c++逆向代码建模工具概要设计说明书

|  |  |
| --- | --- |
| 编制 | 杨铮 |
| 审核 |  |
| 批准 |  |

# 软件架构说明

## 系统说明

c++逆向代码建模工具，是基于doxygen静态代码分析工具产生的xml数据库，进行c++语法元素（包括函数、复合数据类型、宏定义、变量、枚举类型、c++内建对象）解析、索引，数据类型间相关性追溯的工具。软件基于python语言开发，提供GUI界面的检索工具和python软件开发包。

## 运行环境

软件的可运行环境，包括所有能够安装和执行python 3.5的环境。

python三方依赖库：PyQt4、ElementTree、sax、re、numpy、sicpy

## 基本设计概念及模块划分列表

软件通过将功能划分为若干个模块，尽可能降低模块间耦合度，模块划分如下：

|  |  |  |
| --- | --- | --- |
| 模块编号 | 模块名 | 模块来源 |
| PD-001 | metadata模块 | 新建 |
| PD-002 | xml软件模型解析模块 | 新建 |
| PD-003 | 复合数据类型结构解析模块 | 新建 |

其中，metadata模块定义了描述c++基本语法元素的对象及其行为；xml软件模型解析模块，功能是解析doxygen工具产生的xml软件模型，并建立语法元素间的索引，该模块又包含全局索引文件解析模块和描述文件解析模块，分别用于解析全局索引xml文件和语法元素的描述xml文件；由于doxygen工具产生的xml模型不能识别c++复合数据类型中的内联匿名复合数据类型，因此需要使用Visual AssistX产生的outline信息辅助建模，以构建正确完整的复合数据类型内部结构模型，而复合数据类型结构解析模块则负责解析复合数据类型的outline信息。

内部架构图如下：

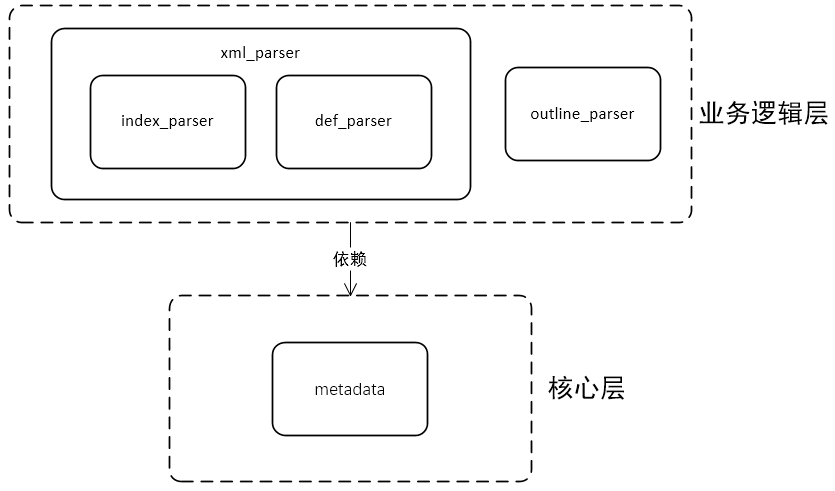


图 软件架构图

## 模块与应用场景描述

利用工具得到c++头文件的软件模型后，可以将模型映射为其他类型的编程语言，比如映射为jna头文件、c#头文件。

## 可测试性、维护及升级设计

无

## 可配置设计

无

## 尚未解决问题列表

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 问题名称 | 问题描述 | 预计解决日期 |
| YL-001 | class类型未处理 | 由于目前建模的头文件代码中未出现class类型，所以目前暂未处理 | 未定 |
| YL-002 | 内联匿名复合类型重命名问题 | 若一个复合数据类型中包含多个匿名内敛复合数据类型，且匿名内联复合类型中包含了同名成员变量。此时doxygen建立的软件模型有误，会漏掉匿名内联复合类型中同名成员。<Doxygen技术限制> | 未定 |

# 模块说明

## PD-001 metadata模块

### 模块功能描述

metadata模块，是c++逆向建模工具的核心模块，其中定义了对c++基本语法元素的描述，包括函数、复合数据类型、枚举类型、变量、重命名、宏定义等。其他模块均依赖于此模块。

