(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 109313547 B
(45)授权公告日 2022.04.15

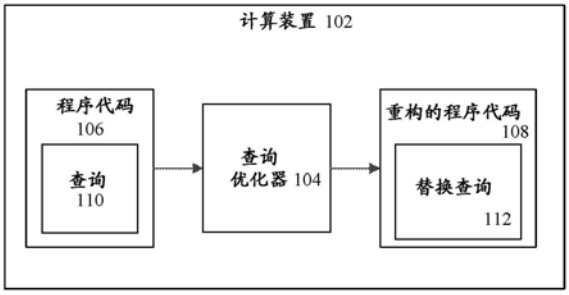
(21)申请号 201780035355.6
(22)申请日 2017.05.31
(65)同一申请的已公布的文献号
申请公布号 CN 109313547 A
(43)申请公布日 2019.02.05
(30)优先权数据
15/174,688 2016.06.06 US
(85)PCT国际申请进入国家阶段日
2018.12.06
(86)PCT国际申请的申请数据
PCT/US2017/035085 2017.05.31
(87)PCT国际申请的公布数据
WO2017/213917 EN 2017.12.14
(73)专利权人 微软技术许可有限责任公司
地址 美国华盛顿州
(72)发明人 E·艾瓦尼 G·塔米尔
A·埃鲁克 E·科勒
(74)专利代理机构 北京市金杜律师事务所 11256
代理人 王茂华
(51)Int.Cl.
G06F 8/72(2018.01)
(56)对比文件
US 2012078878 A1,2012.03.29
US 2016098448 A1,2016.04.07
CN 103970870 A,2014.08.06
CN 101763428 A,2010.06.30
CN 102982103 A,2013.03.20
US 2004210882 A1,2004.10.21
Derek G. Murray.Steno: Automatic Optimization of Declarative Queries.《ACM》.2011,
审查员 罗思异
权利要求书3页 说明书19页 附图6页

(54)发明名称

用于CPU利用率和代码重构的查询优化器

(57)摘要

提供了一种用于提高程序代码中的查询效率的方法、系统、设备和计算机程序产品。在程序代码中检测到多个查询。通过其评估程序代码中的查询的惰性被扩展。查询被分解为多个查询部件。包括多个规则的规则集被应用于查询部件，以生成对多个查询的功能上等效的查询集，该功能上等效的查询集相对于多个查询更高效地评估。



CN 109313547 B

1.一种由计算设备执行的方法,包括:

检测程序代码中的多个查询;

扩展在所述程序代码中评估所述查询所使用的惰性;

在所述程序代码的运行期间,确定所述多个查询将要在其上执行的所述计算设备的运行条件;

确定针对其惰性已经被扩展的查询被配置为:检索数据集,排序所述数据集,并且利用已经被排序的所述数据集的第一元素或者所述数据集的最后元素中的至少一项;以及

在所述程序代码的运行期间,生成功能上等效的查询集,所述功能上等效的查询集相对于所述查询更高效地评估关于所确定的所述运行条件,惰性已经基于规则针对所述查询而被扩展,所述规则最小化所述数据集的检索,所述规则包括在规则集中,所述规则集响应于以下而基于所述计算设备的所述运行条件来确定所述功能上等效的查询集:所述确定针对其惰性已经被扩展的所述查询被配置为:检索所述数据集,排序所述数据集,以及利用已经被排序的所述数据集的所述第一元素或者所述数据集的所述最后元素中的所述至少一项,

所述功能上等效的查询集被配置为检索所述数据集的所述第一元素或者所述数据集的所述最后元素中的所述至少一项,以代替检索所述数据集的全部并且排序所述数据集。

2.根据权利要求1所述的方法,其中所述扩展在所述程序代码中评估所述查询所使用的惰性包括:

检测所述查询中评估可扩展的一个或多个查询;以及

形成包括所述评估可扩展的一个或多个查询的单个查询表达式。

3.根据权利要求1所述的方法,其中所述生成包括:

确定多个查询组件之间的公共逻辑上下文;以及

将所述规则集中对应于所述公共逻辑上下文的第二规则应用于所述多个查询组件。

4.根据权利要求1所述的方法,其中生成的所述功能上等效的查询集通过以下中的至少一项来相对于所述多个查询更高效地评估:与所述多个查询的查询结果相比,生成消耗更少的存储空间的查询结果;与所述多个查询相比,花费更少的时间来执行;与所述多个查询相比,消耗更少的网络带宽;或与所述多个查询相比,消耗更少的处理功率。

5.根据权利要求1所述的方法,其中应用规则集包括:

评估所述规则集中的多个规则组合到查询组件的应用,以生成多个候选的功能上等效的查询集;以及

选择所述多个候选的功能上等效的查询集中的具有最大效率增益的候选的功能上等效的查询集,作为生成的所述功能上等效的查询集。

6.根据权利要求1所述的方法,其中应用规则集包括:

在运行期间,评估规则集中的多个规则组合到查询组件的应用,以生成多个候选的功能上等效的查询集;以及

在运行期间,选择多个候选的功能上等效的查询集中的具有最大效率增益的候选的功能上等效的查询集,作为生成的所述功能上等效的查询集。

7.根据权利要求6所述的方法,其中由所述候选的功能上等效的查询集中的一个或多个提供的所述效率增益至少部分地取决于运行时的执行条件。

8.根据权利要求1所述的方法,进一步包括:

使用机器学习生成用以添加到所述规则集的至少一个规则。

9.一种计算设备,包括:

至少一个处理器电路;以及

至少一个存储器,存储有被配置为由所述至少一个处理器电路执行的程序代码,所述程序代码被配置为执行操作,所述操作包括:

检测程序代码中的多个查询;

扩展在所述程序代码中评估所述查询所使用的惰性;

在所述程序代码的运行期间,确定所述多个查询将要在其上执行的所述计算设备的运行条件;

确定针对其惰性已经被扩展的查询被配置为:检索数据集,排序所述数据集,并且利用已经被排序的所述数据集的第一元素或者所述数据集的最后元素中的至少一项;以及

在所述程序代码的运行期间,生成功能上等效的查询集,所述功能上等效的查询集相对于所述查询更高效地评估关于所确定的所述运行条件,惰性已经基于规则针对所述查询而被扩展,所述规则最小化所述数据集的检索,所述规则包括在规则集中,所述规则集响应于以下而基于所述计算设备的所述运行条件来确定所述功能上等效的查询集:所述确定针对其惰性已经被扩展的所述查询被配置为:检索所述数据集,排序所述数据集,以及利用已经被排序的所述数据集的所述第一元素或者所述数据集的所述最后元素中的所述至少一项,

所述功能上等效的查询集被配置为检索所述数据集的所述第一元素或者所述数据集的所述最后元素中的所述至少一项,以代替检索所述数据集的全部并且排序所述数据集。

10.根据权利要求9所述的计算设备,其中扩展在所述程序代码中评估所述查询所使用的惰性包括:

检测所述查询中评估可扩展的一个或多个查询;以及

形成包括所述评估可扩展的一个或多个查询的单个查询表达式。

11.根据权利要求9所述的计算设备,其中所述生成包括:

确定多个查询组件之间的公共逻辑上下文;以及

将所述规则集中对应于所述公共逻辑上下文的第二规则应用于所述多个查询组件。

12.根据权利要求9所述的计算设备,其中所述生成包括:

评估所述规则集中的多个规则组合到查询组件的应用,以生成多个候选的功能上等效的查询集;以及

选择所述多个候选的功能上等效的查询集中的具有最大效率增益的候选的功能上等效的查询集,作为生成的所述功能上等效的查询集。

13.根据权利要求9所述的计算设备,其中所述生成包括:

在运行期间,评估规则集中的多个规则组合到查询组件的应用,以生成多个候选的功能上等效的查询集;以及

在运行期间,选择多个候选的功能上等效的查询集中的具有最大效率增益的候选的功能上等效的查询集,作为生成的所述功能上等效的查询集。

14.根据权利要求13所述的计算设备,其中由所述候选的功能上等效的查询集中的一

个或多个提供的所述效率增益至少部分地取决于运行时的执行条件。

15.根据权利要求9所述的计算设备，其中所述操作进一步包括：

使用机器学习生成用以添加到所述规则集的至少一个规则。

16.一种计算设备，包括：

至少一个处理器电路；以及

至少一个存储器，存储有被配置为由所述至少一个处理器电路执行的程序代码，所述程序代码包括：

查询检测器，被配置为检测程序代码中的多个查询；

惰性扩展器，被配置为扩展在所述程序代码中评估所述查询所使用的惰性；

等效查询集生成器，被配置为：

在所述程序代码的运行期间，确定所述多个查询将要在其上执行的所述计算设备的运行条件；

确定针对其惰性已经被扩展的查询被配置为：检索数据集，排序所述数据集，并且利用已经被排序的所述数据集的第一元素或者所述数据集的最后元素中的至少一项；以及

在所述程序代码的运行期间，生成功能上等效的查询集，所述功能上等效的查询集相对于所述查询更高效地评估关于所确定的所述运行条件，惰性已经基于规则针对所述查询而被扩展，所述规则最小化所述数据集的检索，所述规则包括在规则集中，所述规则集响应于以下而基于所述计算设备的所述运行条件来确定所述功能上等效的查询集：所述确定针对其惰性已经被扩展的所述查询被配置为：检索所述数据集，排序所述数据集，以及利用已经被排序的所述数据集的所述第一元素或者所述数据集的所述最后元素中的所述至少一项，

所述功能上等效的查询集被配置为检索所述数据集的所述第一元素或者所述数据集的所述最后元素中的所述至少一项，以代替检索所述数据集的全部并且排序所述数据集。

17.根据权利要求16所述的计算设备，其中所述惰性扩展器被配置为通过以下方式扩展惰性：

检测所述查询中评估可扩展的一个或多个查询；以及

形成包括所述评估可扩展的一个或多个查询的单个查询表达式。

18.根据权利要求16所述的计算设备，其中所述等效查询集生成器被配置为通过以下方式生成所述功能上等效的查询集：

确定多个查询组件之间的公共逻辑上下文；以及

将所述规则集中对应于所述公共逻辑上下文的第二规则应用于所述多个查询组件。

19.根据权利要求16所述的计算设备，其中所述等效查询集生成器被配置为通过以下方式生成所述功能上等效的查询集：

在运行期间，评估规则集中的多个规则组合到查询组件的应用，以生成多个候选的功能上等效的查询集；以及

在运行期间，选择多个候选的功能上等效的查询集中的具有最大效率增益的候选的功能上等效的查询集，作为生成的所述功能上等效的查询集。

20.根据权利要求19所述的计算设备，其中由所述候选的功能上等效的查询集中的一个或多个提供的所述效率增益至少部分地取决于运行时的执行条件。

用于CPU利用率和代码重构的查询优化器

背景技术

[0001] 存在各种类型的软件开发应用以供软件开发者使用来开发软件。集成开发环境(IDE)是一种在一个套装中包含多个开发工具的软件开发应用。IDE可以包括诸如源代码编辑器(“代码编辑器”)、建立自动化工具、以及调试器等工具。IDE的示例包括由加拿大渥太华的Eclipse基金会开发的Eclipse™、由加拿大温哥华的ActiveState开发的ActiveState Komodo™、由捷克共和国的JetB rains开发的IntelliJ IDEA、由加利福尼亚州红木城的甲骨文公司开发的Oracle JDeveloper™、由甲骨文公司开发的NetBeans、由加利福尼亚州旧金山的Codenvy开发的Codenvy™、由加利福尼亚州库比蒂诺的苹果公司开发的Xcode®、以及由华盛顿州雷德蒙的微软公司开发的Microsoft®VisualStudio®。

[0002] 许多现代编程语言本身支持对数据的查询。例如,由微软公司开发的Microsoft.NET Framework支持LINQ(语言集成查询)形式的查询,而由加利福尼亚州红木城的甲骨文公司开发的Java®支持流形式的查询。编程语言中对查询的本机支持使开发者能够专注于其程序代码的逻辑部分,因为集成查询功能负责处理对他们的查询的实际实现。这使得开发者可以加速他们的编码。

[0003] 然而,对低效查询的后果不了解或了解不多可能导致低效的程序代码开发。此外,由于当今开发软件的最佳实践,开发者倾向于在他们的代码中将较大的方法/过程拆分成较小的方法/过程以提高可读性,这也会导致在程序代码中查询的低效。

