

# DNet 分布式网络软件 用户开发使用手册

版本:v1.1

北京执竹科技有限公司  
<http://www.zhulinspace.com>

2025 年 5 月

# 目录

|       |                         |    |
|-------|-------------------------|----|
| 1     | 产品介绍 .....              | 1  |
| 1.1   | 产品概述 .....              | 1  |
| 1.2   | 产品适用群体 .....            | 2  |
| 1.3   | 软件开发和使用环境 .....         | 2  |
| 1.3.1 | Windows .....           | 2  |
| 1.3.2 | Linux .....             | 2  |
| 1.3.3 | 网络硬件环境 .....            | 3  |
| 1.4   | 软件授权 license 技术支持 ..... | 4  |
| 2     | 软件安装和配置 .....           | 7  |
| 2.1   | 安装步骤 .....              | 7  |
| 2.1.1 | Windows .....           | 7  |
| 2.1.2 | Linux .....             | 9  |
| 2.2   | 卸载说明 .....              | 9  |
| 2.2.1 | Windows .....           | 9  |
| 2.2.2 | Linux .....             | 10 |
| 3     | 软件功能使用介绍 .....          | 11 |
| 3.1   | 软件功能说明 .....            | 11 |
| 3.2   | 软件安装目录 .....            | 14 |
| 3.3   | DNetApp 服务器使用说明 .....   | 15 |
| 3.3.1 | DNetApp 启动服务器 .....     | 16 |

|         |                        |    |
|---------|------------------------|----|
| 3.3.2   | DNetApp 配置项.....       | 17 |
| 3.3.3   | 记录回放 .....             | 18 |
| 3.4     | 客户端间交互模式 .....         | 21 |
| 3.5     | 网络文件 .....             | 24 |
| 3.5.1   | 网络文件设计原理 .....         | 24 |
| 3.5.2   | 网络文件配置说明 .....         | 27 |
| 3.5.2.1 | 网络文件基本结构 .....         | 27 |
| 3.5.2.2 | 数据类型定义 .....           | 28 |
| 3.5.2.3 | 网络对象定义 .....           | 30 |
| 3.5.2.4 | 网络请求消息定义 .....         | 32 |
| 3.5.3   | 示例文件说明 .....           | 35 |
| 3.6     | DNetAppTool 使用说明 ..... | 38 |
| 3.6.1   | 新建网络文件 .....           | 39 |
| 3.6.2   | 加载网络文件 .....           | 39 |
| 3.6.3   | 编辑网络文件 .....           | 40 |
| 3.6.4   | 保存网络文件 .....           | 42 |
| 3.6.5   | 另存为网络文件 .....          | 42 |
| 3.6.6   | 全部保存 .....             | 42 |
| 3.6.7   | 指定网络对象代码生成 .....       | 42 |
| 3.6.8   | 所有网络对象代码生成 .....       | 43 |
| 3.6.9   | 指定网络请求代码生成 .....       | 43 |
| 3.6.10  | 所有网络请求代码生成 .....       | 44 |

|         |                    |    |
|---------|--------------------|----|
| 3.6.11  | 网络文件代码生成 .....     | 44 |
| 3.6.12  | 检查网络文件 .....       | 44 |
| 4       | DNet 开发库开发说明 ..... | 45 |
| 4.1     | 客户端开发说明 .....      | 45 |
| 4.1.1   | 客户端开发步骤和说明 .....   | 45 |
| 4.1.2   | 示例 .....           | 48 |
| 4.1.3   | API 接口说明文档 .....   | 48 |
| 4.1.4   | 连接模式说明 .....       | 48 |
| 4.1.4.1 | UDP 组播连接模式 .....   | 48 |
| 4.1.4.2 | TCP/IP 直接连接 .....  | 49 |
| 4.2     | 服务器端二次开发说明 .....   | 52 |
| 4.2.1   | 服务器端二次开发说明 .....   | 52 |
| 4.2.2   | API 接口说明文档 .....   | 52 |
| 5       | 常见问题说明 .....       | 53 |
| 5.1     | 常见错误 .....         | 53 |
| 5.2     | 日志文件 .....         | 55 |
| 6       | 附录 .....           | 56 |
| 6.1     | 版本历史 .....         | 56 |
| 6.2     | 版权信息 .....         | 56 |
| 6.3     | 第三方库声明 .....       | 56 |

# 1 产品介绍

## 1.1 产品概述

DNet 软件 (Distributed Network Application) 是一款专为构建分布式系统而设计的网络通信基础应用。该软件采用 C++ 语言开发, 支持 C/S (客户端/服务器) 架构的通信模式, 并确保线程安全, 为开发者提供稳定可靠的通信基础。

DNet 提供了一套完整的服务器系统应用、系统服务开发接口以及客户端服务开发接口, 能够帮助开发者快速构建分布式网络的底层模块, 为分布式网络节点的搭建提供强有力的支撑。其核心优势包括:

**高效通信:** 通过优化通信机制, 显著减少网络流量, 提升响应速度。

**敏捷开发:** 符合敏捷开发理念, 支持快速原型设计, 便于快速构建底层通信关系。

**灵活可控:** 提供完整的开发接口和模块化设计, 满足多样化的分布式系统需求。

DNet 由北京执竹科技有限公司完全自主研发, 拥有全部源代码, 确保了系统的可靠性和可控性。无论是构建复杂的分布式系统, 还是开发高性能的网络应用, DNet 都能为开发者提供强大的技术支持。

## 1.2 产品适用群体

DNet 软件属于货架式产品，使用该框架的软件产品可以直接运行其框架之上。用户可以自行二次开发自定义产品，二次开发需要一定的开发经验。

- 仿真应用人员
- 仿真系统开发人员
- 系统通信开发人员
- C++二次开发人员

## 1.3 软件开发和使用环境

### 1.3.1 Windows

Windows 支持兼容内容请参考对应版本的《发布说明》文档，里面描述了相关内容。

推荐 Windows 开发环境：

操作系统：Win7 或 Win10 64 位

开发环境：Visual Studio 2017 以上版本

Qt 版本：5.14.1

### 1.3.2 Linux

Linux 支持兼容内容请参考对应版本的《发布说明》文档，里面描述了相关内容。

推荐 Linux 开发环境：

操作系统:

银河麒麟 V10 Linux 操作系统 或 Ubuntu Linux 20.04 以上操作系统。

开发环境: VSCode 或 Vim 编辑器

银河麒麟操作系统的 GCC 版本是 9.4.0 版本。

Ubuntu20.04 操作系统的 GCC 版本是 9.3.0 版本。

Ubuntu Qt 版本: 5.12.8

麒麟 V10 Qt 版本: 5.12.12

### 1.3.3 网络硬件环境

硬件环境:

- 通用以太网卡
- 支持 UDP 组播协议的路由器(注意有些路由器或交换机不支持 UDP 组播协议)
- 支持 TCP/IP 协议的网络环境(必须)
- 网络环境不支持 UDP 组播协议的路由器,可以使用仅 TCP/IP 协议完成,UDP 组播支持自动加入网络。

## 1.4 软件授权 license 技术支持

请到竹林空间([www.zhulinspace.com](http://www.zhulinspace.com))中申请和购买 license, 每月提供 7 天免费 license 使用。具体步骤如下所示:

1. 到网站申请 license, 在以下界面中获取 license。



The screenshot shows the '您正在申请 DNet分布式网络软件 的license' (You are applying for DNet distributed network software license) page. It includes a navigation bar with links to Home, Software, Application Services, Technical Support, and About Us. The main content area displays the software information: 'DNet免费7天license' (DNet free 7-day license) and '免费License' (Free License) with benefits: 7-day duration, renewal every 30 days, and no node limits. Below this is a form to '请填写license信息' (Please fill in license information), including a field for '您的公司(选填):' (Your company, optional) and a field for '网卡MAC地址(请仔细检查, 申请之后无法修改):' (Network card MAC address, please check carefully, cannot be modified after application) with an example: 'A1-B2-C3-D4-E5-F6 (六组由0-9,A-F的字母构成)'. At the bottom, it shows the current application expiration time as '2025-04-18 11:35:58' and a green button labeled '申请并且下载license' (Apply and download license).

