Swig programming introduction

출처 : 1. Swig documentation – www.swig.org
번역 및 추가 : 김성근 (danguria@gmail.com)
테스트 환경 : Ubunto, java1.6, g++, vim, swig1.3.40

☐ Swig Overview

SWIG는 Simplified Wrapper and Interface Generator의 약자로, 스크립트 언어와 C/C++로 구현된 프로그램 사이의 인터페이스를 해주는 툴입니다. 버전이 올라가면서, 자바와 같은 스크립트 언어가 아닌 경우도 지원해주고 있습니다. 본 문서에서는 자바와 C/C++사이에서 어떻게 Swig를 사용하여 인터페이스과정을 쉽게 구축하는지 알아보도록 하겠습니다.

1. What is the Swig?

Swig는 C/C++과 다른 언어 사이의 인터페이스를 쉽게 구축 할 수 있도록 도와 주는 소프트웨어 개발 툴입니다. 각 언어에서 제공하는 다른 언어 사이의 인터페이스를 구축하는 일은 각 언어마다 다릅니다. 매번 이것을 이해 하고 사용하는 것은 힘든 일입니다. Swig는 이런 일을 대신 해줌으로써, 개발자는 중요한 일에 더욱 집중 할 수 있도록 도와주는 역할을 담당하고 있습니다.

2. Why, when use the Swig?

Swig를 사용해야 하는 경우는 다음의 경우로 생각해 볼 수 있습니다.

A. 기존의 C프로그램과 인터페이스 하는 프로그램을 만들 때
B. 빠르게 응용프로그램의 prototype을 만들고자 할 때
C. 상호 작용하는 디버깅을 하려고 할 때
D. 그래픽 UI를 만들고자 할 때
E. 스크립트 언어를 이용하여 C/C++로 구현된 라이브러리를 테스트해보고자 할 때.
F. 스크립트 언어에서 사용하려고 하는 성능에 민감한 C/C++모듈을 만들고자 할 때

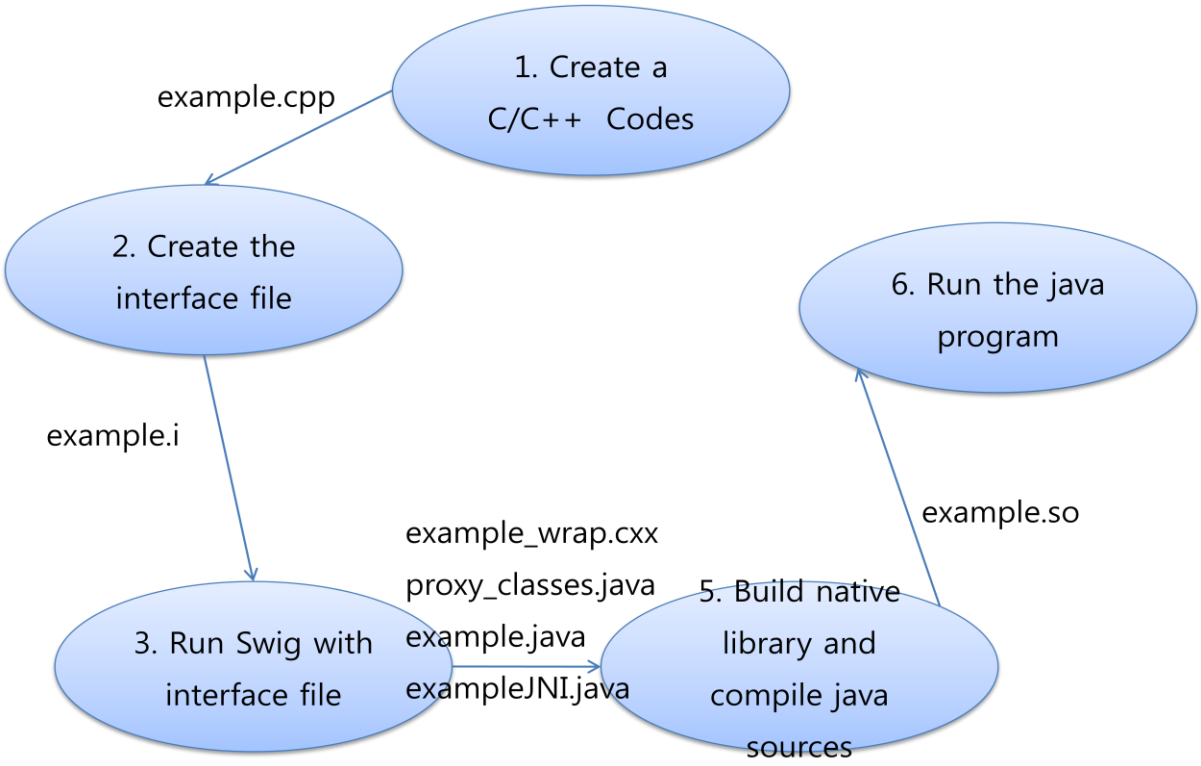
Swig를 사용할 때의 장점은 다음과 같습니다.

