包名：dHYuZGFubWFrdS5iaWxp

版本：6.24.0

播放量接口

播放视频-播放次数加1

抓包分析

在点开视频前清除一下日志，然后去抓包，会抓到很多个，然后逐一查看，重新发包，看播放量会不会上涨，多次测试发现是这个请求：

请求网址：https://api.bilibili.com/x/report/click/android2
请求方式：POST
请求头：

```
        content-length          320
        buvid                   XY2CAAD2CD8168403CCB095E408A467773352
        device-id               bg0-CTOOb1ZvDTxaJlom
        fp_local
efbaa891c759dbcacebbf89bb3517b7f20250423085841a27a01b53814242b3b
        fp_remote
efbaa891c759dbcacebbf89bb3517b7f20241207150740ab3e564147129a8976
        session_id              fafa8e36
        env                     prod
        app-key                 android
        user-agent              Mozilla/5.0 BiliDroid/6.24.0
(bbcallen@gmail.com) os/android model/Pixel mobi_app/android build/6240300
channel/xxl_gdt_wm_253 innerVer/6240300 osVer/10 network/2
        bili-bridge-engine      cronet
        content-type            application/octet-stream
        accept-encoding         gzip, deflate, br
```

请求体：是加了密的

请求体

我们从请求网址入手吧，去jadx中搜索一下找找入口。

找到了定义的请求接口：


```
11
12 /* compiled from: BL */
13 @BaseUrl("https://api.bilibili.com")
14 /* loaded from: classes5.dex */
15 public interface a {
16     @POST("/x/report/click/android2")
17     @RequestInterceptor(com.bilibili.okretro.f.c.class)
18     com.bilibili.okretro.d.a<String> reportClick(@Body c0 c0Var);
```

我们去看看**reportClick**的用例，这个 `c0 c0Var` 就是请求体

```
        public final void a() {
            long j2;
            long i = c2.f.f.c.j.a.i() / 1000;
            c2.f.b0.c.b.b.a.a E = c2.f.b0.c.b.b.a.a.E();
            x.h(E, "EnvironmentPrefHelper.getInstance()");
            long A = E.A();
            if (A == -1) {
                c2.f.b0.c.b.b.a.a E2 = c2.f.b0.c.b.b.a.a.E();
                x.h(E2, "EnvironmentPrefHelper.getInstance()");
                E2.V(i);
                j2 = i;
            } else {
                j2 = A;
            }
            c0 create = c0.create(w.d(com.hpplay.sdk.source.protocol.h.E), d.this.H7(this.b.a(), this.b.b(), this.b.h(), i, j2, this.b.n(), this.b.m(), this.b.k(), this.b.c(), this.b.e(), this.b.l(), this.b.f()));
            x.h(create, "RequestBody.create(Media…ion/octet-stream\"), body)");
            l<String> execute = ((tv.danmaku.biliplayerimpl.report.heartbeat.a) com.bilibili.okretro.c.a(tv.danmaku.biliplayerimpl.report.heartbeat.a.class)).reportClick(create).execute();
            int b = execute.b();
            String h = execute.h();
            BLog.i("HeartBeatTracker", "player report click(vv): responseCode:" + b + ", responseMsg:" + h + ", responseBody:" + execute.a());
        }
```

可以看到在调用这个**reportClick**发送请求的时候，传进去的是`create`,它是一个`c0`类的对象，这个就是请求体。

```
        public final void a() {
            long j2;
            long i = c2.f.f.c.j.a.i() / 1000;
            c2.f.b0.c.b.b.a.a E = c2.f.b0.c.b.b.a.a.E();
            x.h(E, "EnvironmentPrefHelper.getInstance()");
            long A = E.A();
            if (A == -1) {
                c2.f.b0.c.b.b.a.a E2 = c2.f.b0.c.b.b.a.a.E();
                x.h(E2, "EnvironmentPrefHelper.getInstance()");
                E2.V(i);
                j2 = i;
            } else {
                j2 = A;
            }
            c0 create = c0.create(w.d(com.hpplay.sdk.source.protocol.h.E), d.this.H7(this.b.a(), this.b.b(), this.b.h(), i, j2, this.b.n(), this.b.m(), this.b.k(), this.b.c(), this.b.e(), this.b.l(), this.b.f()));
            x.h(create, "RequestBody.create(Media…ion/octet-stream\"), body)");
            l<String> execute = ((tv.danmaku.biliplayerimpl.report.heartbeat.a) com.bilibili.okretro.c.a(tv.danmaku.biliplayerimpl.report.heartbeat.a.class)).reportClick(create).execute();
            int b = execute.b();
            String h = execute.h();
            BLog.i("HeartBeatTracker", "player report click(vv): responseCode:" + b + ", responseMsg:" + h + ", responseBody:" + execute.a());
        }

        @Override // java.util.concurrent.Callable
        public /* bridge */ /* synthetic */ Object call() {
            a();
            return kotlin.w.a;
        }
    }
```

这个`create`是通过调用`c0.create( w.d(参数...),d.this.H7(参数...) )`得到的，传进去了`w.d(参数...)`和`d.this.H7(参数...)`的返回值

`c0.create`代码如下：

```
c0.create:
public static c0 create(w wVar, byte[] bArr) {
    return create(wVar, bArr, 0, bArr.length);
}
```

在`c0.create`中又调用了 `create(wVar, bArr, 0, bArr.length)`

我们去看看`create(wVar, bArr, 0, bArr.length)`的代码逻辑：

```
public static c0 create(w wVar, byte[] bArr, int i, int i2) {
    if (bArr != null) {
        okhttp3.j0.c.f(bArr.length, i, i2);
        return new b(wVar, i2, bArr, i);
    }
    throw new NullPointerException("content == null");
}

okhttp3.j0.c.f(bArr.length, i, i2)的代码逻辑如下：
//这里是判断了一下如果(j4 | j5) < 0 || j4 > j2 || j2 - j4 < j5的话就报错
```

所以在`create(wVar, bArr, 0, bArr.length)`中最终调用了：`new b(wVar, i2, bArr, i)`

我们去看看`new b(wVar, i2, bArr, i)`的代码逻辑：

```
 public class b extends c0 {
     final  w a;
     final  int b;
     final  byte[] f22525c;
     final   int d;

     b(w wVar, int i, byte[] bArr, int i2) {
         this.a = wVar;
         this.b = i;
         this.f22525c = bArr;
         this.d = i2;
     }

...
 }
```

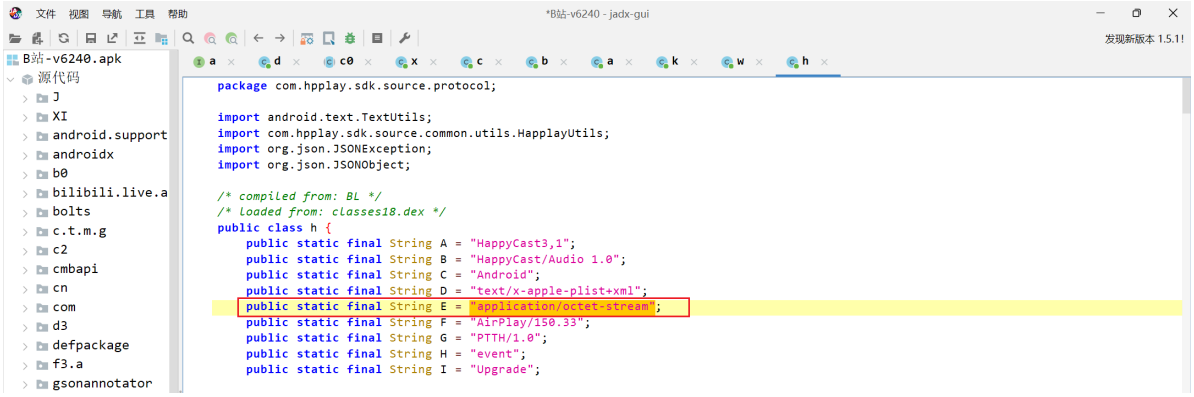
然后new了一个b的对象，在调用**reportClick**的时候把b传了进去，所以这个create是一个b类的对象，b类又是继承了c0类的。

我们现在再去分析一下`c0.create(w.d(参数...),d.this.H7(参数...))`的参数：

先看这个`w.d(参数...)`

它的原型是：`w.d(com.hpplay.sdk.source.protocol.h.E)`

其中`com.hpplay.sdk.source.protocol.h.E`是一个常量字符串：`application/octet-stream`：



So这里就是`w.d("application/octet-stream")`,我们去看看`w.d`干了什么：

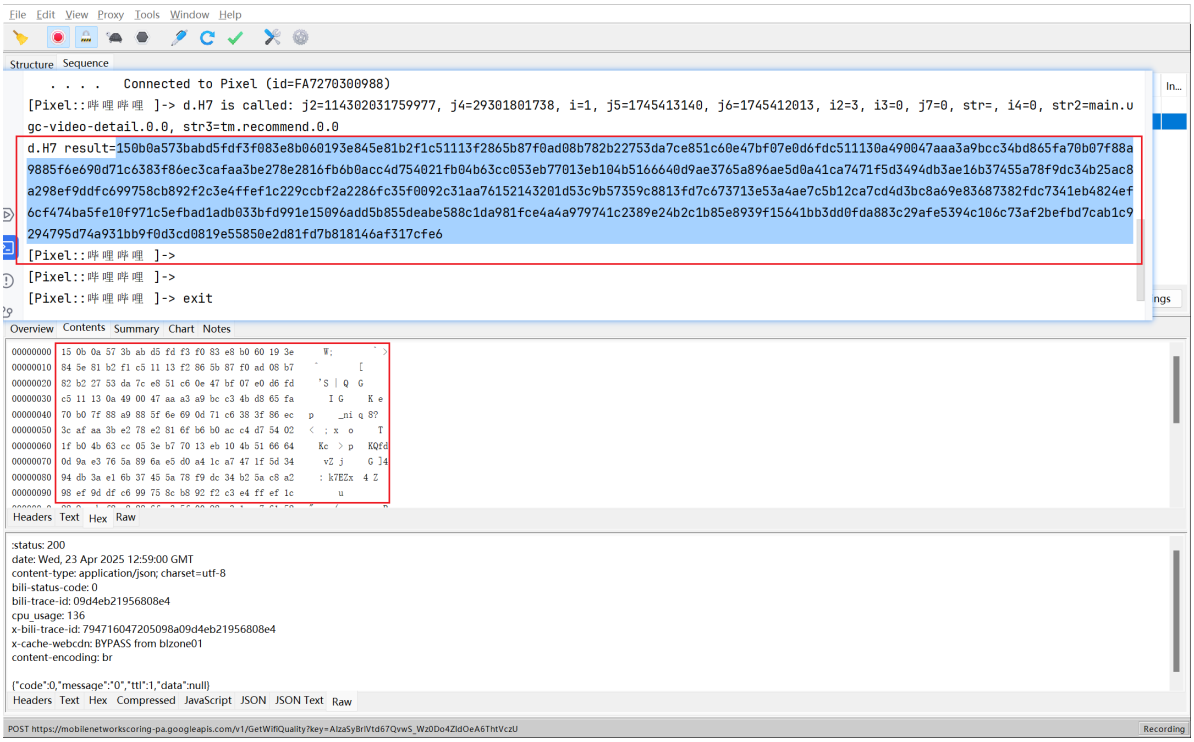
在`w.d`这个函数中，主要是做了： **一个结构化的媒体类型解析器**，通过正则表达式匹配和严格校验，确保输入字符串符合规范，并提取关键信息（类型、子类型、字符集）。其设计体现了对协议规范的严格遵守和鲁棒性要求，适用于需要处理标准化字符串格式的底层库或框架。

其实就是确定了`content-type`,即内容类型。

再看另一个参数`d.this.H7(参数...)`

它的原型是：`d.this.H7( this.b.a(), this.b.b(), this.b.h(), i, j2, this.b.n(), this.b.m(), this.b.k(), this.b.c(), this.b.e(), this.b.l(), this.b.f() )`

我们先去hook一下这个H7函数，看看参数和返回值:



我们发现这个返回值就是我们的请求体，所以我们需要搞定H7的逻辑以及调用H7时传进去的参数，这样就能解决这个请求体了。

先去看看`d.this.H7(参数...)`的代码逻辑:

```
public final byte[] H7(long j2, long j4, int i, long j5, long j6, int i2, int
i3, long j7, String str, int i4, String str2, String str3) throws Exception {
    long j8;
    int i5;
    Application f2 = BiliContext.f();
    com.bilibili.lib.accounts.b client = com.bilibili.lib.accounts.b.f(f2);
    AccountInfo h = BiliAccountInfo.f.a().h();
    if (h != null) {
        j8 = h.getMid();
        i5 = h.getLevel();
    } else {
        j8 = 0;
        i5 = 0;
    }
    TreeMap treeMap = new TreeMap();
    treeMap.put("aid", String.valueOf(j2));
    treeMap.put("cid", String.valueOf(j4));
```

```
    treeMap.put("part", String.valueOf(i));
    treeMap.put(EditCustomizeSticker.TAG_MID, String.valueOf(j8));
    treeMap.put("lv", String.valueOf(i5));
    treeMap.put("ftime", String.valueOf(j6));
    treeMap.put("stime", String.valueOf(j5));
    treeMap.put("did", com.bilibili.lib.biliid.utils.f.a.c(f2));
    treeMap.put("type", String.valueOf(i2));
    treeMap.put("sub_type", String.valueOf(i3));
    treeMap.put("sid", String.valueOf(j7));
    treeMap.put("epid", str);
    treeMap.put("auto_play", String.valueOf(i4));
    x.h(client, "client");
    if (client.r()) {
        treeMap.put("access_key", client.g());
    }
    treeMap.put("build", String.valueOf(com.bilibili.api.a.f()));
    treeMap.put("mobi_app", com.bilibili.api.a.l());
    treeMap.put("spmid", str2);
    treeMap.put("from_spmid", str3);
    StringBuilder sb = new StringBuilder();
    for (Map.Entry entry : treeMap.entrySet()) {
        String str4 = (String) entry.getValue();
        sb.append((String) entry.getKey());
        sb.append('=');
        if (str4 == null) {
            str4 = "";
        }
        sb.append(str4);
        sb.append('&');
    }
    sb.deleteCharAt(sb.length() - 1);
    String sb2 = sb.toString();
    x.h(sb2, "builder.toString()");
    String b2 = t3.a.i.a.a.a.b.e.b(sb2);
    BLog.i("HeartBeatTracker", "player report click(vv), params: " + sb2 + " &
sign=" + b2);
    sb.append("&sign=");
    sb.append(b2);
    String sb3 = sb.toString();
    x.h(sb3, "builder.toString()");
    return t3.a.i.a.a.a.b.e.a(sb3);
}
```

大概看了一下，`d.this.H7(参数...)`主要是包括五部分：

1、生成一些上层没有传过来的参数

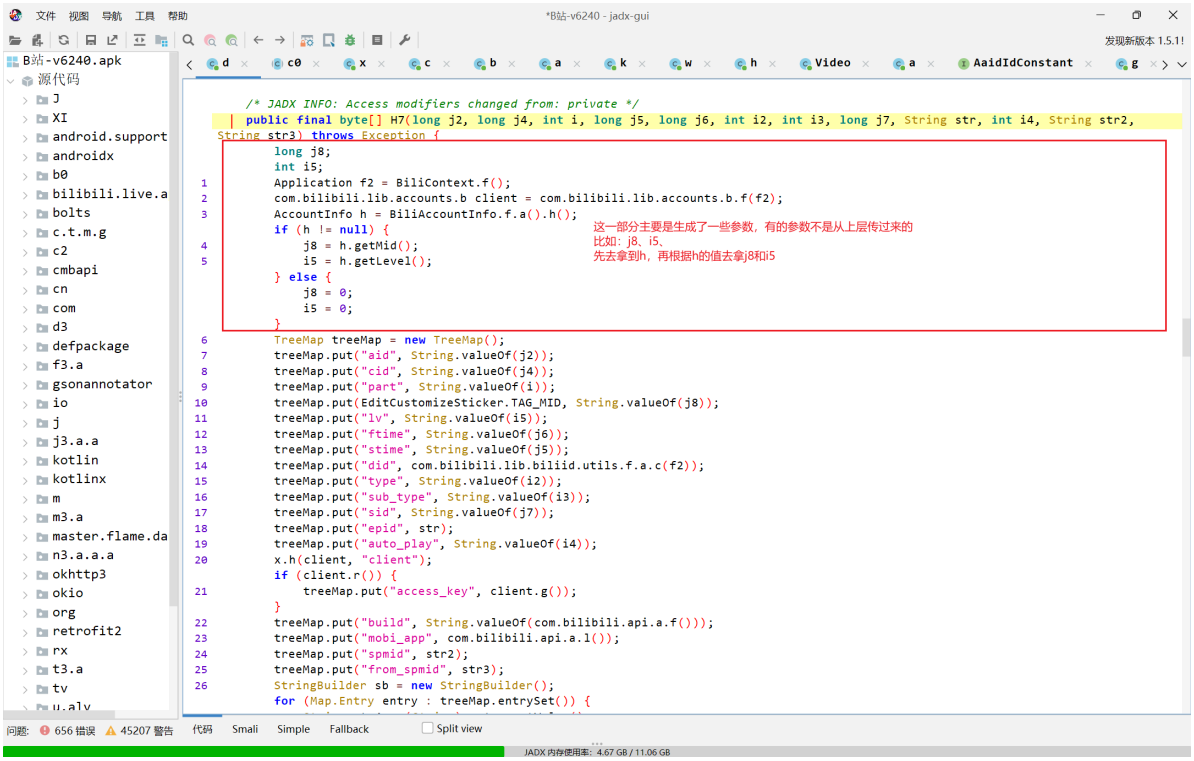
2、把这些参数put到treeMap中

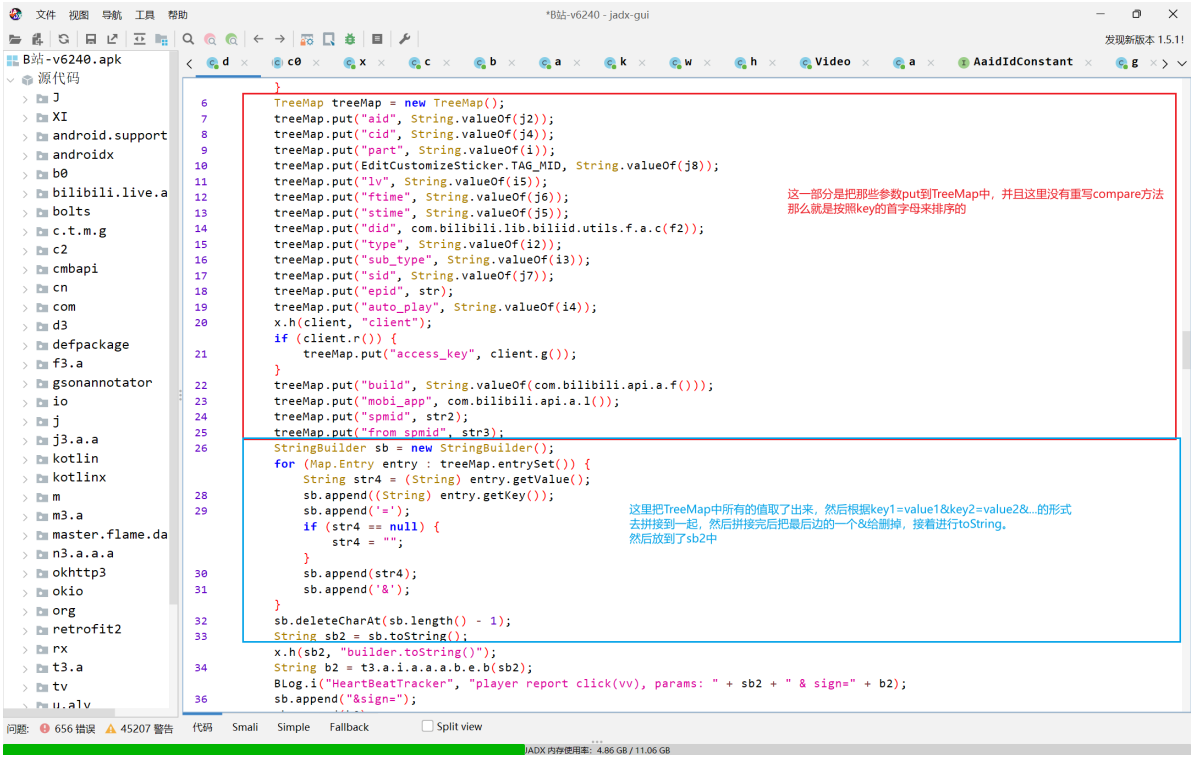
3、把treeMap遍历出来，使用key1=value1&key2=value2&key3=value3&...的方式拼接起来

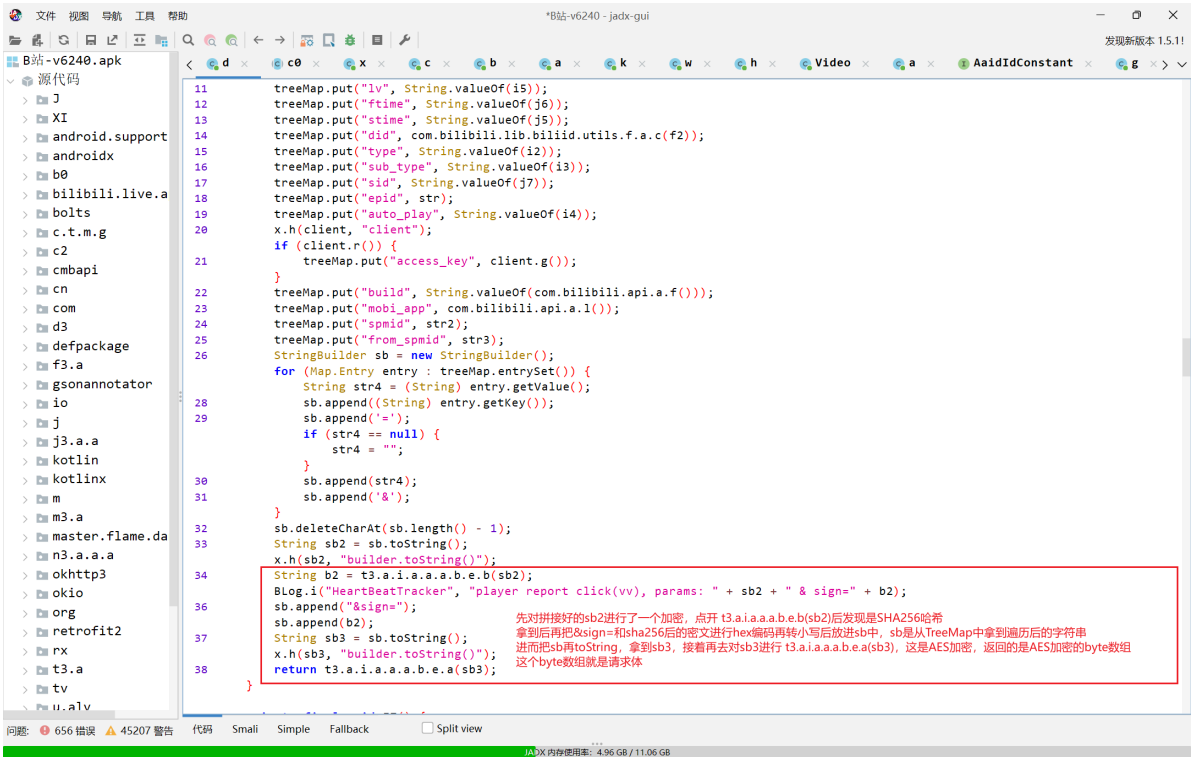
4、对这些拼接起来的参数进行`t3.a.i.a.a.a.b.e.b(sb2);`这个是sha256加密，然后再拼接一个&sign=，再拼接一个密文，也就是说对这些参数进行sha256后就是sign的值。sign是进行sha256后先hex编码然后转小写

5、把sign拼进去后再调用`t3.a.i.a.a.a.b.e.a(sb3)`这是一个AES加密，然后返回去就是请求体，AES加密后是直接返回的byte数组

逐一分析一下代码：







我们现在先去hook一下这个sha256加密，看看它的明文都是什么：

```
aid=114272302600996          &
auto_play=0                  &
build=6240300                &
cid=29210513232              &
did=bg0-CTOOb1ZvDTxaJlom     &
epid=                        &
from_spmid=tm.recommend.0.0 &
ftime=1745369985             &
lv=0                         &
mid=0                        &
mobi_app=android             &
part=1                       &
sid=0                        &
spmid=main.ugc-video-detail.0.0 &
stime=1745393699             &
sub_type=0                   &
type=3
```

我们现在分析一下每个参数怎么来的，进而就确定了sign的明文 了

aid与cid

```
treeMap.put("aid", String.valueOf(j2));
```

它是通过j2的值，j2是从上层传过来的，往上看：

```
d.this.H7(this.b.a(), this.b.b(), this.b.h(), i, j2, this.b.n(), this.b.m(),
this.b.k(), this.b.c(), this.b.e(), this.b.l(), this.b.f())
```

j2就是this.b.a()

this.b.a()方法如下：

```
private long a;

public final long a() {
    return this.a;
}
```

a是这么赋值的：

```
 public final void p(long j2) {
     this.a = j2;
 }
```

我们去看看p的用例：

有很多，我们去hook一下这个p，然后去打印一下堆栈，看看打开视频的时候是在哪里调用的，

再打开视频的时候， 有三个地方都调用了p方法，我们都去看看：

```
 第一处：
 at tv.danmaku.biliplayerv2.service.Video$h.p(Native Method)
        at tv.danmaku.bili.ui.video.playerv2.r.s(BL:2)
        at tv.danmaku.biliplayerv2.service.report.d.S(BL:4)
        at tv.danmaku.biliplayerv2.service.report.d$f.o(BL:1)
        at
tv.danmaku.biliplayerimpl.videodirector.VideosPlayDirectorService$e$e.b(BL:3)
        at
tv.danmaku.biliplayerimpl.videodirector.VideosPlayDirectorService$e$e.a(BL:1)
        at tv.danmaku.biliplayerv2.t.n$c.a(BL:3)
        at
tv.danmaku.biliplayerimpl.videodirector.VideosPlayDirectorService$e.d(BL:2)
        at tv.danmaku.biliplayerv2.service.NormalVideoPlayHandler.r(BL:9)
```

第二处：

```
 at tv.danmaku.biliplayerv2.service.Video$h.p(Native Method)
        at tv.danmaku.bili.ui.video.playerv2.r.s(BL:2)
        at
com.bilibili.ad.adview.videodetail.danmakuv2.AdDanmakuService$m.o(BL:5)
        at
tv.danmaku.biliplayerimpl.videodirector.VideosPlayDirectorService$e$e.b(BL:3)
        at
tv.danmaku.biliplayerimpl.videodirector.VideosPlayDirectorService$e$e.a(BL:1)
        at tv.danmaku.biliplayerv2.t.n$c.a(BL:3)
        at
tv.danmaku.biliplayerimpl.videodirector.VideosPlayDirectorService$e.d(BL:
```

第三处：

```
  at tv.danmaku.biliplayerv2.service.Video$h.p(Native Method)
        at tv.danmaku.bili.ui.video.playerv2.r.s(BL:2)
        at tv.danmaku.biliplayerv2.service.NormalVideoPlayHandler$c.e(BL:10)
        at tv.danmaku.biliplayerv2.service.resolve.j.q(BL:4)
        at t3.a.d.m.a$a.handleMessage(BL:18)
```

先看第一处：

```
 at tv.danmaku.biliplayerv2.service.Video$h.p(Native Method)
        at tv.danmaku.bili.ui.video.playerv2.r.s(BL:2)
        at tv.danmaku.biliplayerv2.service.report.d.S(BL:4)
        at tv.danmaku.biliplayerv2.service.report.d$f.o(BL:1)
        at
tv.danmaku.biliplayerimpl.videodirector.VideosPlayDirectorService$e$e.b(BL:3)
        at
tv.danmaku.biliplayerimpl.videodirector.VideosPlayDirectorService$e$e.a(BL:1)
        at tv.danmaku.biliplayerv2.t.n$c.a(BL:3)
        at
tv.danmaku.biliplayerimpl.videodirector.VideosPlayDirectorService$e.d(BL:2)
        at tv.danmaku.biliplayerv2.service.NormalVideoPlayHandler.r(BL:9)
```

我们先去看看tv.danmaku.bili.ui.video.playerv2.r.s(BL:2)
看看代码逻辑：

```
 private long s;

 public Video.h s() {
        Video.h hVar = new Video.h();
        hVar.p(this.s);
 }
```

再去看看给s赋值的地方，查找s的用例：

```
public final void n0(long j2) {
        this.s = j2;
    }
```

我们再去查找n0的用例：
去hook一下n0，打印个堆栈：

```
    at tv.danmaku.bili.ui.video.playerv2.r.n0(Native Method)
        at tv.danmaku.bili.ui.video.playerv2.datasource.c.v1(BL:87)
        at tv.danmaku.bili.ui.video.playerv2.datasource.b.o1(BL:5)
```

```
        at tv.danmaku.bili.ui.video.player.VideoDetailPlayer.sO(BL:7)
        at tv.danmaku.bili.ui.video.player.VideoDetailPlayer.fs(BL:16)
        at tv.danmaku.bili.ui.video.x.j(BL:23)
```

我们去看看这个tv.danmaku.bili.ui.video.playerv2.datasource.c.v1

```
    public void v1(BiliVideoDetail biliVideoDetail, Bundle extra) {
            ...

        rVar2.nO(j2);


            ...
}
```

