

如何在一台未安装 tesseract 等相应库的电脑上（linux 系统）使用 jni 调用 tesseract 去做 OCR 识别？

以下介绍均以附件中的 demo 为例。

1、JNI 调用 tesseract 前期准备

首先需要注意的是 JNI 只能加载动态库，不能直接调用静态库，参考这一篇文章。

http://blog.csdn.net/dream_it_life/article/details/8598616

文章部分摘录：加载 so 问题，标准方法就可以了，有的时候需要用到静态库，即*.a，这时候 System.load 和 System.loadLibrary 无法将其加载，解决方法是将其引用在 C++ 的开发环境中配置好，一同编译成 so 文件，供 JNI 调用。

为了解决 JNI 只能加载动态库的问题，想到两种方法：

<1>、JNI 调用 C++ 接口 ocr_java.cpp 编译成动态库，即 libocr_java.so，然后由它再链接到 tesseract、lept 的静态库；

经过验证，这种方法行不通，调用时会提示一个错误，具体分析如下：

```
/usr/bin/ld: ../../../libraries/log4cplus/liblog4cplus.a(fileappender.o): relocation R_X86_64_32S against `a local symbol' can not be used when making a shared object; recompile with -fPIC
```

从错误 log 中看到 recompile with -fPIC，起初是以为 Makefile 中生成 libocr_java.so 需要加-fPIC，尝试后发现仍提示该错误。

参考网上资料得知，这个 recompile 指的是动态库调用的静态库 libtesseract.a、liblept.a、libtiff.a 这些等必须是由-fPIC 编译而成，如果其中某个静态库编译时未使用-fPIC 则会报上述错误。

如何知道静态库是否由-fPIC 编译而成？通过如下命令：

```
ar -x liblog4cplus.a //将静态库解成.o 文件
```

```
readelf --relocs fileappender.o | egrep '(GOT|PLT|JUMP_SLOT)' //查找其中.o 文件是否包含 GOT 或者 PLT 等字样
```

查看 liblept.a 解压得到的 dwacomb.2.o 不包含上述字符串，而 libtesseract.a 解压得到的 resultiterator.o 包含 PLT 字符串，可得知 liblept.a 编译未加-fPIC，而 libtesseract.a 是由-fPIC 编译而成：

```
root@ubuntu:/home/cloud/tesseract/lib# readelf --relocs dwacomb.2.o | egrep '(GOT|PLT|JUMP_SLOT)'
root@ubuntu:/home/cloud/tesseract/lib# ar x libtesseract.a
root@ubuntu:/home/cloud/tesseract/lib# readelf --relocs resultiterator.o | egrep '(GOT|PLT|JUMP_SLOT)'
000000000002e 0082000000004 R_X86_64_PLT32 0000000000000000 _Znam - 4
0000000000060 0083000000004 R_X86_64_PLT32 0000000000000000 _ZdaPv - 4
00000000000ae 0082000000004 R_X86_64_PLT32 0000000000000000 _Znam - 4
00000000000dc 0083000000004 R_X86_64_PLT32 0000000000000000 _ZdaPv - 4
0000000000128 0087000000004 R_X86_64_PLT32 0000000000000000 _ZN9tesseract12Pa
```

所以如果使用这种方式，将需要重编这些静态库，这个工程比较庞大，所以不推荐

使用这种方式。

具体参考这篇文章：

<http://stackoverflow.com/questions/19768267/relocation-r-x86-64-32s-against-linker-error>

<2>、JNI 全部调用动态库；

该方法验证可行，具体步骤参考如下介绍，以 demo 为例。

2、JNI 调用动态库具体配置步骤

<1>、首先保证电脑环境中安装有 JDK，以及配置好 JDK 相关路径（路径以自己电脑 JDK 路径为准）。

```
export JAVA_HOME=/usr/lib/jvm/java7
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib
export PATH=${JAVA_HOME}/bin:$PATH
```

如果系统已经安装 JDK 且配置好路径的话，则跳过该步骤。

<2>、将 demo 中 tesseract 文件夹拷贝到电脑中某个目录下，tesseract 文件夹中包含如下内容：

```
root@ubuntu:/home/cloud/tesseract# ls
auto_java_demo.sh      configure.sh            ocr_tesseract.h
auto_java_release.sh   eng_jpg                ocr_tesseract_share.tar.gz
auto_test_chi.sh       include                ocr_tesseract_static.tar.gz
auto_test_eng.sh       Makefile               readme.txt
chi_jpg                ocr_java.cpp           tessdata
com                    ocr_main.cpp
com_ocr_java.h         ocr_tesseract.cpp
```

