我们需要遍历所有节点，甚至是复合节点内部的模块，代码示例如下：

```
/*
 * 在所有节点中寻找一个 ID 等于当前模块的 headId 号的模块
 */
void Node::doNext()
{
    cModule *parent = getParentModule();
    cModule *mod,*Head,*midmod;

    //网络中的所有节点都遍历一次，包括复合模块下的子模块
    for(int i=1;i<=cSimulation::getActiveSimulation()->getLastComponentId();i++){
        int number_of_Bees = cSimulation::getActiveSimulation()->getLastComponentId();
        cSimulation *simobj = cSimulation::getActiveSimulation();
        //这里需要优化
        mod = cSimulation::getActiveSimulation()->getModule(i);
        if(strcmp(mod->getName(),"CenController") == 0){
            //如果遍历到一个模块名为 CenController 的节点
            continue;
        }
        else{
            int j=0;
            while(1){
                string modname = cSimulation::getActiveSimulation()->getModule(i)->getName
                midmod=cSimulation::getActiveSimulation()->getModule(i);
                Head=midmod->getSubmodule(this->clustername.c_str(),j)->getSubmodule("Wire
                if(((Node*)Head)->myId == this->headId){
                    //找到簇头节点, 退出 while 循环
                    break;
                }
                j++;
            }
            // 在到满足条件的 Head 节点，开始执行相关操作
            ...
            ...
            ...
            break;
        }
    }
}
```

在上面的代码段中，可能有些诸如 “Wireless” 相关的过程与我实验源代码本身功能相关，本例只提供一种可参考的代码，具体运用于读者自己的项目中还需要做部分修改。为了让读者更快的掌握这种方法，下面就代码段中的重要接口做一个简单的分析：**- for(int i=1;i<=cSimulation::getActiveSimulation()->getLastComponentId();i++)**

这一句 for 循环遍历当前网络场景中的模块，只遍历仿真场景中的节点，不包括节点内部的模块，下面结合一个网络拓扑文件说明：

```
network simplenet
{
    parameters:
        ...
        ...
    submodules:

        node1[x]: typeA {
            parameters:
                ...
        }

        node2[y]: typeB{
            parameters:
                ...
        }

        node3[z]: typeD {
            parameters:
                ...
        }

    connections allowunconnected:
        ...
                ...
}
```

对于上述网络拓扑，使用上面的 for 循环只能遍历 node1/node2/node3，对于它们内部的子模块不在其内。当最终需要寻找的模块是其中一个的子模块，需要先遍历父模块，然后使用 getSubmodule 函数遍历子模块。

- midmod = cSimulation::getActiveSimulation()->getModule(i)

紧接着上面的 for 循环，得到第 **i** 个模块的地址，如果该模块在网络中描述是用向量的方式需要使用：

$$getSubmodule(node_name, j)$$

即可得到 node_name[j] 所代表的模块。

- getSubmodule(“modname”)

该接口似乎使用频率较高，如何得到一个复合模块的指针，即可通过该接口得到内部子模块的指针，然后访问相关数据。

5.2.5 技巧五：如何得到某一个模块引用的 ned 路径

为什么需要在一个程序中得到该 “.ned” 引用的路径呢？因为在 OMNeT++ 中，我们在设计一个复合模块的内部结构时，可以直接采用图形的方式编辑，相当于我们可以直接拖动设计好的简单模块到复合模块中，而有些简单模块在不同的复合模块中其功能还有所不同，因此在为该简单模块编写.cc 文件时，我们需要检测一下当前本模块在什么模块下使用的，比如是在端系统还是交换机。得到一个模块的引用路径，其实就是一个接口函数的事，如下代码段：

```
cModule *parent = getParentModule();
const char *name = parent->getNedTypeName();

if (strcmp(name, "SimpleNetwork.Node.SimpleNode") == 0){
    cGate *outgate = gate("line$o");
    cChannel *chan = outgate->findTransmissionChannel();
    linkspeed = chan->getNominalDatarate();

}
else if (strcmp(name, "SimpleNetwork.Switch.SwitchPort") == 0){
    //int id = parent->findGate("line$o");
    cGate *outgate = parent->gate("line$o");
    cChannel *chan = outgate->findTransmissionChannel();
    linkspeed = chan->getNominalDatarate();
}
```

该接口函数便是 getNedTypeName，得到完整的路径后，使用 c 库函数 strcmp 进行判断即可。

5.2.6 技巧六：使用 **cTopology** 类遍历拓扑初始化路由表

这是个好东西，其实在 OMNeT++ 中其实提供的大量的接口函数，只是在不知道的前提下写相似的功能函数比较麻烦，这个接口函数完美解决我们寻找路由的门问题，在使用 send 函数传输消息的时候只要知道我们传输的目的节点便可，直接利用一个路由表即可，代码示例如下：

```
/*
 * 探测交换机网络的拓扑
 */
void Router::TopoFind()
{
    cTopology *topo = new cTopology("topo");

    topo->extractByNedTypeName(cStringTokenizer("SimpleNetwork.Node.SimpleNode SimpleNetwo

    EV << "cTopology found " << topo->getNumNodes() << " nodes\n";

    //得到表示本节点的对象
    cTopology::Node *thisNode = topo->getNodeFor(getParentModule());

    // find and store next hops
    for (int i = 0; i < topo->getNumNodes(); i++){
        if (topo->getNode(i) == thisNode)
            continue; // skip ourselves
        //采用迪杰斯特拉算法计算到节点 i 的最短距离
        topo->calculateUnweightedSingleShortestPathsTo(topo->getNode(i));
        //本节点与外界连接的通道
        if (thisNode->getNumPaths() == 0)
            continue; // not connected

        cGate *parentModuleGate = thisNode->getPath(0)->getLocalGate();
        int gateIndex = parentModuleGate->getIndex();
        int address = topo->getNode(i)->getModule()->par("address");
        rtable[address] = gateIndex;
        EV << "  towards address " << address << " gateIndex is " << gateIndex  << endl;
    }
    delete topo;
}
```

该函数有三个比较重要的步骤：- [1] **extractByNedTypeName**

为了得到一个路由表，我们需要指明需要遍历的节点类型。该函数便是指明遍历哪些节点。

- [2] **calculateUnweightedSingleShortestPathsTo**

得到路由表也涉及到路由算法的选择，在 **ctopology.h** 文件中有以下两个路由算法可供选择：

```
/** @name Algorithms to find shortest paths. */
/*
* To be implemented:
*    -  void unweightedMultiShortestPathsTo(Node *target);
*    -  void weightedMultiShortestPathsTo(Node *target);
*/

//@{

/**
* Apply the Dijkstra algorithm to find all shortest paths to the given
* graph node. The paths found can be extracted via Node's methods.
*/
virtual void calculateUnweightedSingleShortestPathsTo(Node *target);

/**
* Apply the Dijkstra algorithm to find all shortest paths to the given
* graph node. The paths found can be extracted via Node's methods.
* Uses weights in nodes and links.
*/
virtual void calculateWeightedSingleShortestPathsTo(Node *target);
```

代入参数就是目的节点地址，其他内容读者可自行探索。

- [3] **topo->getNode(i)->getModule()->par("address");**

这里比较重要的便是 "address" 形参，在以太网中相当于 **IP** 地址，最终得到的 rtable[] 表其索引就是目的地址的 address，索引对应的值就是该节点的门，从该门出去到目的节点路径最短。

5.2.7 技巧七：如何使用 OpenSceneGraph

其实在 OMNeT++ 中是可以直接使用 OpenSceneGraph 的，可怜的我尝试了安装了一下午，才知道 OMNet++ 已经支持 OpenSceneGraph 了，以后补充这一点可以看：

omnetpp-5.2/doc/manual/index.html#sec:graphics:opp-api-for-osg

samples 里已经有支持三维显示的仿真程序了，读者可自行运行看。

5.2.8 技巧八：如何多次利用同一个 msg

在 OMNeT++ 中，凡是使用 scheduleAt 调度的消息属于 Self-Messages，其作用是用在模块本身调度事件使用的。有时需要利用同一个 msg，但是中间必须使用 cancelEvent 函数取消掉上次，如下片段：

```
//cMessage *msg
if (msg->isScheduled())
    cancelEvent(msg);
scheduleAt(simTime() + delay, msg);
```

该代码段没有什么特别大的功能，主要是重复利用已经定义好的 msg 变量。

5.2.9 技巧九：initialize 函数的不同

在每一个简单模块对应的.cc/.h 文件中会有一个 initialize 函数，其功能是在仿真程序开始执行前将会执行的函数，与类的构造函数不同，引出的问题就是：> 如果在其他成员函数中给一个指针数组成员赋值，当离开这个函数后，该指针数组值将会回到原来的值，该函数赋的值没有任何作用，但是如果在 initialize 函数中初始化这个指针数组，将会达到我们想要的结果。

5.2.10 技巧十：如何从仿真场景读取节点坐标

也许作者的用词不明，这里的仿真场景指的是运行仿真后出现的仿真界面。必须提到的是这个 OMNeT++ 的仿真场景，节点在该场景上的位置，不一定是它的属性里边的地址，它们可以不同，感觉似乎是 OMNeT++ 开发者提供的缺口，不知这个是好还是坏，但是好消息就是这些开发者提供了读取场景上节点属性的坐标和在程序中设置该坐标（目的就是让这个显示坐标更新），简而言之，你的节点坐标更新需要你自己在程序中完成，OMNeT++ 不会自动帮你完成。程序 5.2.9-1 是关于读取坐标和更新场景坐标的显示的代码段：

```
程序 5.2.9-1
// 按照最开始的网络拓扑（按圆形分布），得到每一个节点的坐标
this->xpos = atof(parentdispStr.getTagArg("p", 0));
```

```
this->ypos = atof(parentdispStr.getTagArg("p", 1));

coord_X.setDoubleValue(this->xpos); //将仿真界面上的 xpos 改变
coord_Y.setDoubleValue(this->ypos); //将仿真界面上的 ypos 改变
```

需要再次提示的是这个坐标读取的是显示的节点的坐标，与节点在仿真场景上显示的位置可能没有关系。

5.2.11 技巧十一：如何调用 INET 中的类

有时候在仿真程序中有种需求：> 需要在一个仿真程序中调用其他库中的函数，例如需要使用 INET 中相关类，那这时候的逻辑是什么？

与在某一个工程下需要 import INET 中的 NED 模型，我们需要在工程的属性中 Project References 中勾上我们需要 import 的库，然后在工程的 ned 文件中添加 ned 模型路径。同时当我们设置了工程 Project References，当编译该工程时，将会链接 Project References 中勾上的工程编译生成的库文件，其中涉及以下编译设置：

```
// macros needed for building Windows DLLs
#if defined(_WIN32)
#  define OPP_DLLEXPORT  __declspec(dllexport)
#  define OPP_DLLIMPORT  __declspec(dllimport)
#else
#  define OPP_DLLIMPORT
#  define OPP_DLLEXPORT
#endif
```

以上摘取自 INET 开源库中 platdefs.h 文件，其中比较重要的是当编译 INET 库时，编译默认选项会使用 __declspec(dllexport)，当另一个仿真工程（使用了 INET 库中的类）编译时，将会以 __declspec(dllimport)，因此工程不需要设置其他编译选项，但是需要将诸如 INET 编译生成的.dll 或者.a 拷贝一份到该工程目录下。

- 注意：

如果以上关系都满足了，再出现在链接工程的错误可能其其他导致的。

7 错误记录

在我刚开始使用 OMNeT++ 进行实验的时候，总是会出现一些稀奇古怪的错误，每一个错误都花了我大量的时间，所以在最后一章中总结出来，也许你正好用上。

7.1 在模块加入移动模块之后，仿真出现 nan 错误

问题描述：

> 对于某些工程加入移动模块以后，编译过程没有出错，但是当执行仿真程序的时候，出现一些 nan 错误提示。需要在仿真配置文件加入：

```
**.constraintAreaMinX = 0m
**.constraintAreaMinY = 0m
**.constraintAreaMinZ = 0m
**.constraintAreaMaxX = 5000m
**.constraintAreaMaxY = 5000m
**.constraintAreaMaxZ = 0m
```

移动模块配置：

```
**.UAV[*].mobilityType = "MassMobility"
**.UAV[*].mobility.initFromDisplayString = true
**.UAV[*].mobility.changeInterval = truncnormal(2s, 0.5s)
**.UAV[*].mobility.changeAngleBy = normal(0deg, 30deg)
**.UAV[*].mobility.speed = truncnormal(250mps, 20mps)
**.UAV[*].mobility.updateInterval = 100ms
```

7.3 工程的例如 cModule 之类的类不能高亮显示？

问题描述：

> 原工程是可以高亮显示的，但是由于我在备份这个程序的时候可能方式不对，我采用的是在文件资源管理器的窗口复制原工程文件夹，没有在软件的窗口进行 rename，可能是这个原因造成的。

解决办法：

在软件的窗口，对工程进行 rename 就行，编译一次，cMdoule 等等关键词就可以高亮了。

8 移动自组网中的路由协议

移动自组网是由一群可以移动的、兼具终端及路由功能的移动节点，通过无线链路形成的无中心、多跳、临时性自治系统。[1] 本章主要介绍移动自组网中的常见路由协议。

DSDV

DSR

ADOV

参考文献

[1] 潘宇航. 基于 QoS 感知的移动自组织网络路由协议设计与仿真研究. MS thesis. 西南交通大学, 2014.

