华东师范大学软件学院课程作业

课程名称：软件质量分析	年级：2023 级本科	姓名：张梓卫
作业主题：开源软件兼容性可信分析	学号：10235101526	作业日期：2024/12/17
指导老师：陈仪香	组号：	

目录

本次作业按照 *PPT* 内的作业要求完成（共六题）

一 开源软件的定义及特点

开源软件（Open-Source Software）是通过特定类型的许可证发布的软件，这种许可证能让最终用户合法地使用其源代码。此类许可证有许多种（GPL、MIT、BSD…），但通常开源软件必须符合以下条件：

- 以源代码形式提供，无需额外费用：这意味着用户可以查看组成该软件的代码并对其进行所需的任何更改。
- 源代码可重新用于其他新软件：这意味着任何人都可以获取源代码并利用它来分发自己的程序。

其具有以下的**特点**：

- 开源软件数量众多、依赖关系复杂
- 开源软件持续更新、版本众多

二 开源软件 API 的定义及兼容性

API（Application Programming Interface, 应用程序编程接口）是一组规则或协议，可支持软件应用程序相互通信，以交换数据、特性和功能。API 允许开发人员集成来自其他应用程序的数据、服务和功能，而不是从头开始开发它们，从而简化和加速软件开发。

若对于相同的输入能产生相同的行为和输出，则认为新旧版本 API 的语义（行为）一致，新版本能够兼容旧版本。

兼容性的定义主要来源于几个方面：

API 的语法（签名）兼容性

API 兼容性首先关注 **语法兼容性**，即 API 的 **签名**是否发生变动。例如：

- 类或方法被删除；方法参数数量或类型发生变化；方法返回值类型变动。

这些变动通过 **静态分析**（如 JAPICC、Revapi、ABICC 等工具）检测，能够直接导致主软件的编译或链接错误。

API 的语义（行为）兼容性

API 兼容性还包括 **语义兼容性**，即相同的 API 输入是否产生 **相同的行为和输出**。
这种变动通过 **动态测试**（如回归测试）检测，例如：

- 方法的行为或结果改变。

即使 API 语法未改变，语义变动也可能导致运行时错误。

三 版本间兼容性和软件间兼容性

1 定义

1. **版本间兼容性**
   定义：同一开源软件不同版本之间，是否**向下/向上兼容**。

2. **软件间兼容性**
   定义：主软件或开源软件与**其他开源软件共存**，并能够正常调用其依赖的功能。

2 差异性

- **版本间兼容性：**
  - 主要关注同一开源软件不同版本之间的兼容性。
  - 将同一开源软件的不同版本之间的兼容性问题称为开源软件版本间兼容性问题，具体而言，我们需要检测同一开源软件不同版本之间的 API 变动是否会影响兼容性，以及影响的严重程度如何。
  - 从开源软件的开发者角度，不知道主软件的情况（哪些 API 被调用，调用情况如何），关注版本间的所有 API 变动（差异）。
- **软件间兼容性：**
  - 主要关注不同开源软件之间，尤其是主软件调用依赖软件 API 的兼容性。
  - 开源软件当前版本所提供的 API 能够满足主软件当前版本的所有 API 调用需求。
  - 从开源软件的使用者角度，知道主软件的情况（哪些 API 被调用，调用情况如何），关注所使用的开源软件版本间 API 变动（差异）对 API 调用造成的影响。

四 兼容性问题严重程度

1 各种工具之下的严重等级

不同工具:
JAPICC 根据其兼容性规则，检查多种类型的 API 变动，并给出兼容性问题的严重程度等级，具体包括：

- **High**：高严重性变动。
- **Medium**：中严重性变动。

- **Low**：低严重性变动。
- **Safe**：安全变动，无影响。

Revapi 根据其兼容性规则对 API 变动进行识别和分类，严重程度分级如下：

- **Breaking**：破坏性变动。
- **Potentially Breaking**：潜在破坏性变动。
- **Equivalent / Non Breaking**：等价或非破坏性变动。

