HAN

2681037482@qq.com

摘要

基于DATATOOL需求的软件架构设计

DataTool

软件架构设计

|  |  |  |  |
| --- | --- | --- | --- |
| **修订记录** | | | |
| 编号 | 日期 | 内容 | 编辑人 |
|  | 2024/5/1 | 初版 | HAN |
|  |  |  |  |

目录

[1 文档 1](#_Toc167919475)

[**1.1** **概述** 1](#_Toc167919476)

[**1.2** **引用文件** 1](#_Toc167919477)

[2 架构设计 1](#_Toc167919478)

[**2.1** **编码规范** 1](#_Toc167919479)

[**2.2** **软件架构** 2](#_Toc167919480)

[**2.2.1** **cmd命令行模块架构** 4](#_Toc167919481)

[**2.2.2** **图形界面模块架构** 4](#_Toc167919482)

[**2.2.2.1** **主窗口架构** 4](#_Toc167919483)

[**2.2.2.2** **工具层架构** 4](#_Toc167919484)

[**2.3** **主程序详细设计** 5](#_Toc167919485)

[**2.3.1** **主程序** 5](#_Toc167919486)

[**2.3.2** **工具表** 6](#_Toc167919487)

[**2.3.2.1** **ini配置数据** 8](#_Toc167919488)

[**2.3.2.1.1** **主程序配置数据** 8](#_Toc167919489)

[**2.3.2.1.2** **工具窗口配置数据** 8](#_Toc167919490)

[**2.3.2.2** **读取ini配置文件** 10](#_Toc167919491)

[**2.3.2.2.1** **读取主程序ini配置** 10](#_Toc167919492)

[**2.3.2.2.2** **读取工具窗口ini配置** 10](#_Toc167919493)

[**2.3.2.3** **保存ini配置文件** 11](#_Toc167919494)

[**2.3.2.3.1** **保存主程序ini配置** 11](#_Toc167919495)

[**2.3.2.3.2** **保存工具窗口ini配置** 11](#_Toc167919496)

[**2.3.3** **图形界面模式** 11](#_Toc167919497)

[**2.3.3.1** **主窗口** 11](#_Toc167919498)

[**2.3.3.2** **翻页功能（选项卡控件）** 12](#_Toc167919499)

[**2.3.3.3** **工具窗口** 12](#_Toc167919500)

[**2.3.3.4** **伪代码说明** 13](#_Toc167919501)

[**2.3.3.4.1** **定义工具表 & 读取ini配置** 13](#_Toc167919502)

[**2.3.3.4.2** **创建工具窗口** 14](#_Toc167919503)

[**2.3.4** **cmd命令行模式** 15](#_Toc167919504)

[**2.3.4.1** **指令表** 16](#_Toc167919505)

[**2.3.4.2** **识别指令** 18](#_Toc167919506)

[**2.3.4.3** **获取参数** 18](#_Toc167919507)

[**2.3.4.4** **打印指令说明** 18](#_Toc167919508)

[**2.3.4.5** **伪代码说明** 18](#_Toc167919509)

[**2.4** **工具窗口详细设计** 20](#_Toc167919510)

[**2.4.1** **文件转换** 20](#_Toc167919511)

[**2.4.1.1** **注册窗口类的接口** 20](#_Toc167919512)

[**2.4.1.2** **窗口过程回调函数架构** 21](#_Toc167919513)

[**2.4.1.3** **创建窗口消息（WM\_CREATE）** 21](#_Toc167919514)

[**2.4.1.4** **子窗口消息（WM\_COMMAND）** 21](#_Toc167919515)

[**2.4.1.5** **导入DLL文件** 22](#_Toc167919516)

[**2.4.1.6** **解析脚本** 22](#_Toc167919517)

1. **文档**
   1. **概述**

本文档旨在根据DataTool的需求进行架构设计，实现所有需求。

本文档对下一阶段的设计文件起指导性作用。单元设计文件应基于本文档展开设计，实现本文档的全部模块。

* 1. **引用文件**
* DataTool需求

1. **架构设计**
   1. **编码规范**

必须项：

* 编译器警告等级开至最高，且无报警；
* gcc编译器必须使用“-Wall”和“-Wextra”选项；
* 编写代码过程中须注意对ANSI和UNICODE编码的兼容性：
  + 定义字符型变量或数组时，须使用TCHAR类型；
* 涉及字符串相关的处理时须使用如“\_tprintf”之类的兼容两种编码的函数；
  + 使用字符常量或字符串常量时，须使用TEXT()宏修饰；
* 在用到只接收UNICODE字符串或只接收ANSI字符串的函数时，必须注意先使用MultiByteToWideChar函数将ANSI编码字符串转换成UNICODE编码字符串，或使用WideCharToMultiByte将UNICODE编码转换成ANSI编码；
* 读写文件及窗口打印文本时须注意\r\n和\n；

