Protocol buffers 序列化

一、思考的来源

在公司内部使用 grpc 做为RPC框架的同时，序列化的过程是RPC调用不可或缺的部分。事实上对我而言，用的最多的就是使用JSON作为序列化方案。为什么是这样呢？调研显示：JSON一般使用在了Restful接口调用的过程中，很明显的，JSON缺少类型信息。在RPC调用中难以看见用JSON作为跨语言的序列化方案。

当然，不可避免的，另一方面的揣测来源于grpc 与 protobuf 来源于 Google。仅此而已？下面我将对 protobuf 进一步的了解与学习。

二、解决方案

1. 官方概述

没有什么解释，是比官网更具有权威性的

> **What are protocol buffers?**
>
> Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler. You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages.

翻译大意：protobuf 是谷歌的一种语言无关，平台无关，可扩展的结构化数据序列化机制，类似于XML，但是它更小(体积)，更快(序列化效率)，更简单(结构简单)。你只需定义一次你想要的结构化数据，然后利用ProtoBuf的代码生成工具生成相关的代码，就可以轻松的用不同的语言去读写来自各个数据流的结构化数据。

很显然，官网介绍的优势：

- 平台无关
- 语言无关
- 可扩展
- 效率更高
- 结构简单
- 体积更小

2. 由来

Protocol Buffer 翻译过来 “协议缓冲区” ，为什么起这个名字呢？

Protobuf 最开始被用于处理新旧版本的兼容性问题。在早期的Google，有另外一套方式去处理 请求/响应 的编解码问题时，需要手动处理以支持多版本协议，代码不够优雅！

为了解决新旧版本与协议的兼容性问题，Protobuf由此诞生，最初被赋予了两个特点：

- **更容易引入新的字段**，可以简单地解析并传递数据，无需了解所有字段。
- **数据格式更加具有自我描述性**，可以用各种语言来处理。

但是，这个版本的Protobuf 仍需手写解析的代码。

随着系统的发展，演进，ProtoBuf具有了更多的特性：

- 自动生成的序列化和反序列化代码避免了手动解析的需要（官方提供自动生成代码工具，各个语言平台的基本都有）。
- 除了用于数据交换之外，ProtoBuf被用作持久化数据的便捷自描述格式。

显然，后期 Protobuf 开始用于数据交换与存储。

> *Why the name "Protocol Buffers"?*
>
> *The name originates from the early days of the format, before we had the protocol buffer compiler to generate classes for us. At the time, there was a class called ProtocolBuffer which actually acted as a buffer for an individual method. Users would add tag/value pairs to this buffer individually by calling methods like AddValue(tag, value). The raw bytes were stored in a buffer which could then be written out once the message had been constructed.*
>
> *Since that time, the "buffers" part of the name has lost its meaning, but it is still the name we use. Today, people usually use the term "protocol message" to refer to a message in an abstract sense, "protocol buffer" to refer to a serialized copy of a message, and "protocol message object" to refer to an in-memory object representing the parsed message.*

三、快速入门

1. 约定

a. 版本：proto3
b. 官网地址：https://developers.google.com/protocol-buffers/docs/proto3
c. 说明：很多内容都来自于网上的学习，特别是官网，学习也只是一个思考，整理与鉴别的过程。

