



(12) 发明专利申请

(10) 申请公布号 CN 115380278 A

(43) 申请公布日 2022. 11. 22

(21) 申请号 202180027752.5

(22) 申请日 2021.01.20

(30) 优先权数据

16/809,025 2020.03.04 US

(85) PCT国际申请进入国家阶段日

2022.10.11

(86) PCT国际申请的申请数据

PCT/US2021/014127 2021.01.20

(87) PCT国际申请的公布数据

W02021/178056 EN 2021.09.10

(71) 申请人 甲骨文国际公司

地址 美国加利福尼亚

(72) 发明人 P·布扎克

(74) 专利代理机构 中国贸促会专利商标事务所
有限公司 11038

专利代理师 冯薇

(51) Int.Cl.

G06F 11/36 (2006.01)

G06F 8/41 (2006.01)

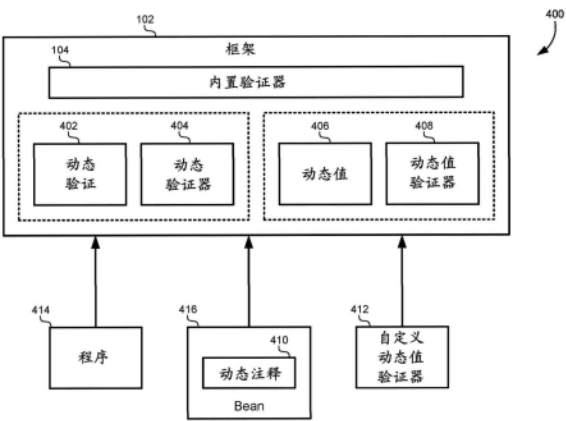
权利要求书2页 说明书19页 附图19页

(54) 发明名称

对象属性动态验证

(57) 摘要

可以增强编程语言框架以提供动态验证。动态验证允许在运行时选择用于任何变量的验证器函数,代替在编程时静态声明。代替用引用特定验证器函数或约束类型的注释来对变量进行注释,程序员可以用指示将在运行时动态选择验证器函数的注释来对变量进行注释。当创建变量的运行时实例时,编程语言框架可以识别变量上的动态验证注释,然后使用变量中的运行时值来确定应当使用哪个(哪些)验证器函数。



1. 一种包括指令的非暂态计算机可读介质,所述指令在由一个或多个处理器执行时,使得所述一个或多个处理器执行包括以下的操作:

在编程语言框架处接收对象的实例,其中对象的定义用约束进行注释;

接收用注释进行注释的一个或多个验证器,该注释识别:

对象的定义中的属性;以及

属性的值;

识别所述一个或多个验证器中的验证器,其中对象的实例中的属性的值与该验证器的注释中的属性的值匹配;以及

使用对象的实例来执行验证器。

2. 如权利要求1所述的非暂态计算机可读介质,其中所述验证器验证对象的定义中的第二属性的值。

3. 如权利要求1所述的非暂态计算机可读介质,其中所述一个或多个验证器包括第二验证器,其中对象的实例中的属性的值与该第二验证器的注释中的属性的值不匹配。

4. 如权利要求1所述的非暂态计算机可读介质,其中用其对对象的定义进行注释的约束使得编程语言框架使用动态验证而不是静态验证。

5. 如权利要求4所述的非暂态计算机可读介质,其中编程语言框架包括针对用其对对象的定义进行注释的约束的定义,其中针对约束的所述定义指定该约束可以在对象类型目标上执行。

6. 如权利要求1所述的非暂态计算机可读介质,其中编程语言框架包括针对用其对所述一个或多个验证器进行注释的注释的定义,其中该定义包括属性名称和属性值模式。

7. 如权利要求1所述的非暂态计算机可读介质,其中编程语言框架包括所述一个或多个验证器中的每个验证器从其继承的抽象类的定义。

8. 如权利要求7所述的非暂态计算机可读介质,其中所述抽象类包括确定约束是否对对象有效的抽象函数。

9. 如权利要求7所述的非暂态计算机可读介质,其中所述抽象类包括对照自定义约束对对象进行验证的受保护函数。

10. 如权利要求1所述的非暂态计算机可读介质,其中编程语言框架包括扫描类路径并识别该类路径中的所述一个或多个验证器中的每个验证器的类的定义。

11. 如权利要求10所述的非暂态计算机可读介质,其中类的定义包括过滤掉所述一个或多个验证器中的验证器的函数,其中对象的实例中的属性的值与所述一个或多个验证器中的所述验证器的注释中的属性的值不匹配。

12. 如权利要求10所述的非暂态计算机可读介质,其中编程语言框架基于类的定义来创建实例,作为在接收到对象的实例时用于动态验证的入口点。

13. 如权利要求10所述的非暂态计算机可读介质,其中类的定义在执行任何动态验证之前首先执行任何静态验证。

14. 如权利要求1所述的非暂态计算机可读介质,其中对象的实例包括所述属性和第二属性。

15. 如权利要求14所述的非暂态计算机可读介质,其中所述属性确定应当使用所述一个或多个验证器中的哪个验证器来验证指派给第二属性的值。

16. 如权利要求15所述的非暂态计算机可读介质,其中所述属性包括键-值对中的键并且第二属性包括该键-值对中的值。

17. 如权利要求16所述的非暂态计算机可读介质,其中键指示值的数据类型。

18. 如权利要求1所述的非暂态计算机可读介质,其中:

验证器用第二注释进行注释,该第二注释识别对象的定义中的第二属性和第二属性的第二值;以及

对象的实例中的第二属性的值与验证器的第二注释中的第二属性的第二值匹配。

19. 一种系统,包括:

一个或多个处理器;以及

一个或多个存储器设备,包括指令,所述指令在由所述一个或多个处理器执行时,使得所述一个或多个处理器执行包括以下的操作:

在编程语言框架处接收对象的实例,其中对象的定义用约束进行注释;

接收用注释进行注释的一个或多个验证器,该注释识别:

对象的定义中的属性;以及

属性的值;

识别所述一个或多个验证器中的验证器,其中对象的实例中的属性的值与该验证器的注释中的属性的值匹配;以及

使用对象的实例来执行验证器。

20. 一种在编程语言框架中执行动态验证的方法,所述方法包括:

在编程语言框架处接收对象的实例,其中对象的定义用约束进行注释;

接收用注释进行注释的一个或多个验证器,该注释识别:

对象的定义中的属性;以及

属性的值;

识别所述一个或多个验证器中的验证器,其中对象的实例中的属性的值与该验证器的注释中的属性的值匹配;以及

使用对象的实例来执行验证器。

对象属性动态验证

[0001] 相关申请的交叉引用

[0002] 本申请要求于2020年3月4日提交的标题为“DYNAMIC VALIDATION FRAMEWORK EXTENSION”的编号为16/809,025的美国专利申请的权益和优先权,该申请通过引用全文并入本文。

背景技术

[0003] 编程语言最近提供了用于对成员变量、类和函数强制执行约束的高级机构。约束可以被用于验证不同的数据字段,以确保它们满足一个或多个预定义要求中的一些。传统上,约束是使用包括在整个软件中的定制的错误检查代码来强制执行的。但是,作为编程语言框架的一部分,现代编程语言已经开始提供针对约束集合验证数据的机构。程序员无需为每个项目和每个数据集重写约束和验证代码,而是可以代替地使用编程语言框架的约束强制执行机构来提高软件开发的效率和可以验证常见数据类型的一致性。

[0004] 例如,Java编程语言提供对强加在类模块或“bean”上的约束的验证。bean验证在JSR 380规范中定义,它允许对Java bean进行静态验证。在静态验证中,bean内部的属性(例如,成员变量)满足一些预定义准则,诸如非空值、非空集、在指定的数值范围内等。如果未满足这些预定义准则,那么框架报告约束违反。验证框架还允许用户创建用户定义的约束和用户定义的验证器。这些允许用户定制各种验证例程以匹配期望的数据格式。

发明内容

[0005] 现代编程语言框架提供了使用预定义约束对运行时值执行静态验证的构造。当创建运行时实例时,可以将存储在变量和/或成员属性中的运行时值提供给验证器函数,这确保该值符合预定义约束。预定义约束可以包括数字范围、可接受的字符串模式、日期范围、特定格式化和/或可以对数据值强制执行的其它要求。为了激活约束,变量或成员属性可以用引用特定验证器函数的文本字符串进行注释。因此,要用于特定变量的验证器函数是在编程时而不是在运行时设置的。

[0006] 本文描述的实施例增强了编程语言框架以提供动态验证。动态验证允许在运行时选择用于任何变量的验证器函数,而不是在编程时静态声明。代替用引用特定验证器函数或约束类型的注释来对变量进行注释,程序员可以用指示将在运行时动态选择验证器函数的注释来对变量进行注释。当创建变量的运行时实例时,编程语言框架可以识别变量上的动态验证注释,然后使用变量中的运行时值来确定应当使用哪个(哪些)验证器函数。

[0007] 可以修改编程语言框架以包括附加注释定义。一个注释定义可以被用于注释应当经受如上所述的动态验证的变量。另一个注释定义可以被用于注释用户定义动态值验证器函数本身。第二个注释可以接收要被动态验证的属性的名称和/或值。例如,类定义可以包括作为键-值对操作的两个成员属性(例如,字符串键、字符串值)。第二个注释可以引用“键”属性并确定“键”属性中的值是否指示这个键-值对存储了电话号码(例如,键=“phone”;值=“571-555-1534”)。验证器函数上使用的注释可以指定当名为“key(键)”的属

性具有“phone (电话)”值时应当使用它,然后验证器函数内部的代码可以验证“值”属性以确定它是否存储了格式正确的电话号码。如果“键”值存储了一些不同的东西,诸如“日期”或“时间”,那么这个验证器将不会在这个类实例上执行。这个决定是由框架在运行时做出的。

[0008] 当编程语言框架在运行时接收到被注释用于动态验证的变量时,框架可以首先执行任何传统的静态验证。如果静态验证失败,那么可能不必进行动态验证。在静态验证完成之后,编程语言框架可以生成类路径或程序目录中可用的所有动态验证器函数的列表。可用动态验证器的列表可以包括已经由程序员定义的任何自定义动态值验证器函数。然后,框架可以循环通过可用动态值验证器函数的列表,并且识别带注释变量的运行时值满足动态值验证器函数上带注释约束的任何内容。继续上面的示例,框架可以识别类路径中的电话号码值验证器函数,并且将函数注释中的attributeName=“key”和attributeValue=“phone”约束与变量的运行时值进行比较。然后可以对运行时值执行任何识别出的验证器函数。在一些实施例中,框架可以提供包括用于执行动态验证的受保护的辅助函数的抽象基类,并且这个抽象基类可以在程序员编写的自定义动态值验证器中被覆盖(overridden)。

[0009] 这些编程语言框架增强允许程序员在运行时动态选择的不同验证器函数中封装成员属性之间的不同依赖关系(dependency)。当仅使用静态验证时,要求验证器函数包括比较不同可能值的if/then语句的分支。随着代码的发展,这些语句难以随着时间的推移而维护。但是,当使用动态验证时,用于验证的每个变量依赖关系都可以被单独且独立地处理,并且可以通过用添加的依赖关系对新的验证器函数进行注释来轻松添加新的依赖关系而不受限制。

附图说明

[0010] 通过参考本说明书的其余部分和附图,可以实现对各种实施例的本质和优点的进一步理解,其中,贯穿若干附图使用相似的附图标记指代相似的组件。在一些情况下,子标记与附图标记相关联以表示多个相似组件中的一个。当在没有指定现有子标记的情况下引用附图标记时,旨在指代所有这样的多个相似组件。

[0011] 图1图示了根据一些实施例的用于编程语言的约束验证框架的图。

[0012] 图2A图示了根据一些实施例的使用内置约束注释的bean的示例。

[0013] 图2B图示了自定义验证器/约束的定义。

[0014] 图3图示了根据一些实施例的验证bean中的约束的程序。

[0015] 图4图示了根据一些实施例的用于动态验证的框架扩展的图。

[0016] 图5图示了根据一些实施例的动态验证约束注释。

[0017] 图6图示了根据一些实施例的使用动态验证的bean的示例。

[0018] 图7图示了根据一些实施例的添加到定义动态值之间的关系的框架的新注释。

[0019] 图8图示了根据一些实施例的用于DynamicValueValidator的抽象类的示例。

[0020] 图9A图示了根据一些实施例的DynamicValidator类的定义。

[0021] 图9B图示了根据一些实施例的用于DynamicValidator的isValid()函数。

[0022] 图9C图示了根据一些实施例的用于DynamicValidator的方法以在DynamicValue

注释中定义的元数据和对应的bean之间执行静态验证和匹配。

[0023] 图10A图示了根据一些实施例的用于动态验证的用户定义的自定义动态值验证器的示例。

[0024] 图10B图示了根据一些实施例的具有多个约束注释的自定义动态值验证器的示例。

[0025] 图11图示了根据一些实施例的其中可以调用动态验证的程序的示例。

[0026] 图12图示了根据一些实施例的用于在运行时动态验证变量依赖关系的方法的流程图。

[0027] 图13图示了用于实现实施例中的一些实施例的分布式系统的简化框图。

[0028] 图14图示了系统环境的组件的简化框图,通过该系统环境,实施例系统的组件所提供的服务可以作为云服务来供应。

[0029] 图15图示了其中可以实现各种实施例的示例性计算机系统。

具体实施方式

[0030] 在Java编程语言中,各种标准(JSR 380、JSR 349、JSR 303等)描述了可以被用于验证Java bean内的属性的框架。当数据从表示层流向持久层时,通常使用bean验证。在这个框架之前,用户通常必须在其应用的每一层中复制他们的验证代码。验证框架允许用户经由添加到域或类的元数据使用单个机构来执行验证。元数据可以包括经由XML描述符添加的约束。例如,用于Java类的默认元数据源可以包括添加到目标的Java注释。然后可以在运行时对照由注释引用的约束来验证目标数据。