再去看看这个j2是怎么来的：

```
     long j2 = extra.getLong("avid");
```

通过extra.getLong("avid")得到的

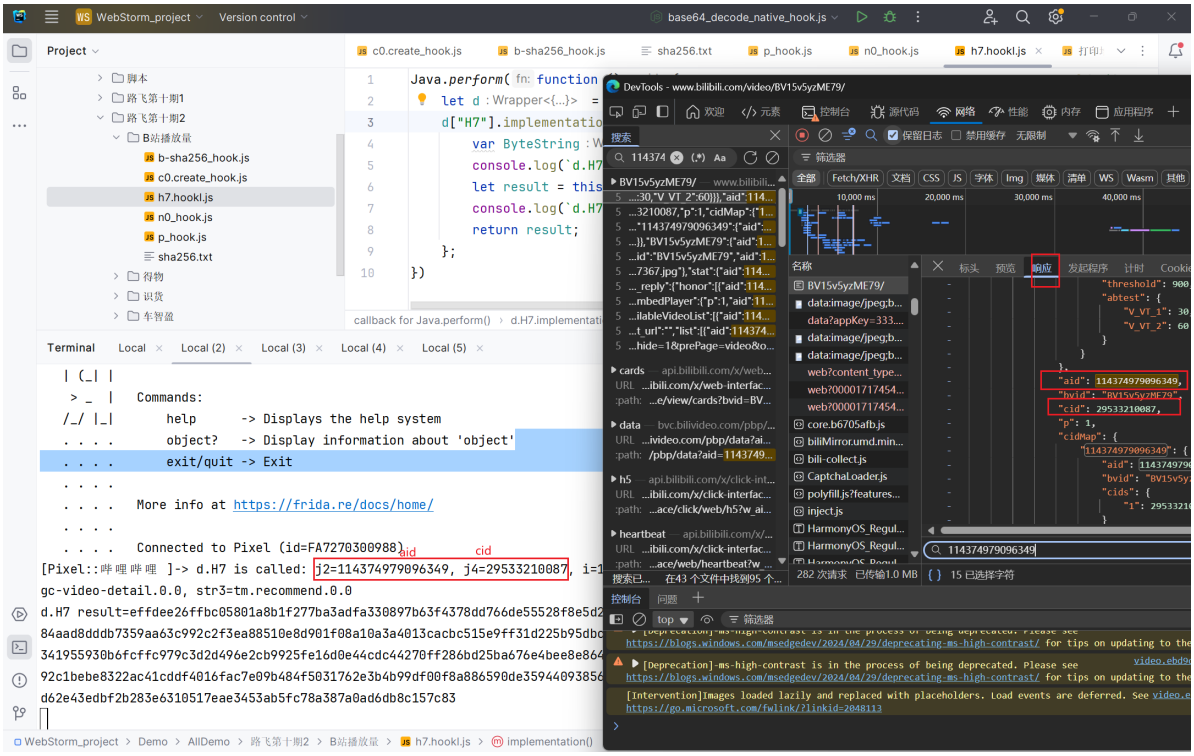
再往下就走不通了，看不了extra.getLong("avid")的代码逻辑了，先分析到这里，看看其他的。

第二处：

```
     at tv.danmaku.biliplayerv2.service.Video$h.p(Native Method)
        at tv.danmaku.bili.ui.video.playerv2.r.s(BL:2)
        at
com.bilibili.ad.adview.videodetail.danmakuv2.AdDanmakuService$m.o(BL:5)
        at
tv.danmaku.biliplayerimpl.videodirector.VideosPlayDirectorService$e$e.b(BL:3)
        at
tv.danmaku.biliplayerimpl.videodirector.VideosPlayDirectorService$e$e.a(BL:1)
        at tv.danmaku.biliplayerv2.t.n$c.a(BL:3)
        at
tv.danmaku.biliplayerimpl.videodirector.VideosPlayDirectorService$e.d(BL:
```

我们发现第二处，第三处也是通过`tv.danmaku.bili.ui.video.playerv2.r.s(BL:2)`来调用的。

后边分析发现，这个aid和cid是用来标识唯一视频的



我们打开这个视频的时候，会在响应体中有这个aid和cid。

did

与请求头中device-id生成逻辑一样，可以去看device-id参数的分析，那里写的比较细致！

我们去看其他的参数，did

```
did=bg0-CTOOb1ZvDTxaJlom    &
```

我们看到这个did好像是加密的，我们先去看看did是怎么得到的：

```
Application f2 = BiliContext.f();
com.bilibili.lib.accounts.b client = com.bilibili.lib.accounts.b.f(f2);
treeMap.put("did", com.bilibili.lib.biliid.utils.f.a.c(f2));
```

可以看到是通过`BiliContext.f()`拿到的f2，f2是Application类的对象，应该就是Context。

然后就是`treeMap.put("did", com.bilibili.lib.biliid.utils.f.a.c(f2));`

put的时候传进去的value是:`com.bilibili.lib.biliid.utils.f.a.c(f2)`

我们去看看`com.bilibili.lib.biliid.utils.f.a.c(f2)`里边的操作：

可以看到这里边接收的参数就是Context类的对象，和之前分析的一样！

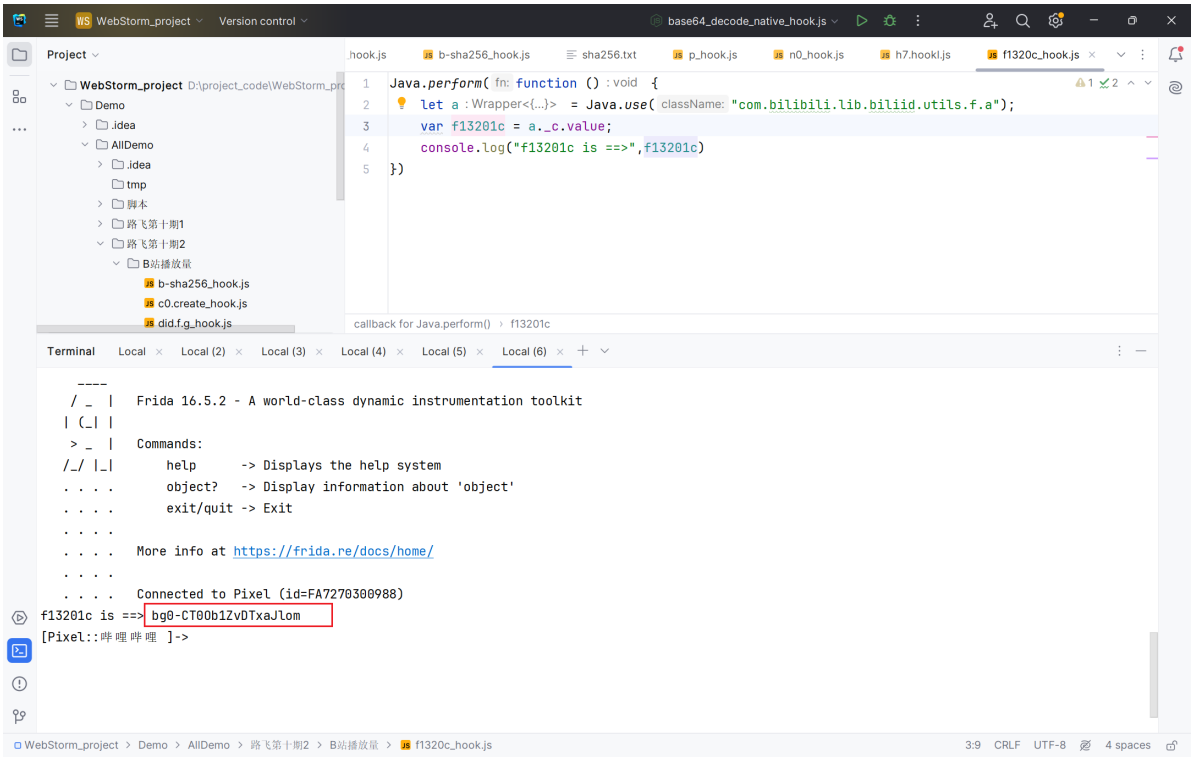
```
public static String c(@Nullable Context context) {
    if (TextUtils.isEmpty(f13201c)) {
        //进来先判断f13201c是否为空，不为空直接返回
        if (context == null) {
            return "";
        }
        //如果Context为null就返回空，显示不为空
        String f = c2.f.b0.c.a.e.k().f(context);
        //通过调用c2.f.b0.c.a.e.k().f(context)拿到一个字符串
```

```
            f13201c = f;
            //把字符串给了f13201c
            if (!TextUtils.isEmpty(f)) {
                return f13201c;
                //如果字符串不为空，就返回字符串

            }
            //如果f字符串为空，就执行下边的操作
            f13201c = g(context);
            //通过g(context)拿到一个字符串
            c2.f.b0.c.a.e.k().x(f13201c, context);
            return f13201c;
            //返回g(context)得到的字符串
        }
        return f13201c;
    }
```

我们同时hook一下`c2.f.b0.c.a.e.k().f(context)`和`g(context)`,看看走的哪个逻辑：

额，hook了这两个函数，都没走，那就说明，进来之后f13201c是有值的，然后就返回了

我们去hook一下这个f13201c：



`did=bg0-CT0Ob1ZvDTxaJlom`

并且和我们hook n0得到的是一样的

那我们去看看这个f13201c是在哪里赋值的：没有找到是在哪里赋值的，但是我们重启手机，重新安装app，这个值都没变化。但是换手机后会变化，应该是根据我们的手机信息来生成的。

我们再分析的过程中发现，如果f13201c没有值的话，他也会调用其他的两个方法去生成f13201c的值。我们在使用的过程中，也可以根据那两个函数的规则，去动态生成f13201c的值==>也就是说did的值==>也就是说did的值是非固定的。我们去看看那两个函数的规则。

`String f = c2.f.b0.c.a.e.k().f(context)`：

```
public String f(Context context) {
    SharedPreferences c3 = l().c();
    String string = c3.getString("persist.c.bl.did", null);
    if (!TextUtils.isEmpty(string)) {
        com.bilibili.lib.biliid.internal.storage.external.d.y(string);
        return string;
    }
    String f = com.bilibili.lib.biliid.internal.storage.external.d.f();
    if (!TextUtils.isEmpty(f)) {
        c3.edit().putString("persist.c.bl.did", f).apply();
    }
    return f;
}
```

这个是先去内存中找，找不到的话去xml中继续找。如果还找不到应该就会返回空，进而去另一个函数中生成，即`f13201c = g(context)`

我们再看`f13201c = g(context)`的规则:

```
static String g(Context context) {
    String f = f(context);
    if (f.length() < 4) {
        f = Settings.Secure.getString(context.getContentResolver(),
"android_id") + "@" + g.g(Build.MODEL);
    }
    return b(f);
}

public static String f(Context context) {
    StringBuffer stringBuffer = new StringBuffer();
    String j2 = j(context);
    if (j2 != null) {
        String lowerCase = j2.replaceAll("[^0-9A-Fa-f]", "").toLowerCase();
        if (k(lowerCase)) {
            stringBuffer.append(lowerCase);
        }
    }
    stringBuffer.append('|');
    String a2 = z.a("persist.service.bdroid.bdaddr");
    if (a2.length() > 0) {
        String lowerCase2 = a2.replaceAll("[^0-9A-Fa-f]", "").toLowerCase();
        if (k(lowerCase2)) {
            stringBuffer.append(lowerCase2);
        }
    }
    stringBuffer.append('|');
    String h = h();
    if (h != null) {
        stringBuffer.append(h.toLowerCase());
    }
    stringBuffer.append('|');
    String i = i();
    if (i != null) {
        stringBuffer.append(i.toLowerCase());
    }
```

```
    return stringBuffer.toString();
}

 public static String b(String str) {
     byte[] bytes = str.getBytes();
     bytes[0] = (byte) (bytes[0] ^ ((byte) (bytes.length & 255)));
     for (int i = 1; i < bytes.length; i++) {
         bytes[i] = (byte) ((bytes[i - 1] ^ bytes[i]) & 255);
     }
     try {
         return new String(Base64.encode(bytes, 11));
     } catch (Exception unused) {
         return str;
     }
```

这个函数的生成逻辑是：把 mac地址|蓝牙地址|设备总线|sn号 拼成字符串 ， 拼好后，再调用b方法进行一下编码

再使用过程中，可以采用这种方式去生成，mac，蓝牙地址，设备总线，sn我们都可以自己瞎写

stime与ftime

```
ftime=1745369985            &
stime=1745393699            &
```

接下来看stime和ftime

着看着像时间戳，一般情况请求中只有一个时间戳，但是这里stime和ftime都是时间戳

我们去看看这个stime和ftime的来路

```
treeMap.put("ftime", String.valueOf(j6));
treeMap.put("stime", String.valueOf(j5));
```

j5、j6都是从上层传过来的

```
 public final void a() {
     long j2;
     long i = c2.f.f.c.j.a.i() / 1000;
     c2.f.b0.c.b.b.a.a E = c2.f.b0.c.b.b.a.a.E();
     x.h(E, "EnvironmentPrefHelper.getInstance()");
     long A = E.A();
     if (A == -1) {
         c2.f.b0.c.b.b.a.a E2 = c2.f.b0.c.b.b.a.a.E();
         x.h(E2, "EnvironmentPrefHelper.getInstance()");
         E2.V(i);
         j2 = i;
     } else {
         j2 = A;
     }
    d.this.H7(this.b.a(), this.b.b(), this.b.h(), i, j2, this.b.n(), this.b.m(),
this.b.k(), this.b.c(), this.b.e(), this.b.l(), this.b.f())
    //j5对应i，j6对应j2
```

stime

先看i，也就是j5，也就是stime

```
long i = c2.f.f.c.j.a.i() / 1000;

 c2.f.f.c.j.a.i()代码逻辑：
public static final long i() {
        long currentTimeMillis;
        synchronized (a) {
            if (b != -1 && f1829c != -1) {
                currentTimeMillis = b + (SystemClock.elapsedRealtime() -
f1829c);
            }
            currentTimeMillis = System.currentTimeMillis();
        }
        return currentTimeMillis;
    }
//这里就是获取了当前的毫秒级时间戳，然后除1000，转为秒级时间戳
```

这里就相当于是获取了打开视频的时间

ftime

再看j2，也就是j6，也就是ftime

```
long j2;
long i = c2.f.f.c.j.a.i() / 1000;
//获取打开视频的秒级时间戳
c2.f.b0.c.b.b.a.a E = c2.f.b0.c.b.b.a.a.E();
//这里是进行了new a()，返回了一个a类的对象给了E
long A = E.A();
//通过调用E下边的A()方法，返回值给了A
if (A == -1) {
    c2.f.b0.c.b.b.a.a E2 = c2.f.b0.c.b.b.a.a.E();
    E2.V(i);
    j2 = i;
    //如果A==-1，就把打开的视频的时间给了j2，这里应该是没打开的话可能ftime就和stime的值一样
} else {
    j2 = A;
    //如果A不等于-1的话就把A的值给了j2，这样就拿到了ftime
}
```

我们现在去看看这个long A = E.A();的逻辑

```
public long A() {
        return c().getLong(e, -1L);
    }
```

这个ftime就是播放视频的时间

其他的参数都是固定的，我们直接固定下来就行

sign的明文整理：

```
aid=114272302600996          &   //视频id
auto_play=0                  &   //固定
build=6240300                &   //固定
cid=29210513232              &   //视频id
did=bg0-CTOOb1ZvDTxaJlom     &   //可以通过手机信息伪造
epid=                        &   //空
from_spmid=tm.recommend.0.0  &   //固定的
ftime=1745369985             &   //播放视频的时间
lv=0                         &   //固定
mid=0                        &   //固定
mobi_app=android             &   //固定
part=1                       &   //固定
sid=0                        &   //固定
spmid=main.ugc-video-detail.0.0 &   //固定
stime=1745393699             &   //打开视频的时间
sub_type=0                   &   //固定
type=3                           //固定
```

sign

现在去分析sign的生成，先去看代码逻辑：

```
String b2 = t3.a.i.a.a.a.b.e.b(sb2);
sb.append("&sign=");
sb.append(b2);
```

可以看到是通过`t3.a.i.a.a.a.b.e.b(sb2)`得到的sign。

其中sb2就是拼接好的字符串，我们去hook一下这个`t3.a.i.a.a.a.b.e.b(sb2)`方法

```
b.b is called: params=
aid=114386303713502&auto_play=0&build=6240300&cid=29566765103&did=bg0-
CTOOb1ZvDTxaJlom&epid=&from_spmid=tm.recommend.0.0&ftime=1745412013&lv=0&mid=0&m
obi_app=android&part=1&sid=0&spmid=main.ugc-video-
detail.0.0&stime=1745477677&sub_type=0&type=3

b.b result=ecf1df680fddad47df06a3109a2b94f27835d7a2cc4266d525dc6163264a3d20
```

我们去看看`t3.a.i.a.a.a.b.e.b(sb2)`的代码逻辑：

```
public final String b(String params) {
    x.q(params, "params");
    Charset charset ="UTF-8";
    //拿到字符编码 "UTF-8"
    x.h(charset, "Charsets.UTF_8");
    byte[] bytes = params.getBytes("UTF-8");
    //把传进来的明文进行getBytes，并且指定了字符集编码
    x.h(bytes, "(this as java.lang.String).getBytes(charset)");
    String str = d;
    //d是一个静态变量，d是根据一些判断，拿到一个字符串，一会再分析具体拿到的是哪个
```

```
    Charset charset2 = "UTF-8";
    x.h(charset2, "Charsets.UTF_8");
    if (str != null) {
        byte[] bytes2 = str.getBytes("UTF-8");
        //把拿到的d进行getBytes，并且指定了字符集编码"UTF-8"
        x.h(bytes2, "(this as java.lang.String).getBytes(charset)");
        String g = com.bilibili.commons.m.a.g(bytes, bytes2);
        //把明文和拿到的字符串d传到com.bilibili.commons.m.a.g()进行操作，返回值给了g
        x.h(g, "DigestUtils.sha256(param…yteArray(Charsets.UTF_8))");
        Locale locale = Locale.US;
        x.h(locale, "Locale.US");
        if (g != null) {
            //如果g不为空，就执行以下操作
            String lowerCase = g.toLowerCase(locale);
            //把g转为小写
            x.h(lowerCase, "(this as java.lang.String).toLowerCase(locale)");
            return lowerCase;
            //返回小写的字符串
        }
        throw new TypeCastException("null cannot be cast to non-null type
java.lang.String");
    }
    throw new TypeCastException("null cannot be cast to non-null type
java.lang.String");
}
```

这里边核心的逻辑是：

```
byte[] bytes = params.getBytes("UTF-8");
String str = d;
String g = com.bilibili.commons.m.a.g(bytes, bytes2);
String lowerCase = g.toLowerCase(locale);
```

我们要去确定str是什么，也就是d的值是什么，这样才知道传进去的是什么

```
private static final String d;
    static {
        String str3 = (String) a.C1483a.a(ConfigManager.Companion.b(),
"videodetail.report_click_salt", null, 2, null);
        if (str3 == null) {
            str3 = "9cafa6466a028bfb";
        }
        d = str3;
    }
```

d是通过这个静态代码块进行赋值的，我们直接去hook一下这个d，看看在打开视频的场景中，它的值是什么：

```
   . . . .
   . . . .   Connected to Pixel (id=FA7270300988)
d is ==> 9cafa6466a028bfb
[Pixel::哔哩哔哩 ]->
```

它的值就是`9cafa6466a028bfb`

然后调用了`com.bilibili.commons.m.a.g(bytes, bytes2)`

传进去的是明文和`9cafa6466a028bfb`字符串

去看看com.bilibili.commons.m.a.g(bytes, bytes2)里边的逻辑：

```
public static String g(byte[] bArr, byte[] bArr2) {
    try {
        MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");
        //拿到sha-256的实例
        messageDigest.reset();
        //清除一下空间
        messageDigest.update(bArr);
        //把第一个参数压进去
        if (bArr2 != null) {
            messageDigest.update(bArr2);
            //再压进第二个参数
        }
        return g.H(messageDigest.digest());
        //对加密结果再进行 g.H()
    } catch (NoSuchAlgorithmException e) {
        throw new AssertionError(e);
    }
}
//g.H()的逻辑：
public static String H(byte[] bArr) {
     StringBuilder sb = new StringBuilder();
     for (byte b2 : bArr) {
         int i = b2 & 255;
         if (i < 16) {
             sb.append('0');
         }
         sb.append(Integer.toHexString(i));
     }
     return sb.toString();
 }
//这里就是进行hex编码的
```

总结来说就是进行了一个sha256加密，其中加了盐，盐就是：`9cafa6466a028bfb`

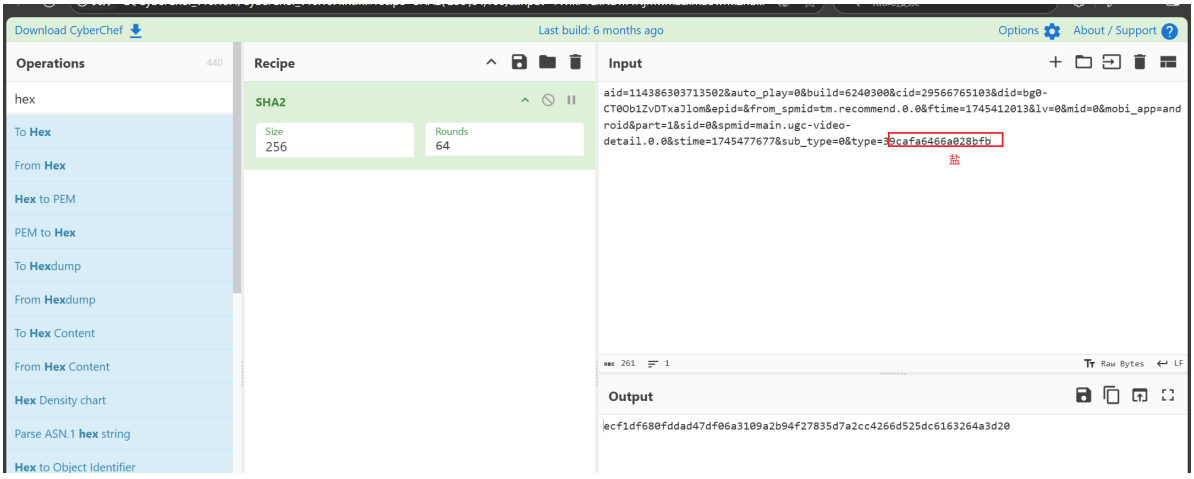
所以sign就是对其他的参数拼接完后再进行sha256加密，不过加密的时候加入了盐：`9cafa6466a028bfb`

其中明文拼接是按如下顺序，之前是put进一个TreeMap的，这里没有重写compare方法所以会按照key进行排序：

```
aid=114272302600996          &
auto_play=0                  &
build=6240300                &
cid=29210513232              &
did=bg0-CTOOb1ZvDTxaJlom     &
epid=                        &
from_spmid=tm.recommend.0.0  &
ftime=1745369985             &
lv=0                         &
mid=0                        &
mobi_app=android             &
part=1                       &
sid=0                        &
spmid=main.ugc-video-detail.0.0 &
stime=1745393699             &
sub_type=0                   &
type=3盐
```

我们去拿验证一下是不是：

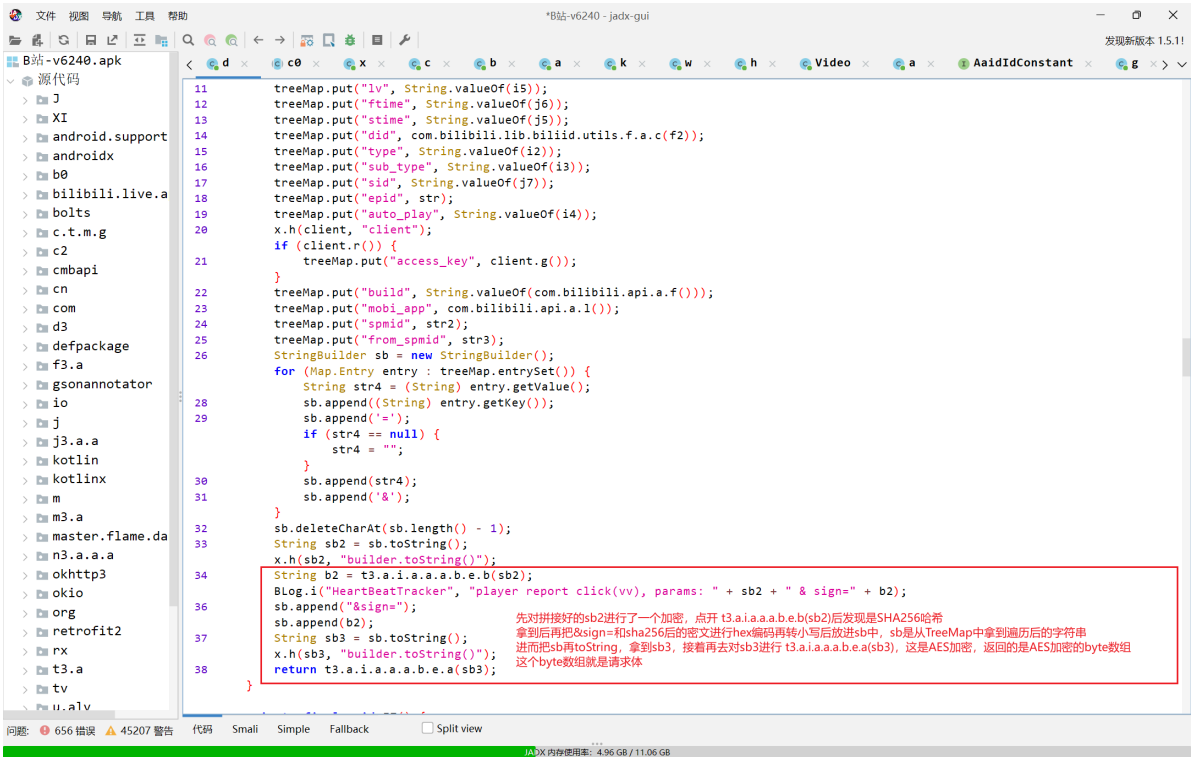
```
b.b is called: params=

aid=114386303713502&auto_play=0&build=6240300&cid=29566765103&did=bg0-
CT0Ob1ZvDTxaJlom&epid=&from_spmid=tm.recommend.0.0&ftime=1745412013&lv=0&mid=0&m
obi_app=android&part=1&sid=0&spmid=main.ugc-video-
detail.0.0&stime=1745477677&sub_type=0&type=3



b.b result=ecf1df680fddad47df06a3109a2b94f27835d7a2cc4266d525dc6163264a3d20
```



没得问题，sign解决了

请求体最终

接下来就是最终的请求体了，先看生成逻辑：



sb3是加入sign之后的明文，然后调用`t3.a.i.a.a.a.b.e.a(sb3)`后就return了，所以`t3.a.i.a.a.a.b.e.a(sb3)`是我们的关键代码：

先去hook一下，拿个参数和返回值：

```
b.a is called: body=

aid=114379760600102&auto_play=0&build=6240300&cid=29546712369&did=bg0-
CTOOb1ZvDTxaJlom&epid=&from_spmid=tm.recommend.0.0&ftime=1745412013&lv=0&mid=0&m
obi_app=android&part=1&sid=0&spmid=main.ugc-video-
detail.0.0&stime=1745482090&sub_type=0&type=3&sign=0c753d3f889bca410dc72d7edd391
b114871ae555aceb670654c3983679ef21e


b.a result=
046b0e451a52c511a53233a258cb1a2b468abd3f736977c963a6a3fa0076e6d2663f5e1d92b9578a
8ff5170598cf3be6ed46b08142602c8568d7b85ee3e7d7ed98b919efb2da93d3ea6fcca6628d16e4
1dcc5b7f07c8890e9994a5de63d5e0828a91f4b7ba9b4c9950e81e87b8f6970b170348993de8f52d
2a3e0a89ee9e8fbefd16742d191ebe5a599964ffe360fcf847accad62ef6b25ad4f21a8a811f1502
88d4d5d4c32d1bc578226baa2bf4a0763b21aaa6747fb65a80f3a67f733590ba251f3ddfefccbe7a
ff2d6844ab8fa051d708cf1002a24944b1463d84e18c7cd845bd02f5eb938312dcfa70263c227166
cbd781b85b51893cdd04eb2c713fb6beda6405ab226cc060c3914b6b1cba1cf641fe43541110ab4e
3967608d2588e34f049c5eab3ecd1f0d2c0191b62db312e4a13e789ca0c145826c9fd80012f86153
```

