目 录

第1章　ETest 简介

1.1 ETest

ETest 是嵌入式系统半实物仿真测试开发环境，有如下特点：

1. 种类齐全的接口/总线

PXIe、PCIe、USB 等多种总线可选，支持 RS232/422/485、1553B、CAN、FlexRay、TCP、UDP、AD、DA、DI、DO、ARINC429、AFDX、RapidIO、I2C、SPI、FC-AE-ASM、USB 等多种总线接口

2. 可视化与脚本开发方式

既可以可视化创建状态机、通信时序、信号处理等多种可执行模型，也可以使用脚本编程实现灵活丰富的动态控制功能；内置百余项 API 和界面组件，让测控系统开发变得轻松、简单

3. 强大的自动化测试功能

自动化执行测试用例的同时，也支持加入人工参与环节；提供全方位的执行过程监控手段，测试过程数据自动记录；内置高性能实时数据库，支持海量数据存储与处理

4. 内置多种用例自动生成算法

只需创建因果图、组合对、流程图、协议定义等模型，即可自动生成测试用例；支持等价类、边界值、组合算法、路径深度算法、模糊突变算法等多种测试用例生成算法

5. 自动生成结果分析报告

支持自动生成 office 格式的测试文档，测试文档格式可定制；支持多版本测试结果管理；内置回归分析、统计分析、时域/频域分析等多种常用数据分析功能

6. 强实时性、高可靠性

自主研发的实时调度模块，支持高可靠性测试和强实时测试，响应时间小于 1ms，同步传送和抖动时间小于 10μs；配合 FPGA 使用可满足更高实时性需求场景

7. 快速生成完整测试系统

内置多种自动生成功能，可用于快速创建功能模块；具有完整开发工具链，支持一键打包输出测控程序，输出的程序可再次分发、支持跨平台部署、支持单机或分布式部署

8. 与建模工具无缝集成

兼容 FMI 标准模型，支持加载 FMU 模型至半实物仿真环境；运行实时硬件在环仿真的同时，支持动态调整模型参数；与 Simulink、同元 MWorks 等建模工具无缝集成

9. 部署灵活且易于扩展

支持 windows、Linux、RTLinux、中标麒麟、银河麒麟等多操作系统部署，支持单机部署、分布式部署；硬件板卡和软件模块均可自由组合配置，支持自定义集成与扩展

1.2 安装部署

1.2.1 Windows

1. 下载并解压安装包
2. 打开解压目录下的 dev 目录，双击该路径下的 exe 可执行程序即可完成启动。

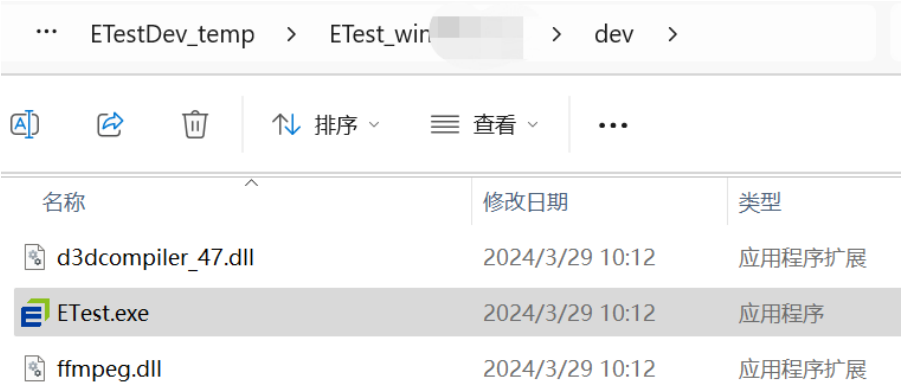

图 1-1 双击 ETest.exe

注意：
软件为绿色免安装版，无需安装，解压即可启动运行。
首次启动时系统会提示是否允许访问网络，必须选择“允许”。
配置 Python 环境：

1、 CMD 命令终端默认设置为 powershell。
2、 将安装路径下 Python 设置为系统环境 Python。

如果 ETestD4 无法启动，可能是系统缺少运行时库，请执行以下操作：
（1） 双击附件 VC_redist.x64
（2） 勾选我同意许可条款和条件后，点击安装。
（3） 等待安装完成，安装完成后重启计算机。

1.2.2 Linux

1. 桌面操作系统

（1） 复制软件安装包至 Linux 系统文件路径下。
（2） 解压安装包。
（3） 进入 bin 目录，终端管理员权限执行命令 sudo ./ETestD4
（4） 进入 dev 目录，点击可执行程序 ETest 启动 IDE。
（5） 右下角显示绿色在线状态即表示启动成功。

2. 实时下位机

以 IDE 安装在 windows 上，执行器安装在 LinuxRT 上为例

（1） 复制软件安装包至 Linux 系统文件路径下。

（2） 解压安装包。

（3） 在解压路径下执行命令 cd setup 进入安装目录。

（4） 执行 sudo sh install setup.sh 即可完成安装。

（5） 重启 LinuxRT 计算机

（6） 在 windows 上打开 ETest.exe->设置->执行器连接设置->填写实时下位机的 IP 地址->点击启用

图 1-2 打开执行器连接设置

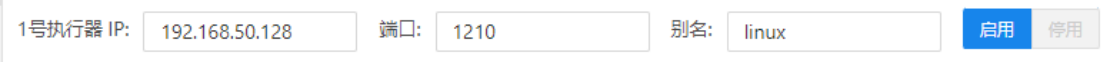

图 1-3 添加 Linux 的 IP 地址

1.3 界面布局

ETest 主界面（IDE）被分为以下 6 个区域:

1. 工具栏：位于 IDE 的最上方，提供多种工具。
2. 活动栏：位于最左侧，方便在不同视图之间切换。
3. 侧边栏：位于活动栏的右侧，通过 Ctrl + B 实现打开或者关闭。
4. 编辑器：这里是编辑区，支持打开多个编辑器。
5. 面板：编辑器的下方展示不同的面板，包括诊断、警告、输出、终端。
6. 状态栏：显示当前打开的文件、项目、执行器连接状态、执行按钮等其他基本信息。

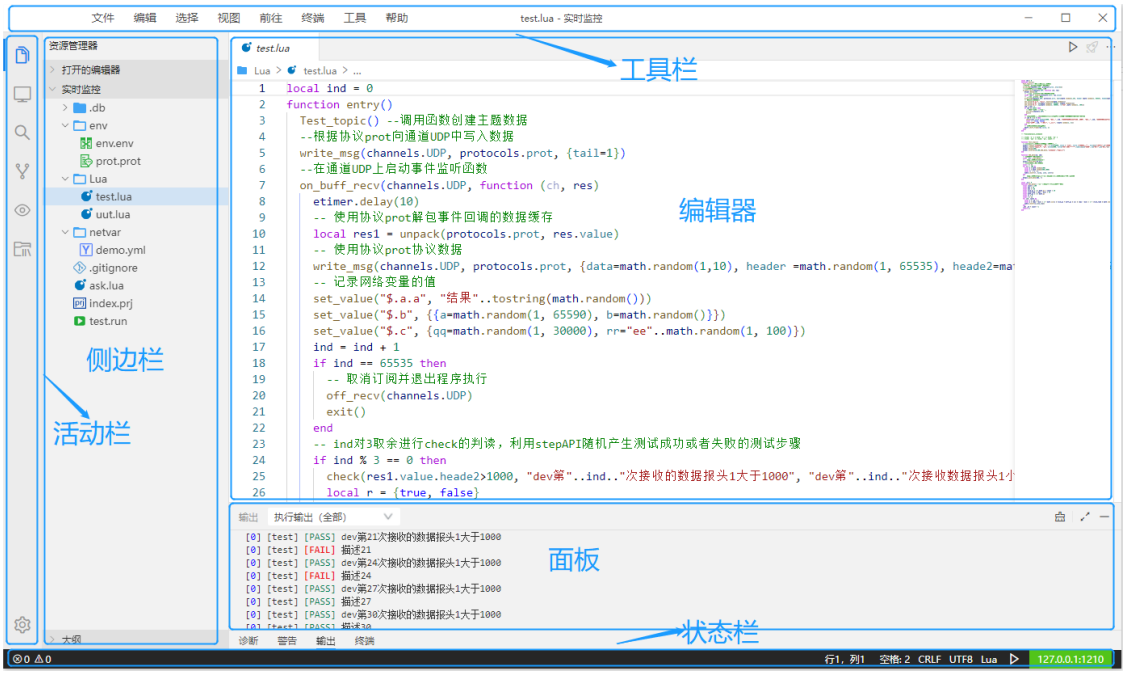

图 1-4 界面布局

1.4 工作模式

ETest 可以在嵌入式设备开发的各个阶段使用。本文介绍三种工作模式：自动化测试，工装开发，批量执行，实时监控，实时仿真等。详细内容参考后续章节。

1.4.1 自动化测试

用户用编写测试脚本的方式，完成多个测试用例；ETest 自动执行测试用例，执行后，自动记录所有的测试数据；并且对测试结果数据自动判读，形成测试报告。适用于批量测试、无人值守测试。

主要特点：自动执行，可定制报告模板。

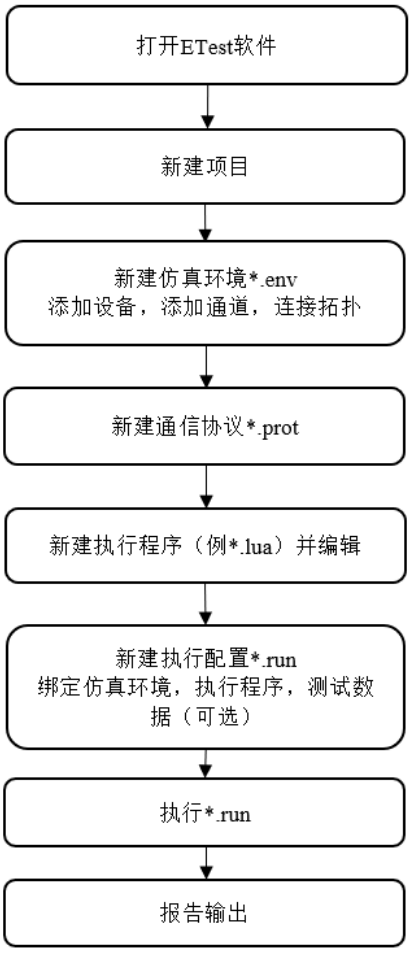

图 1-5 自动化测试流程

1.4.2 工装开发

ETest 自带常见的控件，可快速创建自定义用户界面，并通过数据绑定实现控件与测试数据的关联，形成测试工装应用。整个应用能够按照用户的方式进行运行，使用用户选择或输入的测试数据；完成测试后可以按照用户习惯的方式显示测试数据，并进行报警记录操作，方便用户操作，并能够进行实时分析。

主要特点：可视化界面设计，输出独立运行程序。

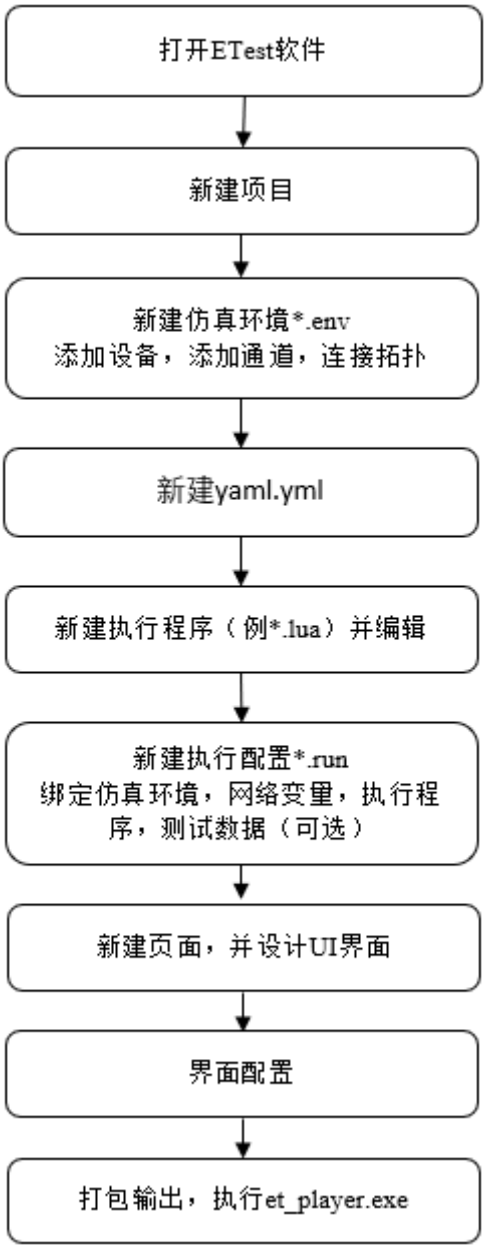

图 1-6 工装开发流程

1.5 使用示例项目

我们准备了大量的示例项目，方便大家在实践中学习。下面以 UDP 为例，介绍示例从下载到执行的过程。

1. 下载示例

帮助->文档->右上角输入框输入【UDP】->点放大镜->选择【使用 UDP 通道】->点击示例->另存为->成功下载示例

图 1-7 帮助文档

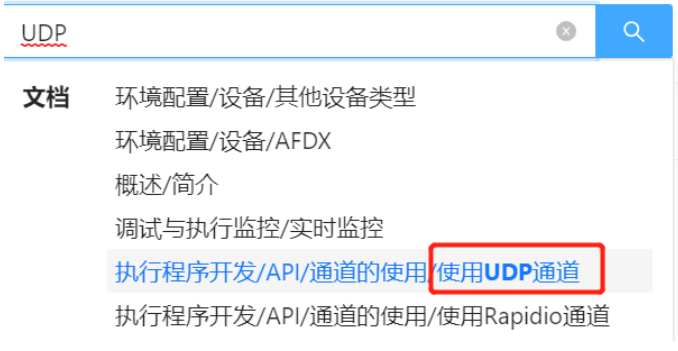

图 1-8 文档右上角搜索 UDP

图 1-9 点击示例，另存为

2. 打开示例项目

文件->打开项目->选择文件夹【UDP 通道测试】

图 1-10 打开项目

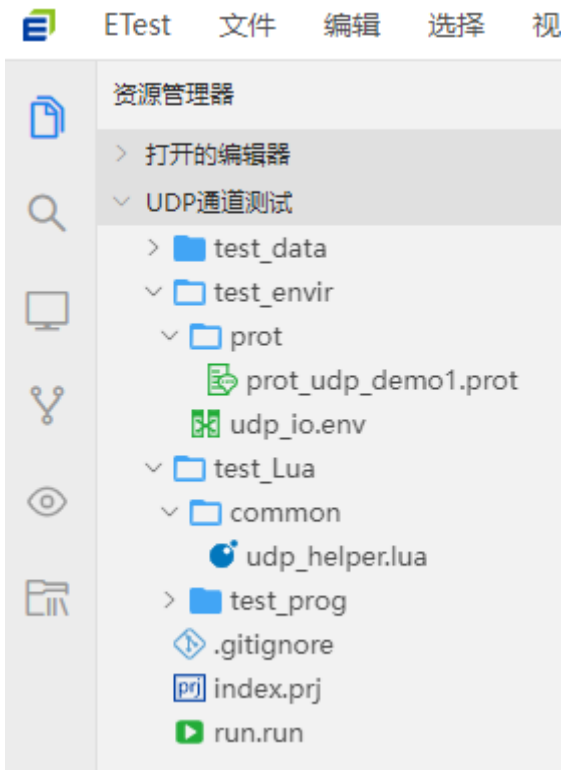

图 1-11 打开项目【UDP 通道测试】

3. 执行：打开 run.run 文件，点击右上角的三角执行。

图 1-12 执行

1.6 说明

1.6.1 文件夹和文件的名称

ETest 的文件夹，文件的名称都可自定义（除了系统文件和文件夹，例：index.prj）。但是必须遵循命名规则：名字为中文，英文，数字，下划线。

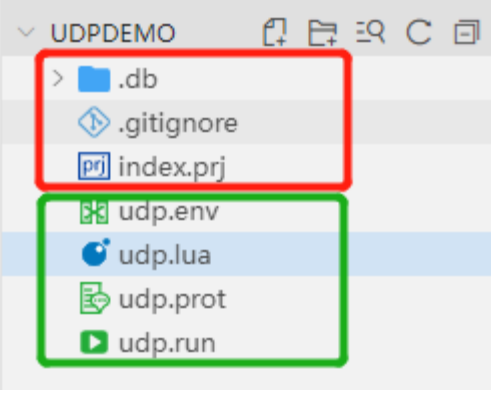

图 1-13 红框为系统文件夹和文件不可修改，绿框的为用户自己添加的文件可以修改

1.6.2 API 的提示功能和查看文档

当打开一个 lua 脚本，遇到不太熟悉的 API（函数），鼠标悬浮到该 API 上。以 write_msg 为例，鼠标放到 write_msg 上面就会显示关于 write_msg 的功能，参数和返回值有哪些。还可以点击【查看文档】，跳转到对应的说明。

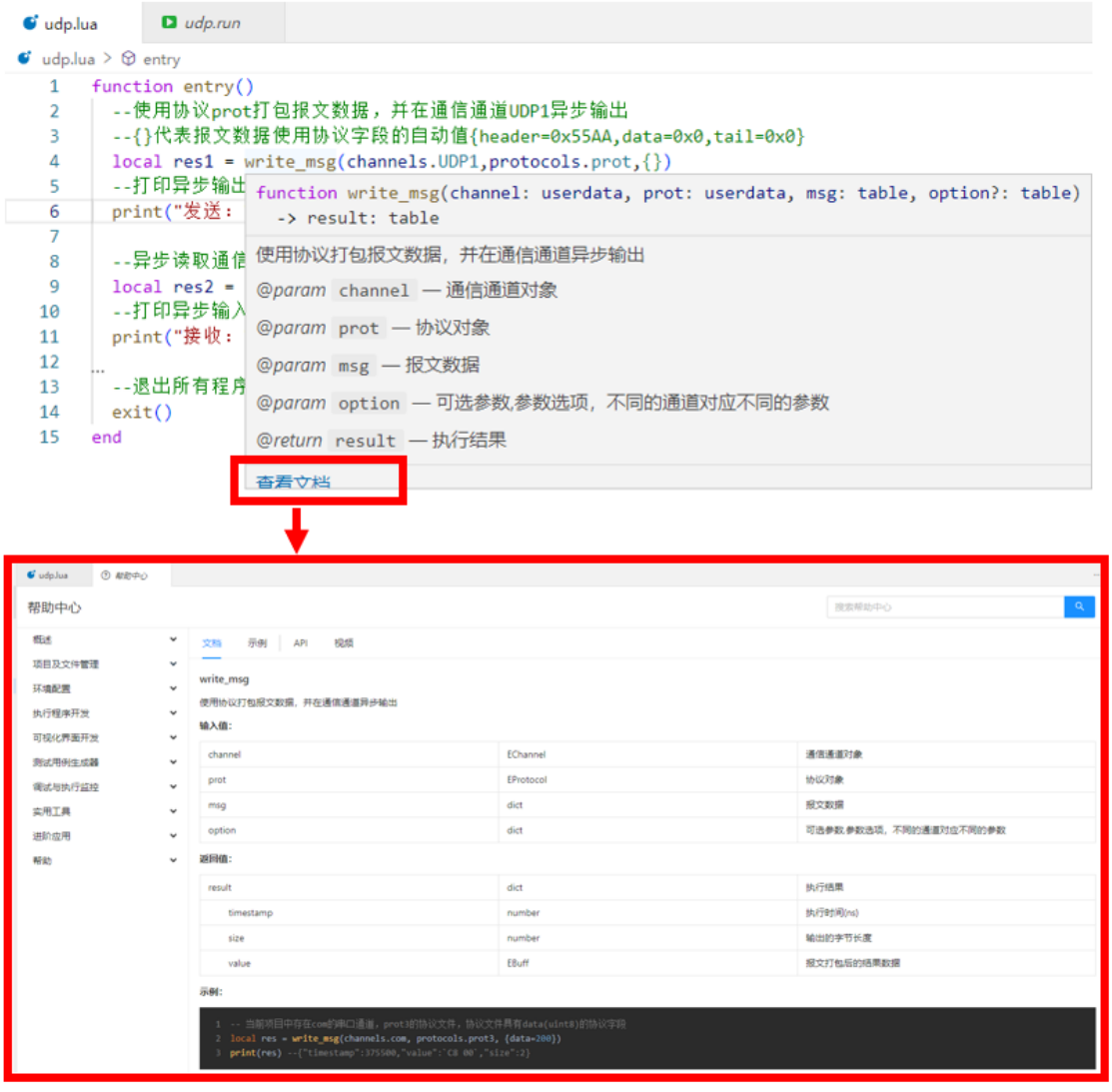

图 1-14 write_msg 的提示和查看文档

第2章　自动化测试

2.1 Lua 实现 UDP 通道测试

1. 双击 exe 可执行程序，启动 IDE

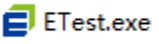

图 2-1 打开 ETest

2. 新建一个空项目
   依次选择文件->新建项目->空项目->新建文件夹（udpdemo）->选择该文件夹

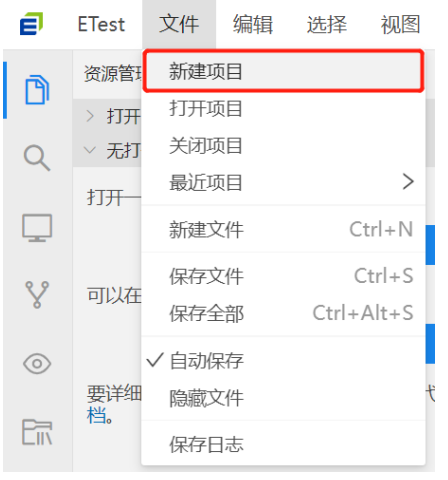

图 2-2 新建项目

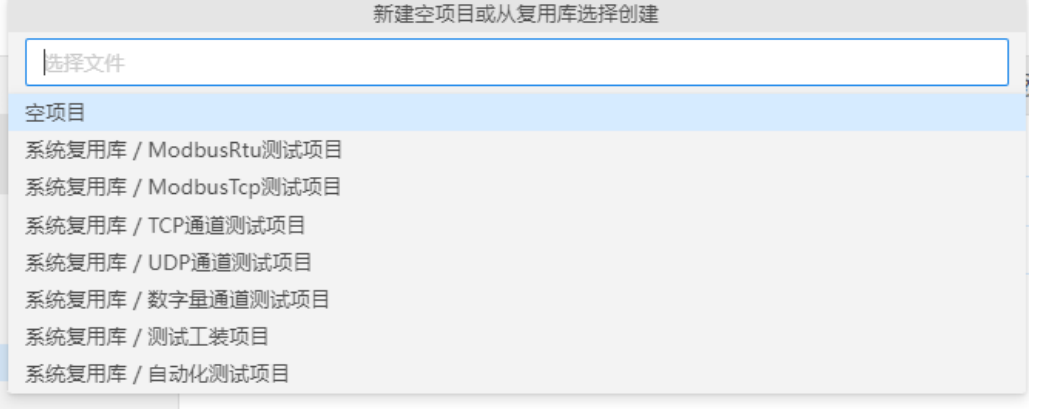

图 2-3 点击空项目

图 2-4 项目新建成功

3. 新建【仿真环境】udp.env
资源管理器->空白处右键->新建文件->仿真环境->输入仿真环境名称 udp

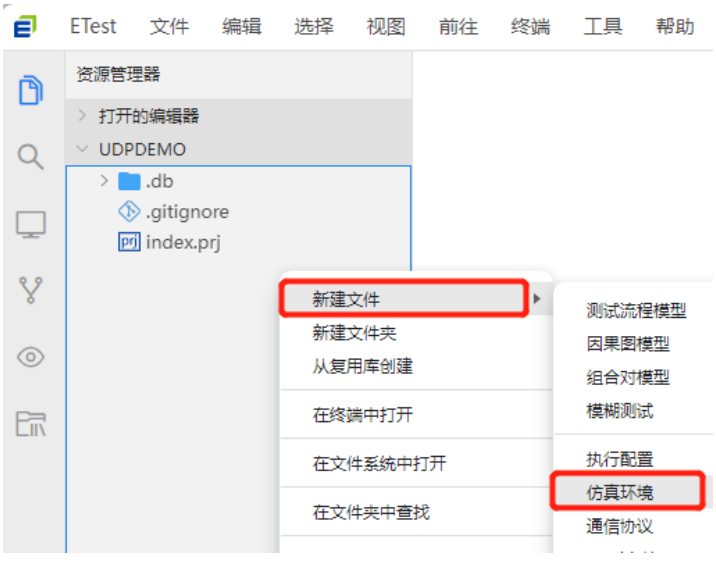

图 2-5 新建仿真环境

4. 添加仿真设备：打开 udp.env 文件，拖拽【仿真设备(脚本)】到编辑器

图 2-6 添加仿真设备

5. 设备说明
   - 名称：设备名称可以修改，设备名称可以是中文、英文；设备名称不可以重复。
   - 类型：添加设备后，类型可以进行修改为仿真设备或实物设备。
   - 配置：配置信息显示的是仿真设备进行通信时使用的执行器信息；实物设备不显示配置信息。
   - 配置删除：当该设备添加通道，击 X，可以删除该设备下所有通道的【配置】。
   - 删除设备：点击设备后的垃圾桶按钮，设备可以被删除；选中一个设备或多个设备后，鼠标右键菜单点击删除或操作快捷键 Delete，删除设备。

6. 添加 UDP 通道：
   拖拽两个【UDP 通道】到【仿真设备(脚本)】下，一个名称为 UDP1，端口号默认，另一个名称为 UDP2，端口号改成 4001。【配置】栏需要绑定通道 TCPIP::UDP::0

图 2-7 添加通道

图 2-8 端口设置

7. 通道说明
   - 名称：通道名称可以修改，名称是由字母、数字、下划线组成，不能以数字开头，不能是关键字，不能是中文；同一个设备下的通道名称不可以重复。
   - 类型：通道添加后，通道类型可以进行修改。
   - 配置：下拉列表选中要绑定的通道，该操作是把对应的执行器逻辑通道绑定到板卡中的物理通道上或模拟的物理通道上，只有绑定通道后，才能实现数据通信；同时也支持自动绑定，点击配置后的↓图标，会自动进行绑定。
   - 删除配置：点击 X，删除该通道的【配置】。
   - 删除通道：点击通道后的垃圾桶按钮，通道可以被删除；选中一个通道或多个通道后，鼠标右键菜单点击删除或操作快捷键 Delete，删除通道。

8. 通道的右侧面板
   - 说明，引脚编号：仅用于显示。
   - 地址，端口：IP 地址当前设备的网络标识。端口号范围 0-65535。
   - 存活周期：网络传输过程中经路由分发的次数。例，存活周期是 5 的时候。则表示只能分发五次，第六次就无效了。类型为 number。
   - 绑定组播地址(Linux)：只在 Linux 下设置。需要组播的时候，填写组播的 IP 地址。
   - 复用地址端口：boolean 类型，勾选时候，代表多个应用程序可以同时使用同一个 UDP 端口进行通信。这样多个程序可以监听同一个端口并处理不同的请求。

9. 连接拓扑

   点击【切换】，切换拓扑结构，连接两个 UDP 通道（鼠标放到 UDP1 后面空白处，拖拽到 UDP2 的前面空白处，然后松手）

图 2-9 切换

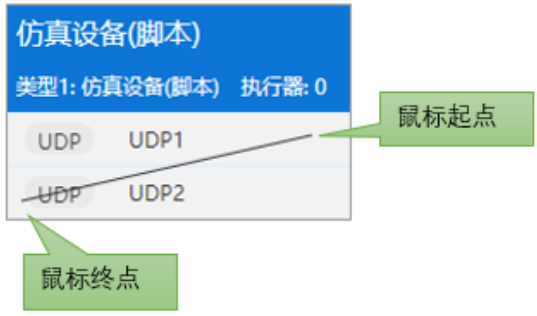

图 2-10 连线的起点和重点

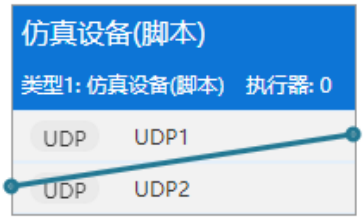

图 2-11 连线成功

10. 新建通信协议 udp.prot，协议内容默认即可。

图 2-12 新建通信协议

图 2-13 udp.prot 的默认内容

11. 新建 udp.lua

图 2-14 新建 lua

12. 编辑 udp.lua

```
function entry()
  --使用协议 udp 打包报文数据，并在通信通道 UDP1 异步输出
  --{}代表报文数据使用协议字段的自动值{header=0x55AA,data=0x0,tail=0x0}
  local res1 = write_msg(channels.UDP1,protocols.udp,{})
  --打印异步输出的执行结果
  print("发送：",res1)
  --异步读取通信通道 UDP2 的输入数据并用协议 udp 解析
  local res2 = read_msg(channels.UDP2,protocols.udp)
  --打印异步输入的执行结果
  print("接收：",res2)
  --退出所有程序执行
  exit()
end
```

13. 新建执行配置 udp.run

【执行配置】是配置文件，可以告诉 ETest 用什么仿真环境的什么设备来执行什么程序，同时可以填写一些说明性文字，用于输出报告。还可以选择协议通道等在实时监控中显示。

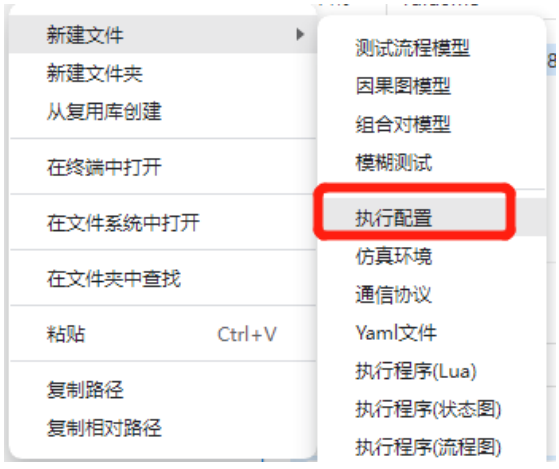

图 2-15 新建执行配置

14. 打开 udp.run，绑定仿真环境，执行程序

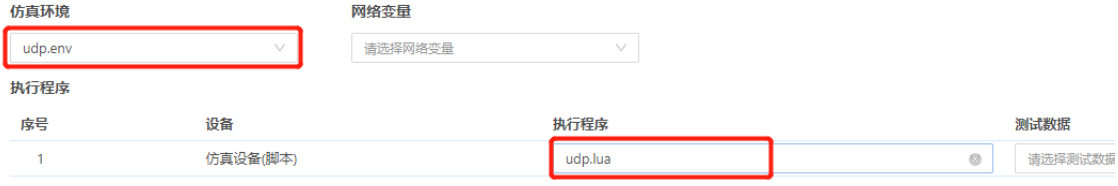

图 2-16 设置执行配置

15. 执行：点击 udp.run 或者 udp.lua 脚本的右上角的三角符号执行，在下方的【执行输出】看执行结果。

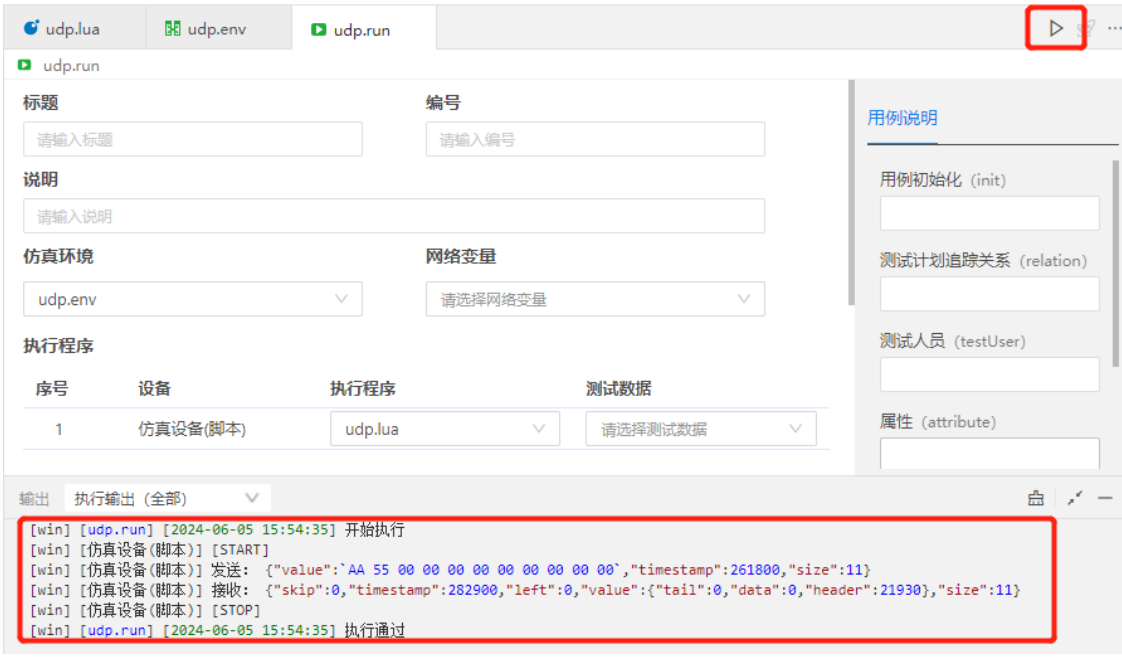

图 2-17 执行

2.2 Python 实现 COM 通道测试

1. 双击 exe 可执行程序，启动 IDE

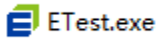

图 2-18 打开 ETest

2. 新建一个空项目
   依次选择文件->新建项目->空项目->新建文件夹（python_demo）->选择该文件夹

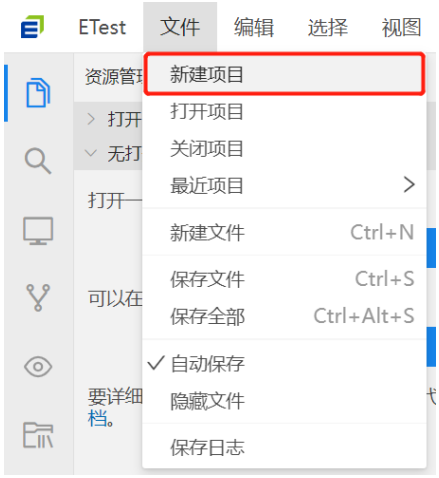

图 2-19 新建项目

图 2-20 点击空项目

图 2-21 项目新建成功

3. 新建【仿真环境】python_demo.env
资源管理器->空白处右键->新建文件->仿真环境->输入仿真环境名称 python_demo

图 2-22 新建仿真环境

4. 添加仿真设备：打开 python_demo.env 文件，拖拽【仿真设备(脚本)】到编辑器

图 2-23 添加仿真设备

5. 设备说明
   - 名称：设备名称可以修改，设备名称可以是中文、英文；设备名称不可以重复。
   - 类型：添加设备后，类型可以进行修改为仿真设备或实物设备。
   - 配置：配置信息显示的是仿真设备进行通信时使用的执行器信息；实物设备不显示配置信息。
   - 配置删除：当该设备添加通道，击 X，可以删除该设备下所有通道的【配置】。
   - 删除设备：点击设备后的垃圾桶按钮，设备可以被删除；选中一个设备或多个设备后，鼠标右键菜单点击删除或操作快捷键 Delete，删除设备。

6. 添加 232 串口通道：
   拖拽两个【232 串口通道】到【仿真设备(脚本)】下，一个名称为 COM1，另一个名为 COM2，【配置】栏分别绑定通道 SERIAL::S232::0、SERIAL::S232::1

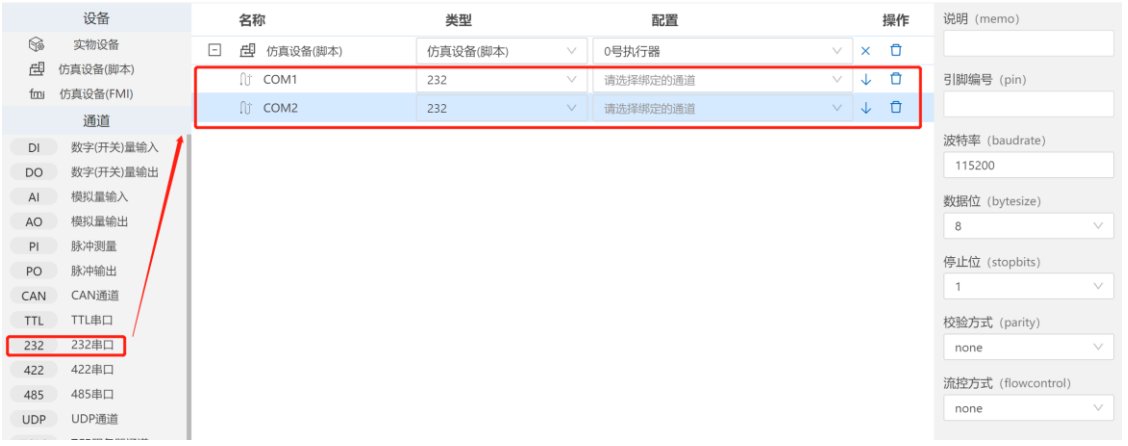

图 2-24 添加通道

图 2-25 通道绑定

7. 串口名称设置

工具->串口名称设置

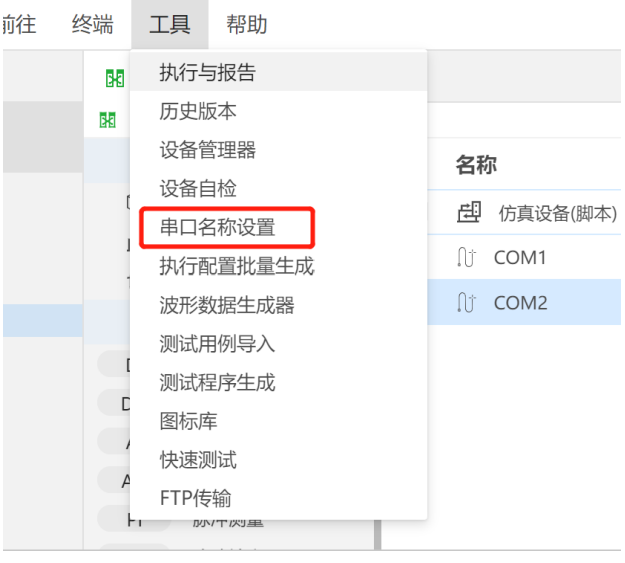

图 2-26 串口名称设置

8. 设置物理串口

串口名称设置界面 232 串口 1 与 232 串口 2 分别设置 COM9 与 COM10

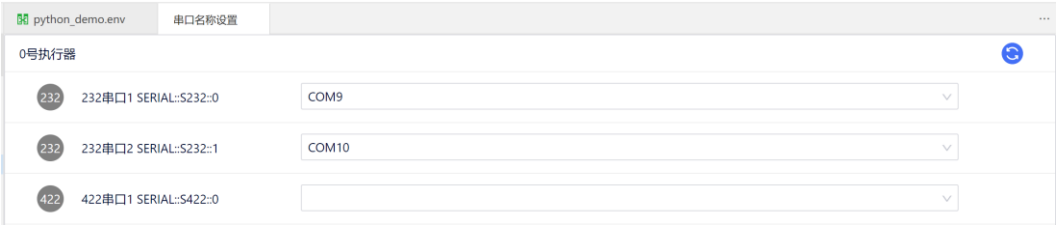

图 2-27 串口名称设置

注意:

通过设备管理器查看串口号，本机 COM9 与 COM10 为虚拟串口。

也可连接实际的串口设备进行设置。

不同的机器串口名称也不一致。

9. 通道说明

- 名称：通道名称可以修改，名称是由字母、数字、下划线组成，不能以数字开头，不能是关键字，不能是中文；同一个设备下的通道名称不可以重复。
- 类型：通道添加后，通道类型可以进行修改。
- 配置：下拉列表选中要绑定的通道，该操作是把对应的执行器逻辑通道绑定到板卡中的物理通道上或模拟的物理通道上，只有绑定通道后，才能实现数据通信；同时也支持自动绑定，点击配置后的↓图标，会自动进行绑定。
- 删除配置：点击 X，删除该通道的【配置】。
- 删除通道：点击通道后的垃圾桶按钮，通道可以被删除；选中一个通道或多个通道后，鼠标右键菜单点击删除或操作快捷键 Delete，删除通道。

10. 通道的右侧面板

- 说明，引脚编号：仅用于显示。
- 波特率：数据的传输速率，支持下拉选择与手动输入。
- 数据位：在串口通信中传输的每个数据字符所占用的位数。
- 停止位：在每个数据帧结束时需要发送的标志位。
- 校验方式：根据通信双方的协商和具体应用场景的要求选择不同的校验方式。
- 流控方式：用于控制数据在串行通信中传输的速度。

11. 连接拓扑

点击【切换】，切换拓扑结构，连接两个 232 串口通道（鼠标放到 COM1 后面空白处，拖拽到 COM2 的前面空白处，然后松手）

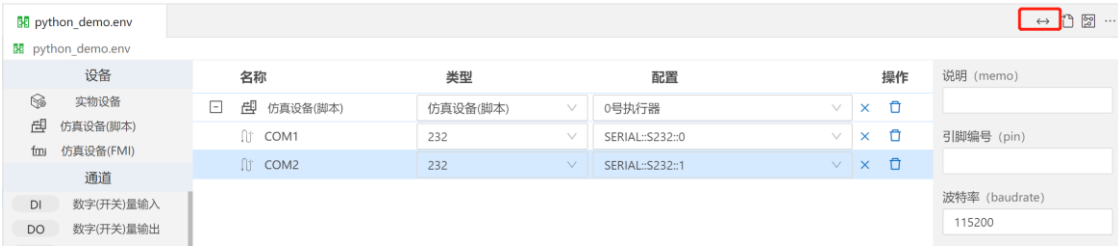

图 2-28 切换

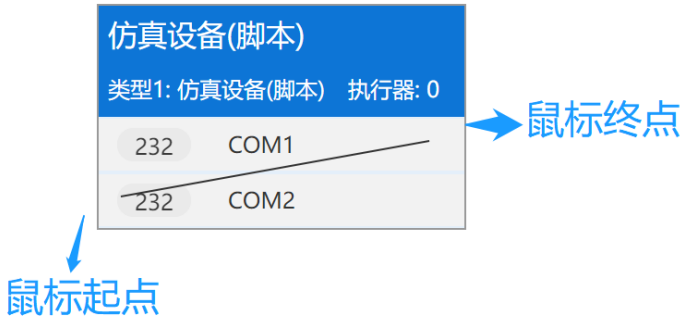

图 2-29 连线的起点和重点

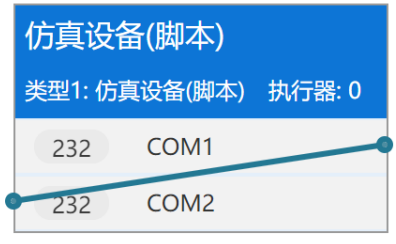

图 2-30 连线成功

注意:

连接拓扑图中的连线仅表示设备之间的连接关系，如果与实物通信要进行物理连接

仿真设备表示测试设备(ETest 进行控制)，实物设备表示被测设备（ETest 无法控制）

12. 新建通信协议 python_demo.prot，协议内容默认即可。

图 2-31 新建通信协议

图 2-32 python_demo.prot 的默认内容

13. 新建 python_demo.py

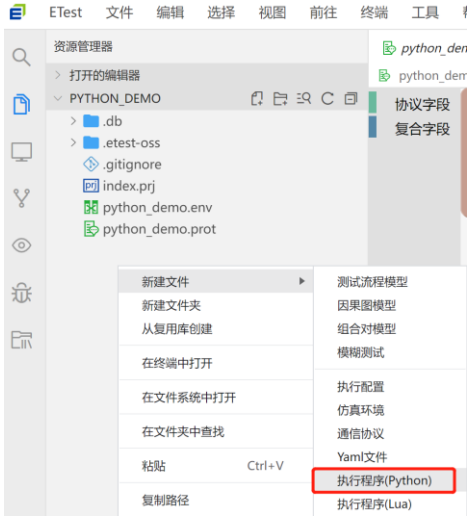

图 2-33 新建 python

14. 编辑 python_demo.py

```
from pysdk import Esys, Eio, Erun #导入模块
import binascii #导入模块
```

```
async def entry():
    # #向 COM1 通道写入数据，发送 16 进制原始串
    await Eio.write_buff_sync("仿真设备(脚本)", "COM1", "01 02 03 04 05 05")
    #从 COM2 通道读数据
    status, res = await Eio.read_buff_sync("仿真设备(脚本)", "COM2", 0, 500)
    #输出状态与字节流
    print(status, res)
    #  字节流输出字串
    value = binascii.hexlify(res).decode("utf-8")
    print(value)
    #构建协议数据
    data = {"header": 0xffff}
     #COM2 通道使用 python_demo 协议写入数据
    await Eio.write_msg_sync("仿真设备(脚本)", "COM2", "python_demo", data)
    #COM1 通道使用 python_demo 协议写入数据
     status, res = await Eio.read_msg_sync("仿真设备(脚本)", "COM1", "python_demo",
500)
    print("协议数据", status, res)
    await Erun.stop()
    await Esys.close()
```

15. 新建执行配置 python_demo.run

【执行配置】是配置文件，可以告诉 ETest 用什么仿真环境的什么设备来执行什么程序，同时可以填写一些说明性文字，用于输出报告。还可以选择协议通道等在实时监控中显示。

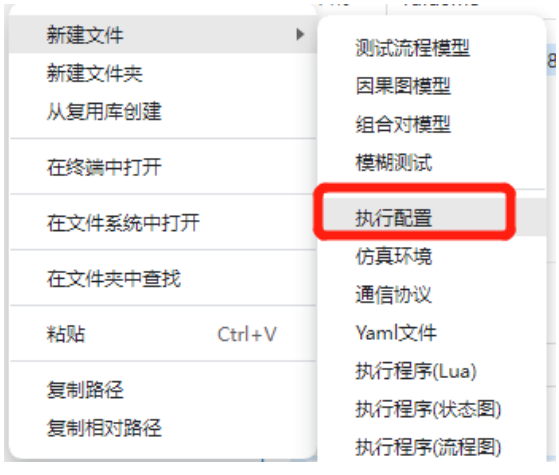

图 2-34 新建执行配置

16. 打开 python_demo.run，绑定仿真环境，执行程序

图 2-35 设置执行配置

17. 执行：点击 python_demo.run 或者 python_demo.py 脚本的右上角的三角符号执行，在下方的【终端】看执行结果。

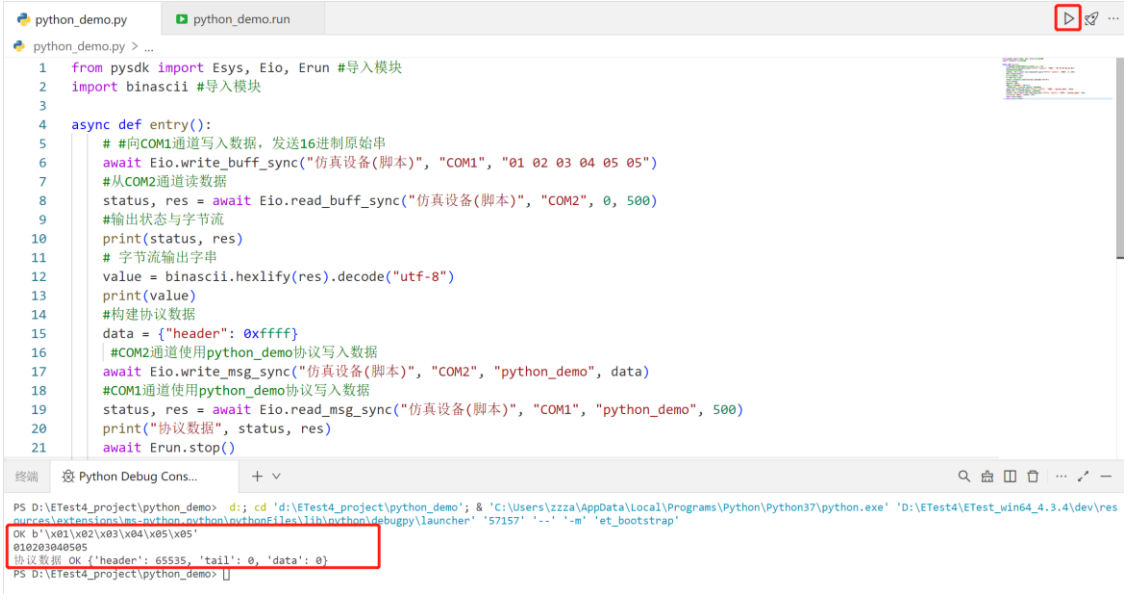

图 2-36 执行

2.3 表格实现 UDP 通道测试

表格可以被视为一系列连续动作的集合，这些动作遵循从上到下的逻辑顺序进行执行。它提供了一种图形化的方式来搭建脚本，使得用户可以直观地理解和操作复杂的流程。更为重要的是，这种表格形式的脚本能够自动转化为 Lua 脚本，极大地提高了脚本编写的灵活性和效率。通过表格与 Lua 脚本的相互转换，用户可以在图形化编辑和文本编辑之间自由切换，以满足不同场景下的需求。

1. 双击 exe 可执行程序，启动 IDE

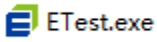

图 2-37 打开 ETest

2. 新建一个空项目

依次选择文件->新建项目->空项目->新建文件夹（表格 demo）->选择该文件夹

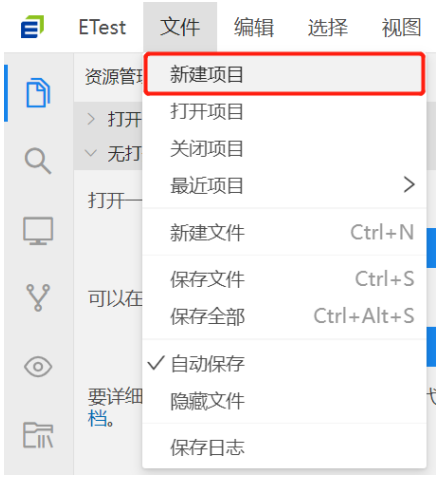

图 2-38 新建项目

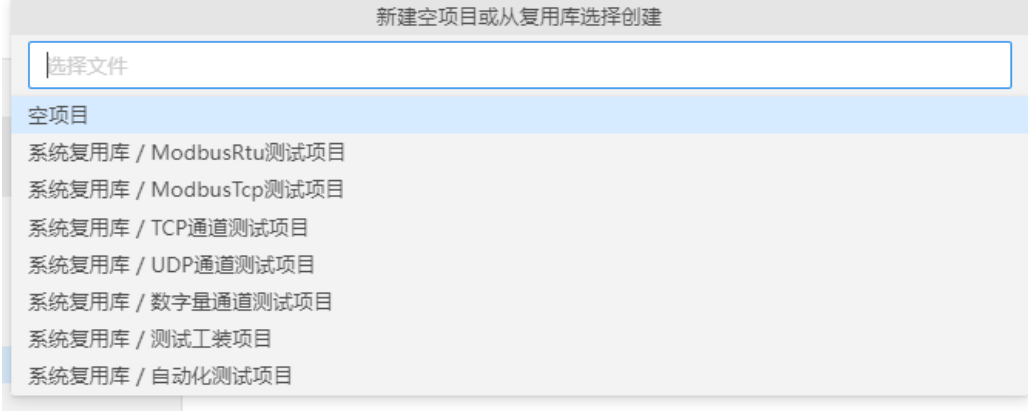

图 2-39 点击空项目

图 2-40 项目新建成功

3. 新建【仿真环境】tce.env
资源管理器->空白处右键->新建文件->仿真环境->输入仿真环境名称 tce

图 2-41 新建仿真环境

4. 添加仿真设备：打开 tce.env 文件，拖拽【仿真设备(脚本)】两次，第一次拖拽后命名为“dev1”，第二次拖拽后命名为“dev2”。

图 2-42 添加仿真设备并重命名

5. 设备说明
   - 名称：设备名称可以修改，设备名称可以是中文、英文；设备名称不可以重复。
   - 类型：添加设备后，类型可以进行修改为仿真设备或实物设备。
   - 配置：配置信息显示的是仿真设备进行通信时使用的执行器信息；实物设备不显示配置信息。
   - 配置删除：当该设备添加通道，击 X，可以删除该设备下所有通道的【配置】。
   - 删除设备：点击设备后的垃圾桶按钮，设备可以被删除；选中一个设备或多个设备后，鼠标右键菜单点击删除或操作快捷键 Delete，删除设备。

