多语言代码映射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

多语言代码映射工具，在c++逆向建模工具产生的软件模型基础上，根据c++到目标语言的映射规则，自动产生目标语言代码。工具的目标是批量产生c++到目标语言的数据结构及接口代码。

## 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：无

## 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 | 映射框架模块 | 新建 |
| PD-002 | jna代码映射规则模块 | 新建 |
| PD-003 | csharp代码映射规则模块 | 新建 |

其中，为了统一代码映射的接口和调用，设计映射框架模块。jna代码映射规则模块规定了c++代码到jna代码的映射规则，并产生jna代码。csharp代码映射规则模块规定了c++代码到csharp代码的映射规则，并产生charp代码。

## 模块与应用场景描述

利用工具得到jna代码或者csharp代码，可以直接在java、Android、csharp等程序中集成hcnetsdk，大大减少工作量，提高开发效率。

## 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 | 内联匿名复合类型重命名问题 | c++逆向建模工具遗留问题YL-001导致。 | 未定 |

# 模块说明

## PD-001 映射框架模块

### 模块功能描述

映射框架模块统一代码映射的接口和调用方式。

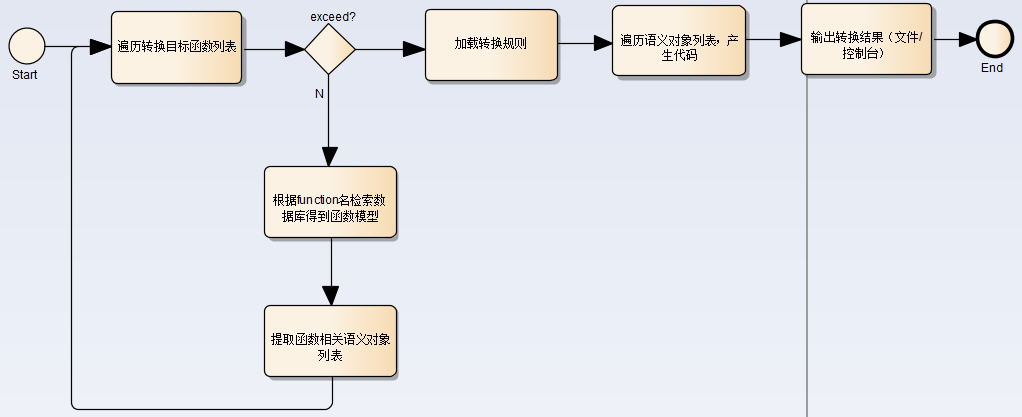


图 代码映射流程图

### 模块设计说明

类图如下：

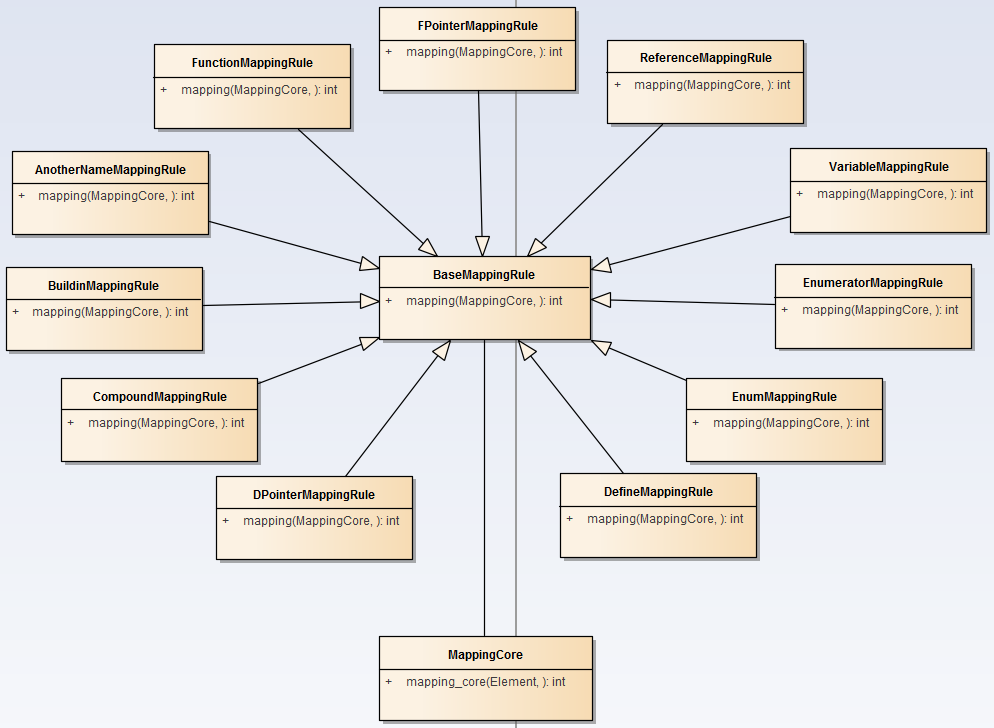


图 映射框架模块类图

BaseMappingRule是一个结构对象，提供共有方法mapping()，实现代码映射。每一种c++语法元素均从其派生。为了实现多语言代码转换，设计对象MappingCore对象，该对象是具体语言的代码映射规则。Jna映射模块和chsarp映射模块均从此对象派生。

