PC wpf

nuget工具篇

nuget包工具命令

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

课纲目录

模块 I：WPF高阶技术精讲

深入掌握自定义控件

主要是附加属性与控件加载部分

深入掌握控件模板与数据模板

理解控件模板当中包含数据模板

数据模板当中又可以包含控件模板

来回组合学习

最终以模板选择器来弥补遗漏的部分

深入掌握资源样式与动态主题

资源字典如何加载

静态资源与动态资源区别

主题动态切换核心理念

以及后续写项目的规范

埋坑

此处本来是想带大家完完全全手写一个 wpf控件库

但是由于很多技术涉及到 c#代码，所以先讲后续，高阶通信模块讲完之后再返回这儿

然后还有 MAUI部分公开课要把讲 xaml基础控件正好中间多学点儿做个过渡。

但是主要的 xaml技术就是这三节内容和 B站 wpf公开课那一套视频看完

模块 II：高阶通信与高阶模式加并发控制

基础知识

1[illegible]

2[illegible]

3[illegible]

4[illegible]

5[illegible]

6[illegible]

7[illegible] C# [illegible]

8[illegible] Thread[illegible] Task[illegible]

9[illegible]

10[illegible] WPF [illegible]

11[illegible] UI [illegible]

12[illegible] Dispatcher[illegible] UI[illegible]

[illegible]

1[illegible] Span<T> [illegible] Span<T>[illegible]

2[illegible] CancellationToken[illegible]

3[illegible] WPF [illegible] WPF [illegible]

[illegible]

1[illegible] TcpListener [illegible]

[illegible] TCP [illegible]

[illegible]

2[illegible] SignalR [illegible]

[illegible] SignalR [illegible]

3[illegible] WPF [illegible]

[illegible] WPF [illegible]

[illegible]

[illegible]

1[illegible] MonitorMutex,Semaphore[illegible] Dispatcher[illegible] BackgroundWorker[illegible]

2[illegible] async/await[illegible] TaskScheduler[illegible]

3[illegible] TPL Dataflow [illegible] -[illegible]

4[illegible] WPF [illegible] WPF [illegible] UI [illegible]

[illegible]

1[illegible] TPL [illegible]

2[illegible] Span<T>[illegible] Memory<T>[illegible]