图 1 license 申请页面

2. 填写需要申请机器的网卡 MAC 地址, 具体查看 Window 或者 Linux 机器网卡 MAC 地址的方法, 在申请页面中可以看到, 按照示例方法, 查看申请机器的网卡 MAC 地址。
3. 申请好后, 可以会自动下载, 如因浏览器原因未自动下载, 可以自行在订单中心中再次下载。
4. 授权 License 文件下载后, 放到 DNet 软件的安装目录中, 推



荐直接放到 bin 文件夹中，Window 和 Linux 是一样的。

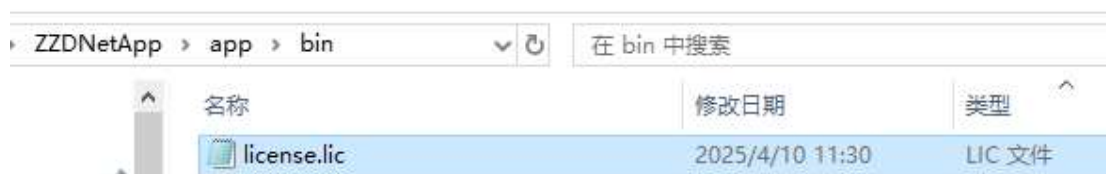


图 2 license 推荐位置

5. 运行 ZZDNetApp.exe，如未修改名称 license.lic 的文件名，无需修改，如修改了文件夹名称或在其他路径，请将点击菜单“网络”->“系统和授权配置”项，弹出对话框。



图 3 修改 license

6. 点击“浏览 License 文件”，选择 license 文件后，点击确定即可。这时 license 就配置好了。直接运行即可。



图 4 license 配置

请到北京执竹科技有限公司网站查看申请内容，如遇系统问题需要技术支持请联系我们。

邮箱：zhizhutech@126.com

或到网站留言：

网站：<http://www.zhulinspace.com>

## 2 软件安装和配置

### 2.1 安装步骤

#### 2.1.1 Windows

下载 windows 版软件后，点击 DNetAppv1.1.exe 安装文件，运行后，弹出安装应用界面。



图 5 安装界面 1

点击下一步。



图 6 安装界面 2

点击下一步。

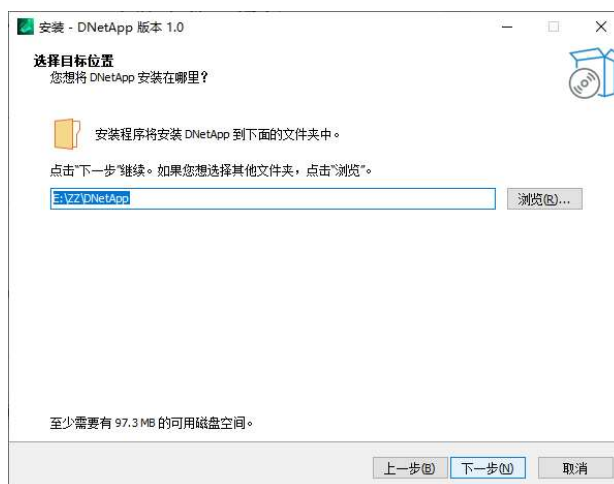


图 7 安装界面 3

选择安装文件夹，点击下一步。

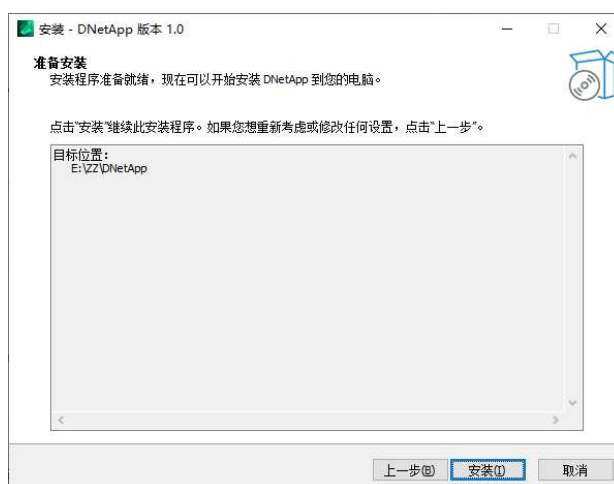


图 8 安装界面 4

直到安装界面，点击安装，直到安装完成。

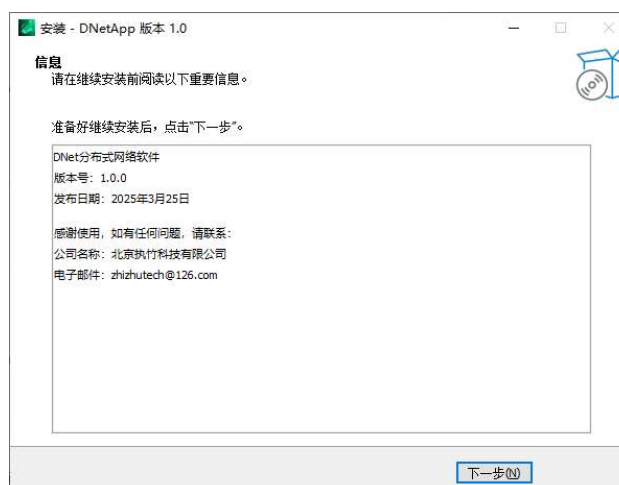


图 9 安装完成

安装完成后，运行时，碰到未找到 msvc140(xxx).dll 的情况。

可以安装 Visual Studio 2017, 或使用 vc\_redist\_x64\_vc15.exe  
(在文件夹中) 安装 Windows SDK, 保证 DNetApp 正常运行。

## 2.1.2 Linux

下载 Linux 版软件后, 将 DNetApp.tar.gz 文件解压到目标文件夹即可。

程序无法运行时, 用户需要自行在 Linux 中安装 OpenSSL 安装包。  
在 Linux 上, 通常对应的包是 libssl-dev, 在终端中执行以下命令  
安装 OpenSSL 开发库 (包含头文件和 .so 动态库):

Bash(终端)

```
sudo apt update
```

```
sudo apt install libssl-dev
```

libDNet.so 用于开发安装 openssl 后, 即可使用。

其他问题请阅读 “Linux 版本安装必读.txt” 文档。

## 2.2 卸载说明

### 2.2.1 Windows

找到安装目录, 点击 unins000.exe, 根据提示卸载文件即可。

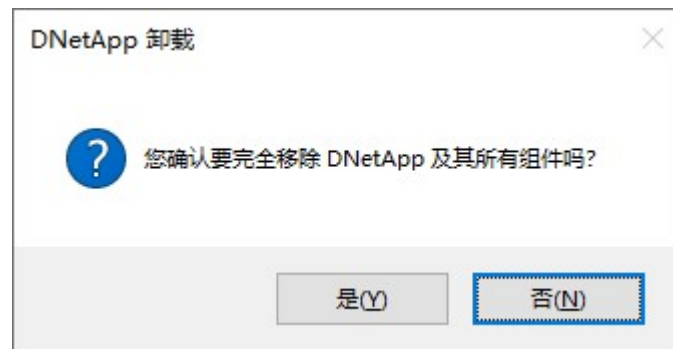


图 10 卸载软件

## 2.2.2 Linux

直接删除 DNetApp 所在文件夹即可。

## 3 软件功能使用介绍

### 3.1 软件功能说明

DNet 框架由两个核心模块组成：**ZZDNet** 和 **ZZDNetApp**。这两个模块分别负责底层通信框架的实现和上层应用的开发支持，共同构成了一个高效、灵活的分布式通信中间件。以下是对这两个模块的详细说明。

**ZZDNet** 是 DNet 框架的**底层通信实现模块**，提供了客户端和服务端的两套开发接口，用户可以根据需求选择使用。

**客户端与服务器端分离：**ZZDNet 模块分别提供了客户端和服务端的 API，开发者可以根据需要选择使用。客户端 API 用于连接到服务器并参与数据交互。服务器端 API 用于管理客户端连接、处理数据更新和分发。

**底层通信框架：**实现了基于 UDP 组播和 TCP/IP 的通信机制，支持自动组网和直接连接两种模式。提供了高效的数据传输和更新机制，如增量更新、消息广播等。

**线程安全：**通信模型设计为线程安全，支持多线程环境下的高效数据交互。

**灵活性与可扩展性：**支持复杂数据结构的定义和动态扩展，适应多样化的应用场景。

软件适用场景包括需要自定义底层通信逻辑的开发场景、对通信性能有较高要求的分布式系统、需要同时开发客户端和服务端的项目。

目。

**ZZDNetApp** 是基于 ZZDNet 模块的**上层应用开发模块**，主要面向服务器端应用的快速开发。它基于 Qt 框架，并提供了代码生成工具，帮助开发者快速构建应用程序。

**基于 Qt 的应用程序：**ZZDNetApp 模块基于 Qt 框架开发，提供了丰富的 GUI 支持和跨平台能力。开发者可以快速构建具有图形界面的服务器端应用程序。

**代码生成工具：**提供了代码生成工具，能够根据网络文件（JSON 格式）自动生成底层数据结构和通信代码。极大地减少了开发者的工作量，提升了开发效率。

**服务器端 API 封装：**对 ZZDNet 模块的服务器端 API 进行了封装，提供了更易用的接口。开发者可以专注于业务逻辑的实现，而无需过多关注底层通信细节。

**快速集成：**生成的代码可以直接集成到现有应用程序中，快速实现分布式通信功能。

DNet 框架可以作为**软件通信中间件**，广泛应用于各种分布式系统中。其核心优势包括：

**高效通信：**通过增量更新、消息广播等机制，减少了网络带宽的占用，提升了通信效率。

**灵活扩展：**支持复杂数据结构的定义和动态扩展，适应多样化的应用需求。

**跨平台支持：**基于 Qt 框架开发，支持 Windows、Linux、macOS 等



主流操作系统。

**快速开发：**提供了代码生成工具和易用的 API，显著降低了开发难度和时间成本。

在军事仿真、工业仿真等领域，DNet 可以高效地处理大量实体的状态更新和事件交互。适用于物联网、智能交通等需要实时更新和数据分发的场景。在分布式计算、协同办公等场景中，DNet 可以实现高效的节点间通信和数据共享。

