



(12) 发明专利

(10) 授权公告号 CN 112052008 B

(45) 授权公告日 2022. 02. 18

(21) 申请号 202010948751.1

(22) 申请日 2020.09.10

(65) 同一申请的已公布的文献号

申请公布号 CN 112052008 A

(43) 申请公布日 2020.12.08

(73) 专利权人 腾讯科技(深圳)有限公司

地址 518057 广东省深圳市南山区高新区

科技中一路腾讯大厦35层

(72) 发明人 尹飞

(74) 专利代理机构 北京三高永信知识产权代理

有限责任公司 11138

代理人 徐立

(51) Int. Cl.

G06F 8/41 (2018.01)

G06F 11/36 (2006.01)

(56) 对比文件

CN 111625230 A, 2020.09.04

CN 108614707 A, 2018.10.02

CN 105404584 A, 2016.03.16

CN 111090423 A, 2020.05.01

US 10133561B1 B1, 2018.11.20

审查员 余晓

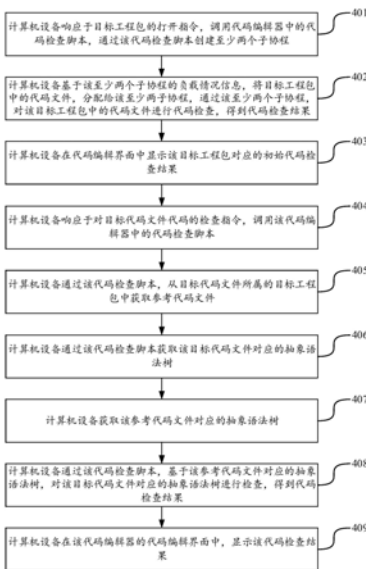
权利要求书3页 说明书19页 附图8页

(54) 发明名称

代码检查方法、装置、计算机设备及计算机可读存储介质

(57) 摘要

本申请公开了一种代码检查方法、装置、计算机设备及计算机可读存储介质,属于计算机技术领域。本申请通过在代码编辑器中设置代码检查脚本,在对目标代码文件的编辑过程中,响应于代码检查指令,触发该代码检查脚本运行,由于目标代码文件的内容更新,仅会对与该目标代码文件相关联的其他代码文件产生影响,因此获取与该目标代码文件之间存在调用关系的参考代码文件,基于获取到的部分参考代码文件中的内容,对目标代码文件进行代码检查,而无需读取目标工程包中的全部代码文件,降低代码检查的耗时,将检查结果实时显示在编辑界面,使用户在代码编辑过程中可以实时看到代码检查结果,提高代码检查效率。



1. 一种代码检查方法,其特征在于,所述方法包括:

响应于对目标工程包的打开指令,调用代码编辑器中的代码检查脚本;

通过所述代码检查脚本创建至少两个子协程,所述子协程用于执行代码检查任务;

基于所述至少两个子协程的负载情况信息,将所述目标工程包中的至少两个代码文件,分配给所述至少两个子协程;

在任一子协程对任一代码文件进行代码检查的过程中,通过所述任一子协程对所述任一代码文件进行词法分析、语法分析及语义分析,得到各个阶段的检查结果,将所述各个阶段的检查结果,确定为所述目标工程包对应的初始代码检查结果;

在所述代码编辑器的代码编辑界面中,显示所述初始代码检查结果;

响应于对所述目标工程包中的目标代码文件的检查指令,调用所述代码检查脚本;

通过所述代码检查脚本,从所述目标工程包中获取参考代码文件,所述参考代码文件为与所述目标代码文件之间存在调用关系的代码文件;

通过所述代码检查脚本,基于所述参考代码文件中源代码的语法结构,对所述目标代码文件中源代码的语法结构进行检查,得到代码检查结果;

在所述代码编辑界面中,显示所述代码检查结果。

2. 根据权利要求1所述的方法,其特征在于,所述通过所述代码检查脚本,基于所述参考代码文件中源代码的语法结构,对所述目标代码文件中源代码的语法结构进行检查,得到代码检查结果,包括:

获取所述目标代码文件对应的抽象语法树,所述抽象语法树用于指示代码文件的源代码的语法结构;

获取所述参考代码文件对应的抽象语法树;

通过所述代码检查脚本,基于所述参考代码文件对应的抽象语法树,对所述目标代码文件对应的抽象语法树进行检查,得到所述代码检查结果。

3. 根据权利要求2所述的方法,其特征在于,所述获取所述目标代码文件对应的抽象语法树,包括:

对所述目标代码文件的源代码进行词法分析,得到所述目标代码文件对应的单词序列;

对所述目标代码文件对应的单词序列进行语法分析,得到所述目标代码文件对应的抽象语法树。

4. 根据权利要求2所述的方法,其特征在于,所述获取所述参考代码文件对应的抽象语法树,包括:

基于所述参考代码文件的文件标识,从目标存储空间中获取所述参考代码文件对应的抽象语法树,所述目标存储空间用于存储所述目标工程包中各个代码文件对应的抽象语法树。

5. 根据权利要求4所述的方法,其特征在于,所述获取所述目标代码文件对应的抽象语法树之后,所述方法还包括:

将所述目标存储空间中已存储的所述目标代码文件的抽象语法树,替换为本次获取到的所述目标代码文件对应的抽象语法树。

6. 根据权利要求2所述的方法,其特征在于,所述通过所述代码检查脚本,基于所述参

考代码文件对应的抽象语法树,对所述目标代码文件对应的抽象语法树进行检查,得到所述代码检查结果,包括:

通过所述代码检查脚本,遍历所述目标代码文件以及所述参考代码文件对应的抽象语法树,生成符号列表,所述符号列表用于指示所述目标代码文件以及所述参考代码文件所包括的变量;

基于所述符号列表以及所述参考代码文件对应的抽象语法树,对所述目标代码文件对应的抽象语法树进行检查,得到所述代码检查结果。

7. 根据权利要求1所述的方法,其特征在于,所述代码检查结果包括所述目标代码文件中的词法错误信息、所述目标代码文件中的语法错误信息以及所述目标代码文件中的语义错误信息。

8. 根据权利要求1所述的方法,其特征在于,所述在所述代码编辑界面中,显示所述代码检查结果,包括下述至少一项:

基于所述代码检查结果,在所述代码编辑界面对所述目标代码文件中的错误源代码进行突出显示;

基于所述代码检查结果,在所述代码编辑界面显示所述目标代码文件对应的错误数目;

基于所述代码检查结果,在所述代码编辑界面显示所述目标代码文件对应的错误代码目录,所述错误代码目录中的一条数据用于表示一处源代码的错误信息。

9. 根据权利要求1所述的方法,其特征在于,所述在任一子协程对任一代码文件进行代码检查的过程中,通过所述任一子协程对所述任一代码文件进行词法分析、语法分析及语义分析,得到各个阶段的检查结果,将所述各个阶段的检查结果,确定为所述目标工程包对应的初始代码检查结果,包括:

在所述任一子协程对所述任一代码文件进行代码检查的过程中,通过所述任一子协程对所述任一代码文件的源代码进行词法分析,得到第一初始结果以及所述任一代码文件对应的单词序列,所述第一初始结果用于指示代码文件中的词法错误;

对所述任一代码文件对应的单词序列进行语法分析,得到第二初始结果以及所述任一代码文件对应的抽象语法树,所述第二初始结果用于指示代码文件中的语法错误;

对所述目标工程包中的代码文件对应的抽象语法树进行语义分析,得到第三初始结果,所述第三初始结果用于指示代码文件中的语义错误;

将所述第一初始结果、所述第二初始结果以及所述第三初始结果,确定为所述目标工程包对应的初始代码检查结果。

10. 根据权利要求1所述的方法,其特征在于,所述在任一子协程对任一代码文件进行代码检查的过程中,通过所述任一子协程对所述任一代码文件进行词法分析、语法分析及语义分析,得到各个阶段的检查结果,将所述各个阶段的检查结果,确定为所述目标工程包对应的初始代码检查结果之后,所述方法还包括:

获取对所述目标工程包进行代码检查的耗时;

所述在所述代码编辑器的代码编辑界面中,显示所述初始代码检查结果,包括:

在所述代码编辑界面中,显示所述初始代码检查结果及所述耗时。

11. 一种代码检查装置,其特征在于,所述装置包括:

检查模块,用于响应于对目标工程包的打开指令,调用代码编辑器中的代码检查脚本,对所述目标工程包中的代码文件进行代码检查,得到所述目标工程包对应的初始代码检查结果;

显示模块,用于在所述代码编辑器的代码编辑界面中,显示所述初始代码检查结果;

所述检查模块,包括:

调用子模块,用于响应于对目标工程包的打开指令,调用代码编辑器中的代码检查脚本;

创建子模块,用于通过所述代码检查脚本创建至少两个子协程,所述子协程用于执行代码检查任务;

分配子模块,用于基于所述至少两个子协程的负载情况信息,将所述目标工程包中的至少两个代码文件,分配给所述至少两个子协程;

检查子模块,用于在任一子协程对任一代码文件进行代码检查的过程中,通过所述任一子协程对所述任一代码文件进行词法分析、语法分析及语义分析,得到各个阶段的检查结果,将所述各个阶段的检查结果,确定为所述目标工程包对应的初始代码检查结果;

调用模块,用于响应于对所述目标工程包中的目标代码文件的检查指令,调用所述代码检查脚本;

获取模块,用于通过所述代码检查脚本,从所述目标工程包中获取参考代码文件,所述参考代码文件为与所述目标代码文件之间存在调用关系的代码文件;

所述检查模块,还用于通过所述代码检查脚本,基于所述参考代码文件中源代码的语法结构,对所述目标代码文件中源代码的语法结构进行检查,得到代码检查结果;

所述显示模块,还用于在所述代码编辑界面中,显示所述代码检查结果。

12.一种计算机设备,其特征在于,所述计算机设备包括一个或多个处理器和一个或多个存储器,所述一个或多个存储器中存储有至少一条程序代码,所述至少一条程序代码由所述一个或多个处理器加载并执行以实现如权利要求1至权利要求10任一项所述的代码检查方法所执行的操作。

13.一种计算机可读存储介质,其特征在于,所述计算机可读存储介质中存储有至少一条程序代码,所述至少一条程序代码由处理器加载并执行以实现如权利要求1至权利要求10任一项所述的代码检查方法所执行的操作。

代码检查方法、装置、计算机设备及计算机可读存储介质

技术领域

[0001] 本申请涉及计算机技术领域,特别涉及一种代码检查方法、装置、计算机设备及计算机可读存储介质。

背景技术

[0002] 在应用程序开发过程中,对已编写的代码进行检查是一个重要步骤。以对Lua语言编写的代码进行检查为例,Lua语言因其语法简单、使用灵活、便于热更的特性,被广泛应用于游戏开发等领域。目前,在对Lua语言编写的代码进行检查时,通常是在编辑器中对代码文件编写完成后,再将各个代码文件导入代码检查工具,通过代码检查工具来检查代码文件中存在语法错误、逻辑错误等问题,将检查结果以邮件的形式发送给开发人员。

[0003] 在上述代码检查过程中,应用的是离线检查的方式,即将编辑好的代码从代码编辑器导出到离线的代码检查工具,代码检查的过程和代码编辑的过程是分离的,导致在各个代码文件的编辑过程中,开发人员无法实时了解到代码中存在的问题,代码检查效率极低。

发明内容

[0004] 本申请实施例提供了一种代码检查方法、装置、计算机设备及计算机可读存储介质,可以提高代码检查的效率。该技术方案如下:

[0005] 一方面,提供了一种代码检查方法,该方法包括:

[0006] 响应于对目标代码文件代码的检查指令,调用代码编辑器中的代码检查脚本;

[0007] 通过该代码检查脚本,从该目标代码文件所属的目标工程包中获取参考代码文件,该参考代码文件为与该目标代码文件之间存在调用关系的代码文件;

[0008] 通过该代码检查脚本,基于该参考代码文件中源代码的语法结构,对该目标代码文件中源代码的语法结构进行检查,得到代码检查结果;

