GO-06 05: Benchmarks

Описание:

В прошлых заданиях мы писали тесты, которые проверяли правильность нашего кода, но мы ни разу не проверяли оптимальность нашего решения. Бывают такие случаи, когда код, который мы написали, работает правильно и проходит все тесты, как unit, так и тесты, которые написали QA-специалисты, но когда мы выкатили код на продакшн и стали наблюдать за метриками, то начинаем замечать, что:

- постоянно растет потребление памяти;
- постоянно растет время выполнения, либо изначально время выполнения очень высокое.

Итог этого - необходимость оптимизации функционала.
Но как проверить, что после проведенной оптимизации все стало лучше? Снова раскатывать код на продакшн - вариант не самый удобный. Есть более удобный вариант проверки - это написание теста на производительность или Benchmark. К счастью, go поддерживает написание бенчмарков из коробки и нам не надо писать для этого решения на коленке или подключать сторонние библиотеки.
Как выглядит benchmark

```
func BenchmarkMyFunc(b *testing.B) {
    for i := 0; i < b.N; i++ {
        myFunc(10)
    }
}
```

Первое отличие - это префикс. Тогда как в обычных тестах у функции имеется префикс Test, у бенчмарков, соответственно, префиксом является Benchmark. Второе отличие - в аргументе. У пакета testing есть собственный тип testing.B, в котором есть все, что нужно для написания бенчмарков. Например:

- b.N в объекте testing.B отвечает за то, сколько операций будет запущено (этот параметр изменяется самостоятельно. Например, если мы хотим просто выполнить бенчмарк по стандарту за 1 секунду, тогда скорее всего это значение будет равно 1, но если мы захотим выполнять бенч в течение, например, 10 секунд, тогда это число может измениться).
- b.ReportAllocs() отвечает за вывод информации о выделении памяти в конкретном бенчмарке.
- b.SetBytes(1000) отвечает за то, сколько будет выделено байт для каждой операции бенча.

И так далее. С полным списком можно будет ознакомиться в документации (в разделе Полезные ссылки ниже).
Запуск benchmark

Для запуска бенчмарков можно просто выполнить команду go test с флагом -bench=regexp, где regexp - это аргумент, который нужен для того, чтобы определять, какие бенчмарки запускать.
Например, запуск всех бенчмарков в пакете:


```
~# go test -bench=.
goos: linux
goarch: amd64
pkg: module06
BenchmarkMyFunc-8                   500      2184963 ns/op
BenchmarkMyFuncV2-8                2892       357753 ns/op
PASS
ok    module06   2.417s
```

Запуск только одного бенчмарка BenchmarkMyFuncV2:

```
~# go test -bench=BenchmarkMyFuncV2
goos: linux
goarch: amd64
pkg: module06
BenchmarkMyFuncV2-8                5386       218244 ns/op
PASS
ok    module06   1.838s
```

На выходе мы видим количество операций, которые выполнялись в ходе бенчмарка, и время одной операции.
Еще мы можем посмотреть на информацию о памяти при помощи флага -benchmem:

```
~# go test -bench=. -benchmem
goos: linux
goarch: amd64
pkg: module06
BenchmarkMyFunc-8                   525      2311594 ns/op 14632275 B/op
35 allocs/op
BenchmarkMyFuncV2-8                3384       389710 ns/op  2400275 B/op
1 allocs/op
PASS
ok    module06   3.743s
```

Тут уже дополнительно показывается, сколько байт было выделено в каждую операцию и количество аллокаций памяти.
Также мы можем настраивать:

- продолжительность бенчмарка через флаг -benchtime t;
- количество итераций всего бенчмарка через флаг -count n.

Например, при запуске go test -bench=. -benchmem -benchtime 30s -count 2 бенчмарк нашего пакета будет запущен 2 раза по 30 секунд, а параметр b.N в функциях бенчмарка будет увеличен ровно настолько, чтобы бенч выполнялся 30 секунд.
Более подробную информацию о флагах можно будет найти по ссылке в разделе Полезные ссылки.
Сравнение бенчмарков
Давайте представим, что мы хорошо поработали над оптимизацией какой-либо важной функции в нашем проекте. Мы запускаем бенчмарк, дабы убедиться, что все работает достаточно быстро. Но тут встает вопрос. Как понять, что теперь все стало быстрее? Для этого нам нужно сравнить то, что было, и то, что стало, и для этого нам на помощь приходит утилита benchstat.
Для ее установки нам надо набрать следующую команду:

```
~# go get -u golang.org/x/perf/cmd/benchstat
```

Теперь нам нужно сохранить результаты бенчмарков старой функции и новой, для этого просто перенаправим вывод из консоли в файл следующим образом:

```
# Результаты старого запуска
go test -bench=MyFunc -benchmem -count=5 > old.txt

# Результаты нового запуска
go test -bench=MyFunc -benchmem -count=5 > new.txt
```

Теперь в файле old.txt мы можем увидеть следующее:

```
BenchmarkMyFunc-8          559      2008772 ns/op 14632389 B/op
37 allocs/op
BenchmarkMyFunc-8          595      2028446 ns/op 14632356 B/op
36 allocs/op
BenchmarkMyFunc-8          578      2066826 ns/op 14632349 B/op
36 allocs/op
BenchmarkMyFunc-8          556      2203445 ns/op 14632358 B/op
36 allocs/op
BenchmarkMyFunc-8          535      2100350 ns/op 14632347 B/op
36 allocs/op
```

А в файле new.txt вот это:

```
BenchmarkMyFunc-8         3938       297798 ns/op  2400282 B/op
1 allocs/op
BenchmarkMyFunc-8         3790       309053 ns/op  2400284 B/op
1 allocs/op
BenchmarkMyFunc-8         3888       309782 ns/op  2400282 B/op
1 allocs/op
```

```
BenchmarkMyFunc-8          3571        301259 ns/op  2400281 B/op
1 allocs/op
BenchmarkMyFunc-8          3925        311407 ns/op  2400282 B/op
1 allocs/op
```

Теперь сравним эти два файла при помощи утилиты benchstat:

```
~# benchstat old.txt new.txt

name       old time/op    new time/op    delta
MyFunc-8     2.08ms ± 6%    0.31ms ± 3%  -85.31%  (p=0.008 n=5+5)

name       old alloc/op   new alloc/op   delta
MyFunc-8     14.6MB ± 0%     2.4MB ± 0%  -83.60%  (p=0.008 n=5+5)

name       old allocs/op  new allocs/op  delta
MyFunc-8       36.0 ± 0%       1.0 ± 0%     ~     (p=0.079 n=4+5)
```

Тут мы видим разницу между результатами первого и второго прогона бенчмарка по времени выполнения, размеру используемой памяти и количеству аллокаций.

Полезные ссылки:

- Бенчмарки в go
- How to write benchmarks in Go
- Testing package (benchmark)
- Testing flags
- Пакет для анализа бенчмарков benchstat

Задание:

1. В этом задании нам потребуется написать бенчмарк двух функций, а затем попытаться улучшить результат, а именно:
    - Fib(n int) int - считает числа Фибоначчи;
    - MakeSlice(l int) []int - создает слайс заданного размера.
2. В вашем проекте module06 сделайте новую ветку module06_05.
3. Создайте файл util_bench_test.go в каталоге ./internal/pkg/util.
4. Напишите простой бенчмарк вычисления 35 числа Фибоначчи и бенчмарк для создания слайса с 3000000 элементами.
5. Запустите бенчмарки и проанализируйте результаты:
    - сколько времени затрачено на каждую операцию;

    - сколько операций вообще было;
    - какое потребление памяти.

Совет: Лучше запускать бенчмарк не на 1 секунду, а на более продолжительный срок. Например, секунд на 10.

6. Выполните изменения (улучшения) в функциях:
    - в функции Fib попробуйте отказаться от рекурсивного подхода в пользу итеративного;
    - в функции MakeSlice посмотрите в сторону slice capacity.
7. Снова прогоняем бенчмарк. Смотрим, как улучшился или ухудшился результат при помощи утилиты benchstat.
8. В качестве ответа пришлите ссылку на merge request в ветку master вашего проекта ветки module06_05, в которой должны быть:

- Скриншоты сравнения бенчмарков через benchstat.
- Файл util_bench_test.go в каталоге ./internal/pkg/util с бенчмарками.
- Улучшенные версии функций Fib и MakeSlice из пакета ./internal/pkg/util, если получилось их улучшить.