发明内容

[0004] 提供本发明内容以便以简化的形式对将在下面具体实施方式中进一步描述的一系列概念进行介绍。发明内容既不旨在标识所要求保护的主题的关键特征或必要特征,也不旨在用于限制所要求保护的主题的范围。

[0005] 提供了用于提高程序代码中的查询效率的多种方法、系统、设备和计算机程序产品。诸如当在代码编辑器中输入程序代码时或在编译时间期间,在程序代码中检测到多个查询。在程序代码中评估查询的惰性(laziness)被扩展。此外,包括多个规则的规则集被应用于检测到的查询,以生成功能上等效的查询集,该功能上等效的查询集更高效地对检测到的查询进行评估。

[0006] 包括在规则集中的规则可以以任意方式生成,包括由用户手动生成或由算法自动生成,诸如机器学习、代码示例的大数据分析等。一组这样的规则可以显著改进开发者日常工作中出现的最常见错误。

[0007] 单个功能上等效的查询集可以被生成并输入到程序代码中来代替一个或多个现有查询,或者多个候选的功能上等效的查询集可以被生成,从多个候选功能上等效的查询集中选择要被输入到程序代码中的一个查询集。用户可以选择要被输入到程序代码中的候选的功能上等效的查询集,或者候选的功能上等效的查询集可以被自动选择。候选的功能上等效的查询集的选择可能受到在开发期间(例如,在代码输入期间或在编译时间)或者在运行期间是否进行选择的影响。在运行期间,可以(至少部分地)基于运行因素(例如,执行

条件，诸如网络可用性、可用处理能力等)来从多个候选中选择候选的功能上等效的查询集，以作为到程序代码中的输入。

[0008] 下面参照附图详细描述本发明的另外的特征和优点、以及本发明的各个实施例的结构和操作。应该注意的是，本发明不限于本文所述的特定实施例。本文提供这些实施例仅用于说明性目的。基于本文包含的教导，对于相关领域的技术人员来说，另外的实施例将是显而易见的。

附图说明

[0009] 包含在本文中并形成说明书一部分的附图示出了本申请的实施例，并且与说明书一起进一步用于解释实施例的原理并且使相关领域的技术人员能够制造并且使用实施例。

[0010] 图1示出了根据示例实施例的包含被配置为提高程序代码中的查询执行的效率的查询优化器的计算设备的框图。

[0011] 图2示出了根据示例实施例的包括具有被配置为提高程序代码中的查询执行的效率的查询优化器的开发应用的计算设备的框图。

[0012] 图3示出了根据示例实施例的提供用于提高程序代码中的查询执行的效率的过程的流程图。

[0013] 图4示出了根据示例实施例的查询优化器的框图。

[0014] 图5示出了根据示例实施例的提供用于在代码编辑器中生成并呈现建议的替换查询集的过程的流程图。

[0015] 图6示出了根据示例实施例的计算设备的框图，计算设备包括代码编辑器窗口，其显示具有建议的替换查询集的程序代码。

[0016] 图7示出了根据示例实施例的提供用于在编译时间处生成和实施替换查询集的过程的流程图。

[0017] 图8示出了根据示例实施例的提供用于在程序代码中扩展查询执行的惰性的过程的流程图。

[0018] 图9示出了根据示例实施例的惰性扩展器的框图。

[0019] 图10示出了根据示例实施例的提供用于应用规则集以确定用于程序代码的替换查询集的过程的流程图。

[0020] 图11示出了根据示例实施例的等效查询集生成器的框图。

[0021] 图12示出了根据示例实施例的提供用于在多个候选的功能上等效的查询集之间生成和选择的过程的流程图。

[0022] 图13示出了可以用于实施实施例的示例计算设备的框图。

[0023] 通过下面结合附图给出的详细描述，本发明的特征和优点将变得更加明显，其中，相同的附图标记始终表示相应的元件。在附图中，相同的附图标记通常表示相同的、功能相似的和/或结构相似的元件。元件首次出现的附图由相应附图表及中最左边的数字表示。

具体实施方式

[0024] I.简介

[0025] 本说明书和附图公开了包含本发明的特征的一个或多个实施例。本发明的范围不

限于公开的实施例。公开的实施例仅例示了本发明，公开的实施例的变型的版本也由本发明涵盖。本发明的实施例由所附权利要求限定。

[0026] 说明书中对“一个实施例”、“实施例”、“示例实施例”等的引用表示所描述的实施例可以包括特定特征、结构或特性，但是每个实施例可以不必包括特定特征、结构或特性。此外，这些短语不一定指代同一个实施例。此外，当结合实施例描述特定特征、结构或特性时，认为结合其它实施例实现这种特征、结构或特性在本领域技术人员的理解内，而无论是否明确描述。

[0027] 在讨论中，除非另有说明，否则修饰本公开的实施例的特征或多个特征的条件或关系特性的诸如“基本上”和“约”的副词被理解为表示条件或特征被限定在对于预期的应用的实施例的操作可接受的公差内。

[0028] 如下描述了多个示例性实施例。注意的是，本文提供的任何部分/子部分标题不旨在限制。贯穿本文档描述了实施例，并且任何类型的实施例可以包括在任何部分/子部分下。此外，在任何部分/子部分中公开的实施例可以与以任何方式与在相同部分/子部分和/或不同部分/子部分中描述的任何其它实施例组合。

[0029] II.用于优化程序代码中的查询的示例实施例

[0030] 许多现代编程语言本身支持查询。例如，由微软公司开发的Microsoft.NET Framework支持LINQ(语言集成查询)形式的查询，而由加利福尼亚红木城的甲骨文公司开发的Java®支持流形式的查询。编程语言中对查询的本机支持通过处理对它们的查询的实际实现，来使得开发者能够来专注于他们的程序代码的逻辑部分。这使得开发者可以加速他们的编码。然而，对低效查询的后果不了解或了解不多可能导致运行不良的程序代码。例如，这样的程序代码可以包括如下查询，即，检索最终未被使用的数据、检索冗余数据量和/或比所需更频繁地或更早地检索数据。查询可以以许多不同的方式被编写，这些不同的方式产生相同的逻辑结果，但对性能的影响却截然不同。

[0031] 此外，由于当今用于开发软件的最佳实践，开发者倾向于在其代码中将较大的方法/过程拆分为较小的方法/过程，以提高可读性。当前可用的代码优化器可能难以对已经以这种方式开发的代码进行优化。

[0032] 根据实施例，针对低效实现的查询，程序代码被自动分析，并且用一组更高效但逻辑上等效的查询来替换这些查询。这样的实施方式可以实现以在编译时间期间起作用(例如，用于CPU优化)和/或实现为IDE工具和/或IDE插件中的重构建议。例如，在一个实施例中，代码编辑器被配置为确定更高效的查询集并且向开发者建议该更高效的查询集以在开发者的代码中实现。实施例允许可读代码能够由开发者开发，然后自动地将代码转换为更高效的代码。

[0033] 在一个实施例中，查询优化器可以被配置为：(a)标识程序代码中的查询(例如，LINQ、流等)；(b)检测查询的结果以确定其使用/上下文；(c)将查询分解为原子单元；(d)综合(a)、(b)和(c)以确定可以实现查询优化的位置。例如，可以针对一组规则来执行模式匹配，这组规则可以是被显示定义的规则和/或自动生成的规则。例如，规则可以通过算法自动生成，诸如机器学习、代码示例的大数据分析等。这样的一组规则可以显著改进开发者日常工作中出现的最常见的错误。

[0034] 因此，各种实施例提供以下中的一个或多个：(1)用于分析并推荐改进查询性能的

代码变化的工具和过程;(2)用于自动地将查询改变为等效的更优化的查询的工具和过程;(3)用于对查询的自动性能改进的编译器的预处理代码分析工具和过程;(4)用于分析并推荐关于对数据库的查询的代码重构的自动工具和过程(例如.NET中的“实体框架”和其它语言中的等效部分),以及用于实现机器学习的代码优化工具和过程,该机器学习在运行期间对等效查询进行采样,并为场景自动选择并替换最佳查询。

[0035] 如本文所使用的,查询是对数据的查询(或请求),并且也可以称为数据查询。可以向包括数据库、应用、API(应用编程接口)等的任何数据源提供查询。对数据库的查询可以称为“数据库查询”。

[0036] 实施例可以以各种方式实现。例如,图1示出了根据示例实施例的包括被配置为提高程序代码中的查询执行效率的查询优化器104的计算设备102的框图。如图1所示,查询优化器104接收由开发者(写入/输入/修改程序代码的人员)录入的程序代码106。程序代码106包括多个查询110。查询110包括多个查询,其可以包括一个或多个单独的查询运算符和/或一个或多个查询表达式(字符串/一系列查询运算符)。查询优化器104被配置为分析并生成用于查询110的替换查询集,从而输出包括替换查询112的重构程序代码108。重构程序代码108是与原始程序代码106逻辑上等效(执行相同的功能)但是用替换查询112重新编写的程序代码。替换查询112被配置为比原始查询110更高效地执行,诸如通过避免检索程序代码未使用的数据,避免做出冗余数据检索请求,避免执行冗余操作,和/或不太频繁和/或稍后检索数据(增加惰性)。

[0037] 查询优化器104可以独立地实现或者包括在可以由开发者使用以输入或处理程序代码的任何系统或工具中,诸如代码编辑器、代码编译器、代码调试器等。例如,图2示出了根据示例实施例的包括开发应用200的计算设备102的框图,开发应用200包括查询优化器104的。开发应用200是集成开发环境(IDE)的示例。如图2所示,计算设备102包括开发应用200、存储装置210和通信接口208。存储装置210存储程序代码106和108以及规则集216。开发应用200包括源代码编辑器202、编译器204和调试器工具206。源代码编辑器202包括用户接口212。如虚线箭头所示,查询优化器104可以在源代码编辑器202、编译器204和/或调试器工具206中的任何一个或多个中实现或调用。注意的是,示出开发应用200以用于说明的目的并且作为示例实施例,而不是开发应用200的所有特征都必须存在于所有实施例中。此外,在一些实施例中可以存在图2中未示出的附加特征。图2中所示的开发应用200的特征包括描述如下。

[0038] 如图2所示,开发应用200可以在一个或多个计算设备102中实现。例如,源代码编辑器202、编译器204和调试器工具206可以包括在同一计算设备中,或者源代码编辑器202、编译器204和调试器工具206中的一个或多个可以在与源代码编辑器202、编译器204和调试器工具206中的其它分开的一个或多个计算设备中实现。

[0039] 计算设备102可以是任何类型的固定或移动计算设备中的一个或多个,包括移动计算机或移动计算设备(例如,Microsoft®Surface®装置、个人数字助理(PDA)、膝上型计算机、笔记本计算机、诸如Apple iPad™的平板计算机、上网本等)、移动电话、可穿戴计算设备或其它类型的移动装置、或例如台式计算机或PC(个人计算机)的固定计算设备。

[0040] 代码编辑器202可以是被配置为用于编辑本文中其它地方提到的或者否则已知的程序代码的任何专有或常规代码编辑器(例如,Eclipse™的代码编辑器、ActiveState

Komodo™、IntelliJ IDEA、Oracle JDeveloper™、NetBeans、Codenvy ™、Xcode®、Microsoft®VisualStudio®等)。

[0041] 当为应用生成源代码时,开发者可以与源代码编辑器202交互以输入并修改程序代码。例如,开发者可以与源代码编辑器202的用户接口212交互以诸如通过键入、通过语音输入、通过选择建议的代码块等来添加、修改或删除程序代码文本。相应地,用户接口212可以包括一个或多个文本输入框/窗口(例如,图6的代码编辑器窗口604)、语音/话语识别、一个或多个图形用户界面元件(例如,按钮、复选框、单选按钮、下拉菜单等),和/或开发者可与其交互的其它用户界面元件。当完成时或在其它区间处,用户可以通过与“保存”按钮或其它用户界面元件进行交互来保存程序代码。

