

编程语言的类型推断的方法及装置

本申请要求在 2023 年 12 月 13 日提交俄罗斯联邦国家知识产权局、申请号为 2023132940 的俄罗斯联邦专利申请的优先权，发明名称为“编程语言的类型推断的方法及装置”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

本申请涉及信息技术领域，更具体地，涉及一种编程语言的类型推断的方法及装置。

背景技术

编程语言的类型推断 (type inference) 是提高程序的可靠性的重要途径，在程序理解和防止错误等方面具有重要的作用。以集成开发环境 (integrated development environment, IDE) 为例，对编程语言表达式的类型推断是很多 IDE 功能的基础，例如，源代码中的限定解析、代码补全以及程序语义的校验等功能，因此，类型推断是 IDE 的代码模型组件的重要任务之一。

相关方案通常需要基于整个项目来进行类型推断。若源代码发生更改，则需要为整个项目重新推断类型。类型推断需要占用内存，尤其是在需要处理大型项目或者大批量文件时，类型推断会占用大量内存，造成较为严重的内存消耗。

发明内容

本申请提供一种编程语言的类型推断的方法及装置，有利于减少类型推断所占用的内存。

第一方面，提供了一种编程语言的类型推断的方法，该方法可以应用于 IDE，该方法包括：在程序代码开发的过程中，获取抽象语法树中的待推断的节点，待推断的节点指示程序代码中的待推断的字段，待推断的字段为如下的一项：变量、类、函数或函数；获取与待推断的节点相关的约束，约束用于指示待推断的节点在抽象语法树中涉及的关系；根据约束确定待推断的节点的类型，待推断的节点的类型包括如下的一项：数字类型、序列类型、映射类型、集合类型、迭代器类型、上下文管理器类型、模块类型、元类型、空值类型或可调用类型。

在本申请实施例的方案中，基于与待推断的节点相关的约束来推断待推断的节点的类型，即基于待推断的节点在 AST 中涉及的关系来确定待推断的节点的类型，无需对整个项目的类型进行推断，有利于降低内存消耗，同时有利于提高类型推断的效率，从而实现类型的实时刷新，有利于提高用户体验。

同时，在本申请实施例的方案中，基于与待推断的节点相关的约束来获取待推断的节点的类型推理结果，有利于实现类型的增量推断，即实现类型的递增性修改。例如，在源代码发生更改的情况下，可以仅针对发生更改的源代码文件和相应的依赖项进行类型的推断，无需对整个项目的类型重新进行推断。

以 Python 为例，该待推断的节点可以包括以下任一项：变量节点、类节点、函数节点或参数节点等。

示例性地，节点约束可以包括以下至少一项：节点的基础类类型约束、节点的解析引用约束、节点的属性读取约束、节点的默认值约束、节点的调用约束、节点的赋值约束、节点的二元约束、节点的参数传递约束或节点的返回约束。

结合第一方面，在第一方面的某些实现方式中，待推断的节点的类型用于以下至少一项：显示待推断的节点的类型、代码补全、代码重构、代码检查或快速修复。

结合第一方面，在第一方面的某些实现方式中，获取抽象语法树中的待推断的节点，包括：从推理队列中取出第一推理项，第一推理项用于确定待推断的节点的类型，推理队列中包括一个或多个推理项；获取待推断的节点。

在本申请实施例中，可以采用推理队列来处理类型推断的任务，允许延迟模式下的类型推断，有利于推升推断性能。

结合第一方面，在第一方面的某些实现方式中，约束包括抽象语法树中的第一节点的解析引用约束，

第一节点的解析引用约束用于指示待推断的节点对第一节点的引用关系，方法还包括：获取第一节点的类型，以及根据约束确定待推断的节点的类型，包括：根据第一节点的类型和约束确定待推断的节点的类型。

在本申请实施例的方案中，若待推断的节点涉及对其他节点的引用，则先获取被引用的节点的类型，然后基于被引用的节点的类型来推断待推断的节点的类型，无需对整个项目的类型进行推断。

结合第一方面，在第一方面的某些实现方式中，约束包括第二节点的赋值约束，第二节点的赋值约束用于指示第二节点涉及的赋值关系，第二节点中被赋值的子节点的类型是根据赋值号右侧的类型确定的。

结合第一方面，在第一方面的某些实现方式中，约束包括第三节点的属性读取约束，第三节点的属性读取约束用于指示第三节点涉及的从属关系，第三节点中的属性的所有者的类型包括属性的类型。

结合第一方面，在第一方面的某些实现方式中，方法还包括：创建待推断的节点对应的类型变量；以及根据约束确定待推断的节点的类型，包括：根据约束确定待推断的节点对应的类型变量的值，待推断的节点对应的类型变量的值用于指示待推断的节点的类型。

创建的新的类型变量可以称为唯一自由类型变量。唯一自由类型变量指的是目前未与其他任何具体的类型相关联的变量。

以 python 语言为例，示例性地，待推断的字段可以包括以下任一项：类、静态变量、实例类变量、函数、参数或返回值。可以为类、静态变量、实例类变量、函数、参数或返回值等分别创建新的类型变量。

结合第一方面，在第一方面的某些实现方式中，方法还包括：将待推断的节点的类型存储至类型环境中，类型环境用于存储抽象语法树中的节点的类型。

结合第一方面，在第一方面的某些实现方式中，方法还包括：输出推断结果，该推断结果指示该待推断的节点的类型。

第二方面，提供了一种编程语言的类型推断的装置，装置应用于集成开发环境 IDE，装置包括：第一获取模块，用于在程序开发的过程中，获取抽象语法树中的待推断的节点，待推断的节点指示程序代码中的待推断的字段，待推断的字段包括如下的一项：变量、类、函数或参数；第二获取模块，用于获取与待推断的节点相关的约束，约束用于指示待推断的节点在抽象语法树中涉及的关系；处理模块，用于根据约束确定待推断的节点的类型，待推断的节点的类型包括以下至少一项：数字类型、序列类型、映射类型、集合类型、迭代器类型、上下文管理器类型、模块类型、元类型、空值类型或可调用类型。

结合第二方面，在第二方面的某些实现方式中，待推断的节点的类型用于以下至少一项：显示待推断的节点的类型、代码补全、代码重构、代码检查或快速修复。

结合第二方面，在第二方面的某些实现方式中，第一获取模块具体用于：从推理队列中取出第一推理项，第一推理项用于确定待推断的节点的类型，推理队列中包括一个或多个推理项；获取待推断的节点。

结合第二方面，在第二方面的某些实现方式中，约束包括抽象语法树中的第一节点的解析引用约束，第一节点的解析引用约束用于指示待推断的节点对第一节点的引用关系，方法还包括第三获取模块，用于获取第一节点的类型，以及处理模块具体用于：根据第一节点的类型和约束确定待推断的节点的类型。

结合第二方面，在第二方面的某些实现方式中，约束包括第二节点的赋值约束，第二节点的赋值约束用于指示第二节点涉及的赋值关系，第二节点中被赋值的子节点的类型是根据赋值号右侧的类型确定的。

结合第二方面，在第二方面的某些实现方式中，约束包括第三节点的属性读取约束，第三节点的属性读取约束用于指示第三节点涉及的从属关系，第三节点中的属性的所有者的类型包括属性的类型。

结合第二方面，在第二方面的某些实现方式中，装置还包括创建模块，用于创建待推断的节点对应的类型变量；以及处理模块具体用于：根据约束确定待推断的节点对应的类型变量的值，待推断的节点对应的类型变量的值用于指示待推断的节点的类型。