### 模块设计说明

类图如下：

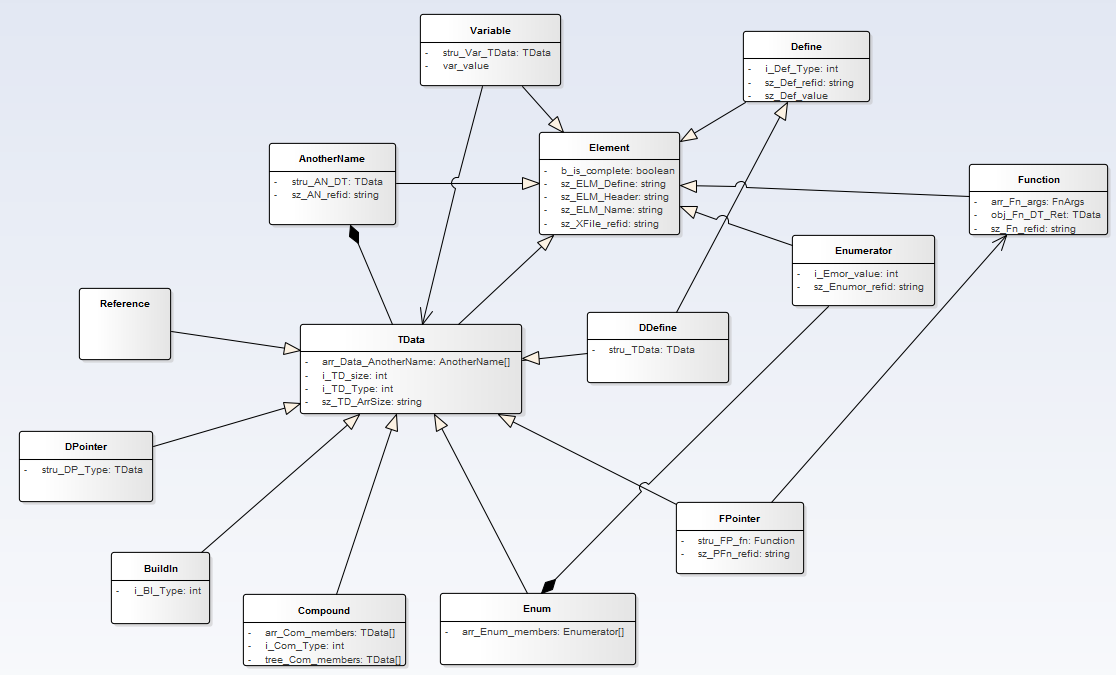


图 metadata模块类图

经过对c++基本语法的分析，将其语法元素分为数据类型语法元素和非数据类型语法元素两大类。所有语法元素都派生于基类Element，其成员如下：

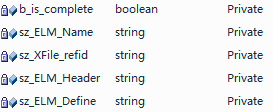


图 Element成员变量

每一个语法元素都必须包含sz\_ELM\_Define原始代码声明，sz\_ELM\_Header声明头文件，sz\_XFile\_refid xml索引id，sz\_ELM\_Name元素名及b\_is\_complete表示语法元素是否建立检索几种成员变量。

非数据类语法元素包括Define宏定义语法元素、Function函数语法元素、Variable变量语法元素、AnotherName重命名语法元素、Enumerator枚举项语法元素。

其中，Function成员定义如下：

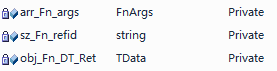


图 Function成员变量

arr\_Fn\_args为参数列表，其中每一个元素必须是一个TData或者其派生对象，sz\_Fn\_refid为函数对象的索引ID，obj\_Fn\_DT\_Ret为返回值对象，必须是TData数据类型。

