1 实验环境配置

实验环境：Ubuntu20.04，protobuf-all-3.11.4，jansson-2.12

1.1 protobuf 的安装

（1）安装相关工具包。

> sudo apt-get install autoconf automake libtool curl make g++ unzip

执行完上述命令后，采用 gcc –version 查看 gcc 版本。要求 gcc 版本必须在 6.0 及以上才可以。若采用 Ubuntu16.04，执行该命令后，gcc 版本无法达到要求，需额外更新 gcc 版本。

（2）获取 protobuf 的压缩包。

> git clone https://github.com/protocolbuffers/protobuf.git

> cd protobuf

> git submodule update --init --recursive

> ./autogen.sh

执行上述命令时若下载速度缓慢，可更换软件源或者直接通过官方网站下载压缩包，若采取从官网直接下载的方式，则略过上述命令。Protobuf 版本无要求，2.X 与 3.X 均可。如果只需要 C ++，下载 protobuf-cpp-[VERSION].tar.gz（本实验中仅需要 C++即可）。

（3）运行配置文件并安装。

> ./configure

> make

> make check

> sudo make install

> sudo ldconfig # refresh shared library cache.

注意在有些机器上 make check 命令可能会失败，但仍然可以安装，但是该库的某些功能可能无法在您的系统上正常工作。

部分安装过程如图 1-1、1-2 所示：

```
root@ubuntu:/home/hui# git clone https://github.com/protocolbuffers/protobuf.git
Cloning into 'protobuf'...
remote: Enumerating objects: 33, done.
remote: Counting objects: 100% (33/33), done.
remote: Compressing objects: 100% (28/28), done.
Receiving objects: 100% (72268/72268), 59.35 MiB | 16.00 KiB/s, done.
remote: Total 72268 (delta 11), reused 11 (delta 2), pack-reused 72235
Resolving deltas: 100% (49639/49639), done.
Updating files: 100% (2392/2392), done.
```

图 1-1 protobuf 安装过程

```
root@ubuntu:/home/hui/protobuf# make install
Making install in .
make[1]: Entering directory '/home/hui/protobuf'
make[2]: Entering directory '/home/hui/protobuf'
make[2]: Nothing to be done for 'install-exec-am'.
 /usr/bin/mkdir -p '/usr/local/lib/pkgconfig'
 /usr/bin/install -c -m 644 protobuf.pc protobuf-lite.pc '/usr/local/lib/pkgconfig'
```

图 1-2 protobuf 安装过程 2

1.2 jansson 的安装

（1）获取 jansson 的压缩包，解压并进入文件夹。

可直接通过官方网站下载压缩包，jansson 版本要求 2.12 及以上，若采用 jansson-2.1 版本，会由于与 gcc9.3.0 版本不兼容，安装过程报错的问题。

> wget "http://www.digip.org/jansson/releases/jansson-2.12.tar.gz "

> tar -zxvf jansson-2.12.tar.gz

> cd jansson-2.12

（2）运行配置文件并安装

> ./configure

> make

> sudo make install

部分安装过程如图 1-3、1-4 所示：

```
root@ubuntu:/home/hui/jansson-2.12# ./configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /usr/bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
```

图 1-3 jansson 安装过程(1)

```
root@ubuntu:/home/hui/jansson-2.12#  make install
Making install in doc
make[1]: Entering directory '/home/hui/jansson-2.12/doc'
make[2]: Entering directory '/home/hui/jansson-2.12/doc'
make[2]: Nothing to be done for 'install-exec-am'.
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/home/hui/jansson-2.12/doc'
make[1]: Leaving directory '/home/hui/jansson-2.12/doc'
```

图 1-4 jansson 安装过程(2)

2 PB 文件读取（代码实现及接口说明）

函数声明：vector<string> readBinaryData(const string &filePath)

函数参数：序列化二进制文件的路径。

函数功能：读取二进制文件，返回多条二进制数据组成的 vector。首先获取整个

文件的长度，开辟对应大小的缓冲区再进行文件的读入。
核心代码如下：

```
ifstream fin(filePath, ios::in | ios::binary);
fin.seekg(0, std::ios::end);        // go to the end
length = fin.tellg();               // report location (this is the length)
fin.seekg(0, std::ios::beg);        // go back to the beginning
buffer = new char[length];          // allocate memory for a buffer of appropriate dimension
fin.read(buffer, length);           // read the whole file into the buffer
fin.close();                        // close file handle
```

3 动态解析方法（代码实现及接口说明）

（1）string analyPackage(string protoPath);

