(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 104866320 B
(45)授权公告日 2018.02.23

(21)申请号 201510307296.6

(22)申请日 2012.08.31

(65)同一申请的已公布的文献号
申请公布号 CN 104866320 A

(43)申请公布日 2015.08.26

(30)优先权数据
13/225,132 2011.09.02 US

(62)分案原申请数据
201210319613.2 2012.08.31

(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 S·卢科 L·拉弗雷尼尔
C·C-C·曼 P·A·莱瑟斯

(74)专利代理机构 上海专利商标事务所有限公司 31100
代理人 杨洁

(51)Int.Cl.
G06F 9/44(2006.01)

(56)对比文件
CN 101263482 A,2008.09.10,
CN 101706753 A,2010.05.12,
US 2004034814 A1,2004.02.19,
US 2009089797 A1,2009.04.02,
CN 1855052 A,2006.11.01,

审查员 郑舒玲

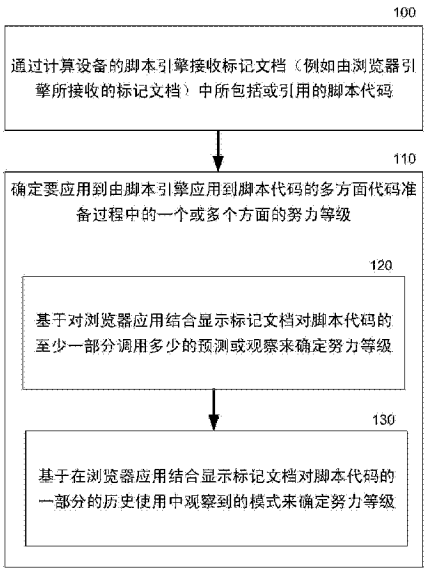
权利要求书1页 说明书14页 附图12页

(54)发明名称

具有脚本代码的标记内容的快速呈现方法

(57)摘要

本发明涉及具有脚本代码的标记内容的快速呈现。提供动态代码生成和协调技术，用于显示包括脚本代码的动态标记文档。一种代码生成过程不仅被代码生成阶段和子阶段的推迟来引导，而且通过解释或对执行的观察来被关于该代码本身的所拥有的各个信息等级所通知，以便不仅生成经修改的代码，而且还为替代情形生成替代代码(例如，生成不同的循环体，此后可取决于浏览器应用所调用的给定函数来容易地换入或换出所述不同的循环体)。通过为用户体验来非对称地确保区分网站呈现和功能的优先级，多核架构进一步改进了用户体验。



CN 104866320 B

1.一种由包括至少两个处理核的计算设备执行的方法,所述方法包括:

通过所述至少两个处理核中的第一处理核来处理(900)包括或引用脚本代码的标记文档,包括:通过所述第一处理核生成可执行代码,所述可执行代码使能由所述标记文档的所述脚本代码所表示的功能;

基于所述脚本代码的性质,通过所述至少两个处理核中的第二处理核,选择性地生成(910)不同于所述可执行代码的替代可执行代码;以及

用所述替代可执行代码取代(920)所述可执行代码,用于所述第一处理核对所述脚本代码的进一步执行。

2.如权利要求1所述的方法,其特征在于,所述生成(910)所述替代可执行代码包括,基于以下的至少一个来选择性地生成所述替代可执行代码:所述脚本代码的至少一个循环的大小与所述脚本代码的总大小的比值的函数、所述脚本代码的计算强度的测量、或对基于所述脚本代码构造的调用树的分析。

3.如权利要求1所述的方法,其特征在于,所述取代(920)包括用所述替代可执行代码更换所述可执行代码而不干扰所述第一处理核对所述可执行代码的当前执行。

4.如权利要求1所述的方法,其特征在于,所述取代(920)包括换入所述替代可执行代码的地址以用于由所述第一处理核的进一步执行。

5.如权利要求1所述的方法,其特征在于,所述生成(910)所述替代可执行代码包括基于分配给所述脚本代码的优先级来生成所述替代可执行代码,分配给所述脚本代码的优先级如在其中表示所述脚本代码的一组优先级工作队列中所表示的。

6.如权利要求1所述的方法,其特征在于,所述通过所述第二处理核生成(910)所述替代可执行代码包括生成为所述可执行代码的动态变量的类型修改的代码,所述类型能够针对适用于所述可执行代码的执行的一组环境而推断或观察出来。

7.如权利要求1所述的方法,其特征在于,所述方法还包括:

由所述至少两个处理核中的第三处理核,回收(930),用于与所述标记文档相关的、不再用于所述标记文档的显示的对象的存储器。

具有脚本代码的标记内容的快速呈现方法

[0001] 本申请是申请日为2012年8月31日，申请号为201210319613.2，名为“具有脚本代码的标记内容的快速呈现”的申请的分案申请。

技术领域

[0002] 本发明涉及通过用于减少与基于脚本代码动态生成可执行代码相关联的代码准备的各阶段(stage)的传统延迟的各种技术，快速呈现具有脚本代码的标记内容。

背景技术

[0003] 随着web浏览体验从用最小的交互性平面地呈现信息向在客户端侧具有大量交互性的更丰富的应用(application)或小程序(applet)体验(或一般地信息显示画面和与显示画面上的对象的更丰富的交互性的混合)不断地发展，基于原本主要为基于客户端上的本机代码的平面信息呈现的旧文档对象模型(DOM)向流动地处理脚本代码(诸如JavaScript对象)的体验的发展存在各种挑战。例如，加速用户体验仍然是一项挑战。

[0004] 例如，使用过去的飞出菜单，web体验基于与服务器的通信对延迟进行闪烁。脚本使得小的程序能够在进行中(on the fly)修改DOM而无需返回到服务器来获得额外的帮助，例如异步JavaScript和可扩展标记语言(Ajax)。由于开发者和消费者可能想要在不返回到服务器的情况下在进行中做更多事情，使脚本代码在客户端侧上快速执行已成为挑战。

[0005] 因为用户体验集中于使DOM脚本化(scripting)，所以缓慢地改变DOM带来不好的交互响应。在过去，由于对包括使任何对象可脚本化的一组接口(iDispatch、iActiveScript等)的OLE自动化的使用，脚本引擎和DOM的本机类之间的通信是不良的。此外，如今各种组件对象模型(COM)对象可被无区别地创建，但这可能是不必要的，减慢了性能。

[0006] 通过避免与服务器的一次或多次往返通信，客户端侧脚本化能够使网页对一旦在客户端浏览器上做出的用户输入更具响应性，客户端侧脚本化一般是指web上在客户端侧通过用户的web浏览器执行而不是在服务器侧(在web服务器上)执行的那类计算机程序。作为示例，客户端侧脚本化是动态超文本标记语言(动态HTML或DHTML)的一部分，使得网页能够被按照脚本化语言(诸如JavaScript(客户端侧JavaScript)和VBScript)脚本化，以取决于用户输入、环境条件(诸如在一天中的时间)或其他变量而具有不同且不断变化的内容。

[0007] 嵌入标记文档内的客户端侧脚本有时被称为“嵌入式脚本”，而包含在分开的文件(这些文件被使用该文件的文档所引用)中的脚本有时被称为“外部脚本”。响应于请求，这些脚本文件被这些文件所在的网络计算设备(诸如web服务器)发送到用户的计算机。关于此，为了执行脚本并显示包括来自脚本的任何可见输出的标记文档，web浏览器与脚本引擎一起工作以解析并编译脚本代码。客户端侧脚本还可包含浏览器响应于某些用户动作(例如，点击按钮)要遵循的指令。通常，这些指令可被遵循而无需与服务器进一步通信。

[0008] 作为一个趋势，web浏览器和网页趋向于采用越来越多的客户端侧脚本化，这对其

中用户没有体验到不友善的网页“刷新”的改进的用户界面具有贡献，但是反而看见，例如，表示动画图形交换格式(GIF)文件的图标以及网页的该部分将在很短时间内被更新，其中该图标表示发生对GIF文件的请求。除了JavaScript能力之外，Ajax还允许客户端机器和脚本化的文档的开发者在后台与网络计算设备(诸如web服务器)通信而不需要请求并呈现新的网页版本。

[0009] 尽管一般会带来用户体验的改进，但是这导致了在设置时间在客户端侧上花太久的其他等待时间。例如，在大量脚本化在网站中被采用的情况下(前100个web特性通常是这样)，或者在网络数据传输速率很慢的情况下，或在客户端上的处理另外地受限的情况下，脚本引擎可能花太久来交付所有的可执行代码以用于标记文档的内容的呈现，导致不佳的用户体验。

[0010] 关于此，传统web浏览已按照导致脚本引擎低效地创建可执行代码的方式发展，因为脚本引擎关于脚本代码创建没有区分，导致在脚本引擎完成其工作以呈现脚本化的标记文档时的不想要的等待时间。这种不想要的等待时间可能明显拖用户体验的后腿，特别是在用户可能不需要或想要在开始交互之前等待所有网站功能加载的情况下。

[0011] 如今的脚本引擎和代码生成的上述缺点仅仅旨在提供对常规系统的一些问题的总览，并且不旨在是穷尽性的。在仔细阅读了以下详细描述后，现有技术的其他问题和各非限制性性实施例的对应好处可变得显而易见。

发明内容

[0012] 此处提供了简化的发明内容以帮助能够对以下更详细的描述和附图中的示例性、非限制性实施例的各方面有基本或大体的理解。然而，本发明内容并不旨在是详尽的或穷尽的。相反，本发明的唯一目的在于，以简化的形式提出与一些示例性、非限制性实施例相关的一些概念，作为以下各实施例的更详细的描述的序言。

[0013] 在一示例实施例中，一种方法包括通过计算设备的脚本引擎接收脚本代码，其中所述计算设备的浏览器应用所接收的标记文档包括或引用所述脚本代码，以及确定要应用到由所述脚本引擎应用到所述脚本代码的多方面代码准备过程的至少一个方面(phase)的努力等级。该努力等级可通过以下来确定：基于由该浏览器应用对该脚本代码的一部分的调用的预测或分析，或基于在由该浏览器应用对该脚本代码的该部分的历史使用中观察到的模式。

[0014] 在另一示例实施例中，计算设备包括浏览器应用和脚本引擎组件，浏览器应用被配置成显示包括或引用脚本代码的标记文档，脚本引擎组件被配置成接收该脚本代码、分析该脚本代码的一部分或该脚本代码的执行期间该部分的使用来确定一组条件，在该组条件下该部分的性能通过执行经修改的部分而增加，其中，响应于该组条件被满足，该经修改的部分而不是该部分由该浏览器应用执行。

[0015] 在另一示例实施例中，一种计算机可读存储介质包括计算机可读指令，该计算机可读指令响应于执行致使包括至少两个处理核的计算设备执行操作，所述操作包括通过所述至少两个处理核中的第一处理核处理包括或引用脚本代码的标记文档，包括由该第一处理核生成可执行代码，该可执行代码使能由该标记文档的该脚本代码所表示的功能。该操作还包括：基于该脚本代码的性质，通过该至少两个处理核中的第二处理核选择性地生成

不同于该可执行代码的替代可执行代码，并用该替代可执行代码取代该可执行代码以用于该第一处理核对该脚本代码的进一步执行。

[0016] 生成该替代可执行代码可包括基于以下来选择性地生成该替代可执行代码：该脚本代码的至少一个循环的大小与该脚本代码的总大小的比值的函数、该脚本代码的计算强度的测量、或对基于该脚本代码构造的调用树的分析。

[0017] 以下更详细地描述其他实施例和各非限制性性示例、场景和实现。

[0018] 附图描述

[0019] 参考附图进一步描述各非限制性实施例，在附图中：

[0020] 图1示出流程图，该流程图示出用于确定应用到多方面代码准备过程的一个或多个阶段的努力等级的示例方法。

[0021] 图2示出根据各实施例在代码准备期间代码可能经历的各个示例阶段。

[0022] 图3是根据各实施例可能存在的不同努力等级可被应用到的给定方面的多个子等级的框图；

[0023] 图4是示出在一实施例中基于不同执行条件生成代码的替代循环体的框图；

[0024] 图5示出在一实施例中基于动态键入在动态代码生成和执行期间的代码交换(code swapping)的另一非限制性示例；

[0025] 图6是在一实施例中在浏览器应用和脚本引擎之间的示例交换的流程图；

[0026] 图7是示出在一个实施例中在运行解释代码的线程和生成优化代码和相应数据结构集合的线程之间的平衡的框图；

[0027] 图8是示出根据一实施例的为代码优化而重新区分工作项的优先级的框图；

[0028] 图9是示出在一实施例中用于生成用于取代的优化代码的非限制性过程的流程图；

[0029] 图10是示出根据各实施例的包括脚本引擎和浏览器应用之间的交互的示例性的、非限制性的设备的框图；

[0030] 图11是表示其中可实现在此处所述的各个实施例的示例性、非限制性联网环境的框图；以及

[0031] 图12是表示其中可实现此处所述的各个实施例的一个或多个方面的示例性、非限制性计算系统或操作环境的框图。

具体实施方式

[0032] 概览

[0033] 如在背景技术中所指示的，围绕在标记文档(诸如DHTML文档)中包括越来越多的嵌入的或外部的脚本化的代码(诸如JavaScript)的趋势可能导致某些不佳的用户体验，其中因为传统脚本引擎所应用的传统动态代码生成过程而发生过度的延迟。

