用于软件开发的方法及系统

本申请要求在 2024 年 3 月 27 日提交俄罗斯联邦知识产权局、申请号为 2024107965 的俄罗斯联邦专利申请的优先权，发明名称为“用于软件开发的方法及系统”的俄罗斯联邦专利申请的优先权，其全部内容通过引用结合在本申请中。

技术领域

本申请涉及计算机技术领域，更具体地，涉及一种用于软件开发的方法及系统。

背景技术

随着信息技术的发展，软件开发的场景越来越多样化，例如，软件开发过程中可能涉及到多种编程语言、多种操作系统或多种硬件架构。开发场景的不同意味着整套开发编译链（或开发环境）的差异。这使得软件开发对开发环境和开发工具的要求也越来越高。

集成开发环境（integrated development environment，IDE）是用于提供程序开发环境的应用程序。不同的 IDE 的版本对开发场景的支持情况存在差异。例如，软件开发过程可能需要使用多种编程语言，不同的编程语言可能需要由不同的 IDE 版本来支持，在该情况下，用户需要选择多个版本的 IDE 以实现对对应的编程语言的支持。再如，软件开发过程可能涉及不同的操作系统，而 IDE 的版本通常与操作系统和芯片架构绑定，在该情况下，用户需要选择多个版本的 IDE 来支持在对应的操作系统中的软件开发。在上述过程中，用户在软件开发时，需要根据开发场景配置多种硬件和/或软件，同时还需要对不同的 IDE 中的软件开发进程等进行管理。这使得用户难以专注于软件开发的业务逻辑的实现，严重影响了用户的开发效率，从而影响了用户的开发体验。

发明内容

本申请提供一种用于软件开发的方法及系统，有利于提高用户的开发体验。

第一方面，提供了一种用于软件开发的方法，应用于集成开发环境 IDE，IDE 包括 IDE 前端和第一 IDE 后端，第一 IDE 后端支持运行于多种类型的环境，多种类型的环境包括以下至少一项：多种类型的操作系统或多种类型的硬件架构，该方法包括：IDE 前端获取用户的配置信息，配置信息包括以下至少一项：硬件架构的类型或操作系统的类型；IDE 前端根据配置信息在第一环境部署第一 IDE 后端，以使得通过 IDE 前端实现在第一环境中的第一 IDE 后端上的软件开发，第一环境为第一 IDE 后端支持的多种类型的环境中按照配置信息确定的环境。

根据本申请实施例的方案，IDE 分为 IDE 前端和 IDE 后端（如第一 IDE 后端）。软件开发的具体操作可以由第一 IDE 后端完成。相同的 IDE 后端支持运行于多种类型的环境中，有利于实现对多种开发场景的支持，提高了开发模式的灵活性。这样，用户可以根据需要选择合适的开发场景、合适的硬件架构以及操作系统等来进行软件开发，而无需考虑 IDE 版本问题，无需自行配置安装开发环境，这使得用户可以专注于软件开发的业务逻辑自身，有利于提高开发效率，从而提高用户的开发体验。

示例性地，IDE 前端可以部署于本地系统中。

示例性地，第一 IDE 后端的数量可以为一个，也可以为多个。即 IDE 前端可以和一个 IDE 后端进行交互，也可以和多个 IDE 后端进行交互。

第一环境可以包括以下任一项或多项：本地容器、本地虚拟机、远端容器或远端虚拟机。

结合第一方面，在第一方面的某些实现方式中，IDE 前端运行于 IDE 的客户端程序或浏览器的客户端程序。

根据本申请实施例的方案，IDE 前端可以部署于 IDE 的客户端程序，也可以部署于浏览器的客户端程序中，为用户提供了多种选择方案，进一步提高了开发模式的灵活性，有利于进一步提高开发效率，从而提高进一步用户的开发体验。

结合第一方面，在第一方面的某些实现方式中，IDE 前端包括用户界面 UI 组件和适配层，UI 组件用

于提供用户界面，适配层用于为 UI 组件提供统一接口，以使 UI 组件运行于一种或多种客户端程序中。

根据本申请实施例的方案，适配层可以为 UI 组件提供统一接口，屏蔽不同的客户端程序的 API 接口之间的差异，以实现 UI 组件与桌面应用框架层的交互，这样，运行于不同客户端程序/框架下的前端 UI 可以由同一套 UI 组件代码实现。或者说，前端 UI 可以支持同一套 UI 组件代码通过跨平台桌面框架适配，从而运行于多种客户端程序和/或框架中。

结合第一方面，在第一方面的某些实现方式中，第一 IDE 后端包括 IDE 内核，IDE 内核用于提供多种编程语言的语言特性。

示例性地，语言特性可以包括以下任一项或多项：代码补全、浏览、跳转、检查、编译构建、测试、调试或重构等。

根据本申请实施例的方案，IDE 后端可以采用统一的 IDE 内核，同一个 IDE 内核可以支持多种编程语言。这样，用户可以根据需要选择合适的编程语言来进行软件开发，无需考虑 IDE 版本问题，进一步提高了开发模式的灵活性，有利于进一步提高开发效率，从而提高进一步用户的开发体验。

结合第一方面，在第一方面的某些实现方式中，IDE 内核包括：索引系统，用于存储源代码的索引信息，语言解析适配层，用于调用源代码的编程语言对应的语言解析器以解析源代码，语言特性层，用于提供多种编程语言的语言特性，多种编程语言包括源代码的编程语言。

根据本申请实施例的方案，IDE 后端可以采用统一的 IDE 内核，该 IDE 内核包括语言相关层和语言无关层，语言相关层，即语言特性层可以提供多种编程语言的语言特性，语言无关层中的索引系统和语言解析适配层与语言无关，可以适用于所有编程语言的语言特性，这使得同一个 IDE 内核可以支持多种编程语言。

结合第一方面，在第一方面的某些实现方式中，多种类型的环境包括 IDE 前端所在的本地系统和/或本地系统以外的其他环境。

结合第一方面，在第一方面的某些实现方式中，第一环境包括一个或多个实例，IDE 前端根据配置信息在第一环境上部署第一 IDE 后端，包括：IDE 前端根据配置信息获取一个或多个实例，一个或多个实例中部署有第一 IDE 后端；或者，IDE 前端根据配置信息获取一个或多个实例；在一个或多个实例的部分或全部实例中未部署第一 IDE 后端的情况下，IDE 前端在未部署第一 IDE 后端的实例中部署第一 IDE 后端。结合第一方面，在第一方面的某些实现方式中，方法还包括：IDE 前端启动一个或多个实例中的第一实例中的第一 IDE 后端，以使得第一实例中的第一 IDE 后端提供第一实例支持的第一编程语言的语言特性。

结合第一方面，在第一方面的某些实现方式中，方法还包括：IDE 前端与第一实例中的第一 IDE 后端建立长连接。

这样可以使得第一实例中的第一 IDE 后端随时处于待命状态，从而提高交互的效率。

结合第一方面，在第一方面的某些实现方式中，IDE 前端启动一个或多个实例中的第一实例中的第一 IDE 后端，包括：IDE 前端根据第一操作启动第一实例中的第一 IDE 后端，第一操作指示第一编程语言。

结合第一方面，在第一方面的某些实现方式中，第一操作包括第一代码文件的打开操作，第一编程语言为第一代码文件的编程语言。

结合第一方面，在第一方面的某些实现方式中，配置信息由携带验证信息的登录请求指示。

结合第一方面，在第一方面的某些实现方式中，在第一 IDE 后端运行于本地系统以外的其他环境的情况下，IDE 前端和第一 IDE 后端之间通过安全外壳 SSH 隧道进行通信。

结合第一方面，在第一方面的某些实现方式中，该方法还包括：IDE 前端将第一请求发送至第一环境中的第一 IDE 后端；第一环境中的第一 IDE 后端生成对第一请求的第一响应；第一环境中的第一 IDE 后端将第一响应返回至 IDE 前端。

IDE 前端和第一环境中的第一 IDE 后端配合，以实现在第一 IDE 后端所在的系统中的软件开发。

结合第一方面，在第一方面的某些实现方式中，IDE 还包括运行于第二环境的第二 IDE 后端，该方法还包括：IDE 前端将第二请求发送至第一环境中的第一 IDE 后端；第一环境中的第一 IDE 后端将第二请求转发至第二环境中的第二 IDE 后端；第二环境中的第二 IDE 后端生成对第二请求的第二响应；第二环境中的第二 IDE 后端将第二响应发送至第一环境中的第一 IDE 后端；第一环境中的第一 IDE 后端将第二响应返回至 IDE 前端。

第一 IDE 后端和第二 IDE 后端可以运行于不同环境中，即第一环境和第二环境的类型可以不同。

IDE 前端、第一 IDE 后端和第二 IDE 后端配合，以实现第二 IDE 后端所在的系统中的软件开发。

结合第一方面，在第一方面的某些实现方式中，该方法还包括：第一环境中的第一 IDE 后端通过显示服务器协议将远程环境中运行的图形用户界面程序投射到 IDE 前端所在的本地系统。

这样，可以使得远程环境中的进程的图形用户界面可以输出到 IDE 前端所在的操作系统中。

第二方面，提供了一种用于软件开发的系统，包括：IDE 前端和第一 IDE 后端，第一 IDE 后端支持运行于多种类型的环境，多种类型的环境包括以下至少一项：多种类型的操作系统或多种类型的硬件架构，IDE 前端用于：获取用户的配置信息，配置信息包括以下至少一项：硬件架构的类型或操作系统的类型；根据配置信息在第一环境部署第一 IDE 后端，以使得通过 IDE 前端实现在第一环境中的第一 IDE 后端上的软件开发，第一环境为第一 IDE 后端支持的多种类型的环境中按照配置信息确定的环境。

根据本申请实施例的方案，IDE 分为 IDE 前端和 IDE 后端（如第一 IDE 后端）。软件开发的具体操作可以由第一 IDE 后端完成。相同的 IDE 后端支持运行于多种类型的环境中，有利于实现对多种开发场景的支持，提高了开发模式的灵活性。这样，用户可以根据需要选择合适的开发场景、合适的硬件架构以及操作系统等来进行软件开发，而无需考虑 IDE 版本问题，无需自行配置安装开发环境，这使得用户可以专注于软件开发的业务逻辑自身，有利于提高开发效率，从而提高用户的开发体验。

结合第二方面，在第二方面的某些实现方式中，IDE 前端运行于 IDE 的客户端程序或浏览器的客户端程序。

结合第二方面，在第二方面的某些实现方式中，IDE 前端包括用户界面 UI 组件和适配层，UI 组件用于提供用户界面，适配层用于为 UI 组件提供统一接口，以使 UI 组件运行于一种或多种客户端程序中。

结合第二方面，在第二方面的某些实现方式中，第一 IDE 后端包括 IDE 内核，IDE 内核用于提供多种编程语言的语言特性。

结合第二方面，在第二方面的某些实现方式中，IDE 内核包括：索引系统，用于存储源代码的索引信息，语言解析适配层，用于调用源代码的编程语言对应的语言解析器以解析源代码，语言特性层，用于提供多种编程语言的语言特性，多种编程语言包括源代码的编程语言。

结合第二方面，在第二方面的某些实现方式中，多种类型的环境包括 IDE 前端所在的本地系统和/或本地系统以外的其他环境。

结合第二方面，在第二方面的某些实现方式中，第一环境包括一个或多个实例，IDE 前端具体用于：根据配置信息获取一个或多个实例，一个或多个实例中部署有第一 IDE 后端；或者，根据配置信息获取一个或多个实例；在一个或多个实例的部分或全部实例中未部署第一 IDE 后端的情况下，在未部署第一 IDE 后端的实例中部署第一 IDE 后端。

结合第二方面，在第二方面的某些实现方式中，IDE 前端还用于：启动一个或多个实例中的第一实例中的第一 IDE 后端，以使得第一实例中的第一 IDE 后端提供第一实例支持的第一编程语言的语言特性。

结合第二方面，在第二方面的某些实现方式中，IDE 前端还用于：与第一实例中的第一 IDE 后端建立长连接。

结合第二方面，在第二方面的某些实现方式中，IDE 前端具体用于：根据第一操作启动第一实例中的第一 IDE 后端，第一操作指示第一编程语言。

结合第二方面，在第二方面的某些实现方式中，第一操作包括第一代码文件的打开操作，第一编程语言为第一代码文件的编程语言。

结合第二方面，在第二方面的某些实现方式中，配置信息由携带验证信息的登录请求指示。

结合第二方面，在第二方面的某些实现方式中，IDE 前端还用于：将第一请求发送至第一环境中的第一 IDE 后端；以及第一环境中的第一 IDE 后端用于：生成对第一请求的第一响应；将第一响应返回至 IDE 前端。

结合第二方面，在第二方面的某些实现方式中，系统还包括运行于第二环境的第二 IDE 后端，以及 IDE 前端还用于：将第二请求发送至第一环境中的第一 IDE 后端；第一环境中的第一 IDE 后端用于将第二请求转发至第二环境中的第二 IDE 后端；第二环境中的第二 IDE 后端用于：生成对第二请求的第二响应；将第二响应发送至第一环境中的第一 IDE 后端；以及第一环境中的第一 IDE 后端还用于：将第二响应返回至 IDE 前端。

结合第二方面，在第二方面的某些实现方式中，第一环境中的第一 IDE 后端还用于通过显示服务器协议将远程环境中运行的图形用户界面程序投射到 IDE 前端所在的本地系统。

应理解，在上述第一方面中对相关内容的扩展、限定、解释和说明也适用于第二方面中相同的内容。

第三方面，提供了一种计算设备集群，包括至少一个计算设备，每个计算设备包括处理器和存储器。至少一个计算设备的处理器用于执行至少一个计算设备的存储器中存储的指令，以使得计算设备集群执行第一方面或第一方面的任意一种实现方式中的方法。

