(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 112965695 A
(43)申请公布日 2021.06.15

(21)申请号 202110270341.0

(22)申请日 2021.03.12

(71)申请人 中国平安财产保险股份有限公司
地址 518048 广东省深圳市福田区益田路5033号平安金融中心12、13、38、39、40层

(72)发明人 朱涛

(74)专利代理机构 北京市京大律师事务所 11321
代理人 姚维

(51)Int.Cl.
G06F 8/20(2018.01)
G06F 8/41(2018.01)
G06F 8/61(2018.01)
G06F 8/71(2018.01)
G06F 11/36(2006.01)

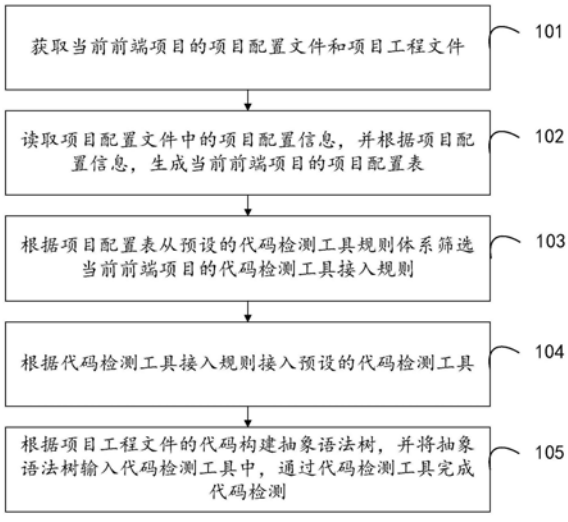
权利要求书2页 说明书13页 附图7页

(54)发明名称

前端代码接入检测方法、装置、设备及存储介质

(57)摘要

本发明涉及计算机软件开发领域，公开了一种前端代码接入检测方法、装置、设备及存储介质，用于进行前端代码接入检测，该方法包括：获取当前前端项目的项目配置文件和项目工程文件；读取所述项目配置文件中的项目配置信息，并根据所述项目配置信息，生成当前前端项目的项目配置表；根据所述项目配置表从预设的代码检测工具规则体系筛选所述当前前端项目的代码检测工具接入规则；根据所述代码检测工具接入规则接入预设的代码检测工具；根据所述项目工程文件的代码构建抽象语法树，并将所述抽象语法树输入所述代码检测工具中，通过所述代码检测工具完成代码检测。此外，本发明还涉及区块链技术，所述项目配置文件和项目工程文件可存储于区块链中。



CN 112965695 A

1.一种前端代码接入检测方法,其特征在于,所述前端代码接入检测方法包括:

获取当前前端项目的项目配置文件和项目工程文件;

读取所述项目配置文件中的项目配置信息,并根据所述项目配置信息,生成当前前端项目的项目配置表;

根据所述项目配置表从预设的代码检测工具规则体系筛选所述当前前端项目的代码检测工具接入规则;

根据所述代码检测工具接入规则接入预设的代码检测工具;

根据所述项目工程文件的代码构建抽象语法树,并将所述抽象语法树输入所述代码检测工具中,通过所述代码检测工具完成代码检测。

2.根据权利要求1所述的前端代码接入检测方法,其特征在于,所述根据所述项目工程文件的代码构建抽象语法树,并将所述抽象语法树输入所述代码检测工具中,通过所述代码检测工具完成代码检测包括:

将所述项目工程文件输入预设的语法分析器,通过所述语法分析器对所述项目工程文件中的代码进行词法分析,将所述项目工程文件转换为由词法单元组成的数组;

根据所述数组生成所述项目工程文件的抽象语法树;

将所述抽象语法树输入所述代码检测工具中,通过所述代码检测工具完成代码检测。

3.根据权利要求2所述的前端代码接入检测方法,其特征在于,所述将所述项目工程文件输入预设的语法分析器,通过所述语法分析器对所述项目工程文件中的代码进行词法分析,将所述项目工程文件转换为由词法单元组成的数组包括:

将所述项目工程文件输入所述语法分析器,通过所述语法分析器对所述项目工程文件的代码进行逐行扫描并分解成词法单元,其中,所述词法单元为编程语言中字符串形式的基本单词符号;

将所述词法单元序列组合成数组。

4.根据权利要求3所述的前端代码接入检测方法,其特征在于,所述根据所述数组生成所述项目工程文件的抽象语法树包括:

根据所述数组对所述代码进行语法分析,判断所述代码的语法是否正确;

若是,则根据预设的源代码语言规范,生成与所述代码对应的语法分析树;

调用所述语法分析树中各节点对应的节点对象创建方法,创建节点对象,根据所述节点对象生成抽象语法树。

5.根据权利要求1-4中任一项所述的前端代码接入检测方法,其特征在于,所述将所述抽象语法树输入所述代码检测工具中,通过所述代码检测工具完成代码检测包括:

启动多线程,每个线程通过命令行调用所述代码检测工具,通过所述代码检测工具对所述抽象语法树进行检测;

等待所述多线程中的全部线程执行完毕,生成所述代码检测工具的检测结果;

分析所述检测结果,并根据预设的结果过滤条件,将所述检测结果中误报和无用的结果过滤,得到最终检测结果;

将所述最终检测结果以工程为单位保存。

6.根据权利要求5所述的前端代码接入检测方法,其特征在于,在所述根据所述代码检测工具接入规则接入预设的代码检测工具之前,还包括:

获取所述代码检测工具的版本类型；

根据所述版本类型获取所述代码检测工具的依赖列表；

根据所述依赖列表获取所述代码检测工具的依赖包并安装所述依赖包。

7.根据权利要求6所述的前端代码接入检测方法，其特征在于，所述根据所述依赖列表获取所述代码检测工具的依赖包并安装所述依赖包括：

根据所述依赖列表获取所述代码检测工具的依赖包；

对所述依赖包进行合法性检验，得到检验结果；

若所述检验结果为合法，则安装所述依赖包。

8.一种前端代码接入检测装置，其特征在于，所述前端代码接入检测装置包括：

文件获取模块，用于获取当前前端项目的项目配置文件和项目工程文件；

信息读取模块，用于读取所述项目配置文件中的项目配置信息，并根据所述项目配置信息，生成当前前端项目的项目配置表；

规则筛选模块，用于根据所述项目配置表从预设的代码检测工具规则体系筛选所述当前前端项目的代码检测工具接入规则；

接入模块，用于根据所述代码检测工具接入规则接入预设的代码检测工具；

检测模块，用于根据所述项目工程文件的代码构建抽象语法树，并将所述抽象语法树输入所述代码检测工具中，通过所述代码检测工具完成代码检测。

9.一种前端代码接入检测设备，其特征在于，所述前端代码接入检测设备包括：存储器和至少一个处理器，所述存储器中存储有指令，所述存储器和所述至少一个处理器通过线路互连；

所述至少一个处理器调用所述存储器中的所述指令，以使得所述前端代码接入检测设备执行如权利要求1-7中任一项所述的前端代码接入检测方法的步骤。

10.一种计算机可读存储介质，所述计算机可读存储介质上存储有计算机程序，其特征在于，所述计算机程序被处理器执行时实现如权利要求1-7中任一项所述的前端代码接入检测方法的步骤。

前端代码接入检测方法、装置、设备及存储介质

技术领域

[0001] 本发明涉及计算机软件开发领域，尤其涉及一种前端代码接入检测方法、装置、设备及存储介质。

背景技术

[0002] 对于大多数的编程语言来说，为了保证代码的质量一般来说变异程序会内置一些代码检测工具。对于前端开发工程来说，基本是基于JavaScript,HTML,CSS来进行开发。但由于JavaScript,HTML,CSS没有编译程序，为了寻找JavaScript代码错误通常需要在执行过程中不断调试。而在现代的前端体系中，基于webpack的工程化搭建，同时配套着webpack的eslint,stylelint等插件，为前端工程提供了代码检测的能力。在我们启动工程时，webpack会将我们的工程文件读入，再以二进制文件流或者AST语法树的方式传递给我们的lint插件，再在lint插件中检测我们的代码，帮助开发人员提升工程代码质量。通常我们的项目如果需要接入lint工具，首先我们需要安装相应的工具依赖包，然后引入相应的规则，再修改webpack的配置，才能够完成lint工具的接入工作。

[0003] 但由于各个lint工具的规则相对自由，需要开发人员花费大量时间才能完成规则配置。同时安装lint工具时整体可能会依赖许多外部第三方依赖，webpack的lint配置较为繁琐，开发人员完成lint工具接入的整体难度较高。这导致了现在产险的大部分前端项目未接入lint规则，或者接入的规则不统一，各个项目组开发规范不一致等问题。

发明内容

[0004] 本发明的主要目的在于解决现有的lint工具接入规则不统一导致lint工具接入的整体难度较高的技术问题。

[0005] 本发明第一方面提供了一种前端代码接入检测方法，包括：

[0006] 获取当前前端项目的项目配置文件和项目工程文件；

[0007] 读取所述项目配置文件中的项目配置信息，并根据所述项目配置信息，生成当前前端项目的项目配置表；

[0008] 根据所述项目配置表从预设的代码检测工具规则体系筛选所述当前前端项目的代码检测工具接入规则；

[0009] 根据所述代码检测工具接入规则接入预设的代码检测工具；

[0010] 根据所述项目工程文件的代码构建抽象语法树，并将所述抽象语法树输入所述代码检测工具中，通过所述代码检测工具完成代码检测。

[0011] 可选的，在本发明第一方面的第一种实现方式中，所述根据所述项目工程文件的代码构建抽象语法树，并将所述抽象语法树输入所述代码检测工具中，通过所述代码检测工具完成代码检测包括：