Variable对象描述一个变量，成员如下：



图 Variable成员变量

stru\_Var\_TData为变量数据类型，var\_value为变量值。

AnotherName对象描述一个重命名对象，成员变量如下：



图 AnotherName成员变量

stru\_AN\_DT为原数据类型，sz\_AN\_refid为索引ID。

数据类型语法元素对象以TData为基类，成员变量如下：

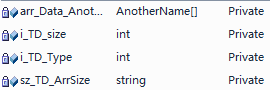


图 TData成员变量

i\_TD\_Type为标识数据类型，数据类型如下，

dict\_TD\_Type = {

0: 'Unknown',

1: 'DPointer',

2: 'Reference',

3: 'BuildIn',

4: 'Compound',

5: 'Enum',

6: 'DDefine',

}

sz\_TD\_ArrSize为如果是类型数组，标记数组大小，字符串类型的原因是可能为宏定义或数值常量，i\_TD\_size类型占用内存大小，arr\_Data\_AnotherName重命名列表。

其中，Compound对象描述一个复合数据类型，成员如下：

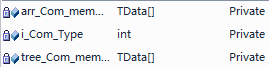


图 Compound成员

arr\_Com\_members成员列表，元素类型也必须是Variable 或者 Function；i\_Com\_Type复合类型，如下：

dict\_Com\_Type = {

0: 'Unknown',

1: 'Struct',

2: 'Class',

3: 'Union',

}

tree\_Com\_members结构化的成员列表，为了适配包含内联复合类型成员的情况

## PD-002 xml软件模型解析模块

### 模块功能描述

xml软件模型解析模块的主要功能是解析，doxygen工具产生的c++代码静态xml软件模型。识别各种数据类型，并建立语法元素间关系，实现对语法元素的检索。

### 子模块划分列表

|  |  |  |
| --- | --- | --- |
| 模块编号 | 模块名称 | 模块来源 |
| PD-002-001 | 全局索引文件解析模块 | 新建 |
| PD-002-002 | 语法元素描述文件解析模块 | 新建 |

### 子模块设计说明

针对doxygen产生的xml文件节点组织遵循固定的格式，文件大小较大的特点并结合解析功能中按顺序读取的xml文件且不更改文件的需求。决定在第一次读取xml文件时采用回调的思想解析xml文件，优点是占用内存少。在第一次遍历xml文件时，需要建立节点的索引，需要记住{节点名、节点类型、文件相对位置}，一遍之后更能够快速的索引到文件的指定节点，使用python xml.sax包解析xml文件。

采用此方法，有个前提是假设用户知道要解析的文件的格式。

#### PD-002-001 全局索引文件解析模块

**功能描述：**

全局索引文件解析模块的主要功能是解析doxygen产生的index.xml文件。index.xml文件的结构如下：

|  |
| --- |
| <doxygenindex>  <compound refid=$(com\_refid) kind=${struct | class | union | file | dir}>  <name>$(com\_name)</name>  <member refid=$(mem\_refid) kind=${(variable | function) | (define | enum | enumvalue | typedef | function) }>  <name>$(mem\_name)</name>  </member>  </compound>  ...  </doxygenindex> |

从index文件中可以得到全局所有对象的索引键值，其中$(com\_refid)是复合语法结构或者其他语法结构汇总的索引键值，同时也是描述文件的文件名。案例：

<compound refid="struct\_n\_e\_t\_\_\_d\_v\_r\_\_\_e\_t\_h\_e\_r\_n\_e\_t" kind="struct">...</compound>



特别的，当compound.kind = dir时，compound节点描述文件夹路径，可忽略。

compound.name节点的内容$(com\_name)，在kind取struct | class | union时表示复合数据结构名，需要与compound->refid进行关联。

compound.member表示成员语法对象，若kind取struct | class | union时表示一个元素，当kind取file时表示一个独立语法单元。$(mem\_refid)成员的全文检索键值，member->kind为成员的类型，若compound-> kind取struct | class时可取variable | function；若compound-> kind取union时可取variable；若compound-> kind取file时可取define | enum | enumvalue | typedef | function。