第四方面，提供一种计算机可读介质，包括计算机程序指令，当计算机程序指令由计算设备集群执行时，计算设备集群执行第一方面或第一方面的任意一种实现方式中的方法。

第五方面，提供一种包含指令的计算机程序产品，当指令被计算设备集群运行时，使得计算设备集群执行上述第一方面或第一方面的任意一种实现方式中的方法。

附图说明

图 1 是一种异构操作系统的架构的示意性结构图。

图 2 是本申请实施例的一种 IDE 架构的示意性结构图。

图 3 是本申请实施例的一种跨平台桌面框架的示意性结构图。

图 4 是本申请实施例的另一种 IDE 架构的示意性结构图。

图 5 是本申请实施例的一种 IDE 内核的示意性结构图。

图 6 是本申请实施例的另一种 IDE 架构的示意性结构图。

图 7 是本申请实施例的一种应用投射的示意图。

图 8 是本申请实施例的两种场景下的应用投射的示意图。

图 9 是本申请实施例的一种用于软件开发的方法的示意性流程图。

图 10 是本申请实施例的方案应用于本地 IDE 开发场景下的示意图。

图 11 是本申请实施例的方案应用于 WebIDE 开发场景下的示意图。

图 12 是本申请实施例的方案应用于另一种 WebIDE 开发场景下的示意图。

图 13 是本申请实施例的方案应用于本地 IDE 在远程开发的开发场景下的示意图。

图 14 是本申请实施例的方案应用于 WebIDE 在远程开发的开发场景下的示意图。

图 15 是本申请实施例的方案应用于本地 IDE 开发场景下的示意图。

图 16 是本申请实施例的方案应用于 WebIDE 开发场景下的示意图。

图 17 是本申请实施例的方案应用于本地 IDE 在远程开发的开发场景下的示意图。

图 18 是本申请实施例的另一种用于软件开发的方法的流程图。

图 19 是本申请实施例的一种用于软件开发的系统的示意性框图。

图 20 是本申请实施例的一种计算设备的示意性框图。

图 21 是本申请实施例的一种计算设备集群的示意性框图。

图 22 是本申请实施例的另一种计算设备集群的示意性框图。

具体实施方式

下面将结合附图，对本申请中的技术方案进行描述。

以下实施例中所使用的术语只是为了描述特定实施例的目的，而并非旨在作为对本申请的限制。如在本申请的说明书和所附权利要求书中所使用的那样，单数表达形式“一个”、“一种”和“该”旨在包括例如“一个或多个”这种表达形式，除非其上下文中明确地有相反指示。还应当理解，在本申请以下各实施例中，“至少一个”、“至少一项”、“一个或多个”是指一个、两个或两个以上。“第一”、“第二”以及各种数字编号只是为了描述方便进行的区分，并不用来限制本申请实施例的范围。“和/或”，用于描述对应对象的对应关系，表示可以存在三种关系，例如，“A 和/或 B”可以表示：只存在 A，只存在 B 以及同时存在 A 和 B 三种情况，其中 A，B 可以是单数或复数。字符“/”一般表示前后关联对象是一种“或”的关系。下文各过程的序号的大小并不意味着执行顺序的先后，各过程的执行顺序应以其功能和内在逻辑确定，而不应对本申请实施例的实施过程构成任何限定。例如，本申请实施例中，“301”、“401”、“501”等字样仅为了描述方便作出的标识，并不是对执行步骤的次序进行限定。

在本说明书中描述的参考“一个实施例”或“一些实施例”等意味着在本申请的一个或多个实施例中包括结合该实施例描述的特定特征、结构或特点。本申请中，“示例性的”或者“例如”等词用于表示作例子、例证或说明。本申请中被描述为“示例性的”或者“例如”的任何实施例或设计方案不应被

解释为比其他实施例或设计方案更优选或更具优势。确切而言，使用“示例性的”或者“例如”等词旨在以具体方式呈现相关概念。术语“包括”、“包含”、“具有”及它们的变形都意味着“包括但不限于”，除非是以其他方式另外特别强调。在本申请实施例中，“当……时”、“在……的情况下”、“若”以及“如果”等描述均指在某种客观情况下设备会做出相应的处理，并非是限定时间，且也不要求设备在实现时一定要有判断的动作，也不意味着存在其它限定。

在本申请中，“用于指示”可以包括用于直接指示和用于间接指示。当描述某一指示信息用于指示A时，可以包括该指示信息直接指示A或间接指示A，而并不代表该指示信息中一定携带有A。

为了便于本领域技术人员更好地理解本申请的技术方案，下面先对本申请实施例可能涉及的一些术语进行说明。

1. 集成开发环境（integrated development environment，IDE）：

IDE是用于提供程序开发环境的应用程序，一般包括代码编辑器、编译器、调试器和图形用户界面等工具，是集成了代码编写功能、分析功能、编译功能、调试功能等一体化的开发软件服务套。

IDE可以包括本地IDE和网页（web）IDE。

本地IDE也可以称为桌面IDE。本地IDE安装于用户的本地设备中。即开发人员可以在本地设备上安装开发环境。示例性地，本地设备可以为终端设备，例如，台式电脑、笔记本电脑或手机等。

Web IDE指的是在线IDE，即针对开发人员的在线集成开发环境。Web IDE可以包括IDE前端和IDE后端，其中，IDE前端运行于浏览器中，IDE后端可以运行于远程环境中。例如，远程环境可以为云服务器。这样，编程人员无需在本地设备上安装开发环境，只需打开浏览器即可进行软件开发，同时Web IDE还可以支持断点调试、版本管理以及团队开发等功能。

2. 异构系统：

异构系统指的是由不同类型的硬件或软件组成的系统，这些组成部分可以是异构的。异构系统的各个组成部分具有自身的自治性，在实现数据共享的前提下，仍保留自己的应用特性、完整特征和安全特性。

异构操作系统（operating system, OS）中可以包括多个异构的操作系统，如鸿蒙操作系统和欧拉操作系统等。一些操作系统可以在不同的硬件架构上运行。硬件架构也可以称为处理器架构、芯片架构或硬件平台等。不同的硬件架构也可以替换为不同的处理器架构、不同的芯片架构、不同的硬件平台、不同的处理器类型、不同的硬件类型、不同的芯片类型或不同的中央处理器（central processing unit，CPU）等。以鸿蒙操作系统和欧拉操作系统为例，这些系统可以兼容的硬件架构包括但不限于：鲲鹏、X86和ARM等。异构操作系统的设计和实现具备高性能、可扩展性和灵活性。

图1示出了一种异构操作系统的架构的示意性结构图。如图1所示，异构操作系统的架构可以包括用户交互层、应用层、系统服务层、异构操作系统内核子系统和硬件平台。

用户交互层用于提供人机交互界面，如图形用户界面（graphical user interface，GUI）或命令行界面（command-line interface，CLI）等。

应用层用于为应用程序提供多语言的用户程序框架，以及各种软硬件服务对外开放的多语言的框架的应用程序编程接口（application programming interface, API）；同时为采用该系统的设备提供了多语言的框架的API。

系统服务层包括以下一项或多项服务：系统基本能力、基础软件服务、增强软件服务或硬件服务等，用于为分布式应用在该系统多设备上的运行、调度、迁移等操作提供了基础能力，可以包括以下任一项或多个子系统：分布式软总线、分布式数据管理、分布式任务调度、多语言运行时、公共基础库、多模输入、图形、安全或AI等子系统。

异构操作系统内核子系统用于通过屏蔽多内核差异，对上层提供基础的内核能力，可以包括以下任一项或多项：进程/线程管理、内存管理、文件系统、网络管理或外设管理等。同时可以为硬件生态提供统一外设访问能力和驱动开发、管理框架。

如图1所示，可以在系统中创建基于不同的操作系统的容器，如图1中的容器#1、容器#2和容器#3。容器#1为基于OS #A的容器，容器#2为基于OS #B的容器，OS #A和OS #B为不同类型的操作系统。

示例性地，图1中的硬件平台可以包括鲲鹏、X86或ARM中的任一项。

随着信息技术的发展，软件开发的场景越来越多样化，这使得软件开发对开发环境和开发工具的要求也越来越高。不同的IDE版本对开发场景的支持情况存在差异。用户需要根据开发场景自行配置、创

建适合的开发环境，然后才能进行软件开发。

以操作系统和硬件架构为例， IDE 的版本（version）通常是和操作系统以及硬件架构绑定的。换言之，一款 IDE 产品对应多个版本，一个版本只能运行在一种操作系统和硬件架构上。例如，某个 IDE 包括 4 个版本：版本#1，版本#2，版本#3 和版本#4。该 4 个版本分别运行于以下 4 种场景：OS #1+硬件类型#1，OS #1+硬件类型#2，OS #2+硬件类型#3，OS #2+硬件类型#4。用户需要在某种硬件架构上进行软件开发，则需要安装对应的 IDE 版本，创建相应的开发环境。例如，用户需要基于硬件类型#1 进行软件开发，则需要安装能够支持硬件类型#1 的 IDE 的版本，即版本#1。

以远程开发场景为例，用户在桌面应用程序中启用远程开发工作，根据开发场景来配置远端的开发环境。具体地，可以包括如下步骤：

1）IDE 前端通过 SSH 连接到远端服务器上。同时用户需要确保待开发的操作系统已经安装在该远端服务器上。

例如，若需要开发操作系统#1 中的应用程序，则需要确保操作系统#1 已经安装在该远端服务器上了。

2）配置开发环境。

用户需要根据开发场景选择合适的 IDE 的版本，实现开发环境的配置。该版本的 IDE 用于运行用户的开发项目。

具体地，用户需要根据编程语言以及芯片架构等确定合适的 IDE。

3）创建开发环境。

在用户选择之后，系统会将相应的 IDE 后端下载到远端服务器上，实现远端的开发环境的创建。即开发环境作为 IDE 后端运行。

4）连接开发环境。

即 IDE 前端和 IDE 后端的连接。之后用户可以进行远程开发。

从上述方案可以看出，在不同的开发场景下进行软件开发时，用户需要根据不同的开发场景来选择相应的硬件、软件以及适配的 IDE 版本等，相应地，用户可能需要配置和/或安装多种硬件和软件。此外，用户还需要对不同的 IDE 中的软件开发进程等进行管理。这使得用户无法专注于软件开发本身，严重影响了开发效率，从而影响用户的开发体验。

有鉴于此，本申请实施例提供了一种用于软件开发的系统架构，能够支持多种开发场景，例如，多种编程语言、多种操作系统、多种硬件架构以及多种开发模式等，有利于保证用户的开发效率，提高用户的开发体验。

在本申请实施例中，用于软件开发的系统架构也可以称为 IDE 架构。本申请实施例中的 IDE 架构可以称为“一核多模”架构，即同一个 IDE 内核可以支持多种开发场景。

示例性地，多种操作系统可以包括以下任一项或多项：欧拉操作系统、鸿蒙操作系统或其他操作系统。例如，其他操作系统可以为信创操作系统。

示例性地，多种硬件架构可以包括以下任一项或多项：鲲鹏、X86 或 ARM 等。

多种开发模式的具体描述可以参考后文，此处不展开描述。

图 2 示出了本申请实施例的一种 IDE 架构的示意性结构图。下面结合图 2 对本申请实施例的系统架构进行说明。

如图 2 所示，系统 200 可以包括 IDE 前端 210 和第一 IDE 后端 220。IDE 前端 210 和第一 IDE 后端 220 之间可以进行通信。

“第一 IDE 后端”中的“第一”仅用于说明该 IDE 后端可以与该 IDE 前端进行通信，以便实现软件开发，不具有其他限定作用。

IDE 前端也可以称为前端 IDE，IDE 后端也可以称为后端 IDE。

下面对 IDE 前端 210 进行说明。

IDE 前端 210 可以部署于本地操作系统中。在本申请实施例中，本地操作系统也可以称为本地系统。在一种可能的理解方式中，IDE 前端 210 所在的操作系统可以视为本地操作系统。

IDE 前端 210 可以用于提供用户界面（user interface, UI）。

或者说，IDE 前端 210 可以包括前端 UI。例如，IDE 前端 210 可以为部署于本地操作系统中的轻量化的前端 UI。

用户可以与 IDE 前端 210 进行交互。

IDE 前端 210 支持运行于多种客户端程序/框架中。

可选地，IDE 前端 210 可以运行于 IDE 的客户端程序中。该 IDE 的客户端程序可以部署于本地操作系统中。

用户可以下载 IDE 的客户端程序，并将其安装在本地操作系统中，通过运行 IDE 的客户端程序实现 IDE 前端 210 的运行。

IDE 的客户端程序也可以称为桌面 IDE 的客户端程序、本地 IDE 的客户端程序或 IDE 的客户端程序。

可选地，IDE 前端 210 可以运行于浏览器的客户端程序中。该浏览器的客户端程序可以部署于本地操作系统中。

用户无需下载 IDE 的客户端程序，在浏览器中即可实现 IDE 前端 210 的运行。

图 3 示出了本申请实施例的一种跨平台桌面框架的示意图。如图 3 所示，IDE 前端 210 部署于该框架中。

如图 3 所示，该框架还包括桌面应用框架层 310。桌面应用框架层 310 可以包括一个或多个客户端程序。该一个或多个客户端程序用于构建跨平台的应用程序。例如，桌面应用框架层 310 的客户端程序可以包括以下任一项或多项：浏览器、Electron、Webview2、QT 或 Tauri 等。该浏览器可以为 Chromium。

该一个或多个客户端程序也可以替换为一个或多个框架。

不同的客户端程序和/或框架可能提供不同的应用程序编程接口（application programming interface, API）。

在一种可能的实现方式中，IDE 前端 210 可以包括 UI 组件 211 和适配层 212，如图 3 所示。

UI 组件 211 用于提供用户界面。

示例性地，UI 组件 211 可以包括以下任一项或多项：按钮、输入框、菜单、编辑器、窗口、对话框或终端（terminal）命令行窗口等。

适配层 212 用于为 UI 组件 211 提供统一接口，以使 UI 组件 212 运行于一种或多种客户端程序中。

或者说，适配层 212 用于为 UI 组件 211 提供统一接口，以实现 UI 组件与桌面应用框架层的交互。

适配层 212 也可以称为跨平台桌面框架适配层 212。

这样，运行于不同客户端程序/框架下的前端 UI 可以由同一套 UI 组件代码实现。或者说，前端 UI 可以支持同一套 UI 组件代码通过跨平台桌面框架适配，从而运行于多种客户端程序和/或框架中。

适配层 212 可以为 UI 组件 211 屏蔽不同的 API 之间的差异，为 UI 组件提供统一的接口，从而使得同一套 UI 组件代码可以适配于多种客户端程序和/或框架。

如图 3 所示，该框架还可以包括基础运行环境 320 和操作系统 330。基础运行环境 320 用于提供运行对应编程语言程序的软件环境。

示例性地，基础运行环境 320 可以包括以下任一项或多项：Node JS、Python 运行环境、Java 运行环境或底层虚拟机（low level virtual machine，LLVM）。

