# 陈硕的Blog

吾尝终日而思矣，不如须臾之所学也。吾尝跂而望矣，不如登高之博见也。……君子生非异也，善假于物也。

@bnu_chenshuo



个人资料

陈硕



# 一种自动反射消息类型的 Google Protobuf 网络传输方案

标签： 网络 google descriptor prototype string header

2011-04-03 15:57 58124人阅读 评论(126) 收藏 举报







陈硕 (giantchen_AT_gmail)

Blog.csdn.NET/Solstice t.sina.com.cn/giantchen

这篇文章要解决的问题是：在接收到 protobuf 数据之后，如何自动创建具体的 Protobuf Message 对象，再做的反序列化。“自动”的意思是：当程序中新增一个 protobuf Message 类型时，这部分代码不需要修改，不需要自己去注册消息类型。其实，Google Protobuf 本身具有很强的反射(reflection)功能，可以根据 type name 创建具体类型的 Message 对象，我们直接利用即可。

本文假定读者了解 Google Protocol Buffers 是什么，这不是一篇 protobuf 入门教程。

本文以 C++ 语言举例，其他语言估计有类似的解法，欢迎补充。

本文的示例代码在： https://github.com/chenshuo/recipes/tree/master/protobuf

## 网络编程中使用 **protobuf** 的两个问题

Google Protocol Buffers (Protobuf) 是一款非常优秀的库，它定义了一种紧凑的可扩展二进制消息格式，特别适合网络数据传输。它为多种语言提供 binding，大大方便了分布式程序的开发，让系统不再局限于用某一种语言来编写。

在网络编程中使用 protobuf 需要解决两个问题：

- 长度，protobuf 打包的数据没有自带长度信息或终结符，需要由应用程序自己在发生和接收的时候做正确的切分；
- 类型，protobuf 打包的数据没有自带类型信息，需要由发送方把类型信息传给给接收方，接收方创建具体的 Protobuf Message 对象，再做的反序列化。

第一个很好解决，通常的做法是在每个消息前面加个固定长度的 length header，例如我在 《Muduo 网络编程示例之二： Boost.Asio 的聊天服务器》 中实现的 LengthHeaderCodec，代码见 http://code.google.com/p/muduo/source/browse/trunk/examples/asio/chat/codec.h

第二个问题其实也很好解决，Protobuf 对此有内建的支持。但是奇怪的是，从网上简单搜索的情况看，我发现了很多山寨的做法。

# 山寨做法

以下均为在 protobuf data 之前加上 header，header 中包含 int length 和类型信息。类型信息的山寨做法主要有两种：

- 在 header 中放 int typeId，接收方用 switch-case 来选择对应的消息类型和处理函数；
- 在 header 中放 string typeName，接收方用 look-up table 来选择对应的消息类型和处理函数。

这两种做法都有问题。

第一种做法要求保持 typeId 的唯一性，它和 protobuf message type 一一对应。如果 protobuf message 的使用范围不广，比如接收方和发送方都是自己维护的程序，那么 typeId 的唯一性不难保证，用版本管理工具即可。如果 protobuf message 的使用范围很大，比如全公司都在用，而且不同部门开发的分布式程序可能相互通信，那么就需要一个公司内部的全局机构来分配 typeId，每次增加新 message type 都要去注册一下，比较麻烦。

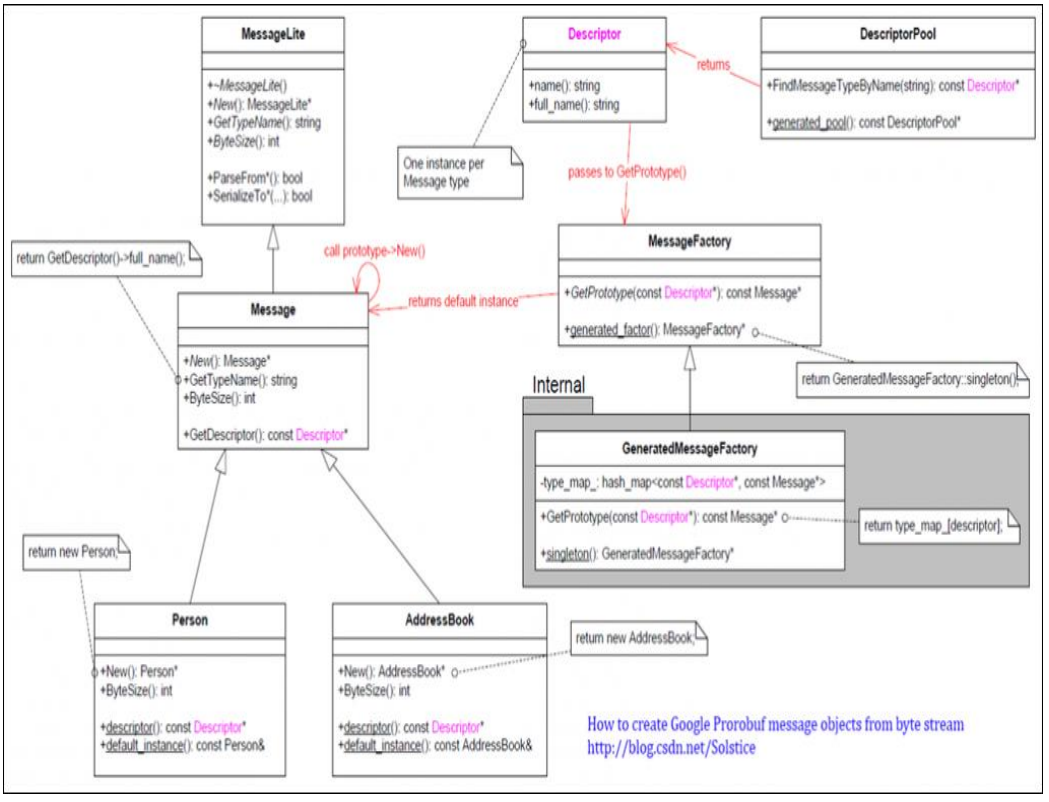
第二种做法稍好一点。typeName 的唯一性比较好办，因为可以加上 package name（也就是用 message 的 fully qualified type name），各个部门事先分好 namespace，不会冲突与重复。但是每次新增消息类型的时候都要去手工修改 look-up table 的初始化代码，比较麻烦。