[0031] 约束一般由两个单独的部分组成。约束的第一部分包括约束注释,其包括可以添加到代码以识别运行时约束的注释。通用约束注释可以以字段、方法、构造函数(constructor)、参数、类型等为目标。约束的第二部分可以包括实现约束的验证器的集合。例如,约束可以包括成员变量的可允许数值范围,并且验证器可以包括isValid()函数,该函数包括确定成员变量的运行时值是否落入可允许数值范围内的代码。因此,约束注释和用于该约束注释的验证器一起工作以对程序数据执行约束的运行时验证。

[0032] 图1图示了根据一些实施例的用于编程语言的约束验证框架102的图100。在计算机编程中,软件框架是一种抽象,其中提供通用功能性的软件可以通过用户代码来增强。框架102一般提供标准化的库和方法来构建和部署应用,并且它可以提供作为更大软件平台的一部分的功能以促进软件应用的开发。例如,框架102可以包括支持程序、编译器、代码库、工具集、应用编程接口(API)和其它组件或开发环境。例如,Java编程语言可以与提供可以被用于实现约束验证的实用程序、编程语言特征和代码库的各种框架组合。如本文所使用的,框架102本身可以与用户代码或自定义、用户定义的验证器和约束注释区分开来。

[0033] 用户程序106可以包括作为软件应用被编译、汇编、解释和/或执行的用户代码。例如,程序106可以包括实例化类对象、执行函数、存储变量和/或以其它方式实现软件应用的各个方面的main()函数。程序106可以被认为为用户代码,并且可以区别于框架102。此外,用户代码可以包括一个或多个自定义类/对象定义。在Java编程语言中,这些类可以组织在称为“bean”的包中。程序106可以使用bean108中的类定义来实例化运行时对象。

[0034] 如许多类定义中常见的那样,bean 108内的类可以包括成员变量、成员函数和其

它数据。约束注释110可以在不同位置被添加到bean108中的类定义。注释110的范围可以基于注释110的位置和/或对应约束注释定义的定义而变化。如将在下面的示例中描述的,注释110通常包括识别约束并且可以为约束提供值的代码行,诸如可以在验证约束时使用的消息和/或其它信息。

[0035] 为了验证约束,验证器可以由程序106执行。在一些情况下,框架102可以包括一个或多个内置验证器104。内置验证器可以包括验证内置约束定义的集合的代码。例如,一些Java框架可以包括对简单约束的内置支持。javax.validation.constraints包(package)包括内置约束,诸如强制执行最小/最大值、强制执行空/非空值、强制执行预定义正则表达式模式、强制执行预定尺寸等的约束。bean 108可以通过简单地将对应的注释110添加到bean 108的代码来利用这些内置的验证器/约束104。

[0036] 除了使用内置的验证器/约束104之外,程序106还可以使用自定义验证器/约束112。为了使用自定义验证器和/或自定义约束,用户定义的验证器定义和/或约束定义可以被提供给程序106。自定义验证器/约束112可以各自实现由框架102提供的预定义接口和/或抽象类,使得自定义验证器/约束112可以由框架102执行,其方式与在运行时使用内置验证器/约束104相同。

[0037] 图2A图示了根据一些实施例的使用内置约束注释110的bean108的示例。这个bean包括封装键-值数据结构的公共类的定义。许多现代数据结构使用这种范式以统一的格式存储许多不同的数据类型。通用格式可以使用键-值结构,其中数据被存储为键-值对。对的键部分可以被用于描述该对的值部分的数据类型(或任何其它元数据类型)。例如,对的键部分可以定义数据类型,诸如电话号码、地址、用户名等。而对的对应值部分可以包括特定的电话号码、特定的地址、特定的用户名等。这种灵活的系统允许存储任何类型的数据,只要键定义数据类型并且值定义数据值即可。这个KeyValue类可以在本文用作可以使用如下所述的动态验证的bean的示例。但是,这个类仅作为示例使用,并不意味着进行限制。动态验证可以用于任何数据结构,包括类和数据的其它封装。

[0038] bean 108可以包括用于键和值两者的私有成员变量。图2A的示例包括可以被用于验证字符串的值的约束注释110。一些Java框架中的内置约束中的一些可以包括@NotBlank约束,它表示键变量中的字符串值不能为空。注释110包括将伴随约束违反的消息设置为指定值的参数,诸如“键为空”。代替编写自定义验证代码,用户可以简单地包括约束注释110以强制执行可以由框架中的函数自动验证的预定义约束,而无需编写附加代码。

[0039] 除了使用内置约束之外,编程语言框架还可以允许用户定义他们自己的自定义约束注释和自定义验证器以验证这些约束。图2B图示了用于自定义验证器/约束112的定义。首先,提供了约束定义220的示例,其允许用户定义他们自己的约束注释。约束定义220可以指定可以对其应用约束的目标(例如,字段、方法、类型等)。约束定义220还可以包括消息、组和/或有效载荷字段。可以添加其它字段以提供可以增强验证的信息或其它设置。例如,消息字段可以被用于创建错误消息,该错误消息可以被如图2A中所示的参数覆写。组字段可以定义约束所属的组。有效载荷字段可以指定可以与约束相关联的其它数据。例如,有效载荷可以被用于将严重性与约束相关联。

[0040] 与约束定义220一起,还提供了验证器定义222以具体验证约束注释定义220的约束。Java编程语言上下文中的验证器定义222可以实现ConstraintValidator接口,并且可

以覆盖两个公共功能。`Initialize()`函数可以被用于从约束注释接收数据并初始化验证器。`isValid()`函数保持用于针对对应对象验证对应约束的代码。如果在与验证器定义222中的对象类型对应的目标对象上使用用于约束定义220的注释,那么可以验证约束。在一些情况下,验证器定义222可以被一般地定义为使得目标类是`Object`(对象)类型。在这种情况下,框架为使用约束进行注释的所有目标调用用户定义的验证代码。例如,验证器定义222可以验证用具有`MyObject`类型的`@MyConstraint`约束注释的所有目标bean。注意的是,不能基于对象中的属性的运行时值来选择验证器定义222。

[0041] 图3图示了根据一些实施例的验证bean 108中的约束的程序106。首先,程序106实例化默认类型的验证器302。程序106然后定义对参数列表中的指定对象列表执行验证的`validate()`函数308。程序106的`main()`函数然后实例化具有键和值属性的指定字符串的新`KeyValue`对象。然后将这个实例化的对象作为参数306传递到验证函数。验证函数308然后在对象上执行指定的验证器并报告任何约束违反。

[0042] 虽然用户可以在现有编程语言框架中定义自定义验证器和约束,但是本文描述的实施例改进了现有编程语言框架以提供用于基于运行时值通过框架验证属性之间的依赖关系的机构。这些改进允许轻松实现自定义验证器和约束以提供深度验证,其中一个属性的值可以影响在另一个属性上被验证的约束的类型。例如,如果`KeyValue`对象中的键的值包括字符串值“phone(电话)”,那么用户可能希望验证值属性中的字符串以确保它是格式正确的电话号码。在另一个示例中,如果键是“date(日期)”,那么用户可能希望验证该值以确保字符串是正确格式化的日期。这不能通过静态验证来执行,因为只验证了目标的单个值。并且,虽然用户可以通过定义他们自己的约束和提供验证代码来验证关系,但用户需要为目标中的每个关系定义约束和验证器的新组合。这个代码很难维护并且很难解析。本文描述的实施例代替地描述对验证框架的扩展,该框架以通用方式解决这个问题,而不要求针对每个关系的特定代码。

[0043] 图4图示了根据一些实施例的用于动态验证的框架扩展的图400。除了框架102已经扩展为包括多个附加模块:动态验证402注释、动态验证器404、动态值注释406和动态值验证器408之外,这个图400与图1的图100相似。下面将更详细地描述这些模块中的每一个。简而言之,这些模块允许框架处置`@DynamicValidation`注释,然后基于运行时值动态地验证不同数据字段之间的不同关系,诸如上述`KeyValue`类中的键和值之间的关系。应当注意的是,这些模块是编程语言框架的一部分,并且不是要求用户编写的自定义类。这些是可以与框架102一起部署的模块,因此可以区别于用户可以编写以与框架102交互的代码。

[0044] 在这个示例中,bean 416可以包括动态注释410,诸如下面详细描述的`@DynamicValidation`注释。当程序414创建在bean 416中定义的对象实例时,动态注释410将指示框架102使用用户定义的自定义动态值验证器412来执行动态验证。如下所述,自定义动态值验证器412可以扩展在框架102的扩展中可用的抽象基类,从而最小化可能要求用户编写的代码。

[0045] 图5图示了根据一些实施例的动态验证约束注释。动态验证的出发点是约束注释及其对应的验证器的定义。动态验证约束定义402定义可以添加到类来向框架指示应当执行动态验证而非传统验证的注释。由于动态验证可以被用于评估类的不同成员变量之间的关系,因此约束定义402将目标值设置为`TYPE`(类型),这允许访问对象定义内的所有属性。

为这个约束定义402给出的名称是DynamicValidation(这与@DynamicValidation注释对应)。但是,这个名称仅作为示例使用并且不意味着限制。任何其它名称都可以被用于动态验证约束定义402。在一些实施例中,不需要用注释定义默认消息502。如下所述,在这个注释内定义的任何消息都可以在验证器验证期间被省略和替换。还要注意的,定义502指定这个约束可以由DynamicValidator类来验证,其名称也仅作为示例使用,并且其功能在下面详细描述。

[0046] 图6图示了根据一些实施例的使用动态验证的bean 416的示例。使用上面图5中描述的定义,可以使用@DynamicValidation约束注释对KeyValue类进行注释。这允许框架调用验证器,该验证器动态评估键和值成员属性的运行时值之间的关系。注意的是,因为用于动态验证约束定义402的目标值是TYPE,所以可以在类级别而不是在个体成员变量级别应用动态注释410。基于动态验证约束定义402的这个动态注释410可以被添加在应当调用动态验证的任何bean的类似位置处。

[0047] 图7图示了根据一些实施例的添加到定义动态值之间的关系的框架的新注释。如下文详细描述的,这个注释可以被用于自定义动态值验证器,以指示可以使得验证器被调用的运行时属性值。具体而言,图7图示了定义值之间的关系的动态值注释定义406。这个特定的注释包括两个方法:attributeName和valuePattern.attributeName方法返回将在目标bean中检查值的属性的名称.valuePattern方法返回用于检查值的正则表达式。这个注释可以被用于下面详细描述的DynamicValueValidator的实例。这个扩展确保只有满足@DynamicValue中定义的关系定义的bean才需要被进一步验证。此外,这个注释可以多次用于目标DynamicValueValidator实例,以允许用户定义多个关系。例如,如果对于DynamicValueValidator有更多个@DynamicValue注释,那么可以进一步验证由注释定义的这些关系中的每一个。

[0048] 图8图示了根据一些实施例的用于DynamicValueValidator的抽象类的示例。首先,注意DynamicValueValidator类是抽象类,这意味着由用户定义的任何自定义动态值验证器类都应当扩展DynamicValueValidator 408,并且扩展应当像上面描述的那样用至少一个@DynamicValue进行注释。这个抽象类可以在一些Java框架中的javax.validation.ConstraintValidator抽象类之后被模式化(patterned),其由常规用户定义的验证器扩展。

[0049] 扩展DynamicValueValidator类的每个验证器都应当提供isValid()方法802的实施方式。如果在@DynamicValue中定义的关系对于已验证的bean满足,那么该方法可以由扩展类调用。具体而言,用于提供实际验证的代码可以在自定义动态值验证器中在isValid()函数中提供,该函数覆盖抽象基类中的这个抽象函数。

[0050] 此外,DynamicValueValidator 408包括两个受保护的方法:validateCustomConstraints()804和useConstraint()806,它们可以被用于促进用户定义的验证器。如果validateCustomConstraints()方法804定义了它自己的要被静态验证的对象,那么该方法804可以在用户创建的验证器中使用。验证背后的核心概念之一是验证器定义自己的内部类,该内部类具有用约束注释的属性。然后,用来自正被验证的bean的数据填充这些属性。然后将该类的实例传递到validateCustomConstraints()方法804。如果这个方法804识别出经验证的对象中的任何约束违反,那么它从违反中获取第一消息,将其传递到ConstraintValidationContext,并且返回False(假)。如果用户指定对失败的bean

的深度验证(例如,用户验证是否可以打开到所提供的URL的连接),那么帮助者方法 `useConstraint()` 806可以被用于提供用户定义的消息。这两个函数都可以由扩展这个类的用户编写的自定义动态值验证器调用。作为框架扩展的一部分,这些函数减少了编写自定义动态值验证器所需的代码量。

[0051] 图9A图示了根据一些实施例的用于 `DynamicValidator` 404类的定义。`DynamicValidator`实现了框架中已经定义的 `ConstraintValidator`模板。重要的是, `DynamicValidator` 404用作由本文描述的实施例添加的用于执行动态验证的框架扩展的入口点。这个类首先创建用于检查静态定义的约束的验证器,创建从类路径扫描加载的动态验证器的列表,并且创建 `VALIDATOR`的静态实例。应当注意的是, `DynamicValidator`可以被用于任何类型的bean。当创建 `ConstraintValidator`的实例时,它加载扩展上述 `DynamicValueValidator`的所有类。构造函数902根据图9A中的代码创建 `DynamicValidator`的实例。具体而言,如果类路径中的任何动态验证器尚未被实例化,那么它们被同步加载和实例化。如图9A中所示,代码过滤掉任何抽象类、任何没有无自变量构造函数的类、以及不具有至少一个正确的 `@DynamicValidation`注释的类。接下来,这些类中的每一个被实例化和高速缓存。一些实施例可以使用完整的类路径扫描,而其它实施例可以使用上下文和依赖关系注入 (Contexts and Dependency Injection, CDI) 来获得具有 `@DynamicValidation`注释的 `DynamicValueValidator`的所有实例。如果使用CDI,那么用户定义动态bean验证器应当将其范围定义为对CDI查找可见。这种方法在Java EE环境中可以是更优选的。