ABICC 使用内置的兼容性规则对两个版本间 API 变动进行分类和分析，严重程度等级包括：

- **High**：高严重性变动。
- **Medium**：中严重性变动。
- **Low**：低严重性变动。
- **Safe**：安全变动，无影响。

JCBJARE 工具根据兼容性度量指标对兼容性问题进行严重程度划分，具体如下：

- **High**：高风险变动。
- **Medium**：中风险变动。
- **Low**：低风险变动。
- **No**：无风险。

CABICC 工具直接参照 ABICC 的兼容性规则，将兼容性度量指标的严重程度划分为：

- **High**：高风险变动。
- **Medium**：中风险变动。
- **Low**：低风险变动。
- **No**：无风险。

■

2 整合后得到的表格

工具	最高等级	中级	较低等级	最低等级
JAPICC	**High**	**Medium**	**Low**	**Safe**
Revapi	**Breaking**	**Potentially Breaking**	**-**	**Equivalent / Non Breaking**
ABICC	**High**	**Medium**	**Low**	**Safe**
JCBJARE	**High**	**Medium**	**Low**	**No**
CABICC	**High**	**Medium**	**Low**	**No**

表 1: 不同工具的兼容性严重程度等级划分

五 版本间兼容性可信度量公式及等级划分

1 可信度量公式

可信值 K 可以根据**玻尔兹曼加权熵** H，通过以下公式得到 K：

$$K = e^{-H} \tag{1}$$

H：软件版本间不兼容严重程度的熵，取值范围为 $[0, +\infty)$，该值越大，说明软件两版本间不兼容程度越大，两者越不兼容。
K：软件版本间兼容性可信度量值，取值范围为 $(0, 1]$。该值越大，说明软件两版本间兼容性可信性越高，两版本间越兼容。

2 映射可信值范围

软件版本间兼容性可信度量模型：此处已经将**最终可信值** T 范围映射到 $[1, 10]$

$$T = \begin{cases} 10 \cdot e^{-H} & (0.1 \le e^{-H} \le 1) \\ 10 & (0 \le e^{-H} < 0.1) \end{cases} \tag{2}$$

其中，

$$H = \sum_i \alpha_i \cdot \lg(n_i + 1), \quad \alpha_i = \frac{\lambda_i}{\lambda_h + \lambda_m + \lambda_l + \lambda_o}, \quad i \in \{h, m, l, o\} \tag{3}$$

参数定义：

- $\alpha_h, \alpha_m, \alpha_l, \alpha_o$ 分别表示 High、Medium、Low、No 的风险因子归一化权重；
- n_h, n_m, n_l, n_o 分别表示检测出的严重程度为 High、Medium、Low 和 No 的兼容性问题数量；
- $\lambda_h, \lambda_m, \lambda_l, \lambda_o$ 分别表示 High、Medium、Low 和 No 的风险因子。

风险因子根据**近似黄金分割比**设置，具体值为：

$$\lambda_h = 0.9, \quad \lambda_m = 0.7, \quad \lambda_l = 0.4, \quad \lambda_o = 0.1 \tag{4}$$

3 等级划分表

等级	T	d_h	d_m	d_l	**或满足**
V 级	$9 \le T$	$d_h = 0$	$d_m = 0$	$d_l \le 0.4$	无
IV 级	$7 \le T < 9$	$d_h = 0$	$d_m \le 0.4$	$d_l \le 0.7$	或 $9 \le T$ 且不能评为 V 级别
III 级	$4 \le T < 7$	$d_h \le 0.4$	$d_m \le 0.7$	-	或 $7 \le T$ 且不能评为 IV 级别及以上
II 级	$2 \le T < 4$	$d_h \le 0.7$	-	-	或 $4 \le T$ 且不能评为 III 级别及以上
I 级	$1 \le T < 2$	-	-	-	或 $2 \le T$ 且不能评为 II 级别及以上

