

# JavaAPI-Scanner

一个对 **Java** 软件包中 **API** 进行提取的扫描器

队 伍: mymk\_one (1186152014)

主要成员: 赵雯 <withzhaowen@126.com>

2021/03/27

# 目录

JavaAPI-Scanner .....	1
一个对 Java 软件包中 API 进行提取的扫描器.....	1
一、项目简介 .....	3
1.项目背景.....	3
2.技术环境简介 .....	3
2.2.1 开发环境.....	3
2.2.2 运行环境 .....	3
2.2.3 架构图 .....	4
3.文件结构.....	4
二、功能描述 .....	5
1.Scanner 扫描器：从软件包源码中提取方法信息 .....	5
2.Compiler 解析器：对方法进行解析 .....	6
3.Job 任务：完成一次完整的源码解析操作 .....	8
4.JSON：保存输出结果 .....	9
三、使用方法 .....	10
1.项目环境构建 .....	10
将项目克隆后导入 Python.....	10
2 运行项目 .....	10
3.2.1 运行 main.py 文件.....	10
3.2.2 根据提示输入一个文件路径 .....	10
3.2.3 控制台提示写入成功后，在项目目录中也出现了一个新的 Json 文件 .....	11
四、参考资料 .....	12

## 一、项目简介

### 1.项目背景

现有的 openEuler 系统有提供应用程序二进制接口，可以支持编程语言编译运行环境的构建，但是由于软件包因开发语言、环境与性能的差异，往往会影响在不同操作系统上的兼容性。

因此从上述问题出发，为了更好的保证软件包在 openEuler 操作系统上的运行状况，于是提出通过获取软件包 API 这一方式，来对操作系统软件兼容性分析提供一定的参考依据，从而提高操作系统支撑软件的兼容性。

因 Java 项目在开发以及运行时都会依托于 JVM（Java 虚拟机），因此对于提取 Java 编写的软件包 API 功能是有一定规律可循的，在本项目中，使用 Fastjson 作为待提取示例。FastJson 是一个基于 Java 语言、运行于 Java 虚拟机（JVM）的序列化软件包；因此分析时应参考 Java 语法的特性，从语法规则出发，使用分别以「扫描器」与「解析器」两种形式相结合的操作来完成 API 的提取与分析操作。

