

[19] 中华人民共和国国家知识产权局



[12] 发明专利说明书

专利号 ZL 200410058970.3

[51] Int. Cl.

G06F 9/44 (2006.01)

G06F 9/45 (2006.01)

[45] 授权公告日 2009 年 4 月 15 日

[11] 授权公告号 CN 100478874C

[22] 申请日 2004.7.23

[21] 申请号 200410058970.3

[30] 优先权

[32] 2003.7.23 [33] US [31] 10/626,251

[73] 专利权人 微软公司

地址 美国华盛顿州

[72] 发明人 J·伯格 D·R·小塔迪蒂

C·L·米切尔 A·E·艾尔斯

V·K·格罗弗

[56] 参考文献

US6330717B1 2001.12.11

US5509123A 1996.4.16

US5920725A 1999.7.6

审查员 王 可

[74] 专利代理机构 上海专利商标事务所有限公司

代理人 陈 烽

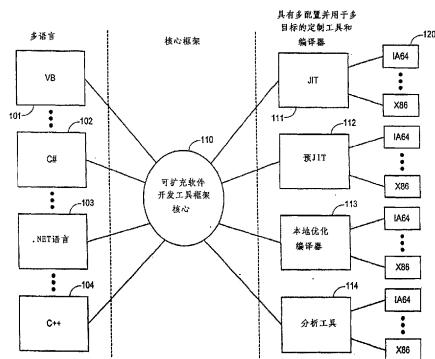
权利要求书 4 页 说明书 13 页 附图 14 页

[54] 发明名称

用于扩充配置相关可扩充软件的方法和系统

[57] 摘要

描述了用于扩充配置相关可扩充软件程序的方法和系统。可以通过添加配置相关扩展来扩充核心可扩充软件程序的类。扩展可在编译软件程序的核心版本之前静态地添加，或者在运行时刻动态地添加。可扩充核心类的声明可包括具有该类是静态可扩充还是动态可扩充的指示的可扩充属性。也描述了用于适当地声明配置相关可扩充类的对象描述语言。也描述了用于处理该对象描述语言来生成可扩充类声明及其扩展的源代码表示的预处理器。源代码表示然后可以用来生成核心软件程序的扩充版本。



1. 一种生成以面向对象编程语言书写的软件的扩充版本、其通过多个对软件的扩展来提供对象类的方法，其特征在于，所述方法包括：

接收包括对要通过所述软件的扩充版本实现的相应软件开发情形的扩展的多个软件开发情形类扩展集的调用，其中所述相应软件开发情形指定多个目标处理器体系架构和多个编译情形；以及

如所述扩展所指示的扩充所述软件的一个或多个类。

2. 如权利要求 1 所述的方法，其特征在于，所述接收所述扩展集的至少一个出现在所述软件的运行时刻。

3. 如权利要求 1 所述的方法，其特征在于，所述扩充包括对所述扩展输出源代码。

4. 如权利要求 1 所述的方法，其特征在于，所述扩展集包括：

用于实现目标体系结构的扩展集。

5. 如权利要求 4 所述的方法，其特征在于，所述扩展集包括：

用于实现编译情形的扩展集。

6. 如权利要求 4 所述的方法，其特征在于，所述扩展集包括用于实现管理代码情形的扩展集，所述方法还包括：

基于所述接收，扩充所述类来提供管理代码功能。

7. 如权利要求 1 所述的方法，其特征在于，对所述扩展集指示的至少一个的调用在运行时刻接收。

8. 如权利要求 1 所述的方法，其特征在于，所述扩展的至少一个指示对所述软件的所述对象类的至少一个的额外的类成员。

9. 一种通过向软件的核心版本添加扩展来扩充以编程语言书写的软件以生成所述软件的扩充版本的方法，其特征在于，所述方法包括：

接收所述软件的扩充版本的配置；

依照所述软件的扩充版本的配置以对象描述语言接收对所述软件的核心版本的类的扩展的定义，其中，所述软件的核心版本的类以所述对象描述语言指示为在编译时刻之前静态可扩充或在运行时刻动态可扩充，其中指示为静态可扩充的所述软件的核心版本的类与其相应的扩展一起处理，以及其中指示为动态可扩充的所述

软件的核心版本的类与其相应的扩展分开处理；以及

处理所述软件的核心版本的类以及所述扩展来生成所述软件的扩充版本，其中与其相应的扩展一起处理所述软件的核心版本的类包括：

使用对象描述语言预处理器来生成具有包括所述软件的核心版本的类及其相应的扩展的扩充类的源代码表示的头文件；以及  
编译所述头文件来生成所述软件的扩充版本。。

10. 如权利要求 9 所述的方法，其特征在于，与其相应的扩展分开处理所述软件的核心版本的类包括：

使用对象描述语言预处理器来生成包括所述软件的核心版本的类的源代码版本的头文件；以及

编译所述头文件来生成所述软件的核心版本的类的计算机可执行版本。

11. 如权利要求 9 所述的方法，其特征在于，与其相应的扩展分开处理所述软件的核心版本的类包括：

使用对象描述语言预处理器来生成包括对所述软件的核心版本的类的扩展的源代码版本的头文件；以及

编译所述头文件来生成对所述软件的核心版本的类的所述扩展的计算机可执行版本。

12. 如权利要求 11 所述的方法，其特征在于，它还包括处理对所述软件的核心版本的类的所述扩展的计算机可执行版本以通过在运行时刻将所述软件的核心版本的类连接到其相应的扩展来生成扩充类。

13. 如权利要求 9 所述的方法，其特征在于，所述软件的核心版本的类的定义包括用于指示在与所述软件的核心版本相关的代码内注入与对所述软件的核心版本的类的所述扩展相关的代码的点的一个或多个扩展点。

14. 如权利要求 9 所述的方法，其特征在于，所述软件的核心版本是可扩充核心编译器框架，并且所述软件的扩充版本是定制编译器，并且接收所述软件的扩充版本的配置包括获取编译器类型。

15. 如权利要求 14 所述的方法，其特征在于，所述编译器类型从包括 JIT 编译器、预 JIT 编译器和本地优化编译器的一个组中选择。

16. 如权利要求 9 所述的方法，其特征在于，所述软件的核心版本是可扩充核心软件开发工具框架，并且所述软件的扩充版本是定制软件开发工具，并且接收

---

所述软件的扩充版本的配置包括获取目标类型。

17. 如权利要求 9 所述的方法，其特征在于，所述软件的核心版本是可扩充核心软件开发工具框架，并且所述软件的扩充版本是定制的软件开发工具，并且接收所述软件的扩充版本的配置包括获取特征类型。

18. 一种通过向软件的核心版本添加扩展来生成所述软件的扩充版本以扩充软件的系统，其特征在于，所述系统包括：

对象描述语言预处理器，适用于以对象描述语言接收对所述软件的核心版本的类的扩展、并生成对所述软件的核心版本的类的扩展的源代码版本，其中，所述软件的核心版本的类在所述对象描述语言中指示为在运行时刻动态可扩充或在编译时刻之前静态可扩充，其中所述对象描述语言预处理器被编程为用于与其相应的扩展分开地处理指示为动态可扩充的所述软件的核心版本的类，以及其中所述对象描述语言预处理器适用于与其相应的扩展一起处理指示为静态可扩充的所述软件的核心版本的类，来生成具有包括所述软件的核心版本的类及其相应扩展的源代码版本的扩充类的源代码版本的头文件。

19. 如权利要求 18 所述的系统，其特征在于，它还包括用于编译所述扩充类的源代码版本来生成用于生成所述软件的扩充版本的所述扩充类的计算机可执行版本的编译器。

20. 如权利要求 18 所述的系统，其特征在于，所述软件的核心版本的类还包括用于指示在与所述软件的核心版本相关的代码内注入与所述扩展相关的代码的位置的扩展点。

21. 如权利要求 18 所述的系统，其特征在于，它还包括用于编译对所述软件的核心版本的类的所述扩展的源代码版本来生成所述扩展的计算机可执行版本的编译器。

22. 如权利要求 21 所述的系统，其特征在于，它还包括用于执行所述扩展的计算机可执行版本以在运行时刻通过将所述软件的核心版本的类连接到其相应的扩展来生成所述软件的扩充版本的计算机处理器。

23. 如权利要求 22 所述的系统，其特征在于，所述处理器通过在运行时刻调用所述软件的核心版本的计算机可执行版本并将所述扩展注入到与所述软件的核心版本相关的代码中来执行所述扩展的计算机可执行版本。

24. 如权利要求 18 所述的系统，其特征在于，所述扩展对应于所述软件的扩

---

充版本的配置。

25. 如权利要求 24 所述的系统，其特征在于，所述软件的核心版本是可扩充的核心软件开发工具框架，并且所述软件的扩充版本是定制的软件开发工具，并且所述软件的扩充版本的配置包括目标类型。

26. 如权利要求 24 所述的系统，其特征在于，所述软件的核心版本是可扩充的核心软件开发工具框架，并且所述软件的扩充版本是定制的软件开发工具，并且所述软件的扩充版本的配置包括编译器类型。

27. 如权利要求 24 所述的系统，其特征在于，所述软件的核心版本是可扩充的核心软件开发工具框架，并且所述软件的扩充版本是定制的软件开发工具，并且所述软件的扩充版本的配置包括特征类型。

28. 一种用于通过向软件的核心版本添加扩展来生成所述软件的扩充版本以扩充所述软件的系统，其特征在于，所述系统包括：

用于以对象描述语言接收对所述软件的核心版本的类的扩展并生成对所述软件的核心版本的类的所述扩展的源代码版本的装置，其中，所述软件的核心版本的类在所述对象描述语言中指示为运行时刻动态可扩充或编译时刻之前静态可扩充，其中指示为动态可扩充的所述软件的核心版本的类与其相应的扩展分开处理，并且其中指示为静态可扩充的所述软件的核心版本的类与其相应的扩展一起处理，以生成具有包括所述软件的核心版本的类及其相应扩展的源代码版本的扩充类的源代码版本的头文件；以及