2. 简单认识

2.1 定义消息类型

a. 案例：定义一个请求消息格式，用 `.proto` 文件进行描述

ProtoBuf

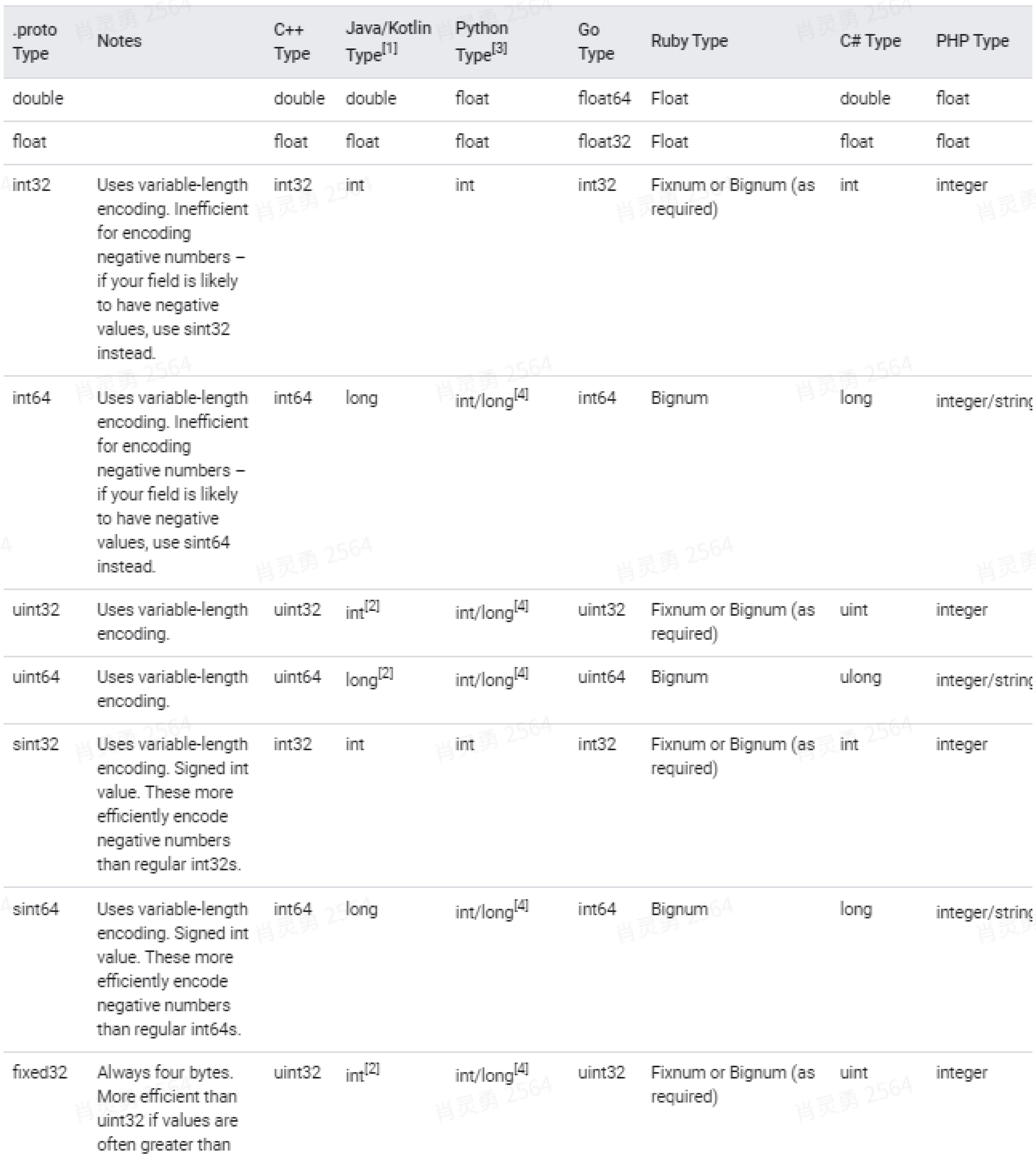
```protobuf
/*
 * 定义一个SearchRequest消息，
 * 一个 .proto 文件可以存放多个 message
 */
syntax = "proto3";

message SearchRequest {
  string query = 1;  // string 类型
  int32 page_number = 2; // 32bit int类型
  int32 result_per_page = 3; // 32bit int类型
}

message SearchResponse {
...
}
```