[0009] 在该代码编辑器的代码编辑界面中,显示该代码检查结果。

[0010] 在一种可能实现方式中,将该第一初始结果、该第二初始结果以及该第三初始结果,确定为该目标工程包对应的初始代码检查结果之后,该方法还包括:

[0011] 将该初始代码检查结果以及该代码文件对应的抽象语法树,存储至目标存储空间。

[0012] 一方面,提供了一种代码检查装置,该装置包括:

[0013] 调用模块,用于响应于对目标代码文件代码的检查指令,调用代码编辑器中的代码检查脚本;

[0014] 获取模块,用于通过该代码检查脚本,从该目标代码文件所属的目标工程包中获取参考代码文件,该参考代码文件为与该目标代码文件之间存在调用关系的代码文件;

[0015] 检查模块,用于通过该代码检查脚本,基于该参考代码文件中源代码的语法结构,对该目标代码文件中源代码的语法结构进行检查,得到代码检查结果;

- [0016] 显示模块,用于在该代码编辑器的代码编辑界面中,显示该代码检查结果。
- [0017] 在一种可能实现方式中,该检查模块包括:
- [0018] 第一获取子模块,用于获取该目标代码文件对应的抽象语法树,该抽象语法树用于指示代码文件的源代码的语法结构;
- [0019] 第二获取子模块,用于获取该参考代码文件对应的抽象语法树;
- [0020] 检查子模块,用于通过该代码检查脚本,基于该参考代码文件对应的抽象语法树,对该目标代码文件对应的抽象语法树进行检查,得到该代码检查结果。
- [0021] 在一种可能实现方式中,该第一获取子模块,用于:
- [0022] 对该目标代码文件的源代码进行词法分析,得到该目标代码文件对应的单词序列;
- [0023] 对该目标代码文件对应的单词序列进行语法分析,得到该目标代码文件对应的抽象语法树。
- [0024] 在一种可能实现方式中,该第二获取子模块,用于:
- [0025] 基于该参考代码文件的文件标识,从目标存储空间中获取该参考代码文件对应的抽象语法树,该目标存储空间用于存储该目标工程包中各个代码文件对应的抽象语法树。
- [0026] 在一种可能实现方式中,该装置还包括:
- [0027] 存储模块,用于将该目标存储空间中已存储的该目标代码文件的抽象语法树,替换为本次获取到的该目标代码文件对应的抽象语法树。
- [0028] 在一种可能实现方式中,该检查子模块,用于:
- [0029] 通过该代码检查脚本,遍历该目标代码文件以及该参考代码文件对应的抽象语法树,生成符号列表,该符号列表用于指示该目标代码文件以及该参考代码文件所包括的变量;
- [0030] 基于该符号列表以及该参考代码文件对应的抽象语法树,对该目标代码文件对应的抽象语法树进行检查,得到该代码检查结果。
- [0031] 在一种可能实现方式中,该代码检查结果包括该目标代码文件中的词法错误信息、该目标代码文件中的语法错误信息以及该目标代码文件中的语义错误信息。
- [0032] 在一种可能实现方式中,该显示模块,用于执行下述至少一项:
- [0033] 基于该代码检查结果,在该代码编辑界面对该目标代码文件中的错误源代码进行突出显示;
- [0034] 基于该代码检查结果,在该代码编辑界面显示该目标代码文件对应的错误数目;
- [0035] 基于该代码检查结果,在该代码编辑界面显示该目标代码文件对应的错误代码目录,该错误代码目录中的一条数据用于表示一处源代码的错误信息。
- [0036] 在一种可能实现方式中,该检查模块用于响应于对该目标工程包的打开指令,调用该代码编辑器中的该代码检查脚本对该目标工程包中的代码文件进行代码检查,得到该目标工程包对应的初始代码检查结果;
- [0037] 该显示模块,用于在该代码编辑界面中,显示该目标工程包对应的初始代码检查结果。
- [0038] 在一种可能实现方式中,该检查模块包括:
- [0039] 调用子模块,用于响应于对该目标工程包的打开指令,调用该代码检查脚本;

[0040] 创建子模块,用于创建通过该代码检查脚本创建至少两个子协程,该子协程用于执行代码检查任务;

[0041] 分配子模块,用于基于该至少两个子协程的负载情况信息,将该目标工程中的代码文件,分配给该至少两子协程;

[0042] 该检查子模块,用于通过该至少两个子协程,对该目标工程包中的代码文件进行代码检查,得到代码检查结果。

[0043] 在一种可能实现方式中,该检查子模块,用于:

[0044] 在任一子协程对任一代码文件进行代码检查的过程中,通过该任一子协程对该任一代码文件的源代码进行词法分析,得到第一初始结果以及该任一代码文件对应的单词序列,该第一初始结果用于指示代码文件中的词法错误;

[0045] 对该任一代码文件对应的单词序列进行语法分析,得到第二初始结果以及该任一代码文件对应的抽象语法树,该第二初始结果用于指示代码文件中的语法错误;

[0046] 对该目标工程包中的代码文件对应的抽象语法树进行语义分析,得到第三初始结果,该第三初始结果用于指示代码文件中的语义错误;

[0047] 将该第一初始结果、该第二初始结果以及该第三初始结果,确定为该目标工程包对应的初始代码检查结果。

[0048] 在一种可能实现方式中,该存储模块用于:

[0049] 将该初始代码检查结果以及该代码文件对应的抽象语法树,存储至目标存储空间。

[0050] 在一种可能实现方式中,该装置还包括:

[0051] 耗时获取模块,用于获取对该目标工程包进行代码检查的耗时;

[0052] 该显示模块,用于在该代码编辑界面中,显示对该目标工程包进行代码检查的耗时。

[0053] 一方面,提供了一种计算机设备,该计算机设备包括一个或多个处理器和一个或多个存储器,该一个或多个存储器中存储有至少一条程序代码,该至少一条程序代码由该一个或多个处理器加载并执行以实现该代码检查方法所执行的操作。

[0054] 一方面,提供了一种计算机可读存储介质,该计算机可读存储介质中存储有至少一条程序代码,该至少一条程序代码由处理器加载并执行以实现该代码检查方法所执行的操作。

[0055] 一方面,提供了一种计算机程序产品,该计算机程序产品包括至少一条程序代码,该至少一条程序代码存储在计算机可读存储介质中。计算机设备的处理器从计算机可读存储介质读取该至少一条程序代码,处理器执行该至少一条程序代码,使得该计算机设备实现该代码检查方法所执行的操作。

[0056] 本申请实施例提供的技术方案,本申请实施例提供的技术方案,通过在代码编辑器中设置代码检查脚本,在对目标代码文件的编辑过程中,响应于代码检查指令,触发该代码检查脚本运行,由于目标代码文件的内容更新,仅会对与该目标代码文件相关联的其他代码文件产生影响,因此获取与该目标代码文件之间存在调用关系的参考代码文件,基于获取到的部分参考代码文件中的内容,对目标代码文件进行代码检查,而无需读取目标工程包中的全部代码文件,降低代码检查的耗时,将检查结果实时显示在编辑界面,使用户在

代码编辑过程中可以实时看到代码检查结果,提高代码检查效率。

附图说明

[0057] 为了更清楚地说明本申请实施例中的技术方案,下面将对实施例描述中所需要使用的附图作简单地介绍,显而易见地,下面描述中的附图仅仅是本申请的一些实施例,对于本领域普通技术人员来讲,在不付出创造性劳动的前提下,还可以根据这些附图获得其他的附图。

[0058] 图1是本申请实施例提供的一种代码检查方法的实施环境示意图;

[0059] 图2是本申请实施例提供的一种代码检查方法的流程图;

[0060] 图3是本申请实施例提供的一种代码编辑界面的示意图;

[0061] 图4是本申请实施例提供的一种代码检查方法的流程图;

[0062] 图5是本申请实施例提供的一种多协程架构图;

[0063] 图6是本申请实施例提供的一种插件运行机制的示意图;

[0064] 图7是本申请实施例提供的一种词法分析流程图;

[0065] 图8是本申请实施例提供的一种语法分析流程图;

[0066] 图9是本申请实施例提供的一种代码检查流程图;

[0067] 图10是本申请实施例提供的一种初始代码检查结果的显示示意图;

[0068] 图11是本申请实施例提供的一种代码编译流程图;

[0069] 图12是本申请实施例提供的一种代码检查装置的结构示意图;

[0070] 图13是本申请实施例提供的一种终端的结构示意图;

[0071] 图14是本申请实施例提供的一种服务器的结构示意图。

具体实施方式

[0072] 为使本申请的目的、技术方案和优点更加清楚,下面将结合附图对本申请实施方式作进一步地详细描述,显然,所描述的实施例是本申请一部分实施例,而不是全部的实施例。基于本申请中的实施例,本领域普通技术人员在没有做出创造性劳动前提下所获得的所有其他实施例,都属于本申请保护的范围。

[0073] 本申请中术语“第一”“第二”等字样用于对作用和功能基本相同的相同项或相似项进行区分,应理解,“第一”、“第二”、“第n”之间不具有逻辑或时序上的依赖关系,也不对数量和执行顺序进行限定。

[0074] 为了便于理解本申请实施例的技术过程,下面对本申请实施例所涉及的一些名词进行解释:

[0075] Lua:一种编程语言,由标准C语言编写而成,能够在多种操作系统和平台上编译和运行,语法简单、使用灵活、便于热更,目前广泛应用于游戏开发领域。

[0076] VSCode(Visual Studio Code)编辑器:是微软推出的一种跨平台的代码编辑器,能够用于编辑任一语言的代码。VSCode支持插件扩展,VSCode插件安装在VSCode编辑器下,能够提高代码编辑的效率。其中,LuaPanda插件是VSCode编辑器下的一种Lua代码插件,由Typescript语言编写;EmmyLua插件也是VSCode编辑器下一款Lua代码插件,由Java语言编写。

[0077] 实时检查:是指在VSCode编辑中编辑代码过程时,只要一保存代码,就能够立即对代码进行检查,以告警的形式将检查结果提供给用户。

[0078] Go语言:是Google推出一种静态编程语言,它的特点是简洁、高效、安全,且便于实现。

[0079] 图1是本申请实施例提供的一种代码检查方法的实施环境示意图,参见图1,该实施环境包括终端110和服务器140。

[0080] 该终端110可以是智能手机、平板电脑、笔记本电脑、台式计算机、智能音箱、智能手表等,但并不局限于此。终端110安装和运行有支持应用程序开发的代码编辑器,例如,VSCode编辑器等。终端110可以泛指多个终端中的一个,本实施例仅以终端110来举例说明。

[0081] 服务器140可以是独立的物理服务器,也可以是多个物理服务器构成的服务器集群或者分布式系统,还可以是提供云服务、云数据库、云计算、云函数、云存储、网络服务、云通信、中间件服务、域名服务、安全服务、CDN(Content Delivery Network,内容分发网络)、以及大数据和人工智能平台等基础云计算服务的云服务器。服务器140用于为终端110所运行的代码编辑器提供后台服务器。例如,该代码编辑器是VSCode编辑器,在本申请实施例中,VSCode编辑器安装有代码检查插件,该代码检查插件的后端程序部署在服务器140上。可选地,服务器140承担主要的代码检查工作,终端110承担次要的代码检查工作;或者,服务器140承担次要的代码检查工作,终端110承担主要的代码检查工作;或者,服务器140和终端110分别单独承担代码检查工作。

[0082] 上述终端110与服务器140可以通过有线或无线通信方式进行直接或间接地连接,本申请实施例对此不作限定。

[0083] 本领域技术人员可以知晓,上述终端的数量可以更多或更少。比如上述终端可以仅为一个,或者上述终端为几十个或几百个,或者更多数量。本申请实施例对终端的数量和设备类型不加以限定。

[0084] 本申请实施例提供的代码检查方法可以与多种应用场景相结合。例如,在VSCode编辑器下对Lua语言编写的代码文件进行代码检查,该VSCode编辑器中设置有代码检查脚本,在VSCode编辑器中,对代码文件的内容更新可以触发该代码检查脚本运行,对本次更新的代码文件以及与该代码文件相关联的其他代码文件进行检查,将检查结果实时显示在VSCode编辑器的编辑界面中,使用户在代码编辑过程中,不离开当前的编辑环境,即能够实时了解到代码文件中存在的问题,提高代码检查的效率,进而提高应用程序开发效率。需要说明的是,本申请实施例提供的技术方案,还可以应用于其他代码编辑器,对其他编程语言所编写的代码文件进行检查,本申请实施例对此不作限定。

