在Ubuntu20.04环境下安装GRPC

一、安装GRPC

GRPC可以完成跨语言之间的通信，它是一种协议。我们可以通过在proto文件中定义数据结构和接口函数来实现数据传输。

1、使用cmake --version先查看camke版本

测试环境: cmake version 3.16.3

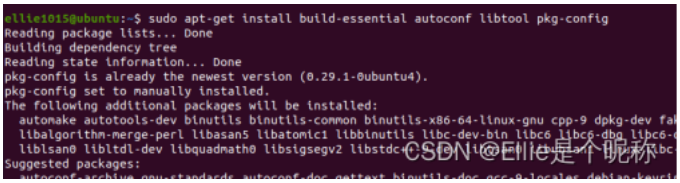
```
可进入https://github.com/Kitware/CMake/releases自行下载，我选择的是 cmake-3.22.3-linux-x86_64.sh
放入虚拟机后更名为cmake-linux.sh(右键更改文件名)

运行cmake-linux.sh文件，通过脚本在本地目录安装更新的cmake版本，--skip-license安装时不会要求用户进行许可协议确认，--prefix=$MY_INSTAL
sudo sh cmake-linux.sh -- --skip-license --prefix=$MY_INSTALL_DIR

查看cmake是否安装成功：cmake --version
```

2、安装构建grpc所需的基本工具

安装build-essential、autoconf 、libtool 、pkg-config：
##build-essential作用是提供编译程序必须软件包的列表信息，这样编译程序才知道头文件和库函数在哪，才会下载依赖的软件包
##autoconf是一种生成shell脚本的工具，可以自动配置软件源代码包
##libtool是自动生成并调用动态库的工具，包装了gcc或者其他编译器
##pkg-config提供功能：检查库的版本号；获得编译预处理参数，如宏定义、头文件的位置； 获得链接参数，如库及依赖的其他库的位置，文件名及其

命令如下：

```
更新package：
sudo apt update

安装所需库：
sudo apt-get install build-essential autoconf libtool pkg-config
```



3、克隆grpc存储库

命令如下：

```
安装git：
sudo apt install git
克隆存储库：
git clone -b v1.28.x https://gitee.com/slrom/grpc
```

4、下载grpc子模块的代码

这里需要注意的是，由于github访问不稳定，所以我们可以修改 grpc目录下的.gitmodules文件中各子模块的地址，便于更好下载子模块的代码。

.gitmodules文件这样修改(find ./grpc -name .gitmodules)：

```
[submodule "third_party/zlib"]
    path = third_party/zlib
    url = https://gitee.com/slrom/zlib.git
    # When using CMake to build, the zlib submodule ends up with a
    # generated file that makes Git consider the submodule dirty. This
    # state can be ignored for day-to-day development on gRPC.
    ignore = dirty
[submodule "third_party/protobuf"]
    path = third_party/protobuf
    url = https://gitee.com/slrom/protobuf.git
```

```
    branch = 3.0.x
[submodule "third_party/gflags"]
    path = third_party/gflags
    url = https://gitee.com/slrom/gflags.git
[submodule "third_party/googletest"]
    path = third_party/googletest
    url = https://gitee.com/slrom/googletest.git
[submodule "third_party/benchmark"]
    path = third_party/benchmark
    url = https://gitee.com/slrom/benchmark.git
[submodule "third_party/boringssl-with-bazel"]
    path = third_party/boringssl-with-bazel
    url = https://gitee.com/slrom/boringssl.git
[submodule "third_party/boringssl"]
    path = third_party/boringssl
    url = https://gitee.com/slrom/boringssl.git
[submodule "third_party/cares/cares"]
    path = third_party/cares/cares
    url = https://gitee.com/slrom/c-ares.git
    branch = cares-1_12_0
[submodule "third_party/bloaty"]
    path = third_party/bloaty
    url = https://gitee.com/slrom/bloaty.git
[submodule "third_party/abseil-cpp"]
    path = third_party/abseil-cpp
    url = https://gitee.com/slrom/abseil-cpp.git
    branch = lts_2020_02_25
[submodule "third_party/envoy-api"]
    path = third_party/envoy-api
    url = https://gitee.com/slrom/data-plane-api.git
[submodule "third_party/googleapis"]
    path = third_party/googleapis
    url = https://gitee.com/slrom/googleapis.git
[submodule "third_party/protoc-gen-validate"]
    path = third_party/protoc-gen-validate
    url = https://gitee.com/slrom/protoc-gen-validate.git
[submodule "third_party/udpa"]
    path = third_party/udpa
    url = https://gitee.com/slrom/udpa.git
[submodule "third_party/libuv"]
    path = third_party/libuv
    url = https://gitee.com/slrom/libuv.git
[submodule "third_party/libcxx"]
    path = third_party/libcxx
    url = https://gitee.com/slrom/libcxx.git
[submodule "third_party/libcxxabi"]
    path = third_party/libcxxabi
    url = https://gitee.com/slrom/libcxxabi.git

