□□□

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; }
    }
```

版本号 #2

由 叶子 创建于 5 三月 2025 13:50:18

由 叶子 更新于 5 三月 2025 13:50:49