

# wpf□□□□

□□□

- nuget□□□
- □□□□
- □□□
- □□□□□□□
- □□□
- □□□
- □□□

# nuget

## nuget

```
//
```

```
dotnet nuget delete -s https://nuget.lingyanspace.com/v3/index.json LingYanAutoUpdate 1.0.0 -k  
nugetlingyanspace --non-interactive
```



**WPFWPF**



wpf

C#

MAUI

xaml

xaml

B wpf

**WPFWPF**



1 线程池  
2 线程池  
3 线程池  
4 线程池  
5 线程池  
6 线程池  
7 C# 线程池  
8 Thread Task  
9 线程池  
10 WPF 线程池  
11 UI 线程池  
12 Dispatcher UI

线程池

1 Span<T> 线程池 Span<T> 线程池  
2 线程池 线程池 CancellationToken  
3 WPF 线程池 WPF 线程池

线程池

1 TcpListener 线程池  
线程池 TCP 线程池  
线程池  
2 SignalR 线程池  
线程池 SignalR 线程池  
3 WPF 线程池  
WPF 线程池  
线程池

线程池

1 线程池 MonitorMutex, Semaphore 线程池 Dispatcher BackgroundWorker  
2 线程池 async/await 线程池 TaskScheduler 线程池  
3 TPL Dataflow 线程池 线程池 -线程池  
4 WPF 线程池 WPF 线程池 UI 线程池

线程池

1 TPL 线程池 线程池  
2 线程池 线程池 Span<T> Memory<T> 线程池

3 WPF UI

WPF UI

UI

1 WebSocket UI System.Net.WebSockets WebSocket UI

2 SignalR UI SignalR UI

3 UI gRPC RESTful API UI

4 WPF UI WPF UI WebSocket SignalR UI

1 UI UI

2 WPF UI UI UI

SignalR WebSocket

WPF UI

SignalR UI

Dispatcher UI

TPL Dataflow UI

TaskScheduler UI

## III 数据库

### 3.1 数据库

#### 3.1.1 数据库

- 数据库 **ICollectionView**数据库
- 数据库 **VisualStateManager**数据库

#### 3.1.2 数据库

- 数据库 **NLog**数据库 **Debug/Info/Error**数据库
- 数据库 **RBAC**数据库

### 3.2 数据库

#### 3.2.1 数据库

- 数据库 **OxyPlot**数据库
- 数据库 **WriteableBitmap**数据库

## IV 数据库

### 4.1 数据库

#### 4.1.1 .NET MAUI数据库

- 数据库 **UI**数据库
- 数据库 **SkiaSharp**数据库

#### 4.1.2 WPF数据库 Web数据库

- 数据库 **WebView2**数据库
- 数据库 **WebAssembly**数据库 **Blazor**数据库

### 4.2 数据库

#### 4.2.1 数据库

- 数据库 **ML.NET**数据库
- 数据库 **ONNX**数据库

#### 4.2.2 数据库

- 数据库 **LiveCharts**数据库
- 数据库 **Parallel.For**数据库



inno



ChineseSimplified.isl

ChineseSimplified.isl

isl



Name: "english"; MessagesFile: "compiler:Default.isl"  
Name: "chinesesimplified"; MessagesFile: "compiler:Languages\ChineseSimplified.isl"



# OSI



層	機能	標準 / 技術	プロトコル
物理層	データの物理的な伝送	RS-232, RS-485, USB, Ethernet, CAN, Wi-Fi, ZigBee, LoRa, NFC	Modbus RTU, RS-485, CANopen, CAN
データリンク層	データの正確な伝送	Ethernet, IEEE 802.3, Wi-Fi, IEEE 802.11, PPP, HDLC	Profinet, EtherCAT
ネットワーク層	データの経路制御	IPv4, IPv6, ICMP, ARP	Modbus TCP, IP, CIP, Common Industrial Protocol
トランスポート層	データの分割と再組み立て	TCP, UDP	MQTT, TCP, CoAP, UDP
セッション層	データのセッション管理	SMB, RPC	OPC UA
表示層	データの形式変換	TLS/SSL, JSON, XML	
アプリケーション層	データのアプリケーション処理	HTTP/HTTPS, FTP/SFTP, SMTP/IMAP/POP3, WebSocket, gRPC	Modbus TCP, TCP, OPC UA, BACnet

