# %TDO

http://gityuan.com/2016/09/24/android-thread/

1. **概述**

Sdfg

基于Android 6.0源码剖析，分析Android线程的创建过程

android-7.1.1\_r1\libcore\ojluni\src\main\java\java\lang\Thread.java

/android/libcore/libart/src/main/java/java/lang/Thread.java

/art/runtime/native/java\_lang\_Thread.cc

/art/runtime/native/java\_lang\_Object.cc

/art/runtime/thread.cc

/system/core/libutils/Threads.cpp

/system/core/include/utils/AndroidThreads.h

/frameworks/base/core/jni/AndroidRuntime.cpp

Android平台上的Java线程，就是Android虚拟机线程，而虚拟机线程由是通过系统调用而创建的Linux线程。纯粹的Linux线程与虚拟机线程的区别在于虚拟机线程具有运行Java代码的Runtime. 除了虚拟机线程，还有Native线程，对于Native线程有分为是否具有访问Java代码的两类线程。接下来，本文分析介绍这3类线程的创建过程。

**二. Java线程**

**2.1 Thread.start**

[-> Thread.java]

public synchronized void start() {

checkNotStarted(); //保证线程只有启动一次

hasBeenStarted = true;

//[见流程2.2]

nativeCreate(this, stackSize, daemon);

}

nativeCreate()这是一个native方法，那么其所对应的JNI方法在哪呢？在java\_lang\_Thread.cc中通过gMethods是一个JNINativeMethod数组，其中一项为：

NATIVE\_METHOD(Thread, nativeCreate, "(Ljava/lang/Thread;JZ)V"),

这里的NATIVE\_METHOD定义在java\_lang\_Object.cc文件，如下：

#define NATIVE\_METHOD(className, functionName, signature) \

{ #functionName, signature, reinterpret\_cast<void\*>(className ## \_ ## functionName) }

将宏定义展开并代入，可得所对应的方法名为Thread\_nativeCreate,那么接下来进入该方法。