(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 102511033 B
(45)授权公告日 2014.09.03

(21)申请号 201080042419.3
(22)申请日 2010.09.22
(30)优先权数据
2,678,095 2009.09.25 CA
(85)PCT国际申请进入国家阶段日
2012.03.23
(86)PCT国际申请的申请数据
PCT/CA2010/001504 2010.09.22
(87)PCT国际申请的公布数据
WO2011/035431 EN 2011.03.31
(73)专利权人 国际商业机器公司
地址 美国纽约
(72)发明人 F·A·冈多尔菲
R·M·N·克拉雷尔
(74)专利代理机构 中国国际贸易促进委员会专利商标事务所 11038
代理人 杜娟
(51)Int.Cl.
G06F 9/45(2006.01)

(56)对比文件
US 5613120 A,1997.03.18,
US 5339438 A,1994.08.16,
CN 1949175 A,2007.04.18,

审查员 丁君军

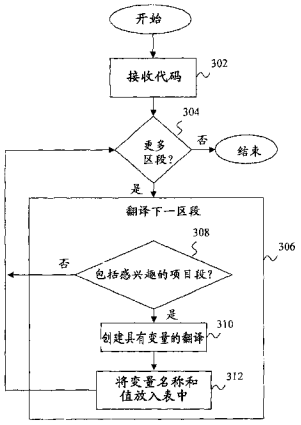
权利要求书2页 说明书10页 附图4页

(54)发明名称

对象级别兼容性和使用语义值的类大小调整

(57)摘要

一种在计算设备上将以高级语言编写的软件代码转换成二进制对象的方法，包括在计算设备上接收以高级语言编写的软件代码，以及在计算设备上将软件代码翻译成二进制对象文件。翻译包括确定软件代码包括需要被硬编码到二进制对象中的值的感兴趣的项，并且类是可调整大小的，以及创建语义变量表示硬编码值，并将语义变量和硬编码值存储在二进制对象中的表中。



CN 102511033 B

1. 一种在计算设备上将以高级语言编写的软件代码转换成二进制对象的方法，该方法包括：

在计算设备上接收以高级语言编写的软件代码；以及

在计算设备上将软件代码翻译成二进制对象文件，该翻译包括：

确定软件代码包括需要被硬编码到二进制对象中的值的感兴趣的项，并且类是可调整大小的；以及

创建语义变量来表示硬编码值，并将语义变量和硬编码值存储在二进制对象中的表中；

将所创建的语义变量替代所述值编码到二进制对象。

2. 根据权利要求 1 的方法，其中高级语言是 C++。

3. 根据权利要求 1 方法，其中感兴趣的项是下述之一：大小类类型，到非静态数据成员的偏移和虚拟函数表偏移。

4. 根据权利要求 1 的方法，其中该表是虚函数表偏移的表。

5. 根据权利要求 1 的方法，其中该表是类类型大小的表。

6. 根据权利要求 1 的方法，其中该表是在包含类中的子对象偏移的表。

7. 根据权利要求 1 的方法，其中该表是非静态数据成员偏移的表。

8. 根据权利要求 1 的方法，其中该表存储在二进制对象的头部。

9. 根据权利要求 8 的方法，还包括：

在计算设备上在链接器处接收多个二进制对象；

将多个二进制对象互相链接；以及

解决在具有基于预定义偏好的表的每个对象中的表之间的差异。

10. 根据权利要求 9 的方法，其中每个二进制对象包括至少一个表。

11. 根据权利要求 10 的方法，其中至少一个二进制对象不包括表。

12. 根据权利要求 9 的方法，其中预定义偏好使得较新的表优于较老的表而被选择。

13. 根据权利要求 9 的方法，其中预定义偏好使得具有较高版本号的表优于具有较低版本号的表而被选择。

14. 根据权利要求 9 的方法，其中预定义偏好使得包括作为二进制对象之间差异来源的虚拟表的对象优于不具有该虚拟表的对象而被选择。

15. 一种从多个二进制对象创建可执行程序的方法，该方法包括：

在计算设备处接收第一组多个二进制对象，至少一个二进制对象包括语义变量表，该语义变量代表需要硬编码值的至少一个值的硬编码值；

将第一组多个二进制对象链接在一起，并且用至少一个表中的硬编码值替换至少一个二进制对象中的语义变量；以及

将第一组多个二进制对象装载到计算设备中，以便与一个或多个另外的二进制对象相结合来操作。

16. 根据权利要求 15 的方法，其中需要硬编码值的值是下述之一：大小类类型、到非静态数据成员的偏移以及虚拟函数表偏移。

17. 根据权利要求 15 的方法，其中所述至少一个表是虚函数表偏移的表。

18. 根据权利要求 15 的方法，其中所述至少一个表是类类型大小的表。

19. 根据权利要求 15 的方法，其中所述至少一个表是包含类中子对象偏移的表。

20. 根据权利要求 15 的方法，其中所述至少一个表存储在二进制对象的头部中。

对象级别兼容性和使用语义值的类大小调整

技术领域

[0001] 本发明涉及为计算机系统创建可执行程序，并且更具体地，涉及通过使用语义值来翻译软件代码。

背景技术

[0002] 计算机程序通常由被称为二进制对象的一个或多个组件构成。每个二进制对象代表可执行代码或模块（其组成了它作为组件的程序）的一部分。二进制对象通常由将以更高级语言编写的软件代码转换成二进制对象的编译器（也被称为翻译器）产生。

