

# V+ 远程用户管理开发文档

**SDK版本: 1.0.0**

适用框架: .NET Framework 4.5+ / .NET Core / .NET 5+

适用版本: V+ 4.3.1 rev.15251+

三方依赖: 无 (基于原生 System.Net.Sockets 与 System.Xml.Serialization)

## 1. 项目背景与设计理念

**VPClientSDK** 是专为 V+ 远程用户管理设计的开发包。它旨在简化客户端（如 WinForms/WPF 管理工具）对V+进行远程用户登陆注册的流程，封装了底层 TCP 通信、加密握手、多线程并发以及本地配置持久化等复杂逻辑。

### 核心特性

**原生轻量:** 完全基于 .Net Framework 原生框架, 不依赖任何第三方 NuGet 包, 易于集成到老旧或受限的开发环境中。

**同步阻塞 (Synchronous):** API 设计为同步调用, 逻辑直观。注意：调用方需自行处理线程（如 Task.Run）以防 UI 卡死。

**并发加速:** SDK 内部在处理扫描和批量操作时, 自动进行多线程加速。

**安全通信:** 全链路采用 AES-256-CBC 加密, 配合 SHA256 密钥派生。

**功能配置:** 内置 XML 持久化模块, 自动记忆扫描范围、历史服务器凭据等。

## 2. 快速开始

### 2.1 初始化 SDK

在使用任何功能前, 需要在你的 Form 或 ViewModel 中实例化 SDK。

```

using VP.SDK;

public class MainForm : Form
{
    private VPCClientSdk _sdk;

    public MainForm()
    {
        InitializeComponent();

        // 初始化 SDK
        // timeoutMs: 单次 TCP 连接/读取的超时时间 (默认 3000ms)
        _sdk = new VPCClientSdk(timeoutMs: 3000);
    }
}

```

## 3. 配置与持久化 (Configuration)

SDK 会在运行目录下自动维护 vp\_config.xml 文件。你可以通过 \_sdk.Config 访问相关数据。

### 3.1 获取/保存扫描设置

自动记忆用户上一次输入的 IP 范围和端口。

调用示例:

```

// [读取] 窗体加载时恢复数据
string lastStartIp = _sdk.Config.Data.LastStartIp; // e.g. "192.168.1.1"
string lastEndIp   = _sdk.Config.Data.LastEndIp;   // e.g. "192.168.1.255"
string lastAdmin   = _sdk.Config.Data.LastAdminPwd;

// [保存] 扫描开始前保存当前设置
_sdk.Config.UpdateScanSettings(
    "192.168.1.1",
    "192.168.1.255",
    "8080,9090",
    "admin123"
);

```

### 3.2 凭据管理 (自动填充)

用于实现“选择服务器自动填号”、“输入账号自动填密”的功能。

方法说明:

GetSavedUsernames(ip, port): 获取某服务器下所有保存过密码的用户名。

GetLastUsedUser(ip, port): 获取某服务器最后一次登录的用户名。

GetPassword(ip, port, username): 获取保存的密码 (Base64存储)。

AddOrUpdateCredential(...): 保存凭据 (通常在登录成功后自动调用, 无需手动调用)。

调用示例:

```
// 场景: 用户在表格中选中了某台服务器 (server)
// 1. 获取该服务器所有历史用户, 用于自动补全
List<string> historyUsers = _sdk.Config.GetSavedUsernames(server.IP, server.Port);

// 2. 获取上次使用的用户名, 自动填入输入框
string lastUser = _sdk.Config.GetLastUsedUser(server.IP, server.Port);
txtUsername.Text = lastUser;

// 3. (配合 TextChanged 事件) 根据用户名自动获取密码
string savedPwd = _sdk.Config.GetPassword(server.IP, server.Port, txtUsername.Text);
if (!string.IsNullOrEmpty(savedPwd))
{
    txtPassword.Text = savedPwd; // 自动填充密码
}
```

## 4. 网络发现 (Discovery)

扫描指定 IP 范围内的所有 VP 服务器。支持跨网段扫描。

**List Discovery(string startIp, string endIp, int[] ports, CancellationToken token)**

调用示例:

```

// 准备取消令牌 (用于中途停止扫描)
var cts = new CancellationTokenSource();

// 注意：必须在后台线程运行！
Task.Run(() =>
{
    try
    {
        // 开始扫描 192.168.1.1 到 192.168.2.255
        var servers = _sdk.Discovery(
            "192.168.1.1",
            "192.168.2.255",
            new int[] { 8080, 9090 },
            cts.Token
        );

        // 更新 UI
        this.Invoke(new Action(() => {
            dataGridView.DataSource = servers;
            Console.WriteLine($"发现 {servers.Count} 台服务器");
        }));
    }
    catch (OperationCanceledException)
    {
        Console.WriteLine("扫描已中止");
    }
});