[0034] 关于此，动态代码生成由于它的某些性质而具有某些固有的挑战。例如，在动态键入的语言(诸如JavaScript)的情况下，通常直到代码实际执行才有可能分辨数据的各项呈现什么类型，例如，可能直到代码实际执行才知道动态键入的变量是整数、浮点数、还是串。

[0035] 此外，如何托管(host)脚本代码的更广的上下文带来其他的差异，因为对于具有成为网页的一部分的目标的程序(诸如CNN.com的程序)来说，在同时运行各程序的同时对

程序的动态编译以使用本机代码指令的要求可能变得很显著。例如，前100个网站的一个性质在于：它们加载比它们实际运行的多得多的代码。关于此，挑战不仅是尝试快速运行代码。因为某些网站最终仅为给定用户或使用运行它们加载的代码的约15％-20％，所以在仅仅是快速执行代码之外存在另一个挑战，这是由于在该过程中存在其他瓶颈，诸如加载、解析等。这是与仅快速运行Java不同的上下文，因为涉及到最小化加载和解析以及减少为不被使用或不太可能被使用的代码所完成的代码准备的量的不同目标。

[0036] 术语“热点(hot spot)”是用于函数的行话(vernacular term)，它多被用在正在运行的程序中。然而，在此处各实施例中此处所解决的问题更广泛，因为它们还涉及避免代码准备穿过从接收标记页面到呈现该网页并执行所请求的功能的整个一连串的事，例如，避免或推迟加载、避免或推迟解析、避免或推迟编译、避免或推迟对正在执行的代码的优化等。本发明从而涉及剪裁(tailor)呈现内容中所引用或接收的脚本代码(诸如但不限于响应于网页请求从网络接收的标记内容中所引用的脚本代码)的准备，以允许以减小的与准备相关联的延迟来向用户显示该呈现内容。

[0037] 相应地，在本申请中出现的各实施例中，减少了代码准备，例如，以准备尽可能少的代码以尽可能快地呈现与标记文档相关联的UI、以及尽可能多地推迟加载、解析、解释等为目标，例如，推迟干扰迅速的UI呈现的任何项的各代码准备阶段或子阶段，直到该代码被实际请求或满足某些条件。

[0038] 此外，通过在解释或执行期间对代码的观察，如果可为该代码确定常见执行路径情形，则在下面详细呈现的各实施例中，可维护该本机代码的两个版本，一个版本是为常见情形优化或另外改进的，而另一个版本仅用于不常见的情形。可将这些思想扩展到超过两个版本的代码和相关联使用情形，其中维持超过两个使用情形的效率比切换成本更重要。

[0039] 在下面更详细地呈现的又一实施例中，通过在一个核或一组核上维持用于呈现信息的UI线程，而在另一个核或另一组核上执行或应用对代码准备过程的上述优化和管理，可以按照响应于信息被准备好而区分信息的显示的优先级的方式来使用多个核。关于此，可以维护数据结构，诸如工作队列，用于在生成经修改的本机代码的优化和管理核以及尽可能快地运行网站向用户的显示的UI核之间共享信息。而且，在各实施例中，向UI核提供替代代码段可用于为给定情况运行的通知，以便该替代代码段可被切换入(switch in)以供UI核执行，而不受UI核当前正在执行的代码的干扰。

[0040] 以下更详细地描述其他实施例和各非限制性性示例、场景和实现。

[0041] 用于内容的快速呈现的脚本代码的推迟的或优化的代码准备

[0042] 如同提到过的，在此处呈现的各实施例中，减少或推迟了代码准备，例如，以准备尽可能少的代码以尽可能快地呈现与标记文档相关联的UI、以及尽可能多地推迟加载、解析、解释、优化等为非限制性目标，例如，推迟干扰迅速的UI呈现的任何项的各代码准备阶段或子阶段，直到稍后或被实际请求。

[0043] 图1示出了示出用于确定应用到多方面代码准备过程的一个或多个方面的努力等级的示例方法的流程图，例如，没有努力、仅部分解析、解析、解释、监视代码的执行以优化该代码等。在100，通过计算设备的脚本引擎接收脚本代码，该计算设备的浏览器应用所接收的标记文档中包括或引用该脚本代码。在110，确定要应用到由脚本引擎应用到脚本代码的多方面代码准备过程中的至少一个方面的努力等级。努力等级的确定可包括：在120，基

于对浏览器应用结合显示标记文档对脚本代码的至少一部分调用多少的预测或观察来确定努力等级，或者在130，基于在浏览器应用结合显示标记文档对脚本代码的一部分的历史使用中观察到的模式来确定努力等级，或执行120和130两者。

[0044] 例如，如图2所示，作为接收代码之后的初始阶段，可选择地可应用较少的努力，而不是完全解析代码。例如，可不执行解析，或可通过仅确定函数的开头和结尾(例如，通过括号匹配)来执行部分解析。例如，通过括号匹配，脚本引擎可接收代码200并确定在何处N个函数Method1，Method2，Method3，Method4，…，MethodN的边界是被部分解析的代码202，直到找到浏览器应用所调用的函数。例如，如果代码202的Method4实际上被调用，则在已经仅确定了Method1、Method2和Method3的开头和结尾的情况下，脚本引擎可开始解析并解释Method4，导致经解析和解释的Method4 204。解析和解释因此是代码准备阶段的示例，其中可确定并应用这种努力等级以用更多资源对待更高优先级的代码，并用更少资源或不用资源对待具有更低优先级的代码，如努力等级的特定确定可指示的。

[0045] 在解释期间，脚本引擎还可观察Method4的性质，以改善为执行Method4而生成的可执行代码，例如，通过改善Method4的存储器(memory)使用，或以其他方式使Method4更高效。例如，脚本引擎可确定在Method4中存在运行很多的循环，并向该循环应用更多优化资源，或脚本引擎可确定一段代码中某个变量的类型可预测地或确定性地为整数，在这种情况下可使用更高效的表示。关于此，尽管动态程序可具有在运行时(run-time)采用不同类型的变量，编程者为了设计或表示的简单仍然趋向于开发具有模式(pattern)的代码，其中例如编程者将一贯地使用一变量来存储整数，或者反而是可预测地或确定性地为串的另一变量。关于此，在解释期间，脚本引擎可观察这样的模式，并随后简化用于Method4的可执行代码206的生成/编译的代码准备。

[0046] 作为又一个附加操作，如下面在其他实施例中更详细地描述的，也可在代码执行期间观察可执行代码206的行为，在这种情况下，进一步的观察可针对该代码在典型情形下运行的方式来做出，从而允许以经修改的可执行代码208的形式对可执行代码206做出进一步的修改，对于未来的执行或未来的执行中的某一些，随后可用经修改的可执行代码208代替可执行代码206。相应地，在各实施例中，在代码准备的多个方面的任何方面(例如，解析、解释、编译、执行)中、或任何方面的子方面中，因为这些活动中的每一个均可被分为可向其应用一努力等级的各种代码准备步骤，可确定努力等级，并且如果该努力等级不需要即时的资源，可推迟相关联的代码准备方面或子方面，有效地向浏览器应用所实际请求的代码准备、或有可能被浏览器应用很多地实际请求或在不远的未来实际请求的代码准备分配即时的资源。

[0047] 相应地，在一个方面中，贯穿整个代码准备并在代码准备的各阶段和子阶段的任何可能的地方，推迟代码准备。这不仅是推迟从脚本代码到本机处理器代码的转换，而是更广泛地适用，因为本文描述的各实施例基于为给定代码段所应用的相关联的努力等级来推迟解析、解释、字节码生成等，即，代码准备的任何阶段或子阶段。

[0048] 如所提到的，目前描述的实施例比术语“热点”的行话使用更广泛，该行话使用观察热点函数(例如，运行很多的函数)并对准备运行时的热函数投入比冷函数更多的努力。这些实施例可考虑准备代码的任何数量N个方面，并且可基于该代码的所预测的或观察到的使用来为每个方面确定努力等级。此外，给定方面中可能存在可向其应用不同努力等级

的多个子等级。

[0049] 这在图3的框图中示出，图3示出了代码准备的三个阶段300、310和320，分别具有子阶段集合302、304和306，312、314和316，和322、324和326。关于此，例如基于所预测的代码使用或对实际代码使用的观察，可向不同阶段和/或子阶段分配不同的努力等级。例如，代码优化阶段(诸如阶段300、310或320)的子阶段等级处的示例优化技术可包括循环不变代码运动分析、常见子表达消除、常数传播等。

[0050] 尽管图3中示出了具有三个子阶段的三个阶段，注意，对于替代实施例考虑任何数量的阶段和子阶段，并且根据任何分层关系，或根据可向该代码准备过程的给定分支或节点分配努力等级的任何其他随意的或可选的代码准备过程。例如，作为另一非限制性示例，脚本引擎可决定运行“热”代码上的总共10个方面中的3个方面，仅运行“冷”代码上的10个方面中的一个方面，但是运行最热代码的子集上的全部10个方面。换言之，对于代码准备过程的任何数量的方面，可基于正在运行的程序正多么多地使用代码以及关于该代码的信息和其使用，在发射本机指令的最有效序列处将努力等级作为时间的函数来调整或调节。

[0051] 相应地，除了推迟代码准备方面或子方面外，此处的各实施例描述了当资源变得可用或致力于优化给定代码或函数集合时，基于关于该代码的执行的预测性的或实际的观察，随时间生成替代代码。作为一个示例，高度使用网站(诸如纽约时报的在线站点)的平均使用体验可以是约5-10分钟，但是从用户的立场看只花费时间的一小部分来使代码引擎观察纽约时报的核心函数中的大部分，例如，如何布局页面、取类型信息、取数据等。关于此，不存在数百种布局新页面的方式，并且因此脚本引擎可相对快地、完全地或大部分地观察读取纽约时报上的文章的常见代码通过(pass)，并随后开始为阅读相同的或其他的文章生成替代代码。

[0052] 图4是示出基于不同执行条件生成代码的替代循环体的框图。例如，取决于执行条件420，循环体410、循环体412或循环体414可以最快或另外地更高效地应用到当前条件。相应地，具有循环体410的可执行代码400、具有循环体412的可执行代码402、具有循环体414的可执行代码404每个可被预先生成，或者循环体410、412或414可被换入(swap in)或换出(swap out)呈现线程所定位的(address)的现有代码以分别执行用于适当条件C1、C2或C3的替代代码。从而，对于任何预定义的集合条件A、B、C等(这些条件单独地或组合地带来条件C1、C2或C3)，不同循环体可作为改善代码的执行效率的努力的结果而被换入。在本示例中，跨越各替代来循环体执行，然而，当前描述的实施例适用于优化、或换入或换出不同于循环体的代码或代码部分的其他方式。

[0053] 作为生成替代代码的进一步非限制性示例，如果可以确定，则作为代码的预测使用或代码的实际观察执行的结果，超过95％的时间，将为或正为给定数据项存储一整数，则可生成优化的本机代码，该优化的本机代码使用本机整数寄存器格式以与x86处理器一起直接使用，并因此，对于这种大多数情况下发生的情形，可使用优化的本机代码。并且随后，当不常见的情形在小于5％的时间里发生时，可用本机代码的另一版本将该变量作为浮点数(或比本机整数寄存器格式低效的东西)键入，对于特殊情况可将其换入。

[0054] 图5示出在基于动态键入在动态代码生成和执行期间的代码交换的另一非限制性示例。关于此，例如通过观察交互期间代码500的性质或作为观察执行期间代码500的结果，在可确定存在常见代码路径或路径集合的情况下，则可生成适合于常见代码路径的本机代

码502或其他替代，而随后当事实上遇到较不常见的情形时或如果遇到较不常见的情形则可为该较不常见的情形换入代码504。下面更详细地描述了关于换入为应用到本机代码的不同使用情形而优化过的不同本机代码版本的这些方面。例如，在下面更详细地描述的其他实施例中，可向是否向一段代码应用及时(just in time)编译器并随后执行“经及时化(jitted)”的代码还是在该段代码的一组给定情况下简单地解释该段代码的决策应用试探(heuristic)。