[0003] 二进制对象然后链接在一起，组成完整的可执行程序。完成这项任务的软件工具被称为链接器。

[0004] 在程序能够运行之前，其必须被装载到计算机存储器内。完成这项任务的操作系统的组件被称为程序装载器。由于各种原因，一些程序组件可能不用链接器被链接到程序。相反这些组件被装载器添加到可执行文件。这些组件通常被称为共享对象、共享库或动态装载的库。

[0005] 将程序分割为多个二进制对象的好处之一是模块化。能够改变对应于一个二进制对象的源代码而不需要改变每个其它的二进制对象。具体地，仅仅是其源代码发生了改变的二进制对象需要使用编译器被再次翻译。未被影响的模块，包括共享库，不需要被重新翻译。

[0006] 在当代操作系统（包括但不限于 AIX、Linux、Windows、和 Z/OS）中普遍使用的链接器 / 装载器技术是在常用的面向对象编程语言出现之前设计的。这样，这些语言引入了当今可用的模块化类型不能很好地解决的新的复杂性。具体地，在面向对象的编程员的日常工作中通常产生的一些活动要求整个程序被重翻译，而不仅仅是直接受影响的二进制对象。这些活动可包括但不限于：添加虚拟函数到类接口；从类接口移除虚拟函数；把虚拟函数作为基本类接口的因子；添加数据成员到类接口；以及添加新的基本类到现有的类。

[0007] 不重新编译程序的所有构成二进制对象，包括其共享库，就无法进行这些活动，这是一个巨大的限制。单个共享库可以被很多独立的应用程序使用。因此，如果任何这些改变在共享库中做出，则共享库作为其组件的每个应用程序必须被重翻译（重编译）。这往往不可能，因为应用程序的用户不是熟练的编程人员，不拥有合适的软件工具或不能访问程序源代码。另外，用户无法控制模块向将来或从过去的开发中链接。因此，共享库的作者在他们能够对于其软件进行的各种改变方面严格受约束。在一些情况下，这些约束使得不能修补甚至很简单的缺陷。这在文献中被称为版本间二进制兼容性 (RRBC) 问题。

发明内容

[0008] 根据本发明的一个实施例，公开了一种在计算设备上将以高级语言编写的软件代码转换成二进制对象的方法。该实施例包括在计算设备上接收以高级语言编写的软件代码；以及在计算设备上将软件代码翻译成二进制对象文件。翻译包括确定软件代码包括需

要被硬编码到二进制对象中的值的感兴趣的项，并且类是可调整大小 (resizable) 的。本实施例中的翻译还包括创建语义变量来表示硬编码值，并将语义变量和硬编码值存储在二进制对象中的表中。

[0009] 本发明的另一实施例涉及一种从多个二进制对象创建可执行程序的方法。该实施例的方法包括在计算设备处接收第一组多个二进制对象，至少一个二进制对象包括语义变量表，语义变量代表需要硬编码值的至少一个值的硬编码值，并且将第一组多个二进制对象链接在一起，以及用至少一个表中的硬编码值替换至少一个二进制对象中的语义变量。该方法还包括将第一组多个二进制对象装载到计算设备中，与一个或多个另外的二进制对象相结合来操作。

[0010] 另外的特征和优点通过本发明的技术来实现。本发明的其他实施例和方面在此处详细描述，并且认为是所要求保护的发明的一部分。为了更好地理解本发明的优点和特征，参考说明书和附图。

附图说明

[0011] 被视为本发明的主题在说明书结束部分的权利要求中被特别地指出并被清楚地要求保护。本发明的前述的和其他特征以及优点能结合附图从以下详细描述中清楚地看到，其中：

[0012] 图 1 是根据本发明实施例的计算设备的系统图表；

[0013] 图 2 是显示本发明实施例执行的处理的示例的数据流图；

[0014] 图 3 示出了根据本发明实施例的翻译方法；

[0015] 图 4 示出了根据本发明实施例的链接二进制对象的方法。

具体实施方式

[0016] 本发明实施例涉及编译器 - 链接器 - 装载器系统，其中通常被编译器写入二进制对象的特定值被此后由链接器或装载器写入的变量和值所替换。这样的一种系统对于以例如 C 或 C++ 的高级编程语言编写的程序尤其有用。当然本发明不仅仅限于以 C 或 C++ 编写的程序，并且可以用于以其他语言编写的程序。

[0017] 传统 C++ 实现方案在翻译器执行期间（也就是在编译时）计算下述项：任何类类型的大小；任何数据成员在类中的偏移；以及任何虚拟函数指针在类的虚拟表（虚函数表或等同物）中的偏移。这些值由翻译器写入对象文件。在本发明中，翻译器替代地创建包含链接到这些值的变量的表。所述变量，而非所述值，被写入对象中，并且由链接器或装载器使用所述变量来访问所述表以代替所述值。因此，仅仅那些改变了上述项的定义的对象文件需要被重新编译，而不是程序的所有部分。

[0018] 图 1 显示了计算系统的例子，在其上可以实现本发明的实施例。应当理解的是，系统可以是分布式的，并且以下描述的系统各部分可以存在于彼此不同的物理位置处。

[0019] 在该实施例中，系统 100 具有一个或多个中央处理单元（处理器）101a、101b、101c 等（共同地或一般性地称为处理器 101）。在一实施例中，每个处理器 101 可以包括精简指令集计算机 (RISC) 微处理器。处理器 101 通过系统总线 113 耦接到系统存储器 114 以及各种其他组件。只读存储器 (ROM) 102 耦接到系统总线 113，并且可包括基本输入 / 输出系

