Привет всем!

Сегодня мы с вами поговорим о любви C/C++ к Java. Связь не очевидна, но она есть. Итак, поехали! ;)

В прошлый раз мы с вами говорили об обратной задаче, а стоил ли когда-нибудь выполнять эту?

Очевидно, что и такое нам тоже нужно, когда например нашему методу передаётся объект, который предоставляет нам какие-то услуги, которые мы в свою очередь должны выполнить. Ну или опять банальный пример, нам попросту нужно вызывать выполнять некоторые задачи из другого проекта в среде своего проекта.

Рассмотрим опять что-нибудь не замысловатое, чтобы не пудрить сильно мозги. ;)

Допустим, у нас есть Java класс, в котором реализован метод, который приветствует другой объект. Вот как выглядит класс с методом:

public class Hello {

public static int HelloObject(PeopleObject people) {

System.out.println("Height" + people.height);

System.out.println("Name:" + people.name);

System.out.println("Town:" + people.town);

System.out.println("Age:" + people.age);

return OK;

}

}

Вот как выглядит сам класс:

public class PeopleObject {

public int height;

public String name;

public String town;

public int age;

public PeopleObject(int height, String name, String town, int age) {

this.height = height;

this.name = name;

this.town = town;

this.age = age;

}

}

Со стороны Java всё готово, теперь подготовим площадку в C/C++, для вызова нашего метода.

Для того что бы вызвать наш метод из C/C++, вначале нам понадобится загрузить (создать) JVM (виртуальную Java машину) используя следующее:

JNIEnv \* loadJVM(JavaVM \*\* jvm) {

JavaVMOption jvmOptions;

jvmOptions.optionString = "-Djava.class.path=C:\\JavaClassFiles";

JavaVMInitArgs jvmArgs;

jvmArgs.version = JNI\_VERSION\_1\_6;

jvmArgs.nOptions = 1;

jvmArgs.options = &jvmOptions;

jvmArgs.ignoreUnrecognized = JNI\_TRUE;

JNIEnv \* env;

int OK = JNI\_CreateJavaVM(jvm, (void\*\*)&env, &jvmArgs);

if(OK < 0) printf("\nUnable to Launch JVM\n");

return env;

}

И так по порядку:

jvmOptions.optionString = "-Djava.class.path=C:\\JavaClassFiles";

в этой строке мы указываем папку исходников, где располагаются файлы классов.

Далее

jvmArgs

описывает аргументы для вызова Java машины.

jvmArgs.version = JNI\_VERSION\_1\_6;

jvmArgs.nOptions = 1;

jvmArgs.options = &jvmOptions;

jvmArgs.ignoreUnrecognized = JNI\_TRUE;

в первой строчке мы задаём версию Java, во второй мы задаём кол-во опций, так как они хранятся в массиве, в данном случае мы задаём опцию для Java машины, и в четвёртой строке мы указываем игнорировать все нераспознанные опции, строка которых начинается с "-X" or "\_".

Теперь вызовем создание Java машины:

JNIEnv \*env;

JavaVM \* jvm;

env = loadJVM(&jvm);

if (env == NULL)

return BAD;

А теперь всё готово для начала работы с Java методами, объектами.

Для начала найдём два наших класса:

jclass clsH = env->FindClass("Hello");

jclass clsP = env->FindClass("PeopleObject");

Метод “FindClass” ищет класс по имени.

Теперь у класса “Hello”, получим метод “HelloObject”.

jmethodID midHelloObject = NULL;

if (clsH != NULL) {

midHelloObject = env->GetStaticMethodID(clsH,"HelloObject","(LPeopleObject;)I");

} else {

printf("\nUnable to find the requested class\n");

}

Тут тоже всё довольно понятно, по сути, мы просто работаем со всеми методами, только через сторонний интерфейс.

Метод “GetStaticMethodID” ищет метод в классе “clsH” с именем “HelloObject” и нужной сигнатурой. О том как генерировать сигнатуру не ручками, скажем позже.

Теперь получим конструктор класса “PeopleObject”, для того, чтобы в него передать нужные параметры для создания объекта.

jmethodID midPeopleObjectConst = NULL;

if(clsP != NULL) {

midPeopleObjectConst = env->GetMethodID(clsP, "<init>", "(ILjava/lang/String;Ljava/lang/String;I)V");

} else {

printf("\nUnable to find the requested class\n");

}

Словом “<init>” мы показываем, что это будет именно конструктор.

Ну а теперь сделаем то, что задумали:

jobject jobjPeople = NULL;

if (midHelloObject!=NULL) {

if(clsP != NULL && midHelloObjectConst != NULL) {

char cName[255] = “Yuki”;

jstring name = env->NewStringUTF(cName);

char cTown[255] = “Tokyo”;

jstring town = env->NewStringUTF(cTown);

int cHeight = 169;

jint height = (jint) cHeight;

int cAge = 17;

jint age = (jint) cAge;

jobjPeople = env->NewObject(clsP, midPeopleObjectConst, height, name, town, age);

}

if(jobjPeople != NULL && midHelloObject!= NULL)

env->CallStaticIntMethod(clsH,midHelloObject,jobjPeople);

}

Что тут происходит? Да всё тоже самое, подготавливаются параметры для создания объекта “PeopleObject”. Далее создаётся объект класса “PeopleObject”, через его конструктор. А дальше вызывается метод из класса “Hello”, с передачей в функцию “HelloObject”, объекта класса “PeopleObject”. Вот и всё ;)

Ах, да, как же генерировать правильно сигнатуру, это можно делать и руками, но можно легко ошибиться, да и очень много работы. Генерировать сигнатуры можно при помощи утилиты “javap” входящая в состав JDK.

Например если запустить её для класса “Hello”:

javap -s -p Hello

то мы получим:

Compiled from "Hello.java"

public class Hello extends java.lang.Object{

public Hello ();

Signature: ()V

public static void main(java.lang.String[]);

Signature: ([Ljava/lang/String;)V

public static int HelloObject(PeopleObject);

Signature: (LPeopleObject;)I

}

Так же, скажу на последок, что если у вас 64-bit архитектура, то у вас возможно возникнут проблемы с компиляцией и запуском приложения. Дело в том что JDK 64-bit поставляет не совсем пригодные для таких операций библиотеки, поэтому лучше скачать JDK 32-bit и подключать либы оттуда ;)

Подробнее обо всё можно почитать по-прежнему на сайте Oracle, пока!