进程通信

父子进程匿名管道通信

父进程

```
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!");
    }
}
```

子进程

```
 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($"收到父进程消息            : {message}");
```

```
        }
    }
}
```

普通命名管道通信

服务端

```
Console.WriteLine("pipe管道服务已启动            ...");

var pipeServer = new NamedPipeServerStream("mypipe", PipeDirection.InOut,
NamedPipeServerStream.MaxAllowedServerInstances, PipeTransmissionMode.Byte, PipeOptions.Asynchronous);
await pipeServer.WaitForConnectionAsync();
Console.WriteLine("客户端已连接           ");

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($"收到消息       : {message}");
    }
}
```

客户端

```
using (var pipeClient = new NamedPipeClientStream(".", "mypipe", PipeDirection.InOut,
PipeOptions.Asynchronous))
{
    await pipeClient.ConnectAsync();
    Console.WriteLine("已连接到管道服务器                ");

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

跨设备命名管道进程通信

```
//服务端
var pipeServer = new NamedPipeServerStream("mypipe", PipeDirection.InOut,
NamedPipeServerStream.MaxAllowedServerInstances, PipeTransmissionMode.Byte, PipeOptions.Asynchronous);
await pipeServer.WaitForConnectionAsync();
Console.WriteLine("客户端已连接          ");

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($"收到消息       : {message}");
    }
}
//客户端
using (var pipeClient = new NamedPipeClientStream("ServerMachine", "mypipe", PipeDirection.InOut,
PipeOptions.Asynchronous))
{
    await pipeClient.ConnectAsync();
    Console.WriteLine("已连接到管道服务器              ");

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

PIPE管道通信库封装

```
// 管道服务器类
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;
    }

    // 启动管道服务器
    public void Start()
    {
        Task.Run(() => ListenForClients());
    }

    // 监听客户端连接
    private async Task ListenForClients()
    {
        while (true)
        {
            using (var namedPipeServerStream = new NamedPipeServerStream(_pipeName, _direction, _maxInstances, _transmissionMode, _options))
            {
                try
                {
                    await namedPipeServerStream.WaitForConnectionAsync();
                    Console.WriteLine("进程连接管道          ");

                    // 读取客户端发送的进程名称
                    byte[] nameBytes = new byte[1024];
                    int nameBytesRead = await namedPipeServerStream.ReadAsync(nameBytes, 0, nameBytes.Length);
```

```
                string processName = Encoding.UTF8.GetString(nameBytes, 0, nameBytesRead);
                Console.WriteLine($"□□□□□□        : {processName}");

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

 // □□□□□□□
 protected virtual void OnMessageReceived(string message)
 {
     MessageReceived?.Invoke(this, message);
 }
}
```

```
// □□□□□□□
public class PipeServerBuilder
{
    private string _pipeName;
    private PipeDirection _direction;
    private int _maxInstances;
    private PipeTransmissionMode _transmissionMode;
    private PipeOptions _options;

```

```
// 设置管道名称
public PipeServerBuilder WithPipeName(string pipeName)
{
    _pipeName = pipeName;
    return this;
}

// 设置管道方向
public PipeServerBuilder WithDirection(PipeDirection direction)
{
    _direction = direction;
    return this;
}

// 设置最大实例数
public PipeServerBuilder WithMaxInstances(int maxInstances)
{
    _maxInstances = maxInstances;
    return this;
}

// 设置传输模式
public PipeServerBuilder WithTransmissionMode(PipeTransmissionMode transmissionMode)
{
    _transmissionMode = transmissionMode;
    return this;
}

// 设置管道选项
public PipeServerBuilder WithOptions(PipeOptions options)
{
    _options = options;
    return this;
}

// 构建管道服务器
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("已连接到管道服务器                ");

            // 发送进程名称
            byte[] nameBytes = Encoding.UTF8.GetBytes(processName);
            await _namedPipeClientStream.WriteAsync(nameBytes, 0, nameBytes.Length);
            await _namedPipeClientStream.FlushAsync();

            // 订阅消息发送事件
            MessageToSend += async (sender, message) =>
            {
                byte[] messageBytes = Encoding.UTF8.GetBytes(message);
                await _namedPipeClientStream.WriteAsync(messageBytes, 0, messageBytes.Length);
                await _namedPipeClientStream.FlushAsync();
            };

            Console.WriteLine("客户端已启动，等待发送消息                    ...");
        }
```

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

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

```
// 创建并配置管道服务器
var pipeServer = new PipeServerBuilder()
    .WithPipeName("mypipe")
    .WithDirection(PipeDirection.InOut)
    .WithMaxInstances(NamedPipeServerStream.MaxAllowedServerInstances)
    .WithTransmissionMode(PipeTransmissionMode.Byte)
    .WithOptions(PipeOptions.Asynchronous)
    .Build();
// 启动管道服务器
pipeServer.Start();
```

```
// 创建并启动客户端
var pipeClient = new PipeClient("mypipe");
await pipeClient.StartAsync();

// 发送不同的消息
while (true)
{
    string message = Console.ReadLine();
    pipeClient.SendMessage(message);
}
```

版本号 #2
由 叶子 创建于 12 四月 2025 00:18:58
由 叶子 更新于 12 四月 2025 00:37:22