DNet 框架作为通信中间件，具有高效、灵活、易用的特点，适用于各种分布式系统的开发和应用。无论是需要自定义底层通信逻辑，还是快速构建上层应用，DNet 都能提供强大的支持。

软件目前包含的结构如图 1 所示：

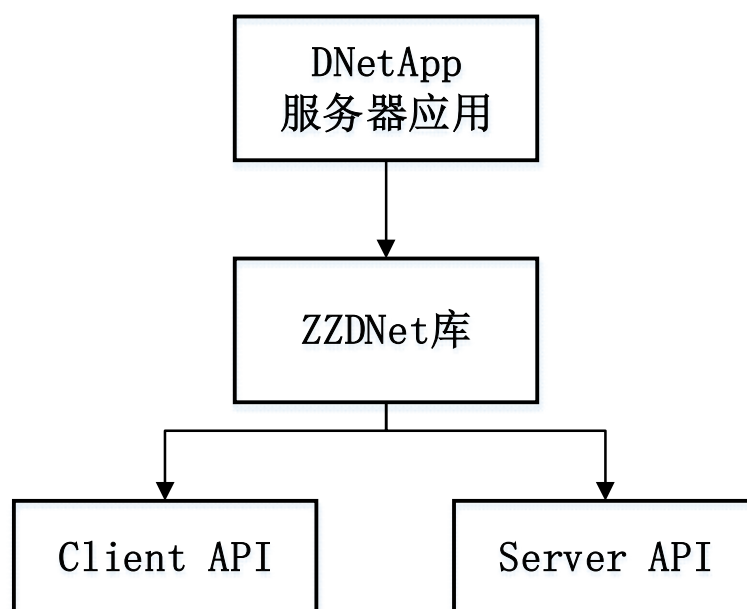


图 11 DNetApp 系统结构

## 3.2 软件安装目录

DNet 软件安装目录包含了 bin、config、example、include、lib、network 等文件夹。

文件夹 bin 是软件的可执行目录，包含了可执行文件 exe、示例应用程序 exe、动态链接库 dll。可在该目录下进行调试开发，用户开发的软件可以从该文件夹下拷贝 dll 到自定义开发的文件夹中，保证软件正常的运行。

文件夹 config 包含了 ZZDNetApp 服务器端的参数定义，可以在应用程序中的界面进行修改，也可以自行定义。

文件夹 example 包含了示例程序代码。

文件夹 include 包含了二次开发的头文件，包含了客户端开发和服务器开发。

文件夹 lib 包含了二次开发的 lib 文件，包含了客户端开发和服务器开发，具备 Debug 模式和 Release 模式两种。

文件夹 network 包含了软件自带的网络文件以及仿真引擎定义的网络文件。

|         |                 |     |
|---------|-----------------|-----|
| bin     | 2025/2/19 14:17 | 文件夹 |
| config  | 2025/2/19 14:17 | 文件夹 |
| example | 2025/2/19 14:17 | 文件夹 |
| include | 2025/2/19 14:17 | 文件夹 |
| lib     | 2025/2/24 10:54 | 文件夹 |
| network | 2025/2/19 14:17 | 文件夹 |

图 12 安装目录

### 3.3 DNetApp 服务器使用说明

在 bin 文件夹中 ZZDNetApp.exe 是 DNet 自带的服务器端应用，ZZDNetAppd.exe 是 DNet 调试版本的服务器端应用。在每次启动客户端的时候，需要首先启动该应用，界面如下所示：



图 13 DNet 服务器系统

### 3.3.1 DNetApp 启动服务器

DNetApp 界面中点击“启动服务器”，查看运行记录中的状态。



图 14 启动界面

显示加入组播地址后，并且按钮变成了“停止服务器”后，说明已经启动完成。

这时，客户端就可以连接使用了。

### 3.3.2 DNetApp 配置项

DNetApp 服务器应用的配置主要配置以下内容：

网络名称配置，可以在界面中的设置当前服务器的网络名称，默认为 MyServer，可以自行设置配置。如无需求，则可以不修改。

服务器网络选择配置，可以在界面中的网络选择页面进行配置，默认是本机 127.0.0.1。点击下拉框可以看到当前机器的 IP 地址：如图所示：



图 15 网卡选择配置

选择后，启动服务器即绑定了对应网卡通信。

端口配置，在网络端口中一项，配置服务器的通信端口，选择一个无冲突的端口即可。注意：最好配置 4000 端口以后的，防止产生冲突，无法启动服务器。系统限制端口使用 1000-65535 的端口，防

止端口冲突。端口冲突会产生 Server TCP 信息:SOCKET\_BIND\_ERROR 的错误。服务器停止的时候也会生成类似的消息，可以忽略。

点击菜单中网络->配置或界面中的配置，会弹出组播地址的配置对话框，可以配置组播 IP 地址和端口，组播 IP 地址是由 224.0.0.0 到 239.255.255.255，用户可以自行定义或者使用默认地址即可。

### 3.3.3 记录回放

DNet 服务器系统包含了记录回放系统，可以将记录时的网络数据和交互数据记录到文件中，在界面中有开始记录按钮，在启动服务器之后（服务器未启动的话，是无记录效果的），点击开始记录，记录已停止指示器会变为正在记录，这是记录系统就开始工作了。

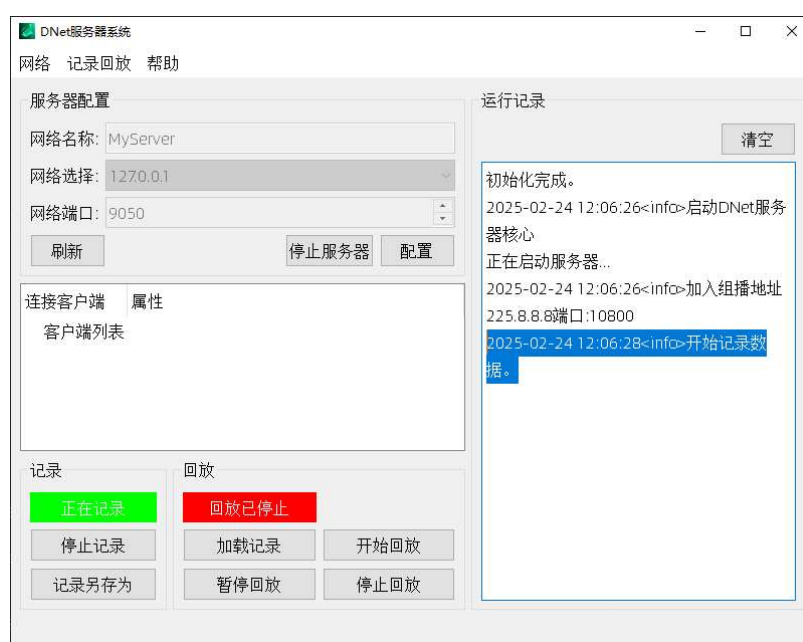


图 16 记录状态

点击停止记录，就完成了记录过程，之后可以点击“记录另存为”，这是可以将记录文件保存到其他文件。如果未保存时，记录会被覆盖，丢失原有的记录内容。文件扩展名为 zrd。

回放已记录的文件，点击加载记录后，加载记录的文件，这时会显示回放已加载，需要手动点击开始回放，来开始回放过程。



图 17 回放加载

点击“开始回放”，这时就开始回放记录。



图 18 回放中

播放过程中，可以“暂停回放”，也可以手动“停止回放”。文件播放完成后，会有提示回放文件完成，停止回放后，会显示回放已停止。



图 19 回放停止



### 3.4 客户端间交互模式

DNet 框架通过**模型定义**的方式构建网络架构，提供了灵活且高效的开发模式，相较于 HLA、DDS 等分布式仿真协议，DNet 在交互模式和开发便利性上具有显著优势。以下是对 DNet 框架的核心设计和工作原理的详细说明：

DNet 使用 **JSON 文件格式**来定义网络数据结构，称为**网络文件**。

JSON 格式具有以下优势：

- **结构化**：支持基本数据类型、数组、对象等复杂数据结构，便于扩展和维护。
- **易读性**：JSON 文件易于阅读和编辑，降低了开发者的学习成本。
- **灵活性**：可以动态调整数据结构，适应不同的仿真需求。