用于编译用于生成所述软件的扩充版本的对所述软件的核心版本的类的扩展的所述源代码版本的装置。

## 用于扩充配置相关可扩充软件的方法和系统

### 技术领域

技术领域涉及可扩充软件系统，尤其涉及可扩充类的使用。

### 背景技术

由于多编程语言、多样的处理器以及多操作系统环境的不断增加，计算的领域每一天都在变得越来越复杂。现在有许多具有特殊能力的编程语言（如，C++、Java、C#）可用来向程序员在编写各种计算任务的程序中提供特殊的优点。类似地，有各种处理器（如 X86、IA-64、AMD 等等）可用来对执行特定的任务提供特殊的优点。例如，嵌入式处理器尤其适合处理电子设备内较好地定义的任务，而诸如 Intel® Pentium® 处理器的通用处理器更为灵活且能够处理复杂的任务。因此，计算环境、配置和设备的多样性正在增加。

对多样性的增加的需求将构建编译程序的已高度复杂的领域复杂化。传统地，书写编译程序来编译以特定源代码语言书写的软件，并以特定类型的处理器体系结构（如 IA-64、X86、AMD、ARM 等等）为目标。近来，引入了翻译程序，将以多源代码语言书写的程序转换为单个中间语言表示（如，CIL（C++中间语言）和 MSIL（用于.NET 的 Microsoft® 中间语言））。然而，改变若干不同类型的目标体系结构中的一个源代码程序的编译的目标仍是复杂且耗时的。

减小对诸如多语言和多目标的多软件开发情形构建编译器和其它软件开发工具（如，分析工具、优化器）的复杂性的一个合适的方法是开发一个可扩充核心基础结构或框架，可以向其添加软件扩展来构建特殊配置的编译器和其它软件开发工具。例如，如果用户想要构建为特定目标体系结构配置的及时（JIT，运行时编译执行的技术）编译器，可以通过重新使用核心编译器框架的代码并使用对 JIT 编译器类型情形特定的代码添加扩展来生成 JIT 编译器以简化他或她的任务。对其它软件开发工具，如优化工具、分析工具、性能工具、测试工具等等，也可以设想同一方法。

使用这一可扩充核心编译器和工具框架来构建这一定制编译器和其它软件开

发工具并没有其自己的复杂性集合。这对根据要构建的语言、目标体系结构和编译器类型（如，JIT、预 JIT、本地优化编译器）可以以多种不同的方法配置来反映多种不同软件情形的可扩充编译器和工具框架来说尤为真实。一个这样的复杂性涉及以可扩充方式定义核心框架的数据结构（如，面向对象语言中的对象类），使能够添加依赖于多个不同软件情形的扩展域来扩充核心框架的数据结构。能够使用向类定义添加扩展域的传统技术，但是这仅以由降低的性能和增加的代码复杂度所付出的重大代价来实现，导致增加的开发和维护成本。

因此，需要一种简单而有效的方法，使用依赖于多个不同的可能的软件开发情形的多个不同类扩展来扩充核心框架软件系统的对象类。

### 发明内容

这里描述了通过提供配置相关扩充类来扩充软件程序的方法和系统。在一个方面，可以提供并添加依赖于各种软件开发情形的类扩展来扩充核心类。可以组合各种类扩展来开发特别配置的类。在一个方面，可以将核心软件程序的类定义为静态或动态可扩充。如果核心类被声明为静态可扩充，可以生成组合核心类定义及其相应的类扩展的头文件，并对它们进行共同编译来生成扩充类。可以使用这一扩充类来以配置相关方式扩充核心软件程序。

然而，如果核心类被声明为运行时刻动态可扩充，则可以生成包括该核心类声明的单独的头文件和包括扩展声明的单独文件。然后分开编译对应于核心类的头文件和对应于类扩展的文件来生成计算机可执行文件，其相互之间有连接，使能够在运行时刻添加类扩展来扩充核心类。

在另一方面，可以在核心类声明中提供扩充点，来特别指示核心类声明中应当注入类扩展的点。这里也描述了用于定义可扩充类和类扩展的采用合适语法的对象描述语言。

这里还描述了能够以对象描述语言接收输入并以源代码表示生成输出来生成软件程序的扩充版本的预处理器程序。在另一方面，该预处理器能够以最终能够被编译为可由计算机执行的形式的任一语言生成输出。

### 附图说明

图 1 所示是示例性配置相关可扩充核心软件框架的结构图。

图 2A 所示是在示例性面向对象编程语言中实现为类和对象的数据结构的结构图。

图 2B 所示是可扩充核心软件程序的类及其相应的扩展之间的关系的结构图。

图 3A 所示是通过添加类扩展来扩充类定义的总方法的流程图。

图 3B 所示是生成核心软件程序的配置相关扩充版本的总方法的流程图。

图 4A 所示是核心软件框架的扩充版本的结构图，其中，该扩充在编译时刻之前静态地实现。

图 4B 所示是核心软件程序的扩充版本的结构图，其中，该扩充在运行时刻动态地实现。

图 5 所示是用于静态地扩充核心软件程序的方法的流程图。

图 6 所示是如图 5 所示静态地扩充核心软件程序的系统的结构图。

图 7 所示是用于动态地扩充核心软件程序的方法的流程图。

图 8 所示是如图 7 所示动态地扩充核心软件程序的系统的结构图。

图 9A 所示是对象描述语言中静态可扩充核心类的核心类声明的清单。

图 9B 所示是对图 9A 的静态可扩充核心类声明的两个类扩展的清单。

图 9C 所示是通过将图 9A 的核心类声明与图 9B 的类扩展相关联而生成的扩充类声明的源代码表示的清单。

图 10A 所示是对象描述语言中动态可扩充核心类的核心类声明的清单。

图 10B 所示是对图 10A 的动态可扩充核心类声明的扩展的清单。

图 10C 所示是通过将图 10A 的核心类声明与图 10B 的类扩展相关联而生成的扩充类声明的源代码表示的清单。

图 11A 所示是指示用于注入类扩展的扩展点的核心类声明的清单。

图 11B 所示是对图 11A 的核心类定义的扩展的清单。

图 12 所示是依照多个软件开发情形构造软件开发工具的示例性实现的结构图。

## 具体实施方式

### 示例性软件开发工具

尽管这里描述的多种技术已经通过使用编译器的示例初步地说明，然而可以结合其它软件开发工具（如调试器、优化器、反汇编器、仿真器以及软件分析工具）

使用这些技术的任一种。

### 可扩充软件开发工具框架

图 1 说明了可以被扩充来构建多个不同配置的自定义编译器和其它软件开发工具来反映多个软件开发情形的示例性核心软件框架。核心 110 提供了可用作构建块来构建定制软件开发工具 111 - 114 的可扩充体系结构。可以通过添加与一个或多个软件开发情形相关的软件扩展来扩充核心 110 软件。例如，可以通过向核心 110 提供软件扩展来构建以 IA-64 处理器 121 为目录的 JIT (及时) 编译器 111。在这一情况下，编译器为 JIT 编译器 111 并且以特定的处理器体系结构 (121 的 IA-64 处理器) 为目录的事实可以确定对核心 110 的软件扩展的形式和内容。由此，可以使用与 JIT 编译器情形和 IA-64 目标情形相关的软件扩展来指定用于构建定制软件开发工具的配置。其它情形因素，如源语言 101 - 104 以及可以基于特定软件开发情形而开启或关闭的工具的特征，也可能通过扩充诸如图 1 所示的标准核心框架向构建自定义软件开发工具的任务添加复杂度。

### 示例性软件开发情形

有许多软件开发情形影响要结合进核心框架 110 的软件扩展的选择。例如，特定软件开发工具的软件开发情形可包括该工具将用作目标的各种处理器体系结构 (如 IA-64、X86、AMD、ARM 等等)。软件开发情形也可以与执行的编译类型 (如，JIT、预 JIT、本地优化编译器) 相关。软件开发情形也可以与由软件开发工具执行的其它类型的功能，如分析、优化、仿真、调试、代码生成等等的类型相关。另一软件开发情形可以与特定的编程语言 (如 Java、C++、C# 等等) 相关，软件开发工具可以特别地为这些编程语言配置。此外，软件开发情形也可以与该工具是否与管理执行环境 (如，由微软.NET 框架提供的微软 CLR 环境) 使用相关。上述示例提供了可影响适当地扩充核心框架 110 所需要的扩展的选择的软件开发情形或因素的有限集合。类似地，其它情形也能够影响配置自定义软件开发工具所需要的软件扩展的选择。这类情形的集合可以被称为配置。然而，自定义软件开发工具的特定配置可能被单个情形所影响。

### 示例性对象

在面向对象编程中，可以使用对象来储存数据及对该数据的访问功能。通过提供类定义来定义对象，然后可以使用类定义来例示属于该类的对象。

图 2A 说明了对具有变量或数据成员（例如，或者域）201A - C 和方法 202A - C 的示例性类型“自行车”的类定义 201。类 201 的一个具体实例是对象 202。一般而言，数据成员可以描述对象的状态，而方法用来向对象授予行为。在该示例中，当例示类 201 的对象 202 时，对变量 201A - C 给予具体值 202A - C。数据成员和方法有时候也共同称为“类成员”。

类似地，可以提供对诸如 110 的可扩充核心框架的类定义来描述实现核心 110 所需要的数据结构。此外，为依照各种软件开发情形扩充核心框架 110，可以根据这一软件开发情形改变或扩充核心类。

### 示例性软件扩充

依照选择的软件开发情形扩充核心软件框架的一种方式是扩充软件类。能够通过改变对象类的定义来扩充软件类。

示例性类扩展可包括一个或多个类扩展成员（如数据成员或方法）。当扩充该类时，修改类定义来包括指定的类扩展成员。

由此，可以通过定义额外的类成员并将其结合进其核心类定义中来扩充核心类。这些额外的类成员可以被称为扩充核心类的类扩展成员。总起来说，这些类扩展成员可以被称为“类扩展”。