其中，T 表示最终可信值，d_h, d_m, d_l 分别是 **High、Medium、Low 的变更密度**，计算公式如下所示：

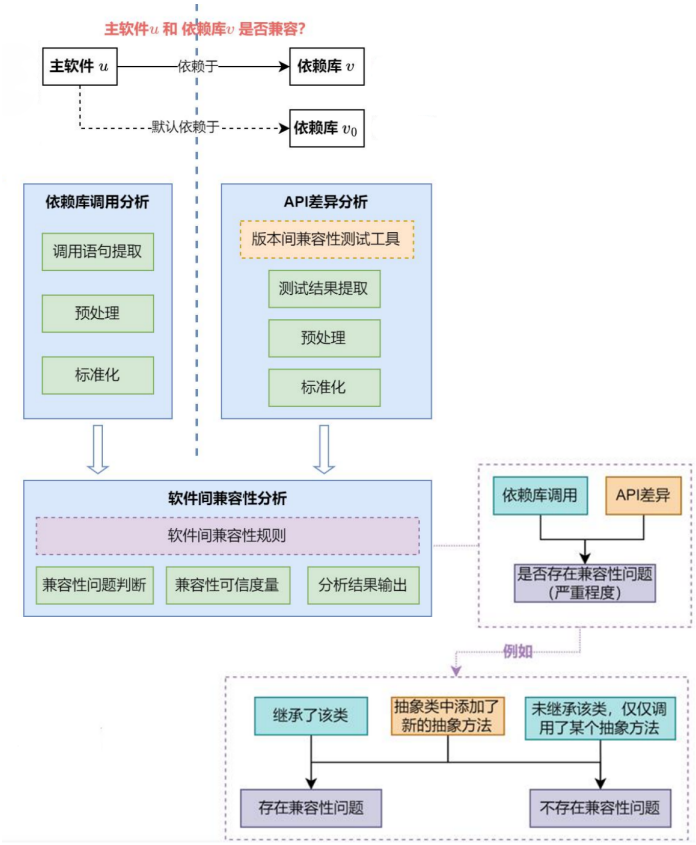
$$d_i = \frac{n_i}{n_h + n_m + n_l + n_o}, \quad i \in \{h, m, l, o\} \tag{5}$$

n_h, n_m, n_l, n_o 分别表示检测出的 **严重程度**为 High、Medium、Low 和 No 的 **兼容性问题数量**。

六 软件间兼容性测试框架

1 兼容性测试框架

我们基于预先设定的软件间兼容性规则，来判断是否会存在兼容性问题，以及兼容性问题的严重程度。
兼容性测试框架如图所示：



1. 依赖库调用分析

- **任务**：提取主软件源代码中对依赖库的 **API 调用语句**，以便分析主软件如何调用依赖库。
- **分析内容**：
  - API 类型（类、接口、方法等）
  - 调用形式（类继承、方法调用、方法重载等）
- **流程**：
  1. 调用语句提取
  2. 数据预处理
  3. 数据标准化

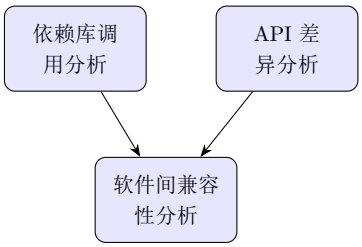
2. API 差异分析

- **任务**：比较当前版本和默认版本依赖库的 **API 差异**，识别 API 变更点。
- **分析内容**：
  - API 的**类移除、方法参数数量变化、返回值类型变化**等。
  - 借助开源的版本兼容性测试工具进行分析。
- **流程**：
  1. 提取 API 差异结果
  2. 数据预处理与标准化

3. 软件间兼容性分析

- **任务**：基于提取的 API 调用信息和 API 差异，评估 API 差异对主软件的**影响**。
- **分析内容**：
  - 通过**软件间兼容性规则**判断是否存在兼容性问题。
  - 评估兼容性问题的**严重程度**。
- **流程**：
  1. 兼容性问题判断
  2. 兼容性可行度量
  3. 分析结果输出