[0085] 图2是本申请实施例提供的一种代码检查方法的流程图。该方法可以应用于上述终端或者服务器,而终端和服务器均可以视为一种计算机设备,在本申请实施例中,以计算机设备作为执行主体,对该代码检查方法进行介绍,参见图2,该实施例具体可以包括以下步骤:

[0086] 201、计算机设备响应于对目标代码文件的检查指令,调用代码编辑器中的代码检查脚本。

[0087] 其中,该代码编辑器用于对目标代码文件进行编辑;该代码编辑器中设置有用对代码文件进行问题检查的代码检查脚本,该代码检查脚本可以以插件的形式运行在该代

码编辑器中。

[0088] 在一种可能实现方式中,在代码编辑器上对目标代码文件进行编辑的过程中,用户对目标代码文件的保存操作,可以触发该检查指令,计算机设备响应于该检查指令,调用该代码检查脚本行,由该代码检查脚本执行后续的代码检查步骤。当然,该代码检查指令也可以由其他方式触发,本申请实施例对此不作限定。

[0089] 202、计算机设备通过该代码检查脚本,从该目标代码文件所属的目标工程包中获取参考代码文件,该参考代码文件为与该目标代码文件之间存在调用关系的代码文件。

[0090] 其中,该目标工程包中有至少一个代码文件,该目标代码文件是该目标工程包中的任一代码文件。该参考代码文件是目标工程包中与该目标代码文件相关联的代码文件,例如,该参考代码文件调用了该目标代码文件中的某一参数,或者该目标代码文件调用了该参考代码文件中的某一参数,本申请实施例对此不作限定。需要说明的是,本申请实施例对获取到的参考代码文件的数目也不作限定。在本申请实施例中,每次目标代码文件更新后,仅获取与该目标代码文件相关联的参考代码文件,执行后续的代码检查步骤,即进行增量检查,而不是全量检查,能够有效缩短代码检查的耗时,提高代码检查效率。

[0091] 203、计算机设备通过该代码检查脚本,基于该参考代码文件中源代码的语法结构,对该目标代码文件中源代码的语法结构进行检查,得到代码检查结果。

[0092] 在一种可能实现方式中,该计算机设备可以通过该代码检查脚本,对该目标代码文件进行词法分析、语法分析得到该目标代码文件中源代码的语法结构,再基于参考代码文件中源代码的语法结构,对该目标代码文件中源代码的语法结构进行检查,例如,对目标代码文件中源代码的语法结构进行语义分析,得到代码检查结果。其中,源代码的语法结构可以由抽象语法树来表示,即对代码文件进行词法分析分析、语法分析分析,得到代码文件对应的抽象语法树。当然,计算机设备也可以先对目标代码文件进行词法分析和语法分析,再获取与目标代码文件之间存在调用关系的参考代码文件,也即是,计算机设备在确定目标代码文件编辑完成之后,先对目标代码文件进行初步检查,再确定目标工程包中各个代码文件之间的调用关系,获取参考代码文件,执行后续的语义分析步骤。需要说明的是本申请实施例对目标代码初步检查的步骤和代码文件调用关系分析的步骤的具体执行顺序不作限定。

[0093] 在一种可能实现方式中,计算机设备通过代码检查脚本进行代码检查的各个环节均可以进行问题定位,即在词法分析、语法分析以及语义分析环节均可以进行问题定位,所获取到的代码检查结果可以包括目标代码文件中的词法错误、语法错误、语义错误等,本申请实施例对此不作限定。

[0094] 204、计算机设备在该代码编辑器的代码编辑界面中,显示该代码检查结果。

[0095] 图3是本申请实施例提供的一种代码编辑界面的示意图,以图3所示的界面为例,计算机设备在该代码编辑界面的编辑区域301中,例如,在错误代码的下方显示参考标记,以对错误代码突出显示,如图3所示该参考标记可以是波浪线,还可以将错误代码显示为参考颜色。其中,该参考标记、参考颜色均可以由开发人员进行设置,本申请实施例对此不作限定。计算机设备也可以在该代码编辑界面显示目标代码文件所包括的错误数目302;还可以显示错误代码目录303,该错误代码目录303中的一条数据可以指示一条错误代码的错误类型、错误原因等,点击该错误代码目录中的任一条数据,可以直接跳转至该任一条数据所

对应错误代码的位置,即在代码编辑区域显示该任一条数据所对应错误代码。

[0096] 本申请实施例提供的技术方案,通过在代码编辑器中设置代码检查脚本,在对目标代码文件的编辑过程中,响应于代码检查指令,触发该代码检查脚本运行,由于目标代码文件的内容更新,仅会对与该目标代码文件相关联的其他代码文件产生影响,因此获取与该目标代码文件之间存在调用关系的参考代码文件,基于获取到的部分参考代码文件中的内容,对目标代码文件进行代码检查,而无需读取目标工程包中的全部代码文件,降低代码检查的耗时,且在代码检查过程中无需将代码文件导出,降低用户操作的复杂度,将检查结果实时显示在编辑界面,使用户在代码编辑过程中可以实时看到代码检查结果,提高代码检查效率。

[0097] 上述实施例是对本申请实施方式的一个简要介绍,图4是本申请实施例提供的一种代码检查方法的流程图,在本申请实施例中,在目标工程包初次在代码编辑器中打开时,可以先对目标工程包中的各个代码文件进行全量检查,在对目标工程包中各个代码文件进行编辑时,进行实时增量检查。以下结合图4,对上述代码检查过程进行具体说明。

[0098] 401、计算机设备响应于目标工程包的打开指令,调用代码编辑器中的代码检查脚本,通过该代码检查脚本创建至少两个子协程。

[0099] 其中,该子协程用于执行代码检查任务。在本申请实施例中,目标工程包中有多个代码文件时,可以应用多协程并发技术,创建多个子协程,通过多个子协程同时对目标工程包中的多个代码文件进行代码检查,以降低代码检查的耗时。

[0100] 在本申请实施例中,用于在代码编辑器中对目标工程包的打开操作,会触发目标工程包的打开指令,计算机设备响应于该打开指令,激活该代码编辑器中的代码检查脚本,其中,该代码检查脚本可以以插件的形式运行在代码编辑器中。在一种可能实现方式中,该代码检查脚本初次运行时,会启动一个主协程,由该主协程获取计算机设备配置的CPU核心数,基于该CPU核心数动态创建多个子协程,例如,所创建子协程的数目与CPU核心数相等。在一种可能实现方式中,以该代码编辑器是VSCode编辑器、该目标工程包应用Lua语言编写为例,该VSCode编辑器中设置有助于对Lua代码进行检查的代码检查插件LuaHelper,当检测到用户VSCode编辑器上打开Lua工程项目,即该目标工程包时,计算机设备激活代码检查插件LuaHelper,触发插件前端程序与插件后端程序lua lsp.exe建立连接,由插件后端程序lua lsp.exe启动一个主协程,再由主协程来动态的创建多个子协程。其中,该LuaHelper插件的实现遵从了Language Server Protocol协议,可以运行在多种环境中,例如Windows、Linux、Mac等,以Windows环境为例,该Windows环境的配置包括CPU为intel5、内存为8G、系统为Win7或Win10。插件前端程序可以基于TypeScript或JS实现,插件后端程序可以基于Go语言实现;该插件前端程序和插件后端程序之间可以建立起管道通信,该管道通信采用JsonRpc协议。图5是本申请实施例提供的一种多协程架构图,如图5所示,主协程501与各个子协程502之间基于管道通信建立联系。需要说明的是,插件前端程序和插件后端程序可以部署在相同的计算机设备上,例如,该计算机设备是用户所使用的终端,插件前端程序和插件后端程序可以均部署在该终端上;可选地,该插件前端程序和插件后端程序部署在不同计算机设备上,例如,插件前端程序部署在用户所使用的终端上,插件后端部署在服务器中,本申请实施例对此不作限定。图6是本申请实施例提供的一种插件运行机制的示意图,如下图6所示,插件前端程序601向插件后端程序602发送请求,例如,查找定义、查找引用、

代码补全等,由插件后端程序返回与请求相对应的内容。

[0101] 需要说明的是,上述对代码检查脚本调用方法和子协程创建方法的说明,仅是一种可能实现方式的示例性说明,本申请实施例对具体采用哪种方法调用代码检查脚本以及创建子协程不作限定。

[0102] 402、计算机设备基于该至少两个子协程的负载情况信息,将目标工程包中的代码文件,分配给该至少两子协程,通过该至少两个子协程,对该目标工程包中的代码文件进行代码检查,得到代码检查结果。

[0103] 在一种可能实现方式中,主协程基于各个子协程的负载情况信息,将该目标工程包中的各个代码文件依次分发给空闲的子协程,由于子协程对代码文件进行代码检查。在本申请实施例中,由多个子协程对目标工程包中的代码文件进行并行处理,可以有效提高代码检查的效率。

[0104] 以任一子协程对目标工程包中的任一代码文件进行代码检查为例进行说明,在一种可能实现方式中,该代码检查过程可以包括以下步骤:

[0105] 步骤一、计算机设备通过任一子协程对该任一代码文件的源代码进行词法分析,得到第一初始结果以及该任一代码文件对应的单词序列。

[0106] 其中,词法分析是指将字符序列转换为单词(Token)序列的过程,也即是将源码字符串流转换为单词流,也称记号流。单词流中的每个单词具有多种属性值,例如,单词的类型、具体值以及位置属性,该单词的类型有标识符、符号、整数、浮点数以及字符串等,该具体值为具体的整数值、浮点数值、字符串值等,该位置属性是指单词的位置信息,具体可以表示为单词的起始行与列,单词的终端行与列。

[0107] 图7是本申请实施例提供的一种词法分析流程图,如图7所示,计算机设备通过该任一子协程,将字符流701,即该任一代码文件中的源代码输入词法分析器702,由词法分析器702将源代码转换为单词序列703。在一种可能实现方式中,该词法分析器是基于手工编码实现法构建的,以这种方法构建的词法分析器灵活度高,字符转换效率高,能够大幅度降低词法分析阶段的耗时,满足实时代码检查的需求。当然,该词法分析器也可以是由词法分析器的生成器得到的,例如该词法分析器的生成器是lex、flex等,本申请实施例对具体采用哪种方法得到词法分析器不作限定。在本申请实施例中,在将源代码转换为单词序列的过程中,计算机设备可以对源代码进行初步问题定位,得到第一初始结合,该第一初始结果用于指示代码文件中的词法错误。

[0108] 步骤二、计算机设备对该任一代码文件对应的单词序列进行语法分析,得到第二初始结果以及该任一代码文件对应的抽象语法树。

[0109] 其中,语法分析是指在词法分析的基础上将单词序列组合成各类语法短语,得到代码文件对应的抽象语法树,该抽象语法树用于指示代码文件中源代码的语法结构。该第二初始结果用于指示代码文件中的语法错误。

[0110] 图8是本申请实施例提供的一种语法分析流程图,如图8所示,计算机设备通过该任一子协程,将该任一代码文件对应的单词序列801输入语法分析器802,由语法分析器802基于编程语言对应的语法规则,生成该任一代码文件对应的抽象语法树803。也即是,在语法分析过程中,需要把词法分析阶段得到的单词序列按照语法规则进行推导,判断单词序列是否满足语法结构,如果不满足语法结构,则不满足语法结构的单词所在的位置需要进

行检查报错,即生成第二初始结果;如果满足语法结构,需要生成对应的抽象语法树。其中,不同编程语言对应于不同的语法规则。以该任一代码文件是由Lua语言编写的为例,计算机设备可以基于上下文无关文法(CFG),将该目标代码文件对应的单词序列构建为抽象语法树。其中,该上下文无关文法可以表示为一个四元组(N、T、P、S),N表示非终结符集合,T表示终结符集合,P表示一组产生式规则,S表示上下文无关文法的唯一开始符号, $S \in N$ 。需要说明的是,上述对语法分析方法的说明,仅是一种可能实现方式的示例性说明,本申请实施例对具体采用哪种方法进行语法分析不作限定。

