# Protobuf编译使用说明

## 编译环境说明

操作系统：windows 7 64位

编译工具：cmake-3.9.0-rc3-win64-x64

IDE：Visual Studio 2015

语言：C++

## 编译Protobuf步骤

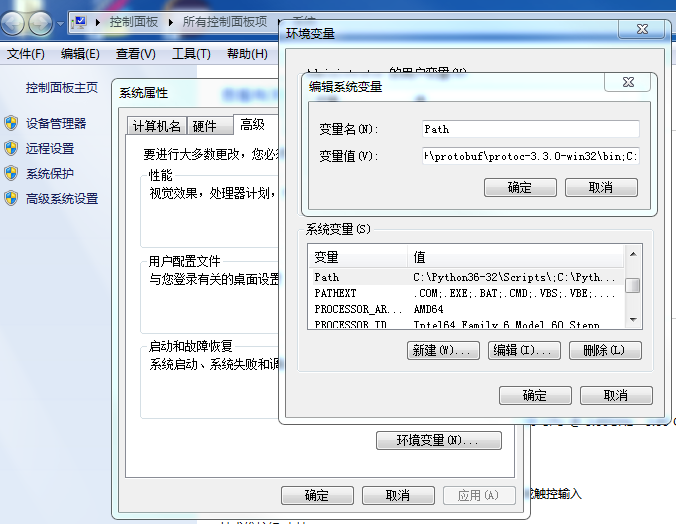
1. 下载protobuf-cpp-3.3.0.tar.gz（从群共享或者从以下网址自行下载https://github.com/google/protobuf/releases）并解压到任意目录
2. 下载安装cmake-3.9.0-rc3-win64-x64.msi（从群共享或者从以下网址自行下载网址https://cmake.org/download/）编译软件
3. 按照下列网址的说明,进行编译

https://github.com/google/protobuf/blob/master/cmake/README.md

**注意**：由于我们已经下载了源代码，所有不需要前面的那些步骤，只需要执行CMake Configuration（其中的创建Visual Studio解决方案文件也不需要执行）以及Compiling两节中的步骤。Testing及以下的步骤也不需要执行。

执行完上述步骤，在debug目录下就得到libprotobufd.lib、 libprotobuf-lited.lib、libprotocd.lib、protoc.exe几个文件，在release目录下就得到libprotobuf.lib、libprotobuf-lite.lib、libprotoc.lib、protoc.exe几个文件。

1. 将protoc.exe所在文件夹添加到环境变量里面，如下所示：



## 安装Visual Studio protobuf编辑插件

下载Protobuf.vsix（从群共享或者从以下网址自行下载https://visualstudiogallery.msdn.microsoft.com/4bc0f38c-b058-4e05-ae38-155e053c19c5/file/118353/22/Protobuf.vsix），双击运行即可。安装完成后重启Visual Studio。

