[原创]Protobuf协议解析文档-某音弹幕 优

天荒怨未泯 5

极客

14小时前

0x00：前言

近期因为工作原因着手分析某音协议,在做到直播间这步的时候通过抓包发现其直播间内弹幕数据为protobuf协议,之前用xposed做wx的时候虽然接触过,但在下从来都是只要结果的人,直接调用wx内部函数转成对象即可,谁理你什么格式,但现在做的既然是协议,那自然是无端可用,什么东西都要自己处理,这里分享一下记录的分析流程.

0x01：准备工作

在开始分析之前需要准备好:

1.protobuf(安装方法百度即可);

2.直播间弹幕抓包;

3.通过protoc工具测试:将数据包里返回的弹幕内容复制到文本,通过指令

protoc --decode_raw < xxx.bin 查看解析后的效果:

```
C:\Users\Kanon\Desktop
λ protoc --decode_raw < 000.bin
1 {
  1: "WebcastRoomMessage"
  2 {
    1 {
      1: "SystemMessage"
      2: 1599463242822895501
      3: 6869606498666810126
      4: 1599463242822
    }
    2: "\346\254\242\350\277\216\346\235\245\345\210\260\347\233\264\346\222\255\351\227\264\357\274\201\346\212\226\351\237\263\344\270\245\347\246\201\346\234\252\346\210\220\345\271\264\344\272\272\350\277\2
3\350\241\214\347\233\264\346\222\255\346\210\226\346\211\223\350\265\217\357\274\214\350\257\267\345\244\247\345\256\266\345\205\261\345\220\214\351\201\265\345\256\210\343\200\201\347\233\221\347\235\243\343\
00\202\347\233\264\346\222\255\351\227\264\345\206\205\344\270\245\347\246\201\345\207\272\347\216\260\350\277\235\346\263\225\350\277\235\350\247\204\343\200\201\344\275\216\344\277\227\350\211\262\346\203\205
343\200\201\345\220\270\347\203\237\351\205\227\351\205\222\347\255\211\345\206\205\345\256\271\357\274\214\350\213\245\346\234\211\350\277\235\350\247\204\350\241\214\344\270\272\350\257\267\345\217\212\346\22
\266\344\270\276\346\212\245\343\200\202\345\246\202\344\270\273\346\222\255\345\234\250\347\233\264\346\222\255\350\277\207\347\250\213\344\270\255\344\273\245\351\231\252\347\216\251\343\200\201\351\200\201\3
7\244\274\347\255\211\346\226\271\345\274\217\350\277\233\350\241\214\350\257\261\345\257\274\346\211\223\350\265\217\343\200\201\347\247\201\344\270\213\344\272\244\346\230\223\357\274\214\350\257\267\350\260\
50\346\205\216\345\210\244\346\226\255\357\274\214\344\273\245\351\230\262\344\272\272\350\272\253\346\210\226\350\264\242\344\272\247\346\215\237\345\244\261\343\200\202\350\257\267\345\244\247\345\256\266\346
263\250\346\204\217\350\264\242\344\272\247\345\256\211\345\205\250\357\274\214\350\260\250\351\230\262\347\275\221\347\273\234\350\257\210\351\252\227\343\200\202 "
  }
  3: 1599463242822917947
}
2: "1599463242875_6869642315544174994_1_1"
3: 1000
4: 1599463242875
5: "fetch_time:1599463242875|start_time:1599463238307|ack_ids:6869642295172813837_1814,6869642295936308228_1816,6869642300197178124_1816,6869639343049493255_1816,6869642316152853517_181e|flag:1|seq:1"

C:\Users\Kanon\Desktop
λ
```

可以被工具正常解析,到这里准备工作就完成了, 接下来本文档以图中数据为例,进入分析流程;

0x02：分析流程

上面通过protoc解析出了个大概,那么里面的数据究竟是什么意思呢? 1 2 3, 1: 2:这些都是什么呢? 网上所搜一番得知:解析出来的数据中开头的1 2 3..或 1: 2: 3:的均为protobuf的tag, 是解析数据的关键值,带":"号的是字段,没":"号的是 repeated 集合.

