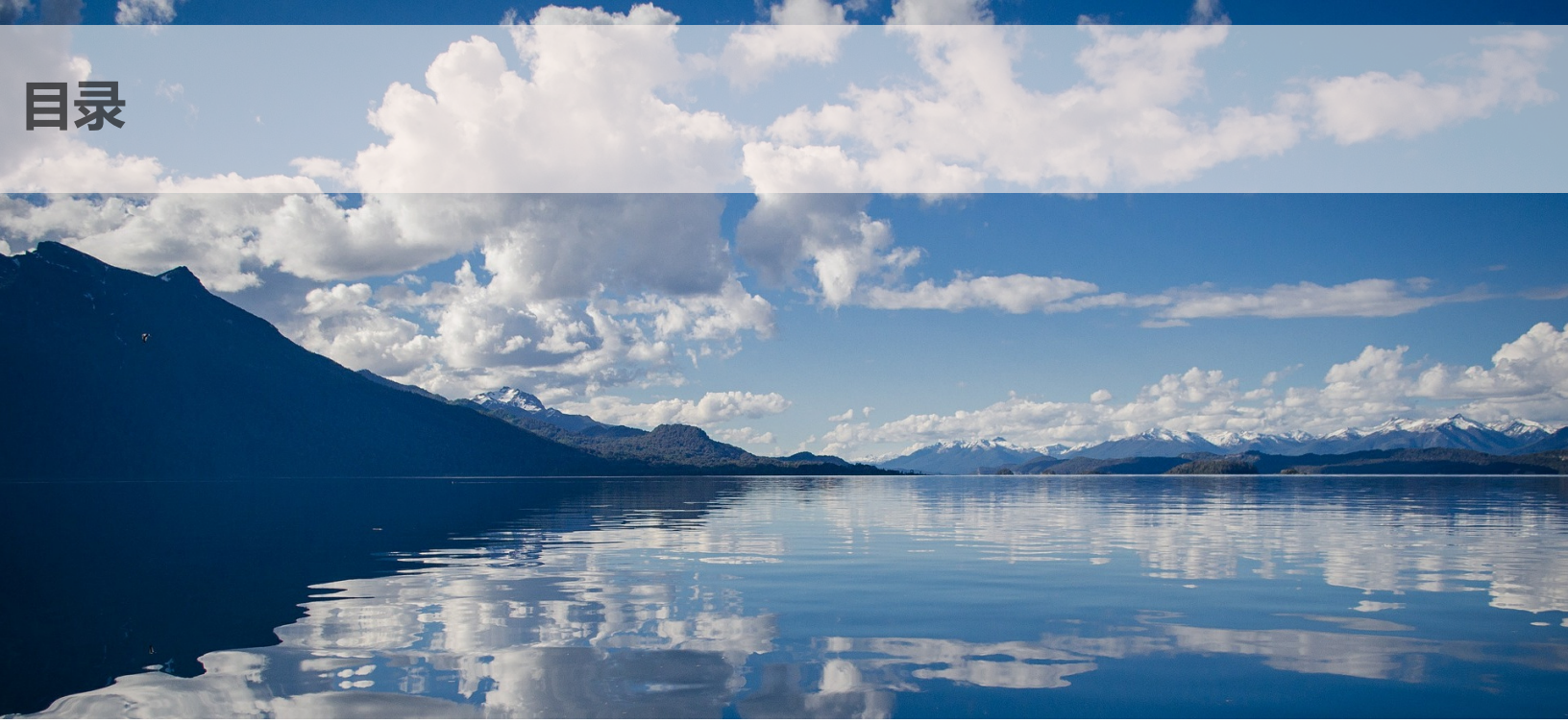
OMNeT++ 学习笔记

该文档记录了作者从在设计一个无人机蜂群网络的时候，从初学 OMNeT++ 软件开始遇到的各种问题，苦于当初无处找到答案，只能上 google-group 提问题，阅读大量的网络仿真程序，慢慢的才对这个软件的各种接口使用和设置才熟悉，特此，在该文档下记录各种 OMNeT++ 的操作，来减少读者的开发和网络仿真的烦恼。