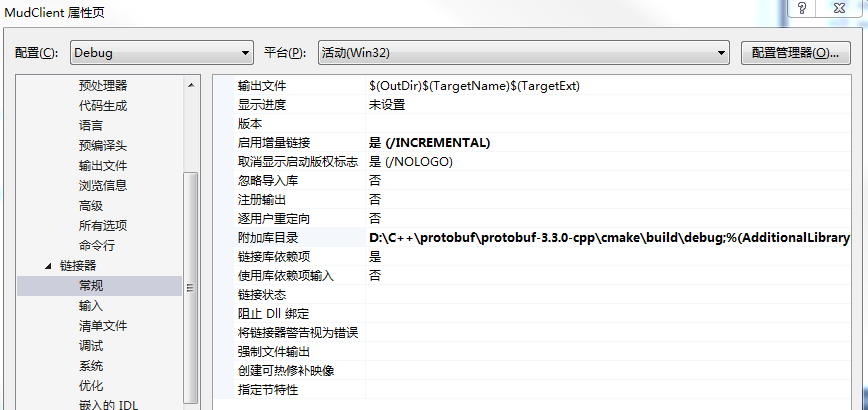
## 测试步骤

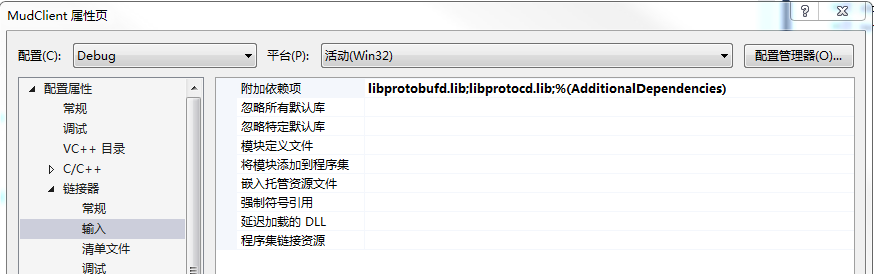
下面是使用Protobuf的例子。

1. 创建一个项目，打开项目属性对话框，在其中C/C++ ->常规页面的附加包含目录中添加protobuf头文件目录（根据各自存放protobuf源码的实际路径进行修改），如下所示：

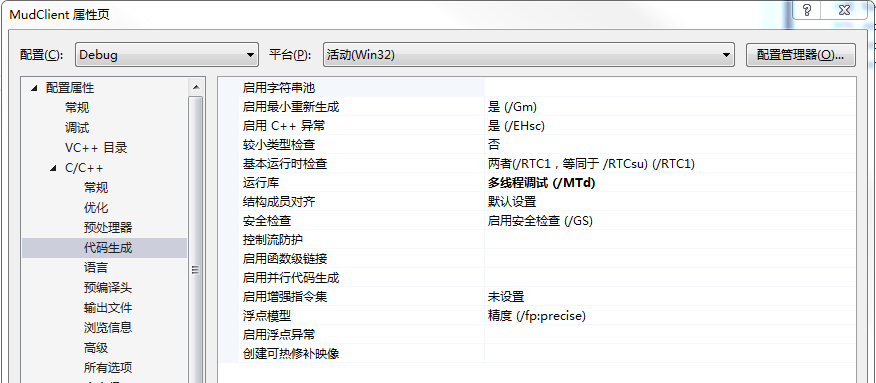


1. 中添加对protobuf库文件的包含（根据各自protobuf编译后lib文件的实际路径进行修改）。**注意：**为了避免MSVC debug版本和release版本运行时库之间的冲突，在编译应用程序的调试版本时，需要链接到libprotobufd.lib调试版本。类似地，release版本应该链接到libprotobuf.lib库。截图如下：





1. 修改运行库设置为/MTd，如下图所示：



1. 按照官方提供的案例，编写addressbook.proto文件。

syntax = "proto2";

package tutorial;

message Person {

required string name = 1;

required int32 id = 2;

optional string email = 3;

enum PhoneType {

MOBILE = 0;

HOME = 1;

WORK = 2;

}

message PhoneNumber {

required string number = 1;

optional PhoneType type = 2 [default = HOME];

}

repeated PhoneNumber phones = 4;

}

message AddressBook {

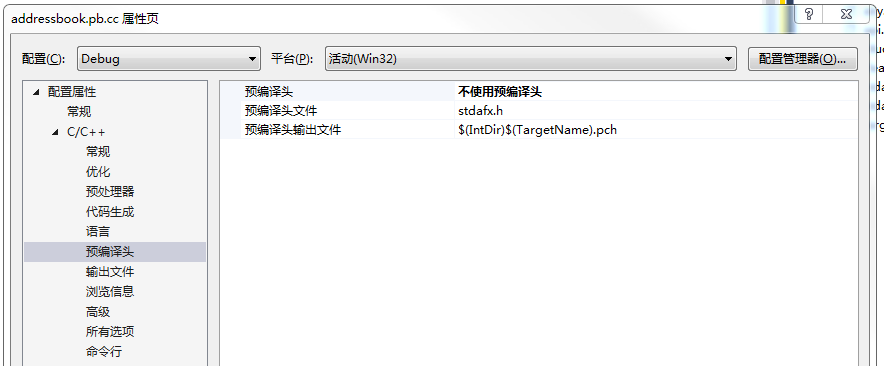
repeated Person people = 1;

}

1. 在dos命令行窗口，切换到addressbook.proto文件所在目录，然后运行以下命令，生成addressbook.pb.cc和addressbook.pb.h文件

protoc -I=./ --cpp\_out=./ ./addressbook.proto

1. 将addressbook.pb.cc和addressbook.pb.h文件添加到vs工程里面。右击addressbook.pb.cc选择属性，在预编译头页面，设置不使用预编译头，如下图所示：