```

## 5. 单机用户管理

针对单台服务器的指令操作。这些方法是同步阻塞的，请勿在 UI 线程直接调用。

### 5.1 注册用户 (UserRegister)

**string UserRegister(string ip, int port, string adminPwd, string newUser, string newPwd)**

调用示例:

```
Task.Run(() => {
    string result = _sdk.UserRegister("192.168.1.100", 8080, "admin", "alice", "123456");
    if (result == "VP_OK")
        Console.WriteLine("注册成功");
    else
        Console.WriteLine($"注册失败: {result}");
});
```

## 5.2 切换用户/登录 (UserSwitch)

**string UserSwitch(string ip, int port, string adminPwd, string targetUser, string targetPwd)**

调用示例:

```
Task.Run(() => {
    string result = _sdk.UserSwitch("192.168.1.100", 8080, "admin", "alice", "123456");
    if (result == "VP_OK")
        Console.WriteLine("登录成功, 当前用户已切换");
});
```

## 5.3 注销 (UserLogout)

**string UserLogout(string ip, int port, string adminPwd)**

调用示例:

```
Task.Run(() => {
    _sdk.UserLogout("192.168.1.100", 8080, "admin");
    Console.WriteLine("注销指令已发送");
});
```

## 5.4 获取在线用户列表 (GetUserList)

**List GetUserList(string ip, int port, string adminPwd)**

调用示例:

```
Task.Run(() => {  
    try {  
        var users = _sdk.GetUserList("192.168.1.100", 8080, "admin");  
        foreach(var u in users) Console.WriteLine($"在线用户: {u}");  
    } catch (Exception ex) {  
        Console.WriteLine($"获取失败: {ex.Message}");  
    }  
});
```

## 6. 批量操作 (Batch Operations)

SDK 内部封装了 `Parallel.ForEach`，可同时对数十台服务器执行指令。

通用参数说明

`targets` : `List<VpsServerInfo>` (选中的服务器列表)。

`onProgress` : 回调函数 (`server, isSuccess, message`)，用于实时刷新 UI 日志。

`token` : 取消令牌。

### 6.1 批量注册 (BatchRegister)

调用示例:

```

var cts = new CancellationTokenSource();
var selectedServers = GetSelectedServers(); // 获取 UI 选中的行

Task.Run(() => {
    // 执行批量注册
    var summary = _sdk.BatchRegister(
        selectedServers,
        "adminPwd", "newUser", "newPwd",
        (server, success, msg) =>
        {
            // [回调] 实时更新 UI 日志
            this.Invoke(new Action(() => {
                string status = success ? "成功" : "失败";
                txtLog.AppendText($"{server.IP} {status}: {msg}\r\n");
            }));
        },
        cts.Token
    );

    // [结束] 显示汇总
    this.Invoke(new Action(() => {
        MessageBox.Show($"成功: {summary.SuccessCount}, 失败: {summary.FailCount}");
    }));
});

```

## 6.2 批量切换 (BatchSwitch)

调用示例:

```

// 逻辑同上，核心调用：
_sdk.BatchSwitch(selectedServers, "admin", "user1", "pwd", (s, ok, msg) => {
    this.Invoke(new Action(() => {
        // 如果成功，建议刷新表格里的 "CurrentUser" 列
        if (ok) {
            s.CurrentUser = "user1";
            dgvServers.Refresh();

            // 建议在此处保存凭据到 Config
            _sdk.Config.AddOrUpdateCredential(s.IP, s.Port, s.VpsName, "user1", "pwd");
        }
    }));
}, cts.Token);

```

## 6.3 批量注销 (BatchLogout)

调用示例:

```
_sdk.BatchLogout(selectedServers, "admin", (s, ok, msg) => {  
    // 日志回调...  
}, cts.Token);
```

# 7. 注意事项与最佳实践

## 7.1 线程安全 (Thread Safety)

严禁在 UI 主线程（如 `Button_Click`）直接调用 `Discovery` 或 `User...` 方法，这会导致界面假死。

请务必使用 `Task.Run(() => { ... })` 包裹 SDK 调用。

在回调函数或 `Task` 结束后更新界面控件（如 `TextBox`, `DataGrid`），必须使用 `this.Invoke (WinForms)` 或 `Dispatcher.Invoke (WPF)`。