5.1 经验之谈

欢迎读者来到第五章的学习，本章我打算从工程应用的角度，结合现有的仿真经验分享一些技巧，用套路二字来形容也不为过。　　本章涉及的内容包括信道模型应用、节点分布相关、节点之间如何建立通信以及门向量的相关设置，同时也会涉及以上代码相关的说明，简而言之，本章采用情景分析的方法进行说明。也许你会发现本章好多内容可以在 OMNeT++ 社区提供的 Simulation Manual 手册中发现，所以推荐读者后续再阅读 Simulation Manual 手册进行深度研究。

5.2 设计技巧

5.2.1 技巧一：信道模型很重要

据说理想的运放可以摧毁整个地球，那么是不是理想的充电宝是不是充不满电，偏题了，那么理想的信道呢？当初我初次使用 OMNeT++ 时，遇到一个问题：

- 在节点之间传输消息的时候，如何加快消息的传输速度？当节点数量较大的时候，需要较快的实现消息传送的效果。

有此疑问是在运行社区提供的相关工程时发现在他们的的仿真场景中，两个节点似乎可以同时发送消息出去，给人一种并行运行的感觉，让我不得不怀疑是不是需要调用并行接口才能达到这种效果，并且也发现他们的仿真程序运行时间特别小，换句话说就是接近现实的时间级，而我的仿真程序中两个节点传输一个消息都到秒级了，问题很大。　　最后发现这个问题与信道模型有关，也与下一小节的 send 函数相关。在 OMNeT++ 中仿真的时候，如果没有添加信道模型，消息在两个节点之间传输线就是理想的信道模型，这个仿真信道会影响什么呢？

- 仿真结果
- 仿真现象

影响仿真结果好理解，仿真现象呢，那我们来看看仿真模型：

```
channel Channel extends DatarateChannel
{
    delay = default(uniform(20ns, 100ns));
    datarate = default(1000Mbps);
}
```

以上代码是一个简单的信道模型，将这个信道加入到传输线上将会有意想不到的效果。

5.2.2 技巧二：**send** 函数有套路

不知道读者有时候有没有感觉到 send 函数很麻烦，send 函数用于两个模块之间的消息传输，但是当我们需要发送多条消息的时候，我们不能使用 for 循环直接就上，其主要原因就上我们使用 send 函数发送的消息还没有到达目的节点，此时我们不能使用 send 函数发送下一条消息，那么怎么办呢？这里有两种方案：

- 利用 scheduleAt 函数

```
void Node::handleMessage(cMessage* msg)
{
    if(msg->isSelfMessage()){
        if(msg->getKind()==SMSG_INIT){
                ...
                ...
        cMessage* cloudMsg = new cMessage("hello");
        cloudMsg->setKind(SMSG_INIT);//设置节点类型
        scheduleAt(simTime()+0.01,cloudMsg); //调度一个事件，发送消息给自己
        }
    }
}
```

通过使用 scheduleAt 函数使仿真时间走动，完成上一个消息的完成，这里补充一点，如果读者想使用延时来等待消息传输完成是不可行的，因为使用这种方法仿真时间是不会走动的。例如下面一段代码：

```
time1 = simTime();
func();
time2 = simTime();
```

在上面这段代码中我们的使用 func 函数想使时间走动，但是实验结果告诉我们：$time1 == time2$，经过多次多个地方验证，发现在 OMNeT++ 中如果不调用与仿真时间相关的函数，仿真时间是不会走动的，与上面的实验现象是一致的。因此为了实现仿真时间的走动我们可以采用上面 scheduleAt 函数自我调度一个时间然后再发送下一个消息。

- 一定要采用 send 函数呢？

上述采用 scheduleAt 的方法太麻烦，需要 **new** 一个消息，然后还需要定义一个 SMSG_INIT，另外无端增多 handleMessage 函数内容，这种方法的确不是特别简洁。这里再分享另一种方法：

```
cPacket *pkt = ...; // packet to be transmitted
cChannel *txChannel = gate("out")->getTransmissionChannel();
simtime_t txFinishTime = txChannel->getTransmissionFinishTime();
if (txFinishTime <= simTime())
{
    // channel free; send out packet immediately
    send(pkt, "out");
}
else
{
    // store packet and schedule timer; when the timer expires,
    // the packet should be removed from the queue and sent out
    txQueue.insert(pkt);
    scheduleAt(txFinishTime, endTxMsg);
}
```

上面的代码用于通过 out 门发送一个 pkt 包，但是在传输前需要得到该门上传输的消息的完成时间，需要注意的是当 txFinishTime 为-1 时，说明该门没有消息传输，可以直接发送，如果 txFinishTime 为一个大于 0 的值，说明有消息正在传输，需要等待。所以在判断时我们采用 $txFinishTime <= simTime()$。　通过这种方式，我们可以在 for 循环中发送多个消息。但是对于有些需求不得不使用 scheduleAt 函数完成。

- 提一提 sendDirect 函数！"'c sendDirect(cMessage *msg, cModule* mod, int gateId) sendDirect(cMessage *msg, cModule* mod, const char *gateName, int index=-1) sendDirect(cMessage* msg, cGate *gate)

sendDirect(cMessage *msg, simtime_t propagationDelay, simtime_t duration, cModule* mod, int gateId) sendDirect(cMessage *msg, simtime_t propagationDelay, simtime_t duration, cModule* mod, const char *gateName, int index=-1) sendDirect(cMessage* msg, simtime_t propagationDelay, simtime_t duration, cGate *gate)

"'　对于其他 send 类似的函数都是有线的传输方式，需要我们将节点连接才能发送消息，那么如何实现无线的发送方式呢？这个也正是 OMNeT++ 中 wireless 仿真程序中使用的函数，该函数的参数与其他 send 函数不同，它需要指定目的节点，以及目的节点的门。相关详细可以阅读 INET 库代码。　这里有一个问题，当采用前三个函数进行消息传输时，传输的效果为一个圆点，如图 **5-1** 所示：

图 5-1 普通传输效果图

如果在设计网络时，需要将包传输效果设置成图 5-2 所示，对于有线连接和无线连接的两个节点方法不

同，对于有线连接的 send 函数无法在函数的参数上设置。需要在网络拓扑连接时设置好信道，如代码段 5-1 所示：

```
channel Channel extends DatarateChannel
{
    delay = default(uniform(20ns, 100ns));
    datarate = default(2000Mbps);
}
```

对于无线连接的 sendDirect 函数，要想达到相同的效果，就没有设置 channel 一说了，在使用 sendDirect 函数时，有三个重载函数包括有两个参数 simtime_t propagationDelay/simtime_t duration，一个是传播延时时间和持续时间，通过设置这两个参数可以达到图 5-2 的效果。

图 5-2 设置传输延迟和持续时间

5.2.3 技巧三：如何访问同一级的其他模块

在设计网络拓扑时，我们有时需要在一个模块中直接访问同一级其他模块的相关参数，不再经过消息之间传输进行传输。这种接口在 OMNeT++ 下也被提供了，如下一个代码示例：

```
cModule *parent = getParentModule();

// 取出父模块下的 beBuffer 模块
cModule *psubmodBE = parent->getSubmodule("beBuffer");
BEBuffer *pBEBuffer = check_and_cast<BEBuffer *>(psubmodBE);

cModule *psubmodRC = NULL;
RCBuffer *pRCBuffer = NULL;
// 取出父模块下的 rcBuffer 模块
psubmodRC = parent->getSubmodule("rcBuffer");
pRCBuffer = check_and_cast<RCBuffer *>(psubmodRC);
```

上面的代码片段主要通过 getParentModule 和 getSubmodule 两个接口得到指向目的模块的指针，得到指针相当于我们拿到了这个目的模块的所有，需要注意的是这种方式的前提是目的模块是一个简单模块，需要与复合模块区分开，在 OMNeT++ 中复合模块只有对应的**.ned** 文件，其描述方式如下：

```
module Node{
        parameters:
```

```
        ...
    gates:
    ...
}
```

而简单模块有三个文件：**.nde**、**.cc**、**.h**，其.ned 文件中描述方式如下：

```
simple Node{
    parameters:
        ...
    gates:
    ...
}
```

因此对于没有.cc/.h 文件的复合模块，在编写代码时就没有对应的 C++ 类，因此使用上述方法就出现问题，无法事先知道指针类型，那么对于复合模块的访问，我们可以通过下面的代码实现：

```
// 得到当前父模块下的所以模块
for(cModule::SubmoduleIterator iter(getParentModule()); !iter.end(); iter++){
        string ES = string("ES");
        cModule *submodule = *iter;
        string ESnode = string(submodule->getFullName(),0,2);
        // 判断是否是 ES 节点
        if(ESnode == ES){
                // 访问父模块参数
                string realname = string(submodule->getFullName());
                EScpu += submodule->par("cpu").longValue();
                ESmem += submodule->par("mem").longValue();

                cout<<EScpu<<" "<<ESmem<<endl;
        }
        else{
                continue;
        }
}
```

上面的代码段涉及到了 OMNeT++ 下的 SubmoduleIterator 迭代器，该迭代器在较多的库中都有使用，比如：INET，当然这种方式可以经过简单的修改也可以对简单的模块进行访问。在上面的代码段中，getParentModule() 指明是当前模块的父模块，该代码目的就是在当前简单模块中，得到同一父模块下的 ES 复合模块的 cpu/mem 两个参数。

5.2.4 技巧四：遍历所有模块

在有些场景下，我们需要遍历所有节点，甚至是复合节点内部的模块，代码示例如下：

```
/*
 * 在所有节点中寻找一个 ID 等于当前模块的 headId 号的模块
 */
void Node::doNext()
{
    cModule *parent = getParentModule();
    cModule *mod,*Head,*midmod;

    //网络中的所有节点都遍历一次，包括复合模块下的子模块
    for(int i=1;i<=cSimulation::getActiveSimulation()->getLastComponentId();i++){
        int number_of_Bees = cSimulation::getActiveSimulation()->getLastComponentId();
        cSimulation *simobj = cSimulation::getActiveSimulation();
        //这里需要优化
        mod = cSimulation::getActiveSimulation()->getModule(i);
        if(strcmp(mod->getName(),"CenController") == 0){
            //如果遍历到一个模块名为 CenController 的节点
            continue;
        }
        else{
            int j=0;
            while(1){
                string modname = cSimulation::getActiveSimulation()->getModule(i)->getName
                midmod=cSimulation::getActiveSimulation()->getModule(i);
                Head=midmod->getSubmodule(this->clustername.c_str(),j)->getSubmodule("Wire
                if(((Node*)Head)->myId == this->headId){
                    //找到簇头节点, 退出 while 循环
                    break;
                }
                j++;
            }
            // 在到满足条件的 Head 节点，开始执行相关操作
            ...
            ...
            ...
            break;
        }
    }
}
```

在上面的代码段中，可能有些诸如 “Wireless” 相关的过程与我实验源代码本身功能相关，本例只提供一种可参考的代码，具体运用于读者自己的项目中还需要做部分修改。为了让读者更快的掌握这种方法，下面就代码段中的重要接口做一个简单的分析：**- for(int i=1;i<=cSimulation::getActiveSimulation()->getLastComponentId();i++)**

这一句 for 循环遍历当前网络场景中的模块，只遍历仿真场景中的节点，不包括节点内部的模块，下面结合一个网络拓扑文件说明：

```
network simplenet
{
    parameters:
        ...
        ...
    submodules:

        node1[x]: typeA {
            parameters:
                ...
        }

        node2[y]: typeB{
            parameters:
                ...
        }

        node3[z]: typeD {
            parameters:
                ...
        }

    connections allowunconnected:
        ...
                ...
}
```

对于上述网络拓扑，使用上面的 for 循环只能遍历 node1/node2/node3，对于它们内部的子模块不在其内。当最终需要寻找的模块是其中一个的子模块，需要先遍历父模块，然后使用 getSubmodule 函数遍历子模块。

- midmod = cSimulation::getActiveSimulation()->getModule(i)

紧接着上面的 for 循环，得到第 **i** 个模块的地址，如果该模块在网络中描述是用向量的方式需要使用：

$$getSubmodule(node_name, j)$$

即可得到 node_name[j] 所代表的模块。

- getSubmodule(“modname”)

该接口似乎使用频率较高，如何得到一个复合模块的指针，即可通过该接口得到内部子模块的指针，然后访问相关数据。

5.2.5 技巧五：如何得到某一个模块引用的 ned 路径

为什么需要在一个程序中得到该 “.ned” 引用的路径呢？因为在 OMNeT++ 中，我们在设计一个复合模块的内部结构时，可以直接采用图形的方式编辑，相当于我们可以直接拖动设计好的简单模块到复合模块中，而有些简单模块在不同的复合模块中其功能还有所不同，因此在为该简单模块编写.cc 文件时，我们需要检测一下当前本模块在什么模块下使用的，比如是在端系统还是交换机。得到一个模块的引用路径，其实就是一个接口函数的事，如下代码段：

```
cModule *parent = getParentModule();
const char *name = parent->getNedTypeName();

if (strcmp(name, "SimpleNetwork.Node.SimpleNode") == 0){
    cGate *outgate = gate("line$o");
    cChannel *chan = outgate->findTransmissionChannel();
    linkspeed = chan->getNominalDatarate();

}
else if (strcmp(name, "SimpleNetwork.Switch.SwitchPort") == 0){
    //int id = parent->findGate("line$o");
    cGate *outgate = parent->gate("line$o");
    cChannel *chan = outgate->findTransmissionChannel();
    linkspeed = chan->getNominalDatarate();
}
```