b. 语法说明：
   i. `syntax:` 指定正在使用的是 `proto3` 语法。
   ii. `message`：定义消息格式
   iii. `SearchRequest` 消息内部的指定了三个字段，表示这条消息要包含的数据，
   iv. 字段声明 `string query = 1`：`类型  字段名称 = 字段编号`
      1. 字段编号：消息定义中的每个字段都有一个**唯一的编号**。这些字段编号用于在 消息二进制格式中标识字段，如果我们的消息类型被使用就不能更改它。
      2. 范围：1 - 536,870,911(2的29次方 - 1)
         a. 1 - 15 的字段，占用一个字节，其中包括字段编号和字段类型
         b. 16 - 2047 的字段，占用两个字节。所以我们要为现在或者将来经常出现的消息元素保留 1 到 15 的数字
         c. 19000 - 19999：不能被使用，protobuf 保留的编号
      3. 消息字段规则：
         - `singular`：0/1个(不超过一个)。这是 proto3 语法的默认字段规则。
         - `repeated`：0..n 字段可以在消息中重复任意次数(包括零次)。重复值的顺序将被保留。`repeated` 字段默认使用的编码 `packed`。
   v. 注释
      1. 单行：`//`
      2. 多行：`/* ... */`
   vi. 保留字段

1. 作用：移除某一个字段类型，或者是为将来频繁使用的字段预留编号，不能使用注解或者直接删除的方法，因为用户可能会重新的使用这个字段，那么加载老版本时会出现数据损坏等问题。
2. 语法：
   - 保留字段：`reserved 1, 2, 5 to 11`
   - 保留编号：`reserved "id", "name"`

2.2 标量类型

a. 说明：字段类型对应语言的类型
b. 类型：



.proto Type	Notes	C++ Type	Java/Kotlin Type[1]	Python Type[3]	Go Type	Ruby Type	C# Type	PHP Type
double		double	double	float	float64	Float	double	float
float		float	float	float	float32	Float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int	int32	Fixnum or Bignum (as required)	int	integer
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long[4]	int64	Bignum	long	integer/string
uint32	Uses variable-length encoding.	uint32	int[2]	int/long[4]	uint32	Fixnum or Bignum (as required)	uint	integer
uint64	Uses variable-length encoding.	uint64	long[2]	int/long[4]	uint64	Bignum	ulong	integer/string
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int	int32	Fixnum or Bignum (as required)	int	integer
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long[4]	int64	Bignum	long	integer/string
fixed32	Always four bytes. More efficient than uint32 if values are often greater than	uint32	int[2]	int/long[4]	uint32	Fixnum or Bignum (as required)	uint	integer

	2^{28}.							
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^{56}.	uint64	long[2]	int/long[4]	uint64	Bignum	ulong	integer/strin
sfixed32	Always four bytes.	int32	int	int	int32	Fixnum or Bignum (as required)	int	integer
sfixed64	Always eight bytes.	int64	long	int/long[4]	int64	Bignum	long	integer/strin
bool		bool	boolean	bool	bool	TrueClass/FalseClass	bool	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text, and cannot be longer than 2^{32}.	string	String	str/unicode[5]	string	String (UTF-8)	string	string
bytes	May contain any arbitrary sequence of bytes no longer than 2^{32}.	string	ByteString	str (Python 2) bytes (Python 3)	[]byte	String (ASCII-8BIT)	ByteString	string

2.3 枚举Enum

a. 案例：期望字段的值在预定义的列表里面

b. 代码实现：

```ProtoBuf
message SearchRequest {
  ...
  enum Corpus {
      UNIVERSAL = 0;
      WEB = 1;
      IMAGES = 2;
      LOCAL = 3;
      NEWS = 4;
      PRODUCTS = 5;
      VIDEO = 6;
  }
  Corpus corpus = 4;
}
```

c. 注意:

- 每个 enum 的字段常量都必须有一个 0，默认情况下，我们使用 0 作为默认值
- 0 必须是第一个元素，为了兼容 proto2 语义：规定第一个元素是默认值

d. 设置别名:

i. 在同一个 enum 中，将相同的值设置给不同的枚举常量

ProtoBuf

```protobuf
message MyMessage1 {
  enum EnumAllowingAlias {
      option allow_alias = true;
      UNKNOWN = 0;
      STARTED = 1;
      RUNNING = 1;
  }
}
message MyMessage2 {
  enum EnumNotAllowingAlias {
      UNKNOWN = 0;
      STARTED = 1;
      // RUNNING = 1;  // Uncommenting this line will cause a compile error inside Google and a warning message outside.
  }
}
```

ii. 枚举常量必须在 32 位整数范围内，由于使用的是varint编码，所以负数的效率低，不推荐使用。

iii. Enum 可以定义在 .proto 文件中，被其他的message使用，也可以利用 _MessageType_._EnumType_ 将一条消息中声明的字段类型在其他消息的字段中使用。

e. 保留值

i. 类似于前面定义消息时的保留字段，只不过这个作用于 enum

ii. 代码实现：

Apache

```
enum Foo {
    reserved 2, 15, 9 to 11, 40 to max;
    reserved "FOO", "BAR";
}
```

iii. 注意：不能在同一个 reserved 中同时使用字段名称和数值

2.4 自定义消息类型

a. 案例：想要在每一个 SearchResponse 中包含一个 Result message

b. 代码实现：

ProtoBuf

```
message SearchResponse {
  // 自定义类型 Result
  repeated Result results = 1;
}

message Result {
  string url = 1;
  string title = 2;
  repeated string snippets = 3;
}
```

c. 导入类型
    i. 导入其他 .proto 文件的类型，需要在文件的最上面，添加 import 语句

ProtoBuf

```
import "myproject/other_protos.proto";
```

    ii. 注意：
        1. proto2 和 proto3 的message类型可以相互引用
        2. proto2 的 enum 类型不能被proto3 直接使用，需要通过 proto2 message 引用 enum，再让 proto3 message引用 proto2 message。

2.5 嵌套消息

a. 说明：允许在一个消息类型里面嵌套消息类型，且支持多层嵌套
b. 代码实现：

ProtoBuf

```
message SearchResponse {
  message Result {
      string url = 1;
      string title = 2;
      repeated string snippets = 3;
  }
  repeated Result results = 1;
}
```

在其他 message 中引用

ProtoBuf

```protobuf
message SomeOtherMessage {
    SearchResponse.Result result = 1;
}
```

2.6 未知字段

a. 说明：如果旧二进制文件解析新二进制文件的新字段，新字段将无法被识别，也就是未知的字段。在proto 3.5之后，未知的字段将会保留，在此之前的版本都会丢掉该字段。

2.7 Any

a. 说明：Any 是一个消息类型，可以代表任何一个以字节形式存在的序列化消息类型
b. 使用：

ProtoBuf

```protobuf
import "google/protobuf/any.proto";

message ErrorStatus {
    string message = 1;
    repeated google.protobuf.Any details = 2;
}
```

2.8 Oneof

a. 案例：如果一个消息里面有很多的字段，但是想要最多只有一个值被设置
b. 使用：

ProtoBuf

```protobuf
message SampleMessage {
  oneof test_oneof {
    string name = 4;
        SubMessage sub_message = 9;
  }
}
```

c. 特征：
   - 设置oneof字段某个值的同时，会取消oneof前面设置过的所有值
   - oneof 字段不能被 repeat d 修饰
   - 如果oneof字段设置了默认值，那么这个oneof字段会被序列化成对应值

2.9 Map

a. 案例：创建一个具有映射关系的字段
b. 使用：

ProtoBuf

```
map<key_type, value_type> map_field = N;
```

c. 注意：
    i. key可以是int，string类型，不能是浮点和字节，枚举类型
    ii. value可以是除了map之外的所有类型
    iii. repeated 不可以作用在map字段上
    iv. map映射没有特定的顺序
    v. map的key是int类型的话，默认以此排序
    vi. map的key不可以重复

2.10 Packages

a. 案例：为了防止消息类型的冲突，需要对proto文件进行特殊说明
b. 使用：

ProtoBuf

```
package foo.bar;

message Open { ... }
```

引用上面的消息类型

ProtoBuf

```
message Foo {
   ...
   foo.bar.Open open = 1;
}
```

想要影响生成的代码，这个依赖于所选择的语言，例如：在Java中，需要在proto文件中，显示的提供一个java_package，表示是一个Java package