统 (BIOS)，其控制系统 100 的特定基本功能。

[0020] 图 1 还描述了输入 / 输出 (I/O) 适配器 107 和耦接到系统总线 113 的网络适配器 106。I/O 适配器 107 可以是小型计算机系统接口 (SCSI) 适配器，其与硬盘 103 和 / 或磁带存储驱动器 105 或任何其他类似组件通信。I/O 适配器 107、硬盘 103 和磁带存储驱动器 105 此处共同地称为大容量存储器 104。网络适配器 106 将总线 113 与外部网络 116 互连，使得数据处理系统 100 能够与其他这种系统通信。屏幕（例如显示监视器）115 通过显示适配器 112 连接到系统总线 113，显示适配器 112 包括用以提高图形密集应用的性能的图形适配器和视频控制器。在一实施例中，适配器 107、106 和 112 可以连接到一个或多个 I/O 总线，后者通过中间总线桥（未显示）连接到系统总线 113。用于连接例如像硬盘控制器、网络适配器和图形适配器的外围设备的合适的 I/O 总线通常包括常见协议，例如外围元件接口 (PCI)。附加的输入 / 输出设备被示出为通过用户接口适配器 108 和显示器适配器 112 连接到系统总线 113。键盘 109、鼠标 110 和扬声器 111 都通过用户接口适配器 108 互连到总线 113，用户接口适配器 108 包括例如将多个设备适配器集成在单个集成电路中的超级 I/O 芯片。

[0021] 这样，如图 1 所配置的，系统 100 包括处理器 101 形式的处理装置、包括系统存储器 114 和大容量存储器 104 的存储装置、例如键盘 109 和鼠标 110 的输入装置，以及包括扬声器 111 和显示器 115 的输出装置。在一实施例中，系统存储器 114 和大容量存储器 104 的一部分共同地存储一操作系统，例如来自**IBM®**公司的**AIX®**操作系统以协调图 1 中所示的各个组件的功能。

[0022] 应当理解，系统 100 可以是任何合适的计算机或计算平台，并且可以包括终端、无线设备、信息电器、设备、工作站、微型计算机、大型机、个人数字助理 (PDA) 或其他计算设备。应当理解的是，系统 100 可以包括被通信网络连接在一起的多个计算设备。例如，在两个系统之间可存在客户机 - 服务器关系，并且处理可以分摊在两者上。

[0023] 系统 100 能够支持的操作系统的例子包括**Windows®** 95、**Windows®** 98、Windows **NT®** 4.0、Windows **XP®**、**Windows®** 2000、 **Windows®** CE、Windows **Vista®**、Mac OS、**Java®**、 **AIX®**、LINUX 和**UNIX®**或任何其他合适的操作系统。系统 100 还包括通过网络 116 通信的网络接口 106。网络 116 可以是局域网 (LAN)、城域网 (MAN) 或广域网 (WAN)，例如因特网或万维网。

[0024] 系统 100 的用户能通过任何合适的网络接口 116 连接，例如标准电话线、数字用户线、LAN 或 WAN 链接（例如 T1、T3）、宽带连接（帧中继、ATM）和无线连接（例如 802.11(a)、802.11(b)、802.11(g)），而连接到网络。

[0025] 如此处所公开的，系统 100 包括存储在机器可读介质（例如硬盘 104）上的机器可读指令，用于捕获和交互显示在用户屏幕 115 上示出的信息。如此处所讨论的，指令被称为“软件”120。如本领域所公知的，软件 120 可使用软件开发工具产生。如本领域所公知的，软件 120 可包括多种工具和特征以提供用户交互的能力。

[0026] 在一些实施例中，软件 120 被提供为覆盖另一程序。例如，软件 120 可以被提供为对应用（或操作系统）的内插式程序 (add-in)。注意术语“内插式程序”一般指本领域公知的补充程序代码。在这样的实施例中，软件 120 可替代与其协作的应用或操作系统的结构或对象。

[0027] 图 2 是示出根据一实施例的系统中数据移动的数据流程图。系统包括翻译器 202。翻译器 202 可基于任何已知的编译器。翻译器 202 接收一个或多个源代码文件 204，并将每个源代码文件转换成对象文件 206。当然，源代码文件的确切数量是可变的，并可以包括例如被很多应用所使用的共享库或其他源代码文件。

[0028] 但是翻译器 202 与典型的翻译器不同之处在于它如何从源代码创建一个对象文件。具体地，翻译器 202 可在每个对象文件中创建表 208，包括链接到通常被传统编译器硬编码到对象文件中的特定值的变量。变量而不是值被写入对象，并且链接器 210 或装载器 212 使用变量访问所述表，此时变量被值代替。

[0029] 系统还包括链接器 210。链接器 210 可与传统链接器相似。链接器或链接编辑器是获取由编译器 202 产生的一个或多个对象 206 并将它们组合在单个可执行程序 212 中的程序。但是，根据本发明一实施例的链接器 210 可被配置成使得它查阅每个对象文件 206 的表 208，以便用通常被现有技术编译器写入对象文件 206 的静态值填入变量。这些值被硬编码到可执行文件中。硬编码值是不会改变的值。硬编码值的一个例子是何时变量被指定为一特定整数。当然，不是所有包含在表中的值可转换成硬值，并且可执行文件可包括一可执行变量表 214。

[0030] 系统还可包括装载器 216。装载器是操作系统的一部分，其负责将程序从可执行体（例如可执行文件）装载到存储器中，让它们准备执行并且然后执行它们。在现有技术中，装载器一般是操作系统内核的一部分，并一般在系统启动时装载，在存储器中保持到系统重启、关闭或断电。一些具有可分页内核的操作系统在存储器的可分页部分有装载器，因此装载器有时会被交换出存储器。所有支持程序装载的操作系统都具有装载器。