推荐项：

* 建议使用MISRA C 2012规则；
  1. **软件架构**

软件基于Windows API开发。由两大模块构成：cmd命令行模块（紫色部分）和图形界面模块（橙色和蓝色部分）。

图形界面的架构基于WinMain函数及消息循环机制展开设计。

cmd界面直接使用Windows API的启动参数和各工具提供的接口。

软件架构见下方：

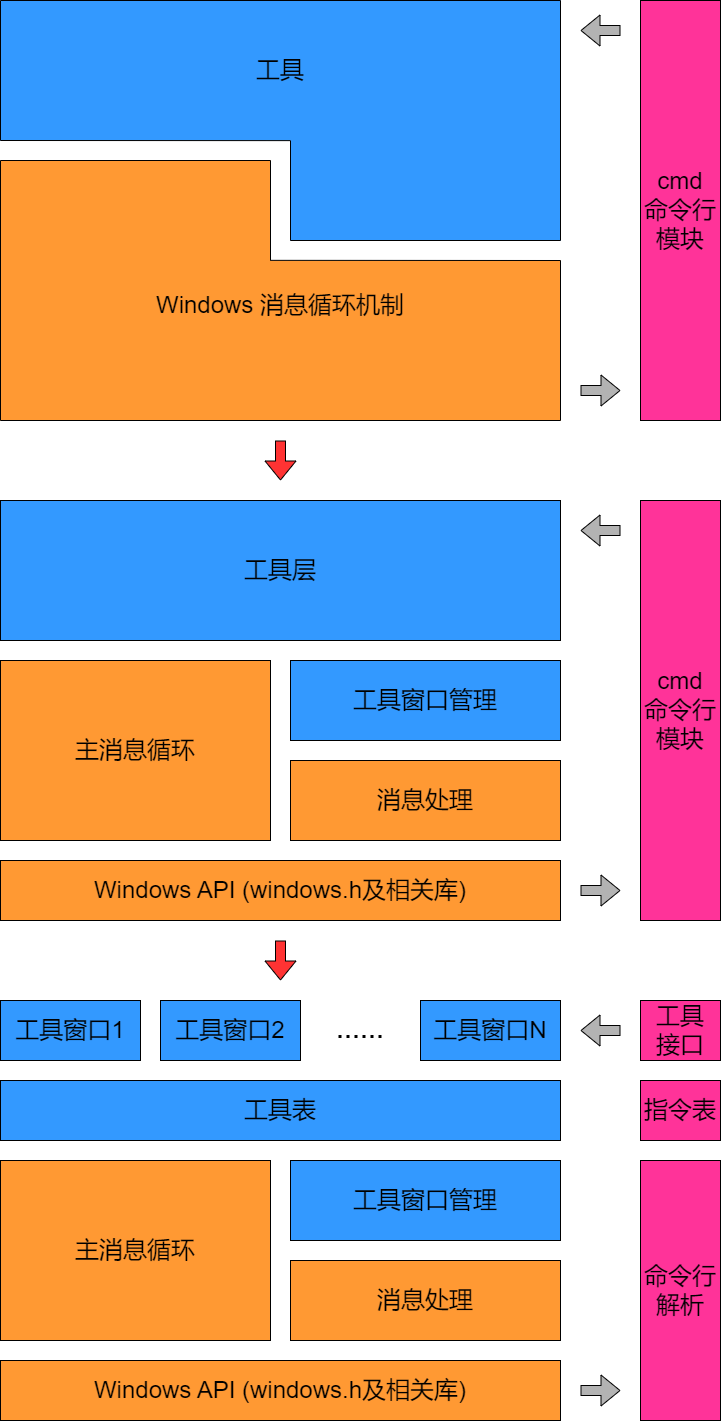


图 1 软件架构

* + 1. **cmd命令行模块架构**

cmd命令行模块由Windows API向其提供启动指令，模块在解析到有效指令后使用工具层提供的接口，执行操作。该模块跳过中间的消息循环机制。

* + 1. **图形界面模块架构**

图形界面模块通过Windows API的消息机制实现窗口。窗口分为主窗口和多个工具窗口，图 1 软件架构的主消息循环、消息处理、工具窗口的管理实现主窗口（即主函数进入消息循环前创建的窗口）。工具层实现工具窗口（工具窗口1、工具窗口2……工具窗口N）。

所有工具窗口都是一个独立可注册和创建的窗口类，并且拥有自己的窗口过程回调函数，并提供cmd命令行模块需要用到的接口。

* + - 1. **主窗口架构**

主窗口包含一个选项卡控件窗口，每一个选项对应一个工具窗口。

同时主窗口还需实现工具窗口的管理，在用户点击选项卡的选项后，隐藏当前活动的工具窗口，显示目标工具窗口。

选项的切换是一个简单的隐藏/显示窗口的过程，这意味着从工具窗口1切换至别的工具窗口再切换回工具窗口1时，该窗口的内容应当保持不变。

* + - 1. **工具层架构**

工具层由多个工具窗口构成。任何工具窗口类的实例都应有一个指针拓展（即窗口类的cbWndExtra成员值为sizeof(void\*)，32位程序此值为4，64位程序此值为8）用来存放指针。

窗口过程回调函数须在WM\_CREATE消息中执行以下步骤：

1. 申请足够的空间供窗口实例使用，申请的空间须是动态内存，不随回调函数的返回而释放。
2. 使用SetWindowLongPtr将所申请的内存的首地址存放进窗口实例的拓展内存中。后续任何需要处理的消息都首先通过GetWindowLongPtr读取该指针。
3. 初始化工具窗口，包括创建子窗口、初始化申请的动态内存。