[0055] 关于此，图4和图5示出了此处描述的各实施例的另一方面，在于代码生成过程不仅由代码准备阶段或子阶段的推迟引导，而且由关于代码本身的所拥有的各个信息等级通知(通过解释或对执行的观察)，以不仅在替代代码能在所有情况下简单地取代代码的情况下生成替代代码，而且还为替代情形生成替代代码(例如，生成不同的循环体，所述循环体此后能取决于浏览器应用的给定函数调用而被容易地换入或换出)。

[0056] 相应地，如上所述，代码的行为可被推断或观察，导致用于不同情形的不同代码的生成，例如，常见情形对比不常见情形，并为常见情形执行替代并从而改善整体性能而没有最小公分母的情况的约束。从而，用于代码生成的智能选择可由关于该代码的信息来通知，例如，该代码可预测地或确定性地表现为将变量定义为整数数组，在这种情况下可生成在循环的顶部执行快速检查的代码，该快速检查确认它是整数数组，并且假定如此，则使用为整数数据构造剪裁的更高效的循环。

[0057] 图6是在浏览器应用630和脚本引擎640之间的示例交换的流程图。浏览器应用630所接收的脚本代码660不带关于脚本代码660的重要性的任何优先级信息地被发送到脚本引擎640。在600，脚本引擎640分析脚本代码660的循环体或该循环体的使用。在610，可生成经修改的循环体以在某些条件下使用，例如，为脚本代码660的常见使用情形优化过的代码。在620，响应于该组条件被满足而用经修改的循环体来取代该循环体，或替代地，可在该循环体的顶部进行检查以确认是否满足该常见使用情况，并作为结果执行该循环体或该经修改的循环体。相应的可执行代码670被返回到浏览器应用630以供执行。

[0058] 关于此，参考后面的非限制性实施例，进一步描述了将上述技术应用于指导脚本引擎的有限代码生成资源，其目标是快速地向用户呈现信息而无需等待较低优先级的代码被解析、生成、优化等。

[0059] 起先，浏览器应用接收具有样式表、JavaScript文件等的标记文件(诸如HTML或DHTML)。浏览器应用继续以将标记文件的不同部分分开，并随后用并发连接下载标记中所定义的JavaScript或其他文件，并发连接中的每个连接用于各个不同的段。当接收到各个不同的段之后，浏览器应用将JavaScript文件转发到JavaScript引擎，但是不带关于任何给定段的优先级或重要性的任何信息。关于时机，还可命令脚本引擎准备好运行给定代码段。然而，运行给定代码块的请求可能在浏览器应用已接收到该代码块的仅1ms内发生，而如果花30-40ms来生成相应代码，并且花另外的5ms在要显示的UI路径中等的话，则这些延迟可加总并潜在地影响用户体验。相应地，如同上面指出的，如果可向代码准备阶段或子阶段的推迟或避免应用智能，则是有利的。例如，在人类视觉和认知欣赏方面，100ms或更多的延迟变成可察觉的延迟。

[0060] 相应地，作为另一示例，对于给定代码段，脚本引擎可反而花1-2ms来查看函数边界在何处，但是还没有开始任何解析，而脚本引擎可将“稍后解析我”存根(stub)插入该代

码并仅在1-2ms的通过之后将其传回。然后，稍后，如果浏览器应用想要调用例如该代码段的函数F，则可花费另外的1-2ms来解析或运行该代码段的函数F，同时仍旧避免解析整个代码段。

[0061] 根据上面描述的其他实施例，可使该示例再进一步，由此脚本引擎可在为该代码段的函数F解析并生成代码的同时，观察变量V表现为就像它将可预测地或确定性地为整数，或在运行F若干次之后，可以确认V可预测地或确定性地为整数，则脚本引擎可生成专用于将V表示为整数的代码。关于此，脚本引擎尽力尽可能晚地响应于所需要的，同时推测性地观察至今发生了什么，进而直到朝创建在常见情形下发生的更快且更高效的代码演变。

[0062] 在本发明的其他方面中，使用多个核在一方面对代码生成的上述管理和优化，另一方面快速向用户呈现UI而不等待该管理和优化完成之间实现平衡。相应地，在各实施例中，优化和管理在与主UI线程不同的线程上进行(主UI线程对诸如鼠标点击等输入做出响应并立即运行相关联的代码，例如，响应于输入而相应地改变视图)，但是其他处理核可并行地基于上述智能代码生成实施例中的任何一个来生成替代代码。

[0063] 关于此，在各实施例中，在驱动标记文档的呈现的UI核和优化代码生成的优化/管理核之间应用平衡过程。在一个实施例中，该优化/管理核管理数据结构以用于在生成替代本机代码的优化/管理核和向用户呈现该网站的UI核之间共享信息。因为UI核心关注快速呈现并且从而通常是面向用户的，在一个实施例中，UI核和优化/管理核包括共享机制，以使得UI核在给定代码段的替代或优化版本变得可用时运行该替代或优化版本，而不等待，即，将替代代码换入显示该网页的活动而不减慢该活动。关于此，如在下面的各实施例中所描述的，可提供避免与在执行代码期间用替代代码盖写该代码相关联的任何问题的机制。

[0064] 如在一个实施例中所描述的，对平衡过程的任何非对称设计确保优化/管理核承担维护工作队列、换入和换出的协调、代码在后台的生成的任务，以使得UI核永远不会等待并且能够继续进行它的向用户显示信息的任务。

[0065] 在一个非限制性实现红，通过调用入或检查被称为function_info(函数_信息)的函数(该函数对每个函数均实现)，UI线程可检查是否存在要调用的本地进入点，并且如果存在，则该UI线程调用该本机进入点。本机进入点可以是在代码生成的各个方面处对一段代码的存根，例如，本机进入点可以是“已生成的代码”存根或“解析我”存根(该存根导致该代码的解析)或者“调用解释器来运行该代码”存根。关于此，UI线程对存根的内容是不可知的，而仅是继续并且根据存根调用来运行该时刻在那里的代码。当可产生替代代码时，则当UI线程下一次传递该代码时，替代代码将已利用存根检查取代了较旧的代码，即，下一次通过将致使UI线程经由指向新地址的“已生成的代码”存根跳到替代代码。

[0066] 例如，作为朝已生成的代码逐步移动的代码的一部分，在一个实施例中，在初始通过中，不重要的代码不被解析，但是“解析我”存根被插入从而在下一个通过上或当UI线程接下来遇到该代码时该代码被解析。然后在下一次调用该代码时，生成字节码，然后调用解释器来运行该字节码，等等。UI核还可用无锁队列来放弃(drop off)一工作项，这是一种排齐工作项而不花费锁的方式。关于此，下面更详细地描述如何组织这些队列、关于你何时对代码做何事的优先级、你花多少努力来解析、编译等以及为何如此的试探。相应地，大量工作在后台进行而用户并不知晓，这些工作正识别常见网页和应用场景、并将后台核(优化/管理核)的资源定向到它们最有效的地方。

[0067] 图7示出一个示例实现，其中由第一核或第一组核处理的UI线程750以及由第二核或另一组核处理的JIT线程760针对何时运行被解释器解释的经解释的代码770、或何时“及时化(jit)”该代码(使该代码通过JIT编译器)并随后运行经及时化的代码780来实现上面描述的平衡。

[0068] 作为使用存根的示例，诸如形实转换程序(thunk)，如上面提到的，在此情形中，初始时，每个形实转换程序可被赋予空(NULL)值，并且如果该值为空，UI线程700可继续使用经解释的代码770，然而如果该字段不是空(即，如果存在指向经及时化的代码的地址)，则UI线程700将转而使用经及时化的代码780。同时，优化器/管理器核继续尝试在资源和时间允许时及时化越来越多的代码以便最终所有工作都被完成，但是同时，用户尽可能快地体验了该网页，即使并非所有代码均已被加载、解析、编译、及时化等。

[0069] 非限制性地、更详细地描述此示例，存在两种在这种环境中运行代码的方式：代码可运行通过解释器(一般更快)，或该代码可被“及时化”(运行通过被称为及时化器(jitter)的及时编译器，一般稍慢)。一般而言，用解释器解释线性代码序列比将代码及时化、等待及时化器完成并随后运行经及时化的代码更快。关于此，及时化器趋向于对该代码采用多次通过以形成优化的代码。相应地，如果看上去代码将被使用很多，则尽管有额外的代码准备时间，使用及时化器通常仍是性能改进，但是如果代码的使用将很低，例如，使用一次，则使用解释器来快速运行代码通常是性能改进。从而，在各实施例中，在经及时化的代码和经解释的代码之间维持平衡。

[0070] 初始时，在UI线程上，解释器开始解释代码，而在JIT线程上，标识用于及时化的候选。例如，对及时化表现良好的函数是大部分代码在循环中的小函数。从而，根据一轮推测性及时化，这些函数首先被及时化，其中基于一组试探选择排名靠前的候选。通过推测性及时化，不仅所调用的函数被及时化，而且某些函数即便没被调用但由于其将被调用的可能性也被及时化。随后，一旦这些函数被及时化，则进入点被从经解释的代码770切换到经及时化的代码780。

[0071] 关于此，由后台(优化器/管理器)核维护多个具有不同优先级的队列。其示例在图7中示出，其中具有工作负载槽700、702、……、704的最高优先级序列被维护，具有工作负载槽710、712、……、714的第二级优先级序列，具有工作负载槽720、722、……、724的第三级优先级序列，……，以及具有工作负载槽730、734、……、736的第N级优先级序列。例如，在一个实施例中，从高优先级到低优先级选择7个不同的优先级队列。在初始通过期间，在接收到代码时根据所接收的代码的性质将工作分槽(slot)，然而，在初始通过期间，推测性的及时化可操作以在工作满足某些推测性标准(如下面描述的)时将其指定在较低的优先级队列上。在推测性及时化后，例如，工作项714、722和730可能是用于及时化的好的候选，并且从而对工作负载槽的顶级候选700、702、……、704以及对在推测过程期间所选择的那些候选执行及时化。

[0072] 在一轮推测性及时化后，进一步的及时化开始于最高优先级的工作项，而所述工作项中剩余的在优先级方面被向上渗透，如下。关于此，如在图8中对第三优先级代码中的代码工作项724所示，每次运行该代码时，每次经解释的代码770被调用时将其在优先级上向上移动一级。从而，在初始通过之后，每次使用经解释的代码770时，其优先级向上移动，直到它达到最高优先级。在一个实现中，当第二优先级代码达到最高优先级时(如由工作项

710所示)，它实际上移动到所有其他工作前面，并且及时化器在下一次机会时开始它对该代码的通过。

[0073] 作为结果，如果代码被使用得很多，但是具有低初始优先级并且被该推测性的轮次略过，则它将向上渗透并很快变得优化。如果该代码不被调用得很多，则它将不被尽快优化。一旦所有最高优先级项目被及时化，及时化器可开始优化第二优先级项目，如此等等，直到所有工作项目被消耗，而同时UI线程运行最优代码而不减慢。

[0074] 关于如何区分脚本项的代码生成的优先级的试探，一种试探是该代码是多么循环化(loopy)，其是除了是循环的一切东西以外的代码大小的测量，其是总代码大小与它里面有多少是循环的比值，或反过来，循环与代码大小的比值。从而，如果一个函数大部分是循环，则与具有小循环而大部分代码不是循环的函数相比它具有较高的优化优先级。

[0075] 此外，与不进行繁重数学计算的代码相比，正进行繁重的数学计算的代码从被及时化中获得更多的益处。优化对执行大量数学的代码工作得很好，因为可利用处理器的本机寄存器，本机寄存器与在其他存储器中设置分开的存储器结构的代码相比非常快。

[0076] 在另一非限制性实施例中，构造正在运行的脚本的调用树，这有助于通知优先级次序。关于此，调用树有助于预计什么函数将是热的。例如，如果注意到第一段代码将可预测地或确定性地在第二段代码之前运行，则使第一段代码比第二段代码具有更高的优先级是有意义的。或者，例如，如果存在没有if语句的全局函数，则在其具有更高优先级的情况下执行该代码存在很高的机率。作为包含函数Foo的一点代码的假想示例：

[0077] Foo

[0078] {

[0079] Bar;

[0080] If X,then Y(如果X,则Y)

[0081] }

[0082] 在此情况下，显然，如果Foo被调用则Bar将被调用，而因为X可能不是真的，则Y可能不被调用，并且从而Bar比Y具有更高的优化优先级。有许多不同的方式来采用这些试探作为输入并形成相对的优先级。一种方式是取决于给定代码段满足所选择的一组试探的程度来分配从0到100的优先级值，并随后大致相等地将其分为那个数量的队列。