结合第二方面，在第二方面的某些实现方式中，装置还包括存储模块，用于将待推断的节点的类型存储至类型环境中，类型环境用于存储抽象语法树中的节点的类型。

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以及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以及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，当所述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以及第一方面的任意一种实现方式中的方法。

附图说明

图1是本申请实施例的一种编程语言的类型推断的方法的示意性流程图；

图2是本申请实施例的一种赋值约束的示意图；

图3是本申请实施例的另一种编程语言的类型推断的方法的示意性流程图；

图4是本申请实施例的一种编程语言的类型推断的装置的示意性框图；

图5是本申请实施例的一种计算设备的示意性框图；

图6是本申请实施例的一种计算设备集群的示意性框图；

图7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，并非限定时间，且也不要要求设备在实现时一定要有所判断的动作，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安装于用户的本地开发操作环境中。示例性地，本地开发操作环境可以为终端设备，例如，台式电脑、笔记本电脑或手机等。

Web IDE指的是在线的IDE服务，包括IDE前端和IDE后端。IDE后端运行于远程环境中。例如，远程环境可以为云服务器，即以云服务的形式为用户提供IDE。用户可以通过访问网页的形式购买云服

务，云服务提供商创建 IDE 实例。用户可以基于该 IDE 实例进行程序开发。

2. 抽象语法树 (abstract syntax tree, AST) :

AST 可以简称为语法树 (syntax tree)，是源代码的语法结构的一种抽象表示。AST 以树状的形式表示源代码的语法结构，树上的每个节点都表示源代码中的一种结构，如变量声明、表达式、函数调用或控制结构等。树的根节点通常表示整个源代码文件，子节点表示具体的语法元素及其关系。

之所以说语法是“抽象”的，是因为这里的语法并不会表示出真实语法中出现的每个细节。比如，嵌套括号被隐含在树的结构中，并没有以节点的形式呈现；而类似于 if-condition-then 这样的条件跳转语句，可以使用带有三个分支的节点来表示。

AST 的作用非常广泛，例如可以用于代码语法的检查、代码的高亮、代码错误的检查、代码的补全、代码的优化、代码的修复、代码的生成以及代码的重构等。

静态语言 (例如，Java 或 C# 等) 的类型推断主要通过编译器完成。代码模型可以通过类型绑定获得所需要的类型信息。类型绑定通常存在于程序的元数据 metadata 中。

动态语言的类型推断主要由 IDE 完成，由于缺少编译元数据的支持，动态语言的类型推断是一个业界难题。以 Python 为例，Python 具有一个完全动态严格的类型系统，类型 (type) 在运行时动态绑定到变量 (variable)，变量和类型都可以在运行时动态被改变，这增加了实现可靠的类型推断的难度。

一些相关方案可以实现对动态语言的类型推断。例如，采用静态语言的类型推断的方式来实现动态语言的类型推断，即通过解析生成程序的元数据 (.pyi 文件)，进而根据元数据获取所需要的类型信息。在目前的方案中，需要基于整个项目来进行类型推断。若源代码发生更改，则需要为整个项目重新推断类型。类型推断需要占用内存，尤其是在需要处理大型项目或者大批量文件时，类型推断会占用大量内存，造成较为严重的内存消耗。

有鉴于此，本申请实施例提供了一种编程语言的类型推断的方法，有利于降低内存消耗。

本申请实施例的方法可以应用于需要进行编程语言的类型推断的场景中。

示例性地，本申请实施例的方案可以应用于需要显示字段的类型的场景中。例如，在鼠标悬停在某个变量上时，可以显示该变量的类型。该变量的类型可以通过本申请实施例的方法确定的。

示例性地，本申请实施例的方案可以应用于需要基于编程语言的类型进行后续处理的场景。例如，本申请实施例的方案可以应用于代码补全、代码重构、代码检查或快速修复等场景中。

图 1 出了本申请实施例的一种编程语言的类型推断的方法，图 1 所示的方法 100 可以由编程语言的类型推断的装置执行。

示例性地，编程语言的类型推断的装置可以为开发工具或开发工具的功能模块。例如，开发工具可以为 IDE。即方法 100 可以应用于 IDE 中。

示例性地，编程语言的类型推断的装置也可以为与开发工具配合使用的装置。

IDE 可以包括桌面 IDE 或 Web IDE。Web IDE 的后端运行于远程环境中。例如，远程环境可以为云服务器。

以上仅为示例，本申请实施例的方案还可以由其他装置执行，本申请实施例对此不做限定。

如图 1 所示，方法 100 包括如下步骤。

110，获取 AST 中的待推断的节点。

120，获取与该待推断的节点相关的约束。该约束用于指示该待推断的节点在该 AST 中涉及的关系。

130，根据该约束确定该待推断的节点的类型。

在本申请实施例的方案中，基于与待推断的节点相关的约束来推断待推断的节点的类型，即基于待推断的节点在 AST 中涉及的关系来确定待推断的节点的类型，无需对整个项目的类型进行推断，有利于降低内存消耗，同时有利于提高类型推断的效率，从而实现类型的实时刷新，有利于提高用户体验。

同时，在本申请实施例的方案中，基于与待推断的节点相关的约束来获取待推断的节点的类型推理结果，有利于实现类型的增量推断，即实现类型的递增性修改。例如，在源代码发生更改的情况下，可以仅针对发生更改的源代码文件和相应的依赖项进行类型的推断，无需对整个项目的类型重新进行推断。

在步骤 110 中，待推断的节点可以指示代码段中的待推断的字段。待推断的节点的类型即待推断的字段类型。

示例性地，以 Python 为例，该待推断的节点可以包括以下任一项：变量节点、类节点、函数节点或

参数节点等。待推断的字段可以包括以下任一项：变量、类、函数或参数。

相应地，待推断的节点的类型可以包括以下任一项：变量的类型、参数的类型、类的类型和函数的类型。

示例性地，以 Python 为例，待推断的节点的类型包括以下至少一项：数字（number）类型、序列（sequence）类型、映射（mapping）类型、集合（set）类型、迭代器（iterator）类型、上下文管理器（context manager）类型、模块（module）类型、元（metaclass）类型、空值（none）类型或可调用（callable）类型。