3、WPF 中的应用：在 WPF 中实现高性能的数据流处理，示例：实时处理传感器数据并在 UI 上显示。

高级通信技术模块

1、WebSocket 通信：使用 System.Net.WebSockets实现 WebSocket 客户端和服务端。

2、SignalR 实时通信：构建基于 SignalR 的实时应用

3、跨平台通信：使用 gRPC 或 RESTful API 实现跨平台通信。

4、WPF 中的应用：在 WPF 中集成 WebSocket 或 SignalR，实现一个实时更新的仪表盘或聊天界面。

性能优化与调试模块

1、常见问题与解决方案：死锁与竞争条件。内存泄漏与资源耗尽。

2、WPF 中的性能优化：减少 UI 线程负担。使用虚拟化技术优化列表显示。

实践巩固

实际项目实践模块

目标：

通过实际项目巩固所学知识。

项目主题：

实现一个简单的聊天应用。

构建一个实时监控系统。

功能要求：

支持多线程和异步通信。

使用协议解析和错误处理机制。

集成 SignalR 或 WebSocket。

WPF 界面开发：

设计一个用户友好的界面。

实现动态更新和多线程交互。

聊天应用：

使用 SignalR 实现实时消息传递。

使用 Dispatcher更新聊天记录。

监控系统：

使用 TPL Dataflow 处理传感器数据。

使用 TaskScheduler优化任务调度。

模块 **III**：高阶项目实战（全栈）

3.1 综合项目案例

3.1.1 虚拟监控系统

- 使用 **ICollectionView**实现动态数据过滤
- 基于 **VisualStateManager**的报警状态可视化

3.1.2 日志与权限管理

- 使用 **NLog**实现日志分级（**Debug/Info/Error**）
- 基于角色的权限系统（**RBAC**）

3.2 精细化案例

3.2.1 数据可视化

- 使用 **OxyPlot**实现动态波形图
- 基于 **WriteableBitmap**的实时图像处理

模块 **IV**：前沿技术与扩展

4.1 跨平台开发

4.1.1 .NET MAUI深度集成

- 共享业务逻辑层与 **UI**分离设计
- 使用 **SkiaSharp**实现跨平台绘图

4.1.2 WPF与 **Web**技术结合

- 嵌入 **WebView2**实现混合开发
- 使用 **WebAssembly**与 **Blazor**交互

4.2 人工智能集成

4.2.1 机器学习模型集成

- 使用 **ML.NET**实现本地预测
- 基于 **ONNX**的图像识别

4.2.2 数据分析与可视化

- 使用 **LiveCharts**实现动态仪表盘
- 基于 **Parallel.For**的并行数据处理

打包篇

inno

中文环境【 ChineseSimplified.isl】

ChineseSimplified.isl

isl环境中文

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

完整的通信链路

OSI 七层模型与协议对照表

层次	功能	常见协议 /标准	工业协议
物理层	定义硬件接口、电气特性、信号传输方式。	RS-232、RS-485、USB、Ethernet、CAN、Wi-Fi、蓝牙、ZigBee、LoRa、NFC	Modbus RTU（基于 RS-485）、CANopen（基于 CAN）
数据链路层	提供可靠的数据帧传输，处理错误检测和纠正。	Ethernet（IEEE 802.3）、Wi-Fi（IEEE 802.11）、PPP、HDLC	Profinet、EtherCAT
网络层	负责路由选择和逻辑地址分配，确保数据包跨网络传输。	IPv4、IPv6、ICMP、ARP	Modbus TCP（基于 IP）、CIP（Common Industrial Protocol）
传输层	提供端到端的通信服务，负责数据分段、重组和流量控制。	TCP、UDP	MQTT（基于 TCP）、CoAP（基于 UDP）
会话层	管理会话（连接的建立、维护和终止）。	SMB、RPC	OPC UA
表示层	数据格式转换、加密解密、数据压缩。	TLS/SSL、JSON、XML	无特定工业协议，但加密和数据格式在工业通信中广泛使用。
应用层	为用户提供直接的网络服务，定义应用程序之间的通信规则。	HTTP/HTTPS、FTP/SFTP、SMTP/IMAP/POP3、WebSocket、gRPC	Modbus TCP（基于 TCP）、OPC UA、BACnet（楼宇自动化控制网络）

IPC（进程间通信）方式与层次对照表

通信方式	功能	适用层次	应用场景	优点	缺点
匿名管道	单向或双向通信，适用于父子进程之间的数据传递。	操作系统层（不属于 OSI 模型，但常用于应用层之上）。	本地进程间通信，适用于简单的数据传递。	简单易用，操作系统原生支持。	仅限本地通信，数据量较小，父子进程关系限制。
命名管道	支持无亲缘关系的进程间通信，支持本地或网络通信。	操作系统层（不属于 OSI 模型，但常用于应用层之上）。	本地或网络进程间通信，适用于跨进程的数据传递。	支持跨进程和跨网络通信，灵活性高。	实现复杂度较高，性能受限于操作系统。

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

D-Bus	Linux 系统中用于进程间通信的消息总线。	操作系统层（不属于 OSI 模型，但常用于应用层之上）。	Linux 系统中的进程间通信（如桌面环境组件之间的通信）。	高效，支持广播和点对点通信，适合 Linux 环境。	仅适用于 Linux 系统，跨平台支持较差。
ZeroMQ	高性能消息队列库，支持多种通信模式（如发布/订阅、请求/响应）。	应用层	分布式系统中的高性能通信。	高性能，支持多种通信模式，跨平台支持好。	需要手动管理消息格式和连接，学习曲线较陡峭。

补充说明

关于 **Modbus** 的位置说明

- **Modbus RTU**：基于 RS-485 的物理层和数据链路层协议，主要用于工业现场设备之间的通信。
- **Modbus TCP**：基于 TCP/IP 的应用层协议，适用于工业自动化中设备与服务器之间的通信。

协议的组合应用示例

1. 下位机与上位机

- 场景：工业现场的传感器（下位机）通过 RS-485 与 PLC（上位机）通信。
- 协议组合：
  - 物理层：RS-485。
  - 数据链路层：Modbus RTU。
  - 应用层：Modbus 协议解析传感器数据。

2. 上位机与远程服务器

- 场景：PLC（上位机）通过以太网将数据上传到远程服务器。
- 协议组合：
  - 物理层：Ethernet。
  - 数据链路层：以太网帧。
  - 网络层：IPv4。
  - 传输层：TCP。
  - 应用层：HTTP 或 MQTT。

