РОССИЙСКАЯ ФЕДЕРАЦИЯ

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ

(19) **RU** (11) **2 520 344**(13) **C2**

(51) МПК
G06F 9/30 (2006.01)

(12) ОПИСАНИЕ ИЗОБРЕТЕНИЯ К ПАТЕНТУ

(21)(22) Заявка: **2011114807/08, 30.09.2009**

(24) Дата начала отсчета срока действия патента:
30.09.2009

Приоритет(ы):
(30) Конвенционный приоритет:
15.10.2008 US 12/251,497

(43) Дата публикации заявки: **20.10.2012** Бюл. № **29**

(45) Опубликовано: **20.06.2014** Бюл. № **17**

(56) Список документов, цитированных в отчете о поиске: **US 2002/0062354 A1, 23.05.2002. US 2004/0210865 A1, 21.10.2004. US 2005/0246697 A1, 03.11.2005. US 2004/0186862 A1, 23.09.2004. RU 2287181 C2, 10.11.2006**

(85) Дата начала рассмотрения заявки PCT на национальной фазе: **14.04.2011**

(86) Заявка PCT:
US 2009/059125 (30.09.2009)

(87) Публикация заявки PCT:
WO 2010/045027 (22.04.2010)

Адрес для переписки:
129090, Москва, ул.Б.Спасская, 25, строение 3, ООО "Юридическая фирма Городисский и Партнеры", пат.пов. А.В.Миц, рег.N 364

(72) Автор(ы):
ХЕРРИНГ Натан (US),
РАЙТОН Дэвид К. (US)

(73) Патентообладатель(и):
МАЙКРОСОФТ КОРПОРЕЙШН (US)

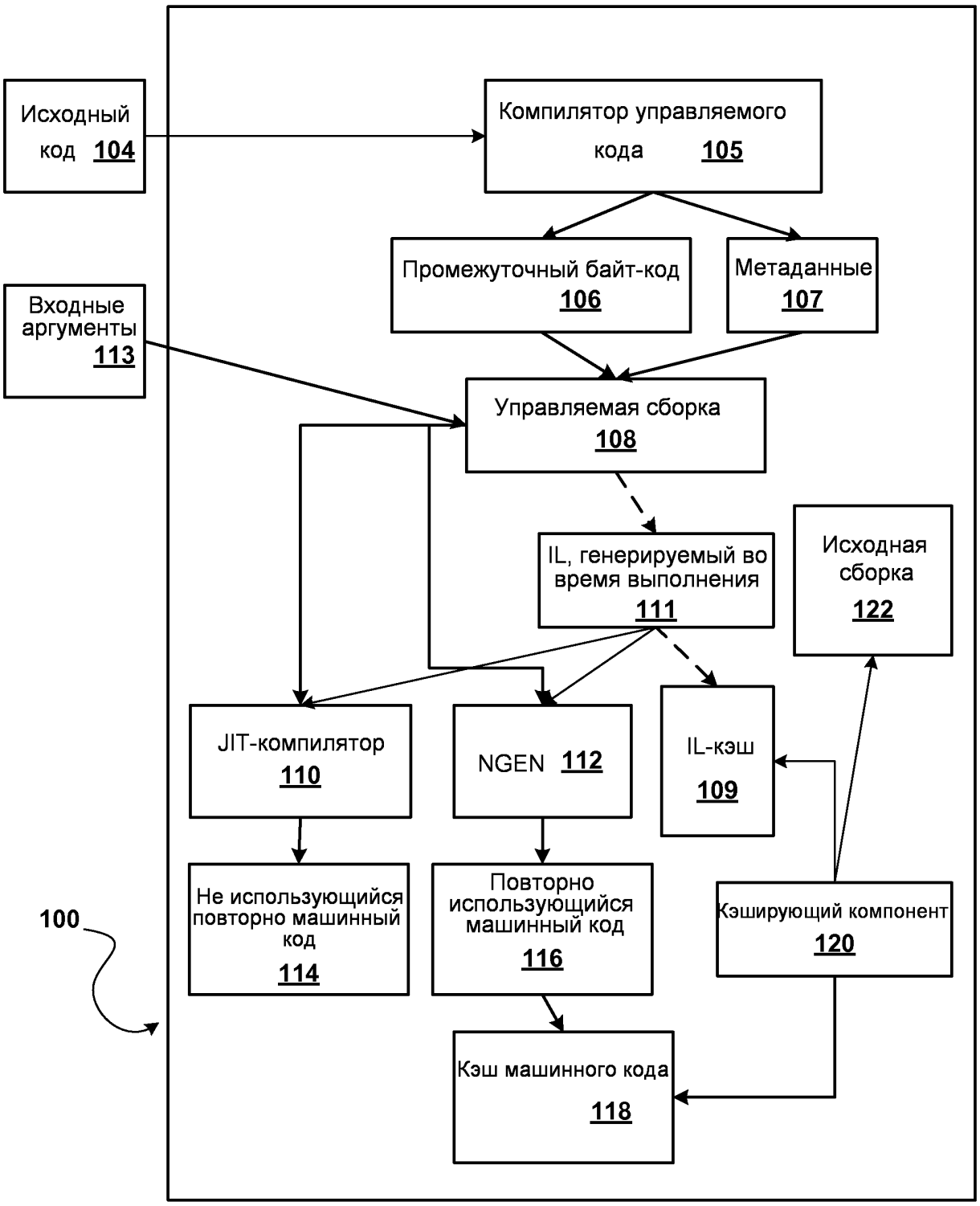
(54) КЭШИРОВАНИЕ ГЕНЕРИРУЕМОГО ВО ВРЕМЯ ВЫПОЛНЕНИЯ КОДА

(57) Реферат:

Изобретение относится к области генерируемого во время выполнения кода. Техническим результатом является кэширование генерируемого во время выполнения кода. Идентифицируется программный объект, который генерирует код, но не изменяет глобальное состояние. Коду, произведенному идентифицированным программным объектом, может быть назначен идентификатор, и код кэшируется в первый раз, когда он выполняется. Последующие выполнения программного объекта могут пропускать генерацию кода и/или трансляцию сгенерированного кода в собственный двоичный код. Генерируемый во время выполнения код и собственный двоичный код могут быть кэшированы в кэш масштаба машины или могут быть добавлены к метаданным сборки, сгенерированной из исходного кода программного объекта. 3 н. и 12 з.п. ф-лы, 6 ил.

RU 2520344 C2

RU 2520344 C2



ФИГ. 1

RU 2520344 C2

RUSSIAN FEDERATION

(19) **RU** (11) **2 520 344** (13) **C2**

(51) Int. Cl.
G06F 9/30 (2006.01)

FEDERAL SERVICE
FOR INTELLECTUAL PROPERTY

(12) ABSTRACT OF INVENTION

(21)(22) Application: **2011114807/08, 30.09.2009**

(24) Effective date for property rights:
30.09.2009

Priority:
(30) Convention priority:
15.10.2008 US 12/251,497

(43) Application published: **20.10.2012** Bull. № **29**

(45) Date of publication: **20.06.2014** Bull. № **17**

(85) Commencement of national phase: **14.04.2011**

(86) PCT application:
US 2009/059125 (30.09.2009)

(87) PCT publication:
WO 2010/045027 (22.04.2010)

Mail address:
129090, Moskva, ul.B.Spasskaja, 25, stroenie 3, OOO "Juridicheskaja firma Gorodisskij i Partnery", pat.pov. A.V.Mits, reg.N 364

(72) Inventor(s):
KhERRING Natan (US),
RAJTON Dehvid K. (US)

(73) Proprietor(s):
MAJKROSOFT KORPOREJShN (US)

RU 2520344 C2

(54) CACHING RUNTIME GENERATED CODE

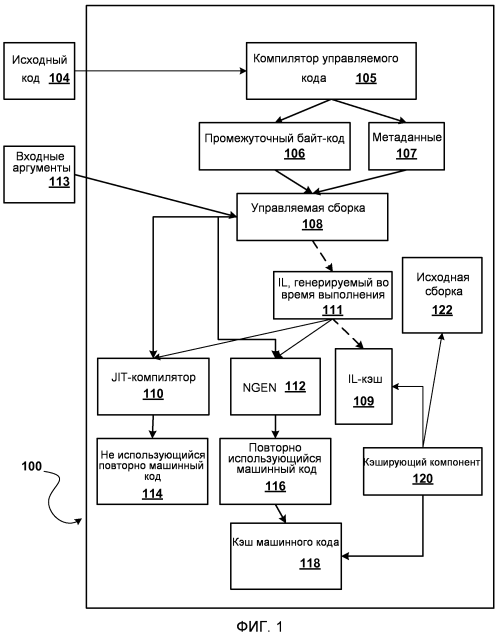
(57) Abstract:

FIELD: physics, computer engineering.

SUBSTANCE: invention relates to runtime generated code. A program entity that generates code but does not alter global state is identified. Code produced by the identified program entity can be assigned an identifier and cached the first time it is executed. Subsequent executions of the program entity can allow generation of the code and/or translation of the generated code into native binary code. The runtime generated code and native binary code can be cached in a machine-wide cache, or can be added to the metadata of the assembly generated from the source code of the program entity.

EFFECT: caching runtime generated code.