`t3.a.i.a.a.a.b.e.a(sb3)`:

```
public final byte[] a(String body) {
    x.q(body, "body");
    try {
        String str = b;
        //首先拿了个b，这个和sha256的盐在同一个地方
        Charset charset = "UTF-8";
        x.h(charset, "Charsets.UTF_8");
        if (str != null) {
```

```
            byte[] bytes = str.getBytes("UTF-8");
            //对刚刚拿的b进行了getBytes
            x.h(bytes, "(this as java.lang.String).getBytes(charset)");
            SecretKeySpec secretKeySpec = new SecretKeySpec(bytes, "AES");
            //把刚刚拿的b做了AES的key
            String str2 = f22911c;
            //又拿了f22911c,它和刚刚的key，sha256的盐是在一起的
            Charset charset2 = "UTF-8";
            x.h(charset2, "Charsets.UTF_8");
            if (str2 != null) {
                byte[] bytes2 = str2.getBytes("UTF-8");
                //对刚刚取的字符串进行了getBytes
                x.h(bytes2, "(this as java.lang.String).getBytes(charset)");
                IvParameterSpec ivParameterSpec = new IvParameterSpec(bytes2);
                //把这个字符串给了AES的iv
                Charset charset3 = "UTF-8";
                x.h(charset3, "Charsets.UTF_8");
                byte[] bytes3 = body.getBytes("UTF-8");
                //对传进来的明文进行了getBytes
                x.h(bytes3, "(this as java.lang.String).getBytes(charset)");
                byte[] i = com.bilibili.droid.g0.a.i(secretKeySpec,
ivParameterSpec, bytes3);
                //调用了i方法，传过去了AEs的key，iv，以及明文，返回值给了i
                x.h(i, "AES.encryptToBytes(Secre…yteArray(Charsets.UTF_8))");
                return i;
                //把i返回去了
            }
            throw new TypeCastException("null cannot be cast to non-null type
java.lang.String");
        }
        throw new TypeCastException("null cannot be cast to non-null type
java.lang.String");
    } catch (Exception e2) {
        BLog.e(a, e2);
        Charset charset4 = com.bilibili.commons.c.b;
        x.h(charset4, "Charsets.UTF_8");
        byte[] bytes4 = body.getBytes(charset4);
        x.h(bytes4, "(this as java.lang.String).getBytes(charset)");
        return bytes4;
    }
}
```

这个里边的核心逻辑如下：

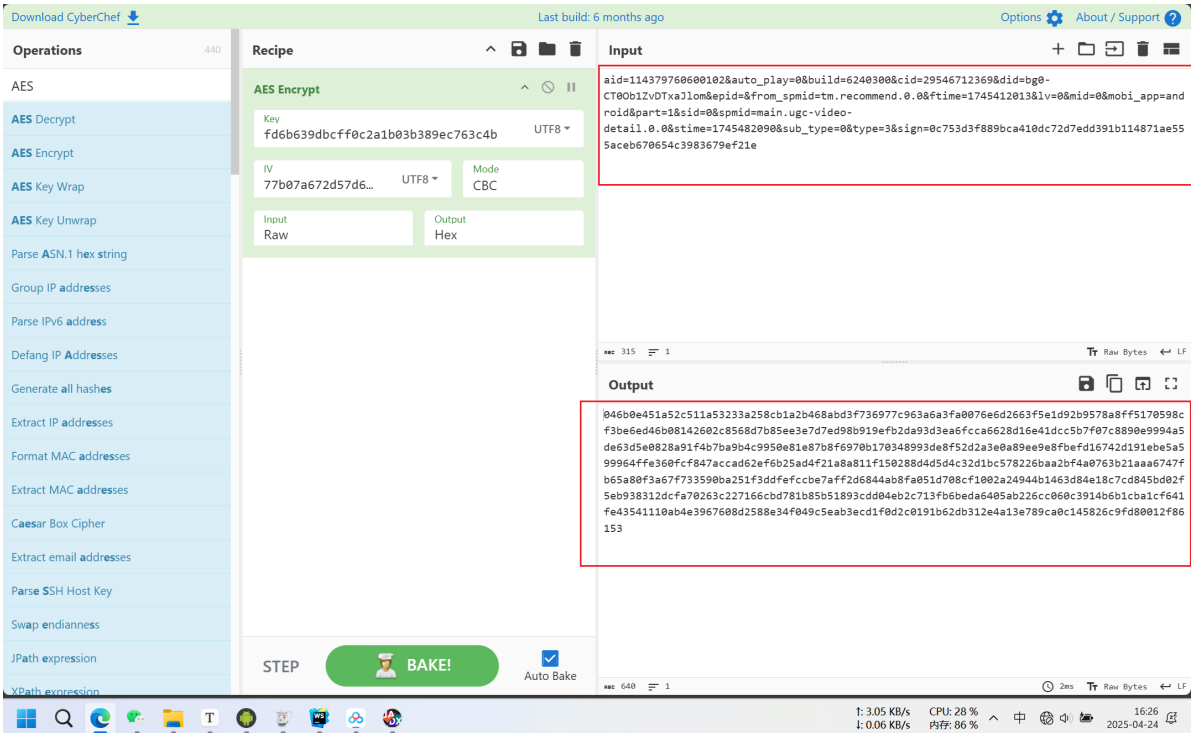
```
public final byte[] a(String body) {
     String str = b;
    //拿到key，经hook确认为："fd6b639dbcff0c2a1b03b389ec763c4b"
     if (str != null) {
         byte[] bytes = str.getBytes("UTF-8");
         SecretKeySpec secretKeySpec = new SecretKeySpec(bytes, "AES");
         //创建密钥
         String str2 = f22911c;
         //拿到iv，经hook确认为:"77b07a672d57d64c"
         if (str2 != null) {
             byte[] bytes2 = str2.getBytes("UTF-8");
```

```
            IvParameterSpec ivParameterSpec = new IvParameterSpec(bytes2);
            //创建iv
            byte[] bytes3 = body.getBytes("UTF-8");
            byte[] i = com.bilibili.droid.g0.a.i(secretKeySpec,
ivParameterSpec, bytes3);
            //调用com.bilibili.droid.g0.a.i加密
            return i;
        }
    }
}

com.bilibili.droid.g0.a.i(secretKeySpec, ivParameterSpec, bytes3)代码逻辑:
public static byte[] i(SecretKey secretKey, IvParameterSpec ivParameterSpec,
byte[] bArr) throws GeneralSecurityException {
        try {
            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
            cipher.init(1, secretKey, ivParameterSpec);
            return cipher.doFinal(bArr);
        } catch (NoSuchAlgorithmException | NoSuchPaddingException e) {
            throw new AssertionError(e);
        }
    }
这里就是进行了"AES/CBC/PKCS5Padding"加密
```

到此请求体的加密就完成了:

```
加密方式："AES/CBC/PKCS5Padding"
key：fd6b639dbcff0c2a1b03b389ec763c4b
iv：77b07a672d57d64c
明文：aid=114379760600102&auto_play=0&build=6240300&cid=29546712369&did=bg0-
CT0Ob1ZvDTxaJlom&epid=&from_spmid=tm.recommend.0.0&ftime=1745412013&lv=0&mid=0&m
obi_app=android&part=1&sid=0&spmid=main.ugc-video-
detail.0.0&stime=1745482090&sub_type=0&type=3&sign=0c753d3f889bca410dc72d7edd391
b114871ae555aceb670654c3983679ef21e
结果：
046b0e451a52c511a53233a258cb1a2b468abd3f736977c963a6a3fa0076e6d2663f5e1d92b9578a
8ff5170598cf3be6ed46b08142602c8568d7b85ee3e7d7ed98b919efb2da93d3ea6fcca6628d16e4
1dcc5b7f07c8890e9994a5de63d5e0828a91f4b7ba9b4c9950e81e87b8f6970b170348993de8f52d
2a3e0a89ee9e8fbefd16742d191ebe5a599964ffe360fcf847accad62ef6b25ad4f21a8a811f1502
88d4d5d4c32d1bc578226baa2bf4a0763b21aaa6747fb65a80f3a67f733590ba251f3ddfefccbe7a
ff2d6844ab8fa051d708cf1002a24944b1463d84e18c7cd845bd02f5eb938312dcfa70263c227166
cbd781b85b51893cdd04eb2c713fb6beda6405ab226cc060c3914b6b1cba1cf641fe43541110ab4e
3967608d2588e34f049c5eab3ecd1f0d2c0191b62db312e4a13e789ca0c145826c9fd80012f86153
```

我们去复现一下:



没得问题，到这里请求体就ok了。

请求头

请求网址：https://api.bilibili.com/x/report/click/android2
请求方式：POST
请求头：

```
        content-length          320
        buvid                   XY2CAAD2CD8168403CCB095E408A467773352
        device-id               bg0-CT0Ob1ZvDTxaJlom
        fp_local
efbaa891c759dbcacebbf89bb3517b7f20250423085841a27a01b53814242b3b
        fp_remote
efbaa891c759dbcacebbf89bb3517b7f20241207150740ab3e564147129a8976
        session_id              fafa8e36
        env                     prod
        app-key                 android
        user-agent              Mozilla/5.0 BiliDroid/6.24.0
(bbcallen@gmail.com) os/android model/Pixel mobi_app/android build/6240300
channel/xxl_gdt_wm_253 innerVer/6240300 osVer/10 network/2
        bili-bridge-engine      cronet
        content-type            application/octet-stream
        accept-encoding         gzip, deflate, br
```

请求头只有`buvid`、`device-id`、`fp_local`、`fp_remote`、`session_id`是需要逆向的，并且这个`device-id`就是请求体中的`did`

我们搜索链接的时候发现，在发送这个请求的时候，有设置自定义注解式拦截器：

```
@POST("/x/report/click/android2")
@RequestInterceptor(com.bilibili.okretro.f.c.class)
com.bilibili.okretro.d.a<String> reportClick(@Body c0 c0Var);
```

```
public class c extends a {
    @Override
    public final b0 a(b0 b0Var) {
        b0.a h = b0Var.h();
        e(h);
        return h.b();
    }
}

//这里有混淆，正常public class c extends a中的a应该是interceptor
//然后a方法是重写的intercept(Chain chain)
//h是拿到了对应的请求
//然后调用e(h)去添加拦截器要添加的参数
//最终调用h.b()返回，返回的就是chain对象
```

我们去看看这个e(h)方法：里边是设置请求头的

```
public void e(b0.a aVar) {
    String a2 = com.bilibili.api.d.a();
    if (!TextUtils.isEmpty(a2)) {
        aVar.h("Display-ID", a2);
    }
    String a3 = com.bilibili.api.c.a();
    if (!TextUtils.isEmpty(a3)) {
        aVar.h("Buvid", a3);
    }
    String c3 = com.bilibili.api.a.c();
    if (!TextUtils.isEmpty(c3)) {
        aVar.h("User-Agent", c3);
    }
    String a4 = com.bilibili.api.e.a();
    if (!TextUtils.isEmpty(a4)) {
        aVar.h("Device-ID", a4);
    }
    String j2 = com.bilibili.api.a.j();
    if (!TextUtils.isEmpty(j2)) {
        aVar.h("fp_local", j2);
    }
    String k = com.bilibili.api.a.k();
    if (!TextUtils.isEmpty(k)) {
        aVar.h("fp_remote", k);
    }
    String m2 = com.bilibili.api.a.m();
    if (TextUtils.isEmpty(m2)) {
        return;
    }
    aVar.h("session_id", m2);
}
```

buvid

我们去看看buvid的生成逻辑：

```
String a3 = com.bilibili.api.c.a();
if (!TextUtils.isEmpty(a3)) {
    aVar.h("Buvid", a3);
}
```

buvid的值是通过`String a3 = com.bilibili.api.c.a();`得到的：

```
//com.bilibili.api.c.a():
public static String a() {
     return a;
 }
public static void b(String str) {
    a = str;
}
```

调用`com.bilibili.api.c.a()`返回的是变量a的值，变量a的值是通过调用b方法去赋值的，所以我们去hook一下b方法，看看调用栈：

```
[Pixel::tv.danmaku.bili ]->

c.b is called: str=XY2CAAD2CD8168403CCB095E408A467773352
java.lang.Throwable
        at com.bilibili.api.c.b(Native Method)
        at c2.f.b0.c.a.d.e(BL:1)
        at c2.f.b0.c.a.d.a(BL:11)
        at tv.danmaku.bili.utils.x.a(BL:14)
        at tv.danmaku.bili.proc.y.f(BL:1)
        at tv.danmaku.bili.proc.c.run(Unknown Source:2)
```

去看看`c2.f.b0.c.a.d.e(BL:1)`在调用b方法时传入的参数是怎么来的：

```
//c2.f.b0.c.a.d.e:

private void e(@Nullable String str) {
        com.bilibili.api.c.b(str);
    }
//参数是调用e方法的时候传进来的，继续往上追
//c2.f.b0.c.a.d.a:
public String a() {
    String str;
    synchronized (d.class) {
        str = TextUtils.isEmpty(this.a) ? "" : this.a;
    }
    if (TextUtils.isEmpty(str)) {
        com.bilibili.droid.thread.d.h(2, new Runnable() {
            @Override
            public final void run() {
                d.this.d();
            }
```

```
        });
        synchronized (d.class) {
            str = this.a;
        }
        e(str);
    }
    return str;
}
```

到这里我们看到调用e方法的时候传入的是str，str的值又是通过this.a给赋值的，所以我们要去看c2.f.b0.c.a.d.a中的 this.a

```
public final class d {
    private String a;

    public /* synthetic */ void d() {
        String d = MiscHelperKt.d(e.k().b());
        if (!TextUtils.isEmpty(d)) {
            synchronized (d.class) {
                this.a = d;
            }
            return;
        }
        String d2 = MiscHelperKt.d(e.k().c());
        if (!TextUtils.isEmpty(d2)) {
            synchronized (d.class) {
                this.a = d2;
            }
            return;
        }
        String upperCase = c2.f.b0.c.b.a.c.a().toUpperCase();
        synchronized (d.class) {
            this.a = upperCase;
            if (!TextUtils.isEmpty(upperCase)) {
                e.k().u(this.a);
            }
        }
    }

}
```

this.a是在这个d方法中赋值的，我们分析一下d方法

```
//大体来看，分为了三种请求
//1.若 String d = MiscHelperKt.d(e.k().b());能拿到非空的值，那么就把d的值赋值给this.a
//2.若 d为空且String d2 = MiscHelperKt.d(e.k().c());能拿到非空的值，那么就把d2的值赋值给this.a
//3.若 d、d2都为空，且String upperCase = c2.f.b0.c.b.a.c.a().toUpperCase();能拿到非空的值，就把upperCase的值赋值给this.a
```

buvid值的第一种生成方法

先看`String d = MiscHelperKt.d(e.k().b());`

```
//MiscHelperKt.d():
public static final String d(String str) {
      int length;
      if (str == null || (length = str.length()) == 0 || length > 64) {
          return null;
      }
      for (int i = 0; i < length; i++) {
          char charAt = str.charAt(i);
          if (('A' > charAt || charAt > 'Z') && ('a' > charAt || charAt > 'z')
&& !('0' <= charAt && charAt <= '9') || charAt == '-' || charAt == '_'))) {
              return null;
          }
      }
      return str;
  }
//这个函数主要是在做检测,检测传进来的参数
//首先检测是否为null，长度是否为0，长度是否大于64，若为其中一种情况，则直接返回null
//然后去对逐个字符进行检测，检测是否为大写英文、小写英文、是否是0-9、是否包括-或者_ 如果不是上述这几种就返回null
//所以这个函数是进行检测的，关键的地方是传进来的参数
//传进来的参数是：e.k().b()
@Nullable
public String b() {
    String t = l().t();
    if (!TextUtils.isEmpty(t)) {
        com.bilibili.lib.biliid.internal.storage.external.d.u(t);
        return t;
    }
    String b2 = com.bilibili.lib.biliid.internal.storage.external.d.b();
    if (!TextUtils.isEmpty(b2)) {
        l().o(b2);
    }
    return b2;
}
//在b方法中，主要是做了两件事
//1. String t = l().t(); 若t不为空，返回t
//2. String b2 = com.bilibili.lib.biliid.internal.storage.external.d.b();若b2不为空，返回b2

//先看String t = l().t();
public String t() {
        return c().getString("buvid", "");
    }
//从SharedPreferences（XML存储）中取buvid的值

//再看String b2 = com.bilibili.lib.biliid.internal.storage.external.d.b();
public static String b() {
        a.readLock().lock();
        try {
            return f13195c.buvid;
        } finally {
```

```
            a.readLock().unlock();
        }
    }
//从内存中取值
```

总的来说buvid的第一种取法就是：XML中取或者内存中取

buvid值的第二种生成方法

再看第二种方法：`String d2 = MiscHelperKt.d(e.k().c());`

```
//这里也是先进行一下检测，看看有没有非法字符
//所以这里的buvid 是e.k().c()
//e.k().c()：
@Nullable
public String c() {
    String u2 = l().u();
    if (!TextUtils.isEmpty(u2)) {
        com.bilibili.lib.biliid.internal.storage.external.d.v(u2);
        return u2;
    }
    String c3 = com.bilibili.lib.biliid.internal.storage.external.d.c();

    if (!TextUtils.isEmpty(c3)) {
        l().P(c3);
    }
    return c3;
}

//同样是两种方式得到的：
//String u2 = l().u();
//String c3 = com.bilibili.lib.biliid.internal.storage.external.d.c();

//先看第一种：String u2 = l().u();
private static final String h = "buvid2";
public String u() {
        return c().getString(h, "");
}
//是从xml中取buvid2的值，然后放进去


//接着看第二种：String c3 =
com.bilibili.lib.biliid.internal.storage.external.d.c();
@Nullable
public static String c() {
        a.readLock().lock();
        try {
            return f13195c.buvid2;
        } finally {
            a.readLock().unlock();
        }
 }
//是去内存中取buvid2的值
```

总的来说，第二种和第一种的方式一致，都是先去xml中搜索，然后再去内存中搜索,只不过这这里是去找buvid2的值

buvid值的第三种生成方法

再看`String upperCase = c2.f.b0.c.b.a.c.a().toUpperCase();`

```
//String upperCase = c2.f.b0.c.b.a.c.a().toUpperCase();
//先通过c2.f.b0.c.b.a.c.a()拿到一个字符串，然后再转大写
@NonNull
public static String a() {
    Application f = BiliContext.f();
    String b2 = com.bilibili.lib.biliid.utils.f.c.b(f);
    if (!TextUtils.isEmpty(b2) && com.bilibili.lib.biliid.utils.f.d.b(b2)) {
        String d = com.bilibili.commons.m.a.d(b2);
        return "XZ" + e(d) + d;
    }
    String j2 = com.bilibili.lib.biliid.utils.f.a.j(f);
    if (!TextUtils.isEmpty(j2) && com.bilibili.lib.biliid.utils.f.d.c(j2)) {
        String d2 = com.bilibili.commons.m.a.d(j2);
        return "XY" + e(d2) + d2;
    }
    String a2 = com.bilibili.lib.biliid.utils.f.c.a(f);
    if (!TextUtils.isEmpty(a2) && com.bilibili.lib.biliid.utils.f.d.a(a2)) {
        String d4 = com.bilibili.commons.m.a.d(a2);
        return "XX" + e(d4) + d4;
    }
    String replace = e.k().i().replace(com.bilibili.base.util.d.f, "");
    return "XW" + e(replace) + replace;
}
//根据我们抓包得到的buvid来看，这里是返回的XY开头的，所以咱们直接去分析XY打头的这个
String j2 = com.bilibili.lib.biliid.utils.f.a.j(f);
if (!TextUtils.isEmpty(j2) && com.bilibili.lib.biliid.utils.f.d.c(j2)) {
    String d2 = com.bilibili.commons.m.a.d(j2);
    return "XY" + e(d2) + d2;
}
```

```
String j2 = com.bilibili.lib.biliid.utils.f.a.j(f);
//取了某些信息，返回来一个字符串
if (!TextUtils.isEmpty(j2) && com.bilibili.lib.biliid.utils.f.d.c(j2)) {
    String d2 = com.bilibili.commons.m.a.d(j2);
    //对取的这个字符串进行md5加密
    return "XY" + e(d2) + d2;
    //返回XY+(MD5hex编码的第2、12、22位，这三个字符拼成的字符串)+MD5后的结果
}

//现在大体逻辑捋清楚了，接下来的任务就是去把这个字符串的生成过程去捋清
//com.bilibili.lib.biliid.utils.f.a.j(f):
//最终是调用的下边这个函数:
@Nullable
public static String c(@NonNull Context context) {
    WifiManager wifiManager;
    WifiInfo connectionInfo;
    try {
```

```
        if (f12933c != null) {
            return f12933c;
        }
        String a2 = z.a("wlan.lge.wifimac");
        f12933c = a2;
        if (a2.length() > 0) {
            return f12933c;
        }
        if (context != null && (wifiManager = (WifiManager)
context.getApplicationContext().getSystemService("wifi")) != null &&
(connectionInfo = wifiManager.getConnectionInfo()) != null) {
            String macAddress = connectionInfo.getMacAddress();
            f12933c = macAddress;
            if (!DeviceUtil.FAKE_MAC.equals(macAddress)) {
                return f12933c;
            }
        }
        String a3 = z.a("wifi.interface");
        if (TextUtils.isEmpty(a3)) {
            a3 = "wlan0";
        }
        String t = com.bilibili.commons.k.a.t(new File("/sys/class/net/" + a3 +
"/address"));
        f12933c = t;
        if (TextUtils.isEmpty(t)) {
            try {
                Enumeration<NetworkInterface> networkInterfaces =
NetworkInterface.getNetworkInterfaces();
                if (networkInterfaces == null) {
                    return f12933c;
                }
                while (networkInterfaces.hasMoreElements()) {
                    NetworkInterface nextElement =
networkInterfaces.nextElement();
                    if (nextElement.getName().equalsIgnoreCase(a3)) {
                        byte[] hardwareAddress =
nextElement.getHardwareAddress();
                        if (hardwareAddress == null) {
                            break;
                        }
                        StringBuilder sb = new StringBuilder();
                        int length = hardwareAddress.length;
                        for (int i = 0; i < length; i++) {
                            sb.append(String.format("%02X:",
Byte.valueOf(hardwareAddress[i])));
                        }
                        if (sb.length() > 0) {
                            sb.deleteCharAt(sb.length() - 1);
                        }
                        f12933c = sb.toString();
                    }
                }
            } catch (SocketException e) {
                e.printStackTrace();
            }
```

```
        }
        return f12933c;
    } catch (Exception unused) {
        return f12933c;
    }
}
//这个函数就是获取手机的mac地址
```

总的来说就是先取mac地址，进行MD5加密，加密后按照

"XY" +(MD5hex编码的第2、12、22位，这三个字符拼成的字符串)+MD5后的结果来进行拼接

app是采取了这种mac地址的方式进行操作的

其他三种是分别获取了其他信息，然后进行MD5后再按照"XZ/X/w" +(MD5hex编码的第2、12、22位，这三个字符拼成的字符串)+MD5后的结果来进行拼接拼接

device-id

接着去看device-id：

```
String a4 = com.bilibili.api.e.a();
if (!TextUtils.isEmpty(a4)) {
    aVar.h("Device-ID", a4);
}
//a4就是device-id，a4通过com.bilibili.api.e.a()得到的
//com.bilibili.api.e.a():
public class e {
    private static String a;
    public static String a() {
        return a;
    }
    public static void b(String str) {
        a = str;
    }
}

//a4就是通过调用类e中a方法得到的。a方法的返回值是类e中属性a的值，属性a是通过b方法进行赋值的，我们去hook一下b方法，看看调用栈
[Pixel::tv.danmaku.bili ]->

e.b is called: str=bg0-CTOOb1ZvDTxaJlom
java.lang.Throwable
        at com.bilibili.api.e.b(Native Method)
        at tv.danmaku.bili.report.j$a.run(BL:4)

//去看下tv.danmaku.bili.report.j$a.run方法
public void run() {
    if (!j.a) {
        com.bilibili.api.d.b(j.c(this.a));
    }
    com.bilibili.api.e.b(com.bilibili.lib.biliid.utils.f.a.c(this.a));
        }
//调用b方法对a进行赋值的时候传入的是：com.bilibili.lib.biliid.utils.f.a.c(this.a)
//这里this.a是一个context
//我们去看看com.bilibili.lib.biliid.utils.f.a.c()方法
```

```
public static String c(@Nullable Context context) {
    if (TextUtils.isEmpty(f13201c)) {
        if (context == null) {
            return "";
        }
        String f = c2.f.b0.c.a.e.k().f(context);
        f13201c = f;
        if (!TextUtils.isEmpty(f)) {
            return f13201c;
        }
        f13201c = g(context);
        c2.f.b0.c.a.e.k().x(f13201c, context);
        return f13201c;
    }
    return f13201c;
}
//首先判断f13201c是否为空，不为空直接返回，为空的话就去给f13201c赋值再返回
//我们先去看f13201c为空时，怎么去给f13201c赋值
//这里主要是有两种
//分别通过String f = c2.f.b0.c.a.e.k().f(context);或者f13201c = g(context);
//先看String f = c2.f.b0.c.a.e.k().f(context);
public String f(Context context) {
    SharedPreferences c3 = l().c();
    String string = c3.getString("persist.c.bl.did", null);
    if (!TextUtils.isEmpty(string)) {
        com.bilibili.lib.biliid.internal.storage.external.d.y(string);
        return string;
    }
    String f = com.bilibili.lib.biliid.internal.storage.external.d.f();
    if (!TextUtils.isEmpty(f)) {
        c3.edit().putString("persist.c.bl.did", f).apply();
    }
    return f;
    }
//这里就是两个方向，1、去xml中取；2、去内存中取
//再看另一个f13201c = g(context);
static String g(Context context) {
    String f = f(context);
    if (f.length() < 4) {
        f = Settings.Secure.getString(context.getContentResolver(),
"android_id") + "@" + g.g(Build.MODEL);
    }
    return b(f);
}
//进来后先通过f方法去拿到一个字符串，然后再判断长度是否小于4，若小于4的话会掉用其他方法重新生成一个，最终返回b(f)
//这里device-id的值不包含"@"，所以长度肯定不小于4
//所以我们先去看看String f = f(context);是怎么得到的字符串
public static String f(Context context) {
    StringBuffer stringBuffer = new StringBuffer();
    String j2 = j(context);
    if (j2 != null) {
        String lowerCase = j2.replaceAll("[^0-9A-Fa-f]", "").toLowerCase();
        if (k(lowerCase)) {
            stringBuffer.append(lowerCase);
```

```
        }
    }
    stringBuffer.append('|');
    String a2 = z.a("persist.service.bdroid.bdaddr");
    if (a2.length() > 0) {
        String lowerCase2 = a2.replaceAll("[^0-9A-Fa-f]", "").toLowerCase();
        if (k(lowerCase2)) {
            stringBuffer.append(lowerCase2);
        }
    }
    stringBuffer.append('|');
    String h = h();
    if (h != null) {
        stringBuffer.append(h.toLowerCase());
    }
    stringBuffer.append('|');
    String i = i();
    if (i != null) {
        stringBuffer.append(i.toLowerCase());
    }
    return stringBuffer.toString();
}
//这里总体来看是 j2|a2|h|i
//j2、a2、h、i分别是通过其他函数返回的字符串，分别去看下

//String j2 = j(context);
@Nullable
public static synchronized String j(@NonNull Context context) {
    String c3;
    synchronized (a.class) {
        c3 = com.bilibili.droid.g.c(context);
    }
    return c3;
}
//j2是mac地址

//String a2 = z.a("persist.service.bdroid.bdaddr");
private static final Method a = d(c("android.os.SystemProperties"));
@NonNull
public static String a(String str) {
    Method method = a;
    if (method != null) {
        try {
            Object invoke = method.invoke(null, str);
            return invoke == null ? "" :
com.bilibili.commons.g.K(invoke.toString());
        } catch (Exception unused) {
        }
    }
    return "";
}
//a2是蓝牙地址

// String h = h();
private static synchronized String h() {
```

```
    synchronized (a.class) {
        if (a != null) {
            return a;
        }
        String a2 = b.a();
        a = a2;
        return a2;
        }
    }
//h是设备序列号

//String i = i();
private static synchronized String i() {
    synchronized (a.class) {
        if (b != null) {
            return b;
        }
        String a2 = e.a();
        b = a2;
        return a2;
    }
}
//i是sn总线


//总的来说j2|a2|h|i就是：mac地址|蓝牙地址|设备序列号|sn总线
//所以f是mac地址|蓝牙地址|设备序列号|sn总线
//最终返回的是b(f)
//再去看看b方法
public static String b(String str) {
    byte[] bytes = str.getBytes();
    bytes[0] = (byte) (bytes[0] ^ ((byte) (bytes.length & 255)));
    for (int i = 1; i < bytes.length; i++) {
        bytes[i] = (byte) ((bytes[i - 1] ^ bytes[i]) & 255);
    }
    try {
        return new String(Base64.encode(bytes, 11));
    } catch (Exception unused) {
        return str;
    }
}
//这里相当于是自写了一个简单的混淆算法
//它首先进行了一系列的异或、位与等操作
//对操作好的数据进行base64编码，然后返回
//但是这里base64中的标致，它写的是11也就是Base64.URL_SAFE
//Base64.URL_SAFE情况下会将标准 Base64 字符 + 和 / 替换为 - 和 _，避免在 URL 或文件名中产生歧义
```

device-id总结

首先获取由mac地址|蓝牙地址|设备序列号|sn总线拼成的字符串，且用|分割
然后对这个字符串进行一些位与 异或操作
对操作好的字符串进行base64 URL_SAFE的编码
然后就得到的了device-id

