文章目录

一、前言

嗨，大家好，我是新发。

最近在搞服务端 `Skynet` 框架，今天我想写一下 `Unity` 通过 `sproto` 协议与 `Skynet` 服务端通信的流程，画成图是这样子：

话不多说，我们开始吧~

二、搭建Skynet服务端

关于搭建 `Skynet` 服务端，我前两篇文章写了教程，建议先阅读我之前这两篇文章：

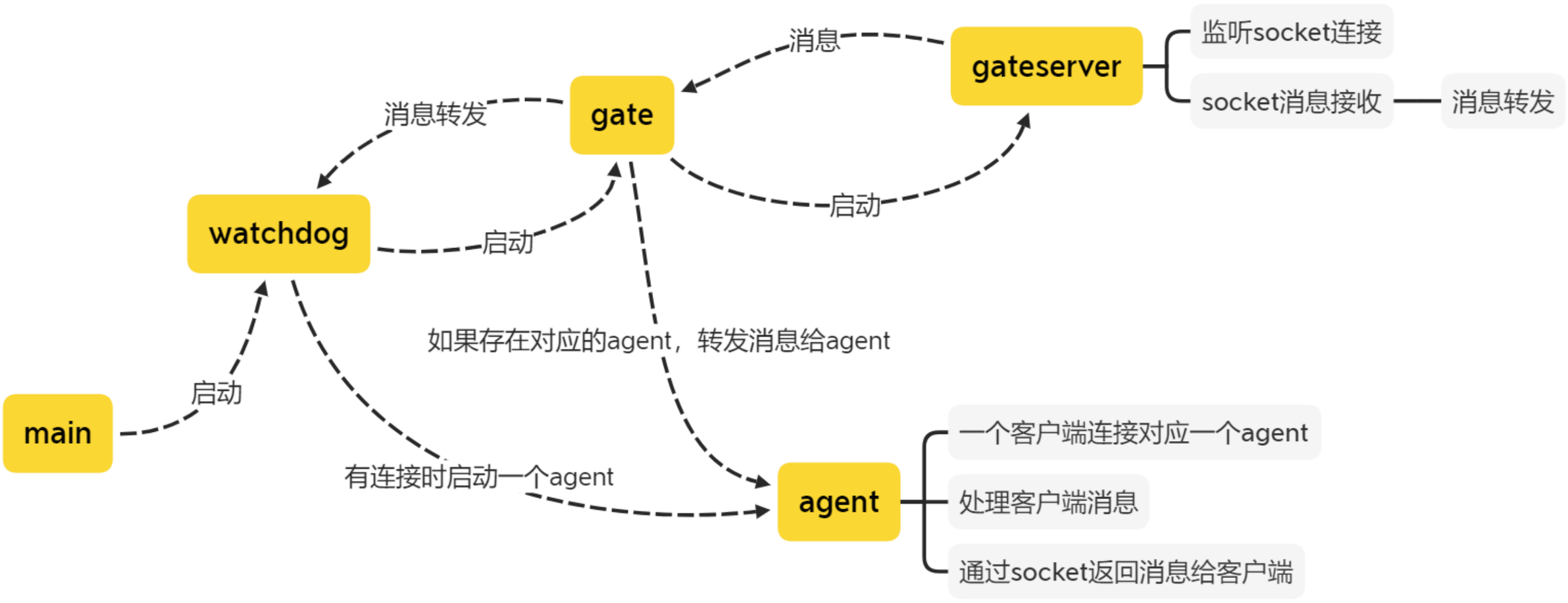
【游戏开发实战】手把手教你从零跑一个Skynet，详细教程，含案例讲解（服务端 | Skynet | Ubuntu）

【游戏开发实战】手把手教你在Windows上通过WSL运行Skynet，不用安装虚拟机，方便快捷（WSL | Linux | Ubuntu | Skynet | VSCode）

本文我就不过多赘述，搭建好环境后，运行 `Skynet` ，效果如下：

```
linxinfa@linxinfapc:/mnt/e/wsl/skynet$ . start.sh
[:01000002] LAUNCH snlua bootstrap
[:01000003] LAUNCH snlua launcher
[:01000004] LAUNCH snlua cmaster
[:01000004] master listen socket 0.0.0.0:2013
[:01000005] LAUNCH snlua cslave
[:01000005] slave connect to master 127.0.0.1:2013
[:01000006] LAUNCH harbor 1 16777221
[:01000004] connect from 127.0.0.1:16306 4
[:01000004] Harbor 1 (fd=4) report 127.0.0.1:2526
[:01000005] Waiting for 0 harbors
[:01000005] Shakehand ready
[:01000007] LAUNCH snlua datacenterd
[:01000008] LAUNCH snlua service_mgr
[:01000009] LAUNCH snlua main
[:01000009] Server start
[:01000009] hello skynet
[:0100000a] LAUNCH snlua protoloader
[:0100000b] LAUNCH snlua console
[:0100000c] LAUNCH snlua debug_console 8000
[:0100000c] Start debug console at 127.0.0.1:8000
[:0100000d] LAUNCH snlua simpledb
[:0100000e] LAUNCH snlua watchdog
[:0100000f] LAUNCH snlua gate
[:0100000f] Listen on 0.0.0.0:8888
[:01000009] Watchdog listen on 8888
[:01000009] KILL self
[:01000002] KILL self
```