该接口函数便是 getNedTypeName，得到完整的路径后，使用 c 库函数 strcmp 进行判断即可。

5.2.6 技巧六：使用 cTopology 类遍历拓扑初始化路由表

这是个好东西，其实在 OMNeT++ 中其实提供的大量的接口函数，只是在不知道的前提下写相似的功能函数比较麻烦，这个接口函数完美解决我们寻找路由的门问题，在使用 send 函数传输消息的时候只要知道我们传输的目的节点便可，直接利用一个路由表即可，代码示例如下：

```
/*
 * 探测交换机网络的拓扑
 */
void Router::TopoFind()
{
    cTopology *topo = new cTopology("topo");

    topo->extractByNedTypeName(cStringTokenizer("SimpleNetwork.Node.SimpleNode SimpleNetwo

    EV << "cTopology found " << topo->getNumNodes() << " nodes\n";

    //得到表示本节点的对象
    cTopology::Node *thisNode = topo->getNodeFor(getParentModule());

    // find and store next hops
    for (int i = 0; i < topo->getNumNodes(); i++){
        if (topo->getNode(i) == thisNode)
            continue; // skip ourselves
        //采用迪杰斯特拉算法计算到节点 i 的最短距离
        topo->calculateUnweightedSingleShortestPathsTo(topo->getNode(i));
        //本节点与外界连接的通道
        if (thisNode->getNumPaths() == 0)
            continue; // not connected

        cGate *parentModuleGate = thisNode->getPath(0)->getLocalGate();
        int gateIndex = parentModuleGate->getIndex();
        int address = topo->getNode(i)->getModule()->par("address");
        rtable[address] = gateIndex;
        EV << "  towards address " << address << " gateIndex is " << gateIndex  << endl;
    }
    delete topo;
}
```

该函数有三个比较重要的步骤：- [1] **extractByNedTypeName**

为了得到一个路由表，我们需要指明需要遍历的节点类型。该函数便是指明遍历哪些节点。

- [2] **calculateUnweightedSingleShortestPathsTo**

得到路由表也涉及到路由算法的选择，在 **ctopology.h** 文件中有以下两个路由算法可供选择：

```
/** @name Algorithms to find shortest paths. */
/*
* To be implemented:
*    -  void unweightedMultiShortestPathsTo(Node *target);
*    -  void weightedMultiShortestPathsTo(Node *target);
*/

//@{

/**
* Apply the Dijkstra algorithm to find all shortest paths to the given
* graph node. The paths found can be extracted via Node's methods.
*/
virtual void calculateUnweightedSingleShortestPathsTo(Node *target);

/**
* Apply the Dijkstra algorithm to find all shortest paths to the given
* graph node. The paths found can be extracted via Node's methods.
* Uses weights in nodes and links.
*/
virtual void calculateWeightedSingleShortestPathsTo(Node *target);
```

代入参数就是目的节点地址，其他内容读者可自行探索。

- [3] **topo->getNode(i)->getModule()->par("address");**

这里比较重要的便是 "address" 形参，在以太网中相当于 **IP** 地址，最终得到的 rtable[] 表其索引就是目的地址的 address，索引对应的值就是该节点的门，从该门出去到目的节点路径最短。

5.2.7 技巧七：如何使用 **OpenSceneGraph**

其实在 OMNeT++ 中是可以直接使用 OpenSceneGraph 的，可怜的我尝试了安装了一下午，才知道 OMNet++ 已经支持 OpenSceneGraph 了，以后补充这一点可以看：

omnetpp-5.2/doc/manual/index.html#sec:graphics:opp-api-for-osg

samples 里已经有支持三维显示的仿真程序了，读者可自行运行看。

5.2.8 技巧八：如何多次利用同一个 msg

在 OMNeT++ 中，凡是使用 scheduleAt 调度的消息属于 Self-Messages，其作用是用在模块本身调度事件使用的。有时需要利用同一个 msg，但是中间必须使用 cancelEvent 函数取消掉上次，如下片段：

```
//cMessage *msg
if (msg->isScheduled())
    cancelEvent(msg);
scheduleAt(simTime() + delay, msg);
```

该代码段没有什么特别大的功能，主要是重复利用已经定义好的 msg 变量。

5.2.9 技巧九：initialize 函数的不同

在每一个简单模块对应的.cc/.h 文件中会有一个 initialize 函数，其功能是在仿真程序开始执行前将会执行的函数，与类的构造函数不同，引出的问题就是：> 如果在其他成员函数中给一个指针数组成员赋值，当离开这个函数后，该指针数组值将会回到原来的值，该函数赋的值没有任何作用，但是如果在 initialize 函数中初始化这个指针数组，将会达到我们想要的结果。

5.2.10 技巧十：如何从仿真场景读取节点坐标

也许作者的用词不明，这里的仿真场景指的是运行仿真后出现的仿真界面。必须提到的是这个 OMNeT++ 的仿真场景，节点在该场景上的位置，不一定是它的属性里边的地址，它们可以不同，感觉似乎是 OMNeT++ 开发者提供的缺口，不知这个是好还是坏，但是好消息就是这些开发者提供了读取场景上节点属性的坐标和在程序中设置该坐标（目的就是让这个显示坐标更新），简而言之，你的节点坐标更新需要你自己在程序中完成，OMNeT++ 不会自动帮你完成。程序 5.2.9-1 是关于读取坐标和更新场景坐标的显示的代码段：

```
程序 5.2.9-1
// 按照最开始的网络拓扑（按圆形分布），得到每一个节点的坐标
this->xpos = atof(parentdispStr.getTagArg("p", 0));
```

```
this->ypos = atof(parentdispStr.getTagArg("p", 1));

coord_X.setDoubleValue(this->xpos); //将仿真界面上的 xpos 改变
coord_Y.setDoubleValue(this->ypos); //将仿真界面上的 ypos 改变
```

需要再次提示的是这个坐标读取的是显示的节点的坐标，与节点在仿真场景上显示的位置可能没有关系。

5.2.11 技巧十一：如何调用 INET 中的类

有时候在仿真程序中有种需求：> 需要在一个仿真程序中调用其他库中的函数，例如需要使用 INET 中相关类，那这时候的逻辑是什么？

与在某一个工程下需要 import INET 中的 NED 模型，我们需要在工程的属性中 Project References 中勾上我们需要 import 的库，然后在工程的 ned 文件中添加 ned 模型路径。同时当我们设置了工程 Project References，当编译该工程时，将会链接 Project References 中勾上的工程编译生成的库文件，其中涉及以下编译设置：

```
// macros needed for building Windows DLLs
#if defined(_WIN32)
#  define OPP_DLLEXPORT  __declspec(dllexport)
#  define OPP_DLLIMPORT  __declspec(dllimport)
#else
#  define OPP_DLLIMPORT
#  define OPP_DLLEXPORT
#endif
```

以上摘取自 INET 开源库中 platdefs.h 文件，其中比较重要的是当编译 INET 库时，编译默认选项会使用 __declspec(dllexport)，当另一个仿真工程（使用了 INET 库中的类）编译时，将会以 __declspec(dllimport)，因此工程不需要设置其他编译选项，但是需要将诸如 INET 编译生成的.dll 或者.a 拷贝一份到该工程目录下。

- 注意：

如果以上关系都满足了，再出现在链接工程的错误可能其其他导致的。

1.1 OMNeT++ 简介

OMNeT++，一个基于 eclipse 开发套件的开源网络仿真工具，目前主要在高校实验室进行一些网络仿真测试，对一些算法进行对比，它可以供使用者进行完成以下开发：- **C/C++** 开发 - 网络仿真程序设计

毫无疑问，基于 eclipse 的开发工具肯定能支持普通的 C/C++ 工程。另外，在 OMNeT++ 上网络仿真设计领域的优势在于，它是一个开源的项目，对大量的网络模型都提供代码支持。但是问题在于国内的确没有什么社区支持，出现问题只能自己解决，其实对于开源的项目大多存在这种问题，往往开源的项目，使用起来难度较大，开源项目往往比那些商业的软件开发难度较大，支持也较少，开源可不代表简单。 OMNeT++ 对初学者能力要求高，它假定使用者对编程有一定了解的，对 eclipse 开发环境也是特别熟悉的，另外这是一个网络仿真的软件，需要你对计算机网络有足够的认识，它提供了大量现有各种网络的仿真例子，如果你对网络认识足够强，那么这个软件你用起来会感到特别顺手。 目前有大量的开源仿真库用于 OMNeT++ 环境，拥有丰富的外文资料，官方将其分为两类，包括 Supported Models 和 Contributed Models：- Supported Models 模型库的开发处于激活状态，有开发者在维护，定期会推出新的版本。- Contributed Models 完成后只推出过一次或几次版本，目前没有人在维护。

1.2 OMNeT++ 开源库

下面简单介绍一下几种常见的开源库。

- INET

由 Simucraft 公司主持开发，用于仿真有线及无线网络。

应用层协议：

- HTTP、FTP、Telnet、不同优先级的 Video、Ping

传输层协议：

- TCP、UDP、RTP (RealtimeTransport Protocol)

网络层协议：

- IPv4、IPv6、ICMP、ARP、MPLS、LDP、RSVP、OSPF、Mobile IPV6、AODV、DSDV、DSR

数据链路层协议：

- Ethernet、PPP、IEEE 802.11、FDDI、Token Ring
- 官网：http://inet.omnetpp.org
- INETMANET

由 Simucraft 公司主持开发，用于仿真无线、有线网络，在 INET 的基础上增加了大量的 MANET 协议，INETMANET= INET+MANET，在 INET 的基础上增加：

- 802.11a,g:Ieee80211aMac, Ieee80211gMac, Ieee80211aRadioModel, Ieee80211gRadioModel
- Ieee80211Mesh,Ieee80211MeshMgmt
- radiomodels: TwoRayModel, ShadowingModel, qamMode
- Ns2MotionMobility
- ARP:global ARP cache
- AODV,DSDV, DSR, DYMO, OLSR
- 官网：http://inet.omnetpp.org
- Mobility Framework
- 由 Simucraft 公司主持开发
- 是一个无线传感器仿真模型库
- 绝大多数协议已经被 INET 吸收
- 官网：http://mobility-fw.sourceforge.net/hp/index.html
- SensorSimulator
- 美国路易斯安娜州立大学开发
- 用于仿真无线传感器网络
- 官网：http://csc.lsu.edu/sensor_web/
- Castalia
- 澳大利亚国家信息技术中心（NICTA）开发
- 是一个基于 OMNeT++ 的侧重于无线网络的仿真器

- 基于实测数据的高级 channel/radio 模型
- Radio 详细的状态转移，允许多传输功率电平
- 高度灵活的 physical process model
- 感应设备的噪声、偏差（bias）和功耗
- 节点时钟漂移，CPU 功耗
- 资源监控，如超出功率限制（如 CPU 或内存）
- 拥有大量可调参数的 mac 协议
- 用于设计优化和扩展
- 官网：https://github.com/boulis/Castalia
- OverSim
- 德国德国卡尔斯鲁厄大学开发
- 用于仿真点对点（p-to-p）协议，如 chard，GIA 等
- 官网：http://www.oversim.org

1.3 目录

本手册与现有的那两本书风格不同，我希望读者通过此手册可以快速的上手 OMNeT++，快速的掌握 OMNeT++ 提供的各种接口，目前包括以下内容：- [1] OMNeT++ 的安装 - [2] INET 库的安装 && INET 库的基本使用 - [3] OMNeT++ 个性化设置 - [4] OMNeT++ 工程设计技巧 - [5] cModule | cPar | cGate | cTopology 相关类使用 - [6] 仿真结果分析 - [7] 仿真错误记录

2.1 OMNeT++ 下载

OMNeT++ 可以直接从网上下载，网站地址是：https://www.omnetpp.org, 但是国内直接从该网站下载，下载较慢，同时时常在安装下载过程中出现下载中断的情况，导致前功尽弃，下载成功较难。

2.2 OMNeT++ 安装

2.2.1 安装准备

由于 OMNeT++ 支持多个操作系统环境的安装，包括 **MacOS**、**linux** 和 **Windows**，在这里只描述 **Windows** 环境下的安装。软件的安装说明肯定在软件的安装文件有说明，我们没有必要每次安装一个软件的时候都去百度一下软件安装的过程，作者的观点是对于一些破解较难，安装复杂的软件安装可以写写 blog，记录记录。我们可以在 OMNeT++ 的安装包下发现 readme 文件和 doc 目录下的 installguide，去看看吧，总会发现我们的安装执行步骤，掌握这种办法，断网了也能安装、无论过多久还能记得安装过程。好了，废话不多说了。下面是几个你在安装过程中可能会用到的命令：

- [1] **./configure**

在 PC 机上第一次安装的时候，需要根据配置文件配置一下具体我们需要的软件的功能：静态编译程序、依赖库路径、其他什么文件路径

- [2] **make**

在 PC 机上第一次安装的时候，需要根据配置文件配置一下具体我们需要的软件的功能：静态编译程序、依赖库路径、其他什么文件路径

- [3] **make clean**

清除前面安装过程中产生的中间二进制文件，这个命令主要用于重新安装软件的过程中，如果遇到 make 出错的问题，可以选择这个命令清除到二进制文件，然后在使用 make 命令编译安装（因为有些时候下载的安装包不是原始文件）。

2.2.2 图文并茂

其实这一部分没有说明必要，姑且就当作者无聊，还是想写写，我的原则就是坚持把故事讲得透彻明白，有些时候，在阅读别人博客的时候，老是会有很多疑问，其实博主以为读者懂，但读者的专业背景不一样，导致可能很简单的问题，还得下边留个言......好了，我们还是回到本节的话题上。　　以下三张图：

图 2-4-1 doc 目录

上图文件是 OMNeT++ 团队提供给开发者的基本帮助文档，我在写这个文档的时候，自我觉得还没有把这些文档都翻开看一遍，查阅这些文档久了，就会慢慢觉得这些资料本身已经够用了......我会在后续的文档中，描述一下 OMNeT++ 提供给我们的地图。