2.11 定义Service

a. 案例：在RPC过程中，使用我们的消息类型

b. 代码实现：

ProtoBuf

```
service SearchService {
  rpc Search(SearchRequest) returns (SearchResponse);
}
```

a. 效果：协议缓冲区编译器会以选择的语言，生成服务接口代码和存根

2.12 JSON 映射

a. 说明：将JSON转为protobuf，当json的数据为null，解析到protobuf时，会有对应的默认值；如果在 proto 文件有设置默认值，将优先选择对应值。
b. 规范：

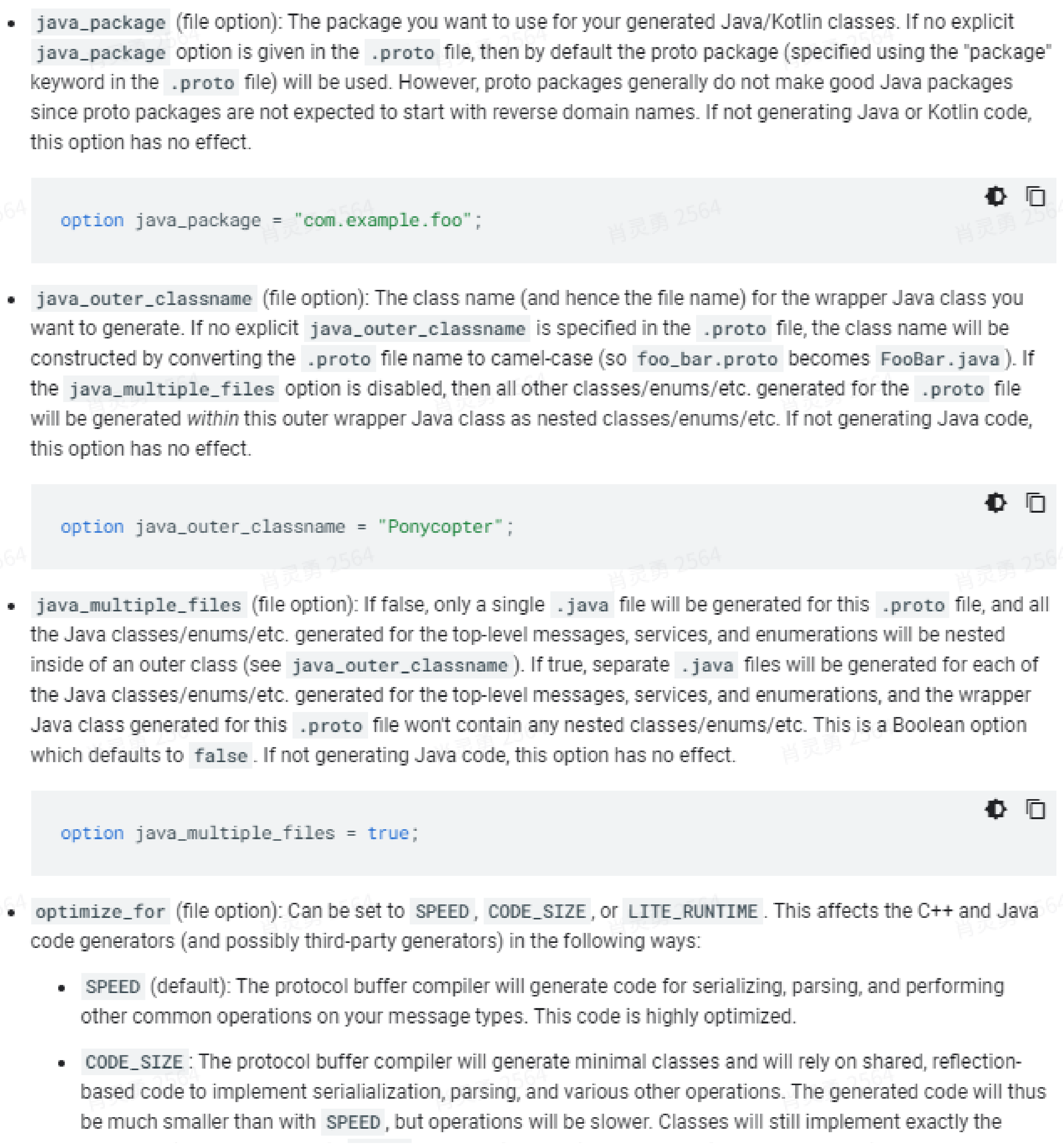
proto3	JSON	JSON example	Notes
message	object	`{"fooBar": v, "g": null, …}`	Generates JSON objects. Message field names are mapped to lowerCamelCase and become JSON object keys. If the `json_name` field option is specified, the specified value will be used as the key instead. Parsers accept both the lowerCamelCase name (or the one specified by the `json_name` option) and the original proto field name. `null` is an accepted value for all field types and treated as the default value of the corresponding field type.
enum	string	`"FOO_BAR"`	The name of the enum value as specified in proto is used. Parsers accept both enum names and integer values.
map<K,V>	object	`{"k": v, …}`	All keys are converted to strings.
repeated V	array	`[v, …]`	`null` is accepted as the empty list [ ].
bool	true, false	`true, false`	
string	string	`"Hello World!"`	
bytes	base64 string	`"YWJjMTIzIT8kKiYoKSctPUB+"`	JSON value will be the data encoded as a string using standard base64 encoding with paddings. Either standard or URL-safe base64 encoding with/without paddings are accepted.
int32, fixed32, uint32	number	`1, -10, 0`	JSON value will be a decimal number. Either numbers or strings are accepted.
int64, fixed64, uint64	string	`"1", "-10"`	JSON value will be a decimal string. Either numbers or strings are accepted.
float, double	number	`1.1, -10.0, 0, "NaN", "Infinity"`	JSON value will be a number or one of the special string values "NaN", "Infinity", and "-Infinity". Either numbers or strings are accepted. Exponent notation is also accepted. -0 is considered equivalent to 0.
