1.实验一

1.1思路

运用Windows系统提供的Advapi32.dll动态链接库，从外部引用一些对注册表进行操作的函数，具体引用见WindowsHomework2工程form1.cs的209-261行。

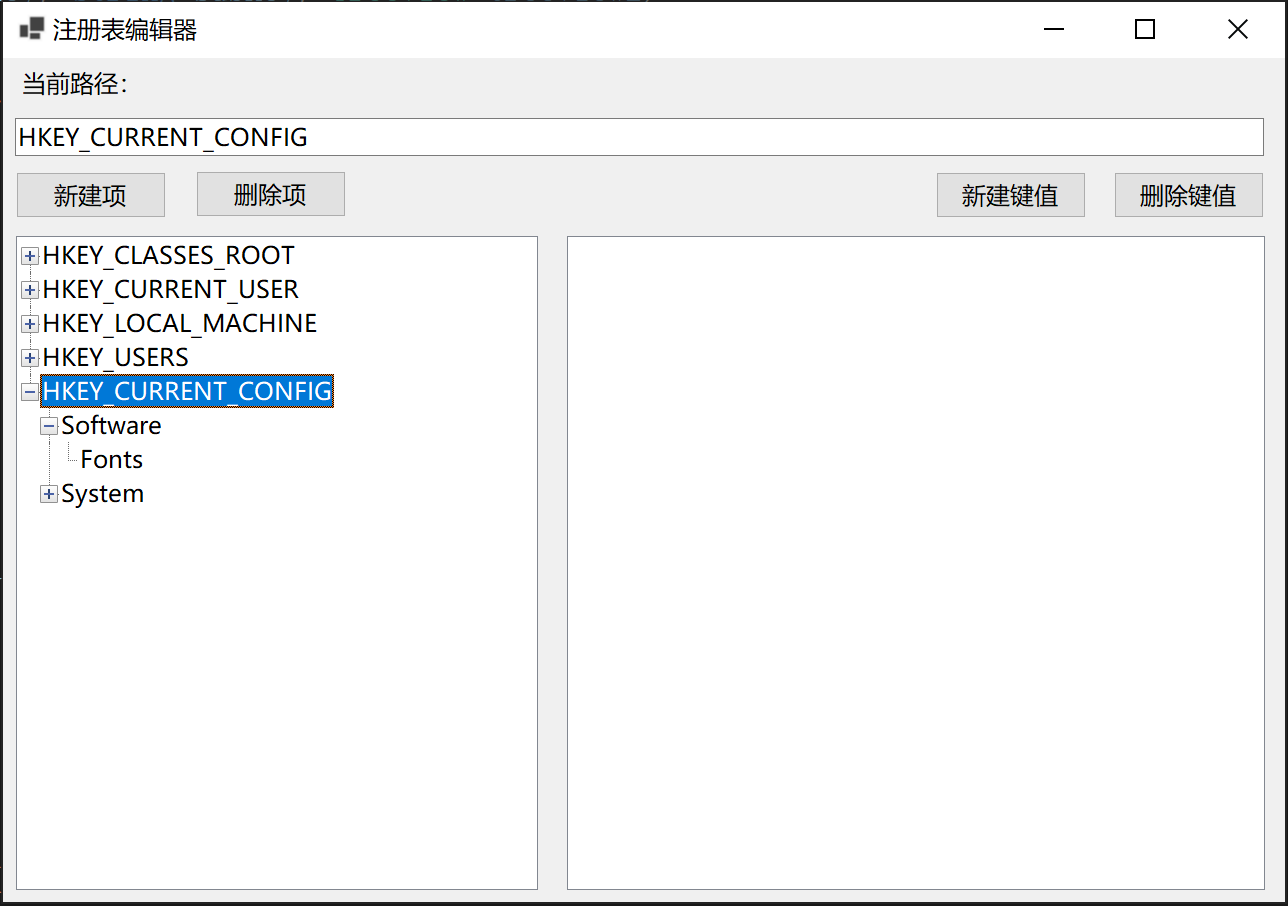
之后运用RegEnumKey()函数读取出整个注册表文件（由于整个注册表节点较多，代码中限制了每个根节点仅读取100个子节点作为演示，在第354行修改条件即可全部遍历），运用递归的思路，将每一个节点存放在treenode中，建立起一个注册表树，树的建立过程可见48-58行的writeData()函数和348-378行的getEnumKey()函数。

建立完成后，treeview1中将会显示出注册表树，当前路径会显示在textbox1中，双击项，右侧treeview2中会显示该项中的键值，分别运用AddKey(),AddKeyValue(),deleteKey(),