流程图：

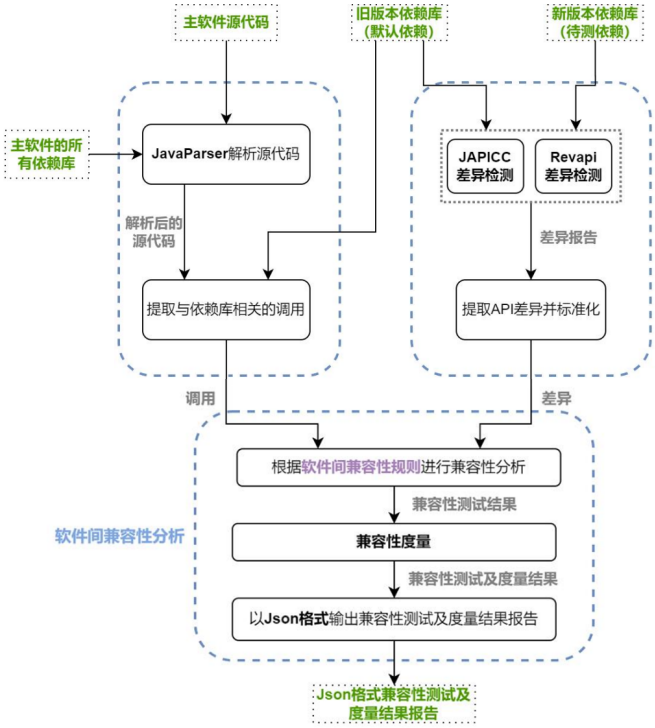


流程分析：

主软件通过调用依赖库的 API，检测当前版本与默认版本之间的差异，评估这些差异是否引发兼容性问题。

当直接依赖软件的某个抽象类中添加了新的抽象方法时，只有主软件继承了该类才会出现兼容性问题。

2 面向 Java 的工具平台框架



1. 主软件解析与依赖提取

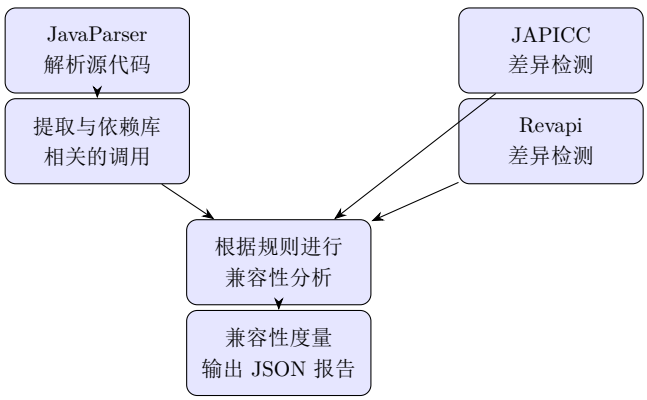
- **任务**：解析主软件源代码，提取所有直接依赖的调用关系。
- **工具**：使用 **JavaParser** 工具对主软件源代码进行语法解析。
- **流程**：
  1. 主软件的所有依赖库收集。
  2. 使用 JavaParser 解析源代码。
  3. 提取与依赖库相关的 API 调用语句。

2. API 差异检测与标准化

- **任务**：检测旧版本与新版本依赖库的 API 差异，并对差异进行标准化处理。
- **工具**：
  - **JAPICC**：用于检测接口层面的 API 差异。
  - **Revapi**：用于检测 API 变更的详细差异。
- **流程**：
  1. 比较旧版本（默认依赖）和新版本（待测依赖）的 API。
  2. 生成差异报告。
  3. 对差异报告进行标准化处理。

3. 兼容性分析与结果输出

- **任务**：基于差异报告和兼容性规则，分析兼容性问题，并输出结果。
- **实现**：
  - 根据 **软件间兼容性规则**进行兼容性分析。
  - 计算 **兼容性度量**，评估兼容性问题的严重程度。
- **输出**：
  - 生成兼容性测试和度量结果。
  - 以 **JSON 格式**输出最终结果报告。