[0042] 例如,如图2所示,开发者可以与源代码编辑器202的用户界面212交互以生成程序代码106。程序代码106是源代码,其是使用人类可读计算机编程语言编写的计算机指令(可能具有注释)的集合。合适的人类可读计算机编程语言的示例包括C#、C++、Java等。程序代码106可以以一个或多个文件或其它形式接收。例如,程序代码106可以作为一个或多个“.c”文件(当使用C编程语言时)、作为一个或多个“.cpp”文件(当使用C++编程语言时)等接收。当查询优化器104由源代码编辑器202使用以重构程序代码106时,重构程序代码108可以由源代码编辑器202生成并保存。在实施例中,源代码编辑器202可以将规则集216中的规则应用于程序代码106的查询组件(也称为“查询运算符”),以在重构程序代码108中创建更高效的查询集。

[0043] 如图2所示,程序代码106和/或108可以存储在存储装置210中。存储装置210可以包括用于存储数据的任何类型的物理存储硬件/电路中的一种或多种,包括磁盘(例如,在硬盘驱动器中)、光盘(例如,在光盘驱动器中)、磁带(例如,在磁带驱动器中)、诸如RAM装置、ROM装置等的存储器装置、和/或任何其它合适类型的物理存储硬件/电路。

[0044] 可以以诸如通过命令线、图形用户界面等的任何方式调用编译器204。当调用编译器204以执行完全编译时,可以使用“-full”开关或其它开关。编译器204被配置为接收并编译程序代码106(或程序代码108)以生成机器代码222。具体地,编译器204被配置为将程序代码106和/或108转换为以另一计算机语言的形式的机器代码222,其通常具有二进制形式,被称为机器代码或对象代码。在一些情况下,编译器204可以包括多个阶段,并且可以首先将程序代码106转换为中间形式(例如,中间语言),其随后被转换为机器代码222。

[0045] 编译器204可以被配置为在生成机器代码222时对程序代码106和/或108执行一种或多种类型的优化。优化的建立导致在语义上等效于在没有优化的情况下生成的机器代码,但是被配置为在优化的机器代码的执行期间使用较少资源的方式(例如,较少的存储器、较少的进程调用等)。可以执行的优化的示例包括循环优化、数据流优化、基于SSA的优化、代码生成器优化、功能语言优化、过程间优化、和/或相关领域的技术人员已知的其它类型的优化。存在许多特定类型的优化。例如,可以执行“内联(inlining)”,其中由调用者函数调用的被调用者函数被复制到调用者函数的主体中。在特定优化的另一示例中,可以执行“公共子表达消除”,其中单个代码实例用于在源代码中多次计算的量。当编译器204使用查询优化器104来重构程序代码106时,重构程序代码108可以被生成并用于由编译器204生成机器代码222。

[0046] 机器代码222可以被包括在文件(例如,对象或“.obj”文件)中,或者可以以另一种

形式创建/存储，以形成可执行程序或应用。机器代码222可以可选地存储在存储装置210中。

[0047] 当程序代码106和/或108由编译器204编译以用于开发的调试阶段时，调试器工具206可以接收机器代码222。调试器工具206被配置为运行在由机器代码222表示的应用上的调试器(或“调试”、“程序调试”)会话。在调试器会话中，可以使开发者能够逐步执行机器代码222的代码，同时查看由机器代码222的执行产生的变量、阵列、属性和/或输出的值(例如，寄存器、GUI等的内容)，包括访问录入到程序代码106和/或108中的任何调试代码/语句的效果(并且由编译器204传递给机器代码222，以用于调试的目的)。以这种方式，开发者可以能够测试或排查(“调试”)程序代码106和/或108，基于调试器会话的结果使用源代码编辑器202对程序代码106和/或108进行编辑。程序代码106和/或108的修改版本可以由编译器204编译并由调试器工具206接收以进行进一步调试。在调试期间，调试器工具206可以建议和/或重新编写程序代码106中的查询以生成程序代码108。调试器工具206可以包括执行机器代码222的物理的和/或虚拟的一个或多个处理器(例如，中央处理单元(CPU))。

[0048] 当调试器工具206的调试完成并且程序代码106和/或108处于其最终版本时，编译器204可以编译程序代码106和/或108以生成用于开发的发布阶段的机器代码222。可以发布机器代码222的发布版本以供用户使用。

[0049] 通信接口208被配置为将程序代码106和/或108发送到远程实体，以根据实施例接收用于改进程序代码的规则，和/或根据专有或常规的任何合适的通信协议来传送其它数据。通信接口和通信协议的进一步示例将在下一部分中描述。

[0050] 查询优化器104可以以各种方式被配置以执行其功能。例如，图3示出了根据示例实施例的提供用于提高程序代码中的查询执行效率的过程的流程图300。在实施例中，查询优化器104可以根据流程图300进行操作。下面参照图1和图4描述流程图300。图4示出了根据示例实施例的查询优化器104的框图。如图4所示，查询优化器104包括查询检测器402、惰性扩展器404和等效查询集生成器406，这些在下文参照流程图300进行描述。

[0051] 流程图300以步骤302开始。在步骤302中，检测程序代码中的多个查询。如图1所示，查询优化器104接收程序代码106，程序代码106包括查询110。图4的查询检测器402被配置为通过程序代码106进行解析来检测查询。查询检测器402可以以任何方式检测查询，诸如通过将程序代码106中的代码项与已知查询运算符/组成的预定列表等进行比较等。例如，当搜索LINQ查询时，查询检测器402可以通过程序代码106进行解析，用于已知的LINQ运算符，诸如Select、Where、Sum、Min、Max、Average、Aggregate、Join、GroupJoin、OrderBy、GroupBy、Union、Contains、Count、ToList、ToDictionary等。每个找到的查询运算符(例如，由文本匹配找到)由查询检测器402指示作为对程序代码106的查询。

[0052] 在一个说明性示例中，查询检测器402可以解析以下程序代码以用于查询：

[0053]
```
public void Main()
    var 1stOfPrimes = ReturnListOfPrimes(1000);
    var 1stOrdered = 1stOfPrimes.OrderBy(x => x).ToList();
    var 1stOrderedDesc = 1stOrdered.OrderByDescending(x => x).ToList();
     Console.WriteLine("Biggest prime in range is: {0}",
1stOrderedDesc.First());
```

[0054] 在该示例中，查询检测器402可以检测第三行代码中的Orderby(x=>x)和ToList()的LINQ查询，以及第四行代码中的OrderByDescending(x=>x)和ToList()的LINQ查询。

[0055] 注意的是，步骤302(和流程图300的其余部分)可以由查询优化器104在任何合适的代码开发工具或应用中实现。例如，图5示出了根据示例实施例的提供用于在代码编辑器中生成和呈现建议的替换查询集的过程的流程图500。在实施例中，图3的流程图300可以实现图5的流程图500。例如，流程图300的步骤302可以实现流程图500的步骤502，并且步骤504可以是流程图300的附加步骤。流程图500描述如下。

[0056] 在步骤502中，在代码编辑器中检测程序代码中的查询。在实施例中，查询检测器402可以被配置为在代码编辑器(诸如代码编辑器202(图2))中的程序代码106中检测查询，其中开发者录入并编辑程序代码。每当开发者保存代码时，查询检测器402可以在开发者录入代码时响应于开发者的请求(例如，通过点击“查询检测”按钮)而基于周期和/或以任何其它期望的时间或基础在程序代码106中执行查询项搜索。

[0057] 例如，图6示出了根据示例实施例的包括显示程序代码106的代码编辑器窗口604的计算设备102的框图。显示器602可以是任何合适类型的显示装置或屏幕，包括LCD(液晶显示器)、CRT(阴极射线管)显示器、LED(发光二极管)显示器、等离子显示器等。代码编辑器窗口604是在显示器602中由代码编辑器202显示的窗口(框架或无框架)，作为用于与代码编辑器窗口604中显示的程序代码106交互的图形用户界面(GUI)。代码行606、608和610各自是任何合适的编程语言的一个或多个代码行，正如相关领域的技术人员所知的。在实施例中，查询检测器402根据步骤502解析包括代码行606、608和610的程序代码106以用于查询。

[0058] 在步骤504，呈现用于代码编辑器的选项，以自动重构程序代码来用功能上等效的查询集替换多个查询。在实施例中，在查询优化器104检测到查询并确定出更高效的查询以替换在程序代码106中的检测到的查询之后(如本文其它地方进一步描述的)，代码编辑器202可以提供用于开发者接受或拒绝所确定的更高效的查询的选项。例如，如图6所示，建议的替换查询集612可以显示在代码编辑器窗口604中，以供由开发者接受或拒绝。建议的替换查询集612是由查询优化器104确定的比在程序代码106中检测到的查询集更高效的建议查询集。如果开发者接受建议的替换查询集612(例如，通过点击按钮)，则代码编辑器202可以用程序代码106中的建议替换查询集612来替换检测到的查询集，以在代码编辑器窗口604中生成并显示重构程序代码108。如果开发者拒绝建议的替换查询集612，则不对程序代码106进行改变，并且不再显示建议。注意的是，在另一实施例中，建议的替换查询集612可以在程序代码106中自动实现(不需要请求开发者接受)。

[0059] 在另一实施例中，如上所述，查询检测器402可以被配置为在编译期间在程序代码

106中检测查询。例如，图7示出了根据示例实施例的提供用于在编译时间处生成和实现替换查询集的过程的流程图700。在实施例中，图3的流程图300可以实现图7的流程图700。例如，流程图300的步骤302可以实现流程图700的步骤702，并且步骤704可以是流程图300的附加步骤。流程图700描述如下。

[0060] 在步骤702中，在编译期间在程序代码中检测查询。在实施例中，查询检测器402可以被配置为在编译程序代码106时在编译器(诸如编译器204(图2))中的程序代码106中检测查询(例如，通过文本匹配等)。

[0061] 在步骤704中，基于程序代码的版本在编译器中生成编译的代码，其中多个查询用功能上等效的查询集替换。在实施例中，如下面进一步详细描述的，查询优化器104被配置为由查询检测器402针对在程序代码106中检测到的查询生成更高效的替换查询集，以在程序代码106的副本中用替换查询集替换检测到的查询，以生成重构程序代码108，并通过编译重构程序代码108生成编译版本的程序代码106。以这种方式，生成包括更高效操作查询集的编译机器代码而不改变程序代码106，以使得当随后在代码编辑器202中编辑程序代码106时，开发者不会看到改变的查询。然而，在另一实施例中，替换查询集可以在编译时间处在程序代码106中自动实现(不需要请求开发者接受)。

[0062] 返回参照图3的流程图300，在步骤304中，惰性被扩展，其中，通过该惰性评估程序代码中的查询。在实施例中，惰性扩展器404被配置为分析由查询检测器402在程序代码106中检测到的查询，并且扩展查询的评估的惰性(当执行程序代码时)。“惰性评估”是指延迟表达式的评估直到需要其值的评估技术。惰性扩展器404被配置为尽可能晚地接收查询并推迟尽可能多的其运算符的执行。因此，惰性扩展器404分析程序代码106以确定可以使其评估延迟的查询，并确定以延迟方式执行查询的等效的一个或多个查询语句。

[0063] 例如，图8示出了根据示例实施例的提供用于在程序代码中扩展查询执行的惰性的过程的流程图800。在实施例中，惰性扩展器404根据流程图800进行操作。流程图800如下参照图9进行描述。图9示出了根据示例实施例的惰性扩展器404的框图。如图9所示，惰性扩展器404包括可扩展查询检测器902和查询表达式汇编器904。

[0064] 流程图800从步骤802开始。在步骤802中，检测评估可扩展的一个或多个查询。在实施例中，惰性扩展器404的可扩展查询检测器902分析程序代码106以确定可以使其评估延迟的查询，因为其输出值在程序代码106中的当前位置并不需要，而是稍后使用。在这样的实施例中，可扩展查询检测器902被配置为针对特定查询，在程序代码106中解析查询以在需要查询的值处或附近找到一个位置(例如，表达式、代码行等)，诸如将值直接向用户输出的位置，或者表达式使用该值来为用户生成输出的位置。

[0065] 在实施例中，可扩展查询检测器902通过为检测到的查询906中的每个查询建立诸如操作树的数据结构以开始，其中，检测到的查询906包括由查询检测器402在程序代码以及非查询程序代码操作中检测到的查询。可扩展查询检测器902在同一操作树下一起列出多个相关查询。这样的操作树可以表示为一个图表，其中每个节点表示不从数据源获取实际数据(例如，对数据执行操作)的查询的匹配查询运算符，并且操作树的每个叶表示从数据源获取实际数据的查询的查询运算符。可扩展查询检测器902通过遵循程序代码中的每个变量/查询运算符/方法/代码语句并递归地将数据添加到操作树来建立操作树。因此，可扩展查询检测器902接收查询并将它们转换为相关的树。