1. 编写本机序列化和反序列化测试代码

#include <iostream>

#include <fstream>

#include <string>

#include "addressbook.pb.h"

using namespace std;

// This function fills in a Person message based on user input.

void PromptForAddress(tutorial::Person\* person) {

*cout* << "Enter person ID number: ";

int id;

*cin* >> id;

person->set\_id(id);

*cin*.*ignore*(256, '\n');

*cout* << "Enter name: ";

*getline*(*cin*, \*person->mutable\_name());

*cout* << "Enter email address (blank for none): ";

*string* email;

*getline*(*cin*, email);

if (!email.*empty*()) {

person->set\_email(email);

}

while (true) {

*cout* << "Enter a phone number (or leave blank to finish): ";

*string* number;

*getline*(*cin*, number);

if (number.*empty*()) {

break;

}

tutorial::Person::PhoneNumber\* phone\_number = person->add\_phones();

phone\_number->set\_number(number);

*cout* << "Is this a mobile, home, or work phone? ";

*string* type;

*getline*(*cin*, type);

if (type == "mobile") {

phone\_number->set\_type(tutorial::Person::MOBILE);

}

else if (type == "home") {

phone\_number->set\_type(tutorial::Person::HOME);

}

else if (type == "work") {

phone\_number->set\_type(tutorial::Person::WORK);

}

else {

*cout* << "Unknown phone type. Using default." << *endl*;

}

}

}

// Iterates though all people in the AddressBook and prints info about them.

void ListPeople(const tutorial::AddressBook& address\_book) {

for (int i = 0; i < address\_book.people\_size(); i++) {

const tutorial::Person& person = address\_book.people(i);

*cout* << "Person ID: " << person.id() << *endl*;

*cout* << " Name: " << person.name() << *endl*;

if (person.has\_email()) {

*cout* << " E-mail address: " << person.email() << *endl*;

}

for (int j = 0; j < person.phones\_size(); j++) {

const tutorial::Person::PhoneNumber& phone\_number = person.phones(j);

switch (phone\_number.type()) {

case tutorial::Person::MOBILE:

*cout* << " Mobile phone #: ";

break;

case tutorial::Person::HOME:

*cout* << " Home phone #: ";

break;

case tutorial::Person::WORK:

*cout* << " Work phone #: ";

break;

}

*cout* << phone\_number.number() << *endl*;

}

}

}

// Main function: Reads the entire address book from a file,

// adds one person based on user input, then writes it back out to the same

// file.

int main(int argc, char\* argv[]) {

// Verify that the version of the library that we linked against is

// compatible with the version of the headers we compiled against.

GOOGLE\_PROTOBUF\_VERIFY\_VERSION;

tutorial::AddressBook address\_book;

// Read the existing address book.

*fstream* input("address\_book", *ios*::*in* | *ios*::*binary*);

if (!input) {

*cout* << ": File address\_book not found. Creating a new file." << *endl*;

}

else if (!address\_book.ParseFromIstream(&input)) {

*cerr* << "Failed to parse address book." << *endl*;

return -1;

}

// Add an address.

PromptForAddress(address\_book.add\_people());

// Write the new address book back to disk.

*fstream* output("address\_book", *ios*::*out* | *ios*::*trunc* | *ios*::*binary*);

if (!address\_book.SerializeToOstream(&output)) {

*cerr* << "Failed to write address book." << *endl*;

return -1;

}

// Optional: Delete all global objects allocated by libprotobuf.

google::protobuf::ShutdownProtobufLibrary();

return 0;

}

1. 编写反序列化测试代码，将上面的main函数的内容替换为下面的代码

// Main function: Reads the entire address book from a file and prints all

// the information inside.

int main(int argc, char\* argv[]) {

// Verify that the version of the library that we linked against is

// compatible with the version of the headers we compiled against.

GOOGLE\_PROTOBUF\_VERIFY\_VERSION;

tutorial::AddressBook address\_book;

// Read the existing address book.

*fstream* input("address\_book", *ios*::*in* | *ios*::*binary*);

if (!address\_book.ParseFromIstream(&input)) {

*cerr* << "Failed to parse address book." << *endl*;

return -1;

}

ListPeople(address\_book);

// Optional: Delete all global objects allocated by libprotobuf.

google::protobuf::ShutdownProtobufLibrary();

*getchar*();

return 0;

}