上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

BACHELOR'S THESIS

论文题目：上海交通大学学位论文 LaTeX 模板示例文档

学生姓名：某　某

学生学号：0010900990

专　　业：某某专业

指导教师：某某教授

学 院 (系)：某某系

上海交通大学

学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

上海交通大学

学位论文使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。

本学位论文属于 □**公开论文**

□**内部论文**，□ 1 年/□ 2 年/□ 3 年　解密后适用本授权书。

□**秘密论文**，____ 年（不超过 10 年）解密后适用本授权书。

□**机密论文**，____ 年（不超过 20 年）解密后适用本授权书。

（请在以上方框内打“✓”）

学位论文作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

上海交通大学学位论文 LaTeX 模板示例文档

摘　要

中文摘要应该将学位论文的内容要点简短明了地表达出来，应该包含论文中的基本信息，体现科研工作的核心思想。摘要内容应涉及本项科研工作的目的和意义、研究方法、研究成果、结论及意义。注意突出学位论文中具有创新性的成果和新见解的部分。摘要中不宜使用公式、化学结构式、图表和非公知公用的符号和术语，不标注引用文献编号。硕士学位论文中文摘要字数为 500 字左右，博士学位论文中文摘要字数为 800 字左右。英文摘要内容应与中文摘要内容一致。

摘要页的下方注明本文的关键词（4～6 个）。

关键词：上海交大，饮水思源，爱国荣校

A SAMPLE DOCUMENT FOR LaTeX-BASED SJTU THESIS TEMPLATE

ABSTRACT

Shanghai Jiao Tong University (SJTU) is a key university in China. SJTU was founded in 1896. It is one of the oldest universities in China. The University has nurtured large numbers of outstanding figures include JIANG Zemin, DING Guangen, QIAN Xuesen, Wu Wenjun, WANG An, etc.

SJTU has beautiful campuses, Bao Zhaolong Library, Various laboratories. It has been actively involved in international academic exchange programs. It is the center of CERNet in east China region, through computer networks, SJTU has faster and closer connection with the world.

Key words: SJTU, master thesis, XeTeX/LaTeX template

目　录

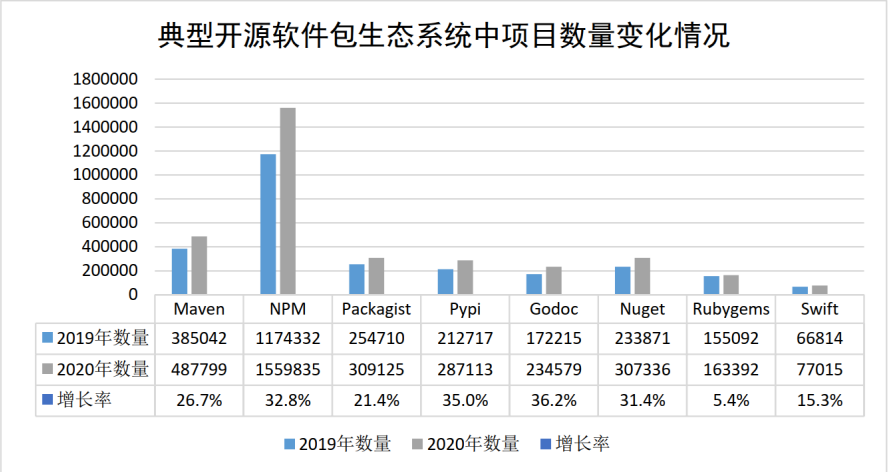
第一章 绪论 …… 1
1.1 软件供应链 …… 1
1.2 软件供应链安全 …… 1
1.3 安卓软件的供应链安全 …… 2
第二章 研究现状 …… 3
2.1 检测混淆库 …… 3
2.2 检测未知库 …… 3
2.3 检测已知库 …… 4
2.4 检测标准库的版本 …… 4
第三章 研究方法 …… 5
3.1 方法概述 …… 5
3.2 包的预处理 …… 5
3.2.1 Dex 与 Class 简介 …… 5
3.2.2 Apk 与 jar 的预处理 …… 5
3.3 构建特征树 …… 6
3.3.1 两级特征 …… 6
3.3.2 特征树的实现 …… 8
3.3.3 特征存储 …… 10
3.3.4 特征匹配 …… 11
参考文献 …… 15
附录 A Maxwell Equations …… 17
附录 B 绘制流程图 …… 18
致　谢 …… 19
学术论文和科研成果目录 …… 20
个人简历 …… 21

第一章　绪论

1.1　软件供应链

随着容器、微服务等新技术日新月异，开源软件成为业界主流形态，软件行业快速发展。现代软件大多数是被“组装”出来的，不是被“开发”出来的。据 Forrester 统计，软件开发中，80-90% 的代码来自于开源软件。因此，现代软件的源代码绝大多数是混源代码，由企业自主开发的源代码和开源软件代码共同组成。

根据奇安信代码安全实验室的检测与统计[1]，八个典型的开源软件包生态系统发展迅猛，呈现繁荣态势，包括 Maven、NPM、Packagist、Pypi、Godoc、Nuget、Rubygems、Swift。2019 年与 2020 年各开源软件包生态系统增长情况如图1–1：



	Maven	NPM	Packagist	Pypi	Godoc	Nuget	Rubygems	Swift
2019年数量	385042	1174332	254710	212717	172215	233871	155092	66814
2020年数量	487799	1559835	309125	287113	234579	307336	163392	77015
增长率	26.7%	32.8%	21.4%	35.0%	36.2%	31.4%	5.4%	15.3%

图 1–1　2019 和 2020 年八个典型开源软件包生态系统的增长情况

1.2　软件供应链安全

软件供应链的上游软件可能悄无声息地影响着下游产品。开源软件之间的依赖关系错综复杂，在开发过程中，开发者通常借助包管理程序实现自动管理，因此可能意识不到产品中包含里数量巨大的开源软件。一旦某个上游的开源软件被发现安全漏洞，软件开发者无法立即意识到漏洞同时被引入到了产品中，隐含里巨大的软件供应链安全风险。

