# Win10编译grpc并使用Visual Studio 2022配置grpc

gRPC是Google开发的一种高性能、开源的远程过程调用（RPC）框架。它可以让客户端应用程序像调用本地服务一样轻松地调用远程服务，并提供了多种语言的支持，如C++、Java、Python、Go等。

gRPC使用Protocol Buffers作为数据格式，可以在不同的平台上进行应用程序之间的通信，支持多种编程语言和多种操作系统。它采用基于HTTP/2的协议，提供了高效、快速且可扩展的远程调用功能，并带有负载均衡、认证、监控等功能，方便用户管理和维护分布式系统。

gRPC可用于构建各种类型的分布式应用程序，如微服务、云原生应用程序、大规模Web应用程序、移动应用程序等场景。由于其高性能和可扩展性，越来越多的企业和组织开始采用gRPC来构建他们的应用程序和服务。

# grpc下载：

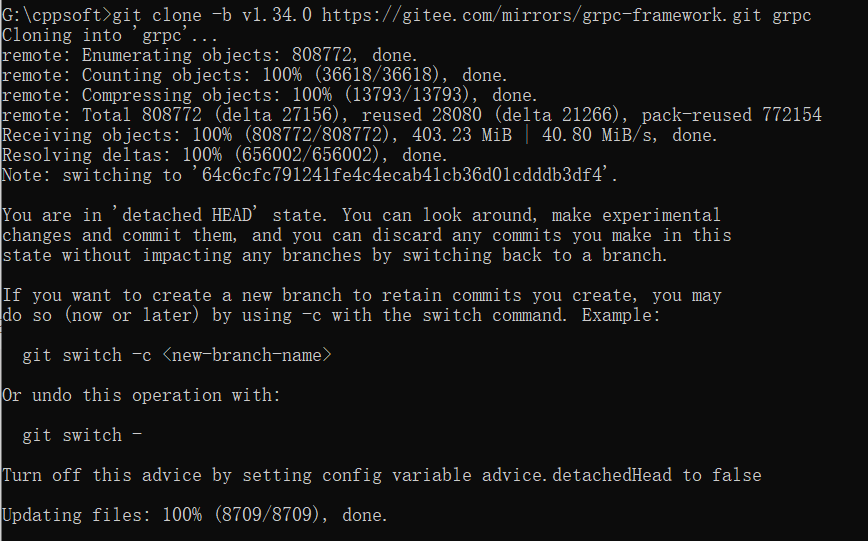
由于国内环境，grpc下载极其困难，grpc项目在github上，源码地址为[https://github.com/grpc/grpc](https://github.com/grpc/grpc" \t "_blank),我们可以通过如下命令克隆grpc源码进行编译

git clone https://github.com/grpc/grpc.git

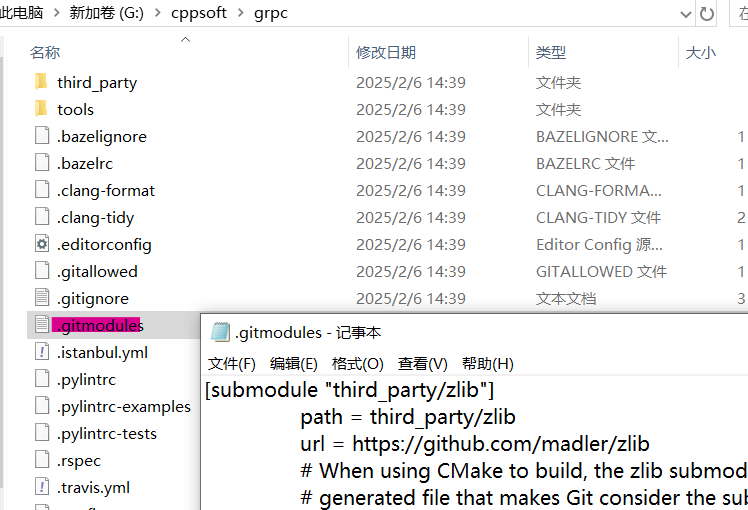
git submodule update --init

但是国内网络环境执行submodule update时会失败,所以可以用国内的代码管理工具gitee进行克隆。 注意,目前在Gitee上只能找到gRPC依赖的部分"官方"镜像仓库,网友提供的镜像仓库较旧,因而只能构造v1.34.0版本.通过上述指令可以将v1.34.0版本的gRPC代码下载到grpc目录. 我们选择一个稳定的分支进行克隆