[0052] 图9B图示了根据一些实施例的用于 `DynamicValidator`的 `isValid()` 910函数。一旦在 `DynamicValidator` 404中调用 `isValid()` 方法910,它就可以对bean内的任何静态定义的约束执行验证(912)。如果违反了任何静态约束,那么可能不必进行动态验证,因此 `isValid()`方法可以返回并退出。在一些实施例中,静态定义的约束可以具有比动态定义的约束更高的优先级。可替代地,如果静态定义的约束上没有约束违反,那么可以开始动态验证。可以过滤上面在构造函数902中填充的动态验证器的列表,以找到为这种特定类型的bean定义并且满足 `@DynamicValue`中定义的关系的验证器(914)。注意的是,如果属性不具有String类型,那么可以调用 `toString()`方法来检索属性的字符串表示。最后,满足这些条件的任何验证器都可以被传递到它们相应的 `isValid()`方法(916)。

[0053] 图9C图示了根据一些实施例的用于 `DynamicValidator`在 `DynamicValue`注释中定义的元数据与对应的bean之间执行静态验证和匹配的方法。`isStaticValid()`函数920可以对在当前类中找到的所有特性执行验证,以及在任何父类中递归地执行验证。这个私有函数920在上面的图9B中被调用以在执行任何动态验证之前首先确定是否有任何静态验证失败。`matchDynamicAnnotation()`函数922可以执行在 `DynamicValue`注释中定义的元数据与bean之间的匹配。如果注释中定义的数据与bean中找到的数据匹配,那么这个私有成员函数可以返回true(真)。当过滤不具有指定关系条件的动态验证器时,这个函数922在上面的图9B中被调用。

[0054] 图7-图9C图示了根据一些实施例的可以被包括在用于执行动态验证的框架扩展中的各种模块的实施方式。这些示例模块可以使用Java编程语言和特定Java构造作为示例来提供实现公开。但是,这些示例并不意味着限制。用于扩展框架以提供动态验证的原理可以应用于任何编程语言,并且可以具有除这些图中作为示例具体使用的实施方式以外的不

同实施方式。

[0055] 图10A图示了根据一些实施例的用于动态验证的用户定义的自定义动态值验证器412的示例。自定义动态值验证器412可以是需要由用户编写以采用动态验证的唯一代码。首先,验证器412应当包括一个或多个@DynamicValue注释1002的实例。验证器412还应当扩展1004上面图8中描述的DynamicValueValidator抽象类。这个扩展的类从父类提供了用于抽象isValid()函数1006的实施方式。一些实施例还可以提供私有帮助者函数以执行验证。

[0056] 动态@DynamicValue注释1002指定将触发这个动态验证的变量和该变量的值。在这个示例中,当目标对象包括具有属性名称“key(键)”的变量并且该属性的值包括具有注释1002中描绘的“phone number(电话号码)”文本的字符串时,将触发动态验证。验证器412扩展1004专门用于上面图2A和图6中描述的KeyValue类型的对象的DynamicValueValidator抽象父类。isValid()接收目标KeyValue对象,然后使用私有PhoneNumber类来验证电话号码。此时,KeyValue目标对象中的值变量预期具有要验证的有效电话号码,因为键变量动态地指示这一点。注意的是,如果KeyValue对象中的属性的值与自定义动态值验证器412的注释中的值不匹配,那么不会对对象的实例执行验证器。

[0057] 图10B图示了根据一些实施例的具有多个约束注释的自定义动态值验证器的示例。虽然图10A中的示例仅使用单个@DynamicValue注释,但其它实施方式可以使用多个@DynamicValue注释。例如,上面描述的KeyValue类在单个值中包括单个键。其它实施例包括包含两个键和单个值的TwoKeyValue类。第一个键(key1)仍可以用于指定值是电话号码,而第二个键(key2)可以用于指定国家代码。因为每个国家的电话号码可以具有不同的格式,所以两个键都可以单独包括在@DynamicValue注释1010中,以便调用这个特定的自定义动态值验证器412。除了可以使用TwoKeyValue类而不是KeyValue类来扩展1012类以外,这个验证器412可以与图10A中的验证器相同地操作。类似地,isValid()函数1014可以接收TwoKeyValue对象而不是KeyValue对象。

[0058] 图11图示了根据一些实施例的其中可以调用动态验证的程序414的示例。如上文关于图6所述,KeyValue类已经用@DynamicValidation约束进行注释。当创建KeyValue类的新实例1104时,可以为键和值属性提供字符串值。在这个示例中,键被设置为“phone(电话)”,这将符合图10中的自定义动态值验证器412。然后可以调用validate()函数来动态检查键(“phone”)的值并调用自定义动态值验证器412以确保在值属性中提供的电话号码具有正确的格式。

[0059] 图12图示了根据一些实施例的用于在运行时动态验证变量依赖关系的方法的流程图1200。该方法可以包括在编程语言框架处接收对象的实例(1202)。对象的实例可以从对象定义中实例化。例如,可以基于类定义文件来实例化类对象。类定义可以是bean或其它软件模块的一部分。对象的定义(诸如类定义)可以用约束进行注释。约束可以指示对象的实例应当接受动态验证而不是静态验证或其它形式的自定义、用户定义的验证例程。动态验证可以暗示应用于对象的实例的验证例程和约束的类型取决于对象实例中的变量运行时值。例如,@DynamicValidation约束注释可以用在类定义上,如图6中所示。对象(例如,KeyValue对象)的实例可以被实例化并作为验证函数的一部分传递到框架,如图3中所示。

[0060] 该方法还可以包括接收用注释进行注释的一个或多个验证器,该注释识别对象的

定义中的属性和该属性的值 (1204)。例如,图10A和图10B中的自定义动态值验证器可以使用图7中的@DynamicValue注释进行注释。注释可以包括识别对象的定义中的属性的属性名称,以及用于该属性的对应值。例如,该属性可以由诸如“key”之类的属性名称来标识,并且值可以包括值字符串,诸如在上面的示例中使用的“phone number”。验证器可以由框架接收,因为它扩展了抽象基类。验证器还可以实现布尔函数,该函数返回值是否通过了约束的指示(例如,isValid()函数)。验证器可以包括实现约束(诸如格式、最小值、最大值、范围和/或任何其它变量约束)的用户定义的代码。

[0061] 该方法可以附加地包括识别一个或多个验证器中的验证器,其中对象的实例中的属性的值与该验证器的注释中的属性的值匹配 (1206)。例如,图9A中的DynamicValidator类可以识别运行时路径中的所有验证器,或者以其它方式与程序相关联。然后,用于DynamicValidator的构造函数可以循环通过每个可用的验证器并识别其注释(例如,DynamicValue参数)与对象实例中对应属性的值匹配的验证器。在一些实施例中,每个验证器可以具有多于一个(即,多个)DynamicValue注释,可以满足注释中的每一个以在对象实例上执行验证器。

[0062] 该方法还可以包括使用对象的实例来执行验证器 (1208)。识别出的验证器以及与注释准则匹配的任何其它验证器都可以在对象的实例上执行。例如,可以针对约束来验证属性值,并且可以基于另一个属性的值来选择约束,从而允许属性依赖关系影响执行哪些验证器以及这些验证器的结果。

[0063] 应当认识到的是,图12中所示的特定步骤提供了根据各种实施例的执行动态验证的特定方法。根据替代实施例,也可以执行其它步骤顺序。例如,替代实施例可以以不同次序执行上面列出的步骤。而且,图12中所示的各个步骤可以包括多个子步骤,这些子步骤可以根据各个步骤适当地以各种顺序执行。此外,可以根据特定应用添加附加的步骤或移除步骤。

[0064] 本文描述的每个方法可以由计算机系统实现。这些方法的每个步骤可以由计算机系统自动执行,和/或可以设置有涉及用户的输入/输出。例如,用户可以为方法中的每个步骤提供输入,并且这些输入中的每一个可以响应于请求这种输入的具体输出,其中输出是由计算机系统生成的。可以响应于对应的请求输出而接收每个输入。此外,输入可以从用户接收、作为数据流从另一个计算机系统接收、从存储器位置检索、通过网络检索、从web服务请求等。同样,可以将输出提供给用户、作为数据流提供给另一个计算机系统、保存在存储器位置中、通过网络发送、提供给web服务等。简而言之,本文描述的方法的每个步骤都可以由计算机系统执行,并且可以涉及去往和来自计算机系统的可能涉及或可能不涉及用户的任何数量的输入、输出和/或请求。可以说,不涉及用户的那些步骤是由计算机系统自动执行的,而无需人工干预。因此,根据本公开将理解的是,本文描述的每种方法的每个步骤可以被更改为包括去往和来自用户的输入和输出,或者可以由计算机系统自动地完成而无需人工干预,其中任何确定都是由处理器做出的。此外,本文描述的每个方法的一些实施例可以被实现为存储在有形的,非暂态的存储介质上以形成有形的软件产品的指令集合。

[0065] 图13描绘了用于实现实施例之一的分布式系统1300的简化图。在所示的实施例中,分布式系统1300包括一个或多个客户端计算设备1302、1304、1306和1308,每个客户端计算设备通过一个或多个网络1310执行和操作客户端应用,诸如web浏览器、专有客户端

(例如, Oracle Forms) 等。服务器1312可以经由网络1310与远程客户端计算设备1302、1304、1306和1308通信地耦合。

[0066] 在各种实施例中, 服务器1312可以适于运行由系统的一个或多个组件提供的一个或多个服务或软件应用。在一些实施例中, 这些服务可以作为基于web的服务或云服务或者在软件即服务 (SaaS) 模型下被供应给客户端计算设备1302、1304、1306和/或1308的用户。操作客户端计算设备1302、1304、1306和/或1308的用户进而可以利用一个或多个客户端应用来与服务器1312交互以利用由这些组件提供的服务。

[0067] 在图中描绘的配置中, 系统1300的软件组件1318、1320和1322被示出为在服务器1312上实现。在其它实施例中, 系统1300的一个或多个组件和/或由这些组件提供的服务也可以由客户端计算设备1302、1304、1306和/或1308中的一个或多个来实现。然后, 操作客户端计算设备的用户可以利用一个或多个客户端应用来使用由这些组件提供的服务。这些组件可以用硬件、固件、软件或其组合来实现。应该认识到的是, 各种不同的系统配置是可能的, 这些配置可能与分布式系统1300不同。因此, 图中所示的实施例是用于实现实施例系统的分布式系统的一个示例, 而不旨在是限制性的。

[0068] 客户端计算设备1302、1304、1306和/或1308可以是便携式手持设备(例如, **iPhone®**、蜂窝电话、**iPad®**、计算平板电脑、个人数字助理 (PDA)) 或可穿戴设备(例如, Google眼镜 (Google **Glass®**) 头戴式显示器), 运行软件(诸如Microsoft Windows **Mobile®**) 和/或各种移动操作系统(诸如iOS、Windows Phone、Android、BlackBerry 10、Palm OS等)的, 并且启用互联网、电子邮件、短消息服务 (SMS)、**Blackberry®**或其它通信协议。客户端计算设备可以是通用个人计算机, 例如包括运行各种版本的Microsoft **Windows®**、Apple **Macintosh®**和/或Linux操作系统的个人计算机和/或膝上型计算机。客户端计算设备可以是运行各种可商业获得的 **UNIX®**或类UNIX操作系统(包括但不限于各种GNU/Linux操作系统, 诸如例如Google Chrome OS) 中的任何操作系统的工作站计算机。替代地或附加地, 客户端计算设备1302、1304、1306和1308可以是能够通过(一个或多个)网络1310进行通信的任何其它电子设备, 诸如瘦客户端计算机、启用互联网的游戏系统(例如, 具有或不具有 **Kinect®**手势输入设备的Microsoft Xbox游戏控制台) 和/或个人消息传送设备。

[0069] 虽然示出了具有四个客户端计算设备的示例性分布式系统1300, 但是可以支持任何数量的客户端计算设备。其它设备(诸如具有传感器等的设备等)可以与服务器1312交互。

[0070] 分布式系统1300中的(一个或多个)网络1310是可以使用各种可商业获得的协议中的任何协议来支持数据通信的任何类型的网络, 其中协议包括但不限于TCP/IP (传输控制协议/互联网协议)、SNA (系统网络体系架构)、IPX (互联网分组交换)、AppleTalk等。仅仅作为示例, (一个或多个)网络1310可以是局域网 (LAN), 诸如基于以太网、令牌环等的LAN。(一个或多个)网络1310可以是广域网和互联网。它可以包括虚拟网络, 包括但不限于虚拟专用网络 (VPN)、内联网、外联网、公共交换电话网 (PSTN)、红外网络、无线网络(例如, 依据电气和电子协会 (IEEE) 802.11协议套件、蓝牙 (**Bluetooth®**) 和/或任何其它无线协议的任何协议而操作的网络); 和/或这些网络和/或其它网络的任何组合。

[0071] 服务器1312可以包括一个或多个通用计算机、专用服务器计算机(例如包括PC(个人计算机)服务器、**UNIX®**服务器、中档服务器、大型计算机、机架安装的服务器等)、服务器场、服务器集群或任何其它适当的布置和/或组合。在各种实施例中,服务器1312可以适于运行在前述公开中描述的一个或多个服务或软件应用。例如,服务器1312可以与用于执行以上根据本公开的实施例描述的处理的服务器对应。