注：

node.son 表示父子节点

node->pro表示节点属性

本模块处理的流程图如下：

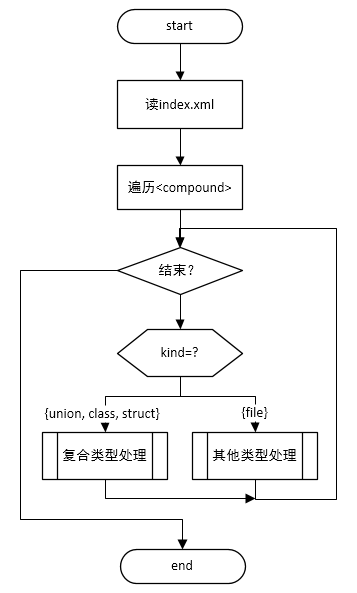


图 全局索引文件解析模块处理流程图

其中，复合类型的处理和其他语法元素类型的处理是不同的，其处理流程分别如下：

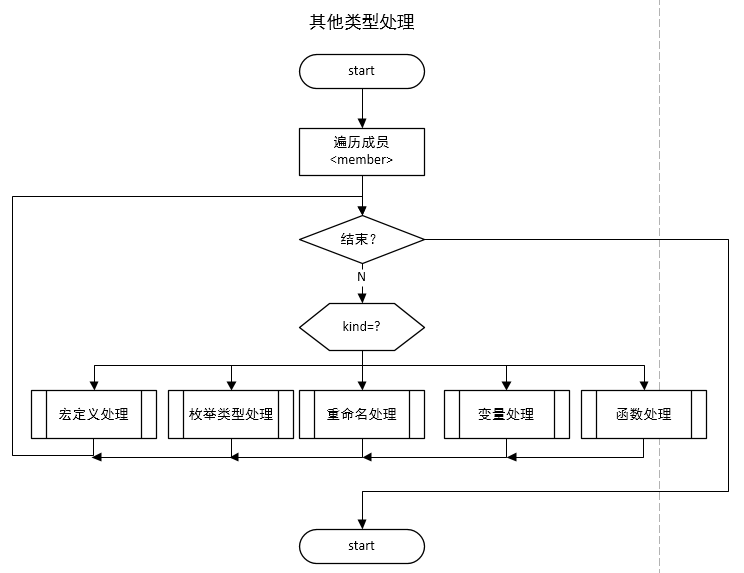
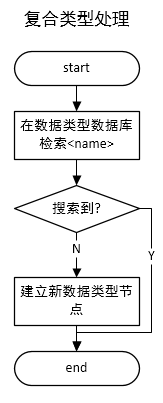


图 （左）复合类型处理流程，（右）其他类型处理流程

模块类图：

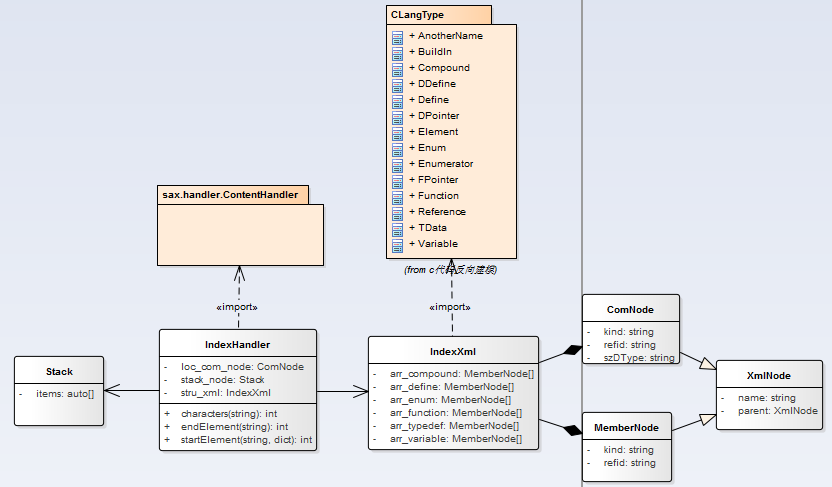


图 PD-002-001模块类图

其中，IndexHandler对象遍历index.xml文件，并解析xml文件中的元素节点，然后将感兴趣的节点，如<compound>、<func>等组织为XmlNode及其派生类对象，按照语法元素类型分别放入IndexXml对象的对应列表中。