## 2.技术环境简介

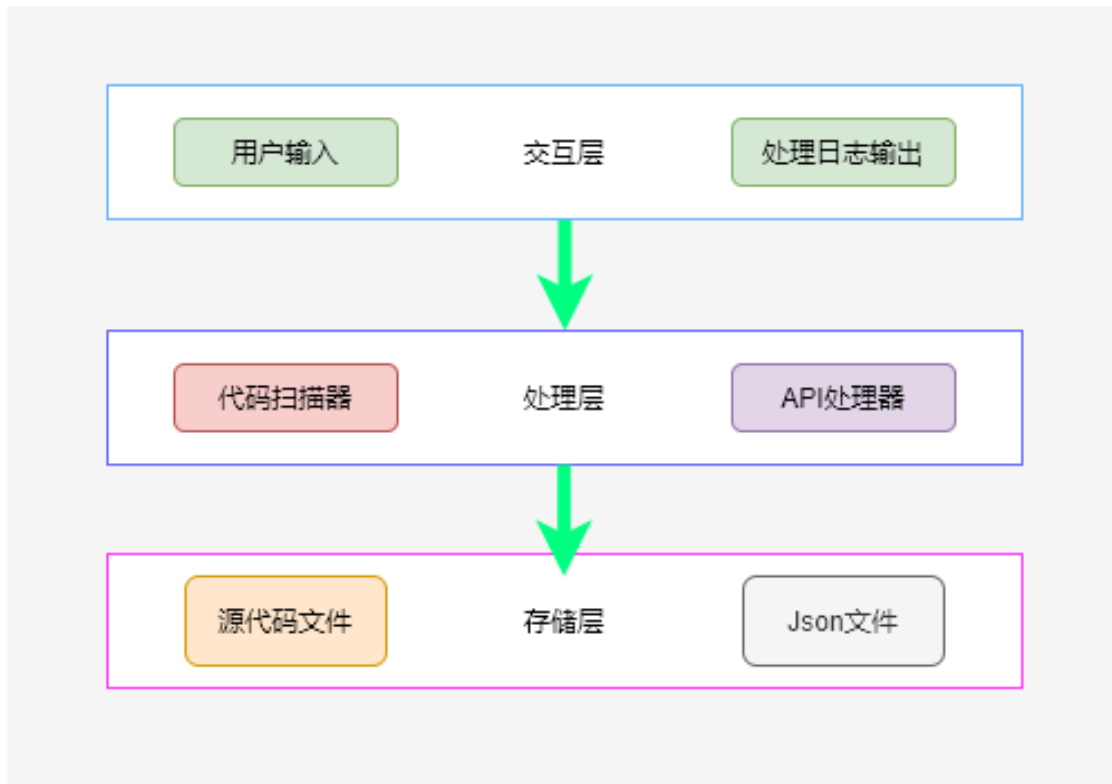
### 2.2.1 开发环境

名称	版本	备注
Python	3.9.2	Python

### 2.2.2 运行环境

名称	版本	备注
Microsoft Windows	win 10	开发环境

### 2.2.3 架构图



### 3. 文件结构

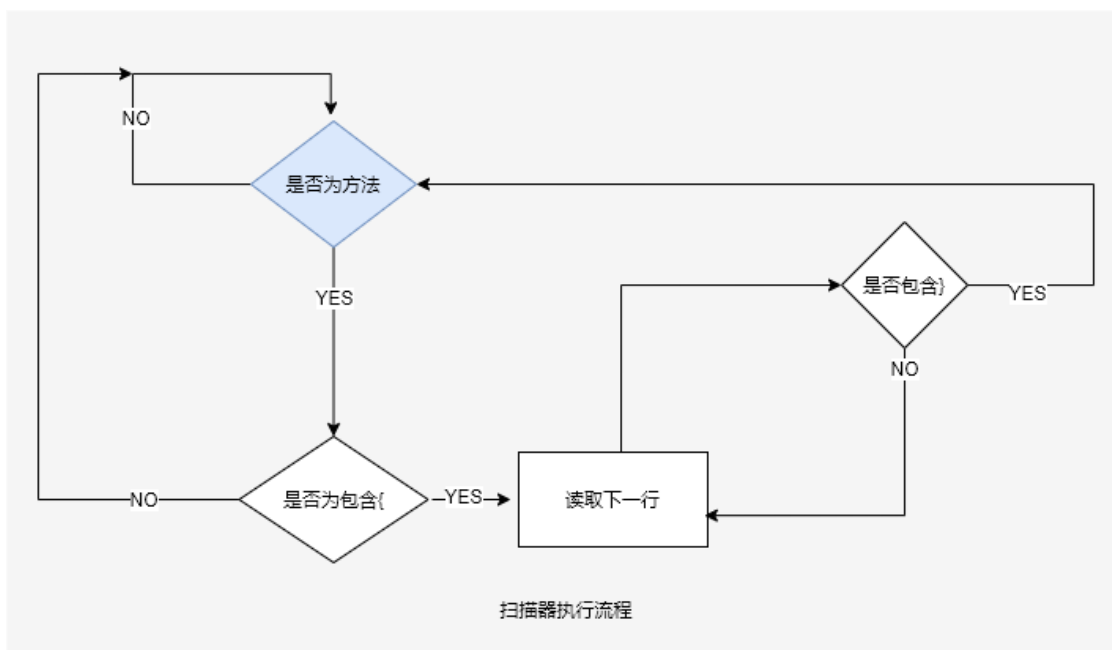
```
├── api-scanner # 项目源码
│   ├── method_analysis_utils # 方法提取工具
│   │   ├── scanner # 扫描器: 扫描源码中的方法
│   │   └── complier # 解析器: 解析方法中内外部访问 API
│   ├── source # 源码、临时文件存放
│   │   ├── code # fastjson 源码
│   │   └── save # 编码过程中一些临时文件
│   ├── temp # 默认输出目录
│   ├── main.py # 项目运行入口
│   ├── method_analysis_job.py # 扫描器与编译器的封装任务文件
│   ├── file_utils.py # 关于文件 IO 流的工具类
│   ├── logging.conf # logging 日志基本配置
│   ├── method_testcase.py # 测试文件: 测试 scanner 和 complier
│   └── file_testcase.py # 测试文件: 测试文件 io 流程相关
├── document # 比赛项目文档、图片等静态文件存放处
│   ├── 项目功能说明书.pdf # 项目功能说明书 pdf 版
│   └── 【HTML 版】方法文档 # pydoc 生成的 API 文档
```