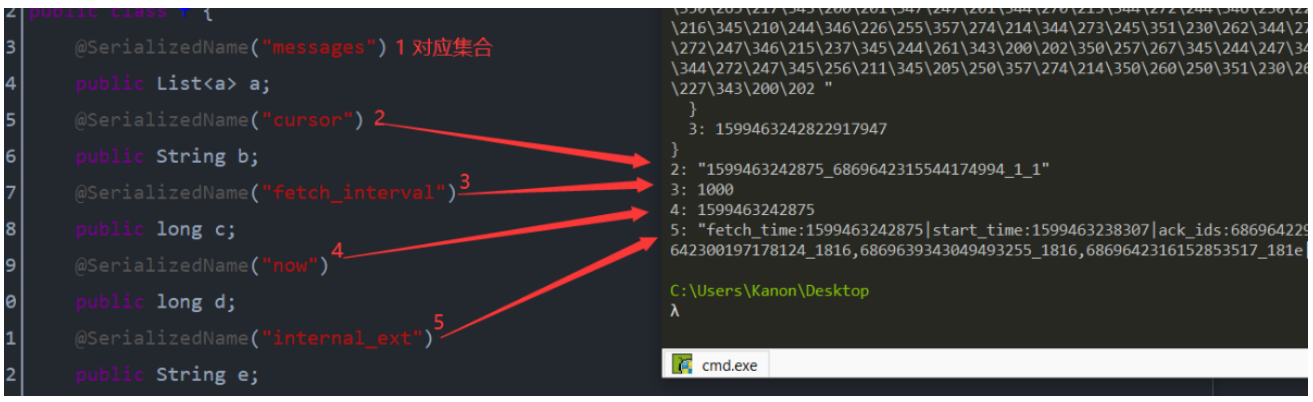
那么知道了这些,要怎么还原数据呢,逆向中是没有原始.proto文件的,对于protobuf的解析就需要依靠分析源码,自己生成一份.proto文件,以上图为例,在源码中定位到解析代码:

```
1  private ProtoApiResult a(b bVar) throws Exception {
2          PatchProxyResult proxy = PatchProxy.proxy(new Object[]{bVar}, this, a, false, 22172);
3          if (proxy.isSupported) {
4              return (ProtoApiResult) proxy.result;
5          }
6          com.bytedance.android.tools.pbadapter.a.b protoDecoder = ((INetworkService)
7  d.a(INetworkService.class)).getProtoDecoder(com.bytedance.android.livesdkapi.message.f.class);
8          if (protoDecoder != null) {
9              com.bytedance.android.livesdkapi.message.f fVar = (com.bytedance.android.livesdkapi.message.f) protoDecoder.decode(this.s.a(bVar));
10             ProtoApiResult protoApiResult = new ProtoApiResult();
11             protoApiResult.cursor = fVar.b;
12             protoApiResult.fetchInterval = fVar.c;
13             protoApiResult.now = fVar.d;
14             protoApiResult.messages = new LinkedList();
15             this.m = fVar.e;
16             long currentTimeMillis = System.currentTimeMillis();
17             this.n = currentTimeMillis - this.k;
18             ag.b = (fVar.d + ((currentTimeMillis - this.k) / 2)) - currentTimeMillis;
19             if (Lists.isEmpty(fVar.a)) {
20                 return protoApiResult;
21             }
               //省略......
```

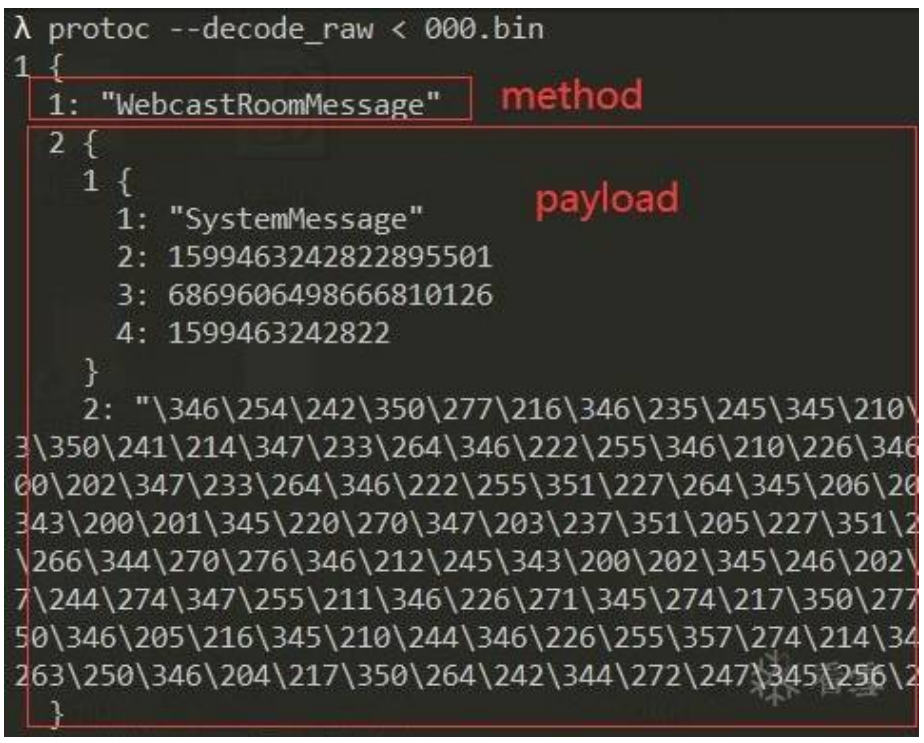
其中参数(b bVar)权当作http请求后的body即可,无关紧要,重点是com.bytedance.android.livesdkapi.message.f 这个类,进去看一下类里的结构:

```java
package com.bytedance.android.livesdkapi.message;
public class f {
    @SerializedName("messages")
    public List<a> a;
    @SerializedName("cursor")
    public String b;
    @SerializedName("fetch_interval")
    public long c;
    @SerializedName("now")
    public long d;
    @SerializedName("internal_ext")
    public String e;
    public static final class a {
        @SerializedName("method")   //这个实际上是类名,根据这个字段获取对应解析的类.
        public String a;
        @SerializedName("payload")  //解析的数据内容,同样是protobuf.
        public byte[] b;
    }
}
```

