


aaron20127的ChinaUnix博客

首页 | 博文目录 | 关于我



aaron20127

博文语言: 6027

博文数量: 82

博客积分: 0

博客等级: 民兵

技术积分: 482

用户 组: 普通用户

注册时间: 2016-06-13 10:58

认证徽章: [去认证会员](#)

加关注

短消息

论坛

加好友

- 文章分类
- 全部博文 (82)

机器人 (1)

文集 (1)

安装包 (2)

编译 (3)

编程 (1)

CentOS (5)

tcp/ip (3)

正则表达式 (1)

下载 (1)

C++ (5)

tools (3)

openWrt (7)

C (13)

Linux (11)

Linux (25)

未分配的博文 (0)

- 文章存档
- 2018年 (2)

2017年 (9)

2016年 (71)

我的朋友

最近访客

11937841

toddhai

1404

ysl115

王傲_n

gqifree

张圣博

pinkhal

gqiyb

- 推荐博文
- [XIV](#)

• [CSS3盒模型](#)

• [Kafka 跨集群同步方案](#)

• [kali linux 安装keybase 并...](#)

• [Oracle 11g R2-RAC+ASM+Oracle...](#)

- 相关博文
- [Linux基础命令——service](#)

• [Linux从初学到高级全套视频...](#)

• [Linux一些命令总结](#)

• [阿里中间件开源组件: Sentinel...](#)

• [Linux基础命令——last](#)

• [Linux基础命令——arch](#)

• [Linux基础命令——eject](#)

• [Linux基础命令——hwclock](#)

• [Linux基础命令——lspci](#)

• [Linux基础命令——lssub](#)

C语言中使用protocolbuffer

分类: C/C++ 2016-12-25 17:06:53

protocolbuffer(以下简称PB)是google 的一种数据交换的格式,它独立于语言,独立于平台。google 提供了多种语言的实现: java, c#, c++, go 和 python,每一种实现都包含了相应语言的编译器以及库文件。由于它是一种二进制的格式,比使用xml进行数据交换快许多,可以把它用于分布式应用之间的数据通信或者异构环境下的数据交换。作为一种效率和兼容性都很优秀的二进制数据传输格式,可以用于诸如网络传输、配置文件、数据存储等诸多领域。

最近有个项目在c语言中用到了protobuf,所以在这里记录一下protobuf的安装和使用方法,以备后用。

1. 下载和编译安装包

要使用c语言版的protobuf需要下载两个安装包protobuf-2.6.1.tar.gz (protobuf库) 和 protobuf-c-1.1.1.tar.gz (可生成c语言格式的protobuf安装包)。

protobuf库现在已更新到protobuf-3.1.0.tar.gz,其中支持C++, Java, Python, C#, JavaNmo, JavaScript, Ruby, Go, Objective-c。可以发现原生库并不支持C语言,所以需要安装生成 protobuf-c 的安装包。

(1) 下载

protobuf库下载地址: <https://github.com/google/protobuf/releases>

protobuf-c下载地址: <https://github.com/protobuf-c/protobuf-c/releases>

(2) 编译 (以protobuf-2.6.1.tar.gz和protobuf-c-1.1.1.tar.gz为例)

```
root@192 ~# tar xvf protobuf-2.6.1.tar.gz
root@192 ~# cd protobuf-2.6.1 -prefix/usr
root@192 ~# ./configure
root@192 ~# make
```

编译protobuf-c-1.1.1

```
root@192 ~# tar xvf protobuf-c-1.1.1.tar.gz
root@192 ~# cd protobuf-c-1.1.1
root@192 ~# ./configure --prefix=/usr protobuf_CFLAGS=-I/home/aaron/Study/protobuf/protobuf-2.6.1/src LDFLAGS=-L/home/aaron/Study/protobuf/protobuf-2.6.1/lib -lprotobuf -lprotobuf_PR0T
root@192 ~# make
```