git clone -b v1.34.0 https://gitee.com/mirrors/grpc-framework.git grpc



克隆之后，我们进入自动生成的grpc文件夹，修改.gitmodules内仓库地址：



替换为：

[submodule "third\_party/zlib"]

path = third\_party/zlib

#url = https://github.com/madler/zlib

url = https://gitee.com/mirrors/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://github.com/google/protobuf.git

url = https://gitee.com/local-grpc/protobuf.git

[submodule "third\_party/googletest"]

path = third\_party/googletest

#url = https://github.com/google/googletest.git

url = https://gitee.com/local-grpc/googletest.git

[submodule "third\_party/benchmark"]

path = third\_party/benchmark

#url = https://github.com/google/benchmark

url = https://gitee.com/mirrors/google-benchmark.git

[submodule "third\_party/boringssl-with-bazel"]

path = third\_party/boringssl-with-bazel

#url = https://github.com/google/boringssl.git

url = https://gitee.com/mirrors/boringssl.git

[submodule "third\_party/re2"]

path = third\_party/re2

#url = https://github.com/google/re2.git

url = https://gitee.com/local-grpc/re2.git

[submodule "third\_party/cares/cares"]

path = third\_party/cares/cares

#url = https://github.com/c-ares/c-ares.git

url = https://gitee.com/mirrors/c-ares.git

branch = cares-1\_12\_0

[submodule "third\_party/bloaty"]

path = third\_party/bloaty

#url = https://github.com/google/bloaty.git

url = https://gitee.com/local-grpc/bloaty.git

[submodule "third\_party/abseil-cpp"]

path = third\_party/abseil-cpp

#url = https://github.com/abseil/abseil-cpp.git

url = https://gitee.com/mirrors/abseil-cpp.git

branch = lts\_2020\_02\_25

[submodule "third\_party/envoy-api"]

path = third\_party/envoy-api

#url = https://github.com/envoyproxy/data-plane-api.git

url = https://gitee.com/local-grpc/data-plane-api.git

[submodule "third\_party/googleapis"]

path = third\_party/googleapis

#url = https://github.com/googleapis/googleapis.git

url = https://gitee.com/mirrors/googleapis.git

[submodule "third\_party/protoc-gen-validate"]

path = third\_party/protoc-gen-validate

#url = https://github.com/envoyproxy/protoc-gen-validate.git

url = https://gitee.com/local-grpc/protoc-gen-validate.git

[submodule "third\_party/udpa"]

path = third\_party/udpa

#url = https://github.com/cncf/udpa.git

url = https://gitee.com/local-grpc/udpa.git

[submodule "third\_party/libuv"]

path = third\_party/libuv

#url = https://github.com/libuv/libuv.git

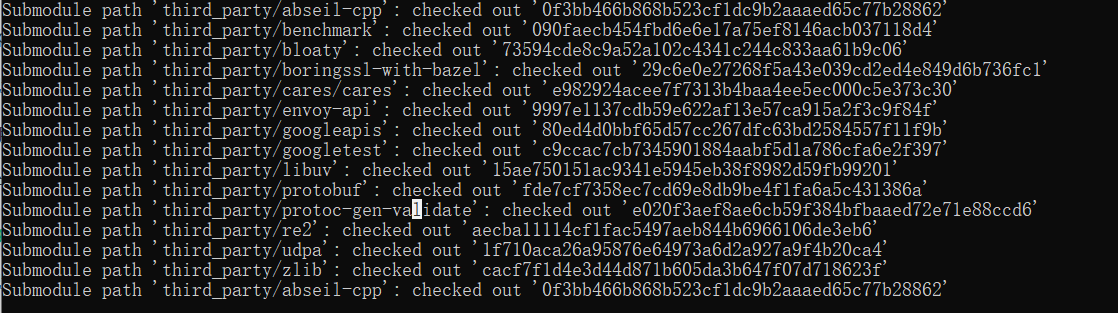
url = <https://gitee.com/mirrors/libuv.git>