其实，不需要自己重新发明轮子，protobuf 本身已经自带了解决方案。

# 根据 **type name** 反射自动创建 **Message** 对象

Google Protobuf 本身具有很强的反射(reflection)功能，可以根据 type name 创建具体类型的 Message 对象。但是奇怪的是，其官方教程里没有明确提及这个用法，我估计还有很多人不知道这个用法，所以觉得值得写这篇 blog 谈一谈。

以下是陈硕绘制的 Protobuf class diagram，点击查看原图。



我估计大家通常关心和使用的是图的左半部分：MessageLite、Message、Generated Message Types (Person, AddressBook) 等，而较少注意到图的右半部分：Descriptor, DescriptorPool, MessageFactory。

上图中，其关键作用的是 Descriptor class，每个具体 Message Type 对应一个 Descriptor 对象。尽管我们没有直接调用它的函数，但是Descriptor在“根据 type name 创建具体类型的 Message 对象”中扮演了重要的角色，起了桥梁作用。上图的红色箭头描述了根据 type name 创建具体 Message 对象的过程，后文会详细介绍。

## 原理简述

Protobuf Message class 采用了 prototype pattern，Message class 定义了 New() 虚函数，用以返回本对象的一份新实例，类型与本对象的真实类型相同。也就是说，拿到 Message* 指针，不用知道它的具体类型，就能创建和它类型一样的具体 Message Type 的对象。

每个具体 Message Type 都有一个 default instance，可以通过 ConcreteMessage::default_instance() 获得，也可以通过 MessageFactory::GetPrototype(const Descriptor*) 来获得。所以，现在问题转变为 1. 如何拿到 MessageFactory；2. 如何拿到 Descriptor*。

当然，ConcreteMessage::descriptor() 返回了我们想要的 Descriptor*，但是，在不知道 ConcreteMessage 的时候，如何调用它的静态成员函数呢？这似乎是个鸡与蛋的问题。

我们的英雄是 DescriptorPool，它可以根据 type name 查到 Descriptor*，只要找到合适的 DescriptorPool，再调用 DescriptorPool::FindMessageTypeByName(const string& type_name) 即可。眼前一亮？

在最终解决问题之前，先简单测试一下，看看我上面说的对不对。

## 简单测试

本文用于举例的 proto 文件：query.proto，见
https://github.com/chenshuo/recipes/blob/master/protobuf/query.proto

```
package muduo;

message Query {
  required int64 id = 1;
  required string questioner = 2;

  repeated string question = 3;
}

message Answer {
  required int64 id = 1;
  required string questioner = 2;
  required string answerer = 3;

  repeated string solution = 4;
}

message Empty {
  optional int32 id = 1;
}
```

其中的 Query.questioner 和 Answer.answerer 是我在前一篇文章这提到的《分布式系统中的进程标识》。

以下代码验证 ConcreteMessage::default_instance()、ConcreteMessage::descriptor()、MessageFactory::GetPrototype()、DescriptorPool::FindMessageTypeByName() 之间的不变式 (invariant)：

https://github.com/chenshuo/recipes/blob/master/protobuf/descriptor_test.cc#L15

```
  typedef muduo::Query T;

  std::string type_name = T::descriptor()->full_name();
```

性之助

```
  cout << type_name << endl;

  const Descriptor* descriptor = DescriptorPool::generated_pool()->FindMessageTypeByName(type_name);
  assert(descriptor == T::descriptor());
  cout << "FindMessageTypeByName() = " << descriptor << endl;
  cout << "T::descriptor()         = " << T::descriptor() << endl;
  cout << endl;

  const Message* prototype = MessageFactory::generated_factory()->GetPrototype(descriptor);
  assert(prototype == &T::default_instance());
  cout << "GetPrototype()         = " << prototype << endl;
  cout << "T::default_instance() = " << &T::default_instance() << endl;
  cout << endl;

  T* new_obj = dynamic_cast(prototype->New());
  assert(new_obj != NULL);
  assert(new_obj != prototype);
  assert(typeid(*new_obj) == typeid(T::default_instance()));
  cout << "prototype->New() = " << new_obj << endl;
  cout << endl;
  delete new_obj;
```

### 根据 **type name** 自动创建 **Message** 的关键代码

好了，万事具备，开始行动：

1. 用 DescriptorPool::generated_pool() 找到一个 DescriptorPool 对象，它包含了程序编译的时候所链接的全部 **protobuf Message types**。
2. 用 DescriptorPool::FindMessageTypeByName() 根据 type name 查找 Descriptor。
3. 再用 MessageFactory::generated_factory() 找到 MessageFactory 对象，它能创建程序编译的时候所链接的全部 protobuf Message types。
4. 然后，用 MessageFactory::GetPrototype() 找到具体 Message Type 的 default instance。
5. 最后，用 prototype->New() 创建对象。