例如，数字类型可以包括以下任一项：整数类型、浮点数类型、复数类型、布尔（bool）类型或长整型等。

例如，序列类型可以包括以下任一项：字符串（string）类型、列表（list）类型、元组（tuple）类型或字节（bytes）类型。

例如，映射类型可以包括字典类型。

例如，集合类型可以包括：集合和不可变集合。

例如，可调用类型可以包括：函数、方法或类。

例如，迭代器类型可以包括以下任一项：生成器（generator）、生成器函数（generator function）或内置迭代器。

例如，上下文管理器类型可以包括实现了 enter（）和 exit（）方法的对象。

例如，模块类型可以包括以下任一项：模块（module）或包（package）。

示例性地，变量的类型或参数的类型可以包括以下任一项：数字类型、序列类型、映射类型、集合类型或空值类型。

示例性地，函数的类型可以包括可调用类型。进一步地，函数的类型还可以包括该函数的参数的类型和返回值的类型。

进一步地，待推断的节点的类型还可以包括：任意类型或未知类型。

应理解，以上仅为示例，不对本申请实施例中的字段的类型构成限定。

在本申请实施例的方案中，代码中的“函数”也可以称为“方法”，本申请实施例对此不做区分。

待推断的节点的数量可以为一个，也可以为多个。本申请实施例对待推断的节点的数量不做限定。

与待推断的节点相关的约束的数量可以为一个，也可以为多个。本申请实施例对约束的数量不做限定。

需要说明的是，若该待推断的节点的数量为多个，则该待推断的节点可以是一次性获取的，也可以是多次获取的，本申请实施例对此不做限定。

方法 100 可以应用于程序代码的开发过程中。

方法 100 可以通过多种方式触发。

在一种可能的实现方式中，方法 100 可以由源代码文件的打开操作触发。

例如，源代码文件之前未执行过类型推断的任务，在用户打开源代码文件时，方法 100 被触发执行。

该 AST 可以为源代码文件对应的 AST。该待推断的节点可以为该 AST 中的部分或全部节点。

若源代码文件较大，一次性对源代码文件的全部代码进行类型推断可能会占用较多的内存，同时会影响处理效率。在本申请实施例的方案中，可以对源代码文件中的部分代码进行类型推断，或者说，可以将源代码文件中的代码分块进行处理，无需一次性处理整个源代码文件，从而有利于降低内存消耗，有利于保证处理效率。

示例性地，该待推断的节点可以来自显示在目标区域中的代码段中的字段对应的节点。

这样，可以对源代码文件显示在目标区域中的代码段进行类型推断。

若源代码文件较大，源代码文件中的代码可能无法同时展示在一个界面中。在该情况下，可以对显示在同一个区域（例如，目标区域）的代码段进行类型推断。

例如，方法 100 可以由 IDE 执行，目标区域可以为 IDE 中的代码编辑框。在打开源代码文件后，可以对当前显示在代码编辑框中的代码段执行方法 100。例如，将当前显示在代码编辑框中的代码段中的字段对应的节点作为待推断的节点，基于待推断的节点执行方法 100，以得到当前显示在代码编辑框中的代码段的类型的推断结果。该推断结果可以展示给用户。

在一种可能的实现方式中，方法 100 可以由鼠标事件触发。

示例性地，方法 100 可以由以下任一项或多项触发：鼠标的悬停操作、鼠标的单击操作或鼠标的双击操作等。

以方法 100 由鼠标的悬停操作触发为例，例如，用户若需要获取某个变量的类型，可以将鼠标悬停在该变量上。响应于该鼠标悬停的操作，执行方法 100。例如，将该变量对应的节点作为待推断的节点，并基于待推断的节点执行方法 100，以得到该变量的类型的推断结果，并将该推断结果展示给用户。

在一种可能的实现方式中，方法 100 可以由键盘事件触发。

示例性地，方法 100 可以由预设键的按压操作触发。

例如，用户若需要获取当前界面上的代码段的类型，则按下预设键。响应于该操作，执行方法 100。比如，将当前界面上的代码段中的字段的节点作为待推断的节点，基于待推断的节点执行方法 100，以得到当前界面上的代码段的类型的推断结果，并将该推断结果展示给用户。

再如，用户若需要获取某一代码段的类型，可以选中该代码段，按下预设键。响应于该操作，执行方法 100。比如，将被选中的代码段中的字段的节点作为待推断的节点，基于待推断的节点执行方法 100，得到被选中的代码段的类型的推断结果，并将该推断结果展示给用户。

应理解，以上触发方式仅为示例，不对本申请实施例的方案构成限定。

可选地，该待推断的节点的类型由该待推断的节点对应的类型变量的值指示。

推断待推断的节点的类型，即推断待推断的节点对应的类型变量的值。

类型变量用于描述类型。类型变量，可以理解为，用变量来表达类型。或者说，类型可以作为一个变量，该变量即为类型变量。节点对应的类型变量的值可以指示该节点的类型。例如，一个函数节点对应的类型变量的值用于指示该函数的类型。再如，一个参数节点对应的类型变量的值用于指示该参数的类型。

可选地，方法 100 还可以包括：为该待推断的节点分别创建对应的类型变量。

该步骤也可以称为类型变量的初始化。可以基于该代码段中的类的声明、变量的声明、方法的声明以及参数的声明等进行类型变量的初始化。

具体地，分别为该待推断的节点创建新的类型变量。例如，该待推断的节点可以来自该代码段中的类的声明、变量的声明、方法的声明以及参数的声明等节点。

创建的新的类型变量可以称为唯一自由类型变量。唯一自由类型变量指的是目前未与其他任何具体的类型相关联的变量。这些变量关联的类型，即唯一自由变量的值，需要通过推断得到。即该待推断的节点的类型变量均未与其他具体的类型相关联。

以 python 语言为例，示例性地，待推断的字段可以包括以下任一项：类、静态变量、实例类变量、函数、参数或返回值。可以为类、静态变量、实例类变量、函数、参数或返回值等分别创建新的类型变量。

应理解，以上仅为示例，节点的类型还可以表示为其他形式，本申请实施例对此不做限定。

约束可以用于描述代码段中的关系。例如，python 程序代码段中的关系可以包括：赋值、函数调用、类实例创建或二元操作等。

以 python 为例，示例性地，该约束可以包括以下至少一项：基础类类型约束、解析引用约束（resolved reference constraint）、属性读取约束、赋值约束（assignment constraint）、二元约束（binary constraint）、默认值约束、参数传递约束、调用约束或返回约束。