6. 添加 UDP 通道：
   拖拽两个【UDP 通道】到界面，并将它们分别放置到两个【仿真设备(脚本)】下。第一个【UDP 通道】命名为“UDP1”，保持端口号为默认设置 4000；第二个【UDP 通道】命名为“UDP2”，并将其端口号修改为 4001。两个 UDP 通道的【配置】栏绑定通道 TCPIP::UDP::0（根据系统要求填写正确的端口号或通道编号）

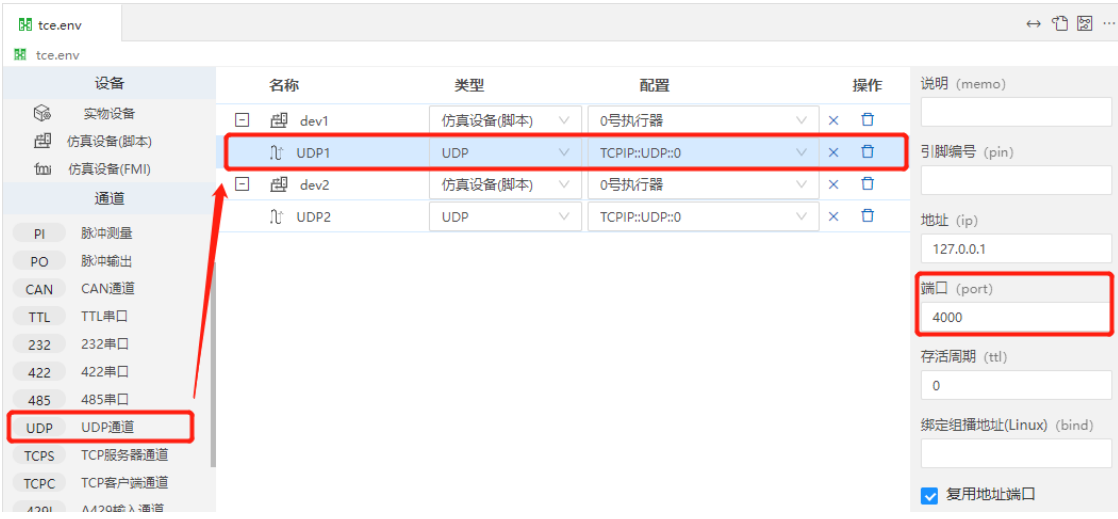

图 2-43 添加通道 UDP1

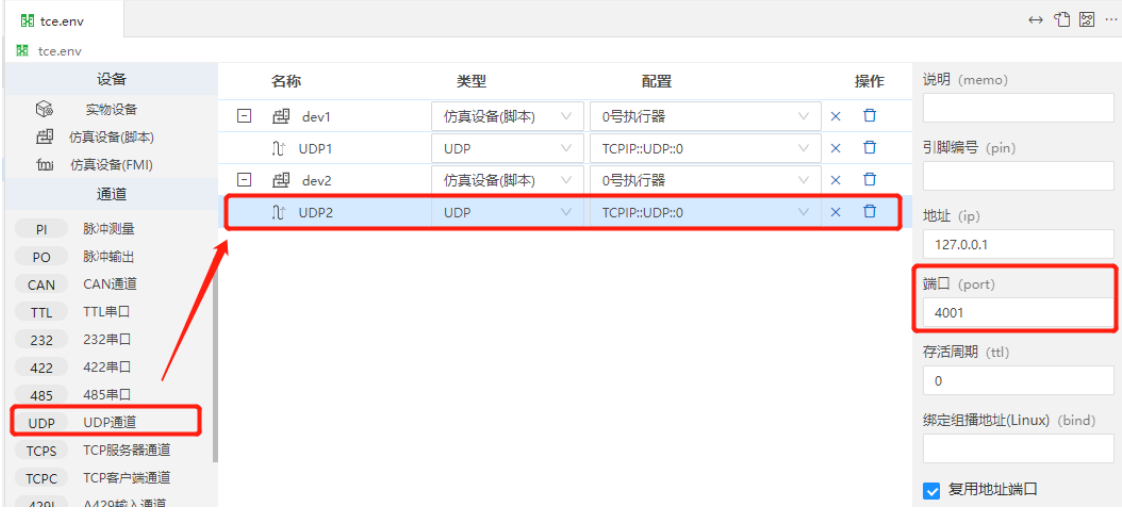

图 2-44 添加通道 UDP2，并修改端口

7. 通道说明

- 名称：通道名称可以修改，名称是由字母、数字、下划线组成，不能以数字开头，不能是关键字，不能是中文；同一个设备下的通道名称不可以重复。
- 类型：通道添加后，通道类型可以进行修改。
- 配置：下拉列表选中要绑定的通道，该操作是把对应的执行器逻辑通道绑定到板卡中的物理通道上或模拟的物理通道上，只有绑定通道后，才能实现数据通信；同时也支持自动绑定，点击配置后的↓图标，会自动进行绑定。
- 删除配置：点击 X，删除该通道的【配置】。
- 删除通道：点击通道后的垃圾桶按钮，通道可以被删除；选中一个通道或多个通

道后，鼠标右键菜单点击删除或操作快捷键 Delete，删除通道。

8. 通道的右侧面板
   - 说明，引脚编号：仅用于显示。
   - 地址，端口：IP 地址当前设备的网络标识。端口号范围 0-65535。
   - 存活周期：网络传输过程中经路由分发的次数。例，存活周期是 5 的时候。则表示只能分发五次，第六次就无效了。类型为 number。
   - 绑定组播地址(Linux)：只在 Linux 下设置。需要组播的时候，填写组播的 IP 地址。
   - 复用地址端口：boolean 类型，勾选时候，代表多个应用程序可以同时使用同一个 UDP 端口进行通信。这样多个程序可以监听同一个端口并处理不同的请求。

9. 连接拓扑
   点击【切换】，切换拓扑结构，连接两个 UDP 通道（鼠标放到 UDP1 后面空白处拖拽到 UDP2 上）

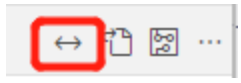

图 2-45 切换

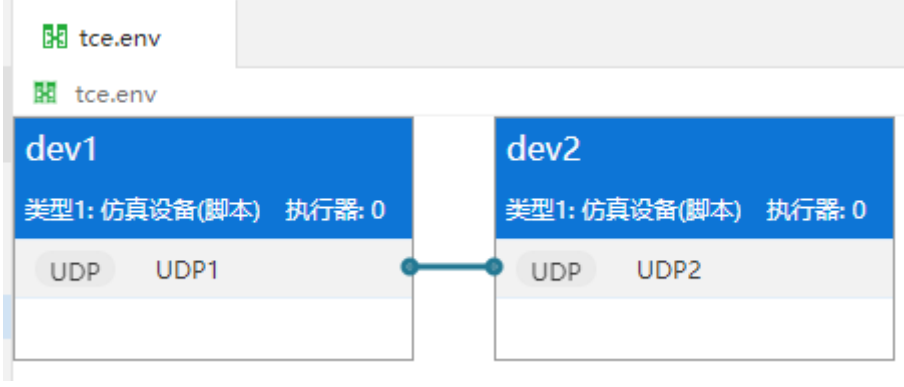

图 2-46 连接拓扑

10. 再点一下 tce.env 右上角的【切换】图标即可回到原来的界面。
11. 新建通信协议 tce.prot，协议内容默认即可。

图 2-47 新建通信协议

图 2-48 协议内容默认

12. 新建两个表格【发送.tce】【接收.tce】

图 2-49 新建表格

13. 新建执行配置 tce.run

【执行配置】是配置文件，可以告诉 ETest 用什么仿真环境的什么设备来执行什么程序，同时可以填写一些说明性文字，用于输出报告。还可以选择协议通道等在实时监控中显示。

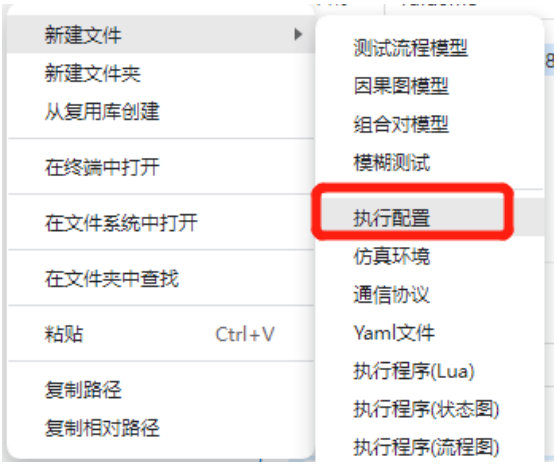

图 2-50 新建执行配置

14. 打开 tce.run，绑定仿真环境，执行程序

图 2-51 设置执行配置

15. 编辑【发送.tce】

- 点击右侧【全局配置】，绑定设备【dev1】，该设备是为了后续绑定对应设备的通道用的。

图 2-52 绑定设备

- 点击左侧动作栏的【输出】，拖拽【报文发送】到动作界面空白处。

图 2-53 添加动作报文发送

- 选中报文发送，点击右侧动作设置，编辑内容如下。这里的通道就是刚才在全局配置里绑定设备后才能选的。动作设置后看到【脚本】立刻显示了对应的 Lua 脚本。

图 2-54 设置报文发送

- 动作【报文发送】：将报文数据{data=11，header=0x55AA，tail=0}（没在报文数据输入框赋值的协议字段 header，tail 采用协议 tce.prot 里自动值），用协议 tce.prot 打包后，发送给通道 UDP1

图 2-55 协议字段的自动值

16. 编辑【接收.tce】

- 点击右侧【全局配置】，绑定设备【dev2】

图 2-56 绑定设备

- 点击左侧动作栏的【常用动作】，拖拽【声明】到动作界面空白处，并点击【动作设置】，输入标题，点击加号添加变量。

图 2-57 添加声明并设置

- 点击【输入】，拖拽【报文接收】到界面空白处，并编辑【动作设置】

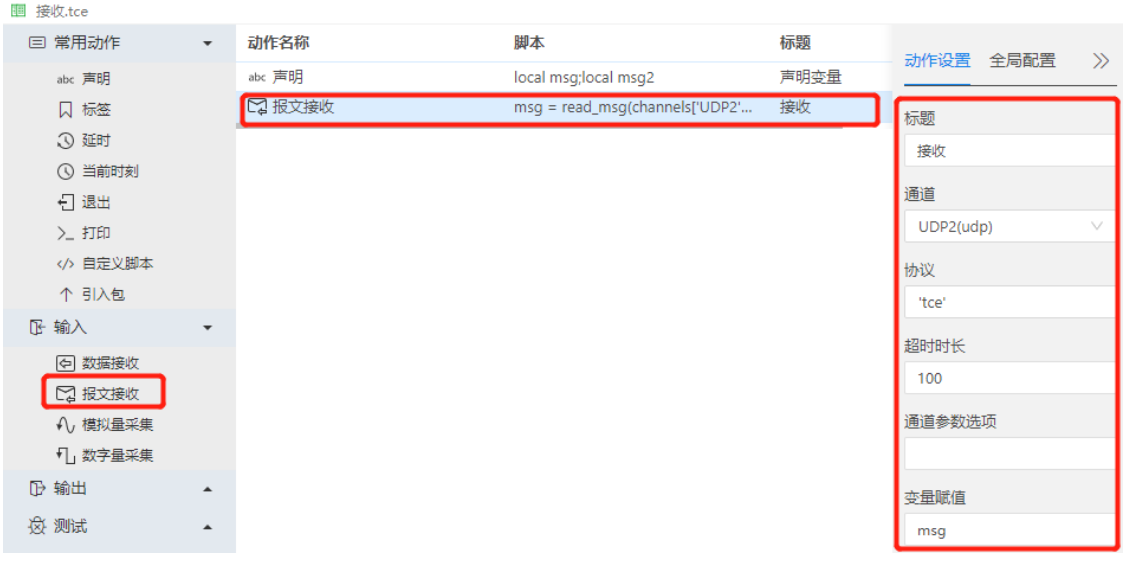

图 2-58 报文接收

- 动作【报文接收】：读取通道 UDP2 里的数据流，并用协议 tce.prot 解包后报文数据，赋值给变量 msg。超过 100ms 就不再读取。

- 拖拽【常用动作】的【退出】到界面空白处

图 2-59 退出

- 添加【延时】到【报文接收】前，延时时机为 100

图 2-60 延时

17. 执行：点击【发送.tce】或【接收.tce】或【tce.run】右上角的三角符号执行，在下方的【执行输出】看执行结果。可以看到发送 data 值为 11，接收后 data 也是 11.