Any	`object`	`{"@type": "url", "f": v, … }`	If the Any contains a value that has a special JSON mapping, it will be converted as follows: `{"@type": xxx, "value": yyy}`. Otherwise, the value will be converted into a JSON object, and the `"@type"` field will be inserted to indicate the actual data type.
Timestamp	string	`"1972-01-01T10:00:20.021Z"`	Uses RFC 3339, where generated output will always be Z-normalized and uses 0, 3, 6 or 9 fractional digits. Offsets other than "Z" are also accepted.
Duration	string	`"1.000340012s", "1s"`	Generated output always contains 0, 3, 6, or 9 fractional digits, depending on required precision, followed by the suffix "s". Accepted are any fractional digits (also none) as long as they fit into nano-seconds precision and the suffix "s" is required.
Struct	`object`	`{ … }`	Any JSON object. See `struct.proto`.
Wrapper types	various types	`2, "2", "foo", true, "true", null, 0, …`	Wrappers use the same representation in JSON as the wrapped primitive type, except that `null` is allowed and preserved during data conversion and transfer.
FieldMask	string	`"f.fooBar,h"`	See `field_mask.proto`.
ListValue	array	`[foo, bar, …]`	
Value	value		Any JSON value. Check google.protobuf.Value for details.
NullValue	null		JSON null
Empty	object	`{}`	An empty JSON object

2.13 Options

a. 说明：在proto文件中添加不同的option，会随之改变一些处理的行为

b. 参数：

- java_package：该字段用于标识生成的java文件的package。如果没有指定，则使用proto里定义的package，如果package也没有指定，那就会生成在根目录下；
- java_outer_classname：用于指定proto文件生成的java类的outerclass类名。什么是outerclass？简单来说就是用一个class文件来定义所有的message对应的java类。这个class就是outerclass；如果没有指定，默认是proto文件的驼峰式；
- java_multiple_files：如果是true，那么每一个message文件都会有一个单独的class文件；否则，message全部定义在outerclass文件里。