[0072] 服务器1312可以运行包括以上讨论的任何操作系统中的操作系统,以及任何可商业获得的服务器操作系统。服务器1312还可以运行各种附加的服务器应用和/或中间层应用中的任何一种,包括HTTP(超文本传输协议)服务器、FTP(文件传输协议)服务器、CGI(公共网关接口)服务器、**JAVA®**服务器、数据库服务器等。示例性数据库服务器包括但不限于可从Oracle、Microsoft、Sybase、IBM(国际商业机器)等商业获得的数据库服务器。

[0073] 在一些实施方式中,服务器1312可以包括一个或多个应用,以分析和整合从客户端计算设备1302、1304、1306和1308的用户接收到的数据馈送和/或事件更新。作为示例,数据馈送和/或事件更新可以包括但不限于,**Twitter®**馈送、**Facebook®**更新或者从一个或多个第三方信息源和连续数据流接收到的实时更新,实时更新可以包括与传感器数据应用、金融价格收报机(financial ticker)、网络性能测量工具(例如,网络监视和流量管理应用)、点击流分析工具、汽车交通监视等相关的实时事件。服务器1312还可以包括一个或多个应用,以经由客户端计算设备1302、1304、1306和1308的一个或多个显示设备来显示数据馈送和/或实时事件。

[0074] 分布式系统1300还可以包括一个或多个数据库1314和1316。数据库1314和1316可以驻留在各种位置。作为示例,数据库1314和1316中的一个或多个可以驻留在服务器1312本地(和/或驻留在服务器1312中)的非暂态存储介质上。可替代地,数据库1314和1316可以远离服务器1312并且经由基于网络的连接或专用的连接与服务器1312进行通信。在一组示例中,数据库1314和1316可以驻留在存储区域网络(SAN)中。类似地,用于执行服务器1312所具有的功能的任何必要文件都可以适当地本地存储在服务器1312上和/或远程存储。在一组实施例中,数据库1314和1316可以包括适于响应SQL格式的命令而存储、更新和检索数据的关系数据库,诸如由Oracle提供的数据库。

[0075] 图14是根据本公开的实施例的可以将由实施例系统的一个或多个组件提供的服务作为云服务提供的系统环境1400的一个或多个组件的简化框图。在所示的实施例中,系统环境1400包括可以由用户使用以与提供云服务的云基础设施系统1402交互的一个或多个客户端计算设备1404、1406和1408。客户端计算设备可以被配置为操作客户端应用,诸如web浏览器、专有客户端应用(例如,Oracle Forms)或某种其它应用,这些应用可以由客户端计算设备的用户用来与云基础设施系统1402交互以使用由云基础设施系统1402提供的服务。

[0076] 应当认识到的是,图中描绘的云基础设施系统1402可以具有与所描绘的组件不同的其它组件。另外,图中所示的实施例仅仅是可以结合一些实施例的云基础设施系统的一个示例。在一些其它实施例中,云基础设施系统1402可以具有比图中所示更多或更少的组件、可以组合两个或更多个组件、或者可以具有组件的不同配置或布置。

[0077] 客户端计算设备1404、1406和1408可以是与上面针对1302、1304、1306和1308所描述的设备类似的设备。

[0078] 虽然示例性系统环境1400被示出具有三个客户端计算设备,但是可以支持任何数量的客户端计算设备。其它设备(诸如具有传感器的设备等)可以与云基础设施系统1402交互。

[0079] (一个或多个)网络1410可以促进客户端1404、1406和1408与云基础设施系统1402之间的数据的通信和交换。每个网络可以是可以使用各种商业上可获得的协议(包括上面针对(一个或多个)网络1310所描述的协议)中的任何一种支持数据通信的任何类型的网络。

[0080] 云基础设施系统1402可以包括一个或多个计算机和/或服务器,其可以包括上面针对服务器1312所描述的计算机和/或服务器。

[0081] 在某些实施例中,由云基础设施系统提供的服务可以包括按需对云基础设施系统的用户可用的许多服务,诸如在线数据存储和备份解决方案、基于Web的电子邮件服务、被托管的办公室(office)套件和文档协作服务、数据库处理、受管理的技术支持服务等。由云基础设施系统提供的服务可以动态地伸缩(scale)以满足其用户的需要。由云基础设施系统提供的服务的具体实例化在本文中被称为“服务实例”。一般而言,从云服务提供商的系统经由通信网络(诸如互联网)对用户可用的任何服务被称为“云服务”。通常,在公共云环境中,构成云服务提供商的系统的服务器和系统与用户自己的本地服务器和系统不同。例如,云服务提供商的系统可以托管应用,并且用户可以经由诸如互联网的通信网络按需订购和使用应用。

[0082] 在一些示例中,计算机网络云基础设施中的服务可以包括对存储装置、被托管的数据库、被托管的web服务器、软件应用或由云供应商向用户提供的其它服务等受保护的计算机网络访问。例如,服务可以包括通过互联网对云上的远程存储装置进行密码保护的访问。作为另一个示例,服务可以包括基于web服务的被托管的关系数据库和脚本语言中间件引擎,以供联网的开发人员私有使用。作为另一个示例,服务可以包括对在云供应商的网站上托管的电子邮件软件应用的访问。

[0083] 在某些实施例中,云基础设施系统1402可以包括以自助服务、基于订阅、弹性可扩展、可靠、高度可用和安全的方式递送给客户的应用、中间件和数据库服务供应的套件。这种云基础设施系统的示例是由本受让方提供的Oracle公共云。

[0084] 在各种实施例中,云基础设施系统1402可以适于自动供给、管理和跟踪客户对云基础设施系统1402供应的服务的订阅。云基础设施系统1402可以经由不同的部署模型来提供云服务。例如,可以依据公共云模型提供服务,其中云基础设施系统1402被销售云服务的组织拥有(例如,被Oracle拥有),并且服务对一般公众或不同行业的企业可用。作为另一个示例,可以依据私有云模型来提供服务,其中云基础设施系统1402仅针对单个组织操作,并且可以为该组织内的一个或多个实体提供服务。还可以依据社区云模型来提供云服务,其中云基础设施系统1402和由云基础设施系统1402提供的服务由相关社区中的若干组织共享。云服务还可以依据混合云模型被提供,该混合云模型是两个或更多个不同模型的组合。

[0085] 在一些实施例中,由云基础设施系统1402提供的服务可以包括在软件即服务(SaaS)类别、平台即服务(PaaS)类别、基础设施即服务(IaaS)类别或包括混合服务的其它服务类别下提供的一个或多个服务。客户经由订阅订单可以订购由云基础设施系统1402提供的一个或多个服务。云基础设施系统1402然后执行处理以提供客户的订阅订单中的服

务。

[0086] 在一些实施例中,由云基础设施系统1402提供的服务可以包括但不限于应用服务、平台服务和基础设施服务。在一些示例中,应用服务可以由云基础设施系统经由SaaS平台提供。SaaS平台可以被配置为提供落入SaaS类别的云服务。例如,SaaS平台可以提供在集成开发和部署平台上构建和递送按需应用套件的能力。SaaS平台可以管理和控制用于提供SaaS服务的底层软件和基础设施。通过利用由SaaS平台提供的服务,客户可以利用在云基础设施系统上执行的应用。客户可以获取应用服务,而无需客户购买单独许可证和支持。可以提供各种不同的SaaS服务。示例包括但不限于为大型组织提供销售绩效管理、企业集成和商务灵活性的解决方案的服务。

[0087] 在一些实施例中,平台服务可以由云基础设施系统经由PaaS平台提供。PaaS平台可以被配置为提供落入PaaS类别的云服务。平台服务的示例可以包括但不限于使组织(诸如Oracle)能够在共享的公共体系架构上整合现有应用以及充分利用平台提供的共享服务来构建新应用的能力的服务。PaaS平台可以管理和控制用于提供PaaS服务的底层软件和基础设施。客户可以获取由云基础架构系统提供的PaaS服务,而无需客户购买单独的许可证和支持。平台服务的示例包括但不限于Oracle Java云服务(JCS)、Oracle数据库云服务(DBCS)等。

[0088] 通过利用由PaaS平台提供的服务,客户可以采用由云基础设施系统支持的编程语言和工具,并且还控制所部署的服务。在一些实施例中,由云基础设施系统提供的平台服务可以包括数据库云服务、中间件云服务(例如,Oracle融合中间件服务)和Java云服务。在一个实施例中,数据库云服务可以支持共享服务部署模型,该模型使得组织能够汇集数据库资源并且以数据库云的形式向客户供应数据库即服务。中间件云服务可以为客户提供开发和部署各种商务应用的平台,并且Java云服务可以为客户提供在云基础设施系统中部署Java应用的平台。

[0089] 各种不同的基础设施服务可以由云基础设施系统中的IaaS平台提供。基础设施服务促进底层计算资源(诸如存储装置、网络和其它基础计算资源)的管理和控制,以供客户利用由SaaS平台和PaaS平台提供的服务。

[0090] 在某些实施例中,云基础设施系统1402还可以包括基础设施资源1430,用于向云基础设施系统的客户提供用于提供各种服务的资源。在一个实施例中,基础设施资源1430可以包括预先集成和优化的硬件(诸如服务器、存储装置和联网资源)的组合,以执行由PaaS平台和SaaS平台提供的服务。

[0091] 在一些实施例中,云基础设施系统1402中的资源可以由多个用户共享并且根据需要动态重新分配。此外,可以将资源分配给在不同时区的用户。例如,云基础设施系统1430可以使在第一时间区中的第一组用户能够在指定的小时数内利用云基础设施系统的资源,并且然后使相同资源能够被重新分配给位于不同时间区的另一组用户,从而使资源的利用率最大化。

[0092] 在某些实施例中,可以提供由云基础设施系统1402的不同组件或模块以及由云基础设施系统1402提供的服务共享的多个内部共享服务1432。这些内部共享服务可以包括但不限于:安全和身份服务、集成服务、企业储存库服务、企业管理器服务、病毒扫描和白名单服务、高可用性、备份和恢复服务、启用云支持的服务、电子邮件服务、通知服务、文件传输

服务等。

[0093] 在某些实施例中,云基础设施系统1402可以提供云基础设施系统中的云服务(例如,SaaS、PaaS和IaaS服务)的综合管理。在一个实施例中,云管理功能可以包括用于供给、管理和跟踪由云基础设施系统1402接收到的客户订阅等的能力。

[0094] 在一个实施例中,如图中所描绘的,云管理功能可以由一个或多个模块(诸如订单管理模块1420、订单编排模块1422、订单供给模块1424、订单管理和监视模块1426,以及身份管理模块1428)提供。这些模块可以包括一个或多个计算机和/或服务器或者使用一个或多个计算机和/或服务器来提供,这些计算机和/或服务器可以是通用计算机、专用服务器计算机、服务器场、服务器集群或任何其它适当的布置/或组合。

[0095] 在示例性操作1434中,使用客户端设备(诸如客户端设备1404、1406或1408)的客户可以通过请求由云基础设施系统1402提供的一个或多个服务并且下订阅由云基础设施系统1402供应的一个或多个服务来的订单来与云基础设施系统1402交互。在某些示例中,客户可以访问云用户界面(UI)、云UI 1412、云UI 1414和/或云UI 1416并经由这些UI下订阅订单。云基础设施系统1402响应于客户下订单而接收到的订单信息可以包括识别客户以及客户想要订阅的云基础设施系统1402供应的一个或多个服务的信息。

[0096] 在客户已经下订单之后,经由云UI 1412、1414和/或1416,接收订单信息。

[0097] 在操作1436处,订单存储在订单数据库1418中。订单数据库1418可以是由云基础设施系统1018操作和与其它系统元件一起操作的若干数据库之一。

[0098] 在操作1438处,订单信息被转发到订单管理模块1420。在一些情况下,订单管理模块1420可以被配置为执行与订单相关的计费和记账功能,诸如验证订单,并且在验证后,预订订单。

[0099] 在操作1440处,将关于订单的信息传送到订单编排模块1422。订单编排模块1422可以利用订单信息为客户下的订单编排服务和资源的供给。在一些情况下,订单编排模块1422可以使用订单供给模块1424的服务来编排资源的供给以支持订阅的服务。

[0100] 在某些实施例中,订单编排模块1422使得能够管理与每个订单相关联的业务流程并应用业务逻辑来确定订单是否应该进行到供给。在操作1442处,在接收到新订阅的订单时,订单编排模块1422向订单供给模块1424发送请求以分配资源并配置履行订阅订单所需的那些资源。订单供给模块1424使得能够为客户订购的服务分配资源。订单供给模块1424提供由云基础设施系统1400提供的云服务和用于供给用于提供所请求的服务的资源的物理实现层之间的抽象层。因此,订单编排模块1422可以与实施细节(诸如服务和资源是否实际上即时供给或预先供给并仅在请求后才分配/指派)分离。

[0101] 在操作1444处,一旦供给了服务和资源,就可以通过云基础设施系统1402的订单供给模块1424向客户端设备1404、1406和/或1408上的客户发送所提供服务的通知。

[0102] 在操作1446处,订单管理和监视模块1426可以管理和跟踪客户的订阅订单。在一些情况下,订单管理和监视模块1426可以被配置为收集订阅订单中的服务的使用统计数据,诸如所使用的存储量、传输的数据量、用户的数量,以及系统运行时间和系统停机时间量。

[0103] 在某些实施例中,云基础设施系统1400可以包括身份管理模块1428。身份管理模块1428可以被配置为提供身份服务,诸如云基础设施系统1400中的访问管理和授权服务。

在一些实施例中,身份管理模块1428可以控制关于希望利用由云基础设施系统1402提供的服务的客户的信息。这样的信息可以包括认证这些客户的身份的信息以及描述这些客户被授权相对于各种系统资源(例如,文件、目录、应用、通信端口、存储器段等)执行哪些动作的信息。身份管理模块1428还可以包括对关于每个客户的描述性信息以及关于如何和由谁来访问和修改该描述性信息的管理。