[0031] 根据本发明实施例的装载器 216 的操作类似于现有技术的装载器。但是链接器 216 还查阅每个可执行文件 212 的可执行变量表 214 以便用通常被现有技术编译器写入对象文件 206 的硬值填入变量。

[0032] 图 3 显示了根据本发明一个实施例的翻译器 202（图 2）的操作方法。在块 302 中，要被翻译的所有部分或一个部分在翻译器处被接收。翻译器（或编译器）是计算机程序（或一组程序），其将以计算机语言（源语言）编写的源代码转换成另一计算机语言（目标语言，经常具有被认为是对象代码的二进制形式）。希望转换源代码的最常见原因是创建可执行程序。

[0033] 在块 304 中，确定是否有更多的区段（整个源代码文件或其一个区段）需要被翻译。如果没有，该方法结束并且翻译完成。

[0034] 如果还要进行更多的翻译，下一区段或文件在块 306 中被翻译。作为块 306（翻译）的一部分，在块 308 确定该段是否包含感兴趣的项。术语“感兴趣的项”在此应指代现有技术翻译器可能已在翻译期间为其创建静态值的代码段。感兴趣的项的示例可以包括但不限于：类类型的大小，用来访问类对象中非静态数据成员的偏移；以及虚拟函数表偏移。可理解的是，传统的 C++ 实现方案在翻译器执行期间（也就是在编译时）计算以下的项：任何类类型的大小，任何数据成员在类中的偏移；以及任何虚拟函数指针在类的虚函数表（虚拟表或等同物）中的偏移。这些计算得到的值是上述的预先被翻译器写入对象文件的“静态的”或硬编码值。

[0035] 通过解释，C++ 编程语言允许编程者通过使用类来定义程序特定数据类型。这些

数据类型的实例被称为对象，并且其可以包括程序员定义的成员变量、常量、成员函数及过载运算符。语法上，类是 C 结构体的延伸，不能包括函数或过载运算符。下面的描述将解释 C++ 语言的例子，但是应当理解的是，其他语言在预期中，并在本发明的范围内。

[0036] 在块 310，创建具有变量名称的翻译。这不同于现有技术，如通过非限制性的以下示例所示。

[0037] “sizeof”表达式的值在编译时被 C++ 语言翻译器计算。sizeof 操作符产生关于字符类型大小的其操作数的大小。传统 C++ 实现方案用等同于其操作数大小（以八位字节计数）的常值来替代 sizeof 表达式。例如，语句：

[0038]

```
struct S {
int datum;
};
const size_t char_size = sizeof(S);
```

[0039] 将被翻译成如下：

[0040] const size_t char_size = 4 ;

[0041] 其中值“4”将被硬编码到二进制对象中。

[0042] 本发明实施例不同于传统实现方案之处在于，当 sizeof 操作符的操作数是类类型或具有类类型，则转换的表达式用符号常量而不是顺序值来替代 sizeof 表达式。例如，本发明的翻译器可以将以上的语句翻译为：

[0043] const size_t char_size = _SYMBOL_sizeof_S.

[0044] 在块 312，_SYMBOL_sizeof_S 的硬值（本例中为 4）存储在表中。该表然后被链接器或装载器使用，以便此后将值 4 代入表达式 _SYMBOL_sizeof_S。这允许在一个区段中做出改变，而不需要重新编译整个程序，只有发生改变的区段（例如，改变的对象文件）要被重编译。这减轻了以上描述的一部分或全部问题。

[0045] 以下部分描述了可以对编译器做出的其他改变，以及如何理解这些改变。例如，在现有技术中当动态分配存储器时（例如“new”表达式），例如下述的语句：

[0046]

```
struct S {
int datum;
};
S*sptr = new S;
```

可以被转换成语句：

```
struct S {
int datum;
};
S*sptr = ::operator new(4);
```

[0047] 如上，值“4”将被硬编码到二进制对象中，并且代表存储器分配器请求的存储器的量。该值作为自变量传递给函数 ::operator new，其调用存储器分配器。本发明的实施例可不同于这种传统实现方案之处在于，当对于 new 表达式的操作数是类类型时，转换的 new 表达式将一符号常量而不是顺序值作为自变量传送给 ::operator new。例如，上述语句可根据本发明实施例转换为：

[0048]

```
struct S {
int datum;
};
S*sptr = ::operator new (_SYMBOL_sizeof_S）;
```

[0049] 作为另一个例子，在本发明中，自动作用域对象的分配有所不同。在现有技术中，自动作用域对象（包括值函数参数）被分配到程序堆栈上。在堆栈上为一给定类对象预留的存储器量被硬编码到二进制对象中。本发明实施例可以不同于传统实现方案之处在于：当要被分配的对象具有类类型时，按以上所述的方式，预留的存储器量由符号常量而不是顺序值确定。

[0050] 作为另一示例，考虑静态作用域对象分配的情况。通常，静态分配的类对象存储在当装载程序时为它们留出的存储器中。为此目的预留的存储器量在为此目的而存在的二进制对象文件的一个区段中被指定。例如，在通用对象文件格式 (COFF) 中，要为每个静态类对象预留的存储器量在对象文件的 .bss 区段中被指定。在本发明的实施例中，翻译器根据传统实践编码对象文件的 .bss 区段（或等同物），只是 .bss 区段（或等同物）中的每个条目将包含附加信息，该附加信息说明其静态分配正被表示的类对象的类型。该附加信息可以被程序装载器使用。