网络文件通常存放在安装目录下的 `network` 文件夹中，分为以下三类：

- **网络对象定义 (network-def)**：定义分布式网络中常驻的对象。
- **数据交互请求定义 (request-def)**：定义事件、数据传输、消息交互等一次性通信。
- **数据类型定义**：支持复杂数据结构的定义，增强数据交互的表达能力。

网络对象是分布式网络中的核心数据交互单元，具有以下特点：

**所有权机制**：一个节点（通常是服务器或某个客户端）负责定义

并拥有某个网络对象的所有权。该节点负责更新网络对象的数据，其他节点只能订阅该对象的数据，**无法修改**。

**数据更新机制：**当网络对象的数据段发生变化时，只有**变化的数据段**会被发送到订阅节点，而不是整个对象。这种**增量更新**机制显著减少了网络带宽的占用，提升了通信效率。

**复杂数据类型支持：**网络对象可以包含复杂的数据结构（如嵌套对象、数组等），能够满足多样化的仿真需求。

网络消息请求用于处理分布式网络中的**一次性事件**，具有以下特点：

**一对多通信：**一个节点发出消息请求，所有订阅该消息类型的节点都会收到该消息。这种模式适用于事件通知、数据传输、消息广播等场景。

**灵活性：**消息请求可以自定义，甚至可以用于文件传输等复杂任务。

**轻量级：**消息请求通常只包含必要的**数据**，减少了网络负载。

DNet 框架的通信模型是**线程安全**的，具有以下优势：

- **多线程支持：**网络对象和消息请求的创建、调用可以在任意线程中进行，无需担心线程冲突。
- **高效性：**通过优化底层通信机制，DNet 能够在多线程环境中高效处理大量数据交互。

DNet 框架在以下方面优于 HLA、DDS 等分布式仿真协议：

**开发便利性：**使用 JSON 文件定义网络数据结构，简化了开发流

程。提供了更直观的 API 和通信模型，降低了开发者的学习成本。

**通信效率：**通过增量更新机制和轻量级消息请求，减少了网络带宽的占用。

**灵活性：**支持复杂数据结构和动态调整，适应多样化的仿真需求。

在以下的典型应用场景中可以使用。分布式仿真：在军事仿真、工业仿真等领域，DNet 可以高效地处理大量实体的状态更新和事件交互。实时数据分发：适用于需要实时更新和分发的场景，如物联网、智能交通等。文件传输与消息广播：通过消息请求机制，DNet 可以完成文件传输、事件通知等任务。

DNet 框架通过**模型定义**的方式构建网络架构，提供了高效、灵活且易用的开发模式。其核心特点包括：使用 JSON 文件定义网络数据结构，便于扩展和维护。通过网络对象和消息请求实现高效的数据交互。支持复杂数据类型和增量更新机制，减少网络负载。线程安全的通信模型，适用于多线程环境。相较于 HLA、DDS 等协议，DNet 在开发便利性、通信效率和灵活性上具有显著优势，是分布式仿真和实时数据交互的理想选择。

## 3.5 网络文件

### 3.5.1 网络文件设计原理

DNet 网络文件采用 **JSON 格式**，主要是因为 JSON 具有字典（键值对 key-value 结构）和数组两种结构，且字典结构不允许重复的关键字（key），这在一定程度上避免了命名错误的问题。此外，JSON 文件解析技术成熟，文件大小相对较小，结构清晰，编码规则固定，非常适合用于网络文件的定义。

网络文件主要包含两种结构：**network-def（对象定义）**和**request-def（消息定义）**，并支持复合数据结构（data-type）的定义。

**JSON 格式的优势在于：**

**字典和数组结构：**JSON 支持字典（键值对）和数组两种数据结构，能够清晰地表达复杂的数据关系。

**唯一关键字：**字典结构不允许重复的 key，避免了命名冲突和错误。

**文件大小：**相比 XML，JSON 文件更小，解析速度更快。

**解析成熟：**JSON 解析库广泛且成熟，支持多种编程语言。

**结构清晰：**JSON 的层次结构清晰，易于阅读和修改。

网络文件的结构定义包含了 **network-def(对象定义)**和 **request-def（消息定义）**。

(1) **network-def（对象定义）**

**复合数据结构：**对象定义支持复合数据结构（data-type），适用于复杂数据的解析和内存管理。

**更新机制：**对象定义的更新是**单项更新**，即只更新变化的部分，而不是整体更新。这种机制优化了通信效率，减少了带宽占用。

**继承模式：**对象定义支持继承，可以从基本对象派生出新的对象，便于扩展。

**固定性：**对象定义通常比较固定，除非必要，否则不会频繁修改。复合数据结构的设计符合这种固定性需求。

## (2) request-def（消息定义）

**基本类型定义：**消息定义使用基本类型（如 int、float、string 等），而不是复合数据结构。

**一次性通信：**消息定义的内容是一次性的，不经过服务器存储，只通过转发。这种设计使得消息定义更加灵活。

**扩展性：**消息定义需要支持未来的自定义消息类型，使用基本类型定义可以避免因复合数据结构导致的扩展困难。

**大数据传输：**消息定义支持无固定长度的定义方式，适合传输大数据（如文件）。

**复合数据结构的使用时，针对对象定义和消息定义的不同制定以下策略。**  
**对象定义：**复合数据结构适用于对象定义，因为对象定义的更新是单项更新，且数据结构相对固定。复合数据结构可以有效地组织复杂数据，同时优化通信效率。  
**通信优化：**对象定义的更新是单项更新，复合数据结构可以减少通信数据量。  
**固定性：**对象定义的结构

相对固定，复合数据结构符合其设计需求。**继承扩展：**对象定义支持继承，复合数据结构便于扩展。

**消息定义：**消息定义不适用复合数据结构，因为消息定义需要更高的灵活性和扩展性，且复合数据结构可能导致通信带宽的浪费。**灵活性：**消息定义需要支持未来的自定义消息类型，基本类型定义更加灵活。**一次性通信：**消息定义的内容是一次性的，不需要复杂的复合数据结构。**大数据传输：**基本类型定义支持无固定长度的数据传输，适合传输文件等大数据。

为了提高开发效率和正确性，DNet 提供了**工具支持**，用于生成和验证网络文件：

**代码生成工具：**根据 JSON 文件自动生成对象定义和消息定义的代码，减少手动编写的工作量。

**验证工具：**检查 JSON 文件的语法和逻辑错误，防止因定义错误导致的 bug。

DNet 网络文件采用 JSON 格式，结合 network-def（对象定义）和 request-def（消息定义）两种结构，实现了高效、灵活的网络通信：

**对象定义：**使用复合数据结构，支持单项更新和继承扩展，适合固定且复杂的数据结构。**消息定义：**使用基本类型定义，支持灵活扩展和大数据传输，适合一次性通信场景。通过工具支持，DNet 进一步提升了开发效率和正确性，确保网络文件的定义清晰、可靠。这种设计不仅优化了通信效率，还为未来的扩展和维护提供了便利。

## 3.5.2 网络文件配置说明

网络文件使用 Json 格式定义，遵守 Json 格式规则。

### 3.5.2.1 网络文件基本结构

网络文件节结构

```
{
  "network": {
    "network-def": {
      // 定义网络对象
    },
    "request-def": {
      // 定义请求消息对象
    },
    "data-type": {
      // 定义数据类型
    }
  }
}
```

第一层定义为“network”，该层是顶层，标准定义

第二层包含了“network-def”，“request-def”和“data-type”。

network-def 中定义网络对象，request-def 中定义请求消息对象，

data-type 中定义数据类型定义，为网络对象服务。

在文件中这三个定义前后顺序无限制，哪个在前在后都可以。

### 3.5.2.2 数据类型定义

这里首先介绍数据类型定义是因为网络对象定义中需要使用，其定义的数据为定长数据，不能为变长数据，变长数据可以出现在网络请求消息中。以 test.json 为例，以下是该文件的数据类型定义：

```
"data-type": {  
  "double": [{  
    "type": "double",  
    "count": 1  
  }],  
  "base-string": [{  
    "type": "char",  
    "count": 16  
  }],  
  "zz-vector": [{  
    "type": "double",  
    "count": 3  
  }]  
}
```

可以看到 data-type 中定义了三个对象类型:double,base,zz-vector 三种定义。以 double 为例，其值定义为一个数组，这代表着可以定义多个类型定义,如下所示，定义内容用户根据内容可以酌情定义，这样可以最大程度的扩展复杂数据定义类型。

```
"double-2": [{  
  "type": "double",  
  "count": 1  
},  
{  
  "type": "int",  
  "count": 1  
}],
```

定义了 type 和 count。type 可以定义的类型如下所示（数据类型大小以 C++定义准）：



- bool (1 字节)
- char (1 字节)
- short (2 字节)
- int (4 字节)
- float (4 字节)
- double (8 字节)
- long (4 字节)
- long long (8 字节)
- long double (8 字节)

这些类型可以满足所有的类型定义，count 则是可以定义该类型的个数，比如 count:1 是定义一个 double，如 base-string 定义了一个 16 个 char 类型的结构。