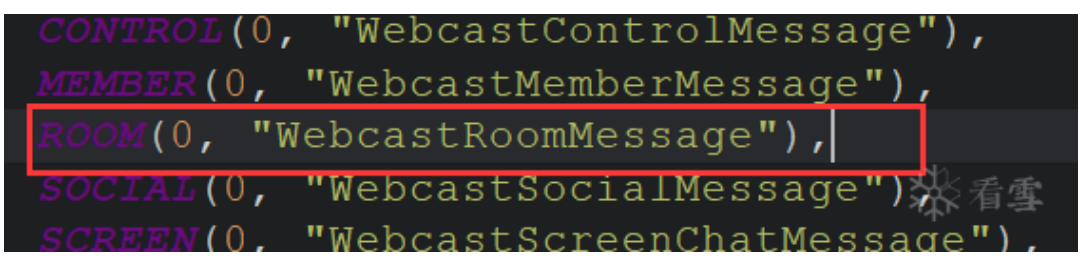
对比一下解析出来的数据图,发现对应tag:2,3,4,5



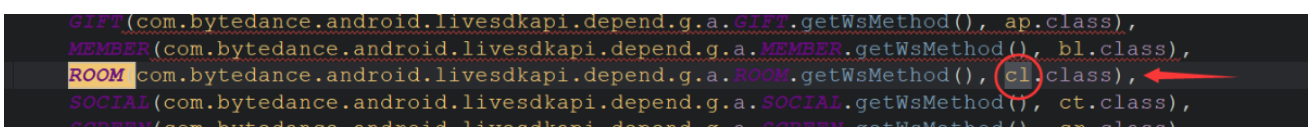
tag1是个数组,类型是内部类a, 内部类中剩下的2个字段是用于动态定位接下来解析的类:
method => 解析剩下数据包中的类名, payload => 数据内容,method 也就是这部分:



分析到此数据结构有了大致了解,com.bytedance.android.livesdkapi.message.f是某音弹幕数据的最外层,用来保存直播间弹幕所有的消息类型,从格式上不难理解,会有多种method,这里先以"WebcastRoomMessage"为例查找对应关系:
在代码里搜索一下很快找到了 com.bytedance.android.livesdkapi.depend.g.a:

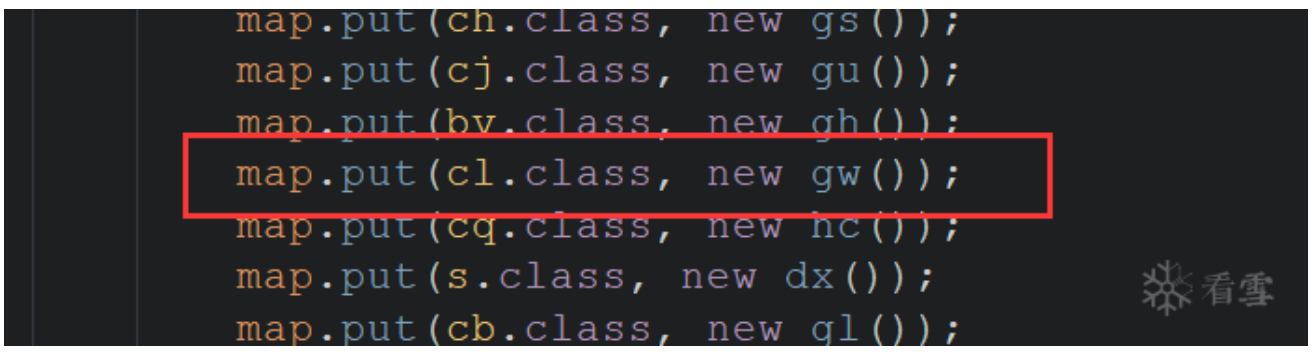


找到对这个枚举的引用,定位到 com.bytedance.android.livesdk.chatroom.bl.a:

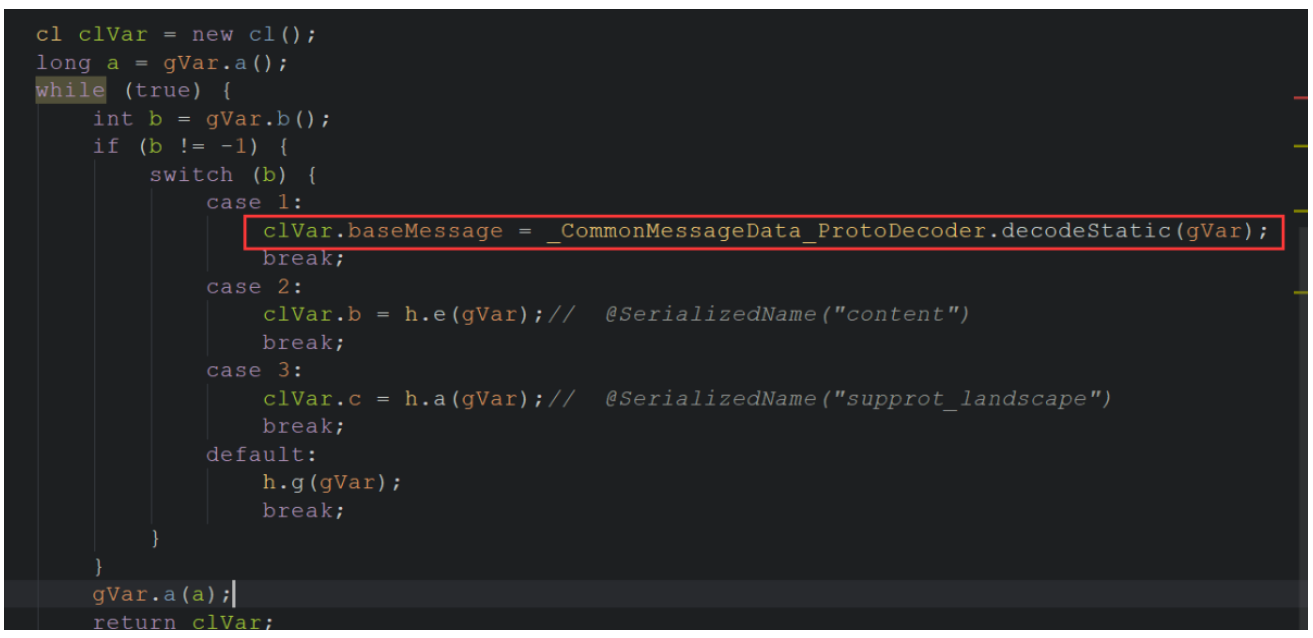


```java
public class cl extends d {             //注意继承关系
    public static ChangeQuickRedirect a;
    @SerializedName("content")
    public String b;
    @SerializedName("supprot_landscape")
    public boolean c;
    public cl() {
        this.type = com.bytedance.android.livesdkapi.depend.g.a.ROOM;    <-枚举类型
    }
```

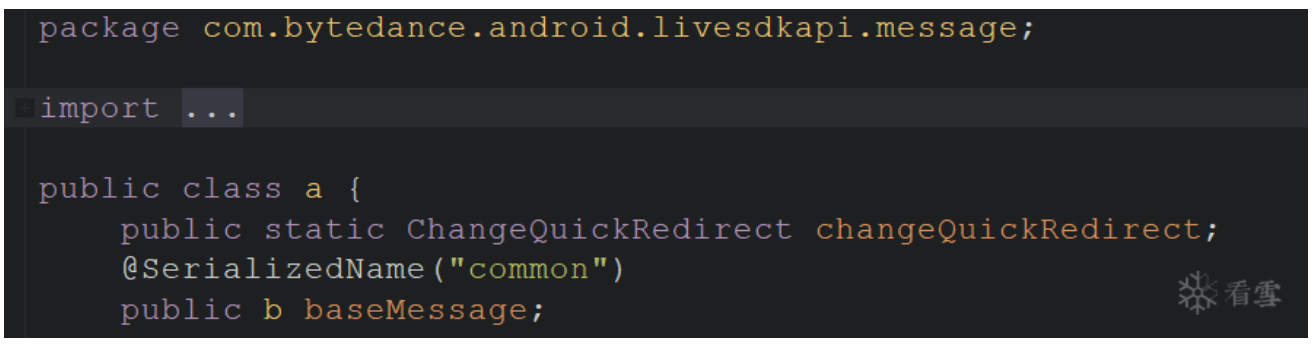

"WebcastRoomMessage"中只有2个字段,"content"和"supprot_landscape",可是根据工具解析出来的数据可以看到,还有一大坨呢,剩下的数据在哪,并且这2个的tag排列关系又是什么? 那只能继续查找cl这个类是怎么解析的,经过一番查找,找到了解析对应关系,com.bytedance.android.live.base.model.proto.d , 在这里可以看到负责解析cl.class的类是 gw.class:



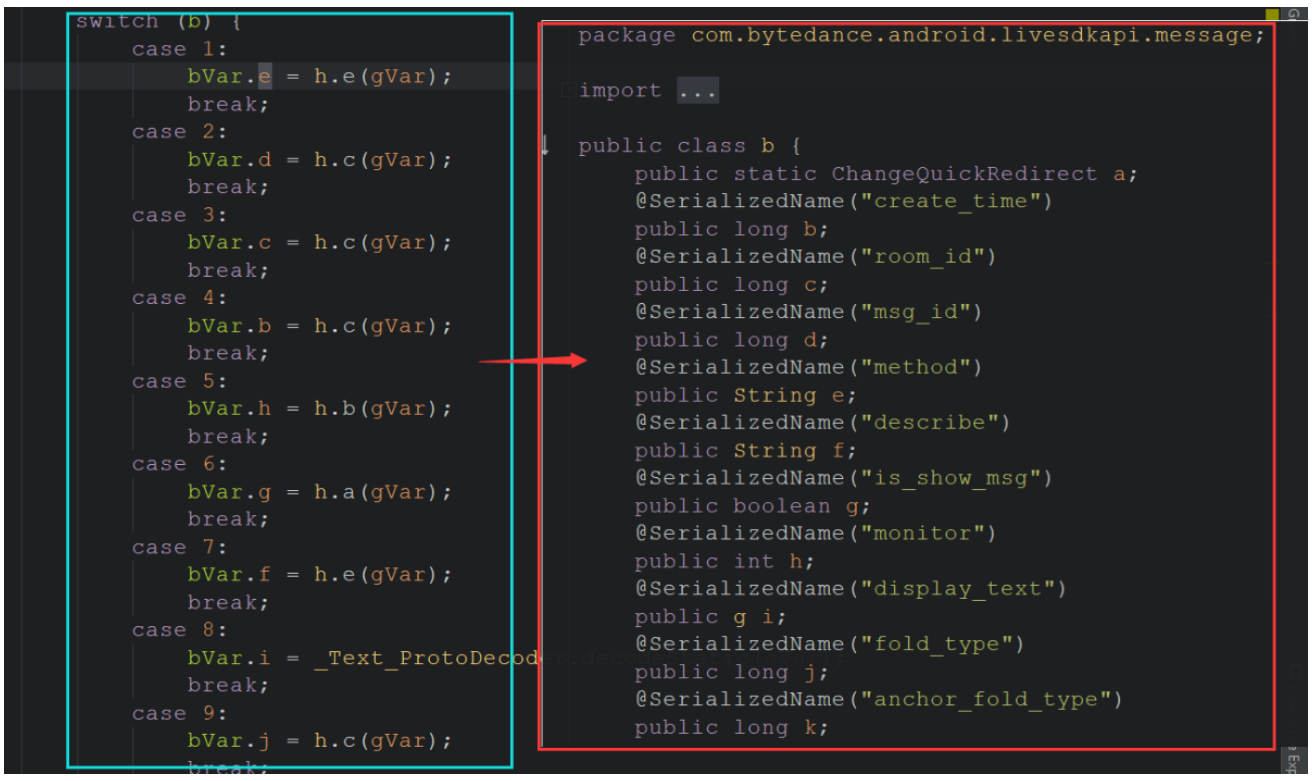
进入gw.class:



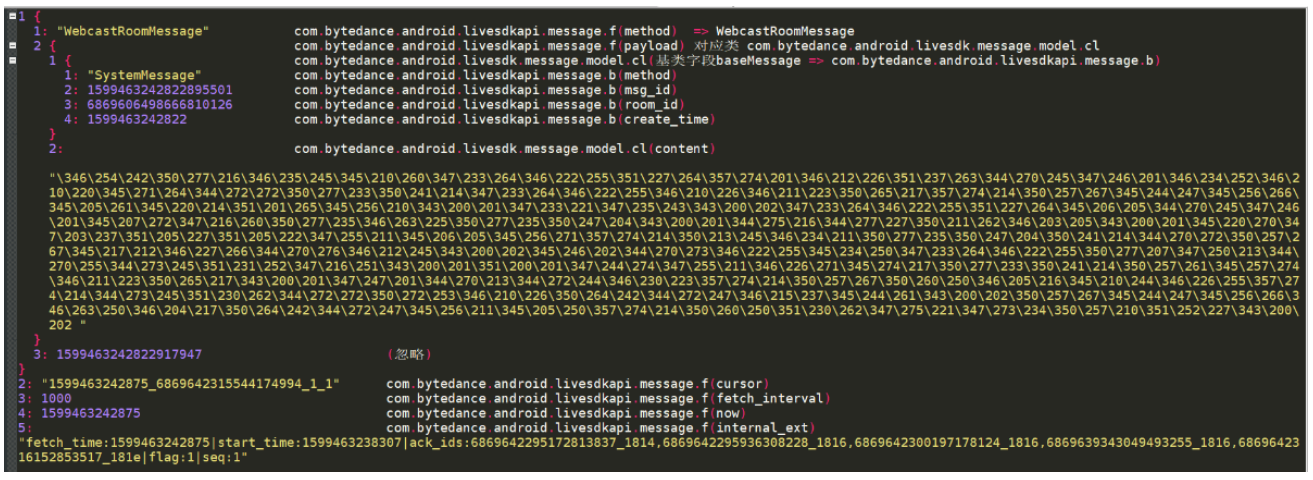
在这里知道了"content"和"supprot_landscape"分别对应tag 2和3,还有第三个名为"baseMessage"的字段要解析,这个是cl.class的基类成员,通过多个数据包验证得知,所有method中,都继承同一个基类:



之前没解析出来的数据也在其中,看下这个名为baseMessage成员的内部结构,并找一下"tag表":



分析到此,弹幕结构渐渐清晰了, 如下图:



现在可以根据分析结果开始写proto文件了.

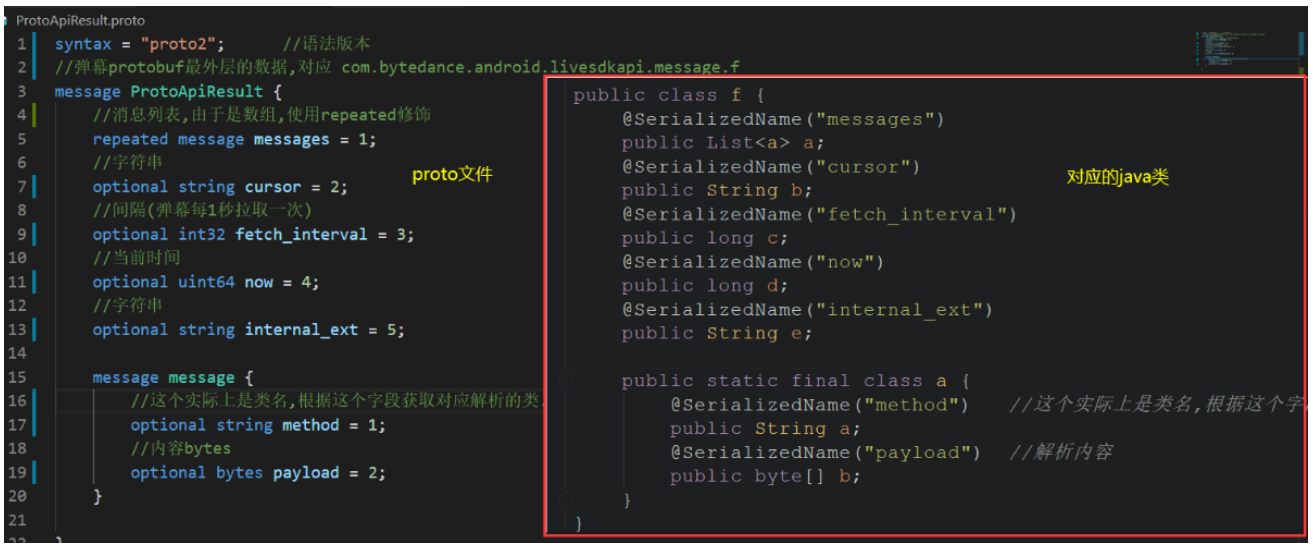
0x03：proto文件编写

关于proto文件语法网上资料很多,这里先说几个关键的,proto语法有"proto2"和"proto3",我没看出来对于咱们逆向人员选择"proto3"有什么用处,并且3的语法是不需要字段修饰符的,这只会让我们在写proto文件的时候更加迷糊,所以我们选择"proto2"来写.

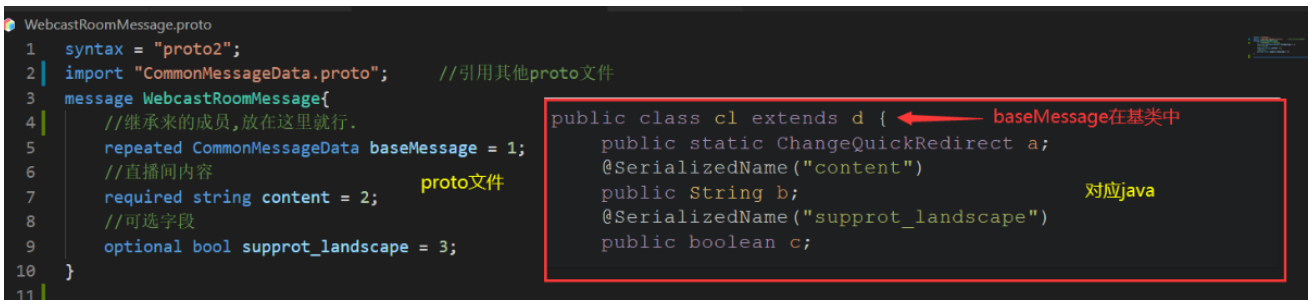
或许你已经发现,基类(baseMessage)有很多字段,怎么解析出来的就"method","msg_id","room_id","create_time"这么几个,其实这也是很正常的情况,并不是每一个字段都是必选的.proto有3个字段修饰符required(必选),optional(可选),repeated(重复),说白了就是咱们的proto文件不使用required,只用optional,因为咱们也不知道哪个字段什么时候才有值,如果是数组一律用repeated修饰.