[0111] 在本申请实施例中,在进行语法分析时,采用自顶向下分析中的递归下降分析算法,一方面,这种分析算法容易实现,便于手工编码;一方面,在语法分析过程中,可以避免回溯,缩短语法分析阶段的耗时;另一方面,能够准确定位出代码文件中的语法错误,得到准确的诊断信息。

[0112] 步骤三、计算机设备对该目标工程包中的代码文件对应的抽象语法树进行语义分析,得到第三初始结果。

[0113] 其中,语义分析也即是上下文相关分析,是对结构上正确的源代码进行上下文有关性质的检查、类型检查等,来判断源代码中是否存在语义错误,即得到第三初始结果,该第三初始结果用于指示代码文件中的语义错误。

[0114] 在本申请实施例中,计算机设备可以基于语法分析阶段得到的抽象语法树,来对各个代码文件进行语义分析,例如,计算机设备可以深度遍历抽象语法树,来模拟程序运行时的行为,从而检查代码文件中的语义错误,而不必真实的运行程序。以对Lua代码进行语义分析为例,在对Lua代码进行检查时,可以包括以下代码检查类型:1)代码文件中的变量在使用前是否有定义;2)代码文件中函数调用的参数列表个数与函数定义的是否一致;3)函数定义的参数名是否重复;4)已定义的局部变量是否有被使用到;5)列表(table)构造时是否包含重复的关键词(key);6)引用其他Lua代码文件是否能够找到被引用的Lua代码文件。以检查变量在使用前是否有定义为例,计算机设备需对各个代码文件对应的抽象语法树进行两次遍历,在第一遍历过程中,生成变量的全局符号列表,该全局符号列表包括在目标工程包中已定义的变量;在第二次遍历过程中,基于该全局符号列表,来查找各个变量在使用前是否有定义。在一种可能实现方式中,查找顺序为先查找当前作用域的local变量;然后,查找当前作用域的upvalue变量;最后,查找第一遍中的全局符号表。需要说明的是,上述对语义分析过程的说明,仅是一种可能实现方式的示例性说明,本申请实施例对具体采用哪种方法进行语义分析不作限定。

[0115] 图9是本申请实施例提供的一种代码检查流程图,以图9为例,首先计算机设备通过代码检查脚本对源代码901,进行词法分析902,将字符流转换为单词流,再基于单词流进行语法分析903,得到抽象语法树,基于抽象语法树进行语义分析904,得到中间代码。在本申请实施例中,在目标工程包初始运行时就立即触发代码检查脚本的运行,对目标工程包进行全量检查,确定出目标工程包存在的词法错误、语法错误以及语义错误,便于开发人员进行问题定位,提高开发效率。

[0116] 步骤四、计算机设备将该第一初始结果、该第二初始结果以及该第三初始结果,确定为该目标工程包对应的初始代码检查结果。

[0117] 在一种可能实现方式中,子协程可以将词法分析、语法分析以及语义分析阶段得

到的检查结果,即第一初始结果、第二初始结果以及第三初始结果,发送给主协程,由主协程基于各个阶段的检查结果得到目标工程包定位的初始代码检查结果。

[0118] 在一种可能实现方式中,主协程还可以将该初始代码检查结果以及该目标工程包中各个代码文件对应的抽象语法树,存储至目标存储空间。

[0119] 需要说明的是,上述步骤401和步骤402,是响应于对该目标工程包的打开指令,调用该代码编辑器中的该代码检查脚本对该目标工程包中的代码文件进行代码检查,得到该目标工程包对应的初始代码检查结果的步骤。在本申请实施例中,在目标工程包初次打开时,就调用代码编辑器中的代码检查脚本,扫描目标工程包中的全部代码文件,对目标工程包进行全量检查,一方面,能够全面获取到目标工程包中存在的代码错误,便于开发人员进行问题定位,提高代码检查效率和开发效率;另一方面,采用多协程并发技术,由多个子协程对代码文件进行并行处理,能够有效缩短代码检查的耗时,提高代码检查效率。

[0120] 403、计算机设备在代码编辑界面中显示该目标工程包对应的初始代码检查结果。

[0121] 在一种可能实现方式中,以该代码检查脚本是LuaHelper插件为例,插件后端程序获取到初始代码检查结果后,可以将初始代码检查结果推送至插件前端程序进行显示。例如,插件后端程序向插件前端程序发送publishDiagnostics请求,该publishDiagnostics请求可以携带有初始代码检查结果。

[0122] 在一种可能实现方式中,计算机设备还可以获取对该目标工程包进行代码检查的耗时,将该目标工程包进行代码检查的耗时以及该初始代码检查结果,在该代码编辑界面进行显示。图10是本申请实施例提供的一种初始代码检查结果的显示示意图,参见图10,在一种可能实现方式中,计算机设备基于该初始代码检查结果,确定该目标工程包中错误代码的位置,在代码编辑界面对该目标代码文件中的错误源代码进行突出显示,如1001区域所示,在错误代码下方显示波浪线;基于该初始代码检查结果,确定目标工程包中的错误数目,在该代码编辑界面显示该目标工程包对应的错误数目,如图中1002所示;基于该初始代码检查结果,在该代码编辑界面显示该目标工程包对应的错误代码目录,如图中1003所示,其中,该错误代码目录中的一条数据用于表示一处源代码的出错信息;在代码编辑界面显示本次代码检查的耗时,如图中1004所示。需要说明的是,上述对初始代码检查结果显示方式的说明,仅是一种示例性说明,本申请实施例对具体采用哪种方法对初始代码检查结果进行显示不作限定。

[0123] 上述步骤401至步骤403,是对目标工程包进行全量检查的过程,在全量检查完目标工程包的各个代码文件之后,在代码编辑界面进行错误提示,针对检查错误提示的地方,用户进行修改保存后,上述图10中1001、1002以及1003区域所示的内容会实时联动变化,显示最新的错误信息。

[0124] 404、计算机设备响应于对目标代码文件代码的检查指令,调用该代码编辑器中的代码检查脚本。

[0125] 其中,该目标代码文件是目标工程包中的任一代码文件。

[0126] 在本申请实施例中,在对目标代码文件的编辑过程中,可以对该目标代码文件进行实时代码检查,也即是,在对目标代码文件的一次编辑完成,可以触发对目标代码文件的检查指令,计算机设备响应于该检查指令进行增量检查。在一种可能实现方式中,用于在代码编辑器中对目标代码文件的保存操作,可以触发该检查指令,计算机设备响应于该检查

指令,调用该代码检查脚本。以该代码检查脚本实现为插件的形式为例,检测到对代码编辑器中目标代码文件的保存操作时,触发检查指令,调用代码编辑器中的插件前端程序,由插件前端程序向插件后端程序发送didChange请求,该didChange请求用于指示该目标代码文件的内容已修改,插件后端程序基于该请求执行后续的代码检查步骤。需要说明的是,该插件前端程序在不同场景下可以向插件后端程序发送不同请求,例如,在变量查找场景中,可以向插件后端程序发送definition请求,插件后端程序响应于该请求会解析变量对应的文件位置信息,并把结果发送给插件前端程序,插件前端程序再自动跳转到相应的位置。需要说明的是,本申请对代码检查脚本的应用场景不作限定,在本申请实施例中,仅以应用在代码检查场景中为例进行说明。

[0127] 在一种可能实现方式中,该代码编辑器中还可以设置有代码检查控件,该代码检查控件用于触发代码检查脚本对代码文件进行代码检查。例如,在目标代码文件编辑过程中,用户对代码检查控件的触发操作,可以触发该检查指令,计算机设备响应于该检查指令,调用该代码检查脚本,对目标代码文件进行代码检查。

[0128] 需要说明的是,上述对代码检查脚本调用方法的说明,仅是一种可能实现方式的示例性说明,本申请实施例对具体采用哪种方法触发代码检查脚本的调用不作限定。

[0129] 405、计算机设备通过该代码检查脚本,从目标代码文件所属的目标工程包中获取参考代码文件。

[0130] 其中,该参考代码文件是与该目标代码文件之间存在调用关系的代码文件。

[0131] 在本申请实施例中,由于对目标代码文件的内容编辑,会影响到与该目标代码文件相关联的其他代码文件,也即是,目标工程包中与该目标代码文件之间存在调用关系的代码文件。在本申请实施例中,需获取与该目标代码文件之间存在调用关系的参考代码文件,基于该目标代码文件与参考代码文件执行后续的代码检查步骤。需要说明的是,本申请实施例中,对获取到的参考代码文件的数目不作限定。若该目标工程包中没有与目标代码文件之间具有调用关系的代码文件,或者该目标工程包中仅包括该目标代码文件,则计算机设备可以基于该目标代码文件执行后续的代码检查步骤。

[0132] 406、计算机设备通过该代码检查脚本获取该目标代码文件对应的抽象语法树。

[0133] 在一种可能实现方式中,计算机设备通过该代码检查脚本先对该目标代码文件的源代码进行词法分析,得到该目标代码文件对应的单词序列;再对该目标代码文件对应的单词序列进行语法分析,得到该目标代码文件对应的抽象语法树。当然,还可以获取到词法分析阶段、语法分析阶段的检查结果,即得到该目标代码文件中的词法错误信息、语法错误信息。需要说明的是,该词法分析、语法分析的步骤与上述步骤402中词法分析、语法分析的过程同理,在此不做赘述。

[0134] 在本申请实施例中,该计算机设备可以将该词法错误信息、语法错误信息缓存至该目标存储空间,还可以将该目标存储空间中已存储的该目标代码文件的抽象语法树,替换为本次获取到的该目标代码文件对应的抽象语法树,以确保在后续对其他代码文件进行增量检查时,可以获取到准确的抽象语法树。

[0135] 需要说明的是,在本申请实施例中,以先执行参考代码文件确定的步骤405,再执行获取目标代码文件的抽象语法树的步骤406的顺序进行描述,在一些实施例中,也可以先执行步骤406,再执行步骤405,或者两个步骤同时执行,本申请实施例对此不作限定。

[0136] 407、计算机设备获取该参考代码文件对应的抽象语法树。

[0137] 在本申请实施例中,计算机设备可以从已缓存的抽象语法树中,获取参考代码文件对应的抽象语法树。例如,目标存储空间中存储有各个代码文件对应的抽象语法树,各个抽象语法树与各个代码文件的文件标识相关联存储。在一种可能实现方式中,计算机设备基于该参考代码文件的文件标识,从目标存储空间中获取该参考代码文件对应的抽象语法树,该目标存储空间用于存储该目标工程包中各个代码文件对应的抽象语法树。需要说明的是,上述对获取参考代码文件的抽象语法树的说明,仅是一种可能实现方式的示例性说明,本申请实施例对具体采用哪种方法获取参考代码文件的抽象语法树不作限定。

[0138] 需要说明的是,在本申请实施例中,以先执行获取目标代码文件对应的抽象语法树的步骤,再执行获取参考代码文件对应的抽象语法树的步骤进行描述,在一些实施例中,也可以先执行获取参考代码文件对应的抽象语法树的步骤,再执行获取目标代码文件对应的抽象语法树的步骤,或者两个步骤同时执行,本申请实施例对此不作限定。

[0139] 408、计算机设备通过该代码检查脚本,基于该参考代码文件对应的抽象语法树,对该目标代码文件对应的抽象语法树进行检查,得到代码检查结果。

[0140] 其中,该代码检查结果包括该目标代码文件中的词法错误信息、该目标代码文件中的语法错误信息以及该目标代码文件中的语义错误信息。

[0141] 在本申请实施例中,计算机设备可以基于目标代码文件和参考代码文件对应的抽象语法树,进行语义检查,得到该目标代码文件中的语义错误信息。例如,计算机设备通过该代码检查脚本,遍历该目标代码文件以及该参考代码文件对应的抽象语法树,生成符号列表,该符号列表用于指示该目标代码文件以及该参考代码文件所包括的变量;基于该符号列表、该目标代码文件和该参考代码文件对应的抽象语法树,进行代码检查。需要说明的是,该语义检查的过程与上述步骤402中语义检查的过程同理,在此不做赘述。

