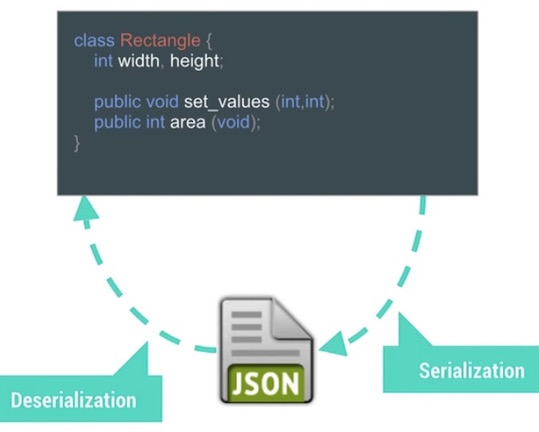
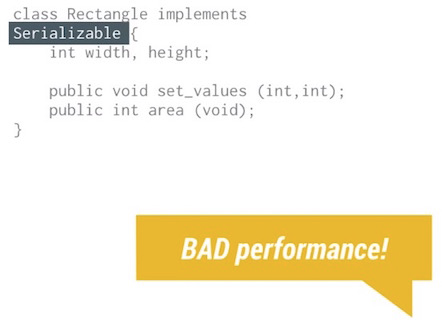
FlatBuffers(数据序列化的性能优化)

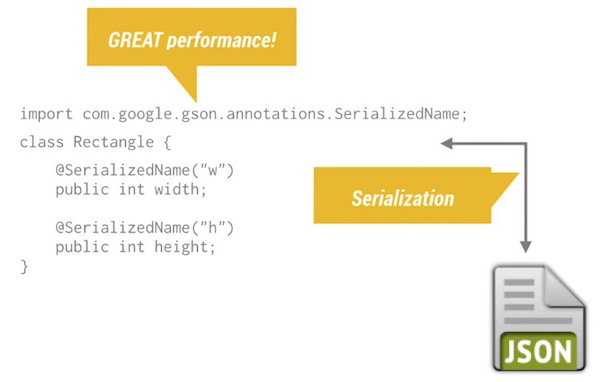
数据的序列化是程序代码里面必不可少的组成部分，当我们讨论到数据序列化的性能的时候，需要了解有哪些候选的方案，他们各自的优缺点是什么。首先什么是序列化？用下面的图来解释一下：



数据序列化的行为可能发生在数据传递过程中的任何阶段，例如网络传输，不同进程间数据传递，不同类之间的参数传递，把数据存储到磁盘上等等。通常情况下，我们会把那些需要序列化的类实现Serializable接口(如下图所示)，但是这种传统的做法效率不高，实施的过程会消耗更多的内存。



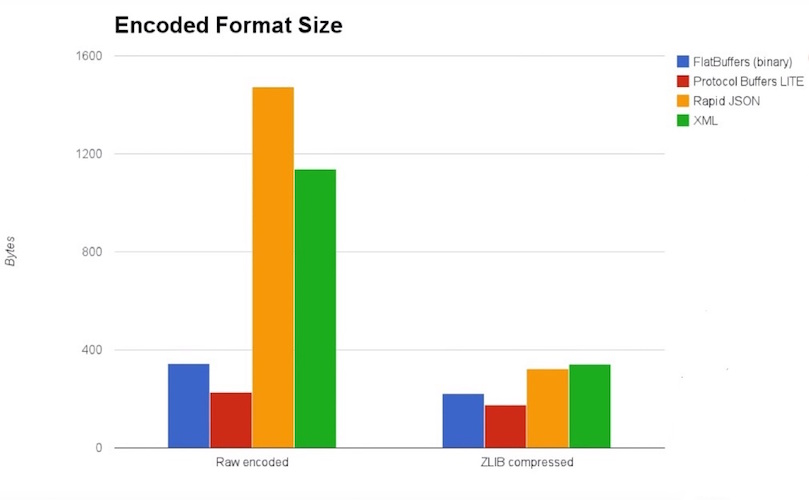
但是我们如果使用GSON库来处理这个序列化的问题，不仅仅执行速度更快，内存的使用效率也更高。Android的XML布局文件会在编译的阶段被转换成更加复杂的格式，具备更加高效的执行性能与更高的内存使用效率。