15 cl, 6 dwg



ФИГ. 1

Уровень техники

Традиционно компиляторы программного обеспечения производят собственный код, т.е. двоичный код, который является зависимым от машины, на которой этот код будет запускаться. Собственный (неуправляемый) код, производимый традиционными компиляторами, типично включает в себя все, что требует система для запуска этого кода, и небольшое дополнение. В противоположность, в средах виртуальных машин исходный код компилируется в промежуточное представление байт-кода, которое не является зависимым от любой конкретной машины. В дополнение выходной результат компилятора управляемого кода типично включает в себя значительно больше информации, чем только двоичный код. Дополнительная информация описывает характеристики двоичного кода и иногда называется метаданными: общее обозначение для данных, которые описывают другие данные, где в этом случае описываемые данные являются двоичным кодом. Контейнер, который содержит в себе промежуточный байт-код и метаданные, иногда называется управляемой сборкой. Эти контейнеры также могут быть именованы файлами классов, Java-архивами или Java-модулями. Термин "сборка" или "управляемая сборка", использующийся в настоящей заявке, ссылается на любой такой контейнер байт-кода и метаданных.

В неуправляемой среде во время выполнения предварительно существующие собственные команды загружаются в память и выполняются. В управляемой среде во время выполнения управляемая сборка может быть скомпилирована или транслирована в собственные двоичные инструкции сразу перед выполнением. Т.е. управляемая сборка может быть загружена в память и откомпилирована динамичным (JIT) или динамическим компилятором в машинно-зависимые и зависимые от рабочей среды инструкции, которые затем выполняются. Тогда как фаза компиляции/трансляции управляемой среды включает дополнительную обработку, которая может воздействовать на производительность, с одной стороны, JIT/динамический компилятор может быть в состоянии сделать зависимые от среды оптимизации, преимуществом которых не может воспользоваться неуправляемая среда. Например, если JIT/динамический компилятор владеет информацией о том, что код, который исполняется, запускается с полным доверием, он может пропустить некоторые требующие больших затрат проверки системы защиты, которые не могут быть пропущены в неуправляемой среде.

В управляемой среде вместо компиляции управляемой сборки, перед тем как она будет выполняться, исходный код или промежуточный байт-код может быть скомпилирован в собственное двоичное представление оптимизирующим компилятором и сохранен (кэширован), перед тем как быть запущенным на машине, выполняющей запрошенную программу. Например, генератор образов в собственном коде, такой как NGEN, может производить собственный двоичный образ для среды, конвертируя промежуточный байт-код в собственные машинные инструкции в двоичном файле, перед тем как запрос принимается для выполнения сборки. Таким образом, ранее существующий исполняемый код может быть загружен и запущен без второй фазы компиляции/трансляции способом, подобным способу традиционной среды собственного кода. Хотя кэширование предназначается для того, чтобы сделать выполнение сборки быстрее, удаляя процесс JIT/динамической компиляции/трансляции во время выполнения, производительность может не улучшиться, так как оптимизации, зависящие от среды, осуществляемые JIT/динамическим компилятором, такие как оптимизация, описанная выше, не могут быть сделаны, когда предварительно осуществляется кэширование сборок.

Как сказано выше, большинство собственных языков не отслеживают информацию

о внутренней структуре исходного кода. В противоположность, из-за того что метаданные включаются с промежуточным байт-кодом, что производится в управляемых средах, код, который использует преимущества в текущее время существующих объектов, создает новые объекты, осуществляет наследование от существующих объектов и т.д., может быть без труда сгенерированным во время выполнения. Генерируемый во время выполнения код может иметь потребность в специальном механизме, который должен быть использован для создания или загрузки динамического кода без загрузки генерируемого во время выполнения кода с постоянного запоминающего носителя. Такой механизм будет указываться как порожденное отображение (Reflection Emit) в оставшейся части этого документа. Код, который генерируется во время выполнения, который также называется динамически-генерируемым кодом, типично компилируется во время каждого отдельного случая процесса в промежуточный байт-код и затем обрабатывается JIT/динамическим компилятором. Это является производственными издержками, которые могут быть существенными.

Раскрытие изобретения

Если объект, такой как управляемая сборка или часть управляемой сборки, такая как метод или функция, генерирует дополнительный код во время выполнения, если этот метод или функция являются чистыми, дополнительный генерируемый во время выполнения код кэшируется, так что при последующих выполнениях объекта кэшированный сгенерированный во время выполнения код мог бы быть извлечен из кэша поиском по его идентификатору и извлечением идентифицированного кода из кэша, пропуская одну или более фаз генерации кода. Дополнительно, кэшированный код может быть предварительно откомпилирован компилятором и/или может быть добавлен в качестве дополнительных данных к метаданным исходной сборки. Поэтому генерируемый во время выполнения код может быть кэширован и повторно использован во всех копиях выполняемой программы. Во время выполнения один или более кэшей генерируемого во время выполнения кода просматриваются, и если идентифицированный объект присутствует в кэше, он загружается и используется, устраняя фазу генерации промежуточного байт-кода, фазу компиляции/трансляции или обе фазу генерации промежуточного байт-кода и фазу компиляции/трансляции.

Данная сущность предусмотрена для предоставления выбора концепций в упрощенной форме, которые дополнительно описаны ниже в подробном описании. Данная сущность не предназначена для идентификации ключевых признаков или существенных признаков заявленного объекта изобретения и не подразумевает быть использованной для ограничения объема заявленного объекта изобретения.

Краткое описание чертежей

На чертежах:

Фиг.1 - блок схема примера системы для кэширования генерируемого во время выполнения кода в соответствии с аспектами объекта изобретения, раскрытого в материалах настоящей заявки;

Фиг.2a - способ для выполнения динамической программы, как известно в данной области техники;

Фиг.2b - пример способа для кэширования генерируемого во время выполнения кода в соответствии с аспектами объекта изобретения, раскрытого в материалах настоящей заявки;

Фиг.2c - другой пример способа для кэширования генерируемого во время выполнения кода в соответствии с аспектами объекта изобретения, раскрытого в

материалах настоящей заявки;

Фиг.3 - блок схема, иллюстрирующая пример вычислительной среды, в которой могут быть реализованы аспекты объекта изобретения, раскрытого в материалах настоящей заявки; и

Фиг.4 - блок схема примера интегрированной среды разработки в соответствии с аспектами объекта изобретения, раскрытого в материалах настоящей заявки.

Подробное описание

Обзор

Порожденное отражение является механизмом, который дает возможность создавать динамически во время выполнения и задействовать дополнительные программные инструкции. Т.е. сама программная архитектура может быть определена во время выполнения на основе данных, служб и индивидуальных операций, которые являются соответствующими периоду выполнения. Например, предположим, что конкретная программа выполняет математические операции и пользователь хочет выполнить вычисления на основе матрицы определенного размера. Одним из подходов является написание генерирующей функции, допускающей выполнение вычислений для матрицы с любым размером. Такая функция является статической, поскольку алгоритм, по которому она работает, напрямую решает желаемую проблему. Такая программа, вероятно, должна иметь низкую производительность, поскольку одно из ее требований - работать на любом размере матрицы. В альтернативном варианте динамическое решение проблемы может включать генерацию настраиваемого алгоритма для решения проблемы. Такое решение включало бы статическую функцию A, которая для заданных различных входных аргументов использовала бы некоторые из этих аргументов, чтобы генерировать новую динамическую функцию B, а затем передавать различные входные аргументы для этой динамической функции. Эта специальная динамическая функция удовлетворительно работала бы только на конкретных входных данных, но она могла быть написанной способом, который позволял бы иметь наилучшую производительность при наличии такого ограниченного входного набора. Например, статическая функция A может принимать запрос на перемножение друг на друга двух матриц с размерами 16x16. Статическая функция A может сгенерировать новую динамическую функцию B, которая допускает только перемножение друг на друга матриц 16x16. Затем статическая функция A может делегировать к динамической функции B фактическую задачу по перемножению матриц друг на друга.

При использовании такого механизма как порожденное отражение во время выполнения могут быть сделаны дополнения к программным инструкциям программным путем (динамически). Т.е. для того чтобы продолжить вышеприведенный пример, используя механизм, такой как порожденное отражение, для исходных программных инструкций во время выполнения могут быть сделаны оптимизации программным путем (динамически) для оптимизации этого конкретного выполнения программы для матрицы 16x16. Динамические модификации, сделанные механизмом, таким как порожденное отражение, могут улучшить производительность, увеличить совместимость или предоставить другие полезные преимущества. В пределах контекста объектно-ориентированной среды порожденное отражение дает возможность создания новых типов и новых методов на типах и может задать алгоритм, который реализуют новые методы. В пределах среды управляемого кода промежуточный байт-код может быть сгенерирован через порожденное отражение. В известных существующих системах генерируемый во время выполнения промежуточный байт-код должен быть скомпилирован/конвертирован в собственные машинные инструкции каждый раз, когда

запускается программа.

