Какие основные преимущества Go?

Простота синтаксиса, высокая производительность, компилируется в машинный код. Многопоточность, есть сборщик мусора для автоматического управления памятью.

Go - компилируемый язык, исходный код преобразуется компилятором в машинный код и записывается в файл для последующей идентификации файла как исполняемого операционной системой.

Чем отличаются массивы и слайсы?

Длинна массива не может изменяться. Массив может копироваться при передаче в другую переменную.

Слайс позволяют изменять свою длину. Имеет параметр длины и вместимости. Если мы переполним длину слайса - вместимость умножится на 2. При копировании мы не создаем новый слайс.

Как работает функция append?

Используется для добавления элементов в слайс. Она принимает в качестве параметров слайс и значения, которые необходимо в него добавить. Во время выполнения программы функция append самостоятельно решает, нужно ли создавать новый слайс.

Что произойдет при попытке записи за пределы массива?

программа не скомпилируется

Какие бывают типы данных в Go?

Целочисленные типы

Числа с плавающей точкой

Комплексные числа

Логические (Тип bool)

Строки

Составные типы данных:

массивы

Срезы

структуры

карты (map)

Как передаются аргументы в функцию — по значению или по ссылке?

передаётся копия фактического значения, но можно использовать указатели для передачи фактического значения.

Что такое мапа и какие ключи могут быть у мапы?

Мапы - это встроенный тип данных в Go, который представляет собой ассоциативный массив или хеш-таблицу. Они позволяют хранить пары "ключ-значение", где каждому уникальному ключу соответствует определенное значение.

Ключи в мапе должны быть сравнимыми типами, такими как строки, числа или булевы значения. Значения же могут быть любого типа, включая другие мапы или структуры.

Потокобезопасна ли мапа в Go?

Мапы в Go не являются потокобезопасными.

Если попытаться одновременно читать и записывать данные в одну и ту же мапу, runtime вызовет панику.

Чтобы обеспечить потокобезопасность при работе с мапой из нескольких потоков, необходимо использовать механизмы синхронизации, такие как sync.Mutex или sync.RWMutex.

Как обрабатываются коллизии в мапе?

перезапись старого значения происходит только в том случае, если ключ действительно совпадает с существующим. Если ключ уникален, он просто добавляется в соответствующую корзину или цепочку.

Как взять адрес элемента мапы и почему?

Взять адрес элемента мапы (map) в Golang невозможно.

Это связано с тем, что Go динамически перераспределяет память для map, и когда в неё добавляются новые элементы, старые адреса становятся недействительными.

Чем отличается sync.Map от обычной мапы?

Обычная мапа в Go не является потокобезопасной, одновременная запись в мапу из нескольких горутин может привести к состоянию гонки. Sync.Map разработана для использования в многопоточных ситуациях и обеспечивает потокобезопасность.

Sync.Map — это сложная структура, обычно состоящая из двух мап — одной для чтения и одной для новых элементов.

Чем отличается горутина от системного потока?

Потоки управляются ядром ОС, горутины - рантаймом Go.

Горутины более легковесные, 2KB, потоки 1-2MB.

Размер стека потоков определяется во время компиляции и не может увеличиваться. Размер стэека горутин определяется во время рантайма и может расти вплоть до 1GB.

У потоков нет простого способа коммутации между собой. Такая коммутация имеет большую задержку. Горутины используют каналы для быстрого общения между собой с маленькой задержкой.

Какие механизмы синхронизации предоставляет Go?

Мьютексы (sync.Mutex) Разрешает только одной горутине удерживать блокировку для доступа к общему ресурсу в любой момент.

Мьютексы для чтения и записи (sync.RWMutex). Позволяют одновременно выполняться либо произвольному количеству операций чтения, либо одной операции записи.

Условные переменные (sync.Cond). Позволяют группе горутин ожидать выполнения определённого условия. Одна горутина блокирует сама себя, а другим нужно её «пробудить».

WaitGroup. Позволяет дождаться завершения выполнения всех горутин, запущенных в рамках программы.

Once. Позволяет выполнить определённую функцию только один раз.

Как избежать гонки данных в go?

Применить мьютексы (позволят гарантировать что одновременно к ресурсу может обращаться лишь одна горутина).

Использовать каналы для передачи данных между горутинами.

Пакет sync/atomic в Go предоставляет функции для безопасного доступа из нескольких горутин.

Что делает sync.WaitGroup ?

Позволяет дождаться выполнения всех горутин программы.

sync.WaitGroup содержит счётчик горутин, завершения которых нужно ждать.

WaitGroup состоит из 3 методов.

Add(delta int) — изменить значение счётчика на указанную величину.

Done() — уменьшить значение счётчика на единицу.

Wait() — ожидать, когда значение счётчика будет равно нулю.

Чем отличается Mutex от RWMutex ?

Mutex предоставляет взаимоисключающую блокировку, которая позволяет только одной горутине в любой момент времени иметь доступ к защищенным данным.

RWMutex позволяет множеству горутин получить параллельный доступ для чтения, но только одной горутине получить эксклюзивный доступ на запись.

Какие бывают типы каналов в Go? Чем отличаются буферизированные и небуферизированные каналы?

Двунаправленные каналы. Стандартный тип, поддерживает и отправку, и получение данных.

Однонаправленные каналы. Ограничивают операции только отправкой или только приёмом данных.

Буферизированные каналы. Имеют ограничение ёмкости, отправка данных в канал блокируется только при полном буфере.

Небуферизированные каналы не имеют внутренней ёмкости, то есть не могут хранить значения. Каждая операция отправки или получения на таком канале блокирует горутину, пока другая горутина не выполнит соответствующую операцию.

Что произойдет при записи в закрытый канал?

Попытка записи в закрытый канал в Go приведёт к панике во время выполнения программы.