Cover design by 花满心时亦满楼 lollipop



目录

第一章 OMNeT++ 7

1.1 OMNeT++ **简介** 7

1.2 OMNeT++ **开源库** 8

1.3 **目录** 9

第二章 OMNeT++ **安装** 10

2.1 OMNeT++ **安装** 10

2.1.1 **安装准备** 10

2.1.2 **图文并茂** 11

2.2 INET **库** 11

2.2.1 INET **库的介绍** 11

2.2.2 INET **库的安装** 12

2.3 **常规操作** 13

2.3.1 **导入工程** 13

2.3.2 **程序执行与调试** 13

第三章 学习 OMNeT++Map 19
3.1 学习 map 19
3.1.1 OMNeT++ 文档与指导书 19
3.1.2 tictoc 指导手册 20
3.1.3 仿真手册 21
3.2 个性化 IDE 21
3.2.1 CPP 高亮设置 21
第四章 OMNeT++ 编程接口 23
4.1 循规蹈矩 23
4.2 类说明 23
4.2.1 cModule 23
4.2.2 cPar 33
4.2.3 cGate 33
4.2.4 cTopology 33
4.2.5 cExpression 33
第五章 OMNeT++ 设计经验 34
5.1 经验之谈 34
5.1.1 技巧一：信道模型很重要 34
5.1.2 技巧二：send 函数有套路 35
5.1.3 技巧三：如何访问同一级其他模块 36
5.1.4 技巧四：遍历所有模块 36
5.1.5 技巧五：如何得到摸一个模块引用的 ned 路径 36
5.1.6 技巧六：使用 cTopology 类遍历拓扑初始化路由表 36
5.1.7 技巧七：如何使用 OpenSceneGraph 36
5.1.8 技巧八：如何多次利用同一个 msg 36
5.1.9 技巧九：initialize 函数的不同 36
5.1.10 技巧十：如何从仿真场景读取节点坐标 36
5.1.11 技巧十一：如何调用 INET 中的类 36

5.2 **设计技巧** 36

Literature 37

1. OMNeT++

1.1 OMNeT++ 简介 7

1.2 OMNeT++ 开源库 8

1.3 目录 9

OMNeT++ 是一个网络仿真工具，支持以太网、无线网等协议仿真，同时提供友好的仿真界面以及 3D 显示。

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：

- OMNeT++ 的安装
- INET 库的安装 INET 库的基本使用
- OMNeT++ 个性化设置
- OMNeT++ 工程设计技巧
- cModule | cPar | cGate | cTopology 相关类使用
- 仿真结果分析
- 仿真错误记录

2. OMNeT++ 安装

2.1 OMNeT++ 安装 10
2.1.1 安装准备 10
2.1.2 图文并茂 11
2.2 INET 库 11
2.2.1 INET 库的介绍 11
2.2.2 INET 库的安装 12
2.3 常规操作 13
2.3.1 导入工程 13
2.3.2 程序执行与调试 13

