Integração entre linguagens. Java -> C++

27/03/2015 Guilherme Lucas Equipe

A comunicação entre Java Android e o c++ não acontece de maneira direta, é necessária uma “ponte” chamada Jni.

O JNI

A Java Native Interface, é um framework que fornece recursos, com a Máquina Virtual Java, para utilização de recursos específicos em determinado Sistema Operacional, ou biblioteca compartilhada e vice-versa. Pelo motivo o qual os Sistemas Virtuais não tem acesso direito à funções do hardware e etc. Com a JNI esta barreira é quebrada. É óbvio que a portabilidade do Java desaparecerão, porém recursos como impressoras fiscais, porta COMX, Modem, USB, e etc podem ser facilmente acessados.

A JNI permite que um código escrito em Java, utilize a implementação de uma biblioteca escrita em C/C++, Assembler, e outras tantas linguagens de programação.

Usa-se JNI em determinadas situações em que um código Java não pode acessar certos recursos, por causa da estrutura dos Sistemas Vituais (sob qual a Máquina Virtual

Search...

Recent Posts

- “Code Signing ” Stack – iOS
- Integração entre linguagens. C++ -> C#
- Integração entre linguagens. C# -> C++
- Integração entre linguagens. C++ -> Obj-c
- Integração entre linguagens. Obj C -> C++

Archives

- June 2015
- May 2015
- April 2015
- March 2015

Java foi implementada) ou, como é nosso caso, quando queremos dar mais segurança à um código crítico, visto que a engenharia reversa num código c/c++ é muito mais complexa do que em um código java.

Além disso, você pode ter um conjunto de bibliotecas escritas em outras linguagens, e pode querer disponibilizá-las em seus programas Java. Isso facilita até o reuso de aplicações não-java. Pode ser usado para implementação de pontos críticos, em aplicações que exigam recursos de baixo-nível como código IN LINE, Assembler, e sua aplicação Java pode chamar todas essas funções.

A JNI, também habilita a você o uso das vantagens da Linguagem de Programação Java em seus métodos nativos, você pode capturar e lançar exceções de uma biblioteca qualquer, e tratá-las dentro de seu programa Java.

Depois dessa visão geral, vamos à implantação.

Requisitos:
Android Studio (utilizei a versão 1.1.0) – Download
Android NDK – Download

Configuração
(Presumindo que você já esteja com o Android Studio instalado e configurado)
Descompactar o ndk e copiar para o mesmo nível da pasta sdk do Android.
Criar uma referência para o ndk-build para facilitar o acesso de qualquer pasta.

Crie um projeto de novo.

Criar a classe nova (além da classe principal, *MainActivity* por padrão, criada automaticamente pelo Android Studio) que utilizará o método nativo. Nesse exemplo, chamarei de Wrapper.
O nosso método se chamará generate(). O *native* indica que ela retornará algo via JNI, nesse caso, a chave criada no c++.

- February 2015
- December 2014
- November 2014
- October 2014
- September 2014

Categories

Artigos
Equipe
Eventos
Infraestrutura
Métricas
Plataformas
Posicionamentos
Processos
Roadmap
Uncategorized

Meta

- Log in
- Entries RSS
- Comments RSS
- WordPress.org

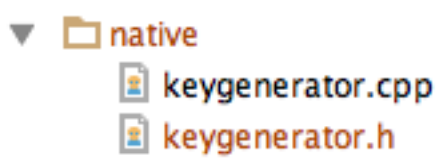
```
public class Wrapper {
     public native String generate();
}
```

Na *MainActivity*, crie um TextView para exibir essa informação e nomeie-o como **tvSenha**
Dentro no *onCreate* do MainActivity cole este código.

```
Wrapper wrapper = new Wrapper();
TextView tv = (TextView) findViewById(R.id.tvSenha);
tv.setText(wrapper.generate());
```

Crie um pasta, chamarei de *native*, no mesmo nível da pasta java, onde estarão os arquivos do c/c++ com a lógica propriamente dita.

Os arquivos se chamarão *keygenerator*, a classe declarada vai ser a *KeyGenerator* e o método chamado *generate()*.
A estrutura da pasta ficará assim:



keygenerator.h

```
#include
using namespace std;

class KeyGenerator{
    public:
        std::string generate();
};
```

keygenerator.cpp

```

```

A chave é somente a diferença em segundos entre a data atual e o primeiro segundo de 2015.
Esse é um exemplo simples, apenas para demonstrar a utilização.

Crie uma pasta chamada jni dentro do diretorio */src/main/java/* com os arquivos *Android.mk*, *Application.mk* e um arquivo *.cpp* (usarei *wrapper.cpp* nesse exemplo).

No aplicativo *Android.mk*, copie esse código:

O pontos mais importantes são :
LOCAL_MODULE := keygenerator
Nome que você dará ao seu módulo c++.