示例代码见 https://github.com/chenshuo/recipes/blob/master/protobuf/codec.h#L69

```
Message* createMessage(const std::string& typeName)
{
  Message* message = NULL;
  const Descriptor* descriptor = DescriptorPool::generated_pool()->FindMessageTypeByName(typeName);
  if (descriptor)
  {
    const Message* prototype = MessageFactory::generated_factory()->GetPrototype(descriptor);
    if (prototype)
    {
      message = prototype->New();
    }
  }
  return message;
}
```

调用方式：https://github.com/chenshuo/recipes/blob/master/protobuf/descriptor_test.cc#L49

```
  Message* newQuery = createMessage("muduo.Query");
  assert(newQuery != NULL);
  assert(typeid(*newQuery) == typeid(muduo::Query::default_instance()));
  cout << "createMessage(/"muduo.Query/") = " << newQuery << endl;
```

古之人不余欺也 :-)

注意，createMessage() 返回的是动态创建的对象的指针，调用方有责任释放它，不然就会内存泄露。在 muduo 里，我用 shared_ptr 来自动管理 Message 对象的生命期。

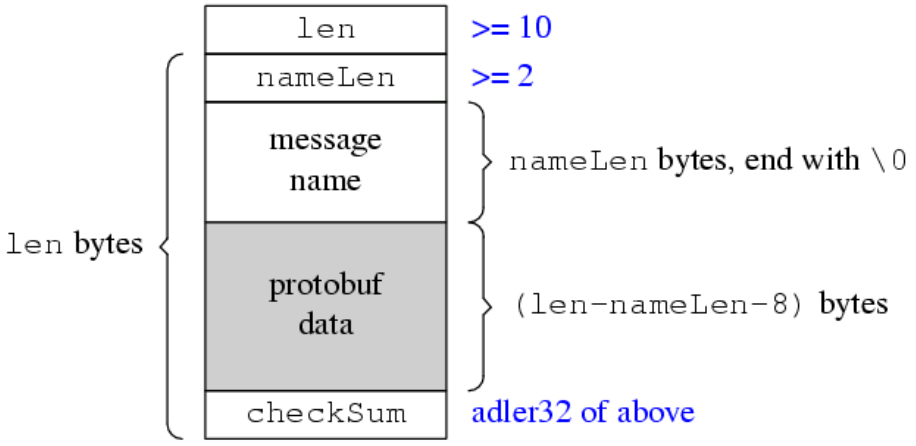
### 线程安全性

Google 的文档说，我们用到的那几个 MessageFactory 和 DescriptorPool 都是线程安全的，Message::New() 也是线程安全的。并且它们都是 const member function。

关键问题解决了，那么剩下工作就是设计一种包含长度和消息类型的 **protobuf** 传输格式。

## Protobuf 传输格式

陈硕设计了一个简单的格式，包含 protobuf data 和它对应的长度与类型信息，消息的末尾还有一个 check sum。格式如下图，图中方块的宽度是 32-bit。



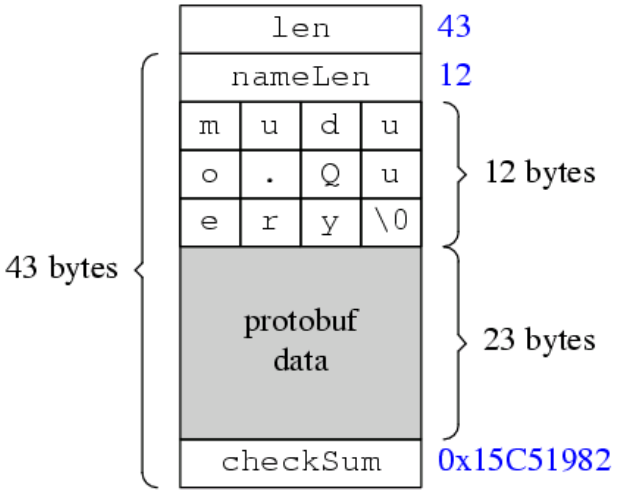

用 C struct 伪代码描述：

```
struct ProtobufTransportFormat __attribute__ ((__packed__))
{
  int32_t  len;
  int32_t  nameLen;
  char     typeName[nameLen];
  char     protobufData[len-nameLen-8];
  int32_t  checkSum; // adler32 of nameLen, typeName and protobufData
};
```

注意，这个格式不要求 32-bit 对齐，我们的 decoder 会自动处理非对齐的消息。

### 例子

用这个格式打包一个 muduo.Query 对象的结果是：




设计决策

以下是我在设计这个传输格式时的考虑：

- **signed int**。消息中的长度字段只使用了 signed 32-bit int，而没有使用 unsigned int，这是为了移植性，因为 Java 语言没有 unsigned 类型。另外 Protobuf 一般用于打包小于 1M 的数据，unsigned int 也没用。
- **check sum**。虽然 TCP 是可靠传输协议，虽然 Ethernet 有 CRC-32 校验，但是网络传输必须要考虑数据损坏的情况，对于关键的网络应用，check sum 是必不可少的。对于 protobuf 这种紧凑的二进制格式而言，肉眼看不出数据有没有问题，需要用 check sum。
- **adler32** 算法。我没有选用常见的 CRC-32，而是选用 adler32，因为它计算量小、速度比较快，强度和 CRC-32差不多。另外，zlib 和 java.unit.zip 都直接支持这个算法，不用我们自己实现。
- **type name** 以 **'/0'** 结束。这是为了方便 troubleshooting，比如通过 tcpdump 抓下来的包可以用肉眼很容易看出 type name，而不用根据 nameLen 去一个个数字节。同时，为了方便接收方处理，加入了 nameLen，节省 strlen()，空间换时间。
- 没有版本号。Protobuf Message 的一个突出优点是用 optional fields 来避免协议的版本号（凡是在 protobuf Message 里放版本号的人都没有理解 protobuf 的设计），让通信双方的程序能各自升级，便于系统演化。如果我设计的这个传输格式又把版本号加进去，那就画蛇添足了。具体请见本人《分布式系统的工程化开发方法》第 57 页：消息格式的选择。