[0051] 作为另一例子，考虑在出现多重继承时子对象偏移的情况。在出现多重继承的情况下，当将类对象的类型从最衍生的类型转换成基本类型或在基本类型之间转换时，必须用偏移调整“本 (this)”指针。例如，考虑以下表达式：

[0052]

```
struct Left{
int datum1;
};
struct Right{
int datum2;
};
```

[0053]

```
struct Derived: Left,Right{
int datum3;
};
void fn(Right *arg);
int main(){
Derived dobj;
Fn(&dobj);
}
```

[0054] 在本例中的类对象 dobj 包括 Left 类型的子对象和 Right 类型的另一子对象。传统 C++ 编译器将 Left 类型的子对象定位到 dobj 中偏移 0 的位置处。Right 类型的子对象将被定位到 dobj 中的如下计算的偏移处 :使得 _SYMBOL_sizeof_Left 成为代表类类型 Left 的大小的符号常量。使得 DerivedMembers 成为代表类类型 Derived 的所有非静态数据成员的总体大小的符号常量,不包括从其基本类继承的那些成员。DerivedMembers 还包括任何所需要的对齐填充的大小。dobj 中的类型 Right 的子对象的偏移是 _SYMBOL_sizeof_Left 与 DerivedMembers 的和。在传统 C++ 翻译器中,该偏移在需要时计算并被直接编码到对象文件中。例如,在调用名为 fn() 的函数时,表达式 &dobj 的类型必须从类型 Derived *转换成类型 Right *。为了完成此,由表达式 &dobj 代表的指针必须被调整,使得其代表的不是对象 dobj 的地址,而是 dobj 所包含的类型 Right 的子对象的地址。该调整通过将 Right 子对象在 dobj 的偏移添加到 dobj 的地址来实现。假设一个 int 的 sizeof 是 4,并且一个 int 的 alignmnt(对齐) 是 4, Right 在 dobj 中的偏移是 8。这样,传统 C++ 编译器将把以上函数 main() 转换为以下的形式 :

[0055]

```
int main(){
Derived dobj;
fn(static_cast<Right*>(&dobj+8));
}
```

[0056] 偏移在所产生的代码中直接以数字值表示。在本发明中,这样的偏移不被表示为

顺序值，而是表示为符号常量：

[0057]

```
int main(){
Derived dobj;
fn(static_cast<Right*>(&dobj+_SYMBOL__offsetof_Right_within_Derived));
}
```

[0058] 当调用虚拟函数时，使用这些相同的偏移。但是，由于偏移的值取决于用于分派虚拟函数的对象的动态类型，因此偏移不能被编码到函数调用地址中，因为对象的动态类型直到程序执行时才可知。因此，偏移在虚函数表中被编码为“本调整 (this adjustment)”。有两种完成此的常用技术：(1) 虚函数表中的每个条目是由到虚拟函数的指针和适当的偏移构成的对，或 (2) 虚函数表仅包括指向可执行代码的指针；并且由本指针引用的代码可以是虚拟函数或在偏移非零时的“adjustor thunk”，adjustor thunk 将正确的偏移添加到的本指针，并且然后分支到虚拟函数。在任一情况下，传统 C++ 翻译器使用数字值来表示偏移。在本发明中，偏移使用符号常量表示，如上所述。

[0059] 另一例子可能与动态转换表 (dynamic cast table) 的例子。C++ 实现方案产生动态转换表以允许在程序执行期间计算类类型的互相转换。动态转换表包括子对象偏移，与已描述的那些类似。在本发明中，这些偏移如上所示使用符号常量表示。

[0060] 除了类大小之外，传统 C++ 实现方案还硬编码用来访问类对象中的非静态数据成员的偏移。在本发明中，偏移使用符号常量来表示。该技术应用于直接成员访问，以及到非静态数据成员的指针。

[0061] 另外，本发明的实施例可以将以上技术应用到虚拟函数表偏移的情况中。类的虚函数表中的各个条目通过虚函数表内的偏移来访问。传统 C++ 实现方案将这些偏移硬编码到函数调用地址中。例如，

[0062]

```
struct Base{
virtual void foo() const{}
```

[0063]

```
virtual void bar() const{}
};
struct Derived : Base{
virtual void bar() const{}
};
int main(){
const Base&ref = Derived();
ref.bar();
}
```

[0064] 如果每个虚函数表条目占据 4 字节，并且指向虚函数表的指针定位在从类类型 Base 或 Derived 的对象起始偏移为 0 的位置处，则在 main() 中函数 bar() 的调用将如下被转换：

[0065] (**((void(**)()const)(ref->0+4)))(ref)

[0066] 子表达式 ref->0 表示虚函数表起始的地址，并且数字 4 是代表虚拟成员函数 bar() 的条目在虚函数表中的偏移。在本发明中，该数字被符号常量替代。

[0067] 类的虚函数表还可以包括虚拟基本指针 (VBP)。这些等同于包含类对象内基本类子对象的偏移，并且如上描述地进行处理。

[0068] 以上的描述详述了在块 310 处所进行的替换的示例。应当理解的是，这些值和用于表示它们的符号常量在块 312 处存储在对象文件中的表中。

[0069] 以上所确定的每个符号常量可以被定义在四个表之一中，这些表位于在块 312 处创建的对象文件格式的可选头部或尾部区段。程序链接器和装载器可根据本发明扩展以允许它们读取这个区段。在一实施例中，四个表可包括：1) 类类型大小的表，其对于每个类类型给予单独一行。在该表中，给定的行包括三项信息：代表类类型大小的符号常量的名称，使用常用的方法被语言翻译器确定的类类型的大小，以及指示类类型是否能够被链接器 / 装载器调整大小的标志；2) 在包含类中子对象偏移的表，如上所述。一给定行包括两项信息：符号常量的名称以及偏移的值；3) 非静态数据成员偏移的表，其中一给定行包括两项信息：符号常量的名称和值以及偏移的值；以及 4) 虚函数表偏移的表。

[0070] 在一实施例中，如果目标可执行文件没链接到动态或共享库，则所有的上述符号常量都可以被链接器解析。出现在可执行代码中的每个符号常量被相应的值所替代，如合适表所确定的。由于包含这些表的头部或尾部区域是可选的，具体实现本发明的链接器可以消耗包含以及不包含这些文件的二进制对象。因此，现有 C++ 编译器能够被扩展来实现本发明，并且由先前版本的编译器所创建的二进制对象能够链接到由产生表的新版本编译器所创建的二进制对象。

[0071] 图 4 显示了链接器或装载器如何根据本发明实施例操作的流程图。在块 402，由翻译器创建的二进制对象在链接器处被接收。可以从与链接器位于相同或者不同位置的编译器接收。根据本发明，在块 404，链接器检查对象文件以确定它们是否包含任何表。在表

存在的情况下，在块 406，由链接器用静态值替代符号常量。该过程在以下详细描述。在块 408，链接器继续如现有技术中那样链接对象。

[0072] 如上所讨论地，在块 406，符号常量被替换。一般地，类对象在使用它的每个程序模块中被定义。如果类对象以某种方式被修改，则对应于定义类类型的程序模块的每个二进制对象必须被重新编译。在传统 C++ 翻译器 / 链接器环境中，以某种方式对类进行修改之后如果不能重新翻译包含类定义的每个模块，则将导致在执行期间链接时间错误或程序故障。根据本发明实施例的链接器可以解决该问题。考虑类定义以下述方式被修改，即改变其大小，或任何静态数据成员的位置或其内部的基础类子对象发生了改变。上述表包括足够的信息以允许链接器检测二进制对象之间在给定类类型的定义上的差异。这些差异的调和可以下述任何方式来实现 :1) 时间和日期标记 :使用来自最近创建的二进制对象的信息 ;2) 类版本控制 :允许程序员通过编译指示来将版本号与其类相关联以及使用与类的最近版本相关联的信息 ;3) 程序员交互 :允许程序员在调用链接器时直接规定如何解决差异 ;4) 产生导致创建最大类的类定义的版本 ;以及 5) 从二进制对象产生信息，该二进制对象包含对应于作为差异来源的类的虚函数表。

[0073] 应当注意，在使用一个或多个共享库的情况下，装载器可以以如上对于链接器所述的相同方式操作。

[0074] 当然，本领域技术人员应当理解，本发明的实施例在一些分立的情况下会遇到困难。这些可能包括，例如，向多态类增加项或从其移除，函数协调以及用模版处理。这样，这些类的类型被标记为不可调整大小的，并且编译器如现有技术一样对它们进行操作。

[0075] 此处所用术语仅是为了描述特定实施例的目的，而不为了限制本发明。如此处所用，单数形式“一个”、“一”及“该”也包括复数形式，除非上下文另外清楚指示。还应当理解，术语“包括”用在本说明书中，表示所声明特征、整数、步骤、操作、元件和 / 或组件的存在，但并不排除存在或增加一个或多个其它特征、整数、步骤、操作、元件组件和 / 或它们的组。

[0076] 在以下权利要求中出现的所有装置或步骤加功能要素的相应结构、材料、动作和等同物，意指包括与特别要求保护的其他要求保护的要素相结合来执行所述功能的任何结构、材料或动作。本发明的描述是为了说明和描述的目的，但不为了穷举或以所公开的形式限制本发明。很多改进和变形对于本领域技术人员来说是显然的，不背离本发明的范围和精神。实施例的选择和描述是为了最好地解释本发明的原理和实际的应用，并且为了使得本领域其他技术人员能够理解本发明具有不同改进的不同实施例也适于预期的特殊应用。

[0077] 此处所描述的流程图仅是一个示例。此处所描述的该图或步骤（或操作）可以有多种变化，这不偏离发明的精神。例如，步骤可以不同的顺序执行，或者可以添加、删除、修改步骤。所有这些变化被认为是所要求保护的发明的一部分。

[0078] 虽然已经描述了本发明的优选实施例，但是应当理解的是，本领域技术人员，不管是现在还是将来可以做出各种改进和增强，这都落入后述权利要求的范围之内。这些权利要求被解释为对上述发明维持适当保护。