fp_local

```
String j2 = com.bilibili.api.a.j();
if (!TextUtils.isEmpty(j2)) {
    aVar.h("fp_local", j2);
}

//去看看com.bilibili.api.a.j()方法：
public static String j() {
    a();
    return b.F();
}
//这里的b是一个接口，F是b接口中的方法
//我们要看这个b真实的赋值的是哪个类的对象

public class a {
 private static b b;
 public static void o(b bVar) {
        b = bVar;
    }
}
//这里是通过o方法给b赋值的，所以我们去看调用o方法时传进来的对象是哪个类的，那个类必然是实现了接口b，也实现了F方法
//我们去hook一下o，看看调用栈

[Pixel::tv.danmaku.bili ]->
    a.o is called: bVar="<instance: com.bilibili.api.a$b, $className:
tv.danmaku.bili.utils.p$a>"
java.lang.Throwable
        at com.bilibili.api.a.o(Native Method)
        at tv.danmaku.bili.utils.p.b(BL:1)

//我们去看看tv.danmaku.bili.utils.p.b调用o方法时传进去的对象
public static final void b(Application app, boolean z) {
        kotlin.jvm.internal.x.q(app, "app");
        com.bilibili.api.a.o(new a(app));
    }
//这里是new了一个a类的对象，我们去看看这个a类怎么实现的F方法
public static final class a implements a.b {
    @Override // com.bilibili.api.a.b
    public String F() {
        String a = c2.f.b0.c.a.c.a();
        kotlin.jvm.internal.x.h(a, "BiliIds.buvidLocal()");
        return a;
    }
}
//是通过 String a = c2.f.b0.c.a.c.a()生成一个字符串，然后进行判断是否为空，为空就报错，不为空就返回

// String a = c2.f.b0.c.a.c.a():
public static String a() {
    return Fingerprint.h.c();
}
//又调用了Fingerprint.h.c()
public final String c() {
```

```
        String str = "";
        if (k()) {
            ReentrantReadWriteLock.ReadLock r = e;
            x.h(r, "r");
            r.lock();
            try {
                if (a != null && (str = a) == null) {
                    x.Q("buvidLocal");
                }
            } finally {
                r.unlock();
            }
        }
        return str;
    }
}
//大致看了一眼 首先定义一个str，然后判断k()的返回值，若为真就执行里边的操作，为假就继续向下走然后返回str，抓到的数据不是空，这里k()肯定返回的真
//进入if之后，首先是获取读锁对象，加读锁
//然后进入try-catch，判断a不为null的话，就把a赋值给str，然后打个日志，然后释放读锁，返回str
//这里的重点就是a，a就是我们最终的fp_local
//去看下a
public final class Fingerprint {
    public static String a;//a就是fp_local
    private final void e() {
    String d2 = k.d();
    if ((d2 == null || (s.x1(d2))) ||
!com.bilibili.lib.biliid.internal.fingerprint.a.a.d(k)) {
         if (g) {
             BLog.dfmt("biliid.fingerprint", "No buvidLocal in env, calculate
right now.", new Object[0]);
             b = DataKt.a();
             d c3 = d.c();
             x.h(c3, "BuvidHelper.getInstance()");
             String a2 = c3.a();
             x.h(a2, "buvidLegacy");
             com.bilibili.lib.biliid.internal.fingerprint.b.a aVar = b;
             if (aVar == null) {
                 x.Q("data");
             }
             d2 = com.bilibili.lib.biliid.internal.fingerprint.a.a.a(a2, aVar);
             k.v(d2);
             String encode = Uri.encode(Build.BRAND);
             String encode2 = Uri.encode(Build.MODEL);
             k.s(PersistEnv.KEY_PUB_BRAND, encode);
             k.s(PersistEnv.KEY_PUB_MODEL, encode2);
            } else {
             BLog.d("biliid.fingerprint", "Do not calculate buvidLocal on other
process.");
             d2 = "";
            }
        }
        ReentrantReadWriteLock.WriteLock writeLock = f;
        x.h(writeLock, b.w);
        writeLock.lock();
```

```
        if (d2 == null) {
            try {
                x.K();
            } catch (Throwable th) {
                writeLock.unlock();
                throw th;
            }
        }
        a = d2;
    }
}
//大概例流程是这样
//1、首先通过String d2 = k.d();去获取d2的值，如果获取到了就赋值给a
//2、若k.d()获取的值为null，就通过d2 =
com.bilibili.lib.biliid.internal.fingerprint.a.a.a(a2, aVar);去获取，获取后再赋值给a
//3、如果通过第二种方式也没获取到，就会抛出错误

//先看第一种获取方式：String d2 = k.d();
public String d() {
    String w = l().w();
    if (!TextUtils.isEmpty(w)) {
        d.w(w);
        return w;
    }
    String d = d.d();
    if (!TextUtils.isEmpty(d)) {
        l().R(d);
    }
    return d;
}
//如果String w = l().w();不为空就返回w
//如果String d = d.d();不为空就返回d

//先看String w = l().w();
public String w() {
    return c().getString(r, "");
}
//是去xml中取

//再看String d = d.d();
public static String d() {
    a.readLock().lock();
    try {
        return f13235c.buvidLocal;
    } finally {
        a.readLock().unlock();
    }
}
//去内存中取
//总的来说String d2 = k.d();就是去xml中或者内存中取

//先看第二种获取方式：d2 = com.bilibili.lib.biliid.internal.fingerprint.a.a.a(a2,
aVar);
public static final String a(String str,
com.bilibili.lib.biliid.internal.fingerprint.b.a aVar) {
```

```
    x.q(str, "buvidLegacy");
    x.q(aVar, "data");
    String str2 = MiscHelperKt.a(f(str, aVar)) + h() + MiscHelperKt.a(g());
    return str2 + b(str2);
}
//首先判断buvidLegacy和data的值为不为空
//然后通过String str2 = MiscHelperKt.a(f(str, aVar)) + h() + MiscHelperKt.a(g());
获取一个字符串str2
//然后返回 str+b(str2)

//我们先分析str2的生成是由三部分组成的：MiscHelperKt.a(f(str, aVar)) + h() +
MiscHelperKt.a(g())
//先去看MiscHelperKt.a(),我们发现有两部分都是要通过MiscHelperKt.a()所以我们先去看看这个方
法
public static final String a(byte[] bArr) {
    x.q(bArr, "$this$asHex");
    return ArraysKt___ArraysKt.Oe(bArr, "", null, null, 0, null,
MiscHelperKt$asHex$1.INSTANCE, 30, null);
}
//里边又调用了ArraysKt___ArraysKt.Oe(bArr, "", null, null, 0, null,
MiscHelperKt$asHex$1.INSTANCE, 30, null);
//看看ArraysKt___ArraysKt.Oe(bArr, "", null, null, 0, null,
MiscHelperKt$asHex$1.INSTANCE, 30, null)：
public static /* synthetic */ String Oe(byte[] bArr, CharSequence charSequence,
CharSequence charSequence2, CharSequence charSequence3, int i2, CharSequence
charSequence4, kotlin.jvm.c.l lVar, int i3, Object obj) {
    if ((i3 & 1) != 0) {
        charSequence = ", ";
    }
    CharSequence charSequence5 = "";
    CharSequence charSequence6 = (i3 & 2) != 0 ? charSequence5 : charSequence2;
    if ((i3 & 4) == 0) {
        charSequence5 = charSequence3;
    }
    int i4 = (i3 & 8) != 0 ? -1 : i2;
    if ((i3 & 16) != 0) {
        charSequence4 = "...";
    }
    if ((i3 & 32) != 0) {
        lVar = null;
    }
    return Fe(bArr, charSequence, charSequence6, charSequence5, i4,
charSequence4, lVar);
}
//这个里边又调用了Fe(bArr, charSequence, charSequence6, charSequence5, i4,
charSequence4, lVar)
//进来后是根据i3和i2的的值计算出charSequence、charSequence6、charSequence5、i4、
charSequence4、lVar
//但是i3和i2的值是在a方法中调用Oe方法时就写死的，i3是30,i2是0
//所以在调用Fe方法时传进去的参数除了bArr都是固定的
//所以我们可以hook一下Fe方法，拿到这几个固定的参数
charSequence  is ==>
charSequence2 is ==>
charSequence3 is ==>
i2 is            ==> -1
```

```
charSequence4 is ==> ...
lVar is ==> [object Object]

//看看Fe(bArr, charSequence, charSequence6, charSequence5, i4, charSequence4,
lVar)
public static final String Fe(byte[] bArr, CharSequence charSequence,
CharSequence charSequence2, CharSequence charSequence3, int i2, CharSequence
charSequence4, kotlin.jvm.c.l<? super Byte, ? extends CharSequence> lVar) {
    String sb = ((StringBuilder) ne(bArr, new StringBuilder(), charSequence,
charSequence2, charSequence3, i2, charSequence4, lVar)).toString();
    return sb;
}

//在这里边又调用了了ne(bArr, new StringBuilder(), charSequence, charSequence2,
charSequence3, i2, charSequence4, lVar))
//去看看ne
public static final <A extends Appendable> A ne(byte[] bArr, A a2, CharSequence
charSequence, CharSequence charSequence2, CharSequence charSequence3, int i2,
CharSequence charSequence4, kotlin.jvm.c.l<? super Byte, ? extends CharSequence>
lVar) {
    a2.append(charSequence2);
    int i3 = 0;
    for (byte b2 : bArr) {
        i3++;
        if (i3 > 1) {
            a2.append(charSequence);
        }
        if (i2 >= 0 && i3 > i2) {
            break;
        }
        if (lVar != null) {
            a2.append((CharSequence) lVar.invoke(Byte.valueOf(b2)));
        } else {
            a2.append(String.valueOf((int) b2));
        }
    }
    if (i2 >= 0 && i3 > i2) {
        a2.append(charSequence4);
    }
    a2.append(charSequence3);
    return a2;
}
//去hook一下ne方法，看看入参和返回值
bArr is ==> -17,-70,-88,-111,-57,89,-37,-54,-50,-69,-8,-101,-77,81,123,127
a2 is ==> [object Object]
charSequence is ==>
charSequence2 is ==>
charSequence3 is ==>
i2 is ==> -1
charSequence4 is ==> ...
lVar is ==> [object Object]
result is ==> efbaa891c759dbcacebbf89bb3517b7f
//然后我们去分析一下ne方法，这就是我们的MiscHelperKt.a(byte[] bArr)的核心
//这里的核心就是把字节数组中的每一位元素转为十六进制，在这个场景下是转为16进制后直接拼一起的，有
的场景会用冒号分隔，用什么分割取决于charSequence
```

```
//那么好，MiscHelperKt.a(byte[] bArr)的核心就是把字节数组的每一位进行hex编码

//回过来说，我们的fp_local就是通过String str2 = MiscHelperKt.a(f(str, aVar)) + h() +
MiscHelperKt.a(g());获取一个字符串str2，然后返回 str2+b(str2)
//str又是由三部分组成的MiscHelperKt.a(f(str, aVar)) + h() + MiscHelperKt.a(g());
//现在搞清楚MiscHelperKt.a()传的参数以及h()是什么，那么就搞清楚fp_local了

//我们看看第一部分MiscHelperKt.a(f(str, aVar))的参数f(str, aVar)
private static final byte[] f(String str,
com.bilibili.lib.biliid.internal.fingerprint.b.a aVar) {
    Map<String, String> a2 = aVar.a();
    return e(str + a2.get("model") + a2.get("band"));
}
//我们去hook一下这个f方法，看看参数和返回值
str is ==> XY2CAAD2CD8168403CCB095E408A467773352
aVar is ==> 一大堆，但是我们发现在f方法只用aVar取了一个map，所以我们可以暂时先不关注它
//我们注意到这里的str就是我们的buvid
//然后它是拼了一个字符串，然后调用了e方法，e方法是MD5加密
//所以这个拼的字符串比较重要str + a2.get("model") + a2.get("band")
//str就是buvid
//a2是通过aVar取的map，然后去取key为"model"和"band"的值

//我们去看下aVar对象的map
public final class a {
    public final Map<String, String> a() {
        return this.a;
    }
    public String toString() {
        return "Data(main=" + this.a + ", propery=" + this.b + ", sys=" +
this.f13229c + ")";
    }
}
//看完发现，aVar是a类的一个对象，在构造aVar这个对象时，会构造出来这个map，并且看到a类的
toString方法的main=后边就是这个map，所以我们在hook 这个f方法时还是需要去打印一下aVar，这样就
会调用a类的toString方法，这样就可以看到对应的aVar这个对象的map，这样的话就可以找
到"model"和"band"的值
//我们去重新hook一下这个f方法，打印一下参数
str is ==> XY2CAAD2CD8168403CCB095E408A467773352
aVar is ==> Data(main={...,model=Pixel, band=8996-130361-1905270421,... },
propery={...}, sys={...})
//因为信息有很多，所以我就删除了一些，只留下了需要使用的
//model就是我们的手机型号
//band不确定是什么，我们去看看new这个aVar对象的时候，怎么构造的
Set<String> b = a.g().b();
return new a(b.contains("main") ? k0.u() : b(b), b.contains("property") ? k0.u()
: c.a(), b.contains("sys") ? k0.u() :
com.bilibili.lib.biliid.internal.fingerprint.data.android.a.c());
//跟到了这里，new这个对象的时候，传进去的是b.contains("main") ? k0.u() : b(b)就是包
含"model"和"band"的map
//这里是判断b这个集合中是否包含main，进而确定走k0.u()还是b(b)
//经过hook可以确定，这个b此时不包含main，所以走的是b(b)
//我又去hook了这个b方法，在进入b方法前，这个参数b是空的，在b方法结束后返回的时候，就已经包含了
所有的数据，所以对于hashMap的赋值就是在这个b方法中搞的
//我们去看看b方法：
private static final Map<String, String> b(Set<String> set) {
```

```
    HashMap hashMap = new HashMap();
    hashMap.putAll(AppKt.c());

 hashMap.putAll(com.bilibili.lib.biliid.internal.fingerprint.data.android.a.a())
;
    if (!set.contains("hook")) {
        hashMap.putAll(HookKt.d());
    }
    if (!set.contains("va")) {
        hashMap.putAll(VaKt.e());
    }
    if (!set.contains("vm")) {
        hashMap.putAll(d.c());
    }
    if (!set.contains("bluetooth")) {

 hashMap.putAll(com.bilibili.lib.biliid.internal.fingerprint.data.hw.d.a());
    }
    if (!set.contains("camera")) {
        hashMap.putAll(CameraKt.b());
    }
    if (!set.contains("core")) {
        hashMap.putAll(e.a());
    }
    if (!set.contains(TencentLocation.NETWORK_PROVIDER)) {
        hashMap.putAll(NetworkKt.f());
    }
    if (!set.contains("screen")) {
        hashMap.putAll(f.c());
    }
    if (!set.contains("sensor")) {
        hashMap.putAll(SensorKt.c());
    }
    hashMap.putAll(com.bilibili.lib.biliid.internal.fingerprint.data.hw.c.a());
    hashMap.putAll(AppKt.h());
    hashMap.putAll(AppKt.b());
    hashMap.putAll(AppKt.f());
    hashMap.putAll(AppKt.e());
    hashMap.putAll(AppKt.g());
    hashMap.putAll(AppKt.i());
    hashMap.putAll(AppKt.a());
    return hashMap;
}
//这是一系列的put，我们去看看我们要找的model和band在哪里put的
//是在
hashMap.putAll(com.bilibili.lib.biliid.internal.fingerprint.data.android.a.a());
里边put的mopdel和band
//hashMap.putAll是把一个hashMap中的元素，拷贝到hashMap中，所以就是把
com.bilibili.lib.biliid.internal.fingerprint.data.android.a.a()返回的hashMap拷贝到
hashMap中
//在com.bilibili.lib.biliid.internal.fingerprint.data.android.a.a()中，有put model
和band
public static final Map<String, String> a() {
    HashMap hashMap = new HashMap();
    ...
```

```
    String str3 = Build.MODEL;//public static final String KEY_PUB_MODEL =
"model"
    hashMap.put("model", str3);
    String radioVersion = Build.getRadioVersion();
    if (radioVersion == null) {
        radioVersion = "";
    }
    hashMap.put("band", radioVersion);
    ...
    return hashMap;
}
//model就是我们的手机型号，我这里是Pixel，重点是看看band
//band的值是通过android.os.Build.getRadioVersion()得到的，并且这里如果返回的null用""也可
以
//android.os.Build.getRadioVersion()：是一个用于获取设备基带(Baseband)版本信息的方法,同
一批生产的手机返回值可能相同，所以，这个根据不同手机返回的格式进行随机生成即可
//到这里这个band和model的值就搞定了

//再回过来说，我们的fp_local是通过返回str+b(str2)得到的
//str2又是通过String str2 = MiscHelperKt.a(f(str, aVar)) + h() +
MiscHelperKt.a(g())
//其中 MiscHelperKt.a(btye[] byte)在这个场景下是进行hex编码
//我们刚刚分析了第一部分MiscHelperKt.a(f(str, aVar))，
//它的参数f(str, aVar)就是MD5( buvid的值+hashMap.get("model")+hashMap.get("band") )
//其中model就是手机型号，band就是设备基带号


//接着看String str2 = MiscHelperKt.a(f(str, aVar)) + h() + MiscHelperKt.a(g())的第
二部分h()
private static final String h() {
    String format = a.format(new Date(System.currentTimeMillis()));
    x.h(format, "SDF.format(Date(System.currentTimeMillis()))");
    return format;
}
//这里就是获取时间戳，不过是又进行了new Date()转为了：20250505123928这种形式，年月日时分秒

//接着去看String str2 = MiscHelperKt.a(f(str, aVar)) + h() + MiscHelperKt.a(g())的
第三部分MiscHelperKt.a(g())
//外层就是转hex编码，内层就是g()，我们去看看g()：
private static final byte[] g() {
    byte[] a2 = com.bilibili.commons.e.a(8);
    x.h(a2, "RandomUtils.nextBytes(SALT_BYTES)");
    return a2;
}
//又调用了byte[] a2 = com.bilibili.commons.e.a(8);
public static byte[] a(int i) {
    h.b(i >= 0, "Count cannot be negative.", new Object[0]);
    byte[] bArr = new byte[i];
    a.nextBytes(bArr);
    return bArr;
}
//这个就是随机生成一个长度为8的字节数组

//到这里fp_local已经差不多了
//String str2 = MiscHelperKt.a(f(str, aVar)) + h() + MiscHelperKt.a(g())
```

```
//hex(MD5(buvid的值+hashMap.get("model")+hashMap.get("band"))) +时间戳年月日时分秒
+hex(长度为8的随机字节数组)
//这就是str2
//fp_local是str2+b(str2)
//现在str2解决了，接下来就是去搞定b(String str),进而就搞定fp_local了
//b(String str):
public static final String b(String str) {
    int i;
    i iVar = q.SO(q.n1(0, Math.min(str.length() - 1, 62)), 2);
    int g = iVar.g();
    int h = iVar.h();
    int i2 = iVar.i();
    if (i2 < 0 ? g >= h : g <= h) {
        i = 0;
        while (true) {
            String substring = str.substring(g, g + 2);
            i += Integer.parseInt(substring, b.a(16));
            if (g == h) {
                break;
            }
            g += i2;
        }
    } else {
        i = 0;
    }
    e0 e0Var = e0.a;
    String format = String.format("%02x", Arrays.copyOf(new Object[]
{Integer.valueOf(i % 256)}, 1));
    return format;
}
//我们去hook一下这个方法，看看入参和返回值
para is ==> efbaa891c759dbcacebbf89bb3517b7f20250505225142f63c7ae460d208a0
result is ==> 2f
//该方法主要是对输入字符串的特定部分进行处理，将每两个字符作为一个十六进制数进行解析并求和，最后
取模转换为十六进制字符串输出。
//也就是说比如efbaa891c759dbcacebbf89bb3517b7f20250505225142f63c7ae460d208a0
//会先把每两位作为十六进制去看，进而转换为=>0x2f、0xba、0xa8、0x91、0xc7...
//然后把他们转为十进制累加起来
//然后把累加的总和对256求余，然后再进行转为十六进制输出

//总结：fp_local就是:hex(MD5(buvid的值+hashMap.get("model")+hashMap.get("band")))
+时间戳年月日小时分钟秒毫秒+hex(长度为8的随机字节数组)+b(hex(MD5(buvid的值
+hashMap.get("model")+hashMap.get("band"))) +时间戳年月日小时分钟秒毫秒+hex(长度为8的随
机字节数组))
//其中b方法就是对hex(MD5(buvid的值+hashMap.get("model")+hashMap.get("band"))) +时间戳
年月日小时分钟秒毫秒+hex(长度为8的随机字节数组)生成的串进行每两位看成一个十六进制==>转10进制
==>累加起来==>对256取余==>转为16进制
```

fp_remot

现在看看fp_remote

```
String k = com.bilibili.api.a.k();
if (!TextUtils.isEmpty(k)) {
    aVar.h("fp_remote", k);
```

```
}

//去看看com.bilibili.api.a.k()
public static String k() {
    a();
    return b.H();
}

//点进去后发现b是接口，我们继续去看b接收的是哪个类的对象
private static b b;
public static void o(b bVar) {
    b = bVar;
}
//进来后发现和fp_local一样，直击腹背
public static final void b(Application application, boolean z) {
    x.q(application, "app");
    com.bilibili.api.a.o(new a(application));
}
//还是去看a类下的H方法
public String H() {
    String b2 = c.b();
    return b2 != null ? b2 : "";
}
//再去看c.b()
public static String b() {
    return Fingerprint.h.d();
}
//再去看Fingerprint.h.d()
public final String d() {
    if (!k()) {
        return null;
    }
    ReentrantReadWriteLock.ReadLock readLock = e;
    x.h(readLock, "r");
    readLock.lock();
    try {
        return f13228c;
    } finally {
        readLock.unlock();
    }
}
//返回的是f13228c,这个f13228c就是我们的fp_remot,我们去看看它在哪里赋值的。找到了两个地方：
public final void m(String str) {
    f13228c = str;
}

private final void e() {
...
}
//我们分别去hook一下这两个方法，看看有没有走这里
//hook m方法：没反应
//hook e方法：打印日志，走了
//所以我们去看看e方法：
private final void e() {
    e k = e.k();
```

```
    x.h(k, "env");
    String d2 = k.d();
    if ((d2 == null || (s.x1(d2))) ||
!com.bilibili.lib.biliid.internal.fingerprint.a.a.d(k)) {
        if (g) {
            b = DataKt.a();
            d c3 = d.c();
            String a2 = c3.a();
            com.bilibili.lib.biliid.internal.fingerprint.b.a aVar = b;
            if (aVar == null) {
                x.Q("data");
            }
            d2 = com.bilibili.lib.biliid.internal.fingerprint.a.a.a(a2, aVar);
            k.v(d2);
            String encode = Uri.encode(Build.BRAND);
            String encode2 = Uri.encode(Build.MODEL);
            k.s(PersistEnv.KEY_PUB_BRAND, encode);
            k.s(PersistEnv.KEY_PUB_MODEL, encode2);
        } else {
            d2 = "";
        }
    }
    ReentrantReadWriteLock.WriteLock writeLock = f;
    x.h(writeLock, b.w);
    writeLock.lock();
    if (d2 == null) {
        try {
            x.K();
        } catch (Throwable th) {
            writeLock.unlock();
            throw th;
        }
    }
    a = d2;
    f13228c = k.e();
    SharedPreferences.Editor edit =
com.bilibili.base.d.s(com.bilibili.lib.foundation.e.a()).edit();
    String str = a;
    if (str == null) {
        x.Q("buvidLocal");
    }
    edit.putString("fp_local", str).apply();
    w wVar = w.a;
    writeLock.unlock();
    Object[] objArr = new Object[2];
    String str2 = a;
    if (str2 == null) {
        x.Q("buvidLocal");
    }
    objArr[0] = str2;
    objArr[1] = f13228c;
    BLog.vfmt("biliid.fingerprint", "Init fingerprint from env, buvidLocal=%s,
buvidServer=%s.", objArr);
    if (g) {
        com.bilibili.droid.thread.d.g(2, new Fingerprint$doInit$2(k));
```

```
    }
}
//又调用了f13228c = k.e();给f13228c赋值的
public String e() {
    String x = l().x();
    if (!TextUtils.isEmpty(x)) {
        d.x(x);
        return x;
    }
    String e = d.e();
    if (!TextUtils.isEmpty(e)) {
        l().S(e);
    }
    return e;
}
//k.e()只做了两件事情，去xml中取或者去内存中取。
//我们清除缓存，重新抓包发现，是通过向服务器发送请求返回的fp_remot，然后存到xml中的
//所以我们要去分析一下拿fp_remot的接口
```

抓包分析

```
//抓包分析：
请求网址：https://app.bilibili.com/x/resource/fingerprint
请求方式：POST
请求头：
    content-length      17111
    buvid               XY2CAAD2CD8168403CCB095E408A467773352
    device-id           bg0-CTOOb1ZvDTxaJlom
    fp_local
efbaa891c759dbcacebbf89bb3517b7f20250506171O031ae6e44effceafc9b2
    session_id          cec1c8e4
    env                 prod
    app-key             android
    user-agent
    bili-bridge-engine  cronet
    content-type        application/json; charset=utf-8
    accept-encoding     gzip, deflate, br
请求体：
    "key":
    "content":
请求参数：
    appkey              1d8b6e7d45233436
    build               6240300
    c_locale            zh-Hans_CN
    channel             xxl_gdt_wm_253
    mobi_app            android
    platform            android
    s_locale            zh-Hans_CN
    statistics          {"appId":1,"platform":3,"version":"6.24.0","abtest":""}
    ts                  1746522603
    sign                54eb7475e0daee35ea518c458720a11f
```

请求头

请求头都是我们刚刚分析过的，搜索url定位到后发现也是在拦截器中进行设置的

```
public class e extends a {
    @Override
    public final b0 a(b0 b0Var) {
        b0.a h = b0Var.h();//拿到请求
        e(h);//添加请求头，和之前分析的是同一个位置
        d(b0Var.k(), h);//这里是添加请求参数的地方
        return h.b();
    }
```