## 二、功能描述

### 1.Scanner 扫描器：从软件包源码中提取方法信息

利用正则表达式、Python API 的方式根据特定规则从源码文件中提取到 Java 代码中的所有方法头、所在包信息以及当前类名称。

- 执行流程

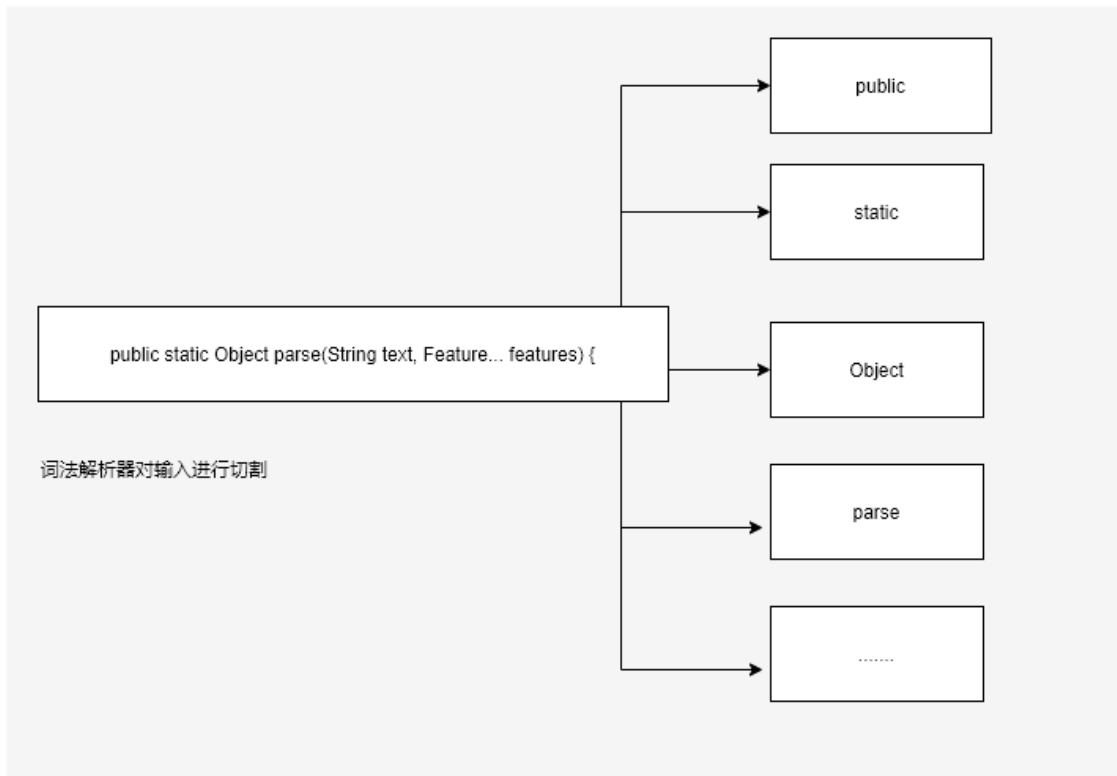


- 运行效果

```
2021-03-24 21:10:00,570 - root - INFO - =====Code Scanner Start=====
2021-03-24 21:10:00,570 - root - INFO - 开始扫描文件
2021-03-24 21:10:00,585 - root - INFO - private static void config(Properties properties)
2021-03-24 21:10:00,585 - root - INFO - {'package': 'com.alibaba.fastjson:', 'class': 'JSON'}
2021-03-24 21:10:00,585 - root - INFO - private static void config(Properties properties)
2021-03-24 21:10:00,585 - root - INFO - public static void setDefaultTypeKey(String typeKey)
2021-03-24 21:10:00,585 - root - INFO - public static Object parse(String text)
2021-03-24 21:10:00,585 - root - INFO - public static Object parse(String text, ParserConfig config)
2021-03-24 21:10:00,585 - root - INFO - public static Object parse(String text, ParserConfig config, Feature... features)
2021-03-24 21:10:00,585 - root - INFO - public static Object parse(String text, ParserConfig config, int features)
```

