УДК 004.9

О.М. ЯШИНА, О.А. КРАВЧУК
Хмельницький національний університет

ВИКОРИСТАННЯ МОВИ GO У РОЗРОБЦІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У даній статті розглядаються теоретичні аспекти мови програмування Go, її переваги та недоліки перед іншими мовами, порівняння. Також обґрунтовується доцільність та протиріччя використання мови Go в розробці програмного забезпечення.

Ключові слова: *програмний продукт, інформаційні технології, програмне забезпечення, Go, Golang, мова програмування.*

O.M. YASHYNA, O.A. KRAVCHUK
Khmelnytsky National University

USAGE OF GO LANGUAGE IN SOFTWARE DEVELOPMENT

The theoretical aspects of the programming language Go, advantages and disadvantages over other languages, comparisons. Also justifies and contradicts the use of Go in software development. Go (often also Golang) is a multithreaded programming language developed by Google. The first development of the language took place in September 2007 and was aimed at reducing the resource costs for the creation of local projects in the company, its first release took place in 2009. Supported by all popular operating systems, which makes it very convenient for any developer. Go language was developed as a language of system programming to create highly effective programs that work on modern distributed systems and multi-core processors. It can be seen as an attempt to replace such languages as Java, C ++, Php. The greatest attention was paid to the highly effective compilation of programs. Applications written on Go do not require a virtual machine to execute, since they are compiled into an object code. The language continues to evolve, and developers are considering the possibility of incorporating generalized programming into the language. In the "Frequently Asked Questions" language is based on arguments against the use of allegations, and inheritance without specifying the type, on the contrary, stands out. The syntax of the Go language is similar to the syntax of the C language, with separate elements borrowed from Oberon and script languages. This language is already used by its authors - Google, as well as many other big firms such as: Abode, BBC, IBM, Intel and even Medium. This suggests that Go is capable of satisfying any business model, as well as solving a lot of problems, even in large projects, which hints at its use in its own startups or ideas.

Keywords: software product, information technology, software, Go, Golang, programming language.

Вступ. До недавнього часу найбільш розповсюджені мови програмування, такі як Java, C++, Php, охоплювали основний ринок розробки програмних продуктів. З часом ці мови стали доволі гнучкими та складними, завдяки ним можна написати проект любої складності. Але постає нова проблема – велика ресурсозатратність для бізнесу. Наймати спеціалістів цих мов доволі дорого, а також постають проблеми у супроводі продукту, якщо його взяла інша команда розробників. Вперше, над цією проблемою задумалася корпорація Google і було вирішено створити просту але ефективну у застосуванні мову програмування. Це б дозволило вирішити такі проблеми:

1. Створення допоміжних продуктів, котрі не виходять за межі корпорації.
2. Низькі затрати ресурсів на створення таких продуктів.
3. Навчання програмістів єдиної мови програмування.
4. Легка заміна програміста іншим спеціалістом, котрий за вихідні розбереться з проектом.

Для вирішення таких проблем і була створена мова програмування Go, котра з часом набула популярності, вийшла у світ і наразі використовується багатьма фірмами та стартапами для швидкого, дешевого та ефективного створення програмного продукту.

Метою статті є визначення теоретичних аспектів мови програмування Go, а також обґрунтування доцільності та виокремлення протиріч використання цієї мови при створенні програмного продукту.

Аналіз досліджень та публікацій. Методологічною основою нашого дослідження стали праці таких дослідників як Батчер Метт, Фарина Метт, Алан А.А. Донован, Брайан У. Керниган, та багато інших. Загалом ці дослідники описують саму мову та деякі її переваги та недоліки над іншими, а також застосування кращих сторін Go на практиці.

Не зважаючи на велику кількість документації та статей по даній темі, залишається відкритим питання використання на практиці мови, що і обґрунтовує науковий та практичний інтерес до цією теми.

