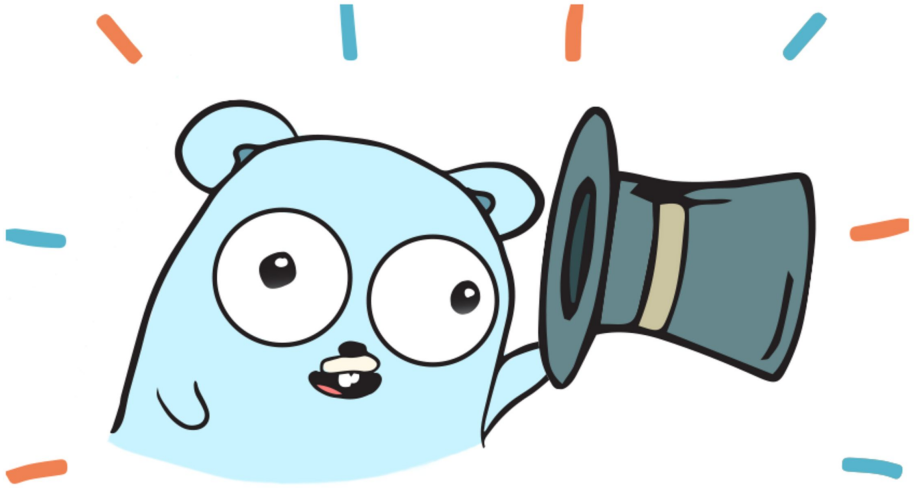


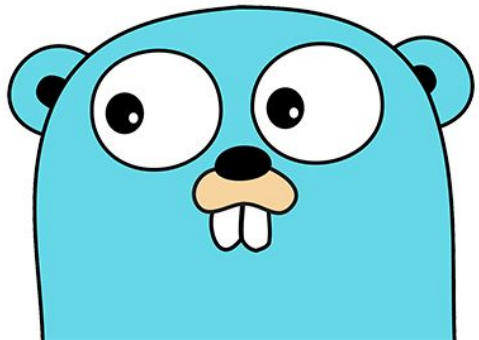
GOLANG SHOW

Новый заказ

Очередные грабли или прорыв? Опыт в go

GO

Профилирование и оптимизация



Что такое профилирование и для чего оно нужно

Профилирование — это сбор характеристик программы во время ее выполнения с целью их дальнейшей оптимизации.

Цели профилирования
– Оценка эффективности работы программы или вычислительной системы
– Определение критических участков программы (hotspots) или компонентов вычислительной системы

Что может встроенный профилировщик:

- CPU
- Heap
- Goroutines
- Block

Профилирование CPU

go tool pprof http://127.0.0.1:8080/debug/pprof/profile?seconds=5

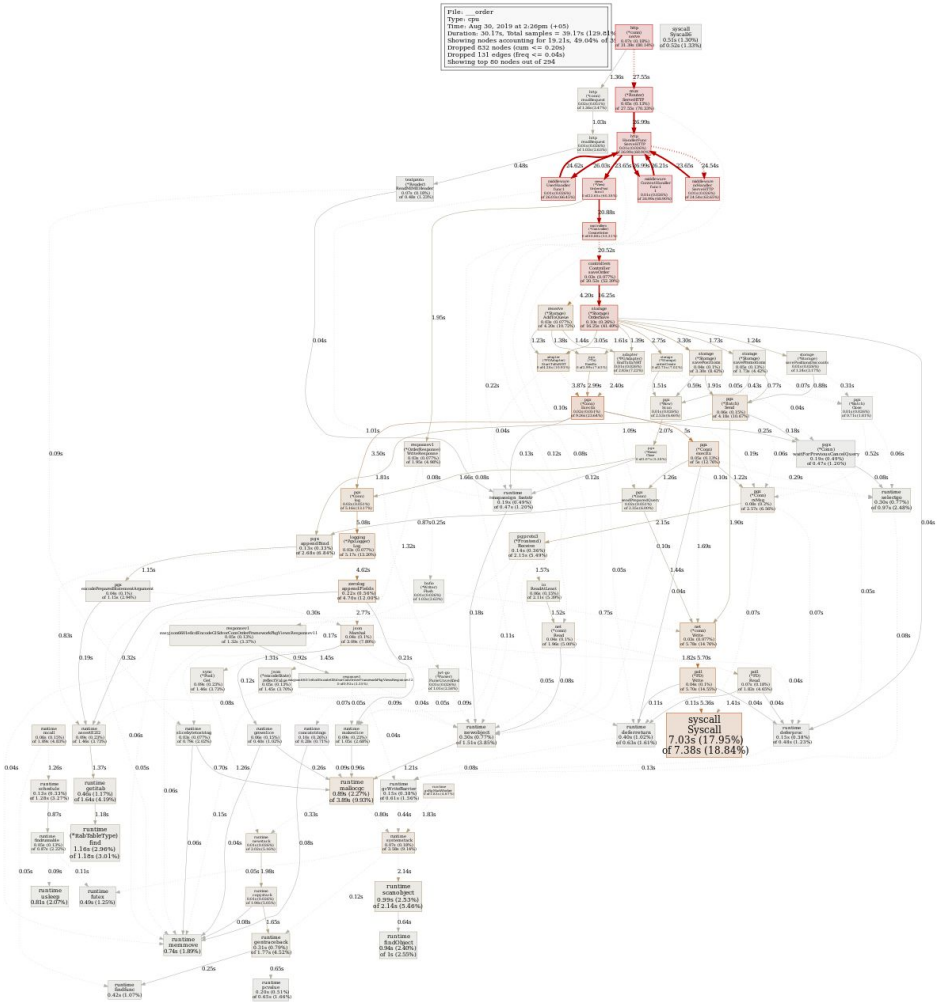
Профайлер CPU по умолчанию работает в течение 30 секунд. Он использует выборку, чтобы определить, какие функции тратят большую часть процессорного времени. Рантайм Go останавливает выполнение каждые десять миллисекунд и записывает текущий стек вызовов всех работающих горутин.

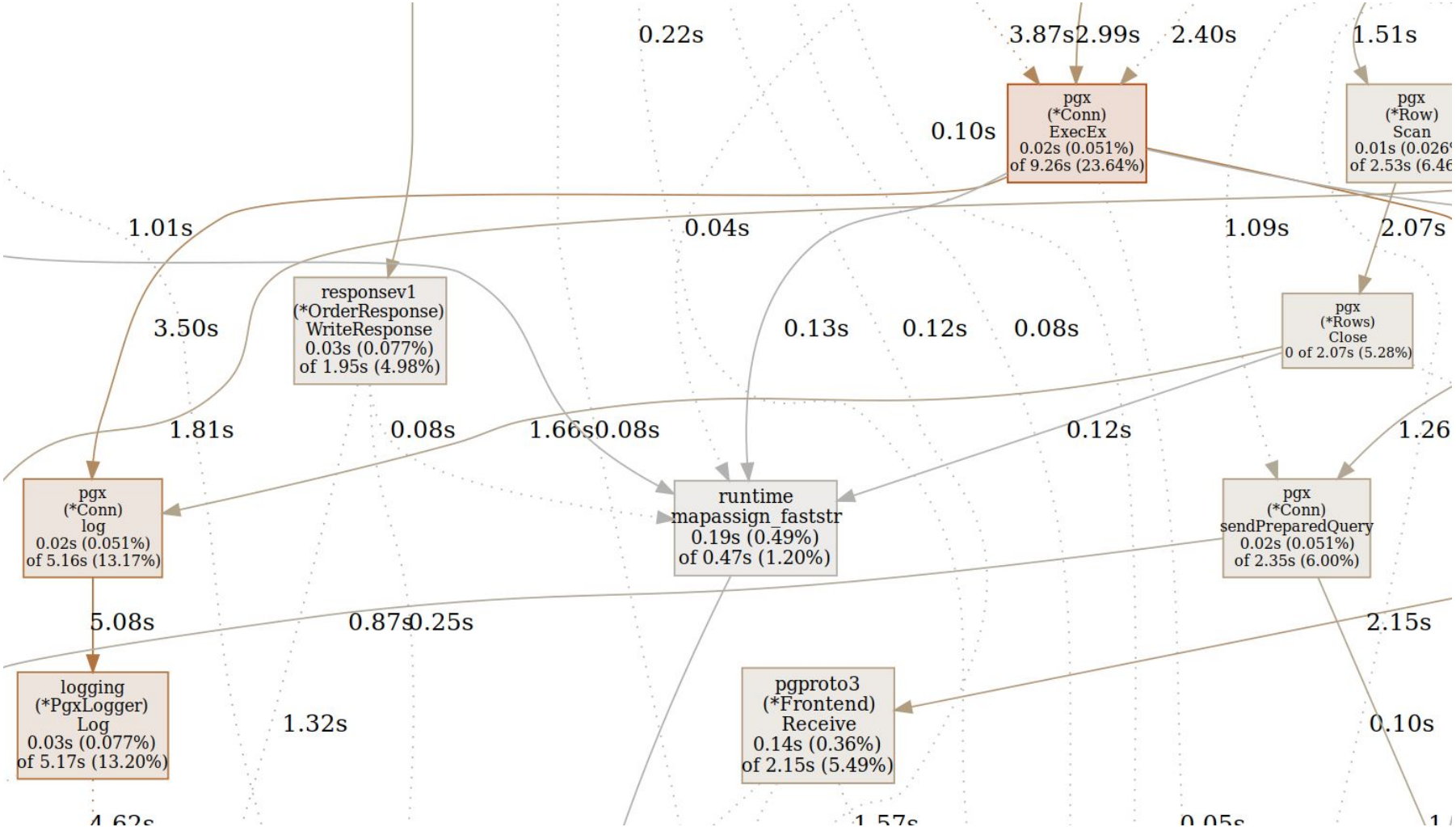
команда top

```
(pprof) top10
Showing nodes accounting for 12.53s, 53.80% of 23.29s total
Dropped 629 nodes (cum <= 0.12s)
Showing top 10 nodes out of 252
      flat  flat%   sum%        cum   cum%
     3.70s 15.89% 15.89%      3.98s 17.09%  syscall.Syscall
     1.75s  7.51% 23.40%      1.75s  7.51%  runtime.futex
     1.66s  7.13% 30.53%      1.66s  7.13%  runtime.usleep
     1.15s  4.94% 35.47%      1.15s  4.94%  runtime.epollwait
     0.49s  2.10% 37.57%      0.50s  2.15%  runtime.(*itabTableType).find
     0.38s  1.63% 39.20%      1.59s  6.83%  runtime.mallocgc
     0.31s  1.33% 40.53%      0.31s  1.33%  runtime.memmove
     0.31s  1.33% 41.86%      0.31s  1.33%  runtime.nextFreeFast
     0.30s  1.29% 43.15%      0.33s  1.42%  runtime.step
     0.29s  1.25% 44.40%      0.58s  2.49%  runtime.scanobject
(pprof)
```

команда web

строит граф вызовов в формате svg




0.22s
3.87s
2.99s
2.40s
1.51s
pgx
(*Conn)
ExecEx
0.02s (0.051%)
of 9.26s (23.64%)
pgx
(*Row)
Scan
0.01s (0.026%)
of 2.53s (6.46%)
0.10s
1.01s
0.04s
1.09s
2.07s
responsev1
(*OrderResponse)
WriteResponse
0.03s (0.077%)
of 1.95s (4.98%)
3.50s
0.13s
0.12s
0.08s
pgx
(*Rows)
Close
0 of 2.07s (5.28%)
1.81s
0.08s
1.66s
0.08s
0.12s
1.26
pgx
(*Conn)
log
0.02s (0.051%)
of 5.16s (13.17%)
runtime
mapassign_faststr
0.19s (0.49%)
of 0.47s (1.20%)
pgx
(*Conn)
sendPreparedQuery
0.02s (0.051%)
of 2.35s (6.00%)
5.08s
0.87s
0.25s
2.15s
logging
(*PgxLogger)
Log
0.03s (0.077%)
of 5.17s (13.20%)
1.32s
pgproto3
(*Frontend)
Receive
0.14s (0.36%)
of 2.15s (5.49%)
0.10s
4.62s
1.57s
0.05s