[0083] 如所提到的，一旦经及时化的可执行代码变得可用，经及时化的可执行代码的地址被插入到解释器代码形实转换程序(thunk)上方。如果解释器代码形实转换程序仍旧在附近，则进行检查以确认是否已进行了或正在进行代码生成，而如果不是，则该工作项在优先级方面向上猛冲。如果它已经被及时化，则用经及时化的地址取代该形实转换程序。换言之，形实转换程序存在以帮助标识使用经解释的代码的情况，直到经及时化的代码变得可用，而随后形实转换程序消失/被新的存储器位置取代，以使得该代码操作以跳到不同的存储器位置以运行经及时化的代码，例如，使UI线程前进或跳跃到新的代码。在一个实施例中，在将工作项移动到最高优先级队列前方时可采用短期锁(short lock)以避免冲突，但是因为这仅是项在列表中的重新定位，所以它是非常快且最小的锁定时间。

[0084] 如上面描述的，使用2个核，1个用于解释器而1个用于及时化器，然而，在一附加实施例中，可能存在3个或更多核：1个用于解释器、1个用于快速及时化器(较不全面)、而1个用于缓慢及时化器(全面的)。以此方式，对于会受益于某些优化但不是完整优化的某些项

目，这些项目可作为中间步骤被改进。

[0085] 此外，可使用补充核结合网站的显示来执行不再被任何其他对象使用或引用的所有对象的所有垃圾收集，以不干扰其他核正在进行的重要工作。

[0086] 图9是示出在一实施例中用于生成用于取代的优化代码的非限制性过程的流程图；在900，利用第一处理核，显示包括或引用脚本代码的标记文档，并且生成使能该标记文档的脚本代码所表示的功能的可执行代码。在910，利用第二处理核，基于脚本代码的性质，选择性地生成不同于该可执行代码的替代可执行代码。在920，利用第二处理核，用该替代可执行代码取代该可执行代码用于第一处理核对该脚本代码的进一步执行。可选地，在930，第三处理核可执行垃圾收集过程，该过程回收用于与标记文档有关的、不再用于标记文档或其功能的对象的存储器。

[0087] 图10是示出根据各实施例的包括脚本引擎1040和浏览器应用1030之间的交互的示例性的、非限制性的设备1000的框图。如图所示，浏览器应用1030接收包括脚本代码1032的标记文档1034，浏览器应用1030随后将脚本代码1060转发到脚本引擎1040以进行如在此处的一个或多个实施例中所述的智能代码生成。关于此，维持各中间数据结构1050，通过中间数据结构1050，向给定脚本代码段1060的代码生成的各阶段的优化分配优先级。这些结构可被维持在存储器1010中，在适当情况下被高速缓存在高速缓存存储器1015中。如所提到的，在各实施例中，多个处理核1020正根据如在上面的一个或多个实施例中描述的分工来执行并用分开的核使用户体验优先。在根据此处描述的各实施例的各优化或完成等级处，脚本引擎1040最终将可执行代码1070传递回浏览器应用。同时，主核一能完成其任务就尽快地处理标记文档1044的呈现而不等待脚本引擎1040完成其对所有工作项的工作，而可执行代码1070的输出被表示为标记文档1044的输出1042。

[0088] 在各实施例中，启用智能编译技术和试探，所述智能编译技术和试探利用分开的核来加速客户端对某些网页“热点”(例如，具有高交互性的区域)的代码生成。为web定制试探，例如，可使用数据结构的web高速缓存来指导代码生成选择(例如，通过在高速缓存中存储关于使用给定网页的上一次时间的信息)，并且随后作为结果，可使用于所标识的热点的代码更快。以此方式，随着时间，因为相同网站趋向于被调用很多，所以网站解释参考保持在高速缓存中的已构造结构随着时间改进。

[0089] 某些性能结果已显示，上面描述的各实施例和技术有助于相对于不应用此处描述的各实施例的先前的网页加载和交互的40到150倍的速度改进。

[0090] 示例性联网以及分布式环境

[0091] 本领域技术人员可以理解，此处描述的用于动态代码生成的各实施例可结合任何计算机或其它客户机或服务器设备来实现，其可被部署为计算机网络的部分或在分布式计算环境中，并且可以被连接到任何类型的数据存储。在这一点上，此处描述的各实施例可在具有任何数量的存储器或存储单元的、并且任何数量的应用和进程跨任何数量的存储单元发生的任何计算机系统或环境中实现。这包括但不限于具有部署在具有远程或本地存储的网络环境或分布式计算环境中的服务器计算机和客户机计算机的环境。

[0092] 分布式计算通过计算设备和系统之间的通信交换提供了计算机资源和服务的共享。这些资源和服务包括信息的交换、对于诸如文件等对象的高速缓存存储和盘存储。这些资源和服务还包括多个处理单元之间的处理能力共享以便进行负载平衡、资源扩展、处理

专门化，等等。分布式计算利用网络连接，从而允许客户机利用它们的集体力量来使整个企业受益。就此，各种设备可具有可如参考本发明的各实施例描述地参与用于动态代码生成的机制的应用、对象或资源。

[0093] 图11提供了示例性的联网或分布式计算环境的示意图。该分布式计算环境包括计算对象1110、1112等以及计算对象或设备1120、1122、1124、1126、1128等，这些计算对象或设备可包括如由应用1130、1132、1134、1136、1138和数据存储1140表示的程序、方法、数据存储、可编程逻辑等。可以理解，计算对象1110、1112等以及计算对象或设备1120、1122、1124、1126、1128等可包括不同的设备，诸如个人数字助理(PDA)、音频/视频设备、移动电话、MP3播放器、个人计算机、膝上型计算机等。

[0094] 每一个计算对象1110、1112等以及计算对象或设备1120、1122、1124、1126、1128等可通过通信网络1142直接或间接与一个或多个其他计算对象1110、1112等以及计算对象或设备1120、1122、1124、1126、1128等进行通信。即使在图11中被示为单个元件，但通信网络1142可包括向图11的系统提供服务的其他计算对象或计算设备，和/或可表示多个互连网络(未示出)。每个计算对象1110、1112等或计算对象或设备1120、1122、1124、1126、1128等还可以含有应用，诸如可以利用API或其他对象、软件、固件和/或硬件的、适于实现或与根据各实施例所提供用于动态代码生成的技术进行通信的应用1130、1132、1134、1136、1138。

[0095] 存在支持分布式计算环境的各种系统、组件和网络配置。例如，计算系统可由有线或无线系统、本地网络或广泛分布的网络连接在一起。当前，许多网络被耦合至因特网，后者为广泛分布的计算提供了基础结构并包含许多不同的网络，但任何网络基础结构都可用于便于与如各实施例中所描述的用于动态代码生成的系统的示例性通信。

[0096] 由此，可使用诸如客户机/服务器、对等、或混合体系结构之类的网络拓扑结构和网络基础结构的主机。“客户端”是使用与它无关的另一类或组的服务的一个类或组中的成员。客户端可以是进程，即大致上是请求由另一程序或进程提供的服务的一组指令或任务。客户端进程利用所请求的服务，而不必“知道”有关其他程序或服务本身的任何工作细节。

[0097] 在客户端/服务器体系结构中，尤其在联网系统中，客户端通常是访问另一计算机(例如，服务器)所提供的共享网络资源的计算机。在图11的图示中，作为非限制性示例，计算对象或设备1120、1122、1124、1126、1128等可被认为是客户机而计算对象1110、1112等可被认为是服务器，其中计算对象1110、1112等担当提供数据服务的服务器，诸如从客户机计算对象或设备1120、1122、1124、1126、1128等接收数据、存储数据、处理数据、向客户机计算对象或设备1120、1122、1124、1126、1128发送数据等，但任何计算机都可取决于环境而被认为是客户机、服务器或两者。

[0098] 服务器通常是可通过诸如因特网或无线网络基础结构之类的远程网络或本地网络访问的远程计算机系统。客户机进程可在第一计算机系统中活动，而服务器进程可在第二计算机系统中活动，它们通过通信介质相互通信，由此提供分布式功能并允许多个客户机利用服务器的信息收集能力。按照此处所描述的技术来利用的任何软件对象可以被单独提供或分布在多个计算设备或对象上。

[0099] 在其中通信网络1142或总线例如是因特网的网络环境中，计算对象1110、1112等可以是其他计算对象或设备1120、1122、1124、1126、1128等通过诸如超文本传输协议(HTTP)等多种已知协议中的任一种与其通信的web服务器。担当服务器的计算对象1110、

1112等也可用作客户端，例如计算对象或设备1120、1122、1124、1126、1128等，这是分布式计算环境的特性。

[0100] 示例性计算设备

[0101] 如上所述，有利的是，此处所描述的技术可适用于期望在计算系统中执行动态代码生成的任何设备。因此，可以理解，构想了结合各实施例使用的所有种类的手持式、便携式和其它计算设备和计算对象，即，在设备的资源使用可理想地优化的任何地方。因此，以下在图12中所述的通用远程计算机只是计算设备的一个示例。

[0102] 尽管并非所需，但各实施例可部分地经由操作系统来实现，以供设备或对象的服务开发者使用和/或被包括在用于执行此处所述的各实施例的一个或多个功能方面的应用软件内。软件可以在由诸如客户端工作站、服务器或其它设备等一个或多个计算机执行的诸如程序模块等计算机可执行指令的通用上下文中描述。本领域的技术人员可以理解，计算机系统具有可用于传递数据的各种配置和协议，并且由此没有特定配置或协议应当被认为是限制性的。

[0103] 因此，图12示出了其中可实现各实施例的一个或多个方面的合适的计算系统环境1200的一个示例，尽管如上所述，计算系统环境1200仅为合适的计算环境的一个示例，并非对使用范围或功能提出任何限制。也不应当将计算系统环境1200解释为对在示例性计算系统环境1200中所示的组件中的任何一个或其组合有任何依赖或要求。

[0104] 参考图12，用于实现一个或多个实施例的示例性远程设备包括计算机1210形式的通用计算设备。计算机1210的组件可包括，但不限于，处理单元1220、系统存储器1230、以及将包括系统存储器的各种系统组件耦合到处理单元1220的系统总线1222。

[0105] 计算机1210通常包括各种计算机可读介质，并且可以是可由计算机1210访问的任何可用介质。系统存储器1230可包括诸如只读存储器(ROM)和/或随机存取存储器(RAM)之类的易失性和/或非易失性存储器形式的计算机存储介质。作为示例而非限制，系统存储器1230还可包括操作系统、应用程序、其他程序模块、以及程序数据。根据进一步的示例，计算机1210还可以包括各种其它介质(未示出)，可以包括，但不是限制，RAM，、ROM、EEPROM、闪存存储器或其它存储器技术，CD ROM、数字多功能盘(DVD)或其它光盘存储、磁带盒、磁带、磁盘存储或其它磁存储设备或其它可用于存储所需信息的有形的和/或非瞬时介质。

[0106] 用户可通过输入设备1240向计算机1210输入命令和信息。监视器或其他类型的显示设备也经由诸如输出接口1250之类的接口连接到系统总线1222。除监视器以外，计算机还可包括诸如扬声器和打印机之类的其他外围输出设备，它们可通过输出接口1250连接。

[0107] 计算机1210可使用到一个或多个其他远程计算机(诸如远程计算机1270)的诸如网络接口1260的逻辑连接在联网或分布式环境中操作。远程计算机1270可以是个人计算机、服务器、路由器、网络PC、对等设备或其他常见网络节点、或者任何其他远程媒体消费或传输设备，并且可包括以上关于计算机1210所述的任何或全部元件。图12所示的逻辑连接包括诸如局域网(LAN)或广域网(WAN)之类的网络1272，但也可包括其他网络/总线。这些联网环境在家庭、办公室、企业范围的计算机网络、内联网和因特网中是常见的。

[0108] 如上所述，尽管结合各种计算设备和网络体系结构描述了各示例性实施例，但底层概念可被应用于任何网络系统和任何计算设备或系统。

[0109] 此外，存在实现相同或相似功能的多种方法，例如适当的API、工具箱、驱动程序代

码、操作系统、控件、独立或可下载软件对象等，它们使得应用和服务能够使用此处提供的技术。由此，此处的各实施例从API(或其他软件对象)的观点以及从实现如此处描述的一个或多个实施例的软件或硬件对象构想。由此，此处所述的各实施例可具有完全采用硬件、部分采用硬件并且部分采用软件、以及采用软件的方面。

[0110] 本文中所使用的词语“示例性”意味着用作示例、实例、或说明。为避免疑惑，本文所公开的主题不限于这些示例。另外，在此所述的被描述为“示例性”的任意方面或设计并不一定要被解释为相比其它方面或设计更优选或有利。此外，在使用术语“包括”、“具有”、“包含”和其他类似词语的程度上，为避免疑惑，这些术语旨在以类似于术语“包括”作为开放的过渡词的方式是包含性的而不排除任何附加或其他元素。

[0111] 如所述的，此处所述的各种技术可结合硬件或软件或，在适当时，以两者的组合来实现。如此处所使用的，术语“组件”、“系统”等同样旨在指计算机相关实体，或者是硬件、硬件和软件的组合、软件或者是执行中的软件。例如，组件可以是，但不限于是，在处理器上运行的进程、处理器、对象、可执行码、执行的线程、程序和/或计算机。作为说明，在计算机上运行的应用和计算机都可以是组件。一个或多个组件可以驻留在进程和/或执行线程中，并且组件可以位于一个计算机内和/或分布在两个或更多计算机之间。