函数参数：proto 文件的路径。

函数功能：根据 proto 文件解析出对应的包名，若没有则返回空。如果 proto 文件中使用了 package 语句，则需要使用对应的 package_name 才能访问其内部的 message 类型。注意解析时首先应对程序中的注释进行相应的处理，proto 文件中的注释采用双斜杠(//)语法格式。采取正则表达式的方式来实现：

```
regex reg("package\\s+.*?\\s*;");
smatch m;
auto ret = regex_search(content, m, reg);
```

（2）string analyMessage(string protoPath);

函数参数：proto 文件的路径。

函数功能：保证程序的健壮性。我们要求序列化的文件一定要有分隔符，若没有检测到分隔符，则默认该条数据对应 proto 文件的第一个 Message 类型，此时需要从 proto 文件解析其第一条 Message 类型。同样首先对文档注释进行处理，采取正则表达式的方式来实现：

```
regex reg("message\\s+.*?\\s*\\{");
smatch m;
auto ret = regex_search(content, m, reg);
```

（3）string regexClassName(string p_str);

函数参数：给定的字符串（含有自定义的分隔符）。

函数功能：从分隔符中获取二进制数据所对应的类型。采取正则表达式的方式来实现：

```
regex reg("\\^@UCAS@(.*)\\^");
smatch m;
auto ret = regex_search(p_str, m, reg);
```

（4）vector<string> split(const string &str， const string &delim);

函数参数：待分割字符串 str；分割符（串）delim。

函数功能：对字符串 str 按照给定的分割串进行划分，返回结果对应的 vector。在 analyPackage、analyMessage 时均要使用此函数。注意先将要切割的字符串从 string 类型转换为 char*类型，再进行操作。

（5）int dynamicParseFromProtoFile(const string &filePath, const string &messageName, function<void(::google::protobuf::Message *msg)> callBack)

函数参数：proto 文件的路径；proto 文件中第一个 message 类型；回调函数。

函数功能及说明：对给定的一条二进制数据 msg 进行动态解析，若某条二进制数据不含有分隔符，则默认其对应 proto 文件的第一个 message 类型。回调函数就是一个通过函数指针调用的函数。回调函数不是由该函数的实现方直接调用，而是在特定的事件或条件发生时由另外的一方调用的，用于对该事件或条件进行响应。本函数的第三个参数为一个回调函数，该函数的参数为一个::google::protobuf::Message 类型的指针，用于承接解析出来的 pb 数据。由于 C++具有多态性，因此任一子对象的引用可以赋值给父类对象的指针。

核心代码如下：

```
// 进行文件系统的配置
::google::protobuf::compiler::DiskSourceTree sourceTree;
// 将当前路径映射为项目根目录
sourceTree.MapPath("", path);
// 进行动态编译器的配置
::google::protobuf::compiler::Importer importer(&sourceTree, NULL);
// 动态编译 proto 源文件
importer.Import(fileName);
// 从编译器中提取具体 Message 类型的描述信息.
const ::google::protobuf::Descriptor *descriptor =
    importer.pool()->FindMessageTypeByName(messageName);
// 创建一个动态的消息工厂
::google::protobuf::DynamicMessageFactory factory;
// 从消息工厂中创建出一个类型原型.
const ::google::protobuf::Message *message = factory.GetPrototype(descriptor);
// 构造一个可用的消息.
::google::protobuf::Message *msg = message->New();
// 之后即可通过反射接口来对字段进行赋值等相关操作
```

4 转换方法（代码实现及接口说明）

本模块最外层实现了两个接口函数，在不同的条件下调用。

（1）char *pb2json(Message *msg, char *buffer)

函数参数：空消息对象 msg，二进制数据缓冲区 buffer

函数功能：适用于实参是二进制数据 buffer，在函数里先进行反序列化成 pb 数据，保存在提供的空 Message 对象中，然后再进行转换。返回值是 json 的字符串形式。

（2）char *pb2json(Message *msg)

函数参数：消息对象 msg

函数功能：适用于实参是解析好的 pb 数据，直接对 msg 的内容进行转换；其内部功能由以下函数实现：

1）json_t *parseFromMsg(const Message *msg)

函数参数：消息对象 msg

函数功能：对 msg 的每个 field 进行判断和处理，生成对应的 json 对象作为返回值。

2）json_t *parseFromRepeatField(const Message *msg, const FieldDescriptor *field)

函数参数：消息对象 msg，field 的描述类对象

函数功能：对判断为重复属性的 field 进行解析，利用 for 循环对所有项进行处理，返回一个 json 对象。

3）string hexEncode(string binInput)

函数参数：二进制的字符串 binInput

函数功能：将每个二进制字符转为十六进制字符，然后拼接成字符串，返回十六进制字符串。