> system/core/libutils/Looper.cpp

# TODO

C++源码调用图生成器实现: <https://segmentfault.com/a/1190000008314896>

Backtrace源码分析与使用

# Libutils

## 打印堆栈CallStack

### 为什么要打印函数调用堆栈

打印调用堆栈可以直接把问题发生时的函数调用关系打出来，非常有利于理解函数调用关系。比如函数A可能被B/C/D调用，如果只看代码，B/C/D谁调用A都有可能，如果打印出调用堆栈，直接就把谁调的打出来了。

不仅如此，打印函数调用堆栈还有另一个好处。在Android代码里，函数命名很多雷同的，虚函数调用，几个类里的函数名相同等，即使用source insight工具看也未必容易看清函数调用关系。如果用了堆栈打印，很容易看到函数调用逻辑。

那么一个问题来了，Android/kernel本身在发生问题(kernel panic, tombstone, …)时，都可以打出详细的堆栈信息，这里干嘛还要费劲研究打堆栈？答案是发生问题时的堆栈的确很详细，但这里研究的是不影响(准确说是基本不影响)系统运行的境况下，打印出某个情形下的堆栈信息，这个对源代码逻辑研究很有帮助。

### 如何调用

#### Linux Kernel

Kernel里最简单，直接有几现成的函数可以使用：

dump\_stack() 这个函数打出当前堆栈和函数调用backtrace后接着运行

WARN\_ON(x) 这个函数跟dump\_stack很像，它有个条件，如果条件满足了就把stack打出来。

打印出来的结果都在kernel log里，一般dmesg命令就可以看到了

#### Native C++

Android在新版(至少5.0, 6.0)里加入了CallStack类，这个类可以打出当前的backtrace。用法很简单：

* 前面确保包含头文件#include <utils/CallStack.h>
* Android.mk的库依赖列表(LOCAL\_SHARED\_LIBRARIES)里包含libutils，一般都已经包含了。
* 然后在要打印堆栈处加入android::CallStack cs("haha");

"haha"是在logcat输出的TAG，这里可以自己定义。如果上下文已经在android namespace里，”android::”前缀就不必加了。

Native C++的输出log可以在logcat里看到。

注意，错误用法，在新版Android里编译不过，在网上的一些文档里说要这么用：

CallStack stack;

stack.update();

stack.dump();

#### Native C

Android对C的堆栈打印支持不太好。一个简单方法是用C语言调C++的函数，对，就是extern “C”。

1.先在项目里加入一个c++文件，比如mycallstack.cpp，里面是：

#include <utils/CallStack.h>

extern "C" void dumping\_callstack(void);

void dumping\_callstack(void)

{

android::CallStack cs("haha");

}

2.在项目里再加入一个c++的头文件，比如mycallstack.h，里面是：

void dumping\_callstack(void);

3.在Android.mk里源文件列表LOCAL\_SRC\_FILES里加入callstack.cpp，确保libutils在依赖列表里。

4.在native C里include callstack.h后直接调用dumping\_callstack()就可以了。

这个log也可以在logcat里看到。

过时方法，新版Android上libcorkscrew已经被拿掉了，网上的加载libcorkscrew库的方法自然就不能用了。过去网上的文章一般是推荐libcorkscrew.so，并加入大段代码来unwind\_backtrace已经不可行了。。

#### Java

Java最简单，它的backtrace最详细，连文件名和行号都打出来了:

Exception e = new Exception("haha");

e.printStackTrace();

log在logcat里看以看到。

???

### 如何查看

Log级别

### 源码分析

#### CallStack

CallStack(const char\* logtag, int32\_t ignoreDepth=1);

this->update(ignoreDepth+1);

this->log(logtag);

#### update

void CallStack::update(int32\_t ignoreDepth, pid\_t tid) {

mFrameLines.clear();

std::unique\_ptr<Backtrace> backtrace(Backtrace::Create(BACKTRACE\_CURRENT\_PROCESS, tid));

if (!backtrace->Unwind(ignoreDepth)) {

ALOGW("%s: Failed to unwind callstack.", \_\_FUNCTION\_\_);

}

for (size\_t i = 0; i < backtrace->NumFrames(); i++) {

mFrameLines.push\_back(String8(backtrace->FormatFrameData(i).c\_str()));

}

#### log

void CallStack::log(const char\* logtag, android\_LogPriority priority, const char\* prefix) const {

LogPrinter printer(logtag, priority, prefix, /\*ignoreBlankLines\*/false);

print(printer);

}

在Android framework的C++层或其他C++服务中需要打印调用栈时，可以使用android中的CallStack类进行打印，该类的实现如下：

#### print

void CallStack::print(Printer& printer) const {

for (size\_t i = 0; i < mFrameLines.size(); i++) {

printer.printLine(mFrameLines[i]);

}

}

//TODO

[Android 平台 Native 代码的崩溃捕获机制及实现](https://my.oschina.net/bugly/blog/1354954)

[JNI中如何打印Call Stack](https://blog.csdn.net/xxooyc/article/details/51001345)

[Android下打印调试堆栈方法](https://blog.csdn.net/freshui/article/details/9456889)

### REF

[Android C++层打印调用](https://blog.csdn.net/smilefyx/article/details/54896758)栈

[Android下面打印进程函数调用堆栈(dump backtrace)的方法](https://www.cnblogs.com/CoderTian/p/6149332.html)

## 消息循环机制