2.3 INET 库

2.3.1 INET 库的介绍

从一个初学者的角度，当安装 OMNeT++ 后，大多数的情况下是需要安装 INET 库的，这个集成库包含了丰富的仿真模型，多数时候，读者如果设计一个网络仿真程序，有不想重新编写代码，这时候，可以在 INET 下寻找是否有满足要求的 example，包括的网络有：

- adhoc
- aodv
- ethernet
- ipv6

等等，上面列举出的只是其中经常用到的一小部分，但是这也存在读者的不同研究背景，可能其中涉及的还不算很全。目前，我对于 INET 的使用较浅薄，水平还停留在调用 INET 库中的 ned 文件中的节点类型，或者其他诸如移动模型的水平上。在该小节，作者先为读者描述一下如何在 OMNeT++ 下快速的使用 INET 库和目前我经常使用的技巧。

2.3.2 INET 库的安装

通常有两种方法安装 INET，在安装之前，首先需要到: https://inet.omnetpp.org 下载合适的版本，由于前面的 OMNeT++ 使用的 5.2 的版本，这里我们可以选择 **inet-3.6.2**，下载结束以后，将 inet 解压到

omnetpp 的安装路径下的 samples 文件下，此时 inet 文件的路径可能是：**xxx/omnetpp-5.2/samples/inet** (解压 INET-3.6.2 文件后只有一个 inet 文件)。接下来，我们需要：- 方法一：命令窗口安装 INET

其实，如果需要为 omnetpp 安装新的插件或者库，都可以通过命令行的形式进行安装，甚至，你可以在命令行的环境下对编写好的网络进行编译和运行。　　我编写这个学习手册的原则，就是为读者提供一个学习 OMNeT++ 的地图，而不是特别详细的字典，可能我的水平还远远没有达到写一本学习 OMNeT++ 的大全。首先，安装这个 INET 库，我们到 inet 文件下看看有什么有用的文件没有，当然是先看看 **README.md** 了，这个文件提示我们安装请看：**INSTALL**，下面是这个 INSTALL 的英文：

```
If you are building from command line:
--------------------------------------
3. Change to the INET directory.

4. Type "make makefiles". This should generate the makefiles for you automatically.

5. Type "make" to build the inet executable (debug version). Use "make MODE=release"
   to build release version.

6. You can run specific examples by changing into the example's directory and executing "./
run"
```

当然，你可以选择 **mingwenv.cmd** 命令窗口，输入以上指令进行编译安装。上面的英文安装较为简洁，下面是我使用命令窗口安装的过程：

- [1] 在安装 INET 库之前，应先确保 OMNeT++ 已经安装成功。进入到 OMNeT++ 安装路径，找到 **mingwenv.cmd** 文件，双击执行，进入下图：

- [2] 接下来，使用命令：**cd samples/inet**，进入到 samples 下的 inet，另可使用 ls 命令查看当前 inet 文件下各子文件。
- [3] 然后，执行 **make makefiles** 命令生成编译整个 inet 库的 makefile 文件，结束以后输入命令 **make**。到这来，使用命令窗口编译 inet 库就结束了。
- [4] 最后需要在 OMNeT++ IDE 中 Project Explore 窗口空白处右击, 如图，使用 Import 功能导入已经编译好的 inet 库。过程如下图：

- 方法二：OMNeT++ 窗口安装

一样的，在 **INSTALL** 下命令行安装方式下面就是使用 IDE 的安装方式，这个 IDE 的使用方式就是将 INET 库使用 OMNeT++ 打开，当然此时库文件 inet 已经在 **samples** 文件下，我们需要做的就是打开 OMNeT++ IDE ，然后导入整个 inet 工程。

```
If you are using the IDE:
-------------------------
3. Open the OMNeT++ IDE and choose the workspace where you have extracted the inet directo
   The extracted directory must be a subdirectory of the workspace dir.

4. Import the project using: File | Import | General | Existing projects into Workspace.
   Then select the workspace dir as the root directory, and be sure NOT to check the
   "Copy projects into workspace" box. Click Finish.

5. Open the project (if already not open) and wait until the indexer finishes.
   Now you can build the project by pressing CTRL-B (Project | Build all)

6. To run an example from the IDE open the example's directory in the Project Explorer vie
   find the corresponding omnetpp.ini file. Right click on it and select Run As / Simulati
   This should create a Launch Configuration for this example.

If the build was successful, you may try running the demo simulations.
Change into examples/ and type "./rundemo".
```

根据上面的步骤，需要点击：**File | Import | General | Existing projects into Workspace**，导入 inet 整个工程文件，对整个工程进行编译即可。

2.4 常规使用

2.4.1 导入工程

其实我觉得还是有必要把这一小节的内容加入其中，考虑了一下，这个软件的有些操作还是不太一样，可能初学者自己去找需要花大量的时间。　在学习如何导入工程前，先观察一张图：

图 2-4-1 IDE 视图

图 2-4-1 中，左边窗口为 **Project Explorer**，在软件安装首次打开 **IDE** 时，**Project Explorer** 中已经默认有相关的 **samples** 目录下的工程，如果开发者想要打开已有的工程，需要在 **Project Explorer** 窗口空白处右击鼠标，进行 **Import | General | Existing Projects into Workspace**，最后选择工程文件即可，不需要任何设置直接 **finish** 即可。其中相关的中间过程如下：

图 2-4-2 点击 Import 后视图

图 2-4-3 点击 Existing Projects into Workspace 后视图

前面已经描述了相关过程，需要注意的是保证工程文件不要放在有中文名的路径下，如果包括的中文路径，在后期编译工程时，可能在 **ned** 文件下出现大量错误，无法识别 **ned** 文件路径。

2.4.2 程序执行与调试

导入了工程，如何执行程序和调试还是很重要的，尤其是对于 OMNeT++ 工程，在 **omnetpp** 工程下有三种文件：**ned** 、**cpp** 和 **ini**，下面是这三种文件的介绍：**- ned** ：网络拓扑描述文件、简单节点模型和复合节点模型；**- cpp** ：cpp 文件为描述简单节点编程，定义简单节点各种行为；**- ini** ：ned 文件中相关参数的配置，在 ned 文件中一般会设置诸如节点数量的变量，一般有默认值，但是为了修改方便，可以在 ini 文件里边直接配置修改；**- msg** ：消息描述文件，会被 opp_msgc 转化成 *_m.cc/h 文件。

为了更好的说明以上三种文件在一个工作里边的关系，下面展示一张图：

图 2-4-4 编译与执行仿真流程

这张图还是比较重要的，尤其是当你发现你的 ned 或者 ini 文件不启作用的时候，可以根据上面的仿真程序流程思考一下找到 bug 所在。再这张图中，可以看出 **Simulation program** 就是我们工程生成的可执行文件 **exe**，也许你会发现在我们执行程序的时候有两种选择：

- **Local C/C++ Application**
- **OMNeT++ Simulation**

这两种方式执行仿真程序有何不同了，结合图 **2-4-4**，选择第一种执行方式其实就是执行的 **Simulation program**，但是这种执行方式运行的仿真程序没有加入 **ned** 文件和 **ini** 配置文件，因此就是模型节点参数没有配置好或者就没有配置。第二种执行方式就比较完整了，其模型加入了 **ned** 文件和 **ini** 配置文件。其他类似问题读者可自行揣测。　　前面介绍这么多，还是直接进入主题，我们执行 OMNeT++ 工程大致有三种方式：- [1] 直接右击工程文件**->RUN as->Local C/C++ Application** 或者 **OMNeT++ Simulation**；- [2] 选中描述网络的 **ned** 文件，右击执行上面一样的操作；- [3] 选择配置网络参数的 **omnetppp.ini** 文件，右击执行上面一样的操作。

对于上面执行工程的后两种方式一般都是可以的，但是对于第一种方式来说，需要执行 **exe** 文件直接在工程目录下，不能在工程目录的子文件中，否则就只能选择后面两种执行方式。　　关于 OMNeT++ 工程如何调试不再说明，其调试的方式与程序执行的方式相似，同时与其他程序的调试一样使用 **gdb** 调试，其中设置断点、单步调试或者进入函数内部等基本一样，以及添加观察变量。

8 移动自组网中的路由协议

移动自组网是由一群可以移动的、兼具终端及路由功能的移动节点，通过无线链路形成的无中心、多跳、临时性自治系统。[1] 本章主要介绍移动自组网中的常见路由协议。

DSDV

DSR

ADOV

参考文献

[1] 潘宇航. 基于 QoS 感知的移动自组织网络路由协议设计与仿真研究. MS thesis. 西南交通大学, 2014.

3.1 初入 OMNeT++

欢迎来到第三章，本章主要介绍 OMNeT++ 官方已经提供的学习资料有哪些，并以 OMNeT++ 内一系类 tictoc 作为实例进行简单的设计说明，通过本章你可以快速的了解到如何学习 OMNeT++、掌握官方的学习资料和利用 OMNeT++ 可以做哪些事情。

3.2 学习 map

就目前学习 OMNeT++ 的资料来说，网上的资料有：- [1] 《omnet++ 中文使用手册》- [2] 《OMNeT++ 与网络仿真》- [3] 《OMNeT++ 网络仿真》（紫色）

目前较全的资料就上面三种，其中前两种参考价值比较好一些，其中第一本就是 OMNeT++ 官方提供的资料的翻译版，主要介绍范范的仿真程序设计，不能称其为学习教程，应该叫参考资料。第二本《OMNeT++ 与网络仿真》与第一本相比，在仿真程序的设计时更有价值一些，对部分函数接口有介绍，但是没有给出使用场景。其实到目前，作者认为还是官方提供的入门手册对初学者较友好一些，但是问题在于初学的时候我们不知道它的存在，包括我在初学的时候也是恍恍惚惚的，为了使读者在初学的时候就更好的利用这些资料，我在这里总结出官方到底提供了哪些资料。

3.2.1 OMNeT++ 文档与指导书

在 OMNeT++ 安装路径下，官方提供了较多的使用指南，大多数以网页的形式给出。第一个要介绍的就是包括安装手册在内的多个文档入口：

- 路径：**omnetpp-5.2/doc/index.html**

其内容包括从软件安装、初学 Tictoc 多个仿真例子、API 参考到提升篇：IDE 自定义指南和并行仿真指南等，详细如下：

介绍、指导手册 - 安装指导 - IDE 浏览 - TicToc 指导手册

文档 - 仿真手册 - IDE 用户指南 - API 参考书

其他 - IDE 开发者指导 - IDE 自定义指南 - 并行仿真指南 - NEDXML 接口函数

这里都是以中文的形式展现出官方提供的资料目录，而原目录都是以英文的形式给出。

3.2.2 tictoc 指导手册

tictoc 相当于程序中的 **hello world** 级别的例子，初学 OMNeT++ 一般通过仿真修改 **tictoc** 例子，其路径在软件的安装路径下，点击该路径下的 **index.html**：- 路径：**omnetpp-5.2/doc/tictoc-tutorial/index.html**

包括的内容如下：- 开始：一个简单的仿真模型（**tictoc1.ned txc1.cc omnetpp.ini**）- 仿真程序的执行和仿真 - 改进两个节点仿真模型（**tictoc9.ned txc9.cc omnetpp.ini**）- 一个复杂的网络（**tictoc13.ned, tictoc13.msg, txc13.cc, omnetpp.ini**）- 如何添加统计量（**tictoc17.ned, tictoc17.msg, txc17.cc, omnetpp.ini**）- 如何可视化观察仿真结果 - 如何添加参数（在 **omnetpp.ini** 中配置**.ned** 文件需要的参数）

对于以上资料是目前入学 OMNeT++ 较全系统的资料，从工程搭建、调试到添加统计量这些都是实际的网络仿真程序中一般会用到的，比如统计量，一般在网络中包括端到端延迟、入队排队时间、丢包数等。最后的仿真结果可视化观察，OMNeT++ 仿真程序结束后，在 out 文件下会生成仿真结果文件，OMNeT++ 提供可视化工具观察程序中统计的变量，可以转换成直方图和折线图，在后续会详细说明如何使用 OMNeT++ 提供的观察和分析仿真结果工具。

3.2.3 仿真手册

官方也提供了一个较详细的仿真手册，这里还是介绍一下这个手册。

- 路径：**omnetpp-5.2/doc/manual/index.html**

入口如下（绿色部分标出）

Simulation Manual 仿真手册提供了设计一个 OMNeT++ 工程各方面详细的介绍，从某种意义上来说，本手册也许就是 Simulation Manual 手册的一个子集，但是为了使本手册的意义更大，我将结合自己的几个月使用 OMNeT++ 的经验，提供一些在设计网络时可能会出现的问题，以及解决办法。

3.3 个性化 **IDE**

添加这一节，只是个人乐趣而已，我曾经修改代码高亮花了一个下午的时间，每次总是很难找到相关的设置地方，只能说 OMNeT++ 代码高亮设置放置的太隐秘了，到目前为止，我还是不能找到修改.ned 文件的高亮设置（似乎没有这个功能）。相关.cc 文件的设置地方下面会简单指明，其他更为详细的内容需要读者自行发挥。