В соответствии с аспектами объекта изобретения, раскрытого в материалах настоящей заявки, когда динамическая программа запускается в первый раз, создается промежуточный байт-код, задается идентификатор, зависимый от контекста выполнения, и код кэшируется. Когда программа запускается в следующий раз, промежуточный код не должен повторно создаваться. Например, представим, что существует динамическая функция (т.е. функция, генерирующая промежуточный байт-код во время выполнения). Дополнительно предположим, что функция принимает два аргумента, которые могут или не могут меняться между выполнениями функции. В зависимости от значений принятых аргументов будет создаваться различный промежуточный байт-код. В соответствии с аспектами объекта изобретения, раскрытого в материалах настоящей заявки, идентификатор, основанный на значениях аргумента и имени динамически генерирующей код функции, может быть создан и ассоциирован с промежуточным байт-кодом, производимым выполнением этой функции. Идентифицированный байт-код может быть кэширован либо в кэш промежуточного байт-кода, либо в часть метаданных сборки, так что когда в следующий раз функция выполняется с теми же значениями для аргументов, кэшированный байт-код может быть извлечен из кэша и загружен прямо в память, устраняя потребность в повторной генерации промежуточного байт-кода. Подобным образом идентификатор для динамической функции может быть основан на значениях аргумента, заданных для функции, которая генерирует динамическую функцию. Идентификатор для кэшируемой динамической функции может быть основан на алгоритме, который использует генерирующая функция для генерации динамической функции. Идентификатор для динамической функции может идентифицировать конкретную версию динамически генерирующей код функции.

Более того, если промежуточный байт-код конвертируется в собственный код, собственный код может быть кэширован либо в кэш собственного кода, либо в часть метаданных исходной сборки. Когда программа выполняется в следующий раз, не должна выполняться ни генерация промежуточного байт-кода, ни конвертация байт-кода в машинный двоичный код. Собственный двоичный код может быть загружен и выполнен напрямую из кэша собственного кода. Кэшированный собственный двоичный код может быть идентифицирован, как описано в предшествующем абзаце.

Кэширование генерируемого во время выполнения кода

Фиг.1 иллюстрирует пример системы 100, которая кэширует генерируемый во время волнения код, в соответствии с аспектами объекта изобретения, раскрытого в материалах настоящей заявки. Система 100 может включать в себя одно или более из следующего: один или более компонентов 120, которые кэшируют генерируемый во время выполнения код и/или выполняют операцию поиска в кэше генерируемого во время выполнения кода, таком как кэш 109 промежуточного байт-кода и/или кэш 118 многократно используемого exe-файла (машинного кода) и/или кэш 122 исходной сборки, и возвращают идентифицированный код к инициатору запроса, компилятору 105, JIT или динамическому компилятору 110 и одному или более кэшам, таким как кэш 109 промежуточного байт-кода и/или кэш 118 многократно используемого exe-файла (машинного кода) и кэш 112 исходной сборки. JIT или динамический компилятор может генерировать в памяти выполняемый, но не являющийся многократно используемым собственный код. Входные аргументы, такие как входные аргументы 113, могут быть предоставлены, когда должна быть запущена программа.

Система 100 может также включать в себя генератор образов в собственном коде,

такой как NGEN 112. Генератор образов в собственном коде может генерировать и сохранять многократно используемый exe-файл 116 машинного кода в собственном двоичном файле (машинном языке). Многократно используемый exe-файл 116 машинного кода может быть сохранен в кэше 118 многократно используемого exe-файла (машинного кода). Промежуточный байт-код может быть сохранен в кэше 109 промежуточного байт-кода. Собственный двоичный или промежуточный байт-код может быть сохранен в кэше 122 исходной сборки. Кэш 109 промежуточного байт-кода, кэш 118 многократно используемого exe-файла (машинного кода) и/или кэш 122 исходной сборки могут быть кэшами масштаба машины или масштаба системы. Все или части системы 100 могут находиться на одном или более компьютерах, таких как компьютеры, описанные ниже по отношению к фиг.3. Система 100 или ее части могут содержать часть интегрированной среды 600 разработки (IDE), такой как среда, описанная и проиллюстрированная ниже по отношению к фиг.4, находящаяся на одном или более компьютерах, таких как компьютеры, описанные по отношению к фиг.3, также описанной ниже. В альтернативном варианте система 100 или ее части могут быть предоставлены в качестве автономных систем или в качестве съемного модуля.

Компилятор 105 может содержать .NET-компилятор, который компилирует исходный код, написанный на языке .NET, в промежуточный байт-код (например, общий промежуточный язык или CIL). Языки .NET включают в себя, но не ограничены: Visual Basic, Visual J#, C++, C#, J#, Java Script, APL, COBOL, Pascal, Eiffel, Haskell, ML, Oberon, Perl, Python, Scheme, Smalltalk или любые другие языки .NET. Компилятор 102 может содержать компилятор JAVA, которые компилирует исходный код, написанный на JAVA в байт-код JAVA.

Компилятор 105 может откомпилировать исходный код 104 в одну или более управляемые сборки 108 и т.д. Исходный код 104 может включать в себя динамический программный код, т.е. код, который, когда выполняется, генерирует дополнительный код во время выполнения, например генерируемый во время выполнения промежуточный байт-код 111. Управляемая сборка может включать в себя промежуточный байт-код 106 и метаданные 107. Метаданные 107 могут включать в себя информацию, касающуюся настраиваемых атрибутов, которую идентифицируют, если объект, такой как сборка или часть сборки, такой как функция или метод сборки, генерирует код во время выполнения и если объект подходит для кэширования. Вообще программный объект, который подходит для кэширования, не изменяет глобальное состояние. Такой объект называется "чистым" или "чисто функциональным". Например, чистая функция не изменяет глобальное состояние. Метаданные могут быть помещены в генерирующую функцию, в данные аргумента, которые передаются в генератор динамического кода или в другое место. Исходные сборки могут поддерживаться в кэше (библиотеке) исходной сборки, такой как кэш 122 исходной сборки.

Как известно в данной области техники, когда сборка выполняется, сборка загружается в память. Если предварительно сгенерированный exe-файл является доступным (созданным генератором образов в собственном коде, таким как NGEN 112, например), собственное двоичное представление может быть загружено и выполнено. Если предварительно сгенерированный exe-файл не является доступным, JIT или динамический компилятор 110 может преобразовывать управляемую сборку 108 в непригодные для повторного использования собственные двоичные файлы 114 в памяти. В известных системах, даже если сама программа была предварительно скомпилирована в собственном двоичном формате, код, который генерируется программой во время исполнения, должен повторно компилироваться каждый раз,

когда программа запускается. Вызывается механизм, такой как порожденное отражение, который генерирует дополнительный программный код (например, порожденное отражение генерирует промежуточный байт-код, такой как генерируемый во время выполнения промежуточный байт-код 111, использующий дополнительную информацию, сохраненную в метаданных сборки). Генерируемый во время выполнения промежуточный байт-код 111 генерируется, и JIT или динамический компилятор преобразует генерируемый во время выполнения промежуточный байт-код в непригодные для повторного использования собственные двоичные файлы 114 (машинный код).

В противоположность, в соответствии с аспектами объекта изобретения, раскрытого в материалах настоящей заявки, генерируемый во время выполнения код при некоторых обстоятельствах может быть кэширован (например, в кэш 109 промежуточного байт-кода и/или кэш 118 многократно использующегося exe-файла (машинного кода) и/или кэш 122 исходной сборки). Например, предположим, программа включает в себя чистую функцию, которая генерирует дополнительный код во время выполнения. Чистая функция в качестве используемой в материалах настоящей заявки ссылается к функции, которая не изменяет глобальное состояние. Предположим, функция A принимает два аргумента, аргумент a и аргумент b. Промежуточный код генерируется, когда функция A вызывается с аргументом a = значению 1 и аргументом b = значению 2 и имеет один и тот же результат. Поэтому созданием конкретной идентификации для метода, генерирующего код во время выполнения, для конкретных значений аргумента или аргументов (зависимых от выполнения характеристик) для этого метода и кэшированием сгенерированного промежуточного кода генерация промежуточного языка может быть устранена для последующих вызовов этой функции.

Поэтому, когда функция выполняется первый раз, промежуточный язык может быть сгенерирован, идентифицирован и кэширован. Когда функция выполняется второй раз при использовании значения 1 для аргумента a и значения 2 для аргумента b, промежуточный код не имеет необходимости в том, чтобы быть генерированным, потому что он был кэширован. Взамен этого промежуточный код, идентифицируемый именем функции и значениями аргумента, может быть извлечен из кэша, снабженного ключом по идентификатору. Более того, если собственный код был сгенерирован из промежуточного кода (например, через генератор образов в собственном коде, такой как NGEN), машинный код может быть извлечен из кэша машинного кода, устраняя необходимость генерировать промежуточный байт-код и необходимость генерировать собственный машинный код из промежуточного байт-кода.

Могут быть отработаны обычные сценарии, и может быть произведен промежуточный байт-код, который обычно генерируется, и/или могут быть произведены и предварительно кэшированы собственные двоичные файлы. В соответствии с некоторыми аспектами объекта изобретения, раскрытого в материалах настоящей заявки, промежуточный байт-код и/или собственный двоичный образ может быть включен в качестве дополнительных метаданных в сборку, так что даже когда сборка первый раз запускается на пользовательской машине и кэш промежуточного байт-кода масштаба машины и/или кэш собственного двоичного машинного кода является пустым, промежуточный байт-код или собственный двоичный файл может быть найден в самой сборке. В соответствии с некоторыми аспектами объекта изобретения, раскрытого в материалах настоящей заявки, сборка, которая была обновлена динамически генерирующей код функцией, может принимать новый номер версии и/или предыдущая (необновленная или исходная) информация кэша для этой сборки может быть отброшена

или не использоваться.