A. C/C++로 라이브러리를 잘 만들어 놓으면 쉽게 다른 언어에서 사용할 수 있습니다.
B. 좋은 성능을 요구하는 경우(수치계산, 데이터 처리, 그래픽 연산)에 C/C++작성해야 합니다. 이때 라이브러리는 C/C++로 만들고 스크립트 언어에서 이를 쉽게 사용할 수 있습니다.
C. 시스템에 의존적인 프로그래밍을 할 때 유용합니다.
D. 많은 사람이 프로그램을 사용할 수 있습니다.

3. How to use the swig?

자바에서 Swig를 이용하기 위해서는 필요한 파일은 사용하고자 하는 C/C++ 소스 파일과 interface파일이 필요합니다. 그 결과로 JNI관련 java파일과 cpp파일이 나오게 되며, C/C++소스 파일에 있는 클래스를 사용하기 위한 proxy class들이 생성됩니다.
다음부터 Swig를 사용하여 C/C++로 작성된 factorial 함수를 자바에서 사용하는 예제를 작성해 보도록 하겠습니다. 전체적인 흐름을 설명하기 위한 것이므로 자세한 설명은 다음 섹션에서 진행하도록 하겠습니다.

A. Steps for Running swig with simple example



<Steps in Writing and Running the example Program>

B. Create a C/C++ codes

Factorial 함수를 구현한 C++파일을 작성합니다.

```
1 double My_variable = 3.0;
2
3 /* compute factorial of n */
4 int fact(int n){
5       if (n <= 1) return 1;
6       else return n * fact(n-1);
7 }
```

C. Create the interface file

이전 단계에서 만든 C/C++파일을 참고 하여 interface파일을 만듭니다.

```
1 /* File : example.i */
2 %module example
3 %{
4       /* put headers and other declarations here */
5       extern double My_variable;
6       extern int      fact(int);
7
8       %}
9
10 extern double My_variable;
11 extern int      fact(int);
```

D. Run swig with interface file

이전 단계에서 만든 interface file을 가지고 swig명령어를 실행합니다.

```
swig  –c++  -java  example.i
```

명령어에서 –c++옵션은 사용되는 C/C++코드가 C++로 작성되었음을 Swig에게 알리는 것입니다. -java옵션은 java에서 C/C++코드를 사용할 것이므로 적절한 소스 코드를 생성해줄 것을 알리는 옵션입니다. 이 명령의 결과로 생기는 파일은 example.java exampleJNI.java파일과 example_wrapp.cxx파일이 생성됩니다.

- example.java – example.i에서 %module example이라고 한 것에 대한 결과. 전역변수에 대한 api가 노출 되어 있습니다.
- exampleJNI.java – 모든 C/C++파일에서 작성한 모든 함수의 native method들이 정의 되어 있습니다.
- example_wrapp.cxx – exampleJNI.java에 있는 native method가 구현된 C/C++소스

E. Build native library compile java sources

첫 번째 단계에서 작성한 C/C++파일과 이전단계에서 작성한 example_wrap.cpp파일을 이용하여 공유 라이브러리를 만듭니다.

```
g++ -c -fPIC -I$JAVA_HOME/include -I$JAVA_HOME/include/linux example.cpp example_wrap.cxx
g++ -shared -o libExample.so example.o example_wrap.o
```

위의 명령어는 동적 라이브러리를 만드는 명령어 입니다. 자세한 사항은 저의 "공유 라이브러리 만들기" 스프링노트를 참고 하세요.

F. Run the java program

이전 단계에서 만들어진 라이브러리를 테스트 하기 위해서 "ExamTest.java"파일을 작성하여 컴파일 하고 실행 합니다.

```
1 public class ExamTest{
2       static {
3             System.loadLibrary("Example");
4       }
5       public static void main(String args[]){
6             System.out.println(example.fact(3));
7       }
8 }
```

☐ Swig basic with Java

1. Overview

순수한 자바의 노력은 굉장히 좋은 컨셉을 가지고 있습니다. 하지만 경우에 따라서는 기존에 존재 하는 코드를 사용해야 하는 경우가 있습니다. 자바는 이를 해결 하기 위해서 JNI(Java Native Interface)라는 툴을 제공함으로써 이를 해결 하고 있습니다. 하지만 JNI는 사용하기에 번거로운 점이 있습니다. C/C++코드를 Wrapping하기 위해서 javah툴을 이용하여 native method의 prototype을 생성하고 직접 JNI함수를 구현해야 합니다. 하지만 함수들이 많을 수록, Class의 복잡성이 증가 할수록 JNI함수를 구현하는 것은 여간 힘든 일이 아닙니다. 중요하지 않은 일에 코드 작성과 디버깅하는 데에 많은 시간을 들여야 하는 데에 반해, Swig를 이용하면 이런 일들을 자동으로 해결 할 수 있습니다. 앞으로 자바에서의 Swig를 사용하는 데에 기본적인 특징들을 알아보도록 하겠습니다.

2. Modules, Packages, generated Java classes

A. Modules

%module directive는 자동으로 생성될 java파일들 중 사용자가 주로 사용할게 될 class의 이름을 지정해 주는 역할을 합니다. 모듈 클래스에는 전역 변수, 전역 함수, enum과 같은 값에 대한 접근을 하게 할 함수들이 들어있습니다. 앞의 Swig Overview 섹션에서 보았던 예제에서 fact함수와 My_variable 변수는 전역 변수이기 때문에 example 클래스에서 해당 변수와 함수를 접근 할 수 있는 method들이 정의 되어 있는 것을 알 수 있습니다.

B. Packages

자동으로 생성되어 지는 파일은 기본적으로 interface파일이 있는 곳에 생성됩니다. 일반적으로 자바는 패키지를 만들어 각각의 소스코드를 관리 하기 때문에 swig명령어 옵션에서 적절한 처리를 하여 패키지 관리를 편리하게 할 수 있습니다.

```
Swig –java –package org.ssm.danguria –outdir org/ssm/danguria example.i
```

기본적인 swig에서는 –outdir 옵션으로 생성될 파일들이 저장될 경로를 지정할 수 있습니다. 자바는 추가적으로 -package옵션을 줌으로써, 생성되는 자바 파일 안에 "package 패키지경로" 코드를 자동으로 생성하여 줍니다. 패키지 폴더는 자동으로 생성해 주지 않기 때문에 직접 폴더를 생성해 주어야 하는 것을 잊지 마세요.

C. Generated java classes

이전까지의 예제에서는 전역함수와 전역 변수를 만들었습니다. 만일 클래스, 구조체, 공용체와 같은 자료 구조를 Swig에서는 어떻게 Wrapping할까요? 바로 Proxy class로 Wrapping합니다. 자동으로 생성되는 java파일에 C/C++에서 만들어진 클래스에 해당하는 java클래스들이 생성됩니다. 내부를 보면 C/C++의 클래스에서 정의한 함수들을 호출 하고 있습니다. 바로 C/C++함수를 호출 할 수 없기 때문에 native method가 xxxJNI.java파일에 있고(xxx는 %module 로 지정한 모듈 명), 이 native method를 호출함으로써 C/C++의 클래스기능을 수행 하게 됩니다.

