

NDK Integration (JNI)

Lecture 6

Android Native Development Kit

1 April 2014

Array Operations

NIO Operations

Accessing Fields

Calling Methods

Handling Exceptions

Local & Global References

Threads

Standard JNI vs. Android JNI

Keywords

Array Operations

NIO Operations

Accessing Fields

Calling Methods

Handling Exceptions

Local & Global References

Threads

Standard JNI vs. Android JNI

Keywords

- ▶ Java arrays - reference type in JNI
- ▶ JNI treats primitive arrays and object arrays differently
- ▶ Primitive arrays contain primitives
- ▶ Object arrays contain class instances or other arrays
 - ▶ `Object[]` and `int[][]` are object arrays
- ▶ `jarray` and subtypes (`jintArray`, `jobjectArray`)

- ▶ `New<Type>Array` where `Type` is `Int`, `Char`, `Boolean`, etc.

```
jintArray javaArray = env->NewIntArray(10);
```

- ▶ In case of memory overflow
 - ▶ Function returns `NULL`
 - ▶ Exception is thrown in the VM
 - ▶ Native code should be stopped

- ▶ Copy Java array into C array or obtain a direct pointer to the array elements
- ▶ Copy Java array into C array
 - ▶ Get<Type>ArrayRegion

```
jint nativeArray[10];
env->GetIntArrayRegion(javaArray, 0, 10, nativeArray);
```

- ▶ Make changes on the array elements
- ▶ Copy C array back into Java array
 - ▶ Set<Type>ArrayRegion
- ▶ Performance problem for big array size

```
env->SetIntArrayRegion(javaArray, 0, 10, nativeArray);
```

- ▶ Obtain a direct pointer to the array elements when possible
- ▶ `Get<Type>ArrayElements`

```

jint* nativeDirectArray;
jboolean isCopy;
nativeDirectArray = env->GetIntArrayElements(
    javaArray, &isCopy);

```

- ▶ `isCopy` - the C array points to a copy or a pinned array in heap
- ▶ Returns NULL if operation fails

- ▶ Release array returned by `Get<Type>ArrayElements`
- ▶ `Release<Type>ArrayElements`

```
env->ReleaseIntArrayElements(javaArray,
nativeDirectArray, 0);
```

- ▶ Last parameter - release mode
 - ▶ 0 - copy back content, free native array
 - ▶ `JNI_COMMIT` - copy back content, do not free native array (update Java array)
 - ▶ `JNI_ABORT` - do not copy back content, free native array
- ▶ `GetArrayLength`
- ▶ `Get/ReleasePrimitiveArrayCritical`

- ▶ Create new object array

- ▶ NewObjectArray

```
objectArray arr = env->NewObjectArray(size,
    javaClass, NULL);
```

- ▶ Params: length, class and initialization value

- ▶ Obtain an element from an object array

- ▶ GetObjectArrayElement

- ▶ Cannot obtain all object elements

```
jstring js = (jstring)env->
    GetObjectArrayElement(arr, i);
```

- ▶ Update an element in an object array

- ▶ SetObjectArrayElement

```
env->SetObjectArrayElement(arr, i, js);
```

Array Operations

NIO Operations

Accessing Fields

Calling Methods

Handling Exceptions

Local & Global References

Threads

Standard JNI vs. Android JNI

Keywords

- ▶ Native I/O - buffer management, scalable network and file I/O
- ▶ Better performance - deliver data between native and Java app
- ▶ Create a direct byte buffer to be used in the Java app

- ▶ `NewDirectByteBuffer`

```
unsigned char* buffer = (unsigned char*) malloc(1024);
jobject directBuffer;
directBuffer = env->NewDirectByteBuffer(buffer, 1024);
```

- ▶ Based on a native byte array
- ▶ Obtain native byte array from Java byte buffer

- ▶ `GetDirectBufferAddress`

```
unsigned char* buffer;
buffer = (unsigned char*) env->GetDirectBufferAddress
(directBuffer);
```

- ▶ The direct byte buffer can also be created in the Java app

Array Operations

NIO Operations

Accessing Fields

Calling Methods

Handling Exceptions

Local & Global References

Threads

Standard JNI vs. Android JNI

Keywords

- ▶ In Java: static fields and instance fields
- ▶ Each instance has its own copy of the instance fields

```
private String instanceField = "Instance Field";
```

- ▶ All instances share the same static fields

```
private static String staticField = "Static Field";
```

- ▶ JNI functions for both types of fields

- ▶ Obtain class object from instance

- ▶ GetObjectClass

```
jclass cl = env->GetObjectClass(instance);
```

- ▶ Obtain field ID of an instance field

- ▶ GetFieldID

```
jfieldID instanceFieldId;
instanceFieldId = env->GetFieldID(cl,
    "instanceField", "Ljava/lang/String;");
```

- ▶ Last parameter - field descriptor

- ▶ Obtain field ID of static field

- ▶ GetStaticFieldID

```
jfieldID staticFieldId;
staticFieldId = env->GetStaticFieldID(cl,
    "staticField", "Ljava/lang/String;");
```