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。

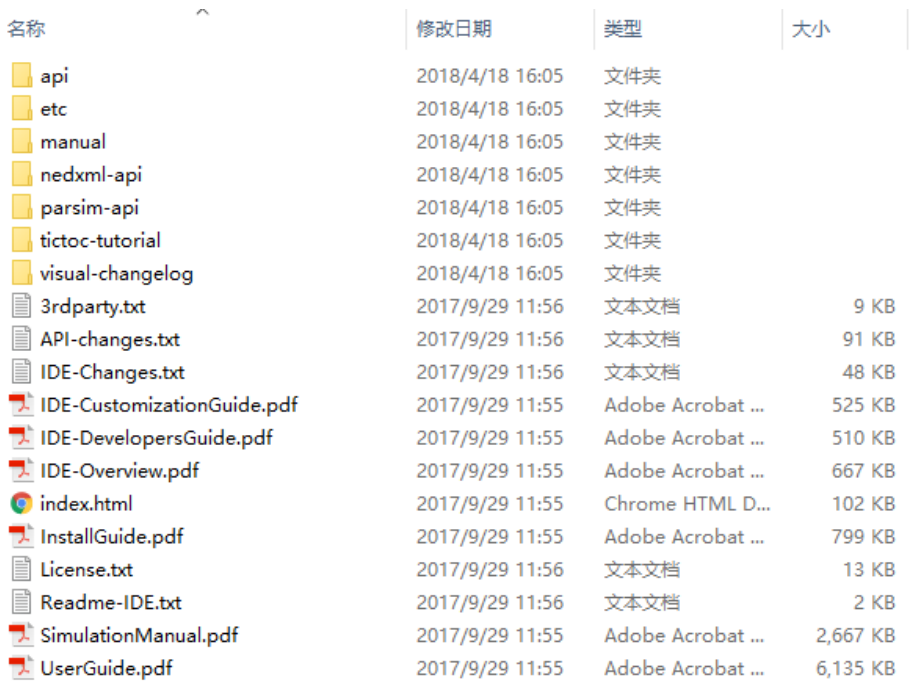
2.1 OMNeT++ 安装

2.1.1 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：

- ./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
- 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
- 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）。



名称	修改日期	类型	大小
api	2018/4/18 16:05	文件夹	
etc	2018/4/18 16:05	文件夹	
manual	2018/4/18 16:05	文件夹	
nedxml-api	2018/4/18 16:05	文件夹	
parsim-api	2018/4/18 16:05	文件夹	
tictoc-tutorial	2018/4/18 16:05	文件夹	
visual-changelog	2018/4/18 16:05	文件夹	
3rdparty.txt	2017/9/29 11:56	文本文档	9 KB
API-changes.txt	2017/9/29 11:56	文本文档	91 KB
IDE-Changes.txt	2017/9/29 11:56	文本文档	48 KB
IDE-CustomizationGuide.pdf	2017/9/29 11:55	Adobe Acrobat ...	525 KB
IDE-DevelopersGuide.pdf	2017/9/29 11:55	Adobe Acrobat ...	510 KB
IDE-Overview.pdf	2017/9/29 11:55	Adobe Acrobat ...	667 KB
index.html	2017/9/29 11:55	Chrome HTML D...	102 KB
InstallGuide.pdf	2017/9/29 11:55	Adobe Acrobat ...	799 KB
License.txt	2017/9/29 11:56	文本文档	13 KB
Readme-IDE.txt	2017/9/29 11:56	文本文档	2 KB
SimulationManual.pdf	2017/9/29 11:55	Adobe Acrobat ...	2,667 KB
UserGuide.pdf	2017/9/29 11:55	Adobe Acrobat ...	6,135 KB

图 2.1: doc 目录

2.1.2　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：

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。

2.2　INET 库

2.2.1　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，我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。

2.2.2 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：

- 命令窗口安装 INET
  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。

  > Definition 2.1 If you are using the IDE:
  > 3. Open the OMNeT++ IDE and choose the workspace where you have extracted the inet directory.
  > 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 view, find the corresponding omnetpp.ini file. Right click on it and select Run As / Simulation. 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”.

- 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。

> Definition 2.2 If you are using the IDE:
> 3. Open the OMNeT++ IDE and choose the workspace where you have extracted the inet directory. 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 view, find the corresponding omnetpp.ini file. Right click on it and select Run As / Simulation. 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。