2020 年 5 月，GitHub 披露了 Octopus Scanner 漏洞[2]，该漏洞是针对 Apache NetBeans IDE 项目的开源软件供应链攻击，影响到了 26 个开源项目。

2020 年 12 月，安全公司 FireEye 发现全球著名的网络安全管理软件供应商 SolarWinds 遭遇国家级 APT 团伙高度复杂的供应链攻击。该攻击在 SolarWinds 的一个数字签名组件 DLL 中插入后门，该后门通过 HTTP 协议与第三方服务器通信。

1.3 安卓软件的供应链安全

Appbrain[3]追踪了 450 个流行的库，统计结果显示它们在安卓生态系统中有着广泛的使用，广告库、社交网络库、以及手机设备分析库尤为受欢迎。如此广泛的第三方库使用在加速开发过程、避免重复造轮子的同时，也吸引着攻击者将目标向软件供应链上游移动，通过利用受欢迎的库的漏洞来达到攻击应用的目的。atvhunter 2-4。来自 Trend Micro 的安全研究团队披露百度提供的 SDK 中的 Moplus 包含的功能可能被恶意使用，以向用户设备植入后门[4]。这一处于软件供应链上游的漏洞已经流入超过 14000 款安卓 APP，可能使得约 1 亿用户处于黑客的攻击风险中。

2022 年 4 月 Google Play 商店内的安卓应用超过 260 万，3 月与 4 月新增应用数量均在 2 万左右，来自其他市场的应用更是不计其数。

如此数量的 APP 包含着不可忽视的供应链风险，但是由于 APP 包含着敏感信息或者具有商业价值的运行逻辑，大部分开发者基于安全和产权的考虑都会将产品进行混淆后再发布。这导致在对 APP 进行安全性检查时更加困难，识别混淆 APP 中引入的上游软件成为了亟待解决的问题。事实上，约 78% 的漏洞都是在间接的依赖中找到，可能带来的安全风险则更加难以发现[1]。

第二章　研究现状

2.1　检测混淆库

随着 APP 混淆技术的成熟，以第三方库能够被容易地区分为前提的方法已不适用，标识符被混淆为无意义的简短的字母组合，比如 *com.google* 可能被混淆为 *a.c*，无法提供关于库的任何信息。图2–1为一个代码混淆的示例，仅从名称无法获得任何关于包的信息。

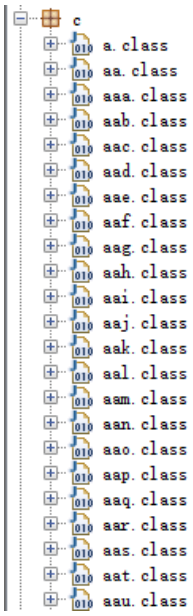


图 2–1　一款 360 软件的 apk 解压后得到的经过重命名混淆的 class 文件

T. Book 等人的工作通过白名单的方法检测 APP 内的第三方库[5]，但这类方法显然无法解决标识符重命名的问题。PEDAL[6]借助机器学习方法，从 SDK 中提取代码特征，并使用包之间的关系信息训练了分类器来识别第三方库。

2.2　检测未知库

一些研究工作提出了在没有已知第三方代码的数据库知识情况下检测 APP 组成成分的方法。此类方法通常首先把开发者代码与第三方代码进行分类，再将第三方代码聚类成不同的组件，组件即一个可能的库的候选，进一步评估候选之间的相似度，当超过相似度阈值的候选的出现次数足够多时就认为找到了一个库。

如 Chen 等人[7]从大量的 APP 中获取库，进行聚类和检测然而这一方法在混淆的情况下表现不佳，因为其假设不同 APP 中包含的库的相同实例拥有相同的包名，混淆打破了这一基本的假设。

为解决包名混淆问题，LibRadar[8]使用特征哈希的方法，不需要基于包名的聚类，而是借助包中的目录结构来识别库的候选，具体来说是将一个候选表示为一个目录树的结构。这引入一个新的假设，即包的结构在混淆过程中不改变。但混淆工具可以将不同的包合并为一个包，很容易打破这一假设。

WuKong[9]和 AnDarwin[10]用控制流图和 API 数量来定义哈希特征，用来计算各候选库的相似度。考虑到混淆工具可能修改一个方法的控制流图，或者移除在 APP 运行中未真正使用的方法，哈希的质量影响着这两类方法的表现性能。

2.3 检测已知库

基于已知库的检测要求关于现存库的知识，如库的基本信息、哈希特征等，在混淆 APP 第三方库识别的场景下，用构建知识数据库的代价换取了更好的表现。

具有代表性的一个工具是 LibScout[11]，用包的结构以及类的哈希作为特征，进行 APP 与数据库中第三方库的匹配，在控制流篡改和包/类/描述符重命名情况下依然有效。但是随着数据库中的标准库代码特征增多，哈希特征的计算也应当考虑更多信息，导致特征生成时间与匹配时间增加。

2.4 检测标准库的版本

现有工作中以版本为目标实现精确检测的并不多，AdDetect[12]仅能够区分广告和非广告的库，基于聚类的方法如 LibRadar[8]，LibD[13]等都没有声明能够检测库的特定版本。

实现版本的检测仍面临着很多问题：

1. 需要处理庞大的数据集。第三方库本身就纷繁复杂，如果再将各个版本考虑进去，将导致需要处理的数据成倍增长。
2. 缺乏精确的表示。一个库的不同版本可能差异微小，如何找到合适的特征来区分这一差别非常关键。
3. 代码混淆的干扰。代码混淆同样会导致库的代码发生改变，这种改变是由不同库引起还是由同一库的不同版本引起，需要被准确的区分。

第三章　研究方法

在参考了多篇文献后，我提出了一种适用于包的结构混淆、包/类/标识符重命名场景的，基于已知标准库的数据库，利用两类信息生成粗粒度/细粒度两级哈希特征的安卓应用第三方库及其特定版本的检测方法。