请求体

```
//我们去搜索请求网址定位一下：
public interface e {
    @POST("/x/resource/fingerprint")
    @RequestInterceptor(com.bilibili.okretro.f.e.class)
    com.bilibili.okretro.d.a<GeneralResponse<FingerprintResponse>>
fingerprint(@Query("access_key") String str, @Body c0 c0Var);
}
//直接查找用例，看下是在哪里发送的请求：
public static final String c(String str, String str2) {
    c0 requestBody = c0.create(w.d(com.hpplay.sdk.source.protocol.d.f18989u),
str2);
    x.h(requestBody, "requestBody");
    FingerprintResponse fingerprintResponse = (FingerprintResponse)
com.bilibili.okretro.i.a.b(((e)
com.bilibili.okretro.c.a(e.class)).fingerprint(str, requestBody).execute());
    if (fingerprintResponse != null) {
        return fingerprintResponse.getBuvidServer();
    }
    return null;
}
//我们hook了这个c方法，其中str2就是我们的请求体，返回值
fingerprintResponse.getBuvidServer()就是我们的fp_remot
//我们再去hook一下c方法，打印一个调用栈，看看调用c方法时传来的参数是怎么构造出来的
java.lang.Throwable
    at com.bilibili.lib.biliid.internal.fingerprint.sync.http.c.c(Native Method)
    at com.bilibili.lib.biliid.internal.fingerprint.sync.http.c.a(BL:1)
    at com.bilibili.lib.biliid.internal.fingerprint.sync.http.d.run(BL:1)
//去看看com.bilibili.lib.biliid.internal.fingerprint.sync.http.c.a
//发现并没有a方法，于是我用GDA反编译了一下，发现有一个a方法，jadx没有反编译出来
public static final String a(String p0,String p1){
    return c.c(p0, p1);
}
//那p1就是我们的请求体，p0是"access_key"的值不过是null
//我们看看谁调用了a，再看看调用a的时候p1是怎么来的
public void run(){
    String str = c.a(this.c, this.d);
    Object[] objArray = new Object[]{str};
```

```
    BLog.ifmt("biliid.httpposter", "Fingerprint buvidServer returns %s.",
objArray);
    int i = (str != null && !k.x1(str))? 0: 1;
    if (!i) {
        this.e.invoke(str);
    }
    return;
}
//this.d就是我们的请求体
//this.d是在构造方法d中进行赋值的
public void d(String p0,String p1,l p2){
    x.q(p1, "jsonStr");
    x.q(p2, "cb");
    super();
    this.c = p0;
    this.d = p1;
    this.e = p2;
    this.b = b.a();
}
//再去看谁调用了构造方法d，第二个参数就是请求体
public static final void d(String p0,String p1,String p2,l p3){
    x.q(p1, "key");
    x.q(p2, "ciphered");
    x.q(p3, "cb");
    c.a.post(new d(p0, new PostBodyModel(p1, p2).toJsonString(), p3));
}
//是在sync.http.c.d方法中调用了
c.a.post(new d(p0, new PostBodyModel(p1, p2).toJsonString(), p3));
//请求体就是new PostBodyModel(p1, p2)
public void PostBodyModel(String p0,String p1){
    x.q(p0, "key");
    x.q(p1, "content");
    super();
    this.key = p0;
    this.content = p1;
}
//参数p1就是key，参数p2就是content。我们要看调用sync.http.c.d时的p1和p2


//我们去看d的的用例，跟好第二和第三个参数
public static final void a(String str,
com.bilibili.lib.biliid.internal.fingerprint.b.a aVar, l<? super String, w>
lVar) {
    x.q(str, "buvidLocal");
    x.q(aVar, "raw");
    x.q(lVar, "cb");
    com.bilibili.lib.biliid.utils.a a = com.bilibili.lib.biliid.utils.a.d.a();
    if (a == null) {
        x.K();
    }
    String c3 = a.c();
    Pair<String, String> a2 =
c.a(com.bilibili.lib.biliid.internal.fingerprint.d.b.a.b(str, aVar));
    String component1 = a2.component1();
    String component2 = a2.component2();
```

```
    if ((!s.x1(component1)) && (!s.x1(component2))) {
        //这里调用的d
        com.bilibili.lib.biliid.internal.fingerprint.sync.http.c.d(c3,
component1, component2, lVar);
    }
}
//调用d的时候传入的是c3, component1, component2, lVar
// component1, component2就是key和ciphered
//component1, component2是调用Pair类对象a2的component1()方法得到的:
public final class Pair<A, B> implements Serializable {
    private final A first;
    private final B second;

    public Pair(A a, B b) {
        this.first = a;
        this.second = b;
    }

public final A component1() {
    return this.first;
    }

public final B component2() {
    return this.second;
    }
}
//并且只有在构造器中有对first和second进行赋值，我们去看看获取a2的地方:
Pair<String, String> a2 =
c.a(com.bilibili.lib.biliid.internal.fingerprint.d.b.a.b(str, aVar));
//c.a(byte[] r5)方法反编译不出来里边的逻辑;
//com.bilibili.lib.biliid.internal.fingerprint.d.b.a.b(str, aVar)返回的是一个byte数
组，参数1是buvidf，参数2是一个对象，里边很多数据
//a2的初始化是在c.a(byte[] r5)里边进行的
//换GDA试试，全路径为:com.bilibili.lib.biliid.internal.fingerprint.d.b.b.c.a
 public static final Pair a(byte[] p0){
     Application uApplication;
     Pair pair;
     String str = "plain";
     try{
         x.q(p0, str);
         if ((uApplication = BiliContext.f()) == null) {
             x.K();
             label_0010 :
             String str1 = a.e();
             str = b.b(str1,
b.d(uApplication.getAssets().open("rsa_public_key.pem")));
             String str2 = a.b(p0, str1);
             str1 = 0;
             int i = (str != null && !k.x1(str))? 0: 1;
             if (!i) {
                 if (str2 == null || k.x1(str2)) {
                     str1 = 1;
                 }
                 if (!str1) {
                     if (str2 == null) {
```

```
                        x.K();
                    }
                    pair = new Pair(str, str2);
                }
            }
            pair = new Pair("", "");
        }else {
            goto label_0010 ;
        }
    }catch(java.lang.Exception e5){
        BLog.e("biliid.security", e5.getCause());
        pair = new Pair("", "");
    }
    return pair;
}


//直接找new Pair的
pair = new Pair(str, str2);//最终返回的也是pair，str、str2就是我们的key和content
//看了一下str和str2的生成都会用的str1，我们先把str1搞出来，str1可能是密钥
String str1 = a.e();

public static String e(){
    return a.a(a.d().getEncoded());
}

public static SecretKeySpec d(){
    String str = "AES";
    try{
        KeyGenerator instance = KeyGenerator.getInstance(str);
        instance.init(128, new SecureRandom());
        return new SecretKeySpec(instance.generateKey().getEncoded(), str);
    }catch(java.security.NoSuchAlgorithmException e0){
        e0.printStackTrace();
        return null;
    }
}

//经过hook验证，str1就是AES的随机密钥，那么这样的话就一定有一个和服务器发送AES密钥的过程。接下来先看str和str2是怎么来的
//先看str2：
String str2 = a.b(p0, str1);//传过去的是明文字节数组和AES的随机密钥
public static String b(byte[] p0,String p1){
    return a.a(a.c(p0, a.f(p1)));
}
//进来后先又调用了return a.a(a.c(p0, a.f(p1)));
//从外往里看，先看a.a
public static String a(byte[] p0){
    if (p0 == null) {
        return null;
    }
    StringBuffer str = new StringBuffer((p0.length * 2));
    for (int i = 0; i < p0.length; i = i + 1) {
        int i1 = p0[i] & 0x00ff;
        String str1 = Integer.toHexString(i1);
```

```
        if (str1.length() == 1) {
            str = str.append("0");
        }
        str = str.append(str1);
    }
    return str.toUpperCase();
}
//这是字节数组转hex编码然后大写

//再往里看a.c:
public static byte[] c(byte[] p0,byte[] p1){
    try{
        Cipher instance = Cipher.getInstance("AES/ECB/PKCS5Padding");
        instance.init(1, new SecretKeySpec(p1, "AES"));
        return instance.doFinal(p0);
    }catch(java.lang.Exception e4){
        e4.printStackTrace();
        return null;
    }
}
//这里就是进行AES加密了，密钥为p1，p0是明操作后的字节数组(p0的生成后边分析)

//再往里看 a.f(p1):
private static byte[] f(String p0){
    int i = 0;
    if (p0 != null) {
        int i1 = 2;
        if (p0.length() >= i1) {
            p0 = p0.toLowerCase();
            int i2 = p0.length() / i1;
            byte[] uobyteArray1 = new byte[i2];
            for (; i < i2; i = i + 1) {
                int i3 = i * 2;
                int i4 = i3 + 2;
                String str = p0.substring(i3, i4);
                i3 = Integer.parseInt(str, 16) & 0x00ff;
                uobyteArray1[i] = (byte)i3;
            }
            return uobyteArray1;
        }
    }
    byte[] uobyteArray = new byte[i];
    return uobyteArray;
}
//f是把十六进制字符串转为字节数组的。因为我们生成AES密钥后返回的是String类型的，但是在加密的时候需要字节数组，所以这里需要还原回去
//到这里str2就解决了，是用AES随机密钥对传过来的字节数组进行加密，然后hex编码，然后转大写，完事之后就是我们抓包得到的content



//下边去看str的生成:
str = b.b(str1, b.d(uApplication.getAssets().open("rsa_public_key.pem")));
//其中str1是AES随机生成的密钥，
b.d(uApplication.getAssets().open("rsa_public_key.pem"))是拿到rsa公钥
```

```
public static RSAPublicKey d(InputStream p0){
    String str1;
    try{
        BufferedReader uBufferedRea = new BufferedReader(new
InputStreamReader(p0));
        StringBuilder str = "";
        while ((str1 = uBufferedRea.readLine()) != null) {
            if (str1.charAt(0) == '-') {
                continue ;
            }else {
                str = str.append(str1).append(13);
            }
        }
        return b.e(str);
    }catch(java.io.IOException e0){
        Exception uException = new Exception("公钥数据流读取错误");
        throw uException;
    }catch(java.lang.NullPointerException e0){
        throw new Exception("公钥输入流为空");
    }
}

//然后调用b.b把明文和rsa公钥传过去了，公钥在jadx-assets目录下找到了
public static String b(String p0,RSAPublicKey p1){
    String[] stringArray = b.f(p0, ((p1.getModulus().bitLength() / 8) - 11));
    int len = stringArray.length;
    String str = "";
    for (int i = 0; i < len; i = i + 1) {
        str = "".append(str).append(b.a(b.c(stringArray[i].getBytes(),
p1))).toString();
    }
    return str;
}


//用pyhton对b方法进行了重写，把rsa公钥内置进去了。python代码放在章节末尾了！

//现在已经搞清楚了new Pair时的参数，也就是请求体的key和content
//key：是随机生成的AES密钥，用rsa公钥分组加密后再转hex编码转大写
//content是对com.bilibili.lib.biliid.internal.fingerprint.d.b.a.b(buvidLocal,
raw)的结果用AES密钥进行加密后转为hex编码再转大写

//接下来把com.bilibili.lib.biliid.internal.fingerprint.d.b.a.b(buvidLocal, raw)搞清
楚后就能完全复现content了
public static final byte[] b(String buvidLocal,
com.bilibili.lib.biliid.internal.fingerprint.b.a data) {
    x.q(buvidLocal, "buvidLocal");
    x.q(data, "data");
    if (tv.danmaku.android.util.a.b.b()) {
        BLog.v("fptag", "buvid_local: " + buvidLocal);
        for (Map.Entry<String, String> entry : data.a().entrySet()) {
            BLog.v("fptag", entry.getKey() + ": " + entry.getValue());
        }
        BLog.v("fptag", "sys:");
        for (Map.Entry<String, String> entry2 : data.c().entrySet()) {
```

```
            BLog.v("fptag", entry2.getKey() + ": " + entry2.getValue());
        }
        BLog.v("fptag", "props:");
        for (Map.Entry<String, String> entry3 : data.b().entrySet()) {
            BLog.v("fptag", entry3.getKey() + ": " + entry3.getValue());
        }
    }
    byte[] byteArray = a(buvidLocal, data).toByteArray();
    x.h(byteArray, "adapt(buvidLocal, data).toByteArray()");
    return byteArray;
}
//代码这么多主要都是打日志关键的就是a(buvidLocal, data).toByteArray();它的返回值就是最终结果，所以要继续去看a方法
```

```
private static final b a(String p0,a p1){
    String str;
    Long longx;
    Integer integer;
    Long longx1;
    String str1;
    int i;
    String str2;
    String[] stringArray;
    List list;
    Iterator iterator;
    Float uFloat;
    b$a uoa = b.w2();
    Map map = p1.a();
    if ((str = map.get("sdkver")) != null) {
        uoa.I1(str);
    }
    if ((str = map.get("app_id")) != null) {
        uoa.s(str);
    }
    if ((str = map.get("app_version")) != null) {
        uoa.t(str);
    }
    if ((str = map.get("app_version_code")) != null) {
        uoa.v(str);
    }
    if ((str = map.get("mid")) != null) {
        uoa.r1(str);
    }
    if ((str = map.get("chid")) != null) {
        uoa.P(str);
    }
    if ((str = map.get("fts")) != null && (longx = k.G0(str)) != null) {
        uoa.J0(longx.longValue());
    }
    uoa.J(p0);
    if ((p0 = map.get("first")) != null && (integer = k.E0(p0)) != null) {
        uoa.G0(integer.intValue());
    }
    if ((integer = map.get("proc")) != null) {
        uoa.D1(integer);
```

```
    }
    if ((integer = map.get("net")) != null) {
        uoa.t1(integer);
    }
    if ((integer = map.get("band")) != null) {
        uoa.z(integer);
    }
    if ((integer = map.get("osver")) != null) {
        uoa.C1(integer);
    }
    if ((integer = map.get("t")) != null && (longx1 = k.G0(integer)) != null) {
        uoa.X1(longx1.longValue());
    }
    if ((longx1 = map.get("cpuCount")) != null && (integer = k.E0(longx1)) !=
null) {
        uoa.T(integer.intValue());
    }
    if ((integer = map.get("model")) != null) {
        uoa.s1(integer);
    }
    if ((integer = map.get("brand")) != null) {
        uoa.E(integer);
    }
    if ((integer = map.get("screen")) != null) {
        uoa.H1(integer);
    }
    if ((integer = map.get("cpuModel")) != null) {
        uoa.t0(integer);
    }
    if ((integer = map.get("btmac")) != null) {
        uoa.H(integer);
    }
    if ((integer = map.get("boot")) != null && (longx1 = k.G0(integer)) != null)
{
        uoa.D(longx1.longValue());
    }
    if ((longx1 = map.get("emu")) != null) {
        uoa.E0(longx1);
    }
    if ((longx1 = map.get("oid")) != null) {
        uoa.w1(longx1);
    }
    if ((longx1 = map.get("network")) != null) {
        uoa.u1(longx1);
    }
    if ((longx1 = map.get("mem")) != null && (longx1 = k.G0(longx1)) != null) {
        uoa.o1(longx1.longValue());
    }
    if ((longx1 = map.get("sensor")) != null) {
        uoa.J1(longx1);
    }
    if ((longx1 = map.get("cpuFreq")) != null && (longx1 = k.G0(longx1)) !=
null) {
        uoa.i0(longx1.longValue());
    }
```

```
    if ((longx1 = map.get("cpuVendor")) != null) {
        uoa.u0(longx1);
    }
    if ((longx1 = map.get("sim")) != null) {
        uoa.K1(longx1);
    }
    if ((longx1 = map.get("brightness")) != null && (integer = k.E0(longx1)) !=
null) {
        uoa.F(integer.intValue());
    }
    uoa.f(p1.b());
    uoa.g(p1.c());
    if ((integer = map.get("wifimac")) != null) {
        uoa.p2(integer);
    }
    if ((integer = map.get("adid")) != null) {
        uoa.m(integer);
    }
    if ((integer = map.get("os")) != null) {
        uoa.B1(integer);
    }
    if ((integer = map.get("imei")) != null) {
        uoa.V0(integer);
    }
    if ((integer = map.get("cell")) != null) {
        uoa.O(integer);
    }
    if ((integer = map.get("imsi")) != null) {
        uoa.W0(integer);
    }
    if ((integer = map.get("iccid")) != null) {
        uoa.T0(integer);
    }
    if ((integer = map.get("camcnt")) != null && (integer = k.E0(integer)) !=
null) {
        uoa.K(integer.intValue());
    }
    if ((integer = map.get("campx")) != null) {
        uoa.M(integer);
    }
    if ((integer = map.get("totalSpace")) != null && (longx1 = k.G0(integer)) !=
null) {
        uoa.Z1(longx1.longValue());
    }
    if ((longx1 = map.get("axposed")) != null) {
        uoa.x(longx1);
    }
    if ((longx1 = map.get("maps")) != null) {
        uoa.k1(longx1);
    }
    if ((longx1 = map.get("files")) != null) {
        uoa.F0(longx1);
    }
    if ((longx1 = map.get("virtual")) != null) {
        uoa.i2(longx1);
```

```
    }
    if ((longx1 = map.get("virtualproc")) != null) {
        uoa.l2(longx1);
    }
    if ((longx1 = map.get("gadid")) != null) {
        uoa.LO(longx1);
    }
    if ((longx1 = map.get("glimit")) != null) {
        uoa.MO(longx1);
    }
    if ((longx1 = map.get("apps")) != null) {
        uoa.w(longx1);
    }
    if ((longx1 = map.get("guid")) != null) {
        uoa.QO(longx1);
    }
    if ((longx1 = map.get("uid")) != null) {
        uoa.e2(longx1);
    }
    if ((longx1 = map.get("root")) != null && (integer = k.EO(longx1)) != null)
{
        uoa.E1(integer.intValue());
    }
    if ((integer = map.get("camzoom")) != null) {
        uoa.N(integer);
    }
    if ((integer = map.get("camlight")) != null) {
        uoa.L(integer);
    }
    if ((integer = map.get("oaid")) != null) {
        uoa.v1(integer);
    }
    if ((integer = map.get("udid")) != null) {
        uoa.c2(integer);
    }
    if ((integer = map.get("vaid")) != null) {
        uoa.h2(integer);
    }
    if ((integer = map.get("aaid")) != null) {
        uoa.h(integer);
    }
    if ((integer = map.get("androidapp2O")) != null) {
        uoa.o(integer);
    }
    if ((integer = map.get("androidappcnt")) != null) {
        uoa.p(Integer.parseInt(integer));
    }
    if ((integer = map.get("androidsysapp2O")) != null) {
        uoa.r(integer);
    }
    if ((integer = map.get("battery")) != null && (integer = k.EO(integer)) !=
null) {
        uoa.A(integer.intValue());
    }
    if ((integer = map.get("batteryState")) != null) {
```

```
            uoa.B(integer);
        }
        if ((integer = map.get("bssid")) != null) {
            uoa.G(integer);
        }
        if ((integer = map.get("build_id")) != null) {
            uoa.I(integer);
        }
        if ((integer = map.get("countryIso")) != null) {
            uoa.S(integer);
        }
        if ((integer = map.get("free_memory")) != null && (longx1 = k.GO(integer))
!= null) {
            uoa.HO(longx1.longValue());
        }
        if ((longx1 = map.get("fstorage")) != null) {
            uoa.IO(longx1);
        }
        if ((longx1 = map.get("kernel_version")) != null) {
            uoa.a1(longx1);
        }
        if ((longx1 = map.get("languages")) != null) {
            uoa.b1(longx1);
        }
        if ((longx1 = map.get("mac")) != null) {
            uoa.p2(longx1);
        }
        if ((longx1 = map.get("ssid")) != null) {
            uoa.O1(longx1);
        }
        if ((longx1 = map.get("systemvolume")) != null && (integer = k.EO(longx1))
!= null) {
            uoa.V1(integer.intValue());
        }
        if ((integer = map.get("wifimaclist")) != null) {
            uoa.q2(integer);
        }
        if ((integer = map.get("memory")) != null && (longx1 = k.GO(integer)) !=
null) {
            uoa.p1(longx1.longValue());
        }
        if ((longx1 = map.get("str_battery")) != null) {
            uoa.S1(longx1);
        }
        if ((longx1 = map.get("is_root")) != null) {
            uoa.YO(Boolean.parseBoolean(longx1));
        }
        if ((longx1 = map.get("str_brightness")) != null) {
            uoa.U1(longx1);
        }
        longx1 = "str_app_id";
        if ((str1 = map.get(longx1)) != null) {
            uoa.R1(str1);
        }
        if ((longx1 = map.get(longx1)) != null) {
```

```
        uoa.R1(longx1);
    }
    if ((longx1 = map.get("light_intensity")) != null) {
        uoa.e1(longx1);
    }
    longx1 = map.get("device_angle");
    str1 = ",";
    if (longx1 != null) {
        i = (longx1 != null && longx1.length())? 0: 1;
        str2 = ((i ^ 1))? longx1: null;
        if (str2 != null) {
            stringArray = new String[]{str1};
            if ((list = k.n4(str2, stringArray, false, 0, 6, null)) != null) {
                ArrayList uArrayList = new ArrayList(n.Q(list, 10));
                iterator = list.iterator();
                while (iterator.hasNext()) {
                    float f = ((uFloat = k.q0(iterator.next())) != null)?
uFloat.floatValue(): 0;
                    uArrayList.add(Float.valueOf(f));
                }
                uoa.c(uArrayList);
            }
        }
    }
    if ((iterator = map.get("gps_sensor")) != null && (longx1 = k.G0(iterator))
!= null) {
        uoa.N0(longx1.longValue());
    }
    if ((longx1 = map.get("speed_sensor")) != null && (longx1 = k.G0(longx1)) !=
null) {
        uoa.L1(longx1.longValue());
    }
    if ((longx1 = map.get("linear_speed_sensor")) != null && (longx1 =
k.G0(longx1)) != null) {
        uoa.f1(longx1.longValue());
    }
    if ((longx1 = map.get("gyroscope_sensor")) != null && (longx1 =
k.G0(longx1)) != null) {
        uoa.S0(longx1.longValue());
    }
    if ((longx1 = map.get("biometric")) != null && (longx1 = k.G0(longx1)) !=
null) {
        uoa.C(longx1.longValue());
    }
    if ((longx1 = map.get("biometrics")) != null) {
        i = (longx1 != null && longx1.length())? 0: 1;
        str2 = ((i ^ 1))? longx1: null;
        if (str2 != null) {
            stringArray = new String[]{str1};
            if ((list = k.n4(str2, stringArray, false, 0, 6, null)) != null) {
                uoa.b(list);
            }
        }
    }
```

```
    if ((list = map.get("last_dump_ts")) != null && (longx1 = k.GO(list)) !=
null) {
        uoa.c1(longx1.longValue());
    }
    if ((longx1 = map.get("location")) != null) {
        uoa.i1(longx1);
    }
    if ((longx1 = map.get("country")) != null) {
        uoa.R(longx1);
    }
    if ((longx1 = map.get("city")) != null) {
        uoa.Q(longx1);
    }
    if ((longx1 = map.get("data_activity_state")) != null && (integer =
k.EO(longx1)) != null) {
        uoa.yO(integer.intValue());
    }
    if ((integer = map.get("data_connect_state")) != null && (integer =
k.EO(integer)) != null) {
        uoa.BO(integer.intValue());
    }
    if ((integer = map.get("data_network_type")) != null && (integer =
k.EO(integer)) != null) {
        uoa.DO(integer.intValue());
    }
    if ((integer = map.get("voice_network_type")) != null && (integer =
k.EO(integer)) != null) {
        uoa.m2(integer.intValue());
    }
    if ((integer = map.get("voice_service_state")) != null && (integer =
k.EO(integer)) != null) {
        uoa.o2(integer.intValue());
    }
    if ((integer = map.get("usb_connected")) != null && (integer =
k.EO(integer)) != null) {
        uoa.g2(integer.intValue());
    }
    if ((integer = map.get("adb_enabled")) != null && (integer = k.EO(integer))
!= null) {
        uoa.l(integer.intValue());
    }
    if ((integer = map.get("ui_version")) != null) {
        uoa.d2(integer);
    }
    if ((integer = map.get("accessibility_service")) != null) {
        uoa.a(JSON.parseArray(integer, String.class));
    }
    if ((integer = map.get("sensors_info")) != null) {
        uoa.e(a.c(integer));
    }
    GeneratedMessageLite generatedMes = uoa.build();
    x.h(generatedMes, "DeviceInfo.newBuilder\(\).…orList\(\)\) }\n    }.build\
(\)");
    return generatedMes;
}
```

```
//然后再对a方法的返回值做toByteArray()就得到字节数组了，然后再用AES加密然后转hex编码转大写即可
toByteArray():

public byte[] toByteArray() {
    try {
        byte[] bArr = new byte[getSerializedSize()];
        CodedOutputStream newInstance = CodedOutputStream.newInstance(bArr);
        writeTo(newInstance);
        newInstance.checkNoSpaceLeft();
        return bArr;
    } catch (IOException e) {
        throw new RuntimeException(getSerializingExceptionMessage("byte array"),
e);
    }
}
```

请求参数-拦截器分析

```
请求参数：
    appkey              1d8b6e7d45233436
    build               6240300
    c_locale            zh-Hans_CN
    channel             xxl_gdt_wm_253
    mobi_app            android
    platform            android
    s_locale            zh-Hans_CN
    statistics          {"appId":1,"platform":3,"version":"6.24.0","abtest":""}
    ts                  1746522603
    sign                54eb7475e0daee35ea518c458720a11f
//在定义这个请求接口的时候，只定义了一个请求参数：@Query("access_key") String str和一个请求体@Body c0 c0Var，并没有定义其他的请求参数，并且在我们抓包得到的数据中并没有定义的这个参数，是因为在调用的时候，传入的是null，所以在最终发送请求的时候就会过滤掉。(经hook确定传入的是null)


//这里没有定义其他的参数，但是定义了自定义注解式拦截器，我们去看下拦截器
public interface e {
    @POST("/x/resource/fingerprint")
    @RequestInterceptor(com.bilibili.okretro.f.e.class)
    com.bilibili.okretro.d.a<GeneralResponse<FingerprintResponse>>
fingerprint(@Query("access_key") String str, @Body c0 c0Var);
}

//去看看拦截器定义的地方：@RequestInterceptor(com.bilibili.okretro.f.e.class)
public class e extends a {
    @Override
    public final b0 a(b0 b0Var) {
        b0.a h = b0Var.h();//拿到请求
```

```
            e(h);//添加请求头，和之前分析的是同一个位置
            d(bOVar.k(), h);//这里是添加请求参数的地方
            return h.b();
        }
    }
    //去看看d(bOVar.k(), h)：
    public void d(u uVar, bO.a aVar) {
        HashMap hashMap = new HashMap();
        int L = uVar.L();
        for (int i = 0; i < L; i++) {
            g(uVar.H(i), uVar.J(i), hashMap);
        }
        b(hashMap);
        aVar.s(uVar.s().n(h(hashMap).toString()).h());
    }
    //首先是先new一个HashMap，然后通过for循环，往HashMap中put一些值，put完了之后调用b方法，把刚刚的hashMap传过去，再进行一下操作
    //b方法是在拦截器中的一个方法
    //在获取fp_remot的场景下，uVar.L()返回的是0，所以没走for循环，因此hashMap是空的，然后进入b方法后put了除ts、sign之外的一些数据
    //b方法：
    public void b(Map<String, String> map) {
        map.put("platform", "android");
        map.put("mobi_app", com.bilibili.api.a.l());
        map.put("appkey", f());
        map.put("build", String.valueOf(com.bilibili.api.a.f()));
        map.put("channel", com.bilibili.api.a.g());
        String b = com.bilibili.api.a.b();
        if (!TextUtils.isEmpty(b)) {
            map.put("access_key", b);
        }
        Map<String, String> i = com.bilibili.api.a.i();
        if (i != null) {
            map.putAll(i);
        }
        map.put("c_locale", com.bilibili.api.a.h());
        map.put("s_locale", com.bilibili.api.a.n());
    }

    //com.bilibili.api.a.i()：
    public final Map<? extends String, String> c() {
        HashMap hashMap = new HashMap();
        hashMap.put("statistics", new wO(1, 3, "6.24.0", ABTesting.b()).a());
        return hashMap;
    }
    //跟随com.bilibili.api.a.i()一路，最终追到了c方法中，发现是只put了"statistics"

    //最终的请求参数：
        appkey              1d8b6e7d45233436
        build               6240300
        c_locale            zh-Hans_CN
        channel             xxl_gdt_wm_253
        mobi_app            android
        platform            android
        s_locale            zh-Hans_CN
```

```
    statistics          {"appId":1,"platform":3,"version":"6.24.0","abtest":""}
    ts                  1746522603
    sign                54eb7475e0daee35ea518c458720a11f
//请求参数的总结：
//appkey、build、c_locale、channel、mobi_app、platform、s_locale、statistics这几个是直接在b方法中put的
//ts、sign是通过调用native层的方法放进去的；hook native层的方法，观察入参和返回值确定的


//在调用完b方法后，会调用h方法，然后把刚刚的hashMap传过去
h(hashMap).toString()

public SignedQuery h(Map<String, String> map) {
  return LibBili.g(map);
}
//在h方法中调用了LibBili.g(map)

public static SignedQuery g(Map<String, String> map) {
    return s(map == null ? new TreeMap() : new TreeMap(map));
}
//在g方法中，会把之前put进map的参数放到treeMap中，进行默认排序，然后调用s方法

static native SignedQuery s(SortedMap<String, String> sortedMap);
 //然后在native方法s中会添加一个ts，添加一个sign。添加完后还会根据key进行排序，然后返回一个SignedQuery对象，
//其中SignedQuery下的a属性为除sign之外的参数，b属性为sign
public final class SignedQuery {
    public final String a;
    public final String b;
    public SignedQuery(String str, String str2) {
        this.a = str;
        this.b = str2;
    }
    public String toString() {
        String str = this.a;
        if (str == null) {
            return "";
        }
        if (this.b == null) {
            return str;
        }
        return this.a + "&sign=" + this.b;
    }
}
```

appkey

```
//刚刚分析了appkey是在b方法中put进去的，所以我们去看看appkey的值是怎么得到的
map.put("appkey", f());

//再去看看f():
public String f() {
    return com.bilibili.api.a.d();
}
```

```
//接着再去看看 com.bilibili.api.a.d():
public static String d() {
    return LibBili.e(l());
}

//接着再去看LibBili.e(l())
public static String e(String str) {
    return a(str);
}

//接着再去看a(str)
private static native String a(String str);

//发现a是一个native方法，我们去看看参数。
//在com.bilibili.api.a.d()里边才加入的参数：LibBili.e(l())
//所以我们要去看l()返回的是什么
//l()最终调用的A()：返回的是"android"
public String A() {
    return "android";
}
//参数是固定的"android"
//经过hook多次验证，返回值也是固定的，里边没有添加变化的值，所以这个appkey就是固定的
//但是为了学习，我们去so中分析一下这个appkey是怎么来的

static {
    com.getkeepsafe.relinker.c.c("bili");
}
//在本类中，我们看到它加载了一个so==>libbili.so
//我们去看看这个so中有没有对a方法的注册，只有32位的，那就拿32位的看看
```