[0066] 例如,在一个示例中,可扩展查询检测器902可以分析以下程序代码以确定可以使其评估延迟的一个或多个LINQ查询:

[0067] var lstOfPrimes＝ReturnListOfPrimes(100);

[0068] var lstOrdered＝lstOfPrimes.OrderBy(x＝＞x).ToList();

[0069] Console.WriteLine(“Smallestprime in range is:{0}”,lstOrdered.First());

[0070] 该示例程序代码包括检测到的OrderBy、ToList和First的LINQ查询。可扩展查询检测器902指定以下操作树:

[0071] stOfPrimes-ReturnListOfPrimes-OrderBy-ToList-First

[0072] 其中lstOfPrimes是操作树的最高级元素,First是操作树的最低级元素。在该示例中,可扩展查询检测器902将OrderBy和ToList指示为可扩展的(延迟的执行),因为lstOrdered的值不是立即需要的(是后续代码行的输入),但是First是叶节点,因此不指示为可扩展的。

[0073] 在另一示例中,可扩展查询检测器902可以分析以下程序代码以确定可以使其评估延迟的一个或多个LINQ查询:

[0074] var lstOfPrimes＝ReturnListOfPrimes(1000);

[0075] var lstOrdered＝lstOrPrimes.OrderBy(x＝＞x).ToList();

[0076] var LastOrdered＝lstOfPrimes.OrderByDescending(x＝＞x).ToList();

[0077] Console.WriteLine(“Smallest prime in rangeis:{0}”,lstOrdered.First());

[0078] Console.WriteLine("Biggest prime in range is:{0}”,LastOrdered.First());

[0079] 该示例程序代码包括检测到的OrderBy、ToLis、OrderByDescending、ToList、First和First的LINQ查询。可扩展查询检测器902指定具有以下第一和第二分支的操作树:

[0080] lstOfPrimes-ReturnListOfPrimes-OrderBy-ToList-First

[0081] 和

[0082] lstOfPrimes-ReturnListOfPrimes-OrderByDescending-ToList-First

[0083] 在该操作树中,公共的lstOfPrimes元素是最高级别的元素,OrderBy和OrderByDescending节点从链接到lstOfPrimes元素的公共ReturnListOfPrimes节点分支。在该示例中,可扩展查询检测器902将第一分支中的OrderBy和ToList以及第二分支中的OrderByDescending和ToList指示为可扩展(延迟的执行),因为lstOrdered和LastOrdered的值不是立即需要的(是后续代码行的输入),但是第一个分支中的First和第二个分支中的First均是叶节点,因此不指示为可扩展的。

[0084] 在又一示例中,可扩展查询检测器902可以分析以下示例程序代码以确定可以使其评估延迟的一个或多个LINQ查询。该示例程序代码被设计为查找2到1000范围内的最后一个素数。

```
        public List<int> ReturnListOfPrimes(int range)
        {
            return Enumerable.Range(2, range).Where(IsPrime).ToList();
        }
        public Dictionary<int, double. ReturnDictionaryOfPower(List<int> 1stOfInts, int
pow)
        {
            return 1stOfInts.ToDictionary(x => x, x => Math.Pow(x, pow));
        }
        public void MainMethod()
        {
            var 1stOfPrimes = ReturnListOfPrimes(1000);
            var power = ReturnDictionaryOfPower(1stOfPrimes, 2);
            var lastItem = power.Last();
            Console.WriteLine("Last prime : {0}, {1}", lastItem,Key, lastItem.Value);
        }
```

[0085]

[0086] 在该示例中，第三方法（“MainMethod”）引用了两个早期方法（“ReturnListOfPrimes”和“ReturnDictionaryOfPower”）。第一方法包括Range、Where和ToList的查询组件。第二方法包括ToDictionary查询组件。第三方法包括Last查询组件。查询检测器402以三种方法检测这些查询(图3的步骤302)。如图9所示，可扩展查询检测器902接收检测到的查询906，其包括在程序代码中由查询检测器402检测到的查询。在实施例中，可扩展查询检测器902分析检测到的查询906和程序代码106(例如，通过生成操作树，这里为了简洁未示出)以确定是否可以在程序代码106中惰性地评估检测到的查询。

[0087] 特别地，第三方法包括四个代码行。当执行第三方法时，第一行针对第一值访问第一方法，第二行针对第二值(基于第一值)访问第二方法，第三行执行与第二值相关的查询(Last)以生成第三值，第四行基于第三值生成输出。因此，可扩展查询检测器902确定第一和第二行生成不立即需要(例如，不向用户或数据库输出)的值，而是用作后续代码行的输入，因此第一行和第二行中的查询的评估可以延迟。如图9所示，可扩展查询检测器902输出可扩展查询908，其指示由可扩展查询检测器902确定为可扩展的查询。

[0088] 返回参照图8中的流程图800，在步骤804中，形成包括一个或多个评估可扩展查询的单个查询表达式。在实施例中，查询表达式汇编器904接收可扩展查询908，并生成包括被确定为评估可扩展的查询(在步骤802中)的单个查询表达式。如图9所示，查询表达式汇编器904生成修改的查询910，其包括单个查询表达式，其组合由可扩展查询检测器902确定的针对程序代码的评估可扩展查询。注意的是，对于特定程序代码，查询表达式汇编器904可以生成这样的组合评估可扩展查询的查询表达式中的一个或多个。由查询表达式汇编器904执行以将单独的查询组合成单个查询表达式的技术对于相关领域的技术人员将是已知的，并且将取决于特定的查询语言。在一个示例中，表达式中的变量可以由在其它地方使用

的表达式来替换以确定该变量。

[0089] 例如,关于上述示例程序代码,查询表达式汇编器904可以针对“var lastItem”表达式生成如下单个查询表达式,其包括第一和第二方法的LINQ查询以及第三方法的前三行:

[0090]
```
Public void MainMethod()
{
    var lastItem = Enumerable.Range(2, 1000).Where(IsPrime).ToList().ToDictionary(x => x, x => Math.Pow(x, pow)).Last();
    Console.WriteLine("Last prime : {0},{1}", lastItem.Key, lastItem.Value);
}
```

[0091] 如上所示,“var lastItem”表达式是一组聚合的查询组件,其包括所有三种方法的Range、Where、ToList、ToDictionary和Last查询组件,并且在功能上等效于上面进一步示出的原始程序代码。以这种方式,在“MainMethod”被执行时,Range、Where、ToList、ToDictionary和Last查询组件被惰性评估,仅在紧邻“lastItem”输出被呈现之前进行评估。

[0092] 注意的是,在实施例中,查询表达式汇编器904可以通过接收由可扩展查询检测器902针对查询而生成的操作树来操作,并且在操作树数据结构中,对这样的查询操作符进行表示/标记,即,在不是叶节点的情况下明确地使得查询执行被“禁用”。换言之,查询表达式汇编器904可以以这样的方式标记操作树,即,查询执行仅在叶节点上发生,从而将查询的惰性扩展到最新的时间点。

[0093] 返回参照图3中的流程图300,在步骤306中,将包括多个规则的规则集应用于查询组件,以生成对多个查询的功能上等效的查询集,功能上等效的查询集相对于多个查询更高效地评估。在实施例中,等效查询集生成器406(图4)被配置为分析检测到的查询(图1的查询110)并生成备选查询集(图1的替换查询112),该备选查询集与检测到的查询在功能上等效,但比检测到的查询的原始配置更高效地执行。例如,等效查询集生成器406可以生成替换查询112以避免检索未被程序代码使用的数据,以避免进行数据的冗余检索,以避免对对输出没有影响的数据执行操作等。为了实现这些效率,等效查询集生成器406可以移除查询组件、添加查询组件和/或修改查询组件。

[0094] 等效查询集生成器406可以以各种方式执行其功能。例如,图10示出了根据示例实施例的提供用于应用规则集以确定用于程序代码的替换查询集的过程的流程图1000。在实施例中,等效查询集生成器406可以根据流程图1000进行操作。以下关于图11描述流程图1000。图11示出了根据示例实施例的等效查询集生成器406的框图。如图11所示,等效查询集生成器406包括上下文确定器1104和规则选择器1106,并且与包含规则集216的存储装置210通信地耦合。

[0095] 流程图1000以步骤1002开始。在步骤1002中,确定多个查询组件之间的公共逻辑上下文。如图1所示。如图11所示,上下文确定器1104接收修改的查询910,其是由查询检测器402检测并且由惰性扩展器404重新写入用于惰性评估的查询。上下文确定器1104针对公共逻辑上下文分析修改的查询910。例如,上下文确定器1104可以将修改的查询910(其可以

是单个查询表达式)分解成查询组件集,并将查询组件彼此进行比较。当检测到类似或相关类型的查询组件(例如,OrderBy和OrderByDescending等)时,建立查询组件之间的公共逻辑上下文。如图11所示,上下文确定器1104输出上下文相关联的查询1114,其包括上下文相关联的修改的查询910的查询组件组。

[0096] 在步骤1004中,将规则集中的对应于公共逻辑上下文的规则应用于多个查询组件。如图11所示,规则选择器1106接收上下文相关联的查询1114。规则选择器1106分析上下文相关联的查询1114,以适用于规则集216中的规则,诸如第一规则1108、第二规则1110等。规则集216中的每个规则被配置为被应用于相应公共上下文的查询组件,并且重新编写查询组件以实现更高的效率,诸如通过添加、修改和/或删除上下文相关联的查询的查询组件。规则集216中可以包括任何数量和种类的规则。为了说明的目的,下文描述了可以包括在规则集216中的规则的七个示例。每个规则通过示例规则名称、摘要以及规则的功能/机制的描述来在下文描述:

[0097] (A)规则名称:OrderBy最小化器

[0098] 摘要:用于消除不必要的OrderBy操作。

[0099] 规则机制:OrderBy最小化器逐步执行程序代码中找到的OrderBy运算符列表,并找到其结果未在程序代码中使用的任何OrderBy运算符(其中"使用的"表示OrderBy运算符执行的排序不依赖后续表达式或方法—不由后续表达式或方法输出或请求)。对于找到的这种OrderBy运算符,OrderBy最小化器从程序代码中删除作为冗余的OrderBy运算符,并重新编写受删除代码行语句(例如,替换变量名称等)影响的程序代码的任何其它行。

[0100] (B)规则名称:ToCurrent集合

[0101] 摘要:用于执行以下"ToCurrent"类型的运算符连续多次—"ToList"、"ToDictionary"等的复杂查询的情况。这些ToCurrent运算符中的每一个较昂贵,并且在它们的序列中,除了序列中的最后一个运算符之外,它们是冗余的。

[0102] 规则机制:以下模式被检测—聚合的查询组件集中的两个或更多个这样的运算符(例如,在流程图800的步骤804中生成的包括多个查询组件的单个查询表达式)。全部ToCurrent类型运算符除了这一个之外(在不改变逻辑的情况下)将从聚合的查询组件集中丢弃。

[0103] (C)规则名称:DataRetrival最小化器—类型1

[0104] 摘要:在查询运算符请求的大量数据最终未被使用时使用,因此可以安全地减少或根本不执行数据检索。

[0105] 规则机制:以下模式被检测—查询运算符检索数据,检索的数据被排序,并且仅使用排序数据的第一个或最后一个元素。上述查询被仅分别从数据执行最小值元素或最大值元素检索的查询所替换。

[0106] (D)规则名称:DataRetrival最小化器-类型2

[0107] 摘要:在查询运算符请求的大量数据最终未使用时使用,因此可以安全地减少或根本不执行数据检索。

[0108] 规则机制:以下模式被检测—查询运算符检索数据,并且仅使用数据的第一个或最后一个元素。上述查询被仅从数据中检索任何随机数据元素的查询替换。

[0109] (E)规则名称:DataRetrival最小化器—类型3

[0110] 摘要:在检索到单个非常大的数据实体但仅一部分数据被最终使用时使用。

[0111] 规则机制:以下模式被检测—请求可能是大数据集的数据(例如,XML数据、JSON数据、整个文件等),同时仅将使用检索到的数据的子集。上述查询被仅检索数据子集的查询替换。

[0112] “DataRetrival最小化器—类型3”规则的适用性的示例包括:

[0113] (1)请求检索文件,但最终仅使用文件的元数据来确定文件的大小(而不是使用文件的内容)。该规则可以用于避免使用查询运算符来检索整个文件,而是使用查询运算符来检索文件元数据(其可以是小于整个文件大小几倍)。

[0114] (2)请求单个数据库行或实体,并且最终仅使用该行/实体的单个列。整个DB行/实体可能非常大。该规则可以用于将查询运算符替换为仅检索所需列的另一查询运算符。

[0115] (F)规则名称:枚举迭代最小化器

[0116] 摘要:在查询利用单个值时使用,因此无需在存储器中建立整个数据结构。

[0117] 规则机制:以下模式被检测-在“ToCurrent”类型运算符之后仅使用单个值。用后一个切换第一个。

[0118] (G)规则名称:使用Last()运算符时的反转枚举

[0119] 摘要:用于仅需要最后一项的情况,因此自始至终枚举效率低。通过以相反的顺序枚举,我们提高了性能。

[0120] 规则机制:以下模式被检测-在“ToCurrent”类型运算符之后仅使用最后一个值。将枚举替换为其“反转”匹配运算符和/或反转枚举范围值,并用运算符“First”替换运算符“Last”。

[0121] 作为进一步的说明,针对以上生成的单个查询表达式应用示例规则,为了便于描述,以下重复该表达式:

[0122] var lastItem＝Enumerable.Range(2,1000).Where(IsPrime).ToList().ToDictionary(x＝＞x,

[0123] x＝＞Math.Pow(x,pow)).Last();

[0124] 如上所述,惰性评估被扩展(流程图300的步骤304;惰性扩展器404)以生成该功能上等效的聚合查询组件集。此外,通过应用规则集216中的规则(流程图300的步骤306;等效查询集生成器406),可以进一步提高该单个查询表达式的效率。

[0125] 例如,可以应用ToCurrent集合最小化器规则(上述规则(B))。在这种情况下,上下文确定器1104检测上述查询语句中的ToList和ToDictionary运算符,其建立公共逻辑上下文。规则选择器1106应用ToCurrent集合最小化器规则,该规则与规则集216中的该公共逻辑上下文相关联。ToCurrent集合最小化器规则从查询语句中删除除最后一个之外的所有ToCurrent运算符,以生成以下更高效但功能上等效的查询表达式:

[0126] var lastItem＝Enumerable.Range(2,1000).Where(IsPrime).ToDictionary(x＝＞x,x＝＞Math.Pow(x,pow)).Last();

[0127] 此外,可以使用枚举迭代最小化器规则(上述规则(F))以便将Last()/First0运算符重新定位在位于ToCurrent语句(在这种情况下为ToDictionary)之前。以这种方式,查询语句迭代直到找到第一个项,因此所有数据结构未被填充。相反,ToDictionary运算符将在单个实体上运行。因此,上下文确定器1104在上述查询语句中确定紧随Last运算符的

ToDictionary运算符,以建立公共逻辑上下文。规则选择器1106应用可枚举迭代规则,该规则与该公共逻辑上下文相关联。可枚举迭代规则将Last运算符重新定位在ToDictionary运算符之前,以生成以下更高效的但功能上等效的查询表达式:

[0128] var lastItem＝Enumerable.Range(2,1000).Where([IsPrime).Last().ToDictionary(x＝＞x,x＝＞Math.Pow(x,pow))

[0129] 更进一步,可以使用"当使用Last()运算符时反转枚举"(上述规则(G)),以便反转搜索顺序来搜索最后一项。这是因为在查找最后一项时以反转的顺序进行更高效。因此,该规则以相反的顺序枚举数量并取出第一个元素而不是最后一个元素。因此,上下文确定器1104在上述查询语句中确定ToDictionary运算符之前的Last运算符以建立公共逻辑上下文。规则选择器1106应用优化可枚举规则,该规则与该公共逻辑上下文相关联。优化可枚举规则用First运算符替换Last运算符,以生成如下更高效的但功能上等效的查询表达式:

[0130] var lastItem＝Enumerable.Range(1000,2).Where(IsPrime).First().ToDictionary(x＝＞x,x＝＞ath.Pow(x,pow))

[0131] 这个最终查询在从1100开始的整数上向下迭代,找到其中的第一个素数,并且比上面进一步示出的原始的三个方法版本更高效地执行(例如,使用更少的存储器来存储数据,并且更好地使用CPU)。

[0132] 注意的是,在实施例中,等效查询集生成器406可以顺序地应用规则集216中的规则,直到不再有规则可以被应用于确定单个功能上等效的查询集。在另一实施例中,可以由等效查询集生成器406确定多组不同的功能上等效的查询集,其中每个确定的查询集适用于替换在程序代码106中的检测到的查询。在这样的实施例中,规则选择器1106可以使开发者能够手动选择功能上等效的查询集中的一个以应用于程序代码106,以生成重构的程序代码108。备选地,规则选择器1106可以自动进行选择。

[0133] 例如,图12示出了根据示例实施例的提供用于在多个候选功能上等效的查询集之间生成并选择的过程的流程图1200。在实施例中,规则选择器1106可以执行流程图1200。流程图1200描述如下。

[0134] 在步骤1202中,规则集中的多个规则组合对查询组件的应用被评估,以生成多个候选功能上等效的查询集。在实施例中,规则选择器1106可以将规则的不同组合和/或不同顺序应用于程序代码以生成不同的功能上等效的查询集。例如,特定程序代码可以包括查询表达式中的OrderBy、GroupBy和ThenBy查询组件。规则选择器1106可以将第一规则应用于查询表达式,以用第一查询运算符替换OrderBy和GroupBy,以生成第一功能上等效的查询集,然后备选地,可以将第二规则应用于查询表达式,以用第二查询运算符替换GroupBy和Thenby,以生成第二功能上等效的查询集。以这种方式,为同一查询表达式生成两个功能上等效的查询集,其中每个查询集可以具有其自身的效率特征。可以根据查询表达式中的特定查询运算符以及规则集216中可用的规则来为同一查询表达式生成任意数量的功能上等效的查询集。

[0135] 在步骤1204中,将多个候选的功能上等效的查询集中的具有最大效率增益的一个候选的功能上等效的查询集选择为所生成的功能上等效的查询集。在上述示例中,规则选择器1106可以使得开发者能够选择第一和第二功能上等效的查询集中的哪一个来重构到程序代码中,或者可以使得自动选择第一和第二查询组件中的哪一个来重构到程序代码

中。例如，规则选择器1106可以选择可用的功能上等效的查询集中具有最高效率的一个。这种最大效率可以以各种方式确定。

[0136] 例如，规则集216中的每个规则可以具有相应的效率值(例如，0-1的数字)，其指示针对该规则的相对效率。规则选择器1106可以针对特定的功能上等效的查询集，将针对每个规则的效率值进行组合，以确定该功能上等效的查询集的整体效率值，然后可以比较所有功能上等效的查询集的整体效率值，以确定哪一个效率最高。在其它实施例中，规则选择器1106可以以其它方式确定最大效率。

[0137] 注意的是，在实施例中，流程图1200可以在编译时间或在运行时执行。例如，在运行时，程序代码106的可执行版本或重构的程序代码108(例如，机器代码222)可以由执行引擎(例如，在操作系统等中)执行。根据运行实施例，可以以考虑实际系统健康状态的方式确定最佳候选功能上等效的查询集。这是因为由一个或多个候选功能上等效的查询集提供的效率增益可以至少部分地取决于运行时的执行条件。例如，在运行处，程序代码的可执行版本可以执行，并且在特定查询将被执行时，可以通过程序代码对规则选择器1106进行调用(例如，可以由执行引擎实现)，以选择哪一个候选功能上等效的查询集来实现基于运行条件(例如，网络条件、可用处理带宽、可用处理能力等)。

[0138] 例如，在正常运行条件(例如，全网络可用性等)中，候选的功能上等效的查询集A可能比功能上等效的查询集B更高效。然而，在特定运行期间，不期望的某些情况可能干扰功能上等效的查询集A的查询组件的性能，诸如较慢的因特网连接等。因此，在这样的运行条件(例如，不良的网络响应)中，功能上等效的查询集B可以实现更高的效率，因此可以由规则选择器1106选择以在该特定运行(而不是查询集A)期间执行。因此，在实施例中，规则选择器1106可以被配置为使得能够在运行处选择候选功能上等效的查询集。

[0139] 注意的是，可以以包括手动或自动的任何方式将规则添加到规则集216。例如，开发者可以基于开发者的经验将规则添加到规则集216，包括希望用开发者看到或已经知道的查询来修复特定的效率问题。在另一实施例中，自动机制可以生成新规则，包括结合机器学习的自动机制。机器学习可以用于在运行期间对等效查询进行采样以得到相对效率，并且基于此，自动为该场景选择最有效的查询。对于被提供以规则的每个查询优化，查询优化器104可以维持由查询优化产生的影响的详细分析。

[0140] III.示例移动和固定装置实施例

[0141] 计算设备102、查询优化器104、编译器204、开发应用200、源代码编辑器202、编译器204、调试器工具206、查询检测器402、惰性扩展器404、等效查询集生成器406、可扩展查询检测器902、查询表达式汇编器904、内容确定器1104、规则选择器1106、流程图300、流程图500、流程图700、流程图800、流程图1000和流程图1200可以以硬件或与软件和/或固件组合的硬件来实现。例如，查询优化器104、编译器204、开发应用200、源代码编辑器202、编译器204、调试器工具206、查询检测器402、惰性扩展器404、等效查询集生成器406、可扩展查询检测器902、查询表达式汇编器904、内容确定器1104、规则选择器1106、流程图300、流程图500、流程图700、流程图800、流程图1000和/或流程图1200可以被实现为配置为在一个或多个处理器中执行并存储在计算机可读存储介质中的计算机程序代码/指令。可选地，计算设备102，查询优化器104，编译器204，开发应用200，源代码编辑器202、编译器204、调试器工具206、查询检测器402、惰性扩展器404、等效查询集生成器406、可扩展查询检测器902、

查询表达式汇编器904、内容确定器1104、规则选择器1106、流程图300、流程图500、流程图700、流程图800、流程图1000和/或流程图1200可以实现为硬件逻辑/电路。

[0142] 例如，在实施例中，查询优化器104、编译器204、开发应用200、源代码编辑器202、编译器204、调试器工具206、查询检测器402、惰性扩展器404、等效查询集生成器406、可扩展查询检测器902、查询表达式汇编器904、内容确定器1104、规则选择器1106、流程图300、流程图500、流程图700、流程图800、流程图1000和/或流程图1200中的一个或多个以任意组合可以一起在SoC中实现。SoC可以包括集成电路芯片，其包括处理器(例如，中央处理单元(CPU)、微控制器、微处理器、数字信号处理器(DSP)等)、存储器、一个或多个通信接口中和/或其它电路中的一个或多个，并且可以可选地执行接收的程序代码和/或包括嵌入式固件以执行功能。

[0143] 图13描绘了其中可以实现实施例的计算设备1300的示例性实现。例如，计算设备102和/或客户端计算设备104可以在类似于固定或移动计算机实施例中的计算设备1300的一个或多个计算设备中实现，包括计算设备1300的一个或多个特征和/或可选特征。这里提供的计算设备1300的描述被提供以用于说明的目的，而不旨在限制。如相关领域的技术人员已知的，实施例可以在其它类型的计算机系统中实现。

[0144] 如图13所示，计算设备1300包括一个或多个处理器，称为处理器电路1302、系统存储器1304、以及将包括系统存储器1304的各种系统组件耦合到处理器电路1302的总线1306。处理器电路1302是在一个或多个物理硬件电路装置元件和/或集成电路装置(半导体材料芯片或管芯)作为中央处理单元(CPU)、微控制器、微处理器和/或其它物理硬件处理器电路中实现的电气和/或光学电路。处理器电路1302可以执行存储在计算机可读介质中的程序代码，诸如操作系统1330的程序代码、应用程序1332、其它程序1334等。总线1306表示几种类型的总线结构中的任何一种或多种，包括存储器总线或存储器控制器、外围总线、加速图形端口以及使用各种总线架构中的任何一种的处理器或局部总线。系统存储器1304包括只读存储器(ROM)1308和随机存取存储器(RAM)1310。基本输入/输出系统1312(BIOS)存储在ROM 1308中。

