Привет всем!

Сегодня мы поговорим с вами о любви Java к C++. Связь наверно не очевидна, но она есть. И так поехали!

Порой в нашей жизни требуется использовать код, написанный на другом языке. Опишем несколько случаев такой необходимости:

Пускай у вас есть проверенный на 100% код на другом языке. Для того что бы его перевести на язык Java, потребуется немало времени, а полученный код нужно опять проверять и отлаживать. Особенно если это огромная библиотека или проект.

Или нам нужны низко уровненные операции к аппаратным средствам, в таком случае Java не очень подходит для таких операций.

Или если нам нужно множественное применение сложных операций с машиной, требующих очень оптимизированного кода, чего на Java опять не добьёшься (хотя этому можно поставить в противовес just-in-time компиляцию).

Если всё же мы столкнулись с одной из выше перечисленных проблем, то, скорее всего нам таки придётся научиться вызывать платформенно-ориентированный код прямиком из Java. Для этого в Java предусмотрены специальные способы работы с системными библиотеками, а JDK содержит инструменты, которые позволят нам справиться с этими проблемами.

Конечно, всё это не избавляет нас от полноценного решения, так как для каждой операционной системы потребуются библиотеки, ориентированные именно на эту среду, и хоть и Java очень хорошо контролируемый язык, то C++ в свою очередь неконтролируем, поэтому возможно, что методы из C++ выведут из строя виртуальную Java машину, либо вызовет сбой системы или что ещё.

Но выбор за вами, стоит ли это делать в ситуации, с которой вы столкнулись или нет ;)

Я не буду описывать кучу методов, кучу ситуаций, и тому подобное, это лучше всего расскажет специализированная книжка, а тут будет приведены примеры почти стандартных задач, вида Hello “damn” world! ☺

И так наша задача, допустим, на языке C/C++ есть функция, которая выполняет какие-то сумасшедше-полезные действия. И мы хотим её вызвать из Java (нам что лень её переписать на Java? ☺). Пускай это функция вывода printf (можно использовать и другие, но будем ориентироваться на C), которая приветствует передаваемую строку и какое-то число.

Для того что бы в Java показать что функция будет платформенно-ориентированной, есть ключевое слово “native”, которое предупреждает компилятор о внешнем определении данного метода. Саму функцию нужно описывать в классе.

public static native void welcome(String str, int value);

В данном случае функция объявлена как статическая, но это не обязательное условие. Теперь поместим её в класс.

class Hello {

public static native void welcome(String str, int value);

}

Такой класс откомпилировать вы сможете, но вылетит исключение, говорящее, что программа не может найти метод. Для того что бы метод был найден, его тело нужно создать (КЭП) в C/C++, причём не просто метод, а метод говорящий что его будут вызывать из Java. Но об этом попозже, просто знайте, что мы создадим разделяемую библиотеку для экспорта на C/C++, а далее её загрузим в Java. И тогда теперь программа найдёт этот метод, и сможет успешно выполняться.

Библиотеку удобнее всего подключить в статическом блоке, что бы при вызове функции, она уже была подгружена. И вот что получается.

class Hello {

public static native void welcome(String str, int value);

static {

System.loadLibrary("JAVALOVECHELLO");

}

}

Теперь настало время для создания C/C++ функции, имя которой должно СТРОГО (значит точно) соответствовать соглашениям об именовании:

Имя метода в C/C++ должно полностью соответствовать ПОЛНОМУ имени в Java программе. Например, если мы посетим программы в пакет com.intel, тогда полное имя Java метода com.intel.Hello.welcome. Теперь нужно заменить все точки на нижние подчёркивания (\_), и добавить префикс “Java\_”, тогда получим “Java\_com\_intel\_Hello\_welcome”, или если пакет стоит “default”, то “Java\_Hello\_welcome”. Так же стоит быть внимательными с различными Unicode-символами, в таком случае их нужно заменять на их 16-ричное представление, но об этом не будем тут. На самом деле и это ещё не все соглашения, но сейчас расскажем, как не делать этого ручками.

Для правильно описанных методов не стоит их создавать руками, для этого воспользуемся специальной утилитой “javah”, которая есть в JDK. Для начала откомпилируем наш Java файл, используя “javac”.

“javac Hello.java”

Теперь воспользуемся “javah”.

“javah Hello”

В результате выполнения этих операций, мы получим файл “Hello.h”

/\* DO NOT EDIT THIS FILE - it is machine generated \*/

#include <jni.h>

/\* Header for class Hello \*/

#ifndef \_Included\_Hello

#define \_Included\_Hello

#ifdef \_\_cplusplus

extern "C" {

#endif

/\*

\* Class: Hello

\* Method: welcome

\* Signature: ()V

\*/

JNIEXPORT void JNICALL Java\_Hello\_welcome

(JNIEnv \*, jclass, jstring, jint);

#ifdef \_\_cplusplus

}

#endif

#endif

Как видно, так как мы пишем на C++, то добавлена строчка extern "C", которая об этом сообщает. Имена JNIEXPORT и JNICALL описаны в “jni.h”, которые задают спецификаторы функций, экспортируемых из динамических библиотек .

В данном файле всё довольно понятно, всё как мы и предполагали (на самом деле не всё, но об этом позже).

Осталось для данного файла заголовка написать соответствующий файл исходного кода, где мы и опишем тело метода.

Приведём сразу, как же будет выглядеть файл исходного кода.

#include "Hello.h"

#include <stdio.h>

JNIEXPORT void JNICALL Java\_Hello\_welcome(JNIEnv\* env, jclass cl, jstring str, jint value)

{

const char \* ccstr;

ccstr = (\*env).GetStringUTFChars(str, NULL);

printf("Hello damn World, %s, %d!\n", ccstr, (int)value);

}

Во всех вызовах JNI-функций есть указатель env, который является 1-ым параметром любого платформенно-ориентированного метода. Параметр env указывается на таблицу функций , содержащую указатели функций. Как наверно можно догадаться jstring, jint это по сути те же string, int, но только в понимании JNI.

Функция GetStringUTFChars() возвращает указатель “const jbyte \*” на символы строки , представленные в формате UTF-8. Так же не помешает после использования строк освобождать память. Если же этого не делать, то это может повлиять на работу метода (память не безгранична). Для этих целей нужно использовать метод ReleaseStringUTFChars().

Очень подробное описание всех функций (API JNI) можно найти сайте Oracle, по этой ссылке - <http://docs.oracle.com/javase/6/docs/technotes/guides/jni/spec/jniTOC.html>.

Как компилировать динамическую библиотеку каждый выбирает сам, это можно сделать вручную, или можно предоставить всё MVS.

Нужные вам библиотеку, для успешной компиляции C/C++ программы лежат в JDK/include.

Ну и осталось написать вызов функции из Java программы.

class HelloTest {

public static void main(String[] args) {

Hello.welcome("I'm a string", 100500);

}

}

На этом закончим, хоть и было рассмотрено материала 7% от общего, но тут главное смысл, а дальше уже как по маслу, берёшь и намазываешь ;)