```

完成文件修改后，命令如下：

```
下载子模块的代码
到达grpc目录下：
cd grpc
根据.gitmodules文件中的路径访问相应子模块并下载：
git submodule update --init
```

```
ellie1015@ubuntu:~/grpc$ git submodule update --init
Submodule 'third_party/abseil-cpp' (https://gitee.com/slrom/abseil-cpp.git) regi
stered for path 'third_party/abseil-cpp'
Submodule 'third_party/benchmark' (https://gitee.com/slrom/benchmark.git) regist
ered for path 'third_party/benchmark'
Submodule 'third_party/bloaty' (https://gitee.com/slrom/bloaty.git) registered f
or path 'third_party/bloaty'
Submodule 'third_party/boringssl-with-bazel' (https://gitee.com/slrom/boringssl.
git) registered for path 'third_party/boringssl-with-bazel'
Submodule 'third_party/cares/cares' (https://gitee.com/slrom/c-ares.git) registe
red for path 'third_party/cares/cares'
Submodule 'third_party/envoy-api' (https://gitee.com/slrom/data-plane-api.git) r
egistered for path 'third_party/envoy-api'
Submodule 'third_party/gflags' (https://gitee.com/slrom/gflags.git) registered f
or path 'third_party/gflags'
```


5、构建和安装grpc和protocol buffers

命令如下：

```
到达grpc目录下：
cd grpc

若该文件夹不存在则创建文件夹：
mkdir -p cmake/build

到达cmake/build目录下：
cd cmake/build

编译构建依赖库，通过cmake安装grpc，设置-DgRPC_INSTALL=ON，-DgRPC_BUILD_TESTS=OFF；构建安装路径，通过CMAKE_INSTALL_PREFIX变量指定
cmake -DgRPC_INSTALL=ON \
      -DgRPC_BUILD_TESTS=OFF \
      -DCMAKE_INSTALL_PREFIX=/usr/ \
      -DBUILD_SHARED_LIBS=ON \
      ../..
编译：
make -j16

安装：
sudo make install
```

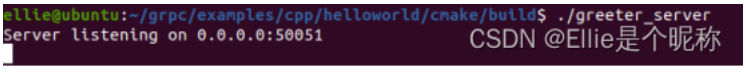
二、测试(依赖grpc++ protobuf pthread)

按照以上步骤，安装就基本完成了，但还是添加一步简单的测试验证下是否安装成功，如需要使用不是GRPC中自带的例子完成测试的话，可以去看看

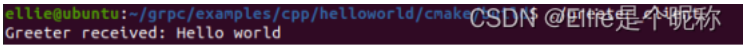
命令如下：

```
cd grpc/examples/cpp/helloworld
mkdir -p cmake/build
cd cmake/build
cmake ../..
make -j16
```


```
ls
./greeter_server
```




```
新开终端
cd grpc/examples/cpp/helloworld/cmake/build
ls
./greeter_client
```



如果到这一步没有出现问题的话，就说明安装成功了。

三、 protoc 编译proto文件

protobuf 有自己的编译器 protoc，可以将 .proto 编译成.cc文件，使之成为一个可以在 C++ 工程中直接使用的类。

1、linux环境

使用protoc命令，由proto文件生成.pb.{h,cc}和.grpc.pb.{h,cc}。
.pb.h：声明生成的消息类的头文件
.pb.cc：包含消息类的实现
.grpc.pb.h：声明生成的服务类的头文件
.grpc.pb.cc：包含服务类的实现

在grpc/examples/cpp目录下新建文件夹，例如service_error，将service_Error.proto放入该新建文件夹中，在该文件夹目录下输入命令：

