# Valgrind

目录

[Valgrind 1](#_Toc495664965)

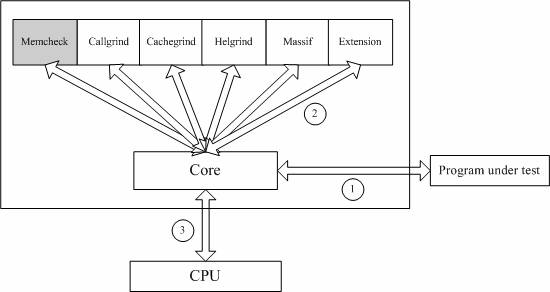
[Valgrind工具集 1](#_Toc495664966)

[构建 Valgrind 2](#_Toc495664967)

[ROOT设备 3](#_Toc495664968)

[配置设备 3](#_Toc495664969)

[Valgrind](http://valgrind.org/)是一套多平台，开放源代码（GPLV2）的仿真调试工具的集合。Valgrind由内核（core）以及基于内核的其他调试工具组成。内核类似于一个框架（framework），它模拟了一个CPU环境，并提供服务给其他工具；而其他工具则类似于插件 (plug-in)，利用内核提供的服务完成各种特定的内存调试任务其他工具则类似于插件(plug-in)，利用内核提供的服务完成各种特定的内存调试任务。



Valgrind工具集

⚫ Memcheck：这是valgrind应用最广泛的工具，一个重量级的内存检查器，能

够发现开发中绝大多数内存错误使用情况，比如：使用未初始化的内存，使用

已经释放了的内存，内存访问越界等。

⚫ Callgrind：它主要用来检查程序中函数调用过程中出现的问题。

⚫ Cachegrind：它主要用来检查程序中缓存使用出现的问题。

⚫ Helgrind：它主要用来检查多线程程序中出现的竞争问题。

⚫ Massif：它主要用来检查程序中堆栈使用中出现的问题。

⚫ Extension：可以利用core提供的功能，自己编写特定的内存调试工具。

|  |
| --- |
| ★注意：尽管您可以使用 Valgrind 对 Android 进行调试，但大多数 Android 平台开发人员都会使用 AddressSanitizer。 |

构建 Valgrind

首先需要到<http://valgrind.org/downloads/current.html> 获取valgrind的源码，或者使用git 工具clone 源码：

|  |
| --- |
| git clone git://sourceware.org/git/valgrind.git |

下载好源码后，打开源码根目录，在这个目录有一个README.android，参照里面的说明编译valgrind。一下为编译脚本样例：

|  |
| --- |
| #!/bin/bash  cd /Users/libo/WorkStation/GitHub/valgrind  export HWKIND=generic  export NDKROOT=/Users/libo/WorkStation/SDKs/android-ndk-r13b  export APP\_PLATFORM=android-15  export TOOLCHAIN\_SET=arm-linux-androideabi-4.9  export LOCAL\_HOST=darwin-x86\_64  #配置编译工具  export AR=$NDKROOT/toolchains/$TOOLCHAIN\_SET/prebuilt/$LOCAL\_HOST/bin/arm-linux-androideabi-ar  export LD=$NDKROOT/toolchains/$TOOLCHAIN\_SET/prebuilt/$LOCAL\_HOST/bin/arm-linux-androideabi-ld  export CC=$NDKROOT/toolchains/$TOOLCHAIN\_SET/prebuilt/$LOCAL\_HOST/bin/arm-linux-androideabi-gcc  #配置编译选项  export RANLIB=$NDKROOT/toolchains/$TOOLCHAIN\_SET/prebuilt/$LOCAL\_HOST/bin/arm-linux-androideabi-ranlib  export STRIP=$NDKROOT/toolchains/$TOOLCHAIN\_SET/prebuilt/$LOCAL\_HOST/bin/arm-linux-androideabi-strip  export CPPFLAGS="--sysroot=$NDKROOT/platforms/$APP\_PLATFORM/arch-arm -DANDROID\_HARDWARE\_$HWKIND"  export CFLAGS="--sysroot=$NDKROOT/platforms/$APP\_PLATFORM/arch-arm"  export LIBS="-L$NDKROOT/platforms/$APP\_PLATFORM/arch-arm/usr/lib"  #检查编译环境  ./autogen.sh  #执行valgrind源码配置  ./configure --prefix=/data/local/Inst --host=armv7-unknown-linux --target=armv7-unknown-linux --with-tmpdir=/sdcard  #编译  make -j8  make -j8 install DESTDIR=`pwd`/Inst |

编译完成后在valgrind的根目录下的生成Inst目录，这里面就是valgrind的可执行程序。在正式使用valgrind之前，我们还需要对设备进行root。

ROOT设备

首先下载[root精灵](http://www.shuame.com/root/) 或者其他的一些root工具，要获得更多的root工具，可以在[百度](https://www.baidu.com/)或者[google](http://www.google.com)中搜索Android Root。安装完工具后插上设备，开启设备的开发者选项以及打开USB调试，然后对设备进行ROOT，ROOT完毕后在命令行中执行 adb root,如果提示 adbd cannot run as root in production builds，那么说明设备还没有完全root，这时候我们可以安装【超级adbd】来进行root权限提升，在设备上安装完超级adbd后，再次运行adb root，如果提示adbd is already running as root那说明已经可以使用设备root权限。另外可以安装【Root Explorer】来进行文件操作。

配置设备

ROOT完设备后，我们需要把valgrind 安装到设备上,将 valgrind更目录下的内容拷贝到设备根目录下：

|  |
| --- |
| adb push /Users/libo/WorkStation/GitHub/valgrind/Inst/data/ / |

要让valgrind能够检查我们的程序，还需要对valgrind启动项进行配置，我们使用一个wrapper来完成这项工作。将这段代码保存为 valgrind\_wrapper。

|  |
| --- |
| #!/system/bin/sh  VGPARAMS='-v \  --error-limit=no \  --trace-children=yes \  --track-fds=yes \  --log-file=/sdcard/$package.%p.valgrind.log \  --tool=memcheck \  --leak-check=full \  --track-origins=yes \  --show-reachable=yes'  export TMPDIR=/data/data/$package  exec /data/local/Inst/bin/valgrind \$VGPARAMS \$\* |

|  |
| --- |
| ★valgrind 的参数介绍：  用法: valgrind [options] prog-and-args [options]: 常用选项，适用于所有Valgrind工具  1. -tool=<name> 最常用的选项。运行 valgrind中名为toolname的工具。默认memcheck。  2. h –help 显示帮助信息。  3. -version 显示valgrind内核的版本，每个工具都有各自的版本。  4. q –quiet 安静地运行，只打印错误信息。  5. v –verbose 更详细的信息, 增加错误数统计。  6. -trace-children=no|yes 跟踪子线程? [no]  7. -track-fds=no|yes 跟踪打开的文件描述？[no]  8. -time-stamp=no|yes 增加时间戳到LOG信息? [no]  9. -log-fd=<number> 输出LOG到描述符文件 [2=stderr]  10. -log-file=<file> 将输出的信息写入到filename.PID的文件里，PID是运行程序的进行ID  11. -log-file-exactly=<file> 输出LOG信息到 file  12. -log-file-qualifier=<VAR> 取得环境变量的值来做为输出信息的文件名。 [none]  13. -log-socket=ipaddr:port 输出LOG到socket ，ipaddr:port  LOG信息输出  1. -xml=yes 将信息以xml格式输出，只有memcheck可用  2. -num-callers=<number> show <number> callers in stack traces [12]  3. -error-limit=no|yes 如果太多错误，则停止显示新错误? [yes]  4. -error-exitcode=<number> 如果发现错误则返回错误代码 [0=disable]  5. -db-attach=no|yes 当出现错误，valgrind会自动启动调试器gdb。[no]  6. -db-command=<command> 启动调试器的命令行选项[gdb -nw %f %p]  适用于Memcheck工具的相关选项：  1. -leak-check=no|summary|full 要求对leak给出详细信息? [summary]  2. -leak-resolution=low|med|high how much bt merging in leak check [low]  3. -show-reachable=no|yes show reachable blocks in leak check? [no] |

valgrind\_wrapper编写好了以后我们需要将它拷贝到设备上的/data/local/目录下：

|  |
| --- |
| adb push /Users/libo/valgrind\_wrapper /data/local/valgrind\_wrapper |

