

接力成熟结构软件进行二次开发的经验与实践

摘要：主要介绍了接续结构分析软件 STAAD 进行二次开发的一些经验，比较了存储、查询数据的两种方案的效率。

Abstract: Some experience and advice have been shared when developing STAAD's post-progress program, and the efficiency of two plans to store and search the data have been compared in this article.

关键词：STAAD，二次开发，散列表

Key Words: STAAD, Post-processor, Hash Table, CMap

缘起

越来越多的设计单位开始接触国外工程，通常会采用某种国际知名的结构设计软件以便存档和校审，常见的有 STAAD/PRO, GTSTRUDL, SAP2000 等。在构件设计和优化方面，这几种软件都内置了多种国外规范（如美国规范、加拿大规范、欧洲规范等），但是在钢结构节点设计方面，这些软件目前还比较欠缺。通常国外的工程设计要求计算书透明、步步可查、易于校审和存档。我们需要根据 STAAD 计算出的各工况下的杆端力来进行节点设计，所以对 STAAD 进行了二次开发，把杆件的几何信息以及内力信息和相邻杆件的信息整合到一起，输出到 EXCEL 文件中，并在 EXCEL 文件中采用 VBA 编程来设计节点。

前提

接续某软件进行二次开发，很重要的一个前提就是对该软件的重要文件的格式要有清楚而全面的了解，二次开发流程无非读取输入文件和输出文件、合理高效存储模型的几何信息和输出文件中的内力或位移等信息、分析这些存储的信息、按照需要的格式和组织方式输出相关信息。

前面提到的软件有一个共同的优点：结构模型文件 and 内力结果文件的格式都是公开而且比较固定的，更为重要的是，这些软件的说明书中都有详细的文字来解释这些命令的含义，所有信息对用户而言都是透明的，非常有利于在此基础上进行二次开发，而国内开发的一些结构分析设计软件，在公开这些信息方面做的就很不尽人意：或频繁改动模型文件的结构，或把模型文件做成二进制格式，外人根本无法了解其结构，更无法知晓各个参数的含义，二次开发也无门可入，这种闭关自守的思想对扩大软件的影响力没有什么好处。

工作介绍

笔者已经完成了 GTSTRUDL 和 STAAD/PRO 这两个软件的一些二次开发工作，目前实现的功能主要有两个：1.读入模型文件,自动计算出柱子的计算长度系数；2.根据模型文件 and 内力分析文件整理数据输出到 EXCEL 软件中。现总结一下开发思路，希望能起到抛砖引玉的功效，对接续其他软件进行二次开发也能有一点借鉴意义。

GTSTRUDL 和 STAAD/PRO 这两个都提供了大量的命令来完成建模、分析、设计和优化（由于二者的大部分命令极其相似，使用者在精通了其中一个软件之后，很容易掌握另一种），此外，二者都提供了比较完备的图形界面，不熟悉或不习惯命令流方式的用户可以通过相对比较直观的图形平台进行建模、分析等一系列工作，程序会自动把用户在图形平台上进行的操作保存成相应的命令。命令流的优点是简捷、便于校核、也便于新手通过命令流来向有经验的前辈学习，所有信息都在命令流中表达清楚，打印在一两张纸上就足以把握整个结构。

本文主要介绍接续 STAAD 进行二次开发的一些经验。该思路同样适用于接力其他结构软件进行二次开发，只要其输入输出文件的格式是公开且比较固定的就可以。

接续 STAAD 进行二次开发

接续 STAAD 进行二次开发，至少有两种方式：1.采用 openSTAAD 提供的诸多函数。2.读取 STAAD 的相关文件，自行开发所需要的函数。第一种方法相对比较容易入门，但只适

合针对 STAAD 的二次开发。第二种方案难度相对大一些，但通用性好，可以囊括对其他成熟的结构软件的开发。本文采用第二种方案。

首先，我们需要了解其输入文件(后缀为 std)和输出文件(后缀为 anl)的格式。输入文件包含了节点信息、杆件关联号、杆件截面信息、杆件 BETA 角度、杆端约束信息、支座约束信息以及荷载信息等。输出文件的内容的多寡可以由用户通过 STAAD 的命令流来控制，一般对二次开发有用的包括各杆件在各种工况下的内力以及截面特性等。