[0145] 计算设备1300还具有以下驱动器中的一个或多个：用于从硬盘读取并写入硬盘的硬盘驱动器1314；用于从可移除磁盘1318读取或写入可移除磁盘1318的磁盘驱动器1316；以及从可移除光盘1322读取或写入可移除光盘1322的光盘驱动器1320，诸如CD ROM、DVD ROM或其它光学介质。硬盘驱动器1314、磁盘驱动器1316和光盘驱动器1320分别通过硬盘驱动器接口1324、磁盘驱动器接口1326和光盘驱动器接口1328连接到总线1306。驱动器及其关联的计算机可读介质为计算机提供计算机可读指令、数据结构、程序模块和其它数据的非易失性存储。虽然描述了硬盘、可移除磁盘和可移除光盘，但是可以使用其它类型的基于硬件的计算机可读存储介质来存储数据，诸如闪存卡、数字视频盘、RAM、ROM和其它硬件存储介质。

[0146] 许多程序模块可以存储在硬盘、磁盘、光盘、ROM或RAM上。这些程序包括操作系统1330、一个或多个应用程序1332、其它程序1334和程序数据1336。应用程序1332或其它程序1334可以包括例如用于实现查询优化器104、编译器204、开发应用200、源代码编辑器202、编译器204、调试器工具206、查询检测器402、惰性扩展器404、等效查询集生成器406、可扩展查询检测器902、查询表达式汇编器904、内容确定器1104、规则选择器1106、流程图300、

流程图500、流程图700、流程图800、流程图1000和/或流程图1200(包括流程图300、500、700、800、1000、1200的任何合适步骤),和/或在此描述的其它实施例的计算机程序逻辑(例如,计算机程序代码或指令)。

[0147] 用户可以通过诸如键盘1338和定点装置1340的输入装置将命令和信息输入到计算设备1300中。其它输入装置(未示出)可以包括麦克风、操纵杆、游戏手柄、卫星天线、扫描仪、触摸屏和/或触摸板、用于接收语音输入的语音识别系统、用于接收手势输入的手势识别系统等。这些和其它输入装置通常通过耦合到总线1306的串行端口接口1342连接到处理器电路1302,然而可以通过诸如并行端口、游戏端口或通用串行总线(USB)的其它接口连接。

[0148] 显示屏1344还经由诸如视频适配器1346的接口连接到总线1306。显示屏1344可以在计算设备1300的外部或并入计算设备1300中。显示屏1344可以显示信息,并且可以是用于(例如,通过触摸、手指手势、虚拟键盘等)接收用户命令和/或其它信息的用户接口。除了显示屏1344之外,计算设备1300可以包括其它外围输出装置(未示出),诸如扬声器和打印机。

[0149] 计算设备1300通过适配器或网络接口1350、调制解调器1352或用于通过网络建立通信的其它装置连接到网络1348(例如,因特网)。可以是内部或外部的调制解调器1352可以如图13所示通过串行端口接口1342连接到总线1306,或者可以使用包括并行接口的另一接口类型连接到总线1306。

[0150] 如这里所使用的,术语“计算机程序介质”、“计算机可读介质”和“计算机可读存储介质”被用于指物理硬件介质,诸如与硬盘驱动器1314关联的硬盘、可移除磁盘1318、可移除光盘1322、诸如RAM、ROM、闪存卡、数字视频盘、zip盘、MEM、基于纳米技术的存储装置的其它物理硬件介质,以及其它类型的物理/有形硬件存储介质(包括图13的存储器1320)。这种计算机可读存储介质与通信介质不同且不重叠(不包括通信介质)。通信介质以诸如载波的调制数据信号来体现计算机可读指令、数据结构、程序模块或其它数据。术语“调制数据信号”表示以对信号中的信息进行编码的方式设置或改变其一个或多个特征的信号。通过示例而非限制的方式,通信介质包括诸如声学、RF、红外和其它无线介质的无线介质以及有线介质。实施例还涉及与涉及计算机可读存储介质的实施例分离且不重叠的这种通信介质。

[0151] 如上所述,计算机程序和模块(包括应用程序1332和其它程序1334)可以存储在硬盘、磁盘、光盘、ROM、RAM或其它硬件存储介质上。还可以经由网络接口1350、串行端口接口1342或任何其它接口类型来接收这种计算机程序。当由应用执行或加载时,这种计算机程序使计算设备1300能够实现这里讨论的实施例的特征。因此,这种计算机程序代表计算设备300的控制器。

[0152] 实施例还涉及包括存储在任何计算机可读介质上的计算机代码或指令的计算机程序产品。这种计算机程序产品包括硬盘驱动器、光盘驱动器、存储器装置封装、便携式存储棒、存储卡和其它类型的物理存储硬件。

[0153] IV.示例实施例

[0154] 在实施例中,一种方法包括:检测程序代码中的多个查询;扩展在程序代码中评估查询所使用的惰性;以及将包括多个规则的规则集应用于检测到的查询,以生成功能上等效的查询集,功能上等效的查询集相对于检测到的查询更高效地评估。

[0155] 在实施例中，检测包括：在代码编辑器中检测程序代码中的查询；该方法进一步包括：为代码编辑器呈现选项，以自动重构程序代码，以使用功能上等效的查询集替换多个查询。

[0156] 在实施例中，检测包括：在编译期间检测程序代码中的查询；该方法进一步包括：基于程序代码的版本在编译器中生成编译代码，其中多个查询被功能上等效的查询集替换。

[0157] 在实施例中，扩展在程序代码中评估查询所使用的惰性包括：检测查询中评估可扩展的一个或多个查询；以及形成包括评估可扩展的一个或多个查询的单个查询表达式。

[0158] 在实施例中，应用规则集包括：确定多个查询组件之间的公共逻辑上下文；以及将规则集中的对应于公共逻辑上下文的规则应用于多个查询组件。

[0159] 在实施例中，生成的功能上等效的查询集通过以下中的至少一项来相对于所述多个查询更高效地评估：与多个查询的查询结果相比，生成消耗更少的存储空间的查询结果；与多个查询相比，花费更少的时间来执行；与多个查询相比，消耗更少的网络带宽；或者与多个查询相比，消耗更少的处理功率。

[0160] 在实施例中，应用规则集包括：评估规则集中的多个规则组合到查询组件的应用，以生成多个候选的功能上等效的查询集；以及选择多个候选功能上等效的查询集中的具有最大效率增益的候选的功能上等效的查询集，作为生成的功能上等效的查询集。

[0161] 在实施例中，应用规则集包括：在运行期间，评估规则集中的多个规则组合到查询组件的应用，以生成多个候选的功能上等效的查询集；以及在运行期间，选择多个候选功能上等效的查询集中的具有最大效率增益的候选的功能上等效的查询集，作为生成的功能上等效的查询集。

[0162] 在实施例中，由候选的功能上等效的查询集中一个或多个提供的效率增益至少部分地取决于运行时的执行条件。

[0163] 在实施例中，该方法进一步包括：使用机器学习生成用以添加到规则集的至少一个规则。

[0164] 在另一实施例中，一种计算设备包括：至少一个处理器电路；以及至少一个存储器，存储有被配置为由至少一个处理器电路执行的程序代码，程序代码被配置为执行操作，操作包括：检测程序代码中的多个查询；扩展在程序代码中评估查询所使用的惰性；以及将包括多个规则的规则集应用于检测到的查询，以生成功能上等效的查询集，功能上等效的查询集相对于检测到的查询更高效地评估。

[0165] 在实施例中，检测包括：在代码编辑器中检测程序代码中的查询；该方法进一步包括：为代码编辑器呈现选项，以自动重构程序代码，以使用功能上等效的查询集替换多个查询。

[0166] 在实施例中，检测包括：在编译期间检测程序代码中的查询；该方法进一步包括：基于程序代码的版本在编译器中生成编译代码，其中多个查询被功能上等效的查询集替换。

[0167] 在实施例中，扩展在程序代码中评估查询所使用的惰性包括：检测查询中评估可扩展的一个或多个查询；以及形成包括评估可扩展的一个或多个查询的单个查询表达式。

[0168] 在实施例中，应用规则集包括：确定多个查询组件之间的公共逻辑上下文；以及将

规则集中对应于公共逻辑上下文的规则应用于多个查询组件。

[0169] 在实施例中，应用规则集包括：评估规则集中的多个规则组合到查询组件的应用，以生成多个候选的功能上等效的查询集；以及选择多个候选的功能上等效的查询集中的具有最大效率增益的候选的功能上等效的查询集，作为生成的功能上等效的查询集。

[0170] 在实施例中，应用规则集包括：在运行期间，评估规则集中的多个规则组合到查询组件的应用，以生成多个候选的功能上等效的查询集；以及在运行期间，选择多个候选功能上等效的查询集中的具有最大效率增益的候选的功能上等效的查询集，作为生成的功能上等效的查询集。

[0171] 在实施例中，由一个或多个候选功能上等效的查询集提供的效率增益至少部分地取决于运行处的执行条件。

[0172] 在实施例中，操作进一步包括：使用机器学习生成用以添加到规则集的至少一个规则。

[0173] 在另一实施例中，一种计算设备包括：至少一个处理器电路；以及至少一个存储器，存储有被配置为由至少一个处理器电路执行的程序代码，程序代码包括：查询检测器，被配置为检测程序代码中的多个查询；惰性扩展器，被配置为用于扩展惰性，通过该惰性评估程序代码中的查询；以及等效查询集生成器，被配置为将包括多个规则的规则集应用于检测到的查询，以生成功能上等效的查询集，功能上等效的查询集相对于检测到的查询更高效地评估。

[0174] V.结论

[0175] 尽管上文已经描述了本发明的各个实施例，但是应该理解的是，它们仅以示例而非限制的方式提供。相关领域的技术人员将理解，在不脱离如所附权利要求限定的本发明的精神和范围的情况下，可以在形式和细节上进行各种改变。因此，本发明的广度和范围不应该受任何上述示例性实施例的限制，而应该仅根据所附权利要求及其等效物来限定。

