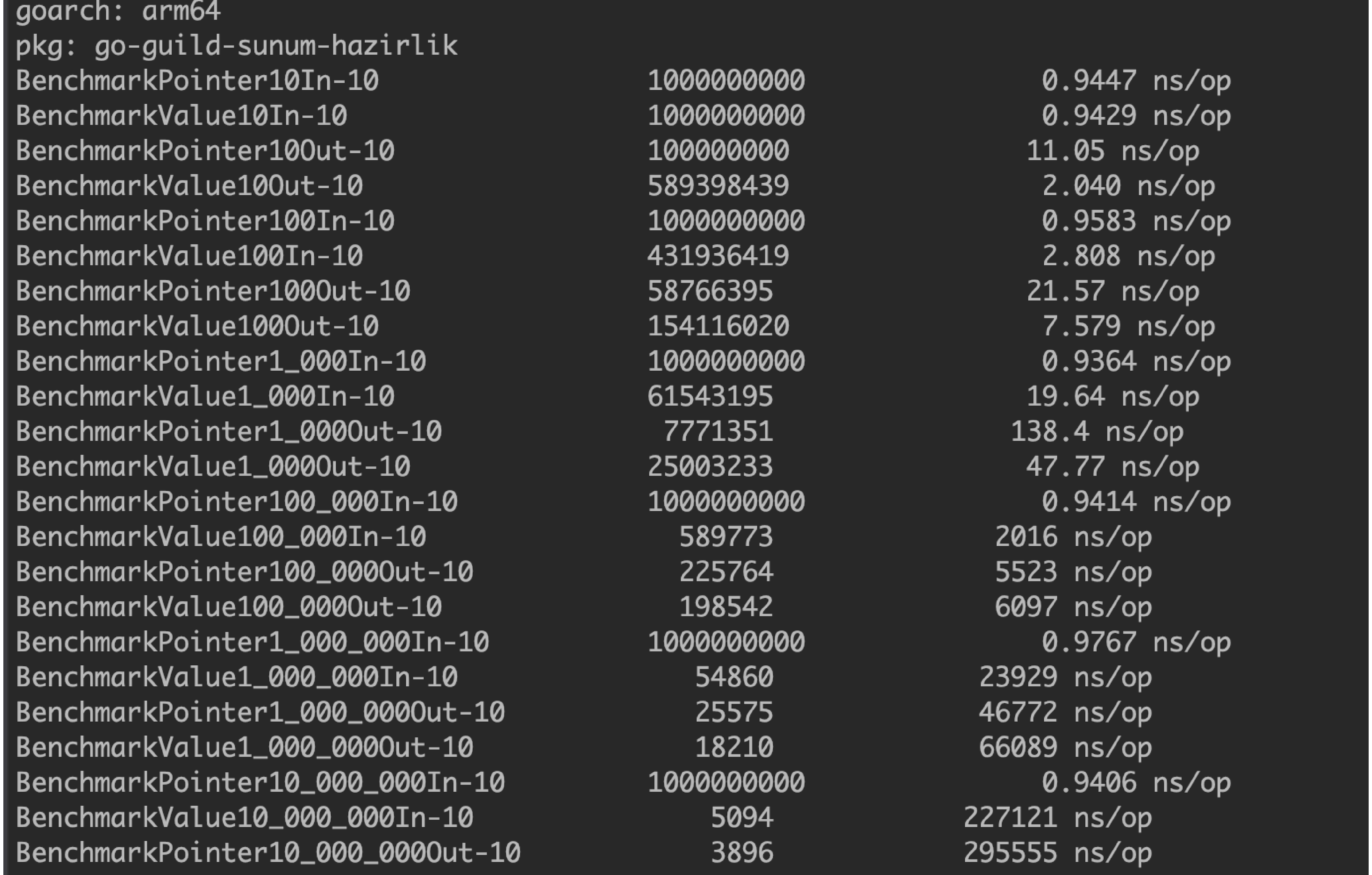
Good to know these Go Tricks

Abdulsamet İleri

Table of contents

- Pointer Passing Performance
- Pointer Receiver Nil Check
- Custom Marshal/Unmarshal Functions
- Read HttpBody even if don't need to read
- Run your code exactly once (Singleton)
- The beautify of `sync.Pool`



```
goarch: arm64
pkg: go-guild-sunum-hazirlik
BenchmarkPointer10In-10                 1000000000               0.9447 ns/op
BenchmarkValue10In-10                   1000000000               0.9429 ns/op
BenchmarkPointer10Out-10                100000000               11.05 ns/op
BenchmarkValue10Out-10                  589398439                2.040 ns/op
BenchmarkPointer100In-10                1000000000               0.9583 ns/op
BenchmarkValue100In-10                  431936419                2.808 ns/op
BenchmarkPointer100Out-10               58766395                21.57 ns/op
BenchmarkValue100Out-10                 154116020                7.579 ns/op
BenchmarkPointer1_000In-10              1000000000               0.9364 ns/op
BenchmarkValue1_000In-10                61543195                19.64 ns/op
BenchmarkPointer1_000Out-10              7771351               138.4 ns/op
BenchmarkValue1_000Out-10               25003233                47.77 ns/op
BenchmarkPointer100_000In-10            1000000000               0.9414 ns/op
BenchmarkValue100_000In-10                589773              2016 ns/op
BenchmarkPointer100_000Out-10             225764              5523 ns/op
BenchmarkValue100_000Out-10               198542              6097 ns/op
BenchmarkPointer1_000_000In-10          1000000000               0.9767 ns/op
BenchmarkValue1_000_000In-10               54860             23929 ns/op
BenchmarkPointer1_000_000Out-10            25575             46772 ns/op
BenchmarkValue1_000_000Out-10              18210             66089 ns/op
BenchmarkPointer10_000_000In-10         1000000000               0.9406 ns/op
BenchmarkValue10_000_000In-10               5094            227121 ns/op
BenchmarkPointer10_000_000Out-10            3896            295555 ns/op
```

```go
func (it *IntTree) Insert(val int) *IntTree {
    if it == nil {
        return &IntTree{val: val}
    }
    if val < it.val {
        it.left = it.left.Insert(val)
    } else if val > it.val {
        it.right = it.right.Insert(val)
    }
    return it
}

func main() {
    var it *IntTree
    it = it.Insert(5)
    it = it.Insert(3)
    it = it.Insert(10)
}
```

- In value receiver, it is not possible to check nil and panic comes

```go
type Person struct {
        Name    string
        Surname string
}

func (p Person) FullName() string {
        return p.Name + " " + p.Surname
```

```go
type Marshaler interface {
    MarshalJSON() ([]byte, error)
}
```

- When Go is decoding some JSON, it will check to see if the destination type satisfies the json.Unmarshaler interface. If it does satisfy the interface, then Go will call it’s UnmarshalJSON() method to determine how to decode the provided JSON into the target type.

```go
type Unmarshaler interface {
    UnmarshalJSON([]byte) error
}
```

```go
func (payload *MessagePayload) UnmarshalJSON(data []byte) error {
        type innerPayload MessagePayload
        inner := &innerPayload{}

        if err := json.ConfigWithDisallowUnknownFields.Unmarshal(data, inner); err != nil {
                return err
        }

        if inner.ContentID == 0 {
                return errors.New("invalid message content id must not be zero")
        }

        if inner.Culture == "" {
                return errors.New("ignore empty culture")
        }

        uppercaseReason := strings.ToUpper(string(inner.Reason))
        inner.Reason = Reason(uppercaseReason)
```

Read HttpBody even if don't need to read

- If we close the body without a read, the default HTTP transport may close the connection.
- If we close the body following a read, the default HTTP transport won’t close the connection; hence, it may be reused.

```
func (h handler) getStatusCode(body io.Reader) (int, error) {
    resp, err := h.client.Post(h.url, "application/json", body)
    if err != nil {
        return 0, err
    }
    defer resp.Body.Close()

    _, _ = io.Copy(io.Discard, resp.Body)
    return resp.StatusCode, nil
}
```

Run your code exactly once

- If GetLocationByZoneID is called more than once, once.Do will not execute the closure again.

```go
var (
	locationByZoneIDInstance *LocationByZoneID
	once                     sync.Once
)

func GetLocationByZoneID() *LocationByZoneID {
	once.Do(func() {
		locationByZoneIDInstance = &LocationByZoneID{
			mu:    &sync.RWMutex{},
			store: make(map[string]*time.Location),
		}
	})
	return locationByZoneIDInstance
}

func (m *LocationByZoneID) GetLocation(zoneID string) (*time.Location, bool) {
	m.mu.RLock()
	defer m.mu.RUnlock()

	val, ok := m.store[zoneID]
	return val, ok
}

func (m *LocationByZoneID) SetLocation(zoneID string) *time.Location {
	tzInZone, _ := time.LoadLocation(zoneID)

	m.mu.Lock()
	m.store[zoneID] = tzInZone
	m.mu.Unlock()

	return m.store[zoneID]
}
```

The beautify of sync.Pool

```go
var bufferPool = sync.Pool{
	New: func() any { return newBuffer() },
}

func newBuffer() *bytes.Buffer {
	b := new(bytes.Buffer)
	b.Grow(65536)
	return b
}

func acquireBuffer() *bytes.Buffer {
	return bufferPool.Get().(*bytes.Buffer)
}

func releaseBuffer(b *bytes.Buffer) {
	if b != nil {
		b.Reset()
		bufferPool.Put(b)
	}
}
```