以为节点设计准备数据为例，需要从输入文件中 (*.std) 读出几何信息（节点坐标、杆件关联号、杆件 BETA 角、约束信息、杆件截面名等），并由此分析出一个节点上连接了几根杆件、节点连接形式（刚接还是铰接、接在什么构件上）等等，还要从 *.anl 文件中读出各种工况下的杆端力、杆件截面尺寸等数值。

以下文本是一个典型的 STAAD 的输入文件，我们可以看到其中有很多英文单词，这些就是 STAAD 的命令。二次开发的主要工作就是通过读取输入文件，解析这些命令，把这些命令中包含的一些信息用适当的数据结构存储，以便于进行一些重新组合或分析后输出更多信息。

“ JOINT COORDINATES 1000 0 0 0; 222 6 0 0; 4 0 4.5 0; 5 6 4.5 0; ***** * STAAD.Pro Generated Comment * ***** MEMBER INCIDENCES 1 1000 4; 2 222 5; 6 4 5; 9 1 5; 10 2 4; MEMBER PROPERTY AUSTRALIAN 1 2 TABLE ST UB310X40.4 6 TABLE ST UB410X53.7 9 10 TABLE LD A150X150X10 MEMBER TRUSS 9 10 DEFINE MATERIAL START ISOTROPIC MATERIAL1 E 2.1e+008 POISSON 0.3 END DEFINE MATERIAL UNIT MMS KN CONSTANTS	MATERIAL MATERIAL1 MEMB 1 2 6 9 10 SUPPORTS 1000 222 PINNED UNIT METER KN LOAD 1 DEAD AND LIVE LOAD MEMBER LOAD 6 UNI GY -4.5 LOAD 2 WIND FROM LEFT JOINT LOAD 4 FX 135 LOAD 3 WIND FROM RIGHT LOAD COMBINATION 4 1 0.75 2 0.75 LOAD COMBINATION 5 1 0.75 3 0.75 PERFORM ANALYSIS PRINT MEMBER FORCES print mem pro FINISH”
---	--

由于 STAAD 的命令有很多变体（例如输出杆件内力的命令既可以用 “PRINT MEM FORCES”,也可以用 “PRINT ANALYSIS RESULT”,而且很多命令都可以用简写),所以读取 STAAD 的输入文件 (*.std) 比读取 STAAD 的输出文件 (*.anl) 稍微复杂一些。根据笔者经验，如果能处理好以下细节，基本上能保证二次开发的程序有良好的适应性和健壮性，能够正确读取大多数 STAAD 文件的信息：

1. 命令大小写是不敏感的，即无论用户输入的命令采用的是大写还是小写还是大小写混杂，都应该正确识别出来；
2. 命令行前面允许有空格，空格数目不限；

3. 允许命令行之间有空行和注释行，注释行以“*”开头，注释行中可以有命令，所以在二次开发工作中，发现注释行中的命令时要忽略，以免造成误判；
4. 几乎所有的命令都有缩写形式，例如“**JOINT COORDINATES**”可以写作“Joi COo”；
5. 有些命令一定会出现在各个模型的输入文件中，比如描述节点坐标和杆件关联号的命令，而有些命令就不一定在每个模型中都出现，比如“MEMB TRUSS”和“BETA”等；
6. 某些命令在输入文件中出现的先后顺序没有强制要求，比如可以先定义支座再定义杆件截面，反之也可以；
7. 如果希望读取输出文件中的内力，需要提示用户在输入文件中加入“PRINT (MEMBER) FORCES”或“PRINT ANALYSIS RESULTS”这几个命令；如果需要读取输出文件中的节点位移，还要检查输入文件中是否有“PRINT (JOINT) DISPLACEMENTS”命令；如果需要获取某根杆件的截面特性参数（比如 IZ, IY 等），请检查输入文件中是否有“PRI MEM PROP”或“PRI ALL”这两个命令中的一个；
8. 命令中经常会有“TO”或“BY”出现，需要对这几个关键字进行解析，避免遗漏信息，比如“PRINT MEMBER FORCES LIST 10 15 to 18”的命令，表示只输出杆件“10”、“15”、“16”、“17”、“18”这 5 根杆件的内力，需要对类似信息做出正确解析；
9. “MEMBER RELEASE”中，允许分别定义杆件的起始端和末端的约束情况，也可能出现只释放起端或只释放末端约束的情况。而且“*** START MY MZ”或“*** END MY MZ”可以多次出现，并可以交替出现；

读取文件，可以采用 c++ 的标准流文件读取，也可以采用 MFC 的 CStdioFile 类，读取的每行信息可以存到 CString 的对象中，CString 类中有足够丰富的函数来对 STAAD 输入文件中的信息进行解析。