图 2-61 执行

2.4 状态图

说明：本示例用状态图完成电灯开关的功能。在状态“开”点击确认，转化为状态“关”；在状态“关”点击确认，转化为状态“开”，开关加起来的次数大于 5 时，退出程序。开关的动作通过 ask 实时交互来完成。

1. 双击 exe 可执行程序，启动 IDE

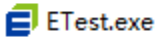

图 2-62 打开 ETest

2. 新建一个空项目
   依次选择文件->新建项目->空项目->新建文件夹（状态图 demo）->选择该文件夹

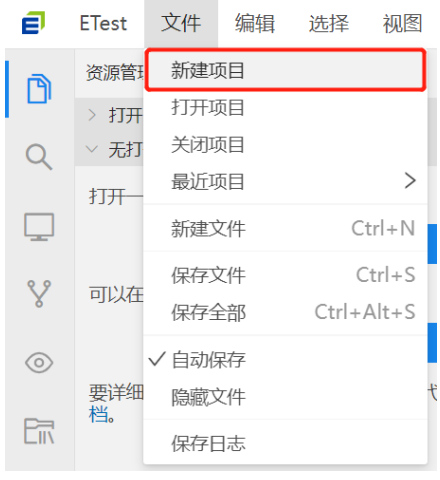

图 2-63 新建项目

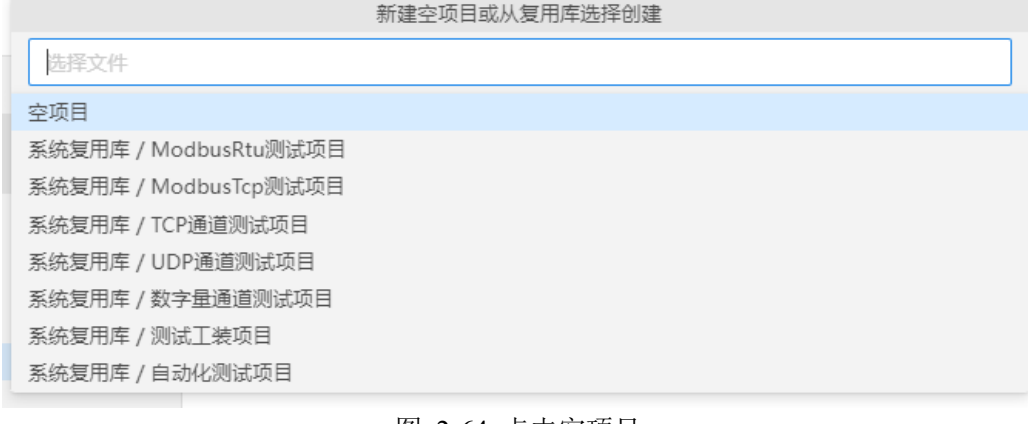

图 2-64 点击空项目

图 2-65 项目新建成功

3. 新建状态图【test.sm】

图 2-66 新建状态图

4. 状态变量

(1) 状态变量是在【状态图】中定义和使用的 local 变量。

(2) result = estate.get_value(key)

- 将状态变量 key 的值给变量 result

(3) estate.set_value(key, value)

- 设置状态变量 key 的值为 value

5. 事件：当检测到有事件抛出（estate.throw）时，就会立刻捕获（catch）到该事件，返回 true

(1) estate.throw("事件名")

- 功能：抛出事件
- 用法：将 estate.throw("事件名")写在 lua 脚本中。

(2) catch(事件名)

- 功能：当捕获到该【事件名】的时候，会返回一个 true，没检测到返回 false。
- 用法：作为状态迁移的条件，将 catch(事件名)写在箭头线上。

6. 新建并编辑【on.lua】

```
function entry()
  print(estate.current())  --打印出当前状态名称“开灯”
  local val = ask("ok", {title='', msg='关', timeout=0}) --暂定执行等待用户确认，确认后继续执行后续程序，用户界面提示信息为'开'
  local lc=estate.get_value('count')  --取得状态变量 count 的值，赋值给本地变量 lc
  lc=lc+1 --lc 的值加一
  estate.set_value('count',lc) --将 lc 的值赋值给状态变量 count
  estate.throw('off') --抛出事件 off
end
```

7. 新建并编辑【off.lua】

```
function entry()
  print(estate.current())  --打印出当前状态名称“关灯”
  local val = ask("ok", {title='', msg='开', timeout=0}) --暂定执行等待用户确认，确认后继续执行后续程序，用户界面提示信息为'关'
  local lc=estate.get_value('count')  --取得状态变量 count 的值，赋值给本地变量 lc
  lc=lc+1 --lc 的值加一
  estate.set_value('count',lc) --将 lc 的值赋值给状态变量 count
  estate.throw('on') --抛出事件 on
end
```

8. 状态图的工具栏

- 开始：开始启动，开始图标只能指向下一级进行连接，一个状态图只能有一个开始。
- 结束：执行结束，结束图标只能被上一级连接。
- 状态：对象在其生命周期中的一种状况，处于某个特定状态中的对象必然会满足某些条件、执行某些动作或者是等待某些事件。

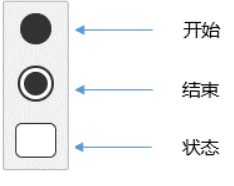

图 2-67 工具栏

9. 编辑【test.sm】

- 拖拽一个开始，两个状态，两个结束

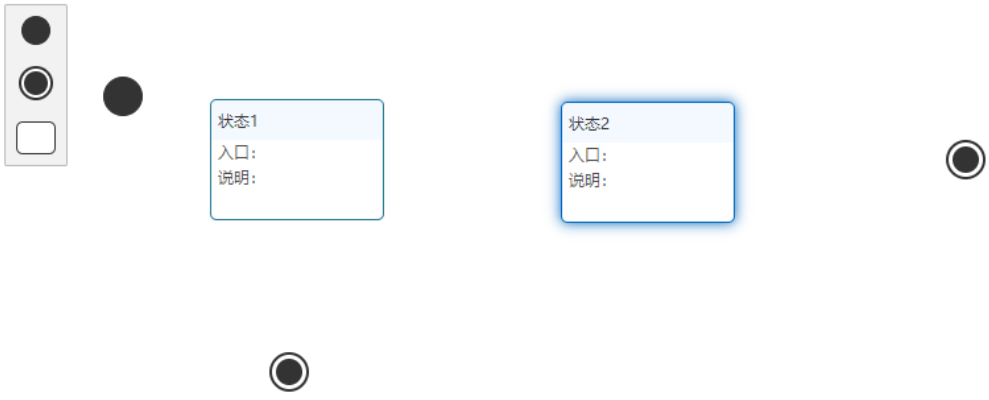

图 2-68 添加元素

- 连线：连线箭头指明了状态迁移的方向。鼠标放到【开始】边缘，当鼠标变成十字的时候，按住并拖动鼠标到【状态 1】的边缘，松手即可连线成功。同理，添加其他连线。

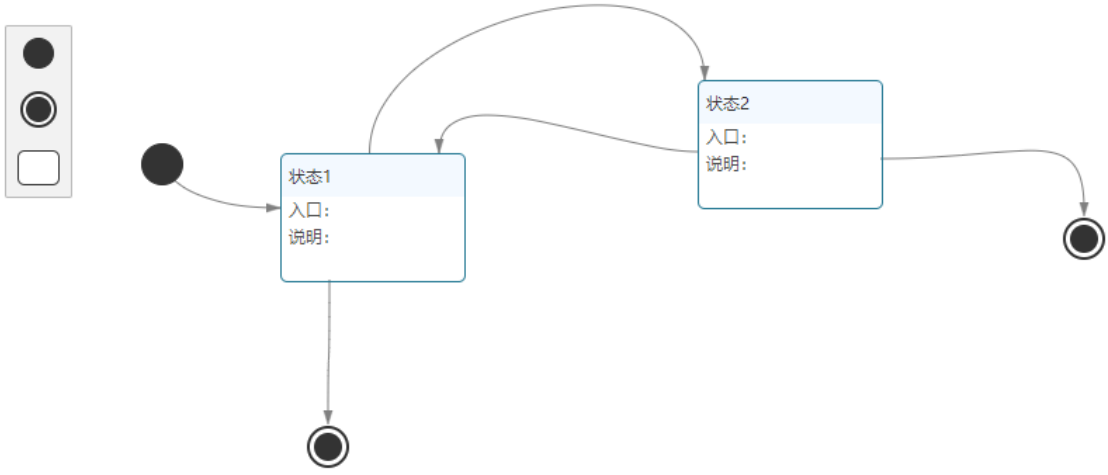

图 2-69 连线

- 添加状态变量：点击加号，添加状态变量 count，初期值设置为 0

图 2-70 状态变量

- 状态 1 的属性设置：点击【状态 1】，点击右侧【属性设置】修改名称，绑定入口程序

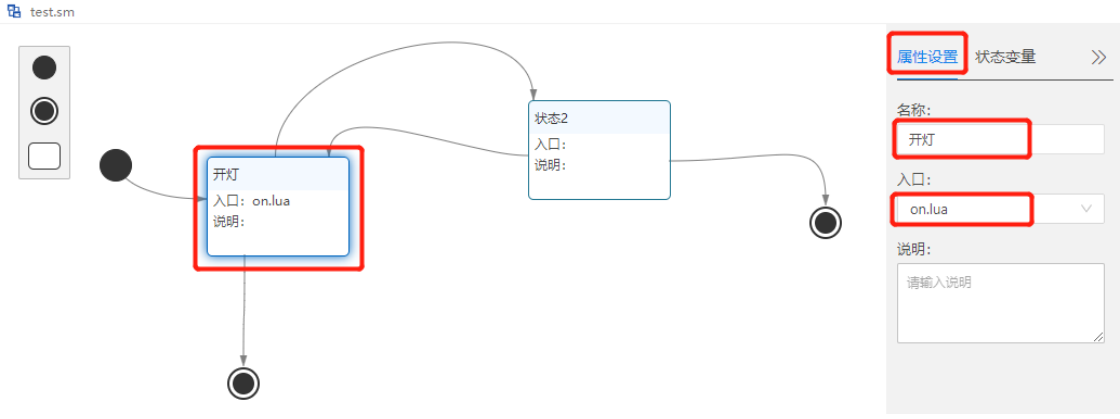

图 2-71 状态 1 的属性设置

- 状态 2 的属性设置：点击【状态 2】，点击右侧【属性设置】修改名称，绑定入口程序

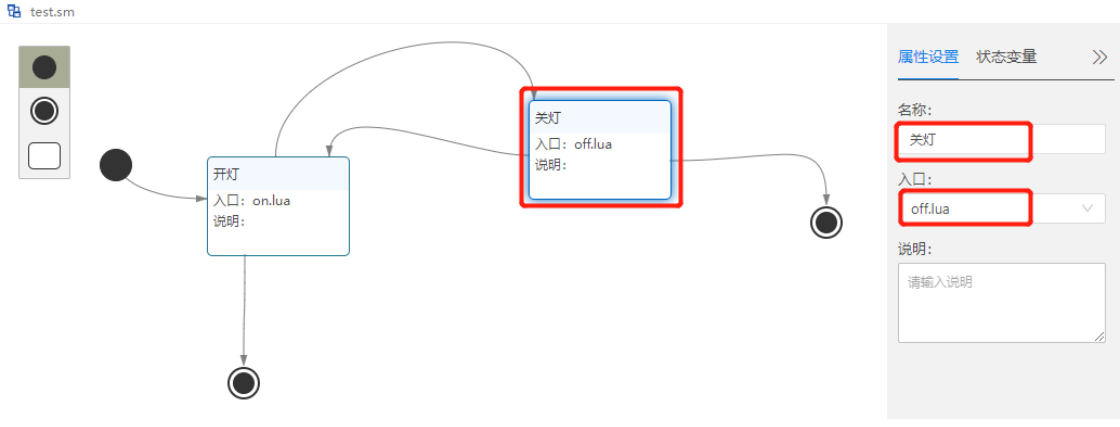

图 2-72 状态 2 的属性设置

- 添加状态迁移条件：双击箭头线添加变迁条件，中括号[]里放置条件表达式，大括号{}里放的是执行语句(四则运算，赋值语句)，注意执行语句后面要加分号【;】可以添加多个执行语句。