进来先看导出表中有没有静态注册的。不是静态注册的。那我们直接去hook RegisterNatives

```
com.bilibili.nativelibrary.LibBili a (Ljava/lang/String;)Ljava/lang/String;
0xbf857c7d libbili.so 0x1c7d

com.bilibili.nativelibrary.LibBili ao (Ljava/lang/String;II)Ljava/lang/String;
0xbf857c83 libbili.so 0x1c83

com.bilibili.nativelibrary.LibBili b
(Ljava/lang/String;)Ljavax/crypto/spec/IvParameterSpec; 0xbf857c91 libbili.so
0x1c91

com.bilibili.nativelibrary.LibBili s
(Ljava/util/SortedMap;)Lcom/bilibili/nativelibrary/SignedQuery; 0xbf857c97
libbili.so 0x1c97

com.bilibili.nativelibrary.LibBili so
(Ljava/util/SortedMap;II)Lcom/bilibili/nativelibrary/SignedQuery; 0xbf857c9d
libbili.so 0x1c9d

com.bilibili.nativelibrary.LibBili so (Ljava/util/SortedMap;
[B)Lcom/bilibili/nativelibrary/SignedQuery; 0xbf857cab libbili.so 0x1cab

com.bilibili.nativelibrary.LibBili getCpuCount ()I 0xbf857cb3 libbili.so 0x1cb3
```

```
com.bilibili.nativelibrary.LibBili getCpuId ()I 0xbf857cb7 libbili.so 0x1cb7
```

```
//直接跳到 0x1c7d这个位置，这里就是a方法在c层的位置
jstring __fastcall sub_1C7C(JNIEnv *a1, jclass a2, int str)
{
  return sub_373C(a1, str, 0, 0);
}
//进来后发现又调用了sub_373C(a1, str, 0, 0);。

//进sub_373C(a1, str, 0, 0);看看， str是java层传来的"android"、a3、a4都是0
jstring __fastcall sub_373C(JNIEnv *env, int jstr, int a3, int a4)
{
  const char *v8; // r9
  char **v9; // r6
  const char *v10; // r5

  v8 = (*env)->GetStringUTFChars(env, jstr, 0);
  //v8是jstr转为了cstr

  if ( a3 == 1 )
  {
    v9 = &off_B0A4;
    if ( (a4 | 1) != 3 )
      v9 = &off_B0C0;
  }
  else
  {
    v9 = &off_B0DC;
  }
  //首先判断a3是否等于1，这里传过来的是0，所以为假，进而把 v9 = &off_B0DC;
  //off_B0DC就是"1d8b6e7d45233436"
  if ( strcmp(v8, "android") )//这里直接判断strcmp(v8, "android")，这个场景下，v8确实是"android"进而返回0，if直接结束
  {
    if ( !strcmp(v8, "android_i") )
    {
      ++v9;
    }
    else if ( !strcmp(v8, "android_b") )
    {
      v9 += 2;
    }
    else if ( !strcmp(v8, "android_tv") )
    {
      v9 += 3;
    }
    else if ( !strcmp(v8, "biliLink") )
    {
      v9 += 4;
    }
    else if ( !strcmp(v8, "android_bilithings") )
    {
      v9 += 5;
    }
    else if ( !strcmp(v8, "bstar_a") )
```

```
    {
      v9 += 6;
    }
  }
  v10 = *v9;
   //if结束后，把v9给了v10，然后把v10返回java层
  (*env)->ReleaseStringUTFChars(env, (jstring)jstr, v8);
  return (*env)->NewStringUTF(env, v10);
}


//总的来说
//appkey就是通过调用native方法a来拿到的，调用a的时候传入了固定字符串"android"，在so层通过判断是不是"android"进而返回“1d8b6e7d45233436”
```

build

```
 map.put("build", String.valueOf(com.bilibili.api.a.f()));
//直接把com.bilibili.api.a.f())的返回值转为了String
//我们去看com.bilibili.api.a.f())
public static int f() {
    a();
    return b.J();
}
//b是接口，看b赋值的具体类
@Override // com.bilibili.api.a.b
public int J() {
    return 6240300;
}

//直接返回6240300，所以build是固定的
```

c_locale

```
map.put("c_locale", com.bilibili.api.a.h());
//去看看 com.bilibili.api.a.h()、
public static String h() {
    a();
    return b.B();
}
//也是去看b的实现类里边的B方法
@Override // com.bilibili.api.a.b
public String B() {
    return tv.danmaku.bili.b0.l.a.a.a();
}

//看tv.danmaku.bili.b0.l.a.a.a()
public final String a() {
    try {
        return c(tv.danmaku.bili.j0.a.a.b());
    } catch (Exception e) {
        BLog.efmt("bilow.locale", "Exception when get current locale " + e, new
Object[0]);
        return "";
    }
```

```
}

//里边有调用了c(tv.danmaku.bili.j0.a.a.b())
private final String c(Locale locale) {
    boolean x1;
    String script;
    String str = "";
    String str2 = (locale == null || (str2 = locale.getLanguage()) == null) ? ""
: "";
    if (Build.VERSION.SDK_INT >= 21 && locale != null && (script =
locale.getScript()) != null) {
        str = script;
    }
    String country = locale != null ? locale.getCountry() : null;
    x1 = s.x1(str);
    if (!(!x1)) {
        return str2 + '_' + country;
    }
    return str2 + '-' + str + '_' + country;
}

//这个主要是灵活处理语言、脚本和国家代码，生成符合Android多语言支持的标识符，是国际化开发中的关
键工具方法
//这里的值是zh-Hans_CN（简体中文-中国）即语言代码-脚本代码_国家代码
```

channel

```
map.put("channel", com.bilibili.api.a.g());

//去看 com.bilibili.api.a.g()
public static String g() {
    a();
    return b.getChannel();
}

//直接去看b接口实现类的getChannel方法
@Override // com.bilibili.api.a.b
public String getChannel() {
    return w.e.b();
}

//再看w.e.b()
public final String b() {
    Application f = BiliContext.f();
    if (f == null) {
        kotlin.jvm.internal.x.K();
    }
    if (TextUtils.isEmpty(w.f24149c)) {
        c2.f.b0.c.b.b.a.a env = c2.f.b0.c.b.b.a.a.E();
        kotlin.jvm.internal.x.h(env, "env");
        String y = env.y();
        if (TextUtils.isEmpty(y)) {
            y = a();
            b0.b bVar = new b0.b();
            bVar.n(f);
```

```
            bVar.m(y);
            b0 a = bVar.a();
            String b = a != null ? a.b() : null;
            if (!TextUtils.isEmpty(b)) {
                if (b == null) {
                    kotlin.jvm.internal.x.K();
                }
                w.d = b;
                y = b;
            } else {
                w.d = w.b;
            }
            env.T(y);
            Application f2 = BiliContext.f();
            if (f2 == null) {
                kotlin.jvm.internal.x.K();
            }
            com.bilibili.base.d.s(f2).edit().putString(w.a, w.d).commit();
        }
        w.f24149c = y;
    }
    String str = w.f24149c;
    if (str == null) {
        kotlin.jvm.internal.x.K();
    }
    return str;
}

//最终返回的是str，str就是我们的channel
//str是由w.f24149c赋值的
//w.f24149c是由y赋值的
//y又是由三种方式赋值的：String y = env.y();、 y = a();、y = b;

//首先看第一种：String y = env.y();
public String y() {
    return c().getString("channel_id", null);
}
//去xml中取

//再看第二种y = a()
private final String a() {
    String i = com.bilibili.droid.p.i();
    kotlin.jvm.internal.x.h(i, "PackageManagerHelper.getChannel()");
    return i;
}

public static String i() {
    return com.bilibili.lib.foundation.d.i().d().getChannel();
}


public String getChannel() {
    f fVar = this.a;
    k kVar = k[0];
    return (String) fVar.getValue();
```

```
}

private final f a;
this.a = c3;

c3 = i.c(new kotlin.jvm.c.a<String>() { // from class:
com.bilibili.lib.foundation.DefaultApps$channel$2
    @Override // kotlin.jvm.c.a
    public final String invoke() {
        String str;
        d.b c13 = e.c();
        String apkPath = e.a().getApplicationInfo().sourceDir;
        try {
            x.h(apkPath, "apkPath");
            str = com.bilibili.lib.foundation.g.a.b.d(apkPath);
        } catch (IOException e) {
            c13.b(e);
            str = null;
        }
        boolean z = false;
        if (str == null || str.length() == 0) {
            try {
                x.h(apkPath, "apkPath");
                okio.c cVar =
com.bilibili.lib.foundation.g.a.b.e(apkPath).get(1903654775);
                if (cVar != null) {
                    Json nonstrict = Json.Companion.getNonstrict();
                    KSerializer map =
ShorthandsKt.getMap(m.a(ShorthandsKt.serializer(e0.a),
ShorthandsKt.serializer(e0.a)));
                    String p2 = cVar.p2();
                    x.h(p2, "buffer.readUtf8()");
                    str = (String) ((Map) nonstrict.parse(map,
p2)).get("channel");
                }
            } catch (IOException e2) {
                c13.b(e2);
            } catch (RuntimeException e3) {
                c13.b(e3);
            }
        }
        if ((str == null || str.length() == 0) ? true : true) {
            return "master";
        }
        if (str == null) {
            x.K();
            return str;
        }
        return str;
    }
});

public static <T> f<T> c(kotlin.jvm.c.a<? extends T> initializer) {
    kotlin.jvm.internal.x.q(initializer, "initializer");
    return new SynchronizedLazyImpl(initializer, null, 2, null);
```

```
}


public /* synthetic */ SynchronizedLazyImpl(kotlin.jvm.c.a aVar, Object obj, int
i, kotlin.jvm.internal.r rVar) {
    this(aVar, (i & 2) != 0 ? null : obj);
}

//i传过来的 是2，对2做位与的结果为1 传进去的是null 进而就执行了this(aVar, null);
//最终返回的是一个SynchronizedLazyImpl的对象，给了c3

//然后this.a就是c3的值也就是SynchronizedLazyImpl类的对象，然后把this.a赋值给了f fVar =
this.a;
//进而返回了return (String) fVar.getValue();
//所以我们要去看SynchronizedLazyImpl类的getValue()方法
public T getValue() {
    T t;
    T t2 = (T) this._value;
    if (t2 != t.a) {
        return t2;
    }
    synchronized (this.lock) {
        t = (T) this._value;
        if (t == t.a) {
            kotlin.jvm.c.a<? extends T> aVar = this.initializer;
            if (aVar == null) {
                kotlin.jvm.internal.x.K();
            }
            t = aVar.invoke();
            this._value = t;
            this.initializer = null;
        }
    }
    return t;
}
//在这个场景下，返回的就是固定的：xxl_gdt_wm_253
```

mobi_app

```
map.put("mobi_app", com.bilibili.api.a.l());

public static String l() {
    a();
    return b.A();
}

 public String A() {
     return "android";
 }
//固定值："android"
```

platform

```
map.put("platform", "android");
```

s_locale

```
map.put("s_locale", com.bilibili.api.a.n());

public static String n() {
    a();
    return b.C();
}

Override // com.bilibili.api.a.b
    public String C() {
    return tv.danmaku.bili.b0.l.a.a.b();
}

public final String b() {
     try {
         return c(tv.danmaku.bili.j0.a.a.a());
     } catch (Exception e) {
         BLog.efmt("bilow.locale", "Exception when get system locale " + e, new
Object[0]);
         return "";
     }
 }

private final String c(Locale locale) {
    boolean x1;
    String script;
    String str = "";
    String str2 = (locale == null || (str2 = locale.getLanguage()) == null) ? ""
: "";
    if (Build.VERSION.SDK_INT >= 21 && locale != null && (script =
locale.getScript()) != null) {
        str = script;
    }
    String country = locale != null ? locale.getCountry() : null;
    x1 = s.x1(str);
    if (!(!x1)) {
        return str2 + '_' + country;
    }
    return str2 + '-' + str + '_' + country;
}


//和c_local殊途同归
```

statistics

```
Map<String, String> i = com.bilibili.api.a.i();
if (i != null) {
    map.putAll(i);
}

//去看看com.bilibili.api.a.i()
public static Map<String, String> i() {
    return b.I();
}

//直接去看b接口的实现类的I方法
@Override // com.bilibili.api.a.b
public Map<String, String> I() {
    com.bilibili.app.comm.restrict.a.j(this.a);
    this.a.putAll(ps.a.c());
    return this.a;
}

private final HashMap<String, String> a = new HashMap<>(1);
//a是new了一个hashMap，容量为1
//然后调用p.a.c()把返回值放到a中

//p.a.c():
public final Map<? extends String, String> c() {
    HashMap hashMap = new HashMap();
    hashMap.put("statistics", new w0(1, 3, "6.24.0", ABTesting.b()).a());
    return hashMap;
}

//在put的时候value是一个对象，然后调用w0下的a方法，把这个对象转为了Json字符串
//{"appId":1,"platform":3,"version":"6.24.0","abtest":""}
//再看我们抓包得到的：appId是固定的1、platform是固定的3、version是固定的6.24.0、abtest是空

//所以这个statistics届就可以认为是固定的
```

在这个b方法中就是对以上参数进行了赋值，之前也分析过，t和sign是在其他函数中进行的，我们去看看

ts和sign

```
//在调用完b方法后，会调用h方法，然后把刚刚的hashMap传过去
h(hashMap).toString()

public SignedQuery h(Map<String, String> map) {
  return LibBili.g(map);
}
//在h方法中调用了LibBili.g(map)

public static SignedQuery g(Map<String, String> map) {
    return s(map == null ? new TreeMap() : new TreeMap(map));
}
```

```
//在g方法中，会把之前put进map的参数放到treeMap中，进行默认排序，然后调用s方法

static native SignedQuery s(SortedMap<String, String> sortedMap);
 //然后在native方法s中会添加一个ts，添加一个sign。添加完后还会根据key进行排序，然后返回一个SignedQuery对象，
//其中SignedQuery下的a属性为除sign之外的参数，b属性为sign
public final class SignedQuery {
    public final String a;
    public final String b;
    public SignedQuery(String str, String str2) {
        this.a = str;
        this.b = str2;
    }
    public String toString() {
        String str = this.a;
        if (str == null) {
            return "";
        }
        if (this.b == null) {
            return str;
        }
        return this.a + "&sign=" + this.b;
    }
}
//ts和sign是在native函数s中拿到的
static native SignedQuery s(SortedMap<String, String> sortedMap);
static {
    com.getkeepsafe.relinker.c.c("bili");
}
//在这个类中有加载libbili.so ，我们去找到这个so看一下是否是静态注册==>发现不是静态注册的。我们去 hook一下RegisterNatives
//搜索LibBili这个类，看看动态注册的地址
```

```
com.bilibili.nativelibrary.LibBili a (Ljava/lang/String;)Ljava/lang/String; 0xb6926c7d libbili.so 0x1c7d
com.bilibili.nativelibrary.LibBili ao (Ljava/lang/String;II)Ljava/lang/String; 0xb6926c83 libbili.so 0x1c83
com.bilibili.nativelibrary.LibBili b (Ljava/lang/String;)Ljavax/crypto/spec/IvParameterSpec; 0xb6926c91 libbili.so 0x1c91
com.bilibili.nativelibrary.LibBili s (Ljava/util/SortedMap;)Lcom/bilibili/nativelibrary/SignedQuery; 0xb6926c97 libbili.so 0x1c97
com.bilibili.nativelibrary.LibBili so (Ljava/util/SortedMap;II)Lcom/bilibili/nativelibrary/SignedQuery; 0xb6926c9d libbili.so 0x1c9d
com.bilibili.nativelibrary.LibBili so (Ljava/util/SortedMap;[B)Lcom/bilibili/nativelibrary/SignedQuery; 0xb6926cab libbili.so 0x1cab
com.bilibili.nativelibrary.LibBili getCpuCount ()I 0xb6926cb3 libbili.so 0x1cb3
com.bilibili.nativelibrary.LibBili getCpuId ()I 0xb6926cb7 libbili.so 0x1cb7
```

```
//就是在libbili.so  偏移为0x1c97的地方
int __fastcall sub_1C96(JNIEnv env, jclass cla, int map)
{
  return sub_2F88(env, map, 0, 0);
}

//可以看到，进来后又调用了sub_2F88(env, map, 0, 0)，把env、map、0、0 传进去了
//我们跟进sub_2F88(env, map, 0, 0)，进来后发现可读性一般，我们由果溯因
//先看return的是什么，在整个函数中有三个return：
return 0;
return (int)v12->NewObject(env, (jclass)dword_B0FC, (jmethodID)dword_B100, 0, 0);
return (int)(*env)->NewObject(env, (jclass)dword_B0FC, (jmethodID)dword_B100, v16, v17);
//可以看到第一个return 0 肯定不是我们要的结果，因为我们返回的不是0
```

//再看第二个，第三个。可以清晰的看到是进行了NewObject，这也和我们java层呼应上了。在java层，s方法返回的是SignedQuery对象，并且我们去看这个对象的构造方法时也发现它是两个参数的，和这里很吻合。
//并且可以倒推出dword_B0FC为SignedQuery类，dword_B100为init的方法id
//因为我们最终得到的SignedQuery的对象中的属性的值并不是0，所以就不可能走第二个return，因为第二个NewObject时传过去的参数是0

//综上所述，最终是通过return (int)(*env)->NewObject(env, (jclass)dword_B0FC, (jmethodID)dword_B100, v16, v17);来进行返回的。因此可以推出，v16，v17是SignedQuery的两个属性，即v16是a，v17是b。 再看SignedQuery的toString方法，可以发现b就是sign，a就是除了sign之外的其他参数。 回到sub_2F88(env, map, 0, 0)中，我们去看v17怎么来的：

```
    for ( j = 0; j != 16; ++j )
    {
      sprintf(v29, "%02x", (unsigned __int8)v37[j]);
      v29 += 2;
    }
    free(v20);
    v16 = v31;
    v17 = (*env)->NewStringUTF(env, s);
  }
  else
  {
    v17 = 0;
  }
}
sub_45C8(env, v34);
return (int)(*env)->NewObject(env, (jclass)dword_B0FC, (jmethodID)dword_B100, v16, v17);
}
```

//v17是sign，我们知道它不为0，所以v17是通过v17 = (*env)->NewStringUTF(env, s);得到的，那么我们的目标就变成看s是如何来的，因为s就是v17也就是sign

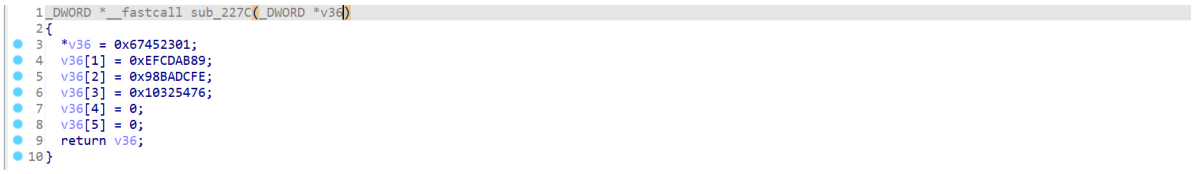
```
    memset(s, 0, 33u);
    v27 = strlen(v32);
    memset(v37, 0, sizeof(v37));
    memset(v36, 0, sizeof(v36));
    sub_227C(v36);
    sub_22B0(v36, v32, v27);
    sprintf(v37, "%08x", v22);
    sub_22B0(v36, v37, 8);
    for ( i = 1; i != 4; ++i )
    {
      sprintf(v37, "%08x", v20[i]);
      sub_22B0(v36, v37, 8);
    }
    sub_2AE0(v37, v36);
    v29 = s;
    for ( j = 0; j != 16; ++j )
    {
      sprintf(v29, "%02x", (unsigned __int8)v37[j]);
      v29 += 2;
    }
    free(v20);
    v16 = v31;
    v17 = (*env)->NewStringUTF(env, s);
  }
  else
  {
    v17 = 0;
  }
}
```

//首先把s指向的内存的前33位清零了
//然后让v29指向了s的首地址
//接着进入for循环，把v37进行了hex编码，放进v29指向的内存了，也就是s指向的内存

//因此我们继续更换目标，换为v37.找到v37怎么来的，进行hex编码后就是s，然后转为jstr。下边追v37

```
  v22 = v21[v19];
  v23 = &v21[v19];
  v24 = v23[7];
  v25 = v23[14];
  v26 = v23[21];
  *v20 = v22;
  v20[1] = v24;
  v20[2] = v25;
  v20[3] = v26;
  memset(s, 0, 33u);
  v27 = strlen(v32);
  memset(v37, 0, sizeof(v37));
  memset(v36, 0, sizeof(v36));
  sub_227C(v36);
  sub_22B0(v36, v32, v27);
  sprintf(v37, "%08x", v22);
  sub_22B0(v36, v37, 8);
  for ( i = 1; i != 4; ++i )
  {
    sprintf(v37, "%08x", v20[i]);
    sub_22B0(v36, v37, 8);
  }
  sub_2AE0(v37, v36);
  v29 = s;
  for ( j = 0; j != 16; ++j )
  {
    sprintf(v29, "%02x", (unsigned __int8)v37[j]);
    v29 += 2;
  }
  free(v20);
  v16 = v31;
  v17 = (*env)->NewStringUTF(env, s);
 }
 else
 {
  v17 = 0;
 }
}
00003178 sub_2F88:112 (3178)
```

```
//首先把v37、v36进行清零。这是两个数组char v36[88];char v37[24];
//清零后先进行了sub_227C(v36); 这一步应该是往v36中放了东西，我们去sub_227C(v36)中看看
```



```
1 _DWORD *__fastcall sub_227C(_DWORD *v36)
2 {
3   *v36 = 0x67452301;
4   v36[1] = 0xEFCDAB89;
5   v36[2] = 0x98BADCFE;
6   v36[3] = 0x10325476;
7   v36[4] = 0;
8   v36[5] = 0;
9   return v36;
10 }
```

```
//进来后发现是MD5的初始化常量，所以v36应该是MD5
//调用完sub_227C(v36)后接着调用了 sub_22B0(v36, v32, v27);
v27 = strlen(v32);
memset(v37, 0, sizeof(v37));
memset(v36, 0, sizeof(v36));
sub_227C(v36);

sub_22B0(v36, v32, v27);
//调用sub_22B0时传进去的三个参数v36, v32, v27。
//其中v36是MD5初始化常量、v32应该是明文，因为v27是对v32取的长度、v27是v32的长度
//sub_22B0(v36, v32, v27)是MD5的update update到v36中了
//完事之后继续向下执行：
sprintf(v37, "%08x", v22);
sub_22B0(v36, v37, 8);
//这里是把v22转为了长度为8位的16进制，若不够8位，用0补齐，然后放到v37中。
//然后再把v37 update进v36中
//接着进行了一个for循环：
for ( i = 1; i != 4; ++i )
{
    sprintf(v37, "%08x", v20[i]);
    sub_22B0(v36, v37, 8);
}
//是把v20[1]、v20[2]、v20[3]都转为长度为8位的16进制，然后update进去
//往前看发现：
*v20 = v22;
//也就是说 for循环+之前的一次操作就是把v20这个数组中的东西转为16进制然后update进去
//接着调用了sub_2AE0(v37, v36);  v36是刚刚update的明文以及MD5初始化常量，v37是存放MD5的结果的。sub_2AE0(v37, v36)就是digest
```

```
//然后就和最开始分析的呼应了。把v37进行hex编码放到了v29中，v29和s是指向同一片内存区域，最终把s转为jstr，然后进行NewObject
```

```
//到这里为止：我们需要解决两个事情：1、每次update的是什么，这样也就能搞清楚明文了 2、sub_2AE0(v37, v36)是否为标准的MD5
//所以我们同时去hook一下 update和digest函数，也就是sub_22B0(v36, v37, 8)和sub_2AE0(v37, v36)
```

```
hook sub_2AE0(v37, v36)
OnEnter:
v37：基本为空
v36：一大堆


OnLeave:
v37:
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
cbdf1380  fd 76 39 37 fb 7e 9c 0f 80 e9 97 d7 ac ac d6 83  .v97.~..........
v36:
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
cbdf1328  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
最终sign的值：fd763937fb7e9c0f80e997d7acacd683
```

```
hook sub_22B0(v36, v37, len)
和我们之前分析的一样，一共执行5次，第一次是放一些明文，剩余四次是放v20数组里的东西转8位16进制放进去

第一次：
sub_22B0 args[0] v36 is :
sub_22B0 args[1] v37 is : appkey=1d8b6e7d45233436&build=6240300&c_locale=zh-Hans_CN&channel=xxl_gdt_wm_253&mobi_app=xxl_gdt_wm_253&platform=android&s_locale=zh-Hans_CN&statistics=%7B%22appId%22%3A1%2C%22platform%22%3A3%2C%22version%22%3A%226.24.0%22%2C%22abtest%22%3A%22%22%7D&ts=1747551688
sub_22B0 args[2] length is : 0x108
第二次：
sub_22B0 args[0] v36 is :
sub_22B0 args[1] v37 is : 560c52cc
sub_22B0 args[2] length is : 0x8
第三次：
sub_22B0 args[0] v36 is :
sub_22B0 args[1] v37 is : d288fed0
sub_22B0 args[2] length is : 0x8
第四次：
sub_22B0 args[0] v36 is :
sub_22B0 args[1] v37 is : 45859ed1
sub_22B0 args[2] length is : 0x8
第五次：
sub_22B0 args[0] v36 is :
sub_22B0 args[1] v37 is : 8bffd973
sub_22B0 args[2] length is : 0x8
```

可以看到第一次放进去的是就是我们在java层传进来的treemap，但是他这里取出来所有元素进行key=value&key=value..这样拼接了。但是它取了一个秒级时间戳，秒级时间戳并不是直接拼到最后边的，它是先取了时间戳放进treemap中后再进行key=value&key=value..的。因为在其他接口的时候，会有参数在ts后边，就是因为它拿到ts后放进map中进行了排序，然后拼接起来的。

我们仔细看，它还进行了url编码！

然后在v20数组中取到的数据就是盐：560c52ccd288fed045859ed18bffd973，把盐拼到后边，然后进行了MD5，经过验证为标准MD5

ts、sign验证：
我们主动调用一下java层的s方法，在原有的map的基础上，再加一组数组，这组数组的key要比ts排名靠后：user_status   0

主动调用脚本：

```
function invoke_s() {
    Java.perform(function () {
        let LibBili = Java.use("com.bilibili.nativelibrary.LibBili");
        var TreeMap = Java.use('java.util.TreeMap');
        var treeMap = TreeMap.$new();
        treeMap.put('appkey', '1d8b6e7d45233436');
        treeMap.put('build', '6240300');
        treeMap.put('c_locale', 'zh-Hans_CN');
        treeMap.put('channel', 'xxl_gdt_wm_253');
        treeMap.put('mobi_app', 'xxl_gdt_wm_253');
        treeMap.put('platform', 'android');
        treeMap.put('platform', 'android');
        treeMap.put('user_status', '0');
        treeMap.put('s_locale', 'zh-Hans_CN');
        treeMap.put('statistics',
'{"appId":1,"platform":3,"version":"6.24.0","abtest":""}');
        var res = LibBili.s(treeMap)
        console.log("invoke s res is ==>",res)
    })
}
```

hook sub_22B0打印每次update的数据，同时hook sub_2AE0打印MD5的密文：

```
hook sub_22B0:
sub_22B0 args[0] v36 is :
sub_22B0 args[1] v37 is : appkey=1d8b6e7d45233436&build=6240300&c_locale=zh-
Hans_CN&channel=xxl_gdt_wm_253&mobi_app=xxl_gdt_wm_253&platform=android&s_locale
=zh-
Hans_CN&statistics=%7B%22appId%22%3A1%2C%22platform%22%3A3%2C%22version%22%3A%22
6.24.0%22%2C%22abtest%22%3A%22%22%7D&ts=1747552257&user_status=0
sub_22B0 args[2] length is : 0x116
sub_22B0 args[0] v36 is :
sub_22B0 args[1] v37 is : 560c52cc
sub_22B0 args[2] length is : 0x8
sub_22B0 args[0] v36 is :
sub_22B0 args[1] v37 is : d288fed0
sub_22B0 args[2] length is : 0x8
sub_22B0 args[0] v36 is :
sub_22B0 args[1] v37 is : 45859ed1
sub_22B0 args[2] length is : 0x8
```

```
sub_22B0 args[0] v36 is :
sub_22B0 args[1] v37 is : 8bffd973
sub_22B0 args[2] length is : 0x8

hook sub_2AE0:
           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
cbef2380  7f 4e d5 b8 1b de ad 7e 40 1f bb 15 9c f2 af 13  .N.....~@.......
```

主动调用s打印的返回值：

```
appkey=1d8b6e7d45233436&build=6240300&c_locale=zh-
Hans_CN&channel=xxl_gdt_wm_253&mobi_app=xxl_gdt_wm_253&platform=android&s_locale
=zh-
Hans_CN&statistics=%7B%22appId%22%3A1%2C%22platform%22%3A3%2C%22version%22%3A%22
6.24.0%22%2C%22abtest%22%3A%22%22%7D&ts=1747552257&user_status=0&sign=7f4ed5b81b
dead7e401fbb159cf2af13
```

经过上边验证可以确定是先拿到秒级时间戳ts然后丢进treemap中后，再进行key=value&key=value..拼接的。

ts和sign总结：
首先通过java层传过来的是一个treemap，到so层后会先拿一个秒级时间戳put进去
之后把treemap进行key=value&key=value..拼接
拼接之后进行url编码
然后加盐：560c52ccd288fed045859ed18bffd973
之后进行md5加密转hex编码即可

session_id

现在去看看"session_id"

```
String m2 = com.bilibili.api.a.m();
if (TextUtils.isEmpty(m2)) {
    return;
}
aVar.h("session_id", m2);

//去看看m2 com.bilibili.api.a.m():
@NonNull
public static String m() {
    a();
    return b.getSessionId();
}

//是通过b.getSessionId()拿到的
//但是b是一个接口，getSessionId()是接口里的一个方法
//所以要去看看给b赋值的时候是赋的哪个对象，那个对象所属类的在定义的时候一定是实现了b接口，进而也会重写getSessionId()方法
private static b b;
public static void o(b bVar) {
```

```
    b = bVar;
}
//首先是定义了一个接口b，然后在o方法中，给b赋值的，我们hook一下o，打印一下调用栈，看看在调用的时候传的是哪个类的对象
[Pixel::tv.danmaku.bili ]->
a.o is called: bVar=[object Object]
java.lang.Throwable
        at com.bilibili.api.a.o(Native Method)
        at tv.danmaku.bili.utils.p.b(BL:1)
        at tv.danmaku.bili.proc.MainBiliAppProc.v(BL:2)
        at tv.danmaku.bili.proc.k.invoke(Unknown Source:2)
        at tv.danmaku.bili.delaytask.a.d(BL:5)
//去看看tv.danmaku.bili.utils.p.b调用o方法时传过来的对象是什么类的
public static final void b(Application app, boolean z) {
        kotlin.jvm.internal.x.q(app, "app");
        com.bilibili.api.a.o(new a(app));
    }
//这里是new了一个a类的对象new a(app)
//去看看这个a类里边如何实现的getSessionId()
public static final class a implements a.b {
    @Override // com.bilibili.api.a.b
    public String getSessionId() {
        return com.bilibili.lib.foundation.e.b().getSessionId();
    }
}
//是通过com.bilibili.lib.foundation.e.b()的返回值，再去调用getSessionId()方法
//com.bilibili.lib.foundation.e.b():
public static final a b() {
    return d.g.b().d();
}
//返回的是d.g.b().d()
public final com.bilibili.lib.foundation.a d() {
    return this.a;
}
//又返回了this.a
private final com.bilibili.lib.foundation.a a;
//点进去看到这个a也是一个接口，所以我们要去看在哪里赋值的、赋值的是哪个类的对象
this.a = new DefaultApps(this.e.a());
//是在构造器中进行赋值的，赋值了一个DefaultApps类的对象，
//所以说com.bilibili.lib.foundation.e.b()返回的就是DefaultApps类的对象
//com.bilibili.lib.foundation.e.b().getSessionId()就等同于DefaultApps类的对象.getSessionId()
//我们去看下在DefaultApps类中，是如何实现getSessionId()方法的
public final class DefaultApps implements com.bilibili.lib.foundation.a {
    @Override // com.bilibili.lib.foundation.a
    public String getSessionId() {
        String string = e.e().getString("foundation:session_id", 1);
        if (string == null) {
            x.K();
        }
        return string;
    }
}
//首先是去xml中去取"foundation:session_id"的值，如果没取到就用默认值1
//若是string空，就报错
```

```
//所以重点是l
public final class DefaultApps implements com.bilibili.lib.foundation.a {
private static final String l;
static {
        byte[] bArr = new byte[4];
        new Random().nextBytes(bArr);
        String hex = ByteString.of(bArr, 0, 4).hex();
        x.h(hex, "ByteString.of(bytes, 0, bytes.size).hex()");
        l = hex;
    }
}
//首先是定义了一个私有变量 l
//接着在静态代码块中为l进行了赋值
//是吧hex的值赋值给了l
//我们逐一分析下hex的生成过程
byte[] bArr = new byte[4];
new Random().nextBytes(bArr);
//先new一个长度为4的byte数组
//把随机生成的数据填满这个byte数组
//这两步执行完，bArr中就会填满随机生成的数据
String hex = ByteString.of(bArr, 0, 4).hex();
//然后调用 ByteString.of(bArr, 0, 4)会把bArr中的数据cope一份，然后再new一个ByteString
//再调用ByteString下的hex方法，把随机生成的这个bArr进行hex编码(hex方法是自写的hex编码)
//ByteString.of(bArr, 0, 4):
public static ByteString of(byte[] bArr, int i, int i2) {
    if (bArr != null) {
        w.b(bArr.length, i, i2);
        byte[] bArr2 = new byte[i2];
        System.arraycopy(bArr, i, bArr2, 0, i2);
        return new ByteString(bArr2);
    }
    throw new IllegalArgumentException("data == null");
}
//首先判断是否为空、w.b判断是否为空、然后new一个新的byte[]、把传过来的bArr cope到新new的数组中、然后再new成ByteString返回
//这里new ByteString(bArr2) 是把bArr2给了 ByteString类下的byte数组常量 data
//接着去调用ByteString下的hex方法，进行hex编码，hex方法就是对ByteString下的byte数组常量进行hex编码
//hex方法:
public String hex() {
    byte[] bArr = this.data;
    char[] cArr = new char[bArr.length * 2];
    int i = 0;
    for (byte b : bArr) {
        int i2 = i + 1;
        char[] cArr2 = HEX_DIGITS;
        cArr[i] = cArr2[(b >> 4) & 15];
        i = i2 + 1;
        cArr[i2] = cArr2[b & 15];
    }
    return new String(cArr);
}
//这里HEX_DIGITS是hex的码表
```