[0142] 需要说明的是,上述步骤406至步骤408,是通过该代码检查脚本,基于该参考代码文件中源代码的语法结构,对该目标代码文件中源代码的语法结构进行检查,得到代码检查结果的步骤。在本申请实施例中,对编辑完成的目标代码文件进行实时增量检查,在实时增量检查的过程中,仅对与该目标代码文件相关联的部分参考代码文件进行检查,能够有效降低代码检查过程的运算量,提高代码检查效率;且从已缓存的抽象语法树中,获取参考代码文件对应的抽象语法树,无需重新构建,能够避免重复构建抽象语法树,避免重复运算,缩短代码检查的耗时,提高代码检查效率。

[0143] 409、计算机设备在该代码编辑器的代码编辑界面中,显示该代码检查结果。

[0144] 在一种可能实现方式中,计算机设备可以基于该代码检查结果,在该代码编辑界面对该目标代码文件中的错误源代码进行突出显示,例如,在错误代码下方显示参考标记,将错误代码显示为参考颜色等,其中,该参考标记、参考颜色均可以由开发人员进行设置,本申请实施例对此不作限定。计算机设备还可以基于该代码检查结果,在该代码编辑界面显示该目标代码文件对应的错误数目;基于该代码检查结果,在该代码编辑界面显示该目标代码文件对应的错误代码目录,该错误代码目录中的一条数据用于表示一处源代码的出错信息。需要说明的是,上述对代码检查结果显示方法的说明,仅是一种示例性说明,本申请实施例对具体采用哪种方法对代码检查结果进行显示不作限定。

[0145] 本申请实施例提供的技术方案,通过在代码编辑器中设置代码检查脚本,在对目

标代码文件的编辑过程中,响应于代码检查指令,触发该代码检查脚本运行,由于目标代码文件的内容更新,仅会对与该目标代码文件相关联的其他代码文件产生影响,因此获取与该目标代码文件之间存在调用关系的参考代码文件,基于获取到的部分参考代码文件中的内容,对目标代码文件进行代码检查,而无需读取目标工程包中的全部代码文件,降低代码检查的耗时,且在代码检查过程中无需将代码文件导出,降低用户操作的复杂度,将检查结果实时显示在编辑界面,使用户在代码编辑过程中可以实时看到代码检查结果,提高代码检查效率。

[0146] 图11是本申请实施例提供的一种代码编译流程图,结合图11,对上述代码检查过程进行说明,在代码编译过程中,计算机设备将字符流形式的源代码。进行词法分析1101,将源代码转换为单词流;对单词流进行语法分析1102,得到抽象语法树;再基于抽象语法树进行语义分析1103,得到中间代码。上述通过词法分析、语法分析以及语义分析得到中间代码的过程,即代码编译前端的过程,在这一过程中,计算机设备通过代码检查插件的后端程序对源代码进行错误检查,验证代码是否满足规范性、安全性、可靠性、可维护性。计算机设备再对获取到的中间代码进行代码优化以及代码生成,实现代码编译后端的过程。在本申请实施例中,代码检查插件的后端程序是采用Go语言实现的,Go语言的性能接近C、C++编程语言,在词法分析阶段,采用高效的手工编码的实现方式;在语法分析阶段,采用的递归是下降分析方法,在这种方式中,采用线性搜索时间,能够减少不必要的回溯,提高语法分析效率;在语义分析阶段,优化了中间数据表示结构,提高了性能。而且,在本申请实施例中,应用代码检查插件,以增量检查的方式,对代码编辑其中完成编辑的代码文件进行实时检查,增量检查比全量检查的耗时大幅降低,在100毫秒左右即可完成一次增量检查,极大的提升了开发人员的开发效率,将检查出的问题实时显示在代码编辑界面,使开发人员可以及时了解开发过程中的问题,及时修复,确保了线上服务质量。

[0147] 上述所有可选技术方案,可以采用任意结合形成本申请的可选实施例,在此不再一一赘述。

[0148] 图12是本申请实施例提供的一种代码检查装置的结构示意图,参见图12,该装置包括:

[0149] 调用模块1201,用于响应于对目标代码文件代码的检查指令,调用代码编辑器中的代码检查脚本;

[0150] 获取模块1202,用于通过该代码检查脚本,从该目标代码文件所属的目标工程包中获取参考代码文件,该参考代码文件为与该目标代码文件之间存在调用关系的代码文件;

[0151] 检查模块1203,用于通过该代码检查脚本,基于该参考代码文件中源代码的语法结构,对该目标代码文件中源代码的语法结构进行检查,得到代码检查结果;

[0152] 显示模块1204,用于在该代码编辑器的代码编辑界面中,显示该代码检查结果。

[0153] 在一种可能实现方式中,该检查模块1203包括:

[0154] 第一获取子模块,用于获取该目标代码文件对应的抽象语法树,该抽象语法树用于指示代码文件的源代码的语法结构;

[0155] 第二获取子模块,用于获取该参考代码文件对应的抽象语法树;

[0156] 检查子模块,用于通过该代码检查脚本,基于该参考代码文件对应的抽象语法树,

对该目标代码文件对应的抽象语法树进行检查,得到该代码检查结果。

[0157] 在一种可能实现方式中,该第一获取子模块,用于:

[0158] 对该目标代码文件的源代码进行词法分析,得到该目标代码文件对应的单词序列;

[0159] 对该目标代码文件对应的单词序列进行语法分析,得到该目标代码文件对应的抽象语法树。

[0160] 在一种可能实现方式中,该第二获取子模块,用于:

[0161] 基于该参考代码文件的文件标识,从目标存储空间中获取该参考代码文件对应的抽象语法树,该目标存储空间用于存储该目标工程包中各个代码文件对应的抽象语法树。

[0162] 在一种可能实现方式中,该装置还包括:

[0163] 存储模块,用于将该目标存储空间中已存储的该目标代码文件的抽象语法树,替换为本次获取到的该目标代码文件对应的抽象语法树。

[0164] 在一种可能实现方式中,该检查子模块,用于:

[0165] 通过该代码检查脚本,遍历该目标代码文件以及该参考代码文件对应的抽象语法树,生成符号列表,该符号列表用于指示该目标代码文件以及该参考代码文件所包括的变量;

[0166] 基于该符号列表以及该参考代码文件对应的抽象语法树,对该目标代码文件对应的抽象语法树进行检查,得到该代码检查结果。

[0167] 在一种可能实现方式中,该代码检查结果包括该目标代码文件中的词法错误信息、该目标代码文件中的语法错误信息以及该目标代码文件中的语义错误信息。

[0168] 在一种可能实现方式中,该显示模块1204,用于执行下述至少一项:

[0169] 基于该代码检查结果,在该代码编辑界面对该目标代码文件中的错误源代码进行突出显示;

[0170] 基于该代码检查结果,在该代码编辑界面显示该目标代码文件对应的错误数目;

[0171] 基于该代码检查结果,在该代码编辑界面显示该目标代码文件对应的错误代码目录,该错误代码目录中的一条数据用于表示一处源代码的错误信息。

[0172] 在一种可能实现方式中,该检查模块1203用于响应于对该目标工程包的打开指令,调用该代码编辑器中的该代码检查脚本对该目标工程包中的代码文件进行代码检查,得到该目标工程包对应的初始代码检查结果;

[0173] 该显示模块1204,用于在该代码编辑界面中,显示该目标工程包对应的初始代码检查结果。

[0174] 在一种可能实现方式中,该检查模块1203包括:

[0175] 调用子模块,用于响应于对该目标工程包的打开指令,调用该代码检查脚本;

[0176] 创建子模块,用于创建通过该代码检查脚本创建至少两个子协程,该子协程用于执行代码检查任务;

[0177] 分配子模块,用于基于该至少两个子协程的负载情况信息,将该目标工程中的代码文件,分配给该至少两子协程;

[0178] 该检查子模块,用于通过该至少两个子协程,对该目标工程包中的代码文件进行代码检查,得到代码检查结果。

[0179] 在一种可能实现方式中,该检查子模块,用于:

[0180] 在任一子协程对任一代码文件进行代码检查的过程中,通过该任一子协程对该任一代码文件的源代码进行词法分析,得到第一初始结果以及该任一代码文件对应的单词序列,该第一初始结果用于指示代码文件中的词法错误;

[0181] 对该任一代码文件对应的单词序列进行语法分析,得到第二初始结果以及该任一代码文件对应的抽象语法树,该第二初始结果用于指示代码文件中的语法错误;

[0182] 对该目标工程包中的代码文件对应的抽象语法树进行语义分析,得到第三初始结果,该第三初始结果用于指示代码文件中的语义错误;

[0183] 将该第一初始结果、该第二初始结果以及该第三初始结果,确定为该目标工程包对应的初始代码检查结果。

[0184] 在一种可能实现方式中,该存储模块用于:

[0185] 将该初始代码检查结果以及该代码文件对应的抽象语法树,存储至目标存储空间。

[0186] 在一种可能实现方式中,该装置还包括:

[0187] 耗时获取模块,用于获取对该目标工程包进行代码检查的耗时;

[0188] 该显示模块1204,用于在该代码编辑界面中,显示对该目标工程包进行代码检查的耗时。

[0189] 本申请实施例提供的装置,通过在代码编辑器中设置代码检查脚本,在对目标代码文件的编辑过程中,响应于代码检查指令,触发该代码检查脚本运行,由于目标代码文件的内容更新,仅会对与该目标代码文件相关联的其他代码文件产生影响,因此获取与该目标代码文件之间存在调用关系的参考代码文件,基于获取到的部分参考代码文件中的内容,对目标代码文件进行代码检查,而无需读取目标工程包中的全部代码文件,降低代码检查的耗时,且在代码检查过程中无需将代码文件导出,降低用户操作的复杂度,将检查结果实时显示在编辑界面,使用户在代码编辑过程中可以实时看到代码检查结果,提高代码检查效率。

[0190] 需要说明的是:上述实施例提供的代码检查装置在代码检查时,仅以上述各功能模块的划分进行举例说明,实际应用中,可以根据需要而将上述功能分配由不同的功能模块完成,即将装置的内部结构划分成不同的功能模块,以完成以上描述的全部或者部分功能。另外,上述实施例提供的代码检查装置与代码检查方法实施例属于同一构思,其具体实现过程详见方法实施例,这里不再赘述。

[0191] 上述技术方案所提供的计算机设备可以实现为终端或服务器,例如,图13是本申请实施例提供的一种终端的结构示意图。该终端1300可以是:智能手机、平板电脑、MP3播放器(Moving Picture Experts Group Audio Layer III,动态影像专家压缩标准音频层面3)、MP4(Moving Picture Experts Group Audio Layer IV,动态影像专家压缩标准音频层面4)播放器、笔记本电脑或台式电脑。终端1300还可能被称为用户设备、便携式终端、膝上型终端、台式终端等其他名称。

[0192] 通常,终端1300包括有:一个或多个处理器1301和一个或多个存储器1302。

[0193] 处理器1301可以包括一个或多个处理核心,比如4核心处理器、8核心处理器等。处理器1301可以采用DSP(Digital Signal Processing,数字信号处理)、FPGA(Field-

Programmable Gate Array,现场可编程门阵列)、PLA(Programmable Logic Array,可编程逻辑阵列)中的至少一种硬件形式来实现。处理器1301也可以包括主处理器和协处理器,主处理器是用于对在唤醒状态下的数据进行处理的处理单元,也称CPU(Central Processing Unit,中央处理器);协处理器是用于对在待机状态下的数据进行处理的低功耗处理器。在一些实施例中,处理器1301可以在集成有GPU(Graphics Processing Unit,图像处理器),GPU用于负责显示屏所需要显示的内容的渲染和绘制。一些实施例中,处理器1301还可以包括AI(Artificial Intelligence,人工智能)处理器,该AI处理器用于处理有关机器学习的计算操作。

[0194] 存储器1302可以包括一个或多个计算机可读存储介质,该计算机可读存储介质可以是非暂态的。存储器1302还可包括高速随机存取存储器,以及非易失性存储器,比如一个或多个磁盘存储设备、闪存存储设备。在一些实施例中,存储器1302中的非暂态的计算机可读存储介质用于存储至少一条程序代码,该至少一条程序代码用于被处理器1301所执行以实现本申请中方法实施例提供的代码检查方法。

[0195] 在一些实施例中,终端1300还可选包括有:外围设备接口1303和至少一个外围设备。处理器1301、存储器1302和外围设备接口1303之间可以通过总线或信号线相连。各个外围设备可以通过总线、信号线或电路板与外围设备接口1303相连。具体地,外围设备包括:射频电路1304、显示屏1305、摄像头组件1306、音频电路1307、定位组件1308和电源1309中的至少一种。

[0196] 外围设备接口1303可被用于将I/O(Input/Output,输入/输出)相关的至少一个外围设备连接到处理器1301和存储器1302。在一些实施例中,处理器1301、存储器1302和外围设备接口1303被集成在同一芯片或电路板上;在一些其他实施例中,处理器1301、存储器1302和外围设备接口1303中的任意一个或两个可以在单独的芯片或电路板上实现,本实施例对此不加以限定。

[0197] 射频电路1304用于接收和发射RF(Radio Frequency,射频)信号,也称电磁信号。射频电路1304通过电磁信号与通信网络以及其他通信设备进行通信。射频电路1304将电信号转换为电磁信号进行发送,或者,将接收到的电磁信号转换为电信号。可选地,射频电路1304包括:天线系统、RF收发器、一个或多个放大器、调谐器、振荡器、数字信号处理器、编解码芯片组、用户身份模块卡等等。射频电路1304可以通过至少一种无线通信协议来与其它终端进行通信。该无线通信协议包括但不限于:城域网、各代移动通信网络(2G、3G、4G及5G)、无线局域网和/或WiFi(Wireless Fidelity,无线保真)网络。在一些实施例中,射频电路1304还可以包括NFC(Near Field Communication,近距离无线通信)有关的电路,本申请对此不加以限定。

[0198] 显示屏1305用于显示UI(User Interface,用户界面)。该UI可以包括图形、文本、图标、视频及其它们的任意组合。当显示屏1305是触摸显示屏时,显示屏1305还具有采集在显示屏1305的表面或表面上方的触摸信号的能力。该触摸信号可以作为控制信号输入至处理器1301进行处理。此时,显示屏1305还可以用于提供虚拟按钮和/或虚拟键盘,也称软按钮和/或软键盘。在一些实施例中,显示屏1305可以为一个,设置终端1300的前面板;在另一些实施例中,显示屏1305可以为至少两个,分别设置在终端1300的不同表面或呈折叠设计;在一些实施例中,显示屏1305可以是柔性显示屏,设置在终端1300的弯曲表面上或折叠面

上。甚至,显示屏1305还可以设置成非矩形的不规则图形,也即异形屏。显示屏1305可以采用LCD(Liquid Crystal Display,液晶显示屏)、OLED(Organic Light-Emitting Diode,有机发光二极管)等材质制备。

[0199] 摄像头组件1306用于采集图像或视频。可选地,摄像头组件1306包括前置摄像头和后置摄像头。通常,前置摄像头设置在终端的前面板,后置摄像头设置在终端的背面。在一些实施例中,后置摄像头为至少两个,分别为主摄像头、景深摄像头、广角摄像头、长焦摄像头中的任意一种,以实现主摄像头和景深摄像头融合实现背景虚化功能、主摄像头和广角摄像头融合实现全景拍摄以及VR(Virtual Reality,虚拟现实)拍摄功能或者其它融合拍摄功能。在一些实施例中,摄像头组件1306还可以包括闪光灯。闪光灯可以是单色温闪光灯,也可以是双色温闪光灯。双色温闪光灯是指暖光闪光灯和冷光闪光灯的组合,可以用于不同色温下的光线补偿。

[0200] 音频电路1307可以包括麦克风和扬声器。麦克风用于采集用户及环境的声波,并将声波转换为电信号输入至处理器1301进行处理,或者输入至射频电路1304以实现语音通信。出于立体声采集或降噪的目的,麦克风可以为多个,分别设置在终端1300的不同部位。麦克风还可以是阵列麦克风或全向采集型麦克风。扬声器则用于将来自处理器1301或射频电路1304的电信号转换为声波。扬声器可以是传统的薄膜扬声器,也可以是压电陶瓷扬声器。当扬声器是压电陶瓷扬声器时,不仅可以将电信号转换为人类可听见的声波,也可以将电信号转换为人类听不见的声波以进行测距等用途。在一些实施例中,音频电路1307还可以包括耳机插孔。

[0201] 定位组件1308用于定位终端1300的当前地理位置,以实现导航或LBS(Location Based Service,基于位置的服务)。定位组件1308可以是基于美国的GPS(Global Positioning System,全球定位系统)、中国的北斗系统、俄罗斯的格雷纳斯系统或欧盟的伽利略系统的定位组件。

[0202] 电源1309用于为终端1300中的各个组件进行供电。电源1309可以是交流电、直流电、一次性电池或可充电电池。当电源1309包括可充电电池时,该可充电电池可以支持有线充电或无线充电。该可充电电池还可以用于支持快充技术。

[0203] 在一些实施例中,终端1300还包括有一个或多个传感器1310。该一个或多个传感器1310包括但不限于:加速度传感器1311、陀螺仪传感器1312、压力传感器1313、指纹传感器1314、光学传感器1315以及接近传感器1316。

[0204] 加速度传感器1311可以检测以终端1300建立的坐标系的三个坐标轴上的加速度大小。比如,加速度传感器1311可以用于检测重力加速度在三个坐标轴上的分量。处理器1301可以根据加速度传感器1311采集的重力加速度信号,控制显示屏1305以横向视图或纵向视图进行用户界面的显示。加速度传感器1311还可以用于游戏或者用户的运动数据的采集。

[0205] 陀螺仪传感器1312可以检测终端1300的机体方向及转动角度,陀螺仪传感器1312可以与加速度传感器1311协同采集用户对终端1300的3D动作。处理器1301根据陀螺仪传感器1312采集的数据,可以实现如下功能:动作感应(比如根据用户的倾斜操作来改变UI)、拍摄时的图像稳定、游戏控制以及惯性导航。

[0206] 压力传感器1313可以设置在终端1300的侧边框和/或显示屏1305的下层。当压力

传感器1313设置在终端1300的侧边框时,可以检测用户对终端1300的握持信号,由处理器1301根据压力传感器1313采集的握持信号进行左右手识别或快捷操作。当压力传感器1313设置在显示屏1305的下层时,由处理器1301根据用户对显示屏1305的压力操作,实现对UI界面上的可操作性控件进行控制。可操作性控件包括按钮控件、滚动条控件、图标控件、菜单控件中的至少一种。

[0207] 指纹传感器1314用于采集用户的指纹,由处理器1301根据指纹传感器1314采集到的指纹识别用户的身份,或者,由指纹传感器1314根据采集到的指纹识别用户的身份。在识别出用户的身份为可信身份时,由处理器1301授权该用户执行相关的敏感操作,该敏感操作包括解锁屏幕、查看加密信息、下载软件、支付及更改设置等。指纹传感器1314可以被设置终端1300的正面、背面或侧面。当终端1300上设置有物理按键或厂商Logo时,指纹传感器1314可以与物理按键或厂商Logo集成在一起。

[0208] 光学传感器1315用于采集环境光强度。在一个实施例中,处理器1301可以根据光学传感器1315采集的环境光强度,控制显示屏1305的显示亮度。具体地,当环境光强度较高时,调高显示屏1305的显示亮度;当环境光强度较低时,调低显示屏1305的显示亮度。在另一个实施例中,处理器1301还可以根据光学传感器1315采集的环境光强度,动态调整摄像头组件1306的拍摄参数。

[0209] 接近传感器1316,也称距离传感器,通常设置在终端1300的前面板。接近传感器1316用于采集用户与终端1300的正面之间的距离。在一个实施例中,当接近传感器1316检测到用户与终端1300的正面之间的距离逐渐变小时,由处理器1301控制显示屏1305从亮屏状态切换为息屏状态;当接近传感器1316检测到用户与终端1300的正面之间的距离逐渐变大时,由处理器1301控制显示屏1305从息屏状态切换为亮屏状态。

[0210] 本领域技术人员可以理解,图13中示出的结构并不构成对终端1300的限定,可以包括比图示更多或更少的组件,或者组合某些组件,或者采用不同的组件布置。

[0211] 图14是本申请实施例提供的一种服务器的结构示意图,该服务器1400可因配置或性能不同而产生比较大的差异,可以包括一个或多个处理器(Central Processing Units, CPU) 1401和一个或多个的存储器1402,其中,该一个或多个存储器1402中存储有至少一条程序代码,该至少一条程序代码由该一个或多个处理器1401加载并执行以实现上述各个方法实施例提供的方法。当然,该服务器1400还可以具有有线或无线网络接口、键盘以及输入输出接口等部件,以便进行输入输出,该服务器1400还可以包括其他用于实现设备功能的部件,在此不做赘述。

[0212] 在示例性实施例中,还提供了一种计算机可读存储介质,例如包括至少一条程序代码的存储器,上述至少一条程序代码可由处理器执行以完成上述实施例中的代码检查方法。例如,该计算机可读存储介质可以是只读存储器(Read-Only Memory, ROM)、随机存取存储器(Random Access Memory, RAM)、只读光盘(Compact Disc Read-Only Memory, CD-ROM)、磁带、软盘和光数据存储设备等。

[0213] 在示例性实施例中,还提供了一种计算机程序产品,该计算机程序产品包括至少一条程序代码,该至少一条程序代码存储在计算机可读存储介质中。计算机设备的处理器从计算机可读存储介质读取该至少一条程序代码,处理器执行该至少一条程序代码,使得该计算机设备实现该代码检查方法所执行的操作。

[0214] 本领域普通技术人员可以理解实现上述实施例的全部或部分步骤可以通过硬件来完成,也可以通过程序来至少一条程序代码相关的硬件完成,该程序可以存储于一种计算机可读存储介质中,上述提到的存储介质可以是只读存储器,磁盘或光盘等。