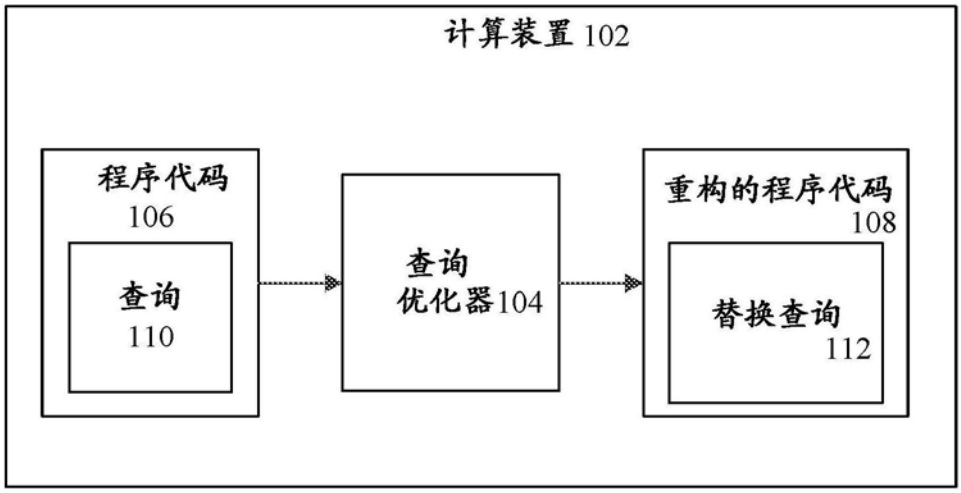

计算装置 102
程序代码
106
查询
110
查询
优化器104
重构的程序代码
108
替换查询
112

图1

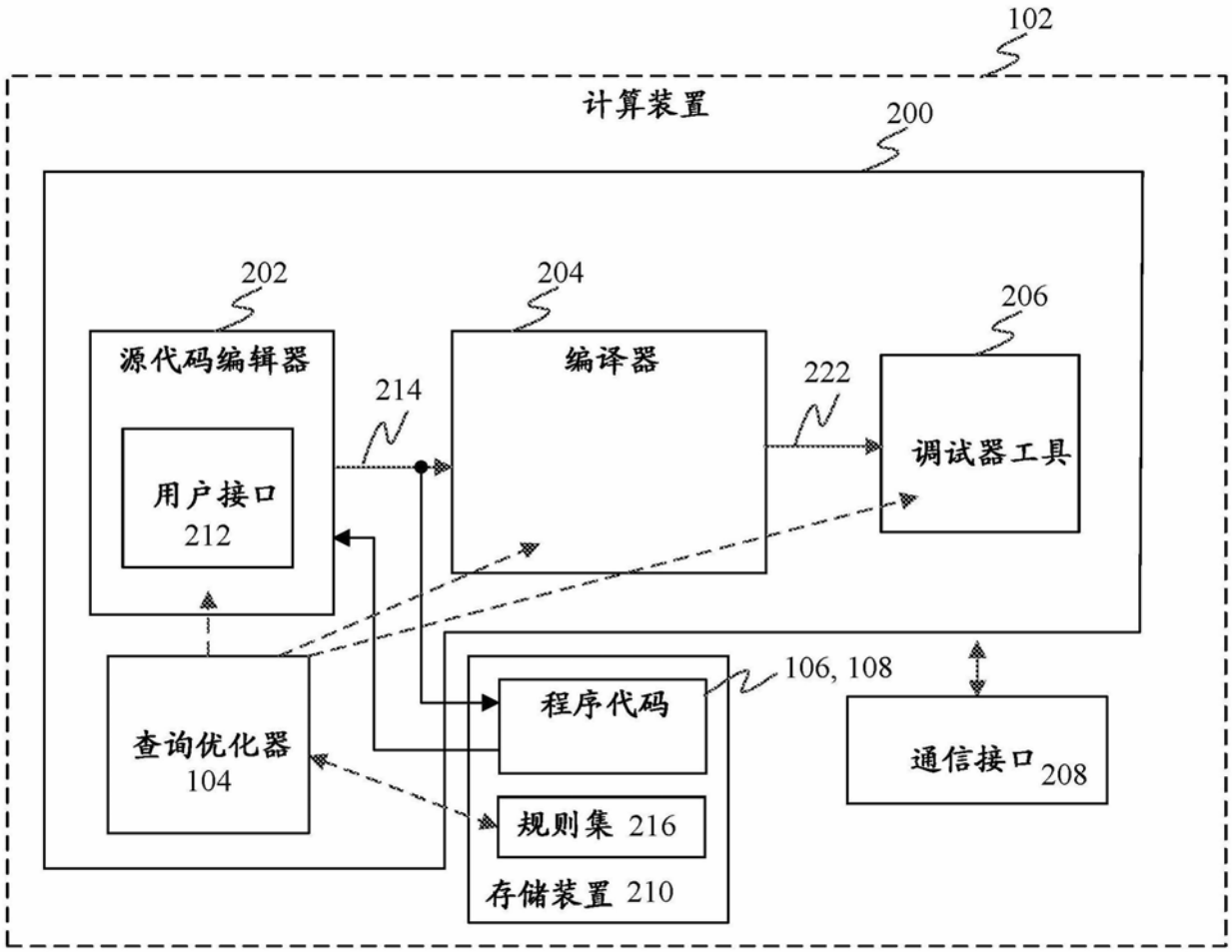

102
计算装置
200
202
源代码编辑器
用户接口
212
214
204
编译器
222
206
调试器工具
查询优化器
104
106, 108
程序代码
规则集 216
存储装置 210
通信接口 208

图2

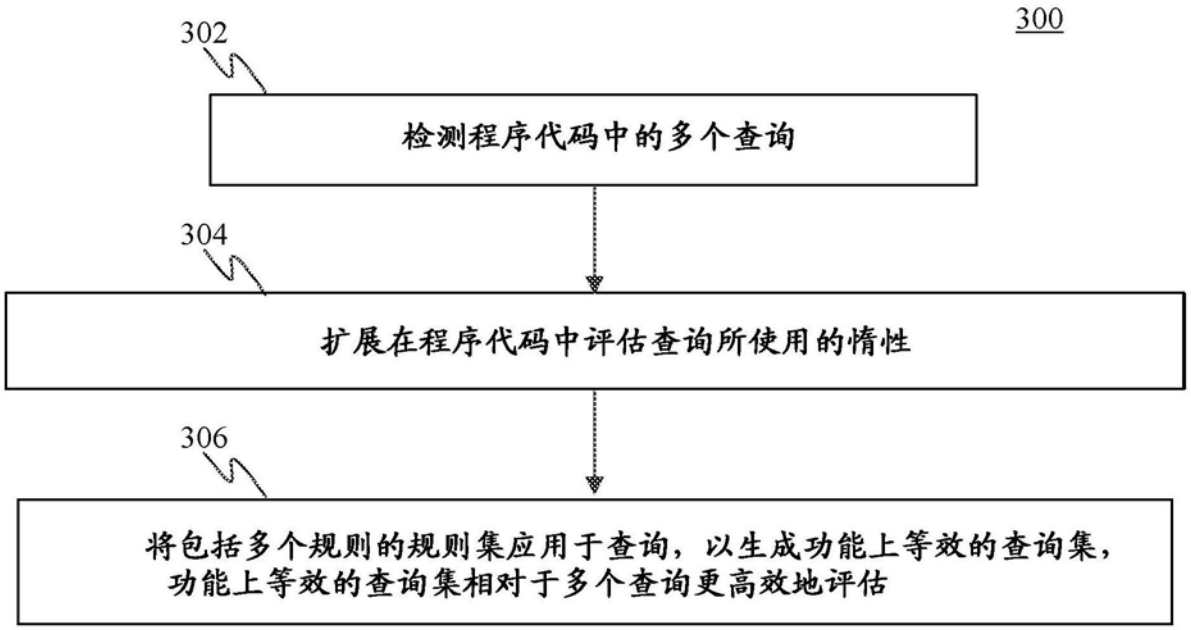

300
302
检测程序代码中的多个查询
304
扩展在程序代码中评估查询所使用的惰性
306
将包括多个规则的规则集应用于查询，以生成功能上等效的查询集，
功能上等效的查询集相对于多个查询更高效地评估

图3


查询优化器 104
查询检测器 402
惰性扩展器 404
等效查询集生成器 406

图4

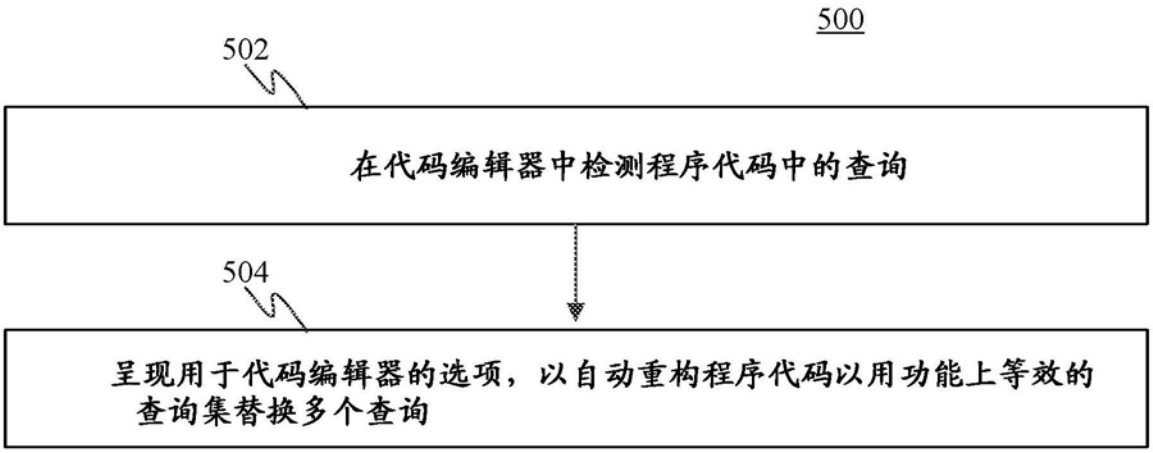

500
502
在代码编辑器中检测程序代码中的查询
504
呈现用于代码编辑器的选项，以自动重构程序代码以用功能上等效的查询集替换多个查询

图5

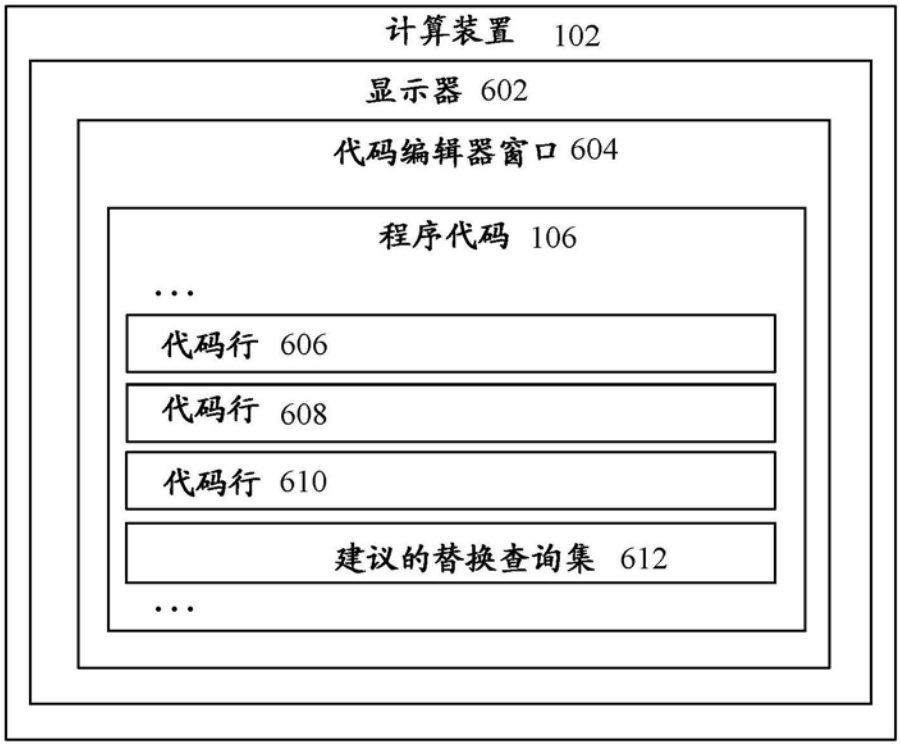

计算装置 102
显示器 602
代码编辑器窗口 604
程序代码 106
...
代码行 606
代码行 608
代码行 610
建议的替换查询集 612
...