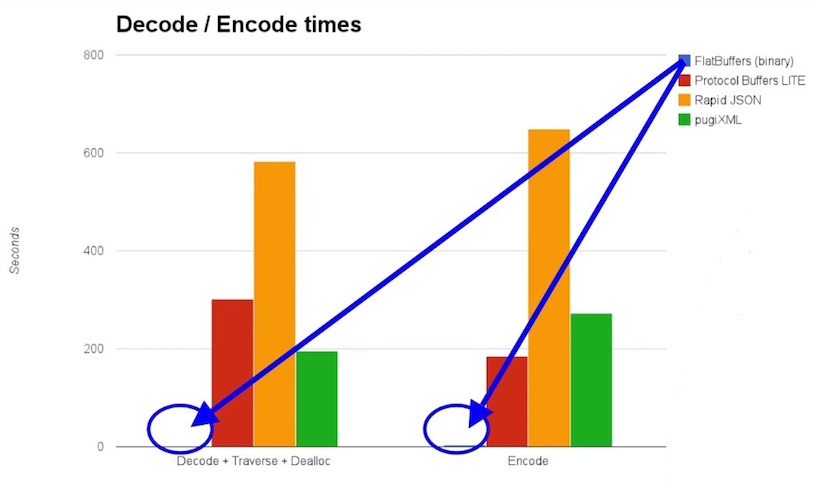


下面介绍三个数据序列化的候选方案：

* [**Protocal Buffers**](https://developers.google.com/protocol-buffers/?utm_campaign=android_series_serialization_performance_101315&utm_source=anddev&utm_medium=yt-annt)：强大，灵活，但是对内存的消耗会比较大，并不是移动终端上的最佳选择。
* [**Nano-Proto-Buffers**](https://android.googlesource.com/platform/external/protobuf/+/master/java/README.txt?utm_campaign=android_series_serialization_performance_101315&utm_source=anddev&utm_medium=yt-annt)：基于Protocal，为移动终端做了特殊的优化，代码执行效率更高，内存使用效率更佳。
* [**FlatBuffers**](https://google.github.io/flatbuffers/)：这个开源库最开始是由Google研发的，专注于提供更优秀的性能。

上面这些方案在性能方面的数据对比如下图所示：





可见，**FlatBuffers** 几乎从空间和时间复杂度上完胜其他技术。

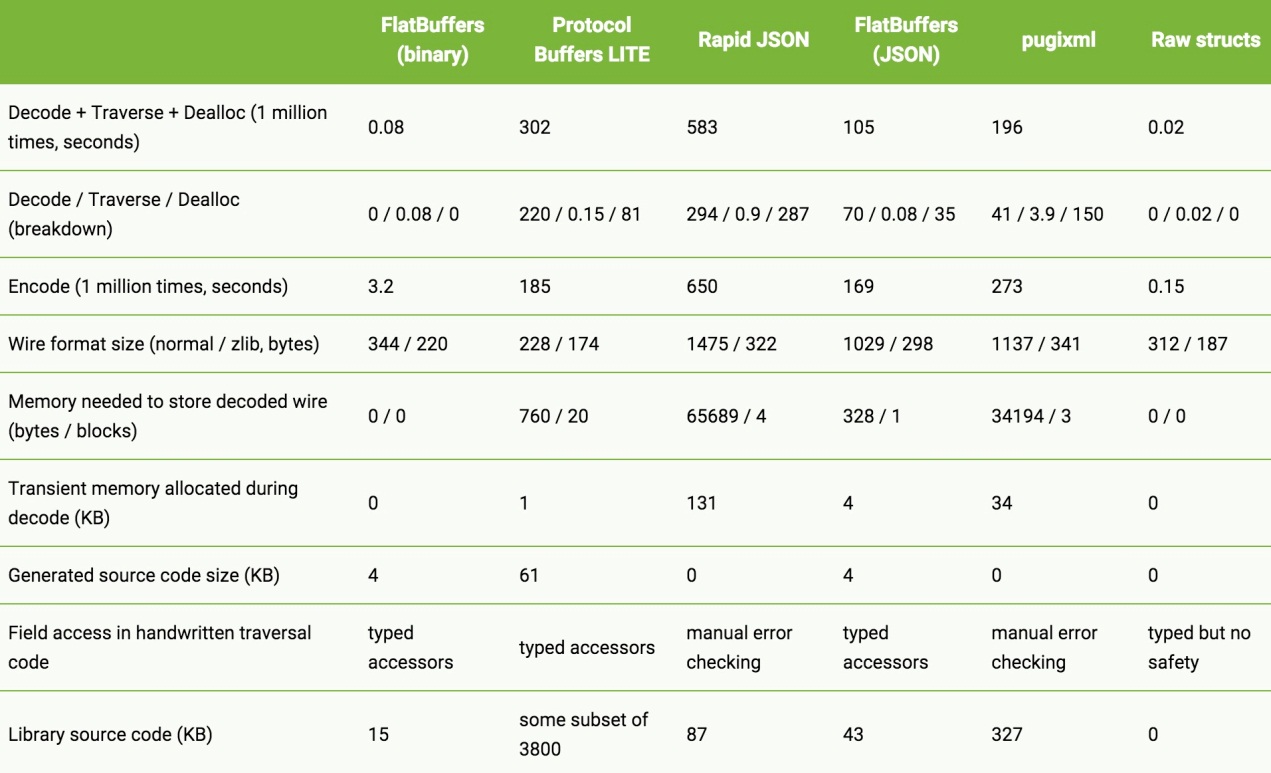
**FlatBuffers** 是一个开源的跨平台数据序列化库，可以应用到几乎任何语言（C++, C#, Go, Java, JavaScript, PHP, Python），最开始是 Google 为游戏或者其他对性能要求很高的应用开发的。项目地址在 [GitHub](https://github.com/google/flatbuffers) 上。官方的文档在 [这里](http://google.github.io/flatbuffers/index.html)。

