Under Linux cross compiler to generate the iOS tool chain

1.guide

IntroductionCompany requirements on Linux system to realize the cross compile and dynamic distributed Android and iOS static library, or dynamic libraries, and one of the most critical is the Android the generation of the NDK and iOS tool chain.Because the Android it is relatively easy to implement, here focuses on the iOS side generation tool chain implementation process.All of the following operation was done on the Macbook Pro.

2.The resources needed to

2.1 VWware Fusion 10 (https://my.vmware.com/web/vmware/details?DownloadGroup = FUS - 1001 & productId = 688 & rPId = 18811)

2.2 Ubuntu (version 16.04 and above) (https://www.ubuntu.com/download/desktop)

2.3 LLVM / Clang (> = 5.0) (http://releases.llvm.org/download.html#5.0.0)

2.4 openssl (https://github.com/openssl/openssl)

2.5 automake (http://ftp.gnu.org/gnu/automake/)

2.6 autogen (http://ftp.gnu.org/gnu/autogen/)

2.7 libtool (http://ftp.gnu.org/gnu/libtool/)

2.8 autoconf (http://ftp.gnu.org/gnu/autoconf/)

2.9 libssl - dev (https://pkgs.org/download/libssl-dev)

2.10 Python (https://www.python.org/downloads/)

2.11 cctools - port (https://github.com/tpoechtrager/cctools-port)

2.12 cmake (https://cmake.org/download/)

2.13 IOS - SDK (http://resources.airnativeextensions.com/ios/)

3. The resource configuration and installation

3.1 VWware Fusion 10 and Ubuntu installation method

http://blog.csdn.net/jackjia2015/article/details/50757430

3.2 LLVM / Clang installation method

3.2.1: download LLVM, Clang, Compiler - rt (optional), the Clang - tools - extra source and extract the (optional).LLVM decompression after the file name instead of to the file name to Clang Clang decompression after, the Compiler - rt decompression after the file name to the Compiler - rt, Clang - tools - extra decompression after the file name is changed to Clang - tools - extra

3.2.2: put the clang LLVM/tools/directory, the compiler - rt into LLVM/projects/directory, the clang - tools - extra into LLVM/tools/clang/tools/directory

3.2.3: return to the LLVM directory, at the same level of the mkdir llvmbuild (file name themselves, reasonable)Llvmbuild

3.2.4: cd, then execute the cmake../ LLVM - DLLVM\_ENABLE\_ASSERTIONS = OFF - DCMAKE\_BUILD\_TYPE = RELEASE

3.2.5: after the 3.2.4 perform, execute the make - Jacqueline Nottingham (where N Settings for your virtual machine for you on the number of processor cores, as shown in the figure below)As shown above, I set up for my virtual machine a processor core, so I should write the make - j1

3.2.6:3.2.5 completes after the success, the implementation of sudo make install, to the LLVM, the clang installation has been completed

3.3 openssl installation

3.3.1 the official method is given

3.3.1.1 unzip the filesThe tar - xf openssl - g.t ar 1.0.1. Gz (recommended to download the latest version)

3.3.1.2 configuration. / config -- prefix = / usr/local/openssl - openssldir = / usr/local/openssl

3.3.1.3 compilationThe make

3.3.1.4 installationSudo make install

3.3.1.5 set the environment variableSudo gedit ~ / bashrc, add the export PATH = / usr/local/openssl/bin: $PATH, save and launch.Finally perform source ~ /. Bashrc gives effect to environment variables.