Фиг.2a иллюстрирует способ для генерируемого во время выполнения кода, как это известно в данной области техники. На этапе 202 исходный код, содержащий динамическую программу (программу, которая производит генерируемый во время выполнения код), может быть скомпилирован компилятором, таким как компилятор, описанный выше, по отношению к фиг.1. Выходные данные компилятора могут быть сборкой, содержащей промежуточный байт-код и метаданные, ассоциированные с промежуточным произведенным байт-кодом. На этапе 204 может быть принят запрос на выполнение сборки. На этапе 206 сборка может быть загружена в память. На этапе 208 может быть определен поиск на основе ключа идентификатора, если доступен предварительно скомпилированный собственный двоичный exe-файл. На этапе 210 процесс поиска может определить, что собственный двоичный exe-файл является доступным и может загрузить и выполнить собственный двоичный exe-файл. На этапе 208 поиск может определить, что предварительно скомпилированный собственный двоичный exe-файл не является доступным.

В ответ на это определение на этапе 212 JIT или динамический компилятор может компилировать или транслировать промежуточный байт-код в собственный двоичный, загружать и выполнять двоичный файл. На этапе 214, или предварительно скомпилированная или скомпилированная JIT, динамическая программа может быть выполнена. На этапе 216 динамическая программа может генерировать дополнительный программный код, так как он выполняется вызовом механизма, такого как порожденное отражение. Этот генерируемый во время выполнения код может быть порожден порождающим отражением в качестве промежуточного байт-кода. Перед тем как генерируемый во время выполнения код может быть выполнен, он должен быть скомпилирован в собственный двоичный код JIT или динамическим компилятором. Здесь не существует возможности предварительной компиляции генерируемого во время выполнения кода, и не существует способа создать собственный двоичный exe-файл для генерируемого во время выполнения кода.

Фиг.2b иллюстрирует пример способа для кэширования генерируемого во время выполнения кода в соответствии с аспектами объекта изобретения, раскрытого в материалах настоящей заявки. На этапе 302 может быть получен запрос о генерации программного объекта, такого как динамическая программа или часть программы, такого как динамический метод или функция. Программный объект, подходящий для кэширования, может быть определен наличием конкретного указателя в метаданных, ассоциированного с методом, т.е. с генерирующимся программным объектом. Метод или функция, которые подходят для того, чтобы их результаты кэшировались, являются не изменяющими глобальное состояние. Метод или функция, которая не производит неожиданного изменения глобального состояния, называется "чистым" методом (или "чистой" функцией). В соответствии с аспектами объекта изобретения, раскрытого в материалах настоящей заявки, указатель для такого метода или функции может быть настраиваемым атрибутом. На этапе 304 идентификатор может быть вычислен или сгенерирован в соответствии с заданным алгоритмом или схемой.

Например, предположим функция A принимает два аргумента, аргумент a и аргумент b. Промежуточный код генерируется, когда функция A вызывается с аргументом a = значению 1 и аргументом b = значению 2, который отличается от промежуточного кода, который сгенерирован, когда функция A вызывалась с аргументом a ≠ значению 1 и/или аргументом b ≠ значению 2. Промежуточный код, который генерируется в следующий раз, когда вызывается функция A с аргументами a = значению 1 и и

аргументом b = значению 2, является таким же, как в первый раз, когда функция A вызывалась с аргументом a = значению 1 и аргументом b = значению 2, если только функция A не менялась между вызовами. Чтобы удостовериться, что устаревший кэш не используется и автоматически подавляется, может быть сгенерирована конкретная идентификация для генерирующей код во время выполнения функции A, используя имя функции (например, "[Namespace::]ObjectName::A") и в дополнение включить конкретные значения аргументов (значение 1 и значение 2), имеющего силу имя сборки, содержащей функцию A. Т.е. использование конкретных значений аргументов в идентификаторе препятствует использованию неверного промежуточного кода, и использование имеющего силу имени сборки, содержащей функцию A, используется для того, чтобы гарантировать, что промежуточный код для неверной версии сборки не используется (например, сборка 1 означает вызов старой функции A, а сборка 2 означает вызов обновленной функции A).

На этапе 306 первый кэш, такой как кэш промежуточного байт-кода, может быть просмотрен на наличие идентифицированного программного объекта. Если на этапе 308 вычисленный идентификатор не найден в первом кэше (например, кэше промежуточного байт-кода), на этапе 330 может быть осмотрен второй кэш. Вторичный кэш может быть кэшем собственного двоичного кода. На этапе 332, если идентифицированный программный объект найден во вторичном кэше, собственный код может быть проверен по отношению к результатам поиска в кэше промежуточного байт-кода для определения, имеет ли он ту же версию, как и в кэше промежуточного кода, и на этапе 346, если собственный код является актуальным (например, у него та же версия, как и у промежуточного байт-кода на этапе 344 для идентифицированного программного объекта), двоичный файл с собственным кодом может быть загружен и выполнен. Если на этапе 332 идентифицированный программный объект не найден во вторичном кэше (собственного кода), собственный код может быть сгенерирован (JIT, или динамическим компилятором, или NGEN для последующего кэширования) на этапе 334, загружен и выполнен на этапе 336 и, если сгенерирован NGEN, кэширован на этапе 338.

Если на этапе 308 промежуточный код для программного объекта не найден в первичном кэше, на этапе 340 промежуточный код может быть сгенерирован на этапе 340 (например, задействуя порожденное отражение) и кэширован на этапе 342. После генерации промежуточного кода на этапе 340, вторичный кэш может быть просмотрен на этапе 330 на наличие программного объекта. Вторичный кэш может быть кэшем собственного двоичного кода. На этапе 332, если идентифицированный программный объект найден во вторичном кэше, собственный код может быть проверен относительно результатов поиска в кэше промежуточного байт-кода для определения, имеет ли он ту же самую версию, как и в кэше промежуточного кода на этапе 344, и на этапе 346, если собственный код является актуальным (например, той же версии, как промежуточный байт-код для идентифицированного программного объекта), двоичный файл с собственным кодом может быть загружен и выполнен. Если на этапе 332 идентифицированный программный объект не найден во вторичном кэше (собственного кода), собственный код может быть сгенерирован (JIT или динамическим компилятором или NGEN для последующего кэширования) на этапе 334, загружен и выполнен на этапе 336 и, если сгенерирован NGEN, кэширован на этапе 338. Будет принято во внимание, что кэш собственного двоичного кода может находиться в пределах самой сборки или может быть кэшем масштаба машины. Подобным образом будет принято во внимание, что первый кэш может быть кэшем промежуточного байт-кода или кэшем исходной

сборки.

В соответствии с аспектами объекта изобретения, раскрытого в материалах настоящей заявки, со ссылкой на фиг.2c на этапе 306 первый кэш, такой как кэш собственного двоичного кода, может быть просмотрен на наличие идентифицированного программного объекта. Если на этапе 308 вычисленный идентификатор находят в первом кэше (например, кэше собственного двоичного кода), на этапе 324 собственный двоичный код может быть извлечен из кэша собственного двоичного кода, загружен и выполнен. Если вычисленный идентификатор не найден в первом кэше (например, кэше собственного двоичного кода), поиск может быть выполнен во втором кэше (например, кэше промежуточного байт-кода) на наличие идентифицированного объекта на этапе 310. Если на этапе 312 промежуточный код для идентифицированного объекта найден в кэше промежуточного байт-кода, промежуточный байт-код может быть извлечен и оттранслирован/скомпилирован в собственный код на этапе 314. Не обязательно собственный код может быть кэширован. На этапе 316 собственный код может быть загружен и выполнен. Если промежуточный байт-код для идентифицированного объекта не найден в кэше промежуточного байт-кода, промежуточный байт-код может быть сгенерирован (например, задействованием порожденного отражения), промежуточный байт-код генерируется на этапе 318, промежуточный байт-код может быть кэширован на этапе 320, загружен и выполнен на этапе 322. Будет принято во внимание, что кэш собственного двоичного кода может быть в пределах самой сборки или может быть кэшем масштаба машины. Подобным образом будет принято во внимание, что первый кэш может быть кэшем промежуточного байт-кода или кэшем исходной сборки и наоборот.

Пример подходящей вычислительной среды

Чтобы обеспечить контекст для различных аспектов объекта изобретения, раскрытого в материалах настоящей заявки, фиг.3 и следующее обсуждение предназначены для предоставления краткого общего описания подходящей вычислительной среды 510, в которой могут быть реализованы различные варианты осуществления. Тогда как объект изобретения, раскрытый в материалах настоящей заявки, описан в общем контексте машинно-исполняемых инструкций, таких как программные модули, выполняемые одним или более компьютерами или вычислительными устройствами, специалисты в данной области техники будут осознавать, что части объекта изобретения, раскрытого в материалах настоящей заявки, могут также быть реализованы в объединении с другими программными модулями и/или в объединении аппаратных средств и программного обеспечения. Обычно программные модули включают в себя процедуры, подпрограммы, объекты, физические артефакты, структуры данных и т.д., которые выполняют конкретные задачи или реализуют конкретные типы данных. Типично, функциональные возможности программных модулей могут быть объединены или распределены так, как желательно в различных вариантах осуществления. Вычислительная среда 510 является всего лишь одним из примеров подходящей операционной среды и не имеет намерением ограничивать область использования функциональности объекта изобретения, раскрытого в материалах настоящей заявки.