3.3.1 CPP 高亮设置

首先，进入到 IDE 设置界面：Window 》》Preferences，如图：

cpp 高亮设置

从上图也可以看出，我们需要选择 c/c++》》Syntax Coloring，根据图中的窗口选择我们需要修改的高亮块设置。其中包括以下五种：

- Code: 代码块
- Assembly: 汇编块
- Comments: 注释块
- Preprocessor: 预处理器块
- Doxygen: 其他

你可以根据自己的爱好选择不同的块设置，可选择颜色、加粗、斜体或者下划线等。

3.3.2 其他设置

3.3.2.1 显示行号

在 OMNeT++ 下默认没有显示行号的，但是毕竟有这个毛病，不显示行号就写不下去程序。　如下图所示：

其他设置界面

在这一界面中，我们 Tab 键宽度、显示行号或者当前行背景, 其他设置读者可自行编辑看看。

7 错误记录

在我刚开始使用 OMNeT++ 进行实验的时候，总是会出现一些稀奇古怪的错误，每一个错误都花了我大量的时间，所以在最后一章中总结出来，也许你正好用上。

7.1 在模块加入移动模块之后，仿真出现 nan 错误

问题描述：

对于某些工程加入移动模块以后，编译过程没有出错，但是当执行仿真程序的时候，出现一些 nan 错误提示。需要在仿真配置文件加入：

```
**.constraintAreaMinX = 0m
**.constraintAreaMinY = 0m
**.constraintAreaMinZ = 0m
**.constraintAreaMaxX = 5000m
**.constraintAreaMaxY = 5000m
**.constraintAreaMaxZ = 0m
```

移动模块配置：

```
**.UAV[*].mobilityType = "MassMobility"
**.UAV[*].mobility.initFromDisplayString = true
**.UAV[*].mobility.changeInterval = truncnormal(2s, 0.5s)
**.UAV[*].mobility.changeAngleBy = normal(0deg, 30deg)
**.UAV[*].mobility.speed = truncnormal(250mps, 20mps)
**.UAV[*].mobility.updateInterval = 100ms
```

7.3 工程的例如 **cModule** 之类的类不能高亮显示？

问题描述：

原工程是可以高亮显示的，但是由于我在备份这个程序的时候可能方式不对，我采用的是在文件资源管理器的窗口复制原工程文件夹，没有在软件的窗口进行 rename，可能是这个原因造成的。

解决办法：

在软件的窗口，对工程进行 rename 就行，编译一次，cMdoule 等等关键词就可以高亮了。

4.1 循规蹈矩

在完成第五章后，考虑需要在之前加一章节关于 OMNeT++ 类说明，在这个仿真软件中，主要使用的语言是 C++，因此大多数数据类型是类或者结构，本章还是走其他技术书一样的老路线，注释这些数据类型，对类成员函数进行说明，可能与第五章有些重复的地方，但是其五章更多的偏向于实际应用，可能读者看过这里后，会发现 OMNeT++ 接口是真好用。

4.2 类说明

4.2.1 cModule

为了能更好的解释这个的库的使用，程序清单 4.1 为类 cModule 原型，cModule 类在 OMNeT++ 中表示一个节点的对象，这个节点可以是复合节点或者简单节点，通过这个类，程序员可以访问描述这个节点的.ned 文件中设置的参数，或者是由 omnetpp.ini 传入的参数。简而言之，我们最后就是面向这些类进行网络设计。

程序清单 4.1

```
class SIM_API cModule : public cComponent //implies noncopyable
{
    friend class cGate;
    friend class cSimulation;
    friend class cModuleType;
    friend class cChannelType;

  public:
    /*
     * 模块门的迭代器
     * Usage:
     * for (cModule::GateIterator it(module); !it.end(); ++it) {
     *     cGate *gate = *it;
     *     ...
     * }
     */
    class SIM_API GateIterator
    {
      ...
    };

    /*
```

```
     * 复合模块的子模块迭代器
     * Usage:
     * for (cModule::SubmoduleIterator it(module); !it.end(); ++it) {
     *     cModule *submodule = *it;
     *     ...
     * }
     */
    class SIM_API SubmoduleIterator
    {
      ...
    };

    /*
     * 模块信道迭代器
     * Usage:
     * for (cModule::ChannelIterator it(module); !it.end(); ++it) {
     *     cChannel *channel = *it;
     *     ...
     * }
     */
    class SIM_API ChannelIterator
    {
      ...
    };

  public:
    // internal: currently used by init
    void setRecordEvents(bool e)  {setFlag(FL_RECORD_EVENTS,e);}
    bool isRecordEvents() const  {return flags&FL_RECORD_EVENTS;}

  public:
#ifdef USE_OMNETPP4x_FINGERPRINTS
    // internal: returns OMNeT++ V4.x compatible module ID
    int getVersion4ModuleId() const { return version4ModuleId; }
#endif

    // internal: may only be called between simulations, when no modules exist
    static void clearNamePools();

    // internal utility function. Takes O(n) time as it iterates on the gates
    int gateCount() const;

    // internal utility function. Takes O(n) time as it iterates on the gates
    cGate *gateByOrdinal(int k) const;
    // internal: calls refreshDisplay() recursively
    virtual void callRefreshDisplay() override;

    // internal: return the canvas if exists, or nullptr if not (i.e. no create-on-demand)
    cCanvas *getCanvasIfExists() {return canvas;}

    // internal: return the 3D canvas if exists, or nullptr if not (i.e. no create-on-dema
    cOsgCanvas *getOsgCanvasIfExists() {return osgCanvas;}

  public:

    /** @name Redefined cObject member functions. */
    //@{

    /**
     * Calls v->visit(this) for each contained object.
     * See cObject for more details.
     */
    virtual void forEachChild(cVisitor *v) override;

    /**
     * Sets object's name. Redefined to update the stored fullName string.
     */
    virtual void setName(const char *s) override;

    /**
     * Returns the full name of the module, which is getName() plus the
     * index in square brackets (e.g. "module[4]"). Redefined to add the
     * index.
     */
    virtual const char *getFullName() const override;

    /**
     * Returns the full path name of the module. Example: <tt>"net.node[12].gen"</tt>.
     * The original getFullPath() was redefined in order to hide the global cSimulation
     * instance from the path name.
     */
    virtual std::string getFullPath() const override;

    /**
     * Overridden to add the module ID.
     */
    virtual std::string str() const override;
    //@}
```

```
/** @name Setting up the module. */
//@{

/**
 * Adds a gate or gate vector to the module. Gate vectors are created with
 * zero size. When the creation of a (non-vector) gate of type cGate::INOUT
 * is requested, actually two gate objects will be created, "gatename$i"
 * and "gatename$o". The specified gatename must not contain a "$i" or "$o"
 * suffix itself.
 *
 * CAUTION: The return value is only valid when a non-vector INPUT or OUTPUT
 * gate was requested. nullptr gets returned for INOUT gates and gate vectors.
 */
virtual cGate *addGate(const char *gatename, cGate::Type type, bool isvector=false);

/**
 * Sets gate vector size. The specified gatename must not contain
 * a "$i" or "$o" suffix: it is not possible to set different vector size
 * for the "$i" or "$o" parts of an inout gate. Changing gate vector size
 * is guaranteed NOT to change any gate IDs.
 */
virtual void setGateSize(const char *gatename, int size);

/*
 * 下面的接口是关于模块自己的信息
 */
// 复合模块还是简单模块
virtual bool isSimple() const;

/**
 * Redefined from cComponent to return KIND_MODULE.
 */
virtual ComponentKind getComponentKind() const override  {return KIND_MODULE;}

/**
 * Returns true if this module is a placeholder module, i.e.
 * represents a remote module in a parallel simulation run.
 */
virtual bool isPlaceholder() const  {return false;}

// 返回模块的父模块，对于系统模块，返回 nullptr
virtual cModule *getParentModule() const override;

/**
 * Convenience method: casts the return value of getComponentType() to cModuleType.
 */
cModuleType *getModuleType() const  {return (cModuleType *)getComponentType();}

// 返回模块属性，属性在运行时不能修改
virtual cProperties *getProperties() const override;

// 如何模块是使用向量的形式定义的，返回 true
bool isVector() const  {return vectorSize>=0;}

// 返回模块在向量中的索引
int getIndex() const  {return vectorIndex;}

// 返回这个模块向量的大小，如何该模块不是使用向量的方式定义的，返回 1
int getVectorSize() const  {return vectorSize<0 ? 1 : vectorSize;}

// 与 getVectorSize() 功能相似
_OPPDEPRECATED int size() const  {return getVectorSize();}


/*
 * 子模块相关功能
 */

// 检测该模块是否有子模块
virtual bool hasSubmodules() const {return firstSubmodule!=nullptr;}

// 寻找子模块 name，找到返回模块 ID，否则返回-1
// 如何模块采用向量形式定义，那么需要指明 index
virtual int findSubmodule(const char *name, int index=-1) const;

// 直接得到子模块 name 的指针，没有这个子模块返回 nullptr
// 如何模块采用向量形式定义，那么需要指明 index
virtual cModule *getSubmodule(const char *name, int index=-1) const;

/*
 * 一个更强大的获取模块指针的接口，通过路径获取
 *
 * Examples:
 *   "" means nullptr.
 *   "." means this module;
 *   "<root>" means the toplevel module;
 *   ".sink" means the sink submodule of this module;
 *   ".queue[2].srv" means the srv submodule of the queue[2] submodule;
 *   "^.host2" or ".^.host2" means the host2 sibling module;
 *   "src" or "<root>.src" means the src submodule of the toplevel module;
```

```
     *   "Net.src" also means the src submodule of the toplevel module, provided
     *   it is called Net.
     *
     *  @see cSimulation::getModuleByPath()
     */
    virtual cModule *getModuleByPath(const char *path) const;

    /*
     * 门的相关操作
     */

    /**
     * Looks up a gate by its name and index. Gate names with the "$i" or "$o"
     * suffix are also accepted. Throws an error if the gate does not exist.
     * The presence of the index parameter decides whether a vector or a scalar
     * gate will be looked for.
     */
    virtual cGate *gate(const char *gatename, int index=-1);

    /**
     * Looks up a gate by its name and index. Gate names with the "$i" or "$o"
     * suffix are also accepted. Throws an error if the gate does not exist.
     * The presence of the index parameter decides whether a vector or a scalar
     * gate will be looked for.
     */
    const cGate *gate(const char *gatename, int index=-1) const {
        return const_cast<cModule *>(this)->gate(gatename, index);
    }


    /**
     * Returns the "$i" or "$o" part of an inout gate, depending on the type
     * parameter. That is, gateHalf("port", cGate::OUTPUT, 3) would return
     * gate "port$o[3]". Throws an error if the gate does not exist.
     * The presence of the index parameter decides whether a vector or a scalar
     * gate will be looked for.
     */
    const cGate *gateHalf(const char *gatename, cGate::Type type, int index=-1) const {
        return const_cast<cModule *>(this)->gateHalf(gatename, type, index);
    }

    // 检测是否有门
    virtual bool hasGate(const char *gatename, int index=-1) const;

    // 寻找门，如果没有返回-1，找到返回门 ID
```

```
virtual int findGate(const char *gatename, int index=-1) const;

// 通过 ID 得到门地址，目前我还没有用到过
const cGate *gate(int id) const {return const_cast<cModule *>(this)->gate(id);}

// 删除一个门（很少用）
virtual void deleteGate(const char *gatename);


//返回模块门的名字，只是基本名字 (不包括向量门的索引, "[]" or the "$i"/"$o")
virtual std::vector<const char *> getGateNames() const;

// 检测门（向量门）类型，可以标明"$i","$o"
virtual cGate::Type gateType(const char *gatename) const;

// 检测是否是向量门，可以标明"$i","$o"
virtual bool isGateVector(const char *gatename) const;

// 得到门的大小，可以指明"$i","$o"
virtual int gateSize(const char *gatename) const;

// 对于向量门，返回 gate0 的 ID 号
// 对于标量 ID，返回 ID
// 一个公式：ID = gateBaseId + index
// 如果没有该门，抛出一个错误
virtual int gateBaseId(const char *gatename) const;

/**
 * For compound modules, it checks if all gates are connected inside
 * the module (it returns <tt>true</tt> if they are OK); for simple
 * modules, it returns <tt>true</tt>. This function is called during
 * network setup.
 */
virtual bool checkInternalConnections() const;

/**
 * This method is invoked as part of a send() call in another module.
 * It is called when the message arrives at a gates in this module which
 * is not further connected, that is, the gate's getNextGate() method
 * returns nullptr. The default, cModule implementation reports an error
 * ("message arrived at a compound module"), and the cSimpleModule
 * implementation inserts the message into the FES after some processing.
 */
virtual void arrived(cMessage *msg, cGate *ongate, simtime_t t);
//@}
```

```
/*
 * 公用的
 */
// 在父模块中寻找某个参数，没找到抛出 cRuntimeError
virtual cPar& getAncestorPar(const char *parname);

/**
 * Returns the default canvas for this module, creating it if it hasn't
 * existed before.
 */
virtual cCanvas *getCanvas() const;

/**
 * Returns the default 3D (OpenSceneGraph) canvas for this module, creating
 * it if it hasn't existed before.
 */
virtual cOsgCanvas *getOsgCanvas() const;

// 设置是否在此模块的图形检查器上请求内置动画。
virtual void setBuiltinAnimationsAllowed(bool enabled) {setFlag(FL_BUILTIN_ANIMATIONS,

/**
 * Returns true if built-in animations are requested on this module's
 * graphical inspector, and false otherwise.
 */
virtual bool getBuiltinAnimationsAllowed() const {return flags & FL_BUILTIN_ANIMATIONS
//@}

/** @name Public methods for invoking initialize()/finish(), redefined from cComponent
 * initialize(), numInitStages(), and finish() are themselves also declared in
 * cComponent, and can be redefined in simple modules by the user to perform
 * initialization and finalization (result recording, etc) tasks.
 */
//@{
/**
 * Interface for calling initialize() from outside.
 */
virtual void callInitialize() override;

/**
 * Interface for calling initialize() from outside. It does a single stage
 * of initialization, and returns <tt>true</tt> if more stages are required.
 */
virtual bool callInitialize(int stage) override;
```

```
    /**
     * Interface for calling finish() from outside.
     */
    virtual void callFinish() override;


    /*
     * 动态模块创建
     */

    /**
     * Creates a starting message for modules that need it (and recursively
     * for its submodules).
     */
    virtual void scheduleStart(simtime_t t);

    // 删除自己
    virtual void deleteModule();

    // 移动该模块到另一个父模块下，一般用于移动场景。规则较复杂，可到原头文件查看使用说明
    virtual void changeParentTo(cModule *mod);
};
```

cModule 是 OMNeT++ 中用于代表一个模块的对象实体，如果你在编写网络仿真代码时，这个模块可以是简单模块或者复合模块，当需要得到这个模块相关属性时可以考虑到这个 **cModule** 类里边找找，说不定有意外的惊喜，也许有现成的函数实现你需要的功能。下面将这个类原型解剖看看：