类扩展可以以许多不同的方式变化。例如，某些类扩展所需要的特定类成员可能不被其它类所需要。同样，特定类扩展内的方法、函数和接口可能在其它类中完全不存在，或者如果存在，可能不同地定义。这些类扩展成员可与软件开发情形相关。例如，要向编译器框架的核心类添加来构建 JIT 编译器 111 的附加域或类成员可以远少于与本地优化编译器 113 相关的类扩展所需要的域或类成员数。

### 示例性软件开发情形类扩展集

一旦开发者确定了用于配置软件开发工具的软件开发情形，可以指定其相应的类扩展来开发核心软件框架的扩充版本。合适的软件（如处理器或编译器）然后能够接收指定的情形，并包括对该情形合适的扩展的集合。由此，用于特定软件开发情形的类扩展能够组合成具有一个或多个对该情形合适的类扩展的软件开发情

形类扩展集。当开发软件开发工具时，可以调用该软件开发情形类扩展集来扩充合适的类。在开发过程中或在运行时刻可以扩充该类。

例如，使用各种软件类来实现核心 110。如果开发者指定 JIT 编译和 IA-64 目标体系结构的软件开发情形，与 JIT 编译器相关的类扩展和与 IA-64 目标体系结构相关的类扩展被结合进核心类定义中来扩充生成核心框架的扩充版本中所使用的核心框架 110 的类。

#### 解释核心类及其扩展之间的关系

图 2B 所示是核心类定义及其类扩展之间的关系。核心节点 210 可与核心类的定义相关，并示出具有类成员 1 和 2。220 的扩充类定义可对实现特定软件开发情形必须，并且可通过扩展 225 添加额外的类成员 3 和 4。扩充类定义 230 与 220 相同，可以通过在扩展 225 中向核心类定义 210 添加相同的额外的成员 3 和 4 来生成。然而，扩充的类定义 260 是不同的，并且可以通过在扩展 265 中添加额外的类成员 5 和 6 生成。以类似的方式，可以通过分别向类定义 220 和 230 添加扩展 245 和 255 来扩充扩充类定义 240 和 250。同样，可以通过向扩充类定义 260 添加扩展 255 来生成扩充类定义 270。

在图 2B 的示例中，说明类扩展具有多于一个类扩展成员。然而，类扩展可以具有一个或多个类扩展成员，或可以替换核心类定义的一个或多个现有类成员。此外，类扩展可以是以核心类定义中找到的方法成员的函数的形式。

由此，对核心软件开发工具框架的类定义的扩展可以如图 2B 所示，额外的类扩展依赖编译情形（JIT、预 JIT、本地优化编译器等等）、语言情形（C#、Visual Basic 等等）、目标情形（IA-64、X86、AMD、ARM 等等）以及可影响图 1 的示例性可扩充软件开发工具框架的扩充版本的特定配置的其它变量。

然而，表示或解释对可能依赖于多个软件开发情形（即，目标、编译等等）的具有特定配置的核心框架的类扩展能够立即变得十分复杂。例如，图 2B 描述了一些简化的扩展情况，借此扩充类定义 220、230、240 和 250 以有序的方式从先前定义的类继承，而不使用任一单个扩展来扩充多个父类。然而，当表示对软件开发工具的特定配置的扩充类时可能就不是这一情况，类的层次中的不同层级可能需要以多个不同的方法扩充。

例如，可能需要使用一些相同的扩展来扩充多个父类，而不是扩充单个父类。

例如，可以使用扩展 255 来扩充类定义 230 以及类定义 260。由此，根据软件开发情形，可使用同一扩展来扩充不同的父类。当与软件开发情形的绝对数字和相关的扩展配置相组合时的这一多继承的可能性为计算机程序员带来的巨大的负担。

在一种方法中，可以将核心类定义和扩充核心类定义之间的关系编程为基类及其子类之间的继承的链。然而，这一方法往往导致类膨胀，并且很快对程序员来说变得繁重。采用这一方法，程序员不仅有手动地开发扩展子类的任务，而且他或她还有拆毁未使用的对象来管理有限存储器资源的使用的附加任务。

扩充基类定义的另一手动方法可以是使用 IF-DEF 语句或其它用于实现条件编译的方法。在这一方法中，程序员可设置一个基类定义，并手动地添加 IF-DEF 语句或其它条件编译语句，在其体中包括扩展的定义来条件地向基添加扩展。这一方法的确具有仅当需要时扩充类的优点，并且因此可以是比导致不需要的额外开销的对每一扩展生成新子类更好的方法。然而，这一方法实际上也是手动的，并且需要程序员对每一可能的配置在许多位置包括 IF-DEF 语句，导致代码由于大量这类 IF-DEF 语句而变得杂乱。

#### 使用对象描述语言生成类扩展及其对核心类关联的示例性总过程

图 3A 描述了通过扩充核心框架扩充核心类定义来构建编译器或工具的总过程。首先，在 302 遇到指示扩展的数据，并且在 304 如扩展所指示的扩充软件开发工具的类。

图 3B 描述了通过使用用于扩充核心框架的软件情形相关扩展来构建编译器或软件开发工具的总过程。在 310，可以使用简化的对象定义语言（ODL）来定义核心类。然后在 320，基于软件开发情形确定对特定软件开发工具的配置，软件开发情形包括其编译器类型情形以及对其进行构建的特定目标情形。然后，基于影响配置的每一情形因素，在 330 可以使用对象描述语言定义扩展，来表示扩充该核心类所需要的额外的或不同的类扩展成员。在 340，可以将该扩展与核心类相关联，来适当地扩充核心类定义。对象描述语言的语法应当提供定义核心类为可扩充与否并进一步将类扩展成员的特定集合关联为选择的核心类的扩展。对这一描述语言合适的语法在下文使用示例进一步描述。此外，可以使用预处理器翻译程序以将数据或对象描述语言翻译为编程语言的源代码。在这一预处理之后，在 350，可以进一步处理扩充类定义并使用其通过扩充核心框架来实现特定配置的编译器或其它软

件开发工具。

使用上述过程，能够单独地提供扩展的多个不同定义，并且每一扩展能够简单地在需要时扩充核心或基类，而不需要维护任一复杂的继承关系。提供核心类的特定扩展的程序员不需要知道该核心类的其它扩展。这不仅简化了定义扩展的任务，而且也使扩充的核心类的用户仅需要知道核心类名来使用该扩充的核心类。由此，当使用扩充类定义时，程序员可以解除记住类定义间复杂的继承关系的任务。

#### 在运行时刻动态地扩充核心框架程序

#### 以及在编译核心程序之前静态地扩充核心框架程序

扩充核心框架程序的一种方法可以是获取对核心程序的源代码文件的访问以及在需要时通过使用对象描述语言定义扩展来静态地扩充核心类，然后可以处理该扩展来生成与扩充类相关的源代码。可选地，可以通过手动地以源代码编程语言向源代码直接添加扩展来生成扩充类。图 4A 说明了这一方法，其中向核心框架文件 410 添加扩展 420、430 和 440 来对其进行扩充，然后将扩展 420、430 和 440 作为现在已扩充的核心框架文件 410 的一部分来编译。

然而，这一方法可能不是对所有目的都合适，因为提供诸如 420、430 和 440 的扩展的定义的程序员需要具有对核心框架 410 的源代码的访问。在核心框架 410 的提供者希望保持核心框架源代码隐秘或不改变的情况下，这可能是不期望的。在这一情况下，可以使用图 4B 描述的第二种方法，其中，核心编译器和工具框架 450 作为从扩展 460、470 和 480 分离的文件来编译。

在第二种方法中，扩展 460、470 和 480 以及核心框架 450 可以适用于彼此具有连接，使得在运行时刻将扩展连接到核心框架，以适当地扩充核心框架。连接可以实现为指定要使用哪一些扩展来扩充特定核心类的简单的连接列表。这也可以通过当需要时使用适当地将扩展关联到核心类的简单命名约定来实现。与第一种方法相比较，第二种方法可能需要与运行时刻连接方面相关的额外开销处理，并且因此可能是较慢的实现。另一方面，第二种方法的确提供允许不具有对核心框架的源代码的访问的开发者扩充核心类的灵活性。

#### 在编译之前静态地扩充核心类的示例性方法

图 5 说明了如参考以上图 4A 所示的在编译时刻之前静态地扩充与核心框架程

序相关的类的方法。可以使用对象描述语言来定义核心类及其扩展。核心类和类扩展的定义不需要同时或共同生成。然而，添加类扩展需要对核心程序的源代码的一些访问。一旦获取这一类定义，然后在 510，核心类及其扩展的定义将由 ODL 预处理器共同处理，ODL 预处理器能够将对象描述语言表示翻译为源代码表示。因此在 520，由 ODL 处理器预处理的结果可以是头文件并可能是以诸如 C++ 的源代码语言表示核心类及其扩展的定义的一些其它代码。进一步在 530，然后与涉及现在已扩充的核心框架的代码的剩余部分一起编译具有包括核心类成员和类扩展成员的扩充类定义的头文件，来生成自定义配置的编译器以及其它软件开发工具。

图 6 说明了实现图 5 的过程的示例性系统。如图 6 所示，对核心类定义 620 的扩展 610 的多个定义可以储存为对象描述语言文件。可以提供能够接收分别与核心类定义和类扩展定义对应的文件 610 和 620 的 ODL 预处理器 630。预处理器也应当能够将文件 610 和 620 从其对象描述语言形式翻译为源代码表示 640。源代码表示能够以最终能够被编译为可由计算机处理器执行的形式的任一语言的形式。预处理器 630 生成的源代码 640 可包括通常储存类定义的头文件。可以提供适合预处理器 630 产生的源代码 640 的语言的源代码编译器 650 用于编译源代码表示 640 来创建诸如编译器和其它软件开发工具的核心软件程序的定制扩充版本。