编译完成后会在protobuf-c-1.1.1/protoc-c/目录下生成protoc-c可执行程序,这个程序是用来通过 proto文件生成pb-c.c和pb-c.h文件的。注意,由于以上两个包没有安装,执行protoc-c时,应该执行protobuf-c-1.1.1/protoc-c/目录下的protoc-c;如果要安装的话两个包都安装,make install即可,这时可以直接运行protoc-c命令的;执行protoc-c命令时需要加载libprotobuf.so.9动态库,该库在protobuf-2.6.1/src/.libs目录下,执行失败证明没有正确地加载该动态库路径。

2 .proto格式文件

.proto文件是用来交换数据的格式,你需要使用protobuf打包的数据结构都应该先写在.proto文件中。.proto文件中的数据和c语言中的数据结构有点不一样,具体来看一下边这个例子。

文件名: game.proto

```
/**
 * 角色的个人信息
 */
message personal_info {
    required string name = 1; /* 角色名字 */
    required int32 year = 2; /* 角色年龄 */
    required string weapon = 3; /* 武器 */
    repeated string armor = 4; /* 防具 */
    optional int32 money = 5; /* 携带的钱 */
}

/**
 * 团队信息
 */
message team_info {
    required personal_info leader = 1; /* leader */
    optional personal_info aaron = 2; /* 队员 */
}
```

比如我现在要使用protobuf传输一个游戏团队的基本信息,于是我在game.proto文件中写了两个message结构体。message team_info表示团队成员,leader是必须的,所以他的类型是required。而队员aaron可能不存在,所以他的类型是optional。message personal_info表示每个成员的基本信息。名字name必须是字符串,所以name的类型是required string, string表示字符串,而不是c语言中使用char*表示。年龄是int整型,用int32表示。再看一 Farmor, 它的类型是repeated,表示是一个数组。每个数组成员是string类型,这意思是说防具可能多个。而money可能没有,所以用optional表示,当然money的值为0,也可以表示没有, 这里设置成optional只是做的一个例子,并不是很合理。

更多参考: <https://github.com/protobuf-c/protobuf-c>

1.3 生成并使用.pb-c.c和.pb-c.h文件

(1) 通过使用protoc-c命令使game.proto文件生成相应的c文件。

```
root@192 ~# ./protobuf-c-1.1.1/protoc-c/protoc-c --c_out=./ game.proto
```

执行完上边的命令后,可以在game.proto所在的文件夹中看见game.pb-c.c和game.pb-c.h两个文件,编写代码时就会用到这两个文件中的结构体。

(2) 使用.pb-c.c和.pb-c.h文件

除了生成的game.pb-c.c和game.pb-c.h会用到,还需要用到protobuf-c-1.1.1/protobuf-c路径下的 protobuf-c.h, game.pb-c.c,和game.pb-c.h共5个文件。

