# □□□□ -□□□□

□□□□□□□□□□

□□□
```
 using (var pipeServer = new AnonymousPipeServerStream(PipeDirection.Out, HandleInheritability.Inheritable))
 {
    Process childProcess = new Process
    {
       StartInfo = new ProcessStartInfo
       {
          FileName = "dotnet",
          Arguments = $"run --project Hub1 --no-build -- ClientProgram {pipeServer.GetClientHandleAsString()}",
          UseShellExecute = false,
          RedirectStandardOutput = true,
          RedirectStandardError = true
       }
    };

    childProcess.Start();

    using (StreamWriter writer = new StreamWriter(pipeServer) { AutoFlush = true })
    {
       await writer.WriteLineAsync("Hello from parent!");
    }
 }
```
□□□
```
 if (args.Length > 0)
 {
    string pipeHandle = args[0];

    using (var pipeClient = new AnonymousPipeClientStream(PipeDirection.In, pipeHandle))
    {
       using (StreamReader reader = new StreamReader(pipeClient))
       {
          string message = await reader.ReadLineAsync();
          Console.WriteLine($"□□□□□□□          : {message}");
```

```
        }
    }
}
```

□□□□□□□□

□□□
```
Console.WriteLine("pipe□□□□□□□           ...");

var pipeServer = new NamedPipeServerStream("mypipe", PipeDirection.InOut,
NamedPipeServerStream.MaxAllowedServerInstances, PipeTransmissionMode.Byte, PipeOptions.Asynchronous);
await pipeServer.WaitForConnectionAsync();
Console.WriteLine("□□□□□□          ");

while (true)
{
    byte[] buffer = new byte[1024];
    int bytesRead = await pipeServer.ReadAsync(buffer, 0, buffer.Length);
    if (bytesRead > 0)
    {
        string message = Encoding.UTF8.GetString(buffer, 0, bytesRead);
        Console.WriteLine($"□□□□      : {message}");
    }
}
```
□□□
```
using (var pipeClient = new NamedPipeClientStream(".", "mypipe", PipeDirection.InOut,
PipeOptions.Asynchronous))
{
    await pipeClient.ConnectAsync();
    Console.WriteLine("□□□□□□□□□              ");

    while (true)
    {
        string message = Console.ReadLine();
        byte[] buffer = Encoding.UTF8.GetBytes(message);
        await pipeClient.WriteAsync(buffer, 0, buffer.Length);
        await pipeClient.FlushAsync();
    }
}
```

□□□□□□□□□□□

```
//□□□
var pipeServer = new NamedPipeServerStream("mypipe", PipeDirection.InOut,
NamedPipeServerStream.MaxAllowedServerInstances, PipeTransmissionMode.Byte, PipeOptions.Asynchronous);
await pipeServer.WaitForConnectionAsync();
Console.WriteLine("□□□□□□          ");

while (true)
{
    byte[] buffer = new byte[1024];
    int bytesRead = await pipeServer.ReadAsync(buffer, 0, buffer.Length);
    if (bytesRead > 0)
    {
        string message = Encoding.UTF8.GetString(buffer, 0, bytesRead);
        Console.WriteLine($"□□□□      : {message}");
    }
}
//□□□
using (var pipeClient = new NamedPipeClientStream("ServerMachine", "mypipe", PipeDirection.InOut,
PipeOptions.Asynchronous))
{
    await pipeClient.ConnectAsync();
    Console.WriteLine("□□□□□□□□□              ");

    while (true)
    {
        string message = Console.ReadLine();
        byte[] buffer = Encoding.UTF8.GetBytes(message);
        await pipeClient.WriteAsync(buffer, 0, buffer.Length);
        await pipeClient.FlushAsync();
    }
}
```