3.1　方法概述

此方法不依赖于包中的目录结构以及各级名称，因此可以抵抗结构混淆以及重命名混淆，包括了四个步骤：

1. 预处理 jar、aar 和 apk。将来自 Maven 仓库的 jar 包、aar 包以及待检测 apk 处理成便于构建树结构的形式。
2. 构建特征树。根据上一阶段输出，将每个包作为根节点构建特征树，该包内的所有类，不论是根包的类还是子包的类，一律作为树的中间层节点，各类的方法作为叶子节点。特征分为粗粒度、细粒度两级特征。粗粒度特征为方法的描述符的返回值以及参数类型，细粒度特征为该方法的字节码，首先生成叶子节点的两级特征，再利用叶子节点生成中间层节点即类节点的特征。
3. 构建数据库与匹配。根据以上特征生成方法，计算 Maven 仓库中的标准库的特征，并存储到数据库中。对待检测 APP，首先生成粗粒度特征，确定所包含的库，再根据细粒度特征，确定各库的具体版本。

3.2　包的预处理

3.2.1　Dex 与 Class 简介

DEX 文件　是 Android 系统中的一种文件，是一种特殊的数据格式，能够被 Dalvik 虚拟机识别并加载执行，类似于 Windows 上的 EXE 可执行文件。将 APK 安装包解压后得到的文件就包含了 DEX 文件，它记载了应用程序的全部操作指令以及运行时数据。当 java 程序编译成 class 文件后，还需要使用 dx 工具将所有的 class 文件整合到一个 DEX 文件里，目的是其中各个类能够共享数据，在一定程度上降低了冗余，同时也使文件结构更加紧凑。DEX 文件大小通常是传统 jar 包的 50% 左右。

CLASS 文件　是能够被 java 虚拟机识别，加载并执行的文件格式，通过 javac 程序可以从 java 源文件生成 class 文件。class 文件记录了一个类文件的所有信息，不仅包含了 java 源代码中的信息，还包括了 this、super 等关键字的信息。作为一种 8 位字节的二进制六文件，class 中的数据按顺序紧密排列，没有间隙，从而让 JVM 加载更加迅速，每一个类、接口或者枚举都单独占据一个 class 文件。

3.2.2　Apk 与 jar 的预处理

预处理流程如图3–1所示。

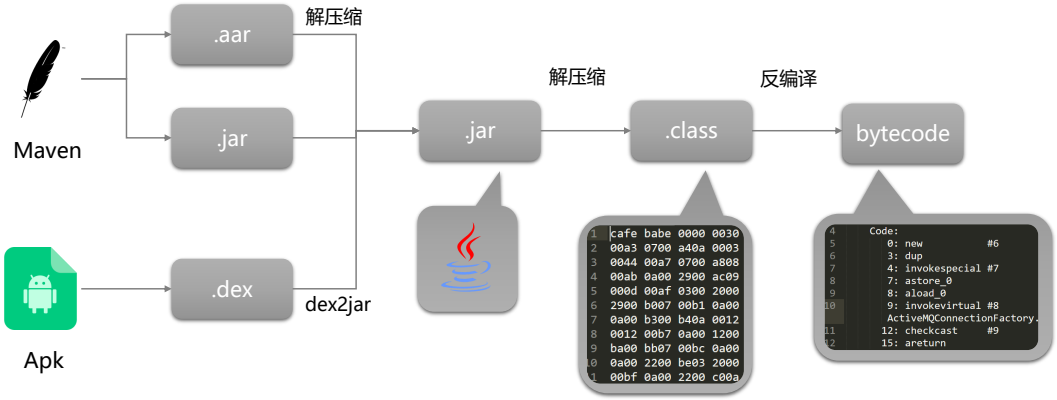


图 3–1 Apk 与标准库转换到 java 字节码的预处理流程

Apk 预处理： 从应用市场获取待检测的应用的安装包，即 apk 文件。Apk 文件本质上为 zip 格式的压缩文件，解压后可以得到 dex 文件。借助 dex 到 jar 的转换工具 dex2jar[14]将 DEX 文件转换为包含多个 class 文件。此处的 class 文件通常是经过软件开发者混淆的，大部分类的名称和函数的名称经过了混淆处理，不能提供有效信息，转换得到的包的目录结构也不可靠。

Jar 预处理： Maven 社区提供的中央仓库[15]包含了大量常用的库，包括绝大多数流行的 Java 开源构件，源码，许可证信息等，一般来说简单的 Java 项目依赖的构建都可以满足，因此选择 Maven 仓库中的部分 jar 包作为构建数据库的基础。我利用爬虫从 Maven 中央仓库中获取了大量的 jar 包或 aar 包，aar 包需要先解压一次获得其中的 jar 包，将 jar 包解压就可以获得该标准库的各个 class 文件。在爬取标准库的过程中，需要跳过一些带有 javadoc 或者 sources 字样的链接，这表示该包为一个说明文档包或者源代码包，在此次工作中用不到。

字节码的生成： Java 字节码是一种程序的低级表示，能够直接被 java 虚拟机所理解和翻译，以实现 java 跨平台的特性。具有二进制格式的 class 文件，实质上就是 java 的字节码，属于独立于平台和操作系统的指令集。为了删除 class 文件中的常量信息，我使用 javap 程序将 class 文件反编译为具有可读性的助记符形式的字节码，可以清楚地区分开描述符，代码，常量池等部分，并存储为文本文件供后续使用。

3.3 构建特征树

3.3.1 两级特征

如果特征的计算方法过于复杂，将会导致在获取 apk 特征时用时过长，因此本方法中采用了两级特征——粗粒度特征与细粒度特征，来解决这一问题，同时实现第三方库的版本检测。粗粒度特征计算方法简短、生成速度快、相应的精准度有所下降，用于确定待测 apk 中

的包是数据库中的哪一个标准库。细粒度特征计算方法复杂、生成速度慢、但可以精确到单条字节码操作指令的层面。对于细粒度特征而言，即便同一个库的版本不同所导致的细微代码差异也能够体现出来，因此可以在包成功匹配的情况下进一步匹配具体版本。