# 示例代码

为了简单起见，采用 std::string 来作为打包的产物，仅为示例。

打包 encode 的代码：https://github.com/chenshuo/recipes/blob/master/protobuf/codec.h#L35

解包 decode 的代码：https://github.com/chenshuo/recipes/blob/master/protobuf/codec.h#L99

测试代码： https://github.com/chenshuo/recipes/blob/master/protobuf/codec_test.cc

如果以上代码编译通过，但是在运行时出现“cannot open shared object file”错误，一般可以用 sudo ldconfig 解决，前提是 libprotobuf.so 位于 /usr/local/lib，且 /etc/ld.so.conf 列出了这个目录。

```
$ make all # 如果你安装了 boost，可以 make whole

$ ./codec_test
./codec_test: error while loading shared libraries: libprotobuf.so.6: cannot open shared object file: No such file or directory

$ sudo ldconfig
```

# 与 muduo 集成

muduo 网络库将会集成对本文所述传输格式的支持（预计 0.1.9 版本），我会另外写一篇短文介绍 Protobuf

Message <=> muduo::net::Buffer 的相互转化，使用 muduo::net::Buffer 来打包比上面 std::string 的代码还简单，它是专门为 non-blocking 网络库设计的 buffer class。

此外，我们可以写一个 codec 来自动完成转换，就行 asio/char/codec.h 那样。这样客户代码直接收到的就是 Message 对象，发送的时候也直接发送 Message 对象，而不需要和 Buffer 对象打交道。

### 消息的分发 **(dispatching)**

目前我们已经解决了消息的自动创建，在网络编程中，还有一个常见任务是把不同类型的 Message 分发给不同的处理函数，这同样可以借助 Descriptor 来完成。我在 muduo 里实现了 ProtobufDispatcherLite 和 ProtobufDispatcher 两个分发器，用户可以自己注册针对不同消息类型的处理函数。预计将会在 0.1.9 版本发布，您可以先睹为快：

初级版，用户需要自己做 down casting：
https://github.com/chenshuo/recipes/blob/master/protobuf/dispatcher_lite.cc

高级版，使用模板技巧，节省用户打字： https://github.com/chenshuo/recipes/blob/master/protobuf/dispatcher.cc

### 基于 **muduo** 的 **Protobuf RPC?**

Google Protobuf 还支持 RPC，可惜它只提供了一个框架，没有开源网络相关的代码，muduo 正好可以填补这一空白。我目前还没有决定是不是让 muduo 也支持以 protobuf message 为消息格式的 RPC，muduo 还有很多事情要做，我也有很多博客文章打算写，RPC 这件事情以后再说吧。

注：Remote Procedure Call (RPC) 有广义和狭义两种意思。狭义的讲，一般特指 ONC RPC，就是用来实现 NFS 的那个东西；广义的讲，“以函数调用之名，行网络通信之实”都可以叫 RPC，比如 **Java** RMI，.Net Remoting，Apache Thrift，libevent RPC，XML-RPC 等等。

(待续)



上一篇 构建易于维护的分布式程序
下一篇 在 muduo 中实现 protobuf 编解码器与消息分发器


查看评论

66楼 liuchuan98 2016-10-25 09:49发表

陈老师你好，我最近在看你的linux网络多线程的书，收获颇多。
关于rpc调用，你说：你调用 Foo()，那么输入和输出就是 FooRequest 和 FooResponse；调用 Bar() 就是 BarRequest 和 BarResponse。在另一个层面解决了消息类型的问题。

那如果Foo带参数，特别是参数是自定义的，那参数是怎么传递的？ 比如Foo(int a, string b, myStruct temp)这种？ 我以前用C#做的时候采用一个通用请求类如：
class rpcRequest
{
public string methodName;
public List<object> methodParams;
}
然后发请求的时候把这个结构填充然后序列化为bytes发出去请求远程调用。
c++里面貌似可以用boost::any，但它不支持序列化，所以没办法用这种方式。或者将参数都转成string？ 请问你有没什么办法

Re: 陈硕 2016-10-25 13:11发表

回复liuchuan98：Foo 的参数只能有一个，就是 FooRequest，FooRequest 是个protobuf message。

Re: liuchuan98 2016-10-25 14:15发表

哦，那我理解不同的request会有不同的类型，每一个request都要重新添加proto文件了？ 比如FooRequest：
message FooRequest {
required uint64 a= 1;
required float cost = 2;
required string tag = 3;
}
有没有通用一点的方法

Re: 陈硕 2016-10-26 00:17发表

回复liuchuan98：这就是最通用的方法。
一个 proto 文件可以定义多个 messages。

65楼 keynumber 2016-09-01 10:22发表

如果 protobuf message 的使用范围很大，比如全公司都在用，而且不同部门开发的分布式程序可能相互通信，那么就需要一个公司内部的全局机构来分配 typeId，每次增加新 message type 都要去注册一下，比较麻烦。