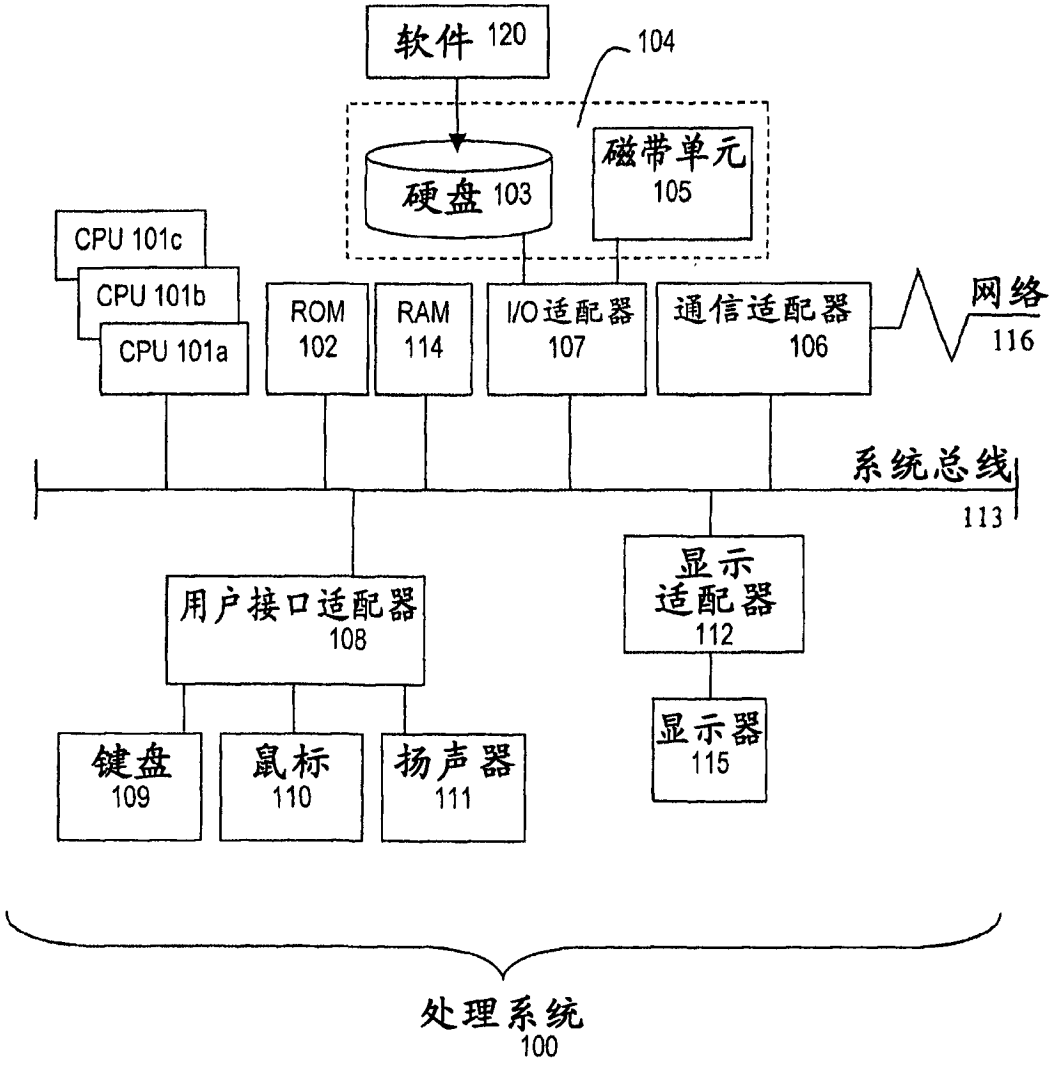

软件 120
104
硬盘 103
磁带单元
105
CPU 101c
CPU 101b
CPU 101a
ROM
102
RAM
114
I/O适配器
107
通信适配器
106
网络
116
系统总线
113
用户接口适配器
108
显示
适配器
112
显示器
115
键盘
109
鼠标
110
扬声器
111
处理系统
100