# IPC



機能	層	標準	技術	プロトコル	プロトコル
物理層	データの物理的な伝送	IEEE 802.3, OSI	イーサネット	イーサネット	イーサネット
データリンク層	データの正確な伝送	IEEE 802.3, OSI	イーサネット	イーサネット	イーサネット
ネットワーク層	データの経路制御	IEEE 802.3, OSI	イーサネット	イーサネット	イーサネット
トランスポート層	データの分割と再組み立て	IEEE 802.3, OSI	イーサネット	イーサネット	イーサネット
セッション層	データのセッション管理	IEEE 802.3, OSI	イーサネット	イーサネット	イーサネット
表示層	データの形式変換	IEEE 802.3, OSI	イーサネット	イーサネット	イーサネット
アプリケーション層	データのアプリケーション処理	IEEE 802.3, OSI	イーサネット	イーサネット	イーサネット



应用层	应用层 应用层 应用层	应用层 应用层 应用层 应用层	应用层 应用层 应用层	应用层 应用层 应用层	应用层 应用层 应用层
应用层	应用层 应用层 应用层	应用层 应用层 应用层 应用层	应用层 应用层	应用层 应用层	应用层 应用层 应用层 应用层
应用层	应用层 应用层	应用层 应用层 应用层 应用层	应用层 应用层	应用层 应用层	应用层 应用层
应用层 Socket 应用层	应用层 应用层 TCP/UDP 应用层	应用层	应用层 应用层 应用层	应用层 应用层 应用层	应用层 应用层 应用层
应用层 Signal 应用层	应用层 应用层	应用层 应用层 应用层	应用层 应用层 应用层	应用层 应用层	应用层 应用层 应用层
应用层 Memory-Mapped Files 应用层	应用层 应用层 应用层	应用层 应用层 应用层 应用层	应用层 应用层 应用层	应用层 应用层 应用层	应用层 应用层 应用层
应用层 gRPC	应用层 HTTP/2 应用层 RPC 应用层	应用层	应用层 应用层	应用层 应用层 应用层	应用层 应用层 应用层 应用层
应用层 REST API	应用层 HTTP 应用层 应用层	应用层	应用层 - 应用层 应用层 应用层 应用层 Web 应用层	应用层 应用层 应用层 -应用层	应用层 应用层 应用层
应用层 WebSocket	应用层 TCP 应用层 应用层	应用层	应用层 应用层 应用层 应用层	应用层 应用层 应用层	应用层 应用层 应用层
应用层 SignalR	应用层 WebSocket/SSE/Long Polling 应用层	应用层	应用层 应用层	应用层 应用层 应用层	应用层 WebSocket 应用层 应用层 Long Polling 应用层
应用层 MQTT	应用层 TCP 应用层 / 应用层	应用层	应用层 应用层	应用层 应用层 QoS 应用层 应用层 应用层	应用层 MQTT Broker 应用层 应用层 应用层

D-Bus	Linux D-Bus D-Bus	OS OS OS	Linux D-Bus D-Bus D-Bus	Linux D-Bus Linux	Linux D-Bus D-Bus
ZeroMQ	D-Bus D-Bus D-Bus / D-Bus / D-Bus	D-Bus	D-Bus D-Bus	D-Bus D-Bus D-Bus	D-Bus D-Bus D-Bus

## Modbus

- Modbus RTU RS-485
- Modbus TCP TCP/IP

### 1.

- RS-485 PLC
- Modbus RTU
- Modbus TCP

### 2.

- PLC
- Ethernet
- IPv4
- TCP
- HTTP MQTT

### 3.

- Web

- 数据源
  - 有线网络 Wi-Fi
  - 无线网络 Wi-Fi
  - 网络 IPv4/IPv6
  - 传输 TCP
  - 加密 TLS HTTPS
  - 接口 REST API WebSocket