其它消息（如WM\_COMMAND和WM\_NOTIFY）实现具体的功能。

* 1. **主程序详细设计**
     1. **主程序**

主程序采用WinMain函数的架构，总体流程见下方：

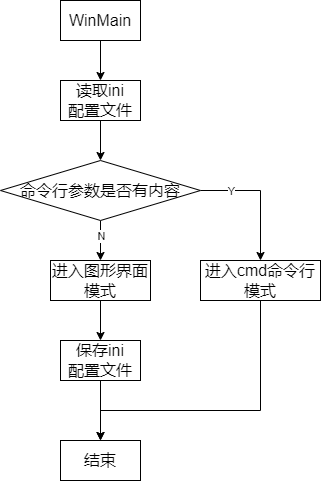


图 2 主程序流程图

主程序通过查表法来实现该流程。为了实现工具层的可拓展性，应用程序首先定义一个工具表用来存储所有工具的信息，包括：ini配置文件的读写、工具的名称、工具的窗口类等。程序在执行到“读取ini配置文件”、“进入图形界面模式”、“保存ini配置文件”时，通过遍历工具表并调用对应接口，来实现的工具的管理。当需要修改或添加工具窗口时，只对工具表中的元素作修改或添加，而不必修改主窗口逻辑。

应用程序首先遍历查询工具表，执行所有工具的“读取ini配置文件”操作（如果该工具有配置相关接口）。然后根据启动指令判断进入图形界面模式还是cmd命令行模式。如果进入图形界面模式，遍历注册所有工具的窗口类并创建窗口，在结束时遍历执行所有工具的“保存ini配置文件”操作（如果该工具有配置相关接口）。

* + 1. **工具表**

如上一章节所述，工具表是主程序使用及管理任何工具的唯一入口。应用程序通过遍历该表来获取工具的接口并执行。

工具表通过结构体数组实现，结构体定义如下：

* 窗口标题（常量）；
* 窗口类名（常量）；
* 窗口句柄（程序运行中赋值）；
* 注册窗口类（常量）；
* 保存ini配置的接口（常量）；
* 读取ini配置的接口（常量）；
* ini配置数据（程序运行中赋值）；