***LOCAL_C_INCLUDES : ../native/ ***
Local onde estão armazenados os arquivos c++ do seu módulo.

(Se quiser saber mais sobre como configurar os campos e as opções disponíveis, acesse esse link : Android Mk)

No *Application.mk*, cole o seguinte código:

Se quiser saber mais sobre o funcionamento, acesse o link :
Application Mk

O arquivo *wrapper.cpp* na pasta jni que de fato realiza a comunicação do nativo com o java.
É um dos arquivos mais importantes e cheio de detalhes, qualquer erro pode impossibilitar a comunicação.
Ele vai depender muito da localização da classe que o implementa pois a referencia estaticamente. Então, a qualquer alteração nessa localização, este arquivo deve ser revisado.
Minha estrutura está assim :

O código necessário será este:

Onde:
KeyGenerator é a classe que você declarou no nativo (no meu caso em /native/keygenerator.h)
jstring – Retorno do metodo
Java_com_example_guilhermelucas_config_Wrapper_generate – Caminho completo da classe que utiliza o método nativo.
Dividindo em:

- **Java_com_example_guilhermelucas_config** – Caminho
- **Wrapper** – Classe
- **generate** – método
- **(JNIEnv* env, jobject obj)** – Parâmetros obrigatórios. Se houverem mais é só acrescentar.

Entre pelo terminal no pasta jni criada. (o Android Studio possui um terminal interno em *View -> Tools Windows -> Terminal*)

O comando ndk-build irá gerar as libs .so que serão utilizadas no projeto:

```
$ ndk-build -B -j4
```

(Os outros parâmetros você pode conferir em NDK-Build)

Isso irá gerar 2 novas pastas, “libs” e “obj”

O Android Studio possui uma configuração ativa que realiza um build automático das classes c/c++ ao tentar dar build no sistema. No nosso cenário, vai dar erro de *NDK not configured*. Para desativar isso, insira o seguinte código no build.gradle, dentro do corpo de *android*.
Ele também já aponta a pasta libs como diretório jni.

```
sourceSets.main {
    jniLibs.srcDir 'src/main/libs'
    jni.srcDirs = []
}
```

O gradle vai pedir para sincronizar, aceite.

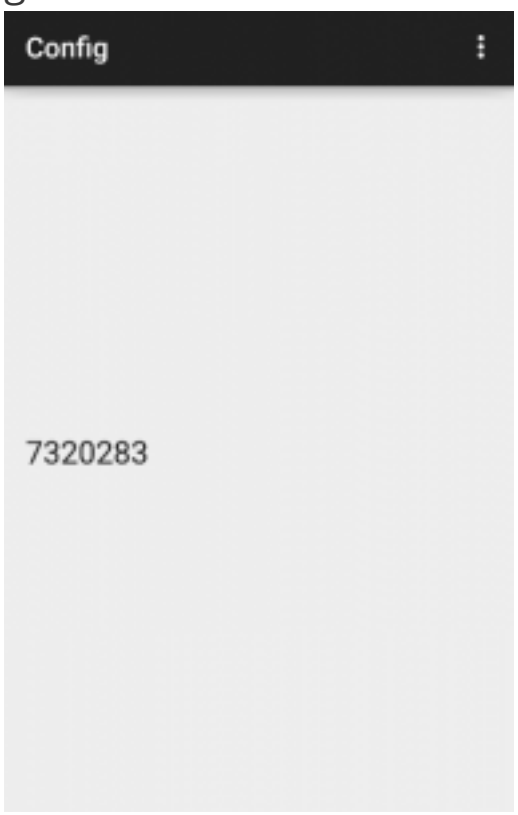
Agora volte à classe Wrapper para adicionar a referencia ao .so recém criado. Ficará assim:

```
public class Wrapper {
     static{
        System.loadLibrary("keygenerator");
     }

     public native String generate();
}
```

Obs: o nome da library é o que você deu no Android.mk

Ao executar, o resultado é um TextView com a senha gerada



Related Articles

Integração entre linguagens.

C++ -> C#

Integração entre linguagens.

C# -> C++

Integração entre linguagens.

Obj C -> C++

Integração entre linguagens.

C++ -> Java

Eficiência da Equipe

Autenticação

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

You may use these HTML tags and attributes:

```
<a href="" title=""> <abbr title=""> <acronym title=
""> <b> <blockquote cite=""> <cite> <code class="" t
itle="" data-url=""> <del datetime=""> <em> <i> <q c
ite=""> <strike> <strong> <pre class="" title="" dat
a-url=""> <span class="" title="" data-url="">
```

Post Comment

Search...

Meta

- Log in
- Entries RSS
- Comments RSS
- WordPress.org

Contact: mobile@gastecnologia.com.br