如 double-2 中定义了 1 个 double，1 个 int 结构的复杂数据类型。

数据类型定义，以这种规则可完成对一个数据类型或复杂数据类型的定义。

### 3.5.2.3 网络对象定义

网络对象是常驻网络中的数据对象，公布该对象的节点不退出或删除该对象，对象数据将一致存在于网络中，该节点也具有该对象数据的管理权和所有权。网络对象包含在 network-def 层中，以下是一个示例：

```
"network-def": {  
  "net-object-state": {  
    "base": "",  
    "data": {  
      "name": {  
        "type": "base-string"  
      },  
      "id": {  
        "type": "base-string"  
      },  
      "value": {  
        "type": "double"  
      },  
      "position": {  
        "type": "zz-vector"  
      }  
    }  
  }  
},
```

首先在下一层定义使用键(key 关键字)值(value 值)对的结构，网络对象名称作为关键字，示例中是 net-object-state，这个名称与其对应的 C++类的 type 定义一致。由于 Json 规则中定义，同一层结构中不允许出现相同的键定义，所以，这里 net-object-state 在该层中是唯一的。

net-object-stat 节点值为数据定义对象。该值包含两个定义：

第一个定义 base 是继承的网络对象，如果继承，该对象首先包含

了继承的网络对象中所有的内容，包括继承的网络对象中所定义的 base 网络对象。所以网络对象是可以被继承的，可以将已有的网络对象进行重用。同时对应的 C++ 类也需要继承对应的类。

第二个定义 data 是定义当前网络对象包含的数据定义对象，同样是键值对的定义方式。同样因为键在同层中是唯一的，所以，数据名称是唯一的。示例中 data 定义了四个对象分别是 name, id, value, position, 这四个对象最好在 C++ 定义名称一致，这样可读性比较高。如果有特殊情况，也可以不一致，但是需要定义的类型大小与 C++ 类中一致，否则会出现错误。以 name 为例，其内部定义了“type”: “base-string”，由 3.5.2.2 数据类型定义中定义的数据对象类型。在数据类型定义中定义了 base-string 是一个 16 字节的字符串，如下定义：

```
"base-string": [{  
  "type": "char",  
  "count": 16  
}],
```

这样 net-object-state 中 data 的第一个 name 定义就完成了，其他的以此类推进行定义。

### 3.5.2.4 网络请求消息定义

网络请求消息是网络一次性的消息通信，可以代表事件、消息、文件传输等网络事件。只有订购对应消息的节点才可以收到对应类型的消息。还以 test.json 定义为例，网络请求消息定义如下所示：

```
"request-def": {
  "custom-user-request": {
    "BoolValue": {
      "type": "bool",
      "len": 1
    },
    "CharValue": {
      "type": "char",
      "len": 1
    },
    "IntValue": {
      "type": "int",
      "len": 4
    },
    "ShortValue": {
      "type": "short",
      "len": 2
    },
    "FloatValue": {
      "type": "float",
      "len": 4
    },
    "LongValue": {
      "type": "long",
      "len": 4
    },
    "LongLongValue": {
      "type": "long long",
      "len": 8
    },
    "DoubleValue": {
      "type": "double",
      "len": 8
    },
    "Name1": {
```

```
    "type": "string",
    "len": 16
  },
  "Name2": {
    "type": "string",
    "len": 16
  },
  "Data1": {
    "type": "data",
    "len": -1
  },
  "Data2": {
    "type": "data",
    "len": -1
  }
}
```

其第一层使用 request-def 定义,在其中定义网络请求消息对象,由示例可以看到定义了一个名为 custom-user-request 的请求消息对象,名字在该层中唯一,名称与 C++类中定义的 type 一致。

在 custom-user-request 中定义若干个数据对象,其数据对象定义以 BoolValue 为例:

```
"BoolValue": {
  "type": "bool",
  "len": 1
},
```

其内部定义的键值对有两个: type 和 len, type 是当前对象的数据类型, len 代表当前类型的长度, type 是 bool 类型, len 是 bool 数据长度为 1。这两者要对应, type 和 len 是固定的,无需修改, DNetAppTool 设置好 type 后 len 会自动配置,具体 type 和 len 对应列表如下:

| 可定义的类型 | 长度定义 |
|--------|------|
|--------|------|

|           |                         |
|-----------|-------------------------|
| bool      | 1                       |
| char      | 1                       |
| int       | 4                       |
| short     | 2                       |
| float     | 4                       |
| long      | 4                       |
| long long | 8                       |
| double    | 8                       |
| string    | char 的倍数，定长字符串          |
| data      | 定义为-1。不定长数据，可以是文本、可以是数据 |

这里固定长度的不做过多解释，data 定义为-1，代表不定长数据，无论是文本还是二进制数据都可以放到 data 中，这样可以传递不定长数据。

### 3.5.3 示例文件说明

以一个示例说明一下文件，在安装目录下 network 文件夹中有一个名为 test.json 的网络文件，内容如下：

```
{
  "network": {
    "network-def": {
      "net-object-state": {
        "base": "",
        "data": {
          "name": {
            "type": "base-string"
          },
          "id": {
            "type": "base-string"
          },
          "value": {
            "type": "double"
          },
          "position": {
            "type": "zz-vector"
          }
        }
      }
    },
    "request-def": {
      "custom-user-request": {
        "BoolValue": {
          "type": "bool",
          "len": 1
        },
        "CharValue": {
          "type": "char",
          "len": 1
        },
        "IntValue": {
          "type": "int",
          "len": 4
        },
        "ShortValue": {
          "type": "short",
```

```
        "len": 2
    },
    "FloatValue": {
        "type": "float",
        "len": 4
    },
    "LongValue": {
        "type": "long",
        "len": 4
    },
    "LongLongValue": {
        "type": "long long",
        "len": 8
    },
    "DoubleValue": {
        "type": "double",
        "len": 8
    },
    "Name1": {
        "type": "string",
        "len": 16
    },
    "Name2": {
        "type": "string",
        "len": 16
    },
    "Data1": {
        "type": "data",
        "len": -1
    },
    "Data2": {
        "type": "data",
        "len": -1
    }
}

"data-type": {
    "double": [{
        "type": "double",
        "count": 1
    }],
    "base-string": [{
        "type": "char",
        "count": 16
    }]
```



```
    }},  
    "zz-vector": [{  
        "type": "double",  
        "count": 3  
    }]  
}  
}  
}
```

上面这个文件中在最上面定义了名为 network 基础 Json 键值对象，在这个对象中定义 network-def 网络对象定义、request-def 网络消息请求定义和 data-type 数据类型定义。这个定义的先后顺序无影响。

在 network-def 里面定义了 net-object-state 的 Json 键值对象，这个就是对应 C++开发中的集成 ZZDNetObject 对象类，定义变量和变量类型和保持一致即可。这个键值对象中包含了两项：base 和 data。base 项代表继承了之前定义项，如果为空就是没有继承项。

### 3.6 DNetAppTool 使用说明

DNetAppTool 工具是用来配置 Network 网络文件的工具，用该工具可以生成网络文件，保证文件的正确性，以及将网络文件或文件中某一项生成对应的代码，以完成快捷的二次开发，减少用户工作量。