3. 远程服务器与前端客户

- 场景：远程服务器通过 Web 服务向前端客户提供实时数据。
- 协议组合：

  - 物理层：光纤或 Wi-Fi。
  - 数据链路层：以太网或 Wi-Fi。
  - 网络层：IPv4/IPv6。
  - 传输层：TCP。
  - 表示层：TLS（HTTPS）。
  - 应用层：REST API 或 WebSocket。

总结

- 清晰的分层职责：每一层都有明确的功能和协议，避免混淆。
- **Modbus** 的位置：RTU 属于物理层和数据链路层，TCP 属于应用层。
- **IPC** 的层次划分：大部分 IPC 通信方式位于操作系统层，但也可以延伸到传输层和应用层。
- 协议组合的灵活性：根据场景需求，协议可以跨层组合使用，形成完整的通信链路。

□□□

□□□ 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□□□□

定义方式	优点	缺点	适用场景
普通字符串键	简单直观，易于使用	命名冲突风险，无法跨程序集 共享	小型项目，局部资源
类型键	自动应用，无需显式引用	灵活性较低	全局样式，基础样式复用
静态资源键	强类型支持，可维护性高	定义稍复杂	大型项目，组件化开发
`ComponentResourceKey`	跨程序集支持，语义化标识	定义和使用复杂	组件库开发，主题或样式库
动态资源键	动态绑定，灵活性高	性能开销	主题切换，多语言支持

```
//普通字符串键
<Style x:Key="MyButtonStyle" TargetType="Button" />


//类型键（隐式样式）
<Style TargetType="Button">
    <Setter Property="Background" Value="LightBlue" />
</Style>


//静态资源键
public static class ResourceKeys
{
    public static readonly string CloseButtonStyle = "CloseButtonStyle";
}


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


//组件资源键         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}}">


//静态资源键与组件资源键结合


public static class ResourceKeys
{
```

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

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

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

编译篇

编译后事件

```
:: 检查  Lib、Dll文件夹路径是否存在
IF NOT EXIST "$(TargetDir)libs" (
    MD "$(TargetDir)libs"
)

:: 将指定的  dll、xml、pdb、config文件移动到  libs文件夹
move "$(TargetDir)*.dll" "$(TargetDir)libs"
move "$(TargetDir)*.xml" "$(TargetDir)libs"
move "$(TargetDir)*.pdb" "$(TargetDir)libs"
move "$(TargetDir)*.config" "$(TargetDir)libs"

:: 将 runtimes文件夹移动到  libs文件夹
move "$(TargetDir)runtimes" "$(TargetDir)libs"

:: 把主程序的相关文件从  libs转移出来
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"
```

加扫描文件夹

```
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <!-- 添加对  libs文件夹的搜索路径  -->
    <probing privatePath="libs"/>
  </assemblyBinding>
</runtime>
```

□□□

HttpCilent□□□□□

```
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 bIfc = rootTaskworkFloder.PathCombine("bifc").FileCombine("default.ifc");
                if (File.Exists(bIfc) && needUploadCloudModel.BIFC)
                {
                    AddFile(multipartFormData, "bIfc", bIfc);
                }
                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={lcaolSelectTeam.Data.Id}&teamTas
kwrokId={teamTaskwrokId}", multipartFormData);
                if (response.IsSuccessStatusCode)
                {
                    var data = await response.Content.ReadAsStringAsync();
                    var jsonBody = JsonConvert.DeserializeObject<ResponceBody<string>>(data);
                    return jsonBody;
                }
                else
                {
                    throw new Exception(await response.Content.ReadAsStringAsync());
                }
            }
        }
        catch (Exception ex)
        {
            return new ResponceBody<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)
    {
        // 如果 y 为空，则 y 应该排在最后面
        if (string.IsNullOrEmpty(y)) return -1;
        // 如果 x 为空，而 y 不为空，则 x 应该排在 y 之前
        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(numericy)) return -1;
                    if (decimal.Parse(numericx) > decimal.Parse(numericy)) return 1;
                }
                return CharCompare(cy, cy, MatchCase);
            }
            if ((cx >= 48 && cx <= 57) && (cy >= 48 && cy <= 57)) continue;
            if (numericx.Length > 0 && numericy.Length > 0)
            {
                if (decimal.Parse(numericx) < decimal.Parse(numericy)) return -1;
                if (decimal.Parse(numericx) > decimal.Parse(numericy)) 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($"□□□□□        {netWrokUrl}□ TotalBytesToReceive□□   ");
            }
        });
        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>
        /// 下载文件集合
        /// </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>();
            // 首先，预估所有文件的大小（可以通过                    HEAD请求或是其他方式获取）
            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;
                }
            }
            // 存储每个文件的已接收字节
            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;
                        // 计算总体进度
                        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,
cancellationToken).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; }
    }
```