## PD-002 jna代码映射规则模块

### 模块功能描述

利用c++ xml软件模型，实现到jna代码的映射，产生jna代码。

### 模块设计说明

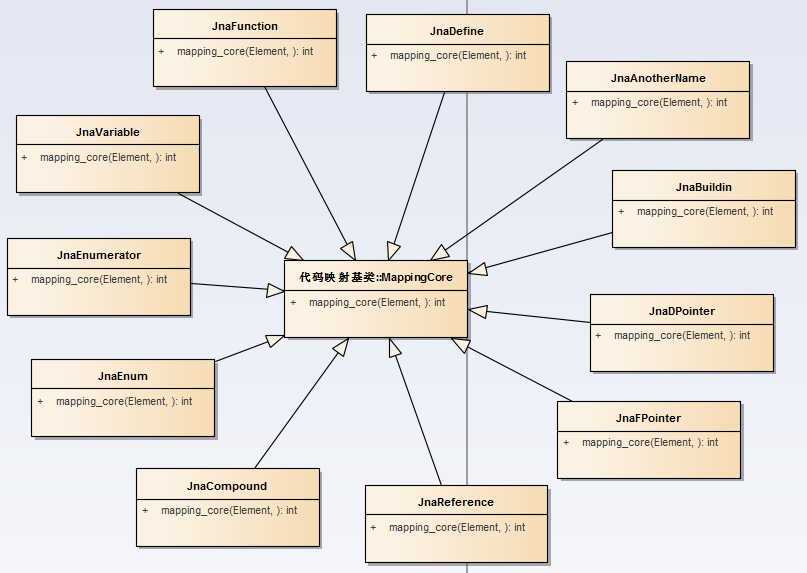


图 jna代码映射模块类图

其中，派生于MappingCore对象的为各语法元素的c++->jna映射股则

转换规则：

Java和C原生数据类型的对应表如下：

|  |  |  |
| --- | --- | --- |
| **Java 类型** | **C 类型** | **内存表现** |
| boolean | int | 32位整数 (可定制) |
| byte | char | 8位整数 |
| char | wchar\_t | 平台依赖 |
| short | short | 16位整数 |
| int | int | 32位整数 |
| long | long long, \_\_int64 | 64位整数 |
| float | float | 32位浮点数 |
| double | double | 64位浮点数 |
| Buffer/Pointer | pointer | 平台依赖(32或 64位指针) |
| <T>[] (基本类型的数组) | pointer/array | 32或 64位指针(参数/返回值)  邻接内存(结构体成员) |
| String | char\* | /0结束的数组 (native encoding or jna.encoding) |
| WString | wchar\_t\* | /0结束的数组(unicode) |
| String[] | char\*\* | /0结束的数组的数组 |
| WString[] | wchar\_t\*\* | /0结束的宽字符数组的数组 |
| Structure | struct\*/struct | 指向结构体的指针 (参数或返回值) (或者明确指定是结构体指针) 结构体(结构体的成员) (或者明确指定是结构体) |
| Union | union | 等同于结构体 |
| Structure[] | struct[] | 结构体的数组，邻接内存 |
| Callback | <T> (\*fp)() | Java函数指针或原生函数指针 |
| NativeMapped | varies | 依赖于定义 |
| NativeLong | long | 平台依赖(32或64位整数) |
| PointerType | pointer | 和 Pointer相同 |