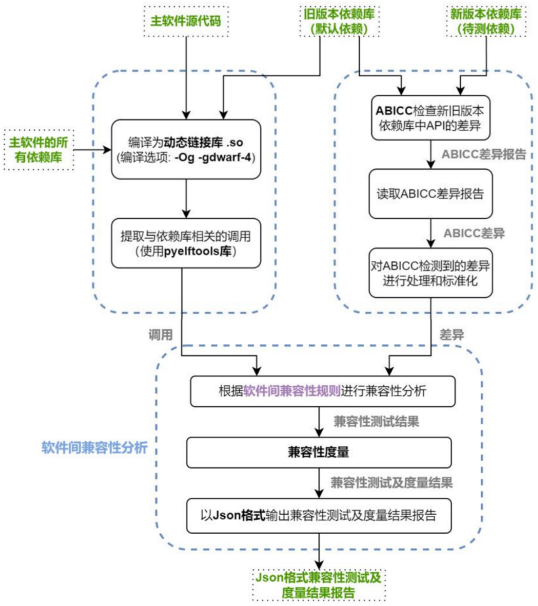
流程分析：

- JavaParser 解析主软件源码，提取对依赖库的调用。
- 通过工具检测新旧版本依赖库之间的 API 差异。
- 基于兼容性规则分析差异的影响，度量兼容性问题，并输出 JSON 格式的结果报告。

注意事项：当一个 API 调用受到多个 API 差异的影响时，受到影响的严重程度取高值。

High > Medium > Low > No

3 面向 Cpp 的工具平台框架



1. 依赖库调用分析

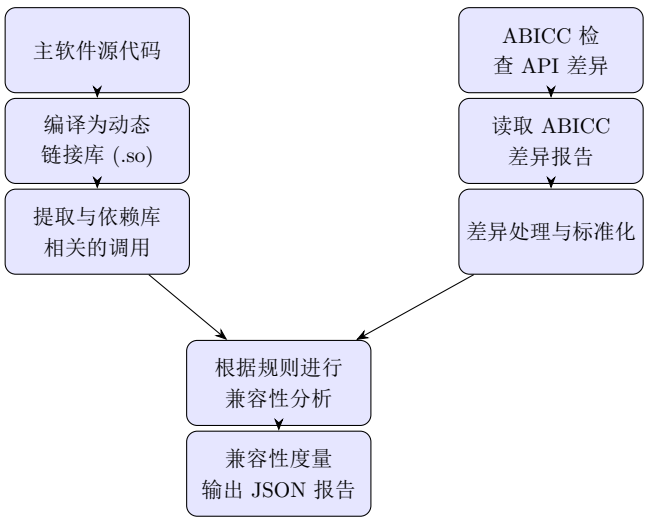
- **任务**：编译主软件源代码，提取依赖库相关的 API 调用信息。
- **工具**：
  - 编译工具链：生成动态链接库。
  - **pyelftools**:读取调试信息,提取调用关系。
- **流程**：
  1. 编译源代码，保留调试信息。
  2. 使用 **pyelftools** 提取依赖库的调用信息。

2. API 差异分析

- **任务**：检测新旧版本依赖库的 API 差异，并标准化处理差异信息。
- **工具**：**ABICC**：检测 API 变更及差异。
- **流程**：
  1. 使用 **ABICC** 检查当前版本和默认版本的 API 差异。
  2. 读取并解析差异报告。
  3. 对差异报告进行标准化处理，确保数据一致性。

3. 兼容性分析与结果输出

- **任务**:基于 API 差异和规则分析兼容性问题，生成兼容性度量报告。
- **实现**：
  - 应用 **C/C++ 软件间兼容性规则**分析 API 调用的影响。
  - 确定兼容性影响的严重程度 (**High** > **Medium** > **Low** > **No**)。
- **输出**：
  - 计算并生成兼容性测试与度量结果。
  - 以 **JSON 格式**输出最终结果报告，便于进一步分析。



流程分析：

- 编译主软件源代码，生成动态链接库，并保留调试信息。
- 使用 **ABICC** 检测新旧版本依赖库的 API 差异，标准化差异报告。
- 根据兼容性规则评估 API 差异影响，计算兼容性度量，输出 **JSON 格式**报告。

注意事项：当一个 API 调用受到多个差异的影响时，取最高严重程度：

High > Medium > Low > No