(1) 粗粒度特征

粗粒度特征由函数的描述符生成，由于函数描述符包含了函数名、参数名，极易受到重命名混淆影响，因此我将名称部分删去，以*返回值类型（参数 1 类型，参数 2 类型，...，参数 n 类型）*的字符串作为描述符，计算其 md5 哈希值，得到该方法的签名，一个例子如表3–1所示。尽管此签名可能在多个包中，乃至一个包中的不同类中出现，但是结合该类下的各个方法的签名，方法的序列，可以减少碰撞的概率，用于初步表示一个类。如表3–4所示，标准库 *org.codehaus.avtivemq* 的两个类 *ActiveMQMessageConsumer* 和 *BrokerClientImpl* 在方法“*public java.lang.String toString()*”上发生了碰撞，但是在其他方法上有很大差异，方法的总数也不同，从而类的层面的特征有所区别。

表 3–1 方法描述符的处理

原始描述符	public static ActiveMQConnection makeConnection(String user, String password, String uri)
删除名称后的描述符	public static org.codehaus.activemq.ActiveMQConnection makeConnection(java.lang.String, java.lang.String, java.lang.String)
md5 哈希值	befc542005082b1940176d89035826ab

表 3–2 标准库 org.codehaus.activemq 中的两个类包含方法（部分）的情况

描述符	ActiveMQMessageConsumer	BrokerClientImpl
public java.lang.String toString();	✓	✗
protected long getStartTime();	✓	✗
protected void setBrowser(boolean);	✓	✗
public void updateBrokerCapacity(int);	✗	✓

(2) 细粒度特征

细粒度特征基于函数的字节码生成。*ActiveMQConnection* 类的一个方法 *makeConnection* 的字节码如下所示：

```
public static org.codehaus.activemq.ActiveMQConnection makeConnection(
java.lang.String) throws javax.jms.JMSException;
  Code:
     0: new           #6
     3: dup
     4: aload_0
     5: invokespecial #10
```

```
 8: astore_1
 9: aload_1
10: invokevirtual #8
13: checkcast     #9
16: areturn
```

Java 字节码是基于堆栈结构的，上面字节码“Code”部分的每一行对应于一条操作指令，仅仅表示对堆栈的操作，而不包含操作数。源代码中所定义的常量、变量名等静态成员存放在常量池中，因而避免了名称混淆的问题。每一条操作指令对应于一个十六进制的操作码，表3–3展示了部分对应关系及其说明。

表 3–3 Java 字节码助记符与十六进制操作码对应关系（部分）

助记符	操作码	说明
new	0xbb	创建一个对象，并将其引用值压入栈顶
dup	0x59	复制栈顶数值，并将复制值压入栈顶
aload_0	0x2a	将第一个引用类型本地变量推送至栈顶
invokespecial	0xb7	调用超类构造方法，实例初始化方法，私有方法
astore_1	0x4c	将栈顶引用型数值存入第二个本地变量
aload_1	0x2b	将第一个引用类型本地变量推送至栈顶
invokevirtual	0xb6	调用实例方法
checkcast	0xc0	检验类型转换，检验未通过将抛出异常
areturn	0xb0	从当前方法返回对象引用

3.3.2 特征树的实现

(1) 字节码特征提取

如图3–2所示，字节码解析器布置在树的下方，其接收字节码文件，从文件头开始读入，利用正则表达式匹配每一个函数的描述符。当匹配成功时，表明一个函数的读入即将开始，则解析器依次读取函数描述符和操作指令序列，并将原始特征记录为一个特征对。当读至文件尾时，该 class 文件代表的类的所有成员函数全部读取完毕，将各特征对传递给上方的树。

(2) 方法层

方法实现为 *mNode*（Method Node）类。根据提取出来的字节码的原始特征，对于每一对粗/细粒度原始特征，生成一个方法节点，对原始特征进行处理。方法节点记录了原始的函数描述符，文件路径等基本信息，同时生成了处理后的特征的 MD5 哈希值，作为此方法的两级特征。表为 *ActiveMQConnection* 类中方法 *createSession* 的各个属性值。

(3) 类层

类实现为 *cNode*（Class Node）类，类层根据该 class 文件中包含的方法，将各个方法对应的 *mNode* 节点作为自己的子节点。不同类的方法数大相径庭，有的类没有方法，只有数据成员，有的类则包含几十甚至上百个函数成员。再考虑到类的不同版本之间可能差异很

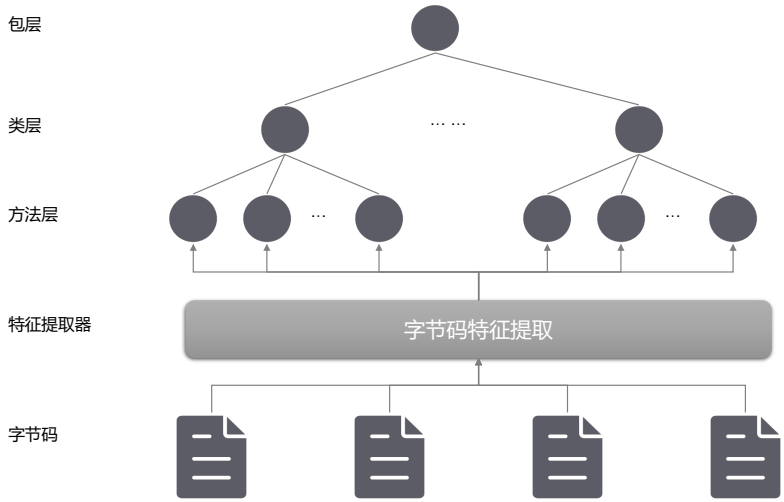


图 3–2 使用特征提取器从字节码文件中获取特征并存储为树型数据结构

小，仅仅体现在个别方法的几个函数上，类的特征必须能体现函数成员的特征。因此我采用了模糊哈希的方法，算法如3–1所示，将类下各方法的特征进行排序、连接，计算模糊哈希值作为类的粗/细粒度特征。

此处模糊哈希的用法与 ATVHunter[16]有相似之处，都是为了降低代码混淆对最后特征带来的影响，但是 ATVHunter 的目标是减少部分指令改变对函数签名的影响，而此处是减少部分函数改变对类的特征的影响。如图3–3所示，模糊哈希的应用步骤为：