команда list

```
(pprof) list .OrdersPost.func1
Total: 23.29s
ROUTINE ======================== gl.sdvor.com/order/framework/pkg/views.(*View).OrdersPost.func1 in
/home/mishunix/projects/gl.sdvor.com/order/framework/pkg/views/orders.go
      10ms     11.59s (flat, cum) 49.76% of Total
         .          .     54:}
         .          .     55:
         .          .     56:// POST /api/v1/orders
         .          .     57:func (v *View) OrdersPost() http.HandlerFunc {
         .          .     58:return func(w http.ResponseWriter, r *http.Request) {
         .       10ms     59:        var requestBody RequestOrderToAdd
         .          .     60:        ctx := GetContext(r).SetOperation(appctx.OpOrderCreate).
         .      310ms     61:                Decode(r.Body, &requestBody).
         .          .     62:                CheckAPIVersion(requestBody.APIVersion).
         .          .     63:                Validate(&requestBody.Data)
         .          .     64:        if ctx.Error != nil {
         .          .     65:                google.WriteError(w, ctx.APIVersion, ctx.Error)
         .          .     66:                return
         .          .     67:        }
      10ms     10.52s     68:        if order, err := v.controller.CreateOrder(ctx, &requestBody.Data); err == nil {
         .      750ms     69:                responsev1.New(order).WriteResponse(ctx, w)
         .          .     70:        } else {
         .          .     71:                google.WriteError(w, ctx.APIVersion, err)
         .          .     72:        }
         .          .     73:}
         .          .     74:}
(pprof)
```

Профилирование памяти (heap)

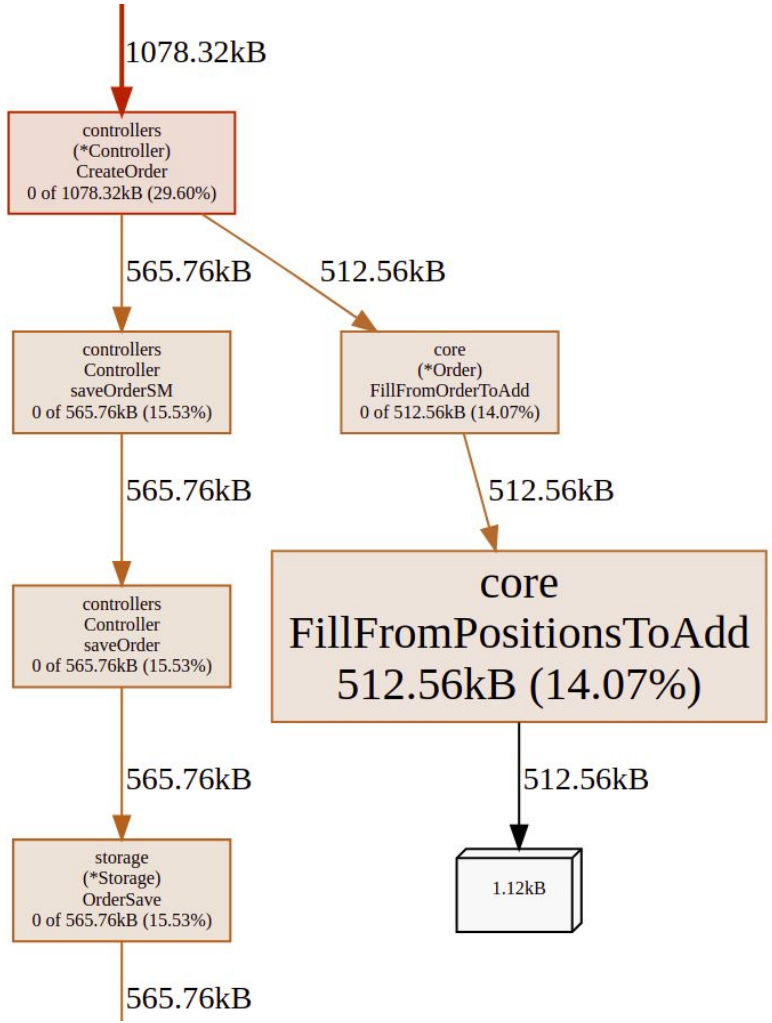
go tool pprof http://127.0.0.1:8080/debug/pprof/heap