示例性地，操作系统 330 可以包括以下任一项或多项：鸿蒙 OS、欧拉 OS 或其他 XC OS。

下面对本申请实施例的第一 IDE 后端 220 进行说明。

第一 IDE 后端 220 可以部署于第一环境中。

“第一环境”中的“第一”仅用于说明该环境为第一 IDE 后端 220 所在的环境，不具有其他限定作用。

第一 IDE 后端 220 可以包括 IDE 内核。IDE 内核可以用于提供一种或多种编程语言的语言特性。

IDE 内核可以包括语言相关层和语言无关层。在 IDE 内核可以用于提供多种编程语言的语言特性的情况下，该 IDE 内核也可以称为多语言 IDE 内核。

图 4 示出了本申请实施例的另一种 IDE 架构的示意性结构图，图 4 所示的方案可以视为图 2 所示的方案的两种具体实现方式。

如图 4 所示，语言相关层可以包括语言特性层。语言无关层可以包括语言解析适配层和索引系统。

语言特性层可以用于提供多种编程语言的语言特性。

示例性地，多种编程语言可以包括以下任一项或多项：python、java、C++、C#或 JavaScript 等。

图 5 示出了本申请实施例的一种 IDE 内核的示意图。例如，如图 5 所示，该 IDE 内核的语言特性层包括四种语言特性：python 语言特性、java 语言特性、C++语言特性和 C#语言特性。应理解， 图 5 中的语言特性仅为示例，在其他实现方式中，语言特性层还可以提供其他数量的语言特性和/或其他编程语言

的语言特性，本申请实施例对此不做限定。

示例性地，语言特性可以包括以下任一项或多项：代码补全、浏览、跳转、检查、编译构建、测试、调试或重构等。

语言无关层可以被所有的语言特性共享。

语言解析适配层可以用于调用源代码的编程语言对应的语言解析器来解析源代码，以生成源代码的索引信息。源代码的索引信息可以被提供给语言特性使用。

索引系统可以用于存储源代码的索引信息。

第一 IDE 后端还可以用于提供其他功能，例如，存储代码模型或存储源代码等。

具体地，在软件开发过程中，以第一源代码为例，语言解析适配层可以调用第一源代码的编程语言对应的语言解析器以解析第一源代码，以生成第一源代码的索引信息供第一源代码的编程语言的语言特性使用。索引系统可以存储第一源代码的索引信息。语言特性层可以提供第一源代码的编程语言的语言特性。或者说，语言特性层中的第一源代码的编程语言的语言特性被激活，使得语言特性层提供该编程语言的语言特性。

例如，第一源代码可以为当前正在处理的源代码，或者，当前打开的代码文件中的源代码等。

第一 IDE 后端 220 的数量可以为一个，如图 4 的（a）所示，也可以为多个，如图 4 的（b）所示。或者说，IDE 前端可以与一个第一 IDE 后端 220 建立连接，也可以与多个第一 IDE 后端 220 建立连接。或者说，IDE 前端可以与一个第一 IDE 后端 220 交互，也可以与多个第一 IDE 后端 220 交互。

多个 IDE 后端 220 的区别可能包括以下任一项或多项：操作系统、硬件架构或编程语言等。

示例性地，用户可能需要开发多语言项目，即涉及多种编程语言的项目。在该情况下，每种编程语言可以由一个第一 IDE 后端 220 提供对应的服务。

以两种编程语言（如 python 和 java）为例，IDE 前端可以分别与两个第一 IDE 后端 220 进行交互。例如，该两个第一 IDE 后端 220 的 IDE 内核可以为图 5 所示的 IDE 内核。用户在对 python 文件进行操作时，则由其中一个第一 IDE 后端 220 的语言特性层提供 python 的语言特性，用户在对 java 文件进行操作时，则由另一个第一 IDE 后端 220 的语言特性层提供 java 的语言特性。

在本申请实施例的方案中，IDE 内核可以支持多种编程语言，即可以为多种编程语言提供对应的语言服务。在需要为不同的编程语言提供对应的语言服务时，可以由相同的 IDE 内核来实现。这样，用户无需根据编程语言来选择对应的 IDE 版本，有利于简化用户使用的复杂度，从而提高用户体验。

第一 IDE 后端 220 支持运行于多种类型的环境中。即第一环境的类型可以为多种类型。

该多种类型的环境包括以下至少一项：多种类型的操作系统或多种类型的硬件架构。

即在不同的操作系统或不同的硬件架构中均可以部署相同的 IDE 后端。即第一 IDE 后端 220 可以支持运行于多种类型的操作系统中。第一 IDE 后端可以支持运行于多种类型的硬件架构上。

或者说，第一 IDE 后端 220 的部署环境为可配置项。例如，用户可以自行配置一个或多个硬件架构的类型，和/或，一个或多个操作系统的类型，相同的第一 IDE 后端 220 可以分别被部署在用户配置的环境中。

示例性地，以操作系统为例，为不同的操作系统提供对应的运行时，使得第一 IDE 后端 220 可以运行于不同的操作系统中。例如，以编程语言为 Java，操作系统为欧拉 OS 为例，第一环境的操作系统可以为欧拉 OS，第一环境提供 Java 运行时，这样，第一 IDE 后端 220 可以运行于第一环境中。

下面对不同的划分方式进行示例性说明。

第一环境可以为本地系统或远程环境。即第一 IDE 后端 220 可以部署于本地系统或远程环境。

第一环境为本地系统，指的是，IDE 前端 210 和第一 IDE 后端 220 运行于同一本地系统中，或者说，同一操作系统中。在一种可能的理解方式中，IDE 前端 210 所在的本地系统以外的其他环境均可以视为远程环境。

以第一环境为远程环境为例，IDE 前端 210 还可以用于与第一环境建立连接。若第一环境中已经部署了第一 IDE 后端 220，则 IDE 前端 210 可以用于启动该第一 IDE 后端 220。若第一环境中未部署第一 IDE 后端 220，则 IDE 前端 210 可以在第一环境中部署第一 IDE 后端 220，并启动第一 IDE 后端 220。具体描述可以参考后文的方法 600，此处不展开描述。

第一环境所属的系统可以为同构系统或异构系统。在本申请实施例的方案中，同构系统也可以理解为同构操作系统，即同构系统中的操作系统均为同类型的操作系统。异构系统也可以理解为异构操作系

统，即异构系统中包括多种类型的操作系统。

第一环境可以包括本地计算资源或远端计算资源。

即第一 IDE 后端 220 可以部署于本地计算资源或远端计算资源中。

示例性地，计算资源可以包括容器和/或虚拟机。例如，第一 IDE 后端 220 可以部署于以下任一项：本地容器、本地虚拟机、远端容器或远端虚拟机。

以上仅为示例，不对本申请实施例的方案构成限定。在具体实现中，第一 IDE 后端 220 可以根据需要进行部署，例如，第一 IDE 后端 220 的部署环境可以由用户指定，即第一 IDE 后端 220 可以部署于用户指定的环境中。

如前所述，第一 IDE 后端 220 的数量可以为一个，也可以为多个。在第一 IDE 后端 220 的数量为多个的情况下，该多个第一 IDE 后端 220 的部署环境的类型可以相同，也可以不同。

在本申请实施例的方案中，第一 IDE 后端可以支持多种类型的环境，即能够运行在不同类型的环境中。这样，用户无需根据开发场景来选择对应的 IDE 版本，有利于简化用户使用的复杂度，从而提高用户体验。

进一步地，第一 IDE 后端 220 还可以包括后端服务代理（proxy）。后端服务 proxy 可以用于将 IDE 前端 210 的请求转发给 IDE 内核进行处理，然后将其响应返回给 IDE 前端 210。

下面以两种方式（方式#1 和方式#2）为例对软件开发的过程进行示例性说明。

方式#1：

在一种可能的实现方式中，IDE 前端 210 和第一 IDE 后端 220 之间相互配合，以完成软件开发。

在方式#1 中，可以由第一 IDE 后端 220 完成软件开发相关的操作，或者说，IDE 前端 210 和第一 IDE 后端 220 配合以实现第一 IDE 后端 220 所在的系统中的软件开发。

可选地，IDE 前端 210 可以用于将第一请求发送至第一 IDE 后端 220。

第一 IDE 后端 220 可以用于生成第一请求的第一响应，并第一响应返回至 IDE 前端 210。

“第一请求”中的“第一”仅用于说明该请求是由第一 IDE 后端 220 处理的，不具有其他限定作用。“第一响应”即为对“第一请求”的响应，不具有其他限定作用。

在该开发场景下，IDE 前端 210 将请求发送至第一 IDE 后端 220 进行处理。第一 IDE 后端 220 可以将该请求的响应返回至 IDE 前端。

示例性地，IDE 前端 210 发送的请求可以是基于用户的操作生成的。或者说，IDE 前端 210 将用户的操作以请求的形式发送至第一 IDE 后端 220。第一 IDE 后端 220 可以对该请求进行处理，并将处理结果以响应的形式返回给 IDE 前端 210。

示例性地，用户的操作可以为用户的代码操作，或者说，代码活动，例如，编辑、调试、测试、编译以及构建等。

例如，用户的代码操作可以为“编辑”操作，IDE 前端 210 将“编辑”请求发送至第一 IDE 后端 220，第一 IDE 后端 220 中的 IDE 内核可以对该请求进行处理，给出对应的操作或指令等，并返回给 IDE 前端 210。

方式#1 下的开发场景的具体示例可以参考后文中的图 10 和图 11。

方式#2：

图 6 示出了本申请实施例的另一种 IDE 架构的示意图。

在另一种可能的实现方式中，系统 200 还可以包括第二 IDE 后端 230，如图 6 所示。IDE 前端 210、第一 IDE 后端 220 和第二 IDE 后端 230 相互配合，以完成软件开发。

第二 IDE 后端 230 可以部署于第二环境中。第二环境可以为远程环境。

第一环境和第二环境可以为不同类型的环境。“第二环境”中的“第二”仅用于区别于第一 IDE 后端 220 所在的环境，不具有其他限定作用。

第二环境所属的系统可以为同构系统或异构系统。

在方式#2 中，可以由第二 IDE 后端 230 完成软件开发相关的操作，或者说，IDE 前端 210、第一 IDE 后端 220 和第二 IDE 后端 230 配合以实现第二 IDE 后端 230 所在的系统中的软件开发。

在一种可能的理解方式中，第一 IDE 后端 220 和第二 IDE 后端 230 的区别可以理解为，第一 IDE 后端 220 无需经过其他 IDE 后端的转发来实现与 IDE 前端的通信，第二 IDE 后端 230 需要经过其他 IDE 后端的转发来实现与 IDE 前端的通信。“第二 IDE 后端”中的“第二”仅用于说明该 IDE 后端需要经过其

他 IDE 后端的转发来实现与 IDE 前端的通信，不具有其他限定作用。

第二 IDE 后端 230 包括语言相关层和语言无关层，前文中对第一 IDE 后端 220 的描述也适用于第二 IDE 后端 230，此处不再赘述。

第一环境和第二环境之间可以建立连接，以使得 IDE 前端 210 和第二 IDE 后端 230 之间可以通过第一 IDE 后端 210 的转发实现通信。

可选地，IDE 前端 210 可以用于将第二请求通过第一 IDE 后端 220 发送至第二 IDE 后端 230。第二 IDE 后端 230 可以用于生成第二请求的第二响应，并第二响应通过第一 IDE 后端 220 返回至 IDE 前端 210。

“第二请求”中的“第二”仅用于说明该请求是由第二 IDE 后端 230 处理的，不具有其他限定作用。“第二响应”即为对“第二请求”的响应，不具有其他限定作用。

在该开发场景下，IDE 前端 210 发送的请求经由第一 IDE 后端 220 转发至第二 IDE 后端 230 进行处理，由第二 IDE 后端 230 将该请求的响应通过第一 IDE 后端 220 返回至 IDE 前端 210。

其他描述可以参考前文，此处不再赘述。

方式#2 下的开发场景的具体示例可以参考后文中的图 13 和图 14。

本申请实施例的方案中，IDE 分为 IDE 前端和 IDE 后端，IDE 前端用于提供用户界面，软件开发的具体操作可以由 IDE 后端完成。相同的 IDE 后端支持运行于多种类型的环境中，有利于实现对多种开发场景的支持，提高了开发模式的灵活性。这样，用户可以根据需要选择合适的开发场景、合适的硬件架构以及操作系统等来进行软件开发，而无需考虑 IDE 版本问题，无需自行配置安装开发环境，这使得用户可以专注于软件开发的业务逻辑自身，有利于提高开发效率，从而提高用户的开发体验。

此外，本申请实施例的方案中，IDE 前端可以部署于 IDE 的客户端程序，也可以部署于浏览器的客户端程序中，为用户提供了多种选择方案，进一步提高了开发模式的灵活性，有利于进一步提高开发效率，从而提高进一步用户的开发体验。

此外，本申请实施例的方案中，IDE 后端可以采用统一的 IDE 内核，同一个 IDE 内核可以支持多种编程语言。这样，用户可以根据需要选择合适的编程语言来进行软件开发，无需考虑 IDE 版本问题，进一步提高了开发模式的灵活性，有利于进一步提高开发效率，从而提高进一步用户的开发体验。

IDE 后端可以采用统一的 IDE 内核，该 IDE 内核包括语言相关层和语言无关层，语言相关层，即语言特性层可以提供多种编程语言的语言特性，语言无关层中的索引系统和语言解析适配层与语言无关，可以适用于所有编程语言的语言特性，这使得同一个 IDE 内核可以支持多种编程语言。

在本申请实施例的方案中，IDE 前端的部署环境、IDE 后端的部署环境以及编程语言均可以为可配置项，该 IDE 架构可以支持多种开发场景和多种编程语言，大大提高了软件开发方式的灵活性和可扩展性。用户可以根据自身的需要灵活地选择合适的开发场景和编程语言，而无需考虑开发环境的部署，有利于提高用户的开发体验。

进一步地，第一 IDE 后端 220 还可以用于应用投射。

可选地，第一 IDE 后端 220 还可以用于通过显示服务器协议，将第三环境中的 GUI 程序投射到 IDE 前端 210 所在的操作系统（如本地系统）中。第三环境为远程环境。

将 GUI 程序投射到 IDE 前端 210 所在的操作系统，即将 GUI 程序的 UI 发送到该操作系统中。

应用投射可以由用户发起。示例性地，用户可以通过 IDE 的客户端程序或浏览器的客户端程序连接到远程环境，通过命令行启动远程环境中的 GUI 程序。第一 IDE 后端 220 可以作为 proxy 将 GUI 程序的 UI 通过显示服务器协议发送到 IDE 的客户端程序或浏览器的客户端程序。