图6

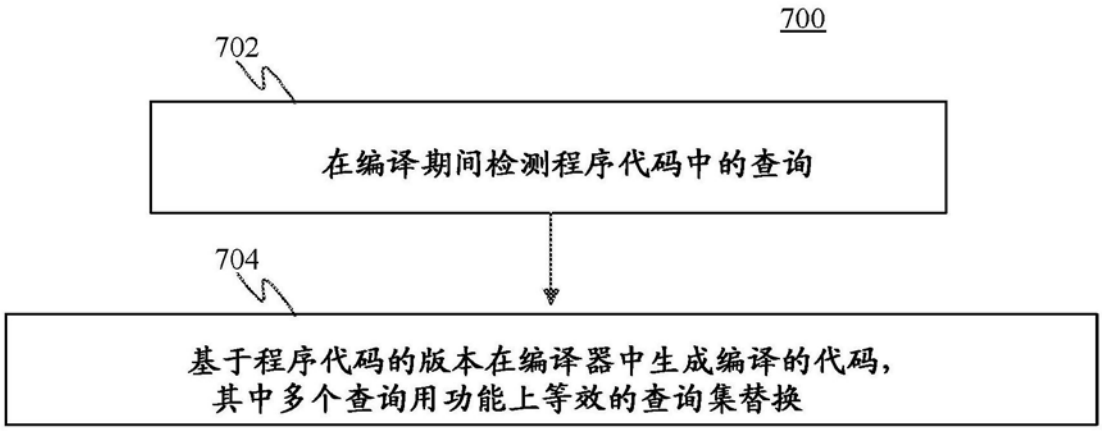

700
702
在编译期间检测程序代码中的查询
704
基于程序代码的版本在编译器中生成编译的代码，
其中多个查询用功能上等效的查询集替换

图7

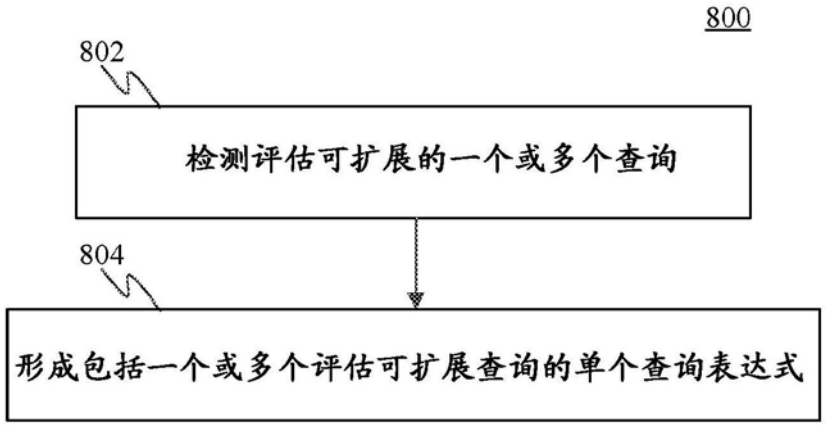

800
802
检测评估可扩展的一个或多个查询
804
形成包括一个或多个评估可扩展查询的单个查询表达式

图8

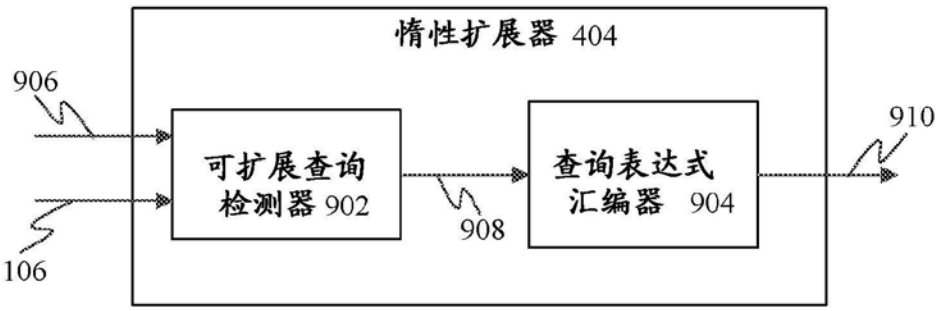

惰性扩展器 404
906
106
可扩展查询
检测器 902
908
查询表达式
汇编器 904
910

图9

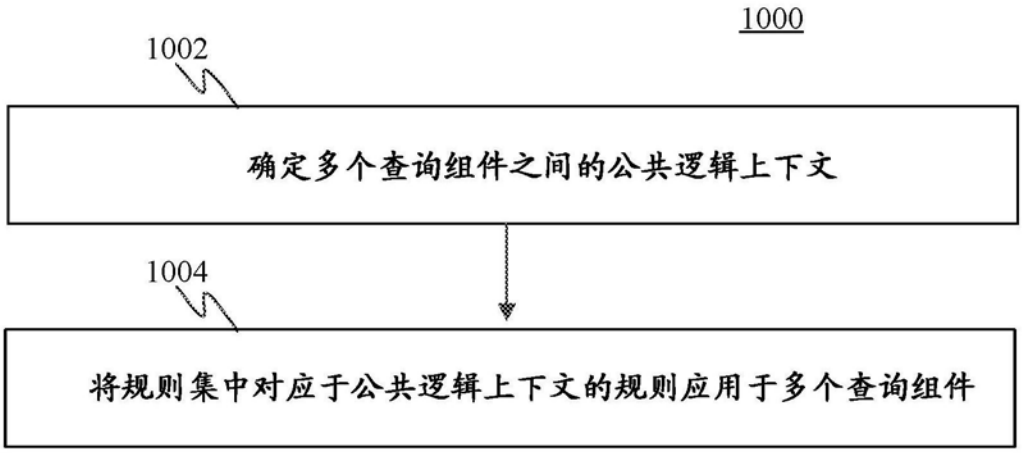

1000
1002
确定多个查询组件之间的公共逻辑上下文
1004
将规则集中对应于公共逻辑上下文的规则应用于多个查询组件

图10

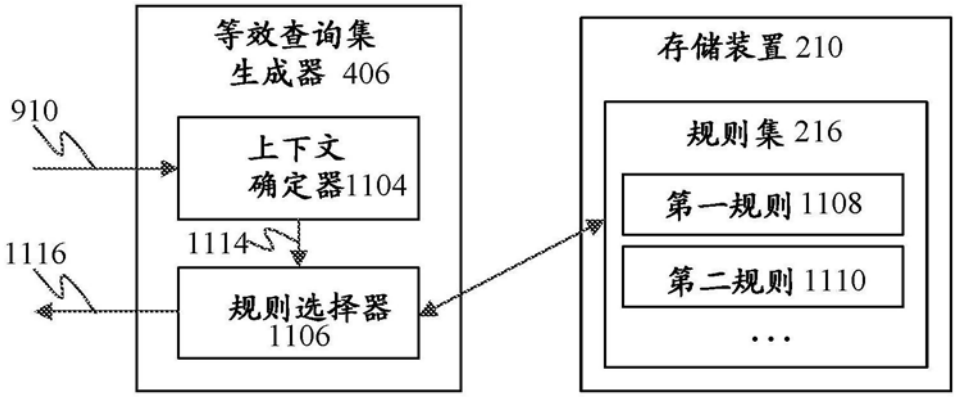

等效查询集
生成器 406
910
上下文
确定器1104
1114
1116
规则选择器
1106
存储装置 210
规则集 216
第一规则 1108
第二规则 1110
...

图11

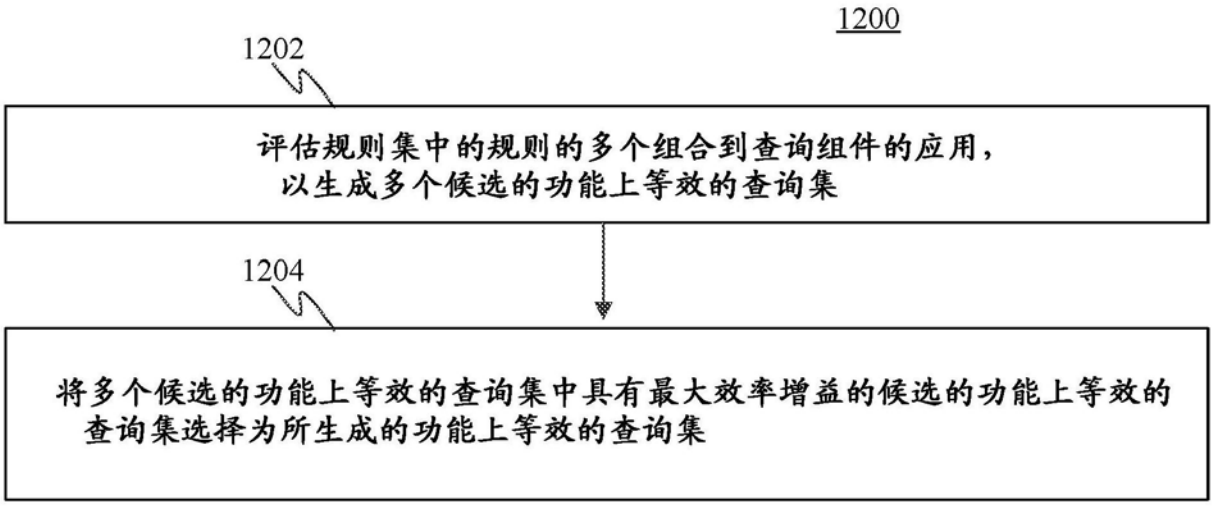

1200
1202
评估规则集中的规则的多个组合到查询组件的应用，
以生成多个候选的功能上等效的查询集
1204
将多个候选的功能上等效的查询集中具有最大效率增益的候选的功能上等效的
查询集选择为所生成的功能上等效的查询集

图12

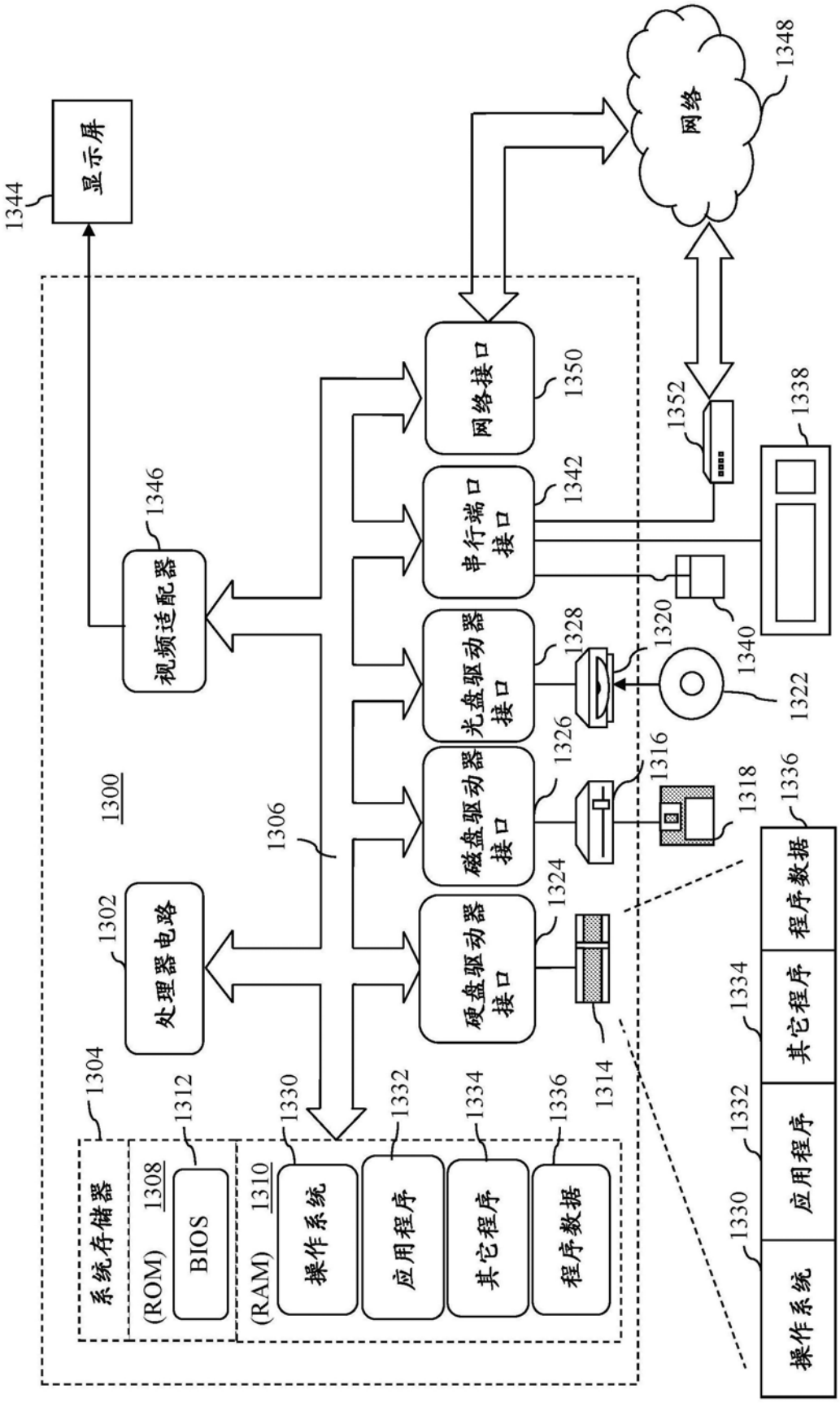

显示屏
1344
网络
1348
1300
1306
视频适配器
1346
网络接口
1350
串行端口接口
1342
光盘驱动器接口
1328
磁盘驱动器接口
1326
硬盘驱动器接口
1324
处理器电路
1302
系统存储器
1304
(ROM) 1308
BIOS
1312
(RAM) 1310
操作系统
1330
应用程序
1332
其它程序
1334
程序数据
1336
1314
1352
1338
1320
1340
1322
1316
1318
1336
程序数据
1334
其它程序
1332
应用程序
1330
操作系统

图13