现在我们可以看到很多 App 在请求 API 的时候都有加密参数，前面我们也介绍了一种利用 mitmdump 来实时抓取数据的方法，但是这总归还有些不方便的地方。

如果要想拿到 App 发送的请求中包含哪些加密参数，就得剖析本源，深入到 App 内部去找到这些加密参数的构造逻辑，理清这些逻辑之后，我们就能自己用算法实现出来了。这其中就需要一定的逆向操作，我们可能需要对 App 进行反编译，然后通过分析源码的逻辑找到对应的加密位置。

所以，本课时我们来用一个示例介绍App 逆向相关操作。

案例介绍

这里我们首先以一个 App 为例介绍这个 App 的抓包结果和加密情况，然后我们对这个 App 进行逆向分析，最后模拟实现其中的加密逻辑。

App 的下载地址为：https://app5.scrape.center/

我们先运行一下这个 App，上拉滑动，一些电影数据就会呈现出来了，界面如下：

这时候我们用 Charles 抓包来试一下，可以看到类似 API 的请求 URL 类似如下：https://app5.scrape.center/api/movie/?offset=0&limit=10&token=NDVjMTdjNjk5YWM2NWZkOGU5ZjFjNWEyN2MzNjhiYjIwMzRlZDU3ZiwxNTkxMjgyMzcz%0A，这里我们可以发现有三个参数，分别为 offset、limit 还有 token，其中 token 是一个非常长的加密字符串，我们也不好直观地推测其生成逻辑。

本课时我们就来介绍一下逆向相关的操作，通过逆向操作获得 apk 反编译后的代码，然后追踪这个 token 的生成逻辑是怎样的，最后我们再用代码把这个逻辑实现出来。