第三环境的数量可以为一个，也可以为多个。一个第三环境中的 GUI 程序的数量可以为一个，也可以为多个。例如，第一 IDE 后端 220 可以用于通过显示服务器协议，将一个远程环境中的一个或多个 GUI 程序投射到 IDE 前端 210 所在的操作系统（如本地系统）中。再如，第一 IDE 后端 220 可以用于通过显示服务器协议，将多个远程环境中的多个 GUI 程序投射到 IDE 前端 210 所在的操作系统（如本地系统）中。

“第三环境”中的“第三”仅用于说明 GUI 程序所在的环境，不具有其他限定作用。

第三环境可以为同构系统，也可以为异构系统。

在第一 IDE 后端 220 运行于远程环境情况下，GUI 程序可以是运行在第一 IDE 后端 220 所在的环境中的 GUI 程序，也可以是运行于其他环境中的 GUI 程序。本申请实施例对此不做限定。示例性地，显示服务器协议可以包括以下任一项：X、Wayland 或远程桌面协议（remote desktop protocol，RDP）。例如，

X 协议可以为 X11 协议，即 X 协议的第 11 个版本。

具体地，通过 SSH 和显示服务器协议可以将远程环境中的 GUI 程序投射到 IDE 前端 210 所在的操作系统中。

以 X 协议为例，X 协议可以包括 X 服务端（sever）和 X 客户端（client）。X sever 用于显示屏幕画面，并将输入的动作通知 X client。X client 用于负责程序逻辑的处理。X client 即为 GUI 程序。X sever 部署于 IDE 前端 210。X-Server 可以运行在独立进程中，与投射的 GUI 程序相互独立。通过 SSH X11 转发（forward）即可实现将 GUI 程序投射到本地系统中，如图 8 所示。图 8 中本地系统中的 GUI 程序即为投射的 GUI 程序。

这样可以使得远程环境中的进程的图形用户界面可以输出到 IDE 前端所在的操作系统中。

图 7 示出了本申请实施例的一种应用投射的示意图。如图 7 所示，本申请实施例的方案可以实现应用投射，即将远程环境中的 GUI 程序投射到本地系统（即本地操作系统）中。

示例性地，可以通过 SSH 和显示服务器协议将远程环境中的 GUI 程序投射到本地系统中。

该远程环境可以为同构系统，也可以为异构系统。

如前所述，在一些实现方式中，第一 IDE 后端 220 可以部署于远程环境中。在该情况下，GUI 程序所在的远程环境与第一 IDE 后端 220 所在的远程环境可以为同一远程环境，也可以为不同的远程环境。

图 8 示出了两种场景下的应用投射的示意图。

在图 8 的（a）中，第一环境和第三环境为同一环境，即第一 IDE 后端 220 和 GUI 程序部署于同一环境中。在图 8 的（b）中，第一环境和第三环境为不同的环境，即第一 IDE 后端 220 和 GUI 程序部署于不同的环境中，该两个环境之间可以进行通信，以便实现应用投射。

以图 8 的（a）为例，后端服务 proxy 可以将 IDE 前端 210 的请求转发给第一 IDE 后端 220 中的 IDE 内核进行处理，然后将其响应返回给 IDE 前端 210。后端服务 proxy 还可以将 GUI 程序的 UI 通过显示服务器协议发送到本地系统。

应理解，图 8 中仅为示例，不对本申请实施例的方案构成限定。例如，图 8 中仅以投射一个远程环境中的一个 GUI 程序作为示例，在其他实现方式中，还可以投射其他数量的 GUI 程序，GUI 程序所在的远程环境也可以为其他数量。

IDE 前端和 IDE 后端之间可以通过多种方式进行通信。IDE 前端和 IDE 后端之间的通信方式与具体的开发模式相关。具体描述可以参见后文，此处不展开描述。

图 9 示出了本申请实施例的一种用于软件开发的方法的示意性流程图。图 9 所示的方法 600 可以由图 2 所示的系统 200 执行。方法 600 中的 IDE 前端可以为图 2 中的 IDE 前端 210。方法 600 中的第一 IDE 后端可以为图 2 中的第一 IDE 后端 220。具体描述可以参考第一 IDE 后端 220。

如图 9 所示，方法 600 包括如下步骤。

610，IDE 前端获取用户的配置信息，配置信息包括以下至少一项：硬件架构的类型或操作系统的类型。

620，IDE 前端根据配置信息在第一环境上部署第一 IDE 后端，以使得通过 IDE 前端实现在第一环境中的第一 IDE 后端上的软件开发。

步骤 620 也可以理解为，IDE 前端根据配置信息在第一环境部署第一 IDE 后端，以使得用户通过与 IDE 前端的交互能够实现在第一环境中的第一 IDE 后端上的软件开发，或者说，以使得通过对 IDE 前端的操作能够实现在第一环境中的第一 IDE 后端上的软件开发。

第一环境即第一 IDE 后端的部署环境。第一环境的类型是根据配置信息确定的。或者说，第一环境的类型是由配置信息指示的。或者说，第一环境为第一 IDE 后端支持的多种类型的环境中按照配置信息确定的环境。

如前所述，第一 IDE 后端的部署环境为可配置项。在步骤 610 中，可以获取部署环境的相关配置信息，从而确定第一 IDE 后端的部署环境。进而在符合该配置信息的环境（如第一环境）中部署第一 IDE 后端。

配置信息可以包括一个或多个硬件架构的类型，和/或，一个或多个操作系统的类型。本申请实施例对配置的环境的类型的数量不做限定。

本申请实施例中，第一 IDE 后端支持部署于多种类型的环境中。为了便于描述，后文中主要以第一 IDE 后端部署于远程环境中为例对方法 600 进行示例性说明。部署于其他环境中的第一 IDE 后端的方案

根据需要适应性修改即可。第一 IDE 后端也可以替换为第一 IDE 后端服务。

进一步地，方法 600 还可以包括步骤 611（图中未示出）。

611，IDE 前端根据配置信息获取第一环境。

在步骤 611 中可以获取第一 IDE 后端的部署环境，即待开发的程序所需要的环境。

第一环境可以包括一个或多个实例。该一个或多个实例也可以理解为一个或多个第一环境。即第一环境的数量可以为一个，也可以为多个。

步骤 611 也可以理解为 IDE 前端根据配置信息获取一个或多个实例。该一个或多个实例用于部署第一 IDE 后端。

示例性地，IDE 前端根据配置信息获取一个或多个实例，可以包括：IDE 前端根据配置信息创建一个或多个实例。

可替换地，IDE 前端根据配置信息获取一个或多个实例，可以包括：IDE 前端根据配置信息确定一个或多个实例。例如，云服务可以提供多个候选实例，并根据配置信息将一个或多个实例分配给该 IDE 前端。

一个或多个实例的获取可以通过多种方式触发。下面以一个或多个实例的创建为例进行说明。

示例性地，一个或多个实例的创建可以由 IDE 的启动触发。IDE 的启动也可以替换为 IDE 前端的启动。

即用户启动 IDE 的情况下，触发一个或多个实例的创建。用户启动 IDE 可以理解为，响应于用户的 IDE 启动请求，启动 IDE。

用户可以通过启动本地 IDE 的客户端程序来启动 IDE 前端。或者，用户可以从浏览器的客户端程序启动 IDE 前端。

可替换地，一个或多个实例的创建可以由用户的登录请求触发，或者说，由用户的登录操作触发。

该登录请求携带验证信息。该登录请求可以用于请求登录用户的账号。

以一个或多个实例部署于云计算服务上为例，用户在启动 IDE 后，可以登录账号。IDE 前端可以通过该账号连接到云计算服务，在云计算服务上创建一个或多个实例。

一个或多个实例的获取还可以由其他方式触发，本申请实施例对此不做限定。

对于不同的用户，为其获取的实例的类型可以相同，也可以不同。

示例性地，不同的用户可以通过用户的账号信息进行区分。

示例性地，该一个或多个实例可以包括以下任一项或多项：容器或虚拟机。

在一种可能的实现方式中，步骤 611 中可以获取多个实例。在该情况下，该多个实例对应的编程语言可以相同，也可以不同。

一个实例对应的编程语言可以理解为该实例支持的编程语言。或者说，该实例可以为该编程语言提供对应的运行时环境。该实例也可以称为该编程语言的实例。

例如，一个实例对应的编程语言为 Java，即该实例提供 Java 运行时。再如，一个实例对应的编程语言为 Python，即该实例提供 Python 运行时。

可选地，该一个或多个实例中的每个实例分别对应不同的编程语言。换言之，在步骤 611 中，可以获取一个或多个编程语言的实例。

该一个或多个实例可以包括第一 IDE 后端支持的部分或全部编程语言的实例。

第一 IDE 后端支持的编程语言可以理解为第一 IDE 后端包括该编程语言的语言特性。

以该一个或多个实例包括第一 IDE 后端支持的全部编程语言的实例为例，例如，第一 IDE 后端支持五种编程语言，在步骤 620 中，可以创建五个实例。该五个实例分别对应于该五种编程语言。

该一个或多个实例对应的编程语言可以通过多种方式确定的。

示例性地，该一个或多个实例对应的编程语言可以是预先设置的。

作为一种示例，预先设置的编程语言可以为系统默认的编程语言或用户预先配置的编程语言。例如，系统默认的编程语言包括 Java 和 Python，或者，用户预先配置编程语言包括 Java 和 Python，在步骤 611 中，可以创建 Java 的实例和 Python 实例。

可替换地，该一个或多个实例对应的编程语言可以是根据用户的相关信息确定的。

例如，可以根据用户的账号的历史记录确定用户之前使用的编程语言，从而创建相应的实例。

再如，可以根据用户的账号确定用户已存储的项目，基于该项目使用的编程语言创建相应的实例。

可替换地，该一个或多个实例对应的编程语言可以是由用户输入的。

例如，在启动 IDE 后，用户可以配置本次软件开发需要使用的编程语言包括 Java 和 Python，该配置信息可以携带在第一创建请求中。在步骤 611 中，可以创建 Java 的实例和 Python 实例。

该一个或多个实例对应的编程语言可以由步骤 610 中的配置信息指示，也可以由其他信息指示。

应理解，以上仅为示例，具体创建哪一种或哪几种编程语言的实例可以还通过其他方式确定。例如，该一个或多个实例对应的编程语言可以是随机选择的。本申请实施例对此不做限定。

该一个或多个实例的硬件架构的类型可以通过多种方式确定。

在一种可能的实现方式中，在步骤 611 中，可以创建多个实例。在该情况下，该多个实例的硬件架构的类型可以相同，也可以不同。

为了便于描述，下面先以一个硬件架构的类型为例进行说明。

示例性地，该一个或多个实例的硬件架构的类型可以是预先设置的。

作为一种示例，预先设置的硬件架构的类型可以为系统默认的硬件架构的类型或者用户预先配置的硬件架构的类型。例如，用户预先配置硬件架构的类型为鲲鹏，相应地，配置信息可以指示硬件架构的类型为鲲鹏。在步骤 611 中，可以在鲲鹏架构的硬件上创建该一个或多个实例。

可替换地，该一个或多个实例的硬件架构的类型可以是根据用户的相关信息确定的。

例如，可以根据用户的账号的历史记录确定用户之前使用的硬件架构的类型，从而创建相应的实例。

再如，可以根据用户的账号确定用户已存储的项目，基于该项目使用的硬件架构的类型创建相应的实例。

可替换地，该一个或多个实例的硬件架构的类型可以是由用户输入的。

例如，在启动 IDE 后，用户可以配置本次软件开发需要使用的硬件架构的类型为鲲鹏，相应地，配置信息可以指示硬件架构的类型为鲲鹏。在步骤 611 中，可以在鲲鹏架构的硬件上创建该一个或多个实例。

如上所述，该一个或多个实例的硬件架构的类型可以由配置信息指示。该配置信息的形式可以通过多种方式实现。例如，对于用户预先配置的硬件架构的类型，配置信息可以为该硬件架构的类型的标识。或者，配置信息也可以为空字段，若硬件架构的类型的配置信息为“空”，则系统按照之前配置的硬件架构的类型创建实例。

应理解，以上仅为示例，具体使用的硬件架构的架构可以还通过其他方式确定。本申请实施例对此不做限定。

如前所述，在步骤 620 中，可以创建多个实例，若涉及多个硬件架构的类型，即需要在多个类型的硬件架构上创建实例，各个硬件架构创建实例的方式可以参考前文。下面举例说明。

示例性地，该多个硬件架构的类型中的每个类型的硬件架构上创建的实例可以包括第一 IDE 后端支持的全部编程语言的实例。例如，第一 IDE 后端支持五种编程语言，在步骤 611 中，可以在每个类型的硬件架构上创建五个实例。该五个实例分别对应于该五种编程语言。

示例性地，该多个硬件架构的类型中的每个类型的硬件架构上创建的实例对应的编程语言可以是预先设置的。

例如，系统默认的编程语言包括 Java 和 Python，或者，用户预先配置编程语言包括 Java 和 Python，在步骤 611 中，可以在每个类型的硬件架构上创建 Java 的实例和 Python 实例。

示例性地，该多个硬件架构的类型中的每个类型的硬件架构上创建的实例对应的编程语言可以是由用户配置的。

例如，用户可以配置本次软件开发需要使用的硬件架构的类型为鲲鹏 ARM，鲲鹏上的实例对应的编程语言为 Java，ARM 上的实例对应的编程语言为 Python。在步骤 611 中，可以在鲲鹏上创建支持 Java 的实例，在 ARM 上创建支持 Python 实例。

以上仅为示例，其他实现方式可以参考前文，此处不再赘述。

在一种可能的实现方式中，在步骤 620 中，可以创建多个实例。在该情况下，该多个实例的操作系统可以相同，也可以不同。

为了便于描述，下面先以一个操作系统为例进行说明。

示例性地，该一个或多个实例的操作系统的类型可以是预先设置的。

作为一种示例，预先设置的操作系统的类型可以为系统默认的操作系统的类型或者用户预先配置的

操作系统的类型。例如，用户预先配置操作系统的类型为欧拉 OS，相应地，配置信息可以指示操作系统的类型为欧拉 OS。在步骤 611 中，可以创建一个或多个实例，该一个或多个实例的操作系统可以为欧拉 OS。

可替换地，该一个或多个实例的操作系统的类型可以是由用户输入的。

例如，在启动 IDE 后，用户可以配置本次软件开发需要使用的操作系统的类型为欧拉 OS，相应地，配置信息可以指示操作系统的类型为欧拉 OS。在步骤 611 中，可以创建一个或多个实例，该一个或多个实例的操作系统可以为欧拉 OS。