1. 将类下各方法的特征连接成一个序列。
2. 使用滑动窗口（滚动哈希）将序列分割成不同的切片。
3. 对每个切片，计算其 MD5 哈希值，取最后 6 位作为压缩映射值。
4. 最后将所有映射值连接起来作为最终值。

表 3–4 标准库 org.codehaus.activemq 中的两个类包含方法（部分）的情况

方法	org.codehaus.activemq.ActiveMQConnection$createSession
描述符特征	public javax.jms.Session(boolean, int)
字节码指令序列	2a b6 2a b6 bb 59 2a 1b 99 03 a7 1c b7 b0
粗粒度特征	d8a58a75cf9b3272e8736cd74256fdc7
细粒度特征	45422e23d690d166bc6dd144f226076f

(4) 包层

包实现为 *pNode*（Package Node）类，是一个特征树的根节点。包层不产生特征，用于管理属于该包下的各个类。为了解决 APK 产生的包中可能存在目录结构的混淆问题，每一个包都采用根包作为唯一的根节点，其下包含的子包以及子包中的二级子包等所拥有的类

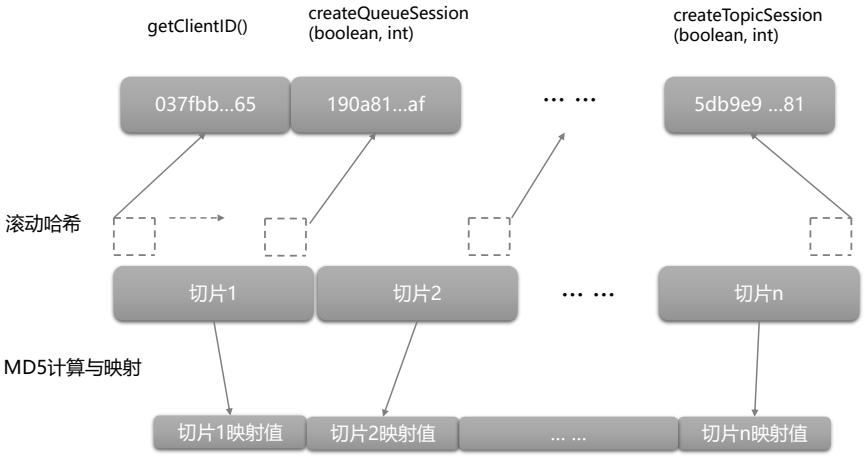


图 3–3 使用特征提取器从字节码文件中获取特征并存储为树型数据结构

算法 3–1 类的模糊哈希值计算

Data: 类节点 cNode
Result: 类的模糊哈希值 Fuzzy_hash

```
initialization;
sort(cNode.getMethods());
Sequence←null;
for mNode ∈ cNode.getMethods() do
    Sequence ← Sequence cascade mNode.getFeature();
end
Slices←rolling_hash(Sequence);
Fuzzy_hash←null;
for slice ∈ Slices do
    F_hash_slice←mapping_function(MD5(slice));
    Fuzzy_hash←Fuzzy_hash cascade F_hash_slice;
end
```

都归为根包的类。

最后将从包到方法的各个节点封装为一个 *javaTree* 类，负责统筹管理从文件的筛选到各级节点的生成等诸多工作。

3.3.3 特征存储

为了便于特征的查找与管理，我将哈希树输出的特征存放在 MySQL 数据库中，步骤如下：

1. 设计数据库 scheme：tpl 如表3–5所示，在 tpl 下创建表 maven，存储从 Maven 仓库获取的标准库。

2. 遍历本地爬取到的 Maven 仓库，当文件夹中包含 *META-INF* 时表明当前目录为一个根包的目录。
3. 在根包的目录上构建哈希树，解析目录中的文件，形成该包下各类、各方法的特征。
4. 用插入语句将特征存储到数据库中

表 3–5 MySQL 数据库中 tpl 的 scheme

字段	package	class	method	coarse feature	fine feature
数据类型	varchar(255)	varchar(255)	varchar(1024)	varchar(255)	varchar(255)

3.3.4 特征匹配

表 3–6 标准库与安卓应用的相关符号及说明

符号	说明
T_{sim}	相似度阈值，相似度超过此值的两个特征认为互相匹配
$feature_{lib}$	来源于标准库的一个类或者方法的特征
$feature_{app}$	来源于安卓应用的一个类或者方法的特征
C_{lib}	标准库所包含的类的集合
C_{app}	安卓应用所包含的类的集合
M_{class}	类所包含的方法的集合
$Similarity_p$	两个包之间的相似度
$Similarity_c$	两个类之间的相似度

(1) 相似度

为了表征标准库与来自 Apk 的包的相似程度，我引入了相似度来量化这一概念。相似度适用于不同来源（Maven/APP）的类或方法的匹配，是一个介于 0 和 1 之间的数，1 表示完全匹配。若要判断 APP 所包含的包是否存在于数据库中，需要查看数据库中是否存在足够数量的类，使得这些类的特征值与 APP 中类的特征值的相似度都超过了一定阈值。由于类的特征值是根据方法的特征值生成的，因此类的相似程度能过说明其内各个方法的相似程度。类和方法的特征均为字符串序列表示的哈希值，因此用编辑距离来计算两个序列之间的相似度：

$$Similarity(feature_{lib}, feature_{app}) = \frac{edit\_distance(feature_{lib}, feature_{app})}{max\{length(feature_{lib}), length(feature_{app})\}} \quad (3\text{–}1)$$

对于待定的阈值 T_{sim}，定义来自标准库的特征 $feature_{lib}$ 与来自 APP 的特征 $feature_{app}$ 匹配，如果满足：

$$Similarity(feature_{lib}, featureapp) \geq T_{sim} \quad (3\text{–}2)$$

同时，标准库的规模也应该纳入考虑范畴。一些大规模的类、或者相似的类，可能在部分方法上有相似之处，在实现逻辑上有相同点，因此不能仅仅因为成功匹配其下的一部分类

就将其所谓候选库。而一些小规模的类，方法数可能很少，一定数量的类匹配就说明其有很大概率就是 APP 所使用到的库。一种简明的相似度计算方法如下：

$$Similarity_p(lib, app) = \frac{|\{c_1, c_2, \ldots, c_n\}|}{|C_{lib}|} \in [0, 1] \tag{3–3}$$

其中，c_i 满足：

$$(a)\ c_i \in C_{lib} \tag{3–4a}$$

$$(b)\ \exists\, c' \in C_{app} Similarity(c_i, c') \geq T_{sim} \tag{3–4b}$$

类似地，用 $Similarity_c$ 表示方法相似度，也可以在方法粒度上实现更为精确的匹配：

$$Similarity_c(class_{lib}, calss_{app}) = \frac{|\{m_1, m_2, \ldots, m_n\}|}{|M_{class_{lib}}|} \in [0, 1] \tag{3–5}$$

其中，m_i 满足：

$$(a)\ m_i \in M_{class_{lib}} \tag{3–6a}$$

$$(b)\ \exists\, c' \in M_{class_{app}} Similarity(m_i, m') \geq T'_{sim} \tag{3–6b}$$

(2)　匹配策略

来自 Maven 中央仓库的标准库数量巨大，一个包中可能包含多个子包，包的版本数可能超过 50 个。一种朴素的匹配方法是：将来自 Maven 的标准库与来自 App 的包两两匹配，计算其下类的相似度，但这会产生以亿为单位的配对数，耗时严重。因此我引入了如图3–4所示的两种策略来加速匹配的过程，分别是两级特征与优先级队列。

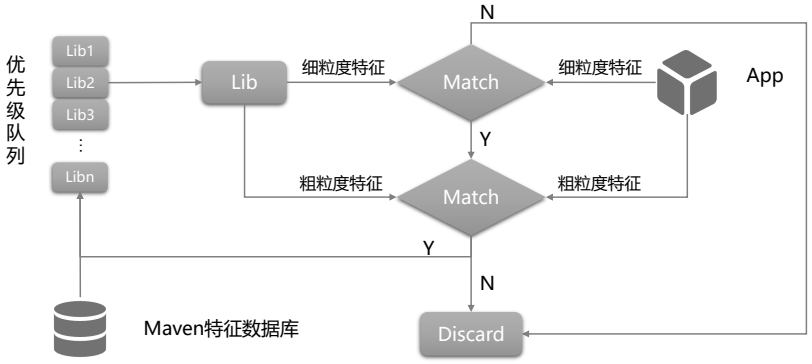


图 3–4　数据库匹配的两种加速策略——两级特征与优先级队列

两级特征策略 包含粗粒度、细粒度两种计算方法复杂度不同、精确度不同的特征，思想为首先利用粗粒度特征初步缩小匹配范围，即 App 中使用到了 Maven 的哪些标准库，再用细粒度特征确定这些库的具体版本。一个库如果没有通过粗粒度筛选，说明其方法的数量、各方法的参数数量、返回值及参数的类型这些特征与 App 存在较大差异，因此它几乎不可能在 App 中使用到，对这样的库进行具体版本的匹配也是没有意义的。对于通过粗粒度筛选的库，说明它在方法层面与 App 中的包有一定相似程度，此时再通过基于字节码生成的细粒度特征来精准匹配，不同版本所带来的细微差异也能够体现在特征中。

优先级队列策略 基于一个“局部性”假设：前一个来自 App 的类匹配完毕时，新到来的类有较大可能与前一个类处于同一个包中。

这一假设在直观上是合理的，因为在匹配的过程中，我按照 App 目录中的文件顺序进行处理，相邻的类处于同一个目录下。尽管 App 可能经过了目录结构的混淆处理，即将来自不同包的类混入到一个目录下，但除非所有的包都做了混淆且每一个类都被单独打乱路径而非以多个类为批次操作，否则此假设在大多数情况下都成立。

为了充分利用这一假设，对待匹配的 App 设置一个优先级队列。初始时优先级队列为空，从数据库中按序取出 K 个类的记录加入到队列当中。队列中元素的优先级 $Priority$ 定义为与当前待匹配 App 的类的相似度，其中细粒度特征的相似程度由于包含了更多的原始信息而被赋予更大的权重：

$$\begin{aligned} Priority(c_{lib}, c_{app}) = 0.3 \times Similarity(feature_{c_{lib}}^{coarse}, feature_{c_{app}}^{coarse}) \\ + 0.7 \times Similarity(feature_{c_{lib}}^{fine}, feature_{c_{app}}^{fine}), \\ c_{lib} \in C_{lib}, c_{app} \in C_{app} \end{aligned} \tag{3–7}$$

匹配算法如3–2所示，依次将优先级队列中的记录取出，计算与待匹配类的相似度，进一步计算优先级，按照优先级从大到小重排队列。队列中的记录全部使用完毕后，从数据库中取出一条新的记录，计算相似度与优先级，并插入到队列的合适位置，如果优先级小于队列尾部记录的优先级则不进入队列。以此类推，将数据库中的所有记录进行匹配。当前待测类匹配流程完毕后，从 App 中取出下一个待测类，首先与优先级队列中的记录匹配，队列匹配完毕后再从数据库中取得记录，按照此方法将 App 中的所有类进行匹配。

对于某次匹配结果：

1. 如果完全匹配，即相似度为 1，则中止当前流程，直接记录此类属于数据库中类所在的标准库。
2. 如果相似度小于 1，则以优先级队列中优先级最高的记录为最佳匹配，并将该记录来自的标准库作为当前待测类的识别结果。

算法 3–2 标准库与 App 的匹配算法

Data: 标准库特征数据库 C_{lib}，App 特征数据库 C_{app}
Result: 匹配结果 $Package$

Initialize priority queue $Queue_p$;
Initialize mapping relation $Package \leftarrow empty\ dictionary$;
for $c_{app} \in C_{app}$ **do**
  if $isEmpty(Queue_p)$ **then**
    for $c_{lib} \in C_{lib}$ **do**
      $priority \leftarrow Priority(c_{lib}, c_{app})$;
      $Queue_p$ėn$Queue(c_{lib}, priority)$;
      if *priority=1* **then**
        $break$
      else
        $continue$;
      end
    end
  else
    for $c_{lib} \in Queue_p$ **do**
      $priority \leftarrow Priority(c_{lib}, c_{app})$;
      if *priority=1* **then**
        $break$
      else
        $continue$;
      end
    end
    for $c_{lib} \in C_{lib}$ **do**
      $priority \leftarrow Priority(c_{lib}, c_{app})$;
      $Queue_p.enQueue(c_{lib}, priority)$;
      if *priority=1* **then**
        $break$
      else
        $continue$;
      end
    end
  end
  $c_{lib}^{match} \leftarrow Queue_p.first()$;
  $Package[c_{app}] \leftarrow c_{lib}^{match}.get\_package()$;
end

参考文献

[1] 2021 中国软件供应链安全分析报告[Z]. https://www.qianxin.com/news/detail?news_id=1108. Accessed May 9, 2022.

[2] LAB G S. The Octopus Scanner Malware: Attacking the open source supply chain[Z]. https://securitylab.github.com/research/octopus-scanner-malware-open-source-supply-chain/. Accessed May 9, 2022.

[3] AppBrain. Android library statistics[Z]. https://www.appbrain.com/stats/libraries. Accessed May 9, 2022.

[4] Thehackernews.com. Backdoor in Baidu Android SDK Puts 100 Million Devices at Risk[Z]. https://thehackernews.com/2015/11/android-malware-backdoor.html. Accessed May 9, 2022.

[5] BOOK T, PRIDGEN A, WALLACH D S. Longitudinal analysis of android ad library permissions[J]. arXiv preprint arXiv:1303.0857, 2013.

[6] LIU B, LIU B, JIN H, et al. Efficient privilege de-escalation for ad libraries in mobile apps[C]//Proceedings of the 13th annual international conference on mobile systems, applications, and services. 2015: 89-103.

[7] CHEN K, WANG X, CHEN Y, et al. Following devil's footprints: Cross-platform analysis of potentially harmful libraries on android and ios[C]//2016 IEEE Symposium on Security and Privacy (SP). 2016: 357-376.

[8] MA Z, WANG H, GUO Y, et al. Libradar: fast and accurate detection of third-party libraries in android apps[C]//Proceedings of the 38th international conference on software engineering companion. 2016: 653-656.

[9] WANG H, GUO Y, MA Z, et al. Wukong: A scalable and accurate two-phase approach to android app clone detection[C]//Proceedings of the 2015 International Symposium on Software Testing and Analysis. 2015: 71-82.

[10] CRUSSELL J, GIBLER C, CHEN H. Andarwin: Scalable detection of android application clones based on semantics[J]. IEEE Transactions on Mobile Computing, 2014, 14(10): 2007-2019.

[11] BACKES M, BUGIEL S, DERR E. Reliable third-party library detection in android and its security applications[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016: 356-367.

[12] NARAYANAN A, CHEN L, CHAN C K. Addetect: Automated detection of android ad libraries using semantic analysis[C]//2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP). 2014: 1-6.

[13] LI M, WANG W, WANG P, et al. Libd: Scalable and precise third-party library detection in android markets[C]//2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). 2017: 335-346.

[14] https://github.com/wangzifan184/dex2jar. Accessed May 9, 2022.

[15] https://repo.maven.apache.org/. Accessed May 9, 2022.

[16] ZHAN X, FAN L, CHEN S, et al. Atvhunter: Reliable version detection of third-party libraries for vulnerability identification in android applications[C]//2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). 2021: 1695-1707.

附录 A　Maxwell Equations

选择二维情况，有如下的偏振矢量：

$$\mathbf{E} = E_z(r,\theta)\hat{\mathbf{z}}, \tag{A–1a}$$

$$\mathbf{H} = H_r(r,\theta)\hat{\mathbf{r}} + H_\theta(r,\theta)\hat{\boldsymbol{\theta}}. \tag{A–1b}$$

对上式求旋度：

$$\nabla\times\mathbf{E} = \frac{1}{r}\frac{\partial E_z}{\partial\theta}\hat{\mathbf{r}} - \frac{\partial E_z}{\partial r}\hat{\boldsymbol{\theta}}, \tag{A–2a}$$

$$\nabla\times\mathbf{H} = \left[\frac{1}{r}\frac{\partial}{\partial r}(rH_\theta) - \frac{1}{r}\frac{\partial H_r}{\partial\theta}\right]\hat{\mathbf{z}}. \tag{A–2b}$$

因为在柱坐标系下，$\overline{\overline{\mu}}$ 是对角的，所以 Maxwell 方程组中电场 $\mathbf{E}$ 的旋度：

$$\nabla\times\mathbf{E} = \mathrm{i}\omega\mathbf{B}, \tag{A–3a}$$

$$\frac{1}{r}\frac{\partial E_z}{\partial\theta}\hat{\mathbf{r}} - \frac{\partial E_z}{\partial r}\hat{\boldsymbol{\theta}} = \mathrm{i}\omega\mu_r H_r\hat{\mathbf{r}} + \mathrm{i}\omega\mu_\theta H_\theta\hat{\boldsymbol{\theta}}. \tag{A–3b}$$

所以 $\mathbf{H}$ 的各个分量可以写为：

$$H_r = \frac{1}{\mathrm{i}\omega\mu_r}\frac{1}{r}\frac{\partial E_z}{\partial\theta}, \tag{A–4a}$$

$$H_\theta = -\frac{1}{\mathrm{i}\omega\mu_\theta}\frac{\partial E_z}{\partial r}. \tag{A–4b}$$

同样地，在柱坐标系下，$\overline{\overline{\epsilon}}$ 是对角的，所以 Maxwell 方程组中磁场 $\mathbf{H}$ 的旋度：

$$\nabla\times\mathbf{H} = -\mathrm{i}\omega\mathbf{D}, \tag{A–5a}$$

$$\left[\frac{1}{r}\frac{\partial}{\partial r}(rH_\theta) - \frac{1}{r}\frac{\partial H_r}{\partial\theta}\right]\hat{\mathbf{z}} = -\mathrm{i}\omega\overline{\overline{\epsilon}}\mathbf{E} = -\mathrm{i}\omega\epsilon_z E_z\hat{\mathbf{z}}, \tag{A–5b}$$

$$\frac{1}{r}\frac{\partial}{\partial r}(rH_\theta) - \frac{1}{r}\frac{\partial H_r}{\partial\theta} = -\mathrm{i}\omega\epsilon_z E_z. \tag{A–5c}$$