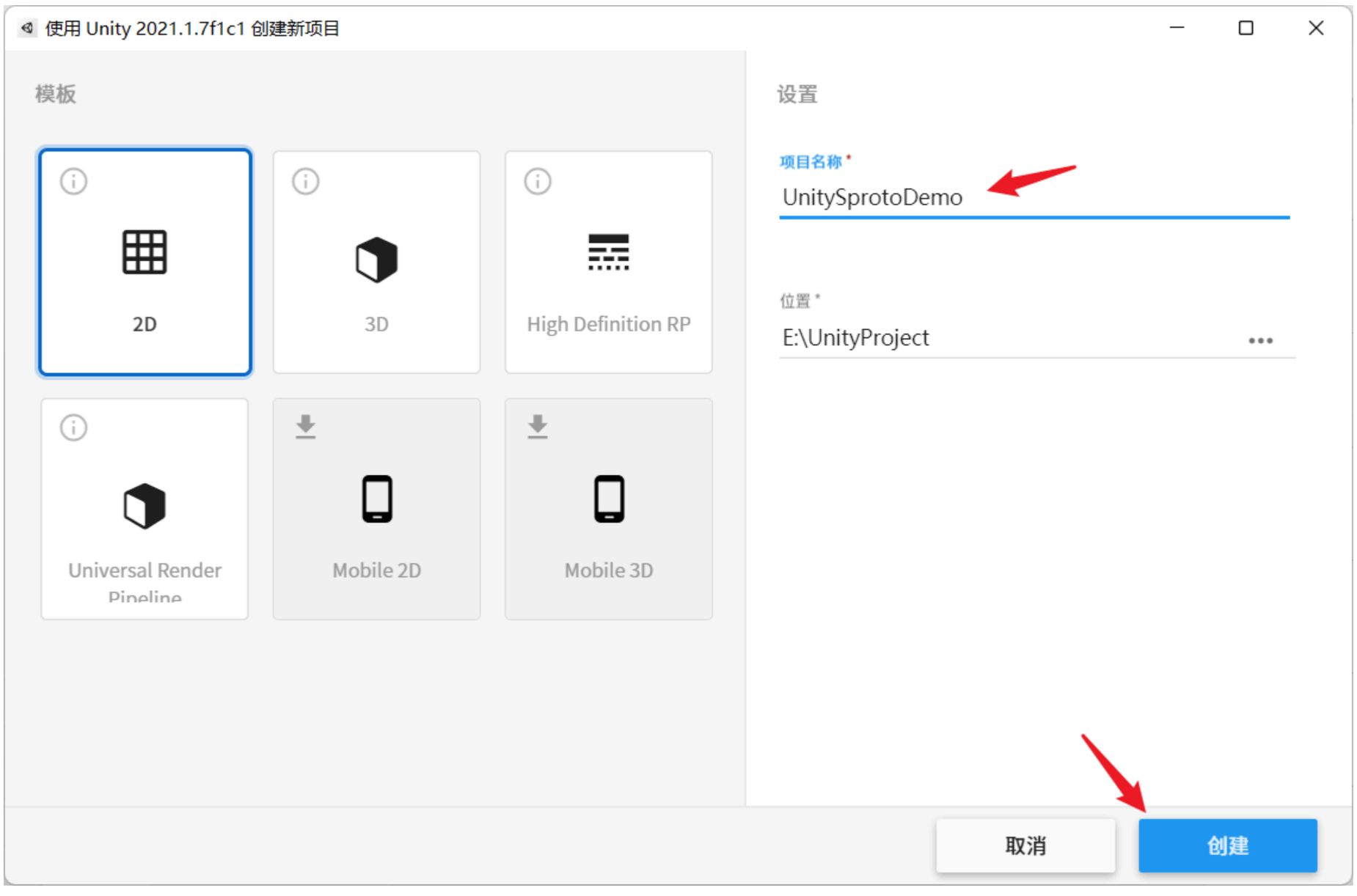
服务端模块架构如下：



三、Unity客户端

1、创建Unity工程

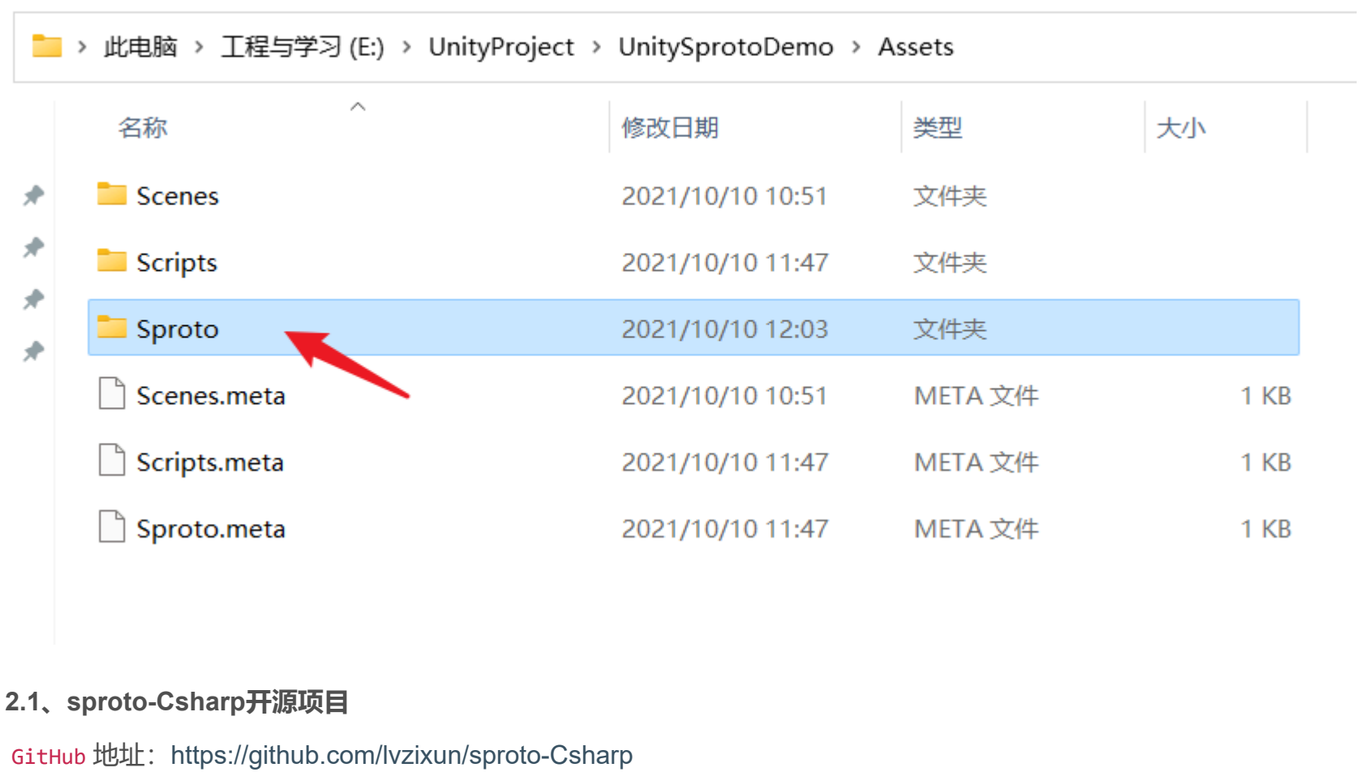
我使用的 `Unity` 版本为 `Unity 2021.1.7f1c1` ，因为这里我只演示客户端与服务端通过 `sproto` 协议通信的流程，不需要使用 `3D` 相关的功能，所以我创建的是一个 `2D` 模板的工程，工程名叫 `UnitySprotoDemo` ，如下：



2、导入开源项目

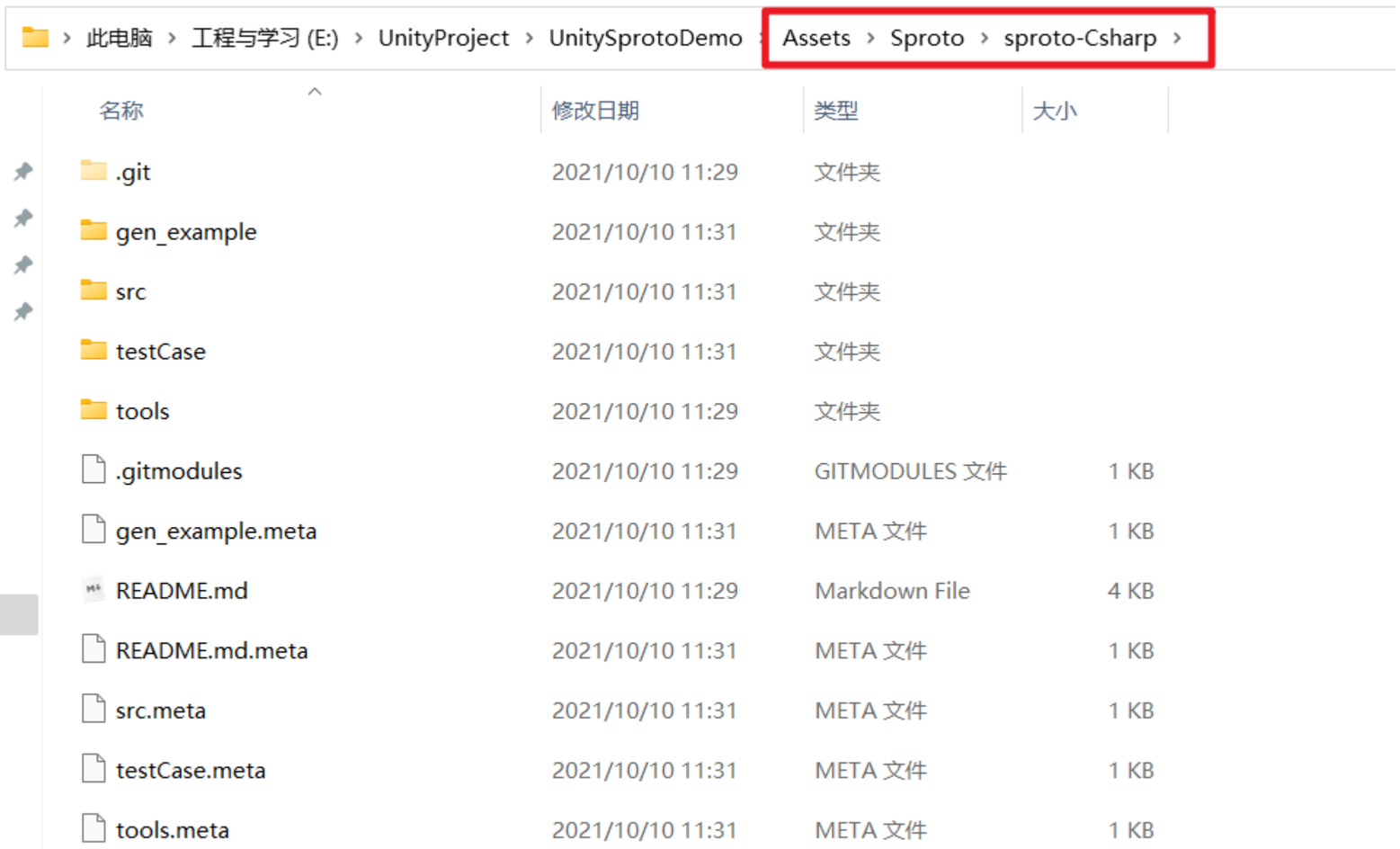
我们要在 Unity 中使用 sproto 协议进行通信，那就需要一套 sproto 协议的 C# 实现和工具。这种嘛，首选在 GitHub 中搜索相关的开源项目，避免不必要的重复造轮子。
关于 GitHub 的使用，我之前写过一篇教程，感兴趣的同学可以看看：《GitHub使用教程与常见问题解决——上传本地工程到GitHub仓库》

我找到了一套可以在 Unity 中使用的 sproto 的 C# 实现与工具，我们先在 Unity 工程的 Assets 文件夹中新建一个 sproto 文件夹，用于存放从 GitHub 中下载下来的 sproto 开源项目，

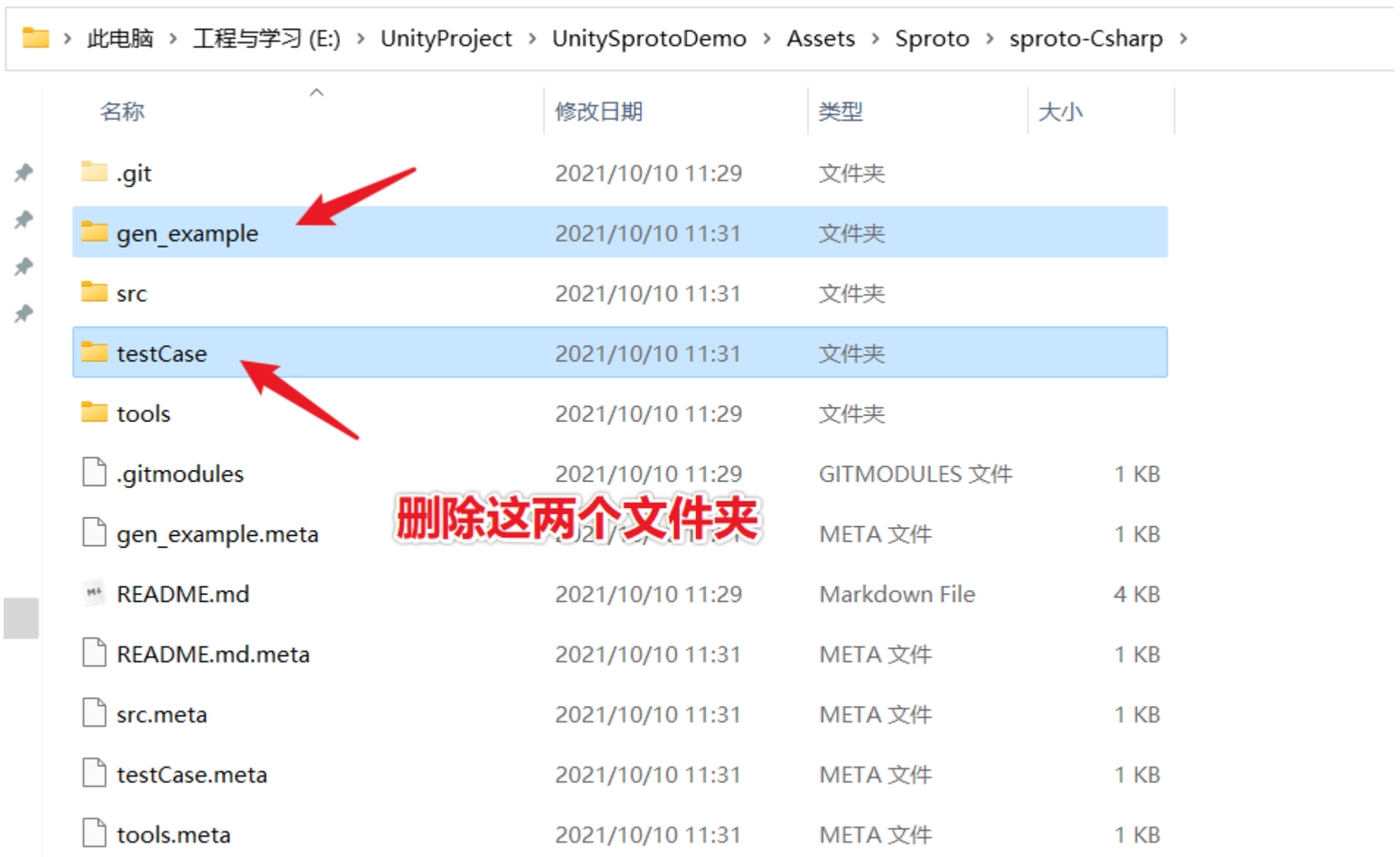


2.1、sproto-Csharp开源项目

GitHub 地址：https://github.com/lvzixun/sproto-Csharp
sproto-Csharp 是 sproto 的纯 C# 实现。
我们先把它下载下来，放到 Assets/Sproto/sproto-Csharp 文件夹中，如下：