- `java_package` (file option): The package you want to use for your generated Java/Kotlin classes. If no explicit `java_package` option is given in the `.proto` file, then by default the proto package (specified using the "package" keyword in the `.proto` file) will be used. However, proto packages generally do not make good Java packages since proto packages are not expected to start with reverse domain names. If not generating Java or Kotlin code, this option has no effect.

```
option java_package = "com.example.foo";
```

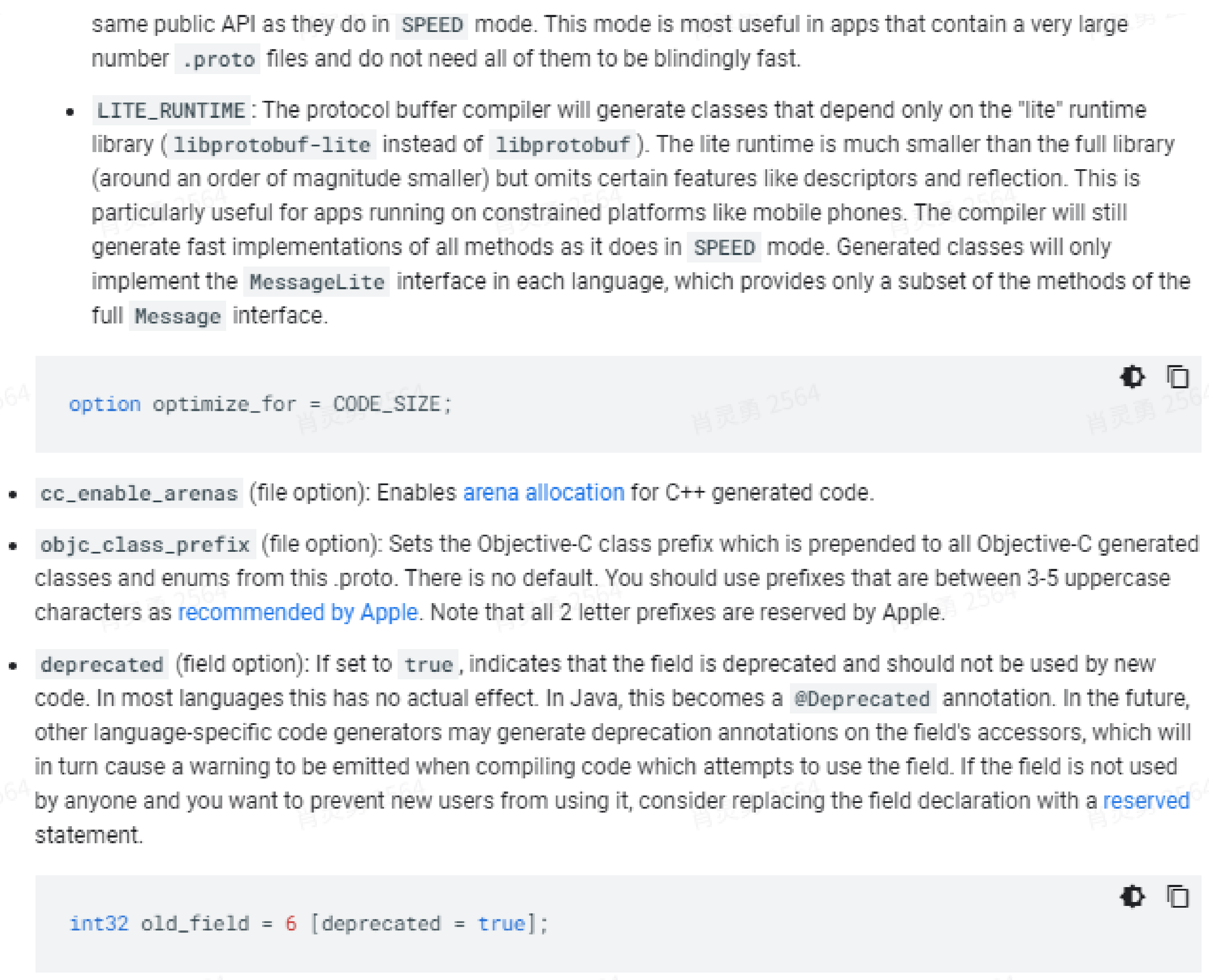
- `java_outer_classname` (file option): The class name (and hence the file name) for the wrapper Java class you want to generate. If no explicit `java_outer_classname` is specified in the `.proto` file, the class name will be constructed by converting the `.proto` file name to camel-case (so `foo_bar.proto` becomes `FooBar.java`). If the `java_multiple_files` option is disabled, then all other classes/enums/etc. generated for the `.proto` file will be generated *within* this outer wrapper Java class as nested classes/enums/etc. If not generating Java code, this option has no effect.

```
option java_outer_classname = "Ponycopter";
```

