文档整理了bazel 工具交叉编译mace 库的方法。参考了网友给出的makefile工程(https://github.com/zhy520xp/mace-makefile-project)，加上自己探索的编译方法终于把mace 使用bazel工具编译出了链接库。[文中

1. 准备工具链
2. 准备编译文件

三、修改编译规则

四、准备测试工程

# 准备工具链

使用git 工具下载mace 源文件，一定要git clone下来的。

参考了https://www.cnblogs.com/jojodru/p/7744630.html ,这里使用hisiv400工具链。

目录结构：

--mace

---Workspace

---arm\_compiler

--BUILD

--CROSSTOOL

--cross\_toolchain\_target\_armv7.BUILD

BUILD文件不用更改，而CROSSTOOL和cross\_toolchain\_target\_armv7.BUILD需要根据编译链的目录更改。另外include path和library可能在编译的时候报错等问题来修改。这个工具链的修改维护比较麻烦，很多人看到这个就直接放弃。**Protobuf 和mace.pb.h这两个相关的头文件是在最后编译的时候遇到错误的时候再添加上去的**

修改好编译链相关文件之后，需要在Workspace 文件中添加下面的语句：

*new\_local\_repository(*

*name ='toolchain\_target\_armv7',*

*path ='/opt/arm-hisiv400-linux',*

*build\_file = 'arm\_compiler/cross\_toolchain\_target\_armv7.BUILD'*

*)*

*个人建议是定义好编译链文件之后找个简单的工程测试一下。*

bazel build //xxxx:target --crosstool\_top=//arm\_compiler:toolchain --cpu=armv7 --copt="-march=armv7-a" --copt="-mfpu=neon" --copt="-funsafe-math-optimizations" \

--copt="-Wno-unused-function" \

--copt="-Wno-sign-compare" \

--copt="-ftree-vectorize" \

--copt="-fomit-frame-pointer" \

--cxxopt="-fPIC" \

--cxxopt="-march=armv7-a" \

--cxxopt="-mfpu=neon"

对于简单的代码工程，使用以下命令进行编译，验证编译链文件是否能够正常工作

# 二、准备编译文件

对于一般的工程，如果没有依赖外部的库和包，定义了编译链文件之后就可以直接编译的。但是对于mace,由于依赖了protobuf 工具，直接编译的过程中会调用交叉编译产生一ARM上的执行文件（js\_embed），但这执行文件实际上是在PC上执行的。跟尴尬的是protobuf是在build过程中从网上下载的，不能在编译过程中修改规则。所以想到的是通过修改mace中的规则使用本地的protobuf。

使用bazel build //:js\_embed 会直接产生一个js\_embed文件，然后把这个执行文件放src/google/protbuf/compiler/js/well\_know\_types目录下，使用命令行产生well\_known\_types\_embed.cc文件。

js\_embed any.js struct.js timestamp.js >well\_known\_types\_embed.cc

*另外需要本地编译和交叉编译protobuf。使用configure就可以了，首先进行本地编译，只需要指定—prefix。交叉编译的时候还需要指定编译链。*

*本地编译以后主要是使用protoc 来把模型文件产生.cc 和 .h 。*

*在mace/proto 目录下执行这个命令*

*Protoc cpp\_out=./ mace.proto*

*以下不需要了*

*~~修改mace的workspace文件，~~*

~~local\_repository(~~

~~name = "com\_google\_protobuf",~~

~~path = "/root/mace/proto/protobuf-3.4.0",~~

~~)~~

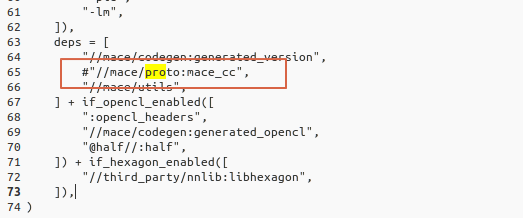
*在mace根目录下执行下面两个语句（可能报错，都是些工具的安装问题，很好解决）*

mkdir -p mace/codegen/version && bash mace/tools/git/gen\_version\_source.sh mace/codegen/version/version.cc

mkdir -p mace/codegen/tuning && python mace/python/tools/binary\_codegen.py --output\_path=mace/codegen/tuning/tuning\_params.cc

# 三修改编译规则

*修改mace core中的文件，注释下面这行*



*注释掉mace/proto中build的全部。*

*此时就可以build libmace.so了*

bazel build //mace/libmace:libmace.so --crosstool\_top=//arm\_compiler:toolchain --cpu=armv7 --copt="-march=armv7-a" --copt="-mfpu=neon" --cxxopt="-mfloat-abi=softfp" --copt="-Wno-unused-function" --copt="-Wno-sign-compare" --copt="-ftree-vectorize" --copt="-fomit-frame-pointer" --cxxopt="-fPIC" --cxxopt="-march=armv7-a" --cxxopt="-mfpu=neon" --cxxopt="-mfloat-abi=softfp" --define neon=true

*如果需要头文件的问题，修改cross\_tool 中添加以下两句*

cxx\_builtin\_include\_directory: "/root/mace/google\_include" #protobuf的头文件目录

cxx\_builtin\_include\_directory: "/root/mace/mace\_v400/mace"#mace.pb.h文件目录

*本方法编译出来的库文件在最后参与链接的时候会报出很多函数没有定义，这个过程卡住了几天，最后发现这些文件在mace.pb.cc中，所以需要把mace.pb.cc另外作为一个库编译出来，也可以作为最后的测试工程源文件。我采取了第一种方法把这个文件也编译出来了一些库。*

*把mace/proto文件夹拷贝出来，建立workspace 和build 文件，workspace 中需要添加交叉编译链的指定，build文件中添加库的编译规则：*

cc\_library(

name = "mace\_proto",

srcs = ["mace.pb.cc"],

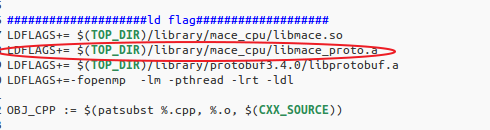
hdrs = ["mace.pb.h"],

)

*这样库的编译就完成了。*

# 四 测试工程编译

*我暂时不知道 模型文件的data怎么产生，这里使用了*<https://github.com/zhy520xp/mace-makefile-project>的test\_cpu工程。编译之前需要更新下对应的库文件和交叉编译链。另外由于我又编译出来了一个mace\_proto相关的库，使用的时候也需要添加上去。



*编译时候加上了—define neon=true 报错*

*说明：硬件上实测时间很长，由于我使用的3516D单核，600MHz。同时我不知道OpenCl，neno和OpenMP 这些选项默认是否打开，bazel中的规则是通过def 和config setting 来控制的，这里还需要再关注研究下。感谢*<https://github.com/zhy520xp/mace-makefile-project原作者> @你身边的小太阳 在我研究的过程中提供的帮助。[编译选项中加速 –define neon=true使能neon,感谢@iihe大神告知,已更新]