- 迭代器：**GateIterator**

```
usage:
for (cModule::GateIterator it(module); !it.end(); ++it) {
        cGate *gate = *it;
        ...
}
```

该迭代器可用于遍历模块 **module** 的门向量，得到该门可用于其他作用。

- 迭代器：**SubmoduleIterator**

```
usage:
for (cModule::SubmoduleIterator it(module); !it.end(); ++it) {
        cModule *submodule = *it;
        ...
}
```

对于一个复合模块，包括多个简单模块或者复合模块，可使用该迭代器进行遍历操作，在第五章涉及到这个迭代器的使用。

- 迭代器：**ChannelIterator**

```
usage:
for (cModule::ChannelIterator it(module); !it.end(); ++it) {
        cChannel *channel = *it;
        ...
}
```

可用于遍历该模块的所有的信道。

4.2.2 cPar

cPar 同样是我们设置网络时不可避免的类，通过 cPar 得到节点在网络拓扑文件和配置文件中设置的参数，浏览完 cPar 所有成员函数，可以看出 cPar 基本提供了网络设计者想要的所有数据转换接口。

```
class SIM_API cPar : public cObject
{
    friend class cComponent;
  public:
    enum Type {
        BOOL = 'B',
        DOUBLE = 'D',
        LONG = 'L',
        STRING = 'S',
        XML = 'X'
    };

  private:
    cComponent *ownerComponent;
    cParImpl *p;
    cComponent *evalContext;

  private:
    // private constructor and destructor -- only cComponent is allowed to create paramete
    cPar() {ownerComponent = evalContext = nullptr; p = nullptr;}
    virtual ~cPar();
    // internal, called from cComponent
    void init(cComponent *ownercomponent, cParImpl *p);
    // internal
    void moveto(cPar& other);
    // internal: called each time before the value of this object changes.
    void beforeChange();
    // internal: called each time after the value of this object changes.
    void afterChange();

  public:
    // internal, used by cComponent::finalizeParameters()
    void read();
    // internal, used by cComponent::finalizeParameters()
    void finalize();
    // internal: applies the default value if there is one
    void acceptDefault();
    // internal
    void setImpl(cParImpl *p);
    // internal
    cParImpl *impl() const {return p;}
    // internal
    cParImpl *copyIfShared();

#ifdef SIMFRONTEND_SUPPORT
    // internal
    virtual bool hasChangedSince(int64_t lastRefreshSerial);
#endif

  public:
    /** @name Redefined cObject methods */
    //@{
    /**
     * Assignment operator.
     */
    void operator=(const cPar& other);

    // 返回参数的名字
    virtual const char *getName() const override;
```

```
// 以字符串的形式返回参数
virtual std::string str() const override;

/**
 * Returns the component (module/channel) this parameter belongs to.
 * Note: return type is cObject only for technical reasons, it can be
 * safely cast to cComponent.
 */
virtual cObject *getOwner() const override; // note: cannot return cComponent* (covari

/**
 * Calls v->visit(this) for contained objects.
 * See cObject for more details.
 */
virtual void forEachChild(cVisitor *v) override;
//@}

/** @name Type, flags. */
//@{
/**
 * Returns the parameter type
 */
Type getType() const;

/**
 * Returns the given type as a string.
 */
static const char *getTypeName(Type t);

/**
 * Returns true if the stored value is of a numeric type.
 */
bool isNumeric() const;

/**
 * Returns true if this parameter is marked in the NED file as "volatile".
 * This flag affects the operation of setExpression().
 */
bool isVolatile() const;

/**
 * Returns false if the stored value is a constant, and true if it is
 * an expression. (It is not examined whether the expression yields
 * a constant value.)
 */
bool isExpression() const;

/**
 * Returns true if the parameter value expression is shared among several
 * modules to save memory. This flag is purely informational, and whether
 * a parameter is shared or not does not affect operation at all.
 */
bool isShared() const;

/**
 * Returns true if the parameter is assigned a value, and false otherwise.
 * Parameters of an already initialized module or channel are guaranteed to
 * assigned, so this method will return true for them.
 */
bool isSet() const;

/**
 * Returns true if the parameter is set (see isSet()) or contains a default
 * value, and false otherwise. Parameters of an already initialized module or
 * channel are guaranteed to be assigned, so this method will return true for them.
 */
bool containsValue() const;

/**
 * Return the properties for this parameter. Properties cannot be changed
 * at runtime.
 */
cProperties *getProperties() const;
//@}

/** @name Setter functions. Note that overloaded assignment operators also exist. */
//@{

/**
 * Sets the value to the given bool value.
 */
cPar& setBoolValue(bool b);

/**
 * Sets the value to the given long value.
 */
cPar& setLongValue(long l);

/**
```

```
     * Sets the value to the given double value.
     */
    cPar& setDoubleValue(double d);

    /**
     * Sets the value to the given string value.
     * The cPar will make its own copy of the string. nullptr is also accepted
     * and treated as an empty string.
     */
    cPar& setStringValue(const char *s);

    /**
     * Sets the value to the given string value.
     */
    cPar& setStringValue(const std::string& s)  {setStringValue(s.c_str()); return *this;}

    /**
     * Sets the value to the given cXMLElement.
     */
    cPar& setXMLValue(cXMLElement *node);

    /**
     * Sets the value to the given expression. This object will assume
     * the responsibility to delete the expression object.
     *
     * The evalcontext parameter determines the module or channel in the
     * context of which the expression will be evaluated. If evalcontext
     * is nullptr, the owner of this parameter will be used.
     *
     * Note: if the parameter is marked as non-volatile (isVolatile()==false),
     * one should not set an expression as value. This is not enforced
     * by cPar though.
     *
     * @see getOwner(), getEvaluationContext(), setEvaluationContext()
     */
    cPar& setExpression(cExpression *e, cComponent *evalcontext=nullptr);

    /**
     * If the parameter contains an expression (see isExpression()), this method
     * sets the evaluation context for the expression.
     *
     * @see getEvaluationContext(), isExpression(), setExpression()
     */
    void setEvaluationContext(cComponent *ctx)  {evalContext = ctx;}
    //@}
```

```
/** @name Getter functions. Note that overloaded conversion operators also exist. */
//@{

/**
 * Returns value as a boolean. The cPar type must be BOOL.
 */
bool boolValue() const;

/**
 * Returns value as long. The cPar type must be LONG or DOUBLE.
 */
long longValue() const;

/**
 * Returns value as double. The cPar type must be LONG or DOUBLE.
 */
double doubleValue() const;

/**
 * Returns the parameter's unit ("s", "mW", "Hz", "bps", etc),
 * as declared with the @unit property of the parameter in NED,
 * or nullptr if no unit was specified. Unit is only valid for LONG and DOUBLE
 * types.
 */
const char *getUnit() const;

/**
 * Returns value as const char *. The cPar type must be STRING.
 * This method may only be invoked when the parameter's value is a
 * string constant and not the result of expression evaluation, otherwise
 * an error is thrown. This practically means this method cannot be used
 * on parameters declared as "volatile string" in NED; they can only be
 * accessed using stdstringValue().
 */
const char *stringValue() const;

/**
 * Returns value as string. The cPar type must be STRING.
 */
std::string stdstringValue() const;

/**
 * Returns value as pointer to cXMLElement. The cPar type must be XML.
 *
```

```
     * The lifetime of the returned object tree is undefined, but it is
     * valid at least until the end of the current simulation event or
     * initialize() call. Modules are expected to process their XML
     * configurations at once (within one event or within initialize()),
     * and not hang on to pointers returned from this method. The reason
     * for the limited lifetime is that this method may return pointers to
     * objects stored in an internal XML document cache, and the simulation
     * kernel reserves the right to discard cached XML documents at any time
     * to free up memory, and re-load them on demand (i.e. when xmlValue() is
     * called again).
     */
    cXMLElement *xmlValue() const;

    /**
     * Returns pointer to the expression stored by the object, or nullptr.
     */
    cExpression *getExpression() const;

    /**
     * If the parameter contains an expression, this method returns the
     * module or channel in the context of which the expression will be
     * evaluated. (The context affects the resolution of parameter
     * references, and NED operators like <tt>index</tt> or <tt>sizeof()</tt>.)
     * If the parameter does not contain an expression, the return value is
     * undefined.
     *
     * @see isExpression(), setEvaluationContext()
     */
    cComponent *getEvaluationContext() const  {return evalContext;}
    //@}

    /** @name Miscellaneous utility functions. */
    //@{
    /**
     * For non-const values, replaces the stored expression with its
     * evaluation.
     */
    void convertToConst();

    /**
     * Converts the value from string, and stores the result.
     * If the text cannot be parsed, an exception is thrown, which
     * can be caught as std::runtime_error& if necessary.
     *
     * Note: this method understands expressions too, but does NOT handle
     * the special values "default" and "ask".
     */
    void parse(const char *text);
    //@}

    /** @name Overloaded assignment and conversion operators. */
    //@{

    /**
     * Equivalent to setBoolValue().
     */
    cPar& operator=(bool b)  {return setBoolValue(b);}

    /**
     * Converts the argument to long, and calls setLongValue().
     */
    cPar& operator=(char c)  {return setLongValue((long)c);}

    /**
     * Converts the argument to long, and calls setLongValue().
     */
    cPar& operator=(unsigned char c)  {return setLongValue((long)c);}

    /**
     * Converts the argument to long, and calls setLongValue().
     */
    cPar& operator=(int i)  {return setLongValue((long)i);}

    /**
     * Converts the argument to long, and calls setLongValue().
     */
    cPar& operator=(unsigned int i)  {return setLongValue((long)i);}

    /**
     * Converts the argument to long, and calls setLongValue().
     */
    cPar& operator=(short i)  {return setLongValue((long)i);}

    /**
     * Converts the argument to long, and calls setLongValue().
     */
    cPar& operator=(unsigned short i)  {return setLongValue((long)i);}

    /**
     * Equivalent to setLongValue().
```

```
 */
cPar& operator=(long l)  {return setLongValue(l);}

/**
 * Converts the argument to long, and calls setLongValue().
 */
cPar& operator=(unsigned long l) {return setLongValue((long)l);}

/**
 * Equivalent to setDoubleValue().
 */
cPar& operator=(double d)  {return setDoubleValue(d);}

/**
 * Converts the argument to double, and calls setDoubleValue().
 */
cPar& operator=(long double d)  {return setDoubleValue((double)d);}

// 等同于 setStringValue() 函数
cPar& operator=(const char *s)  {return setStringValue(s);}

// 等同于 setStringValue() 函数
cPar& operator=(const std::string& s)  {return setStringValue(s);}

 // 等同于 setXMLValue() 函数
cPar& operator=(cXMLElement *node)  {return setXMLValue(node);}

operator bool() const  {return boolValue();}

operator char() const  {return (char)longValue();}

operator unsigned char() const  {return (unsigned char)longValue();}

operator int() const  {return (int)longValue();}

operator unsigned int() const  {return (unsigned int)longValue();}

operator short() const  {return (short)longValue();}

operator unsigned short() const  {return (unsigned short)longValue();}

// 返回 long 值，与 longValue() 相同
operator long() const  {return longValue();}

/**
```

```
    // 调用 longValue()，转换结果为 unsigned long 类型
     */
    operator unsigned long() const  {return longValue();}

    // 返回 double 值，与 doubleValue() 相同
    operator double() const  {return doubleValue();}

    /**
    // 调用 doubleValue()，将结果转换成 long double 类型返回
     */
    operator long double() const  {return doubleValue();}

    // 与 stringValue()
    operator const char *() const  {return stringValue();}

    // 与 stdstringValue() 功能一样
    operator std::string() const  {return stdstringValue();}

    // 与 xmlVlaue() 等同。注意：返回对象树的生命周期被限制了，具体看 xmlValue 说明。
    operator cXMLElement *() const  {return xmlValue();}
};
```