## PIPE□□□□□□□

```
// □□□□□□
public class PipeServer
{
    private readonly string _pipeName;
```

```
    private readonly PipeDirection _direction;
    private readonly int _maxInstances;
    private readonly PipeTransmissionMode _transmissionMode;
    private readonly PipeOptions _options;

    public event EventHandler<string> MessageReceived;

    public PipeServer(string pipeName, PipeDirection direction, int maxInstances, PipeTransmissionMode
transmissionMode, PipeOptions options)
    {
        _pipeName = pipeName;
        _direction = direction;
        _maxInstances = maxInstances;
        _transmissionMode = transmissionMode;
        _options = options;
    }

    // □□□□□□□
    public void Start()
    {
        Task.Run(() => ListenForClients());
    }

    // □□□□□□□
    private async Task ListenForClients()
    {
        while (true)
        {
            using (var namedPipeServerStream = new NamedPipeServerStream(_pipeName, _direction,
_maxInstances, _transmissionMode, _options))
            {
                try
                {
                    await namedPipeServerStream.WaitForConnectionAsync();
                    Console.WriteLine("□□□□□□          ");

                    // □□□□□□□□□□□□
                    byte[] nameBytes = new byte[1024];
                    int nameBytesRead = await namedPipeServerStream.ReadAsync(nameBytes, 0,
nameBytes.Length);
```

```
                    string processName = Encoding.UTF8.GetString(nameBytes, 0, nameBytesRead);
                    Console.WriteLine($"连接的进程名称          : {processName}");

                    // 持续读取客户端发送的消息
                    while (namedPipeServerStream.IsConnected)
                    {
                        byte[] bytes = new byte[1024];
                        int bytesRead = await namedPipeServerStream.ReadAsync(bytes, 0, bytes.Length);
                        if (bytesRead > 0)
                        {
                            string msg = Encoding.UTF8.GetString(bytes, 0, bytesRead);
                            Console.WriteLine($"进程   : {msg}");
                            OnMessageReceived(msg);
                        }
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine($"错误   : {ex.Message}");
                }
            }
        }
    }

    // 触发消息接收事件
    protected virtual void OnMessageReceived(string message)
    {
        MessageReceived?.Invoke(this, message);
    }
}
```

```
// 管道服务器构建器类
public class PipeServerBuilder
{
    private string _pipeName;
    private PipeDirection _direction;
    private int _maxInstances;
    private PipeTransmissionMode _transmissionMode;
    private PipeOptions _options;

```

```
// □□□□□□□
public PipeServerBuilder WithPipeName(string pipeName)
{
    _pipeName = pipeName;
    return this;
}

// □□□□□□□
public PipeServerBuilder WithDirection(PipeDirection direction)
{
    _direction = direction;
    return this;
}

// □□□□□□□□
public PipeServerBuilder WithMaxInstances(int maxInstances)
{
    _maxInstances = maxInstances;
    return this;
}

// □□□□□□□
public PipeServerBuilder WithTransmissionMode(PipeTransmissionMode transmissionMode)
{
    _transmissionMode = transmissionMode;
    return this;
}

// □□□□□□□
public PipeServerBuilder WithOptions(PipeOptions options)
{
    _options = options;
    return this;
}

// □□□□□□□□
public PipeServer Build()
{
    return new PipeServer(_pipeName, _direction, _maxInstances, _transmissionMode, _options);
}
```

```
    }
```

```
public class PipeClient
{
    private readonly string _pipeName;
    private NamedPipeClientStream _namedPipeClientStream;

    public event EventHandler<string> MessageToSend;

    public PipeClient(string pipeName)
    {
        _pipeName = pipeName;
    }

    public async Task StartAsync()
    {
        string processName = Process.GetCurrentProcess().ProcessName;

        _namedPipeClientStream = new NamedPipeClientStream(".", _pipeName, PipeDirection.InOut,
PipeOptions.Asynchronous);
        try
        {
            await _namedPipeClientStream.ConnectAsync();
            Console.WriteLine("□□□□□□□□□□                ");

            // □□□□□□
            byte[] nameBytes = Encoding.UTF8.GetBytes(processName);
            await _namedPipeClientStream.WriteAsync(nameBytes, 0, nameBytes.Length);
            await _namedPipeClientStream.FlushAsync();

            // □□□□□□□□
            MessageToSend += async (sender, message) =>
            {
                byte[] messageBytes = Encoding.UTF8.GetBytes(message);
                await _namedPipeClientStream.WriteAsync(messageBytes, 0, messageBytes.Length);
                await _namedPipeClientStream.FlushAsync();
            };

            Console.WriteLine("□□□□□□□□□□□□□□                    ...");
        }
```

```
        catch (Exception ex)
        {
            Console.WriteLine($"□□  : {ex.Message}");
        }
    }

    public void SendMessage(string message)
    {
        MessageToSend?.Invoke(this, message);
    }
}
```

```
 // □□□□□□□□□□
 var pipeServer = new PipeServerBuilder()
     .WithPipeName("mypipe")
     .WithDirection(PipeDirection.InOut)
     .WithMaxInstances(NamedPipeServerStream.MaxAllowedServerInstances)
     .WithTransmissionMode(PipeTransmissionMode.Byte)
     .WithOptions(PipeOptions.Asynchronous)
     .Build();
 // □□□□□□□
 pipeServer.Start();
```

```
// □□□□□□□□
var pipeClient = new PipeClient("mypipe");
await pipeClient.StartAsync();

// □□□□□□□
while (true)
{
    string message = Console.ReadLine();
    pipeClient.SendMessage(message);
}
```

---

□□□  #3
□ □□  □□□  12 □□  2025 00:18:58
□ □□  □□□  15 □□□  2026 01:02:04