IndexXml对象是index.xml的结构化对象，除了负责保存索引文件结构化数据外，还负责语法元素的检索、元素类型判别等操作。

#### PD-002-002 语法元素描述文件解析模块

**功能描述：**

复合数据结构描述文件解析：

|  |
| --- |
| <doxygen>  <compounddef id=$(com\_id) kind=$(struct | class | union) prot=${public | protected | private}>  <compoundname>$(com\_name)</compoundname>  <includes>$(com\_header)</includes>  <innerclass refid=$(inner\_refid) prot=${public | protected | private}>$(iner\_name)</innerclass>  <sectiondef kind=${public-attrib | protected-attrib | private-attrib}>  <memberdef kind=${variable | function} id=$(mem\_id) prot=${ public | protected | private} static=${yes | no}> //若compounddef->kind=union，只能取variable  <type>$(mem\_type)</type> //若compounddef->kind=union, 为空  <argsstring>$(mem\_args)</argsstring>  <name>$(mem\_name)</name>  <param>  <type>$(arg\_type)</type>  <declname>$(arg\_name)</declname>  </param>  ...  </memberdef>  ...  </sectiondef>  ...  </compounddef>  </doxygen> |

$(inner\_refid)是复合数据对象内部定义的复合对象的全局索引键值，也是描述文件名。

$(iner\_name)是内部复合对象名，带作用域

$(mem\_args)是成员变量为数组时这个值为数组长度。当成员类型为variable且复合类型不为union，若成员为数组则此节点定义数组长度，如[16]。特别的，若数组长度定义为常量或宏时，doxygen不会直接为其建立索引，例子：

|  |
| --- |
| <memberdef>  ...  <argsstring>[PATHNAME\_LEN]</argsstring>  ...  </memberdef> |

因此，需要开发人员自己去掉中括号，判断长度值是否是数字，若不是数字，则从已有的索引中匹配是否是宏定义或者常量，然后替换。

当memberdef->kind=function时$(mem\_args)是成员函数的参数列表，不过由于接下来有更详细的参数列表描述，所以忽略。

全局语法元素描述文件解析：

|  |
| --- |
| <doxygen>  <compounddef id=$(com\_id) kind=“file”prot=${public | protected | private}>  <compoundname>$(com\_name)</compoundname>  <innerclass refid=$(inner\_refid) prot=${public | protected | private}>$(iner\_name)</innerclass>  <sectiondef kind=${define | enum | typedef | var | func}>  <memberdef kind=${ define | enum | typedef | variable | function} id=$(mem\_id) prot=${ public | protected | private} static=${yes | no}> //若compounddef->kind=union，只能取variable  $(content)  </memberdef>  ...  </sectiondef>  ...  </compounddef>  </doxygen> |

$(content)是节点的中的内容，由于全局语法元素描述文件定义的语法元素类型多，按照sectiondef->kind划分为若干模块。一个模块中定义若干独立语法元素<memberdef>，因此根据语法元素类型不同，节点中的内容$(content)也不同。

**memberdef->kind="define"**

|  |
| --- |
| <name>$(define\_name)</name>  <initializer>$(define\_init\_value)</initializer> |

特别的，需要识别$(define\_init\_value)是否是内建对象、其他独立宏、类型重命名，若是则需要追溯到已知索引对象或未知对象。

**memberdef->kind="enum"**

|  |
| --- |
| <type></type> // 忽略  <name>$(elem\_name)</name>  <enumvalue id=$(enum\_valu\_id) prot="public">  <name>$(elem\_valu\_name)</name>  <initializer>$(enum\_valu)</initializer>  </enumvalue>  ... |

特别的$(enum\_valu)可能为空也可能有初值，若第一个枚举值有初值，后面的枚举值依次递增

**memberdef->kind="typedef"**

|  |
| --- |
| <type>$(ttype)</type>  <definition>...</definition> //申明，可忽略  <argsstring>$(targs)</argsstring>//当此节点为非空时，表示一个函数指针  <name>$(tname)</name> |

$(ttype)

* 若原类型为内建对象或者未知数据类型，则type节点中直接就是类型名
* 若原类型也是重命名类型，则用<ref refid=$(refid) kindref="member">$(tname)</ref>，此时可以通过索引键值检索到该重命名对象。
* 若原类型为复合类型则<type>{union | struct | class} <ref refid=$(refid) kindref="compound">$(com\_name)</ref> {\*| }</type>