2.3 常规操作

2.3.1 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：

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。

2.3.2 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

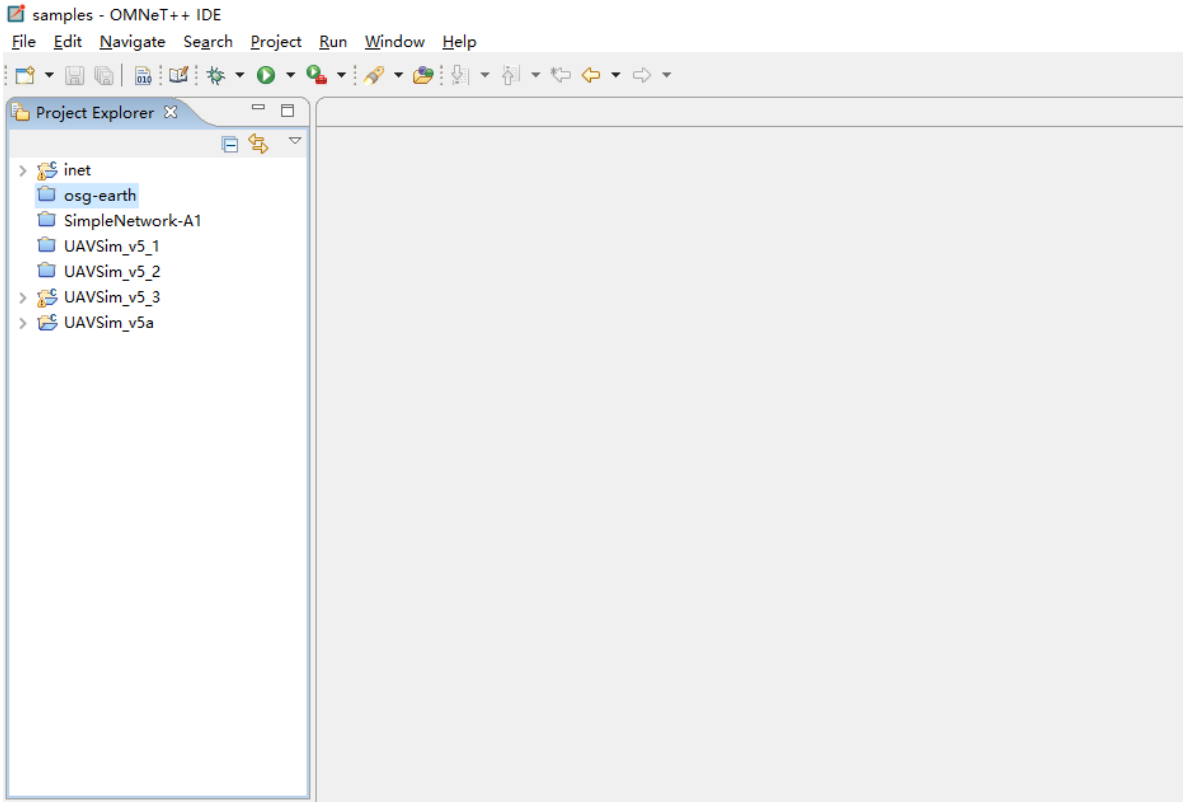

samples - OMNeT++ IDE
File Edit Navigate Search Project Run Window Help
Project Explorer
inet
osg-earth
SimpleNetwork-A1
UAVSim_v5_1
UAVSim_v5_2
UAVSim_v5_3
UAVSim_v5a