<3>、修改 Makefile 中的 JDK 路径，改成自己电脑上的 JDK 路径

```
INCLUDE += -I /usr/lib/jdk1.8.0_101/include
```

<4>、执行 configure.sh，解压已经打包好的 ocr_tesseract_share.tar.gz 相应 lib，会在目录下生成一个 share_lib 用于存放依赖的库。

```
root@ubuntu:/home/cloud/tesseract# ./configure.sh
share_lib/
share_lib/libliblept.so
share_lib/libtiff.so.5
share_lib/libtiff.so
share_lib/libjpeg.so.8
share_lib/libjbig.so
share_lib/libtesseract.so
share_lib/libjpeg.so
share_lib/libpng.so
share_lib/libpng12.so.0
share_lib/libliblept.so.4
share_lib/libtesseract.so.3
```

<5>、将库路径加入到系统默认的 LD 链接路径中。（这部很重要，不然运行时系统会找不到本地的 lib）

vim /etc/ld.so.conf 增加解压的 lib 路径即可：（第一句是系统默认的，第二句是新增的路径）

```
root@ubuntu: /home/cloud/tesseract
include /etc/ld.so.conf.d/*.conf
/home/cloud/tesseract/share_lib
```

修改完 重新执行 ldconfig

```
root@ubuntu: /home/cloud/tesseract# ldconfig
root@ubuntu: /home/cloud/tesseract#
```

<6>、执行 `make clean;make` 生成动态库 `libocr_java.so`, 至此 JNI C++ 部分已经配置完成, 接下来配置 java demo 运行环境。

<7>、执行 `./auto_java_release.sh`, 重编 java 程序

<8>、修改 `auto_java_demo.sh` 中 `tesseract` 字符库训练路径, 如下我的 `tessdata` 是放在 `tesseract` 目录下:

```
root@ubuntu: /home/cloud/tesseract
#!/bin/bash

export TESSDATA_PREFIX=/home/cloud/tesseract/tessdata
echo $TESSDATA_PREFIX

java -Djava.library.path=. com.ocr_java eng_jpg/ocr1.tif eng_jpg/result.txt
```

<9>、执行 `./auto_java_demo.sh` 可看到运行结果, 调用成功。

```
root@ubuntu: /home/cloud/tesseract# ./auto_java_demo.sh
/home/cloud/tesseract/tessdata
input: eng_jpg/ocr1.tif
output: eng_jpg/result.txt
type: eng
ocr_tesseract start
OCR output: 1098765443

ocr finish
result is 1098765443

root@ubuntu: /home/cloud/tesseract#
```

3、接下来尝试非 JNI 调用的移植, 即直接在 linux 环境下运行可执行档。

- 静态库调用, 验证成功。
- 动态库调用, 验证成功。

<1>、前期配置步骤与 JNI 调用相同, 执行 `configure.sh`,

将已经打包好的 `ocr_tesseract_share.tar.gz` 相应动态库解压到 `share_lib` 下用于存放依赖的动态库供动态库调用方式使用;

同时将 `ocr_tesseract_static.tar.gz` 相应静态库解压到 `lib` 目录下, 同时将这些静态库合成为一个静态库 `libocr_tesseract.a`, 提供给静态库调用方式使用。

题外话: 为什么静态库可以合成一个, 动态库不行?

- 静态库.a 可以理解为一系列目标文件 .o 的打包，比如 libleft.a 就是 left 的源文件生成的.o 打包，其他的类似。所以如果要将这些.a 静态库合成一个静态库，需要首先将.a 文件解压生成.o 文件(通过命令 ar x 实现)，然后再把所有静态库的.o 文件再次打包成一个.a(通过命令 ar csru 实现)。

