**Protocol Buffers**

Github地址：<https://github.com/google/protobuf>

使用Protocol Buffers, 你要写一个XXX.proto的文件，这个文件时描述你要存储的数据结构的。接着使用Protocol complier 生成一个类，这个生成类用高效的二进制格式实现了自动编码和解析protocol buffer的数据。这个生成的类对其成员变量提供getter 和 setter方法。这些成员变量组成了protocol buffer 并且实现了将protocol buffer的整体读写。(大概意思是说 他封装好了序列化和反序列化的东西 你直接get 和 set就行了)。

Protocol complier卸载过程：

Protocol complier安装过程：

1. 安装brew

最新安装命令请查看： <http://brew.sh/index_zh-cn.html>

安装与使用请查看：<http://linfan.info/blog/2012/02/25/homebrew-installation-and-usage>

MacOSX是基于Unix的操作系统，可以安装大部分为Unix/Linux开发的软件。然而，如果只是以使用为目的，对每个软件都进行手工编译不是很方便，也不利于管理已安装的软件，于是出现了类似于Linux中APT、Yum等类似的软件包管理系统，其中最著名的有MacPorts、Fink、Homebrew等。

基本用法:

* brew search formula # 搜索软件包
* brew install formula # 安装软件包
* brew remove formula # 移除软件包
* brew cleanup formula # 清除旧包
* brew list # 列出已安装的软件包
* brew update # 更新 Homebrew
* brew upgrade # 升级软件包
* brew home formula # 用浏览器打开
* brew info formula # 显示软件内容信息
* brew deps formula # 显示包的依赖
* brew server # 启动 web 服务器，可以通过浏览器访问 http://localhost:4567 来通过网页来管理包
* brew -h # 帮助

brew versions formula # 列出软件包的版本

1. 安装automake、autoconf、libtool

brew install automake

brew install libtool

1. 安装protoc

$ ./autogen.sh

$ autoconf

$ ./configure

  $ make

  $ make check

  $ sudo make install

1. 生成objc文件

1. protoc --proto\_path =.proto文件的上级目录 --objc\_out=文件的输出路径 .proto文件路径

2. protoc -I=.proto文件的上级目录 --objc\_out=文件的输出路径 .proto文件路径

例如：

protoc --proto\_path=/Users/ios\_feng/protobuf-master/objectivec/Tests --objc\_out=/Users/ios\_feng/Desktop/ /Users/ios\_feng/protobuf-master/objectivec/Tests/unittest\_cycle.proto

或

protoc -I=/Users/ios\_feng/protobuf-master/objectivec/Tests --objc\_out=/Users/ios\_feng/Desktop/ /Users/ios\_feng/protobuf-master/objectivec/Tests/unittest\_cycle.proto

语法解释：

在.proto文件中，最基本的数据类型为message，其定义如上所示，由message引导，之后是message类型的名字，之后是由{}包含的各个域(fields)。

required string name = 1; 域字义的一个例子。

required 表示这个域是必需的

optional 该域选，出现０次或１次

repeated 重复出现，０次或多次

string是域的类型，可是简单的标量类型（如bool,int32,float,double,string等），也可是复合类型(message,enum等)

name是域的名字，=1是给域一个数字标签，这会影响到该域在二进制文件中顺序。

关于这个数字标签也是有说明的，1到15是只使用一个字节编号，而其他的使用多个字节，所以应把1-15编号给最经常使用的域。数字标签的最大值为2\*\*29 - 1（或536,870,911），其中还有一段是保留用于proto的实现，从19000到19999 （FieldDescriptor::kFirstReservedNumber 到FieldDescriptor::kLastReservedNumber）。

有optional说明的域可以有一个默认值，在不指定该域时使用，如optional PhoneType type = 2 [default = HOME];

* 指定field规则

message SearchRequest{

        required string query = 1;

        optional int32 page\_number = 2;

        optional int32 result\_per\_page =3;

        repeated int32 samples = 4 **[**packed=true**]**;

}

由于历史原因，repeated字段如果是基本数字类型的话，不能有效地编码。现在代码可以使用特殊选项[packed=true]来得到更有效率的编码。

注： 由于required是永远的，应该非常慎重地给message某个字段设置为required。如果未来你希望停止写入或者输出某个required字段，那就会成为问题；因为旧的reader将以为没有这个字段无法初始化message，会丢掉这部分信息。一些来自google的工程师们指出使用required弊大于利，尽量使用optional和repeated。