[0012] 将所述项目工程文件输入预设的语法分析器，通过所述语法分析器对所述项目工程文件中的代码进行词法分析，将所述项目工程文件转换为由词法单元组成的数组；

[0013] 根据所述数组生成所述项目工程文件的抽象语法树；

[0014] 将所述抽象语法树输入所述代码检测工具中，通过所述代码检测工具完成代码检测。

[0015] 可选的，在本发明第一方面的第二种实现方式中，所述将所述项目工程文件输入预设的语法分析器，通过所述语法分析器对所述项目工程文件中的代码进行词法分析，将所述项目工程文件转换为由词法单元组成的数组包括：

[0016] 将所述项目工程文件输入所述语法分析器，通过所述语法分析器对所述项目工程文件的代码进行逐行扫描并分解成词法单元，其中，所述词法单元为编程语言中字符串形式的基本单词符号；

[0017] 将所述词法单元序列组合成数组。

[0018] 可选的，在本发明第一方面的第三种实现方式中，所述根据所述数组生成所述项目工程文件的抽象语法树包括：

[0019] 根据所述数组对所述代码进行语法分析，判断所述代码的语法是否正确；

[0020] 若是，则根据预设的源代码语言规范，生成与所述代码对应的语法分析树；

[0021] 调用所述语法分析树中各节点对应的节点对象创建方法，创建节点对象，根据所述节点对象生成抽象语法树。

[0022] 可选的，在本发明第一方面的第四种实现方式中，其特征在于，在所述根据所述项目工程文件的代码构建抽象语法树，并将所述抽象语法树输入所述代码检测工具中，通过所述代码检测工具完成代码检测之后，还包括：

[0023] 启动多线程，每个线程通过命令行调用所述代码检测工具，通过所述代码检测工具对所述项目工程文件进行检测；

[0024] 等待所述多线程中的全部线程执行完毕，生成所述代码检测工具的检测结果；

[0025] 分析所述检测结果，并根据预设的结果过滤条件，将所述检测结果中误报和无用的结果过滤，得到最终检测结果；

[0026] 将所述最终检测结果以工程为单位保存。

[0027] 可选的，在本发明第一方面的第五种实现方式中，在所述根据所述代码检测工具接入规则接入预设的代码检测工具之前，还包括：

[0028] 获取所述代码检测工具的版本类型；

[0029] 根据所述版本类型获取所述代码检测工具的依赖列表；

[0030] 根据所述依赖列表获取所述代码检测工具的依赖包并安装所述依赖包。

[0031] 可选的，在本发明第一方面的第六种实现方式中，所述根据所述依赖列表获取所述代码检测工具的依赖包并安装所述依赖包包括：

[0032] 根据所述依赖列表获取所述代码检测工具的依赖包；

[0033] 对所述依赖包进行合法性检验，得到检验结果；

[0034] 若所述检验结果为合法，则安装所述依赖包。

[0035] 本发明第二方面提供了一种前端代码接入检测装置，包括：

[0036] 文件获取模块，用于获取当前前端项目的项目配置文件和项目工程文件；

[0037] 信息读取模块，用于读取所述项目配置文件中的项目配置信息，并根据所述项目配置信息，生成当前前端项目的项目配置表；

[0038] 规则筛选模块，用于根据所述项目配置表从预设的代码检测工具规则体系筛选所述当前前端项目的代码检测工具接入规则；

[0039] 接入模块，用于根据所述代码检测工具接入规则接入预设的代码检测工具；

[0040] 检测模块，用于根据所述项目工程文件的代码构建抽象语法树，并将所述抽象语法树输入所述代码检测工具中，通过所述代码检测工具完成代码检测。

[0041] 可选的，在本发明第二方面的第一种实现方式中，所述检测模块包括：

[0042] 词法分析单元，用于将所述项目工程文件输入预设的语法分析器，通过所述语法分析器对所述项目工程文件中的代码进行词法分析，将所述项目工程文件转换为由词法单元组成的数组；

[0043] 语法树生成单元，用于根据所述数组生成所述项目工程文件的抽象语法树；

[0044] 输入单元，用于将所述抽象语法树输入所述代码检测工具中，通过所述代码检测工具完成代码检测。

[0045] 可选的，在本发明第二方面的第二种实现方式中，所述词法分析单元具体用于：

[0046] 将所述项目工程文件输入所述语法分析器，通过所述语法分析器对所述项目工程文件的代码进行逐行扫描并分解成词法单元，其中，所述词法单元为编程语言中字符串形式的基本单词符号；

[0047] 将所述词法单元序列组合成数组。

[0048] 可选的，在本发明第二方面的第三种实现方式中，所述语法树生成单元具体用于：

[0049] 根据所述数组对所述代码进行语法分析，判断所述代码的语法是否正确；

[0050] 若是，则根据预设的源代码语言规范，生成与所述代码对应的语法分析树；

[0051] 调用所述语法分析树中各节点对应的节点对象创建方法，创建节点对象，根据所述节点对象生成抽象语法树。

[0052] 可选的，在本发明第二方面的第四种实现方式中，所述输入单元具体用于：

[0053] 启动多线程，每个线程通过命令行调用所述代码检测工具，通过所述代码检测工具对所述抽象语法树进行检测；

[0054] 等待所述多线程中的全部线程执行完毕，生成所述代码检测工具的检测结果；

[0055] 分析所述检测结果，并根据预设的结果过滤条件，将所述检测结果中误报和无用的结果过滤，得到最终检测结果；

[0056] 将所述最终检测结果以工程为单位保存。

[0057] 可选的，在本发明第二方面的第五种实现方式中，所述前端代码接入检测装置还包括依赖配置模块，所述依赖配置模块包括：

[0058] 版本获取单元，用于获取所述代码检测工具的版本类型；

[0059] 列表获取单元，用于根据所述版本类型获取所述代码检测工具的依赖列表；

[0060] 依赖包安装单元，用于根据所述依赖列表获取所述代码检测工具的依赖包并安装所述依赖包。

[0061] 可选的，在本发明第二方面的第六种实现方式中，所述依赖包安装单元具体用于：

[0062] 根据所述依赖列表获取所述代码检测工具的依赖包；

[0063] 对所述依赖包进行合法性检验，得到检验结果；

[0064] 若所述检验结果为合法，则安装所述依赖包。

[0065] 本发明第三方面提供了一种前端代码接入检测设备，包括：存储器和至少一个处理器，所述存储器中存储有指令，所述存储器和所述至少一个处理器通过线路互连；所述至少一个处理器调用所述存储器中的所述指令，以使得所述前端代码接入检测设备执行上述的前端代码接入检测方法的步骤。

[0066] 本发明的第四方面提供了一种计算机可读存储介质，所述计算机可读存储介质中存储有指令，当其在计算机上运行时，使得计算机执行上述的前端代码接入检测方法的步骤。

[0067] 本发明的技术方案中，通过获取当前前端项目的项目配置文件和项目工程文件；读取所述项目配置文件中的项目配置信息，并根据所述项目配置信息，生成当前前端项目的项目配置表；根据所述项目配置表从预设的代码检测工具规则体系筛选所述当前前端项目的代码检测工具接入规则；根据所述代码检测工具接入规则接入预设的代码检测工具；根据所述项目工程文件的代码构建抽象语法树，并将所述抽象语法树输入所述代码检测工具中，通过所述代码检测工具完成代码检测。通过本方法读取前端项目的配置信息生成项目配置表，再根据项目配置表进行lint工具接入规则的获取，针对不同前端项目进行不同lint工具接入规则的获取，避免由于lint工具接入规则不统一，导致lint工具接入难度大的问题，便于对当前项目接入lint工具，进行项目文件代码的质量检测。

附图说明

[0068] 图1为本发明实施例中前端代码接入检测方法的第一个实施例示意图；

[0069] 图2为本发明实施例中前端代码接入检测方法的第二个实施例示意图；

[0070] 图3为本发明实施例中前端代码接入检测方法的第三个实施例示意图；

[0071] 图4为本发明实施例中前端代码接入检测方法的第四个实施例示意图；

[0072] 图5为本发明实施例中前端代码接入检测方法的第五个实施例示意图；

[0073] 图6为本发明实施例中前端代码接入检测装置的一个实施例示意图；

[0074] 图7为本发明实施例中前端代码接入检测装置的另一个实施例示意图；

[0075] 图8为本发明实施例中前端代码接入检测设备的一个实施例示意图。

具体实施方式

[0076] 本发明的技术方案中，通过获取当前前端项目的项目配置文件和项目工程文件；读取所述项目配置文件中的项目配置信息，并根据所述项目配置信息，生成当前前端项目的项目配置表；根据所述项目配置表从预设的代码检测工具规则体系筛选所述当前前端项目的代码检测工具接入规则；根据所述代码检测工具接入规则接入预设的代码检测工具；根据所述项目工程文件的代码构建抽象语法树，并将所述抽象语法树输入所述代码检测工具中，通过所述代码检测工具完成代码检测。通过本方法读取前端项目的配置信息生成项目配置表，再根据项目配置表进行lint工具接入规则的获取，针对不同前端项目进行不同lint工具接入规则的获取，避免由于lint工具接入规则不统一，导致lint工具接入难度大的问题，便于对当前项目接入lint工具，进行项目文件代码的质量检测。

[0077] 本发明的说明书和权利要求书及上述附图中的术语“第一”、“第二”、“第三”、“第四”等(如果存在)是用于区别类似的对象，而不必用于描述特定的顺序或先后次序。应该理

解这样使用的数据在适当情况下可以互换，以便这里描述的实施例能够以除了在这里图示或描述的内容以外的顺序实施。此外，术语“包括”或“具有”及其任何变形，意图在于覆盖不排他的包含，例如，包含了一系列步骤或单元的过程、方法、系统、产品或设备不必限于清楚地列出的那些步骤或单元，而是可包括没有清楚地列出的或对于这些过程、方法、产品或设备固有的其它步骤或单元。

[0078] 为便于理解，下面对本发明实施例的具体流程进行描述，请参阅图1，本发明实施例中前端代码接入检测方法的第一个实施例包括：

[0079] 101、获取当前前端项目的项目配置文件和项目工程文件；

[0080] 可以理解的是，本发明的执行主体可以为前端代码接入检测装置，还可以是终端或者服务器，具体此处不做限定。本发明实施例以服务器为执行主体为例进行说明。

[0081] 需要强调的是，为保证数据的私密和安全性，上述项目配置文件和项目工程文件可以存储于一区块链的节点中。

[0082] 在本实施例中，所述项目配置文件为package.json文件，主要包括前端的配置依赖以及项目工程描述和部分项目参数，其中，项目工程描述包括于描述该项目工程运行环境，具体包括该环境下使用的CPU、操作系统，用于选择项目工程编译环境的编译器、以及项目工程构建需要的所有模块，项目参数有项目地址、项目包名称、项目类型、项目配置信息，项目地址可以为当前前端项目的保存地址；项目包名称可以为当前前端项目的全部相关的包(package)名称；项目类型可以为当前前端项目的类型，具体为：当项目只包括一个模块文件时，则该项目的类型可以为单模块应用，当项目包括多个模块文件时，则该项目的类型为多模块应用，项目配置信息可以为与当前前端项目相关的配置信息。

[0083] 在本实施例中，项目工程文件为js(JavaScript)文件，js是一种具有函数优先的轻量级，解释型或即时编译型的高级编程语言，可以作为浏览器Web页面的脚本语言，JavaScript基于原型编程、多范式的动态脚本语言，并且支持面向对象、命令式和声明式(如函数式编程)风格，常用来为浏览器的网页添加各式各样的动态功能，为用户提供更流畅美观的浏览效果。

[0084] 102、读取所述项目配置文件中的项目配置信息，并根据所述项目配置信息，生成当前前端项目的项目配置表；

[0085] 在本实施例中，通过项目配置文件中的判断当前项目的框架以及语言，根据配置名称在lint规则总库中进行匹配。再把匹配到的规则生成项目配置表，生成项目配置表的目的在于本方案对lint规则进行了拓展分类。每一类规则可能就有好几百条。比如vue有针对vue的一套规则，react有react的一套规则。同时每一个配置对应的npm插件也不一样。所以这一套规则其实配置信息到实际规则以及实际需要安装的npm插件的映射关系集合，生成项目配置表方便拓展也方便后期维护。

[0086] 103、根据所述项目配置表从预设的代码检测工具规则体系筛选所述当前前端项目的代码检测工具接入规则；

[0087] 在实际应用中，由于JavaScript的动态语言类型，产生了风格多样的开发范式，同时也带来了一些问题，从运行时常见的undefined、null报错，到代码随意的加减分号、换行、空格，引起的视觉混乱，如果是团队开发，则这种情况会更加的严重，必须加以约束，进行代码检测。

[0088] 在本实施例中，所述代码检测工具为lint，lint为一种静态代码分析工具，为C语言工具之一，lint可以对程序进行更加广泛的错误分析，它不但可以检查出可移植性问题，而且可以检查出那些虽然可移植并且完全合乎语法但却很可能是错误的特性。

[0089] 104、根据所述代码检测工具接入规则接入预设的代码检测工具；

[0090] 在实际应用中，接入lint工具，首先需要安装相应的工具依赖包，并引入代码检测工具接入规则，再修改webpack的配置，才能够完成lint工具的接入工作。

[0091] 105、根据所述项目工程文件的代码构建抽象语法树，并将所述抽象语法树输入所述代码检测工具中，通过所述代码检测工具完成代码检测；

[0092] 在本实施例中，启动工程时，webpack会将工程文件读入，再以二进制文件流或者AST语法树的方式传递给我们的lint插件，再在lint插件中检测的代码，帮助开发人员提升工程代码质量。

[0093] 在本实施例中，主要通过语法分析器将项目工程文件转换为抽象语法树，语法分析器主要为Babel编译器，语法分析器(Parser)通常是作为编译器或解释器的组件出现的，它的作用是进行语法检查、并构建由输入的单词组成的数据结构(一般是语法分析树、抽象语法树等层次化的数据结构)。语法分析器通常使用一个独立的词法分析器从输入字符流中分离出一个个的“单词”，并将单词流作为其输入。实际开发中，语法分析器可以手工编写，也可以使用工具(半)自动生成。babel是一个JavaScript编译器。宏观来说，它分3个阶段运行代码：解析(parsing)—将代码字符串转换成AST抽象语法树，转译(transforming)—对抽象语法树进行变换操作，生成(generation)—根据变换后的抽象语法树生成新的代码字符串。

[0094] 在本实施例中，语法分析器将所述项目工程文件解析为抽象语法树的过程包括了词法分析和语法分析，其中词法分析主要是读取输入的项目工程文件的源代码的字符流后根据构词规则识别token(也称单词符号或符号)，所述语法分析主要是是在词法分析的基础上将单词序列组合成各类语法短语，如“程序”“语句”“表达式”等等，形成AST(Abstract Syntax Tree，抽象语法树)。

[0095] 在本实施例中，通过获取当前前端项目的项目配置文件和项目工程文件；读取所述项目配置文件中的项目配置信息，并根据所述项目配置信息，生成当前前端项目的项目配置表；根据所述项目配置表从预设的代码检测工具规则体系筛选所述当前前端项目的代码检测工具接入规则；根据所述代码检测工具接入规则接入预设的代码检测工具；根据所述项目工程文件的代码构建抽象语法树，并将所述抽象语法树输入所述代码检测工具中，通过所述代码检测工具完成代码检测。通过本方法读取前端项目的配置信息生成项目配置表，再根据项目配置表进行lint工具接入规则的获取，针对不同前端项目进行不同lint工具接入规则的获取，避免由于lint工具接入规则不统一，导致lint工具接入难度大的问题，便于对当前项目接入lint工具，进行项目文件代码的质量检测。此外，本发明还涉及区块链技术，所述项目配置文件和项目工程文件可存储于区块链中。

[0096] 请参阅图2，本发明实施例中前端代码接入检测方法的第二个实施例包括：

[0097] 201、获取当前前端项目的项目配置文件和项目工程文件；

[0098] 202、读取项目配置文件中的项目配置信息，并根据项目配置信息，生成当前前端项目的项目配置表；

[0099] 203、根据项目配置表从预设的代码检测工具规则体系筛选当前前端项目的代码检测工具接入规则；

[0100] 本实施例中的步骤201-203与第一实施例中的步骤101-103相似，此处不再赘述。

[0101] 204、获取代码检测工具的版本类型；

[0102] 205、根据版本类型获取代码检测工具的依赖列表；

[0103] 206、根据依赖列表获取代码检测工具的依赖包；

[0104] 在本实施例中，所述代码检测工具有多个版本类别，例如eslint，sytlelint，commitlint等多个流lint工具的各个版本，初始化lint规则配置的时候会并发的去安装需要用到的依赖包，并根据维护工具版本列表检测版本，保证项目的lint工具以及相关依赖的可以用性。

[0105] 在实际应用中，由于操作系统都是模块化的设计，也就是说功能互相依靠，有些功能需要一些其他功能来支撑，这样可以提高代码的可重用性，也就是某部分功能的实现，需要对应的其他功能支撑，也就是依赖包，大部分依赖包都是一些库文件，有动态库也有静态库，一个程序的依赖包如果没有安装，只安装了这个程序本身是不能使用，例如安装了某个软件，这个软件又依赖于某个开发包，这个开发包包含这个软件所要运行的环境文件，这就是依赖关系。

[0106] 207、对依赖包进行合法性检验，得到检验结果；

[0107] 208、若检验结果为合法，则安装依赖包；

[0108] 在本实施例中，所述对依赖包的合法性检验包括基础应用框架，依赖包的稳定性测试，依赖包的安全测试，依赖包的依赖管理检查，依赖包的上架规范检查和依赖包的依赖大小检查等。

[0109] 209、根据代码检测工具接入规则接入预设的代码检测工具；

[0110] 210、根据项目工程文件的代码构建抽象语法树，并将抽象语法树输入代码检测工具中，通过代码检测工具完成代码检测。

[0111] 本实施例中的步骤209-210与第一实施例中的步骤104-105相似，此处不再赘述。

[0112] 本实施例在上一实施例的基础上，增加了添加依赖包的过程，通过获取代码检测工具的版本类型；根据版本类型获取代码检测工具的依赖列表；根据依赖列表获取代码检测工具的依赖包；对依赖包进行合法性检验，得到检验结果；若检验结果为合法，则安装依赖包。通过本方法保证项目的代码检测工具以及相关依赖的可以用性。通过对依赖包的合法性检测，避免获取的依赖包故障的问题发生。

[0113] 请参阅图3，本发明实施例中前端代码接入检测方法的第三个实施例包括：

[0114] 301、获取当前前端项目的项目配置文件和项目工程文件；

[0115] 302、读取项目配置文件中的项目配置信息，并根据项目配置信息，生成当前前端项目的项目配置表；

[0116] 303、根据项目配置表从预设的代码检测工具规则体系筛选当前前端项目的代码检测工具接入规则；

[0117] 304、根据代码检测工具接入规则接入预设的代码检测工具；

[0118] 本实施例中的步骤301-304与第一实施例中的步骤101-104相似，此处不再赘述。

[0119] 305、将项目工程文件输入语法分析器，通过语法分析器对项目工程文件的代码进

行逐行扫描并分解成词法单元；

[0120] 306、将词法单元序列组合成数组；

[0121] 在本实施例中，在本实施例中，在词法分析过程中扫描源代码的每条语句，边扫描边进行分词，得到源代码的每条语句的词法单元。词法单元是语句的基本组成单元，词法单元可以是关键字、变量名、操作符、常量等，例如，获取到源代码为“const a＝1,const b＝a+1;”，扫描第一条语句“const a＝1,”，首先会扫描到“const”，生成一个词法单元表示“const”为关键字；接着扫描到“a”，生成一个词法单元表示“a”为变量名；然后扫描到“＝”，生成一个词法单元表示“＝”为操作符；再扫描到常量“1”，生成一个词法单元表示“1”为常量；最后扫描到标点符号“,”，生成一个词法单元表示“,”为标点符号。词法分析过程中将语句“const a＝1,”的各个词法单元组成数组{const a＝1,}。类似地，词法分系统将语句“const b＝a+1;”转换为由词法单元组成的数组{const b＝a+1;}。

[0122] 307、根据数组生成项目工程文件的抽象语法树；

[0123] 308、将抽象语法树输入代码检测工具中，通过代码检测工具完成代码检测。

[0124] 在本实施例中，主要通过词法分析和语法分析两个过程构建抽象语法树，其中，其中词法分析主要是读取输入的项目工程文件的源代码的字符流后根据构词规则识别token(也称单词符号或符号)，所述语法分析主要是是在词法分析的基础上将单词序列组合成各类语法短语，如“程序”“语句”“表达式”等等，形成AST(Abstract Syntax Tree，抽象语法树)。

[0125] 本实施例在前实施例的基础上，详细描述了根据所述项目工程文件的代码构建抽象语法树，并将所述抽象语法树输入所述代码检测工具中，通过所述代码检测工具完成代码检测的过程。通过将所述项目工程文件输入预设的语法分析器，通过所述语法分析器对所述项目工程文件中的代码进行词法分析，将所述项目工程文件转换为由词法单元组成的数组；根据所述数组生成所述项目工程文件的抽象语法树；将所述抽象语法树输入所述代码检测工具中，通过所述代码检测工具完成代码检测。通过本方法读取前端项目的配置信息生成项目配置表，再根据项目配置表进行lint工具接入规则的获取，针对不同前端项目进行不同lint工具接入规则的获取，避免由于lint工具接入规则不统一，导致lint工具接入难度大的问题，便于对当前项目接入lint工具，进行项目文件代码的质量检测。

[0126] 请参阅图4，本发明实施例中前端代码接入检测方法的第四个实施例包括：

[0127] 401、获取当前前端项目的项目配置文件和项目工程文件；

[0128] 402、读取项目配置文件中的项目配置信息，并根据项目配置信息，生成当前前端项目的项目配置表；

[0129] 403、根据项目配置表从预设的代码检测工具规则体系筛选当前前端项目的代码检测工具接入规则；

[0130] 404、根据代码检测工具接入规则接入预设的代码检测工具；

[0131] 本实施例中的步骤401-404与第一实施例中的步骤101-104相似，此处不再赘述。

[0132] 405、将项目工程文件输入预设的语法分析器，通过语法分析器对项目工程文件中的代码进行词法分析，将项目工程文件转换为由词法单元组成的数组；

[0133] 406、根据数组对代码进行语法分析，判断代码的语法是否正确；

[0134] 407、若是，则根据预设的源代码语言规范，生成与代码对应的语法分析树；

[0135] 408、调用语法分析树中各节点对应的节点对象创建方法，创建节点对象，根据节点对象生成抽象语法树；

[0136] 在本实施例中，若源代码的语法错误，可以将错误信息反馈给用户，接收用户返回的修改后的源代码，对修改后的源代码进行词法分析，将修改后的源代码转换为由词法单元组成的数组，根据修改后的源代码的由词法单元组成的数组生成所述抽象语法树。例如，语法分析器对源代码“const a＝1，const b＝a+1；”进行分析，确定所述源代码的第一条语句“const a＝1，”的语法错误，该语句的结尾应该是分号而不是逗号。语法分析器将错误信息反馈给用户，接收用户返回的修改后的源代码。例如，接收用户返回的修改后的源代码“const a＝1；const b＝a+1；”。词法分析器对修改后的源代码进行词法分析，得到修改后的源代码的由词法单元组成的数组，例如得到{const a＝1；}，{const b＝a+1；}，语法分析器根据修改后的源代码的由词法单元组成的数组生成所述抽象语法树。

[0137] 在本实施例中，抽象语法树是在语法分析树的基础上建立的。其工作过程是依据自定义的抽象语法树的文法，解析分析语法树中的字符串并提取Token信息，同时根据各级不同的节点，如根节点、类节点类、方法节点类，分别调用各自对应的方法，创建节点对象，继而生成程序抽象语法树。

[0138] 409、将抽象语法树输入代码检测工具中，通过代码检测工具完成代码检测。

[0139] 本实施例在前实施例的基础上，详细描述了根据数组生成项目工程文件的抽象语法树的过程，根据数组对代码进行语法分析，判断代码的语法是否正确；若是，则根据预设的源代码语言规范，生成与代码对应的语法分析树；调用语法分析树中各节点对应的节点对象创建方法，创建节点对象，根据节点对象生成抽象语法树。通过本方法读取前端项目的配置信息生成项目配置表，再根据项目配置表进行lint工具接入规则的获取，针对不同前端项目进行不同lint工具接入规则的获取，避免由于lint工具接入规则不统一，导致lint工具接入难度大的问题，便于对当前项目接入lint工具，进行项目文件代码的质量检测。

[0140] 请参阅图5，本发明实施例中前端代码接入检测方法的第五个实施例包括：

[0141] 501、获取当前前端项目的项目配置文件和项目工程文件；

[0142] 502、读取项目配置文件中的项目配置信息，并根据项目配置信息，生成当前前端项目的项目配置表；

[0143] 503、根据项目配置表从预设的代码检测工具规则体系筛选当前前端项目的代码检测工具接入规则；

[0144] 504、根据代码检测工具接入规则接入预设的代码检测工具；

[0145] 本实施例中的步骤501-504与第一实施例中的步骤101-104相似，此处不再赘述。

[0146] 505、将项目工程文件输入预设的语法分析器，通过语法分析器对项目工程文件中的代码进行词法分析，将项目工程文件转换为由词法单元组成的数组；

[0147] 506、根据数组生成项目工程文件的抽象语法树；

[0148] 507、启动多线程，每个线程通过命令行调用代码检测工具，通过代码检测工具对抽象语法树进行检测；

[0149] 508、等待多线程中的全部线程执行完毕，生成代码检测工具的检测结果；

[0150] 509、分析检测结果，并根据预设的结果过滤条件，将检测结果中误报和无用的结果过滤，得到最终检测结果；

[0151] 510、将最终检测结果以工程为单位保存。

[0152] 在本实施例中，通过对多个工程实行多线程并发处理，提高了检查效率；通过对单个工程的多个源码文件进一步实行多线程并发检查，进一步提高了检查效率，不仅缩短了一个工程的检查时间，也大大缩短了总体检查时间。

[0153] 本实施例中，详细描述了将所述抽象语法树输入所述代码检测工具中，通过所述代码检测工具完成代码检测的过程，通过启动多线程，每个线程通过命令行调用所述代码检测工具，通过所述代码检测工具对所述抽象语法树进行检测；等待所述多线程中的全部线程执行完毕，生成所述代码检测工具的检测结果；分析所述检测结果，并根据预设的结果过滤条件，将所述检测结果中误报和无用的结果过滤，得到最终检测结果；将所述最终检测结果以工程为单位保存。通过本方法对多个工程实行多线程并发处理，提高了检查效率；通过对单个工程的多个源码文件进一步实行多线程并发检查，进一步提高了检查效率，不仅缩短了一个工程的检查时间，也大大缩短了总体检查时间。

[0154] 上面对本发明实施例中前端代码接入检测方法进行了描述，下面对本发明实施例中前端代码接入检测装置进行描述，请参阅图6，本发明实施例中前端代码接入检测装置一个实施例包括：

[0155] 文件获取模块601，用于获取当前前端项目的项目配置文件和项目工程文件；

[0156] 信息读取模块602，用于读取所述项目配置文件中的项目配置信息，并根据所述项目配置信息，生成当前前端项目的项目配置表；

[0157] 规则筛选模块603，用于根据所述项目配置表从预设的代码检测工具规则体系筛选所述当前前端项目的代码检测工具接入规则；

[0158] 接入模块604，用于根据所述代码检测工具接入规则接入预设的代码检测工具；

[0159] 检测模块605，用于根据所述项目工程文件的代码构建抽象语法树，并将所述抽象语法树输入所述代码检测工具中，通过所述代码检测工具完成代码检测。

[0160] 需要强调的是，为保证数据的私密和安全性，上述待推送数据可以存储于一区块链的节点中。

[0161] 本发明实施例中，所述前端代码接入检测装置运行上述前端代码接入检测方法，所述前端代码接入检测方法包括：通过获取当前前端项目的项目配置文件和项目工程文件；读取所述项目配置文件中的项目配置信息，并根据所述项目配置信息，生成当前前端项目的项目配置表；根据所述项目配置表从预设的代码检测工具规则体系筛选所述当前前端项目的代码检测工具接入规则；根据所述代码检测工具接入规则接入预设的代码检测工具；根据所述项目工程文件的代码构建抽象语法树，并将所述抽象语法树输入所述代码检测工具中，通过所述代码检测工具完成代码检测。通过本方法读取前端项目的配置信息生成项目配置表，再根据项目配置表进行lint工具接入规则的获取，针对不同前端项目进行不同lint工具接入规则的获取，避免由于lint工具接入规则不统一，导致lint工具接入难度大的问题，便于对当前项目接入lint工具，进行项目文件代码的质量检测。此外，本发明还涉及区块链技术，所述项目配置文件和项目工程文件可存储于区块链中。

[0162] 请参阅图7，本发明实施例中前端代码接入检测装置的第二个实施例包括：

[0163] 文件获取模块601，用于获取当前前端项目的项目配置文件和项目工程文件；

[0164] 信息读取模块602，用于读取所述项目配置文件中的项目配置信息，并根据所述项

目配置信息,生成当前前端项目的项目配置表;

[0165] 规则筛选模块603,用于根据所述项目配置表从预设的代码检测工具规则体系筛选所述当前前端项目的代码检测工具接入规则;

[0166] 接入模块604,用于根据所述代码检测工具接入规则接入预设的代码检测工具;

[0167] 检测模块605,用于根据所述项目工程文件的代码构建抽象语法树,并将所述抽象语法树输入所述代码检测工具中,通过所述代码检测工具完成代码检测。

[0168] 其中,所述检测模块605包括:

[0169] 词法分析单元6051,用于将所述项目工程文件输入预设的语法分析器,通过所述语法分析器对所述项目工程文件中的代码进行词法分析,将所述项目工程文件转换为由词法单元组成的数组;

[0170] 语法树生成单元6052,用于根据所述数组生成所述项目工程文件的抽象语法树;

[0171] 输入单元6053,用于将所述抽象语法树输入所述代码检测工具中,通过所述代码检测工具完成代码检测。

[0172] 可选的,所述词法分析单元具体用于:

[0173] 将所述项目工程文件输入所述语法分析器,通过所述语法分析器对所述项目工程文件的代码进行逐行扫描并分解成词法单元,其中,所述词法单元为编程语言中字符串形式的基本单词符号;

[0174] 将所述词法单元序列组合成数组。

[0175] 可选的,所述语法树生成单元6052具体用于:

[0176] 所述语法树生成单元具体用于:

[0177] 根据所述数组对所述代码进行语法分析,判断所述代码的语法是否正确;

[0178] 若是,则根据预设的源代码语言规范,生成与所述代码对应的语法分析树;

[0179] 调用所述语法分析树中各节点对应的节点对象创建方法,创建节点对象,根据所述节点对象生成抽象语法树。

[0180] 可选的,所述输入单元6053具体用于:

[0181] 启动多线程,每个线程通过命令行调用所述代码检测工具,通过所述代码检测工具对所述抽象语法树进行检测;

[0182] 等待所述多线程中的全部线程执行完毕,生成所述代码检测工具的检测结果;

[0183] 分析所述检测结果,并根据预设的结果过滤条件,将所述检测结果中误报和无用的结果过滤,得到最终检测结果;

[0184] 将所述最终检测结果以工程为单位保存。

[0185] 其中,所述前端代码接入检测装置还包括依赖配置模块606,所述依赖配置模块606包括:

[0186] 版本获取单元6061,用于获取所述代码检测工具的版本类型;

[0187] 列表获取单元6062,用于根据所述版本类型获取所述代码检测工具的依赖列表;

[0188] 依赖包安装单元6063,用于根据所述依赖列表获取所述代码检测工具的依赖包并安装所述依赖包。

[0189] 可选的,所述依赖包安装单元6063具体用于:

[0190] 根据所述依赖列表获取所述代码检测工具的依赖包;

[0191] 对所述依赖包进行合法性检验，得到检验结果；

[0192] 若所述检验结果为合法，则安装所述依赖包。

[0193] 本实施例在上一实施例的基础上，详细描述了各个模块的具体功能以及部分模块的单元构成，通过本装置，能够读取前端项目的配置信息生成项目配置表，再根据项目配置表进行lint工具接入规则的获取，针对不同前端项目进行不同lint工具接入规则的获取，避免由于lint工具接入规则不统一，导致lint工具接入难度大的问题，便于对当前项目接入lint工具，进行项目文件代码的质量检测。

[0194] 上面图6和图7从模块化功能实体的角度对本发明实施例中的中前端代码接入检测装置进行详细描述，下面从硬件处理的角度对本发明实施例中前端代码接入检测设备进行详细描述。

[0195] 图8是本发明实施例提供的一种前端代码接入检测设备的结构示意图，该前端代码接入检测设备800可因配置或性能不同而产生比较大的差异，可以包括一个或一个以上处理器(central processing units,CPU)810(例如，一个或一个以上处理器)和存储器820，一个或一个以上存储应用程序833或数据832的存储介质830(例如一个或一个以上海量存储设备)。其中，存储器820和存储介质830可以是短暂存储或持久存储。存储在存储介质830的程序可以包括一个或一个以上模块(图示没标出)，每个模块可以包括对前端代码接入检测设备800中的一系列指令操作。更进一步地，处理器810可以设置为与存储介质830通信，在前端代码接入检测设备800上执行存储介质830中的一系列指令操作，以实现上述前端代码接入检测方法的步骤。

[0196] 前端代码接入检测设备800还可以包括一个或一个以上电源840，一个或一个以上有线或无线网络接口850，一个或一个以上输入输出接口860，和/或，一个或一个以上操作系统831，例如Windows Serve,Mac OS X,Unix,Linux,FreeBSD等等。本领域技术人员可以理解，图8示出的前端代码接入检测设备结构并不构成对本申请提供的前端代码接入检测设备的限定，可以包括比图示更多或更少的部件，或者组合某些部件，或者不同的部件布置。

[0197] 本发明所指区块链是分布式数据存储、点对点传输、共识机制、加密算法等计算机技术的新型应用模式。区块链(Blockchain)，本质上是一个去中心化的数据库，是一串使用密码学方法相关联产生的数据块，每一个数据块中包含了一批次网络交易的信息，用于验证其信息的有效性(防伪)和生成下一个区块。区块链可以包括区块链底层平台、平台产品服务层以及应用服务层等。

[0198] 本发明还提供一种计算机可读存储介质，该计算机可读存储介质可以为非易失性计算机可读存储介质，该计算机可读存储介质也可以为易失性计算机可读存储介质，所述计算机可读存储介质中存储有指令，当所述指令在计算机上运行时，使得计算机执行所述前端代码接入检测方法的步骤。

[0199] 所属领域的技术人员可以清楚地了解到，为描述的方便和简洁，上述描述的系统或装置、单元的具体工作过程，可以参考前述方法实施例中的对应过程，在此不再赘述。

[0200] 所述集成的单元如果以软件功能单元的形式实现并作为独立的产品销售或使用时，可以存储在一个计算机可读取存储介质中。基于这样的理解，本发明的技术方案本质上或者说对现有技术做出贡献的部分或者该技术方案的全部或部分可以以软件产品的形式

体现出来,该计算机软件产品存储在一个存储介质中,包括若干指令用以使得一台计算机设备(可以是个人计算机,服务器,或者网络设备等)执行本发明各个实施例所述方法的全部或部分步骤。而前述的存储介质包括:U盘、移动硬盘、只读存储器(read-only memory,ROM)、随机存取存储器(random access memory,RAM)、磁碟或者光盘等各种可以存储程序代码的介质。

[0201] 以上所述,以上实施例仅用以说明本发明的技术方案,而非对其限制;尽管参照前述实施例对本发明进行了详细的说明,本领域的普通技术人员应当理解:其依然可以对前述各实施例所记载的技术方案进行修改,或者对其中部分技术特征进行等同替换;而这些修改或者替换,并不使相应技术方案的本质脱离本发明各实施例技术方案的精神和范围。

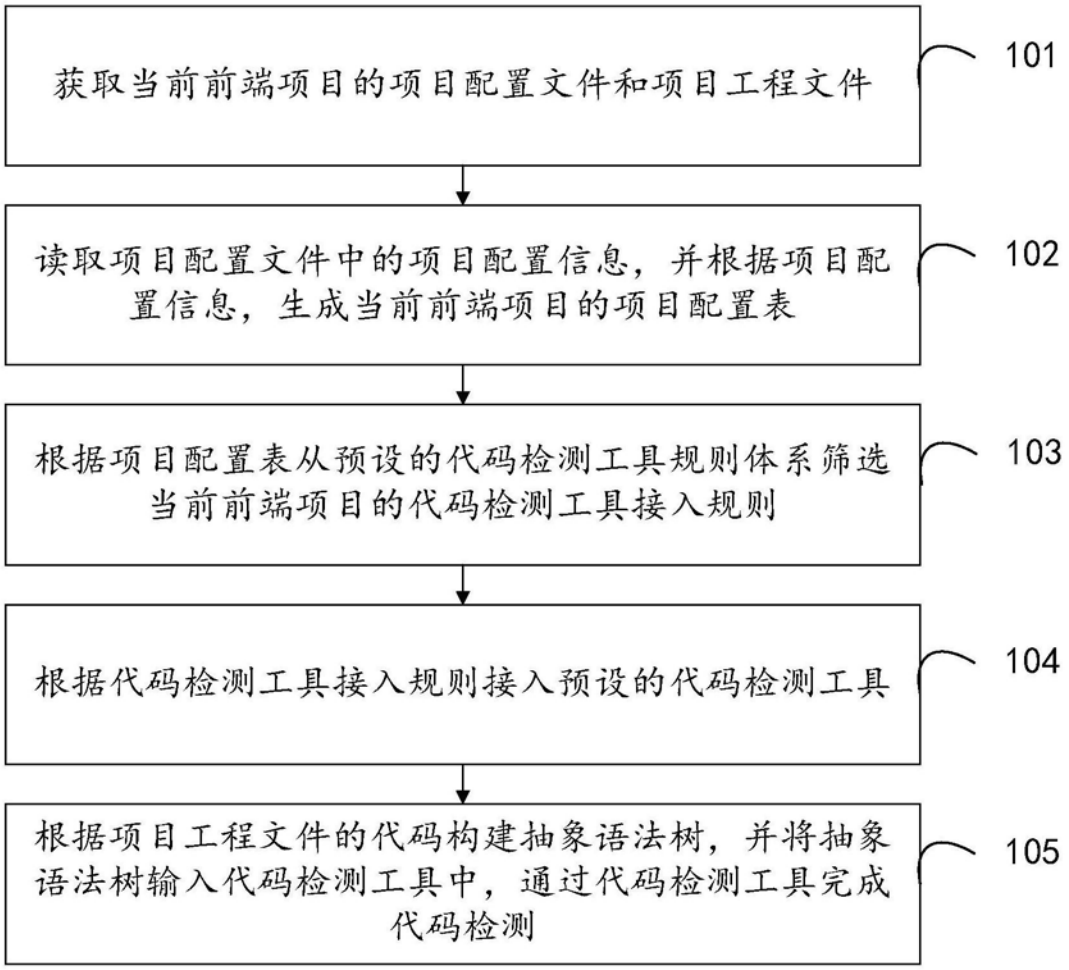

获取当前前端项目的项目配置文件和项目工程文件
101
读取项目配置文件中的项目配置信息，并根据项目配置信息，生成当前前端项目的项目配置表
102
根据项目配置表从预设的代码检测工具规则体系筛选当前前端项目的代码检测工具接入规则
103
根据代码检测工具接入规则接入预设的代码检测工具
104
根据项目工程文件的代码构建抽象语法树，并将抽象语法树输入代码检测工具中，通过代码检测工具完成代码检测
105

图1

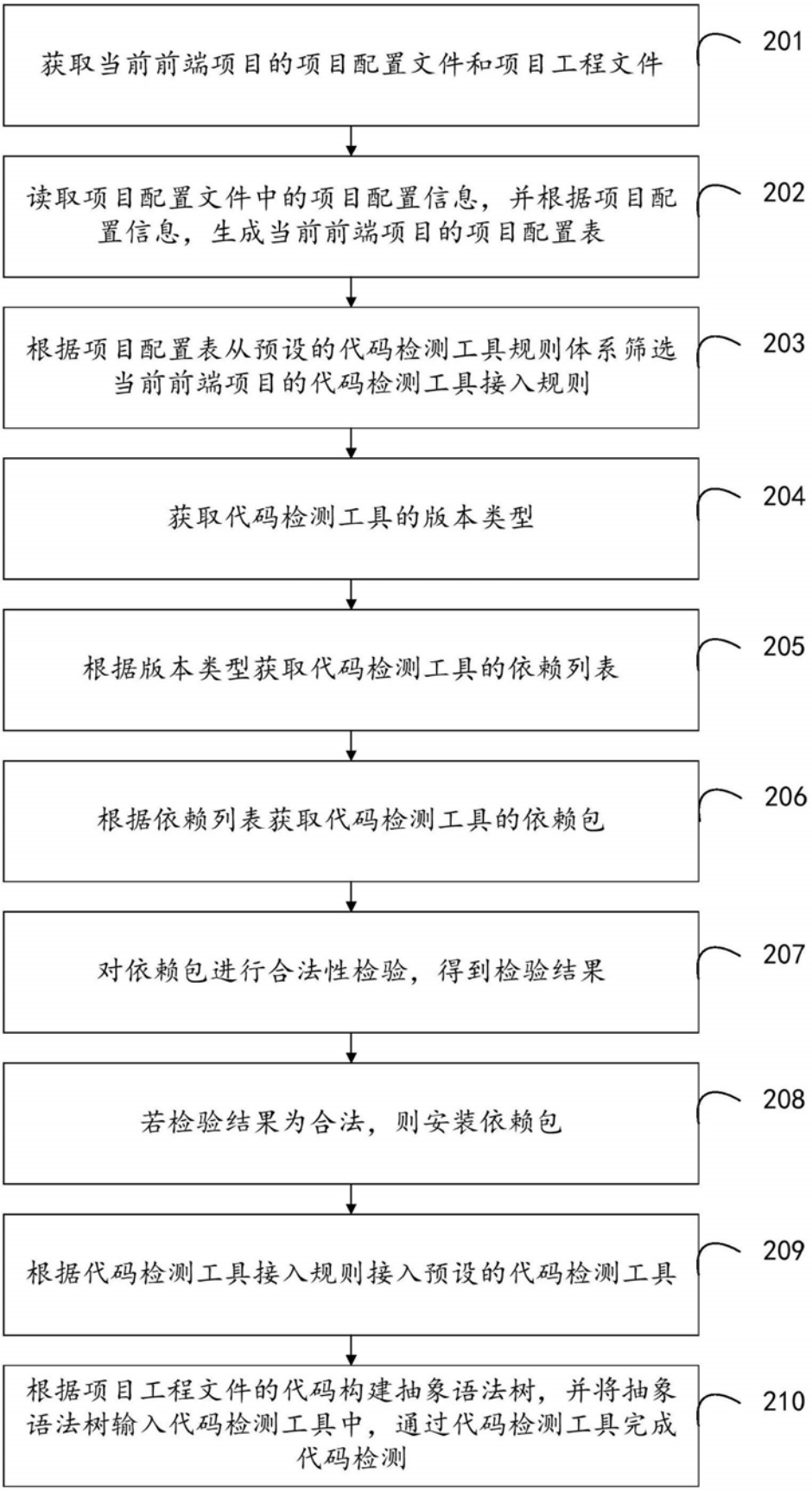

获取当前前端项目的项目配置文件和项目工程文件
201
读取项目配置文件中的项目配置信息，并根据项目配置信息，生成当前前端项目的项目配置表
202
根据项目配置表从预设的代码检测工具规则体系筛选当前前端项目的代码检测工具接入规则
203
获取代码检测工具的版本类型
204
根据版本类型获取代码检测工具的依赖列表
205
根据依赖列表获取代码检测工具的依赖包
206
对依赖包进行合法性检验，得到检验结果
207
若检验结果为合法，则安装依赖包
208
根据代码检测工具接入规则接入预设的代码检测工具
209
根据项目工程文件的代码构建抽象语法树，并将抽象语法树输入代码检测工具中，通过代码检测工具完成代码检测
210

图2

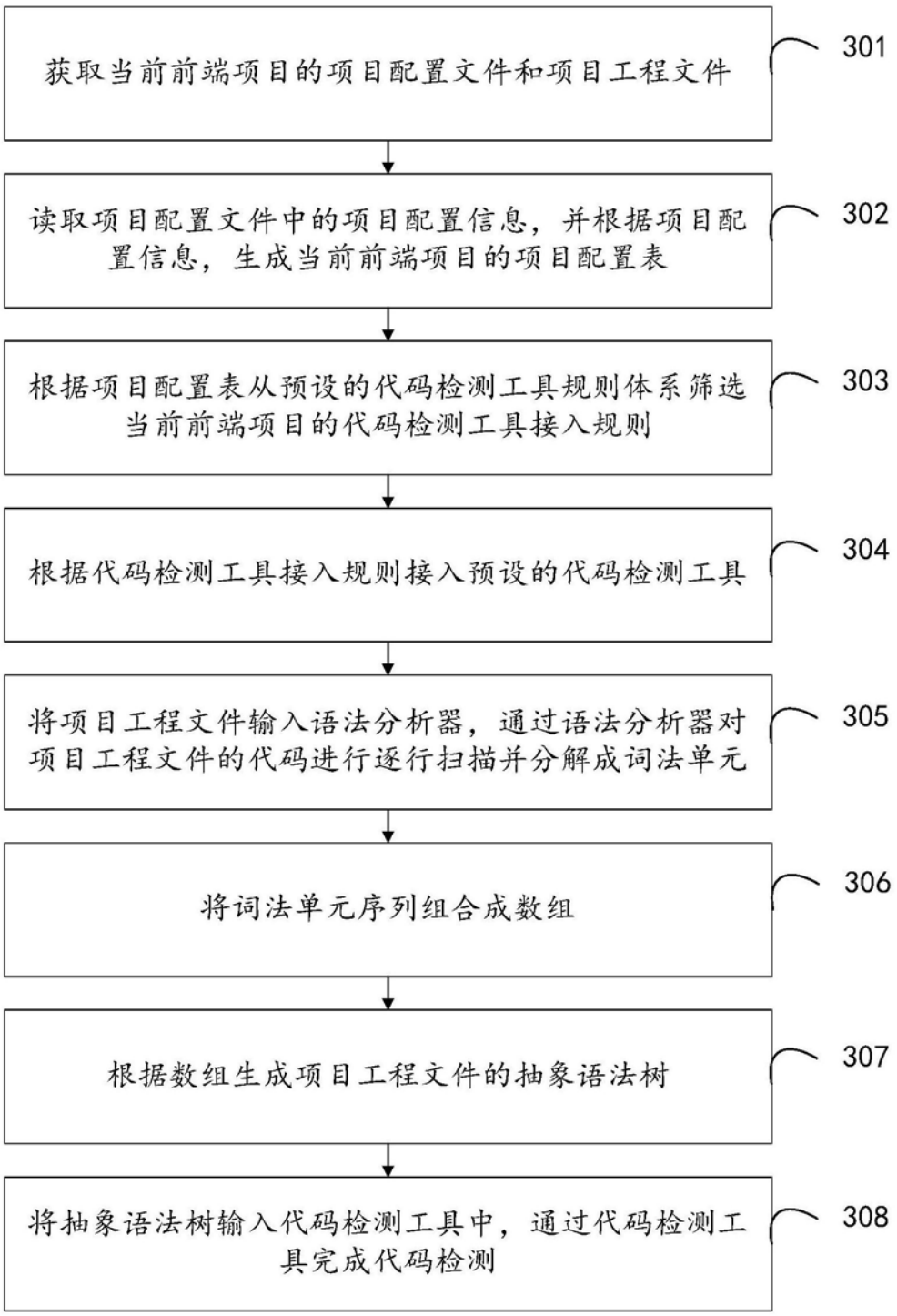

获取当前前端项目的项目配置文件和项目工程文件
301
读取项目配置文件中的项目配置信息，并根据项目配置信息，生成当前前端项目的项目配置表
302
根据项目配置表从预设的代码检测工具规则体系筛选当前前端项目的代码检测工具接入规则
303
根据代码检测工具接入规则接入预设的代码检测工具
304
将项目工程文件输入语法分析器，通过语法分析器对项目工程文件的代码进行逐行扫描并分解成词法单元
305
将词法单元序列组合成数组
306
根据数组生成项目工程文件的抽象语法树
307
将抽象语法树输入代码检测工具中，通过代码检测工具完成代码检测
308

图3

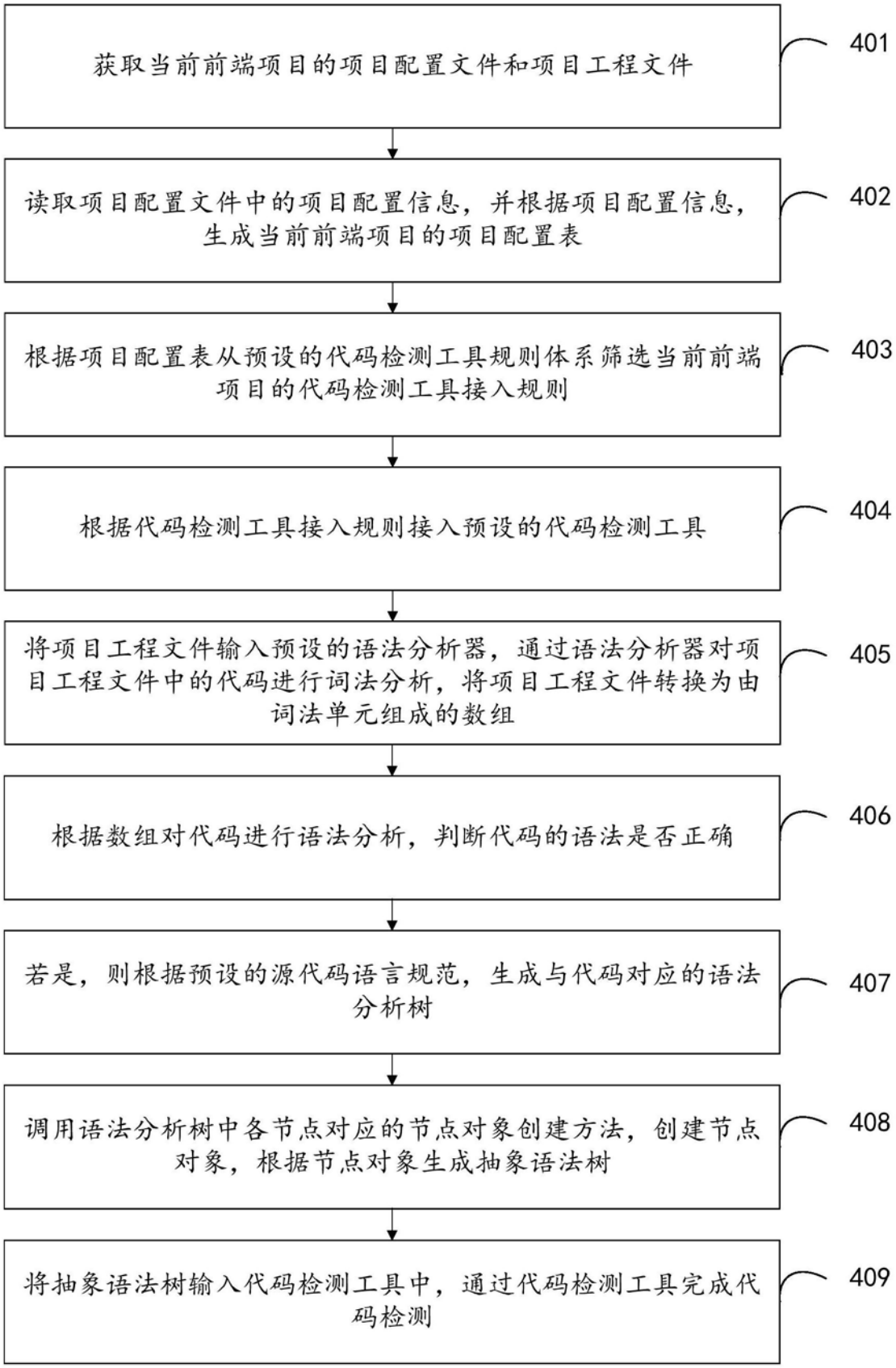

获取当前前端项目的项目配置文件和项目工程文件
401
读取项目配置文件中的项目配置信息，并根据项目配置信息，生成当前前端项目的项目配置表
402
根据项目配置表从预设的代码检测工具规则体系筛选当前前端项目的代码检测工具接入规则
403
根据代码检测工具接入规则接入预设的代码检测工具
404
将项目工程文件输入预设的语法分析器，通过语法分析器对项目工程文件中的代码进行词法分析，将项目工程文件转换为由词法单元组成的数组
405
根据数组对代码进行语法分析，判断代码的语法是否正确
406
若是，则根据预设的源代码语言规范，生成与代码对应的语法分析树
407
调用语法分析树中各节点对应的节点对象创建方法，创建节点对象，根据节点对象生成抽象语法树
408
将抽象语法树输入代码检测工具中，通过代码检测工具完成代码检测
409

