protobuf

初识protobuf

protocol buffers 是一种语言无关、平台无关、可扩展的序列化结构数据的方法，它可用于（数据）通信协议、数据存储等。

Protocol Buffers 是一种灵活，高效，自动化机制的结构数据序列化方法－可类比 XML，但是比 XML 更小（3 ~ 10倍）、更快（20 ~ 100倍）、更为简单。

你可以定义数据的结构，然后使用特殊生成的源代码轻松的在各种数据流中使用各种语言进行编写和读取结构数据。你甚至可以更新数据结构，而不破坏由旧数据结构编译的已部署程序。

简单来讲， ProtoBuf 是结构数据序列化[1] 方法，可简单类比于 XML[2]，其具有以下特点：

- 语言无关、平台无关。即 ProtoBuf 支持 Java、C++、Python 等多种语言，支持多个平台
- 高效。即比 XML 更小（3 ~ 10倍）、更快（20 ~ 100倍）、更为简单
- 扩展性、兼容性好。你可以更新数据结构，而不影响和破坏原有的旧程序

> 序列化[1]：将结构数据或对象转换成能够被存储和传输（例如网络传输）的格式，同时应当要保证这个序列化结果在之后（可能在另一个计算环境中）能够被重建回原来的结构数据或对象。
> 更为详尽的介绍可参阅 维基百科。
> 类比于 XML[2]：这里主要指在数据通信和数据存储应用场景中序列化方面的类比，但个人认为 XML 作为一种扩展标记语言和 ProtoBuf 还是有着本质区别的。

安装编译

以Ubuntu和c++为例:

1. 安装前需要的编译工具：
   - autoconf
   - automake
   - libtool
   - make
   - g++
   - unzip
2. 安装工具：

```
$ sudo apt-get install autoconf automake libtool curl make g++
unzip
```

3. 下载protobuf源码，此处用git来下载

```
$ git clone https://github.com/protocolbuffers/protobuf.git
```

4. 编译

```
# 进入到protobuf目录
$ cd protobuf

# 执行脚本
$ ./autogen.sh

# 构建和安装 C++ 协议缓冲区运行时和协议缓冲区编译器 (protoc)
$ ./configure
$ make
$ make check
$ sudo make install
$ sudo ldconfig # refresh shared library cache.
```

5. 编译依赖包，要先安装pkg-config

```
# 安装pkg-config管理器
$ sudo apt install pkg-config

# 符号说明
pkg-config --cflags protobuf         # print compiler flags
pkg-config --libs protobuf           # print linker flags
pkg-config --cflags --libs protobuf  # print both

# 编译时的句式，根据具体文件需要来设置
$ g++ my_program.cc my_proto.pb.cc `pkg-config --cflags --libs
protobuf`
```

实现protobuf官网示例

1. 编写addressbook.proto文件

```
syntax = "proto2";

package tutorial;

message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    optional string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phones = 4;
}
```

```
message AddressBook {
  repeated Person people = 1;
}
```

2. 编译addressbook.proto文件

```
# 路径根据具体情况设置
命令: protoc -I=$SRC_DIR --cpp_out=$DST_DIR
$SRC_DIR/addressbook.proto

# 本实例路径，全部在当前目录
$ protoc -I=. --cpp_out=. ./addressbook.proto
```

编译后生成如下两个文件:

addressbook.pb.h, 声明生成的类的头

addressbook.pb.cc, 它包含类的实现

3. 将protobuf头文件加入到项目目录中:

头文件路径：**protobuf/src/google**

将整个google文件夹拷贝到项目中

4. 发送消息端代码，send_main.cpp：

```
#include <iostream>
#include <fstream>
#include <string>
#include "addressbook.pb.h"

using namespace std;

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;
        }
    }
}
```

```
int main(int argc, char* argv[])
{
    /*
    请注意GOOGLE_PROTOBUF_VERIFY_VERSION宏,在使用C++协议缓冲库之前，执行这个宏是一个很好的　　实践--虽然不是绝对必要的。它验证您没有意外地链接到与您编译的标题版本不兼容的库版本。如果检测　　到版本不匹配，程序将中止。请注意，每个. pb.cc文件都会在启动时自动调用该宏。
    */
    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(!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;
        }
    }

    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;
        }
    }
```

```
    /*
    请注意在程序结束时对ShutdownProtobufLibrary()的调用。所有这些都是销毁由Protocol Buffer库分配的所有全局对象。对于大多数程序来说，这是不必要的，因为无论如何这个进程都将退出，操作系统将负责回收所有的内存。但是，如果您使用的内存泄漏检查器要求释放最后一个对象，或者如果您正在编写一个库，该库可能由一个进程多次加载和卸载，那么您可能希望强制Protocol Buffer销毁所有内容。
    */
    google::protobuf::ShutdownProtobufLibrary();

    return 0;
}
```

5. 接收消息端代码，recv_main.cpp：

```
#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 information:" << endl;
        cout << "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;
}
```

6. 编译两个源文件

```
# 编译send_main.cpp
$ g++ -o send send_main.cpp addressbook.pb.cc `pkg-config --
cflags --libs protobuf`

# 编译recv_main.cpp
$ g++ -o recv recv_main.cpp addressbook.pb.cc `pkg-config --
cflags --libs protobuf`
```

7. 执行程序

```
# 写入文件
sky@sky:~/demo3/testProtobuf$ ./send log0
log0: File not found.  Creating a new file.
Enter person ID number: 123
Enter name: liming
Enter email address (blank for none): 1234@qq.com
Enter a phone number (or leave blank to finish): 123456789
Is this a mobile, home, or work phone? mobile
Enter a phone number (or leave blank to finish):

# 读取文件
sky@sky:~/demo3/testProtobuf$ ./recv log0
Person information:
ID: 123
Name: liming
E-mail address: 1234@qq.com
Mobile phone #: 123456789
```

注：可以参考开发学院的中文文档或者protobuf官方文档