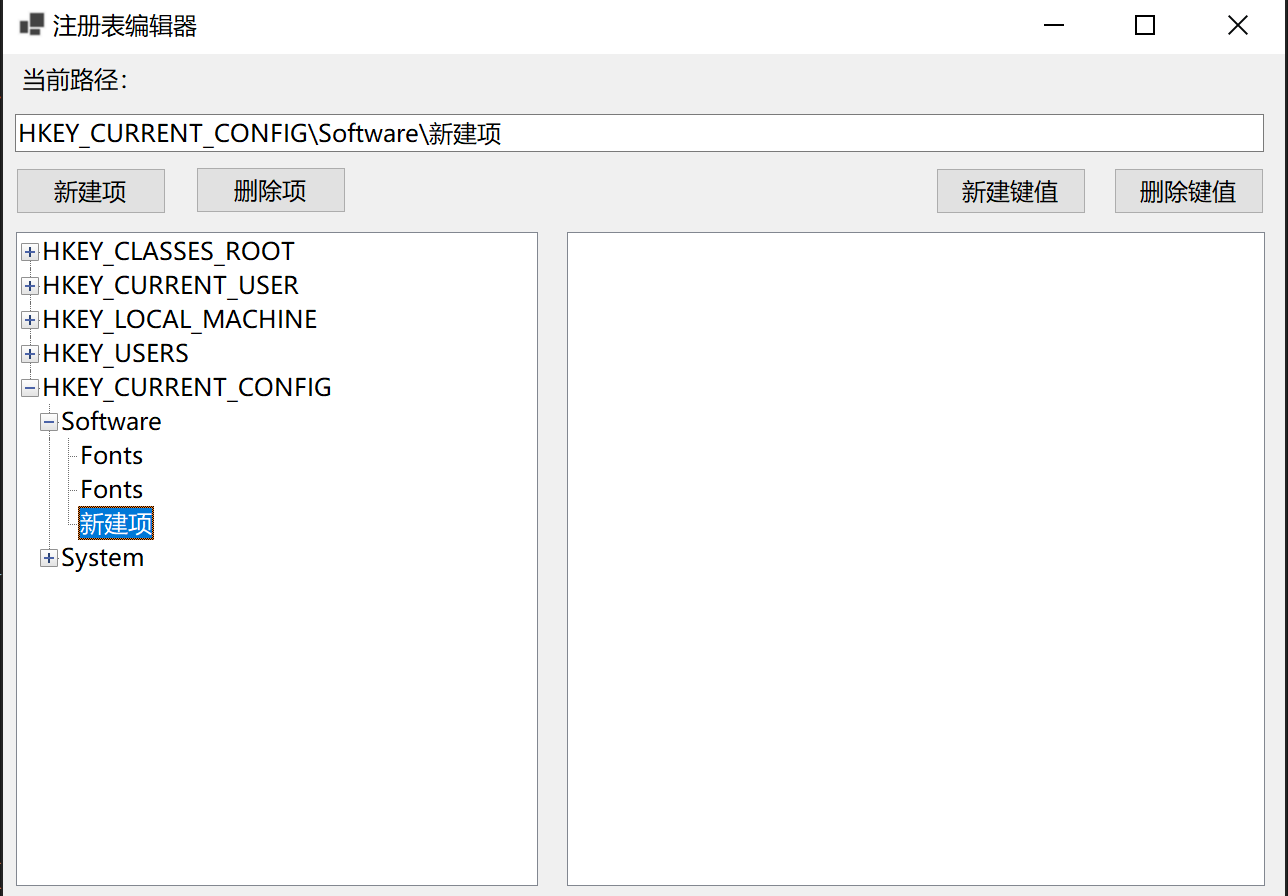
DeleteKeyValue()函数添加项、键值，删除项、键值，四个函数分别被绑定在相应的按钮上，每个函数中都运用了从Advapi32.dll引用的外部函数。双击键值可以修改键值数据。

1.2 使用

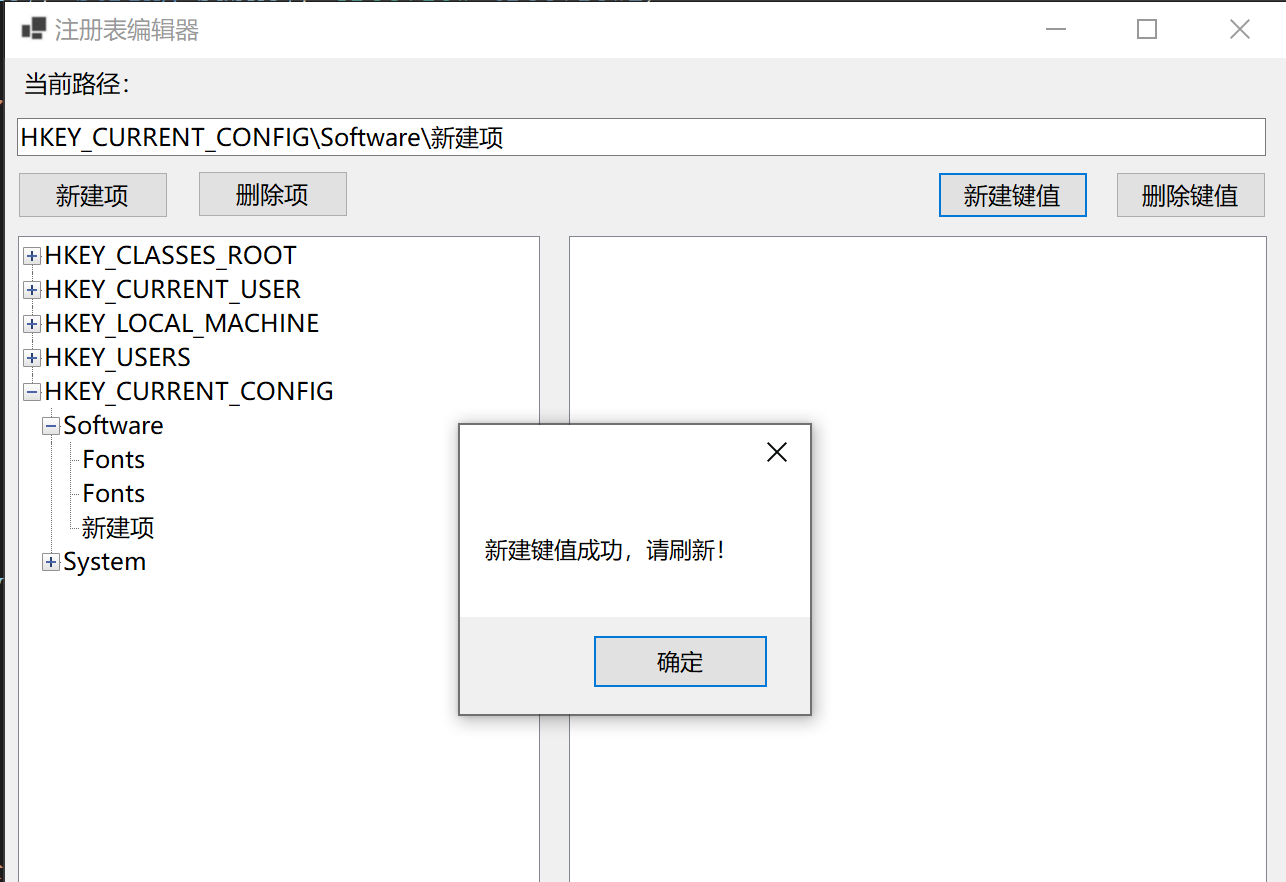
（1）初始化界面如图所示。

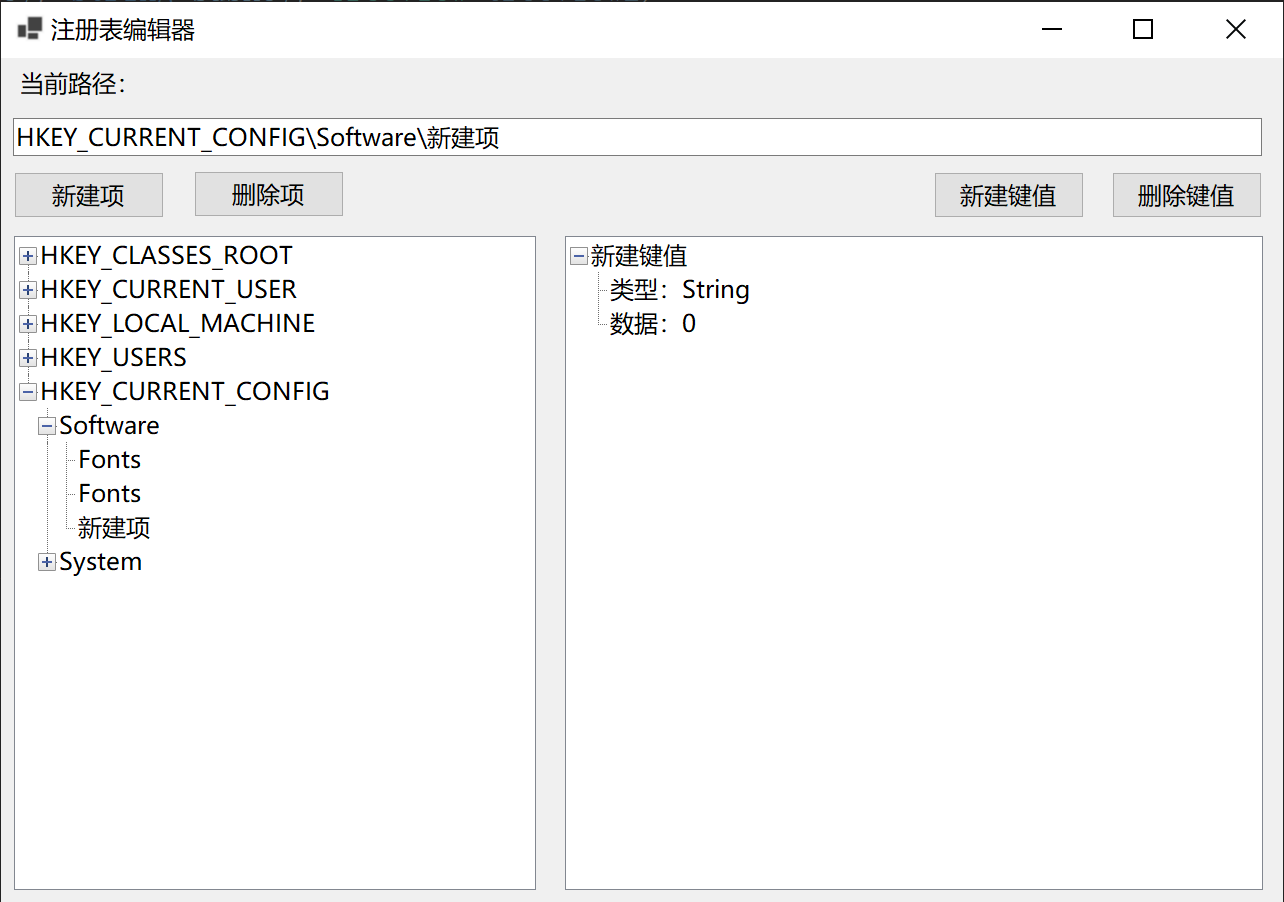