数据源

- 数据源 数据源
- Modbus 数据源 RTU TCP
- IPC 数据源 IPC
- 数据源 数据源





## propdb

```
public int MyProperty
{
    get { return (int)GetValue(MyPropertyProperty); }
    set { SetValue(MyPropertyProperty, value); }
}
```

// Using a DependencyProperty as the backing store for MyProperty. This enables animation, styling, binding, etc...

```
public static readonly DependencyProperty MyPropertyProperty =
    DependencyProperty.Register("MyProperty", typeof(int), typeof(ownerclass), new PropertyMetadata(0));
```



## propa

```
public static int GetMyProperty(DependencyObject obj)
{
    return (int)obj.GetValue(MyPropertyProperty);
}
```

```
public static void SetMyProperty(DependencyObject obj, int value)
{
    obj.SetValue(MyPropertyProperty, value);
}
```

// Using a DependencyProperty as the backing store for MyProperty. This enables animation, styling, binding, etc...

```
public static readonly DependencyProperty MyPropertyProperty =
    DependencyProperty.RegisterAttached("MyProperty", typeof(int), typeof(ownerclass), new
PropertyMetadata(0));
```

# xaml[ ] [ ] [ ] [ ] [ ]

属性	值	类型	用途
Style	MyButtonStyle	Button	按钮样式
Style	CloseButtonStyle	Button	关闭按钮样式
Style	CloseButtonStyle	Button	关闭按钮样式
Style	CloseButtonStyle	Button	关闭按钮样式
Style	CloseButtonStyle	Button	关闭按钮样式
Style	CloseButtonStyle	Button	关闭按钮样式
Style	CloseButtonStyle	Button	关闭按钮样式
Style	CloseButtonStyle	Button	关闭按钮样式
Style	CloseButtonStyle	Button	关闭按钮样式
Style	CloseButtonStyle	Button	关闭按钮样式

```
//关闭按钮
<Style x:Key="MyButtonStyle" TargetType="Button" />

//关闭按钮
<Style TargetType="Button">
    <Setter Property="Background" Value="LightBlue" />
</Style>

//关闭按钮
public static class ResourceKeys
{
    public static readonly string CloseButtonStyle = "CloseButtonStyle";
}

<Style x:Key="{x:Static local:ResourceKeys.CloseButtonStyle}" TargetType="Button" />

//关闭按钮    ComponentResourceKey

public partial class DataTemplateKeys
{
    public static ComponentResourceKey ItemClose => new
ComponentResourceKey(typeof(DataTemplateKeys), "S.DataTemplate.Item.Close");
}

<DataTemplate x:Key="{ComponentResourceKey ResourceId=S.DataTemplate.Item.Close,
TypeInTargetAssembly={x:Type local:DataTemplateKeys}}">
```

```
//[1111111111]
```

```
public static class ResourceKeys
```

```
{
```

```
    public static readonly ComponentResourceKey CloseButtonStyleKey = new  
ComponentResourceKey(typeof(ResourceKeys), "CloseButtonStyle");
```

```
}
```

```
<Style x:Key="{x:Static local:ResourceKeys.CloseButtonStyleKey}" TargetType="Button" />
```

```
//[1111]
```

```
<Button Style="{DynamicResource MyButtonStyle}" />
```



```

:: if Lib Dll
IF NOT EXIST "$(TargetDir)libs" (
    MD "$(TargetDir)libs"
)

:: Copy dll xml pdb config files to libs
move "$(TargetDir)*.dll" "$(TargetDir)libs"
move "$(TargetDir)*.xml" "$(TargetDir)libs"
move "$(TargetDir)*.pdb" "$(TargetDir)libs"
move "$(TargetDir)*.config" "$(TargetDir)libs"

:: Copy runtimes files to libs
move "$(TargetDir)runtimes" "$(TargetDir)libs"

:: Copy NLog.config to libs
move "$(TargetDir)libs\NLog.config" "$(TargetDir)NLog.config"
move "$(TargetDir)libs\$(ProjectName).exe.config" "$(TargetDir)\$(ProjectName).exe.config"
move "$(TargetDir)libs\$(ProjectName).exe.xml" "$(TargetDir)\$(ProjectName).exe.xml"
move "$(TargetDir)libs\$(ProjectName).pdb" "$(TargetDir)\$(ProjectName).pdb"