#### 在运行时刻动态地扩充核心类的示例性方法

图 7 说明了通过在运行时刻将扩展连接到适当的核心类来扩充可扩充核心框架软件程序的核心类定义的方法。可以使用对象描述语言单独地解释核心类定义和扩展。描述语言可以适合解释一核心类定义是动态可扩充的。这一语言也可以适合解释特定核心类定义及其扩展之间的关联。一个这类合适的语言的语法在后文进一步详细描述。一旦解释了定义，在 710 可以使用 ODL 预处理器以将对象描述语言表示的定义翻译为 720 的源代码表示。然而，与静态过程（图 6）不同，在图 7 的动态过程中，ODL 预处理器不与其扩展的定义一起共同处理核心类定义。作为替代，单独地生成对应于核心类定义的源代码头文件和对应于类扩展定义的源代码头文件。可以由不同的 ODL 预处理器生成这些头文件，但这样做不是必须的。此外，在 730，单独地编译包含核心类定义的头文件和包含类扩展定义的头文件来创建可由计算机执行的单独文件。然而，在 740，在运行时刻，可以将类扩展定义连接到适当的核心类定义来如所定义地扩展核心类。

图 8 说明了实现图 7 的过程的示例性系统。如图 8 所示，以对象描述语言提供了类扩展定义并将其储存在文件 810 中。不需要将每一类扩展储存为如图所示的单独文件。也以对象描述语言提供了核心类定义并储存在文件 820 中。依照图 7 描述的过程，提供了 ODL 预处理器 825，用于通过将核心类定义从对象描述语言表示翻译为将储存为头文件 835 的源代码语言表示来处理核心类定义。类似地，可以提供另一 ODL 预处理器 830，用于处理类扩展文件 810 来生成包括类扩展的源代码头文件 840。可以提供源代码编译器 845，用于编译类扩展头文件 840 来生成包含类扩展定义的计算机可执行文件 860。类似地，可以提供源编译器 850，用于编译包含核心类定义的头文件 835 来生成包含核心类定义的计算机可执行文件 855。然后在运行时刻，当执行对应于核心类 855 的可执行文件和对应于类扩展的可执行文件时，在核心和扩展类中提供的连接 870 可适当地扩充核心类。

#### 提供静态可扩充类定义的对象描述语言

如上所述，可以使用简单的对象描述语言来提供依赖于可扩充核心框架程序的期望配置的类定义。例如，核心框架的特定配置可基于软件开发情形，如编译器类型（如，JIT、预 JIT、本地优化等等）、目标类型（如，IA-64、X86、ARM 等等）以及其它情形因素对类定义有全异的需求。以下几部分描述了这一可由 ODL 预处理器处理来以如上述参考图 5 到 8 所描述的源代码语言生成类定义的语言。

图 9A 描述了使用对象描述语言的标准类声明。一般而言，声明 900 包括头 911 和体 912。头可以用来声明类的可见性（如，公有、私有等等）、类名等等。如在 913 所示，头也包括如以下示例由方括号包围的属性定义：

[attribute\_name] （属性\_名称）

这一声明可用来定义类的属性，如是否为管理类（如，在.NET 情形中）以及类的可扩充性属性。例如，可以通过声明 [gc] 属性来指示自动完成对该类的对象的无用单元收集来定义该类是否为管理类。对于类可扩充性属性，一种方法是假定所有类都如上所述地为静态可扩充。由此，不需要特殊的属性来特别地声明一个类为静态可扩充。然而，如图 10A 所示，要在运行时刻（即，动态）扩充的类可以使用诸如 [extensible]（可扩充）的特定属性来声明。

在图 9A 所示的示例中，类 SYM 是静态可扩充类，在其体中具有核心类成员 914 和 915。类 900 可以是核心类，类成员 TYPE 914 和 NAME 915 可以是对依赖

于核心类的所有配置相关扩充类公用的类成员。一旦定义了核心类 900，可以提供用于对核心框架的特定配置生成核心类定义的扩充版本的类扩展。

图 9B 以对象描述语言示出了两个这样的类扩展定义 920。为简明性，类扩展的形式很类似于核心类声明或定义的形式。扩展 925 示出了类成员 926 与用于从核心框架构建 JIT 编译器的 JIT 编译情形相关。JIT 编译情形可能需要特定的类成员 926，如对 JIT 编译器配置特定但对其它配置不是必需的接口、方法和变量。由此，可以提供扩展 925，当由 ODL 预处理器处理时，扩充图 9A 的核心类定义。927 的关键字“extend”指示这是对核心类 SYM 的静态扩展。同时，属性说明[JIT] 928 指示仅当扩充核心框架来实现 JIT 编译器时应用该扩展。类似地，可以提供扩展 930 来添加对构建用于以 IA-64 处理器为目标的工具特定的类成员 931。两个扩展 925 和 930 彼此不相关，可以通过不同方提供，并且这些扩展不需要如使用传统的类—子类依赖性来扩充核心类定义的情况那样以任一方式相互依赖。此外，程序员不需要跟踪复杂的依赖性关系。

可以提供与其它因素有关的其它扩展。例如，在 C++的中间语言中，可能需要将两个函数符号连接在一起。可以将该函数添加为对配置来处理 C++的中间语言的软件开发工具特定的扩展，其中，该扩展可包括用于将两个函数符号连接在一起的方法。

当 ODL 预处理器生成如图 9C 所示的扩充类定义的源代码表示时，可以实现示例性 JIT 编译器相关的扩展 925 和示例性 IA-64 目标相关的扩展 930 的实际的条件实现。示出扩充类 940 不仅具有与核心类定义相关的原始类成员，而且现在也具有添加的类成员 926 和 931。由此，所示出的已扩充类是对用于以 IA-64 处理器为目标的 JIT 编译器的配置的类定义。以同一方式，可以继续条件地添加基于多个不同软件开发情形的多个不同扩展来静态地扩展核心类。当使用适当的源代码编译器编译时，扩充核心类 940 将帮助生成核心框架的定制版本。

#### 提供动态可扩充类定义的对象描述语言

静态扩展的一个缺点是它需要一起编译核心框架和扩展来生成可由计算机执行的一个单个文件。这意味着那些提供扩展的也必需具有对核心框架的源代码访问来与扩展一起对其重新编译。为避免由于许多不同原因而不希望的这一情况，可以在运行时刻生成扩充类定义。

图 10A 描述了使用对象描述语言 (ODL) 来基于核心框架的扩充版本的特定配置动态地扩充核心类定义的类声明的一个示例。类 INSTR 101 的核心类定义具有 1011 的带有属性 “extensible” 的头，用于指示这是一个动态可扩充类声明。类声明的体 1012 具有对框架公用的类成员。然而，当由 ODL 预处理器处理时，1011 的[extensible]属性生成并向源代码注入扩展对象，可用作用于进一步添加随后在运行时刻由适当的连接扩展提供的类成员的位置标志符。图 10B 说明了与特定目标情形关联的一个这样的扩展 1020。在该示例中，扩展 1020 添加对实现用于 IA-64 目标处理器的软件开发工具特定的 HINTBITS 1021 和 PREDICATES 1022 类扩展成员。属性[IA-64] 1023 用来指示这一特定的扩展仅适用于以 IA-64 处理器为目标的核心编译器和工具框架的定制配置。添加关键字 1024“extends”来指示扩展 1020 是 1025 的类 INSTR 的动态扩展。

如上所述，在动态生成扩充类的情况下，处理核心类定义 1010 和类扩展定义 1020 来创建核心类定义及其扩展的单独的源代码表示。随后也单独地编译这些单独的源代码表示来生成可由计算机执行的单独的文件。在这一情况下，与上述静态扩展不同，扩充核心类定义所需要的类扩展成员不是简单地使用扩充类定义添加到源代码的头文件。作为替代，如图 10C 所示，也可以向动态可扩充类定义希望在运行时刻向核心类定义 1010 添加的 GET 和 SET 类扩展添加其它代码 1031。由此，与静态扩展相比，动态扩展可具有必需执行一些另外的过程 1031 以在运行时刻适当地扩充类定义的附加的额外开销。由此，通常动态可扩充性降低了处理速度，而另一方面提供了用于提供扩展的额外的灵活性，因为在这一方法中，可以单独地编译核心和扩展。这允许第三方向核心类定义提供扩展而不需要对核心框架的源代码的访问。

图 11A 说明了动态可扩充核心类定义 1110 的另一示例。关键字 “extensible” 1111 指示该类 INSTR 是可扩充的，类的体也在添加任何扩展之前为类提供了类成员。类成员方法 1112 之一具有关键字 “extension point”，指示类扩展之一通过特别地定义方法或接口 FOO() 在指示的扩展点 1112 结合。

图 11B 描述了适合扩充图 11A 所示的可扩充类的动态扩展 1120，通过不仅提供以 IA-64 处理器为目标的配置所需要的 HINTBITS 1121 和 PREDICATES 1122 类扩展成员，还提供由扩展点 1112 指示的 FOO() 方法 1123 的定义来扩充。