例如，该约束包括节点#1 的基础类类型约束，节点#1 的基础类类型约束用于指示节点#1 为基础类。

再如，该约束包括节点#2（第一节点的一例）的解析引用约束，节点#2 的解析引用约束用于指示节点#2 的引用关系。

再如，该约束包括节点#3（第三节点的一例）的属性读取约束，节点#3 的属性读取约束用于指示节点#3 中的从属关系。

再如，该约束包括节点#4 的默认值约束，节点#4 的默认值约束用于指示节点#4 被赋予默认值。

再如，该约束包括节点#5 的调用约束，节点#5 的调用约束用于指示节点#5 为可调用函数。

再如，该约束包括节点#6（第二节点的一例）的赋值约束，节点#6 的赋值约束用于指示节点#6 对应的赋值关系。

再如，该约束包括节点#7 的二元约束，节点#7 的二元约束用于指示节点#7 对应的二元操作关系。

再如，该约束包括节点#8 的参数传递约束，节点#8 的参数传递约束用于指示节点#8 对应的参数传递

关系。

再如，该约束包括节点#9的返回约束，节点#9的返回约束用于指示节点#9对应的返回关系。

该约束可以通过多种方式获取。

示例性地，该约束可以是预先创建的，在该情况下，步骤120可以包括：读取该约束。

示例性地，该约束可以是在方法100的执行过程中创建的。在该情况下，步骤120可以包括：创建该约束。该约束可以根据AST指示的语法元素之间的关系创建的。

示例性地，创建该约束，可以包括以下至少一项：

为属于基础类的节点创建基础类类型约束；

为涉及引用的节点创建解析引用约束；

为涉及从属关系的节点创建属性读取约束；

为被设置为默认值的节点创建默认值约束；

为涉及函数调用的节点创建调用约束；

为涉及赋值关系的节点创建赋值约束；

为涉及二元操作的节点创建二元约束；

为涉及参数传递关系的节点创建参数传递约束；或者

为作为返回值的节点创建返回约束。

例如，在节点#1为基础类的情况下，为节点#1创建基础类类型约束。

例如，AST中涉及对节点#2的引用，为节点#2创建解析引用约束。

为涉及从属关系的节点创建属性读取约束，即为涉及属性读取的节点创建属性读取约束。

例如，AST中的节点#3中涉及从属关系，为节点#3创建属性读取约束。

例如，AST中的节点#4被赋予默认值，为节点#4创建默认值约束。

例如，AST中的节点#5为可调用函数，为节点#5创建调用约束。

例如，AST中的节点#6对应赋值语句，为节点#6创建赋值约束。

例如，AST中的节点#7对应二元操作的表达式，为节点#7创建二元约束。

例如，AST中的节点#8涉及函数的参数传递，为节点#8创建参数传递约束。

例如，AST中的节点#9对应返回语句，为节点#9创建返回约束。

与待推断的节点相关的约束可以包括与待推断的节点直接相关的约束。与待推断的节点直接相关的约束可以理解为，该约束用于指示该待推断的节点与其他语法元素的关系。

进一步地，与待推断的节点相关的约束还可以包括与待推断的节点间接相关的约束。

与待推断的节点间接相关的约束可以包括通过对与待推断的节点直接相关的约束进行分解得到的多个嵌套约束。

图2示出了本申请实施例的一个赋值约束的示意图。下面结合图2对与变量“a”相关的约束进行示例性说明。为了更直观地描述本方案，下面主要以代码片段为例进行说明。在实际执行过程中，各个语法元素之间的关系可以通过AST得到，各个约束可以基于节点创建。例如，为“a=b+c”创建赋值约束，即为“a=b+c”对应的赋值节点创建赋值约束。再如，为变量“b”创建解析引用约束，即为变量“b”的节点（即变量节点“b”）创建解析引用约束。为了便于描述，本申请实施例对此不做区分。

例如，图2所示的代码片段“a=b+c”为赋值语句，涉及赋值关系，为“a=b+c”创建赋值约束。该赋值约束可以视为与变量“a”直接相关的约束。

如图2所示，该赋值约束可以分解为以下几个嵌套约束：“b”和“c”为被引用的变量，为“b”和“c”创建解析引用约束；“b+c”涉及二元操作，为“b+c”创建二元约束。解析引用约束以及二元约束可以视为与变量“a”间接相关的约束。此外，“a”也可以被视为被引用的变量，为“a”创建解析引用约束。

应理解，以上仅以python为例进行说明，不对本申请实施例的方案构成限定，本申请实施例的方案还可以应用于其他动态语言的类型推断。

该约束还可以通过其他方式获取，本申请实施例对此不做限定。

可选地，若该约束包括解析引用约束，获取该解析引用约束指示的依赖项的类型。

换言之，若该待推断的节点涉及对其他节点的引用，被引用的节点即为依赖项，则先获取依赖项的类型，然后可以基于依赖项的类型判断待推断的节点的类型。

例如，节点#2 的解析引用约束指示对节点#2 的引用。节点#2 即为依赖项。方法 100 可以包括：获取节点#2 的节点类型。步骤 130 可以包括：根据节点#2 的类型和该约束确定该待推断的节点的类型。

若该解析引用约束指示了多个依赖项，该多个依赖项的类型的获取方式可以相同，也可以不同。

进一步地，可选地，方法 100 可以包括：判断依赖项的类型是否已经被推断。

可选地，可以根据类型环境判断依赖项的类型是否已经被推断。类型环境可以用于存储 AST 中的节点的类型。

即类型环境可以用于存储源代码文件中的字段的类型。

示例性地，节点的类型可以由节点对应的类型变量的值指示。在该情况下，类型环境可以用于存储类型变量的值。

例如，判断节点的类型是否被推断，可以为，判断类型环境中是否存在该节点对应的类型变量的值。若存在，则该节点的类型已经被推断。若不存在，则该节点的类型还未被推断。

若该依赖项的类型之前已经被推断，则获取该依赖项的类型可以包括读取该依赖项的类型。

若该依赖项的类型还未被推断，则获取该依赖项的类型可以包括推断该依赖项的类型。例如，若某个依赖项的类型还未被推断，则停止当前的推断过程，转向推断依赖项的类型。可以采用方法 100 确定依赖项的类型，即将该依赖项作为待推断的节点，以得到该依赖项的类型。

在完成所有解析引用约束指示的依赖项的类型的推断后，根据所有依赖项的类型和该约束中除了解析引用约束以外的其他约束确定该待推断的节点的类型。若待推断的节点的类型由类型变量的值指示，确定待推断的节点的类型，即为确定待推断的节点的类型变量的值。

在本申请实施例的方案中，可以基于依赖项的类型推断待推断的字段的类型，无需对整个项目的类型进行推断。

下面对除了解析引用约束以外的其他约束的处理进行说明。

示例性地，若该约束包括节点#1 的基础类类型约束，则节点#1 的类型包括基础类。

示例性地，若该约束包括节点#3 的属性读取约束，则节点#3 中的属性的类型来自该属性的所有者的类型。或者说，节点#3 中的属性的所有者的类型包括该属性的类型。

例如，可以根据所有者的类型确定属性的类型。若所有者的类型已知，则可以在所有者的类型中请求属性的类型，即通过限定符的类型请求属性的类型。

再如，可以根据属性的类型确定所有者的类型。若所有者的类型未知，可以根据属性的类型确定属性的所有者的类型，即所有者的类型包括该属性的类型。

示例性地，若该约束包括节点#4 的默认值约束。若节点#4 的类型可以根据默认值的类型确定。

示例性地，若该约束包括节点#5 的调用约束，则节点#5 的类型为可调用函数。

示例性地，若该约束包括节点#6 的赋值约束，则节点#6 中被赋值的子节点的类型是根据赋值号“=”右侧的类型确定的。

进一步地，被赋值的子节点的类型是根据被赋值的子节点当前的类型和赋值号“=”右侧的类型做联合（unification/unionization）得到的。被赋值的子节点当前的类型，可以称为，赋值号“=”左侧的类型。

若该子节点当前的类型为某个具体的类型，例如，该子节点的类型之前已经定义为类型 A，则将当前的“=”左侧的类型（即类型 A）和“=”右侧的类型（例如，类型 B）做联合（A|B）后，得到的该子节点的类型可以包括“=”左侧的类型或“=”右侧的类型，即该子节点的类型可以包括类型 A 或类型 B。若该子节点当前的类型未知，则该子节点的类型可以包括“=”右侧的类型。

示例性地，若该约束包括节点#8 的参数传递约束，节点#8 中的参数子节点的类型根据节点#8 中的参数传递关系确定。

示例性地，若该约束包括节点#9 的返回约束，则节点#9 涉及的返回值的类型为节点#9 的返回结果的类型。

下面结合图 2 对变量“a”的类型的推断过程进行示例性说明。

如图 2 所示，变量“a”的类型的推断过程可以包括如下步骤。

(1) 推断“b”和“c”的类型。

如前所述，图 2 中的约束包括解析引用约束。解析引用约束指示的依赖项包括“b”和“c”。首先推断“b”和“c”的类型。

(2) 推断加法操作（add operation）的类型。

图 2 中的约束还包括二元约束。该二元操作指示 b 和 c 之间的二元加法操作。基于该二元操作推断加法操作的类型。

(3) 获取变量“a”当前的类型或者“a”的类型变量。

图 2 中的约束包括解析引用约束，该解析引用约束指示的依赖项可以包括“a”，获取“a”的类型。若之前已经定义过变量“a”的类型，则获取变量“a”当前的类型。

若变量“a”之前未被推断，则获取变量“a”的类型变量。

(4) 根据步骤(2)得到的类型和步骤(3)得到的类型确定变量“a”的类型。

图 2 中的约束还包括赋值约束。步骤(2)得到的类型即为“=”右侧的类型。步骤(3)得到的类型即为“=”左侧的类型。

变量“a”的类型的推断结果是步骤(2)得到的类型和步骤(3)得到的类型的联合。

例如，步骤(2)得到的加法操作的类型为 t1，步骤(3)获取到的变量“a”当前的类型为 t2，对 t1 和 t2 作联合{t2|t1}。变量“a”的类型的推断结果包括 t1 或 t2，即变量“a”的类型包括 t1 或 t2。

再如，步骤(2)得到的加法操作的类型为 t1，步骤(3)获取到“a”的类型变量为 t3。对 t1 和该类型变量 t3 做联合，得到 t3={t1}，即类型变量 t3 的值包括 t1，即变量“a”的类型包括 t1。

进一步地，方法 100 还可以包括：判断该待推断的节点的类型是否被推断。

若该待推断的节点的类型未被推断，可以执行步骤 130。

判断该待推断的节点的类型是否被推断，可以包括：判断该待推断的节点中的部分节点的类型是否被推断。或者，判断该待推断的节点的类型是否被推断，可以包括：判断该待推断的节点中的全部节点的类型是否被推断。

示例性地，该待推断的节点包括 AST 的根节点，若根节点的类型未被推断，则可以执行步骤 130，若根节点的类型已经被推断，则不执行步骤 130。

根节点的类型是否被推断可以根据类型环境确定。例如，检查类型环境，若类型环境中存在根节点的类型，则说明根节点的类型已完成推断。若类型环境中不存在根节点的类型，则说明根节点的类型未完成推断。若根节点的类型已经被推断，则该源代码文件之前已经执行过推断任务，可以直接输出推断结果。推断结果可以根据类型环境确定。

这样可以避免类型的重复推断，有利于进一步减少内存的消耗，提高推断效率。

进一步地，方法 100 还可以包括：创建第一推理项，将第一推理项添加至推理队列中。第一推理项用于确定该待推断的节点的类型。

或者说，第一推理项指示该待推断的节点的类型推断任务。

可选地，步骤 110 可以包括：从推理队列中提取第一推理项，获取该待推断的节点。

推理队列指的是用于存储推理项的队列。推理队列中包括一个或多个推理项。推理项用于指示类型的推断任务。

推理队列可以采用先进先出（first in first out, FIFO）的数据结构。

在本申请实施例中，可以采用推理队列来处理类型推断的任务，允许延迟模式下的类型推断，有利于推升推断性能。延迟模式也可以称为懒惰模式（lazy mode）。

示例性地，在步骤 130 中，若依赖项的类型未被推断，则停止第一推理项的推断过程。创建第二推理项，将第二推理项添加到推理队列中。第二推理项指示该依赖项的类型的推断任务。

“第一推理项”和“第二推理项”仅用于区分不同的节点的类型推断任务，不具有其他限定作用。

若解析引用约束指示的依赖项中存在多个依赖项的类型未被推断，则可以为该多个依赖项分别创建推理项，将该多个依赖项对应的推理项均添加至推理队列中。该多个依赖项对应的推理项均可以视为第二推理项。

在所有依赖项对应的推理项均从推理队列中取出并完成推断后，再继续处理第一推理项。

进一步地，方法 100 还可以包括：将该待推断的节点的类型存储于类型环境中。

进一步地，方法 100 还可以包括：输出推断结果，该推断结果指示该待推断的节点的类型。

示例性地，输出推断结果，可以包括，显示该待推断的节点的类型。

可选地，该待推断的节点的类型还可以用于以下至少一项：代码补全、代码重构、代码检查或快速修复。

图 3 示出了本申请实施例提供的一种编程语言的类型推断的方法。图 3 所示的方法 300 可以视为方法

100 的一种具体实现方式。相关描述可以参考方法 100，为避免重复，在描述方法 300 时适当省略部分描述。

为了更好地说明本申请实施例的方案，下面主要结合如下 python 代码对方法 300 进行示例性说明。示例性地，方法 300 可以是由源代码文件打开操作触发的。代码段#1 可以为 IDE 代码编辑框中当前显示的代码段。方法 300 主要以该场景为例进行说明，不对本申请实施例的方案构成限定。

代码段#1

```

0  import winapi
1  from multiprocessing.connection import _ConnectionBase
2  import multiprocessing
3  import io
4
5  class CustomConnection(_ConnectionBase):
6      _write = multiprocessing.send
7      _read = multiprocessing.recv
8
9      def _send(self, buf, write=_write):
10         remaining = len(buf)
11         while True:
12             n = write(self._handle, buf)
13             remaining -= n
14             if remaining == 0:
15                 break
16             buf = buf[n:]
17
18     def _recv(self, size, receiver):
19         receiver._init_receiver(CustomConnection._read, size)
20         return receiver.process_receive_operation()

```

310，获取 AST 的待推断的节点。

320，判断 AST 的根节点的类型是否已经推断。

从该 AST 中获取根节点，基于类型环境判断根节点的类型是否已经推断。

例如，如图 3 所示，从类型环境中读取根节点的类型变量的值，若类型环境中不存在根节点的类型变量的值，则根节点的类型变量未推断。若类型环境中存在根节点的类型变量的值，则根节点的类型变量已经推断。

以代码段#1 为例，从 AST 中获取代码段#1 的根节点，即 CustomConnection。基于类型环境判断 CustomConnection 的类型是否已经推断。

若根节点的类型已经推断，可以结束当前的推断过程。

进一步地，例如，如图 3 所示，若根节点的类型已经被推断，可以输出推断结果。例如，输出代码段#1 中的所有字段的类型。

若根节点的类型未被推断，执行步骤 330。

330，创建推理项。

若步骤 330 由步骤 320 触发，则创建根节点的推理项。

以代码段#1 为例，例如，在步骤 330 中可以创建 CustomConnection 的推理项，该推理项用于指示代码段#1 的推断任务。

若步骤 330 由步骤 380 触发，则创建依赖项的推理项。

340，将推理项添加至推理队列中。

例如，如图 3 所示，该推理队列中包括推理项 1、推理项 2、推理项 3 和推理项 4。

350, 从推理队列中提取首个推理项。

以图3所示的推理队列为例, 当前的首个推理项为推理项1, 从推理队列中提取推理项1。

从推理队列中提取出推理项后, 执行推理项所指示的推断任务。下面以根节点的推理项为例进行说明。

360, 执行类型变量初始化。

具体地, 为待推断的节点分别创建新的类型变量, 即唯一自由类型变量。

例如, 待推断的节点可以为类、静态变量、实例类变量、函数、函数参数或返回值等对应的节点。在步骤360中, 可以为类、静态变量、实例类变量、函数、函数参数或返回值等分别创建唯一自由类型变量。

以当前处理的推理项为 CustomConnection 的推理项为例, 为代码段#1 中的类、静态变量、实例类变量、函数、函数参数或返回值等分别创建唯一自由类型变量。

初始化处理后的 CustomConnection 类的代码片段可以表示如下代码段#2。

代码段#2

```

0  class CustomConnection(p):
1      _write :  $\eta$ 
2      _read :  $\lambda$ 
3
4      def _send(self : CustomConnection, buf :  $\phi$ , write :  $h$ ) -> T
5          ...
6
7      def _recv(self : CustomConnection, size :  $\alpha$ , receiver :  $\beta$ ) -> Y
8          ...