```



```
<runtime>  
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">  
    <!-- [ ] libs[ ] -->  
    <probing privatePath="libs"/>  
  </assemblyBinding>  
</runtime>
```



## HttpClient

```
try
{
    var lcaolSelectTeam = await this.ToGetSelectTeam();
    if (lcaolSelectTeam.Code != 20000)
    {
        throw new Exception(lcaolSelectTeam.Message);
    }
    var localToken = await this.ToGetUserToken();
    if (localToken.Code != 20000)
    {
        throw new Exception(localToken.Message);
    }
    var taskworkFloderBody = await this.ToGetTaskworkProxyFloder();
    if (taskworkFloderBody.Code != 20000)
    {
        throw new Exception(taskworkFloderBody.Message);
    }
    var rootTaskworkFloder = taskworkFloderBody.Data.PathCombine(teamTaskwrokId.ToString());
    HttpClientHandler handler = new HttpClientHandler();
    ProgressMessageHandler progressMessageHandler = new ProgressMessageHandler(handler);
    progressMessageHandler.HttpSendProgress += (sender, e) =>
    {
        action.Invoke(e.ProgressPercentage);
    };
    using (HttpClient httpClient = new HttpClient(progressMessageHandler))
    {
        httpClient.BaseAddress = new Uri("https://lycg.lingyanspace.com/");
        httpClient.DefaultRequestHeaders.Add("Authorization", localToken.Data);
        using (var multipartFormData = new MultipartFormDataContent())
        {
            var bom = rootTaskworkFloder.PathCombine("bom").FileCombine("default.json");
            if (File.Exists(bom) && needUploadCloudModel.BOM)
```



```

{
    AddFile(multipartFormData, "bom", bom);
}

var blfc = rootTaskworkFloder.PathCombine("blfc").FileCombine("default.ifc");
if (File.Exists(blfc) && needUploadCloudModel.BIFC)
{
    AddFile(multipartFormData, "blfc", blfc);
}

var nc1Files = Directory.GetFiles(rootTaskworkFloder.PathCombine("nc1"), "*.nc1",
SearchOption.TopDirectoryOnly).ToList();
if (nc1Files != null && nc1Files.Count > 0 && needUploadCloudModel.NC1)
{
    nc1Files.ForEach(f =>
    {
        AddFile(multipartFormData, "nc1Files", f);
    });
}

var dxfFiles = Directory.GetFiles(rootTaskworkFloder.PathCombine("dxf"), "*.dxf",
SearchOption.TopDirectoryOnly).ToList();
if (dxfFiles != null && dxfFiles.Count > 0 && needUploadCloudModel.DXF)
{
    dxfFiles.ForEach(f =>
    {
        AddFile(multipartFormData, "dxfFiles", f);
    });
}

var aifcFiles = Directory.GetFiles(rootTaskworkFloder.PathCombine("aifc"), "*.ifc",
SearchOption.TopDirectoryOnly).ToList();
if (aifcFiles != null && aifcFiles.Count > 0 && needUploadCloudModel.AIFC)
{
    aifcFiles.ForEach(f =>
    {
        AddFile(multipartFormData, "aifcFiles", f);
    });
}

var pifcFiles = Directory.GetFiles(rootTaskworkFloder.PathCombine("pifc"), "*.ifc",
SearchOption.TopDirectoryOnly).ToList();
if (pifcFiles != null && pifcFiles.Count > 0 && needUploadCloudModel.PIFC)
{
    pifcFiles.ForEach(f =>