[0104] 图15图示了其中可以实现各种实施例的示例性计算机系统1500。系统1500可以用于实现上述计算机系统1500中的任何计算机系统。如图所示,计算机系统1500包括经由总线子系统1502与多个外围子系统通信的处理单元1504。这些外围子系统可以包括处理加速单元1506、I/O子系统1508、存储子系统1518和通信子系统1524。存储子系统1518包括有形计算机可读存储介质1522和系统存储器1510。

[0105] 总线子系统1502提供用于让计算机系统1500的各种组件和子系统按意图彼此进行通信的机制。虽然总线子系统1502被示意性地示出为单条总线,但是总线子系统的替代实施例可以利用多条总线。总线子系统1502可以是若干种类型的总线结构中的任何类型,这些类型的总线结构包括存储器总线或存储器控制器、外围总线、以及使用各种总线体系架构中的任何体系架构的局部总线。例如,这种体系架构可以包括工业标准体系架构 (ISA) 总线、微通道体系架构 (MCA) 总线、增强型ISA (EISA) 总线、视频电子标准协会 (VESA) 局部总线和外围组件互连 (PCI) 总线,这些总线可以被实现为按IEEE P1086.1标准制造的Mezzanine总线。

[0106] 可以被实现为一个或多个集成电路(例如,常规微处理器或微控制器)的处理单元1504控制计算机系统1500的操作。一个或多个处理器可以被包括在处理单元1504中。这些处理器可以包括单核处理器或多核处理器。在某些实施例中,处理单元1504可以被实现为一个或多个独立的处理单元1532和/或1534,其中在每个处理单元中包括单核处理器或多核处理器。在其它实施例中,处理单元1504也可以被实现为通过将两个双核处理器集成到单个芯片中形成的四核处理单元。

[0107] 在各种实施例中,处理单元1504可以响应于程序代码执行各种程序并且可以维护多个并发执行的程序或进程。在任何给定的时间,要执行的程序代码中的一些或全部代码可以驻留在(一个或多个)处理器1504中和/或存储子系统1518中。通过适当的编程,(一个或多个)处理器1504可以提供上述各种功能。计算机系统1500可以附加地包括处理加速单元1506,该处理加速单元1506可以包括数字信号处理器 (DSP)、专用处理器等。

[0108] I/O子系统1508可以包括用户接口输入设备和用户接口输出设备。用户接口输入设备可以包括键盘、诸如鼠标或轨迹球的指向设备、结合到显示器中的触摸板或触摸屏、滚动轮、点击轮、拨盘、按钮、开关、小键盘、具有语音命令识别系统的音频输入设备、麦克风以及其它类型的输入设备。用户接口输入设备可以包括例如运动感测和/或手势识别设备,诸如Microsoft **Kinect®**运动传感器,该运动传感器使得用户能够通过使用手势和语音命令的自然用户接口来控制诸如Microsoft **Xbox®**360游戏控制器的输入设备并与该输入设备交互。用户接口输入设备也可以包括眼睛姿势识别设备,诸如从用户检测眼睛活动(例如,当拍摄照片和/或进行菜单选择时的“眨眼”)并且将眼睛姿势变换为进入输入设备(例如,Google **Glass®**)中的输入的Google **Glass®**眨眼检测器。此外,用户接口输入设备可

以包括使得用户能够通过语音命令与语音识别系统(例如,**Siri®**导航器)交互的语音识别感测设备。

[0109] 用户接口输入设备也可以包括但不限于:三维(3D)鼠标、操纵杆或指向棒(pointing stick)、游戏面板和绘图板、以及音频/视觉设备,诸如扬声器、数码相机、数码摄录机、便携式媒体播放器、网络摄像头、图像扫描仪、指纹扫描仪、条形码阅读器3D扫描仪、3D打印机、激光测距仪和视线跟踪设备。此外,用户接口输入设备可以包括例如医学成像输入设备,诸如计算机断层扫描、磁共振成像、正电子发射断层显像、医疗超声设备。用户接口输入设备也可以包括例如音频输入设备,诸如MIDI键盘、数字乐器等。

[0110] 用户接口输出设备可以包括显示子系统、指示灯,或者诸如音频输出设备的非可视显示器等。显示子系统可以是阴极射线管(CRT)、诸如使用液晶显示器(LCD)或等离子显示器的平板设备、投影设备、触摸屏等。一般而言,术语“输出设备”的使用旨在包括用于从计算机系统1500向用户或其它计算机输出信息的所有可能类型的设备和机制。例如,用户接口输出设备可以包括但不限于:可视地传达文本、图形和音频/视频信息的各种显示设备,诸如监视器、打印机、扬声器、耳机、汽车导航系统、绘图仪、语音输出设备以及调制解调器。

[0111] 计算机系统1500可以包括包含软件元件的被示出为当前位于系统存储器1510内的存储子系统1518。系统存储器1510可以存储在处理单元1504上可加载且可执行的程序指令,以及在这些程序的执行期间所生成的数据。

[0112] 取决于计算机系统1500的配置和类型,系统存储器1510可以是易失性的(诸如随机存取存储器(RAM))和/或非易失性的(诸如只读存储器(ROM)、闪存存储器等)。RAM通常包含可被处理单元1504立即访问和/或目前正在被处理单元1504操作和执行的数据和/或程序模块。在一些实施方式中,系统存储器1510可以包括多种不同类型的存储器,例如静态随机存取存储器(SRAM)或动态随机存取存储器(DRAM)。在一些实施方式中,诸如在启动期间,包含有助于在计算机系统1500内的元件之间传送信息的基本例程的基本输入/输出系统(BIOS)通常可以被存储在ROM中。作为示例但不是限制,系统存储器1510也示出了可以包括客户端应用、Web浏览器、中间层应用、关系数据库管理系统(RDBMS)等的应用程序1512,程序数据1514、以及操作系统1516。作为示例,操作系统1516可以包括各种版本的Microsoft **Windows®**, Apple **Macintosh®** 和/或Linux操作系统、各种可商业获得的 **UNIX®** 或类UNIX操作系统(包括但不限于各种GNU/Linux操作系统、Google **Chrome®** OS等)和/或移动操作系统(诸如iOS、**Windows®** Phone、**Android®** OS、**BlackBerry®** 100S 和 **Palm®** OS操作系统)。

[0113] 存储子系统1518也可以提供用于存储提供一些实施例的功能的基本编程和数据构造的有形计算机可读存储介质。当被处理器执行时提供上述功能的软件(程序、代码模块、指令)可以被存储在存储子系统1518中。这些软件模块或指令可以被处理单元1504执行。存储子系统1518也可以提供用于存储根据一些实施例使用的数据的储存库。

[0114] 存储子系统1500也可以包括可以进一步连接到计算机可读存储介质1522的计算机可读存储介质读取器1520。与系统存储器1510一起并且可选地与系统存储器1510相结合,计算机可读存储介质1522可以全面地表示用于临时和/或更持久地包含、存储、发送和

检索计算机可读信息的远程、本地、固定和/或可移动的存储设备加存储介质。

[0115] 包含代码或代码的一部分的计算机可读存储介质1522也可以包括任何适当的介质,包括存储介质和通信介质,诸如但不限于:以用于存储和/或传输信息的任何方法或技术实现的易失性和非易失性、可移动和不可移动的介质。这可以包括有形计算机可读存储介质,诸如RAM、ROM、电可擦除可编程ROM(EEPROM)、闪存存储器或其它存储器技术、CD-ROM、数字多功能盘(DVD)或其它光学储存装置、磁带盒、磁带、磁盘存储装置或其它磁存储设备、或者其它有形计算机可读介质。这也可以包括非有形计算机可读介质,诸如数据信号、数据传输、或者可以用于传输期望信息并且可以被计算系统1500访问的任何其它介质。

[0116] 作为示例,计算机可读存储介质1522可以包括从不可移动的非易失性磁介质读取或写入到不可移动的非易失性磁介质的硬盘驱动器、从可移动的非易失性磁盘读取或写入到可移动的非易失性磁盘的磁盘驱动器、以及从可移动的非易失性光盘(诸如CD ROM、DVD和蓝光(**Blu-Ray®**)盘或其它光学介质)读取或写入到可移动的非易失性光盘的光盘驱动器。计算机可读存储介质1522可以包括但不限于:**Zip®**驱动器、闪存卡、通用串行总线(USB)闪存驱动器、安全数字(SD)卡、DVD盘、数字音频带等。计算机可读存储介质1522也可以包括:基于非易失性存储器的固态驱动器(SSD)(诸如基于闪存存储器的SSD、企业闪存驱动器、固态ROM等)、基于易失性存储器的SSD(诸如固态RAM、动态RAM、静态RAM,基于DRAM的SSD、磁阻RAM(MRAM) SSD)、以及使用基于DRAM的SSD和基于闪存存储器的SSD的混合SSD。盘驱动器及其关联的计算机可读介质可以为计算机系统1500提供计算机可读指令、数据结构、程序模块以及其它数据的非易失性存储。

[0117] 通信子系统1524提供到其它计算机系统和网络的接口。通信子系统1524用作用于从其它系统接收来自计算机系统1500的数据和从计算机系统1500向其它系统发送数据的接口。例如,通信子系统1524可以使得计算机系统1500能够经由互联网连接到一个或多个设备。在一些实施例中,通信子系统1524可以包括用于访问无线语音和/或数据网络的射频(RF)收发器组件(例如,使用蜂窝电话技术、先进数据网络技术,诸如3G、4G或EDGE(用于全球演进的增强型数据速率)、WiFi(IEEE 802.11系列标准)或其它移动通信技术、或者其任何组合)、全球定位系统(GPS)接收器组件和/或其它组件。在一些实施例中,作为无线接口的附加或者替代,通信子系统1524可以提供有线网络连接(例如,以太网)。

[0118] 在一些实施例中,通信子系统1524也可以代表可以使用计算机系统1500的一个或多个用户来接收以结构化和/或非结构化数据馈送1526、事件流1528、事件更新1530等形式的输入通信。

[0119] 作为示例,通信子系统1524可以被配置为实时地从社交网络和/或其它通信服务的用户接收数据馈送1526,诸如**Twitter®**馈送、**Facebook®**更新、诸如丰富站点摘要(RSS)馈送的web馈送和/或来自一个或多个第三方信息源的实时更新。

[0120] 此外,通信子系统1524也可以被配置为接收以连续数据流形式的数据,该数据可以包括本质上可以是连续的或无界的没有明确终止的实时事件的事件流1528和/或事件更新1530。生成连续数据的应用的示例可以包括例如传感器数据应用、金融价格收报机、网络性能测量工具(例如,网络监视和流量管理应用)、点击流分析工具、汽车流量监视等。

[0121] 通信子系统1524也可以被配置为向一个或多个数据库输出结构化和/或非结构化

的数据馈送1526、事件流1528、事件更新1530等,这一个或多个数据库可以与耦合到计算机系统1500的一个或多个流式传输数据源计算机进行通信。

[0122] 计算机系统1500可以是各种类型中的一种类型,包括手持便携式设备(例如,**iPhone®**蜂窝电话、**iPad®**计算平板电脑、PDA)、可穿戴设备(例如,Google **Glass®**头戴式显示器)、PC、工作站、大型机、信息站(kiosk)、服务器机架、或任何其它数据处理系统。

[0123] 由于计算机和网络的不断变化的本质,在图中绘出的计算机系统1500的描述仅旨在作为具体的示例。具有比图中绘出的系统更多或更少组件的许多其它配置是可能的。例如,也可以使用自定义硬件,和/或可以在硬件、固件、软件(包括小程序应用(applets))或其组合中实现特定的元件。另外,可以采用到诸如网络输入/输出设备的其它计算设备的连接。基于本文提供的公开和教导,可以使用其它方式和/或方法来实现各种实施例。

[0124] 在前面的描述中,出于解释的目的,阐述了许多具体细节以便提供对各种实施例的透彻理解。但是,将显而易见的是,可以在没有这些具体细节中的一些的情况下实践这些实施例。在其它情况下,以框图形式示出了众所周知的结构和设备。

[0125] 前面的描述仅提供示例性实施例,并且不旨在限制本公开的范围、适用性或配置。更确切地说,示例性实施例的前述描述提供用于实现至少一个实施例的使能描述。应当理解的是,在不脱离所附权利要求书所阐述的一些实施例的精神和范围的情况下,可以对元件的功能和布置进行各种改变。

[0126] 在前面的描述中给出了具体细节以提供对实施例的透彻理解。但是,可以在没有这些具体细节的情况下实践实施例。例如,电路、系统、网络、处理和其它组件可能已经被示为框图形式的组件,以便不以不必要的细节使实施例晦涩难懂。在其它情况下,可以已经示出了众所周知的电路、处理、算法、结构和技术而没有不必要的细节,以避免使实施例晦涩难懂。

[0127] 而且,要注意的是,各个实施例可以已经被描述为被描绘为流程图、流图、数据流图、结构图或框图的处理。虽然流程图可以已经将操作描述为顺序处理,但是许多操作可以并行或并发地执行。此外,可以重新布置操作的次序。处理在器操作完成时终止,但可以具有图中未包括的附加步骤。处理可以与方法、函数、过程、子例程、子程序等对应。当处理与函数对应时,其终止可以与该函数返回到调用函数或主函数对应。