1. 在software目录下点击新建项，即可新建一个项，如图所示。

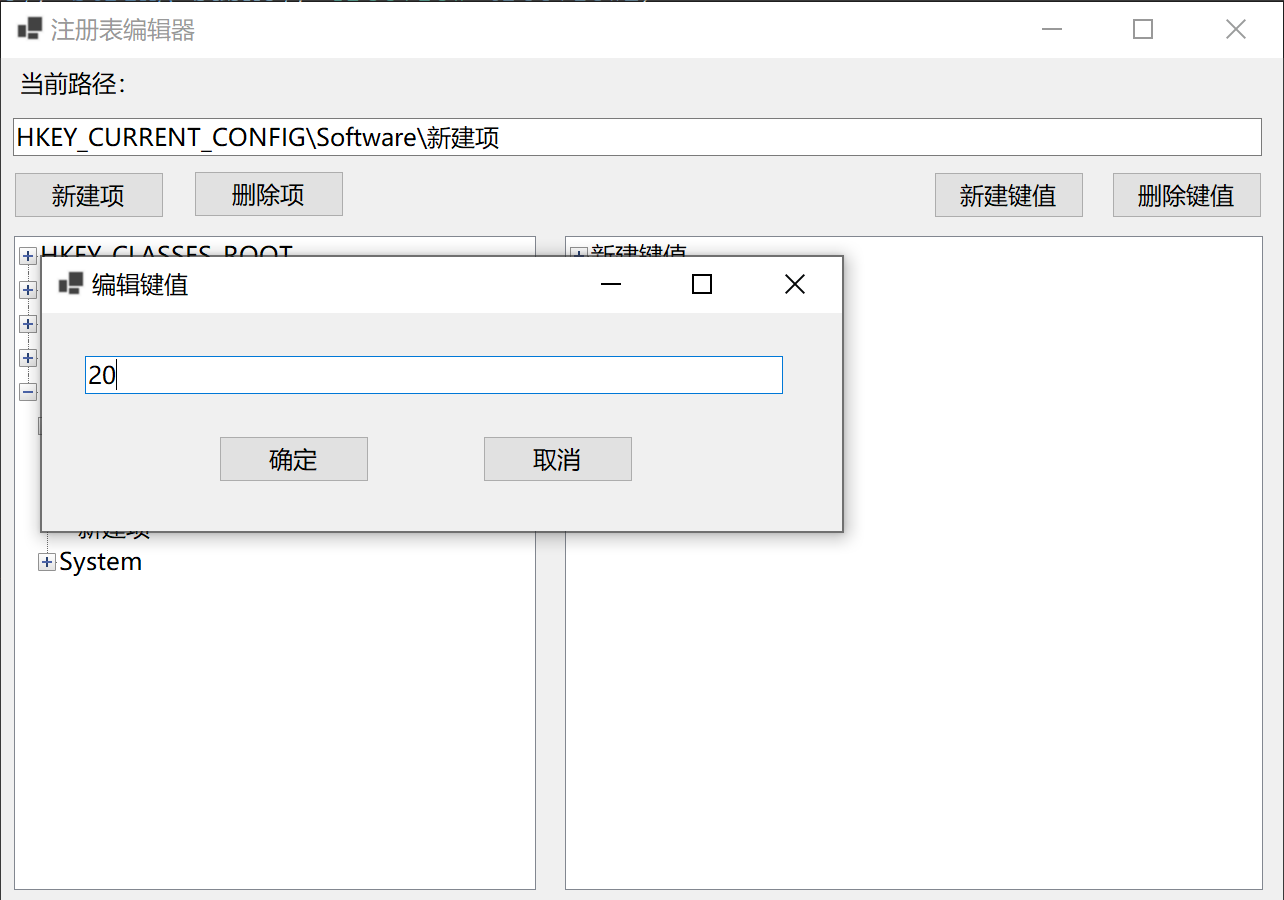