若<argsstring>非空，表示为一个函数重命名（函数指针），此节点中值为函数指针的参数列表，此时需要开发者手动解析。此时$(tname)为函数指针名。

**memberdef->kind="variable"**

|  |
| --- |
| <type>$(vtype)</type>  <definition>...</definition> //申明，可忽略  <argsstring>$(vtargs)</argsstring>  <name>$(vname)</name>  <initializer>$(init\_value)</initializer> |

需要关注memberdef–>static属性

$(vtype)表示变量类型

$(vtargs)变量为数组是有效

$(init\_value)初始化值

**memberdef->kind="function"**

|  |
| --- |
| <type>$(ftype)</type>  <definition>...</definition> //申明，可忽略  <argsstring>...</argsstring>//参数列表，忽略  <name>$(fname)</name>  <param>  <type><ref refid=$(refid) kindref=${member | compound}>$(arg\_type)</ref></type>  <declname>$(arg\_name)</declname>  </param>  ... |

需要关注memberdef–>static属性

$(ftype)表示函数返回值，当有若干修饰符时，会定义在多个<ref>中，开发者需要判断其中哪个才是真正的返回类型，需要排除修饰符；当然也有可能返回值是未知类型，这种情况需要提示。

$(fname)函数名

<param>描述一个参数，param.type为参数类型，同样面临有限定符的情况，也需要从中判断真正的数据类型，限定符只可能是宏定义，此处需要将其展开。

$(arg\_name)为参数名

**模块流程图：**

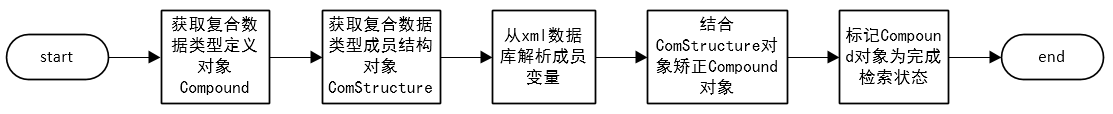


图 复合数据结构描述文件解析流程



图 函数对象的解析流程

**模块类图：**

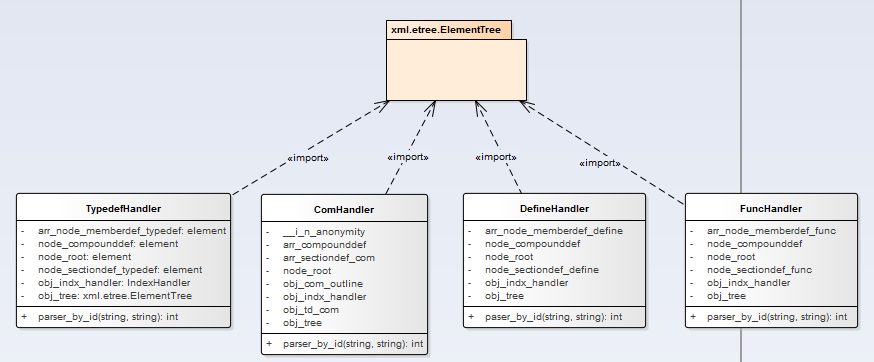


图 模块类图

语法元素描述文件的解析，最主要的就是解析复合数据类型、重命名类型、宏定义及函数对象。每种语法元素分别使用对用的处理对象进行解析。

## PD-003 复合数据类型结构解析模块

### 模块功能描述

由于doxygen产生的c++ xml静态软件模型，不能很好识别复合数据类型中的匿名内联复合类型，导致的到的软件模型存在复合数据类型结构错误的问题。为了解决此问题，增加outline辅助信息，目的是修正复合数据类型软件模型中的成员结构错误。