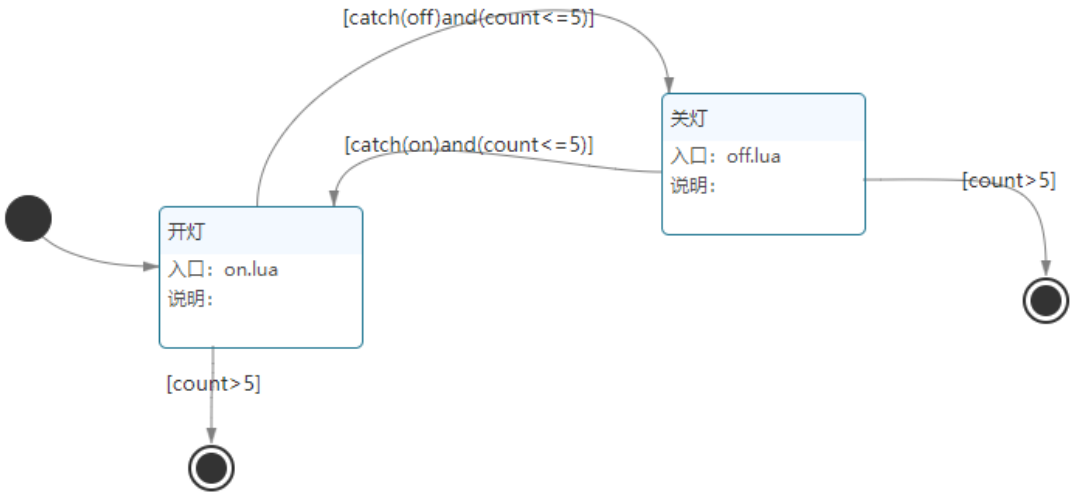

图 2-73 添加迁移条件

10. 状态迁移的触发条件
在状态入口绑定文件中，执行到 estate.set_value(key, value)或者 estate.throw("事件名")时，才会检查箭头线的事件表达式是否成立。“条件表达式”成立，则执行箭头线上的“执行语句”，并按照箭头线的方向迁移到下一个状态；“条件表达式”不成立，则停留在当前状态，继续执行后续代码。

11. 执行：点击状态图【test.sm】右上角的执行图标（三角形）。右侧出现【实时交互】，点击确认即可。

图 2-74 实时交互

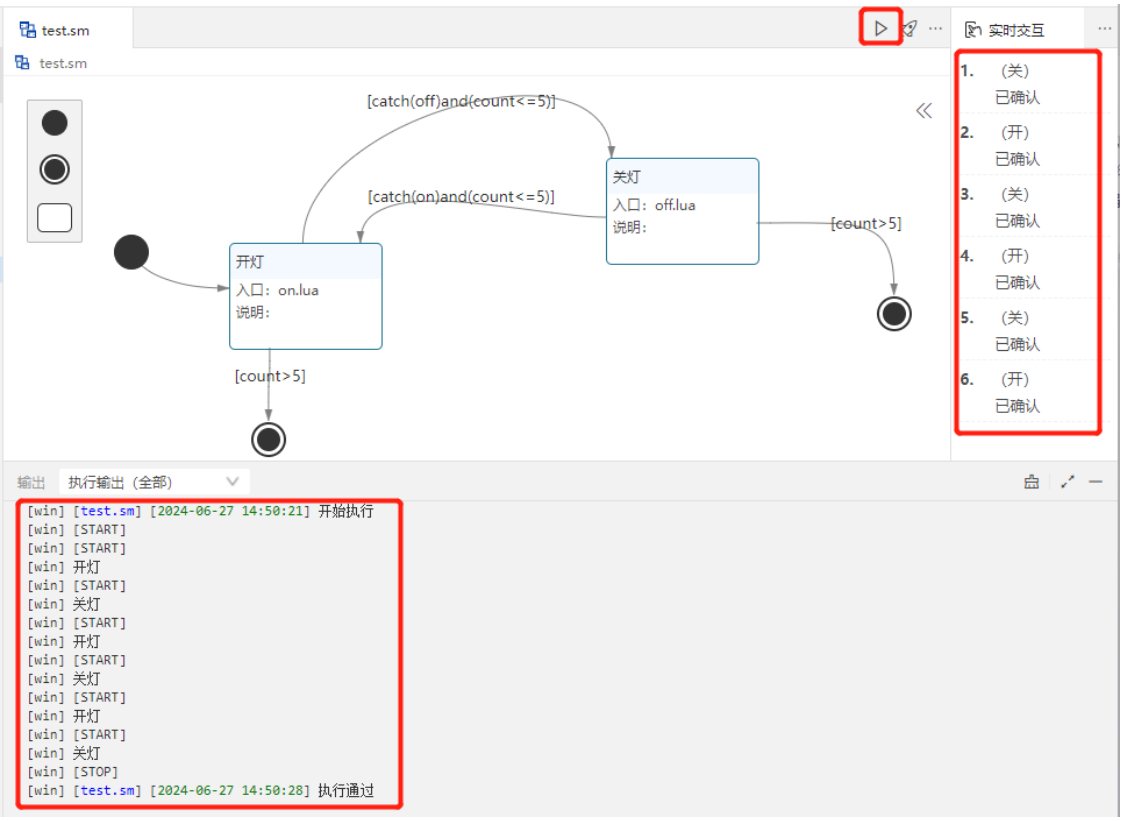
test.sm
test.sm
[catch(off)and(count<=5)]
[catch(on)and(count<=5)]
关灯
入口：off.lua
说明：
[count>5]
开灯
入口：on.lua
说明：
[count>5]
实时交互
1. (关)
已确认
2. (开)
已确认
3. (关)
已确认
4. (开)
已确认
5. (关)
已确认
6. (开)
已确认
输出
执行输出（全部）
[win] [test.sm] [2024-06-27 14:50:21] 开始执行
[win] [START]
[win] [START]
[win] 开灯
[win] [START]
[win] 关灯
[win] [START]
[win] 开灯
[win] [START]
[win] 关灯
[win] [START]
[win] 开灯
[win] [START]
[win] 关灯
[win] [STOP]
[win] [test.sm] [2024-06-27 14:50:28] 执行通过

图 2-75 执行

第3章　工装开发

1. 双击 exe 可执行程序，启动 IDE

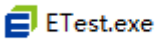

图 3-1 打开 ETest

2. 新建一个空项目：依次选择文件->新建项目->空项目->选择一个空文件夹

图 3-2 新建项目

图 3-3 新建成功

3. 新建【仿真环境】test.env

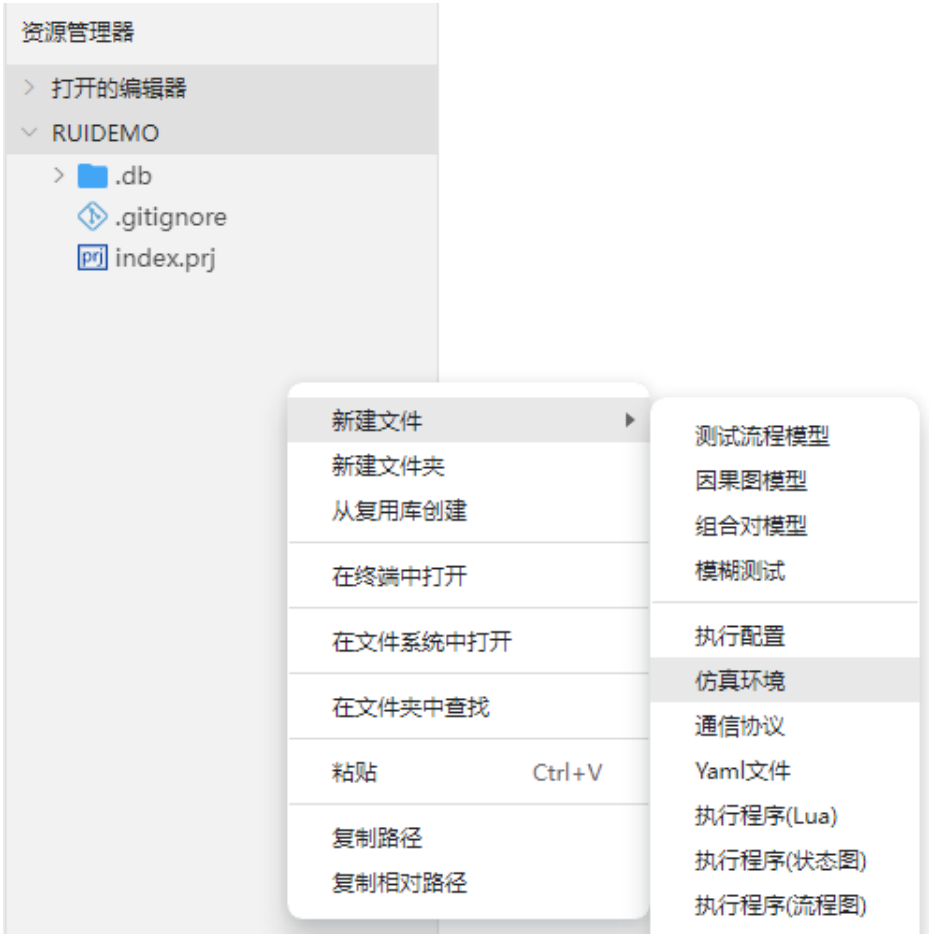

图 3-4 新建仿真环境

4. 拖拽【仿真设备（脚本）】到编辑器

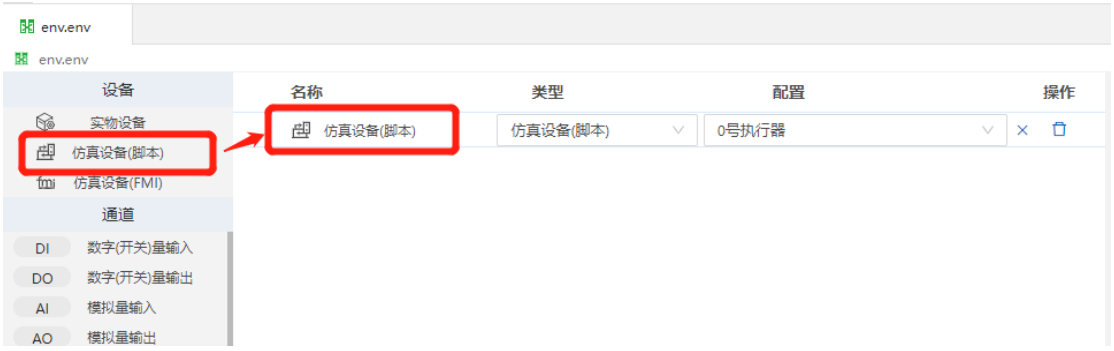

图 3-5 添加仿真设备

5. 拖拽【UDP 通道】到【仿真设备（脚本）】下

图 3-6 添加通道

6. 新建 test.yml 文件(用于填写网络变量的声明)，并编辑

图 3-7 添加 yaml 文件

```
demo: "NG"
```

7. 新建 test.lua，并编辑

```
function entry()
end
function Send()
  print("成功~")
end
```