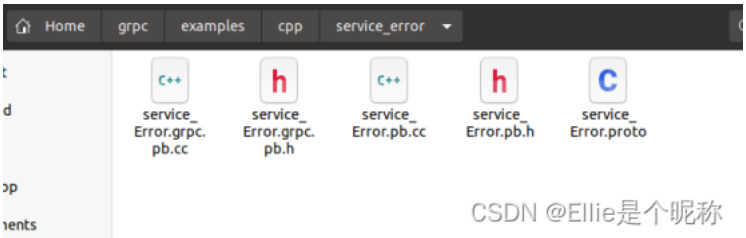
```
protoc --proto_path=. --cpp_out=. ./service_Error.proto
```

protoc：解析proto文件并根据给定的选项生成输出，其命令格式是protoc [OPTION] PROTO_FILES，最后一项是待编译的proto文件的位置
–proto_path：指定要在其中搜索导入(import)的目录，可指定多次，目录将按顺序搜索，若没有指定，则使用当前工作目录。如果proto文件里面import不指定)
–cpp_out：指定.pb.cc和.pb.h文件的输出目录

(生成grpc相关代码)查看grpc_cpp_plugin的所在路径，grpc_cpp_plugin是grpc的protoc插件。whereis命令用于查找文件。

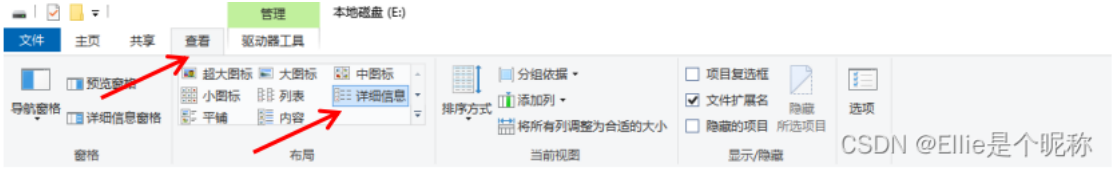
```
whereis grpc_cpp_plugin
```

```
输入命令：
protoc --proto_path=./ --grpc_out=./ --plugin=protoc-gen-grpc=/usr/bin/grpc_cpp_plugin service_Error.proto

#--grpc_out：指定.grpc.pb.cc和.grpc.pb.h文件的输出目录
#--plugin：指定要使用的插件可执行文件。可执行文件的形式是名称=路径，在这种情况下给定的插件名称映射到给定的可执行文件。
```



2、windows环境

新建文件夹放入proto文件，例如service_Error.proto文件。在该文件夹下新建文档，修改后缀为bat。在该文件夹中新建文件夹，名称为gen-cpp，用于
可以通过以下方式显示后缀便于修改：

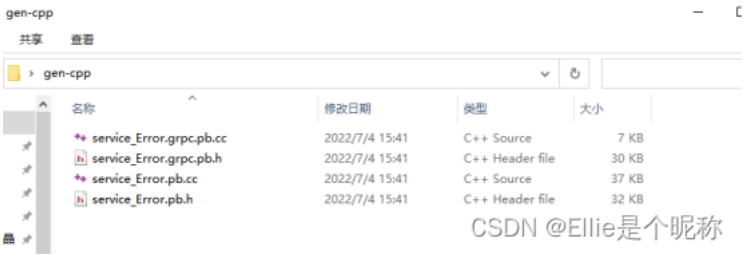



```
查找protoc.exe文件的路径
查找grpc_cpp_plugin.exe文件路径
```

```
选中bat文件，右键“编辑”，输入命令，例如proto.exe路径为：
E:\workspace\ui\thirdparty\bin\protoc.exe
则输入命令（注意修改为自己相应文件所在路径）：

E:\workspace\ui\thirdparty\bin\protoc.exe  -I=./ --cpp_out=./gen-cpp ./service_Error.proto

E:\workspace\ui\thirdparty\bin\protoc.exe -I=./ --grpc_out=./gen-cpp --plugin=protoc-gen-grpc="E:\workspace\ui\thirdpa

pause
```

第一条命令中-I和-cpp_out选项与linux中protoc命令的选项–proto_path和-cpp_out作用相同。
第二条命令中–grpc_out和–plugin选项与linux中–grpc_out和–plugin选项作用相同。
pause 暂停CMD窗口操作

保存后，选中bat文件，右键“以管理员身份运行”，打开gen-cpp文件夹，已生成.pb.{h,cc}和.grpc.pb.{h,cc}文件



以上就是安装GRPC和PROTOC以及编译proto文件的整个过程。