到此session_id就解决了，就是生成一个长度为4的随机字节数组，并用随机数据填充。然后进行hex编码

完播率接口

抓包分析

打开视频的时候会发送一个请求，视频播放结束了再发一个请求，以此记录完播率

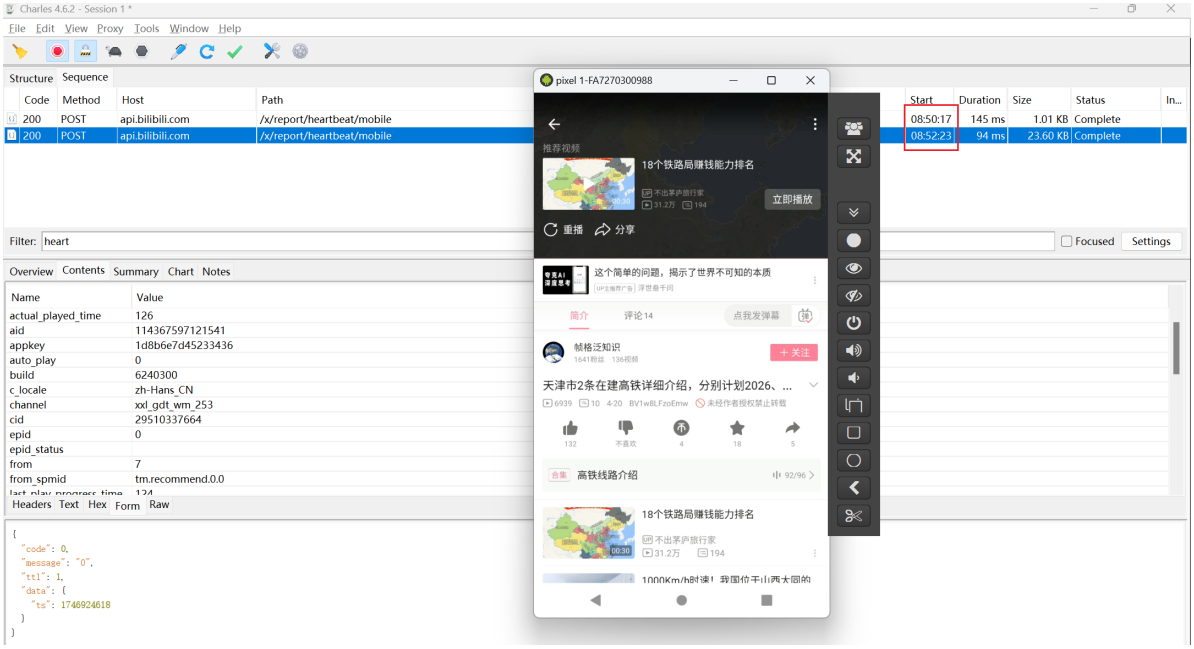
请求网址:https://api.bilibili.com/x/report/heartbeat/mobile
请求方式：POST
请求头：

```
    content-length  777
    buvid   XY2CAAD2CD8168403CCB095E408A467773352
    device-id   bg0-CTOOb1ZvDTxaJlom
    fp_local    efbaa891c759dbcacebbf89bb3517b7f20250509194124301343fe74b3e982a8
    fp_remote   efbaa891c759dbcacebbf89bb3517b7f20241207150740ab3e564147129a8976
    session_id  a9e7d2eb
    env prod
    app-key android
    user-agent  Mozilla/5.0 BiliDroid/6.24.0 (bbcallen@gmail.com) os/android
model/Pixel mobi_app/android build/6240300 channel/xxl_gdt_wm_253
innerVer/6240300 osVer/10 network/2
    bili-bridge-engine  cronet
    content-type    application/x-www-form-urlencoded; charset=utf-8
    accept-encoding gzip, deflate, br
```

请求体：

```
    actual_played_time  126
    aid 114367597121541
    appkey  1d8b6e7d45233436
    auto_play   0
    build   6240300
    c_locale    zh-Hans_CN
    channel xxl_gdt_wm_253
    cid 29510337664
    epid    0
    epid_status
    from    7
    from_spmid  tm.recommend.0.0
    last_play_progress_time 124
    list_play_time  0
    max_play_progress_time  124
    mid 0
    miniplayer_play_time    0
    mobi_app    android
    network_type    1
    paused_time 0
    platform    android
    play_status 0
    play_type   1
    played_time 126
    quality 32
```

```
s_locale    zh-Hans_CN
session 50734e566f37523b73bd464c8a23ed7f36cc71b4
sid 0
spmid   main.ugc-video-detail.0.0
start_ts    1746924618
statistics  {"appId":1,"platform":3,"version":"6.24.0","abtest":""}
sub_type    0
total_time  126
ts  1746924710
type    3
user_status 0
video_duration  124
sign    e2dfbd16f818e2b3e940ab4561737c1f
```



请求头

```
content-length  777
buvid   XY2CAAD2CD8168403CCB095E408A467773352
device-id   bg0-CTOOb1ZvDTxaJlom
fp_local    efbaa891c759dbcacebbf89bb3517b7f20250509194124301343fe74b3e982a8
fp_remote   efbaa891c759dbcacebbf89bb3517b7f20241207150740ab3e564147129a8976
session_id  a9e7d2eb
env prod
app-key android
user-agent  Mozilla/5.0 BiliDroid/6.24.0 (bbcallen@gmail.com) os/android
model/Pixel mobi_app/android build/6240300 channel/xxl_gdt_wm_253
innerVer/6240300 osVer/10 network/2
bili-bridge-engine  cronet
content-type    application/x-www-form-urlencoded; charset=utf-8
accept-encoding gzip, deflate, br
```

```
//先去搜索请求url：
@FormUrlEncoded
@POST("/x/report/heartbeat/mobile")
```

```
com.bilibili.okretro.d.a<GeneralResponse<String>> reportV2(@FieldMap
HeartbeatParams heartbeatParams);

//我们看到，他并没有在这里添加请求头。它是在公共拦截器中设置的，在new retrofit对象的时
候.clinet(公共拦截器)，我们找到拦截器的interceptor方法看看，它是被混淆过的，a方法酒输
public bO a(bO bOVar) {
    bO.a h = bOVar.h();
    e(h);
    if ("GET".equals(bOVar.g())) {
        d(bOVar.k(), h);
    } else if ("POST".equals(bOVar.g())) {
        c(bOVar.k(), bOVar.a(), h);
    }
    return h.b();
}

//可以看到，无论是get请求还是post请求，进来后都要先执行 e(h);
//e方法里边是添加请求头的，h是chain.request()得到的
//因此请求头不需要再进行分析了，和之前的殊途同归
```

请求体

```
actual_played_time  126
    aid 114367597121541
    appkey  1d8b6e7d45233436
    auto_play   0
    build   6240300
    c_locale    zh-Hans_CN
    channel xxl_gdt_wm_253
    cid 29510337664
    epid    0
    epid_status
    from    7
    from_spmid  tm.recommend.0.0
    last_play_progress_time 124
    list_play_time  0
    max_play_progress_time  124
    mid 0
    miniplayer_play_time    0
    mobi_app    android
    network_type    1
    paused_time 0
    platform    android
    play_status 0
    play_type   1
    played_time 126
    quality 32
    s_locale    zh-Hans_CN
    session 50734e566f37523b73bd464c8a23ed7f36cc71b4
    sid 0
    spmid   main.ugc-video-detail.0.0
    start_ts    1746924618
    statistics  {"appId":1,"platform":3,"version":"6.24.0","abtest":""}
```

```
    sub_type    0
    total_time  126
    ts  1746924710
    type    3
    user_status 0
    video_duration  124
    sign    e2dfbd16f818e2b3e940ab4561737c1f
```

```
//我们先去搜索请求链接：
//搜索的时候找到了两个相同的接口，位于不同的包下，但是其他的都一样！
@FormUrlEncoded
@POST("/x/report/heartbeat/mobile")
com.bilibili.okretro.d.a<GeneralResponse<String>> reportV2(@FieldMap
HeartbeatParams heartbeatParams);


@FormUrlEncoded
@POST("/x/report/heartbeat/mobile")
com.bilibili.okretro.d.a<GeneralResponse<String>> reportV2(@FieldMap
HeartbeatParams heartbeatParams);

//分别去找调用reportV2的实例，然后同时hook这三个方法，再触发心跳函数，看哪个方法被调用了，最终发现是走的这个类下的：package tv.danmaku.biliplayerimpl.report.heartbeat;

//看看它的用例：
private final void O7() {调用 reportV2}
private final void P7() {调用 reportV2}

//经过hook得知，再打开的视频时在P7方法中调用的reportV2，视频播放完的时候在O7中调用的reportV2
//所以我们要去分析P7和O7中参数的构成，这样才能完成这两次请求，成功的增加完播率
```

```
//先分别看看P7和O7

//P7：
private final void P7() {
    h hVar = this.f24360c;
    if (hVar != null) {
        if (hVar == null) {x.K();}
        if (hVar.v1() == 0) { return;}
        h hVar2 = this.f24360c;if (hVar2 == null) { x.K(); }

        HeartbeatParams N7 = N7(hVar2, true);

        h hVar3 = this.f24360c;
        if (hVar3 == null) {x.K();}
        ((tv.danmaku.biliplayerimpl.report.heartbeat.a)
com.bilibili.okretro.c.a(tv.danmaku.biliplayerimpl.report.heartbeat.a.class)).re
portV2(N7).y(new f(hVar3.r1()));
    }
}

//O7：
```

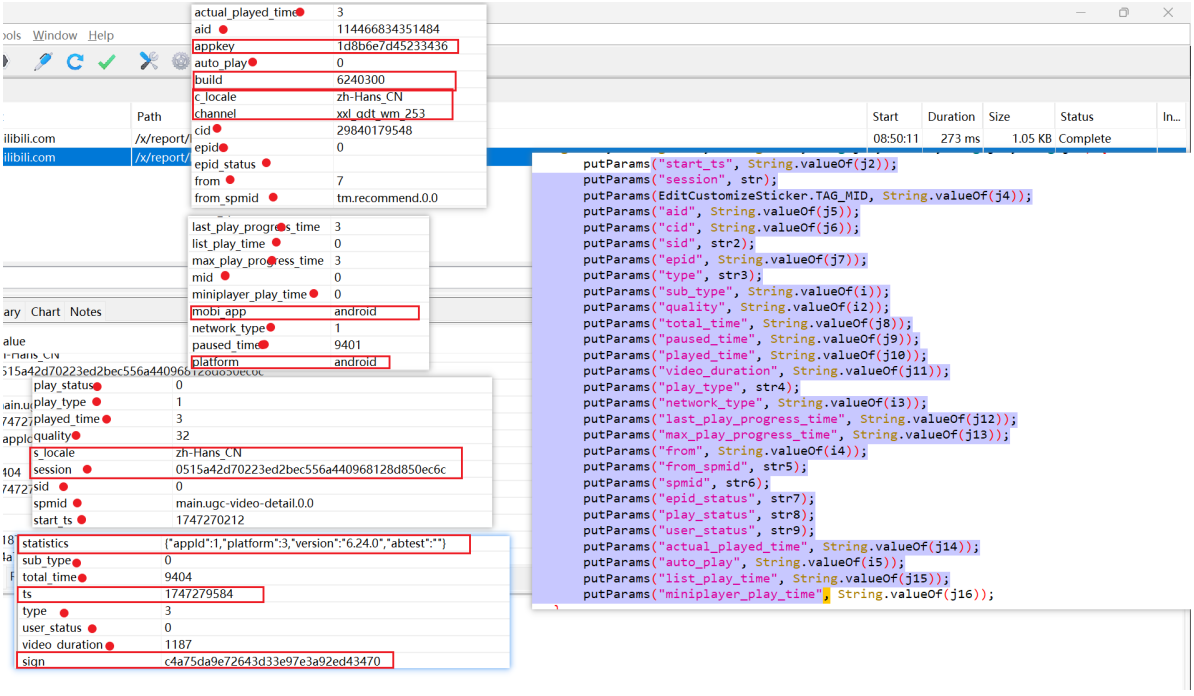
```
private final void O7() {
    h hVar = this.f24360c;
    if (hVar == null) {return;}
    if (hVar == null) {x.K();}
    if (hVar.v1() == 0) {
        h hVar2 = this.f24360c;
        if (hVar2 == null) {x.K();}
        hVar2.y2(f24359j.b());
    }
    h hVar3 = this.f24360c;
    if (hVar3 == null) {x.K();}

    HeartbeatParams N7 = N7(hVar3, false);

    h hVar4 = this.f24360c;
    if (hVar4 == null) {
        x.K();
    }
    ((tv.danmaku.biliplayerimpl.report.heartbeat.a)
com.bilibili.okretro.c.a(tv.danmaku.biliplayerimpl.report.heartbeat.a.class)).re
portV2(N7).y(new e(hVar4.E0()));
    this.f24360c = null;
}



//对比发现，他们调用reportV2的参数都是N7对象，N7又都是通过调用N7((h hVar, boolean z)得到
的，在N7里边又是调用了 new HeartbeatParams(...)，只是在调用N7的时候传的参数不一同，我们分别
去看看他们的参数


//在P7中：
HeartbeatParams N7 = N7(hVar2, true);
//传过去的是hVar2和true
private final HeartbeatParams N7(h hVar, boolean z) {
    return new HeartbeatParams(hVar.p1(), hVar.r1(), hVar.a1(), hVar.H0(),
hVar.J0(), hVar.s1(), hVar.M0(), hVar.C1(), hVar.w1(), hVar.o1(), z ? 0L :
hVar.B1(), z ? 0L : hVar.e1(), z ? 0L : hVar.k1(), hVar.E1(), hVar.i1(),
hVar.c1(), z ? 0L : hVar.V0(), z ? 0L : hVar.Y0(), hVar.S0(), hVar.N0(),
hVar.t1(), hVar.L0(), hVar.f1(), hVar.D1(), z ? 0L : hVar.G0(), hVar.I0(), z ?
0L : hVar.W0(), z ? 0L : hVar.b1());
}



//在O7中：
HeartbeatParams N7 = N7(hVar3, false);
//传过去的是hVar3和true
private final HeartbeatParams N7(h hVar, boolean z) {
    return new HeartbeatParams(hVar.p1(), hVar.r1(), hVar.a1(), hVar.H0(),
hVar.J0(), hVar.s1(), hVar.M0(), hVar.C1(), hVar.w1(), hVar.o1(), z ? 0L :
hVar.B1(), z ? 0L : hVar.e1(), z ? 0L : hVar.k1(), hVar.E1(), hVar.i1(),
hVar.c1(), z ? 0L : hVar.V0(), z ? 0L : hVar.Y0(), hVar.S0(), hVar.N0(),
hVar.t1(), hVar.L0(), hVar.f1(), hVar.D1(), z ? 0L : hVar.G0(), hVar.I0(), z ?
0L : hVar.W0(), z ? 0L : hVar.b1());
```

```
}


//对比发现，参数获取的方法是一模一样的，有的是根据z来获取，有的是通过调用对象的方法获取。
//因此我们在分析的过程中，可以直接分析打开视频和播放完视频时发送请求的参数
```

接下来去看HeartbeatParams的构造方法:

```
public final class HeartbeatParams extends ParamsMap {
    public HeartbeatParams(long j2, String str, long j4, long j5, long j6, String str2, long j7, String str3, int i, int
i2, long j8, long j9, long j10, long j11, String str4, int i3, long j12, long j13, int i4, String str5, String str6,
String str7, String str8, String str9, long j14, int i5, long j15, long j16) {
        putParams("start_ts", String.valueOf(j2));
        putParams("session", str);
        putParams(EditCustomizeSticker.TAG_MID, String.valueOf(j4));
        putParams("aid", String.valueOf(j5));
        putParams("cid", String.valueOf(j6));
        putParams("sid", str2);
        putParams("epid", String.valueOf(j7));
        putParams("type", str3);
        putParams("sub_type", String.valueOf(i));
        putParams("quality", String.valueOf(i2));
        putParams("total_time", String.valueOf(j8));
        putParams("paused_time", String.valueOf(j9));
        putParams("played_time", String.valueOf(j10));
        putParams("video_duration", String.valueOf(j11));
        putParams("play_type", str4);
        putParams("network_type", String.valueOf(i3));
        putParams("last_play_progress_time", String.valueOf(j12));
        putParams("max_play_progress_time", String.valueOf(j13));
        putParams("from", String.valueOf(i4));
        putParams("from_spmid", str5);
        putParams("spmid", str6);
        putParams("epid_status", str7);
        putParams("play_status", str8);
        putParams("user_status", str9);
        putParams("actual_played_time", String.valueOf(j14));
        putParams("auto_play", String.valueOf(i5));
        putParams("list_play_time", String.valueOf(j15));
        putParams("miniplayer_play_time", String.valueOf(j16));
    }
```

```
//可以看到在这个HeartbeatParams的构造方法中，调用putParams()方法，往这个对象中放了好多数据，这这些数据都是我们请求体中的
//但是我拿这些数据去和抓包得到数据对比，发现少了一些数据
```



```
//对比发现：appkey、build、c_locale、channel、mobi_app、platform、s_locale、
statistics、ts、sign
//这个参数并没有在HeartbeatParams的构造方法中创建，因为这几个参数是在拦截器中构建的
public b0 a(b0 b0Var) {
    b0.a h = b0Var.h();
    e(h);
    if ("GET".equals(b0Var.g())) {
        d(b0Var.k(), h);
    } else if ("POST".equals(b0Var.g())) {
        c(b0Var.k(), b0Var.a(), h);
    }
    return h.b();
}

//可以看到如果是POST请求的话，会调用一个c方法： c(b0Var.k(), b0Var.a(), h)
//我们去看下这个方法:
public void c(u uVar, c0 c0Var, b0.a aVar) {
    if (c0Var instanceof x) {
        return;
    }
    try {
        if (!(c0Var instanceof r)) {
            if (c0Var.contentLength() > 0) {
                return;
            }
        }
        HashMap hashMap = new HashMap();
        if (c0Var instanceof r) {
            r rVar = (r) c0Var;
            int size = rVar.size();
            for (int i = 0; i < size; i++) {
                g(rVar.d(i), rVar.f(i), hashMap);
            }
        }
        int L = uVar.L();
        for (int i2 = 0; i2 < L; i2++) {
            g(uVar.H(i2), uVar.J(i2), hashMap);
        }
        b(hashMap);
        u h = uVar.s().A(null).h();
        aVar.s(h).l(c0.create(w.d("application/x-www-form-urlencoded;
charset=utf-8"), h(hashMap).toString()));
    } catch (IOException unused) {
    }
}

// aVar.s(h).l(c0.create(w.d("application/x-www-form-urlencoded; charset=utf-8"),
h(hashMap).toString()));
//这里是添加请求体的逻辑： h(hashMap).toString()就是请求体
```

接下来开始分析请求体的参数

拦截其中的请求参数

appkey、build、c_locale、channel、mobi_app、platform、s_locale、statistics、ts、sign
直接看fp_remote中的请求体即可，都是在拦截器中进行操作的！

```
//我们接着看HeartbeatParams的构造方法看看这些参数怎么来的
```

aid和cid

这个是视频的标识

session

```
//在P7中，即打开视频发送请求时：

putParams("session", str);
//str是调用new HeartbeatParams()的时候传过来的，第二个参数
hVar.r1()
//通过调用 hVar.r1()得到的：
private String d;
public final String r1() {
    return this.d;
}

//去看看d是在哪里赋值的：
public final void t2(String str) {
    this.d = str;
}

//是调用t2的时候传过来的str给了d
//看看谁调用的t2
hVar.t2(g.a.a());

//g.a.a()的返回值就是session
public final String a() {
    Random random = new Random();
    StringBuilder sb = new StringBuilder();
    c2.f.b0.c.b.b.a.a E = c2.f.b0.c.b.b.a.a.E();
    x.h(E, "EnvironmentPrefHelper.getInstance()");
    sb.append(E.t());
    sb.append(System.currentTimeMillis());
    sb.append(random.nextInt(1000000));
    String sb2 = sb.toString();
    String sha1 = com.bilibili.commons.m.a.i(sb2);
    if (TextUtils.isEmpty(sha1)) {
        return sb2;
    }
    x.h(sha1, "sha1");
    return sha1;
}
```

```
//大体逻辑为 首先new一个StringBuilder 然后append三个数据，
//分别是：E.t()、System.currentTimeMillis()、random.nextInt(1000000)
//然后转为 String类的 sb2，然后调用com.bilibili.commons.m.a.i(sb2) 把结果返回

//先看append的东西
//E.t()
public String t() {
    return c().getString("buvid", "");
}
//去xml中拿buvid的值，若拿不到就拿空

//System.currentTimeMillis()
//获取毫秒级时间戳

//random.nextInt(1000000)
//随机生成一个[0,1000000)的数字

//到这里String类的sb2就出来了==>buvid+毫秒级时间戳+[0,1000000)的随机数


//接着去看com.bilibili.commons.m.a.i(sb2)，看看对sb2做了什么：
private static String h(byte[] bArr, MessageDigest messageDigest) {
    messageDigest.update(bArr);
    byte[] digest = messageDigest.digest();
    char[] cArr = new char[digest.length * 2];
    int i = 0;
    for (byte b : digest) {
        int i2 = i + 1;
        char[] cArr2 = a;
        cArr[i] = cArr2[(b >>> 4) & 15];
        i = i2 + 1;
        cArr[i2] = cArr2[b & cv.f24887m];
    }
    return new String(cArr).toLowerCase();
}
//messageDigest传过来的是sha1的实例
//bArr是sb2.getBytes()
//最终进行hex编码，然后转为小写

//我们去hook一下com.bilibili.commons.m.a.i(sb2)看看参数和返回值，验证一下分析的正确不正确
a.i is called: str=1747307815513998860
a.i result=0f2369c3e5dc44bfc61e6eda1d1ad781c97b6aaa

//和我们抓包得到的是一样的，这样看明文就是只有毫秒级时间戳+随机数，buvid返回的是空
//1747307815513    998860

//我们验证一下是不是sha1加密==>sha1加密
//到这里为止，打开视频时的session就解决了==>buvid或空+毫秒级时间戳+[0,1000000)的随机数 拼起来再进行sha1加密
```

```
//在O7中，即退出视频发送请求时：
//也是同样的流程，我们去抓包看看，在O7和P7中是否用的同一个session
//经过验证，同一个视频播放与退出或者完成播放时发送请求用的是同一个session，但是第二次打开这个视频就会生成一个新的session
```

```
//session总结
//1.打开视频和结束视频发送请求用的同一个session
//2.第二次打开视频时会生成新的session
//3.session的生成规则：buvid或空+毫秒级时间戳+[0,1000000)的随机数 拼起来再进行sha1加密
==>hex编码==>小写
```

auto_play

```
//在P7中：
putParams("auto_play", String.valueOf(i5));
//i5是上一层传过来的，是倒数第三个参数

hVar.IO();
private int B;
public final int IO() {
    return this.B;
}

public final void L1(int i) {
    this.B = i;
}
this.B = commonParams.e();
//this.B就是auto_play的值
//this.B有两种获取方式：调用L1方法、调用commonParams.e()

//先看L1：
public final void L1(int i) {
    this.B = i;
}

//看看是谁调用的L1，参数就是auto_play
hVar.L1(commonParams.e());

//我们发现最终还是调用commonParams.e()得到的
//所以我们直接去看commonParams.e()
public final int e() {
    return this.f24412j;
}

private int f24412j;

public final void t(int i) {
    this.f24412j = i;
}

//commonParams.e()返回的是this.f24412j
//this.f24412j是通过t方法赋值的，再看谁调用的t方法
//直接查找用例有点多，我们hook一下t方法，看看调用栈
//在打开视频的时候有三个地方调用，不过都是通过 hVar.t(k());调用的
//所以我们直接去看k()即可
public final int k() {
    return this.r;
```

```
}

private int r;

public final void O(int i) {
    this.r = i;
}

m2.O(0);


//跟到最后发现是写死的0
//所以aotu_play为0


//在O7中也是这样
```

```
//auto_play总结
//在打开和结束的时候都是0。
//看名字像是是否为自动播放，不过不太重要，给0就好
```

epid

```
putParams("epid", String.valueOf(j7));
//j7是第七个参数

hVar.M0();

private long i;

public final long M0() {
    return this.i;
}

public final void S1(long j2) {
    this.i = j2;
}

hVar.S1(G0 != null ? G0.longValue() : 0L);

//经hook验证，在打开视频的时候，G0是null所以返回的是0
//因此epid也就是0
```

epid_status

```
//在P7中：
putParams("epid_status", str7);
//str7是倒数第7个参数，我们去上层看看
```

```
hVar.L0();

public final String L0() {
    return this.s;
}

private String s;

//找到两处对s赋值的地方
//1、
this.s = commonParams.d();

//2、
public final void R1(String str) {
    this.s = str;
}

//根据刚刚的经验，我们先看2、
hVar.R1(commonParams.d());
//果然最终都是调用的commonParams.d()

//所以去看commonParams.d()
public final String d() {
    return this.l;
}

private String l = "";


//这里也是固定的""

//O7中也是这样
```

```
//epid_status总结
//无论是打开视频还是视频播放结束，用的都是空，即""
```

from

```
//在P7中：
putParams("from", String.valueOf(i4))
//i4是倒数第10个参数，去上层看

hVar.S0()

public final int S0() {
    return this.p;
}

private int p;

//第一种：
 this.p = E0 != null ? E0.intValue() : 6;
```

```
//第二种
public final void X1(int i) {
    this.p = i;
}
hVar.X1(E0 != null ? E0.intValue() : 6);


//最终还是根据E0的值返回的

//直接去看E0 != null ? E0.intValue() : 6;

E0 = r.E0(commonParams.g());

 public final String g() {
     return this.f24411c;
 }

public static Integer E0(String toIntOrNull) {
    Integer F0;
    x.q(toIntOrNull, "$this$toIntOrNull");
    F0 = F0(toIntOrNull, 10);
    return F0;
}

//经hook验证，这个场景下就是7

//在O7中也是
```

```
//form总结
//无论是打开视频还是视频播放结束，用的都是7
```

from_spmid

```
//在P7中：
putParams("from_spmid", str5);
//str5是倒数九个参数，去上边看看

public final String N0() {
    return this.q;
}

@JSONField(name = "from_spmid")
private String q;

//q也是有两种赋值的方式
//方式1：
this.q = commonParams.f();
//方式2：
public final void U1(String str) {
    this.q = str;
}
```

```
hVar.U1(commonParams.f());

//所以直接去看commonParams.f()
public final String f() {
    return this.e;
}

private String e = "";

public final void u(String str) {
    kotlin.jvm.internal.x.q(str, "<set-?>");
    this.e = str;
}


//hook u得知是tv.danmaku.bili.ui.video.playerv2.r.s调用的u

hVar.u(l != null ? l : "");

//很明显这里不是""，所以我们去看看l的值
String l = l();

public final String l() {
    return this.f24409j;
}

private String f24409j;

public final void P(String str) {
    this.f24409j = str;
}

//hook打印堆栈得知是tv.danmaku.bili.ui.video.playerv2.datasource.c.v1调用的p
rVar.P(extra.getString("from_spmid"));

//从 Extra 数据包中获取数据，可以作为固定的用

//在O7中：同P7
```

```
//from_spmid总结
//使用固定的：tm.recommend.0.0
```