看不懂这个需求，为什么typeid会有全局唯一这个需求呢？求举例子

64楼 pascal4 2016-08-09 16:42发表

不知道楼主测过性能没，GetTypeName的开销特别大，在我的测试代码里占去了大部分时间

Re: pascal4 2016-08-09 20:34发表

回复pascal4：额，当我没说，首次启动会很慢。。。

63楼 zlyangQQ 2016-05-26 19:31发表

请教， 这种方式能避免写后继的switch case处理函数吗？

Re: 陈硕 2016-05-29 00:11发表

回复zlyangQQ：见文中“消息的分发 (dispatching)”一节。

62楼 不想用了这个号 2016-05-06 13:53发表

看了挺好
不过锁的问题是硬伤
名字的问题解决了,但是对于数据量小,频繁调用,但是名字长的结构就悲剧了

61楼 DaHuZiNanRen 2016-03-22 11:12发表

struct ProtobufTransportFormat
为何不用protobuf定义一个
自己发明轮子

60楼 qq_33474661 2016-01-05 00:26发表

包头里带ID与len,这个是传统做法吧,不应该是山寨.
单纯论运行效率,其实没差别.
要是说亮点,只有一个.包长少了ID的4字节.
缺点: 传统做法比较安全的,把收到buf转成对应类型.PB增加耦合性,2者有依赖,主要是PB内部一直在更新改动,今时不同往日,官方有明确说明才算
折中办法:先用传统的稳定方法.等反射应用广泛与经受考验后在做考虑

59楼 murphy0626 2015-07-31 10:27发表

大牛，谢谢。用到了

58楼 hp0632 2015-02-05 17:47发表

你好，问一下，这个是关于C++版本的protobuf动态加载，那问一下java版本的动态加载有没有相应的接口或是方式吗？如何？

Re: applepwc 2015-06-01 17:40发表

回复hp0632：你解决了这个问题没?

57楼 sarlmolchen 2014-07-21 19:22发表

这种用法只是看起来高大上而已，无端端增加了strcmp的开销，现实中没人这么用

Re: 陈硕 2014-07-23 02:29发表

回复sarlmolchen：好奇特的理由——strcmp？

56楼 coderchenjingui 2014-04-05 14:48发表

```
const Message* prototype = MessageFactory::generated_factory()->GetPrototype(descriptor);
if (prototype)
{
message = prototype->New();
}
```

这个地方我觉得直接使用protype就可以了吧，前提是逻辑是单线程的，不会有多个线程同时使用同一个默认实例。

这样是不是就可以省去大量的new和delete呢？

Re: 陈硕 2014-04-05 23:45发表

回复QQ575787460：你验证过你说的做法吗？
编译能通过吗？
程序功能正常吗？
性能有显著提升吗？

Re: coderchenjingui 2014-04-07 23:17发表

回复Solstice：硕哥息怒，我前天网上已经去验证了，编译是不通过的，因为返回的是个const Message*

的。

55楼 zhuwei1111 2014-02-21 10:31发表

你好，我们公司现在做的项目有一模块需要调用另一公司提供的动态库里的方法，获取数据，其中有一个方法返回一个c++结构体，当我在java中获取到后怎么将这个结构体信息转换成我定义的一个javabean对象，急急急！也可qq联系，244840434

54楼 阳光梦 2014-01-15 18:16发表

您的
EchoClient client(&loop, serverAddr, size);
client.connect();
loop.loop();
是在main函数中调用的，而我现在需要将
EchoClient client(&loop, serverAddr, size);
client.connect();
在其他函数中使用，这样会有问题的。

53楼 阳光梦 2014-01-15 17:22发表

陈哥好：请教，当服务器端某函数中调用
CheckDevPortMap *client = CheckDevPortMap (loop_, serverAddr, devsid);
client.connect();
由于是异步连接我怎样释放该client 指向内存啊？

52楼 hzhxxx 2014-01-12 11:35发表

1. namelen 是否太长了，是否考虑使用 short 就能支持，能省两个字节。
2. 使用 typeName ，内部使用 反射机制来动态创建消息对象，效率比较低。
http://www.searchtb.com/2012/09/protocol-buffers.html
网上的这篇文章进行了解释。

Re: 陈硕 2014-01-12 11:44发表

回复hzhxxx：1. 看不出有多大意义。
2. 那篇文章跟我说的完全不是一回事。

Re: hzhxxx 2014-01-12 12:11发表

回复Solstice：如果协议上要支持校验和算法选择，应该是扩展 naleLen 字段呢，还是用新增加一个字段。毕竟要综合考虑一些服务调用者不方便使用的问题，包括数据打包，不能很好的支持位运算。

Re: 陈硕 2014-01-12 12:35发表

回复hzhxxx：Checksum算法用统一的一种就行，没必要留灵活性。

51楼 阳光梦 2014-01-03 10:32发表

陈大哥，我主要想问：当服务器调用conn->shutdown()断开客户端时候或者由于客户端规定时间内未发送心跳服务器超时断开是否调用EchoServer::onConnection()？这样我就可以在该函数中做一些资源回收工作。

Re: 陈硕 2014-01-03 10:55发表

回复yangguangmeng：书第7.2节。

50楼 阳光梦 2014-01-03 10:14发表

按理应该是服务器收到的包格式不对应该干掉客户端，服务器收到的包格式对但是内容不满足要求还得干掉客户端，你的代码里面体现在哪块？

Re: 陈硕 2014-01-03 10:24发表

回复yangguangmeng：Echo服务哪有你说的这些问题？

49楼 阳光梦 2014-01-03 10:08发表

我是故意这么测试的啊，目的测试服务器主动断开以及超时主动断开时候是否调用EchoServer::onConnection()函数，异常断开服务器只能超时断开啊。

Re: 陈硕 2014-01-03 10:13发表

回复yangguangmeng：书第7.2节。

48楼 阳光梦 2014-01-03 09:57发表

陈大哥好：您用muduo\examples\idleconnection\sortedlist.cc作为服务端，下面作为客户端：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include <arpa/inet.h>

int main(int argc, char *argv[])
{
int fd = 0;
char buf[100]="";

struct sockaddr_in addr = {AF_INET};
addr.sin_addr.s_addr = inet_addr("192.168.1.114");
addr.sin_port = htons(7000);

fd = socket(AF_INET, SOCK_STREAM, 0);
if( fd < 0 )
{
perror("socket");
exit(1);
}

if (connect(fd, (struct sockaddr*)&addr, sizeof(addr)) !=0)
{
perror("connect");
exit(1);
}
printf("connect sucess!\n");

send(fd, "nihao", strlen("nihao"),0);
while(1){sleep(5);}

return 0;
}
```

服务端在EchoServer::onMessage()里面收到消息就调用 conn->shutdown();您看是否会调用EchoServer::onConnection()，我测试服务器主动调用conn->shutdown()不会执行EchoServer::onConnection()函数的。
还有就是服务器超时断开还是不会执行EchoServer::onConnection()函数的。您不是说这个函数为了客户连接时候的初始化工作以及(不管服务器主动断开还是客户主动断开的资源回收操作)?

Re: 陈硕 2014-01-03 10:04发表

回复yangguangmeng：你的客户端没有调用close()。

47楼 阳光梦 2014-01-02 19:57发表

如果服务器主动断开连接，不调用onConnection()，那么该如何delete节点啊？

46楼 阳光梦 2014-01-02 18:30发表

我写了个线程隔几秒给server_threadpool服务器发数据，而且是错误的数据，但是并未见服务器调用onConnection()

Re: 陈硕 2014-01-03 08:18发表

回复yangguangmeng：Shoe me your code and steps to reproduce.

45楼 阳光梦 2014-01-02 17:40发表

硕哥：您上次说不管服务器主动断开还是客户主动断开，服务器均会执行onConnection()函数，但是您的代码中服务器主动断开并没有执行该函数的？

44楼 阳光梦 2014-01-02 16:53发表

加锁已经明白了，但是不是(one pthread, one loop)吗，为何还要isInLoopThread()呢？

```
void EventLoop::runInLoop(const Functor& cb)
{
if (isInLoopThread())
{
cb();
}
else
{
queueInLoop(cb);
}
}
```

43楼 阳光梦 2014-01-02 15:32发表

硕哥：为什么要加锁呢，目的是？

```
EventLoop* EventLoopThread::startLoop()
```

```
{
assert(!thread_.started());
thread_.start();

{
MutexLockGuard lock(mutex_);
while (loop_ == NULL)
{
cond_.wait();
}
}

return loop_;
}
```

42楼 阳光梦 2013-12-31 14:16发表

硕哥好，
请问::getenv("MUDUO_USE_POLL")，默认是不是不采用的，那如果配置了环境变量会干些什么？ 谢谢！

41楼 阳光梦 2013-12-28 14:32发表

陈老师好，请教：针对您这篇，网络传输加密问题怎么做呢？

Re: 陈硕 2013-12-29 00:26发表

回复yangguangmeng：SSL/TLS

40楼 q598162221 2013-11-10 15:07发表

博主的这种方式，基于一个事实 Google protobuf 中不对string进行编码.....我觉得string不写在Google protobuf 的结构中反而更好，取出string在调用createMessage函数...这样就可以保证string不被protobuf 编码，也不用在不同的语言，中写不同的解析函数 switch case.....

39楼 漫步月夜 2013-08-14 11:37发表

很感谢，我已经采用你的这种机制！

38楼 潇亦潇 2013-08-06 11:13发表

其实无需反射类型，只要反射静态实例化函数即可得到具体的对象

37楼 newzai 2013-04-23 11:47发表

该山寨用法的时候，还是需要的。个人倾向于把protobuf作为编解码使用，解放劳动力，其它的消息分发还是使用传统的山寨方式比较高效方便。
何况protobuf有很多种变体，C，java，object-c等。在不同的语言之间通信时，还不得不使用山寨方式。
倒是网络抓包分析是个问题，不像SNMP比较成熟，用个mib文件就可以分析协议。

36楼 leewon1988 2013-04-06 16:04发表

作者笔误：
用 DescriptorPool::generated_pool() 找到一个 DescriptorPool 对象，它包含了程序编译的时候所链接的全部 protobuf Message types。
改为protobuf descriptor types

Re: 陈硕 2013-04-06 22:56发表

回复leewon1988：没有笔误，就是 protobuf Message types

35楼 tones 2013-03-12 20:08发表

楼主在讲解如何根据type_name创建具体的消息的时候还是绕进了鸡生蛋的问题。如果知道了type_name,还需要那些descriptor吗，显然，楼主假定已经知道type_name。文中重点是说将type_name放在自定义的数据结构，和data一起发送到网络，网络另一端接收后，根据type_name创建具体的消息然后界面。
最复杂的部分实际是封装了一个函数，根据type_name创建具体消息的方法的通用实现，而不是一般的做法，通过switch，case或者if判断，又或者自建的查表法创建。

Re: q598162221 2013-11-10 15:00发表

回复kingsleer：按照博主的意思好像是 不用再专门为每种语言写不同的解析代码，而是用google protocol buf 自带的代码来自动 解析这些事.....

Re: tones 2013-03-12 20:09发表

回复kingsleer：根据type_name创建具体的消息然后解码

Re: newzai 2013-10-28 09:58发表

回复kingsleer：得到具体的消息后，每个消息的处理方法不一样，还不是一样要从新分发一次？？？这和在编解码之前分发改善不了什么东西，无非就是把解码这个方法调用提前了，类似与把某个语句提取到for循环之外差不多。

34楼 zyy_huaweiren 2012-11-26 15:09发表

提供任意的schema描述（.proto文件），那能通过一个已经编译好的二进制获取提供schema里面的任意字段么？

我目前需要开发一个工具，想读入提供的schema，然后获取指定属性的value。不知道能不能实现？
目前一个山寨的想法是，我给出一个固定的schema，只包含一个required的字段。强制要求用户的schema必须包含这个字段（类似继承），程序中也只解析这个字段。显然这个很不灵活。

求大牛指导。

Re: 陈硕 2012-11-26 23:02发表

回复zyy_huaweiren：http://cxwangyi.blogspot.com/2010/06/google-protocol-buffers-proto.html

33楼 HH2012yls 2012-10-05 11:49发表

你设计的这个消息格式如下：
陈硕设计了一个简单的格式，包含 protobuf data 和它对应的长度与类型信息，消息的末尾还有一个 check sum。
其中protobuf data存放的是什么内容？是muduo.Query对象？还是muduo.Query序列化后的string ?
如果是muduo.Query对象，接收端如何将该对象还原？
如果是序列化后的string，那么protobuf，所谓紧凑的消息格式，就没有体现出来！序列化后的string长度并没有比真实的数据内容短！
请陈老师指教！

Re: 陈硕 2012-10-05 12:32发表

回复HH2012yls：紧凑!=压缩

32楼 HH2012yls 2012-09-23 17:25发表

最近在研究Google Protocol Buffer ，想用它来做网络通信的数据结构。现有以下不解之处，请高手指点：
如何把Google Protocol Buffer类型的数据通过网络发送出去？
我通过它的SerializeToString发放，把内容序列化到string 中，然后把string 通过网络发送。接收端，收到后，用ParseFromString可以获得 Google Protocol Buffer对象。
这样过程数据都是正确的。但是Google Protocol Buffer号称的编码紧凑，数据量小优点有一点都体现不了。序列化到string后，string的长度比各个属性字符串加起来的长度还要略长.
Google Protocol Buffer对象本身，是很小，但是直接发送到网络上，接收端如何还原该Google Protocol Buffer对象？直接类型转换，不可以，我试了！

Re: 陈硕 2012-09-24 09:44发表

回复HH2012yls：你确定看了文章正文吗？

Re: HH2012yls 2012-09-24 10:03发表

回复Solstice：你好，你的帖子我看了，你的文章说的比较深。我现在还处于对protocol buffer比较低级的用法上面。
我现在想向你请教，如何把protocol buffer内容通过网络发送和接收？我用SerializeToString方法，发现并没有减少数据长度，且也不是二进制的，而是明文，但是官方文档说序列化到string里面是二进制的，不知道怎么回事？请赐教！

Re: 把自己熬成金黄 2015-08-11 17:21发表

回复HH2012yls：留待后人，具体见：http://agapple.iteye.com/blog/859052，少与不少是相对的。

31楼 herm_lib 2012-08-08 10:53发表

楼主介绍的protobuf编译时的类型查找和反射的用法是不错的。但但。。。 楼主把编译时类型查找和反射功能的用在文中的这个地方是非常不妥的，本来是一件非常简单的事情，把他复杂化了。小心误导了小朋友。呵呵。

30楼 whitetao 2011-11-25 10:47发表

终于看完了
感觉做法有些问题
你的做法是基于知道传过来的消息是什么类型的
那么假设现在B 给A传来了两个个消息（一个是Accept,一个是Reject）
虽然你都能够正确的decode出来一个Message*来，但是你怎么知道要用那一个类去解析这个Message呢？你还是取不出数据来

Re: 陈硕 2011-11-26 16:58发表

回复whitetao：文中有讲，“消息的分发 (dispatching)”

29楼 dingyan1225 2011-07-25 13:37发表

想请教一个问题，在Mac下怎样安装protobuf，怎样运行protobuf编译器，谢谢