## 7.2 资源释放

SDK 的网络连接是短连接模式。每次调用方法时建立连接，获取结果后立即断开（`Dispose`）。

不需要手动编写 `using` 语句来管理连接，SDK 内部已处理。

## 7.3 数据持久化安全

目前的 `vp_config.xml` 使用 `Base64` 编码存储密码。这只能防止明文窥探，不能防止恶意破解。

如果应用于高敏感环境，建议修改 `VPConfigManager` 中的 `AddOrUpdateCredential` 方法，引入 `AES` 或 `Windows DPAPI (ProtectedData)` 加密。

## 7.4 异常处理

SDK 的 `Discovery` 和 `Batch` 方法内部会捕获网络异常（如连接超时），并通过回调返回失败信息，不会导致程序崩溃。

单机操作方法（如 `UserRegister`）如果遇到网络不通，可能会抛出异常或返回空字符串，建议在外层包裹 `try-catch`。



## 8. 高级主题 (Advanced)

本章节面向有特殊需求的高级开发者，介绍了如何自定义底层通信层，以及脱离 SDK 进行原生开发的协议细节。

### 8.1 自定义 IVPTransport (替换底层通信)

默认情况下，SDK 使用基于 `System.Net.Sockets.TcpClient` 的 `DefaultTcpTransport`。如果您需要支持 **Socks5 代理**、**流量监控**、或者集成到 **Netty / SuperSocket** 等框架中，可以通过实现 `IVPTransport` 接口来替换底层逻辑。

#### 步骤 1: 实现接口

```
using VP.SDK;

public class MyCustomTransport : IVPTransport
{
    /// <summary>
    /// 实现发送并接收的逻辑
    /// </summary>
    /// <param name="ip">目标 IP</param>
    /// <param name="port">目标端口</param>
    /// <param name="data">指令字符串 (如 "VP_CONNECT")</param>
    /// <param name="timeoutMs">超时时间</param>
    /// <returns>服务器响应的字符串</returns>
    public string SendAndReceive(string ip, int port, string data, int timeoutMs)
    {
        // 示例：这里可以写你自己的 Socket 逻辑，或者调用第三方库
        Console.WriteLine($"[MyTransport] 正在连接 {ip}:{port}...");

        // 模拟发送和接收
        // var response = MySocketLib.Send(ip, port, data);

        // 返回结果
        return "VP_OK";
    }
}
```

#### 步骤 2: 注入 SDK

在实例化 SDK 时，将自定义的 `Transport` 传入构造函数。

```
// 使用自定义通信层的
SDK var transport = new MyCustomTransport(); var sdk = new VPCClientSdk(transport, timeoutMs: 5000);

// 之后的所有调用 (Discovery, UserSwitch 等) 都会走 MyCustomTransport
var result = sdk.UserSwitch(...);
```

## 8.2 纯 TCP 通信协议 (脱离 SDK 开发)

如果您无法使用 .NET 环境（例如使用 Python, C++, Java 开发客户端），或者不想引入 SDK，可以参考以下协议规范直接与 VP 服务器通信。但是您可能自行实现加密方法 `VP.RemoteUACSDK.CryptoHelper`

### 1. 通信流程 VP 服务器采用 短连接 (Short-Lived Connection) 模式：

建立 TCP 连接。

发送 UTF-8 编码的指令字符串。

接收 UTF-8 编码的响应字符串。

断开连接。

### 2. 加密算法规范 服务器对敏感数据（如密码、用户名）使用 AES-256-CBC 加密。

Key (密钥): 对原始密码字符串进行 SHA256 哈希，生成的 32 字节数组即为 Key。

IV (向量): 每次加密需生成随机的 16 字节 IV。

Mode: CBC

Padding: PKCS7

最终数据格式: Base64( IV(16字节) + CipherText )

伪代码示例 (Python):

```
import hashlib
import base64 from Crypto.Cipher
import AES from Crypto.Util.Padding
import pad
import os

def vp_encrypt(plain_text, password):
    # 1. 生成 Key (SHA256)
    key = 'VP_REMOTE_UAC_PRIVATE_KEY'

    # 2. 生成随机 IV
    iv = os.urandom(16)

    # 3. AES 加密
    cipher = AES.new(key, AES.MODE_CBC, iv)
    encrypted_bytes = cipher.encrypt(pad(plain_text.encode('utf-8'), AES.block_size))

    # 4. 拼接 IV + 密文, 并转为 Base64
    return base64.b64encode(iv + encrypted_bytes).decode('utf-8')
```