使用该结构体定义一个全局数组，每个元素存放一个工具的信息，该数组即为工具表，工具表的用法见下方：

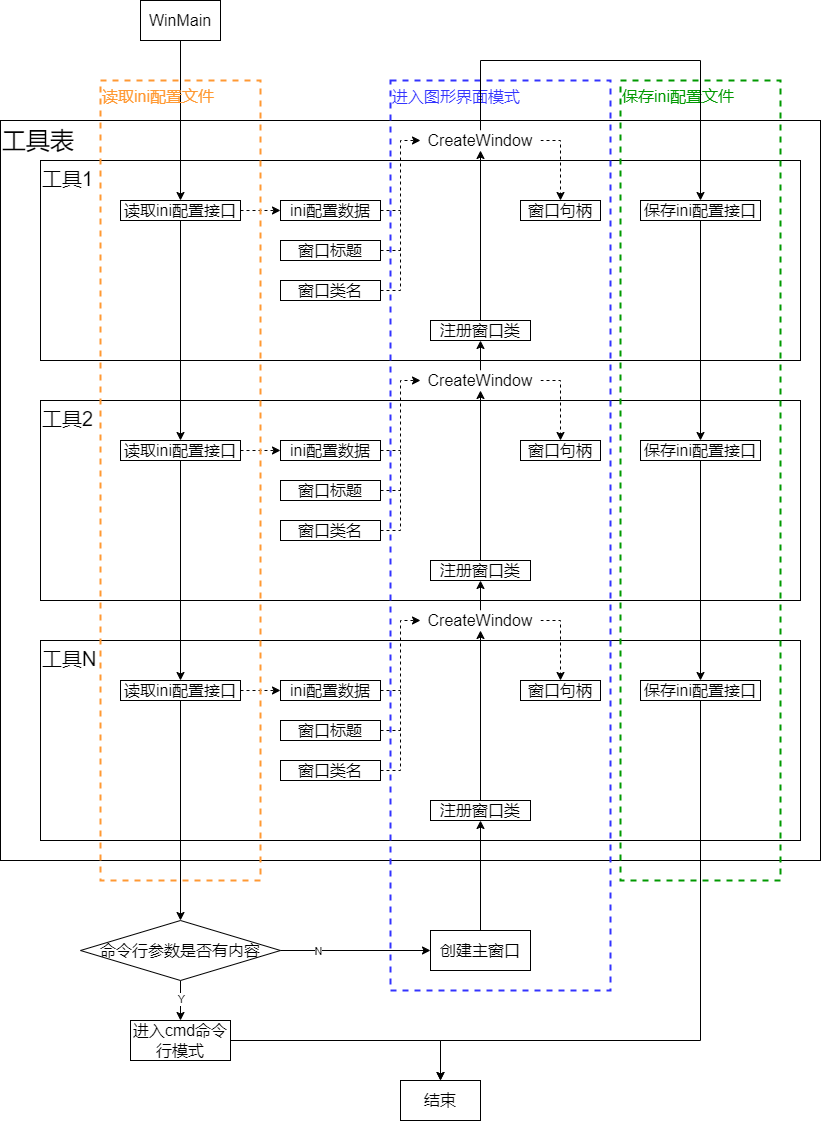


图 3 工具表用法

上图是对工具表的使用说明，也是“图 2 主程序流程图”的详细展开。

橙色虚线框是“读取ini配置文件”的详细步骤；

蓝色虚线框是“进入图形界面模式”的详细步骤；

绿色虚线框是“保存ini配置文件”的详细步骤；

虚线箭头是数据的传递。

工具表结构体中“窗口标题”、“窗口类名”、“注册窗口类”、“保存ini配置的接口”、“读取ini配置的接口”这些成员是在程序编码阶段预先定义好的内容，不可在运行中被程序修改。必须在定义处赋予初始值。建议使用const修饰符声明为常量。其中“保存ini配置的接口”和“读取ini配置的接口”允许为NULL，如果这些成员为NULL，表示工具没有需要存储或读取的配置信息，应用程序必须对这些成员做NULL值判断并跳过，以防程序崩溃。

“窗口句柄”、“ini配置数据”是程序运行阶段产生的数据，由系统分配或环境变化等因素生成，不在定义处赋初始值。

* + - 1. **ini配置数据**

ini配置数据是为了实现记录用户操作的功能。在用户打开程序时，程序需要恢复至上次关闭时的状态。

* + - * 1. **主程序配置数据**

主程序在启动时首先读取自己所需的ini配置数据，在图形界面的关闭消息中保存对应的数据。需要处理的ini配置数据如下：

* 程序关闭时所选中的工具窗口；
  + - * 1. **工具窗口配置数据**

在开发工具窗口时，需要先定义一个可以被主窗口访问到的用来存储配置数据的结构体及对应变量，并将该变量的地址存放到工具表，应用程序在执行“读取ini配置的接口”阶段时，将该地址通过参数的方式传递给工具窗口。

结构体的定义在工具的头文件中实现，变量在主文件中定义。

主程序对该地址的操作仅限于传参，不对地址的有效性作判断。ini配置数据的读写逻辑见下方：

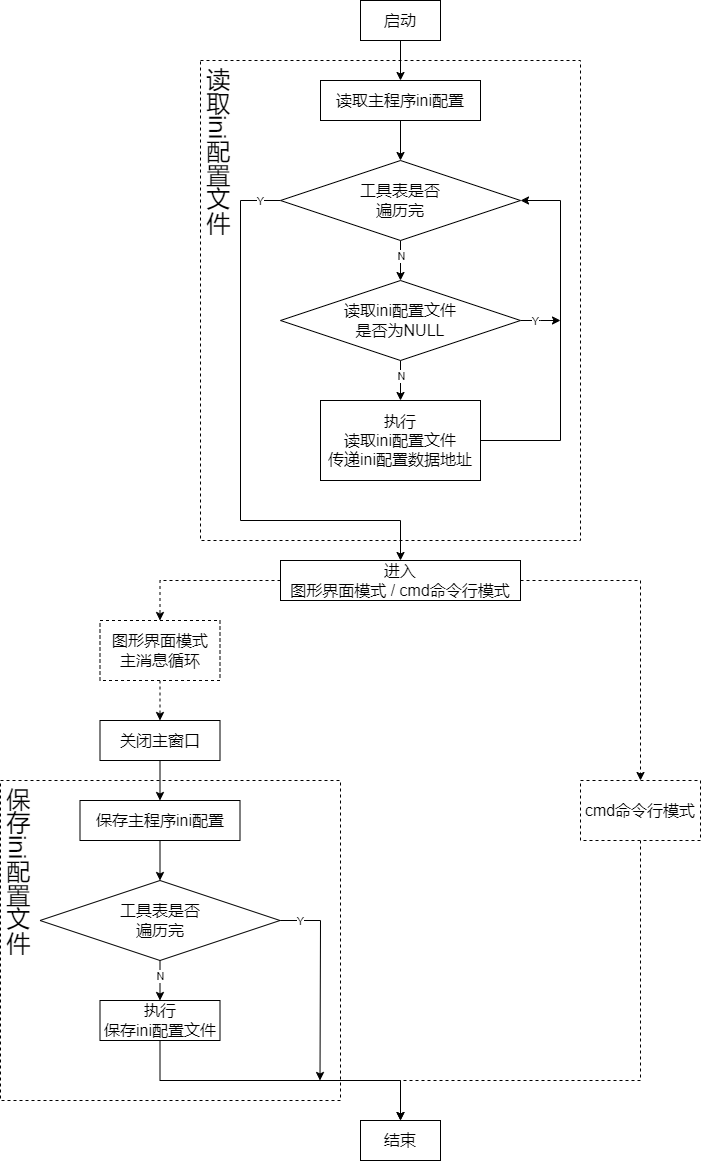


图 4 ini配置数据读写流程

* + - 1. **读取ini配置文件**
         1. **读取主程序ini配置**

主程序的ini配置数据在启动时直接读取，该操作须位于判断进入图形界面模式还是cmd命令行模式之前。

* + - * 1. **读取工具窗口ini配置**

在读取完主程序的ini配置后，通过查询工具表调用接口的方式读取工具窗口的ini配置。

接口的参数为ini文件的路径名称和ini配置数据的地址，无返回值，接口从传入的文件内读取到自身需要的配置项，存入ini配置数据。应用程序在创建工具时将配置数据通过传参的方式提供给工具的窗口过程回调函数。