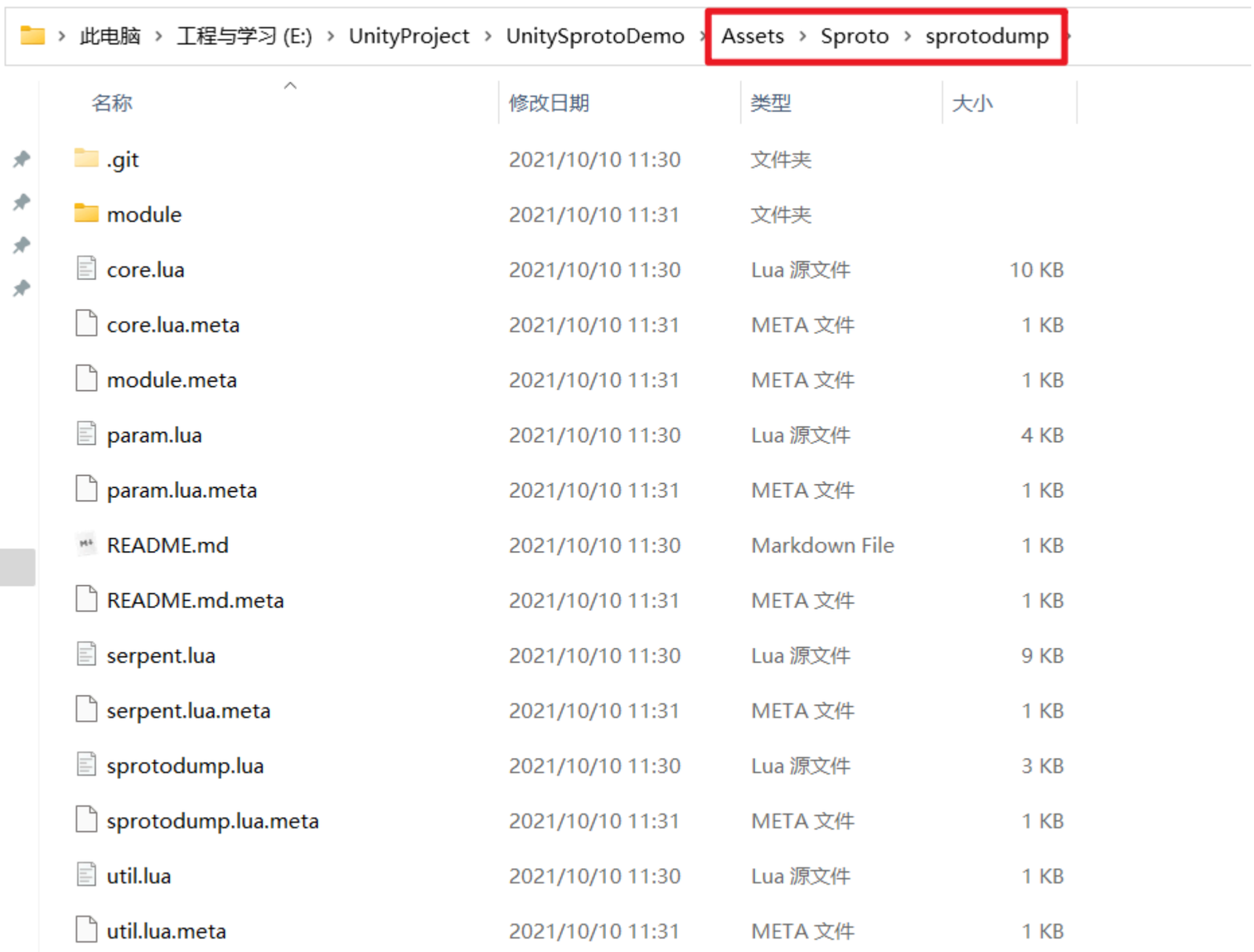
有一些测试用的代码，我们可以把它删掉，如下：



2.2、sprotodump开源项目

GitHub 地址：https://hub.fastgit.org/lvzixun/sprotodump
sprotodump 是将 .sproto 文件转为 .cs 、 .spb 、 .spb 、 .go 、 .md 、 .lua 等文件的工具，下文我会讲如何使用这个工具。

我们先把它下载下来，放到 `Assets/Sproto/sprotodump` 文件夹中，如下：

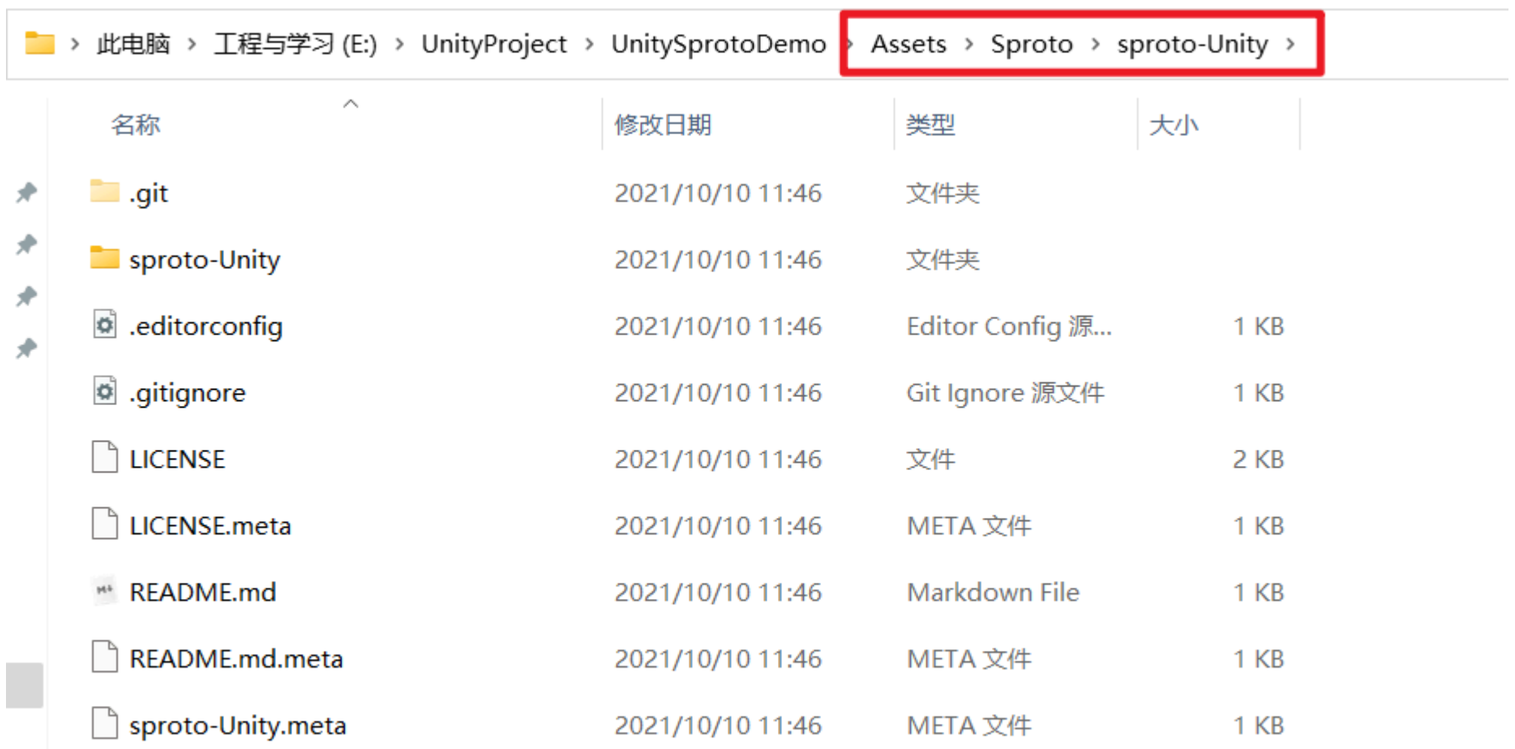


2.3、sproto-Unity开源项目

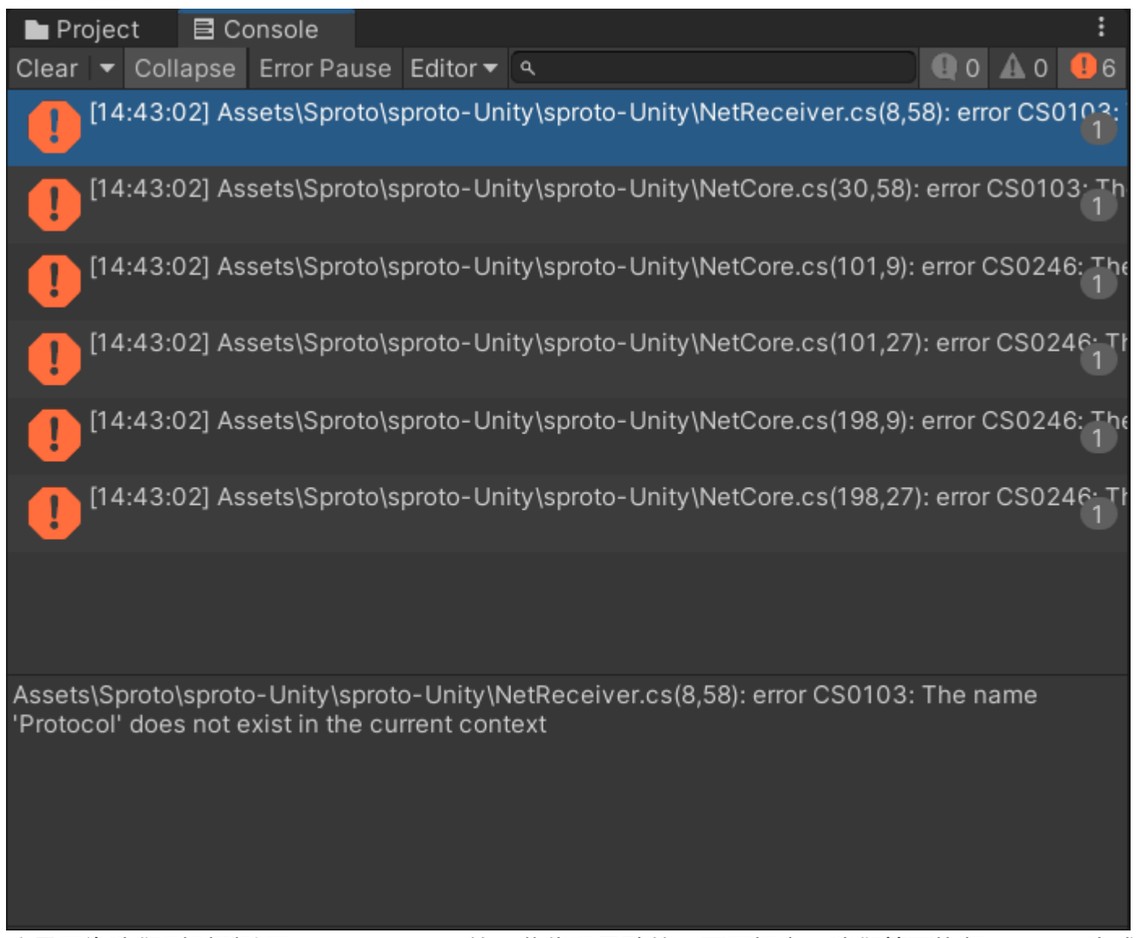
`GitHub` 地址：https://github.com/m2q1n9/sproto-Unity