**2025年2月6日时，gitee上的有些链接已经不可用，试着替换为GitHub上的链接使用。**

替换完了之后，使用如下命令拉取gRPC所有依赖：

cd grpc

git submodule update --init

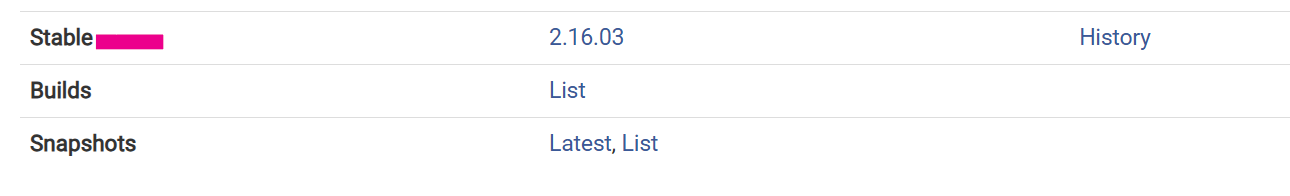


克隆完成。

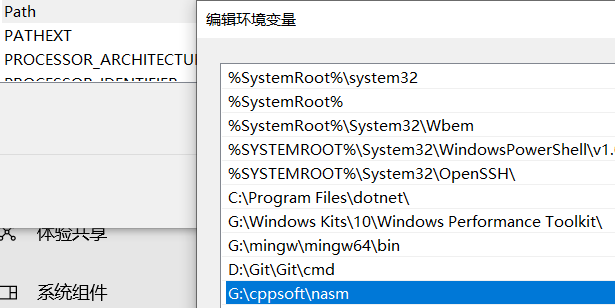
# 编译grpc的前置应用安装：

## NASM下载：

编译grpc需要准备nasm,所以也要下载nasm, 下载地址为[https://www.nasm.us/](https://www.nasm.us/" \t "_blank),点击下载stable版，使用管理员权限安装即可：



记得安装后把安装的路径配置在环境变量里：

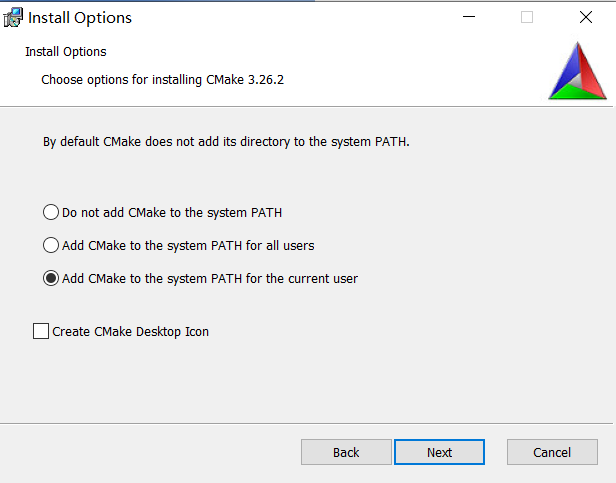


## CMake下载：

为了编译grpc，我们需要下载cmake，cmake是一个跨平台编译工具，在之前我们编译protobuf时已经下载过了，这里再次介绍一下 CMake是一个跨平台的开源构建工具，可以用于自动化构建、测试和打包C++代码。与其他构建工具相比，CMake的优点之一是它的配置文件具有可读性和可维护性，而且支持多种编译器、操作系统和构建工具。

我们要在windows上生成protobuf对应的库，就需要用到cmake。 官方下载地址<https://cmake.org/download/>， 选择Windows Source下载。 由于官方下载较慢，可以去llfc的网盘下载 <https://pan.baidu.com/s/1Yg9Usdc3T-CYhyr9GiePCw?pwd=ng6x>

验证码ng6x



## Go安装：

windows环境下下载go，地址[https://studygolang.com/dl](https://studygolang.com/dl" \t "_blank)。 选择我们对应系统的安装包下载即可。一路下载安装，最后go会自动写入我们的环境变量。 Linux可以选择手动安装二进制包再配置环境变量。 安装好后，因为go的库包默认在外网，下载很慢，所以要切换代理：