“extension point” 关键字提供了指示可注入扩展的核心类定义的特定点的控制的

细粒度。此外，这可以通过使用对适当的扩展自动生成适当的指针的对象定义语言以简单的方式完成。类似地，也可以以静态可扩充核心类定义表示扩展点。

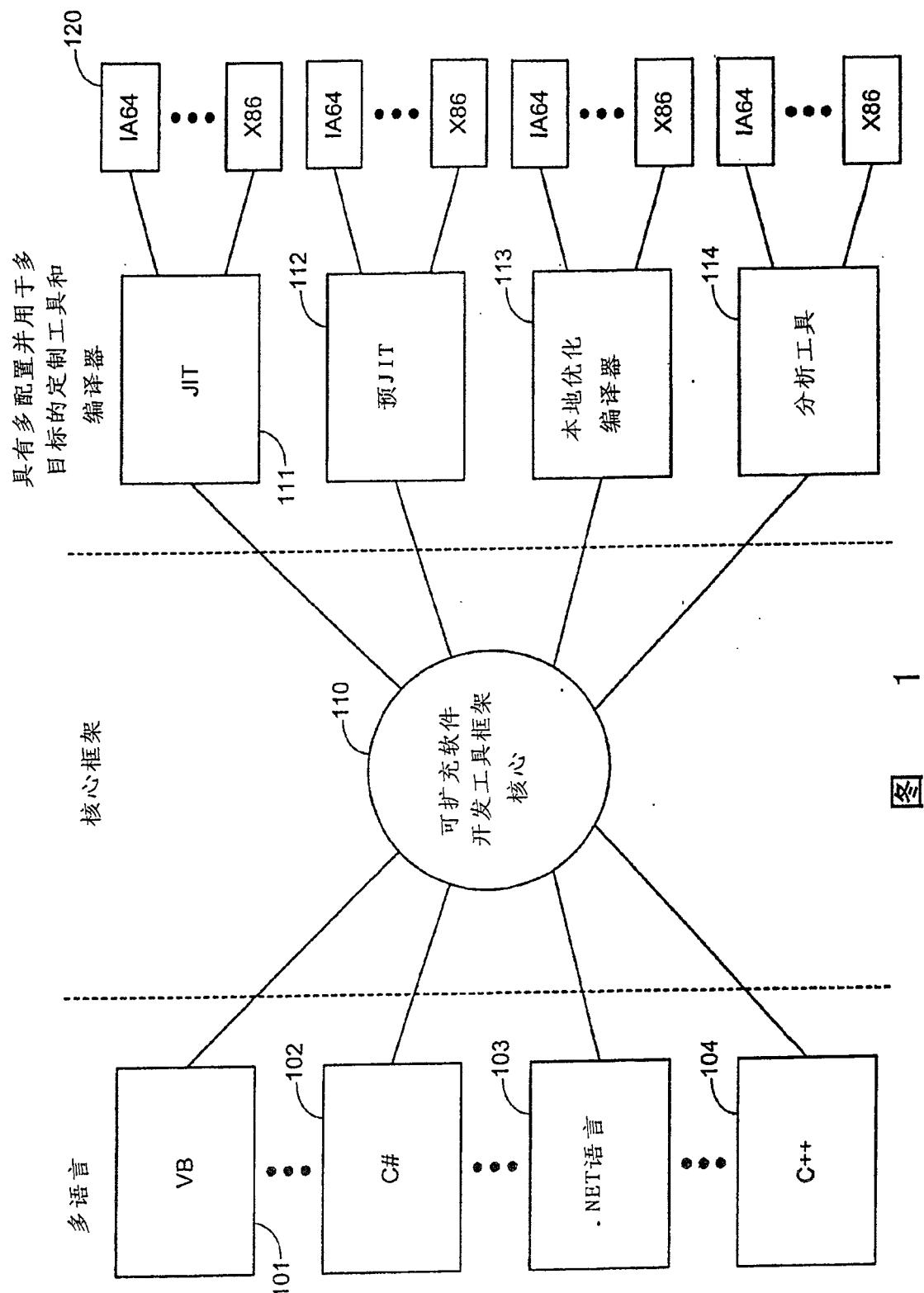
### 定制软件开发工具的示例性实现

图 12 说明了依照多个软件开发情形构造软件开发工具的示例性实现。可以通过添加与各种软件开发情形相关的类扩展来扩充核心类 1210。例如，类扩展可以与特定目标体系结构 1220（如，X86、IA-64 等等）的软件开发情形相关，其它扩展可与编译情形 1230（如，JIT、本地等等）相关。类似地，其它软件开发环境可影响扩展的选择。例如，可以有对软件开发工具的管理代码实现 1240 特定的扩展，或扩展可指示处于一组编程语言之外的编程语言。由此，可以添加这些扩展的各种配置来扩充核心类 1210，以使用可扩充核心框架来构建软件开发工具 1250。

### 替换方案

参考描述的实施例描述并说明了本发明的原理之后，可以认可，可以在不脱离这些原理的情况下在布置和细节上对所描述的实施例进行修改。尽管通过使用编译器的示例来说明这里所描述的技术，任一这些技术可以使用其它软件开发工具（如，调试器、优化器、仿真器和软件分析工具）。此外，这里参考扩充核心类初步地描述了生成扩展的原理，但是可以同等地应用同一原理来扩充核心类的任一扩展或子类。同样，上文所称 ODL 处理器能够接收对象描述语言并生成源代码语言。然而，ODL 预处理器的输出不必要仅限制在源代码语言上。它也可以提供为输出、诸如微软.NET 的 CIL 的中间语言表示或中间语言或可由处理器执行的任一形式。

同样，应当理解，这里描述的程序、过程或方法不相关或限于任一特定类型的计算机装置。依照这里描述的教导可以使用各种类型的通用或专用计算机装置或执行操作。这里描述的行动可以由包括用于执行这些行动的计算机可执行指令的计算机可读媒质实现。以软件形式示出的说明的实施例的元件可以以硬件形式实现，反之亦然。考虑到可应用本发明的原理的许多可能的实施例，应当认可，详细的实施例仅为说明性的，不应当认为限制本发明的范围。相反，将本发明要求权利为处于权利要求书及其等效技术方案的范围和精神之内的所有这类实施例。