由此我们可以得到关于 E_z 的波函数方程：

$$\frac{1}{\mu_\theta\epsilon_z}\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial E_z}{\partial r}\right) + \frac{1}{\mu_r\epsilon_z}\frac{1}{r^2}\frac{\partial^2 E_z}{\partial\theta^2} + \omega^2 E_z = 0. \tag{A–6}$$

附录 B　绘制流程图

图 B–1 是一张流程图示意。使用 tikz 环境，搭配四种预定义节点（`startstop`、`process`、`decision`和`io`），可以容易地绘制出流程图。

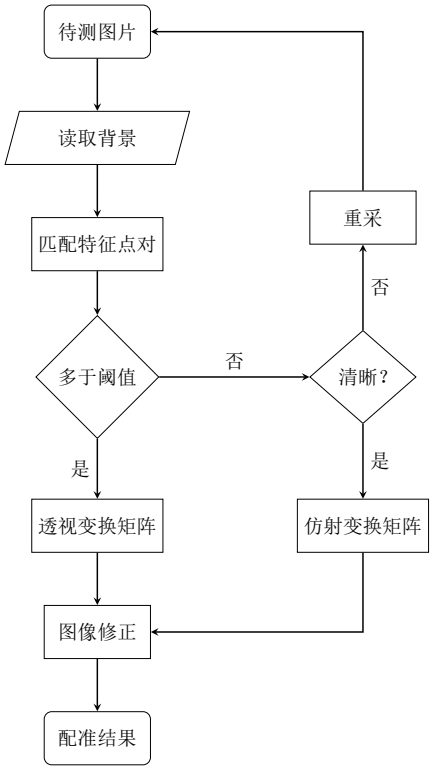


图 B–1　绘制流程图效果

Figure B–1　Flow chart

致　谢

感谢那位最先制作出博士学位论文 LaTeX 模板的交大物理系同学！
感谢 William Wang 同学对模板移植做出的巨大贡献！
感谢 @weijianwen 学长一直以来的开发和维护工作！
感谢 @sjtug 以及 @dyweb 对 0.9.5 之后版本的开发和维护工作！
感谢所有为模板贡献过代码的同学们, 以及所有测试和使用模板的各位同学！
感谢 LaTeX 和 SJTUThesis，帮我节省了不少时间。

学术论文和科研成果目录

学术论文

[1] Chen H, Chan C T. Acoustic cloaking in three dimensions using acoustic metamaterials[J]. Applied Physics Letters, 2007, 91:183518.

[2] Chen H, Wu B I, Zhang B, et al. Electromagnetic Wave Interactions with a Metamaterial Cloak[J]. Physical Review Letters, 2007, 99(6):63903.

专利

[3] 第一发明人，“永动机”，专利申请号 202510149890.0

个人简历

基本情况

某某，yyyy 年 mm 月生于 xxxx。

教育背景

- yyyy 年 mm 月至今，上海交通大学，博士研究生，xx 专业
- yyyy 年 mm 月至 yyyy 年 mm 月，上海交通大学，硕士研究生，xx 专业
- yyyy 年 mm 月至 yyyy 年 mm 月，上海交通大学，本科，xx 专业

研究兴趣

LaTeX 排版

联系方式

- 地址：上海市闵行区东川路 800 号，200240
- E-mail: xxx@sjtu.edu.cn

A SAMPLE DOCUMENT FOR LaTeX-BASED SJTU THESIS TEMPLATE

An imperial edict issued in 1896 by Emperor Guangxu, established Nanyang Public School in Shanghai. The normal school, school of foreign studies, middle school and a high school were established. Sheng Xuanhuai, the person responsible for proposing the idea to the emperor, became the first president and is regarded as the founder of the university.

During the 1930s, the university gained a reputation of nurturing top engineers. After the foundation of People's Republic, some faculties were transferred to other universities. A significant amount of its faculty were sent in 1956, by the national government, to Xi'an to help build up Xi'an Jiao Tong University in western China. Afterwards, the school was officially renamed Shanghai Jiao Tong University.

Since the reform and opening up policy in China, SJTU has taken the lead in management reform of institutions for higher education, regaining its vigor and vitality with an unprecedented momentum of growth. SJTU includes five beautiful campuses, Xuhui, Minhang, Luwan Qibao, and Fahua, taking up an area of about 3,225,833 m2. A number of disciplines have been advancing towards the top echelon internationally, and a batch of burgeoning branches of learning have taken an important position domestically.

Today SJTU has 31 schools (departments), 63 undergraduate programs, 250 masters-degree programs, 203 Ph.D. programs, 28 post-doctorate programs, and 11 state key laboratories and national engineering research centers.

SJTU boasts a large number of famous scientists and professors, including 35 academics of the Academy of Sciences and Academy of Engineering, 95 accredited professors and chair professors of the "Cheung Kong Scholars Program" and more than 2,000 professors and associate professors.

Its total enrollment of students amounts to 35,929, of which 1,564 are international students. There are 16,802 undergraduates, and 17,563 masters and Ph.D. candidates. After more than a century of operation, Jiao Tong University has inherited the old tradition of "high starting points, solid foundation, strict requirements and extensive practice." Students from SJTU have won top prizes in various competitions, including ACM International Collegiate Programming Contest, International Mathematical Contest in Modeling and Electronics Design Contests. Famous alumni include Jiang Zemin, Lu Dingyi, Ding Guangen, Wang Daohan, Qian Xuesen, Wu Wenjun, Zou Taofen, Mao Yisheng, Cai Er, Huang Yanpei, Shao Lizi, Wang An and many more. More than 200 of the academics of the Chinese Academy of Sciences and Chinese Academy of Engineering are alumni of Jiao Tong University.