应理解，以上仅为示例，具体使用的操作系统的类型可以还通过其他方式确定。本申请实施例对此不做限定。

如前所述，在步骤 620 中，可以创建多个实例，若涉及多个操作系统的类型，基于各个操作系统创建实例的方式可以参考前文。该多个实例对应的编程语言、操作系统的类型以及硬件架构的类型均可以根据需要设置，例如由用户配置或系统默认配置。

例如，用户可能需要在多种类型的硬件架构上的多种类型的操作系统中开发多种编程语言的应用程序，在该情况下，可以为用户在该多种类型的硬件架构上创建基于该多种类型的操作系统的支持该多种编程语言的多个实例。具体描述可以参考前文，此处不再赘述。该多个实例对应的编程语言、操作系统的类型以及硬件架构的类型均可以由配置信息指示。不同项的配置信息的获取方式可以相同，也可以不同。例如，操作系统的类型是由用户指示的，即操作系统的配置信息是基于用户的输入确定的。再如，硬件架构的类型是系统默认类型，即硬件架构的配置信息可以是从系统中获取的。

可选地，配置信息由携带验证信息的登录请求指示。例如，用户在登录时，可以配置硬件架构的类型和/或操作系统的类型。相应的配置信息可以由登录请求指示。

以上仅为示例，本申请实施例对配置信息的获取方式不做限定。

步骤 620 可以包括如下情况。

如前所述，IDE 前端可以根据配置信息获取一个或多个实例。该一个或多个实例中可能已经部署了第一 IDE 后端，也可能未部署第一 IDE 后端。

若该一个或多个实例中已经部署了第一 IDE 后端，则获取一个或多个实例的过程也可以视为步骤 620。

若该一个或多个实例中的部分或全部实例中未部署第一 IDE 后端，则步骤 620 还可以包括：在未部署第一 IDE 后端的实例中部署第一 IDE 后端。

示例性地，IDE 前端可以将第一 IDE 后端的安装包发送到未部署第一 IDE 后端的实例中，以实现第一 IDE 后端在相应实例中的部署。

进一步地，方法 600 还可以包括步骤 630。

630，IDE 前端启动第一实例中的第一 IDE 后端。第一实例属于步骤 611 中获取的一个或多个实例。

第一实例中的第一 IDE 后端被启动后，可以提供第一实例支持的第一编程语言的语言特性。

或者说，IDE 前端启动第一实例中的第一 IDE 后端，以使得该第一 IDE 后端提供第一实例支持的第一编程语言的语言特性。

本申请实施例中，第一 IDE 后端可以用于提供多种编程语言的语言特性，第一 IDE 后端被启动后，相应的语言特性可以被激活。启动对应的实例中的第一 IDE 后端也可以视为激活对应的语言服务中的一个步骤。

在一种实现方式中，在步骤 630 中，可以启动所有实例的第一 IDE 后端。

在另一种实现方式中，在步骤 630 中，可以启动特定的实例（如第一实例）的第一 IDE 后端。

可选地，步骤 630 可以包括：根据第一操作启动第一实例中的第一 IDE 后端。

第一操作也可以视为激活操作。

在一种实现方式中，第一操作可以为作用于第一项目的操作。

第一项目为任意软件开发项目。“第一项目”中的“第一”仅为描述方便使用，不具有限定作用。

在一种可能的实现方式中，第一项目可以为本次启动 IDE 后创建的项目。

在该情况下，方法 600 还可以包括步骤 631。

631，创建第一项目。

示例性地，第一项目可以是基于项目模板创建的。例如，第一项目可以为多语言项目，用户可以通过系统提供的对应的多语言项目的模板创建第一项目。用户创建第一项目可以理解为，响应于用户的项

目创建请求，创建第一项目，该项目创建请求指示创建第一项目。

在另一种可能的实现方式中，第一项目可以为本次启动 IDE 后打开的已存储的项目。

在该情况下，方法 600 还可以包括步骤 632。

632，打开第一项目。

用户在启动 IDE 后可以打开之前存储的第一项目。用户打开第一项目可以理解为，响应于用户的项目打开请求，打开第一项目，该项目打开请求指示打开第一项目。

语言服务由第一 IDE 后端提供。在本申请实施例的方案中，不同的编程语言的语言服务均由统一的第一 IDE 后端提供，即不同的编程语言的语言服务由相同架构的 IDE 内核提供。第一 IDE 后端的具体描述可以参考图 2，此处不再赘述。

语言服务也可以称为语言服务器，用于提供编程支持，或者说，用于支持语言特性，例如，代码补全、代码格式化、代码诊断、跳转、悬停、高亮显示以及类型推断等。第一 IDE 后端用于提供多种编程语言的语言特性也可以替换为第一 IDE 后端用于提供多种编程语言的语言服务。

示例性地，语言服务可以基于当前 IDE 前端的状态，例如，光标当前在 IDE 前端中的位置或鼠标当前悬停的位置等，提供编程支持，例如提供一组操作或指令等。

一种编程语言的语言服务可以用于提供针对该编程语言的编程支持，即在基于该编程语言进行软件开发时提供的编程支持。

语言服务和 IDE 前端之间可以通过 JavaScript（JS）对象简谱（JavaScript object notation，JSON）远程过程调用（JSON remote procedure call，JSON-RPC）进行通信。

IDE 前端可以将用户执行的操作通知给语言服务。例如，IDE 前端可以将用户执行的代码操作以请求的形式发送至语言服务。语言服务可以对其进行处理，并以响应的形式将处理结果返回给 IDE 前端。

在步骤 630 中，第一实例即为支持第一编程语言的实例。该实例中的第一 IDE 后端即用于提供该编程语言的语言服务。步骤 630 也可以理解为，激活与第一编程语言的语言服务。

第一操作可以用于指示第一编程语言。例如，第一编程语言可以通过第一操作时的上下文确定。

步骤 630 也可以理解为根据第一操作时的上下文激活对应的语言服务。

“第一操作”中的“第一”仅用于限定该操作与激活第一 IDE 后端相关，不具有其他限定作用。

下面对第一操作进行示例性说明。下面以对第一操作是基于第一项目的操作为例进行示例性说明。

示例性地，对第一项目的第一操作可以为对第一项目的创建操作。

相应地，对第一项目的第一操作可以为步骤 631。

用户在创建第一项目时，系统可以激活与第一操作相关的编程语言的语言服务。例如，系统可以激活第一项目中的所有编程语言的语言服务。

示例性地，对第一项目的第一操作可以为对第一项目的打开操作。

相应地，对第一项目的第一操作可以为步骤 632。

例如，用户在首次打开第一项目时，系统可以激活与第一操作相关的编程语言的语言服务。例如，系统可以激活第一项目中的所有编程语言的语言服务。

在一种可能的实现方式中，第一操作可以为对第一项目中的文件的操作。

示例性地，第一操作可以为对第一项目中的第一文件的打开操作。

例如，用户在首次打开第一文件时，系统可以激活与第一操作相关的编程语言的语言服务。用户打开第一文件可以理解为，响应于用户的文件打开请求，打开第一文件，该文件打开请求指示打开第一文件。

第一文件可以为代码文件，也可以为非代码文件。

“第一文件”中的“第一”仅为了描述方便使用，不具有限定作用。

以第一文件为代码文件为例，例如，第一文件为第一代码文件，第一操作可以为对第一代码文件的打开操作，第一编程语言可以为第一代码文件的编程语言。用户在首次打开第一代码文件时，可以激活第一代码文件所使用的编程语言的语言服务。例如，第一代码文件为 Java 文件，用户在打开 Java 文件时，可以激活 Java 的语言服务。

以第一文件为非代码文件为例，用户在首次打开第一文件时，可以激活第一项目中的所有编程语言的语言服务。例如，第一文件可以为第一项目中被打开的第一个文件，即用户在第一项目中打开第一个文件时，可以激活第一项目中的所有编程语言的语言服务。

根据第一操作激活对应的语言服务可以是响应于第一操作，立即激活对应的语言服务。或者，根据第一操作激活对应的语言服务也可以是响应于第一操作，一段时间后激活对应的语言服务。本申请实施例对此不做限定。

应理解，以上仅为第一操作的示例，本申请实施例对第一操作的具体内容不做限定。

激活实例中的第一 IDE 后端可以包括启动实例中的第一 IDE 后端。

在激活第一实例中的第一 IDE 后端的过程中，可以先检查第一实例中是否部署有第一 IDE 后端。

例如，IDE 前端可以检查该第一实例中是否部署有第一 IDE 后端。

可选地，激活第一实例中的第一 IDE 后端，可以包括，在第一实例中部署有第一 IDE 后端的情况下，启动该第一 IDE 后端。

可选地，激活第一实例中的第一 IDE 后端，可以包括，在第一实例中未部署第一 IDE 后端的情况下，在第一实例中部署第一 IDE 后端，并启动该第一 IDE 后端。

进一步地，方法 600 还可以包括根据第二操作关闭第一 IDE 后端。

或者说，根据第二操作停止第一 IDE 后端服务，也可以理解为，停止与第二操作相关的编程语言的语言服务。

“第二操作”中的“第二”仅用于限定第二操作与关闭第一 IDE 后端相关，不具有其他限定作用。

具体关闭哪些第一 IDE 后端可以根据需要设置。例如，被关闭的第一 IDE 后端与第二操作相关。

下面对第二操作进行示例性说明。

示例性地，第二操作可以作用于第一项目。

例如，对第一项目的第二操作可以包括对第一项目的关闭操作。

用户在关闭第一项目时，系统可以关闭第一 IDE 后端。例如，系统可以关闭所有已启动的第一 IDE 后端。用户关闭第一项目可以理解为，响应于用户的项目关闭请求，关闭第一项目，该项目关闭请求指示关闭第一项目。

示例性地，第二操作可以作用于 IDE 前端。

例如，对 IDE 前端的第二操作可以包括对 IDE 前端的关闭操作。

用户关闭 IDE 前端时，系统可以关闭第一 IDE 后端。例如，系统可以关闭所有已启动的第一 IDE 后端。用户关闭 IDE 前端可以理解为，响应于用户的 IDE 关闭请求，关闭 IDE 前端，该 IDE 关闭请求指示关闭 IDE 前端。

根据第二操作关闭第一 IDE 后端，可以为，响应于第二操作，立即关闭第一 IDE 后端。或者，根据第二操作关闭第一 IDE 后端，也可以为，响应于第二操作，一段时间后关闭第一 IDE 后端。本申请实施例对此不做限定。

应理解，以上仅为第二操作的示例，本申请实施例对第二操作的具体内容不做限定。

此外，方法 600 可以应用于远程开发的场景中，即第一 IDE 后端部署于远端环境中。

在该情况下，在启动第一实例中的第一 IDE 后端之前，IDE 前端可以与第一实例建立连接。例如，IDE 前端可以与第一实例建立安全外壳协议（secure shell protocol，SSH）连接。

在该情况下，激活实例中的第一 IDE 后端还可以包括建立 IDE 前端和该第一 IDE 后端之间的长连接，以使得该第一 IDE 后端可以随时处于待命状态，从而提高交互的效率。

示例性地，在启动第一实例中的第一 IDE 后端后，可以将该第一 IDE 后端监测的远程端口发送至 IDE 前端。IDE 前端可以建立本地端口和远程端口之间的 SSH 隧道（本地端口：远程端口），并将超文本传输协议（hypertext transfer protocol，HTTP）连接升级为 Websocket 连接。后续的服务请求可以通过该连接进行交互。

此外，在关闭第一 IDE 后端的情况下，还可以关闭 SSH 隧道。

在步骤 630 中，在激活对应的语言服务后，即可由对应的语言服务进行相应的处理。具体地，用户的代码操作，例如，编辑、调试、测试、编译以及构建等，可以以请求的方式由 IDE 前端发送至对应的语言服务进行处理。语言服务可以将处理结果返回给 IDE 前端，由 IDE 前端将处理结果展示给用户。

可选地，方法 600 可以包括：IDE 前端将第一请求发送至第一环境中的第一 IDE 后端；第一环境中的第一 IDE 后端生成对第一请求的第一响应；第一环境中的第一 IDE 后端将第一响应返回至 IDE 前端。

在该情况下，IDE 前端和第一环境中的第一 IDE 后端配合，以实现在第一 IDE 后端所在的系统中的软件开发。即软件开发的具体操作可以由第二 IDE 后端完成。第一请求可以是基于用户的代码操作生成

的。

可选地，IDE还可以包括运行于第二环境的第二IDE后端。在该情况下，该方法600还可以包括：

IDE前端将第二请求发送至第一环境中的第一IDE后端；

第一环境中的第一IDE后端将第二请求转发至第二环境中的第二IDE后端；

第二环境中的第二IDE后端生成对第二请求的第二响应；

第二环境中的第二IDE后端将第二响应发送至第一环境中的第一IDE后端；

第一环境中的第一IDE后端将第二响应返回至IDE前端。

IDE前端、第一IDE后端和第二IDE后端配合，以实现第二IDE后端所在的系统中的软件开发。即软件开发的具体操作可以由第二IDE后端完成。第二请求可以是基于用户的代码操作生成的。

第二IDE后端的相关描述可以参考前文，此处不再赘述。

进一步地，方法600还可以包括：第一IDE后端可以通过显示服务器协议将远程环境中运行的图形用户界面程序投射到IDE前端所在的本地系统。

应用投射的描述可以参考前文，此处不再赘述。

进一步地，方法600还可以包括步骤640。

640，销毁该一个或多个实例。

可选地，步骤640可以包括：根据第三操作销毁该一个或多个实例。

“第三操作”中的“第三”仅用于限定第三操作与销毁实例相关，不具有其他限定作用。

下面对第三操作进行示例性说明。

示例性地，第三操作可以作用于IDE前端。

例如，对IDE前端的第三操作可以包括对IDE前端的关闭操作。

用户关闭IDE前端的情况下，系统可以销毁该一个或多个实例。例如，系统可以销毁与该IDE前端相关的所有实例。

第三操作和第二操作可能为同一操作，也可能为不同的操作。例如，第三操作和第二操作可以为对IDE前端的关闭操作。即在用户关闭IDE前端的情况下，系统可以关闭第一IDE后端，销毁与该IDE前端相关的所有实例。再如，第三操作可以为对IDE前端的关闭操作，第二操作可以为对第一项目的关闭操作。即在用户关闭第一项目的情况下，系统可以关闭第一IDE后端，在用户关闭IDE前端的情况下，系统可以销毁与该IDE前端相关的所有实例。