App 逆向其实多数情况下就是反编译得到 App 的源码，然后从源码里面找寻特定的逻辑，本课时就来演示一下 App 的反编译和入口点查找操作。

环境准备

在这里我们使用的逆向工具叫作 JEB。

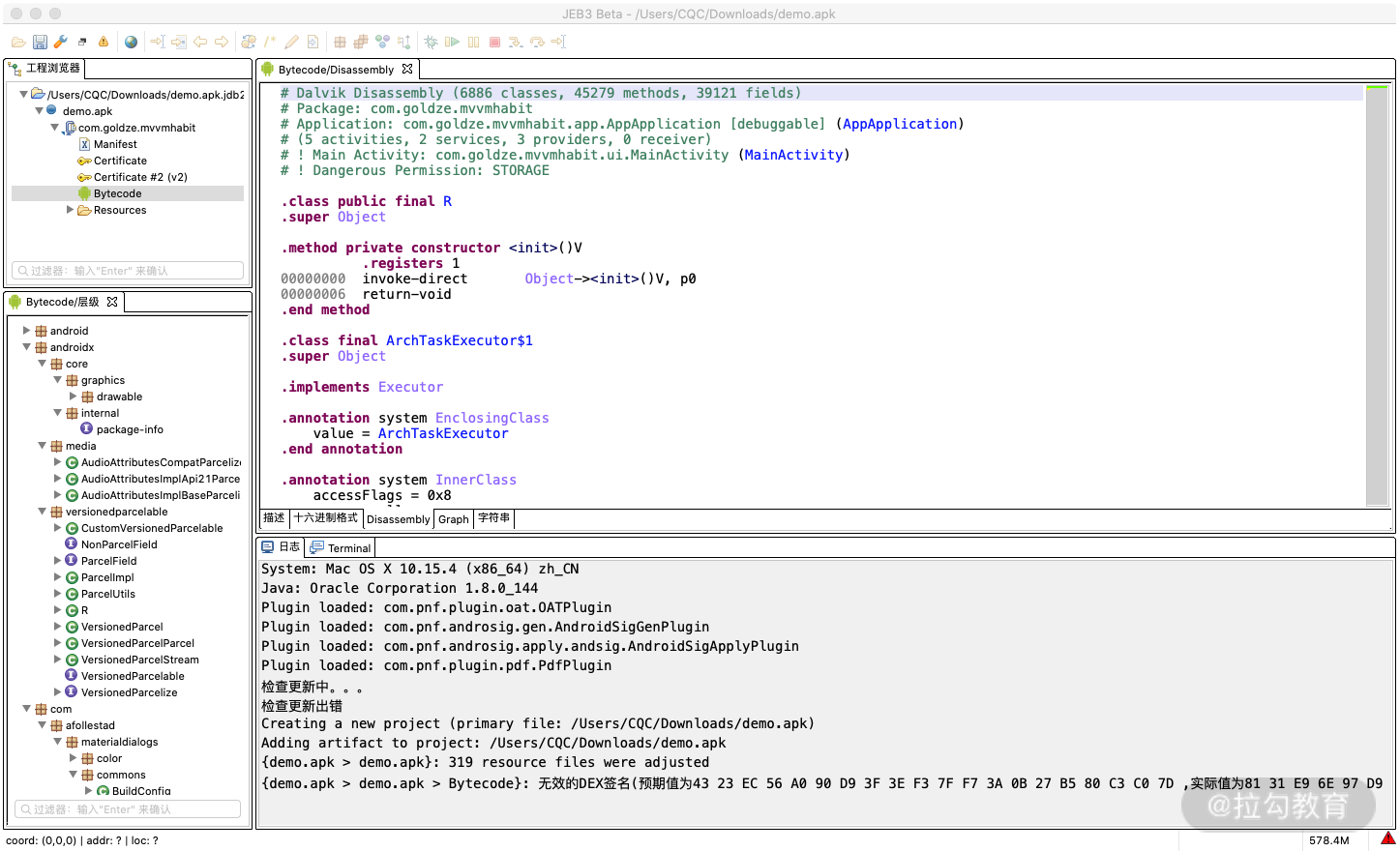
JEB 是一款专业的安卓应用程序的反编译工具，适用于逆向和审计工程，功能非常强大，可以帮助逆向人员节省很多逆向分析时间。利用这个工具我们能方便地获取到 apk 的源码信息，逆向一个 apk 不在话下。

JEB 支持 Windows、Linux、Mac 三大平台，其官网地址为 https://www.pnfsoftware.com/，你可以在官网了解下其基本介绍，然后通过搜索找到一些完整版安装包下载。下载之后我们会看到一个 zip 压缩包，解压压缩包之后会得到如下的内容：

在这里我们直接运行不同平台下的脚本文件即可启动 JEB。比如我使用的是 Mac，那我就可以在此目录下执行如下命令：

```
sh jeb_macos.sh
```

这样我们就可以打开 JEB 了。打开 JEB 之后，我们把下载的 apk 文件直接拖拽到 JEB 里面，经过一段时间处理后，会发现 JEB 就已经将代码反编译完成了，如图所示：



这时候我们可以看到在左侧 Bytecode 部分就是反编译后的代码，在右侧显示的则是 Smali 代码，通过 Smali 代码我们大体能够看出一些执行逻辑和数据操作等过程。

现在我们得到了这些反编译的内容，该从哪个地方入手去找入口呢？

由于这里我们需要找的是请求加密参数的位置，那么最简单的当然是通过 API 的一些标志字符串来查找入口了。API 的 URL 里面包含了关键字 /api/movie，那么我们自然就可以通过这个来查找了。