Со ссылкой на фиг.3 описано вычислительное устройство общего назначения в форме компьютера 512. Компьютер 512 может включать в себя устройство 514 обработки, системную память 516 и системную шину 518. Устройство 514 обработки может быть любым из различных доступных процессоров. В качестве устройства 514 обработки также могут быть использованы сдвоенные микропроцессоры и другие многопроцессорные архитектуры. Системная память 516 может включать в себя

энергозависимую память 520 и энергонезависимую память 522. Энергонезависимая память 522 может включать в себя постоянное запоминающее устройство (ПЗУ, ROM), программируемое ПЗУ (ППЗУ, PROM), электрически программируемое ПЗУ (EPROM) или флэш-память. Энергозависимая память 520 может включать в себя оперативное запоминающее устройство (ОЗУ, RAM), которое может действовать в качестве внешней кэш-памяти. Системная шина 518 соединяет системные физические артефакты, включающие в себя артефакты от системной памяти 516 до устройства 514 обработки. Системная шина 518 может быть любой из нескольких типов, включающих в себя шину памяти, контроллер памяти, периферийную шину, внешнюю шину или локальную шину, и может использовать любой вид доступных архитектур шин.

Компьютер 512 типично включает в себя многообразие машиночитаемых носителей, таких как энергозависимые и энергонезависимые носители, съемные и несъемные носители. Компьютерный запоминающий носитель может быть реализован любым способом или технологией для хранения информации, такой как машиночитаемые инструкции, структуры данных, программные модули или другие данные. Компьютерные запоминающие носители включают в себя, но не в качестве ограничения, оперативное запоминающее устройство (ОЗУ, RAM), постоянное запоминающее устройство (ПЗУ, ROM), электрически стираемое и программируемое ПЗУ (ЭСППЗУ, EEPROM), флэш-память или другую технологию памяти, CD-ROM, цифровой многофункциональный диск (DVD) или другие оптические дисковые носители, магнитные кассеты, магнитную ленту, магнитные запоминающие диски или другие магнитные запоминающие устройства либо любой другой носитель, который может быть использован для хранения требуемой информации и к которому может быть осуществлен доступ компьютером 512.

Будет принято во внимание, что фиг.3 изображает программное обеспечение, которое может действовать как посредник между пользователями и компьютерными ресурсами. Это программное обеспечение может включать в себя операционную систему 528, которая может храниться на дисковом запоминающем устройстве 524 и которая может управлять и выделять ресурсы компьютерной системы 512. Дисковое запоминающее устройство 524 может быть жестким диском, соединенным с системной шиной 518 через интерфейс несъемной памяти, такой как интерфейс 526. Системные приложения 530 используют преимущество управления ресурсами операционной системы 528 через программные модули 532 и программные данные 534, сохраненные либо в системной памяти 516, либо на дисковом запоминающем устройстве 524. Будет принято во внимание, что компьютеры могут быть реализованы с различными операционными системами или сочетаниями операционных систем.

Пользователь может вводить команды или информацию в компьютер 512 через устройство(а) 536 ввода. Устройства 536 ввода включают в себя, но не в качестве ограничения, координатно-указательное устройство, такое как мышь, шаровой манипулятор, перо, сенсорную панель, клавиатуру, микрофон и тому подобное. Эти и другие устройства ввода подключаются к устройству 514 обработки посредством системной шины 518 через интерфейсный порт(ы) 538. Интерфейсный порт(ы) 538 может представлять порт с последовательным вводом/вводом данных, порт с параллельным вводом/выводом данных, универсальную последовательную шину (USB) и подобные. Устройство(а) 540 вывода может использовать те же самые порты, что и устройство(а) ввода. Адаптер 542 вывода предусмотрен, чтобы проиллюстрировать, что существует несколько устройств 540 вывода, подобных дисплеям, динамикам и принтерам, которые требуют конкретных адаптеров. Адаптеры 552 вывода включают в себя, но не в качестве ограничения, видео и звуковые карты, которые предоставляют

соединение между устройством 540 вывода и системной шиной 518. Другие устройства и/или системы или устройства, такие как удаленный компьютер(ы) 544 могут предоставлять возможности и ввода, и вывода.

Компьютер 512 может работать в сетевой среде с использованием логических соединений с одним или более удаленными компьютерами, такими как удаленный компьютер(ы) 544. Удаленный компьютер 544 может быть еще одним персональным компьютером, сервером, маршрутизатором, сетевым ПК, одноранговым устройством или другим общим узлом сети и типично включает в себя многие или все из элементов, описанных выше относительно компьютера 512, хотя на фиг.3 было проиллюстрировано только запоминающее устройство 546 памяти. Удаленный компьютер(ы) 544 может быть логически соединен через соединение 550 связи. Сетевой интерфейс 548 включает в себя сети связи, такие как локальные сети (LAN), глобальные сети (WAN), но может также включать в себя другие сети. Соединение(я) 550 связи указывает ссылкой на аппаратные средства/программное обеспечение, применяемые, чтобы подключать сетевой интерфейс 548 к шине 518. Соединение 550 может быть внутренним или внешним для компьютера 512 и может включать в себя внутренние или внешние технологии, такие как модемы (телефонный, кабельный, DSL и беспроводной) и ISDN-адаптеры, Ethernet-карты и т.д.

Будет принято во внимание, что показанные сетевые соединения являются только примерами и может быть использовано другое средство установления линии связи между компьютерами. Любой рядовой специалист в данной области техники может принять во внимание, что компьютер 512 или другое клиентское устройство может быть развернуто в качестве части компьютерной сети. В этом отношении объект изобретения, раскрытый в материалах настоящей заявки, может относиться к любой вычислительной системе, имеющей любое количество памяти и запоминающих устройств и любое количество приложений и процессов, происходящих на любом числе запоминающих устройств и объемов памяти. Аспекты объекта изобретения, раскрытые в материалах настоящей заявки, могут применяться в среде с серверными компьютерами и клиентскими компьютерами, развернутыми в сетевой среде, имеющей удаленное и локальное запоминающее устройство. Аспекты объекта изобретения, раскрытые в материалах настоящей заявки, могут также применяться к автономному вычислительному устройству, имеющему функциональность языка программирования, возможности интерпретации и выполнения.

Фиг.4 иллюстрирует интегрированную среду 600 разработки (IDE) и общеязыковую среду 602 исполнения. IDE 600 может позволять пользователю (например, разработчику, программисту, дизайнеру, кодировщику и т.д.) разрабатывать, кодировать, компилировать, тестировать, запускать, редактировать, отлаживать или строить программу, набор программ, web-сайты, web-приложения и web-службы в компьютерной системе. Компьютерные программы могут включать в себя исходный код (компонент 610), созданный в одном или более языков исходного кода (например, Visual Basic, Visual J#, C++, C#, J#, Java Script, APL, COBOL, Pascal, Eiffel, Haskell, ML, Oberon, Perl, Python, Scheme, Smalltalk и подобных). IDE 600 может предоставлять среду разработки собственного кода или может предоставлять разработку управляемого кода, которая запускается на виртуальной машине или может предоставлять их объединение. IDE 600 может предоставлять среду разработки управляемого кода, использующую платформу .NET Framework. Компонент 650 промежуточного языка может быть создан из компонента 610 исходного кода и компонента 611 собственного кода, используя компилятор 620 зависимого от языка источника, и компонент 611 собственного кода

(например, машиноисполняемые инструкции) создается из компонента 650 промежуточного языка, используя компилятор 660 промежуточного языка (например, динамичный (JIT) компилятор), когда приложение выполняется. Т.е. когда IL-приложение выполняется, оно компилируется, перед тем как выполняться, в подходящий машинный язык для платформы, на которой он выполняется, таким образом, образуя код, переносимый на несколько платформ. В альтернативном варианте, в других вариантах осуществления программы могут быть скомпилированы в собственном коде машинного языка (не показан), подходящего для его предназначенной платформы.

Пользователь может создать и/или отредактировать компонент исходного кода согласно известным технологиям программирования программного обеспечения и определенным логическим и синтаксическим правилам, ассоциированным с конкретным языком источника через пользовательский интерфейс 640 и редактор 651 исходной кода в IDE 600. После этого компонент 610 исходного кода может быть скомпилирован через компилятор 620 исходного кода, на основании чего может быть создано представление промежуточного языка для программы, такое как сборка 630. Сборка 630 может содержать компонент 650 промежуточного языка и метаданные 642. Перед развертыванием можно проверить достоверность архитектуры приложения.

Различные технологии, описанные в материалах настоящей заявки, могут быть реализованы в связи с аппаратными средствами или программным обеспечением или при необходимости их сочетанием. Таким образом, способы и устройство, описанное в материалах настоящей заявки, или их определенные аспекты или части могут принимать форму программного кода (т.е. инструкций), заключенного в материальные носители, такие как гибкие диски, CD-ROM, жесткие диски или любой другой машиночитаемый запоминающий носитель, в котором, когда программный код загружен и выполняется машиной, такой как компьютер, машина становится устройством для осуществления аспектов объекта изобретения, раскрытого в материалах настоящей заявки. В случае выполнения программного кода на программируемых компьютерах, вычислительное устройство, как правило, будет включать в себя процессор, запоминающий носитель, доступный для чтения процессором (включая энергозависимую и энергонезависимую память и/или элементы запоминающего устройства), по меньшей мере, одно устройство ввода и, по меньшей мере, одно устройство вывода. Одна или более программ, которые могут использовать создание и/или реализацию зависимых от домена аспектов моделей программирования, например, посредством использования API-интерфейса обработки данных и т.п., могут быть реализованы в объектно-ориентированном языке программирования высокого уровня, чтобы обмениваться данными с компьютерной системой. Тем не менее при необходимости программы могут быть реализованы на языке сборки или машины. В любом случае язык может быть компилируемым или интерпретируемым языком и может быть объединен с реализациями в аппаратных средствах.