应理解，以上仅为第三操作的示例，本申请实施例对第三操作的具体内容不做限定。

根据第三操作销毁该一个或多个实例，可以为，响应于第三操作，立即销毁该一个或多个实例。或者，根据第三操作销毁该一个或多个实例，也可以为，响应于第三操作，一段时间后销毁该一个或多个实例。本申请实施例对此不做限定。

图10至图17示出了本申请实施例的方案应用于不同的开发场景下的示意图。图10至图17中IDE架构可以为图2所示的IDE架构。该IDE架构的具体描述可以参考图2，为避免重复，在描述图10至图17时适当省略部分描述。图10至图17所示的方案中，本地系统也可以理解为本地操作系统。

为了便于描述，在图10至图17的方案中，IDE前端表示为前端UI。用户可以与前端UI进行交互。

图10示出了本申请实施例的方案应用于本地IDE开发场景下的示意图。在本地IDE开发场景中，IDE前端和IDE后端（如图10的第一IDE后端）运行于同一本地操作系统上。本地IDE开发场景也可以替换为桌面IDE开发场景。

在涉及本地IDE的开发场景中，前端UI运行于本地IDE的客户端程序中，本地IDE的客户端程序部署于本地系统中。本地IDE的客户端程序也可以替换为桌面IDE的客户端程序。

具体地，在如图10所示的开发场景中，第一IDE后端也部署于该本地系统中。第一IDE后端所在的第一环境即为IDE前端所在的本地系统。即IDE前端和IDE后端运行于同一本地操作系统中。

在该场景下，IDE前端和第一IDE后端之间的通信可以采用进程间的通信方式实现。例如，如图10所示，IDE前端和第一IDE后端之间可以通过进程间通信（inter-process communication，IPC）来进行交互。

图11示出了本申请实施例的方案应用于WebIDE开发场景下的示意图。在图11所示的开发场景下，用户可以通过本地浏览器访问后端服务器，以实现软件开发。该后端服务器上部署有IDE后端（如图11所示的第一IDE后端）。或者，也可以理解为，该后端服务器即用于提供IDE后端服务，也可以称为

IDE 后端服务器。该 IDE 后端也可以称为 WebIDE 的 IDE 后端。

在涉及 WebIDE 的开发场景中，IDE 前端运行于浏览器的客户端程序中，如图 11 所示。浏览器的客户端程序部署于本地系统中。本地浏览器即部署于本地系统中的浏览器的客户端程序。

第一 IDE 后端所在的环境可以为本地的后端服务器，也可以为远端的后端服务器。即第一 IDE 后端可以部署于本地的后端服务器，也可以部署于远端的后端服务器。相应地，用户可以通过本地浏览器访问 IDE 后端，以便在本地或远端实现软件开发。

为了便于描述，本申请实施例的方案中，本地的后端服务器和远端的后端服务器均可以视为远程环境。

如图 11 所示，第一环境所属的系统可以为同构系统，也可以为异构系统。即第一 IDE 后端可以部署于同构系统，也可以部署于异构系统。即用户可以通过本地浏览器访问该 IDE 后端，以实现同构系统或异构系统中的软件开发。实现同构系统或异构系统中的软件开发，也可以替换为，开发同构系统或异构系统中的应用程序。

第一 IDE 后端可以部署于多种类型的计算资源上。示例性地，第一 IDE 后端可以部署于容器或虚拟机上。例如，远程环境的计算资源可以包括以下任一项或多项：本地容器、本地虚拟机、远端容器或远端虚拟机。第一 IDE 后端可以部署于上述任一项中。

在图 11 所示的开发场景下，IDE 前端和第一 IDE 后端之间的通信可以采用浏览器和后端服务器之间的通信方式实现。例如，如图 11 所示，IDE 前端和第一 IDE 后端之间可以通过 HTTP 来进行交互。

图 12 示出了本申请实施例的方案应用于另一种 WebIDE 开发场景下的示意图。图 11 和图 12 的主要区别在于，图 12 中的第一 IDE 后端运行于浏览器的客户端程序中。换言之，第一 IDE 后端的主进程（host process）可以为浏览器进程，IDE 前端和第一 IDE 后端可以运行在不同的浏览器进程中。

在图 12 所示的开发场景下，IDE 前端和第一 IDE 后端之间的通信可以采用进程之间的通信方式实现。例如，如图 12 所示，IDE 前端和第一 IDE 后端之间可以通过 IPC 来进行交互。

图 13 示出了本申请实施例的方案应用于本地 IDE 在远程开发的开发场景下的示意图。用户可以通过本地 IDE 的客户端程序访问后端服务器，以实现软件开发。该后端服务器上部署有 IDE 后端（如图 13 中的第一 IDE 后端）。

在涉及本地 IDE 的开发场景中，前端 UI 运行于本地 IDE 的客户端程序中，本地 IDE 的客户端程序部署于本地系统中。

第一环境可以为本地的后端服务器，也可以为远端的后端服务器。即第一 IDE 后端可以部署于本地的后端服务器，也可以部署于远端的后端服务器。相应地，用户可以通过本地 IDE 的客户端程序访问 IDE 后端，以便在本地或远端实现软件开发。

如图 13 所示，第一环境所属的系统可以为同构系统和/或异构系统。即第一 IDE 后端可以部署于同构系统，也可以部署于异构系统。即用户可以通过本地 IDE 的客户端程序访问该 IDE 后端，以实现同构系统或异构系统中的软件开发。

第一 IDE 后端可以部署于多种类型的计算资源上。示例性地，第一 IDE 后端可以部署于容器或虚拟机上。例如，远程环境的计算资源可以包括以下任一项或多项：本地容器、本地虚拟机、远端容器或远端虚拟机。第一 IDE 后端可以部署于上述任一项中。

在图 13 所示的开发场景下，IDE 前端和第一 IDE 后端之间的通信可以采用客户端和服务器之间的通信方式实现。例如，如图 13 所示，IDE 前端和第一 IDE 后端之间可以通过 SSH 隧道来进行交互，IDE 前端和对应的语言服务之间可以基于 JSON RPC 进行通信。具体描述可以参考图 9 所示的方法 600。

图 14 示出了本申请实施例的方案应用于 WebIDE 在远程开发的开发场景下的示意图。在涉及 WebIDE 的开发场景中，前端 UI 运行于浏览器的客户端程序中，浏览器的客户端程序部署于本地系统中。

在图 14 所示的开发场景中，用户可以通过本地浏览器访问远程环境#1（第一环境的一例）中的第一 IDE 后端，并通过远程环境#1 中的第一 IDE 后端与远程环境#2（第二环境的一例）上的第二 IDE 后端进行交互，以便在远程环境#2 中进行软件开发。远程环境#1 的第一 IDE 后端即为 WebIDE 的 IDE 后端。远程环境#1 和远程环境#2 为不同的远程环境。即用户可以从本地浏览器的 IDE 前端在 WebIDE 的 IDE 后端所在的远程环境以外的其他远程环境中开发应用程序。

如图 14 所示，IDE 后端可以部署于不同的远程环境中。

第一环境所属的系统可以为同构系统和/或异构系统。即第一 IDE 后端可以部署于同构系统，也可以

部署于异构系统。第二环境所属的系统可以为同构系统和/或异构系统。即第二 IDE 后端可以部署于同构系统，也可以部署于异构系统。即用户可以从本地浏览器的 IDE 前端实现远程环境#2 的同构系统或异构系统中的软件开发。

IDE 后端可以部署于多种类型的计算资源上。具体描述可以参考前文，此处不再赘述。

下面对图 14 所示的开发场景进行示例性说明。

例如，用户可以通过本地浏览器登录云账号，运行于本地浏览器中的 IDE 前端通过云账号连接云服务器上的计算服务，创建服务实例。用户预先配置的硬件架构的类型可能为类型#1，在该情况下，即在类型#1 的硬件上创建服务实例，并启动该服务实例上的 IDE 后端（第一 IDE 后端的一例），并进行远程环境#1（第一环境的一例）中的软件开发。该 IDE 后端即为远程环境#1 上的 IDE 后端。在开发过程中，用户可能需要在其他硬件架构的类型（如类型#2）的硬件上进行软件开发，在该情况下，采用本申请实施例的方案，远程环境#1 上的 IDE 后端可以与远程环境#2（第二环境的一例）建立连接，远程环境#2 的硬件包括类型#2 的硬件。在类型#2 的硬件上创建服务实例，并启动该服务实例上的 IDE 后端（第二 IDE 后端的一例），从而实现远程环境#2 中的软件开发。

在图 14 所示的开发场景下，IDE 前端和 WebIDE 的 IDE 后端之间的通信可以采用浏览器和服务器之间的通信方式，WebIDE 的 IDE 后端和其他远程环境中的 IDE 后端可以采用服务器之间的通信方式。

例如，如图 14 所示，在软件开发过程中，IDE 前端和 WebIDE 的 IDE 后端之间可以通过 HTTP 进行通信，WebIDE 的 IDE 后端之间和其他远程环境中的 IDE 后端之间可以通过 SSH 隧道来进行交互，如基于 JSON RPC 进行交互。

如前文的方法 600 所述，用户的代码活动，例如，编辑、调试、测试、编译、构建等，可以以请求的方式由 IDE 前端发送至对应的语言服务进行处理，语言服务可以将处理结果返回给 IDE 前端，由 IDE 前端将处理结果展示给用户。在图 14 所示的方案中，语言服务由远程环境#2 中的 IDE 后端（图 14 中的第二 IDE 后端）提供。具体地，IDE 前端可以通过 HTTP 请求将用户的代码活动的请求发送至 WebIDE 的 IDE 后端（图 14 中的第一 IDE 后端），由 WebIDE 的 IDE 后端通过 WebIDE 的 IDE 后端与远程环境#2 建立的 SSH 隧道，将该请求转发至远程环境#2 的 IDE 后端，即落地到该远程环境的 IDE 后端组件中，从而使得用户可以在本地浏览器的 IDE 前端开发远程环境#2 的同构或异构系统中的应用程序。

图 15 至图 17 示出了本申请实施例的三种应用投射的示意图。图 15 至图 17 中本地系统中的 GUI 程序 A 至 GUI 程序 D 即为投射的 GUI 程序 A 至 GUI 程序 D。

图 15 示出了本申请实施例的方案应用于本地 IDE 开发场景下的示意图。图 15 的方案和图 10 的方案的主要区别在于，图 15 所示的方案中涉及应用投射。为避免重复，在描述图 15 所示的方案时主要说明其与图 10 的区别之处，其他描述可以参考图 10 的方案。

第一 IDE 后端可以与一个或多个远程环境建立连接，以使得该一个或多个远程环境中的 GUI 程序可以投射到本地系统中。

示例性地，第一 IDE 后端可以与该一个或多个远程环境分别建立 SSH 隧道，以使得该一个或多个远程环境中的 GUI 程序可以投射到本地系统中。

该一个或多个远程环境中的 GUI 程序的数量可以为一个，也可以为多个。例如，如图 15 所示，第一 IDE 后端与两个远程环境分别建立 SSH 隧道，以使得该两个远程环境中的 4 个 GUI 程序（即 GUI 程序 A、GUI 程序 B、GUI 程序 C 和 GUI 程序 D）可以投射到本地系统中，该 4 个 GUI 程序可以同时显示在前端 UI 中。

应理解，图 15 中的 GUI 程序所在的远程环境的数量以及每个远程环境中的 GUI 程序的数量仅为示例，不对本申请实施例的方案构成限定。

图 16 示出了本申请实施例的方案应用于 WebIDE 开发场景下的示意图。图 16 的方案和图 11 的方案的主要区别在于，图 16 所示的方案中涉及应用投射。为避免重复，在描述图 16 所示的方案时主要说明其与图 11 的区别之处，其他描述可以参考图 11 的方案。

图 16 第一 IDE 后端部署于第一环境中，在图 16 所示的方案中，第一环境和第三环境不同。第一 IDE 后端可以与一个或多个第三环境建立连接，以使得该一个或多个第三环境中的 GUI 程序可以投射到本地系统中。

示例性地，第一 IDE 后端可以与该一个或多个第三环境建立 SSH 隧道，以使得该一个或多个第三环境中的 GUI 程序可以投射到本地系统中。

如前所述，用户可以通过本地浏览器访问部署在第一环境中的第一 IDE 后端。示例性地，本地浏览器可以与第一环境建立 SSH 隧道，即 SSH 隧道浏览器（SSH-in-browser），第一环境与第三环境建立 SSH 隧道，以使得第三环境中的 GUI 程序可以投射到本地系统中。

该一个或多个第三环境中的 GUI 程序的数量可以为一个，也可以为多个。例如，如图 16 所示，用户可以通过本地浏览器访问部署在两个第一环境中的第一 IDE 后端。本地浏览器分别与该两个第一环境建立 SSH 隧道，该两个第一环境分别与两个第三环境建立 SSH 隧道，通过显示服务器协议使得该两个第三环境中的 4 个 GUI 程序可以投射到本地系统中，该 4 个 GUI 程序可以同时显示在前端 UI 中。

应理解，图 16 中的 GUI 程序所在的远程环境的数量以及每个远程环境中的 GUI 程序的数量仅为示例，不对本申请实施例的方案构成限定。

图 17 示出了本申请实施例的方案应用于本地 IDE 在远程开发的开发场景下的示意图。图 17 的方案和图 13 的方案的主要区别在于，图 17 所示的方案中涉及应用投射。为避免重复，在描述图 17 所示的方案时主要说明其与图 13 的区别之处，其他描述可以参考图 13 的方案。

第一 IDE 后端部署于第一环境中，在图 17 所示的方案中，第一环境和第三环境不同。图 17 第一 IDE 后端可以与一个或多个第三环境建立连接，以使得该一个或多个第三环境中的 GUI 程序可以投射到本地系统中。

示例性地，第一 IDE 后端可以与该一个或多个第三环境建立 SSH 隧道，以使得该一个或多个第三环境中的 GUI 程序可以投射到本地系统中。

用户可以通过本地 IDE 的客户端程序访问第一环境中的后端服务器，以实现软件开发。示例性地，本地 IDE 的客户端程序可以与第一环境建立 SSH 隧道，第一环境与第三环境建立 SSH 隧道，以使得第三环境中的 GUI 程序可以投射到本地系统中。