PD-003模块的功能是解析outline文件。

### 模块设计说明

outline文件内容格式如下，对于如下三种定义结构体名的方式，在outline文件中的表现如下表所示：

|  |  |
| --- | --- |
| c++定义 | outline定义 |
| struct \_\_PLAYRECT {  int x;  int y;  int uWidth;  int uHeight;  } | \_\_PLAYRECT  x;  y;  uWidth;  uHeight; |
| typedef struct {  DWORD dwYear;  DWORD dwMonth;  DWORD dwDay;  DWORD dwHour;  DWORD dwMinute;  DWORD dwSecond;  }NET\_DVR\_TIME, \*LPNET\_DVR\_TIME; | struct  dwYear;  dwMonth;  dwDay;  dwHour;  dwMinute;  dwSecond;  NET\_DVR\_TIME, \*LPNET\_DVR\_TIME; |
| typedef struct tagNET\_DVR\_TIME\_V30 {  WORD wYear;  BYTE byMonth;  BYTE byDay;  BYTE byHour;  BYTE byMinute;  BYTE bySecond;  BYTE byRes;  WORD wMilliSec;  BYTE byRes1[2];  }NET\_DVR\_TIME\_V30,\*LPNET\_DVR\_TIME\_V30; | tagNET\_DVR\_TIME\_V30  wYear;  byMonth;  byDay;  byHour;  byMinute;  bySecond;  byRes;  wMilliSec;  byRes1[2];  NET\_DVR\_TIME\_V30,\*LPNET\_DVR\_TIME\_V30; |

对于内部含有匿名内联复合数据结构的复杂数据类型，outline中以缩进的方式描述各成员变量的相对关系。如下：

|  |  |
| --- | --- |
| c++定义 | typedef struct tagNET\_DVR\_PUSHALARMINFO\_V40 {  DWORD dwAlarmType;  union {  BYTE byRes[300];  struct {  DWORD dwAlarmInputNumber;  BYTE byAlarmOutputNumber[MAX\_ALARMOUT\_V30];  BYTE byAlarmRelateChannel[MAX\_CHANNUM\_V30];  BYTE byRes1[97];  BYTE byDeviceID[NAME\_LEN];  BYTE byRes2[7];  }struIOAlarm;  struct {  BYTE byStreamID[STREAM\_ID\_LEN];  BYTE byRes1[132];  BYTE byChannel[MAX\_CHANNUM\_V30];  BYTE byRes2[33];  BYTE byDeviceID[NAME\_LEN];  BYTE byRes3[7];  }struStreamIDorChannel;  struct {  BYTE byRes1[228];  BYTE byDiskNumber[MAX\_DISKNUM\_V30];  BYTE byDeviceID[NAME\_LEN];  BYTE byRes2[7];  }struDiskAlarm;  }uAlarmInfo;  }NET\_DVR\_PUSHALARMINFO\_V40, \*LPNET\_DVR\_PUSHALARMINFO\_V40; |
| outline定义 | tagNET\_DVR\_PUSHALARMINFO\_V40  dwAlarmType;  union  byRes[300];  struct  dwAlarmInputNumber;  byAlarmOutputNumber[MAX\_ALARMOUT\_V30];  byAlarmRelateChannel[MAX\_CHANNUM\_V30];  byRes1[97];  byDeviceID[NAME\_LEN];  byRes2[7];  struIOAlarm;  struct  byStreamID[STREAM\_ID\_LEN];  byRes1[132];  byChannel[MAX\_CHANNUM\_V30];  byRes2[33];  byDeviceID[NAME\_LEN];  byRes3[7];  struStreamIDorChannel;  struct  byRes1[228];  byDiskNumber[MAX\_DISKNUM\_V30];  byDeviceID[NAME\_LEN];  byRes2[7];  struDiskAlarm;  uAlarmInfo;  NET\_DVR\_PUSHALARMINFO\_V40, \*LPNET\_DVR\_PUSHALARMINFO\_V40; |