- ▶ Obtain an instance field

- ▶ Get<Type>Field

```

jstring instanceField;
instanceField = env->GetObjectField(instance,
instanceFieldId);
    
```

- ▶ Obtain a static field

- ▶ GetStatic<Type>Field

```

jstring staticField;
staticField = env->GetStaticObjectField(cl,
staticFieldId);
    
```

- ▶ Type = Object, Primitive type

- ▶ Return NULL in case of memory overflow

- ▶ Performance overhead

- ▶ Recommended to pass parameters to native methods

Array Operations

NIO Operations

Accessing Fields

Calling Methods

Handling Exceptions

Local & Global References

Threads

Standard JNI vs. Android JNI

Keywords

- ▶ In Java: instance and static methods

- ▶ Instance method

```
private String instanceMethod() {
    return "Instance Method";
}
```

- ▶ Static method

```
private static String staticMethod() {
    return "Static Method";
}
```

- ▶ JNI functions to access both types

- ▶ Obtain method ID of an instance method

- ▶ GetMethodID

```
jmethodID instanceMethodId;
instanceMethodId = env->GetMethodID(c1,
"instanceMethod", "()Ljava/lang/String;");
```

- ▶ Last parameter - method descriptor (signature)

- ▶ Obtain method ID of a static method

- ▶ GetStaticMethodID

```
jmethodID staticMethodId;
staticMethodId = env->GetStaticMethodID(c1,
"staticMethod", "()Ljava/lang/String;");
```

- ▶ Call instance method

- ▶ Call<Type>Method

```

jstring instanceMethodResult;
instanceMethodResult = env->CallObjectMethod
(instance, instanceMethodId);
    
```

- ▶ Call static method

- ▶ CallStatic<Type>Field

```

jstring staticMethodResult;
staticMethodResult = env->CallStaticObjectMethod
(cl, staticMethodId);
    
```

- ▶ Type = Void, Object, Primitive type
- ▶ Specify method arguments after method ID
- ▶ Return NULL in case of memory overflow
- ▶ Performance overhead
 - ▶ Minimize transitions between Java and native code

Java Type	Signature
Boolean	Z
Byte	B
Char	C
Short	S
Int	I
Long	J
Float	F
Double	D
fully-qualified-class	Lfully-qualified-class
type[]	[type
method type	(arg-type)ret-type

- ▶ Use javap to obtain descriptors associated to a Java class

Array Operations

NIO Operations

Accessing Fields

Calling Methods

Handling Exceptions

Local & Global References

Threads

Standard JNI vs. Android JNI

Keywords

- ▶ Handling exceptions is important in Java
- ▶ VM catches exception, clears exception and executes handling block
- ▶ In native code developers must implement exception handling flow
- ▶ Catch an exception generated while calling a Java method
 - ▶ ExceptionOccurred

```

env->CallVoidMethod(instance, methodID);
jthrowable ex = env->ExceptionOccurred();
if (ex != NULL) {
    env->ExceptionClear();
    /* Handle exception here */
}

```

- ▶ Native code can throw Java exceptions
- ▶ First obtain exception class
- ▶ Throw exception

- ▶ **ThrowNew**

```

jclass cl = env->FindClass
("java/lang/NullPointerException");
if (cl != NULL) {
    env->ThrowNew(cl, "Message");
}
    
```

- ▶ Does not automatically stop native method and transfer control to exception handler
 - ▶ Should free resources and return

Array Operations

NIO Operations

Accessing Fields

Calling Methods

Handling Exceptions

Local & Global References

Threads

Standard JNI vs. Android JNI

Keywords

- ▶ The VM tracks object references and garbage collects the ones that are not referenced
- ▶ JNI allows native code to manage object references and lifetimes
- ▶ 3 types of references: local, global and weak global

- ▶ Most JNI functions return local references
- ▶ Cannot be cached and reused in subsequent invocations
- ▶ Lifetime limited to the native method - freed when method returns
- ▶ Minimum 16 local references for the native code in the VM
- ▶ Free local references while making memory-intensive operations
- ▶ Manually free local reference
 - ▶ DeleteLocalRef

```

jclass cl = env->FindClass("java/lang/String");
env->DeleteLocalRef(cl);

```

- ▶ Valid during subsequent invocations of the native method
- ▶ Until explicitly freed
- ▶ Create new global reference

- ▶ `NewGlobalRef`

```

jclass localCl = env->FindClass("java/lang/String");
jclass globalCl = env->NewGlobalRef(localCl);
env->DeleteLocalRef(localCl);

```

- ▶ Delete global reference when no longer used

- ▶ `DeleteGlobalRef`

```

env->DeleteGlobalRef(globalCl);

```

- ▶ Can be used by other native methods or native threads

- ▶ Valid during subsequent invocations of the native method
- ▶ The object can be garbage collected
- ▶ Create new weak global reference
 - ▶ `NewWeakGlobalRef`

```
jclass weakGlobalCl;  
weakGlobalCl = env->NewWeakGlobalRef(localCl);
```

- ▶ Verify if the reference is still pointing to an instance

- ▶ IsSameObject

```
if (env->IsSameObject(weakGlobalCl, NULL) == JNI_-
FALSE) {
    /* Object is still live */
} else {
    /* Object is garbage collected */
}
```

