<https://developers.google.com/protocol-buffers/docs/overview>

开发指南

欢迎来到protocol buffer的开发者指南文档，一种语言无关、平台无关、扩展性好的用于通信协议、数据存储的结构化数据序列化方法。

本文档是面向计划将protocol buffer使用的到自己的Java、C++或Python应用程序中的开发者的。这个概览介绍了protocol buffer，并告诉你如何开始，你随后可以跟随编程指导（<https://developers.google.com/protocol-buffers/docs/tutorials> ）去深入研究protocol buffer编码方式(<https://developers.google.com/protocol-buffers/docs/encoding> )。同时API参考文档(<https://developers.google.com/protocol-buffers/docs/reference/overview> )也是提供了这三种编程语言的版本，以及可以参考语言指南(<https://developers.google.com/protocol-buffers/docs/proto> )和样式指南(<https://developers.google.com/protocol-buffers/docs/style> )去编写 .proto 文件。

## 什么是protocol buffer

ProtocolBuffer是用于结构化数据序列化的灵活、高效、自动的方法，类似XML，不过它更小、更快、也更简单。你可以定义自己的数据结构，然后使用代码生成器生成的代码来读写这个数据结构。你甚至可以在无需重新部署程序的情况下更新数据结构。

## 他们如何工作

你首先需要在一个 .proto 文件中定义你需要做序列化的数据结构信息。每个ProtocolBuffer信息是一小段逻辑记录，包含一系列的键值对。这里有个非常简单的 .proto 文件定义了个人信息：

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;

}

有如你所见，消息格式很简单，每个消息类型拥有一个或多个特定的数字字段，每个字段拥有一个名字和一个值类型。值类型可以是数字(整数或浮点)、布尔型、字符串、原始字节或者其他ProtocolBuffer类型，还允许数据结构的多层次嵌套。你可以指定可选字段，必选字段和重复字段。你可以在(<https://developers.google.com/protocol-buffers/docs/proto> )找到更多关于如何编写 .proto 文件的信息。

一旦你定义了自己的报文格式(message)，你就可以运行ProtocolBuffer编译器，将你的 .proto 文件编译成特定语言的类。这些类提供了简单的方法访问每个字段(像是 name() 和 set\_name() )，像是访问类的方法一样将结构序列化或反序列化。例如你可以选择C++语言，运行编译如上的协议文件生成类叫做 Person 。随后你就可以在应用中使用这个类来序列化的读取报文信息。你可以这么写代码：

Person person;  
person.set\_name("John Doe");  
person.set\_id(1234);  
person.set\_email("jdoe@example.com");  
fstream output("myfile", ios::out | ios::binary);  
person.SerializeToOstream(&output);

然后，你可以读取报文中的数据：

fstream input("myfile", ios::in | ios::binary);  
Person person;  
person.ParseFromIstream(&input);  
cout << "Name: " << person.name() << endl;  
cout << "E-mail: " << person.email() << endl;

你可以在不影响向后兼容的情况下随意给数据结构增加字段，在反序列化的时候，旧有的数据会忽略新的字段。所以如果使用ProtocolBuffer作为通信协议，你可以无须担心破坏现有代码的情况下扩展协议。

你可以在API参考(<https://developers.google.com/protocol-buffers/docs/reference/overview> ) 中找到完整的参考，并且关于ProtocolBuffer的报文格式编码则可以在(<https://developers.google.com/protocol-buffers/docs/encoding> )中找到参考。

## 为什么不用XML？

ProtocolBuffer拥有多项比XML更高级的序列化结构数据的特性，ProtocolBuffer：

· 更简单

· 小3-10倍

· 快20-100倍

· 更少的歧义

· 可以方便的生成数据存取类

例如，让我们看看如何在XML中建模Person的name和email字段：

<person>

<name>John Doe</name>

<email>jdoe@example.com</email>

</person>

对应的ProtocolBuffer报文则如下：

# Textual representation of a protocol buffer.

# This is \*not\* the binary format used on the wire.

person {

name: "John Doe"

email: "jdoe@example.com"

}

当这个报文编码到ProtocolBuffer的二进制格式

(<https://developers.google.com/protocol-buffers/docs/encoding> )时(上面的文本仅用于调试和编辑)，它只需要28字节和100-200ns的解析时间。而XML的版本需要69字节(除去空白)和5000-10000ns的解析时间。

