Handing errors & Invocation API

# Handing errors

Các hàm native code được xem là làm mất an toàn cho chương trình java vì “C runtime system” không có cơ chế xử lý ngoại lệ như: truy cập ngoài vùng mảng, truy cập địa chỉ không hợp lệ… Do đó một vấn đề quan trọng trong lập trình native là phải kiểm tra các lỗi có thể phát sinh và quăng –“throw” về cho chương trình java bắt.

## Throw một ngoại lệ cho chương trình java

JNI cung cấp hai hàm để throw lỗi:

jint Throw(JNIEnv \*env, jthrowable obj);

**Parameters**

env: the JNIEnv interface pointer.

obj: a java.lang.Throwable object.

**Return Values**: trả về 0 nếu thành công, ngược lại trả về số âm.

Để dùng hàm này thì ta phải tạo một đối tượng có interface là Throwable (cũng lấy từ JNI). Sau đó mới gọi hàm Throw. Còn hàm ThrowNew thì ta chỉ cần truyền vào class cài đặt interface Throwable không cần phải tạo đối tượng.

jint ThrowNew(JNIEnv \*env, jclass clazz, const char \*message);

**Parameters**

env: the JNIEnv interface pointer.

clazz: một lớp cài đặt của java.lang.Throwable.

message: chuỗi thông báo dùng để dựng đối tượng Throwable kiểu clazz được truyền vô.

**Return Values**: trả về 0 nếu thành công, ngược lại trả về số âm.

Lưu ý là khi Throw một ngoại lệ thì native code vẫn tiếp tục chạy. Khi kết thúc hàm native thì JVM mới phát sinh ra ngoại lệ. Nêu nếu muốn native code không chạy tiếp khi có ngoại lệ thì cần gọi return ngay sau lệnh Throw.

## Nhận ngoại lệ từ chương trình Java

Khi gọi thực thi một method java từ navite code. Chúng ta sẽ cần biết method chạy thế nào, có bị ngoại lệ gì không. JNI có hỗ trợ cho chúng ta làm việc này.

JNI cũng cấp hai hàm kiểm tra xem có ngoại lệ xảy ra không và hàm lấy về đối tượng ngoại lệ:

jboolean ExceptionCheck(JNIEnv \*env);

**Return Values** trả về JNI\_TRUE nếu có ngoại lệ sảy ra, hoặc JNI\_FALSE nếu ngược lại.

jthrowable ExceptionOccurred(JNIEnv \*env);

**Return Values** trả về đối tượng có interface Throwable nếu có ngoại lệ sảy ra, hoặc NULL nếu ngược lại.

# Sử dụng Invocation API

Giả sử bạn có một chương trình C/C++ và muốn gọi Java code. Chính lúc này invocation API của JVM sẽ giúp bạn. Cho phép chúng ta tạo một Java Virtual Machine trong chương trình C/C++.

JNI cung cấp hai hàm để tạo và hủy máy ảo java.

## Tạo JVM

jint JNI\_CreateJavaVM(JavaVM\*\* p\_jvm, void\*\* p\_env, JavaVMInitArgs\* vm\_args)

Khởi tạo một JVM. Hàm trả về 0 nếu thành công, JNI\_ERR khi ngược lại.

| **Parameters:** | p\_jvm | Con trỏ có kiểu JavaVM, dùng biến này để thao tác lên JVM tạo ra. |
| --- | --- | --- |
|  | p\_env | Con trỏ kiểu JNIEnv, dùng biến này để thực hiện các lệnh JNI (như gọi chạy hàm main chẳng hạn). |
|  | vm\_args | Biến kiểu JavaVMInitArgs làm tham sô truyền vào cho JVM. |

Mô tả của cấu trúc tham số JavaVMInitArgs

typedef struct [JavaVMOption](http://xdprof.sourceforge.net/doxygen/structJavaVMOption.html) {

char \*optionString;

void \*extraInfo;

} [JavaVMOption](http://xdprof.sourceforge.net/doxygen/structJavaVMOption.html);

typedef struct [JavaVMInitArgs](http://xdprof.sourceforge.net/doxygen/structJavaVMInitArgs.html) {

jint version; // phiên bản của jni có JNI\_VERSION\_1\_2 hoặc JNI\_VERSION\_1\_1

jint nOptions;

[JavaVMOption](http://xdprof.sourceforge.net/doxygen/structJavaVMOption.html) \*options;

jboolean ignoreUnrecognized;

} [JavaVMInitArgs](http://xdprof.sourceforge.net/doxygen/structJavaVMInitArgs.html);

Một tham số quan trọng cần truyền vào khi tạo JVM là class\_path. Để thêm vị trí hiện tại vào class\_path ta tạo option sau:

JavaVMOption options[1];

options[0].optionString = "-Djava.class.path=.";

Sau đó biến options đẽ được gán vào trương option của struct [JavaVMInitArgs](http://xdprof.sourceforge.net/doxygen/structJavaVMInitArgs.html)

## Hủy JVM

Để hủy một JVM đã tạo ta dùng hàm:

jint DestroyJavaVM(JavaVM\* jvm)

Returns: trả về 0 nếu thành công, một số âm nếu thất bại. Hàm này cần phải được gọi thông qua một con trỏ JavaVM được truyền vô lúc tạo JVM.