[0112] 如前所述的系统已经参考若干组件之间的交互来描述。可以理解，这些系统和组件可包括组件或指定的子组件、某些指定的组件或子组件和/或附加的组件，并且根据上述内容的各种置换和组合。子组件还可作为通信地耦合到其他组件的组件来实现，而不是被包括在父组件内(层次性)。另外，应注意到一个或多个组件可被组合成提供聚集功能的单个组件，或被分成若干单独的子组件，且诸如管理层等任何一个或多个中间层可被设置成通信耦合到这样的子组件以便提供集成功能。此处所述的任何组件也可与一个或多个此处未专门描述的但本领域技术人员一般已知的其他组件进行交互。

[0113] 鉴于以上所述的示例性系统，参考各附图的流程图还可理解根据所述的主题实现的方法。尽管为了说明简洁起见，作为一系列框示出和描述的方法，但是应当理解，各实施例不仅仅限于框的次序，因为一些框可以与此处所描绘和描述的框不同的次序发生和/或与其他框并发地发生。尽管经由流程图示出了非顺序或分支的流程，但可以理解，可实现达到相同或类似结果的各种其他分支、流程路径和框的次序。此外，并非全部所示的框都是实现下面所述的方法所必需的。

[0114] 除了此处所描述的各实施例之外，可以理解，可以使用其他相似的实施例或者可对所述实施例作出修改和添加以便执行对应的实施例的相同或等效的功能而不背离这些实施例。此外，多个处理芯片或多个设备可共享此处所述的一个或多个功能的性能，并且类似地，存储可跨多个设备实现。因此，本发明不应限于任何单个实施例，而是应当根据所附权利要求书的广度、精神和范围来解释。

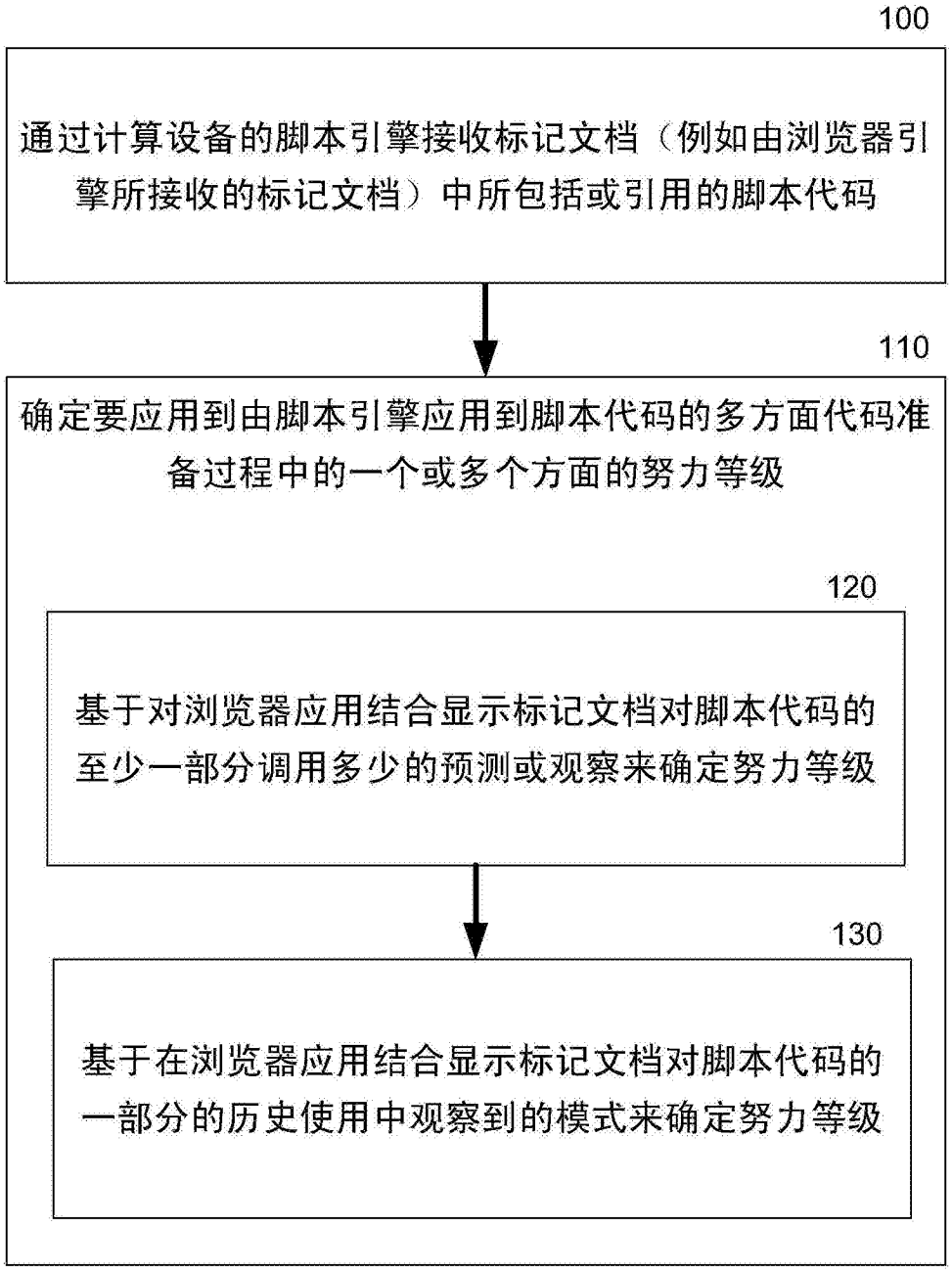

100
通过计算设备的脚本引擎接收标记文档（例如由浏览器引擎所接收的标记文档）中所包括或引用的脚本代码
110
确定要应用到由脚本引擎应用到脚本代码的多方面代码准备过程中的一个或多个方面的努力等级
120
基于对浏览器应用结合显示标记文档对脚本代码的至少一部分调用多少的预测或观察来确定努力等级
130
基于在浏览器应用结合显示标记文档对脚本代码的一部分的历史使用中观察到的模式来确定努力等级

图1

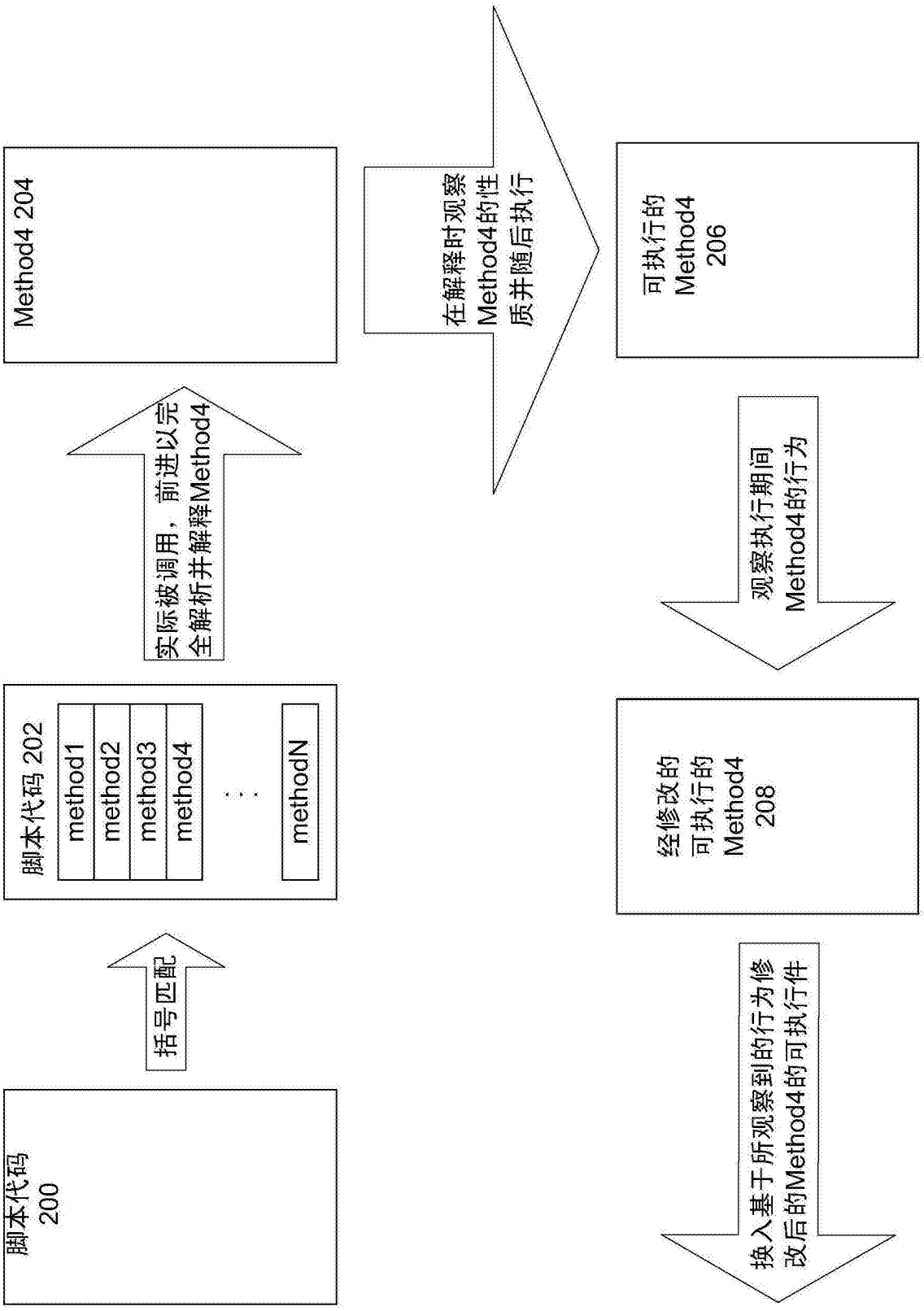

脚本代码
200
括号匹配
脚本代码 202
method1
method2
method3
method4
methodN
实际被调用，前进以完
全解析并解释Method4
Method4 204
在解释时观察
Method4的性
质并随后执行
可执行的
Method4
206
观察执行期间
Method4的行为
经修改的
可执行的
Method4
208
换入基于所观察到的行为修
改后的Method4的可执行件

图2

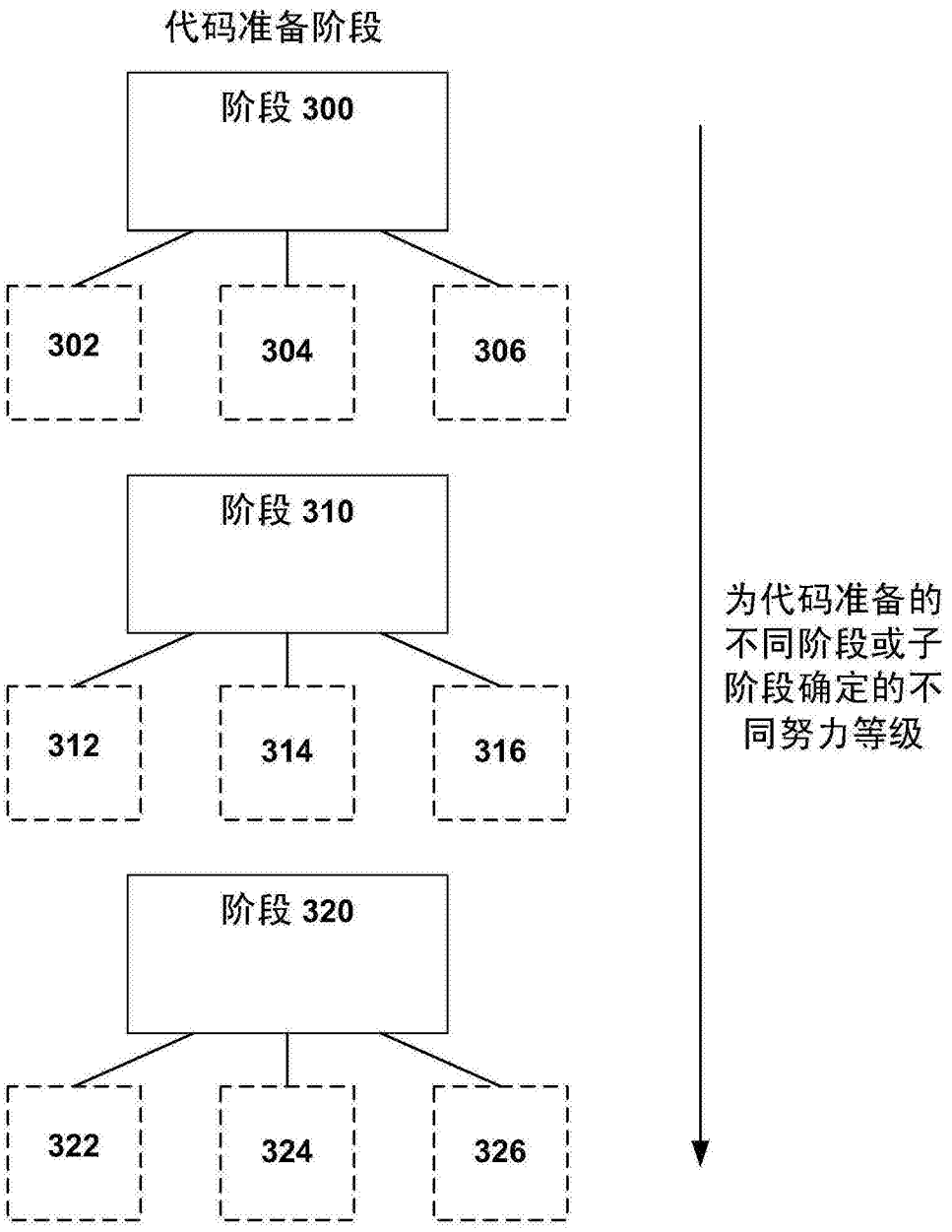

代码准备阶段
阶段 300
302
304
306
阶段 310
312
314
316
阶段 320
322
324
326
为代码准备的不同阶段或子阶段确定的不同努力等级