```

```

        {
            AddFile(multipartFormData, "pifcFiles", f);
        });
    }

    var drawingFiles = Directory.GetFiles(rootTaskworkFloder.PathCombine("drawing"), "*.pdf",
SearchOption.TopDirectoryOnly).
        Concat(Directory.GetFiles(rootTaskworkFloder.PathCombine("drawing"), "*.dwg",
SearchOption.TopDirectoryOnly)).ToList();

    if (drawingFiles != null && drawingFiles.Count > 0 && needUploadCloudModel.Drawing)
    {
        drawingFiles.ForEach(f =>
        {
            AddFile(multipartFormData, "drawingFiles", f);
        });
    }

    var response = await
httpClient.PutAsync($"/api/Team/UploadTeamTaskworkBatchData?teamId={IcaolSelectTeam.Data.Id}&teamTas
kwrokId={teamTaskwrokId}", multipartFormData);

    if (response.IsSuccessStatusCode)
    {
        var data = await response.Content.ReadAsStringAsync();
        var jsonBody = JsonConvert.DeserializeObject<ResponseBody<string>>(data);
        return jsonBody;
    }
    else
    {
        throw new Exception(await response.Content.ReadAsStringAsync());
    }
}

}

catch (Exception ex)
{
    return new ResponseBody<string>(40000, ex.Message, null);
}

```



```

public class StringSortComparer : IComparer<string>
{
    public bool MatchCase { get; }
    public StringSortComparer(bool matchCase)
    {
        MatchCase = matchCase;
    }
    private int CharCompare(char a, char b, bool matchCase)
    {
        char _a = char.MinValue, _b = char.MinValue;
        if (matchCase) { _a = a; _b = b; }
        else { _a = char.ToUpper(a); _b = char.ToUpper(b); }
        if (_a > _b) return 1;
        if (_a < _b) return -1;
        return 0;
    }
    public int Compare(string x, string y)
    {
        // [] y [][] [] y [][] []
        if (string.IsNullOrEmpty(y)) return -1;
        // [] x [][] [] y [][] [] x [][] y []
        if (string.IsNullOrEmpty(x)) return 1;
        int len;
        if (x.Length > y.Length) len = x.Length;
        else len = y.Length;
        string numericx = "";
        string numericy = "";
        for (int i = 0; i < len; i++)
        {
            char cx = char.MinValue;
            char cy = char.MinValue;
            if (i < x.Length) cx = x[i];
            if (i < y.Length) cy = y[i];
            if (cx >= 48 && cx <= 57) numericx += cx;
            if (cy >= 48 && cy <= 57) numericy += cy;
            if (i == len - 1)
            {
                if (numericx.Length > 0 && numericy.Length > 0)

```

```

    {
        if (decimal.Parse(numericx) < decimal.Parse(mericy)) return -1;
        if (decimal.Parse(numericx) > decimal.Parse(mericy)) return 1;
    }
    return CharCompare(cy, cy, MatchCase);
}
if ((cx >= 48 && cx <= 57) && (cy >= 48 && cy <= 57)) continue;
if (numericx.Length > 0 && mericy.Length > 0)
{
    if (decimal.Parse(numericx) < decimal.Parse(mericy)) return -1;
    if (decimal.Parse(numericx) > decimal.Parse(mericy)) return 1;
}
if (CharCompare(cx, cy, MatchCase) == 0) continue;
return CharCompare(cx, cy, MatchCase);
}
return 0;
}
}

```



```

public class HttpHelper
{
    /// <summary>
    /// 
    /// </summary>
    /// <param name="action"></param>
    /// <param name="netWrokUrl"></param>
    /// <param name="localUrl"></param>
    /// <returns></returns>
    /// <exception cref="Exception"></exception>
    public static async Task<long> DownloadSingleFile(Action<double> action, string netWrokUrl, string
localUrl)
    {
        long totalBytesReceived = 0;
        var progress = new Progress<HttpDownloadProgress>(p =>
        {
            if (p.TotalBytesToReceive.HasValue)

```

```

    {
        totalBytesReceived = (long)p.BytesReceived;
        double percent = (double)p.BytesReceived / p.TotalBytesToReceive.Value * 100.0;
        action.Invoke(percent);
    }
    else
    {
        LoggerHelper.DefaultLogger($"[INFO] {netWrokUrl} TotalBytesToReceive{totalBytesReceived} ");
    }
});
var fileBytes = await new HttpClient().GetByteArrayAsync(new Uri(netWrokUrl), progress,
CancellationToken.None);
if (File.Exists(localUrl))
{
    File.Delete(localUrl);
}
await localUrl.SaveLocalFileAsync(new MemoryStream(fileBytes));
return totalBytesReceived;
}

private static async Task<long> DownloadSingleFile(Action<long, long> progressAction, string networkUrl,
string localUrl, long totalBytes)
{
    long bytesReceived = 0;
    var progress = new Progress<HttpDownloadProgress>(p =>
    {
        bytesReceived = (long)p.BytesReceived;
        progressAction(bytesReceived, totalBytes);
    });
    using (var httpClient = new HttpClient())
    {
        var fileBytes = await httpClient.GetByteArrayAsync(new Uri(networkUrl), progress,
CancellationToken.None);
        if (File.Exists(localUrl))
        {
            File.Delete(localUrl);
        }
        using (var fileStream = new FileStream(localUrl, FileMode.CreateNew))
        {
            await fileStream.WriteAsync(fileBytes, 0, fileBytes.Length);
        }
    }
}