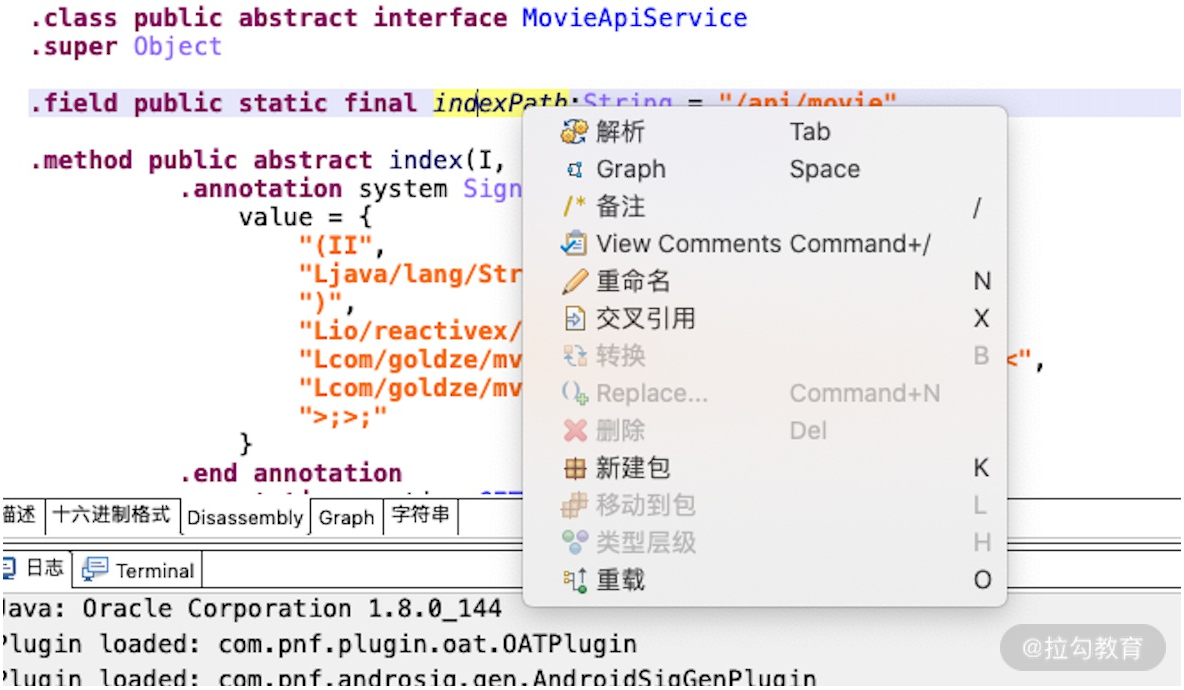
我们可以在 JEB 里面打开查找窗口，查找 /api/movie，如图所示：

这时候我们发现就找到了一个对应的声明如下：

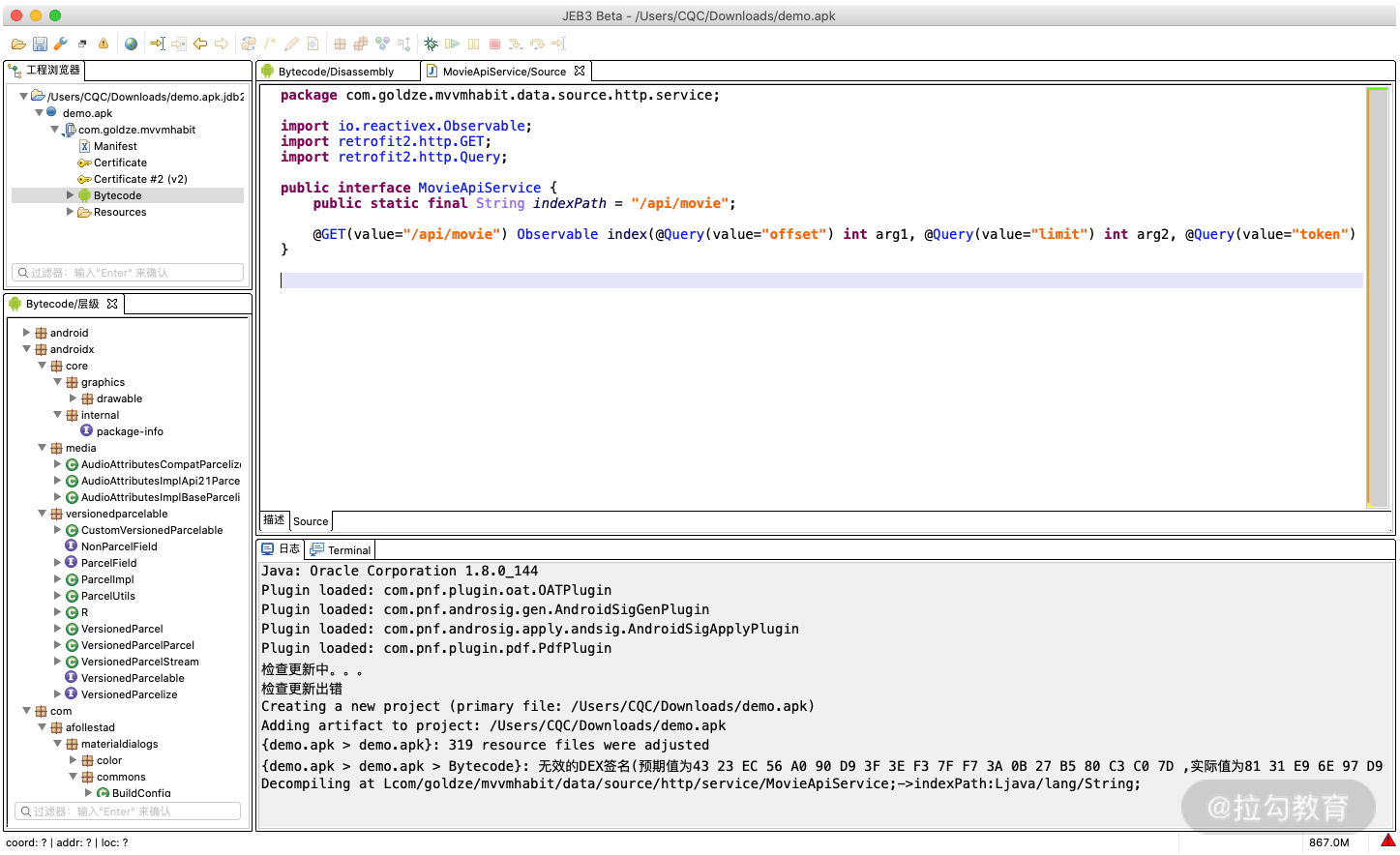
```
.field public static final indexPath:String = "/api/movie"
```

这里其实就是声明了一个静态不可变的字符串，叫作 indexPath。但这里是 Smali 代码呀？我们怎么去找到它的源码位置呢？

这时候我们可以右键该字符串，选择解析选项，这时 JEB 就可以成功帮我们定位到 Java 代码的声明处了。



这时候我们便可以看到其跳转到了如下页面：



这里我们就能看到 indexPath 的原始声明，同时还看到了一个 index 方法的声明，包含三个参数 offset、limit 还有 token，由此可以发现，这参数和声明其实恰好和 API 的请求 URL 格式是相同的。