Виклад основного матеріалу. Go (часто також Golang) — багатопоточна мова програмування, розроблена компанією Google. Перша розробка мови відбувалася у вересні 2007 року і мала на меті знизити ресурсні затрати на створення локальних проектів в компанії, перший її реліз відбувся у 2009 році. Підтримується усіма популярними операційними системами, що робить її дуже зручною для любого розробника [1, 3].

Мова Go розроблялася як мова системного програмування для створення високоефективних програм, котрі працюють на сучасних розподілених системах і багатоядерних процесорах. Вона може розглядатися як спроба замінити такі мови як Java, C++, Php. Найбільша увага приділялося високоефективній компіляції програм. Програми, написані на Go не потребують віртуальної машини для виконання, так як вони компілюються в об'єктний код.

Основні можливості мови Go:

- Go — мова зі строгою типізацією. Доступний автоматичний вивід типів, для користувацьких типів.
- Повноцінна підтримка вказівників, але без можливостей застосовувати до них математичні операції.

- Стрічковий тип зі вбудованою підтримкою юнікоду.
- Використання динамічних масивів, хеш-таблиць, цикли для обходу колекцій.
- Засоби для функціонального програмування: функції без імені, замикання, передача функції в параметрі, повернення значень функції
- Автоматичне керування пам'яттю для garbage collector
- Засоби ООП, але без підтримки наслідування реалізації. Підтримуються тільки інтерфейси. Завдяки цьому Go являється більше процедурною мовою з підтримкою інтерфейсів.
- Засоби паралельного програмування: вбудовані в мову потоки, взаємодія потоків через канали та інші засоби організації багатопоточних програм.
- Лаконічний та простий синтаксис, основою для якого служить C.

Можливості, котрі забрані з Go:

- Структурний запис обробників виключень, замість неї пропонується повернення перевірки кодів повернення з використанням багатозначних функцій, відкладених функцій та спеціального інтерфейсу error для перехвату виключень.
- Наслідування реалізації. На заміну приходить анонімне вкладення типів.
- Використання стверджень.
- Перевизначення методів.

Мова продовжує розвиватися, і розробники розглядають можливість включення в мову засобів узагальненого програмування. У «Частих питаннях» за мовою наводяться аргументи проти використання стверджень, а успадкування без вказування типу, навпаки, відстоюється.

Синтаксис мови Go схожий з синтаксисом мови Сі, з окремими елементами, запозиченими з Оберона і скриптових мов.

Будь-яка локальна змінна обов'язково повинна бути використана, тобто її значення має брати участь в будь-якій операції в межах функції, де вона оголошена. На відміну від Паскаля і Сі, де оголошення локальної змінної і подальше її невикористання або втрата значення, присвоєння локальній змінній (коли змінній присвоюється значення, яке потім ніде не читається), може лише викликати попередження (warning) компілятора, в Go така ситуація вважається мовною помилкою і призводить до неможливості компіляції програми. Це означає, зокрема, що програміст не може проігнорувати значення (або одне зі значень), що повертається функцією, просто присвоївши його якій-небудь змінній і відмовитися від її подальшого використання. Якщо виникає необхідність ігнорувати одне зі значень, що повертаються викликом функції, використовується зумовлена псевдозмінна з ім'ям «_» (один знак підкреслення). Вона може бути вказана в будь-якому місці, де повинна бути змінна, що приймає значення. Відповідне значення не буде надано жодній змінній і просто загубиться. Сенс такого архітектурного рішення – виявлення на стадії компіляції можливої втрати результатів обчислень: випадковий пропуск обробки значення буде виявлений компілятором, а використання псевдозмінної «_» вкаже на те, що програміст свідомо проігнорував результати [2].