界面如下所示：

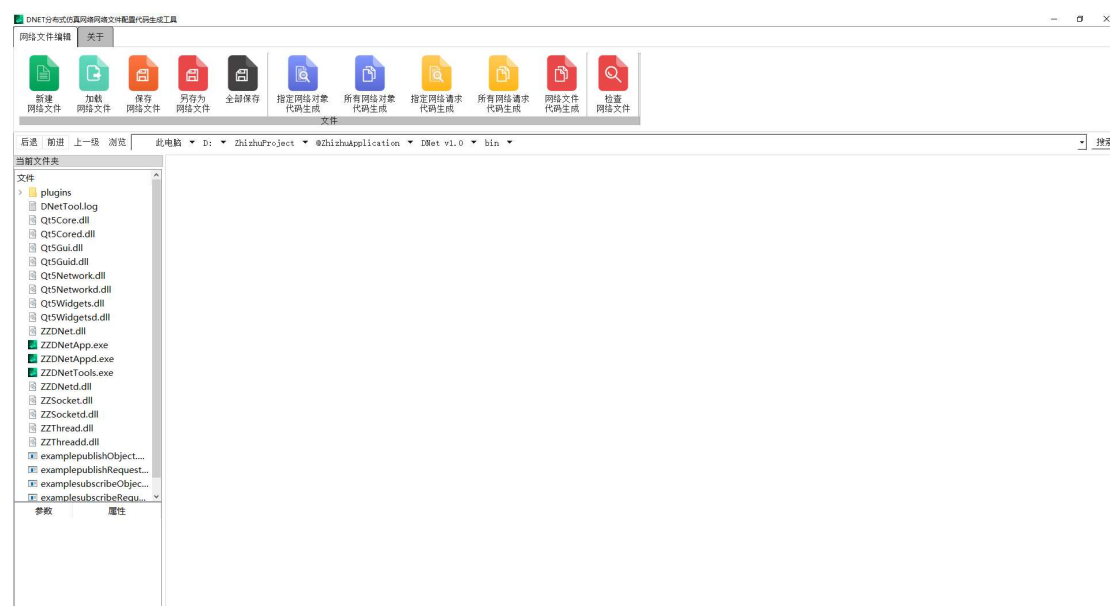


图 20 DNetAppTool 工具

界面中可以系统文件夹结构直接操作文件，界面上会比较方便。

功能：

- 生成网络文件
- 配置网络文件
- 生成网络文件网络对象或网络请求的代码
- 检查文件

### 3.6.1 新建网络文件

点击新建网络文件，这时会出现一个“未命名”的编辑界面，在这个里面编辑网络文件。

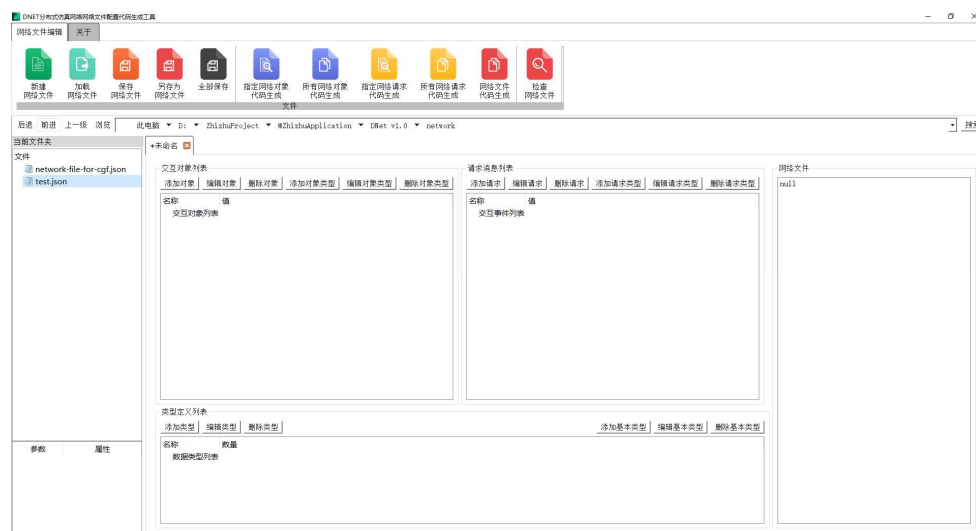


图 21 新建网络文件

### 3.6.2 加载网络文件

点击加载网络文件，在文件系统中选择路径和文件，双击就会生成编辑网络文件界面。

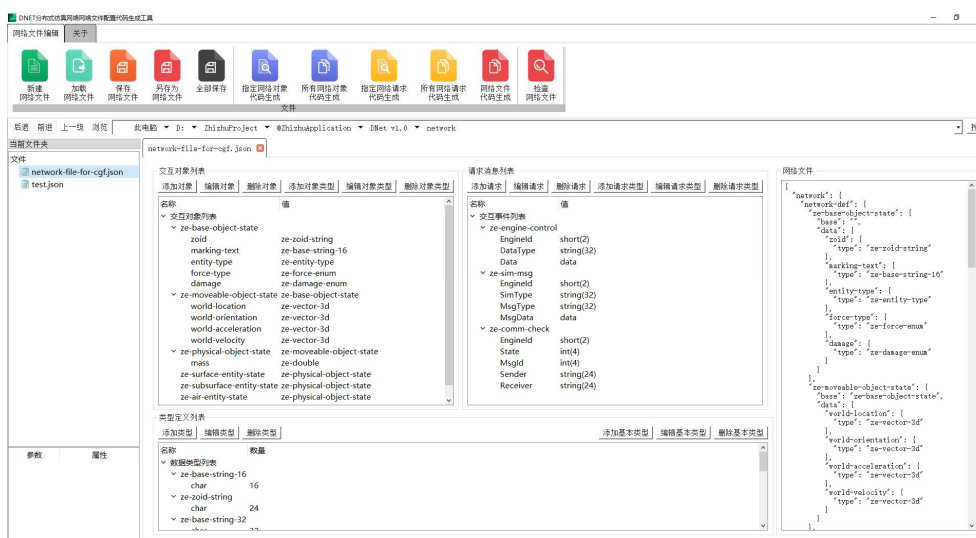


图 22 加载网络文件

### 3.6.3 编辑网络文件

在编辑界面中点击“添加对象”，弹出对话框输入对象名称和继承的对象。

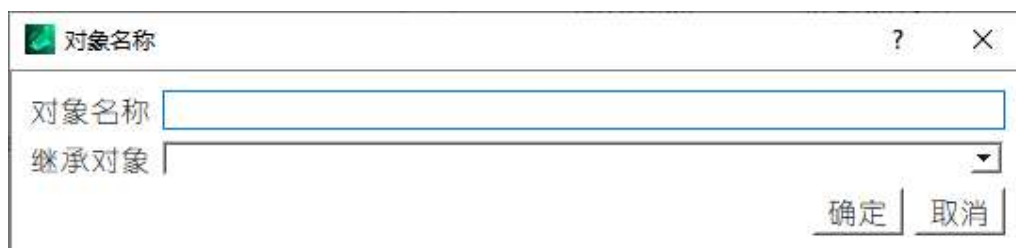


图 23 添加网络对象

输入后点击确定会增加一个网络对象。

选择一个网络对象，点击“编辑对象”，弹出对话框，在里面对内容进行修改。

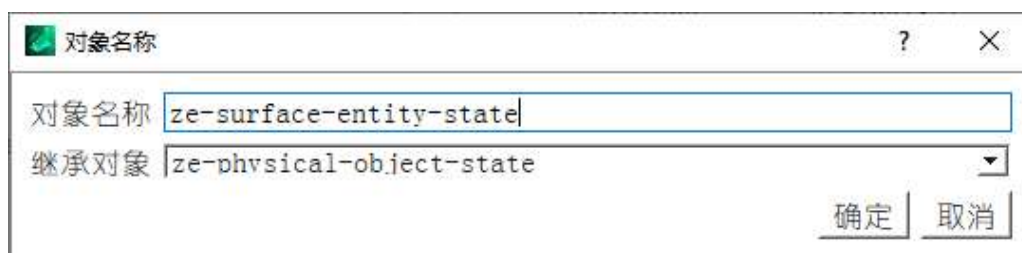


图 24 编辑网络对象

选择一个网络对象，点击“删除对象”，确定删除后，即可删除一个网络对象。

选择一个网络对象的节点，点击“添加对象类型”，弹出对话框中编辑，即可向网络对象中添加一个对象数据类型。

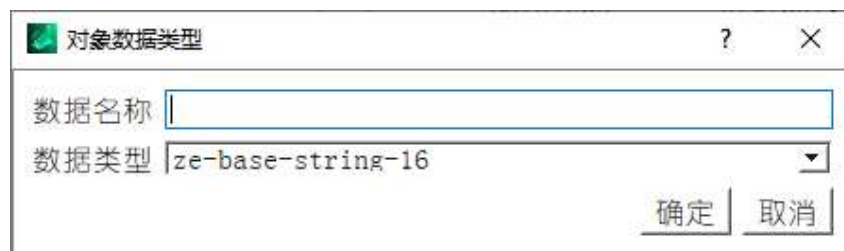


图 25 添加对象数据类型

选择一个对象数据类型节点，点击“编辑对象类型”，弹出对话框

进行编辑。点击确定即可。

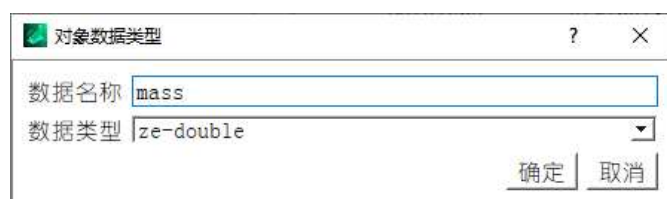


图 26 编辑对象数据类型

选择一个对象数据类型节点，点击“删除对象类型”，确定即可删除。

点击“添加请求”，弹出对话框输入对象名称，点击确定即可增加一个新的请求消息。

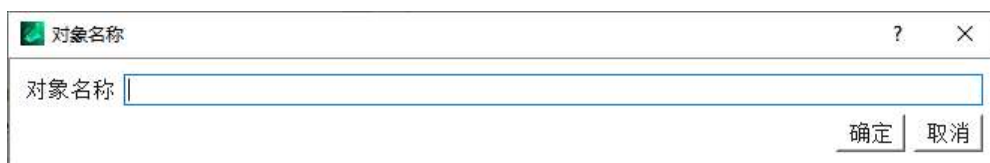


图 27 添加新请求

选择一个请求对象，点击编辑请求，弹出对话框编辑，点击确定即可完成修改。



图 28 编辑请求对象

选择一个请求消息对象节点，点击“添加请求类型”，弹出对话框，增加请求消息对象的数据类型。

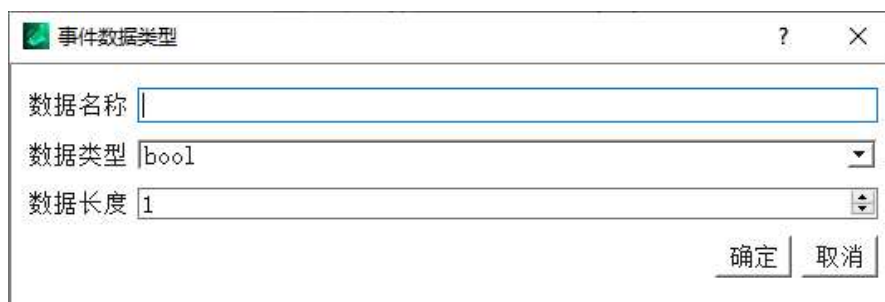


图 29 添加请求数据类型

选择一个请求消息对象的数据类型节点，点击“添加请求类型”，

弹出对话框，编辑请求消息对象的数据类型，点击确定完成修改。



图 30 编辑请求数据类型

选择一个请求消息对象的数据类型节点，点击“删除请求类型”，确定后删除。

### 3.6.4 保存网络文件

点击“保存网络文件”，文件将被保存。

### 3.6.5 另存为网络文件

点击“另存为网络文件夹”，选择一个新的文件位置，即可另存为网络文件。

### 3.6.6 全部保存

点击全部保存，所有打开的网络文件都会被保存。

### 3.6.7 指定网络对象代码生成

选择一个网络对象节点，点击“指定网络对象代码生成”。会弹出对话框，如下：

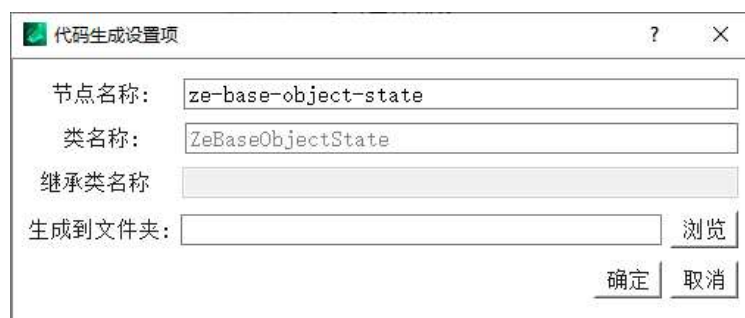


图 31 网络对象代码生成

点击浏览，选择一个文件夹，点击确定，即可在该文件夹下生成该网络对象的类。

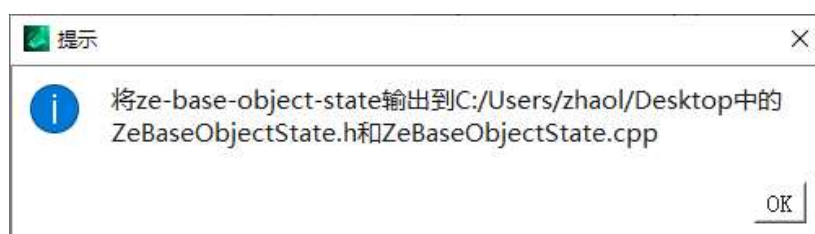


图 32 生成完成

### 3.6.8 所有网络对象代码生成

点击“所有网络对象代码生成”，选择输出的文件夹，即可在该文件夹下生成当前页面中的所有网络对象的代码。

### 3.6.9 指定网络请求代码生成

选择一个网络请求消息节点，点击“指定网络请求代码生成”，会弹出对话，如下所示：

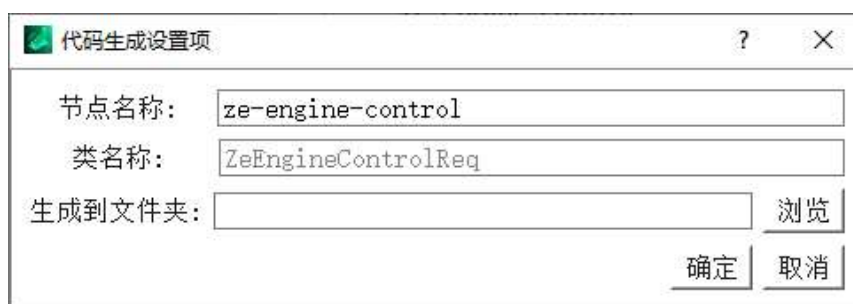


图 33 网络请求代码生成

点击浏览，选择一个文件，点击确定，即可在该文件夹下生成该网络请求消息的类。

### 3.6.10 所有网络请求代码生成

点击“所有网络请求代码生成”，选择输出的文件夹，即可在该文件夹下生成当前页面中的所有网络请求消息的代码。

### 3.6.11 网络文件代码生成

点击“网络文件代码生成”，选择输出的文件夹，即可在该文件夹下生成当前网络文件的所有代码。