- 设计思路

在设计本项目中 Scanner 时，大量参考了编程语言中词法分析器的概念：在分析源码时，将空格作为分隔符，把输入分割为一个个算式。



在此过程中会定义很多的词法规范，此处我沿用一個常見稱呼，稱其為 token；因此，我們可以認為匹配的一種規範為一種 token，根據 token 的類型不同，我們可以匹配到不同的字符。

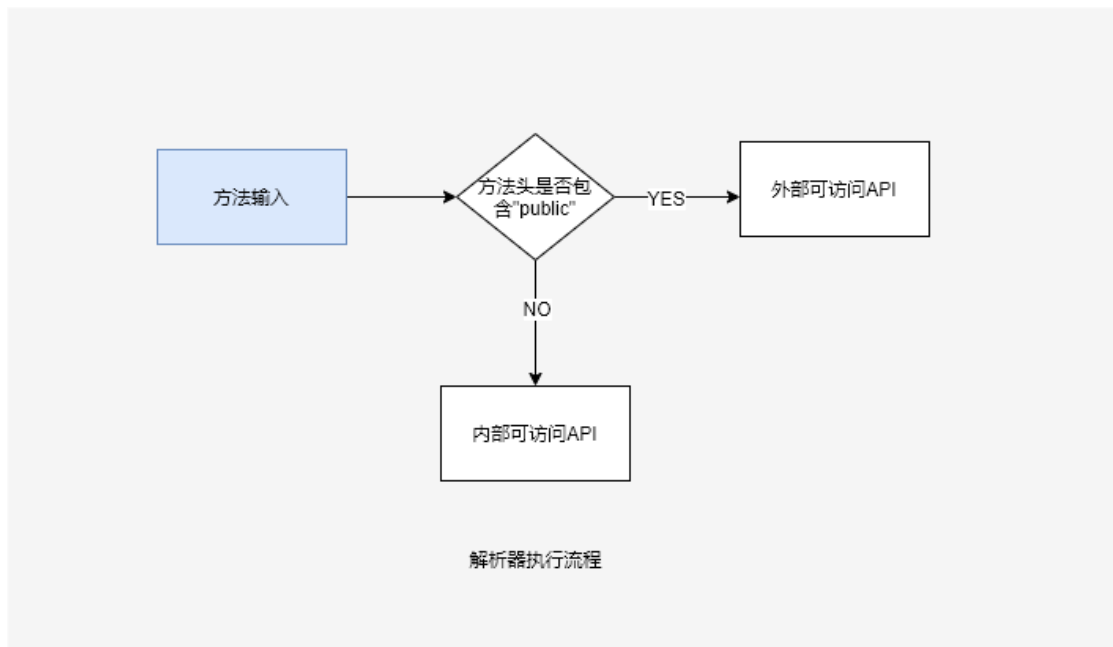
在本項目中，根據需求共設定了 12 種 token 類型：

**accesstoken** 訪問控制符標識 例如：**public private** 等  
**keytoken** 關鍵字標識 例如：**static final** 等  
**nexttoken** 下一行標識 針對方法頭不在一行的標識  
**nextmethodtoken** 下一行方法標識 對於有下一行標識的方法頭最後一行進行判定  
**imptoken** 方法必要標識 對於一個方法來講必須擁有的元素構成  
**invalidtoken** 無用字符標識 例如注釋、導包信息都是無用信息  
**interfacetoken** 接口標識 使用 **interface** 關鍵字標識的行信息  
**classtoken** 類標識 使用 **class** 關鍵字標識的行信息  
**packagetoken** 包標識 使用 **package** 關鍵字標識的行信息  
**leftsingletoken** { 符號標識  
**rightsingletoken** } 符號標識  
**allsingletoken** { 和 } 同時存在標識