4.2.3 cGate

如果你需要在网络仿真运行时，动态实现两个节点之间的连接或者断开，那么你就需要在程序中用到这个类。

```
class SIM_API cGate : public cObject, noncopyable
{
    friend class cModule;
    friend class cModuleGates;
    friend class cPlaceholderModule;

  public:
    /**
     * Gate type
     */
    enum Type {
        NONE = 0,
        INPUT = 'I',
        OUTPUT = 'O',
        INOUT = 'B'
    };

  protected:
    // internal
    struct SIM_API Name
    {
        opp_string name;  // "foo"
        opp_string namei; // "foo$i"
        opp_string nameo; // "foo$o"
        Type type;
        Name(const char *name, Type type);
        bool operator<(const Name& other) const;
    };

  public:
    // Internal data structure, only public for technical reasons (GateIterator).
    // One instance per module and per gate vector/gate pair/gate.
    // Note: gate name and type are factored out to a global pool.
    // Note2: to reduce sizeof(Desc), "size" might be stored in input.gatev[0],
    // although it might not be worthwhile the extra complication and CPU cycles.
    //
    struct Desc
    {
        cModule *owner;
        Name *name;  // pooled (points into cModule::namePool)
        int vectorSize; // gate vector size, or -1 if scalar gate; actually allocated size
        union Gates { cGate *gate; cGate **gatev; };
        Gates input;
        Gates output;

        Desc() {owner=nullptr; vectorSize=-1; name=nullptr; input.gate=output.gate=nullptr;
        bool inUse() const {return name!=nullptr;}
        Type getType() const {return name->type;}
        bool isVector() const {return vectorSize>=0;}
        const char *nameFor(Type t) const {return (t==INOUT||name->type!=INOUT) ? name->nar
        int indexOf(const cGate *g) const {return (g->pos>>2)==-1 ? 0 : g->pos>>2;}
        bool deliverOnReceptionStart(const cGate *g) const {return g->pos&2;}
        Type getTypeOf(const cGate *g) const {return (g->pos&1)==0 ? INPUT : OUTPUT;}
        bool isInput(const cGate *g) const {return (g->pos&1)==0;}
        bool isOutput(const cGate *g) const {return (g->pos&1)==1;}
        int gateSize() const {return vectorSize>=0 ? vectorSize : 1;}
        void setInputGate(cGate *g) {ASSERT(getType()!=OUTPUT && !isVector()); input.gate=g
        void setOutputGate(cGate *g) {ASSERT(getType()!=INPUT && !isVector()); output.gate=
        void setInputGate(cGate *g, int index) {ASSERT(getType()!=OUTPUT && isVector()); in
        void setOutputGate(cGate *g, int index) {ASSERT(getType()!=INPUT && isVector()); ou
        static int capacityFor(int size) {return size<8 ? (size+1)&~1 : size<32 ? (size+3)
    };

  protected:
    Desc *desc; // descriptor of the gate or gate vector, stored in cModule
    int pos;    // b0: input(0) or output(1); b1: deliverOnReceptionStart bit;
                // rest (pos>>2): array index, or -1 if scalar gate

    int connectionId;   // uniquely identifies the connection between *this and *nextgatep
    cChannel *channel;  // channel object (if exists)
    cGate *prevGate;    // previous and next gate in the path
    cGate *nextGate;

    static int lastConnectionId;

  protected:
    // internal: constructor is protected because only cModule is allowed to create instan
    explicit cGate();

    // also protected: only cModule is allowed to delete gates
    virtual ~cGate();

    // internal
    static void clearFullnamePool();

    // internal
    void installChannel(cChannel *chan);

    // internal
    void checkChannels() const;

#ifdef SIMFRONTEND_SUPPORT
    // internal
    virtual bool hasChangedSince(int64_t lastRefreshSerial);
#endif

  public:
    /** @name Redefined cObject member functions */
    //@{
    /*
     * 例如返回门 out
     */
    virtual const char *getName() const override;

    /*
```

```
     * 与 getName() 不同，需要返回门索引，例如 out[4]
     */
    virtual const char *getFullName() const override;

    /**
     * Calls v->visit(this) for each contained object.
     * See cObject for more details.
     */
    virtual void forEachChild(cVisitor *v) override;

    /**
     * Produces a one-line description of the object's contents.
     * See cObject for more details.
     */
    virtual std::string str() const override;

    /**
     * Returns the owner module of this gate.
     */
    virtual cObject *getOwner() const override; // note: cannot return cModule* (covariant
    //@}

    /**
     * This function is called internally by the send() functions and
     * channel classes' deliver() to deliver the message to its destination.
     * A false return value means that the message object should be deleted
     * by the caller. (This is used e.g. with parallel simulation, for
     * messages leaving the partition.)
     */
    virtual bool deliver(cMessage *msg, simtime_t at);

    /** @name Connecting the gate. */
    //@{
    /**
     * Connects the gate to another gate, using the given channel object
     * (if one is specified). This method can be used to manually create
     * connections for dynamically created modules.
     *
     * This method invokes callInitialize() on the channel object, unless the
     * compound module containing this connection is not yet initialized
     * (then it assumes that this channel will be initialized as part of the
     * compound module initialization process.) To leave the channel
     * uninitialized, specify true for the leaveUninitialized parameter.
     *
     * If the gate is already connected, an error will occur. The gate
```

```
 * argument cannot be nullptr, that is, you cannot use this function
 * to disconnect a gate; use disconnect() for that.
 *
 * Note: When you set channel parameters after channel initialization,
 * make sure the channel class is implemented so that the changes take
 * effect; i.e. the channel should either override and properly handle
 * handleParameterChange(), or should not cache any values from parameters.
 */
cChannel *connectTo(cGate *gate, cChannel *channel=nullptr, bool leaveUninitialized=fa

/**
 * Disconnects the gate, and also deletes the associated channel object
 * if one has been set. disconnect() must be invoked on the source gate
 * ("from" side) of the connection.
 *
 * The method has no effect if the gate is not connected.
 */
void disconnect();

/**
 * Disconnects the gate, then connects it again to the same gate, with the
 * given channel object (if not nullptr). The gate must be connected.
 *
 * @see connectTo()
 */
cChannel *reconnectWith(cChannel *channel, bool leaveUninitialized=false);
//@}

/** @name Information about the gate. */
//@{
/**
 * Returns the gate name without index and potential "$i"/"$o" suffix.
 */
const char *getBaseName() const;

/**
 * Returns the suffix part of the gate name ("$i", "$o" or "").
 */
const char *getNameSuffix() const;

/**
 * Returns the properties for this gate. Properties cannot be changed
 * at runtime.
 */
cProperties *getProperties() const;
```

```
/**
 * Returns the gate's type, cGate::INPUT or cGate::OUTPUT. (It never returns
 * cGate::INOUT, because a cGate object is always either the input or
 * the output half of an inout gate ("name$i" or "name$o").
 */
Type getType() const  {return desc->getTypeOf(this);}

/**
 * Returns the given type as a string.
 */
static const char *getTypeName(Type t);

/**
 * Returns a pointer to the owner module of the gate.
 */
cModule *getOwnerModule() const;

/**
 * Returns the gate ID, which uniquely identifies the gate within the
 * module. IDs are guaranteed to be contiguous within a gate vector:
 * <tt>module->gate(id+index) == module->gate(id)+index</tt>.
 *
 * Gate IDs are stable: they are guaranteed not to change during
 * simulation. (This is a new feature of \opp 4.0. In earlier releases,
 * gate IDs could change when the containing gate vector was resized.)
 *
 * Note: As of \opp 4.0, gate IDs are no longer small integers, and
 * cannot be used for iterating over the gates of a module.
 * Use cModule::GateIterator for iteration.
 */
int getId() const;

/**
 * Returns true if the gate is part of a gate vector.
 */
bool isVector() const  {return desc->isVector();}

/**
 * If the gate is part of a gate vector, returns the ID of the first
 * element in the gate vector. Otherwise, it returns the gate's ID.
 */
int getBaseId() const;

/**
```

```
     * If the gate is part of a gate vector, returns the gate's index in the vector.
     * Otherwise, it returns 0.
     */
    int getIndex() const  {return desc->indexOf(this);}

    /**
     * If the gate is part of a gate vector, returns the size of the vector.
     * For non-vector gates it returns 1.
     *
     * The gate vector size can also be obtained by calling the cModule::gateSize().
     */
    int getVectorSize() const  {return desc->gateSize();}

    /**
     * Alias for getVectorSize().
     */
    int size() const  {return getVectorSize();}

    /**
     * Returns the channel object attached to this gate, or nullptr if there is
     * no channel. This is the channel between this gate and this->getNextGate(),
     * that is, channels are stored on the "from" side of the connections.
     */
    cChannel *getChannel() const  {return channel;}

    /**
     * This method may only be invoked on input gates of simple modules.
     * Messages with nonzero length then have a nonzero
     * transmission duration (and thus, reception duration on the other
     * side of the connection). By default, the delivery of the message
     * to the module marks the end of the reception. Setting this bit will cause
     * the channel to deliver the message to the module at the start of the
     * reception. The duration that the reception will take can be extracted
     * from the message object, by its getDuration() method.
     */
    void setDeliverOnReceptionStart(bool d);

    /**
     * Returns whether messages delivered through this gate will mark the
     * start or the end of the reception process (assuming nonzero message length
     * and data rate on the channel.)
     *
     * @see setDeliverOnReceptionStart()
     */
    bool getDeliverOnReceptionStart() const  {return pos&2;}
```

```
//@}

/** @name Transmission state. */
//@{
/**
 * Typically invoked on an output gate, this method returns <i>the</i>
 * channel in the connection path that supports datarate (as determined
 * by cChannel::isTransmissionChannel(); it is guaranteed that there can be
 * at most one such channel per path). If there is no such channel,
 * an error is thrown.
 *
 * This method only checks the segment of the connection path that
 * <i>starts</i> at this gate, so, for example, it is an error to invoke
 * it on a simple module input gate.
 *
 * Note: this method searches the connection path linearly, so at
 * performance-critical places it may be better to cache its return
 * value (provided that connections are not removed or created dynamically
 * during simulation.)
 *
 * @see cChannel::isTransmissionChannel()
 */
cChannel *getTransmissionChannel() const;

/**
 * Like getTransmissionChannel(), but returns nullptr instead of throwing
 * an error if there is no transmission channel in the path.
 */
cChannel *findTransmissionChannel() const;

/**
 * Typically invoked on an input gate, this method searches the reverse
 * path (i.e. calls getPreviousGate() repeatedly) for the transmission
 * channel. It is guaranteed that there can be at most one such channel
 * per path. If no transmission channel is found, the method throws an error.
 *
 * @see getTransmissionChannel(), cChannel::isTransmissionChannel()
 */
cChannel *getIncomingTransmissionChannel() const;

/**
 * Like getIncomingTransmissionChannel(), but returns nullptr instead of
 * throwing an error if there is no transmission channel in the reverse
 * path.
 */
```

```
cChannel *findIncomingTransmissionChannel() const;
//@}

/** @name Gate connectivity. */
//@{

/**
 * Returns the previous gate in the series of connections (the path) that
 * contains this gate, or nullptr if this gate is the first one in the path.
 * (E.g. for a simple module output gate, this function will return nullptr.)
 */
cGate *getPreviousGate() const {return prevGate;}

/**
 * Returns the next gate in the series of connections (the path) that
 * contains this gate, or nullptr if this gate is the last one in the path.
 * (E.g. for a simple module input gate, this function will return nullptr.)
 */
cGate *getNextGate() const   {return nextGate;}

/**
 * Returns an ID that uniquely identifies the connection between this gate
 * and the next gate in the path (see getNextGate()) during the lifetime of
 * the simulation. (Disconnecting and then reconnecting the gate results
 * in a new connection ID being assigned.) The method returns -1 if the gate
 * is unconnected.
 */
int getConnectionId() const  {return connectionId;}

/**
 * Return the ultimate source of the series of connections
 * (the path) that contains this gate.
 */
cGate *getPathStartGate() const;

/**
 * Return the ultimate destination of the series of connections
 * (the path) that contains this gate.
 */
cGate *getPathEndGate() const;

/**
 * Determines if a given module is in the path containing this gate.
 */
bool pathContains(cModule *module, int gateId=-1);
```

```
/**
 * Returns true if the gate is connected outside (i.e. to one of its
 * sibling modules or to the parent module).
 *
 * This means that for an input gate, getPreviousGate() must be non-nullptr; for an ou
 * gate, getNextGate() must be non-nullptr.
 */
bool isConnectedOutside() const;

/**
 * Returns true if the gate (of a compound module) is connected inside
 * (i.e. to one of its submodules).
 *
 * This means that for an input gate, getNextGate() must be non-nullptr; for an output
 * gate, getPreviousGate() must be non-nullptr.
 */
bool isConnectedInside() const;

/**
 * Returns true if the gate fully connected. For a compound module gate
 * this means both isConnectedInside() and isConnectedOutside() are true;
 * for a simple module, only isConnectedOutside() is checked.
 */
bool isConnected() const;

/**
 * Returns true if the path (chain of connections) containing this gate
 * starts and ends at a simple module.
 */
bool isPathOK() const;
//@}

/** @name Display string. */
//@{
/**
 * Returns the display string for the gate, which controls the appearance
 * of the connection arrow starting from gate. The display string is stored
 * in the channel associated with the connection. If there is no channel,
 * this call creates an installs a cIdealChannel to hold the display string.
 */
cDisplayString& getDisplayString();

/**
 * Shortcut to <tt>getDisplayString().set(dispstr)</tt>.
```

```
     */
    void setDisplayString(const char *dispstr);
    //@}
};
```