- ▶ Delete weak global reference

- ▶ DeleteWeakGlobalRef

```
env->DeleteWeakGlobalRef(weakGlobalCl);
```

Array Operations

NIO Operations

Accessing Fields

Calling Methods

Handling Exceptions

Local & Global References

Threads

Standard JNI vs. Android JNI

Keywords

- ▶ Multithreaded environment
- ▶ Threads vs. references
 - ▶ Local references valid only in the thread context executing the native method
 - ▶ Local references cannot be shared between multiple threads
 - ▶ Global references can be shared between multiple threads
- ▶ Threads vs. JNIEnv
 - ▶ Interface pointer valid only in the thread executing the native method
 - ▶ Cannot be cached and used by other threads

- ▶ Use threads for running tasks in parallel
- ▶ Linux threads, scheduled by the kernel
- ▶ Started from managed code with `Thread.start`
- ▶ Can also be started with `pthread_create`
- ▶ Native threads not known by the VM until they are attached
 - ▶ No `JNIEnv`
 - ▶ Cannot make JNI calls

- ▶ First attach native thread to the VM

- ▶ AttachCurrentThread

```
JavaVM* cachedJvm;
JNIEnv* env;
cachedJvm->AttachCurrentThread(&env, NULL);
```

- ▶ Obtain a JNIEnv interface pointer for the current thread
 - ▶ java.lang.Thread object added to main ThreadGroup
 - ▶ Last argument: JavaVMAttachArgs structure - can specify other thread group

- ▶ After communication with the Java app, detach from VM

- ▶ DetachCurrentThread

```
cachedJvm->DetachCurrentThread();
```

- ▶ Synchronization using monitors based on Java objects
- ▶ Only one thread can hold a monitor at a time
- ▶ Acquire a monitor
 - ▶ `MonitorEnter`

```
env->MonitorEnter(obj);
```
 - ▶ `obj` is a Java object
 - ▶ If another thread owns the monitor, waits until it's released
 - ▶ If no other thread owns the monitor, becomes owner, entry counter = 1
 - ▶ If the current thread owns the monitor, increments the entry counter

▶ Release monitor

▶ `MonitorExit`

```
env->MonitorExit(obj);
```

- ▶ The current thread must be the owner of the monitor
- ▶ Entry counter is decremented
- ▶ When counter = 0, the current thread releases the monitor

Array Operations

NIO Operations

Accessing Fields

Calling Methods

Handling Exceptions

Local & Global References

Threads

Standard JNI vs. Android JNI

Keywords

- ▶ All JNI 1.6 features are supported by Android JNI
- ▶ Exception: DefineClass not implemented
 - ▶ No Java bytecodes or class files in Android
 - ▶ Not useful
- ▶ JNI does not include proper error checks
- ▶ Android includes CheckJNI mode
 - ▶ Performs series of checks before the actual JNI function is called
 - ▶ Enable CheckJNI from `adb shell`
 - ▶ Enabled by default on emulator

- ▶ Attempt to allocate negative-sized arrays
- ▶ Passing a bad pointer(jobject, jclass, jarray, jstring) to a JNI call
- ▶ Passing a NULL pointer to a JNI call when argument should not be NULL
- ▶ Passing a class name not correctly specified to a JNI call
- ▶ Making a JNI call in a critical region
- ▶ Passing invalid arguments to NewDirectByteBuffer
- ▶ Making a JNI call while an exception is pending

- ▶ Using JNIEnv* in the wrong thread
- ▶ Using NULL, wrong type fieldID, static/instance mismatch
- ▶ Using invalid jmethodID, incorrect return type, static/instance mismatch, invalid instance/class
- ▶ Using DeleteGlobal/LocalRef on the wrong reference
- ▶ Passing a bad release mode, other than 0, JNI_COMMIT, JNI_ABORT
- ▶ Returning incompatible type from native method
- ▶ Passing invalid UTF-8 sequence to a JNI call

- ▶ <http://www.soi.city.ac.uk/~kloukin/IN2P3/material/jni.pdf>
- ▶ <http://docs.oracle.com/javase/6/docs/technotes/guides/jni/spec/jniTOC.html>
- ▶ <http://download.java.net/jdk8/docs/technotes/guides/jni/spec/functions.html>
- ▶ <http://developer.android.com/training/articles/perf-jni.html>
- ▶ Onur Cinar, Pro Android C++ with the NDK, Chapter 3
- ▶ Sylvain Ratabouil, Android NDK, Beginner's Guide, Chapter 3

Array Operations

NIO Operations

Accessing Fields

Calling Methods

Handling Exceptions

Local & Global References

Threads

Standard JNI vs. Android JNI

Keywords

- ▶ Primitive array
- ▶ Object array
- ▶ Direct pointer
- ▶ Native I/O
- ▶ Static/instance fields
- ▶ Static/instance methods
- ▶ Field/method ID
- ▶ Field/method descriptor
- ▶ Catch/throw exception
- ▶ Local/global/weak global references
- ▶ Native threads
- ▶ Monitor
- ▶ CheckJNI