[0128] 术语“计算机可读介质”包括但不限于便携式或固定存储设备、光学存储设备、无线信道以及能够存储、包含或携带(一个或多个)指令和/或数据的各种其它介质。代码段(code segment)或机器可执行指令可以表示过程、函数、子程序、程序、例程、子例程、模块、软件包、类、或者指令、数据结构或程序语句的任意组合。代码段可以通过传递和/或接收信息、数据、自变量、参数或存储器内容而耦合到另一个代码段或硬件电路。信息、自变量、参数、数据等可以经由任何合适的方式(包括存储器共享、消息传递、令牌传递、网络传输等)进行传递、转发或传输。

[0129] 此外,实施例可以通过硬件、软件、固件、中间件、微代码、硬件描述语言或其任何组合来实现。当以软件、固件、中间件或微代码实现时,可以将执行必要任务的程序代码或代码段存储在机器可读介质中。(一个或多个)处理器可以执行必要的任务。

[0130] 在前述说明书中,参考其具体实施例描述了各种实施例的方面,但并非所有实施例都限于此。上述实施例的各个特征和方面可以被单独使用或联合使用。另外,在不脱离本

说明书的更广泛精神和范围的情况下,实施例可以在除本文所述的那些环境和应用之外的任何数量的环境和应用中被使用。相应地,本说明书和附图应当被认为是说明性的而不是限制性的。

[0131] 此外,出于说明的目的,以特定次序描述了方法。应该认识到的是,在替代示例中,可以以与所描述的次序不同的次序执行方法。还应该认识到的是,上述方法可以由硬件组件执行,或者可以以机器可执行指令的序列被实施,该机器可执行指令的序列可以用于使得机器(诸如编程有指令的通用或专用处理器或逻辑电路)执行方法。这些机器可执行指令可以被存储在一个或多个机器可读介质上,诸如CD-ROM或其它类型的光盘、软盘、ROM、RAM、EPROM、EEPROM、磁卡或光卡、闪存、或适用于存储电子指令的其它类型的机器可读介质。替代地,可以通过硬件和软件的组合来执行方法。

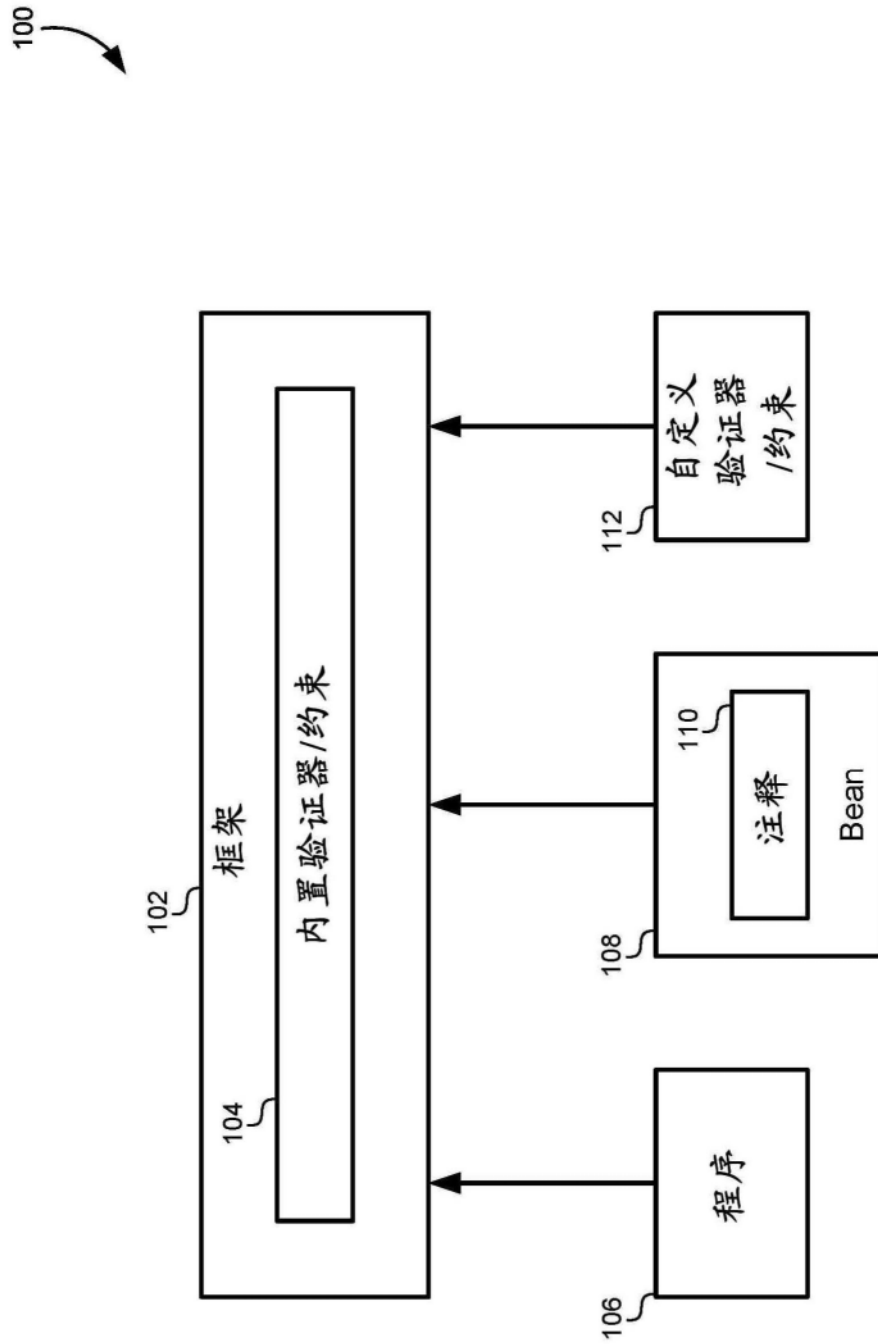


图1

```
public class KeyValue {  
    @NotBlank(message = "key is blank")  
    private String key;  
    private String value;  
  
    public KeyValue(String key, String value){  
        this.key = key;  
        this.value = value;  
    }  
  
    public String getKey() {  
        return key;  
    }  
  
    public void setKey(String key){  
        this.key = key;  
    }  
  
    public String getValue() {  
        return value;  
    }  
  
    public void setValue() {  
        this.value = value;  
    }  
  
    @Override  
    public String toString(){  
        return "KeyValue{" + "key='" + key + '\'' + ", value='" + value + '\'' +  
    }  
}
```

108

110

图2A

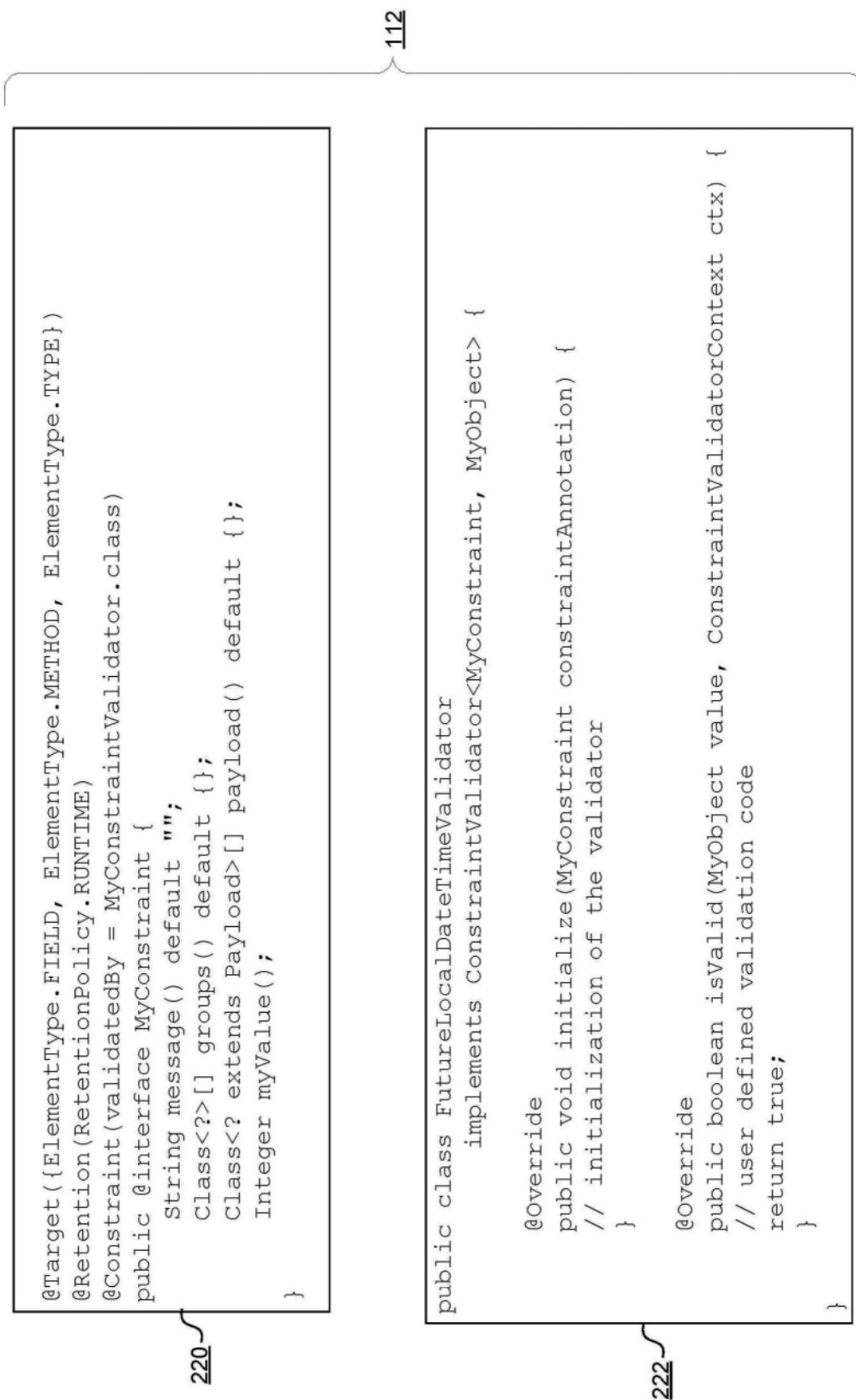


图2B

```
public class Main {  
    private static Validator VALIDATOR;  
  
    static {  
        VALIDATOR = Validation.byDefaultProvider()  
            .configure()  
            .buildValidatorFactory()  
            .getValidator();  
    }  
  
    public static main (String... args) {  
        KeyValue keyValue = new KeyValue("key", "value");  
        validate(keyValue);  
    }  
  
    private static void validate (Object... toValidate) {  
        Arrays.stream(toValidate)  
            .forEach(object -> {  
                System.out.println("\n" + object);  
                Set<ConstraintViolation<Object>> constraints = VALIDATOR.validate(object);  
                if (constraints.isEmpty()) {  
                    System.out.println("No constraints were violated");  
                } else {  
                    System.out.println("Violated constraints: " + constraints);  
                }  
            })  
    }  
}
```

106

302

304

306

308

图3

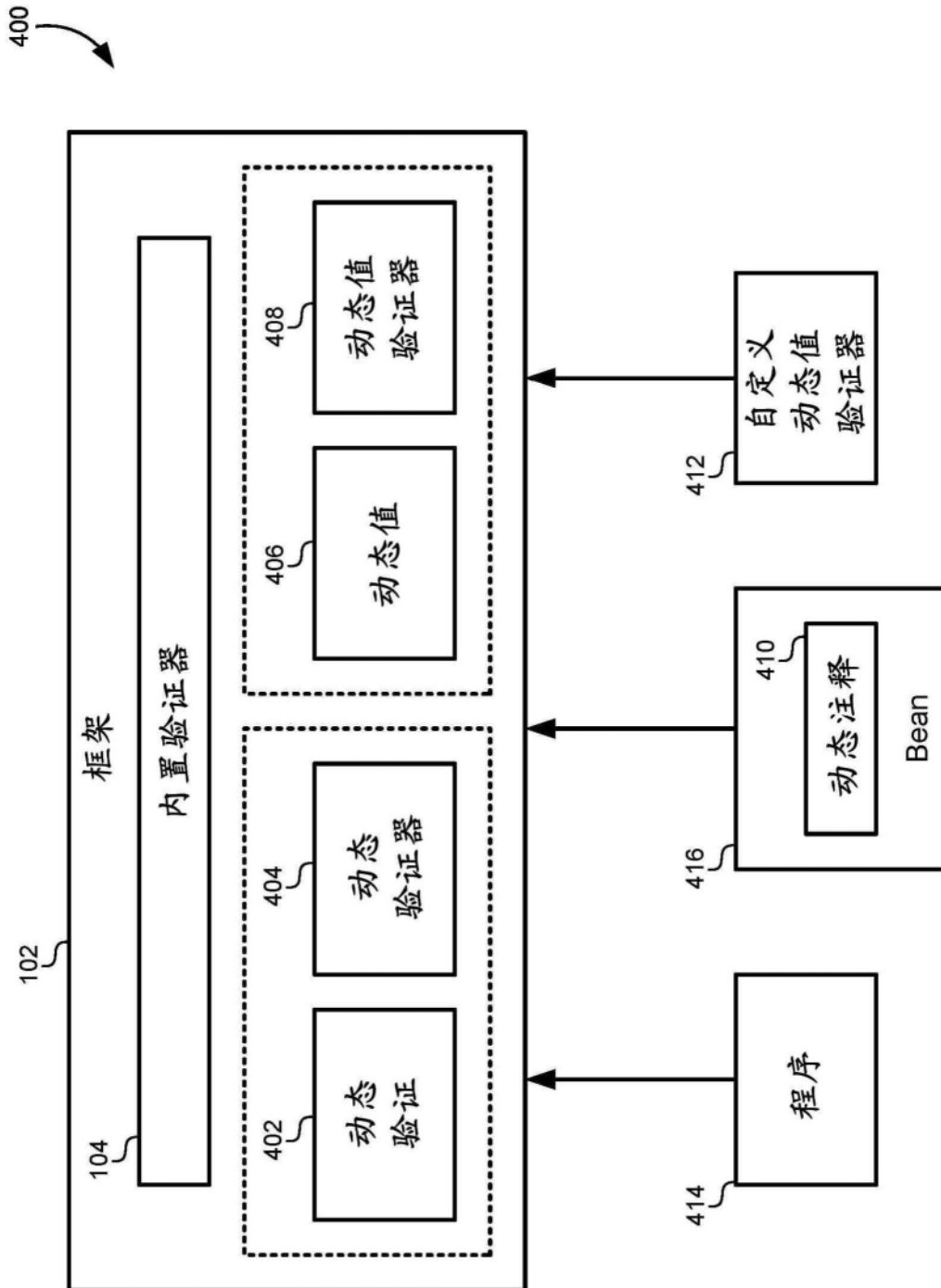


图4

DynamicValidation (动态验证)