WPF大基本功

Application的生命周期

OnStartup: 表示启动应用程序时

OnActivated: 表示激活应用程序时

OnDeactivated: 表示由激活状态变为非激活状态时

OnExit: 表示退出应用程序时

Window窗体的生命周期

SourceInitialized创建窗体源时引发此事件

Activated当前窗体成为前台窗体时引发此事件

Loaded当前窗体内部所有元素完成布局和呈现时引发此事件

ContentRendered当前窗体的内容呈现之后引发此事件

------Deactivated当前窗体成为后台窗体时引发此事件 ====Activated当前窗体成为前台窗体时引发此事件

Closing当前窗体关闭之前引发此事件

Deactivated当前窗体成为后台窗体时引发此事件

Closed当前窗体关闭之后引发此事件

Unloaded当前窗体从元素树中删除时引发此事件

ContentControl

Content属性在 ContentControl类中

加载和解析 XAML

编译器会将 XAML 文件（如 MainWindow.xaml）编译为 BAML (Binary Application Markup Language)

，这是一种压缩和优化的二进制格式。

运行时， CLR 会加载 BAML 并将其解析回一个对象树。这个过程会:

实例化 XAML 中定义的每一个元素（如 <Window>， <Grid>， <Button>）为 .NET 对象。

设置每个对象的属性（如 Width, Height, Content）。

根据元素的嵌套关系，建立起逻辑树 (Logical Tree)。

对象初始化过程中，会按顺序引发一系列事件

Initialized: 元素已被初始化并从 XAML 中加载了基本属性。此时逻辑树已建立，但尚未进行布局测量和排列。

Loaded:

[illegible]

- **[illegible] (Logical Tree)**[illegible] XAML [illegible]
- **[illegible] (Visual Tree)**[illegible] `VisualTreeHelper` [illegible]
- **[illegible] (Layout System)**[illegible] **(Measure) -> [illegible] (Arrange)** [illegible] (Layout Pass)[illegible]
- **[illegible] (Retained Mode Graphics)**[illegible] “[illegible]”[illegible] WinForms [illegible] `Paint` [illegible] WPF [illegible] “[illegible]” [illegible] “[illegible]”[illegible] “[illegible]”[illegible]

[illegible]

```
Button->ButtonBase->ContentControl->Control->FrameworkElement->UIElement->Visual->DependencyObject-
>DispatcherObject
StackPanel->Panel->FrameworkElement->UIElement->Visual->DependencyObject->DispatcherObject
Rectangle->Shape->FrameworkElement->UIElement->Visual->DependencyObject->DispatcherObject
```

WPF[illegible] UI[illegible]

DispatcherObject[illegible] Dispatcher[illegible] Invoke[illegible] BeginInvoke[illegible] UI [illegible]

C++音视频

对齐大厂音视频岗面试标准，我将 W1~W4 的技术栈、C++/QML 分工、具体用途、关键实现点拆解到底。原则只有一个：媒体管线 **100% C++**，**QML** 只做 **UI** 与状态展示。

核心分工原则（面试必讲）

层级	语言	职责	绝不越界
Media Engine	C++	拉流、解复用、解码、AVSync、JitterBuffer、硬解管理、OpenGL 渲染线程	不碰 `QQuickItem`，不依赖 Qt 事件循环
UI / Business	QML	播放控制、进度条、状态提示、调试面板、主题切换	不读 `AVFrame`，不调 `avcodec_*`，不阻塞主线程
Bridge	C++ (QObject)	`Q_PROPERTY` 暴露状态、`Q_INVOKABLE` 暴露控制接口、`QQuickFramebufferObject` 渲染视频纹理	仅传递数据指针/ID，不跨线程传递裸对象

W1~W4 逐周技术拆解

W1：FFmpeg 拉流 + 解码 + OpenGL 渲染跑通

技术	语言	用途	关键实现点
libavformat	C++	打开文件/网络流，读取 `AVPacket`，解析音视频流信息	`avformat_open_input` → `avformat_find_stream_info` → `av_read_frame`
libavcodec	C++	软解 H.264/H.265 视频、AAC/Opus 音频到 `AVFrame`	`avcodec_send_packet` / `avcodec_receive_frame` 异步非阻塞调用
libswscale	C++	像素格式转换（YUV420P → RGBA）仅作为 **CPU** 降级方案	实际渲染应弃用，改由 OpenGL Shader 做 YUV→RGB