Автоматическое управление памятью – вещь удобная, но в мире, увы, нет ничего бесплатного. Выделение памяти на куче не только значительно медленнее, чем выделение на стеке, но ещё и косвенно влияет на производительность. Каждый фрагмент памяти, который вы выделяете в куче, добавляет работы сборщику мусора и заставляет использовать больше ресурсов процессора. Единственный способ заставить приложение тратить меньше времени на сборку мусора – сократить количество аллокаций.

команда top

```
(pprof) top10
Showing nodes accounting for 3643.46kB, 100% of 3643.46kB total
Showing top 10 nodes out of 43
      flat  flat%   sum%        cum   cum%
    1027kB 28.19% 28.19%     1027kB 28.19%  bufio.NewWriterSize
  565.76kB 15.53% 43.72%   565.76kB 15.53%  bytes.makeSlice
     514kB 14.11% 57.82%      514kB 14.11%  bufio.NewReaderSize
  512.56kB 14.07% 71.89%   512.56kB 14.07%  .../core.FillFromPositionsToAdd
  512.12kB 14.06% 85.95%   512.12kB 14.06%  net/http.readRequest
  512.01kB 14.05%   100%  2563.70kB 70.36%  net/http.(*conn).serve
         0     0%   100%      514kB 14.11%  bufio.NewReader
         0     0%   100%      514kB 14.11%  bufio.NewWriter
         0     0%   100%   565.76kB 15.53%  bytes.(*Buffer).Grow
         0     0%   100%   565.76kB 15.53%  bytes.(*Buffer).grow
(pprof)
```

команда web


1078.32kB
controllers
(*Controller)
CreateOrder
0 of 1078.32kB (29.60%)
565.76kB
512.56kB
controllers
Controller
saveOrderSM
0 of 565.76kB (15.53%)
core
(*Order)
FillFromOrderToAdd
0 of 512.56kB (14.07%)
565.76kB
512.56kB
controllers
Controller
saveOrder
0 of 565.76kB (15.53%)
core
FillFromPositionsToAdd
512.56kB (14.07%)
565.76kB
512.56kB
storage
(*Storage)
OrderSave
0 of 565.76kB (15.53%)
1.12kB
565.76kB

дополнительные опции

-inuse_space — показывает количество памяти, занятое в текущий момент (для работающих приложений), применяется по умолчанию;

-inuse_objects — то же самое, но показывает количество объектов;

-alloc_space — показывает количество памяти, выделенное за все время работы программы;

-alloc_objects — аналогично, только для объектов;