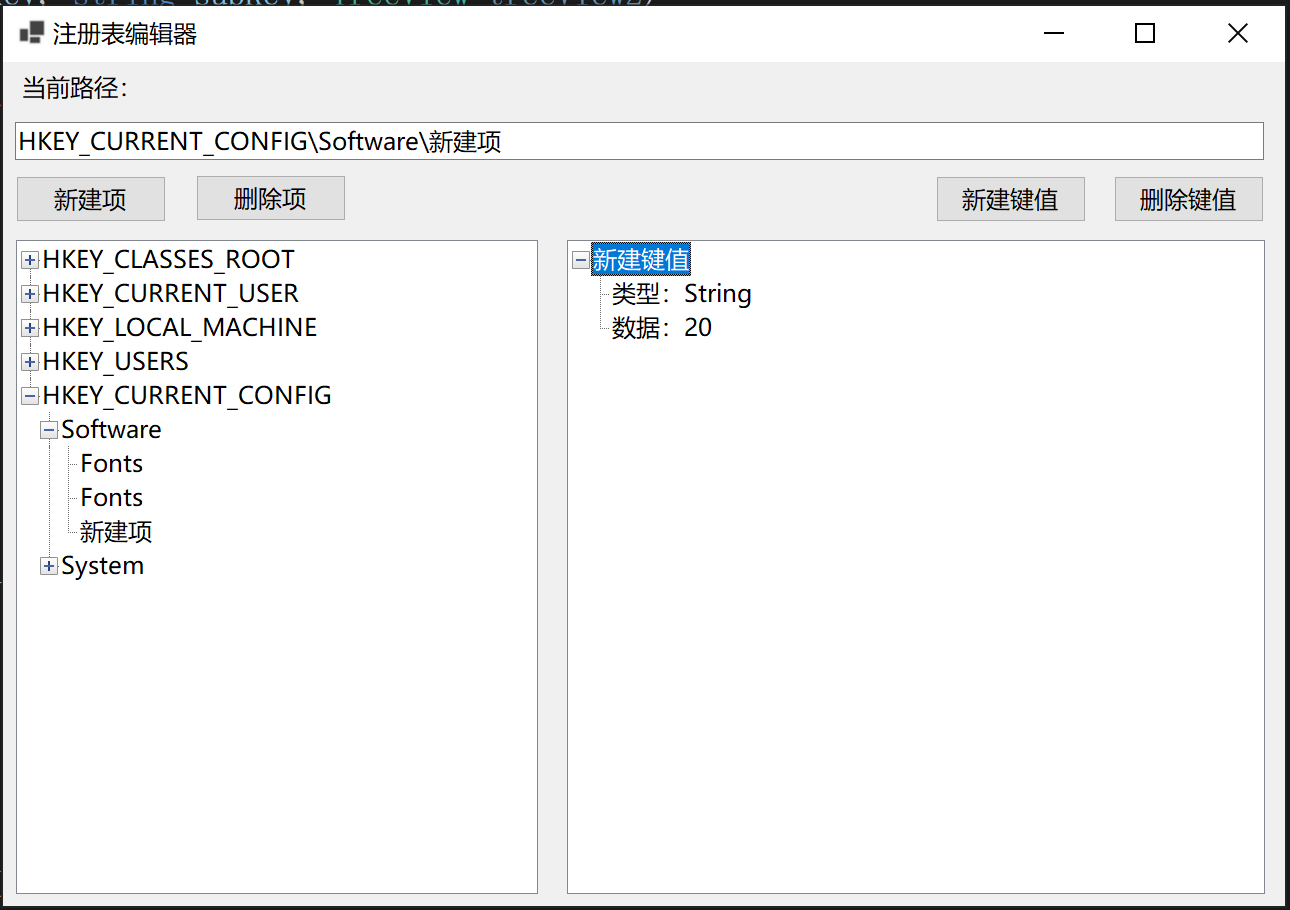
（3）选中该新建项后，点击新建键值，即可新建一个键值.



