1、在.proto文件中定义消息格式

2、使用protobuf编译器

3、使用c++ api来读写消息

0、为何使用protobuf?

1、原始内存数据结构，可以以二进制方式sent/saved.这种方式需要相同的内存布局和字节序。

2、以ad-hoc方式将数据项编码成一个简单字符串----比如，将4个int类型编码成"12:3:-23:67"。这种方式简灵活。适用于简单数据。

3、将数据序列化为XML。这种方式很流行，因为xml可读性好，编码解码方便，性能也好。仅仅XML dom树比较复杂。

protobuf可以很好的解决上述问题。你编写一个.proto文件来描述数据结构。protobuf编译器使用它创建一个类，使用二进制方式自动编码/解码该数据结构。生成的类提供getter/setter方法。

最重要的是，protobuf支持在此基础上进行格式扩展。

[示例](http://code.google.com/p/protobuf/downloads/)

1、定义协议格式

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 phone = 4;

}

message AddressBook {

repeated Person person = 1;

}

该结构与c++或java很像.

.proto文件以包声明开始，防止名字冲突。

简单类型：bool, int32, float, double, string.

其它类型：如上述的Person, PhoneNumber

类型可以嵌套。

“=1”， “=2”标识唯一“tag”.tag数1-15需要至少一个字节。

required: 必须设置它的值

optional: 可以设置，也可以不设置它的值

repeated: 可以认为是动态分配的数组

google工程师认为使用required威害更大， 他们更喜欢使用optional, repeated.

2、编译你的协议

运行protoc 来生成c++文件：

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

生成的文件为：

addressbook.pb.h,

addressbook.pb.cc

3、protobuf API

生成的文件中有如下方法：

// name

inline bool has\_name() const;

inline void clear\_name();

inline const ::std::string& name() const;

inline void set\_name(const ::std::string& value);

inline void set\_name(const char\* value);

inline ::std::string\* mutable\_name();

// id

inline bool has\_id() const;

inline void clear\_id();

inline int32\_t id() const;

inline void set\_id(int32\_t value);

// email

inline bool has\_email() const;

inline void clear\_email();

inline const ::std::string& email() const;

inline void set\_email(const ::std::string& value);

inline void set\_email(const char\* value);

inline ::std::string\* mutable\_email();

// phone

inline int phone\_size() const;

inline void clear\_phone();

inline const ::google::protobuf::[RepeatedPtrField](https://developers.google.com/protocol-buffers/docs/reference/cpp/google.protobuf.repeated_field.html?hl=zh-CN#RepeatedPtrField)< ::tutorial::Person\_PhoneNumber >& phone() const;

inline ::google::protobuf::[RepeatedPtrField](https://developers.google.com/protocol-buffers/docs/reference/cpp/google.protobuf.repeated_field.html?hl=zh-CN#RepeatedPtrField)< ::tutorial::Person\_PhoneNumber >\* mutable\_phone();

inline const ::tutorial::Person\_PhoneNumber& phone(int index) const;

inline ::tutorial::Person\_PhoneNumber\* mutable\_phone(int index);

inline ::tutorial::Person\_PhoneNumber\* add\_phone();

4、枚举与嵌套类

生成的代码包含一个PhoneType枚举。Person::PhoneType, Person:MOBILE, Person::HOME, Person:WORK.

编译器生成的嵌套类称为Person::PhoneNumber. 实际生成类为Person\_PhoneNumber.

5、标准方法

bool IsInitialized() const: 确认required字段是否被设置

string DebugString() const: 返回消息的可读表示，用于调试

void CopyFrom(const Person& from): 使用给定消息值copy

void Clear(): 清除所有元素为空状态

6、解析与序列化

bool SerializeToString(string\* output) const: 序列化消息，将存储字节的以string方式输出。注意字节是二进制，而非文本；

bool ParseFromString(const string& data): 解析给定的string

bool SerializeToOstream(ostream\* output) const: 写消息给定的c++ ostream中

bool ParseFromIstream(istream\* input): 从给定的c++ istream中解析出消息

7、protobuf和 oo设计

不要继承生成类并在此基础上添加相应的行为

8、写消息

示例：它从一个文件中读取AddressBook，基于io添加一个新的Person，并将新的AddressBook写回文件。

#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\_phone();

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;

}

}

}

// 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;

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\_person());

{

// Write the new address book back to disk.

fstream output(argv[1], 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;

}

注意使用GOOGLE\_PROTOBUF\_VERIFY\_VERSION宏。每一个.pb.cc文件在启动时都将自动调用该宏。

注意在程序结尾处调用ShutdownProtobufLibrary()。

9、读消息

#include <iostream>

#include <fstream>

#include <string>

#include "addressbook.pb.h"

using namespace std;

// 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.person\_size(); i++) {

const tutorial::Person& person = address\_book.person(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.phone\_size(); j++) {

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

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 (!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();

return 0;

}

10、扩展protobuf

如果希望向后兼容，必须遵循：

a、不必更改tag数

b、不必添加或删除任何required字段

c、可以删除optional或repeated字段

d、可以添加新的optional或repeated字段，但你必须使用新的tag数。

11、优化

c++的protobuf库，已经极大地优化了。合理使用可以改善性能。

a、如果可能，复用message对象。

b、关于多线程的内存分配器

12、高级用法

protobuf的消息类的一个关键特性是，反射(reflection)。可以使用xml或json来实现。[参考](https://developers.google.com/protocol-buffers/docs/reference/cpp/google.protobuf.message.html?hl=zh-CN#Message.Reflection)。

================================================================

常见问题：

1、undefined reference to `pthread\_once'

使用-lpthread: