## Virtual box共享文件夹及自动挂载

[**https://blog.csdn.net/Glow\_Y/article/details/68237640**](https://blog.csdn.net/Glow_Y/article/details/68237640)

**devices->insert guest additions cd images**

**mount -t vboxsf 挂载文件名 挂载点：mount -t vboxsf Downloads /mnt/share**

## 安装

1. **下载：<https://github.com/google/protobuf/releases/tag/v3.5.1>**
2. **解压：tar -zxvf xx.tar.gz**
3. **./configure**
4. **make**
5. **make check**
6. **make install**

## 程序演示

<https://www.ibm.com/developerworks/cn/linux/l-cn-gpb/>

|  |
| --- |
| 编写proto文件：  package lm;  message helloworld  {     required int32     id = 1;  // ID     required string    str = 2;  // str     optional int32     opt = 3;  //optional field  } |
| 编译proto：  protoc -I=$SRC\_DIR --cpp\_out=$DST\_DIR $SRC\_DIR/mes.proto  生成两个文件：  mes(文件名).helloworld（message名）.pb.cc mes(文件名).helloworld（message名）.pb.h |
| 编写write和read  #include "mes.helloworld.pb.h"  …     int main(void)   {      lm::helloworld msg1;    msg1.set\_id(101);    msg1.set\_str(“hello”);      // Write the new address book back to disk.    fstream output("./log", ios::out | ios::trunc | ios::binary);      if (!msg1.SerializeToOstream(&output)) {        cerr << "Failed to write msg." << endl;        return -1;    }    return 0;   }  #include "mes.helloworld.pb.h"  …   void ListMsg(const lm::helloworld & msg) {    cout << msg.id() << endl;    cout << msg.str() << endl;   }     int main(int argc, char\* argv[]) {      lm::helloworld msg1;      {      fstream input("./log", ios::in | ios::binary);      if (!msg1.ParseFromIstream(&input)) {        cerr << "Failed to parse address book." << endl;        return -1;      }    }      ListMsg(msg1);    …   } |
| 编译生成可执行文件：  g++ write.cpp mes.helloworld.pb.cc -o write  g++ read.cpp mes.helloworld.pb.cc -o read |

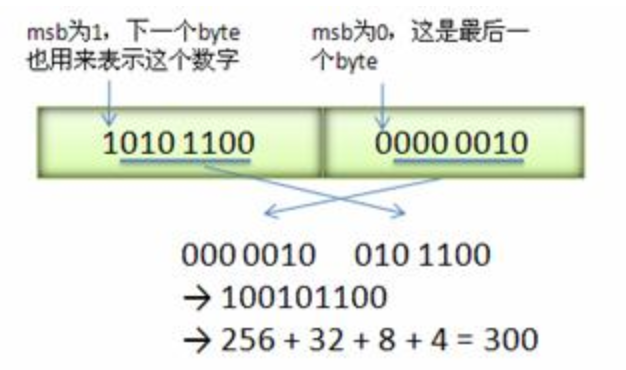
## 高级应用

### 动态编译（待学习）

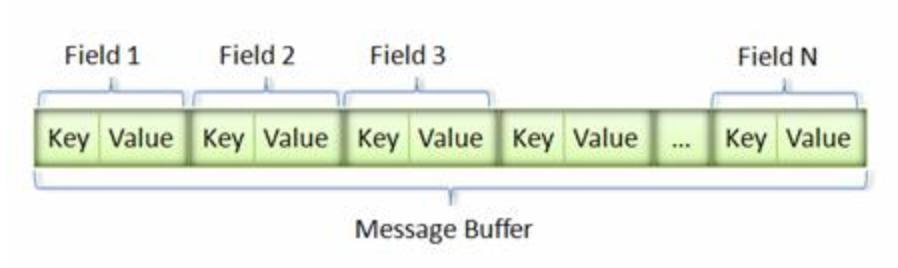
无法预知.proto文件时，需要动态处理。使用google::protobuf::compiler包，主要使用importer类，定义在importer.h中。Importer类主要包含三个对象：错误处理MultiFileErrorCollector类，定义.proto文件源目录的SourceTree类

