Learning LLVM Part 2

作者：Sky3
原文链接：https://ld246.com/article/1541748937641
来源网站：链滴

Understanding LLVM Part 2

咕咕咕，接上篇《Learning LLVM Prat 1》。这篇大概梳理一下这两个月摸鱼时候遇到的坑。

一些坑

如果要编译成功LLVM树，那么熟练的使用各种方式把代码拖下来，再用合适的环境编译必不可少。

其实编译这个巨大的东西，还需要好一点的硬件设备（一般我们会都卡在硬盘IO上，IO卡住了就会Cache在内存里，然后内存爆炸，PC就什么也动不了了），比如在目前的这个工作站上：

```
sk in llvm-detectDataPass/skeleton at Lab on  master
 screenfetch
                    ./+o+-       sk@Lab
                yyyyy- -yyyyyy+      OS: Ubuntu 16.04 xenial
             ://+//////-yyyyyyo      Kernel: x86_64 Linux 4.15.0-38-generic
         .++ .:/++++++/-.+sss/`       Uptime: 28m
       .:++o:  /++++++++/:--:/-       Packages: 2031
      o:+o+:++.`..```.-/oo+++++/     Shell: zsh 5.1.1
     .:+o:+o/.          `+sssoo+/    CPU: Intel Core i7-8700K CPU @ 4.7GHz
 .++/+:+oo+o:`             /sssooo.   GPU: GeForce GTX 1080 Ti
/+++//+:`oo+o               /::--:.   RAM: 1280MiB / 64357MiB
\+/+o+++`o++o                ++////.
 .++.o+++oo+:`             /dddhhh.
     .+.o+oo:.            `oddhhhh+
      \+.++o+o``-````.:ohdhhhhh+
       `:o+++ `ohhhhhhhhyo++os:
         .o:`.syhhhhhhh/.oo++o`
             /osyyyyyyo++ooo+++/
                 ````  +oo+++o\:
                        `oo++.
```

硬盘是Intel 760P 512G的，Ninja编译完成Release版本的LLVM+clang-6.0.1十来分钟就搞定了（逃

## gcc版本问题

以我现在正在使用的LLVM 6.0.1为例，首先要把代码下载到本地，解压，这部分就不细说了，上篇提了。

然后就是配置合适的gcc和g++版本，在文档http://releases.llvm.org/6.0.1/docs/GettingStarted.html#software中可以看到gcc/g++的版本是>= 4.8.0，而我们一般Ubuntu自带的版本是5.4.0，如果行使用不恰当的版本编译，会被恶心的C++标准实现报错导致编译不成功，因此在Ubuntu 16.04 x64 Desktop LTS上，我们要使用恰当的gcc/g++版本。首先安装gcc-4.8和g++-4.8:

```
sudo apt install gcc-4.8 g++-4.8
```

先安装版本：

```
$ sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.8 60
$ sudo update-alternatives --install /usr/bin/g++ g++ /usr/bin/g++-4.8 60
```
```

切换版本：

```
sk in ~ at Lab took 7s
 sudo update-alternatives --config gcc
There are 2 choices for the alternative gcc (providing /usr/bin/gcc).

  Selection    Path              Priority   Status
------------------------------------------------------------
  0            /usr/bin/gcc-4.8   60        auto mode
* 1            /usr/bin/gcc-4.8   60        manual mode
  2            /usr/bin/gcc-5     60        manual mode

Press <enter> to keep the current choice[*], or type selection number: 1
# g++ 也是一样
sk in ~ at Lab took 3s
 sudo update-alternatives --config g++
update-alternatives: warning: alternative /usr/bin/g++-4.7 (part of link group g++) doesn't ex
st; removing from list of alternatives
There are 2 choices for the alternative g++ (providing /usr/bin/g++).

  Selection    Path              Priority   Status
------------------------------------------------------------
  0            /usr/bin/g++-5     80        auto mode
* 1            /usr/bin/g++-4.8   60        manual mode
  2            /usr/bin/g++-5     80        manual mode

Press <enter> to keep the current choice[*], or type selection number: 1
```

查看g++/gcc版本，如果是文档要求的版本就可以开始下一步了：

```
sk in ~ at Lab took 3s
 g++ -v
Using built-in specs.
COLLECT_GCC=g++
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/4.8/lto-wrapper
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 4.8.5-4ubuntu2' --with-bugur
=file:///usr/share/doc/gcc-4.8/README.Bugs --enable-languages=c,c++,java,go,d,fortran,objc
obj-c++ --prefix=/usr --program-suffix=-4.8 --enable-shared --enable-linker-build-id --libex
cdir=/usr/lib --without-included-gettext --enable-threads=posix --with-gxx-include-dir=/usr/
nclude/c++/4.8 --libdir=/usr/lib --enable-nls --with-sysroot=/ --enable-clocale=gnu --enable
libstdcxx-debug --enable-libstdcxx-time=yes --enable-gnu-unique-object --disable-libmudfl
p --enable-plugin --with-system-zlib --disable-browser-plugin --enable-java-awt=gtk --enabl
-gtk-cairo --with-java-home=/usr/lib/jvm/java-1.5.0-gcj-4.8-amd64/jre --enable-java-home -
with-jvm-root-dir=/usr/lib/jvm/java-1.5.0-gcj-4.8-amd64 --with-jvm-jar-dir=/usr/lib/jvm-exp
rts/java-1.5.0-gcj-4.8-amd64 --with-arch-directory=amd64 --with-ecj-jar=/usr/share/java/ecli
se-ecj.jar --enable-objc-gc --enable-multiarch --disable-werror --with-arch-32=i686 --with-ab
=m64 --with-multilib-list=m32,m64,mx32 --with-tune=generic --enable-checking=release --
uild=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 4.8.5 (Ubuntu 4.8.5-4ubuntu2)
```

debug版本文件夹过大的问题

这个问题需要在cmake生成编译文件的时候添加Release版本的参数：