* + - 1. **保存ini配置文件**

保存ini文件通过查表的方式来实现，并且仅在图形界面模式中有效。并且在关闭主窗口的消息（WM\_DESTROY）中执行。

* + - * 1. **保存主程序ini配置**

在主程序的窗口过程回调函数的WM\_DESTROY消息中首先保存主程序的ini配置。

* + - * 1. **保存工具窗口ini配置**

在保存完主程序的ini配置后，通过查询工具表调用接口的方式保存工具窗口的ini配置。

接口的参数为ini文件的路径名称和子窗口句柄，无返回值。接口根据句柄获取窗口内各模块的状态，存放到目标路径。

* + 1. **图形界面模式**

图形界面包含三个阶段：

1. 创建主窗口。
2. 创建选项卡控件窗口。
3. 遍历工具表创建标签页和工具窗口。
4. 处理选项卡控件窗口。
   * + 1. **主窗口**

主程序在进入图形界面模式后使用CreateWindow创建一个具有WS\_OVERLAPPEDWINDOW和WS\_SYSMENU风格的主窗口，在更新窗口后进入消息死循环。

WS\_OVERLAPPEDWINDOW表示该窗口具有标题、最小化按钮、最大化/还原按钮、关闭按钮。

WS\_SYSMENU用来预留制作菜单功能。

程序在创建主窗口后，开始在主窗口的窗口过程回调函数的WM\_CREATE消息中使用查表法遍历工具表制作翻页功能和工具窗口。

* + - 1. **翻页功能（选项卡控件）**

翻页功能应当与工具表是紧密相关的。

程序首先创建一个选项卡控件窗口，然后开始遍历工具表，在每个循环中先添加一个选项，然后创建工具窗口，创建的工具窗口句柄保存到工具表对应的“窗口句柄”成员中，以此实现工具窗口和选项的绑定。选项的文本为工具表结构体的窗口标题成员。

遍历完工具表后，根据ini文件中读取到的配置设置标签窗口的选中项。

当用户点击某个选项时，程序隐藏当前显示的工具窗口，显示目标工具窗口。

* + - 1. **工具窗口**

如上一章节所述，工具窗口使用遍历工具表来实现。

在遍历工具表的每个循环中添加完选项后，先注册当前循环对应的窗口类，然后使用CreateWindow创建该类的子窗口（窗口风格使用WS\_CHILD，父窗口使用主窗口句柄）。

工具窗口的类名、ini配置数据都使用结构体成员中的对应内容。窗口句柄成员用来保存CreateWindow返回的句柄。

工具窗口默认隐藏（窗口风格不要使用WS\_VISIBLE），在遍历完工具表后，根据ini文件中读取到的配置显示选中的窗口。

本架构采用Windows API的消息循环机制，所有工具窗口的架构设计都从窗口过程回调函数展开设计。

* + - 1. **伪代码说明**
         1. **定义工具表 & 读取ini配置**

以下是一段定义工具表并读取ini配置数据的伪代码，供单元设计参考，分为主程序文件和工具窗口文件：

|  |  |
| --- | --- |
| 定义工具表并读取ini配置数据伪代码（主程序文件.c） | |
| 1. struct 工具表[N] = { 2. [0] = { 3. .窗口标题 = TEXT("工具1标题"), 4. .窗口类名 = 工具1类名, 5. .注册窗口类 = 工具1注册窗口类, 6. .读取ini配置的接口 = 工具1读取ini配置, 7. .保存ini配置的接口 = 工具1保存ini配置, 8. .ini配置数据 = 工具1配置数据, 9. }, 10. [1] = { 11. .窗口标题 = TEXT("工具2标题"), 12. .窗口类名 = 工具2类名, 13. .注册窗口类 = 工具2注册窗口类, 14. .读取ini配置的接口 = NULL, // 无相关配置，填NULL 15. .保存ini配置的接口 = NULL, // 无相关配置，填NULL 16. .ini配置数据 = NULL, // 无相关配置，填NULL 17. },   ****   1. [N - 1] = { 2. .窗口标题 = TEXT("工具N标题"), 3. .窗口类名 = 工具N类名, 4. .注册窗口类 = 工具N注册窗口类, 5. .读取ini配置的接口 = NULL, 6. .保存ini配置的接口 = NULL, 7. .ini配置数据 = NULL, 8. }, 9. }; 10. int WINAPI \_tWinMain(…) 11. { 12. size\_t iLoop 13. for (iLoop = 0; iLoop < N; iLoop++) 14. { 15. if (NULL != 工具表[iLoop]. 读取ini配置的接口) 16. { 17. 工具表[iLoop].读取ini配置的接口( 18. 工具表[iLoop]. ini配置数据, 19. TEXT("cfg.ini") 20. ); 21. } 22. } 23. } | |
| 工具1伪代码（工具1.h） |
| 1. #define 工具1类名 TEXT("工具1类名") 2. typedef struct 工具1配置数据结构体; 3. void 工具1注册窗口类(hInst); 4. void 工具1读取ini配置(struct 工具1配置数据结构体 \*, TCHAR\*); 5. void 工具1保存ini配置(struct 工具1配置数据结构体 \*, HWND); |
| 工具2伪代码（工具2.h） |
| 1. // 该工具没有设计配置相关接口，工具表中相关内容全填NULL 2. #define 工具2类名 TEXT("工具2类名") 3. void 工具2注册窗口类(HINSTANCE hInst); |
| 工具N的形式与工具1和工具2相同，不再重复 |