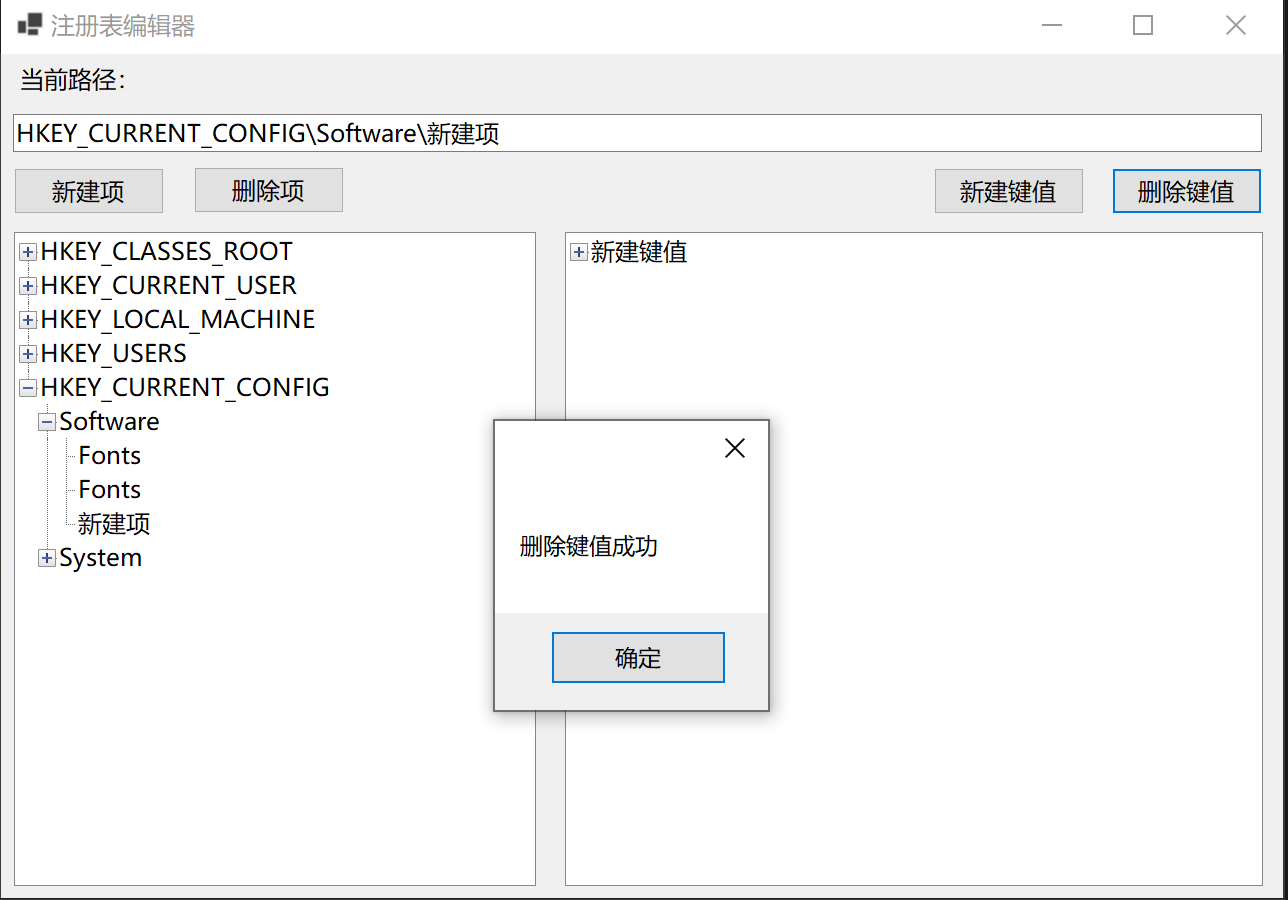


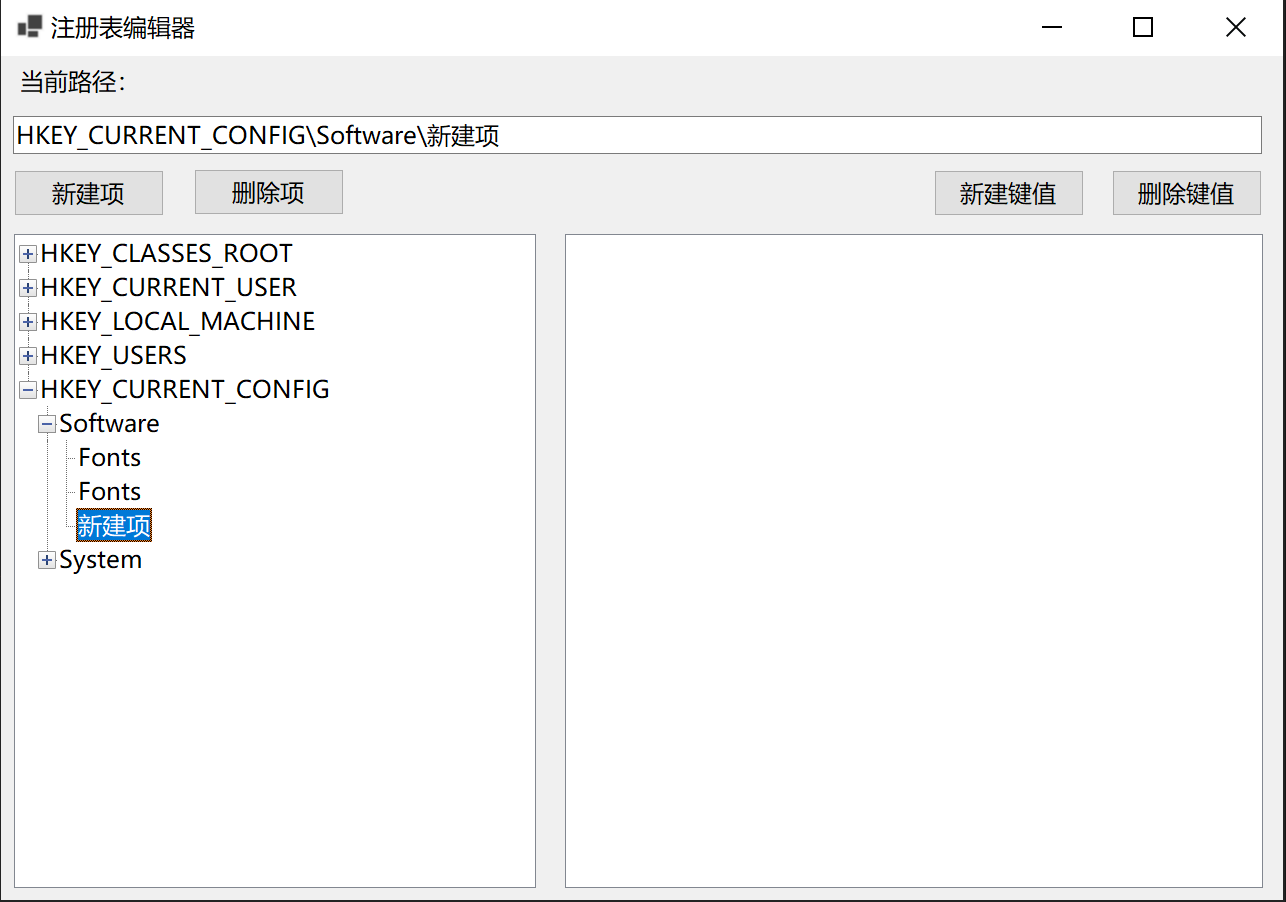
1. 双击新建的键值，即可以修改数据。



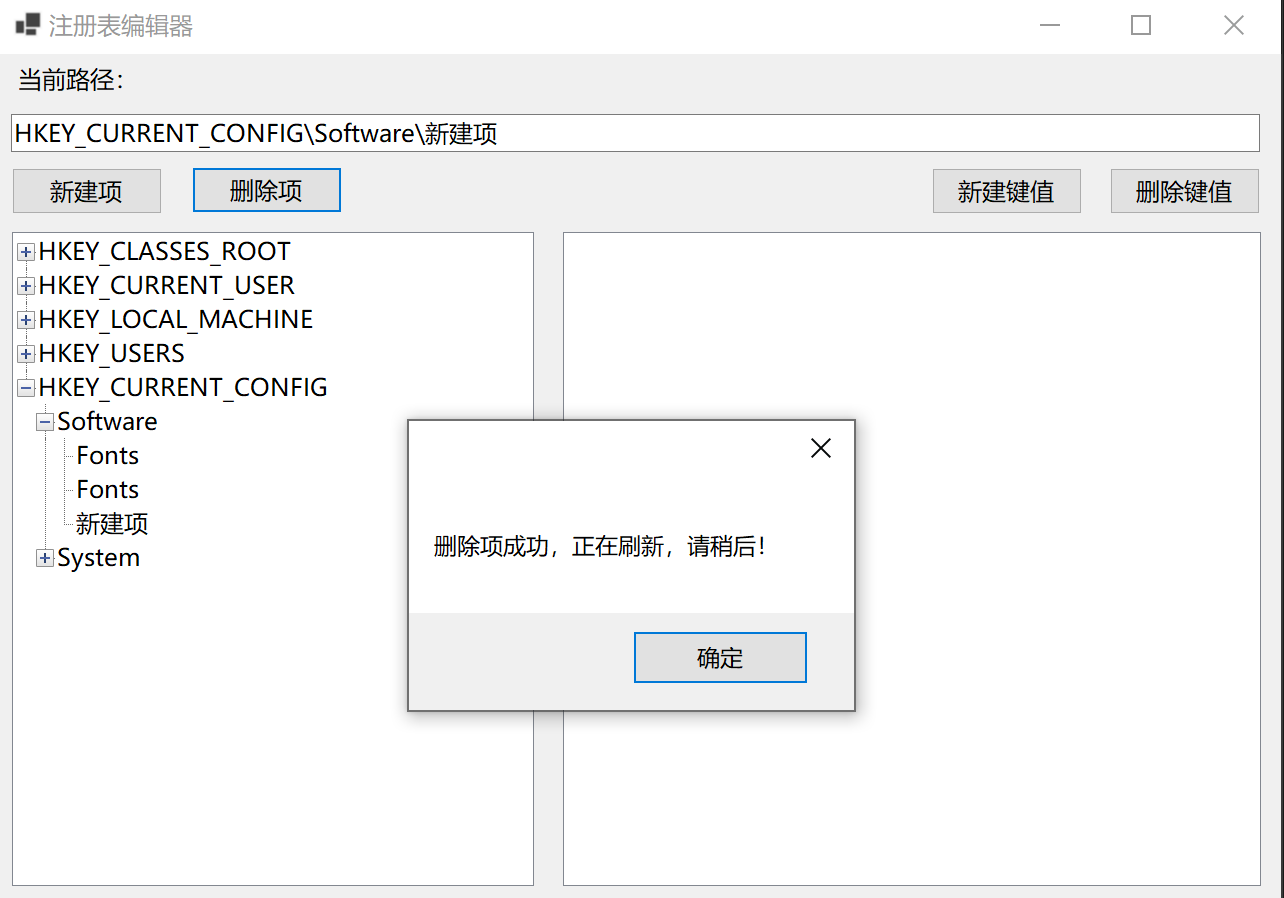


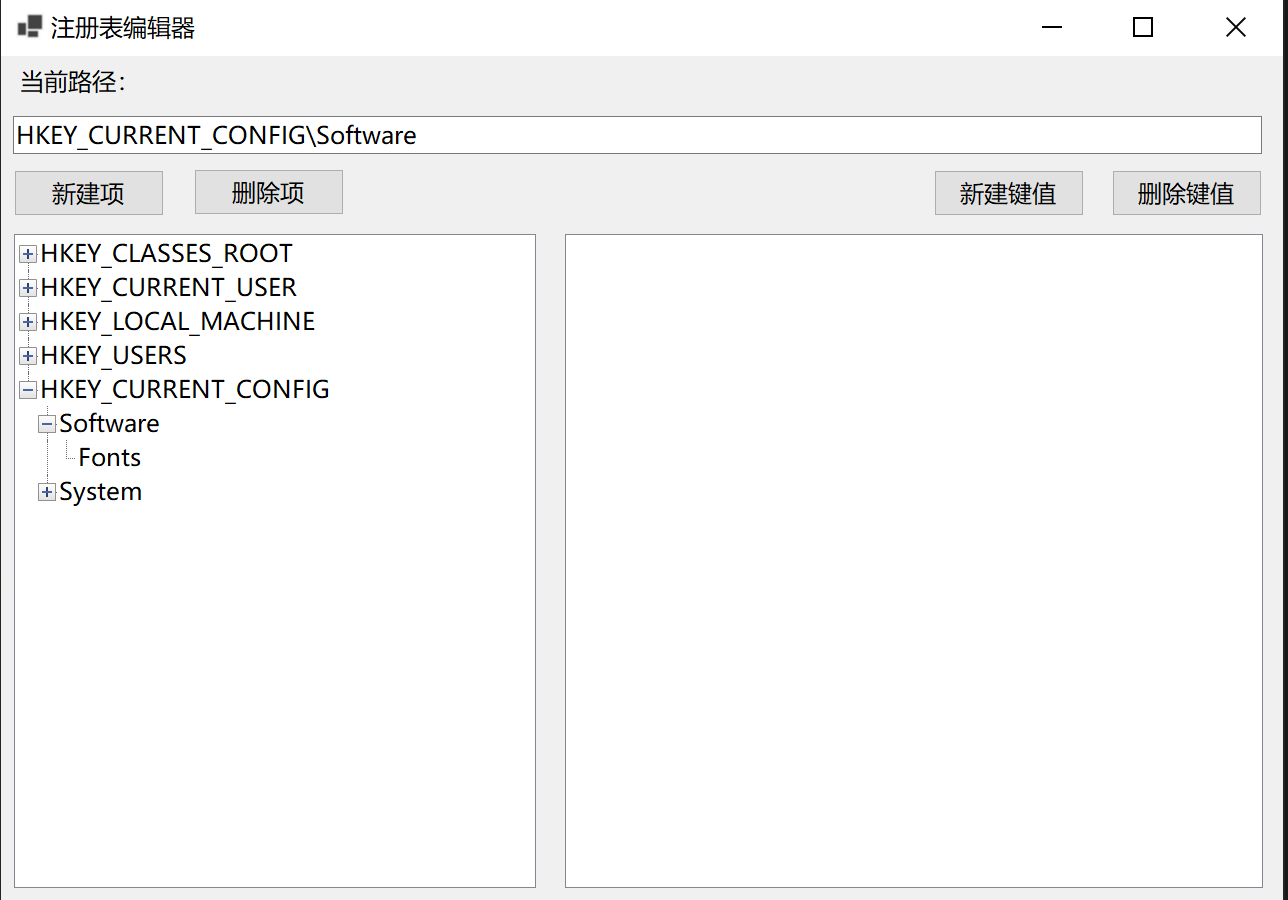
1. 选中新建键值后，点击删除键值，即可以删除新建的键值。





1. 选中新建项后，点击删除项，即可以删除选中项。





1.3 关键函数

1.3.1 初始化注册表树

本实验最大的难点在于合理的显示注册表的层次结构，而注册表本身为树形结构，故直接采用树形结构进行展示最为合适，所以选择了treeview控件来做显示。

对每个根节点都调用getEnumKey()函数，该函数初始化一个index值，该值是用于控制phkey遍历到第index个子节点，每遍历到一个子节点就创建一个treenode并添加到treview对应位置中，再对这个子节点进行递归遍历，直到遍历完整个树。

|  |
| --- |
| public static void getEnumKey(string key, string subKey,TreeNodeCollection currentNode)  {  int ret = 0;  int len = 2000;  StringBuilder b = new StringBuilder(2000);  int index = 0;  while (ret == 0 && index<100)  {  try  {  IntPtr hKey = TransferKeyName(key);  IntPtr pHKey = IntPtr.Zero;  ret = RegUtil.RegOpenKeyEx(hKey, subKey, 0, KEY\_ALL\_ACCESS, out pHKey);  if (ret == 0)  {  ret = RegUtil.RegEnumKey(pHKey, index, b, len);  }  if (ret == 0) {  TreeNode node1 = new TreeNode(b.ToString());  currentNode.Add(node1);  getEnumKey(key, subKey+b.ToString()+"\\", node1.Nodes);    Console.WriteLine(b);  index++;  }  }  catch  { }  }  } |