-alloc_objects

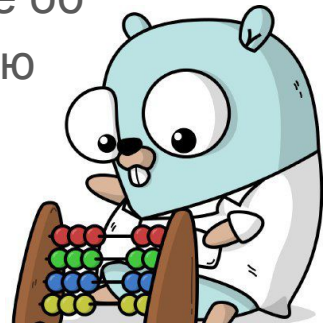
```
(pprof) top
Showing nodes accounting for 4362038, 36.53% of 11940560 total
Dropped 194 nodes (cum <= 59702)
Showing top 10 nodes out of 221
      flat  flat%   sum%        cum   cum%
    917529  7.68%  7.68%     917529  7.68%  .../core/money.Money.MarshalJSON
    655367  5.49% 13.17%     655367  5.49%  github.com/jackc/pgx/pgtype.underlyingNumberType
    612195  5.13% 18.30%     612195  5.13%  gl.sdvor.com/order/framework/pkg/storage.glob..func5
    525450  4.40% 22.70%     558218  4.67%  net/textproto.(*Reader).ReadMIMEHeader
    360452  3.02% 25.72%     360452  3.02%  github.com/json-iterator/go.(*Iterator).ReadString
    327683  2.74% 28.46%     327683  2.74%  .../pgproto3.(*CommandComplete).Decode
    268117  2.25% 30.71%     268117  2.25%  gl.sdvor.com/order/framework/pkg/storage.glob..func3
    262147  2.20% 32.90%     262147  2.20%  github.com/jackc/pgx/pgtype.underlyingStringType
    219881  1.84% 34.75%     383723  3.21%  github.com/jackc/pgx/pgproto3.(*RowDescription).Decode
    213217  1.79% 36.53%     213217  1.79%  github.com/jackc/pgx.(*Conn).rxRowDescription
(pprof)
```

-alloc_space

```
(pprof) top
Showing nodes accounting for 174.09MB, 34.08% of 510.84MB total
Dropped 158 nodes (cum <= 2.55MB)
Showing top 10 nodes out of 198
      flat  flat%   sum%        cum   cum%
   27.51MB  5.39%  5.39%    27.51MB  5.39%  github.com/rs/zerolog.Logger.With
   22.02MB  4.31%  9.70%    22.02MB  4.31%  .../core.FillFromPositionsToAdd
   22.01MB  4.31% 14.01%    22.51MB  4.41%  net/textproto.(*Reader).ReadMIMEHeader
   18.52MB  3.63% 17.63%    19.52MB  3.82%  .../views/responsev1.(*Order).convertPositions
   17.20MB  3.37% 21.00%    17.20MB  3.37%  github.com/jackc/pgx.(*Conn).getRows
   16.50MB  3.23% 24.23%    16.50MB  3.23%  net/http.(*Request).WithContext
   13.50MB  2.64% 26.87%    13.50MB  2.64%  encoding/base64.(*Encoding).DecodeString
      13MB  2.55% 29.42%       13MB  2.55%  gl.sdvor.com/order/framework/pkg/storage.glob..func5
   12.01MB  2.35% 31.77%    40.02MB  7.83%  .../core/appctx.NewContext
   11.81MB  2.31% 34.08%    11.81MB  2.31%  regexp.(*bitState).reset
(pprof)
```

Оптимизация

- Избегайте ненужных выделений памяти в куче.
- Для небольших структур используйте передачу параметров по значению, а не по ссылке.
- Заранее выделяйте память под maps и slices, если вам известен размер.
- Не логируйте без необходимости.
- Используйте буферизованный ввод-вывод, если выполняете много последовательных операций чтения или записи.
- Если ваше приложение широко использует JSON, то подумайте об использовании парсеров/ сериализаторов (лично я предпочитаю easyjson).



```
siege -c250 --time=1min --content-type "application/json" --header="cache-control: no-cache" \
--header="svyabk: RU-RUB-ru-ENRG-ennergiia.com" \
--header="Authorization: Bearer ..." \
--header="user-role: MIM" \
'http://127.0.0.1:8000/api/v1/orders POST {
  "apiVersion": "1.0",
  "data":{
    "priceZone": 31,
    "availZone": 5,
    "type": "offer",
    "positions": [{
      "productItemId": "165234",
      "basePrice": 2500,
      "measureUnit": "ST",
      "quantity": 11
    },{
      "productItemId": "312323",
      "basePrice": 1250,
      "price": 750,
      "priceType": "red",
      "measureUnit": "ST",
      "quantity": 1050
    }]
  }
}'
```