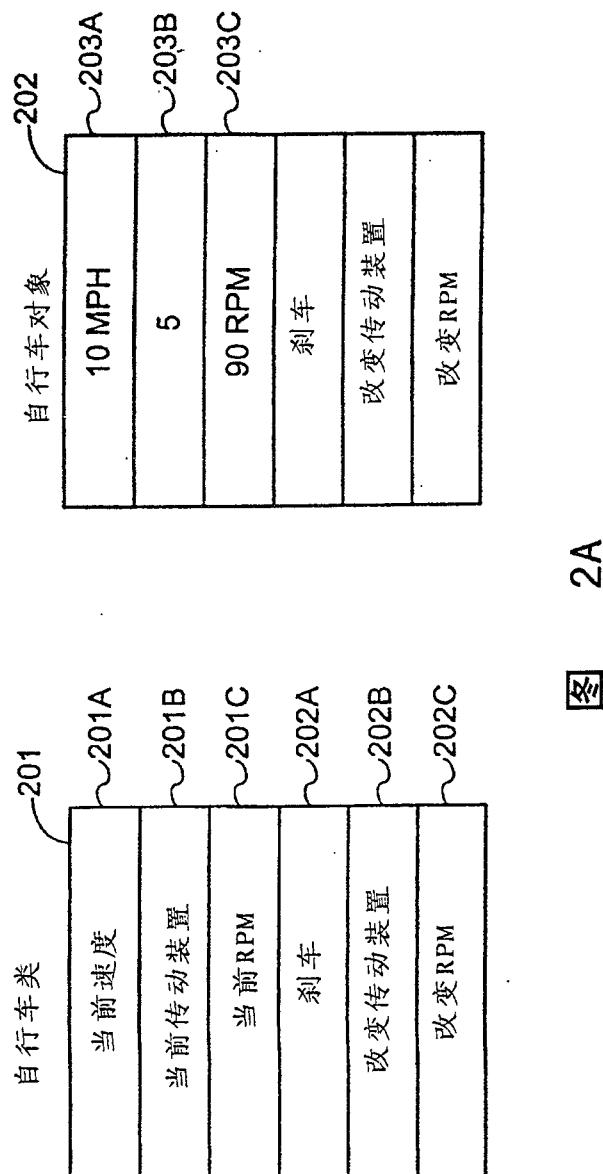


图 2A

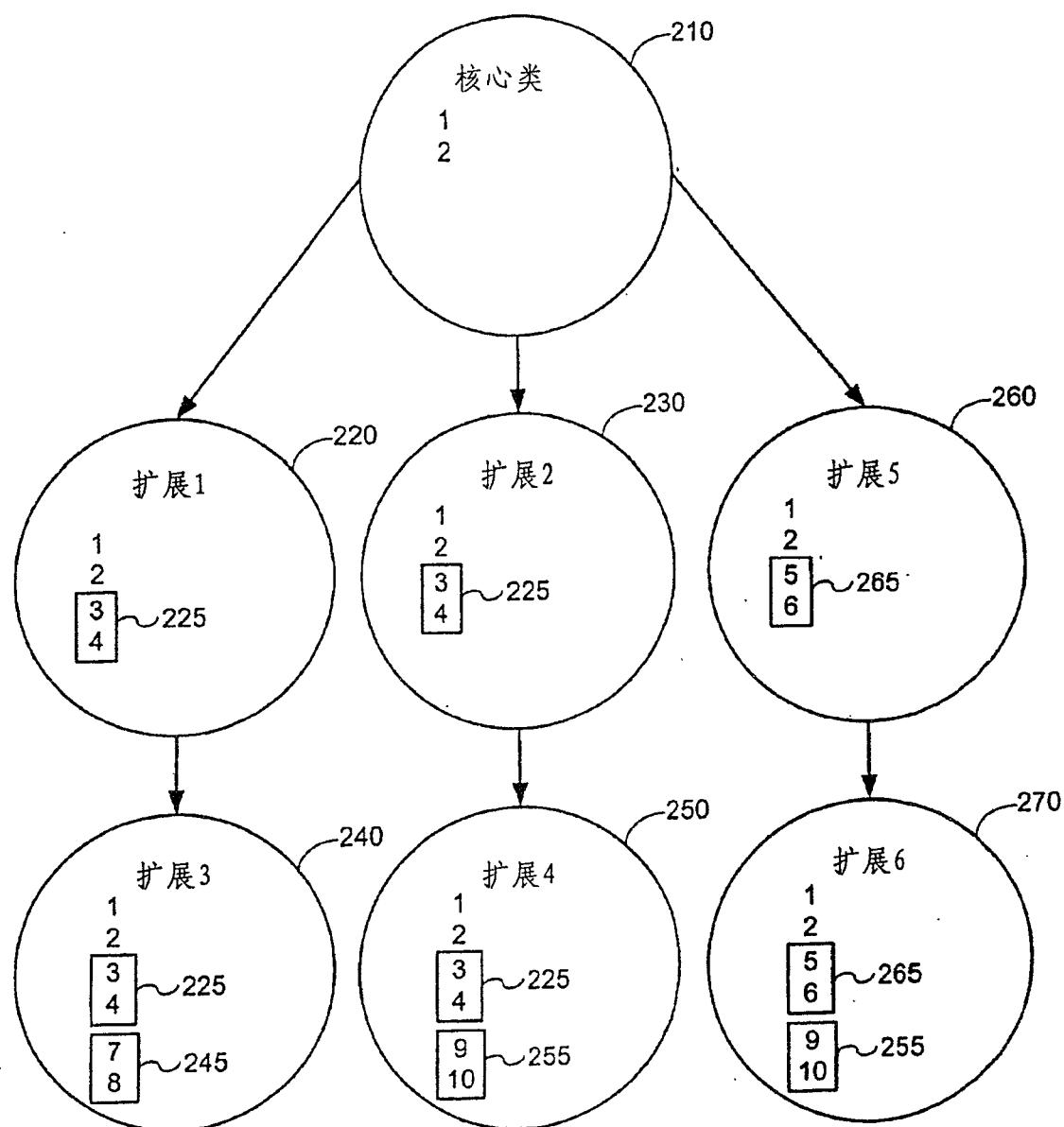


图 2B

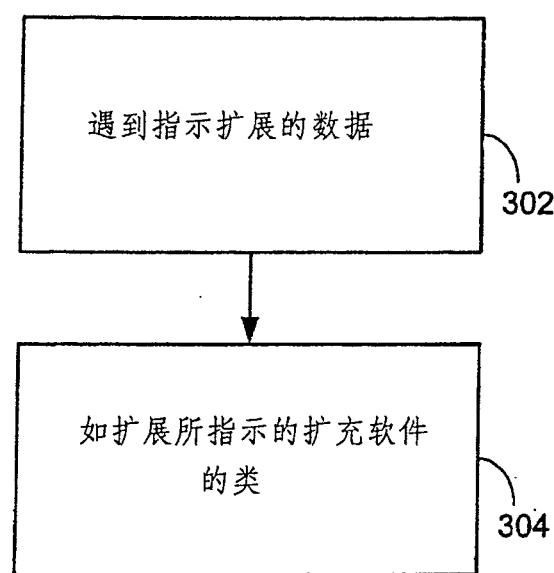


图 3A

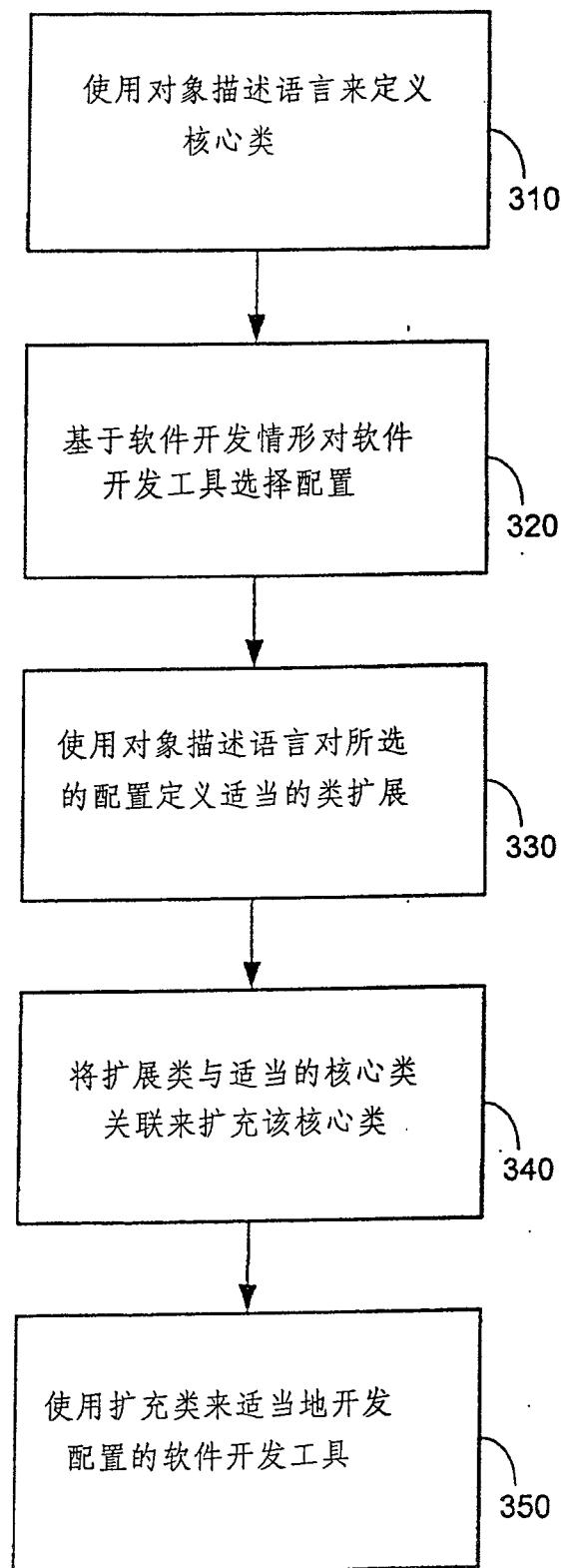


图 3B

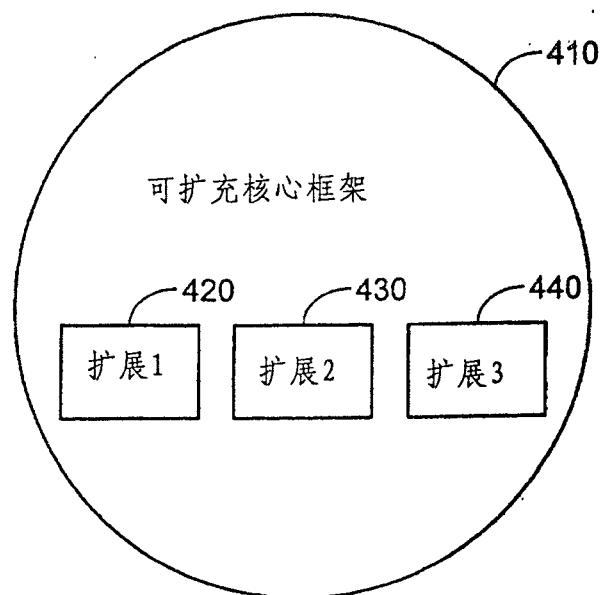