图 2.2: IDE 视图

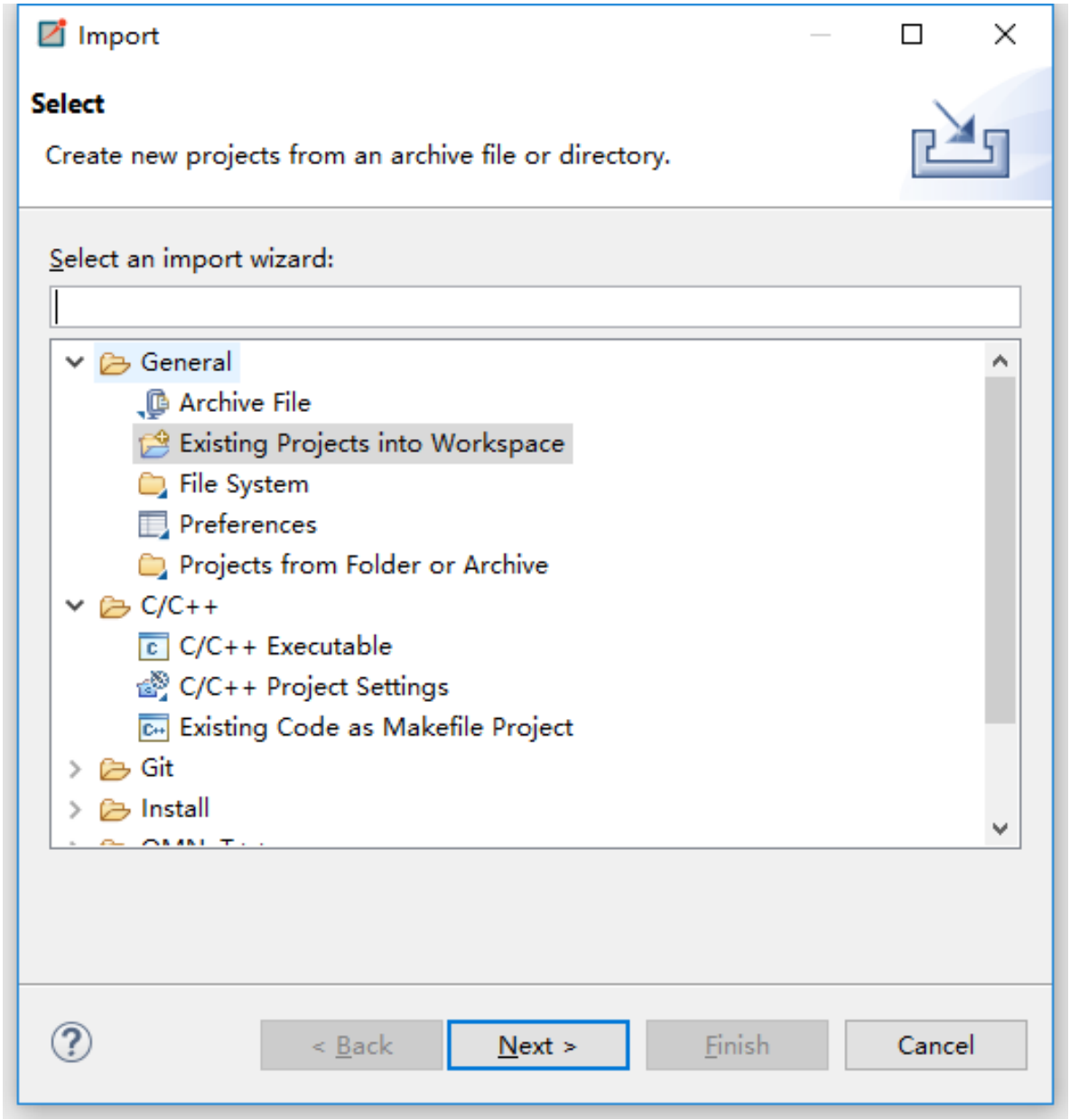


图 2.3: 点击 Import 后视图

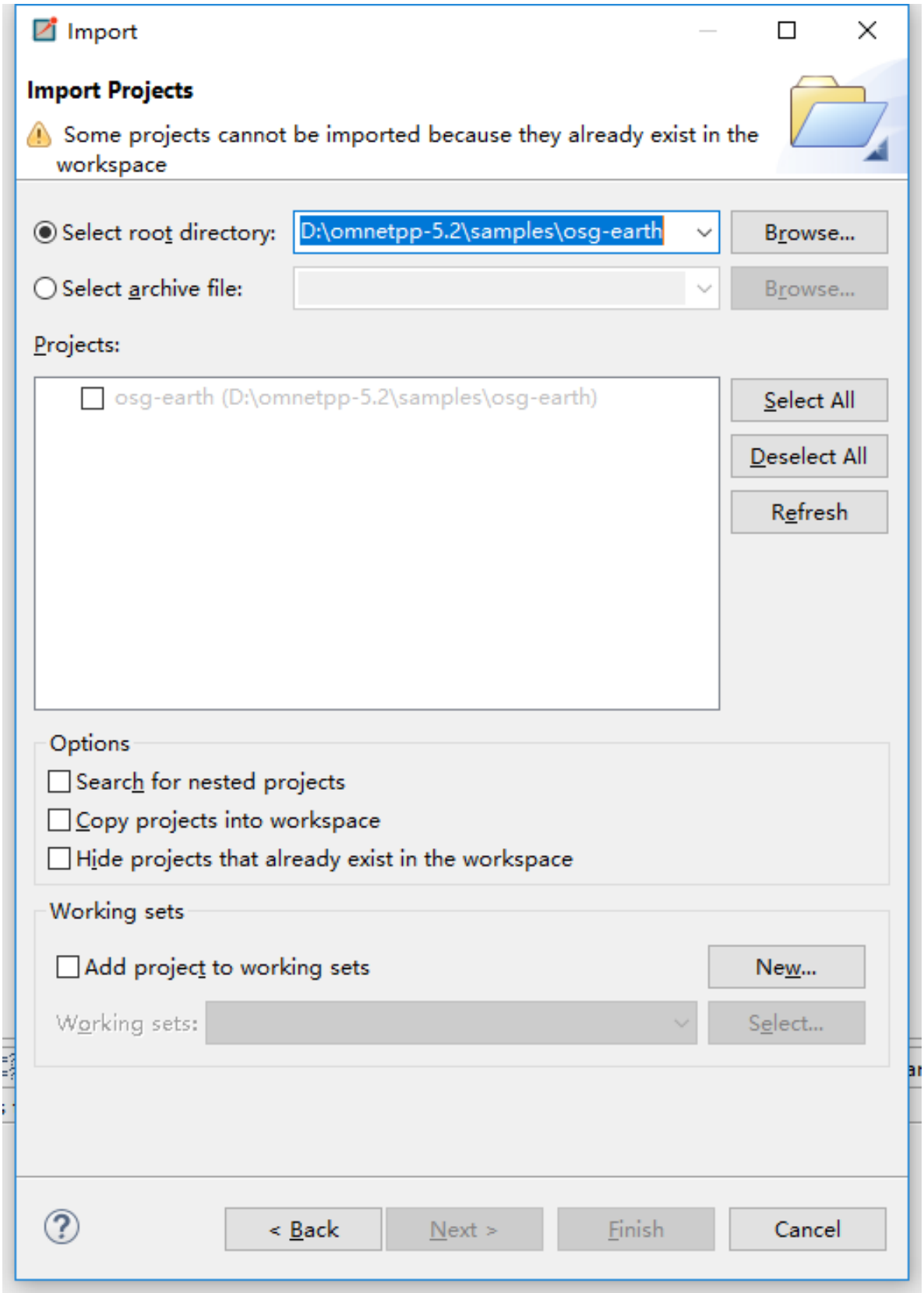


图 2.4: 点击 Existing Projects into Workspace 后视图