但这里还观察到这个是一个接口声明，一定有某个类实现了这个接口。

我们这时候可以顺着 index 方法来查询是什么类实现了这个 index 方法，在 index 方法上面右键选择“交叉引用”，如图所示：

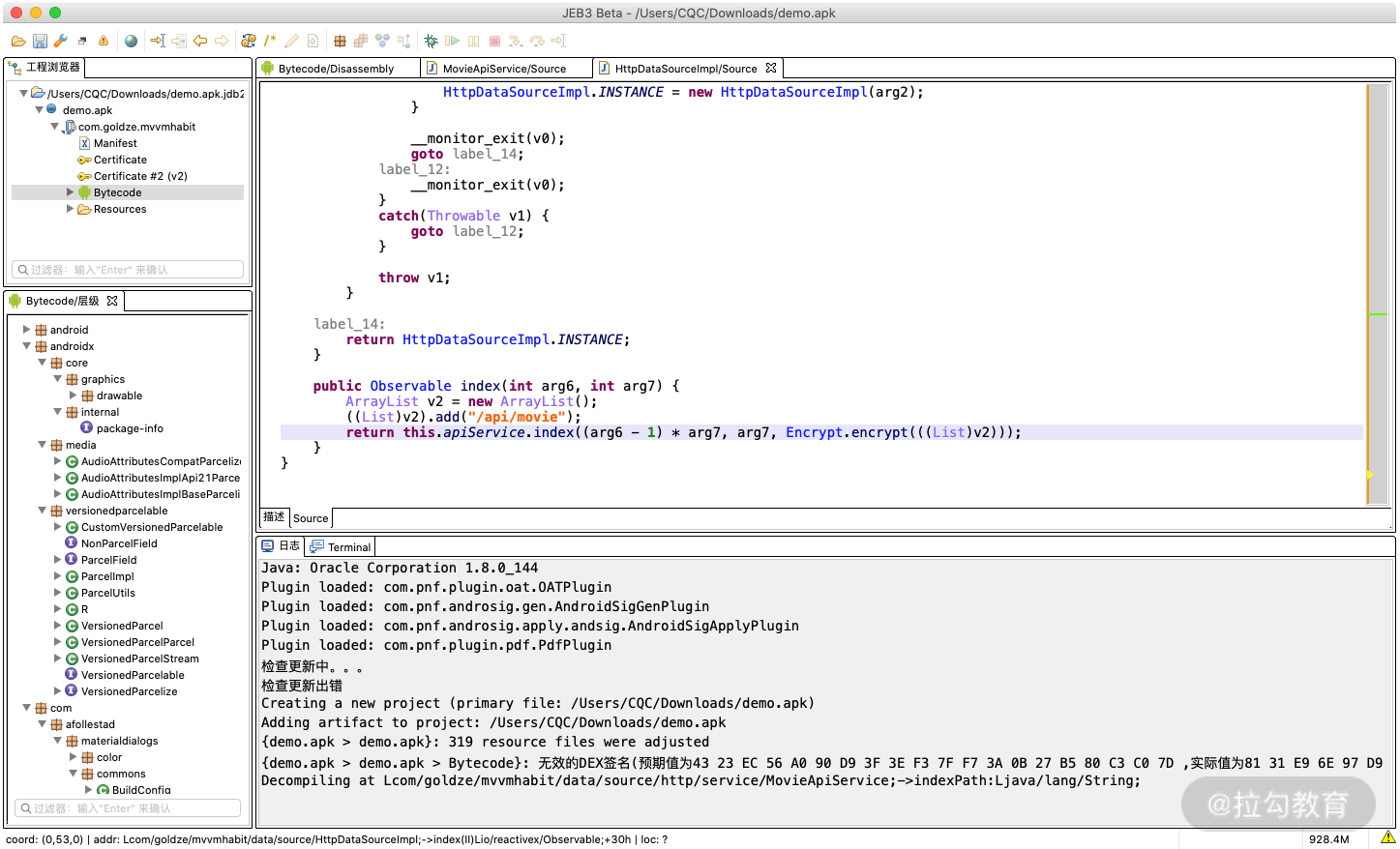
```
import io.reactivex.Observable;
import retrofit2.http.GET;
import retrofit2.http.Query;

public interface MovieApiService {
    public static final String indexPath = "/api/movie";

    @GET(value="/api/movie") Observable ind
}
```