图 1

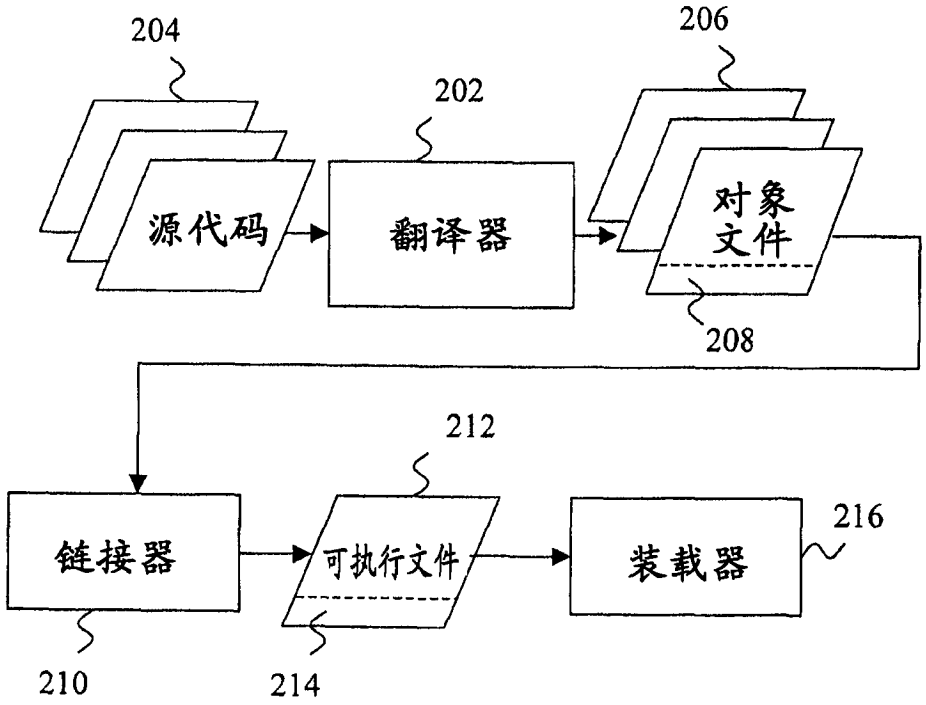


图 2

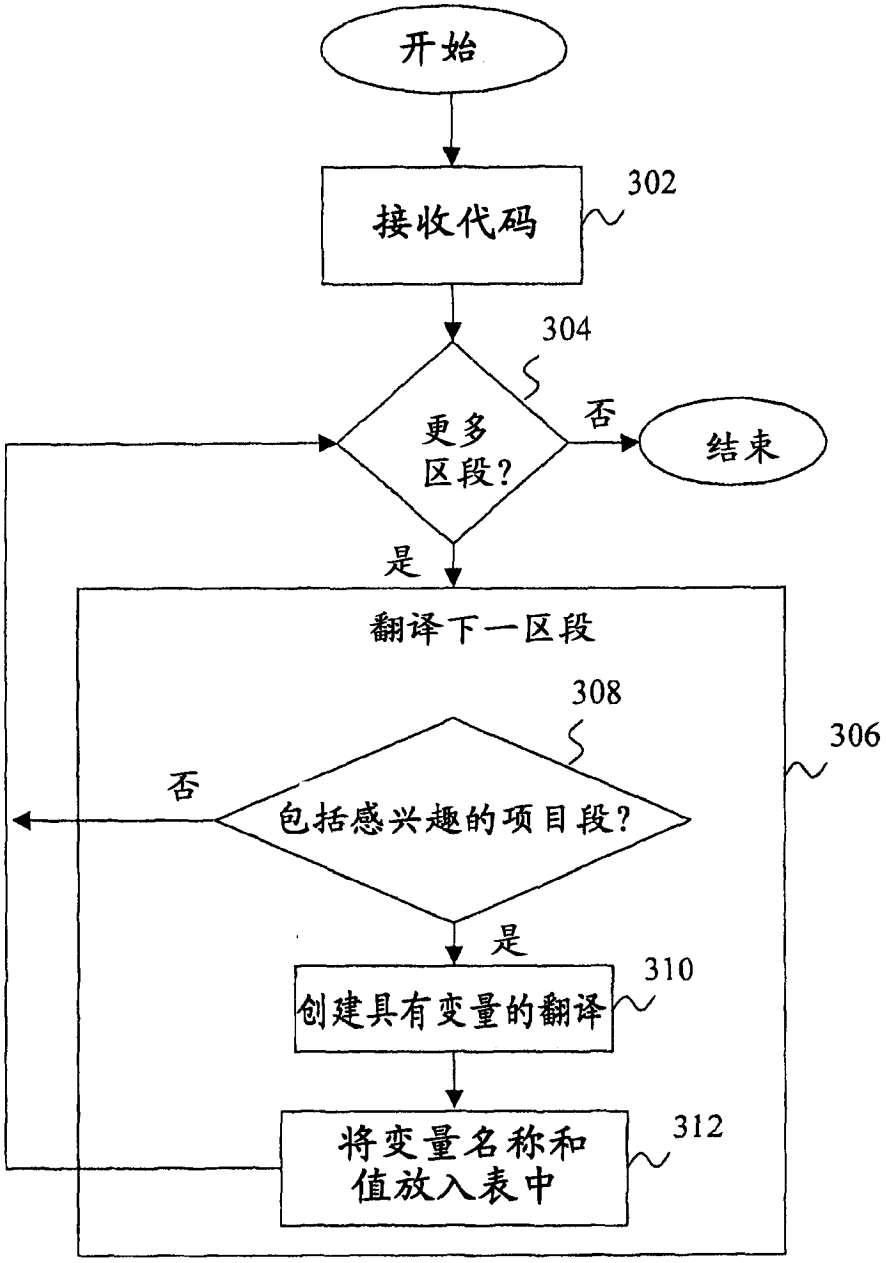

开始
接收代码
302
304
更多区段?
否
结束
是
翻译下一区段
308
306
否
包括感兴趣的项目段?
是
310
创建具有变量的翻译
将变量名称和值放入表中
312

图 3

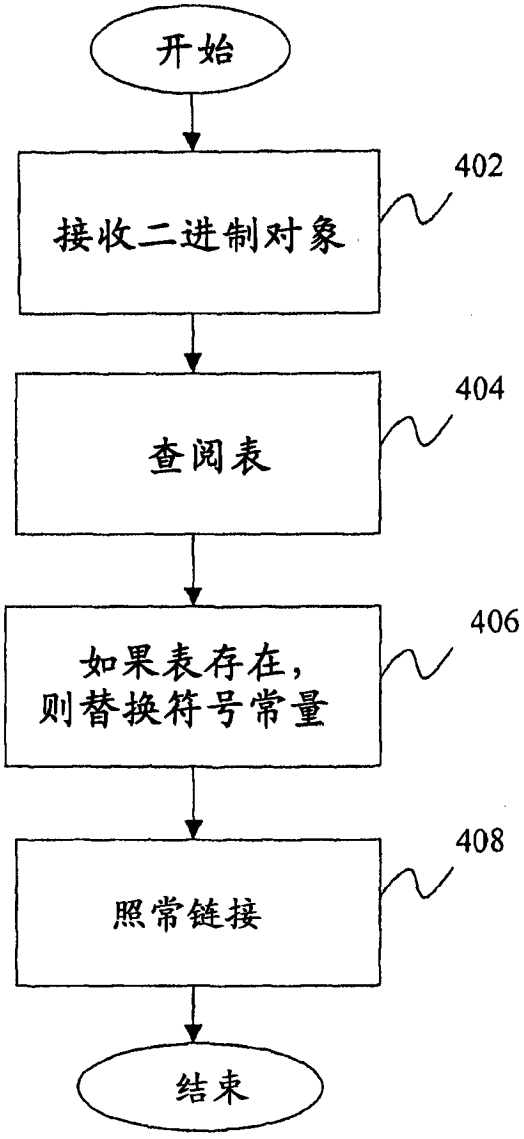


图 4