## 2.Complier 解析器：對方法進行解析

以檢測 Java 語法訪問控制符來對方法進行解析，分別得出外部 API 和內部 API。

- 执行流程



- 运行效果

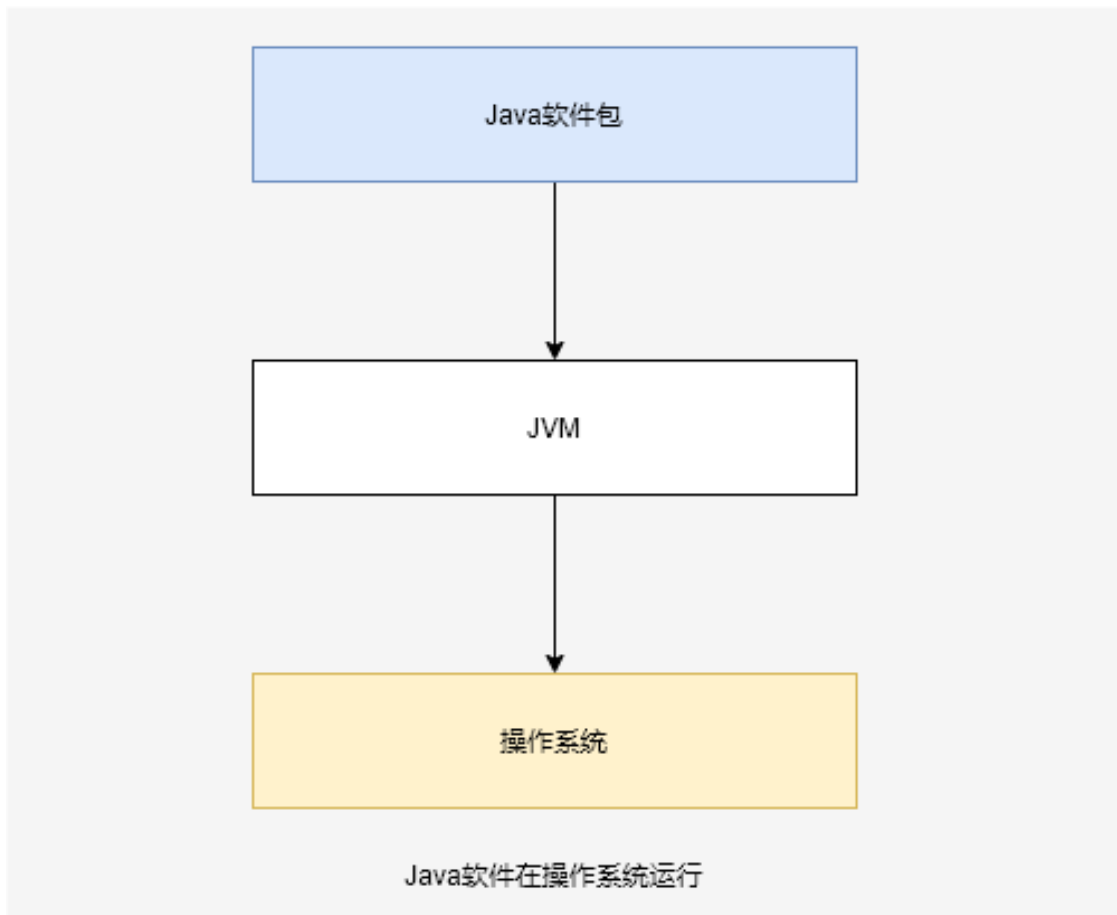
```
2021-03-24 21:10:00,588 - root - INFO - =====Code Compiler Start=====
2021-03-24 21:10:00,588 - root - INFO - unpublic -> private static void config(Properties properties)
2021-03-24 21:10:00,588 - root - INFO - unpublic -> private static void config(Properties properties)
2021-03-24 21:10:00,588 - root - INFO - public -> public static void setDefaultTypeKey(String typeKey)
2021-03-24 21:10:00,588 - root - INFO - public -> public static Object parse(String text)
2021-03-24 21:10:00,588 - root - INFO - public -> public static Object parse(String text, ParserConfig config)
2021-03-24 21:10:00,588 - root - INFO - public -> public static Object parse(String text, ParserConfig config, Feature... features)
2021-03-24 21:10:00,588 - root - INFO - public -> public static Object parse(String text, ParserConfig config, int features)
2021-03-24 21:10:00,588 - root - INFO - public -> public static Object parse(String text, int features)
2021-03-24 21:10:00,588 - root - INFO - public -> public static Object parse(String text, Feature... features)
2021-03-24 21:10:00,588 - root - INFO - public -> public static JSONObject parseObject(String text, Feature... features)
2021-03-24 21:10:00,588 - root - INFO - public -> public static JSONObject parseObject(String text, Feature... features)
```