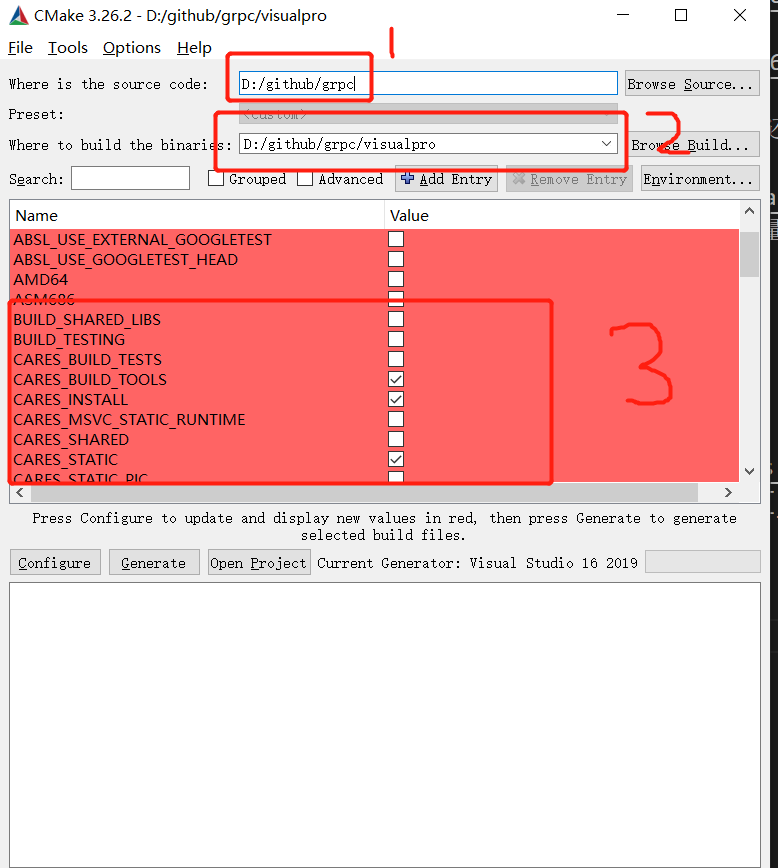
go env -w GOPROXY=https://goproxy.cn,direct # 设置代理，国内网络开启

## Perl安装：

Perl安装可以去官网下载[https://www.activestate.com/products/perl/](https://www.activestate.com/products/perl/" \t "_blank)，并按照提示一步步安装，也可以去llfc的网盘下载，直接安装，网盘地址<https://pan.baidu.com/s/1i3GLKAp> 一路next安装完成。

# 使用CMake生成grpc：

启动cmake，接下来我们需要用CMake编译grpc来生成我们需要的visual studio 项目：

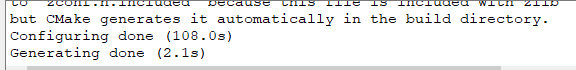


1是grpc源码目录

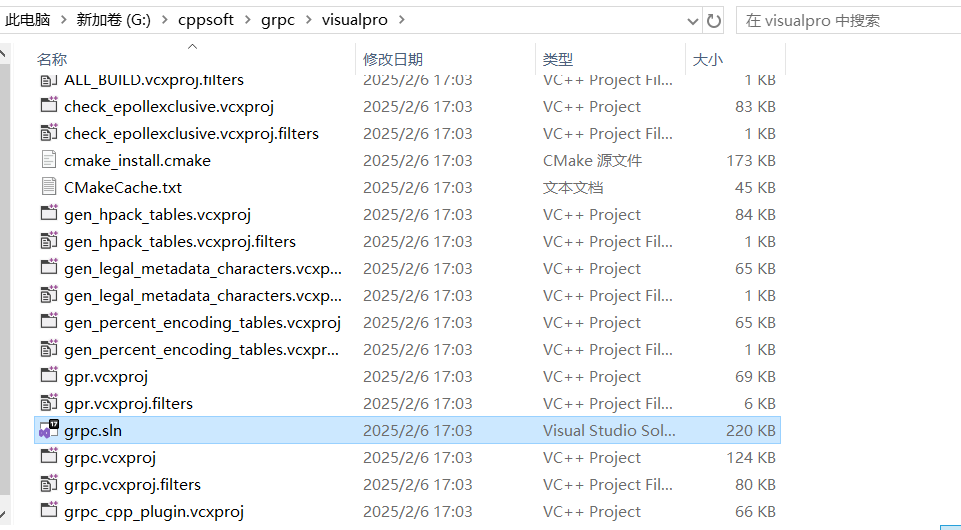
2是在grpc下新建一个visualpro文件夹存放生成的visual studio项目目录