该一个或多个第三环境中的 GUI 程序的数量可以为一个，也可以为多个。例如，如图 17 所示，用户可以通过本地 IDE 的客户端程序访问部署在两个第一环境中的第一 IDE 后端。本地 IDE 的客户端程序分别与该两个第一环境建立 SSH 隧道，该两个第一环境分别与两个第三环境建立 SSH 隧道，通过显示服务器协议使得该两个第三环境中的 4 个 GUI 程序可以投射到本地系统中，该 4 个 GUI 程序可以同时显示在前端 UI 中。

应理解，图 17 中的 GUI 程序所在的远程环境的数量以及每个远程环境中的 GUI 程序的数量仅为示例，不对本申请实施例的方案构成限定。

图 18 示出了本申请实施例的一种用于软件开发的方法的流程图。图 18 所示的方法可以视为图 9 所示的方法 600 的一种具体实现方式。具体描述可以参考方法 600，为避免重复，在描述方法 700 时适当省略部分描述。方法 700 中的 IDE 采用图 2 所示的 IDE 架构。即方法 700 中的 IDE 后端服务可以为图 2 中的第一 IDE 后端 220 和/或图 6 中的第一 IDE 后端，方法 700 中的 IDE 前端可以为图 2 中的 IDE 前端 210。方法 700 中的服务实例即为方法 600 中的实例。

图 18 所示的方法 700 可以应用于本地 IDE 在远程开发的开发场景，例如，图 13 所示的开发场景。

示例性地，图 18 所示的方法 700 可以应用于本地设备结合云服务器的场景中，或者说，方法 700 可以应用于本地设备结合云计算服务的场景中，即第一 IDE 后端 220 可以部署于云计算服务上。

如图 18 所示，方法 700 可以包括如下步骤。

710，启动 IDE。

用户启动本地 IDE 的客户端程序，即启动 IDE 前端，或者说启动 IDE。

IDE 前端可以通过配置的账号自动连接云计算服务，创建服务实例。

示例性地，用户启动 IDE 后，可以显示登录界面，提示用户进行账号登录。在用户登录后，IDE 前端可以通过用户的账号连接云计算服务，为用户创建各种编程语言的服务实例。例如，如图 18 所示，为用户创建 Java 的服务实例和 Python 的服务实例。

应理解，图 18 所示的方案中的服务实例由容器实现仅为示例，不对本申请实施例的方案构成限定。例如，在其他实现方式中，服务实例可以由虚拟机实现。

应理解，图 18 所示的方案中的服务实例的数量以及对应的编程语言仅为示例，不对本申请实施例的方案构成限定。在其他实现方式中，还可以创建其他数量的服务实例和/或其他编程语言的服务实例。

其中，服务实例的硬件架构的类型可以由用户预先配置。例如，用户预先配置服务实例的硬件架构的类型为硬件类型#1，则基于硬件类型#1 的硬件架构为用户创建服务实例。

720，创建项目。

示例性地，用户可以通过IDE提供的项目模板，创建项目。

以多语言项目为例，用户可以通过 IDE 提供的项目模板，创建多语言项目。例如，该多语言项目涉及的编程语言为 Java 和 Python。

730，获取激活操作（第一操作的一例）。

在 IDE 前端获取到用户的激活操作时，可以激活对应的语言服务。

用户执行某些操作时，IDE 可以根据当前上下文自动激活对应的语言服务，即实现上下文感知的语言服务激活，这些操作即为激活操作。

示例性地，激活操作可以为打开项目中的代码文件，即语言文件。例如，用户首次打开 Java 文件时，IDE 可以基于当前的上下文激活 Java 的语言服务。再如，用户首次打开 Python 文件时，IDE 可以基于当前的上下文激活 Python 的语言服务。

下面以激活 Java 的语言服务为例对激活过程进行示例性说明。例如，在用户创建了包括 Java 和 Python 的多语言项目后，打开该项目中的 Java 文件，可以触发 Java 的语言服务的激活过程。激活过程可以包括如下步骤。

731，IDE 前端与 Java 的服务实例建立 SSH 连接。

732，启动 Java 的服务实例中的 IDE 后端服务，并将该 IDE 后端服务监测的远程端口发送至 IDE 前端。

具体地，IDE 前端检查该服务实例中的容器（即 Java 容器）是否部署有 IDE 后端服务。若是，则启动 IDE 后端服务。若否，则在该容器中部署 IDE 后端服务，并启动 IDE 后端服务。

将该 IDE 后端服务监测的远程端口发送至 IDE 前端。

733，IDE 前端建立本地端口和远程端口之间的 SSH 隧道（本地端口：远程端口），将 HTTP 连接升级为 Websocket 连接。这样，之后的服务请求均可以通过该连接进行交互。

若用户当前不需要 IDE 后端服务，则可以停止 IDE 后端服务。停止 IDE 后端服务的触发方式可以根据需要设置。

示例性地，若项目关闭，则可以停止与该项目相关的 IDE 后端服务。例如，用户创建了包括 Java 和 Python 的多语言项目后，执行了激活操作，使得 Java 的语言服务和 Python 的语言服务被激活，即用于提供 Java 的语言服务和 Python 的语言服务的 IDE 后端服务被启动。若用户关闭了该语言项目，则可以停止用于提供 Java 的语言服务和 Python 的语言服务的 IDE 后端服务。

示例性地，若 IDE 前端关闭，则可以停止与该 IDE 前端交互的所有 IDE 后端服务。

进一步地，还可以关闭 SSH 隧道。

740，获取代码操作。

在 IDE 前端获取到用户的代码操作时，可以将用户的代码操作以服务请求的方式发送至语言服务进行处理，语言服务处理后将处理结果返回给 IDE 前端，由 IDE 前端显示给用户。

750，关闭 IDE。

若用户关闭 IDE，IDE 前端可以通知计算服务销毁服务实例。

例如，用户关闭 IDE，IDE 前端可以通知计算服务销毁该 IDE 前端启动时创建的所有服务实例。

以图 18 所示的方案为例，在用户关闭 IDE 的情况下，可以销毁图 18 中的两个服务实例。

下面结合图 19 至图 22 对本申请实施例的装置进行说明。应理解，下面描述的装置能够执行前述本申请实施例的方法，为了避免不必要的重复，下面在介绍本申请实施例的装置时适当省略重复的描述。

图 19 是本申请实施例的用于软件开发的系统的示意性框图。图 19 所示的系统 2000 可以用于执行图 9 和/或图 18 所示的方法。

如图 19 所示，系统 2000 可以包括 IDE 前端 2010 和第一 IDE 后端 2020。IDE 前端可以为图 2 中的 IDE 前端 210。第一 IDE 后端 2020 可以为图 2 中的第一 IDE 后端 220。

第一 IDE 后端 2020 支持运行于多种类型的环境，多种类型的环境包括以下至少一项：多种类型的操作系统或多种类型的硬件架构。

IDE 前端 2010，用于获取用户的配置信息，配置信息包括以下至少一项：硬件架构的类型或操作系统的类型；根据配置信息在第一环境部署第一 IDE 后端 2020，以使得通过 IDE 前端 2010 实现在第一环境中的第一 IDE 后端 2020 上的软件开发，第一环境为第一 IDE 后端支持的多种类型的环境中按照配置信

息确定的环境。

可选地，IDE 前端 2010 运行于 IDE 的客户端程序或浏览器的客户端程序。

可选地，IDE 前端 2010 包括用户界面 UI 组件和适配层，UI 组件用于提供用户界面，适配层用于为 UI 组件提供统一接口，以使 UI 组件运行于一种或多种客户端程序中。

可选地，第一 IDE 后端 2020 包括 IDE 内核，IDE 内核用于提供多种编程语言的语言特性。

可选地，DE 内核包括：索引系统，用于存储源代码的索引信息，语言解析适配层，用于调用源代码的编程语言对应的语言解析器以解析源代码，语言特性层，用于提供多种编程语言的语言特性，多种编程语言包括源代码的编程语言。

可选地，多种类型的环境包括 IDE 前端 2010 所在的本地系统和/或本地系统以外的其他环境。

可选地，第一环境包括一个或多个实例，IDE 前端 2010 具体用于：根据配置信息获取一个或多个实例，一个或多个实例中部署有第一 IDE 后端 2020；或者，根据配置信息获取一个或多个实例；在一个或多个实例的部分或全部实例中未部署第一 IDE 后端 2020 的情况下，在未部署第一 IDE 后端 2020 的实例中部署第一 IDE 后端 2020。

可选地，IDE 前端 2010 还用于：启动一个或多个实例中的第一实例中的第一 IDE 后端 2020，以使得第一实例中的第一 IDE 后端 2020 提供第一实例支持的第一编程语言的语言特性。

可选地，IDE 前端 2010 还用于：与第一实例中的第一 IDE 后端 2020 建立长连接。

可选地，IDE 前端 2010 具体用于：根据第一操作启动第一实例中的第一 IDE 后端 2020，第一操作指示第一编程语言。

可选地，第一操作包括第一代码文件的打开操作，第一编程语言为第一代码文件的编程语言。

可选地，配置信息由携带验证信息的登录请求指示。

可选地，IDE 前端 2010 还用于：将第一请求发送至第一环境中的第一 IDE 后端 2020；以及第一环境中的第一 IDE 后端 2020 用于：生成对第一请求的第一响应；将第一响应返回至 IDE 前端 2010。

可选地，系统还包括运行于第二环境的第二 IDE 后端，以及 IDE 前端 2010 还用于：将第二请求发送至第一环境中的第一 IDE 后端 2020；第一环境中的第一 IDE 后端 2020 用于将第二请求转发至第二环境中的第二 IDE 后端；第二环境中的第二 IDE 后端用于：生成对第二请求的第二响应；将第二响应发送至第一环境中的第一 IDE 后端 2020；以及第一环境中的第一 IDE 后端 2020 还用于：将第二响应返回至 IDE 前端 2010。

可选地，第一环境中的第一 IDE 后端 2020 还用于通过显示服务器协议将远程环境中运行的图形用户界面程序投射到 IDE 前端 2010 所在的本地系统。

具体描述可以参考前文中的方法 600 和方法 700，此处不再赘述。

其中，系统 2000 中的各个模块均可以通过软件实现，或者可以通过硬件实现。示例性的，接下来以第一 IDE 后端 2020 为例，介绍第一 IDE 后端 2020 的实现方式。类似的，其他模块的实现方式可以参考第一 IDE 后端 2020 的实现方式。

模块作为软件功能单元的一种举例，第一 IDE 后端 2020 可以包括运行在计算实例上的代码。其中，计算实例可以包括物理主机（计算设备）、虚拟机、容器中的至少一种。进一步地，上述计算实例可以是一台或者多台。例如，第一 IDE 后端 2020 可以包括运行在多个主机/虚拟机/容器上的代码。需要说明的是，用于运行该代码的多个主机/虚拟机/容器可以分布在相同的区域（region）中，也可以分布在不同的 region 中。进一步地，用于运行该代码的多个主机/虚拟机/容器可以分布在相同的可用区（availability zone，AZ）中，也可以分布在不同的 AZ 中，每个 AZ 包括一个数据中心或多个地理位置相近的数据中心。其中，通常一个 region 可以包括多个 AZ。

同样，用于运行该代码的多个主机/虚拟机/容器可以分布在同一个虚拟私有云（virtual private cloud，VPC）中，也可以分布在多个 VPC 中。其中，通常一个 VPC 设置在一个 region 内，同一 region 内两个 VPC 之间，以及不同 region 的 VPC 之间跨区通信需在每个 VPC 内设置通信网关，经通信网关实现 VPC 之间的互连。

模块作为硬件功能单元的一种举例，第一 IDE 后端 2020 可以包括至少一个计算设备，如服务器等。或者，第一 IDE 后端 2020 也可以是利用专用集成电路（application-specific integrated circuit，ASIC）实现、或可编程逻辑器件（programmable logic device，PLD）实现的设备等。其中，上述 PLD 可以是复杂程序逻辑器件（complex programmable logical device，CPLD）、现场可编程门阵列（field-programmable gate

array，FPGA）、通用阵列逻辑（generic array logic，GAL）或其任意组合实现。

第一 IDE 后端 2020 包括的多个计算设备可以分布在相同的 region 中，也可以分布在不同的 region 中。第一 IDE 后端 2020 包括的多个计算设备可以分布在相同的 AZ 中，也可以分布在不同的 AZ 中。同样，第一 IDE 后端 2020 包括的多个计算设备可以分布在同一个 VPC 中，也可以分布在多个 VPC 中。其中，所述多个计算设备可以是服务器、ASIC、PLD、CPLD、FPGA 和 GAL 等计算设备的任意组合。

需要说明的是，在其他实施例中，第一 IDE 后端 2020 可以用于执行用于软件开发的方法中的任意步骤，其他模块可以用于实现用于软件开发的方法中的任意步骤，各个模块负责实现的步骤可根据需要指定，通过各个模块分别实现用于软件开发的方法中不同的步骤来实现系统 2000 的全部功能。

本申请还提供一种计算设备 1000。如图 20 所示，计算设备 1000 包括：总线 1002、处理器 1004、存储器 1006 和通信接口 1008。处理器 1004、存储器 1006 和通信接口 1008 之间通过总线 1002 通信。计算设备 1000 可以是服务器或终端设备。应理解，本申请不限定计算设备 1000 中的处理器、存储器的个数。

总线 1002 可以是外设部件互连标准（peripheral component interconnect，PCI）总线或扩展工业标准结构（extended industry standard architecture，EISA）总线等。总线可以分为地址总线、数据总线、控制总线等。为便于表示，图 20 中仅用一条线表示，但并不表示仅有一根总线或一种类型的总线。总线 1002 可包括在计算设备 1000 各个部件（例如，存储器 1006、处理器 1004、通信接口 1008）之间传送信息的通路。

处理器 1004 可以包括中央处理器(central processing unit，CPU)、图形处理器（graphics processing unit，GPU）、微处理器（micro processor，MP）或者数字信号处理器（digital signal processor，DSP）等处理器中的任意一种或多种。

存储器 1006 可以包括易失性存储器(volatile memory)，例如随机存取存储器(random access memory，RAM)。处理器 1004 还可以包括非易失性存储器(non-volatile memory)，例如只读存储器(read-only memory，ROM)，快闪存储器，机械硬盘（hard disk drive，HDD）或固态硬盘（solid state drive，SSD）。

存储器 1006 中存储有可执行的程序代码，处理器 1004 执行该可执行的程序代码以分别实现前述各个模块的功能，从而实现用于软件开发的方法。也即，存储器 1006 上存有用于执行用于软件开发的方法的指令。