注意proto的数据类型,以java为例.String => string ; int => int32; long => uint64;...更多类型可自行查找资料.

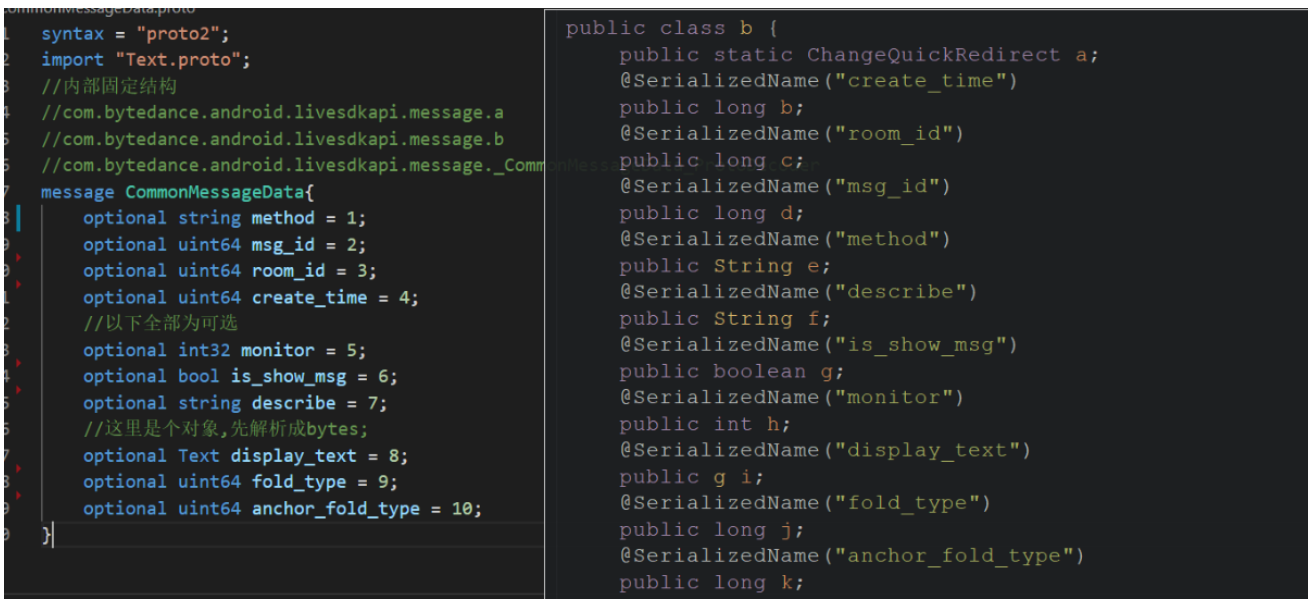
开始吧,先写一份 ProtoApiResult.proto,对应上面的 com.bytedance.android.livesdkapi.message.f ,用来保存弹幕整体数据:



外层有了,现在写"WebcastRoomMessage"对应的类:



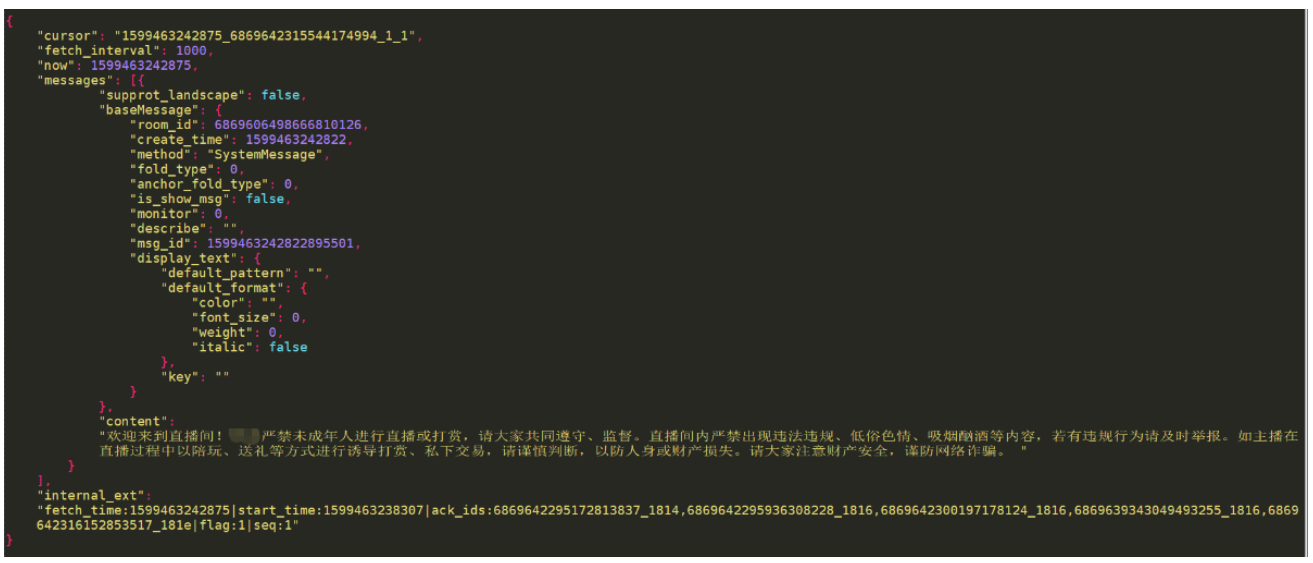
接下来最后一个"CommonMessageData":