`sproto-Unity` 封装了三个类： `NetCore` 、 `NetSender` 、 `NetReceiver` ，下文我会讲下如何使用。

我们先把它下载下来，放到 `Assets/Sproto/sproto-Unity` 文件夹中，如下：



此时工程会报错，



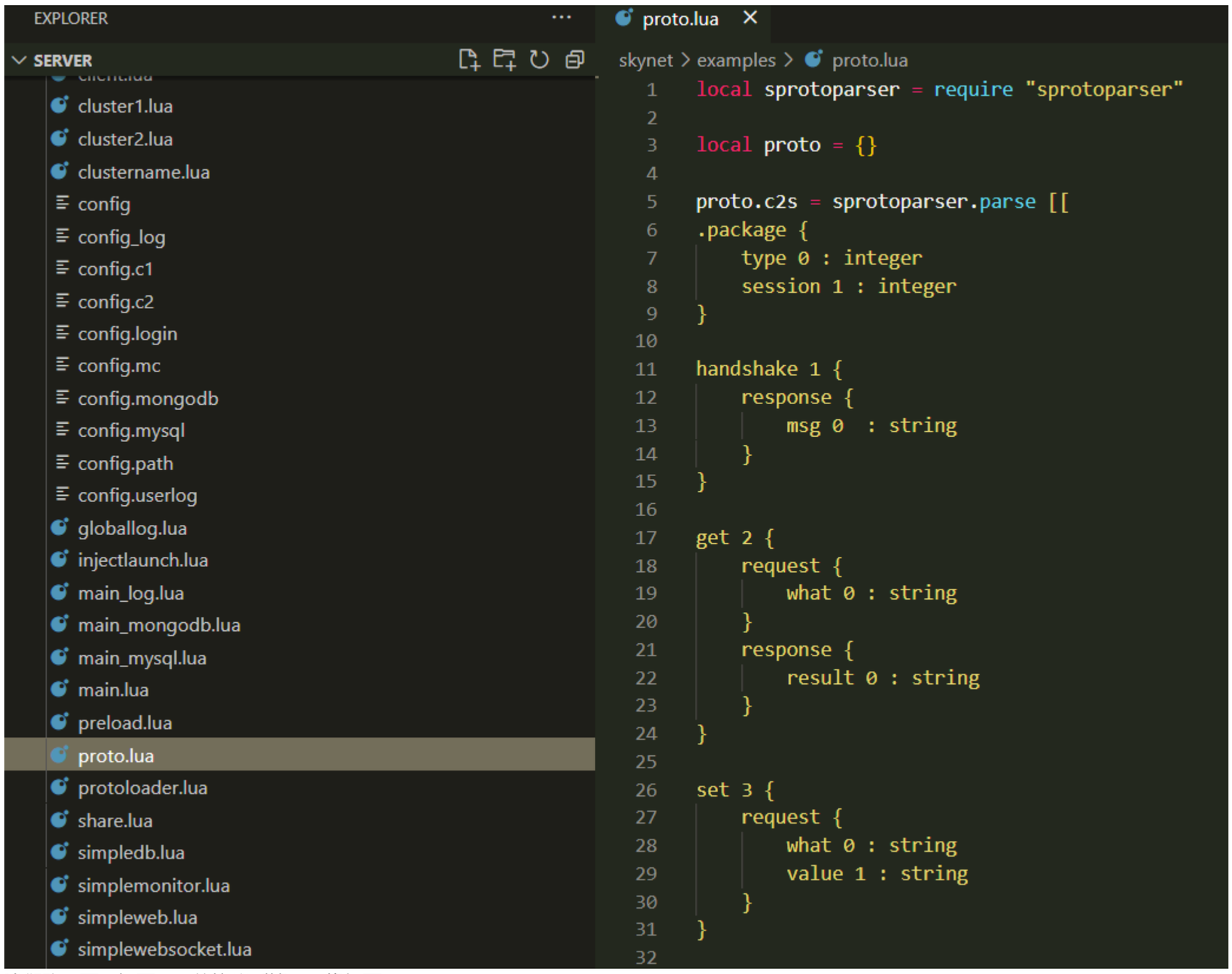
这是因为我们刚刚删除了 `sproto-Csharp` 的一些代码导致的，不用担心，我们等下执行 `.sproto` 生成 `cs` 就自动解决了。

3、编写.sproto协议文件

现在我们来写协议文件，包括服务端和客户端。

3.1、服务端协议文件：proto.lua

skynet 框架中的 examples 里，已经为我们准备好了一个协议文件： proto.lua ，



我们改一下，把不需要的协议删掉，最终如下：

```
local sprotoparser = require "sprotoparser"

local proto = {}

proto.c2s = sprotoparser.parse [[
.package {
    type 0 : integer
    session 1 : integer
}

sayhello 1 {
    request {
      what 0 : string
    }
    response {
      error_code 0 : integer
      msg 1 : string
    }
}

]]

proto.s2c = sprotoparser.parse [[
.package {
    type 0 : integer
    session 1 : integer
}

heartbeat 2 {}
]]

return proto
```

> 注：由于客户端的 sproto 工具的不能支持 c2s 和 s2c 的协议使用相同的 tag ， 比如 sayhello 消息的 tag 是 1 ， 那么 heartbeat 消息就不可以使用 1 作为 tag ， 这里我是使用 2 作为 heartbeat 的 tag 。

3.2、客户端协议文件：game.sproto

我们在 Unity 工程的 Assets/Sproto 目录中新建一个文件夹 protocol ， 用于存放协议文件，

在 protocol 文件夹中新建一个 game.proto 文件，内容如下：

```
.package {
  type 0 : integer
  session 1 : integer
}

sayhello 1 {
  request {
    what 0 : string
  }
  response {
    error_code 0 : integer
    msg 1 : string
  }
}

heartbeat 2 {}
```

4、客户端.sproto文件转C#脚本

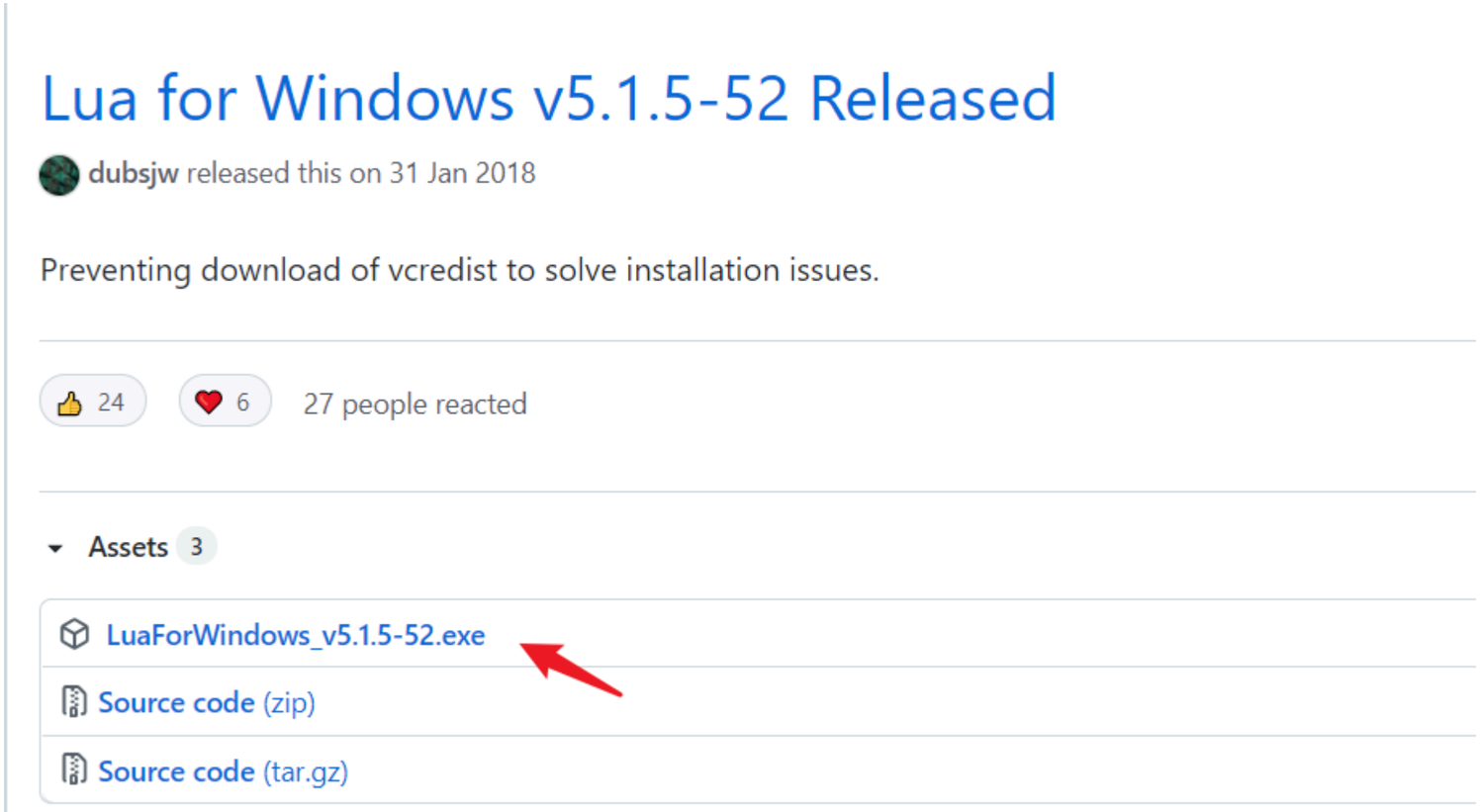
接下来，我们要使用 sprotodump 将客户端的 game.proto 文件转成 C# 脚本。