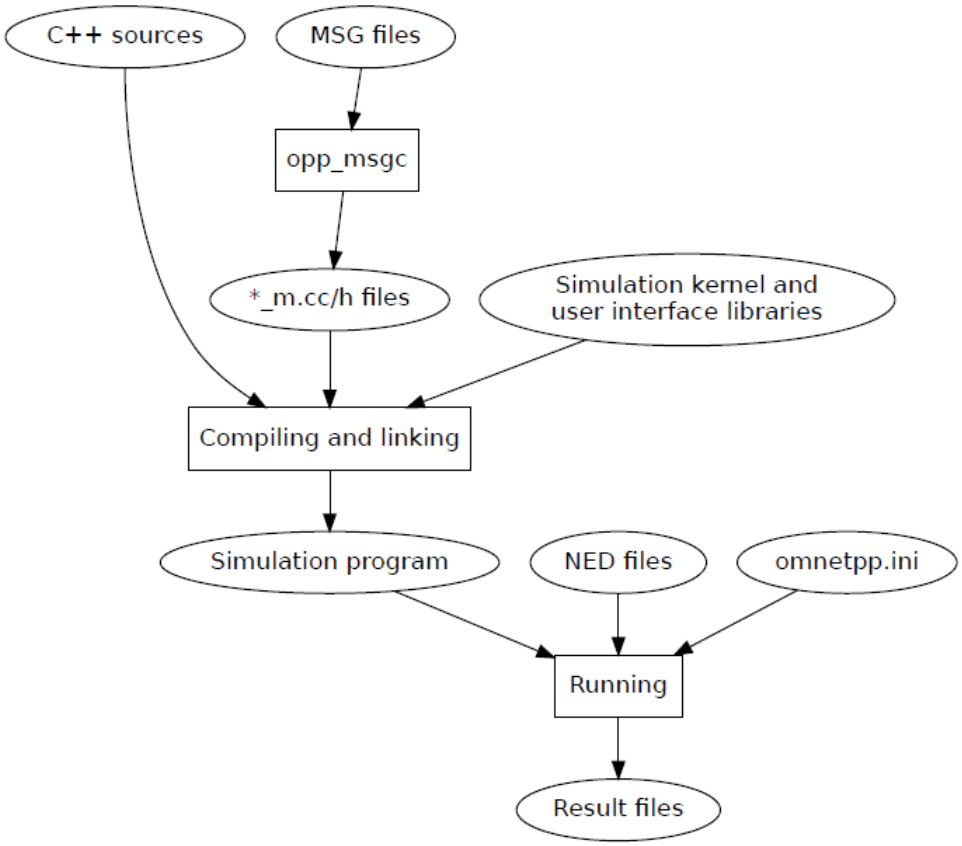


图 2.5: 编译与执行仿真流程

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 oppmsgc 转化成 *m.cc 或者 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：