1.3.2 显示键值

根据当前选择的路径，对路径最后一个treenode节点，进行遍历，运用RegEnumValue（）函数即可读出键名，并在右侧treeview中添加treenode节点，再运用RegQueryValueEx（）函数读出键值和值的类型，最后将读出来的值建立treenode并作为键名节点的子节点。

全部完成后，index增加1，然后对下一个节点进行遍历。具体代码如下。

|  |
| --- |
| ret = RegUtil.RegEnumValue(pHKey, index, valueName, ref valueNameLen, IntPtr.Zero, ref valuetype, IntPtr.Zero, IntPtr.Zero);  if (ret != 0) return;  TreeNode node1;  string valuenm = valueName.ToString();  if (valuenm == "")  {  node1 = new TreeNode("(默认)");  }  else  {  node1 = new TreeNode(valueName.ToString());  }    treeView2.Nodes.Add(node1);    uint len = 1024;  int ret2 = RegUtil.RegQueryValueEx(pHKey, valueName.ToString(), 0, ref valuetype, value, ref len);  TreeNode node2 = new TreeNode("类型："+valuetype.ToString());  TreeNode node3 = new TreeNode("数据：" + value.ToString());  node1.Nodes.Add(node2);  node1.Nodes.Add(node3);  index++; |

1. 实验二-用c++创建dll

2.1 创建dll文件

首先用visual studio创建一个dll工程，然后在头文件pch.h中声明函数，在pch.cpp中对函数进行定义，具体代码如下。

|  |
| --- |
| extern "C" \_declspec(dllexport) int Add(int a, int b);  extern "C" \_declspec(dllexport) int Decline(int a, int b); |