图3

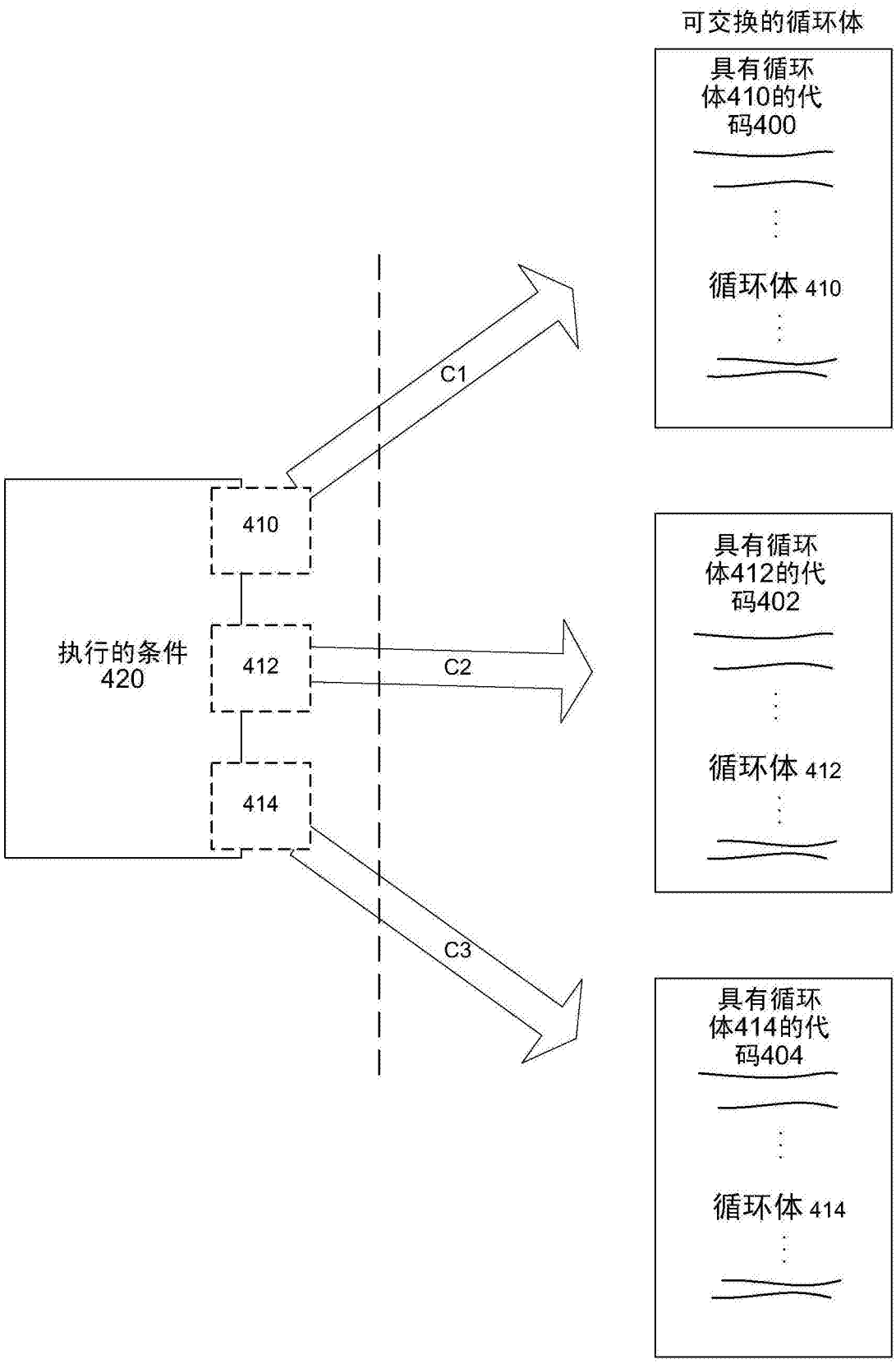

可交换的循环体
具有循环体410的代码400
循环体 410
具有循环体412的代码402
循环体 412
具有循环体414的代码404
循环体 414
C1
C2
C3
410
412
414
执行的条件420

图4

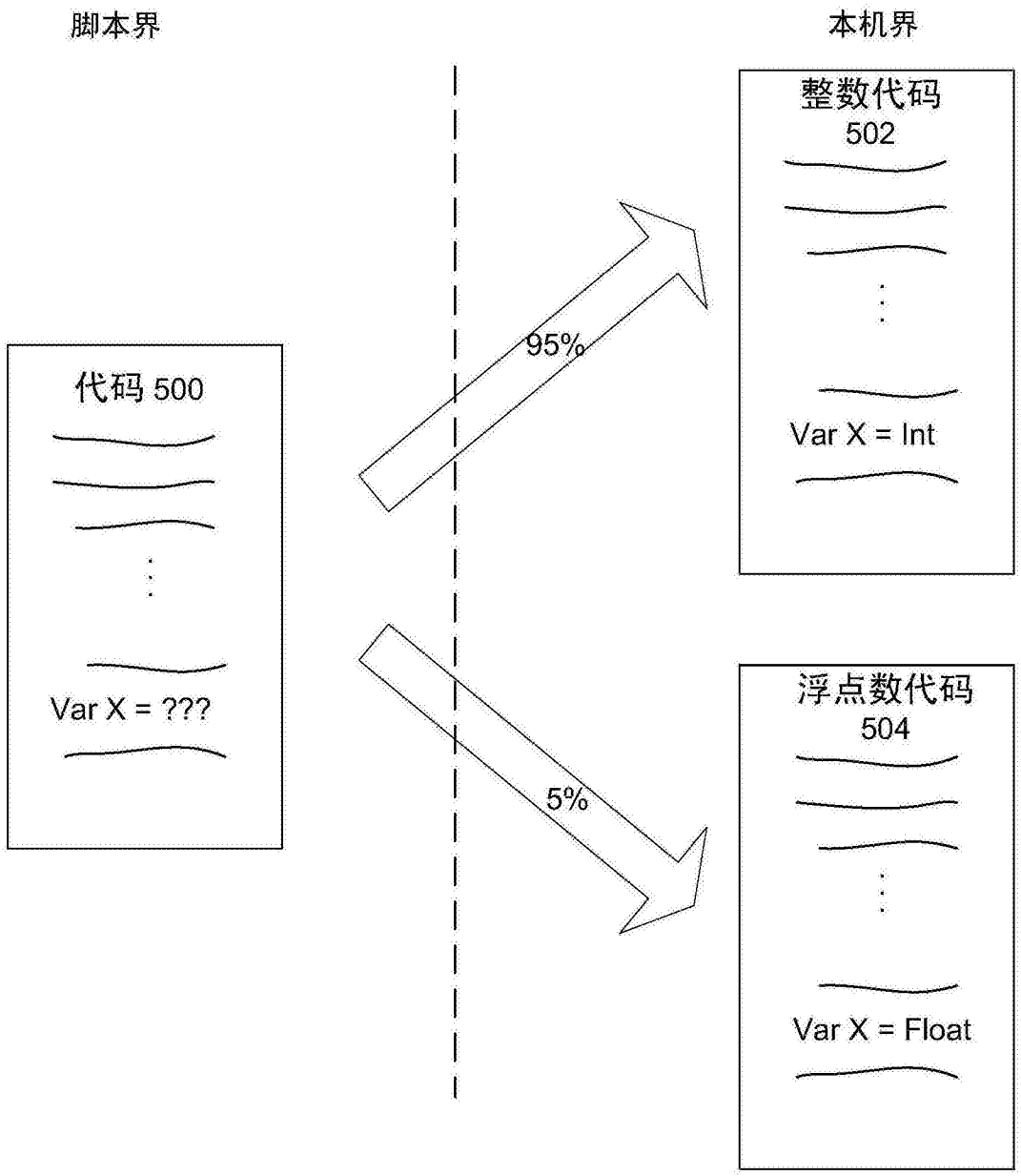


图5

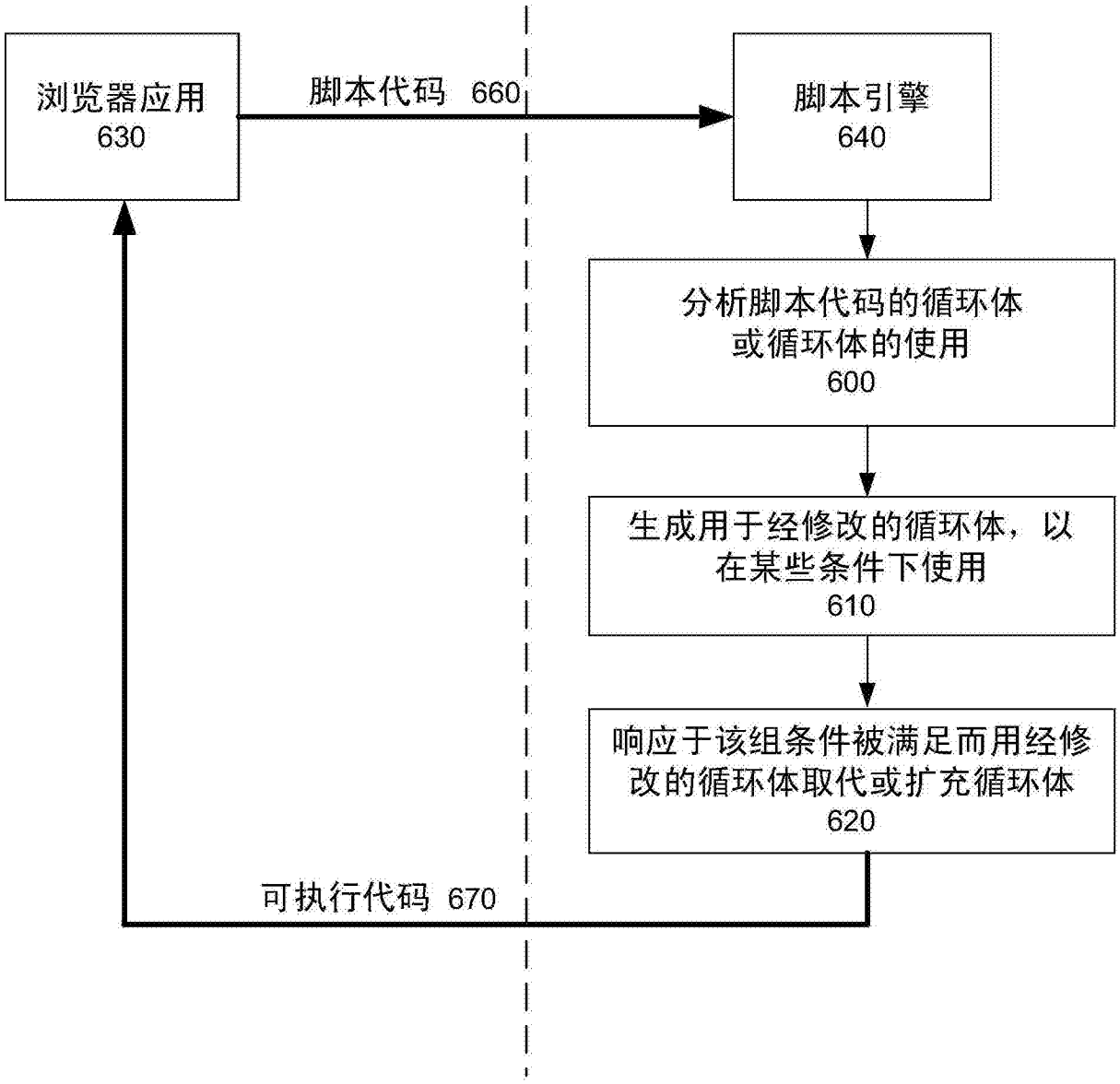

浏览器应用
630
脚本代码 660
脚本引擎
640
分析脚本代码的循环体
或循环体的使用
600
生成用于经修改的循环体，以
在某些条件下使用
610
响应于该组条件被满足而用经修
改的循环体取代或扩充循环体
620
可执行代码 670