Между тем как объект изобретения, раскрытый в материалах настоящей заявки, был описан в связи с фигурами, будет понятно, что для выполнения одних и тех же функций различными способами могут быть сделаны модификации.

Формула изобретения

1. Система (100), которая кэширует генерируемый во время выполнения код, содержащая:

компонент (120), выполняющийся в управляемой среде выполнения кода на компьютере, при этом компонент кэширует объект (116, 111) в первом кэше (109, 118)

или во втором кэше (109, 118), при этом кэшируемый объект (116, 111) содержит генерируемый во время выполнения код, причем генерируемый во время выполнения код генерируется динамически генерирующим код объектом, при этом динамически генерирующий код объект не изменяет глобальное состояние и содержит метаданные (107), которые идентифицируют, является ли объект пригодным для кэширования, при этом кэшируемый объект (116, 111) извлекается из первого кэша (109, 118) или второго кэша (109, 118) при последующих выполнениях кэшированного объекта (116, 111), пропуская по меньшей мере одну фазу генерации кода, при этом пропускаемая по меньшей мере одна фаза генерации кода содержит генерацию промежуточного байт-кода или генерацию собственного двоичного кода.

2. Система по п.1, в которой первый кэш (109, 118) представляет собой кэш, который хранит промежуточный байт-код, представляющий генерируемый во время выполнения код, при этом пропускаемая по меньшей мере одна фаза генерации кода содержит генерацию промежуточного байт-кода для генерируемого во время выполнения кода.

3. Система по п.1, в которой второй кэш (109, 118) представляет собой кэш, который хранит собственный двоичный код, при этом пропускаемая по меньшей мере одна фаза генерации кода содержит генерацию собственного двоичного кода для генерируемого во время выполнения кода.

4. Система по п.1, в которой первый кэш является кэшем исходной сборки (122), при этом пропускаемая по меньшей мере одна фаза генерации содержит генерацию промежуточного байт-кода для генерируемого во время выполнения кода или генерацию собственного двоичного кода для генерируемого во время выполнения кода.

5. Система по п.1, в которой промежуточный байт-код для генерируемого во время выполнения кода извлекается по идентификатору, состоящему из имени, ассоциированного с динамически генерирующим код объектом, по меньшей мере одного аргумента, передаваемого в динамически генерирующий код объект, и указателя версии для динамически генерирующего код объекта.

6. Способ кэширования генерируемого во время выполнения кода, содержащий этапы, на которых:

вычисляют идентификатор (304) для объекта (116, 111), представляющего собой генерируемый во время выполнения код, сгенерированный чисто функциональным генерирующим во время выполнения код объектом, при этом объект в виде сгенерированного во время выполнения кода содержит промежуточный байт-код, созданный механизмом отражения/порождения (Reflection.Emit), при этом вычисленный идентификатор содержит зависящие от контекста выполнения характеристики генерируемого во время выполнения кода, при этом динамически генерирующий код объект не изменяет глобальное состояние и содержит метаданные (107), которые идентифицируют, является ли объект пригодным для кэширования;

сохраняют объект в первом кэше (109, 118) с возможностью извлечения по вычисленному идентификатору в качестве ключа при первом выполнении объекта (320);

возвращают (322) сохраненный объект в ответ на второе выполнение сохраненного объекта, при этом генерацию промежуточного байт-кода для сохраненного объекта пропускают для второго выполнения сохраненного объекта.

7. Способ по п.6, дополнительно содержащий этап, на котором генерируют собственный двоичный код для генерируемого во время выполнения кода из промежуточного байт-кода для сохраненного объекта (111, 116).

8. Способ по п.7, дополнительно содержащий этап, на котором кэшируют

сгенерированный собственный двоичный код в кэше собственного двоичного кода, который представляет собой второй кэш (109, 118).

9. Способ по п.8, в котором первый кэш или второй кэш является исходной сборкой (122), содержащей сохраненный объект (111, 116).

10. Способ по п.6, в котором вычисленный идентификатор представляет собой идентификатор, состоящий из: имени, ассоциированного с чисто функциональным генерирующим код во время выполнения объектом, по меньшей мере одного аргумента, передаваемого в чисто функциональный генерирующий во время выполнения код объект, и указателя версии для чисто функционального генерирующего во время выполнения код объекта.

11. Машиночитаемый носитель информации, содержащий машиноисполняемые инструкции, которые при их исполнении предписывают управляемой компьютерной среде:

кэшировать объект (116, 111) в первом кэше или во втором кэше (109, 118), при этом кэшируемый объект содержит генерируемый во время выполнения код, генерируемый генерирующим во время выполнения код объектом, который не изменяет глобальное состояние и содержит метаданные (107), которые идентифицируют, является ли объект пригодным для кэширования, причем кэшированный объект извлекается при последующих выполнениях объекта, пропуская по меньшей мере одну фазу генерации кода в этих последующих выполнениях.

12. Машиночитаемый носитель информации по п.11, содержащий дополнительные машиноисполняемые инструкции, которые при их исполнении предписывают компьютерной среде пропускать генерацию промежуточного байт-кода для генерируемого во время выполнения кода для последующих выполнений кэшированного объекта (111, 116).

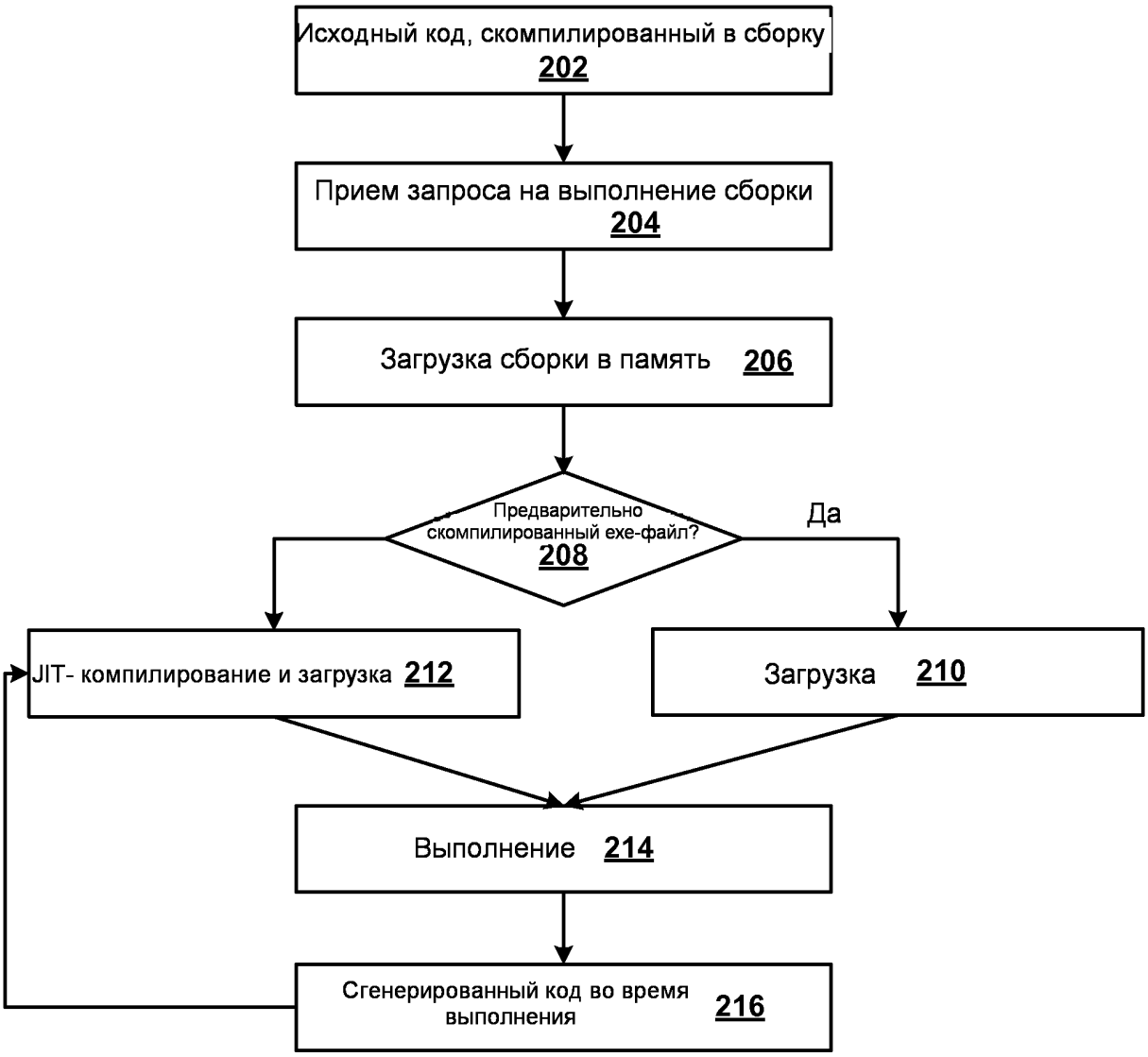
13. Машиночитаемый носитель информации по п.11, содержащий дополнительные машиноисполняемые инструкции, которые при их исполнении предписывают компьютерной среде пропускать генерацию собственного двоичного кода из промежуточного байт-кода для кода, генерируемого во время выполнения, для упомянутых последующих выполнений кэшированного объекта (111, 116).