last_play_progress_time

```
//在P7中
putParams("last_play_progress_time", String.valueOf(j12));
//j12是倒数第十二个参数
z ? 0L : hVar.v0()
HeartbeatParams N7 = N7(hVar2, true);
//可以看到在调用N7的时候就是直接给的true，所以这里会直接返回0
```

```
//在O7中：
putParams("last_play_progress_time", String.valueOf(j12));
//j12是倒数第十二个参数
z ? 0L : hVar.v0()
HeartbeatParams N7 = N7(hVar2, true);
//在O7中也就是退出播放的时候是给的false。所以我就会走hVar.V0()这个逻辑：
public final long V0() {
    return this.v;
}

@JSONField(name = "last_play_progress_time")
private long v;


public final void I1(int i) {
    long j2 = i / 1000;
    this.v = j2;
    e2(j2);
    this.x = d.f24359j.a() - this.f24362c;
}

public final void c2(long j2) {
    this.v = j2;
}


//在O7中会走到I1. 这里是记录了观看这个视频的时常的毫秒
//进去的时候会在P7中发送一个心跳请求，那会是0
//出来的时候会在O7中发送一个心跳请求，那会是此视频播放的毫秒数/1000，也就是秒
```

```
//last_play_progress_time总结
//1.打开视频时，也就是在P7中，直接给的0
//2.播放结束时，也就是在O7中，是根据播放的时间的毫秒数/1000然后返回的，用int变量接收的，因此只取整数部分
//这个记录的就是上次播放了多少秒
```

list_play_time

```
//在P7中：
putParams("list_play_time", String.valueOf(j15));
//j15是倒数第二个参数
HeartbeatParams N7 = N7(hVar2, true);
z ? 0L : hVar.W0()
//在P7中调用N7函数时传的z是true，因此这里list_play_time的值就为0


//在O7中：
putParams("list_play_time", String.valueOf(j15));
//j15是倒数第二个参数
HeartbeatParams N7 = N7(hVar2, true);
z ? 0L : hVar.W0()
//在O7中调用N7函数时传的z是false，因此这里list_play_time的值就是hVar.W0()的返回值
//hVar.W0()：
```

```
public final long W0() {
    return this.C;
}

@JSONField(name = "list_play_time")
private long C;

public final void d2(long j2) {
    this.C = j2;
}
//跟到这里是通过d2给C赋值的，C就是"list_play_time"的值，我们去hook看看谁调用的d2，其参数就是我们要的值
//hook发现，d2没被调用。但是我们hook W0发现确实返回了this.C，其值为0
//这里应该可以默认为也是0，因为并没有对他进行赋值，那就是默认值，long型默认值为0
```

```
//list_play_time总结：

//在打开视频发送请求时为0
//在结束视频发送请求时也为0
```

max_play_progress_time

```
//在P7中：
putParams("max_play_progress_time", String.valueOf(j13));
//j13是倒数第十一个参数，去上层看看
z ? 0L : hVar.Y0()
//在P7中，z为true，所以直接就是0


//在O7中：
putParams("max_play_progress_time", String.valueOf(j13));
//j13是倒数第十一个参数，去上层看看
z ? 0L : hVar.Y0()
//在O7中z就为false了，进而走的是hVar.Y0()
public final long Y0() {
    return this.w;
}

@JSONField(name = "max_play_progress_time")
private long w;

public final void e2(long j2) {
    if (j2 > this.w) {
        this.w = j2;
    }
}

//这里会先判断j2是否比之前的值大，如果大就把这个值给了this.w
//再去看看谁调用了e2
public final void I1(int i) {
    long j2 = i / 1000;
    this.v = j2;
```

```
    e2(j2);
    this.x = d.f24359j.a() - this.f24362c;
}
//就是I1，这个I1咱们刚刚见过，就是在last_play_progress_time里边，播放完了会走I1记录一下播放了多少秒，记录完了也会把最新的播放时间传到e2中，如果他比之前的大，那么就把它改了，不大的话就没事
```

```
//总结max_play_progress_time：
//在打开视频发送请求的时候是0
//在播放结束发送请求到时候就是最长播放了多少时间，单位是秒
```

```
//接下来测试一下last_play_progress_time与max_play_progress_time的具体意义

//首先从名字来看：
//last_play_progress_time就是最后一次播放了多长时间，也就可以确定上次用户看到了哪里
//max_play_progress_time就是最多的一次播放了多长时间，我觉得这也是后台计算完播率的一个依据之一

//测试思想：
//首先我们打开一个视频，让其播放完：记录下来last_play_progress_time与max_play_progress_time的值
//接着我们重新播放这个视频，播放10秒退出，记录下来last_play_progress_time与max_play_progress_time的值
//再接着我们再打开视频，会跳到上次播放的地方，我们拉进度条把他拉回来
//对比这三组值观察其变化


//完整播放视频后：
last_play_progress_time 92
max_play_progress_time  92
//重播视频9.x秒后退出：
last_play_progress_time 9
max_play_progress_time  9
//重播视频9.x秒后退出，然后再打开视频，会跳到上次退出的地方，然后把进度条拉回1秒的位置然后退出：
last_play_progress_time 1
max_play_progress_time  9
//我们发现max_play_progress_time是记录上次播放了多少秒，而ax_play_progress_time记录的是最长的时候播放了多少秒
```

mid

```
//在P7中：
putParams(EditCustomizeSticker.TAG_MID, String.valueOf(j4));
public static final String TAG_MID = "mid";
//j4是第三个参数，去上层看看
hVar.a1()

public final long a1() {
    return this.e;
}

@JSONField(name = "mid")
private long e;
//找到两个赋值的地方：
```

```
//1、
this.e = com.bilibili.lib.accounts.b.f(BiliContext.f()).H();
//2、
public final void g2(long j2) {
    this.e = j2;
}
hVar.g2(com.bilibili.lib.accounts.b.f(BiliContext.f()).H());

//归根到底还是走的com.bilibili.lib.accounts.b.f(BiliContext.f()).H();
//直接去看com.bilibili.lib.accounts.b.f(BiliContext.f()).H():
public long H() {
    return w();
}

public long w() {
    com.bilibili.lib.accounts.model.a v = v();
    if (v == null) {
        return 0L;
    }
    return v.b;
}

//在w方法中，首先通过v()去拿一个 v
//然后判断v是否为空，为空直接返回0，不为空返回v.b v.b是v下的一个属性，我们只需要去看在new这个对象的时候有没有初始化，如果没有初始化就是默认值，因为在这里拿到v后并没有对v下边的b属性进行操作，所以我们重点去看如何new的，也就是去看v()

//v():
public com.bilibili.lib.accounts.model.a v() {
    return this.a.f();
}

public com.bilibili.lib.accounts.model.a f() {
    if (AccountConfig.e.a().invoke("account_access_token_optimize",
Boolean.TRUE).booleanValue()) {
        return h();
    }
    return g();
}


//hook验证得知，v是null所以直接返回了0


//在07中亦如此
```

```
//mid总结

//在打开视频发送请求的时候是0
//在播放结束发送请求的时候也是0
```

miniplayer_play_time

```
//感觉像最少播放了多长时间。查看抓到的请求发现，都是0.
```

```
//在P7中：
putParams("miniplayer_play_time", String.valueOf(j16));
//是倒数第一个参数，去上层看看
z ? 0L : hVar.b1()
//在P7中直接给的0，因为z是true


//在O7中：
putParams("miniplayer_play_time", String.valueOf(j16));
//是倒数第一个参数，去上层看看
z ? 0L : hVar.b1()
//在O7中走的hVar.b1()，因为z是false
//hVar.b1()：
public final long b1() {
    return this.D;
}

@JSONField(name = "miniplayer_play_time")
private long D;

public final void h2(long j2) {
    this.D = j2;
}

//通过hook发现走了b1(),但是没走h2()
//也就是说明并没有给D这个变量去赋值，因此是默认值0
```

```
//miniplayer_play_time总结：
//在打开视频发送请求的时候是0
//在播放结束发送请求的时候也是0
```

network_type

```
这个看着像网络类型，我抓到的所有都为1
```

```
//在P7中：
putParams("network_type", String.valueOf(i3));
//i3是倒数第13个参数，去看看
hVar.c1()

public final int c1() {
    return this.o;
}

@JSONField(name = "network_type")
private int o;
```

```
//对o进行赋值有两个地方:
//1:
this.o = H.c(f);
//2:
public final void i2(int i) {
    this.o = i;
}

hVar.i2(h.H.c(f));

//验证发现2和1是相同的逻辑：h.H.c(f)
Application f = BiliContext.f();
h.H.c(f)
//f是拿到的上下文Context
//主要逻辑就是h.H.c()
public final int c(Context context) {
    int b = com.bilibili.lib.media.d.b.b(context);
    if (b == -1) {
        return 2;
    }
    if (b == 0) {
        j b2 = j.b();
        x.h(b2, "FreeDataStateMonitor.getInstance()");
        int a = b2.a();
        if (a == 1 || a == 2 || a == 3 || a == 4 || a == 5) {
            return 3;
        }
    } else if (b != 1) {
        return b;
    }
    return 1;
}
//首先通过com.bilibili.lib.media.d.b.b(context);拿到一个int型值b，根据b来判断返回几
//我们去看下com.bilibili.lib.media.d.b.b(context)
public static int b(Context context) {
    if (context != null && c(context)) {
        return d(context) ? 1 : 0;
    }
    return -1;
}
//进来后判断context是否为空，然后调用c判断了一下是否连接网络了，然后调用d(context) ? 1 : 0
private static boolean d(Context context) {
    NetworkInfo a;
    return context != null && (a = a(context)) != null && 1 == a.getType() &&
a.isConnected();
}
//这个是判断当前是否连接了wifi，是的话返回真，我连接的wifi所以返回真；然后到b方法中就会返回1
//再到c方法中返回1.
//相反如果不是连接的wifi在b方法中那就返回0，然后到c方法中应该会返回3，或者1；1还是3就需要根据
int a = b2.a();的值来判断了

//在07中：同上
```

network_type总结：
1、和我们猜测的相同，是根据网络类型来决定是什么值的、
2、如果是wifi返回1
3、如果是数据流量返回3或者1
在打开视频和播放结束时都遵循这个规则

paused_time

这个看着像暂停次数？或者暂停时间？
测试了一下是暂停了多长时间，单位是秒。
最终结果是暂停的总时长，不过打开视频发送请求时为0

```
//在P7中：
putParams("paused_time", String.valueOf(j9));
//j9是正数第12个参数，去看看
z ? 0L : hVar.e1()
//在P7中是0，因为z是true


//在O7中：
z ? 0L : hVar.e1()
//返回的就是hVar.e1()

public final long e1() {
    return this.y;
}

@JSONField(name = "paused_time")
private long y;

public final void l2(long j2) {
    this.y = j2;
}

private final void I7() {
    h hVar = this.f24360c;
    if (hVar != null) {
        long a2 = f24359j.a();
        hVar.l2(hVar.e1() + (a2 - hVar.T0()));
        hVar.z1(a2);
    }
}

//在I7中有一个 hVar.l2(hVar.e1() + (a2 - hVar.T0()));就是设置的paused_time的值
//每暂停一次再播放后就会计算这次暂停+之前暂停的总时长
```

paused_time总结
1、打开视频发送请求时为0
2、播放结束发送请求时为暂停的总时长，单位秒

play_status

不知道是啥。看抓到的所有都是0

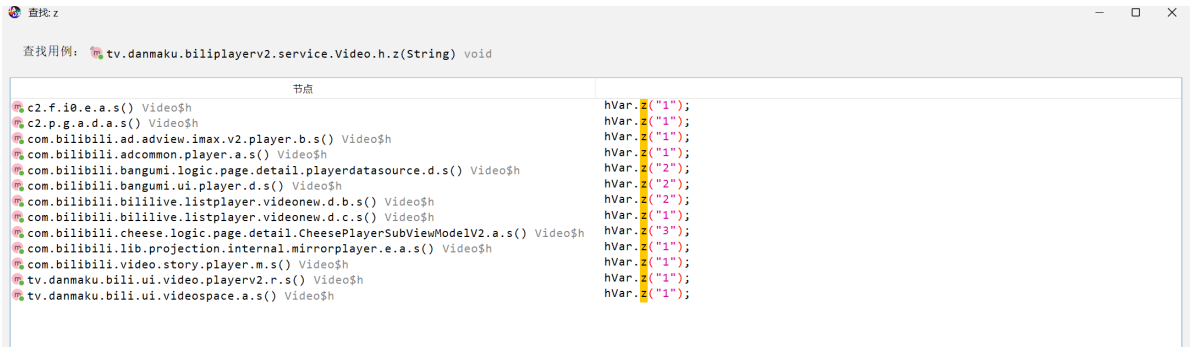
```
//在P7中：
putParams("play_status", str8);
//是倒数第六个，去看看
hVar.f1()

public final String f1() {
    return this.t;
}

@JSONField(name = "play_status")
private String t;

//有两种方式给t赋值
//1、
this.t = commonParams.i();
//2、
public final void m2(String str) {
    this.t = str;
}

hVar.m2(commonParams.i());

//最终都是调用的commonParams.i()，所以现在有两种情况。1是通过commonParams.i()来给t赋值2、没赋值，即默认值
//我们去hook一下commonParams.i()看看执行了吗==>执行了，是通过commonParams.i()赋值的，其值为0
//去看看commonParams.i()的逻辑
public final String i() {
    return this.f24413m;
}

private String f24413m = "";

public final void y(String str) {
    kotlin.jvm.internal.x.q(str, "<set-?>");
    this.f24413m = str;
}

//因为y的用例很多，有13个，但是有10个是直接给的"0".我们直接去hook一下打印个调用栈
//经hook验证都是在tv.danmaku.bili.ui.video.playerv2.r.s中调用的
hVar.y("0");
//这里边是直接写死的"0"


//在O7中：同上
```

play_status总结
打开视频和播放结束都为0，默认的

play_type

玩的类型？ 不知其意！！

```
//在P7中：
putParams("play_type", str4);
//倒数第十四个参数
hVar.i1()

public final String i1() {
    return this.n;
}

@JSONField(name = "play_type")
private String n;

//两种方法：
//1、
this.n = commonParams.j();
//2、
public final void o2(String str) {
    this.n = str;
}

hVar.o2(commonParams.j());

//还是同一种方法：commonParams.j()
public final String j() {
    return this.k;
}

private String k = "";

public final void z(String str) {
    kotlin.jvm.internal.x.q(str, "<set-?>");
    this.k = str;
}
//查找z的用例，发现全是写死的，如下图：
```




```
//在这个场景下为1，所以固定就是1
```

play_type总结

打开视频和播放结束发送请求时都是固定的1

played_time

这个看着像完了多长时间了，应该是打开app多久了？
看了看抓包每次都不是固定的
多次测试发现是播放时长，暂停了不算时间
不过打开视频的时候是0

```
//在P7中：
putParams("played_time", String.valueOf(j10));
//j10是第十三个参数
z ? 0L : hVar.k1()
//在P7中是0，z是true


//在O7中：
putParams("played_time", String.valueOf(j10));
//j10是第十三个参数
z ? 0L : hVar.k1()
//z是false，所以其值为：hVar.k1()
public final long k1() {
    return this.z;
}

@JSONField(name = "played_time")
private long z;

public final void p2(long j2) {
    this.z = j2;
}

private final void K7() {
    h hVar = this.f24360c;
    if (hVar != null) {
        long a2 = f24359j.a();
        long T0 = a2 - hVar.T0();
        hVar.p2(hVar.k1() + T0);
        hVar.J1(hVar.G0() + (((float) T0) * this.e));
        if (this.g) {
            hVar.h2(hVar.b1() + T0);
        }
        hVar.Z1(a2);
        if (this.f) {
            hVar.d2(hVar.k1());
        }
    }
}

//这个就是播放时长通过hVar.p2(hVar.k1() + T0);得到的 每暂停一次就会记录一次，然后会累加
```

played_time总结

在打开视频发送请求时为0
在播放结束发送请求时为播放时长
这个参数也可能会放到完播率校验中或者风控中。假如这个值和总时长对不上，但是其他的能对上就说明有问题

quality

quality：质量
画质?音质?
看了看抓包记录一直是32

```
//在P7中：
putParams("quality", String.valueOf(i2));
//i2是第十个参数
hVar.o1()

public final int o1() {
    return this.l;
}

@JSONField(name = "quality")
private int l;

//找到两个地方给l赋值
//1、
public final void H1(Video.h commonParams, int i, int i2) {
    ...
    if (f != null) {
        ...
        this.l = i;
        ...
    }
}

//2、
public final void q2(int i) {
    this.l = i;
}

//先去hook一下，看看是走的哪个给l赋值的
//在P7中走的是q2，我们继续往上跟
//经过漫长的hook 验证，最终找到了
public int o1() {
    PlayIndex i2;
    MediaResource mediaResource = this.q;
    if (mediaResource == null || (i2 = mediaResource.i()) == null) {
        return 0;
    }
    return i2.b;
}
```

```
//经验证这个是关于画质问题的，找了半天没有找到在哪里设置画质，就没测试了


//在O7中：
putParams("quality", String.valueOf(i2));
//i2是第十个参数
hVar.o1()

public final int o1() {
    return this.l;
}

@JSONField(name = "quality")
private int l;

//找到两个地方给l赋值
//1、
public final void H1(Video.h commonParams, int i, int i2) {
    ...
    if (f != null) {
        ...
        this.l = i;
        ...
    }
}

//2、
public final void q2(int i) {
    this.l = i;
}

//先去hook一下，看看是走的哪个给l赋值的
//在O7中走的是 H1，我们继续往上跟
//跟过来后，也是最终调用了o1()拿到的
```

quality总结
在打开视频发送请求的时候和结束的时候调用的同一个方法，应该是返回的画质，32 16都可以，后剿应该不做校验

sid

不知道是什么。大致看了几个都是0

```
//在P7中：
putParams("sid", str2);
//str2是第六个参数
hVar.s1()

public final String s1() {
    return this.h;
}

@JSONField(name = "sid")
```

```
private String h;

//发现有两个给h赋值的地方，和之前一样，最终走的都是同一个逻辑commonParams.k()
//1、
this.h = String.valueOf(commonParams.k());
//2、
public final void u2(String str) {
        this.h = str;
    }

hVar.u2(String.valueOf(commonParams.k()));


//直接去看commonParams.k()
public final long k() {
    return this.f;
}

private long f;

//接下来去看f赋值的地方只有一个A(),所以只有两种情况，一种没有赋值为默认值0，一种是通过A()进行赋值，我们直接去hook一下A看看
public final void A(long j2) {
    this.f = j2;
}


//经过hook发现并没有走A方法，那就是默认值

//在O7中：同上
```

sid总结
打开视频和结束的时候都是默认值0

spmid

不知道是什么哈哈哈，直接去看把

```
//在P7中：
putParams("spmid", str6);
//str6是倒数第八个参数
hVar.t1()

public final String t1() {
    return this.r;
}

@JSONField(name = "spmid")
private String r;

//找到两种赋值方式
//1、
this.r = commonParams.l();
```

```
//2、
public final void v2(String str) {
    this.r = str;
}
hVar.v2(commonParams.l());

//最终都是调用commonParams.l()
public final String l() {
    return this.d;
}

private String d = "";

public final void B(String str) {
    kotlin.jvm.internal.x.q(str, "<set-?>");
    this.d = str;
}

//发现B有好多用例，直接去hook
//都是走的 tv.danmaku.bili.ui.video.playerv2.r.s
public Video.h s() {
    ...
    String y = y();
    if (y == null) {
        y = "";
    }
    hVar.B(y);
    ...
}

//通过String y = y();得到一个y，然后把y传到 B(String str)中进行赋值的
public final String y() {
    return this.i;
}

private String i;

public final void V(String str) {
    this.i = str;
}
//发现V又有很多用例，再去hook V
public void v1(BiliVideoDetail biliVideoDetail, Bundle extra) {
    ...
    rVar2.V(extra.getString("spmid"));
    ...
}

//从 Extra 数据包中获取数据，可以作为固定的用
```

spmid总结
打开视频和播放结束发送请求都是固定的：main.ugc-video-detail.0.0
是从Extra 数据包中获取的数据

start_ts

这个应该是开始播放的时间把，或者打开视频的时间
经过抓包验证，打开视频的时候为0
在视频播放结束发送请求的时候，start_ts的值是打开视频时的时间

```
//先看P7中：
putParams("start_ts", String.valueOf(j2));
//j2是第一个参数
hVar.p1()

public final long p1() {
    return this.F;
}

@JSONField(name = "server_time")
private long F;

public final void r2(long j2) {
    this.F = j2;
}

public void onDataSuccess(String str) {
    h hVar = d.this.f24360c;
    if (hVar != null) {
        if ((str == null || str.length() == 0) || !x.g(hVar.r1(), this.b)) {
            return;
        }
        JSONObject jSONObject = new JSONObject(str);
        h hVar2 = d.this.f24360c;
        if (hVar2 == null) {
            x.K();
        }
        hVar2.r2(jSONObject.optLong("ts", 0L));
    }
}

//在onDataSuccess中调用了hVar2.r2(jSONObject.optLong("ts", 0L));
//是从jSONObject中取的字段为ts的值
//可以看到new jSONObject的时候传入的是str，str是调用onDataSuccess的时候传进来的，看谁调用的onDataSuccess
//用例比较多，去hook打印个堆栈
//上层是com.bilibili.okretro.b.onResponse(BL:9)
public void onResponse(retrofit2.b<GeneralResponse<T>> bVar,
l<GeneralResponse<T>> lVar) {
    if (isCancel()) {
        return;
    }
    if (lVar.g() && !isCancel()) {
        GeneralResponse<T> a = lVar.a();
        if (a == null) {
            onDataSuccess(null);
            return;
        } else if (a.code != 0) {
```

```
            if (com.bilibili.api.f.a.a() && a.code == -400) {
                BLog.e("BiliApi", "WTF?! Check your parameters!");
            }
            onFailure(bVar, new BiliApiException(a.code, a.message));
            return;
        } else {
            onDataSuccess(a.data);
            return;
        }
    }
    onFailure(bVar, new HttpException(lVar));
}
//跟到这里发现是向服务器发送请求的时候去设置的start_ts
//发现是在P7中也就是打开视频向服务端发送请求后，会返回一个时间戳，然后把这个时间戳作为O7也就是播放结束后发送请求的start_ts
```

start_ts总结
1、在打开视频的时候start_ts默认为0，然后会发送一个请求，这个请求会返回一个JSON类型字符串：{"ts":1747401074}
2、然后会把这个字符串，放到start_ts中，在播放结束后再发请求就会拿刚刚这个去发

sub_type

看了几个都是0

```
//在P7中：
putParams("sub_type", String.valueOf(i));
//i是第九个参数
hVar.w1()

public final int w1() {
    return this.k;
}

@JSONField(name = "sub_type")
private int k;

//发现了两个地方给k赋值
//1、
public final void B2(int i) {
    this.k = i;
}
hVar.B2(commonParams.m());
//2、
this.k = commonParams.m();

//最终都是走的this.k = commonParams.m(); 所以k就只有两种情况：1：没赋值，是默认值 2：通过this.k = commonParams.m();进行的赋值。所以我们去hook一下这个commonParams.m()==>发现走了commonParams.m()继续往上跟
public final int m() {
    return this.h;
}
```

```
private int h;

public final void C(int i) {
    this.h = i;
}

//有好几个方法中调用了C，hook一下看看堆栈==>到这里发现并没有触发C函数，那么就是说在这个场景下h为默认值0



//在O7中：同上
```

sub_type总结：
在打开视频时为0 在视频结束时也为0

total_time

total_time？ 总时长？ 不清楚！ 去Charles中看看抓到的是什么！！
每个不一样，但是发现大多数和played_time一样，因为大多数没暂停，我暂停一下看看，通过paused_time+played_time看看total_time是不是总时长
打开视频时：total_time为0
视频结束时：played_time:11、paused_time:27、total_time：38
哈哈哈哈！猜出来了，果然如此！看来还是要学好英语的，尤其是对付大厂的app，他们都比较规范，这样能让我们省不少时间呢！！
接下来跟着代码去分析分析：

```
//在P7中：
putParams("total_time", String.valueOf(j8));
//第十一个参数
z ? 0L : hVar.B1()
//在P7中为0，因为z为true


//在O7中
putParams("total_time", String.valueOf(j8));
//第十一个参数
z ? 0L : hVar.B1()
//这里就是hVar.B1()因为z为false
hVar.B1()

public final long B1() {
    return this.x;
}

@JSONField(name = "total_time")
private long x;

public final void I1(int i) {
    long j2 = i / 1000;
    this.v = j2;
    e2(j2);
    this.x = d.f24359j.a() - this.f24362c;
```

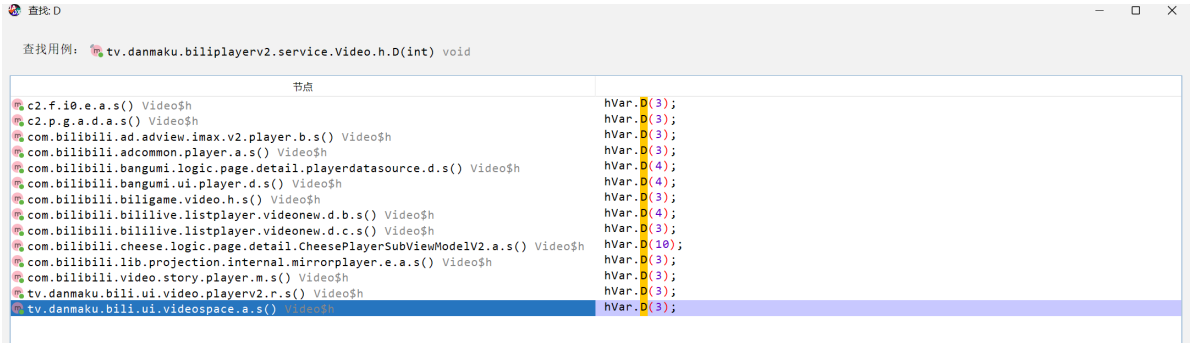
```
}
//也调用了I1，然后通过d.f24359j.a() - this.f24362c;得出的

d.f24359j.a():
public final long a() {
    return SystemClock.elapsedRealtime() / 1000;
}
//获取设备启动后经过的秒级相对时间

//this.f24362c是视频播放时获取的时间戳，做完减法就是在这个视频页面停留的总时长
```

```
total_time总结
1、打开视频时为：0
2、视频播放完时为：在视频页面停留的总时长
```

type

```
type 类型？ 不知道是什么的类型！看了看抓到的都是3
```

```
//在P7中：
putParams("type", str3);
//str3是第8个参数
hVar.C1()

public final String C1() {
    return this.f24363j;
}

@JSONField(name = "type")
private String f24363j;

//有两种赋值方式：
//1：
public final void C2(String str) {
    this.f24363j = str;
}
hVar.C2(String.valueOf(commonParams.n()));

//2：
this.f24363j = String.valueOf(commonParams.n());

//最终走的都是commonParams.n()
public final int n() {
    return this.g;
}

private int g;

public final void D(int i) {
    this.g = i;
}
```




```
//发现都是常量，在这个情景下就是固定的3


//在O7中：同上
```

```
type总结
1、打开视频时为：3
2、视频播放完了：3
```

user_status

```
user_status：用户状态？ 用户地位？  抓包测试了几次都是0！
```

```
//在P7中：
putParams("user_status", str9);
//str9是倒数第五个参数
hVar.D1()

public final String D1() {
    return this.f24365u;
}

@JSONField(name = "user_status")
private String f24365u;

//找到两处对f24365u赋值的地方
//1、
this.f24365u = (l == null || !l.isEffectiveVip()) ? "0" : "1";
//2、
public final void E2(String str) {
    this.f24365u = str;
}

//去hook一下 看看走的哪个==>打开视频走的E2，退出视频走的另一个
//在p7中我们就去看E2，看看谁调用了E2
//发现是在b方法中，用了和退出视频时的同一个方式：hVar.E2((l == null ||
!l.isEffectiveVip()) ? "0" : "1");

//这里应该是判断是否为有效的vip，是的话为1，不是的话为0。因为我不是vip所以就返回的0
public boolean isEffectiveVip() {
    int i = this.vipType;
```

```
    return (i == 1 || i == 2) && this.vipStatus == 1;
}
//这里通过this.vipType拿到是否为会员。 1代表是会员然后就返回true
```

user_status总结：
判断用户是否为会员，在P7或者O7皆如此！

video_duration

video_duration 视频持续时间？ 我记得total_time是总时间啊？ 去抓包看看！ 好像是app打开了多长时间！

经过多次抓包验证，这个是视频的总时长-1秒，以秒为单位

```
//在P7中：
putParams("video_duration", String.valueOf(j11));
//j11是第十四个参数
hVar.E1()

public final long E1() {
    return this.f24364m;
}

@JSONField(name = "video_duration")
private long f24364m;

public final void F2(long j2) {
    this.f24364m = j2;
}
    ....

    ...

//最终调用的这个拿到了总时长
PlayerCoreServiceV2.this.getDuration()
```

video_duration总结：
视频总时长-1秒

1