### 3.6.12 检查网络文件

完成网络文件的配置和设定后，点击“检查网络文件”，如果网络文件正确，会弹出：



图 34 检查网络文件

否则会弹出错误消息。



## 4 DNet 开发库开发说明

### 4.1 客户端开发说明

#### 4.1.1 客户端开发步骤和说明

客户端开发时遵照以下步骤即可：

- 定义网络文件
  - 网络文件定义原则：不同节点的网络文件可以不同，网络对象和请求消息定义可以不一样，多个节点定义了多个网络文件时，只需要保证公有的网络对象、请求消息、数据类型定义一致即可。因为有些网络对象和请求消息不是所有的网络节点都需要，只需要定义自己需要的那部分即可。
  - 使用 DNetAppTool 定义网络文件，将网络文件保存到可以访问的目录下即可。
  - 使用 DNetAppTool 中生成网络文件的 C++代码，将代码加到工程中，如无其他需求一般可以直接使用，也可以在其基础上做一些处理，比如加密、解密、有需要的代码增加等。
- 在开发的工程中增加代码
  - 将生成的代码加入工程。
  - 在工程中增加 `#include <ZZDNet.h>` 头文件定义，代码中增加 ZZDNet 类的实例：

```
ZZDNet net("publisher", "../network/test.json"); // 必要增加
```

```
net.setLocalNetwork("192.168.1.101"); // 非必要增加, 选择网络, 默认是127.0.0.1
net.setUdpNetwork("225.8.8.8", 10800); // 非必要增加, 设置组播
```

注意 192.168.1.101 是当前网络中计算机的网卡地址, 会选择当前地址的网卡进行连接, 225.8.8.8 为组播地址, 与服务器一致即可, IPv4 的全局组播地址范围是 224.0.1.0 - 238.255.255.255, 按照这个范围定义即可。然后启动:

```
net.start(); // 组播寻找服务器
或
net.startDirectMode("192.168.1.100", 9050); // 直连服务器, 假如服务器地址是
192.168.1.100
```

使用 start 或 startDirectMode 选择一个启动接口启动 DNet。返回值为 0 代表成功, -1 代表网络配置文件错误, -2 代表应用线程启动失败。

- 公布网络对象, 创建网络对象, 在需要设置的地方设置对象的参数:

```
NetObjectStateObject obj(&net);
obj.setName("abcdef");
```

调用 net.tick() 接口, 网络对象会被发送到网络中。

- 订购网络对象, 创建 ZZDNetObjectList 对象:

```
ZZDNetObjectList *list = new ZZDNetObjectList(&net, NetObjectStateObject::create);
```

接口使用 ZZDNet 实例和网络对象的创建接口, 然后遍历列表就可以获得网络中的网络对象了。

- 公布或订购请求消息前, 注册请求消息对象:

```
net.registerRequest(UserCustomRequestType, UserCustomRequest::create);
```

这样就可以在内部创建或使用该对象了。

- 公布请求消息, 从 net 中创建 req, req 返回 nullptr 的话, 代码没有请求消息注册, 设置好参数和值后, 使用 send 发送请

求消息：

```
UserCustomRequest *req =  
dynamic_cast<UserCustomRequest *>(net.createRequest(UserCustomRequestType));  
// 设置参数和值  
// ... ..  
// ... ..  
req->send(&net);
```

- 订购请求消息，创建一个回调函数，然后设置订购请求消息接口，当有请求消息返回时，会自动调用回调函数。

```
void UserRequestCallback(ZZDNetRequest *request, void *usr)  
{  
// 处理代码  
}  
net.subscribeRequest(UserCustomRequestType, UserRequestCallback, 0);
```