首先需要将game.pb-c.h中的#include 修改成#include "protobuf-c.h"。接下来直接看main.c

文件名: main.c

```
1. #include <stdio.h>
2. #include <string.h>
3. #include <stdlib.h>
4. #include "game.pb-c.h"
5.
6. static void proto_store_string(
7.     char **addr,
8.     const char * str,
9.     int max_str_len)
10. {
11.     *addr = (char *)malloc(max_str_len);
12.     strcpy(*addr, str, max_str_len);
13.     (*addr)[max_str_len - 1] = '\0';
14. }
15.
16. int protobuf_game_pack(
17.     unsigned char * out_buff,
18.     int buff_len)
19. {
20.     /** 1.Assembly and packaging data */
21.     #define STR_LEN 20
22.
23.     /**
24.      * 1.初始化方式有两种,一种是定义变量时,如下边的TEAM_INFO_INIT;
25.      * 另一种是变量已经定义好了,则需要使用函数初始化,superpersonal_info_init().
26.      * 2.每一句初始化都必须的初始化之后才能使用,所有的初始化生成或数据都类似,pc-c.h中.
27.      * 3.如果结构体成员是指向结构体的指针,则需要为其动态分配空间,原因是栈上的空间有可能不足,
28.      * 堆上分配空间更安全,如下边的teaminfo.leader.
```

```
29.  */
30.  teaminfo teaminfo = TEAM_INFO_INIT;
31.  teaminfo.leader = (PersonalInfo *)malloc(sizeof(PersonalInfo));
32.  teaminfo.aaron = (PersonalInfo *)malloc(sizeof(PersonalInfo));
33.  personal_info_init(teaminfo.leader);
34.  personal_info_init(teaminfo.aaron);
35.
36.  /**
37.   * 1.对于string成员也需要动态分配空间,如teaminfo.leader->name
38.   * 2.对于数组成员,常常是二维指针,也需要分配空间,如teaminfo.leader->armor;
39.   * 在结构体成员中也多出了1~n_armor成员,该成员需要输入数组的长度,方便打包时确认.
40.   * 3.对于整型成员,如teaminfo.leader->money,发现它也多出了一个has_money成员,因为
41.   * 在game.proto中我们对money定义的属性是optional,如果有has_money=0,表示money成员没有数据;
42.   * 反之,若money成员有数据.
43.   * 4.对于string结构体等成员,不管他们的类型是否是optional的,都不需要增加多余的成员.原因是这些成
44.   * 员在结构体中是由指针表示的,可以直接检查该指针是否为空来判断该成员是否有数据.例如teaminfo.aaron
45.   */
46.  /** Assemble leader */
47.  proto_store_string(&teaminfo.leader->name, "leader", STR_LEN);
48.  teaminfo.leader->year = 20;
49.  proto_store_string(&teaminfo.leader->weapon, "Lightning sword", STR_LEN);
50.  teaminfo.leader->n_armor = 3;
51.  teaminfo.leader->armor =
52.    (char **)malloc(sizeof(char *) * teaminfo.leader->n_armor);
53.  proto_store_string(&teaminfo.leader->armor[0], "Lightning armor", STR_LEN);
54.  proto_store_string(&teaminfo.leader->armor[1], "Lightning pants", STR_LEN);
55.  proto_store_string(&teaminfo.leader->armor[2], "Lightning boots", STR_LEN);
56.  teaminfo.leader->has_money = 1;
57.  teaminfo.leader->money = 1000;
58.
59.  /** Assemble aaron */
60.  proto_store_string(&teaminfo.aaron->name, "aaron", STR_LEN);
61.  teaminfo.aaron->year = 18;
62.  proto_store_string(&teaminfo.aaron->weapon, "Fire sword", STR_LEN);
63.  teaminfo.aaron->n_armor = 3;
64.  teaminfo.aaron->armor =
65.    (char **)malloc(sizeof(char *) * teaminfo.aaron->n_armor);
66.  proto_store_string(&teaminfo.aaron->armor[0], "Fire armor", STR_LEN);
67.  proto_store_string(&teaminfo.aaron->armor[1], "Fire pants", STR_LEN);
68.  proto_store_string(&teaminfo.aaron->armor[2], "Fire boots", STR_LEN);
69.  teaminfo.aaron->has_money = 0;
70.
71.  /**
72.   * 1.结构体填充完数据后,我们需要将数据编码打包并复制到缓冲上.
73.   * 2.在数据进行打包之前需要判断打包后数据的长度是否比缓冲区大.
74.   * 这时可以用结构体对应的判断数据长度的函数,如team_info_get_packed_size().
75.   * 3.确认打包后长度确实比可用的缓冲区小时,则可以直接调用结构体的打包函数将数据打包到
76.   * 缓冲区上.如team_info_pack().
77.   */
78.  /** packaging data */
79.  int len = team_info_get_packed_size(&teaminfo);
80.
81.  if (len > buff_len) {
82.    printf("not enough memory.\n");
83.    len = 0;
84.  } else {
85.    if (len < team_info_pack(&teaminfo, out_buff)) {
86.      len = 0;
87.    }
88.  }
89.
90.  /**
91.   * 1.释放动态分配的空间.
92.   */
93.  /** free */
94.  int i = 0;
95.  for (; i < teaminfo.leader->n_armor; i++) {
96.    free(teaminfo.leader->armor[i]);
97.  }
98.  free(teaminfo.leader->armor);
99.  free(teaminfo.leader->weapon);
100. free(teaminfo.leader->name);
101. free(teaminfo.leader);
102.
103.
104. i = 0;
105. for (; i < teaminfo.aaron->n_armor; i++) {
106.   free(teaminfo.aaron->armor[i]);
107. }
108. free(teaminfo.aaron->armor);
109. free(teaminfo.aaron->weapon);
110. free(teaminfo.aaron->name);
111. free(teaminfo.aaron);
112.
113. return len;
114. }
115.
116. int protobuf_game_parse(
117.   unsigned char * in_buff,
118.   int buff_len)
119. {
120.   /**
121.    * 调用解包函数,得到结构体指针
122.    */
123.   /** parse data */
124.   TeamInfo * teaminfo = team_info_unpack(NULL, buff_len, in_buff);
125.
126.   /**
127.    * 获取数据,只要成员是指针都必须判断该指针是否为空,optional的其它类型也需要进行判断
128.    */
129.   if (teaminfo->leader) {
130.     PersonalInfo *leader = teaminfo->leader;
131.     printf("leader->name [%s]\n", leader->name);
132.     printf("leader->year [%d]\n", leader->year);
133.     printf("leader->weapon [%s]\n", leader->weapon);
134.
135.     int i = 0;
136.     for (; i < leader->n_armor; i++) { //数组需要判断长度
137.       printf("leader->armor[%d] [%s]\n", i, leader->armor[i]);
138.     }
139.
140.     if (leader->has_money) { //判断是否有money
141.       printf("leader->money [%d]\n", leader->money);
142.     }
143.   }
144.
145.   printf("\n");
146.
147.   if (teaminfo->aaron) {
148.     PersonalInfo *aaron = teaminfo->aaron;
149.     printf("aaron->name [%s]\n", aaron->name);
150.     printf("aaron->year [%d]\n", aaron->year);
151.     printf("aaron->weapon [%s]\n", aaron->weapon);
152.
153.     int i = 0;
154.     for (; i < aaron->n_armor; i++) {
155.       printf("aaron->armor[%d] [%s]\n", i, aaron->armor[i]);
156.     }
157.
158.     if (aaron->has_money) {
159.       printf("leader->money [%d]\n", aaron->money);
160.     }
161.   }
162.
163.   /** free memory */
164.   team_info_free_unpacked(teaminfo, NULL);
165.   return 0;
166. }
167.
168. void proto_test_demo(void)
169. {
170.   #define BUFF_LEN 1024
171.   int len = 0;
172.   int i = 0;
173.   /** 打包数据 */
174.   len = protobuf_game_pack(buff, BUFF_LEN);
175.
176.   /** 解包数据 */
177.   protobuf_game_parse(buff, len);
178. }
179.
180.
181. int main(int argc, char **argv)
182. {
183.   proto_test_demo();
184. }
```