3.3.2 rainfall distribution on 10-12 recommended installation method

3.3.2.1 directly using sudo apt - get the install openssl commands to installAutomake

3.4 installation

3.4.1 track advice directly to use sudo apt - get the install automake installation instruction

3.5 installation autogen

3.5.1 track of advice directly to use sudo apt - get the install autogen installation instructionLibtool

3.6 installation

3.6.1 advice directly to use sudo apt - get the install libtool installation instruction

3.7 install autoconf

3.7.1 advice directly to use sudo apt - get the install autoconf installation instruction

3.8 installation libssl - dev

3.8.1 advice directly to use sudo apt - get the install libssl - dev instruction installation

3.9 Install python

3.9.1 advice directly to use sudo apt - get the install python instructions

3.10 cctools - port

3.10.1 extract directly, without other actions

3.11 install cmake

3.11.1 decompression after download the source code

3.11.2 cd file, execute. / the bootstrap instructions

3.11.3 make - Jacqueline Nottingham (here the N distribution of virtual machine according to you on the number of processor cores and decide)

3.11.4 front all no problem, after executing sudo make install, you can perform cmake - version after the completion of all to verify whether the installation is successfulIOS SDK

3.12 packaging steps (here I am using Xcode9.1, packaged iOS11.1 SDK.If you need to pack iOS8, iOS9, iOS10 SDK, the following method is also applicable, pay attention to the following TMP is our temporary create path)

3.12.1: SDK = $(ls -l/Applications/Xcode. The app/Contents/Developer/Platforms/iPhoneOS platform/Developer/SDKs | grep "- > iPhoneOS. The SDK" | head - n1 | awk '} {print $9 ')

3.12.2: cp - r/Applications/Xcode. App/Contents/Developer/Platforms/iPhoneOS platform/Developer/SDKs/iPhoneOS SDK/TMP / $1 > SDK/dev/null

3.12.3: cp - r/Applications/Xcode. App/Contents/Developer/Toolchains/XcodeDefault xctoolchain/usr/include/c + + / v1 / TMP / $SDK/usr/include/c + + 1 > / dev/null

3.12.4: pushd/TMP

3.12.5: tar - CVZF $SDK. Tar. Gz $SDK (at this point we need the corresponding version of the iOS SDK has been packaged complete)

3.12.6: rm - rf $SDK

3.12.7: $mv SDK. Tar. Gz ~

3.12.8: popd (at this point we're going to generate the iOS preparations have completed before tool chain)

4. Generate iOS tool chain (with the support of the generated armv7, armv7s tool chain, I am using iOS9.3 SDK)

4.1 generate arm64 tool chain

4.1.1 cd cctools - port

4.1.2 execution IPHONEOS\_DEPLOYMENT\_TARGET = 8.0 usage\_examples/ios\_toolchain/build. Sh ~ / iPhoneOS11.1 SDK. Tar. Gz arm64

4.1.3 tools after the success of the chain will be prompted to \* \* \* \* \* \* all done, now cctools - port/usage\_examples/ios\_toolchain/next target is the tool chain (now arm64 tool chain has generated)

4.1.4 into/usr/local, we create a file, named iOS - arm64, and then return to cctools - port/directory, executing sudo usage\_examples mv/ios\_toolchain/target/usr/local/iOS - arm64

4.1.5 executing sudo cp/usr/local/iOS - arm64 / target/lib/libtapi. So/usr/lib /

4.1.6 execute the export PATH = $PATH: / usr/local/iOS - arm64 / bin

4.1.7 verify whether our tool chain can use (it is very important to the image below)

4.1.7.1 cd cctools - port - master/usage examples/ios\_toolchain/target /

4.1.7.2 mkdir SRC (SRC) to create file

4.1.7.3 vim test. C (create a c file)

4.1.7.4../ bin/arm - apple - darwin11 - clang - xc - c test. C (compile. C file, generated. O file)

4.1.7.5../ bin/arm - apple - darwin11 - ar cr libtest. A test. O (. O documents compiled into. A library, I take the name of the static library for libtest. A)

4.1.7.6 get here if you are successful, Congratulations, you 0.0000001 meters short from success, here's the test. The c and libtest. Copy out a document, in your Xcode validation we use iOS - arm64 static library generated by the tool chain, can be used normally.

4.2 generate other tool chain method basic same as above