图 4A

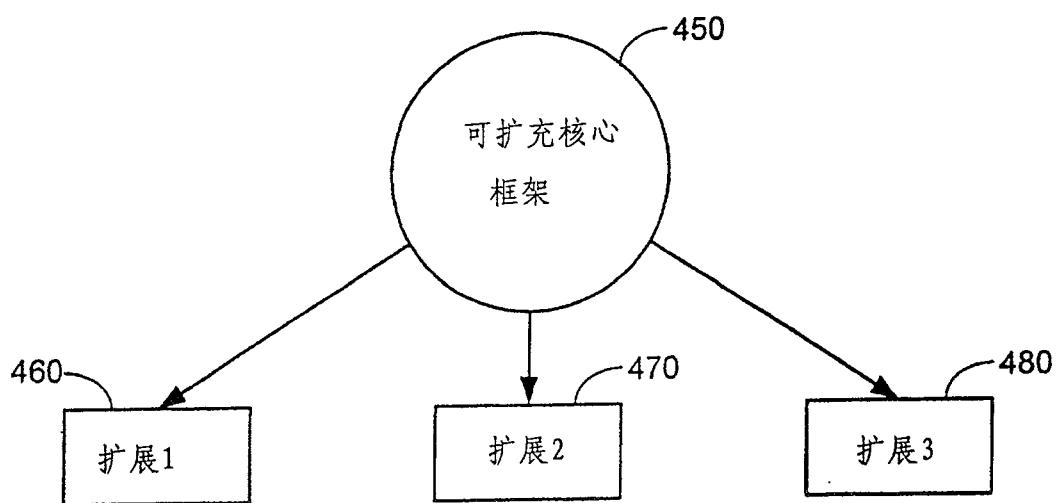


图 4B

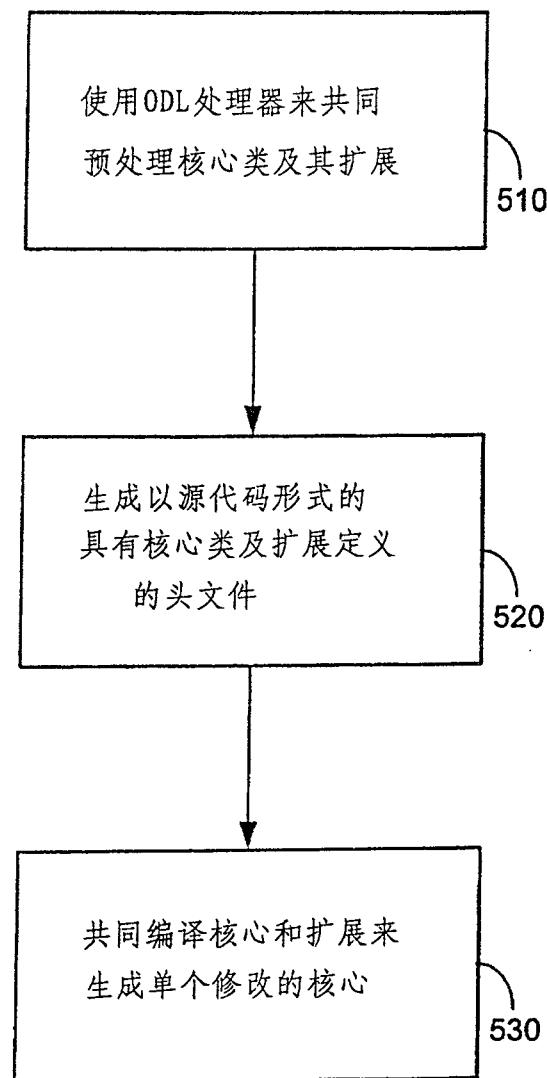


图 5

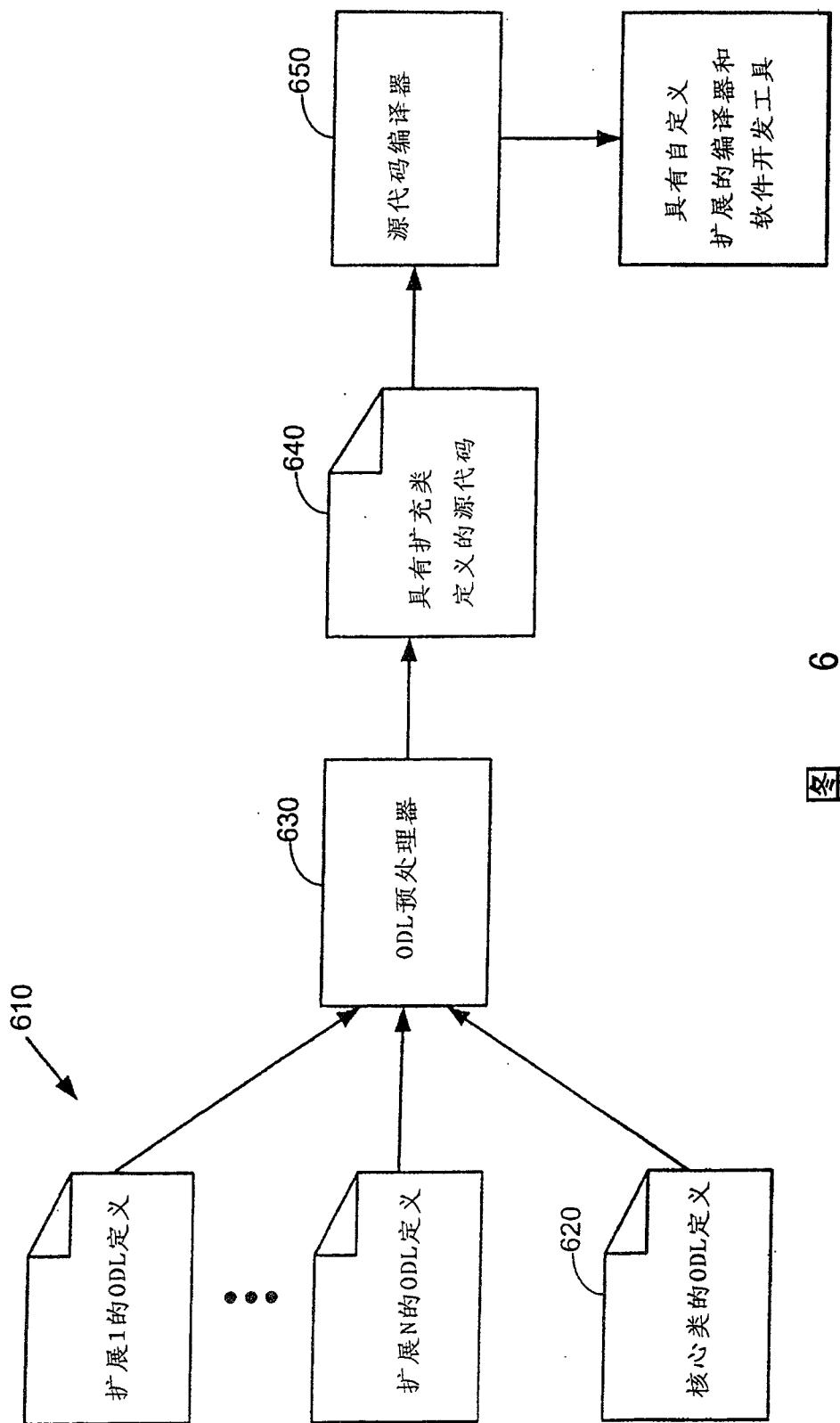
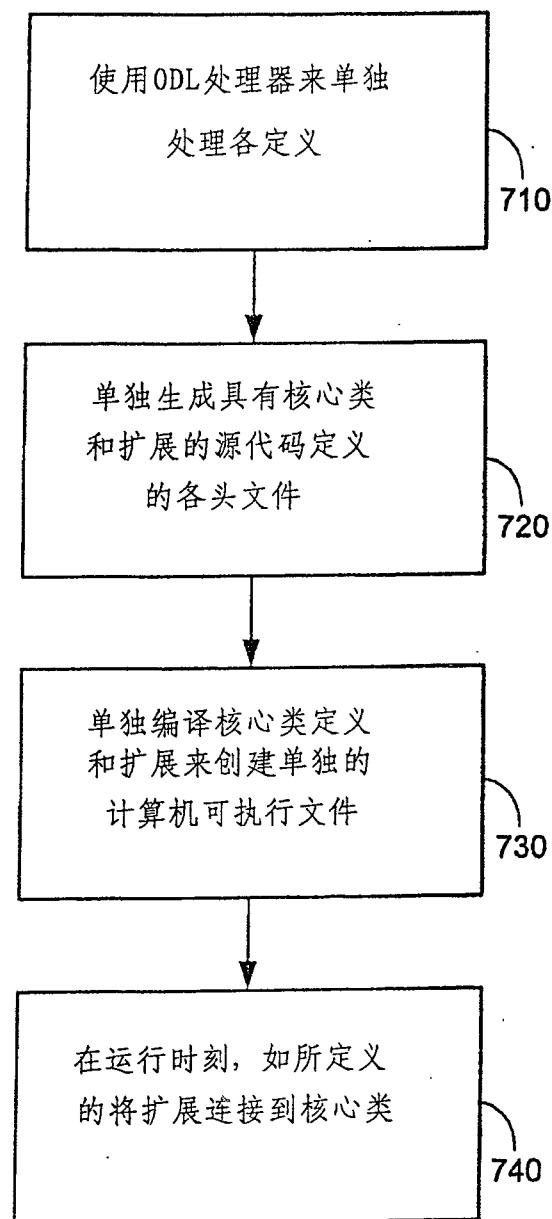
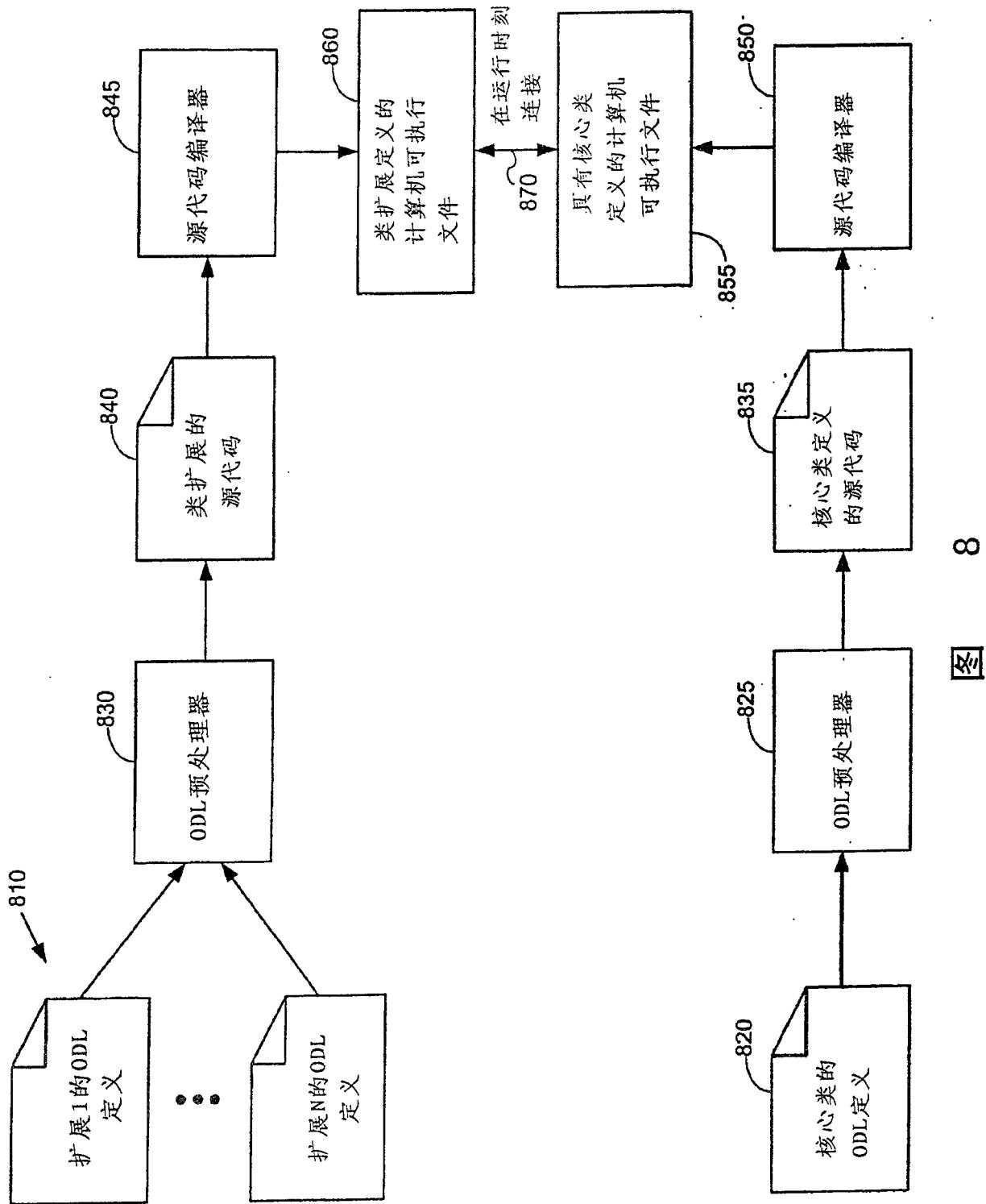