14. Машиночитаемый носитель информации по п.13, содержащий дополнительные машиноисполняемые инструкции, которые при их исполнении предписывают компьютерной среде кэшировать собственный двоичный код или промежуточный байт-код в исходной сборке (122) кэшированного объекта (111, 116).

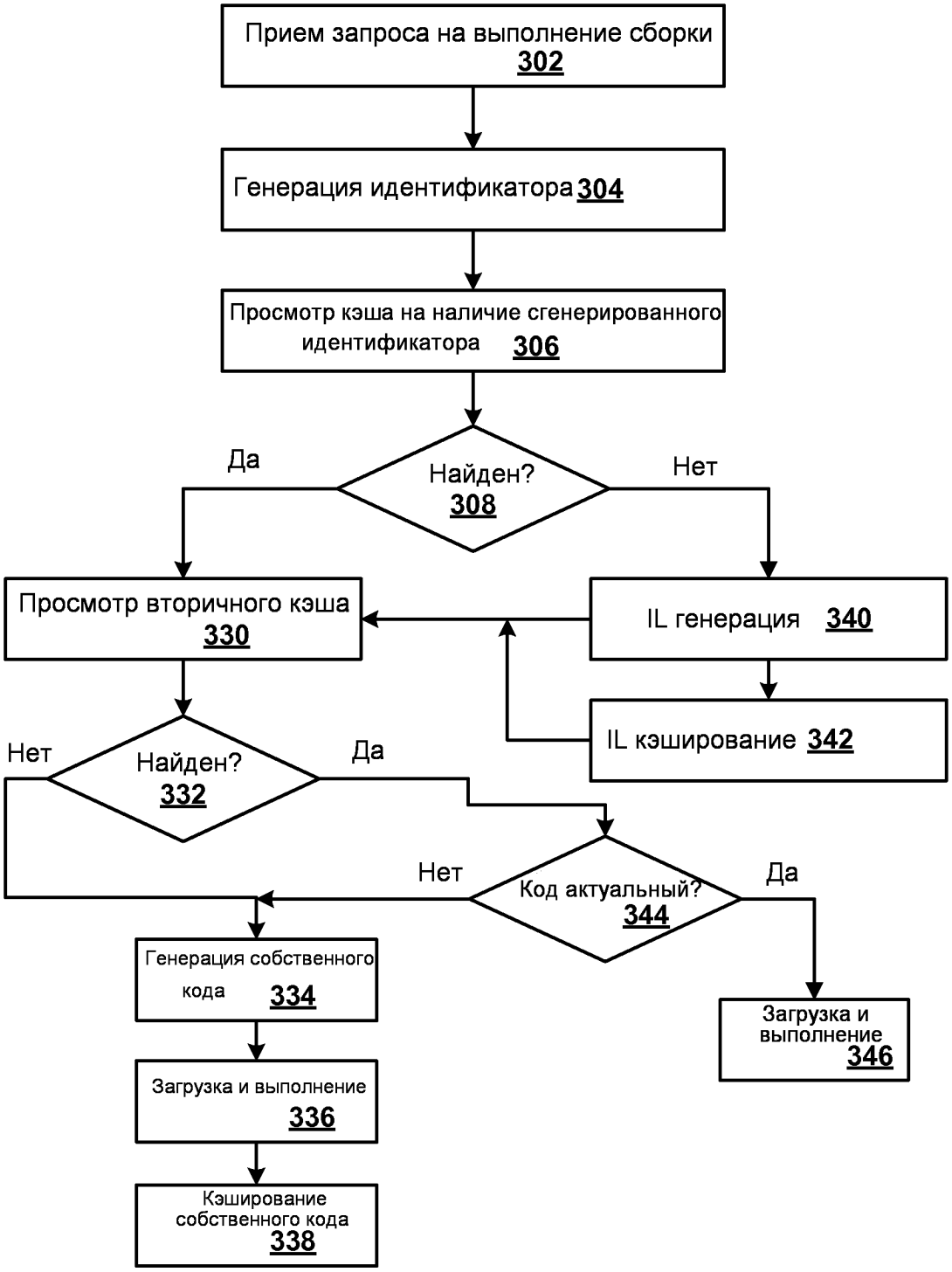
15. Машиночитаемый носитель информации по п.13, содержащий дополнительные машиноисполняемые инструкции, которые при их исполнении предписывают компьютерной среде:

обеспечивать объекту (111, 116) конкретный идентификатор, используя зависящие от выполнения характеристики кэшируемого объекта; и

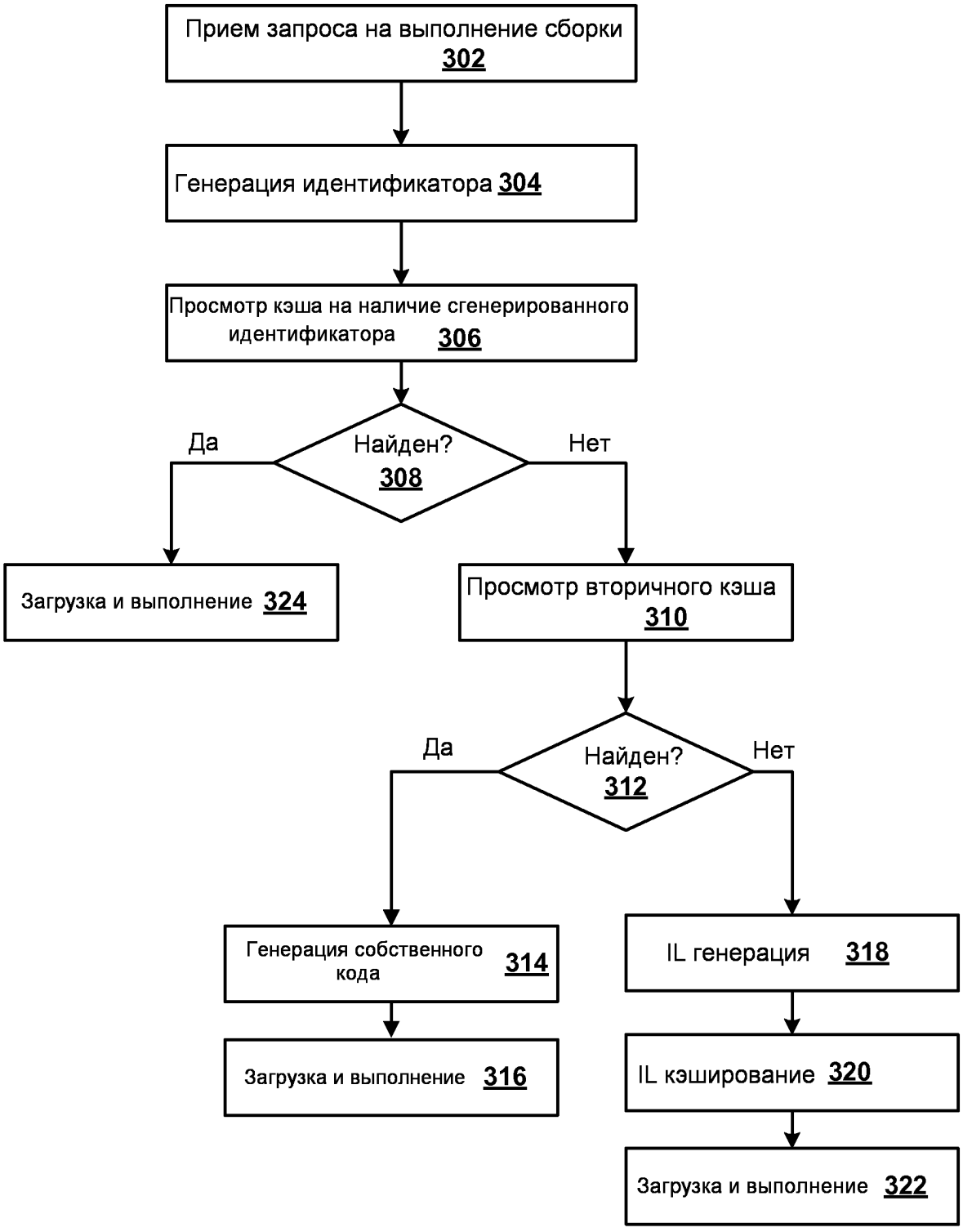
извлекать кэшированный объект из первого кэша или второго кэша, используя в качестве ключа этот конкретный идентификатор.