[0215] 上述仅为本申请的可选实施例,并不用以限制本申请,凡在本申请的精神和原则之内,所作的任何修改、等同替换、改进等,均应包含在本申请的保护范围之内。

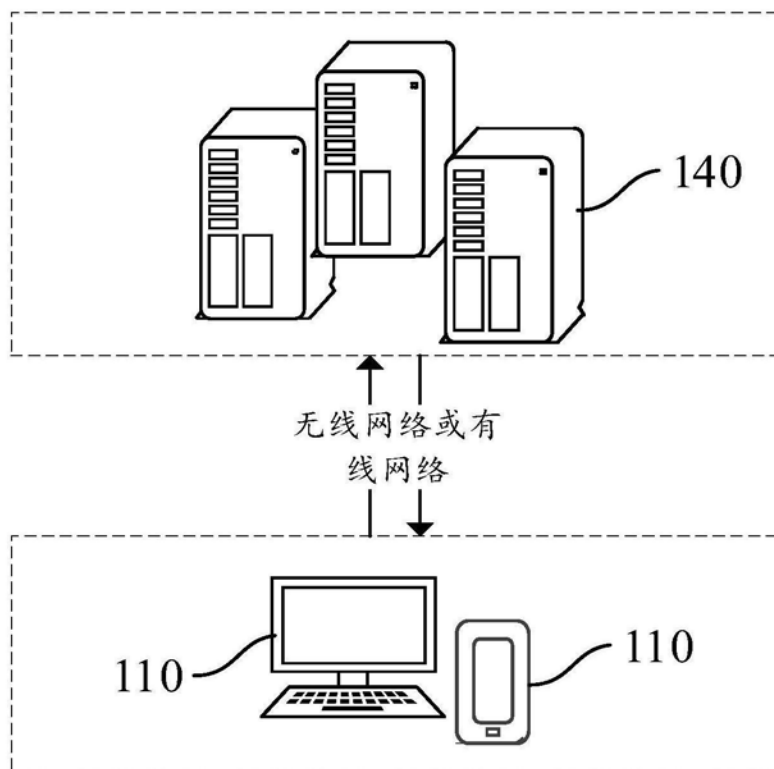


图1

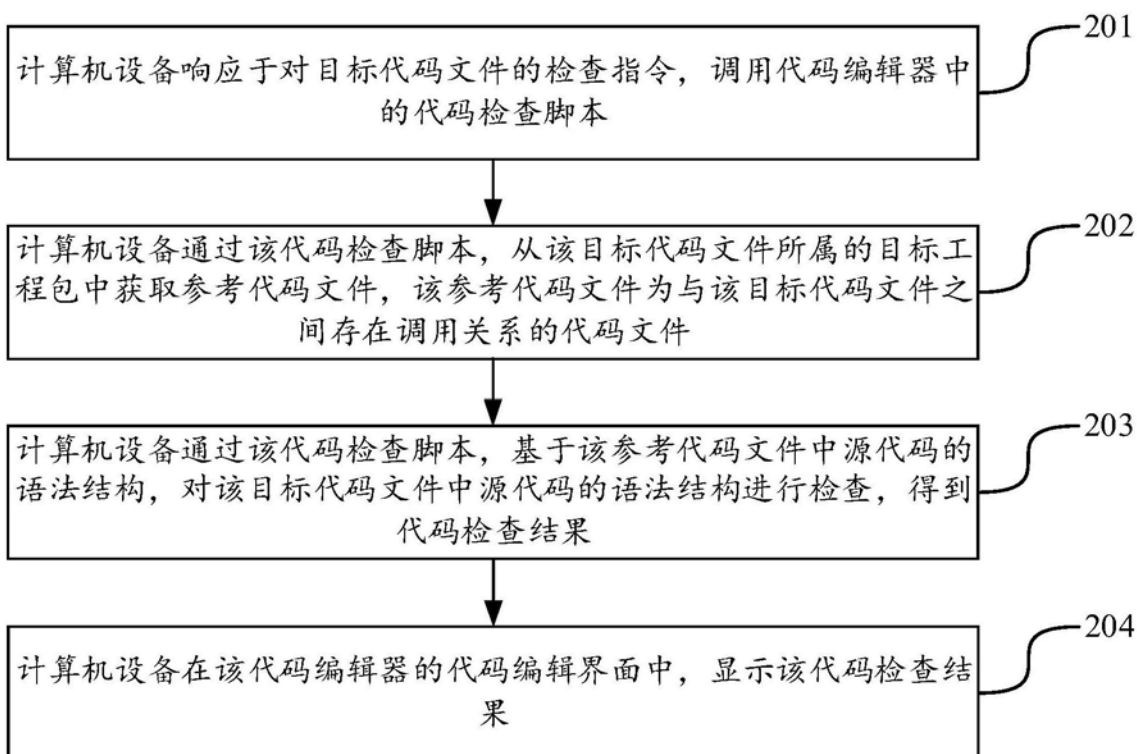


图2

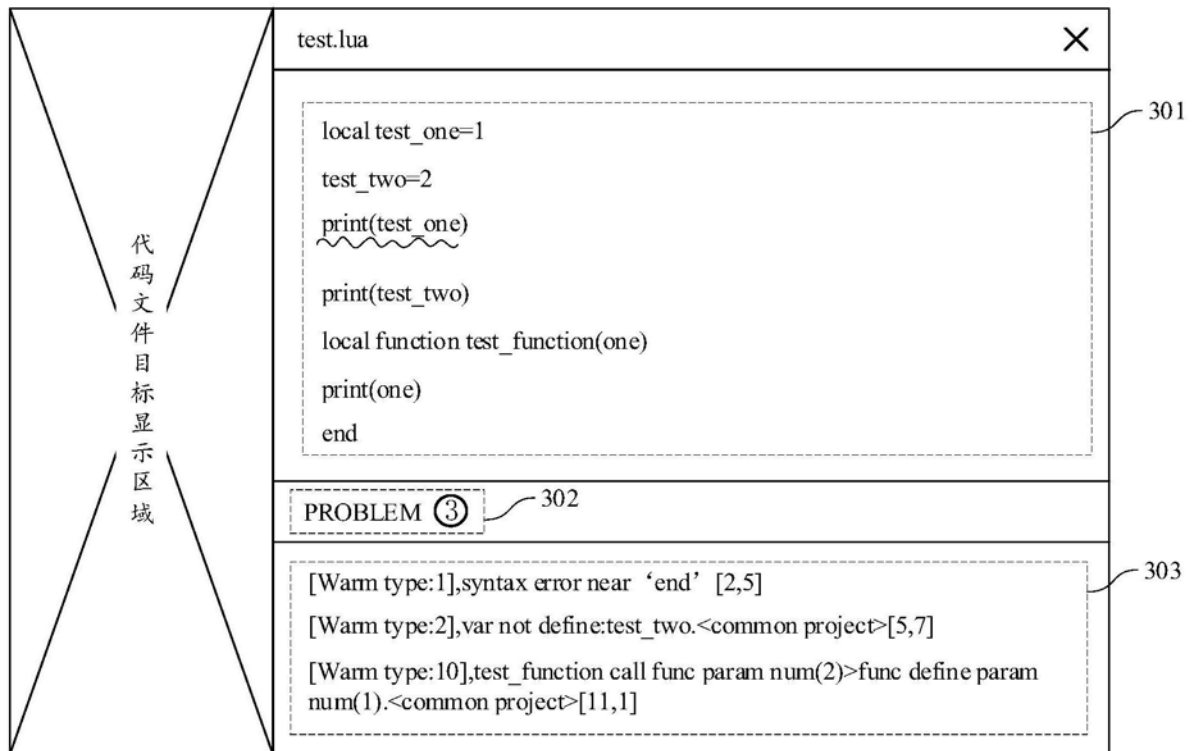


图3

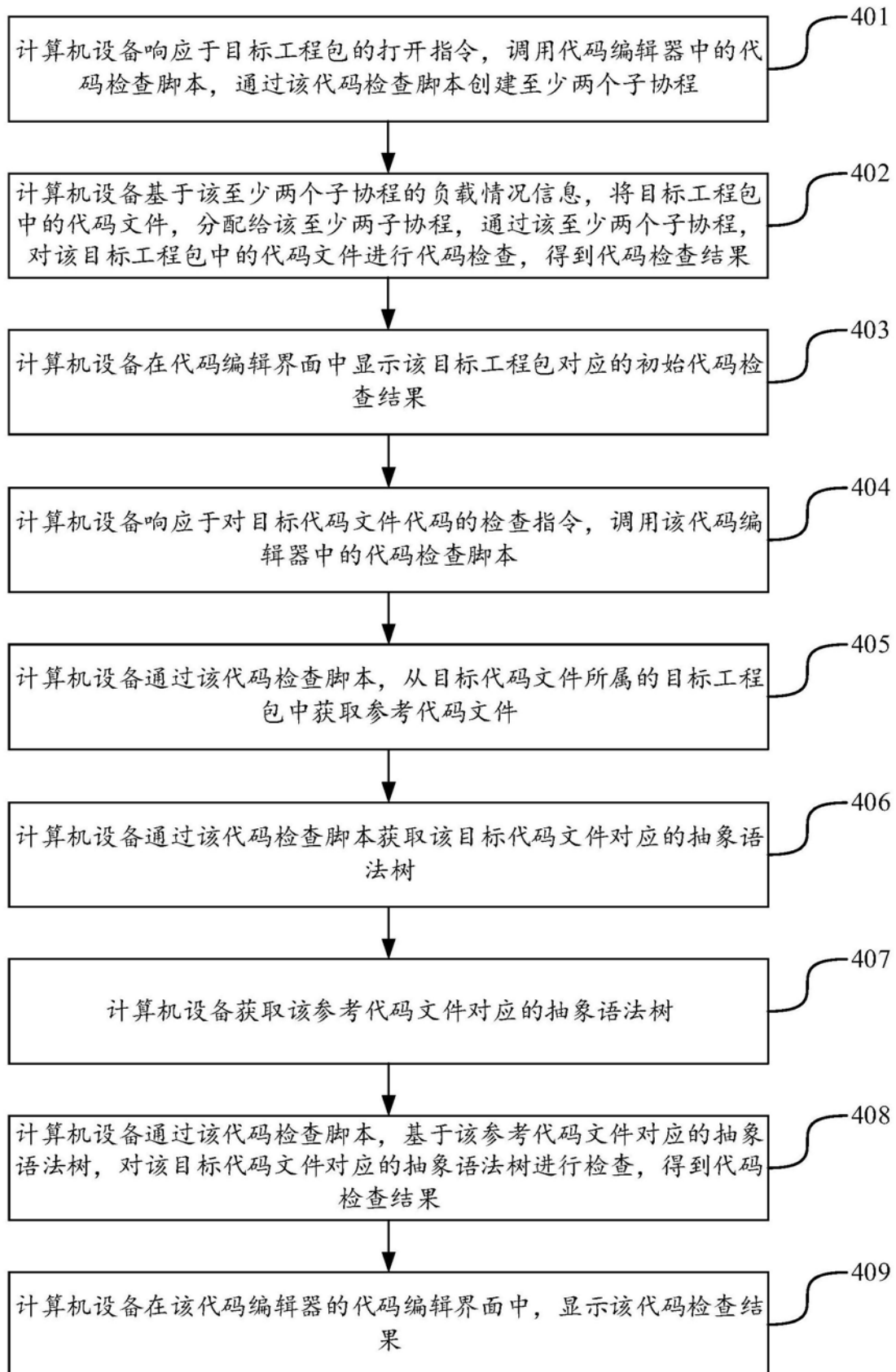


图4

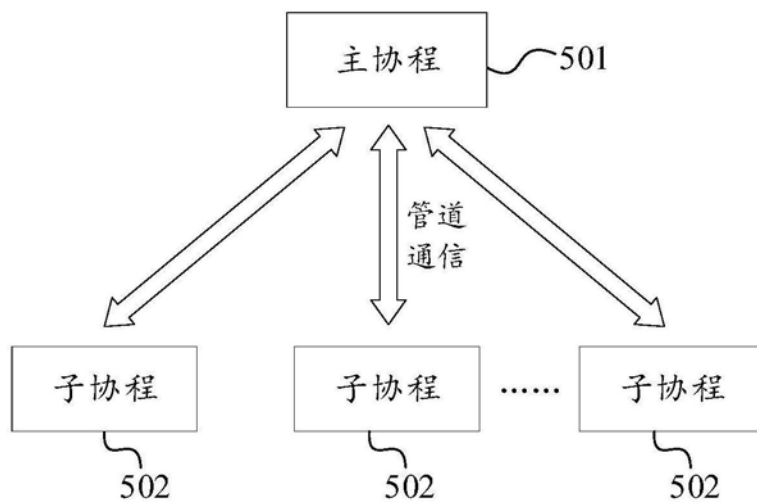


图5

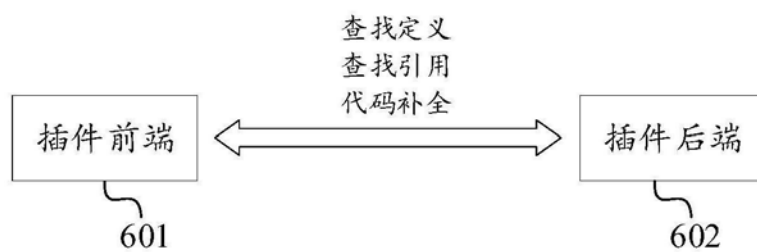


图6