针对问题 1，可以用 CString::MakeUpper() 把读入的行都强制改成大写。

针对问题 2，可以使用 CString::TrimLeft() 去除命令行行首的空格；

问题 3 和问题 4，都可以用 CString::Find() 函数来解决；

针对问题 6，可以用 CStdioFile 的 GetPosition() 函数先把一些关键词在输入文件中的位置给记录下来，需要定位到待解析的命令时，用 Seek() 函数就可以。问题 5, 7 也就迎刃而解。

问题 8，MFC 提供了一个叫做动态数组的类 CArray，可以动态增长，用 CArray::SetAtGrow() 即可增加元素，这样就既不浪费内存，又可灵活适应各种规模的模型。先把每行的信息分解成单个单词，存储在一个临时的 CArray 对象中，然后判断其中如果有关键词“TO”时，把其中隐含的信息显示展开，存到 CArray 对象中或者 CList 链表对象中。

存储数据的两种方案的对比：

文件读取工作比较简单一些，读取完毕后采用什么样子的数据结构来存储、以及如何对存储的信息进行一些几何拓扑关系的分析就稍微复杂一些。比如，有以下问题需要解决：

a. 一般而言，每个模型中包含的节点个数不同、杆件个数不同、荷载工况数目也不同，而且 STAAD 还提供了命令，可以灵活控制输出某几种荷载情况下某几根杆件的内力信息或节点位移信息等。这样输出文件 (*.anl) 文件中关于杆件内力的数据的行数也因模型而不同，如何存储这些信息既不浪费内存又能灵活适应各种规模的模型？

b. 对于钢结构的节点设计，要分清哪些杆件是梁、哪些杆件是柱、柱的 BETA 角是多少，梁连到柱的翼缘上还是连到柱的腹板上，连接方式是刚接还是铰接、主次梁的连接等等，这些信息如何存储才便于后面的分析工作？

c. 如果要自动计算柱子的计算长度系数，还要考虑到模型中一般有很多次梁，主梁被次梁打断而产生了很抽象的节点，如何不被这些抽象的节点所迷惑而求出真实的梁的长

度？对于柱子也有类似问题。

由于篇幅有限，本文只讨论问题a：我们可以定义一个临时对象CArray<CString,CString> arstrJointCoordinate 来存储从std文件读出的“**JOINT COORDINATES**”命令后的所有节点节点编号和节点坐标：（节点1000的编号、节点1000的x坐标值、节点1000的y坐标值、节点1000的z坐标值；节点222的编号、节点222的x坐标值、节点222的y坐标值、节点222的z坐标值.....）。存储信息的目的是为了查找，arstrJointCoordinate的这种组织方式过于零散，只是一个临时居所，为了使数据结构更加清晰，我们可以让这些更紧凑些，现在再定义一个节点类wJoint，该类中有4个成员变量：

```
Class wJoint
{
    Private:
        double m_dX;           /// x 坐标
        double m_dY;           /// y 坐标
        double m_dZ;           /// z 坐标
        int m_nID;             /// 节点编号
};
```

这样，arstrJointCoordinate中的节点信息就可以转存到这个节点类里。如何转存，这里有两个方案：一、定义一个CArray<wJoint,wJoint> m_arJoint对象。在STAAD中，节点的编号只能是正整数，一个模型当中的节点编号可以不连续，可以跳跃很大，而且STAAD定义节点坐标时，对节点顺序没有任何限制，可以先定义节点1000，再定义节点222，我们读取“**JOINT COORDINATES**”命令后的节点编号和节点坐标后存到rstrJointCoordinate中的节点信息的顺序跟用户定义的顺序完全一致。当我们把arstrJointCoordinate中的信息顺序转存到m_arJoint中之后，m_arJoint[0]的信息不是节点“0”的信息，而是节点“1000”的信息，m_arJoint[1]的信息就不是节点“1”的信息，而是节点“222”的信息。这样我们就无法通过数组下标来随机读取（random access）节点信息，只好遍历所有节点信息来匹配，直到找到所需要的节点的信息，比如，我们可能需要这样一个循环来找节点编号为iJointID的节点的x坐标：

```
double FindCoordiante_X (int iJointID)
{
    for(idx = 0; idx < m_arJoint.GetSize(); idx++)
    {
        if(iJointID == m_arJoint[idx].m_nID )
        {
            return m_arJoint[idx].m_dX;
        }
    }
}
```