这时候我们可以发现这里弹出了一个窗口，找到了对应的位置，如图所示：

我们选中它，点击确定，这时候就跳转到了对应的 index 实现的位置了，如图所示：

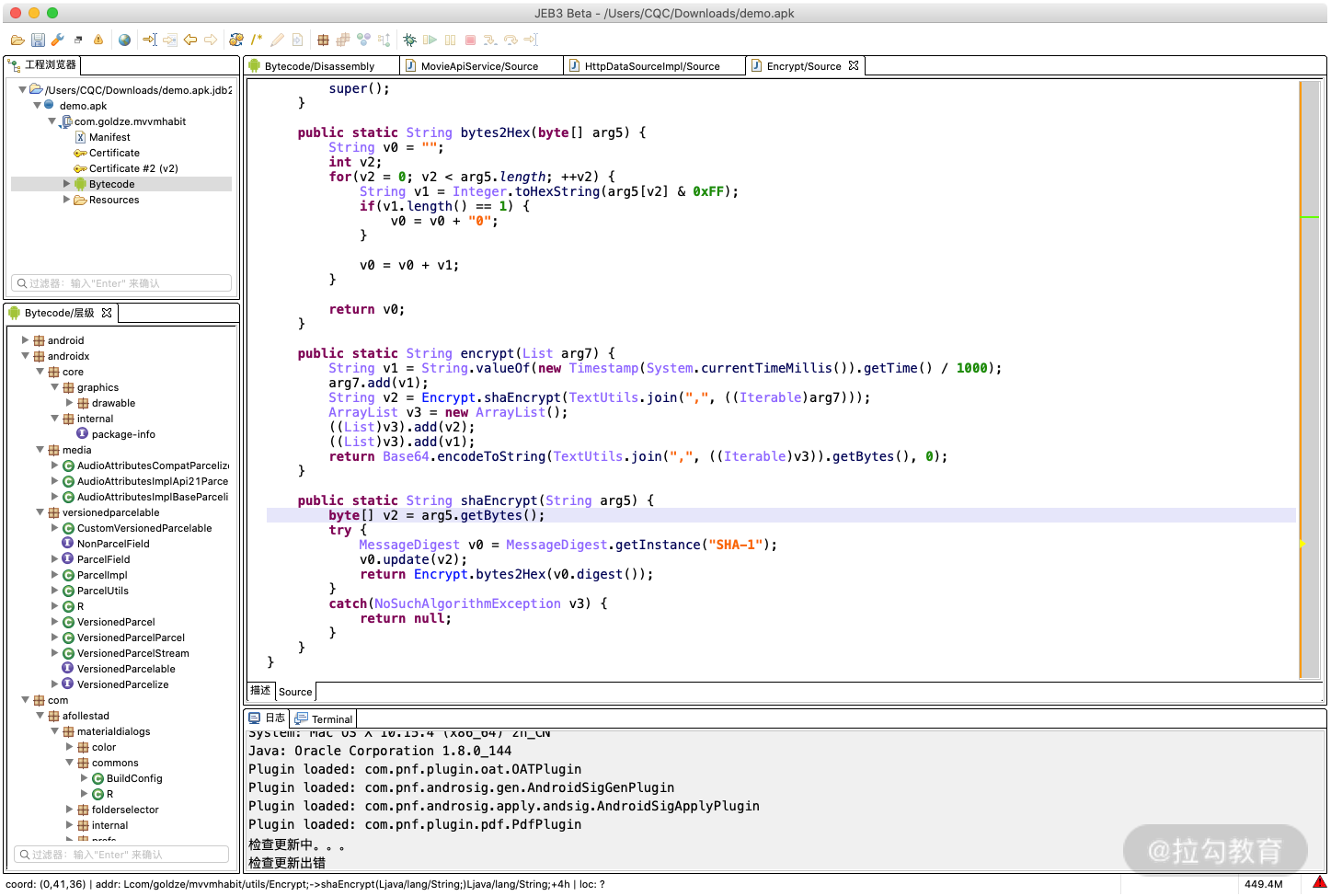


这里 index 方法的实现如下：

```
public Observable index(int arg6, int arg7) {
```

```
    ArrayList v2 = new ArrayList();
    ((List)v2).add("/api/movie");
    return this.apiService.index((arg6 - 1) * arg7, arg7, Encrypt.encrypt(((List)v2)));
}
```