Исходный код, скомпилированный в сборку
202
Прием запроса на выполнение сборки
204
Загрузка сборки в память 206
Предварительно скомпилированный exe-файл?
208
Да
JIT- компилирование и загрузка 212
Загрузка 210
Выполнение 214
Сгенерированный код во время выполнения 216

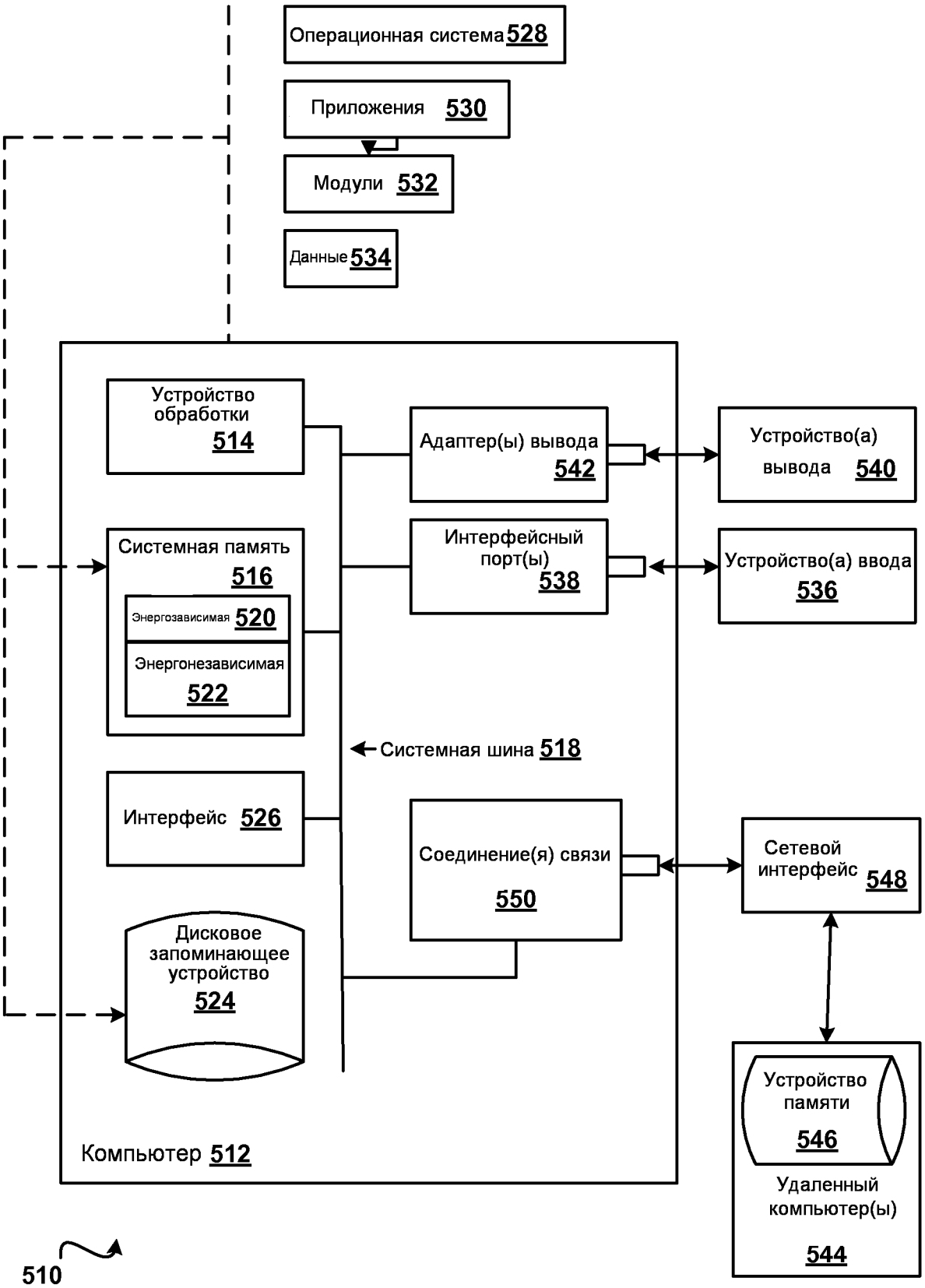
ФИГ. 2а


Прием запроса на выполнение сборки
302
Генерация идентификатора 304
Просмотр кэша на наличие сгенерированного идентификатора 306
Найден?
308
Да
Нет
Просмотр вторичного кэша
330
IL генерация 340
IL кэширование 342
Найден?
332
Нет
Да
Код актуальный?
344
Нет
Да
Генерация собственного кода 334
Загрузка и выполнение
336
Кэширование собственного кода
338
Загрузка и выполнение
346

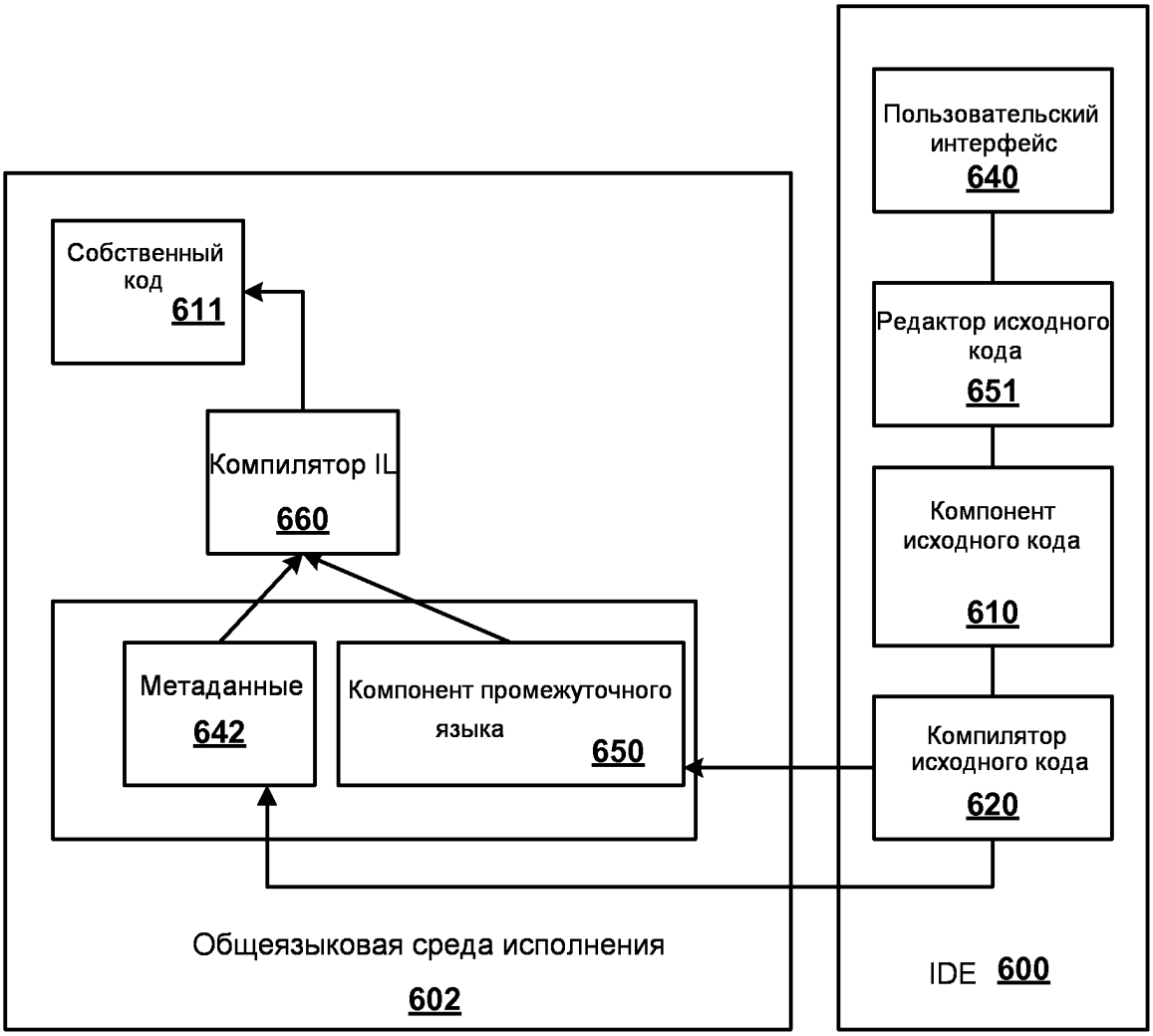
ФИГ. 2b


Прием запроса на выполнение сборки 302
Генерация идентификатора 304
Просмотр кэша на наличие сгенерированного идентификатора 306
Найден? 308
Да
Нет
Загрузка и выполнение 324
Просмотр вторичного кэша 310
Найден? 312
Да
Нет
Генерация собственного кода 314
Загрузка и выполнение 316
IL генерация 318
IL кэширование 320
Загрузка и выполнение 322

ФИГ. 2c


Операционная система 528
Приложения 530
Модули 532
Данные 534
Устройство обработки 514
Системная память 516
Энергозависимая 520
Энергонезависимая 522
Интерфейс 526
Дисковое запоминающее устройство 524
Системная шина 518
Адаптер(ы) вывода 542
Интерфейсный порт(ы) 538
Соединение(я) связи 550
Устройство(а) вывода 540
Устройство(а) ввода 536
Сетевой интерфейс 548
Устройство памяти 546
Удаленный компьютер(ы) 544
Компьютер 512
510

ФИГ. 3


Пользовательский интерфейс
640
Редактор исходного кода
651
Компонент исходного кода
610
Компилятор исходного кода
620
IDE 600
Собственный код
611
Компилятор IL
660
Метаданные
642
Компонент промежуточного языка
650
Общеязыковая среда исполнения
602

ФИГ. 4