* + - * 1. **创建工具窗口**

以下是一段创建工具窗口的伪代码，供单元设计参考：

|  |
| --- |
| 创建工具窗口伪代码（主程序文件.c） |
| 1. struct 工具表[N]; // 上一段伪代码已定义了 2. size\_t iLoop; 3. 创建翻页窗口(); 4. for (iLoop = 0; iLoop < N; iLoop++) 5. { 6. 添加标签(标签句柄, 工具表[iLoop].窗口标题); 7. 工具表[iLoop].注册窗口类(); 8. 工具表[iLoop].窗口句柄 = CreateWindow( 9. 工具表[iLoop].窗口类名, 10. NULL, 11. WD\_CHILD, 12. x, 13. y, 14. w, 15. h, 16. 主窗口句柄, 17. iLoop, 18. 主窗口实例句柄, 19. 工具表[iLoop].ini配置数据 20. ); 21. } 22. 设置翻页窗口的选中项(); |

* + 1. **cmd命令行模式**

程序在执行完所有工具的“读取ini配置文件”操作后，需要对主函数的命令行参数进行判断，当检测到存在指令后，程序进入cmd命令行模式。

这意味着程序支持.bat脚本之类的文件。

和主程序的工具表架构类似，cmd命令行模式也是通过查表的方式实现。

根据WinMain函数的结构，命令行参数会将用户输入的所有指令集中在一个字符串中。即参数lpCmdLine = "cmd1 cmd2 cmd3"，

在进入命令行模式后，程序首先使用CommandLineToArgvW（注意该函数只有宽字符版本）将主程序的命令行转换成多个单条指令构成的字符串数组。即参数TCHAR型的lpCmdLine转换成WCHAR\* pCmd[] = { L"cmd1", L"cmd2", L"cmd3", }形式的字符串数组。该字符串数组下文称为“命令组”。转换方式见下方：