- 断开连接

直接释放 ZZDNet 的实例，即可断开连接。

以上是客户端开发步骤和必要代码说明，具体请参考示例程序。

net.tick() 接口说明，net.tick() 接口最好放到一个循环中使用，这里面是对于网络通信的更新。

## 4.1.2 示例

示例查看安装目录下 example 文件夹中的示例,均使用 test.json:

publishObject (公布网络对象)

publishRequest (公布网络请求消息对象)

subscribeObject (订购网络对象)

subscribeRequest (订购网络请求消息对象)

## 4.1.3 API 接口说明文档

在 doc 文件夹中有 API 接口说明文档,文档位于安装目录下的 doc 文件夹中:安装文件夹\doc\api\DNet\index.html。

## 4.1.4 连接模式说明

DNet 开发库通信连接模式分为两种:

第一种是使用 UDP 组播自动寻找已有服务器进行组网。由不同 UDP 组播地址,来完成客户端分组加入不同的服务器。

```
net.start(); // 组播寻找服务器
```

第二种是使用 TCP/IP 协议直接定义 IP 地址和端口直接连接服务器。

```
net.startDirectMode("192.168.1.100", 9050); // 直连服务器, 假如服务器地址是 192.168.1.100
```

### 4.1.4.1 UDP 组播连接模式

**工作原理:** 客户端通过 UDP 组播地址自动寻找网络中的服务器,

并加入相应的组网。不同的 UDP 组播地址可以将客户端分组连接到不同的服务器。

**适用场景：**适用于网络硬件环境支持 UDP 组播协议的情况。UDP 组播能够有效地减少网络流量，特别适合在局域网内进行高效的组网通信。

**优点：**

**自动化：**客户端可以自动发现并连接到服务器，减少了手动配置的复杂性。

**高效性：**UDP 组播可以减少网络中的冗余数据包，提升通信效率。

**缺点：**

**硬件依赖：**需要网络硬件支持 UDP 组播协议，某些网络环境可能不支持。

**配置复杂：**在某些复杂的网络环境中，UDP 组播可能需要额外的配置和管理。

#### 4.1.4.2 TCP/IP 直接连接

**工作原理：**客户端通过 TCP/IP 协议直接连接到指定的服务器 IP 地址和端口。这种方式不依赖于 UDP 组播，而是通过明确的 IP 地址和端口进行通信。

**适用场景：**适用于网络硬件不支持 UDP 组播协议的环境，或者需要更稳定、可靠的连接场景。

**优点：**

**稳定性:** TCP/IP 协议提供了可靠的连接, 确保数据包的顺序和完整性。

**通用性:** 几乎所有的网络环境都支持 TCP/IP 协议, 适用性广泛。

**缺点:**

**手动配置:** 需要手动指定服务器的 IP 地址和端口, 自动化程度较低。

**网络负载:** 相比 UDP 组播, TCP/IP 连接可能会产生更多的网络流量, 尤其是在大规模组网时。

**UDP 组播**适合在局域网内进行高效的自动组网, 尤其是在硬件支持 UDP 组播的情况下。它能够减少网络负载并简化客户端的连接过程。

**TCP/IP 直接连接**则更适合在广域网或不支持 UDP 组播的环境中使用, 提供了更高的稳定性和通用性, 但需要手动配置连接信息。

与 HLA 协议的 RTI 对比, HLA (High-Level Architecture) 协议中的 RTI (Run-Time Infrastructure) 也面临类似的问题, 特别是在 UDP 组播不可用或不稳定的情况下。DNet 开发库提供的第二种连接方式 (TCP/IP 直接连接) 类似于 HLA 中的备选方案, 确保了在网络环境不支持 UDP 组播时, 仍然能够通过 TCP/IP 进行可靠的通信。

**UDP 组播:** 适合局域网内的高效自动组网, 依赖硬件支持。

**TCP/IP 直接连接:** 适合广域网或不支持 UDP 组播的环境, 提供稳定可靠的连接。

开发者可以根据具体的网络环境和需求选择合适的通信模式。



## 4.2 服务器端二次开发说明

### 4.2.1 服务器端二次开发说明

服务器端提供API可以自行定义类似ZZDNetApp的服务器端应用。

### 4.2.2 API 接口说明文档

在 doc 文件夹中有 API 接口说明文档，文档位于安装目录下的 doc 文件夹中：安装文件夹\doc\api\DNet\index.html。使用浏览器查看。



图 35 API 接口文档界面



## 5 常见问题说明

### 5.1 常见错误

- 应用程序无法执行

建议安装路径中不要带中文，可能造成应用程序启动错误，主要是由于 Qt 库导致的。尤其是银河麒麟 V10 的版本，不要将内容放到 /home/文档或其他/home 中的文件夹下，其是默认中文目录，虽然测试过可以运行，但是还是建议可以将软件放到数据盘/data/目录下。

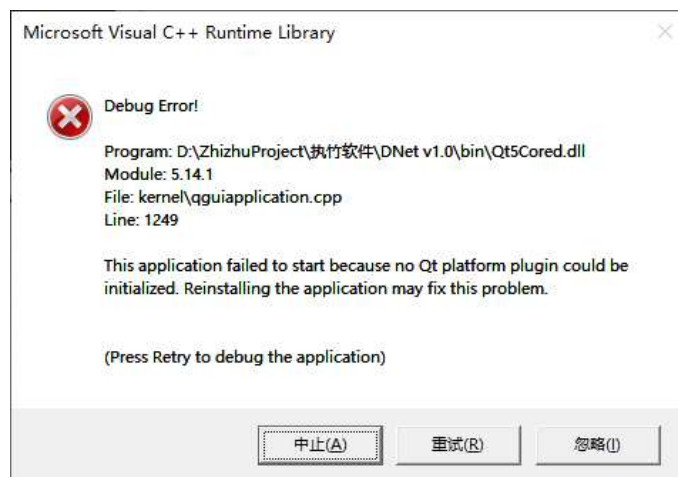


图 36 应用程序无法执行信息

请查看一下安装的 Qt 版本：

Window 版本：Qt5.14.1

银河麒麟 V10：Qt5.12.10

Ubuntu20.04：Qt5.12.8

其他版本请联系我们，可以提供帮助。

- Linux 下编译和启动错误

因为 Linux 下的编译程序过程需要设置相关路径，本软件都使用

相对路径，一般都是./当前路径，可以使用 `ldd XXXX.so` 来查看，保证 `so` 调用路径的正确性。在编译过程中，会遇到 `undefine` 的情况，该情况解决方法是查看链接过程中，由于 C++ 中部分类是继承的关系，需要调整 `Makefile` 中的设置：

```
LDLFLAGS = -lZZDNet -lZZSocket -lZZThread -L$(TARGET_DIR)
-Wl,-rpath,./ -lpthread

$(CXX) -o $@ $^ $(LDLFLAGS)
```

`LDLFLAGS` 一定要在最后，这个可以参考 `example` 中的 `Makefile` 编写方式。

银河麒麟操作系统的 GCC 版本是 9.4.0 版本。

Ubuntu20.04 操作系统的 GCC 版本是 9.3.0 版本。

GCC 编译器一般不会出现问題，但版本差异较大的话，可能需要提供对应版本，协助生成。

如遇其他问题，可以联系我们，我们可以提供技术支持。

启动过程中遇到问题，请检查 `so` 文件、运行路径、环境变量、Qt 库是否正确，如遇其他问题，可以联系我们，我们可以提供技术支持。

### ● 网卡地址未选择

网卡地址选择错误，可能导致客户端无法连接到服务器端，在客户端一直在连接，结果无法连接到服务器的时候，可以对 DNet 服务器端的网络选择进行检查，查看客户端是否连接的是当前对应的网络。

### ● 网卡端口冲突

DNet 服务器端无法正常启动服务器时，查看一下运行记录的内容，

是否存在网络端口冲突的情况。在冲突的情况下，运行记录中会产生 Server TCP 信息:SOCKET\_BIND\_ERROR

- 客户端自动连接无响应

由于组播 IP 地址是有限制的，地址由 224.0.0.0 到 239.255.255.255。如果组播地址错误的话，是无法通过 UDP 组播获得服务器地址的。所以，请检查组播 IP 地址是否错误。如无错误，请检查当前网络环境硬件是否支持 UDP 组播协议，如不支持，请使用直连的方式连接服务器，或更换网络硬件设备。

注意：由于自带的示例程序都是基于 127.0.0.1 的地址，可以自行修改示例代码中的绑定的网卡地址，让其处于同一网络中即可。

## 5.2 日志文件

客户端日志文件在执行客户端程序的文件夹路径下会生成 DNet.log。

服务器端日志文件在服务器执行目录下生成 DNetServer.log。

## 6 附录

### 6.1 版本历史

V1.0 发布于 2025 年 3 月 25 日

V1.1 发布于 2025 年 5 月 26 日

### 6.2 版权信息

北京执竹科技有限公司张三 版权所有 © 2025 保留所有权利

### 6.3 第三方库声明

界面开发库,使用基于 LGPL 许可下的模块的 Qt5.14.1 版本开发,  
具体信息参考 Qt 的 LGPL 协议。

字体库使用 AlibabaSans-Light.otf。