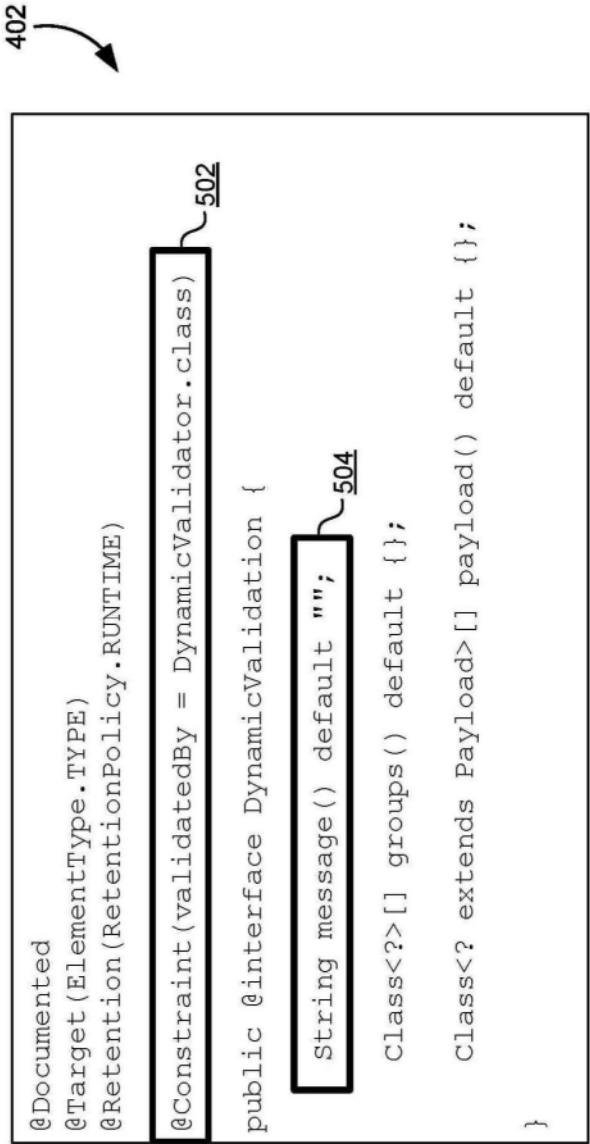


图5

416

410

```
@DynamicValidation
public class KeyValue {
    private String key;
    private String value;

    public KeyValue(String key, String value) {
        this.key = key;
        this.value = value;
    }

    public String getKey() {
        return key;
    }

    public void setKey(String key) {
        this.key = key;
    }
    .
    .
    .
}
```

图6

DynamicValue (动态值)

```
@Documented
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Repeatable(DynamicValues.class)
public @interface DynamicValue {

    String attributeName(); 702
    String valuePattern(); 704

}
```

406



图7

DynamicValueValidator (动态值验证器)

```
public abstract class DynamicValueValidator<E> {  
    private static Validator VALIDATOR;  
  
    static {  
        VALIDATOR = Validation.byDefaultProvider()  
            .configure()  
            .buildValidatorFactory()  
            .getValidator();  
    }  
  
    public abstract boolean isValid(E value, ConstraintValidatorContext ctx);  
  
    protected final boolean validateCustomConstraints(Object toValidate,  
        ConstraintValidatorContext ctx) {  
        Set<ConstraintViolation<Object>> violations = VALIDATOR.validate(toValidate);  
        if (!violations.isEmpty()) {  
            useConstraint(violations.iterator().next().getMessageTemplate(), ctx);  
            return false;  
        }  
        return true;  
    }  
  
    protected void useConstraint(String template,  
        ConstraintValidatorContext ctx) {  
        ctx.disableDefaultConstraintViolation();  
        ctx.buildConstraintViolationWithTemplate(template).addConstraintViolation();  
    }  
}
```

408

802

804

806

图8

```

public class DynamicValidator implements ConstraintValidator<DynamicValidation, Object> {

    private static Validator VALIDATOR;
    private static List<DynamicValueValidator> dynamicValidators;

    static {
        VALIDATOR = Validation.byDefaultProvider()
            .configure()
            .buildValidatorFactory()
            .getValidator();
    }

    public DynamicValidator() {
        synchronized (DynamicValueValidator.class) {
            if (dynamicValidators == null) {
                Reflections reflections = Reflections(DynamicValidator.class.getClassLoader());
                Set<Class<? extends DynamicValueValidator>> allValidatorClasses =
                    reflections.getSubTypesOf(DynamicValueValidator.class);
                dynamicValidators = allValidatorClasses.stream()
                    .filter(c -> !c.isInterface() && !Modifier.isAbstract(c.getModifiers()))
                    .filter(c -> c.isAnnotationPresent(DynamicValue.class))
                    .filter(c -> Stream.of(c.getConstructors()).anyMatch(cnst ->
                        cnst.getParameterCount() == 0))
                    .map(c -> {
                        try {
                            return c.newInstance();
                        } catch (InstantiationException | IllegalAccessException e) {
                            return null;
                        }
                    })
                    .filter(Objects::nonNull)
                    .collect(Collectors.toList());
            }
        }
    }
}

```

404a

902

图9A

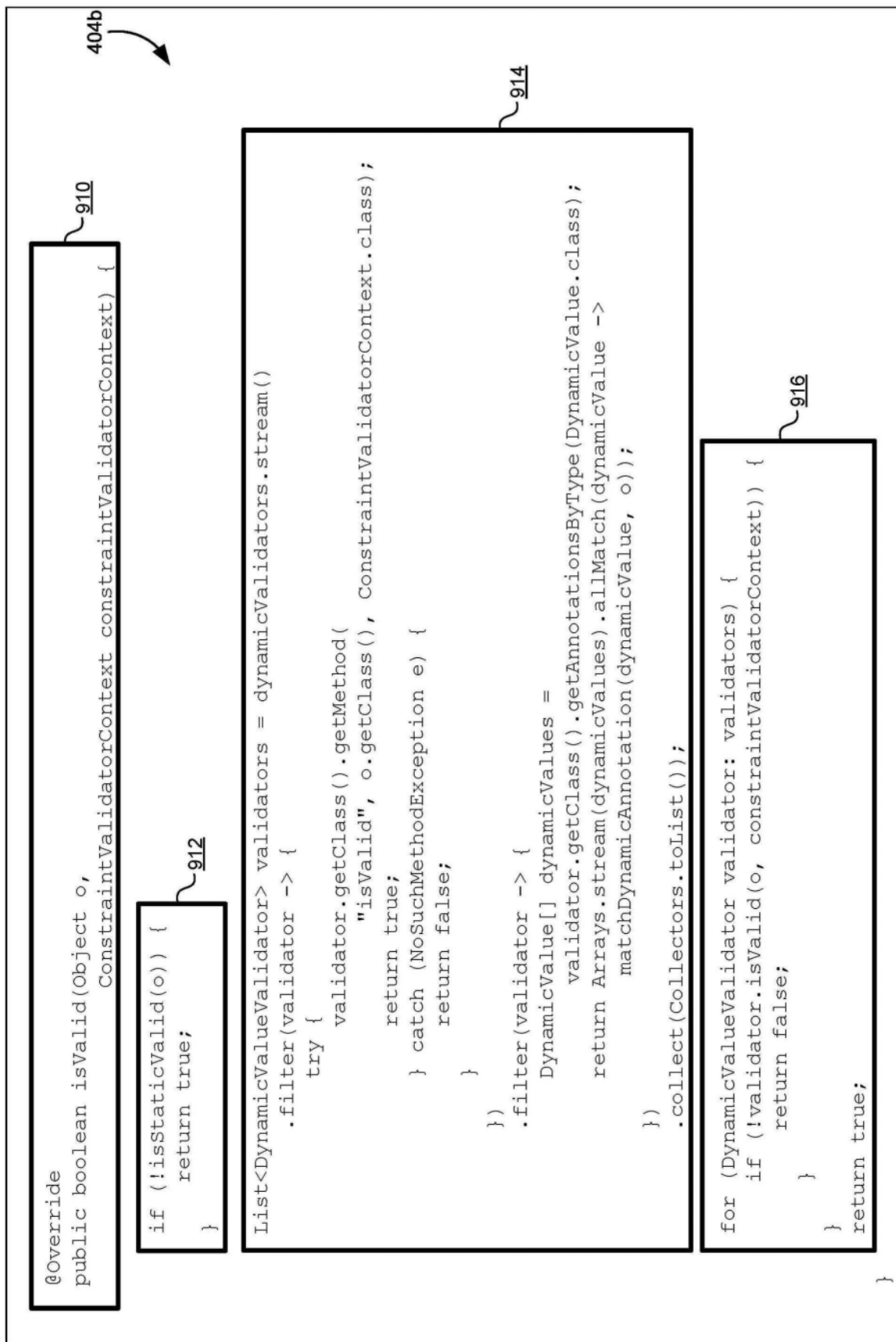


图9B

```

private boolean isStaticValid(Object o) { 920
    Set<String> propertyNames = new HashSet<>();
    Class<?> clazz = o.getClass();
    while (clazz != null) {
        propertyNames.addAll(Arrays.stream(clazz.getDeclaredFields())
            .map(Field::getName)
            .collect(Collectors.toSet())
        );
        clazz = clazz.getSuperclass();
    }
    return propertyNames.stream().allMatch(p -> VALIDATOR.validateProperty(o, p).isEmpty());
}

private boolean matchDynamicAnnotation(DynamicValue dynamicValue, Object o) { 922
    Class<?> clazz = o.getClass();
    while (clazz != null) {
        try {
            Field f = clazz.getDeclaredField(dynamicValue.attributeName());
            f.setAccessible(true);
            Object propertyValue = f.get(o);
            return propertyValue != null
                && propertyValue.toString().matches(dynamicValue.valuePattern());
        } catch (NoSuchFieldException ignored) {
        } catch (IllegalAccessException e) {
            return false;
        }
        clazz = clazz.getSuperclass();
    }
    return false;
}
}

```

404c

图9C

自定义动态值验证器

412

```
@DynamicValue(attributeName = "key", valuePattern = ".*phone\\.number.*")
public class PhoneNumberValidator extends DynamicValueValidator<KeyValue> {

    public boolean isValid(KeyValue value,
        ConstraintValidatorContext ctx) {
        PhoneNumber number = new PhoneNumber(value.getValue());
        return validateCustomConstraints(number, ctx);
    }

    private static class PhoneNumber {

        @Pattern(regexp = "^[+]?[0-9]{1,15}$", message = "value.phone.number")
        private String value;

        PhoneNumber(String value) {
            this.value = value;
        }

        public String getValue() {
            return value;
        }
    }
}
```

图10A

自定义动态值验证器

412

```
@DynamicValue(attributeName = "key1", valuePattern = ".*phone\\.\\.number.*")
@DynamicValue(attributeName = "key2", valuePattern = "CZ")

public class PhoneNumberValidator extends DynamicValueValidator<TwoKeyValue> {

    public boolean isValid(TwoKeyValue value,
        ConstraintValidatorContext ctx) {
        PhoneNumber number = new PhoneNumber(value.getValue());
        return validateCustomConstraints(number, ctx);
    }

    private static class PhoneNumber {

        @Pattern(regexp = "^[+]?[0-9]{1,15}$", message = "value.phone.number")
        private String value;

        PhoneNumber(String value) {
            this.value = value;
        }

        public String getValue() {
            return value;
        }
    }
}
```

图10B

414

```
public class Main {  
  
    private static Validator VALIDATOR;  
  
    static {  
        VALIDATOR = Validation.byDefaultProvider()  
            .configure()  
            .buildValidatorFactory()  
            .getValidator();  
    }  
  
    public static main (String... args) {  
  
        KeyVal keyVal = new KeyVal("phone", "+42067078047");  
        validate(keyVal);  
    }  
    .  
    .  
    .  
}
```