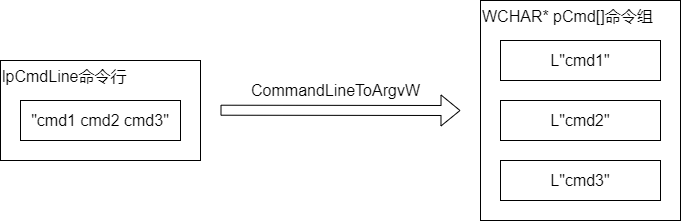


图 5 命令行转换为命令组

随后遍历命令组逐个获取每条命令，每获取一条命令查询一次指令表。如果找到有效指令，执行指令；如果指令表中查询不到有效指令，终止程序。

* + - 1. **指令表**

如上一章节所述，指令表是命令行模式调用工具接口的唯一入口，应用程序通过遍历该表来识别指令及获取工具的接口并执行。

工具表通过结构体数组实现，结构体定义如下：

* 指令字符串；
* 参数个数；
* 工具接口；
* 指令说明打印接口；

使用该结构体定义一个全局数组，每个元素存放一个指令的信息，该数组即为指令表，指令表的用法见下方：

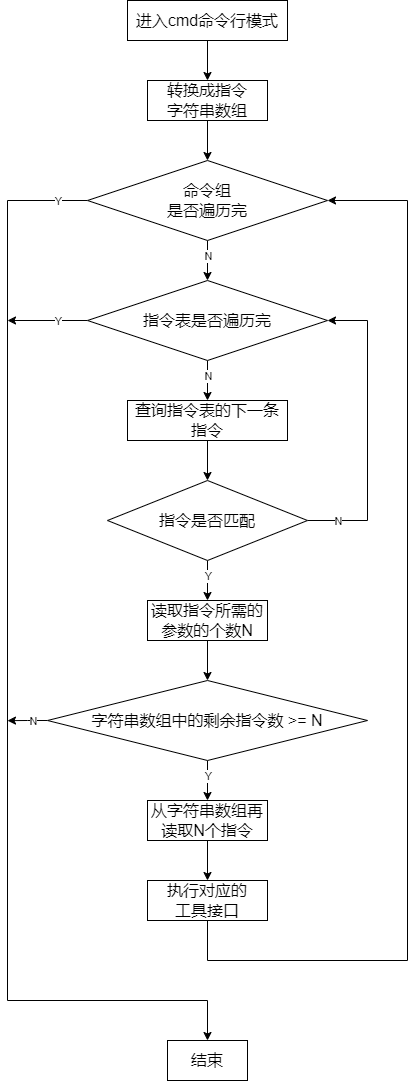


图 6 指令表用法

上图是对指令表的使用说明，也可以视作是对“图 4 ini配置数据读写流程”中的“cmd命令行模式”的详细展开。

处理命令组大致上分为两个步骤：识别指令和获取参数。处理过程就是在这两个步骤中来回循环，直到命令组中所有命令都做了处理。

* + - 1. **识别指令**

原则上刚进入cmd命令行模式时首先尝试识别指令，识别的方式为将当前命令和指令表中的“指令字符串”成员进行字符串比较（比较的时候不对大小写作严格要求，代码实现时可以不区分大小写，也可以区分大小写）。如果一致，开始获取参数。

* + - 1. **获取参数**

当程序识别出指令后，开始获取参数。先读取该指令需要的参数个数，然后判断输入指令中的剩余命令数是否大于等于需要的参数个数，如果满足，则取若干个命令用作参数。否则终止程序。

当获取了足够的参数后，应用程序调用指令表中的“工具接口”成员，并传入参数。对参数的处理需要工具接口自己实现，主程序只负责将命令字符串原封不动地作为参数传递给接口。

* + - 1. **打印指令说明**

cmd命令行模式需要支持一个类似“help”指令的特殊的功能。

用户在输入该指令后，遍历执行指令表的所有“指令说明打印接口”成员，打印每条指令的说明。

* + - 1. **伪代码说明**

以下是一段定义指令表及处理命令行的伪代码，供单元设计参考：

|  |
| --- |
| 定义工具表并读取ini配置数据伪代码（主程序文件.c） |
| 1. struct 指令表[N] = { 2. [0] = { 3. .指令字符串 = L"cmd1", 4. .参数个数 = 0, 5. .工具接口 = 指令1接口, 6. .指令说明打印接口 = 指令1说明打印接口, 7. }, 8. [1] = { 9. .指令字符串 = L"cmd2", 10. .参数个数 = 1, 11. .工具接口 = 指令2接口, 12. .指令说明打印接口 = 指令2说明打印接口, 13. },   ****   1. [N - 1] = { 2. .指令字符串 = L"cmdN", 3. .参数个数 = 0, 4. .工具接口 = 指令N接口, 5. .指令说明打印接口 = 指令N说明打印接口, 6. }, 7. }; 8. size\_t iLoop; 9. size\_t jLoop; 10. WCHAR\*\* pCmd; 11. int nCmdCnt; 12. size\_t nParamCnt; 13. pCmd = CommandLineToArgvW(lpCmdLine, &nCmdCnt); 14. for (iLoop = 0; iLoop < nCmdCnt; iLoop++) 15. { 16. for (jLoop = 0; jLoop < N; jLoop++) 17. { 18. if (0 == wcscmp(pCmd[iLoop], 指令表[jLoop].指令字符串)) 19. { 20. nParamCnt =指令表[jLoop].参数个数; 21. if ((iLoop + nParamCnt) < (nCmdCnt – 1)) 22. { 23. 指令表[jLoop].工具接口(&pCmd[iLoop + 1]); 24. iLoop += nParamCnt; 25. } 26. else 27. { 28. break; 29. } 30. } 31. } 32. } |

* 1. **工具窗口详细设计**

本章节基于上文，对各个工具窗口进行详细设计。包括但不限于工具的介绍、工具表结构体的各个成员的实现、cmd模式的接口设计、工具内部的逻辑设计。

* + 1. **文件转换**

文件转换是一个独立的工具。该工具作为工具表中的一个元素而存在，需要遵循工具表的架构开发，并为工具结构体的所有成员分配具体的对象。

主程序在注册并创建该窗口类后，所有文件转换相关功能都在该类的窗口过程回调函数中完成。

头文件需要对外声明至少以下内容：

* 注册窗口类的接口（供图形界面模式使用）；
* 保存ini配置的接口（供图形界面模式使用）；
* 读取ini配置的接口（供所有模式使用）；
* 转换文件的接口（供所有模式使用）；
* 转换信息结构体（供转换文件的接口传参用）；
  + - 1. **注册窗口类的接口**

该接口使用RegisterClass或RegisterClassEx注册预先定义好的窗口类名和窗口过程回调函数，同时需要为该类的实例申请额外的空间存放一个指针。该指针指向的结构体是窗口实例存放数据的唯一途径，需要在创建实例时由程序分配好内存并指明。