通信接口 1008 使用例如但不限于网络接口卡、收发器一类的收发模块，来实现计算设备 1000 与其他设备或通信网络之间的通信。

本申请实施例还提供了一种计算设备集群。该计算设备集群包括至少一台计算设备。该计算设备可以是服务器，例如是中心服务器、边缘服务器，或者是本地数据中心中的本地服务器。在一些实施例中，计算设备也可以是台式机、笔记本电脑或者智能手机等终端设备。

如图 21 所示，所述计算设备集群包括至少一个计算设备 1000。计算设备集群中的一个或多个计算设备 1000 中的存储器 1006 中可以存有相同的用于执行用于软件开发的方法的指令。

在一些可能的实现方式中，该计算设备集群中的一个或多个计算设备 1000 的存储器 1006 中也可以分别存有用于执行用于软件开发的方法的部分指令。换言之，一个或多个计算设备 1000 的组合可以共同执行用于软件开发的方法的指令。

需要说明的是，计算设备集群中的不同的计算设备 1000 中的存储器 1006 可以存储不同的指令，分别用于执行用于软件开发的系统的部分功能。也即，不同的计算设备 1000 中的存储器 1006 存储的指令可以实现系统 2000 中的一个或多个模块的功能。

在一些可能的实现方式中，计算设备集群中的一个或多个计算设备可以通过网络连接。其中，所述网络可以是广域网或局域网等等。

图 22 示出了一种可能的实现方式。如图 22 所示，两个计算设备 1000A 和 1000B 之间通过网络进行连接。具体地，通过各个计算设备中的通信接口与所述网络进行连接。在这一类可能的实现方式中，计算设备 1000A 中的存储器 1006 中存有执行 IDE 前端 2010 的功能的指令。同时，计算设备 1000B 中的存储器 1006 中存有执行第一 IDE 后端 2020 的功能的指令。

应理解，图 22 中示出的计算设备 1000A 的功能也可以由多个计算设备 1000 完成。同样，计算设备 1000B 的功能也可以由多个计算设备 1000 完成。

本申请实施例还提供了一种包含指令的计算机程序产品。所述计算机程序产品可以是包含指令的，能够运行在计算设备上或被储存在任何可用介质中的软件或程序产品。当所述计算机程序产品在至少一

个计算设备上运行时，使得至少一个计算设备执行用于软件开发的方法。

本申请实施例还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可以是计算设备能够存储的任何可用介质或者是包含一个或多个可用介质的数据中心等数据存储设备。所述可用介质可以是磁性介质，（例如，软盘、硬盘、磁带）、光介质（例如，DVD）、或者半导体介质（例如固态硬盘）等。该计算机可读存储介质包括指令，所述指令指示计算设备执行用于软件开发的方法。

本领域普通技术人员可以意识到，结合本文中所公开的实施例描述的各示例的单元及算法步骤，能够以电子硬件、或者计算机软件和电子硬件的结合来实现。这些功能究竟以硬件还是软件方式来执行，取决于技术方案的特定应用和设计约束条件。专业技术人员可以对每个特定的应用来使用不同方法来实现所描述的功能，但是这种实现不应认为超出本申请的范围。

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、装置和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。

在本申请所提供的几个实施例中，应该理解到，所揭露的系统、装置和方法，可以通过其它的方式实现。例如，以上所描述的装置实施例仅仅是示意性的，例如，所述单元的划分，仅仅为一种逻辑功能划分，实际实现时可以有另外的划分方式，例如多个单元或组件可以结合或者可以集成到另一个系统，或一些特征可以忽略，或不执行。另一点，所显示或讨论的相互之间的耦合或直接耦合或通信连接可以是通过一些接口，装置或单元的间接耦合或通信连接，可以是电性，机械或其它的形式。

所述作为分离部件说明的单元可以是或者也可以不是物理上分开的，作为单元显示的部件可以是或者也可以不是物理单元，即可以位于一个地方，或者也可以分布到多个网络单元上。可以根据实际的需要选择其中的部分或者全部单元来实现本实施例方案的目的。

另外，在本申请各个实施例中的各功能单元可以集成在一个处理单元中，也可以是各个单元单独物理存在，也可以两个或两个以上单元集成在一个单元中。

所述功能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，本申请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）执行本申请各个实施例所述处理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。

以上所述，仅为本申请的具体实施方式，但本申请的保护范围并不局限于此，任何熟悉本技术领域的技术人员在本申请揭露的技术范围内，可轻易想到变化或替换，都应涵盖在本申请的保护范围之内。因此，本申请的保护范围应以所述权利要求的保护范围为准。

权 利 要 求 书

1. 一种用于软件开发的方法，其特征在于，应用于集成开发环境 IDE，所述 IDE 包括 IDE 前端和第一 IDE 后端，所述第一 IDE 后端支持运行于多种类型的环境，所述多种类型的环境包括以下至少一项：多种类型的操作系统或多种类型的硬件架构，所述方法包括：

所述 IDE 前端获取用户的配置信息，所述配置信息包括以下至少一项：硬件架构的类型或操作系统的类型；

所述 IDE 前端根据所述配置信息在第一环境部署所述第一 IDE 后端，以使得通过所述 IDE 前端实现在所述第一环境中的所述第一 IDE 后端上的软件开发，所述第一环境为所述第一 IDE 后端支持的多种类型的环境中按照所述用户的配置信息确定的环境。

2. 根据权利要求 1 所述的方法，其特征在于，所述 IDE 前端运行于所述 IDE 的客户端程序或浏览器的客户端程序。

3. 根据权利要求 2 所述的方法，其特征在于，所述 IDE 前端包括用户界面 UI 组件和适配层，所述 UI 组件用于提供所述用户界面，所述适配层用于为所述 UI 组件提供统一接口，以使所述 UI 组件运行于一种或多种客户端程序中。

4. 根据权利要求 1 至 3 中任一项所述的方法，其特征在于，所述第一 IDE 后端包括 IDE 内核，所述 IDE 内核用于提供多种编程语言的语言特性。

5. 根据权利要求 4 所述的方法，其特征在于，所述 IDE 内核包括：

索引系统，用于存储源代码的索引信息，

语言解析适配层，用于调用所述源代码的编程语言对应的语言解析器以解析所述源代码，

语言特性层，用于提供所述多种编程语言的语言特性，所述多种编程语言包括所述源代码的编程语言。

6. 根据权利要求 1 至 5 中任一项所述的方法，其特征在于，所述多种类型的环境包括所述 IDE 前端所在的本地系统和/或所述本地系统以外的其他环境。

7. 根据权利要求 1 至 6 中任一项所述的方法，其特征在于，所述第一环境包括一个或多个实例，以及所述 IDE 前端根据所述配置信息在第一环境上部署所述第一 IDE 后端，包括：

所述 IDE 前端根据所述配置信息获取所述一个或多个实例，所述一个或多个实例中部署有所述第一 IDE 后端；或者，

所述 IDE 前端根据所述配置信息获取所述一个或多个实例；

在所述一个或多个实例的部分或全部实例中未部署所述第一 IDE 后端的情况下，所述 IDE 前端在未部署所述第一 IDE 后端的实例中部署所述第一 IDE 后端。

8. 根据权利要求 7 所述的方法，其特征在于，所述方法还包括：

所述 IDE 前端启动所述一个或多个实例中的第一实例中的所述第一 IDE 后端，以使得所述第一实例中的所述第一 IDE 后端提供所述第一实例支持的第一编程语言的语言特性。

9. 根据权利要求 8 所述的方法，其特征在于，所述方法还包括：

所述 IDE 前端与所述第一实例中的所述第一 IDE 后端建立长连接。

10. 根据权利要求 8 或 9 所述的方法，其特征在于，所述 IDE 前端启动所述一个或多个实例中的第一实例中的所述第一 IDE 后端，包括：

所述 IDE 前端根据第一操作启动所述第一实例中的所述第一 IDE 后端，所述第一操作指示所述第一编程语言。

11. 根据权利要求 10 所述的方法，其特征在于，所述第一操作包括第一代码文件的打开操作，所述第一编程语言为所述第一代码文件的编程语言。

12. 根据权利要求 1 至 11 中任一项所述的方法，其特征在于，所述配置信息由携带验证信息的登录请求指示。

13. 根据权利要求 1 至 12 中任一项所述的方法，其特征在于，所述方法还包括：

所述 IDE 前端将第一请求发送至所述第一环境中的所述第一 IDE 后端；

所述第一环境中的所述第一 IDE 后端生成对所述第一请求的第一响应；

所述第一环境中的所述第一 IDE 后端将所述第一响应返回至所述 IDE 前端。

14. 根据权利要求 1 至 12 中任一项所述的方法，其特征在于，所述 IDE 还包括运行于第二环境的第二 IDE 后端，以及所述方法还包括：

所述 IDE 前端将第二请求发送至所述第一环境中的所述第一 IDE 后端；

所述第一环境中的所述第一 IDE 后端将所述第二请求转发至所述第二环境中的所述第二 IDE 后端；

所述第二环境中的所述第二 IDE 后端生成对所述第二请求的第二响应；

所述第二环境中的所述第二 IDE 后端将所述第二响应发送至所述第一环境中的所述第一 IDE 后端；

所述第一环境中的所述第一 IDE 后端将所述第二响应返回至所述 IDE 前端。

15. 根据权利要求 1 至 14 中任一项所述的方法，其特征在于，所述方法还包括：

所述第一环境中的所述第一 IDE 后端通过显示服务器协议将远程环境中运行的图形用户界面程序投射到所述 IDE 前端所在的本地系统。

16. 一种用于软件开发的系统，其特征在于，包括：集成开发环境 IDE 前端和第一 IDE 后端，所述第一 IDE 后端支持运行于多种类型的环境，所述多种类型的环境包括以下至少一项：多种类型的操作系统或多种类型的硬件架构，

所述 IDE 前端用于：

获取用户的配置信息，所述配置信息包括以下至少一项：硬件架构的类型或操作系统的类型；

根据所述配置信息在第一环境部署所述第一 IDE 后端，以使得通过所述 IDE 前端实现在所述第一环境中的所述第一 IDE 后端上的软件开发，所述第一环境为所述第一 IDE 后端支持的多种类型的环境中按照所述用户的配置信息确定的环境。

17. 根据权利要求 16 所述的系统，其特征在于，所述 IDE 前端运行于 IDE 的客户端程序或浏览器的客户端程序。

18. 根据权利要求 17 所述的系统，其特征在于，所述 IDE 前端包括用户界面 UI 组件和适配层，所述 UI 组件用于提供所述用户界面，所述适配层用于为所述 UI 组件提供统一接口，以使所述 UI 组件运行于一种或多种客户端程序中。

19. 根据权利要求 16 至 18 中任一项所述的系统，其特征在于，所述第一 IDE 后端包括 IDE 内核，所述 IDE 内核用于提供多种编程语言的语言特性。

20. 根据权利要求 19 所述的系统，其特征在于，所述 IDE 内核包括：

索引系统，用于存储源代码的索引信息，

语言解析适配层，用于调用所述源代码的编程语言对应的语言解析器以解析所述源代码，

语言特性层，用于提供所述多种编程语言的语言特性，所述多种编程语言包括所述源代码的编程语言。

21. 根据权利要求 16 至 20 中任一项所述的系统，其特征在于，所述多种类型的环境包括所述 IDE 前端所在的本地系统和/或所述本地系统以外的其他环境。

22. 根据权利要求 16 至 21 中任一项所述的系统，其特征在于，所述第一环境包括一个或多个实例，所述 IDE 前端具体用于：

根据所述配置信息获取所述一个或多个实例，所述一个或多个实例中部署有所述第一 IDE 后端；或者，

根据所述配置信息获取所述一个或多个实例；

在所述一个或多个实例的部分或全部实例中未部署所述第一 IDE 后端的情况下，在未部署所述第一 IDE 后端的实例中部署所述第一 IDE 后端。

23. 根据权利要求 22 所述的系统，其特征在于，所述 IDE 前端还用于：

启动所述一个或多个实例中的第一实例中的所述第一 IDE 后端，以使得所述第一实例中的所述第一 IDE 后端提供所述第一实例支持的第一编程语言的语言特性。

24. 根据权利要求 23 所述的系统，其特征在于，所述 IDE 前端还用于：

与所述第一实例中的所述第一 IDE 后端建立长连接。

25. 根据权利要求 23 或 24 所述的系统，其特征在于，所述 IDE 前端具体用于：

根据第一操作启动所述第一实例中的所述第一 IDE 后端，所述第一操作指示所述第一编程语言。

26. 根据权利要求 25 所述的系统，其特征在于，所述第一操作包括第一代码文件的打开操作，所述

第一编程语言为所述第一代码文件的编程语言。

27. 根据权利要求 16 至 26 中任一项所述的系统，其特征在于，所述配置信息由携带验证信息的登录请求指示。

28. 根据权利要求 16 至 27 中任一项所述的系统，其特征在于，所述 IDE 前端还用于：

将第一请求发送至所述第一环境中的第一 IDE 后端；以及所述第一环境中的第一 IDE 后端用于：

生成对所述第一请求的第一响应；

将所述第一响应返回至所述 IDE 前端。

29. 根据权利要求 16 至 27 中任一项所述的系统，其特征在于，所述系统还包括运行于第二环境的第二 IDE 后端，以及所述 IDE 前端还用于：

将第二请求发送至所述第一环境中的第一 IDE 后端；

所述第一环境中的所述第一 IDE 后端用于将所述第二请求转发至所述第二环境中的所述第二 IDE 后端；

所述第二环境中的所述第二 IDE 后端用于：

生成对所述第二请求的第二响应；

将所述第二响应发送至所述第一环境中的所述第一 IDE 后端；以及所述第一环境中的所述第一 IDE 后端还用于：

将所述第二响应返回至所述 IDE 前端。

30. 根据权利要求 16 至 29 中任一项所述的系统，其特征在于，所述第一环境中的第一 IDE 后端还用于通过显示服务器协议将远程环境中运行的图形用户界面程序投射到所述 IDE 前端所在的本地系统。

31. 一种计算设备集群，其特征在于，包括至少一个计算设备，每个计算设备包括处理器和存储器；

所述至少一个计算设备的处理器用于执行所述至少一个计算设备的存储器中存储的指令，以使得所述计算设备集群执行如权利要求 1 至 15 中任一项所述的方法。

32. 一种计算机可读存储介质，其特征在于，包括计算机程序指令，当所述计算机程序指令由计算设备集群执行时，所述计算设备集群执行如权利要求 1 至 15 中任一项所述的方法。

33. 一种包含指令的计算机程序产品，其特征在于，当所述指令被计算设备集群运行时，使得所述计算设备集群执行如权利要求 1 至 15 中任一项所述的方法。