- `java_multiple_files` (file option): If false, only a single `.java` file will be generated for this `.proto` file, and all the Java classes/enums/etc. generated for the top-level messages, services, and enumerations will be nested inside of an outer class (see `java_outer_classname`). If true, separate `.java` files will be generated for each of the Java classes/enums/etc. generated for the top-level messages, services, and enumerations, and the wrapper Java class generated for this `.proto` file won't contain any nested classes/enums/etc. This is a Boolean option which defaults to `false`. If not generating Java code, this option has no effect.

```
option java_multiple_files = true;
```

- `optimize_for` (file option): Can be set to `SPEED`, `CODE_SIZE`, or `LITE_RUNTIME`. This affects the C++ and Java code generators (and possibly third-party generators) in the following ways:
  - `SPEED` (default): The protocol buffer compiler will generate code for serializing, parsing, and performing other common operations on your message types. This code is highly optimized.
  - `CODE_SIZE`: The protocol buffer compiler will generate minimal classes and will rely on shared, reflection-based code to implement serialialization, parsing, and various other operations. The generated code will thus be much smaller than with `SPEED`, but operations will be slower. Classes will still implement exactly the



same public API as they do in `SPEED` mode. This mode is most useful in apps that contain a very large number `.proto` files and do not need all of them to be blindingly fast.

- `LITE_RUNTIME`: The protocol buffer compiler will generate classes that depend only on the "lite" runtime library (`libprotobuf-lite` instead of `libprotobuf`). The lite runtime is much smaller than the full library (around an order of magnitude smaller) but omits certain features like descriptors and reflection. This is particularly useful for apps running on constrained platforms like mobile phones. The compiler will still generate fast implementations of all methods as it does in `SPEED` mode. Generated classes will only implement the `MessageLite` interface in each language, which provides only a subset of the methods of the full `Message` interface.

```
option optimize_for = CODE_SIZE;
```

- `cc_enable_arenas` (file option): Enables arena allocation for C++ generated code.
- `objc_class_prefix` (file option): Sets the Objective-C class prefix which is prepended to all Objective-C generated classes and enums from this .proto. There is no default. You should use prefixes that are between 3-5 uppercase characters as recommended by Apple. Note that all 2 letter prefixes are reserved by Apple.
- `deprecated` (field option): If set to `true`, indicates that the field is deprecated and should not be used by new code. In most languages this has no actual effect. In Java, this becomes a `@Deprecated` annotation. In the future, other language-specific code generators may generate deprecation annotations on the field's accessors, which will in turn cause a warning to be emitted when compiling code which attempts to use the field. If the field is not used by anyone and you want to prevent new users from using it, consider replacing the field declaration with a reserved statement.

```
int32 old_field = 6 [deprecated = true];
```

3. Java 使用

a. 参考网址：https://developers.google.com/protocol-buffers/docs/javatutorial

b. 步骤：

   i. 定义一个 .proto 文件

   ii. 使用Protobuf 编译器

   iii. 使用 Java的 protobuf API去I去读写数据

四、原理

暂未更新......