因为 sprotodump 需要是用 lua 来执行，所以这里我们需要先安装 lua 环境。

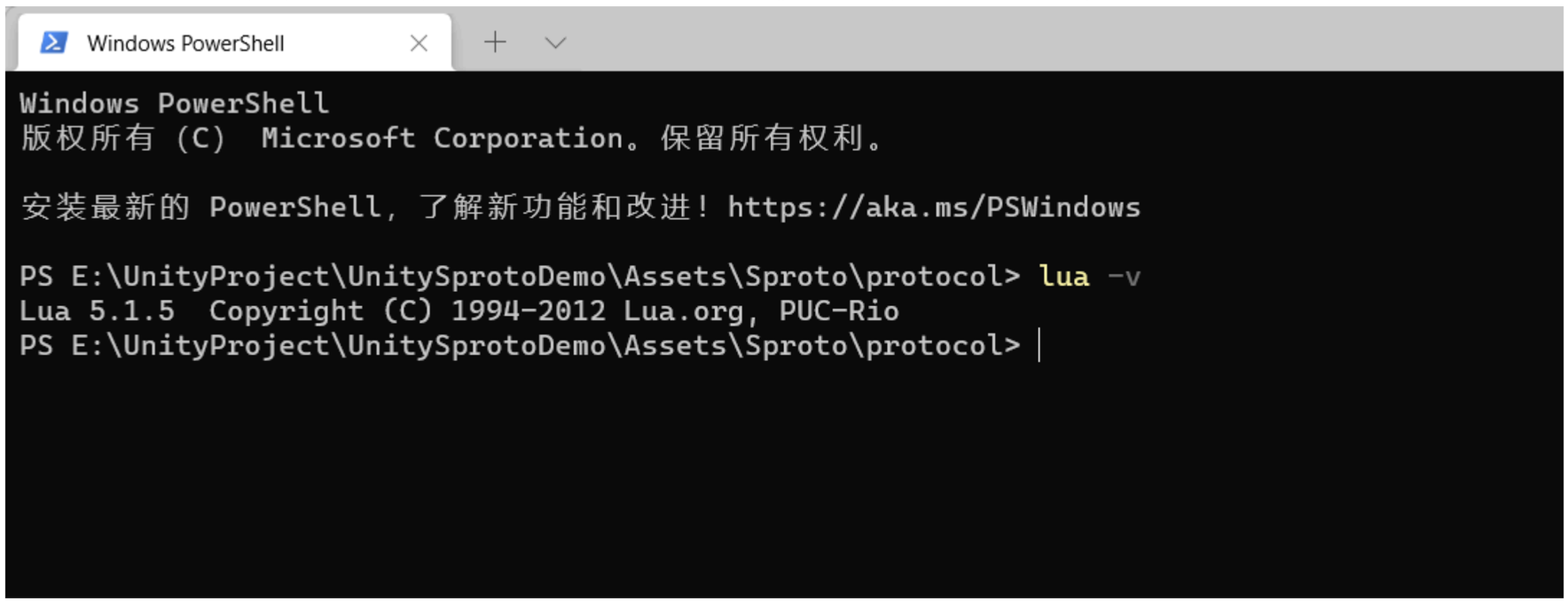
4.1、安装lua

lua 官网：http://www.lua.org/

lua Windows版 ： https://github.com/rjpcomputing/luaforwindows/releases



我们下载下来然后安装即可，完整完毕后，打开终端，执行 `lua -v` ，如果能输出版本号，则说明 `lua` 环境弄好了。



4.2、sprotodump工具：sproto文件生成C#脚本

`sprotodump` 是将 `.sproto` 文件转为 `.cs` 、 `.spb` 、 `.spb` 、 `.go` 、 `.md` 、 `.lua` 等文件的工具，我们这里是要把 `game.proto` 文件转成 `C#` 脚本。

`sprotodump` 的使用方法如下：

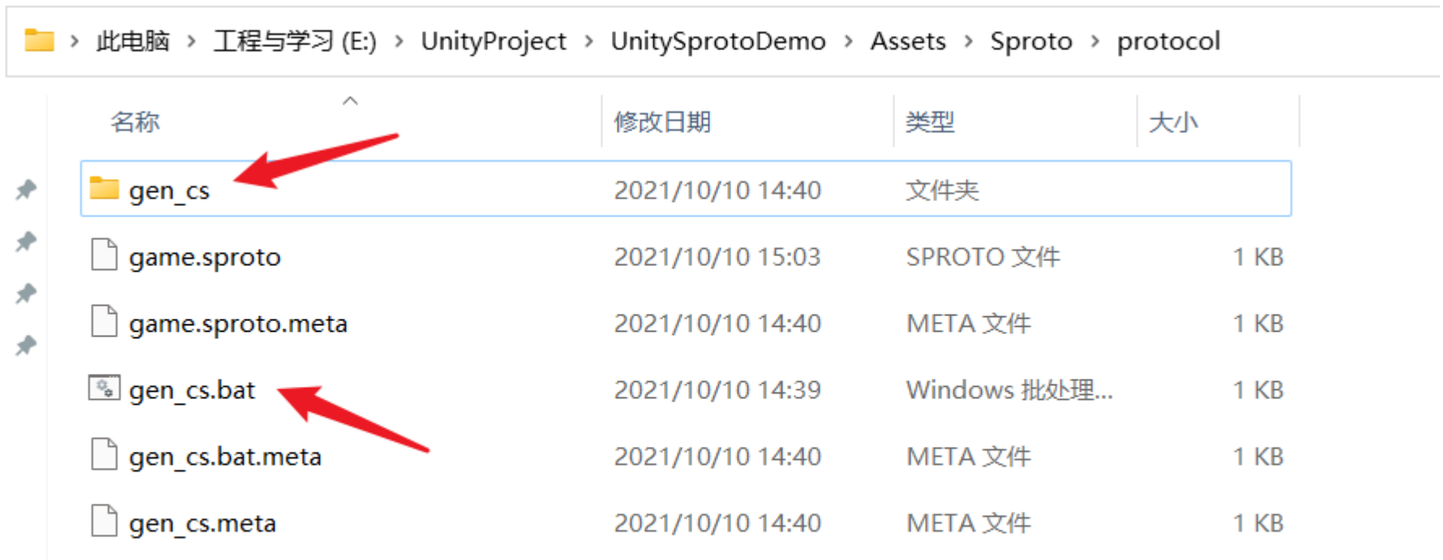
```
usage: lua sprotodump.lua <option> <sproto_file1 sproto_file2 ...> [[<out_option> <outfile>] ...] [namespace_option]

    option:
        -cs             dump to cSharp code file
        -spb            dump to binary spb  file
        -go             dump to go code file
        -md             dump to markdown file
        -lua            dump to lua table

    out_option:
        -d <dircetory>              dump to speciffic dircetory
        -o <file>                   dump to speciffic file
        -p <package name>           set package name(only cSharp code use)

    namespace_option:
        -namespace      add namespace to type and protocol
```

例：

```
lua lua sprotodump.lua -cs game.sproto -o gamesproto.cs
```

我们把命令写到 `bat` 脚本中，双击执行即可，提高工作效率。

在 `Assets/Sproto/protocol` 目录中创建 `gen_cs.bat` 文件，再创建一个 `gen_cs` 文件夹用于存放生成的 `C#` 脚本。



`gen_cs.bat` 脚本内容如下，我写了详细注释，大家应该能看懂，

```
echo off
:: 当前路径
set curdir=%~dp0
:: 进入sprotodump目录
cd /d %curdir%/../sprotodump
:: 将.sproto文件转为C#脚本，存放在gen_cs文件夹中
lua ./sprotodump.lua -cs %curdir%/game.sproto -o %curdir%/gen_cs/gamesproto.cs
:: 输出完成
echo sproto to cs, done
:: 按任意键退出
pause
```

我们执行 gen_cs.bat 脚本时，可能会报错：

```
PS E:\UnityProject\UnitySprotoDemo\Assets\Sproto\protocol> .\gen_cs.bat

E:\UnityProject\UnitySprotoDemo\Assets\Sproto\protocol>echo off
lua: ./sprotodump.lua:9: nesting of [[...]] is deprecated near '['
sproto to cs, done
请按任意键继续. . .
```

我们打开 sprotodump.lua 脚本，把 README 后面的 [[和]] 改成 [=[和]=] 即可，

```
5   ------------------------------ readme -------------------------------------
6   local README = [[
7   sprotodump is a simple tool to convert sproto file to spb binary.
8
9   usage: lua sprotodump.lua <option> <sproto_file1 sproto_file2 ...> [[<out_option> <outfile>] ...] [namespace_option]
10
11      option:
12          -cs             dump to cSharp code file
13          -spb            dump to binary spb  file
14          -go             dump to go code file
15          -md             dump to markdown file
16          -lua            dump to lua table
17
18      out_option:
19          -d <dircetory>              dump to speciffic dircetory
20          -o <file>                   dump to speciffic file
21          -p <package name>           set package name(only cSharp code use)
22
23      namespace_option:
24          -namespace      add namespace to type and protocol
25  ]]
26
27
```

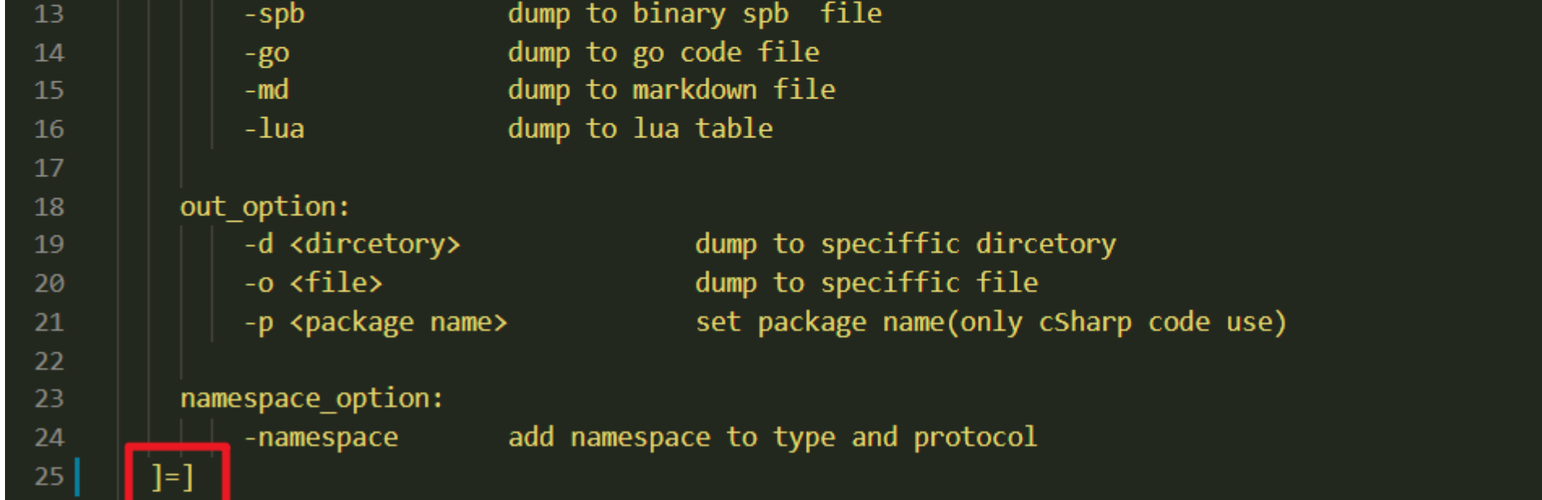
如下：

```
5   ------------------------------ readme -------------------------------------
6   local README = [=[
7   sprotodump is a simple tool to convert sproto file to spb binary.
8
9   usage: lua sprotodump.lua <option> <sproto_file1 sproto_file2 ...> [[<out_option> <outfile>] ...] [namespace_option]
10
11      option:
12          -cs             dump to cSharp code file
13          -spb            dump to binary spb  file
14          -go             dump to go code file
15          -md             dump to markdown file
16          -lua            dump to lua table
17
18      out_option:
19          -d <dircetory>              dump to speciffic dircetory
20          -o <file>                   dump to speciffic file
21          -p <package name>           set package name(only cSharp code use)
22
23      namespace_option:
24          -namespace      add namespace to type and protocol
25  ]=]
26
```