上述搜索方式就是所谓的线性搜索（即顺序搜索），在等概率情形下，搜到一个目标的平均搜索长度为 $(n+1)/2$ ，众所周知，这种搜索方式的效率不够高。

所以，CArray<wJoint,wJoint>这种存储方式不够好。对于这种根据关键码（此处关键码为节点号）迅速查找相关信息的操作，选用一种叫做哈希表（Hash table，也被译作“散列表”）的数据结构再合适不过了，以下介绍用这种方式来存储节点信息的方案（即方案二）：

MFC 里提供了一个叫做 CMap 的类，该类就是用哈希表这种数据结构实现的。我们可以

这样定义一个对象，CMap<int,int,WJoint,WJoint> m_mapJoint。把 arstrJointCoordinate 中的信息转存到 m_mapJoint 中，这时我们再搜寻某节点信息时，就不用费劲从循环中匹配了，只需 m_mapJoint.Lookup(nJointID,wJot)一行代码就得到了节点 nJointID 对应的节点信息 wJot(wJot 是 wJoint 类的一个对象)。更为重要的是，只要对 Map 进行正确设置，Lookup 函数通常能够一次到位查找到任意元素，而很少需要进行两次或者三次以上的查找比对。

函数 Lookup(ARG_KEY key, VALUE& rValue) 就是通过哈希算法来找到关键码为 key 的 Value。查找效率主要取决于产生哈希值的算法（计算简单而且产生的哈希值尽量不重复者为佳）和哈希表的大小。哈希表的大小的设置可以用 CMap::InitHashTable(nhashSize) 来完成，微软建议 nhashSize 最好是质数，其大小最好是实际存储元素数目的 120%左右，其默认值是 17。如果您对效率还不满意，还可以自定义哈希函数 HashKey()，以减少哈希地址的冲突，对于大多数数据类型而言，MFC 的 CMap 的 HashKey() 的默认实现方式是对 key 右移四位（即除以 16）。

对于杆件，也可以用类似的方法，定义一个杆件类，其中的成员变量包括杆件编号，起点、终点、杆件长度、杆件 BETA 角、杆件两端的约束情况等等，然后定义一个 CMap<int,int,WMember,WMember> m_mapMember 样子的对象来存储各杆件信息就可以了。

除了上述几何信息之外，另外比较有用的数据就是 STAAD 的杆端力，这些信息在 STAAD 的输出文件中（后缀为 anl），每根杆件在各种工况下的起点和末点的 6 个自由度的内力列举得一清二楚，格式如下：

MEM BER	LOAD	JT	AXIAL	SHEAR -Y	SHEAR- Z	TORSIO N	MOM- Y	MOM-Z
1	100	1	164.73	-1.76	-1.62	0	0	0.00
		2	-164.43	1.76	1.62	0	0.49	-0.53
	101	1	213.51	-2.61	-2.03	0	0	0.00
		2	-213.26	2.61	2.03	0	0.61	-0.78
2	100	2	126.97	-14.11	0.38	0	-0.49	-18.07
		3	-125.01	14.11	-0.38	0	-0.28	-10.16
	101	2	158.71	-21.82	0.48	0	-0.61	-26.95
		3	-157.03	21.82	-0.48	0	-0.35	-16.69
3	100	3	58.57	-14.73	-0.14	0	0.28	-10.78
		4	-56.61	14.73	0.14	0	0	-18.68
	101	3	65.06	-15.17	-0.18	0	0.35	-14.03
		4	-63.38	15.17	0.18	0	0	-16.32
4	100	5	164.93	1.71	-1.61	0	0	0.00
		6	-164.63	-1.71	1.61	0	0.48	0.51
	101	5	213.68	2.58	-2.01	0	0	0.00
		6	-213.43	-2.58	2.01	0	0.6	0.77

上述信息如何存储呢？一种方式还是用 CArray，但缺点是将来一旦需要搜索，效率会比较低，单一的 CMap 也力不能及了，因为 CMap 中的 (key,value) 对儿 (pair) 是不允许有重复的 keys 的，以杆件编号为 key 的话，只能保存一种工况下的内力。这时我们可以采用 STL (Standard Template Library, STL 有很多版本，本文采用的是微软 Visual Studio 2003 的 STL) 中的 multimap，multimap 是允许有重复的 keys 的：

```
multimap<int,WMemForces> m_MtlMapMemID_MemForces;
```