使用静态库编译生成的可执行档会把所依赖的静态库全部打包进去，这样可执行档的文件会比较大，但也有好处即对系统环境依赖比较小。

- 动态库并不是单纯的.o 打包，其中会包含 elf 链接文件，告诉系统调用时需要链接到哪个路径的动态库。

使用动态库编译生成的可执行档只打包本身的.o 文件，而不会把依赖库包含进去，优点是可执行档文件比较小，缺点是对系统依赖大，需要系统配合 LD 动态链接路径等等。

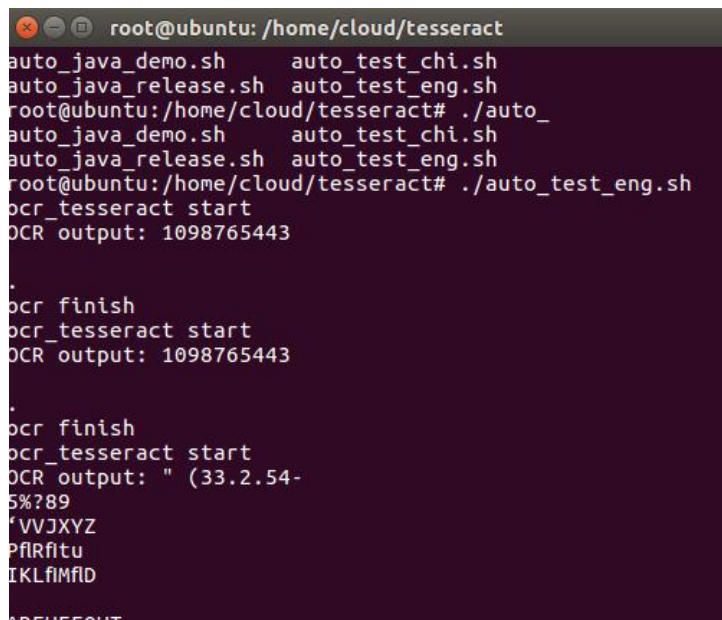
如下图对比可知动态库调用的执行档才 15K 左右，而静态库调用的执行档有 42M 大小，比较浪费资源，特别是系统有多个程序共享一个库的时候，采用动态库调用时最佳选择。

```
-rwxr-xr-x  1 root  root    15064 Aug 28 09:00 ocr_share_demo*  
-rwxr-xr-x  1 root  root 42545304 Aug 28 09:00 ocr_static_demo*
```

参考文档: <http://blog.chinaunix.net/uid-23069658-id-3142046.html>

<2>、执行 make clean;make 生成 ocr_share_demo 和 ocr_static_demo 可执行档;

<3>、执行 auto_test_eng.sh 可看到 ocr_share_demo 和 ocr_static_demo 运行结果一致。



```
root@ubuntu: /home/cloud/tesseract  
auto_java_demo.sh      auto_test_chi.sh  
auto_java_release.sh   auto_test_eng.sh  
root@ubuntu:/home/cloud/tesseract# ./auto_  
auto_java_demo.sh      auto_test_chi.sh  
auto_java_release.sh   auto_test_eng.sh  
root@ubuntu:/home/cloud/tesseract# ./auto_test_eng.sh  
ocr_tesseract start  
OCR output: 1098765443  
.  
ocr finish  
ocr_tesseract start  
OCR output: 1098765443  
.  
ocr finish  
ocr_tesseract start  
OCR output: " (33.2.54-  
5%?89  
'VVJXYZ  
PflRftu  
IKLflMfD  
ADFH55QHT
```

题外话:

<1>、其他库的调用可参考上述方法，如何知道要包含哪些.a 或者.so，比如 tesseract 依赖 left、jpeg、jbig、tiff、png 等等，可以通过网上搜索需要哪些 LIB，一般写的不是很全，这个时候可以尝试只加所知道的 lib，通过执行 make 发现提示错误信息，从而可以查找到有哪些库有遗漏，然后在/usr 目录下去搜索，并拷贝过来。

<2>、opencv 除了需要拷贝 include 头文件和 lib 文件外，还需要配置一些编译选项宏，所以上述方法可能不适用，还需要稍作修改才能用，这个暂时未验证，可通过执行 `pkg config` 看需要如何设置。

<3>、Makefile 中的库路径有先后关系，写反了编译会提示错误，比如 `ltesseract` 依赖 `llept`，而 `llept` 又依赖 `lpthread`，则顺序如下：

```
STATIC_LIBS = `pkg-config --libs tesseract`  
STATIC_LIB_PATH += -L .  
LIBS += -ltesseract -llept -ljpeg -ltiff -lpng -ljpeg  
LIB_PATH += -L . -L ./share_lib  
INCLUDE += -I ./include
```

<4>、tesseract 头文件调用不是放在系统默认路径下，是直接拷贝过来放在 `tesseract` 目录下的，所以程序里的

```
#include <tesseract/baseapi.h>
```

需要改成相对路径

```
#include "tesseract/baseapi.h"
```