```

应理解, 以上代码段#2 仅为了直观展示类型变量的创建情况, 而并非将代码段#1 中的 CustomConnection 类的代码段实际修改为代码段#2。

其中, α , β , Y , λ , η , ϕ , h , T , p 均为唯一自由类型变量。待推理的节点包括: _ConnectionBase、_write、_read、方法_send、方法_recv、buf、write、size 和 receiver。例如, 为 _ConnectionBase 创建唯一自由类型变量 p , 两者之间的关系表示为 _ConnectionBase : p 。再如, 为 _write 创建唯一自由类型变量 η , 两者之间的关系表示为 _write : η 。以此类推, 其他待推理的节点以及唯一自由类型变量之间的对应关系分别表示如下: _read : λ ; buf : ϕ ; write : h ; 方法_send : T ; size : α ; receiver : β ; 方法_recv : Y 。

370, 获取约束。

下面以代码段#1 中的部分待推断的节点为例进行说明。

1) _ConnectionBase;

为 _ConnectionBase 创建基础类类型约束。

2) _write 和 _read;

2.1) _write 依赖 _multiprocessing, 为 _multiprocessing 创建解析引用约束。_read 依赖 _multiprocessing, 为 _multiprocessing 创建解析引用约束。_multiprocessing 的解析引用约束可以用于指示对 _multiprocessing 的引用。

2.2) 为 _multiprocessing.send 创建属性读取约束。_multiprocessing.send 的属性读取约束可以用于指示读取 _multiprocessing 的属性 send 返回的类型。为 _multiprocessing.recv 创建属性读取约束。_multiprocessing.recv 的属性读取约束可以用于指示读取 _multiprocessing 的属性 recv 返回的类型。

2.3) 为 “_write = _multiprocessing.send” 创建赋值约束。为 “_read = _multiprocessing.recv” 创建赋值约束。

3) 方法_send;

在方法_send 中, 参数 write 被赋值为默认值 (_write), 为 “write=_write” 创建默认值约束。

write 依赖 _write, 为 _write 创建解析引用约束, _write 的解析引用约束用于指示对 _write 的引用。

方法_send 中还可以创建其他约束, 此处不展开描述, 具体示例可以参考后文方法_recv 的描述。

4) 方法_recv;

4.1) 基于 `receiver.init_receiver(CustomConnection._read, size)` 创建如下约束。或者说, `receiver.init_receiver(CustomConnection._read, size)` 被分解为如下约束。

为 `receiver`、`CustomConnection` 和 `size` 分别创建解析引用约束。

为 `receiver.init_receiver` 和 `CustomConnection._read` 分别创建属性读取约束。

为 `(CustomConnection._read, size)` 创建参数传递约束。

为 `init_receiver(*)` 创建调用约束。

4.2) 基于 `return receiver.process_receive_operation()` 创建如下约束。或者说, `return receiver.process_receive_operation` 被分解为如下约束。

为 `receiver` 创建解析引用约束。

为 `receiver.process_receive_operation` 创建属性读取约束。

为 `process_receive_operation()` 创建调用约束。

为 `_recv` 返回值创建返回约束。

在步骤 370 中可以收集与步骤 360 中生成的类型变量相关的约束, 之后可以基于这些约束进行处理, 以得到这些类型变量的推断结果。

380, 依赖检查。

依赖检查包括处理当前推理项的解析引用约束。具体地, 检查当前推理项的所有依赖项的类型是否已经完成推断。

若存在尚未完成推断的依赖项, 则停止当前的处理, 转而推断该依赖项的类型。依赖项的类型可以采用本申请实施例中的方法获取。例如, 针对该依赖项执行步骤 330 至步骤 390, 以得到该依赖项的类型。

若不存在尚未完成推断的推理项, 例如, 在完成当前推理项的所有依赖项的类型的推断后, 执行步骤 390。

以步骤 370 中的约束为例, 假设 `CustomConnection` 的依赖项为步骤 370 中的解析引用约束所指示的所有依赖项。依赖项如下所示:

(a1) `_multiprocessing`, 来自 `_multiprocessing.send` 和 `_multiprocessing.recv`;

(a2) `_write`, 来自 `write=_write`;

(a3) `receiver`, `CustomConnection` 和 `size`, 来自 `receiver.init_receiver(CustomConnection._read, size)`;

(a4) `receiver`, 来自 `receiver.process_receive_operation()`。

其中 (a3) 和 (a4) 中的 `receiver` 为相同的依赖项。

判断 `_multiprocessing`、`_write`、`receiver`、`CustomConnection` 和 `size` 的类型是否均已完成推断。

若 `_multiprocessing`、`_write`、`receiver`、`CustomConnection` 和 `size` 中存在尚未完成推断的依赖项, 则停止当前的处理, 转而推断该依赖项的类型。

假设存在尚未完成推断的依赖项 `size`, 则停止当前的处理 (即 `CustomConnection` 的推理项的处理), 转而推断 `size` 的类型。例如, 创建 `size` 的推理项, 该推理项指示 `size` 的推断任务。将 `size` 的推理项添加到推理队列中。在取出 `size` 的推理项后获取约束, 并将根据约束获取 `size` 的推理结果。具体的推断过程可以参考步骤 330 至步骤 390 的描述, 为避免重复, 此处不再赘述。

以上仅为示例, 若 `_multiprocessing`、`_write`、`receiver`、`CustomConnection` 和 `size` 中存在多个尚未完成的依赖项, 则分别推断该多个依赖项的类型。例如, 若 `_multiprocessing`、`_write`、`receiver`、`CustomConnection` 和 `size` 的类型均未完成推断, 则可以分别创建 `_multiprocessing` 的推理项、`_write` 的推理项、`receiver` 的推理项、`CustomConnection` 的推理项和 `size` 的推理项, 并将这些推理项添加到推理队列中。在从推理队列中取出推理项后, 执行相应的推断任务。

在 `CustomConnection` 的推理项的所有依赖项完成推断后, 即在 `_multiprocessing`、`_write`、`receiver`、`CustomConnection` 和 `size` 的类型均完成推断后, 执行步骤 390。

390, 推断处理。

在步骤 390 中可以处理当前推理项的除了解析引用约束以外的其他约束, 以得到推断结果。

下面分别以代码片段 “`_read=_multiprocessing.recv`” 和方法 `_recv` 的代码片段为例进行说明。

1) `_read=_multiprocessing.recv`;

如前所述, 与上述代码片段相关的约束包括: `_multiprocessing` 的解析引用约束, `_multiprocessing.recv` 的属性读取约束, 以及 `_read = _multiprocessing.recv` 的赋值约束。

`_multiprocessing` 的解析引用约束已经在步骤 380 中进行了处理, 得到 `_multiprocessing` 的类型为 `_multiprocessing` 包。

根据 `_multiprocessing.recv` 的属性读取约束, 从 `_multiprocessing` 的类型 (即 `_multiprocessing` 包) 中确定属性 `recv` 函数的类型。

根据 `_read = _multiprocessing.recv` 的赋值约束, 将 “=” 左侧的类型和 “=” 右侧的类型进行联合。“=” 左侧的类型, 即 `_read` 的类型, 为唯一自由类型变量 Λ 。假设 “=” 右侧的类型为 B 类型, 则联合后的结果可以表示为 $\Lambda=\{B\}$, 表示唯一自由类型变量 Λ 包含 B 类型。

2) 方法 `_recv`;

2.1) `receiver._init_receiver(CustomConnection._read, size)`;

如前所述, 与代码片段 `receiver._init_receiver(CustomConnection._read, size)` 相关的约束包括: `receiver` 的解析引用约束, `CustomConnection` 的解析引用约束, `size` 的解析引用约束, `receiver._init_receiver` 的属性读取约束, `CustomConnection._read` 的属性读取约束, `(CustomConnection._read, size)` 的参数传递约束和 `init_receiver(*)` 的调用约束。

`receiver` 的解析引用约束, `CustomConnection` 的解析引用约束, `size` 的解析引用约束已经在步骤 380 中进行了处理。

根据 `CustomConnection._read` 的属性读取约束, 从 `CustomConnection` 的类型中确定属性 `_read` 的类型。

根据 `receiver._init_receiver` 的属性读取约束, 从 `receiver` 的类型中确定属性 `_init_receiver` 的类型。如前所述, `receiver` 是一个唯一自由类型变量 β 。当从唯一自由类型变量中请求属性的类型时, 可以确定该唯一自由类型变量包括该属性的类型。相应地, β 包括 `_init_receiver` 的类型。根据 `init_receiver(*)` 的调用约束确定 `_init_receiver` 的类型为可调用 (callable) 函数, 进而得到 $\beta=\{_init_receiver: \text{Callable}\}$, 表示 β 包括 `_init_receiver` 的类型, `_init_receiver` 的类型为可调用函数。根据 `(CustomConnection._read, size)` 的参数传递约束可以得到 `_init_receiver` 的参数类型。

2.2) `return receiver.process_receive_operation()`;

如前所述, 与代码片段 `return receiver.process_receive_operation()` 相关的约束包括: `receiver` 的解析引用约束, `receiver.process_receive_operation` 的属性读取约束, `process_receive_operation()` 的调用约束和 `_recv` 返回值的返回约束。

`receiver` 的解析引用约束已经在步骤 380 中进行了处理。

在 2.1) 中得到的 `receiver` 的类型表示为 $\beta=\{_init_receiver: \text{Callable}\}$ 。

根据 `receiver.process_receive_operation` 的属性读取约束, 从 `receiver` 的类型中确定属性 `process_receive_operation` 的类型。相应地, β 包括 `process_receive_operation` 的类型, 将 $\beta=\{_init_receiver: \text{Callable}\}$ 修改为 $\beta=\{_init_receiver: \text{Callable}, \text{process_receive_operation}\}$ 。

根据 `process_receive_operation()` 的调用约束确定 `process_receive_operation` 为可调用函数, 即 `process_receive_operation: Callable`。基于此对 `receiver` 的类型进行修改, 以得到 $\beta=\{_init_receiver: \text{Callable}, \text{process_receive_operation}: \text{Callable}\}$ 。

根据 `_recv` 返回值的返回约束确定方法 `_recv` 的返回类型为方法 `process_receive_operation` 运行时调用的返回类型。方法 `_recv` 的返回类型为唯一自由类型变量 Υ , 即 Υ 为方法 `process_receive_operation` 运行时调用的返回类型。

示例性地, 在基于上述约束处理完成后, 得到方法 `_recv` 的推断结果如下:

```
def _recv(
    self: CustomConnection,
    size: T0,
    receiver: { _init_receiver: Callable[[CustomConnection._read, T0], Any],
               process_receive_operation: Callable[[T1], T1] } -> T1
```

其中, $T0$ 表示 `size` 的类型, $T0$ 为未知类型; $T1$ 表示方法 `process_receive_operation` 的返回类型, 即方法 `_recv` 的返回类型, $T1$ 为未知类型。Any 表示未知类型。

`_init_receiver: Callable[[CustomConnection._read, T0], Any]` 表示 `init_receiver` 为可调用函数, 参数的类型为 `CustomConnection._read` 和 $T0$, 返回类型为未知类型。

`process_receive_operation: Callable[[T1], T1]` 表示 `process_receive_operation` 为可调用函数, 无参数, 返回

类型为 T1。

步骤 390 得到的推断结果可以存储在类型环境中。

400, 输出推断结果。推断结果可以指示待推断的节点的类型。

下面结合图 4 至图 7 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 4 是本申请实施例的编程语言的类型推断的装置的示意性框图。图 4 所示的装置 2000 可以用于执行图 1 或图 3 所示的方法。装置 2000 包括第一获取模块 2010、第二获取模块 2020 和处理模块 2030。

在一种可能的实现方式中, 装置 2000 可以用于执行图 1 所示的方法。

第一获取模块 2010, 用于获取抽象语法树中的待推断的节点, 待推断的节点对应于程序代码中的待推断的字段。待推断的字段为如下的一项: 变量、类、函数或参数。

第二获取模块 2020, 用于获取与待推断的节点相关的约束, 约束用于指示待推断的节点在抽象语法树中涉及的关系。

处理模块 2030, 用于根据约束确定待推断的节点的类型。待推断的节点的类型为如下的一项: 数字类型、序列类型、映射类型、集合类型、迭代器类型、上下文管理器类型、模块类型、元类型、空值类型或可调用类型。

第一获取模块 2010 和第二获取模块 2020 可以为同一模块, 也可以为不同模块。

可选地, 待推断的节点的类型用于以下至少一项: 显示待推断的节点的类型、代码补全、代码重构、代码检查或快速修复。

可选地, 第一获取模块 2010 具体用于从推理队列中取出第一推理项, 第一推理项用于确定待推断的节点的类型, 推理队列中包括一个或多个推理项; 获取待推断的节点。

可选地, 约束包括抽象语法树中的第一节点的解析引用约束, 第一节点的解析引用约束用于指示待推断的节点对第一节点的引用关系, 装置 2000 还包括第三获取模块 (图中未示出), 用于获取第一节点的类型, 以及处理模块 2030 具体用于根据第一节点的类型和约束确定待推断的节点的类型。

可选地, 约束包括第二节点的赋值约束, 第二节点的赋值约束用于指示第二节点涉及的赋值关系, 第二节点中被赋值的子节点的类型是根据赋值号右侧的类型确定的。

可选地, 约束包括第三节点的属性读取约束, 第三节点的属性读取约束用于指示第三节点涉及的从属关系, 第三节点中的属性的所有者的类型包括属性的类型。

可选地, 装置 2000 还包括创建模块 (图中未示出), 用于创建待推断的节点对应的类型变量; 以及处理模块 2030 具体用于根据约束确定待推断的节点对应的类型变量的值, 待推断的节点对应的类型变量的值用于指示待推断的节点的类型。

可选地, 装置 2000 还包括存储模块 (图中未示出), 用于将待推断的节点的类型存储至类型环境中, 类型环境用于存储抽象语法树中的节点的类型。

其中, 装置 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, 接下来以处理模块 2030 为例, 介绍处理模块 2030 的实现方式。类似的, 其他模块的实现方式可以参考处理模块 2030 的实现方式。

模块作为软件功能单元的一种举例, 处理模块 2030 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, 处理模块 2030 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, 处理模块 2030 可以包括至少一个计算设备, 如服务器等。或者, 处理模块 2030 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。

处理模块 2030 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。处理模块 2030 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, 处理模块 2030 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, 处理模块 2030 可以用于执行编程语言的类型推断的方法中的任意步骤, 第一获取模块 2010 可以用于执行编程语言的类型推断的方法中的任意步骤, 第二获取模块 2020 可以用于执行编程语言的类型推断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编程语言的类型推断的方法中不同的步骤来实现装置 2000 的全部功能。