图11

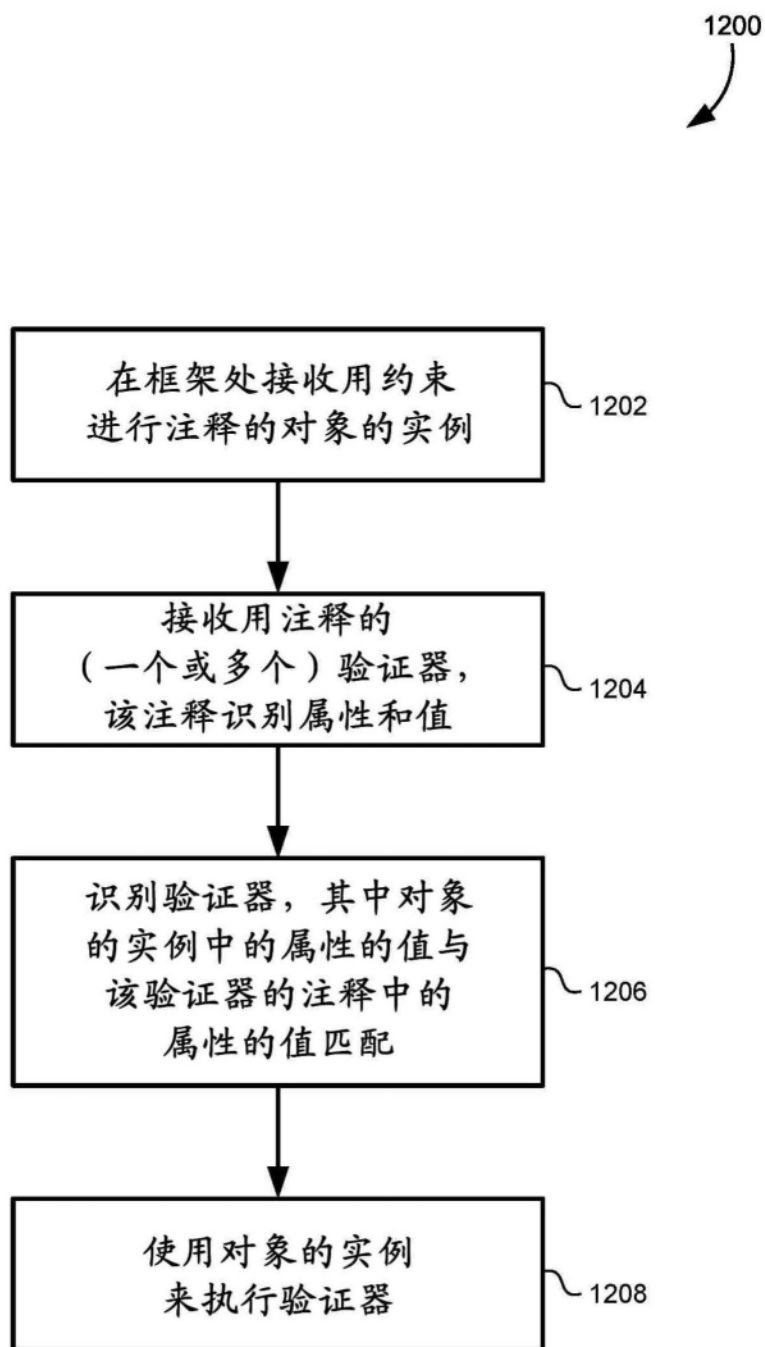


图12

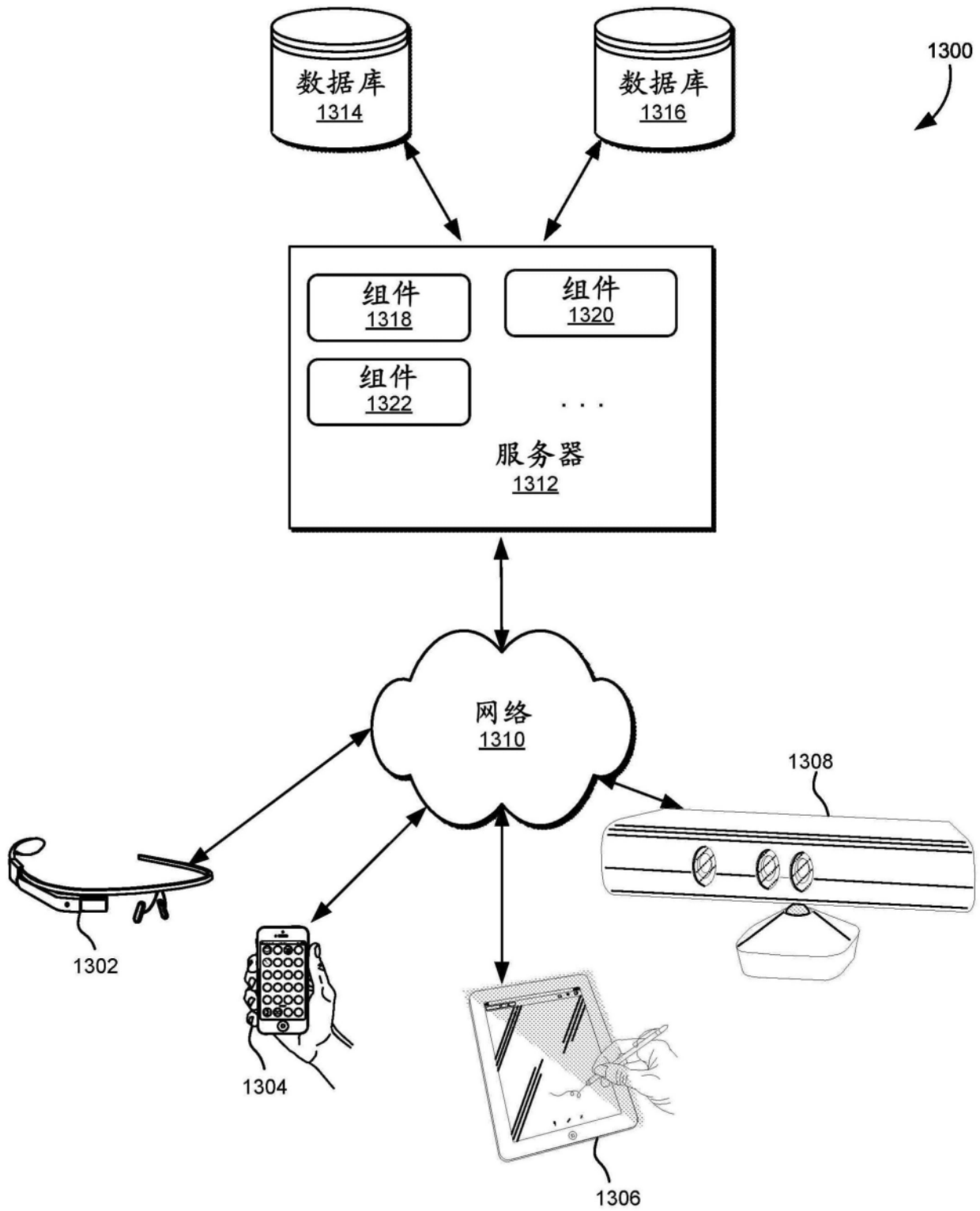


图13

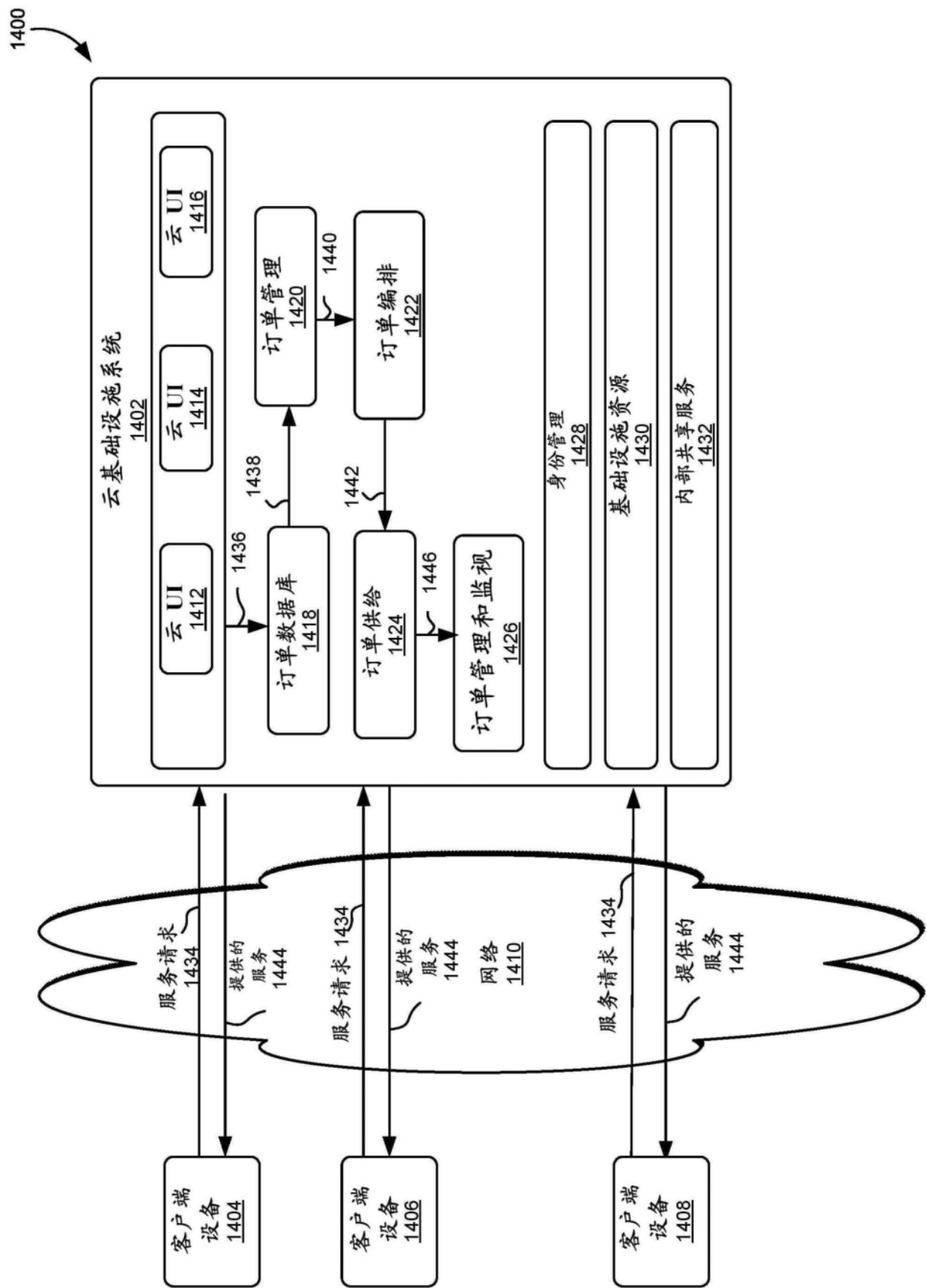


图14

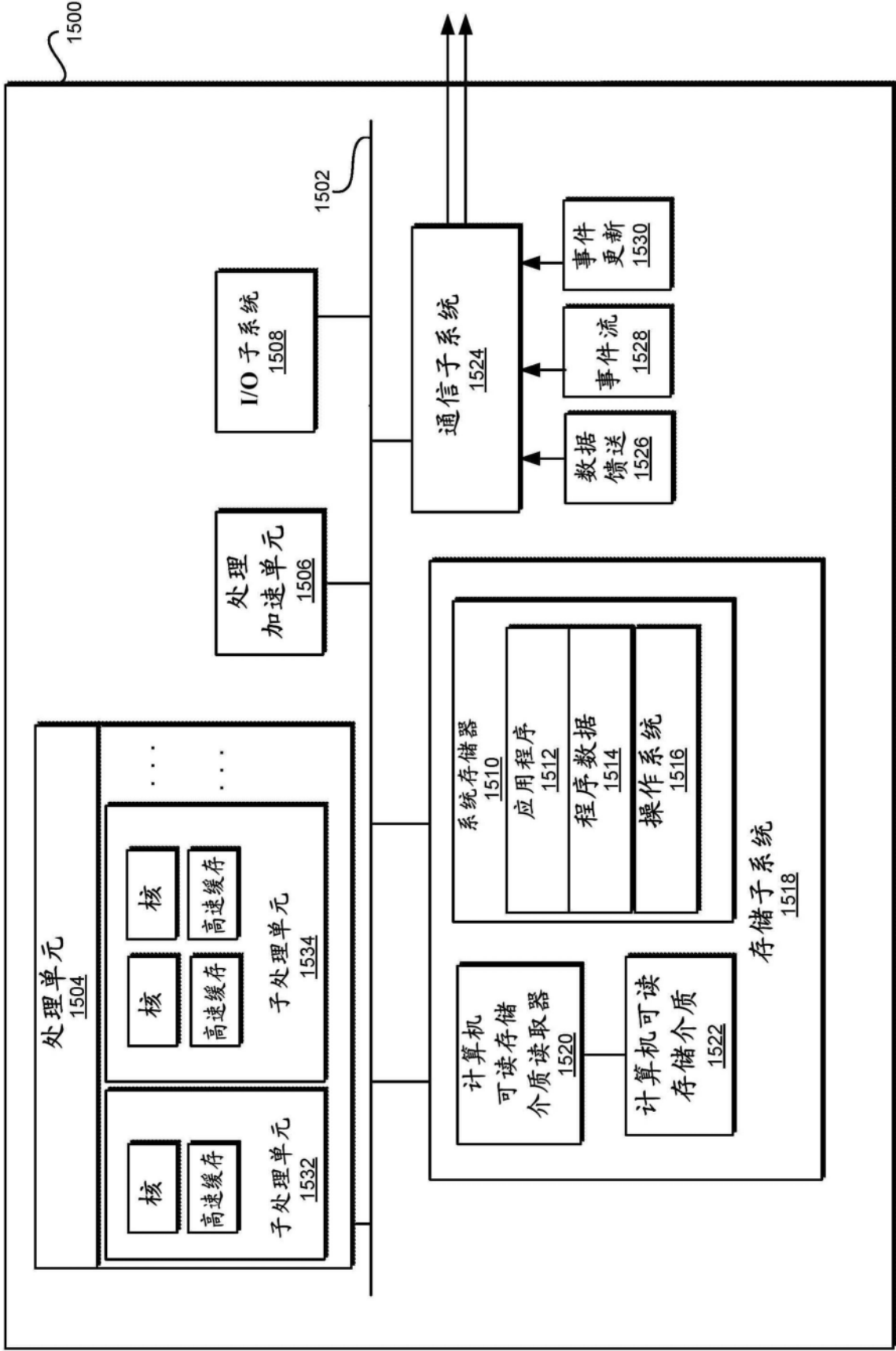


图15