重新执行 gen_cs.bat 脚本，

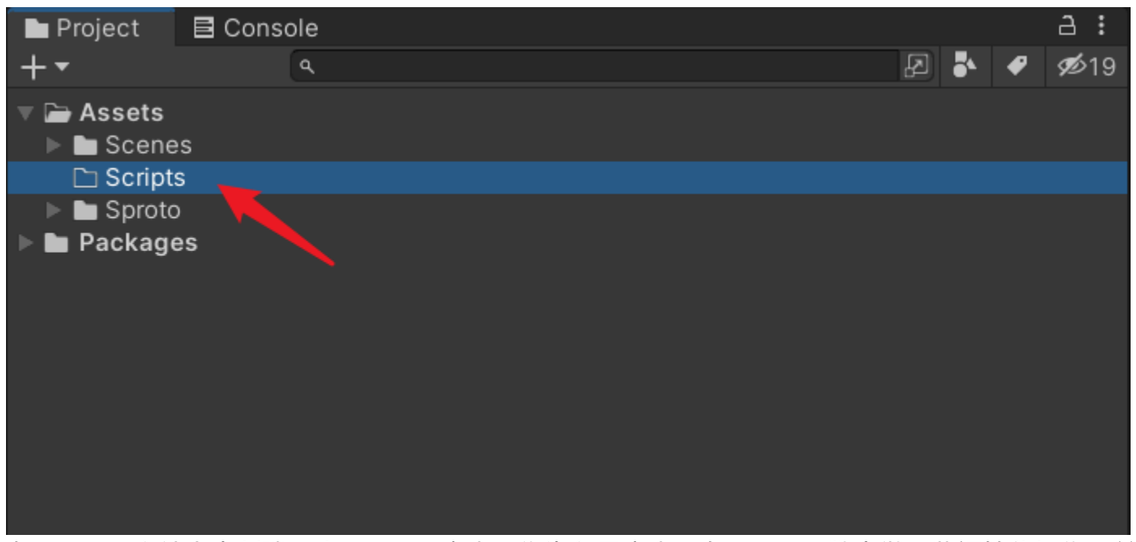
```
PS E:\UnityProject\UnitySprotoDemo\Assets\Sproto\protocol> .\gen_cs.bat

E:\UnityProject\UnitySprotoDemo\Assets\Sproto\protocol>echo off
dump to E:\UnityProject\UnitySprotoDemo\Assets\Sproto\protocol\/gen_cs/gamesproto.cs file
sproto to cs, done
请按任意键继续. . .
```

生成成功，可以看到 gen_cs 文件夹中生成了 gamesproto.cs 脚本，



5、客户端连接服务端

我们先新建一个 Scripts 文件夹，用于存放我们写的游戏逻辑脚本，



在 Scripts 文件夹中创建一个 Main.cs 脚本，作为入口脚本，在 Start 函数中做一些初始化操作，并在 Update 中驱动 NetCore 的消息分发，

```
// Main.cs

using UnityEngine;

public class Main : MonoBehaviour
{
    void Start()
    {
        // 初始化
        NetCore.Init();
        NetSender.Init();
        NetReceiver.Init();
        NetCore.enabled = true;
    }

    void Update()
    {
        // 驱动消息分发
        NetCore.Dispatch();
    }
}
```

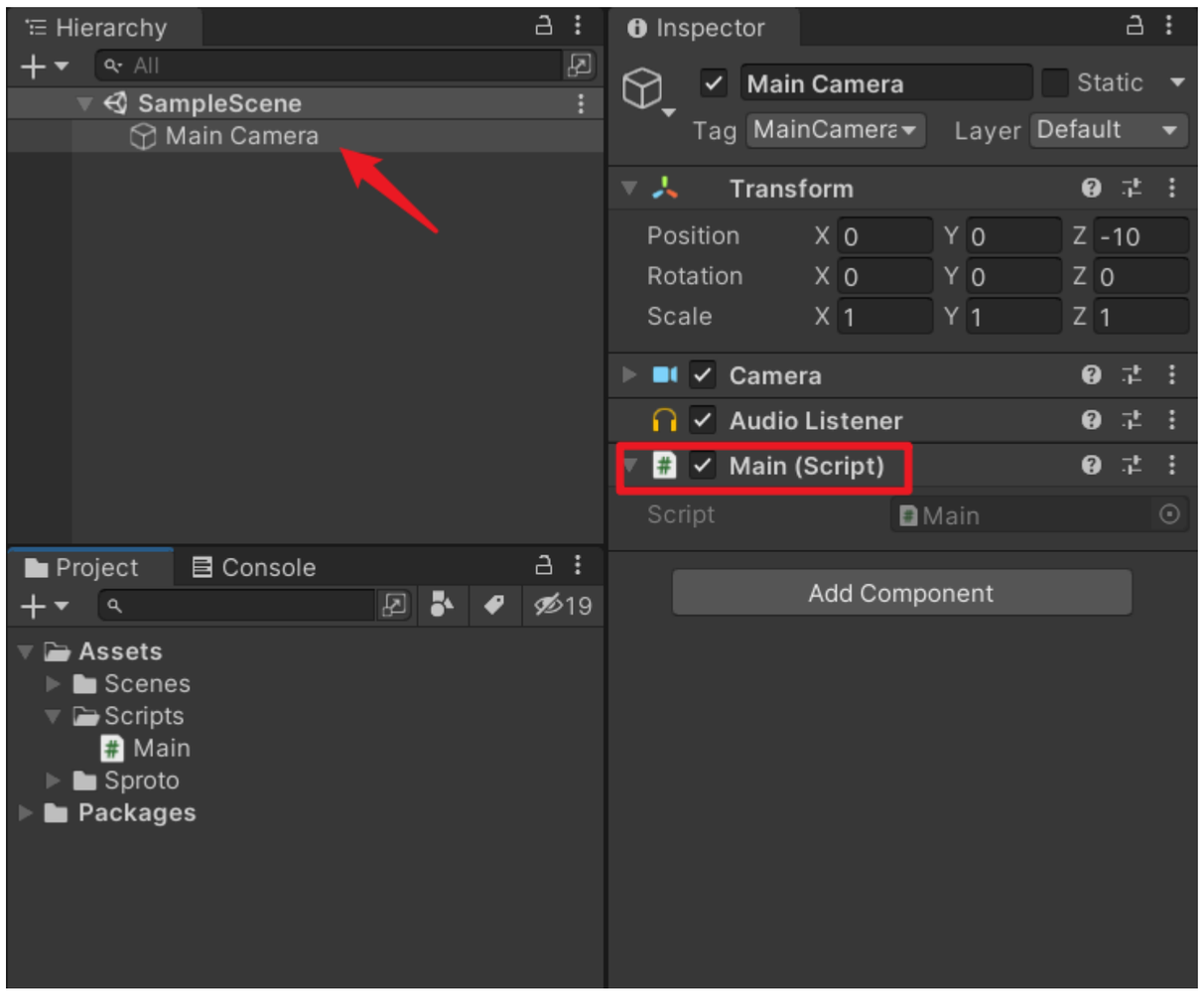
接着，我们就可以连接服务端了，由于是本地连接，所以 IP 地址使用 127.0.0.1 即可，端口的话，我们可以看到服务端监听的端口是 8888 ，

```
TERMINAL  DEBUG CONSOLE  PROBLEMS  OUTPUT
linxinfa@LAPTOP-KD7VD0IS:~$ . cd_skynet.sh
linxinfa@LAPTOP-KD7VD0IS:/mnt/e/UnityProject/UnitySprotoDemo/Server/skynet$ . start.sh
[:01000002] LAUNCH snlua bootstrap
[:01000003] LAUNCH snlua launcher
[:01000004] LAUNCH snlua cmaster
[:01000004] master listen socket 0.0.0.0:2013
[:01000005] LAUNCH snlua cslave
[:01000005] slave connect to master 127.0.0.1:2013
[:01000004] connect from 127.0.0.1:56848 4
[:01000006] LAUNCH harbor 1 16777221
[:01000004] Harbor 1 (fd=4) report 127.0.0.1:2526
[:01000005] Waiting for 0 harbors
[:01000005] Shakehand ready
[:01000007] LAUNCH snlua datacenterd
[:01000008] LAUNCH snlua service_mgr
[:01000009] LAUNCH snlua main
[:01000009] Server start
[:0100000a] LAUNCH snlua protoloader
[:0100000b] LAUNCH snlua console
[:0100000c] LAUNCH snlua debug_console 8000
[:0100000c] Start debug console at 127.0.0.1:8000
[:0100000d] LAUNCH snlua simpledb
[:0100000e] LAUNCH snlua watchdog
[:0100000f] LAUNCH snlua gate
[:0100000f] Listen on 0.0.0.0:8888
[:01000009] Watchdog listen on 8888
[:01000009] KILL self
[:01000002] KILL self
^C
```