```
(~/Work/llvm-6.0.1.build)$ cmake -G "Ninja" -DCMAKE_BUILD_TYPE=Release ../llvm-6.0.1.src
```

这样，编译生成的LLVM树就很小了：

```
sk in ~/Work/llvm-6.0.1.build at Lab
 du -sh
2.1G    .
```

看到没，只有2.1G，如果是debug版本的LLVM，需要45G。

使用CLion编译Pass

我们编写一Pass，还要把源码复制到LLVM源代码树里，和整个LLVM一起编译，这样也太不优雅了所以我们稍加配置，在CLion里编译（你电edu邮箱有CLion授权，我可是用正版的乖孩子）。

首先，我们需要在/etc/profile配置一个名为LLVM_HOME的环境变量，就是我们编译生成的LLVM树目录：

```
# /etc/profile添加一行
export LLVM_HOME=/home/sk/Work/llvm-6.0.1.build
```

然后我们需要配置cmake工程如下：

```
sk in llvm-detectDataPass/skeleton at Lab on  master
 tree
.
├── CMakeLists.txt
├── README.md
└── skeleton
    ├── CMakeLists.txt
    └── Skeleton.cpp
```

在根目录下的CMakeLists.txt内容如下：

```
sk in llvm-detectDataPass/skeleton at Lab on  master
 cat CMakeLists.txt
cmake_minimum_required(VERSION 3.4)
if(NOT DEFINED ENV{LLVM_HOME})
   message(FATAL_ERROR "$LLVM_HOME is not defined")
endif()
if(NOT DEFINED ENV{LLVM_DIR})
   set(ENV{LLVM_DIR} $ENV{LLVM_HOME}/lib/cmake/llvm)
endif()
find_package(LLVM REQUIRED CONFIG)
add_definitions(${LLVM_DEFINITIONS})
include_directories(${LLVM_INCLUDE_DIRS})
link_directories(${LLVM_LIBRARY_DIRS})

add_subdirectory(skeleton)  # Use your pass name here.
```

其中主要内容是设置编译环境使用的，不同版本的LLVM可能set(ENV{LLVM_DIR} $ENV{LLVM_HOE}/lib/cmake/llvm)的位置有所差别，这个需要视版本更改。最后一行是我们Pass的目录，这个也很

单。

在Pass目录的CMakeLists.txt的内容如下所示：

```
sk in llvm-detectDataPass/skeleton at Lab on  master
 cat skeleton/CMakeLists.txt
add_library(SkeletonPass MODULE
        # List your source files here.
        Skeleton.cpp
        )

## Use C++11 to compile your pass (i.e., supply -std=c++11).
target_compile_features(SkeletonPass PRIVATE cxx_range_for cxx_auto_type)

# LLVM is (typically) built with no C++ RTTI. We need to match that;
# otherwise, we'll get linker errors about missing RTTI data.
set_target_properties(SkeletonPass PROPERTIES
        COMPILE_FLAGS "-D__GLIBCXX_USE_CXX11_ABI=0 -fno-rtti"
        )
```

第一行add_library()中添加的是我们Pass的源代码，最后一行注意添加编译选项COMPILE_FLAGS "-D_GLIBCXX_USE_CXX11_ABI=0 -fno-rtti"，否则会出现下面这种找不到符号的问题(https://stackoverflow.com/questions/37366291/undefined-symbol-for-self-built-llvm-opt)：

```
$ clang -Xclang -load -Xclang build/skeleton/libSkeletonPass.so program/test.c
error: unable to load plugin 'build/skeleton/libSkeletonPass.so': 'build/skeleton/libSkeletonPass.so:
     undefined symbol: _ZN4llvm23EnableABIBreakingChecksE'
```

如果一切正常，那么在CLion中导入这个工程，点击右上角的编译按钮，就会在cmake-build-debug/keleton下生成我们想要的Pass了，加载试一下：

```
~/Work/llvm-6.0.1.build
 $LLVM_HOME/bin/clang -Xclang -load -Xclang ~/Desktop/llvm-detectDataPass/skeleton/cake-build-debug/skeleton/libSkeletonPass.so /tmp/tmp.c
I saw a function called add!
I saw a function called main!
```

优雅。

开始写Pass

咕咕咕

参考资料

- 官方文档： http://releases.llvm.org/6.0.1/docs/index.html
- LeadroyaL 大佬 blog: http://www.leadroyal.cn/?p=647