其中的 WMemForces 类的成员函数包括了荷载编号、节点编号以及六个自由度的内力。这样，

上述表格中的每行数据就可以打包成一个（杆件ID，WMemForces对象）的对儿，把这个对儿作为一个个元素填入m_MtlMapMemID_MemForces中就可以了。查找杆件iMemID在其端点strJointID处在工况strLoadCase下的杆端力的函数就可以写做：

```
void wReadAnlFile::SearchMemForce(int iMemID,CString strJointID,CString strLoadCase,CString
&strFX,CString &strFY,CString &strMZ)
{
    multimap<int,WMemForces>::iterator iposBeginTmp;
    for (iposBeginTmp = m_MtlMapMemID_MemForces.lower_bound(iMemID); iposBeginTmp!=
m_MtlMapMemID_MemForces.upper_bound(iMemID); ++iposBeginTmp)
    {
        if (iposBeginTmp->second.m_strLoadID == strLoadCase)
        {
            strFX = iposBeginTmp->second.m_strFX;
            strFY = iposBeginTmp->second.m_strFY;
            strMZ = iposBeginTmp->second.m_strMZ;
            return;
        }
    }
}
```

这样的代码看起来比较容易理解，而且查找效率比较高。

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	杆件 编号	荷 载 编 号	杆 件 端 点	FY	MZ	FX	连接类型	杆件 属性	Beta 角	杆件截面	被 连 接 杆 件	被连接 杆件属 性	被连接杆 件截面	被 连 接 杆 件	反端 杆件 编 号	反端 杆件 截 面	反端 杆端 FY	反端 杆端 MZ	斜撑 夹 角	斜撑 轴 力
2	7	7	2	-20.57	-56.57	-4.84	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	-0.86
3	7	8	2	-0.01	0.11	-2.89	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	-0.73
4	7	77	2	-3.46	-9.51	-0.8	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	-0.14
5	7	88	2	0	0.02	-0.45	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	-0.12
6	7	1	2	16.34	13.28	-8.4	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	4.42
7	7	2	2	9.35	7.37	-4.95	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	1.16
8	7	3	2	-0.98	-2.62	-2.52	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	-1.86
9	7	4	2	0	0.58	7.85	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	24.11
10	7	5	2	-20.98	-57.69	2.38	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	-0.47
11	7	6	2	0	0.01	-4.07	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	3.52
12	7	55	2	-14.88	-40.92	1.7	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	-0.33
13	7	66	2	0	0.01	-2.9	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	2.51
14	7	9	2	0	0	0	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	0
15	7	14	2	0	0	0	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	0
16	7	100	2	22.87	18.59	-11.75	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	6.19
17	7	101	2	34.58	27.74	-18	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	7.16
18	7	102	2	33	23.55	-22.03	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	4.19
19	7	103	2	36.13	31.93	-13.97	104BCFlan-Fix	BEAM	0	ST W12X30	1	COLUMN	ST W12X40	0					1.141	10.13

算例：

数据量和数据内容：84584 行数据（194 根杆件×2 个端点×218 种工况），每行数据融合了 STAAD 输入文件中的杆件编号、杆件端点号、杆件截面、杆件 BETA 角、被连接杆件编号、被连接杆件截面、梁反端的梁的编号和截面以及 STAAD 输出文件中的相应的杆件的荷载编号、杆端剪力、弯矩和轴力、梁之反端梁的内力、杆件端点处连接的斜撑的内力等信息，如上图所示。上述信息不但罗列了*.anl 文件中的各工况下各梁端内力信息，如果该梁端还有水平支撑，还要把其在各相应工况下的内力也一并输出，有大量的搜寻匹配操作。

计算机配置：WindowsXP 操作系统，512M 内存，CPU: Intel Pentium 4 2.93GHz。

其他外部条件：测试程序时均无其他程序运行。

采用 CArray 存储信息并进行分析（第一种方案），输出这些信息共耗费 36 分钟 18 秒。

采用 CMap 以及 multimap 存储信息并进行分析（第二种方案），读取同样模型输出同样信息只耗费不到 7 秒钟，大约是第一种方案的 1/300。

结论：

在大型的结构分析程序中，采用散列表这种数据结构来存储结构几何信息、内力信息等，比用数组来储要更灵活、更高效，特别是有许多查找操作的时候。

参考文献：

《C++标准程序库—自修教程与参考手册》作者： NICOLAI M.JOSUTTIS
译者：侯捷 孟岩 出版社：华中科技大学出版社 出版日期：2002-9-1