3 点击configure先择vs2022加载要生成哪些项目。

接下来点击generate生成项目：

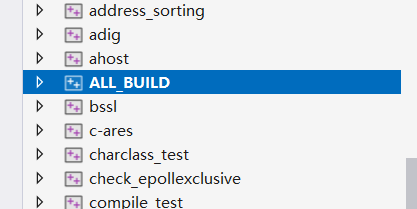


打开之前新建的visualpro文件夹，就能找到grpc.sln文件，双击打开：



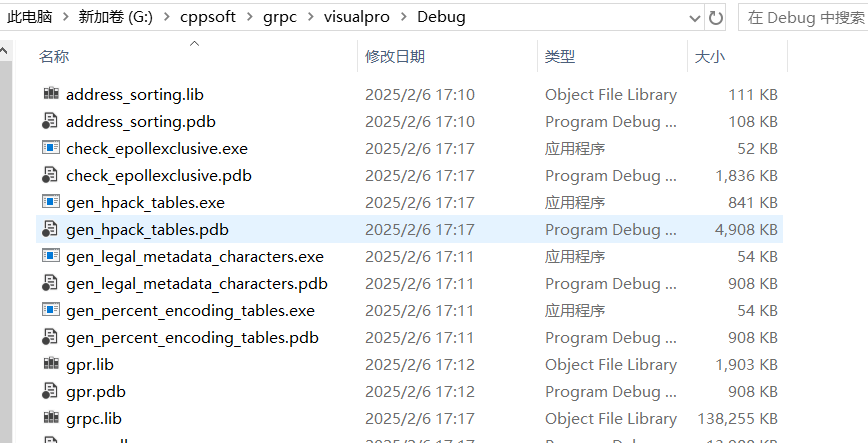
# 使用Visual Studio 2022编译生成grpc：

右键ALL\_BUILD选择重新生成选项：



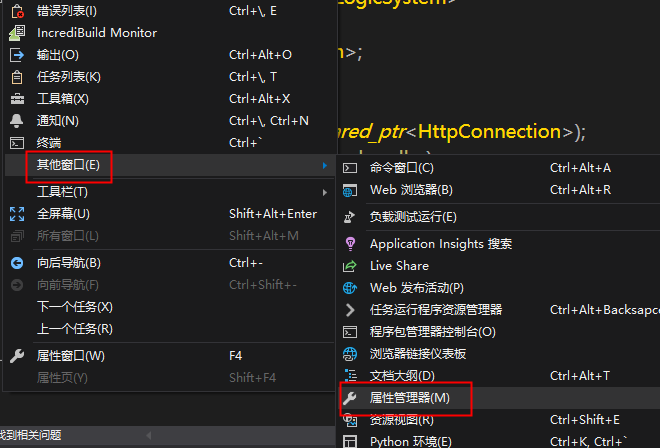


生成成功！回到visualpro文件夹，生成了Debug文件夹(因为之前选择的Debug生成模式)：

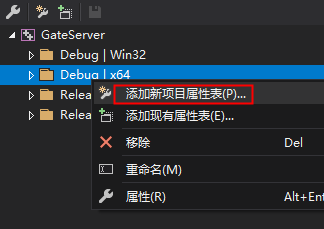


# 使用Visual Studio 2022配置使用grpc：

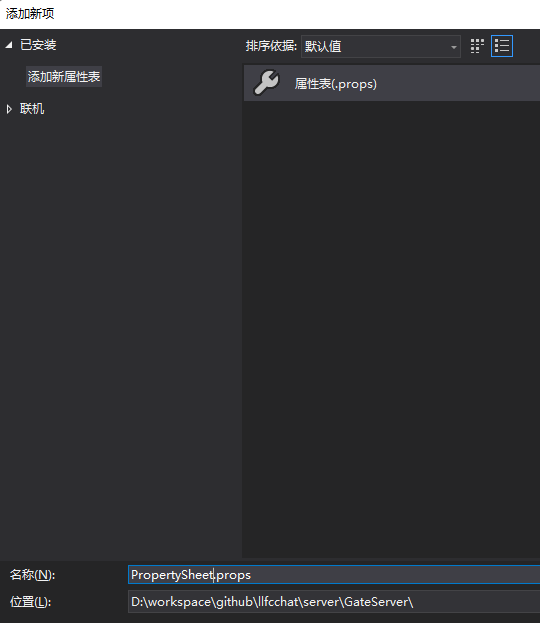
属性管理器：因为要配置的太多了，不能每次有了新项目就配置。所以我们新建一个属性管理器，每次使用时复制到项目中即可，视图里选择其他窗口，再选择属性管理器：