3. Global functions and variables

전역 함수와 변수에 대한 Wrapping을 자세히 알아보도록 하겠습니다. 전역함수와 변수를 Wrapping하기 위해서 interface파일에 전역변수와 함수의 사용을 다음과 같이 알려야 합니다.

```
 1 /* File : example.i */
 2 %module example
 3 %{
 4       /* put headers and other declarations here */
 5       extern double My_variable;
 6       extern int      fact(int);
 7
 8       %}
 9
10 extern double My_variable;
11 extern int      fact(int);
```

A. %{ ..... }% directive

%{...}%directive는 swig가 동작할 때 무시하는 코드를 모아 두는 곳입니다. swig명령어를 통해 자동으로 파일들이 생성될 때 일부 소스 코드를 직접 넣어 주어야 할 경우가 있습니다. 이 때 %{}% directive를 사용하면 됩니다. 우리가 이전 섹션에서 본 예제에서는 example_wrap.cxx파일을 유심히 봐야 할 필요가 있습니다. Example_wrap.cxx에서는 example.cpp에서 작성된 My_variable과 fact함수를 사용해야 합니다. 이 사용을 알리는 extern 코드를 넣어 주는 일은 개발자의 몫입니다. 그 외에도 적절한 해더파일을 포함하기 위해서도 이 directive가 사용됩니다.

B. Notify swig to extern variables and functions

위 코드의 10, 11째 줄을 보면 extern으로 외부 파일에 있는 변수와 함수를 알리고 있습니다. 이 코드에 의해서 swig는 자동으로 적절한 파일과 코드들을 생성해 주게 됩니다.

C/C++의 전역 함수와 변수들은 module명.java에 있는 클래스에 Wrapping 됩니다. C/C++의 전역함수는 자바에서 static함수로 정의 되고, 전역 변수 역시 static함수로 만들어진 set, get 함수가 만들어 집니다.

```
 10 public class example {
 11    public static void setMy_variable(double value) {
 12       exampleJNI.My_variable_set(value);
 13    }
 14
 15    public static double getMy_variable() {
 16       return exampleJNI.My_variable_get();
 17    }
 18
 19    public static int fact(int arg0) {
 20       return exampleJNI.fact(arg0);
 21    }
 22
 23 }
```

이를 사용하기 위해서는 다음과 같은 코드를 작성하면 됩니다.

```
System.out.println(example.fact(3));
```

4. Constants

C/C++에서의 상수는 자바에서 static final 변수로 Wrapping됩니다. Java 코드를 생성하기 위해서 interface파일에 다음과 같은 코드를 작성해야 합니다.

```
#define PI 3.14
#define VERSION "1.0"
%constant int FOO = 43;
%constant const char *path = "/usr/local";
```

위의 interface파일에 의해서 생성되는 java코드는 다음과 같습니다.

```
public interface exampleConstants{
        public final static double PI = exampleJNI.PI_get();
        public final static String VERSION = exampleJNI.VERSION_get();
        public final static int FOO = exampleJNI.FOO_get();
        public final static String path = exampleJNI.path_get();
}
```

5. Enumerations

A. Anonymous enums

다음과 같은 enum을 Wrapping하는 법을 알아 보겠습니다.

```
Enum { SSM, DAEGU, DANGURIA };
```

위의 코드를 Wrapping하면 다음과 같은 자바 코드가 생성됩니다.

```
public interface exampleConstants{
        public final static int SSM = exampleJNI. SSM _get();
        public final static int DAEGU = exampleJNI. DAEGU _get();
        public final static int DANGURIA = exampleJNI. DANGURIA _get();
}
```

생성된 자바 코드는 상수를 Wrapping한 결과와 비슷한 것을 할 수 있습니다. C/C++의 enum은 Anonymouse enum외에도 typefafe enum과 같은 방식으로 Wrapping할 수 있습니다. 그 밖에 다양한 방법이 있는데 자세한 것은 www.swig.org를 참고하시기 바랍니다.