## 序列化原理

Virant数字表示法：每个比特最高位有特殊含义，若为1，表示后续的byte也是该数的一部分，若为0则结束。其他7bit都表示数字。比如300（0000 0001 0010 1100），从右到左7位一分隔（1010 1100 ，00000 010），



Message存储是一系列key-value形式存储：



举例：

Msg1.id=101

Msg1.str=“hello”

Vim -b Log

%！xxd：

0000 1000 0110 0101 0001 0010 0000 0101 0110 1000 0101 0101 0110 1100 0110 1100 0110 1111

最终会有两个k-v对，一个对应id，一个对应str。

Key定义：(filed\_number<<3)|write\_type，比如lm.helloworld中id的field\_number是1.第二部分是write\_type，表示value传输类型。

|  |  |  |
| --- | --- | --- |
| 0 | Varint | Int32、int64、uint32、uint64、sint32、sint64，bool、enum |
| 1 | 64-bit | Fixed64 、sfixed64、double |
| 2 | Length-delimi | String、bytes、embeded message、packed repeated fields |
| 3 | Start group | 已移除 |
| 4 | End group | 已移除 |
| 5 | 32-bit | Fixed32，sfixed32，float |

Sint使用zigzag编码，因为负数一般会被表示成很大的整数，使用zigzag，绝对值小的数，无论正负都可以用较少的byte表示。

Lengtg-delimi采用数据库中varchar的表示方法，即用一个varint表示length，然后其余部分紧跟在这个长度之后。

## 反序列化原理

将二进制流读入内存，使用简单的数学运算完成，无需复杂的词法语法分析。

## 出错

1. **protoc：error while loading shared libraries：libprotobuf.so.15 can not open shared object file: no such file or directory**

**原因是protobuf的连接库默认安装路径是/usr/local/lib，不是linux连接库的LD\_LIBRARY\_PATH路径这个环境变量中，所以找不到lib。**

**解决：<https://stackoverflow.com/questions/25518701/protobuf-cannot-find-shared-libraries>**

[**https://blog.csdn.net/k346k346/article/details/51754431**](https://blog.csdn.net/k346k346/article/details/51754431)

**export LD\_LIBRARY\_PATH=/usr/local/lib**

1. **g++时出现大量未定义**

**解决：<http://www.cnblogs.com/darkknightzh/p/5804395.html>**

**g++ x.cpp xx.cc -o x -lprotobuf**

## c++ Reference

### message

1. 所有的类继承google::protobuf::Message
2. 派生类默认实现所有的方法以达到最快的速度

如果有option optimize\_for = CODE\_SIZE：派生类重最少的方法减少代码size

如果有option optimize\_for = LITE\_RUNTIME：将会实现google::protobuf::MessafeLite接口，该接口只包含Message的一部分方法，连接libprotobuf-lie.so，适用于移动设备。

1. 除了继承而来的方法，派生类还定义了如下方法：

* Foo（）：构造函数
* ～Foo（）
* Foo（const Foo& other）：拷贝构造函数
* Foo& operator=（const Foo& other）：赋值操作符
* Void swap（Foo\* other）：交换函数
* Const UnknownFieldSet& unknown\_fields() const:返回不能识别的字段
* UnknownFieldSet\* mutable\_unknown\_fields():返回指向mutable的未知字段的指针
* Static const Descriptor\* descriptor():返回类型描述，包括了各字段和其类型信息。
* Static const Foo& default\_instance():返回const 的单例对象

1. 嵌套类

Message Foo { message Bar{ }}

编译生成的类位Foo和Foo\_Bar，Foo类中有typedef Foo\_Bar Bar，可以把Bar当作嵌套类使用

### Field

比如message helloworld{required int32 id = 1}对应到c++中是static const int kIdFiedlNumber = 1;

### Any

### Oneof

## C++ arena allocation guide

### 作用

1. 内存的申请和释放花费大量cpu时间，arena用来减少这种性能消耗。
2. 设置缓存

### 开始

需要在proto文件中加入option cc\_enable\_arenas = true

### API

## 源码剖析