**FlatBuffer 的优点**

FlatBuffer 相对于其他序列化技术，例如 XML，JSON，Protocol Buffers 等，有哪些优势呢？官方文档的说法如下：

1. **直接读取序列化数据**，而不需要解析（Parsing）或者解包（Unpacking）：FlatBuffer 把数据层级结构保存在一个扁平化的二进制缓存（一维数组）中，同时能够保持直接获取里面的结构化数据，而不需要解析，并且还能保证数据结构变化的前后向兼容。
2. 高效的内存使用和速度：FlatBuffer 使用过程中，不需要额外的内存，几乎接近原始数据在内存中的大小。
3. 灵活：数据能够前后向兼容，并且能够灵活控制你的数据结构。
4. 很少的代码侵入性：使用少量的自动生成的代码即可实现。
5. 强数据类性，易于使用，跨平台，几乎语言无关。

官方提供了一个性能对比表如下：



在做 Android 开发的时候，JSON 是最常用的数据序列化技术。我们知道，JSON 的可读性很强，但是序列化和反序列化性能却是最差的。解析的时候，JSON 解析器首先，需要在内存中初始化一个对应的数据结构，这个事件经常会消耗 100ms ~ 200ms[2](http://www.race604.com/flatbuffers-intro/#fn:2)；解析过程中，要产生大量的临时变量，造成 Java 虚拟机的 GC 和内存抖动，解析 20KB 的数据，大概会消耗 100KB 的临时内存[2](http://www.race604.com/flatbuffers-intro/#fn:2)。FlatBuffers 就解决了这些问题。

## 使用方法

简单来说，FlatBuffers 的使用方法是，首先按照使用特定的 IDL 定义数据结构 schema，然后使用编译工具 flatc 编译 schema 生成对应的代码，把生成的代码应用到工程中即可。下面详细介绍每一步。

首先，我们需要得到 flatc，这个需要从源码编辑得到。从 GitHub 上 Clone 代码，

$ git clone https://github.com/google/flatbuffers

首先要使用 FlatBuffers 的 IDL 定义好数据结构 Schema，编写 Schema 的详细文档在[这里](http://google.github.io/flatbuffers/flatbuffers_guide_writing_schema.html)。其语法和 C 语言类似，比较容易上手。我们这里引用一个简单的例子[2](http://www.race604.com/flatbuffers-intro/#fn:2)，假设数据结构如下：

class Person {

String name;

int friendshipStatus;

Person spouse;

List<Person>friends;

}

编写成 Schema 如下，文件名为 Person.fbs：

// Person schema

namespace com.race604.fbs;

enum FriendshipStatus: int {Friend = 1, NotFriend}

table Person {

name: string;

friendshipStatus: FriendshipStatus = Friend;

spouse: Person;

friends: [Person];

}

root\_type Person;

然后，使用 flatc 可以把 Schema 编译成多种编程语言，我们仅仅讨论 Android 平台，所以把 Schema 编译成 Java，找到flatc.exe执行命令如下：

$ ./flatc –j -b Person.fbs

在当前目录生成如下文件：

.

└── com

   └── race604

   └── fbs

   ├── FriendshipStatus.java

   └── Person.java

Person 类有响应的函数直接获取其内部的属性值，使用非常简单：

Person person = ...;

// 获取普通成员

String name = person.name();

int friendshipStatus = person.friendshipStatus();

// 获取数组

int length = person.friendsLength()

for (int i = 0; i < length; i++) {

Person friends = person.friends(i);

...