通过outline文件，可以清楚获得任意复合数据类型的成员结构。

**模块流程图：**

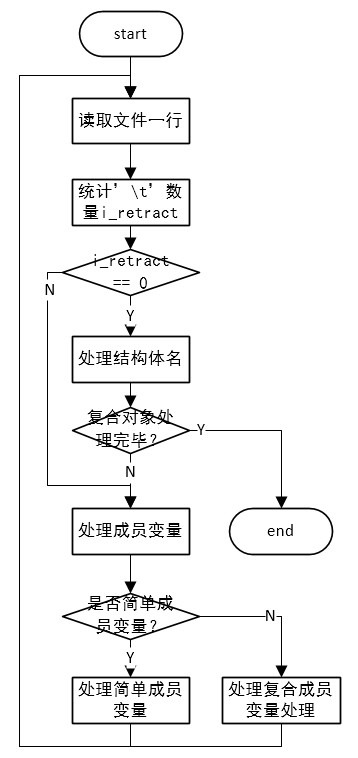


图 PD-003模块流程图

每一次处理流程只会解析一个结构体对象，为了找寻指定名称结构体，需在解析完成后，匹配目标对象名和当前对象的全部名称。

**模块类图：**

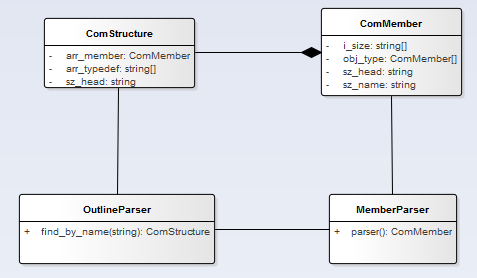


图 PD-003模块类图

其中，OutlineParser对象负责解析一个复合类型对象，结果保存在ComStructure对象中；MemberParser负责解析复合类型的成员变量，结果保存在ComMember对象中，ComStructure中包含若干ComMember。

# 接口说明

c++逆向建模工具除了提供界面外，也提供软件开发包，供其他开发人员二次开发。

## PD-002 xml软件模型解析包接口

通过PD-002 xml软件模型解析包提供的接口，开发者可以获取并使用解析后的结构化软件模型。

### 全局索引文件index.xml解析接口

**接口原型：**

def **index\_execute**(\_path) –> IndexHandler

**参数：**

\_path [in] (str) index.xml文件的路径

**返回值：**

返回IndexHandler类型对象，其中包含了本次解析的结果

### 根据索引id解析复合类型接口

**接口原型：**

def **find\_com\_by\_id**(\_refid, \_hIndx=None) –> Compound

**参数：**

\_refid [in] (str) 复合数据类型的索引ID

\_hIndx [in] (IndexHandler) 全局索引句柄对象，index\_execute()接口的返回值

**返回值：**

解析成功返回Compound类型对象，失败则返回空

### 根据索引id解析函数对象接口

**接口原型：**

def **find\_fn\_by\_id**(\_refid, \_def\_file, \_hIndx=None) –> Function

**参数：**

\_refid [in] (str) 复合数据类型的索引ID

\_def\_file [in] (str) 描述函数对象的描述文件路径

\_hIndx [in] (IndexHandler) 全局索引句柄对象，index\_execute()接口的返回值

**返回值：**

解析成功返回Function类型对象，失败则返回空

### 根据索引id解析重命名对象接口

**接口原型：**

def **find\_an\_by\_id**(\_refid, \_def\_file, \_hIndx=None) –> Function

**参数：**

\_refid [in] (str) 复合数据类型的索引ID

\_def\_file [in] (str) 描述函数对象的描述文件路径

\_hIndx [in] (IndexHandler) 全局索引句柄对象，index\_execute()接口的返回值

**返回值：**

解析成功返回AnotherName类型对象，失败则返回空

## PD-003 复合数据类型结构解析包接口

### 根据复合类型名称检索outline文件接口

**接口原型：**

def **get\_outline\_by\_name** (\_file\_path, \_name) –> ComStructure

**参数：**

\_file\_path [in] (str) outline文件路径

\_name [in] (str) 要检索的复合类型名

**返回值：**

检索成功返回ComStructure类型对象，失败则返回空

# 异常处理

## 函数返回值错误判断

按照python的开发规范，程序设计的所有接口或成员函数返回值，若接口调用失败或错误均返回空None。

因此需要接口调用后需要作如下判断：

|  |
| --- |
| def func():  …  return ret  if func() is None:  // TODO  else:  // TODO: |

## 函数参数有效性检测

按照python开发规范，带参数的函数都须做有效性检测。因为python属于弱类型语言，不对变量做类型限制，所以实际传入函数的参数可以是任意类型，为了避免程序崩溃方便排查问题，需要对参数类型有效性检测。

|  |
| --- |
| def func1(sz\_name, i\_size):  if (type(arg1) is not str) or (type(i\_size) is not int):  raise “parm type error” |

## 关键代码执行结果异常处理

按照python规范，对文件的操作使用with as语法，保证文件操作完成后的close操作。对有明确执行结果要求的关键语句，做assert保护，保证出现异常第一时间捕获并修复。