Для обробки помилок творці мови рекомендують використовувати повернення помилки як одного з результатів виклику функції і перевірку його на місці виклику. Типовий порядок, який витримує в стандартних бібліотеках Go, виглядає наступним чином:

- В одному з параметрів (зазвичай останньому) функція повертає об'єкт-помилку. В якості типу помилки зазвичай використовується бібліотечний інтерфейс error. Типовою практикою є повернення порожнього покажчика nil, якщо функція виконалася без помилок.
- Після виклику отриманий з функції об'єкт перевіряється і помилка, якщо вона виникла, обробляється.
- Проігнорувати помилку, яка повертається з функції неможливо, так як ініціалізація змінної без подальшого використання в мові Go призводить до помилки компіляції. Звичайно, цей ефект можна обійти підстановкою замість err псевдозмінної «_», але це вважається поганою практикою і, у всякому разі, явно кидається в очі при перегляді коду.

Початківці програмісти на Go нерідко критикують його ідеологію обробки помилок, заявляючи, що численні перевірки помилок засмічують код і ускладнюють його сприйняття, тоді як механізм винятків дозволяє зосередити всю обробку помилок в блоках catch. Насправді ідеологія Go цілком дозволяє обробляти помилки елегантно і економно, в літературі з мови описаний ряд патернів для цього.

При виникненні фатальних помилок, які роблять неможливим подальше виконання програми, виникає стан «паніки» (panic). Типовий приклад виникнення паніки - поділ на нуль в процесі обчислень або звернення за межі масиву. За відсутності обробки паніка призводить до аварійного завершення програми з видачею повідомлення про помилку і трасування стека викликів. Для забезпечення відмовостійкості програми паніка теж може бути перехоплена і оброблена. Для цього використовується механізм відкладеного виконання defer. Інструкція defer, як говорилося вище, отримує в якості параметра виклик функції (тобто фактично створює замикання), який проводиться тоді, коли виконання програми залишає поточну область видимості. Це відбувається навіть у випадку паніки. Для її перехоплення в функції, що викликається в defer, необхідно викликати стандартну функцію recover () – вона припиняє системну обробку паніки і повертає її причину у вигляді об'єкта error. Далі програміст може обробити отриману помилку будь-яким бажаним чином, в тому числі і відновити паніку, викликавши стандартну функцію panic (err error).

Go дає можливість створити новий потік виконання програми (go-процедуру) за допомогою ключового слова go, яке запускає анонімну або іменовану функцію в заново створеної go-процедурі (аналог

співпрограми). Всі go-процедури в рамках одного процесу використовують загальний адресний простір, виконуючись над ОС-потоками, але без жорсткої прив'язки до останніх, що дозволяє виконуваній go-процедурі залишати потік із заблокованої go-процедурою (очікуючій, наприклад, надсилання і прийняття повідомлення з каналу) і продовжувати роботу далі [2].

В об'єктно-орієнтовному програмування спеціальне ключове слово для оголошення класу в Go відсутня, але для будь-якого іменованого типу, включаючи структури і базові типи накшталт int, можна визначити методи, так що в сенсі ООП все такі типи є класами.

Синтаксис визначення методу запозичений з мови Оберон-2 і відрізняється від звичайного визначення функції тим, що після ключового слова func в круглих дужках оголошується так званий «одержувач» (англ. Receiver), тобто об'єкт, для якого викликається метод, і тип, до якого належить метод. Якщо в традиційних об'єктних мовах одержувач зазвичай має стандартне ім'я (в C ++ або Java - «this», в ObjectPascal - «self» і т. п.), То в Go він вказується явно і його ім'я може бути будь-яким правильним Go-ідентифікатором.

Спадкування класів (структур) в Go формально відсутнє, але є технічно близький до нього механізм вбудовування (англ. Embedding). В описі структури можна використовувати так зване анонімне поле – поле, для якого не вказується ім'я, а тільки тип. В результаті такого опису всі елементи вбудованої структури стануть однойменними елементами вбудованої. На відміну від класичного наслідування, вбудовування НЕ тягне поліморфну поведінку (об'єкт вбудованого класу не може виступати в якості об'єкта вбудованого без перетворення типів).