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：

- 直接右击工程文件->**RUN as**->**Local C/C++ Application** 或者 **OMNeT++ Simulation**；
- 选中描述网络的 **ned** 文件，右击执行上面一样的操作；
- 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. 学习 OMNeT++Map

3.1 学习 map 19

3.1.1 OMNeT++ 文档与指导书 19

3.1.2 tictoc 指导手册 20

3.1.3 仿真手册 21

3.2 个性化 IDE 21

3.2.1 CPP 高亮设置 21

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。

3.1 学习 map

就目前学习 OMNeT++ 的资料来说，网上的资料有：

- 《OMNeT++ 中文使用手册》
- 《OMNeT++ 与网络仿真》
- 《OMNeT++ 网络仿真》

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。

3.1.1 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。

3.1.2 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。

3.1.3　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
  SimulationManual 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。

3.2　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。

3.2.1　CPP 高亮设置

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

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。

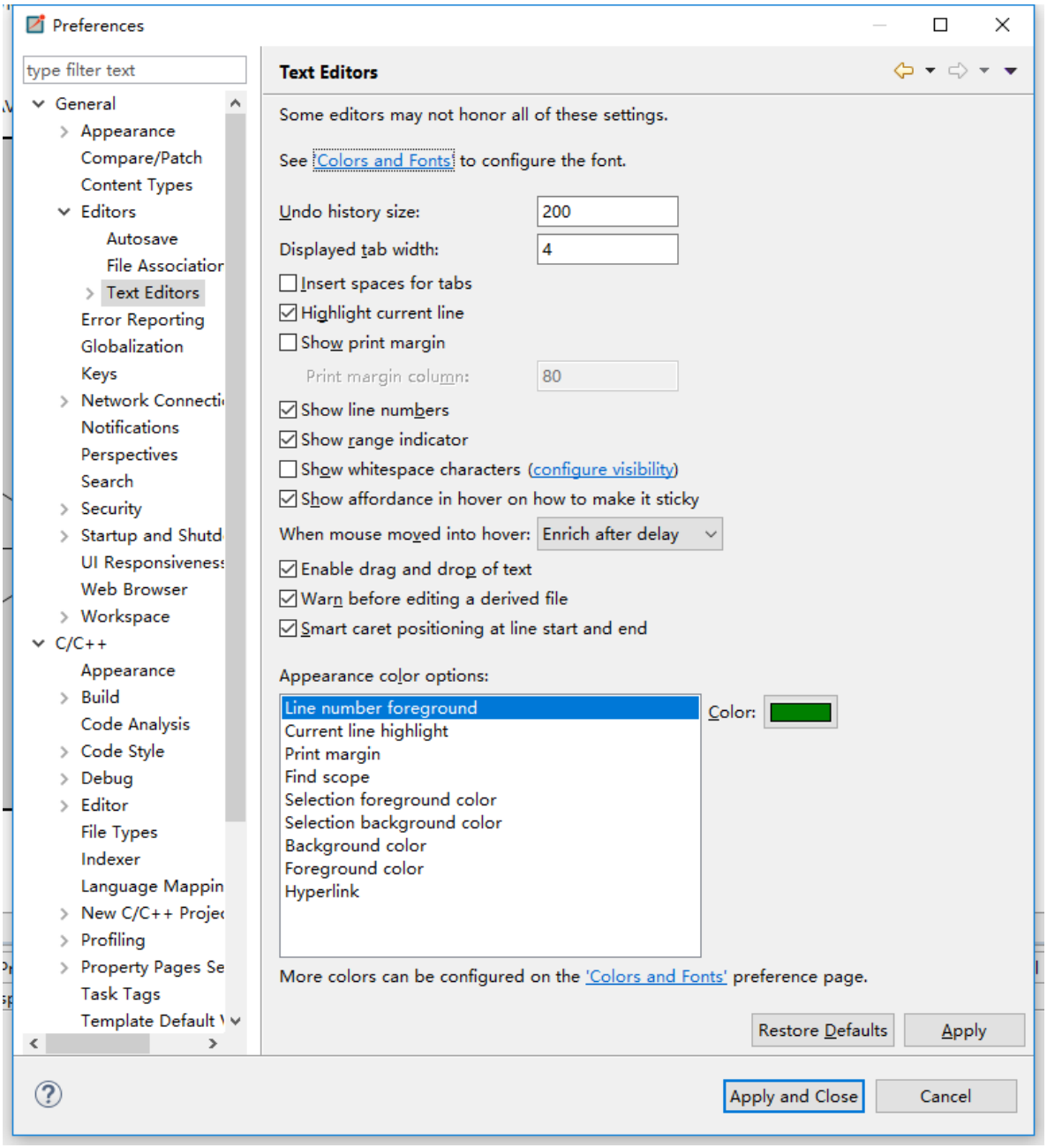

Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
Autosave
File Association
Text Editors
Error Reporting
Globalization
Keys
Network Connecti
Notifications
Perspectives
Search
Security
Startup and Shutd
UI Responsiveness
Web Browser
Workspace
C/C++
Appearance
Build
Code Analysis
Code Style
Debug
Editor
File Types
Indexer
Language Mappin
New C/C++ Proje
Profiling
Property Pages Se
Task Tags
Template Default
Text Editors
Some editors may not honor all of these settings.
See Colors and Fonts to configure the font.
Undo history size: 200
Displayed tab width: 4
Insert spaces for tabs
Highlight current line
Show print margin
Print margin column: 80
Show line numbers
Show range indicator
Show whitespace characters (configure visibility)
Show affordance in hover on how to make it sticky
When mouse moved into hover: Enrich after delay
Enable drag and drop of text
Warn before editing a derived file
Smart caret positioning at line start and end
Appearance color options:
Line number foreground
Current line highlight
Print margin
Find scope
Selection foreground color
Selection background color
