Google Protocol Buffer的原理和使用方法

1 概述

Google Protocol Buffer(简称 Protobuf) 是Google公司内部的混合语言数据标准，目前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。

Protocol Buffers是一种轻便高效的结构化数据存储格式，可以用于结构化数据串行化，或者说序列化。它很适合做数据存储或 RPC 数据交换格式。可用于通讯协议、数据存储等领域的语言无关、平台无关、可扩展的序列化结构数据格式。目前提供了C++、Java、Python 三种语言的 API。

2 编译安装

```
# 在该地址中下载release版本，并解压
https://github.com/protocolbuffers/protobuf/releases
./configure --prefix=/usr/local
make -j 100
sudo make install -j 100
sudo ldconfig   #刷新so库所在的目录，如果configure目录选择/usr/local，那么so库将被安装在/usr/local/lib目录下
```

3 使用案例

案例描述：该程序分为两个部分：分别是Writer和Reader，Writer向磁盘中写入数据，Reader中磁盘中读取数据并打印到屏幕上。

- 书写.proto文件

首先我们需要编写一个 proto 文件，定义我们程序中需要处理的结构化数据，在

protobuf 的术语中，结构化数据被称为 Message。proto 文件非常类似 java 或者 C 语言的数据定义。

```
syntax = "proto2";
package lm;
message helloworld {
  required string id = 1;
  required string str = 2;
  optional int32 opt  = 3;
}
```

- 编译.proto文件

```
protoc -I=. --cpp_out=. ./lm.helloworld.proto   #编译.proto文件，其中-I目录指定.proto所在路径，-cpp_out指定输出文件路径 ./lm.helloworld.proto表示.proto所在路径，编译之后会生成.h和.cc文件，如图1
```

图1 生成.h和.cc文件

- 编写Writer和Reader函数

```
//writer.cpp
#include "lm.helloworld.pb.h"
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main(void){

  lm::helloworld msg1;    //创建一个对象，类名为包名::消息名
  msg1.set_id(101);  //设置消息中的id值
  msg1.set_str("Hello World!");  //设置消息中的string值

  // Write the new address book back to disk.
  fstream output("./log", ios::out | ios::trunc | ios::binary); //将消息写出到.log文件中
```

```
15
16    if (!msg1.SerializeToOstream(&output)) {    //序列化写出
17        cerr << "Failed to write msg." << endl; //如果写出错误，则输出错误提示
18        return -1;
19    }
20    return 0;
21  }
```

```
1  //reader.cpp
2  #include "lm.helloworld.pb.h"
3  #include <iostream>
4  #include <fstream>
5  #include <string>
6  using namespace std;
7  void ListMsg(const lm::helloworld & msg) {
8    cout << msg.id() << endl; //打印出id值
9    cout << msg.str() << endl;   //打印出str值
10 }
11
12  int main(int argc, char* argv[]) {
13
14    lm::helloworld msg1;
15
16    {
17      fstream input("./log", ios::in | ios::binary); //从./log文件中建立输入流
18      if (!msg1.ParseFromIstream(&input)) {   //解析输入流
19        cerr << "Failed to parse address book." << endl;
20        return -1;
21      }
22    }
23
24    ListMsg(msg1);
25  }
```

- 编译.cpp文件

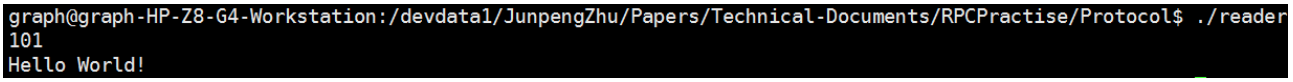
```
```

```
g++ -std=c++11 -I/usr/local/include -L/usr/local/lib -o writer writer.cpp lm.helloworld.pb.cc -lprotobuf -pthread # 编译writer.cpp文件
g++ -std=c++11 -I/usr/local/include -L/usr/local/lib -o reader reader.cpp lm.helloworld.pb.cc -lprotobuf -pthread #编译reader.cpp文件
```

注意：在编译上述两个文件时，一定要加上lm.helloworld.pb.cc文件，否则会显示找不到函数

- 检验结果

```
./writer   #运行writer，将结果写入到log文件中
./reader   #运行reader，从log中读出内容，结果如图2
```



从log中读出序列化后的内容

4 参考文献

1. 谷歌官方使用demo https://developers.google.com/protocol-buffers/docs/cpptutorial
2. 编译Protocol Buffer时应该正确的编译选项 https://stackoverflow.com/questions/10404027/cant-compile-example-from-google-protocol-buffers