文件转换窗口结构体至少需要包含以下成员：

* 所有子窗口句柄；
* 文件转换配置；
* 从DLL中读到的文件转换信息；
  + - 1. **窗口过程回调函数架构**

文件转换窗口过程回调函数流程见下方：

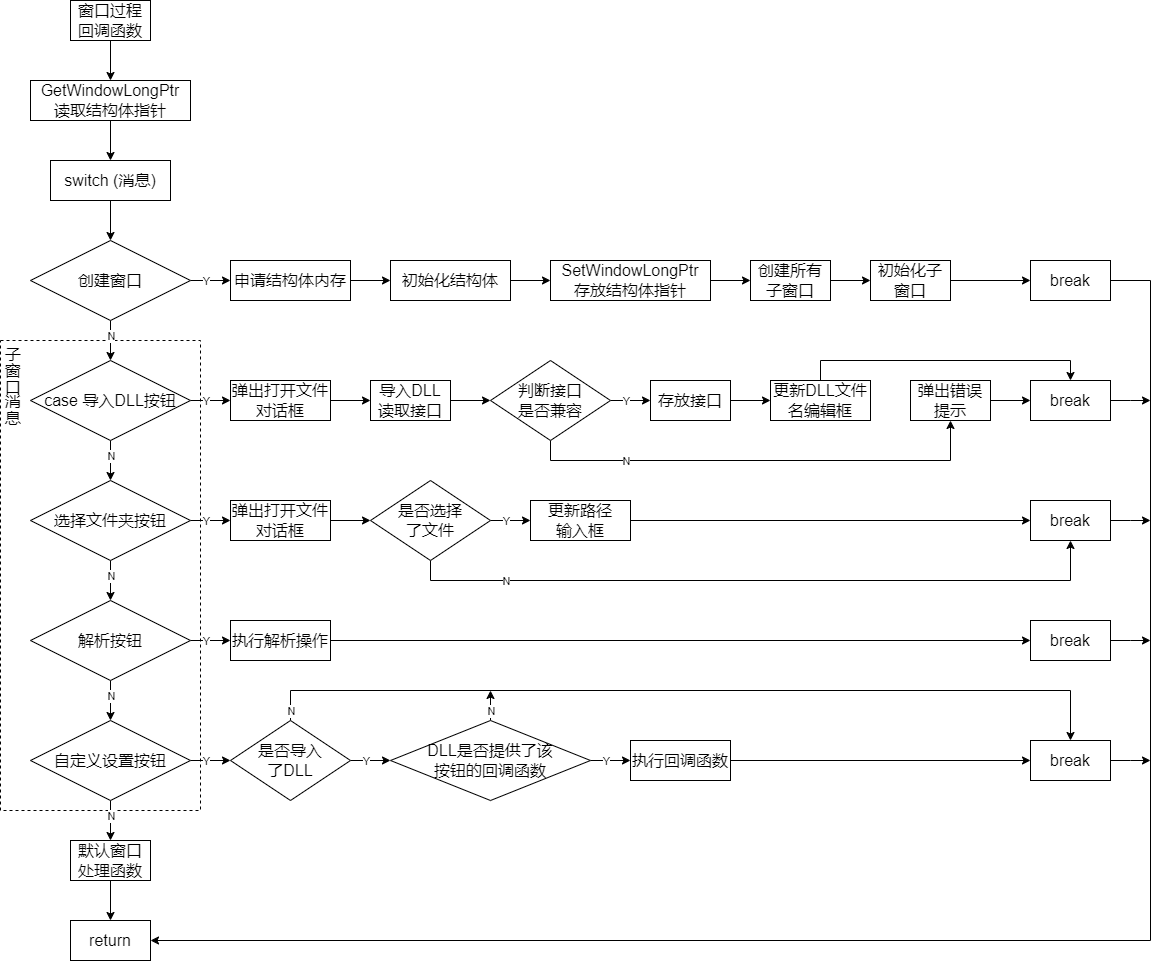


图 7 文件转换消息过程回调函数架构图

无论收到任何信息，首先使用GetWindowLongPtr获取指针。然后根据消息执行对应代码。

* + - 1. **创建窗口消息（WM\_CREATE）**

在创建窗口消息中，先对窗口指针进行初始化再保存。其它消息则直接使用该指针。处理完指针后，开始创建并初始化子窗口。

* + - 1. **子窗口消息（WM\_COMMAND）**

需要处理消息的子窗口包含以下部分：

1. 导入DLL按钮。在此窗口的消息中，程序执行导入DLL文件操作，相关详细设计参考章节“2.4.1.5导入DLL文件”。
2. 选择文件夹按钮。
3. 解析按钮。
4. 自定义设置按钮。

除以上窗口外，其它窗口都使用windows默认的窗口过程处理函数,无需特殊处理。

* + - 1. **导入DLL文件**
      2. **解析脚本**

解析脚本是需要由用户自己制作的DLL文件。该文件需要定义一个可以被DataTool获取到的全局变量。该变量的结构需要具有以下成员：

* 打开源文件时的回调函数；
* 协议信息，该成员是一个数组，每个元素对应一个协议，该成员需要具有以下子成员：
  + 协议名称（字符串）；
  + 帧头数据；
  + 帧头长度；
  + 新文件的文件头；
  + 文件头长度；
  + 读取到帧头的回调函数；
* 协议个数；
* 自定义配置：该成员需要具有以下子成员：
  + 初始化配置回调函数（该函数在DataTool启动时调用）；
  + 点击自定义设置按钮的回调函数；
  + 保存配置的回调函数（该函数在DataTool关闭时调用）；