图6

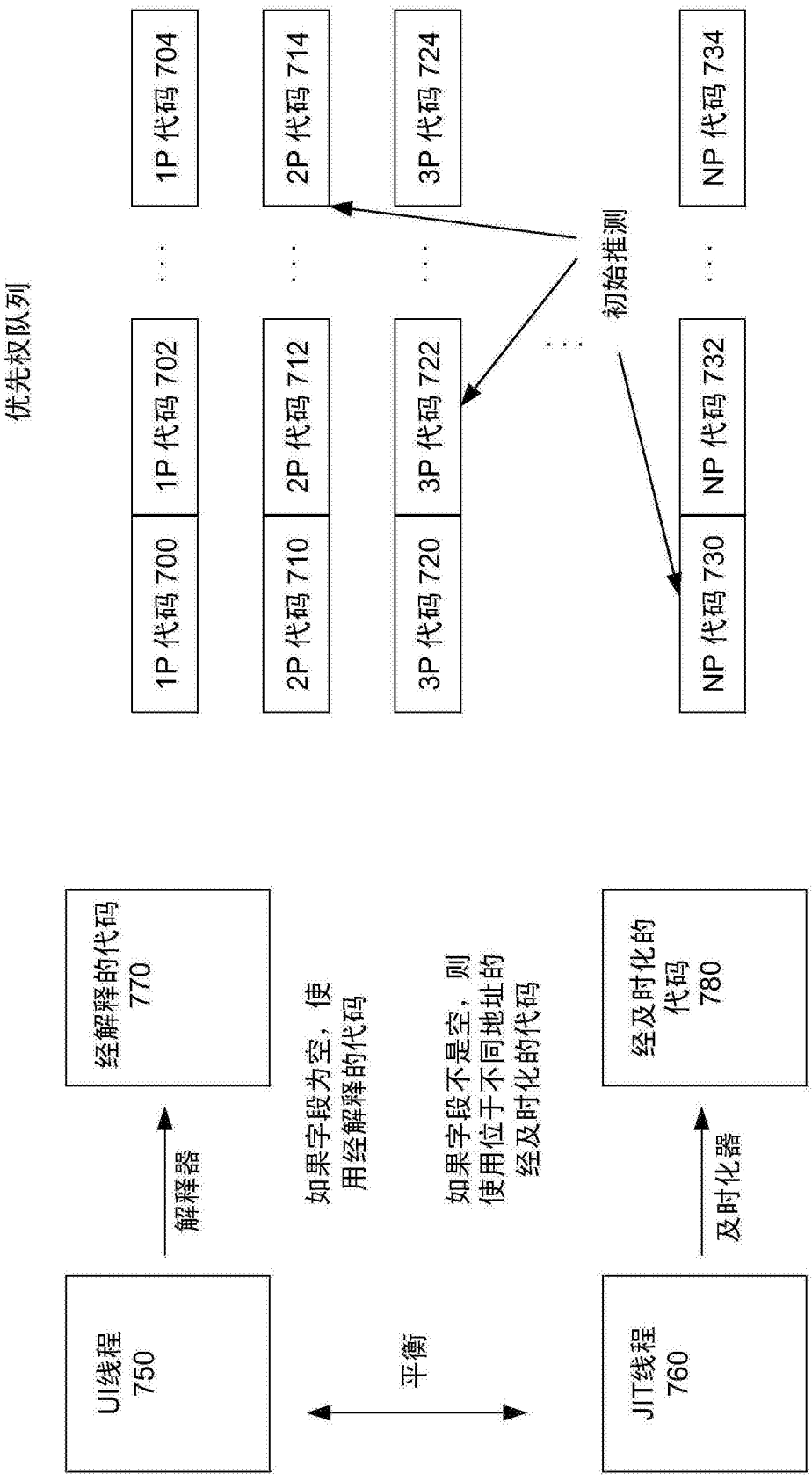

优先权队列
1P 代码 704
1P 代码 702
1P 代码 700
2P 代码 714
2P 代码 712
2P 代码 710
3P 代码 724
3P 代码 722
3P 代码 720
NP 代码 734
NP 代码 732
NP 代码 730
初始推测
经解释的代码 770
解释器
如果字段为空，使用经解释的代码
如果字段不是空，则使用位于不同地址的经及时化的代码
经及时化的代码 780
及时化器
UI线程 750
平衡
JIT线程 760

图7

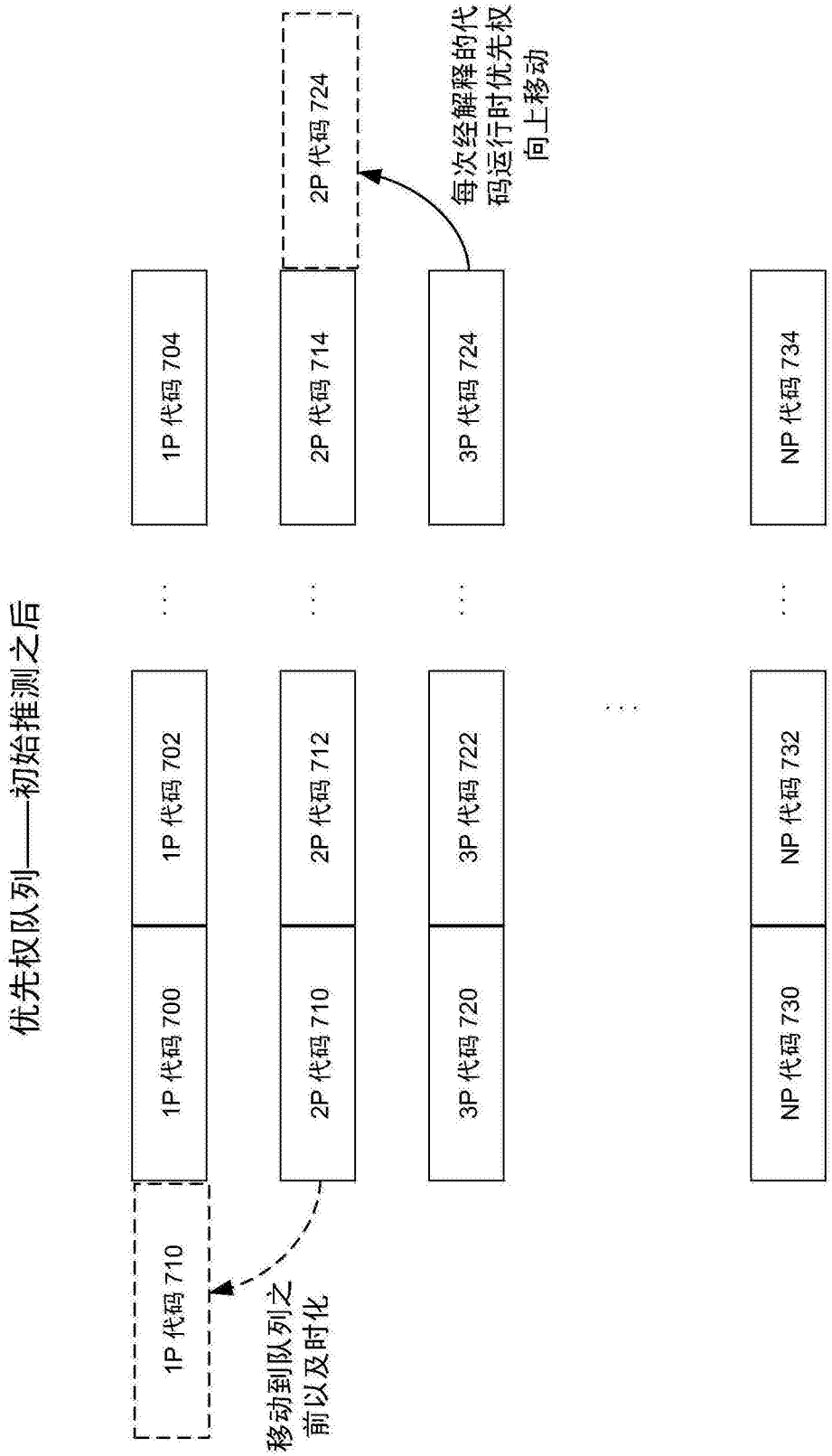

优先权队列——初始推测之后
1P 代码 700
1P 代码 702
1P 代码 704
2P 代码 710
2P 代码 712
2P 代码 714
3P 代码 720
3P 代码 722
3P 代码 724
NP 代码 730
NP 代码 732
NP 代码 734
1P 代码 710
2P 代码 724
移动到队列之前以及时化
每次经解释的代码运行时优先权向上移动

图8

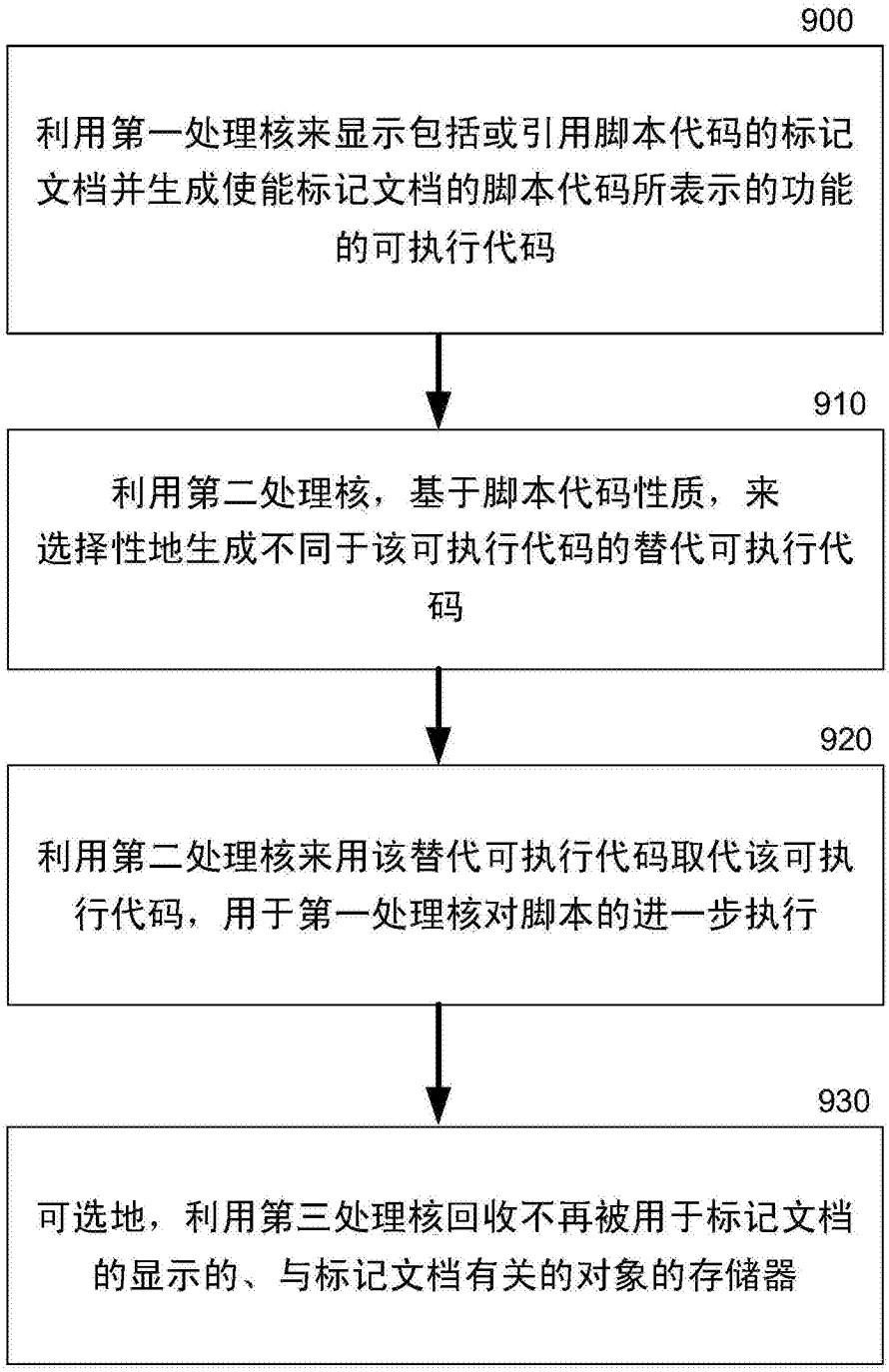

900
利用第一处理核来显示包括或引用脚本代码的标记文档并生成使能标记文档的脚本代码所表示的功能的可执行代码
910
利用第二处理核，基于脚本代码性质，来选择性地生成不同于该可执行代码的替代可执行代码
920
利用第二处理核来用该替代可执行代码取代该可执行代码，用于第一处理核对脚本的进一步执行
930
可选地，利用第三处理核回收不再被用于标记文档的显示的、与标记文档有关的对象的存储器