结构体转换规则：

|  |
| --- |
| c头文件 |
| struct UserStruct {  long id;  wchar\_t\* name;  int age;  };  #define MYLIBAPI extern "C" \_\_declspec( dllexport )  MYLIBAPI void sayUser(UserStruct\* pUserStruct); |
| jna定义代码 |
| /\*  \* 定义一个类，模拟C语言的结构  \* \*/  public static class UserStruct extends Structure{             public NativeLong id;             public WString name;             public int age;  }  public void sayUser(UserStruct.ByReference struct); |
| Java调用代码 |
| UserStruct userStruct=new UserStruct ();  userStruct.id=new NativeLong(100);  userStruct.age=30;  userStruct.name=new WString("沈东良");  TestDll1.INSTANCE.sayUser(userStruct); |

union对象映射关系：

|  |
| --- |
| c头文件 |
| union UTest4  {  struct UTest4\_S0  {  int m\_0\_0;  char m\_0\_1;  }m\_0;  struct UTest4\_S1  {  long m\_1\_0;  char\* m\_1\_1;  }m\_1;  STest0 m\_2;  };  JNA\_TEST\_DLL\_API UTest4\* fnjna\_test\_dll3(UTest4 arg0, UTest4\* arg1); |
| jna定义代码 |
| public static class UTest4 extends Union {  // 定义内联结构体  public static class UTest4\_S0 extends Structure {  public int m\_0\_0;  public byte m\_0\_1;  @Override  protected List<String> getFieldOrder() {  return Arrays.asList("m\_0\_0", "m\_0\_1");  }  public static class ByReference extends UTest4\_S0 implements Structure.ByReference{  }  public static class ByValue extends UTest4\_S0 implements Structure.ByValue{  }  }  // 定义内联结构体  public static class UTest4\_S1 extends Structure {  public NativeLong m\_1\_0 = new NativeLong();  public Pointer m\_1\_1;  @Override  protected List<String> getFieldOrder() {  return Arrays.asList("m\_1\_0", "m\_1\_1");  }  public static class ByReference extends UTest4\_S1 implements Structure.ByReference {  }  public static class ByValue extends UTest4\_S1 implements Structure.ByValue {  }  }  // 成员变量  public UTest4\_S0 m\_0 = new UTest4\_S0();  public UTest4\_S1 m\_1 = new UTest4\_S1();  public STest0 m\_2 = new STest0();  @Override  protected List<String> getFieldOrder() {  return Arrays.asList("m\_0", "m\_1", "m\_2");  }  public static class ByReference extends UTest4 implements Union.ByReference {  }  public static class ByValue extends UTest4 implements Union.ByValue {  }  }  public UTest4.ByReference fnjna\_test\_dll3(UTest4 arg0, UTest4 arg1); |
| java调用 |
| jna\_test1.jna\_test1\_dll.UTest4 union7 = new jna\_test1.jna\_test1\_dll.UTest4();  jna\_test1.jna\_test1\_dll.UTest4 union8 = new jna\_test1.jna\_test1\_dll.UTest4();  // 要从jna获取联合体中某个类型数据，必须先设置联合体元素类型  union7.setType(jna\_test1.jna\_test1\_dll.UTest4.UTest4\_S0.class);  // 获取值  jna\_test1.jna\_test1\_dll.UTest4.UTest4\_S0 utest4\_s0 = (jna\_test1.jna\_test1\_dll.UTest4.UTest4\_S0 utest4\_s0)union7. m\_0. getTypedValue(jna\_test1.jna\_test1\_dll.UTest4.UTest4\_S0 utest4\_s0.class);  utest4\_s0. m\_0\_0 = 100;  utest4\_s0. m\_0\_1 = ‘A’;  jna\_test1.jna\_test1\_dll.mInstance.fnjna\_test\_dll3(union7, union8); |

## PD-003 csharp代码映射规则模块

### 模块功能描述

利用c++ xml软件模型，实现到csharp代码的映射，产生csharp代码。

### 模块设计说明

无。

# 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。

## 复合数据类型映射接口

**接口原型：**

def mapping\_jna\_compound (\_ref) –> str

**参数：**

\_ref [in] (str) 复合类型全局索引值

**返回值：**

返回映射产生的jna代码

**接口原型：**

def mapping\_charp\_compound (\_ref) –> str

**参数：**

\_ref [in] (str) 复合类型全局索引值

**返回值：**

返回映射产生的csharp代码

## 复合数据类型映射接口

**接口原型：**

def **mapping\_jna\_fn** (\_refid) –> str

**参数：**

\_refid [in] (str) 函数的索引ID

**返回值：**

返回映射产生的jna代码

**接口原型：**

def **mapping\_csharp\_fn** (\_refid) –> str

**参数：**

\_refid [in] (str) 函数的索引ID

**返回值：**

返回映射产生的csharp代码

## 复合数据类型映射接口

**接口原型：**

def **mapping\_jna\_cb** (\_refid) –> str

**参数：**

\_refid [in] (str) 回调函数类型的索引ID

**返回值：**

返回映射产生的jna代码

**接口原型：**

def **mapping\_csharp\_cb** (\_refid) –> str

**参数：**

\_refid [in] (str) 回调函数类型的索引ID

**返回值：**

返回映射产生的csharp代码