- 设计思路

对于如何判别 Java 代码的内部 API 或者外部 API 这个问题，原本做了三种设想：

- 1.想尽办法获取到操作系统与软件包的交互联系
- 2.获取操作系统与 JVM 之间的兼容联系
- 3.直接根据访问修饰符以“内外部可访问”这个方向来进行判定

针对一、二点、对于利用 Java 语言实现的软件包与操作系统之间的关系，从实际状况上发现意义不大，因为 Java 代码并不是直接执行于操作系统之上的，而是要通过 Java 虚拟机，如图：



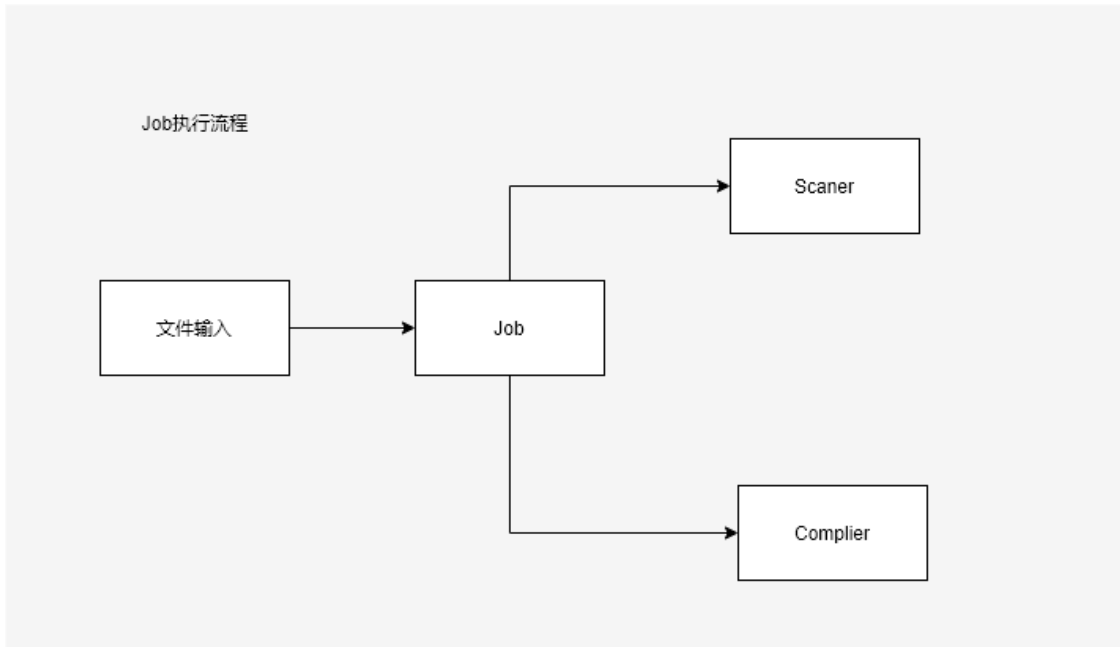
因此第一点不太容易实现且没有很大意义，第二点与题目要求有些差距，因此此处 Compiler 实现解析方法来判断外部可访问 API 和内部可访问 API 的方法就是参照第三点：通过访问修饰符来判定归类。

### 3.Job 任务：完成一次完整的源码解析操作

分别调用 Scanner 扫描器和 Compiler 解析器，对文件进行扫描和解析操作。



- 执行流程



## 4.JSON: 保存输出结果

将提取分析出来的结果以 JSON 格式的方式进行写出，写出格式如下：

```
{
  "info": {
    "package": "com.alibaba.fastjson.asm",
    "class": "TypeCollector"
  },
  "method": {
    "public": [
      "public TypeCollector(String methodName, Class<?>[] parameterTypes)",
      "public void visitAnnotation(String desc)",
      "public String[] getParameterNamesForMethod()",
      "public boolean matched()",
      "public boolean hasJsonType()"
    ],
    "unpublic": [
      "protected MethodCollector visitMethod(int access, String name, String desc)",
      "private boolean correctTypeName(Type type, String paramTypeName)"
    ]
  }
}
```

## 三、使用方法

### 1.项目环境构建

将项目克隆后导入 **Pycharm**

<https://gitee.com/openeuler2020/team-1186152014.git>

## 2 运行项目

### 3.2.1 运行 main.py 文件

```
C:\Users\erbin\AppData\Local\Programs\Python\Python39\python.exe E:/2020开发者大赛/代码托管/gitee/team-1186152014/api-scanner/main.py
=====Java API 提取器=====
| 队伍名称: mymk_one(1186152014) |
| 主要成员: 赵雯<withzhaowen@126.com> |
| 版本信息: v1.0 |
=====
请设置输入路径（输入后请回车，如果要退出请输入exit）
-- >
```

