# Protocol Buffer Basics: C++

C++ 程序员如何使用 protocol buffers 做一个基本介绍。通过创建一个简单的示例应用程序，它向你展示：

* 如何在一个 .proto 文件中定义 message
* 如何使用 protocol buffer 编译器
* 解析和序列化
* 如何使用 C++ protocol buffer 的 API 读写 message

## 1. 定义你的 protocol 格式

要创建地址簿应用程序，从 .proto 文件开始。.proto 文件中的定义很简单：为要序列化的每个数据结构添加 message 定义，然后为 message 中的每个字段指定名称和类型。下面就是定义相关 message 的 .proto 文件，addressbook.proto。

//这个字段必须有

syntax = "proto2";

//syntax = “proto3”;

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;

}

.proto 文件以 package 声明开头，这有助于防止不同项目之间的命名冲突。在 C++ 中，生成的类将放在与包名匹配的 namespace （命名空间）中。

message 只是包含一组类型字段的集合。许多标准的简单数据类型都可用作字段类型，包括 bool、int32、float、double 和 string。可以使用其他 message 类型作为字段类型在消息中添加更多结构 - 可以定义嵌套在其他 message 中的 message 类型 -​​ 可以定义枚举类型 -

每个元素上的 "=1"，"=2" 标记表示该字段在二进制编码中使用的唯一 “标记”。标签号 1-15 比起更大数字需要少一个字节进行编码，因此以此进行优化，你可以决定将这些标签用于常用或重复的元素，将标记 16 和更高的标记留给不太常用的可选元素。repeated 字段中的每个元素都需要重新编码 Tag，

必须使用以下修饰符之一注释每个字段：

* required: 必须提供该字段的值，否则该消息将被视为“未初始化”。如果是在调试模式下编译 libprotobuf，则序列化一个未初始化的 message 将将导致断言失败。在优化的构建中，将跳过检查并始终写入消息。但是，解析未初始化的消息将始终失败（通过从解析方法返回 false）。除此之外，required 字段的行为与 optional 字段完全相同。

【注释】 proto3 定义中该字段请已取消了。

* optional: 可以设置也可以不设置该字段。如果未设置可选字段值，则使用默认值。对于简单类型，你可以指定自己的默认值，就像我们在示例中为电话号码类型所做的那样。否则，使用系统默认值：数字类型为 0，字符串为空字符串，bools 为 false。对于嵌入 message，默认值始终是消息的 “默认实例” 或 “原型”，其中没有设置任何字段。调用访问器以获取尚未显式设置的 optional（或 required）字段的值始终返回该字段的默认值。
* repeated: 该字段可以重复任意次数（包括零次）。重复值的顺序将保留在 protocol buffer 中。可以将 repeated 字段视为动态大小的数组。

## 2 .编译你的 Protocol Buffers

有了一个 .proto 文件，下一件事就是生成你需要读写AddressBook（以及 Person 和 PhoneNumber ） message 所需的类。

命令如下：

proto -I=$SRC\_DIR --cpp\_out=$DST\_DIR $SRC\_DIR/addressbook.proto

这将在指定的目标目录中生成以下文件：

* addressbook.pb.h： 类声明的头文件
* addressbook.pb.cc：类实现

## 3 .解析和序列化

每个 protocol buffer 类都有使用 protocol buffer [二进制格式](https://www.jianshu.com/p/82ff31c6adc6) 读写所选类型 message 的方法。包括：

* bool SerializeToString(string\* output) const;:序列化消息并将字节存储在给定的字符串中。请注意，字节是二进制的，而不是文本;我们只使用 string 类作为方便的容器。
* bool ParseFromString(const string& data);: 解析给定字符串到 message
* bool SerializeToOstream(ostream\* output) const;: 将 message 写入给定的 C++ 的 ostream
* bool ParseFromIstream(istream\* input);: 解析给定 C++ istream 到 message

## 4 使用 C++ protocol buffer 的 API 读写 message的样例

## 4.1 写入一个 Message

根据用户输入向其添加一个新 Person，并将新的 AddressBook 重新写回文件。其中直接调用或引用 protocol 编译器生成的代码部分将高亮显示。

#include <iostream>

#include <fstream>

#include <string>

#include "addressbook.pb.h"//改文件是protoc 命令生成的

using namespace std;

//using namespace tutorial; //可以增加，消除代码中 tutorial::

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());//从命令行夺取，不要使用cin， 会分割字段

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;

}

}

}

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

GOOGLE\_PROTOBUF\_VERIFY\_VERSION;

if (argc != 2) {

cerr << "Usage: " << argv[0] << " ADDRESS\_BOOK\_FILE" << endl;

return -1;

}

tutorial::AddressBook address\_book;

{

// Read the existing address book.

fstream input(argv[1], ios::in | ios::binary);

if (!input) {

cout << argv[1] << ": File 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());

{

.

fstream output(argv[1], ios::out | ios::trunc | ios::binary);

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

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

return -1;

}

}

google::protobuf::ShutdownProtobufLibrary();

return 0;

}

## 4.2 读取一个 Message

读取上面示例创建的文件并打印其中的所有信息。

#include <iostream>

#include <fstream>

#include <string>

#include "addressbook.pb.h"

using namespace std;

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;

}

}

}

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

GOOGLE\_PROTOBUF\_VERIFY\_VERSION;

if (argc != 2) {

cerr << "Usage: " << argv[0] << " ADDRESS\_BOOK\_FILE" << endl;

return -1;

}

tutorial::AddressBook address\_book;

{

fstream input(argv[1], ios::in | ios::binary);

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

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

return -1;

}

}

ListPeople(address\_book);

google::protobuf::ShutdownProtobufLibrary();

return 0;

}