图9

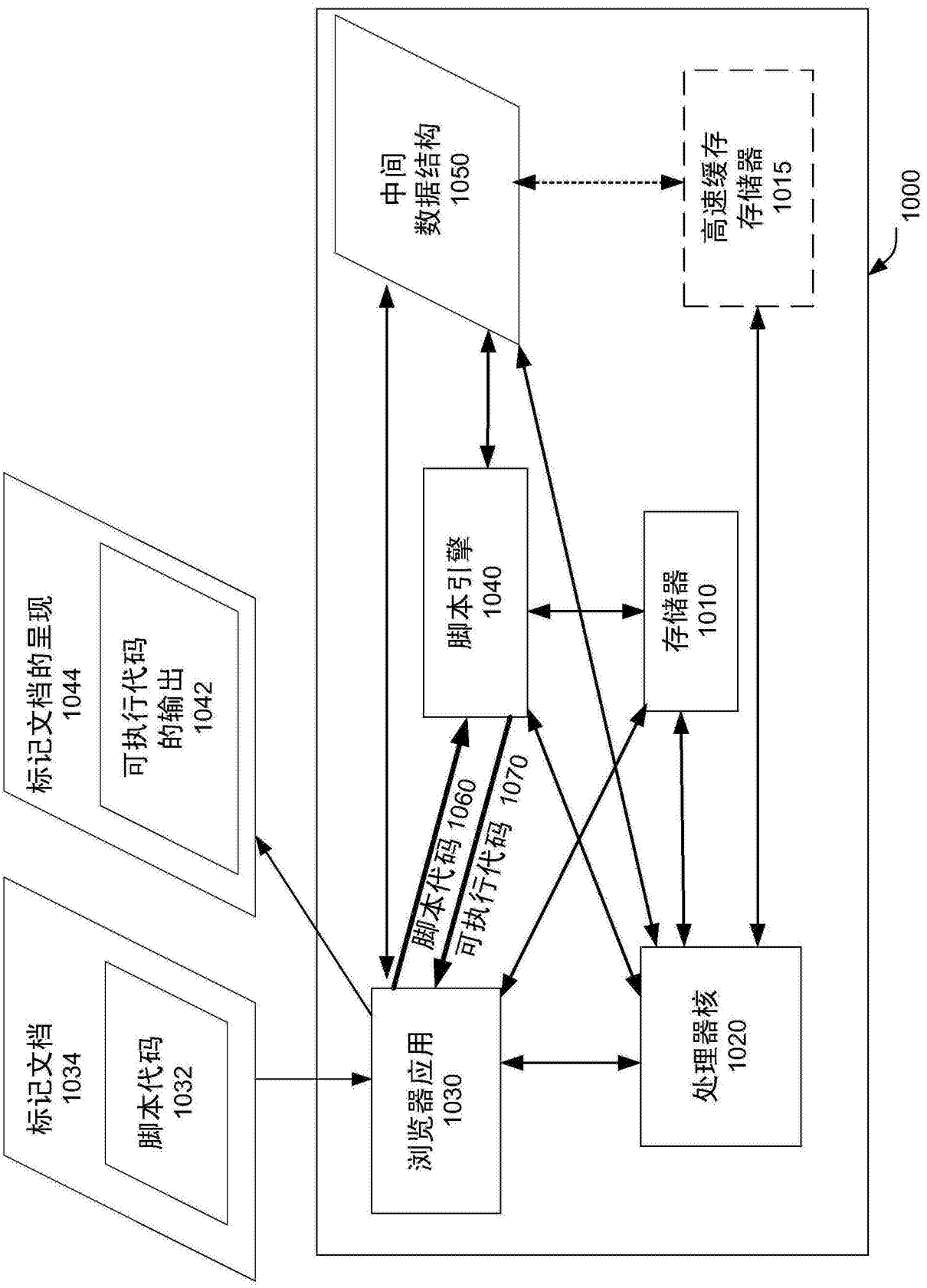

中间
数据结构
1050
高速缓存
存储器
1015
1000
脚本引擎
1040
存储器
1010
标记文档的呈现
1044
可执行代码
的输出
1042
脚本代码 1060
可执行代码 1070
标记文档
1034
脚本代码
1032
浏览器应用
1030
处理器核
1020

图10

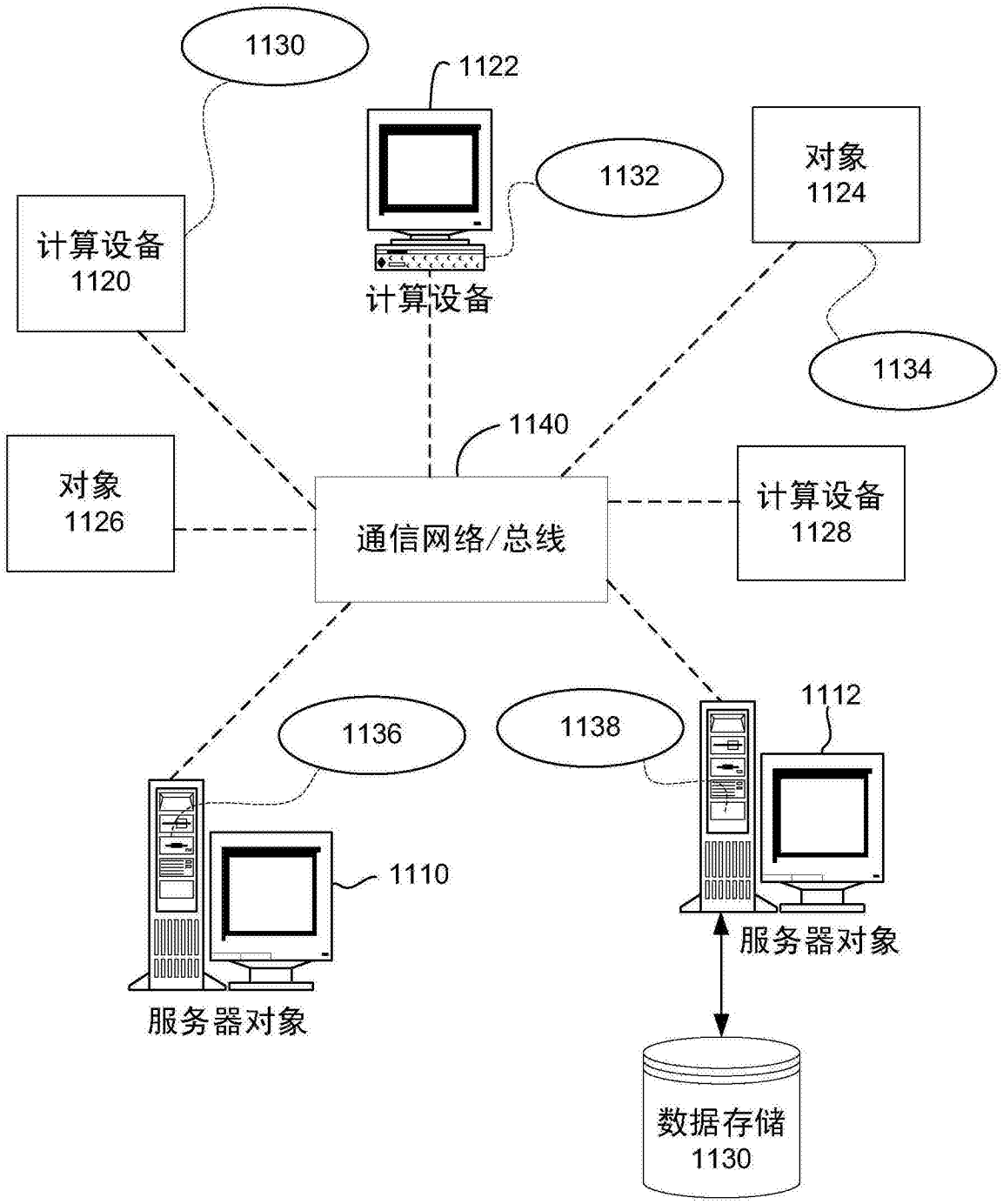

1130
1122
1132
对象
1124
计算设备
1120
计算设备
1134
1140
对象
1126
通信网络/总线
计算设备
1128
1112
1136
1138
1110
服务器对象
服务器对象
数据存储
1130

图11

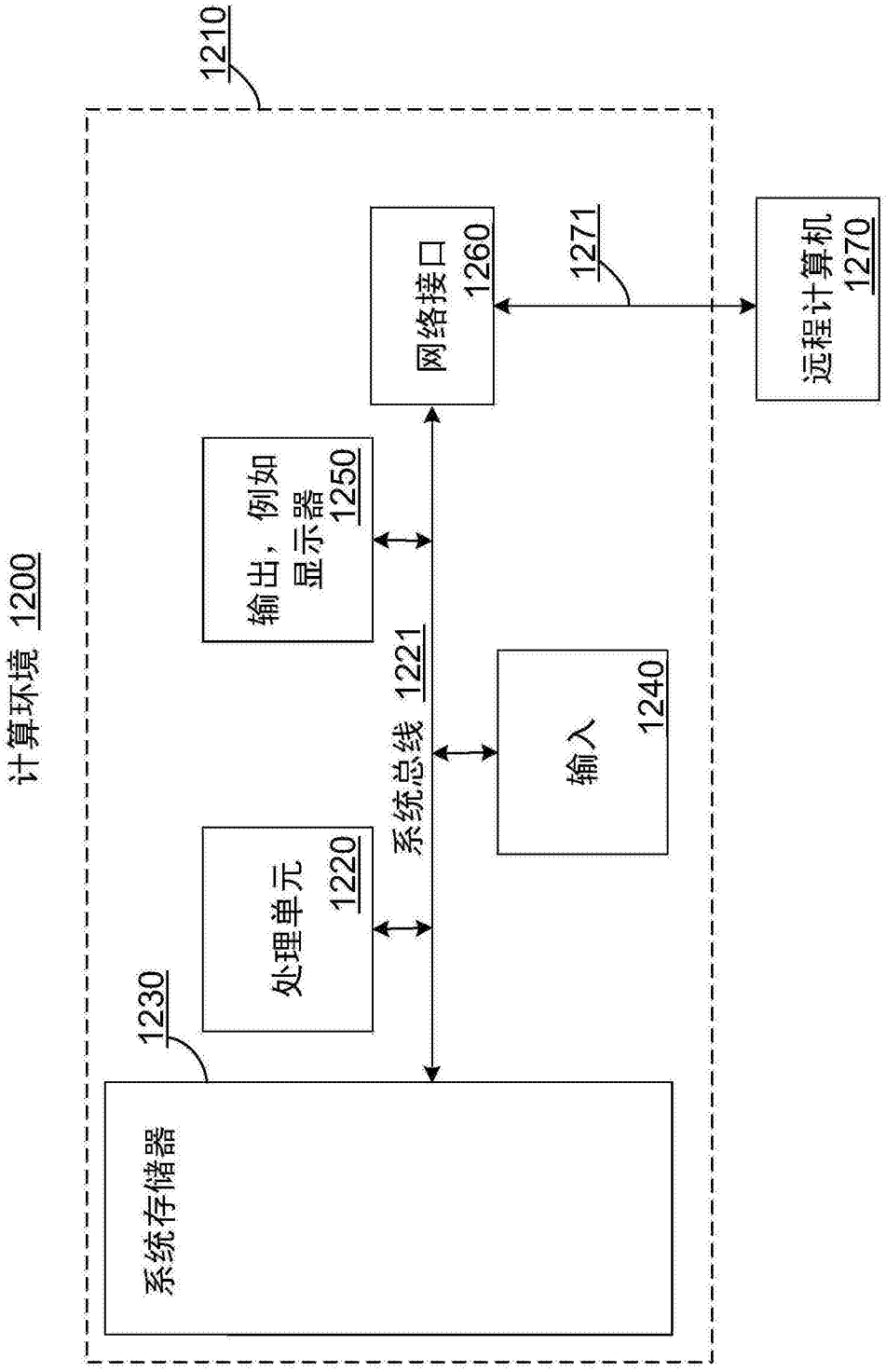

计算环境 1200
1210
网络接口 1260
1271
远程计算机 1270
输出，例如显示器 1250
系统总线 1221
输入 1240
处理单元 1220
1230
系统存储器

图12