}

下面我们来构建一个 Person 对象，名字是 "John"，其配偶（spouse）是 "Mary"，还有两个朋友，分别是 "Dave" 和 "Tom"，实现如下：

private ByteBuffer createPerson() {

FlatBufferBuilder builder = new FlatBufferBuilder(0);

int spouseName = builder.createString("Mary");

int spouse = Person.createPerson(builder, spouseName, FriendshipStatus.Friend, 0, 0);

int friendDave = Person.createPerson(builder, builder.createString("Dave"),

FriendshipStatus.Friend, 0, 0);

int friendTom = Person.createPerson(builder, builder.createString("Tom"),

FriendshipStatus.Friend, 0, 0);

int name = builder.createString("John");

int[] friendsArr = new int[]{ friendDave, friendTom };

int friends = Person.createFriendsVector(builder, friendsArr);

Person.startPerson(builder);

Person.addName(builder, name);

Person.addSpouse(builder, spouse);

Person.addFriends(builder, friends);

Person.addFriendshipStatus(builder, FriendshipStatus.NotFriend);

int john = Person.endPerson(builder);

builder.finish(john);

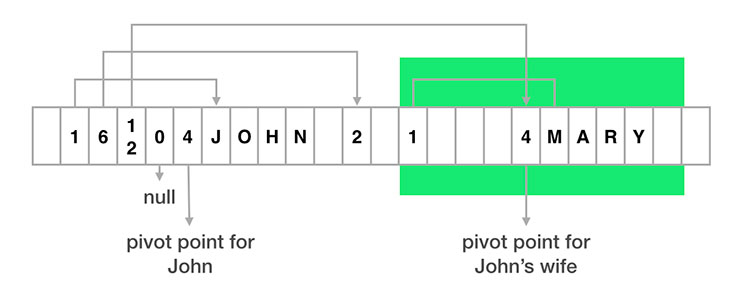
return builder.dataBuffer();

}

基本方法就是通过 FlatBufferBuilder 工具，往里面填写数据，详细的写法可以参考官方文档[3](http://www.race604.com/flatbuffers-intro/#fn:3)。可见，其实写法略显繁琐，不太直观。

## 基本原理

如官方文档的介绍，**FlatBuffers** 就像它的名字所表示的一样，就是把结构化的对象，用一个扁平化（Flat）的缓冲区保存，简单的来说就是把内存对象数据，保存在一个一维的数组中。借用 Facebook 文章[2](http://www.race604.com/flatbuffers-intro/#fn:2)的一张图如下：



可见，FlatBuffers 保存在一个 byte 数组中，有一个“支点”指针（pivot point）以此为界，存储的内容分为两个部分：元数据和数据内容。其中元数据部分就是数据在前面，其长度等于对象中的字段数量，每个 byte 保存对应字段内容在数组中的索引（从支点位置开始计算）。

如图，上面的 Person 对象第一个字段是 name，其值的索引位置是 1，所以从索引位置 1 开始的字符串，就是 name 字段的值 "John"。第二个字段是 friendshipStatus，其索引值是 6，找到值为 2， 表示 NotFriend。第三个字段是 spouse，也一个 Person 对象，索引值是 12，指向的是此对象的支点位置。第四个字段是一个数组，图中表示的数组为空，所以索引值是 0。

通过上面的解析，可以看出，FlatBuffers 通过自己分配和管理对象的存储，使对象在内存中就是**线性结构化**的，直接可以把内存内容保存或者发送出去，加载“解析”数据只需要把 byte 数组加载到内存中即可，不需要任何解析，也不产生任何中间变量。

它与具体的机器或者运行环境无关，例如在 Java 中，对象内的内存不依赖 Java 虚拟机的堆内存分配策略实现，所以也是跨平台的。

## 使用建议

通过前面的体验，FlatBuffers 几乎秒杀了 JSON

下面说说 FlatBuffers 的几点缺点：

1. FlatBuffers 需要生成代码，对代码有侵入性；
2. 数据序列化没有可读性，不方便 Debug；
3. 构建 FlatBuffers 对象比较麻烦，不直观，特别是如果对象比较复杂情况下需要写大段的代码；
4. 数据的所有内容需要使用 Schema 严格定义，灵活性不如 JSON。

所以，在什么情况下选择使用 FlatBuffers 呢？个人感觉需要满足以下几点：

1. 项目中有大量数据传输和解析，使用 JSON 成为了性能瓶颈；
2. 稳定的数据结构定义。