28楼 dagongxinhao 2011-04-09 21:33发表

谢谢你了 哈哈

27楼 hzminghe4 2011-04-08 01:57发表

我想google 也不可能所有的系统都工作在rpc层吧。一个port上有多个不同类型的消息是不可避免的，这个时候还得根据不同的消息类型做dispatching

Re: 陈硕 2011-04-08 06:37发表

回复 hzminghe4：你调用 Foo()，那么输入和输出就是 FooRequest 和 FooResponse；调用 Bar() 就是 BarRequest 和 BarResponse。在另一个层面解决了消息类型的问题。

Re: wwooha 2011-04-09 11:33发表

回复 Solstice：看得出你在protobuf消息的网络传输那一方面了解挺深入的，但是protobuf的优秀还不止于此，希望和你聊一聊，相信我们都不会失望的。（QQ: 457034877）

Re: 陈硕 2011-04-09 15:45发表

回复 wwooha：我的 gmail 账号是 giantchen。

Re: 陈硕 2011-04-08 06:35发表

回复 hzminghe4：“一个port上有多个不同类型的消息是不可避免的”，对。这时根据 method name 做 dispatching，由 protoc 自动生成的 RPC service stub 做 dispatching，不需要程序员动手。

26楼 hzminghe4 2011-04-08 01:56发表

你指的是一个port,一个service, 一个request message,一个response message, caller发送request, 接收response, callee接收request,发送 response吗？

25楼 陈硕 2011-04-07 08:11发表

而 Foo 和 Bar 的 stubs 是 protoc 自动生成的，里边包含了消息类型。只要 RPC method 能对应，输入输出消息类型不在话下。

24楼 陈硕 2011-04-07 08:10发表

因为 google 是以 RPC 方式传输 protobuf，在 RPC method 一级已经确定了消息的类型，例如 FooResponse Foo(FooRequest), BarResponse Bar(BarRequest)。

23楼 陈硕 2011-04-07 08:10发表

忽然明白 protobuf 的文档为什么没有提我写的那种自动反射类型的方法。

22楼 felix0730 2011-04-06 22:23发表

[e03][e03][e03]

21楼 solo_coder 2011-04-06 22:18发表

int型的typeID方式看上去不时髦，但是其实没有比这运行效率再高的法子了（使用指针数组，解析一条消息就是常量时间，没有更加优化的法子了）

20楼 stone__liu 2011-04-06 19:33发表

[e03][e03][e03][e03][e03][e03][e03]

不懂呃。。。。

19楼 zwb10086 2011-04-06 17:17发表

[e08]

Re: zwb10086 2011-04-06 17:18发表

回复 zwb10086：这也太菜了

18楼 zhaofeng_001 2011-04-06 17:09发表

完全不懂Google Protocol Buffers的路过，嘿嘿[e04]

17楼 benjiam 2011-04-06 13:34发表

void protobuf_RegisterTypes(const ::std::string&) {
}
做的就是这个事情，没什么高端的，比山寨的高端一点，但都是一路货色。一个靠protoc 做的，一个自己写。自动 还算不上！

16楼 benjiam 2011-04-06 13:30发表

否则a b 通信,a 端修改了协议，然后传送一个新的类给b b 能解析吗？ b 仍然不能， 因为protobuff 不是自解析的

15楼 benjiam 2011-04-06 13:30发表

：Amazon S3 就遇到过，所以他们吸取教训加了 check sum
也只是降低了出现概率的。
你的做法其实和 山寨做法 并无区别。
唯一的区别就是 靠google protbuff 帮你完成了类的注册,解析等方法。

14楼 chjttony 2011-04-06 12:46发表

[e01]

13楼 kanchangcheng 2011-04-06 10:47发表

[e01]

12楼 良少 2011-04-06 10:45发表

我觉得json协议比较好。

11楼 hzhxxx 2011-04-06 09:51发表

[e01][e02][e03]

10楼 healer_kx 2011-04-06 09:36发表

古之人不余欺也 :-) ==》 Google之人不余欺也 :-)

9楼 Jiang 2011-04-06 09:04发表

[e03][e03][e03][e03][e03][e03][e03][e03][e03][e02][e02][e01]

8楼 Blue_may 2011-04-06 08:46发表

可以用google::protobuf::io::QueueInputStream封装一个解码器，这样就不用关心报文具体长度，收多少都往里丢，让解码器自己判断解码成功还是需要保留等待其他数据。

Re: wwooha 2011-04-09 11:22发表

回复 Blue_may：我猜你是继承ZeroCopyInputStream自己实现的是吧？

Re: 陈硕 2011-04-06 18:21发表

回复 Blue_may： 哪里有 QueueInputStream？

7楼 hzminghe4 2011-04-04 20:10发表

还是 待续 再说吧

Re: 陈硕 2011-04-06 12:46发表

回复 hzminghe4：我删除了您的一条评论，评论内容是：应用层校验，可以可以作为一个可选参数吧，看具体应用，加了校验，也可能碰撞，只不过出错概率大大降低，……

6楼 hzminghe4 2011-04-04 20:08发表

xxConnection/xxSession是一台根据不同的message做出相应状态转移的状态机， 我想用 mfc消息映射，填表的方式可能跟靠谱一点，添加一个消息，只需增加一个表项和一个成员函数。

5楼 hzminghe4 2011-04-04 20:08发表

onQuery(muduo::Query* query), onAnswer(muduo::Answer* answer)是 xxConnection/xxSession 的成员函数，

4楼 hzminghe4 2011-04-04 20:08发表

服务端处理的是xxConnection/xxSession, dispatcher得跟 xxConnection/xxSession 结合起来

3楼 hzminghe4 2011-04-04 20:08发表

我想了一下，也看了你关于dispatcher部分的代码，还是觉得不不太对劲，反射固然好，但到手只是一个Message基类，归根结底还得再 dynamic_cast， 已经突破我的底线

Re: 陈硕 2011-04-04 20:58发表

回复 hzminghe4：实在看不惯 dynamic_cast 那就换成 static_cast 呗，程序一样用。

Re: hzminghe4 2011-04-04 21:09发表

回复 Solstice：可能是看不惯吧，因为都是编译时模板推导，两个类是明确的继承关系，怎么转换都是安全的。

2楼 hzminghe4 2011-04-03 18:00发表

需要启用 RTTI吗？
这是个问题， Google C++ Style Guide本身就不主张.

Re: 陈硕 2011-04-03 20:03发表

回复 hzminghe4：G++ 默认是启用 RTTI 的，除非你手动关闭它。Google建议：Do not use dynamic_cast except in test code. Do not use RTTI, except in unittests.

1楼 hzminghe4 2011-04-03 17:47发表

好！这也证明智慧分享的重要性，闭门造车的可怕性！