技术	语言	用途	关键实现点
OpenGL 3.3+	C++	GPU 渲染视频帧: YUV 纹理上传、 Shader 转换、双缓冲	顶点着色器传 UV 坐标，片段着色器用 texture2D 采样 Y/U/V 平面，矩阵乘法转 RGB
std::thread / std::mutex / std::condition_variable	C++	解复用线程、解码线程、渲染线程隔离，线程安全队列	用 std::queue<AVPacket*> + 条件变量控制生产消费，避免内存碎片
QML	QML	仅显示 “播放 /暂停 ”按钮 + FPS 文本	通过 Q_PROPERTY 绑定 fps ，不干预渲染逻辑

□ □□□□ : 1080p MP4 循环播放稳定 60 FPS, valgrind / AddressSanitizer 零泄漏, CPU 占用 <15% （软解）。

□□ W2: 接入 SRT/UDP, 实现基础 AVSync

技术	语言	用途	关键实现点
FFmpeg SRT 协议支持 或 libsrt	C++	替换文件 I/O 为低延迟网络传输	FFmpeg 原生支持 srt:// 协议; 或 srt_create_socket + avio_alloc_context 注入自定义读写回调
SRTO_LATENCY / SRTO_TSBPDDELAY	C++	控制 SRT 缓冲延迟，抗网络抖动	设置 latency=120 （毫秒级），面试需解释 TSBPD（Time-Sender-Based Packet Delivery）原理
音频主时钟	C++	以音频播放进度为基准，计算音画时间差	audio_clock = sample_pos / sample_rate ; video_delay = video_pts - audio_clock
动态丢帧 /重复帧	C++	根据 video_delay 调整渲染节奏	delay > 40ms 丢帧; delay < -30ms 重复渲染上一帧; std::this_thread::sleep_until 精准睡眠
QAudioSink (Qt6) 或 PortAudio	C++	音频播放输出，提供高精度时钟源	回调函数中更新 audio_clock ，确保时钟与硬件播放进度严格一致
QML	QML	网络状态指示、延迟显示、手动同步开关	绑定 latency_ms 、 sync_status ，提供 UI 调试入口

□ □□□□ : 局域网 1v1 实时流，端到端延迟 ≤200ms, 音画同步误差 ≤±20ms （手机慢动作录制验证）。

□□ W3: 硬解 Fallback + 零拷贝渲染 + 弱网 JitterBuffer

技术	语言	用途	关键实现点
`AVHWDeviceContext` / `AVHWFramesContext`	C++	管理硬件解码上下文（VAAPI/DXVA2/VideoToolbox）	`av_hwdevice_ctx_create` → 绑定到 `AVCodecContext` → 捕获 `AVERROR(EAGAIN)` 降级软解
`PBO (Pixel Buffer Object)` 或 `EGLImageKHR`	C++	零拷贝渲染：GPU 解码帧直接映射为 OpenGL 纹理	`glGenBuffers` + `glMapBufferRange` 避免 `glTexImage2D` CPU-GPU 同步阻塞；Linux 用 `EGLImage` 直通
动态 Jitter Buffer	C++	抗弱网：根据 RTT/丢包率动态调整缓冲队列大小	滑动窗口计算 `jitter = max(inter_arrival) - min(inter_arrival)`；丢包 >10% 扩容 +插值，<3% 缩容降延迟
`PLC (Packet Loss Concealment)`	C++	音频丢包隐藏（基础版：线性插值 /静音填充）	音频帧丢失时，用上一帧末尾波形外推，避免“咔嗒”爆音
`tc qdisc` (Linux) / `Clumsy` (Win)	工具	模拟弱网：丢包、延迟、乱序、带宽限制	`tc qdisc add dev eth0 root netem loss 15% delay 50ms 20ms`
`perf` / `RenderDoc` / `nvidia-smi`	工具	性能剖析：CPU 热点、GPU 渲染管线、显存占用	`perf record -g ./app` → `perf report`；RenderDoc 抓帧验证零拷贝路径
`QML`	QML	实时性能面板：FPS、延迟、JitterBuffer 长度、硬解状态	使用 `ChartView` 或自定义 `Canvas` 绘制波形，绑定 C++ 信号

□ □□□□：压测报告（PDF/Markdown）含：15% 丢包卡顿率 <2%、硬解 CPU <8%、PBO 渲染无 `glReadPixels`、72h 内存波动 <3%。