我们选择要配置的类型，我选择Debug 64位的配置，添加新项目属性表：



选择创建属性的名字：



双击新建的属性，接下来配置grpc头文件包含目录，C++ 常规-> 附加包含目录添加如下：

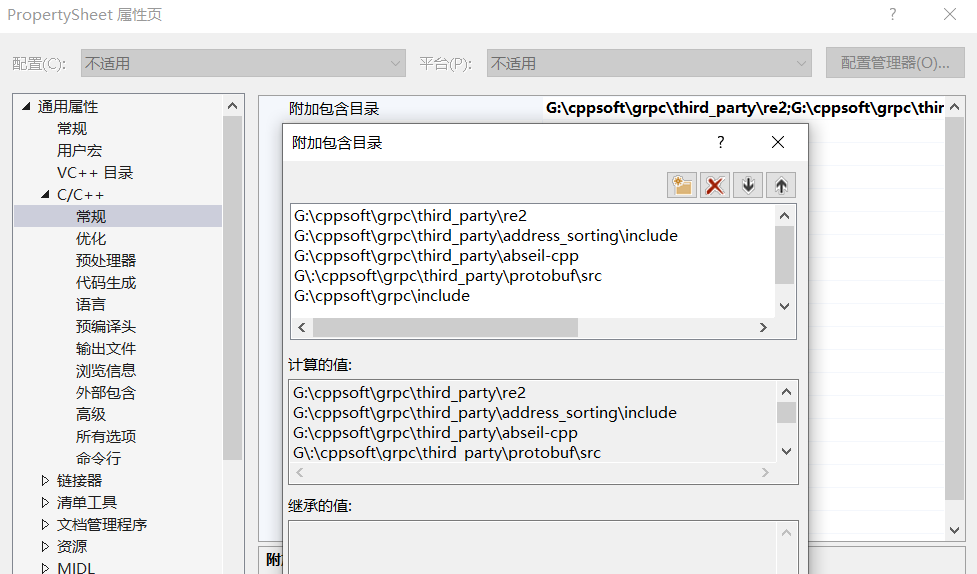
G:\cppsoft\grpc\third\_party\re2

G:\cppsoft\grpc\third\_party\address\_sorting\include

G:\cppsoft\grpc\third\_party\abseil-cpp

G:\cppsoft\grpc\third\_party\protobuf\src

G:\cppsoft\grpc\include



接下来配置库路径, 在链接器常规选项下，点击附加库目录，添加我们需要的库目录：

G:\cppsoft\grpc\visualpro\third\_party\re2\Debug

G:\cppsoft\grpc\visualpro\third\_party\abseil-cpp\absl\types\Debug

G:\cppsoft\grpc\visualpro\third\_party\abseil-cpp\absl\synchronization\Debug

G:\cppsoft\grpc\visualpro\third\_party\abseil-cpp\absl\status\Debug

G:\cppsoft\grpc\visualpro\third\_party\abseil-cpp\absl\random\Debug

G:\cppsoft\grpc\visualpro\third\_party\abseil-cpp\absl\flags\Debug