当然，操作ProtocolBuffer也很简单：

  cout << "Name: " << person.name() << endl;  
  cout << "E-mail: " << person.email() << endl;

而XML的你需要：

cout << "Name: "  
       << person.getElementsByTagName("name")->item(0)->innerText()  
       << endl;  
  cout << "E-mail: "  
       << person.getElementsByTagName("email")->item(0)->innerText()  
       << endl;

当然，ProtocolBuffer并不是在任何时候都比XML更合适，例如ProtocolBuffer无法对一个基于标记文本的文档建模，因为你根本没法方便的在文本中插入结构。另外，XML是便于人类阅读和编辑的，而ProtocolBuffer则不是。还有XML是自解释的，而 ProtocolBuffer仅在你拥有报文格式定义的 .proto 文件时才有意义。

## 听起来像是我要的解决方案，如何开始?

下载包(<https://developers.google.com/protocol-buffers/docs/downloads.html> )，包含了Java、Python、C++的ProtocolBuffer编译器，用于生成你需要的IO类。构建和安装你的编译器，按照README的说明进行就可以做到。

一旦你安装好了，就可以跟着编程指导

(<https://developers.google.com/protocol-buffers/docs/tutorials> )来选择语言-随后就是使用ProtocolBuffer创建一个简单的应用了。

## 介绍proto3

我们的最新版本3 alpha (<https://github.com/google/protobuf/releases> )版本引入了一个新的语言版本- Protocol Buffers语言版本3(又名proto3)，以及一些新特性在我们现有的语言版本(又名proto2)。Proto3简化了Protocol Buffers语言,易用性更高使它可以成为一个广泛的编程语言。我们目前的alpha版本允许您生成Protocol Buffers代码在Java、c++、Python、JavaNano,和Ruby。此外您可以生成proto3代码去使用最新的protoc插件,可以从golang / protobuf Github库。更多的语言是管道。

我们目前推荐尝试proto3只有以下情况时使用：

* 如果你想要尝试使用protocol buffers在我们的一个新支持的语言。
* 如果你想尝试我们的新开源RPC实现gRPC (<http://github.com/grpc/grpc-common> )(目前也是alpha版本),我们建议使用所有新gRPC proto3服务器和客户端,避免兼容性问题。

注意，两个语言版本的api不是完全兼容。为了避免对现有用户的不便,我们将在新Protocol Buffers正式版本中继续支持以前的语言版本。

(如果名字proto2和proto3似乎有点让人困惑,因为当我们最初开源Protocol Buffers它实际上是google的第二个版本的语言——也称为proto2。这也是为什么我们的开源版本号从v2.0.0)。

## 一点历史

ProtocolBuffer最初是在Google开发的，用以解决索引服务器的请求、响应协议。在使用ProtocolBuffer之前，有一种格式用以处理请求和响应数据的编码和解码，并且支持多种版本的协议。而这最终导致了丑陋的代码，有如：

 if (version == 3) {  
   ...  
 } else if (version > 4) {  
   if (version == 5) {  
     ...  
   }  
   ...  
 }

通信协议因此变得越来越复杂，因为开发者必须确保，发出请求的人和接受请求的人必须同时兼容，并且在一方开始使用新协议时，另外一方也要可以接受。

ProtocolBuffer设计用于解决这一类问题：

· 很方便引入新字段，而中间服务器可以忽略这些字段，直接传递过去而无需理解所有的字段。

· 格式可以自描述，并且可以在多种语言中使用(C++、Java等)

然而用户仍然需要手写解析代码。

随着系统的发展，它拥有了一些其它的特性和功能：

· 自动生成编码和解码代码，而无需自己编写解析器。

· 除了用于简短的RPC(Remote Procedure Call)请求，人们使用ProtocolBuffer来做数据存储格式(例如BitTable)。

· RPC服务器接口可以作为 .proto 文件来描述，而通过ProtocolBuffer的编译器生成存根(stub)类供用户实现服务器接口。

ProtocolBuffer现在已经是Google的混合语言数据标准了，现在已经正在使用的有超过48,162种报文格式定义和超过12,183个 .proto 文件。他们用于RPC系统和持续数据存储系统。