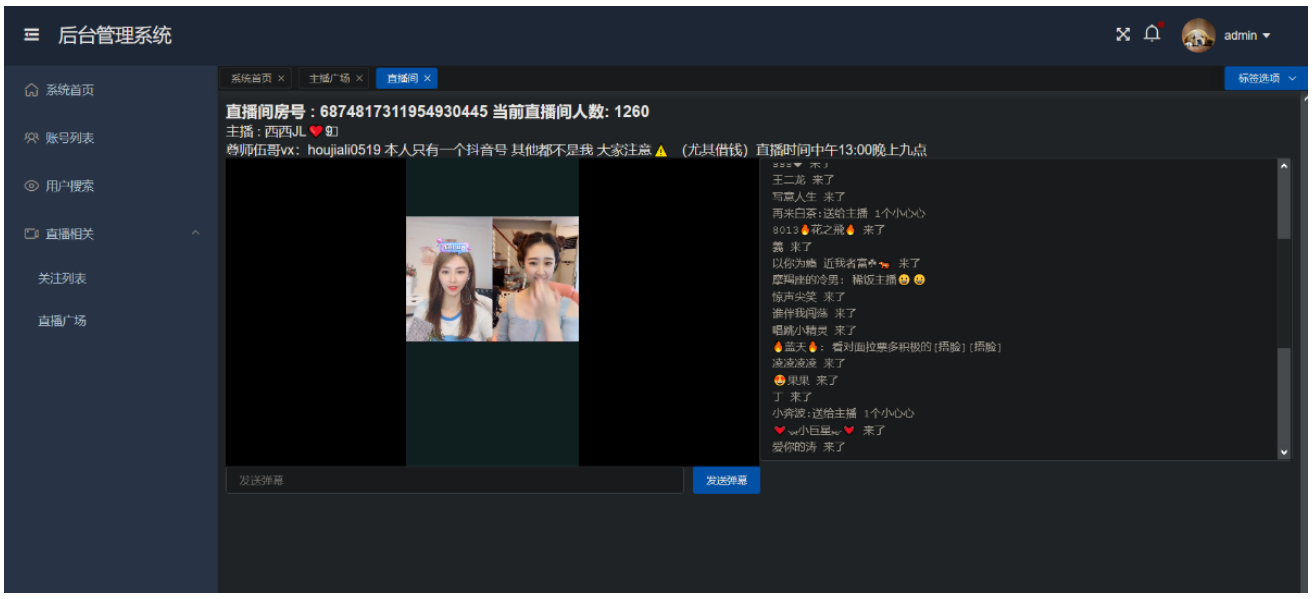
注意这里的display_text字段,这还是一个对象,里面又是N多字段,这里我已经解析过了, 大家自己在分析过程中如果不想解析那么多,可以删除不需要的字段,或者将类型定义为bytes即可.只要tag不乱就行.

OK,将.proto打包成java文件: protoc --java_out=out .*.proto

之后在out目录就生成了可以使用的java文件, 导入到项目中即可,最后看下解码效果:



嗯,费了一大堆事最后只解码一个系统提示...,这也没办法,某音一次请求动辄数万字节的返回数据,用那些做例子的话,我这图都没法截,迫不得已找了个最小的,根据上述流程其实其他类型的数据解析都是一样的,没什么区别,最后上个最终演示效果:



首次发帖,多多关照.

[公告]请完善个人简历信息，好工作来找你！

收藏 · 5　点赞 · 3　打赏　分享

最新回复 (2)

LowRebSwrd 5 2 13小时前　2 楼　0

版主

赞，还有个后台，

小豆芽 2 13小时前　3 楼　0

极客

赞，还有个后台，

wx_nu无情

内容

回帖　表情　高级回复

返回