6. Pointers

Swig에서는 pointer를 완벽하게 Wrapping해주고 있습니다. C/C++과 자바 사이의 타입문제를 완벽하게 해결해주고 있습니다. 아래와 같은 예를 통해 pointer를 Wrapping하는 법을 알아 보도록 하겠습니다.

```
%module example

FILE *fopen(const char *filename, const char *mode);
int fputs(const char*, FILE *);
int fclose(FILE*);
```

Swig가 위의 코드를 Wrapping하게 되면 자바에서는 위 함수들을 아무 문제 없이 사용할 수 있습니다. 아래는 위 함수를 사용하는 간단한 예제입니다.

```
SWIGTYPE_p_FILE f = example.fopen("danguria", "w");
Example.fputs("Hello danguria₩n", f);
Example.fclose(f);
```

다른 언어들은 String형태로 저장되지만 C/C++포인터는 자바에서 long타입 변수에 저장 됩니다. 다른 스크립트 언어들은 컴파일러나 인터프리터가 static type check기능이 없기 때문에 swig runtime에서 제공하는 dynamic type checker가 사용됩니다. 하지만 자바는 자바 클래스로

Wrapping되기 때문에 type checking에 대해서 신경 쓰지 않아도 됩니다. 위의 예제에서 FILE* 포인터는 SWIGTYPE_p_FILE이라는 클래스로 Wrapping됩니다. 자바에서 type wrapper object가 일단 획득되면 이 오브젝트를 매개변수로 받는 함수에 인자로 줄 수 있고, C/C++에서도 자유롭게 사용할 수 있습니다. 이런 오브젝트에 값을 변경하는 등의 행위는 자칫 오동작을 일으킬 수 있습니다. 형 변환이나 기타 자세한 사항은 www.swig.org를 통해 확인하기 바랍니다.

7. Structures

C/C++의 구조체는 자바에서 해당 변수들에 대한 setter, getter들이 포함된 클래스로 Wrapping됩니다.

```
struct Vector{
        double x, y, z;
};
```

위의 코드와 같은 간단한 구조체는 자바에서 아래와 같이 사용될 수 있습니다.

```
Vector v = new Vector();
v.setX(2.3);
v.setY(3.5);
double x = v.getX();
double y = v.getY();
```

A. %immutable

상수화 된 변수들은 자동으로 setter함수가 만들어 지지 않습니다. 하지만 상수화 되지 않은 변수를 강제로 read-only 로 만들기 위해서 해당 변수 위아래에 %immutable directive를 쓰면 됩니다.

```
struct Danguria{
        %immutable
        int x;              // read-only members
        char *name;         // read-only members
        %immutable
};
```

A. Arrays

배열은 pointer와 같은 형식으로 자바로 Wrapping됩니다.

```
struct Danguria{
        int   x[16];
};
```

위의 구조체는 자바에서 다음과 같은 방식으로 이용할 수 있습니다.

```
Danguria d = new Danguria();
SWIGTYPE_p_int   x = d.getX();
Danguria e = new Danguria();
e.setX(x);
```

하지만 위와 같은 사용은 배열의 범위 체크를 해 주시 않기 때문에 조심해서 사용해야 합니다. 이를 해결하기 위한 방안은 www.swig.org의 "Wrapping C Arrays with Java arrays"와 "Unbound C Arrays" 섹션을 읽어 보시기 바랍니다.

8. C++ classes

C/C++의 클래스는 구조체와 마찬가지로 자바의 클래스로 Wrapping됩니다.

```
class List{
public:
        List();
        ~List();
        int search(char *item);
        void insert(char *item);
        char* get(int n);
        int length;
};
```

위와 같은 C/C++ 코드는 자바에서 다음과 같이 사용될 수 있습니다.

```
List list = new List();
l.insert("hi");
String item = l.get(0);
int length = l.getLength();
```