Background color
Foreground color
Hyperlink
Color:
More colors can be configured on the 'Colors and Fonts' preference page.
Restore Defaults
Apply
Apply and Close
Cancel

图 3.1: 其他设置界面

4. OMNeT++ 编程接口

4.1 循规蹈矩 23

4.2 类说明 23

4.2.1 cModule 23

4.2.2 cPar 33

4.2.3 cGate 33

4.2.4 cTopology 33

4.2.5 cExpression 33

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。

4.1 循规蹈矩

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。

```
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-de
cCanvas *getCanvasIfExists() {return canvas;}

// internal: return the 3D canvas if exists, or nullptr if not (i.e. no create-on
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 cSimulatio
* 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=fals

/**
* 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_ANIMA

/**
* 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_ANIMA
//@}

/** @name Public methods for invoking initialize()/finish(), redefined from cCom
* 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
virtual void changeParentTo(cModule *mod);
};
```

4.2.2 cPar

4.2.3 cGate

4.2.4 cTopology

4.2.5 cExpression

5. OMNeT++ 设计经验

5.1 经验之谈 34

5.1.1 技巧一：信道模型很重要 34

5.1.2 技巧二：send 函数有套路 35

5.1.3 技巧三：如何访问同一级其他模块 36

5.1.4 技巧四：遍历所有模块 36

5.1.5 技巧五：如何得到摸一个模块引用的 ned 路径 36

5.1.6 技巧六：使用 cTopology 类遍历拓扑初始化路由表 36

5.1.7 技巧七：如何使用 OpenSceneGraph 36

5.1.8 技巧八：如何多次利用同一个 msg 36

5.1.9 技巧九：画圆的不同 36

5.1.10 技巧十：如何从仿真场景读取节点坐标 36

5.1.11 技巧十一：如何调用 INET 中的类 36

5.2 设计技巧 36

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。

5.1 经验之谈

5.1.1 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。

5.1.2 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
                        //调度一个事件，发送消息给自己
                        scheduleAt(simTime()+0.01,cloudMsg);
                }
        }
}
```

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：

5.1.3 技巧三：如何访问同一级其他模块

5.1.4 技巧四：遍历所有模块

5.1.5 技巧五：如何得到摸一个模块引用的 ned 路径

5.1.6 技巧六：使用 cTopology 类遍历拓扑初始化路由表

5.1.7 技巧七：如何使用 OpenSceneGraph

5.1.8 技巧八：如何多次利用同一个 msg

5.1.9 技巧九：initialize 函数的不同

5.1.10 技巧十：如何从仿真场景读取节点坐标

5.1.11 技巧十一：如何调用 INET 中的类

5.2 设计技巧