Поліморфізм класів забезпечується в Go механізмом інтерфейсів (схоже на повністю абстрактні класи в C ++). Інтерфейс описується за допомогою ключового слова interface, всередині (на відміну від описів типів-класів) опису оголошуються надавані інтерфейсом методи.

У Go немає можливості явно вказати, що певний тип реалізує певний інтерфейс. Замість цього діє правило: кожен тип, що надає методи, позначені в інтерфейсі, може бути використаний як реалізація цього інтерфейсу [4].

Хоча в принципі можливо побудувати в програмі на Go і ієрархію інтерфейсів, як це практикується в інших об'єктних мовах, і навіть імітувати спадкування, це вважається поганою практикою. Мова диктує не ієрархічний, а композиційний підхід до системи класів і інтерфейсів. Класи-структури при такому підході взагалі можуть залишатися формально незалежними, а інтерфейс не об'єднуються в єдину ієрархію, а створюються для конкретних застосувань, при необхідності вставляють вже наявні. Неявна реалізація інтерфейсів в Go забезпечує надзвичайну гнучкість цих механізмів і мінімум технічних труднощів при їх використанні.

Такий підхід до спадкоємства відповідає деяким практичним тенденціям сучасного програмування. Так в знаменитій книзі «банди чотирьох» (Еріх Гамма і ін.) Про патерни проектування, зокрема, написано: Залежність від реалізації може спричинити за собою проблеми при спробі повторного використання підкласу. Якщо хоча б один аспект успадкованої реалізації непридатний для нової предметної області, то доводиться переписувати батьківський клас або замінювати його чимось більш підходящим. Така залежність обмежує гнучкість і можливості повторного використання. З проблемою можна впоратися, якщо наслідувати тільки абстрактні класи, оскільки в них зазвичай зовсім немає реалізації або вона мінімальна.

У Go немає поняття віртуальної функції. Поліморфізм забезпечується за рахунок інтерфейсів. Якщо для виклику методу використовується змінна звичайного типу, то такий виклик зв'язується статично, тобто завжди викликається метод, певний для даного конкретного типу. Якщо ж метод викликається для змінної типу «інтерфейс», то такий виклик зв'язується динамічно, і в момент виконання для запуску вибирається той варіант методу, який визначений для типу об'єкта, фактично присвоєного в момент виклику цієї змінної [5].

З технічного боку, ніша у Go досить скромна: мережа, утиліти, бекенд. Якщо є складна мережа або багато нод, для яких потрібен окремий підхід, то Go – це хороший вибір (судячи з досвіду CloudFlare). Якщо потрібна консольна утиліта, начебто докера або консула, то Go теж підійде. Якщо потрібен швидкий і в принципі ефективний бекенд, то теж можна вибрати Go.

Висновки. Отже, проаналізувавши різні джерела можна дійти до висновку, що мова програмування Go має своє майбутнє і може використовуватися багатьма корпораціями і фірмами для вирішення різноманітних проблем. Хоча ця мова і відрізняється від багатьох звичних, вона є дуже потужною і дає таку ж високу ефективність як C, C++, швидку обробку багатозадачності як Java, а також дає можливість легкого написанню коду. Ця мова вже використовується її авторами – Google, а також багатьма іншими великими фірмами, такими як: Abode, BBC, IBM, Intel и навіть Medium. Це говорить про те, що мова Go здатна задовольнити будь-яку бізнес-модель, а також вирішити багато проблем, навіть у великих проектах, що натякає на її використання у власних стартапах або ідеях.

Література

1. https://golang.org/doc/effective_go.html.
2. https://tour.golang.org.
3. https://habrahabr.ru/hub/go/
4. https://uk.wikipedia.org/wiki/Go_(мова_програмування).
5. Matt Butcher. Go in Practice. – M. : DMK Press, 2017. – 374 p.

Рецензія/Peer review : 2.05.2018 р. Надрукована/Printed :15.05.2018 р.
Рецензент: док. фіз-мат. наук, проф. Бедратюк Л.П.