4.2.4 cTopology

## 4.2.5 cExpression

4.2.6 EV 类

一个对调试程序有帮助的类。

```
//==========================================================================
//   CLOG.H  -  header for
//                     OMNeT++/OMNEST
//            Discrete System Simulation in C++
//
//==========================================================================

/*--------------------------------------------------------------*
  Copyright (C) 1992-2017 Andras Varga
  Copyright (C) 2006-2017 OpenSim Ltd.

  This file is distributed WITHOUT ANY WARRANTY. See the file
  `license' for details on this and other legal matters.
*--------------------------------------------------------------*/

#ifndef __OMNETPP_CLOG_H
#define __OMNETPP_CLOG_H

#include <ctime>
#include <sstream>
#include "simkerneldefs.h"

namespace omnetpp {

class cObject;
class cComponent;

/**
 * @brief Classifies log messages based on detail and importance.
 *
 * @ingroup Logging
 */
enum LogLevel
{
    /**
     * The lowest log level; it should be used for low-level implementation-specific
     * technical details that are mostly useful for the developers/maintainers of the
     * component. For example, a MAC layer protocol component could log control flow
     * in loops and if statements, entering/leaving methods and code blocks using this
     * log level.
     */
    LOGLEVEL_TRACE,

    /**
     * This log level should be used for high-level implementation-specific technical
     * details that are most likely important for the developers/maintainers of the
     * component. These messages may help to debug various issues when one is looking
     * at the code. For example, a MAC layer protocol component could log updates to
     * internal state variables, updates to complex data structures using this log level.
     */
    LOGLEVEL_DEBUG,

    /**
     * This log level should be used for low-level protocol-specific details that
     * may be useful and understandable by the users of the component. These messages
     * may help to track down various protocol-specific issues without actually looking
     * too deep into the code. For example, a MAC layer protocol component could log
     * state machine updates, acknowledge timeouts and selected back-off periods using
     * this level.
     */
    LOGLEVEL_DETAIL,

    /**
     * This log level should be used for high-level protocol specific details that
     * are most likely important for the users of the component. For example, a MAC
     * layer protocol component could log successful packet receptions and successful
     * packet transmissions using this level.
     */
    LOGLEVEL_INFO,
    /**
     * This log level should be used for exceptional (non-error) situations that
     * may be important for users and rarely occur in the component. For example,
     * a MAC layer protocol component could log detected bit errors using this level.
     */
    LOGLEVEL_WARN,

    /**
     * This log level should be used for recoverable (non-fatal) errors that allow
     * the component to continue normal operation. For example, a MAC layer protocol
     * component could log unsuccessful packet receptions and unsuccessful packet
     * transmissions using this level.
     */
    LOGLEVEL_ERROR,

    /**
     * The highest log level; it should be used for fatal (unrecoverable) errors
     * that prevent the component from further operation. It doesn't mean that
     * the simulation must stop immediately (because in such cases the code should
     * throw a cRuntimeError), but rather that the a component is unable to continue
     * normal operation. For example, a special purpose recording component may be
     * unable to continue recording due to the disk being full.
     */
    LOGLEVEL_FATAL,

    /**
     * Not a real log level, it completely disables logging when set.
     */
    LOGLEVEL_OFF,
};

/**
 * @brief For compile-time filtering of logs.
 *
 * One is free to define this macro before including <omnetpp.h>, or redefine
 * it any time. The change will affect subsequent log statements.
 * Log statements that use lower log levels than the one specified
 * by this macro will not be compiled into the executable.
 *
 * @ingroup Logging
 */
#ifndef COMPILETIME_LOGLEVEL
#ifdef NDEBUG
#define COMPILETIME_LOGLEVEL omnetpp::LOGLEVEL_DETAIL
#else
```

```
#define COMPILETIME_LOGLEVEL omnetpp::LOGLEVEL_TRACE
#endif
#endif

/**
 * @brief This predicate determines if a log statement gets compiled into the
 * executable.
 *
 * One is free to define this macro before including <omnetpp.h>, or redefine
 * it any time. The change will affect subsequent log statements.
 *
 * @ingroup Logging
 */
#ifndef COMPILETIME_LOG_PREDICATE
#define COMPILETIME_LOG_PREDICATE(object, logLevel, category) (logLevel >= COMPILETIME_LOGI
#endif

/**
 * @brief This class groups logging related functionality.
 *
 * @see LogLevel
 * @ingroup Logging
 */
class SIM_API cLog
{
  public:
    typedef bool (*NoncomponentLogPredicate)(const void *object, LogLevel logLevel, const c
    typedef bool (*ComponentLogPredicate)(const cComponent *object, LogLevel logLevel, cons

  public:
    /**
     * This log level specifies a globally applied runtime modifiable filter. This is
     * the fastest runtime filter, it works with a simple integer comparison at the call
     * site.
     */
    static LogLevel logLevel;

    /**
     * This predicate determines if a log statement is executed for log statements
     * that occur outside module or channel member functions. This is a customization
     * point for logging.
     */
    static NoncomponentLogPredicate noncomponentLogPredicate;

    /**
```

```
     * This predicate determines if a log statement is executed for log statements
     * that occur in module or channel member functions. This is a customization
     * point for logging.
     */
    static ComponentLogPredicate componentLogPredicate;

  public:
    /**
     * Returns a human-readable string representing the provided log level.
     */
    static const char *getLogLevelName(LogLevel logLevel);

    /**
     * Returns the associated log level for the provided human-readable string.
     */
    static LogLevel resolveLogLevel(const char *name);

    static inline bool runtimeLogPredicate(const void *object, LogLevel logLevel, const cha
    { return noncomponentLogPredicate(object, logLevel, category); }

    static inline bool runtimeLogPredicate(const cComponent *object, LogLevel logLevel, con
    { return componentLogPredicate(object, logLevel, category); }

    static bool defaultNoncomponentLogPredicate(const void *object, LogLevel logLevel, cons
    static bool defaultComponentLogPredicate(const cComponent *object, LogLevel logLevel, c
};

// Creates a log proxy object that captures the provided context.
// This macro is internal to the logging infrastructure.
//
// NOTE: the (void)0 trick prevents GCC producing statement has no effect warnings
// for compile time disabled log statements.
//
#define OPP_LOGPROXY(object, logLevel, category) \
    ((void)0, !(COMPILETIME_LOG_PREDICATE(object, logLevel, category) && \
    omnetpp::cLog::runtimeLogPredicate(object, logLevel, category))) ? \
    omnetpp::cLogProxy::dummyStream : omnetpp::cLogProxy(object, logLevel, category, __FIL


// Returns nullptr. Helper function for the logging macros.
inline void *getThisPtr() {return nullptr;}

/**
 * @brief Use this macro when logging from static member functions.
 *
 * Background: EV_LOG and derived macros (EV_INFO, EV_DETAIL, etc) will fail
 * to compile when placed into static member functions of cObject-derived classes
 * ("cannot call member function 'cObject::getThisPtr()' without object" in GNU C++,
 * and "C2352: illegal call of non-static member function" in Visual C++).
 * To fix it, add this macro at the top of the function; it contains local declarations
 * to make the code compile.
 *
 * @ingroup Logging
 * @hideinitializer
 */
#define EV_STATICCONTEXT  void *(*getThisPtr)() = omnetpp::getThisPtr;

/**
 * @brief This is the macro underlying EV_INFO, EV_DETAIL, EV_INFO_C, and
 * similar log macros.
 *
 * This macro should not be used directly, but via the logging macros
 * EV, EV_FATAL, EV_ERROR, EV_WARN, EV_INFO, EV_DETAIL, EV_DEBUG, EV_TRACE,
 * and their "category" versions EV_C, EV_FATAL_C, EV_ERROR_C, EV_WARN_C,
 * EV_INFO_C, EV_DETAIL_C, EV_DEBUG_C, EV_TRACE_C.
 *
 * Those macros act as C++ streams: one can write on them using the
 * left-shift (<<) operator. Their names refer to the log level they
 * represent (see LogLevel). The "category" (_C) versions accept a category
 * string. Each category acts like a separate log channel; for example,
 * one can use the "test" category to log text intended for consumption
 * by an automated test suite.
 *
 * Log statements are wrapped with compile-time and runtime guards at the
 * call site to efficiently prevent unnecessary computation of parameters
 * and log content. Compile-time guards are COMPILETIME_LOGLEVEL and
 * COMPILETIME_LOG_PREDICATE. Runtime guards (runtime log level) can be
 * set up via omnetpp.ini.
 *
 * Under certain circumstances, compiling log statements may result in errors.
 * When that happens, it is possible that the EV_STATICCONTEXT macro needs to
 * be added to the code; please review its documentation for more info.
 *
 * Examples:
 *
 * \code
 * EV_INFO << "Connection setup complete" << endl;
 * EV_INFO_C("test") << "ESTAB" << endl;
 * \endcode
 *
 * @see LogLevel, EV_STATICCONTEXT, EV_INFO, EV_INFO_C, COMPILETIME_LOGLEVEL, COMPILETIME_
 * @ingroup Logging
 * @hideinitializer
 */
#define EV_LOG(logLevel, category) OPP_LOGPROXY(getThisPtr(), logLevel, category).getStrea

/**
 * @brief Short for EV_INFO.
 * @see EV_INFO @ingroup Logging
 */
#define EV        EV_INFO

/**
 * @brief Pseudo-stream for logging local fatal errors. See EV_LOG for details.
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_FATAL  EV_LOG(omnetpp::LOGLEVEL_FATAL, nullptr)

/**
 * @brief Pseudo-stream for logging local recoverable errors. See EV_LOG for details.
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_ERROR  EV_LOG(omnetpp::LOGLEVEL_ERROR, nullptr)

/**
 * @brief Pseudo-stream for logging warnings. See EV_LOG for details.
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_WARN   EV_LOG(omnetpp::LOGLEVEL_WARN, nullptr)

/**
 * @brief Pseudo-stream for logging information with the default log level. See EV_LOG for
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_INFO   EV_LOG(omnetpp::LOGLEVEL_INFO, nullptr)

/**
 * @brief Pseudo-stream for logging low-level protocol-specific details. See EV_LOG for de
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_DETAIL EV_LOG(omnetpp::LOGLEVEL_DETAIL, nullptr)

/**
 * @brief Pseudo-stream for logging state variables and other low-level information. See E
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_DEBUG  EV_LOG(omnetpp::LOGLEVEL_DEBUG, nullptr)

/**
 * @brief Pseudo-stream for logging control flow information (entering/exiting functions,
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_TRACE  EV_LOG(omnetpp::LOGLEVEL_TRACE, nullptr)

/**
 * @brief Short for EV_INFO_C.
 * @see EV_INFO_C @ingroup Logging
 */
#define EV_C(category)        EV_INFO_C(category)

/**
 * @brief Pseudo-stream for logging local fatal errors of a specific category. See EV_LOG
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_FATAL_C(category)  EV_LOG(omnetpp::LOGLEVEL_FATAL, category)

/**
 * @brief Pseudo-stream for logging local recoverable errors of a specific category. See E
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_ERROR_C(category)  EV_LOG(omnetpp::LOGLEVEL_ERROR, category)

/**
 * @brief Pseudo-stream for logging warnings of a specific category. See EV_LOG for detail
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_WARN_C(category)   EV_LOG(omnetpp::LOGLEVEL_WARN, category)

/**
 * @brief Pseudo-stream for logging information with the default log level of a specific c
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_INFO_C(category)   EV_LOG(omnetpp::LOGLEVEL_INFO, category)

/**
 * @brief Pseudo-stream for logging low-level protocol-specific details of a specific cate
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_DETAIL_C(category) EV_LOG(omnetpp::LOGLEVEL_DETAIL, category)
```

```
/**
 * @brief Pseudo-stream for logging state variables and other low-level information of a s
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_DEBUG_C(category)  EV_LOG(omnetpp::LOGLEVEL_DEBUG, category)

/**
 * @brief Pseudo-stream for logging control flow information (entering/exiting functions,
 * @see EV_LOG @hideinitializer @ingroup Logging
 */
#define EV_TRACE_C(category)  EV_LOG(omnetpp::LOGLEVEL_TRACE, category)

/**
 * @brief This class holds various data that is captured when a particular
 * log statement executes. It also contains the text written to the log stream.
 *
 * @see cEnvir::log(cLogEntry*)
 * @ingroup Internals
 */
class SIM_API cLogEntry
{
  public:
    // log statement related
    LogLevel logLevel;
    const char *category;

    // C++ source related (where the log statement appears)
    const void *sourcePointer;
    const cObject *sourceObject;
    const cComponent *sourceComponent;
    const char *sourceFile;
    int sourceLine;
    const char *sourceFunction;

    // operating system related
    clock_t userTime;

    // the actual text of the log statement
    const char *text;
    int textLength;
};


//
// This class captures the context where the log statement appears.
// NOTE: This class is internal to the logging infrastructure.
//
class SIM_API cLogProxy
{
  private:
    // This class is used for buffering the text content to be able to send whole
    // lines one by one to the active environment.
    class LogBuffer : public std::basic_stringbuf<char> {
      public:
        LogBuffer() { }
        bool isEmpty() { return pptr() == pbase(); }
      protected:
        virtual int sync() override;  // invokes getEnvir()->log() for each log line
    };

    // act likes /dev/null
    class nullstream : public std::ostream {
      public:
        nullstream() : std::ostream(nullptr) {}  // results in rdbuf==0 and badbit==true
    };

  public:
    static nullstream dummyStream; // EV evaluates to this when in express mode (getEnvir(

  private:
    static LogBuffer buffer;  // underlying buffer that contains the text that has been wr
    static std::ostream stream;  // this singleton is used to avoid allocating a new strea
    static cLogEntry currentEntry; // context of the current (last) log statement that has
    static LogLevel previousLogLevel; // log level of the previous log statement
    static const char *previousCategory; // category of the previous log statement

  private:
    void fillEntry(LogLevel logLevel, const char *category, const char *sourceFile, int so

  public:
    cLogProxy(const void *sourcePointer, LogLevel logLevel, const char *category, const cha
    cLogProxy(const cObject *sourceObject, LogLevel logLevel, const char *category, const c
    cLogProxy(const cComponent *sourceComponent, LogLevel logLevel, const char *category, c
    ~cLogProxy();

    std::ostream& getStream() { return stream; }
    static void flushLastLine();
};

}  // namespace omnetpp
```

```
#endif
```

4.3 虚函数

4.3.1 initialize 函数

4.3.1 handleMessage 函数

4.3.1 refreshDisplay 函数

4.3.1 finish 函数