图 6





```

911 Public [...] ~913
TYPE SYM {
912 PHX::TYPE TYPE; ~914
NAME ::NAME ~915
};

```

900

图 9A

```

Extend class SYM [JIT]
Class JITSYM {
    JIT相关扩展
}
Extend class SYM [IA-64]
Class IA-64SYM {
    IA-64相关扩展
};

```

920

925  
~926

930  
~931

927  
928

图 9B

```

Public
TYPE SYM {
    PHX::TYPE TYPE; ~914
    NAME ::NAME; ~915
    JIT相关扩展 ~926
    IA-64目标相关扩展 ~931
}

```

940

图 9C

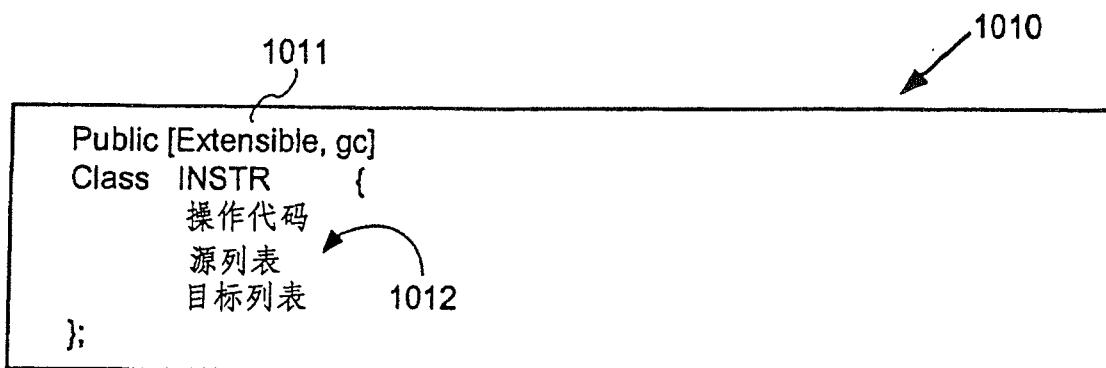


图 10A

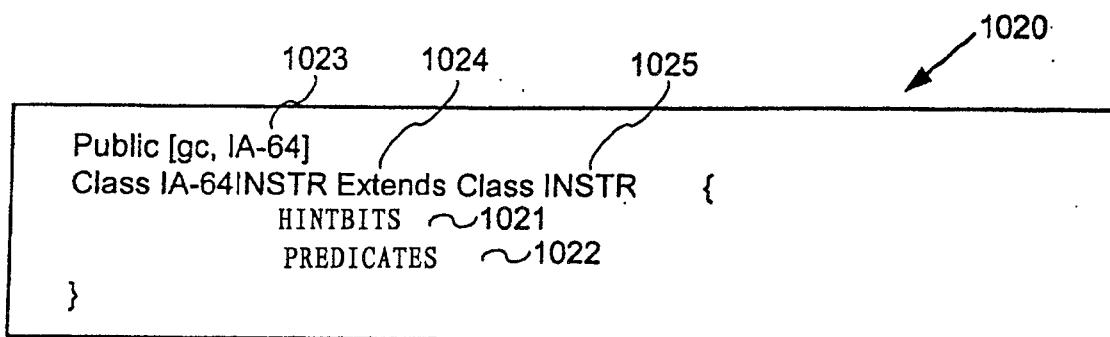


图 10B

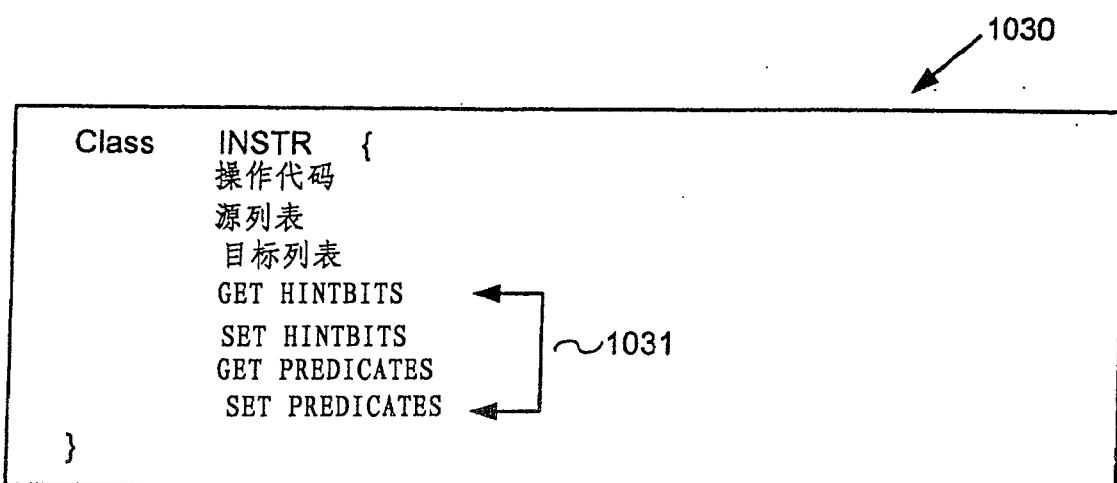


图 10C

1110  
1111  
Public [Extensible, gc]  
Class INSTR {  
 操作代码  
 源列表  
 目标列表  
 扩展点 FOO(); ~1112

图 11A

1120  
PUBLIC [gc, IA-64]  
CLASS IA-64INSTR Extends Class INSTR {  
 HINTBITS ~1121  
 PREDICATES ~1122  
 FOO 的定义 ~1123  
};

图 11B

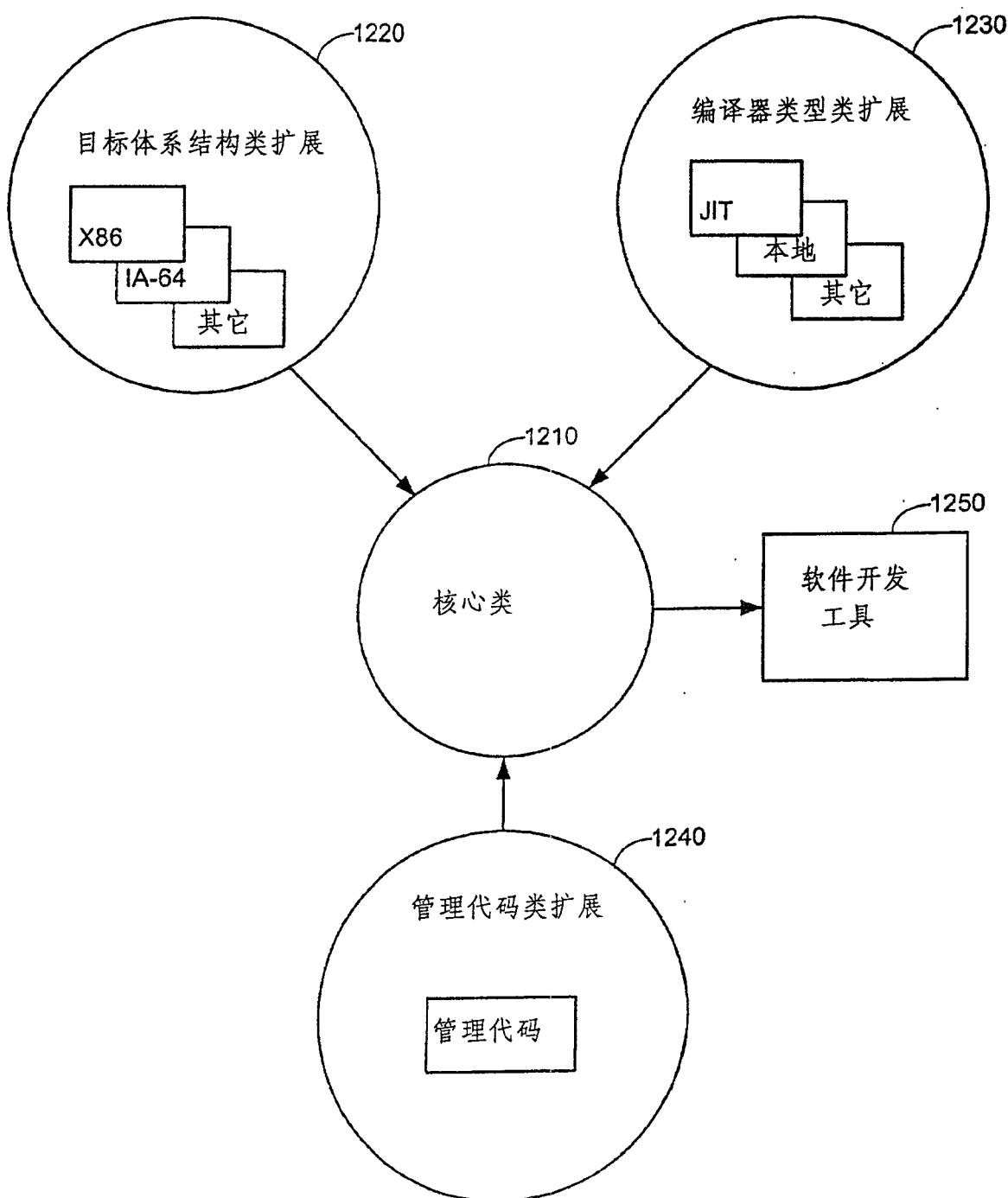


图 12