G:\cppsoft\grpc\visualpro\third\_party\abseil-cpp\absl\debugging\Debug

G:\cppsoft\grpc\visualpro\third\_party\abseil-cpp\absl\container\Debug

G:\cppsoft\grpc\visualpro\third\_party\abseil-cpp\absl\hash\Debug

G:\cppsoft\grpc\visualpro\third\_party\boringssl-with-bazel\Debug

G:\cppsoft\grpc\visualpro\third\_party\abseil-cpp\absl\numeric\Debug

G:\cppsoft\grpc\visualpro\third\_party\abseil-cpp\absl\time\Debug

G:\cppsoft\grpc\visualpro\third\_party\abseil-cpp\absl\base\Debug

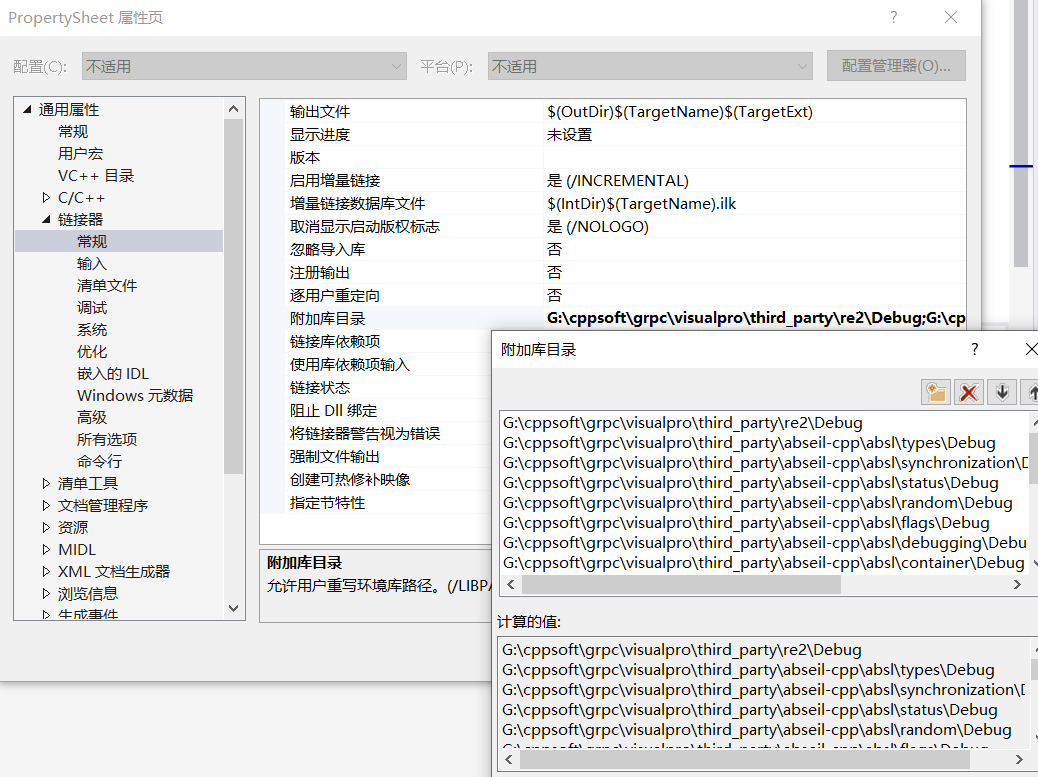
G:\cppsoft\grpc\visualpro\third\_party\abseil-cpp\absl\strings\Debug

G:\cppsoft\grpc\visualpro\third\_party\protobuf\Debug

G:\cppsoft\grpc\visualpro\third\_party\zlib\Debug

G:\cppsoft\grpc\visualpro\Debug

G:\cppsoft\grpc\visualpro\third\_party\cares\cares\lib\Debug



在链接器->输入->附加依赖项中添加:

libprotobufd.lib

gpr.lib

grpc.lib

grpc++.lib

grpc++\_reflection.lib

address\_sorting.lib

ws2\_32.lib

cares.lib

zlibstaticd.lib

upb.lib

ssl.lib

crypto.lib

absl\_bad\_any\_cast\_impl.lib

absl\_bad\_optional\_access.lib

absl\_bad\_variant\_access.lib

absl\_base.lib

absl\_city.lib

absl\_civil\_time.lib

absl\_cord.lib

absl\_debugging\_internal.lib

absl\_demangle\_internal.lib

absl\_examine\_stack.lib

absl\_exponential\_biased.lib

absl\_failure\_signal\_handler.lib

absl\_flags.lib

absl\_flags\_config.lib

absl\_flags\_internal.lib

absl\_flags\_marshalling.lib

absl\_flags\_parse.lib

absl\_flags\_program\_name.lib

absl\_flags\_usage.lib

absl\_flags\_usage\_internal.lib

absl\_graphcycles\_internal.lib

absl\_hash.lib

absl\_hashtablez\_sampler.lib

absl\_int128.lib

absl\_leak\_check.lib

absl\_leak\_check\_disable.lib

absl\_log\_severity.lib

absl\_malloc\_internal.lib

absl\_periodic\_sampler.lib

absl\_random\_distributions.lib

absl\_random\_internal\_distribution\_test\_util.lib

absl\_random\_internal\_pool\_urbg.lib

absl\_random\_internal\_randen.lib

absl\_random\_internal\_randen\_hwaes.lib

absl\_random\_internal\_randen\_hwaes\_impl.lib

absl\_random\_internal\_randen\_slow.lib

absl\_random\_internal\_seed\_material.lib

absl\_random\_seed\_gen\_exception.lib

absl\_random\_seed\_sequences.lib

absl\_raw\_hash\_set.lib

absl\_raw\_logging\_internal.lib

absl\_scoped\_set\_env.lib

absl\_spinlock\_wait.lib

absl\_stacktrace.lib

absl\_status.lib

absl\_strings.lib

absl\_strings\_internal.lib

absl\_str\_format\_internal.lib

absl\_symbolize.lib

absl\_synchronization.lib

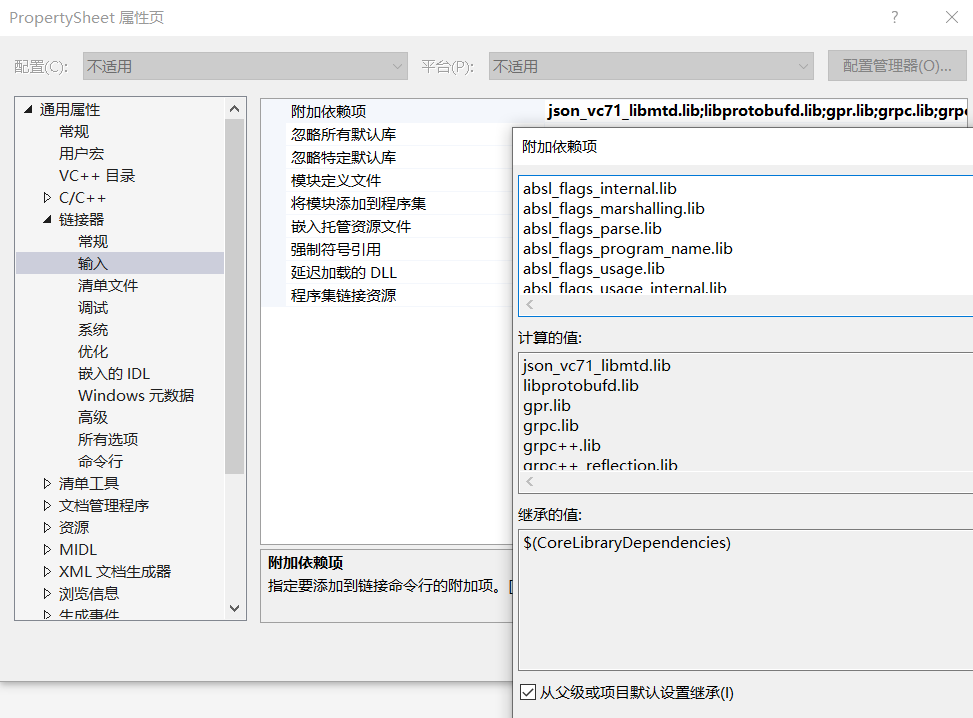
absl\_throw\_delegate.lib

absl\_time.lib

absl\_time\_zone.lib

absl\_statusor.lib

re2.lib



之后点击保存会看到项目目录下生成了PropertySheet.props文件.