Это одно из ключевых правил работы с каналами в Go: после закрытия канала нельзя больше отправлять в него данные, хотя читать из канала можно, пока в нём остаются данные.

Как проверить, что канал закрыт в go?

Чтобы проверить, закрыт ли канал в Go, можно использовать конструкцию val, ok := <- channel.

ok будет истиной, если канал открыт или операция чтения может быть выполнена. False — если канал закрыт и отсутствуют данные для чтения из него.

Для чего используется select при работе с каналами?

Select при работе с каналами в Go используется для того, чтобы ожидать данных из нескольких каналов одновременно. Он выбирает первый готовый канал и получает сообщение из него или передаёт сообщение через него.

Когда готовы несколько каналов, получение сообщения происходит из случайно выбранного готового канала. Если ни один из каналов не готов, оператор блокирует ход программы до тех пор, пока какой-либо из каналов не будет готов к отправке или получению.

Что такое интерфейс в Go и как он работает?

Это специальный тип, который определяет набор методов, позволяет описывать поведение типов.

Можно рассматривать как пару из значения и типа. Интерфейс является договоренностью, какой будет структура объекта в ООП.

Что такое пустой интерфейс и зачем он нужен?

Пустой интерфейс interface{} - это тип данных, который не содержит методов.

По сути может рассмотрен как любой тип данных. Нужен для работы с заранее неизвестным типом данных.

Также для функций, которые работают с любыми аргументами.

Чем отличается any от пустого интерфейса?

С точки зрения функциональности разницы между any и interface{} нет, но any считается более выразительным и предпочтительным вариантом начиная с Go 1.18

Что такое defer и зачем он используется?

Это ключевое слово, которое позволяет отложить выполнение функции до момента завершения текущей функции.

Например закрыть базу данных после того как пройдет извлечение необходимых данных.

Какой порядок выполнения defer в функции?

Порядок выполнения операторов defer в функции в Go — «последний вошёл — первый вышел» (LIFO).

Таким образом выполняются в обратном порядке своего появления в коде.

Для чего используется context ?

Используется для управленя временем выполнения операций и передачи данных между процессами.

Устанавливает дедлайны и таймауты. Например если операция выполняется слишком долго - ее нужно отменить.

Background. Базовый контекст, который обычно используется, когда другой контекст не доступен. Создаётся с помощью функции context.Background()

TODO. Используется, когда контекст пока неизвестен или в процессе разработки. Этот контекст может быть временным, его легко заменить на Background или другие типы. Создаётся с помощью функции context.TODO()

WithCancel. Создаёт контекст, который можно отменить вручную. Отмена происходит, когда вызывается cancel() для этого контекста, что приводит к завершению всех операций, использующих данный контекст. Создаётся с помощью функции context.WithCancel(parentContext)

WithDeadline и WithTimeout. Эти контексты предоставляют возможность установить время, после которого контекст будет автоматически отменён.

WithValue. Этот контекст предоставляет возможность связать значения с контекстом, которые затем могут быть извлечены в другом месте в коде. Создаётся с помощью функции context.WithValue(parentContext, key, value)

Как отменить выполнение операции через контекст?

По явному сигналу отмены. context.WithCancel

По наступлению временной отметки или дедлайна. context.WithDeadline.

По истечению промежутка времени. context.WithTimeout

Как работает сборщик мусора в Go?

Определяет корневые объекты (глобальные и локальные переменные функций и значения в регистрах).

Находит объекты на которые идут ссылки от корневых объектов.

Удаляет неиспользуемые объекты, на которые нет ссылок.

Сжимает память при необходимости, чтобы уменьшить фрагментацию (мелкие разрозненные в памяти).

Где хранятся переменные — в стеке или куче?

Локальные переменные - в стеке. Когда функция завершает выполнение, все данные на стеке автоматически освобождаются.

В куче находятся глобальные переменные, куча используетстся для больших объемов данных, которые могут существовать дольше чем вызов функций.

Как избежать утечек памяти в Go?

Закрывать каналы (могут стать утечой памяти). Завершать горутины (например через контекст).

Освобождать память от глобальных переменных при завершении функции defer cleanup(testMap).

Что такое escape analysis в Go? Что быстрее — работа со стеком или кучей?

"Анализ побега" - оптимизация компилятора, которая определяет, выдеяется ли объект в стеке или в куче.

Позволяет повысить производительность программы.

Компилятор Go проводит статический анализ кода, чтобы определить, какие объекты могут "убежать" за пределы стека. Если объект "убегает", то он размещается в куче.

Выделение объектов в стеке значительно быстрее, чем в куче, так как стековые операции выполняются почти мгновенно.

Чем конкурентность отличается от параллелизма?

Конкурентность - это многозадачность в рамках одного процессора, когда процессы используют его мощность по очереди с помощью планировщиков.

Параллелизм - одновременное выполнение задач на разных ядрах процессора.

Как работают горутины и каналы в Go?

Горутины - легковесные потоки выполнения. Управляются средой Go. Позволяют выполнять несколько функций одновременнож

Независят от операционной системы, т.к. реализованы на уровне языка.

Каналы - механизм коммуникации между горутинами. Обеспечивают безопасный способ передачи данных, синхронизируя их работу.

Бывают типизированными (принимают определенный тип), блокирующими (блокируют горутину до тех пор пока другая горутина не полочит/отправит данные).

Бывают буферезированные и небуферизированные. буферезированные позволяют отправлять данные без блокировки, пока не заполнится их буфер.

Какие проблемы могут возникнуть при работе в несколько потоков?

Могут возникнуть гонки данных (одновременная попытка изменить переменную горутинами например).

Deadlock. Когда 2 и более горутины ждут друг друга образуя замкнутый круг ожидания.