| int Add(int a, int b)  {  return a + b;  }  int Decline(int a, int b)  {  return a - b;  } |

之后切换为release模式，生成工程，即可在release文件夹中生成dll文件。

2.2 测试dll文件

在winform工程中引入dll文件中的函数并测试，方法如下：

|  |
| --- |
| [DllImport(@"WindowsHomework2-CppDll.dll", EntryPoint = "Add", SetLastError = true, CharSet = CharSet.Ansi, ExactSpelling = false, CallingConvention = CallingConvention.StdCall)]  public extern static int Add(int a, int b);  [DllImport(@"WindowsHomework2-CppDll.dll", EntryPoint = "Decline", SetLastError = true, CharSet = CharSet.Ansi, ExactSpelling = false, CallingConvention = CallingConvention.StdCall)]  public extern static int Decline(int a, int b); |

需要注意的是用此方法生成的dll调用的是x86平台的接口，需要将winform工程的目标平台也改为x86才能正常运行。

1. 实验三-用c#创建dll文件

3.1 生成dll文件

用c#创建dll文件则更为简单，只需要创建一个c#类库工程，在类中定义出相应的函数，生成工程，即可生成dll文件。

调用函数时，首先将生成的dll文件添加到工程中，然后运用如下的方法引用函数。

|  |
| --- |
| [DllImport(@"WindowsHomework-csDll.dll")]  public extern static int Add(int a, int b);  [DllImport(@"WindowsHomework-csDll.dll")]  public extern static int Decline(int a, int b); |

在主类中运用如下的方法调用外部函数：

|  |
| --- |
| WindowsHomework\_csDll.Class1 callDll = new WindowsHomework\_csDll.Class1();  callDll.Add(a, b)； |