本申请还提供一种计算设备 1000。如图 5 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 5 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) 之间传递信息的通路。

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第一获取模块 2010、第二获取模块 2020 和处理模块 2030 的功能, 从而实现编程语言的类型推断的方法。也即, 存储器 1006 上存有用于执行编程语言的类型推断的方法的指令。

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 6 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编程语言的类型推断的方法的指令。

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编程语言的类型推断的方法的部分指令。换言之, 一个或多个计算设备 1000 的组合可以共同执行用于执行编程语言的类型推断的方法的指令。

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, 分别用于执行编程语言的类型推断的装置的部分功能。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第一获取模块 2010、第二获取模块 2020 和处理模块 2030 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 7 示出了一种可能的实现方式。如图 7 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 中存有执行第一获取模块 2010 的功能的指令。同时, 计算设备 1000B 中的存储器 1006 中存有执行处理模块 2030 和第二获取模块 2020 的功能的指令。

图 7 所示的计算设备集群之间的连接方式可以是考虑到本申请提供的编程语言的类型推断的方法可能需要存储数据, 因此考虑将处理模块 2030 和第二获取模块 2020 实现的功能交由计算设备 1000B 执行。

应理解，图 7 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编程语言的类型推断的方法。

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编程语言的类型推断的方法。

最后应说明的是：以上实施例仅用以说明本申请的技术方案，而非对其限制；尽管参照前述实施例对本申请进行了详细的说明，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申请各实施例技术方案的保护范围。

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，可以存储在一个计算机可读存储介质中。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权利要求的保护范围为准。

权利要求书

1. 一种编程语言的类型推断的方法，其特征在于，所述方法应用于集成开发环境 IDE，所述方法包括：

在程序代码的开发过程中，获取抽象语法树中的待推断的节点，所述待推断的节点指示所述程序代码中的待推断的字段，所述待推断的字段为如下的一项：变量、类、函数或参数；

获取与所述待推断的节点相关的约束，所述约束用于指示所述待推断的节点在所述抽象语法树中涉及的关系；

根据所述约束确定所述待推断的节点的类型，所述待推断的节点的类型为如下的一项：数字类型、序列类型、映射类型、集合类型、迭代器类型、上下文管理器类型、模块类型、元类型、空值类型或可调用类型。

2. 根据权利要求 1 所述的方法，其特征在于，所述待推断的节点的类型用于以下至少一项：显示所述待推断的节点的类型、代码补全、代码重构、代码检查或快速修复。

3. 根据权利要求 1 或 2 所述的方法，其特征在于，所述获取抽象语法树中的待推断的节点，包括：

从推理队列中取出第一推理项，所述第一推理项用于确定所述待推断的节点的类型，所述推理队列中包括一个或多个推理项；

获取所述待推断的节点。

4. 根据权利要求 1 至 3 中任一项所述的方法，其特征在于，所述约束包括所述抽象语法树中的第一节点的解析引用约束，所述第一节点的解析引用约束用于指示所述待推断的节点对所述第一节点的引用关系，所述方法还包括：

获取所述第一节点的类型，以及所述根据所述约束确定所述待推断的节点的类型，包括：

根据所述第一节点的类型和所述约束确定所述待推断的节点的类型。

5. 根据权利要求 1 至 4 中任一项所述的方法，其特征在于，所述约束包括第二节点的赋值约束，所述第二节点的赋值约束用于指示所述第二节点涉及的赋值关系，所述第二节点中被赋值的子节点的类型是根据赋值号右侧的类型确定的。

6. 根据权利要求 1 至 5 中任一项所述的方法，其特征在于，所述约束包括第三节点的属性读取约束，所述第三节点的属性读取约束用于指示所述第三节点涉及的从属关系，所述第三节点中的属性的所有者的类型包括所述属性的类型。

7. 根据权利要求 1 至 6 中任一项所述的方法，其特征在于，所述方法还包括：

创建所述待推断的节点对应的类型变量；以及

所述根据所述约束确定所述待推断的节点的类型，包括：

根据所述约束确定所述待推断的节点对应的类型变量的值，所述待推断的节点对应的类型变量的值用于指示所述待推断的节点的类型。

8. 根据权利要求 1 至 7 中任一项所述的方法，其特征在于，所述方法还包括：

将所述待推断的节点的类型存储至类型环境中，所述类型环境用于存储所述抽象语法树中的节点的类型。

9. 一种编程语言的类型推断的装置，其特征在于，所述装置应用于集成开发环境 IDE，所述装置包括：

第一获取模块，用于在程序代码的开发过程中，获取抽象语法树中的待推断的节点，所述待推断的节点指示所述程序代码中待推断的字段，所述待推断的字段为如下的一项：变量、类、函数或参数；

第二获取模块，用于获取与所述待推断的节点相关的约束，所述约束用于指示所述待推断的节点在所述抽象语法树中涉及的关系；

处理模块，用于根据所述约束确定所述待推断的节点的类型，所述待推断的节点的类型为如下的一项：数字类型、序列类型、映射类型、集合类型、迭代器类型、上下文管理器类型、模块类型、元类型、空值类型或可调用类型。

10. 根据权利要求 9 所述的装置，其特征在于，所述待推断的节点的类型用于以下至少一项：显示所述待推断的节点的类型、代码补全、代码重构、代码检查或快速修复。

11. 根据权利要求 9 或 10 所述的装置，其特征在于，所述第一获取模块具体用于：

从推理队列中取出第一推理项，所述第一推理项用于确定所述待推断的节点的类型，所述推理队列中包括一个或多个推理项；

获取所述待推断的节点。

12. 根据权利要求 9 至 11 中任一项所述的装置，其特征在于，所述约束包括所述抽象语法树中的第一节点的解析引用约束，所述第一节点的解析引用约束用于指示所述待推断的节点对所述第一节点的引用关系，所述装置还包括第三获取模块，用于获取所述第一节点的类型，以及所述处理模块具体用于：

根据所述第一节点的类型和所述约束确定所述待推断的节点的类型。

13. 根据权利要求 9 至 12 中任一项所述的装置，其特征在于，所述约束包括第二节点的赋值约束，所述第二节点的赋值约束用于指示所述第二节点涉及的赋值关系，所述第二节点中被赋值的子节点的类型是根据赋值号右侧的类型确定的。

14. 根据权利要求 9 至 13 中任一项所述的装置，其特征在于，所述约束包括第三节点的属性读取约束，所述第三节点的属性读取约束用于指示所述第三节点涉及的从属关系，所述第三节点中的属性的所有者的类型包括所述属性的类型。

15. 根据权利要求 9 至 14 中任一项所述的装置，其特征在于，所述装置还包括创建模块，用于

创建所述待推断的节点对应的类型变量；以及

所述处理模块具体用于：

根据所述约束确定所述待推断的节点对应的类型变量的值，所述待推断的节点对应的类型变量的值用于指示所述待推断的节点的类型。

16. 根据权利要求 9 至 15 中任一项所述的装置，其特征在于，所述装置还包括存储模块，用于将所述待推断的节点的类型存储至类型环境中，所述类型环境用于存储所述抽象语法树中的节点的类型。

17. 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 8 中任一项所述的方法。

18. 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 8 中任一项所述的方法。

19. 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 8 中任一项所述的方法。