修改valgrind和valgrind\_wrapper程序权限

|  |
| --- |
| adb shell chown root.shell /data/local/Inst/bin/valgrind /data/local/valgrind\_wrapper  adb shell chmod 777 /data/local/Inst/bin/\*  adb shell chmod 777 /data/local/Inst/lib/\*  adb shell chmod 777 /data/local/Inst/lib/valgrind/\*-arm-linux  adb shell chmod 777 /data/local/valgrind\_wrapper |

到此，valgrind运行所需的基本都准备完毕，接下来就是要在某个程序上运行valgrind，比如我们需要在一个叫com.example.test的应用上运行valgrind，我们需要执行以下命令

|  |
| --- |
| adb shell su -c "setprop wrap.com.example.test 'logwrapper /data/local/valgrind\_wrapper'" |

这时候我们启动程序，就是以valgrind的方式启动了，为了valgrind更好的工作，在编译我们android程序原生代码的时候，需要加入一些编译配置：

|  |
| --- |
| LOCAL\_CPPFLAGS += -g -fno-omit-frame-pointer -O0  LOCAL\_CFLAGS += -g -fno-omit-frame-pointer -O0 |

如果可以的话也可以使用debug模式，在Application.mk中配置

|  |
| --- |
| APP\_CPPFLAGS += -DDEBUG  APP\_OPTIM := debug |

配置好了以后就可以编译运行程序了。运行一段时间后退出程序，在 /sdcard/目录下回生成com.example.test.$PID.valgrind.log文件，打开文件就可以查找和自己程序相关的问题了。

为了方便部署，我们可以使用以下脚本来部署valgrind到设备上，将以下代码保存为valgrind\_setup，用法valgrind\_setup –package com.example.test

|  |
| --- |
| #!/bin/bash  set -e  HERE="$(cd "$(dirname "$0")" && pwd)"  revert=no  package=  device=  function usage {  echo "usage: $0 [--revert] [--device device-id] [--package package-name]"  echo " --revert: Uninstall valgrind from the device."  echo " --package: The application package name."  echo " --device: Install to the given device. Use 'adb devices' to find"  echo " device-id."  echo  exit 1  }  while [[ $# > 0 ]]; do  case $1 in  --revert)  revert=yes  ;;  --package)  shift  if [[ $# == 0 ]]; then  echo "--package requires an argument."  exit 1  fi  package="$1"  ;;  --device)  shift  if [[ $# == 0 ]]; then  echo "--device requires an argument."  exit 1  fi  device="$1"  ;;  \*)  usage  ;;  esac  shift  done  ADB=${ADB:-adb}  if [[ x$device != x ]]; then  ADB="$ADB -s $device"  fi  $ADB shell am force-stop $package  echo '>> Remounting /system rw'  $ADB root  $ADB wait-for-device  $ADB remount  $ADB wait-for-device  if [[ x$revert == xyes ]]; then  echo '>> Uninstalling ASan'  $ADB shell rm /data/local/Inst  $ADB shell rm /data/local/valgrind\_wrapper  $ADB shell setprop wrap.$package ""  echo '>> Restarting shell'  echo '>> Done'  exit 0  fi  cat <<EOF >"$HERE/valgrind\_wrapper"  #!/system/bin/sh  VGPARAMS='-v \  --error-limit=no \  --trace-children=yes \  --track-fds=yes \  --log-file=/sdcard/$package.%p.valgrind.log \  --tool=memcheck \  --leak-check=full \  --track-origins=yes \  --show-reachable=yes'  export TMPDIR=/data/data/$package  exec /data/local/Inst/bin/valgrind \$VGPARAMS \$\*  EOF  $ADB push $HERE/Inst/data/ /  $ADB push $HERE/valgrind\_wrapper /data/local/valgrind\_wrapper  $ADB shell chown root.shell /data/local/Inst/bin/valgrind /data/local/valgrind\_wrapper  $ADB shell chmod 777 /data/local/Inst/bin/\*  $ADB shell chmod 777 /data/local/Inst/lib/\*  $ADB shell chmod 777 /data/local/Inst/lib/valgrind/\*-arm-linux  $ADB shell chmod 777 /data/local/valgrind\_wrapper  $ADB shell su -c "setprop wrap.$package 'logwrapper /data/local/valgrind\_wrapper'"  echo "wrap.$package: $(adb shell getprop wrap.$package)"  # $ADB logcat -c  # $ADB logcat |