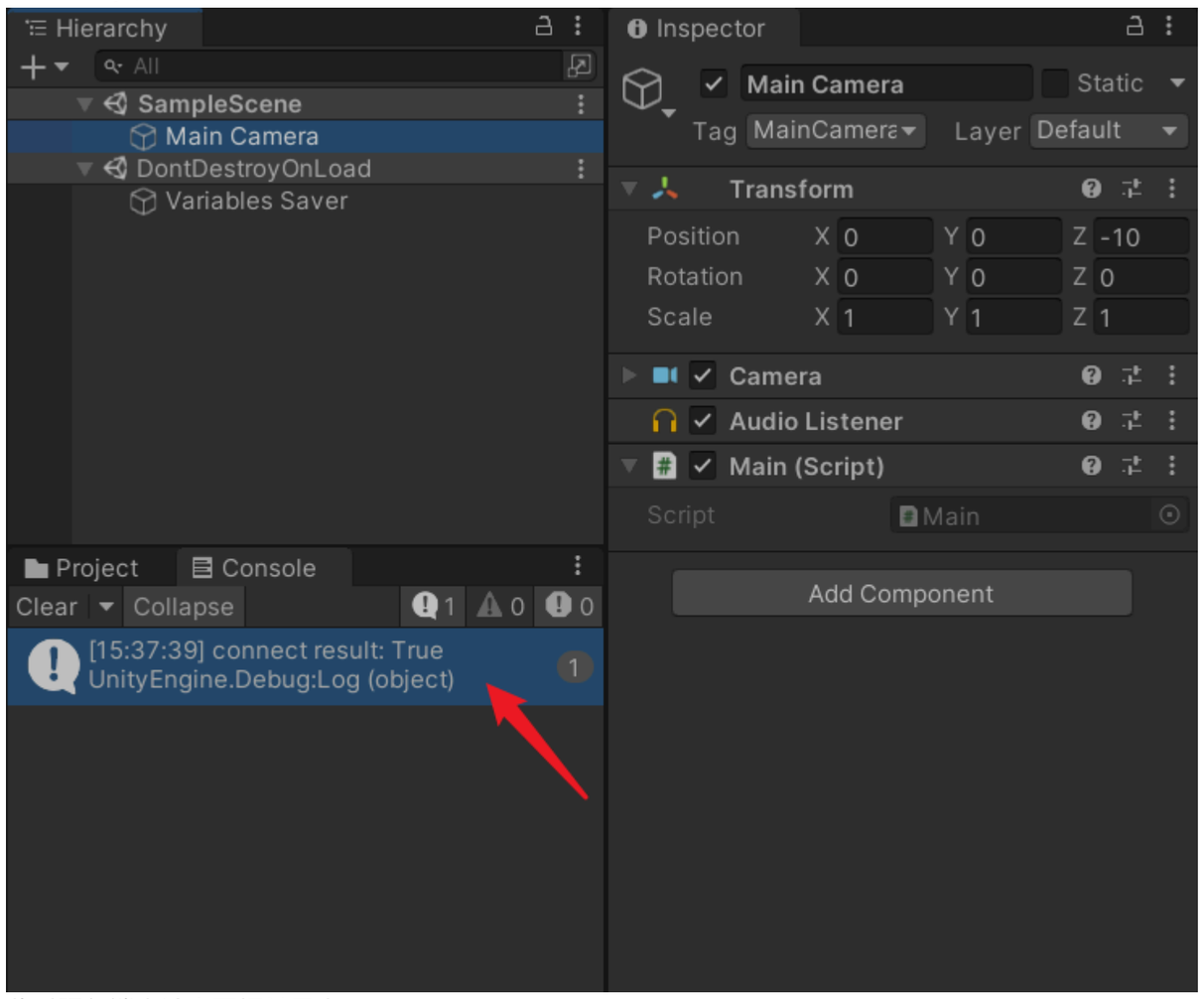
客户端连接服务端代码如下：

```
// Main.cs

void Start()
{
    // ...

    // 连接服务端
    NetCore.Connect("127.0.0.1", 8888, () =>
    {
        // 连接结果
        Debug.Log("connect result: " + NetCore.connected);
    });
}
```

我们把 `Main.cs` 脚本挂到 `Main Camera` 上，



运行 `Unity`，可以看到输出了 `connect result: True`，说明连接服务器成功了，



此时服务端也输出了相关日志，

四、客户端与服务端通信

1、客户端发消息给服务端：c2s

1.1、客户端部分

现在，我们让客户端给服务端发一条 `sayhello` 消息，

```
// Main.cs
void Start()
{
    // ...

    NetCore.Connect("127.0.0.1", 8888, () =>
    {
```

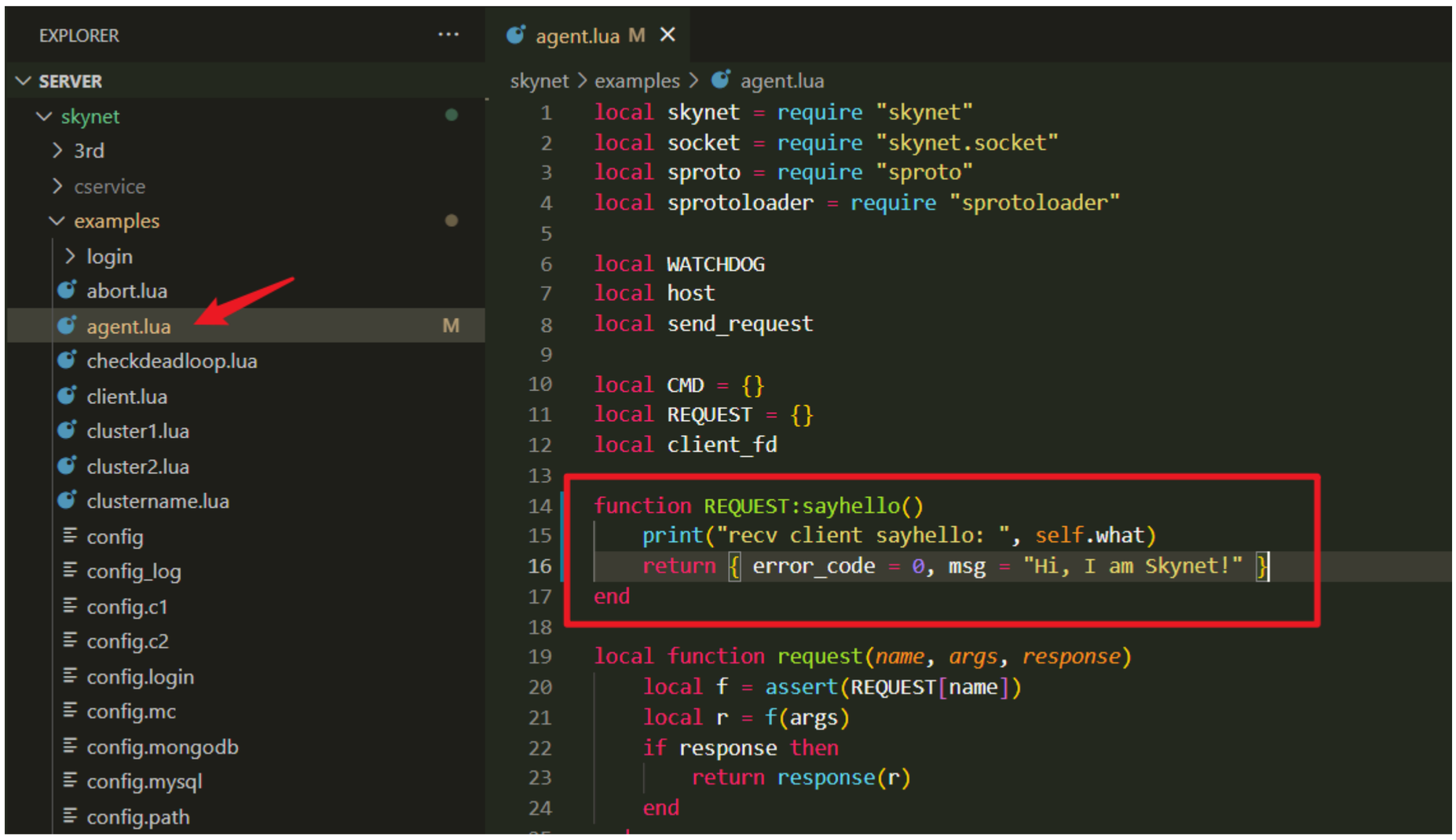
```
        Debug.Log("connect result: " + NetCore.connected);
        if (NetCore.connected)
        {
            // 给服务端发送一条sayhello消息
            SendSayHello();
        }
    });
}

void SendSayHello()
{
    var req = new SprotoType.sayhello.request();
    req.what = "Hi, I am Unity!";
    Debug.Log("发送sayhello消息给服务端");
    NetSender.Send<Protocol.sayhello>(req, (data) =>
    {
        var rsp = data as SprotoType.sayhello.response;
        Debug.LogFormat("服务端sayhello返回, error_code: {0}, msg: {1}", rsp.error_code, rsp.msg);
    });
}
```

1.2、服务端部分

我们给服务端的 `agent.lua` 脚本添加 `sayhello` 的响应，

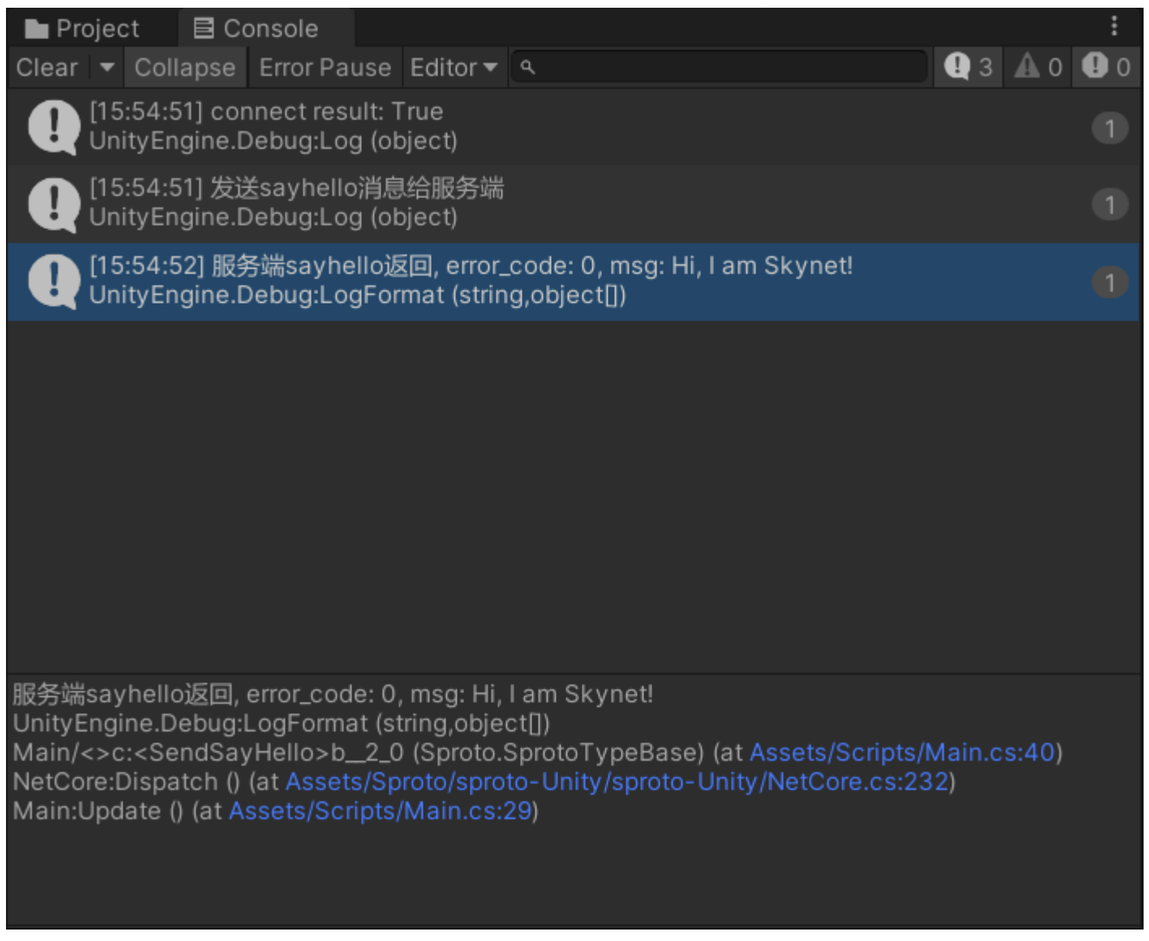
```
function REQUEST:sayhello()
    print("recv client sayhello: ", self.what)
    return { error_code = 0, msg = "Hi, I am Skynet!" }
end
```