### 3.2.2 根据提示输入一个文件路径

此处文件路径为软件源码包所在的位置（可传入一个文件夹路径），如 FastJson 软件包源码在本机的绝对地址：

[E:\2020 开发者大赛\代码托管\gitee\team-1186152014\api-scanner\source\code\fastjson](#)

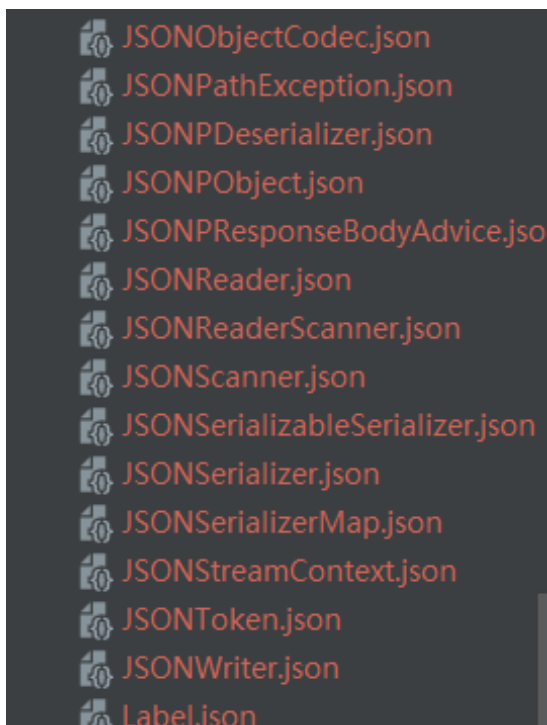
```
C:\Users\erbin\AppData\Local\Programs\Python\Python39\python.exe E:/2020开发者大赛/代码托管/gitee/team-1186152014/api-scanner/main.py
=====Java API 提取器=====

|  队伍名称: mymk_one(1186152014)  |
|  主要成员: 赵雯<withzhaowen@126.com>  |
|  版本信息: v1.0  |
|-----|

请设置输入路径（输入后请回车，如果要退出请输入exit）
-- > g:\2020开发者大赛\代码托管\gitee\team-1186152014\api-scanner\source\code\fast.json
```

### 3.2.3 控制台提示写入成功后，在 temp 目录中出现一些新的 Json 文件

```
2021-03-27 17:55:07,817 - root - INFO - unpublic -> private static CoderResult xflow(Buffer src, int sp, int sl, Buffer dst, int dp, int nb)
2021-03-27 17:55:07,817 - root - INFO - unpublic -> private CoderResult decodeArrayLoop(ByteBuffer src, CharBuffer dst)
2021-03-27 17:55:07,817 - root - INFO - unpublic -> protected CoderResult decodeLoop(ByteBuffer src, CharBuffer dst)
2021-03-27 17:55:07,817 - root - INFO - =====Code Compiler End=====
2021-03-27 17:55:07,817 - root - INFO - > 正在写入中.....
2021-03-27 17:55:07,817 - root - INFO - 写出路径为: E:\2020开发者大赛\代码托管\gitee\team-1186152014\api-scanner\temp\UTF8Decoder.json
2021-03-27 17:55:07,818 - root - INFO - > 第191个文件写入成功
2021-03-27 17:55:07,818 - root - INFO - 操作完毕
请设置输入路径（输入后请回车，如果要退出请输入exit）
-- >
```



- JSONObjectCodec.json
- JSONPathException.json
- JSONPDeserializer.json
- JSONPObject.json
- JSONPResponseBodyAdvice.json
- JSONReader.json
- JSONReaderScanner.json
- JSONScanner.json
- JSONSerializableSerializer.json
- JSONSerializer.json
- JSONSerializerMap.json
- JSONStreamContext.json
- JSONToken.json
- JSONWriter.json
- Label.json

这些文件就是通过工具提取到的 Java API 信息

## 四、参考资料

《自制编程语言》 - [日] 前桥和弥

<https://github.com/zsthampi/Compiler>