图 3-8 新建 lua 脚本

8. 新建执行配置 test.run

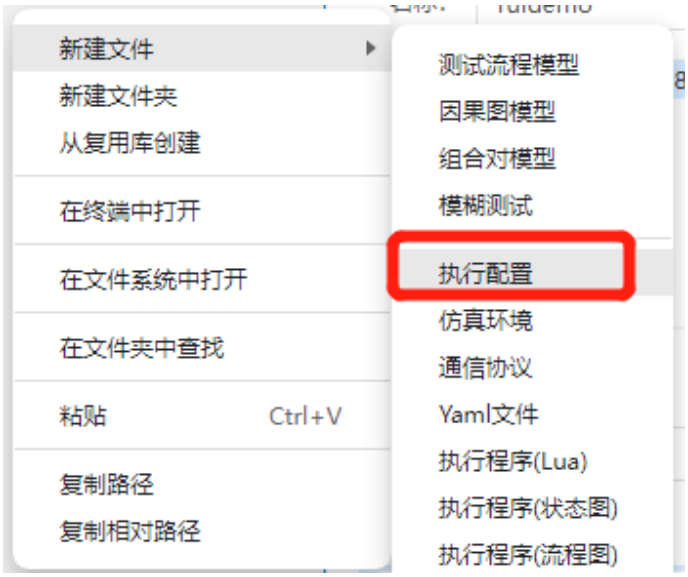

图 3-9 新建执行配置

9. 绑定仿真环境，网络变量，执行程序

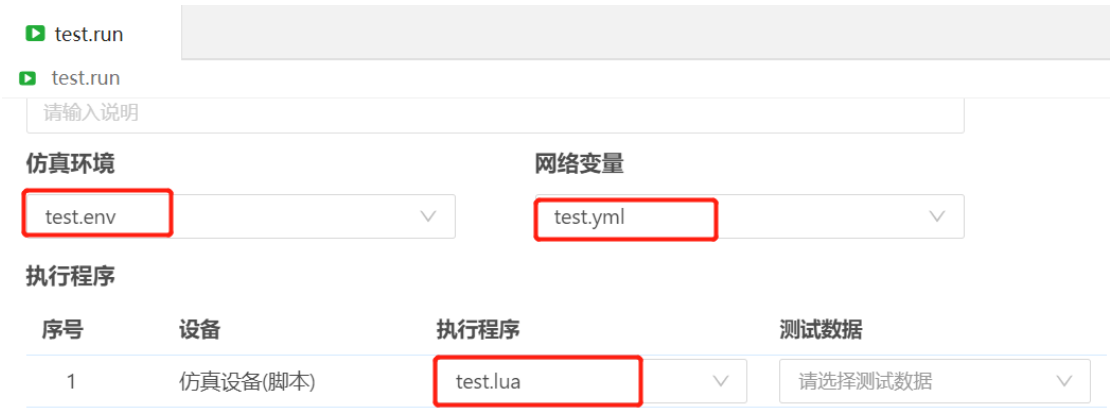

图 3-10 设置执行配置

10. 点击 UI 设计器->页面右键->重命名->demo

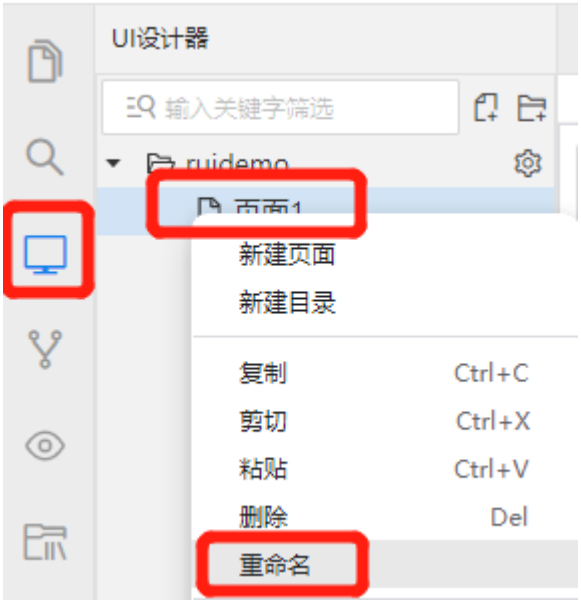

图 3-11 重命名

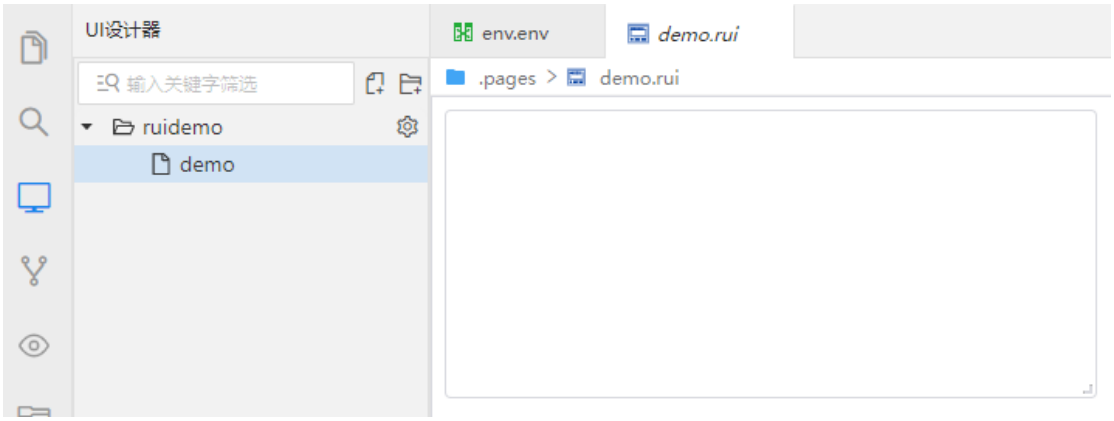

图 3-12 demo.rui

11. 拖拽输出组件【按钮】到面板上

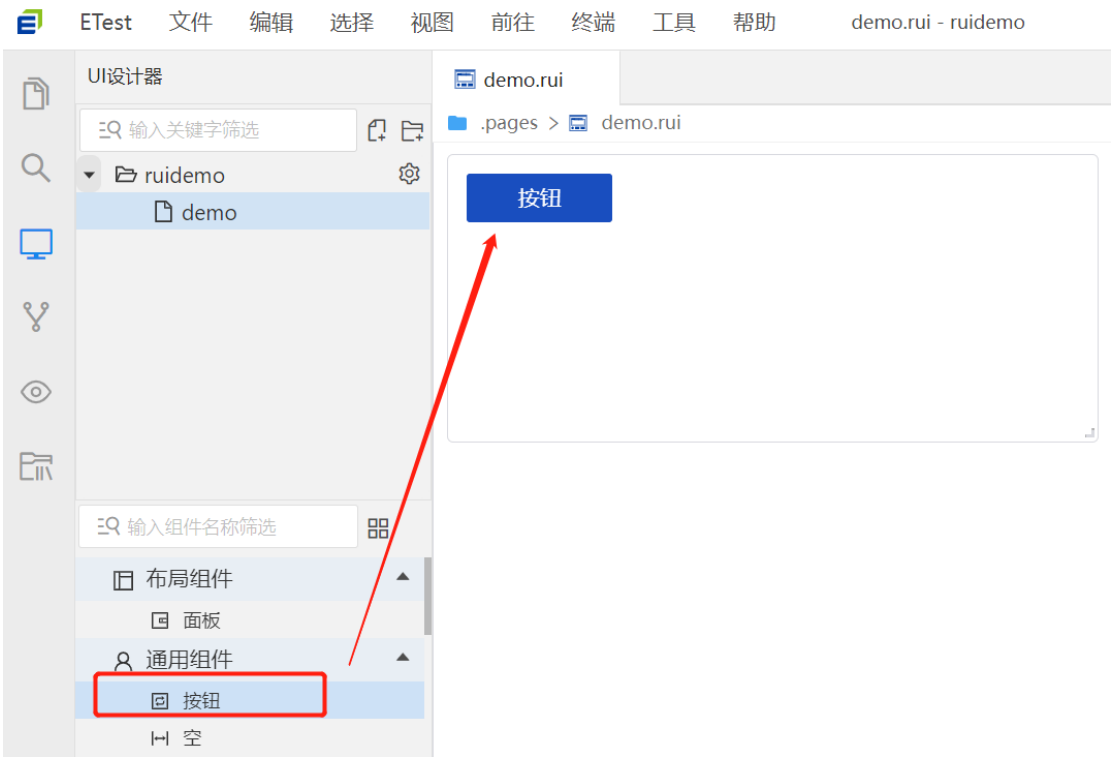

图 3-13 添加组件

12. 点击空白处，【执行配置】选择 test.run，网络变量会自动选中。

图 3-14 设置执行配置

13. 设置组件按钮

(1) 按钮文本：按钮
(2) 绑定变量：$.demo
(3) 事件类型：自定义代码
(4) 参数：(api) => {api.set_value('$.demo', 'OK');api.cmd("Send",[]) }
上述代码的含义：当点击按钮的时候，将网络变量 demo 设置成 OK，并调用 Send 函数。

图 3-15 按钮的属性面板

14. 点击页面右上角【打开调试模式】进行调试（点击按钮变 OK，并在执行输出【成功~】）,确认无误后进行下一步。点返回到编辑模式。

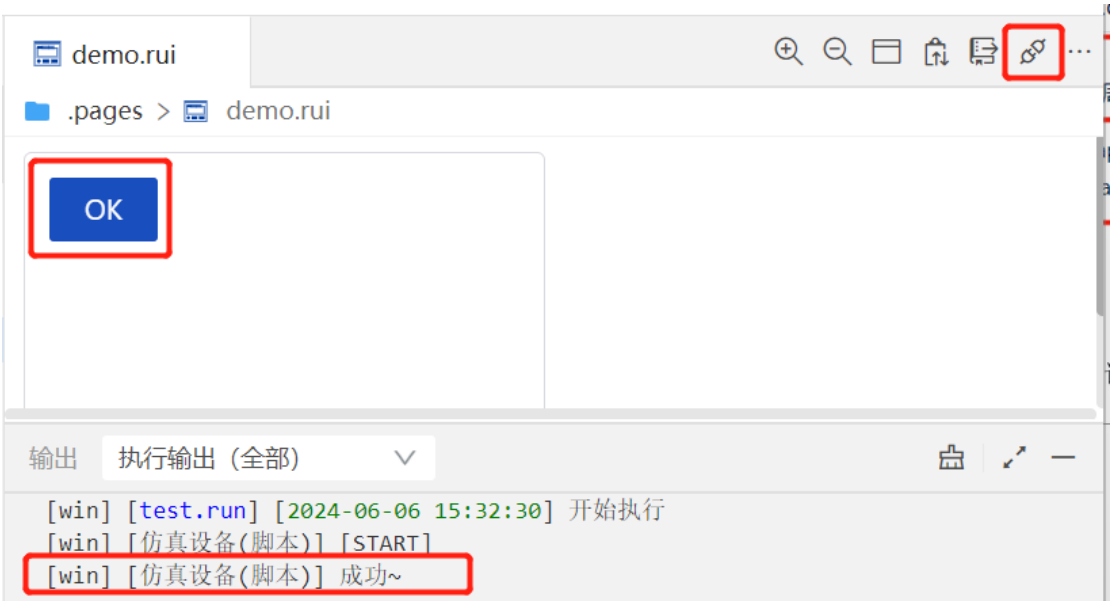

图 3-16 打开调试模式

15. 打开界面配置，并配置

(1) 通用配置-输出目录：点击选择打包输出的路径
(2) 菜单配置-首页：点击选择 demo 页面
(3) 菜单配置-界面：点击选择 demo 页面
(4) 菜单配置-执行配置：空（使用界面绑定的*.run 文件）

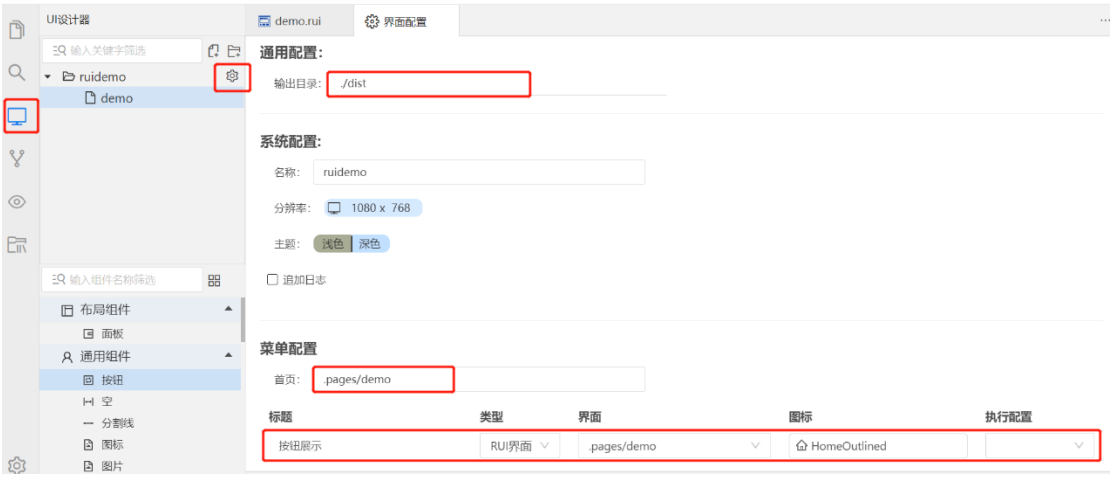

图 3-17 界面配置

16. 打包输出

点击“文件”-“打包输出”，开始打包。

图 3-18 打包输出

17. 打包成功后提示“打包成功”，生成 et_player 文件夹。

图 3-19 打包成功

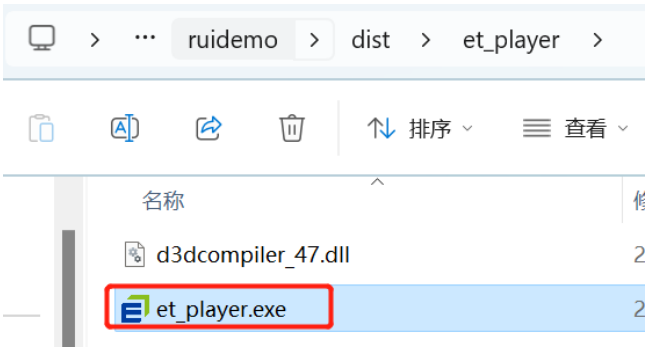

图 3-20 打包输出后的文件在项目 ruidemo 的 dist/et_player

18. 启动 et_player

找到打包输出的文件夹，进入 et_player 文件夹，双击“et_player.exe”，启动程序。

启动“et_player.exe”的前提条件：启动 ETest，或者将 ETest 安装路径下的 bin 文件夹拷贝到 et_player 文件夹下。

图 3-21 启动 et_player.exe

19. 点击按钮【NG】，按钮变【OK】，并在 ETest 的执行输出显示【成功~】

图 3-22 执行