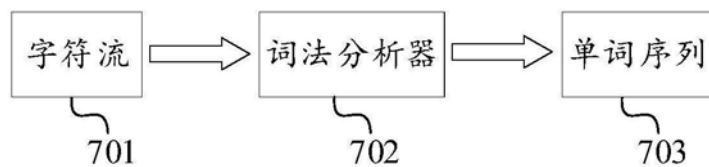


图7

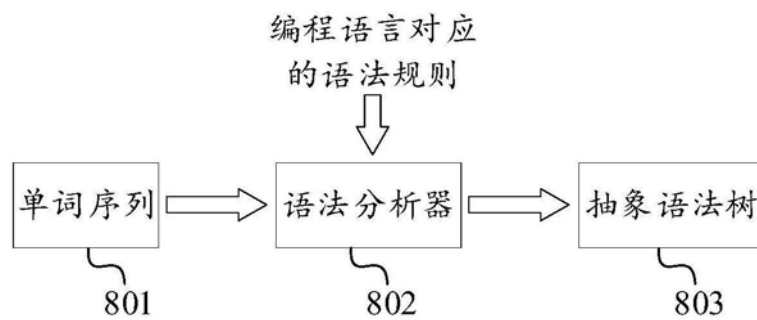


图8

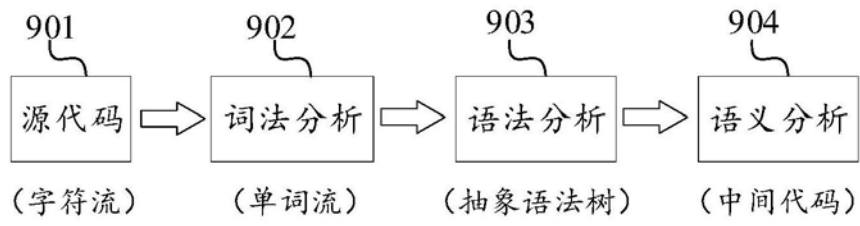


图9

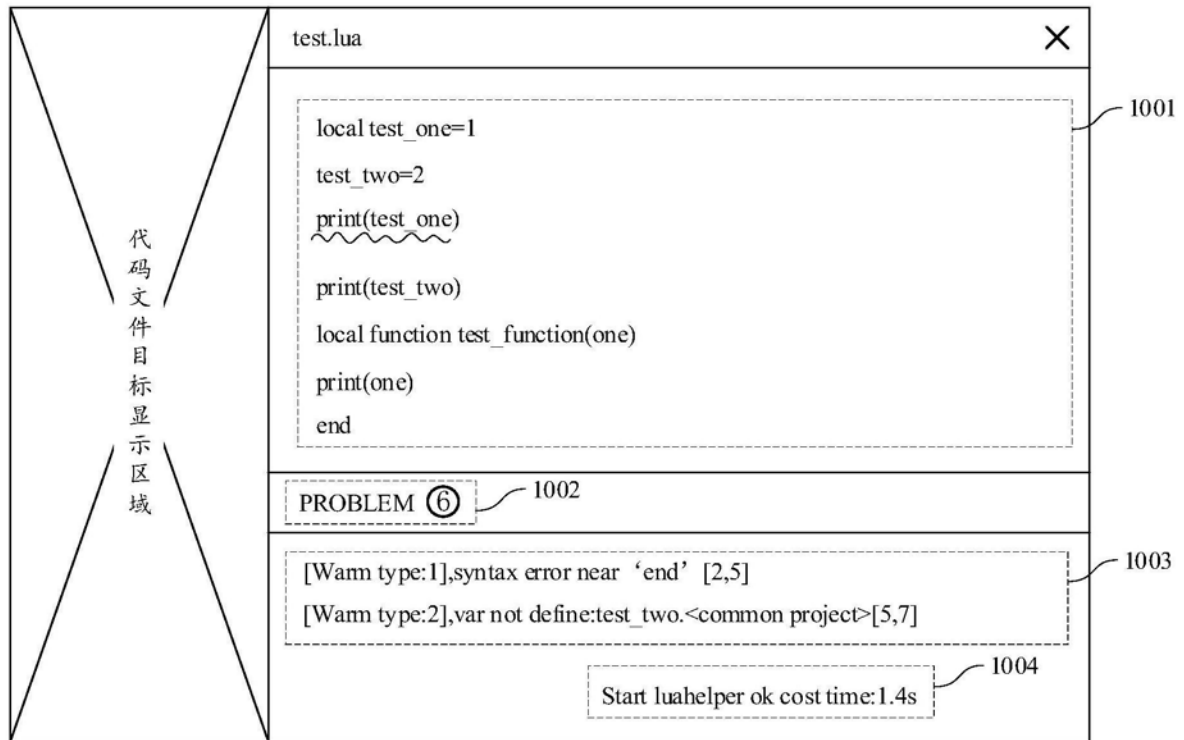


图10

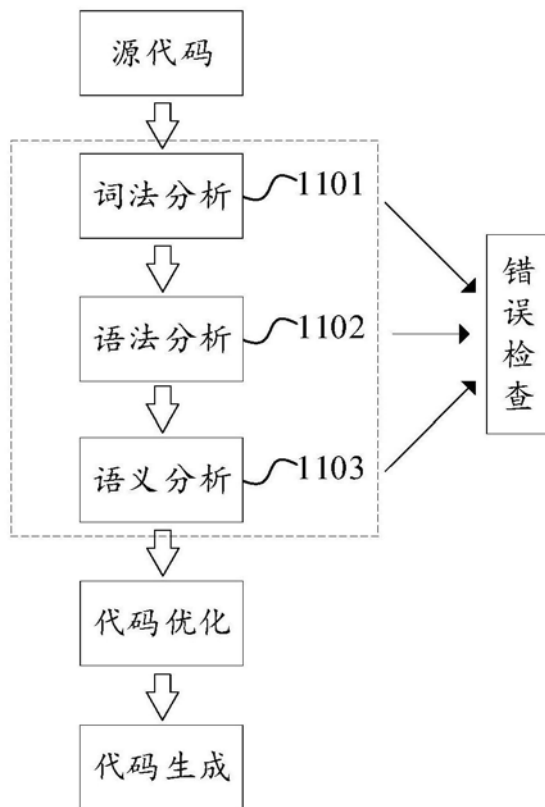


图11

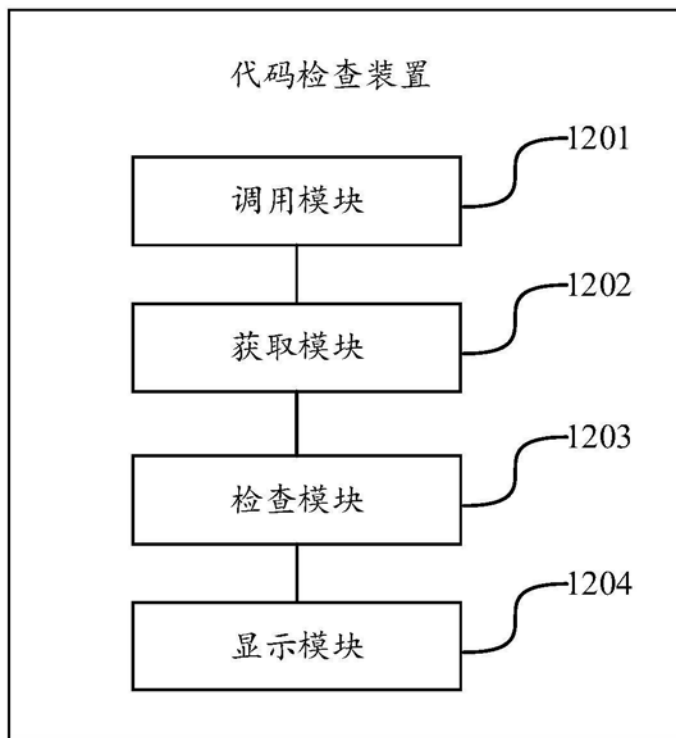


图12

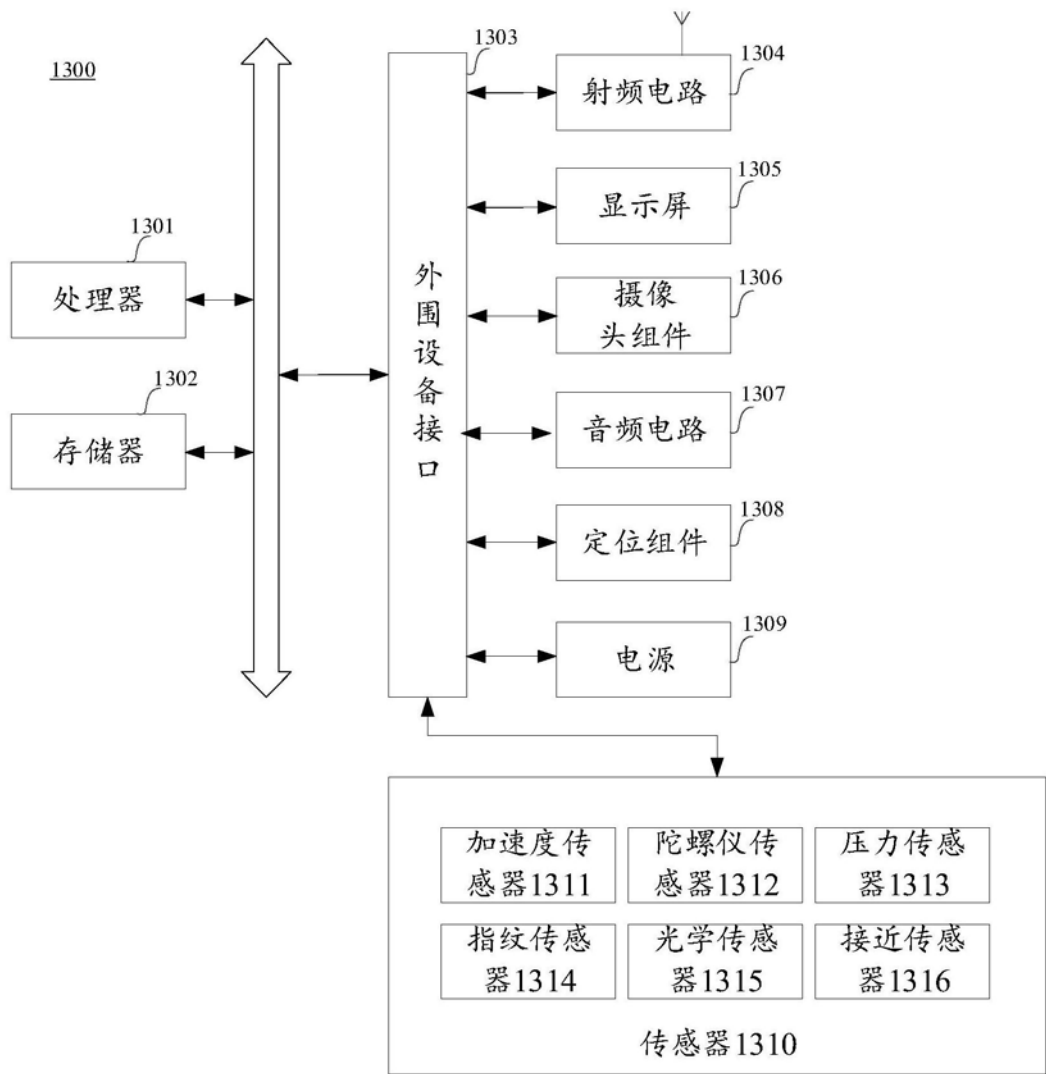


图13

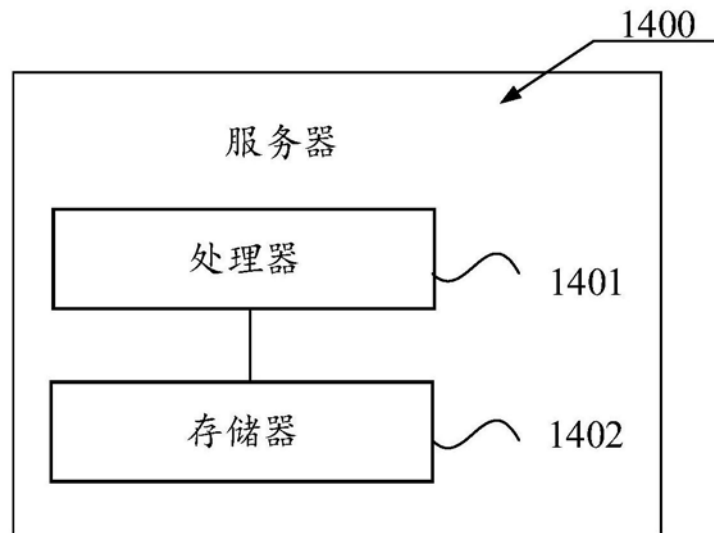


图14