□□ W4: QML 对接 + 状态机 + 日志 /Profiler 注入

技术	语言	用途	关键实现点
`QQuickFramebufferObject` 或 `QSGTexture`	C++	将 OpenGL 渲染上下文安全嵌入 QML 场景图	重写 `createRenderer()` 返回自定义 `Renderer`，在 `render()` 中绑定 FBO 或纹理，禁用 **`QQuickPaintedItem`**
`QStateMachine` (Qt6)	C++	播放生命周期管理：`IDLE → CONNECTING → STREAMING → RECONNECTING → ERROR`	状态迁移绑定网络事件 / 用户操作，避免 `if-else` 嵌套地狱

技术	语言	用途	关键实现点
spdlog 或 QtLogging	C++	结构化日志：时间戳、线程 ID、模块名、性能指标	SPDLOG_INFO("decode", "pts={} delay={}ms", pts, delay)；异步日志线程防阻塞
CMake + vcpkg/Conan	构建系统	跨平台依赖管理、编译选项优化（-O3、-march=native）	显式链接 FFmpeg::avformat 等 Modern CMake Target，避免硬编码路径
QML	QML	完整演示端：自定义按钮、主题切换、调试面板、状态反馈	使用你提供的 CustomButton，通过 Qt.binding() 响应引擎状态变化
draw.io / PlantUML	文档	架构图、线程模型、数据流图、状态迁移图	面试直接投屏讲解，标注线程边界与队列容量

□ □□□□：完整可运行 Demo + 架构图 + 压测报告 + perf/RenderDoc 截图 + 5 分钟演示视频。

技术栈归属速查表（面试防御用）

模块	C++	QML	说明
网络拉流 (SRT/UDP)	□	□	avio 回调或 libsrt 封装
解复用/解码 (FFmpeg)	□	□	纯 C++ 异步管线
音视频同步 (AVSync)	□	□	音频主时钟 + 丢帧/重复帧逻辑
硬解 Fallback	□	□	AVHWDeviceContext 管理
零拷贝渲染 (PBO/EGL)	□	□	OpenGL + QQuickFramebufferObject
Jitter Buffer / PLC	□	□	动态队列 + 音频插值
状态机 / 日志 / 性能探针	□	□	QStateMachine + spdlog
UI 控件 / 主题 / 动画	□	□	纯 QML，绑定 Q_PROPERTY
视频画面显示	⚠ C++ 渲染，QML 占位	□	QQuickItem 仅占位，纹理由 C++ 注入

面试怎么讲这套技术链？（话术模板）

> “项目采用 **C++** 媒体引擎 **+ QML** 表现层，严格分离架构。底层基于 FFmpeg 实现解复用与软硬解码管线，通过 AVHWDeviceContext 管理硬件解码器，失败时自动 Fallback 到软解。渲染端使用 PBO 零拷贝技术将 GPU 帧直接映射为 OpenGL 纹理，通过 QQuickFramebufferObject 安全注入 QML 场景图，彻底避开 CPU-GPU

> 同步瓶颈。
> 音视频同步采用 音频主时钟策略 ，根据 PTS 与硬件播放进度差值动态丢帧 / 重复帧，同步误差控制在 ±15ms。弱网层实现动态 Jitter Buffer，基于 RTT 方差与丢包率滑动窗口调整缓冲深度，配合基础 PLC 保证 15% 丢包下卡顿率 <2%。
> 全链路通过 `spdlog` 埋点与 `perf`/RenderDoc 压测，首帧 ≤450ms， 1080p@30fps 硬解 CPU <7%，内存 72h 压测波动 <3%。”

本项目升级路线 阶段一 基础播放管线：脱离 QtMultimedia，直接使用 FFmpeg API 实现解复用，软解码，音画同步，完成 MP4与 HLS基础播放 阶段二 硬件加速与渲染：接入 DXVA2或 NVDEC 实现硬解，完成 YUV到 RGB色彩空间转换，使用 OpenGL或 DirectX实现零拷贝渲染，优化首屏耗时 阶段三 流控与质量保障：实现 HLS 自适应码率切换逻辑，设计多级缓冲策略，添加网络状态监控，实现卡顿统计与播放质量数据上报 阶段四 工程化架构重构：将播放引擎抽离为独立 C 加加动态库，提供线程安全接口，实现配置热加载，日志分级，内存池管理，补充单元测试与压测脚本