如下：



1.3、运行测试

我们重新启动服务端，然后运行客户端，可以看到客户端发送了 `sayhello` 消息，并受到了服务端的返回，



我们看服务端的日志，也输出了相关日志，说明正常接收到了客户端的消息了，

```
[:0100000d] LAUNCH snlua simpledb
[:0100000e] LAUNCH snlua watchdog
[:0100000f] LAUNCH snlua gate
[:0100000f] Listen on 0.0.0.0:8888
[:01000009] Watchdog listen on 8888
[:01000009] KILL self
[:01000002] KILL self
[:0100000e] New client from : 127.0.0.1:40352
[:01000010] LAUNCH snlua agent
[:01000010] TRACE nil
[:01000010] <TRACE :01000010-1> 10396761805500 trace
[:01000010] <TRACE :01000010-1> 10396761818200 call : @./examples/agent.lua:73 @./ex
[:0100000f] <TRACE :01000010-1> 10396761915700 request
[:0100000f] <TRACE :01000010-1> 10396761947700 response
recv client sayhello:   Hi, I am Unity!
[:0100000f] <TRACE :01000010-1> 10396761951200 end
[:01000010] <TRACE :01000010-1> 10396761982600 resume
[:01000010] <TRACE :01000010-1> 10396761985800 response
[:01000010] <TRACE :01000010-1> 10396761987700 end
[:01000010] TRACE nil
[:01000010] <TRACE :01000010-2> 10396762072700 trace
[:01000010] <TRACE :01000010-2> 10396762154400 end
```

2、服务端发消息给客户端：s2c

服务端每隔 5秒 给客户端发送一条 heartbeat 消息，消息定义如下：

```
heartbeat 2 {}
```

2.1、服务端部分

如下，服务端受到客户端连接后，会循环每隔 5秒 给客户端发送一条 heartbeat 消息。

```lua
function CMD.start(conf)
    local fd = conf.client
    local gate = conf.gate
    WATCHDOG = conf.watchdog
    -- slot 1,2 set at main.lua
    host = sprotoloader.load(1):host "package"
    send_request = host:attach(sprotoloader.load(2))
    skynet.fork(function()
        while true do
            send_package(send_request("heartbeat"))
            skynet.sleep(500)
        end
    end)

    client_fd = fd
    skynet.call(gate, "lua", "forward", fd)
end
```

注意：因为 heartbeat 消息没有参数，所以这里不用传其他参数，如果 heartbeat 的消息定义中有参数，比如这样

```
heartbeat 2 {
    request {
      cnt 0 : integer
    }
}
```

那么发消息时传 cnt 参数是这样子的：

```
send_package(send_request("heartbeat", { cnt = 666 }))
```

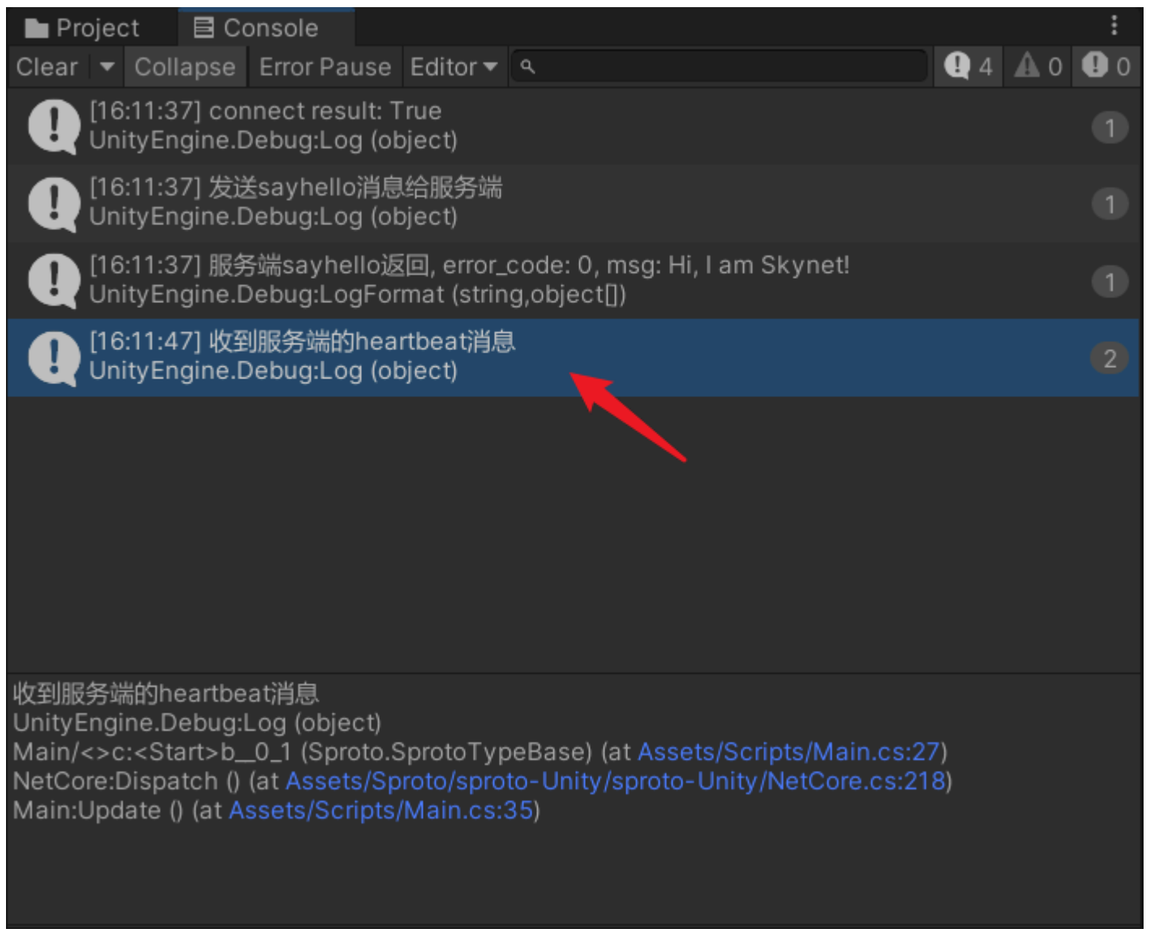
2.2、客户端部分

客户端部分需要通过 NetReceiver 注册消息的响应函数，如下：

```
NetReceiver.AddHandler<Protocol.heartbeat>((data) =>
{
    Debug.Log("收到服务端的heartbeat消息");
    return null;
});
```

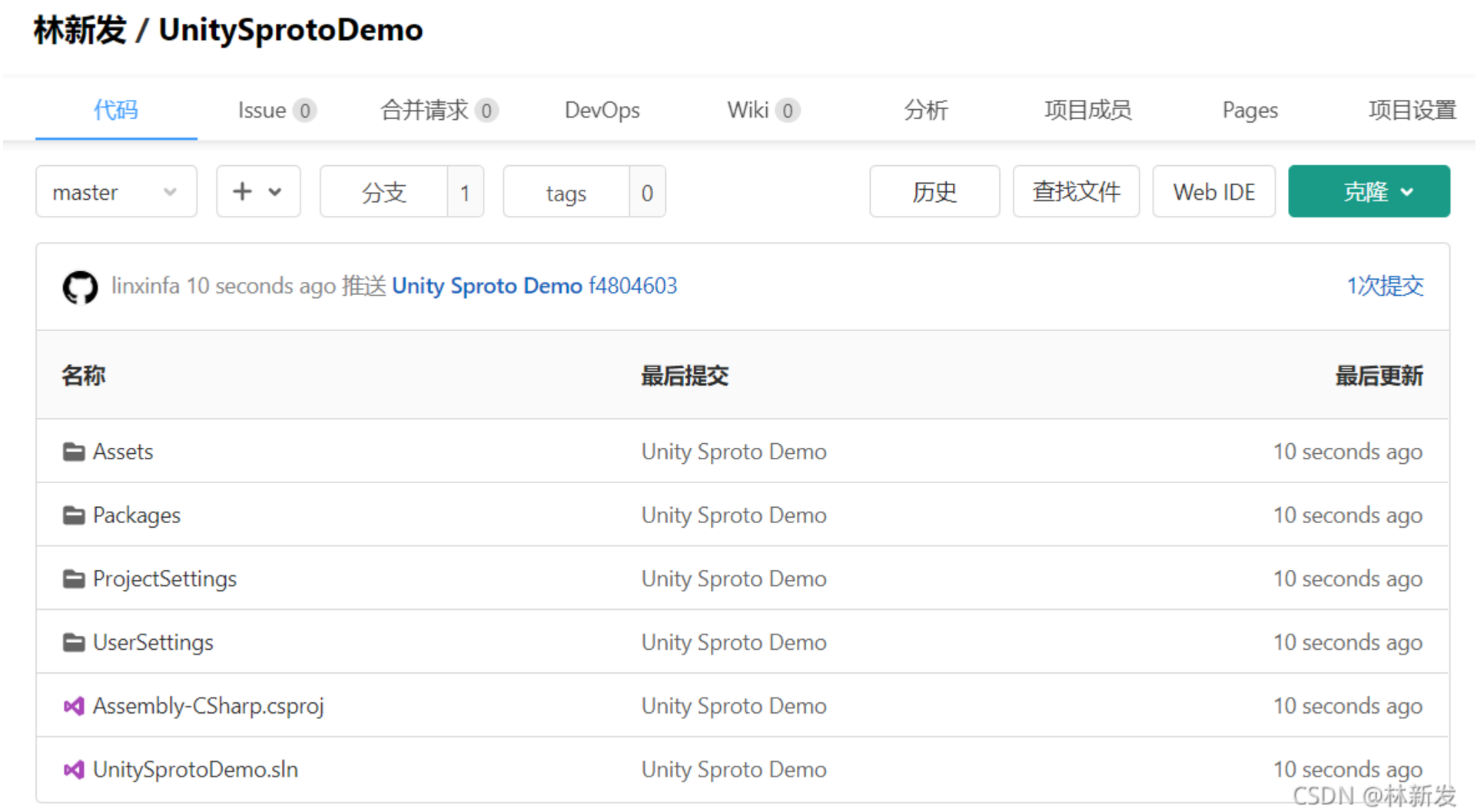
2.3、运行测试

我们重新启动服务端，然后运行客户端，可以看到客户端收到了服务端发送的 `heartbeat` 消息了，



五、工程源码

本文工程源码，我已上传到 `CODE CHINA` ，感兴趣的同学可自行下载学习，地址：
https://codechina.csdn.net/linxinfa/UnitySprotoDemo



五、完毕

好了，就先写这么多吧，关于 `Skynet` 、 `Sproto` 还有很多很多内容，本文只是一个入门，希望可以帮助到新手同学~
我是林新发：https://blog.csdn.net/linxinfa
原创不易，若转载请注明出处，感谢大家~
喜欢我的可以点赞、关注、收藏，如果有什么技术上的疑问，欢迎留言或私信~