运行结果:

```
[root@i92 test_game]# cc -o main game.pb-c.c main.c protobuf-c.c
[root@i92 test_game]# ./main
leader->name [leader]
leader->year [20]
leader->weapon [Lightning sword]
leader->armor[0] [Lightning armor]
leader->armor[1] [Lightning pants]
leader->armor[2] [Lightning boots]
leader->money [1000]

aaron->name [aaron]
aaron->year [18]
aaron->weapon [Fire sword]
aaron->armor[0] [Fire armor]
aaron->armor[1] [Fire pants]
aaron->armor[2] [Fire boots]
```

通过理解以上代码基本上可以使用protobuf-c了,更多的细节请参考: <https://github.com/protobuf-c/protobuf-c>

1.4 注意事项

1. 大端机编译protobuf-c.c时要定义WORDS_BIGENDIAN宏，不然float类型的打包会有问题。
2. protobuf-c的打包采用了zigzag编码，压缩了数据。可以参靠下边这个链接，学习一下zigzag编码原理。

<http://blog.csdn.net/zgwangbo/article/details/51590186>

阅读 (825) | 评论 (0) | 转发 (0) |

上一篇: c语言源码、反码和补码和符号变量移位
下一篇: gcc编译时去掉__LINE__的绝对路径

1

给主人留下些什么吧! ^^

评论热议

请登录后评论。
[登录](#) [注册](#)