图4

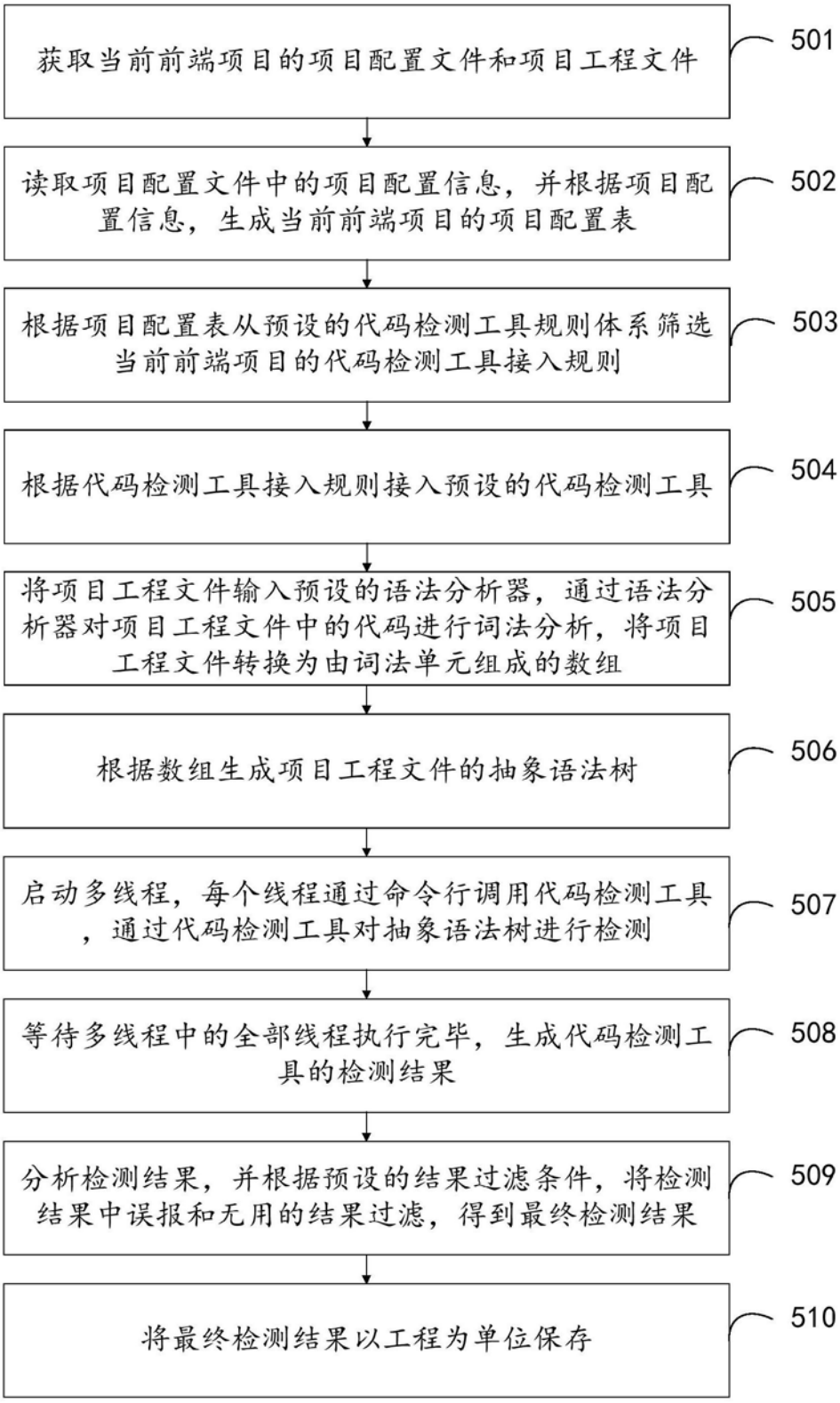

获取当前前端项目的项目配置文件和项目工程文件
501
读取项目配置文件中的项目配置信息，并根据项目配置信息，生成当前前端项目的项目配置表
502
根据项目配置表从预设的代码检测工具规则体系筛选当前前端项目的代码检测工具接入规则
503
根据代码检测工具接入规则接入预设的代码检测工具
504
将项目工程文件输入预设的语法分析器，通过语法分析器对项目工程文件中的代码进行词法分析，将项目工程文件转换为由词法单元组成的数组
505
根据数组生成项目工程文件的抽象语法树
506
启动多线程，每个线程通过命令行调用代码检测工具，通过代码检测工具对抽象语法树进行检测
507
等待多线程中的全部线程执行完毕，生成代码检测工具的检测结果
508
分析检测结果，并根据预设的结果过滤条件，将检测结果中误报和无用的结果过滤，得到最终检测结果
509
将最终检测结果以工程为单位保存
510

图5

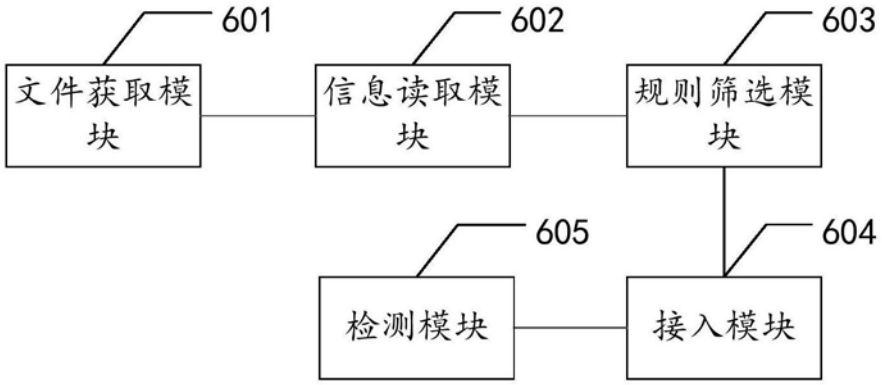

601
文件获取模块
602
信息读取模块
603
规则筛选模块
605
检测模块
604
接入模块

图6

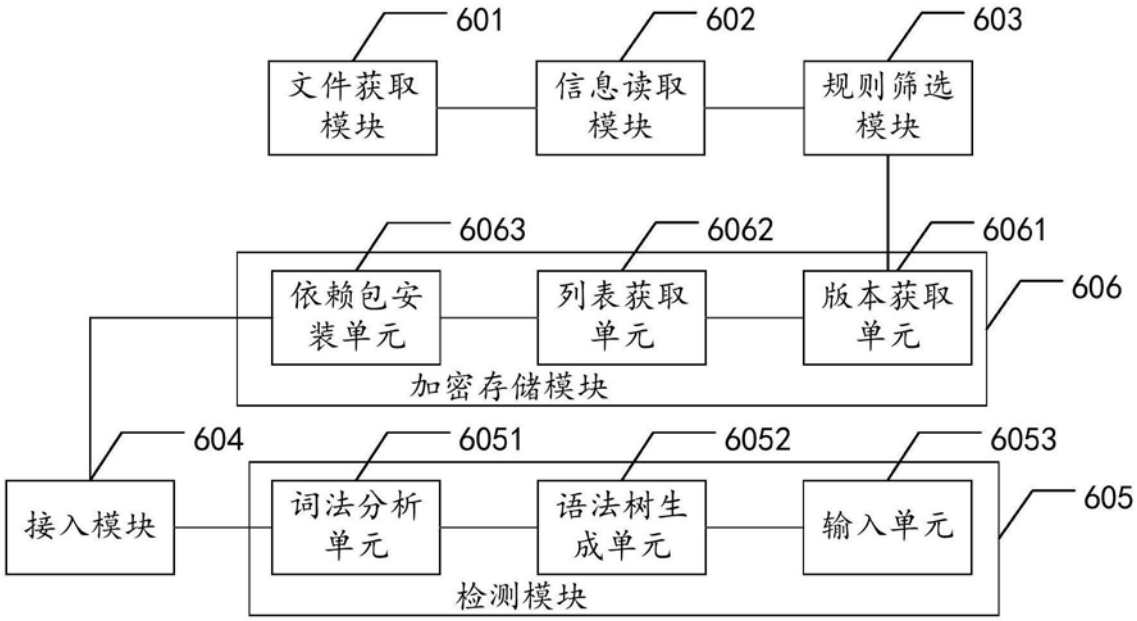

601
文件获取模块
602
信息读取模块
603
规则筛选模块
6063
依赖包安装单元
6062
列表获取单元
6061
版本获取单元
606
加密存储模块
604
接入模块
6051
词法分析单元
6052
语法树生成单元
6053
输入单元
605
检测模块

图7

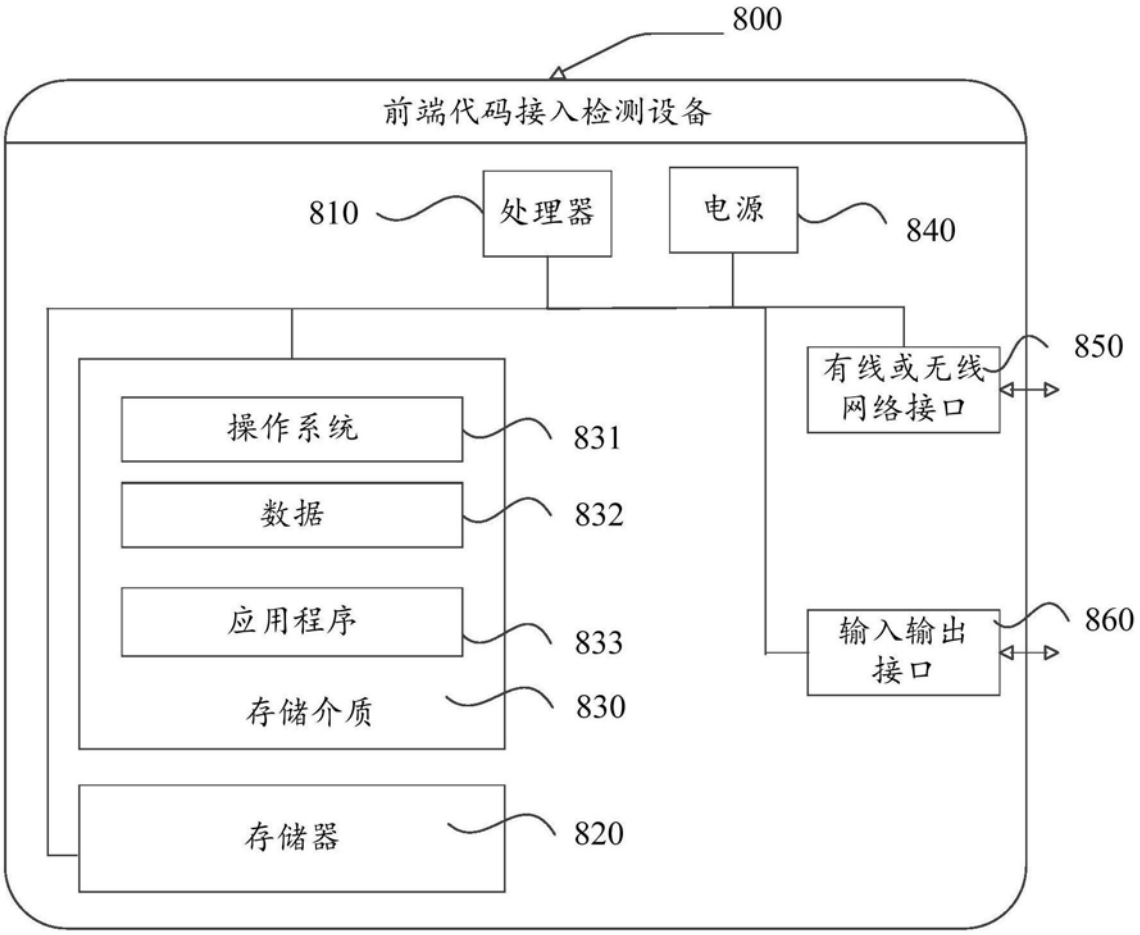

800
前端代码接入检测设备
810
处理器
电源
840
850
有线或无线
网络接口
操作系统
831
数据
832
应用程序
833
存储介质
830
860
输入输出
接口
存储器
820

图8