就能很轻易地发现一个类似 encrypt 的方法，代表加密的意思，其参数就是 v2，而 v2 就是一个 ArrayList，包含一个元素，就是 /api/movie 这个字符串。
这时候我们再通过交叉引用找到 Encrypt 的定义，跳转到如图所示的位置：



这里可以发现 encrypt 的方法实现如下：

```
public static String encrypt(List arg7) {
    String v1 = String.valueOf(new Timestamp(System.currentTimeMillis()).getTime() / 1000);
    arg7.add(v1);
    String v2 = Encrypt.shaEncrypt(TextUtils.join(",", ((Iterable)arg7)));
    ArrayList v3 = new ArrayList();
    ((List)v3).add(v2);
    ((List)v3).add(v1);
    return Base64.encodeToString(TextUtils.join(",", ((Iterable)v3)).getBytes(), 0);
}
```

这里我们分析一下，传入的参数就是 arg7，刚才经过分析可知 arg7 其实就是一个长度为 1 的列表，其内容就是一个字符串，即 `["/api/movie"]`。
紧接着看逻辑，这里又定义了一个 v1 的字符串，其实就是获取了时间戳信息，然后把结果加入 arg7，现在 arg7 就有两个内容了，一个是 `/api/movie`，另一个是时间戳。

接着又声明了 v2，这里经过分析可知是将 arg7 使用逗号拼接起来，然后调用了 shaEncrypt 操作，而 shaEncrypt 经过观察其实就是 SHA1 算法。

紧接着又声明了一个 ArrayList，把 v2 和 v1 的结果加进去。最后把 v3 的内容使用逗号拼接起来，然后 Base64 编码即可。

好，现在整体的 token 加密的逻辑就理清楚了。

模拟

了解了基本的算法流程之后，我们可以用 Python 把这个流程实现出来，代码实现如下：

```
import hashlib
import time
import base64
from typing import List, Any
import requests

INDEX_URL = 'https://app5.scrape.cuiqingcai.com/api/movie?limit={limit}&offset={offset}&token={token}'
LIMIT = 10
OFFSET = 0

def get_token(args: List[Any]):
    timestamp = str(int(time.time()))
    args.append(timestamp)
    sign = hashlib.sha1(','.join(args).encode('utf-8')).hexdigest()
    return base64.b64encode(','.join([sign, timestamp]).encode('utf-8')).decode('utf-8')

args = ['/api/movie']
token = get_token(args=args)
index_url = INDEX_URL.format(limit=LIMIT, offset=OFFSET, token=token)
response = requests.get(index_url)
print('response', response.json())
```

这里最关键的就是 token 的生成过程，我们定义了一个 get_token 方法来实现，整体上思路就是上面梳理的内容：

- 列表中加入当前时间戳；
- 将列表内容用逗号拼接；
- 将拼接的结果进行 SHA1 编码；
- 将编码的结果和时间戳再次拼接；
- 将拼接后的结果进行 Base64 编码。

最后运行结果如下：

```
response {'count': 100, 'results': [{'id': 1, 'name': '霸王别姬', 'alias': 'Farewell My Concubine', 'cover': 'https://p0.meituan.net/movie/ce4da3e03e655b5b88ed31b5cd7896cf62472.jpg@464w_
```

这里就以第一页的数据为示例来演示了，其他的页面我们通过修改 page 的值就可以拿到了。

总结

以上我们便通过一个样例讲解了一个比较基本的 App 的逆向过程，包括 JEB 的使用、追踪代码的操作等等，最后通过分析代码理清了基本逻辑，最后模拟实现了 API 的参数构造和请求发送，得到最终的数据。