```

```

        return bytesReceived;
    }
}
/// <summary>
/// 
/// </summary>
/// <param name="overallProgressAction"></param>
/// <param name="files"></param>
/// <returns></returns>
public static async Task DownloadMultipleFiles(Action<double> overallProgressAction, Dictionary<string,
string> files)
{
    var downloadTasks = new List<Task<long>>();
    long totalFileSize = 0;
    Dictionary<string, long> fileSizes = new Dictionary<string, long>();
    // 
    HEAD
    using (var httpClient = new HttpClient())
    {
        foreach (var file in files)
        {
            var response = await httpClient.SendAsync(new HttpRequestMessage(HttpMethod.Head, file.Key));
            long contentLength = response.Content.Headers.ContentLength ?? 0;
            fileSizes[file.Key] = contentLength;
            totalFileSize += contentLength;
        }
    }
    // 
    Dictionary<string, long> receivedBytes = new Dictionary<string, long>();
    foreach (var file in files)
    {
        string networkUrl = file.Key;
        string localUrl = file.Value;
        Task<long> downloadTask = DownloadSingleFile(
            (bytesReceived, totalBytes) =>
            {
                receivedBytes[networkUrl] = bytesReceived;
                // 
                long totalReceived = 0;
                foreach (var received in receivedBytes.Values)
                {

```

```

        totalReceived += received;
    }
    double overallProgress = (double)totalReceived / totalFileSize * 100.0;
    overallProgressAction(overallProgress);
},
networkUrl,
localUrl,
fileSizes[networkUrl]
);
downloadTasks.Add(downloadTask);
}
// ██████████
long[] results = await Task.WhenAll(downloadTasks);
}
}

```

## byte

```

public static class HttpClientExtension
{
    private const int BufferSize = 262144;

    public static async Task<byte[]> GetByteArrayAsync(this HttpClient client, Uri requestUri,
IProgress<HttpDownloadProgress> progress, CancellationToken cancellationToken)
    {
        if (client == null)
        {
            throw new ArgumentNullException(nameof(client));
        }

        using (var responseMessage = await client.GetAsync(requestUri,
HttpCompletionOption.ResponseHeadersRead, cancellationToken).ConfigureAwait(false))
        {
            responseMessage.EnsureSuccessStatusCode();

            var content = responseMessage.Content;
            if (content == null)
            {

```

```

        return Array.Empty<byte>();
    }

    var headers = content.Headers;
    var contentLength = headers.ContentLength;
    using (var responseStream = await content.ReadAsStreamAsync().ConfigureAwait(false))
    {
        var buffer = new byte[BufferSize];
        int bytesRead;
        var bytes = new List<byte>();

        var downloadProgress = new HttpDownloadProgress();
        if (contentLength.HasValue)
        {
            downloadProgress.TotalBytesToReceive = (ulong)contentLength.Value;
        }
        progress?.Report(downloadProgress);

        while ((bytesRead = await responseStream.ReadAsync(buffer, 0, BufferSize,
cancellationTokens).ConfigureAwait(false)) > 0)
        {
            bytes.AddRange(buffer.Take(bytesRead));

            downloadProgress.BytesReceived += (ulong)bytesRead;
            progress?.Report(downloadProgress);
        }

        return bytes.ToArray();
    }
}

public struct HttpDownloadProgress
{
    public ulong BytesReceived { get; set; }

    public ulong? TotalBytesToReceive { get; set; }
}

```